vSphere IaaS 제어부에서 TKG 서비스 사용

업데이트 3

VMware vSphere 8.0
VMware vCenter 8.0
VMware ESXi 8.0

vmware
by Broadcom

VMware by Broadcom 웹 사이트

https://docs.vmware.com/kr에서 최신 기술 문서를 찾을 수 있습니다.

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

목차

vSphere IaaS 제어부에서 TKG 서비스 사용

이 설명서에서는 vSphere IaaS control plane의 감독자에서 Tanzu Kubernetes Grid 클러스터의 수명 주기를 관리하기 위한 정보를 제공합니다.

감독자에서 Tanzu Kubernetes Grid를 사용하면 기본값이 잘 정의된 Tanzu Kubernetes 클러스터 및 광범위한 정의 옵션이 있는 Cluster Class 클러스터를 비롯한 다양한 스타일의 워크로드 클러스터를 프로비저닝할 수 있습니다. 여러 vSphere 영역에 감독자를 배포하여 무장애 워크로드를 지원하는 고가용성 TKG 클러스터를 프로비저닝할 수 있습니다. 업데이트된 Tanzu Kubernetes 릴리스 형식은 기본적으로 Tanzu 패키지 저장소를 통합하고 여러 운영 체제를 지원합니다. vCenter Single Sign-On 및 vSphere에 대한 Kubernetes CLI 도구를 사용하거나 외부 ID 제공자 및 Tanzu CLI를 사용하여 감독자에서 TKG 클러스터에 액세스하고 관리할 수 있습니다.

대상 사용자

이 설명서의 내용은 vSphere IaaS control plane의 감독자에서 TKG 클러스터의 수명 주기를 관리하고 프로비저닝하려는 vSphere 관리자 및 Kubernetes 운영자를 대상으로 합니다.

적용 가능한 vSphere 및 TKG 버전

달리 명시되지 않은 한, 이 설명서는 vSphere 8 버전의 vSphere IaaS control plane에 적용되며 감독자에서 TKG v3.x를 지원하도록 업데이트되었습니다. 설명서 업데이트는 누적됩니다. 이전 릴리스의 보관된 설명서는 VMware 설명서 사이트의 **아카이브 패키지** 제목 아래에서 다운로드할 수 있습니다.

업데이트된 정보

1

이 설명서는 필요에 따라 새로운 정보와 수정 사항으로 정기적으로 업데이트됩니다.

이 설명서의 업데이트 기록이 표에 정리되어 있습니다.

개정	설명

2024년 6월 25일	다음을 포함하여 vSphere 8 업데이트 3 및 TKG 서비스 3.0을 지원하도록 설명서가 업데이트되었습니다.
	■ 장 3 TKG 서비스 설치 및 업그레이드
	■ 장 10 TKG 서비스 클러스터 자동 스케일링
	■ 장 20 TKG 서비스 클러스터 및 워크로드 백업 및 복원
	■ TKG 서비스 클러스터에 대해 Antrea-NSX 어댑터 사용
	■ NSX 관리 프록시 서비스 설치
	■ 장 11 TKG 서비스 클러스터에 표준 패키지 설치
	■ v1beta1 클러스터에 대한 MachineHealthCheck 구성
	■ Cluster v1beta1 API (업데이트)
	  ■ controlPlaneCertificateRotation(업데이트됨)
	  ■ podSecurityStandard(신규)
	■ v1beta1 예: 사용자 지정 ClusterClass 기반 클러스터(vSphere 8 U2 이상 워크플로)(업데이트)
	■ TKG 클러스터에 vSphere용 Velero 플러그인 설치 및 구성(업데이트)
	■ Kubectl을 사용하여 수동으로 클러스터 확장/축소(업데이트)

2024년 3월 8일	■ 최소 버전이 vSphere 8.0 업데이트 2+(TKr v1.26 이상 포함)임을 나타내도록 CSI 스냅샷 요구 사항이 업데이트되었습니다. 장 15 TKG 서비스 클러스터에서 스냅샷 생성의 내용을 참조하십시오. ■ CSI 스냅샷이 있는 Velero를 사용하여 TKGS 클러스터 워크로드를 백업하고 복원하기 위한 지침이 추가되었습니다. CSI 스냅샷과 함께 Velero를 사용하여 백업 및 복원의 내용을 참조하십시오. ■ FQDN을 사용하여 v1beta1 API 클러스터를 프로비저닝하기 위한 예제 및 확인 단계가 업데이트되었습니다. v1beta1 예: FQDN이 있는 클러스터의 내용을 참조하십시오. ■ vSphere 네임스페이스에 대한 '볼 수 있음' 역할 사용 권한 관련 설명 및 Kubernetes 제한 사항이 업데이트되었습니다. 역할 사용 권한 및 바인딩의 내용을 참조하십시오. ■ Linux 클라이언트 호스트에 vCenter가 신뢰하는 루트 CA 인증서를 설치하기 위한 명시적 지침이 추가되었습니다. vCenter에 대한 신뢰할 수 있는 루트 CA 인증서를 다운로드하여 Ubuntu 클라이언트에 설치의 내용을 참조하십시오.

개정	설명
2024년 2월 29일	■ vSphere 8 U2 P03의 GA 릴리스를 지원하기 위해 설명서가 업데이트되었습니다. ■ FQDN을 사용한 클러스터 생성을 지원하는 설명서가 추가되었습니다. Cluster v1beta1 API 및 v1beta1 예: FQDN이 있는 클러스터의 내용을 참조하십시오. ■ 사용하도록 설정된 경우 FQDN을 사용할 수 있음을 나타내기 위해 감독자 로그온 명령이 업데이트되었습니다. ■ 추가 예제로 PSA 설명서가 업데이트되었습니다. TKR 1.25 이상에 대한 PSA 구성의 내용을 참조하십시오. ■ vGPU 워크로드 실행을 위해 TKGS 워크로드 클러스터에 NVIDIA Network Operator를 설치하기 위한 지침이 추가되었습니다. TKG 서비스 클러스터에 AI/ML 워크로드를 배포하기 위한 클러스터 운영자 워크플로의 내용을 참조하십시오. ■ VM 클래스 생성 마법사를 사용하기 위한 컨텐츠, 특히 NVIDIA Grip vGPU용 사용자 지정 VM 클래스를 생성하기 위한 컨텐츠가 추가되었습니다. 장 13 TKG 서비스 클러스터에 AI/ML 워크로드 배포의 내용을 참조하십시오. ■ Velero의 버전 호환성 매트릭스에 대한 링크로 #unique_22 및 TKG 클러스터에 vSphere용 Velero 플러그인 설치 및 구성항목이 업데이트되었습니다.
2024년 2월 2일	■ vSphere 8에서 v1alpha2 API가 v1alpha3 API로 자동 변환됨을 나타내기 위해 TKG 클러스터 프로비저닝 설명서가 업데이트되었습니다. TKG 클러스터 프로비저닝 정보의 내용을 참조하십시오.
2024년 1월 31일	■ 기타 버그 수정 및 편집.
2024년 1월 19일	■ TKGS 클러스터를 개인 컨테이너 레지스트리와 통합하기 위한 항목이 업데이트되었습니다. TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합의 내용을 참조하십시오.
2024년 1월 18일	■ `defaultRegistrySecret` 변수가 내장된 Harbor 레지스트리와 함께 사용하도록 예약되어 있음을 명확히 하기 위해 Cluster v1beta1 API 항목이 업데이트되었습니다.
2024년 1월 16일	■ TKG 표준 패키지 저장소 버전 v2023.9.19 이상을 사용해야 함을 나타내기 위해 장 15 TKG 서비스 클러스터에서 스냅샷 생성항목이 업데이트되었습니다.
2024년 1월 8일	■ 기타 부분적 업데이트.
2024년 1월 2일	■ 기타 부분적 업데이트.
2023년 12월 20일	■ 사용자 지정 TKr 이미지 구축을 위한 설명서 링크로 TKG 서비스 클러스터에서 Kubernetes 릴리스 사용 항목이 업데이트되었습니다. ■ 기타 부분적 업데이트.
2023년 12월 6일	■ 새로운 Tanzu Kubernetes 릴리스로 TKR 릴리스 정보가 업데이트되었습니다. ■ 시스템 네임스페이스의 PSA를 변경할 수 없음을 명시하기 위해 TKR 1.25 이상에 대한 PSA 구성 항목이 업데이트되었습니다. ■ 여러 노드 풀 사용 시의 고려 사항으로 TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해 항목이 업데이트되었습니다.
2023년 11월 21일	■ /21이 라우팅 가능한 포드 네트워크의 최소 크기임을 문서화하기 위해 v1alpha3 예시: 라우팅 가능한 포드 네트워크가 있는 TKC 및 v1beta1 예시: 라우팅 가능한 포드 네트워크가 있는 클러스터 항목이 업데이트되었습니다. ■ vSphere 7 감독자 업데이트에 대해 롤링 업데이트가 트리거될 수 있는 시기를 명확히 하기 위해 TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해 항목이 업데이트되었습니다. ■ TKr에 PSP가 필요한 시기를 지정하기 위해 컨테이너 배포 오류 문제 해결항목이 업데이트되었습니다.

개정	설명
2023년 11월 13일	■ 유효한 구성 키가 영숫자, '-', '_' 또는 '.'로 구성되어야 함을 명시하기 위해 설명서가 업데이트되었습니다. (예: 'key.name', 'KEY_NAME' 또는 'key-name').
2023년 11월 7일	■ TKr 버전 문자열 업그레이드를 위한 항목이 업데이트되었습니다. TKR 버전을 편집하여 TKG 클러스터 업데이트의 내용을 참조하십시오.
2023년 10월 16일	■ 링크가 수정되었습니다.
2023년 10월 13일	■ v1beta1 API의 사용과 관련된 예제를 포함하도록 TKG 클러스터 확장 지침이 업데이트되었습니다. Kubectl을 사용하여 수동으로 클러스터 확장/축소의 내용을 참조하십시오.
2023년 10월 11일	■ Tanzu CLI를 사용하여 OIDC사용자로 TKG 클러스터에 연결하기 위한 지침이 업데이트되었습니다. Tanzu CLI를 사용하여 TKG 클러스터에 OIDC 사용자로 연결의 내용을 참조하십시오.
2023년 10월 9일	■ 컨텐츠 라이브러리를 vSphere 네임스페이스와 연결하기 위한 지침이 업데이트되었습니다. TKR 컨텐츠 라이브러리를 TKG 서비스와 연결의 내용을 참조하십시오.
2023년 10월 3일	■ 관리되지 않는 사용자 지정 ClusterClass 생성을 위한 주석을 포함하도록 vSphere 8 U2 사용자 지정 ClusterClass 설명서가 업데이트되었습니다. v1beta1 예: 사용자 지정 ClusterClass 기반 클러스터(vSphere 8 U2 이상 워크플로)의 내용을 참조하십시오.
2023년 9월 21일	다음을 포함한 vSphere 8 U2 릴리스 기능에 대한 설명서가 업데이트되었습니다. ■ TKG 클러스터에 대한 볼륨 스냅샷 생성. 장 15 TKG 서비스 클러스터에서 스냅샷 생성의 내용을 참조하십시오. ■ TKG 클러스터에 패키지 설치. 장 11 TKG 서비스 클러스터에 표준 패키지 설치의 내용을 참조하십시오. ■ v1beta1 API를 사용하여 사용자 지정 ClusterClass 클러스터 프로비저닝. 클러스터 v1beta1 API 사용의 내용을 참조하십시오. ■ TKG 클러스터에 대한 롤링 업데이트 고려 사항. TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해의 내용을 참조하십시오. ■ 기타 유지 보수 업데이트 및 편집. **참고** vSphere IaaS control plane 및 Tanzu Kubernetes 릴리스에 대한 릴리스 정보를 확인하십시오.
2023년 8월 16일	■ 라우팅 가능한 포드가 있는 v1alpha3 TKC 및 사용자 지정 ClusterClass가 있는 v1beta1 클러스터에 대한 TKG 클러스터 프로비저닝 예시가 업데이트되었습니다. 장 7 TKG 서비스 클러스터 프로비저닝의 내용을 참조하십시오.
2023년 8월 15일	■ Tanzu 패키지 설치 설명서가 업데이트되었습니다. 장 11 TKG 서비스 클러스터에 표준 패키지 설치의 내용을 참조하십시오. ■ 부분적 편집 및 수정.
2023년 8월 10일	■ 공용 Harbor 레지스트리의 Tanzu 패키지 이미지를 감독자 서비스로 실행되는 개인 Harbor 레지스트리로 푸시하는 설명서가 추가되었습니다. 장 14 TKG 서비스 클러스터에서 개인 레지스트리 사용의 내용을 참조하십시오. ■ 부분적 편집 및 수정.
2023년 8월 7일	■ NSX를 사용하는 서비스 유형: LoadBalancer에 대한 ExternalTrafficPolicy 및 LoadBalancerSourceRanges에 대한 설명서가 제거되었습니다. 이러한 기능은 지원되지 않습니다.
2023년 8월 2일	■ Tanzu CLI 및 kubectl을 사용하여 TKG 클러스터에 연결하기 위한 항목이 추가되었습니다. #unique_36의 내용을 참조하십시오. ■ 부분적 편집 및 수정.

개정	설명
2023년 7월 31일	■ 부분적 편집 및 수정.
2023년 7월 27일	■ TKG 2.2를 지원하도록 설명서가 업데이트되었습니다.
2023년 7월 21일	■ Tanzu Kubernetes 릴리스의 컨텐츠 라이브러리를 관리하는 데 필요한 사용 권한 목록이 추가되었습니다.
2023년 7월 12일	■ 편집하고 볼 편집자 및 뷰어 역할의 이름이 업데이트되었습니다. TKG 서비스 클러스터에 대한 ID 및 액세스 관리 정보의 내용을 참조하십시오.
2023년 7월 10일	■ 감독자의 TKG 클러스터에서 사용할 수 있는 컨텐츠 라이브러리를 편집하기 위한 지침이 추가되었습니다. 기존 컨텐츠 라이브러리 편집의 내용을 참조하십시오. ■ 감독자에서 TKG 클러스터에 대한 지원 번들 수집에 대한 항목이 업데이트되었습니다. 감독자의 TKG 클러스터 문제 해결을 위해 로그 끌어오기의 내용을 참조하십시오.
2023년 7월 7일	■ 감독자에서 TKG 2.0을 사용하여 프로비저닝할 수 있는 두 가지 다른 클러스터 유형을 설명하는 항목이 업데이트되었습니다. TKG 클러스터 프로비저닝 정보의 내용을 참조하십시오. ■ 사용자 지정 노드 볼륨 마운트를 사용하여 클러스터를 프로비저닝하기 위한 고려 사항이 포함된 항목이 추가되었습니다. 노드 볼륨 마운트 사용에 대한 고려 사항의 내용을 참조하십시오.
2023년 7월 3일	■ vSphere IaaS control plane 운영자 그룹 및 역할 생성에 대한 항목이 업데이트되었습니다. 플랫폼 운영자를 위한 전용 그룹 및 역할 생성의 내용을 참조하십시오.
2023년 6월 30일	■ Cert Manager를 설치하기 위한 스크립트가 업데이트되었습니다. #unique_41의 내용을 참조하십시오. ■ Tanzu Mission Control Self-Managed 인스턴스를 감독자에 등록하기 위한 지침이 추가되었습니다. Tanzu Mission Control Self-Managed를 감독자에 등록의 내용을 참조하십시오. ■ vDS 환경에 대한 VM 점프 호스트 지침이 업데이트되었습니다. Linux 점프 호스트 VM 생성의 내용을 참조하십시오.
2023년 6월 26일	■ vSphere 8 감독자에서 지원되는 세부 정보로 TKG 클러스터 API 목록이 업데이트되었습니다. TKG 클러스터 프로비저닝 정보의 내용을 참조하십시오. ■ 전용 TKG 클러스터 운영자 역할을 생성하기 위한 권한 목록이 업데이트되었습니다. 플랫폼 운영자를 위한 전용 그룹 및 역할 생성의 내용을 참조하십시오. ■ Tanzu CLI를 사용하여 Contour를 설치하기 위한 명령이 업데이트되었습니다. 패키지 저장소 생성의 내용을 참조하십시오.
2023년 6월 13일	■ 전용 TKG 클러스터 운영자 그룹 및 역할 생성에 대한 항목이 추가되었습니다. 플랫폼 운영자를 위한 전용 그룹 및 역할 생성의 내용을 참조하십시오. ■ 라우팅 가능한 포드 네트워크를 사용하여 v1beta1 클러스터를 생성하는 항목이 업데이트되었습니다. v1beta1 예시: 라우팅 가능한 포드 네트워크가 있는 클러스터의 내용을 참조하십시오. ■ 알려진 문제로 인해 라우팅 가능한 포드 네트워크가 있는 v1alpha3 Tanzu Kubernetes 클러스터 생성에 대한 항목이 제거되었습니다. 자세한 내용은 릴리스 정보를 참조하십시오.
2023년 6월 9일	■ 감독자에서 vSphere 네임스페이스를 제거하는 항목이 추가되었습니다(해당 vSphere 네임스페이스에 프로비저닝된 TKG 클러스터를 먼저 삭제하는 사전 요구 사항 포함). vSphere 네임스페이스 제거의 내용을 참조하십시오.
2023년 6월 6일	■ 문서 집합으로 구성된 다양한 자료를 탐색하기 위한 vSphere IaaS control plane 설명서 맵 그래픽이 추가되었습니다. vSphere IaaS 제어부에서 TKG 서비스 사용의 내용을 참조하십시오.

개정	설명
2023년 6월 5일	■ 장 2 TKG 서비스 클러스터 실행 및 장 9 TKG 서비스 클러스터 업데이트 섹션의 다양한 항목에 대한 편집이 변경되었습니다. ■ vSphere with Tanzu 인증서 가이드 및 기타 변경 사항에 대한 참조로 TKG 서비스 클러스터에 대한 TLS 인증서 관리 항목이 업데이트되었습니다.
2023년 6월 1일	■ 클러스터 프로비저닝 예시가 업데이트되었습니다. TanzuKubernetesCluster v1alpha3 API 사용 및 클러스터 v1beta1 API 사용의 내용을 참조하십시오. ■ 클러스터 확장/축소 예시가 업데이트되었습니다. Kubectl을 사용하여 수동으로 클러스터 확장/축소의 내용을 참조하십시오.
2023년 5월 26일	■ TKG 클러스터에 대한 롤링 업데이트 항목이 추가 세부 정보로 업데이트되었습니다. TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해의 내용을 참조하십시오.
2023년 5월 23일	■ TKG 2 대신 TKG 2.0을 사용하도록 설명서가 수정되었습니다.
2023년 5월 22일	■ 네임스페이스 포드 보안 정책을 생성하는 명령이 업데이트되었습니다. TKG 서비스 클러스터에 기본 포드 보안 정책 적용의 내용을 참조하십시오. ■ 업그레이드 워크플로 설명서가 업데이트되었습니다. #unique_51의 내용을 참조하십시오.
2023년 5월 12일	■ 3개 vSphere 영역에 배포된 감독자 인스턴스에서만 새 TKG 2 클러스터를 프로비저닝할 수 있다는 참고 사항이 추가되었습니다.
2023년 5월 11일	■ 영구 볼륨 확장에 대한 항목이 추가되었습니다. TKG 서비스 클러스터를 위한 영구 볼륨 확장의 내용을 참조하십시오.
2023년 5월 9일	■ 이 예에서 포드가 여러 영역에 배포되었음을 나타내기 위해 영역 토폴로지 그래픽이 업데이트되었습니다. TKG 서비스 클러스터용 참조 아키텍처의 내용을 참조하십시오. ■ 추가 스토리지 세부 정보로 vSphere 네임스페이스 클러스터에 TKG 서비스 사용 항목이 업데이트되었습니다. ■ 영구 스토리지 모니터링에 대한 추가 세부 정보로 장 8 TKG 서비스 클러스터 운영 항목이 업데이트되었습니다.
2023년 5월 4일	■ `--package`를 사용하도록 패키지 저장소 생성 항목이 업데이트되었습니다. ■ TKG 2 클러스터에 PSP가 여전히 필요하고 포드 보안 승인 컨트롤러가 지원되지 않는다는 점을 명확히 하기 위해 TKR 1.24 및 이전 버전에 대한 PSP 구성 항목이 업데이트되었습니다. ■ #unique_22을 업데이트했습니다.
2023년 5월 1일	■ 링크가 수정되었습니다.
2023년 4월 4일	■ vSphere 8 업데이트 1 릴리스에 대한 일반 업데이트 및 기능 향상이 적용되었습니다. ■ 감독자 및 TKG 2와의 TKR 호환성 확인, TKR NAME 문자열에서 `-zshippable` 접미사 제거, 고유한 TKR 구축을 포함하여 vSphere 8 U1 릴리스에 대해 TKG 서비스 클러스터에서 Kubernetes 릴리스 사용이 업데이트되었습니다. ■ Cluster v1beta1 API 규격 설명서가 업데이트되었습니다. ■ 예제와 함께 TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합이 업데이트되었습니다. ■ Windows에 대한 추가 예제와 함께 Kubectl용 텍스트 편집기 구성이 업데이트되었습니다. ■ 버그 수정 사항으로 Restic과 함께 독립형 Velero를 사용하여 감독자에서 TKG 클러스터 워크로드 백업 및 복원 항목이 업데이트되었습니다.

TKG 서비스 클러스터 실행

2

이 섹션에서는 구성 요소에 대해 설명하고 TKG 서비스 클러스터를 실행하기 위한 요구 사항을 나열합니다.

다음으로 아래 항목을 읽으십시오.

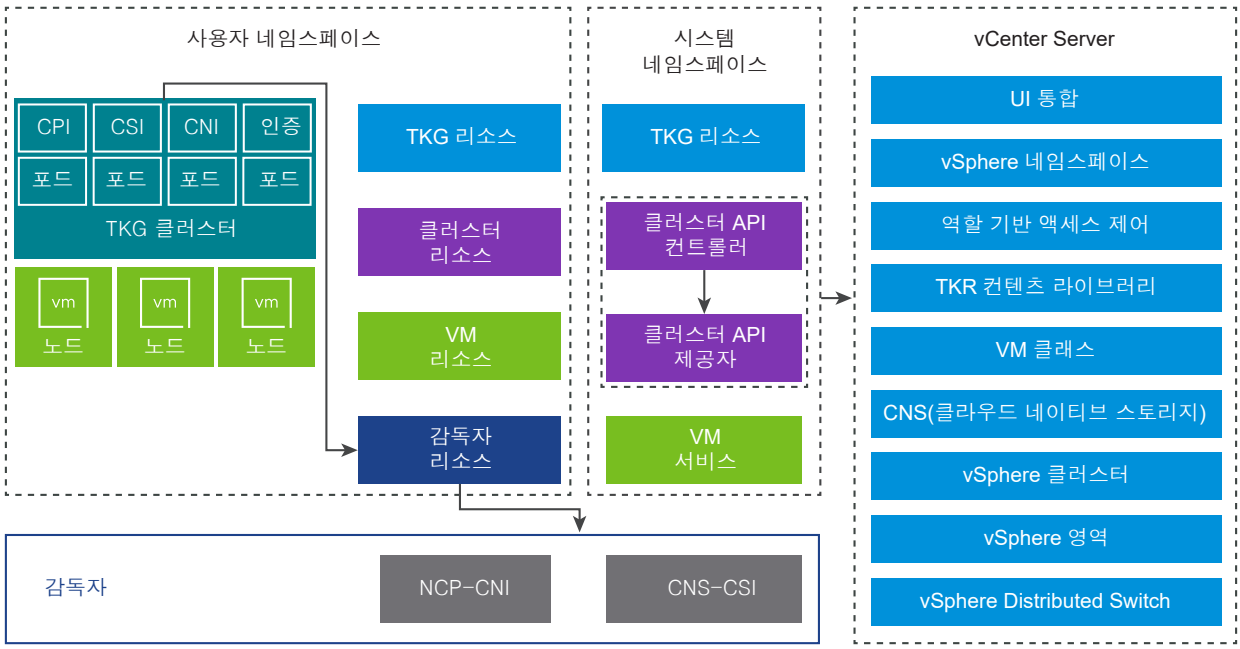
- TKG 서비스 구성 요소
- TKG 서비스 클러스터 배포
- TKG 서비스 클러스터용 참조 아키텍처

TKG 서비스 구성 요소

TKG 서비스는 Kubernetes 워크로드 클러스터의 셀프 서비스 수명 주기 관리를 제공합니다. 감독자의 TKG 서비스는 vSphere 환경에 최적화되어 있으며 vCenter, ESXi, 가상 네트워킹 및 클라우드 네이티브 스토리지를 포함한 기본 인프라와 통합됩니다. TKG 서비스를 사용하면 규격 Kubernetes 클러스터를 프로비저닝하고 업스트림 동시성을 유지할 수 있습니다.

TKG 서비스에는 vSphere IaaS control plane과 통합된 여러 구성 요소가 포함됩니다.

그림 2-1. TKG 서비스 구성 요소



워크로드 관리

워크로드 관리는 네이티브 vSphere 인프라에서 Kubernetes 제어부를 제공하는 VMware 솔루션입니다. 워크로드 관리를 사용하도록 설정하면 vSphere 환경에 하나 이상의 감독자를 배포할 수 있습니다. 워크로드 관리는 vCenter Server와 함께 번들로 제공되지만 별도로 라이센스가 부여됩니다.

감독자

감독자는 워크로드 클러스터를 관리하기 위한 제어부 역할을 하는 Kubernetes 클러스터입니다. Tanzu Kubernetes Grid 클러스터를 프로비저닝하고 운영하는 개발자를 지원하도록 감독자를 구성하십시오.

감독자 배포: vSphere 클러스터

감독자를 배포하는 기존 방법은 단일 vSphere 클러스터에 배포하는 것입니다. vSphere 클러스터는 vCenter Server에서 관리하는 ESXi 호스트의 모음입니다. vSphere 클러스터는 감독자를 지원하는 다음을 비롯한 특정 기능으로 구성되어야 합니다.

- VDS(vSphere Distributed Switch)로 연결된 여러 ESXi 호스트
- vSAN 또는 NFS와 같은 공유 스토리지가 구성됨
- vSphere HA 및 완전히 자동화된 DRS를 사용하도록 설정됨
- Lifecycle Manager를 사용하도록 설정됨
- 내장형 로드 밸런서가 있는 NSX 또는 외부 로드 밸런서가 있는 VDS로 네트워킹이 구성됨

구조적으로 감독자에서 TKG 운영 환경을 배포하는 경우 vCenter Server가 실행되는 관리부 호스트 또는 클러스터를 감독자를 사용하도록 설정할 계산부 클러스터에서 분리해야 합니다. vSphere 클러스터를 사용하여 vCenter Server를 호스팅하는 경우 이 클러스터에 DRS를 사용하도록 설정되어 있지 않아야 합니다. 자세한 내용은 vCenter 설명서를 참조하십시오.

감독자 배포: vSphere 영역

vSphere 8에는 vSphere 영역이 도입되었습니다. vSphere 클러스터를 vSphere 영역에 할당하여 감독자에 대한 고가용성 및 Fault Tolerance를 제공할 수 있습니다. 여러 vSphere 영역에 감독자를 배포하면 특정 가용성 영역에 TKG 클러스터를 프로비저닝할 수 있습니다.

단일 vSphere 클러스터에 감독자를 배포하는 경우 감독자와 vSphere 클러스터 사이에는 일대일 관계가 있습니다. 영역화된 감독자 배포에서 감독자는 3개 vSphere 클러스터에 걸쳐 확장되어 감독자의 TKG 클러스터에 대한 고가용성 및 장애 도메인을 제공합니다.

vSphere 관리자는 3개 vSphere 영역을 생성하여 vSphere 클러스터와 연결합니다. 3개 vSphere 영역에 감독자를 배포하려면 vSphere 클러스터 요구 사항 외에 다음과 같은 특별한 요구 사항이 있습니다.

- 3개 vSphere 영역이 VDS(vSphere Distributed Switch)로 연결됨
- 각 vSphere 영역에 단일 vSphere 클러스터가 포함됨
- 감독자를 정확히 3개 vSphere 영역에 배포해야 함

- vSphere 영역에서 vSphere 스토리지 정책을 사용할 수 있음

vSphere 네임스페이스

vSphere 네임스페이스는 하나 이상의 Tanzu Kubernetes Grid 클러스터가 프로비저닝되는 감독자의 네임스페이스입니다. 각 vSphere 네임스페이스에 대해 역할 기반 액세스 제어, 영구 스토리지, 리소스 제한, 이미지 라이브러리 및 가상 시스템 클래스를 구성합니다.

TKG 서비스

TKG 서비스는 Kubernetes 클러스터의 수명 주기를 관리하기 위한 사용자 지정 리소스 및 컨트롤러 집합을 정의하는 오픈 소스 Cluster API 프로젝트의 구현입니다. Tanzu Kubernetes Grid는 감독자의 구성 요소입니다.

TKG 서비스에는 TKG 클러스터(가상 시스템 서비스, Cluster API 및 클라우드 제공자 플러그인 포함)의 수명 주기를 관리하기 위한 세 가지 컨트롤러 계층이 있습니다.

VM Operator

가상 시스템 서비스 컨트롤러는 VM 및 연결된 vSphere 리소스 관리를 위한 선언적 Kubernetes 스타일의 API를 제공합니다. 가상 시스템 서비스는 재사용 가능한 추상적인 하드웨어 구성을 나타내는 가상 시스템 클래스의 개념을 소개합니다. TKG 서비스는 가상 시스템 서비스를 사용하여 워크로드 클러스터를 호스팅하는 제어부 및 작업자 노드 VM의 수명 주기를 관리합니다.

클러스터 API

클러스터 API 컨트롤러는 클러스터 생성, 구성 및 관리를 위한 선언적 Kubernetes 스타일의 API를 제공합니다. 클러스터 API에 대한 입력에는 클러스터를 설명하는 리소스, 클러스터를 구성하는 가상 시스템을 설명하는 리소스 집합 및 클러스터 추가 기능을 설명하는 리소스 집합이 포함됩니다.

클라우드 제공자 플러그인

TKG 서비스는 기본 vSphere 네임스페이스 리소스와 통합하는 데 필요한 구성 요소를 포함하는 워크로드 클러스터를 프로비저닝합니다. 이러한 구성 요소에는 감독자와 통합되는 클라우드 제공자 플러그인이 포함됩니다. TKG는 클라우드 제공자 플러그인을 사용하여 VMware CNS(클라우드 네이티브 스토리지)와 통합된 감독자에 영구 볼륨에 대한 요청을 전달합니다.

Tanzu Kubernetes 릴리스

Tanzu Kubernetes 릴리스는 Tanzu Kubernetes Grid 클러스터와 함께 사용하기 위해 VMware에서 서명 및 지원하는 Kubernetes 소프트웨어 배포및 추가 기능을 제공합니다.

각 Tanzu Kubernetes 릴리스는 가상 시스템 템플릿(OVA 파일)으로 배포됩니다. Tanzu Kubernetes Grid는 OVA 형식을 사용하여 TKG 클러스터의 가상 시스템 노드를 구성합니다. Tanzu Kubernetes 릴리스는 Kubernetes 버전 관리에 따라 버전이 지정되며 vSphere 인프라에 대한 OS 사용자 지정 및 최적화를 포함합니다.

Tanzu Kubernetes 릴리스 목록 및 감독자와의 호환성에 대해서는 Tanzu Kubernetes 릴리스의 릴리스 정보를 참조하십시오. vSphere IaaS control plane 지원 정책도 참조하십시오.

컨텐츠 라이브러리

Tanzu Kubernetes 릴리스는 vCenter 컨텐츠 라이브러리를 사용하여 TKG 클러스터에서 사용할 수 있습니다. 구독 컨텐츠 라이브러리를 생성하고 VMware에서 사용할 수 있게 되면 자동으로 TKR 을 받을 수 있습니다. 또는 로컬 컨텐츠 라이브러리를 사용하고 수동으로TKR을 업로드할 수 있습니다.

TKG 서비스 클러스터 구성 요소

TKG 서비스 클러스터에서 실행되는 구성 요소는 제품의 네 가지 영역(인증, 스토리지, 네트워킹 및 로드 밸런싱)에 대한 기능을 제공합니다.

인증 Webhook

인증 Webhook은 클러스터 내에서 포드로 실행되어 사용자 인증 토큰의 유효성을 검사합니다.

TKG 클러스터는 vCenter Single Sign-On 사용 및 OIDC(OpenID Connect) 프로토콜을 지원하는 외부 ID 제공자 사용이라는 두 가지 방법으로 인증을 지원합니다.

TKG는 감독자 및 TKG 클러스터 노드에서 Pinniped OIDC 클라이언트를 실행합니다. 감독자용 외부 OIDC 제공자를 구성하면 Pinniped 구성 요소가 자동으로 구성됩니다.

TKG는 kubectl용 vSphere 플러그인 및 Tanzu CLI를 포함하여 감독자 인증을 위한 다양한 클라이언트를 지원합니다.

CSI(Container Storage Interface)

반가상화 CSI 플러그인은 TKG 클러스터 내에서 실행되고 감독자를 통해 VMware Cloud CNS(네이티브 스토리지)와 통합되는 Kubernetes 포드입니다. TKG 클러스터 내에서 실행되는 Kubernetes 포드는 세 가지 유형의 가상 디스크(사용 후 삭제, 영구 볼륨, 컨테이너 이미지)를 마운트할 수 있습니다.

임시 스토리지

포드는 Kubernetes 개체와 같은 사용 후 삭제 데이터를 로그, 볼륨 및 구성 맵으로 저장하기 위한 임시 스토리지가 필요합니다. 임시 스토리지는 포드가 존재하는 동안 지속됩니다. 사용 후 삭제 데이터는 컨테이너를 다시 시작해도 유지되지만 포드가 삭제되면 사용 후 삭제 데이터를 저장하는 가상 디스크가 사라집니다.

영구 스토리지

TKG는 vSphere 스토리지 정책 프레임워크를 활용하여 스토리지 클래스를 정의하고 영구 볼륨을 예약합니다. TKG 클러스터는 VMware CNS(클라우드 네이티브 스토리지)와 통합된 감독자에 영구 볼륨에 대한 요청을 전달합니다. 영구 볼륨은 스토리지 클래스를 사용하여 동적으로 또는 수동으로 클러스터 노드에 프로비저닝됩니다.

컨테이너 이미지 스토리지

Kubernetes 포드 내의 컨테이너는 실행할 소프트웨어가 포함된 이미지를 사용합니다. 포드는 컨테이너에서 사용하는 이미지를 이미지 가상 디스크로 마운트합니다. 포드의 수명 주기가 완료되면 이미지 가상 디스크가 포드에서 분리됩니다. Kubelet은 이미지 레지스트리에서 컨테이너 이미지를 끌어와서 가상 디스크로 변환하여 포드 내에서 실행하는 작업을 담당합니다.

CNI(Container Network Interface)

컨테이너 네트워크 인터페이스 플러그인은 포드 네트워킹을 제공하는 CNI 플러그인입니다.

TKG 클러스터는 Antrea(기본값) 및 Calico와 같은 CNI(Container Network Interface) 옵션을 지원합니다. 또한 TKG는 라우팅 가능한 포드 네트워킹을 구현하기 위해 Antrea NSX Routed CNI를 제공합니다.

이 표에는 TKG 클러스터 네트워킹 기능과 구현이 요약되어 있습니다.

표 2-1. TKG 서비스 클러스터 네트워킹

끝점	제공자	설명
포드 연결	Antrea 또는 Calico	포드용 컨테이너 네트워크 인터페이스입니다. Antrea는 Open vSwitch를 사용합니다. Calico는 BGP와 함께 Linux 브리지를 사용합니다.
서비스 유형: ClusterIP	Antrea 또는 Calico	클러스터 내에서만 액세스할 수 있는 기본 Kubernetes 서비스 유형입니다.
서비스 유형: NodePort	Antrea 또는 Calico	Kubernetes 네트워크 프록시에 의해 각 작업자 노드에 열린 포트를 통한 외부 액세스를 허용합니다.
네트워크 정책	Antrea 또는 Calico	선택한 포드 및 네트워크 끝점 사이에서 허용되는 트래픽을 제어합니다. Antrea는 Open vSwitch를 사용합니다. Calico는 Linux IP 테이블을 사용합니다.

클라우드 제공자 구현

클라우드 제공자 구현을 통해 Kubernetes 로드 밸런서 및 수신 서비스를 생성할 수 있습니다.

표 2-2. TKG 로드 밸런싱

끝점	제공자	설명
서비스 유형: LoadBalancer	NSX 내장된 로드 밸런서(NSX 네트워크 스택의 일부) NSX Advanced Load Balancer(VDS 네트워킹에 사용하려면 별도 설치) HAProxy(VDS네트워킹에 사용하려면 별도 설치)	NSX 내장된 로드 밸런서의 경우 서비스 유형 정의당 가상 서버 하나. NSX Advanced Load Balancer는 이 설명서에서 해당 섹션을 참조하십시오. HAProxy는 이 설명서에서 해당 섹션을 참조하십시오. **참고** 일부 로드 밸런싱 기능은 지원되는 각 로드 밸런서 유형에서 사용하지 못할 수도 있습니다(예: 정적 IP).
클러스터 수신	타사 수신 컨트롤러	인바운드 포드 트래픽에 대한 라우팅을 제공합니다. Contour와 같은 타사 수신 컨트롤러를 사용할 수 있습니다.

TKG 서비스 클러스터 API

TKG 서비스는 TKG 클러스터의 수명 주기를 프로비저닝하고 관리하기 위한 두 가지 API를 제공합니다.

- Tanzu Kubernetes 클러스터에 대한 API 버전 `v1alpha3`
- ClusterClass를 기반으로 하는 클러스터에 대한 API 버전 `v1beta1`

v1alpha3 API를 사용하면 `TanzuKubernetesCluster` 유형의 규격 Kubernetes 클러스터를 생성할 수 있습니다. 이 유형의 클러스터는 빠른 프로비저닝을 위한 공통 기본값으로 미리 구성되며 사용자 지정할 수 있습니다. v1beta1 API를 사용하면 VMware에서 제공하는 기본 `ClusterClass`를 기반으로 `Cluster` 유형의 규격 Kubernetes 클러스터를 생성할 수 있습니다.

참고 vSphere IaaS control plane를 vSphere 7에서 vSphere 8로 업그레이드하려면 TKG 클러스터가 v1alpha2 API를 실행 중이어야 합니다. v1alpha2 API는 v7.0 업데이트 3에서 도입되었습니다. v1alpha1 API는 더 이상 사용되지 않습니다. 자세한 내용은 #unique_51 항목을 참조하십시오.

TKG 서비스 클러스터 클라이언트

vSphere 8 감독자의 TKG는 클러스터를 프로비저닝, 모니터링 및 관리하기 위한 다양한 클라이언트 인터페이스를 지원합니다.

- 감독자를 구성하고 배포된 TKG 클러스터를 모니터링하기 위한 vSphere Client.
- vCenter Single Sign-On을 사용하여 감독자 및 TKG 클러스터로 인증하기 위한 kubectl용 vSphere 플러그인.
- TKG 클러스터의 수명 주기를 선언적으로 프로비저닝 및 관리하고 감독자와 상호 작용하기 위한 kubectl.
- 컨테이너 레지스트리에/에서 이미지를 푸시하고 끌어오기 위한 vSphere Docker 자격 증명 도우미.
- 명령을 사용하여 클러스터를 프로비저닝하고 Tanzu 패키지를 설치하기 위한 Tanzu CLI.

- TKG 클러스터를 관리하기 위한 Tanzu Mission Control 웹 인터페이스.

TKG 서비스 클러스터 배포

TKG 서비스 클러스터를 배포하려면 TKG 서비스를 설치 또는 업그레이드하고, 컨텐츠 라이브러리를 사용하여 Tanzu Kubernetes 릴리스를 저장하고, TKG클러스터를 호스팅하도록 하나 이상의 vSphere 네임스페이스를 구성합니다.

TKG 서비스 클러스터 배포

TKG 서비스 클러스터를 배포하려면 **워크로드 관리** 사용 설정 프로세스를 완료하고 감독자 인스턴스를 구성합니다.

참고 지침은 "vSphere IaaS 제어부 설치 및 구성" 항목을 참조하십시오.

워크로드 관리를 사용하도록 설정하고 감독자를 배포한 후에는 TKG 서비스 클러스터 프로비저닝을 준비하기 위해 다음 작업을 완료합니다.

작업	설명
TKG 서비스 설치 또는 업그레이드	장 3 TKG 서비스 설치 및 업그레이드의 내용을 참조하십시오.
CLI를 설치하고 감독자에 연결	장 4 TKG 서비스 클러스터에 대한 ID 및 액세스 구성의 내용을 참조하십시오.
TKr(Tanzu Kubernetes 릴리스)에 대한 컨텐츠 라이브러리 생성 및 동기화	컨텐츠 라이브러리는 TKG 환경을 VMware 컨텐츠 전송 네트워크와 통합하며 이를 통해 TKr이 전송됩니다. 요구 사항에 따라 구독 또는 로컬 컨텐츠 라이브러리를 사용할 수 있습니다. 새 TKG 기능을 활용하려면 기존 TKr을 업그레이드해야 합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.
TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성 및 구성	vSphere 네임스페이스를 정의하면 TKG 서비스 클러스터를 호스팅하기 위한 네트워크 세그먼트가 생성됩니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오. 각 vSphere 네임스페이스는 다음을 사용하여 구성해야 합니다. ■ 클러스터 사용자 및 역할 ■ 동기화된 TKr 컨텐츠 라이브러리 ■ 연결된 VM 클래스 ■ TKG 서비스 클러스터 노드 및 영구 볼륨에 대한 vSphere 스토리지 정책 3개 vSphere 영역에 배포된 감독자를 사용하는 경우 vSphere 스토리지 정책이 `Zonal` 토폴로지를 지정해야 합니다. TKG 서비스 클러스터에 대한 vSphere 스토리지 정책 생성의 내용을 참조하십시오.
TKG 서비스 클러스터 프로비저닝	장 7 TKG 서비스 클러스터 프로비저닝의 내용을 참조하십시오.
TKG 서비스 클러스터 운영	장 8 TKG 서비스 클러스터 운영의 내용을 참조하십시오.
클러스터 워크로드 배포	장 12 TKG 서비스 클러스터에 워크로드 배포의 내용을 참조하십시오.

작업	설명
TKG 서비스 클러스터 유지 보수	장 9 TKG 서비스 클러스터 업데이트의 내용을 참조하십시오.
TKG 서비스 클러스터 문제 해결	장 21 TKG 서비스 클러스터 문제 해결의 내용을 참조하십시오.

TKG 서비스 클러스터용 참조 아키텍처

이 항목에서는 다양한 배포 토폴로지가 있는 TKG 서비스 클러스터에 대한 참조 아키텍처 집합을 제공합니다.

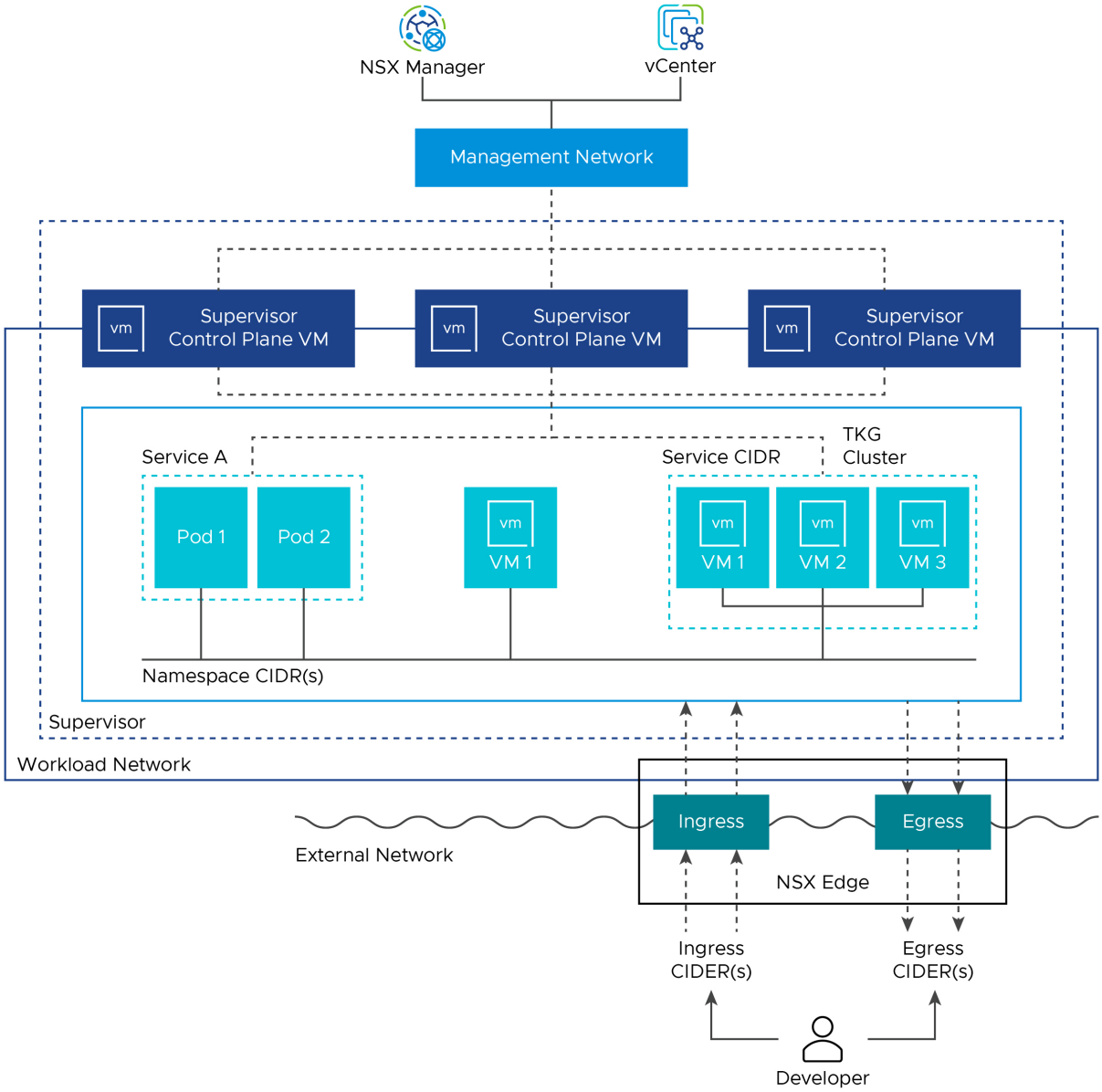
NSX 토폴로지가 있는 TKG 서비스 클러스터

참조 아키텍처는 NSX 네트워킹을 사용하는 감독자의 TKG 클러스터를 보여줍니다. 이러한 환경에서 관리부는 vCenter Server 및 NSX Manager를 호스팅합니다. 계산부는 NSX Edge 노드 및 감독자 노드를 호스팅합니다.

다음과 같은 NSX 네트워크 개체가 있습니다.

- Tier-1 게이트웨이(라우터)
- 게이트웨이에 연결된 세그먼트(스위치)
- 로드 밸런서 서버
- 각 TKG 클러스터 제어부 가상 서버에 대한 서버 풀
- 각 Kubernetes 서비스 로드 밸런서 인스턴스에 대한 가상 서버

그림 2-2. NSX 토폴로지가 있는 TKG 서비스 클러스터

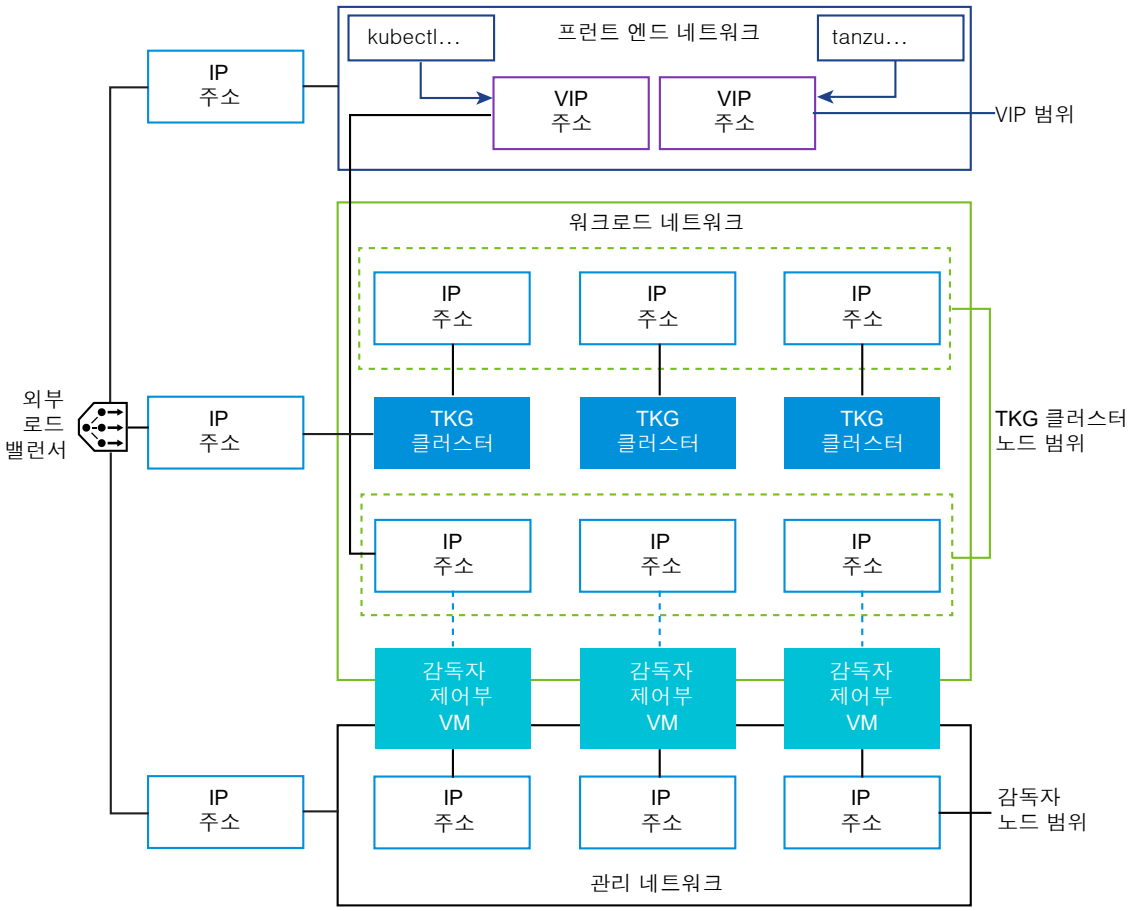


VDS 토폴로지가 있는 TKG 서비스 클러스터

참조 아키텍처는 VDS 네트워킹 및 외부 로드 밸런서를 사용하는 TKG 서비스 클러스터를 보여줍니다. 이러한 환경에는 다음과 같은 네트워크가 있습니다.

- 감독자 제어부 VM에 대한 관리 네트워크
- TKG 클러스터에 대한 워크로드 네트워크
- 개발자가 TKG 서비스 클러스터에 연결하는 데 사용하는 프런트 엔드 네트워크

그림 2-3. VDS 토폴로지가 있는 TKG 서비스 클러스터



vSphere 영역 토폴로지가 있는 TKG 서비스 클러스터

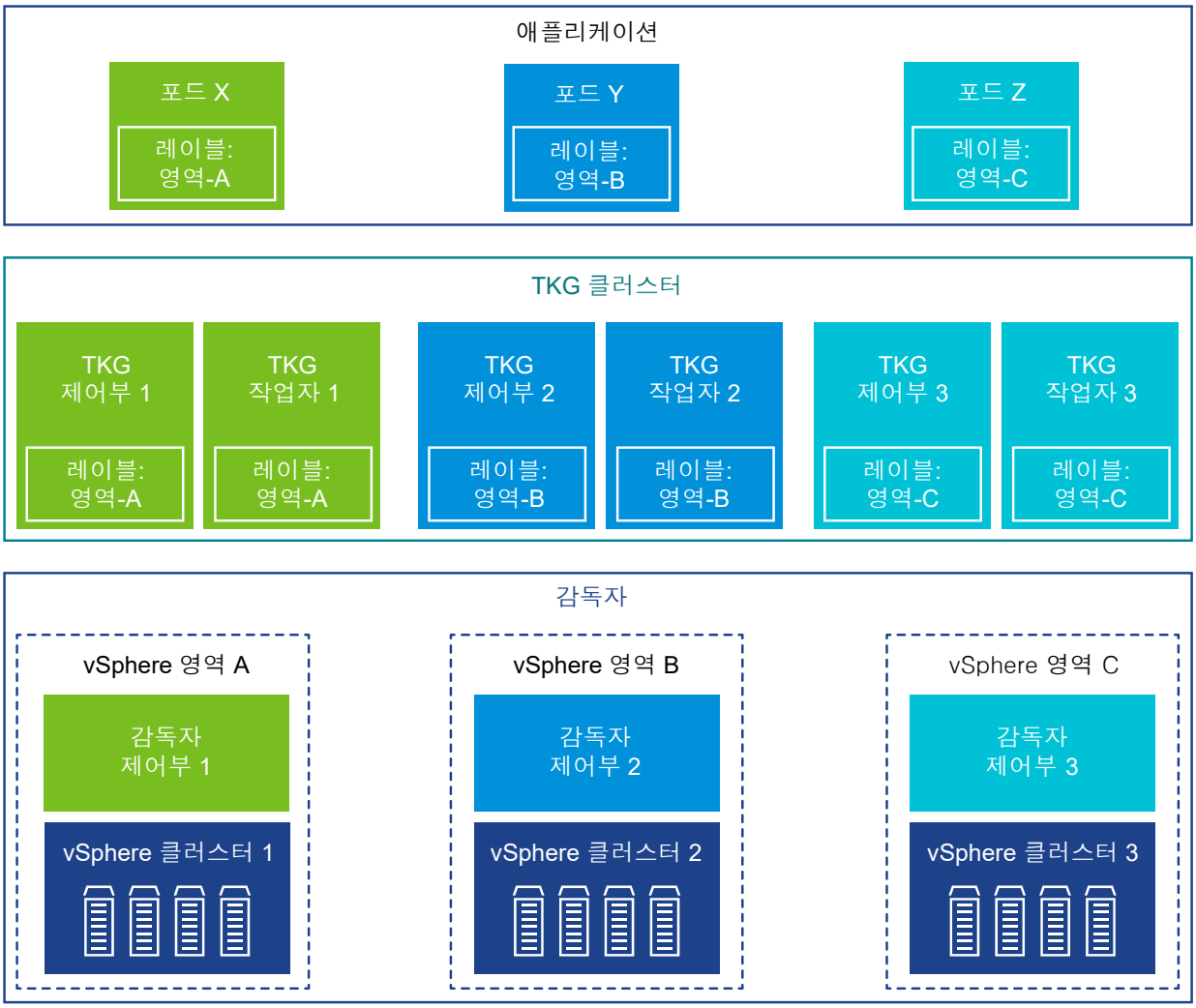
참조 아키텍처는 여러 vSphere 영역에 배포된 TKG 클러스터를 나타냅니다. 각 vSphere 영역은 공유 스토리지 및 서비스 품질을 사용하도록 설정된 vSphere 클러스터(vCenter에서 관리되고 vSphere Distributed Switch에 의해 연결된 ESXi 호스트의 모음)에 매핑됩니다.

영역이 지정된 토폴로지에서는 세 개 vSphere 영역에 감독자를 배포합니다. 시스템은 각 vSphere 영역에 감독자 제어부를 배치하여 장애 발생 시 고가용성을 지원합니다.

감독자에서 TKG 클러스터를 프로비저닝할 때 클러스터는 vSphere 영역를 인식합니다. 영역 토폴로지는 고가용성 워크로드에 대한 장애 도메인을 지원합니다. 필요한 경우 주석을 사용하여 특정 영역에서 워크로드를 실행할 수 있습니다.

참고 vSphere 영역은 vSphere 8의 새로운 기능입니다. 따라서 vSphere 영역은 감독자 및 TKG 클러스터의 새로운 배포만 지원합니다.

그림 2-4. vSphere 영역 토폴로지가 있는 TKG 서비스 클러스터



TKG 서비스 설치 및 업그레이드

3

이 섹션은 TKG 서비스를 설치 및 업그레이드하는 방법에 대한 정보를 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 사용
- TKG 서비스 상태 확인
- 새 버전의 TKG 서비스 등록
- TKG 서비스 버전 업그레이드
- TKG 서비스 문제 해결

TKG 서비스 사용

VMware Tanzu Kubernetes Grid 서비스(TKG 서비스)를 사용하면 vSphere IaaS control plane에서 Kubernetes 워크로드 클러스터를 배포할 수 있습니다. 이 TKG 서비스는 워크로드 중단 없이 독립적인 릴리스 및 비동기 업그레이드를 제공합니다.

TKG 서비스 소개

vSphere 8.0 업데이트 3부터 Tanzu Kubernetes Grid는 감독자 서비스로 설치됩니다. 이 아키텍처 변경은 vSphere IaaS control plane 릴리스에서 TKG를 분리하고 vCenter Server및 감독자와 관계없이 TKG 서비스를 업그레이드할 수 있도록 합니다.

TKG 서비스 3.0은 감독자 제어부 노드에 설치되고 실행됩니다. TKG 서비스는 Carvel 패키지의 중첩된 모음으로 제공됩니다. 코어 감독자 서비스인 TKG 서비스는 인터넷이 제한된 환경에서도 업그레이드할 수 있지만 제거하거나 다운그레이드할 수는 없습니다. **워크로드 관리 > 서비스** 탭에서 TKG 서비스를 모니터링하고 관리할 수 있습니다. TKG 서비스 버전 업그레이드의 내용을 참조하십시오.

TKG 서비스 버전 3.1은 독립적으로 업그레이드하는 최초의 릴리스입니다. TKG 서비스 vCenter에 새 TKG 서비스 버전 등록 및 TKG 서비스 버전 업그레이드는 별개의 프로세스입니다.

TKG 서비스 3.0 설치

vSphere IaaS control plane 구성 요소를 필요한 버전으로 업그레이드하면 TKG 서비스 설치가 자동으로 수행됩니다. 자세한 내용은 TKG 서비스 릴리스 정보를 참조하십시오.

vCenter에 새 TKG 서비스 버전 등록

TKG 서비스 패키지는 vCenter Server와 함께 게시되고 VMware 공용 레지스트리로 푸시됩니다. TKG 서비스 등록은 vCenter Server 수준에서 수행됩니다. 새 버전의 TKG 서비스를 등록하는 옵션은 동기식 및 비동기식의 두 가지가 있습니다.

표 3-1. TKG 서비스 버전 등록 옵션

등록 방법	설명
동기식	최신 vCenter Server 릴리스에 대한 업데이트가 새 버전의 TKG 서비스로 자동 등록될 때까지 기다린 다음, 감독자를 업데이트하여 내장된 레지스트리를 새 버전으로 채웁니다.
비동기식	공용 레지스트리에서 새 TKG 서비스 버전 정의를 다운로드한 다음 vCenter Server에 수동으로 등록합니다.

동기식 등록을 수행하려면 시스템 업데이트가 필요합니다. vCenter Server를 업데이트하는 경우 새 TKG 서비스 버전이 감독자에 자동으로 등록됩니다. 그러나 자동 등록된 (새) 버전을 사용하려면 해당 vCenter Server에서 제공하는 vSphere 네임스페이스 릴리스와 함께 제공되는 버전으로 감독자를 업데이트해야 합니다. 감독자 업데이트 시 TKG 서비스용 Carvel 패키지 번들을 감독자의 내장된 레지스트리에서 사용할 수 있으며 배포할 준비가 되었습니다. 감독자 업그레이드는 자동으로 TKG 서비스를 업그레이드하지 않습니다. 원하는 버전을 배포하도록 선택해야 합니다.

현재 감독자 버전이 지원 기간 내에 있다고 가정하면 비동기 등록에는 vCenter Server 및 감독자 업데이트가 필요하지 않습니다. 비동기 등록에는 다음과 같은 워크플로가 있습니다.

1 감독자 서비스용 공용 레지스트리 사이트에서 서비스 정의 YAML 파일을 다운로드합니다.

2 서비스 정의를 vCenter Server로 업로드하여 새 버전의 TKG 서비스를 등록합니다.

TKG 서비스의 등록 세부 정보가 표에 요약되어 있습니다.

표 3-2. TKG 서비스 버전 등록

TKG 서비스 속성	vCenter 번들	공용 레지스트리
새 버전 등록	자동 등록	수동 등록
새로 등록된 버전 삭제	허용되지 않음	허용됨
이미지 위치	감독자 제어부의 내장된 레지스트리	공용 레지스트리

TKG 서비스 버전 업그레이드

감독자 수준에서 TKG 서비스 버전 업그레이드가 수행됩니다. TKG 서비스가 등록되면 대상 감독자에서 감독자 서비스로 배포하여 TKG 서비스를 업그레이드합니다. TKG 서비스 버전 업그레이드의 내용을 참조하십시오.

인터넷이 제한된 환경("에어갭")에서 TKG 서비스를 업그레이드하려면 vCenter Server를 업데이트하여 새 TKG 서비스 버전을 동기식으로 등록합니다. 설치할 버전을 선택하면 로컬 레지스트리가 새 TKG 서비스 버전을 설치하는 데 사용됩니다. vCenter에 새 TKG 서비스 버전 등록의 내용을 참조하십시오.

TKG 서비스 버전을 업그레이드할 때 시스템은 사전 검사를 수행하고 다음 두 가지 심각도 수준을 보고합니다.

- 비차단 주의
- 차단 오류

Kubernetes 버전 검사는 비차단 주의 검사의 예입니다. 감독자 버전 검사는 차단 오류의 예입니다. 자세한 내용은 감독자 서비스 설명서를 참조하십시오.

TKG 서비스 상태 확인

TKG 서비스의 상태를 확인하려면 이 항목을 참조하십시오.

이 작업을 완료하여 TKG 서비스가 핵심 감독자 서비스로 설치되었는지 확인하고 해당 상태를 확인합니다.

상태를 확인하는 방법에는 vSphere Client를 사용하는 방법과 kubectl을 사용하는 방법 두 가지가 있습니다. vSphere Client를 사용하여 상태를 확인하려면 vCenter Server에 로그인하고 **워크로드 관리 > 서비스**로 이동합니다.

kubectl을 사용하여 상태를 확인하려면 다음 단계를 완료합니다.

사전 요구 사항

이 작업은 모든 시스템 구성 요소를 업그레이드하고 TKG 서비스 3.0을 설치했다고 가정합니다. #unique_67/unique_67_Connect_42_TABLE_CED10728-0714-4D00-BF58-D4BBAA2A4D8E의 내용을 참조하십시오.

절차

1 kubectl을 사용하여 감독자에 로그인합니다.

```
kubectl vsphere login --server=<SUPERVISOR-IP-or-FQDN> --vsphere-username <VCENTER-SSO-USER>
```

2 다음 명령을 실행합니다.

```
kubectl get packageinstall --namespace vmware-system-supervisor-services
```

TKG 서비스가 설치되어 있는지 확인해야 합니다.

```
NAMESPACE                          NAME                PACKAGE NAME                    PACKAGE VERSION  DESCRIPTION                  AGE
vmware-system-supervisor-services  svc-tkg.vsphere.vmware.com tkg.vsphere.vmware.com 0.0.1-b836be7   Reconcile succeeded 17h
```

새 버전의 TKG 서비스 등록

TKG 서비스의 비동기식 업그레이드를 위해 vCenter Server에 새 버전의 TKG 서비스를 수동으로 등록하려면 이 항목을 참조하십시오.

이 작업은 TKG 서비스 버전을 비동기식으로 업그레이드하려는 경우에만 필요합니다. vCenter에 새 TKG 서비스 버전 등록의 내용을 참조하십시오.

사전 요구 사항

이 작업은 모든 시스템 구성 요소를 업그레이드하고 TKG 서비스 3.0을 설치했다고 가정합니다. TKG 서비스 3.0 설치의 내용을 참조하십시오.

절차

1. 브라우저를 사용하여 감독자 서비스 배포판 사이트 https://www.vmware.com/go/supervisor-service 로 이동합니다.
2. 사이트에서 TKG 서비스 `package.yaml` 파일을 다운로드합니다.
3. vSphere Client를 사용하여 vCenter Server에 로그인합니다.
4. **워크로드 관리 > 서비스**로 이동합니다.
5. 이름이 **Tanzu Kubernetes Grid Service**인 서비스 타일을 찾습니다.
6. **작업 > 새 버전 추가**를 선택합니다.
7. **업로드**를 클릭합니다.
8. 다운로드한 `package.yaml` 파일을 선택하고
9. **완료**를 클릭합니다.

결과

새 서비스 정의를 등록한 후 TKG 서비스 타일에 사용할 수 있는 TKG 서비스 버전이 둘 이상 표시될 수 있습니다. TKG 서비스를 업그레이드할 때 대상 버전을 선택합니다.

다음에 수행할 작업

TKG 서비스 버전 업그레이드.

TKG 서비스 버전 업그레이드

TKG 서비스 버전을 업그레이드하려면 이 항목을 참조하십시오.

다음 단계를 완료하여 TKG 서비스 버전을 업그레이드합니다.

감독자 수준에서 TKG 서비스 버전 업그레이드가 수행됩니다. vCenter Server에서 여러 감독자를 호스팅하는 경우 대상 감독자를 선택해야 합니다.

사전 요구 사항

이 작업은 모든 시스템 구성 요소를 업그레이드하고 TKG 서비스 3.0을 설치했다고 가정합니다. TKG 서비스 3.0 설치의 내용을 참조하십시오.

이 작업에서는 새 버전의 TKG 서비스 등록 또는 vCenter에 새 TKG 서비스 버전 등록 TKG 서비스의 새 버전을 등록했다고 가정합니다.

절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리** > **서비스**로 이동합니다.

3 **Tanzu Kubernetes Grid Service** 타일을 찾습니다.

4 **작업** > **감독자에 설치**를 선택합니다.

5 업그레이드할 대상 버전의 TKG 서비스를 선택합니다.

6 업그레이드할 TKG 서비스를 호스팅하는 대상 감독자를 선택합니다.

7 서비스 호환성을 확인하고 **확인**을 클릭합니다.

8 TKG 서비스가 업그레이드되었는지 확인합니다.

워크로드 관리 > **서비스** 페이지의 **Tanzu Kubernetes Grid Service** 타일에는 버전과 상태가 반영됩니다. `kubectl get tkr`을 사용하여 상태를 확인할 수도 있습니다.

TKG 서비스 문제 해결

TKG 서비스의 문제를 해결하려면 이 항목을 참조하십시오.

TKG 서비스 지원 번들

TKG 서비스 지원 번들은 감독자 지원 번들에 포함되어 있습니다. 지침은 감독자용 지원 번들 수집을 참조하십시오.

감독자 지원 번들 내에서 TKG 서비스 로그는 `var/logs/tkg-svs` 폴더에 있습니다.

감독자 서비스 및 코어 서비스 컨트롤러에서 관리되는 코어 서비스 컨트롤러의 로그는 `/var/log/vmware/wcp/`에 있습니다.

다음 명령을 사용하여 애플리케이션 플랫폼 로그를 끌어올 수 있습니다.

```
kubectl logs vmware-system-appplatform-lifecycle-xxx -n vmware-system-appplatform-operator-system
```

TKG 서비스 클러스터에 대한 ID 및 액세스 구성

4

이 섹션에서는 감독자의 TKG 서비스 클러스터에 대한 사용자 인증 및 권한 부여를 구성하기 위한 정보를 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터에 대한 ID 및 액세스 관리 정보
- TKG 서비스 클러스터용 CLI 도구 설치
- vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결
- 외부 ID 제공자를 사용하여 감독자의 TKG 클러스터에 연결
- Kubernetes 관리자 및 시스템 사용자로 TKG 서비스 클러스터에 연결
- 플랫폼 운영자를 위한 전용 그룹 및 역할 생성

TKG 서비스 클러스터에 대한 ID 및 액세스 관리 정보

DevOps 엔지니어는 vSphere IaaS control plane에 연결하여 TKG 서비스 클러스터의 수명 주기를 관리하고 프로비저닝합니다. 개발자는 TKG 서비스 클러스터에 연결하여 패키지, 워크로드 및 서비스를 배포합니다. 관리자는 문제를 해결하기 위해 TKG 서비스 클러스터 노드에 직접 액세스해야 할 수도 있습니다. 플랫폼에서는 각 사용 사례와 역할을 지원하는 ID 및 액세스 관리 도구와 방법을 제공합니다.

TKG 서비스 클러스터 액세스 범위가 vSphere 네임스페이스로 지정됨

vSphere 네임스페이스에서 TKG 서비스 클러스터를 프로비저닝합니다. vSphere 네임스페이스를 구성할 때 해당 DevOps 사용 권한(ID 소스, 사용자 및 그룹, 역할 포함)을 설정합니다. 플랫폼은 이러한 사용 권한을 해당 vSphere 네임스페이스에서 프로비저닝된 각 TKG 서비스 클러스터에 전파합니다. 플랫폼은 두 가지 인증 방법인 vCenter Single Sign-On 및 OIDC 준수 외부 ID 제공자를 지원합니다.

vCenter Single Sign-On 및 Kubectl을 사용한 인증

기본적으로 vCenter Single Sign-On은 감독자 및 TKG 서비스 클러스터를 포함한 환경에서 인증하는 데 사용됩니다. vCenter Single Sign-On은 vSphere 인프라에 대한 인증을 제공하고 AD/LDAP 시스템과 통합할 수 있습니다. 자세한 내용은 vCenter Single Sign-On을 사용한 vSphere 인증을 참조하십시오.

vCenter Single Sign-On을 사용하여 인증하려면 kubectl용 vSphere 플러그인을 사용합니다. 인증되면 kubectl을 사용하여 TKG 서비스 클러스터의 수명 주기를 선언적으로 프로비저닝 및 관리하고 감독자와 상호 작용합니다.

kubectl용 vSphere 플러그인 플러그인은 kubectl에 따라 다릅니다. `kubectl vsphere login` 명령으로 인증하는 경우 플러그인은 기본 인증을 사용하여 감독자의 /wcp/login 끝점에 대해 POST 요청을 발급합니다. vCenter Server는 감독자가 신뢰하는 JWT(JSON Web Token)를 발급합니다.

vCenter Single Sign-On을 사용하여 연결하려면 vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결 항목을 참조하십시오.

외부 ID 제공자 및Tanzu CLI를 사용하여 인증

OpenID Connect 프로토콜을 지원하는 외부 ID 제공자로 감독자를 구성할 수 있습니다. 일단 구성이 완료되면 감독자는 OAuth 2.0 클라이언트로 작동하며 Pinniped 인증 서비스를 통해 Tanzu CLI를 사용하여 클라이언트 연결을 제공합니다. Tanzu CLI는 TKG 서비스 클러스터의 수명 주기 관리 및 프로비저닝을 지원합니다. 각 감독자 인스턴스는 단일 외부 ID 제공자를 지원할 수 있습니다.

pinniped-auth CLI가 작동하도록 인증 플러그인 및 OIDC 발급자가 적절하게 구성되면 `tanzu login --endpoint`를 사용하여 감독자에 로그인할 때 시스템은 잘 알려진 몇 가지 configmaps를 조회하여 `pinniped config` 구성 파일을 구축합니다.

외부 OIDC 제공자를 사용하여 연결하려면 외부 ID 제공자를 사용하여 감독자의 TKG 클러스터에 연결 항목을 참조하십시오.

하이브리드 방식을 사용한 인증: Tanzu CLI를 사용한 vCenter SSO

vCenter Single Sign-On을 ID 제공자로 사용 중이고 Tanzu CLI를 사용하려는 경우 하이브리드 접근 방식을 사용하고 두 도구를 모두 사용하여 감독자에 로그인할 수 있습니다. 이 방식은 표준 패키지를 설치하는 데 유용할 수 있습니다. Tanzu CLI 및 vCenter SSO 인증을 사용하여 감독자에 연결의 내용을 참조하십시오.

DevOps 사용자 및 그룹

vSphere 네임스페이스를 구성할 때 설정하는 사용 권한은 DevOps 사용자가 TKG 서비스 클러스터의 수명 주기를 관리하기 위한 것입니다. 사용 권한을 할당할 DevOps 사용자 또는 그룹은 ID 소스에 있어야 합니다. DevOps 사용자는 ID 제공자 자격 증명을 사용하여 인증합니다.

DevOps 사용자는 프로비저닝된 클러스터에 워크로드를 배포하려는 개발자와 같은 다운스트림 사용자에게 클러스터 액세스 권한을 부여할 책임이 있습니다. 이러한 사용자는 ID 제공자를 사용하거나 Kubernetes에 대한 클러스터 역할 및 바인딩을 사용하여 인증합니다. 이에 대해서는 다음 섹션에 자세히 설명되어 있습니다.

참고 vSphere 네임스페이스 사용 권한은 TKG 서비스 클러스터를 생성하고 관리해야 하는 DevOps 사용자를 위한 전용 권한입니다. 이러한 클러스터의 사용자(예: 개발자)는 Kubernetes 인증 메커니즘을 사용합니다.

역할 사용 권한 및 바인딩

TKGS 클러스터를 위한 RBAC(역할 기반 액세스 제어) 시스템에는 vSphere 네임스페이스 사용 권한 및 Kubernetes RBAC 권한 부여의 두 가지 유형이 있습니다. vSphere 관리자는 사용자가 TKG 서비스 클러스터를 생성하고 운영할 수 있도록 vSphere 네임스페이스 사용 권한을 할당합니다. 클러스터 운영자는 Kubernetes RBAC를 사용하여 클러스터 액세스 권한을 부여하고 개발자에게 역할 사용 권한을 할당합니다. 개발자에게 TKG 서비스 클러스터에 대한 vCenter SSO 액세스 권한 부여의 내용을 참조하십시오.

vSphere 네임스페이스는 **편집할 수 있음**, **볼 수 있음** 및 **소유자**의 세 가지 역할을 지원합니다. 역할 사용 권한은 TKG 서비스 클러스터를 호스팅하는 vSphere 네임스페이스에 할당되고 범위가 지정됩니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

vSphere 네임스페이스에 대한 **편집할 수 있음** 역할 사용 권한이 부여된 사용자/그룹은 해당 vSphere 네임스페이스에서 TKG 서비스 클러스터를 생성하고, 읽고, 업데이트하고, 삭제할 수 있습니다. 또한 사용자/그룹을 **편집** 역할에 할당하면 시스템은 해당 vSphere 네임스페이스의 각 클러스터에 RoleBinding을 생성하고 `cluster-admin`이라는 Kubernetes ClusterRole에 권한을 바인딩합니다. `cluster-admin` 역할을 통해 사용자는 대상 vSphere 네임스페이스에서 TKG 서비스 클러스터를 프로비저닝하고 운영할 수 있습니다. 이 매핑은 대상 vSphere 네임스페이스에서 `kubectl get rolebinding` 명령을 사용하여 볼 수 있습니다.

```
kubectl get rolebinding -n tkgs-cluster-namespace
NAME                                                              ROLE
AGE
wcp:tkg-cluster-namespace:group:vsphere.local:administrators   ClusterRole/edit
33d
wcp:tkg-cluster-namespace:user:vsphere.local:administrator     ClusterRole/edit
33d
```

vSphere 네임스페이스에 대한 **볼 수 있음** 역할 사용 권한이 부여된 사용자/그룹은 해당 vSphere 네임스페이스에 프로비저닝된 TKG 서비스 클러스터 개체에 대한 읽기 전용 액세스 권한을 갖습니다. 그러나 **편집할 수 있음** 권한과 달리 **볼 수 있음** 역할의 경우 해당 vSphere 네임스페이스의 TKGS 클러스터에 생성된 Kubernetes RoleBinding이 없습니다. 그 이유는 Kubernetes에는 **볼 수 있음** 사용 권한이 부여된 사용자/그룹에 바인딩할 수 있는 동등한 읽기 전용 역할이 없기 때문입니다. `cluster-admin` 이외의 사용자의 경우 Kubernetes RBAC를 사용하여 액세스 권한을 부여합니다. 개발자에게 TKG 서비스 클러스터에 대한 vCenter SSO 액세스 권한 부여의 내용을 참조하십시오.

vSphere 네임스페이스에 대한 **소유자** 권한이 부여된 사용자/그룹은 해당 vSphere 네임스페이스에서 TKG 서비스 클러스터를 관리할 수 있으며 kubectl을 사용하여 추가 vSphere 네임스페이스를 생성하고 삭제할 수 있습니다. 소유자 역할에 사용자/그룹을 할당하면 시스템에서 ClusterRoleBinding이 생성되어 ClusterRole에 매핑됩니다. 그러면 사용자/그룹이 kubectl을 사용하여 vSphere 네임스페이스를 생성하고 삭제할 수 있습니다. 이 매핑은 동일한 방식을 사용하여 볼 수 없습니다. 이 매핑을 보려면 SSH를 통해 감독자 노드에 연결해야 합니다.

참고 소유자 역할은 vCenter Single Sign-On ID 소스의 사용자에게 지원됩니다. 외부 ID 제공자의 사용자/그룹에는 소유자 역할을 사용할 수 없습니다.

vSphere 사용 권한

이 표에는 다양한 vSphere IaaS control plane 개인 설정에 필요한 vSphere 사용 권한 유형이 있습니다. 필요한 경우 워크로드 관리를 위한 사용자 지정 vSphere SSO 그룹 및 역할을 생성할 수 있습니다. 플랫폼 운영자를 위한 전용 그룹 및 역할 생성의 내용을 참조하십시오.

표 4-1. vSphere with Tanzu 개인 설정에 대한 vSphere 사용 권한

개인 설정	vSphere 역할	vSphere SSO 그룹	vSphere 네임스페이스
VI/클라우드 관리자	관리자	관리자	SSO 사용자 및/또는 AD 사용자
DevOps/플랫폼 운영자	관리자가 아닌 역할 또는 사용자 지정 역할	ServiceProviderUsers	SSO 사용자 및/또는 AD 사용자
개발자	읽기 전용 또는 없음	없음	SSO 사용자 및/또는 AD 사용자

시스템 관리자 연결

관리자는 TKG 서비스 클러스터 노드에 `kubernetes-admin` 사용자로 연결할 수 있습니다. vCenter Single Sign-On 인증을 사용할 수 없는 경우 이 방법이 적합할 수 있습니다. 문제 해결을 위해 시스템 관리자는 SSH 및 개인 키를 사용하여 TKG 서비스에 `vmware-system-user`로 연결할 수 있습니다. Kubernetes 관리자 및 시스템 사용자로 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

TKG 서비스 클러스터용 CLI 도구 설치

vSphere IaaS control plane는 TKG 서비스 클러스터에 연결하고 해당 클러스터의 수명 주기를 관리하기 위한 다양한 CLI 도구를 제공합니다.

vSphere에 대한 Kubernetes CLI 도구 설치

vCenter Single Sign-On 사용자는 vSphere에 대한 Kubernetes CLI 도구를 사용하여 TKG 서비스 클러스터에 연결하고 상호 작용할 수 있습니다.

vSphere에 대한 Kubernetes CLI 도구 정보

vSphere에 대한 Kubernetes CLI 도구 다운로드 패키지(`vsphere-plugin.zip`)에는 두 가지 실행 파일(kubectl 및 kubectl용 vSphere 플러그인)이 포함되어 있습니다.

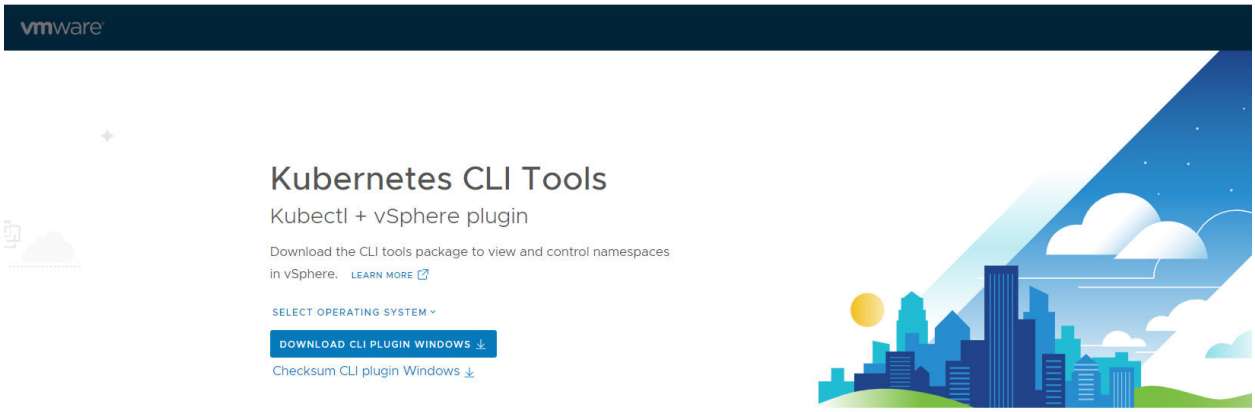
kubectl CLI에는 플러그형 아키텍처가 있습니다. kubectl용 vSphere 플러그인은 vCenter Single Sign-On을 사용하여 감독자 및 TKG 클러스터에 연결할 수 있도록 kubectl에 사용 가능한 명령을 확장합니다.

참고 vSphere IaaS control plane는 x86/64 프로세서용 바이너리를 제공합니다.

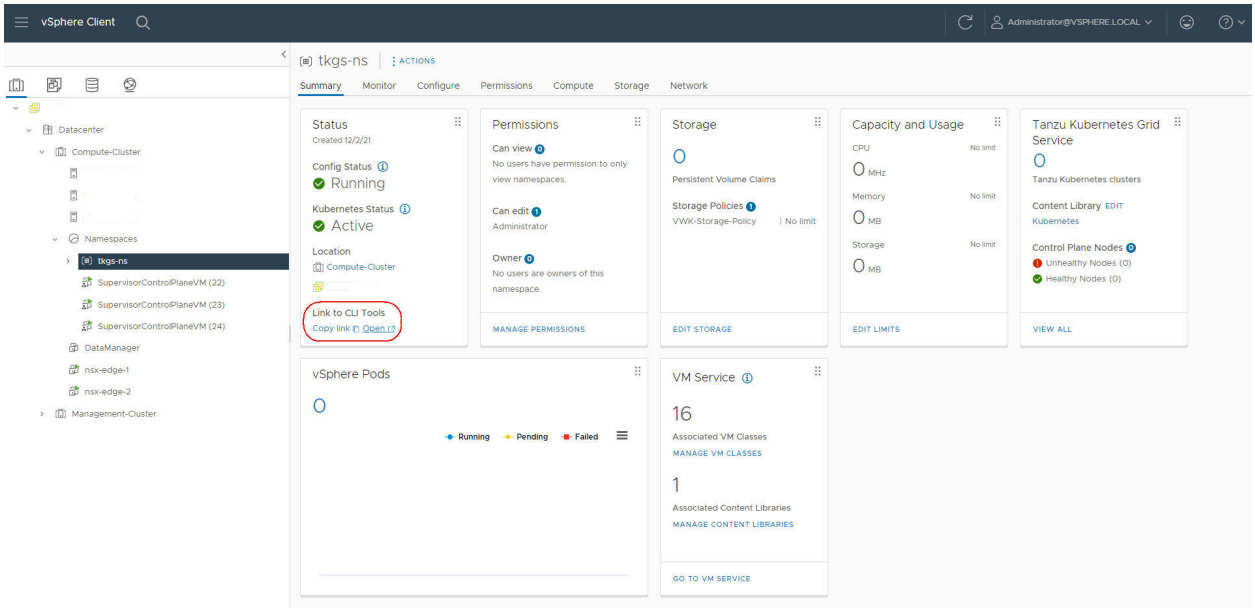
사전 요구 사항: vSphere 네임스페이스가 생성됨

vSphere에 대한 Kubernetes CLI 도구는 vSphere IaaS control plane DevOps 페이지에서 다운로드할 수 있습니다.

그림 4-1. vSphere DevOps 페이지



vSphere 네임스페이스 구성 패널에서 vSphere IaaS control plane DevOps 페이지에 액세스할 수 있습니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.



vSphere Client를 사용하여 vSphere에 대한 Kubernetes CLI 도구 설치

vCenter의 DevOps 페이지에서 vSphere에 대한 Kubernetes CLI 도구를 설치하려면 다음 단계를 완료합니다.

1 vSphere 관리자로부터 Kubernetes CLI 도구 다운로드 페이지 링크를 가져옵니다. 또는 vCenter Server에 액세스할 수 있으면 다음과 같이 링크를 가져옵니다.

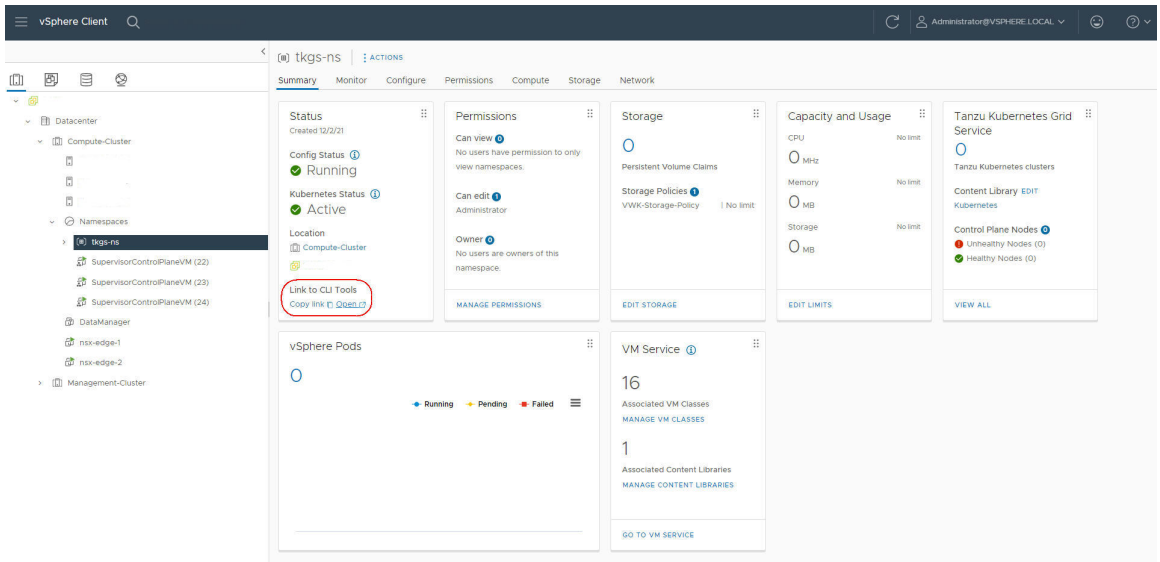
   a vSphere Client를 사용하여 vCenter Server에 로그인합니다.

   b **워크로드 관리 > 네임스페이스**로 이동합니다.

   c 작업 중인 vSphere 네임스페이스를 선택합니다.

   d **요약** 탭을 선택하고 이 페이지의 **상태** 영역을 찾습니다. (이미지를 참조하십시오.)

e **CLI 도구에 연결** 머리글 아래의 **열기**를 선택하여 다운로드 페이지를 엽니다. 또는 링크를 **복사**할 수 있습니다.



2 운영 체제를 선택합니다.

참고 필요에 따라 이 항목의 끝에 있는 OS별 설치 단계를 참조하십시오.

3 `vsphere-plugin.zip` 파일을 다운로드합니다.

4 작업 디렉토리에 ZIP 파일 컨텐츠의 압축을 풉니다.

5 두 실행 파일의 위치를 시스템의 PATH 변수에 추가합니다.

6 kubectl CLI의 설치를 확인하려면 셸, 터미널 또는 명령 프롬프트 세션을 시작하고 `kubectl` 명령을 실행합니다.

kubectl 배너 메시지 및 CLI에 대한 명령줄 옵션 목록이 표시됩니다.

7 kubectl용 vSphere 플러그인의 설치를 확인하려면 `kubectl vsphere` 명령을 실행합니다.

kubectl용 vSphere 플러그인 배너 메시지 및 플러그인에 대한 명령줄 옵션 목록이 표시됩니다.

명령줄에서 vSphere에 대한 Kubernetes CLI 도구 설치

Linux 또는 MacOS를 사용하는 경우 다음 명령을 실행하여 vSphere에 대한 Kubernetes CLI 도구 도구를 다운로드할 수 있습니다.

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리 > 감독자**를 선택합니다.

3 감독자 제어부의 로드 밸런서 IP 주소를 가져옵니다.

4 이에 대한 변수를 생성합니다.

5 다음 명령을 실행하여 도구를 설치합니다. (환경에 맞게 조정해야 할 수도 있습니다.)

Linux:

```
curl -LOk https://${SC_IP}/wcp/plugin/linux-amd64/vsphere-plugin.zip
unzip vsphere-plugin.zip
mv -v bin/* /usr/local/bin/
```

MacOS:

```
curl -LOk https://${SC_IP}/wcp/plugin/macos-darwin64/vsphere-plugin.zip
unzip vsphere-plugin.zip
mv -v bin/* /usr/local/bin/
```

6 `kubectl` 및 `kubectl vsphere` 명령을 실행하고 설치를 확인합니다.

Linux에 vSphere에 대한 Kubernetes CLI 도구 설치

Linux 호스트에 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

1 Linux용 `vsphere-plugin.zip` 파일을 다운로드합니다.

2 두 가지 실행 파일(kubectl 및 `kubectl-vsphere`)인 아카이브의 컨텐츠를 추출합니다.

3 두 실행 파일을 OS의 실행 파일 검색 경로(예: `/usr/local/bin`)에 넣습니다.

4 `kubectl vsphere` 명령을 실행하여 설치를 확인합니다.

5 `kubectl vsphere login --server=Supervisor-IP` 명령을 실행하여 감독자에 로그인합니다.

6 `kubectl config get-contexts` 명령을 실행하여 액세스할 수 있는 vSphere 네임스페이스 목록을 봅니다.

7 `kubectl config use-context CONTEXT` 명령을 실행하여 기본 컨텍스트를 선택합니다.

MacOS에 vSphere에 대한 Kubernetes CLI 도구 설치

MacOS 호스트에 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

1 MacOS용 `vsphere-plugin.zip` 파일을 다운로드합니다.

2 두 가지 실행 파일(kubectl 및 `kubectl-vsphere`)인 아카이브의 컨텐츠를 추출합니다.

3 두 실행 파일을 OS의 실행 파일 검색 경로(예: `/usr/local/bin`)에 넣습니다.

4 `kubectl vsphere` 명령을 실행하여 설치를 확인합니다.

5 `kubectl vsphere login --server=Supervisor-IP` 명령을 실행하여 감독자에 로그인합니다.

6 `kubectl config get-contexts` 명령을 실행하여 액세스할 수 있는 vSphere 네임스페이스 목록을 봅니다.

7 `kubectl config use-context CONTEXT` 명령을 실행하여 기본 컨텍스트를 선택합니다.

Windows에 vSphere에 대한 Kubernetes CLI 도구 설치

Windows 호스트에 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

1 MacOS용 `vsphere-plugin.zip` 파일을 다운로드합니다.

2 두 가지 실행 파일(`kubectl.exe` 및 `kubectl-vsphere.exe`)인 아카이브의 컨텐츠를 추출합니다.

3 두 실행 파일을 모두 시스템 경로에 넣습니다.

4 `kubectl vsphere` 명령을 실행하여 설치를 확인합니다.

5 `kubectl vsphere login --server=Supervisor-IP` 명령을 실행하여 감독자에 로그인합니다.

6 `kubectl config get-contexts` 명령을 실행하여 액세스할 수 있는 vSphere 네임스페이스 목록을 봅니다.

7 `kubectl config use-context CONTEXT` 명령을 실행하여 기본 컨텍스트를 선택합니다.

TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치

Tanzu CLI를 사용하여 TKG 서비스 클러스터의 수명 주기를 프로비저닝하고 관리할 수 있습니다.

TKG 서비스 클러스터 작업을 위한 Tanzu CLI를 설치합니다.

절차

1 설치 지침은 Tanzu CLI 제품 설명서를 참조하십시오.

2 호환성 매트릭스에서 적절한 버전을 확인하십시오.

참고 TKG 서비스 3.0의 경우 v1.1.0보다 큰 Tanzu CLI 버전을 사용하지 마십시오. 예를 들어 Tanzu CLI v1.2.0을 사용하지 마십시오.

참고 TKG 서비스 3.1.0부터 동일한 버전의 TKG 서비스가 있는 pinniped-auth CLI 플러그인을 다운로드하고 최신 버전의 imgpkg CLI 플러그인을 다운로드합니다.

3 대상 버전 관련 지침에 따라 Tanzu CLI를 설치합니다.

예: VMware Tanzu CLI v1.1.x 설치 및 사용: https://docs.vmware.com/kr/VMware-Tanzu-CLI/1.1/tanzu-cli/index.html

vSphere Docker 자격 증명 도우미 설치

vSphere Docker 자격 증명 도우미 CLI를 사용하여 내장된 Harbor 레지스트리에 컨테이너 이미지를 안전하게 푸시하고 내장된 Harbor 레지스트리에서 컨테이너 이미지를 끌어옵니다.

vSphere Docker 자격 증명 도우미를 사용하여 Docker 클라이언트를 컨테이너 레지스트리에 안전하게 연결합니다.

참고 vSphere Docker 자격 증명 도우미는 더 이상 사용되지 않는 내장된 Harbor 레지스트리와 함께 사용하도록 설계되었습니다. Harbor 감독자 서비스를 사용 중인 경우 Harbor 레지스트리 인증서를 사용하여 Docker 클라이언트 구성 항목을 참조하여 Docker를 사용하여 연결하십시오.

사전 요구 사항

Kubernetes CLI 도구 다운로드 페이지에는 vSphere Docker 자격 증명 도우미를 다운로드하는 링크가 포함되어 있습니다.

- Docker 클라이언트를 구성합니다.
- 감독자에서 사용하도록 설정된 내장된 Harbor 레지스트리에 대한 액세스 권한을 얻습니다.
- vSphere 관리자로부터 vSphere에 대한 Kubernetes CLI 도구 다운로드 페이지 링크를 받습니다.
- 또는 vCenter Server에 액세스할 수 있으면 다음과 같이 링크를 가져옵니다.
  - vSphere Client를 사용하여 vCenter Server에 로그인합니다.
  - **워크로드 관리 > 네임스페이스**로 이동하고 대상 vSphere 네임스페이스를 선택합니다.
  - **요약** 탭을 선택하고 **상태** 타일을 찾습니다.
  - **CLI 도구에 연결** 머리글 아래의 **열기**를 선택하여 다운로드 페이지를 엽니다. 또는 링크를 **복사**할 수 있습니다.

절차

1. 브라우저를 사용하여 사용 중인 환경에 해당하는 **Kubernetes CLI 도구** 다운로드 URL로 이동합니다.
2. vSphere Docker 자격 증명 도우미 섹션까지 아래로 스크롤합니다.
3. 운영 체제를 선택합니다.
4. `vsphere-docker-credential-helper.zip` 파일을 다운로드합니다.
5. 작업 디렉토리에 ZIP 파일 컨텐츠의 압축을 풉니다.

   `docker-credential-vsphere` 바이너리 실행 파일을 사용할 수 있습니다.
6. `docker-credential-vsphere` 바이너리를 Docker 클라이언트 호스트에 복사합니다.
7. 바이너리의 위치를 시스템 경로에 추가합니다.

   예를 들어 Linux:

   ```
   mv docker-credential-vsphere /usr/local/bin/docker-credential-vsphere
   ```

8 셸 또는 터미널 세션에서 `docker-credential-vsphere` 명령을 실행하여 vSphere Docker 자격 증명 도우미 설치를 확인합니다.

배너 메시지 및 CLI에 대한 명령줄 옵션 목록이 표시됩니다.

```
vSphere login manager is responsible for vSphere authentication.
It allows vSphere users to securely login and logout to access Harbor images.

Usage:
  docker-credential-vsphere [command]

Available Commands:
  help        Help about any command
  login       Login into specific harbor server and get authentication
  logout      Logout from Harbor server and erase user token

Flags:
  -h, --help   help for docker-credential-vsphere

Use "docker-credential-vsphere [command] --help" for more information about a command.
```

9 레지스트리에 로그인합니다.

우선 사용량을 확인합니다.

```
docker-credential-vsphere login -help
Usage:
  docker-credential-vsphere login [harbor-registry] [flags]

Flags:
  -h, --help               help for login
  -s, --service string     credential store service
      --tlscacert string   location to CA certificate (default "/etc/docker/certs.d/*.crt")
  -u, --user string        vSphere username and password
```

참고 vSphere Docker 자격 증명 도우미에서는 사용자 문자열이 모두 소문자여야 합니다. 모두 소문자를 사용하지 않는 경우 로그인이 작동할 수 있지만 후속 Docker 명령은 작동하지 않을 수 있습니다. 사용자 이름에 모두 소문자를 사용해야 합니다.

그런 후 다음 명령을 사용하여 로그인합니다.

```
docker-credential-vsphere login <container-registry-IP>
```

인증 토큰을 가져와서 저장되고 로그인됩니다.

```
docker-credential-vsphere login 10.179.145.77
Username: administrator@vsphere.local
Password: INFO[0017] Fetched username and password
INFO[0017] Fetched auth token
INFO[0017] Saved auth token
```

10 Harbor 레지스트리에서 로그아웃합니다.

```
docker-credential-vsphere logout 10.179.145.77
```

vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결

vCenter Single Sign-On 및 kubectl용 vSphere 플러그인을 단독으로 사용하거나 Tanzu CLI와 함께 사용하여 TKG 서비스 클러스터에 연결할 수 있습니다.

vCenter Single Sign-On 인증을 위한 보안 로그인 구성

감독자 및 TKG 서비스 클러스터에 안전하게 로그인하려면 적절한 TLS 인증서로 kubectl용 vSphere 플러그인을 구성하고 최신 버전의 플러그인을 실행하고 있는지 확인합니다.

감독자 CA 인증서

vSphere IaaS control plane은 kubectl용 vSphere 플러그인 명령인 `kubectl vsphere login …`을 사용하여 클러스터에 액세스할 수 있도록 vCenter Single Sign-On을 지원합니다.

kubectl용 vSphere 플러그인은 기본적으로 보안 로그인을 사용하고 신뢰할 수 있는 인증서가 필요하며, 기본값은 vCenter Server 루트 CA에서 서명한 인증서입니다. 플러그인은 `--insecure-skip-tls-verify` 플래그를 지원하지만 보안상의 이유로 이 플래그는 권장되지 않습니다.

kubectl용 vSphere 플러그인을 사용하여 감독자 및 TKG 서비스 클러스터에 안전하게 로그인하기 위한 두 가지 옵션이 있습니다.

옵션	지침
각 클라이언트 시스템에 vCenter Server 루트 CA 인증서를 다운로드하고 설치합니다.	Linux의 경우 아래 섹션을 참조하십시오. vCenter에 대한 신뢰할 수 있는 루트 CA 인증서를 다운로드하여 Ubuntu 클라이언트에 설치 Windows와 Mac의 경우 VMware 기술 자료 문서 vCenter Server 루트 인증서를 다운로드하고 설치하는 방법을 참조하십시오.
감독자에 사용되는 VIP 인증서를 각 클라이언트 시스템이 신뢰하는 CA에서 서명한 인증서로 바꿉니다.	"vSphere IaaS 제어부 설치 및 구성" 의 내용을 참조하십시오.

참고 vCenter Single Sign-On, vCenter Server 인증서 관리 및 순환, 문제 해결을 비롯한 vSphere 인증에 대한 자세한 내용은 "vSphere 인증" 설명서를 참조하십시오.

TKG 클러스터 CA 인증서

`kubectl` CLI를 사용하여 TKG 클러스터 API 서버와 안전하게 연결하려면 TKG 클러스터 CA 인증서를 다운로드해야 합니다.

최신 버전의 kubectl용 vSphere 플러그인을 사용하는 경우에는 TKG 클러스터에 처음 로그인할 때 플러그인이 TKG 클러스터 CA 인증서를 kubeconfig 파일에 등록합니다. 이 인증서는 *`TANZU-KUBERNETES-CLUSTER-NAME`*`-ca`이라는 Kubernetes 암호에도 저장됩니다. 플러그인은 인증서를 사용하여 해당 클러스터의 CA(인증 기관) 데이터스토어에 CA 정보를 채웁니다.

감독자를 업데이트한 경우 최신 버전의 플러그인으로 업데이트해야 합니다.

vCenter에 대한 신뢰할 수 있는 루트 CA 인증서를 다운로드하여 Ubuntu 클라이언트에 설치

이 절차에 따라 vCenter Server에 대한 신뢰할 수 있는 루트 CA 인증서를 다운로드하고 Ubuntu 클라이언트에 설치하면 kubectl용 vSphere 플러그인을 사용하여 감독자 및 TKG 서비스 클러스터에 안전하게 로그인할 수 있습니다.

1 kubectl용 vSphere 플러그인을 설치합니다. vSphere에 대한 Kubernetes CLI 도구 설치의 내용을 참조하십시오.

2 **워크로드 관리**를 사용하도록 설정된 vCenter Server에 대한 신뢰할 수 있는 루트 CA 인증서를 다운로드합니다.

```
wget https://VC-IP-or_FQDN/certs/download.zip --no-check-certificate
```

3 현재 디렉토리에 `download.zip` 파일 컨텐츠의 압축을 풉니다.

```
unzip download.zip -d .
```

4 Linux 디렉토리로 경로를 변경합니다.

```
cd /certs/lin
```

5 `/certs/lin` 디렉토리에 CA 인증서를 나열(`ls`)합니다.

PEM 형식의 인증서 파일 두 개(`*.0` 및 `*.r1`)가 표시됩니다. PEM 형식의 인증서는 base64 형식으로 사람이 읽을 수 있으며 `----BEGIN CERTIFICATE----`으로 시작합니다.

6 인증서 파일에 `*.crt` 확장명을 추가합니다. 예:

```
cp dbad4059.0 dbad4059.0.crt
```

```
cp dbad4059.r1 dbad4059.r1.crt
```

7 파일을 `/etc/ssl/certs`의 OpenSSL 인증서 디렉토리에 복사합니다.

```
sudo cp dbad4059.0.crt /etc/ssl/certs
```

```
sudo cp dbad4059.r1.crt /etc/ssl/certs
```

8 감독자에 안전하게 로그인합니다.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USERNAME
```

9 TKG 서비스 클러스터에 안전하게 로그인합니다.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USERNAME --tanzu-kubernetes-cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

vCenter Single Sign-On 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성

vSphere 네임스페이스에 대한 사용 권한을 설정하여 거기에 프로비저닝된 TKG 2 클러스터에 vCenter Single Sign-On 사용자 및 그룹이 액세스할 수 있도록 합니다.

vSphere 네임스페이스를 생성한 후에는 사용자/그룹을 추가하고 역할을 할당하여 TKG 2 클러스터에 대해 구성합니다. TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

사전 요구 사항

사용자, 그룹 및 역할 사용 권한은 vSphere 네임스페이스 수준에서 설정됩니다. 감독자 및 TKG 2 클러스터에 액세스하려면 먼저 vSphere 네임스페이스를 생성해야 합니다. TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.

절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리 > 네임스페이스**를 선택합니다.

3 생성한 vSphere 네임스페이스를 선택합니다.

4 **사용 권한 > 사용 권한 추가**를 선택합니다.

5 **ID 소스**: vCenter SSO 사용자 및 그룹에 대해 **vsphere.local**을 선택합니다.

참고 외부 ID 제공자를 사용하는 경우 외부 ID 제공자를 사용하여 감독자의 TKG 클러스터에 연결 항목을 참조하십시오.

6 **사용자/그룹 검색**: TKG 클러스터 작업 또는 TKG 개발자에 대해 구성된 vCenter SSO 사용자 또는 그룹을 선택합니다.

7 **역할**: **볼 수 있음**, **편집할 수 있음** 또는 **소유자** 중에서 적절한 역할을 선택하여 사용자 또는 그룹을 역할에 할당합니다.

옵션	설명
볼 수 있음	vSphere 네임스페이스에서 TKG 클러스터 개체를 읽을 수 있습니다. Kubernetes 역할에 매핑된 사용 권한이 없습니다. 역할 사용 권한 및 바인딩의 내용을 참조하십시오.
편집할 수 있음	vSphere 네임스페이스에서 TKG 클러스터 개체를 생성, 읽기, 업데이트 및 삭제할 수 있습니다. vSphere 네임스페이스에 프로비저닝된 TKG 클러스터를 Kubernetes `cluster-admin`으로 운영할 수 있습니다. 역할 사용 권한 및 바인딩의 내용을 참조하십시오.
소유자	[편집할 수 있음]과 동일한 사용 권한이 있으며 kubectl을 사용하여 vSphere 네임스페이스를 생성하고 관리할 수 있는 추가 사용 권한이 포함됩니다. vCenter SSO에서만 사용할 수 있습니다. 역할 사용 권한 및 바인딩의 내용을 참조하십시오.

8 vSphere 네임스페이스 구성을 완료합니다. TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

kubectl을 사용하여 vCenter Single Sign-On 사용자로 감독자에 연결

kubectl용 vSphere 플러그인을 사용하여 감독자에 연결하고 vCenter Single Sign-On 자격 증명으로 인증합니다.

감독자에 로그인하면 kubectl용 vSphere 플러그인은 클러스터에 대한 컨텍스트를 생성합니다. Kubernetes에서 구성 컨텍스트에는 클러스터, 네임스페이스 및 사용자가 포함됩니다. `.kube/config` 파일에서 클러스터 컨텍스트를 볼 수 있습니다. 이 파일은 일반적으로 `kubeconfig` 파일이라고 합니다.

참고 기존 `kubeconfig` 파일이 있는 경우 각 클러스터 컨텍스트에 추가됩니다. kubectl용 vSphere 플러그인은 kubectl 자체에서 사용하는 KUBECONFIG 환경 변수를 고려합니다. 필요하지 않더라도 `kubectl vsphere login ...`을 실행하기 전에 이 변수를 설정하면 정보가 현재의 `kubeconfig` 파일에 추가되지 않고 새 파일에 기록되므로 유용할 수 있습니다.

사전 요구 사항

- vSphere에 대한 Kubernetes CLI 도구 설치.
- vCenter Single Sign-On 자격 증명을 가져옵니다.
- vSphere 관리자로부터 감독자 제어부의 IP 주소를 얻습니다.
- vSphere 관리자로부터 vSphere 네임스페이스의 이름을 얻습니다.
- vSphere 네임스페이스에 대한 **편집** 권한이 있는지 확인합니다. vCenter Single Sign-On 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성의 내용을 참조하십시오.
- 서명 CA를 신뢰 루트로 설치하거나 인증서를 신뢰 루트로 직접 추가하여 Kubernetes 제어부가 제공하는 인증서를 시스템에서 신뢰할 수 있는지 확인합니다. vCenter Single Sign-On 인증을 위한 보안 로그인 구성의 내용을 참조하십시오.

절차

1 로그인을 위한 명령 구문 및 옵션을 보려면 다음 명령을 실행합니다.

```
kubectl vsphere login --help
```

2 감독자에 연결하려면 다음 명령을 실행합니다.

```
kubectl vsphere login --server=<KUBERNETES-CONTROL-PLANE-IP-ADDRESS> --vsphere-username
<VCENTER-SSO-USER>
```

FQDN을 사용하여 로그인할 수도 있습니다.

```
kubectl vsphere login --server <KUBERNETES-CONTROL-PLANE-FQDN --vsphere-username <VCENTER-
SSO-USER>
```

예:

```
kubectl vsphere login --server=10.92.42.13 --vsphere-username administrator@example.com
```

```
kubectl vsphere login --server wonderland.acme.com --vsphere-username
administrator@example.com
```

이 작업은 Kubernetes API에 인증하기 위한 JWT(JSON Web Token)가 들어 있는 구성 파일을 생성합니다.

3 인증하려면 사용자 암호를 입력합니다.

감독자에 연결한 후에는 구성 컨텍스트가 액세스할 수 있는 것으로 표시됩니다. 예:

```
You have access to the following contexts:
tanzu-ns-1
tkg-cluster-1
tkg-cluster-2
```

4 액세스할 수 있는 구성 컨텍스트의 세부 정보를 보려면 다음 `kubectl` 명령을 실행합니다.

```
kubectl config get-contexts
```

CLI에는 사용 가능한 각 컨텍스트에 대한 세부 정보가 표시됩니다.

5 컨텍스트 간에 전환하려면 다음 명령을 사용합니다.

```
kubectl config use-context <example-context-name>
```

kubectl을 사용하여 vCenter Single Sign-On 사용자로 TKG 서비스 클러스터에 연결

kubectl용 vSphere 플러그인을 사용하여 TKG 클러스터에 연결하고 vCenter Single Sign-On 자격 증명을 사용하여 인증합니다.

Tanzu Kubernetes 클러스터에 로그인한 후 kubectl용 vSphere 플러그인은 클러스터에 대한 컨텍스트를 생성합니다. Kubernetes에서 구성 컨텍스트에는 클러스터, 네임스페이스 및 사용자가 포함됩니다. `.kube/config` 파일에서 클러스터 컨텍스트를 볼 수 있습니다. 이 파일은 일반적으로 `kubeconfig` 파일이라고 합니다.

참고 기존 `kubeconfig` 파일이 있는 경우 각 클러스터 컨텍스트와 함께 추가됩니다. kubectl용 vSphere 플러그인은 kubectl 자체에서 사용하는 KUBECONFIG 환경 변수를 고려합니다. 필요하지 않더라도 `kubectl vsphere login ...`을 실행하기 전에 이 변수를 설정하면 정보가 현재의 `kubeconfig` 파일에 추가되지 않고 새 파일에 기록되므로 유용할 수 있습니다.

사전 요구 사항

vSphere 관리자로부터 다음 정보를 얻습니다.

- vCenter Single Sign-On 자격 증명을 가져옵니다.

- 감독자 제어부의 IP 주소를 가져옵니다.
- vSphere 네임스페이스의 이름을 가져옵니다.
- vSphere에 대한 Kubernetes CLI 도구 설치.

절차

1 로그인을 위한 명령 구문 및 옵션을 보려면 다음 명령을 실행합니다.

```
kubectl vsphere login --help
```

2 Tanzu Kubernetes 클러스터에 연결하려면 다음 명령을 실행합니다.

```
kubectl vsphere login --server=SUPERVISOR-CLUSTER-CONTROL-PLANE-IP-OR-FQDN
--tanzu-kubernetes-cluster-name TKG-CLUSTER-NAME
--tanzu-kubernetes-cluster-namespace VSPHERE-NAMESPACE
--vsphere-username VCENTER-SSO-USER-NAME
```

예:

```
kubectl vsphere login --server=10.92.42.137
--tanzu-kubernetes-cluster-name tkg-cluster-01
--tanzu-kubernetes-cluster-namespace tkg-cluster-ns
--vsphere-username operator@example.com
```

또는 감독자가 FQDN(정규화된 도메인 이름)으로 사용하도록 설정된 경우:

```
kubectl vsphere login --server=wonderland.acme.com
--tanzu-kubernetes-cluster-name tkg-cluster-01
--tanzu-kubernetes-cluster-namespace tkg-cluster-ns
--vsphere-username operator@example.com
```

이 작업은 Kubernetes API에 인증하기 위한 JWT(JSON Web Token)가 들어 있는 구성 파일을 생성합니다.

3 인증하려면 vCenter Single Sign-On 암호를 입력합니다.

작업이 성공하면 `Logged in successfully` 메시지가 표시되고 클러스터에 기반하여 kubectl 명령을 실행할 수 있습니다. 명령이 `Error from server (Forbidden)`를 반환하는 경우 일반적으로 이 오류는 사용자에게 필요한 사용 권한이 없음을 의미합니다.

4 사용 가능한 컨텍스트의 목록을 가져오려면 다음 명령을 실행합니다.

```
kubectl config get-contexts
```

이 명령은 액세스 권한이 있는 구성 컨텍스트를 나열합니다. `tkg-cluster-01`와 같은 대상 클러스터의 구성 컨텍스트를 볼 수 있습니다.

5 대상 클러스터의 컨텍스트를 사용하려면 다음 명령을 실행합니다.

```
kubectl config use-context CLUSTER-NAME
```

6 클러스터 노드를 나열하려면 다음 명령을 실행합니다.

```
kubectl get nodes
```

이 클러스터의 제어부 및 작업자 노드가 표시됩니다.

7 클러스터 포드를 모두 나열하려면 다음 명령을 실행합니다.

```
kubectl get pods -A
```

사용자에게 액세스 권한이 있는 모든 Kubernetes 네임스페이스에 걸쳐 이 클러스터의 모든 포드가 표시됩니다. 워크로드를 배포하지 않은 경우 기본 네임스페이스에 포드가 표시되지 않습니다.

개발자에게 TKG 서비스 클러스터에 대한 vCenter SSO 액세스 권한 부여

개발자 사용자 및 개발 그룹은 TKG 서비스 클러스터의 대상 사용자입니다. TKG 서비스 클러스터가 프로비저닝되면 vCenter Single Sign-On 인증을 사용하거나 지원되는 외부 ID 제공자를 사용하여 개발자에게 액세스 권한을 부여할 수 있습니다.

개발자에 대한 인증

클러스터 관리자는 개발자와 같은 다른 사용자에게 클러스터 액세스 권한을 부여할 수 있습니다. 개발자는 사용자 계정을 사용하여 직접 클러스터에 포드를 배포하거나 간접적으로 서비스 계정을 사용할 수 있습니다.

- 사용자 계정 인증의 경우 TKG 서비스 클러스터가 vCenter Single Sign-On 사용자 및 그룹을 지원합니다. 사용자 또는 그룹은 vCenter Server에 대해 로컬이거나 지원되는 디렉토리 서버에서 동기화될 수 있습니다.
- 외부 OIDC 사용자 및 그룹은 vSphere 네임스페이스 역할에 직접 매핑됩니다.
- 서비스 계정 인증의 경우 서비스 토큰을 사용할 수 있습니다. 자세한 내용은 Kubernetes 설명서를 참조하십시오.

클러스터에 개발자 사용자 추가

개발자에게 클러스터 액세스 권한을 부여하려면 다음을 수행합니다.

1 사용자 또는 그룹에 대한 역할 또는 ClusterRole을 정의하고 클러스터에 적용합니다. 자세한 내용은 Kubernetes 설명서를 참조하십시오.

2 사용자 또는 그룹에 대한 RoleBinding 또는 ClusterRoleBinding을 생성하고 클러스터에 적용합니다. 다음 예를 참조하십시오.

RoleBinding 예

vCenter Single Sign-On 사용자 또는 그룹에 액세스 권한을 부여하려면 RoleBinding의 주체에 `name` 매개 변수에 대한 다음 값 중 하나를 포함해야 합니다.

표 4-2. 지원되는 사용자 및 그룹 필드

필드	설명
`sso:USER-NAME@DOMAIN`	예를 들어 로컬 사용자 이름(예: `sso:joe@vsphere.local`)을 입력합니다.
`sso:GROUP-NAME@DOMAIN`	예를 들어 디렉토리 서버의 그룹 이름이 vCenter Server(예: `sso:devs@ldap.example.com`)와 통합되었습니다.

다음 RoleBinding 예는 이름이 Joe인 vCenter Single Sign-On 로컬 사용자를 이름이 `edit`인 기본 ClusterRole에 바인딩합니다. 이 역할은 네임스페이스에 있는 대부분의 개체(이 경우 `default` 네임스페이스)에 대한 읽기/쓰기 액세스를 허용합니다.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rolebinding-cluster-user-joe
  namespace: default
roleRef:
  kind: ClusterRole
  name: edit                              #Default ClusterRole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: User
  name: sso:joe@vsphere.local             #sso:<username>@<domain>
  apiGroup: rbac.authorization.k8s.io
```

Tanzu CLI 및 vCenter SSO 인증을 사용하여 감독자에 연결

다음 단계에 따라 Tanzu CLI를 사용하여 감독자에 연결하고 vCenter Single Sign-On 사용자로 인증합니다.

사전 요구 사항

다음 사전 요구 사항을 완료하십시오.

1 vSphere에 대한 Kubernetes CLI 도구를 설치 및 구성합니다. vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

2 Tanzu CLI를 설치하고 초기화합니다. TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치의 내용을 참조하십시오.

Tanzu CLI 및 vCenter SSO를 사용하여 감독자에 연결

다음 단계를 완료하십시오.

1 vCenter SSO 사용자로 감독자에 연결합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS --vsphere-username
VCENTER-SSO-USER
```

이 작업은 kubeconfig를 Tanzu CLI가 사용하는 감독자 컨텍스트로 채웁니다.

2 컨텍스트를 감독자용 vSphere 네임스페이스로 전환합니다.

```
kubectl config get-contexts
```

```
kubectl config use-context <SUPERVISOR-CONTROL-PLANE-IP-ADDRESS>
```

3 Tanzu CLI 및 vCenter SSO를 사용하여 감독자에 로그인합니다.

```
tanzu context create context_name --kubeconfig ~/.kube/config --kubecontext SUPERVISOR-CONTROL-PLANE-IP-ADDRESS
```

형식 설명:

- `context_name`은 이 컨텍스트의 사용자 정의 이름(예: "supervisor")입니다.
- `--kubeconfig ~/.kube/config`는 로컬 `kubeconfig` 파일의 경로이며, 기본값은 `~/.kube/config`이고 `KUBECONFIG` 환경 변수에 의해 설정되며 여기에는 vCenter SSO 사용자에 대한 감독자 구성 컨텍스트가 포함됩니다.
- `--kubecontext SUPERVISOR-CONTROL-PLANE-IP-ADDRESS`는 감독자의 컨텍스트이며, `SUPERVISOR_IP`(예: `10.179.144.55`)와 동일합니다.

4 Tanzu CLI 명령을 실행하고 연결을 확인합니다.

```
tanzu plugin list
```

```
tanzu cluster list -n VSPHERE-NS-FOR-TKG
```

외부 ID 제공자를 사용하여 감독자의 TKG 클러스터에 연결

외부 ID 제공자 및 Kubernetes용 Pinniped 인증 서비스를 사용하여 감독자의 TKG 클러스터에 연결할 수 있습니다.

TKG 서비스 클러스터와 함께 사용할 외부 IDP 구성

Okta와 같은 OIDC 준수 IDP(ID 제공자)를 사용하여 감독자를 구성할 수 있습니다. 통합을 완료하려면 감독자에 대한 콜백 URL을 사용하여 IDP를 구성합니다.

지원되는 외부 OIDC 제공자

OIDC 준수 ID 제공자를 사용하여 감독자를 구성할 수 있습니다. 이 표에는 일반 항목이 나열되어 있으며 구성 지침에 대한 링크가 포함되어 있습니다.

외부 IDP	구성
Okta	Okta를 사용한 OIDC 구성 예시 Okta를 Pinniped용 OIDC 제공자로 구성도 참조하십시오.
Workspace ONE	Workspace ONE Access를 Pinniped용 OIDC 제공자로 구성

외부 IDP	구성
Dex	Dex를 Pinniped용 OIDC 제공자로 구성
GitLab	GitLab을 Pinniped용 OIDC 제공자로 구성
Google OAuth	Google OAuth 2 사용

감독자의 콜백 URL을 사용하여 IDP 구성

감독자는 외부 ID 제공자에 대한 OAuth 2.0 클라이언트 역할을 합니다. 감독자 콜백 URL은 외부 IDP를 구성하는 데 사용되는 리디렉션 URL입니다. 콜백 URL은 "https://SUPERVISOR-VIP/wcp/pinniped/callback" 형식입니다.

참고 IDP 등록을 수행할 때 구성 중인 OIDC 제공자에서 콜백 URL을 "리디렉션 URL"이라고 지칭할 수 있습니다.

감독자의 TKG에서 사용할 외부 ID 제공자를 구성하는 경우 외부 ID 제공자에게 **워크로드 관리 > 감독자 > 구성 > ID 제공자** 화면의 vCenter Server에서 사용할 수 있는 **콜백 URL**을 제공합니다.

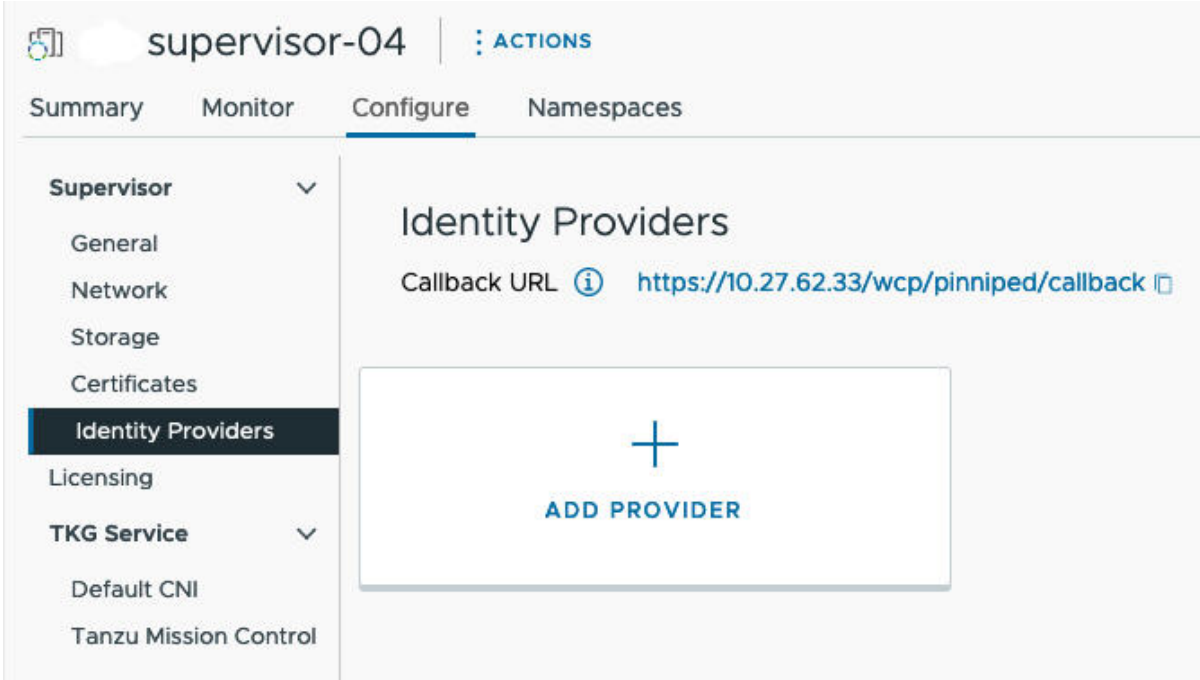
Okta를 사용한 OIDC 구성 예시

Okta를 사용하면 사용자가 OpenID Connect 프로토콜을 사용하여 애플리케이션에 로그인할 수 있습니다. Okta를 감독자의 Tanzu Kubernetes Grid에 대한 외부 ID 제공자로 구성하면 감독자 및 Tanzu Kubernetes Grid 클러스터의 Pinniped 포드가 vSphere 네임스페이스 및 워크로드 클러스터 둘 다에 대한 사용자 액세스를 제어합니다.

1 Okta와 vCenter Server 간에 OIDC 연결을 생성하는 데 필요한 ID 제공자 콜백 URL을 복사합니다.

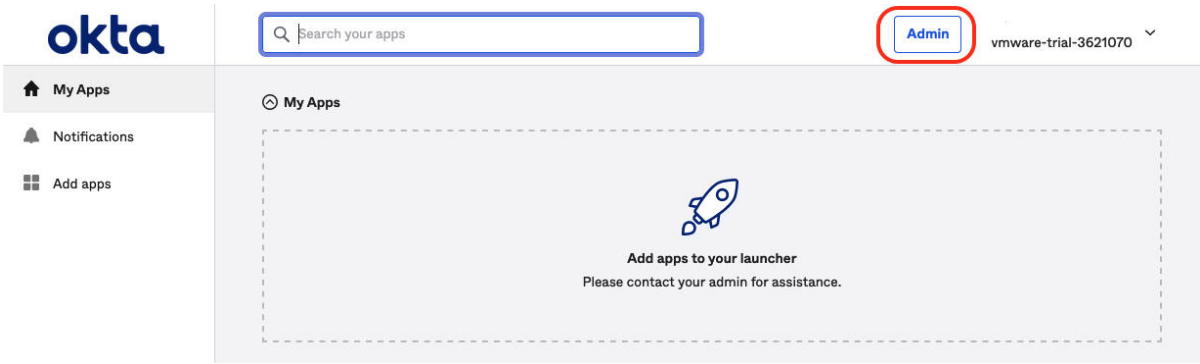
   vSphere Client를 사용하여 **워크로드 관리 > 감독자 > 구성 > ID 제공자**에서 ID 제공자 콜백 URL을 가져옵니다. 이 URL을 임시 위치에 복사합니다.

그림 4-2. IDP 콜백 URL



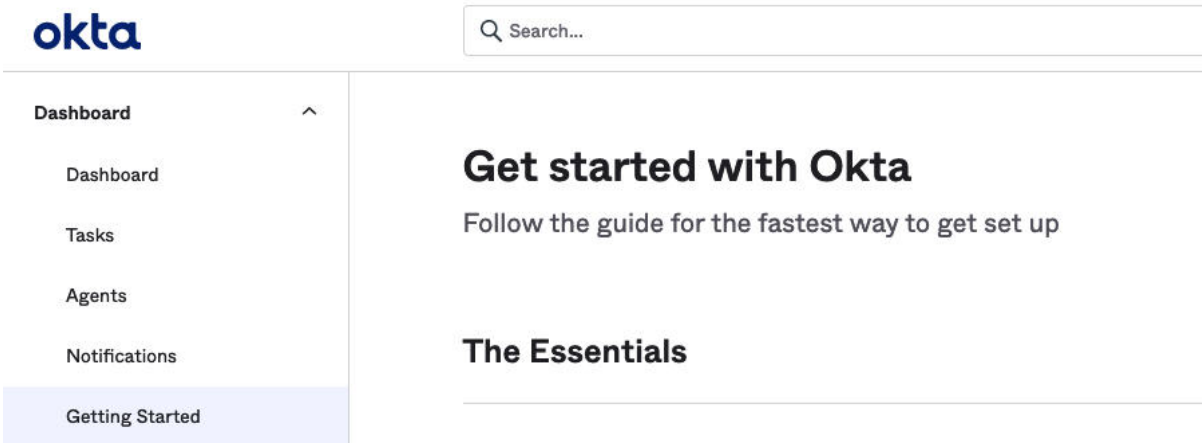
2 조직의 Okta 계정에 로그인하거나 https://www.okta.com/에서 평가판 계정을 생성합니다. **관리자** 버튼을 클릭하여 Okta 관리 콘솔을 엽니다.

그림 4-3. Okta 관리 콘솔



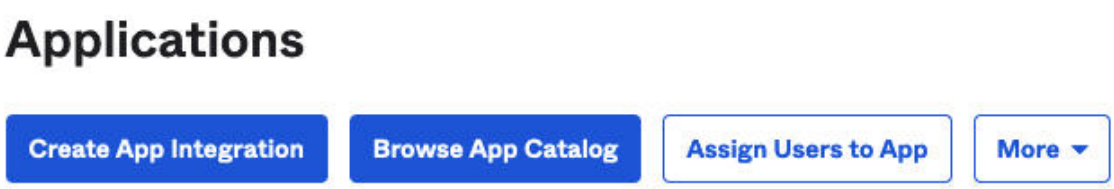
3 관리 콘솔의 Getting Started(시작) 페이지에서 **Applications(애플리케이션) > Applications(애플리케이션)**로 이동합니다.

그림 4-4. Okta Getting Started(시작)



4 **Create App Integration**(애플리케이션 통합 생성) 옵션을 선택합니다.

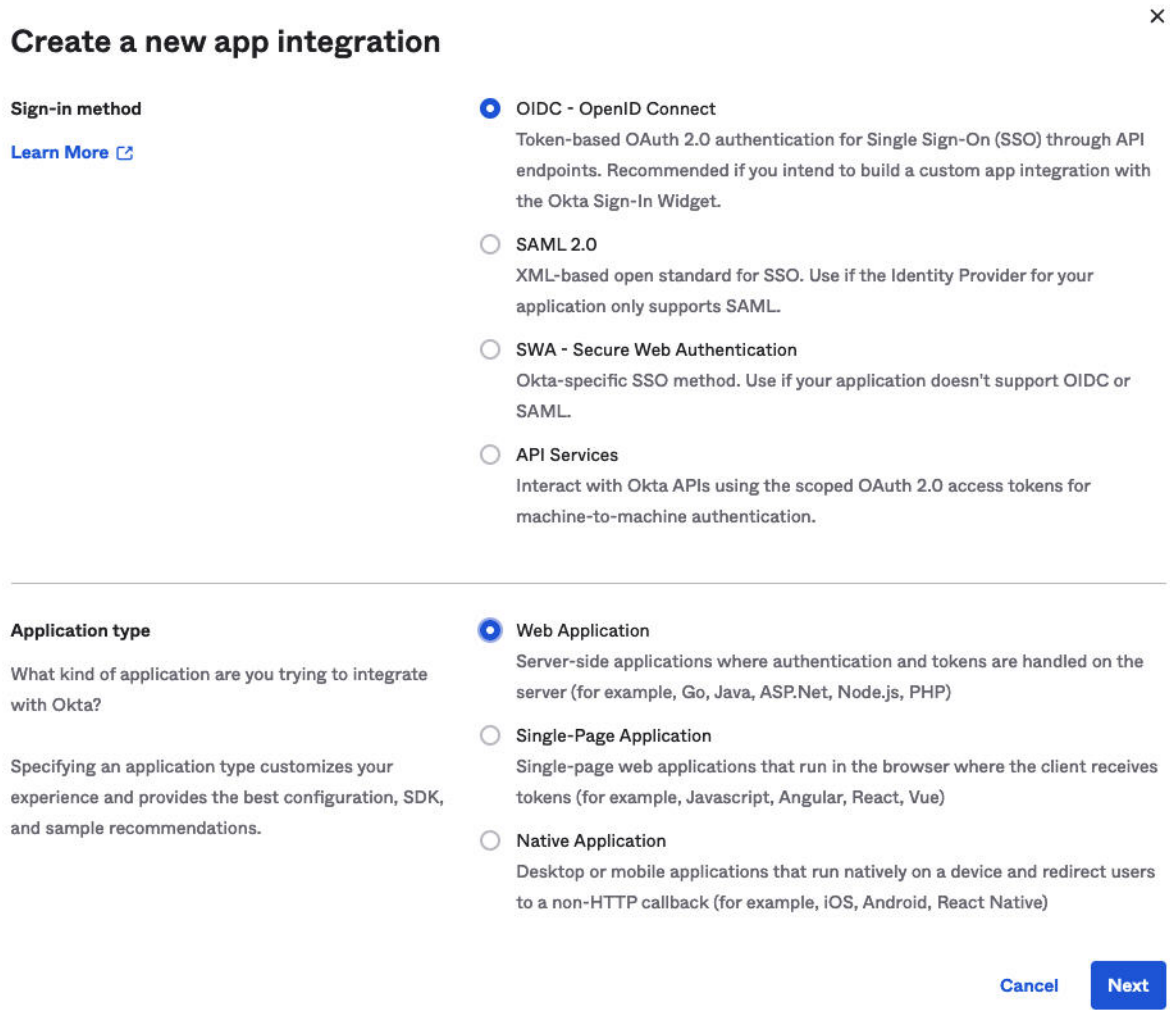
그림 4-5. Okta 애플리케이션 통합 생성



5 새 애플리케이션 통합을 생성합니다.

- Sign-in method(로그인 방법)를 **OIDC - OpenID Connect**로 설정합니다.
- Application type(애플리케이션 유형)을 **Web Application**(웹 애플리케이션)으로 설정합니다.

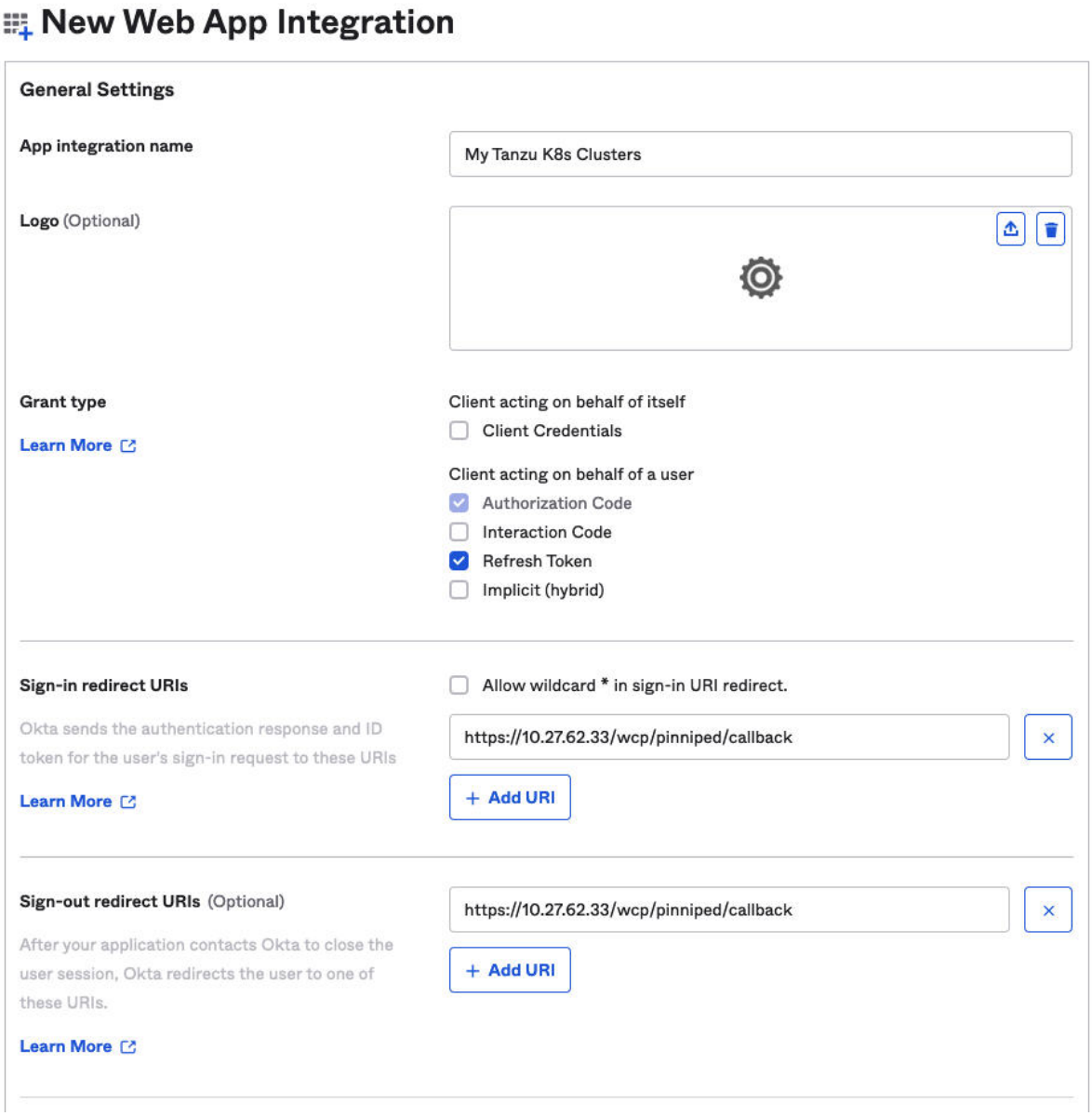
그림 4-6. Okta Sign-On 방법 및 애플리케이션 유형



6 Okta 웹 애플리케이션 통합 세부 정보를 구성합니다.

- 사용자 정의 문자열인 **App integration name**(애플리케이션 통합 이름)을 제공합니다.
- **Grant type**(권한 부여 유형) 지정: **Authorization Code**(인증 코드)를 선택하고 **Refresh Token**(새로 고침 토큰)도 선택합니다.
- Sign-in redirect URIs(로그인 리디렉션 URI): 감독자에서 복사한 ID 제공자 콜백 URL(1단계 참조)(예: https://10.27.62.33/wcp/pinnipend/callback)을 입력합니다.
- Sign-out redirect URIs(로그아웃 리디렉션 URI): 감독자에서 복사한 ID 제공자 콜백 URL(1단계 참조)(예: https://10.27.62.33/wcp/pinnipend/callback)을 입력합니다.

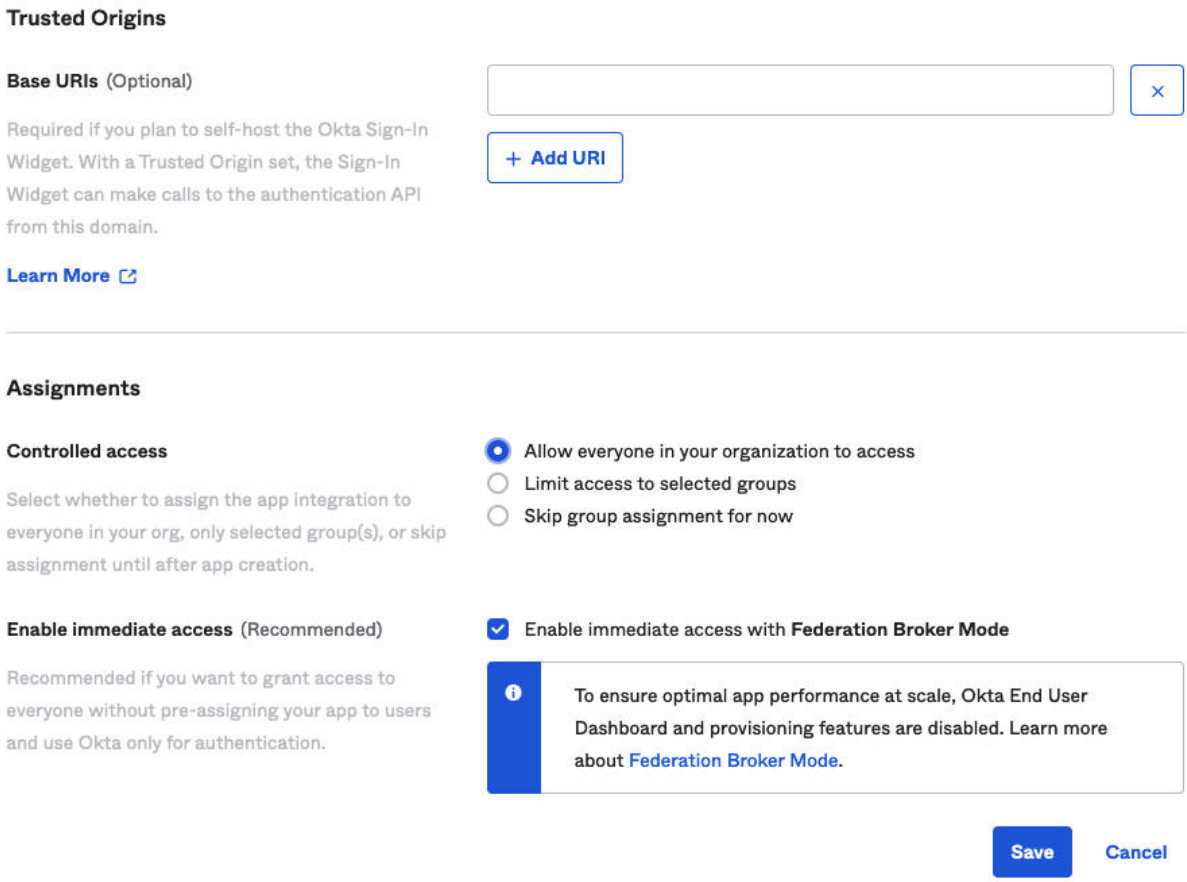
그림 4-7. Okta 웹 애플리케이션 통합 세부 정보



7 사용자 액세스 제어를 구성합니다.

Assignments(할당) > Controlled access(제어된 액세스) 섹션에서 조직에 존재하는 Okta 사용자 중 Tanzu Kubernetes Grid 클러스터에 액세스할 수 있는 사용자를 선택적으로 제어할 수 있습니다. 이 예시에서 조직에 정의된 모든 사람이 액세스할 수 있도록 허용합니다.

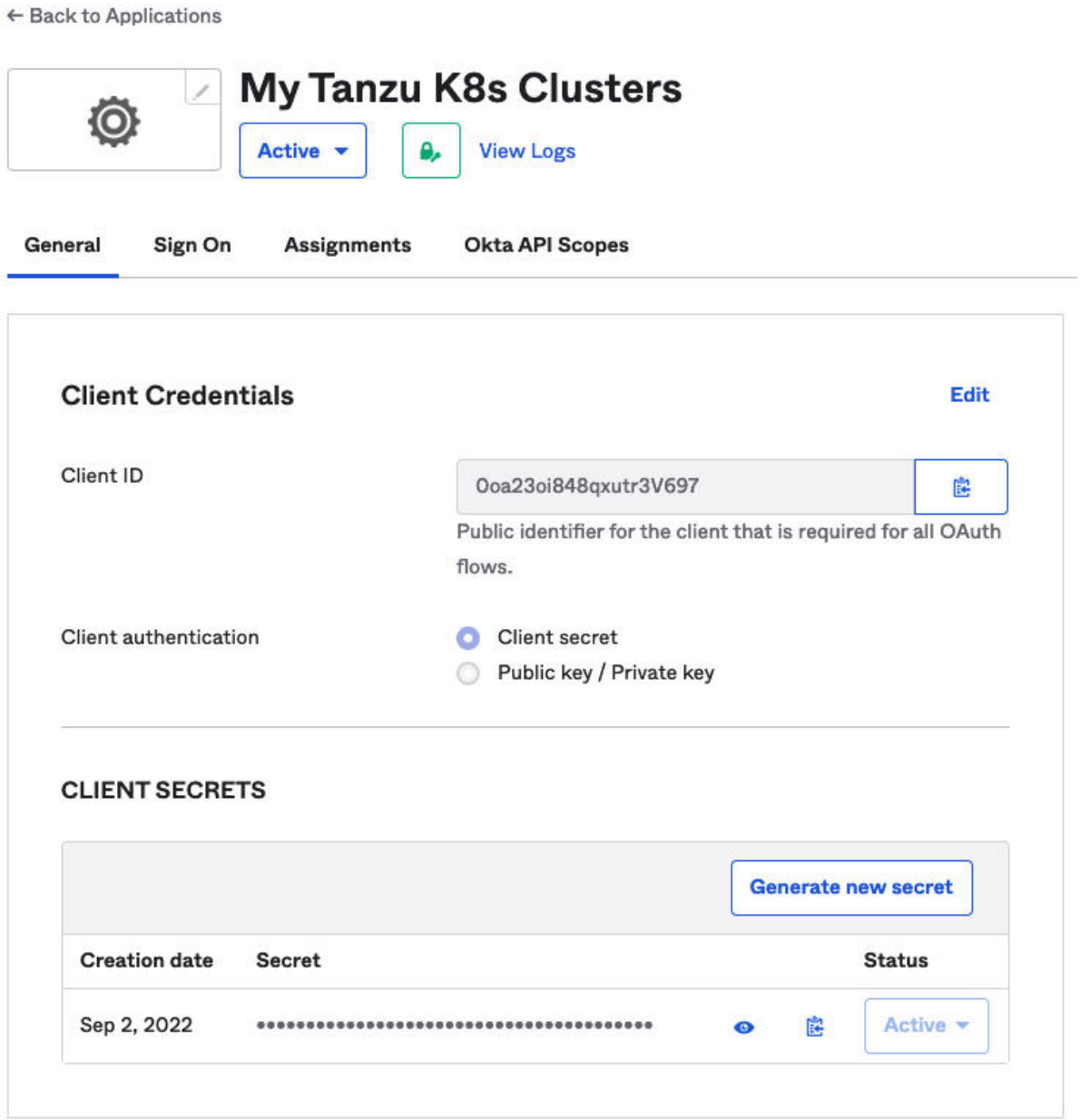
그림 4-8. Okta 액세스 제어



8 **Save**(저장)를 클릭하고 반환되는 **Client ID**(클라이언트 ID) 및 **Client Secret**(클라이언트 암호)을 복사합니다.

OKTA 구성을 저장하면 관리 콘솔에 **클라이언트 ID** 및 **클라이언트 암호**가 제공됩니다. 외부 ID 제공자를 사용하여 감독자를 구성하는 데 필요하므로 두 데이터를 모두 복사합니다.

그림 4-9. OIDC 클라이언트 ID 및 암호

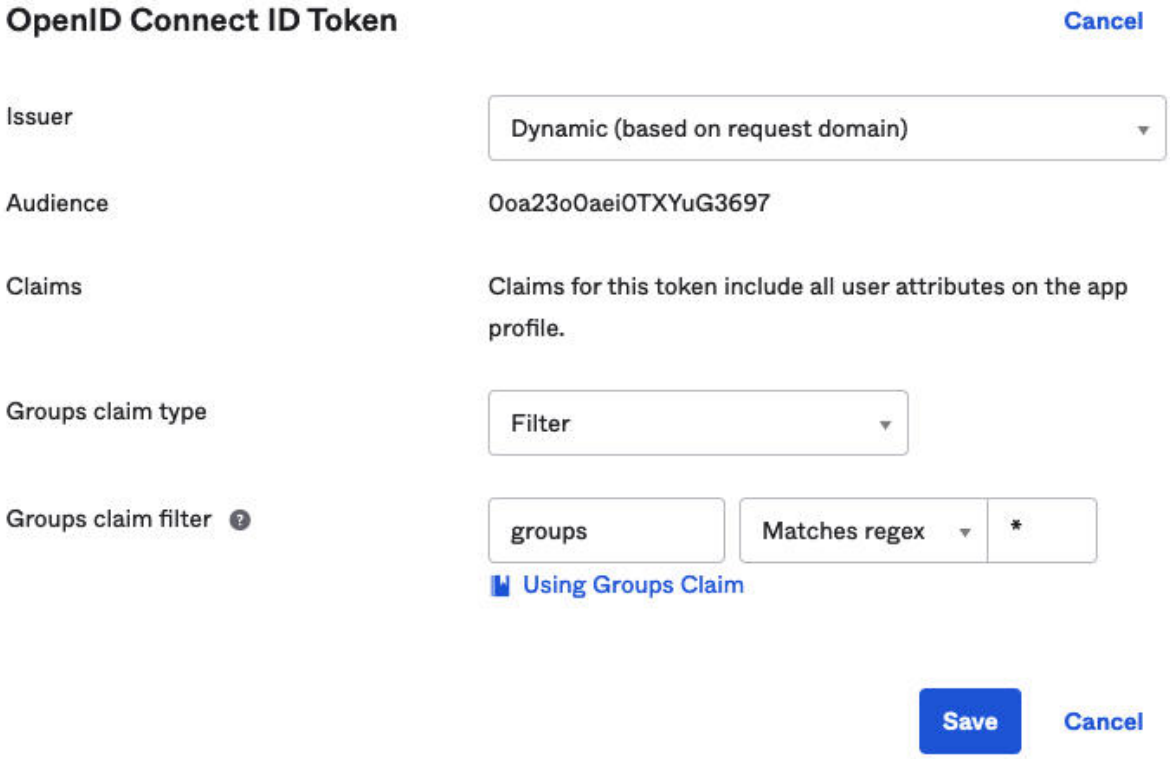


9 OpenID Connect ID 토큰을 구성합니다.

Sign On(로그온) 탭을 클릭합니다. **OpenID Connect ID Token**(OpenID Connect ID 토큰) 섹션에서 Edit(편집) 링크를 클릭하고 **Groups claim type**(그룹 할당 유형) 필터를 입력한 후 **Save**(저장)를 클릭하여 설정을 저장합니다.

예를 들어 클레임 이름 "groups"가 모든 그룹과 일치하도록 하려면 **groups** > **Matches regex(정규식과 일치)** > *를 선택합니다.

그림 4-10. OpenID Connect ID 토큰

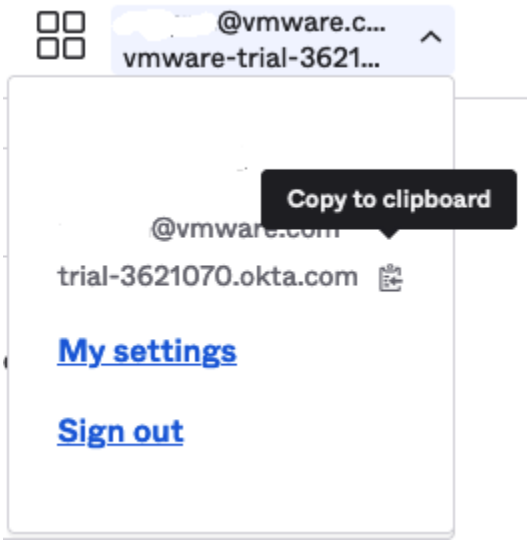


10 **Issuer URL**(발급자 URL)을 복사합니다.

감독자를 구성하려면 **Client ID**(클라이언트 ID) 및 **Client Secret**(클라이언트 암호) 외에 **Issuer URL**(발급자 URL)이 필요합니다.

Okta 관리 콘솔에서 **Issuer URL**(발급자 URL)을 복사합니다.

그림 4-11. Okta 발급자 URL



외부 IDP를 감독자에 등록

Tanzu CLI를 사용하여 감독자의 Tanzu Kubernetes Grid 2.0 클러스터에 연결하려면 OIDC 제공자를 감독자에 등록합니다.

사전 요구 사항

외부 ODIC 제공자를 감독자에 등록하기 전에 다음 사전 요구 사항을 완료하십시오.

- 워크로드 관리를 사용하도록 설정하고 감독자 인스턴스를 배포합니다. 감독자에서 TKG 2.0 클러스터 실행을 참조하십시오.
- 감독자 콜백 URL을 사용하여 외부 OpenID Connect ID 제공자를 구성합니다. TKG 서비스 클러스터와 함께 사용할 외부 IDP 구성의 내용을 참조하십시오.
- 외부 IDP에서 클라이언트 ID, 클라이언트 암호 및 발급자 URL을 가져옵니다. TKG 서비스 클러스터와 함께 사용할 외부 IDP 구성의 내용을 참조하십시오.

외부 IDP를 감독자에 등록

감독자는 Pinniped 감독자 및 Pinniped Concierge 구성 요소를 포드로 실행합니다. Tanzu Kubernetes Grid 클러스터는 Pinniped Concierge 구성 요소만 포드로 실행합니다. 이러한 구성 요소와 구성 요소가 상호 작용하는 방식에 대한 자세한 내용은 Pinniped 인증 서비스 설명서를 참조하십시오.

외부 ID 제공자를 감독자에 등록하면 감독자의 Pinniped 감독자 및 Pinniped Concierge 포드와 Tanzu Kubernetes Grid 클러스터의 Pinniped Concierge 포드가 시스템에서 업데이트됩니다. 해당 감독자 인스턴스에서 실행되는 모든 Tanzu Kubernetes Grid 클러스터는 동일한 외부 ID 제공자를 사용하여 자동으로 구성됩니다.

외부 ODIC 제공자를 감독자에 등록하려면 다음 절차를 완료하십시오.

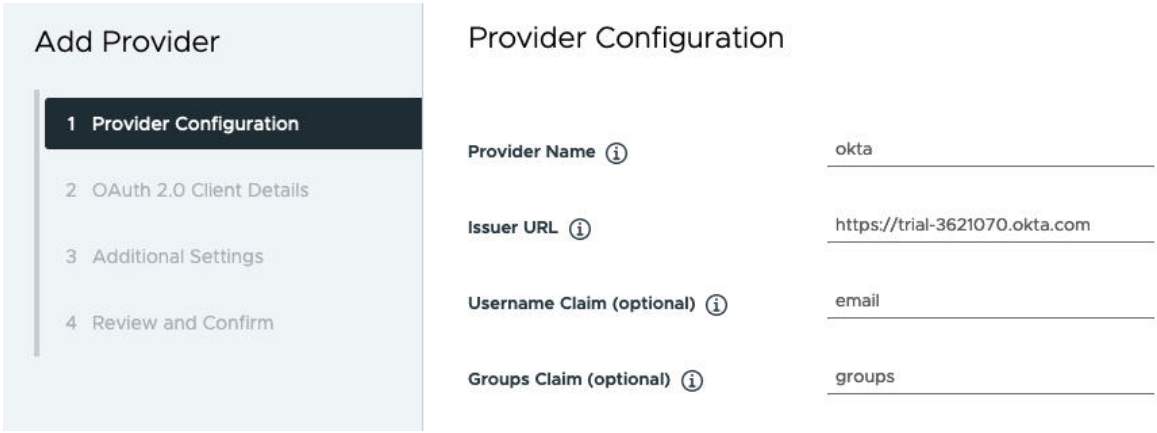
1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리 > 감독자 > 구성 > ID 제공자**를 선택합니다.

3 더하기 기호를 클릭하여 등록 프로세스를 시작합니다.

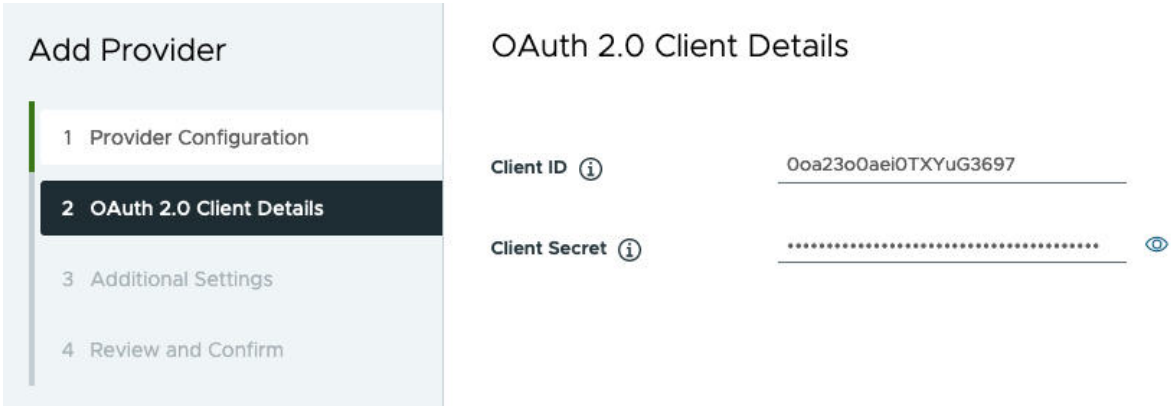
4 제공자를 구성합니다. OIDC 제공자 구성의 내용을 참조하십시오.

그림 4-12. OIDC 제공자 구성



5 OAuth 2.0 클라이언트 세부 정보를 구성합니다. OAuth 2.0 클라이언트 세부 정보의 내용을 참조하십시오.

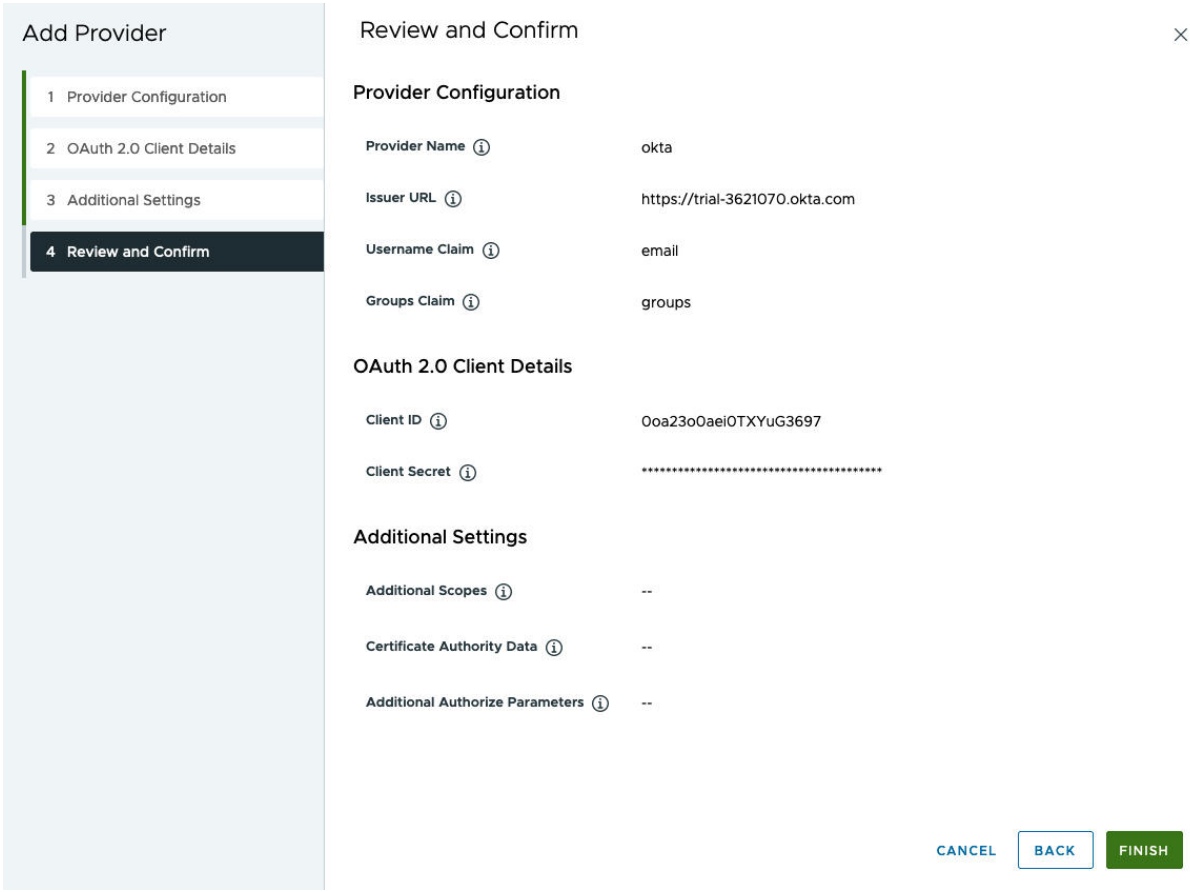
그림 4-13. OAuth 2.0 클라이언트 세부 정보



6 추가 설정을 구성합니다. 추가 설정의 내용을 참조하십시오.

7 제공자 설정을 확인합니다.

그림 4-14. 제공자 설정 확인



8 **마침**을 클릭하여 OIDC 제공자 등록을 완료합니다.

OIDC 제공자 구성

외부 OIDC 제공자를 감독자에 등록할 때 다음 제공자 구성 세부 정보를 참조하십시오.

표 4-3. OIDC 제공자 구성

필드	중요도	설명
제공자 이름	필수	외부 ID 제공자에 대한 사용자 정의 이름입니다.
발급자 URL	필수	토큰을 발급하는 ID 제공자의 URL입니다. OIDC 검색 URL은 발급자 URL에서 파생됩니다. 예를 들어 Okta 발급자 URL은 다음과 같은 모양이며 관리 콘솔에서 구할 수 있습니다. "https://trial-4359939-admin.okta.com" .

표 4-3. OIDC 제공자 구성 (계속)

필드	중요도	설명
사용자 이름 할당	선택 사항	지정된 사용자의 사용자 이름을 가져오기 위해 검사할 업스트림 ID 제공자 ID 토큰 또는 사용자 정보 끝점의 할당입니다. 이 필드를 비워두면 업스트림 발급자 URL이 "sub" 할당과 연결되어 Kubernetes에서 사용할 사용자 이름이 생성됩니다. 이 필드는 Pinniped가 인증을 결정하기 위해 업스트림 ID 토큰에서 확인해야 하는 사항을 지정합니다. 제공하지 않으면 사용자 ID는 "https://IDP-ISSUER?sub=UUID" 형식으로 지정됩니다.
그룹 할당	선택 사항	지정된 사용자의 그룹을 가져오기 위해 검사할 업스트림 ID 제공자 ID 토큰 또는 사용자 정보 끝점의 할당입니다. 이 필드를 비워두면 업스트림 ID 제공자의 그룹이 사용되지 않습니다. 그룹 클레임 필드는 Pinniped가 사용자 ID를 인증하기 위해 업스트림 ID 토큰에서 확인해야 하는 사항을 지정합니다.

OAuth 2.0 클라이언트 세부 정보

외부 OIDC 제공자를 감독자에 등록할 때 다음 제공자 OAuth 2.0 클라이언트 세부 정보를 참조하십시오.

표 4-4. OAuth 2.0 클라이언트 세부 정보

OAuth 2.0 클라이언트 세부 정보	중요도	설명
클라이언트 ID	필수	외부 IDP의 클라이언트 ID
클라이언트 암호	필수	외부 IDP의 클라이언트 암호

추가 설정

외부 OIDC 제공자를 감독자에 등록할 때 다음 추가 설정을 참조하십시오.

표 4-5. 추가 설정

설정	중요도	설명
추가 범위	선택 사항	토큰에서 요청할 추가 범위
CA(인증 기관) 데이터	선택 사항	보안 외부 IDP 연결을 위한 TLS CA(인증 기관) 데이터
추가 인증 매개 변수	선택 사항	OAuth2 인증 요청 중 추가 매개 변수

외부 ID 제공자 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성

OIDC 사용자에 대한 TKG 2.0 클러스터 액세스 권한을 구성하려면 외부 ID 제공자 사용자 및 그룹에 대한 역할 사용 권한으로 vSphere 네임스페이스를 구성합니다.

외부 ID 제공자 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성

감독자의 TKG 2.0 클러스터는 vSphere 네임스페이스에 프로비저닝됩니다. 외부 OIDC 제공자를 감독자에 등록한 후에는 외부 OIDC 제공자 사용자 및 그룹에 대한 역할 사용 권한으로 vSphere 네임스페이스를 구성합니다. 이 작업은 해당 vSphere 네임스페이스의 각 TKG 2.0 클러스터에 외부 OIDC 제공자에 대한 역할 바인딩을 생성합니다. vSphere 네임스페이스가 이미 있는 경우 역할 바인딩이 업데이트됩니다.

참고 감독자에 외부 IDP를 등록하면 해당 감독자에서 생성된 모든 TKG 2.0 클러스터가 Pinniped 구성 요소를 통해 외부 IDP에 자동으로 구성됩니다.

1 외부 ID 제공자를 감독자에 등록합니다.

외부 IDP를 감독자에 등록의 내용을 참조하십시오.

2 하나 이상의 TKG 클러스터에 대한 vSphere 네임스페이스를 생성하거나 기존 vSphere 네임스페이스를 선택합니다.

TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.

3 vSphere 네임스페이스에 대한 사용자 및 역할을 구성합니다.

외부 OIDC 제공자를 ID 소스로 선택하고 사용자를 추가하고 역할을 할당합니다.

a vSphere 네임스페이스를 선택합니다.

b **사용 권한 > 사용 권한 추가**를 선택합니다.

c **ID 소스**: 감독자에 등록된 외부 ID 제공자를 선택합니다.

외부 IDP를 등록하는 데 사용한 **제공자 이름**이 드롭다운 메뉴에 나타납니다. 그렇지 않은 경우 구성을 확인합니다.

d **사용자/그룹 검색**:사용자 또는 그룹 이름을 입력합니다. 텍스트 입력은 자유 형식 문자열입니다.

외부 ID 제공자의 사용자 및 그룹은 vCenter Server와 동기화되지 않으며 선택할 수 없습니다. 문자열 값(일반적으로 이메일 주소)을 입력해야 합니다. 접두사가 없으므로 예를 들어 "jdoe@acme.com"을 입력할 수 있습니다.

e **역할**: **볼 수 있음** 또는 **편집할 수 있음** 중에 역할을 선택하여 사용자 또는 그룹을 역할에 할당합니다.

참고 소유자 역할은 외부 ID 제공자와 함께 사용할 수 없습니다.

4 vSphere 네임스페이스 구성을 완료합니다.

TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

Tanzu CLI 및 외부 IDP를 사용하여 감독자에 연결

다음 단계에 따라 Tanzu CLI를 사용하여 감독자에 연결합니다.

사전 요구 사항

다음 사전 요구 사항을 완료하십시오.

1 OIDC를 준수하는 외부 ID 제공자를 감독자에 등록합니다. 외부 IDP를 감독자에 등록의 내용을 참조하십시오.

2 OIDC 사용자 및 그룹에게 vSphere 네임스페이스에 대한 액세스 권한을 부여합니다. 외부 ID 제공자 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성의 내용을 참조하십시오.

Tanzu CLI를 사용하여 감독자에 연결

다음 단계를 완료하십시오.

참고 TKG 서비스 3.1 이상을 사용하는 경우 동일한 버전의 TKG 서비스가 포함된 pinniped-auth cli 플러그인을 다운로드하고 최신 버전의 imgpkg cli 플러그인을 다운로드합니다. 자세한 내용은 Tanzu CLI 제품 설명서를 참조하십시오.

1 Tanzu CLI를 설치하고 초기화합니다. TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치의 내용을 참조하십시오.

2 다음 명령을 실행하여 감독자에 연결합니다.

```
tanzu context create context_name --endpoint https://10.73.27.32
```

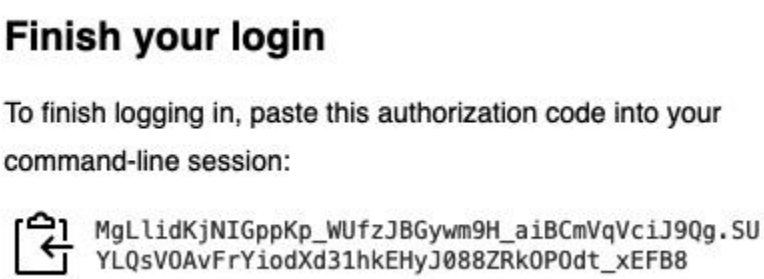
형식 설명:

- `context_name` 값은 액세스 권한이 부여된 OIDC의 이름입니다.
- `--endpoint` 값은 감독자 제어부 IP 주소입니다.

참고 문제를 해결하려면 명령에 `--stderr-only`를 추가합니다(예: `tanzu login --endpoint https://IP --name USER --stderr-only`).

3 챌린지가 발급되면 브라우저를 사용하여 링크를 방문합니다.

그림 4-15. Tanzu CLI 로그인



4 인증 코드를 복사한 후 Tanzu CLI에 붙여넣습니다.

```
tanzu context create context_name --endpoint https://10.73.27.32

Detected a vSphere Supervisor being used
Log in by visiting this link:
...
https://10.27.62.33/wcp/pinniped/oauth2/authorize?..
...
Optionally, paste your authorization code:
G2TcS145Q4e6A1YKf743n3BJlfQAQ_UdjXy38TtEEIo.ju4QV3PTsUvOigVUtQllZ7AJFU0YnjuLHTRVoNxvdZc
...
✔ successfully logged in to management cluster using the kubeconfig oidc-user
Checking for required plugins...
All required plugins are already installed and up-to-date
```

5 인증되면 Tanzu CLI를 사용하여 액세스 권한이 있는 대상 vSphere 네임스페이스에 TKG 클러스터를 프로비저닝할 수 있습니다. Tanzu CLI를 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

Tanzu CLI를 사용하여 TKG 클러스터에 OIDC 사용자로 연결

Tanzu CLI를 사용하여 TKG 클러스터에 연결하고 ODIC 제공자를 사용하여 인증합니다.

사전 요구 사항

이러한 지침에서는 감독자가 지원되는 외부 IDP(ID 제공자)로 구성되었고, 사용자(DevOps 사용자)가 Tanzu CLI를 사용하여 감독자에 연결했으며, TKG 클러스터를 프로비저닝했다고 가정합니다. 필요에 따라 다음 항목을 참조하십시오.

- 외부 ID 제공자를 사용하여 감독자의 TKG 클러스터에 연결
- 장 7 TKG 서비스 클러스터 프로비저닝

DevOps 사용자 워크플로

대상 vSphere 네임스페이스에 대한 편집 사용 권한이 있는 DevOps 사용자는 Tanzu CLI를 사용하여 공유 가능한 `kubeconfig` 파일을 생성한 다음 TKG 클러스터 사용자에게 배포합니다. Kubernetes에서 구성 컨텍스트에는 클러스터, 네임스페이스 및 사용자가 포함됩니다. `.kube/config` 파일에서 클러스터 컨텍스트를 볼 수 있습니다. 이 파일은 일반적으로 `kubeconfig` 파일이라고 합니다.

참고 이러한 단계는 대상 vSphere 네임스페이스에 대한 편집 사용 권한이 있는 DevOps 사용자가 수행해야 합니다.

1 Tanzu CLI 컨텍스트가 감독자로 설정되었는지 확인합니다.

Tanzu CLI 및 외부 IDP를 사용하여 감독자에 연결의 내용을 참조하십시오.

2 vSphere 관리자가 대상 vSphere 네임스페이스에 대한 사용자 사용 권한을 구성했는지 확인합니다.

외부 OIDC 사용자 및 그룹은 vSphere 네임스페이스 역할에 직접 매핑됩니다. 공유 가능한 kubeconfig를 생성하기 전에 클러스터 사용자를 vSphere 네임스페이스에 추가해야 합니다.

외부 ID 제공자 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성의 내용을 참조하십시오.

3 대상 vSphere 네임스페이스에 프로비저닝된 TKG 클러스터를 나열합니다.

```
tanzu cluster list --namespace VSPHERE-NAMESPACE
```

4 대상 TKG 클러스터에 대해 공유 가능한 kubeconfig 파일을 생성합니다.

```
tanzu cluster kubeconfig get CLUSTER-NAME --namespace=NAMESPACE
```

5 공유 kubeconfig 파일을 클러스터 사용자에게 배포하여 TKG 클러스터에 로그인할 수 있도록 합니다.

클러스터 사용자 워크플로

클러스터 사용자로 TKG 클러스터에 로그인하려면 다음 단계를 완료합니다.

1 DevOps 사용자로부터 kubeconfig 파일을 가져옵니다.

2 kubeconfig 파일 및 kubectl을 사용하여 TKGS 클러스터에 로그인합니다.

```
kubectl --kubeconfig
```

3 브라우저 인증 프로세스를 완료합니다.

a 챌린지가 발급되면 브라우저를 사용하여 링크를 방문합니다.

b 인증 코드를 복사한 후 CLI에 붙여넣습니다.

4 `kubectl`을 사용하여 클러스터와 상호 작용합니다.

Kubernetes 관리자 및 시스템 사용자로 TKG 서비스 클러스터에 연결

TKG 서비스 클러스터에 Kubernetes 관리자 및 시스템 사용자로 연결하여 TKG 서비스 클러스터 관리 및 문제 해결을 지원할 수 있습니다.

Kubernetes 관리자로 TKG 서비스 클러스터 제어부에 연결

`kubernetes-admin` 사용자로 TKG 서비스 클러스터 제어부에 연결하여 관리 작업을 수행하고 클러스터 문제를 해결할 수 있습니다.

프로비저닝된 Tanzu Kubernetes 클러스터에 대한 유효한 `kubeconfig` 파일은 감독자에서 *`TKG-CLUSTER-NAME`*`-kubeconfig`이라는 이름의 암호 개체로 제공됩니다. 이 암호를 사용하여 `kubernetes-admin` 사용자로 클러스터 제어부에 연결할 수 있습니다.

절차

1 감독자에 연결합니다.

2 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

3 네임스페이스에서 암호 개체를 봅니다.

```
kubectl get secrets
```

암호 이름은 *TKG-CLUSTER-NAME*-kubeconfig입니다.

```
kubectl config use-context tkg-cluster-ns
Switched to context "tkg-cluster-ns".
ubuntu@ubuntu:~$ kubectl get secrets
NAME                                  TYPE                                  DATA   AGE
...
tkg-cluster-1-kubeconfig              Opaque                                1      23h
...
```

4 다음 명령을 실행하여 암호를 디코딩합니다.

암호는 Base64로 인코딩됩니다. 디코딩하려면 Linux에서는 `base64 --decode`(또는 `base64 -d`)를 사용하고, MacOS에서는 `base64 --Decode`(또는 `base64 -D`)를 사용하고, Windows에서는 온라인 도구를 사용합니다.

```
kubectl get secret TKG-CLUSTER-NAME-kubeconfig -o jsonpath='{.data.value}' | base64 -d > tkgs-cluster-kubeconfig-admin
```

이 명령은 암호를 디코딩하고 `tkgs-cluster-kubeconfig-admin`라는 이름의 로컬 파일에 씁니다. `cat` 명령을 사용하여 파일 컨텐츠를 확인할 수 있습니다.

5 디코딩된 `tkg-cluster-kubeconfig-admin` 파일을 사용하여 TKG 클러스터에 Kubernetes 관리자로 연결합니다.

이 작업을 수행하는 데에는 두 가지 옵션이 있습니다.

옵션	설명
--kubeconfig <path\to\kubeconfig>	`--kubeconfig` 플래그와 로컬 kubeconfig 파일에 대한 경로를 사용합니다. 예를 들어 kubeconfig 파일이 위치한 디렉토리가 명령을 실행하는 디렉토리와 동일하다고 가정하면 다음과 같습니다. `kubectl --kubeconfig tkg-cluster-kubeconfig-admin get nodes`
KUBECONFIG	디코딩된 kubeconfig 파일을 가리키도록 KUBECONFIG 환경 변수를 설정하고 kubectl을 실행합니다(예: `kubectl get nodes`).

클러스터에 노드가 표시됩니다.

6 vSphere 네임스페이스에 대한 편집 권한이 있는 DevOps 사용자가 Tanzu CLI를 사용하여 TKG 클러스터에 관리자로 로그인하려면 다음 명령을 실행합니다.

```
tanzu cluster kubeconfig get CLUSTER-NAME --admin
```

이 명령은 kubernetes-control-plane에 대한 인증서/개인 키가 포함된 `kubeconfig`를 생성합니다(모든 권한 부여를 우회함). 그런 다음 이 `kubeconfig`를 사용하여 클러스터에 로그인할 수 있습니다. #unique_36의 내용을 참조하십시오.

개인 키를 사용하여 시스템 사용자로 TKG 서비스 클러스터 노드에 SSH를 통해 연결

개인 키를 사용하여 `vmware-system-user`로 TKG 클러스터 노드에 SSH를 통해 연결할 수 있습니다.

SSH를 통해 `vmware-system-user` 사용자로 TKG 클러스터 노드에 연결할 수 있습니다. SSH 개인 키가 포함된 암호는 `CLUSTER-NAME-ssh`라고 이름이 지정됩니다. Kubectl을 사용하여 TKG 클러스터 암호 얻기의 내용을 참조하십시오.

개인 키를 사용하여 SSH를 통해 TKG 클러스터 노드에 연결하려면 감독자에서 점프 박스(jump box) vSphere 포드를 생성합니다.

사전 요구 사항

이 작업에서는 vSphere 포드를 SSH 연결을 위한 점프 호스트로 프로비저닝합니다. vSphere 포드에는 감독자용 NSX 네트워킹이 필요합니다. 감독자에 vDS 네트워킹을 사용하는 경우 암호를 사용하여 시스템 사용자로 TKG 서비스 클러스터 노드에 SSH를 통해 연결 항목을 참조하십시오.

절차

1 감독자에 연결합니다.

kubectl을 사용하여 vCenter Single Sign-On 사용자로 감독자에 연결의 내용을 참조하십시오.

2 **NAMESPACE**라는 환경 변수를 생성합니다. 이 값은 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스의 이름입니다.

```
export NAMESPACE=VSPHERE-NAMESPACE
```

3 Tanzu Kubernetes 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context $NAMESPACE
```

4 *TKG-CLUSTER-NAME*-ssh 암호 개체를 확인합니다.

```
kubectl get secrets
```

5 Docker Hub 레지스트리 자격 증명 암호를 생성합니다.

기본적으로 vSphere 포드(PhotonOS)를 생성하는 데 사용되는 이미지는 Docker Hub에서 끌어옵니다. 이 이미지를 끌어오려면 자격 증명 암호가 필요할 수 있습니다. 개인 레지스트리 자격 증명 암호 생성의 내용을 참조하십시오.

6 다음 `jumpbox.yaml`을 사용하여 vSphere 포드를 생성합니다.

`namespace` 값 `YOUR-NAMESPACE`를 대상 클러스터가 프로비저닝된 vSphere 네임스페이스로 바꿉니다.
`secretName` 값 `YOUR-CLUSTER-NAME`-ssh를 대상 클러스터의 이름으로 바꿉니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: jumpbox
  namespace: YOUR-NAMESPACE     #REPLACE
spec:
  containers:
  - image: "photon:3.0"
    name: jumpbox
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "yum install -y openssh-server; mkdir /root/.ssh; cp /root/ssh/ssh-privatekey /
root/.ssh/id_rsa; chmod 600 /root/.ssh/id_rsa; while true; do sleep 30; done;" ]
    volumeMounts:
      - mountPath: "/root/ssh"
        name: ssh-key
        readOnly: true
    resources:
      requests:
        memory: 2Gi
  volumes:
    - name: ssh-key
      secret:
        secretName: YOUR-CLUSTER-NAME-ssh     #REPLACE
  imagePullSecrets:
    - name: regcred
```

7 `jumpbox.yaml` 규격을 적용하여 포드를 배포합니다.

```
kubectl apply -f jumpbox.yaml
```

```
pod/jumpbox created
```

8 포드가 실행 중인지 확인합니다.

```
kubectl get pods
```

```
NAME      READY   STATUS    RESTARTS   AGE
jumpbox   1/1     Running   0          3h9m
```

참고 vSphere 네임스페이스의 vCenter에도 jumpbox 포드가 보여야 합니다.

9 다음 명령 집합을 실행하여 대상 클러스터 노드의 IP 주소를 사용하여 환경 변수를 생성합니다.

a 대상 가상 시스템의 이름을 가져옵니다.

```
kubectl get virtualmachines
```

b 값이 대상 노드의 이름인 환경 변수 `VMNAME`을 생성합니다.

```
export VMNAME=NAME-OF-THE-VIRTUAL-MACHINE
```

c 값이 대상 노드 VM의 IP 주소인 환경 변수 `VMIP`를 생성합니다.

```
export VMIP=$(kubectl -n $NAMESPACE get virtualmachine/$VMNAME -o
jsonpath='{.status.vmIp}')
```

10 다음 명령을 실행하여 점프 박스(jump box) 포드를 사용하여 SSH를 통해 클러스터 노드에 연결합니다.

```
kubectl exec -it jumpbox  /usr/bin/ssh vmware-system-user@$VMIP
```

중요 컨테이너를 생성하고 소프트웨어를 설치하는 데 약 60초가 걸립니다. "error executing command in container: container_linux.go:370: starting container process caused: exec: "/usr/bin/ssh": stat /usr/bin/ssh: no such file or directory"가 표시되면 몇 초 후에 명령을 다시 시도합니다.

11 **yes**를 입력하여 호스트의 신뢰성을 확인합니다.

```
The authenticity of host '10.249.0.999 (10.249.0.999)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.249.0.999' (ECDSA) to the list of known hosts.
Welcome to Photon 3.0
```

12 대상 노드에 `vmware-system-user`로 로그인되어 있는지 확인합니다.

예를 들어, 다음 출력은 사용자가 제어부 노드에 시스템 사용자로 로그인했음을 나타냅니다.

```
vmware-system-user@tkg-cluster-1-control-plane-66tbr [ ~ ]$
```

13 노드에서 원하는 작업을 수행합니다.

주의 노드에서 kubelet 다시 시작과 같은 특정 작업을 수행하려면 `sudo` 또는 `sudo su`를 사용해야 할 수 있습니다.

14 완료되면 **exit**를 입력하여 vSphere 포드의 SSH 세션에서 로그아웃합니다.

15 포드를 삭제하려면 `kubectl delete pod jumpbox` 명령을 실행합니다.

경고 보안을 위해 작업을 완료한 후에는 jumpbox 포드를 삭제하는 것이 좋습니다. 필요하면 나중에 다시 생성할 수 있습니다.

암호를 사용하여 시스템 사용자로 TKG 서비스 클러스터 노드에 SSH를 통해 연결

암호를 사용하여 `vmware-system-user`로 워크로드 클러스터 노드에 SSH를 통해 연결할 수 있습니다.

암호를 사용하여 클러스터 노드에 `vmware-system-user` 사용자로 연결할 수 있습니다. 암호는 `CLUSTER-NAME-ssh-password`라는 이름의 암호로 저장됩니다. 암호는 `.data.ssh-passwordkey`에 Base64로 인코딩됩니다. SSH 세션을 통해 암호를 제공할 수 있습니다. Kubectl을 사용하여 TKG 클러스터 암호 얻기의 내용을 참조하십시오.

사전 요구 사항

SSH 연결을 적절한 워크로드 네트워크로 라우팅하려면 **워크로드 관리**를 사용하도록 설정된 vSphere 환경에서 Linux 점프 호스트 VM을 배포합니다. Linux 점프 호스트 VM 생성의 내용을 참조하십시오.

참고 vDS 네트워킹을 사용 중이고 SSH를 사용하여 클러스터 노드에 연결하려는 경우 점프 호스트 VM을 배포하는 것은 어려운 요구 사항입니다. 개인 키 대신 암호를 사용하여 SSH를 통해 연결하려는 경우에는 NSX 네트워킹에서도 이 방식을 사용할 수 있습니다.

절차

1 점프 호스트 VM의 IP 주소, 사용자 이름 및 암호를 가져옵니다.

Linux 점프 호스트 VM 생성의 내용을 참조하십시오.

2 감독자에 연결합니다.

kubectl을 사용하여 vCenter Single Sign-On 사용자로 감독자에 연결의 내용을 참조하십시오.

3 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

4 대상 클러스터 노드의 IP 주소를 가져옵니다.

노드를 나열합니다.

```
kubectl get virtualmachines
```

노드를 설명하여 대상 노드의 IP 주소를 가져옵니다.

```
kubectl describe virtualmachines
```

5 `TKG-CLUSTER-NAME-ssh-password` 암호를 봅니다.

```
kubectl get secrets
```

6 대상 클러스터에 대한 ssh-passwordkey를 가져옵니다.

```
kubectl get secrets TKG-CLUSTER-NAME-ssh-password -o yaml
```

ssh-passwordkey가 반환됩니다. 예:

```
apiVersion: v1
data:
  ssh-passwordkey: RU1pQllLTC9TRjVFV0RBcCtmd1zwOTROeURYSWNGeXNReXJhaXRBUllYaz0=
```

7 ssh-passwordkey를 디코딩합니다.

암호는 Base64로 인코딩됩니다. 디코딩하려면 Linux에서는 `base64 --decode`(또는 `base64 -d`)를 사용하고, MacOS에서는 `base64 --Decode`(또는 `base64 -D`)를 사용하고, Windows에서는 온라인 도구를 사용합니다.

```
echo <ssh-passwordkey> | base64 --decode
```

8 `vmware-system-user`로 대상 클러스터 노드에 SSH를 통해 연결합니다.

```
ssh vmware-system-user@TKG-CLUSTER-NODE-IP-ADDRESS
```

9 디코딩한 암호를 사용하여 로그인합니다.

Linux 점프 호스트 VM 생성

암호를 사용하여 워크로드 클러스터 노드에 SSH로 연결하려면 워크로드 네트워크에 연결하는 점프 박스(jump box) VM과 SSH 터널링을 위한 관리 또는 프런트 엔드 네트워크를 생성합니다.

Linux 점프 호스트 VM 생성

다음 단계에 따라 Linux 점프 박스(jump-box) VM을 생성합니다. 이것을 수행하는 방법은 여러 가지이며, 다음 방법은 그 중 한 가지입니다. 이 지침에서는 PhotonOS를 사용하며, https://github.com/vmware/photon/wiki/Downloading-Photon-OS에서 다운로드할 수 있습니다.

참고 점프 호스트를 생성하는 이 방법은 vDS 네트워킹 환경을 위한 방법입니다. NSX를 사용하는 경우 vSphere 포드를 사용하여 점프 호스트를 생성합니다. 개인 키를 사용하여 시스템 사용자로 TKG 서비스 클러스터 노드에 SSH를 통해 연결의 내용을 참조하십시오.

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 새 가상 시스템을 생성합니다.

3 Linux 게스트 운영 체제(이 예에서는 VMware Photon OS(64비트))를 선택합니다.

4 운영 체제를 설치합니다. 이렇게 하려면 ISO를 다운로드하여 VM에 연결하고 부팅합니다.

5 [워크로드 네트워크] > [네임스페이스 네트워크]에서 IP 주소를 사용하여 VM을 구성합니다.

참고 워크로드 네트워크 범위가 사용 중인 포트 그룹의 전체 네트워크 공간을 사용하는 경우 IP 충돌을 일으킬 수 있습니다.

6 두 번째 가상 NIC를 VM에 추가하고 관리 또는 프런트 엔드 네트워크에 할당합니다.

7 운영 체제 구성을 완료하고 재부팅 후 VM의 전원을 켭니다.

8 VM의 vSphere 콘솔에 루트 사용자로 로그인합니다.

9 새 NIC에 대한 네트워크 인터페이스를 생성하고 프런트 엔드 네트워크에 IP를 제공합니다.

```
ifconfig eth1 IP-ADDRESS netmask NETMASK up
```

참고 이 방법은 재부팅 시 지속되지 않습니다.

10 이 인터페이스를 통해 게이트웨이 및 DNS 서버를 ping할 수 있는지 확인합니다.

11 VM의 vSphere 콘솔에서 인증서를 사용하여 SSH 사용자를 설정합니다. 중첩된 셸을 생성하여 작동하는지 확인합니다.

12 SSH를 사용하여 프런트 엔드 네트워크에서 Jumpbox에 SSH 사용자로 로그인하여 작동하는지 확인합니다.

13 sshpass를 VM에 설치합니다. (그러면 SSH를 통해 암호를 사용하여 로그인할 수 있습니다.) PhotonOS의 경우 명령은 다음과 같습니다.

```
tdnf install -y sshpass
```

14 클라이언트의 공용 키를 `~/.ssh/authorized_keys` 파일에 추가하고 `sshd` 프로세스를 다시 시작하면 SSH가 암호 없이 작동할 수 있습니다.

- 공용 키를 가져옵니다(예: `cat ~/.ssh/id_rsa.pub`).
- 점프 호스트 VM에 액세스합니다.
- SSH 디렉토리(없는 경우)를 생성합니다. `mkdir -p ~/.ssh`.
- 공용 키를 `authorized_keys` 파일에 추가합니다. `echo ssh-rsa AAAA.... >> ~/.ssh/authorized_keys`. `cat ~/.ssh/id_rsa.pub` 명령에서 출력한 전체 공용 키 문자열로 `ssh-rsa AAAA....`를 대체합니다.
- `~/.ssh` 디렉토리 및 `authorized_keys` 파일에 적절한 사용 권한이 설정되어 있는지 확인합니다(예: `chmod -R go= ~/.ssh`).

플랫폼 운영자를 위한 전용 그룹 및 역할 생성

vSphere IaaS control plane 운영자(감독자 및 TKG 서비스 클러스터 운영 책임자 포함)에게 권한을 할당하는 선택적 방법은 해당 운영자를 위한 전용 vSphere 사용자 그룹 및 역할을 생성하는 것입니다.

플랫폼 운영자 그룹 및 역할 정보

보안 모범 사례로, 플랫폼 운영자에게 vSphere 관리자 역할을 할당하지 않는 것이 좋습니다. 이 역할은 TKG 서비스 클러스터를 운영하는 데 필요한 것보다 더 많은 권한을 부여하기 때문입니다. 최소 권한의 원칙에 따라 플랫폼 운영자가 사용할 전용 사용자 그룹, 서비스(사용자) 계정 및 사용자 지정 역할을 생성한 다음, 사용자 그룹에 vSphere 개체에 대한 사용자 지정 역할 권한을 부여해야 합니다.

참고 이 항목의 모든 작업을 수행하려면 vCenter Server에 vSphere 관리자로 로그인해야 합니다.

참고 vSphere 그룹 및 역할 이름은 사용자 정의 문자열입니다. 여기에 제공된 이름은 예시이며, 보안 및 비즈니스 요구 사항에 따라 채택, 조정 또는 변경할 수 있습니다.

경고 보안 및 비즈니스 요구 사항의 컨텍스트에서 여기에 제공된 역할 권한 예시를 평가하고, 역할을 테스트하여 규정 준수 여부를 확인하고, 그에 따라 조정해야 합니다. 여기에 사용된 일부 사용 권한은 사용자의 요구에 적합하지 않을 수 있으며 추가 사용 권한이 필요할 수 있습니다. vSphere 사용 권한 및 보안 고려 사항의 전체 목록은 vSphere 보안 설명서를 참조하십시오.

1부: 플랫폼 운영자 그룹 및 사용자 생성

vCenter 또는 vCenter와 통합된 AD/LDAP 시스템에서 플랫폼 운영자 그룹 및 초기 사용자 계정을 생성합니다.

1 vSphere Client를 사용하여 vCenter Server에 관리자로 로그인합니다.

2 **관리 > Single Sign-On > 사용자 및 그룹**으로 이동합니다.

3 **그룹** 탭을 선택합니다.

4 **추가**를 클릭하고 새 그룹을 생성합니다.

- 이름: *platform-operators-group*
- 설명: *감독자 및 TKG 서비스 클러스터에 대한 Kubernetes 운영자용 그룹 계정*
- **추가**를 클릭합니다.

5 **사용자** 탭을 선택합니다.

6 **추가**를 클릭하고 테스트 용도의 새 사용자를 생성합니다.

- 이름: *platform-operator-00*
- 암호: 요구 사항을 충족하는 강력한 암호를 입력합니다.
- **추가**를 클릭합니다.

7 **그룹** 탭을 선택합니다.

8 새 사용자를 그룹에 추가합니다.

- *platform-operators-group* 그룹을 선택합니다.
- **멤버 추가**를 클릭합니다.

- **vsphere.local**을 선택합니다.
- 사용자 이름 *platform-operator-00*을 검색합니다.
- 이 사용자를 선택하고 **추가**를 클릭합니다.
- **저장**을 클릭합니다.

2부: 플랫폼 운영자 그룹을 서비스 제공자 사용자 그룹에 추가

플랫폼 운영자 그룹을 서비스 제공자 사용자 그룹에 추가합니다. 이렇게 하면 플랫폼 운영자 그룹의 멤버가 vCenter **인벤토리 > 호스트 및 클러스터** 화면에서 vSphere 네임스페이스를 볼 수 있습니다.

1 **관리 > Single Sign-On > 사용자 및 그룹**으로 이동합니다.

2 **그룹** 탭을 선택합니다.

3 **ServiceProviderUsers** 그룹을 찾습니다.

4 **ServiceProviderUsers** 그룹을 편집하고 **platform-operators-group** 멤버로 추가합니다.

5 **저장**을 클릭합니다.

3부: 플랫폼 운영자 역할 생성

플랫폼 운영자를 위한 사용자 지정 vCenter SSO 역할을 생성합니다.

참고 이 역할에는 컨텐츠 라이브러리 관리를 포함하여 감독자 및 TKG 서비스 클러스터를 프로비저닝하고 운영하는 데 필요한 모든 사용 권한이 포함됩니다. 비즈니스 및 보안 요구 사항에 따라 이 역할에 할당된 사용 권한을 조정해야 할 수도 있습니다. 또한 역할을 테스트하여 요구 사항을 충족하는지 확인해야 합니다.

1 vSphere Client를 사용하여 **관리 > 액세스 제어 > 역할**로 이동합니다.

2 **새로 만들기**를 선택하고 새 역할 이름 **`platform-operators-role`**을 생성합니다.

3 이 역할에 대해 다음 권한을 정의합니다.

4 완료되면 **저장**을 클릭합니다.

```
- Alarms
  - Acknowledge alarm &
  - Create alarm &
  - Disable alarm action on entity &
  - Modify alarm &
  - Remove alarm &
  - Set alarm status &
- Certificate Authority
  - Create/Delete (below Admins priv) &
- Certificate Management
  - Create/Delete (below Admins priv) &
- Cns
  - Searchable * &
- Compute Policy
  - Create and Delete Compute Policy &
```

```
- Content Library
  - Add library item &
  - Check in a template &
  - Check out a template &
  - Create local library &
  - Create subscribed library &
  - Delete library item &
  - Delete local library &
  - Delete subscribed library &
  - Download files &
  - Evict library item &
  - Evict subscribed library &
  - Import storage &
  - Probe subscription information &
  - Read storage &
  - Sync library item &
  - Sync subscribed library &
  - Type introspection &
  - Update configuration settings &
  - Update library &
  - Update library item &
  - Update local library &
  - Update subscribed library &
  - View configuration settings &
- Datastore
  - Allocate space * & $
  - Browse datastore * &
  - Configure datastore &
  - Low level file operations * &
  - Remove file &
  - Rename datastore &
  - Update virtual machine files &
  - Update virtual machine metadata &
- Extension
  - Register extension &
  - Unregister extension &
  - Update extension &
- Folder
  - Create folder &
  - Delete folder &
  - Move folder &
  - Rename folder &
- Global
  - Cancel task &
  - Disable methods * &
  - Enable methods * &
  - Global tag &
  - Health &
  - Licenses * &
  - Log event &
  - Manage custom attributes &
  - Service managers &
  - Set custom attribute &
  - System tag &
- Host
```

```
  - Configuration
    - Network configuration $
- Host profile
  - View &
- Hybrid Linked Mode
  - Manage &
- Namespaces
  - Modify cluster-wide configuration
  - Modify cluster-wide namespace self-service configuration
  - Modify namespace configuration
- Network
  - Assign network * & $
- Resource
  - Apply recommendation &
  - Assign vApp to resource pool * &
  - Assign virtual machine to resource pool &
  - Create resource pool &
  - Modify resource pool &
  - Move resource pool &
  - Query vMotion &
  - Remove resource pool &
  - Rename resource pool &
- Scheduled task
  - Create tasks &
  - Modify task &
  - Remove task &
  - Run task &
- Sessions
  - Message * &
  - Validate session * &
- VM storage policies
  - View VM storage policies *
- Storage views
  - View &
- Supervisor Services
  - Manage Supervisor Services
- Trusted Infrastructure administrator
  - Manage Trusted Infrastructure Hosts &
- vApp
  - Add virtual machine &
  - Assign resource pool &
  - Assign vApp &
  - Clone &
  - Create &
  - Delete &
  - Export &
  - Import * $
  - Move &
  - Power off &
  - Power on &
  - Rename &
  - Suspend &
  - Unregister &
  - View OVF environment &
  - vApp application configuration &
```

```
  - vApp instance configuration &
  - vApp managedBy configuration &
  - vApp resource configuration &
- Virtual machine
  - Change Configuration
    - Acquire disk lease &
    - Add existing disk * & $
    - Add new disk * &
    - Add or remove device * &
    - Advanced configuration * & $
    - Change CPU count * &
    - Change Memory * &
    - Change Settings * &
    - Change Swapfile placement &
    - Change Resource &
    - Configure Host USB device &
    - Configure Raw device * &
    - Configure managedBy &
    - Display connection settings &
    - Extend virtual disk * &
    - Modify device settings * &
    - Query Fault Tolerance compatibility &
    - Query unowned files &
    - Reload from path &
    - Remove disk * &
    - Rename &
    - Reset guest information &
    - Set annotation &
    - Toggle disk change tracking * &
    - Upgrade virtual machine compatibility &
  - Edit Inventory
    - Create from existing * &
    - Create new &
    - Move &
    - Remove * &
    - Register &
    - Unregister &
  - Guest operations
    - Guest operation alias modification &
    - Guest operation alias query &
    - Guest operation modifications &
    - Guest operation program execution &
    - Guest operation queries &
  - Interaction
    - Answer question &
    - Backup operation on virtual machine &
    - Configure CD media &
    - Configure floppy media &
    - Connect devices &
    - Console interaction &
    - Create screenshot &
    - Defragment all disks &
    - Drag and drop &
    - Guest operating system management by VIX API &
    - Inject USB HID scan codes &
```

```
    - Install VMware Tools &
    - Pause or Unpause &
    - Power off * &
    - Power on * &
    - Reset &
    - Suspend &
  - Provisioning
    - Allow disk access &
    - Allow file access &
    - Allow read-only disk access * &
    - Allow virtual machine download * &
    - Allow virtual machine files upload &
    - Clone template &
    - Clone virtual machine &
    - Create template from virtual machine &
    - Customize guest &
    - Deploy template * &
    - Mark as template &
    - Mark as virtual machine &
    - Modify customization specification &
    - Promote disks &
    - Read customization specifications &
  - Service configuration
    - Allow notifications &
    - Allow polling of global event notifications &
    - Manage service configurations &
    - Modify service configuration &
    - Query service configurations &
    - Read service configuration &
  - Snapshot management
    - Create snapshot * &
    - Remove snapshot * &
    - Rename snapshot &
    - Revert to snapshot &
  - vSphere Replication
    - Configure replication &
    - Manage replication &
    - Monitor replication &
- Virtual Machine Classes
  - Manage Virtual Machine Classes
- vSan
  - Cluster &
    - ShallowRekey &
- vService
  - Create dependency &
  - Destroy dependency &
  - Reconfigure dependency configuration &
  - Update dependency &
- vSphere Tagging
  - Assign or Unassign vSphere Tag &
  - Assign or Unassign vSphere Tag on Object &
  - Create vSphere Tag &
  - Create vSphere Tag Category &
  - Delete vSphere Tag &
  - Delete vSphere Tag Category &
```

```
- Edit vSphere Tag &
- Edit vSphere Tag Category &
- Modify UsedBy Field For Category &
- Modify UsedBy Field For Tag &
```

4부: 플랫폼 운영자 그룹 및 역할에 vCenter 개체 사용 권한 할당

플랫폼 운영자 그룹에 감독자 및 TKG 서비스 클러스터가 사용하는 vCenter 개체에 대한 사용 권한을 할당합니다.

1 vCenter에서 **인벤토리** 보기를 선택합니다.

2 아래에 나열된 각 vCenter 개체의 경우 개체를 마우스 오른쪽 버튼으로 클릭하고 **사용 권한 추가**를 선택합니다.

3 사용자/그룹의 경우 **platform-operators-group** 그룹을 선택합니다.

4 역할의 경우 **platform-operators-group-role** 역할을 선택합니다.

5 일부 개체의 경우 표시된 대로 **하위 항목으로 전파**를 선택해야 합니다.

- **호스트 및 클러스터**
  - 루트 vCenter Server 개체입니다.
  - 데이터 센터 및 모든 호스트 및 클러스터 폴더(데이터 센터 개체부터 TKG 배포를 관리하는 클러스터까지).
  - 감독자를 사용하도록 설정되고 **하위 항목으로 전파**를 사용하도록 설정된 대상 vCenter 클러스터(ESXi 호스트 경우 등).
  - **하위 항목으로 전파**를 사용하도록 설정된 대상 리소스 풀.
- **VM 및 템플릿**
  - **하위 항목으로 전파**를 사용하도록 설정된 최상위 데이터 센터 개체.
  - 또는 세분성을 높이기 위해 **하위 항목으로 전파**를 사용하도록 설정된 대상 VM 및 템플릿 폴더.
- **스토리지**
  - **하위 항목으로 전파**를 사용하도록 설정된 최상위 데이터 센터 개체.
  - 또는 **하위 항목으로 전파**를 사용하도록 설정되지 않은 공유 데이터스토어 개체(예: vsanDatastore) 또는 **하위 항목으로 전파**를 사용하도록 설정된 개별 데이터스토어 및 모든 스토리지 폴더(데이터 센터 개체부터 TKG 배포에 사용될 데이터스토어까지).
- **네트워킹**
  - **하위 항목으로 전파**를 사용하도록 설정된 최상위 데이터 센터 개체.
  - 또는 클러스터가 할당될 개별 네트워크, 분산 스위치 및 분산 포트 그룹.

5부: 플랫폼 운영자 그룹 및 역할 연결

플랫폼 운영자 그룹 및 역할에 사용 권한을 할당합니다.

1 vSphere Client를 사용하여 **관리 > 액세스 제어 > 글로벌 사용 권한 > 사용 권한 추가**로 이동합니다.

2 플랫폼 운영자 역할을 플랫폼 운영자 그룹에 추가합니다.

- **추가**를 클릭합니다.
- 도메인으로 **vsphere.local**을 선택합니다.
- 사용자/그룹의 경우 **platform-operators-group** 그룹을 입력합니다.
- 역할의 경우 **platform-operators-role** 역할을 선택합니다.
- **하위 항목으로 전파** 확인란을 선택합니다.
- **확인**을 클릭하여 역할 권한으로 그룹을 업데이트합니다.

3 플랫폼 운영자 그룹에 vSphere Kubernetes 관리자 역할을 추가합니다.

- **추가**를 클릭합니다.
- 도메인으로 **vsphere.local**을 선택합니다.
- 사용자/그룹의 경우 **platform-operators-group** 그룹을 입력합니다.
- 역할에 대해 **vSphere Kubernetes 관리자** 역할을 선택합니다.
- **하위 항목으로 전파** 확인란을 선택합니다.
- **확인**을 클릭하여 역할 권한으로 그룹을 업데이트합니다.

6부: 플랫폼 운영자 그룹 및 역할 검증

새 플랫폼 운영자 그룹 및 역할을 테스트합니다. 그룹 및 역할이 보안 및 비즈니스 요구 사항을 충족하는지 확인하고 적절하게 조정해야 합니다.

1 vSphere Client를 사용하여 **platform-operator-00** 사용자 계정으로 vCenter Server에 로그인합니다.

2 호스트 및 클러스터, VM 및 템플릿, 스토리지 및 네트워킹을 비롯한 vSphere 개체에 대한 읽기 액세스 권한이 있는지 확인합니다.

3 컨텐츠 라이브러리를 생성하고 구성할 수 있는지 확인합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.

4 vSphere 네임스페이스를 생성하고 구성(사용자 추가, 컨텐츠 라이브러리 연결 및 스토리지 정책 할당 포함)할 수 있는지 확인합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

5 vSphere에 대한 Kubernetes CLI 도구를 사용하여 감독자에 로그인할 수 있는지 확인합니다. vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

6 Kubectl을 사용하여 감독자에서 TKG 클러스터를 프로비저닝할 수 있는지 확인합니다. Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

7 Kubectl을 사용하여 감독자에서 TKG 클러스터에 로그인할 수 있는지 확인합니다. kubectl을 사용하여 vCenter Single Sign-On 사용자로 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

8 감독자 TKG 클러스터에 워크로드를 배포할 수 있는지 확인합니다. 장 12 TKG 서비스 클러스터에 워크로드 배포의 내용을 참조하십시오.

9 감독자에서 TKG 클러스터에 대해 다양한 운영 작업을 수행할 수 있는지 확인합니다. 장 8 TKG 서비스 클러스터 운영의 내용을 참조하십시오.

10 vSphere Client를 사용하여 vSphere 네임스페이스의 감독자에서 프로비저닝된 TKG 클러스터를 볼 수 있는지 확인합니다.

11 vSphere Client를 사용하여 감독자 및 TKG 클러스터 개체를 모니터링할 수 있는지 확인합니다.

12 부정 테스트를 수행하여 vSphere 관리자용으로 예약된 특정 작업을 수행할 수 없는지 확인합니다. 예를 들어 새 vSphere 스토리지 정책 또는 vSphere 네트워킹을 생성할 수 없어야 합니다. 또한 워크로드 관리를 사용하지 않도록 설정할 수 없어야 합니다.

13 보안 및 비즈니스 요구 사항을 충족하도록 역할 권한을 적절히 조정합니다.

TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리

5

TKr(Tanzu Kubernetes 릴리스)은 TKG 서비스 클러스터에 대한 Kubernetes 소프트웨어 배포를 제공합니다. TKr은 VMware가 가상 시스템 템플릿으로 배포하며 사용자가 vCenter 컨텐츠 라이브러리를 사용하여 플랫폼과 동기화할 수 있습니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터에서 Kubernetes 릴리스 사용
- 컨텐츠 라이브러리 관리에 필요한 역할 사용 권한
- 구독 컨텐츠 라이브러리 생성
- 에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성
- 로컬 컨텐츠 라이브러리 게시 사용
- 기존 컨텐츠 라이브러리 편집
- 컨텐츠 라이브러리 마이그레이션
- TKr 확인 이해

TKG 서비스 클러스터에서 Kubernetes 릴리스 사용

Tanzu Kubernetes 릴리스(TKr)는 TKG 서비스 클러스터와 함께 사용하기 위해 VMware에서 서명 및 지원하는 Kubernetes 소프트웨어 배포판을 제공합니다. TKr 형식은 패키지 및 여러 운영 체제를 지원하도록 vSphere 8용으로 업데이트되었습니다.

TKr 릴리스 정보

사용 가능한 전체 TKr 목록, 각 릴리스의 새로운 기능, 알려진 문제 및 TKr 호환성에 대한 자세한 내용은 Tanzu Kubernetes 릴리스 정보를 참조하십시오.

TKr 배포 및 사용

VMware는 컨텐츠 전송 네트워크를 통해 Tanzu Kubernetes 릴리스를 배포합니다. vSphere 컨텐츠 라이브러리를 사용하여 TKr을 vSphere 네임스페이스와 연결합니다. TKr 사용을 자동화하려면 구독 컨텐츠 라이브러리 생성를 사용합니다. 인터넷이 제한된 환경의 경우 에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성를 사용합니다.

각 Tanzu Kubernetes 릴리스는 OVA 템플릿으로 배포됩니다. 감독자의 TKr 컨트롤러는 OVA 템플릿을 사용하여 TKG 클러스터 노드에 대한 VM을 구성합니다. VM 디스크 크기는 TKr OVA 템플릿에 의해 설정됩니다. TKG 서비스 클러스터에서 VM 클래스 사용를 사용하여 CPU 및 RAM 리소스를 지정합니다.

TKr 컨텐츠 라이브러리는 네임스페이스 범위가 아닙니다. 모든 vSphere 네임스페이스는 감독자의 TKG에 동일한 TKr 컨텐츠 라이브러리를 사용합니다. vSphere 네임스페이스 하나에 대한 TKr 컨텐츠 라이브러리를 편집하면 다른 모든 항목에 대해 업데이트됩니다.

TKr 이름 문자열

명령 `kubectl get tkr`을 사용하여 vSphere 네임스페이스에서 사용할 수 있는 TKr 이미지를 나열합니다. 예:

```
kubectl get tkr
NAME                                    VERSION                                 READY
COMPATIBLE   CREATED
v1.16.14---vmware.1-tkg.1.ada4837     v1.16.14+vmware.1-tkg.1.ada4837     False
False        19d
v1.17.17---vmware.1-tkg.1.d44d45a     v1.17.17+vmware.1-tkg.1.d44d45a     False
False        19d
v1.18.19---vmware.1-tkg.1.17af790     v1.18.19+vmware.1-tkg.1.17af790     False
False        19d
v1.19.16---vmware.1-tkg.1.df910e2     v1.19.16+vmware.1-tkg.1.df910e2     False
False        19d
v1.20.12---vmware.1-tkg.1.b9a42f3     v1.20.12+vmware.1-tkg.1.b9a42f3     False
False        19d
v1.21.6---vmware.1-tkg.1.b3d708a      v1.21.6+vmware.1-tkg.1.b3d708a      True
True         19d
v1.22.9---vmware.1-tkg.1.cc71bc8      v1.22.9+vmware.1-tkg.1.cc71bc8      True
True         19d
v1.23.8---vmware.2-tkg.2-zshippable   v1.23.8+vmware.2-tkg.2-zshippable   True
True         19d
v1.23.8---vmware.3-tkg.1              v1.23.8+vmware.3-tkg.1              True
True         19d
```

TKr NAME 문자열을 사용하여 감독자에서 TKG 클러스터를 프로비저닝합니다. TanzuKubernetesCluster v1alpha3 API를 사용하는 경우 `tkr.reference.name` 필드에 전체 TKr NAME 문자열을 제공합니다. Cluster v1beta1 API 를 사용하는 경우 `topology.version` 필드에 전체 TKr NAME 문자열을 제공합니다.

참고 클러스터 규격에서 TKr을 참조할 때 VERSION 문자열을 사용하지 마십시오. 형식은 TKr NAME 문자열과 정확하게 일치해야 합니다.

컨텐츠 라이브러리의 TKr 이름은 전체 TKr NAME 문자열이어야 합니다. 구독 컨텐츠 라이브러리를 사용하는 경우에는 TKr 이름 문자열이 생성됩니다. 로컬 컨텐츠 라이브러리를 사용하는 경우 TKr에 지정하는 이름이 TKr NAME 문자열과 일치하는지 확인합니다. 자세한 내용은 에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성를 참조하십시오.

TKG 서비스와의 TKr 호환성

TKr은 TKG 서비스 및 감독자와 독립적으로 릴리스 및 업데이트됩니다.

TKG 클러스터를 프로비저닝하려면 TKr이 TKG 서비스와 호환되어야 합니다. TKG 서비스와 호환되지 않는 TKr은 사용할 수 없습니다. 또한 업그레이드 대상인 버전에 대해 호환되는 TKr을 실행 중인지 확인해야 합니다.

`kubectl get tkr` 명령을 사용하여 TKr 호환성을 확인할 수 있습니다. COMPATIBLE 열은 부울을 반환합니다. `True`는 TKr이 호환됨을 의미하며, `False`는 TKr이 호환되지 않음을 의미합니다.

vSphere와의 TKr 호환성

vSphere 8의 TKr 형식이 업데이트되었습니다. vSphere 8용 TKr은 vSphere 8.x에서만 실행될 수 있습니다. vSphere 7.x용 TKr은 vSphere 7에서 작동하는 레거시 이미지입니다. 이러한 이미지는 vSphere 8에서 실행할 수 있지만 업그레이드용으로만 가능합니다. 레거시 TKr 이미지는 `legacy-tkr` 주석 레이블로 식별됩니다.

`kubectl get tkr -o yaml` 및 `kubectl get tkr --show-labels` 명령을 사용하여 TKr 호환성을 확인할 수 있습니다. `legacy-tkr` 주석 레이블이 있는 경우 TKr은 vSphere 8 기능을 지원하지 않으며 vSphere 7에서 vSphere 8로 업그레이드하는 데에만 사용해야 합니다.

예를 들어 다음 명령은 지정된 이미지가 `legacy-tkr`임을 보여 줍니다.

```
kubectl get tkr v1.23.8---vmware.3-tkg.1 -o yaml
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesRelease
metadata:
  creationTimestamp: "2023-03-15T20:33:17Z"
  finalizers:
  - tanzukubernetesrelease.run.tanzu.vmware.com
  generation: 1
  labels:
    os-arch: amd64
    os-name: photon
    os-type: linux
    os-version: "3.0"
    run.tanzu.vmware.com/legacy-tkr: ""
    v1: ""
    v1.23: ""
    v1.23.8: ""
    v1.23.8---vmware: ""
    v1.23.8---vmware.3: ""
    v1.23.8---vmware.3-tkg: ""
    v1.23.8---vmware.3-tkg.1: ""
  name: v1.23.8---vmware.3-tkg.1
```

다음 예에서는 `--show-labels` 플래그를 사용하여 TKr 호환성을 확인합니다. 레이블 `legacy-tkr`이 있으므로 이미지는 레거시 TKG 클러스터를 생성하는 데만 사용할 수 있습니다.

```
kubectl get tkr v1.23.8---vmware.3-tkg.1 --show-labels
NAME                        VERSION                  READY   COMPATIBLE   CREATED   LABELS
v1.23.8---vmware.3-tkg.1    v1.23.8+vmware.3-tkg.1   True    True         19d       os-
arch=amd64,os-name=photon,os-type=linux,os-version=3.0,run.tanzu.vmware.com/legacy-tkr=,
```

다음 예에서는 `legacy-tkr` 레이블이 레이블 목록에 없기 때문에 TKr이 vSphere 8.x용으로 특별히 구축되었음을 보여줍니다.

```
kubectl get tkr v1.28.8---vmware.1-fips.1-tkg.2 --show-labels
NAME                              VERSION                          READY   COMPATIBLE
CREATED   LABELS
v1.28.8---vmware.1-fips.1-tkg.2   v1.28.8+vmware.1-fips.1-tkg.2   True    True
21d       os-arch=amd64,os-name=photon,os-type=linux,os-version=5.0,tkr.tanzu.vmware.com/
standard=,v1.28.8---vmware.1-fips.1-tkg.2=,v1.28.8---vmware.1-fips.1-tkg=,v1.28.8---vmware.1-
fips.1=,v1.28.8---vmware.1-fips=,v1.28.8---vmware.1=,v1.28.8---vmware=,v1.28.8=,v1.28=,v1=
```

`-o yaml`를 사용하여 해당 명령을 실행하면 `legacy-tkr` 레이블이 없음이 표시되며 이는 TKr이 vSphere 8.x용으로 특별히 구축되었음을 나타냅니다.

```
 kubectl get tkr v1.28.8---vmware.1-fips.1-tkg.2 -o yaml
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesRelease
metadata:
  creationTimestamp: "2024-05-08T20:03:57Z"
  finalizers:
  - tanzukubernetesrelease.run.tanzu.vmware.com
  generation: 2
  labels:
    os-arch: amd64
    os-name: photon
    os-type: linux
    os-version: "5.0"
    tkr.tanzu.vmware.com/standard: ""
    v1: ""
    v1.28: ""
    v1.28.8: ""
    v1.28.8---vmware: ""
    v1.28.8---vmware.1: ""
    v1.28.8---vmware.1-fips: ""
    v1.28.8---vmware.1-fips.1: ""
    v1.28.8---vmware.1-fips.1-tkg: ""
    v1.28.8---vmware.1-fips.1-tkg.2: ""
  name: v1.28.8---vmware.1-fips.1-tkg.2
...
```

TKr 운영 체제 이미지 형식

TKr OS 이미지 형식은 단일 TKr에 대해 여러 운영 체제 이미지를 지원합니다. 따라서 지원되는 모든 운영 체제(현재 PhotonOS 및 Ubuntu)에 대해 특정 Kubernetes 버전에 대한 단일 Tanzu Kubernetes 릴리스가 있습니다. 기본 OS 이미지 형식은 PhotonOS입니다.

기본적으로 명명된 TKr의 PhotonOS 버전은 TKG 클러스터 노드에 사용됩니다. 참조된 TKr이 OS 이미지 형식을 지원하고 Ubuntu OS 버전을 사용할 수 있는 경우 `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` 주석을 사용하여 TKr의 Ubuntu OS 버전을 지정합니다. 예를 들어 다음 장 7 TKG 서비스 클러스터 프로비저닝은 명명된 TKr의 Ubuntu 버전을 사용합니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-ubuntu
  namespace: tkgs-cluster-ns
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
   topology:
     controlPlane:
       ...
       tkr:
         reference:
           name: v1.28.8---vmware.1-fips.1-tkg.2
```

OS 이미지 형식은 이기종 클러스터 배포를 지원합니다. 예를 들어 다음 클러스터 매니페스트는 제어부 노드에 대한 기본 PhotonOS와 작업자 노드에 대한 Ubuntu를 사용하여 Tanzu Kubernetes 클러스터를 생성합니다. TKr 버전은 제어부 섹션에서 참조되며 주석은 명명된 작업자 노드 풀에 대해 Ubuntu를 지정합니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-multiOS
  namespace: tkgs-cluster-ubuntu
  annotations:
    //Worker nodes annotation
    run.tanzu.vmware.com/resolve-os-image.np-1: os-name=ubuntu
spec:
  topology:
    controlPlane:
      tkr:
        reference:
          name: v1.28.8---vmware.1-fips.1-tkg.2
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkgs-storage-profile
    nodePools:
    - replicas: 3
      name: np-1
      vmClass: guaranteed-medium
      storageClass: tkgs-storage-profile
```

시스템이 vSphere 8로 업그레이드되면 기존 TKr은 단일 OS 이미지를 참조하여 TKr OS 이미지 형식으로 자동 변환됩니다. 이를 통해 레거시 TKr을 레거시가 아닌 TKr로 업그레이드할 수 있습니다.

레거시 TKr에는 Photon과 Ubuntu의 두 가지 버전이 있습니다. 레거시 TKr에 Ubuntu 특정 버전이 있는 경우 전체 버전 문자열('ubuntu' 포함)을 사용하고 주석 레이블을 생략하거나 짧은 버전 문자열('ubuntu' 제외)을 사용하고 버전 레이블을 포함할 수 있습니다.

TKr 패키지

vSphere 8과 호환되는 TKr 이미지는 CSI(Container Storage Interface) 및 CNI(Container Network Interface)와 같은 핵심 구성 요소에 대한 패키지 기반 프레임워크로 업데이트됩니다. v1beta API를 사용하는 경우 이러한 구성 요소의 변경 또는 업데이트는 사용자 지정 리소스 정의를 사용하여 수행됩니다.

TKr을 구성하는 패키지를 보려면 다음 명령을 실행합니다.

```
kubectl get tkr TKR-NAME -o yaml
```

예:

```
kubectl get tkr v1.28.8---vmware.1-fips.1-tkg.2 -o yaml
```

이 명령은 TKr의 모든 패키지를 반환합니다. 예:

```
spec:
  bootstrapPackages:
  - name: antrea.tanzu.vmware.com.1.13.3+vmware.3-tkg.2-vmware
  - name: vsphere-pv-csi.tanzu.vmware.com.3.1.0+vmware.1-tkg.6-vmware
  - name: vsphere-cpi.tanzu.vmware.com.1.28.0+vmware.1-tkg.1-vmware
  - name: kapp-controller.tanzu.vmware.com.0.48.2+vmware.1-tkg.1-vmware
  - name: guest-cluster-auth-service.tanzu.vmware.com.1.3.3+tkg.1-vmware
  - name: metrics-server.tanzu.vmware.com.0.6.2+vmware.3-tkg.5-vmware
  - name: secretgen-controller.tanzu.vmware.com.0.15.0+vmware.1-tkg.1-vmware
  - name: pinniped.tanzu.vmware.com.0.25.0+vmware.2-tkg.1-vmware
  - name: capabilities.tanzu.vmware.com.0.32.1+vmware.1
  - name: gateway-api.tanzu.vmware.com.1.0.0+vmware.1-tkg.1-vmware
  - name: calico.tanzu.vmware.com.3.26.3+vmware.1-tkg.1-vmware
```

사용 사례 예시는 v1beta1 예시: Calico CNI를 사용하는 클러스터 항목을 참조하십시오.

TKr OS 유형 마이그레이션

TKr 운영 체제 간의 인플레이스 클러스터 업데이트는 지원되지 않습니다. 예를 들어 TKr v1.27.11 Photon을 사용하는 TKG 클러스터를 TKr v1.28.8 Ubuntu로 업그레이드할 수 없습니다.

TKG 클러스터가 사용 중인 TKr OS 유형을 변경하려면 다음 절차를 고려하십시오. 이 예에서 원본 클러스터는 TKr Photon을 사용하고 대상은 TKr Ubuntu입니다.

- Velero를 사용하여 Photon 기반 TKG 클러스터 워크로드를 백업합니다.

  장 20 TKG 서비스 클러스터 및 워크로드 백업 및 복원의 내용을 참조하십시오.

- Ubuntu TKr을 사용하여 새 TKG 클러스터를 프로비저닝합니다.

  장 7 TKG 서비스 클러스터 프로비저닝의 내용을 참조하십시오.

- Velero를 사용하여 TKG 클러스터 워크로드를 Ubuntu 클러스터로 복원합니다.

  장 20 TKG 서비스 클러스터 및 워크로드 백업 및 복원의 내용을 참조하십시오.

TKr 강화

감독자 및 TKr을 포함한 시스템 구성 요소에 대한 STIG(Security Technical Implementation Guide)를 사용할 수 있습니다. 자세한 내용은 Tanzu STIG 강화를 참조하십시오.

자체 TKr 구축

TKr v1.25.7 for vSphere 8.x부터 vSphere 8에서 TKG 클러스터에 대한 사용자 지정 TKr 시스템 이미지를 구축할 수 있습니다. 사용자 지정 시스템 이미지는 지원되는 운영 체제 및 버전, 릴리스된 TKr을 기준으로 하는 Kubernetes 버전 및 사용자가 지정한 모든 사용자 지정을 패키징합니다.

TKG 클러스터 노드에 대한 사용자 지정 시스템 이미지를 구축하려면 vSphere Tanzu Kubernetes Grid Image Builder를 사용합니다. 사용자 지정 이미지 구축, 지원되는 TKr 버전 및 지원되는 사용자 지정에 대한 자세한 내용은 설명서를 참조하십시오.

컨텐츠 라이브러리 관리에 필요한 역할 사용 권한

컨텐츠 라이브러리를 관리하려면 몇 가지 vSphere 역할 사용 권한이 필요합니다. vSphere 관리자 역할을 사용하지 않는 경우 필요에 따라 사용 권한 목록을 참조하십시오.

컨텐츠 라이브러리 관리에 필요한 사용 권한

컨텐츠 라이브러리를 관리하려면 다음과 같은 vSphere 역할 사용 권한이 필요합니다.

이러한 사용 권한은 vSphere 관리자 역할에 포함되어 있습니다.

이러한 사용 권한을 포함하는 전용 역할을 생성하려면 플랫폼 운영자를 위한 전용 그룹 및 역할 생성 항목을 참조하십시오.

```
- Content Library
  - Add library item &
  - Check in a template &
  - Check out a template &
  - Create local library &
  - Create subscribed library &
  - Delete library item &
  - Delete local library &
  - Delete subscribed library &
  - Download files &
  - Evict library item &
  - Evict subscribed library &
  - Import storage &
  - Probe subscription information &
  - Read storage &
  - Sync library item &
  - Sync subscribed library &
  - Type introspection &
```

```
- Update configuration settings &
- Update library &
- Update library item &
- Update local library &
- Update subscribed library &
- View configuration settings &
```

구독 컨텐츠 라이브러리 생성

감독자에서 TKG 클러스터를 프로비저닝하기 위해 구독 컨텐츠 라이브러리를 생성하고 Tanzu Kubernetes 릴리스를 동기화할 수 있습니다. 구독 컨텐츠 라이브러리를 사용하면 TKR 배포를 자동화하고 최신 릴리스로 최신 상태를 유지할 수 있습니다.

감독자의 TKG에 대해 VMware는 컨텐츠 전송 네트워크에 Tanzu Kubernetes 릴리스를 게시합니다. 이러한 이미지 게시 자료를 구독하는 컨텐츠 라이브러리를 생성할 수 있습니다. 동기화 모드는 '즉시' 또는 '요청 시' 중에 선택합니다.

경고 공용 컨텐츠 전송 네트워크에서 Tanzu Kubernetes 릴리스를 즉시 동기화하면 상당한 시간과 디스크 공간이 소요될 수 있습니다.

사전 요구 사항

컨텐츠 라이브러리 기능은 감독자의 TKG가 의존하는 vCenter Server의 기능입니다. 자세한 내용은 vSphere 가상 시스템 관리 설명서에서 "컨텐츠 라이브러리 사용"을 참조하십시오.

절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **메뉴 > 컨텐츠 라이브러리**를 선택합니다.

3 **생성**을 클릭합니다.

새 컨텐츠 라이브러리 마법사가 열립니다.

4 컨텐츠 라이브러리의 **이름 및 위치**를 지정하고 완료되면 **다음**을 클릭합니다.

필드	설명
이름	알아보기 쉬운 이름을 입력합니다(예: **`TKR-sub`**).
참고	설명을 포함합니다(예: **`Subscription library for TKRs for TKG2`**).
vCenter Server	**워크로드 관리**를 사용하도록 설정된 vCenter Server 인스턴스를 선택합니다.

5 **컨텐츠 라이브러리 구성** 페이지에서 컨텐츠 라이브러리 구독을 구성하고 완료되면 **다음**을 클릭합니다.

a **구독 컨텐츠 라이브러리** 옵션을 선택합니다.

b 게시자의 **구독 URL** 주소를 입력합니다.

```
https://wp-content.vmware.com/v2/latest/lib.json
```

c **컨텐츠 다운로드** 옵션의 경우 다음 중 하나를 선택합니다.

옵션	설명
즉시	구독 프로세스는 라이브러리 메타데이터와 이미지를 모두 동기화합니다. 게시된 라이브러리에서 항목이 삭제된 경우, 해당 컨텐츠는 구독 라이브러리 스토리지에 남아 있으므로 수동으로 삭제해야 합니다.
필요한 경우	구독 프로세스는 라이브러리 메타데이터만 동기화합니다. Tanzu Kubernetes Grid Service는 게시될 때 이미지를 다운로드합니다. 항목이 더 이상 필요하지 않은 경우에는 항목 컨텐츠를 삭제하여 스토리지 공간을 확보할 수 있습니다. 스토리지를 절약하려면 이 옵션을 사용하는 것이 좋습니다.

6 메시지가 표시되면 SSL 인증서 지문을 수락합니다.

SSL 인증서 지문은 인벤토리에서 구독 컨텐츠 라이브러리를 삭제할 때까지 시스템에 저장됩니다.

7 **보안 정책 적용** 페이지에서 OVF 보안 정책을 구성하고 완료되면 **다음**을 클릭합니다.

a **보안 정책 적용**을 선택합니다.

b **OVF 기본 정책**을 선택합니다.

이 옵션을 선택하면 동기화 프로세스 중에 OVF 서명 인증서가 확인됩니다. 인증서 검증을 통과하지 못한 OVF 템플릿은 **확인 실패** 태그로 표시됩니다. 템플릿 메타데이터는 유지되지만 OVF 파일은 동기화할 수 없습니다.

참고 현재 **OVF 기본 정책**은 지원되는 유일한 보안 정책입니다.

8 **스토리지 추가** 페이지에서 컨텐츠 라이브러리 컨텐츠의 스토리지 위치로 사용할 데이터스토어를 선택하고 **다음**을 클릭합니다.

9 **완료 준비** 페이지에서 세부 정보를 검토하고 **마침**을 클릭합니다.

10 **컨텐츠 라이브러리** 페이지에서 생성한 새 컨텐츠 라이브러리를 선택합니다.

11 라이브러리 컨텐츠의 동기화를 확인하거나 완료합니다.

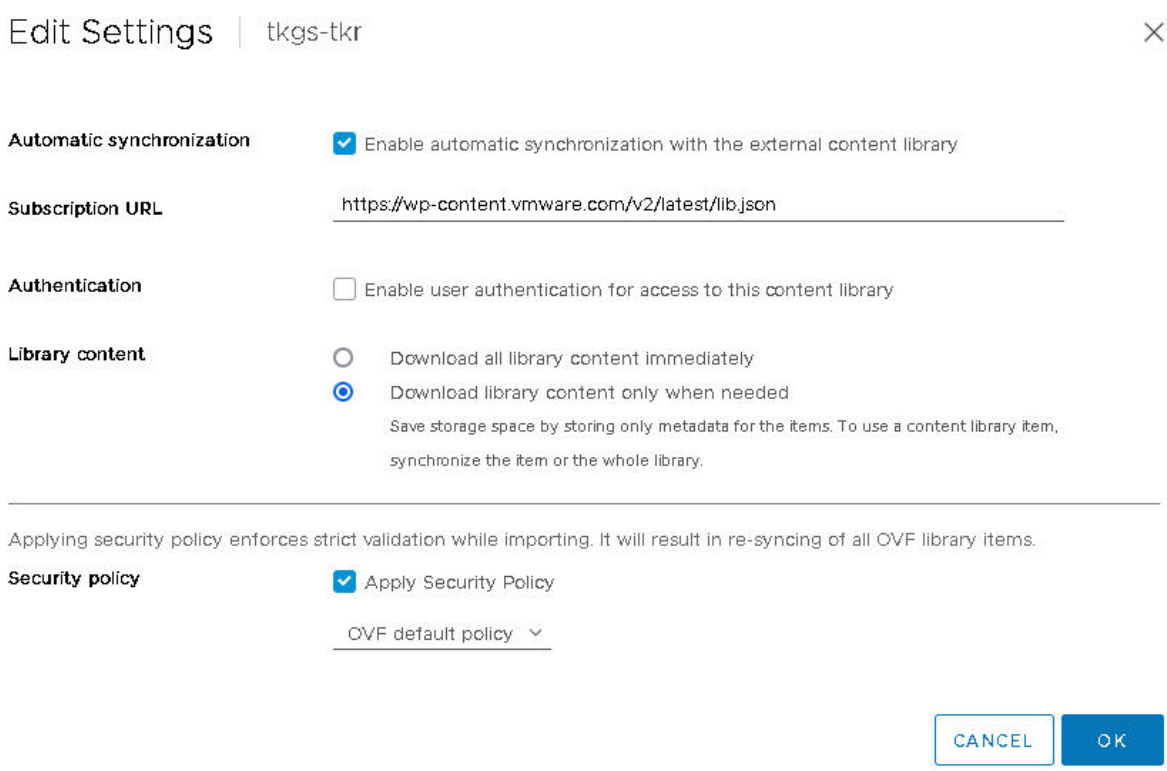
동기화 옵션	설명
즉시	모든 컨텐츠를 즉시 다운로드하도록 선택한 경우 라이브러리가 동기화되었는지 확인합니다. 동기화된 라이브러리 컨텐츠를 보려면 **템플릿 > OVF 및 OVA 템플릿**을 선택합니다.
필요한 경우	요청 시 라이브러리를 동기화하도록 선택한 경우 다음 두 가지 옵션을 사용할 수 있습니다. ■ **작업** > **동기화**를 사용하여 전체 라이브러리를 동기화합니다. ■ 항목을 마우스 오른쪽 버튼으로 클릭하고 **동기화**를 선택하여 항목만 동기화합니다. 동기화된 라이브러리 컨텐츠를 보려면 **템플릿 > OVF 및 OVA 템플릿**을 선택합니다.

12 **필요한 경우** 옵션을 선택한 경우 사용하려는 OVF 템플릿을 다운로드합니다.

필요한 경우 옵션을 선택하면 이미지 파일은 로컬로 저장되지 않고 메타데이터만 저장됩니다. 템플릿 파일을 다운로드하려면 항목을 선택하고 마우스 오른쪽 버튼을 클릭한 다음 **항목 동기화**를 선택합니다.

13 구독 컨텐츠 라이브러리 설정을 업데이트하려면 **작업 > 설정 편집**을 선택합니다.

설정	값
구독 URL	`https://wp-content.vmware.com/v2/latest/lib.json`
인증	사용 안 함
라이브러리 컨텐츠	필요한 경우 다운로드
보안 정책	OVF 기본 정책



다음에 수행할 작업

Tanzu Kubernetes 릴리스 컨텐츠 라이브러리는 TKG 클러스터를 프로비저닝하는 각 vSphere 네임스페이스와 연결되어야 합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성

감독자에서 TKG 클러스터를 프로비저닝하기 위해 로컬 컨텐츠 라이브러리를 생성하고 Tanzu Kubernetes 릴리스를 가져올 수 있습니다. 로컬 컨텐츠 라이브러리의 일반적인 사용 사례는 인터넷이 제한된 환경(에어갭)입니다.

로컬 컨텐츠 라이브러리를 생성하려면 라이브러리를 구성하고 OVA 파일을 다운로드하여 로컬 컨텐츠 라이브러리로 가져와야 합니다.

사전 요구 사항

컨텐츠 라이브러리 기능은 감독자의 TKG가 의존하는 vCenter Server의 기능입니다. 자세한 내용은 컨텐츠 라이브러리 사용을 참조하십시오.

절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **메뉴**를 클릭합니다.

3 **컨텐츠 라이브러리**를 클릭합니다.

4 **생성**을 클릭합니다.

시스템에 **새 컨텐츠 라이브러리** 마법사가 표시됩니다.

5 컨텐츠 라이브러리의 **이름 및 위치**를 지정하고 완료되면 **다음**을 클릭합니다.

필드	설명
이름	알아보기 쉬운 이름을 입력합니다(예: `TKr-local`).
참고	설명을 포함합니다(예: `Local library for TKrs for TKG`).
vCenter Server	vSphere IaaS control plane를 사용하도록 설정된 vCenter Server 인스턴스를 선택합니다.

6 **컨텐츠 라이브러리 구성** 페이지에서 **로컬 컨텐츠 라이브러리**를 선택하고 **다음**을 클릭합니다.

로컬 컨텐츠 라이브러리의 경우 사용하려는 OVF 템플릿을 수동으로 가져옵니다.

7 **보안 정책 적용** 페이지에서 OVF 보안 정책을 구성하고 완료되면 **다음**을 클릭합니다.

a **보안 정책 적용**을 선택합니다.

b **OVF 기본 정책**을 선택합니다.

이 옵션을 선택하면 동기화 프로세스 중에 OVF 서명 인증서가 확인됩니다. 인증서 검증을 통과하지 못한 OVF 템플릿은 **확인 실패** 태그로 표시됩니다. 템플릿 메타데이터는 유지되지만 OVF 파일은 동기화할 수 없습니다.

참고 현재 **OVF 기본 정책**은 지원되는 유일한 보안 정책입니다.

8 **스토리지 추가** 페이지에서 컨텐츠 라이브러리 컨텐츠의 스토리지 위치로 사용할 데이터스토어를 선택하고 **다음**을 클릭합니다.

9 **완료 준비** 페이지에서 세부 정보를 검토하고 **마침**을 클릭합니다.

10 **컨텐츠 라이브러리** 페이지에서 생성한 새 컨텐츠 라이브러리를 선택합니다.

11 로컬 컨텐츠 라이브러리로 가져오려는 각 Tanzu Kubernetes 릴리스에 대한 OVA 파일을 다운로드합니다.

a 브라우저를 사용하여 다음 URL로 이동합니다.

https://wp-content.vmware.com/v2/latest/

b 원하는 이미지의 디렉토리를 클릭합니다. 일반적으로 이 디렉토리는 Kubernetes 배포의 최신 또는 최신 버전입니다.

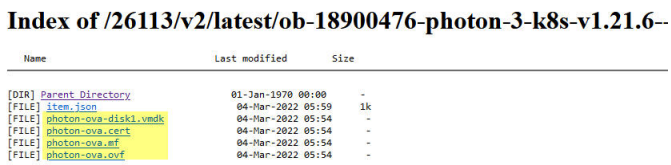
예:

```
ob-18186591-photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067
```

중요 배포 이름을 사용하여 파일을 로컬 컨텐츠 라이브러리로 가져와야 합니다. 절차를 완료할 때까지 대상 이름을 파일에 복사하거나 브라우저를 열어 두어야 합니다. 위 예제를 기반으로 필요한 이름 문자열의 필수 부분은 `photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067`입니다.

c 다음 각 파일에 대해 마우스 오른쪽 버튼을 클릭하고 **다른 이름으로 링크 저장**을 선택합니다.

- `photon-ova-disk1.vmdk`
- `photon-ova.cert`
- `photon-ova.mf`
- `photon-ova.ovf`



d 각 파일이 로컬 파일 시스템으로 다운로드되었는지 확인합니다.

참고 가져오는 파일은 OVF 및 VMDK 파일입니다. 단, 보안 정책이 적용되면 가져오는 동안 소스 디렉터리에 인증서(`*.cert`), 매니페스트(`*.mf`)를 포함한 4개의 파일이 모두 있어야 합니다. 가져오는 동안 인증서 및 매니페스트 파일을 사용할 수 없으면 가져온 Tanzu Kubernetes 릴리스를 사용할 수 없습니다.

12 OVA 및 VMDK 파일을 로컬 컨텐츠 라이브러리로 가져옵니다.

a **메뉴 > 컨텐츠 라이브러리 >** 를 선택합니다.

b **컨텐츠 라이브러리** 목록에서 직접 생성한 로컬 컨텐츠 라이브러리의 이름 링크를 클릭합니다.

c **작업**을 클릭합니다.

d **항목 가져오기**를 선택합니다.

e **라이브러리 항목 가져오기** 창에서 **로컬 파일**를 선택합니다.

f **파일 업로드**를 클릭합니다.

g `photon-ova.ovf` 및 `photon-ova-disk1.vmdk` 파일을 둘 다 선택합니다.

`2 files ready to import` 메시지가 표시됩니다. 각 파일의 이름 옆에 녹색 확인 표시가 나타납니다.

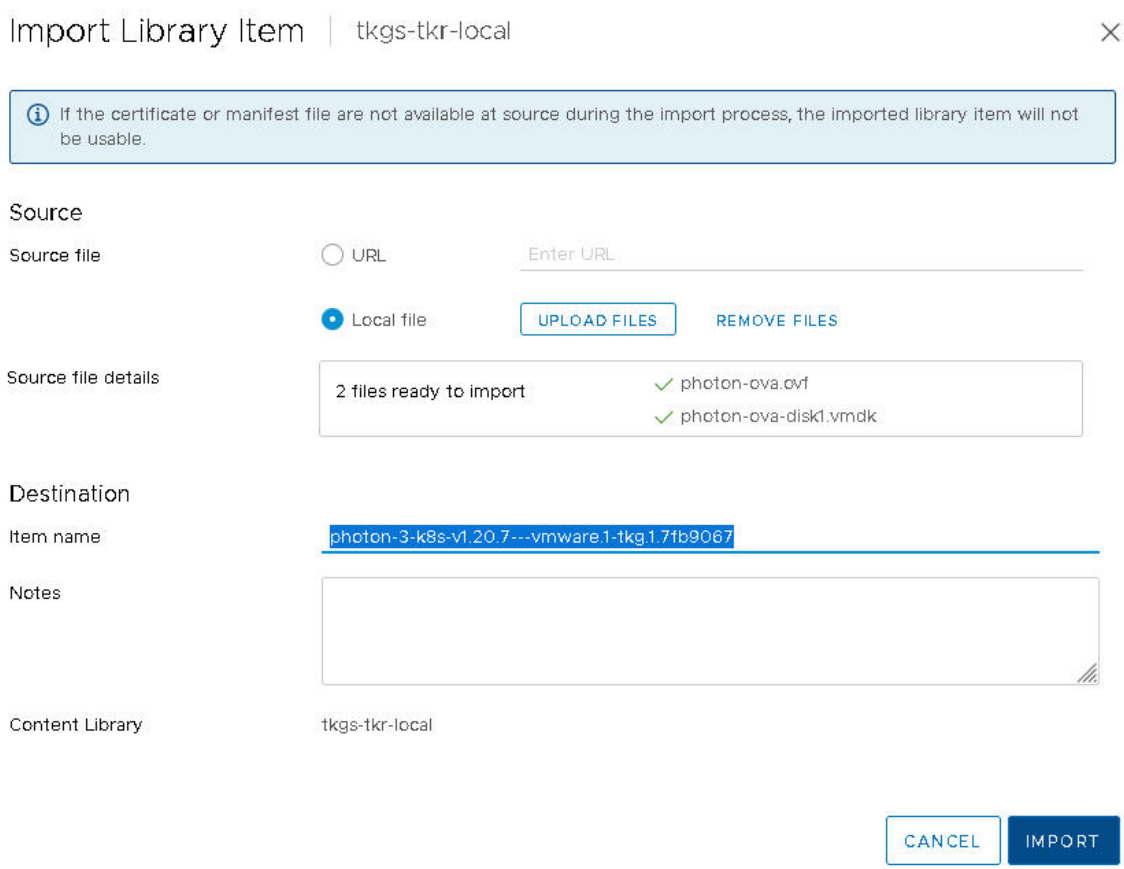
h **대상 항목** 이름을 OS 이미지 버전과 파일을 다운로드한 디렉토리의 Kubernetes 버전으로 변경합니다.

예:

```
photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067
```

경고 컨텐츠 라이브러리 **대상 항목** 이름은 원하는 Tanzu Kubernetes 릴리스의 폴더 이름 문자열과 정확히 일치해야 합니다. 이름이 일치하지 않으면 감독자가 이미지를 유효한 Tanzu Kubernetes 릴리스로 확인할 수 없습니다.

i **가져오기**를 클릭합니다.



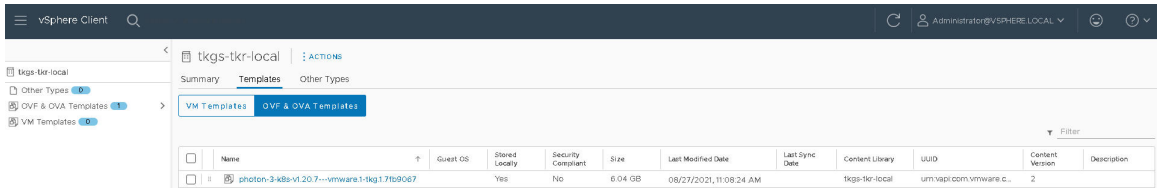
13 로컬 컨텐츠 라이브러리가 Tanzu Kubernetes 릴리스로 채워졌는지 확인합니다.

a 페이지 하단의 **최근 작업** 창을 나타냅니다.

b **라이브러리 항목의 컨텐츠 가져오기** 작업을 모니터링하고 성공적으로 **완료**되었는지 확인합니다.

c 로컬 컨텐츠 라이브러리에서 **템플릿 > OVF 및 OVA 템플릿**을 선택합니다.

d Tanzu Kubernetes 릴리스 메타데이터가 나열되고 해당 컨텐츠가 로컬에 저장되어 있는지 확인합니다.



다음에 수행할 작업

Tanzu Kubernetes 릴리스 컨텐츠 라이브러리는 TKG 클러스터를 프로비저닝하는 각 vSphere 네임스페이스와 연결되어야 합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

로컬 컨텐츠 라이브러리 게시 사용

로컬 컨텐츠 라이브러리를 생성하고 게시가 가능하도록 설정하면 다른 컨텐츠 라이브러리에서 구독할 수 있습니다. 이 방식을 사용하면 TKG 클러스터 운영자에게 구독 스타일의 TKR 서비스를 제공하면서 사용 가능한 Tanzu Kubernetes 릴리스 이미지를 제어할 수 있습니다.

모든 이미지가 게시되고 유지되는 Tanzu Kubernetes 릴리스용 VMware 컨텐츠 전송 네트워크를 구독하는 대신 Tanzu Kubernetes 릴리스의 제한된 하위 집합을 제공하는 로컬 컨텐츠 라이브러리를 생성할 수 있습니다. 이러한 방식은 사용 가능한 이미지를 제어하려는 경우(예: 지속적인 빌드 환경에서) 편리합니다.

절차

1 에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성를 생성하거나 기존 라이브러리의 설정을 편집합니다.

2 **게시 사용** 확인란을 선택합니다.

이 로컬 컨텐츠 라이브러리에 대한 구독 URL을 받게 됩니다.

3 구독 URL을 사용하여 하나 이상의구독 컨텐츠 라이브러리 생성를 생성합니다.

결과

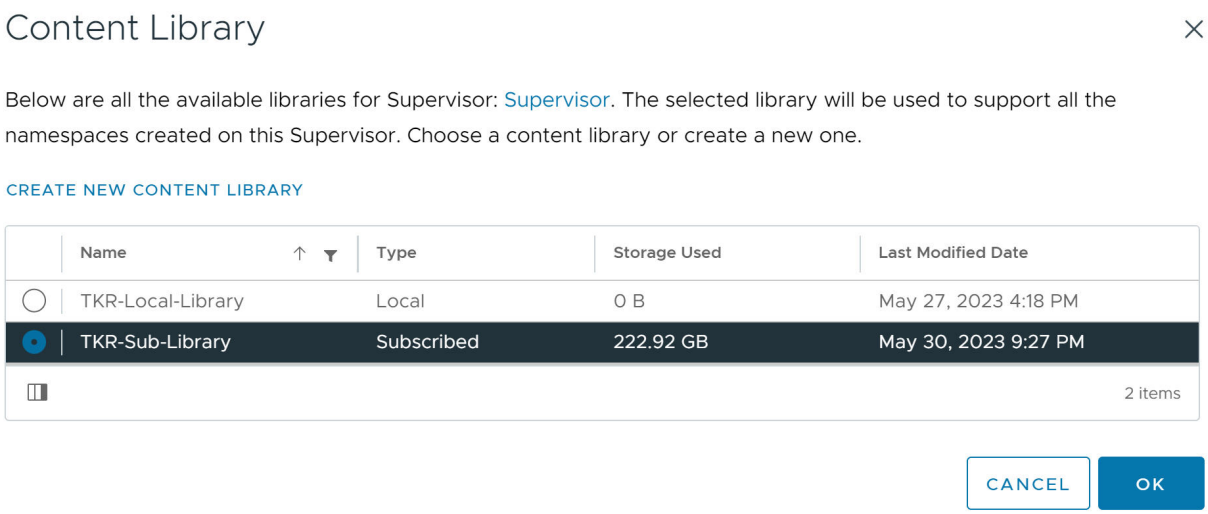
게시가 가능하도록 설정된 로컬 컨텐츠 라이브러리에 Tanzu Kubernetes 릴리스를 업로드하면 구독한 모든 컨텐츠 라이브러리가 동기화 시 이미지를 수신합니다. 이미지를 업로드하는 방법에 대한 지침은 에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성 설명서를 참조하고, 구독 컨텐츠 라이브러리 생성에 대한 설명서는 구독 컨텐츠 라이브러리 생성를 참조하십시오.

기존 컨텐츠 라이브러리 편집

TKG 클러스터에 사용하는 TKR 컨텐츠 라이브러리의 범위는 개별 vSphere 네임스페이스가 아닙니다. 동일한 TKR 컨텐츠 라이브러리가 감독자 인스턴스의 모든 vSphere 네임스페이스에 사용됩니다. 이러한 이유로 TKR 컨텐츠 라이브러리 변경은 감독자 수준에서만 허용됩니다.

vSphere 네임스페이스에서 사용할 기존 TKR 컨텐츠 라이브러리를 변경하려면 다음 단계를 완료합니다.

그림 5-1. TKG에 대한 TKR 컨텐츠 라이브러리 편집



절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리**를 선택합니다.

3 감독자 인스턴스를 선택합니다.

4 **감독자 > 일반**을 선택합니다.

5 Tanzu Kubernetes Grid Service를 선택합니다.

6 **컨텐츠 라이브러리 > 편집**을 선택합니다.

7 사용하려는 **컨텐츠 라이브러리**를 선택하고 **확인**을 클릭합니다.

8 새 컨텐츠 라이브러리를 생성하려면 **새 컨텐츠 라이브러리 생성**을 선택합니다. 구독 컨텐츠 라이브러리 생성 및 에어갭 클러스터 프로비저닝을 위한 로컬 컨텐츠 라이브러리 생성의 내용을 참조하십시오.

결과

선택한 컨텐츠 라이브러리는 감독자에서 생성된 모든 vSphere 네임스페이스에서 Tanzu Kubernetes Grid 서비스에 사용됩니다.

컨텐츠 라이브러리 마이그레이션

TKr 컨텐츠 라이브러리가 용량에 도달하면 스토리지 용량이 더 많은 새 컨텐츠 라이브러리로 마이그레이션할 수 있습니다.

vSphere 관리자가 컨텐츠 라이브러리를 생성할 때 관리자는 라이브러리 컨텐츠를 저장할 데이터스토어를 지정합니다(이 경우 OVA 파일). 더 많은 Kubernetes 버전이 배포되면 각 업데이트에 대해 OVA 파일이 추가되면서 컨텐츠 라이브러리의 크기가 확장됩니다. 컨텐츠 라이브러리에 명시적 용량은 없지만 해당 데이터스토어 용량에 의해 제한됩니다.

컨텐츠 라이브러리가 용량에 도달하면 `Internal error occurred: get library items failed for.` 메시지가 표시될 수 있습니다. 이런 경우 TKG 클러스터를 새 컨텐츠 라이브러리로 마이그레이션하여 스토리지 용량을 늘릴 수 있습니다. 마이그레이션은 vSphere Client를 사용하여 수행됩니다.

절차

1 대상 클러스터에 충분한 용량을 가진 새 컨텐츠 라이브러리를 생성합니다.

2 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

3 **메뉴 > 호스트 및 클러스터**를 선택합니다.

4 Tanzu Kubernetes 클러스터가 포함된 감독자가 프로비저닝된 vSphere 클러스터 개체를 선택합니다.

5 **구성** 탭을 선택합니다.

6 탐색 패널에서 **네임스페이스 > 일반 >** 옵션을 선택합니다.

7 기본 패널의 **컨텐츠 라이브러리** 섹션 옆에 있는 **편집**을 클릭합니다.

8 생성한 새 컨텐츠 라이브러리를 선택하고 **확인**을 클릭합니다.

이 작업은 클러스터 구성에 대한 업데이트를 트리거합니다.

참고 컨텐츠 라이브러리를 수정한 후 TKG 클러스터가 컨텐츠 소스의 변경 내용을 적용하는 데 최대 10분이 소요될 수 있습니다.

TKr 확인 이해

이 항목에서는 시스템이 TKr 이미지를 확인하는 방법을 설명합니다.

TKr 확인

클러스터 개체가 생성되거나 업데이트되면 Kubernetes API 서버는 TKr Resolver 변형 Webhook을 호출합니다. 클러스터(또는 해당 ClusterClass)에는 `run.tanzu.vmware.com/resolve-tkr` 주석이 있어야 합니다. 그렇지 않으면 TKr 확인을 완전히 건너뜁니다. TKr Resolver는 `run.tanzu.vmware.com/resolve-tkr` 주석 값을 레이블 쿼리로 사용하여 후보 TKr 집합을 제한합니다. 빈 문자열은 모든 TKr을 선택합니다.

클러스터 토폴로지 controlPlain 및 machineDeployments의 `run.tanzu.vmware.com/resolve-os-image` 주석 값은 각각 controlPlain 및 machineDeployments와 함께 사용되는 OSImage 개체의 레이블 선택기로 사용됩니다. 확인된 TKr이 제공한 정확히 하나의 OSImage는 controlPlane 또는 machineDeployments 중 하나에 대한 쿼리를 충족해야 합니다.

제공된 클러스터 spec.topology.version은 버전 접두사로 사용됩니다. TKr Resolver Webhook은 위의 제약 조건을 충족하는 사용 가능한 최신 TKR을 찾습니다. 찾을 수 없으면 클러스터 생성/업데이트 요청이 거부됩니다.

TKr Resolver Webhook은 클러스터를 변형합니다.

- run.tanzu.vmware.com/tkr 레이블은 확인된 TKR의 이름으로 설정됩니다.
- 클러스터 spec.topology.version은 확인된 TKR의 Kubernetes 버전으로 설정됩니다.
- 클러스터 변수 TKR_DATA는 Kubernetes 버전에서 controlPlane에 대한 OSImage 및 TKR의 값 집합으로의 매핑을 포함하도록 업데이트됩니다.
- 개별 machineDeployments에 대한 TKR_DATA 변수 재정의는 Kubernetes 버전에서 machineDeployment에 대한 TKR 및 OSImage의 값 집합으로의 매핑을 포함하도록 업데이트됩니다.

TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성

6

vSphere 네임스페이스는 TKG 서비스 클러스터에 대한 런타임 환경을 제공합니다. TKG 서비스 클러스터를 프로비저닝하려면 먼저 사용자, 역할, 계산, 스토리지, 컨텐츠 라이브러리 및 가상 시스템 클래스로 vSphere 네임스페이스를 구성합니다. 이 구성은 해당 네임스페이스에서 실행되는 TKG 서비스 클러스터에 상속됩니다.

다음으로 아래 항목을 읽으십시오.

- vSphere 네임스페이스 클러스터에 TKG 서비스 사용
- TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성
- TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성
- vSphere 네임스페이스에 대한 워크로드 네트워크 설정 재정의
- TKG 서비스 클러스터에서 VM 클래스 사용
- TKG 서비스 클러스터를 호스팅하기 위한 vSphere 네임스페이스 준비 상태 확인
- Kubectl을 사용하여 vSphere 네임스페이스 생성 사용
- vSphere 네임스페이스 제거

vSphere 네임스페이스 클러스터에 TKG 서비스 사용

vSphere 네임스페이스는 감독자의 네트워크 범위 테넌시입니다. vSphere 네임스페이스는 TKG 서비스 클러스터를 호스팅하고 네트워킹, 역할 사용 권한, 영구 스토리지, 리소스 할당량, 컨텐츠 라이브러리 및 VM 클래스 통합을 제공하는 데 사용됩니다.

vSphere 네임스페이스 네트워크

vSphere 네임스페이스 네트워크는 **감독자 > 워크로드 네트워크 > 네임스페이스 네트워크**에서 분할된 서브넷입니다. **네임스페이스 서브넷 접두사**는 각 vSphere 네임스페이스에 대해 예약된 서브넷의 크기를 정의합니다. 기본값은 /28입니다.

vSphere 네임스페이스 네트워크는 TKG 클러스터에 대한 감독자 연결을 제공합니다. 기본적으로 vSphere 네임스페이스는 클러스터 수준 네트워크 구성을 사용하고 서브넷에서 IP 주소를 할당합니다. vSphere 네임스페이스를 생성하면 /28 오버레이 세그먼트 및 해당 IP 풀이 해당 vSphere 네임스페이스의 서비스 포드로 인스턴스화됩니다.

첫 번째 TKG 클러스터가 vSphere 네임스페이스에 프로비저닝되면 TKG 클러스터는 vSphere 네임스페이스와 동일한 서브넷을 공유합니다. 해당 vSphere 네임스페이스에 프로비저닝된 후속 TKG 클러스터 각각에 대해 해당 클러스터에 대한 새 서브넷이 생성되고 vSphere 네임스페이스 게이트웨이에 연결됩니다.

vSphere 네임스페이스에는 kubectl 트래픽을 각 TKG 클러스터 제어부로 라우팅하는 공유 로드 밸런서 인스턴스가 있습니다. 또한 TKG 클러스터에서 리소스가 제공되는 각 Kubernetes 서비스 로드 밸런서에 대해 해당 서비스에 대한 계층 4 로드 밸런서 인스턴스가 생성됩니다.

동일한 vSphere 네임스페이스 내의 TKG 클러스터는 North-South 연결을 위한 SNAT IP를 공유합니다. 네임스페이스 간의 East-West 연결은 SNAT가 아닙니다.

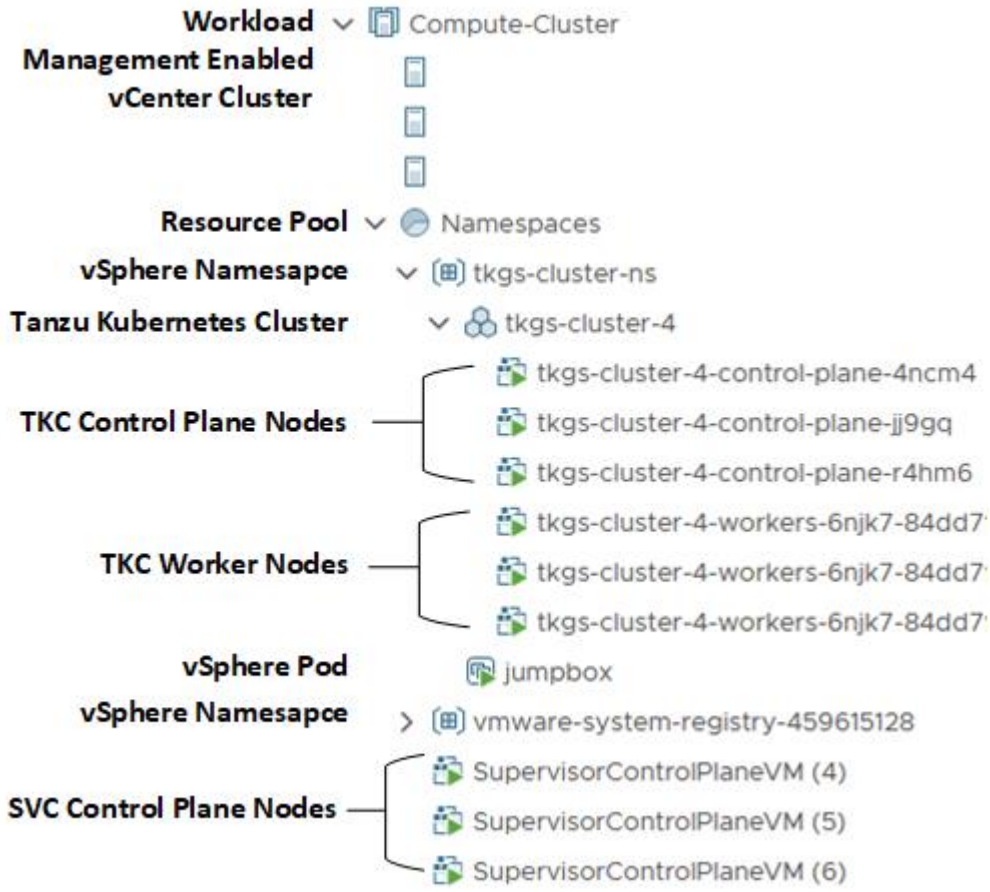
vSphere 네임스페이스는 일반적으로 라우팅할 수 없습니다. 그러나 NSX 네트워킹을 사용하는 경우 라우팅 가능한 서브넷으로 vSphere 네임스페이스 네트워크를 재정의할 수 있습니다. vSphere 네임스페이스에 대한 워크로드 네트워크 설정 재정의의 내용을 참조하십시오.

vSphere 네임스페이스 리소스 풀

단일 vSphere 영역 감독자 배포에서 vSphere 네임스페이스를 생성하면 해당 네임스페이스를 지원하는 리소스 풀이 생성됩니다. vSphere 네임스페이스는 감독자에서 계산, 스토리지, 사용 권한, 클래스 및 이미지를 포함한 리소스의 논리적 단위를 제공합니다. 예를 들어, vSphere 네임스페이스에서 CPU 또는 메모리 제한을 구성하면 해당 네임스페이스를 지원하는 리소스 풀에 동일한 리소스 제한이 적용됩니다. 이런 방식으로 vSphere 네임스페이스는 감독자에서 다중 테넌시를 사용하도록 설정합니다.

동일한 다중 테넌트 환경이 3개 vSphere 영역에 배포된 감독자에 적용됩니다. 영역이 지정된 감독자에 vSphere 네임스페이스가 생성되면 시스템은 해당 감독자를 지원하는 각 vSphere 클러스터에 리소스 풀을 생성합니다. 이렇게 하면 해당 vSphere 네임스페이스에 프로비저닝된 TKG 클러스터를 이 감독자에 속하는 영역에 배포할 수 있습니다.

vSphere Client를 사용하면 **호스트 및 클러스터** 관점을 선택하여 vSphere 네임스페이스 리소스 풀 및 개체를 볼 수 있고 **VM 및 템플릿** 보기를 선택하여 볼 수도 있습니다. TKG 클러스터를 프로비저닝하면 대상 vSphere 네임스페이스에 생성됩니다. 영역이 지정된 감독자 배포에서는 각 vSphere 클러스터에 동일한 리소스 풀이 있습니다.



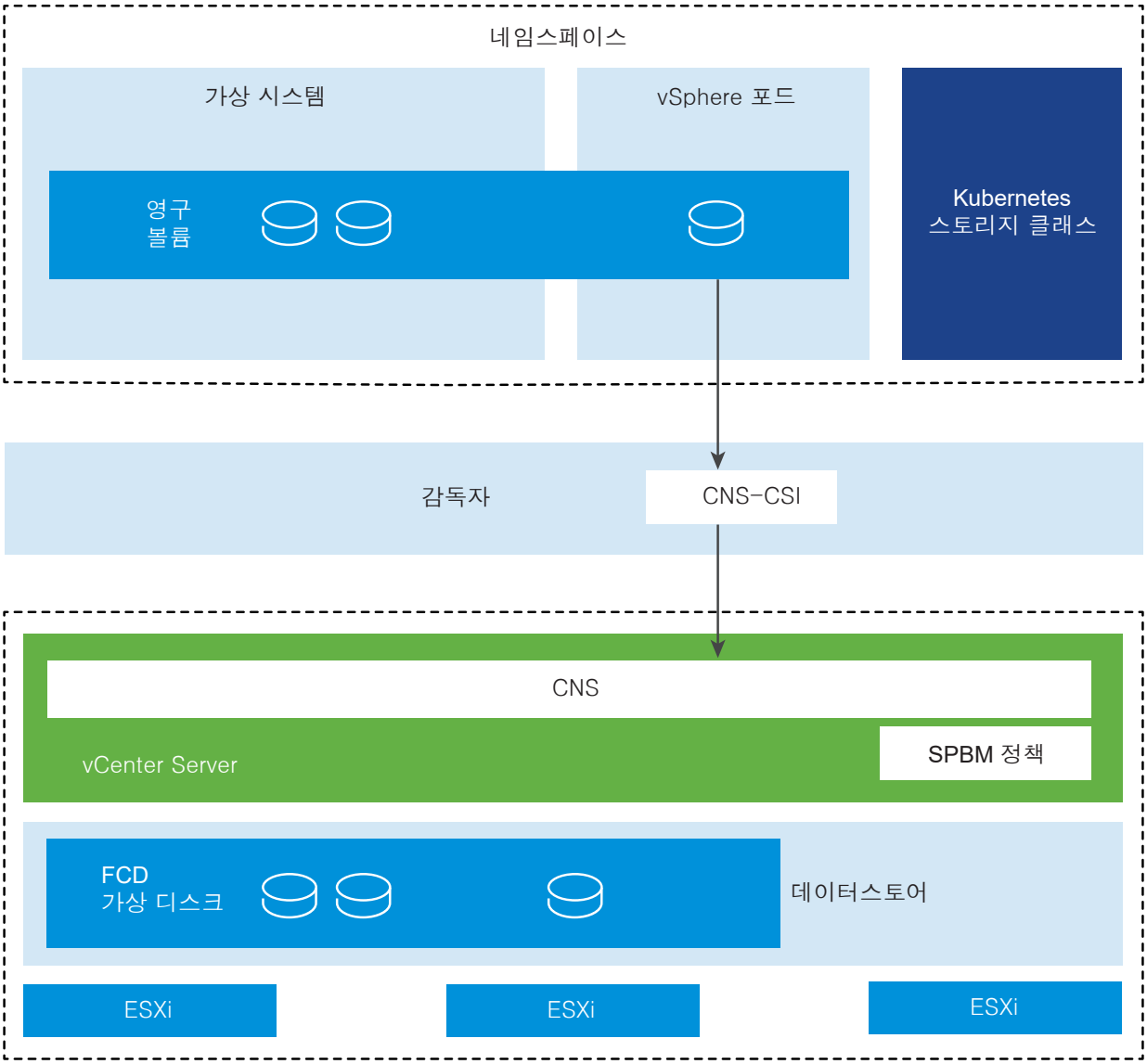
TKG 서비스 클러스터용 vSphere 네임스페이스 스토리지

vSphere CNS(클라우드 네이티브 스토리지)는 Kubernetes 워크로드에 사용할 영구 볼륨 및 백업 가상 디스크의 프로비저닝을 지원하는 스토리지 정책을 제공합니다.

CSI(Container Storage Interface)는 Kubernetes가 컨테이너용 영구 스토리지를 프로비저닝하는 데 사용하는 업계 표준입니다. 감독자는 vSphere 네임스페이스를 통해 vSphere CNS 스토리지를 Kubernetes 환경에 연결하는 CNS-CSI 드라이버를 실행합니다. vSphere CNS-CSI는 vSphere 네임스페이스의 TKG 클러스터에서 전송되는 모든 스토리지 프로비저닝 요청에 대해 CNS 제어부와 직접 통신합니다.

TKG 클러스터는 TKG 클러스터에서 시작되는 모든 스토리지 관련 요청을 담당하는 수정된 버전의 vSphere CNS-CSI 드라이버를 실행합니다. 요청은 감독자의 CNS-CSI로 전달된 다음 vCenter Server의 CNS로 전파됩니다.

이 다이어그램은 vSphere 네임스페이스, 감독자 및 TKG 클러스터 스토리지 메커니즘 간의 관계를 보여줍니다.



TKG 서비스 클러스터용 영구 스토리지 볼륨

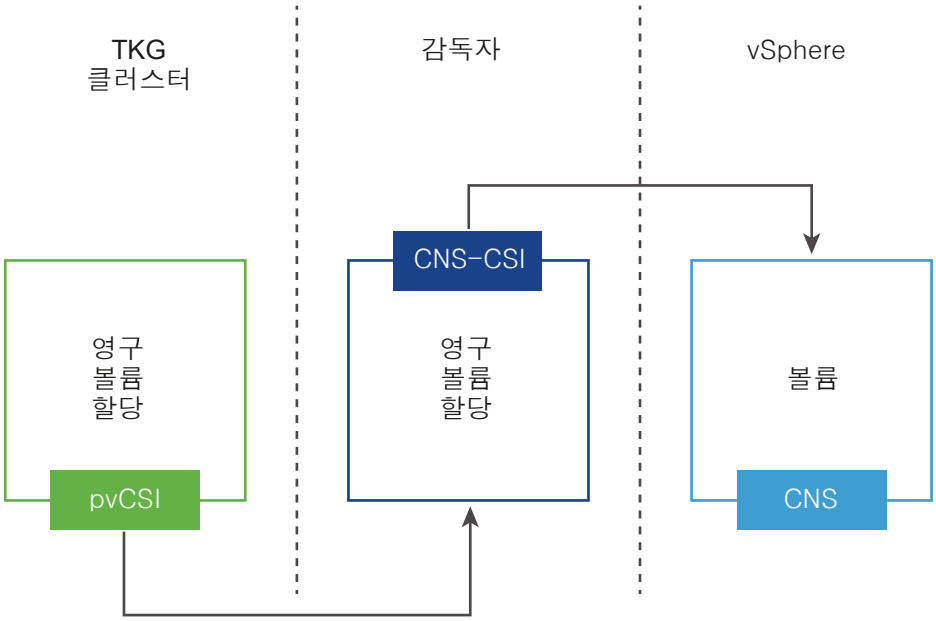
Kubernetes의 상태 저장 애플리케이션에는 영구 볼륨이 필요합니다. 영구 볼륨에 대한 자세한 내용은 Kubernetes 설명서를 참조하십시오.

vSphere 환경에서 영구 볼륨 개체는 데이터스토어에 상주하는 가상 디스크로 백업됩니다. 데이터스토어는 스토리지 정책으로 표시됩니다. vSphere 스토리지 정책을 vSphere 네임스페이스에 할당하면 스토리지 정책을 해당 네임스페이스의 각 TKG 클러스터에 대한 Kubernetes 스토리지 클래스로 사용할 수 있습니다.

TKG는 영구 볼륨의 동적 및 정적 프로비저닝을 지원합니다. 동적 프로비저닝을 사용하면 영구 볼륨을 미리 프로비저닝할 필요가 없습니다. vSphere 네임스페이스에서 사용할 수 있는 스토리지 클래스를 참조하는 PVC(영구 볼륨 할당)를 발급합니다. TKG는 백업 가상 디스크로 해당 영구 볼륨을 자동으로 프로비저닝합니다. 영구 스토리지 볼륨을 동적으로 생성의 내용을 참조하십시오.

정적 프로비저닝을 사용하면 기존 스토리지 개체를 사용하여 클러스터에서 사용할 수 있습니다. 영구 볼륨은 기존 스토리지 개체, 지원되는 구성 및 마운트 옵션의 세부 정보를 제공하여 정의합니다. 영구 스토리지 볼륨을 정적으로 생성의 내용을 참조하십시오.

이 다이어그램은 동적 영구 볼륨 프로비저닝 워크플로를 보여줍니다. TKG 클러스터에서 `kubectl`을 사용하여 PVC를 생성합니다. 이 작업은 감독자에서 일치하는 PVC를 생성하고 CNS 볼륨 생성 API를 호출하는 CNS-CSI 드라이버를 트리거합니다.



TKG 서비스 클러스터용 스토리지 클래스 버전

vSphere 네임스페이스를 구성하려면 하나 이상의 vSphere 스토리지 정책을 할당합니다. vSphere 스토리지 정책이 적용되면 Kubernetes 스토리지 클래스로 변환되고 감독자에 복제됩니다. 마찬가지로 TKG 컨트롤러는 해당 vSphere 네임스페이스에 배포된 각 TKG 클러스터에 스토리지 클래스를 복제합니다.

TKG 클러스터 측에 두 가지 버전의 스토리지 클래스가 표시됩니다. 하나는 vSphere 스토리지 정책이 생성될 때 지정된 사용자 정의 이름이 있는 버전이고 다른 하나는 이름에 `*-latebinding`이 추가된 버전입니다.

스토리지 클래스의 지연 바인딩 버전은 TKG 포드 스케줄러에서 계산 노드를 선택한 후 개발자가 영구 스토리지 볼륨에 바인딩하는 데 사용할 수 있습니다. 사용할 스토리지 클래스와 사용 시기에 대한 자세한 내용은 영구 볼륨에 스토리지 클래스 사용 항목을 참조하십시오.

```
kubectl get sc
NAME                                    PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
wcpglobal-storage-profile               csi.vsphere.vmware.com   Delete
Immediate              true                   2m43s
wcpglobal-storage-profile-latebinding   csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   2m43s
```

vSphere 네임스페이스 생성

vSphere 네임스페이스를 생성하는 방법에는 여러 가지가 있습니다.

관리자는 vSphere Client를 사용하여 vSphere 네임스페이스를 생성할 수 있습니다. TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.

vSphere 네임스페이스에 대한 **소유자** 역할 사용 권한이 부여된 vCenter Single Sign-On 사용자는 `kubectl`을 사용하여 셀프 서비스 방식으로 vSphere 네임스페이스를 생성할 수 있습니다. Kubectl을 사용하여 vSphere 네임스페이스 생성 사용의 내용을 참조하십시오.

VMware는 vSphere 네임스페이스의 수명 주기 관리를 위해 vCenter Server API를 노출하고 다음을 비롯한 SDK(소프트웨어 개발 키트)를 제공합니다.

- 네임스페이스 관리 API는 vSphere 네임스페이스를 관리하기 위한 REST 기반 리소스를 제공합니다.
- 네임스페이스 API는 vSphere 네임스페이스에서 주체의 액세스 제어를 관리하기 위한 REST 기반 리소스를 제공합니다.
- vSphere Automation SDK for Java는 vSphere 네임스페이스의 생성 및 수명 주기 관리를 자동화하기 위한 몇 가지 패키지를 제공합니다.
- 마찬가지로 vSphere Automation SDK for Python은 vSphere 네임스페이스의 생성 및 수명 주기 관리를 자동화하기 위한 몇 가지 패키지를 제공합니다.

TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성

vSphere 네임스페이스에서 TKG 서비스 클러스터를 프로비저닝합니다.

하나 이상의 TKG 서비스 클러스터를 호스팅할 vSphere 네임스페이스를 생성합니다.

절차

1. vSphere Client를 사용하여 vCenter Server에 로그인합니다.
2. **워크로드 관리 > 네임스페이스 > 새 네임스페이스**를 선택합니다.
3. 감독자를 사용하도록 설정된 vSphere 클러스터를 선택합니다.
4. vSphere 네임스페이스의 이름을 입력합니다.

   이름은 DNS 호환 형식이어야 하며 이름은 vCenter Server에서 관리하는 모든 감독자 인스턴스에서 고유해야 합니다.
5. vSphere 네임스페이스에 대한 설명을 입력합니다.

6 감독자에 사용되는 네트워크 스택 유형에 따라 **네임스페이스 네트워크**를 지정합니다.

네트워크 스택	
vSphere 네트워킹	**네트워크** 메뉴에서 워크로드 네트워크를 선택합니다.
NSX 네트워킹	감독자에서 네트워크를 상속합니다.이 경우 작업이 필요하지 않습니다. 또는 **클러스터 네트워크 설정 재정의**를 선택하고 이 vSphere 네임스페이스에 대한 네트워크를 사용자 지정합니다. vSphere 네임스페이스에 대한 워크로드 네트워크 설정 재정의의 내용을 참조하십시오.

7 **생성**을 클릭하여 vSphere 네임스페이스를 생성합니다.

감독자에 네임스페이스가 생성됩니다.

8 TKG 클러스터에 대한 vSphere 네임스페이스를 구성합니다. TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성

vSphere 네임스페이스에 하나 이상의 TKG 서비스 클러스터를 배포합니다. vSphere 네임스페이스에 적용된 구성 설정은 거기에 배포된 각 TKG 서비스 클러스터에 상속됩니다.

vSphere 네임스페이스에 대한 역할 사용 권한 구성

역할 사용 권한의 범위는 vSphere 네임스페이스입니다. TKG 클러스터 사용자 및 그룹에 할당할 수 있는 세 가지 역할 사용 권한은 **소유자**, **편집할 수 있음** 및 **볼 수 있음**입니다. 각 역할에 대한 설명은 다음 표에 있습니다. 자세한 내용은 TKG 서비스 클러스터에 대한 ID 및 액세스 관리 정보의 내용을 참조하십시오.

vCenter Single Sign-On을 사용하는 경우 세 가지 역할을 모두 사용할 수 있습니다. SSO 사용자 및 그룹을 vSphere 네임스페이스에 할당하려면 vCenter Single Sign-On 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성 항목을 참조하십시오.

외부 OIDC 제공자를 사용하는 경우 **소유자** 역할 사용 권한은 사용할 수 없습니다. OIDC 사용자 및 그룹을 vSphere 네임스페이스에 할당하려면 외부 ID 제공자 사용자 및 그룹에 대한 vSphere 네임스페이스 사용 권한 구성 항목을 참조하십시오.

역할	설명
소유자	역할 사용 권한 및 바인딩에 할당된 사용자 및 그룹은 kubectl을 사용하여 vSphere 네임스페이스 개체를 관리하고 TKG 클러스터를 운영할 수 있습니다. Kubectl을 사용하여 vSphere 네임스페이스 생성 사용의 내용을 참조하십시오.
편집할 수 있음	역할 사용 권한 및 바인딩에 할당된 사용자 및 그룹은 vSphere 네임스페이스 개체를 보고 TKG 클러스터를 운영할 수 있습니다. **편집할 수 있음** 사용 권한을 부여받은 vCenter Single Sign-On 사용자/그룹은 해당 vSphere 네임스페이스에 배포된 각 TKG 클러스터의 Kubernetes **cluster-admin** 역할에 바인딩됩니다.
볼 수 있음	역할 사용 권한 및 바인딩에 할당된 사용자 및 그룹은 vSphere 네임스페이스 개체를 볼 수 있습니다. **참고** Kubernetes에는 **볼 수 있음** 사용 권한에 바인딩할 수 있는 동등한 읽기 전용 역할이 없습니다. Kubernetes 사용자에게 클러스터 액세스 권한을 부여하려면 개발자에게 TKG 서비스 클러스터에 대한 vCenter SSO 액세스 권한 부여 항목을 참조하십시오.

vSphere 네임스페이스에 대한 영구 스토리지 구성

vSphere 네임스페이스에 하나 이상의 vSphere 스토리지 정책을 할당할 수 있습니다. 할당된 스토리지 정책은 vSphere 스토리지 환경에서 영구 볼륨의 데이터스토어 배치를 제어합니다.

일반적으로 vSphere 관리자가 vSphere 스토리지 정책을 정의합니다. vSphere 영역을 사용하는 경우에는 스토리지 정책이 `Zonal` 토폴로지로 구성되어야 합니다. TKG 서비스 클러스터에 대한 vSphere 스토리지 정책 생성의 내용을 참조하십시오.

vSphere 네임스페이스에 vSphere 스토리지 정책을 할당하려면 다음을 수행합니다.

1 **워크로드 관리 > 네임스페이스** 및 대상 vSphere 네임스페이스를 선택합니다.

2 **스토리지** 타일에서 **스토리지 추가**를 선택합니다.

3 사용 가능한 옵션 중 하나 이상의 스토리지 정책을 선택합니다.

vSphere 네임스페이스에 할당하는 각 vSphere 스토리지 정책에 대해 일치하는 Kubernetes 스토리지 클래스 두 개가 해당 vSphere 네임스페이스에 생성됩니다. 이러한 스토리지 클래스는 해당 vSphere 네임스페이스에 배포된 각 TKG 클러스터에 복제됩니다. 영구 볼륨에 스토리지 클래스 사용의 내용을 참조하십시오.

vSphere 네임스페이스에 대한 용량 및 사용량 제한 설정

vSphere 네임스페이스를 구성하면 vSphere 네임스페이스에 대한 리소스 풀이 vCenter Server에 생성됩니다. 기본적으로 이 리소스 풀은 용량 및 사용 할당량 없이 구성됩니다. 리소스는 인프라에 의해 제한됩니다.

vSphere 네임스페이스에 대한 **용량 및 사용량** 타일에서 다음과 같은 **제한**을 구성할 수 있습니다.

CPU	vSphere 네임스페이스에 대해 예약할 CPU 리소스 양입니다.
메모리	vSphere 네임스페이스에 대해 예약할 메모리 양입니다.
스토리지	vSphere 네임스페이스에 대해 예약할 총 스토리지 공간의 양입니다.
스토리지 정책 제한	vSphere 네임스페이스와 연결된 각 스토리지 정책에 개별적으로 전용 스토리지 양을 설정합니다.

일반적으로 TKG 클러스터 배포의 경우 vSphere 네임스페이스에서 리소스 할당량을 구성할 필요가 없습니다. 할당량 제한을 지정하는 경우에는 해당 위치에 배포된 TKG 클러스터에 대한 잠재적 영향을 이해하는 것이 중요합니다.

CPU 및 메모리 제한

TKG 클러스터 노드에 보장됨 VM 클래스 유형을 사용하는 경우 vSphere 네임스페이스에 구성된 CPU 및 메모리 제한은 여기에 배포된 TKG 클러스터와 관련이 없습니다. 그러나 TKG 클러스터 노드가 TKG 서비스 클러스터에서 VM 클래스 사용을 사용하는 경우 CPU 및 메모리 제한이 TKG 클러스터에 영향을 미칠 수 있습니다.

사용 시도 VM 클래스 유형을 사용하면 리소스가 오버 커밋될 수 있으므로 TKG 클러스터를 프로비저닝하는 vSphere 네임스페이스에 대해 CPU 및 메모리 제한을 설정한 경우 리소스가 부족해질 수 있습니다. 경합이 발생하고 TKG 클러스터 제어부가 영향을 받는 경우 클러스터 실행이 중지될 수 있습니다. 이러한 이유로 운영 클러스터에는 항상 TKG 서비스 클러스터에서 VM 클래스 사용을 사용해야 합니다. 모든 운영 노드에 대해 보장됨 VM 클래스 유형을 사용할 수 없다면 적어도 제어부 노드에 대해서는 보장됨 VM 클래스 유형을 사용해야 합니다.

스토리지 및 스토리지 정책 제한

vSphere 네임스페이스에 구성된 스토리지 제한에 따라 vSphere 네임스페이스에 배포된 모든 TKG 클러스터에 대해 사용할 수 있는 전체 스토리지 양이 결정됩니다.

vSphere 네임스페이스에 구성된 스토리지 정책 제한에 따라 스토리지 클래스가 복제되는 각 TKG 클러스터에 대해 해당 스토리지 클래스에 사용할 수 있는 스토리지 양이 결정됩니다.

일부 워크로드에는 최소 스토리지 요구 사항이 있습니다. 예제는 #unique_112 항목을 참조하십시오.

TKR 컨텐츠 라이브러리를 TKG 서비스와 연결

TKG 클러스터를 프로비저닝하려면 TKG 서비스를 컨텐츠 라이브러리와 연결합니다. TKr 이미지를 호스팅하기 위한 컨텐츠 라이브러리를 생성하려면 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리 항목을 참조하십시오.

TKr 컨텐츠 라이브러리를 vSphere 네임스페이스와 연결하려면 다음을 수행합니다.

1 **워크로드 관리 > 감독자 > 감독자**(감독자 인스턴스 선택)를 선택합니다.

2 **구성 > 감독자 > 일반 > Tanzu Kubernetes Grid Service**를 선택합니다.

3 **컨텐츠 라이브러리 > 편집**을 선택합니다.

4 TKR 컨텐츠 라이브러리를 선택합니다.

5 vSphere 네임스페이스로 이동하고 **관리 네임스페이스**를 선택합니다.

6 선택한 컨텐츠 라이브러리가 **Tanzu Kubernetes Grid Service** 구성 창에 나타나는지 확인합니다.

TKR 컨텐츠 라이브러리가 네임스페이스 범위가 아니라는 점을 이해하는 것이 중요합니다. 모든 vSphere 네임스페이스는 TKGS(Tanzu Kubernetes Grid 서비스)에 대해 구성된 동일한 TKR 컨텐츠 라이브러리를 사용합니다. TKGS에 대한 TKR 컨텐츠 라이브러리를 편집하면 모든 vSphere 네임스페이스에 적용됩니다.

참고 **VM 서비스** 타일에 참조된 컨텐츠 라이브러리는 TKR 컨텐츠 라이브러리가 아닌 독립형 VM에 사용하기 위한 것입니다. 이 타일에 TKR 컨텐츠 라이브러리를 추가하지 마십시오.

VM 클래스를 vSphere 네임스페이스와 연결

vSphere IaaS control plane에는 몇 가지 기본 TKG 서비스 클러스터에서 VM 클래스 사용가 제공되며 직접 생성할 수도 있습니다.

TKG 클러스터를 프로비저닝하려면 하나 이상의 TKG 서비스 클러스터에서 VM 클래스 사용를 대상 vSphere 네임스페이스와 연결합니다. 바인딩된 클래스는 해당 vSphere 네임스페이스에 배포된 TKG 클러스터 노드에서 사용할 수 있습니다.

기본 VM 클래스를 vSphere 네임스페이스와 연결하려면 vSphere Client를 사용하여 vCenter Server에 로그인하고 다음 절차를 완료합니다.

1 **워크로드 관리 > 네임스페이스** 및 대상 vSphere 네임스페이스를 선택합니다.

2 **VM 서비스** 타일에 대해 **VM 클래스 추가**를 클릭합니다.

3 추가하려는 각 VM 클래스를 선택합니다.

   a 기본 VM 클래스를 추가하려면 목록의 1페이지에서 테이블 헤더의 확인란을 선택하고 2페이지로 이동한 후 해당 페이지의 테이블 헤더에 있는 확인란을 선택합니다. 모든 클래스가 선택되었는지 확인합니다.

   b 사용자 지정 클래스를 생성하려면 **새 VM 클래스 생성**을 클릭합니다. 지침은 VM 서비스 설명서를 참조하십시오.

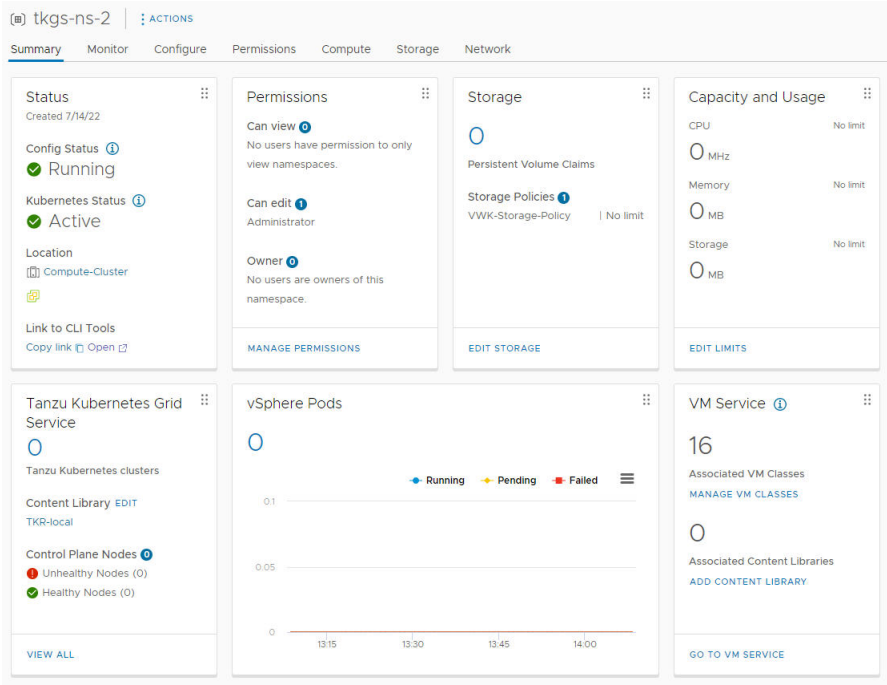
4 **확인**을 클릭하여 작업을 완료합니다.

5 클래스가 추가되었는지 확인합니다. **VM 서비스** 타일에는 **VM 클래스 관리**가 표시됩니다.

vSphere 네임스페이스 구성 확인

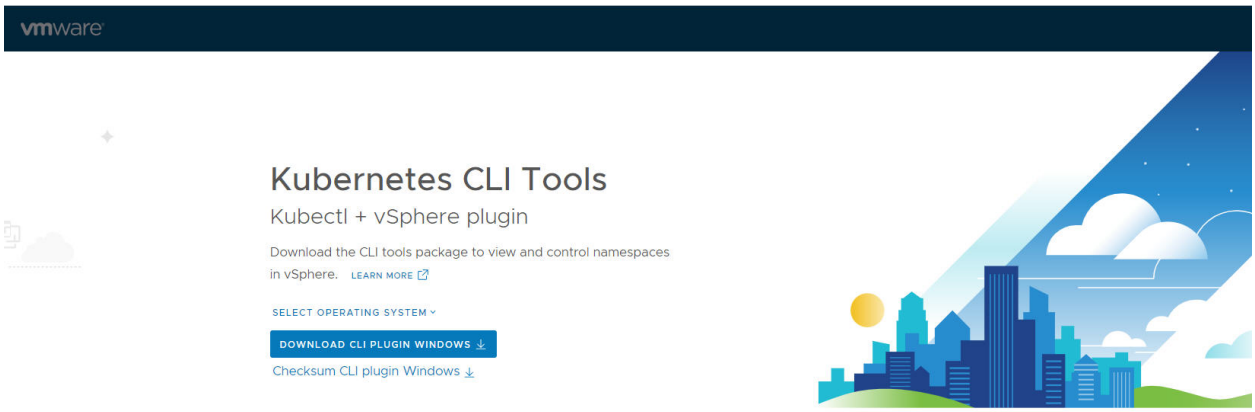
구성된 vSphere 네임스페이스에는 **상태**, **사용 권한**, **스토리지**, **용량 및 사용량**, **TKR 컨텐츠 라이브러리** 및 **VM 클래스**가 포함되어 있습니다.

그림 6-1. vSphere 네임스페이스 구성됨



상태 타일에는 vSphere IaaS control plane CLI 도구에 대한 링크가 포함되어 있습니다. DevOps 페이지는 감독자 제어부 로드 밸런서에서 제공합니다. TKG 클러스터 사용자에게 vSphere에 대한 Kubernetes CLI 도구를 다운로드할 수 있는 링크를 제공합니다. vSphere에 대한 Kubernetes CLI 도구 설치의 내용을 참조하십시오.

그림 6-2. vSphere 네임스페이스 DevOps 페이지



kubectl을 사용하여 vSphere 네임스페이스 구성을 확인하려면 TKG 서비스 클러스터를 호스팅하기 위한 vSphere 네임스페이스 준비 상태 확인 항목을 참조하십시오.

vSphere 네임스페이스에 대한 워크로드 네트워크 설정 재정의

감독자를 배포할 때 워크로드 네트워크에 대한 기본 설정을 구성할 수 있습니다. NSX 네트워킹에서 감독자를 사용하도록 설정한 경우 vSphere 네임스페이스를 생성할 때 기본 워크로드 네트워크 설정을 재정의할 수 있습니다. 재정의된 워크로드 네트워크 설정은 이 vSphere 네임스페이스 세그먼트에만 적용됩니다.

워크로드 네트워크 설정 재정의(NSX에만 해당)

vSphere 네임스페이스를 생성하면 네트워크 세그먼트가 생성됩니다. 기본적으로 이 네트워크 세그먼트는 감독자에 구성된 워크로드 네트워크에서 파생됩니다. 자세한 내용은 vSphere 네임스페이스 네트워크의 내용을 참조하십시오.

감독자가 NSX 네트워킹으로 구성된 경우에는 vSphere 네임스페이스를 생성하는 동안 vSphere 네임스페이스에 대해 **클러스터 네트워크 설정 재정의**를 선택할 수 있습니다. 이 옵션을 선택하면 수신, 송신 및 네임스페이스 네트워크 필드에 CIDR을 추가하여 vSphere 네임스페이스 네트워크를 사용자 지정할 수 있습니다. 추가한 새 CIDR은 이 vSphere 네임스페이스 인스턴스에 대한 기존 CIDR을 재정의합니다.

NSX 버전 4.1.1 이상을 구성했고 NSX에서 Enterprise 라이센스를 사용하여 NSX Advanced Load Balancer 버전 22.1.4 이상을 설치, 구성 및 등록한 경우 NSX에서 사용되는 로드 밸런서는 NSX Advanced Load Balancer입니다. 4.1.1 이전 버전의 NSX를 구성한 경우에는 NSX 로드 밸런서가 사용됩니다. 자세한 내용은 "vSphere IaaS 제어부 개념 및 계획" 에서 감독자 네트워킹을 참조하십시오.

감독자 네트워크 설정을 재정의하는 일반적인 사용 사례는 라우팅 가능한 포드 네트워킹을 사용하여 TKG 클러스터를 프로비저닝하는 것입니다. 이 작업을 수행하는 방법에 대한 자세한 내용 및 예제 링크는 표의 구성 설정을 참조하십시오.

표 6-1. vSphere 네임스페이스 네트워크 계획 시 고려 사항

고려 사항	설명
NSX 필요	특정 vSphere 네임스페이스에 대한 감독자 네트워크 설정을 재정의하려면 감독자를 NSX 네트워킹으로 구성해야 합니다.
NSX 설치	특정 vSphere 네임스페이스에 대한 감독자 네트워크 설정을 재정의하려면 NSX 설치에 Tier-0 게이트웨이(라우터) 전용 Edge 클러스터와 Tier-1 게이트웨이 전용 다른 Edge 클러스터가 포함되어야 합니다. NSX 설치 가이드 "vSphere with Tanzu 설치 및 구성" 을 참조하십시오.
IPAM 필요	특정 vSphere 네임스페이스에 대한 감독자 네트워크 설정을 재정의하는 경우 새 vSphere 네임스페이스 네트워크는 감독자 및 기타 vSphere 네임스페이스 네트워크와는 다른 고유한 수신, 송신 및 네임스페이스 네트워크 서브넷을 지정해야 합니다. 이에 따라 IP 주소 할당을 관리해야 합니다.
감독자 라우팅	감독자는 TKG 클러스터 노드 및 수신 서브넷으로 직접 라우팅할 수 있어야 합니다. vSphere 네임스페이스에 대해 Tier-0 게이트웨이를 선택할 때 필요한 라우팅을 구성하는 두 가지 옵션이 있습니다. ■ VRF(가상 라우팅 및 전달) 게이트웨이를 사용하여 감독자 Tier-0 게이트웨이의 구성을 상속 ■ BGP(Border Gateway Protocol)를 사용하여 감독자 Tier-0 게이트웨이와 전용 Tier-0 게이트웨이 간의 경로를 구성 이러한 옵션에 대한 자세한 내용은 NSX Tier-0 게이트웨이 설명서를 참조하십시오.

감독자 네트워크 설정을 재정의하기 위한 구성 필드입니다.

표 6-2. 워크로드 네트워크 설정을 재정의하기 위한 vSphere 네임스페이스 구성 옵션

구성 요소	구성
Tier-0 게이트웨이	NSX Tier-0 게이트웨이는 감독자를 물리적 네트워크에 연결합니다. 선택한 Tier-0 게이트웨이는 vSphere 네임스페이스에 대해 생성된 Tier-1 게이트웨이와 연결됩니다. 새 Tier-0 게이트웨이를 선택하면 감독자를 사용하도록 설정된 경우 구성된 Tier-0 게이트웨이가 재정의됩니다. 이 경우 새 CIDR 범위를 구성해야 합니다. Tier-0 게이트웨이에 연결된 VRF 게이트웨이를 선택하면 네트워크와 서브넷이 자동으로 구성됩니다. Tier-0 게이트웨이를 선택하고 구성을 완료한 후에는 Tier-0 게이트웨이를 변경할 수 없습니다.
로드 밸런서 크기	vSphere 네임스페이스의 Tier-1 게이트웨이에서 로드 밸런서 인스턴스의 크기를 선택합니다. 로드 밸런서의 크기를 소형(기본값), 중형 또는 대형으로 설정합니다. Edge 노드당 정해진 수의 로드 밸런서 인스턴스만 정의할 수 있습니다. 자세한 내용은 구성 최대값을 참조하십시오. **참고** 이 필드는 NSX Advanced Load Balancer에 적용할 수 없습니다.
NAT 모드	NAT 모드는 기본적으로 선택되어 있습니다. 즉, 네임스페이스 네트워크 서브넷은 라우팅할 수 없으며 네임스페이스 네트워크, 수신 및 송신 CIDR을 구성해야 합니다. NAT 모드를 선택 취소하면 네임스페이스 네트워크에 대해 라우팅 가능한 CIDR 범위를 제공할 것임을 시스템에 알립니다. NAT 모드를 선택 취소하면 Tier-0 게이트웨이 외부에서 TKG 클러스터 노드 IP 주소에 직접 액세스할 수 있으며 송신 CIDR을 구성할 필요가 없습니다. NAT 모드를 사용하지 않고 클러스터를 프로비저닝하려면 NAT 모드의 선택을 취소하고 장 7 TKG 서비스 클러스터 프로비저닝 예시를 참조하십시오.
네임스페이스 네트워크 CIDR	네임스페이스 네트워크 CIDR은 IP 풀로 작동하는 서브넷으로, 네임스페이스 서브넷 접두사는 해당 IP 풀에서 분할되는 모든 후속 CIDR 블록의 크기를 설명합니다. vSphere 네임스페이스가 생성될 때마다 네임스페이스 네트워크의 서브넷이 할당됩니다. 이 블록에서 분할된 서브넷 크기는 /24입니다. 즉, vSphere 네임스페이스당 최대 256개의 포드를 생성할 수 있습니다. 자세한 내용은 구성 최대값을 참조하십시오. 네임스페이스 네트워크 CIDR은 vSphere 네임스페이스 세그먼트에 연결된 TKG 클러스터에 IP 주소를 할당하는 데 사용됩니다. NAT 모드를 선택하면 CIDR은 라우팅할 수 없습니다. NAT 모드를 선택 취소한 경우 네임스페이스 네트워크 CIDR을 라우팅할 수 있어야 합니다.
네임스페이스 서브넷 접두사	네임스페이스 세그먼트용으로 예약된 서브넷의 크기를 지정하는 서브넷 접두사를 입력합니다. Default is 28. 네임스페이스 서브넷 접두사는 각 vSphere 네임스페이스 세그먼트에 대해 생성된 IP 서브넷을 정의합니다. 예를 들어 /24 접두사를 설정하면 거기에 배포된 TKG 클러스터에 할당할 IP 주소가 254개인 IP 서브넷이 있는 vSphere 네임스페이스 세그먼트가 생성됩니다. 추가 예: 네임스페이스 네트워크 CIDR = 192.168.1.0/24 네임스페이스 서브넷 접두사 = /28 이 경우 TKGS는 192.168.1.0/24 서브넷에서 16x 192.168.1.x/28 CIDR 블록을 제공할 수 있습니다. 이를 통해 TKG 서비스 관리 VM(TKC, VMS, vSphere 포드)이 연결된 16개의 TKGS 네임스페이스 네트워크를 인스턴스화할 수 있습니다. 예를 들어 모든 TKC는 전용 네임스페이스 CIDR을 수신하며, 이 경우 192.168.1.0/28이 될 수 있고 다음 TKC 네임스페이스 서브넷은 192.168.0.16/28이 되는 식입니다.

표 6-2. 워크로드 네트워크 설정을 재정의하기 위한 vSphere 네임스페이스 구성 옵션 (계속)

구성 요소	구성
수신	수신 IP CIDR 블록은 모든 vSphere 네임스페이스에서 서비스 유형 로드 밸런서 및 수신 컨트롤러에 의해 게시된 Kubernetes 서비스에 대한 IP 주소를 할당하는 데 사용됩니다. TKG 클러스터 서비스 및 수신은 이 CIDR 블록에서 IP 주소를 가져옵니다. TKG 클러스터용 로드 밸런서 서비스에서 게시한 가상 IP 주소의 수신 IP 범위를 결정하는 CIDR 주석을 입력합니다. **참고** 이 필드는 NSX Advanced Load Balancer에 적용할 수 없습니다.
송신	송신 IP CIDR은 외부 서비스에 액세스하기 위해 vSphere 네임스페이스를 나가는 트래픽에 대해 SNAT(소스 네트워크 주소 변환)에 대한 IP 주소를 할당하는 데 사용됩니다. SNAT IP 주소의 송신 IP 범위를 결정하는 CIDR 주석을 입력합니다.

TKG 서비스 클러스터에서 VM 클래스 사용

TKG 서비스 클러스터 노드의 크기를 조정하려면 VM(가상 시스템) 클래스를 지정합니다. 플랫폼은 기본 VM 클래스를 제공하며 직접 생성할 수도 있습니다. VM 클래스를 사용하려면 대상 vSphere 네임스페이스와 연결하고 클러스터 매니페스트에서 클래스를 참조합니다.

VM 클래스 정보

VM(가상 시스템) 클래스는 CPU 및 메모리(RAM)를 포함한 VM(가상 시스템)의 처리 능력을 위한 리소스 예약 요청입니다. 예를 들어 "guranteed-large"라는 VM 클래스 유형은 4개의 CPU와 16GB의 RAM을 예약합니다.

참고 VM 디스크 크기는 VM 클래스 정의가 아닌 OVA 템플릿에 의해 설정됩니다. Tanzu Kubernetes 릴리스의 경우 디스크 크기는 16GB입니다.

VM 클래스에는 두 가지 예약 유형(보장됨 및 사용 시도)이 있습니다. 보장됨 클래스는 구성된 리소스를 완전히 예약합니다. 즉, 주어진 클러스터에 대해 `spec.policies.resources.requests`가 `spec.hardware` 설정과 일치합니다. 사용 시도 클래스를 사용하면 리소스가 오버 커밋될 수 있습니다. 운영 워크로드의 경우 보장됨 VM 클래스 유형을 사용하는 것이 좋습니다.

경고 사용 시도 VM 클래스 유형을 사용하면 리소스가 오버 커밋될 수 있으므로 TKG 클러스터를 배포하는 vSphere 네임스페이스에 대해 제한을 설정한 경우 리소스가 부족해질 수 있습니다. 경합이 발생하고 제어부가 영향을 받는 경우 클러스터 실행이 중지될 수 있습니다. 이러한 이유로 운영 클러스터에는 보장됨 VM 클래스 유형을 사용합니다. 모든 운영 노드에 대해 보장됨 VM 클래스 유형을 사용할 수 없다면 적어도 제어부 노드에 대해서는 보장됨 VM 클래스 유형을 사용합니다.

TKG 서비스 클러스터에서 VM 클래스 사용

TKG 서비스 클러스터에서 VM 클래스를 사용하려면 클러스터가 프로비저닝된 vSphere 네임스페이스에 VM 클래스가 바인딩되어야 합니다. 이렇게 하려면 클래스를 대상 네임스페이스와 연결합니다. TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

대상 vSphere 네임스페이스에서 사용할 수 있는 VM 클래스를 나열하려면 `kubectl get virtualmachineclass` 명령을 사용합니다.

참고 이 명령에 문제가 발생하는 경우 VM 클래스 오류 문제 해결 항목을 참조하십시오.

VM 클래스 정의는 변경할 수 없습니다. 기본 VM 클래스 정의를 포함하여 모든 VM 클래스는 편집할 수 있습니다. VM 클래스를 편집해도 기존 TKG 클러스터 노드는 영향을 받지 않습니다. 새 TKG 클러스터는 수정된 클래스를 사용합니다.

경고 TKG 클러스터에서 사용 중인 VM 클래스를 편집하고 해당 클러스터를 확장하는 경우 새 노드는 편집된 클래스 정의를 사용하지만 기존 노드는 초기 클래스 정의를 사용하기 때문에 클래스 불일치가 발생합니다.

기본 VM 클래스

아래 표에는 Tanzu Kubernetes 클러스터 노드의 배포 크기로 사용되는 기본 VM 클래스 유형이 나열되어 있습니다.

리소스 오버 커밋을 방지하려면 운영 워크로드에서 보장된 클래스 유형을 사용해야 합니다. 메모리 부족 문제를 방지하려면 모든 환경(개발, 테스트 또는 운영)에서 워크로드를 배포하는 작업자 노드에 대해 소형 또는 초소형 클래스 크기를 사용하지 마십시오.

표 6-3. 기본 VM 클래스

클래스	CPU	메모리(GB)	예약된 CPU 및 메모리
guaranteed-8xlarge	32	128	예
best-effort-8xlarge	32	128	아니요
guaranteed-4xlarge	16	128	예
best-effort-4xlarge	16	128	아니요
guaranteed-2xlarge	8	64	예
best-effort-2xlarge	8	64	아니요
guaranteed-xlarge	4	32	예
best-effort-xlarge	4	32	아니요
guaranteed-large	4	16	예
best-effort-large	4	16	아니요
guaranteed-medium	2	8	예
best-effort-medium	2	8	아니요
guaranteed-small	2	4	예
best-effort-small	2	4	아니요

표 6-3. 기본 VM 클래스 (계속)

클래스	CPU	메모리(GB)	예약된 CPU 및 메모리
guaranteed-xsmall	2	2	예
best-effort-xsmall	2	2	아니요

사용자 지정 VM 클래스

vSphere IaaS control plane는 TKG 서비스 클러스터에서 사용할 사용자 지정 VM 클래스를 지원합니다. 사용자 지정 VM 클래스를 정의한 후에 대상 vSphere 네임스페이스와 연결해야 클러스터에서 사용할 수 있습니다. 세부 정보는 VM 서비스 설명서를 참조하십시오.

TKG 서비스 클러스터를 호스팅하기 위한 vSphere 네임스페이스 준비 상태 확인

vSphere 네임스페이스를 구성했으면 감독자에 로그인하여 vSphere 네임스페이스가 TKG 서비스 클러스터를 호스팅할 준비가 되었는지 확인합니다.

TKG 서비스 클러스터 프로비저닝을 위한 vSphere 네임스페이스 구성 확인

TKG 서비스 클러스터 프로비저닝을 준비할 때 이 작업을 완료하여 vSphere 네임스페이스가 올바르게 구성되었는지 확인합니다.

1 감독자에 로그인합니다.

```
kubectl vsphere login --server IP-ADDRESS-SUPERVISOR-CLUSTER --vsphere-username VCENTER-SSO-USERNAME
```

2 하나 이상의 TKG 클러스터를 프로비저닝할 대상 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE-NAME
```

3 vSphere 네임스페이스에 describe 명령을 실행합니다.

```
kubectl describe ns VSPHERE-NAMESPACE-NAME
```

이 명령은 vSphere 네임스페이스에서 사용할 수 있는 각 스토리지 클래스의 이름과 리소스 할당량을 반환합니다.

4 사용 가능한 Tanzu Kubernetes 릴리스를 나열하고 설명합니다.

```
kubectl get tanzukubernetesreleases
```

이 명령은 vSphere 네임스페이스에 대해 구성된 컨텐츠 라이브러리에 있고 라이브러리와 동기화되거나 라이브러리에 업로드된 TKR을 반환합니다.

5 사용 가능한 가상 시스템 클래스를 나열합니다.

```
kubectl get virtualmachineclass
```

이 명령은 네임스페이스와 연결된 VM 클래스를 반환합니다. 바인딩된 VM 클래스만 TKG 서비스 클러스터 노드를 프로비저닝하는 데 사용할 수 있습니다.

Kubectl을 사용하여 vSphere 네임스페이스 생성 사용

개발자가 `kubectl`을 사용하여 vSphere 네임스페이스의 수명 주기를 관리할 수 있게 vSphere 네임스페이스 서비스를 사용하도록 설정할 수 있습니다.

감독자에서 vSphere 네임스페이스 서비스를 사용하도록 설정하면 vSphere 네임스페이스의 소유자 역할에 할당된 개발자가 `kubectl create namespace <NAME>` 명령을 사용하여 자체 vSphere 네임스페이스를 생성할 수 있습니다.

vSphere 네임스페이스 서비스를 사용하도록 설정할 때 네임스페이스 템플릿을 정의하고 활성화합니다. 소유자 역할에 할당된 개발자는 템플릿을 사용하여 네임스페이스를 생성합니다.

절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리 > 감독자**를 선택하고 감독자 인스턴스를 선택합니다.

3 **구성** 탭에서 **감독자 > 일반**을 선택합니다.

4 **네임스페이스 서비스**를 선택합니다.

5 **상태** 스위치를 전환하여 기능을 사용하도록 설정합니다.

네임스페이스 템플릿 생성 페이지가 나타납니다.

6 **구성** 창에서 네임스페이스에 대한 리소스 제한을 구성합니다.

옵션	설명
CPU	네임스페이스에 대해 예약할 CPU 리소스 양입니다.
메모리	네임스페이스에 대해 예약할 메모리 양입니다.
스토리지	네임스페이스에 대해 예약할 총 스토리지 공간의 양입니다.
스토리지 정책	네임스페이스와 연결한 각 스토리지 정책에 개별적으로 전용 스토리지 양을 설정합니다.
VM 클래스	VM 클래스를 선택합니다. 여러 개를 선택하려면 Ctrl 키를 사용합니다.
컨텐츠 라이브러리	TKR 컨텐츠 라이브러리를 선택합니다.

7 **다음**을 클릭합니다.

8 **사용 권한** 창에서 DevOps 엔지니어 및 그룹을 추가하여 이들이 템플릿을 사용하여 네임스페이스를 생성할 수 있도록 합니다.

   a vsphere.local ID 소스 선택(vSphere SSO 사용)

   b 사용자 또는 그룹 선택

   c 소유자 역할 선택

9 **검토 및 확인** 창에 구성한 속성이 표시됩니다.

   속성을 검토하고 **완료**를 클릭합니다.

10 vSphere 네임스페이스 서비스가 활성인지 확인합니다.

   vSphere 네임스페이스 템플릿이 구성되었고 활성 상태입니다. 소유자 역할에 할당된 vSphere 네임스페이스 사용자/그룹은 템플릿을 사용하여 kubectl 명령 `kubectl create namespace <NAME>`으로 vSphere 네임스페이스를 생성할 수 있습니다.

vSphere 네임스페이스 제거

감독자에서 vSphere 네임스페이스를 제거할 수 있습니다. 이렇게 하기 전에 프로비저닝된 TKG 서비스 클러스터를 삭제해야 합니다.

사전 요구 사항: vSphere 네임스페이스에서 TKG 서비스 클러스터 삭제

vSphere 네임스페이스를 제거하기 전에 거기에 프로비저닝된 TKG 2.0 클러스터를 삭제해야 합니다.

참고 또한 vSphere 네임스페이스에 배포된 vSphere 포드도 제거해야 합니다.

kubectl 또는 Tanzu CLI를 사용하여 TKG 클러스터를 삭제합니다. Kubectl을 사용하여 또는 Tanzu CLI를 사용하여 TKG 클러스터 삭제의 내용을 참조하십시오.

참고 vSphere Client 또는 vCenter Server CLI를 사용하여 TKG 클러스터를 삭제하려고 시도하지 마십시오.

vSphere 네임스페이스 제거

TKG 2.0 클러스터 프로비저닝을 준비할 때 이 작업을 완료하여 vSphere 네임스페이스가 올바르게 구성되었는지 확인합니다.

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 워크로드 관리를 선택합니다.

3 **네임스페이스** 탭을 선택합니다.

   감독자에서 생성된 각 vSphere 네임스페이스가 나열됩니다.

4 제거하려는 vSphere 네임스페이스를 선택합니다.

5 **제거**를 선택합니다

시스템에서 vSphere 네임스페이스가 제거됩니다. 프로세스가 완료되는 데 시간이 걸릴 수 있습니다. 작업 창을 사용하여 진행률을 확인할 수 있습니다.

TKG 서비스 클러스터 프로비저닝

7

TKG 서비스를 사용하면 두 가지 유형의 워크로드 클러스터(Tanzu Kubernetes 클러스터 및 ClusterClass 기반 클러스터)를 프로비저닝할 수 있습니다.

다음으로 아래 항목을 읽으십시오.

- TKG 클러스터 프로비저닝 정보
- Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로
- Tanzu CLI를 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로
- Kubectl을 사용하여 TKG 클러스터 프로비저닝 테스트
- Kubectl을 사용하여 또는 Tanzu CLI를 사용하여 TKG 클러스터 삭제
- 클러스터 v1beta1 API 사용
- TanzuKubernetesCluster v1alpha3 API 사용

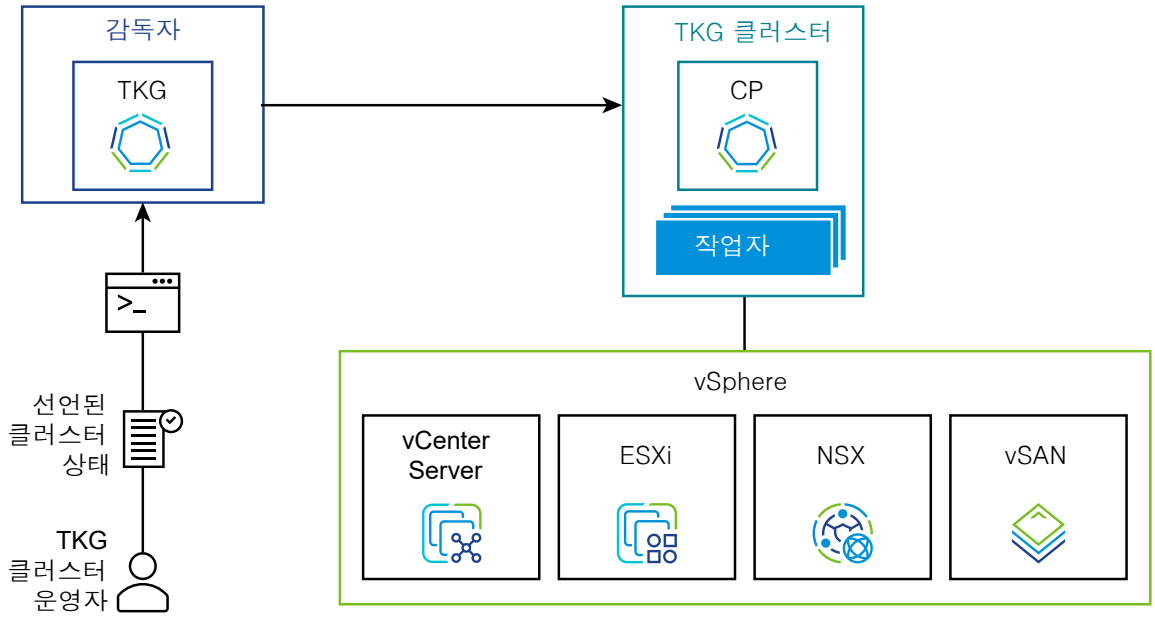
TKG 클러스터 프로비저닝 정보

TKG 서비스는 TKG 클러스터를 프로비저닝하고 수명 주기를 관리하기 위한 두 가지 API와 지원 클라이언트를 제공합니다.

TKG 클러스터 프로비저닝

이 다이어그램은 감독자에서 TKG 클러스터를 프로비저닝하기 위한 워크플로를 보여줍니다.

그림 7-1. 감독자에서 TKG 클러스터 프로비저닝



TKG 클러스터 유형

감독자가 관리 클러스터로 작동하고 Kubernetes CAPI(Cluster API)를 호스팅하는 vSphere IaaS control plane 인프라에서 프로비저닝할 수 있는 두 가지 유형의 Kubernetes 워크로드 클러스터가 있습니다. 각 유형은 ClusterClass를 기반으로 합니다. 지원되는 버전은 TKR 릴리스 정보를 참조하십시오. TKG 서비스 클러스터에서 Kubernetes 릴리스 사용 항목도 참조하십시오.

tanzukubernetescluster라는 기본 ClusterClass를 참조하는 CAPI 클러스터가 있는 TanzuKubernetesCluster

클러스터 서명:

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
```

이 유형의 워크로드 클러스터는 `tanzukubernetescluster`라는 기본 ClusterClass를 참조하는 CAPI(클러스터 API)를 사용하여 구축된 Tanzu Kubernetes 클러스터입니다. 클러스터 유형은 **TanzuKubernetesCluster**이며 프로비저닝 API는 `v1alpha3`입니다. 이것은 CAPI 클러스터 위에 추상화된 형태이므로 백엔드 클러스터 클래스에 대한 참조가 클러스터 매니페스트에 지정되지 않습니다. 참조는 시스템에서 처리됩니다.

이 유형의 워크로드 클러스터에서는 TanzuKubernetesCluster 개체가 최전방이며 추상화 계층 역할을 합니다. 이 유형의 클러스터를 프로비저닝하는 워크플로는 vSphere IaaS control plane 버전 7의 TKGS 클러스터 프로비저닝에서 변경되지 않습니다.

tanzukubernetescluster라는 기본 ClusterClass를 참조하는 CAPI 클러스터

클러스터 서명:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
```

이 유형의 워크로드 클러스터는 `tanzukubernetescluster`라는 기본 ClusterClass를 참조하는 클러스터 API를 사용하여 구축된 순수 CAPI 클러스터입니다. 클러스터 유형은 **Cluster**이며 프로비저닝 API는 `v1beta1`입니다.

이 유형의 워크로드 클러스터에서는 CAPI 클러스터 API가 최전방입니다. TKC 추상화 계층은 없습니다. 인프라를 처리하기 위한 컨트롤러가 시스템에서 제공되므로 ClusterClass 관련 개체를 생성할 필요가 없습니다. 변수가 노출되므로 클러스터를 사용자 지정할 수 있습니다. 클러스터 규격의 필드는 TKC 규격의 필드와 다르지만 프로비저닝 워크플로는 동일합니다.

TKG 클러스터 프로비저닝 API

vSphere 8 감독자의 TKG는 TKG 클러스터의 수명 주기를 관리하기 위한 두 가지 API(`v1alpha3` 및 `v1beta1`)를 제공합니다. 두 API 모두 Kubernetes API와 유사하게 본질적으로 선언적 API입니다. 선언적 클러스터 프로비저닝을 사용하여 TKG 클러스터의 원하는 상태(노드 수, 사용 가능한 스토리지, VM 크기, Kubernetes 소프트웨어 버전)를 지정합니다. TKG는 선언된 상태와 일치하는 클러스터를 프로비저닝하고 유지 보수하는 작업을 수행합니다.

기존 Tanzu Kubernetes 클러스터를 vSphere 8 감독자의 TKG로 업그레이드하는 경우 업그레이드 프로세스를 시작하기 전에 해당 클러스터가 `v1alpha2` API를 사용하고 있어야 합니다. 전체 세부 정보는 업그레이드 설명서 #unique_51에서 참조하십시오.

API	종류	vCenter 버전	설명
v1beta1	클러스터	vCenter 8 이상	클러스터 클래스를 기반으로 클러스터의 수명 주기를 관리하기 위한 새로운 API입니다.
v1alpha3	TanzuKubernetesCluster	vCenter 8 이상	v1alpha2 API의 연속입니다. API v1alpha2에서 지원되는 모든 기능은 v1alpha3 API에서 지원됩니다. 새로운 기능이 추가되었습니다.
v1alpha2	TanzuKubernetesCluster	vCenter 7 U3	vCenter 7 U3 감독자에서 Tanzu Kubernetes 클러스터를 프로비저닝하고 클러스터를 vCenter 8 감독자로 업그레이드하기 위한 레거시 API입니다. vSphere 8로 업그레이드하거나 vSphere 8에서 프로비저닝할 때 v1alpha2 API는 v1alpha3 API로 자동 변환됩니다.
v1alpha1	TanzuKubernetesCluster	vCenter 7 U1, U2	1세대 vCenter 7 감독자에서 Tanzu Kubernetes 클러스터 프로비저닝을 위한 API이며 더 이상 지원되지 않습니다.

TKG 클러스터 프로비저닝 클라이언트

vSphere 8 감독자의 TKG는 TKG 클러스터 프로비저닝을 위한 다양한 클라이언트 워크플로를 지원합니다.

- Kubernetes 스타일 선언적 클러스터 프로비저닝을 위한 Kubectl + YAML. Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.
- 대화형 명령줄 클러스터 프로비저닝을 위한 Tanzu CLI. Tanzu CLI를 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.
- 웹 기반 클러스터 프로비저닝을 위한 Tanzu Mission Control. 감독자로 호스팅되는 Tanzu Mission Control 등록의 내용을 참조하십시오.

Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로

이 워크플로에 따라 kubectl 명령 및 YAML에 정의된 클러스터 규격을 사용하여 TKG 서비스 클러스터를 선언적으로 프로비저닝합니다.

이 워크플로는 kubectl 및 YAML을 사용하여 선언적으로 TKG 클러스터 프로비저닝을 지원합니다.

사전 요구 사항

프로비저닝 워크플로를 시작하기 전에 다음 사전 요구 사항을 확인하거나 완료하십시오.

- 최신 감독자 버전으로 환경을 업데이트하거나 설치합니다. 장 2 TKG 서비스 클러스터 실행의 내용을 참조하십시오.
- 최신 Tanzu Kubernetes 릴리스로 컨텐츠 라이브러리를 생성하거나 업데이트합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.
- TKG 클러스터를 호스팅하기 위한 vSphere 네임스페이스를 생성하고 구성합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

절차

1 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

vSphere에 대한 Kubernetes CLI 도구 설치.

2 kubectl을 사용하여 감독자에서 인증합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-
username USERNAME
```

참고 감독자에 대한 FQDN은 사용하도록 설정된 경우에만 사용할 수 있습니다. 자세한 내용은 감독자 구성 설명서를 참조하십시오.

3 감독자에 성공적으로 로그인되었는지 확인합니다.

다음과 유사한 메시지가 표시됩니다.

```
Logged in successfully.

You have access to the following contexts:
   192.197.2.65
   tkg2-cluster-namespace
```

여기서 `192.197.2.65`는 감독자 컨텍스트이고 `tkg2-cluster-namespace`는 TKG 클러스터를 프로비저닝할 vSphere 네임스페이스의 컨텍스트입니다.

4 대상 vSphere 네임스페이스가 현재 컨텍스트인지 확인합니다.

```
kubectl config get-contexts
```

```
CURRENT   NAME                      CLUSTER
AUTHINFO                                        NAMESPACE
          192.197.2.65              192.197.2.65
wcp:10.197.154.65:administrator@vsphere.local
*         tkg2-cluster-namespace    10.197.154.65
wcp:10.197.154.65:administrator@vsphere.local   tkg2-cluster-namespace
```

대상 vSphere 네임스페이스가 현재 컨텍스트가 아니면 해당 컨텍스트로 전환합니다.

```
kubectl config use-context tkg2-cluster-namespace
```

5 vSphere 네임스페이스에서 사용할 수 있는 VM 클래스를 나열합니다.

```
kubectl get virtualmachineclass
```

대상 네임스페이스에 바인딩된 VM 클래스만 사용할 수 있습니다. VM 클래스가 표시되지 않으면 기본 VM 클래스가 vSphere 네임스페이스와 연결되어 있는지 확인합니다. VM 클래스 오류 문제 해결 항목도 참조하십시오.

6 사용 가능한 영구 볼륨 스토리지 클래스를 가져옵니다.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

이 명령은 `tkg2-storage-policy.storageclass.storage.k8s.io/requests.storage` 형식의 스토리지 클래스를 포함하여 네임스페이스에 대한 세부 정보를 반환합니다. 문자열의 첫 번째 토큰은 스토리지 클래스 이름입니다. 이 예시에서는 `tkg2-storage-policy`입니다. `kubectl describe storageclasses` 명령도 사용 가능한 스토리지 클래스를 반환하지만 vSphere 관리자 권한이 필요합니다.

7 사용 가능한 Tanzu Kubernetes 릴리스를 나열합니다.

다음 명령 중 하나를 사용하여 이 작업을 수행할 수 있습니다.

```
kubectl get tkr
```

```
kubectl get tanzukubernetesreleases
```

이 명령은 이 vSphere 네임스페이스에서 사용 가능한 TKR을 반환하고 배포하는 감독자와의 호환성을 보여 줍니다. 이 명령을 통해 반환된 릴리스만 사용할 수 있습니다. 릴리스 또는 원하는 릴리스가 표시되지 않으면 다음을 수행했는지 확인하십시오. a) TKR 컨텐츠 라이브러리를 생성했습니다. b) 컨텐츠 라이브러리를 원하는 OVA 파일과 동기화했습니다. c) 컨텐츠 라이브러리를 TKG 클러스터를 프로비저닝하는 vSphere 네임스페이스와 연결했습니다.

8 TKG 클러스터를 프로비저닝하기 위한 YAML 파일을 작성합니다.

a 생성할 클러스터 유형을 결정하고 해당 API 및 기능을 검토합니다.

- TanzuKubernetesCluster: TanzuKubernetesCluster v1alpha3 API 사용
- 클러스터: 클러스터 v1beta1 API 사용

b 클러스터 프로비저닝을 위한 예시 YAML 중 하나로 시작합니다.

예:

- v1alpha3 예시: 기본 TanzuKubernetesCluster
- v1beta1 예: 기본 클러스터

c YAML 파일을 `tkg2-cluster-1.yaml` 또는 이와 유사하게 저장합니다.

d 앞의 명령 출력에서 수집한 다음을 비롯한 정보를 사용하여 YAML을 채웁니다.

- 클러스터의 이름(예: `tkg2-cluster-1`)
- 대상 vSphere 네임스페이스
- 바인딩된 VM 클래스(예: `guaranteed-medium`)
- 클러스터 노드 및 영구 볼륨에 대한 스토리지 클래스
- 제어부 및 작업자 노드(복제본) 수
- TKR NAME 문자열로 지정된 Tanzu Kubernetes 릴리스(예: `v1.25.7+vmware.3-fips.1-tkg.1`)

e 필요에 맞게 TGK 클러스터 YAML을 사용자 지정합니다.

- 변동률이 높은 `containerd`와 같은 구성 요소에 대해 별도의 볼륨 추가
- 클러스터 노드 및 영구 볼륨에 대한 기본 영구 스토리지 클래스 지정
- CNI, 포드 및 서비스 CIDR을 포함한 클러스터 네트워킹 사용자 지정

f YAML 구문 검사기를 사용하여 YAML이 유효한지 확인합니다.

이 단계의 결과는 TKG 클러스터를 프로비저닝하기에 유효한 YAML입니다.

9 다음 명령을 실행하여 TKG 클러스터를 프로비저닝합니다.

```
kubectl apply -f tkg2-cluster-1.yaml
```

예상 결과:

```
tanzukubernetescluster.run.tanzu.vmware.com/tkg2-cluster-1 created
```

10 TKG 클러스터의 프로비저닝을 모니터링합니다.

```
kubectl get tanzukubernetesclusters
```

```
kubectl get tkc
```

v1beta1 API를 사용하여 클러스터를 생성하는 경우:

```
kubectl get cluster
```

처음에는 READY 상태가 False입니다. 클러스터가 프로비저닝되고 있기 때문입니다. 몇 분 후에 True가 됩니다.

```
NAME               CONTROL PLANE   WORKER   TKR NAME                              AGE
READY   TKR COMPATIBLE   UPDATES AVAILABLE
tkg2-cluster-1     3               6        v1.25.7+vmware.3-fips.1-tkg.1         49m
True    True
```

추가 명령을 실행하여 클러스터에 대한 세부 정보를 봅니다.

```
kubectl get
tanzukubernetescluster,cluster,virtualmachinesetresourcepolicy,virtualmachineservice,kubead
mcontrolplane,machinedeployment,machine,virtualmachine
```

```
kubectl describe tanzukubernetescluster tkg2-cluster-1
```

11 vSphere Client를 사용하여 클러스터 노드의 배포를 모니터링합니다.

호스트 및 클러스터에 대한 vSphere 인벤토리에서 대상 vSphere 네임스페이스에 배포된 클러스터 노드 VM을 볼 수 있습니다.

12 모든 TKG 클러스터 노드가 [준비] 상태가 되면 kubectl용 vSphere 플러그인을 사용하여 클러스터에 로그인합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN \
--vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

예:

```
kubectl vsphere login --server=192.197.2.65 --vsphere-username user@vsphere.local \
--tanzu-kubernetes-cluster-name tkg2-cluster-1 --tanzu-kubernetes-cluster-namespace tkg2-cluster-namespace
```

참고 로그인 명령은 제어부 노드가 실행 중이고 인증 서비스 플러그인이 시작된 후에만 성공합니다. 작업자 노드가 생성되는 중이면 로그인이 불규칙할 수 있습니다. 모든 클러스터 노드가 준비 상태가 되면 로그인하는 것이 좋습니다.

13 컨텍스트를 TKG 클러스터로 전환하여 현재 컨텍스트가 되도록 합니다.

TKG 클러스터에 로그인하면 다음과 유사한 메시지가 표시될 수 있습니다.

```
Logged in successfully.

You have access to the following contexts:
   192.197.2.65
   tkg2-cluster-namespace
   tkg2-cluster-1
```

여기서 `192.197.2.65`는 감독자 컨텍스트이고, `tkg2-cluster-namespace`는 vSphere 네임스페이스 컨텍스트이고, `tkg2-cluster-1`은 TKG 클러스터 컨텍스트입니다.

TKG 클러스터 컨텍스트로 전환합니다.

```
kubect config use-context tkg2-cluster-1
```

14 TKG 클러스터 리소스를 확인합니다.

```
kubectl get nodes
```

```
kubectl get namespaces
```

```
kubectl get pods -A
```

```
kubectl cluster-info
```

```
kubectl api-resources
```

15 테스트 포드를 배포하여 TKG 클러스터를 실행하고 예상대로 작동하는지 확인합니다.

Kubectl을 사용하여 TKG 클러스터 프로비저닝 테스트의 내용을 참조하십시오.

Tanzu CLI를 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로

이 워크플로에 따라 Tanzu CLI를 사용하여 v1beta1 TKG 클러스터를 프로비저닝합니다.

사전 요구 사항

프로비저닝 워크플로를 시작하기 전에 다음 사전 요구 사항을 확인하거나 완료하십시오.

- 최신 감독자 버전으로 환경을 업데이트하거나 설치합니다. 장 2 TKG 서비스 클러스터 실행의 내용을 참조하십시오.
- 최신 Tanzu Kubernetes 릴리스로 컨텐츠 라이브러리를 생성하거나 업데이트합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.
- TKG 2.0 클러스터를 호스팅하기 위한 vSphere 네임스페이스를 생성하고 구성합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

기본 TKG 클러스터 프로비저닝

Tanzu CLI를 사용하여 기본 v1beta1 클러스터를 프로비저닝하려면 다음 단계를 완료합니다. 추가 지침 또는 문제 해결에 대한 자세한 내용은 독립형 TKG 설명서에서 워크로드 클러스터 생성을 참조하십시오.

1 Tanzu CLI를 설치합니다.

TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치의 내용을 참조하십시오.

2 Tanzu CLI를 사용하여 감독자에 연결합니다.

- Tanzu CLI 및 vCenter SSO 인증을 사용하여 감독자에 연결하거나
- Tanzu CLI 및 외부 IDP를 사용하여 감독자에 연결

3 사용 가능한 TKr을 나열합니다.

```
tanzu kubernetes-release get
```

4 원하는 구성으로 클러스터 매니페스트를 생성합니다.

vSphere 8 감독자의 TKG를 사용하면 Tanzu CLI와 함께 Kubernetes 스타일 개체 규격을 사용하여 ClusterClass를 기반으로 클러스터를 생성할 수 있습니다.

a v1beta1 예: 기본 클러스터부터 시작합니다.

b 필요한 `cidrBlocks`로 `spec.clusterNetwork`를 채웁니다.

c 테이블에 나열된 예상 값으로 `spec.topology`를 채웁니다.

- TKR NAME 문자열(예: `v1.26.13---vmware.1-fips.1-tkg.3`)

- 제어부 노드 수(예: `3`)
- 각 작업자 노드 풀의 이름(예: `node-pool-1`)
- worker 노드 수(예: `3`)
- VM 클래스(예: `guaranteed-medium`)
- 스토리지 클래스(예: `tkg2-storage-policy`)

5 클러스터 매니페스트를 `cluster-default.yaml`로 저장하고 YAML 검사기를 사용하여 유효성을 검사합니다.

6 TKG 클러스터를 생성합니다.

```
tanzu cluster create -f cluster-default.yaml
```

참고 세부 정보를 출력하려면 `-v 8`을 추가합니다.

7 TKG 클러스터가 생성되었는지 확인합니다.

```
Workload cluster 'cluster-default' created
```

8 클러스터가 생성된 후 다음 명령을 실행하여 클러스터의 상태를 확인합니다.

```
tanzu cluster get cluster-default
```

9 클러스터를 나열합니다.

```
tanzu cluster list
```

10 클러스터 노드를 확인합니다.

```
tanzu cluster machinehealthcheck node get cluster-default
```

```
tanzu cluster machinehealthcheck control-plane get cluster-default
```

11 TKG 클러스터에 대한 구성 컨텍스트를 가져옵니다.

```
tanzu cluster kubeconfig get cluster-default -n tkg2-cluster-ns
```

12 클러스터에 액세스합니다.

```
kubectl config use-context tanzu-cli-cluster-default@cluster-default
```

13 테스트 포드를 배포하여 TKG 2.0 클러스터를 실행하고 예상대로 작동하는지 확인합니다.

Kubectl을 사용하여 TKG 클러스터 프로비저닝 테스트의 내용을 참조하십시오.

감독자의 사용자 지정 TKG 클러스터 프로비저닝

사용자 지정 v1beta1 클러스터(예: v1beta1 예시: Calico CNI를 사용하는 클러스터)를 프로비저닝하려면예시에 제공된 대로 모든 규격을 단일 YAML에 넣고 환경에 맞게 특정 값을 변경하고 `kubectl apply -f cluster-calico.yaml`을 실행할 수 있습니다.

Tanzu CLI를 사용하여 동일한 사용자 지정 v1beta1 클러스터를 프로비저닝하려면 클러스터를 생성하기 전에 구성 개체 `CalicoConfig` 및 `ClusterBootstrap`이 있어야 합니다.

Calico CNI를 사용하여 클러스터를 프로비저닝하려면 다음을 수행합니다.

1 각각에 원하는 클러스터 이름 및 네임스페이스를 사용하여 `CalicoConfig` 및 `ClusterBootstrap` 구성 개체에 대한 YAML을 생성합니다.

2 세 가지 구성 개체 각각에 대해 `kubectl apply -f`를 실행하거나 단일 YAML에 넣고 `kubectl apply -f`를 실행합니다.

3 구성 개체의 이름과 네임스페이스와 일치하는 이름 및 네임스페이스 및 기타 원하는 매개변수를 사용하여 클러스터 규격 `cluster-calico.yaml`을 생성합니다.

4 클러스터를 생성합니다.

```
tanzu cluster create -f cluster-calico.yaml
```

Kubectl을 사용하여 TKG 클러스터 프로비저닝 테스트

TKG 클러스터를 프로비저닝했으면 테스트 워크로드를 배포하고 클러스터 기능을 검증하는 것이 좋습니다.

테스트 애플리케이션을 배포하여 TKG 클러스터가 가동되어 실행 중인지 확인합니다.

사전 요구 사항

- TKG 클러스터를 프로비저닝합니다.
- TKG 클러스터에 연결합니다.

절차

1 TKG 클러스터를 프로비저닝합니다.

Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

2 kubectl을 사용하여 감독자에 로그인합니다.

```
kubectl vsphere login --server=<IP or FQDN> --vsphere-username <USERNAME>
```

3 TKGS 클러스터가 프로비저닝된 vSphere 네임스페이스로 구성 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

4 대상 TKG 클러스터에 로그인합니다.

```
kubectl vsphere login --server=<IP or FQDN> --vsphere-username <USERNAME> \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

5 다음 내용으로 `ping-pod.yaml` 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: ping-pod
  namespace: default
spec:
  containers:
  - image: busybox:1.34
    name: busybox
    command: ["ping", "-c"]
    args: ["1", "8.8.8.8"]
  imagePullSecrets:
  - name: regcred
  restartPolicy: Never
```

6 `regcred` 레지스트리 자격 증명을 생성합니다.

이 시나리오(busybox)에 사용되는 컨테이너 이미지는 공용 Docker Hub 레지스트리에서 끌어오기 때문에 이미지 가져오기가 제한될 수 있습니다. 이 경우 포드 규격에서 참조되는 Docker Hub 계정 및 이미지 끌어오기 암호("regcred")가 필요합니다. 이 암호를 생성하려면 개인 레지스트리 자격 증명 암호 생성의 내용을 참조하십시오.

7 필요에 따라 포드 보안을 구성합니다.

Tanzu Kubernetes 릴리스 v1.24 이하를 사용하는 경우 다음 단계를 진행하고 포드를 생성합니다.

Tanzu Kubernetes 릴리스 v1.25를 사용하는 경우 TKR 1.25 이상에 대한 PSA 구성가 사용되도록 설정됩니다. 다음 단계를 진행하고 포드를 생성할 수 있습니다. 포드 보안 위반에 대한 주의가 표시되지만 무시할 수 있습니다.

```
Warning: would violate PodSecurity "restricted:latest": allowPrivilegeEscalation != false
(container "busybox" must set securityContext.allowPrivilegeEscalation=false),
unrestricted capabilities (container "busybox" must set
securityContext.capabilities.drop=["ALL"]),
runAsNonRoot != true (pod or container "busybox" must set
securityContext.runAsNonRoot=true),
seccompProfile (pod or container "busybox" must set securityContext.seccompProfile.type to
"RuntimeDefault" or "Localhost")
```

Tanzu Kubernetes릴리스 v1.26 이상을 사용하는 경우 TKR 1.25 이상에 대한 PSA 구성이 적용됩니다. 다음 단계에 표시된 대로 포드를 생성하려고 하면 작업이 실패하고 다음 오류가 표시됩니다.

```
Error from server (Forbidden): error when creating "ping-pod.yaml": pods "ping-pod" is
forbidden:
violates PodSecurity "restricted:latest": allowPrivilegeEscalation != false
```

```
(container "busybox" must set securityContext.allowPrivilegeEscalation=false),
unrestricted capabilities (container "busybox" must set
securityContext.capabilities.drop=["ALL"]),
runAsNonRoot != true (pod or container "busybox" must set
securityContext.runAsNonRoot=true),
seccompProfile (pod or container "busybox" must set securityContext.seccompProfile.type to
"RuntimeDefault" or "Localhost")
```

이 문제를 해결하려면 포드가 생성된 `default` 네임스페이스에서 다음 명령을 실행합니다. 이렇게 하면 네임스페이스에서 TKR 1.25 이상에 대한 PSA 구성이 제거됩니다.

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=privileged
```

또는 securityContext를 포드에 직접 적용할 수 있습니다. 예:

```
...
spec:
  containers:
  - image: busybox:1.34
    name: busybox
    command: ["ping", "-c"]
    args: ["1", "8.8.8.8"]
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
      runAsNonRoot: true
      runAsUser: 1000
      seccompProfile:
        type: "RuntimeDefault"
...
```

8 YAML을 적용합니다.

```
kubectl apply -f ping-pod.yaml
```

예상 결과:

```
pod/ping-pod created
```

9 포드가 성공적으로 완료되었는지 확인합니다.

```
kubectl get pods -n default
```

```
NAME       READY   STATUS      RESTARTS   AGE
ping-pod   0/1     Completed   0          13s
```

10 포드가 DNS 서버에 대해 ping을 수행했는지 확인합니다.

```
kubectl logs ping-pod -f
```

예상 결과:

```
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=106 time=33.352 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 33.352/33.352/33.352 ms
```

11 포드를 삭제합니다.

```
kubectl delete -f ping-pod.yaml
```

12 포드가 삭제되었는지 확인합니다.

```
kubectl get pods
```

Kubectl을 사용하여 또는 Tanzu CLI를 사용하여 TKG 클러스터 삭제

`kubectl` 또는 Tanzu CLI를 사용하여 TKG 클러스터를 삭제합니다.

`kubectl` 또는 Tanzu CLI를 사용하여 Tanzu Kubernetes 클러스터를 삭제하면 Kubernetes 가비지 수집을 통해 모든 종속 리소스가 삭제됩니다.

참고 vSphere Client 또는 vCenter Server CLI를 사용하여 클러스터를 삭제하려고 시도하지 마십시오.

절차

1 감독자로 인증합니다.

2 삭제하려는 TKG가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 변경합니다.

```
kubectl config use-context CLUSTER-NAMESPACE
```

3 네임스페이스의 TKG 클러스터를 나열합니다.

```
kubectl get clusters
```

```
tanzu cluster list
```

4 다음 구문을 사용하여 TKG 클러스터를 삭제합니다.

kubectl을 사용하는 v1alpha3 클러스터:

```
kubectl delete tanzukubernetescluster --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

kubectl을 사용하는 v1beta1 클러스터:

```
kubectl delete cluster --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

Tanzu CLI를 사용하는 v1alpha3 또는 v1beta1 클러스터:

```
tanzu cluster delete --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

예제 결과:

```
tanzukubernetescluster.run.tanzu.vmware.com "tkg-cluster-1" deleted
```

5 클러스터가 삭제되었는지 확인합니다.

kubectl:

```
kubectl get clusters
```

Tanzu CLI:

```
tanzu cluster list
```

6 `kubeconfig` 파일에서 클러스터 컨텍스트를 삭제합니다.

```
kubectl config delete-context CONTEXT
```

예:

```
kubectl config get-contexts
CURRENT   NAME            CLUSTER         AUTHINFO
NAMESPACE
          192.0.2.1       192.0.2.1       wcp:192.0.2.1:administrator@vsphere.local
          tkg-cluster-1   192.0.2.6       wcp:192.0.2.6:administrator@vsphere.local
*         tkg-ns-1        192.0.2.7       wcp:192.0.2.7:administrator@vsphere.local
tkg-ns-1
```

```
kubectl config delete-context tkg-cluster-1
deleted context tkg-cluster-1 from $HOME/.kube/config
```

```
kubectl config get-contexts
CURRENT   NAME            CLUSTER        AUTHINFO
NAMESPACE
          192.0.2.1       192.0.2.1      wcp:192.0.2.1:administrator@vsphere.local
*         tkg-ns-1        192.0.2.7       wcp:192.0.2.7:administrator@vsphere.local
tkg-ns-1
```

클러스터 v1beta1 API 사용

이 섹션에는 v1beta1 API를 사용하여 클러스터를 프로비저닝하기 위한 참조 컨텐츠가 제공되며, 요구 사항에 맞는 다양한 구성 및 사용자 지정이 포함된 예시가 포함되어 있습니다.

Cluster v1beta1 API

Cluster v1beta1 API를 사용하면 기본 ClusterClass 정의를 기반으로 클러스터를 프로비저닝할 수 있습니다.

ClusterClass API v1beta1

Kubernetes Cluster API는 Kubernetes 클러스터의 선언적 프로비저닝, 업그레이드 및 운영을 제공하는 도구 모음입니다. ClusterClass는 Cluster API가 발전된 것이며 이를 통해 클러스터 집합의 수명 주기를 관리하기 위한 템플릿을 정의할 수 있습니다. TKG 서비스는 v1beta1 API를 사용하여 ClusterClass를 지원합니다.

TKG 서비스에는 `tanzukubernetescluster`라는 기본 ClusterClass 정의가 함께 제공됩니다. `tanzukubernetescluster` ClusterClass는 v1beta API를 사용하여 클러스터 생성을 위한 템플릿을 제공합니다. `tanzukubernetescluster` ClusterClass는 모든 사용자 네임스페이스에서 사용할 수 있습니다. 이 ClusterClass를 기반으로 클러스터를 생성하려면 클러스터 규격에서 참조하십시오. 지침은 v1beta 예시를 참조하십시오.

기본 ClusterClass tanzukubernetescluster

기본 `tanzukubernetescluster` ClusterClass는 변경할 수 없습니다. TKG 서비스의 각 릴리스에서 업데이트될 수 있습니다.

TKG 서비스 인스턴스와 함께 제공되는 기본 `tanzukubernetescluster` ClusterClass를 보려면 다음 단계를 완료합니다.

1 감독자에 로그인합니다.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local
```

2 컨텍스트를 TKGS 클러스터가 프로비저닝된 vSphere 네임스페이스로 전환합니다.

```
kubeclt config use-context VSPEHRE-NS
```

3 기본 `tanzukubernetescluster` ClusterClass를 가져옵니다.

```
kubectl get clusterclass tanzukubernetescluster -o yaml
```

4 필요한 경우 기본 ClusterClass의 출력을 이름이 *tkc-dcc.yaml*인 파일에 쓸 수 있습니다.

```
kubectl get clusterclass tanzukubernetescluster -o yaml > tkc-dcc.yaml
```

클러스터 사용자 지정을 위한 ClusterClass 변수

변수를 사용하여 `tanzukubernetescluster` ClusterClass를 기반으로 클러스터를 사용자 지정합니다. 변수는 이름-값 쌍을 사용하여 정의됩니다. 구문은 openAPIV3Schema를 준수해야 합니다.

v1beta1 API를 사용하여 클러스터를 프로비저닝하려면 두 가지 변수가 필요합니다.

- VM 클래스
- 스토리지 클래스

클러스터를 사용자 정의하기 위해 다음과 같은 추가 변수를 사용할 수 있습니다.

- 프록시
- TLS 인증서

- SSH 키

다음 섹션은 기본 `tanzukubernetescluster` ClusterClass에서 사용할 수 있는 모든 변수의 목록입니다.

중요 유효한 키 이름은 영숫자, 대시(예: `key-name`), 밑줄(예: `KEY_NAME`) 또는 점(예: `key.name`)으로만 구성되어야 합니다. 키 이름에는 공백을 사용할 수 없습니다.

clusterEncryptionConfigYaml

`clusterEncryptionConfigYaml` 변수를 사용하여 클러스터 암호화를 구성합니다.

clusterEncryptionConfigYaml

암호화 구성 세부 정보를 제공하는 YAML 파일인 문자열입니다.

kube-apiserver 암호화 구성 패키지를 사용하여 etcd 데이터베이스의 데이터 암호화를 구성할 수 있습니다. Kubernetes 설명서에서 Encrypting Secret Data at Rest를 참조하십시오.

```
...
    variables:
    #clusterEncryptionConfigYaml specifies the base64 encoded
    #EncryptionConfiguration YAML
    #the YAML contains a base64 encryption configuration for the cluster identity
    #the key is generated randomly
    - name: clusterEncryptionConfigYaml
      value: string which is name of the EncryptionConfiguration YAML
```

controlPlaneCertificateRotation

`controlPlaneCertificateRotation` 변수를 사용하여 TLS 인증서가 만료되기 전에 이러한 인증서의 롤아웃을 트리거하여 제어부 노드에 대한 TLS 인증서를 순환하도록 시스템을 구성합니다. 모든 새 제어부 노드와 기존 제어부 노드에 대해 제어부 인증서 순환이 가능합니다.

controlPlaneCertificateRotation

기능 활성화를 위한 부울 및 인증서 순환을 위한 만료 전 일 수입니다. 자세한 내용은 Kubeadm 제어부 제공자를 사용하여 자동으로 인증서 순환을 참조하십시오.

```
...
    variables:
    - name: controlPlaneCertificateRotation
      value:
        activate: true
        daysBefore: 90
```

형식 설명:

- `activate`는 기능을 활성화하기 위한 부울입니다. 기본값은 `true`입니다.
- `daysBefore`는 만료되기 전의 기간(일 수)입니다. 기본값은 `90`일입니다. 최소값은 만료 `7`일 전입니다.

참고 이 변수는 vSphere 8 업데이트 3(TKG 서비스 3.0)에 대해 업데이트되었습니다.

controlPlaneVolumes

`controlPlaneVolumes` 변수를 사용하여 제어부 노드에 대한 영구 볼륨을 구성합니다.

controlPlaneVolumes

개체의 선택적 어레이이며, 여기에는 `name`, `storageClass` 및 `mountPath`(각각 문자열임)와 `storage` 문자열을 포함하는 선택적 `capacity` 개체가 포함됩니다.

```
...
    variables:
      #controlPlaneVolumes is an optional set of PVCs to create and
      #attach to each node
      - name: controlPlaneVolumes
        value:
          #name of the PVC to be used as the suffix (node.name)
          - name: NAME
            #mountPath is the directory where the volume device is mounted
            #takes the form /dir/path
            mountPath: /dir/path
            #storageClass is the storage class to use for the PVC
            storageClass: tkgs-storage-profile
            #capacity is the PVC storage capacity
            capacity:
              #storage sets the capacity for the disk volume
              #if not specified defaults to storageClass capacity
              storage: 4Gi
```

defaultRegistrySecret

`defaultRegistrySecret` 변수는 클러스터에 대한 기본 컨테이너 레지스트리를 구성합니다.

참고 이 변수는 내장된 Harbor 레지스트리에서 사용하도록 예약되어 있습니다.

defaultRegistrySecret

기본 컨테이너 레지스트리에 대한 공용 키, 인증서 이름 및 네임스페이스가 포함된 개체입니다.

감독자에서 내장된 Harbor 레지스트리를 사용하도록 설정하면 `defaultRegistrySecret` 변수는 클러스터가 신뢰하는 레지스트리 서비스 인증서를 지정합니다. 인증서 암호에는 `managed-by: vmware-vRegistry` 라는 레이블이 지정됩니다. 클러스터 생성 시 defaultRegistry 인증서가 `defaultRegistrySecret` 변수에 삽입됩니다. 클러스터를 생성한 후 변수를 수동으로 업데이트하여 인증서 순환 또는 업데이트를 관리합니다.

```
...
    variables:
    - name: defaultRegistrySecret
      value:
        #data holds the base64 encoded data.ca\.crt content
        #data.ca\.crt is already encoded, so raw cert data is encoded twice
        data: LS0tLS1CRUdJTiBDRVJU...S0tRU5EIENFUlRJRklDQVRFL
        #name specifies the name of the registry cert secret
        name: harbor-ca-key-pair
```

```
        #namespace specifies the ns of the registry cert secret
        namespace: svc-harbor-domain-c9
```

defaultStorageClass

`defaultStorageClass` 변수를 사용하여 클러스터에 대한 기본 스토리지 클래스를 구성합니다.

defaultStorageClass

기본 스토리지 클래스로 사용할 스토리지 클래스를 식별하는 문자열이며 Helm 차트 및 Tanzu 패키지와 같은 특정 애플리케이션에서 종종 필요합니다.

```
...
    variables:
    - name: defaultStorageClass
      value: tkg2-storage-profile
```

extensionCert

`extensionCert` 변수를 사용하여 TLS 인증서를 구성합니다.

extensionCert

`name` 및 `key` 문자열을 포함하는 `contentSecret` 개체를 포함하는 개체입니다. `contentSecret`은 TLS 인증서에 대해 생성된 Kubernetes 암호 개체를 참조합니다.

```
...
    variables:
    #extensionCert specifies the cert and key for Extensions Controller
    #self-signed issuer and certificates must be created in advance
    - name: extensionCert
      value:
        contentSecret:
          #name specifies the name of secret
          name: string
          #key specifies the content of tls\.crt in the secret's data map
          key: string
```

kubeAPIServerFQDNs

`kubeAPIServerFQDNs` 변수를 사용하여 FQDN으로 클러스터를 구성합니다.

kubeAPIServerFQDNs

하나 이상의 FQDN(정규화된 도메인 이름) 어레이입니다.

생성된 Kubernetes API 인증서에는 `kubeAPIServerFQDNs` 변수에 지정한 각 FQDN이 포함됩니다. 시스템에서 kubeconfig를 목록의 첫 번째 FQDN으로 채우고 확인 가능한 것으로 가정합니다. 목록에서 다른 FQDN을 사용하려면 변수 목록에서 원하는 FQDN을 사용하여 생성된 kubeconfig 파일을 수동으로 편집할 수 있습니다.

자세한 내용은 v1beta1 예: FQDN이 있는 클러스터 항목을 참조하십시오.

nodePoolLabels

`nodePoolLabels` 변수를 사용하여 작업자 노드에 대한 레이블을 구성합니다.

nodePoolLabels

하나 이상의 개체 어레이이며, 각 개체에는 키/값(둘 다 문자열) 쌍이 포함됩니다.

레이블을 사용하여 요구 사항에 따라 시스템 개체를 구성하면 쉽게 쿼리하고 보고할 수 있습니다. 사용 세부 정보는 Kubernetes 레이블 설명서를 참조하십시오.

nodePoolTaints

`nodePoolTaints` 변수를 사용하여 작업자 노드에 taint를 적용합니다.

nodePoolTaints

개체 어레이이며, 각 개체에는 작업자 노드에 적용되는 taint가 포함됩니다.

각 taint 개체에는 `key`(문자열), `value`(문자열) 및 `effect`(문자열)가 포함됩니다. 시스템은 생성 또는 업데이트 시 `timeAdded` 필드를 채웁니다.

nodePoolVolumes

`nodePoolVolumes` 변수를 사용하여 클러스터 노드에 대한 영구 볼륨을 지정합니다.

nodePoolVolumes

개체의 선택적 어레이이며, 여기에는 `name`, `storageClass` 및 `mountPath`(각각 문자열임)와 `storage` 문자열을 포함하는 선택적 `capacity` 개체가 포함됩니다.

```
...
    variables:
      #nodePoolVolumes is an optional set of PVCs to create and
      #attach to each node; use for high-churn components like containerd
      - name: nodePoolVolumes
        value: |
          #name of the PVC to be used as the suffix (node.name)
          - name: etcd
            #mountPath is the directory where the volume device is mounted
            #takes the form /dir/path
            mountPath: /var/lib/containerd
            #storageClass is the storage class to use for the PVC
            storageClass: tkgs-storage-profile
            #capacity is the PVC storage capacity
            capacity:
              #storage sets the capacity for the disk volume
```

```
                #if not specified defaults to storageClass capacity
                storage: 4Gi
```

ntp

`ntp` 변수를 사용하여 클러스터에 대한 NTP 서버를 구성합니다.

ntp

NTP 서버의 FQDN 또는 IP 주소인 문자열입니다.

NTP 변수는 예에 표시된 것처럼 NTP 서버의 도메인 이름을 지정합니다. NTP 서버는 클러스터 생성 시 클러스터 변수에 주입됩니다. 클러스터를 생성한 후 클러스터 변수를 수동으로 업데이트하여 서버 이름 순환 또는 업데이트를 관리합니다.

```
...
    variables:
    - name: ntp
      value: time1.vmware.com
```

podSecurityStandard

클러스터 전체 포드 보안을 구성하려면 `podSecurityStandard` 변수를 사용합니다.

참고 이 변수는 vSphere 8 업데이트 3(TKG 서비스 3.0)의 새로운 변수입니다.

podSecurityStandard

TKr v1.26 이상에서는 기본적으로 포드 보안(PSA) 제한이 주석 레이블을 사용하여 네임스페이스 수준에서 적용됩니다. TKR 1.25 이상에 대한 PSA 구성의 내용을 참조하십시오.

또는 v1beta1 클러스터를 프로비저닝하거나 업데이트할 때 `podSecurityStandard` 변수를 사용하여 클러스터 전체 PSA를 구성할 수 있습니다.

`podSecurityStandard` 변수는 다음과 같이 구현할 수 있습니다.

```
...
variables:
- name: podSecurityStandard
  value:
    deactivated: DEACTIVATED
    audit: AUDIT-PROFILE
    enforce: ENFORCE-PROFILE
    warn: WARN-PROFILE
    auditVersion: AUDIT-VERSION
    enforceVersion: ENFORCE-VERSION
    warnVersion: WARN-VERSION
    exemptions:
      namespaces: [EXEMPT-NS]
```

형식 설명:

- 클러스터 전체 PSA를 적용하려면 DEACTIVATED 값이 `false`(기본값)이고 그렇지 않으면 `true`입니다.
- *-PROFILE 값은 각 모드에 대한 PSA 프로파일이며, `"privileged"`, `"baseline"` 또는 `"restricted"`(기본값)일 수 있습니다.
- *-VERSION 값은 각 모드의 Kubernetes 버전(예: `"v1.26"`)입니다. `"latest"` 값이 기본값입니다.
- EXEMPT-NS 값은 PSA 제어에서 제외할 네임스페이스의 쉼표로 구분된 목록입니다.

참고 kube-system, tkg-system 및 vmware-system-cloud-provider를 포함한 시스템 네임스페이스는 포드 보안에서 제외됩니다.

`podSecurityStandard` 변수를 구현하지 않으면 기본 PSA 동작이 유지됩니다. 클러스터 규격에 `podSecurityStandard` 변수를 포함하면 재정의하지 않는 한 변수 설정이 기본값을 포함하여 제어됩니다.

다음 예에서는 기본값을 보여줍니다.

```
...
    variables:
      - name: podSecurityStandard
        value:
          enforce: "restricted"
          enforce-version: "latest"
```

다음 예에서는 현재 포드 강화 모범 사례를 따르지 않지만 알려진 권한 에스컬레이션을 방지하는 최소한의 제한 정책("baseline")만 적용하는 워크로드를 식별하기 위한 감사 로그 및 주의를 제공합니다.

```
...
    variables:
      - name: podSecurityStandard
        value:
          audit: "restricted"
          warn: "restricted"
          enforce: "baseline"
```

다음 예에서는 특정 네임스페이스를 제외하고 제한된 정책을 적용합니다.

```
...
    variables:
      - name: podSecurityStandard
        value:
          audit: "restricted"
          warn: "restricted"
          enforce: "restricted"
          exemptions:
            namesaces: ["privileged-workload-ns"]
```

다음 예에서는 적용을 특정 TKr 버전으로 제한합니다.

```
...
variables:
  - name: podSecurityStandard
    value:
      audit-version: "v1.26"
      warn-version: "v1.26"
      enforce-version: "v1.26"
```

더 많은 예를 보려면 Kubernetes 설명서에서 포드 보안 표준을 참조하십시오.

proxy

`proxy` 변수를 사용하여 클러스터에 대한 프록시 서버를 구성합니다.

proxy

아웃바운드 클러스터 연결을 위해 프록시 서버를 참조하는 매개 변수가 있는 개체입니다.

필수 `proxy` 매개 변수는 `httpProxy`, `httpsProxy`, `noProxy`입니다. 클러스터 정의에 `proxy` 변수를 포함하는 경우 세 필드가 모두 필요합니다.

`httpProxy` 및 `httpsProxy` 필드에는 TKG 클러스터에서 아웃바운드 HTTP 및 HTTPS 연결을 관리하도록 구성된 프록시 서버의 URI를 참조하는 문자열 값이 사용됩니다. HTTP를 사용하여 프록시 서버에 연결할 수 있습니다. HTTPS 연결은 지원되지 않습니다.

`noProxy` 필드는 문자열 어레이입니다. `noProxy` 값은 감독자 워크로드 네트워크에서 가져옵니다. `noProxy` 필드에 네임스페이스 네트워크, 수신 및 송신 서브넷을 포함해야 합니다.

`noProxy` 필드에 서비스 서브넷을 포함할 필요가 없습니다. TKG 클러스터는 이 서브넷과 상호 작용하지 않습니다.

`clusterNetwork.services.cidrBlocks` 및 `clusterNetwork.pods.cidrBlocks`는 `noProxy` 필드에 포함할 필요가 없습니다. 이러한 끝점은 자동으로 프록시로 지정되지 않습니다.

localhost 및 `127.0.0.1`은 `noProxy` 필드에 포함할 필요가 없습니다. 이러한 끝점은 자동으로 프록시로 지정되지 않습니다.

```
...
    variables:
    #proxy specifies a proxy server to be used for the cluster
    #if omitted no proxy is configured
    - name: proxy
      value:
        #httpProxy is the proxy URI for HTTP connections
        #to endpoints outside the cluster
        httpProxy: http://<user>:<pwd>@<ip>:<port>
        #httpsProxy is the proxy URL for HTTPS connections
        #to endpoints outside the cluster
        httpsProxy: http://<user>:<pwd>@<ip>:<port>
        #noProxy is the list of destination domain names, domains,
        #IP addresses, and other network CIDRs to exclude from proxying
```

```
        #must include Supervisor Pod, Egress, Ingress CIDRs
        noProxy: [array of strings, comma-separated]
```

storageClass

`storageClass` 변수를 사용하여 클러스터에 대한 스토리지 클래스를 구성합니다.

storageClass

TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에 할당된 vSphere 스토리지 프로파일의 이름인 문자열입니다.

```
...
    variables:
    - name: storageClass
      value: tkgs-storage-profile
```

다음 명령을 사용하여 사용 가능한 스토리지 클래스를 나열합니다.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

또는 vSphere 관리자 권한이 있는 경우 다음 명령을 사용합니다.

```
kubectl describe storageclasses
```

storageClasses

`storageClasses` 변수를 사용하여 클러스터에 대한 스토리지 클래스 어레이를 구성합니다.

storageClasses

하나 이상의 문자열 어레이이며,각 문자열은 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에 할당된 vSphere 스토리지 프로파일의 이름입니다.

```
...
    variables:
    - name: storageClasses
      value: [tkg2-storage-profile, tkg2-storage-profile-latebinding]
```

다음 명령을 사용하여 사용 가능한 스토리지 클래스를 나열합니다.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

또는 vSphere 관리자 권한이 있는 경우 다음 명령을 사용합니다.

```
kubectl describe storageclasses
```

TKR_DATA

`TKR_DATA` 변수를 사용하여 TKR 정보를 지정합니다.

TKR_DATA

TKR 버전 및 기타 세부 정보를 지정하는 데 사용하는 개체입니다.

`version`은 클러스터 노드에서 사용할 TKR NAME 형식의 문자열입니다.

`legacy-tkr` 레이블이 없는 TKR만 vSphere 9 감독자의 TKG와 호환됩니다. TKG 서비스 클러스터에서 Kubernetes 릴리스 사용의 내용을 참조하십시오.

기본 운영 체제는 PhotonOS입니다. 주석을 사용하여 v1beta1 예시: Ubuntu TKR이 있는 클러스터을 지정합니다.

trust

`trust` 변수를 사용하여 클러스터에 대해 하나 이상의 신뢰할 수 있는 CA 인증서를 지정합니다.

trust

클러스터에 TLS 인증서(추가 CA 또는 최종 인증서)를 추가하기 위한 개체입니다.

값은 문자열 어레이를 포함하는 `additionalTrustedCAs`입니다. 예:

```
#trust-example
    variables:
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
          - name: additional-ca-2
          - name: additional-ca-N
```

각 문자열의 값은 이중 base64로 인코딩된 PEM 형식의 CA 인증서가 포함된 Kubernetes 암호의 데이터 맵 필드에 대한 사용자 정의 이름입니다. 예:

```
#secret-example
apiVersion: v1
data:
  additional-ca-1: TFMwdExTMUNSGlSzZ3Jaa...VVNVWkpRMEMwdExTMHRDZz09
kind: Secret
metadata:
  name: cluster01-user-trusted-ca-secret
  namespace: tkgs-cluster-ns
type: Opaque
```

신뢰 변수의 일반적인 사용 사례는 v1beta1 클러스터를 개인 컨테이너 레지스트리와 통합하는 것입니다. TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합의 내용을 참조하십시오.

user

`user` 변수를 사용하여 클러스터 사용자 자격 증명을 지정합니다.

user

이름 및 키 문자열과 sshAuthorizedKey 문자열이 있는 passwordSecret 개체를 포함하는 개체입니다. 이 변수를 사용하여 원격 SSH 액세스를 위해 사용자의 SSH 키를 클러스터 노드에 추가할 수 있습니다.

사용자 변수는 암호 및 인증 키를 포함한 SSH 로그인 자격 증명을 지정합니다. 기본적으로 사용자 이름은 `vmware-system-user`입니다. 암호를 해시하고 클러스터가 프로비저닝된 동일한 네임스페이스의 암호에 저장해야 합니다. `passwordSecret` 개체는 이 암호를 참조합니다. 예를 들어 Linux에서는 `mkpasswd --method=SHA-512 --rounds=4096`을 사용하여 안전한 해시를 생성할 수 있습니다. 자세한 내용은 사용자 및 그룹 포함을 참조하십시오.

```
...
    variables:
    #user specifies an authorized user and credentials
    - name: user
      value:
        #passwordSecret is an object that contains a Kubernetes secret and key
        passwordSecret:
          #name specifies the secret name
          name: string
          #key specifies the key value pair in the secret's data map
          key: string
        sshAuthorizedKey: string that is the base64-encoded public key
```

vmClass

`vmClass` 변수를 사용하여 클러스터 노드에 대한 VM 클래스를 구성합니다.

vmClass

TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에 바인딩된 VM 클래스의 이름에 매핑되는 필수 문자열입니다.

`vmClass`는 클러스터 노드에 사용할 가상 하드웨어 설정을 설명하는 VirtualMachineClass의 이름입니다. VirtualMachineClass는 노드에서 사용할 수 있는 CPU 및 메모리와 해당 리소스에 대한 요청 및 제한을 제어합니다. TKG 서비스 클러스터에서 VM 클래스 사용의 내용을 참조하십시오.

TKG 서비스 클러스터가 프로비저닝되는 vSphere 네임스페이스와 연결된 VM 클래스만 사용할 수 있습니다. `kubectl get virtualmachineclass` 명령을 사용하여 바인딩된 클래스를 나열합니다.

서로 다른 범위에서 `vmClass` 변수를 정의하여 제어부 노드 및 노드 풀 작업자 노드에 대해 서로 다른 VM 클래스를 사용할 수 있습니다.

예를 들어, 여기에서 인라인 `vmClass` 변수는 이 특정 machineDeployment 토폴로지에 대한 기본 `vmClass` 변수를 `overrides`합니다.

```
...
    workers:
      machineDeployments:
      - class: tkg-worker
        name: compute
        replicas: 3
        variables:
          overrides:
```

```
      - name: vmClass
        value: guaranteed-large
```

v1beta1 예: 기본 클러스터

기본 설정으로 v1beta1 클러스터를 프로비저닝하려면 이 예시를 참조하십시오.

v1beta1 예: 기본 클러스터

다음 예시 YAML은 v1beta1 API를 사용하여 기본 ClusterClass를 기반으로 기본 클러스터를 생성합니다.

이 예시는 v1beta1 API를 사용하여 클러스터를 생성하는 데 필요한 최소 구성을 나타냅니다. 이 예시에는 각 필드에 대한 설명이 주석으로 표시되어 있습니다. 자세한 내용은 소스 코드를 참조하십시오.

이 예시에서는 다음 사항에 유의하십시오.

- TanzuKubernetesCluster v1alpha3 API와 달리 v1beta1 API를 사용하려면 `clusterNetwork`를 지정해야 합니다. 클러스터 유형에 대한 기본 네트워크 설정은 없습니다.
- 기본 ClusterClass는 `tanzukubernetescluster`이며, Cluster v1beta1 API 에 설명되어 있습니다.
- 각 이름-값 쌍인 `variables`를 사용하여 클러스터를 사용자 지정할 수 있습니다. 예시와 같이 최소한 VM 및 스토리지 클래스를 변수로 지정해야 합니다.
- 기술적으로 선택 사항이지만 이 예시에는 `defaultStorageClass` 변수도 포함되어 있습니다. Tanzu 패키지 및 Helm 차트를 비롯한 많은 워크로드에서 클러스터에 기본 스토리지 클래스를 프로비저닝해야 하기 때문입니다.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
#define the cluster
metadata:
  #user-defined name of the cluster; string
  name: cluster-default
  #kubernetes namespace for the cluster; string
  namespace: tkg-cluster-ns
#define the desired state of cluster
spec:
  #specify the cluster network; required, there is no default
  clusterNetwork:
    #network ranges from which service VIPs are allocated
    services:
      #ranges of network addresses; string array
      #CAUTION: must not overlap with Supervisor
      cidrBlocks: ["198.51.100.0/12"]
    #network ranges from which Pod networks are allocated
    pods:
      #ranges of network addresses; string array
      #CAUTION: must not overlap with Supervisor
      cidrBlocks: ["192.0.2.0/16"]
    #domain name for services; string
    serviceDomain: "cluster.local"
  #specify the topology for the cluster
```

```
  topology:
    #name of the ClusterClass object to derive the topology
    class: tanzukubernetescluster
    #kubernetes version of the cluster; format is TKR NAME
    version: v1.26.13---vmware.1-fips.1-tkg.3
    #describe the cluster control plane
    controlPlane:
      #number of control plane nodes
      #integer value 1 or 3
      #NOTE: Production clusters require 3 control plane nodes
      replicas: 3
    #describe the cluster worker nodes
    workers:
      #specifies parameters for a set of worker nodes in the topology
      machineDeployments:
        #node pool class used to create the set of worker nodes
        - class: node-pool
          #user-defined name of the node pool; string
          name: node-pool-1
          #number of worker nodes in this pool; integer 0 or more
          replicas: 3
    #customize the cluster
    variables:
      #virtual machine class type and size for cluster nodes
      - name: vmClass
        value: guaranteed-medium
      #persistent storage class for cluster nodes
      - name: storageClass
        value: tkg-storage-policy
      # default storageclass for control plane and worker node pools
      - name: defaultStorageClass
        value: tkg-storage-policy
```

v1beta1 예시: 기본 ClusterClass를 기반으로 하는 사용자 지정 클러스터

사용자 지정 설정으로 v1beta1 클러스터를 프로비저닝하려면 이 예시를 참조하십시오.

v1beta1 예시: 기본 ClusterClass를 기반으로 하는 사용자 지정 클러스터

다음 예시 YAML은 v1beta1 API를 사용하여 변수를 사용하는 여러 사용자 지정 설정으로 클러스터를 프로비저닝하는 방법을 보여줍니다. 이 예시는 v1beta1 예: 기본 클러스터에 기반합니다.

이 예시에서는 containerd 및 kubelet과 같은 변동률이 높은 구성 요소에 대해 작업자 노드에서 영구 볼륨에 대한 변수를 사용합니다. 또한 `vmClass` 변수는 두 번 선언됩니다. `workers.machineDeployments`에 선언된 `vmClass` 변수는 작업자 노드가 더 큰 VM 클래스로 프로비저닝되도록 전역적으로 선언된 `vmClass` 변수를 덮어씁니다.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-custom
  namespace: tkg-cluster-ns
spec:
```

```
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.2.0.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
          variables:
            overrides:
            - name: vmClass
              value: guaranteed-xlarge
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg-storage-profile
      - name: defaultStorageClass
        value: tkg-storage-profile
      - name: nodePoolVolumes
        value:
          - name: containerd
            capacity:
              storage: 50Gi
            mountPath: /var/lib/containerd
            storageClass: tkg-storage-profile
          - name: kubelet
            capacity:
              storage: 50Gi
            mountPath: /var/lib/kubelet
            storageClass: tkg-storage-profile
```

v1beta1 예시: Calico CNI를 사용하는 클러스터

기본 Antrea CNI 대신 Calico CNI를 사용하여 v1beta1 클러스터를 프로비저닝하려면 이 예시를 참조하십시오. 이 예시에서는 클러스터에 대해 하나 이상의 TKR 패키지를 사용자 지정하는 방법에 대한 지침도 참조할 수도 있습니다.

v1beta1 예시: 사용자 지정 CNI를 사용하는 클러스터

다음 예시 YAML은 v1beta1 API를 사용하여 사용자 지정 CNI를 사용하는 클러스터를 프로비저닝하는 방법을 보여줍니다. 이 예시는 v1beta1 예: 기본 클러스터에 기반합니다.

`tanzukubernetescluster` ClusterClass에 정의된 대로 기본 CNI는 Antrea입니다. 지원되는 다른 CNI는 Calico입니다. CNI를 Antrea에서 Calico로 변경하려면 `CalicoConfig` 사용자 지정 리소스를 참조하는 `ClusterBootstrap` 사용자 지정 리소스를 생성하여 기본 CNI를 오버로드합니다.

`ClusterBootstrap` 사용자 지정 리소스에는 TKR에서 가져온 값과 함께 `spec.cni.refName` 블록이 포함됩니다. (이 필드에 대한 값을 가져오는 방법에 대한 지침은 TKr 패키지 항목을 참조하십시오.) `ClusterBootstrap` 값은 ClusterClass의 기본값을 덮어쓰며 클러스터가 생성될 때 CAPI(Cluster API)에 의해 선택됩니다. `ClusterBootstrap` 사용자 지정 리소스의 이름은 `Cluster`와 동일해야 합니다.

참고 예시는 단일 YAML 파일로 제공되지만 개별 파일로 분리할 수 있습니다. 이렇게 하려면 먼저 `CalicoConfig` 사용자 지정 리소스를 생성한 다음 `ClusterBootstrap`, `cluster-calico` 클러스터 순으로 생성해야 합니다.

```
---
apiVersion: cni.tanzu.vmware.com/v1alpha1
kind: CalicoConfig
metadata:
  name: cluster-calico
spec:
  calico:
    config:
      vethMTU: 0
---
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: ClusterBootstrap
metadata:
  annotations:
    tkg.tanzu.vmware.com/add-missing-fields-from-tkr: v1.23.8---vmware.2-tkg.2-zshippable
  name: cluster-calico
spec:
  cni:
    refName: calico.tanzu.vmware.com.3.22.1+vmware.1-tkg.2-zshippable
    valuesFrom:
      providerRef:
        apiGroup: cni.tanzu.vmware.com
        kind: CalicoConfig
        name: cluster-calico
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-calico
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.23.8---vmware.2-tkg.2-zshippable
```

```
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg2-storage-policy
```

v1beta1 예시: Ubuntu TKR이 있는 클러스터

Tanzu Kubernetes 릴리스의 Ubuntu 버전을 사용하는 v1beta1 클러스터를 프로비저닝하려면 이 예시를 참조하십시오.

v1beta1 예시: Ubuntu TKR이 있는 클러스터

다음 예시 YAML은 v1beta1 API를 사용하여 지정된 TKR의 Ubuntu 버전을 사용하는 클러스터를 프로비저닝하는 방법을 보여줍니다. 이 예시는 v1beta1 예: 기본 클러스터에 기반합니다.

기본적으로 PhotonOS는 클러스터 노드에 사용됩니다. TKR 버전이 여러 OSImage를 지원하는 경우 Photon 대신 Ubuntu를 사용하도록 클러스터 규격에 `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` 주석을 포함합니다. TKr의 Ubuntu 버전을 사용 중인 경우 전체 버전 문자열을 지정해야 하며 클러스터 규격에 OS 주석을 포함해야 합니다. TKR에 대한 자세한 내용은 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리 항목을 참조하십시오.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-ubuntu
  namespace: tkg-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
      metadata:
        annotations:
          run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
    workers:
      machineDeployments:
        - class: node-pool
```

```
          name: node-pool-1
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
        - class: node-pool
          name: node-pool-2
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
        - class: node-pool
          name: node-pool-3
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg-storage-policy
```

v1beta1 예: FQDN이 있는 클러스터

하나 이상의 FQDN(정규화된 도메인 이름)을 사용하여 TKGS 클러스터를 프로비저닝하려면 이 예를 참조하십시오.

FQDN 지원

v1beta1 API를 사용하여 FQDN(정규화된 도메인 이름)으로 TKG 클러스터를 프로비저닝할 수 있습니다. Cluster v1beta1 API에는 Kubernetes API 서버에 대한 TLS 인증서에 생성할 하나 이상의 FQDN 문자열이 포함된 `kubeAPIServerFQDNs`라는 변수가 포함되어 있습니다.

FQDN이 구성된 클러스터에 대해 `kubectl vsphere login` 명령이 실행되면 인증 서비스는 목록에서 첫 번째 FQDN 항목을 선택하고 클러스터와 상호 작용하기 위한 기본 선택 항목으로 kubeconfig에 추가합니다. 목록의 첫 번째 FQDN을 확인할 수 있다고 가정합니다. 클러스터 로그인에는 변경이 필요하지 않습니다.

생성된 Kubernetes API 인증서에는 `kubeAPIServerFQDNs` 변수에 지정한 모든 FQDN이 포함됩니다. 시스템은 목록에서 첫 번째 이외의 다른 FQDN을 사용하려고 시도하지 않습니다. 시스템은 FQDN을 확인하려고 시도하지 않습니다. 목록에서 다른 FQDN을 사용하려는 경우 생성된 kubeconfig 파일을 수동으로 편집하고 원하는 FQDN을 추가할 수 있습니다.

FQDN 요구 사항

FQDN 사용은 선택 사항입니다. FQDN을 사용하지 않는 경우에도 기능은 변경되지 않습니다. 여기에 설명된 기능은 TKG 워크로드 클러스터에만 해당됩니다. 감독자에서 FQDN을 사용하려면 감독자 설명서에서 해당 항목을 참조하십시오.

FQDN을 사용하여 TKG 클러스터를 프로비저닝하려면 다음 요구 사항을 준수합니다.

- vSphere 8.0 U2 P03 이상 환경
- 감독자가 최신 패치 릴리스로 업데이트됨
- v1beta1 API 클러스터만 지원됨. v1alpha3 API 클러스터는 지원되지 않음
- 선택한 FQDN을 유효한 IP 주소로 확인하도록 DNS가 구성됨

중요 FQDN 기능은 v1beta1 API를 사용하여 CAPI 클러스터를 프로비저닝하는 경우에만 사용할 수 있습니다. FQDN과 함께 v1alpha3 API를 사용하여 TKC를 프로비저닝할 수 없습니다.

FQDN 예

Cluster v1beta1 API를 사용하여 FQDN으로 클러스터를 생성합니다.

`spec.topology.variables.kupeAPIServerFQDNs` 값은 FQDN 어레이입니다.

시스템에서 목록의 첫 번째 FQDN을 선택합니다. 이 예에서는 `demo.fqdn.com`입니다.

```
#cluster-example-fqdn.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: tkg-cluster-fqdn
  namespace: tkg-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.52.100.0/12"]
    pods:
      cidrBlocks: ["192.101.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.26.5+vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-01
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkgs-storage-class
      - name: defaultStorageClass
        value: tkg-storage-class
```

```
    - name: kubeAPIServerFQDNs
      value:
        - demo.fqdn.com
        - explore.fqdn.com
```

FQDN 확인

다음 절차를 완료하여 변수 목록의 첫 번째 FQDN이 `kubeconfig` 파일에 포함되어 있고 변수 목록의 모든 FQDN이 Kubernetes API 서버의 TLS 인증서에 있는지 확인합니다.

1 Kubectl을 사용하여 TKG 클러스터에 로그인합니다.

```
kubectl vsphere login --server=SVCP IP or FQDN --vsphere-username USERNAME --tanzu-
kubernetes-cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

2 `kubeconfig` 파일에서 FQDN을 봅니다.

```
cat ~/.kube/config
```

3 목록의 첫 번째 FQDN 변수가 `kubeconfig`에 포함되어 있는지 확인합니다.

예:

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: false
    server: https://10.199.155.77:6443
  name: 10.199.155.77
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJ...DQWRLZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXc
    ...
    CkdiL1pua09rOVVjT3BwSStCTE9ZZDR0RGd2eHo...QUp0SUUKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    server: https://demo.fqdn.com:6443
  name: demo.fqdn.com
```

4 vSphere Client를 사용하여 **워크로드 관리** > ***네임스페이스*** > **계산** > **Tanzu Kubernetes 클러스터** > **제어부 주소**에서 TKGS 클러스터의 IP 주소를 가져옵니다.

5 IP 주소 및 FQDN을 사용하여 로컬 `/etc/hosts` 파일에서 수동 DNS 항목을 만듭니다.

예:

```
sudo vi /etc/hosts
127.0.0.1 localhost
127.0.1.1 ubuntu-client-vm
10.199.155.77 demo.fqdn.com
...
```

6 `openssl s_client` 명령을 사용하여 TLS 인증서를 봅니다.

```
echo | openssl s_client -servername hostname -connect FQDN:PORT 2>/dev/null | openssl x509
-text
```

여기서 `FQDN`은 `kubeAPIServerFQDNs` 변수 목록의 첫 번째 항목입니다.

예:

```
echo | openssl s_client -servername hostname -connect demo.fqdn.com:6443 2>/dev/null |
openssl x509 -text
```

7 `Subject Alternative Name` 필드에는 모든 FQDN이 포함되어야 합니다.

```
X509v3 Subject Alternative Name:
    DNS:demo.fqdn.com, DNS:explore.fqdn.com, DNS:kubernetes, DNS:kubernetes.default,
DNS:kubernetes.default.svc, DNS:kubernetes.def
```

Kubernetes API 서버의 TLS 인증서에는 `kubeAPIServerFQDNs` 목록에 있는 모든 FQDN이 포함되어 있으므로 목록의 두 번째(또는 세 번째 등) FQDN을 사용하도록 `kubeconfig` 파일을 수동으로 업데이트하면 사용이 가능합니다(확인할 수 있다고 가정).

v1beta1 예시: 여러 vSphere 영역의 클러스터

3개 vSphere 영역에 배포된 감독자에서 v1beta1 클러스터를 프로비저닝하려면 이 예시를 참조하십시오.

v1beta1 예시: 여러 vSphere 영역의 클러스터

다음 예시 YAML은 v1beta1 API를 사용하여 vSphere 영역 토폴로지에서 클러스터를 프로비저닝합니다. 이 예시는 v1beta1 예: 기본 클러스터에 기반합니다.

이 예시는 여러 작업자 노드 풀을 구현합니다. 각 노드 풀은 vSphere 영역에 매핑되는 장애 도메인을 참조합니다. vSphere 영역에 대한 자세한 내용은 "vSphere IaaS 제어부 설치 및 구성" 항목을 참조하십시오.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-zoned
  namespace: tkg-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
```

```
    #muliple node pools are used
    machineDeployments:
      - class: node-pool
        name: node-pool-1
        replicas: 3
        #failure domain the machines will be created in
        #maps to a vSphere Zone; name must match exactly
        failureDomain: vsphere-zone1
      - class: node-pool
        name: node-pool-2
        replicas: 3
        #failure domain the machines will be created in
        #maps to a vSphere Zone; name must match exactly
        failureDomain: vsphere-zone2
      - class: node-pool
        name: node-pool-3
        replicas: 3
        #failure domain the machines will be created in
        #maps to a vSphere Zone; name must match exactly
        failureDomain: vsphere-zone3
  variables:
    - name: vmClass
      value: guaranteed-medium
    - name: storageClass
      value: tkg-storage-policy
```

v1beta1 예시: 라우팅 가능한 포드 네트워크가 있는 클러스터

v1beta1 API를 사용하여 라우팅 가능한 포드 네트워크가 있는 클러스터를 생성할 수 있습니다. 이 작업은 기본 클러스터를 `AntreaConfig` 및 `VSphereCPIConfig`에 대한 사용자 지정 구성으로 재정의하여 수행합니다.

v1beta1 API를 사용한 라우팅 가능한 포드 네트워킹 정보

다음 예시 YAML은 v1beta1 API를 사용하여 Antrea RoutablePod 기능을 사용하도록 설정된 클러스터를 프로비저닝하는 방법을 보여줍니다. 이 예시는 v1beta1 예시: 기본 클러스터에 기반합니다.

RoutablePod 기능을 사용하도록 설정하려면 클러스터에 특수 구성이 포함된 `AntreaConfig` 및 `VSphereCPIConfig`가 필요합니다.

`AntreaConfig`는 `trafficEncapMode: noEncap` 및 `noSNAT: true`를 설정해야 합니다.

`VSphereCPIConfig`는 `antreaNSXPodRoutingEnabled: true`, `mode: vsphereParavirtualCPI` 및 다음을 설정해야 합니다.

```
tlsCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_
WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

`AntreaConfig`의 이름은 `<cluster-name>-antrea-package` 형식이어야 합니다. `VSphereCPIConfig`의 이름은 `<cluster-name>-vsphere-cpi-package` 형식이어야 합니다.

구성 파일이 생성되면 구성 파일을 참조하는 클러스터 규격 개체를 생성합니다. 클러스터를 생성하는 동안 구성 파일은 클러스터를 프로비저닝하고 기본 구성을 덮어쓰는 데 사용됩니다.

라우팅 가능한 포드 네트워크 생성: 감독자 구성

라우팅 가능한 포드 네트워크를 생성하려면 감독자 및 TKG 클러스터에 대한 구성이 필요합니다.

참고 라우팅 가능한 포드 네트워킹을 사용하려면 NSX로 감독자를 구성해야 합니다. VDS 네트워킹에는 라우팅 가능한 포드를 사용할 수 없습니다.

감독자에서 라우팅 가능한 포드 네트워크를 구성하려면 다음을 수행합니다.

1 새 vSphere 네임스페이스를 생성합니다.

TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.

2 **감독자 네트워크 설정 재정의** 확인란 옵션을 선택합니다.

지침은 vSphere 네임스페이스에 대한 워크로드 네트워크 설정 재정의 항목을 참조하십시오.

3 다음과 같이 라우팅 가능한 포드 네트워크를 구성합니다.

필드	설명
NAT 모드	NAT(네트워크 주소 변환)를 사용하지 않도록 설정하려면 이 옵션을 선택 취소합니다.
네임스페이스 네트워크	이 필드를 IP 주소/비트 형식의 라우팅 가능한 IP 서브넷으로 채웁니다(예: 10.0.0.6/16). NCP는 네트워크에 대해 지정된 IP 블록에서 하나 이상의 IP 풀을 생성합니다. 최소한 /23 서브넷 크기를 지정해야 합니다. 예를 들어 /23 라우팅 가능한 서브넷을 /28 서브넷 접두사와 함께 지정하면 6 노드 클러스터에 충분한 32개의 서브넷을 가져올 수 있습니다. 접두사가 /28인 /24 서브넷은 2개의 서브넷만 가져오므로 충분하지 않습니다. **주의** 추가하는 라우팅 가능한 IP 서브넷이 클러스터 노드의 IP 주소를 할당하는 **서비스 CIDR**과 겹치지 않는지 확인합니다. **감독자 > 구성 > 네트워크 > 워크로드 네트워크**에서 **서비스 CIDR**을 확인할 수 있습니다.
네임스페이스 서브넷 접두사	예를 들어 /28 형식으로 서브넷 접두사를 지정합니다. 서브넷 접두사는 네임스페이스 네트워크에서 각 노드의 포드 서브넷을 분할하는 데 사용됩니다.

4 **생성**을 클릭하여 라우팅 가능한 포드 네트워크를 생성합니다.

라우팅 가능한 포드 네트워크 생성: TKG 클러스터 구성

다음 예시 YAML은 라우팅 가능한 포드 네트워크로 v1beta1 클러스터를 구성하는 방법을 보여줍니다.

아래 예시에서 볼 수 있듯이 클러스터 규격에서 `spec.clusterNetwork.pod` 섹션을 제거해야 합니다. 포드 IP 주소는 cloud-provider-vsphere에 의해 할당되기 때문입니다.

참고 예시는 단일 YAML 파일로 제공되지만 개별 파일로 분리할 수 있습니다. 이렇게 하려면 먼저 `AntreaConfig` 및 `VSphereCPIConfig` 사용자 지정 리소스를 생성한 다음, target-cluster 클러스터 순으로 생성해야 합니다.

```
---
apiVersion: cni.tanzu.vmware.com/v1alpha1
kind: AntreaConfig
metadata:
 name: target-cluster-antrea-package
spec:
 antrea:
   config:
     defaultMTU: ""
     disableUdpTunnelOffload: false
     featureGates:
       AntreaPolicy: true
       AntreaProxy: true
       AntreaTraceflow: true
       Egress: false
       EndpointSlice: true
       FlowExporter: false
       NetworkPolicyStats: false
       NodePortLocal: false
     noSNAT: true
     tlsCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WI
TH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384
     trafficEncapMode: noEncap
---
apiVersion: cpi.tanzu.vmware.com/v1alpha1
kind: VSphereCPIConfig
metadata:
 name: target-cluster-vsphere-cpi-package
spec:
 vsphereCPI:
   antreaNSXPodRoutingEnabled: true
   insecure: false
   mode: vsphereParavirtualCPI
   tlsCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_
WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
 name: target-cluster
spec:
 clusterNetwork:
   services:
```

```
      cidrBlocks: ["198.51.100.0/12"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg2-storage-policy
```

v1beta1 예: SSL/TLS에 대해 신뢰할 수 있는 추가 CA 인증서가 있는 클러스터

신뢰할 수 있는 추가 CA 인증서가 하나 이상 있는 v1beta1 클러스터를 프로비저닝하려면 이 예를 참조하십시오.

v1beta1 예: 신뢰할 수 있는 추가 CA 인증서가 있는 클러스터

Cluster v1beta1 API 는 하나 이상의 신뢰할 수 있는 CA 인증서를 추가로 사용하여 클러스터를 프로비저닝하기 위한 trust변수를 제공합니다.

표 7-1. v1beta1 API 신뢰 변수

필드	설명
`trust`	섹션 마커. 데이터를 허용하지 않습니다.
`additionalTrustedCAs`	섹션 마커. 각각에 대한 `name`이 있는 인증서 어레이를 포함합니다.
`name`	이중 base64로 인코딩된 PEM 형식의 CA 인증서가 포함된 Kubernetes 암호의 `data` 맵 필드에 대한 사용자 정의 이름입니다. **참고** 이중 base64 인코딩은 필수입니다. 컨텐츠가 이중 base6로 인코딩되지 않은 경우 결과 PEM 파일을 처리할 수 없습니다.

다음 예에서는 CA 인증서가 포함된 Kubernetes 암호를 v1beta1 API 클러스터 규격에 추가하는 방법을 보여 줍니다.

```
#cluster-with-trusted-private-reg-cert.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster01
  namespace: tkgs-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.52.100.0/12"]
```

```
    pods:
      cidrBlocks: ["192.101.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.26.5+vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-01
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkgs-storage-profile
      - name: defaultStorageClass
        value: tkgs-storage-profile
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
```

다음 예에서는 신뢰할 수 있는 추가 CA 인증서가 포함된 Kubernetes 암호를 보여 줍니다.

```
#additional-ca-1.yaml
apiVersion: v1
data:
  additional-ca-1: TFMwdExTMUNSGlSzZ3Jaa...VVNVWkpRMEMwdExTMHRDZz09
kind: Secret
metadata:
  name: cluster01-user-trusted-ca-secret
  namespace: tkgs-cluster-ns
type: Opaque
```

형식 설명:

- 암호의 `data` 맵 값은 CA 인증서의 이름(이 예에서는 `additional-ca-1`)인 사용자 정의 문자열이며, 값은 이중 base64로 인코딩된 PEM 형식의 CA 인증서입니다.
- `metadata` 섹션에서 암호 이름은 *`CLUSTER-NAME`*`-user-trusted-ca-secret`이어야 합니다. 여기서 *CLUSTER-NAME*은 클러스터의 이름입니다. 이 암호는 클러스터와 동일한 vSphere 네임스페이스에서 생성해야 합니다.

CA 인증서의 컨텐츠를 이중 base64로 인코딩합니다.

- Linux: `base64 -w 0 ca.crt | base64 -w 0`
- Windows: https://www.base64encode.org/.

절차: 새 클러스터

새 TKGS 클러스터에 신뢰할 수 있는 CA 인증서를 하나 이상 추가로 포함하려면 다음 절차를 완료합니다.

1 CA 인증서의 컨텐츠를 이중 base64로 인코딩합니다.

2 값이 이중 base64로 인코딩된 PEM 형식의 CA 인증서인 데이터 맵 이름을 포함하는 Kubernetes 암호를 생성합니다.

3 클러스터 규격에서 `trust.additionalTrustedCAs` 변수를 데이터 맵의 이름으로 채웁니다.

4 평소처럼 클러스터를 프로비저닝합니다.

Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

5 클러스터가 성공적으로 프로비저닝되면 추가한 CA 인증서를 클러스터에서 신뢰합니다.

절차: 기존 클러스터

기존 클러스터에 신뢰할 수 있는 CA 인증서를 하나 이상 더 추가하려면 다음 절차를 완료합니다.

1 CA 인증서의 컨텐츠를 이중 base64로 인코딩합니다.

2 값이 이중 base64로 인코딩된 PEM 형식의 CA 인증서인 데이터 맵 이름을 포함하는 Kubernetes 암호를 생성합니다.

3 kubectl 편집을 구성했는지 확인합니다.

Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

4 클러스터 규격을 편집합니다.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
```

5 `trust.additionalTrustedCAs` 섹션을 규격에 추가합니다.

6 이중 base64로 인코딩된 PEM 형식의 CA 인증서가 포함된 암호의 데이터 맵 이름으로 `additionalTrustedCAs` 필드를 채웁니다.

7 텍스트 편집기에서 변경 내용을 저장하고 kubectl이 변경 내용을 등록했는지 확인합니다.

```
kubectl edit cluster/tkgs-cluster-name
cluster.run.tanzu.vmware.com/tkgs-cluster-name edited
```

8 클러스터에 대해 롤링 업데이트가 시작되면 신뢰할 수 있는 CA 인증서가 추가됩니다.

TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해의 내용을 참조하십시오.

신뢰할 수 있는 추가 CA 인증서 확인

클러스터에 추가된 신뢰할 수 있는 추가 CA 인증서가 클러스터의 kubeconfig 파일에 포함됩니다.

인증서 순환

인증서를 순환하려면 새 암호를 생성하고 적절한 값으로 클러스터 규격을 편집합니다. 이렇게 하면 클러스터의 롤링 업데이트가 트리거됩니다.

참고 시스템은 *CLUSTER-NAME*-user-trusted-ca-secret에 대한 변경 내용을 모니터링하지 않습니다. data 맵 값이 변경되어도 이러한 변경 내용이 클러스터에 반영되지 않습니다. trust.additionalTrustCAs에 대한 새 암호와 해당 데이터 맵 name을 생성해야 합니다.

신뢰할 수 있는 추가 CA 인증서 문제 해결

추가적인 신뢰할 수 있는 CA 오류 문제 해결의 내용을 참조하십시오.

사용 사례

가장 일반적인 사용 사례는 컨테이너 레지스트리에 연결하기 위해 신뢰할 수 있는 다른 CA를 추가하는 것입니다. TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합의 내용을 참조하십시오.

v1beta1 예: 사용자 지정 ClusterClass 기반 클러스터(vSphere 8 U2 이상 워크플로)

사용자 지정 ClusterClass를 기반으로 TKG 클러스터를 프로비저닝하려면 다음 지침을 참조하십시오. 이러한 지침은 vSphere 8 U2 이상 환경에만 적용됩니다.

사전 요구 사항

사용자 지정 ClusterClass를 기반으로 TKG 클러스터를 프로비저닝하는 절차가 vSphere 8 U2 릴리스에 대해 업데이트되었습니다.

다음 사전 요구 사항을 준수합니다.

- vSphere 8 U2+ 환경
- 워크로드 관리 사용
- 감독자 구성됨
- vSphere에 대한 Kubernetes CLI 도구가 설치된 Ubuntu 클라이언트

주의 사용자 지정 ClusterClass는 업스트림 클러스터 API 설명서에 따른 실험적인 Kubernetes 기능입니다. 사용자 지정 ClusterClass에서 사용할 수 있는 사용자 지정 범위로 인해 VMware 가능한 모든 사용자 지정을 테스트하거나 검증할 수 없습니다. 고객은 사용자 지정 ClusterClass 클러스터를 테스트, 검증 및 문제 해결해야 합니다. 고객은 사용자 지정 ClusterClass 클러스터와 관련된 지원 티켓을 열 수 있습니다. 다만, VMware 지원 팀이 최선의 노력을 기울이더라도, 사용자 지정 ClusterClass 클러스터에 대해 열린 모든 문제에 대한 해결을 보장할 수는 없습니다. 운영 환경에서 사용자 지정 ClusterClass 클러스터를 배포하기 전에 고객은 이러한 위험을 알고 있어야 합니다.

개략적인 워크플로

개략적인 워크플로는 다음과 같습니다.

시작하려면 다음 워크플로만 있으면 됩니다.

단계	작업	지침
1	기본 ClusterClass를 복제하여 사용자 지정 ClusterClass를 생성합니다.	1: 사용자 지정 ClusterClass 생성
2	사용자 지정 ClusterClass를 기반으로 새 TKG 클러스터를 프로비저닝하고 모든 클러스터 노드가 제대로 작동하는지 확인합니다.	2: 사용자 지정 ClusterClass를 기반으로 TKG 클러스터 생성

사용자 지정 ClusterClass를 변경하고 사용자 지정 ClusterClass 클러스터 노드의 롤링 업데이트를 시작하려면 다음 워크플로를 참조하십시오.

참고 아래 워크플로에 설명된 작업은 사용자 지정 ClusterClass에 대해 수행할 수 있는 작업의 예입니다. 사용 사례는 다를 수 있지만 일반 워크플로를 적용할 수 있어야 합니다.

단계	작업	지침
3	작업자 노드 중 하나에 SSH를 통해 연결하여 업데이트할 패키지가 있는지 확인합니다.	3: 패키지 업데이트의 존재 여부 확인
4	업데이트를 수행하는 새 명령으로 사용자 지정 ClusterClass를 업데이트합니다.	4: 사용자 지정 ClusterClass 업데이트
5	업데이트가 이미 실행된 새 노드의 롤아웃을 확인합니다.	5: 클러스터 노드의 롤링 업데이트 확인

1: 사용자 지정 ClusterClass 생성

첫 번째 부분에는 이름이 `tanzukubernetescluster`인 기본 ClusterClass를 복제하여 `ccc`(`customclusterclass`의 약어)라는 사용자 지정 ClusterClass를 생성하는 작업이 포함됩니다.

참고 사용자 지정 ClusterClass 이름은 사용자가 정의합니다. 다른 이름을 사용하는 경우 지침을 적절히 조정하십시오.

1 이름이 *ccc-ns*인 vSphere 네임스페이스를 생성하고 구성합니다.

사용 권한, 스토리지, 컨텐츠 라이브러리 및 VM 클래스를 구성합니다. 필요한 경우 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성를 참조하십시오.

참고 vSphere 네임스페이스 이름은 사용자가 정의합니다. 다른 이름을 사용하는 경우 지침을 적절히 조정하십시오.

2 감독자에 로그인합니다.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USER@vsphere.local
```

3 기본 ClusterClass의 출력을 이름이 *ccc.yaml*인 파일에 씁니다.

```
kubectl -n ccc-ns get clusterclass tanzukubernetescluster -o yaml > ccc.yaml
```

또는 바로 가기 버전:

```
kubectl -n ccc-ns get cc tanzukubernetescluster -o yaml > ccc.yaml
```

4 복제된 ClusterClass 파일을 편집하기 위해 엽니다.

```
vim ccc.yaml
```

5 `ccc.yaml` 파일을 편집합니다.

- `metadata.creationTimestamp` 줄을 삭제합니다.
- `metadata.generation` 줄을 삭제합니다.
- `metadata.resourceVersion` 줄을 삭제합니다.
- `metadata.uid` 줄을 삭제합니다.
- `metadata.name` 값을 `tanzukubernetescluster`에서 `ccc`로 변경합니다
- `metadata.namespace` 값을 그대로 둡니다. `ccc-ns`
- `run.tanzu.vmware.com/resolve-tkr: ""`에 대해 `metadata.annotations` 값을 그대로 둡니다. 이 주석은 TKR 데이터/확인에 필요합니다.

6 변경 내용을 저장하고 확인합니다.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  annotations:
    run.tanzu.vmware.com/resolve-tkr: ""
  name: ccc
  namespace: ccc-ns
spec:
...
```

7 사용자 지정 ClusterClass 개체를 생성합니다.

```
kubectl apply -f ccc.yaml -n ccc-ns
```

예상 결과:

```
clusterclass.cluster.x-k8s.io/ccc created
```

8 사용자 지정 ClusterClass를 나열합니다.

```
kubectl get cc -n ccc-ns
```

예상 결과:

```
NAME                        AGE
ccc                         3m14s
tanzukubernetescluster      29m
```

2: 사용자 지정 ClusterClass를 기반으로 TKG 클러스터 생성

Cluster v1beta1 API를 사용하여 ClusterClass를 기반으로 클러스터를 생성합니다.

1 클러스터를 프로비저닝하도록 `ccc-cluster.yaml` 매니페스트를 구성합니다.

```
#ccc-cluster.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: ccc-cluster
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.0.2.0/16
    services:
      cidrBlocks:
      - 198.51.100.0/12
    serviceDomain: cluster.local
  topology:
    class: ccc
    version: v1.26.5---vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 1
    workers:
      machineDeployments:
        - class: node-pool
          name: tkgs-node-pool-1
          replicas: 1
    variables:
    - name: vmClass
      value: guaranteed-small
    - name: storageClass
      value: tkg-storage-profile
```

형식 설명:

- `metadata.name` 값은 클러스터: `ccc-cluster`의 이름입니다.
- `spec.topology.class` 값은 사용자 지정 ClusterClass: `ccc`의 이름입니다.
- `spec.topology.version` 값은 TKR 버전입니다.
- `spec.topology.variables.storageClass` 값은 영구 스토리지 클래스의 이름입니다.

참고 테스트 목적으로는 제어부 및 작업자 노드 풀에 대해 복제본 1개면 충분합니다. 운영 환경에서는 제어부에 대해 복제본 3개를 사용하고 각 작업자 노드 풀에 대해 최소 3개의 복제본을 사용합니다.

2 사용자 지정 ClusterClass를 기반으로 TKG 클러스터를 생성합니다.

```
kubectl apply -f ccc-cluster.yaml -n ccc-ns
```

예상 결과:

```
cluster.cluster.x-k8s.io/ccc-cluster created
```

3 클러스터 프로비저닝을 확인합니다.

다음 명령을 실행합니다. 모든 클러스터 노드가 제대로 표시될 때까지 기다립니다.

```
kubectl -n ccc-ns get
cc,clusters,vsphereclusters,kcp,machinedeployment,machineset,machine,vspheremachine,virtual
machineservice
```

참고 5단계에서 롤링 업데이트 진행률을 모니터링할 수 있도록 별도의 세션에서 이 명령을 실행하는 것이 유용합니다.

3: 패키지 업데이트의 존재 여부 확인

작업자 노드 중 하나에 SSH를 통해 연결하여 업데이트할 패키지가 있는지 확인합니다.

참고 이 단계의 목표는 단지 업데이트할 패키지가 있는지 확인하는 것이며 실제로 업데이트하는 것은 아닙니다. 새 클러스터 노드가 롤아웃되면 사용자 지정 ClusterClass에 의해 업데이트됩니다(아 단계 다음). 이 단계와 다음 단계는 사용자 지정 ClusterClass에 대해 수행할 수 있는 작업의 예입니다.

1 다음 명령을 실행하여 SSH 암호를 가져옵니다.

```
export CC=ccc-cluster && kubectl get secret -n ccc-ns ${CC}-ssh -o jsonpath={.data.ssh-
privatekey} | base64 -d > ${CC}-ssh && chomd 4000 ${CC}-ssh
```

2 다음 명령을 실행하여 작업자 노드 VM의 IP 주소를 가져옵니다.

```
kubectl -n ccc-ns get vm -o wide
```

참고 여러 작업자 노드를 배포한 경우 하나를 선택합니다. 제어부 노드를 사용하지 마십시오.

3 다음 명령을 실행하여 SSH를 통해 작업자 노드 VM에 연결합니다.

```
ssh -i ${CC}-ssh vmware-system-user@IP-ADDRESS-OF-WORKER-NODE
```

예:

```
ssh -i ${CC}-ssh vmware-system-user@192.168.128.55
```

참고 연결을 계속하려면 "yes"를 입력합니다.

예상 결과: 호스트에 SSH로 연결하면 다음 메시지가 표시됩니다.

```
tdnf update info not availble yet!
```

4 다음 명령을 실행하고 업데이트를 확인합니다.

```
sudo -i
```

```
tdnf update
```

5 프롬프트에서 no(업데이트 안 함)를 의미하는 "N"을 입력합니다.

예상 결과:

```
Operation aborted
```

참고 여기서 목적은 업데이트를 시작하는 것이 아니라 단지 업데이트가 있는지 확인하는 것입니다. 다음 섹션에서 사용자 지정 ClusterClass에 명령을 추가하여 업데이트를 시작합니다.

6 SSH 세션에서 로그아웃하려면 "exit"를 입력한 다음, "exit"를 다시 입력합니다.

4: 사용자 지정 ClusterClass 업데이트

tdnf 업데이트를 수행하는 새 명령으로 사용자 지정 ClusterClass를 업데이트합니다.

1 이름이 `ccc`인 사용자 지정 ClusterClass를 편집하기 위해 를 엽니다.

```
kubectl edit cc ccc -n ccc-ns
```

2 `postKubeadmCommands`가 있는 다음 섹션까지 아래로 스크롤합니다.

```
  - definitions:
    - jsonPatches:
      - op: add
        path: /spec/template/spec/kubeadmConfigSpec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
      selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: KubeadmControlPlaneTemplate
        matchResources:
          controlPlane: true
    - jsonPatches:
      - op: add
        path: /spec/template/spec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
```

```
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
      selector:
        apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
        kind: KubeadmConfigTemplate
        matchResources:
          machineDeploymentClass:
            names:
            - node-pool
    name: controlPlanePostKubeadmCommandsSuccess
```

두 `valueFrom.template` 필드에 다음 명령을 추가합니다.

```
- tdnf update -y
```

예:

```
  - definitions:
    - jsonPatches:
      - op: add
        path: /spec/template/spec/kubeadmConfigSpec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
            - tdnf update -y
      selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: KubeadmControlPlaneTemplate
        matchResources:
          controlPlane: true
    - jsonPatches:
      - op: add
        path: /spec/template/spec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
            - tdnf update -y
      selector:
        apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
        kind: KubeadmConfigTemplate
        matchResources:
          machineDeploymentClass:
            names:
            - node-pool
    name: controlPlanePostKubeadmCommandsSuccess
```

3 사용자 지정 ClusterClass에 변경 내용을 저장하고 편집기를 닫습니다.

```
wq
```

예상 결과:

```
clusterclass.cluster.x-k8s/ccc edited
```

5: 클러스터 노드의 롤링 업데이트 확인

사용자 지정 ClusterClass를 업데이트하면 해당 ClusterClass를 기반으로 프로비저닝된 클러스터에 대한 클러스터 노드의 롤링 업데이트가 트리거됩니다. 새 노드에 위의 명령이 적용된 상태가 나타납니다.

1 다음 명령을 실행하여 클러스터 프로비저닝을 확인합니다.

모든 클러스터 노드가 제대로 표시될 때까지 기다립니다.

```
kubectl -n ccc-ns get
cc,clusters,vsphereclusters,kcp,machinedeployment,machineset,machine,vspheremachine,virtual
machineservice
```

2 새 UUID가 있는 새 노드가 배포된 것을 볼 수 있습니다.

3 다음 명령을 실행하여 SSH를 통해 작업자 노드 VM에 연결합니다.

```
ssh -i ${CC}-ssh vmware-system-user@IP-ADDRESS-OF-WORKER-NODE
```

예상 결과: 호스트에 SSH로 연결하면 다음 메시지가 표시됩니다.

```
tdnf update info not availble yet!
```

4 다음 명령을 실행합니다.

```
sudo -i
```

```
tdnf update
```

예상 결과: 업데이트해야 하는 패키지가 훨씬 적어야 합니다.

5 프롬프트에서 no(업데이트 안 함)를 의미하는 "N"을 입력합니다.

예상 결과:

```
Operation aborted
```

6 다음 명령을 실행하여 tdnf가 실행되었는지 확인합니다.

```
cat /var/log/cloud-init-output.log | grep -i tdnf
```

7 SSH 세션에서 로그아웃하려면 "exit"를 입력한 다음, "exit"를 다시 입력합니다.

사용자 지정 ClusterClass 유지 관리

TKG 서비스를 새 버전으로 업그레이드한 후 이전 TKG 서비스 버전의 기본 ClusterClass에서 파생된 사용자 지정 ClusterClass가 새 TKG 서비스 버전과 함께 제공되는 기본 ClusterClass에 대한 변경 내용으로 업데이트되었는지 확인해야 합니다.

다음 워크플로를 사용하여 사용자 지정 ClusterClass를 시스템 제공 ClusterClass와 동기화 상태로 유지합니다. 이러한 지침에서는 여기에 설명된 대로 초기 사용자 지정 ClusterClass를 생성한 것으로 가정합니다.

1 TKG 서비스 버전을 업그레이드합니다.

   예를 들어 TKG 서비스 v3.0에서 v3.1로 업그레이드합니다.

   장 3 TKG 서비스 설치 및 업그레이드의 내용을 참조하십시오.

2 여기에 있는 개략적인 워크플로에 따라 새 사용자 지정 ClusterClass를 생성합니다.

   이름에 TKG 서비스 버전(예: `ccc-3.1`)을 추가하여 새 사용자 지정 ClusterClass 버전을 수동으로 지정합니다.

3 이전 사용자 지정 ClusterClass의 사용자 지정 패치 및 변수를 새 사용자 지정 ClusterClass에 추가합니다.

   이렇게 하려면 `cat ccc.yaml`을 실행하고 사용자 지정 패치 및 변수를 `ccc-3.1.yaml`으로 복사합니다.

4 새 사용자 지정 ClusterClass를 적용하고 조정이 성공할 때까지 기다립니다.

5 클러스터 개체의 `spec.topology.class` 필드를 편집하여 이전 사용자 지정 ClusterClass를 사용하는 TKG 클러스터를 새 사용자 지정 ClusterClass로 업데이트합니다.

비관리형 ClusterClass

vSphere 8 U2+에서, TKG 컨트롤러가 사용자 지정 ClusterClass를 관리하는 것을 원치 않는 경우 사용자 지정 ClusterClass에 추가할 수 있는 주석이 있습니다. 이 주석을 추가하는 경우에는 인증서, 암호 등과 같은 모든 기본 Kubernetes 개체를 수동으로 생성해야 합니다. 이 작업을 위한 지침은 vSphere 8 U1 사용자 지정 ClusterClass v1beta1 예: 사용자 지정 ClusterClass 기반 클러스터(vSphere 8 U1 워크플로)를 참조하십시오.

주석은 다음과 같습니다.

주석 키	값
run.tanzu.vmware.com/unmanaged-clusterclass	" "

다음은 이름이 `ccc`인 사용자 지정 ClusterClass에 주석을 추가하는 방법의 예입니다.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  annotations:
    run.tanzu.vmware.com/resolve-tkr: ""
    run.tanzu.vmware.com/unmanaged-clusterclass: ""
  name: ccc
  namespace: ccc-ns
spec:
...
```

v1beta1 예: 사용자 지정 ClusterClass 기반 클러스터(vSphere 8 U1 워크플로)

사용자 지정 ClusterClass를 기반으로 TKG 클러스터를 프로비저닝하려면 다음 지침을 참조하십시오. 이러한 지침은 vSphere 8 U1 환경에만 적용됩니다.

사전 요구 사항

사용자 지정 ClusterClass를 기반으로 TKG 클러스터를 프로비저닝하는 절차는 vSphere 8 U1 릴리스부터 사용할 수 있습니다. vSphere 8 U2를 사용하는 경우 v1beta1 예: 사용자 지정 ClusterClass 기반 클러스터(vSphere 8 U2 이상 워크플로) 항목을 참조하십시오.

다음 사전 요구 사항을 준수합니다.

- vSphere 8 U1 환경
- 워크로드 관리 사용
- 감독자 구성됨
- vSphere에 대한 Kubernetes CLI 도구가 설치된 Ubuntu 클라이언트

주의 사용자 지정 ClusterClass는 업스트림 클러스터 API 설명서에 따른 실험적인 Kubernetes 기능입니다. 사용자 지정 ClusterClass에서 사용할 수 있는 사용자 지정 범위로 인해 VMware 가능한 모든 사용자 지정을 테스트하거나 검증할 수 없습니다. 고객은 사용자 지정 ClusterClass 클러스터를 테스트, 검증 및 문제 해결해야 합니다. 고객은 사용자 지정 ClusterClass 클러스터와 관련된 지원 티켓을 열 수 있습니다. 다만, VMware 지원 팀이 최선의 노력을 기울이더라도, 사용자 지정 ClusterClass 클러스터에 대해 열린 모든 문제에 대한 해결을 보장할 수는 없습니다. 운영 환경에서 사용자 지정 ClusterClass 클러스터를 배포하기 전에 고객은 이러한 위험을 알고 있어야 합니다.

1부: 사용자 지정 ClusterClass 생성

첫 번째 부분에는 이름이 `tanzukubernetescluster`인 기본 ClusterClass를 복제하여 사용자 지정 ClusterClass를 생성하는 작업이 포함됩니다.

1 이름이 *custom-ns*인 vSphere 네임스페이스를 생성합니다.

2 감독자에 로그인합니다.

3 컨텍스트를 이름이 *custom-ns*인 vSphere 네임스페이스로 전환합니다.

4 기본 ClusterClass를 가져옵니다.

```
kubectl get clusterclass tanzukubernetescluster -o json
```

5 기본 ClusterClass를 복제하여 이름이 *custom-cc*인 사용자 지정 ClusterClass를 생성합니다.

```
kubectl get clusterclass tanzukubernetescluster -o json | jq '.metadata.name="custom-cc"'
| kubectl apply -f -
```

예상 결과:

```
clusterclass.cluster.x-k8s.io/custom-cc created
```

6 사용자 지정 ClusterClass를 가져옵니다.

```
kubectl get clusterclass custom-cc -o json
```

필요한 경우 를 적게 사용하여 사용자 지정 ClusterClass를 볼 수 있습니다.

```
kubectl get clusterclass custom-cc -o json | less
```

참고 "q" 명령을 실행하여 더 적게 종료합니다.

2부: TKG 클러스터를 프로비저닝하는 데 필요한 감독자 개체 생성

다음 부분은 사용자 지정 ClusterClass를 사용하여 사용자 지정 TKG 클러스터의 초기 배포에 필요한 감독자 개체를 생성하는 것입니다.

참고 기본적으로 클러스터 이름 "ccc-cluster"를 사용합니다. 다른 클러스터 이름을 사용하는 경우 적절한 필드를 변경해야 합니다.

1 자체 서명된 확장 인증서에 대한 발급자를 생성합니다.

```
#self-signed-extensions-issuer.yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: self-signed-extensions-issuer
spec:
  selfSigned: {}
```

```
kubectl apply -f self-signed-extensions-issuer.yaml -n custom-ns
```

예상 결과:

```
issuer.cert-manager.io/self-signed-extensions-issuer created
```

2 확장 CA 인증서에 대한 암호를 생성합니다.

```
#extensions-ca-certificate.yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ccc-cluster-extensions-ca
spec:
  commonName: kubernetes-extensions
  duration: 87600h0m0s
  isCA: true
  issuerRef:
    kind: Issuer
    name: self-signed-extensions-issuer
  secretName: ccc-cluster-extensions-ca
```

```
  usages:
  - digital signature
  - cert sign
  - crl sign
```

```
kubectl apply -f extensions-ca-certificate.yaml -n custom-ns
```

예상 결과:

```
certificate.cert-manager.io/ccc-cluster-extensions-ca created
```

3 확장 CA 인증서에 대한 발급자를 생성합니다.

```
#extensions-ca-issuer.yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ccc-cluster-extensions-ca-issuer
spec:
  ca:
    secretName: ccc-cluster-extensions-ca
```

```
kubectl apply -f extensions-ca-issuer.yaml -n custom-ns
```

예상 결과:

```
issuer.cert-manager.io/ccc-cluster-extensions-ca-issuer created
```

4 인증 서비스 인증서에 대한 암호를 생성합니다.

```
#auth-svc-cert.yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ccc-cluster-auth-svc-cert
spec:
  commonName: authsvc
  dnsNames:
  - authsvc
  - localhost
  - 127.0.0.1
  duration: 87600h0m0s
  issuerRef:
    kind: Issuer
    name: ccc-cluster-extensions-ca-issuer
  secretName: ccc-cluster-auth-svc-cert
  usages:
  - server auth
  - digital signature
```

```
kubectl apply -f auth-svc-cert.yaml -n custom-ns
```

예상 결과:

```
certificate.cert-manager.io/ccc-cluster-auth-svc-cert created
```

5 발급자 및 인증서의 생성을 확인합니다.

```
kubectl get issuers -n custom-ns
NAME                                READY   AGE
ccc-cluster-extensions-ca-issuer    True    2m57s
self-signed-extensions-issuer       True    14m
```

```
kubectl get certs -n custom-ns
NAME                          READY   SECRET                        AGE
ccc-cluster-auth-svc-cert     True    ccc-cluster-auth-svc-cert     34s
ccc-cluster-extensions-ca     True    ccc-cluster-extensions-ca     5m
```

3부: 사용자 지정 ClusterClass를 기반으로 TKG 클러스터 생성

Cluster v1beta1 API를 사용하여 ClusterClass를 기반으로 클러스터를 생성합니다. 사용자 지정 ClusterClass를 기준으로 하는 v1beta1 클러스터에는 다음과 같은 최소 변수 집합이 필요합니다.

변수	설명
vmClass	TKG 서비스 클러스터에서 VM 클래스 사용의 내용을 참조하십시오.
storageClass	vSphere 네임스페이스에 대한 영구 스토리지 구성의 내용을 참조하십시오.
ntp	감독자를 사용하도록 설정하는 데 사용되는 NTP 서버입니다.
extensionCert	이전 섹션에서 "확장 CA 인증서"가 생성된 후 자동으로 생성됩니다.
clusterEncryptionConfigYaml	아래 섹션은 이 파일을 가져오는 프로세스를 안내합니다.

1 암호화 암호를 생성합니다.

```
#encryption-secret.yaml
apiVersion: v1
data:
  key: all3dzZpODFmRmh6MVlJbUtQQktuN2ViQzREbDBQRHlxVk8yYXRxTW9QQT0=
kind: Secret
metadata:
  name: ccc-cluster-encryption
type: Opaque
```

```
kubectl apply -f encryption-secret.yaml -n custom-ns
```

예상 결과:

```
secret/ccc-cluster-encryption created
```

2 감독자에서 NTP 서버를 수집합니다.

```
kubectl -n vmware-system-vmop get configmap vmoperator-network-config -o
jsonpath={.data.ntpservers}
```

3 클러스터를 프로비저닝하도록 `cluster-with-ccc.yaml` 매니페스트를 구성합니다.

```
#cluster-with-ccc.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: ccc-cluster
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 193.0.0.0/16
    serviceDomain: managedcluster1.local
    services:
      cidrBlocks:
      - 198.201.0.0/16
  topology:
    class: custom-cc
    version: v1.26.5---vmware.2-fips.1-tkg.1
    controlPlane:
      metadata: {}
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          metadata: { }
          name: node-pool-workers
          replicas: 3
    variables:
    - name: vmClass
      value: guaranteed-medium
    - name: storageClass
      value: tkg-storage-profile
    - name: ntp
      value: time.acme.com
    - name: extensionCert
      value:
        contentSecret:
          key: tls.crt
          name: ccc-cluster-extensions-ca
    - name: clusterEncryptionConfigYaml
      value: LS0tCm...Ht9Cg==
```

클러스터 매니페스트에서 다음 필드를 확인하거나 업데이트합니다.

매개 변수	설명
`metadata.name`	v1beta1 클러스터의 이름입니다.
`spec.topology.class`	사용자 지정 ClusterClass의 이름입니다.
`spec.topology.version`	Tanzu Kubernetes 릴리스 버전
`spec.topology.variables.storageClass.value`	클러스터가 프로비저닝될 vSphere 네임스페이스에 연결된 StoragePolicy
`spec.topology.variables.ntp.value`	NTP 서버 주소
`spec.topology.variables.extensionCert.value.contentSecret.name`	확인
`spec.topology.variables.clusterEncryptionConfigYaml.value`	ClusterEncryptionConfig 암호의 `data.key` 값으로 채웁니다.

4 사용자 지정 ClusterClass를 기반으로 클러스터를 생성합니다.

```
kubectl apply -f cluster-with-ccc.yaml -n custom-ns
```

예상 결과:

```
cluster.cluster.x-k8s.io/ccc-cluster created
```

vSphere Client를 사용하여 클러스터가 생성되었는지 확인합니다.

5 TKG 클러스터에 로그인합니다.

```
kubectl vsphere login --server=xxx.xxx.xxx.xxx --vsphere-username USERNAME@vsphere.local --tanzu-kubernetes-cluster-name ccc-cluster --tanzu-kubernetes-cluster-namespace custom-ns
```

4부: TKG 클러스터를 관리하는 데 필요한 감독자 개체 생성

CCC가 있는 클러스터가 적용되면 다양한 컨트롤러가 프로비저닝을 시도합니다. 그러나 기본 인프라 리소스에는 여전히 추가 개체를 올바르게 부트스트랩해야 합니다.

매개 변수	값
인증	인증 값을 수집하고 values.yaml 파일로 업데이트해야 합니다.
Base64로 인코딩됨	values.yaml 파일은 base64 문자열로 인코딩됩니다.
guest-cluster-auth-service-data-values.yaml	이 문자열은 파일을 적용하기 전에 CCC_config_yamls.tar.gz에서 다운로드한 guest-cluster-auth-service-data-values.yaml 파일에 추가됩니다.
GuestClusterAuthSvcDataValues 암호	마지막으로 새로 생성된 GuestClusterAuthSvcDataValues 암호를 참조하도록 게스트 클러스터 부트스트랩을 수정해야 합니다.

1 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context custom-ns
```

2 `authServicePublicKeys` 값을 가져옵니다.

```
kubectl -n vmware-system-capw get configmap vc-public-keys -o
jsonpath="{.data.vsphere\.local\.json}"
```

결과를 이름이 `values.yaml`인 텍스트 파일에 복사합니다.

```
authServicePublicKeys: '{"issuer_url":"https://...SShrDw=="]}]}}'
```

3 클러스터 UID를 가져와서 `authServicePublicKeys`를 업데이트합니다.

```
kubectl get cluster -n custom-ns ccc-cluster -o yaml | grep uid
```

4 `values.yaml` 파일의 `authServicePublicKeys` 섹션에서 클러스터 UID를 "`client_id`" 값에 추가합니다.

구문: `vmware-tes:vc:vns:k8s:`*`clusterUID`*

예:

```
vmware-tes:vc:vns:k8s:7d95b50b-4fd4-4642-82a3-5dbfe87f499c
```

5 인증서 값을 가져옵니다(*ccc-cluster*를 선택한 클러스터 이름으로 대체).

```
kubectl -n custom-ns get secret ccc-cluster-auth-svc-cert -o jsonpath="{.data.tls\.crt}" |
base64 -d
```

6 인증서를 `values.yaml`에 추가합니다.

`authServicePublicKeys` 섹션 아래에 인증서 컨텐츠를 추가합니다.

참고 실패를 방지하려면 인증서를 4칸 들여써야 합니다.

예:

```
authServicePublicKeys: '{"issuer_url":"https://...SShrDw=="]}]}}'
ceritificate: |
    -----BEGIN CERTIFICATE-----
    MIIDPTCCAiWgAwIBAgIQMibGSjeuJelQoPxCof/+xzANBgkqhkiG9w0BAQsFADAg
    ...
    sESk/RDTB1UAvi8PD3zcbEKZuRxuo4IAJqFFbAabwULhjUo0UwT+dIJo1gLf5/ep
    VoIRJS7j6VT98WbKyZp5B4I=
    -----END CERTIFICATE-----
```

7 privateKey 값을 가져옵니다.

```
kubectl -n custom-ns get secret ccc-cluster-auth-svc-cert -o jsonpath="{.data.tls\.key}"
```

8 `values.yaml` 파일을 확인합니다.

```
authServicePublicKeys: '{"issuer_url":"https://10.197.79.141/openidconnect/
vsphere.local","client_id":"vmware-tes:vc:vns:k8s:7d95...499c",...SShrDw=="]}]}}'
certificate: |
    -----BEGIN CERTIFICATE-----
    MIIDPTCCAiWgAwIBAgIQWQyXAQDRMhgrGre8ysVN0DANBgkqhkiG9w0BAQsFADAg
    ...
    uJSBP49sF0nKz5nf7w+BdYE=
    -----END CERTIFICATE-----
privateKey: LS0tLS1CRUdJTi...VktLS0tLQo=
```

9 base64 인코딩을 사용하여 `values.yaml` 파일을 해시하여 `guest-cluster-auth-service-data-values.yaml` 파일에 대한 출력을 수집합니다.

```
base64 -i values.yaml -w 0
```

10 `guest-cluster-auth-service-data-values.yaml` 파일을 생성합니다.

암호에 대한 템플릿은 다음과 같습니다.

```
apiVersion: v1
data:
  values.yaml: YXV0a...ExRbz0K
kind: Secret
metadata:
  labels:
    tkg.tanzu.vmware.com/cluster-name: ccc-cluster
    tkg.tanzu.vmware.com/package-name: guest-cluster-auth-
service.tanzu.vmware.com.1.3.0+tkg.2-vmware
  name: ccc-cluster-guest-cluster-auth-service-data-values
type: Opaque
```

다음 표를 참조하여 예상되는 암호 값을 채웁니다.

매개 변수	값
`data.values.yaml`	Base64로 인코딩된 `values.yaml` 문자열
`metadata.labels.cluster-name`	클러스터의 이름(예: `ccc-cluster`)
`metadata.labels.package-name`	guest-cluster-auth-service.tanzu.vmware.com.*version* 이 값을 가져오려면 `kubectl get tkr v1.26.5---vmware.2-fips.1-tkg.1 -o yaml` 명령을 실행합니다 사용 중인 버전에 따라 TKR 버전을 변경합니다.
`metadata.name`	클러스터의 이름(예: `ccc-cluster`)

11 `guest-cluster-auth-service-data-values.yaml` 암호를 생성합니다.

```
kubectl apply -f guest-cluster-auth-service-data-values.yaml -n custom-ns
```

12 암호를 참조하도록 클러스터 부트스트랩을 편집합니다.

```
kubectl edit clusterbootstrap ccc-cluster -n custom-ns
```

13 `guest-cluster-auth-service.tanzu.vmware.com.version:` 줄 아래에 다음 줄을 추가합니다.

```
valuesFrom:
  secretRef: ccc-cluster-guest-cluster-auth-service-data-values
```

예:

```
spec:
  additionalPackages:
  - refName: guest-cluster-auth-service.tanzu.vmware.com.1.3.0+tkg.2-vmware
    valuesFrom:
      secretRef: ccc-cluster-guest-cluster-auth-service-data-values
```

14 저장한 후 종료하여 clusterbootstrap 수정 사항을 적용합니다.

5부: 포드 보안 구성

TKR 버전 1.25 이상을 사용하는 경우 이름이 `custom-ns`인 vSphere 네임스페이스에 대한 포드 보안을 구성합니다. TKR 1.25 이상에 대한 PSA 구성의 내용을 참조하십시오.

TKR 버전 1.24 이상을 사용하는 경우 클러스터의 포드에는 포드 보안 정책에 대한 바인딩이 필요합니다. 클러스터 수준에서 필요한 리소스 개체를 적용하려면 다음 프로세스를 사용합니다.

1 TKG 클러스터 kubeconfig를 수집합니다.

```
kubectl -n custom-ns get secret ccc-cluster-kubeconfig -o jsonpath="{.data.value}" |
base64 -d > ccc-cluster-kubeconfig
```

2 `psp.yaml` 파일을 생성합니다.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: tanzu-system-kapp-ctrl-restricted
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
    - ALL
  volumes:
    - configMap
    - emptyDir
    - projected
    - secret
    - downwardAPI
    - persistentVolumeClaim
  hostNetwork: false
  hostIPC: false
```

```
  hostPID: false
  runAsUser:
    rule: MustRunAsNonRoot
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: MustRunAs
    ranges:
      - min: 1
        max: 65535
  fsGroup:
    rule: MustRunAs
    ranges:
      - min: 1
        max: 65535
  readOnlyRootFilesystem: false
```

3 포드 보안 정책을 적용합니다.

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl apply -f psp.yaml
```

4 TKG 클러스터에 로그인합니다.

```
kubectl vsphere login --server=10.197.154.66 --vsphere-username
administrator@vsphere.local --insecure-skip-tls-verify --tanzu-kubernetes-cluster-name ccc-
cluster --tanzu-kubernetes-cluster-namespace custom-ns
```

5 네임스페이스를 나열합니다.

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl get ns -A
```

```
NAME                           STATUS   AGE
default                        Active   13d
kube-node-lease                Active   13d
kube-public                    Active   13d
kube-system                    Active   13d
secretgen-controller           Active   13d
tkg-system                     Active   13d
vmware-system-antrea           Active   13d
vmware-system-cloud-provider   Active   13d
vmware-system-csi              Active   13d
vmware-system-tkg              Active   13d
```

6부: vSphere SSO 역할을 사용자 지정 TKG 클러스터와 동기화

개발자가 클러스터 워크로드를 관리할 수 있도록 vSphere 네임스페이스에 구축된 vCenter Single Sign-On 사용자에 대한 Rolebinding을 감독자에서 TKG 클러스터로 동기화해야 합니다.

이 프로세스를 수행하려면 감독자에서 rolebinding 바인딩 목록을 내보내고 "편집" 역할이 있는 rolebinding을 수집하고 `sync-cluster-edit-rolebinding.yaml` 파일을 생성한 다음, KUBECONFIG를 사용하여 TKG 클러스터에 적용해야 합니다.

1 감독자에서 기존 rolebinding을 수집합니다.

```
kubectl get rolebinding -n custom-ns  -o yaml
```

2 반환된 rolebinding 개체 목록에서 `roleRef.name`이 "edit"인 개체를 식별합니다.

예:

```
apiVersion: v1
items:
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T18:44:45Z"
    name: ccc-cluster-8lr5x-ccm
    namespace: custom-ns
    ownerReferences:
    - apiVersion: vmware.infrastructure.cluster.x-k8s.io/v1beta1
      blockOwnerDeletion: true
      controller: true
      kind: ProviderServiceAccount
      name: ccc-cluster-8lr5x-ccm
      uid: b5fb9f01-9a55-4f69-8673-fadc49012994
    resourceVersion: "108766602"
    uid: eb93efd4-ae56-4d9f-a745-d2782885e7fb
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: ccc-cluster-8lr5x-ccm
  subjects:
  - kind: ServiceAccount
    name: ccc-cluster-8lr5x-ccm
    namespace: custom-ns
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T18:44:45Z"
    name: ccc-cluster-8lr5x-pvcsi
    namespace: custom-ns
    ownerReferences:
    - apiVersion: vmware.infrastructure.cluster.x-k8s.io/v1beta1
      blockOwnerDeletion: true
      controller: true
      kind: ProviderServiceAccount
      name: ccc-cluster-8lr5x-pvcsi
      uid: d9342f8f-13d2-496d-93cb-b24edfacb5c1
    resourceVersion: "108766608"
    uid: fd1820c7-7993-4299-abb7-bb67fb17f1fd
  roleRef:
    apiGroup: rbac.authorization.k8s.io
```

```
    kind: Role
    name: ccc-cluster-8lr5x-pvcsi
  subjects:
  - kind: ServiceAccount
    name: ccc-cluster-8lr5x-pvcsi
    namespace: custom-ns
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T16:58:06Z"
    labels:
      managedBy: vSphere
    name: wcp:custom-ns:group:vsphere.local:administrators
    namespace: custom-ns
    resourceVersion: "108714148"
    uid: d74a98c7-e7da-4d71-b1d5-deb60492d429
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: edit
  subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: sso:Administrators@vsphere.local
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T16:58:21Z"
    labels:
      managedBy: vSphere
    name: wcp:custom-ns:user:vsphere.local:administrator
    namespace: custom-ns
    resourceVersion: "108714283"
    uid: 07f7dbba-2670-4100-a59b-c09e4b2edd6b
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: edit
  subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: sso:Administrator@vsphere.local
kind: List
metadata:
  resourceVersion: ""
```

3 `sync-cluster-edit-rolebinding.yaml`이라는 파일을 생성하여 기본 administrator@vsphere.local 이외의 추가 rolebinding을 추가합니다. 예:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    run.tanzu.vmware.com/vmware-system-synced-from-supervisor: "yes"
```

```
  name: vmware-system-auth-sync-wcp:custom-ns:group:vsphere.local:administrators
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: sso:Administrators@vsphere.local
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    run.tanzu.vmware.com/vmware-system-synced-from-supervisor: "yes"
  name: vmware-system-auth-sync-wcp:custom-ns:group:SSODOMAIN.COM:testuser
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: sso:testuser@SSODOMAIN.COM
```

참고 `metadata.name` 필드에서 모든 사용자에 대해 사용자 역할 앞에 `vmware-system-auth-sync-`가 추가됩니다. metadata.name 및 subjects.name 항목은 기본이 아닌 모든 역할을 수정해야 합니다.

4 sync-cluster-edit-rolebinding.yaml 구성을 적용하여 rolebinding을 동기화합니다.

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl apply -f sync-cluster-edit-rolebinding.yaml
```

TanzuKubernetesCluster v1alpha3 API 사용

이 섹션에서는 v1alpha3 API를 사용하여 `TanzuKubernetesCluster` 유형의 TKG 서비스 클러스터를 프로비저닝하기 위한 참조 컨텐츠를 제공합니다. 여기에는 요구 사항에 맞는 다양한 구성 및 사용자 지정이 포함된 예시가 포함됩니다.

TanzuKubernetesCluster v1alpha3 API

v1alpha3 API를 사용하면 감독자의 TKG를 사용하여 TanzuKubernetesCluster를 프로비저닝할 수 있습니다. v1alpha3 API 설명서는 이 항목을 참조하십시오.

TanzuKubernetesCluster v1alpha3 API

규격에는 v1alpha3 API를 사용하여 `TanzuKubernetesCluster`를 프로비저닝하는 데 사용할 수 있는 모든 매개 변수가 나열됩니다.

중요 유효한 키 이름은 영숫자, 대시(예: `key-name`), 밑줄(예: `KEY_NAME`) 또는 점(예: `key.name`)으로만 구성되어야 합니다. 키 이름에는 공백 문자를 사용할 수 없습니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: string
  namespace: string
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=string
spec:
  topology:
    controlPlane:
      replicas: int32
      vmClass: string
      storageClass: string
      volumes:
        - name: string
          mountPath: string
          capacity:
            storage: size in GiB
      tkr:
        reference:
          name: string
      nodeDrainTimeout: string
    nodePools:
    - name: string
      failureDomain: string
      labels: map[string]string
      taints:
        -  key: string
           value: string
           effect: string
           timeAdded: time
      replicas: int32
      vmClass: string
      storageClass: string
      volumes:
        - name: string
          mountPath: string
          capacity:
            storage: size in GiB
      tkr:
        reference:
          name: string
      nodeDrainTimeout: string
  settings:
    storage:
      classes: [string]
```

```
      defaultClass: string
    network:
      cni:
        name: string
      pods:
        cidrBlocks: [string]
      services:
        cidrBlocks: [string]
      serviceDomain: string
      proxy:
        httpProxy: string
        httpsProxy: string
        noProxy: [string]
      trust:
        additionalTrustedCAs:
          - name: string
            data: string
```

TanzuKubernetesCluster v1alpha3 API – 주석 처리됨

주석이 달린 규격에는 각 필드에 대한 설명서와 함께 v1alpha3 API를 사용하여 TanzuKubernetesCluster를 프로비저닝하는 데 사용할 수 있는 모든 매개 변수가 나열됩니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
#valid config key must consist of alphanumeric characters, '-', '_' or '.'
#metadata defines cluster information
metadata:
  #name for this Tanzu Kubernetes cluster
  name: string
  #namespace vSphere Namespace where to provision this cluster
  namespace: string
  #Use annotation to provision non-default OS for the VM nodes
  #PhotonOS is the default OS; use "ubuntu" to specify Ubuntu TKR
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=string
#spec defines cluster configuration
spec:
  #topology describes the number, purpose, organization of nodes
  #and the resources allocated for each
  #nodes are grouped into pools based on their purpose
  #controlPlane is special kind of a node pool
  #nodePools is for groups of worker nodes
  #each node pool is homogeneous: its nodes have the same
  #resource allocation and use the same storage
  topology:
    #controlPlane defines the topology of the cluster
    #controller, including the number of nodes and
    #the resources allocated for each
    #control plane must have an odd number of nodes
    controlPlane:
      #replicas is the number of nodes in the pool
      #the control plane can have 1 or 3 nodes
      #NOTE: production deployments require 3 control plane nodes
```

```
    #defaults to 1 if nil (empty)
    replicas: int32
    #vmClass is the name of the VirtualMachineClass
    #which describes the virtual hardware settings
    #to be used for each node in the node pool
    #vmClass controls the CPU and memory available
    #to the node and the requests and limits on
    #those resources; to list available vm classes run
    #kubectl get virtualmachineclass
    vmClass: string
    #storageClass to be used for storage of the disks
    #which store the root filesystems of the nodes
    #to list available storage classes run
    #kubectl describe storageclasses
    storageClass: string
    #volumes is the optional set of PVCs
    #to create and attach to each control plane node
    volumes:
      #name of the PVC to be used as the suffix (node.name)
      - name: string
        #mountPath is the directory where the volume
        #device is mounted; takes the form /dir/path
        mountPath: string
        #capacity is the PVC capacity
        capacity:
          #storage to be used for the disk
          #volume; if not specified defaults to
          #spec.controlPlane.storageClass
          storage: size in GiB
    #tkr.reference.name is the TKR NAME
    #to be used by control plane nodes
    #format is v1.27.11---vmware.1-fips.1-tkg.2
    #currently all tkr.reference.name fields must match
    tkr:
      reference:
        name: string
    #nodeDrainTimeout is the total amount of time
    #the controller will spend draining a node
    #the default value is 0 which means the node is
    #drained without any time limit
    nodeDrainTimeout: string
  #nodePools is an array that describes a group of
  #worker nodes in the cluster with the same configuration
  nodePools:
  #name of the worker node pool
  #must be unique in the cluster
  - name: string
    #failureDomain is the name of a vSphere Zone
    #failureDomain is required for multi-zoned Supervisor
    #in a multi-zoned Supervisor, you will have 3 node pools
    #each refernecing a different failureDomain zone name
    #refer to the examples
    failureDomain: string
    #labels are an optional map of string keys and values
    #to organize and categorize objects
```

```
      #propagated to the created nodes
      labels: map[string]string
      #taints specifies optional taints to register the
      #Node API object with; user-defined taints are
      #propagated to the created nodes
      taints:
        #key is the taint key to be applied to a node
        -  key: string
        #value is the taint value corresponding to the key
           value: string
        #effect is the effect of the taint on pods
        #that do not tolerate the taint; valid effects are
        #NoSchedule, PreferNoSchedule, NoExecute
           effect: string
        #timeAdded is the time when the taint was added
        #only written by the system for NoExecute taints
           timeAdded: time
      #replicas is the number of nodes in the pool
      #worker nodePool can have from 0 to 150 nodes
      #value of nil means the field is not reconciled,
      #allowing external services like autoscalers
      #to choose the number of nodes for the nodePool
      #by default CAPI's MachineDeployment will pick 1
      #NOTE: a cluster provisioned with 0 worker nodes/nodepools
      #is not assigned any load balancer services
      replicas: int32
      #vmClass is the name of the VirtualMachineClass
      #which describes the virtual hardware settings
      #to be used for each node in the pool
      #vmClass controls the CPU and memory available
      #to the node and the requests and limits on
      #those resources; to list available vm classes run
      #kubectl get virtualmachineclass
      vmClass: string
      #storageClass to be used for storage of the disks
      #which store the root filesystems of the nodes
      #to list available storage classes run
      #kubectl describe ns
      storageClass: string
      #volumes is the optional set of PVCs to create
      #and attach to each node for high-churn worker node
      #components such as the container runtime
      volumes:
        #name of this PVC to be used as the suffix (node.name)
        - name: string
          #mountPath is the directory where the volume
          #device is mounted; takes the form /dir/path
          mountPath: string
          #capacity is the PVC capacity
          capacity:
            #storage to be used for the disk
            #volume; if not specified defaults to
            #topology.nodePools[*].storageClass
            storage: size in GiB
      #tkr.reference.name points to the TKR NAME
```

```
      #to be used by spec.topology.nodePools[*] nodes
      #format is v1.27.11---vmware.1-fips.1-tkg.2
      #currently all tkr.reference.name fields must match
      tkr:
        reference:
          name: string
      #nodeDrainTimeout is the total amount of time
      #the controller will spend draining a node
      #the default value is 0 which means the node is
      #drained without any time limit
      nodeDrainTimeout: string
  #settings are optional runtime configurations
  #for the cluster, including persistent storage
  #for pods and node network customizations
  settings:
    #storage defines persistent volume (PV) storage entries
    #for container workloads; note that the storage used for
    #node disks is defined by topology.controlPlane.storageClass
    #and by spec.topology.nodePools[*].storageClass
    storage:
      #classes is a list of persistent volume (PV) storage
      #classes to expose for container workloads on the cluster
      #any class specified must be associated with the
      #vSphere Namespace where the cluster is provisioned
      #if omitted, all storage classes associated with the
      #namespace will be exposed in the cluster
      classes: [string]
      #defaultClass treats the named storage class as the default
      #for the cluster; because all namespaced storage classes
      #are exposed if specific classes are not named,
      #classes is not required to specify a defaultClass
      #many workloads, including TKG Extensions and Helm,
      #require a default storage class
      #if omitted, no default storage class is set
      defaultClass: string
    #netowrk defines custom networking for cluster workloads
    network:
      #cni identifies the CNI plugin for the cluster
      #use to override the default CNI set in the
      #tkgservicesonfiguration spec, or when customizing
      #network settings for the default CNI
      cni:
        #name is the name of the CNI plugin to use
        #supported values are antrea, calico, antrea-nsx-routed
        name: string
      #pods configures custom networks for pods
      #defaults to 192.168.0.0/16 if CNI is antrea or calico
      #defaults to empty if CNI is antrea-nsx-routed
      #custom subnet size must equal or exceed /24
      #use caution before seeting CIDR range other than /16
      #cannot overlap with Supervisor workload network
      pods:
        #cidrBlocks is an array of network ranges
        #multiple ranges may not be supported by all CNI plugins
        cidrBlocks: [string]
```

```
    #services configures custom network for services
    #defaults to 10.96.0.0/12
    #cannot overlap with Supervisor workload network
    services:
      #cidrBlocks is an array of network ranges
      #multiple ranges many not be supported by all CNI plugins
      cidrBlocks: [string]
    #serviceDomain specifies the service domain for the cluster
    #defaults to cluster.local
    serviceDomain: string
    #proxy configures proxy server to be used inside the cluster
    #if omitted no proxy is configured
    proxy:
      #httpProxy is the proxy URI for HTTP connections
      #to endpoints outside the cluster
      #takes form http://<user>:<pwd>@<ip>:<port>
      httpProxy: string
      #httpsProxy is the proxy URL for HTTPS connections
      #to endpoints outside the cluster
      #takes the frorm http://<user>:<pwd>@<ip>:<port>
      httpsProxy: string
      #noProxy is the list of destination domain names, domains,
      #IP addresses, and other network CIDRs to exclude from proxying
      #must include Supervisor Cluster Pod, Egress, Ingress CIDRs
      noProxy: [string]
    #trust configures additional certificates for the cluster
    #if omitted no additional certificate is configured
    trust:
      #additionalTrustedCAs are additional trusted certificates
      #can be additional CAs or end certificates
      additionalTrustedCAs:
        #name is the name of the additional trusted certificate
        #must match the name used in the filename
        - name: string
          #data holds the contents of the additional trusted cert
          #PEM Public Certificate data as a base64-encoded string
          #such as LS0tLS1C...LS0tCg== where "..." is the
          #middle section of the long base64-encoded string
          data: string
```

v1alpha3 예시: 기본 TanzuKubernetesCluster

v1alpha3 API를 사용하여 기본 TanzuKubernetesCluster를 프로비저닝하려면 예시 YAML을 참조하십시오.

v1alpha3 예시: 기본 TanzuKubernetesCluster

예시 YAML은 v1alpha3 API를 사용하여 기본 TanzuKubernetesCluster를 프로비저닝합니다.

이 예시에서는 TKC를 프로비저닝하는 데 필요한 최소 구성을 보여줍니다. 기본 네트워크 및 스토리지 설정이 사용되므로 YAML에서 제외됩니다. 참조된 TKR은 제어부 및 작업자 노드 둘 다에 사용됩니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-default
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - replicas: 3
      name: worker
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
```

v1alpha3 예시: 기본 스토리지 및 노드 볼륨이 있는 TKC

노드 볼륨에 대한 기본 스토리지 클래스 및 사용자 지정 설정이 포함된 v1alpha3 API를 사용하여 TanzuKubernetesCluster를 프로비저닝하려면 예시 YAML을 참조하십시오.

v1alpha3 예시: 노드 볼륨 및 기본 스토리지가 있는 TKC

예시 YAML은 v1alpha3 API를 사용하여 사용자 지정 TanzuKubernetesCluster를 프로비저닝합니다.

이 예시에서는 다음과 같은 선택적 사용자 지정에 유의하십시오. 자세한 내용은 TanzuKubernetesCluster v1alpha3 API을 참조하십시오.

- 클러스터는 Helm 및 Tanzu 패키지에 의해 배포된 워크로드와 같은 일부 도구에 필요한 기본 스토리지 클래스로 프로비저닝됩니다.
- 작업자 노드 볼륨은 `containerd` 및 `kubelet`과 같이 변동률이 높은 구성 요소에 대해 선언됩니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-custom-storage
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
```

```
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - replicas: 3
      name: worker-np
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
      volumes:
      - name: containerd
        mountPath: /var/lib/containerd
        capacity:
          storage: 50Gi
      - name: kubelet
        mountPath: /var/lib/kubelet
        capacity:
          storage: 50Gi
  settings:
    storage:
      defaultClass: tkg-storage-policy
```

v1alpha3 예시: 사용자 지정 네트워크가 있는 TKC

사용자 지정 네트워크 설정과 함께 v1alpha3 API를 사용하여 TanzuKubernetesCluster를 프로비저닝하려면 예시 YAML을 참조하십시오.

v1alpha3 예시: 사용자 지정 네트워크 설정이 있는 TKC

네트워크는 다음과 같이 사용자 지정됩니다. 자세한 내용은 TanzuKubernetesCluster v1alpha3 API을 참조하십시오.

- 기본 Antrea 대신 Calico CNI가 사용됨
- 포드 및 서비스에 대한 기본값이 아닌 서브넷이 사용됨
- 프록시 서버 및 TLS 인증서가 선언됨

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-custom-network
  namespace: tkg2-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - name: worker
```

```
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
      volumes:
      - name: containerd
        mountPath: /var/lib/containerd
        capacity:
          storage: 50Gi
      - name: kubelet
        mountPath: /var/lib/kubelet
        capacity:
          storage: 50Gi
  settings:
    storage:
      defaultClass: tkg-storage-policy
    network:
      cni:
        name: calico
      services:
        cidrBlocks: ["172.16.0.0/16"]
      pods:
        cidrBlocks: ["192.168.0.0/16"]
      serviceDomain: cluster.local
      proxy:
        httpProxy: http://<user>:<pwd>@<ip>:<port>
        httpsProxy: http://<user>:<pwd>@<ip>:<port>
        noProxy: [10.246.0.0/16,192.168.144.0/20,192.168.128.0/20]
      trust:
        additionalTrustedCAs:
          - name: CompanyInternalCA-1
            data: LS0tLS1C...LS0tCg==
          - name: CompanyInternalCA-2
            data: MTLtMT1C...MT0tPg==
```

TKC 포드 네트워크 사용자 지정을 위한 고려 사항

클러스터 규격 설정 `spec.settings.network.pods.cidrBlocks`의 기본값은 192.168.0.0/16입니다.

사용자 지정하는 경우 최소 포드 CIDR 블록 크기는 /24입니다. 하지만 `pods.cidrBlocks` 서브넷 마스크를 /16 이상으로 제한하는 경우에는 주의해야 합니다.

TKG는 각 클러스터 노드에 `pods.cidrBlocks`에서 분할된 /24 서브넷을 할당합니다. 이러한 할당은 클러스터의 노드 CIDR에 대한 서브넷 마스크 크기를 설정하는 NodeCIDRMaskSize라는 Kubernetes 컨트롤러 관리자 > NodeIPAMController 매개 변수에 의해 정의합니다. IPv4의 경우 기본 노드 서브넷 마스크는 /24입니다.

클러스터의 각 노드는 `pods.cidrBlocks`에서 /24 서브넷을 가져오기 때문에 프로비저닝 중인 클러스터에 너무 제한적인 서브넷 마스크 크기를 사용하면 노드 IP 주소가 부족해질 수 있습니다.

다음 노드 제한은 Antrea 또는 Calico CNI로 프로비저닝된 Tanzu Kubernetes 클러스터에 적합합니다.

/16 == 최대 150개 노드(ConfigMax당)

/17 == 최대 128개 노드

/18 == 최대 64개 노드

/19 == 최대 32개 노드

/20 == 최대 16개 노드

/21 == 최대 8개 노드

/22 == 최대 4개 노드

/23 == 최대 2개 노드

/24 == 최대 1개 노드

v1alpha3 예시: Ubuntu TKR이 있는 TKC

클러스터 노드에 Ubuntu OS를 사용하는 TanzuKubernetesCluster 클러스터를 프로비저닝하려면 여기에 제공된 예시 YAML을 참조하십시오. 이러한 클러스터는 vGPU 워크로드에 사용할 수 있습니다.

v1alpha3 예시: Ubuntu TKR이 있는 TKC

기본적으로 명명된 TKR의 PhotonOS 버전은 TKG 클러스터 노드에 사용됩니다. 참조된 TKR이 OSImage 형식을 지원하고 Ubuntu OS 버전을 사용할 수 있는 경우 `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` 주석을 사용하여 TKR의 Ubuntu OS 버전을 지정합니다. OSImage 형식에 대한 자세한 내용은 TKr 운영 체제 이미지 형식 항목을 참조하십시오.

AI/ML 워크로드에는 Ubuntu TKR이 필요합니다. 각 작업자 노드 풀에는 각각 70GiB 용량의 containerd 런타임 및 kubelet에 대한 별도의 볼륨이 있습니다. 컨테이너 기반 AI/ML 워크로드에는 이 크기의 별도 볼륨을 제공하는 것이 좋습니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-ubuntu-gpu
  namespace: tkg-cluster-ns
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
   topology:
     controlPlane:
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: guaranteed-large
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
     nodePools:
     - name: nodepool-a100-primary
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: vgpu-a100
       tkr:
```

```
          reference:
            name: v1.25.7---vmware.3-fips.1-tkg.1
        volumes:
        - name: containerd
          mountPath: /var/lib/containerd
          capacity:
            storage: 70Gi
        - name: kubelet
          mountPath: /var/lib/kubelet
          capacity:
            storage: 70Gi
      - name: nodepool-a100-secondary
        replicas: 3
        storageClass: tkg-storage-policy
        vmClass: vgpu-a100
        tkr:
          reference:
            name: v1.25.7---vmware.3-fips.1-tkg.1
        volumes:
        - name: containerd
          mountPath: /var/lib/containerd
          capacity:
            storage: 70Gi
        - name: kubelet
          mountPath: /var/lib/kubelet
          capacity:
            storage: 70Gi
    settings:
      storage:
        defaultClass: tkg-storage-policy
      network:
        cni:
         name: antrea
        services:
         cidrBlocks: ["198.51.100.0/12"]
        pods:
         cidrBlocks: ["192.0.2.0/16"]
        serviceDomain: cluster.local
```

v1alpha3 예시: 여러 vSphere 영역의 TKC

v1alpha3 API를 사용하여 여러 vSphere 영역에 TanzuKubernetesCluster를 프로비저닝하려면 예시 YAML을 참조하십시오.

vSphere 영역 및 장애 도메인

vSphere 영역은 감독자에서 고가용성 TKG 클러스터를 생성하는 방법을 제공합니다. 여러 vSphere 영역에 TKG 클러스터를 프로비저닝하는 경우 각 노드 풀에 장애 도메인을 제공해야 합니다.

각 장애 도메인은 vSphere 영역에 매핑되므로 하나의 vSphere 클러스터와 연결됩니다. 장애 도메인(vSphere 장애 도메인이라고도 함)은 vSphere 영역을 생성할 때 vSphere 관리자가 정의하고 관리합니다. TKG 클러스터에 사용하는 스토리지 프로파일은 `zonal`로 구성되어야 합니다. TKG 서비스 클러스터에 대한 vSphere 스토리지 정책 생성의 내용을 참조하십시오.

복제본이 있는 포드를 감독자의 TKG 클러스터에 배포하면 포드 인스턴스가 여러 vSphere 영역에 자동으로 분산됩니다. TKG 클러스터에 POD를 배포하는 동안 영역 세부 정보를 제공할 필요가 없습니다.

TKG 환경에서 vSphere 영역의 가용성을 확인하려면 TKG 클러스터를 프로비저닝할 vSphere 네임스페이스에서 다음 명령 중 하나를 실행합니다.

```
kubectl get vspherezones
```

```
kubectl get availabilityzones
```

두 명령 모두 `system:authenticated` 사용자가 사용할 수 있습니다. vSphere 영역은 감독자 범위의 리소스이므로 네임스페이스를 지정할 필요가 없습니다.

v1alpha3 예시: 여러 vSphere 영역의 TKC

예시 YAML은 여러 vSphere 영역에 TKG 클러스터를 프로비저닝합니다.

이 예시에서는 각 nodePool의 `failureDomain` 매개 변수에 vSphere 영역을 지정합니다. 매개 변수의 값은 vSphere 영역의 이름입니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-zoned
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg2-storage-policy-zonal
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
      - name: nodepool-a01
        replicas: 3
        vmClass: guaranteed-medium
        storageClass: tkg-storage-policy-zonal
        failureDomain: az1
      - name: nodepool-a02
        replicas: 3
        vmClass: guaranteed-medium
        storageClass: tkg-storage-policy-zonal
        failureDomain: az2
      - name: nodepool-a03
        replicas: 3
        vmClass: guaranteed-medium
        storageClass: tkg-storage-policy-zonal
        failureDomain: az3
  settings:
    storage:
      defaultClass: tkg-storage-policy-zonal
```

```
    network:
      cni:
        name: antrea
      services:
        cidrBlocks: ["198.51.100.0/12"]
      pods:
        cidrBlocks: ["192.0.2.0/16"]
      serviceDomain: cluster.local
```

v1alpha3 예시: 라우팅 가능한 포드 네트워크가 있는 TKC

감독자에서 라우팅 가능한 네임스페이스 네트워크를 구성하고 클러스터의 CNI로 `antrea-nsx-routed`를 지정하여 라우팅 가능한 포드 네트워킹으로 TanzuKubernetesCluster를 생성할 수 있습니다.

라우팅 가능한 포드 네트워킹 정보

`antrea` 또는 `calico` CNI 플러그인을 사용하여 Tanzu Kubernetes 클러스터를 프로비저닝하면 기본 포드 네트워크 `192.168.0.0/16`이 생성됩니다. 이 서브넷은 클러스터 내에서만 고유하고 네트워크에서 라우팅할 수 없는 개인 주소 공간입니다.

TKG v1alpha3 API는 `antrea-nsx-routed` CNI 플러그인을 사용하여 라우팅 가능한 포드 네트워킹을 지원합니다. 이 네트워크 인터페이스는 TKG 클러스터에 대해 라우팅 가능한 포드 네트워크를 지원하도록 구성된 사용자 지정된 Antrea 플러그인입니다. 클러스터 규격에서 포드 CIDR 블록 필드는 명시적으로 null이어야 합니다. 그래야 IPAM(IP 주소 관리)이 감독자에서 처리됩니다. 아래 예시를 참조하십시오.

라우팅 가능한 포드 네트워킹을 사용하도록 설정하면 클러스터 외부의 클라이언트에서 직접 포드의 주소를 지정할 수 있습니다. 또한 외부 네트워크 서비스 및 서버가 소스 포드를 식별하고 IP 주소를 기반으로 정책을 적용할 수 있도록 포드 IP 주소가 보존됩니다. 지원되는 트래픽 패턴에는 다음이 포함됩니다.

- TKG 클러스터 포드와 동일한 vSphere 네임스페이스에 있는 vSphere 포드 간에 트래픽이 허용됩니다.
- TKG 클러스터 포드와 다른 vSphere 네임스페이스에 있는 vSphere 포드 간에 트래픽이 삭제됩니다.
- 감독자 제어부 노드는 TKG 클러스터 포드에 연결할 수 있습니다.
- TKG 클러스터 포드는 외부 네트워크에 연결할 수 있습니다.
- 외부 네트워크가 TKG 클러스터 포드에 연결할 수 없습니다. 클러스터 노드의 DFW(분산 방화벽) 격리 규칙에 따라 트래픽이 삭제됩니다.

라우팅 가능한 포드 네트워크 생성: 감독자 구성

라우팅 가능한 포드 네트워크를 생성하려면 감독자 및 TKG 클러스터에 대한 구성이 필요합니다.

참고 라우팅 가능한 포드 네트워킹을 사용하려면 NSX로 감독자를 구성해야 합니다. VDS 네트워킹에는 라우팅 가능한 포드를 사용할 수 없습니다.

감독자에서 라우팅 가능한 포드 네트워크를 구성하려면 다음을 수행합니다.

1 새 vSphere 네임스페이스를 생성합니다.

TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.

2 **감독자 네트워크 설정 재정의** 확인란 옵션을 선택합니다.

지침은 vSphere 네임스페이스에 대한 워크로드 네트워크 설정 재정의 항목을 참조하십시오.

3 다음과 같이 라우팅 가능한 포드 네트워크를 구성합니다.

필드	설명
NAT 모드	라우팅 가능한 서브넷을 사용 중이므로 NAT(네트워크 주소 변환)를 사용하지 않도록 설정하려면 이 옵션을 선택 취소합니다.
네임스페이스 네트워크 CIDR	네임스페이스 네트워크 CIDR은 vSphere 네임스페이스의 IP 풀로 작동하는 서브넷입니다. 네임스페이스 서브넷 접두사는 해당 IP 풀에서 분할된 후속 CIDR 블록의 크기를 설명합니다. 이 필드를 IP 주소/비트 형식의 라우팅 가능한 IP 서브넷으로 채웁니다(예: 10.0.0.6/16). NCP는 네트워크에 대해 지정된 IP 블록에서 하나 이상의 IP 풀을 생성합니다. 최소한 /23 서브넷 크기를 지정해야 합니다. 예를 들어 /23 라우팅 가능한 서브넷을 /28 서브넷 접두사와 함께 지정하면 6 노드 클러스터에 충분한 32개의 서브넷을 가져올 수 있습니다. 접두사가 /28인 /24 서브넷은 2개의 서브넷만 가져오므로 충분하지 않습니다.
네임스페이스 서브넷 접두사	네임스페이스 서브넷 접두사는 네임스페이스 네트워크 IP 풀에서 분할된 후속 CIDR 블록의 크기를 설명합니다. 예를 들어 /28 형식으로 서브넷 접두사를 지정합니다.

4 **생성**을 클릭하여 라우팅 가능한 포드 네트워크를 생성합니다.

라우팅 가능한 포드 네트워크 생성: TKG 클러스터 구성

다음 예제 YAML은 라우팅 가능한 포드 네트워크로 클러스터를 구성하는 방법을 보여줍니다.

클러스터 규격은 `antrea-nsx-routed`를 라우팅 가능한 포드 네트워킹을 사용하도록 설정하기 위한 CNI로 선언합니다. `antrea-nsx-routed`가 지정된 경우 NSX-T 네트워킹이 사용되지 않으면 클러스터 프로비저닝이 실패합니다.

CNI가 `antrea-nsx-routed`로 지정되면 `pods.cidrBlock` 필드는 비어 있어야 합니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-routable-pods
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
```

```
      vmClass: guaranteed-large
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
  settings:
    storage:
      defaultClass: tkg-storage-policy
    network:
      #antrea-nsx-routed is the required CNI
      cni:
        name: antrea-nsx-routed
      services:
        cidrBlocks: ["10.97.0.0/24"]
      #pods.cidrBlocks must be null (empty)
      pods:
        cidrBlocks:
      serviceDomain: cluster.local
```

v1alpha3 예: SSL/TLS에 대해 신뢰할 수 있는 추가 CA 인증서가 있는 TKC

SSL/TLS에 대해 신뢰할 수 있는 추가 CA 인증서가 있는 v1alpha3 API를 사용하여 TanzuKubernetesCluster를 프로비저닝하려면 예제 YAML을 참조하십시오.

v1alpha3 예: 신뢰할 수 있는 추가 CA 인증서가 있는 TKC

클러스터는 다음과 같이 사용자 지정됩니다. 자세한 내용은 TanzuKubernetesCluster v1alpha3 API을 참조하십시오.

- 신뢰할 수 있는 추가 CA 인증서가 클러스터 규격의 `network.trust.additionalTrustedCAs` 섹션에 선언됩니다.
- `additionalTrustedCAs` 필드는 이름-값 쌍의 어레이입니다.
  - `name` 필드는 사용자 정의 문자열입니다.
  - `data` 값은 base64로 인코딩된 PEM 형식의 CA 인증서 컨텐츠입니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-additional-trusted-cas
  namespace: tkgs-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkgs-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - name: worker
```

```
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkgs-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
  settings:
    storage:
      defaultClass: tkgs-storage-policy
    network:
      trust:
        additionalTrustedCAs:
          - name: CompanyInternalCA-1
            data: LS0tLS1C...LS0tCg==
          - name: CompanyInternalCA-2
            data: MTLtMT1C...MT0tPg==
```

절차: 새 클러스터

새 TKGS 클러스터에 신뢰할 수 있는 CA 인증서를 하나 이상 추가로 포함하려면 다음 절차를 완료합니다.

1 `additionalTrustedCAs` 필드를 하나 이상의 CA 인증서에 대한 이름 및 데이터 값으로 채웁니다.

2 평소처럼 클러스터를 프로비저닝합니다.

Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

3 클러스터가 성공적으로 프로비저닝되면 추가한 CA 인증서를 클러스터에서 신뢰합니다.

절차: 기존 클러스터

기존 클러스터에 신뢰할 수 있는 CA 인증서를 하나 이상 더 추가하려면 다음 절차를 완료합니다.

1 kubectl 편집을 구성했는지 확인합니다.

Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

2 클러스터 규격을 편집합니다.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
```

3 `network.trust.additionalTrustedCAs` 섹션을 규격에 추가합니다.

4 `additionalTrustedCAs` 필드를 하나 이상의 CA 인증서에 대한 이름 및 데이터 값으로 채웁니다.

5 텍스트 편집기에서 변경 내용을 저장하고 kubectl이 변경 내용을 등록했는지 확인합니다.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-name edited
```

6 클러스터에 대해 롤링 업데이트가 시작되면 신뢰할 수 있는 CA 인증서가 추가됩니다.

TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해의 내용을 참조하십시오.

신뢰할 수 있는 추가 CA 인증서 확인

클러스터에 추가된 신뢰할 수 있는 추가 CA 인증서가 클러스터의 kubeconfig 파일에 포함됩니다.

신뢰할 수 있는 추가 CA 인증서 문제 해결

추가적인 신뢰할 수 있는 CA 오류 문제 해결의 내용을 참조하십시오.

사용 사례

가장 일반적인 사용 사례는 컨테이너 레지스트리에 연결하기 위해 신뢰할 수 있는 다른 CA를 추가하는 것입니다. TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합의 내용을 참조하십시오.

TKG 서비스 클러스터 운영

8

이 섹션에서는 TKG 서비스 클러스터를 운영하는 방법에 대한 항목을 제공합니다.

다음으로 아래 항목을 읽으십시오.

- Kubectl용 텍스트 편집기 구성
- Kubectl을 사용하여 수동으로 클러스터 확장/축소
- vSphere Client를 사용하여 TKG 클러스터 상태 모니터링
- kubectl을 사용하여 TKG 클러스터 상태 모니터링
- Kubectl을 사용하여 TKG 클러스터 준비 상태 확인
- Kubectl을 사용하여 TKG 클러스터 시스템 상태 확인
- Kubectl을 사용하여 TKG 클러스터 상태 확인
- Kubectl을 사용하여 TKG 클러스터 볼륨 상태 확인
- Tanzu Kubernetes Grid 클러스터의 볼륨 상태 모니터링
- vSphere Client를 사용하여 영구 볼륨 모니터링
- Kubectl을 사용하여 TKG 클러스터 암호 얻기
- Kubectl을 사용하여 TKG 클러스터 네트워킹 확인
- Kubectl을 사용하여 TKG 클러스터 작업 확인
- TKG 클러스터 수명 주기 상태 보기
- Kubectl을 사용하여 TKG 클러스터의 리소스 계층 구조 보기
- v1beta1 클러스터에 대한 MachineHealthCheck 구성

Kubectl용 텍스트 편집기 구성

TKG 클러스터를 운영하고 유지 보수하려면 kubectl에 대한 기본 텍스트 편집기를 구성합니다.

kubectl edit 명령 사용

TKG 클러스터를 프로비저닝한 후에는 운영하고 유지 보수합니다. 일반적인 작업에는 클러스터 노드 크기 조정 및 TKR 버전 업데이트가 포함됩니다. 이러한 작업을 수행하려면 kubectl edit 명령을 사용하여 클러스터 매니페스트를 업데이트합니다.

`kubectl edit` *`CLUSTER-KIND/CLUSTER-NAME`* 명령은 KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 텍스트 편집기에서 클러스터 매니페스트를 엽니다. 매니페스트 변경 내용을 저장하면 `kubectl`은 편집 내용이 기록되었다고 보고하고 이 변경 내용으로 클러스터가 업데이트됩니다.

예:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

변경 내용을 취소하려면 저장하지 않고 편집기를 닫습니다.

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
Edit cancelled, no changes made.
```

kubectl edit 구성

`kubectl edit` 명령을 사용하려면 Linux에서 EDITOR 환경 변수가 설정됩니다. 그렇지 않으면 KUBE_EDITOR 환경 변수를 생성하고 선호하는 텍스트 편집기를 변수 값으로 지정합니다. watch 플래그(`-w`)를 추가하여 변경 내용을 커밋(저장)할 때 kubectl이 알 수 있도록 합니다.

사용 중인 운영 체제의 특정 지침을 참조하십시오.

Linux

Linux(예: Ubuntu)에서 `kubectl edit`을 구성하려면 기본 명령줄 `EDITOR`는 Vim입니다. 이 경우 `kubectl edit` 명령을 사용하는 데 추가 작업이 필요하지 않습니다.

다른 텍스트 편집기를 사용하려면, 선호하는 텍스트 편집기의 경로로 값이 설정된 `KUBE_EDITOR`라는 환경 변수를 생성합니다.

Mac OS

Mac OS에서는 `kubectl edit`를 구성하려면 선호하는 텍스트 편집기의 경로로 값이 설정된 `KUBE_EDITOR` 라는 환경 변수를 생성합니다. 값에 wait 플래그(`--wait` 또는 바로 가기 `-w`)를 추가하면 편집기에서 변경 내용을 커밋(저장)한 시점을 알 수 있습니다.

예를 들어, `.bash_profile`에 다음을 추가하면 Sublime이 kubectl용 기본 텍스트 편집기로 설정되고, 사용자가 변경 사항을 저장하면 편집기에서 알 수 있도록 wait 플래그가 포함됩니다.

```
export KUBE_EDITOR="/Applications/Sublime.app/Contents/SharedSupport/bin/subl -w"
```

Windows

Windows에서는 `kubectl edit`를 구성하려면 선호하는 텍스트 편집기의 경로로 값이 설정된 `KUBE_EDITOR`라는 시스템 환경 변수를 생성합니다. 이 값에 watch 플래그(`-w`)를 추가합니다.

예를 들어, 다음 환경 변수는 Visual Studio Code를 kubectl용 기본 텍스트 편집기로 설정하고, 사용자가 변경 사항을 저장할 때 Kubernetes가 알 수 있도록 watch 플래그를 포함합니다.

```
KUBE_EDITOR=code -w
```

Sublime을 Windows에서 kubectl 편집기로 구성하려면 Sublime 프로그램 디렉토리를 시스템 경로에 추가하고 Sublime 실행 파일의 시스템 변수를 생성합니다. 예:

시스템 경로 추가:

```
C:\Program Files\Sublime Text 3\
```

시스템 변수 이름 및 값:

```
KUBE_EDITOR=sublime_text.exe -w
```

Kubectl을 사용하여 수동으로 클러스터 확장/축소

노드 수를 변경하여 수평으로 또는 노드를 호스팅하는 가상 시스템 클래스를 변경하여 수직으로 TKG 서비스 클러스터를 확장/축소할 수 있습니다. 클러스터 노드에 연결된 볼륨을 확장/축소할 수도 있습니다.

지원되는 수동 확장/축소 작업

다음 표에는 TKG 클러스터에 대해 지원되는 확장/축소 작업이 나열되어 있습니다.

표 8-1. TKGS 클러스터에 대해 지원되는 확장/축소 작업

노드	수평 확장	수평 축소	수직 확장/축소	볼륨 확장/축소
제어부	예	아니요	예	예*
작업자	예	예	예	예

다음 고려 사항에 유의하십시오.

- 제어부 노드의 수는 홀수(1 또는 3)여야 합니다. 제어부 확장은 지원되지만 제어부 축소는 지원되지 않습니다. 제어부 확장의 내용을 참조하십시오.
- 클러스터 노드를 수직으로 확장/축소하는 동안은 사용 가능한 리소스가 부족하여 노드에서 워크로드가 더 이상 실행되지 못할 수 있습니다. 따라서 일반적으로 선호되는 접근 방식은 수평 확장/축소입니다. 작업자 노드 확장의 내용을 참조하십시오.
- VM 클래스는 변경할 수 없습니다. TKG 클러스터에서 사용되는 VM 클래스를 편집한 후 해당 클러스터를 확장하면, 새 클러스터 노드는 업데이트된 클래스 정의를 사용하지만 기존 클러스터 노드는 초기 클래스 정의를 계속 사용하여 불일치가 발생합니다. VM 클래스 정보의 내용을 참조하십시오.

- 작업자 노드 볼륨은 프로비저닝 후 변경할 수 있으며, 마찬가지로 제어부 노드도 vSphere 8 U3 이상 및 호환되는 TKr을 사용하는 경우 변경할 수 있습니다. 클러스터 노드 볼륨 확장/축소의 내용을 참조하십시오.

확장/축소 사전 요구 사항: Kubectl 편집 구성

TKG 클러스터를 확장/축소하려면 `kubectl edit CLUSTER-KIND/CLUSTER-NAME` 명령을 사용하여 클러스터 매니페스트를 업데이트합니다. 매니페스트 변경 내용을 저장하면 클러스터가 변경 내용으로 업데이트됩니다. Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

예:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

변경 내용을 취소하려면 저장하지 않고 편집기를 닫습니다.

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
Edit cancelled, no changes made.
```

제어부 확장

제어부 노드 수를 1에서 3으로 늘려서 TKG 클러스터를 확장합니다.

참고 운영 클러스터에는 3개의 제어부 노드가 필요합니다.

1 감독자에 로그인합니다.

```
kubectl vsphere login --server=SUPERVISOR-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG 클러스터를 실행 중인 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 네임스페이스에서 실행 중인 Kubernetes 클러스터를 나열합니다.

다음 구문을 사용합니다.

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

예를 들어 v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

예를 들어 v1beta1 API 클러스터의 경우:

```
kubectl get cluster -n tkg-cluster-ns
```

4 대상 클러스터에서 실행 중인 노드 수를 구합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

TKG 클러스터에는 제어부 노드 1개와 작업자 노드 3개가 있습니다.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    1               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

v1beta1 API 클러스터의 경우:

```
kubectl get cluster tkg-cluster-1
```

5 `kubectl edit` 명령을 사용하여 편집할 클러스터 매니페스트를 로드합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API 클러스터의 경우:

```
kubectl edit cluster/tkg-cluster-1
```

KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 텍스트 편집기에서 클러스터 매니페스트가 열립니다.

6 매니페스트의 `spec.topology.controlPlane.replicas` 섹션에서 제어부 노드 수를 1에서 3으로 늘립니다.

v1alpha3 API 클러스터의 경우:

```
...
spec:
  topology:
    controlPlane:
      replicas: 1
...
```

```
...
spec:
  topology:
    controlPlane:
      replicas: 3
...
```

v1beta1 API 클러스터의 경우:

```
...
spec:
  ...
  topology:
    class: tanzukubernetescluster
    controlPlane:
      metadata: {}
      replicas: 1
    variables:
...
```

```
...
spec:
  ...
  topology:
    class: tanzukubernetescluster
    controlPlane:
      metadata: {}
      replicas: 3
    variables:
...
```

7 텍스트 편집기에서 파일을 저장하여 변경 내용을 적용합니다. (취소하려면 저장하지 않고 편집기를 닫습니다.)

매니페스트 변경 내용을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 감독자의 가상 시스템 서비스는 새 제어부 노드를 프로비저닝합니다.

8 새 노드가 추가되었는지 확인합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

확장된 제어부에 이제 3개의 노드가 있습니다.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    3               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

v1beta1 API 클러스터의 경우:

```
kubectl get cluster tkg-cluster-1
```

작업자 노드 확장

작업자 노드 수를 늘려서 TKG 클러스터를 확장할 수 있습니다.

1 감독자에 로그인합니다.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG 클러스터를 실행 중인 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 네임스페이스에서 실행 중인 Kubernetes 클러스터를 나열합니다.

다음 구문을 사용합니다.

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

예를 들어 v1alph3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

예를 들어 v1beta1 API 클러스터의 경우:

```
kubectl get cluster -n tkg-cluster-ns
```

4 대상 클러스터에서 실행 중인 노드 수를 구합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

예를 들어 다음 클러스터에는 제어부 노드 3개와 작업자 노드 3개가 있습니다.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    3               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

v1beta1 API 클러스터의 경우:

```
kubectl get cluster tkg-cluster-1
```

5 `kubectl edit` 명령을 사용하여 편집할 클러스터 매니페스트를 로드합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API 클러스터의 경우:

```
kubectl edit cluster/tkg-cluster-1
```

KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 텍스트 편집기에서 클러스터 매니페스트가 열립니다.

6 대상 작업자 노드 풀에 대한 `spec.topology.nodePools.NAME.replicas` 값을 편집하여 작업자 노드 수를 늘립니다.

v1alpha3 API 클러스터의 경우:

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 3
...
```

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 4
...
```

v1beta1 API 클러스터의 경우:

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
```

```
        metadata: {}
        name: node-pool-1
        replicas: 3
...
```

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
        metadata: {}
        name: node-pool-1
        replicas: 4
...
```

7 변경 내용을 적용하려면 텍스트 편집기에서 파일을 저장합니다. 변경 내용을 취소하려면 저장하지 않고 편집기를 닫습니다.

파일을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 감독자의 가상 시스템 서비스는 새 작업자 노드를 프로비저닝합니다.

8 새 노드가 추가되었는지 확인합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

확장한 후에는 클러스터에 작업자 노드가 4개 있습니다.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster      3               4        v1.24.9---vmware.1-tkg.4
5d12h   True
```

v1beta1 API 클러스터의 경우:

```
kubectl get cluster tkg-cluster-1
```

작업자 노드 축소

작업자 노드 수를 줄여서 TKG 클러스터를 축소할 수 있습니다.

1 감독자에 로그인합니다.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG 클러스터를 실행 중인 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 네임스페이스에서 실행 중인 Kubernetes 클러스터를 나열합니다.

다음 구문을 사용합니다.

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

예를 들어 v1alph3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

예를 들어 v1beta1 API 클러스터의 경우:

```
kubectl get cluster -n tkg-cluster-ns
```

4 대상 클러스터에서 실행 중인 노드 수를 구합니다.

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

5 대상 클러스터에서 실행 중인 노드 수를 구합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

예를 들어 다음 클러스터에는 제어부 노드 3개와 작업자 노드 4개가 있습니다.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster      3               4        v1.24.9---vmware.1-tkg.4
5d12h   True
```

v1beta1 API 클러스터의 경우:

```
kubectl get cluster tkg-cluster-1
```

6 `kubectl edit` 명령을 사용하여 편집할 클러스터 매니페스트를 로드합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API 클러스터의 경우:

```
kubectl edit cluster/tkg-cluster-1
```

KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 텍스트 편집기에서 클러스터 매니페스트가 열립니다.

7 대상 작업자 노드 풀에 대한 `spec.topology.nodePools.NAME.replicas` 값을 편집하여 작업자 노드 수를 줄입니다.

v1alpha3 API 클러스터의 경우:

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 4
...
```

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 3
...
```

v1beta1 API 클러스터의 경우:

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
```

```
        metadata: {}
        name: node-pool-1
        replicas: 4
...
```

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
        metadata: {}
        name: node-pool-1
        replicas: 3
...
```

8 변경 내용을 적용하려면 텍스트 편집기에서 파일을 저장합니다. 변경 내용을 취소하려면 저장하지 않고 편집기를 닫습니다.

파일을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 감독자의 가상 시스템 서비스는 새 작업자 노드를 프로비저닝합니다.

9 작업자 노드가 제거되었는지 확인합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

축소한 후에는 클러스터에 작업자 노드가 3개 있습니다.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    3               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

v1beta1 API 클러스터의 경우:

```
kubectl get cluster tkg-cluster-1
```

클러스터를 수직으로 확장/축소

감독자의 TKG는 클러스터 제어부 및 작업자 노드에 대한 수직 확장/축소를 지원합니다. 클러스터 노드에 사용되는 TKG 서비스 클러스터에서 VM 클래스 사용를 변경하여 TKG 클러스터를 수직으로 확장/축소합니다. 사용하는 VM 클래스는 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에 바인딩되어야 합니다.

감독자의 TKG는 시스템에 내장된 롤링 업데이트 메커니즘을 통해 수직 확장/축소를 지원합니다. `VirtualMachineClass` 정의를 변경하면 시스템은 새 클래스를 사용하여 새 노드를 롤아웃하고 이전 노드를 스핀 다운합니다. 장 9 TKG 서비스 클러스터 업데이트의 내용을 참조하십시오.

1 감독자에 로그인합니다.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG 클러스터를 실행 중인 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 네임스페이스에서 실행 중인 Kubernetes 클러스터를 나열합니다.

다음 구문을 사용합니다.

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

예를 들어 v1alph3 API 클러스터의 경우:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

예를 들어 v1beta1 API 클러스터의 경우:

```
kubectl get cluster -n tkg-cluster-ns
```

4 대상 TKG 클러스터에 describe 명령을 실행하고 VM 클래스를 확인합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl describe tanzukubernetescluster tkg-cluster-1
```

예를 들어 다음 클러스터는 best-effort-medium VM 클래스를 사용합니다.

```
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: best-effort-medium
      ...
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: best-effort-medium
      ...
```

v1beta1 API 클러스터의 경우:

```
kubectl describe cluster tkg-cluster-1
```

예를 들어 다음 클러스터는 best-effort-medium VM 클래스를 사용합니다.

```
...
Topology:
      ...
    Variables:
      ...
      Name:   vmClass
      Value:  best-effort-medium
    ...
```

참고 v1beta1 API 클러스터의 경우 기본적으로 vmClass는 전역적으로 단일 변수로 설정됩니다. 이 설정을 재정의하고 제어부 및 작업자 노드에 대해 다른 VM 클래스를 사용할 수 있습니다. API 참조에서 vmClass의 내용을 참조하십시오.

5 사용 가능한 VM 클래스를 나열하고 설명합니다.

```
kubectl get virtualmachineclass
```

```
kubectl describe virtualmachineclass
```

참고 VM 클래스는 vSphere 네임스페이스에 바인딩되어야 합니다. TKG 서비스 클러스터에서 VM 클래스 사용의 내용을 참조하십시오.

6 대상 클러스터 매니페스트를 편집하기 위해 엽니다.

v1alpha3 API 클러스터의 경우:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API 클러스터의 경우:

```
kubectl edit cluster/tkg-cluster-1
```

KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 텍스트 편집기에서 클러스터 매니페스트가 열립니다.

7 VM 클래스를 변경하여 매니페스트를 편집합니다.

v1alpha3 API 클러스터의 경우, 제어부에 대한 VM 클래스를 `guaranteed-medium`로 변경하고 작업자 노드에 대한 VM 클래스를 `guaranteed-large`로 변경합니다.

```
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      ...
```

```
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: guaranteed-large
      ...
```

v1beta API 클러스터의 경우 VM 클래스를 `guaranteed-large`로 변경합니다.

```
...
Topology:
      ...
    Variables:
      ...
      Name:   vmClass
      Value:  guaranteed-large
    ...
```

8 변경 내용을 적용하려면 텍스트 편집기에서 파일을 저장합니다. 변경 내용을 취소하려면 저장하지 않고 편집기를 닫습니다.

파일을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 감독자의 TKG는 TKG 클러스터의 롤링 업데이트를 수행합니다.

9 TKG 클러스터가 새 VM 클래스로 업데이트되었는지 확인합니다.

v1alpha3 API 클러스터의 경우:

```
kubectl describe tanzukubernetescluster tkg-cluster-1
```

v1beta1 API 클러스터의 경우:

```
kubectl describe cluster tkg-cluster-1
```

클러스터 노드 볼륨 확장/축소

노드에 대한 TKG 클러스터 규격에서 필요에 따라 노드에 대해 하나 이상의 영구 볼륨을 선언할 수 있습니다. 노드 볼륨 선언은 작업자 노드의 컨테이너 런타임 및 kubelet과 같이 변동률이 높은 구성 요소에 유용합니다.

클러스터 생성 후 하나 이상의 노드 볼륨을 추가하거나 변경하려면 다음 고려 사항에 유의하십시오.

볼륨 노드	설명
작업자 노드 볼륨 변경이 허용됩니다.	TKG 클러스터가 프로비저닝된 후 작업자 노드 볼륨을 추가하거나 업데이트할 수 있습니다. 롤링 업데이트를 시작하면 클러스터가 새 볼륨 또는 변경된 볼륨으로 업데이트됩니다. **경고** 새 볼륨 또는 변경된 볼륨으로 작업자 노드를 확장하는 경우 롤링 업데이트 중에 현재 볼륨의 데이터가 삭제됩니다. 다음 설명을 참조하십시오. TKG 클러스터 노드에 대해 선언된 볼륨은 사용 후 삭제로 처리됩니다. TKG 클러스터는 vSphere 네임스페이스에서 PVC(영구 볼륨 할당)를 사용하여 볼륨 용량이 TKG 클러스터의 스토리지 할당량에서 차감되도록 합니다. TKC 볼륨의 용량을 늘리면 Kubernetes CAPI(Cluster API)가 새 PVC를 사용하여 새 작업자를 롤아웃합니다. 이 경우 TKG는 데이터 마이그레이션을 수행하지 않지만 Kubernetes는 그에 따라 워크로드 포드를 (다시)스케줄링합니다.
vSphere 8 U3 이상을 사용하는 경우 제어부 노드 볼륨 변경이 허용됩니다.	vSphere 8 U3 이상 및 호환되는 Tanzu Kubernetes 릴리스를 사용하는 경우 TKG 서비스 클러스터가 프로비저닝되면 제어부 노드 볼륨을 추가하거나 업데이트할 수 있습니다. vSphere 8 U3 이상을 사용하지 않는 경우 CAPI(Kubernetes Cluster API)는 `spec.toplogy.controlPlane.volumes`에 대한 생성 후 변경을 금지합니다. 클러스터 생성 후 제어부 볼륨을 추가하거나 변경하려고 하면 요청이 거부되고 "불변 필드에 대한 업데이트는 허용되지 않습니다."라는 오류 메시지가 표시됩니다.

다음은 선언된 노드 볼륨이 있는 v1alpha3 API를 기반으로 하는 클러스터 규격에서 발췌한 것입니다. 필요에 따라 이 발췌를 가져온 전체 TKG 클러스터 예시(v1alpha3 예시: 기본 스토리지 및 노드 볼륨이 있는 TKC)를 참조하십시오. v1beta1 API 클러스터 예는 v1beta1 예시: 기본 ClusterClass를 기반으로 하는 사용자 지정 클러스터 항목을 참조하십시오.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
...
spec:
   topology:
     controlPlane:
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: guaranteed-medium
       tkr:
         reference:
           name: v1.24.9---vmware.1-tkg.4
     nodePools:
     - name: worker-nodepool-a1
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: guaranteed-large
       tkr:
         reference:
           name: v1.24.9---vmware.1-tkg.4
       volumes:
       - name: containerd
         mountPath: /var/lib/containerd
         capacity:
           storage: 50Gi
       - name: kubelet
         mountPath: /var/lib/kubelet
         capacity:
```

```
            storage: 50Gi
      - name: worker-nodepool-a2
        ...
    settings:
      ...
```

vSphere Client를 사용하여 TKG 클러스터 상태 모니터링

vSphere Client를 사용하여 TKG 클러스터의 상태를 모니터링할 수 있습니다.

절차

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **메뉴**에서 **호스트 및 클러스터** 보기를 선택합니다.

3 감독자가 생성된 **데이터 센터 > 클러스터** 개체를 확장합니다.

4 **네임스페이스** 리소스 풀을 확장합니다.

5 Tanzu Kubernetes 클러스터를 배포한 vSphere 네임스페이스를 선택합니다.

각 TKG 클러스터는 해당 네임스페이스 리소스 풀 내에 폴더로 나열됩니다. 각 TKG는 이름 옆 3개의 육각형 아이콘을 통해 그래픽으로 표시됩니다.

6 **메뉴 > VM 및 템플릿** 보기로 전환합니다.

클러스터 폴더 내에 클러스터 노드를 구성하는 VM이 표시됩니다.

7 vSphere 네임스페이스를 선택한 다음, **계산** 탭을 선택합니다.

8 **VMware 리소스**에서 **Tanzu Kubernetes**를 선택합니다.

이 vSphere 네임스페이스에 배포된 각 TKG 클러스터가 나열됩니다.

kubectl을 사용하여 TKG 클러스터 상태 모니터링

kubectl을 사용하여 프로비저닝된 TKG 클러스터의 상태를 모니터링할 수 있습니다.

절차

1 감독자로 인증합니다.

2 클러스터를 실행 중인 vSphere 네임스페이스로 전환합니다.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 네임스페이스에서 실행 중인 Tanzu Kubernetes 클러스터 목록을 봅니다.

```
kubectl get tanzukubernetesclusters
```

이 명령은 클러스터의 상태를 반환합니다.

4 클러스터의 세부 정보를 봅니다.

```
kubectl describe tanzukubernetescluster <cluster-name>
```

이 명령은 클러스터의 세부 정보를 반환합니다. 명령 출력의 [상태] 섹션에 클러스터에 대한 세부 정보가 표시됩니다.

```
...
Status:
  Addons:
    Cni:
      Name:    calico
      Status:  applied
    Csi:
      Name:    pvcsi
      Status:  applied
    Psp:
      Name:    defaultpsp
      Status:  applied
  Cloudprovider:
    Name:  vmware-guest-cluster
  Cluster API Status:
    API Endpoints:
      Host:  10.161.90.22
      Port:  6443
    Phase:   provisioned
  Node Status:
    test-tanzu-cluster-control-plane-0:                ready
    test-tanzu-cluster-workers-0-749458f97c-97ljv:     ready
  Phase:                                               running
  Vm Status:
    test-tanzu-cluster-control-plane-0:                ready
    test-tanzu-cluster-workers-0-749458f97c-97ljv:     ready
Events:                                                <none>
```

5 추가 `kubectl` 명령을 실행하여 클러스터에 대한 자세한 정보를 봅니다. Kubectl을 사용하여 TKG 클러스터 작업 확인의 내용을 참조하십시오.

Kubectl을 사용하여 TKG 클러스터 준비 상태 확인

TKG 컨트롤러가 TKG 클러스터를 프로비저닝할 때 몇 가지 상태 조건이 보고되며, 이것을 사용하여 시스템 상태의 주요 측면에 대한 직접적인 인사이트를 얻을 수 있습니다.

TKG 클러스터 준비 상태 확인

TKG 클러스터 준비 상태 조건을 사용하면 어떤(있는 경우) 단계 또는 구성 요소가 준비되지 않았는지 판단할 수 있습니다.

클러스터 준비 상태 조건을 확인한 후 자세한 진단을 위해 vSphereCluster 및 시스템 조건을 사용하여 장애를 더 자세히 살펴볼 수 있습니다.

TKG 클러스터의 준비 상태를 확인하려면 다음을 수행합니다.

1 감독자에 로그인합니다.

2 대상 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다. 예:

```
kubectl config use-context tkg-cluster-ns
```

3 TKG 클러스터 유형에 따라 `kubectl get tkc -o yaml` 또는 `kubectl get cluster -o yaml` 명령을 실행합니다.

명령은 클러스터 구성 요소의 준비 상태를 반환합니다. 다양한 준비 상태에 대한 설명은 다음 섹션을 참조하십시오.

ControlPlaneReady 조건 및 이유

다음 표에는 `ControlPlaneReady` 조건과 그에 대한 설명이 나와 있습니다.

표 8-2. ControlPlaneReady 조건

조건 유형	설명
`ControlPlaneReady`	제어부 노드가 준비되어 있고 클러스터에 대해 작동하는지 보고합니다.

다음 표에는 `ControlPlaneReady` 조건이 false일 수 있는 이유와 그에 대한 설명이 나와 있습니다.

표 8-3. ControlPlaneReady False 이유

이유	설명
`WaitingForClusterInfrastructure`	클러스터가 시스템 실행에 필요한 사전 요구 사항(예: 로드 밸런서)을 기다리고 있음을 나타냅니다. 이 이유는 InfrastructureCluster가 자체 준비 조건을 보고하지 않는 경우에만 사용됩니다.
`WaitingForControlPlaneInitialized`	첫 번째 제어부 노드가 초기화 중임을 나타냅니다.
`WaitingForControlPlane`	KubeadmControlPlane의 조건을 반영합니다. 이 이유는 KubeadmControlPlane이 자체 준비 조건을 보고하지 않는 경우 사용됩니다.
클러스터 인프라가 준비되기를 기다리는 중	클러스터가 시스템 실행에 필요한 사전 요구 사항(예: 네트워킹 및 로드 밸런서)을 기다리고 있음을 나타내는 메시지입니다.

NodesHealthy 조건 및 이유

다음 표에는 `NodesHealthy` 조건과 그에 대한 설명이 나와 있습니다.

표 8-4. NodesHealthy 조건

조건 유형	설명
`NodesHealthy`	TanzuKubernetesCluster 노드의 상태를 보고합니다.

다음 표에는 `NodesHealthy` 조건이 true가 아닌 이유와 그에 대한 설명이 나와 있습니다.

표 8-5. NodesHealthy False 이유

이유	설명
`WaitingForNodesHealthy`	일부 노드는 정상이 아니라는 설명입니다.

추가 기능 조건 및 이유

다음 표에는 클러스터 추가 기능 구성 요소와 관련된 조건 및 그에 대한 설명이 나와 있습니다.

표 8-6. 추가 기능 조건

조건 유형	설명
`AddonsReady`	TanzuKubernetesCluster 추가 기능(CoreDNS, KubeProxy, CSP, CPI, CNI, AuthSvc)에 대한 조건 요약.
`CNIProvisioned`	TanzuKubernetesCluster CNI(Container Network Interface) 추가 기능의 상태를 설명합니다.
`CSIProvisioned`	TanzuKubernetesCluster CSI(Container Storage Interface) 추가 기능의 상태를 설명합니다.
`CPIProvisioned`	TanzuKubernetesCluster CPI(Cloud Provider Interface) 추가 기능의 상태를 설명합니다.
`KubeProxyProvisioned`	TanzuKubernetesCluster KubeProxy 추가 기능의 상태를 설명합니다.
`CoreDNSProvisioned`	TanzuKubernetesCluster CoreDNS 추가 기능의 상태를 설명합니다.
`AuthServiceProvisioned`	TanzuKubernetesCluster AuthService 추가 기능의 상태를 설명합니다.
`PSPProvisioned`	PodSecurityPolicy의 상태를 설명합니다.

다음 표에는 추가 기능 조건이 true가 아닌 이유와 그에 대한 설명이 나와 있습니다. 주의를 유발하는 증상의 문제를 해결하려면 장 21 TKG 서비스 클러스터 문제 해결 항목을 참조하십시오.

표 8-7. 추가 기능 False 이유

이유	심각도	설명
`AddonsReconciliationFailed`	해당 없음	모든 추가 기능 조정 실패에 대해 요약된 이유입니다.
`CNIProvisioningFailed`	주의	CNI 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.
`CSIProvisioningFailed`	주의	CSI 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.
`CPIProvisioningFailed`	주의	CPI 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.

표 8-7. 추가 기능 False 이유 (계속)

이유	심각도	설명
KubeProxyProvisioningFailed	주의	KubeProxy 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.
CoreDNSProvisioningFailed	주의	CoreDNS 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.
AuthServiceProvisioningFailed	주의	AuthService 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.
AuthServiceUnManaged		AuthService가 컨트롤러에서 관리되지 않음을 설명합니다.
PSPProvisioningFailed	주의	PodSecurityPolicy 추가 기능이 생성 또는 업데이트에 실패했음을 설명합니다.

기타 조건 및 이유

다음 표에는 StorageClass 및 RoleBinding 동기화, ProviderServiceAccount 리소스 조정, ServiceDiscovery, TKG 2.0 클러스터 호환성에 대한 조건과 그에 대한 설명이 나와 있습니다.

표 8-8. 기타 조건

조건	설명
StorageClassSynced	감독자 클러스터에서 워크로드 클러스터로의 StorageClass 동기화 상태를 설명합니다.
RoleBindingSynced	감독자 클러스터에서 워크로드 클러스터로의 RoleBinding 동기화 상태를 설명합니다.
ProviderServiceAccountsReady	제공자 서비스 계정의 상태를 설명하고 관련 역할, RoleBinding 및 암호가 생성됩니다.
ServiceDiscoveryReady	서비스 검색 상태를 설명합니다.
TanzuKubernetesReleaseCompatible	TanzuKubernetesCluster가 TanzuKubernetesRelease와 호환되는지 여부를 나타냅니다.

다음 표에는 다른 조건이 true가 아닌 이유와 그에 대한 설명이 나와 있습니다.

표 8-9. 기타 이유

이유	설명
StorageClassSyncFailed	StorageClass 동기화 실패를 보고합니다.
RoleBindingSyncFailed	RoleBinding 동기화 실패를 보고합니다.
ProviderServiceAccountsReconciliationFailed	제공자 서비스 계정 관련 리소스 조정에 실패했음을 보고합니다.
SupervisorHeadlessServiceSetupFailed	감독자 클러스터 API 서버에 대한 헤드리스 서비스 설정에 실패했음을 설명합니다.

Kubectl을 사용하여 TKG 클러스터 시스템 상태 확인

TKG 컨트롤러가 감독자에서 워크로드 클러스터를 프로비저닝할 때 몇 가지 상태 조건이 보고되며, 이것을 사용하여 시스템 상태의 주요 측면에 대한 직접적인 인사이트를 얻을 수 있습니다.

시스템 상태 조건 정보

TKG 클러스터는 여러 움직이는 부분으로 구성되며, 모두 독립적이지만 관련된 컨트롤러에 의해 운영되고 함께 작동하여 Kubernetes 노드 집합을 구축하고 유지 보수합니다. `TanzuKubernetesCluster` 및 `Cluster` 개체는 시스템 상태에 대해 세분화된 정보를 제공하는 상태 조건을 제공합니다.

시스템 상태 확인

TKG 클러스터 시스템의 상태를 확인하려면 다음을 수행합니다.

1 감독자에 연결하고 로그인합니다.

2 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context CLUSTER-NAME
```

3 `kubectl describe machine` 명령을 실행합니다.

이 명령은 클러스터를 구성하는 가상 시스템 노드의 상태를 반환합니다. `InfrastructureReady`와 같은 시스템 조건이 `True` 및 `Ready`(이)면 시스템의 해당 측면은 정상입니다. 하지만 시스템 조건이 `False`이면 시스템이 준비되지 않은 것입니다. 각 시스템 조건 유형에 대한 설명은 시스템 상태 조건 목록을 참조하십시오.

4 시스템이 준비되지 않은 경우 다음 명령을 실행하고 인프라에 어떤 문제가 있는지 확인합니다.

```
kubectl describe vspheremachine
```

시스템 상태 조건 목록

이 표에는 TKG 클러스터에 사용할 수 있는 시스템 상태 조건이 나열 및 정의되어 있습니다.

조건	설명
`ResourcePolicyReady`	리소스 정책이 성공적으로 생성되었다고 보고합니다.
`ResourcePolicyCreationFailed`	ResourcePolicy를 생성하는 동안 오류가 발생하면 보고됩니다.
`ClusterNetworkReady`	클러스터 네트워크가 성공적으로 프로비저닝되었다고 보고합니다.
`ClusterNetworkProvisionStarted`	클러스터 네트워크가 준비되기를 기다릴 때 보고됩니다.
`ClusterNetworkProvisionFailed`	네트워크 프로비저닝 중에 오류가 발생하면 보고됩니다.
`LoadBalancerReady`	정적 제어부 끝점이 성공적으로 조정되었다고 보고합니다.
`LoadBalancerCreationFailed`	로드 밸런서 관련 리소스 생성이 실패한 경우 보고됩니다.
`WaitingForLoadBalancerIP`	로드 밸런서 IP가 존재하기를 기다릴 때 보고됩니다.
`VMProvisioned`	가상 시스템이 생성되고 전원이 켜지고 IP가 할당된 것을 보고합니다.

조건	설명
WaitingForBootstrapData	프로비저닝 프로세스를 시작하기 전에 vSphereMachine이 부트스트랩 스크립트가 준비될 때까지 기다릴 때 보고됩니다.
VMCreationFailed	VM CRD 또는 해당 부트스트랩 ConfigMap 생성에 실패했다고 보고합니다.
VMProvisionStarted	가상 시스템이 현재 생성 프로세스에 있을 때 보고됩니다.
PoweringOn	가상 시스템에서 현재 전원 켜기 순서를 실행하고 있을 때 보고됩니다.
WaitingForNetworkAddress	시스템 네트워크 설정이 활성 상태가 되기를 기다릴 때 보고됩니다.
WaitingForBIOSUUID	시스템에 BIOS UUID가 지정되기를 기다릴 때 보고됩니다.

조건 필드

각 조건에는 여러 필드가 포함될 수 있습니다.

Type	조건 유형을 설명합니다. 예를 들면 ResourcePolicyReady입니다. Ready 조건의 경우 다른 모든 조건에 대한 요약입니다.
Status	유형의 상태를 설명합니다. 상태는 True, False 또는 Unknown일 수 있습니다.
Severity	Reason의 분류입니다. Info는 조정이 진행 중임을 의미합니다. Warning은 오류가 발생했을 수 있으며 재시도할 수 있음을 의미합니다. Error는 오류가 발생했고 해결하려면 수동 작업이 필요함을 의미합니다.
Reason	상태가 False인 이유를 제공합니다. 준비 대기 또는 실패 이유일 수 있습니다. 일반적으로 상태가 False인 경우에 throw됩니다.
Message	사람이 읽을 수 있는 Reason을 설명하는 정보입니다.

Kubectl을 사용하여 TKG 클러스터 상태 확인

TKG 컨트롤러가 워크로드 클러스터를 프로비저닝할 때 몇 가지 상태 조건이 보고되며, 이것을 사용하여 클러스터 상태의 주요 측면에 대한 직접적인 인사이트를 얻을 수 있습니다.

클러스터 상태 조건 정보

프로비저닝된 TKG 클러스터는 여러 움직이는 부분으로 구성되며, 모두 독립적이지만 관련된 컨트롤러에 의해 운영되고 함께 작동하여 Kubernetes 노드 집합을 구축하고 유지 보수합니다. TanzuKubernetesCluster 및 Cluster 개체는 클러스터 및 시스템 상태에 대해 세분화된 정보를 제공하는 상태 조건을 제공합니다.

클러스터 상태 확인

TKG 클러스터의 상태를 확인하려면 다음을 수행합니다.

1 kubectl describe cluster 명령을 실행합니다.

상태가 Ready이면 클러스터 인프라와 클러스터 제어부 모두가 준비되었음을 의미합니다. 예:

```
Status:
  Conditions:
    Last Transition Time:     2020-11-24T21:37:32Z
    Status:                   True
    Type:                     Ready
    Last Transition Time:     2020-11-24T21:37:32Z
    Status:                   True
    Type:                     ControlPlaneReady
    Last Transition Time:     2020-11-24T21:31:34Z
    Status:                   True
    Type:                     InfrastructureReady
```

하지만 클러스터 조건이 false이면 클러스터가 준비되지 않은 것이며 메시지 필드에 무엇이 잘못되었는지 설명됩니다. 예를 들어 다음은 상태가 False이고 인프라가 준비되지 않았기 때문입니다.

```
Status:
  Conditions:
    Last Transition Time:     2020-11-24T21:37:32Z
    Status:                   False
    Type:                     Ready
    Last Transition Time:     2020-11-24T21:37:32Z
    Status:                   True
    Type:                     ControlPlaneReady
    Last Transition Time:     2020-11-24T21:31:34Z
    Status:                   False
    Type:                     InfrastructureReady
```

2 클러스터가 준비되지 않은 경우 다음 명령을 실행하여 클러스터 인프라에 어떤 문제가 있는지 확인합니다.

```
kubectl describe vspherecluster
```

클러스터 상태 조건 목록

이 표에는 TKG 클러스터에 사용할 수 있는 상태 조건이 나열 및 정의되어 있습니다.

조건	설명
`Ready`	클러스터 API 개체의 작동 상태를 요약합니다.
`Deleting`	상태가 True가 아니며 기본 개체가 현재 삭제되고 있기 때문입니다.
`DeletionFailed`	상태가 True가 아니며 삭제하는 동안 기본 개체에 문제가 발생했기 때문입니다. 조정자가 삭제를 재시도하기 때문에 주의입니다.
`Deleted`	상태가 True가 아니며 기본 개체가 삭제되었기 때문입니다.
`InfrastructureReady`	이 클러스터에 대해 정의된 인프라 개체의 현재 상태에 대한 요약을 보고합니다.

조건	설명
`WaitingForInfrastructure`	클러스터가 기본 인프라를 사용할 수 있기를 기다릴 때 보고됩니다. 참고: 이 조건은 인프라가 준비 상태를 보고하지 않을 때 폴백으로 사용됩니다.
`ControlPlaneReady`	클러스터 제어부가 준비되었을 때 보고됩니다.
`WaitingForControlPlane`	클러스터가 제어부를 사용할 수 있기를 기다릴 때 보고됩니다. 참고: 이 조건은 제어부가 준비 상태를 보고하지 않을 때 폴백으로 사용됩니다.

조건 필드

각 조건에는 여러 필드가 포함될 수 있습니다.

`Type`	조건 유형을 설명합니다. 예를 들면 `ControlPlaneReady`입니다. `Ready` 조건의 경우 다른 모든 조건에 대한 요약입니다.
`Status`	유형의 상태를 설명합니다. 상태는 `True`, `False` 또는 `Unknown`일 수 있습니다.
`Severity`	`Reason`의 분류입니다. `Info`는 조정이 진행 중임을 의미합니다. `Warning`은 오류가 발생했을 수 있으며 재시도할 수 있음을 의미합니다. `Error`는 오류가 발생했고 해결하려면 수동 작업이 필요함을 의미합니다.
`Reason`	상태가 `False`인 이유를 제공합니다. 준비 대기 또는 실패 이유일 수 있습니다. 일반적으로 상태가 `False`인 경우에 throw됩니다.
`Message`	사람이 읽을 수 있는 `Reason`을 설명하는 정보입니다.

Kubectl을 사용하여 TKG 클러스터 볼륨 상태 확인

TKG 클러스터에서 바인딩된 상태의 영구 볼륨 상태를 확인할 수 있습니다.

바인딩된 상태의 각 영구 볼륨에 대한 상태는 영구 볼륨에 바인딩된 영구 볼륨 클레임의 `Annotations: volumehealth.storage.kubernetes.io/messages:` 필드에 표시됩니다. 가능한 상태 값에는 두 가지가 있습니다.

상태	설명
액세스 가능	영구 볼륨에 액세스할 수 있고 사용할 수 있습니다.
액세스할 수 없음	영구 볼륨에 액세스할 수 없고 사용할 수 없습니다. 데이터스토어에 연결하는 호스트가 볼륨을 저장하는 데이터스토어에 연결할 수 없으면 영구 볼륨에 액세스할 수 없게 됩니다.

절차

1 Kubectl을 사용하여 TKG 클러스터에 로그인합니다.

2 영구 볼륨 할당을 생성합니다.

a 영구 볼륨 할당 구성을 포함하는 YAML 파일을 생성합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 2Gi
```

b Kubernetes 클러스터에 영구 볼륨 할당을 적용합니다.

```
kubectl apply -f pvc_name.yaml
```

이 명령은 할당의 스토리지 요구 사항을 충족하는 백업 가상 디스크가 있는 Kubernetes 영구 볼륨과 vSphere 볼륨을 생성합니다.

c 영구 볼륨 할당이 볼륨에 바인딩되어 있는지 확인합니다.

```
kubectl get pvc my-pvc
```

출력은 영구 볼륨 할당 및 볼륨이 바인딩된 상태임을 표시합니다.

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

3 볼륨의 상태를 확인합니다.

다음 명령을 실행하여 영구 볼륨에 바인딩된 영구 볼륨 클레임의 볼륨 상태 주석을 확인합니다.

```
kubectl describe pvc my-pvc
```

다음 샘플 출력에서 `volumehealth.storage.kubernetes.io/messages` 필드에 상태가 액세스 가능한 것으로 표시됩니다.

```
Name:          my-pvc
Namespace:     test-ns
StorageClass:  gold
Status:        Bound
Volume:        my-pvc
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: csi.vsphere.vmware.com
               volumehealth.storage.kubernetes.io/messages: accessible
```

```
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      2Gi
Access Modes:  RWO
VolumeMode:    Filesystem
```

Tanzu Kubernetes Grid 클러스터의 볼륨 상태 모니터링

바인딩된 상태에서 영구 볼륨의 상태를 확인할 수 있습니다.

바인딩된 상태의 각 영구 볼륨에 대한 상태는 영구 볼륨에 바인딩된 영구 볼륨 클레임의 `Annotations: volumehealth.storage.kubernetes.io/messages:` 필드에 표시됩니다. 가능한 상태 값에는 두 가지가 있습니다.

상태	설명
액세스 가능	영구 볼륨에 액세스할 수 있고 사용할 수 있습니다.
액세스할 수 없음	영구 볼륨에 액세스할 수 없고 사용할 수 없습니다. 데이터스토어에 연결하는 호스트가 볼륨을 저장하는 데이터스토어에 연결할 수 없으면 영구 볼륨에 액세스할 수 없게 됩니다.

절차

1 vSphere Kubernetes 환경의 네임스페이스에 액세스합니다.

2 영구 볼륨 할당을 생성합니다.

a 영구 볼륨 할당 구성을 포함하는 YAML 파일을 생성합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 2Gi
```

b Kubernetes 클러스터에 영구 볼륨 할당을 적용합니다.

```
kubectl apply -f pvc_name.yaml
```

이 명령은 할당의 스토리지 요구 사항을 충족하는 백업 가상 디스크가 있는 Kubernetes 영구 볼륨과 vSphere 볼륨을 생성합니다.

c 영구 볼륨 할당이 볼륨에 바인딩되어 있는지 확인합니다.

```
kubectl get pvc my-pvc
```

출력은 영구 볼륨 할당 및 볼륨이 바인딩된 상태임을 표시합니다.

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

3 볼륨의 상태를 확인합니다.

다음 명령을 실행하여 영구 볼륨에 바인딩된 영구 볼륨 클레임의 볼륨 상태 주석을 확인합니다.

```
kubectl describe pvc my-pvc
```

다음 샘플 출력에서 `volumehealth.storage.kubernetes.io/messages` 필드에 상태가 액세스 가능한 것으로 표시됩니다.

```
Name:          my-pvc
Namespace:     test-ns
StorageClass:  gold
Status:        Bound
Volume:        my-pvc
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: csi.vsphere.vmware.com
               volumehealth.storage.kubernetes.io/messages: accessible
```

```
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      2Gi
Access Modes:  RWO
VolumeMode:    Filesystem
```

vSphere Client를 사용하여 영구 볼륨 모니터링

DevOps 엔지니어가 영구 볼륨 할당과 함께 상태 저장 애플리케이션을 배포할 때 vSphere IaaS control plane은 영구 볼륨 개체 및 일치하는 영구 가상 디스크를 생성합니다. vSphere 관리자는 vSphere Client에서 영구 볼륨의 세부 정보를 검토할 수 있습니다. 또한 해당 스토리지 규정 준수 상태와 영구 볼륨 상태를 모니터링할 수 있습니다.

절차

1 vSphere Client에서 영구 볼륨이 있는 네임스페이스로 이동합니다.

   a vSphere Client 홈 메뉴에서 **워크로드 관리**를 선택합니다.

   b **네임스페이스** 탭을 클릭하고 목록에서 네임스페이스를 선택합니다.

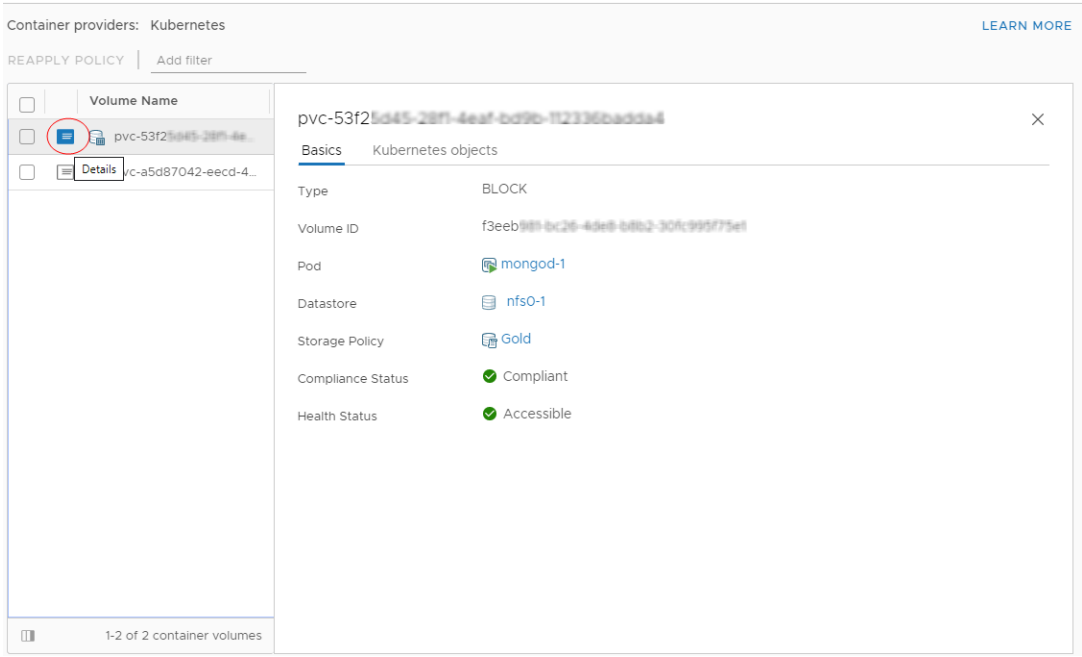
2 **스토리지** 탭을 클릭하고 **영구 볼륨 할당**을 클릭합니다.

   vSphere Client가 모든 영구 볼륨 할당 개체와 네임스페이스에서 사용할 수 있는 해당 볼륨을 나열합니다.

3 선택된 영구 볼륨 할당의 세부 정보를 보려면 **영구 볼륨 이름** 열에서 볼륨의 이름을 클릭합니다.

4 **컨테이너 볼륨** 페이지에서 볼륨의 상태와 스토리지 정책 규정 준수를 확인합니다.

a **세부 정보** 아이콘을 클릭하고 **기본** 및 **Kubernetes 개체** 탭 사이를 전환하여 Kubernetes 영구 볼륨에 대한 추가 정보를 봅니다.



b 볼륨의 상태를 확인합니다.

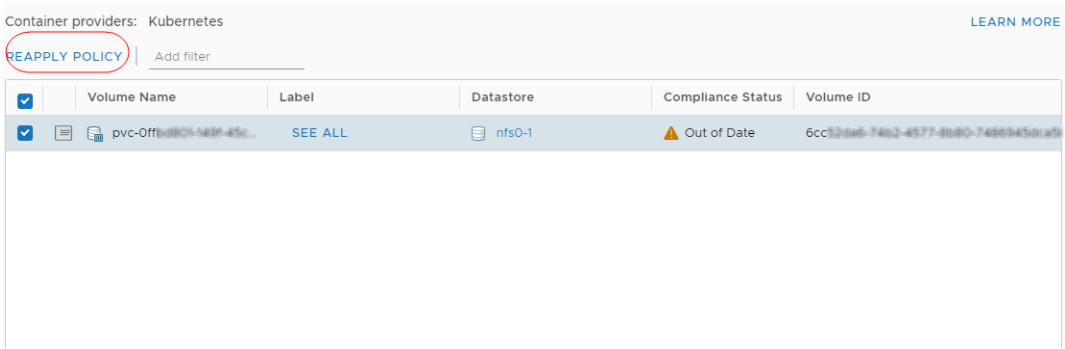
상태	설명
액세스 가능	영구 볼륨에 액세스할 수 있고 사용할 수 있습니다.
액세스할 수 없음	영구 볼륨에 액세스할 수 없고 사용할 수 없습니다. 데이터스토어에 연결하는 호스트가 볼륨을 저장하는 데이터스토어에 연결할 수 없으면 영구 볼륨에 액세스할 수 없게 됩니다.

c 스토리지 규정 준수 상태를 확인합니다.

규정 준수 상태 열에서 다음 중 하나를 볼 수 있습니다.

규정 준수 상태	설명
준수	볼륨 백업 가상 디스크가 상주하는 데이터스토어가 정책에 필요한 스토리지 기능을 갖추고 있습니다.
최신 버전이 아님	이 상태는 정책이 편집되었지만 데이터스토어에 새 스토리지 요구 사항이 전달되지 않았음을 나타냅니다. 변경 사항을 전달하려면 최신 상태가 아닌 볼륨에 정책을 다시 적용합니다.
비준수	데이터스토어가 지정된 스토리지 요구 사항을 지원하지만 현재는 스토리지 정책을 충족할 수 없습니다. 예를 들어 데이터스토어의 물리적 리소스를 사용할 수 없는 경우 상태가 비준수가 될 수도 있습니다. 호스트나 디스크를 클러스터에 추가하는 방법 등으로 호스트 클러스터의 물리적 구성을 변경하면 데이터스토어를 준수 상태로 만들 수 있습니다. 추가적인 리소스가 스토리지 정책을 충족하면 상태가 [준수]로 변경됩니다.
해당 없음	스토리지 정책이 데이터스토어에서 지원되지 않는 데이터스토어 기능을 참조합니다.

d 규정 준수 상태가 [최신 버전이 아님]인 경우 볼륨을 선택하고 **정책 다시 적용**을 클릭합니다.



상태가 [준수]로 변경됩니다.

KubectI을 사용하여 TKG 클러스터 암호 얻기

TKG 클러스터는 암호를 사용하여 운영을 위한 토큰, 키 및 암호를 저장합니다.

TKG 클러스터 암호 목록

Kubernetes 암호는 암호, 토큰 또는 SSH 키와 같은 소량의 중요한 데이터를 저장하는 개체입니다. TKG 클러스터 관리자는 클러스터를 운영하는 동안 몇 가지 암호를 사용할 수 있습니다. 아래 표에는 관리자가 사용할 수 있는 주요 암호 클러스터가 나열되고 설명되어 있습니다.

참고 이 목록은 포괄적이지 않습니다. 문제 해결을 위해 클러스터 노드에 액세스하는 데 사용하거나 수동으로 순환해야 할 수도 있는 암호만 포함됩니다.

암호	설명
TANZU-KUBERNETES-CLUSTER-NAME-ccm-token-RANDOM	반가상화 클라우드 제공자의 클라우드 컨트롤러 관리자가 vSphere 네임스페이스에 연결하는 데 사용하는 서비스 계정 토큰입니다. 이 자격 증명의 순환을 트리거하려면 암호를 삭제합니다.
TANZU-KUBERNETES-CLUSTER-NAME-pvcsi-token-RANDOM	반가상화 CSI 플러그인이 vSphere 네임스페이스에 연결하는 데 사용하는 서비스 계정 토큰입니다. 이 자격 증명의 순환을 트리거하려면 암호를 삭제합니다.
TANZU-KUBERNETES-CLUSTER-NAME-kubeconfig	`kubernetes-admin` 사용자로 클러스터의 제어부에 연결하는 데 사용할 수 있는 kubeconfig 파일입니다. 이 암호는 vCenter Single Sign-On 인증을 사용할 수 없는 경우 클러스터에 액세스하여 문제를 해결하는 데 사용할 수 있습니다.
TANZU-KUBERNETES-CLUSTER-NAME-ssh	`vmware-system-user`로 클러스터 노드에 연결하는 데 사용할 수 있는 SSH 개인 키입니다. 이 암호는 클러스터 노드에 SSH로 사용하여 문제를 해결하는 데 사용할 수 있습니다.
TANZU-KUBERNETES-CLUSTER-NAME-ssh-password	`vmware-system-user`로 클러스터 노드에 연결하는 데 사용할 수 있는 암호입니다.
TANZU-KUBERNETES-CLUSTER-NAME-ca	`kubectl`에서 Kubernetes API 서버에 안전하게 연결하는 데 사용되는 Tanzu Kubernetes 클러스터 제어부의 루트 CA 인증서입니다.

Kubectl을 사용하여 TKG 클러스터 네트워킹 확인

시스템은 노드, 포드 및 서비스에 대한 기본 네트워킹을 사용하여 TKG 클러스터를 프로비저닝합니다. 사용자 지정 kubectl 명령을 사용하여 클러스터 네트워킹을 확인할 수 있습니다.

TKG 클러스터 네트워킹을 확인하는 사용자 지정 명령

다음 명령을 참조하여 클러스터 네트워킹을 확인합니다.

이러한 명령은 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에서 실행해야 합니다. 예:

```
kubectl config use-context tkg2-cluster-ns
```

표 8-10. 클러스터 네트워킹을 확인하는 사용자 지정 kubectl 명령

명령	설명

명령
`kubectl get tkgserviceconfigurations`
샘플 결과
`NAME DEFAULT CNI`
`tkg-service-configuration antrea`
	기본 CNI를 반환하며, 이것은 변경되지 않는 한 `antrea`입니다.
	클러스터 YAML에 명시적으로 재정의된 경우를 제외하고 기본 CNI는 클러스터 생성에 사용됩니다.

명령
`kubectl get virtualnetwork`
샘플 결과
`NAME SNAT`
`READY AGE`
`tkgs-cluster-12-vnet 10.191.152.133`
`True 4h3m`
	클러스터 노드의 가상 네트워크를 반환합니다.
	SNAT(소스 네트워크 주소 변환) IP 주소가 할당되었는지 확인하는 데 사용합니다.

명령
`kubectl get virtualmachines -o wide`
샘플 결과
`NAME`
`POWERSTATE CLASS`
`IMAGE`
`PRIMARY-IP AGE`
`tkg2-cluster-12-control-plane-...`
`poweredOn guaranteed-medium ob-...-`
`v1.23.8---vmware.1-tkg.1.b3d708a`
`10.244.0.66 4h6m`
`tkg2-cluster-12-worker-...`
`poweredOn guaranteed-medium ob-...-`
`v1.22.9---vmware.1-tkg.1.b3d708a`
`10.244.0.68 4h3m`
`tkg2-cluster-12-worker-...`
`poweredOn guaranteed-medium ob-...-`
`v1.21.6---vmware.1-tkg.1.b3d708a`
`10.244.0.67 4h3m`
	클러스터 노드에 대한 가상 네트워크 인터페이스를 반환합니다.
	각 클러스터 노드에 대한 가상 시스템에 IP 주소가 할당되었는지 확인하는 데 사용합니다.

명령
`kubectl get virtualmachineservices`
샘플 결과
`NAME`
`TYPE AGE`
`tkg2-cluster-12-control-plane-service`
`LoadBalancer 3h53m`
	각 클러스터 노드에 대한 가상 시스템 서비스를 반환합니다.
	상태가 업데이트 되었는지 그리고 로드 밸런서 VIP(가상 IP) 주소를 포함하는지 확인하려는 경우 사용합니다.

표 8-10. 클러스터 네트워킹을 확인하는 사용자 지정 kubectl 명령 (계속)

명령	설명

명령
`kubectl get services -n NAMESPACE`
cURL을 사용하여 확인
`curl -k https://EXTERNAL-IP:PORT/healthz`
	클러스터 API 액세스를 위해 생성된 Kubernetes 서비스 로드 밸런서를 반환합니다. 외부 IP가 할당되었는지 확인하려는 경우 사용합니다.
	로드 밸런서 서비스의 외부 IP 주소 및 포트를 사용하여 API에 액세스할 수 있는지 확인하려면 `curl`을 사용합니다.

명령
`kubectl get endpoints`
샘플 결과
`NAME ENDPOINTS AGE tkg2-cluster-12-control-plane-service 10.244.0.66:6443 3h44m`
	클러스터에 대한 끝점(제어부 노드)을 반환합니다. 각 끝점이 생성되었고 끝점 풀에 포함되었는지 확인하는 데 사용합니다.

Kubectl을 사용하여 TKG 클러스터 작업 확인

사용자 지정 kubectl 명령을 사용하여 TKG 클러스터를 관리할 수 있습니다. 이러한 명령은 TKG 컨트롤러에서 관리되는 사용자 지정 리소스에서 사용할 수 있습니다.

TKG 클러스터를 관리하는 사용자 지정 명령

이 표에는 TKG 클러스터를 관리하기 위한 kubectl 명령이 나열 및 설명되어 있습니다.

TKG 클러스터가 프로비저닝된 vSphere 네임스페이스의 컨텍스트에서 각 명령을 실행합니다. 클러스터 컨텍스트에서 이러한 명령을 실행하면 정보가 반환되지 않습니다.

표 8-11. TKG 클러스터를 관리하는 사용자 지정 명령

명령	설명
`kubectl get tanzukubernetescluster`	현재 네임스페이스의 TKC를 나열합니다.
`kubectl get tkc`	앞선 명령의 짧은 버전입니다.
`kubectl get cluster`	네임스페이스의 클러스터를 반환합니다.
`kubectl describe tanzukubernetescluster CLUSTER-NAME`	표시된 상태, 상태 및 이벤트를 표시하는 방식으로 지정된 클러스터를 설명합니다. 프로비저닝이 완료되면 이 명령은 Kubernetes API 끝점을 향하고 있는 로드 밸런서에 대해 생성된 가상 IP를 표시합니다.

표 8-11. TKG 클러스터를 관리하는 사용자 지정 명령 (계속)

명령	설명
`kubectl get cluster-api`	클러스터 API 프로젝트의 리소스와 Tanzu Kubernetes Grid Service에 사용되는 클러스터 API 제공자의 리소스를 포함하여 현재 네임스페이스에서 클러스터를 지원하는 클러스터 API 리소스를 나열합니다.
`kubectl get tanzukubernetesreleases`	사용 가능한 Tanzu Kubernetes 릴리스를 나열합니다.
`kubectl get tkr`	앞선 명령의 짧은 버전입니다.
`kubectl get tkr v1.23.8---vmware.1-tkg.1.5417466 -o yaml`	명명된 Tanzu Kubernetes 릴리스에 대한 세부 정보를 제공합니다.
`kubectl get virtualmachine`	현재 네임스페이스에서 클러스터 노드를 지원하는 가상 시스템 리소스를 나열합니다.
`kubectl get vm`	앞선 명령의 짧은 버전입니다.
`kubectl describe virtualmachine VIRTUAL-MACHINE-NAME`	상태, 현재 상태 및 이벤트를 표시하는 방식으로 지정된 가상 시스템을 설명합니다.
`kubectl describe virtualmachinesetresourcepolicy`	현재 네임스페이스에서 클러스터를 지원하는 가상 시스템 설정 리소스 정책 리소스를 나열합니다. 이 리소스는 클러스터에 사용되는 vSphere 개체 리소스 풀 및 폴더를 나타냅니다.
`kubectl get virtualmachineservice`	현재 네임스페이스에서 클러스터 노드를 지원하는 가상 시스템 서비스 리소스를 나열합니다. 이러한 리소스는 서비스와 유사하지만 포드가 아닌 가상 시스템에 사용됩니다. 가상 시스템 서비스는 클러스터의 제어부 노드에 로드 밸런서를 제공하는 데 사용되며 반가상화 클라우드 제공자가 클러스터 내에서 LoadBalancer 유형의 Kubernetes 서비스를 지원하는 데 사용됩니다.
`kubectl get vmservice`	앞선 명령의 짧은 버전입니다.
`kubectl describe virtualmachineservice VIRTUAL-MACHINE-SERVICE-NAME`	표시된 클러스터 상태, 현재 상태 및 이벤트를 표시하는 방식으로 지정된 가상 시스템 서비스를 설명합니다.
`kubectl get virtualmachineimage`	사용 가능한 가상 시스템 이미지를 나열합니다.
`kubectl get vmimage`	앞선 명령의 바로 가기 버전입니다.
`kubectl describe vmimage VM_IMAGE_NAME`	명명된 VM 이미지에 대한 세부 정보를 봅니다.
`kubectl get virtualnetwork`	클러스터에 사용되는 리소스를 포함하여 현재 네임스페이스의 가상 네트워크 리소스를 나열합니다. 가상 네트워크는 클러스터가 프로비저닝된 각 네임스페이스와 각 클러스터 자체에 대해 생성됩니다.

표 8-11. TKG 클러스터를 관리하는 사용자 지정 명령 (계속)

명령	설명
`kubectl get persistentvolumeclaim`	클러스터에 사용되는 리소스를 포함하여 현재 네임스페이스의 영구 볼륨 할당 리소스를 나열합니다.
`kubectl get cnsnodevmattachment`	현재 네임스페이스의 CNS 노드 가상 시스템 첨부 리소스를 나열합니다. 이러한 리소스는 CNS에서 관리되는 영구 볼륨이 클러스터의 노드로 작동하는 가상 시스템에 연결되었음을 나타냅니다.
`kubectl get configmap`	클러스터 노드 생성에 사용되는 구성 맵을 포함하여 현재 네임스페이스의 구성 맵을 나열합니다. 구성 맵은 사용자가 수정할 수 없으며 모든 변경 내용을 덮어씁니다.
`kubectl get secret`	클러스터 노드의 생성 및 관리에 사용되는 암호를 포함하여 현재 네임스페이스의 암호를 나열합니다.

TKG 클러스터 수명 주기 상태 보기

TKG 클러스터의 수명 주기 상태는 vSphere 인벤토리에서 kubectl을 사용하여 볼 수 있습니다.

vSphere의 TKG 클러스터 수명 주기 상태

아래 표에는 vSphere 인벤토리에 표시되는 TKG 클러스터 상태 정보가 나열되고 설명되어 있습니다.

표 8-12. vSphere 인벤토리의 TKG 클러스터 상태

필드	설명	예
이름	클러스터의 사용자 정의 이름입니다.	`tkg2-cluster-01`
생성 시간	클러스터 생성 날짜 및 시간입니다.	`Mar 17, 2022, 11:42:46 PM`
단계	클러스터의 수명 주기 상태입니다.	`creating`
작업자 수	클러스터의 작업자 노드 수입니다.	`1` 또는 `2` 또는 `5`
배포 버전	클러스터에서 실행 중인 Kubernetes 소프트웨어의 버전입니다.	`v1.22.6+vmware.1-tkg.1.7144628`
제어부 주소	클러스터 제어부 로드 밸런서의 IP 주소입니다.	`192.168.123.2`

kubectl의 TKG 클러스터 수명 주기 상태

아래 표에는 kubectl에 나타나는 TKG 클러스터 상태 정보가 나열되고 설명되어 있습니다.

표 8-13. kubectl의 TKG 클러스터 상태

필드	설명	예
NAME	클러스터의 이름입니다.	`tkg2-cluster-01`
CONTROL PLANE	클러스터의 제어부 노드 수입니다.	`3`

표 8-13. kubectl의 TKG 클러스터 상태 (계속)

필드	설명	예
WORKER	클러스터의 작업자 노드 수입니다.	`5`
DISTRIBUTION	클러스터가 실행되고 있는 Kubernetes 버전입니다.	`v1.22.6+vmware.1-tkg.1.5b5608b`
AGE	클러스터가 실행된 기간(일)입니다.	`13d`
PHASE	클러스터의 수명 주기 상태입니다.	`running`

클러스터 수명 주기 단계 상태

아래 표에는 클러스터 수명 주기의 각 단계에 대한 상태가 나열되고 설명되어 있습니다.

표 8-14. 클러스터 수명 주기 단계 상태

단계	설명
`creating`	클러스터 프로비저닝을 시작할 수 있거나, 제어부가 생성되고 있거나, 제어부가 생성되었지만 초기화되지 않았습니다.
`deleting`	클러스터가 삭제되고 있습니다.
`failed`	클러스터 제어부 생성에 실패했으며 사용자 개입이 필요할 수 있습니다.
`running`	인프라가 생성 및 구성되고 제어부가 완전히 초기화되었습니다.
`updating`	클러스터가 업데이트되고 있습니다.

Kubectl을 사용하여 TKG 클러스터의 리소스 계층 구조 보기

kubectl을 사용하여 TKG 클러스터의 리소스 계층 구조를 볼 수 있습니다. 클러스터 리소스의 전체 목록을 보면 문제를 유발할 수 있는 리소스를 정확하게 찾아낼 수 있습니다.

절차

1 감독자에 연결하고 로그인합니다.

2 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkg2-cluster-ns
```

3 다음 명령을 실행하여 클러스터 API 클러스터 리소스를 확인합니다.

```
kubectl describe clusters.cluster.x-k8s.io CLUSTER-NAME
```

이 명령은 Kubernetes 네임스페이스, API 버전, 리소스 버전을 포함하여 명명된 클러스터의 리소스 계층 구조를 반환합니다.

v1beta1 클러스터에 대한 MachineHealthCheck 구성

이 항목에서는 v1beta1 API를 사용하여 프로비저닝된 TKG 서비스 클러스터에 대해 MachineHealthCheck를 구성하는 방법을 설명합니다.

v1beta1 클러스터에 대한 MachineHealthCheck

MachineHealthCheck는 비정상 시스템에 업데이트를 적용하기 위한 조건을 정의하는 Kubernetes Cluster API 리소스입니다. Kubernetes에서 시스템은 kubelet을 실행할 수 있는 사용자 지정 리소스입니다. vSphere IaaS control plane에서 Kubernetes 시스템 리소스는 vSphere 가상 시스템의 지원을 받습니다. 자세한 내용은 업스트림 설명서를 참조하십시오.

TKG 서비스를 사용하여 클러스터를 프로비저닝하면 시스템은 모든 제어부에 대해 하나씩, 시스템 배포마다 하나씩 기본 MachineHealthCheck 개체를 생성합니다. vSphere 8 업데이트 3부터는 v1beta1 클러스터에 대해 시스템 상태 점검을 구성할 수 있습니다. 지원되는 설정에는 다음이 포함됩니다.

- maxUnhealthy
- nodeStartupTimeout
- unhealthyConditions
- unhealthyRange

이 표에서는 지원되는 시스템 상태 점검 작업에 대해 설명합니다.

표 8-15. 시스템 상태 점검

필드	값	설명
`maxUnhealthy`	문자열 절대 숫자 또는 백분율	비정상 시스템 수가 이 값을 초과하면 업데이트 적용이 수행되지 않습니다.
`nodeStartupTimeout`	문자열 `XhXmXs`(시간, 분, 초) 형식의 기간	생성되는 시스템 중 클러스터에 가입하는 데 소요되는 기간보다 오래 걸리는 시스템은 실패한 것으로 간주되어 업데이트가 적용됩니다.

표 8-15. 시스템 상태 점검 (계속)

필드	값	설명
unhealthyConditions	unhealthyConditions 유형의 어레이 [] 사용 가능한 조건 유형: [Ready, MemoryPressure,DiskPressure, PIDPressure, NetworkUnavailable] 사용 가능한 조건 상태: [True, False, Unknown]	제어부 노드가 비정상으로 간주되는지 여부를 결정하는 조건 목록입니다.
unhealthyRange	문자열	"선택기"가 정상이 아닌 것으로 선택한 시스템 수가 unhealthyRange 범위 내에 있는 경우에만 추가 업데이트 적용이 허용됩니다. maxUnhealthy보다 우선합니다. 예를 들어 "[3-5]"는 (a) 비정상 시스템이 3개 이상 있고 (b) 비정상 시스템이 5개 이하로 있는 경우에만 업데이트 적용이 허용된다는 의미입니다.

참고 MachineHealthCheck 개체는 v1alpha3 클러스터에 대해 배포되지만 구성할 수 없습니다. 자세한 내용은 Kubectl을 사용하여 TKG 클러스터 시스템 상태 확인 항목을 참조하십시오.

MachineHealthCheck 예

다음 예에서는 특정 machineDeployment에 대한 machineHealthCheck를 구성합니다.

```
...
  topology:
    class: tanzukubernetescluster
    version: v1.28.8---vmware.1-fips.1-tkg.2
    controlPlane:
      machineHealthCheck:
        enable: true
        maxUnhealthy: 100%
        nodeStartupTimeout: 4h0m0s
        unhealthyConditions:
        - status: Unknown
          timeout: 5m0s
          type: Ready
       - status: "False"
          timeout: 12m0s
          type: Ready
      ...
    workers:
      machineDeployments:
      - class: node-pool
        failureDomain: np1
        machineHealthCheck:
          enable: true
          maxUnhealthy: 100%
```

```
    nodeStartupTimeout: 4h0m0s
    unhealthyConditions:
    - status: Unknown
      timeout: 5m0s
      type: Ready
    - status: "False"
      timeout: 12m0s
      type: Ready
```

Kubectl을 사용하여 MachineHealthCheck에 패치 적용

프로비저닝된 후 v1beta1 클러스터에 대한 `MachineHealthCheck`를 업데이트하려면 `patch` 메서드를 사용합니다.

경고 이 지침은 기존 클러스터에 패치를 적용하는 일반적인 지침을 제공합니다. 사용하는 값은 사용 중인 환경 및 패치를 적용하려는 배포된 클러스터에 따라 다릅니다. Tanzu CLI를 사용하여 기존 클러스터에 대한 `MachineHealthCheck`에 패치를 적용하는 것이 좋습니다.

1 클러스터 리소스 정의에서 `machineDeployment`를 가져옵니다.

```
kubectl get cluster CLUSTER_NAME -o yaml
```

`spec.topology.workers.machineDeployments` 섹션에서 각 `machineDeployment`를 식별하는 값이 표시됩니다.

2 작업자 노드 MachineHealthCheck를 삭제합니다.

```
kubectl patch cluster <Cluster Name> -n <cluster namespace>  --type json -p='{"op":
"replace", "path": "/spec/topology/workers/machineDeployments/<index>/machineHealthCheck",
"value":{"enable":false}}'
```

3 제어부 MachineHealthCheck를 삭제합니다.

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json -p='{"op":
"replace", "path": "/spec/topology/controlPlane/machineHealthCheck", "value":
{"enable":false}}'
```

4 제어부 MachineHealthCheck를 원하는 설정으로 생성하거나 업데이트합니다.

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json -p='[{"op":
"replace", "path": "/spec/topology/controlPlane/machineHealthCheck", "value":
{"enable":true,"nodeStartupTimeout":"1h58m","unhealthyConditions":
[{"status":"Unknown","timeout":"5m10s","type":"Unknown"},
{"status":"Unknown","timeout":"5m0s","type":"Ready"}],"maxUnhealthy":"100%"}}]'
```

5 작업자 노드 MachineHealthCheck를 원하는 설정으로 생성하거나 업데이트합니다.

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json -p='[{"op":
"replace", "path": "/spec/topology/workers/machineDeployments/<index>/machineHealthCheck",
"value":{"enable":true,"nodeStartupTimeout":"1h58m","unhealthyConditions":
[{"status":"Unknown","timeout":"5m10s","type":"Unknown"},
{"status":"Unknown","timeout":"5m0s","type":"Ready"}],"maxUnhealthy":"100%"}}]'
```

Tanzu CLI를 사용하여 MachineHealthCheck 구성

Tanzu CLI를 사용하여 v1beta1 클러스터에 대한 MachineHealthCheck를 구성할 수 있습니다.

예를 들어 다음 명령을 실행하여 제어부 MachineHealthCheck 설정을 생성하거나 업데이트합니다.

```
tanzu cluster mhc control-plane set <cluster-name> --node-startup-timeout 2h7m10s
```

다음 명령을 실행하여 설정이 업데이트되고 조정되지 않았는지 확인합니다.

```
tanzu cluster mhc control-plane get <cluster-name>
```

다음 명령을 실행하여 시스템 배포 MachineHealthCheck 설정을 생성하거나 업데이트합니다.

```
tanzu cluster mhc node set <cluster-name> --machine-deployment node-pool-1 --node-startup-
timeout 1h59m0s
```

다음 명령을 실행하여 설정이 업데이트되고 조정되지 않았는지 확인합니다.

```
tanzu cluster mhc node get <cluster-name> -m <cluster-name>-node-pool-1-nr7r5
```

get 및 set 외에도 시스템은 delete 작업을 지원합니다. 예:

제어부의 경우 다음 명령을 사용할 수 있습니다.

```
tanzu cluster mhc control-plane delete <cluster-name>
```

노드의 경우 다음 명령을 사용할 수 있습니다.

```
tanzu cluster mhc <cluster-name> --machine-deployment <machine deployment name>
```

TKG 서비스 클러스터 업데이트

9

이 섹션에서는 TKG 서비스 클러스터를 업데이트하는 방법에 대한 지침을 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해
- 업데이트를 위한 TKGS 클러스터 호환성 확인
- TKR 버전을 편집하여 TKG 클러스터 업데이트
- 스토리지 클래스를 편집하여 TKG 클러스터 업데이트
- VM 클래스를 편집하여 TKG 서비스 클러스터 업데이트
- Tanzu CLI를 사용하여 TKG 클러스터 업데이트

TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해

TKG 서비스 클러스터는 롤링 업데이트 모델을 지원합니다. 클러스터 규격을 변경하여 롤링 업데이트를 시작할 수 있습니다. 일부 시스템 작업은 롤링 업데이트를 시작할 수 있습니다. 환경을 업데이트하기 전에 롤링 업데이트 프로세스를 숙지해야 합니다.

TKG 서비스 3.0 이후 TKGS 클러스터를 위한 롤링 업데이트 모델

TKG 서비스 3.0 버전부터 TKG 컨트롤러는 vCenter Server 및 감독자와 독립적입니다. TKG 서비스 사용의 내용을 참조하십시오. 이러한 구성 요소를 업그레이드해도 TKGS 클러스터의 롤링 업데이트가 시작되지는 않습니다.

TKG 서비스 버전을 업그레이드하면 TKGS 클러스터의 롤링 업데이트가 트리거될 수 있습니다.

TKG 서비스 3.0 이전 TKGS 클러스터를 위한 롤링 업데이트 모델

TKG 컨트롤러는 감독자에서 실행됩니다. 감독자를 업데이트하면 TKG 컨트롤러가 자동으로 업데이트됩니다(업데이트를 사용할 수 있는 경우). 각 TKG 컨트롤러 업데이트에는 지원 서비스(예: CNI, CSI, CPI)에 대한 업데이트와 클러스터에 대한 구성 업데이트가 포함될 수 있습니다. 호환성을 지원하기 위해 시스템은 사전 검사를 수행하고 규정 준수를 적용합니다.

vSphere IaaS control plane는 롤링 업데이트 모델을 사용하여 감독자에서 TKG 클러스터를 업데이트합니다. 롤링 업데이트 모델은 클러스터 업데이트 프로세스 중에 다운타임이 최소화되도록 합니다. 롤링 업데이트에는 Kubernetes 소프트웨어 버전 및 클러스터를 지원하는 인프라 및 서비스(예: 가상 시스템 구성 및 리소스, 서비스 및 네임스페이스, 사용자 지정 리소스) 업그레이드가 포함됩니다. 업데이트가 성공하려면 구성이 몇 가지 호환성 요구 사항을 충족해야 합니다. 그래야 시스템이 재확인 조건을 적용하여 클러스터가 업데이트될 준비가 되었는지 확인하고 클러스터 업그레이드가 실패할 경우 롤백을 지원할 수 있습니다.

클러스터 매니페스트의 특정 측면을 변경하여 TKG 클러스터의 롤링 업데이트를 시작할 수 있습니다. 롤링 클러스터 업데이트는 시스템에 의해 시작될 수도 있습니다. 예를 들어 vSphere 네임스페이스 업데이트가 수행되면 시스템은 업데이트된 구성을 모든 워크로드 클러스터에 즉시 전파합니다. 이러한 업데이트는 클러스터 노드의 롤링 업데이트를 트리거할 수 있습니다. 또한 구성 요소를 변경하여 롤링 업데이트를 시작할 수도 있습니다. 예를 들어, 배포 버전에 해당하는 `VirtualMachineImage`을 교체하거나 이름을 변경하면, 시스템이 새 이미지에서 실행되는 모든 노드를 가져오려고 하면서 롤링 업데이트가 시작됩니다. 또한, 감독자를 업데이트하면 여기에 배포된 워크로드 클러스터의 롤링 업데이트가 트리거될 가능성이 높습니다. 예를 들어, `vmware-system-tkg-controller-manager`가 업데이트되면 새 값을 시스템이 매니페스트 생성기에 도입하고 이 값을 배포하기 위해 컨트롤러가 롤링 업데이트를 시작합니다.

클러스터 노드를 교체하는 롤링 업데이트 프로세스는 Kubernetes 배포의 포드에 대한 롤링 업데이트와 비슷합니다. 워크로드 클러스터의 롤링 업데이트를 수행하는 두 가지 개별 컨트롤러가 있습니다. 추가 기능 컨트롤러와 클러스터 컨트롤러입니다. 두 가지 컨트롤러 내에는 롤링 업데이트에 대한 세 가지 주요 단계인 추가 기능 업데이트, 제어부 업데이트 및 작업자 노드 업데이트 단계가 있습니다. 이러한 단계는 순서 대로 진행되며, 이전 단계가 충분히 진행될 때까지 단계를 시작하지 못하게 하는 사전 확인 기능이 있습니다. 이러한 단계가 불필요한 것으로 확인되면 건너뛸 수 있습니다. 예를 들어 업데이트가 작업자 노드에만 영향을 주기 때문에 추가 기능 또는 제어부 업데이트가 필요하지 않을 수 있습니다.

업데이트 프로세스 중에 시스템은 새 클러스터 노드를 추가하고 노드가 대상 Kubernetes 버전으로 온라인 상태가 될 때까지 기다립니다. 그런 다음, 시스템은 이전 노드를 삭제하도록 표시하고 다음 노드로 이동하여 프로세스를 반복합니다. 이전 노드는 모든 포드가 제거될 때까지 삭제되지 않습니다. 예를 들어, 노드가 완전히 드레이닝되는 것을 방지하는 PodDisruptionBudgets를 사용하여 포드를 정의하면 노드가 차단되지만 해당 포드를 제거할 때까지 제거되지 않습니다. 시스템은 모든 제어부 노드를 먼저 업그레이드한 후 작업자 노드를 업그레이드합니다. 업데이트하는 동안 클러스터 상태가 "업데이트 중"으로 변경됩니다. 롤링 업데이트 프로세스가 완료되면 클러스터 상태가 "실행 중"으로 변경됩니다.

복제 컨트롤러가 관리하지 않는 클러스터에서 실행되는 포드는 클러스터 업데이트 동안 작업자 노드 드레이닝의 일부로 Kubernetes 버전 업그레이드 중에 삭제됩니다. 이는 클러스터 업데이트가 vSphere 네임스페이스 또는 감독자 업데이트를 통해 수동으로 또는 자동으로 트리거되는 경우에 적용됩니다. 복제 컨트롤러에서 관리되지 않는 포드에는 배포 또는 ReplicaSet 규격의 일부로 생성되지 않는 포드가 포함됩니다. 자세한 내용은 Kubernetes 설명서에서 포드 수명 주기: 포드 수명을 참조하십시오.

사용자 시작 롤링 업데이트

Tanzu Kubernetes 릴리스 버전을 업데이트하고 가상 시스템 클래스를 업데이트하고 스토리지 클래스를 업데이트하여 감독자에서 TKG 클러스터의 롤링 업데이트를 시작할 수 있습니다. 자세한 내용은 다음 항목 중 하나를 참조하십시오.

- TKR 버전을 편집하여 TKG 클러스터 업데이트
- 스토리지 클래스를 편집하여 TKG 클러스터 업데이트
- VM 클래스를 편집하여 TKG 서비스 클러스터 업데이트
- Tanzu CLI를 사용하여 TKG 클러스터 업데이트

시스템 시작 롤링 업데이트

감독자의 각 릴리스에서 다음 개체 중 하나 이상이 변경될 수 있습니다.

- kubeadmcontrolplanetemplate/kubeadmcontrolplane
- kubeadmconfigtemplate/kubeadmconfig
- vspheremachinetemplate/vspheremachine(vSphere 8.x용)
- wcpmachinetemplate/wcpmachine(vSphere 7.x용)

감독자가 업그레이드되면 핵심 CAPI(Cluster API) 컨트롤러는 위 개체의 원하는 상태를 실행 중인 워크로드 클러스터와 일치시키기 위해 TKG 워크로드 클러스터에 대한 업데이트 롤아웃을 트리거합니다.

vSphere IaaS control plane에서는 감독자에서 실행되는 TKG 컨트롤러가 이러한 개체를 생성하고 시스템 코드와 동기화된 상태로 유지합니다. 즉, 컨트롤러가 최신 코드로 업데이트될 때 위의 개체 중 하나가 변경되면 기존 TKG 클러스터의 롤링 업데이트를 유발합니다. 요컨대, 감독자에 영향을 미치는 시스템 코드 변경으로 인해 TKG 클러스터의 롤링 업데이트가 발생합니다.

이 표에서는 감독자가 업그레이드될 때마다 워크로드 클러스터의 자동화된 롤링 업데이트를 기대할 수 있는 조건을 설명합니다.

업그레이드 시나리오	설명
vCenter Server 7.x 릴리스에서 vCenter Server 릴리스로 업그레이드	모든 Tanzu Kubernetes 클러스터의 롤링 업데이트를 트리거할 수 있습니다. 롤링 업데이트는 vCenter Server 업그레이드 후 감독자를 처음 업그레이드하면 트리거됩니다. 롤링 업데이트는 일반적으로 동일한 vCenter Server에서 감독자를 업그레이드해도 트리거되지 않습니다. 구체적인 세부 내용은 릴리스 정보를 참조하십시오.
vCenter Server 릴리스에서 vCenter Server 8.x 릴리스로 업그레이드	다음 코드 변경 사항을 전파해야 하기 때문에 모든 TKG 클러스터의 롤링 업데이트가 트리거됩니다. ■ 기본 CAPI 제공자를 CAPW에서 CAPV로 이동해야 함 ■ 클래스 없는 CAPI 클러스터에서 고급 CAPI 클러스터로 클러스터 마이그레이션

업그레이드 시나리오	설명
vCenter Server 8.0 GA 릴리스(8.0.0)에서 vCenter Server 8.0.0b 또는 8.0.0c 릴리스로 업그레이드	다음 경우 중 하나라도 해당되는 경우 지정된 TKG 클러스터의 롤링 업데이트가 트리거됩니다. ■ 비어 있지 않은 noProxy 목록이 있는 프록시 설정을 사용하고 있던 모든 TKG 클러스터 ■ 내장된 Habor 레지스트리 서비스가 감독자에서 사용하도록 설정된 경우 모든 TKG 클러스터
vSphere 8.0.0b 릴리스에서 vSphere 8.0.0c 릴리스로 업그레이드	워크로드 클러스터의 자동 롤아웃 없음
vSphere 8.0.0c 릴리스에서 vSphere 8.0 업데이트 1 릴리스(8.0.1)로 업그레이드	워크로드 클러스터의 자동 롤아웃 없음
vSphere 8.x 버전에서 8.0 U2 릴리스(8.0.2)로 업그레이드	다음 변경이 필요하므로 모든 TKC에 대한 롤링 업그레이드가 발생합니다. ■ vSphere 8.0 U2에는 ClusterClass의 일부로 GCM의 TKG 1.0 및 TKG 2.0 TKR 모두에 대한 Kubernetes 수준 STIG 변경 사항이 포함되어 있습니다. ■ 1.23 이상의 TKC는 8.0U2와 호환되므로 모든 클러스터는 롤링 업그레이드를 거칩니다.
8.0 U2(8.0.2) 미만의 모든 vSphere 8.x 버전에서 8.0 U2c 릴리스로 업그레이드	다음 변경이 필요하므로 모든 TKC에 대한 롤링 업그레이드가 발생합니다. ■ 8.0U2에는 클러스터 클래스의 일부로 GCM의 TKG1.0 및 TKG2.0 TKR 모두에 대한 Kubernetes 수준 STIG 변경 사항이 포함되어 있습니다. ■ 1.23 이상의 TKC는 8.0 P03과 호환되므로 모든 클러스터가 롤링 업그레이드됩니다.

또한 TKR 이미지를 호스팅하는 컨텐츠 라이브러리를 변경하면 TKG 클러스터의 롤링 업데이트가 트리거될 수 있습니다. 구독을 통해 또는 수동으로 새 이미지를 추가하면 TKG 클러스터의 롤링 업데이트가 트리거되지 않습니다. 하지만 컨텐츠 라이브러리를 변경하고 이름이 다른 이미지를 추가하면 모든 TKG 클러스터의 롤링 업데이트가 트리거됩니다.

예를 들어 시스템 정의 OVA 이름을 자동으로 사용하는 구독 컨텐츠 라이브러리를 사용하는 시나리오를 고려해 봅니다. 그런 다음 로컬 컨텐츠 라이브러리로 전환하고 동일한 OVA로 채우지만 다른 이름을 지정합니다. 이렇게 하면 모든 TKG 클러스터의 롤링 업데이트가 트리거됩니다. 교체 콘텐츠 라이브러리의 OVA는 동일하지만 사용자 정의 이름이 다르기 때문입니다.

여러 노드 풀이 있는 클러스터의 롤링 업데이트 고려 사항

여러 노드 풀이 있는 TKG 클러스터를 사용하는 경우 롤링 업데이트와 관련하여 다음 정보를 고려하십시오.

작업자 노드 풀

작업자 노드 풀은 vSphere 7 U3와 함께 릴리스된 TKGS v1alpha2 API를 통해 도입되었습니다. 클러스터 API MachineDeployments는 작업자 노드 풀의 기본 Kubernetes입니다.

ClusterClass는 TKGS의 vSphere 8 릴리스에서 도입되었습니다. v1alpha3와 v1beta1 API는 모두 ClusterClass를 기반으로 합니다. (v1alpha3는 ClusterClass 위에 있는 추상화 계층입니다.)

롤링 업데이트 중에 여러 노드 풀이 업데이트되는 방법

여러 노드 풀로 프로비저닝된 TKGS 워크로드 클러스터를 업데이트하는 경우 롤링 업데이트 모델은 사용 중인 vSphere 버전에 따라 달라집니다.

vSphere	TKGS API	업그레이드 동작
vSphere 7 TKGS	v1alpha2 API	동일한 클러스터 내의 여러 노드 풀이 한 번에 업데이트됨(동시에)
vSphere 8 TKGS	v1alpha3 API 및 v1beta1 API	동일한 클러스터 내의 여러 노드 풀이 논리적 순서에 따라 업데이트됨(순차적)

모범 사례 고려 사항

동일한 노드 풀이 여러 개 있는 vSphere 8 TKGS 클러스터를 프로비저닝하는 것은 크기 조정 측면에서 아무런 의미가 없습니다. 노드 풀은 서로 다른 크기, VM 클래스, TKr 버전 등에 사용해야 합니다. 동일한 노드 풀을 여러 개 사용하여 편법을 써서 클러스터를 더 빨리 업그레이드하는 것은 작동하지 않으므로 피하십시오.

포드 중단 예산은 업그레이드가 애플리케이션 실행을 방해하지 않도록 하는 적절한 방법입니다. 이 작업을 처리하는 가장 좋은 방법은 워크로드에 PodDisruptionBudgets를 설정하는 것입니다(https://kubernetes.io/docs/tasks/run-application/configure-pdb/ 참조). 클러스터 API는 이러한 항목을 준수하며 임계값을 초과할 경우 시스템을 종료하지 않습니다.

vSphere 8 TKGS 클러스터의 롤링 업데이트 세부 정보

vSphere 8 TKGS 클러스터 버전을 업데이트하는 중 다음이 발생합니다.

- 제어부 노드가 먼저 업데이트된 다음 작업자 노드가 영역-A 노드 풀부터 시작하여 한 번에 하나씩 롤링됩니다. 두 개의 노드 풀을 사용하는 경우 한 번에 1개의 작업자만 롤아웃됩니다.

클러스터 구성 변수를 업데이트하는 중 다음이 발생합니다.

- 제어부 노드가 먼저 업데이트된 다음 노드 풀당 하나의 작업자 노드가 롤링됩니다. 예를 들어 두 개의 노드 풀을 사용하는 경우 한 번에 2개의 작업자가 롤아웃됩니다.

업데이트를 위한 TKGS 클러스터 호환성 확인

TKGS 워크로드 클러스터를 업그레이드하기 전에 업그레이드를 위한 호환성을 확인해야 합니다. TKG 서비스에 대한 호환성을 확인해야 합니다.

TKG 서비스와의 호환성 확인

워크로드 클러스터를 업그레이드하기 전에 업그레이드를 위한 호환성을 확인해야 합니다. 클러스터가 TKG 서비스와 호환되지 않는 경우 Tanzu Kubernetes 릴리스를 업그레이드합니다. 사용 가능한 TKr에 대한 자세한 내용은 릴리스 정보를 참조하십시오. 온라인 상호 운용성 매트릭스도 참조하십시오.

다음 명령을 사용하여 Tanzu Kubernetes 릴리스를 나열하고 호환성을 볼 수 있습니다.

```
kubectl get tkr
```

COMPATIBLE 열은 Tanzu Kubernetes 릴리스가 설치된 TKG 서비스와 호환되는지 여부를 나타냅니다. TKG 서비스 3.1 릴리스부터 TYPE 열에 호환성 상태도 반환됩니다.

TKGS 클러스터를 지정하면 사용할 수 있는TKr 업데이트를 볼 수 있습니다.

v1alpha3 API 사용:

```
kubectl get tkc <tkgs-cluster-name>
```

또는 v1beta1 API 사용:

```
kubectl get cc <tkgs-cluster-name>
```

UPDATES AVAILABLE 열은 사용 가능한 Kubernetes 업그레이드가 있는지 그리고 사용이 권장되는 다음 Tanzu Kubernetes 릴리스가 있는지 여부를 나타냅니다. 예:

```
kubectl get tkc tkg2-cluster-11-tkc
NAME                  CONTROL PLANE   WORKER   TKR NAME                          AGE
READY   TKR COMPATIBLE   UPDATES AVAILABLE
tkg2-cluster-11-tkc   3               3        v1.25.7---vmware.3-fips.1-tkg.1   13d
True    True             [v1.26.5+vmware.2-fips.1-tkg.1]
```

TKr 형식에는 비레거시 형식과 레거시 형식의 두 가지 유형이 있습니다.

- 비레거시 TKr은 vSphere 8.x용으로 특별히 구축되었으며 vSphere 8.x하고만 호환됩니다.
- 레거시 TKr은 vSphere 7.x와 호환되는 레거시 형식을 사용하지만 업그레이드 목적으로만 vSphere 8.x와도 호환됩니다.

비레거시 TKr을 나열하려면 다음을 수행합니다.

```
kubectl get -l !run.tanzu.vmware.com/legacy-tkr
```

레거시 TKr을 나열하려면 다음을 수행합니다.

```
kubectl get -l !run.tanzu.vmware.com/legacy-tkr
```

TKR 버전을 편집하여 TKG 클러스터 업데이트

이 작업에서는 TKG 클러스터 매니페스트를 편집하여 TKG 클러스터에 대한 Tanzu Kubernetes 릴리스 버전을 업데이트하는 방법에 대해 설명합니다.

kubectl edit 명령을 사용하여 Tanzu Kubernetes 릴리스 버전을 업그레이드하면 TKGS 클러스터의 롤링 업데이트를 시작할 수 있습니다.

참고 kubectl apply 명령을 사용하여 배포된 클러스터의 TKR 버전을 업데이트할 수는 없습니다.

사전 요구 사항

이 작업을 수행하려면 kubectl edit 명령을 사용해야 합니다. 이 명령은 KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 클러스터 매니페스트를 텍스트 편집기에서 엽니다. 파일을 저장하면 클러스터가 변경 사항으로 업데이트됩니다. `kubectl edit` 명령을 실행할 수 있도록 kubectl에 대한 편집기를 구성하려면 Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

절차

1 감독자로 인증합니다.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 대상 워크로드 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 대상 TKG 클러스터 및 버전을 가져옵니다.

v1alpha3 클러스터:

```
kubectl get tanzukubernetescluster
```

v1beta1 클러스터:

```
kubectl get cluster
```

4 사용 가능한 Tanzu Kubernetes 릴리스를 나열합니다.

```
kubectl get tanzukubernetesreleases
```

5 다음 명령을 실행하여 클러스터 매니페스트를 편집합니다.

v1alpha3 클러스터:

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1 클러스터:

```
kubectl edit cluster/CLUSTER-NAME
```

6 Tanzu Kubernetes 릴리스 문자열을 업데이트하여 매니페스트를 편집합니다.

예를 들어 v1alpha3 클러스터의 경우 TKR v1.25.7에서:

```
topology:
  controlPlane:
   replicas: 1
   storageClass: vsan-default-storage-policy
   tkr:
    reference:
     name: v1.25.7---vmware.3-fips.1-tkg.1
```

```
    vmClass: guaranteed-large
  nodePools:
  - name: worker-tkg-pool01
    replicas: 3
    storageClass: vsan-default-storage-policy
    tkr:
      reference:
        name: v1.25.7---vmware.3-fips.1-tkg.1
    vmClass: guaranteed-large
    volumes:
    - capacity:
        storage: 128Gi
      mountPath: /var/lib/containerd
      name: containerd
```

TKR v1.26.5로 변경하는 경우:

```
topology:
  controlPlane:
    replicas: 1
    storageClass: vsan-default-storage-policy
    tkr:
      reference:
        name: v1.26.5---vmware.2-fips.1-tkg.1
    vmClass: guaranteed-large
  nodePools:
  - name: worker-tkg-pool01
    replicas: 3
    storageClass: vsan-default-storage-policy
    tkr:
      reference:
        name: v1.26.5---vmware.2-fips.1-tkg.1
    vmClass: guaranteed-large
    volumes:
    - capacity:
        storage: 128Gi
      mountPath: /var/lib/containerd
      name: containerd
```

참고 제어부 및 작업자 노드의 TKR 버전이 동일해야 합니다. 모든 TKr 인스턴스를 업데이트하거나, 제어부 버전을 업데이트하고 작업자 노드에서 TKR 이름을 제거할 수 있습니다.

예를 들어 v1beta1 클러스터의 경우 TKR v1.25.7에서:

```
apiVersion: cluster.x-k8s.io/v1beta1
  ...
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
```

```
      ...
    variables:
      ...
```

TKR v1.26.5로 변경하는 경우:

```
apiVersion: cluster.x-k8s.io/v1beta1
  ...
  topology:
    class: tanzukubernetescluster
    version: v1.26.5---vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      ...
    variables:
      ...
```

7 변경 내용을 매니페스트 파일에 적용합니다.

파일을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 감독자의 가상 시스템 서비스는 새 작업자 노드를 프로비저닝합니다.

8 kubectl이 매니페스트 편집이 기록되었다고 보고하는지 확인합니다.

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

참고 오류가 발생하거나 kubectl이 클러스터 매니페스트가 편집되었다고 보고하지 않는 경우에는 KUBE_EDITOR 환경 변수를 사용하여 기본 텍스트 편집기를 올바르게 구성했는지 확인합니다. Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

9 클러스터가 업데이트되고 있는지 확인합니다.

```
kubectl get tanzukubernetescluster
NAME             CONTROL PLANE   WORKER   DISTRIBUTION                      AGE   PHASE
tkgs-cluster-1   3               3        v1.26.5---vmware.2-fips.1-tkg.1   21h   updating
```

10 클러스터가 업데이트되었는지 확인합니다.

```
kubectl get tanzukubernetescluster
NAME             CONTROL PLANE   WORKER   DISTRIBUTION                      AGE   PHASE
tkgs-cluster-1   3               3        v1.26.5---vmware.2-fips.1-tkg.1   22h   running
```

스토리지 클래스를 편집하여 TKG 클러스터 업데이트

클러스터 노드에서 사용하는 스토리지 클래스를 변경하여 TKG 클러스터를 업데이트할 수 있습니다.

`kubectl edit` 명령을 사용하여 클러스터 규격에서 `storageClass` 매개 변수의 값을 편집하면 TKG 클러스터의 롤링 업데이트를 시작할 수 있습니다.

참고 `kubectl apply` 명령을 사용하여 배포된 TKG 클러스터를 업데이트할 수는 없습니다.

사전 요구 사항

이 작업을 수행하려면 kubectl edit 명령을 사용해야 합니다. 이 명령은 KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 클러스터 매니페스트를 텍스트 편집기에서 엽니다. 파일을 저장하면 클러스터가 변경 사항으로 업데이트됩니다. kubectl용 편집기를 구성하려면 Kubectl용 텍스트 편집기 구성 항목을 참조하십시오.

절차

1 감독자로 인증합니다.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 대상 워크로드 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 사용 가능한 스토리지 클래스를 확인하고 무엇을 사용할지 결정하려면 다음 명령을 실행합니다.

```
kubectl describe tanzukubernetescluster CLUSTER-NAME
```

4 다음 명령을 실행하여 클러스터 매니페스트를 편집합니다.

v1alpha3 클러스터:

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1 클러스터:

```
kubectl edit cluster/CLUSTER-NAME
```

5 `storageClass` 값을 변경하여 매니페스트를 편집합니다.

예를 들어 v1alpaha3 클러스터의 경우 제어부 및 작업자 노드에 대한 `silver-storage-class` 클래스의 클러스터 매니페스트를:

```
spec:
  topology:
    controlPlane:
      ...
      storageClass: silver-storage-class
    workers:
      ...
      storageClass: silver-storage-class
```

제어부 및 작업자 노드에 대한 `gold-storage-class` 클래스로 변경합니다.

```
spec:
  topology:
    controlPlane:
      ...
      storageClass: gold-storage-class
    workers:
      ...
      storageClass: gold-storage-class
```

마찬가지로 v1beta1 클러스터를 프로비저닝한 경우 클러스터 규격의 `variables.storageclass` 값을 스토리지 클래스의 이름으로 업데이트합니다.

6 변경 내용을 매니페스트 파일에 적용합니다.

파일을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 Tanzu Kubernetes Grid는 새 노드 VM을 프로비저닝하고 이전 노드를 스핀 다운합니다.

7 kubectl이 매니페스트 편집이 기록되었다고 보고하는지 확인합니다.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-1 edited
```

참고 오류가 발생하거나 kubectl이 클러스터 매니페스트가 편집되었다고 보고하지 않는 경우에는 KUBE_EDITOR 환경 변수를 사용하여 기본 텍스트 편집기를 올바르게 구성했는지 확인합니다. Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

8 클러스터가 업데이트되었는지 확인합니다.

v1alpha3 클러스터:

```
kubectl get tanzukubernetescluster
```

v1beta1 클러스터:

```
kubectl get cluster
```

VM 클래스를 편집하여 TKG 서비스 클러스터 업데이트

클러스터 노드를 호스팅하는 데 사용되는 가상 시스템 클래스를 변경하여 TKG 서비스 클러스터를 업데이트할 수 있습니다.

`kubectl edit` 명령을 사용하여 `vmClass` 정의를 편집하면 TKG 서비스 클러스터의 롤링 업데이트를 시작할 수 있습니다. 변경된 클래스를 기반으로 하는 새 노드가 롤아웃되고 이전 노드는 스핀다운됩니다.

참고 `kubectl apply` 명령을 사용하여 배포된 TKG 클러스터를 업데이트할 수는 없습니다.

사전 요구 사항

이 작업을 수행하려면 kubectl edit 명령을 사용해야 합니다. 이 명령은 KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 클러스터 매니페스트를 텍스트 편집기에서 엽니다. 파일을 저장하면 클러스터가 변경 사항으로 업데이트됩니다. kubectl용 편집기를 구성하려면 Kubectl용 텍스트 편집기 구성 항목을 참조하십시오.

절차

1 감독자로 인증합니다.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 대상 TKG 클러스터에 describe 명령을 실행하고 VM 클래스를 확인합니다.

v1alpha3 클러스터:

```
kubectl describe tanzukubernetescluster CLUSTER-NAME
```

v1beta1 클러스터:

```
kubectl describe cluster CLUSTER-NAME
```

4 클러스터가 프로비저닝된 경우 vSphere 네임스페이스에서 사용 가능한 VM 클래스를 나열하고 설명합니다.

```
kubectl get virtualmachineclass
```

참고 TKG 클러스터가 프로비저닝된 경우 대상 VM 클래스는 vSphere 네임스페이스와 연결되어야 합니다. VM 클래스를 vSphere 네임스페이스에 바인딩하는 방법에 대한 자세한 내용은 TKG 서비스 또는 VM 서비스 문서를 참조하십시오.

5 다음 명령을 실행하여 클러스터 매니페스트를 편집합니다.

v1alpha3 클러스터:

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1 클러스터:

```
kubectl edit cluster/CLUSTER-NAME
```

6 VM 클래스 문자열을 변경하여 매니페스트를 편집합니다.

예를 들어 v1alpah3 클러스터를 사용하는 경우 클러스터 매니페스트를 작업자 노드에 대해 `guaranteed-medium` VM 클래스 사용에서:

```
  topology:
```

```
    controlPlane:
      replicas: 3
      storageClass: vwk-storage-policy
      tkr:
        reference:
          name: v1.27.11---vmware.1-fips.1-tkg.2
      vmClass: guaranteed-medium
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      storageClass: vwk-storage-policy
      tkr:
        reference:
          name: v1.27.11---vmware.1-fips.1-tkg.2
      vmClass: guaranteed-medium
```

작업자 노드에 대해 `guaranteed-large` VM 클래스를 사용으로 변경합니다.

```
topology:
   controlPlane:
     replicas: 3
     storageClass: vwk-storage-policy
     tkr:
       reference:
         name: v1.27.11---vmware.1-fips.1-tkg.2
     vmClass: guaranteed-medium
   nodePools:
   - name: worker-nodepool-a1
     replicas: 3
     storageClass: vwk-storage-policy
     tkr:
       reference:
         name: v1.27.11---vmware.1-fips.1-tkg.2
     vmClass: guaranteed-large
```

마찬가지로 v1beta1 클러스터를 프로비저닝한 경우 `variables.vmclass` 값을 대상 VM 클래스로 업데이트합니다.

7 변경 내용을 매니페스트 파일에 적용합니다.

파일을 저장하면 kubectl이 변경 내용을 클러스터에 적용합니다. 백그라운드에서 TKG 컨트롤러는 새 노드 VM을 프로비저닝하고 이전 노드를 스핀 다운합니다.

8 kubectl이 매니페스트 편집이 기록되었다고 보고하는지 확인합니다.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-1 edited
```

참고 오류가 발생하거나 kubectl이 클러스터 매니페스트가 편집되었다고 보고하지 않는 경우에는 KUBE_EDITOR 환경 변수를 사용하여 기본 텍스트 편집기를 올바르게 구성했는지 확인합니다. Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

9 클러스터가 업데이트되었는지 확인합니다.

v1alpha3 클러스터:

```
kubectl get tanzukubernetescluster
```

v1beta1 클러스터:

```
kubectl get cluster
```

Tanzu CLI를 사용하여 TKG 클러스터 업데이트

Tanzu CLI로 Tanzu Kubernetes 릴리스 버전을 업그레이드하여 TKG 클러스터를 업데이트합니다.

Tanzu CLI를 사용하여 Tanzu Kubernetes 릴리스 버전을 업그레이드하면 TKGS 클러스터의 롤링 업데이트를 시작할 수 있습니다.

전체 사용 세부 정보는 "Tanzu CLI 참조 가이드" 를 참조하십시오.

사전 요구 사항

TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치.

절차

1 감독자로 인증합니다.

2 TKG 클러스터를 나열합니다.

```
tanzu cluster list
```

3 TKG 클러스터를 업데이트합니다.

```
tanzu cluster upgrade CLUSTER-NAME --tkr TKR-NAME -n VSPHERE-NAMESPACE
```

형식 설명:

- `CLUSTER-NAME`은 업그레이드 대상인 TKG 클러스터의 이름입니다.
- `TKR-NAME`은 TKR 버전 문자열입니다.
- `VSPHERE-NAMESPACE`는 TKG 클러스터가 프로비저닝되는 vSphere 네임스페이스의 이름입니다.

예:

```
tanzu cluster upgrade tkg-cluster-1 --tkr v1.23.8---vmware.2-tkg.2-zshippable -n tkg2-cluster-ns
```

4 클러스터 업그레이드를 확인합니다.

클러스터가 업그레이드되면 다음과 유사한 메시지가 표시됩니다.

```
Cluster 'tkg-cluster-1' successfully upgraded to kubernetes version 'v1.23.8+vmware.2-
tkg.2-zshippable'
```

TKG 서비스 클러스터 자동 스케일링

10

이 섹션에서는 TKG 서비스 클러스터 자동 스케일링에 대한 정보를 제공합니다.

다음으로 아래 항목을 읽으십시오.

클러스터 자동 스케일링 정보

클러스터 Autoscaler를 배포하여 워크로드 요구량에 따라 TKG 서비스 클러스터의 작업자 노드 수를 자동으로 조정할 수 있습니다.

클러스터 자동 스케일링 정보

TKG 서비스 클러스터 Autoscaler는 Kubernetes 클러스터 Autoscaler를 구현한 것입니다. 자세한 내용은 클러스터 Autoscaler 설명서를 참조하십시오.

클러스터 Autoscaler는 클러스터 노드의 확장 및 축소를 지원합니다. 다중 영역 감독자에서 클러스터를 실행하는 경우 Autoscaler는 특정 가용성 영역에 할당된 노드 풀을 확장/축소할 수 있습니다.

클러스터 Autoscaler는 Kubectl 또는 Tanzu CLI를 사용하여 클러스터에 설치하는 표준 패키지로 제공됩니다. 클러스터 Autoscaler는 서비스 계정 자격 증명을 사용하여 TKG 클러스터에서 배포로 실행됩니다.

Autoscaler 패키지 부 버전과 TKr 부 버전 간에는 일대일 관계가 있습니다. 예를 들어 TKr 1.27.11을 사용하는 경우 Autoscaler v1.27.2를 설치해야 합니다. 버전이 일치하지 않으면 패키지 조정이 실패합니다.

클러스터 Autoscaler는 작업자 노드 확장 및 축소를 모두 지원하지만 일부 유형의 애플리케이션에서는 노드를 축소하지 못하도록 막기 때문에 클러스터 Autoscaler가 노드를 축소하지 못하는 경우가 있습니다. "CA가 노드를 제거하지 못하게 막을 수 있는 포드 유형은 무엇입니까?" 부분을 클러스터 Autoscaler 설명서에서 참조하십시오.

버전 요구 사항

클러스터 Autoscaler에는 다음과 같은 버전 요구 사항이 있습니다.

- 최소 vSphere 버전은 vSphere 8 U3입니다.
- 최소 TKr 버전은 vSphere 8의 경우 TKr 1.27.x입니다.
- TKr의 부 버전과 클러스터 Autoscaler 패키지의 부 버전이 일치해야 합니다.

패키지 요구 사항

클러스터 Autoscaler는 표준 패키지로 제공됩니다. 패키지의 부 버전이 사용 중인 TKr의 부 버전과 일치해야 합니다. 예를 들어 TKr 1.27.11을 사용하는 경우 Autoscaler v1.27.2를 설치해야 합니다. 버전이 일치하지 않으면 패키지 조정이 실패합니다.

후속 저장소 버전에서 대상 패키지를 찾아야 할 수도 있습니다. 예를 들어 Autoscaler v1.27.2는 표준 패키지 저장소의 v2024.4.12 버전에 있습니다. 이후 Autoscaler 패키지 버전(예: 1.28.x, 1.29.x, 1.30.x 등)은 후속 저장소 버전에 있습니다. 모든 표준 패키지 저장소는 다음 명령을 실행하여 찾을 수 있습니다.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

워크플로

클러스터 자동 스케일링을 사용하도록 설정하기 위한 개략적인 워크플로는 다음과 같습니다.

1 Autoscaler 주석을 사용하여 새 TKG 클러스터를 생성하거나 기존 TKG 클러스터를 업데이트하고 `spec.topology.workers.machinedeployments`에서 복제 필드를 제거합니다.

2 생성했거나 업데이트한 TKG 클러스터에 패키지 저장소를 설치합니다.

3 생성했거나 업데이트한 TKG 클러스터에 Autoscaler 패키지를 설치합니다.

Autoscaler는 kube-system 네임스페이스에 배포로 TKG 클러스터에 설치됩니다.

자세한 지침은 다음 항목을 참조하십시오.

- Kubectl을 사용하여 클러스터 Autoscaler 설치
- Tanzu CLI를 사용하여 클러스터 Autoscaler 설치

Kubectl을 사용하여 클러스터 Autoscaler 설치

kubectl을 사용하여 클러스터 Autoscaler 패키지를 설치하고 구성하려면 다음 지침을 참조하십시오.

요구 사항

다음 요구 사항을 준수합니다.

- 최소 vSphere 버전은 vSphere 8 U3입니다.
- 최소 TKr 버전은 vSphere 8의 경우 TKr 1.27.x입니다.

- TKr의 부 버전과 클러스터 Autoscaler 패키지의 부 버전이 일치해야 합니다.

주의 Autoscaler 패키지 부 버전과 TKr 부 버전 간에는 일대일 관계가 있습니다. 예를 들어 TKr 1.27.11을 사용하는 경우 Autoscaler v1.27.2를 설치해야 합니다. 버전이 일치하지 않으면 패키지 조정이 실패합니다.

vSphere 네임스페이스 구성

TKG 클러스터 프로비저닝을 위한 다음 사전 요구 사항 작업을 완료합니다.

1 vSphere 8의 경우 vSphere 8 U3 및 TKr 1.27.x로 환경을 설치하거나 업데이트합니다.

2 최신 Tanzu Kubernetes 릴리스로 컨텐츠 라이브러리를 생성하거나 업데이트합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.

3 TKG 클러스터를 호스팅하기 위한 vSphere 네임스페이스를 생성하고 구성합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

4 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

   다음 예는 명령줄에서 도구를 설치하는 데 사용할 수 있습니다. 추가 지침은 vSphere에 대한 Kubernetes CLI 도구 설치 항목을 참조하십시오.

```
curl -LOk https://${SUPERVISOR_IP-or-FQDN}/wcp/plugin/linux-amd64/vsphere-plugin.zip
unzip vsphere-plugin.zip
mv -v bin/* /usr/local/bin/
```

5 `kubectl` 및 `kubectl vsphere`를 실행하여 설치를 확인합니다.

Autoscaler 주석을 사용하여 TKG 클러스터 생성

지침에 따라 TKG 클러스터를 생성합니다. 추가 지침은 Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로 항목을 참조하십시오.

Autoscaler를 사용하려면 여기에 제공된 클러스터 규격 예에 설명된 대로 Autoscaler 레이블 주석으로 클러스터를 구성해야 합니다. 일반 클러스터 프로비저닝과 달리 작업자 노드 복제본 수를 하드 코딩하지 않습니다. Kubernetes에는 Autoscaler 최소 및 최대 크기 주석을 기반으로 복제본에 대한 기본 논리가 기본 제공됩니다. 이는 새 클러스터이므로 최소 크기는 클러스터를 생성하는 데 사용됩니다. 자세한 내용은 https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling의 내용을 참조하십시오.

1 kubectl을 사용하여 감독자에서 인증합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 클러스터를 호스팅할 대상 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkgs-cluster-namespace
```

3 vSphere 네임스페이스에서 사용할 수 있는 VM 클래스를 나열합니다.

대상 vSphere 네임스페이스에 바인딩된 VM 클래스만 사용할 수 있습니다. TKG 서비스 클러스터에서 VM 클래스 사용의 내용을 참조하십시오.

4 사용 가능한 영구 볼륨 스토리지 클래스를 나열합니다.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

이 명령은 스토리지 클래스를 포함하여 vSphere 네임스페이스에 대한 세부 정보를 반환합니다. `kubectl describe storageclasses` 명령도 사용 가능한 스토리지 클래스를 반환하지만 vSphere 관리자 권한이 필요합니다.

5 사용 가능한 Tanzu Kubernetes 릴리스를 나열합니다.

```
kubectl get tkr
```

이 명령은 이 vSphere 네임스페이스에서 사용할 수 있는 TKr과 해당 호환성을 반환합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.

6 수집한 정보를 사용하여 필요한 클러스터 Autoscaler 구성으로 TKG 클러스터 규격 YAML 파일을 만듭니다.

- 작업자 nodePools에 대한 `*-min-size` 및 `*-max-size` 주석을 사용합니다. 이 예에서 3은 최소값이고 5는 확장 가능한 최대 작업자 노드 수입니다. 기본적으로 클러스터는 작업자 노드 3개를 사용하여 생성됩니다.
- TKr 및 Autoscaler 패키지에 대해 일치하는 부 버전을 사용합니다.
- 사용된 클러스터 `metadata.name` 및 `metadata.namespace` 값은 Autoscaler 패키지 기본값과 일치합니다. 클러스터 규격에서 이러한 값을 변경하는 경우 `autoscaler-data-values`에서 수정해야 합니다(아래 참조).

```
#cc-autoscaler.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
 name: gc1
 namespace: cluster
spec:
 clusterNetwork:
   pods:
     cidrBlocks:
     - 192.0.2.0/16
   serviceDomain: cluster.local
   services:
     cidrBlocks:
     - 198.51.100.0/12
 topology:
   class: tanzukubernetescluster
   controlPlane:
     metadata: {}
     replicas: 3
   variables:
```

```
  - name: storageClasses
    value:
    - wcpglobal-storage-profile
  - name: vmClass
    value: guaranteed-medium
  - name: storageClass
    value: wcpglobal-storage-profile
  #minor versions must match
  version: v1.27.11---vmware.1-fips.1-tkg.2
  workers:
    machineDeployments:
    - class: node-pool
      metadata:
        annotations:
          cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size: "3"
          cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size: "5"
      name: np-1
```

7 클러스터 규격을 적용합니다.

```
kubectl apply -f cc-autoscaler.yaml
```

8 클러스터 생성을 확인합니다.

```
kubectl get cluster,vm
```

9 클러스터 노드 버전을 확인합니다.

```
kubectl get node
```

TKG 클러스터에 패키지 관리자를 생성합니다.

TKG 클러스터가 프로비저닝되면 클러스터에 패키지 관리자를 설치하고 패키지 저장소를 설정합니다.

1 프로비저닝한 TKG 클러스터에 로그인합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN \
--vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

2 Carvel imgpkg 도구를 설치합니다.

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

3 `imgpkg version`을 실행하여 설치를 확인합니다.

4 패키지 저장소 버전을 확인합니다.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

5 패키지 저장소를 설치합니다. 그에 따라 저장소 버전을 업데이트합니다.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageRepository
metadata:
  name: tanzu-standard
  namespace: tkg-system
spec:
  fetch:
    imgpkgBundle:
      image: projects.registry.vmware.com/tkg/packages/standard/repo:v2024.4.12
```

6 패키지 저장소를 확인합니다.

```
kubectl get packagerepository -A
NAMESPACE   NAME       AGE   DESCRIPTION
tkg-system  tanzu-standard  2m22s  Reconcile succeeded
```

7 클러스터 Autoscaler 패키지가 있는지 확인합니다.

```
kubectl get package
NAME                                  PACKAGEMETADATA NAME               VERSION         AGE
cert-manager.tanzu.vmware.com.1.7.2+vmware.3-tkg.1          cert-
manager.tanzu.vmware.com          1.7.2+vmware.3-tkg.1   5s
cert-manager.tanzu.vmware.com.1.7.2+vmware.3-tkg.3          cert-
manager.tanzu.vmware.com          1.7.2+vmware.3-tkg.3   5s
cluster-autoscaler.tanzu.vmware.com.1.25.1+vmware.1-tkg.3       cluster-
autoscaler.tanzu.vmware.com       1.25.1+vmware.1-tkg.3  5s
cluster-autoscaler.tanzu.vmware.com.1.26.2+vmware.1-tkg.3       cluster-
autoscaler.tanzu.vmware.com       1.26.2+vmware.1-tkg.3  5s
cluster-autoscaler.tanzu.vmware.com.1.27.2+vmware.1-tkg.3       cluster-
autoscaler.tanzu.vmware.com       1.27.2+vmware.1-tkg.3  5s
contour.tanzu.vmware.com.1.26.2+vmware.1-tkg.1
contour.tanzu.vmware.com            1.26.2+vmware.1-tkg.1  5s
...
```

Autoscaler 패키지 설치

이제 클러스터 Autoscaler 패키지를 설치할 수 있습니다. 클러스터 Autoscaler는 `kube-system` 네임스페이스에 배포로 설치됩니다.

1 `autoscaler.yaml` 구성 파일을 생성합니다.

- 환경에 적합한 값으로 규격의 `autoscaler-data-values` 섹션을 변경하여 Autoscaler를 사용자 지정할 수 있습니다.

```
#autoscaler.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: autoscaler-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: autoscaler-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: autoscaler-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: autoscaler
  namespace: tkg-system
spec:
  serviceAccountName: autoscaler-sa
  packageRef:
    refName: cluster-autoscaler.tanzu.vmware.com
    versionSelection:
      constraints: 1.27.2+vmware.1-tkg.3
  values:
  - secretRef:
      name: autoscaler-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: autoscaler-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    arguments:
      ignoreDaemonsetsUtilization: true
      maxNodeProvisionTime: 15m
      maxNodesTotal: 0
      metricsPort: 8085
      scaleDownDelayAfterAdd: 10m
      scaleDownDelayAfterDelete: 10s
      scaleDownDelayAfterFailure: 3m
      scaleDownUnneededTime: 10m
    clusterConfig:
      clusterName: "gc1"
      clusterNamespace: "cluster"
    paused: false
```

2 클러스터 Autoscaler 패키지를 설치합니다.

```
kubectl apply -f autoscaler.yaml
```

3 Autoscaler 패키지 설치를 확인합니다.

```
kubectl get pkgi -A | grep autoscaler
```

예상 결과:

```
tkg-system autoscaler cluster-autoscaler.tanzu.vmware.com 1.27.2+vmware.1-tkg.3 Reconcile
succeeded 3m52s
```

4 Autoscaler 배포를 확인합니다.

```
kubectl get pods -n kube-system | grep autoscaler
```

```
cluster-autoscaler-798b65bd9f-bht8n 1/1 Running 0 2m
```

클러스터 자동 크기 조정 테스트

클러스터 자동 크기 조정을 테스트하려면 애플리케이션을 배포하고, 복제본 수를 늘리고, 로드를 처리하기 위해 추가 작업자 노드가 확장되었는지 확인합니다.

클러스터 Autoscaler 테스트의 내용을 참조하십시오.

자동 크기 조정된 클러스터 업그레이드

자동 크기 조정된 클러스터를 업그레이드하려면 Autoscaler 패키지를 일시 중지합니다.

Kubectl을 사용하여 자동 크기 조정된 클러스터 업그레이드의 내용을 참조하십시오.

Tanzu CLI를 사용하여 클러스터 Autoscaler 설치

Tanzu CLI를 사용하여 클러스터 Autoscaler 패키지를 설치하고 구성하려면 다음 지침을 참조하십시오.

요구 사항

다음 요구 사항을 준수합니다.

- 최소 vSphere 버전은 vCenter 및 ESXi 호스트를 포함하여 vSphere 8 U3입니다.
- 최소 TKr 버전은 vSphere 8의 경우 TKr 1.27.x입니다.
- TKr의 부 버전과 클러스터 Autoscaler 패키지의 부 버전이 일치해야 합니다.

참고 Autoscaler 패키지 부 버전과 TKr 부 버전 간에는 일대일 관계가 있습니다. 예를 들어 TKr 1.27.11을 사용하는 경우 Autoscaler v1.27.2를 설치해야 합니다. 버전이 일치하지 않으면 패키지 조정이 실패합니다.

vSphere 네임스페이스 구성

TKG 클러스터 프로비저닝을 위한 다음 사전 요구 사항 작업을 완료합니다.

1 vSphere 8의 경우 vSphere 8 U3 및 TKr 1.27.x로 환경을 설치하거나 업데이트합니다.

2 최신 Tanzu Kubernetes 릴리스로 컨텐츠 라이브러리를 생성하거나 업데이트합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.

3 TKG 클러스터를 호스팅하기 위한 vSphere 네임스페이스를 생성하고 구성합니다. 장 6 TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

4 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

   다음 예는 명령줄에서 도구를 설치하는 데 사용할 수 있습니다. 추가 지침은 vSphere에 대한 Kubernetes CLI 도구 설치 항목을 참조하십시오.

```
wget https://SUPERVISOR-IP-or-FQDN/wcp/plugin/linux-amd64/vsphere-plugin.zip
unzip vsphere-plugin.zip
chmod +x bin/kubectl*
mv bin/kubectl* /usr/bin/kubectl vsphere --help
rm ~/.kube/config
kubectl vsphere login --insecure-skip-tls-verify --server SUPERVISOR-IP-or-FQDN --tanzu-
kubernetes-cluster-namespace VSPHERE-NAMESPACE --vsphere-username VSPHERE-USER
kubectl config use-context VSPHERE-NAMESPACE
```

5 `kubectl` 및 `kubectl vsphere`를 실행하여 설치를 확인합니다.

Autoscaler 주석을 사용하여 TKG 클러스터 생성

지침에 따라 TKG 클러스터를 생성합니다. 추가 지침은 Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로 항목을 참조하십시오.

Autoscaler를 사용하려면 여기에 제공된 클러스터 규격 예에 설명된 대로 Autoscaler 레이블 주석으로 클러스터를 구성해야 합니다. 일반 클러스터 프로비저닝과 달리 작업자 노드 복제본 수를 하드 코딩하지 않습니다. Kubernetes에는 Autoscaler 최소 및 최대 크기 주석을 기반으로 복제본에 대한 기본 논리가 기본 제공됩니다. 이는 새 클러스터이므로 최소 크기는 클러스터를 생성하는 데 사용됩니다. 자세한 내용은 https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling의 내용을 참조하십시오.

1 kubectl을 사용하여 감독자에서 인증합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-
username USERNAME
```

2 클러스터를 호스팅할 대상 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context tkgs-cluster-namespace
```

3 vSphere 네임스페이스에서 사용할 수 있는 VM 클래스를 나열합니다.

대상 vSphere 네임스페이스에 바인딩된 VM 클래스만 사용할 수 있습니다. TKG 서비스 클러스터에서 VM 클래스 사용의 내용을 참조하십시오.

4 사용 가능한 영구 볼륨 스토리지 클래스를 나열합니다.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

이 명령은 스토리지 클래스를 포함하여 vSphere 네임스페이스에 대한 세부 정보를 반환합니다. `kubectl describe storageclasses` 명령도 사용 가능한 스토리지 클래스를 반환하지만 vSphere 관리자 권한이 필요합니다.

5 사용 가능한 Tanzu Kubernetes 릴리스를 나열합니다.

```
kubectl get tkr
```

이 명령은 이 vSphere 네임스페이스에서 사용할 수 있는 TKr과 해당 호환성을 반환합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.

6 수집한 정보를 사용하여 필요한 클러스터 Autoscaler 구성으로 TKG 클러스터 규격 YAML 파일을 만듭니다.

- 작업자 nodePools에 대한 `*-min-size` 및 `*-max-size` 주석을 사용합니다. 이 예에서 3은 최소값이고 5는 확장 가능한 최대 작업자 노드 수입니다. 기본적으로 클러스터는 작업자 노드 3개를 사용하여 생성됩니다.
- TKr 및 Autoscaler 패키지에 대해 일치하는 부 버전을 사용합니다.
- 사용된 클러스터 `metadata.name` 및 `metadata.namespace` 값은 Autoscaler 패키지 기본값과 일치합니다. 클러스터 규격에서 이러한 값을 변경하는 경우 `autoscaler-data-values`에서 수정해야 합니다(아래 참조).

```
#cc-autoscaler.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
 name: tkc
 namespace: cluster
spec:
 clusterNetwork:
   pods:
     cidrBlocks:
     - 192.0.2.0/16
   serviceDomain: cluster.local
   services:
     cidrBlocks:
     - 198.51.100.0/12
 topology:
   class: tanzukubernetescluster
   controlPlane:
     metadata: {}
     replicas: 3
   variables:
```

```
  - name: storageClasses
    value:
    - wcpglobal-storage-profile
  - name: vmClass
    value: guaranteed-medium
  - name: storageClass
    value: wcpglobal-storage-profile
  #minor versions must match
  version: v1.27.11---vmware.1-fips.1-tkg.2
  workers:
    machineDeployments:
    - class: node-pool
      metadata:
        annotations:
          cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size: "3"
          cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size: "5"
      name: np-1
```

7 클러스터 규격을 적용합니다.

```
kubectl apply -f cc-autoscaler.yaml
```

8 클러스터 생성을 확인합니다.

```
kubectl get cluster,vm
```

9 클러스터 노드 버전을 확인합니다.

```
kubectl get node
```

TKG 클러스터에 패키지 저장소를 생성합니다.

TKG 클러스터가 프로비저닝되면 Tanzu CLI를 설치하고 패키지 저장소를 설정합니다.

1 Tanzu CLI를 설치합니다.

TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치의 내용을 참조하십시오.

2 클러스터에 로그인합니다.

```
rm ~/.kube/config
kubectl vsphere login --insecure-skip-tls-verify --server 192.168.0.2 --tanzu-kubernetes-
cluster-namespace autoscaler --vsphere-username administrator@vsphere.local --tanzu-
kubernetes-cluster-name cckubectl
config use-context cc
```

3 패키지 저장소를 생성합니다.

```
#Standard package repository URL might change depending on the required cluster autoscaler
version
tanzu package repository add standard-repo --url projects.registry.vmware.com/tkg/packages/
standard/repo:v2024.4.12 -n tkg-system
tanzu package available list -n tkg-system
tanzu package available get cluster-autoscaler.tanzu.vmware.com -n tkg-system
```

Autoscaler 패키지 설치

클러스터 Autoscaler 패키지를 설치합니다. 클러스터 Autoscaler가 `kube-system` 네임스페이스에 설치됩니다.

1 Tanzu CLI 명령을 사용하여 기본 `values.yaml`을 생성합니다.

```
tanzu package available get cluster-autoscaler.tanzu.vmware.com/1.27.2+vmware.1-tkg.3  -n
tkg-system --default-values-file-output values.yaml
```

2 패키지 설치에 대한 `values.yaml`을 업데이트합니다.

```
arguments:
  ignoreDaemonsetsUtilization: true
  maxNodeProvisionTime: 15m
  maxNodesTotal: 0
  metricsPort: 8085
  scaleDownDelayAfterAdd: 10m
  scaleDownDelayAfterDelete: 10s
  scaleDownDelayAfterFailure: 3m
  scaleDownUnneededTime: 10m
clusterConfig:
  clusterName: "tkc"
  clusterNamespace: "cluster"
paused: false
```

3 Tanzu CLI를 사용하여 클러스터 Autoscaler 패키지를 설치합니다.

```
tanzu package install cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --version 1.27.2+vmware.1-tkg.3 --values-file values.yaml
```

클러스터 자동 크기 조정 테스트

클러스터 자동 크기 조정을 테스트하려면 애플리케이션을 배포하고, 복제본 수를 늘리고, 추가적인 로드를 처리하기 위해 추가 작업자 노드가 확장되었는지 확인합니다.

클러스터 Autoscaler 테스트의 내용을 참조하십시오.

자동 크기 조정된 클러스터 업그레이드

자동 크기 조정된 클러스터를 업그레이드하려면 먼저 Autoscaler 패키지를 일시 중지해야 합니다.

Tanzu CLI를 사용하여 자동 크기 조정된 클러스터 업그레이드의 내용을 참조하십시오.

Kubectl을 사용하여 자동 크기 조정된 클러스터 업그레이드

TKG 클러스터를 업그레이드하기 전에 Autoscaler를 일시 중지해야 합니다. 클러스터의 TKr 버전을 업그레이드한 후 TKr 부 버전과 일치하도록 Autoscaler 패키지 버전을 업데이트해야 합니다.

요구 사항

이 작업에서는 TKG 클러스터에 클러스터 Autoscaler를 설치했다고 가정합니다. Kubectl을 사용하여 클러스터 Autoscaler 설치의 내용을 참조하십시오.

클러스터 업그레이드 전: Autoscaler 일시 중지

Autoscaler가 설치된 TKG 클러스터를 업그레이드하기 전에 먼저 Autoscaler 패키지를 일시 중지해야 합니다.

1 `autoscaler-data-values.yaml` 암호에서 `paused` 부울 값을 `true`로 설정하여 클러스터 Autoscaler 패키지를 일시 중지합니다.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: autoscaler-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    arguments:
      ignoreDaemonsetsUtilization: true
      maxNodeProvisionTime: 15m
      maxNodesTotal: 0
      metricsPort: 8085
      scaleDownDelayAfterAdd: 10m
      scaleDownDelayAfterDelete: 10s
      scaleDownDelayAfterFailure: 3m
      scaleDownUnneededTime: 10m
    clusterConfig:
      clusterName: "gc1"
      clusterNamespace: "cluster"
    paused: true
```

2 `autoscaler-data-values` 암호에 업데이트를 적용합니다.

```
kubectl apply -f autoscaler-data-values.yaml
```

클러스터 업그레이드

Autoscaler가 일시 중지되면 클러스터 업데이트를 계속 진행합니다.

1 TKG 클러스터의 Kubernetes 버전을 업그레이드합니다.

TKR 버전을 편집하여 TKG 클러스터 업데이트의 내용을 참조하십시오.

클러스터 업그레이드 후: Autoscaler 패키지 버전 업데이트

클러스터를 업그레이드한 후 TKr 부 버전과 일치하도록 Autoscaler 패키지 버전을 업데이트하고 일시 중지를 사용하지 않도록 설정합니다.

1 해당하는 Autoscaler 버전을 선택합니다.

TKr 및 Autoscaler 패키지의 부 버전이 일치해야 합니다. 예를 들어 클러스터를 TKr v1.28.8로 업그레이드한 경우 Autoscaler v1.28.x 패키지를 사용해야 합니다.

2 대상 Autoscaler 버전을 설정하고 paused를 false로 재설정하여 Autoscaler 리소스를 업데이트합니다.

```
#autoscaler-package-upgrade.yaml
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: autoscaler
  namespace: tkg-system
spec:
  serviceAccountName: autoscaler-sa
  packageRef:
    refName: cluster-autoscaler.tanzu.vmware.com
    versionSelection:
      constraints: 1.28.0+vmware.1-tkg.1
  values:
  - secretRef:
      name: autoscaler-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: autoscaler-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    arguments:
      ignoreDaemonsetsUtilization: true
      maxNodeProvisionTime: 15m
      maxNodesTotal: 0
      metricsPort: 8085
      scaleDownDelayAfterAdd: 10m
      scaleDownDelayAfterDelete: 10s
      scaleDownDelayAfterFailure: 3m
      scaleDownUnneededTime: 10m
    clusterConfig:
      clusterName: "gc1"
      clusterNamespace: "cluster"
    paused: false
```

3 Autoscaler 패키지에 업데이트를 적용합니다.

```
kubectl apply -f autoscaler-package-upgrade.yaml
```

4 Autoscaler 포드가 kube-system 네임스페이스에서 실행 중인지 확인합니다.

5 클러스터 Autoscaler를 테스트합니다.

클러스터 Autoscaler 테스트.

Tanzu CLI를 사용하여 자동 크기 조정된 클러스터 업그레이드

TKG 클러스터를 업그레이드하기 전에 Autoscaler를 일시 중지해야 합니다. 클러스터의 TKr 버전을 업그레이드한 후 TKr 부 버전과 일치하도록 Autoscaler 패키지 버전을 업데이트해야 합니다.

요구 사항

이 작업에서는 TKG 클러스터에 클러스터 Autoscaler를 설치했다고 가정합니다. Tanzu CLI를 사용하여 클러스터 Autoscaler 설치의 내용을 참조하십시오.

클러스터 업그레이드 전: Autoscaler 일시 중지

Autoscaler가 설치된 TKG 클러스터를 업그레이드하기 전에 먼저 Autoscaler 패키지를 일시 중지해야 합니다.

1 `values.yaml` 구성 파일에서 `paused` 부울 값을 `true`로 설정하여 클러스터 Autoscaler 패키지를 일시 중지합니다.

```
arguments:
  ignoreDaemonsetsUtilization: true
  maxNodeProvisionTime: 15m
  maxNodesTotal: 0
  metricsPort: 8085
  scaleDownDelayAfterAdd: 10m
  scaleDownDelayAfterDelete: 10s
  scaleDownDelayAfterFailure: 3m
  scaleDownUnneededTime: 10m
clusterConfig:
  clusterName: "tkc"
  clusterNamespace: "cluster"
paused: true #set to true before upgrade
```

2 Tanzu CLI를 사용하여 패키지를 업데이트합니다.

```
tanzu package installed update cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --values-file values.yaml
```

클러스터 업그레이드

Autoscaler가 일시 중지되면 클러스터 업데이트를 계속 진행합니다.

1 TKG 클러스터의 Kubernetes 버전을 업그레이드합니다.

TKR 버전을 편집하여 TKG 클러스터 업데이트의 내용을 참조하십시오.

클러스터 업그레이드 후: Autoscaler 패키지 버전 업데이트

클러스터를 업그레이드한 후 TKr 부 버전과 일치하도록 Autoscaler 패키지 버전을 업데이트하고 paused 키를 false로 재설정합니다.

1 해당하는 Autoscaler 버전을 선택합니다.

TKr 및 Autoscaler 패키지의 부 버전이 일치해야 합니다. 예를 들어 클러스터를 TKr v1.28.8로 업그레이드한 경우 Autoscaler v1.28.0 패키지를 사용해야 합니다.

2 Tanzu CLI 명령을 사용하여 기본 `values.yaml`을 생성합니다.

```
tanzu package available get cluster-autoscaler.tanzu.vmware.com/1.28.0+vmware.1-tkg.1  -n
tkg-system --default-values-file-output new-values.yaml
```

3 `new-values.yaml` 파일을 새 패키지 버전으로 업데이트하고 paused를 false로 재설정합니다.

4 Tanzu CLI를 사용하여 클러스터 Autoscaler 설치를 업데이트합니다.

```
tanzu package installed update cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --values-file new-values.yaml --version 1.28.1+vmware.1-tkg.1
```

클러스터 Autoscaler 테스트

설치된 클러스터 Autoscaler를 테스트하려면 다음 지침을 참조하십시오.

요구 사항

이 작업에서는 TKG 클러스터에 클러스터 Autoscaler를 설치했다고 가정합니다.

- Kubectl을 사용하여 클러스터 Autoscaler 설치
- Tanzu CLI를 사용하여 클러스터 Autoscaler 설치

클러스터 Autoscaler 테스트

Autoscaler가 작업자 노드를 자동으로 확장하는지 확인하려면 애플리케이션을 배포한 다음 배포의 복제본 수를 확장합니다. 노드 리소스가 부족하면 Autoscaler가 작업자 노드를 스케일 업합니다.

1 이름이 `app.yaml`인 애플리케이션 정의를 생성합니다.

```
apiVersion: v1
kind: Namespace
```

```
metadata:
  name: app
  labels:
    pod-security.kubernetes.io/enforce: privileged
---
apiVersion: v1
kind: Service
metadata:
  name: application-cpu
  namespace: app
  labels:
    app: application-cpu
spec:
  type: ClusterIP
  selector:
    app: application-cpu
  ports:
    - protocol: TCP
      name: http
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: application-cpu
  namespace: app
  labels:  app: application-cpu
spec:
  selector:
    matchLabels:
      app: application-cpu
    replicas: 1
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxSurge: 1
        maxUnavailable: 0
    template:
      metadata:
        labels:
          app: application-cpu
      spec:
        containers:
        - name: application-cpu
          image: wcp-docker-ci.artifactory.eng.vmware.com/app-cpu:v1.0.0
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          resources:
```

```
            requests:
              memory: 50Mi
              cpu: 500m
            limits:
              memory: 500Mi
              cpu: 2000m
```

2 애플리케이션을 생성합니다.

```
kubectl apply -f app.yaml
```

3 애플리케이션의 복제본을 스케일 업하여 Autoscaler를 트리거합니다.

예를 들어 추가 작업자 노드가 필요하도록 `spec.selector.replicas`의 수를 1에서 더 큰 수로 늘립니다.

4 애플리케이션을 업데이트합니다.

```
kubectl apply -f app.yaml
```

5 로드를 처리하기 위해 추가 작업자 노드가 생성되었는지 확인합니다.

노드 리소스가 부족하면 Autoscaler가 작업자 노드 수를 스케일 업합니다.

클러스터 Autoscaler 삭제

설치된 클러스터 Autoscaler를 삭제하려면 다음 지침을 참조하십시오.

요구 사항

이 작업에서는 TKG 클러스터에 클러스터 Autoscaler를 설치했다고 가정합니다.

- Kubectl을 사용하여 클러스터 Autoscaler 설치
- Tanzu CLI를 사용하여 클러스터 Autoscaler 설치

Kubectl을 사용하여 클러스터 Autoscaler 삭제

1 Kubectl을 사용하여 클러스터 Autoscaler를 삭제하려면 다음 명령을 사용합니다.

```
kubectl delete -f autoscaler.yaml
```

참고 `autoscaler.yaml`이라는 이름은 Autoscaler 패키지를 배포할 때 사용한 이름입니다. 다른 이름을 사용한 경우 적절히 명령을 업데이트하십시오. Kubectl을 사용하여 클러스터 Autoscaler 설치의 내용을 참조하십시오.

Tanzu CLI를 사용하여 클러스터 Autoscaler 삭제

1 Tanzu CLI를 사용하여 클러스터 Autoscaler를 삭제하려면 다음 명령을 사용합니다.

```
tanzu package installed delete -n tkg-system cluster-autoscaler-pkgi
```

TKG 서비스 클러스터에 표준 패키지 설치

11

VMware는 TKG 서비스 클러스터에 설치하여 운영할 수 있는 패키지에 번들로 제공되는 표준 오픈 소스 애플리케이션 집합을 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터용 표준 패키지
- vSphere 8.x용 TKr을 사용하여 TKG 클러스터에 표준 패키지 설치
- 표준 패키지 참조
- vSphere 7.x용 TKr을 사용하여 TKG 클러스터에 표준 패키지 설치

TKG 서비스 클러스터용 표준 패키지

vSphere IaaS control plane은 TKG 서비스 클러스터에 설치하기 위한 표준 패키지를 지원합니다.

지원되는 TKG 서비스 클러스터용 패키지

이 표에는 vSphere 8.x용 TKr로 프로비저닝된 TKG 서비스 클러스터에 설치할 수 있는 표준 패키지가 나열되어 있습니다. 대상 패키지를 설치하기 전에 필요한 사전 요구 사항을 완료하십시오.

패키지	설명	지침
Cert Manager	인증서 관리	Cert Manager 설치
엔보이를 사용하는 Contour	Kubernetes 수신 컨트롤러 및 역방향 프록시	엔보이를 사용하여 Contour 설치
ExternalDNS	Kubernetes 서비스의 DNS 조회	ExternalDNS 설치
Fluent Bit	로그 전달	ExternalDNS 설치
Alertmanager를 사용하는 Prometheus	모니터링 및 경고	Alertmanager를 사용하여 Prometheus 설치
Grafana	시각화	Grafana 설치
Harbor	컨테이너 레지스트리	Harbor 레지스트리 설치
외부 CSI 스냅샷 검증 Webhook vSphere PV CSI Webhook	영구 스토리지 스냅샷 생성	장 15 TKG 서비스 클러스터에서 스냅샷 생성
클러스터 Autoscaler	클러스터 자동 크기 조정	장 10 TKG 서비스 클러스터 자동 스케일링

vSphere 8.x용 TKr을 사용하여 TKG 클러스터에 표준 패키지 설치

vSphere 8.x용 TKr로 프로비저닝된 TKG 서비스 클러스터에 표준 패키지를 설치하려면 이 섹션을 참조하십시오.

일반 요구 사항

TKG 서비스 클러스터에 표준 패키지를 설치하려면 다음과 같은 일반적인 요구 사항을 준수합니다.

플랫폼 요구 사항

이러한 지침은 vSphere 8.x용 TKr로 프로비저닝된 TKG 클러스터에 표준 패키지를 설치하는 방법에만 해당됩니다. 자세한 내용은 TKr 릴리스 정보를 참조하십시오.

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 표준 패키지를 배포하는 경우 vSphere 7.x용 TKr을 사용하여 TKG 클러스터에 표준 패키지 설치의 내용을 참조하십시오.

저장소 요구 사항

vSphere IaaS control plane는 vSphere 8 호환 TKr용 TKG 클러스터에 표준 패키지 설치를 지원합니다. vSphere 8 TKG 서비스 클러스터에서 Kubernetes 릴리스 사용에는 Carvel 패키징 시스템과 Kapp 컨트롤러가 포함되어 있습니다. 두 구성 요소는 TKG 노드의 기반이 되는 TKr 이미지의 일부로 자동으로 관리됩니다. vSphere와 TKr의 호환성은 TKR 릴리스 정보를 참조하십시오.

클라이언트 요구 사항

vSphere 8.x용 TKr로 프로비저닝된 TKG 클러스터에 표준 패키지를 설치하려면 Kubectl 및 kubectl용 vSphere 플러그인 및 Tanzu CLI를 포함한 vSphere에 대한 Kubernetes CLI 도구가 필요합니다. 이러한 도구를 설치하려면 TKG 서비스 클러스터용 CLI 도구 설치의 내용을 참조하십시오.

스토리지 요구 사항

표준 패키지를 배포하는 TKG 클러스터는 기본 스토리지 클래스로 프로비저닝해야 합니다. 특히, Prometheus 및 Grafana 패키지에는 기본 스토리지 클래스가 필요합니다. 기본 스토리지 클래스를 지정하지 않고 TKG 클러스터를 프로비저닝한 경우 기존 스토리지 클래스에 패치를 적용하고 필요한 주석을 추가하여 기본값으로 지정할 수 있습니다. 스토리지 클래스에 패치 적용의 내용을 참조하십시오.

Tanzu 패키지를 설치하는 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에 대한 스토리지 제한은 총 영구 볼륨 할당 크기보다 커야 합니다. vSphere 네임스페이스 스토리지 할당량에 대한 자세한 내용은 TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성 항목을 참조하십시오.

표 11-1. 표준 패키지에 대한 영구 스토리지 요구 사항

구성 요소	TKG 확장	기본 스토리지 크기
Grafana	Grafana	8Gi
Prometheus 서버	Prometheus	8Gi

표 11-1. 표준 패키지에 대한 영구 스토리지 요구 사항 (계속)

구성 요소	TKG 확장	기본 스토리지 크기
Alertmanager	Prometheus	8Gi
Harbor	Harbor 레지스트리	PVC에 따라 다름

TKG 클러스터가 프로비저닝된 vSphere 네임스페이스에 대한 스토리지 제한을 조정하려면 다음을 수행합니다.

1 vSphere Client를 사용하여 vSphere IaaS control plane를 사용하도록 설정된 vCenter Server에 로그인합니다.

2 대상 Tanzu Kubernetes 클러스터가 프로비저닝된 vSphere 네임스페이스를 선택합니다.

3 **구성 > 리소스 제한**을 선택합니다.

4 **편집**을 클릭합니다.

5 스토리지 제한이 Prometheus 및 Grafana 확장에 필요한 영구 볼륨 할당의 총 크기보다 크도록 **스토리지** 제한을 조정합니다.

패키지 저장소 생성

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 표준 패키지 저장소를 설정합니다.

요구 사항

패키지 저장소를 생성하기 전에 다음 요구 사항을 준수하십시오.

- 일반 요구 사항
- vSphere에 대한 Kubernetes CLI 도구 설치
- TKG 서비스 클러스터에서 사용할 Tanzu CLI 설치
- vSphere 8.x용 TKr을 사용하여 TKG 클러스터를 프로비저닝합니다. Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로 및 TKR 릴리스 정보를 참조하십시오.

Carvel imgpkg 설치

Carvel imgpkg(https://carvel.dev/imgpkg/) 도구를 사용하면 표준 패키지 저장소의 사용 가능한 버전을 찾아볼 수 있습니다. 저장소는 공용이므로 로그인할 필요가 없습니다.

Carvel imgpkg를 설치하려면 다음 단계를 완료합니다.

1 다음 명령을 사용하여 imgpkg를 설치합니다.

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

2 설치를 확인합니다.

```
imgpkg version
```

샘플 결과:

```
imgpkg version 0.42.1
```

3 다음 명령을 실행하여 저장소 버전을 나열합니다.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

이 명령은 사용 가능한 표준 패키지 저장소 버전을 반환합니다.

```
Tags

Name
...
v2024.4.12
v2024.4.19
v2024.5.14
v2024.5.16

39 tags

Succeeded
```

패키지 저장소 생성

TKG 클러스터에 로그인하고 패키지 저장소를 생성합니다.

1 클러스터에 로그인합니다.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local --tanzu-kubernetes-cluster-name CLUSTER --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

2 패키지 저장소를 생성합니다.

```
tanzu package repository add standard-repo --url projects.registry.vmware.com/tkg/packages/standard/repo:v2024.5.16 -n tkg-system
```

참고 대상 저장소 버전과 일치하도록 저장소 버전 문자열을 변경합니다.

3 사용 가능한 패키지를 나열합니다.

```
tanzu package available list -n tkg-system
```

참고 저장소의 일부 패키지는 TKG 클러스터에서 지원되지 않습니다. 지원되는 패키지의 공식 목록을 참조하십시오. TKG 서비스 클러스터용 표준 패키지.

4 개별 패키지에 사용 가능한 버전을 나열합니다.

Cert Manager

```
tanzu package available get cert-manager.tanzu.vmware.com -n tkg-system
```

Contour

```
tanzu package available get contour.tanzu.vmware.com -n tkg-system
```

외부 DNS

```
tanzu package available get external-dns.tanzu.vmware.com -n tkg-system
```

Fluent Bit

```
tanzu package available get fluent-bit.tanzu.vmware.com -n tkg-system
```

Grafana

```
tanzu package available get grafana.tanzu.vmware.com -n tkg-system
```

Prometheus

```
tanzu package available get prometheus.tanzu.vmware.com -n tkg-system
```

Cert Manager 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 Cert Manager를 설치합니다.

Cert Manager 정보

Cert Manager는 TKG 서비스 클러스터에 대한 인증서 관리를 제공합니다. Cert Manager는 Contour, ExternalDNS, Prometheus 및 Harbor를 포함한 대부분의 표준 패키지에 대한 사전 요구 사항입니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항.\
- 패키지 저장소 생성

Cert Manager 설치

Cert Manager를 설치하려면 다음 단계를 완료합니다.

1 사용 가능한 Cert Manager 버전을 나열합니다.

```
tanzu package available get cert-manager.tanzu.vmware.com -n tkg-system
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 Cert Manager 네임스페이스를 생성합니다.

```
kubectl create ns cert-manager
```

3 Cert Manager를 설치합니다.

대상 버전을 조정하여 요구 사항을 충족합니다.

```
tanzu package install cert-manager -p cert-manager.tanzu.vmware.com -n cert-manager -v
1.12.2+vmware.2-tkg.2
```

4 Cert Manager의 설치를 확인합니다.

```
tanzu package installed list -n cert-manager
```

```
tanzu package installed get -n cert-manager cert-manager
```

5 패키지 설치로 생성된 리소스에 대한 Cert Manager 네임스페이스를 확인합니다.

```
kubectl -n cert-manager get all
```

```
NAME                                           READY   STATUS    RESTARTS   AGE
pod/cert-manager-b5675b75f-flkjp               1/1     Running   0          6m14s
pod/cert-manager-cainjector-f8dc756cf-f7xsv    1/1     Running   0          6m14s
pod/cert-manager-webhook-6c888c8ddd-5xlnb      1/1     Running   0          6m14s

NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/cert-manager           ClusterIP   10.97.254.59     <none>        9402/TCP   6m14s
service/cert-manager-webhook   ClusterIP   10.105.225.156   <none>        443/TCP    6m14s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cert-manager               1/1     1            1           6m14s
deployment.apps/cert-manager-cainjector    1/1     1            1           6m14s
deployment.apps/cert-manager-webhook       1/1     1            1           6m14s

NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/cert-manager-b5675b75f               1         1         1       6m14s
replicaset.apps/cert-manager-cainjector-f8dc756cf    1         1         1       6m14s
replicaset.apps/cert-manager-webhook-6c888c8ddd      1         1         1       6m14s
```

문제 해결

다음 명령을 사용하여 오류 메시지를 확인합니다.

```
kubectl get pkgi -A
```

```
kubectl describe pkgi -n cert-manager cert-manage
```

엔보이를 사용하여 Contour 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 엔보이를 사용하여 Contour를 설치합니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항
- Contour 패키지 참조
- 패키지 저장소 생성
- Cert Manager 설치

Contour 데이터 값 생성

데이터 값 파일을 생성하여 Contour 설치를 준비합니다.

1 사용 가능한 Contour 패키지 버전을 나열합니다.

```
tanzu package available get contour.tanzu.vmware.com -n tkg-system
```

또는 kubectl 사용:

```
kubectl -n tkg-system get packages | grep contour
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 `contour-default-values.yaml` 파일을 생성합니다.

```
tanzu package available get contour.tanzu.vmware.com/1.28.2+vmware.1-tkg.1 --default-values-file-output contour-data-values.yaml
```

형식 설명:

- *1.28.2+vmware.1-tkg.1*은 대상 패키지 버전입니다.
- *contour-data-values.yaml*은 생성할 데이터 값 파일의 이름과 경로입니다.

3 `contour-data-values.yaml` 파일을 편집합니다.

클러스터 외부의 트래픽이 Kubernetes 서비스에 액세스할 수 있도록 엔보이 서비스를 `LoadBalancer`로 설정합니다. 지침은 다음 예를 참조하십시오.

```
vi contour-data-values.yaml
```

```
---
infrastructure_provider: vsphere
namespace: tanzu-system-ingress
contour:
 configFileContents: {}
 useProxyProtocol: false
 replicas: 2
 pspNames: "vmware-system-restricted"
 logLevel: info
envoy:
```

```
 service:
   type: LoadBalancer
   annotations: {}
   externalTrafficPolicy: Cluster
   disableWait: false
 hostPorts:
   enable: true
   http: 80
   https: 443
 hostNetwork: false
 terminationGracePeriodSeconds: 300
 logLevel: info
certificates:
 duration: 8760h
 renewBefore: 360h
```

Contour 설치

엔보이를 사용하여 Contour 수신을 설치하려면 다음 단계를 완료합니다.

1 Contour 패키지에 대한 고유한 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-ingress
```

2 Contour를 설치합니다.

버전을 조정하여 요구 사항을 충족합니다.

```
tanzu package install contour -p contour.tanzu.vmware.com -v 1.28.2+vmware.1-tkg.1 --
values-file contour-data-values.yaml -n tanzu-system-ingress
```

3 Contour 설치를 확인합니다.

```
tanzu package installed list -n tanzu-system-ingress
```

4 Contour 및 엔보이 개체를 확인합니다.

```
kubectl -n tanzu-system-ingress get all
```

```
NAME                            READY   STATUS    RESTARTS   AGE
pod/contour-777bdddc69-fqnsp    1/1     Running   0          102s
pod/contour-777bdddc69-gs5xv    1/1     Running   0          102s
pod/envoy-d4jtt                 2/2     Running   0          102s
pod/envoy-g5h72                 2/2     Running   0          102s
pod/envoy-pjpzc                 2/2     Running   0          102s

NAME              TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                      AGE
service/contour   ClusterIP      10.105.242.46   <none>
8001/TCP                     102s
service/envoy     LoadBalancer   10.103.245.57   10.197.154.69   80:32642/
TCP,443:30297/TCP   102s
```

```
NAME                    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/envoy   3         3         3       3            3
<none>          102s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/contour   2/2     2            2           102s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/contour-777bdddc69   2         2         2       102s
```

이 예에서, 엔보이 서비스의 외부 IP 주소는 10.197.154.69입니다. 이 IP 주소는 **워크로드 네트워크 > 수신**에 지정된 CIDR 범위에서 분할됩니다. 이 IP 주소에 대해 새 로드 밸런서 인스턴스가 생성됩니다. 이 로드 밸런서에 대한 서버 풀의 멤버는 엔보이 포드입니다. 엔보이 포드는 이러한 포드가 실행되는 작업자 노드의 IP 주소를 가정하기 때문에 클러스터 노드를 쿼리(`kubectl get nodes -o wide`)하여 이러한 IP 주소를 볼 수 있습니다.

문제 해결

필요에 따라 다음 항목을 참조하십시오.

- Contour 패키지 참조.

ExternalDNS 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 ExternalDNS를 설치합니다.

ExternalDNS 정보

ExternalDNS를 사용하면 엔보이를 사용하는 Contour와 같은 수신 구성 요소가 있는 Kubernetes 서비스에 대해 DNS 레코드를 자동으로 생성할 수 있습니다. ExternalDNS 패키지는 AWS Route 53, Azure DNS 및 RFC2136 준수 DNS 서버(예: BIND) 같은 DNS 제공자를 통해 검증됩니다. ExternalDNS 패키지 참조 항목도 참조하십시오.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항.
- 패키지 저장소 생성.
- Cert Manager 설치.
- 엔보이를 사용하여 Contour 설치.

ExternalDNS 데이터 값 생성

ExternalDNS 데이터 값 파일을 생성하여 ExternalDNS 설치를 준비합니다.

1 저장소에서 사용 가능한 ExternalDNS 패키지 버전을 나열합니다.

```
tanzu package available get external-dns.tanzu.vmware.com -n tkg-system
```

또는 kubectl 사용.

```
kubectl -n tkg-system get packages | grep external-dns
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 ExternalDNS 패키지에 대한 데이터 값 파일을 생성합니다.

```
tanzu package available get external-dns.tanzu.vmware.com/0.13.6+vmware.1-tkg.1 --default-values-file-output external-dns-data-values.yaml
```

형식 설명:

- *0.13.6+vmware.1-tkg.1*은 대상 패키지 버전입니다.
- *external-dns-data-values.yaml*은 생성할 데이터 값 파일의 이름과 경로입니다.

3 환경에 맞게 데이터 값을 사용자 지정합니다.

데이터 값은 대상이 되는 지원되는 DNS 서버에 따라 다릅니다. 예제는 ExternalDNS 패키지 참조 항목을 참조하십시오.

4 필요한 경우 ExternalDNS 패키지가 상호 작용할 DNS 서버를 정의하는 configmap을 생성합니다.

예제는 ExternalDNS 패키지 참조 항목을 참조하십시오.

ExternalDNS 설치

TKG 클러스터에 ExternalDNS 패키지를 설치하려면 다음 단계를 완료합니다.

1 ExternalDNS에 대한 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-service-discovery
```

2 Tanzu CLI를 사용하여 ExternalDNS 패키지를 설치합니다.

```
tanzu package install external-dns -p external-dns.tanzu.vmware.com -n tanzu-system-service-discovery -v 0.11.0+vmware.1-tkg.2 --values-file external-dns-data-values.yaml
```

3 Tanzu CLI를 사용하여 패키지가 설치되어 있는지 확인합니다.

```
tanzu package installed list -n tanzu-system-service-discovery

NAME           PACKAGE-NAME                    PACKAGE-VERSION         STATUS
external-dns   external-dns.tanzu.vmware.com   0.11.0+vmware.1-tkg.2   Reconcile succeeded
```

```
kubectl -n tanzu-system-service-discovery get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/external-dns-77d947745-tcjz9     1/1     Running   0          63s

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/external-dns     1/1     1            1           63s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/external-dns-77d947745      1         1         1       63s
```

참조

필요에 따라 다음 항목을 참조하십시오.

- ExternalDNS 패키지 참조.
- 서비스 검색을 위한 ExternalDNS 설치.

Fluent Bit 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 Fluent Bit를 설치합니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항
- 패키지 저장소 생성
- Cert Manager 설치
- Fluent Bit 로그 전달의 지원되는 대상입니다. Fluent Bit 패키지 참조의 내용을 참조하십시오.

Fluent Bit 데이터 값 생성

데이터 값 파일을 생성하여 Fluent Bit 설치를 준비합니다.

1 사용 가능한 Contour 패키지 버전을 나열합니다.

```
tanzu package available get fluent-bit.tanzu.vmware.com -n tkg-system
```

또는 kubectl 사용:

```
kubectl -n tkg-system get packages | grep fluent-bit
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 `fluent-bit-data-values.yaml` 파일을 생성합니다.

```
tanzu package available get fluent-bit.tanzu.vmware.com/2.1.6+vmware.1-tkg.2 --default-values-file-output fluent-bit-data-values.yaml
```

3 `fluent-bit-data-values.yaml` 파일을 편집하고 값을 구성합니다.

사용 가능한 모든 매개 변수의 예와 목록은 Fluent Bit 패키지 참조 항목을 참조하십시오.

Fluent Bit 설치

Fluent Bit 패키지를 설치하려면 다음 단계를 완료하십시오.

1 Fluent Bit에 대한 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-logging
```

2 Fluent Bit를 설치합니다.

```
tanzu package install fluent-bit -p fluent-bit.tanzu.vmware.com -v 2.1.6+vmware.1-tkg.2 --values-file fluent-bit-data-values.yaml -n tanzu-system-logging
```

3 Fluent Bit 설치를 확인합니다.

```
tanzu package installed list -n tanzu-system-logging
```

```
tanzu package installed get fluent-bit -n tanzu-system-logging
```

4 Fluent Bit 개체를 확인합니다.

```
kubectl -n tanzu-system-logging get all
```

Alertmanager를 사용하여 Prometheus 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 Alertmanager를 사용하여 Prometheus를 설치합니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항
- 패키지 저장소 생성.
- Cert Manager 설치.

- 엔보이를 사용하여 Contour 설치(Prometheus 대시보드에 액세스하는 데 필요)
- Prometheus 패키지 참조

Prometheus 데이터 값 생성

데이터 값 파일을 생성하여 Prometheus 설치를 준비합니다.

1 저장소에 대한 최신 Prometheus 패키지 버전을 가져옵니다.

```
tanzu package available get prometheus.tanzu.vmware.com -n tkg-system
```

또는 kubectl 사용.

```
kubectl -n tkg-system get packages | grep prometheus
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 `prometheus-data-values.yaml` 파일을 생성합니다.

```
tanzu package available get prometheus.tanzu.vmware.com/2.45.0+vmware.1-tkg.2 --default-values-file-output prometheus-data-values.yaml
```

형식 설명:

- *2.45.0+vmware.1-tkg.2*는 대상 패키지 버전입니다.
- *prometheus-data-values.yaml*은 생성할 데이터 값 파일의 이름과 경로입니다.

3 `prometheus-data-values.yaml` 파일을 편집하고 Prometheus 대시보드에 액세스하는 데 필요한 다음 값을 구성합니다. 예제 데이터 값 파일 및 전체 구성 매개 변수 목록은 Prometheus 패키지 참조 항목을 참조하십시오.

매개 변수	설명
`ingress.tlsCertificate.tls.crt`	수신을 위해 자체 서명된 TLS 인증서가 생성됩니다. 필요한 경우 자체적으로 재정의하고 제공할 수 있습니다.
`ingress.tlsCertificate.tls.key`	수신을 위해 자체 서명된 TLS 개인 키가 생성됩니다. 필요한 경우 자체적으로 재정의하고 제공할 수 있습니다.
`ingress.enabled`	값을 `true`로 설정합니다(기본값은 false).
`ingress.virtual_host_fqdn`	값을 `prometheus.<your.domain>`으로 설정합니다(기본값은 `prometheus.system.tanzu`).
`alertmanager.pvc.storageClassName`	vSphere 스토리지 정책의 이름을 입력합니다.
`prometheus.pvc.storageClassName`	vSphere 스토리지 정책의 이름을 입력합니다.

Prometheus 설치

다음 단계를 완료하여 Prometheus 패키지를 설치합니다.

1 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-monitoring
```

2 Prometheus를 설치합니다.

```
tanzu package install prometheus -p prometheus.tanzu.vmware.com -v 2.45.0+vmware.1-tkg.2 --
values-file prometheus-data-values.yaml -n tanzu-system-monitoring
```

3 Prometheus 설치를 확인합니다.

```
tanzu package installed list -n tanzu-system-monitoring
```

```
tanzu package installed get prometheus -n tanzu-system-monitoring
```

4 Prometheus 및 Altermanager 개체를 확인합니다.

```
kubectl -n tanzu-system-monitoring get all
```

```
kubectl -n tanzu-system-monitoring get pvc

NAME                STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
alertmanager        Bound    pvc-a53f7091-9823-4b70-a9b4-c3d7a1e27a4b   2Gi
RWO            k8s-policy     2m30s
prometheus-server   Bound    pvc-41745d1d-9401-41d7-b44d-ba430ecc5cda   20Gi
RWO            k8s-policy     2m30s
```

Prometheus 설치 문제 해결

`tanzu package install prometheus` 작업에서 "최종 보급 주소를 가져오지 못함: 개인 IP 주소를 찾을 수 없으며 명시적 IP가 제공되지 않았습니다." 오류를 반환하는 경우 패키지 오버레이를 적용하여 Alertmanager 구성 요소를 재구성합니다.

1 `overlay-alertmanager.yaml` 파일을 생성합니다.

```
---
#@ load("@ytt:overlay", "overlay")

#@overlay/match by=overlay.and_op(overlay.subset({"kind": "Deployment"}),
overlay.subset({"metadata": {"name": "alertmanager"}}))
---
spec:
  template:
    spec:
      containers:
```

```
        #@overlay/match by="name",expects="0+"
        - name: alertmanager
          args:
            - --cluster.listen-address=
```

2 Kubectl을 사용하여 `overlay-alertmanager.yaml` 파일에서 암호를 생성합니다.

```
kubectl create secret generic alertmanager-overlay -n tkg-system -o yaml --dry-run=client
--from-file=overlay-alertmanager.yaml | kubectl apply -f -
```

3 Kubectl을 사용하여 오버레이 암호로 Prometheus 패키지에 주석을 추가합니다.

```
kubectl annotate PackageInstall prometheus -n tkg-system ext.packaging.carvel.dev/ytt-
paths-from-secret-name.1=alertmanager-overlay
```

4 설치 명령을 다시 실행합니다.

```
tanzu package install prometheus -p prometheus.tanzu.vmware.com -v 2.37.0+vmware.3-tkg.1 --
values-file prometheus-data-values.yaml -n tanzu-system-monitoring
```

Prometheus 대시보드 액세스

Prometheus가 설치되면 다음 단계를 완료하여 Prometheus 대시보드에 액세스합니다.

1 `prometheus-data-values.yaml` 파일의 `ingress` 섹션이 모든 필수 필드로 채워져 있는지 확인합니다.

```
ingress:
  enabled: true
  virtual_host_fqdn: "prometheus.system.tanzu"
  prometheus_prefix: "/"
  alertmanager_prefix: "/alertmanager/"
  prometheusServicePort: 80
  alertmanagerServicePort: 80
  #! [Optional] The certificate for the ingress if you want to use your own TLS
certificate.
  #! We will issue the certificate by cert-manager when it's empty.
  tlsCertificate:
    #! [Required] the certificate
    tls.crt:
    #! [Required] the private key
    tls.key:
    #! [Optional] the CA certificate
    ca.crt:
```

2 엔보이 로드 밸런서를 사용하는 Contour의 공용(외부) IP 주소를 가져옵니다.

   엔보이를 사용하여 Contour 설치의 내용을 참조하십시오.

3 사용한 Prometheus FQDN(기본값은 `prometheus.system.tanzu`)을 엔보이 로드 밸런서의 IP 주소에 매핑하는 DNS 레코드를 생성합니다.

4 브라우저를 사용하여 Prometheus FQDN으로 이동하여 Prometheus 대시보드에 액세스합니다.

Grafana 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 Grafana를 설치합니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항.
- 패키지 저장소 생성
- Cert Manager 설치
- 엔보이를 사용하여 Contour 설치
- Alertmanager를 사용하여 Prometheus 설치
- Grafana 패키지 참조

Grafana 데이터 값 생성

데이터 값 파일을 생성하여 Grafana 설치를 준비합니다.

1 저장소에 대한 최신 Prometheus 패키지 버전을 가져옵니다.

```
tanzu package available get grafana.tanzu.vmware.com -n tkg-system
```

또는 kubectl 사용.

```
kubectl -n tkg-system get packages | grep grafana
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 `prometheus-data-values.yaml` 파일을 생성합니다.

```
tanzu package available get grafana.tanzu.vmware.com/10.0.1+vmware.1-tkg.2 --default-
values-file-output grafana-data-values.yaml
```

형식 설명:

- *10.0.1+vmware.1-tkg.2*는 대상 패키지 버전입니다.
- *grafana-data-values.yaml*은 생성할 데이터 값 파일의 이름과 경로입니다.

3 `gafana-data-values.yaml` 파일을 편집하고 값을 업데이트합니다.

TKG 클러스터에서 액세스할 수 있는 vSphere 스토리지 클래스의 이름인 `ingress.pvc: storageClassName` 및 해당 값을 추가합니다.

일반적인 오류를 방지하려면 데이터 값 파일에서 암호를 제거하고 암호를 수동으로 생성합니다. Grafana 설치 문제 해결의 내용을 참조하십시오.

다음은 스토리지 클래스 필드가 추가되고 암호가 제거된 최소 `gafana-data-values.yaml`입니다. 추가 예제 및 전체 매개 변수 목록은 Grafana 패키지 참조 항목을 참조하십시오.

```
grafana:
  deployment:
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
  service:
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
  pvc:
    storageClassName: vSphere-storage-profile
namespace: grafana
```

Grafana 설치

Grafana 패키지를 설치하려면 다음 단계를 완료하십시오.

1 Grafana에 대한 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-dashboards
```

2 Grafana 패키지를 설치합니다.

```
tanzu package install grafana -p grafana.tanzu.vmware.com -v 10.0.1+vmware.1-tkg.2 --
values-file grafana-data-values.yaml -n tanzu-system-dashboards
```

3 Grafana 설치를 확인합니다.

```
tanzu package installed list -n tanzu-system-dashboards
```

```
tanzu package installed get grafana -n tanzu-system-dashboards
```

4 Grafana 개체를 확인합니다.

```
kubectl -n tanzu-system-dashboards get all
```

5 Grafana에서 보유한 영구 볼륨 할당을 확인합니다.

```
kubectl -n tanzu-system-dashboards get pvc
```

Grafana 설치 문제 해결

"기본 YAML 파일에서 Grafana를 설치할 때 암호가 생성되지 않음" 오류를 방지하려면 *grafana-data-values.yaml*에서 `grafana.secret.*`을 제거하고 다음과 같이 수동으로 암호를 생성합니다. 그런 다음 Grafana 패키지를 다시 배포합니다.

```
kubectl create secret generic grafana -n tanzu-system-dashboards  --from-literal=admin=admin
```

Harbor 레지스트리 설치

다음 지침에 따라 vSphere 8.x용 TKr을 실행하는 TKG 서비스 클러스터에 Harbor 컨테이너 레지스트리를 설치합니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- 일반 요구 사항
- 패키지 저장소 생성
- Cert Manager 설치
- 엔보이를 사용하여 Contour 설치
- Harbor 패키지 참조

Harbor 데이터 값 생성

데이터 값 파일을 생성하여 Harbor 설치를 준비합니다.

1 저장소에 대한 최신 Harbor 패키지 버전을 가져옵니다.

```
tanzu package available get harbor.tanzu.vmware.com -n tkg-system
```

또는 kubectl 사용.

```
kubectl -n tkg-system get packages | grep harbor
```

참고 일반적으로 요구 사항이 다른 경우가 아니면 최신 버전을 사용해야 합니다.

2 `harbor-data-values.yaml` 파일을 생성합니다.

```
tanzu package available get harbor.tanzu.vmware.com/2.9.1+vmware.1-tkg.1 --default-values-file-output harbor-data-values.yaml
```

형식 설명:

- *2.9.1+vmware.1-tkg.1*은 대상 패키지 버전입니다.
- *harbor-data-values.yaml*은 생성할 데이터 값 파일의 이름과 경로입니다.

3 `harbor-data-values.yaml` 파일을 편집하고 다음 매개 변수의 값을 업데이트합니다.

필요에 따라 추가 매개 변수를 구성합니다. Harbor 패키지 참조의 내용을 참조하십시오.

필드	설명
`hostname`	Harbor 관리 콘솔 및 레지스트리 서비스에 액세스하기 위한 FQDN입니다. "yourdomain.com"를 고유한 호스트 이름으로 바꿉 있습니다.
`harborAdminPassword`	암호를 강력하고 고유한 암호로 변경합니다(설치 후 UI에서 변경할 수도 있음).
`persistence.persistentVolumeClaim.database.storageClass:`	vSphere 네임스페이스에 대한 vSphere 스토리지 정책의 이름을 입력합니다.
`persistence.persistentVolumeClaim.jobservice.storageClass:`	vSphere 네임스페이스에 대한 vSphere 스토리지 정책의 이름을 입력합니다.
`persistence.persistentVolumeClaim.redis.storageClass:`	vSphere 네임스페이스에 대한 vSphere 스토리지 정책의 이름을 입력합니다.
`persistence.persistentVolumeClaim.registry.storageClass:`	vSphere 네임스페이스에 대한 vSphere 스토리지 정책의 이름을 입력합니다.
`persistence.persistentVolumeClaim.trivy.storageClass:`	vSphere 네임스페이스에 대한 vSphere 스토리지 정책의 이름을 입력합니다.
`tlsCertificate.tlsSecretLabels:`	{"managed-by": "vmware-vRegistry"}

Harbor 설치

다음 단계를 완료하여 Harbor 레지스트리를 설치합니다.

1 Harbor에 대한 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-registry
```

2 Harbor를 설치합니다.

```
tanzu package install harbor --package harbor.tanzu.vmware.com --version 2.9.1+vmware.1-tkg.1 --values-file harbor-data-values.yaml --namespace tanzu-system-registry
```

3 Harbor 설치를 확인합니다.

```
tanzu package installed get harbor --namespace tanzu-system-registry
```

LoadBalancer 유형의 엔보이 서비스를 사용하여 Harbor용 DNS 구성

엔보이 서비스를 사용하는 사전 요구 사항 Contour가 LoadBalancer를 통해 노출되는 경우 로드 밸런서의 외부 IP 주소를 가져오고 Harbor FQDN에 대한 DNS 레코드를 생성합니다.

1 LoadBalancer 유형의 엔보이 서비스에 대한 `External-IP` 주소를 가져옵니다.

```
kubectl get service envoy -n tanzu-system-ingress
```

반환된 `External-IP` 주소가 다음과 같이 표시됩니다.

```
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
envoy   LoadBalancer   10.99.25.220   10.195.141.17   80:30437/TCP,443:30589/TCP   3h27m
```

또는 다음 명령을 사용하여 `External-IP` 주소를 가져올 수 있습니다.

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Harbor 확장 설치를 확인하려면 로드 밸런서의 `External-IP` 주소에 매핑된 Harbor 및 Notary FQDN으로 로컬 `/etc/hosts` 파일을 업데이트합니다. 예를 들면 다음과 같습니다.

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy Load Balancer IP
10.195.141.17 core.harbor.domain
10.195.141.17 core.notary.harbor.domain
```

3 Harbor 확장 설치를 확인하려면 Harbor에 로그인합니다.

4 엔보이 서비스 로드 밸런서 `External-IP` 주소를 Harbor FQDN 및 Notary FQDN에 매핑하는 2개의 CNAME 레코드를 DNS 서버에 생성합니다.

5 외부 DNS 확장을 설치합니다.

NodePort 유형의 엔보이 서비스를 사용하여 Harbor용 DNS 구성

엔보이 서비스를 사용하는 사전 요구 사항 Contour가 NodePort를 통해 노출되는 경우 작업자 노드의 가상 시스템 IP 주소를 가져오고 Harbor FQDN에 대한 DNS 레코드를 생성합니다.

참고 NodePort를 사용하려면 `harbor-data-values.yaml` 파일에 올바른 `port.https` 값을 지정해야 합니다.

1 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

2 클러스터의 노드를 나열합니다.

```
kubectl get virtualmachines
```

3 작업자 노드 중 하나를 선택하고 다음 명령을 사용하여 설명합니다.

```
kubectl describe virtualmachines tkg2-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 가상 시스템의 IP 주소(예: `Vm Ip: 10.115.22.43`)를 찾습니다.

5 Harbor 확장 설치를 확인하려면 작업자 노드 IP 주소에 매핑된 Harbor 및 Notary FQDN으로 로컬 `/etc/hosts` 파일을 업데이트합니다. 예를 들면 다음과 같습니다.

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy NodePort
10.115.22.43 core.harbor.domain
10.115.22.43 core.notary.harbor.domain
```

6 Harbor 확장 설치를 확인하려면 Harbor에 로그인합니다.

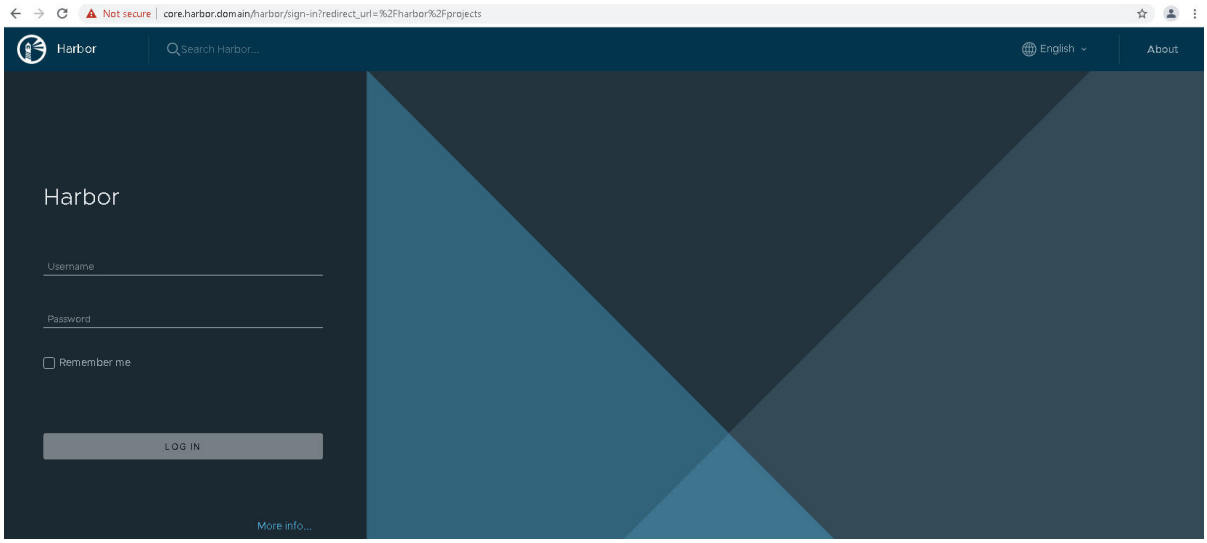
7 작업자 노드 IP 주소를 Harbor FQDN 및 Notary FQDN에 매핑하는 2개의 CNAME 레코드를 DNS 서버에 생성합니다.

8 외부 DNS 확장을 설치합니다.

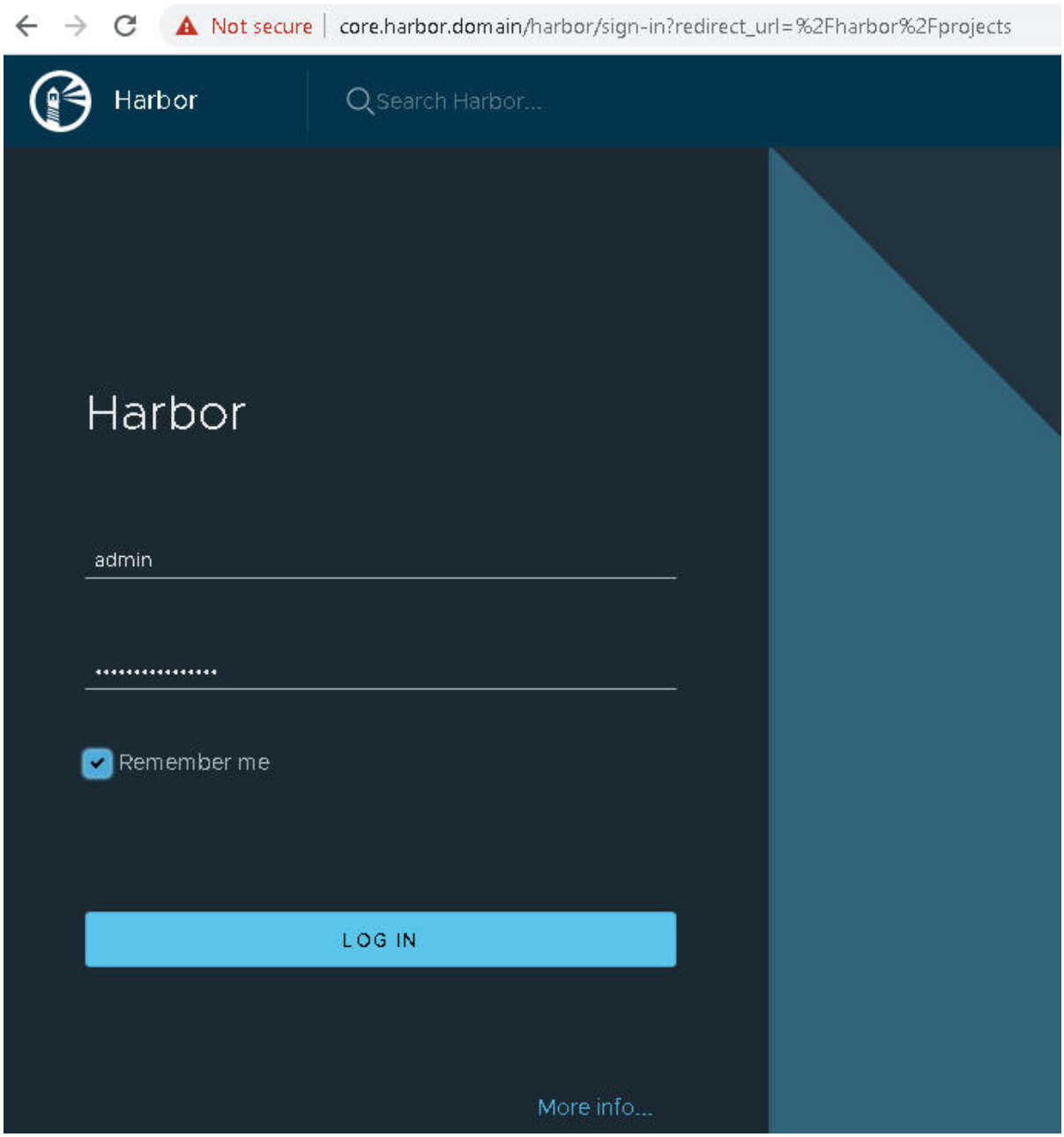
Harbor 웹 인터페이스에 로그인

Harbor가 설치 및 구성되면 로그인하여 사용을 시작합니다.

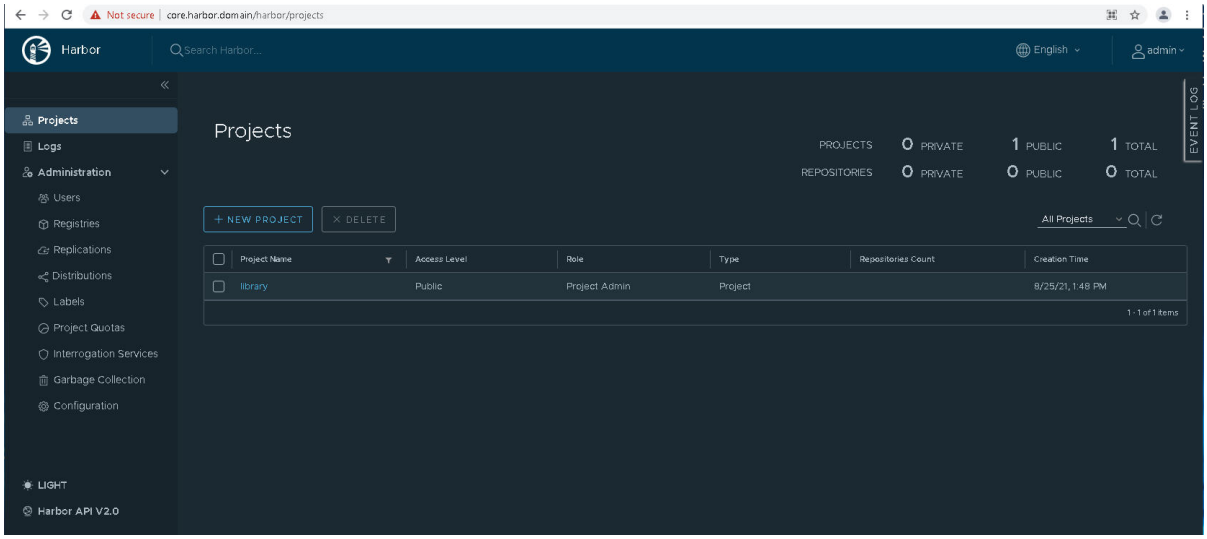
1 Harbor 레지스트리 웹 인터페이스(`https://core.harbor.domain`) 또는 사용한 호스트 이름에 액세스합니다.



2 `harbor-data-values.yaml` 파일에 입력한 사용자 이름 **`admin`**과 생성된 암호를 사용하여 Harbor에 로그인합니다.



3 Harbor 사용자 인터페이스에 액세스할 수 있는지 확인합니다.



4 Harbor CA 인증서를 가져옵니다.

Harbor 인터페이스에서 **프로젝트 > 라이브러리**를 선택하거나 **새 프로젝트**를 생성합니다.

레지스트리 인증서를 클릭하고 Harbor CA 인증서(`ca.crt`)를 다운로드합니다.

5 Harbor 레지스트리에서 컨테이너 이미지를 푸시하고 끌어올 수 있도록 Harbor CA 인증서를 Docker 클라이언트의 신뢰 저장소에 추가합니다. 장 14 TKG 서비스 클러스터에서 개인 레지스트리 사용의 내용을 참조하십시오.

6 Harbor 사용에 대한 자세한 내용은 Harbor 설명서를 참조하십시오.

표준 패키지 참조

이 섹션에서는 TKG 서비스 클러스터에 설치할 수 있는 표준 패키지에 대한 참조 정보를 제공합니다.

Contour 패키지 참조

이 항목에서는 엔보이를 사용하는 Contour 패키지에 대한 참조 정보를 제공합니다.

Contour 및 엔보이 정보

Contour(https://projectcontour.io/)는 엔보이 역방향 HTTP 프록시를 포함하는 Kubernetes 수신 컨트롤러입니다. 엔보이를 사용하는 Contour는 일반적으로 ExternalDNS, Prometheus 및 Harbor와 같은 다른 패키지와 함께 사용됩니다.

TKG 클러스터에 Contour 패키지를 설치하려면 다음 항목을 참조하십시오.

- 엔보이를 사용하여 Contour 설치
- #unique_173

Contour 구성 요소

Contour 패키지에는 Contour 수신 컨트롤러와 엔보이 역방향 HTTP 프록시가 포함됩니다. 이러한 구성 요소는 컨테이너로 설치됩니다. 컨테이너는 패키지 저장소에 지정된 공용 레지스트리에서 가져옵니다.

컨테이너	리소스 유형	복제	설명
엔보이	DaemonSet	3	고성능 역방향 프록시
Contour	배포	2	엔보이용 관리 및 구성 서버

Contour 데이터 값

다음은 `contour-data-values.yaml` 예시입니다.

유일한 사용자 지정은 엔보이 서비스가 LoadBalancer 유형(기본값은 NodePort)이라는 것입니다. 즉, 수신을 위해 클러스터 외부에서 Envoy 서비스에 액세스할 수 있습니다.

```
infrastructure_provider: vsphere
namespace: tanzu-system-ingress
contour:
 configFileContents: {}
 useProxyProtocol: false
 replicas: 2
 pspNames: "vmware-system-restricted"
 logLevel: info
envoy:
 service:
   type: LoadBalancer
   annotations: {}
   nodePorts:
     http: null
     https: null
   externalTrafficPolicy: Cluster
   disableWait: false
 hostPorts:
   enable: true
   http: 80
   https: 443
 hostNetwork: false
 terminationGracePeriodSeconds: 300
 logLevel: info
 pspNames: null
certificates:
 duration: 8760h
 renewBefore: 360h
```

Contour 구성

Contour 패키지 구성 값은 `contour-data-values.yaml`에 설정됩니다. 다음 표에는 사용 가능한 매개 변수와 그에 대한 설명이 나와 있습니다.

표 11-2. Contour 수신 구성 매개 변수

매개 변수	설명	유형	기본값
infrastructure_provider	인프라 제공자. 지원되는 값: vsphere, aws, azure	문자열	필수 매개 변수
contour.namespace	Contour가 배포될 네임스페이스	문자열	tanzu-system-ingress
contour.config.requestTimeout	엔보이에 전달할 클라이언트 요청 시간 초과	time.Duration	0s (자세한 내용은 아래 섹션 참조)
contour.config.server.xdsServerType	사용할 XDS 서버 유형: 지원되는 값: contour 또는 envoy	문자열	Null
contour.config.tls.minimumProtocolVersion	Contour가 협상할 최소 TLS 버전	문자열	1.1
contour.config.tls.fallbackCertificate.name	vhost에 대해 정의된 SNI와 일치하지 않는 요청에 대한 폴백 인증서가 포함된 암호의 이름	문자열	Null
contour.config.tls.fallbackCertificate.namespace	폴백 인증서가 포함된 암호의 네임스페이스	문자열	Null
contour.config.tls.envoyClientCertificate.name	클라이언트 인증서로 사용할 암호의 이름, 백엔드 서비스에 대한 TLS 연결용 개인 키	문자열	Null
contour.config.tls.envoyClientCertificate.namespace	클라이언트 인증서로 사용할 암호의 네임스페이스, 백엔드 서비스에 대한 TLS 연결용 개인 키	문자열	Null
contour.config.leaderelection.configmapName	contour leaderelection에 사용할 configmap의 이름	문자열	leader-elect
contour.config.leaderelection.configmapNamespace	contour leaderelection configmap의 네임스페이스	문자열	tanzu-system-ingress
contour.config.disablePermitInsecure	ingressroute permitInsecure 필드를 사용하지 않도록 설정	부울	false
contour.config.accesslogFormat	액세스 로그 형식	문자열	envoy
contour.config.jsonFields	로깅될 필드	문자열 어레이	envoy package doc
contour.config.useProxyProtocol	https://projectcontour.io/guides/proxy-proto/	부울	false
contour.config.defaultHTTPVersions	Contour가 제공할 엔보이를 프로그래밍해야 하는 HTTP 버전	문자열 어레이	"HTTP/1.1 HTTP2"
contour.config.timeouts.requestTimeout	전체 요청에 대한 시간 제한	time.Duration	Null(시간 제한을 사용하지 않도록 설정함)
contour.config.timeouts.connectionIdleTimeout	유휴 연결을 종료하기 전에 대기할 시간	time.Duration	60s

표 11-2. Contour 수신 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
contour.config.timeouts.streamIdleTimeout	활동이 없는 요청 또는 스트림을 종료하기 전에 대기할 시간	time.Duration	5m
contour.config.timeouts.maxConnectionDuration	활동 여부에 관계없이 연결을 종료하기 전에 대기할 시간	time.Duration	Null(시간 제한을 사용하지 않도록 설정함)
contour.config.timeouts.ConnectionShutdownGracePeriod	초기 및 최종 GOAWAY 전송 사이의 대기 시간	time.Duration	5s
contour.config.cluster.dnsLookupFamily	HTTPProxy 경로에서 externalName 유형 서비스에 대한 업스트림 요청에 사용할 dns-lookup-family	문자열	Null(지원되는 값: auto, v4, v6)
contour.config.debug	Contour 디버깅 켜기	부울	false
contour.config.ingressStatusAddress	모든 수신 리소스의 상태에 설정할 주소	문자열	Null
contour.certificate.duration	Contour 인증서 기간	time.Duration	8760h
contour.certificate.renewBefore	Contour 인증서를 갱신하기 전까지의 기간	time.Duration	360h
contour.deployment.replicas	Contour 복제본 수	정수	2
contour.image.repository	Contour 이미지가 있는 저장소의 위치입니다. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg
contour.image.name	Contour 이미지의 이름	문자열	contour
contour.image.tag	Contour 이미지 태그. Contour 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v1.11.0_vmware.1
contour.image.pullPolicy	Contour 이미지 Pull 정책	문자열	IfNotPresent
envoy.image.repository	엔보이 이미지가 있는 저장소의 위치입니다. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg
envoy.image.name	엔보이 이미지의 이름	문자열	envoy

표 11-2. Contour 수신 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
envoy.image.tag	엔보이 이미지 태그. 엔보이 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	`v1.17.3_vmware.1`
envoy.image.pullPolicy	엔보이 이미지 Pull 정책	문자열	IfNotPresent
envoy.hostPort.enable	호스트에서 엔보이 포트를 노출하는 플래그	부울	true
envoy.hostPort.http	엔보이 HTTP 호스트 포트	정수	80
envoy.hostPort.https	엔보이 HTTPS 호스트 포트	정수	443
envoy.service.type	엔보이를 노출할 서비스 유형. 지원되는 값: ClusterIP, NodePort, LoadBalancer	문자열	vSphere에 대한 필수 매개 변수: NodePort 또는 LoadBalancer, AWS: LoadBalancer, Azure: LoadBalancer
envoy.service.annotations	엔보이 서비스 주석	맵(키-값)	빈 맵
envoy.service.externalTrafficPolicy	엔보이 서비스의 외부 트래픽 정책. 지원되는 값: Local, Cluster	문자열	클러스터
envoy.service.nodePort.http	http 요청에 사용되는 NodePort 유형의 서비스에 필요한 nodePort	정수	Null - Kubernetes가 동적 노드 포트를 할당함
envoy.service.nodePort.https	HTTPS 요청에 사용되는 NodePort 유형의 서비스에 필요한 nodePort	정수	Null - Kubernetes가 동적 노드 포트를 할당함
envoy.deployment.hostNetwork	hostNetwork에서 엔보이 실행	부울	false
envoy.service.aws.LBType	엔보이 서비스를 노출하는 데 사용할 AWS LB 유형. 지원되는 값: classic, nlb	문자열	classic
envoy.loglevel	엔보이에 사용할 로그 수준	문자열	정보

파일 다운로드에 대한 경로 시간 초과

`contour.config.requestTimeout` 매개 변수는Contour 경로 시간 초과 기간을 정의합니다. 기본값은 `0s`입니다. 파일 전송에 Contour를 사용하는 경우 이 값을 조정해야 할 수 있습니다.

Contour 설명서에 따르면 시간 초과 값이 `0s`이면 Contour가 엔보이 시간 초과를 사용하도록 지시합니다. 엔보이 설명서에 따르면 엔보이의 기본 시간 초과 값은 15초입니다. 또한 엔보이는 전체 요청-응답 작업이 시간 초과 간격 내에 완료될 것으로 예상합니다.

따라서 기본 Contour 시간 초과 설정이 `0s`인 경우 파일 전송이 15초 내에 완료되어야 합니다. 대용량 파일 전송에는 이 시간이 충분하지 않을 수 있습니다. 기본 엔보이 시간 초과를 사용하지 않도록 설정하려면 `contour.config.requestTimeout` 값을 `0`으로 설정합니다.

ExternalDNS 패키지 참조

이 항목에서는 ExternalDNS 패키지에 대한 참조 정보를 제공합니다.

ExternalDNS 정보

ExternalDNS는 노출된 Kubernetes 서비스 및 수신을 DNS 제공자와 동기화합니다.

TKG 클러스터에 ExternalDNS를 설치하려면 다음 항목을 참조하십시오.

- vSphere 8x용 TKr: ExternalDNS 설치
- vSphere 7.x용 TKr: ExternalDNS 설치

ExternalDNS 구성 요소

ExternalDNS 패키지는 표에 나열된 컨테이너를 설치합니다. 패키지는 패키지 저장소에 지정된 공용 레지스트리에서 컨테이너를 가져옵니다.

컨테이너	리소스 유형	복제	설명
ExternalDNS	DaemonSet	6	DNS 조회를 위한 Kubernetes 서비스 노출

ExternalDNS 데이터 값

ExternalDNS 데이터 값 파일은 ExternalDNS 구성 요소와 지원되는 DNS 제공자 간에 상호 작용하는 데 사용됩니다. ExternalDNS 패키지는 AWS(Route 53), Azure DNS 및 RFC2136 준수 DNS 서버(예: BIND)와 같은 DNS 제공자를 통해 검증됩니다.

다음 예는 RFC2136 준수 DNS 제공자(예: BIND)에 사용할 수 있습니다.

```
---
# Namespace in which to deploy ExternalDNS pods
namespace: tanzu-system-service-discovery
# Deployment-related configuration
deployment:
args:
   - --registry=txt
   - --txt-owner-id=k8s
   - --txt-prefix=external-dns- #! Disambiguates TXT records from CNAME records
   - --provider=rfc2136
   - --rfc2136-host=IP-ADDRESS #! Replace with IP of RFC2136-compatible DNS server, such as
192.168.0.1
   - --rfc2136-port=53
   - --rfc2136-zone=DNS-ZONE #! Replace with zone where services are deployed, such as my-
zone.example.org
   - --rfc2136-tsig-secret=TSIG-SECRET #! Replace with TSIG key secret authorized to update
DNS server
   - --rfc2136-tsig-secret-alg=hmac-sha256
```

```
   - --rfc2136-tsig-keyname=TSIG-KEY-NAME #! Replace with TSIG key name, such as externaldns-
key
   - --rfc2136-tsig-axfr
   - --source=service
   - --source=ingress
   - --source=contour-httpproxy #! Enables Contour HTTPProxy object support
   - --domain-filter=DOMAIN #! Zone where services are deployed, such as my-zone.example.org
```

다음 예는 AWS DNS 제공자(경로 53)에 사용할 수 있습니다.

```
---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! read Contour HTTPProxy resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --provider=aws
        - --policy=upsert-only #! prevent deleting any records, omit to enable full
synchronization
        - --aws-zone-type=public #! only look at public hosted zones (public, private, no
value for both)
        - --aws-prefer-cname
        - --registry=txt
        - --txt-owner-id=HOSTED_ZONE_ID #! Route53 hosted zone identifier for my-
zone.example.org
        - --txt-prefix=txt #! disambiguates TXT records from CNAME records
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_access_key_id
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_secret_access_key
```

다음 예는 Azure DNS 제공자에 사용할 수 있습니다.

```
---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --provider=azure
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! read Contour HTTPProxy resources
```

```
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --azure-resource-group=my-resource-group #! Azure resource group
        volumeMounts:
        - name: azure-config-file
          mountPath: /etc/kubernetes
          readOnly: true
        #@overlay/replace
        volumes:
        - name: azure-config-file
          secret:
            secretName: azure-config-file
```

ExternalDNS 구성

다음 표에는 ExternalDNS에 대해 사용 가능한 구성 매개 변수와 그에 대한 설명이 나와 있습니다. 추가 지침은 사이트 https://github.com/kubernetes-sigs/external-dns#running-externaldns에서 참조하십시오.

표 11-3. 외부 DNS 패키지 구성

매개 변수	설명	유형	기본값
externalDns.namespace	external-dns가 배포될 네임스페이스	문자열	tanzu-system-service-discovery
externalDns.image.repository	external-dns 이미지가 포함된 저장소	문자열	projects.registry.vmware.com/tkg
externalDns.image.name	external-dns의 이름	문자열	external-dns
externalDns.image.tag	ExternalDNS 이미지 태그	문자열	v0.7.4_vmware.1
externalDns.image.pullPolicy	ExternalDNS 이미지 Pull 정책	문자열	IfNotPresent
externalDns.deployment.annotations	external-dns 배포에 대한 주석	map<string,string>	{}
externalDns.deployment.args	명령줄을 통해 external-dns로 전달된 인수	list<string>	[](필수 매개 변수)
externalDns.deployment.env	external-dns에 전달할 환경 변수	list<string>	[]
externalDns.deployment.securityContext	external-dns 컨테이너의 보안 컨텍스트	보안 컨텍스트	{}
externalDns.deployment.volumeMounts	external-dns 컨테이너의 볼륨 마운트	list<VolumeMount>	[]
externalDns.deployment.volumes	external-dns 포드의 볼륨	list<Volume>	[]

예제 configmap

다음 예제 configmap은 ExternalDNS가 와 상호 연결할 수 있는 Kerberos 구성을 정의합니다. 사용자 지정 항목에는 도메인/영역 이름과 kdc/admin_server 주소가 포함됩니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: krb.conf
  namespace: tanzu-system-service-discovery
data:
  krb5.conf: |
    [logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

    [libdefaults]
    dns_lookup_realm = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
    rdns = false
    pkinit_anchors = /etc/pki/tls/certs/ca-bundle.crt
    default_ccache_name = KEYRING:persistent:%{uid}

    default_realm = CORP.ACME

    [realms]
    CORP.ACME = {
      kdc = controlcenter.corp.acme
      admin_server = controlcenter.corp.acme
    }

    [domain_realm]
    corp.acme = CORP.ACME
    .corp.acme = CORP.ACME
```

Fluent Bit 패키지 참조

이 항목에서는 Fluent Bit 패키지에 대한 참조 정보를 제공합니다.

Fluent Bit 정보

Fluent Bit(https://fluentbit.io/)는 빠르고 가벼운 로그 프로세서 및 전달자이며, 다양한 소스에서 애플리케이션 데이터와 로그를 수집하고 통합하여 여러 대상으로 보내는 데 사용할 수 있습니다.

Fluent Bit를 TKG 클러스터의 로그에 대한 로그 전달자로 사용할 수 있습니다. 로그 저장 및 분석을 위해 로깅 관리 서버를 배포해야 합니다. 지원되는 로깅 서버에는 Syslog, HTTP, Elastic Search, Kafka 및 Splunk가 포함됩니다.

TKG 클러스터에 Fluent Bit 패키지를 설치하려면 다음 항목을 참조하십시오.

- vSphere 8.x TKr: Fluent Bit 설치
- vSphere 7.x TKr: #unique_175

Fluent Bit 구성 요소

Fluent Bit 패키지는 표에 나열된 컨테이너를 클러스터에 설치합니다. 패키지는 패키지 저장소에 지정된 공용 레지스트리에서 컨테이너를 가져옵니다.

컨테이너	리소스 유형	복제	설명
Fluent Bit	DaemonSet	6	로그 수집기, 집계기, 전달자

Fluent Bit 데이터 값

다음 `fluent-bit-data-values.yaml` 예는 Syslog 서버에 사용할 수 있습니다.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "syslog"
  syslog:
    host: "<SYSLOG_HOST>"
    port: "<SYSLOG_PORT>"
    mode: "<SYSLOG_MODE>"
    format: "<SYSLOG_FORMAT>"
```

다음 `fluent-bit-data-values.yaml` 예는 HTTP 끝점에 사용할 수 있습니다.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "http"
  http:
    host: "<HTTP_HOST>"
    port: "<HTTP_PORT>"
    uri: "<URI>"
    header_key_value: "<HEADER_KEY_VALUE>"
    format: "json"
```

다음 `fluent-bit-data-values.yaml` 예는 Elastic Search에 사용할 수 있습니다.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
```

```
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "elasticsearch"
  elasticsearch:
    host: "<ELASTIC_SEARCH_HOST>"
    port: "<ELASTIC_SEARCH_PORT>"
```

다음 `fluent-bit-data-values.yaml` 예는 Kafka에 사용할 수 있습니다.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "kafka"
  kafka:
    broker_service_name: "<BROKER_SERVICE_NAME>"
    topic_name: "<TOPIC_NAME>"
```

다음 `fluent-bit-data-values.yaml` 예는 Splunk에 사용할 수 있습니다.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "splunk"
  splunk:
    host: "<SPLUNK_HOST>"
    port: "<SPLUNK_PORT>"
    token: "<SPLUNK_TOKEN>"
```

Fluent Bit 구성

구성 값은 `fluent-bit-data-values.yaml`에 설정되어 있습니다. 다음 표에는 사용 가능한 매개 변수와 그에 대한 설명이 나와 있습니다.

표 11-4. Fluent Bit 패키지 구성

매개 변수	설명	유형	기본값
logging.namespace	Fluent Bit가 배포될 네임스페이스	문자열	tanzu-system-logging
logging.service_account_name	Fluent Bit 서비스 계정의 이름	문자열	fluent-bit
logging.cluster_role_name	Fluent Bit에 대한 get, watch 및 list 권한을 부여하는 클러스터 역할의 이름	문자열	fluent-bit-read
logging.image.name	Fluent Bit 이미지의 이름	문자열	fluent-bit

표 11-4. Fluent Bit 패키지 구성 (계속)

매개 변수	설명	유형	기본값
logging.image.tag	Fluent Bit 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v1.6.9_vmware.1
logging.image.repository	Fluent Bit 이미지가 있는 저장소의 위치입니다. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg
logging.image.pullPolicy	Fluent Bit 이미지 Pull 정책	문자열	IfNotPresent
logging.update_strategy	DaemonSet 업데이트 시 사용할 업데이트 전략	문자열	RollingUpdate
tkg.cluster_name	Tanzu Kubernetes 클러스터의 이름	문자열	Null(필수 매개 변수)
tkg.instance_name	하나의 배포에서 모든 Tanzu Kubernetes 클러스터 및 감독자 클러스터가 공유하는 TKG 인스턴스의 사용자 정의 이름. 설치와 관련된 모든 이름을 사용할 수 있습니다.	문자열	Null(필수 매개 변수) **참고** 이 필드는 필수이지만 임의의 필드입니다. 로그에 나타나는 이름입니다.
fluent_bit.log_level	Fluent Bit에 사용할 로그 수준	문자열	정보
fluent_bit.output_plugin	Fluent-Bit가 수집한 정보를 플러시해야 하는 백엔드 설정	문자열	Null(필수 매개 변수)
fluent_bit.elasticsearch.host	대상 Elasticsearch 인스턴스의 IP 주소 또는 호스트 이름	문자열	Null(output_plugin이 elasticsearch인 경우 매개 변수)
fluent_bit.elasticsearch.port	대상 Elasticsearch 인스턴스의 TCP 포트	정수	Null(output_plugin이 elasticsearch인 경우 매개 변수)
fluent_bit.elasticsearch.buffer_size	Elasticsearch 서비스의 응답을 읽는 데 사용되는 버퍼 크기를 지정. False인 경우 제한 없음으로 설정	문자열	False
fluent_bit.elasticsearch.tls	Elasticsearch에 대한 TLS의 기본 설정 지정	문자열	꺼짐
fluent_bit.kafka.broker_service_name	Kafka 브로커의 여러 목록 중 하나(예: 192.168.1.3:9092)	문자열	Null(output_plugin이 kafka인 경우 필수 매개 변수)
fluent_bit.kafka.topic_name	Fluent Bit가 Kafka에 메시지를 보내는 데 사용할 단일 항목 또는 (,)로 구분된 항목의 목록	문자열	Null(output_plugin이 kafka인 경우 필수 매개 변수)
fluent_bit.splunk.host	대상 Splunk 서버의 IP 주소 또는 호스트 이름	문자열	Null(output_plugin이 splunk인 경우 필수 매개 변수)

표 11-4. Fluent Bit 패키지 구성 (계속)

매개 변수	설명	유형	기본값
fluent_bit.splunk.port	대상 Splunk 서버의 TCP 포트	정수	Null(output_plugin이 splunk인 경우 필수 매개 변수)
fluent_bit.splunk.token	HTTP 이벤트 수집기 인터페이스에 대한 인증 토큰 지정	문자열	Null(output_plugin이 splunk인 경우 필수 매개 변수)
fluent_bit.http.host	대상 HTTP 서버의 IP 주소 또는 호스트 이름	문자열	Null(output_plugin이 http인 경우 필수 매개 변수)
fluent_bit.http.port	대상 HTTP 서버의 TCP 포트	정수	Null(output_plugin이 http인 경우 필수 매개 변수)
fluent_bit.http.mode	대상 웹 서버에 대한 HTTP URI 지정	문자열	Null(output_plugin이 http인 경우 필수 매개 변수)
fluent_bit.http.header_key_value	HTTP 헤더 키/값 쌍. 여러 헤더를 설정할 수 있음	문자열	Null(output_plugin이 http인 경우 필수 매개 변수)
fluent_bit.http.format	HTTP 요청 본문에 사용할 데이터 형식 지정	문자열	Null(output_plugin이 http인 경우 필수 매개 변수)
fluent_bit.syslog.host	원격 Syslog 서버의 도메인 또는 IP 주소	문자열	Null(output_plugin이 syslog인 경우 필수 매개 변수)
fluent_bit.syslog.port	원격 Syslog 서버의 TCP 또는 UDP 포트	정수	Null(output_plugin이 syslog인 경우 필수 매개 변수)
fluent_bit.syslog.mode	TCP, UDP 및 TLS 중에 전송 유형 지정	문자열	Null(output_plugin이 syslog인 경우 필수 매개 변수)
fluent_bit.syslog.format	HTTP 요청 본문에 사용할 데이터 형식 지정	문자열	Null(output_plugin이 syslog인 경우 필수 매개 변수)
host_path.volume_1	볼륨 1에 대한, 호스트 노드의 파일 시스템에서 포드로의 디렉토리 경로	문자열	/var/log
host_path.volume_2	볼륨 2에 대한, 호스트 노드의 파일 시스템에서 포드로의 디렉토리 경로	문자열	/var/lib/docker/containers
host_path.volume_3	볼륨 3에 대한, 호스트 노드의 파일 시스템에서 포드로의 디렉토리 경로	문자열	/run/log
systemd.path	Systemd 저널 디렉토리의 경로	문자열	/var/log/journal

Prometheus 패키지 참조

이 항목에서는 Prometheus 패키지에 대한 참조 정보를 제공합니다.

Prometheus 및 Altertmanager 정보

Prometheus(https://prometheus.io/)는 시스템 및 서비스 모니터링 시스템입니다. Prometheus는 일정 간격으로 구성된 대상에서 메트릭을 수집하고, 규칙 표현식을 평가하고, 결과를 표시합니다. Alertmanager는 일부 조건이 true로 확인되었을 때 경고를 트리거하는 데 사용됩니다.

Prometheus 패키지를 설치하려면 다음을 수행합니다.

-
-

Prometheus 구성 요소

Prometheus 패키지는 표에 나열된 컨테이너를 TKG 클러스터에 설치합니다. 패키지는 패키지 저장소에 지정된 VMware 공용 레지스트리에서 컨테이너를 가져옵니다.

컨테이너	리소스 유형	복제	설명
`prometheus-alertmanager`	배포	1	Prometheus 서버와 같은 클라이언트 애플리케이션에서 보낸 경고를 처리합니다.
`prometheus-cadvisor`	DaemonSet	5	실행 중인 컨테이너의 리소스 사용량 및 성능 데이터를 분석하고 노출합니다.
`prometheus-kube-state-metrics`	배포	1	노드 상태 및 용량, 복제 세트 규정 준수, 포드, 작업 및 cronjob 상태, 리소스 요청 및 제한을 모니터링합니다.
`prometheus-node-exporter`	DaemonSet	5	커널에 의해 노출되는 하드웨어 및 OS 메트릭용 내보내기입니다.
`prometheus-pushgateway`	배포	1	스크레이핑할 수 없는 작업에서 메트릭을 푸시할 수 있는 서비스입니다.
`prometheus-server`	배포	1	스크레이핑, 규칙 처리 및 경고를 비롯한 핵심 기능을 제공합니다.

Prometheus 데이터 값

다음은 예시 `prometheus-data-values.yaml` 파일입니다.

다음에 유의하십시오.

- 수신이 가능하도록 설정됨(ingress: enabled: true).
- /alertmanager/(alertmanager_prefix:) 및 /(prometheus_prefix:)로 끝나는 URL에 대해 수신이 구성됨.
- Prometheus의 FQDN은 `prometheus.system.tanzu`임(virtual_host_fqdn:).
- 수신 섹션(tls.crt, tls.key, ca.crt)에서 사용자 지정 인증서를 제공합니다.
- alertmanager에 대한 pvc는 2GiB입니다. 기본 스토리지 정책에 대한 `storageClassName`을 제공합니다.

- prometheus에 대한 pvc는 20GiB입니다. vSphere 스토리지 정책에 대한 `storageClassName`을 제공합니다.

```
namespace: prometheus-monitoring
alertmanager:
  config:
    alertmanager_yml: |
      global: {}
      receivers:
      - name: default-receiver
      templates:
      - '/etc/alertmanager/templates/*.tmpl'
      route:
        group_interval: 5m
        group_wait: 10s
        receiver: default-receiver
        repeat_interval: 3h
  deployment:
    replicas: 1
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: default
  service:
    port: 80
    targetPort: 9093
    type: ClusterIP
ingress:
  alertmanager_prefix: /alertmanager/
  alertmanagerServicePort: 80
  enabled: true
  prometheus_prefix: /
  prometheusServicePort: 80
  tlsCertificate:
    ca.crt: |
      -----BEGIN CERTIFICATE-----
      MIIFczCCA1ugAwIBAgIQTYJITQ3SZ4BBS9UzXfJIuTANBgkqhkiG9w0BAQsFADBM
      ...
      w0oGuTTBfxSMKs767N3G1q5tz0mwFpIqIQtXUSmaJ+9p7IkpWcThLnyYYo1IpWm/
      ZHtjzZMQVA==
      -----END CERTIFICATE-----
    tls.crt: |
      -----BEGIN CERTIFICATE-----
      MIIHxTCCBa2gAwIBAgITIgAAAAQnSpH7QfxTKAAAAAAAABDANBgkqhkiG9w0BAQsF
      ...
      YYsIjp7/f+Pk1DjzWx8JIAbzItKLucDreAmmDXqk+DrBP9LYqtmjB0n7nSErgK8G
      sA3kGCJdOkI0kgF10gsinaouG2jVlwNOsw==
      -----END CERTIFICATE-----
    tls.key: |
      -----BEGIN PRIVATE KEY-----
```

```
      MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQDOGHT8I12KyQGS
      ...
      l1NzswracGQIzo03zk/X3Z6P2YOea4BkZ0Iwh34wOHJnTkfEeSx6y+oSFMcFRthT
      yfFCZUk/sVCc/C1a4VigczXftUGiRrTR
      -----END PRIVATE KEY-----
  virtual_host_fqdn: prometheus.system.tanzu
kube_state_metrics:
  deployment:
    replicas: 1
  service:
    port: 80
    targetPort: 8080
    telemetryPort: 81
    telemetryTargetPort: 8081
    type: ClusterIP
node_exporter:
  daemonset:
    hostNetwork: false
    updatestrategy: RollingUpdate
  service:
    port: 9100
    targetPort: 9100
    type: ClusterIP
prometheus:
  pspNames: "vmware-system-restricted"
  config:
    alerting_rules_yml: |
      {}
    alerts_yml: |
      {}
    prometheus_yml: |
      global:
        evaluation_interval: 1m
        scrape_interval: 1m
        scrape_timeout: 10s
      rule_files:
      - /etc/config/alerting_rules.yml
      - /etc/config/recording_rules.yml
      - /etc/config/alerts
      - /etc/config/rules
      scrape_configs:
      - job_name: 'prometheus'
        scrape_interval: 5s
        static_configs:
        - targets: ['localhost:9090']
      - job_name: 'kube-state-metrics'
        static_configs:
        - targets: ['prometheus-kube-state-metrics.prometheus.svc.cluster.local:8080']

      - job_name: 'node-exporter'
        static_configs:
        - targets: ['prometheus-node-exporter.prometheus.svc.cluster.local:9100']

      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
```

```
      - role: pod
      relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
    - job_name: kubernetes-nodes-cadvisor
      kubernetes_sd_configs:
      - role: node
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
      - replacement: kubernetes.default.svc:443
        target_label: __address__
      - regex: (.+)
        replacement: /api/v1/nodes/$1/proxy/metrics/cadvisor
        source_labels:
        - __meta_kubernetes_node_name
        target_label: __metrics_path__
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
    - job_name: kubernetes-apiservers
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
      - action: keep
        regex: default;kubernetes;https
        source_labels:
        - __meta_kubernetes_namespace
        - __meta_kubernetes_service_name
        - __meta_kubernetes_endpoint_port_name
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
```

```
      alerting:
        alertmanagers:
        - scheme: http
          static_configs:
          - targets:
            - alertmanager.prometheus.svc:80
        - kubernetes_sd_configs:
            - role: pod
          relabel_configs:
          - source_labels: [__meta_kubernetes_namespace]
            regex: default
            action: keep
          - source_labels: [__meta_kubernetes_pod_label_app]
            regex: prometheus
            action: keep
          - source_labels: [__meta_kubernetes_pod_label_component]
            regex: alertmanager
            action: keep
          - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_probe]
            regex: .*
            action: keep
          - source_labels: [__meta_kubernetes_pod_container_port_number]
            regex:
            action: drop
    recording_rules_yml: |
      groups:
        - name: kube-apiserver.rules
          interval: 3m
          rules:
          - expr: |2
              (
                (
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[1d]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[1d]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[1d]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[1d]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[1d]))
              )
```

```
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[1d]))
            labels:
              verb: read
            record: apiserver_request:burnrate1d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[1h]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[1h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[1h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[1h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[1h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[1h]))
            labels:
              verb: read
            record: apiserver_request:burnrate1h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[2h]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[2h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
```

```
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[2h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[2h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[2h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[2h]))
            labels:
              verb: read
            record: apiserver_request:burnrate2h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[30m]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[30m]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[30m]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[30m]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[30m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[30m]))
            labels:
              verb: read
            record: apiserver_request:burnrate30m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
```

```
apiservers",verb=~"LIST|GET"}[3d]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[3d]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[3d]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[3d]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[3d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[3d]))
            labels:
              verb: read
            record: apiserver_request:burnrate3d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[5m]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[5m]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[5m]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[5m]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[5m]))
              )
```

```
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[5m]))
            labels:
              verb: read
            record: apiserver_request:burnrate5m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[6h]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[6h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[6h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[6h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[6h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[6h]))
            labels:
              verb: read
            record: apiserver_request:burnrate6h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[1d]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[1d]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[1d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
```

```
PATCH|DELETE"}[1d]))
            labels:
              verb: write
            record: apiserver_request:burnrate1d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[1h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[1h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[1h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[1h]))
            labels:
              verb: write
            record: apiserver_request:burnrate1h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[2h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[2h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[2h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[2h]))
            labels:
              verb: write
            record: apiserver_request:burnrate2h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[30m]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[30m]))
                )
                +
```

```
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[30m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[30m]))
            labels:
              verb: write
            record: apiserver_request:burnrate30m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[3d]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[3d]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[3d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[3d]))
            labels:
              verb: write
            record: apiserver_request:burnrate3d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[5m]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[5m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[5m]))
            labels:
              verb: write
            record: apiserver_request:burnrate5m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[6h]))
```

```
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[6h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[6h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[6h]))
            labels:
              verb: write
            record: apiserver_request:burnrate6h
          - expr: |
              sum by (code,resource) (rate(apiserver_request_total{job="kubernetes-
apiservers",verb=~"LIST|GET"}[5m]))
            labels:
              verb: read
            record: code_resource:apiserver_request_total:rate5m
          - expr: |
              sum by (code,resource) (rate(apiserver_request_total{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))
            labels:
              verb: write
            record: code_resource:apiserver_request_total:rate5m
          - expr: |
              histogram_quantile(0.99, sum by (le, resource)
(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-apiservers",verb=~"LIST|GET"}
[5m]))) > 0
            labels:
              quantile: "0.99"
              verb: read
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |
              histogram_quantile(0.99, sum by (le, resource)
(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[5m]))) > 0
            labels:
              quantile: "0.99"
              verb: write
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |2
              sum(rate(apiserver_request_duration_seconds_sum{subresource!="log",verb!~"LIST|
WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod)
              /
              sum(rate(apiserver_request_duration_seconds_count{subresource!="log",verb!
~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod)
            record: cluster:apiserver_request_duration_seconds:mean5m
          - expr: |
              histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-apiservers",subresource!
="log",verb!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance,
pod))
            labels:
```

```
              quantile: "0.99"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |
              histogram_quantile(0.9,
sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-apiservers",subresource!
="log",verb!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance,
pod))
            labels:
              quantile: "0.9"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |
              histogram_quantile(0.5,
sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-apiservers",subresource!
="log",verb!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance,
pod))
            labels:
              quantile: "0.5"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
        - interval: 3m
          name: kube-apiserver-availability.rules
          rules:
          - expr: |2
              1 - (
                (
                  # write too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"POST|PUT|PATCH|
DELETE"}[30d]))
                  -
                  sum(increase(apiserver_request_duration_seconds_bucket{verb=~"POST|PUT|
PATCH|DELETE",le="1"}[30d]))
                ) +
                (
                  # read too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"LIST|GET"}
[30d]))
                  -
                  (
                    (
                      sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST|
GET",scope=~"resource|",le="0.1"}[30d]))
                      or
                      vector(0)
                    )
                    +
                    sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST|
GET",scope="namespace",le="0.5"}[30d]))
                    +
                    sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST|
GET",scope="cluster",le="5"}[30d]))
                  )
                ) +
                # errors
                sum(code:apiserver_request_total:increase30d{code=~"5.."} or vector(0))
              )
              /
```

```
              sum(code:apiserver_request_total:increase30d)
            labels:
              verb: all
            record: apiserver_request:availability30d
          - expr: |2
              1 - (
                sum(increase(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[30d]))
                -
                (
                  # too slow
                  (
                    sum(increase(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[30d]))
                    or
                    vector(0)
                  )
                  +
                  sum(increase(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[30d]))
                  +
                  sum(increase(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[30d]))
                )
                +
                # errors
                sum(code:apiserver_request_total:increase30d{verb="read",code=~"5.."} or
vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d{verb="read"})
            labels:
              verb: read
            record: apiserver_request:availability30d
          - expr: |2
              1 - (
                (
                  # too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"POST|PUT|PATCH|
DELETE"}[30d]))
                  -
                  sum(increase(apiserver_request_duration_seconds_bucket{verb=~"POST|PUT|
PATCH|DELETE",le="1"}[30d]))
                )
                +
                # errors
                sum(code:apiserver_request_total:increase30d{verb="write",code=~"5.."} or
vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d{verb="write"})
            labels:
              verb: write
            record: apiserver_request:availability30d
          - expr: |
```

```
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
```

```
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code) (code_verb:apiserver_request_total:increase30d{verb=~"LIST|GET"})
            labels:
              verb: read
            record: code:apiserver_request_total:increase30d
          - expr: |
              sum by (code) (code_verb:apiserver_request_total:increase30d{verb=~"POST|PUT|
PATCH|DELETE"})
            labels:
              verb: write
            record: code:apiserver_request_total:increase30d
    rules_yml: |
      {}
  deployment:
    configmapReload:
```

```
      containers:
        args:
          - --volume-dir=/etc/config
          - --webhook-url=http://127.0.0.1:9090/-/reload
    containers:
      args:
        - --storage.tsdb.retention.time=42d
        - --config.file=/etc/config/prometheus.yml
        - --storage.tsdb.path=/data
        - --web.console.libraries=/etc/prometheus/console_libraries
        - --web.console.templates=/etc/prometheus/consoles
        - --web.enable-lifecycle
    replicas: 1
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 20Gi
    storageClassName: default
  service:
    port: 80
    targetPort: 9090
    type: ClusterIP
pushgateway:
  deployment:
    replicas: 1
  service:
    port: 9091
    targetPort: 9091
    type: ClusterIP
```

Prometheus 구성

Prometheus 구성은 `prometheus-data-values.yaml` 파일에서 설정됩니다. 다음 표에는 사용 가능한 매개 변수와 그에 대한 설명이 나와 있습니다.

표 11-5. Prometheus 구성 매개 변수

매개 변수	설명	유형	기본값
monitoring.namespace	Prometheus가 배포될 네임스페이스	문자열	tanzu-system-monitoring
monitoring.create_namespace	이 플래그는 monitoring.namespace를 통해 지정된 네임스페이스를 생성할지 여부를 나타냅니다.	부울	false
monitoring.prometheus_server.config.prometheus_yaml	Prometheus에 전달할 Kubernetes 클러스터 모니터링 구성 세부 정보	yaml 파일	prometheus.yaml

표 11-5. Prometheus 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.prometheus_server.config.alerting_rules_yaml	Prometheus에 정의된 자세한 경고 규칙	yaml 파일	alerting_rules.yaml
monitoring.prometheus_server.config.recording_rules_yaml	Prometheus에 정의된 자세한 기록 규칙	yaml 파일	recording_rules.yaml
monitoring.prometheus_server.service.type	Prometheus를 노출할 서비스 유형. 지원되는 값: ClusterIP	문자열	ClusterIP
monitoring.prometheus_server.enable_alerts.kubernetes_api	Prometheus의 Kubernetes API에 대해 SLO 경고 사용	부울	true
monitoring.prometheus_server.sc.aws_type	AWS의 storageclass에 대해 정의된 AWS 유형	문자열	gp2
monitoring.prometheus_server.sc.aws_fsType	AWS의 storageclass에 대해 정의된 AWS 파일 시스템 유형	문자열	ext4
monitoring.prometheus_server.sc.allowVolumeExpansion	AWS의 storageclass에 대해 볼륨 확장이 허용되는지 정의	부울	true
monitoring.prometheus_server.pvc.annotations	스토리지 클래스 주석	맵	{}
monitoring.prometheus_server.pvc.storage_class	영구 볼륨 할당에 사용할 스토리지 클래스. 기본값은 null이고 기본 프로비저너가 사용됩니다.	문자열	null
monitoring.prometheus_server.pvc.accessMode	영구 볼륨 할당에 대한 액세스 모드 정의. 지원되는 값: ReadWriteOnce, ReadOnlyMany, ReadWriteMany	문자열	ReadWriteOnce
monitoring.prometheus_server.pvc.storage	영구 볼륨 할당에 대한 스토리지 크기 정의	문자열	8Gi
monitoring.prometheus_server.deployment.replicas	Prometheus 복제본 수	정수	1
monitoring.prometheus_server.image.repository	Prometheus 이미지가 있는 저장소의 위치입니다. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/prometheus
monitoring.prometheus_server.image.name	Prometheus 이미지의 이름	문자열	prometheus

표 11-5. Prometheus 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.prometheus_server.image.tag	Prometheus 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v2.17.1_vmware.1
monitoring.prometheus_server.image.pullPolicy	Prometheus 이미지 Pull 정책	문자열	IfNotPresent
monitoring.alertmanager.config.slack_demo	Alertmanager에 대한 Slack 알림 구성	문자열	`slack_demo:   name: slack_demo   slack_configs:   - api_url: https:// hooks.slack.com   channel: '#alertmanager- test'`
monitoring.alertmanager.config.email_receiver	Alertmanager에 대한 이메일 알림 구성	문자열	`email_receiver:   name: email- receiver   email_configs:   - to: demo@tanzu.com   send_resolved: false   from: from- email@tanzu.com   smarthost: smtp.eample.com:25   require_tls: false`
monitoring.alertmanager.service.type	Alertmanager를 노출할 서비스 유형. 지원되는 값: ClusterIP	문자열	ClusterIP
monitoring.alertmanager.image.repository	Alertmanager 이미지가 있는 저장소의 위치입니다. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/prometheus
monitoring.alertmanager.image.name	Alertmanager 이미지의 이름	문자열	alertmanager
monitoring.alertmanager.image.tag	Alertmanager 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v0.20.0_vmware.1
monitoring.alertmanager.image.pullPolicy	Alertmanager 이미지 Pull 정책	문자열	IfNotPresent

표 11-5. Prometheus 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.alertmanager.pvc.annotations	스토리지 클래스 주석	맵	{}
monitoring.alertmanager.pvc.storage_class	영구 볼륨 할당에 사용할 스토리지 클래스. 기본값은 null이고 기본 프로비저너가 사용됩니다.	문자열	null
monitoring.alertmanager.pvc.accessMode	영구 볼륨 할당에 대한 액세스 모드 정의. 지원되는 값: ReadWriteOnce, ReadOnlyMany, ReadWriteMany	문자열	ReadWriteOnce
monitoring.alertmanager.pvc.storage	영구 볼륨 할당에 대한 스토리지 크기 정의	문자열	2Gi
monitoring.alertmanager.deployment.replicas	Alertmanager 복제본 수	정수	1
monitoring.kube_state_metrics.image.repository	kube-state-metircs 이미지가 포함된 저장소. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/prometheus
monitoring.kube_state_metrics.image.name	kube-state-metircs 이미지의 이름	문자열	kube-state-metrics
monitoring.kube_state_metrics.image.tag	kube-state-metircs 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v1.9.5_vmware.1
monitoring.kube_state_metrics.image.pullPolicy	kube-state-metircs 이미지 Pull 정책	문자열	IfNotPresent
monitoring.kube_state_metrics.deployment.replicas	kube-state-metrics 복제본 수	정수	1
monitoring.node_exporter.image.repository	node-exporter 이미지가 포함된 저장소. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/prometheus
monitoring.node_exporter.image.name	node-exporter 이미지의 이름	문자열	node-exporter
monitoring.node_exporter.image.tag	node-exporter 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v0.18.1_vmware.1

표 11-5. Prometheus 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.node_exporter.image.pullPolicy	node-exporter 이미지 Pull 정책	문자열	IfNotPresent
monitoring.node_exporter.hostNetwork	`hostNetwork: true`로 설정된 경우 포드는 노드의 네트워크 네임스페이스 및 네트워크 리소스를 사용할 수 있습니다.	부울	false
monitoring.node_exporter.deployment.replicas	node-exporter 복제본 수	정수	1
monitoring.pushgateway.image.repository	pushgateway 이미지가 포함된 저장소. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/prometheus
monitoring.pushgateway.image.name	pushgateway 이미지의 이름	문자열	pushgateway
monitoring.pushgateway.image.tag	pushgateway 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v1.2.0_vmware.1
monitoring.pushgateway.image.pullPolicy	pushgateway 이미지 Pull 정책	문자열	IfNotPresent
monitoring.pushgateway.deployment.replicas	pushgateway 복제본 수	정수	1
monitoring.cadvisor.image.repository	cadvisor 이미지가 포함된 저장소. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/prometheus
monitoring.cadvisor.image.name	cadvisor 이미지의 이름	문자열	cadvisor
monitoring.cadvisor.image.tag	cadvisor 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v0.36.0_vmware.1
monitoring.cadvisor.image.pullPolicy	cadvisor 이미지 풀 정책	문자열	IfNotPresent
monitoring.cadvisor.deployment.replicas	cadvisor 복제본 수	정수	1

표 11-5. Prometheus 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.ingress.enabled	Prometheus 및 Alertmanager에 수신 사용/사용 안 함	부울	false 수신을 사용하려면 이 필드를 `true`로 설정하고 Contour를 배포합니다. Prometheus에 액세스하려면 `prometheus.system.tanzu`를 작업자 노드 IP 주소에 매핑하는 항목으로 로컬 `/etc/hosts`를 업데이트합니다.
monitoring.ingress.virtual_host_fqdn	Prometheus 및 Alertmanager에 액세스하기 위한 호스트 이름	문자열	prometheus.system.tanzu
monitoring.ingress.prometheus_prefix	Prometheus의 경로 접두사	문자열	/
monitoring.ingress.alertmanager_prefix	Alertmanager의 경로 접두사	문자열	/alertmanager/
monitoring.ingress.tlsCertificate.tls.crt	자체 TLS 인증서를 사용하려는 경우 수신을 위한 선택적 인증서. 자체 서명된 인증서가 기본적으로 생성됨	문자열	생성된 인증서
monitoring.ingress.tlsCertificate.tls.key	자체 TLS 인증서를 사용하려는 경우, 수신을 위한 선택적 인증서 개인 키.	문자열	생성된 인증서 키

표 11-6. Prometheus_Server Configmap에 대한 구성 가능 필드

매개 변수	설명	유형	기본값
evaluation_interval	규칙을 평가하는 빈도	지속 시간	1m
scrape_interval	대상을 스크레이핑하는 빈도	지속 시간	1m
scrape_timeout	스크레이핑 요청 시간이 초과될 때까지 걸리는 시간	지속 시간	10s
rule_files	규칙 파일은 glob 목록을 지정함. 일치하는 모든 파일에서 규칙 및 경고를 읽음	yaml 파일	
scrape_configs	스크레이핑 구성 목록.	목록	
job_name	스크레이핑된 메트릭에 기본적으로 할당되는 작업 이름	문자열	
kubernetes_sd_configs	Kubernetes 서비스 검색 구성 목록.	목록	
relabel_configs	대상 레이블 재지정 구성 목록.	목록	

표 11-6. Prometheus_Server Configmap에 대한 구성 가능 필드 (계속)

매개 변수	설명	유형	기본값
작업	regex 매칭을 기반으로 수행할 작업.	문자열	
regex	추출된 값을 매칭할 정규식.	문자열	
source_labels	소스 레이블은 기존 레이블에서 값을 선택합니다.	문자열	
scheme	요청에 사용되는 프로토콜 체계를 구성합니다.	문자열	
tls_config	스크레이핑 요청의 TLS 설정을 구성합니다.	문자열	
ca_file	API 서버 인증서의 유효성을 검사하는 데 사용할 CA 인증서.	filename	
insecure_skip_verify	서버 인증서의 유효성 검사를 사용하지 않도록 설정.	부울	
bearer_token_file	보유자 토큰 파일 인증 정보(선택 사항).	filename	
replacement	정규식이 일치하는 경우 정규식 대체를 수행할 대체 값.	문자열	
target_label	대체 작업에서 결과 값이 써지는 레이블.	문자열	

표 11-7. Alertmanager Configmap에 대한 구성 가능 필드

매개 변수	설명	유형	기본값
resolve_timeout	ResolveTimeout은 경고에 EndsAt가 포함되지 않은 경우 Alertmanager에서 사용하는 기본값임	지속 시간	5m
smtp_smarthost	이메일 전송에 사용되는 SMTP 호스트.	지속 시간	1m
slack_api_url	Slack webhook URL.	문자열	global.slack_api_url
pagerduty_url	API 요청을 보낼 pagerduty URL.	문자열	global.pagerduty_url
템플릿	사용자 지정 알림 템플릿 정의를 읽을 파일	파일 경로	
group_by	경고를 레이블로 그룹화	문자열	
group_interval	그룹에 추가된 새 경고에 대한 알림을 보내기 전에 대기할 시간 설정	지속 시간	5m
group_wait	경고 그룹에 대한 알림을 보내기 위해 초기에 대기하는 시간	지속 시간	30s

표 11-7. Alertmanager Configmap에 대한 구성 가능 필드 (계속)

매개 변수	설명	유형	기본값
repeat_interval	경고에 대한 알림이 이미 전송된 경우 알림을 다시 보내기 전에 대기할 시간	지속 시간	4h
receivers	알림 수신기 목록.	목록	
severity	인시던트의 심각도.	문자열	
channel	알림을 보낼 대상 채널 또는 사용자.	문자열	
html	이메일 알림의 HTML 본문.	문자열	
text	이메일 알림의 텍스트 본문.	문자열	
send_resolved	해결된 경고에 대해 알릴지 여부.	filename	
email_configs	이메일 통합 구성	부울	

포드에 대한 주석을 사용하면 스크레이핑 프로세스를 세밀하게 제어할 수 있습니다. 이러한 주석은 포드 메타데이터의 일부여야 합니다. Services 또는 DaemonSets와 같은 다른 개체에 설정하면 효과가 없습니다.

표 11-8. Prometheus 포드 주석

포드 주석	설명
`prometheus.io/scrape`	기본 구성은 모든 포드를 스크레이핑하며 false로 설정하면 이 주석이 스크레이핑 프로세스에서 포드를 제외합니다.
`prometheus.io/path`	메트릭 경로가 /metrics가 아니면 이 주석을 사용하여 정의합니다.
`prometheus.io/port`	포드의 선언된 포트 대신 지정한 포트에서 포드를 스크레이핑합니다 (아무것도 선언되지 않은 경우 기본값은 포트 없는 대상임).

아래의 DaemonSet 매니페스트는 Prometheus가 포트 9102에서 모든 포드를 스크레이핑하도록 지시합니다.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: weave
  labels:
    app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
```

```
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/port: '9102'
    spec:
      containers:
      - name: fluentd-elasticsearch
        image: gcr.io/google-containers/fluentd-elasticsearch:1.20
```

Grafana 패키지 참조

이 항목에서는 Grafana 패키지에 대한 참조 정보를 제공합니다.

Grafana 정보

Grafana(https://grafana.com/)는 오픈 소스 시각화 및 분석 소프트웨어입니다. Grafana를 사용하면 저장된 위치에 상관없이 메트릭을 쿼리, 시각화, 경고 및 탐색할 수 있습니다. Grafana는 애플리케이션 데이터에서 그래프 및 시각화를 형성하는 도구를 제공합니다.

TKG 클러스터에 Grafana 패키지를 설치하려면 다음 항목을 참조하십시오.

- vSphere 8.x용 TKr: Grafana 설치
- vSphere 7.x용 TKr: #unique_177

Grafana 패키지 구성 요소

Grafana 패키지는 표에 나열된 컨테이너를 클러스터에 설치합니다. Grafana 패키지는 패키지 저장소에 지정된 공용 레지스트리에서 컨테이너를 가져옵니다.

컨테이너	리소스 유형	복제	설명
Grafana	배포	2	데이터 시각화

Grafana 데이터 값

아래에는 다음과 같은 사용자 지정이 포함된 `grafana-data-values.yaml` 파일의 예시가 있습니다.

- 수신이 가능하도록 설정됨(ingress: enabled: true)
- /(prefix:)로 끝나는 URL에 대해 수신이 구성됨
- Grafana의 FQDN은 grafana.system.tanzu임(virtual_host_fqdn:)
- grafana에 대한 pvc는 2GB이며 기본 vSphere storageClass 아래에 생성됨
- Grafana UI에 대한 관리자 암호(base64로 인코딩됨)(grafana: secret: admin_password:)

```
namespace: grafana-dashboard
grafana:
  deployment:
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
```

```
    storage: 2Gi
    storageClassName: default
  secret:
    admin_password: admin
    admin_user: YWRtaW4=
    type: Opaque
  service:
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
```

Grafana 구성

Grafana 구성은 `grafana-data-values.yaml`에서 설정됩니다. 다음 표에는 사용 가능한 매개 변수와 그에 대한 설명이 나와 있습니다.

표 11-9. Grafana 구성 매개 변수

매개 변수	설명	유형	기본값
monitoring.namespace	Prometheus가 배포될 네임스페이스	문자열	tanzu-system-monitoring
monitoring.create_namespace	이 플래그는 monitoring.namespace를 통해 지정된 네임스페이스를 생성할지 여부를 나타냅니다.	부울	false
monitoring.grafana.cluster_role.apiGroups	grafana clusterrole에 대해 정의된 API 그룹	목록	[""]
monitoring.grafana.cluster_role.resources	grafana clusterrole에 대해 정의된 리소스	목록	["configmaps", "secrets"]
monitoring.grafana.cluster_role.verbs	clusterrole에 대해 정의된 액세스 권한	목록	["get", "watch", "list"]
monitoring.grafana.config.grafana_ini	Grafana 구성 파일 세부 정보	구성 파일	grafana.ini 이 파일에서 `grafana_net` URL은 Grafana와 통합하는 데 사용됨(예: Grafana.com에서 직접 대시보드 가져오기)
monitoring.grafana.config.datasource.type	Grafana 데이터소스 유형	문자열	prometheus
monitoring.grafana.config.datasource.access	액세스 모드. proxy 또는 direct(UI의 서버 또는 브라우저)	문자열	proxy
monitoring.grafana.config.datasource.isDefault	기본 Grafana 데이터소스로 표시	부울	true

표 11-9. Grafana 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.grafana.config.provider_yaml	Grafana 대시보드 제공자를 정의하는 구성 파일	yaml 파일	provider.yaml
monitoring.grafana.service.type	Grafana를 노출할 서비스 유형. 지원되는 값: ClusterIP, NodePort, LoadBalancer	문자열	vSphere: NodePort, aws/azure: LoadBalancer
monitoring.grafana.pvc.storage_class	영구 볼륨 할당에 대한 액세스 모드 정의. 지원되는 값: ReadWriteOnce, ReadOnlyMany, ReadWriteMany	문자열	ReadWriteOnce
monitoring.grafana.pvc.storage	영구 볼륨 할당에 대한 스토리지 크기 정의	문자열	2Gi
monitoring.grafana.deployment.replicas	Grafana 복제본 수	정수	1
monitoring.grafana.image.repository	Grafana 이미지가 있는 저장소의 위치입니다. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/grafana
monitoring.grafana.image.name	Grafana 이미지의 이름	문자열	grafana
monitoring.grafana.image.tag	Grafana 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	v7.3.5_vmware.1
monitoring.grafana.image.pullPolicy	Grafana 이미지 Pull 정책	문자열	IfNotPresent
monitoring.grafana.secret.type	Grafana 대시보드에 대해 정의된 암호 유형	문자열	불투명
monitoring.grafana.secret.admin_user	Grafana 대시보드에 액세스하기 위한 사용자 이름	문자열	YWRtaW4= 값이 base64로 인코딩됨. 디코딩하려면: `echo "xxxxxx" | base64 --decode`
monitoring.grafana.secret.admin_password	Grafana 대시보드에 액세스하기 위한 암호	문자열	null
monitoring.grafana.secret.ldap_toml	ldap auth를 사용하는 경우 ldap 구성 파일 경로	문자열	""
monitoring.grafana_init_container.image.repository	grafana init 컨테이너 이미지가 포함된 저장소. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/grafana

표 11-9. Grafana 구성 매개 변수 (계속)

매개 변수	설명	유형	기본값
monitoring.grafana_init_container.image.name	grafana init 컨테이너 이미지의 이름	문자열	k8s-sidecar
monitoring.grafana_init_container.image.tag	grafana init 컨테이너 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	0.1.99
monitoring.grafana_init_container.image.pullPolicy	grafana init 컨테이너 이미지 풀 정책	문자열	IfNotPresent
monitoring.grafana_sc_dashboard.image.repository	Grafana 대시보드 이미지가 포함된 저장소. 기본값은 공용 VMware 레지스트리입니다. 비공개 저장소(예: 에어갭 환경)를 사용하는 경우 이 값을 변경합니다.	문자열	projects.registry.vmware.com/tkg/grafana
monitoring.grafana_sc_dashboard.image.name	Grafana 대시보드 이미지의 이름	문자열	k8s-sidecar
monitoring.grafana_sc_dashboard.image.tag	Grafana 대시보드 이미지 태그. 버전을 업그레이드하는 경우 이 값을 업데이트해야 할 수 있습니다.	문자열	0.1.99
monitoring.grafana_sc_dashboard.image.pullPolicy	Grafana 대시보드 이미지 Pull 정책	문자열	IfNotPresent
monitoring.grafana.ingress.enabled	grafana에 수신 사용/사용 안 함	부울	true
monitoring.grafana.ingress.virtual_host_fqdn	Grafana에 액세스하기 위한 호스트 이름	문자열	grafana.system.tanzu
monitoring.grafana.ingress.prefix	Grafana의 경로 접두사	문자열	/
monitoring.grafana.ingress.tlsCertificate.tls.crt	자체 TLS 인증서를 사용하려는 경우 수신을 위한 선택적 인증서. 자체 서명된 인증서가 기본적으로 생성됨	문자열	생성된 인증서
monitoring.grafana.ingress.tlsCertificate.tls.key	자체 TLS 인증서를 사용하려는 경우, 수신을 위한 선택적 인증서 개인 키.	문자열	생성된 인증서 키

Harbor 패키지 참조

이 항목에서는 Harbor 레지스트리 패키지에 대한 참조 정보를 제공합니다.

Harbor 레지스트리 정보

Harbor(https://goharbor.io/)는 이미지 저장소, 이미지 취약성 검색 및 프로젝트 관리를 제공하는 오픈 소스 컨테이너 레지스트리 시스템입니다.

감독자의 TKG 클러스터에 Harbor 패키지를 설치하려면 다음 항목을 참조하십시오.

- vSphere 8.x용 TKr: Harbor 레지스트리 설치
- vSphere 7.x용 TKr: #unique_178

Harbor 구성 요소

Harbor 패키지는 표에 나열된 컨테이너를 클러스터에 설치합니다. 패키지는 패키지 저장소에 지정된 공용 레지스트리에서 컨테이너를 가져옵니다.

컨테이너	리소스 유형	복제	설명
harbor-core	배포	1	엔보이용 관리 및 구성 서버
harbor-database	포드	1	Postgres 데이터베이스
harbor-jobservice	배포	1	Harbor 작업 서비스
harbor-notary-server	배포	1	Harbor Notary 서비스
harbor-notary-signer	배포	1	Harbor Notary
harbor-portal	배포	1	Harbor 웹 인터페이스
harbor-redis	포드	1	Harbor Redis 인스턴스
harbor-registry	배포	2	Harbor 컨테이너 레지스트리 인스턴스
harbor-trivy	포드	1	Harbor 이미지 취약성 스캐너

Harbor 데이터 값

다음은 Harbor 설치를 위한 `harbor-data-values` 예입니다.

데이터 값	설명
`hostname: myharbordomain.com`	Harbor 관리 UI 및 레지스트리 서비스에 액세스하기 위한 FQDN입니다.
`harborAdminPassword: change-it`	Harbor 관리자 계정의 초기 암호입니다. 이것은 설치 중에만 적용됩니다. 설치 후 Harbor UI 또는 API를 사용하여 업데이트할 수 있습니다.
`secretKey: 0123456789ABCDEF`	암호화에 사용되는 비밀 키입니다. 16자의 문자열이어야 합니다.
`database.password: change-it`	postgres 데이터베이스의 초기 암호입니다.
`core.secret: change-it`	암호는 코어 서버가 다른 구성 요소와 통신할 때 사용됩니다.
`xsrfKey: 0123456789ABCDEF0123456789ABCDEF`	XSRF 키입니다. 32자의 문자열이어야 합니다.
`jobservice.secret: change-it`	암호는 작업 서비스가 다른 구성 요소와 통신할 때 사용됩니다.
`registry.secret: change-it`	암호는 클라이언트 및 레지스트리 스토리지 백엔드에서 업로드 상태를 보호하는 데 사용됩니다.

데이터 값	설명
`persistence.persistentVolumeClaim.registry.storageClass: mystorageclass`	볼륨을 프로비저닝하는 데 사용되는 vSphere 스토리지 정책을 지정합니다.
`persistence.persistentVolumeClaim.jobservice.storageClass: mystorageclass`	볼륨을 프로비저닝하는 데 사용되는 vSphere 스토리지 정책을 지정합니다.
`persistence.persistentVolumeClaim.database.storageClass: mystorageclass`	볼륨을 프로비저닝하는 데 사용되는 vSphere 스토리지 정책을 지정합니다.
`persistence.persistentVolumeClaim.redis.storageClass: mystorageclass`	볼륨을 프로비저닝하는 데 사용되는 vSphere 스토리지 정책을 지정합니다.
`persistence.persistentVolumeClaim.trivy.storageClass: mystorageclass`	볼륨을 프로비저닝하는 데 사용되는 vSphere 스토리지 정책을 지정합니다.

Harbor 구성

Harbor 구성은 `harbor-data-values.yaml` 파일에서 설정됩니다. 이 표에는 배포에 필요한 최소 필드가 나열 및 설명되어 있습니다.

속성	값	설명
호스트 이름	FQDN	Harbor UI에 액세스하기 위해 그리고 클라이언트 애플리케이션에서 레지스트리를 참조하기 위해 지정한 FQDN입니다. 도메인이 Contour에서 생성된 엔보이 서비스 IP로 확인되도록 외부 DNS 서버에서 구성되어야 합니다.
tlsCertificate.tlsSecretLabels	{"managed-by": "vmware-vRegistry"}	Tanzu Kubernetes Grid가 Harbor CA를 Tanzu Kubernetes Grid 클러스터에 신뢰할 수 있는 루트로 설치하는 데 사용하는 인증서입니다.
persistence.persistentVolumeClaim.registry.storageClass	스토리지 정책 이름입니다.	Harbor 레지스트리 PVC에 사용되는 스토리지 클래스입니다.
persistence.persistentVolumeClaim.jobservice.storageClass	스토리지 정책 이름입니다.	Harbor 작업 서비스 PVC에 사용되는 스토리지 클래스입니다.
persistence.persistentVolumeClaim.database.storageClass	스토리지 정책 이름입니다.	Harbor 데이터베이스 PVC에 사용되는 스토리지 클래스입니다.
persistence.persistentVolumeClaim.redis.storageClass	스토리지 정책 이름입니다.	Harbor Redis PVC에 사용되는 스토리지 클래스입니다.
persistence.persistentVolumeClaim.trivy.storageClass	스토리지 정책 이름입니다.	Harbor trivy PVC에 사용되는 스토리지 클래스입니다.

vSphere 7.x용 TKr을 사용하여 TKG 클러스터에 표준 패키지 설치

지원되는 vSphere 7.x용 TKr로 프로비저닝된 TKG 서비스 클러스터에 지원되는 표준 패키지를 설치하려면 이 섹션을 참조하십시오.

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로

이 섹션에서는 vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 표준 패키지를 설치하기 위한 지침을 제공합니다.

요구 사항

이러한 지침은 vSphere 7.0.3.6용 TKr v1.27.10 및 vSphere 8.0.1.1용 TKr v1.27.10에서 검증되었습니다. 게시 당시 이는 vSphere 7.x에 대해 가장 최근에 사용 가능한 TKr이었습니다. vSphere 7.x용 TKr은 vSphere 7.x에서 vSphere 8.x로 업그레이드할 목적으로 vSphere 8.x에서 실행할 수 있습니다.

다음 사전 요구 사항을 준수합니다.

- 워크로드 관리 사용
- 감독자 배포됨
- vSphere 네임스페이스 생성됨

  TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.
- vSphere에 대한 Kubernetes CLI 도구가 설치된 Linux 클라이언트

  vSphere에 대한 Kubernetes CLI 도구 설치의 내용을 참조하십시오.

참고 vSphere 8.x용 TKr로 프로비저닝된 TKG 클러스터를 사용하는 경우 표준 패키지 설치 지침은 vSphere 8.x용 TKr을 사용하여 TKG 클러스터에 표준 패키지 설치 설명서를 참조하십시오. TKr 버전에 대한 자세한 내용은 릴리스 정보를 참조하십시오.

TKG 클러스터생성

표준 패키지를 호스팅하기 위한 TKG 클러스터를 생성합니다.

1 TKG 클러스터를 생성합니다.

  Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

  TKr v1.27.10의 Photon 버전에 대한 클러스터 규격의 예입니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-photon
  namespace: tkgs-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
      tkr:
        reference:
          name: v1.27.10---vmware.1-fips.1-tkg.1 #TKR for v7
    nodePools:
```

```
    - name: worker
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
  settings:
    storage:
      defaultClass: vsan-esa-default-policy-raid5
```

TKr v1.27.10의 Ubuntu 버전에 대한 클러스터 규격 예입니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-ubuntu
  namespace: tkgs-ns
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
      tkr:
        reference:
          name: v1.27.10---vmware.1-fips.1-tkg.1.ubuntu #TKR for v7
    nodePools:
    - name: worker
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
  settings:
    storage:
      defaultClass: vsan-esa-default-policy-raid5
```

Carvel 패키지 관리자 설치

Carvel 패키지 관리자를 설치합니다.

1 TKG 클러스터에 로그인합니다.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local --tanzu-
kubernetes-cluster-name tkgs-cluster-photon --tanzu-kubernetes-cluster-namespace tkgs-ns
```

2 Carvel 패키지 관리자를 설치합니다.

```
wget -O- https://carvel.dev/install.sh > install.sh
```

```
sed -i 's/wget -nv -O-/wget --no-check-certificate -nv -O-/' install.sh
```

```
sudo bash install.sh
```

3 설치를 확인합니다.

```
imgpkg version
```

Kapp 컨트롤러 설치

vSphere 7.x용 TKr에 Kapp 컨트롤러 설치의 내용을 참조하십시오.

패키지 저장소 추가

원하는 패키지 저장소 버전을 추가합니다.

1 최신 저장소 태그를 나열합니다.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

2 `packagerepo.yaml`을 생성합니다.

대상 버전과 일치하도록 저장소 버전을 업데이트합니다.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageRepository
metadata:
  name: tanzu-standard
  namespace: tkg-system
spec:
  fetch:
    imgpkgBundle:
      image: projects.registry.vmware.com/tkg/packages/standard/repo:v2024.2.1
```

3 패키지 저장소를 설치합니다.

```
kubectl apply -f packagerepo.yaml
```

예상 결과:

```
packagerepository.packaging.carvel.dev/tanzu-standard created
```

4 패키지 저장소를 확인합니다.

```
kubectl get packagerepositories -A
```

예상 결과:

```
NAMESPACE    NAME             AGE    DESCRIPTION
tkg-system   tanzu-standard   3m9s   Reconcile succeeded
```

Cert Manager 설치

vSphere 7.x용 TKr에 Cert Manager 설치의 내용을 참조하십시오.

엔보이를 사용하여 Contour 설치

vSphere 7.x용 TKr에 Contour 설치의 내용을 참조하십시오.

ExternalDNS 설치

vSphere 7.x용 TKr에 ExternalDNS 설치의 내용을 참조하십시오.

로그 전달을 위한 Fluent Bit 설치

vSphere 7.x용 TKr에 Fluent Bit 설치의 내용을 참조하십시오.

Prometheus 설치

vSphere 7.x용 TKr에 Prometheus 설치의 내용을 참조하십시오.

Grafana 설치

vSphere 7.x용 TKr에 Grafana 설치의 내용을 참조하십시오.

Harbor 설치

vSphere 7.x용 TKr에 Harbor 설치의 내용을 참조하십시오.

vSphere 7.x용 TKr에 Kapp 컨트롤러 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 Kapp 컨트롤러를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

Kapp 컨트롤러 설치

중요 이러한 지침은 vSphere 7.x용 TKr에만 해당됩니다. vSphere 8.x용 TKr에는 Kapp 컨트롤러 패키지가 이미 포함되어 있습니다. vSphere 8.x용 TKr에 Kapp 컨트롤러를 수동으로 설치하지 마십시오.

Kapp 컨트롤러를 설치합니다.

1 바인딩을 생성하여 Kapp 컨트롤러 포드를 실행합니다.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=cluster-admin --group=system:authenticated
```

예상 결과:

```
clusterrolebinding.rbac.authorization.k8s.io/default-tkg-admin-privileged-binding created
```

2 `kapp-controller.yaml`을 준비합니다.

자세한 내용은

3 Kapp 컨트롤러를 설치합니다.

```
kubectl apply -f kapp-controller.yaml
```

4 Kapp 컨트롤러 설치를 확인합니다.

```
kubectl get all -n tkg-system
```

샘플 결과:

```
NAME                READY  STATUS   RESTARTS  AGE
pod/kapp-controller-b7576ddd-p8s87  2/2   Running  0     5m33s
NAME          TYPE     CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
service/packaging-api  ClusterIP  198.201.96.77  <none>    443/TCP  5m34s
NAME               READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/kapp-controller  1/1   1      1     5m33s
```

5 Carvel 사용자 지정 리소스를 확인합니다.

```
kubectl get crd | grep carvel
```

예제 결과:

```
internalpackagemetadatas.internal.packaging.carvel.dev  2024-03-12T08:27:21Z
internalpackages.internal.packaging.carvel.dev     2024-03-12T08:27:21Z
packageinstalls.packaging.carvel.dev          2024-03-12T08:27:21Z
packagerepositories.packaging.carvel.dev        2024-03-12T08:27:22Z
```

kapp-controller.yaml

다음 `kapp-controller.yaml`에는 필수 `securityContext` 설정이 포함되어 있습니다.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tkg-system
---
apiVersion: v1
kind: Namespace
metadata:
  name: kapp-controller-packaging-global
---
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: v1alpha1.data.packaging.carvel.dev
spec:
  group: data.packaging.carvel.dev
  groupPriorityMinimum: 100
  service:
    name: packaging-api
```

```
    namespace: tkg-system
  version: v1alpha1
  versionPriority: 100
---
apiVersion: v1
kind: Service
metadata:
  name: packaging-api
  namespace: tkg-system
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: api
  selector:
    app: kapp-controller
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: internalpackagemetadatas.internal.packaging.carvel.dev
spec:
  group: internal.packaging.carvel.dev
  names:
    kind: InternalPackageMetadata
    listKind: InternalPackageMetadataList
    plural: internalpackagemetadatas
    singular: internalpackagemetadata
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              categories:
                description: Classifiers of the package (optional; Array of strings)
                items:
                  type: string
                type: array
```

```
              displayName:
                description: Human friendly name of the package (optional; string)
                type: string
              iconSVGBase64:
                description: Base64 encoded icon (optional; string)
                type: string
              longDescription:
                description: Long description of the package (optional; string)
                type: string
              maintainers:
                description: List of maintainer info for the package. Currently only
                  supports the name key. (optional; array of maintner info)
                items:
                  properties:
                    name:
                      type: string
                  type: object
                type: array
              providerName:
                description: Name of the entity distributing the package (optional;
                  string)
                type: string
              shortDescription:
                description: Short desription of the package (optional; string)
                type: string
              supportDescription:
                description: Description of the support available for the package
                  (optional; string)
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: internalpackages.internal.packaging.carvel.dev
spec:
  group: internal.packaging.carvel.dev
  names:
    kind: InternalPackage
    listKind: InternalPackageList
    plural: internalpackages
    singular: internalpackage
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
```

```
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              capacityRequirementsDescription:
                description: 'System requirements needed to install the package. Note:
                  these requirements will not be verified by kapp-controller on installation.
                  (optional; string)'
                type: string
              includedSoftware:
                description: IncludedSoftware can be used to show the software contents
                  of a Package. This is especially useful if the underlying versions
                  do not match the Package version
                items:
                  description: IncludedSoftware contains the underlying Software Contents
                    of a Package
                  properties:
                    description:
                      type: string
                    displayName:
                      type: string
                    version:
                      type: string
                  type: object
                type: array
              kappControllerVersionSelection:
                description: KappControllerVersionSelection specifies the versions
                  of kapp-controller which can install this package
                properties:
                  constraints:
                    type: string
                type: object
              kubernetesVersionSelection:
                description: KubernetesVersionSelection specifies the versions of
                  k8s which this package can be installed on
                properties:
                  constraints:
                    type: string
                type: object
              licenses:
                description: Description of the licenses that apply to the package
                  software (optional; Array of strings)
                items:
```

```
        type: string
      type: array
    refName:
      description: The name of the PackageMetadata associated with this
        version Must be a valid PackageMetadata name (see PackageMetadata
        CR for details) Cannot be empty
      type: string
    releaseNotes:
      description: Version release notes (optional; string)
      type: string
    releasedAt:
      description: Timestamp of release (iso8601 formatted string; optional)
      format: date-time
      nullable: true
      type: string
    template:
      properties:
        spec:
          properties:
            canceled:
              description: Cancels current and future reconciliations (optional;
                default=false)
              type: boolean
            cluster:
              description: Specifies that app should be deployed to destination
                cluster; by default, cluster is same as where this resource
                resides (optional; v0.5.0+)
              properties:
                kubeconfigSecretRef:
                  description: Specifies secret containing kubeconfig (required)
                  properties:
                    key:
                      description: Specifies key that contains kubeconfig
                        (optional)
                      type: string
                    name:
                      description: Specifies secret name within app's namespace
                        (required)
                      type: string
                  type: object
                namespace:
                  description: Specifies namespace in destination cluster
                    (optional)
                  type: string
              type: object
            deploy:
              items:
                properties:
                  kapp:
                    description: Use kapp to deploy resources
                    properties:
                      delete:
                        description: Configuration for delete command (optional)
                        properties:
                          rawOptions:
```

```
                                description: Pass through options to kapp delete
                                  (optional)
                                items:
                                  type: string
                                type: array
                            type: object
                          inspect:
                            description: 'Configuration for inspect command
                              (optional) as of kapp-controller v0.31.0, inspect
                              is disabled by default add rawOptions or use an
                              empty inspect config like `inspect: {}` to enable'
                            properties:
                              rawOptions:
                                description: Pass through options to kapp inspect
                                  (optional)
                                items:
                                  type: string
                                type: array
                            type: object
                          intoNs:
                            description: Override namespace for all resources
                              (optional)
                            type: string
                          mapNs:
                            description: Provide custom namespace override mapping
                              (optional)
                            items:
                              type: string
                            type: array
                          rawOptions:
                            description: Pass through options to kapp deploy
                              (optional)
                            items:
                              type: string
                            type: array
                        type: object
                    type: object
                type: array
            fetch:
              items:
                properties:
                  git:
                    description: Uses git to clone repository
                    properties:
                      lfsSkipSmudge:
                        description: Skip lfs download (optional)
                        type: boolean
                      ref:
                        description: Branch, tag, commit; origin is the
                          name of the remote (optional)
                        type: string
                      refSelection:
                        description: Specifies a strategy to resolve to
                          an explicit ref (optional; v0.24.0+)
                        properties:
```

```
            semver:
              properties:
                constraints:
                  type: string
                prereleases:
                  properties:
                    identifiers:
                      items:
                        type: string
                      type: array
                  type: object
              type: object
            type: object
          secretRef:
            description: 'Secret with auth details. allowed
              keys: ssh-privatekey, ssh-knownhosts, username,
              password (optional) (if ssh-knownhosts is not
              specified, git will not perform strict host checking)'
            properties:
              name:
                description: Object is expected to be within
                  same namespace
                type: string
            type: object
          subPath:
            description: Grab only portion of repository (optional)
            type: string
          url:
            description: http or ssh urls are supported (required)
            type: string
        type: object
      helmChart:
        description: Uses helm fetch to fetch specified chart
        properties:
          name:
            description: 'Example: stable/redis'
            type: string
          repository:
            properties:
              secretRef:
                properties:
                  name:
                    description: Object is expected to be within
                      same namespace
                    type: string
                type: object
              url:
                description: Repository url; scheme of oci://
                  will fetch experimental helm oci chart (v0.19.0+)
                  (required)
                type: string
            type: object
          version:
            type: string
        type: object
```

```
                        http:
                          description: Uses http library to fetch file
                          properties:
                            secretRef:
                              description: 'Secret to provide auth details (optional)
                                Secret may include one or more keys: username,
                                password'
                              properties:
                                name:
                                  description: Object is expected to be within
                                    same namespace
                                  type: string
                              type: object
                            sha256:
                              description: Checksum to verify after download (optional)
                              type: string
                            subPath:
                              description: Grab only portion of download (optional)
                              type: string
                            url:
                              description: 'URL can point to one of following
                                formats: text, tgz, zip http and https url are
                                supported; plain file, tgz and tar types are supported
                                (required)'
                              type: string
                          type: object
                        image:
                          description: Pulls content from Docker/OCI registry
                          properties:
                            secretRef:
                              description: 'Secret may include one or more keys:
                                username, password, token. By default anonymous
                                access is used for authentication.'
                              properties:
                                name:
                                  description: Object is expected to be within
                                    same namespace
                                  type: string
                              type: object
                            subPath:
                              description: Grab only portion of image (optional)
                              type: string
                            tagSelection:
                              description: Specifies a strategy to choose a tag
                                (optional; v0.24.0+) if specified, do not include
                                a tag in url key
                              properties:
                                semver:
                                  properties:
                                    constraints:
                                      type: string
                                    prereleases:
                                      properties:
                                        identifiers:
                                          items:
```

```
                                type: string
                              type: array
                          type: object
                      type: object
                  type: object
                url:
                  description: 'Docker image url; unqualified, tagged,
                    or digest references supported (required) Example:
                    username/app1-config:v0.1.0'
                  type: string
              type: object
            imgpkgBundle:
              description: Pulls imgpkg bundle from Docker/OCI registry
                (v0.17.0+)
              properties:
                image:
                  description: Docker image url; unqualified, tagged,
                    or digest references supported (required)
                  type: string
                secretRef:
                  description: 'Secret may include one or more keys:
                    username, password, token. By default anonymous
                    access is used for authentication.'
                  properties:
                    name:
                      description: Object is expected to be within
                        same namespace
                      type: string
                  type: object
                tagSelection:
                  description: Specifies a strategy to choose a tag
                    (optional; v0.24.0+) if specified, do not include
                    a tag in url key
                  properties:
                    semver:
                      properties:
                        constraints:
                          type: string
                        prereleases:
                          properties:
                            identifiers:
                              items:
                                type: string
                              type: array
                          type: object
                      type: object
                  type: object
              type: object
            inline:
              description: Pulls content from within this resource;
                or other resources in the cluster
              properties:
                paths:
                  additionalProperties:
                    type: string
```

```
                    description: Specifies mapping of paths to their
                      content; not recommended for sensitive values
                      as CR is not encrypted (optional)
                    type: object
                  pathsFrom:
                    description: Specifies content via secrets and config
                      maps; data values are recommended to be placed
                      in secrets (optional)
                    items:
                      properties:
                        configMapRef:
                          properties:
                            directoryPath:
                              description: Specifies where to place
                                files found in secret (optional)
                              type: string
                            name:
                              type: string
                          type: object
                        secretRef:
                          properties:
                            directoryPath:
                              description: Specifies where to place
                                files found in secret (optional)
                              type: string
                            name:
                              type: string
                          type: object
                      type: object
                    type: array
                type: object
              path:
                description: Relative path to place the fetched artifacts
                type: string
            type: object
          type: array
        noopDelete:
          description: Deletion requests for the App will result in
            the App CR being deleted, but its associated resources will
            not be deleted (optional; default=false; v0.18.0+)
          type: boolean
        paused:
          description: Pauses _future_ reconciliation; does _not_ affect
            currently running reconciliation (optional; default=false)
          type: boolean
        serviceAccountName:
          description: Specifies that app should be deployed authenticated
            via given service account, found in this namespace (optional;
            v0.6.0+)
          type: string
        syncPeriod:
          description: Specifies the length of time to wait, in time
            + unit format, before reconciling. Always >= 30s. If value
            below 30s is specified, 30s will be used. (optional; v0.9.0+;
            default=30s)
```

```
                type: string
              template:
                items:
                  properties:
                    cue:
                      properties:
                        inputExpression:
                          description: Cue expression for single path component,
                            can be used to unify ValuesFrom into a given field
                            (optional)
                          type: string
                        outputExpression:
                          description: Cue expression to output, default will
                            export all visible fields (optional)
                          type: string
                        paths:
                          description: Explicit list of files/directories
                            (optional)
                          items:
                            type: string
                          type: array
                        valuesFrom:
                          description: Provide values (optional)
                          items:
                            properties:
                              configMapRef:
                                properties:
                                  name:
                                    type: string
                                type: object
                              downwardAPI:
                                properties:
                                  items:
                                    items:
                                      properties:
                                        fieldPath:
                                          description: 'Required: Selects
                                            a field of the app: only annotations,
                                            labels, uid, name and namespace
                                            are supported.'
                                          type: string
                                        kappControllerVersion:
                                          description: 'Optional: Get running
                                            KappController version, defaults
                                            (empty) to retrieving the current
                                            running version.. Can be manually
                                            supplied instead.'
                                          properties:
                                            version:
                                              type: string
                                          type: object
                                        kubernetesAPIs:
                                          description: 'Optional: Get running
                                            KubernetesAPIs from cluster, defaults
                                            (empty) to retrieving the APIs
```

```
                                        from the cluster. Can be manually
                                        supplied instead, e.g ["group/version",
                                        "group2/version2"]'
                                      properties:
                                        groupVersions:
                                          items:
                                            type: string
                                          type: array
                                      type: object
                                    kubernetesVersion:
                                      description: 'Optional: Get running
                                        Kubernetes version from cluster,
                                        defaults (empty) to retrieving
                                        the version from the cluster.
                                        Can be manually supplied instead.'
                                      properties:
                                        version:
                                          type: string
                                      type: object
                                    name:
                                      type: string
                                  type: object
                                type: array
                            type: object
                          path:
                            type: string
                          secretRef:
                            properties:
                              name:
                                type: string
                            type: object
                        type: object
                      type: array
                  type: object
                helmTemplate:
                  description: Use helm template command to render helm
                    chart
                  properties:
                    kubernetesAPIs:
                      description: 'Optional: Use kubernetes group/versions
                        resources available in the live cluster'
                      properties:
                        groupVersions:
                          items:
                            type: string
                          type: array
                      type: object
                    kubernetesVersion:
                      description: 'Optional: Get Kubernetes version,
                        defaults (empty) to retrieving the version from
                        the cluster. Can be manually overridden to a value
                        instead.'
                      properties:
                        version:
                          type: string
```

```
      type: object
    name:
      description: Set name explicitly, default is App
        CR's name (optional; v0.13.0+)
      type: string
    namespace:
      description: Set namespace explicitly, default is
        App CR's namespace (optional; v0.13.0+)
      type: string
    path:
      description: Path to chart (optional; v0.13.0+)
      type: string
    valuesFrom:
      description: One or more secrets, config maps, paths
        that provide values (optional)
      items:
        properties:
          configMapRef:
            properties:
              name:
                type: string
            type: object
          downwardAPI:
            properties:
              items:
                items:
                  properties:
                    fieldPath:
                      description: 'Required: Selects
                        a field of the app: only annotations,
                        labels, uid, name and namespace
                        are supported.'
                      type: string
                    kappControllerVersion:
                      description: 'Optional: Get running
                        KappController version, defaults
                        (empty) to retrieving the current
                        running version.. Can be manually
                        supplied instead.'
                      properties:
                        version:
                          type: string
                      type: object
                    kubernetesAPIs:
                      description: 'Optional: Get running
                        KubernetesAPIs from cluster, defaults
                        (empty) to retrieving the APIs
                        from the cluster. Can be manually
                        supplied instead, e.g ["group/version",
                        "group2/version2"]'
                      properties:
                        groupVersions:
                          items:
                            type: string
                          type: array
```

```
                                    type: object
                                  kubernetesVersion:
                                    description: 'Optional: Get running
                                      Kubernetes version from cluster,
                                      defaults (empty) to retrieving
                                      the version from the cluster.
                                      Can be manually supplied instead.'
                                    properties:
                                      version:
                                        type: string
                                    type: object
                                  name:
                                    type: string
                                type: object
                              type: array
                          type: object
                        path:
                          type: string
                        secretRef:
                          properties:
                            name:
                              type: string
                          type: object
                      type: object
                  type: array
              type: object
            jsonnet:
              description: TODO implement jsonnet
              type: object
            kbld:
              description: Use kbld to resolve image references to
                use digests
              properties:
                paths:
                  items:
                    type: string
                  type: array
              type: object
            kustomize:
              description: TODO implement kustomize
              type: object
            sops:
              description: Use sops to decrypt *.sops.yml files (optional;
                v0.11.0+)
              properties:
                age:
                  properties:
                    privateKeysSecretRef:
                      description: Secret with private armored PGP
                        private keys (required)
                      properties:
                        name:
                          type: string
                      type: object
                  type: object
```

```
                                paths:
                                  description: Lists paths to decrypt explicitly (optional;
                                    v0.13.0+)
                                  items:
                                    type: string
                                  type: array
                                pgp:
                                  description: Use PGP to decrypt files (required)
                                  properties:
                                    privateKeysSecretRef:
                                      description: Secret with private armored PGP
                                        private keys (required)
                                      properties:
                                        name:
                                          type: string
                                      type: object
                                  type: object
                              type: object
                            ytt:
                              description: Use ytt to template configuration
                              properties:
                                fileMarks:
                                  description: Control metadata about input files
                                    passed to ytt (optional; v0.18.0+) see https://
carvel.dev/ytt/docs/latest/file-marks/
                                      for more details
                                  items:
                                    type: string
                                  type: array
                                ignoreUnknownComments:
                                  description: Ignores comments that ytt doesn't recognize
                                    (optional; default=false)
                                  type: boolean
                                inline:
                                  description: Specify additional files, including
                                    data values (optional)
                                  properties:
                                    paths:
                                      additionalProperties:
                                        type: string
                                      description: Specifies mapping of paths to their
                                        content; not recommended for sensitive values
                                        as CR is not encrypted (optional)
                                      type: object
                                    pathsFrom:
                                      description: Specifies content via secrets and
                                        config maps; data values are recommended to
                                        be placed in secrets (optional)
                                      items:
                                        properties:
                                          configMapRef:
                                            properties:
                                              directoryPath:
                                                description: Specifies where to place
                                                  files found in secret (optional)
```

```
                                        type: string
                                      name:
                                        type: string
                                    type: object
                                  secretRef:
                                    properties:
                                      directoryPath:
                                        description: Specifies where to place
                                          files found in secret (optional)
                                        type: string
                                      name:
                                        type: string
                                    type: object
                                type: object
                              type: array
                          type: object
                        paths:
                          description: Lists paths to provide to ytt explicitly
                            (optional)
                          items:
                            type: string
                          type: array
                        strict:
                          description: Forces strict mode https://github.com/k14s/ytt/
blob/develop/docs/strict.md
                            (optional; default=false)
                          type: boolean
                        valuesFrom:
                          description: Provide values via ytt's --data-values-file
                            (optional; v0.19.0-alpha.9)
                          items:
                            properties:
                              configMapRef:
                                properties:
                                  name:
                                    type: string
                                type: object
                              downwardAPI:
                                properties:
                                  items:
                                    items:
                                      properties:
                                        fieldPath:
                                          description: 'Required: Selects
                                            a field of the app: only annotations,
                                            labels, uid, name and namespace
                                            are supported.'
                                          type: string
                                        kappControllerVersion:
                                          description: 'Optional: Get running
                                            KappController version, defaults
                                            (empty) to retrieving the current
                                            running version.. Can be manually
                                            supplied instead.'
                                          properties:
```

```
                                        version:
                                          type: string
                                      type: object
                                    kubernetesAPIs:
                                      description: 'Optional: Get running
                                        KubernetesAPIs from cluster, defaults
                                        (empty) to retrieving the APIs
                                        from the cluster. Can be manually
                                        supplied instead, e.g ["group/version",
                                        "group2/version2"]'
                                      properties:
                                        groupVersions:
                                          items:
                                            type: string
                                          type: array
                                      type: object
                                    kubernetesVersion:
                                      description: 'Optional: Get running
                                        Kubernetes version from cluster,
                                        defaults (empty) to retrieving
                                        the version from the cluster.
                                        Can be manually supplied instead.'
                                      properties:
                                        version:
                                          type: string
                                      type: object
                                    name:
                                      type: string
                                  type: object
                                type: array
                              type: object
                            path:
                              type: string
                            secretRef:
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                        type: array
                      type: object
                    type: object
                  type: array
              type: object
            required:
            - spec
            type: object
          valuesSchema:
            description: valuesSchema can be used to show template values that
              can be configured by users when a Package is installed in an OpenAPI
              schema format.
            properties:
              openAPIv3:
                nullable: true
                type: object
```

```
                    x-kubernetes-preserve-unknown-fields: true
                type: object
              version:
                description: Package version; Referenced by PackageInstall; Must be
                  valid semver (required) Cannot be empty
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: apps.kappctrl.k14s.io
spec:
  group: kappctrl.k14s.io
  names:
    categories:
    - carvel
    kind: App
    listKind: AppList
    plural: apps
    singular: app
  scope: Namespaced
  versions:
  - additionalPrinterColumns:
    - description: Friendly description
      jsonPath: .status.friendlyDescription
      name: Description
      type: string
    - description: Last time app started being deployed. Does not mean anything was
        changed.
      jsonPath: .status.deploy.startedAt
      name: Since-Deploy
      type: date
    - description: Time since creation
      jsonPath: .metadata.creationTimestamp
      name: Age
      type: date
    name: v1alpha1
    schema:
      openAPIV3Schema:
        description: 'An App is a set of Kubernetes resources. These resources could
          span any number of namespaces or could be cluster-wide (e.g. CRDs). An App
          is represented in kapp-controller using a App CR. The App CR comprises of
          three main sections: spec.fetch â€" declare source for fetching configuration
          and OCI images spec.template â€" declare templating tool and values spec.deploy
          â€" declare deployment tool and any deploy specific configuration'
        properties:
          apiVersion:
```

```
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              canceled:
                description: Cancels current and future reconciliations (optional;
                  default=false)
                type: boolean
              cluster:
                description: Specifies that app should be deployed to destination
                  cluster; by default, cluster is same as where this resource resides
                  (optional; v0.5.0+)
                properties:
                  kubeconfigSecretRef:
                    description: Specifies secret containing kubeconfig (required)
                    properties:
                      key:
                        description: Specifies key that contains kubeconfig (optional)
                        type: string
                      name:
                        description: Specifies secret name within app's namespace
                          (required)
                        type: string
                    type: object
                  namespace:
                    description: Specifies namespace in destination cluster (optional)
                    type: string
                type: object
              deploy:
                items:
                  properties:
                    kapp:
                      description: Use kapp to deploy resources
                      properties:
                        delete:
                          description: Configuration for delete command (optional)
                          properties:
                            rawOptions:
                              description: Pass through options to kapp delete (optional)
                              items:
                                type: string
                              type: array
                          type: object
                        inspect:
```

```
                description: 'Configuration for inspect command (optional)
                  as of kapp-controller v0.31.0, inspect is disabled by
                  default add rawOptions or use an empty inspect config
                  like `inspect: {}` to enable'
                properties:
                  rawOptions:
                    description: Pass through options to kapp inspect (optional)
                    items:
                      type: string
                    type: array
                type: object
              intoNs:
                description: Override namespace for all resources (optional)
                type: string
              mapNs:
                description: Provide custom namespace override mapping (optional)
                items:
                  type: string
                type: array
              rawOptions:
                description: Pass through options to kapp deploy (optional)
                items:
                  type: string
                type: array
            type: object
        type: object
      type: array
    fetch:
      items:
        properties:
          git:
            description: Uses git to clone repository
            properties:
              lfsSkipSmudge:
                description: Skip lfs download (optional)
                type: boolean
              ref:
                description: Branch, tag, commit; origin is the name of
                  the remote (optional)
                type: string
              refSelection:
                description: Specifies a strategy to resolve to an explicit
                  ref (optional; v0.24.0+)
                properties:
                  semver:
                    properties:
                      constraints:
                        type: string
                      prereleases:
                        properties:
                          identifiers:
                            items:
                              type: string
                            type: array
                        type: object
```

```
                            type: object
                        type: object
                      secretRef:
                        description: 'Secret with auth details. allowed keys: ssh-
privatekey,
                          ssh-knownhosts, username, password (optional) (if ssh-knownhosts
                          is not specified, git will not perform strict host checking)'
                        properties:
                          name:
                            description: Object is expected to be within same namespace
                            type: string
                        type: object
                      subPath:
                        description: Grab only portion of repository (optional)
                        type: string
                      url:
                        description: http or ssh urls are supported (required)
                        type: string
                    type: object
                  helmChart:
                    description: Uses helm fetch to fetch specified chart
                    properties:
                      name:
                        description: 'Example: stable/redis'
                        type: string
                      repository:
                        properties:
                          secretRef:
                            properties:
                              name:
                                description: Object is expected to be within same
                                  namespace
                                type: string
                            type: object
                          url:
                            description: Repository url; scheme of oci:// will fetch
                              experimental helm oci chart (v0.19.0+) (required)
                            type: string
                        type: object
                      version:
                        type: string
                    type: object
                  http:
                    description: Uses http library to fetch file
                    properties:
                      secretRef:
                        description: 'Secret to provide auth details (optional)
                          Secret may include one or more keys: username, password'
                        properties:
                          name:
                            description: Object is expected to be within same namespace
                            type: string
                        type: object
                      sha256:
                        description: Checksum to verify after download (optional)
```

```
                        type: string
                      subPath:
                        description: Grab only portion of download (optional)
                        type: string
                      url:
                        description: 'URL can point to one of following formats:
                          text, tgz, zip http and https url are supported; plain
                          file, tgz and tar types are supported (required)'
                        type: string
                    type: object
                  image:
                    description: Pulls content from Docker/OCI registry
                    properties:
                      secretRef:
                        description: 'Secret may include one or more keys: username,
                          password, token. By default anonymous access is used for
                          authentication.'
                        properties:
                          name:
                            description: Object is expected to be within same namespace
                            type: string
                        type: object
                      subPath:
                        description: Grab only portion of image (optional)
                        type: string
                      tagSelection:
                        description: Specifies a strategy to choose a tag (optional;
                          v0.24.0+) if specified, do not include a tag in url key
                        properties:
                          semver:
                            properties:
                              constraints:
                                type: string
                              prereleases:
                                properties:
                                  identifiers:
                                    items:
                                      type: string
                                    type: array
                                type: object
                            type: object
                        type: object
                      url:
                        description: 'Docker image url; unqualified, tagged, or
                          digest references supported (required) Example: username/app1-
config:v0.1.0'
                        type: string
                    type: object
                  imgpkgBundle:
                    description: Pulls imgpkg bundle from Docker/OCI registry (v0.17.0+)
                    properties:
                      image:
                        description: Docker image url; unqualified, tagged, or digest
                          references supported (required)
                        type: string
```

```
            secretRef:
              description: 'Secret may include one or more keys: username,
                password, token. By default anonymous access is used for
                authentication.'
              properties:
                name:
                  description: Object is expected to be within same namespace
                  type: string
              type: object
            tagSelection:
              description: Specifies a strategy to choose a tag (optional;
                v0.24.0+) if specified, do not include a tag in url key
              properties:
                semver:
                  properties:
                    constraints:
                      type: string
                    prereleases:
                      properties:
                        identifiers:
                          items:
                            type: string
                          type: array
                      type: object
                  type: object
              type: object
          type: object
        inline:
          description: Pulls content from within this resource; or other
            resources in the cluster
          properties:
            paths:
              additionalProperties:
                type: string
              description: Specifies mapping of paths to their content;
                not recommended for sensitive values as CR is not encrypted
                (optional)
              type: object
            pathsFrom:
              description: Specifies content via secrets and config maps;
                data values are recommended to be placed in secrets (optional)
              items:
                properties:
                  configMapRef:
                    properties:
                      directoryPath:
                        description: Specifies where to place files found
                          in secret (optional)
                        type: string
                      name:
                        type: string
                    type: object
                  secretRef:
                    properties:
                      directoryPath:
```

```
                              description: Specifies where to place files found
                                in secret (optional)
                              type: string
                            name:
                              type: string
                          type: object
                      type: object
                    type: array
              type: object
            path:
              description: Relative path to place the fetched artifacts
              type: string
          type: object
      type: array
    noopDelete:
      description: Deletion requests for the App will result in the App
        CR being deleted, but its associated resources will not be deleted
        (optional; default=false; v0.18.0+)
      type: boolean
    paused:
      description: Pauses _future_ reconciliation; does _not_ affect currently
        running reconciliation (optional; default=false)
      type: boolean
    serviceAccountName:
      description: Specifies that app should be deployed authenticated via
        given service account, found in this namespace (optional; v0.6.0+)
      type: string
    syncPeriod:
      description: Specifies the length of time to wait, in time + unit
        format, before reconciling. Always >= 30s. If value below 30s is
        specified, 30s will be used. (optional; v0.9.0+; default=30s)
      type: string
    template:
      items:
        properties:
          cue:
            properties:
              inputExpression:
                description: Cue expression for single path component, can
                  be used to unify ValuesFrom into a given field (optional)
                type: string
              outputExpression:
                description: Cue expression to output, default will export
                  all visible fields (optional)
                type: string
              paths:
                description: Explicit list of files/directories (optional)
                items:
                  type: string
                type: array
              valuesFrom:
                description: Provide values (optional)
                items:
                  properties:
                    configMapRef:
```

```
      properties:
        name:
          type: string
      type: object
    downwardAPI:
      properties:
        items:
          items:
            properties:
              fieldPath:
                description: 'Required: Selects a field
                  of the app: only annotations, labels,
                  uid, name and namespace are supported.'
                type: string
              kappControllerVersion:
                description: 'Optional: Get running KappController
                  version, defaults (empty) to retrieving
                  the current running version.. Can be manually
                  supplied instead.'
                properties:
                  version:
                    type: string
                type: object
              kubernetesAPIs:
                description: 'Optional: Get running KubernetesAPIs
                  from cluster, defaults (empty) to retrieving
                  the APIs from the cluster. Can be manually
                  supplied instead, e.g ["group/version",
                  "group2/version2"]'
                properties:
                  groupVersions:
                    items:
                      type: string
                    type: array
                type: object
              kubernetesVersion:
                description: 'Optional: Get running Kubernetes
                  version from cluster, defaults (empty)
                  to retrieving the version from the cluster.
                  Can be manually supplied instead.'
                properties:
                  version:
                    type: string
                type: object
              name:
                type: string
            type: object
          type: array
      type: object
    path:
      type: string
    secretRef:
      properties:
        name:
          type: string
```

```
                type: object
            type: object
          type: array
      type: object
    helmTemplate:
      description: Use helm template command to render helm chart
      properties:
        kubernetesAPIs:
          description: 'Optional: Use kubernetes group/versions resources
            available in the live cluster'
          properties:
            groupVersions:
              items:
                type: string
              type: array
          type: object
        kubernetesVersion:
          description: 'Optional: Get Kubernetes version, defaults
            (empty) to retrieving the version from the cluster. Can
            be manually overridden to a value instead.'
          properties:
            version:
              type: string
          type: object
        name:
          description: Set name explicitly, default is App CR's name
            (optional; v0.13.0+)
          type: string
        namespace:
          description: Set namespace explicitly, default is App CR's
            namespace (optional; v0.13.0+)
          type: string
        path:
          description: Path to chart (optional; v0.13.0+)
          type: string
        valuesFrom:
          description: One or more secrets, config maps, paths that
            provide values (optional)
          items:
            properties:
              configMapRef:
                properties:
                  name:
                    type: string
                type: object
              downwardAPI:
                properties:
                  items:
                    items:
                      properties:
                        fieldPath:
                          description: 'Required: Selects a field
                            of the app: only annotations, labels,
                            uid, name and namespace are supported.'
                          type: string
```

```
                                kappControllerVersion:
                                  description: 'Optional: Get running KappController
                                    version, defaults (empty) to retrieving
                                    the current running version.. Can be manually
                                    supplied instead.'
                                  properties:
                                    version:
                                      type: string
                                  type: object
                                kubernetesAPIs:
                                  description: 'Optional: Get running KubernetesAPIs
                                    from cluster, defaults (empty) to retrieving
                                    the APIs from the cluster. Can be manually
                                    supplied instead, e.g ["group/version",
                                    "group2/version2"]'
                                  properties:
                                    groupVersions:
                                      items:
                                        type: string
                                      type: array
                                  type: object
                                kubernetesVersion:
                                  description: 'Optional: Get running Kubernetes
                                    version from cluster, defaults (empty)
                                    to retrieving the version from the cluster.
                                    Can be manually supplied instead.'
                                  properties:
                                    version:
                                      type: string
                                  type: object
                                name:
                                  type: string
                              type: object
                            type: array
                        type: object
                      path:
                        type: string
                      secretRef:
                        properties:
                          name:
                            type: string
                        type: object
                    type: object
                type: array
            type: object
          jsonnet:
            description: TODO implement jsonnet
            type: object
          kbld:
            description: Use kbld to resolve image references to use digests
            properties:
              paths:
                items:
                  type: string
                type: array
```

```
                      type: object
                    kustomize:
                      description: TODO implement kustomize
                      type: object
                    sops:
                      description: Use sops to decrypt *.sops.yml files (optional;
                        v0.11.0+)
                      properties:
                        age:
                          properties:
                            privateKeysSecretRef:
                              description: Secret with private armored PGP private
                                keys (required)
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                        paths:
                          description: Lists paths to decrypt explicitly (optional;
                            v0.13.0+)
                          items:
                            type: string
                          type: array
                        pgp:
                          description: Use PGP to decrypt files (required)
                          properties:
                            privateKeysSecretRef:
                              description: Secret with private armored PGP private
                                keys (required)
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                      type: object
                    ytt:
                      description: Use ytt to template configuration
                      properties:
                        fileMarks:
                          description: Control metadata about input files passed to
                            ytt (optional; v0.18.0+) see https://carvel.dev/ytt/docs/latest/
file-marks/
                            for more details
                          items:
                            type: string
                          type: array
                        ignoreUnknownComments:
                          description: Ignores comments that ytt doesn't recognize
                            (optional; default=false)
                          type: boolean
                        inline:
                          description: Specify additional files, including data values
                            (optional)
                          properties:
```

```
                            paths:
                              additionalProperties:
                                type: string
                              description: Specifies mapping of paths to their content;
                                not recommended for sensitive values as CR is not
                                encrypted (optional)
                              type: object
                            pathsFrom:
                              description: Specifies content via secrets and config
                                maps; data values are recommended to be placed in
                                secrets (optional)
                              items:
                                properties:
                                  configMapRef:
                                    properties:
                                      directoryPath:
                                        description: Specifies where to place files
                                          found in secret (optional)
                                        type: string
                                      name:
                                        type: string
                                    type: object
                                  secretRef:
                                    properties:
                                      directoryPath:
                                        description: Specifies where to place files
                                          found in secret (optional)
                                        type: string
                                      name:
                                        type: string
                                    type: object
                                type: object
                              type: array
                          type: object
                        paths:
                          description: Lists paths to provide to ytt explicitly (optional)
                          items:
                            type: string
                          type: array
                        strict:
                          description: Forces strict mode https://github.com/k14s/ytt/blob/
develop/docs/strict.md
                            (optional; default=false)
                          type: boolean
                        valuesFrom:
                          description: Provide values via ytt's --data-values-file
                            (optional; v0.19.0-alpha.9)
                          items:
                            properties:
                              configMapRef:
                                properties:
                                  name:
                                    type: string
                                type: object
                              downwardAPI:
```

```
                                properties:
                                  items:
                                    items:
                                      properties:
                                        fieldPath:
                                          description: 'Required: Selects a field
                                            of the app: only annotations, labels,
                                            uid, name and namespace are supported.'
                                          type: string
                                        kappControllerVersion:
                                          description: 'Optional: Get running KappController
                                            version, defaults (empty) to retrieving
                                            the current running version.. Can be manually
                                            supplied instead.'
                                          properties:
                                            version:
                                              type: string
                                          type: object
                                        kubernetesAPIs:
                                          description: 'Optional: Get running KubernetesAPIs
                                            from cluster, defaults (empty) to retrieving
                                            the APIs from the cluster. Can be manually
                                            supplied instead, e.g ["group/version",
                                            "group2/version2"]'
                                          properties:
                                            groupVersions:
                                              items:
                                                type: string
                                              type: array
                                          type: object
                                        kubernetesVersion:
                                          description: 'Optional: Get running Kubernetes
                                            version from cluster, defaults (empty)
                                            to retrieving the version from the cluster.
                                            Can be manually supplied instead.'
                                          properties:
                                            version:
                                              type: string
                                          type: object
                                        name:
                                          type: string
                                      type: object
                                    type: array
                                  type: object
                                path:
                                  type: string
                                secretRef:
                                  properties:
                                    name:
                                      type: string
                                  type: object
                              type: object
                            type: array
                          type: object
                        type: object
```

```
            type: array
        type: object
      status:
        properties:
          conditions:
            items:
              properties:
                message:
                  description: Human-readable message indicating details about
                    last transition.
                  type: string
                reason:
                  description: Unique, this should be a short, machine understandable
                    string that gives the reason for condition's last transition.
                    If it reports "ResizeStarted" that means the underlying persistent
                    volume is being resized.
                  type: string
                status:
                  type: string
                type:
                  description: ConditionType represents reconciler state
                  type: string
              required:
              - status
              - type
              type: object
            type: array
          consecutiveReconcileFailures:
            type: integer
          consecutiveReconcileSuccesses:
            type: integer
          deploy:
            properties:
              error:
                type: string
              exitCode:
                type: integer
              finished:
                type: boolean
              kapp:
                description: KappDeployStatus contains the associated AppCR deployed
                  resources
                properties:
                  associatedResources:
                    description: AssociatedResources contains the associated App
                      label, namespaces and GKs
                    properties:
                      groupKinds:
                        items:
                          description: GroupKind specifies a Group and a Kind,
                            but does not force a version.  This is useful for
                            identifying concepts during lookup stages without
                            having partially valid types
                          properties:
                            group:
```

```
                              type: string
                            kind:
                              type: string
                          required:
                          - group
                          - kind
                          type: object
                        type: array
                      label:
                        type: string
                      namespaces:
                        items:
                          type: string
                        type: array
                    type: object
              type: object
            startedAt:
              format: date-time
              type: string
            stderr:
              type: string
            stdout:
              type: string
            updatedAt:
              format: date-time
              type: string
        type: object
      fetch:
        properties:
          error:
            type: string
          exitCode:
            type: integer
          startedAt:
            format: date-time
            type: string
          stderr:
            type: string
          stdout:
            type: string
          updatedAt:
            format: date-time
            type: string
        type: object
      friendlyDescription:
        type: string
      inspect:
        properties:
          error:
            type: string
          exitCode:
            type: integer
          stderr:
            type: string
          stdout:
```

```
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              managedAppName:
                type: string
              observedGeneration:
                description: Populated based on metadata.generation when controller
                  observes a change to the resource; if this value is out of data,
                  other status fields do not reflect latest state
                format: int64
                type: integer
              template:
                properties:
                  error:
                    type: string
                  exitCode:
                    type: integer
                  stderr:
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              usefulErrorMessage:
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: packageinstalls.packaging.carvel.dev
spec:
  group: packaging.carvel.dev
  names:
    categories:
    - carvel
    kind: PackageInstall
    listKind: PackageInstallList
    plural: packageinstalls
    shortNames:
    - pkgi
    singular: packageinstall
  scope: Namespaced
  versions:
  - additionalPrinterColumns:
    - description: PackageMetadata name
```

```
      jsonPath: .spec.packageRef.refName
      name: Package name
      type: string
    - description: PackageMetadata version
      jsonPath: .status.version
      name: Package version
      type: string
    - description: Friendly description
      jsonPath: .status.friendlyDescription
      name: Description
      type: string
    - description: Time since creation
      jsonPath: .metadata.creationTimestamp
      name: Age
      type: date
    name: v1alpha1
    schema:
      openAPIV3Schema:
        description: A Package Install is an actual installation of a package and
          its underlying resources on a Kubernetes cluster. It is represented in kapp-
controller
          by a PackageInstall CR. A PackageInstall CR must reference a Package CR.
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              canceled:
                description: Canceled when set to true will stop all active changes
                type: boolean
              cluster:
                description: Specifies that Package should be deployed to destination
                  cluster; by default, cluster is same as where this resource resides
                  (optional)
                properties:
                  kubeconfigSecretRef:
                    description: Specifies secret containing kubeconfig (required)
                    properties:
                      key:
                        description: Specifies key that contains kubeconfig (optional)
                        type: string
                      name:
                        description: Specifies secret name within app's namespace
```

```
                (required)
              type: string
          type: object
        namespace:
          description: Specifies namespace in destination cluster (optional)
          type: string
      type: object
    noopDelete:
      description: When NoopDelete set to true, PackageInstall deletion
        should delete PackageInstall/App CR but preserve App's associated
        resources.
      type: boolean
    packageRef:
      description: Specifies the name of the package to install (required)
      properties:
        refName:
          type: string
        versionSelection:
          properties:
            constraints:
              type: string
            prereleases:
              properties:
                identifiers:
                  items:
                    type: string
                  type: array
              type: object
          type: object
      type: object
    paused:
      description: Paused when set to true will ignore all pending changes,
        once it set back to false, pending changes will be applied
      type: boolean
    serviceAccountName:
      description: Specifies service account that will be used to install
        underlying package contents
      type: string
    syncPeriod:
      description: Controls frequency of App reconciliation in time + unit
        format. Always >= 30s. If value below 30s is specified, 30s will
        be used.
      type: string
    values:
      description: Values to be included in package's templating step (currently
        only included in the first templating step) (optional)
      items:
        properties:
          secretRef:
            properties:
              key:
                type: string
              name:
                type: string
            type: object
```

```
                  type: object
                type: array
            type: object
          status:
            properties:
              conditions:
                items:
                  properties:
                    message:
                      description: Human-readable message indicating details about
                        last transition.
                      type: string
                    reason:
                      description: Unique, this should be a short, machine understandable
                        string that gives the reason for condition's last transition.
                        If it reports "ResizeStarted" that means the underlying persistent
                        volume is being resized.
                      type: string
                    status:
                      type: string
                    type:
                      description: ConditionType represents reconciler state
                      type: string
                  required:
                  - status
                  - type
                  type: object
                type: array
              friendlyDescription:
                type: string
              lastAttemptedVersion:
                description: LastAttemptedVersion specifies what version was last
                  attempted to be installed. It does _not_ indicate it was successfully
                  installed.
                type: string
              observedGeneration:
                description: Populated based on metadata.generation when controller
                  observes a change to the resource; if this value is out of data,
                  other status fields do not reflect latest state
                format: int64
                type: integer
              usefulErrorMessage:
                type: string
              version:
                description: TODO this is desired resolved version (not actually deployed)
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
```

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    packaging.carvel.dev/global-namespace: kapp-controller-packaging-global
  name: packagerepositories.packaging.carvel.dev
spec:
  group: packaging.carvel.dev
  names:
    categories:
    - carvel
    kind: PackageRepository
    listKind: PackageRepositoryList
    plural: packagerepositories
    shortNames:
    - pkgr
    singular: packagerepository
  scope: Namespaced
  versions:
  - additionalPrinterColumns:
    - description: Time since creation
      jsonPath: .metadata.creationTimestamp
      name: Age
      type: date
    - description: Friendly description
      jsonPath: .status.friendlyDescription
      name: Description
      type: string
    name: v1alpha1
    schema:
      openAPIV3Schema:
        description: A package repository is a collection of packages and their metadata.
          Similar to a maven repository or a rpm repository, adding a package repository
          to a cluster gives users of that cluster the ability to install any of the
          packages from that repository.
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              fetch:
                properties:
                  git:
```

```
      description: Uses git to clone repository containing package list
      properties:
        lfsSkipSmudge:
          description: Skip lfs download (optional)
          type: boolean
        ref:
          description: Branch, tag, commit; origin is the name of the
            remote (optional)
          type: string
        refSelection:
          description: Specifies a strategy to resolve to an explicit
            ref (optional; v0.24.0+)
          properties:
            semver:
              properties:
                constraints:
                  type: string
                prereleases:
                  properties:
                    identifiers:
                      items:
                        type: string
                      type: array
                  type: object
              type: object
          type: object
        secretRef:
          description: 'Secret with auth details. allowed keys: ssh-privatekey,
            ssh-knownhosts, username, password (optional) (if ssh-knownhosts
            is not specified, git will not perform strict host checking)'
          properties:
            name:
              description: Object is expected to be within same namespace
              type: string
          type: object
        subPath:
          description: Grab only portion of repository (optional)
          type: string
        url:
          description: http or ssh urls are supported (required)
          type: string
      type: object
    http:
      description: Uses http library to fetch file containing packages
      properties:
        secretRef:
          description: 'Secret to provide auth details (optional) Secret
            may include one or more keys: username, password'
          properties:
            name:
              description: Object is expected to be within same namespace
              type: string
          type: object
        sha256:
          description: Checksum to verify after download (optional)
```

```
                        type: string
                      subPath:
                        description: Grab only portion of download (optional)
                        type: string
                      url:
                        description: 'URL can point to one of following formats: text,
                          tgz, zip http and https url are supported; plain file, tgz
                          and tar types are supported (required)'
                        type: string
                    type: object
                  image:
                    description: Image url; unqualified, tagged, or digest references
                      supported (required)
                    properties:
                      secretRef:
                        description: 'Secret may include one or more keys: username,
                          password, token. By default anonymous access is used for
                          authentication.'
                        properties:
                          name:
                            description: Object is expected to be within same namespace
                            type: string
                        type: object
                      subPath:
                        description: Grab only portion of image (optional)
                        type: string
                      tagSelection:
                        description: Specifies a strategy to choose a tag (optional;
                          v0.24.0+) if specified, do not include a tag in url key
                        properties:
                          semver:
                            properties:
                              constraints:
                                type: string
                              prereleases:
                                properties:
                                  identifiers:
                                    items:
                                      type: string
                                    type: array
                                type: object
                            type: object
                        type: object
                      url:
                        description: 'Docker image url; unqualified, tagged, or digest
                          references supported (required) Example: username/app1-
config:v0.1.0'
                        type: string
                    type: object
                  imgpkgBundle:
                    description: Pulls imgpkg bundle from Docker/OCI registry
                    properties:
                      image:
                        description: Docker image url; unqualified, tagged, or digest
                          references supported (required)
```

```
          type: string
        secretRef:
          description: 'Secret may include one or more keys: username,
            password, token. By default anonymous access is used for
            authentication.'
          properties:
            name:
              description: Object is expected to be within same namespace
              type: string
          type: object
        tagSelection:
          description: Specifies a strategy to choose a tag (optional;
            v0.24.0+) if specified, do not include a tag in url key
          properties:
            semver:
              properties:
                constraints:
                  type: string
                prereleases:
                  properties:
                    identifiers:
                      items:
                        type: string
                      type: array
                  type: object
              type: object
          type: object
      type: object
    inline:
      description: Pull content from within this resource; or other
        resources in the cluster
      properties:
        paths:
          additionalProperties:
            type: string
          description: Specifies mapping of paths to their content;
            not recommended for sensitive values as CR is not encrypted
            (optional)
          type: object
        pathsFrom:
          description: Specifies content via secrets and config maps;
            data values are recommended to be placed in secrets (optional)
          items:
            properties:
              configMapRef:
                properties:
                  directoryPath:
                    description: Specifies where to place files found
                      in secret (optional)
                    type: string
                  name:
                    type: string
                type: object
              secretRef:
                properties:
```

```
                              directoryPath:
                                description: Specifies where to place files found
                                  in secret (optional)
                                type: string
                              name:
                                type: string
                            type: object
                        type: object
                    type: array
                type: object
            type: object
          paused:
            description: Paused when set to true will ignore all pending changes,
              once it set back to false, pending changes will be applied
            type: boolean
          syncPeriod:
            description: Controls frequency of PackageRepository reconciliation
            type: string
        required:
        - fetch
        type: object
      status:
        properties:
          conditions:
            items:
              properties:
                message:
                  description: Human-readable message indicating details about
                    last transition.
                  type: string
                reason:
                  description: Unique, this should be a short, machine understandable
                    string that gives the reason for condition's last transition.
                    If it reports "ResizeStarted" that means the underlying persistent
                    volume is being resized.
                  type: string
                status:
                  type: string
                type:
                  description: ConditionType represents reconciler state
                  type: string
              required:
              - status
              - type
              type: object
            type: array
          consecutiveReconcileFailures:
            type: integer
          consecutiveReconcileSuccesses:
            type: integer
          deploy:
            properties:
              error:
                type: string
              exitCode:
```

```
                type: integer
              finished:
                type: boolean
              kapp:
                description: KappDeployStatus contains the associated AppCR deployed
                  resources
                properties:
                  associatedResources:
                    description: AssociatedResources contains the associated App
                      label, namespaces and GKs
                    properties:
                      groupKinds:
                        items:
                          description: GroupKind specifies a Group and a Kind,
                            but does not force a version.  This is useful for
                            identifying concepts during lookup stages without
                            having partially valid types
                          properties:
                            group:
                              type: string
                            kind:
                              type: string
                          required:
                          - group
                          - kind
                          type: object
                        type: array
                      label:
                        type: string
                      namespaces:
                        items:
                          type: string
                        type: array
                    type: object
                type: object
              startedAt:
                format: date-time
                type: string
              stderr:
                type: string
              stdout:
                type: string
              updatedAt:
                format: date-time
                type: string
            type: object
          fetch:
            properties:
              error:
                type: string
              exitCode:
                type: integer
              startedAt:
                format: date-time
                type: string
```

```
                  stderr:
                    type: string
                  stdout:
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              friendlyDescription:
                type: string
              observedGeneration:
                description: Populated based on metadata.generation when controller
                  observes a change to the resource; if this value is out of data,
                  other status fields do not reflect latest state
                format: int64
                type: integer
              template:
                properties:
                  error:
                    type: string
                  exitCode:
                    type: integer
                  stderr:
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              usefulErrorMessage:
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    kapp-controller.carvel.dev/version: v0.45.2
    kbld.k14s.io/images: |
      - origins:
        - local:
            path: /home/runner/work/kapp-controller/kapp-controller
        - git:
            dirty: true
            remoteURL: https://github.com/carvel-dev/kapp-controller
            sha: e3beee23d49899bfc681c9d980c1a3bdc0fa14ac
            tags:
            - v0.45.2
        url: ghcr.io/carvel-dev/kapp-
```

```
controller@sha256:d5c5b259d10f8a561fe6717a735ceb053ccb13320f55428977d1d8df46b9bc0d
  name: kapp-controller
  namespace: tkg-system
spec:
  replicas: 1
  revisionHistoryLimit: 0
  selector:
    matchLabels:
      app: kapp-controller
  template:
    metadata:
      labels:
        app: kapp-controller
    spec:
      containers:
      - args:
        - -packaging-global-namespace=kapp-controller-packaging-global
        - -enable-api-priority-and-fairness=True
        - -tls-cipher-suites=
        env:
        - name: KAPPCTRL_MEM_TMP_DIR
          value: /etc/kappctrl-mem-tmp
        - name: KAPPCTRL_SIDECAREXEC_SOCK
          value: /etc/kappctrl-mem-tmp/sidecarexec.sock
        - name: KAPPCTRL_SYSTEM_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: KAPPCTRL_API_PORT
          value: "10350"
        image: ghcr.io/carvel-dev/kapp-
controller@sha256:d5c5b259d10f8a561fe6717a735ceb053ccb13320f55428977d1d8df46b9bc0d
        name: kapp-controller
        ports:
        - containerPort: 10350
          name: api
          protocol: TCP
        resources:
          requests:
            cpu: 120m
            memory: 100Mi
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          readOnlyRootFilesystem: true
          runAsNonRoot: true
          seccompProfile:
            type: RuntimeDefault
        volumeMounts:
        - mountPath: /etc/kappctrl-mem-tmp
          name: template-fs
        - mountPath: /home/kapp-controller
          name: home
```

```
      - args:
        - --sidecarexec
        env:
        - name: KAPPCTRL_SIDECAREXEC_SOCK
          value: /etc/kappctrl-mem-tmp/sidecarexec.sock
        - name: IMGPKG_ACTIVE_KEYCHAINS
          value: gke,aks,ecr
        image: ghcr.io/carvel-dev/kapp-
controller@sha256:d5c5b259d10f8a561fe6717a735ceb053ccb13320f55428977d1d8df46b9bc0d
        name: kapp-controller-sidecarexec
        resources:
          requests:
            cpu: 120m
            memory: 100Mi
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          readOnlyRootFilesystem: false
          runAsNonRoot: true
          seccompProfile:
            type: RuntimeDefault
        volumeMounts:
        - mountPath: /etc/kappctrl-mem-tmp
          name: template-fs
        - mountPath: /home/kapp-controller
          name: home
        - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
          name: empty-sa
      serviceAccount: kapp-controller-sa
      volumes:
      - emptyDir:
          medium: Memory
        name: template-fs
      - emptyDir:
          medium: Memory
        name: home
      - emptyDir: {}
        name: empty-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kapp-controller-sa
  namespace: tkg-system
```

vSphere 7.x용 TKr에 Cert Manager 설치

vSphere 7.x용 TKr에 Cert Manager를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

Cert Manager 설치

Cert Manager를 설치합니다.

1 사용 가능한 Cert Manager 패키지 버전을 나열합니다.

```
kubectl -n tkg-system get packages | grep cert-manager
```

2 대상 버전으로 `cert-manager.yaml`을 생성합니다.

cert-manager.yaml의 내용을 참조하십시오.

3 Cert Manager를 설치합니다.

```
kubectl apply -f cert-manager.yaml
```

예상 결과:

```
serviceaccount/cert-manager-sa created
clusterrolebinding.rbac.authorization.k8s.io/admin created
packageinstall.packaging.carvel.dev/cert-manager created
secret/cert-manager-data-values created
```

4 Cert Manager 설치를 확인합니다.

```
kubectl get pkgi -A
```

예상 결과:

```
NAMESPACE   NAME      PACKAGE NAME          PACKAGE VERSION     DESCRIPTION      AGE
tkg-system  cert-manager  cert-manager.tanzu.vmware.com 1.12.2+vmware.2-tkg.2 Reconcile
succeeded  57s
```

5 Cert Manager 포드를 확인합니다.

```
kubectl get pods -A
```

```
NAMESPACE           NAME                                       READY   STATUS
RESTARTS      AGE
tkg-system        cert-manager-666586c866-826rz               1/1     Running
0             48s
tkg-system        cert-manager-cainjector-68697ccc4b-xbfff     1/1     Running
0             48s
tkg-system        cert-manager-webhook-57ccbd4db9-tzw4c        1/1     Running
0             48s
```

cert-manager.yaml

Cert Manager를 설치하려면 다음 `cert-manager.yaml` 예를 참조하십시오. 대상 패키지 버전과 일치하도록 버전 변수를 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cert-manager-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: cert-manager-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: cert-manager
  namespace: tkg-system
spec:
  serviceAccountName: cert-manager-sa
  packageRef:
    refName: cert-manager.tanzu.vmware.com
    versionSelection:
      constraints: 1.12.2+vmware.2-tkg.2 #PKG-VERSION
  values:
  - secretRef:
      name: cert-manager-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: cert-manager-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tkg-system
```

vSphere 7.x용 TKr에 Contour 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 표준 패키지를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

엔보이를 사용하여 Contour 설치

엔보이 서비스를 사용하여 Contour 수신을 설치합니다.

1 저장소에서 사용 가능한 Contour 버전을 나열합니다.

```
kubectl get packages -n tkg-system | grep contour
```

2 `contour.yaml` 규격을 생성합니다.

#unique_187/unique_187_Connect_42_GUID-CC995CF8-0F4B-4D92-A782-A3832C0EA5AE의 내용을 참조하십시오.

3 필요한 경우 환경에 맞게 `contour-data-values`를 사용자 지정합니다.

Contour 패키지 참조의 내용을 참조하십시오.

4 Contour를 설치합니다.

```
kubectl apply -f contour.yaml
```

```
serviceaccount/contour-sa
createdclusterrolebinding.rbac.authorization.k8s.io/contour-role-binding created
packageinstall.packaging.carvel.dev/contour created
secret/contour-data-values created
```

5 Contour 패키지 설치를 확인합니다.

```
kubectl get pkgi -A
```

6 Contour 개체를 확인합니다.

```
kubectl get all -n contour-ingress
```

```
NAME                            READY   STATUS    RESTARTS   AGE
pod/contour-777bdddc69-fqnsp    1/1     Running   0          102s
pod/contour-777bdddc69-gs5xv    1/1     Running   0          102s
pod/envoy-d4jtt                 2/2     Running   0          102s
pod/envoy-g5h72                 2/2     Running   0          102s
pod/envoy-pjpzc                 2/2     Running   0          102s

NAME              TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                      AGE
service/contour   ClusterIP      10.105.242.46   <none>
8001/TCP                     102s
service/envoy     LoadBalancer   10.103.245.57   10.197.154.69   80:32642/
TCP,443:30297/TCP   102s

NAME                    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
```

```
SELECTOR   AGE
daemonset.apps/envoy   3         3         3       3            3
<none>          102s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/contour     2/2     2            2           102s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/contour-777bdddc69      2         2         2       102s
```

Contour 패키지는 2개의 Contour 포드와 3개의 엔보이 포드를 설치합니다. Contour와 엔보이가 모두 서비스로 노출됩니다. 이 예에서, 엔보이 서비스의 외부 IP 주소는 10.197.154.69입니다. 이 IP 주소는 **워크로드 네트워크 > 수신**에 지정된 CIDR에서 분할됩니다. 이 IP 주소에 대해 로드 밸런서 인스턴스가 생성됩니다. 이 로드 밸런서에 대한 서버 풀의 멤버는 엔보이 포드입니다. 엔보이 포드는 해당 주소가 실행되는 작업자 노드의 IP 주소를 가정합니다. 클러스터 노드(`kubectl get nodes -o wide`)를 쿼리하여 이러한 IP를 볼 수 있습니다.

contour.yaml

다음 `contour.yaml`을 사용하여 엔보이로 Contour를 설치합니다. 대상 패키지 버전과 일치하도록 버전 변수를 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: contour-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: contour-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: contour-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tkg-system
spec:
  serviceAccountName: contour-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: 1.26.1+vmware.1-tkg.1 #PKG-VERSION
  values:
```

```
  - secretRef:
      name: contour-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: contour-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-ingress
    contour:
      configFileContents: {}
      useProxyProtocol: false
      replicas: 2
      pspNames: "vmware-system-restricted"
      logLevel: info
    envoy:
      service:
        type: LoadBalancer
        annotations: {}
        externalTrafficPolicy: Cluster
        disableWait: false
      hostPorts:
        enable: true
        http: 80
        https: 443
      hostNetwork: false
      terminationGracePeriodSeconds: 300
      logLevel: info
    certificates:
      duration: 8760h
      renewBefore: 360h
```

vSphere 7.x용 TKr에 ExternalDNS 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 ExternalDNS를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

ExternalDNS 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 ExternalDNS를 설치합니다.

1 저장소에서 사용 가능한 ExternalDNS 버전을 나열합니다.

```
kubectl get packages -n tkg-system | grep external-dns
```

2 ExternalDNS 네임스페이스를 생성합니다.

```
kubectl create namespace tanzu-system-service-discovery --dry-run=client -o yaml | kubectl
apply -f -
```

3 네임스페이스에서 보안 상태를 설정합니다.

```
kubectl label namespace tanzu-system-service-discovery pod-security.kubernetes.io/
enforce=privileged
```

4 바인딩 배포 YAML을 준비합니다.

bind-deployment.yaml의 내용을 참조하십시오.

5 바인딩 DNS 서버를 배포합니다.

```
kubectl apply -n tanzu-system-service-discovery -f bind-deployment.yaml
```

6 ExternalDNS 배포 YAML을 준비합니다.

external-dns-deploy.yaml의 내용을 참조하십시오.

7 `external-dns-default-values.yaml` 파일을 사용하여 암호를 생성합니다.

```
svcip=$(kubectl get svc bind -n tanzu-system-service-discovery -o
jsonpath='{.spec.clusterIP}')sed -i "s/--rfc2136-host=[0-9.]\+/--rfc2136-host=$svcip/g"
external-dns-deploy.yaml
```

```
kubectl create secret generic external-dns-default-values --from-file=values.yaml=external-
dns-deploy.yaml -n tkg-system
```

8 암호를 확인합니다.

```
kubectl get secret external-dns-default-values -n tkg-system
```

```
kubectl get secret external-dns-default-values -n tkg-system -oyaml
```

9 ExternalDNS 패키지 설치 YAML을 준비합니다.

external-dns-packageinstall.yaml의 내용을 참조하십시오.

10 바인딩을 구성합니다.

```
sed -i "s/--rfc2136-host=[0-9.]\+/--rfc2136-host=$svcip/g" external-dns-packageinstall.yaml
```

11 외부 DNS 패키지를 생성합니다.

```
kubectl apply -f external-dns-packageinstall.yaml
```

12 ExternalDNS 설치를 확인합니다.

```
kubectl get all -n tanzu-system-service-discovery
```

bind-deployment.yaml

예 `bind-deployment.yaml`.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: bind-config
data:
  named.conf: |
    key "externaldns-key" {
            algorithm hmac-sha256;
            secret "O0DhTJzZ0GjfuQmB9TBc1ELchv5oDMTlQs3NNOdMZJU=";
    };

    # bind needs to recurse to coredns in the case of resolving CNAME records
    # it may know about to A records. E.g This test runs on AWS which uses
    # CNAMEs for their LoadBalancer Services and bind will want to resolve
    # those CNAME records to A records using an upstream DNS server.
    options {
        recursion yes;
        forwarders {
            COREDNS_CLUSTER_IP;
        };
        forward only;
        dnssec-enable yes;
        dnssec-validation yes;
    };

    zone "k8s.example.org" {
        type master;
        file "/etc/bind/k8s.zone";
        allow-transfer {
            key "externaldns-key";
        };
        update-policy {
            grant externaldns-key zonesub ANY;
        };
    };
  k8s.zone: |
    $TTL 60 ; 1 minute
    @         IN SOA  k8s.example.org. root.k8s.example.org. (
                                    16         ; serial
                                    60         ; refresh (1 minute)
                                    60         ; retry (1 minute)
                                    60         ; expire (1 minute)
                                    60         ; minimum (1 minute)
                                    )
                            NS      ns.k8s.example.org.
    ns                      A       1.2.3.4
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: bind
spec:
  selector:
    matchLabels:
      app: bind
  template:
    metadata:
      labels:
        app: bind
    spec:
      containers:
      - name: bind
        image: docker.io/internetsystemsconsortium/bind9:9.16
        imagePullPolicy: IfNotPresent
        command:
        - 'sh'
        - '-c'
        - |
          /usr/sbin/named -g -c /etc/bind/named.conf
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        volumeMounts:
          - name: named-conf-volume
            mountPath: /etc/bind/named.conf
            subPath: named.conf
          - name: k8s-zone-volume
            mountPath: /etc/bind/k8s.zone
            subPath: k8s.zone
      volumes:
        - name: data
          emptyDir: {}
        - name: named-conf-volume
          configMap:
            name: bind-config
            items:
            - key: named.conf
              path: named.conf
        - name: k8s-zone-volume
          configMap:
            name: bind-config
            items:
            - key: k8s.zone
              path: k8s.zone
---
apiVersion: v1
kind: Service
metadata:
  name: bind
  labels:
    app: bind
```

```
spec:
  selector:
    app: bind
  type: ClusterIP
  ports:
  - port: 53
    targetPort: 53
    protocol: TCP
    name: dns-tcp
  - port: 53
    targetPort: 53
    protocol: UDP
    name: dns
```

external-dns-deploy.yaml

예 `external-dns-deploy.yaml`.

```
deployment:
  args:
  - --source=service
  - --source=ingress
  - --txt-owner-id=k8s
  - --domain-filter=k8s.example.org
  - --namespace=default
  - --provider=rfc2136
  - --rfc2136-host=198.201.49.227
  - --rfc2136-port=53
  - --rfc2136-zone=k8s.example.org
  - --rfc2136-tsig-secret=O0DhTJzZ0GjfuQmB9TBc1ELchv5oDMTlQs3NNOdMZJU=
  - --rfc2136-tsig-secret-alg=hmac-sha256
  - --rfc2136-tsig-keyname=externaldns-key
```

external-dns-packageinstall.yaml

다음 예는 BIND에 사용할 수 있습니다. 필요에 따라 패키지 버전을 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: external-dns-default-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: external-dns-default-sa
```

```
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: external-dns-default-sa
  packageRef:
    refName: external-dns.tanzu.vmware.com
    versionSelection:
      constraints: 0.13.6+vmware.1-tkg.1
  values:
  - secretRef:
      name: external-dns-default-values
---
apiVersion: v1
kind: Secret
metadata:
  name: external-dns-reg-creds
  namespace: tanzu-system-service-discovery
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-service-discovery
    dns:
      deployment:
        args:
        - --txt-owner-id=k8s
        - --provider=rfc2136
        - --rfc2136-host=198.201.49.227 #! IP of compatible DNS server
        - --rfc2136-port=53
        - --rfc2136-zone=mk8s.example.org #! zone where services are deployed
        - --rfc2136-tsig-secret=O0DhTJzZ0GjfuQmB9TBc1ELchv5oDMTlQs3NNOdMZJU= #! TSIG secret
authorized to update DNS
        - --rfc2136-tsig-secret-alg=hmac-sha256
        - --rfc2136-tsig-keyname=externaldns-key
        - --rfc2136-tsig-axfr
        - --source=service
        - --source=ingress
        - --domain-filter=k8s.example.org1 #! zone where services are deployed
```

다음 예는 AWS DNS 제공자(경로 53)에 사용할 수 있습니다. 필요에 따라 패키지 버전을 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
```

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: 0.13.6+vmware.1-tkg.1
  values:
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! configure external-dns to read Contour HTTPProxy
resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --provider=aws
        - --policy=upsert-only #! prevent deleting any records, omit to enable full sync
        - --aws-zone-type=public #! only look at public hosted zones (public, private, no
value for both)
        - --aws-prefer-cname
        - --registry=txt
        - --txt-owner-id=ROUTE_53_HOSTED_ZONE_ID #! Route53 hosted zone identifier for my-
zone.example.org
        - --txt-prefix=txt #! disambiguates TXT records from CNAME records
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
```

```
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_access_key_id
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_secret_access_key
```

다음 예는 Azure DNS 제공자에 사용할 수 있습니다. 필요에 따라 패키지 버전을 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: 0.13.6+vmware.1-tkg.1
  values:
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-service-discovery
    dns:
      pspNames: "vmware-system-restricted"
```

```
    deployment:
      args:
      - --provider=azure
      - --source=service
      - --source=ingress
      - --source=contour-httpproxy #! read Contour HTTPProxy resources
      - --domain-filter=my-zone.example.org #! zone where services are deployed
      - --azure-resource-group=my-resource-group #! Azure resource group
      volumeMounts:
      - name: azure-config-file
        mountPath: /etc/kubernetes
        readOnly: true
      #@overlay/replace
      volumes:
      - name: azure-config-file
        secret:
          secretName: azure-config-file
```

vSphere 7.x용 TKr에 Fluent Bit 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 Fluent Bit를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

Fluent Bit 설치

로그 전달을 위한 Fluent Bit를 설치합니다.

1 저장소에서 사용 가능한 Fluent Bit 버전을 나열합니다.

```
kubectl -n tkg-system get packages | grep fluent-bit
```

2 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-logging
```

3 PSA의 네임스페이스에 레이블을 지정합니다.

```
kubectl label ns fluentbit-logging pod-security.kubernetes.io/enforce=privileged
```

또는 다음을 수행합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: fluentbit-logging
---
apiVersion: v1
```

```
kind: Namespace
metadata:
  name: fluentbit-logging
  labels: pod-security.kubernetes.io/enforce: privileged
```

4 `fluentbit.yaml`을 준비합니다.

자세한 내용은

5 환경에 맞게 `fluentbit-data-values`를 사용자 지정합니다.

구성 매개 변수는 Fluent Bit 패키지 참조 항목을 참조하십시오.

6 Fluent Bit를 설치합니다.

```
kubectl apply -f fluentbit.yaml
```

7 Fluent Bit 설치를 확인합니다.

```
kubectl get all -n fluentbit-logging
```

fluentbit.yaml

다음 예는 Syslog 끝점에 사용할 수 있습니다. 필요에 따라 패키지 버전을 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentbit-sa
  namespace: tkg-system
  annotations:
    pod-security.kubernetes.io/enforce: "privileged"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: fluentbit-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: fluentbit-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: fluentbit
  namespace: tkg-system
spec:
  serviceAccountName: fluentbit-sa
  packageRef:
```

```
    refName: fluent-bit.tanzu.vmware.com
    versionSelection:
      constraints: 2.1.6+vmware.1-tkg.2 #PKG_VERSION
  values:
  - secretRef:
      name: fluentbit-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: fluentbit-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-logging
    tkg:
      instance_name: "guest-cluster"     #TKG_INSTANCE_NAME
      cluster_name: "tkgs-vc-wl"         #TKG_CLUSTER_NAME
    fluentbit:
      output_plugin: "syslog"
      syslog:
        host: "10.202.27.235"           #SYSLOG_HOST
        port: "514"                     #SYSLOG_PORT
        mode: "tcp"                     #SYSLOG_MODE
        format: "rfc5424"               #SYSLOG_FORMAT
```

vSphere 7.x용 TKr에 Prometheus 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 Prometheus를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

Prometheus 설치

Altertmanager를 사용하여 Prometheus를 설치합니다.

1 저장소에서 사용 가능한 Prometheus 패키지 버전을 나열합니다.

```
kubectl get packages -n tkg-system | grep prometheus
```

2 Prometheus 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-monitoring
```

3 Prometheus 네임스페이스에서 PSA를 구성합니다.

```
kubectl label ns prometheus-monitoring pod-security.kubernetes.io/enforce=privileged
```

```
kubectl get ns prometheus-monitoring -oyaml|grep privileged
```

4 `prometheus-data-values.yaml` 파일을 생성합니다.

참조:

5 `prometheus-data-values.yaml`을 입력으로 사용하여 암호를 생성합니다.

참고 `prometheus-data-values`는 크기가 크기 때문에 Prometheus YAML 규격에 포함시키는 것보다 별도로 암호를 생성하는 것이 오류 발생률이 낮습니다.

```
kubectl create secret generic prometheus-data-values --from-file=values.yaml=prometheus-data-values.yaml -n tkg-system
```

```
secret/prometheus-data-values created
```

6 암호를 확인합니다.

```
kubectl get secrets -A
```

```
kubectl describe secret prometheus-data-values -n tkg-system
```

7 필요한 경우 환경에 맞게 `prometheus-data-values`를 사용자 지정합니다.

Prometheus 구성의 내용을 참조하십시오.

`prometheus-data-values.yaml`을 업데이트하는 경우 이 명령을 사용하여 암호를 바꿉니다.

```
kubectl create secret generic prometheus-data-values --from-file=values.yaml=prometheus-data-values.yaml -n tkg-system -o yaml --dry-run=client | kubectl replace -f-
```

```
secret/prometheus-data-values replaced
```

8 `prometheus.yaml` 규격을 생성합니다.

vSphere 7.x용 TKr에 Prometheus 설치의 내용을 참조하십시오.

9 Prometheus를 설치합니다.

```
kubectl apply -f prometheus.yaml
```

```
serviceaccount/prometheus-sa created
clusterrolebinding.rbac.authorization.k8s.io/prometheus-role-binding created
packageinstall.packaging.carvel.dev/prometheus created
```

10 Prometheus 패키지 설치를 확인합니다.

```
kubectl get pkgi -A
```

11 Prometheus 개체를 확인합니다.

```
kubectl get all -n  tanzu-system-monitoring
```

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/alertmanager-757ffd8c6c-97kqd                   1/1     Running   0          87s
pod/prometheus-kube-state-metrics-67b965c5d8-8mf4k   1/1     Running   0          87s
pod/prometheus-node-exporter-4spk9                  1/1     Running   0          87s
pod/prometheus-node-exporter-6k2rh                  1/1     Running   0          87s
pod/prometheus-node-exporter-7z9s8                  1/1     Running   0          87s
pod/prometheus-node-exporter-9d6ss                  1/1     Running   0          87s
pod/prometheus-node-exporter-csbwc                  1/1     Running   0          87s
pod/prometheus-node-exporter-qdb72                  1/1     Running   0          87s
pod/prometheus-pushgateway-dff459565-wfrz5          1/1     Running   0          86s
pod/prometheus-server-56c68567f-bjcn5               2/2     Running   0          87s

NAME                                    TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)         AGE
service/alertmanager                    ClusterIP   10.109.54.17     <none>
80/TCP          88s
service/prometheus-kube-state-metrics   ClusterIP   None             <none>        80/
TCP,81/TCP   88s
service/prometheus-node-exporter        ClusterIP   10.104.132.133   <none>
9100/TCP        88s
service/prometheus-pushgateway          ClusterIP   10.109.80.171    <none>
9091/TCP        88s
service/prometheus-server               ClusterIP   10.103.252.220   <none>
80/TCP          87s

NAME                                      DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/prometheus-node-exporter   6         6         6       6
6           <none>          88s

NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/alertmanager                   1/1     1            1           88s
deployment.apps/prometheus-kube-state-metrics  1/1     1            1           88s
deployment.apps/prometheus-pushgateway         1/1     1            1           87s
deployment.apps/prometheus-server              1/1     1            1           88s

NAME                                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/alertmanager-757ffd8c6c                   1         1         1       88s
replicaset.apps/prometheus-kube-state-metrics-67b965c5d8  1         1         1       88s
replicaset.apps/prometheus-pushgateway-dff459565          1         1         1       87s
replicaset.apps/prometheus-server-56c68567f               1         1         1       88s
```

12 Prometheus PVC를 확인합니다.

```
kubectl get pvc -n tanzu-system-monitoring
```

```
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS             AGE
alertmanager        Bound    pvc-5781956b-abc4-4646-b54c-a3eda1bf140c   2Gi
RWO            vsphere-default-policy   53m
prometheus-server   Bound    pvc-9d45d7cb-6754-40a6-a4b6-f47cf6c949a9   20Gi
RWO            vsphere-default-policy   53m
```

Prometheus 대시보드 액세스

Prometheus가 설치되면 다음 단계를 완료하여 Prometheus 대시보드에 액세스합니다.

1 `prometheus-data-values.yaml` 파일의 `ingress` 섹션이 모든 필수 필드로 채워져 있는지 확인합니다.

```
ingress:
  enabled: true
  virtual_host_fqdn: "prometheus.system.tanzu"
  prometheus_prefix: "/"
  alertmanager_prefix: "/alertmanager/"
  prometheusServicePort: 80
  alertmanagerServicePort: 80
  #! [Optional] The certificate for the ingress if you want to use your own TLS
certificate.
  #! We will issue the certificate by cert-manager when it's empty.
  tlsCertificate:
    #! [Required] the certificate
    tls.crt:
    #! [Required] the private key
    tls.key:
    #! [Optional] the CA certificate
    ca.crt:
```

2 엔보이 로드 밸런서를 사용하는 Contour의 공용(외부) IP 주소를 가져옵니다.

```
kubectl -n tanzu-system-ingress get all
```

3 Prometheus 웹 인터페이스를 시작합니다.

```
kubectl get httpproxy -n tanzu-system-monitoring
```

FQDN은 엔보이 서비스의 공용 IP 주소에서 사용할 수 있어야 합니다.

```
NAME                   FQDN                       TLS SECRET       STATUS   STATUS
DESCRIPTION
prometheus-httpproxy   prometheus.system.tanzu   prometheus-tls   valid    Valid  HTTPProxy
```

4 Prometheus FQDN을 엔보이 로드 밸런서의 외부 IP 주소에 매핑하는 DNS 레코드를 생성합니다.

5 브라우저를 사용하여 Prometheus FQDN으로 이동하여 Prometheus 대시보드에 액세스합니다.

prometheus-data-values.yaml

```
alertmanager:
  config:
    alertmanager_yml: "global: {}\nreceivers:\n- name: default-receiver\ntemplates:\n\
      - '/etc/alertmanager/templates/*.tmpl'\nroute:\n  group_interval: 5m\n  group_wait:\
      \ 10s\n  receiver: default-receiver\n  repeat_interval: 3h\n"
  deployment:
    containers:
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
    rollingUpdate:
      maxSurge: null
      maxUnavailable: null
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    annotations: {}
    storage: 2Gi
    storageClassName: wcpglobalstorageprofile
  service:
    annotations: {}
    labels: {}
    port: 80
    targetPort: 9093
    type: ClusterIP
ingress:
  alertmanagerServicePort: 80
  alertmanager_prefix: /alertmanager/
  enabled: false
  prometheusServicePort: 80
  prometheus_prefix: /
  tlsCertificate:
    ca.crt: null
    tls.crt: null
    tls.key: null
  virtual_host_fqdn: prometheus.system.tanzu
kube_state_metrics:
  deployment:
    containers:
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
  service:
    annotations: {}
    labels: {}
    port: 80
    targetPort: 8080
    telemetryPort: 81
    telemetryTargetPort: 8081
    type: ClusterIP
namespace: tanzu-system-monitoring
```

```
node_exporter:
  daemonset:
    containers:
      resources: {}
    hostNetwork: false
    podAnnotations: {}
    podLabels: {}
    updatestrategy: RollingUpdate
  service:
    annotations: {}
    labels: {}
    port: 9100
    targetPort: 9100
    type: ClusterIP
prometheus:
  config:
    alerting_rules_yml: '{}

      '
    alerts_yml: '{}

      '
    prometheus_yml: "global:\n  evaluation_interval: 1m\n  scrape_interval: 1m\n \
      \ scrape_timeout: 10s\nrule_files:\n- /etc/config/alerting_rules.yml\n- /etc/config/
recording_rules.yml\n\
      - /etc/config/alerts\n- /etc/config/rules\nscrape_configs:\n- job_name: 'prometheus'\n\
      \  scrape_interval: 5s\n  static_configs:\n  - targets: ['localhost:9090']\n\
      - job_name: 'kube-state-metrics'\n  static_configs:\n  - targets: ['prometheus-kube-
state-metrics.tanzu-system-monitoring.svc.cluster.local:8080']\n\
      \n- job_name: 'node-exporter'\n  static_configs:\n  - targets: ['prometheus-node-
exporter.tanzu-system-monitoring.svc.cluster.local:9100']\n\
      \n- job_name: 'kubernetes-pods'\n  kubernetes_sd_configs:\n  - role: pod\n \
      \ relabel_configs:\n  - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_scrape]\n\
      \    action: keep\n    regex: true\n  - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]\n\
      \    action: replace\n    target_label: __metrics_path__\n    regex: (.+)\n\
      \  - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
\n\
      \    action: replace\n    regex: ([^:]+)(?::\\d+)?;(\\d+)\n    replacement:\
      \ $1:$2\n    target_label: __address__\n  - action: labelmap\n    regex:
__meta_kubernetes_pod_label_(.+)\n\
      \  - source_labels: [__meta_kubernetes_namespace]\n    action: replace\n   \
      \ target_label: kubernetes_namespace\n  - source_labels: [__meta_kubernetes_pod_name]\n\
      \    action: replace\n    target_label: kubernetes_pod_name\n- job_name: kubernetes-
nodes-cadvisor\n\
      \  kubernetes_sd_configs:\n  - role: node\n  relabel_configs:\n  - action: labelmap\n\
      \    regex: __meta_kubernetes_node_label_(.+)\n  - replacement:
kubernetes.default.svc:443\n\
      \    target_label: __address__\n  - regex: (.+)\n    replacement: /api/v1/nodes/$1/
proxy/metrics/cadvisor\n\
      \    source_labels:\n    - __meta_kubernetes_node_name\n    target_label:
__metrics_path__\n\
      \  scheme: https\n  tls_config:\n    ca_file: /var/run/secrets/kubernetes.io/
serviceaccount/ca.crt\n\
```

```
      \    insecure_skip_verify: true\n  bearer_token_file: /var/run/secrets/kubernetes.io/
serviceaccount/token\n\
      - job_name: kubernetes-apiservers\n  kubernetes_sd_configs:\n  - role: endpoints\n\
      \  relabel_configs:\n  - action: keep\n    regex: default;kubernetes;https\n\
      \    source_labels:\n    - __meta_kubernetes_namespace\n    -
__meta_kubernetes_service_name\n\
      \    - __meta_kubernetes_endpoint_port_name\n  scheme: https\n  tls_config:\n\
      \    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt\n
insecure_skip_verify:\
      \ true\n  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token\n\
      alerting:\n  alertmanagers:\n  - scheme: http\n    static_configs:\n    - targets:\n\
      \      - alertmanager.tanzu-system-monitoring.svc:80\n  - kubernetes_sd_configs:\n\
      \      - role: pod\n    relabel_configs:\n    - source_labels:
[__meta_kubernetes_namespace]\n\
      \      regex: default\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_label_app]\n\
      \      regex: prometheus\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_label_component]\n\
      \      regex: alertmanager\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_probe]\n\
      \      regex: .*\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_container_port_number]\n\
      \      regex:\n      action: drop\n"
    recording_rules_yml: "groups:\n  - name: kube-apiserver.rules\n    interval: 3m\n\
      \    rules:\n    - expr: |2\n        (\n          (\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[1d]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[1d]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[1d]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[1d]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[1d]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[1d]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate1d\n    - expr: |2\n        (\n          (\n   \
      \     # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[1h]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[1h]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[1h]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[1h]))\n\
      \            )\n          )\n          +\n          # errors\n
```

```
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[1h]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[1h]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate1h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[2h]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[2h]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[2h]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[2h]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[2h]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[2h]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate2h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[30m]))\n            -\n        \
      \    (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[30m]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30m]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[30m]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[30m]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[30m]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate30m\n    - expr: |2\n        (\n          (\n  \
      \          # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[3d]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[3d]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[3d]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[3d]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
```

```
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[3d]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[3d]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate3d\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[5m]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[5m]))\n\
      \               or\n               vector(0)\n             )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[5m]))\n\
      \             +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[5m]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[5m]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[5m]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate5m\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[6h]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[6h]))\n\
      \               or\n               vector(0)\n             )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[6h]))\n\
      \             +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[6h]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[6h]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[6h]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate6h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[1d]))\n          \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[1d]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[1d]))\n        )\n        /\n       \
      \ \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[1d]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate1d\n\
      \    - expr: |2\n        (\n          (\n            # too slow\n          \
      \  \  sum(rate(apiserver_request_duration_seconds_count{job=\"kubernetes-apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[1h]))\n            -\n
```

```
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[1h]))\n  \
      \        )\n          +\n          sum(rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",code=~\"5..\"}[1h]))\n        )\n        /\n\
      \        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\"}[1h]))\n      labels:\n        verb: write\n      record:\
      \ apiserver_request:burnrate1h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[2h]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[2h]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[2h]))\n        )\n        /\n       \
      \ \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[2h]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate2h\n\
      \    - expr: |2\n        (\n          (\n            # too slow\n          \
      \  \
      \  sum(rate(apiserver_request_duration_seconds_count{job=\"kubernetes-apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[30m]))\n            -\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[30m]))\n \
      \         )\n          +\n          sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",code=~\"5..\"}[30m]))\n\
      \        )\n        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[30m]))\n      labels:\n        verb: write\n\
      \      record: apiserver_request:burnrate30m\n    - expr: |2\n        (\n  \
      \      (\n            # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[3d]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[3d]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[3d]))\n        )\n        /\n       \
      \ \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[3d]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate3d\n\
      \    - expr: |2\n        (\n          (\n            # too slow\n          \
      \  \
      \  sum(rate(apiserver_request_duration_seconds_count{job=\"kubernetes-apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[5m]))\n            -\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[5m]))\n  \
      \        )\n          +\n          sum(rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",code=~\"5..\"}[5m]))\n        )\n        /\n\
      \        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\"}[5m]))\n      labels:\n        verb: write\n      record:\
      \ apiserver_request:burnrate5m\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
```

```
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[6h]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[6h]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[6h]))\n        )\n        /\n       \
      \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[6h]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate6h\n\
      \    - expr: |\n        sum by (code,resource) (rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[5m]))\n      labels:\n        verb:\
      \ read\n      record: code_resource:apiserver_request_total:rate5m\n    - expr:\
      \ |\n        sum by (code,resource) (rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[5m]))\n      labels:\n        verb: write\n\
      \      record: code_resource:apiserver_request_total:rate5m\n    - expr: |\n\
      \        histogram_quantile(0.99, sum by (le, resource)
(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[5m]))) > 0\n      labels:\n    \
      \    quantile: \"0.99\"\n        verb: read\n      record:
cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |\n        histogram_quantile(0.99, sum by (le, resource)
(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[5m]))) > 0\n      labels:\n\
      \        quantile: \"0.99\"\n        verb: write\n      record:
cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |2\n        sum(rate(apiserver_request_duration_seconds_sum{subresource!
=\"\
      log\",verb!~\"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT\"}[5m]))
without(instance,\
      \ pod)\n        /\n
sum(rate(apiserver_request_duration_seconds_count{subresource!=\"\
      log\",verb!~\"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT\"}[5m]))
without(instance,\
      \ pod)\n      record: cluster:apiserver_request_duration_seconds:mean5m\n  \
      \  - expr: |\n        histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",subresource!=\"log\",verb!~\"LIST|WATCH|WATCHLIST|
DELETECOLLECTION|PROXY|CONNECT\"\
      }[5m])) without(instance, pod))\n      labels:\n        quantile: \"0.99\"\n\
      \      record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |\n        histogram_quantile(0.9,
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",subresource!=\"log\",verb!~\"LIST|WATCH|WATCHLIST|
DELETECOLLECTION|PROXY|CONNECT\"\
      }[5m])) without(instance, pod))\n      labels:\n        quantile: \"0.9\"\n\
      \      record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |\n        histogram_quantile(0.5,
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",subresource!=\"log\",verb!~\"LIST|WATCH|WATCHLIST|
DELETECOLLECTION|PROXY|CONNECT\"\
      }[5m])) without(instance, pod))\n      labels:\n        quantile: \"0.5\"\n\
      \      record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
```

```
      \  - interval: 3m\n    name: kube-apiserver-availability.rules\n    rules:\n\
      \    - expr: |2\n        1 - (\n          (\n            # write too slow\n\
      \            sum(increase(apiserver_request_duration_seconds_count{verb=~\"\
      POST|PUT|PATCH|DELETE\"}[30d]))\n            -\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      POST|PUT|PATCH|DELETE\",le=\"1\"}[30d]))\n          ) +\n          (\n     \
      \       # read too slow\n
sum(increase(apiserver_request_duration_seconds_count{verb=~\"\
      LIST|GET\"}[30d]))\n            -\n            (\n              (\n        \
      \        sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"LIST|GET\"\
      ,scope=~\"resource|\",le=\"0.1\"}[30d]))\n                or\n             \
      \   vector(0)\n              )\n             +\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30d]))\n             +\n       \
      \        sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"LIST|GET\"\
      ,scope=\"cluster\",le=\"5\"}[30d]))\n            )\n          ) +\n        \
      \  # errors\n          sum(code:apiserver_request_total:increase30d{code=~\"\
      5..\"} or vector(0))\n        )\n        /\n
sum(code:apiserver_request_total:increase30d)\n\
      \      labels:\n        verb: all\n      record: apiserver_request:availability30d\n\
      \    - expr: |2\n        1 - (\n
sum(increase(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[30d]))\n          -\n          (\n\
      \            # too slow\n            (\n
sum(increase(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[30d]))\n\
      \              or\n              vector(0)\n            )\n            +\n \
      \           sum(increase(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30d]))\n            +\n\
      \            sum(increase(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[30d]))\n          )\n      \
      \    +\n          # errors\n
sum(code:apiserver_request_total:increase30d{verb=\"\
      read\",code=~\"5..\"} or vector(0))\n        )\n        /\n
sum(code:apiserver_request_total:increase30d{verb=\"\
      read\"})\n      labels:\n        verb: read\n      record:
apiserver_request:availability30d\n\
      \    - expr: |2\n        1 - (\n          (\n            # too slow\n      \
      \      sum(increase(apiserver_request_duration_seconds_count{verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[30d]))\n            -\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      POST|PUT|PATCH|DELETE\",le=\"1\"}[30d]))\n          )\n          +\n       \
      \    # errors\n          sum(code:apiserver_request_total:increase30d{verb=\"\
      write\",code=~\"5..\"} or vector(0))\n        )\n        /\n
sum(code:apiserver_request_total:increase30d{verb=\"\
      write\"})\n      labels:\n        verb: write\n      record:
apiserver_request:availability30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"2..\"}[30d]))\n      record:
```

```
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"2..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"3..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"4..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
```

```
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"5..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code)
(code_verb:apiserver_request_total:increase30d{verb=~\"\
      LIST|GET\"})\n      labels:\n        verb: read\n      record:
code:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code)
(code_verb:apiserver_request_total:increase30d{verb=~\"\
      POST|PUT|PATCH|DELETE\"})\n      labels:\n        verb: write\n      record:\
      \ code:apiserver_request_total:increase30d\n"
    rules_yml: '{}

      '
  deployment:
    configmapReload:
      containers:
        args:
        - --volume-dir=/etc/config
        - --webhook-url=http://127.0.0.1:9090/-/reload
        resources: {}
    containers:
      args:
      - --storage.tsdb.retention.time=42d
      - --config.file=/etc/config/prometheus.yml
      - --storage.tsdb.path=/data
      - --web.console.libraries=/etc/prometheus/console_libraries2
      - --web.console.templates=/etc/prometheus/consoles
      - --web.enable-lifecycle
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
    rollingUpdate:
      maxSurge: null
      maxUnavailable: null
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    annotations: {}
    storage: 150Gi
    storageClassName: wcpglobalstorageprofile
  service:
    annotations: {}
    labels: {}
    port: 80
```

```
    targetPort: 9090
    type: ClusterIP
pushgateway:
  deployment:
    containers:
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
  service:
    annotations: {}
    labels: {}
    port: 9091
    targetPort: 9091
    type: ClusterIP
```

prometheus.yaml

`prometheus.yaml` 규격은 `prometheus-data-values` 암호를 참조합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus-sa
  namespace: tkg-system

---
# temp
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: prometheus-sa
    namespace: tkg-system

---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: prometheus
  namespace: tkg-system
spec:
  serviceAccountName: prometheus-sa
  packageRef:
    refName: prometheus.tanzu.vmware.com
    versionSelection:
```

```
      constraints: 2.45.0+vmware.1-tkg.2
  values:
  - secretRef:
      name: prometheus-data-values
```

vSphere 7.x용 TKr에 Grafana 설치

vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 Grafana를 설치하려면 다음 지침을 참조하십시오.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

Grafana 설치

Grafana를 설치합니다.

1 저장소에서 사용 가능한 Grafana 버전을 나열합니다.

```
kubectl get packages -n tkg-system | grep grafana
```

2 Grafana 네임스페이스를 생성합니다.

```
kubectl create ns tanzu-system-dashboards
```

3 네임스페이스에 대한 PSA 레이블을 생성합니다.

```
kubectl label namespace tanzu-system-dashboards pod-security.kubernetes.io/
enforce=privileged
```

4 또는 파일 `ns-grafana-dashboard.yaml`을 사용하여 Grafana 네임스페이스 및 레이블을 선언적으로 생성합니다.

```
apiVersion:
v1kind: Namespace
metadata:
  name: grafana-dashboard
---
apiVersion: v1
kind: Namespace
metadata:
  name: tanzu-system-dashboards
  labels:
    pod-security.kubernetes.io/enforce: privileged
```

5 `grafana-data-values.yaml`을 생성합니다.

Grafana 패키지 참조의 내용을 참조하십시오.

6 `grafana-data-values.yaml` 파일을 입력으로 사용하여 암호를 생성합니다.

```
kubectl create secret generic grafana-data-values --from-file=values.yaml=grafana-data-values.yaml -n tkg-system
```

```
secret/grafana-data-values created
```

7 암호를 확인합니다.

```
kubectl get secrets -A
```

```
kubectl describe secret grafana-data-values -n tkg-system
```

8 필요한 경우 환경에 맞게 `grafana-data-values`를 사용자 지정합니다.

Grafana 패키지 참조 항목을 참조하십시오.

데이터 값을 업데이트하는 경우 다음 명령을 사용하여 암호를 업데이트합니다.

```
kubectl create secret generic grafana-data-values --from-file=values.yaml=grafana-data-values.yaml -n tkg-system -o yaml --dry-run=client | kubectl replace -f-
```

```
secret/grafana-data-values replaced
```

9 `grafana.yaml` 규격을 생성합니다.

vSphere 7.x용 TKr에 Grafana 설치의 내용을 참조하십시오.

10 Grafana를 설치합니다.

```
kubectl apply -f grafana.yaml
```

```
serviceaccount/grafana-sa created
clusterrolebinding.rbac.authorization.k8s.io/grafana-role-binding created
packageinstall.packaging.carvel.dev/grafana created
```

11 Grafana 패키지 설치를 확인합니다.

```
kubectl get pkgi -A | grep grafana
```

12 Grafana 개체를 확인합니다.

```
kubectl get all -n tanzu-system-dashboards
```

엔보이 LoadBalancer를 사용하여 Grafana 대시보드에 액세스

LoadBalancer 유형의 필수 Contour 엔보이 서비스가 배포되고 Grafana 구성 파일에서 이를 지정한 경우 로드 밸런서의 외부 IP 주소를 가져오고 Grafana FQDN에 대한 DNS 레코드를 생성합니다.

1 LoadBalancer 유형의 엔보이 서비스에 대한 `External-IP` 주소를 가져옵니다.

```
kubectl get service envoy -n tanzu-system-ingress
```

반환된 `External-IP` 주소가 다음과 같이 표시됩니다.

```
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
envoy   LoadBalancer   10.99.25.220   10.195.141.17   80:30437/TCP,443:30589/TCP   3h27m
```

또는 다음 명령을 사용하여 `External-IP` 주소를 가져올 수 있습니다.

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Grafana 확장 설치를 확인하려면 로드 밸런서의 `External-IP` 주소에 매핑된 Grafana FQDN으로 로컬 `/etc/hosts` 파일을 업데이트합니다. 예를 들면 다음과 같습니다.

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Grafana Extension with Envoy Load Balancer
10.195.141.17 grafana.system.tanzu
```

3 `https://grafana.system.tanzu`로 이동하여 Grafana 대시보드에 액세스합니다.

사이트는 자체 서명된 인증서를 사용하기 때문에 대시보드에 액세스하려면 먼저 브라우저별 보안 주의를 거쳐 이동해야 할 수 있습니다.

4 운영 액세스를 위해 엔보이 서비스 로드 밸런서 `External-IP` 주소를 Grafana 대시보드에 매핑하는 DNS 서버에 두 개의 CNAME 레코드를 생성합니다.

엔보이 NodePort를 사용하여 Grafana 대시보드에 액세스

NodePort 유형의 필수 Contour 엔보이 서비스가 배포되고 Grafana 구성 파일에서 이를 지정한 경우 작업자 노드의 가상 시스템 IP 주소를 가져오고 Grafana FQDN에 대한 DNS 레코드를 생성합니다.

1 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

2 클러스터의 노드를 나열합니다.

```
kubectl get virtualmachines
```

3 작업자 노드 중 하나를 선택하고 다음 명령을 사용하여 설명합니다.

```
kubectl describe virtualmachines tkgs-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 가상 시스템의 IP 주소(예: `Vm Ip: 10.115.22.43`)를 찾습니다.

5 Grafana 확장 설치를 확인하려면 작업자 노드 IP 주소에 매핑된 Grafana FQDN으로 로컬 `/etc/hosts` 파일을 업데이트합니다. 예를 들면 다음과 같습니다.

```
127.0.0.1 localhost
127.0.1.1 ubuntu
# TKG Grafana with Envoy NodePort
10.115.22.43 grafana.system.tanzu
```

6 `https://grafana.system.tanzu`로 이동하여 Grafana 대시보드에 액세스합니다.

사이트는 자체 서명된 인증서를 사용하기 때문에 대시보드에 액세스하려면 먼저 브라우저별 보안 주의를 거쳐 이동해야 할 수 있습니다.

grafana-data-values.yaml

다음 `grafana-data-values.yaml` 파일 예를 참조하십시오.

```
namespace: tanzu-system-dashboards
grafana:
  pspNames: "vmware-system-restricted"
  deployment:
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: wcpglobalstorageprofile
  secret:
    admin_user: YWRtaW4=
    admin_password: YWRtaW4=
    type: Opaque
  service:
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
```

grafana.yaml

다음 `grafana.yaml` 규격 예를 참조하십시오. 필요에 따라 패키지 버전을 업데이트합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: grafana-sa
  namespace: tkg-system
  annotations:
    pod-security.kubernetes.io/enforce: "privileged"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: grafana-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: grafana-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: grafana
  namespace: tkg-system
spec:
  serviceAccountName: grafana-sa
  packageRef:
    refName: grafana.tanzu.vmware.com
    versionSelection:
      constraints: 10.0.1+vmware.1-tkg.2 #PKG-VERSION
  values:
  - secretRef:
      name: grafana-data-values
```

vSphere 7.x용 TKr에 Harbor 설치

다음 지침에 따라 vSphere 7.x용 TKr로 프로비저닝된 TKG 클러스터에 Harbor를 설치합니다.

사전 요구 사항

vSphere 7.x용 TKr에 표준 패키지를 설치하기 위한 워크플로의 내용을 참조하십시오.

Harbor에는 HTTP/S 수신이 필요합니다. Harbor 서비스는 Contour 패키지의 엔보이 서비스를 통해 노출됩니다. 사전 요구 사항으로 Contour 패키지를 배포합니다. vSphere 7.x용 TKr에 Contour 설치의 내용을 참조하십시오.

- 감독자에 NSX 네트워킹을 사용하는 경우 LoadBalancer 유형의 엔보이 서비스를 생성합니다.

- 감독자에 vSphere vDS 네트워킹을 사용하는 경우에는 환경에 따라 LoadBalancer 또는 NodePort 유형의 엔보이 서비스를 생성합니다.

Harbor 확장에는 DNS가 필요합니다. 테스트 및 확인을 위해 Harbor 및 Notary FQDN을 로컬 `/etc/hosts` 파일에 추가합니다. 이 작업을 수행하는 방법은 아래 지침에 설명되어 있습니다.

운영 환경에서 Harbor는 로컬 DNS 서버(예: BIND) 또는 공용 클라우드(예: AWS Route53 또는 Azure DNS)에 DNS 영역이 필요합니다. DNS를 설정한 후 Harbor FQDN을 DNS 서버에 자동으로 등록하려면 ExternalDNS 확장을 설치합니다. vSphere 7.x용 TKr에 ExternalDNS 설치의 내용을 참조하십시오.

Harbor 설치

표준 패키지를 사용하여 Harbor 레지스트리를 설치하려면 다음 단계를 완료합니다.

1 저장소에서 사용 가능한 Harbor 버전을 나열합니다.

```
kubectl get packages -n tkg-system | grep harbor
```

2 `harbor.yaml` 규격을 생성합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: harbor-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: habor-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: harbor-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: harbor
  namespace: tkg-system
spec:
  serviceAccountName: harbor-sa
  packageRef:
    refName: harbor.tanzu.vmware.com
    versionSelection:
      constraints: 2.7.1+vmware.1-tkg.1 #PKG-VERSION
  values:
  - secretRef:
      name: harbor-data-values
```

```
---
apiVersion: v1
kind: Secret
metadata:
  name: harbor-data-values
  namespace: harbor-registry
stringData:
  values.yml: |
    namespace: tanzu-system-registry
    hostname: <ENTER-HARBOR-FQDN>
    port:
      https: 443
    logLevel: info
    tlsCertificate:
      tls.crt: ""
      tls.key: ""
      ca.crt:
    tlsCertificateSecretName:
    enableContourHttpProxy: true
    harborAdminPassword: <ENTER-STRONG-PASSWORD-HERE>
    secretKey: <ENTER-SECRET-KEY>
    database:
      password: <ENTER-STRONG-PASSWORD-HERE>
      shmSizeLimit:
      maxIdleConns:
      maxOpenConns:
    exporter:
      cacheDuration:
    core:
      replicas: 1
      secret: <ENTER-SECRET>
      xsrfKey: <ENTER-XSRF-KEY-WHICH-IS-AN-ALPHANUMERIC-STRING-WITH-32-CHARS>
    jobservice:
      replicas: 1
      secret: <ENTER-SECRET>
    registry:
      replicas: 1
      secret: <ENTER-SECRET>
    trivy:
      enabled: true
      replicas: 1
      gitHubToken: ""
      skipUpdate: false
    persistence:
      persistentVolumeClaim:
        registry:
          existingClaim: ""
          storageClass: "<ENTER-STORAGE-CLASS>"
          subPath: ""
          accessMode: ReadWriteOnce
          size: 50Gi
        jobservice:
          existingClaim: ""
          storageClass: "<ENTER-STORAGE-CLASS>"
          subPath: ""
```

```
          accessMode: ReadWriteOnce
          size: 10Gi
        database:
          existingClaim: ""
          storageClass: "<ENTER-STORAGE-CLASS>"
          subPath: ""
          accessMode: ReadWriteOnce
          size: 10Gi
        redis:
          existingClaim: ""
          storageClass: "<ENTER-STORAGE-CLASS>"
          subPath: ""
          accessMode: ReadWriteOnce
          size: 10Gi
        trivy:
          existingClaim: ""
          storageClass: "<ENTER-STORAGE-CLASS>"
          subPath: ""
          accessMode: ReadWriteOnce
          size: 10Gi
    proxy:
      httpProxy:
      httpsProxy:
      noProxy: 127.0.0.1,localhost,.local,.internal
    pspNames: vmware-system-restricted
    network:
      ipFamilies: ["IPv4", "IPv6"]
```

3 호스트 이름, 암호 및 스토리지 클래스를 포함하여 환경에 적합한 값으로 `harbor.yaml` 규격의 `harbor-data-values` 암호를 사용자 지정합니다.

지침은 Harbor 패키지 참조 항목을 참조하십시오.

4 Harbor를 설치합니다.

```
kubectl apply -f harbor.yaml
```

5 Harbor 설치를 확인합니다.

```
kubectl get all -n harbor-registry
```

엔보이 LoadBalancer를 사용하여 Harbor용 DNS 구성(NSX 네트워킹)

사전 요구 사항 엔보이 서비스가 LoadBalancer를 통해 노출되는 경우 로드 밸런서의 외부 IP 주소를 가져오고 Harbor FQDN에 대한 DNS 레코드를 생성합니다.

1 LoadBalancer 유형의 엔보이 서비스에 대한 `External-IP` 주소를 가져옵니다.

```
kubectl get service envoy -n tanzu-system-ingress
```

반환된 `External-IP` 주소가 다음과 같이 표시됩니다.

```
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
envoy   LoadBalancer   10.99.25.220   10.195.141.17   80:30437/TCP,443:30589/TCP   3h27m
```

또는 다음 명령을 사용하여 `External-IP` 주소를 가져올 수 있습니다.

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Harbor 확장 설치를 확인하려면 로드 밸런서의 `External-IP` 주소에 매핑된 Harbor 및 Notary FQDN으로 로컬 `/etc/hosts` 파일을 업데이트합니다. 예를 들면 다음과 같습니다.

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy Load Balancer IP
10.195.141.17 core.harbor.domain
10.195.141.17 core.notary.harbor.domain
```

3 Harbor 확장 설치를 확인하려면 Harbor에 로그인합니다.

4 엔보이 서비스 로드 밸런서 `External-IP` 주소를 Harbor FQDN 및 Notary FQDN에 매핑하는 2개의 CNAME 레코드를 DNS 서버에 생성합니다.

5 외부 DNS 확장을 설치합니다.

엔보이 NodePort를 사용하여 Harbor용 DNS 구성(vDS 네트워킹)

사전 요구 사항 엔보이 서비스가 NodePort를 통해 노출되는 경우 작업자 노드의 가상 시스템 IP 주소를 가져오고 Harbor FQDN에 대한 DNS 레코드를 생성합니다.

참고 NodePort를 사용하려면 `harbor-data-values.yaml` 파일에 올바른 `port.https` 값을 지정해야 합니다.

1 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

2 클러스터의 노드를 나열합니다.

```
kubectl get virtualmachines
```

3 작업자 노드 중 하나를 선택하고 다음 명령을 사용하여 설명합니다.

```
kubectl describe virtualmachines tkg2-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 가상 시스템의 IP 주소(예: `Vm Ip: 10.115.22.43`)를 찾습니다.

5 Harbor 확장 설치를 확인하려면 작업자 노드 IP 주소에 매핑된 Harbor 및 Notary FQDN으로 로컬 `/etc/hosts` 파일을 업데이트합니다. 예를 들면 다음과 같습니다.

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy NodePort
10.115.22.43 core.harbor.domain
10.115.22.43 core.notary.harbor.domain
```

6 Harbor 확장 설치를 확인하려면 Harbor에 로그인합니다.

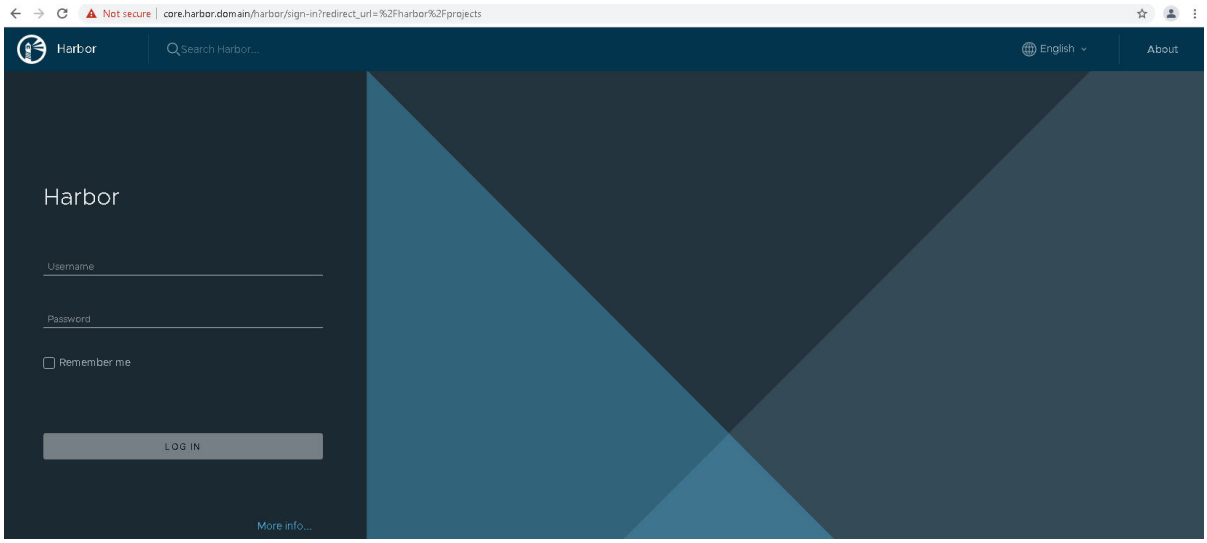
7 작업자 노드 IP 주소를 Harbor FQDN 및 Notary FQDN에 매핑하는 2개의 CNAME 레코드를 DNS 서버에 생성합니다.

8 외부 DNS 확장을 설치합니다.

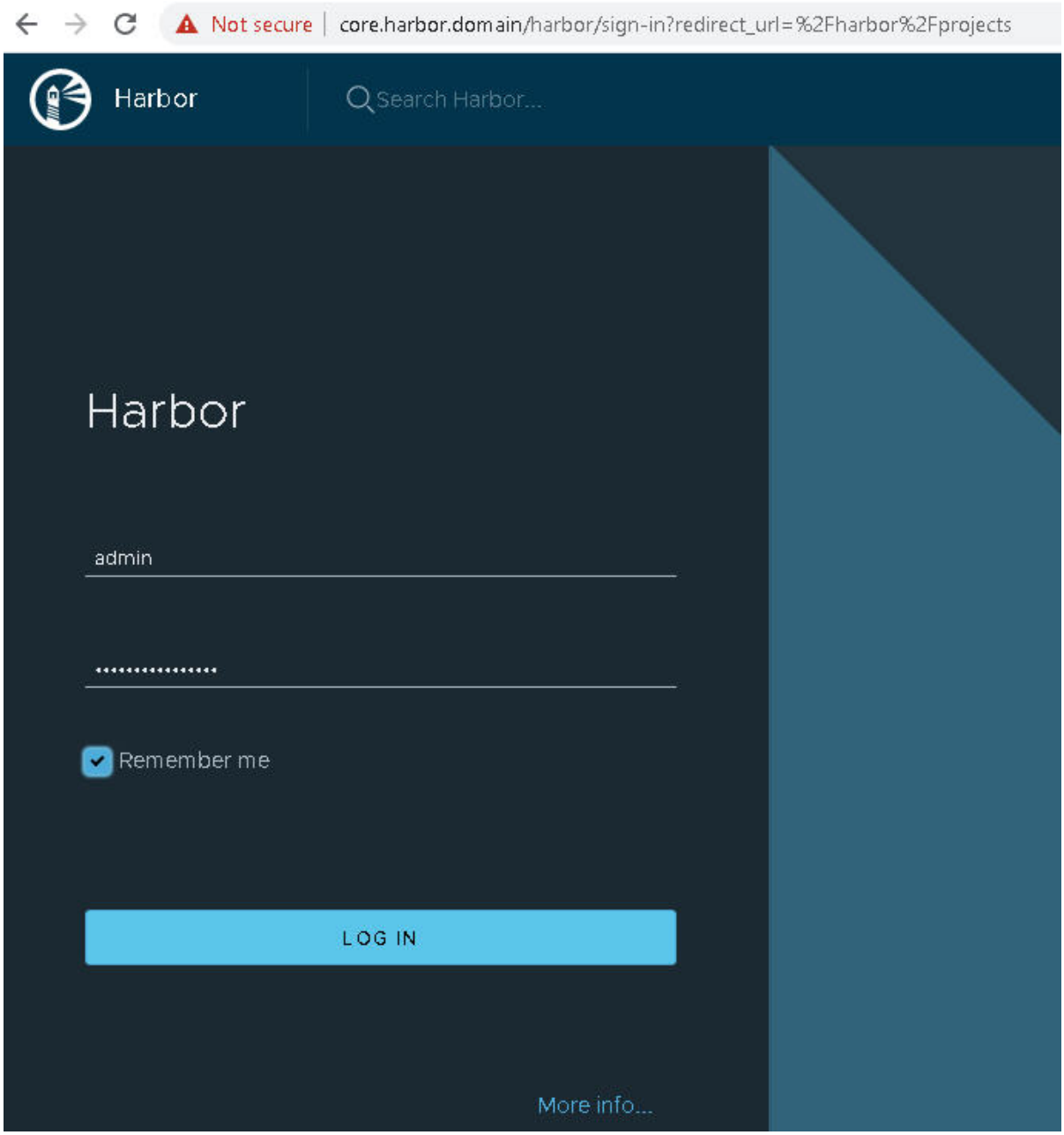
Harbor 웹 인터페이스에 로그인

Harbor가 설치 및 구성되면 로그인하여 사용을 시작합니다.

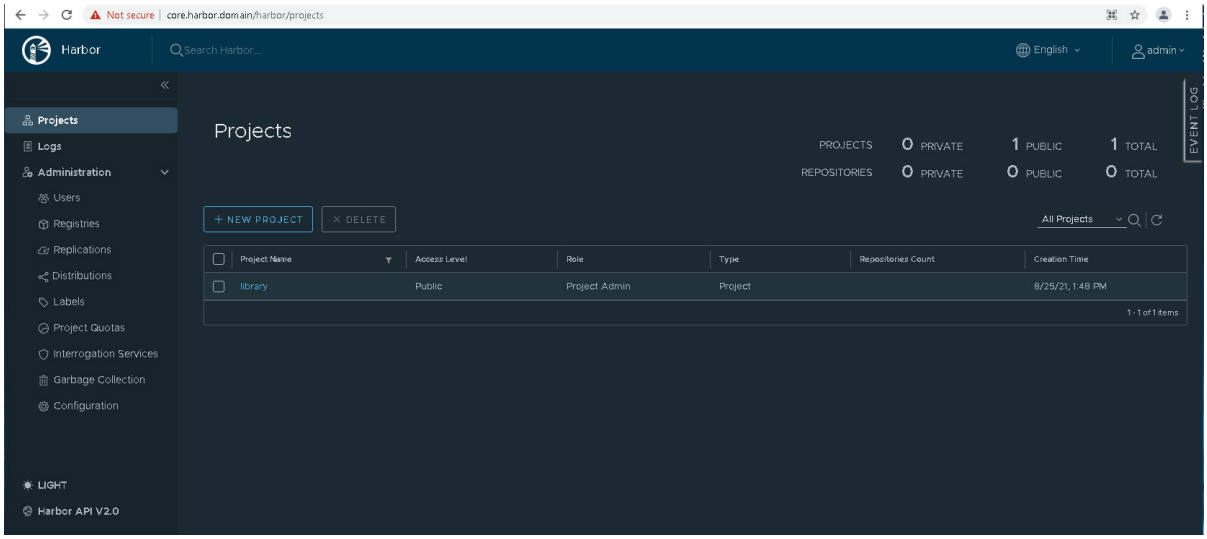
1 Harbor 레지스트리 웹 인터페이스(`https://core.harbor.domain`) 또는 사용한 호스트 이름에 액세스합니다.



2 `harbor-data-values.yaml` 파일에 입력한 사용자 이름 **`admin`**과 생성된 암호를 사용하여 Harbor에 로그인합니다.



3 Harbor 사용자 인터페이스에 액세스할 수 있는지 확인합니다.



4 Harbor CA 인증서를 가져옵니다.

Harbor 인터페이스에서 **프로젝트 > 라이브러리**를 선택하거나 **새 프로젝트**를 생성합니다.

레지스트리 인증서를 클릭하고 Harbor CA 인증서(`ca.crt`)를 다운로드합니다.

5 Harbor 레지스트리에서 컨테이너 이미지를 푸시하고 끌어올 수 있도록 Harbor CA 인증서를 Docker 클라이언트의 신뢰 저장소에 추가합니다. 장 14 TKG 서비스 클러스터에서 개인 레지스트리 사용의 내용을 참조하십시오.

6 Harbor 사용에 대한 자세한 내용은 Harbor 설명서를 참조하십시오.

TKG 서비스 클러스터에 워크로드 배포 12

포드, 서비스, 영구 볼륨 및 더 높은 수준의 리소스(예: 배포 및 복제 집합)를 사용하여 애플리케이션 워크로드를 TKG 서비스 클러스터에 배포할 수 있습니다.

다음으로 아래 항목을 읽으십시오.

- 로드 밸런서 서비스를 사용한 포드 배포
- 정적 IP를 사용하는 로드 밸런서 서비스
- Nginx를 사용한 수신
- Contour를 사용한 수신
- 영구 볼륨에 스토리지 클래스 사용
- 영구 스토리지 볼륨을 동적으로 생성
- 영구 스토리지 볼륨을 정적으로 생성
- TKG 클러스터에 방명록 애플리케이션 배포
- 방명록 애플리케이션 YAML
- 지연 바인딩 볼륨 연결로 여러 vSphere 영역에 StatefulSet 애플리케이션 배포

로드 밸런서 서비스를 사용한 포드 배포

외부 트래픽을 TKG 2.0 클러스터에서 실행되는 포드로 라우팅하려면 LoadBalancer 유형의 서비스를 생성합니다. 로드 밸런서 서비스는 인바운드 트래픽이 포드로 라우팅되는 공용 IP 주소를 노출합니다.

서비스로 노출된 Kubernetes 포드에 대한 외부 로드 밸런서를 프로비저닝할 수 있습니다. 예를 들어 Nginx 컨테이너를 배포하고 LoadBalancer 유형의 Kubernetes 서비스로 노출할 수 있습니다.

사전 요구 사항

- Kubernetes 설명서에서 LoadBalancer 서비스 유형을 검토합니다.
- TKG 클러스터를 프로비저닝합니다.
- TKG 클러스터에 연결합니다.

절차

1 다음 `nginx-lbsvc.yaml` YAML 파일을 생성합니다.

이 YAML 파일은 LoadBalancer 유형의 Kubernetes 서비스를 정의하고 서비스에 대한 외부 로드 밸런서로 Nginx 컨테이너를 배포합니다.

```
kind: Service
apiVersion: v1
metadata:
  name: srvclb-ngnx
spec:
  selector:
    app: hello
    tier: frontend
  ports:
  - protocol: "TCP"
    port: 80
    targetPort: 80
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loadbalancer
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
        tier: frontend
    spec:
      containers:
      - name: nginx
        image: "nginxdemos/hello"
```

2 YAML을 적용합니다.

```
kubectl apply -f nginx-lbsvc.yaml
```

3 Nginx 서비스의 배포를 확인합니다.

```
kubectl get services
```

`srvclb-ngnx`는 외부 및 내부 IP 주소를 사용하여 작동합니다.

```
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)        AGE
srvclb-ngnx   LoadBalancer   10.11.12.19      10.19.15.89     80:30818/TCP   18m
```

4 브라우저를 사용하여 Nginx LoadBalancer 서비스의 외부 IP 주소를 입력합니다.

로드 밸런서의 세부 정보 및 `NGINX` 배너 메시지가 보입니다.

정적 IP를 사용하는 로드 밸런서 서비스

정적 IP 주소를 사용하도록 LoadBalancer 유형의 Kubernetes 서비스를 구성할 수 있습니다. 이 기능을 구현하기 전에 최소 구성 요소 요구 사항, 중요한 보안 고려 사항 및 클러스터 강화 지침을 알고 있어야 합니다.

LoadBalancer 유형의 서비스에 정적 IP 사용

일반적으로 LoadBalancer 유형의 Kubernetes 서비스를 정의할 때는 로드 밸런서에서 할당된 사용 후 삭제 IP 주소를 얻습니다.

또는 로드 밸런서에 대한 정적 IP 주소를 지정할 수 있습니다. 서비스를 생성할 때 로드 밸런서 인스턴스는 사용자가 할당한 정적 IP 주소로 프로비저닝됩니다.

다음 예제 서비스는 정적 IP 주소로 지원되는 로드 밸런서를 구성하는 방법을 보여줍니다. 서비스 규격에는 `loadBalancerIP` 매개 변수와 IP 주소 값(이 예제의 경우 `10.11.12.49`)이 포함됩니다.

```
kind: Service
apiVersion: v1
metadata:
  name: load-balancer-service-with-static-ip
spec:
  selector:
    app: hello-world
    tier: frontend
  ports:
  - protocol: "TCP"
    port: 80
    targetPort: 80
  type: LoadBalancer
  loadBalancerIP: 10.11.12.49
```

NSX Advanced Load Balancer의 경우 로드 밸런서가 설치될 때 구성된 IPAM 풀의 IP 주소를 사용합니다. 서비스가 생성되고 정적 IP 주소가 할당되면 로드 밸런서는 이 서비스를 할당된 것으로 표시하고 사용 후 삭제 IP 주소와 동일한 방식으로 IP 주소의 수명 주기를 관리합니다. 즉, 서비스가 제거되면 IP 주소 할당이 취소되고 재할당이 가능해집니다.

NSX-T 로드 밸런서의 경우 두 가지 옵션이 있습니다. 기본 메커니즘은 NSX Advanced Load Balancer와 동일합니다. 로드 밸런서가 설치될 때 구성된 IP 풀에서 가져온 IP 주소를 사용하는 것입니다. 정적 IP 주소가 할당되면 로드 밸런서는 자동으로 할당된 것으로 표시하고 해당 수명 주기를 관리합니다.

두 번째 NSX-T 옵션은 정적 IP 주소를 수동으로 미리 할당하는 것입니다. 이 경우 로드 밸런서에 할당된 외부 로드 밸런서 IP 풀의 일부가 아닌 부동 IP 풀에서 가져온 IP 주소를 대신 사용합니다. 이 경우 NSX Manager를 사용하여 IP 주소의 할당 및 수명 주기를 수동으로 관리합니다.

중요한 보안 고려 사항 및 강화 요구 사항

이 기능을 사용할 때 주의해야 할 잠재적인 보안 문제가 있습니다. 개발자가 `Service.status.loadBalancerIP` 값에 패치를 적용할 수 있으면 패치가 적용되어 있는 IP 주소로 향하는 클러스터의 트래픽을 개발자가 강탈할 수 있습니다. 특히, `patch` 권한이 있는 Role 또는 ClusterRole이 이 기능이 구현된 클러스터의 서비스 또는 사용자 계정에 바인딩된 경우 해당 계정 소유자는 자체 자격 증명을 사용하여 `kubectl` 명령을 실행하고 로드 밸런서에 할당된 정적 IP 주소를 변경할 수 있습니다.

로드 밸런서 서비스에 정적 IP 할당을 사용할 경우 발생할 수 있는 보안 위험을 피하려면 이 기능을 구현할 각 클러스터를 강화해야 합니다. 이렇게 하려면 개발자에 대해 정의한 Role 또는 ClusterRole이 `apiGroups:""` 및 `resources: services/status`에 대한 `patch` 동사를 허용하지 않아야 합니다. 예제 역할 코드 조각은 이 기능을 구현할 때 하지 말아야 할 내용을 보여줍니다.

패치를 허용하지 안 됩니다.

```
- apiGroups:
  - ""
  resources:
  - services/status
  verbs:
  - patch
```

개발자에게 패치 권한이 있는지 확인하려면 해당 사용자로 다음 명령을 실행합니다.

```
kubectl --kubeconfig <KUBECONFIG> auth can-i patch service/status
```

명령이 `yes`를 반환하면 사용자에게 패치 권한이 있습니다. 자세한 내용은 Kubernetes 설명서에서 API 액세스 확인을 참조하십시오.

개발자에게 클러스터에 대한 액세스 권한을 부여하려면 개발자에게 TKG 서비스 클러스터에 대한 vCenter SSO 액세스 권한 부여 항목을 참조하십시오. 사용자 지정할 수 있는 샘플 역할 템플릿에 대한 자세한 내용은 TKG 서비스 클러스터에 기본 포드 보안 정책 적용 항목을 참조하십시오. 클러스터 액세스를 제한하는 방법에 대한 예는 https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-example에서 참조하십시오.

Nginx를 사용한 수신

Kubernetes 수신 리소스는 클러스터 외부에서 클러스터 내부의 하나 이상의 서비스로 HTTP 또는 HTTPS 라우팅을 제공합니다. TKG 클러스터는 Nginx와 같은 타사 컨트롤러를 통한 수신을 지원합니다.

이 자습서에서는 외부 트래픽을 Tanzu Kubernetes 클러스터의 서비스로 라우팅하기 위해 NGINX를 기반으로 Kubernetes 수신 서비스를 배포하는 방법을 보여줍니다. 수신 서비스에는 수신 컨트롤러가 필요합니다. NGINX 수신 컨트롤러는 Helm을 사용하여 설치합니다. Helm은 Kubernetes용 패키지 관리자입니다.

참고 이 작업을 수행하는 방법은 여러 가지입니다. 여기에 나와있는 단계는 한 가지 접근 방법을 제공합니다. 사용 환경에 따라 다른 방법이 더 적합할 수 있습니다.

사전 요구 사항

- Kubernetes 설명서에서 수신 리소스를 검토합니다.
- Nginx 수신 컨트롤러 설명서를 검토합니다.
- TKG 클러스터를 프로비저닝합니다.
- 필요한 경우 포드 보안 정책을 사용하도록 설정합니다.
- TKG 클러스터에 연결합니다.

절차

1 설명서를 참조하여 Helm을 설치합니다.

2 Helm을 사용하여 NGINX 수신 컨트롤러를 설치합니다.

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm install ingress-nginx ingress-nginx/ingress-nginx
```

3 Nginx 수신 컨트롤러가 LoadBalancer 유형의 서비스로 배포되었는지 확인합니다.

```
kubectl get services
```

```
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP
PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.16.18.20    10.19.14.76   80:30635/
TCP,443:30873/TCP   59m
ingress-nginx-controller-admission   ClusterIP      10.87.41.25    <none>
443/TCP                      59m
```

4 외부 IP 주소를 사용하여 로드 밸런서를 Ping합니다.

```
ping 10.19.14.76
```

```
Pinging 10.19.14.76 with 32 bytes of data:
Reply from 10.19.14.76: bytes=32 time<1ms TTL=62
Reply from 10.19.14.76: bytes=32 time=1ms TTL=62
```

5 Nginx 수신 컨트롤러가 실행 중인지 확인합니다.

```
kubectl get pods
```

```
NAME                                        READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-7c6c46898c-v6blt   1/1     Running   0          76m
```

6 수신 규칙 및 경로를 사용하여 `ingress-hello.yaml`이라는 수신 리소스를 생성합니다.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
```

```
metadata:
  name: ingress-hello
spec:
  rules:
  - http:
      paths:
      - path: /hello
        backend:
          serviceName: hello
          servicePort: 80
```

주의 Kubernetes API `networking.k8s.io/v1beta1`은 Kubernetes v1.22부터 더 이상 사용되지 않습니다. 사용할 새 API는 `networking.k8s.io/v1`이며. 여기 https://kubernetes.io/docs/reference/using-api/deprecation-guide/에 설명된 몇 가지 매니페스트 변경 사항이 포함됩니다.

7 `ingress-hello` 리소스를 배포합니다.

```
kubectl apply -f ingress-hello.yaml
```

```
ingress.networking.k8s.io/ingress-hello created
```

8 수신 리소스가 배포되었는지 확인합니다.

IP 주소는 수신 컨트롤러의 외부 IP에 매핑됩니다.

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-hello   <none>   *       10.19.14.76     80      51m
```

9 `ingress-hello-test.yaml`이라는 hello 테스트 애플리케이션 및 서비스를 생성합니다.

```
kind: Service
apiVersion: v1
metadata:
  name: hello
spec:
  selector:
    app: hello
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      app: hello
      tier: backend
      track: stable
  template:
    metadata:
      labels:
        app: hello
        tier: backend
        track: stable
    spec:
      containers:
        - name: hello
          image: "gcr.io/google-samples/hello-go-gke:1.0"
          ports:
            - name: http
              containerPort: 80
```

10 `ingress-hello-test` 리소스를 배포합니다.

```
kubectl apply -f ingress-hello-test.yaml
```

```
service/hello created
deployment.apps/hello created
```

11 `hello` 배포를 사용할 수 있는지 확인합니다.

```
kubectl get deployments
```

```
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
hello                      3/3     3            3           4m59s
ingress-nginx-controller   1/1     1            1           3h39m
```

12 Nginx 수신 컨트롤러에서 사용하는 로드 밸런서의 공용 IP 주소를 가져옵니다.

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS          PORTS   AGE
ingress-hello   <none>   *       10.19.14.76      80      13m
```

13 브라우저를 사용하여 공용 IP로 이동하고 수신 경로를 포함합니다.

```
http://10.19.14.76/hello
```

"hello" 메시지가 반환됩니다.

```
{"message":"Hello"}
```

결과

클러스터 내에서 실행되는 서비스가 앞에 있는 백엔드 앱은 로드 밸런서의 외부 IP 주소를 사용하여 수신 컨트롤러를 통해 외부에서 브라우저로 액세스합니다.

Contour를 사용한 수신

Kubernetes 수신 리소스는 클러스터 외부에서 클러스터 내부의 하나 이상의 서비스로 HTTP 또는 HTTPS 라우팅을 제공합니다. TKG 클러스터는 Contour와 같은 타사 컨트롤러를 통한 수신을 지원합니다.

이 자습서에서는 외부 트래픽을 TKG 클러스터의 서비스로 라우팅하기 위해 Contour 수신 컨트롤러를 배포하는 방법을 보여줍니다. Contour는 VMware에서 기여하는 오픈 소스 프로젝트입니다.

사전 요구 사항

- Kubernetes 설명서에서 수신 리소스를 검토합니다.
- Contour 수신 컨트롤러를 검토합니다.
- TKG 클러스터를 프로비저닝합니다.
- TKG 클러스터에 연결합니다.

절차

1 서비스 계정을 통해 클러스터의 모든 리소스를 관리할 수 있도록 ClusterRoleBinding을 생성합니다.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding
--clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

참고 보안을 강화해야 하는 경우에는 `projectcontour` 네임스페이스에 RoleBinding을 사용합니다. TKG 서비스 클러스터에 기본 포드 보안 정책 적용의 내용을 참조하십시오.

2 `projectcontour`라는 네임스페이스를 생성합니다.

이것은 Contour 수신 컨트롤러 배포의 기본 네임스페이스입니다.

```
kubectl create ns projectcontour
```

3 Contour 수신 컨트롤러 YAML Contour 수신 배포를 다운로드합니다.

4 텍스트 편집기를 사용하여 `contour.yaml` 파일을 엽니다.

5 다음 두 줄을 주석 처리합니다. 각 줄 앞에 # 기호를 추가하면 됩니다.

줄 1632:

```
# service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
```

줄 1634:

```
# externalTrafficPolicy: Local
```

6 `contour.yaml` 파일을 적용하여 Contour를 배포합니다.

```
kubectl apply -f contour.yaml
```

7 Contour 수신 컨트롤러 및 엔보이 로드 밸런서 서비스가 배포되었는지 확인합니다.

```
kubectl get services -n projectcontour
```

```
NAME      TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)                      AGE
contour   ClusterIP      198.63.146.166   <none>          8001/TCP                     120m
envoy     LoadBalancer   198.48.52.47     192.168.123.5   80:30501/TCP,443:30173/TCP   120m
```

8 Contour 및 엔보이 포드가 실행 중인지 확인합니다.

```
kubectl get pods -n projectcontour
```

```
NAME                       READY   STATUS      RESTARTS   AGE
contour-7966d6cdbf-skqfl   1/1     Running     1          21h
contour-7966d6cdbf-vc8c7   1/1     Running     1          21h
contour-certgen-77m2n      0/1     Completed   0          21h
envoy-fsltp                1/1     Running     0          20h
```

9 외부 IP 주소를 사용하여 로드 밸런서를 Ping합니다.

```
ping 192.168.123.5
```

```
PING 192.168.123.5 (192.168.123.5) 56(84) bytes of data.
64 bytes from 192.168.123.5: icmp_seq=1 ttl=62 time=3.50 ms
```

10 `ingress-nihao.yaml`이라는 수신 리소스를 생성합니다.

YAML을 생성합니다.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nihao
```

```
spec:
  rules:
  - http:
      paths:
      - path: /nihao
        backend:
          serviceName: nihao
          servicePort: 80
```

YAML을 적용합니다.

```
kubectl apply -f ingress-nihao.yaml
```

수신 리소스가 생성되었는지 확인합니다.

```
kubectl get ingress
```

엔보이 로드 밸런서의 외부 IP 주소(이 예에서는 `192.168.123.5`)는 수신 개체에서 사용됩니다.

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-nihao   <none>   *       192.168.123.5     80      17s
```

11 백엔드 애플리케이션으로 테스트 서비스를 배포합니다.

`ingress-nihao-test.yaml`이라는 다음 YAML 파일을 만듭니다.

```
kind: Service
apiVersion: v1
metadata:
  name: nihao
spec:
  selector:
    app: nihao
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nihao
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nihao
      tier: backend
      track: stable
  template:
    metadata:
```

```
      labels:
        app: nihao
        tier: backend
        track: stable
    spec:
      containers:
        - name: nihao
          image: "gcr.io/google-samples/hello-go-gke:1.0"
          ports:
            - name: http
              containerPort: 80
```

YAML을 적용합니다.

```
kubectl apply -f ingress-nihao-test.yaml
```

`nihao` 서비스가 생성되었는지 확인합니다.

```
kubectl get services
```

```
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nihao        ClusterIP   10.14.21.22     <none>        80/TCP     15s
```

백엔드 배포가 생성되었는지 확인합니다.

```
kubectl get deployments
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nihao   3/3     3            3           2m25s
```

백엔드 포드가 존재하는지 확인합니다.

```
kubectl get pods
```

```
NAME                     READY   STATUS    RESTARTS   AGE
nihao-8646584495-9nm8x   1/1     Running   0          106s
nihao-8646584495-vscm5   1/1     Running   0          106s
nihao-8646584495-zcsdq   1/1     Running   0          106s
```

12 Contour 수신 컨트롤러에서 사용하는 로드 밸런서의 공용 IP 주소를 가져옵니다.

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-nihao   <none>   *       10.19.14.76     80      13m
```

13 브라우저를 사용하여 공용 IP로 이동하고 수신 경로를 포함합니다.

```
http://10.19.14.76/nihao
```

"hello" 메시지가 반환됩니다.

```
{"message":"Hello"}
```

결과

클러스터 내에서 실행되는 서비스가 앞에 있는 백엔드 앱은 로드 밸런서의 외부 IP 주소를 사용하여 수신 컨트롤러를 통해 외부에서 브라우저로 액세스합니다.

영구 볼륨에 스토리지 클래스 사용

vSphere 네임스페이스에 할당된 vSphere 스토리지 정책은 영구 볼륨에 사용 가능한 두 가지 버전의 스토리지 클래스를 생성합니다. 선택하는 버전은 요구 사항에 따라 다릅니다.

사용 가능한 두 가지 버전의 스토리지 클래스

vSphere 스토리지 정책을 vSphere 네임스페이스에 할당하면 시스템은 해당 vSphere 네임스페이스에 일치하는 Kubernetes 스토리지 클래스를 생성합니다. 이 스토리지 클래스는 해당 네임스페이스에 프로비저닝된 각 TKG 클러스터에 복제됩니다. 그런 다음 스토리지 클래스를 사용하여 클러스터 워크로드에 대한 영구 스토리지 볼륨을 생성할 수 있습니다.

vSphere 네임스페이스에 할당된 각 vSphere 스토리지 정책에 대해 감독자에는 "즉시" 바인딩 모드의 단일 스토리지 클래스가 있습니다.

```
kubectl describe storageclass tkg-storage-policy
Name:                  tkg-storage-policy
IsDefaultClass:        No
Annotations:           cns.vmware.com/StoragePoolTypeHint=cns.vmware.com/vsan
Provisioner:           csi.vsphere.vmware.com
Parameters:            storagePolicyID=877b0f4b-b959-492a-b265-b4d460987b23
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>
```

해당 vSphere 네임스페이스에 배포된 각 TKG 클러스터에는 두 개의 스토리지 클래스가 있습니다. 하나는 감독자의 해당 스토리지 클래스와 동일하고 다른 하나는 이름에 `*-latebinding`이 추가되고 바인딩 모드로 "WaitForFirstConsumer"가 추가됩니다.

```
kubectl get sc
NAME                              PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
tkg-storage-policy                csi.vsphere.vmware.com   Delete
Immediate              true                   2m43s
tkg-storage-policy-latebinding    csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   2m43s
```

Kubernetes 스케줄러에서 계산을 선택한 후 영구 볼륨을 프로비저닝하려는 경우 스토리지 클래스의 `latebinding` 버전을 사용합니다. 자세한 내용은 Kubernetes 설명서에서 볼륨 바인딩 모드를 참조하십시오.

```
kubectl describe sc tkg-storage-policy
Name:                  tkg-storage-policy
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=tkg-storage-policy
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>

kubectl describe sc tkg-storage-policy-latebinding
Name:                  tkg-storage-policy-latebinding
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=tkg-storage-policy
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

스토리지 클래스에 패치 적용

감독자의 TKG에서는 kubectl 및 YAML을 사용하여 스토리지 클래스를 수동으로 생성할 수 없습니다. vSphere 스토리지 정책 프레임워크를 사용하여 스토리지 클래스를 생성한 다음 vSphere 네임스페이스에 적용할 수 있습니다. 그러면 해당 vSphere 네임스페이스에 프로비저닝된 각 TKG 클러스터에 두 개의 해당 스토리지 클래스가 생성됩니다.

kubectl 및 YAML을 사용하여 스토리지 클래스를 수동으로 생성할 수는 없지만 kubectl을 사용하여 기존 스토리지 클래스를 수정할 수 있습니다. 기본 스토리지 클래스를 지정하지 않고 TKG 클러스터를 프로비저닝했는데 이제는 기본 스토리지 클래스가 필요한 Helm 또는 Tanzu 패키지를 사용하여 애플리케이션을 배포하려는 경우 이렇게 하는 것이 필요할 수 있습니다.

기본 스토리지를 사용하여 완전히 새로운 클러스터를 생성하는 대신 기존 스토리지 클래스에 패치를 적용하고 Kubernetes 설명서 Change the default StorageClass(기본 StorageClass 변경)에 설명된 대로 default = true 주석을 추가할 수 있습니다.

예를 들어 다음 명령은 TKG 클러스터에서 사용 가능한 두 가지 스토리지 클래스를 반환합니다.

```
kubectl describe sc
Name:                  gc-storage-profile
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>


Name:                  gc-storage-profile-latebinding
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

다음 명령을 사용하여 스토리지 클래스 중 하나에 패치를 적용하고 주석을 추가합니다.

```
kubectl patch storageclass gc-storage-profile -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}'
storageclass.storage.k8s.io/gc-storage-profile patched
```

스토리지 클래스를 다시 확인하면 그 중 하나에 패치가 적용되어 기본값이 된 것이 보입니다.

```
kubectl describe sc
Name:                  gc-storage-profile
IsDefaultClass:        Yes
Annotations:           storageclass.kubernetes.io/is-default-class=true
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>


Name:                  gc-storage-profile-latebinding
IsDefaultClass:        No
Annotations:           <none>
```

```
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

영구 스토리지 볼륨을 동적으로 생성

기존 스토리지 클래스 및 PVC(영구 볼륨 할당)를 사용하여 영구 스토리지 볼륨을 동적으로 생성할 수 있습니다.

TKG 클러스터에 대한 동적 PVC

TKG 클러스터에서 상태 저장 워크로드를 실행하려면 기본 스토리지 인프라의 세부 정보를 모르는 상태에서 영구 스토리지 리소스를 요청하는 PVC(영구 볼륨 할당)를 생성할 수 있습니다. PVC에 사용되는 스토리지는 vSphere 네임스페이스에 대한 스토리지 할당량을 넘어 할당됩니다.

이 요청은 영구 볼륨 개체와 일치하는 가상 디스크를 동적으로 프로비저닝합니다. 할당은 영구 볼륨에 바인딩됩니다. 할당을 삭제하면 해당하는 영구 볼륨 개체와 프로비저닝된 가상 디스크도 삭제됩니다.

PVC를 생성하면 지원 영구 볼륨이 동적으로 생성됩니다. PVC는 **`tkg-store`** 스토리지 클래스를 참조합니다. 스토리지 클래스는 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스와 연결됩니다. 자세한 내용은 영구 볼륨에 스토리지 클래스 사용의 내용을 참조하십시오.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: tkg-cluster-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: tkg-store
  resources:
    requests:
        storage: 3Gi
```

PVC를 생성합니다.

```
kubectl apply -f pvc_name.yaml
```

PVC를 확인합니다.

```
kubectl get pvc my-pvc
```

포드 또는 배포 규격에서 PVC를 지정합니다. 예:

```
...
volumes:
    - name: my-pvc
      persistentVolumeClaim:
        claimName: my-pvc
```

영구 스토리지 볼륨을 정적으로 생성

감독자의 PVC(영구 볼륨 할당)를 사용하여 TKG 2.0 클러스터에서 PV(영구 볼륨)를 정적으로 생성할 수 있습니다.

영구 볼륨 정의

다음은 정적 PV(영구 볼륨)에 대한 정의 예시입니다. 정의에는 스토리지 클래스와 볼륨 핸들이 필요합니다. `volumeHandle`은 대상 TKG 클러스터가 프로비저닝된 동일한 vSphere 네임스페이스의 감독자에 생성된 PVC(영구 볼륨 할당)의 이름입니다. 이 PVC는 어떤 포드에도 연결되지 않아야 합니다.

다음 명령을 사용하여 `storageClassName`을 가져옵니다.

```
kubectl get storageclass
```

`volumeHandle`의 경우 감독자의 PVC 이름을 입력합니다.

```
apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: static-tkg-block-pv
     annotations:
       pv.kubernetes.io/provisioned-by: csi.vsphere.vmware.com
   spec:
     storageClassName: gc-storage-profile
     capacity:
       storage: 2Gi
     accessModes:
       - ReadWriteOnce
     persistentVolumeReclaimPolicy: Delete
     claimRef:
       namespace: default
       name: static-tkg-block-pvc
     csi:
       driver: "csi.vsphere.vmware.com"
       volumeAttributes:
         type: "vSphere CNS Block Volume"
       volumeHandle: "supervisor-block-pvc-name"   "#Enter the PVC name from Supervisor."
```

다음을 사용하여 PV를 생성합니다.

```
kubectl apply -f redis-leader-pvc.yaml -n guestbook
```

정적으로 정의된 PV에 대한 PVC(영구 볼륨 할당)

여러 영역에 감독자를 배포한 경우

`storageClassName`을 PV에서와 동일한 값으로 설정합니다.

```
kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: static-tkg-block-pvc
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 2Gi
     storageClassName: gc-storage-profile
     volumeName: static-tkg-block-pv
```

생성한 PV에 PVC가 바인딩되었는지 확인합니다.

```
kubectl get pv,pvc
```

TKG 클러스터에 방명록 애플리케이션 배포

방명록 애플리케이션을 TKG 클러스터에 배포하고 Kubernetes를 살펴봅니다.

방명록 애플리케이션을 배포하는 것은 Kubernetes를 살펴보기에 유용한 방법입니다. 애플리케이션은 배포 및 ReplicaSet 개체를 사용하여 기본 네임스페이스에 포드를 배포하고 서비스를 사용하여 이러한 포드를 노출합니다. 방명록 데이터는 지속되므로 애플리케이션이 다운되더라도 데이터는 그대로 유지됩니다. 이 자습서에서는 기본 스토리지의 세부 정보를 모르는 상태에서 동적 PVC(영구 볼륨 할당)를 사용하여 영구 스토리지 리소스를 요청합니다. PVC에 사용되는 스토리지는 vSphere 네임스페이스에 대한 스토리지 할당량 에서 할당됩니다. 컨테이너는 기본적으로 사용 후 삭제 및 상태 비저장입니다. 상태 저장 워크로드의 경우 일반적인 방식은 PVC(영구 볼륨 할당)를 생성하는 것입니다. PVC를 사용하여 영구 볼륨을 마운트하고 스토리지에 액세스할 수 있습니다. 이 요청은 영구 볼륨 개체와 일치하는 가상 디스크를 동적으로 프로비저닝합니다. 할당은 영구 볼륨에 바인딩됩니다. 할당을 삭제하면 해당하는 영구 볼륨 개체와 프로비저닝된 가상 디스크도 삭제됩니다.

사전 요구 사항

다음 항목을 검토하십시오.

- Kubernetes 설명서의 방명록 애플리케이션 자습서
- TKG 클러스터를 프로비저닝합니다.
- TKG 클러스터에 연결합니다.

절차

1 TKG 클러스터에 로그인합니다.

2 방명록 네임스페이스를 생성합니다.

```
kubectl create namespace guestbook
```

확인:

```
kubectl get ns
```

3 기본 권한이 있는 PSP를 사용하여 역할 기반 액세스 제어를 생성합니다.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

참고 더 많은 보안이 필요한 경우 방명록 네임스페이스에 RoleBinding을 적용합니다.

4 스토리지 클래스를 확인하거나 새로 생성합니다.

기존 스토리지 클래스를 확인하려면:

```
kubectl describe namespace
```

또는 vSphere 관리자 권한이 있는 경우:

```
kubectl get storageclass
```

5 스토리지 클래스를 참조하는 동적 PVC(영구 볼륨 할당) YAML 파일을 생성합니다.

다음 YAML 파일을 사용합니다. 각각을 스토리지 클래스의 이름으로 업데이트합니다.

- Redis 리더 영구 볼륨 할당
- Redis 팔로워 영구 볼륨 할당

6 방명록 PVC를 클러스터에 적용합니다.

```
kubectl apply -f redis-leader-pvc.yaml -n guestbook
```

```
kubectl apply -f redis-follower-pvc.yaml -n guestbook
```

7 PVC의 상태를 확인합니다.

```
kubectl get pvc,pv -n guestbook
```

PVC 및 PV(영구 볼륨)가 나열되고 사용할 수 있습니다.

```
NAME                                        STATUS
VOLUME                                      CAPACITY   ACCESS MODES   STORAGECLASS
AGE
persistentvolumeclaim/redis-follower-pvc    Bound    pvc-37b72f35-3de2-4f84-
be7d-50d5dd968f62   2Gi        RWO            tkgs-storage-class   66s
persistentvolumeclaim/redis-leader-pvc      Bound    pvc-2ef51f31-dd4b-4fe2-bf4c-
```

```
f0149cb4f3da   2Gi        RWO            tkgs-storage-class   66s

NAME                                                    CAPACITY   ACCESS MODES   RECLAIM
POLICY STATUS   CLAIM                  STORAGECLASS
persistentvolume/pvc-2ef51f31-dd4b-4fe2-bf4c   2Gi  RWO            Delete
Bound    guestbook/redis-leader-pvc     tkgs-storage-class
persistentvolume/pvc-37b72f35-3de2-4f84-be7d   2Gi  RWO            Delete
Bound    guestbook/redis-follower-pvc   tkgs-storage-class
```

8 방명록 YAML 파일을 생성합니다.

다음 YAML 파일을 사용합니다.

- Redis 리더 배포
- Redis 팔로워 배포
- Redis 리더 서비스
- Redis 팔로워 서비스
- 방명록 프런트 엔드 배포
- 방명록 프런트 엔드 서비스

9 네임스페이스에 방명록 애플리케이션을 배포합니다.

```
kubectl apply -f . --namespace guestbook
```

10 방명록 리소스의 생성을 확인합니다.

```
kubectl get all -n guestbook
```

```
NAME                                                READY   STATUS
RESTARTS   AGE
pod/guestbook-frontend-deployment-56fc5b6b47-cd58r   1/1     Running
0          65s
pod/guestbook-frontend-deployment-56fc5b6b47-fh6dp   1/1     Running
0          65s
pod/guestbook-frontend-deployment-56fc5b6b47-hgd2b   1/1     Running
0          65s
pod/redis-follower-deployment-6fc9cf5759-99fgw       1/1     Running
0          65s
pod/redis-follower-deployment-6fc9cf5759-rhxf7       1/1     Running
0          65s
pod/redis-leader-deployment-7d89bbdbcf-flt4q         1/1     Running
0          65s

NAME                                TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)        AGE
service/guestbook-frontend          LoadBalancer   10.10.89.59      10.19.15.99
80:31513/TCP   65s
service/redis-follower              ClusterIP      10.111.163.189   <none>
6379/TCP       65s
```

```
service/redis-leader                  ClusterIP      10.111.70.189    <none>
6379/TCP       65s

NAME                                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/guestbook-frontend-deployment    3/3     3            3           65s
deployment.apps/redis-follower-deployment        1/2     2            1           65s
deployment.apps/redis-leader-deployment          1/1     1            1           65s

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/guestbook-frontend-deployment-56fc5b6b47    3         3         3       65s
replicaset.apps/redis-follower-deployment-6fc9cf5759        2         2         1       65s
replicaset.apps/redis-leader-deployment-7d89bbdbcf          1         1         1       65s
```

11 `service/guestbook-frontend` 로드 밸런서의 `External-IP` 주소(이 예에서는 `10.19.15.99`)를 사용하여 방명록 웹 페이지에 액세스합니다.

방명록 웹 인터페이스가 표시되고 방명록 데이터베이스에 값을 입력할 수 있습니다. 애플리케이션을 다시 시작하면 데이터가 유지됩니다.

방명록 애플리케이션 YAML

예제 YAML 파일을 사용하여 영구 데이터 스토리지가 있는 방명록 애플리케이션을 배포합니다.

Redis 리더 영구 볼륨 할당

`redis-leader-pvc.yaml` 파일은 명명된 스토리지 클래스를 참조하는 영구 볼륨 클레임의 예입니다. 이 예제를 사용하려면 스토리지 클래스의 이름을 입력합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-leader-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: tkg-storage-class-name
  resources:
    requests:
      storage: 2Gi
```

Redis 팔로워 영구 볼륨 할당

`redis-follower-pvc.yaml` 파일은 명명된 스토리지 클래스를 참조하는 영구 볼륨 클레임의 예입니다. 이 예제를 사용하려면 스토리지 클래스의 이름을 입력합니다.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-follower-pvc
spec:
  accessModes:
```

```
    - ReadWriteOnce
  storageClassName: tkg-storage-class-name
  resources:
    requests:
      storage: 2Gi
```

Redis 리더 배포

redis-leader-deployment.yaml 파일은 영구 볼륨이 있는 Redis 리더 배포의 예입니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-leader-deployment
spec:
  selector:
    matchLabels:
      app: redis
      role: leader
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: leader
        tier: backend
    spec:
      containers:
      - name: leader
        image: redis:6.0.5
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
        volumeMounts:
        - name: redis-leader-data
          mountPath: /data
      volumes:
      - name: redis-leader-data
        persistentVolumeClaim:
          claimName: redis-leader-pvc
```

Redis 팔로워 배포

redis-follower-deployment.yaml 파일은 영구 볼륨이 있는 Redis 팔로워 배포의 예입니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-follower-deployment
```

```
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: follower
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: follower
        tier: backend
    spec:
      containers:
      - name: follower
        image: gcr.io/google_samples/gb-redis-follower:v2
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 6379
        volumeMounts:
        - name: redis-follower-data
          mountPath: /data
      volumes:
      - name: redis-follower-data
        persistentVolumeClaim:
          claimName: redis-follower-pvc
```

Redis 리더 서비스

`redis-leader-service.yaml` 파일은 Redis 리더 서비스의 예입니다.

```
apiVersion: v1
kind: Service
metadata:
  name: redis-leader
  labels:
    app: redis
    role: leader
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
```

```
  selector:
    app: redis
    role: leader
    tier: backend
```

Redis 팔로워 서비스

`redis-follower-service.yaml` 파일은 Redis 팔로워 서비스의 예입니다.

```
apiVersion: v1
kind: Service
metadata:
  name: redis-follower
  labels:
    app: redis
    role: follower
    tier: backend
spec:
  ports:
  - port: 6379
  selector:
    app: redis
    role: follower
    tier: backend
```

방명록 프런트 엔드 배포

`guestbook-frontend-deployment.yaml` 파일은 방명록 프런트 엔드 배포의 예입니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: guestbook-frontend-deployment
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v5
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
```

```
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

방명록 프런트 엔드 서비스

`guestbook-frontend-service.yaml` 파일은 방명록 프런트 엔드 로드 밸런서 서비스의 예입니다.

```
apiVersion: v1
kind: Service
metadata:
  name: guestbook-frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

지연 바인딩 볼륨 연결로 여러 vSphere 영역에 StatefulSet 애플리케이션 배포

이 예는 StatefulSet 애플리케이션을 감독자의 TKG 클러스터에 배포하는 방법을 보여줍니다.

스토리지 클래스

스토리지 클래스에는 선택할 수 있는 두 가지 버전이 있습니다. 이 배포의 경우 `*-latebinding` 버전을 사용합니다.

```
kubectl get sc
NAME                                PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
zonal-ds-policy-105                 csi.vsphere.vmware.com   Delete
Immediate              true                   17h
zonal-ds-policy-105-latebinding     csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   17h
```

다중 영역 감독자 토폴로지

TKG 클러스터는 여러 vSphere 영역의 감독자에 프로비저닝됩니다.

```
kubectl get nodes -L topology.kubernetes.io/zone
NAME                                                          STATUS
ROLES                  AGE   VERSION            ZONE
test-cluster-e2e-script-105-m72sb-2dnsz                       Ready    control-
```

```
plane,master   18h   v1.22.6+vmware.1   zone-1
test-cluster-e2e-script-105-m72sb-rmtjn                          Ready    control-
plane,master   18h   v1.22.6+vmware.1   zone-2
test-cluster-e2e-script-105-m72sb-rvhb8                          Ready    control-
plane,master   18h   v1.22.6+vmware.1   zone-3
test-cluster-e2e-script-105-nodepool-1-p86fm-6dfcdc77b7-fxm4s   Ready
<none>                 18h   v1.22.6+vmware.1   zone-1
test-cluster-e2e-script-105-nodepool-2-gx5gs-7cf4895b77-6wlb4   Ready
<none>                 18h   v1.22.6+vmware.1   zone-2
test-cluster-e2e-script-105-nodepool-3-fkkc9-856cd45985-d8nsl   Ready
<none>                 18h   v1.22.6+vmware.1   zone-3
```

상태 저장 집합

StatefulSet(`sts.yaml`)은 애플리케이션을 포드에 배포하며, 각 포드에는 스케줄 조정 시 유지 관리하는 영구 식별자가 있습니다.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  serviceName: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: topology.kubernetes.io/zone
                operator: In
                values:
                - zone-1
                - zone-2
                - zone-3
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - nginx
            topologyKey: topology.kubernetes.io/zone
      containers:
```

```
        - name: nginx
          image: gcr.io/google_containers/nginx-slim:0.8
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
            - name: logs
              mountPath: /logs
  volumeClaimTemplates:
    - metadata:
        name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: zonal-ds-policy-105-latebinding
        resources:
          requests:
            storage: 2Gi
    - metadata:
        name: logs
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: zonal-ds-policy-105-latebinding
        resources:
          requests:
            storage: 1Gi
```

애플리케이션 배포

다음과 같이 StatefulSet 애플리케이션을 배포합니다. 배포에 성공하면 지연 바인딩 볼륨 모드로 스토리지 클래스를 사용하는 첫 번째 소비자 대기를 사용하여 3개 vSphere 영역에 애플리케이션 포드가 스케줄링됩니다.

1 StatefulSet을 배포합니다.

```
kubectl create -f sts.yaml
statefulset.apps/web created
```

2 StatefulSet을 확인합니다.

```
kubectl get statefulset
NAME   READY   AGE
web    3/3     112s
```

3 포드를 확인합니다.

```
kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE    IP
NODE                                                          NOMINATED NODE   READINESS
GATES
web-0   1/1     Running   0          117s   172.16.1.2   test-cluster-e2e-script-105-
nodepool-3-fkkc9-856cd45985-d8nsl   <none>           <none>
```

```
web-1   1/1     Running   0          90s    172.16.2.2   test-cluster-e2e-script-105-
nodepool-2-gx5gs-7cf4895b77-6wlb4   <none>           <none>
web-2   1/1     Running   0          53s    172.16.3.2   test-cluster-e2e-script-105-
nodepool-1-p86fm-6dfcdc77b7-fxm4s   <none>           <none>
```

4 여러 영역 및 지연 볼륨 바인딩에서 포드 스케줄링을 확인합니다.

```
kubectl get pv -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.spec.claimRef.name}
{"\t"}{.spec.nodeAffinity}{"\n"}{end}'
pvc-7010597f-31cf-4ab1-bbd7-98ac04e0c603    www-web-2    {"required":{"nodeSelectorTerms":
[{"matchExpressions":[{"key":"topology.kubernetes.io/zone","operator":"In","values":
["zone-1"]}]}]}}
pvc-921fadfc-df89-456d-a341-00f4117035f8    logs-web-0    {"required":{"nodeSelectorTerms":
[{"matchExpressions":[{"key":"topology.kubernetes.io/zone","operator":"In","values":
["zone-3"]}]}]}}
pvc-bcb46a24-58cb-4ec7-a964-391fe80400fc    www-web-1    {"required":{"nodeSelectorTerms":
[{"matchExpressions":[{"key":"topology.kubernetes.io/zone","operator":"In","values":
["zone-2"]}]}]}}
pvc-f51a44e5-ec19-4bec-b67a-3e34512049b8    www-web-0    {"required":{"nodeSelectorTerms":
[{"matchExpressions":[{"key":"topology.kubernetes.io/zone","operator":"In","values":
["zone-3"]}]}]}}
pvc-fa68887a-31dd-4d9e-bb39-88653a9d80c9    logs-web-2    {"required":{"nodeSelectorTerms":
[{"matchExpressions":[{"key":"topology.kubernetes.io/zone","operator":"In","values":
["zone-1"]}]}]}}
pvc-fc2cd6f7-b033-48ee-892d-df5318ec6f3e    logs-web-1    {"required":{"nodeSelectorTerms":
[{"matchExpressions":[{"key":"topology.kubernetes.io/zone","operator":"In","values":
["zone-2"]}]}]}}
```

TKG 서비스 클러스터에 AI/ML 워크로드 배포

13

NVIDIA vGPU를 사용하여 TKG 서비스 클러스터에 AI/ML 워크로드를 배포할 수 있습니다. AI/ML 워크로드를 배포하려면 vSphere 관리자의 초기 설정과 클러스터 운영자의 일부 구성이 필요합니다. 환경이 vGPU 지원으로 설정되면 개발자는 컨테이너 기반 AI/ML 워크로드를 TKG 서비스 클러스터에 배포할 수 있습니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터에 AI/ML 워크로드 배포 정보
- TKGS 클러스터에 AI/ML 워크로드를 배포하기 위한 vSphere 관리자 워크플로
- TKG 서비스 클러스터에 AI/ML 워크로드를 배포하기 위한 클러스터 운영자 워크플로
- NVIDIA vGPU 디바이스에 대한 사용자 지정 VM 클래스 생성

TKG 서비스 클러스터에 AI/ML 워크로드 배포 정보

NVIDIA GPU 기술을 사용하여 TKG 서비스 클러스터에 AI/ML 워크로드를 배포할 수 있습니다.

AI/ML 워크로드에 대한 TKGS 지원

TKG 서비스 클러스터에 계산 집약적인 워크로드를 배포할 수 있습니다. 이 컨텍스트에서 계산 집약적인 워크로드는 GPU 가속기 디바이스를 사용해야 하는 AI(인공 지능) 또는 ML(기계 학습) 애플리케이션입니다.

Kubernetes 환경에서 AI/ML 워크로드의 실행을 용이하게 하기 위해 VMware는 NVIDIA와 제휴하여 NVIDIA GPU Cloud 플랫폼을 지원합니다. TKGS 클러스터에 NGC 카탈로그의 컨테이너 이미지를 배포할 수 있습니다. vSphere 8 NVIDIA GPU 지원에 대한 자세한 내용은 Tech Zone의 vGPU 문서를 참조하십시오.

지원되는 GPU 모드

TKG 서비스 클러스터에 NVIDIA 기반 AI/ML 워크로드를 배포하려면 Ubuntu 버전의 Tanzu Kubernetes 릴리스 버전 1.22 이상을 사용해야 합니다. vSphere는 동적 DirectPath I/O 디바이스를 사용하는 NVIDIA Grid vGPU 및 GPU 패스스루라는 두 가지 모드를 지원합니다. 자세한 내용은 NVIDIA 설명서에서 지원되는 운영 체제 및 Kubernetes 플랫폼을 참조하십시오.

표 13-1. NVIDIA vGPU를 사용하는 vSphere VM

운영 체제	TKr	vSphere with Tanzu	설명
Ubuntu 20.04 LTS	1.22 - 1.2x*(최신, 1.28까지)	7.0 U3c 8.0 U2+	GPU 디바이스는 각 ESXi 호스트에 설치된 NVIDIA 호스트 관리자 드라이버에 의해 가상화됩니다. 그런 다음 GPU 디바이스는 여러 NVIDIA vGPU(가상 GPU)에서 공유됩니다. **참고** vSphere DRS(Distributed Resource Scheduler)는 vSphere 클러스터를 구성하는 호스트 전체에 vGPU VM을 너비 우선 방식으로 분산합니다. 자세한 내용은 vSphere 리소스 관리 가이드에서 vGPU VM의 DRS 배치를 참조하십시오. 각 NVIDIA vGPU는 GPU 디바이스의 메모리 양으로 정의됩니다. 예를 들어 GPU 디바이스의 총 RAM 용량이 32GB인 경우 각각 4GB의 메모리로 8개의 vGPU를 생성할 수 있습니다.

표 13-2. GPU 패스스루를 사용하는 vSphere VM

운영 체제	TKr	vSphere with Tanzu	설명
Ubuntu 20.04 LTS	1.22 - 1.2x*(최신, 1.28까지)	7.0 U3c 8.0 U2+	NVIDIA vGPU 프로파일을 구성하는 동일한 VM 클래스에서, 동적 DirectPath IO를 사용하는 패스스루 네트워킹 디바이스에 대한 지원을 포함합니다. 이 경우 vSphere DRS에 따라 VM 배치가 결정됩니다.

TKGS 클러스터에 AI/ML 워크로드를 배포하기 위한 vSphere 관리자 워크플로

개발자가 TKG 클러스터에 AI/ML 워크로드를 배포할 수 있도록 하려면 vSphere 관리자가 NVIDIA GPU 하드웨어를 지원하도록 감독자 환경을 설정해야 합니다.

관리 1단계: 시스템 요구 사항 검토

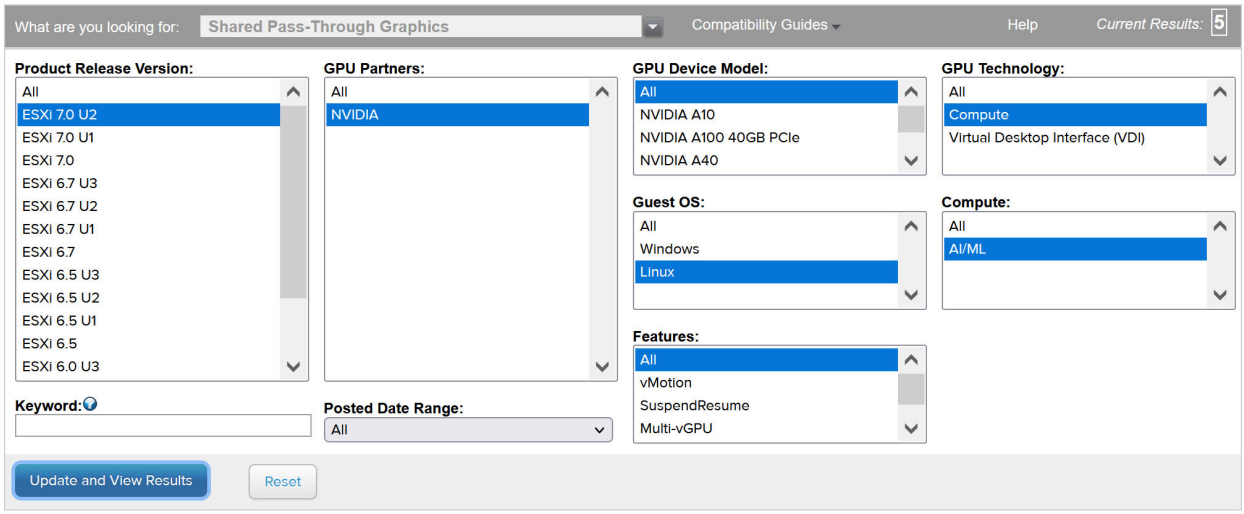
TKG 클러스터에 AI/ML 워크로드를 배포하기 위한 환경을 설정하려면 다음 시스템 요구 사항을 참조하십시오.

요구 사항	설명
vSphere 8 인프라	vCenter Server 및 ESXi 호스트
워크로드 관리 라이센스	vSphere 네임스페이스 및 감독자
TKR Ubuntu OVA	Tanzu Kubernetes 릴리스의 릴리스 정보
NVIDIA vGPU 호스트 드라이버	NGC 웹 사이트에서 VIB를 다운로드합니다. 자세한 내용은 vGPU 소프트웨어 드라이버 설명서를 참조하십시오.
vGPU용 NVIDIA 라이센스 서버	조직에서 제공한 FQDN

관리자 2단계: ESXi 호스트에 지원되는 NVIDIA GPU 디바이스 설치

TKG에 AI/ML 워크로드를 배포하려면 **워크로드 관리**가 사용되도록 설정될 vCenter 클러스터를 구성하는 각 ESXi 호스트에 하나 이상의 지원되는 NVIDIA GPU 디바이스를 설치합니다.

호환되는 NVIDIA GPU 디바이스를 보려면 VMware 호환성 가이드를 참조하십시오.



Click here to **Read** Important Support Information

Click on the 'GPU Device Model' to view more details and to subscribe to RSS feeds.

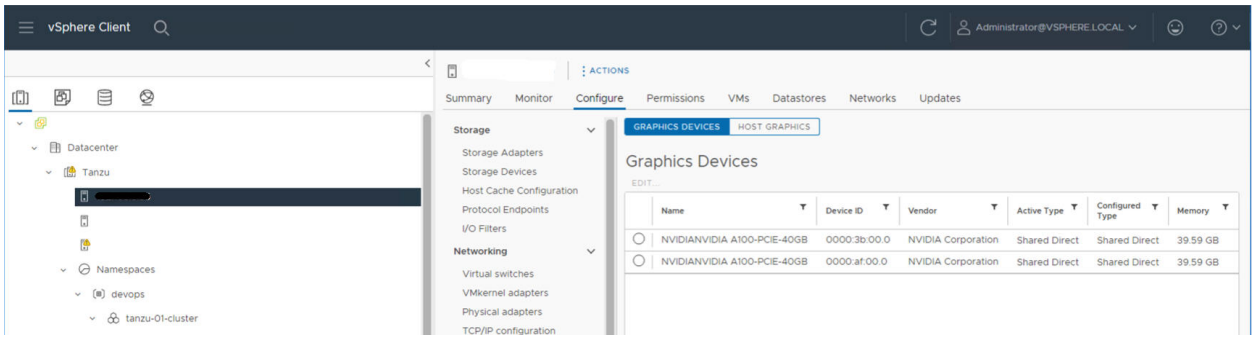
Bookmark | Print | Export to CSV

Search Results: Your search for" Shared Pass-Through Graphics " returned **5 results.** Back to Top Turn Off Auto Scroll Display: 10

GPU Partner	GPU Device Model	ESX Version	Compute
NVIDIA	NVIDIA A10	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A100 40GB PCIe	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A100 80GB PCIe	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A30	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A40	ESXi 7.0 U2	AI/ML

NVIDA GPU 디바이스는 최신 NVAIE(NVIDIA AI Enterprise) vGPU 프로파일을 지원해야 합니다. 지침을 보려면 NVIDIA 가상 GPU 소프트웨어 지원 GPU 설명서를 참조하십시오.

예를 들어 다음 ESXi 호스트에는 두 개의 NVIDIA GPU A100 디바이스가 설치되어 있습니다.



관리자 3단계: vGPU 작업을 위한 각 ESXi 호스트 구성

워크로드 관리가 사용되도록 설정된 vCenter 클러스터를 구성하는 각 ESXi 호스트에 대해 Shared Direct 및 SR-IOV를 사용하도록 설정하여 NVIDIA vGPU에 대한 호스트를 구성합니다.

각 ESXi 호스트에서 Shared Direct 사용

NVIDIA vGPU 기능의 잠금을 해제하려면 **워크로드 관리**가 사용되도록 설정된 vCenter 클러스터를 구성하는 각 ESXi 호스트에서 **Shared Direct** 모드를 사용하도록 설정합니다.

Shared Direct를 사용하도록 설정하려면 다음 단계를 완료하십시오. 추가 지침은 vSphere에서 가상 그래픽 구성을 참조하십시오.

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 vCenter 클러스터에서 ESXi 호스트를 선택합니다.

3 **구성 > 하드웨어 > 그래픽 > 그래픽 디바이스**를 선택합니다.

4 NVIDIA GPU 가속기 디바이스를 선택합니다.

5 그래픽 디바이스 설정을 **편집**합니다.

6 **Shared Direct**를 선택합니다.

7 **공유 패스스루 GPU 할당 정책**의 경우 최고의 성능을 위해 **GPU 전체에 VM 분산**을 선택합니다.

8 **확인**을 클릭하여 구성을 저장합니다.

9 이 설정은 호스트를 다시 시작한 후에 적용됩니다.

10 ESXi 호스트를 마우스 오른쪽 버튼으로 클릭하고 유지 보수 모드로 전환합니다.

11 호스트를 재부팅합니다.

12 호스트가 다시 실행되면 호스트를 유지 보수 모드에서 해제합니다.

13 **워크로드 관리**를 지원하는 vSphere 클러스터의 각 ESXi 호스트에 대해 이 프로세스를 반복합니다.

NVIDIA GPU A30 및 A100 디바이스용 SR-IOV BIOS 켜기

다중 인스턴스 GPU(MIG 모드)에 필요한 NVIDIA A30 또는 A100 GPU 디바이스를 사용 중인 경우 ESXi 호스트에서 SR-IOV를 사용하도록 설정해야 합니다. SR-IOV를 사용하도록 설정하지 않으면 Tanzu Kubernetes 클러스터 노드 VM을 시작할 수 없습니다. 이 문제가 발생하면 **워크로드 관리**가 사용되도록 설정된 vCenter Server의 **최근 작업** 창에 다음과 같은 오류 메시지가 표시됩니다.

```
Could not initialize plugin libnvidia-vgx.so for vGPU nvidia_aXXX-xx. Failed to start the virtual machine. Module DevicePowerOn power on failed.
```

SR-IOV를 사용하도록 설정하려면 웹 콘솔을 사용하여 ESXi 호스트에 로그인합니다. **관리 > 하드웨어**를 선택합니다. NVIDIA GPU 디바이스를 선택하고 **SR-IOV 구성**을 클릭합니다. 여기에서 SR-IOV를 켤 수 있습니다. 추가 지침은 vSphere 설명서에서 SR-IOV(Single Root I/O Virtualization)를 참조하십시오.

동적 DirectPath IO(패스스루 지원 디바이스)가 있는 vGPU

동적 DirectPath IO가 있는 vGPU를 사용하는 경우 다음 추가 구성을 수행합니다.

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 vCenter 클러스터에서 대상 ESXi 호스트를 선택합니다.

3 **구성 > 하드웨어 > PCI 디바이스**를 선택합니다.

4 **모든 PCI 디바이스** 탭을 선택합니다.

5 대상 NVIDIA GPU 가속기 디바이스를 선택합니다.

6 **패스스루 전환**을 클릭합니다.

7 ESXi 호스트를 마우스 오른쪽 버튼으로 클릭하고 유지 보수 모드로 전환합니다.

8 호스트를 재부팅합니다.

9 호스트가 다시 실행되면 호스트를 유지 보수 모드에서 해제합니다.

관리자 4단계: 각 ESXi 호스트에 NVIDIA 호스트 관리자 드라이버 설치

NVIDIA vGPU 그래픽 가속을 사용하여 Tanzu Kubernetes 클러스터 노드 VM을 실행하려면 **워크로드 관리**가 사용되도록 설정될 vCenter 클러스터를 구성하는 각 ESXi 호스트에 NVIDIA 호스트 관리자 드라이버를 설치합니다.

NVIDIA vGPU 호스트 관리자 드라이버 구성 요소는 VIB(vSphere 설치 번들)에 패키지됩니다. NVAIE VIB는 NVIDIA GRID 라이센싱 프로그램을 통해 조직에서 제공합니다. VMware는 NVAIE VIB를 제공하지 않으며 이에 대한 다운로드 서비스도 제공하지 않습니다. 조직에서는 NVIDIA 라이센싱 프로그램의 일부로 라이센싱 서버를 설정합니다. 자세한 내용은 NVIDIA 가상 GPU 소프트웨어 빠른 시작 가이드를 참조하십시오.

NVIDIA 환경이 설정되면 각 ESXi 호스트에서 다음 명령을 실행하고 NVIDIA 라이센스 서버 주소와 NVAIE VIB 버전을 환경에 적합한 값으로 바꿉니다. 추가 지침은 VMware 지원 기술 자료에서 ESXi에서 NVIDIA VIB 설치 및 구성을 참조하십시오.

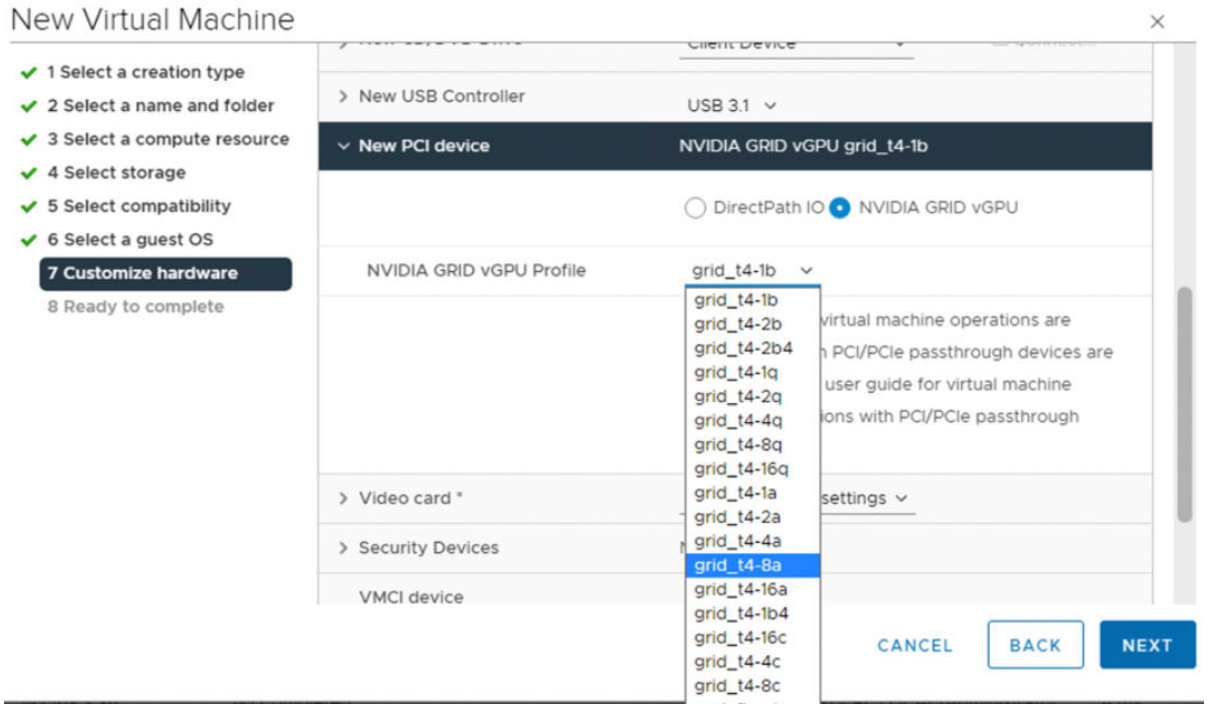
참고 ESXi 호스트에 설치된 NVAIE VIB 버전은 노드 VM에 설치된 vGPU 소프트웨어 버전과 일치해야 합니다. 아래 버전은 예시일 뿐입니다.

```
esxcli system maintenanceMode set --enable true
esxcli software vib install -v ftp://server.domain.example.com/nvidia/signed/
NVIDIA_bootbank_NVIDIA-VMware_ESXi_7.0_Host_Driver_460.73.02-1OEM.700.0.0.15525992.vib
esxcli system maintenanceMode set --enable false
/etc/init.d/xorg restart
```

관리자 5단계: ESXi 호스트가 NVIDIA vGPU 작업을 수행할 준비가 되었는지 확인

각 ESXi 호스트가 NVIDIA vGPU 작업을 수행할 준비가 되었는지 확인하려면 **워크로드 관리**가 사용되도록 설정될 vCenter 클러스터의 각 ESXi 호스트에 대해 다음 검사를 수행합니다.

- SSH를 사용하여 ESXi 호스트에 연결하고 셸 모드로 전환한 후 명령 `nvidia-smi`를 실행합니다. NVIDIA 시스템 관리 인터페이스는 NVIDA vGPU 호스트 관리자가 제공하는 명령줄 유틸리티입니다. 이 명령을 실행하면 호스트에서 GPU 및 드라이버가 반환됩니다.
- 다음 명령을 실행하여 NVIDIA 드라이버가 제대로 설치되었는지 확인합니다. `esxcli software vib list | grep NVIDA`.
- 호스트가 GPU Shared Direct로 구성되어 있고 SR-IOV가 켜져 있는지 확인합니다(NVIDIA A30 또는 A100 디바이스를 사용 중인 경우).
- GPU용으로 구성된 ESXi 호스트에서 vSphere Client를 사용하여 PCI 디바이스가 포함된 새 가상 시스템을 생성합니다. NVIDIA vGPU 프로파일이 표시되면서 선택 가능한 상태가 됩니다.



관리자 6단계: 워크로드 관리 사용

워크로드 관리를 사용하도록 설정하려면 TKG 서비스 클러스터 배포 항목을 참조하십시오.

참고 **워크로드 관리**가 사용되도록 설정된 vSphere 클러스터가 이미 있는 경우 클러스터가 vGPU용으로 구성한 ESXi 호스트를 사용하고 있다고 가정하고 이 단계를 건너뜁니다.

관리자 7단계: TKR Ubuntu로 컨텐츠 라이브러리 생성 또는 업데이트

NVIDIA vGPU에는 Ubuntu 운영 체제가 필요합니다. vGPU 클러스터에는 Tanzu Kubernetes 릴리스의 PhotonOS 버전을 사용할 수 없습니다.

VMware에서는 Tanzu Kubernetes 릴리스의 Ubuntu 버전을 제공합니다. vSphere 8부터 Ubuntu 버전은 클러스터 YAML의 주석을 사용하여 지정됩니다.

지원되는 Ubuntu TKR을 사용하여 기존 컨텐츠 라이브러리를 생성하거나 업데이트합니다. 장 5 TKG 서비스 클러스터에 대한 Kubernetes 릴리스 관리의 내용을 참조하십시오.

참고 vCenter에 기존 TKR 컨텐츠 라이브러리가 이미 구성되어 있는 경우 이 단계를 건너뜁니다. TKR에 대한 두 번째 컨텐츠 라이브러리를 생성하지 마십시오. 그러면 시스템이 불안정해질 수 있습니다.

관리자 8단계: vGPU 프로파일을 사용하여 사용자 지정 VM 클래스 생성

vGPU 프로파일이 있는 사용자 지정 VM 클래스를 생성합니다. 그런 다음 클러스터 규격에서 이 VM 클래스를 사용하여 TKGS 클러스터 노드를 생성합니다. 지침은 NVIDIA vGPU 디바이스에 대한 사용자 지정 VM 클래스 생성 항목을 참조하십시오.

관리자 9단계: vSphere 네임스페이스 구성

프로비저닝하려는 각 TKG vGPU 클러스터에 대해 vSphere 네임스페이스를 생성합니다. TKG 서비스 클러스터 호스팅을 위한 vSphere 네임스페이스 생성의 내용을 참조하십시오.

편집 권한이 있는 vSphere SSO 사용자/그룹을 추가하여 vSphere 네임스페이스를 구성하고 영구 볼륨에 대한 스토리지 정책을 연결합니다. TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성의 내용을 참조하십시오.

원하는 Ubuntu 이미지가 저장된 TKR 컨텐츠 라이브러리를 vSphere 네임스페이스와 연결합니다. TKR 컨텐츠 라이브러리를 TKG 서비스와 연결의 내용을 참조하십시오.

사용자 지정 VM 클래스를 vSphere 네임스페이스와 연결합니다.

- [vSphere 네임스페이스 선택]에서 **VM 서비스** 타일을 선택하고 **VM 클래스 관리**를 클릭합니다.
- 클래스 목록에서 생성한 사용자 지정 VM 클래스를 찾습니다.
- 클래스를 선택하고 **추가**를 클릭합니다.

추가 지침은 VM 클래스를 vSphere 네임스페이스와 연결 항목을 참조하십시오.

관리자 10단계: 감독자가 준비되었는지 확인

마지막 관리 작업은 감독자가 프로비저닝되었는지 그리고 클러스터 운영자가 AI/ML 워크로드에 대한 TKG 클러스터를 프로비저닝하는 데 이를 사용할 수 있는지 확인하는 것입니다.

vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

TKG 서비스 클러스터에 AI/ML 워크로드를 배포하기 위한 클러스터 운영자 워크플로

개발자가 TKG 서비스 클러스터에 AI/ML 워크로드를 배포할 수 있도록 하려면 클러스터 운영자가 하나 이상의 Kubernetes 클러스터를 생성하고 각각에 NVIDIA Network Operator와 NVIDIA GPU Operator를 설치해야 합니다.

운영자 1단계: 사전 요구 사항 확인

이 지침에서는 vSphere 관리자가 NVIDIA GPU에 대한 환경을 설정했다고 가정합니다. TKGS 클러스터에 AI/ML 워크로드를 배포하기 위한 vSphere 관리자 워크플로의 내용을 참조하십시오.

이 지침은 GPU Operator의 NVAIE(NVIDIA AI Enterprise) 버전을 설치한다고 가정합니다. 이 버전은 vSphere IaaS control plane에서 사용하도록 사전 구성되고 최적화되어 있습니다. NVAIE GPU Operator는 공용 NGC 카탈로그에서 사용할 수 있는 GPU Operator와 다릅니다. 자세한 내용은 NVIDIA AI Enterprise를 참조하십시오.

이러한 지침에서는 ESXi에 대해 일치하는 VIB가 있는 vGPU 드라이버 및 NVAIE GPU Operator 버전을 사용하고 있다고 가정합니다. 자세한 내용은 NVIDIA GPU Operator 버전 관리를 참조하십시오.

TKG 클러스터를 프로비저닝할 때 TKR의 Ubuntu 버전을 사용해야 합니다. vSphere 8 감독자의 TKG를 사용하면 Ubuntu 버전이 주석을 사용하여 클러스터 YAML에 지정됩니다.

운영자 2단계: NVIDIA vGPU용 TKGS 클러스터 프로비저닝

VMware에서는 NVIDIA GPU Operator 및 NVIDIA Network Operator를 사용하여 NVIDIA GPU Certified Server에서 NVIDIA 가상 GPU에 대한 네이티브 TKGS 지원을 제공합니다. 이러한 오퍼레이터는 TKGS 워크로드 클러스터에 설치합니다. vGPU 워크로드를 호스팅하기 위해 TKGS 클러스터를 프로비저닝하려면 다음 단계를 완료합니다.

1 vSphere에 대한 Kubernetes CLI 도구를 설치합니다.

vSphere에 대한 Kubernetes CLI 도구 설치의 내용을 참조하십시오.

2 kubectl용 vSphere 플러그인을 사용하여 감독자로 인증합니다.

```
kubectl vsphere login --server=IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

참고 FQDN은 감독자를 사용하도록 설정된 경우에만 사용할 수 있습니다.

3 kubectl을 사용하여 vSphere 관리자가 TKGS vGPU 클러스터에 대해 생성한 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config get-contexts
```

```
kubectl config use-context TKG-GPU-CLUSTER-NAMESPACE
```

4 vSphere 관리자가 생성한 vGPU 프로파일을 사용하여 사용자 지정 VM 클래스의 이름을 가져옵니다.

```
kubectl get virtualmachineclass
```

참고 VM 클래스는 대상 vSphere 네임스페이스에 바인딩되어야 합니다.

5 vSphere 관리자가 컨텐츠 라이브러리에서 동기화하고 vSphere 네임스페이스에 추가한 Ubuntu Tanzu Kubernetes 릴리스에 대한 TKR NAME을 가져옵니다.

```
kubectl get tkr
```

6 vGPU 지원 TKG 클러스터 프로비저닝을 위한 YAML을 만듭니다.

a 사용할 TKGS 클러스터 프로비저닝 API(v1alpha3 API 또는 v1beta1 API)를 결정합니다. TKG 클러스터 프로비저닝 API

b 선택하는 API에 따라 해당 API에 대한 Ubuntu 예를 참조하십시오.

- v1alpha3 예시: Ubuntu TKR이 있는 TKC
- v1beta1 예시: Ubuntu TKR이 있는 클러스터

참고 Ubuntu OS 이미지를 사용해야 합니다. Photon OS는 사용할 수 없습니다.

c 이전 명령의 출력에서 수집한 정보를 사용하여 TKGS 클러스터 규격을 사용자 지정합니다.

7 다음 kubectl 명령을 실행하여 클러스터를 프로비저닝합니다.

```
kubectl apply -f CLUSTER-NAME.yaml
```

예:

```
kubectl apply -f tkg-gpu-cluster-1.yaml
```

8 클러스터 프로비저닝을 확인합니다.

kubectl을 사용하여 클러스터 노드의 배포를 모니터링합니다.

```
kubectl get tanzukubernetesclusters -n NAMESPACE
```

9 kubectl용 vSphere 플러그인을 사용하여 TKGS vGPU 클러스터에 로그인합니다.

```
kubectl vsphere login --server=IP-ADDRESS-or-FQDN --vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace
NAMESPACE-NAME
```

10 클러스터를 확인합니다.

다음 명령을 사용하여 클러스터를 확인합니다.

```
kubectl cluster-info
```

```
kubectl get nodes
```

```
kubectl get namespaces
```

```
kubectl api-resources
```

운영자 3단계: NVIDIA Network Operator 설치

NVIDIA Network Operator는 Kubernetes 사용자 지정 리소스 및 운영자 프레임워크를 활용하여 vGPU에 대한 네트워킹을 최적화합니다. 자세한 내용은 NVIDIA Network Operator를 참조하십시오.

1 TKGS vGPU 워크로드 클러스터에 로그인했고 컨텍스트가 TKGS vGPU 워크로드 클러스터 네임스페이스로 설정되어 있는지 확인합니다.

필요한 경우 운영자 2단계: NVIDIA vGPU용 TKGS 클러스터 프로비저닝지침을 참조하십시오.

2 Helm 설명서를 참조하여 Helm을 설치합니다.

3 NVIDIA Network Operator Helm Chart를 가져옵니다.

```
helm fetch https://helm.ngc.nvidia.com/nvaie/charts/network-operator-v1.1.0.tgz --
username='$oauthtoken' --password=<YOUR API KEY> --untar
```

4 구성 값에 대한YAML 파일을 생성합니다.

```
vi values.yaml
```

5 다음 정보로 `values.yaml` 파일을 채웁니다.

```
deployCR: true
 ofedDriver:
  deploy: true
 rdmaSharedDevicePlugin:
  deploy: true
  resources:
   - name: rdma_shared_device_a
   vendors: [15b3]
   devices: [ens192]
```

6 다음 명령을 사용하여 NVIDIA Network Operator를 설치합니다.

```
helm install network-operator -f ./values.yaml -n network-operator --create-namespace --
wait network-operator/
```

운영자 4단계: NVIDIA GPU Operator 설치

NVIDIA는 NVIDIA AI Enterprise 고객을 위해 사전 구성된 GPU Operator를 제공합니다. 이 지침에서는 미리 구성된 GPU Operator 버전을 사용한다고 가정합니다. 이 지침은 NVIDIA에서 제공한 GPU Operator 설치용 지침을 기반으로 하지만 vSphere 8의 TKG용으로 업데이트되었습니다.

프로비저닝한 TKG 클러스터에 GPU Operator NVIDIA AI Enterprise를 설치하려면 다음 단계를 완료합니다.

1 TKGS vGPU 워크로드 클러스터에 로그인했고 컨텍스트가 TKGS vGPU 워크로드 클러스터 네임스페이스로 설정되어 있는지 확인합니다.

필요한 경우 운영자 2단계: NVIDIA vGPU용 TKGS 클러스터 프로비저닝지침을 참조하십시오.

2 아직 설치되지 않은 경우 Helm 설명서를 참조하여 Helm을 설치합니다.

3 `gpu-operator` Kubernetes 네임스페이스를 생성합니다.

```
kubectl create namespace gpu-operator
```

4 빈 vGPU 라이센스 구성 파일을 생성합니다.

```
sudo touch gridd.conf
```

5 NLS 클라이언트 라이센스 토큰을 생성하고 다운로드합니다.

NVIDIA License System User Guide의 "섹션 4.6. Generating a Client Configuration Token" 을 참조하십시오.

6 다운로드한 NLS 클라이언트 라이센스 토큰의 이름을 `client_configuration_token.tok`으로 바꿉니다.

7 `gpu-operator` 네임스페이스에 `licensing-config` ConfigMap 개체를 생성합니다.

이 ConfigMap에 vGPU 라이센스 구성 파일(`gridd.conf`) 및 NLS 클라이언트 라이센스 토큰(`*.tok`)을 포함합니다.

```
kubectl create configmap licensing-config \
    -n gpu-operator --from-file=gridd.conf --from-file=<path>/
client_configuration_token.tok
```

8 NVIDIA GPU Operator에 사용할 Linux용 컨테이너화된 NVIDIA vGPU 소프트웨어 그래픽 드라이버가 포함된 개인 레지스트리에 대한 이미지 끌어오기 암호를 생성합니다.

레지스트리 암호 이름이 `ngc-secret`이고 개인 레지스트리 이름이 `nvcr.io/nvaie`인 이미지 끌어오기 암호를 `gpu-operator` 네임스페이스에 생성합니다. 표시된 필드에 NGC API 키와 이메일 주소를 포함합니다.

```
kubectl create secret docker-registry ngc-secret \
--docker-server='nvcr.io/nvaie' \
--docker-username='$oauthtoken' \
--docker-password=<YOUR_NGC_API_KEY> \
--docker-email=<YOUR_EMAIL_ADDRESS> \
-n gpu-operator
```

9 NVAIE GPU Operator 버전 2.2에 대한 helm 차트를 다운로드합니다.

YOUR API KEY를 대체합니다.

```
helm fetchhttps://helm.ngc.nvidia.com/nvaie/charts/gpu-operator-2-2-v1.11.1.tgz--username=
'$oauthtoken' \
--password=<YOUR API KEY>
```

10 TKG 클러스터에 NVAIE GPU Operator 버전 2.2를 설치합니다.

```
helm install gpu-operator ./gpu-operator-2-2-v1.11.1.tgz -n gpu-operator
```

운영자 5단계: AI/ML 워크로드 배포

NVIDIA GPU Cloud 카탈로그는 vGPU 지원 Tanzu Kubernetes 클러스터에서 AI/ML 워크로드를 실행하는 데 사용할 수 있는 몇 가지 기본 제공 컨테이너 이미지를 제공합니다. 사용 가능한 이미지에 대한 자세한 내용은 NGC 설명서를 참조하십시오.

NVIDIA vGPU 디바이스에 대한 사용자 지정 VM 클래스 생성

NVIDIA Grid vGPU 디바이스에 대한 사용자 지정 VM 클래스를 생성하려면 이 항목을 참조하십시오.

vGPU 프로파일(v8 U2 P03 이상)을 사용하여 사용자 지정 VM 클래스 생성

NVIDIA vGPU(가상 그래픽 처리 장치)를 사용하면 여러 VM(가상 시스템)이 하나의 물리적 GPU를 공유할 수 있습니다. TKGS 클러스터에서 vGPU를 사용하려면 사용자 지정 VM 클래스를 정의합니다. 이 릴리스부터 사용자 지정 VM 클래스를 정의하기 위한 새로운 마법사가 제공됩니다. 사용자 지정 VM 클래스를 정의하기 위한 vGPU 프로파일(v8 U2 이하)을 사용하여 사용자 지정 VM 클래스 생성과 달리 vGPU 프로파일은 VM 클래스에서 구성되지 않고 디바이스에서 읽습니다.

VM 운영자는 vCenter 인벤토리를 폴링하여 감독자가 배포된 vSphere 클러스터를 구성하는 ESXi 호스트에 설치된 모든 vGPU 디바이스를 가져옵니다. vGPU 디바이스는 해당 프로파일을 정의합니다. vGPU 디바이스 이름은 프로파일이 다중 인스턴스 GPU(MIG)인지 아니면 시간 공유 GPU인지 나타냅니다. MIG는 계산을 분할하고 여러 워크로드를 단일 GPU에서 병렬로 실행할 수 있도록 지원합니다. 시간 공유는 GPU에 대한 공유 액세스를 제공합니다. MIG 모드는 최신 GPU 아키텍처를 기반으로 하며 NVIDIA A100 및 A30 디바이스에서만 지원됩니다. 자세한 내용은 NVIDIA 설명서를 참조하십시오.

예를 들어, GPU 디바이스 "grid-a100-40c"는 40GB 메모리가 있는 NVIDIA A100 GPU 디바이스를 VM에 할당하는 시간 공유 vGPU 프로파일을 제공합니다. 이에 상응하는 MIG 기반 vGPU 프로파일은 "grid-a100-7-40c" 디바이스입니다. 디바이스와 RAM 사이에 추가 번호가 있으므로 MIG 프로파일임을 식별할 수 있습니다. "7"은 GPU 디바이스에 7개의 계산 슬라이스가 있음을 나타냅니다. MIG 기반 vGPU 프로파일에는 1, 2, 3 또는 7개의 계산 슬라이스가 있을 수 있습니다.

1 vSphere Client 홈 메뉴에서 **워크로드 관리 > 서비스**를 선택합니다.

2 **VM 클래스** 탭을 선택합니다.

3 **VM 클래스 생성**을 클릭합니다.

이 작업을 수행하면 VM 클래스 생성을 안내하는 VM 클래스 생성 마법사가 시작됩니다.

4 **이름**에 VM 클래스의 이름을 입력하고 **다음**을 클릭합니다.

VM 클래스 이름은 VM 클래스를 식별합니다. 다음 요구 사항을 따르는 고유한 DNS 규정 준수 이름을 입력합니다.

- 사용자 환경에서 기본 또는 사용자 지정 VM 클래스의 이름과 중복되지 않는 고유한 이름을 사용합니다.
- 영숫자 문자열(최대 길이 63자)을 사용합니다.
- 대문자나 공백은 사용하지 마십시오.
- 대시는 첫 번째 또는 마지막 문자를 제외한 아무 곳에나 사용합니다. 예: **`vm-class1`**.
- VM 클래스를 생성한 후에는 이름을 변경할 수 없습니다.

5 **호환성**의 경우 **ESXi 8.0 U2 이상**을 선택하고 **다음**을 클릭합니다.

자세한 내용은 가상 시스템 호환성을 참조하십시오.

참고 VM 클래스가 생성된 후에는 하드웨어 호환성을 변경할 수 없습니다.

6 **구성 > 가상 하드웨어**의 경우 NVIDIA GPU 디바이스를 VM 클래스에 추가합니다.

a **구성 > 가상 하드웨어 > 새 디바이스 추가 > PCI 디바이스**를 선택합니다.

b 목록에서 원하는 NVIDIA Grid vGPU 디바이스를 선택합니다. NVIDIA Grid vGPU 프로파일에는 **시간 공유**와 **다중 인스턴스 GPU 공유**의 두 가지 유형이 있습니다. 디바이스를 선택하면 시스템에서 프로파일이 감지됩니다.

참고 VM 클래스에는 MIG 프로파일 유형의 NVIDIA GRID vGPU 디바이스를 하나만 추가할 수 있습니다.

c **선택**을 클릭하면 [가상 하드웨어] 탭에 **새 PCI 디바이스**가 나타납니다.

7 **구성 > 가상 하드웨어**의 경우 **CPU**, **메모리**, **새 PCI 디바이스**, **비디오 카드** 및 **보안 디바이스**에 대해 원하는 설정을 지정합니다.

표 13-3. CPU 구성

설정	구성
CPU	VM의 가상 CPU 수를 선택합니다. 자세한 내용은 가상 CPU 구성 및 제한 사항을 참조하십시오.
CPU 토폴로지	전원을 켤 때 할당됨
예약	예약은 0에서 10MHz 사이여야 합니다.
제한	제한은 10MHz보다 크거나 같아야 합니다.
공유	옵션은 낮음, 보통, 높음, 사용자 지정입니다.
하드웨어 가상화	게스트 운영 체제에 하드웨어 지원 가상화를 노출하려면 이 옵션을 선택합니다.
성능 카운터	가상화된 CPU 성능 카운터 사용

표 13-3. CPU 구성 (계속)

설정	구성
스케줄링 선호도	이 가상 시스템에 대한 물리적 프로세서 선호도를 선택합니다. 범위를 나타내려면 '-'를 사용하고 값을 구분하려면 ','를 사용합니다. 예를 들어 "0, 2, 4-7"은 프로세서 0, 2, 4, 5, 6 및 7을 나타냅니다. 선호도 설정을 제거하려면 문자열을 지우십시오.
I/O MMU	메모리 관리 단위(페이지 - 디스크)를 사용하도록 설정하려면 선택합니다.

표 13-4. 메모리 구성

설정	구성
메모리	VM의 메모리 크기를 선택합니다. 자세한 내용은 최대 가상 시스템 메모리를 참조하십시오.
예약	가상 시스템에 보장된 최소 할당을 지정하거나 모든 게스트 메모리를 예약합니다. 예약이 충족되지 않으면 VM을 실행할 수 없습니다.
제한	제한할 메모리 양을 선택하여 VM의 메모리 소비량에 제한을 설정합니다.
공유	공유할 메모리 양을 선택합니다. 공유는 메모리 용량 할당을 위한 상대적인 메트릭을 나타냅니다. 자세한 내용은 메모리 공유를 참조하십시오.
메모리 핫 플러그	전원이 켜진 VM에 메모리 리소스를 추가할 수 있도록 허용하려면 사용하도록 설정(선택)합니다. 자세한 내용은 메모리 무중단 추가 설정을 참조하십시오.

표 13-5. 새 PCI 디바이스 > GPU 공유 구성

시간 공유 모드	MIG 모드
시간 공유 모드에서 vGPU 스케줄러는 vGPU 간에 성능을 밸런싱하는 최선의 목표로 일정 기간 동안 각 vGPU 지원 VM에 대해 직렬로 작업을 수행하도록 GPU에 지시합니다.	MIG 모드에서는 여러 vGPU 지원 VM을 단일 GPU 디바이스에서 병렬로 실행할 수 있습니다. MIG 옵션이 표시되지 않으면 선택한 PCI 디바이스가 MIG 옵션을 지원하지 않는 것입니다.

표 13-6. 비디오 카드 구성

설정	구성
비디오 카드	하드웨어에서 설정을 자동으로 감지하도록 선택하거나 사용자 지정 설정을 입력합니다. 자동 감지를 선택하면 다른 설정은 구성할 수 없습니다.
디스플레이 수	디스플레이 수를 선택합니다.
총 비디오 메모리	총 비디오 메모리(MB)를 입력합니다.
3D 그래픽	3D 지원 사용하려면 선택합니다.

표 13-7. 보안 디바이스 구성

설정	구성
보안 디바이스	SGX 보안 디바이스가 설치된 경우 여기에서 VM 설정을 구성할 수 있습니다. 그렇지 않으면 이 필드를 구성할 수 없습니다. 자세한 내용은 SGX 설명서를 참조하십시오.

8 **구성 > VM 옵션** 탭을 선택하고 추가적인 VM 설정을 구성합니다. 지침은 가상 시스템 옵션 구성을 참조하십시오.

9 **구성 > 고급 매개 변수** 탭을 선택하고 VM 클래스에 대한 특성을 추가합니다.

10 **다음**을 클릭합니다.

11 **검토 및 확인** 페이지에서 세부 정보를 검토하고 **마침**을 클릭합니다.

12 새 VM 클래스를 vSphere 네임스페이스와 연결합니다. VM 클래스를 vSphere 네임스페이스와 연결의 내용을 참조하십시오.

그림 13-1. NVIDIA vGPU 디바이스 선택

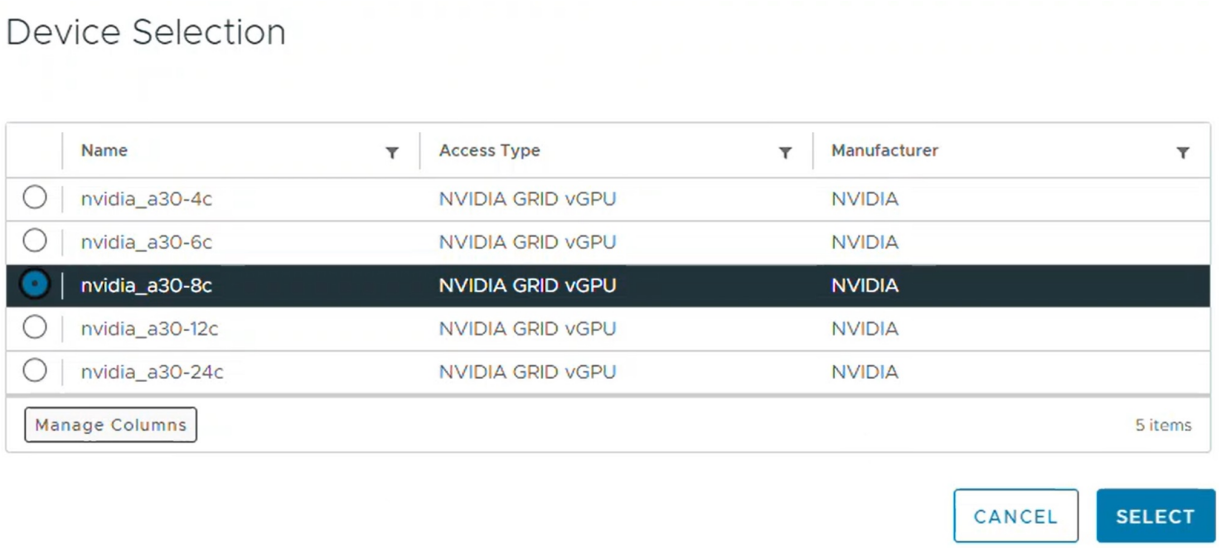
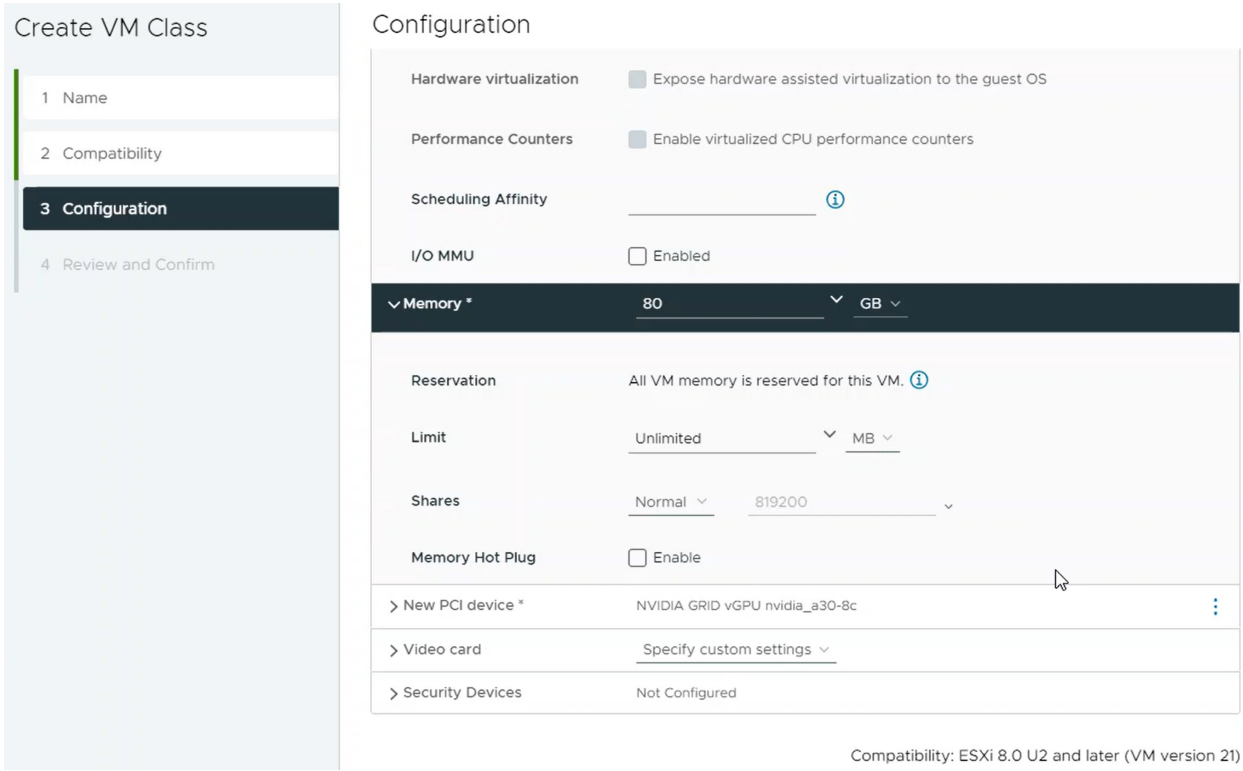


그림 13-2. NVIDIA vGPU 새 PCI 디바이스



vGPU 프로파일(v8 U2 이하)을 사용하여 사용자 지정 VM 클래스 생성

다음 단계에서는 vGPU 프로파일을 사용하여 사용자 지정 VM 클래스를 생성합니다. 시스템에서는 TKG 클러스터 노드를 생성할 때 이 클래스 정의를 사용합니다.

아래 지침에 따라 vGPU 프로파일을 사용하여 사용자 지정 VM 클래스를 생성합니다.

1 vSphere Client를 사용하여 vCenter Server에 로그인합니다.

2 **워크로드 관리**를 선택합니다.

3 **서비스**를 선택합니다.

4 **VM 클래스**를 선택합니다.

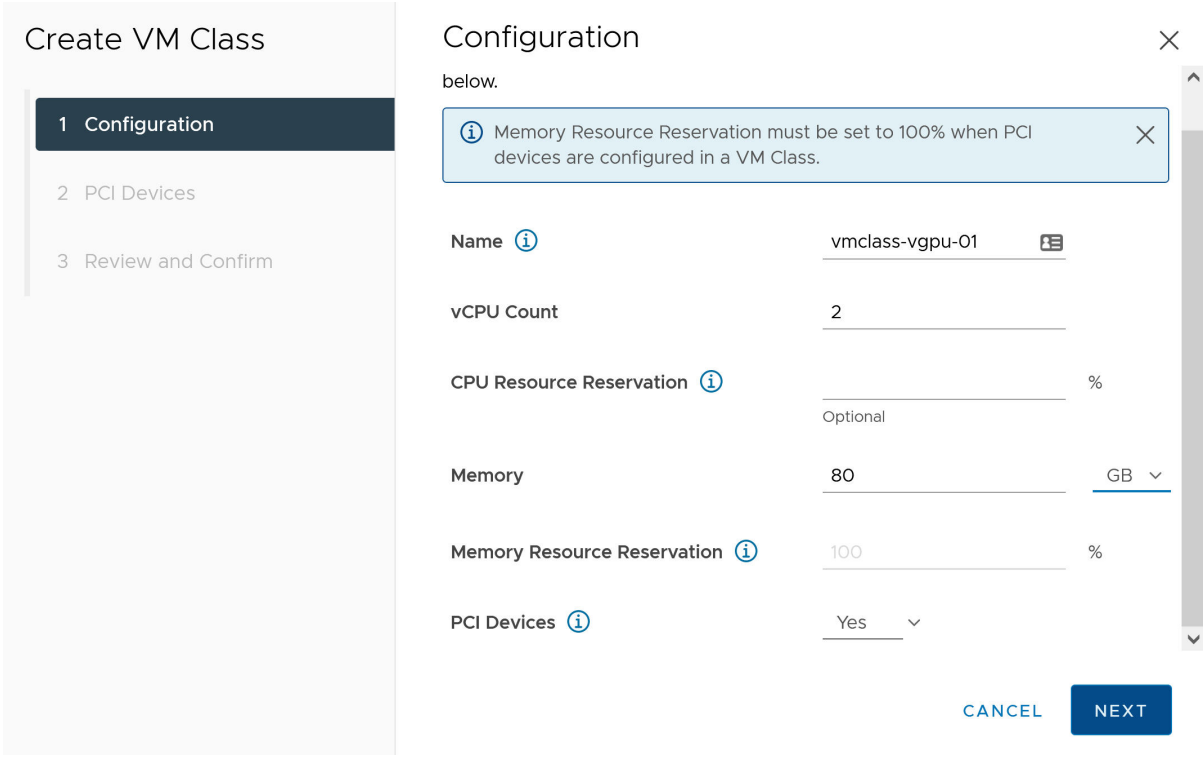
5 **VM 클래스 생성**을 클릭합니다.

6 **구성** 탭에서 사용자 지정 VM 클래스를 구성합니다.

구성 필드	설명
이름	사용자 지정 VM 클래스에 대한 자체 설명 이름을 입력합니다(예: **`vmclass-vgpu-1`**).
vCPU 수	**2**
CPU 리소스 예약	선택 사항, 비워 둘 수 있음
메모리	예: **80GB**

구성 필드	설명
메모리 리소스 예약	**100%**(PCI 디바이스가 VM 클래스에 구성된 경우 필수)
PCI 디바이스	**예** **참고** PCI 디바이스에 대해 [예]를 선택하면 GPU 디바이스를 사용 중임을 시스템에 알리고 vGPU 구성을 지원하도록 VM 클래스 구성이 변경됩니다. 자세한 내용은 vSphere with Tanzu의 VM 클래스에 PCI 디바이스 추가를 참조하십시오.

예:



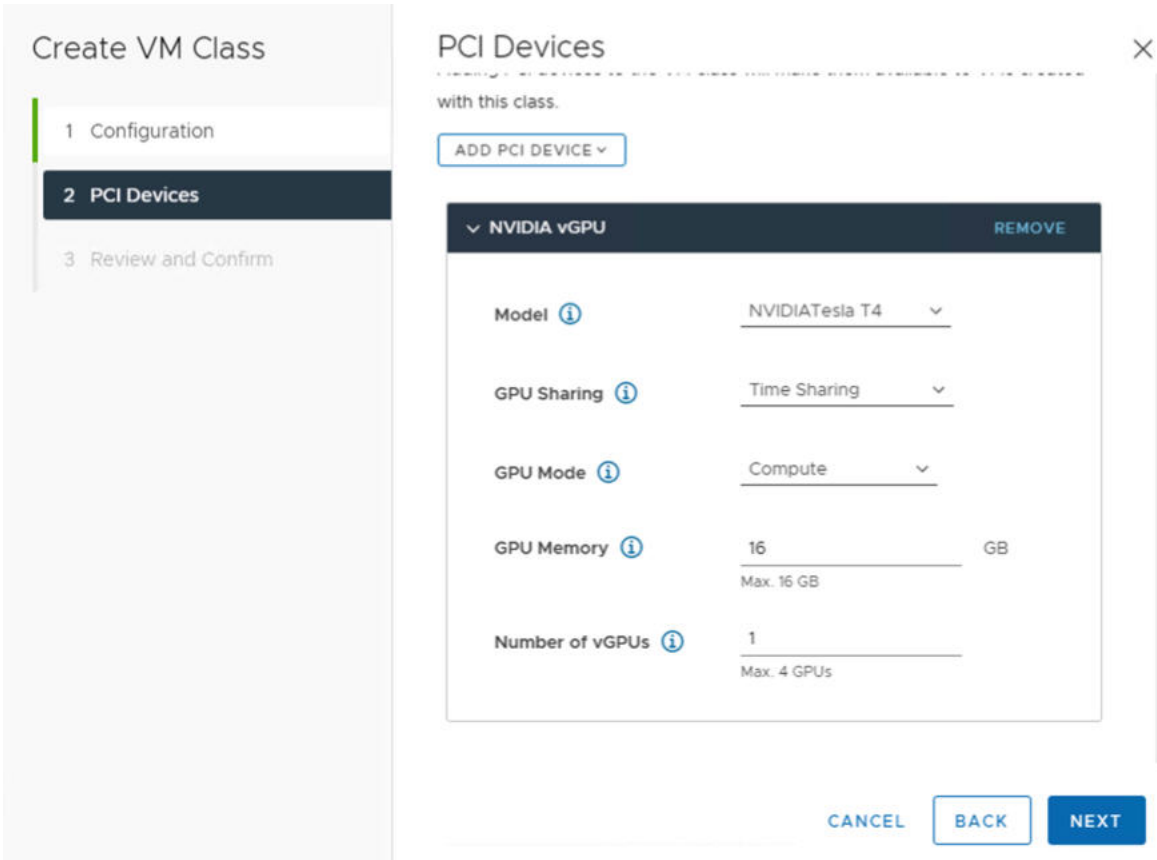
7 **다음**을 클릭합니다.

8 **PCI 디바이스** 탭에서 **PCI 디바이스 추가 > NVIDIA vGPU** 옵션을 선택합니다.

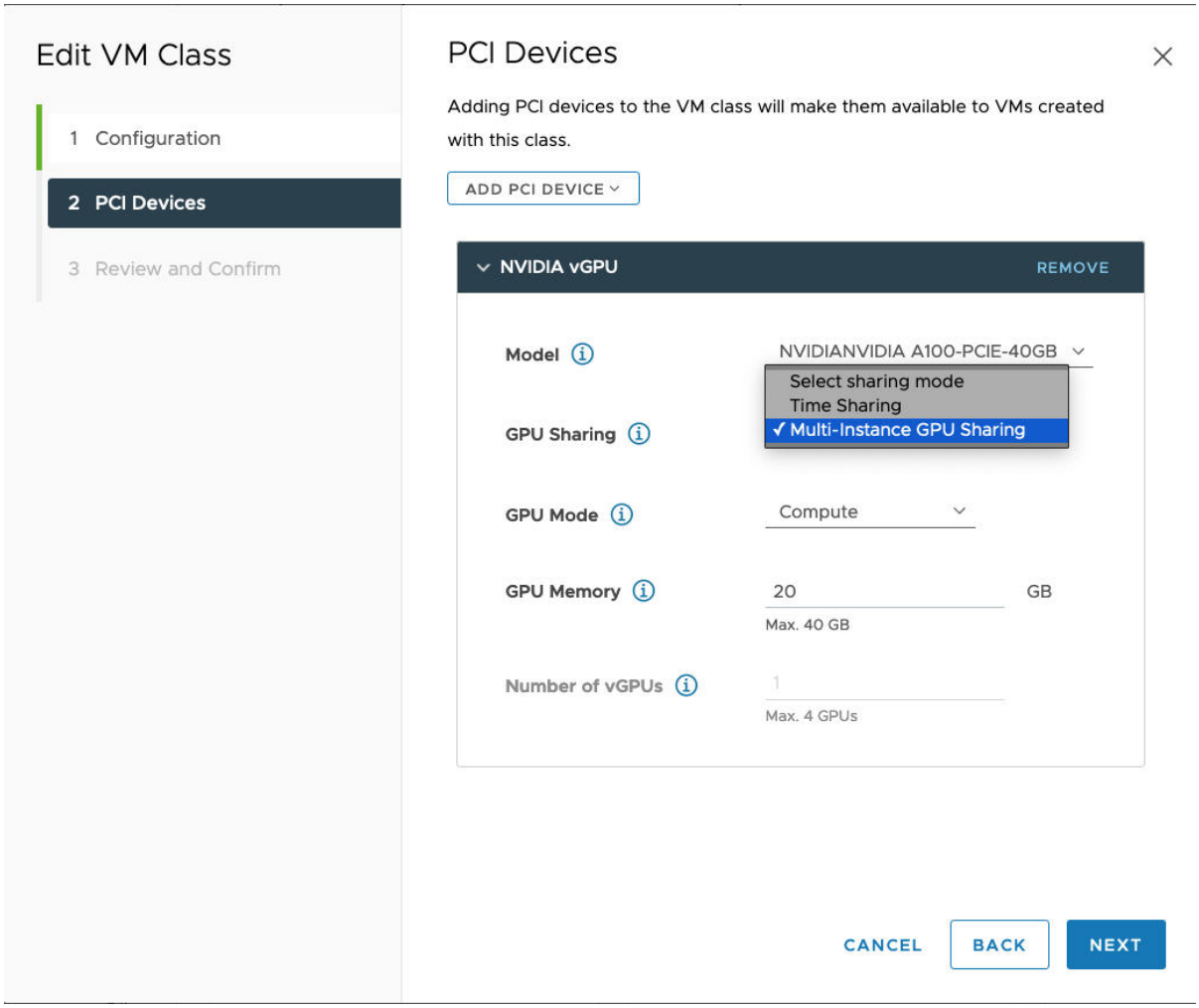
9 NVIDIA vGPU 모델을 구성합니다.

NVIDIA vGPU 필드	설명
모델	**NVIDIA vGPU > 모델** 메뉴에서 사용 가능한 NVIDIA GPU 하드웨어 디바이스 모델을 선택합니다. 시스템에 프로파일이 표시되지 않으면 클러스터의 어떤 호스트에도 지원되는 PCI 디바이스가 없는 것입니다.
GPU 공유	이 설정은 GPU 디바이스가 GPU 지원 VM 간에 공유되는 방식을 정의합니다. vGPU 구현에는 **시간 공유**와 **다중 인스턴스 GPU 공유**의 두 가지 유형이 있습니다. 시간 공유 모드에서 vGPU 스케줄러는 vGPU 간에 성능을 밸런싱하는 최선의 목표로 일정 기간 동안 각 vGPU 지원 VM에 대해 직렬로 작업을 수행하도록 GPU에 지시합니다. MIG 모드에서는 여러 vGPU 지원 VM을 단일 GPU 디바이스에서 병렬로 실행할 수 있습니다. MIG 모드는 최신 GPU 아키텍처를 기반으로 하며 NVIDIA A100 및 A30 디바이스에서만 지원됩니다. MIG 옵션이 표시되지 않으면 선택한 PCI 디바이스가 MIG 옵션을 지원하지 않는 것입니다.
GPU 모드	**계산**
GPU 메모리	예: **8GB**
vGPU 수	예: **1**

예를 들어 시간 공유 모드로 구성된 NVIDIA vGPU 프로파일은 다음과 같습니다.



예를 들어 지원되는 GPU 디바이스에서 MIG 모드로 구성된 NVIDIA vGPU 프로파일은 다음과 같습니다.



10 **다음**을 클릭합니다.

11 선택 항목을 검토하고 확인합니다.

12 **마침**을 클릭합니다.

13 새 사용자 지정 VM 클래스를 VM 클래스 목록에서 사용할 수 있는지 확인합니다.

동적 DirectPath IO가 있는 vGPU

동적 DirectPath IO가 있는 vGPU를 사용하는 경우 다음 추가 구성을 완료합니다. **동적 DirectPath IO**를 지정하고 지원되는 PCI 디바이스를 선택하여 생성한 사용자 지정 VM 클래스에 두 번째 PCI 디바이스 구성을 추가합니다. 이 유형의 VM 클래스가 인스턴스화되면 vSphere DRS(Distributed Resource Scheduler)에 따라 VM 배치가 결정됩니다.

1 **워크로드 관리**를 선택합니다.

2 **서비스**를 선택합니다.

3 **VM 클래스**를 선택합니다.

4 **NVIDIA vGPU** 프로파일로 이미 구성된 사용자 지정 VM 클래스를 편집합니다.

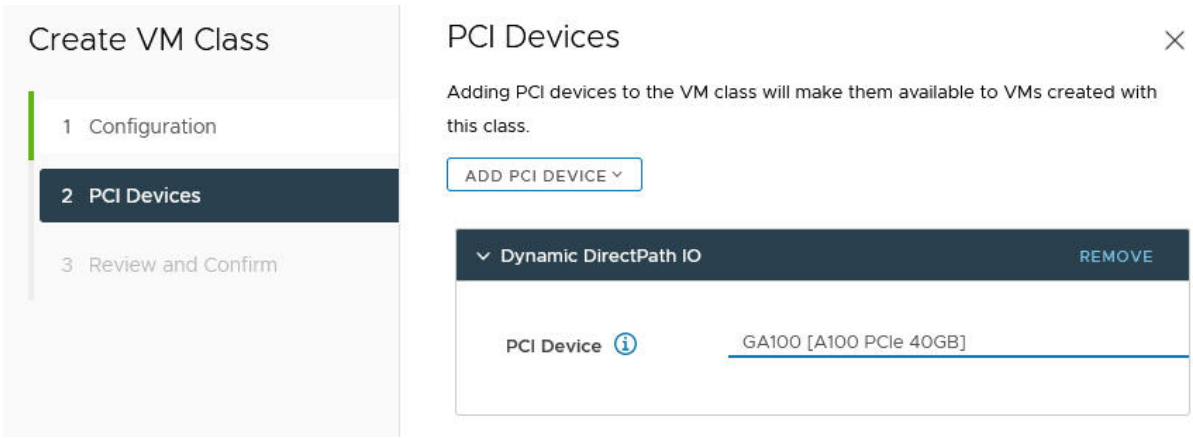
5 **PCI 디바이스** 탭을 선택합니다.

6 **PCI 디바이스 추가**를 클릭합니다.

7 **동적 DirectPath IO** 옵션을 선택합니다.

8 **PCI 디바이스**를 선택합니다.

예:



9 **다음**을 클릭합니다.

10 선택 항목을 검토하고 확인합니다.

11 **마침**을 클릭합니다.

12 새 사용자 지정 VM 클래스를 VM 클래스 목록에서 사용할 수 있는지 확인합니다.

TKG 서비스 클러스터에서 개인 레지스트리 사용

14

컨테이너 레지스트리는 Kubernetes 운영자에게 컨테이너 이미지를 저장하고 공유하기 편리한 저장소를 제공합니다. 이 섹션에서는 TKG 서비스 클러스터에서 개인 레지스트리를 사용하는 방법에 대한 설명서를 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합
- 개인 레지스트리 자격 증명 암호 생성
- 개인 레지스트리의 컨테이너 이미지에서 포드 생성
- Harbor 감독자 서비스 설치
- Harbor 레지스트리 인증서를 사용하여 Docker 클라이언트 구성
- 표준 패키지를 개인 Harbor 레지스트리로 푸시

TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합

TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합하려면 이 항목을 참조하십시오.

개인 컨테이너 레지스트리 사용 사례

컨테이너 레지스트리는 컨테이너 이미지를 저장 및 공유하기 위한 중앙 집중식 저장소 역할을 하는 Kubernetes 배포에 중요한 기능을 제공합니다. 가장 일반적으로 사용되는 공용 컨테이너 레지스트리는 Docker Hub입니다. 개인 컨테이너 레지스트리 오퍼링은 많이 있습니다. VMware Harbor는 감독자와 함께 제공되는 오픈 소스, 클라우드 네이티브, 개인 컨테이너 레지스트리입니다.

개인 컨테이너 레지스트리 통합

개인 레지스트리를 TKG 서비스 클러스터와 통합하려면 HTTPS를 통해 개인 레지스트리 컨텐츠를 제공하도록 하나 이상의 자체 서명된 CA 인증서로 클러스터를 구성합니다. 이렇게 하려면 `additionalTrustedCAs` 값이 있는 `trust` 필드를 클러스터 규격에 포함합니다. TKGS 클러스터가 신뢰해야 하는 자체 서명된 CA 인증서는 원하는 수만큼 정의할 수 있습니다. 이 기능을 사용하면 인증서 목록을 쉽게 정의하고 순환이 필요한 인증서를 업데이트할 수 있습니다.

처음부터 클러스터를 생성할 때 개인 레지스트리 인증서를 구성하거나, 기존 클러스터를 업데이트하고 개인 레지스트리 인증서를 제공할 수 있습니다. 기존 클러스터를 편집하고 개인 레지스트리 인증서를 추가하려면 `kubectl edit` 메서드를 사용합니다. Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

`trust.additionalTrustedCAs` 필드의 구현은 TKGS 클러스터 프로비저닝을 위해 지원되는 API마다 조금씩 다릅니다. 자세한 내용은 표 14-1. v1alpha3 API 신뢰 필드 및 표 14-2. v1beta1 API 신뢰 변수의 내용을 참조하십시오.

v1alpha3 API 예시

다음 예에서는 v1alpha3 API를 사용하여 프로비저닝된 TKG 서비스 클러스터를 CA 인증서를 사용하여 개인 컨테이너 레지스트리와 통합하는 방법을 보여줍니다.

TanzuKubernetesCluster v1alpha3 API를 사용할 경우, `trust.additionalTrustedCAs` 필드에는 각각 개인 레지스트리에 대한 TLS 인증서를 포함할 수 있는 하나 이상의 이름 데이터 쌍이 포함됩니다.

표 14-1. v1alpha3 API 신뢰 필드

필드	설명
`trust`	섹션 마커. 데이터를 허용하지 않습니다.
`additionalTrustedCAs`	섹션 마커. 각각에 대해 `name`과 `data`가 있는 인증서 어레이를 포함합니다.
`name`	CA 인증서의 사용자 정의 이름입니다.
`data`	이중 base64로 인코딩된 PEM 형식의 CA 인증서(`ca.crt`) 컨텐츠입니다. **참고** v1alpha3 API를 사용하려면 인증서 컨텐츠가 단일 base64로 인코딩되어야 합니다. 컨텐츠가 단일 base6로 인코딩되지 않은 경우 결과 PEM 파일을 처리할 수 없습니다.

다음 절차에 따라 Harbor 레지스트리 인증서를 사용하여 Harbor를 v1alpha3 API 클러스터와 통합합니다.

1 **프로젝트 > 저장소** 화면의 Harbor 웹 인터페이스에서 Harbor **레지스트리 인증서**를 다운로드합니다.

CA 인증서 파일이 `ca.crt`로 다운로드됩니다.

2 CA 인증서의 컨텐츠를 단일 base64로 인코딩합니다.

- Linux: `base64 -w 0 ca.crt`
- Windows: https://www.base64encode.org/.

3 클러스터 규격에 `trust.additionalTrustedCAs` 필드를 포함하고 `name` 및 `data` 값으로 채웁니다.

```
#tkc-with-trusted-private-reg-cert.yaml
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc01
  namespace: tkgs-cluster-ns
```

```
spec:
   topology:
     controlPlane:
       replicas: 3
       storageClass: tkgs-storage-policy
       vmClass: guaranteed-medium
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
     nodePools:
     - name: nodepool-01
       replicas: 3
       storageClass: tkgs-storage-policy
       vmClass: guaranteed-medium
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
   settings:
     storage:
       defaultClass: tkgs-storage-policy
     network:
       cni:
         name: antrea
       services:
         cidrBlocks: ["198.51.100.0/12"]
       pods:
         cidrBlocks: ["192.0.2.0/16"]
       serviceDomain: cluster.local
       trust:
         additionalTrustedCAs:
         - name: CompanyInternalCA-1
           data: LS0tLS1C...LS0tCg==
         - name: CompanyInternalCA-2
           data: MTLtMT1C...MT0tPg==
```

4 인증서를 순환하려면 클러스터 규격에 대해 `kubectl edit`을 실행하고 `trust.additionalTrustedCAs.data` 값을 업데이트한 다음 롤링 업데이트를 시작합니다.

v1beta1 API 예시

다음 예에서는 v1beta1 API를 사용하여 프로비저닝된 TKG 서비스 클러스터를 CA 인증서를 사용하여 개인 컨테이너 레지스트리와 통합하는 방법을 설명합니다.

개인 컨테이너 레지스트리를 Cluster v1beta1 API 로 프로비저닝된 TKGS 클러스터와 통합하려면 trust변수를 사용하고 변수를 개인 레지스트리 인증서가 포함된 Kubernetes 암호로 채웁니다.

표 14-2. v1beta1 API 신뢰 변수

필드	설명
`trust`	섹션 마커. 데이터를 허용하지 않습니다.
`additionalTrustedCAs`	섹션 마커. 각각에 대한 이름이 있는 인증서 어레이를 포함합니다.
`name`	이중 base64로 인코딩된 PEM 형식의 CA 인증서가 포함된 Kubernetes 암호의 `data` 맵 필드에 대한 사용자 정의 이름입니다. **참고** v1beta1 API를 사용하려면 인증서 컨텐츠가 이중 base64로 인코딩되어야 합니다. 컨텐츠가 이중 base6로 인코딩되지 않은 경우 결과 PEM 파일을 처리할 수 없습니다.

다음 절차에 따라 Harbor 레지스트리 인증서를 사용하여 Harbor를 v1beta1 API 클러스터와 통합합니다.

1 **프로젝트 > 저장소** 화면의 Harbor 웹 인터페이스에서 Harbor **레지스트리 인증서**를 다운로드합니다.

CA 인증서 파일이 `ca.crt`로 다운로드됩니다.

2 CA 인증서의 컨텐츠를 이중 base64로 인코딩합니다.

- Linux: `base64 -w 0 ca.crt | base64 -w 0`
- Windows: https://www.base64encode.org/.

3 다음 컨텐츠로 Kubernetes 암호 정의 YAML 파일을 생성합니다.

```
#additional-ca-1.yaml
apiVersion: v1
data:
  additional-ca-1: TFMwdExTMUNSGlSzZ3Jaa...VVNVWkpRMEMwdExTMHRDZz09
kind: Secret
metadata:
  name: cluster01-user-trusted-ca-secret
  namespace: tkgs-cluster-ns
type: Opaque
```

형식 설명:

- 암호의 `data` 맵 값은 CA 인증서의 이름(이 예에서는 `additional-ca-1`)인 사용자 정의 문자열이며, 값은 이중 base64로 인코딩된 인증서입니다.
- `metadata` 섹션에서 암호의 이름을 *CLUSTER-NAME*`-user-trusted-ca-secret`으로 지정합니다. 여기서 *CLUSTER-NAME*은 클러스터의 이름입니다. 이 암호는 클러스터와 동일한 vSphere 네임스페이스에서 생성해야 합니다.

4 Kubernetes 암호를 선언적으로 생성합니다.

```
kubectl apply -f additional-ca-1.yaml
```

5 암호 생성을 확인합니다.

```
kubeclt get secret -n tkgs-cluster-ns cluster01-user-trusted-ca-secret
NAME                                          TYPE     DATA   AGE
cluster01-user-trusted-ca-secret              Opaque   12     2d22h
```

6 암호에 대한 데이터 맵의 이름(이 예에서는 `additional-ca-1`)을 참조하는 클러스터 규격에 `trust` 변수를 포함합니다.

```
#cluster-with-trusted-private-reg-cert.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster01
  namespace: tkgs-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.52.100.0/12"]
    pods:
      cidrBlocks: ["192.101.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.26.5+vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-01
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkgs-storage-profile
      - name: defaultStorageClass
        value: tkgs-storage-profile
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
```

7 인증서를 순환하려면 새 암호를 생성하고 적절한 값으로 클러스터 규격을 편집합니다. 이렇게 하면 클러스터의 롤링 업데이트가 트리거됩니다.

참고 시스템은 *CLUSTER-NAME*-user-trusted-ca-secret에 대한 변경 내용을 모니터링하지 않습니다. `data` 맵 값이 변경되어도 이러한 변경 내용이 클러스터에 반영되지 않습니다. `trust.additionalTrustCAs`에 대한 새 암호와 해당 데이터 맵 `name`을 생성해야 합니다.

개인 레지스트리 자격 증명 암호 생성

오류 없이 컨테이너 이미지를 끌어올 수 있도록 포드 및 배포 규격에서 레지스트리 자격 증명 암호 및 참조를 생성합니다.

Docker Hub에는 이미지를 끌어오기 위한 계정이 필요합니다. Docker Hub 자격 증명을 사용하여 Kubernetes 암호를 생성하고 포드 및 배포 규격에서 이 암호를 참조할 수 있습니다.

이 방법은 다른 개인 레지스트리에도 사용할 수 있습니다.

사전 요구 사항

Ubuntu 클라이언트 시스템에 Docker를 설치합니다. https://docs.docker.com/engine/install/ubuntu/의 내용을 참조하십시오.

절차

1 `docker version`을 실행하고 Docker가 설치되어 있는지 확인합니다.

2 `docker login -u USERNAME`을 실행합니다.

3 Docker Hub 암호를 입력합니다.

4 `cat ~/.docker/config.json`을 실행합니다.

5 다음 명령을 실행하여 Docker Hub 액세스 자격 증명이 포함된 `regcred`라는 일반 암호를 생성합니다.

```
kubectl create secret generic regcred \
    --from-file=.dockerconfigjson=/home/ubuntu/.docker/config.json \
    --type=kubernetes.io/dockerconfigjson
```

`secret/regcred created`이 표시되어야 합니다.

6 암호를 확인합니다.

```
kubectl get secrets

NAME                  TYPE                                  DATA   AGE
default-token-w7wqk   kubernetes.io/service-account-token   3      6h28m
regcred               kubernetes.io/dockerconfigjson        1      3h22m
```

YAML에서 regcred 암호를 참조합니다.

7 컨테이너 이미지에 대한 배포 규격 또는 포드의 `imagePullSecrets` 섹션에서 `regcred` 암호를 참조합니다.

예:

```
apiVersion: v1
kind: Pod
metadata:
  name: ping-pod
  namespace: default
spec:
  containers:
```

```
  - image: busybox:1.34
    name: busybox
    command: ["ping", "-c"]
    args: ["1", "8.8.8.8"]
  imagePullSecrets:
  - name: regcred
  restartPolicy: Never
```

개인 레지스트리의 컨테이너 이미지에서 포드 생성

TKG 서비스 클러스터의 개인 컨테이너 레지스트리에서 이미지를 끌어오려면 개인 레지스트리 세부 정보를 사용하여 워크로드 YAML을 구성합니다.

이 절차는 Harbor 레지스트리와 같은 개인 컨테이너 레지스트리에서 이미지를 끌어오는 데 사용할 수 있습니다. 이 예에서는 Harbor 레지스트리에 저장된 이미지를 사용하고 이전에 구성된 이미지 끌어오기 암호를 활용하는 포드 규격을 생성합니다.

사전 요구 사항

레지스트리 자격 증명 암호를 생성합니다. 개인 레지스트리 자격 증명 암호 생성 항목을 참조하십시오.

절차

1 TKG 클러스터를 프로비저닝합니다.

Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

2 클러스터에 로그인합니다.

kubectl을 사용하여 vCenter Single Sign-On 사용자로 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

3 레지스트리 자격 증명 암호를 생성합니다.

개인 레지스트리 자격 증명 암호 생성의 내용을 참조하십시오.

4 개인 레지스트리에 대한 세부 정보를 사용하여 예제 포드 규격을 생성합니다.

`pod-example.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: <workload-name>
  namespace: <kubernetes-namespace>
spec:
  containers:
  - name: private-reg-container
    image: <Registry-IP-or-FQDN>/<vsphere-namespace>/<image-name>:<version>
  imagePullSecrets:
  - name: <registry-secret-name>
```

- `<workload-name>`을 포드 워크로드의 이름으로 바꿉니다.

- `<kubernetes-namespace>`는 포드를 생성할 클러스터의 Kubernetes 네임스페이스로 바꿉니다. 이것은 레지스트리 서비스 이미지 끌어오기 암호가 클러스터 Tanzu Kubernetes에 저장되는 동일한 Kubernetes 네임스페이스(예: 기본 네임스페이스)여야 합니다.
- `<Registry-IP-or-FQDN>`를 감독자에서 실행되는 Harbor 레지스트리 인스턴스의 IP 주소 또는 FQDN으로 바꿉니다.
- *<vsphere-namespace>*는 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 바꿉니다.
- `<image-name>`은 원하는 이미지 이름으로 바꿉니다.
- `<version>`은 적절한 이미지 버전(예: "latest")으로 바꿉니다.
- `<registry-secret-name>`은 이전에 생성한 레지스트리 서비스 이미지 끌어오기 암호의 이름으로 바꿉니다.

5 정의한 포드 규격을 사용하여 워크로드를 생성합니다.

```
kubectl apply -f pod-example.yaml
```

포드는 레지스트리에서 끌어온 이미지에서 생성해야 합니다.

Harbor 감독자 서비스 설치

Harbor 컨테이너 레지스트리를 감독자 서비스로 설치하고 Harbor를 개인 레지스트리로 실행할 수 있습니다.

사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- vSphere 8 감독자를 사용하도록 설정합니다.
- 감독자 서비스를 숙지합니다.

참고 이러한 지침은 vSphere 8 및 NSX 4 네트워킹을 통해 검증됩니다.

필요한 YAML 파일 다운로드

Contour 및 Harbor를 포함하여 필요한 YAML 파일을 다운로드합니다.

1 Kubernetes 수신 컨트롤러 서비스 사이트에서 다음 Contour 파일을 다운로드합니다.

a Contour 서비스 정의 파일: `contour.yml`

b Contour 서비스 구성 파일: `contour-data-values.yml`

2 클라우드 네이티브 레지스트리 서비스 사이트에서 다음 Harbor 파일을 다운로드합니다.

a Harbor 서비스 정의 파일: `harbor.yml`

b Harbor 서비스 구성 파일: `harbor-data-values.yml`

Contour 설치

Harbor를 설치하기 전에 먼저 Contour를 설치해야 합니다.

1 **워크로드 관리 > > 서비스 > 추가**에서 `contour.yml`을 vCenter에 업로드합니다.

2 Contour 서비스 정의가 추가되었는지 확인합니다.

3 **워크로드 관리 > 감독자 > 감독자 > 구성**을 선택합니다.

4 **감독자 서비스 > 개요**를 선택합니다.

5 **사용 가능** 탭을 선택합니다.

6 **Contour**를 선택하고 **설치**를 클릭합니다.

7 `contour-data-values.yml`의 컨텐츠를 복사한 후 "YAML 서비스 구성" 입력 필드에 붙여 넣습니다.

참고 Contour 데이터 값은 있는 그대로 사용할 수 있으며 구성을 변경할 필요가 없습니다. Envoy의 서비스 유형은 LoadBalancer 로 설정됩니다.

8 **확인**을 클릭하여 Contour 설치를 계속 진행합니다.

9 Contour가 설치되어 있는지 확인합니다.

a 이름이 *svc-contour-domain-XXXX*인 vSphere 네임스페이스를 선택합니다.

b **네트워크** 탭을 선택하고 **서비스**를 선택합니다.

c ClusterIP 유형의 Contour 서비스와 LoadBalancer 유형의 엔보이 서비스가 표시됩니다. 엔보이 서비스에는 외부 IP 주소가 있어야 합니다.

Harbor 데이터 값 업데이트

Harbor를 설치하기 전에 데이터 값 파일을 업데이트합니다.

1 텍스트 편집기로 `harbor-data-values.yml` 파일을 엽니다.

2 다음과 같이 편집합니다. 이 내용은 반드시 편집해야 하며 다른 필드는 필요한 경우 편집할 수 있습니다.

3 변경 내용을 저장합니다.

이름	값
hostname	harbordomain.com (고유한 항목 선택)
tlsCertificate.tlsSecretLabels	{"managed-by": "vmware-vRegistry"} (이 값을 확인하고 그대로 유지)
persistence.persistentVolumeClaim.registry.storageClass	"vwt-storage-policy" (감독자에 대한 vSphere 스토리지 정책 이름 입력)
persistence.persistentVolumeClaim.jobservice.storageClass	"vwt-storage-policy" (감독자에 대한 vSphere 스토리지 정책 이름 입력)
persistence.persistentVolumeClaim.database.storageClass	"vwt-storage-policy" (감독자에 대한 vSphere 스토리지 정책 이름 입력)

이름	값
persistence.persistentVolumeClaim.redis.storageClass	"vwt-storage-policy" (감독자에 대한 vSphere 스토리지 정책 이름 입력)
persistence.persistentVolumeClaim.trivy.storageClass	"vwt-storage-policy" (감독자에 대한 vSphere 스토리지 정책 이름 입력)

Harbor 설치

다음 지침을 완료하여 Harbor를 설치합니다.

1 **워크로드 관리 > > 서비스 > 추가**에서 `harbor.yml`을 vCenter에 업로드합니다.

2 Harbor 서비스 정의가 추가되었는지 확인합니다.

3 **워크로드 관리 > 감독자 > 감독자 > 구성**을 선택합니다.

4 **감독자 서비스 > 개요**를 선택합니다.

5 **사용 가능** 탭을 선택합니다.

6 **Harbor**를 선택하고 **설치**를 클릭합니다.

7 편집한 `harbor-data-values.yml`의 컨텐츠를 복사한 후 "YAML 서비스 구성" 입력 필드에 붙여 넣습니다.

8 **확인**을 클릭하여 Harbor 설치를 계속 진행합니다.

9 Harbor가 설치되어 있는지 확인합니다.

   a 이름이 *svc-harbor-domain-XXXX*인 vSphere 네임스페이스를 선택합니다.

   b **네트워크** 탭을 선택하고 **서비스**를 선택합니다.

   c Harbor용으로 설치된 여러 컨테이너가 표시됩니다(각 서비스는 ClusterIP 유형임).

Harbor에 대한 DNS 구성

도메인 이름을 등록하고 Harbor에 대한 DNS 레코드를 설정해야 합니다.

1 **워크로드 관리 > 네임스페이스**를 선택합니다.

2 **Contour** 네임스페이스를 선택합니다.

3 **네트워크 > 서비스**를 선택합니다.

4 엔보이 수신 서비스의 외부 IP 주소(예: 10.197.154.71)를 기록합니다.

5 Harbor 구성에서 지정한 FQDN(Harbor 도메인 이름)을 등록합니다.

6 AWS Route 53 또는 유사한 서비스를 사용하여 DNS "A" 레코드를 생성합니다.

Harbor에 로그인

Harbor DNS가 설정되면 로그인합니다.

1 Harbor에 대해 등록한 도메인 이름으로 이동합니다.

2 Harbor 구성에서 지정한 관리자 | 암호를 사용하여 도메인에 로그인합니다.

3 로그인 후 더 안전한 암호로 변경합니다.

Harbor 레지스트리를 신뢰하도록 감독자 구성(선택 사항)

TKG 클러스터는 TKG 클러스터와 Harbor가 동일한 감독자에 배포될 때 Harbor 감독자 서비스를 신뢰하도록 자동 구성됩니다. 그러나 감독자는 vSphere 포드를 생성할 때 Harbor 감독자 서비스를 신뢰하도록 자동 구성되지 않습니다. Harbor CA 인증서로 configmap을 업데이트하여 감독자 및 Harbor 서비스 간에 신뢰를 설정하려면 다음 단계를 완료합니다.

1 Harbor에서 **관리 > 구성 > 시스템 설정**으로 이동합니다.

2 파일 이름이 `ca.crt`인 레지스트리 루트 인증서를 다운로드합니다.

3 KUBE_EDITOR 환경 변수를 구성합니다.

Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

4 kubectl을 사용하여 감독자에 로그인합니다.

kubectl을 사용하여 vCenter Single Sign-On 사용자로 감독자에 연결의 내용을 참조하십시오.

5 컨텍스트를 감독자 컨텍스트(IP 주소)로 전환합니다.

6 다음 명령을 사용하여 configmap/image-fetch-ca-bundle 을 편집합니다.

```
kubectl edit configmap image-fetcher-ca-bundle -n kube-system
```

7 Harbor ca.crt 파일의 컨텐츠를 복사하고 기존 인증서(감독자용으로, 변경해서는 안 됨) 아래에 configmap을 추가합니다. 파일에 대한 편집 내용을 저장합니다. Kubectl이 "configmap/image-fetcher-ca-bundle edited"를 보고하는 것을 볼 수 있습니다.

Harbor 레지스트리 인증서를 사용하여 Docker 클라이언트 구성

Docker를 사용하여 레지스트리의 컨테이너 이미지로 작업하려면 레지스트리 인증서를 Docker 클라이언트에 추가합니다. 인증서는 레지스트리 로그인 중에 Docker를 인증하는 데 사용됩니다.

Harbor 레지스트리 또는 Docker Hub와 같은 컨테이너 레지스트리와 상호 작용하도록 Docker 클라이언트를 구성합니다. 이 예에서는 Harbor 감독자 서비스를 사용한다고 가정합니다.

사전 요구 사항

이 작업에서는 Docker 데몬이 설치되어 있는 Linux 호스트(Ubuntu)를 사용하고 있다고 가정합니다. Ubuntu 호스트에 Docker Engine(데몬)을 설치하려면 https://docs.docker.com/engine/install/ubuntu/ 항목을 참조하십시오.

Docker가 설치되어 있고 Docker 허브에서 이미지를 끌어올 수 있는지 확인하려면 다음 명령을 실행합니다.

```
docker run hello-world
```

예상 결과:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

참고 이 지침은 Ubuntu 20.04 및 Docker 19.03을 사용하여 확인되었습니다.

절차

1 Harbor 레지스트리에 로그인합니다.

2 **관리 > 구성 > 레지스트리 루트 인증서**를 선택합니다.

3 **다운로드**를 클릭하여 이름이 `ca.crt`인 Harbor 레지스트리 인증서를 다운로드합니다.

참고 필요한 경우 인증서의 이름을 `ca.crt`로 변경합니다.

4 `ca.crt` 파일을 Docker 호스트 클라이언트에 안전하게 복사합니다.

5 Docker 호스트에서 Harbor IP 주소를 사용하여 개인 레지스트리에 대한 디렉토리 경로를 생성합니다.

```
/etc/docker/certs.d/IP-address-or-FQDN-of-harbor/
```

예:

```
mkdir /etc/docker/certs.d/10.179.145.77
```

6 `ca.crt`를 이 디렉토리로 이동합니다.

예:

```
 mv ca.crt /etc/docker/certs.d/10.179.145.77/ca.crt
```

7 Docker 데몬을 다시 시작합니다.

```
sudo systemctl restart docker.service
```

8 Docker 클라이언트를 사용하여 내장형 Harbor 레지스트리에 로그인합니다.

```
docker login https://10.179.145.77
```

다음 메시지가 표시됩니다.

```
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

표준 패키지를 개인 Harbor 레지스트리로 푸시

표준 패키지를 공용 컨테이너 레지스트리에서 감독자 서비스로 실행되는 비공개 Harbor 레지스트리로 푸시하려면 이 항목을 참조하십시오.

사전 요구 사항

다음 사전 요구 사항을 완료하십시오.

- Harbor 감독자 서비스 설치.
- Ubuntu 호스트를 Docker 클라이언트로 구성합니다. Harbor 레지스트리 인증서를 사용하여 Docker 클라이언트 구성의 내용을 참조하십시오.
- Ubuntu 호스트에 jq를 설치합니다. https://jqlang.github.io/jq/download/의 내용을 참조하십시오.

Carvel 이미지 패키지 유틸리티 설치

kubectl이 설치된 Ubuntu 클라이언트에 Carvel `imgpkg` 유틸리티를 설치합니다.

1 `imgpkg`를 설치합니다.

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

2 설치를 확인합니다.

```
imgpkg version
```

예상 결과:

```
imgpkg version 0.41.1
```

각 표준 패키지에 대해 사용 가능한 이미지 나열

Cert Manager

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/cert-manager
```

엔보이를 사용하는 Contour

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/contour
```

ExternalDNS

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/external-dns
```

Alertmanager를 사용하는 Prometheus

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/prometheus
```

Grafana

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/grafana
```

Fluent Bit

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/fluent-bit
```

Harbor

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/harbor
```

공용 VMware 레지스트리에서 표준 패키지 이미지 끌어오기

https://projects.registry.vmware.com/의 공용 VMware 레지스트리에서 표준 Tanzu 패키지를 끌어옵니다. 끌어오려는 버전과 일치하도록 버전을 업데이트합니다.

Cert Manager

```
docker pull projects.registry.vmware.com/tkg/packages/standard/cert-manager:v1.7.2_vmware.3-tkg.3
```

엔보이를 사용하는 Contour

```
docker pull projects.registry.vmware.com/tkg/packages/standard/contour:v1.23.5_vmware.1-tkg.1
```

ExternalDNS

```
docker pull projects.registry.vmware.com/tkg/packages/standard/external-dns:v0.12.2_vmware.5-tkg.1
```

Alertmanager를 사용하는 Prometheus

```
docker pull projects.registry.vmware.com/tkg/packages/standard/prometheus:v2.37.0_vmware.3-tkg.1
```

Grafana

```
docker pull projects.registry.vmware.com/tkg/packages/standard/grafana:v7.5.17_vmware.2-tkg.1
```

Fluent Bit

```
docker pull projects.registry.vmware.com/tkg/packages/standard/fluent-bit:v1.9.5_vmware.1-tkg.2
```

Harbor

```
docker pull projects.registry.vmware.com/tkg/packages/standard/harbor:v2.7.1_vmware.1-tkg.1
```

개인 Harbor 레지스트리에서 프로젝트 생성

Tanzu 패키지 호스팅을 위해 Harbor에서 공용 프로젝트를 생성합니다.

1 개인 Harbor 레지스트리에 로그인합니다.

2 Harbor에서 **프로젝트 > 새 프로젝트**를 선택합니다.

3 이름이 **tanzu-packages**인 새 공용 프로젝트를 생성합니다.

표준 패키지 이미지에 태그 지정

다음 구문을 사용하여 이미지에 태그를 지정합니다.

```
docker tag SOURCE_IMAGE[:TAG] harbordomain.com/tanzu-packages/REPOSITORY[:TAG]
```

형식 설명:

- SOURCE_IMAGE[:TAG]은 가져온 이미지의 이름입니다.
- *harbordomain.com*은 Harbor 서버의 DNS 이름입니다.
- REPOSITORY[:TAG]는 이미지 태그의 이름입니다.

Cert Manager

```
docker tag projects.registry.vmware.com/tkg/packages/standard/cert-manager:v1.7.2_vmware.3-tkg.3 harbordomain.com/tanzu-packages/cert-manager:v1.7.2
```

엔보이를 사용하는 Contour

```
docker tag projects.registry.vmware.com/tkg/packages/standard/contour:v1.23.5_vmware.1-tkg.1 harbordomain.com/tanzu-packages/contour:v1.23.5
```

ExternalDNS

```
docker tag projects.registry.vmware.com/tkg/packages/standard/external-dns:v0.12.2_vmware.5-tkg.1 harbordomain.com/tanzu-packages/external-dns:v0.12.2
```

Alertmanager를 사용하는 Prometheus

```
docker tag projects.registry.vmware.com/tkg/packages/standard/prometheus:v2.37.0_vmware.3-tkg.1 harbordomain.com/tanzu-packages/prometheus:v2.37.0
```

Grafana

```
docker tag projects.registry.vmware.com/tkg/packages/standard/grafana:v7.5.17_vmware.2-tkg.1 harbordomain.com/tanzu-packages/grafana:v7.5.17
```

Fluent Bit

```
docker tag projects.registry.vmware.com/tkg/packages/standard/fluent-bit:v1.9.5_vmware.1-tkg.2 harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

Harbor

```
docker tag projects.registry.vmware.com/tkg/packages/standard/harbor:v2.7.1_vmware.1-tkg.1
harbordomain.com/tanzu-packages/harbor:v2.7.1
```

표준 패키지 이미지를 개인 Harbor 레지스트리로 푸시

다음 구문을 사용하여 이미지를 푸시합니다.

```
docker push harbordomain.com/tanzu-packages/PACKAGE
```

형식 설명:

- *harbordomain.com*은 Harbor 서버의 DNS 이름입니다.
- *tanzu-packages*는 Harbor 프로젝트의 이름입니다.
- *PACKAGE*는 Tanzu 패키지의 이름입니다.
- *vX.X.X*는 패키지의 태그 버전입니다.

Cert Manager

```
docker push harbordomain.com/tanzu-packages/cert-manager:v1.7.2
```

엔보이를 사용하는 Contour

```
docker push harbordomain.com/tanzu-packages/contour:v1.23.5
```

ExternalDNS

```
docker push harbordomain.com/tanzu-packages/external-dns:v0.12.2
```

Alertmanager를 사용하는 Prometheus

```
docker push harbordomain.com/tanzu-packages/prometheus:v2.37.0
```

Grafana

```
docker push harbordomain.com/tanzu-packages/grafana:v7.5.17
```

Fluent Bit

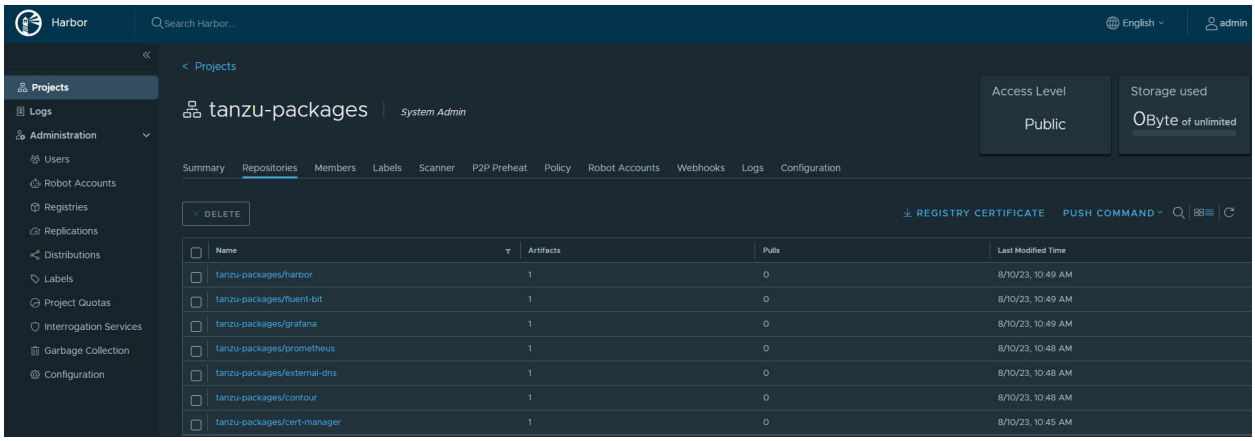
```
docker push harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

Harbor

```
docker push harbordomain.com/tanzu-packages/harbor:v2.7.1
```

이미지를 개인 Harbor 레지스트리로 푸시한 후 Harbor 웹 인터페이스에 각 이미지가 표시되는지 확인합니다.

그림 14-1. 개인 Harbor 레지스트리의 Tanzu 표준 패키지



배포용 컨테이너 이미지 끌어오기

이미지가 개인 Harbor 레지스트리에 있는지 확인하려면 다음 구문을 사용하여 거기에서 이미지를 가져옵니다.

```
docker pull harbordomain.com/tanzu-packages/PACKAGE:TAG
```

예:

```
docker pull harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

cURL을 사용하여 Harbor API를 호출하고 패키지를 나열할 수 있습니다. 이렇게 하려면 Harbor 웹 인터페이스에서 Harbor 인증서를 다운로드한 다음 다음 명령을 실행합니다.

```
curl -X 'GET' 'https://harbordomain.com/api/v2.0/projects/tanzu-packages/repositories?
page=1&page_size=-1' -H 'accept: application/json' --cacert ca.crt | jq '.[].name'
```

명령은 사용 가능한 tanzu-packages를 반환합니다.

```
"tanzu-packages/harbor"
"tanzu-packages/fluent-bit"
"tanzu-packages/grafana"
"tanzu-packages/prometheus"
"tanzu-packages/external-dns"
"tanzu-packages/contour"
"tanzu-packages/cert-manager"
```

TKG 서비스 클러스터에서 스냅샷 생성 15

TKG 서비스 클러스터는 볼륨 스냅샷 및 복원 기능을 지원합니다. DevOps 사용자는 볼륨 스냅샷을 생성하여 TKG 클러스터의 워크로드를 보호할 수 있습니다.

스냅샷을 사용하여 스냅샷 데이터로 미리 채워진 새 볼륨을 프로비저닝할 수 있습니다.

사전 요구 사항

TKG 서비스 클러스터에 대한 CSI 스냅샷을 생성하려면 환경이 다음 사전 요구 사항을 충족해야 합니다.

- vSphere 8.0 업데이트 2 이상.
- CSI 스냅샷을 지원하는 Tanzu Kubernetes 릴리스(vSphere 8.0.2 이상에 대해 TKr v1.26.5 이상). VMware Tanzu Kubernetes 릴리스 정보를 참조하십시오.
- 호환되는 최신 감독자 버전. VMware vSphere with Tanzu 8.0 릴리스 정보를 참조하십시오.

요구 사항

CSI 스냅샷 기능은 TKG 패키지로 제공됩니다. CSI 스냅샷 패키지를 사용하기 위한 요구 사항은 다음과 같습니다.

- TKG 표준 패키지 저장소 버전 v2023.9.19 이상을 사용합니다. Tanzu 표준 패키지 저장소 버전을 참조하십시오.
- Cert Manager 패키지를 설치합니다. VMware Tanzu 패키지 설치 및 사용을 참조하십시오.
- Tanzu CLI를 사용하여 vsphere-pv-csi-webhook을 설치하고 배포합니다. vSphere PVCSI Webhook 설치 및 배포의 내용을 참조하십시오.
- Tanzu CLI를 사용하여 external-csi-snapshot-webhook을 설치하고 배포합니다. 외부 CSI 스냅샷 Webhook 설치 및 배포의 내용을 참조하십시오.

지침 및 제한 사항

TKG 클러스터에서 스냅샷 및 복원 기능을 사용하는 경우 다음 지침을 따르십시오.

- 블록 볼륨만 볼륨 스냅샷 및 복원 작업을 지원합니다. 이러한 작업은 vSphere 파일 볼륨에 사용할 수 없습니다.

- VolumeSnapshot에서 PVC를 생성하는 경우 PVC가 원래 VolumeSnapshot과 동일한 데이터스토어에 상주해야 합니다. 그렇지 않으면 해당 PVC의 프로비저닝이 다음 오류와 함께 실패합니다.

  ```
  failed to provision volume with StorageClass <storage-class-name>: rpc error: code =
  Internal desc = failed to create volume. Error: failed to get the compatible datastore for
  create volume from snapshot <snapshot-name> with error: failed to find datastore with URL
  <datastore-url> from the input datastore list, <[datastore-list]>
  ```

  VolumeSnapshot에서 생성하는 대상 PVC의 데이터스토어는 PVC 정의의 StorageClass에 의해 결정됩니다. 대상 PVC의 StorageClass와 원래 소스 PVC의 StorageClass가 소스 PVC의 데이터스토어인 동일한 데이터스토어를 가리키는지 확인합니다. 이 규칙은 StorageClass 정의의 토폴로지 요구 사항에도 적용됩니다. 또한 요구 사항은 동일한 공통 데이터스토어를 가리켜야 합니다. 토폴로지 요구 사항이 충돌하면 위에 표시된 것과 동일한 오류가 발생합니다.

- 연결된 스냅샷이 포함된 볼륨은 삭제하거나 확장할 수 없습니다. 모든 스냅샷을 삭제하여 소스 볼륨을 확장하거나 삭제합니다.
- 스냅샷에서 볼륨을 생성할 때 볼륨의 크기가 스냅샷의 크기와 일치하는지 확인합니다.
- 스토리지 할당량 모니터링은 스냅샷에 대해 지원되지 않습니다.
- vSphere 구성에서는 볼륨당 최대 스냅샷 수를 구성할 수 없습니다. 성능을 더 높이려면 가상 디스크당 2~3개의 스냅샷을 사용합니다. 자세한 내용은 vSphere 환경에서 VMware 스냅샷을 사용하는 모범 사례를 참조하십시오.

  vSAN ESA의 경우 볼륨당 최대 32개의 스냅샷을 사용합니다. vSAN ESA에 대한 자세한 내용은 vSAN Express Storage Architecture를 참조하십시오.

다음으로 아래 항목을 읽으십시오.

- 외부 CSI 스냅샷 Webhook 설치 및 배포
- vSphere PVCSI Webhook 설치 및 배포
- TKG 서비스 클러스터에서 스냅샷 생성

외부 CSI 스냅샷 Webhook 설치 및 배포

TKG 서비스 클러스터에서 스냅샷 기술을 사용할 수 있으려면 외부 CSI 스냅샷 Webhook을 설치하고 TKG 클러스터에 배포해야 합니다. 외부 CSI 스냅샷 Webhook은 승인 요청에 응답하는 HTTP 콜백이 있는 오픈 소스 구성 요소입니다. 볼륨 스냅샷 개체의 유효성 검사를 담당합니다.

외부 CSI 스냅샷 Webhook은 감독자에 자동으로 설치됩니다. 이 항목은 TKG 서비스 클러스터에만 적용됩니다.

사전 요구 사항

- 감독자가 실행되고 있습니다.
- Tanzu CLI 및 kubectl을 설치했습니다. 자세한 내용은 TKG 서비스 클러스터용 CLI 도구 설치 항목을 참조하십시오.

- TKG 클러스터에 로그인했습니다. 자세한 내용은 vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결 항목을 참조하십시오.

외부 CSI 스냅샷 Webhook 설치를 위해 TKG 서비스 클러스터 준비

다음 단계에 따라 외부 CSI 스냅샷 Webhook을 TKG 서비스 클러스터에 설치합니다.

절차

1 외부 CSI 스냅샷 Webhook을 배포할 TKG 클러스터의 관리 자격 증명을 가져옵니다.

```
tanzu cluster kubeconfig get my-cluster --admin
```

2 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context my-cluster-admin@my-cluster
```

3 패키지 저장소(예: tanzu-standard 저장소)가 클러스터에 없는 경우 하나를 설치합니다.

대상 클러스터가 레거시 계획 기반 클러스터인 경우 이 단계를 건너뛸 수 있습니다. 계획 기반 클러스터의 경우 tanzu-package-repo-global 네임스페이스에서 tanzu-standard 패키지 저장소가 자동으로 사용되도록 설정됩니다.

```
tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --namespace tkg-system
```

- PACKAGE-REPO-NAME은 tanzu-standard와 같은 패키지 저장소의 이름 또는 ADDITIONAL_IMAGE_REGISTRY 변수로 구성된 개인 이미지 레지스트리의 이름입니다.
- PACKAGE-REPO-ENDPOINT는 패키지 저장소의 URL입니다.

4 아직 설치하지 않았다면 cert-manager를 설치합니다.

자세한 내용은 Cert Manager 설치 항목을 참조하십시오.

결과

이제 외부 CSI 스냅샷 Webhook을 배포할 수 있습니다.

외부 CSI 스냅샷 Webhook 배포

다음 단계에 따라 TKG 서비스 클러스터에 외부 CSI 스냅샷 Webhook을 배포합니다.

절차

1 외부 CSI 스냅샷 Webhook 패키지를 클러스터에서 사용할 수 있는지 확인합니다.

```
tanzu package available list -A
```

패키지를 사용할 수 없는 경우 필요한 외부 CSI 스냅샷 Webhook 패키지가 포함된 패키지 저장소가 올바르게 설치되어 있는지 확인합니다. 지침은 외부 CSI 스냅샷 Webhook 설치를 위해 TKG 서비스 클러스터 준비의 3단계를 참조하십시오.

2 사용 가능한 패키지의 버전을 가져옵니다.

```
tanzu package available list external-csi-snapshot-webhook.tanzu.vmware.com -A
```

3 사용 가능한 적절한 버전으로 패키지를 설치합니다.

```
tanzu package install external-csi-snapshot-webhook --package external-csi-snapshot-
webhook.tanzu.vmware.com --version AVAILABLE-PACKAGE-VERSION --namespace kube-system
```

AVAILABLE-PACKAGE-VERSION 은 2단계에서 가져온 패키지 버전을 지정합니다.

4 외부 CSI 스냅샷 Webhook 패키지가 설치되었는지 확인합니다.

```
tanzu package installed list -A
```

패키지에 대한 자세한 내용을 보려면 다음 명령을 실행할 수도 있습니다.

```
tanzu package installed get external-csi-snapshot-webhook --namespace kube-system
```

5 external-csi-snapshot-webhook 애플리케이션이 TARGET-NAMESPACE에서 조정되었는지 확인합니다.

```
kubectl get apps -A
```

상태가 `Reconcile Succeeded`가 아닌 경우 external-csi-snapshot-webhook 애플리케이션의 전체 상태 세부 정보를 확인합니다. 전체 상태를 확인하면 문제를 해결하는 데 도움이 될 수 있습니다.

```
kubectl get app external-csi-snapshot-webhook --namespace kube-system -o yaml
```

문제 해결이 문제를 해결하는 데 도움이 되지 않는 경우 다음 명령을 사용하여 패키지를 제거한 후 다시 설치합니다.

```
tanzu package installed delete external-csi-snapshot-webhook --namespace kube-system
```

6 클러스터의 모든 포드를 나열하여 external-csi-snapshot-webhook이 실행 중인지 확인합니다.

```
kubectl get pods -A
```

external-csi-snapshot-webhook 포드가 kube-system 네임스페이스에 생성되었는지 확인합니다.

vSphere PVCSI Webhook 설치 및 배포

TKG 서비스 클러스터에 vSphere PVCSI Webhook을 설치하고 배포합니다. vSphere PVCSI Webhook은 CSI 승인 요청에 응답하는 콜백이 포함된 구성 요소입니다. 영구 볼륨 할당, 영구 볼륨, 스토리지 클래스 등과 같은 Kubernetes 개체의 유효성 검사를 담당합니다.

vSphere PVCSI Webhook은 감독자에 자동으로 설치됩니다. 이 항목은 TKG 서비스 클러스터에만 적용됩니다.

사전 요구 사항

- 감독자가 실행되고 있습니다.
- Tanzu CLI 및 kubectl을 설치했습니다. 자세한 내용은 TKG 서비스 클러스터용 CLI 도구 설치 항목을 참조하십시오.
- TKG 클러스터에 로그인했습니다. 자세한 내용은 vCenter SSO 인증을 사용하여 TKG 서비스 클러스터에 연결 항목을 참조하십시오.

vSphere PVCSI Webhook 설치를 위해 TKG 서비스 클러스터 준비

vSphere PVCSI Webhook 설치를 위해 TKG 서비스 클러스터를 준비하려면 다음 단계를 따릅니다.

절차

1 vSphere PVCSI Webhook을 배포할 TKG 클러스터의 관리 자격 증명을 가져옵니다.

```
tanzu cluster kubeconfig get my-cluster --admin
```

2 대상 TKG 클러스터가 프로비저닝된 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context my-cluster-admin@my-cluster
```

3 vsphere-pv-csi-webhook 패키지가 설치된 패키지 저장소(예: tanzu-standard 저장소)가 클러스터에 없는 경우 하나를 설치합니다.

대상 클러스터가 레거시 계획 기반 클러스터인 경우 이 단계를 건너뛸 수 있습니다. 계획 기반 클러스터의 경우 tanzu-package-repo-global 네임스페이스에서 tanzu-standard 패키지 저장소가 자동으로 사용되도록 설정됩니다.

```
tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --namespace tkg-system
```

- PACKAGE-REPO-NAME은 tanzu-standard와 같은 패키지 저장소의 이름 또는 ADDITIONAL_IMAGE_REGISTRY 변수로 구성된 개인 이미지 레지스트리의 이름입니다.
- PACKAGE-REPO-ENDPOINT는 패키지 저장소의 URL입니다.

4 아직 설치하지 않았다면 cert-manager를 설치합니다.

자세한 내용은 Cert Manager 설치 항목을 참조하십시오.

결과

이제 vSphere PVCSI Webhook을 배포할 수 있습니다.

vSphere PVCSI Webhook 배포

다음 단계에 따라 TKG 서비스 클러스터에 vSphere PVCSI Webhook을 배포합니다.

절차

1 vSphere PVCSI Webhook 패키지를 클러스터에서 사용할 수 있는지 확인합니다.

```
tanzu package available list -A
```

패키지를 사용할 수 없는 경우 필요한 vSphere PVCSI Webhook 패키지가 포함된 패키지 저장소가 올바르게 설치되어 있는지 확인합니다. 지침은 vSphere PVCSI Webhook 설치를 위해 TKG 서비스 클러스터 준비의 3단계를 참조하십시오.

2 사용 가능한 패키지의 버전을 가져옵니다.

```
tanzu package available list vsphere-pv-csi-webhook.tanzu.vmware.com -A
```

3 사용 가능한 적절한 버전으로 패키지를 설치합니다.

```
tanzu package install vsphere-pv-csi-webhook --package vsphere-pv-csi-
webhook.tanzu.vmware.com --version AVAILABLE-PACKAGE-VERSION --namespace TARGET-NAMESPACE
```

- TARGET-NAMESPACE 는 vsphere-pv-csi-webhook 패키지를 설치할 네임스페이스를 지정합니다.

  참고 TARGET-NAMESPACE 는 vsphere-pv-csi 패키지가 설치된 네임스페이스와 동일해야 합니다.

  `--namespace` 플래그를 지정하지 않으면 Tanzu CLI가 패키지와 해당 리소스를 기본 네임스페이스에 설치합니다(예: vsphere-pv-csi-webhook 패키지의 경우 `vmware-system-csi`). 지정한 네임스페이스가 이미 존재해야 합니다(예: `kubectl create namespace vmware-system-csi` 실행).

- AVAILABLE-PACKAGE-VERSION 은 2단계에서 가져온 패키지 버전을 지정합니다.

4 vSphere PVCSI Webhook 패키지가 설치되었는지 확인합니다.

```
tanzu package installed list -A
```

패키지에 대한 자세한 내용을 보려면 다음 명령을 실행할 수도 있습니다.

```
tanzu package installed get vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE
```

5 vsphere-pv-csi-webhook 애플리케이션이 TARGET-NAMESPACE에서 조정되었는지 확인합니다.

```
kubectl get apps -A
```

상태가 `Reconcile Succeeded`가 아닌 경우 vsphere-pv-csi-webhook 애플리케이션의 전체 상태 세부 정보를 확인합니다. 전체 상태를 확인하면 문제를 해결하는 데 도움이 될 수 있습니다.

```
kubectl get app vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE -o yaml
```

문제 해결이 문제를 해결하는 데 도움이 되지 않는 경우 다음 명령을 사용하여 패키지를 제거한 후 다시 설치합니다.

```
tanzu package installed delete vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE
```

6 클러스터의 모든 포드를 나열하여 vsphere-pv-csi-webhook이 실행 중인지 확인합니다.

```
kubectl get pods -A
```

vsphere-pv-csi-webhook 포드가 vmware-system-csi OR TARGET-NAMESPACE에 생성되었는지 확인합니다.

TKG 서비스 클러스터에서 스냅샷 생성

동적으로 스냅샷을 프로비저닝하거나 TKG 서비스 클러스터에서 블록 볼륨의 미리 프로비저닝된 볼륨 스냅샷을 생성할 수 있습니다. 기존 스냅샷을 복원할 수도 있습니다.

참고 볼륨 스냅샷 클래스를 생성하고 이를 사용하여 볼륨 스냅샷을 생성해서는 안 됩니다. 기존 볼륨 스냅샷 클래스만 사용합니다. 기존 볼륨 스냅샷 클래스는 deletionPolicy 삭제만 지원합니다. deletionPolicy 보존을 사용하여 볼륨 스냅샷 클래스를 생성하는 것은 지원되지 않습니다.

TKG 서비스 클러스터에서 동적으로 프로비저닝된 스냅샷 생성

TKG 서비스 클러스터에서 스냅샷을 동적으로 프로비저닝합니다.

절차

1 StorageClass가 있는지 확인합니다.

```
$ kubectl get sc
NAME                              PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gc-storage-profile                csi.vsphere.vmware.com   Delete
Immediate              true                   11d
gc-storage-profile-latebinding    csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   11d
```

2 1단계의 StorageClass를 사용하여 PVC를 생성합니다.

다음 YAML을 예제로 사용합니다.

```
$ cat example-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-raw-block-pvc
spec:
  volumeMode: Block
  accessModes:
    - ReadWriteOnce
```

```
  resources:
    requests:
      storage: 1Gi
  storageClassName: gc-storage-profile
```

```
$ kubectl apply -f example-pvc.yaml
```

```
$ kubectl get pvc
NAME                     STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS         AGE
example-raw-block-pvc   Bound    pvc-4c0c030d-25ac-4520-9a04-7aa9361dfcfc   1Gi
RWO            gc-storage-profile   2m1s
```

3 VolumeSnapshotClass를 사용할 수 있는지 확인합니다.

```
$ kubectl get volumesnapshotclass
NAME                          DRIVER                    DELETIONPOLICY   AGE
volumesnapshotclass-delete   csi.vsphere.vmware.com   Delete           11d
```

4 3단계에서 얻은 VolumeSnapshotClass를 사용하여 VolumeSnapshot을 생성합니다.

```
$cat example-snapshot.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-raw-block-snapshot
spec:
  volumeSnapshotClassName: volumesnapshotclass-delete
  source:
    persistentVolumeClaimName: example-raw-block-pvc
```

```
$ kubectl apply -f example-snapshot.yaml
```

```
$ kubectl get volumesnapshot
NAME                          READYTOUSE   SOURCEPVC               SOURCESNAPSHOTCONTENT
RESTORESIZE   SNAPSHOTCLASS                SNAPSHOTCONTENT
                             CREATIONTIME   AGE
example-raw-block-snapshot   true         example-raw-block-pvc
1Gi           volumesnapshotclass-delete   snapcontent-ae019c16-
b07c-4a92-868b-029babd641d3   6s             6s
```

TKG 서비스 클러스터에서 미리 프로비저닝된 스냅샷 생성

감독자에서 동일한 블록 볼륨(PVC)의 볼륨 스냅샷을 사용하여 TKG 서비스 클러스터에서 블록 볼륨(PVC)의 미리 프로비저닝된 볼륨 스냅샷을 생성합니다.

다음 단계에 따라 감독자에 남아 있는 기본 스냅샷의 정보를 사용하여 새 TKG 클러스터에 볼륨 스냅샷을 정적으로 생성합니다.

참고 볼륨 스냅샷은 감독자에서 직접 생성할 수 없습니다.

사전 요구 사항

- Kubernetes 스냅샷 생성에 대한 정보를 숙지해야 합니다. 자세한 내용은 Kubernetes 설명서의 볼륨 스냅샷 페이지를 참조하십시오.
- 볼륨 스냅샷이 다음 조건을 충족하는지 확인합니다.
  - 볼륨 스냅샷이 소스 PVC가 상주하는 동일한 네임스페이스에 있습니다.
  - 볼륨 스냅샷이 TKG 클러스터가 상주하는 동일한 네임스페이스에 있습니다.

또한 동일한 네임스페이스의 다른 TKG 클러스터에서 더 이상 필요하지 않은 볼륨 스냅샷을 새 TKG 클러스터에서 재사용할 수도 있습니다. 이렇게 하려면 원래 TKG 클러스터에 있는 `VolumeSnapshotContent`의 `deletionPolicy`를 `Retain`으로 변경한 다음 해당하는 `VolumeSnapshot`과 `VolumeSnapshotContent` 개체를 삭제합니다.

절차

1 감독자에 있는 원래 `VolumeSnapshot` 개체의 이름을 기록해둡니다.

이전 TKG 클러스터의 볼륨 스냅샷을 재사용하는 경우 이전 TKG 클러스터에 있는 이전 `VolumeSnapshotContent` 개체의 `snapshotHandle`에서 감독자 `VolumeSnapshot` 이름을 검색할 수도 있습니다.

2 `VolumeSnapshotContent` 개체를 생성합니다.

YAML 파일에서 다음 항목의 값을 지정합니다.

`snapshotHandle`의 경우 1단계에서 얻은 감독자 `VolumeSnapshot` 이름을 입력합니다.

참고 참고: 다른 TKG 클러스터의 볼륨 스냅샷을 재사용하는 경우 새 TKG 클러스터에서 `VolumeSnapshotContent`를 생성하기 전에 `deletionPolicy`를 [보존]으로 설정하고 이전 TKG 클러스터에서 `VolumeSnapshot` 및 `VolumeSnapshotContent` 개체를 삭제합니다.

다음 YAML 매니페스트를 예제로 사용합니다.

```
--------
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
 name: static-tkg-block-snapshotcontent
spec:
 deletionPolicy: Delete
 driver: csi.vsphere.vmware.com
 source:
  snapshotHandle: "supervisor-block-volumeSnapshot-name" # Enter the VolumeSnapshot name
from the Supervisor.
```

```
 volumeSnapshotRef:
  name: static-tkg-block-snapshot
  namespace: "supervisor-tkg-namespace" # Enter the namespace of Tanzu Kubernetes Grid
Cluster.
---------
```

3 2단계에서 생성한 `VolumeSnapshotContent` 개체와 일치하도록 `VolumeSnapshot`을 생성합니다.

```
---------
apiVersion: snapshot.storage.k8s.io/vl
kind: VolumeSnapshot
metadata:
 name: static-tkg-block-snapshot
spec:
 source:
  volumeSnapshotContentName: static-tkg-block-snapshotcontent
---------
```

4 3단계에서 생성한 `VolumeSnapshot`에서 ReadyToUse가 "true"로 표시되는지 확인합니다.

```
kubecti getvolumesnapshot static-tkg-block-snapshot
NAME                       READYTOUSE SOURCEPVC SOURCESNAPSHOTCONTENT
RESTORESIZE SNAPSHOTCLASS SNAPSHOTCONTENT                  CREATIONTIME AGE
static-tkg-block-snapshot  true                 static-tkg-biock-snapshotcontent
5Gi                       static-tkg-block-snapsnotcontent 76m          22m
```

TKG 서비스 클러스터에서 볼륨 스냅샷 복원

이미 생성된 볼륨 스냅샷을 복원합니다.

절차

1 복원할 볼륨 스냅샷이 TKG 클러스터에 있는지 확인합니다.

```
$ kubectl get volumesnapshot
NAME                          READYTOUSE   SOURCEPVC               SOURCESNAPSHOTCONTENT
RESTORESIZE   SNAPSHOTCLASS
SNAPSHOTCONTENT                                        CREATIONTIME   AGE
example-raw-block-snapshot   true         example-raw-block-pvc
1Gi           volumesnapshotclass-delete   snapcontent-ae019c16-
b07c-4a92-868b-029babd641d3   2m36s          2m36s
```

2 볼륨 스냅샷에서 PVC를 생성합니다.

```
$ cat example-restore.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-raw-block-restore
spec:
  storageClassName: gc-storage-profile
  dataSource:
```

```
    name: example-raw-block-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  volumeMode: Block
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```
$ kubectl apply -f example-restore.yaml
```

```
$ kubectl get pvc
NAME                         STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS         AGE
example-raw-block-pvc        Bound    pvc-4c0c030d-25ac-4520-9a04-7aa9361dfcfc   1Gi
RWO            gc-storage-profile   11m
example-raw-block-restore    Bound    pvc-96eaab16-9ec1-446a-9392-e86d13c9b2e2   1Gi
RWO            gc-storage-profile   2m8s
```

TKG 서비스 클러스터에 대한 스토리지 관리

16

이 섹션에서는 TKG 서비스 클러스터에 대한 스토리지를 관리하기 위한 정보를 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터용 스토리지 개념
- 노드 볼륨 마운트 사용에 대한 고려 사항
- TKG 서비스 클러스터에 대한 vSphere 스토리지 정책 생성
- TKG 서비스 클러스터에서 상태 저장 애플리케이션에 대한 동적 영구 볼륨 프로비저닝
- TKG 서비스 클러스터에서 정적 영구 볼륨 프로비저닝
- TKG 서비스 클러스터를 위한 영구 볼륨 확장

TKG 서비스 클러스터용 스토리지 개념

TKG 클러스터 워크로드에는 영구 스토리지가 필요할 수 있습니다. TKG 서비스 클러스터에 대한 vSphere 스토리지 개념 및 고려 사항에 대해서는 이 항목의 정보를 참조하십시오.

TKG 서비스 클러스터에 대한 vSphere 스토리지 정책

TKG 서비스 클러스터에 영구 스토리지 리소스를 제공하기 위해 vSphere 관리자는 다양한 스토리지 요구 사항을 설명하는 vSphere 스토리지 정책을 구성합니다. 그런 다음 관리자는 TKG 클러스터가 배포될 vSphere 네임스페이스에 하나 이상의 스토리지 정책을 추가합니다. vSphere 네임스페이스에 할당된 스토리지 정책에 따라 TKG 클러스터 노드 및 워크로드가 vSphere 스토리지 환경에 배치되는 방식이 결정되며, TKG 클러스터가 영구 스토리지에 액세스하고 사용할 수 있는 데이터스토어가 결정됩니다.

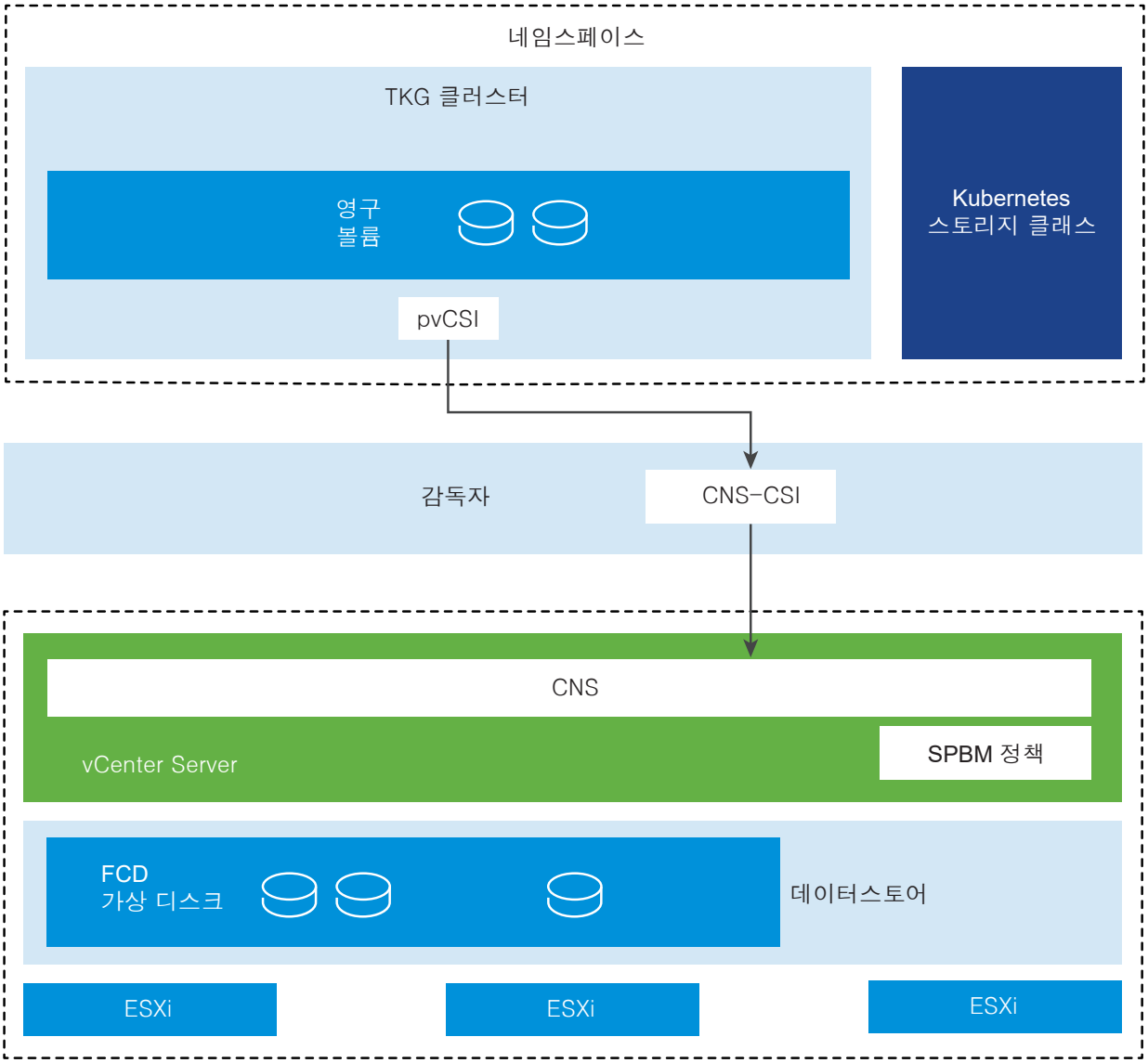
vSphere 스토리지 정책이 vSphere 네임스페이스에 할당되면 해당 vSphere 네임스페이스에 대해 일치하는 Kubernetes 스토리지 클래스가 생성됩니다. 일치하는 Kubernetes 스토리지 클래스는 해당 vSphere 네임스페이스에 프로비저닝된 TKG 클러스터에도 전파됩니다.

TKG 클러스터에서 각 스토리지 클래스는 두 가지 버전(하나는 `Immediate`, 다른 하나는 `WaitForFirstConsumer` 바인딩 모드)으로 나타납니다. 선택하는 버전은 요구 사항에 따라 다릅니다. TKG 서비스 클러스터용 스토리지 클래스 버전의 내용을 참조하십시오.

TKG 서비스 클러스터가 vSphere 스토리지와 통합되는 방식

감독자 및 vSphere 스토리지와 통합하기 위해 TKG 클러스터는 pvCSI(반가상화 CSI)를 사용합니다.

pvCSI는 TKG 클러스터에 대해 수정된 vSphere CNS-CSI 드라이버의 버전입니다. pvCSI는 TKG 클러스터에 상주하며 TKG 클러스터에서 시작되는 모든 스토리지 관련 요청을 담당합니다. 요청은 CNS-CSI로 전달된 다음 vCenter Server의 CNS로 전파됩니다. 결과적으로 pvCSI는 CNS 구성 요소와 직접적으로 통신하지 않지만 대신 CNS-CSI를 통해 모든 스토리지 프로비저닝 작업을 수행합니다. CNS-CSI와 달리 pvCSI에는 인프라 자격 증명이 필요하지 않습니다. pvCSI는 vSphere 네임스페이스에서 서비스 계정으로 구성됩니다.

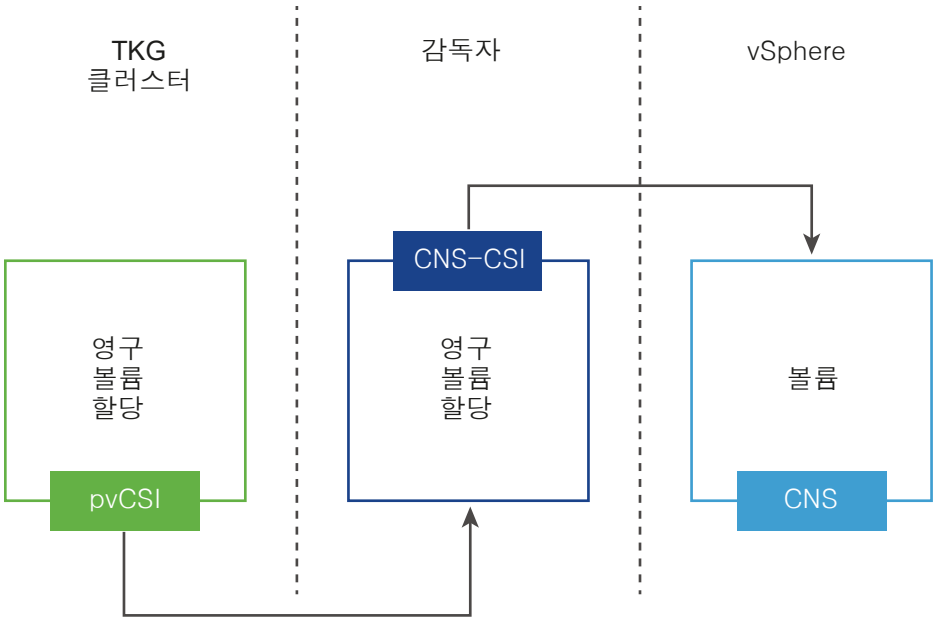


영구 볼륨이 생성되는 방식

이 다이어그램은 PVC(영구 볼륨 할당) 생성과 같이 TKG 클러스터 내에서 스토리지 관련 작업을 위해 여러 구성 요소가 상호 작용하는 방식을 보여줍니다.

DevOps 엔지니어가 TKG 클러스터에서 kubectl을 사용하여 PVC를 생성합니다. 이 작업은 감독자에서 일치하는 PVC를 생성하고 CNS 볼륨 생성 API를 호출하는 CNS-CSI를 트리거합니다.

볼륨 생성이 완료되면 작업이 감독자를 통해 TKG 클러스터로 다시 전파됩니다. 클러스터 사용자는 감독자에서 바인딩된 상태의 영구 볼륨과 영구 볼륨 할당을 볼 수 있습니다. 또한 TKG 클러스터에서 바인딩된 상태의 영구 볼륨과 영구 볼륨 할당을 볼 수 있습니다.



노드 볼륨 마운트 사용에 대한 고려 사항

하나 이상의 노드 볼륨 마운트를 사용하여 TKG 서비스 클러스터를 프로비저닝할 수 있습니다. 그 전에 중요한 고려 사항을 염두에 두십시오.

노드 볼륨 마운트 고려 사항

이 표에는 TKG 서비스 클러스터의 노드 볼륨 마운트 지점이 요약되어 있습니다. 노드 볼륨 마운트 제한 사항에 대한 자세한 내용은 다음 KB 문서를 참조하십시오. https://kb.vmware.com/s/article/92153.

클러스터가 프로비저닝된 후 볼륨 마운트 변경에 대한 자세한 내용은 Kubectl을 사용하여 수동으로 클러스터 확장/축소 항목을 참조하십시오.

볼륨 마운트	노드	지원	내림차순
`/var/lib/containerd`	작업자	전체	컨테이너 이미지 캐싱에 사용 가능한 크기를 늘립니다.
`/var/lib/kubelet`	작업자	전체	사용 후 삭제 컨테이너 스토리지에 사용 가능한 크기를 늘립니다.

볼륨 마운트	노드	지원	내림차순
`/var/lib/etcd`	제어부	없음	etcd의 고정 제한이 2GB이므로 etcd의 크기를 늘리지 않습니다. PVC 시간 초과로 인해 클러스터 생성을 방지할 수도 있습니다.

`/`(root)
`/var`
`/var/lib`
`/etc`
	제어부
	작업자
		없음	핵심 시스템 프로세스에서 사용 중인 디렉토리는 나열된 디렉토리와 다른 디렉토리(목록이 완전하지 않음)를 포함하여 지원되지 않습니다.

TKG 서비스 클러스터에 대한 vSphere 스토리지 정책 생성

vSphere 네임스페이스에 할당하는 vSphere 스토리지 정책은 Kubernetes 스토리지 클래스로 변환됩니다. 이 스토리지 클래스를 사용하여 TKG 클러스터 노드 및 영구 볼륨이 vSphere 데이터스토어 내에 배치되는 방식을 제어합니다. vSphere 영역에 대한 vSphere 스토리지 정책은 영역 토폴로지를 구성하는 모든 vSphere 클러스터의 스토리지와 호환되어야 합니다.

단일 영역 감독자에 대한 vSphere 스토리지 정책 정의

단일 영역 감독자에 대한 스토리지 클래스를 생성하는 단계를 완료하십시오.

1 vSphere Client를 사용하여 **정책 및 프로파일**을 선택합니다.

2 **VM스토리지 정책 > 생성**을 선택합니다.

3 vCenter Server를 선택합니다.

4 스토리지 정책에 알아보기 쉬운 이름을 지정합니다(예: *TKG2-cluster-storage-class*).

생성된 스토리지 클래스에 동일한 이름이 사용됩니다.

5 스토리지 **정책 구조**에서 **"VMFS"스토리지에 대한 규칙 사용**을 선택합니다.

6 VMFS 규칙의 경우 **가능한 경우 공간 절약**을 선택합니다.

7 스토리지 호환성에 대해 검토하고 **다음**을 클릭합니다.

8 **마침**을 클릭하여 스토리지 정책 생성을 완료합니다.

스토리지 정책을 vSphere 네임스페이스에 할당하려면 TKG 서비스 클러스터에 대한 vSphere 네임스페이스 구성 항목을 참조하십시오.

3개 영역 감독자에 대한 vSphere 스토리지 정책 정의

3개의 영역에 배포된 감독자를 사용하는 경우 위에 설명된 것과 다음을 추가하여 위에서 설명한 것과 동일한 단계를 완료합니다.

- 스토리지 **정책 구조**의 경우 **스토리지 토폴로지 > 사용 도메인 사용**을 선택합니다.
- **사용 도메인**의 스토리지 정책 유형으로 **영역**을 지정합니다.

그림 16-1. 영역 스토리지 클래스 1/2

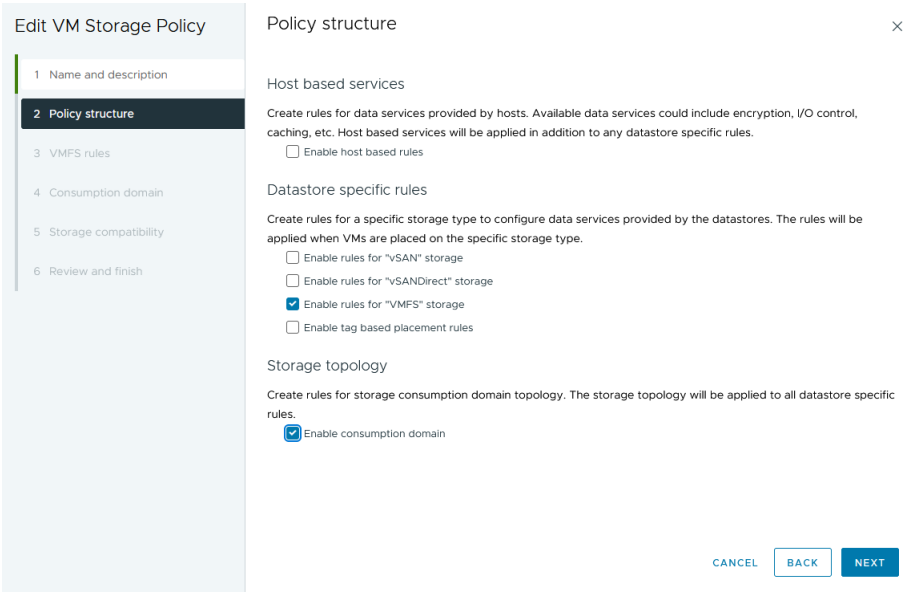
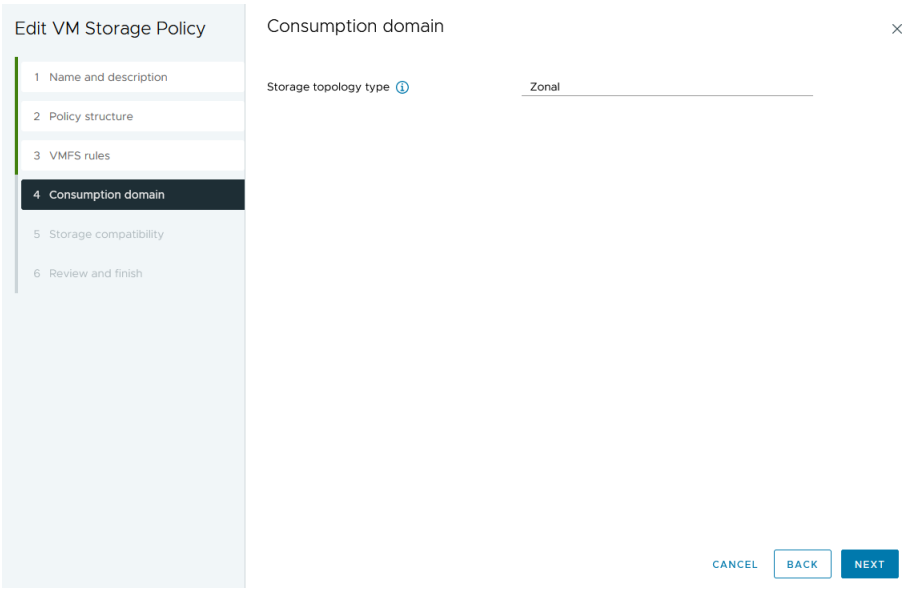


그림 16-2. 영역 스토리지 클래스 2/2



고급 vSphere 스토리지 정책 생성

환경에 따라 vSAN 또는 VVols에 대한 추가 설정이 필요할 수 있습니다. VMFS 및 NFS를 사용하는 경우 vSphere 스토리지 정책에 태그가 포함됩니다.

TKG 클러스터에서 사용하기 위해 vSphere 네임스페이스에 적용할 수 있는 태그 기반 스토리지와 같은 고급 스토리지 정책 유형을 생성하려면 "vSphere IaaS 제어부 설치 및 구성" 항목을 참조하십시오.

TKG 서비스 클러스터에서 상태 저장 애플리케이션에 대한 동적 영구 볼륨 프로비저닝

상태 저장 애플리케이션(예: 데이터베이스)은 세션 간 데이터를 저장하며, 데이터 저장을 위해 영구 스토리지가 필요합니다. 보존된 데이터를 애플리케이션의 상태라고 합니다. 나중에 데이터를 검색하여 다음 세션에서 사용할 수 있습니다. Kubernetes는 해당 상태와 데이터를 보존할 수 있는 개체로 영구 볼륨을 제공합니다.

vSphere 환경에서 영구 볼륨 개체는 데이터스토어에 상주하는 가상 디스크로 백업됩니다. 데이터스토어는 스토리지 정책으로 표시됩니다. vSphere 관리자가 스토리지 정책(예: **`gold`**)을 생성하고 이를 감독자의 네임스페이스에 할당하면 이 스토리지 정책이 vSphere 네임스페이스 및 사용 가능한 모든 TKG 클러스터에 일치하는 Kubernetes 스토리지 클래스로 나타납니다.

DevOps 엔지니어는 스토리지 클래스를 영구 볼륨 할당 규격에 사용할 수 있습니다. 그런 다음 영구 볼륨 할당에서 스토리지를 사용하는 애플리케이션을 배포할 수 있습니다. 이 예제에서는 애플리케이션의 영구 볼륨이 동적으로 생성됩니다.

사전 요구 사항

vSphere 관리자가 적절한 스토리지 정책을 생성했고 해당 정책을 네임스페이스에 할당했는지 확인합니다.

절차

1 vSphere Kubernetes 환경의 네임스페이스에 액세스합니다.

2 스토리지 클래스를 사용할 수 있는지 확인합니다.

3 영구 볼륨 할당을 생성합니다.

a 영구 볼륨 할당 구성을 포함하는 YAML 파일을 생성합니다.

이 예에서 파일은 **gold** 스토리지 클래스를 참조합니다.

ReadWriteMany 영구 볼륨을 프로비저닝하려면 `accessModes`를 `ReadWriteMany`로 설정합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 3Gi
```

b TKG 클러스터에 영구 볼륨 할당을 적용합니다.

```
kubectl apply -f pvc_name.yaml
```

이 명령은 할당의 스토리지 요구 사항을 충족하는 백업 가상 디스크가 있는 Kubernetes 영구 볼륨과 vSphere 볼륨을 동적으로 생성합니다.

c 영구 볼륨 할당의 상태를 확인합니다.

```
kubectl get pvc my-pvc
```

출력은 볼륨이 영구 볼륨 할당에 바인딩되어 있음을 보여 줍니다.

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

4 영구 볼륨을 마운트하는 포드를 생성합니다.

a 영구 볼륨을 포함하는 YAML 파일을 생성합니다.

파일에는 다음과 같은 매개 변수가 포함되어 있습니다.

```
...
volumes:
    - name: my-pvc
      persistentVolumeClaim:
        claimName: my-pvc
```

b YAML 파일에서 포드를 배포합니다.

```
kubectl create -f pv_pod_name.yaml
```

c 포드가 생성되었는지 확인합니다.

```
kubectl get pod
```

```
NAME        READY   STATUS    RESTARTS   AGE
pod_name    1/1     Ready     0          40s
```

결과

구성한 포드는 영구 볼륨 할당에 설명된 영구 스토리지를 사용합니다.

TKG 서비스 클러스터에서 정적 영구 볼륨 프로비저닝

감독자에서 사용되지 않는 PVC(영구 볼륨 할당)를 사용하여 TKG 서비스 클러스터에서 블록 볼륨을 정적으로 생성할 수 있습니다.

PVC는 다음 조건을 충족해야 합니다.

- PVC는 TKG 클러스터가 상주하는 동일한 네임스페이스에 있습니다.
- PVC가 아직 TKG 클러스터의 포드 또는 감독자의 vSphere 포드에 연결되어 있지 않습니다.

정적 프로비저닝을 사용하면 다른 TKG 클러스터에 더 이상 필요하지 않은 PVC를 새 TKG 클러스터에서 재사용할 수 있습니다. 이렇게 하려면 원래 TKG 클러스터에서 PV(영구 볼륨)의 `Reclaim policy`를 `Retain`으로 변경한 후 해당 PVC를 삭제합니다.

다음 단계에 따라 남은 기본 볼륨의 정보를 사용하여 새 TKG 클러스터에 PVC를 정적으로 생성합니다.

절차

1 감독자의 원래 PVC 이름을 기록해둡니다.

이전 TKG 클러스터의 PVC를 재사용하는 경우 TKG 클러스터에 있는 이전 PV 개체의 `volumeHandle`에서 PVC 이름을 검색할 수 있습니다.

2 PV를 생성합니다.

YAML 파일에서 다음 항목의 값을 지정합니다.

- `storageClassName`의 경우 감독자에서 PVC에 사용되는 스토리지 클래스 이름을 입력합니다.
- `volumeHandle`의 경우 얻은 PVC 이름을 입력합니다.

다른 TKG 클러스터의 볼륨을 재사용하는 경우에는 새 TKG 클러스터에 PV를 생성하기 전에 이전 TKG 클러스터에서 PVC 및 PV 개체를 삭제합니다.

다음 YAML 매니페스트를 예제로 사용합니다.

```
apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: static-tkg-block-pv
     annotations:
       pv.kubernetes.io/provisioned-by: csi.vsphere.vmware.com
   spec:
     storageClassName: gc-storage-profile
     capacity:
       storage: 2Gi
     accessModes:
       - ReadWriteOnce
     persistentVolumeReclaimPolicy: Delete
     claimRef:
       namespace: default
       name: static-tkg-block-pvc
     csi:
       driver: "csi.vsphere.vmware.com"
       volumeAttributes:
         type: "vSphere CNS Block Volume"
       volumeHandle: "supervisor-block-pvc-name"   "# Enter the PVC name from the
Supervisor."
```

3 생성한 PV 개체와 일치하도록 PVC를 생성합니다.

`storageClassName`을 PV에서와 동일한 값으로 설정합니다.

```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: static-tkg-block-pvc
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 2Gi
     storageClassName: gc-storage-profile
     volumeName: static-tkg-block-pv
```

4 생성한 PV에 PVC가 바인딩되었는지 확인합니다.

```
$ kubectl get pv,pvc
 NAME                                      CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                            STORAGECLASS         REASON   AGE
 persistentvolume/static-tkg-block-pv    2Gi        RWO            Delete
Bound    default/static-tkg-block-pvc   gc-storage-profile            10s

 NAME                                              STATUS   VOLUME                CAPACITY
ACCESS MODES   STORAGECLASS         AGE
 persistentvolumeclaim/static-tkg-block-pvc   Bound    static-tkg-block-pv   2Gi
RWO            gc-storage-profile   10s
```

TKG 서비스 클러스터를 위한 영구 볼륨 확장

Kubernetes 볼륨 확장 기능을 사용하여 영구 블록 볼륨을 생성한 후 확장할 수 있습니다. TKG 서비스 클러스터는 오프라인 및 온라인 볼륨 확장을 지원합니다.

영구 볼륨 확장 정보

TKG 클러스터 환경에 나타나는 모든 스토리지 클래스는 기본적으로 `allowVolumeExpansion`이 `true`로 설정되어 있습니다. 이 매개 변수를 사용하면 오프라인 또는 온라인 볼륨의 크기를 수정할 수 있습니다.

볼륨이 노드 또는 포드에 연결되어 있지 않으면 오프라인으로 간주됩니다. 온라인 볼륨은 노드 또는 포드에서 사용할 수 있는 볼륨입니다.

볼륨 확장 기능에 대한 지원 수준은 vSphere 버전에 따라 다릅니다. 확장을 지원하는 적절한 버전으로 vSphere 환경을 업그레이드하는 경우 이전 버전의 vSphere에서 생성된 볼륨을 확장할 수 있습니다.

참고 영구 블록 볼륨만 확장할 수 있습니다. 현재 vSphere IaaS control plane는 ReadWriteMany 볼륨에 대한 볼륨 확장을 지원하지 않습니다.

영구 블록 볼륨을 확장할 때는 다음 사항에 유의하십시오.

- 볼륨은 스토리지 할당량에 지정된 제한까지 확장할 수 있습니다. vSphere IaaS control plane는 영구 볼륨 할당 개체에 대한 연속적인 크기 조정 요청을 지원합니다.
- VMFS, vSAN, vSAN Direct, vVols 및 NFS를 포함한 모든 유형의 데이터스토어는 볼륨 확장을 지원합니다.
- 배포 또는 독립형 포드에 대한 볼륨 확장을 수행할 수 있습니다.
- Tanzu Kubernetes Grid 클러스터에서 정적으로 프로비저닝된 볼륨의 크기를 조절할 수 있습니다(볼륨에 연결된 스토리지 클래스가 있는 경우).
- StatefulSet의 일부로 생성된 볼륨은 확장할 수 없습니다.
- 볼륨을 지원하는 가상 디스크에 스냅샷이 있으면 크기를 조정할 수 없습니다.
- vSphere IaaS control plane는 인-트리(in-tree) 또는 마이그레이션된 볼륨에 대한 볼륨 확장을 지원하지 않습니다.

온라인 모드에서 영구 볼륨 확장

온라인 볼륨은 노드 또는 포드에서 사용할 수 있는 볼륨입니다. DevOps 엔지니어는 온라인 영구 블록 볼륨을 확장할 수 있습니다. Tanzu Kubernetes Grid 클러스터는 온라인 볼륨 확장을 지원합니다.

1 다음 명령을 사용하여 크기를 조정할 영구 볼륨 할당을 찾습니다.

이 예에서 볼륨이 사용하는 스토리지 크기는 1Gi입니다.

```
$ kubectl get pv,pvc,pod
NAME                                                        CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS      CLAIM               STORAGECLASS   REASON   AGE
persistentvolume/pvc-5cd51b05-245a-4610-8af4-f07e77fdc984   1Gi        RWO
Delete           Bound       default/block-pvc   block-sc                 4m56s

NAME                                STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/block-pvc   Bound    pvc-5cd51b05-245a-4610-8af4-f07e77fdc984
1Gi        RWO            block-sc       5m3s

NAME            READY   STATUS    RESTARTS   AGE
pod/block-pod   1/1     Running   0          26s
```

2 PVC에 패치를 적용하여 크기를 늘립니다. 예를 들어 크기를 2Gi로 늘립니다.

이 작업은 PVC와 연결된 볼륨에서 확장을 트리거합니다.

```
$ kubectl patch pvc block-pvc -p '{"spec": {"resources": {"requests": {"storage":
"2Gi"}}}}'
persistentvolumeclaim/block-pvc edited
```

3 PVC와 PV의 크기가 모두 증가했는지 확인합니다.

```
$ kubectl get pvc,pv,pod
NAME                                STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/block-pvc   Bound    pvc-5cd51b05-245a-4610-8af4-f07e77fdc984
2Gi        RWO            block-sc       6m18s

NAME                                                        CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS      CLAIM               STORAGECLASS   REASON   AGE
persistentvolume/pvc-5cd51b05-245a-4610-8af4-f07e77fdc984   2Gi        RWO
Delete           Bound       default/block-pvc   block-sc                 6m11s

NAME            READY   STATUS    RESTARTS   AGE
pod/block-pod   1/1     Running   0          101s
```

4 vSphere Client를 사용하여 새 영구 볼륨 크기를 확인합니다.

Tanzu Kubernetes Grid 클러스터의 볼륨 상태 모니터링의 내용을 참조하십시오.

오프라인 모드에서 영구 볼륨 확장

볼륨이 노드 또는 포드에 연결되어 있지 않으면 오프라인으로 간주됩니다. Tanzu Kubernetes Grid 클러스터는 오프라인 볼륨 확장을 지원합니다.

1 기존 스토리지 클래스에 대한 PVC(영구 볼륨 할당)를 생성합니다.

이 예에서 요청된 스토리지의 크기는 1Gi입니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-block-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: example-block-sc
```

```
kubectl apply -f example-block-pvc.yaml
```

2 PVC에 패치를 적용하여 크기를 늘립니다.

PVC가 노드에 연결되어 있지 않거나 포드에서 사용 중이면 다음 명령을 사용하여 PVC에 패치를 적용합니다. 이 예에서 요청된 스토리지 증가량은 2Gi입니다.

이 작업은 PVC와 연결된 볼륨에서 확장을 트리거합니다.

```
kubectl patch pvc example-block-pvc -p '{"spec": {"resources": {"requests": {"storage":
"2Gi"}}}}'
```

3 볼륨의 크기가 증가했는지 확인합니다.

```
kubectl get pv
NAME                                       CAPACITY ACCESS MODES RECLAIM POLICY STATUS
CLAIM                         STORAGECLASS           REASON AGE
pvc-9e9a325d-ee1c-11e9-a223-005056ad1fc1   2Gi            RWO         Delete     Bound
default/example-block-pvc   example-block-sc              6m44s
```

4 PVC의 크기 조정이 보류 중인지 확인합니다.

참고 PVC의 크기는 포드에서 PVC가 사용될 때까지 변경되지 않은 상태로 유지됩니다.

다음 예는 PVC 크기가 포드에서 사용되지 않았기 때문에 변경되지 않았음을 보여줍니다. `kubectl describe pvc`를 실행하면 PVC에 적용된 `FilesystemResizePending` 조건이 표시됩니다. 포드에서 사용되면 크기가 변경됩니다.

```
kubectl get pvc
NAME                STATUS VOLUME                                     CAPACITY ACCESS
MODES   STORAGECLASS       AGE
example-block-pvc   Bound  pvc-9e9a325d-ee1c-11e9-a223-005056ad1fc1   1Gi
RWO       example-block-sc   6m57s
```

5 PVC를 사용하려면 포드를 생성합니다.

포드에서 PVC를 사용하는 경우 파일 시스템이 확장됩니다.

6 PVC의 크기가 수정되었는지 확인합니다.

```
kubectl get pvc
NAME                STATUS VOLUME                                     CAPACITY ACCESS MODES
STORAGECLASS     AGE
example-block-pvc   Bound  pvc-24114458-9753-428e-9c90-9f568cb25788   2Gi         RWO
example-block-sc 2m12s
```

`FilesystemResizePending` 조건이 PVC에서 제거되었습니다. 볼륨 확장이 완료되었습니다.

7 vSphere Client를 사용하여 새 영구 볼륨 크기를 확인합니다.

Tanzu Kubernetes Grid 클러스터의 볼륨 상태 모니터링의 내용을 참조하십시오.

TKG 서비스 클러스터에 대한 네트워킹 관리

17

이 섹션에서는 TKG 서비스 클러스터에 대한 네트워킹을 관리하기 위한 정보를 제공합니다.

다음으로 아래 항목을 읽으십시오.

- NSX 관리 프록시 서비스 설치
- TKG 서비스 클러스터에 대해 Antrea-NSX 어댑터 사용
- Tanzu Kubernetes 클러스터에 대한 기본 CNI 설정
- TKG 클러스터에 대한 TKG 서비스 구성 사용자 지정
- TKG 클러스터에 대한 NSX 네트워킹 개체

NSX 관리 프록시 서비스 설치

NSX 관리 프록시를 Antrea-NSX 어댑터와 함께 사용하여 Antrea 기반 TKG 서비스 클러스터에서 NSX Manager에 연결할 수 있습니다. 감독자 관리 네트워크와 워크로드 네트워크 간에 분리가 있는 경우 NSX 관리 프록시가 필요합니다. 프록시는 NSX 관리 프록시 서비스가 배포된 후 생성된 모든 TKGS 클러스터에 적용됩니다.

사용 사례

NSX 관리 프록시는 TKG 서비스 클러스터에 대해 Antrea-NSX 어댑터 사용와 함께 사용하기 위한 것입니다.

감독자가 일반적인 경우처럼 관리 네트워크와 워크로드 네트워크 간에 분리하여 배포된 경우 Antrea-NSX 어댑터로 구성된 TKGS 클러스터에서 NSX 관리부에 연결하려면 NSX 관리 프록시를 사용해야 합니다. 시스템에서 이 프록시를 감지하면 감독자가 자동으로 프록시를 Antrea-NSX 어댑터 구성으로 전달합니다. 프록시가 설치되어 있지 않으면 관리 네트워크 분리가 있을 때 Antrea-NSX 어댑터가 시작되지 않습니다.

NSX 관리 프록시 서비스가 배포되면 Antrea-NSX 어댑터를 설치할 수 있습니다. TKG 서비스 클러스터에 대해 Antrea-NSX 어댑터 사용의 내용을 참조하십시오.

사전 요구 사항

NSX 관리 프록시는 감독자 서비스로 설치됩니다. 프록시 서비스를 설치하려면 다음 사전 요구 사항을 준수합니다.

- vSphere 8 U3(8.0.3) 이상

- NSX 4.1 이상
- 감독자는 NSX 네트워킹이 사용되도록 설정됩니다.
- vCenter Server에 대한 감독자 서비스 관리 권한 있음
- 감독자 서비스를 잘 알고 있음

필요한 YAML 파일 다운로드

서비스 정의 및 데이터 값을 포함하여 필요한 YAML 파일을 다운로드합니다.

1 감독자 서비스 배포 사이트(https://www.vmware.com/go/supervisor-service)로 이동합니다.

2 NSX 관리 프록시 섹션으로 스크롤하여 다음 파일을 다운로드합니다.

   a NSX 관리 프록시 서비스 정의 파일: `nsx-management-proxy.yml`

   b NSX 관리 프록시 서비스 구성 파일: `nsx-management-proxy-data-values.yml`

NSX 관리 프록시를 서비스로 등록

NSX 관리 프록시를 감독자 서비스로 등록하려면 다음 단계를 완료합니다.

1 vSphere Client를 사용하여 **워크로드 관리** > **서비스**로 이동합니다.

2 **새 서비스 추가** > **추가**를 선택합니다.

3 **업로드**를 클릭합니다.

4 다운로드한 `nsx-management-proxy.yml` 파일을 찾아 선택합니다.

5 NSX 관리 프록시 서비스 정의가 성공적으로 업로드되었는지 확인합니다.

6 **마침**을 클릭합니다.

7 NSX 관리 프록시 서비스에 대한 등록 카드가 **서비스** 탭에 나열되어 있는지 확인합니다.

NSX 관리 프록시 서비스 구성

NSX Manager 프록시 서비스를 설치하기 전에 해당 데이터 값 파일을 환경에 적합한 구성 값으로 업데이트합니다.

1 텍스트 편집기로 `nsx-management-proxy-data-values.yml` 파일을 엽니다.

2 환경과 일치하도록 아래 표에 설명된 속성을 편집합니다.

3 변경 내용을 저장합니다.

이름	값
nsxManagers	NSX Manager IP 주소 목록(필수). VIP(가상 IP 주소)가 아닌 실제 IP 주소를 사용해야 합니다. NSX 관리 클러스터를 사용하는 경우 목록에 3개의 실제 IP 주소를 모두 포함해야 합니다.
loadBalancerIP	감독자 로드 밸런서 IP 풀의 IP(선택 사항). 이 필드의 IP 주소는 워크로드 네트워크의 "수신" CIDR에서 분할됩니다. 처음에 감독자를 사용하도록 설정하는 동안 또는 나중에 vSphere 네임스페이스를 생성하고 네트워크 설정을 재정의할 때 워크로드 네트워크가 생성되면 해당 감독자 인스턴스에서 생성된 모든 vSphere 네임스페이스에서 서비스 유형 로드 밸런서 및 수신을 통해 게시된 서비스에 대한 IP 주소를 할당하는 데 사용되는 IP CIDR 블록을 수락하는 "수신"에 대한 구성 설정이 있습니다. "loadBalancerIP" 필드를 지정하지 않으면 시스템은 "수신" CIDR 범위에서 사용 가능한 IP 주소를 하나 자동으로 할당합니다. "loadBalancerIP"가 지정된 경우 "수신" CIDR 범위 내에 있어야 하며 이미 할당된 IP 주소와 충돌하면 안 됩니다. vSphere 네임스페이스에서 "kubectl get services -o wide -A" 명령을 사용하여 할당된 "수신" IP를 볼 수 있습니다. IP는 kubectl 출력의 "EXTERNAL-IP" 열에 있습니다.

NSX 관리 프록시 서비스 설치

다음 단계를 완료하여 NSX 관리 프록시 서비스를 설치합니다.

1 **워크로드 관리 > > 서비스** 화면으로 이동합니다.

2 **NSX 관리 프록시** 서비스 카드에서 **작업 > 감독자에 설치**를 선택합니다.

3 **사용 가능** 탭을 선택합니다.

4 편집한 `nsx-management-proxy-data-values.yml`의 컨텐츠를 복사한 후 "YAML 서비스 구성" 입력 필드에 붙여 넣습니다.

5 **확인**을 클릭하여 Harbor 설치를 계속 진행합니다.

6 설치를 모니터링하고 확인합니다.

NSX 관리 프록시 서비스 카드의 감독자 필드를 확인하여 설치를 모니터링합니다. 감독자 옆의 숫자가 증가되는 것을 확인할 수 있습니다. 서비스는 원하는 상태에 도달할 때까지 [구성 중] 상태입니다. 원하는 상태에 도달하면 서비스의 상태가 [구성됨]으로 변경됩니다.

7 NSX 관리 프록시에 대한 vSphere 네임스페이스가 있는지 확인합니다.

NSX 관리 프록시가 설치되면 서비스 인스턴스에 대해 vSphere 네임스페이스가 생성됩니다.

8 프록시 로드 밸런서 IP 주소를 가져옵니다.

NSX 관리 프록시 vSphere 네임스페이스의 **네트워크** 탭에서 프록시 로드 밸런서 IP 주소를 볼 수 있습니다.

NSX 관리 프록시 서비스 문제 해결

다음 오류가 표시되면 환경이 호환되지 않음을 의미합니다. 환경이 사전 요구 사항 섹션에 나열된 버전을 준수하는지 확인합니다.

```
Creation of Supervisor Service with ID nsx-management-proxy.nsx.vmware.com is not allowed.
Only service IDs defined in the allow-list file /etc/vmware/wcp/supervisor-services-allow-
list.txt are allowed.
```

TKG 서비스 클러스터에 대해 Antrea-NSX 어댑터 사용

네트워킹 관리 및 모니터링을 위해 Antrea CNI를 사용하는 TKG 서비스 클러스터를 NSX Manager와 통합할 수 있는 Antrea-NSX 어댑터를 사용하도록 설정하려면 이 항목을 참조하십시오.

Antrea-NSX 어댑터에 대한 사전 요구 사항

다음 사전 요구 사항을 준수합니다.

- vSphere 8 U3(8.0.3) 이상
- NSX 4.1 이상
- 감독자는 NSX 네트워킹이 사용되도록 설정됩니다.
- TKG 서비스 3.0 이상
- vSphere 8.x 이상용 Tanzu Kubernetes 릴리스 v1.28.x
- NSX 관리 프록시 서비스 설치일반적인 감독자 토폴로지인 관리 네트워크와 워크로드 네트워크 간에 분리가 있는 경우

Antrea-NSX 어댑터에 대한 요구 사항

Antrea-NSX 어댑터를 사용하면 Antrea 기반 TKG 서비스 클러스터를 NSX Manager와 통합할 수 있습니다. 어댑터가 구성되면 NSX Manager를 사용하여 클러스터의 네트워킹 동작을 관리할 수 있습니다. Antrea-NSX 통합의 기능에 대한 자세한 내용은 NSX 4.1 관리 가이드에서 Antrea CNI와 Kubernetes 클러스터 통합을 참조하십시오.

NSX와 통합하려는 각 TKG 서비스 클러스터에 대해 Antrea-NSX 어댑터를 사용하도록 설정해야 합니다. 즉, 어댑터와 클러스터 간의 비율은 1:1입니다. 또한 새 클러스터 배포에 Antrea-NSX 어댑터만 사용할 수 있습니다. TKG 서비스 클러스터를 생성하기 전에 어댑터를 사용하도록 설정해야 하며, 생성할 클러스터의 이름을 어댑터 리소스 정의에 포함해야 합니다.

TKGS 클러스터에 대한 NSX Tier-0 또는 Tier 1 게이트웨이가 SNAT IP로 구성된 경우 모든 Antrea-NSX 연결이 단일 소스 IP를 공유합니다. NSX는 동일한 IP에서 이를 많은 제어부 연결로 해석하고 연결을 삭제합니다. 이 경우 NSX UA 방화벽 규칙을 수동으로 변경해야 합니다. 자세한 내용은 다음 KB 문서를 참조하십시오. https://knowledge.broadcom.com/external/article?articleNumber=317179

Antrea-NSX 어댑터 사용

Antrea-NSX 어댑터를 사용하도록 설정하려면 다음 단계를 완료합니다.

1 kubectl을 사용하여 감독자에서 인증합니다.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-
username USERNAME
```

2 TKG 서비스 클러스터를 프로비저닝할 대상 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context vsphere-namespace
```

3 `AntreaConfig.yaml` 사용자 지정 리소스 정의를 생성합니다.

```
#AntreaConfig.yaml
apiVersion: cni.tanzu.vmware.com/v1alpha1
kind: AntreaConfig
metadata:
 name: tkgs-cluster-name-antrea-package #prefix required
 namespace: tkgs-cluster-ns
spec:
 antreaNSX:
  enable: true #false by default
```

형식 설명:

- 요소 `antreaNSX.enable.true`는 Antrea-NSX 어댑터를 사용하도록 설정합니다. 기본적으로 이 필드는 false입니다.
- 요소 `metadata.name`에는 필수 접미사 `-antrea-package`가 포함되며 이 접미사 앞에는 나중에 생성하여 이 어댑터를 사용할 TKG 서비스 클러스터의 정확한 이름이 붙습니다.

4 AntreaConfig 사용자 지정 리소스 정의를 적용합니다.

```
kubectl apply -f AntreaConfig.yaml
```

5 TKG 서비스 클러스터를 프로비저닝합니다.

어댑터는 v1alpha3 API 또는 v1beta1 API와 함께 사용할 수 있습니다.

Kubectl을 사용하여 TKG 클러스터를 프로비저닝하기 위한 워크플로의 내용을 참조하십시오.

6 NSX Manager에 로그인하여 어댑터가 작동하는지 확인합니다.

자세한 지침은 NSX 4.1 설명서를 참조하십시오.

Tanzu Kubernetes 클러스터에 대한 기본 CNI 설정

Tanzu Kubernetes 클러스터의 기본 CNI(Container Network Interface)는 Antrea입니다. vSphere Client를 사용하여 기본 CNI를 변경할 수 있습니다.

기본 CNI

vSphere IaaS control plane는 TKG 클러스터에 대해 Antrea 및 Calico라는 두 가지 CNI 옵션을 지원합니다. 시스템 정의 기본 CNI는 Antrea입니다.

vSphere Client를 사용하여 기본 CNI를 변경할 수 있습니다. 기본 CNI를 설정하려면 다음 절차를 완료합니다.

경고 기본 CNI를 변경하는 것은 글로벌 작업입니다. 새로 설정된 기본값은 서비스에서 생성된 모든 새 TKG 클러스터에 적용됩니다. 기존 클러스터는 변경되지 않습니다.

1 vSphere Client를 사용하여 vSphere IaaS control plane 환경에 로그인합니다.

2 **워크로드 관리**를 선택하고 **감독자**를 선택합니다.

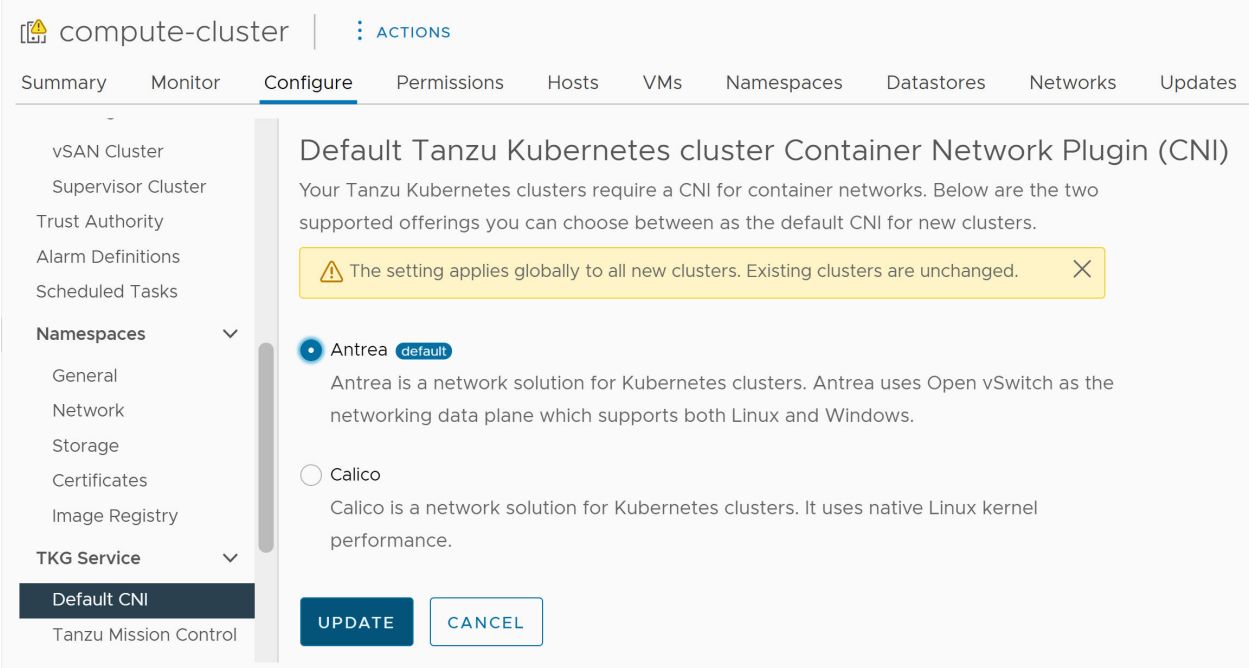
3 목록에서 감독자 인스턴스를 선택합니다.

4 **구성**을 선택하고 **TKG 서비스 > 기본 CNI**를 선택합니다.

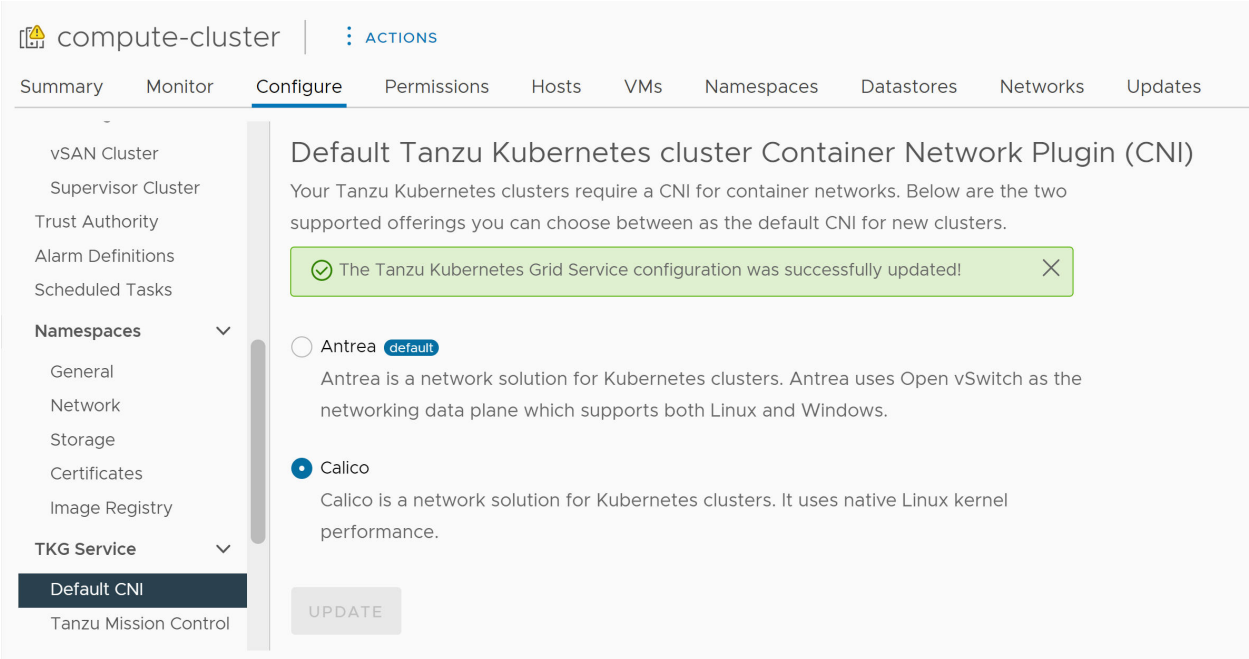
5 새 클러스터에 대한 기본 CNI를 선택합니다.

6 **업데이트**를 클릭합니다.

다음 이미지는 기본 CNI 선택 항목을 보여줍니다.



다음 이미지는 CNI 선택을 Antrea에서 Calico로 변경하는 것을 보여줍니다.



TKG 클러스터에 대한 TKG 서비스 구성 사용자 지정

CNI(Container Network Interface), 프록시 서버 및 TLS 인증서를 포함한 v1alpha3 API를 사용하여 프로비저닝된 TKG 클러스터에 대해 TKG 서비스 구성을 사용자 지정할 수 있습니다.

TkgServiceConfiguration 사용자 지정 정보

TkgServiceConfiguration을 편집하여 Tanzu Kubernetes 클러스터에 대한 글로벌 설정을 구성할 수 있습니다. 이 구성을 통해 기본 CNI를 설정하고, 글로벌 프록시 서버를 추가하고, 하나 이상의 신뢰할 수 있는 TLS 인증서를 추가할 수 있습니다.

경고 TkgServiceConfiguration 사용자 지정은 글로벌 작업입니다. `TkgServiceConfiguration` 개체를 변경하면 해당 서비스에서 프로비저닝된 모든 TKG 클러스터에 적용됩니다. 롤링 업데이트가 시작되면(수동으로 또는 업그레이드를 통해) 변경된 서비스 규격에 따라 클러스터가 업데이트됩니다.

참고 TkgServiceConfiguration 사용자 지정은 v1alpha3 API로 프로비저닝된 Tanzu Kubernetes 클러스터에 적용됩니다. v1beta1 API로 프로비저닝된 클러스터에는 적용되지 않습니다.

TkgServiceConfiguration 규격

`TkgServiceConfiguration` 규격은 Tanzu Kubernetes Grid 인스턴스 구성을 위한 필드를 제공합니다.

중요 유효한 키 이름은 영숫자, 대시(예: `key-name`), 밑줄(예: `KEY_NAME`) 또는 점(예: `key.name`)으로만 구성되어야 합니다. 키 이름에는 공백을 사용할 수 없습니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TkgServiceConfiguration
metadata:
  name: tkg-service-configuration-spec
spec:
  defaultCNI: string
  proxy:
    httpProxy: string
    httpsProxy: string
    noProxy: [string]
  trust:
    additionalTrustedCAs:
      - name: string
        data: string
  defaultNodeDrainTimeout: time
```

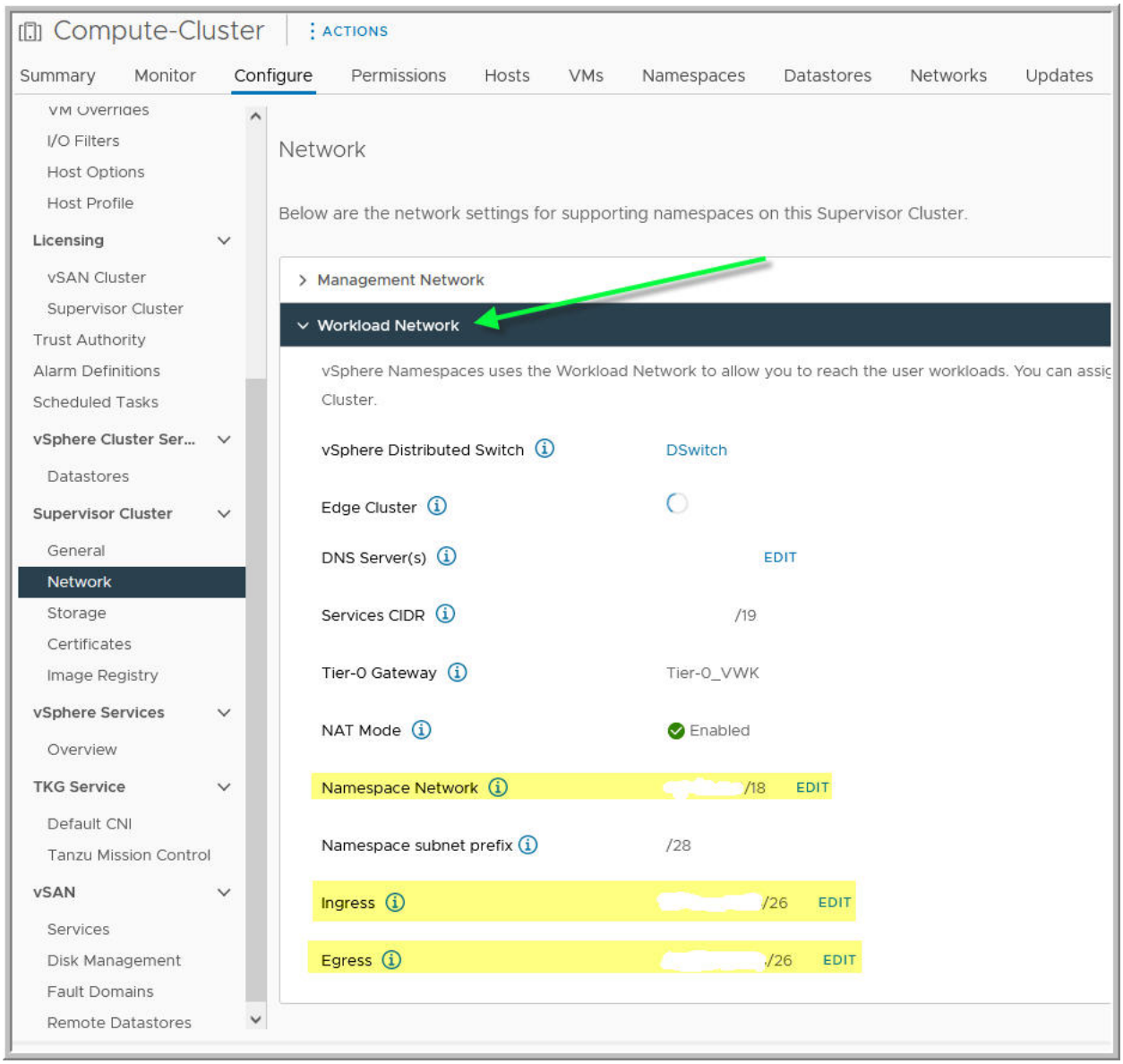
주석이 달린 TkgServiceConfiguration 규격

다음 YAML은 각 `TkgServiceConfiguration` 규격 매개 변수에 대해 구성 가능한 필드를 나열하고 설명합니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TkgServiceConfiguration
#valid config key must consist of alphanumeric characters, '-', '_' or '.'
metadata:
  name: tkg-service-configuration-spec
spec:
  #defaultCNI is the default CNI for all Tanzu Kubernetes
  #clusters to use unless overridden on a per-cluster basis
  #supported values are antrea, calico, antrea-nsx-routed
  #defaults to antrea
  defaultCNI: string
  #proxy configures a proxy server to be used inside all
  #clusters provisioned by this TKGS instance
  #if implemented all fields are required
  #if omitted no proxy is configured
  proxy:
    #httpProxy is the proxy URI for HTTP connections
    #to endpionts outside the clusters
    #takes the form http://<user>:<pwd>@<ip>:<port>
    httpProxy: string
    #httpsProxy is the proxy URI for HTTPS connections
    #to endpoints outside the clusters
    #takes the frorm http://<user>:<pwd>@<ip>:<port>
    httpsProxy: string
    #noProxy is the list of destination domain names, domains,
    #IP addresses, and other network CIDRs to exclude from proxying
```

```
    #must include from Workload Network: [Namespace Netowrk, Ingress, Egress]
    noProxy: [string]
  #trust configures additional trusted certificates
  #for the clusters provisioned by this TKGS instance
  #if omitted no additional certificate is configured
  trust:
    #additionalTrustedCAs are additional trusted certificates
    #can be additional CAs or end certificates
    additionalTrustedCAs:
      #name is the name of the additional trusted certificate
      #must match the name used in the filename
      - name: string
        #data holds the contents of the additional trusted cert
        #PEM Public Certificate data encoded as a base64 string
        data: string
  #defaultNodeDrainTimeout is the total amount of time the
  #controller spends draining a node; default is undefined
  #which is the value of 0, meaning the node is drained
  #without any time limitations; note that `nodeDrainTimeout`
  #is different from `kubectl drain --timeout`
  defaultNodeDrainTimeout: time
```

그림 17-1. 필수 noProxy 값



참고 `TkgServiceConfiguration`에 글로벌 프록시가 구성된 경우 이 프록시 정보는 클러스터의 초기 배포 후 클러스터 매니페스트로 전파됩니다. 글로벌 프록시 구성은 클러스터를 생성할 때 프록시 구성 필드가 없는 경우에만 클러스터 매니페스트에 추가됩니다. 다시 말해, 클러스터별 구성이 우선하며 글로벌 프록시 구성을 덮어씁니다.

TkgServiceConfiguration 규격 예시

다음 YAML은 각 `TkgServiceConfiguration` 규격 매개 변수에 대해 구성 가능한 필드를 나열하고 설명합니다.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TkgServiceConfiguration
metadata:
  name: tkgserviceconfiguration_example
spec:
```

```
defaultCNI: calico
proxy:
  #supported format is `http://<user>:<pwd>@<ip>:<port>`
  httpProxy: http://admin:PaSsWoRd@10.66.100.22:80
  httpsProxy: http://admin:PaSsWoRd@10.66.100.22:80
  #noProxy vaulues are from Workload Network: [Namespace Network, Ingress, Egress]
  noProxy: [10.246.0.0/16,192.168.144.0/20,192.168.128.0/20]
trust:
  additionalTrustedCAs:
    #name is the name of the public cert
    - name: CompanyInternalCA-1
    #data is base64-encoded string of a PEM encoded public cert
      data: LS0tLS1C...LS0tCg==
      #where "..." is the middle section of the long base64 string
    - name: CompanyInternalCA-2
      data: MTLtMT1C...MT0tPg==
defaultNodeDrainTimeout: 0
```

TkgServiceConfiguration 편집

TkgServiceConfiguration 규격을 편집하려면 다음 절차를 참조하십시오.

1 Kubectl 편집을 구성합니다. Kubectl용 텍스트 편집기 구성의 내용을 참조하십시오.

2 감독자로 인증합니다.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

3 컨텍스트를 대상 vSphere 네임스페이스로 전환합니다.

```
kubectl config use-context vSphere-Namespace
```

4 TkgServiceConfiguration 규격을 가져옵니다.

```
kubectl get tkgserviceconfigurations
```

5 TkgServiceConfiguration 규격을 편집하기 위해 로드합니다.

```
kubectl edit tkgserviceconfigurations tkg-service-configuration
```

KUBE_EDITOR 또는 EDITOR 환경 변수로 정의된 기본 텍스트 편집기에서 `tkg-service-configuration` 규격이 열립니다.

6 요구 사항에 따라 TkgServiceConfiguration을 편집합니다.

7 변경 내용을 적용하려면 텍스트 편집기에서 파일을 저장합니다. 취소하려면 저장하지 않고 편집기를 닫습니다.

텍스트 편집기에서 변경 내용을 저장하면 kubectl은 `tkg-service-configuration` 서비스 규격을 업데이트합니다.

8 TkgServiceConfiguration 규격이 업데이트되었는지 확인합니다.

```
kubectl get tkgserviceconfigurations
```

글로벌 구성 변경을 기존 클러스터에 전파

`TkgServiceConfiguration`의 글로벌 수준에서 지정한 설정은 기존 클러스터에 자동으로 전파되지 않을 수 있습니다. 예를 들어 `TkgServiceConfiguration`에서 `proxy` 또는 `trust` 설정을 변경하는 경우 이미 프로비저닝된 클러스터에는 이러한 변경 사항이 영향을 주지 않을 수 있습니다.

기존 클러스터에 글로벌 변경 내용을 수동으로 전파하려면 Tanzu Kubernetes 클러스터에 패치를 적용하여 `TkgServiceConfiguration`에 대한 변경 내용을 클러스터가 상속하도록 해야 합니다.

예:

```
kubectl patch tkc <CLUSTER_NAME> -n <NAMESPACE> --type merge -p "{\"spec\":{\"settings\":
{\"network\":{\"proxy\": null}}}}"
```

```
kubectl patch tkc <CLUSTER_NAME> -n <NAMESPACE> --type merge -p "{\"spec\":{\"settings\":
{\"network\":{\"trust\": null}}}}"
```

TKG 클러스터에 대한 NSX 네트워킹 개체

이 항목에서는 NSX 네트워킹을 사용하는 감독자를 사용할 때 TKG 클러스터에 대해 생성된 네트워킹 개체를 나열합니다.

TKG 클러스터에 대한 NSX 네트워킹 개체

각 TKG 클러스터에는 가상 네트워크, 가상 네트워크 인터페이스 및 가상 시스템 서비스와 같은 네트워크 리소스가 있어야 합니다.

vSphere IaaS control plane가 사용되도록 설정되고 감독자 인스턴스가 배포되면 NSX 내장된 로드 밸런서가 자동으로 프로비저닝됩니다. 이 로드 밸런서는 감독자 제어부를 위한 것으로 Kubernetes API 서버에 대한 액세스를 제공합니다.

TKG 클러스터에 대해 LoadBalancer 유형의 Kubernetes 서비스를 생성하면 NSX 내장된 로드 밸런서가 해당 서비스에 대해 프로비저닝됩니다.

네트워크 개체	네트워크 리소스	설명
VirtualNetwork	Tier-1 라우터 및 연결된 세그먼트	클러스터에 대한 노드 네트워크
VirtualNetworkInterface	세그먼트의 논리적 포트	클러스터 노드에 대한 노드 네트워크 인터페이스
VirtualMachineService	해당 없음	VirtualMachineService가 생성되고 k8s 서비스로 변환됩니다.

네트워크 개체	네트워크 리소스	설명
서비스	VirtualServer 인스턴스 및 연결된 서버 풀(멤버 풀)이 있는 로드 밸런서 서버	TKG 클러스터 API 서버에 액세스하기 위해 Load Balancer 유형의 Kubernetes 서비스가 생성됩니다.
끝점	끝점 멤버(TKG 클러스터 제어부 노드)는 멤버 풀에 있어야 합니다.	모든 TKG 클러스터 제어부 노드를 포함하도록 끝점이 생성됩니다.
감독자의 VirtualMachineService	해당 없음	감독자에서 VirtualMachineService가 생성되고 감독자에서 Kubernetes 서비스로 변환됨
감독자의 로드 밸런서 서비스	TKG 클러스터 로드 밸런서의 VirtualServer 및 연결된 멤버 풀.	이 LB 유형의 서비스에 액세스하기 위해 감독자에서 로드 밸런서 서비스가 생성됨
감독자의 끝점	끝점 멤버(TKG 클러스터 작업자 노드)는 NSX의 멤버 풀에 있어야 합니다.	모든 TKG 클러스터 작업자 노드를 포함하도록 끝점이 생성됩니다.
TKG 클러스터의 로드 밸런서 서비스	해당 없음	사용자가 배포한 TKG 클러스터의 로드 밸런서 서비스의 상태가 로드 밸런서 IP로 업데이트되어야 합니다.

노드 네트워킹

각 TKG 클러스터에는 다음과 같은 네트워크 개체와 연결된 NSX 리소스가 생성되어야 합니다.

네트워크 개체	NSX 리소스	설명	IPAM
VirtualNetwork	Tier-1 게이트웨이 및 연결된 세그먼트	TKG 클러스터에 대한 노드 네트워크	SNAT IP가 할당됨
VirtualNetworkInterface	연결된 세그먼트의 논리적 포트	TKG 클러스터 노드에 대한 노드 네트워크 인터페이스	각 노드에 IP가 할당됨

제어부 로드 밸런서

네트워크 개체	네트워크 리소스	설명	IPAM
VirtualMachineService	해당 없음	VirtualMachineService가 생성되고 Kubernetes 서비스로 변환됩니다.	로드 밸런서 VIP 포함
서비스	VirtualServer 인스턴스 및 연결된 서버 풀(멤버 풀)이 있는 로드 밸런서 서버	TKG 클러스터 API 서버에 액세스하기 위해 Load Balancer 유형의 Kubernetes 서비스가 생성됩니다.	외부 IP가 할당되었습니다.
끝점	끝점 멤버는 TKG 클러스터 제어부 노드이며 멤버 풀에 있어야 합니다.	모든 TKG 클러스터 제어부 노드를 포함하도록 끝점이 생성됩니다.	해당 없음

NSX 로드 밸런서

생성된 각 TKG 클러스터에 대해 시스템은 소규모 NSX 로드 밸런서의 단일 인스턴스를 생성합니다. 이 로드 밸런서에는 다음 표에 나열된 개체가 포함되어 있습니다.

개체 번호	설명
1	포트 8443에서 Kubernetes 제어부 API에 액세스하기 위한 VS(가상 서버)입니다.
1	Kubernetes 제어부 노드 3개가 포함된 서버 풀입니다.
1	HTTP 수신 컨트롤러용 VS입니다.
1	HTTPS 수신 컨트롤러용 VS입니다.

NAT 규칙

생성된 각 TKG 클러스터에 대해 시스템은 Tier-0 논리적 라우터에서 다음과 같은 NSX NAT 규칙을 정의합니다.

개체 번호	설명
1	부동 IP 풀의 IP 1개를 변환된 IP 주소로 사용하여 각 Kubernetes 네임스페이스에 대해 생성된 SNAT 규칙입니다.
1	(NAT 토폴로지만 해당) 부동 IP 풀의 IP 1개를 변환된 IP 주소로 사용하여 각 Kubernetes 클러스터에 대해 생성된 SNAT 규칙입니다. Kubernetes 클러스터 서브넷은 /24 넷마스크를 사용하여 노드 IP 블록에서 파생됩니다.

DFW 규칙

생성된 각 TKG 클러스터에 대해 시스템은 다음과 같은 NSX 분산 방화벽 규칙을 정의합니다.

개체 번호	설명
1	CoreDNS 포드 논리적 포트에 적용되는 `kube-dns`에 대한 DFW 규칙:
1	네임스페이스의 유효성 검사기에 대한 DFW 규칙(유효성 검사기 Pod 논리 포트에 적용됨):

TKG 서비스 클러스터에 대한 보안 관리 18

이 섹션에서는 TKG 서비스 클러스터에 대한 보안을 관리하기 위한 정보를 제공합니다.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터에 대한 보안
- TKR 1.25 이상에 대한 PSA 구성
- TKR 1.24 및 이전 버전에 대한 PSP 구성
- TKG 서비스 클러스터에 기본 포드 보안 정책 적용
- TKG 서비스 클러스터에 대한 TLS 인증서 관리
- NSX 인증서 순환

TKG 서비스 클러스터에 대한 보안

감독자의 TKG 서비스는 vSphere 보안 기능을 활용하며 기본적으로 안전한 워크로드 클러스터를 프로비저닝합니다.

vSphere IaaS control plane는 vCenter Server 및 ESXi에 내장된 보안 기능을 활용할 수 있는 vSphere에 대한 추가 기능 모듈입니다. 자세한 내용은 vSphere 보안 설명서를 참조하십시오.

감독자는 데이터베이스(etcd)에 저장된 모든 암호를 암호화합니다. 암호는 부팅 시 vCenter Server에 의해 제공되는 로컬 암호와 키 파일을 통해 암호화됩니다. 암호 해독 키는 감독자 제어부 노드의 메모리(tempfs) 및 vCenter Server 데이터베이스 내의 디스크에 암호화된 형식으로 저장됩니다. 키는 각 시스템의 루트 사용자에게 일반 텍스트로 사용할 수 있습니다.

각 TKG 클러스터 제어부에 설치된 데이터베이스(etcd)의 데이터에 동일한 암호화 모델이 적용됩니다. 모든 etcd 연결은 설치 시 생성되고 업그레이드 중에 순환되는 인증서를 사용하여 인증됩니다. 현재는 인증서를 수동으로 순환하거나 업데이트할 수 없습니다. 각 워크로드 클러스터의 데이터베이스 내에 유지되는 암호는 일반 텍스트로 저장됩니다.

TKG 클러스터에 인프라 자격 증명이 없습니다. TKG 클러스터 내에 저장된 자격 증명은 TKG 클러스터에 테넌시가 있는 vSphere 네임스페이스에 액세스하는 데에만 충분합니다. 따라서 클러스터 운영자 또는 사용자에 대해 권한을 에스컬레이션할 수 있는 방법은 없습니다.

TKG 클러스터에 액세스하는 데 사용되는 인증 토큰은 감독자 또는 기타 TKG 클러스터에 액세스하는 데 토큰을 사용할 수 없도록 범위가 지정됩니다. 이렇게 하면 클러스터 운영자 또는 클러스터를 손상시키려는 개인이 TKG 클러스터에 로그인할 때 vSphere 관리자의 토큰을 캡처하기 위해 루트 수준 액세스를 사용하는 것을 방지할 수 있습니다.

TKG 클러스터는 기본적으로 안전합니다. Tanzu Kubernetes 릴리스 v1.25부터 TKG 클러스터에는 기본적으로 PSA(포드 보안 승인) 컨트롤러가 사용되도록 설정됩니다. Tanzu Kubernetes 릴리스 v1.24까지 모든 TKG 클러스터에 대해 제한적인 PSP(포드 보안 정책)를 사용할 수 있습니다. 개발자가 권한 있는 포드 또는 루트 컨테이너를 실행해야 하는 경우, 최소한 클러스터 관리자는 권한이 있는 기본 PSP에 대한 액세스 권한을 사용자에게 부여하는 RoleBinding을 생성해야 합니다.

TKR 1.25 이상에 대한 PSA 구성

Tanzu Kubernetes 릴리스 v1.25 이상은 PSA(포드 보안 승인) 컨트롤러를 사용하도록 설정합니다. PSA를 사용하면 네임스페이스 레이블을 사용하여 포드 보안을 균일하게 적용할 수 있습니다.

TKR 1.25 이상에서 PSA 사용

포드 보안 승인 컨트롤러는 TKG 클러스터에서 실행되는 포드에 보안 표준을 적용할 수 있는 Kubernetes 컨트롤러입니다. 기본적으로 Tanzu Kubernetes 릴리스 v1.25 이상은 PSA(포드 보안 승인) 컨트롤러를 사용하도록 설정합니다. PSA 컨트롤러는 더 이상 사용되지 않으며 제거된 PSP(포드 보안 정책) 컨트롤러를 대체합니다. TKR 1.24 및 이전 버전에 대한 PSP 구성 항목도 참조하십시오.

Tanzu Kubernetes 릴리스 v1.25는 PSA가 주의하도록 구성된 전환 릴리스입니다. Tanzu Kubernetes 릴리스 v1.26부터 PSA가 적용됩니다. TKG 클러스터 업그레이드에 대비하여 포드 워크로드를 PSP에서 PSA로 마이그레이션할 계획을 세워야 합니다. 지침은 포드 보안 정책에서 기본 제공 포드 보안 승인 컨트롤러로 마이그레이션을 참조하십시오.

PSA 클러스터 전체 구성

vSphere 8 업데이트 3부터는 v1beta1 API에서 사용할 수 있는 `podSecurityStandard` ClusterClass 변수를 사용하여 PSA 클러스터 전체를 구성할 수 있습니다. Cluster v1beta1 API 의 내용을 참조하십시오.

PSA 모드

PSA 컨트롤러는 `enforce`, `audit` 및 `warn`의 세 가지 포드 보안 모드를 지원합니다. 이 표에는 각 PSA 모드와 그에 대한 설명이 나와 있습니다.

표 18-1. PSA 모드

모드	설명
`enforce`	보안 위반 시 포드가 거부됩니다.
`audit`	보안 위반 시 감사 로그에 기록된 이벤트에 감사 주석이 추가되지만 그 외에는 허용됩니다.
`warn`	보안 위반 시 사용자에게 주의가 표시되지만 그 외에는 허용됩니다.

PSA 표준

PSA 컨트롤러는 보안 스펙트럼을 포괄하는 세 가지 수준의 포드 보안 표준을 정의합니다. 포드 보안 표준은 누적되며 허용에서 제한에 이르기까지 다양합니다. 이 표에는 각 PSA 표준과 그에 대한 설명이 나와 있습니다.

표 18-2. PSA 표준

수준	설명
`privileged`	무제한 제어로, 가장 넓은 범위의 사용 권한 수준을 제공합니다. 이 보안 표준은 알려진 권한 에스컬레이션을 허용합니다.
`baseline`	알려진 권한 에스컬레이션을 방지하는 최소한의 제한적인 제어입니다. 이 보안 표준은 기본(최소로 지정됨) 포드 구성을 허용합니다.
`restricted`	엄격히 제한된 제어로 포드 강화 모범 사례를 따릅니다.

PSA 네임스페이스 레이블

PSA 컨트롤러는 Kubernetes 네임스페이스 수준에서 포드 보안을 적용합니다. 네임스페이스 레이블을 사용하여 지정된 네임스페이스의 포드에 사용할 PSA 모드 및 수준을 정의합니다.

Kubernetes는 네임스페이스에 사용할 표준을 정의하는 데 사용할 수 있는 레이블 집합을 제공합니다. 적용하는 레이블은 PSA 위반이 감지될 경우 Kubernetes 제어부에서 수행되는 작업을 정의합니다. 지정된 Kubernetes 네임스페이스에 대해 임의 또는 모든 모드를 구성하거나 서로 다른 모드에 대해 서로 다른 수준을 설정할 수도 있습니다.

PSA 네임스페이스 레이블 구문은 다음과 같습니다.

```
# MODE must be one of `enforce`, `audit`, or `warn`.
# LEVEL must be one of `privileged`, `baseline`, or `restricted`.
pod-security.kubernetes.io/<MODE>=<LEVEL>
```

보안 표준을 Kubernetes 버전에 고정하는 데 사용할 수 있는 모드별 버전 레이블을 적용할 수도 있습니다. 자세한 내용은 네임스페이스 레이블로 포드 보안 표준 적용을 참조하십시오.

TKG 클러스터에 대한 기본 PSA

기본적으로 Tanzu Kubernetes 릴리스 v1.25로 프로비저닝된 TKG 클러스터는 비시스템 네임스페이스에 대해 PSA 모드 `warn` 및 `audit`이 `restricted`로 설정됩니다. 이것은 비강제 설정으로, PSA 컨트롤러는 포드가 보안을 위반하는 경우 주의 및 감사 알림을 생성하지만 포드는 거부되지 않습니다.

기본적으로 Tanzu Kubernetes 릴리스 v1.26 이상으로 프로비저닝된 TKG 클러스터는 비시스템 네임스페이스에 대해 PSA 모드 `enforce`가 `restricted`로 설정됩니다. 포드가 보안을 위반하면 거부됩니다. 덜 제한적인 제어로 포드를 실행하도록 네임스페이스에서 PSA를 구성해야 합니다.

중요 kube-system, tkg-system 및 vmware-system-cloud-provider 네임스페이스에서 실행되는 일부 시스템 포드에는 상승된 권한이 필요합니다. 이러한 네임스페이스는 포드 보안에서 제외됩니다. 또한 시스템 네임스페이스의 포드 보안을 변경할 수 없습니다.

이 표에는 TKG 클러스터에 대한 기본 PSA 구성이 나열되어 있습니다.

표 18-3. TKG 클러스터에 대한 기본 PSA

TKr 버전	기본 PSA
TKr v1.25	모드: warn | 수준: restricted 모드: audit | 수준: restricted 모드: enforce | 수준: 설정되지 않음
TKr v1.26 이상	모드: enforce | 수준: restricted

네임스페이스 레이블을 사용하여 PSA 구성

Tanzu Kubernetes 릴리스 v1.25의 경우 다음 예제 명령을 사용하여 PSA 주의 및 감사 알림이 생성되지 않도록 지정된 네임스페이스에 대한 보안 수준을 변경합니다.

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/audit=privileged
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/warn=privileged
```

Tanzu Kubernetes 릴리스 v1.26이상의 경우 다음 예제 명령을 사용하여 PSA 표준을 `restricted`에서 `baseline`으로 다운그레이드합니다.

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/enforce=baseline
```

예를 들어 기본 네임스페이스에 `baseline` 표준을 적용하려면 다음을 수행합니다.

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=baseline
```

Tanzu Kubernetes 릴리스 v1.26이상의 경우 다음 예제 명령을 사용하여 PSA 표준을 `restricted`에서 `privileged`으로 다운그레이드합니다.

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/enforce=privileged
```

예를 들어 기본 네임스페이스에 `privileged` 표준을 적용하려면 다음을 수행합니다.

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=privileged
```

Tanzu Kubernetes릴리스 v1.26 이상의 경우 다음 예제 명령을 사용하여 모든 비시스템 네임스페이스에서 PSA를 완화합니다.

```
kubectl label --overwrite ns --all pod-security.kubernetes.io/enforce=privileged
```

```
kubectl label --overwrite ns --all pod-security.kubernetes.io/warn=restricted
```

개별 포드에 대한 보안 컨텍스트 구성

PSA를 위반하는 포드를 실행하려고 하면 이를 나타내는 오류 메시지가 표시됩니다. 예를 들면 다음과 같습니다.

```
{"opType":"CREATE_POD","succeeded":false,"err":"creating pod example-pod: pods
\"example-pod\" is forbidden: violates PodSecurity \"restricted:latest\":
allowPrivilegeEscalation != false (container \"example-container\" must set
securityContext.allowPrivilegeEscalation=false), unrestricted capabilities
(container \"example-container\" must set securityContext.capabilities.drop=[\"ALL\"]),
runAsNonRoot != true (pod or container \"example-container\" must set
securityContext.runAsNonRoot=true),
seccompProfile (pod or container \"example-container\" must set
securityContext.seccompProfile.type to
\"RuntimeDefault\" or \"Localhost\")","events":[]}
```

전체 네임스페이스에 PSA를 설정하는 대신 개별 포드에 대한 보안 컨텍스트를 구성할 수 있습니다. 개별 포드가 실행되도록 허용하려면 포드 규격에서 다음과 같이 보안 컨텍스트를 설정할 수 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - image: gcr.io/google_containers/busybox:1.24
    name: example-container
    command: ["/bin/sh", "-c", "echo 'hello' > /mnt/volume1/index.html && chmod
o+rX /mnt /mnt/volume1/index.html && while true ; do sleep 2 ; done"]
    securityContext:
      allowPrivilegeEscalation: false
      runAsNonRoot: true
      seccompProfile:
        type: "RuntimeDefault"
      capabilities:
        drop: [all]
    volumeMounts:
    - name: example-volume-mount
      mountPath: /mnt/volume1
  restartPolicy: Never
  volumes:
  - name: example-volume
    persistentVolumeClaim:
    claimName: example-pvc
```

TKR 1.24 및 이전 버전에 대한 PSP 구성

감독자의 TKG는 TKR v1.24 및 이전 버전을 사용하는 Tanzu Kubernetes 클러스터에 대해 기본적으로 사용하도록 설정된 포드 보안 정책 승인 컨트롤러를 사용하여 포드 보안을 지원합니다.

사전 요구 사항: TKR 1.24 및 이전 버전

이 항목의 컨텐츠는 TKR v1.24 및 이전 버전으로 프로비저닝된 감독자의 TKG 클러스터에 적용됩니다. 기본적으로 이러한 Tanzu Kubernetes 릴리스는 포드 보안 정책 승인 컨트롤러를 사용하도록 설정합니다.

포드 보안 정책 승인 컨트롤러의 후속 버전은 포드 보안 승인 컨트롤러입니다. TKR v1.25 이상으로 프로비저닝된 감독자의 TKG 클러스터는 포드 보안 승인 컨트롤러를 사용하도록 설정합니다. TKR 1.25 이상에 대한 PSA 구성의 내용을 참조하십시오.

포드 보안 정책 승인 컨트롤러

Kubernetes PSP(포드 보안 정책)는 포드의 보안을 제어하는 클러스터 수준 리소스입니다. PSP를 사용하면 배포할 수 있는 포드 유형 및 배포 가능한 계정 유형을 제어할 수 있습니다.

PodSecurityPolicy 리소스는 포드를 배포하기 위해 포드가 충족해야 하는 조건 집합을 정의합니다. 조건이 충족되지 않으면 포드를 배포할 수 없습니다. 단일 PodSecurityPolicy가 포드를 전체적으로 검증해야 합니다. 두 개의 서로 다른 정책에서 포드의 규칙을 나누어 가질 수 없습니다. 자세한 내용은 Kubernetes 설명서의 포드 보안 정책, RBAC를 참조하십시오.

Kubernetes에서 포드 보안 정책의 사용을 구현하는 방법에는 여러 가지가 있습니다. 일반적인 방식은 RBAC(역할 기반 액세스 제어) 개체를 사용하는 것입니다. ClusterRole 및 ClusterRoleBinding은 클러스터 전체에 적용됩니다. 역할 및 RoleBinding은 특정 네임스페이스에 적용됩니다. RoleBinding이 사용되는 경우에는 포드를 바인딩과 동일한 네임스페이스에만 실행할 수 있습니다. 자세한 내용은 Kubernetes 설명서에서 RBAC를 참조하십시오.

Kubernetes 포드를 생성하는 방법에는 직접과 간접의 두 가지가 있습니다. 사용자 계정을 사용하여 포드 규격을 배포하여 포드를 직접 생성합니다. 배포 또는 DaemonSet 같은 더 높은 수준의 리소스를 정의하여 포드를 간접적으로 생성합니다. 이 경우 서비스 계정은 기본 포드를 생성합니다. 자세한 내용은 Kubernetes 설명서에서 서비스 계정을 참조하십시오.

PSP를 효과적으로 사용하려면 포드 생성 워크플로(직접 및 간접)를 둘 다 고려해야 합니다. 사용자가 포드를 직접 생성하면 사용자 계정에 바인딩된 PSP가 작업을 제어합니다. 사용자가 서비스 계정을 통해 포드를 생성하는 경우 포드를 생성하는 데 사용되는 서비스 계정에 PSP를 바인딩해야 합니다. 포드 규격에 서비스 계정이 지정되지 않은 경우 네임스페이스의 기본 서비스 계정이 사용됩니다.

TKG 클러스터에 대한 기본 PodSecurityPolicy

이 표에는 TKG 클러스터에 대한 권한 있는 및 제한된 기본 포드 보안 정책과 각 정책과 연결된 기본 ClusterRole이 나열 및 설명되어 있습니다.

표 18-4. 연결된 ClusterRole이 있는 기본 PodSecurityPolicy

기본 PSP	사용 권한	설명	연결된 기본 ClusterRole
`vmware-system-privileged`	**임의로 실행**	허용 PSP. PSP 승인 컨트롤러를 사용하도록 설정하지 않고 클러스터를 실행하는 것과 같습니다.	`psp:vmware-system-privileged`가 이 PSP를 사용할 수 있음
`vmware-system-restricted`	**비루트로 실행해야 함**	제한 PSP. 포드 컨테이너에 대한 권한 있는 액세스를 허용하지 않으며, 루트에 대한 가능한 에스컬레이션을 차단하며, 여러 보안 메커니즘을 사용해야 합니다.	`psp:vmware-system-restricted`가 이 PSP를 사용할 수 있음

TKG 클러스터에 대한 역할 및 ClusterRole 바인딩

감독자의 TKG는 TKG 클러스터에 대한 기본 RoleBinding 및 ClusterRoleBinding을 제공하지 않습니다. 사용할 수 있는 예시는 이 설명서에 제공됩니다. TKG 서비스 클러스터에 기본 포드 보안 정책 적용의 내용을 참조하십시오.

vSphere 네임스페이스에 대한 **편집** 권한을 부여받은 vCenter Single Sign-On 사용자는 해당 네임스페이스에 배포된 모든 Tanzu Kubernetes 클러스터에 대한 **cluster-admin** 역할에 할당됩니다. 인증된 클러스터 관리자는 `vmware-system-privileged` PSP를 암시적으로 사용할 수 있습니다. 기술적으로 ClusterRoleBinding은 아니지만 동일한 효과를 갖습니다.

클러스터 관리자는 사용자가 클러스터에 배포할 수 있는 포드 유형을 허용하거나 제한하기 위한 바인딩을 정의해야 합니다. RoleBinding이 사용되는 경우 바인딩은 포드가 바인딩과 동일한 네임스페이스에서만 실행되도록 허용합니다. 이는 네임스페이스에서 실행되는 모든 포드에 대한 액세스 권한을 부여할 시스템 그룹과 연결될 수 있습니다. 클러스터에 인증하는 관리자가 아닌 사용자는 `authenticated` 역할에 할당되며, 이처럼 기본 PSP에 바인딩될 수 있습니다.

포드 보안 정책이 필요한 모든 TKG 클러스터에 대해 다음 동작이 적용됩니다.

- 클러스터 관리자는 자신의 사용자 계정을 사용하여 모든 네임스페이스에 권한 있는 포드를 직접 생성할 수 있습니다.
- 클러스터 관리자는 kube-system 네임스페이스에서 배포, StatefulSets 및 DaemonSet(각각 권한 있는 포드 생성)를 생성할 수 있습니다. 다른 Kubernetes 네임스페이스를 사용하려면 이에 대한 RoleBinding 또는 ClusterRoleBinding을 생성합니다.
- 클러스터 관리자는 자체 PSP(두 개의 기본 PSP 외)를 생성하고 이러한 PSP를 모든 사용자에게 바인딩할 수 있습니다. 자체 PSP를 정의하는 경우 Kubernetes 설명서에서 정책 순서를 참조하십시오.
- 클러스터 관리자가 PSP를 인증된 사용자에게 바인딩할 때까지 인증된 사용자가 권한 있거나 권한 없는 포드를 생성할 수 없습니다.

바인딩 예시는 TKG 서비스 클러스터에 기본 포드 보안 정책 적용의 내용을 참조하십시오.

TKG 서비스 클러스터에 기본 포드 보안 정책 적용

TKR 1.24 이하를 사용하는 TKG 서비스 클러스터에는 권한 있는 워크로드 및 제한된 워크로드 배포를 위해 바인딩할 수 있는 기본 포드 보안 정책이 포함됩니다.

역할 및 ClusterRole 바인딩을 사용하여 기본 포드 보안 정책 적용

vSphere IaaS control plane는 TKR 1.24 이하를 실행하는 감독자의 TKG 클러스터에 적용할 수 있는 TKR 1.24 및 이전 버전에 대한 PSP 구성을 제공합니다. 이 작업은 TKR 1.24 및 이전 버전에 대한 PSP 구성을 참조하는 RoleBinding 및 ClusterRoleBinding 개체를 생성하여 수행할 수 있습니다.

참고 자체 포드 보안 정책을 작성하려면 Kubernetes 설명서를 참조하십시오.

RoleBinding은 특정 네임스페이스 내에서 사용 권한을 부여합니다. ClusterRoleBinding은 클러스터 전체에 사용 권한을 부여합니다. RoleBinding을 사용할지 아니면 ClusterRoleBinding을 사용할지는 사용 사례에 따라 다릅니다. 예를 들어 ClusterRoleBinding을 사용하고 `system:serviceaccounts:<namespace>`를 사용하도록 주체를 구성하는 경우 네임스페이스가 생성되기 전에 PSP에 바인딩할 수 있습니다. 자세한 내용은 Kubernetes 설명서에서 RoleBinding 및 ClusterRoleBinding을 참조하십시오.

다음 섹션에서는 TKR 1.24 및 이전 버전에 대한 PSP 구성을 사용하도록 권한을 부여하는 RoleBinding 및 ClusterRoleBinding 개체를 생성하기 위한 YAML 및 CLI 명령을 제공합니다.

예 1: 권한 있는 워크로드 집합을 실행하는 ClusterRoleBinding

다음 kubectl 명령은 기본 PSP `vmware-system-privileged`를 사용하여 권한 있는 워크로드 집합을 실행하도록 인증된 사용자에게 액세스 권한을 부여하는 ClusterRoleBinding을 생성합니다.

경고 예 1을 선언적으로 또는 명령적으로 적용하면 권한 있는 워크로드를 클러스터 전체에 배포할 수 있습니다. 실제로 예 1은 네이티브 보안 제어를 사용하지 않도록 설정하므로 영향을 충분히 인식하고 주의를 기울여서 사용해야 합니다. 보안을 강화하려면 예 2, 3, 4를 고려하십시오.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: psp:privileged
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - vmware-system-privileged
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: all:psp:privileged
roleRef:
  kind: ClusterRole
  name: psp:privileged
```

```
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

YAML을 적용하는 대신 다음 `kubectl` 명령을 실행할 수 있습니다.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

예 2: 권한 있는 워크로드 집합을 실행하는 RoleBinding

다음 kubectl 명령은 기본 PSP `vmware-system-privileged`를 사용하여 권한 있는 워크로드 집합을 실행하도록 기본 네임스페이스 내의 모든 서비스 계정에 액세스 권한을 부여하는 RoleBinding을 생성합니다.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rolebinding-default-privileged-sa-ns_default
  namespace: default
roleRef:
  kind: ClusterRole
  name: psp:vmware-system-privileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts
```

YAML을 적용하는 대신 다음 kubectl 명령을 실행할 수 있습니다.

```
kubectl create rolebinding rolebinding-default-privileged-sa-ns_default --namespace=default --
clusterrole=psp:vmware-system-privileged --group=system:serviceaccounts
```

예 3: 제한된 워크로드 집합을 실행하는 ClusterRoleBinding

다음 YAML은 기본 PSP `vmware-system-restricted`를 사용하여 제한된 워크로드 집합을 실행하도록 인증된 사용자에게 클러스터 전체 액세스 권한을 부여하는 ClusterRoleBinding을 생성합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: psp:authenticated
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated
```

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp:vmware-system-restricted
```

YAML을 적용하는 대신 다음 kubectl 명령을 실행할 수 있습니다.

```
kubectl create clusterrolebinding psp:authenticated --clusterrole=psp:vmware-system-
restricted --group=system:authenticated
```

예 4: 제한된 워크로드 집합을 실행하는 RoleBinding

다음 YAML은 기본 PSP `vmware-system-restricted`를 사용하여 제한된 워크로드 집합을 실행하도록 특정 네임스페이스 내의 모든 서비스 계정에 액세스 권한을 부여하는 RoleBinding을 생성합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: psp:serviceaccounts
  namespace: some-namespace
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp:vmware-system-restricted
```

YAML을 적용하는 대신 다음 kubectl 명령을 실행할 수 있습니다.

참고 "some-namespace"를 대상 네임스페이스로 변경합니다.

```
kubectl create rolebinding psp:serviceaccounts --clusterrole=psp:vmware-system-restricted --
group=system:serviceaccounts -n some-namespace
```

포드 보안 정책에 대한 역할 예

워크로드를 배포하기 위해 자체 PSP(포드 보안 정책)를 정의하는 경우 이 예를 참조하여 사용자 지정 PSP를 참조하는 Role 또는 ClusterRole을 생성합니다.

이 예는 PodSecurityPolicy에 바인딩된 역할을 보여줍니다. 역할 정의에서 `example-role`에는 직접 정의한 사용자 지정 PSP 리소스에 대한 `use` 동사가 부여됩니다. 또는 기본 PSP 중 하나를 사용합니다. 그런 다음 바인딩을 생성합니다.

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
```

```
metadata:
  name: example-role
  namespace: tkgs-cluster-ns
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - create
  - get
  - list
  - watch
  - update
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - update
  - patch
- apiGroups:
  - extensions
  resourceNames:
  - CUSTOM-OR-DEFAULT-PSP
  resources:
  - podsecuritypolicies
  verbs:
  - use
```

TKG 서비스 클러스터에 대한 TLS 인증서 관리

vSphere IaaS control plane는 TLS(Transport Layer Security) 암호화를 사용하여 구성 요소 간의 통신을 보호합니다. 감독자의 TKG에는 이 암호화 인프라를 지원하는 여러 TLS 인증서가 포함되어 있습니다. 감독자 인증서 순환은 수동입니다. TKG 인증서 순환은 자동화되어 있지만 필요한 경우 수동으로 수행할 수 있습니다.

TKG 서비스 클러스터에 대한 TLS 인증서 정보

vSphere IaaS control plane는 TLS 인증서를 사용하여 다음 구성 요소 간의 통신을 보호합니다.

- vCenter Server
- 감독자 제어부 노드
- vSphere 포드의 작업자 노드로 작동하는 ESXi 호스트
- TKG 클러스터 노드(제어부 및 작업자 모두)

vSphere IaaS control plane는 다음 트러스트 도메인에서 작동하여 시스템 사용자를 인증합니다.

트러스트 도메인	설명
vCenter 트러스트 도메인	이 트러스트 도메인의 TLS 인증서에 대한 기본 서명자는 vCenter Server에 내장된 VMCA(VMware Certificate Authority)입니다.
Kubernetes 트러스트 도메인	이 트러스트 도메인의 TLS 인증서에 대한 기본 서명자는 Kubernetes CA(인증 기관)입니다.

vSphere IaaS control plane 환경에서 사용되는 각 TLS 인증서에 대한 자세한 내용 및 전체 목록은 KB 문서 vSphere with Tanzu 인증서 가이드를 참조하십시오.

TLS 인증서 순환

TLS 인증서 순환 절차는 인증서가 감독자에 사용되는지 또는 TKG 서비스 클러스터에 사용되는지에 따라 다릅니다.

감독자 인증서 순환

감독자용 TLS 인증서는 VMCA 인증서에서 파생됩니다. 감독자 인증서에 대한 자세한 내용은 KB 문서 vSphere with Tanzu 인증서 가이드를 참조하십시오.

감독자 인증서 순환은 수동입니다. WCP Cert Manager 도구를 사용하여 감독자 인증서를 교체하는 방법에 대한 지침은 KB 문서 vSphere with Tanzu 감독자 인증서 교체를 참조하십시오.

TKG 2.0 클러스터 인증서 순환

일반적으로 TKG 클러스터에 대한 TLS 인증서는 수동으로 순환할 필요가 없습니다. TKG 클러스터를 업데이트하면 롤링 업데이트 프로세스가 TLS 인증서를 자동으로 순환하기 때문입니다.

TKG 클러스터에 대한 TLS 인증서가 만료되지 않았고 이러한 인증서를 수동으로 순환해야 하는 경우에는 다음 섹션의 단계를 완료하면 됩니다.

TKG 서비스 클러스터에 대한 TLS 인증서를 수동으로 순환

이 지침에서는 TKG 클러스터 관리에 대한 고급 지식과 경험이 있다고 가정합니다. 또한 이 지침에서는 TLS 인증서가 만료되지 않았다고 가정합니다. 인증서가 만료된 경우 다음 단계를 완료하지 마십시오.

1 이 단계를 실행하려면 SSH를 통해 감독자 노드 중 하나에 연결합니다. Kubernetes 관리자 및 시스템 사용자로 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

2 TKG 클러스터 이름을 가져옵니다.

```
export CLUSTER_NAMESPACE="tkg-cluster-ns"

kubectl get clusters -n $CLUSTER_NAMESPACE
NAME                    PHASE         AGE   VERSION
tkg-cluster             Provisioned   43h
```

3 TKG 클러스터 kubeconfig를 가져옵니다.

```
export CLUSTER_NAME="tkg-cluster"

kubectl get secrets -n $CLUSTER_NAMESPACE $CLUSTER_NAME-kubeconfig -o
jsonpath='{.data.value}' | base64 -d > $CLUSTER_NAME-kubeconfig
```

4 TKG 클러스터 SSH 키를 가져옵니다.

```
kubectl get secrets -n $CLUSTER_NAMESPACE $CLUSTER_NAME-ssh -o jsonpath='{.data.ssh-
privatekey}' | base64 -d > $CLUSTER_NAME-ssh-privatekey
chmod 600 $CLUSTER_NAME-ssh-privatekey
```

5 인증서 순환 전에 환경을 확인합니다.

```
export KUBECONFIG=$CLUSTER_NAME-kubeconfig
```

```
kubectl get nodes -o wide
```

```
kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' \
-l node-role.kubernetes.io/master= > nodes
```

```
for i in `cat nodes`; do
    printf "\n######\n"
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i hostname
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i sudo kubeadm certs check-expiration
done;
```

이전 명령의 예제 결과:

```
tkg-cluster-control-plane-k8bqh
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get
cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE
AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Oct 04, 2023 23:00 UTC
363d                                    no
apiserver                  Oct 04, 2023 23:00 UTC   363d
ca                      no
apiserver-etcd-client      Oct 04, 2023 23:00 UTC   363d            etcd-
ca                  no
apiserver-kubelet-client   Oct 04, 2023 23:00 UTC   363d
ca                      no
controller-manager.conf    Oct 04, 2023 23:00 UTC
363d                                    no
etcd-healthcheck-client    Oct 04, 2023 23:00 UTC   363d            etcd-
ca                  no
```

```
etcd-peer                  Oct 04, 2023 23:00 UTC   363d            etcd-
ca                 no
etcd-server                Oct 04, 2023 23:00 UTC   363d            etcd-
ca                 no
front-proxy-client         Oct 04, 2023 23:00 UTC   363d            front-proxy-
ca          no
scheduler.conf             Oct 04, 2023 23:00 UTC
363d                                 no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Oct 01, 2032 22:56 UTC   9y              no
etcd-ca                 Oct 01, 2032 22:56 UTC   9y              no
front-proxy-ca          Oct 01, 2032 22:56 UTC   9y              no
```

6 TKG 2.0 클러스터에 대한 TLS 인증서를 순환합니다.

다음 단계를 진행하기 전에 컨텍스트를 다시 감독자로 전환합니다.

```
unset KUBECONFIG
kubectl config current-context
kubernetes-admin@kubernetes
```

```
kubectl get kcp  -n $CLUSTER_NAMESPACE $CLUSTER_NAME-control-plane -o
jsonpath='{.apiVersion}{"\n"}'
controlplane.cluster.x-k8s.io/v1beta1
```

```
kubectl get kcp -n $CLUSTER_NAMESPACE $CLUSTER_NAME-control-plane
NAME                         CLUSTER       INITIALIZED   API SERVER AVAILABLE   REPLICAS
READY   UPDATED   UNAVAILABLE   AGE   VERSION
tkg-cluster-control-plane   tkg-cluster   true          true                   3
3       3         0             43h   v1.21.6+vmware.1
```

```
kubectl patch kcp $CLUSTER_NAME-control-plane -n $CLUSTER_NAMESPACE --type merge -p
"{\"spec\":{\"rolloutAfter\":\"`date +'%Y-%m-%dT%TZ'`\"}}"
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/tkg-cluster-control-plane patched
```

시스템 롤아웃 시작됨:

```
kubectl get machines -n $CLUSTER_NAMESPACE
NAME                                              CLUSTER
NODENAME
PROVIDERID                                        PHASE          AGE   VERSION
tkg-cluster-control-plane-k8bqh                   tkg-cluster   tkg-cluster-control-plane-
k8bqh                   vsphere://420a2e04-cf75-9b43-f5b6-23ec4df612eb   Running
43h   v1.21.6+vmware.1
tkg-cluster-control-plane-l7hwd                   tkg-cluster   tkg-cluster-control-plane-
l7hwd                   vsphere://420a57cd-a1a0-fec6-a741-19909854feb6   Running
43h   v1.21.6+vmware.1
tkg-cluster-control-plane-mm6xj                   tkg-cluster   tkg-cluster-control-plane-
mm6xj                   vsphere://420a67c2-ce1c-aacc-4f4c-0564daad4efa   Running
43h   v1.21.6+vmware.1
tkg-cluster-control-plane-nqdv6                   tkg-
cluster
```

```
                  Provisioning   25s   v1.21.6+vmware.1
tkg-cluster-workers-v8575-59c6645b4-wvnlz         tkg-cluster   tkg-cluster-workers-
v8575-59c6645b4-wvnlz         vsphere://420aa071-9ac2-02ea-6530-eb59ceabf87b
Running        43h   v1.21.6+vmware.1
```

시스템 롤아웃 완료됨:

```
kubectl get machines -n $CLUSTER_NAMESPACE
NAME                                                CLUSTER
NODENAME
PROVIDERID                                         PHASE     AGE   VERSION
tkg-cluster-control-plane-m9745                     tkg-cluster   tkg-cluster-control-plane-
m9745                   vsphere://420a5758-50c4-3172-7caf-0bbacaf882d3   Running   17m
v1.21.6+vmware.1
tkg-cluster-control-plane-nqdv6                     tkg-cluster   tkg-cluster-control-plane-
nqdv6                   vsphere://420ad908-00c2-4b9b-74d8-8d197442e767   Running   22m
v1.21.6+vmware.1
tkg-cluster-control-plane-wdmph                     tkg-cluster   tkg-cluster-control-plane-
wdmph                   vsphere://420af38a-f9f8-cb21-e05d-c1bcb6840a93   Running   10m
v1.21.6+vmware.1
tkg-cluster-workers-v8575-59c6645b4-wvnlz         tkg-cluster   tkg-cluster-workers-
v8575-59c6645b4-wvnlz         vsphere://420aa071-9ac2-02ea-6530-eb59ceabf87b   Running
43h   v1.21.6+vmware.1
```

7 TKG 2.0 클러스터에 대한 수동 인증서 순환을 확인합니다.

다음 명령을 실행하여 인증서 순환을 확인합니다.

```
export KUBECONFIG=$CLUSTER_NAME-kubeconfig

kubectl get nodes -o wide
NAME                                          STATUS   ROLES                  AGE
VERSION            INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                KERNEL-
VERSION       CONTAINER-RUNTIME
tkg-cluster-control-plane-m9745             Ready    control-plane,master   15m
v1.21.6+vmware.1   10.244.0.55   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11
tkg-cluster-control-plane-nqdv6             Ready    control-plane,master   21m
v1.21.6+vmware.1   10.244.0.54   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11
tkg-cluster-control-plane-wdmph             Ready    control-plane,master   9m22s
v1.21.6+vmware.1   10.244.0.56   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11
tkg-cluster-workers-v8575-59c6645b4-wvnlz   Ready    <none>                 43h
v1.21.6+vmware.1   10.244.0.51   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11

kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' \
-l node-role.kubernetes.io/master= > nodes

for i in `cat nodes`; do
    printf "\n######\n"
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
```

```
user@$i hostname
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i sudo kubeadm certs check-expiration
done;
```

업데이트된 만료 날짜를 보여주는 예제 결과입니다.

```
######
tkg-cluster-control-plane-m9745
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get
cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE
AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Oct 06, 2023 18:18 UTC
364d                                    no
apiserver                  Oct 06, 2023 18:18 UTC   364d
ca                      no
apiserver-etcd-client      Oct 06, 2023 18:18 UTC   364d            etcd-
ca                  no
apiserver-kubelet-client   Oct 06, 2023 18:18 UTC   364d
ca                      no
controller-manager.conf    Oct 06, 2023 18:18 UTC
364d                                    no
etcd-healthcheck-client    Oct 06, 2023 18:18 UTC   364d            etcd-
ca                  no
etcd-peer                  Oct 06, 2023 18:18 UTC   364d            etcd-
ca                  no
etcd-server                Oct 06, 2023 18:18 UTC   364d            etcd-
ca                  no
front-proxy-client         Oct 06, 2023 18:18 UTC   364d            front-proxy-
ca          no
scheduler.conf             Oct 06, 2023 18:18 UTC
364d                                    no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Oct 01, 2032 22:56 UTC   9y              no
etcd-ca                 Oct 01, 2032 22:56 UTC   9y              no
front-proxy-ca          Oct 01, 2032 22:56 UTC   9y              no
```

8 Kubelet 인증서를 확인합니다.

kubelet 구성의 `rotateCertificates` 매개 변수가 기본 구성인 `true`로 설정되어 있다는 가정 하에 Kubelet 인증서를 순환할 필요가 없습니다.

이 구성은 다음 명령을 사용하여 확인할 수 있습니다.

```
kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' \
-l node-role.kubernetes.io/master!= > workernodes

for i in `cat workernodes`; do
    printf "\n######\n"
```

```
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i hostname
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i sudo grep rotate /var/lib/kubelet/config.yaml
done;
```

예제 결과:

```
######
tkg-cluster-workers-v8575-59c6645b4-wvnlz
rotateCertificates: true
```

NSX 인증서 순환

감독자는 감독자와 NSX 간의 통신에 TLS를 사용합니다. NSX 네트워킹 스택을 사용하여 감독자를 배포한 경우 순환해야 할 수도 있는 다양한 NSX 인증서가 있습니다.

감독자가 사용하는 NSX 인증서

NSX가 있는 WCP는 NSX와의 통합을 위해 다음과 같은 두 가지 인증서를 사용합니다.

- NSX 로드 밸런서 인증서 및 키
- NSX Manager 인증서 및 키

이러한 인증서에 대한 자세한 내용은 "NSX 관리 가이드" 에서 NSX 페더레이션용 인증서를 참조하십시오.

참고 이 항목의 정보는 NSX v3.2를 기준으로 지정됩니다.

NSX 로드 밸런서 인증서 및 키 순환

NSX 로드 밸런서 TLS 인증서 및 키는 **감독자 > 인증서 > NSX 로드 밸런서** 화면에서 순환할 수 있습니다.

- **작업 > CSR 생성**을 선택하여 인증서를 생성합니다.
- **작업 > 인증서 바꾸기**를 선택하여 인증서와 키를 업데이트합니다.

각 NSX Manager 노드에 대한 자체 서명된 인증서 및 키 생성

감독자는 엔터프라이즈 관리자 계정을 사용하여 NSX Manager API에 액세스합니다. NSX Manager 인증서가 만료되면 감독자는 NSX에 액세스할 수 없습니다.

NSX Manager 인증서가 만료되면 감독자 로그를 확인하십시오.

```
tail -f /var/log/vmware/wcp/wcpsvc.log
```

다음과 유사한 오류가 표시될 수 있습니다.

```
error wcp [kubelifecycle/nsx_pi.go:47] ... Error creating WCP service principal identity.
Err: NSX service-wide principal identity creation failed: ... x509: certificate has expired
```

```
error wcp [kubelifecycle/controller.go:554] ... Failed to create WCP service PI in NSX.
Err: WCP service principal idenitity creation failed: NSX service-wide principal identity
creation failed:
... x509: certificate has expired
```

이 문제를 해결하려면 각 NSX Manager 노드에 대한 인증서 및 키를 업데이트합니다. VIP 주소가 있는 3노드 NSX 관리 클러스터를 사용하는 경우에는 감독자가 VIP 주소를 사용하지 않는다는 점에 유의하십시오. 따라서 각 NSX Manager 노드에서 각 인증서를 순환해야 합니다. VIP 인증서를 교체하는 것만으로는 인증서를 순환할 수 없습니다.

1 NSX Manager 노드에 대한 인증서를 순환하려면 이름이 지정된 인증서 서명 요청을 생성하고 아래 내용으로 채웁니다.

   형식 설명:

   - `NSX-MGR-IP-ADDRESS`는 NSX Manager IP 주소입니다.
   - `NSX-MGR-FQDN`은 NSX Manager FQDN 또는 IP 주소입니다.

   `nsx-mgr-01-cert.cnf`

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
distinguished_name = req_distinguished_name
x509_extensions = SAN
req_extensions = v3_ca

[ req_distinguished_name ]
countryName = US
stateOrProvinceName = California
localityName = CA
organizationName = NSX
commonName = NSX-MGR-IP-ADDRESS  #CAN ONLY USE IF SAN IS ALSO USED

[ SAN ]
basicConstraints = CA:false
subjectKeyIdentifier = hash
authorityKeyIdentifier=keyid:always,issuer:always

[ v3_ca ]
subjectAltName = DNS:NSX-MGR-FQDN,IP:NSX-MGR-IP-ADDRESS  #MUST USE
```

예:

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
distinguished_name = req_distinguished_name
x509_extensions = SAN
req_extensions = v3_ca

[ req_distinguished_name ]
countryName = US
stateOrProvinceName = California
localityName = CA
organizationName = NSX
commonName = 10.197.79.122

[ SAN ]
basicConstraints = CA:false
subjectKeyIdentifier = hash
authorityKeyIdentifier=keyid:always,issuer:always

[ v3_ca ]
subjectAltName = DNS:10.197.79.122,IP:10.197.79.122
```

2 OpenSSL을 사용하여 SSL 인증서 및 개인 키를 생성합니다.

```
openssl req -newkey rsa -nodes -days 1100 -x509 -config nsx-mgr-01-cert.cnf -keyout nsx-mgr-01.key -out nsx-mgr-01.crt
```

3 명령을 실행한 후 다음 출력이 표시되는지 확인합니다.

```
Generating a 2048 bit RSA private key
..............+++
..............+++
writing new private key to 'nsx-mgr-01.key'
```

4 3개 파일(초기 서명 요청 및 서명 요청을 실행하여 생성된 인증서 및 개인 키)이 표시되어야 합니다.

5 다음 명령을 실행하여 인증서 및 개인 키를 확인합니다.

```
openssl x509 -in nsx-mgr-01.crt -text -noout
```

6 다중 노드 NSX 관리 클러스터를 사용하는 경우 각 NSX Manager 노드에 대해 이 프로세스를 반복합니다. 인증서 서명 요청에서 IP 주소 및 FQDN을 변경하고 그에 따라 출력 파일 이름을 변경합니다.

SSL 인증서 및 개인 키를 NSX-T 관리 콘솔로 가져오기

다음 단계를 완료하여 각 NSX Manager 노드 인증서 및 개인 키를 NSX로 가져옵니다. `nsx.crt` 및 `nsx.key` 파일을 로컬에 저장한 경우 파일을 NSX에 업로드하거나 컨텐츠를 복사한 후 붙여넣을 수 있습니다.

1 NSX 관리 콘솔에 로그온하고 **시스템 > 인증서** 페이지로 이동합니다.

2 **가져오기 > 인증서 가져오기**를 클릭합니다.

참고 자체 서명된 인증서를 생성했기 때문에 **CA 인증서 가져오기**가 아닌 **인증서 가져오기**를 선택해야 합니다.

3 인증서와 키 쌍의 알아보기 쉬운 **이름**을 입력합니다(예: `nsx-mgr-01-cert-and-key`).

4 인증서 파일을 찾아서 선택하거나 해당 내용(머리글 및 바닥글 포함)을 복사한 후 붙여넣습니다.

예:

```
-----BEGIN CERTIFICATE-----
MIID+zCCAuOgAwIBAgIUCfXaWxNwXvrEFQbt+Dvvp9C/UkIwDQYJKoZIhvcNAQEL
BQAwVTELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExCzAJBgNVBAcM
...
FGlnyT4vxpa2TxvXNTCuXPV9z0VtVBF2QpUJluGH7W1i2wUnApCCXhItcBkfve0f
pCi9YoRoUT8fuMBYo7sL
-----END CERTIFICATE-----
```

5 키를 찾아서 선택하거나 해당 내용(머리글 및 바닥글 포함)을 복사한 후 붙여넣습니다.

예:

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC5GNlUSYHa1p+E
XuOGAsIgiFxUnerRYNm2ARMqRb/xKK6R4xgZhBmpmikpE90vQibvouHqnL13owq7
...
OzbnwMCUI2TeY1iJNx3HNKUrdLvrr8CMh7Looe0L4/2j7ygew2x2C5m272SCJYs/
ly+bOXEYaH4/ORHbvvr0jQ==
-----END PRIVATE KEY-----
```

6 **서비스 인증서** 옵션에 대해 **아니요**를 선택합니다.

7 인증서와 키 쌍에 대한 설명(예: `Cert and Private Key for NSX Manager Node 1`)을 입력합니다.

8 **가져오기**를 클릭합니다.

9 각 NSX Manager 인증서와 키 쌍에 대해 이 프로세스를 반복합니다.

NSX API를 사용하여 NSX Manager 인증서 등록

인증서와 키를 NSX Manager에 업로드했으면 NSX API를 사용하여 등록합니다. "NSX 관리 가이드" 에서 인증서 가져오기 및 교체도 참조하십시오.

1 NSX Manager에서 **시스템 > 인증서**를 선택합니다.

2 ID 열에서 등록하려는 인증서의 ID를 선택하고 팝업 창에서 인증서 ID를 복사합니다.

3 다음 API 호출을 실행하여 인증서를 나열합니다. 업데이트할 각 인증서의 인증서 노드 ID를 가져옵니다.

```
GET https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates
```

4 다음 API 호출을 실행하여 인증서의 유효성을 검사합니다.

```
GET https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>?
action=validate
```

예:

```
https://10.19.92.133/api/v1/trust-management/certificates/070bae44-7548-45ff-
a884-578f079eb6d4?action=validate
```

5 다음 API 호출을 실행하여 NSX Manager 노드의 인증서를 교체합니다.

```
POST https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>?
action=apply_certificate&service_type=API&node_id=<node-id>
```

예:

```
POST https://10.19.92.133/api/v1/trust-management/certificates/070bae44-7548-45ff-
a884-578f079eb6d4?
action=apply_certificate&service_type=API&node_id=e61c7537-3090-4149-b2b6-19915c20504f
```

6 다중 노드 NSX 관리 클러스터를 사용하는 경우 각 NSX Manager 노드에 대해 인증서 교체 프로세스를 반복합니다.

7 완료되면 교체한 각각의 만료된 인증서를 삭제합니다. NSX Manager 인터페이스 또는 NSX API를 사용하여 이 작업을 수행할 수 있습니다.

예:

```
https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>
```

TMC를 TKG 서비스 클러스터와 통합

19

이 섹션에서는 Tanzu Mission Control을 TKG 서비스 클러스터와 통합하는 것과 관련한 항목을 제공합니다.

다음으로 아래 항목을 읽으십시오.

- 감독자로 호스팅되는 Tanzu Mission Control 등록
- Tanzu Mission Control Self-Managed를 감독자에 등록

감독자로 호스팅되는 Tanzu Mission Control 등록

감독자의 Tanzu Kubernetes Grid를 Tanzu Mission Control과 통합할 수 있습니다. 이렇게 하면 TMC 웹 인터페이스를 사용하여 TKG 서비스 클러스터를 관리할 수 있습니다.

Tanzu Mission Control에서 TKG 서비스 클러스터 사용

Tanzu Mission Control 및 Tanzu Application Platform에서 TKG 서비스 클러스터를 사용하기 위한 일반적인 워크플로는 다음과 같습니다.

단계	작업
1	TKG 워크로드 클러스터 프로비저닝 장 7 TKG 서비스 클러스터 프로비저닝 항목을 참조하십시오.
2	TMC에 감독자 등록 Tanzu Mission Control에 감독자 등록 항목을 참조하십시오.
3	TMC에 TKG 워크로드 클러스터 연결 Tanzu Kubernetes 클러스터 수명 주기 관리를 참조하십시오.
4	Tanzu Application Platform 설치 Tanzu Application Platform 설명서를 참조하십시오.

Tanzu Mission Control에 감독자 등록

감독자는 Tanzu Mission Control 전용 vSphere 네임스페이스와 함께 제공됩니다. 감독자를 Tanzu Mission Control에 등록하려면 vSphere 네임스페이스에 TMC 에이전트를 설치하고 TMC를 사용하여 등록 프로세스를 완료합니다.

이 절차를 완료하면 감독자에 TMC 에이전트가 설치됩니다. 이 절차를 수행하려면 vSphere Client와 TMC 웹 인터페이스를 동시에 사용해야 합니다.

1 kubectl용 vSphere 플러그인을 사용하여 감독자로 인증합니다.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 컨텍스트를 감독자로 전환합니다. 예:

```
kubectl config use-context 10.199.95.59
```

3 다음 명령을 실행하여 네임스페이스를 나열합니다.

```
kubectl get ns
```

4 TMC에 대한 vSphere 네임스페이스의 이름은 `svc-tmc-cXX`(여기서 XX는 숫자임)입니다. 이것이 존재하고 활성 상태인지 확인합니다.

5 Tanzu Mission Control 웹 인터페이스에 로그인합니다.

6 Tanzu Mission Control에 감독자를 등록합니다. Tanzu Mission Control에 관리 클러스터 등록을 참조하십시오.

7 Tanzu Mission Control 웹 인터페이스를 사용하여 감독자에 대한 등록 URL(**관리 > 관리 클러스터** 페이지에서 사용 가능)을 복사합니다.

8 필요한 경우 Tanzu Mission Control에 필요한 포트에 대해 감독자에서 방화벽 포트를 엽니다(일반적으로 `443`). 클러스터 에이전트 확장에 의한 아웃바운드 연결을 참조하십시오.

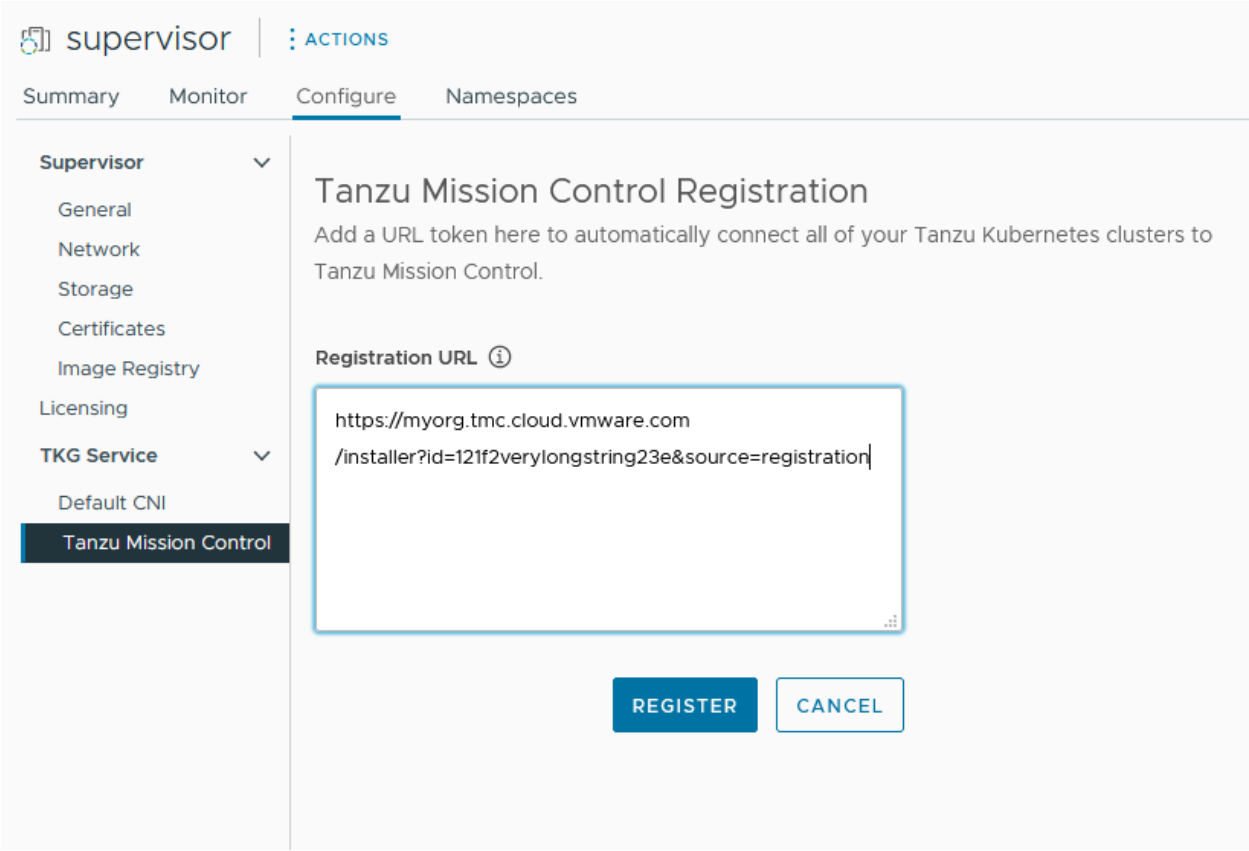
9 vSphere Client를 사용하여 vSphere IaaS control plane 환경에 로그인합니다.

10 **워크로드 관리** 및 대상 감독자를 선택합니다.

11 **구성**을 선택하고 **TKG 서비스 > Tanzu Mission Control**을 선택합니다.

12 **등록 URL** 필드에 등록 URL을 붙여넣습니다.

13 **등록**을 클릭합니다.



감독자가 TMC에 등록되면 TMC 웹 인터페이스를 사용하여 TKG 클러스터를 프로비저닝하고 관리합니다. 자세한 내용은 Tanzu Mission Control 설명서를 참조하십시오.

감독자에서 Tanzu Mission Control 에이전트 제거

감독자에서 Tanzu Mission Control 에이전트를 제거하려면 vSphere with Tanzu의 감독자 클러스터에서 클러스터 에이전트를 수동으로 제거를 참조하십시오.

Tanzu Mission Control Self-Managed를 감독자에 등록

Tanzu Mission Control Self-Managed를 감독자에 등록하려면 TMC 에이전트에 대한 사용자 지정 리소스 정의를 생성하고 적용합니다.

Tanzu Mission Control Self-Managed 정보

Tanzu Mission Control Self-Managed에 대한 자세한 내용(설치 및 구성 방법 포함)은 설명서VMware Tanzu Mission Control Self-Managed 설치 및 실행을 참조하십시오.

Tanzu Mission Control Self-Managed를 감독자에 등록

Tanzu Mission Control Self-Managed를 감독자와 통합하려면 TMC 에이전트를 참조하는 사용자 지정 리소스 정의를 생성합니다. 감독자는 에이전트가 설치된 TMC용 Kubernetes 네임스페이스를 포함합니다.

다음 절차를 완료합니다.

1 설명서에 설명된 대로 Tanzu Mission Control Self-Managed를 설치합니다. VMware Tanzu Mission Control Self-Managed 설치 및 실행을 참조하십시오.

2 웹 브라우저를 사용하여 Tanzu Mission Control Self-Managed 로컬 배포에 액세스합니다.

3 Tanzu Mission Control Self-Managed 설치를 위해 루트 CA 인증서를 내보냅니다.

- 잘 알려진 CA를 사용하는 경우 브라우저의 주소 표시줄 왼쪽에 있는 잠금 아이콘을 클릭하고 인증서를 봅니다. 개인 CA를 사용하는 경우 [보안되지 않음] 버튼을 클릭하고 인증서를 봅니다.
- 인증서 대화상자 팝업에서 `Details` 탭을 선택한 다음 `Export` 버튼을 선택하여 CA 인증서 사본을 다운로드합니다.
- 원하는 텍스트 편집기를 사용하여 CA 인증서 파일을 열고 CA 인증서 컨텐츠에 액세스합니다.

4 kubectl용 vSphere 플러그인을 사용하여 감독자로 인증합니다.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

5 다음 명령을 실행하여 사용 가능한 Kubectl 컨텍스트를 나열합니다.

```
kubectl config get-contexts
```

6 Tanzu Mission Control Self-Managed를 실행하는 TKG 클러스터가 프로비저닝된 대상 vSphere 네임스페이스로 컨텍스트를 전환합니다.

```
kubectl config use-context VSPHERE-NAMESPACE
```

7 다음 명령을 실행하여 Kubernetes 네임스페이스를 나열합니다.

```
kubectl get ns
```

8 TMC용 감독자의 Kubernetes 네임스페이스는 이름이 `svc-tmc-cXXXX`(여기서 XXXX는 숫자임)입니다. 예를 들면 `svc-tmc-c1208`입니다. 이 Kubernetes 네임스페이스가 존재하고 활성 상태인지 확인합니다.

9 텍스트 편집기를 사용하여 이름이 `agentconfig.yaml`인 사용자 지정 리소스 정의를 생성합니다. 이 파일에는 TMC 네임스페이스, TMC Self-Managed 배포의 호스트 이름 및 CA 인증서 컨텐츠가 포함됩니다.

- `namespace` 필드에 TMC용 Kubernetes 네임스페이스의 이름을 입력합니다.
- `caCerts` 필드에 CA 인증서를 입력합니다.
- `allowedHostNames` 필드에 TMC 호스트 이름을 입력합니다.

```
apiVersion: "installers.tmc.cloud.vmware.com/v1alpha1"
kind: "AgentConfig"
metadata:
  name: "tmc-agent-config"
  namespace: "<namespace>"
spec:
  caCerts: |-
```

```
    -----BEGIN CERTIFICATE-----
    Certificate1
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    Certificate2
    -----END CERTIFICATE-----
  allowedHostNames:
    - "google.com"
```

10 생성한 `AgentConfig` yaml 파일을 적용합니다.

```
kubectl apply -f agentconfig.yaml
```

11 Tanzu Mission Control Self-Managed 설치를 위한 웹 인터페이스를 사용하여 등록 프로세스를 완료합니다. 자세한 내용은 Tanzu Mission Control 설명서를 참조하십시오.

TKG 서비스 클러스터 및 워크로드 백업 및 복원

20

TKG 서비스 클러스터 및 워크로드를 백업하고 복원하려면 이 섹션을 참조하십시오.

다음으로 아래 항목을 읽으십시오.

- TKG 서비스 클러스터 및 워크로드 백업 및 복원에 대한 고려 사항
- vSphere용 Velero 플러그인를 사용하여 워크로드 백업 및 복원
- Restic과 함께 독립형 Velero를 사용하여 감독자에서 TKG 클러스터 워크로드 백업 및 복원
- CSI 스냅샷과 함께 Velero를 사용하여 백업 및 복원

TKG 서비스 클러스터 및 워크로드 백업 및 복원에 대한 고려 사항

이 항목에서는 TKG 서비스 클러스터에서 실행되는 워크로드 백업 및 복원에 대한 고려 사항을 제공합니다.

TKG 서비스 클러스터 백업 및 복원

TKG 클러스터를 백업하고 복원하려면 감독자 데이터베이스를 백업합니다. 이렇게 하면 vSphere 네임스페이스 개체 및 TKG 클러스터 노드 VM을 복원할 수 있습니다.

vCenter Server 관리 인터페이스를 통해 사용할 수 있는 vCenter Server 백업 기능을 사용하여 감독자 백업 및 복원을 사용하도록 설정합니다. 자세한 내용은 vSphere IaaS control plane에 대한 백업 복원 자료를 참조하십시오.

참고 감독자 백업은 TKG 클러스터 노드 VM을 복원하는 데에만 사용할 수 있습니다. 감독자 백업을 사용하여 TKG 클러스터에 배포된 워크로드를 복원할 수 없습니다. 워크로드를 별도로 백업한 다음, 클러스터가 복원된 후에 워크로드를 복원해야 합니다.

TKG 서비스 클러스터에서 실행되는 워크로드 백업 및 복원

이 표에는 TKG 클러스터에서 실행되는 상태 비저장 및 상태 저장 워크로드를 백업 및 복원하기 위한 옵션이 요약되어 있습니다.

참고 독립형 Velero를 사용하여 Kubernetes 클러스터를 백업하고 복원하면 이식성이 지원됩니다. 즉, TKG 서비스에 의해 프로비저닝되지 않은 Kubernetes 클러스터로 클러스터 워크로드를 복원하려면 독립형 Velero를 사용하는 것이 좋습니다.

시나리오	도구	주석
감독자의 TKG 클러스터에서 상태 비 저장 및 상태 저장 워크로드를 백업하고 감독자의 TKG 클러스터로 복원합니다.	Velero Plugin for vSphere vSphere용 Velero 플러그인을 사용하여 워크로드 백업 및 복원의 내용을 참조하십시오.	Kubernetes 메타데이터와 영구 볼륨을 모두 백업하고 복원할 수 있습니다. Velero 스냅샷 생성은 상태 저장 애플리케이션이 있는 영구 볼륨에 사용됩니다. vSphere용 Velero 플러그인도 감독자에 설치 및 구성되어 있어야 합니다.
감독자의 TKG 클러스터에서 상태 비 저장 및 상태 저장 워크로드를 백업하고 규격 Kubernetes 클러스터로 복원합니다.	독립형 Velero 및 Restic Restic과 함께 독립형 Velero를 사용하여 감독자에서 TKG 클러스터 워크로드 백업 및 복원 항목을 참조하십시오. .	Kubernetes 메타데이터와 영구 볼륨을 모두 백업하고 복원할 수 있습니다. Restic은 상태 저장 애플리케이션이 있는 영구 볼륨에 사용됩니다. 이식성이 필요한 경우 이 방식을 사용합니다.
감독자의 TKG 클러스터에서 상태 비 저장 및 상태 저장 워크로드를 백업하고 규격 Kubernetes 클러스터로 복원합니다.	CSI 스냅샷 생성 기능을 갖춘 독립형 Velero CSI 스냅샷과 함께 Velero를 사용하여 백업 및 복원 항목을 참조하십시오.	vSphere 8.0에는 vSphere 8.0 U2+ 및 TKr v1.26+가 필요합니다.

vSphere용 Velero 플러그인을 사용하여 워크로드 백업 및 복원

이 섹션에서는 vSphere용 Velero 플러그인을 사용하여 클러스터 워크로드를 백업 및 복원하는 방법에 대한 항목을 제공합니다.

TKG 클러스터에 vSphere용 Velero 플러그인 설치 및 구성

TKGS 클러스터에 vSphere용 Velero 플러그인을 설치하면 vSphere용 Velero 플러그인을 사용하여 해당 클러스터에서 실행되는 워크로드를 백업하고 복원할 수 있습니다.

개요

vSphere용 Velero 플러그인는 TKGS 클러스터 워크로드를 백업하고 복원하기 위한 솔루션을 제공합니다. 영구 워크로드의 경우 vSphere용 Velero 플러그인를 사용하여 영구 볼륨의 스냅샷을 생성할 수 있습니다.

참고 백업 및 복원하려는 TKGS 클러스터 워크로드에 대해 이식성이 필요한 경우에는 vSphere용 Velero 플러그인를 사용하지 마십시오. Kubernetes 클러스터 간 이식성을 위해서는 독립형 Velero를 Restic과 함께 사용하십시오.

사전 요구 사항: 감독자에 vSphere용 Velero 플러그인 설치

TKGS 클러스터에 vSphere용 Velero 플러그인를 설치하려면 감독자에 vSphere용 Velero 플러그인가 설치되어 있어야 합니다. 또한 감독자가 NSX 네트워킹으로 구성되어야 합니다. #unique_22의 내용을 참조하십시오.

1단계: Linux Workstation에서 Velero CLI 설치

Velero CLI는 Velero와 상호 작용하기 위한 표준 도구입니다. Velero CLI는 vSphere용 Velero 플러그인 CLI(`velero-vsphere`)보다 더 많은 기능을 제공하며 Tanzu Kubernetes 클러스터 워크로드를 백업하고 복원하는 데 필요합니다.

Linux 워크스테이션에 Velero CLI를 설치합니다. 이상적으로는 `kubectl`, `kubectl-vsphere`, `velero-vsphere`를 포함한 vSphere IaaS control plane 환경에 대해 연결된 CLI를 실행하는 Linux와 동일한 점프 호스트입니다.

Velero 버전 번호는 `X.Y.Z`로 표시됩니다. 사용할 특정 버전은 Velero 호환성 매트릭스를 참조하고 명령을 실행할 때 그에 따라 대체하십시오.

Velero CLI를 설치하려면 다음 단계를 완료합니다.

1 다음 명령을 실행합니다.

```
$ wget https://github.com/vmware-tanzu/velero/releases/download/vX.Y.Z/velero-vX.Y.Z-linux-amd64.tar.gz
$ gzip -d velero-vX.Y.Z-linux-amd64.tar.gz && tar -xvf velero-vX.Y.Z-linux-amd64.tar
$ export PATH="$(pwd)/velero-vX.Y.Z-linux-amd64:$PATH"

$ which velero
/root/velero-vX.Y.Z-linux-amd64/velero
```

2 Velero CLI 설치를 확인합니다.

```
velero version

Client:
    Version: vX.Y.Z
```

2단계: S3 호환 버킷 세부 정보 가져오기

편의를 위해 이 단계에서는 감독자에 vSphere용 Velero 플러그인를 설치할 때 구성한 것과 동일한 S3 호환 개체 저장소를 사용 중이라고 가정합니다. 운영 환경에서는 별도의 개체 저장소를 생성하는 것이 좋습니다.

vSphere용 Velero 플러그인를 설치하려면 S3 호환 개체 저장소에 대한 다음 정보를 제공해야 합니다.

데이터 항목	예제 값
s3Url	`http://my-s3-store.example.com`
aws_access_key_id	ACCESS-KEY-ID-STRING
aws_secret_access_key	SECRET-ACCESS-KEY-STRING

다음 정보를 사용하여 암호 파일 이름 `s3-credentials`를 생성합니다. 이 파일은 vSphere용 Velero 플러그인를 설치할 때 참조합니다.

```
aws_access_key_id = ACCESS-KEY-ID-STRING
aws_secret_access_key = SECRET-ACCESS-KEY-STRING
```

3단계 옵션 A: 레이블을 사용하여 TKG 클러스터에 vSphere용 Velero 플러그인 설치(새로운 방법)

vSphere 8 업데이트 3 이상을 사용하는 경우 레이블을 추가하여 TKG 클러스터에 vSphere용 Velero 플러그인를 자동으로 설치할 수 있습니다.

1 백업 스토리지 위치에 액세스할 수 있는지 확인합니다.

2 Velero vSphere Operator Core 감독자 서비스가 활성화되었는지 확인합니다.

```
kubectl get ns | grep velero
svc-velero-domain-c9                        Active   18d
```

3 이름이 `velero`인 Kubernetes 네임스페이스가 감독자에 생성되었는지 확인합니다.

```
kubectl get ns | grep velero
svc-velero-domain-c9                        Active   18d
velero                                      Active   1s
```

4 감독자에서 vSphere용 Velero 플러그인 감독자 서비스를 사용하도록 설정되어 있는지 확인합니다.

```
velero version
Client:
        Version: v1.11.1
        Git commit: bdbe7eb242b0f64d5b04a7fea86d1edbb3a3587c
Server:
        Version: v1.11.1
```

```
kubectl get veleroservice -A
NAMESPACE   NAME      AGE
velero      default   53m
```

```
velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX   PHASE       LAST VALIDATED                  ACCESS
MODE   DEFAULT
default   aws        velero          Available   2023-11-20 14:10:57 -0800 PST
ReadWrite     true
```

5 클러스터에 velero 레이블을 추가하여 대상 TKG 클러스터에 대해 Velero를 사용하도록 설정합니다.

```
kubectl label cluster CLUSTER-NAME --namespace CLUSTER-NS velero.vsphere.vmware.com/
enabled=true
```

참고 이 작업은 클러스터가 프로비저닝될 때 vSphere 네임스페이스에서 수행됩니다.

6 Velero가 설치되고 클러스터에 사용할 준비가 되었는지 확인합니다.

```
kubectl get ns
NAME                                STATUS   AGE
...
velero                              Active   2m    <--
velero-vsphere-plugin-backupdriver  Active   2d23h
```

```
kubectl get all -n velero
NAME                                READY   STATUS    RESTARTS   AGE
pod/backup-driver-5945d6bcd4-gtw9d  1/1     Running   0          17h
pod/velero-6b9b49449-pq6b4          1/1     Running   0          18h
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backup-driver     1/1     1            1           17h
```

```
deployment.apps/velero          1/1     1            1           18h
NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/backup-driver-5945d6bcd4   1         1         1       17h
replicaset.apps/velero-6b9b49449           1         1         1       18h
```

```
velero version
Client:
        Version: v1.11.1
        Git commit: bdbe7eb242b0f64d5b04a7fea86d1edbb3a3587c
Server:
        Version: v1.11.1
```

3단계 옵션 B: 수동으로 TKG 클러스터에 vSphere용 Velero 플러그인 설치(기존 방법)

Velero CLI를 사용하여 백업 및 복원하려는 대상 TKG 클러스터에 vSphere용 Velero 플러그인를 설치하려고 합니다.

Velero CLI 컨텍스트는 `kubectl` 컨텍스트를 자동으로 따릅니다. Velero CLI 명령을 실행하여 대상 클러스터에 Velero 및 vSphere용 Velero 플러그인를 설치하기 전에 `kubectl` 컨텍스트를 대상 클러스터로 설정해야 합니다.

1 kubectl용 vSphere 플러그인을 사용하여 감독자를 인증합니다.

2 `kubectl` 컨텍스트를 대상 TKG 클러스터로 설정합니다.

```
kubectl config use-context TARGET-TANZU-KUBERNETES-CLUSTER
```

3 TKG 클러스터에서 `velero-vsphere-plugin-config.yaml`이라는 Velero 플러그인에 대한 configmap을 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: velero-vsphere-plugin-config
data:
 cluster_flavor: GUEST
```

TKG 클러스터에 configmap을 적용합니다.

```
kubectl apply -n <velero-namespace> -f velero-vsphere-plugin-config.yaml
```

configmap을 설치하지 않으면 vSphere용 Velero 플러그인를 설치하려고 할 때 다음 오류가 표시됩니다.

```
Error received while retrieving cluster flavor from config, err: configmaps "velero-
vsphere-plugin-config" not found
Falling back to retrieving cluster flavor from vSphere CSI Driver Deployment
```

4 다음 Velero CLI 명령을 실행하여 대상 클러스터에 Velero를 설치합니다.

BUCKET-NAME, **REGION**(두 개의 인스턴스) 및 **s3Url** 필드에 대한 자리 표시자 값을 적절한 값으로 바꿉니다. 앞의 지침에서 벗어난 값(암호 파일의 이름 또는 위치, 수동으로 생성된 `velero` 네임스페이스의 이름 등)이 있으면 해당 값도 조정합니다.

```
./velero install --provider aws \
--bucket BUCKET-NAME \
--secret-file ./s3-credentials \
--features=EnableVSphereItemActionPlugin \
--plugins velero/velero-plugin-for-aws:vX.Y.Z \
--snapshot-location-config region=REGION \
--backup-location-config region=REGION,s3ForcePathStyle="true",s3Url=http://my-s3-
store.example.com
```

5 대상 클러스터에 vSphere용 Velero 플러그인를 설치합니다. 설치된 Velero는 Kubernetes API 서버와 통신하여 플러그인을 설치합니다.

```
velero plugin add vsphereveleroplugin/velero-plugin-for-vsphere:vX.Y.Z
```

부록: TKG 클러스터에서 vSphere용 Velero 플러그인 제거

vSphere용 Velero 플러그인를 제거하려면 다음 단계를 완료합니다.

1 `kubectl` 컨텍스트를 대상 Tanzu Kubernetes 클러스터로 전환합니다.

```
kubectl config use-context TARGET-TANZU-KUBERNETES-CLUSTER
```

2 플러그인을 제거하려면 다음 명령을 실행하여 Velero 배포에서 velero-plugin-for-vsphere의 InitContainer를 제거합니다.

```
velero plugin remove vsphereveleroplugin/velero-plugin-for-vsphere:vX.Y.Z
```

3 프로세스를 완료하려면 백업 드라이버 배포 및 관련 CRD를 삭제합니다.

```
kubectl -n velero delete deployment.apps/backup-driver
```

```
kubectl delete crds \
backuprepositories.backupdriver.cnsdp.vmware.com \
backuprepositoryclaims.backupdriver.cnsdp.vmware.com \
clonefromsnapshots.backupdriver.cnsdp.vmware.com \
deletesnapshots.backupdriver.cnsdp.vmware.com \
snapshots.backupdriver.cnsdp.vmware.com
```

```
kubectl delete crds uploads.datamover.cnsdp.vmware.com downloads.datamover.cnsdp.vmware.com
```

vSphere용 Velero 플러그인를 사용하여 TKG 클러스터 워크로드 백업 및 복원

vSphere용 Velero 플러그인를 사용하여 감독자의 TKG 클러스터에서 실행되는 워크로드를 백업 및 복원할 수 있습니다.

사전 요구 사항

vSphere용 Velero 플러그인를 사용하여 TKG 클러스터 워크로드를 백업 및 복원하려면 먼저 대상 클러스터에 vSphere용 Velero 플러그인를 설치해야 합니다. TKG 클러스터에 vSphere용 Velero 플러그인 설치 및 구성의 내용을 참조하십시오.

워크로드 백업

다음은 Velero 백업을 생성하는 명령의 예입니다.

```
velero backup create <backup name> --include-namespaces=my-namespace
```

모든 로컬 스냅샷이 생성되고 Kubernetes 메타데이터(볼륨 스냅샷 제외)가 개체 저장소에 업로드되면 Velero 백업이 `Completed`로 표시됩니다. 이 시점에 비동기 데이터 이동 작업(즉 볼륨 스냅샷 업로드)이 백그라운드에서 계속 진행되며 완료하는 데 다소 시간이 걸릴 수 있습니다. 볼륨 스냅샷의 상태는 스냅샷 CR(사용자 지정 리소스)을 모니터링하여 확인할 수 있습니다.

스냅샷

스냅샷은 영구 볼륨을 백업하는 데 사용됩니다. 각 볼륨 스냅샷에 대해 스냅샷이 생성된 PVC(영구 볼륨 할당)와 동일한 네임스페이스에 스냅샷 CR이 생성됩니다.

다음 명령을 실행하여 PVC 네임스페이스의 모든 스냅샷을 가져올 수 있습니다.

```
kubectl get -n <pvc namespace> snapshot
```

스냅샷 CRD(사용자 지정 리소스 정의)에는 `.status.phase` 필드에 대해 다음을 포함한 여러 단계가 있습니다.

스냅샷 단계	설명
신규	아직 처리되지 않았습니다.
Snapshotted	로컬 스냅샷이 생성되었습니다.
SnapshotFailed	로컬 스냅샷을 생성하지 못했습니다.
Uploading	스냅샷이 업로드되고 있습니다.
Uploaded	스냅샷이 업로드되었습니다.
UploadFailed	스냅샷을 업로드하지 못했습니다.
Canceling	스냅샷 업로드가 취소되고 있습니다.
Canceled	스냅샷 업로드가 취소되었습니다.
CleanupAfterUploadFailed	스냅샷 업로드 후 로컬 스냅샷을 정리하지 못했습니다.

워크로드 복원

다음은 Velero 복원 명령의 예입니다.

```
velero restore create --from-backup <velero-backup-name>
```

볼륨 스냅샷 및 기타 Kubernetes 메타데이터가 현재 클러스터에 성공적으로 복원되면 Velero 복원이 `Completed`로 표시됩니다. 이때 이 복원과 관련된 vSphere 플러그인의 모든 작업도 완료됩니다. Velero 백업의 경우처럼 백그라운드에서 비동기식 데이터 이동 작업이 없습니다.

CloneFromSnapshot

각 볼륨 스냅샷에서 복원하기 위해 원래 스냅샷이 생성된 PVC와 동일한 네임스페이스에 `CloneFromSnapshot` CR(사용자 지정 리소스)이 생성됩니다. 다음 명령을 실행하여 PVC 네임스페이스의 모든 CloneFromSnapshot을 가져올 수 있습니다.

```
kubectl -n <pvc namespace> get clonefromsnapshot
```

CloneFromSnapshot CRD에는 `.status.phase` 필드에 대한 몇 가지 주요 단계가 있습니다.

스냅샷 단계	설명
신규	스냅샷에서 복제가 완료되지 않았습니다.
InProgress	원격 저장소에서 vSphere 볼륨 스냅샷을 다운로드하고 있습니다.
Completed	스냅샷에서 복제가 완료되었습니다.
실패	스냅샷에서 복제에 실패했습니다.

Restic과 함께 독립형 Velero를 사용하여 감독자에서 TKG 클러스터 워크로드 백업 및 복원

이 섹션에서는 Restic과 함께 독립형 Velero를 사용하여 감독자에서 실행되는 TKG 클러스터 워크로드를 백업 및 복원하는 방법에 대한 항목을 제공합니다.

TKG 클러스터에 독립형 Velero 및 Restic 설치 및 구성

감독자의 TKG 클러스터에서 실행되는 워크로드를 백업 및 복원하려면 데이터스토어를 생성하고 Kubernetes 클러스터에 Velero를 Restic과 함께 설치합니다.

개요

TKG 클러스터는 가상 시스템 노드에서 실행됩니다. TKG 클러스터 워크로드를 백업하고 복원하려면 클러스터에 Velero 및 Restic을 설치합니다.

사전 요구 사항

Tanzu Kubernetes 클러스터에서 실행되는 워크로드를 백업하고 복원하기 위해 Velero 및 Restic을 설치하려면 환경이 다음 사전 요구 사항을 충족하는지 확인합니다.

- 여러 워크로드 백업을 저장하기에 충분한 스토리지가 있는 Linux VM. 이 VM에 MinIO를 설치합니다.
- kubectl용 vSphere 플러그인 및 kubectl을 포함하여 vSphere에 대한 Kubernetes CLI 도구가 설치되어 있는 Linux VM. 이 클라이언트 VM에 Velero CLI를 설치합니다. 이러한 VM이 없는 경우 Velero CLI를 로컬로 설치할 수 있지만 그에 따라 설치 단계를 조정해야 합니다.

- Kubernetes 환경을 인터넷에 액세스할 수 있으며 클라이언트 VM에서 연결할 수 있습니다.

MinIO 개체 저장소 설치 및 구성

Velero에는 Kubernetes 워크로드 백업의 대상으로 S3 호환 개체 저장소가 필요합니다. Velero는 이러한 개체 저장소 제공자를 여러 개 지원합니다. 간단히 하기 위해 이 지침에서는 개체 저장소 VM에서 로컬로 실행되는 S3 호환 스토리지 서비스인 MinIO를 사용합니다.

1 MinIO를 설치합니다.

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

2 MinIO에 실행 권한을 부여합니다.

```
chmod +x minio
```

3 MinIO용 파일 시스템에 디렉토리를 생성합니다.

```
mkdir /DATA-MINIO
```

4 MinIO 서버를 시작합니다.

```
./minio server /DATA-MINIO
```

5 MinIO 서버가 시작되면 끝점 URL, AccessKey 및 SecretKey를 비롯한 중요 데이터스토어 인스턴스 세부 정보가 제공됩니다. 테이블에 끝점 URL, AccessKey 및 SecretKey를 기록하십시오.

데이터스토어 메타데이터	값
끝점 URL	
AccessKey	
SecretKey	

6 브라우저에서 MinIO 서버 끝점 URL을 열어 MinIO 데이터스토어로 이동합니다.

7 MinIO 서버에 로그인하고 AccessKey 및 SecretKey를 제공합니다.

8 MinIO as a Service를 사용하도록 설정하려면 `minio.service` 스크립트를 다운로드하여 자동 시작을 위해 MinIO를 구성합니다.

```
curl -O https://raw.githubusercontent.com/minio/minio-service/master/linux-systemd/
minio.service
```

9 `minio.service` 스크립트를 편집하고 `ExecStart`에 대해 다음 값을 추가합니다.

```
ExecStart=/usr/local/bin/minio server /DATA-MINIO path
```

10 수정된 스크립트를 저장합니다.

11 다음 명령을 실행하여 MinIO 서비스를 구성합니다.

```
cp minio.service /etc/systemd/system
cp minio /usr/local/bin/
systemctl daemon-reload
systemctl start minio
systemctl status minio
systemctl enable minio
```

12 MinIO 브라우저를 시작하고 개체 저장소에 로그인하여 백업 및 복원을 위한 MinIO 버킷을 생성합니다.

13 버킷 생성 아이콘을 클릭합니다.

14 버킷 이름(예: `my-cluster-backups`)을 입력합니다.

15 버킷이 생성되었는지 확인합니다.

16 기본적으로 새 MinIO 버킷은 읽기 전용입니다. Velero 독립형 백업 및 복원을 위해서는 MinIO 버킷이 읽기-쓰기여야 합니다. 버킷을 읽기-쓰기로 설정하려면 버킷을 선택하고 말줄임표(점) 링크를 클릭합니다.

17 **정책 편집**을 선택합니다.

18 정책을 **읽기 및 쓰기**로 변경합니다.

19 **추가**를 클릭합니다.

20 대화상자를 닫으려면 X를 클릭합니다.

Velero CLI 설치

VM 클라이언트 또는 로컬 시스템에 Velero CLI를 설치합니다.

이 설명서에 사용된 버전은 *Tanzu Kubernetes Grid 2.2.0용 Velero 1.9.7*입니다.

1 VMware Customer Connect 포털의 Tanzu Kubernetes Grid 제품 다운로드 페이지에서 Velero를 다운로드합니다.

> **참고** VMware의 지원을 받으려면 VMware에서 서명한 Velero 바이너리를 사용해야 합니다.

2 명령줄을 열고 디렉토리를 Velero CLI 다운로드로 변경합니다.

3 다운로드 파일의 압축을 풉니다. 예:

```
gunzip velero-linux-vX.X.X_vmware.1.gz
```

4 Velero 바이너리를 확인합니다.

```
ls -l
```

5 Velero CLI에 실행 권한을 부여합니다.

```
chmod +x velero-linux-vX.X.X_vmware.1
```

6 Velero CLI를 시스템 경로로 이동하여 전체적으로 사용할 수 있도록 합니다.

```
cp velero-linux-vX.X.X_vmware.1 /usr/local/bin/velero
```

7 설치를 확인합니다.

```
velero version
```

Tanzu Kubernetes 클러스터에 Velero 및 Restic 설치

Velero CLI 컨텍스트는 kubectl 컨텍스트를 자동으로 따릅니다. Velero CLI 명령을 실행하여 대상 클러스터에 Velero 및 Restic을 설치하기 전에 kubectl 컨텍스트를 설정합니다.

1 MinIO 버킷의 이름을 검색합니다. 예: `my-cluster-backups`

2 MinIO 버킷에 대한 AccessKey 및 SecretKey를 가져옵니다.

3 Velero CLI가 작동할 클러스터를 알 수 있도록 대상 Kubernetes 클러스터에 대한 컨텍스트를 설정합니다.

```
kubectl config use-context tkgs-cluster-name
```

4 `credentials-minio`라는 암호 파일을 생성합니다. 이 파일을 수집한 MinIO 서버 액세스 자격 증명으로 업데이트합니다. 예:

```
aws_access_key_id = 0XXNO8JCCGV41QZBV0RQ
aws_secret_access_key = clZ1bf8Ljkvkmq7fHucrKCkxV39BRbcycGeXQDfx
```

참고 "NoCredentialProviders: 체인에 올바른 제공자가 없습니다."라는 설명과 함께 "백업 저장소를 가져오는 동안 오류가 발생했습니다." 오류 메시지가 표시되면 자격 증명 파일의 시작 부분에 `[default]` 줄을 추가합니다. 예:

```
[default]
aws_access_key_id = 0XXNO8JCCGV41QZBV0RQ
aws_secret_access_key = clZ1bf8Ljkvkmq7fHucrKCkxV39BRbcycGeXQDfx
```

5 파일을 저장하고 파일이 제자리에 있는지 확인합니다.

```
ls
```

6 다음 명령을 실행하여 대상 Kubernetes 클러스터에 Velero 및 Restic을 설치합니다. 두 URL을 모두 MinIO 인스턴스의 URL로 바꿉니다.

```
velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.0.0 \
--bucket tkgs-velero \
--secret-file ./credentials-minio \
--use-volume-snapshots=false \
```

```
--use-restic \
--backup-location-config \
region=minio,s3ForcePathStyle="true",s3Url=http://10.199.17.63:9000,publicUrl=http://
10.199.17.63:9000
```

7 Velero 및 Restic의 설치를 확인합니다.

```
kubectl logs deployment/velero -n velero
```

8 `velero` 네임스페이스를 확인합니다.

```
kubectl get ns
```

9 `velero` 및 `restic` 포드를 확인합니다.

```
kubectl get all -n velero
```

Restic DaemonSet 문제 해결(필요한 경우)

Kubernetes 클러스터에서 3-포드 Restic DaemonSet을 실행하려면 Restic DaemonSet 규격을 업데이트하고 hostPath를 수정해야 할 수 있습니다. 이 문제에 대한 자세한 내용은 Velero 설명서에서 Restic 통합을 참조하십시오.

1 3-포드 Restic DaemonSet을 확인합니다.

```
kubectl get pod -n velero
```

포드가 CrashLoopBackOff 상태인 경우 다음과 같이 편집합니다.

2 `edit` 명령을 실행합니다.

```
kubectl edit daemonset restic -n velero
```

3 hostPath를 `/var/lib/kubelet/pods`에서 `/var/vcap/data/kubelet/pods`로 변경합니다.

```
- hostPath:
    path: /var/vcap/data/kubelet/pods
```

4 파일을 저장합니다.

5 3-포드 Restic DaemonSet을 확인합니다.

```
kubectl get pod -n velero
```

```
NAME                      READY   STATUS    RESTARTS   AGE
restic-5jln8              1/1     Running   0          73s
restic-bpvtq              1/1     Running   0          73s
restic-vg8j7              1/1     Running   0          73s
velero-72c84322d9-le7bd   1/1     Running   0          10m
```

Velero 메모리 제한 조정(필요한 경우)

Velero 백업이 여러 시간 동안 `status=InProgress`를 반환하는 경우 제한 및 요청 메모리 설정을 늘립니다.

1 다음 명령을 실행합니다.

```
kubectl edit deployment/velero -n velero
```

2 제한 및 요청 메모리 설정을 기본값인 `256Mi` 및 `128Mi`에서 `512Mi` 및 `256Mi`로 변경합니다.

```
ports:
- containerPort: 8085
  name: metrics
  protocol: TCP
resources:
  limits:
    cpu: "1"
    memory: 512Mi
  requests:
    cpu: 500m
    memory: 256Mi
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
```

독립형 Velero 및 Restic을 사용하여 클러스터 워크로드 백업 및 복원

독립형 Velero 및 Restic을 사용하여 TKG 클러스터에서 실행되는 워크로드를 백업 및 복원할 수 있습니다. 이 방법은 vSphere용 Velero 플러그인 사용에 대한 대안입니다. 독립형 Velero를 사용하는 주된 이유는 이식성이 필요한 경우입니다. 상태 저장 워크로드에는 Restic이 필요합니다.

사전 요구 사항

독립형 Velero 및 Restic을 사용하여 TKG 클러스터에서 워크로드를 백업하고 복원하려면 대상 클러스터에 독립형 Velero 및 Restic 버전을 설치해야 합니다. 별도의 대상 클러스터에서 복원을 수행하려면 Velero와 Restic도 대상 클러스터에 설치해야 합니다. TKG 클러스터에 독립형 Velero 및 Restic 설치 및 구성의 내용을 참조하십시오.

TKG 클러스터에서 실행되는 상태 비저장 애플리케이션 백업

TKG 클러스터에서 실행되는 상태 비저장 애플리케이션을 백업하려면 Velero를 사용해야 합니다.

이 예에서는 `--include namespaces` 태그(모든 애플리케이션 구성 요소가 해당 네임스페이스에 있는 경우)를 사용하여 예제 상태 비저장 애플리케이션을 백업하고 복원하는 방법을 보여줍니다.

```
velero backup create example-backup --include-namespaces example-backup
```

다음이 표시됩니다.

```
Backup request "example-backup" submitted successfully.
Run `velero backup describe example-backup` or `velero backup logs example-backup` for more
details.
```

생성된 백업을 확인합니다.

```
velero backup get
```

```
velero backup describe example-backup
```

S3 호환 개체 저장소(예: MinIO 서버)에서 Velero 버킷을 확인합니다.

Velero는 Kubernetes CRD(사용자 지정 리소스 정의)에 일부 메타데이터를 씁니다.

```
kubectl get crd
```

Velero CRD를 사용하면 다음과 같은 특정 명령을 실행할 수 있습니다.

```
kubectl get backups.velero.io -n velero
```

```
kubectl describe backups.velero.io guestbook-backup -n velero
```

TKG 클러스터에서 실행되는 상태 비저장 애플리케이션 복원

TKG 클러스터에서 실행되는 상태 비저장 애플리케이션을 복원하려면 Velero를 사용해야 합니다.

예제 애플리케이션의 복원을 테스트하려면 예제 애플리케이션을 삭제합니다.

네임스페이스를 삭제합니다.

```
kubectl delete ns guestbook
namespace "guestbook" deleted
```

애플리케이션을 복원합니다.

```
velero restore create --from-backup example-backup
```

다음이 표시됩니다.

```
Restore request "example-backup-20200721145620" submitted successfully.
Run `velero restore describe example-backup-20200721145620` or `velero restore logs example-
backup-20200721145620` for more details.
```

애플리케이션이 복원되었는지 확인합니다.

```
velero restore describe example-backup-20200721145620
```

다음 명령을 실행하여 확인합니다.

```
velero restore get
```

```
kubectl get ns
```

```
kubectl get pod -n example
```

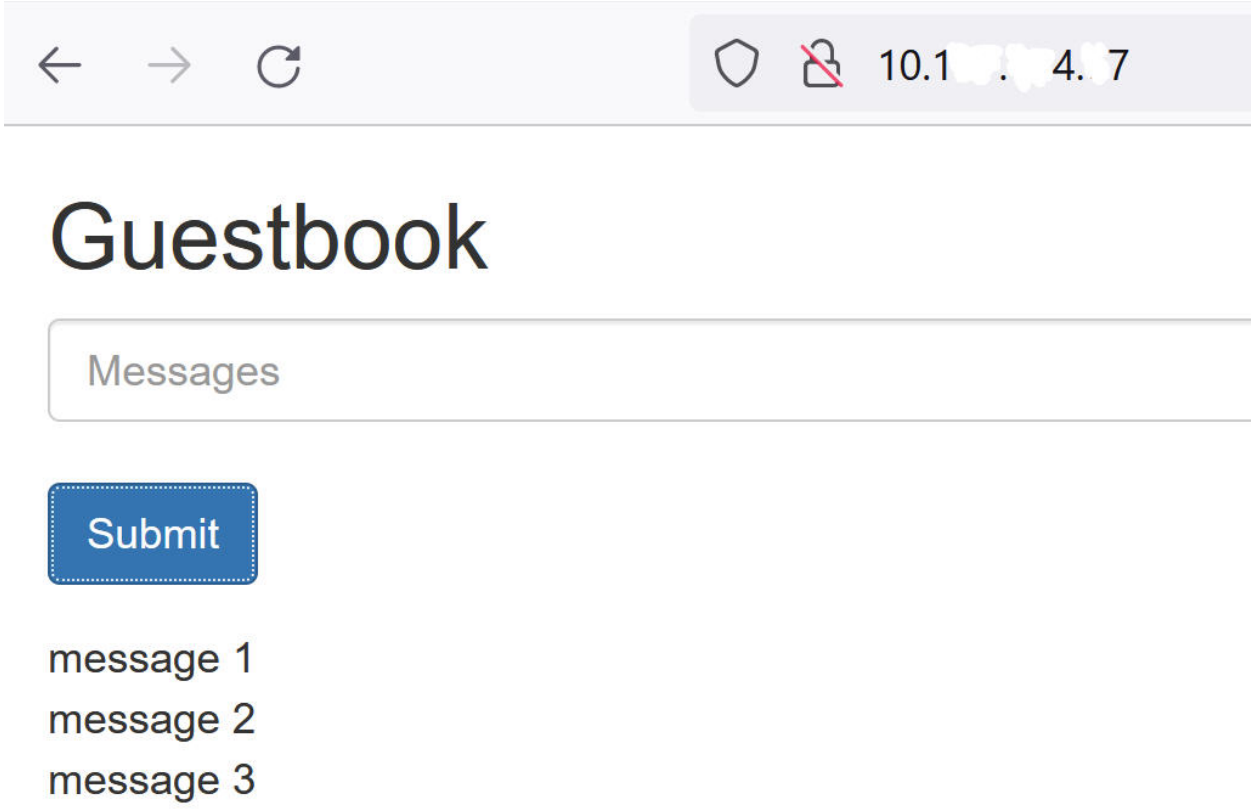
```
kubectl get svc -n example
```

TKG 클러스터에서 실행되는 상태 저장 애플리케이션 백업

TKG 클러스터에서 실행되는 상태 저장 애플리케이션을 백업하려면 애플리케이션 메타데이터 및 영구 볼륨에 저장된 애플리케이션 데이터를 모두 백업해야 합니다. 이렇게 하려면 Velero와 Restic이 모두 필요합니다.

이 예에서는 방명록 애플리케이션을 사용합니다. TKG 클러스터에 방명록 애플리케이션을 배포했다고 가정합니다. TKG 클러스터에 방명록 애플리케이션 배포의 내용을 참조하십시오.

상태 저장 백업 및 복원을 시연할 수 있도록 프런트 엔드 웹 페이지를 사용하여 방명록 애플리케이션에 메시지를 제출하여 메시지가 지속되도록 합니다. 예:



이 예에서는 `--include namespace` 태그와 포드 주석을 사용하여 방명록 애플리케이션을 백업하고 복원하는 방법을 보여줍니다.

참고 이 예에서는 주석을 사용합니다. 하지만 Velero 버전 1.5 이상에서는 더 이상 주석이 필요하지 않습니다. 주석을 사용하지 않으려면 백업을 생성할 때 `--default-volumes-to-restic` 옵션을 사용하면 됩니다. 그러면 Restic을 사용하여 모든 PV가 자동으로 백업됩니다. 자세한 내용은 https://velero.io/docs/v1.5/restic/의 내용을 참조하십시오.

백업 절차를 시작하려면 포드의 이름을 가져옵니다.

```
kubectl get pod -n guestbook
```

예:

```
kubectl get pod -n guestbook

NAME                                          READY   STATUS    RESTARTS   AGE
guestbook-frontend-deployment-85595f5bf9-h8cff  1/1     Running             0          55m
guestbook-frontend-deployment-85595f5bf9-lw6tg  1/1     Running             0          55m
guestbook-frontend-deployment-85595f5bf9-wpqc8  1/1     Running             0          55m
redis-leader-deployment-64fb8775bf-kbs6s        1/1     Running             0          55m
redis-follower-deployment-84cd76b975-jrn8v      1/1     Running             0          55m
redis-follower-deployment-69df9b5688-zml4f      1/1     Running             0          55m
```

영구 볼륨은 Redis 포드에 연결됩니다. Restic을 사용하여 이러한 상태 저장 포드를 백업하기 때문에 `volumeMount`라는 이름으로 상태 저장 포드에 주석을 추가해야 합니다.

상태 저장 포드에 주석을 추가하려면 `volumeMount`를 알고 있어야 합니다. `mountName`을 가져오려면 다음 명령을 실행합니다.

```
kubectl describe pod redis-leader-deployment-64fb8775bf-kbs6s -n guestbook
```

결과에 `redis-leader-data`의 `Containers.leader.Mounts: /data`가 표시됩니다. 이 마지막 토큰이 리더 포드 주석에 사용할 `volumeMount` 이름입니다. 팔로워의 경우 `redis-follower-data`입니다. 소스 YAML에서 `volumeMount` 이름을 가져올 수도 있습니다.

각 Redis 포드에 주석을 추가합니다. 예를 들면 다음과 같습니다.

```
kubectl -n guestbook annotate pod redis-leader-64fb8775bf-kbs6s backup.velero.io/backup-
volumes=redis-leader-data
```

다음이 표시됩니다.

```
pod/redis-leader-64fb8775bf-kbs6s annotated
```

주석을 확인합니다.

```
kubectl -n guestbook describe pod redis-leader-64fb8775bf-kbs6s | grep Annotations
Annotations:  backup.velero.io/backup-volumes: redis-leader-data
```

```
kubectl -n guestbook describe pod redis-follower-779b6d8f79-5dphr | grep Annotations
Annotations:  backup.velero.io/backup-volumes: redis-follower-data
```

Velero 백업을 수행합니다.

```
velero backup create guestbook-backup --include-namespaces guestbook
```

다음이 표시됩니다.

```
Backup request "guestbook-backup" submitted successfully.
Run `velero backup describe guestbook-pv-backup` or `velero backup logs guestbook-pv-backup`
for more details.
```

생성된 백업을 확인합니다.

```
velero backup get

NAME                   STATUS      ERRORS   WARNINGS   CREATED
EXPIRES   STORAGE LOCATION   SELECTOR
guestbook-backup       Completed   0        0          2020-07-23 16:13:46 -0700 PDT
29d       default            <none>
```

백업 세부 정보를 확인합니다.

```
velero backup describe guestbook-backup --details
```

Velero를 사용하면 다음과 같은 다른 명령을 실행할 수 있습니다.

```
kubectl get backups.velero.io -n velero

NAME               AGE
guestbook-backup   4m58s
```

그리고:

```
kubectl describe backups.velero.io guestbook-backup -n velero
```

TKG 2.0 클러스터에서 실행되는 상태 저장 애플리케이션 복원

TKG 클러스터에서 실행되는 상태 저장 애플리케이션을 복원하려면 애플리케이션 메타데이터 및 영구 볼륨에 저장된 애플리케이션 데이터를 모두 복원해야 합니다. 이렇게 하려면 Velero와 Restic이 모두 필요합니다.

이 예에서는 이전 섹션에서 설명한 대로 상태 저장 방명록 애플리케이션을 백업했다고 가정합니다.

상태 저장 애플리케이션의 복원을 테스트하려면 해당 네임스페이스를 삭제합니다.

```
kubectl delete ns guestbook
namespace "guestbook" deleted
```

애플리케이션 삭제를 확인합니다.

```
kubectl get ns
kubectl get pvc,pv --all-namespaces
```

백업에서 애플리케이션을 복원하려면 다음 명령 구문을 사용합니다.

```
velero restore create --from-backup <velero-backup-name>
```

예:

```
velero restore create --from-backup guestbook-backup
```

다음과 유사한 메시지가 표시됩니다.

```
Restore request "guestbook-backup-20200723161841" submitted successfully.
Run `velero restore describe guestbook-backup-20200723161841` or `velero restore logs
guestbook-backup-20200723161841` for more details.
```

상태 저장 방명록 애플리케이션이 복원되었는지 확인합니다.

```
velero restore describe guestbook-backup-20200723161841

Name:         guestbook-backup-20200723161841
Namespace:    velero
Labels:       <none>
Annotations:  <none>

Phase:  Completed

Backup:  guestbook-backup

Namespaces:
  Included:  all namespaces found in the backup
  Excluded:  <none>

Resources:
  Included:        *
  Excluded:        nodes, events, events.events.k8s.io, backups.velero.io,
restores.velero.io, resticrepositories.velero.io
  Cluster-scoped:  auto

Namespace mappings:  <none>

Label selector:  <none>
```

```
Restore PVs:  auto

Restic Restores (specify --details for more information):
  Completed:  3
```

다음 추가 명령을 실행하여 복원을 확인합니다.

```
velero restore get

NAME                                BACKUP                STATUS      ERRORS   WARNINGS
CREATED                         SELECTOR
guestbook-backup-20200723161841      guestbook-backup      Completed   0        0
2021-08-11 16:18:41 -0700 PDT   <none>
```

네임스페이스가 복원되었는지 확인합니다.

```
kubectl get ns

NAME              STATUS   AGE
default           Active   16d
guestbook         Active   76s
...
velero            Active   2d2h
```

애플리케이션이 복원되었는지 확인합니다.

```
vkubectl get all -n guestbook

NAME                                READY   STATUS    RESTARTS   AGE
pod/frontend-6cb7f8bd65-h2pnb       1/1     Running   0          6m27s
pod/frontend-6cb7f8bd65-kwlpr       1/1     Running   0          6m27s
pod/frontend-6cb7f8bd65-snwl4       1/1     Running   0          6m27s
pod/redis-leader-64fb8775bf-kbs6s   1/1     Running   0          6m28s
pod/redis-follower-779b6d8f79-5dphr 1/1     Running   0          6m28s
pod/redis-follower-899c7e2z65-8apnk 1/1     Running   0          6m28s

NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)        AGE
service/guestbook-frontend           LoadBalancer   10.10.89.59      10.19.15.99
80:31513/TCP   65s
service/redis-follower               ClusterIP      10.111.163.189   <none>
6379/TCP       65s
service/redis-leader                 ClusterIP      10.111.70.189    <none>
6379/TCP       65s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/guestbook-frontend-deployment   3/3     3            3           65s
deployment.apps/redis-follower-deployment       1/2     2            1           65s
deployment.apps/redis-leader-deployment         1/1     1            1           65s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/guestbook-frontend-deployment-56fc5b6b47   3         3         3       65s
replicaset.apps/redis-follower-deployment-6fc9cf5759       2         2         1       65s
replicaset.apps/redis-leader-deployment-7d89bbdbcf         1         1         1       65s
```

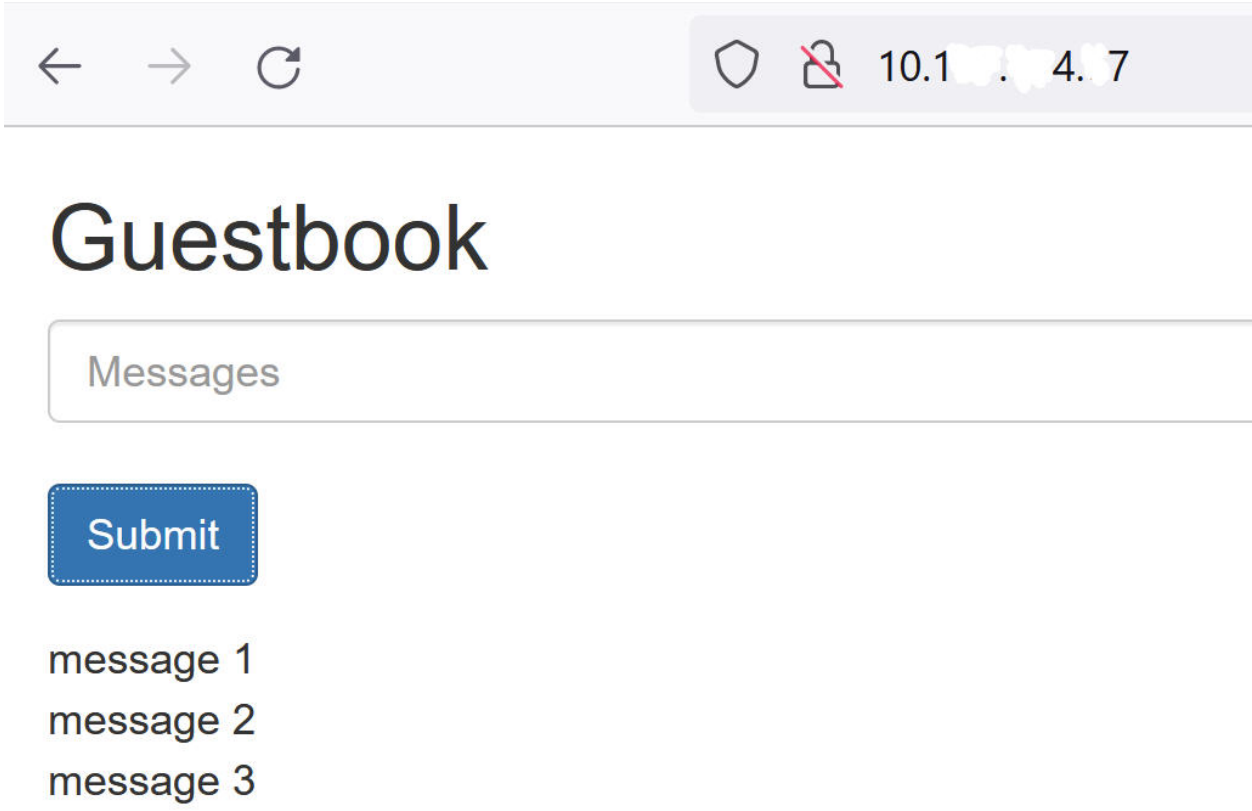
영구 볼륨이 복원되었는지 확인합니다.

```
kubectl get pvc,pv -n guestbook

NAME                                        STATUS
VOLUME                                      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/redis-leader-claim    Bound    pvc-a2f6e6d4-42db-4fb8-
a198-5379a2552509   2Gi        RWO            thin-disk      2m40s
persistentvolumeclaim/redis-follower-claim Bound    pvc-55591938-921f-452a-
b418-2cc680c0560b   2Gi        RWO            thin-disk      2m40s

NAME                                                        CAPACITY   ACCESS MODES   RECLAIM
POLICY   STATUS   CLAIM                                STORAGECLASS   REASON   AGE
persistentvolume/pvc-55591938-921f-452a-b418-2cc680c0560b   2Gi        RWO
Delete           Bound    guestbook/redis-follower-claim    thin-disk               2m40s
persistentvolume/pvc-a2f6e6d4-42db-4fb8-a198-5379a2552509   2Gi        RWO
Delete           Bound    guestbook/redis-leader-claim      thin-disk               2m40s
```

마지막으로 방명록 프런트 엔드 서비스의 외부 IP를 사용하여 방명록 프런트 엔드에 액세스하고 자습서 시작 부분에서 제출한 메시지가 복원되었는지 확인합니다. 예:



CSI 스냅샷과 함께 Velero를 사용하여 백업 및 복원

CSI 스냅샷과 함께 Velero를 사용하여 감독자에 프로비저닝된 TKG 클러스터에서 실행되는 워크로드에 대해 CSI 생성 영구 볼륨을 백업하고 복원할 수 있습니다.

요구 사항

다음 요구 사항을 준수합니다.

- vSphere 8.0 U2 이상
- vSphere 8.x 이상용 Tanzu Kubernetes 릴리스 v1.26.5
- 볼륨 스냅샷을 지원하는 CSI 드라이버를 사용하여 생성된 영구 볼륨

주의 CSI 스냅샷과 함께 Velero를 사용하는 것은 볼륨 스냅샷을 지원하는 CSI 드라이버를 사용하여 생성된 영구 볼륨에만 가능합니다. 자세한 내용은 "vSphere IaaS 제어부에서 TKG 서비스 사용" 에서 TKG 클러스터에서 스냅샷 생성을 참조하십시오.

절차

CSI(Container Storage Interface) 스냅샷과 함께 Velero를 사용하여 TKGS 클러스터에서 실행되는 워크로드를 백업 및 복원할 수 있습니다. Velero node-agent는 CSI 스냅샷 데이터 이동을 사용하여 백업 및 복원의 구체적인 작업을 완료하기 위해 모듈을 호스팅하는 DaemonSet입니다. 자세한 내용은 Velero의 Container Storage Interface 스냅샷 지원을 참조하십시오.

1 S3 호환 스토리지 위치(예: MinIO 또는 AWS S3 버킷)를 생성합니다.

  아래 예에서는 AWS S3 버킷을 사용합니다.

  MinIO를 사용하려면 MinIO 개체 저장소 설치 및 구성 항목을 참조하십시오.

2 kubectl을 실행 중인 클러스터 클라이언트에 Velero CLI를 설치합니다.

  https://github.com/vmware-tanzu/velero/releases에서 다운로드합니다.

  다음 링크 중 하나에서 설치 지침을 참조하십시오.

  - 1단계: Linux Workstation에서 Velero CLI 설치
  - Velero CLI 설치
  - https://velero.io/docs/v1.12/basic-install/#install-the-cli

3 Velero 백업을 실행하려는 TKG 서비스 클러스터에 연결합니다.

  kubectl을 사용하여 vCenter Single Sign-On 사용자로 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

4 Velero 설치 명령을 실행합니다(예: AWS S3 스토리지 및 해당 자격 증명 파일 사용).

```
velero install \
    --provider aws \
    --plugins velero/velero-plugin-for-aws:v1.9.0,velero/velero-plugin-for-csi:v0.7.0 \
    --bucket velero-cpe-backup-bucket \
    --secret-file ./cloud-credential \
    --use-volume-snapshots=true \
    --features=EnableCSI --use-node-agent
```

참고 Velero v1.14 릴리스부터 Velero CSI 플러그인이 Velero와 병합됩니다. 따라서 Velero v1.14 이상을 설치하는 경우 Velero CSI 플러그인을 설치할 필요가 없습니다. 설치하면 Velero 포드가 시작되지 않습니다.

TKG 서비스 클러스터 문제 해결

21

클러스터 노드에 연결, 클러스터 리소스 계층 보기 및 로그 파일 수집을 비롯한 다양한 방법을 사용하여 TKG 서비스 클러스터 문제를 해결할 수 있습니다.

다음으로 아래 항목을 읽으십시오.

- 감독자의 TKG 클러스터 문제 해결을 위해 로그 끌어오기
- 감독자에서 TKG 구성 요소의 상태 확인
- TKG 클러스터 연결 문제 해결 및 로그온 오류
- 컨텐츠 라이브러리 오류 문제 해결
- VM 클래스 오류 문제 해결
- TKGS 클러스터 프로비저닝 오류 문제 해결
- TKG 서비스 클러스터 노드 오류 문제 해결
- TKG 서비스 클러스터 네트워킹 오류 문제 해결
- 실패한 TKG 클러스터 업그레이드 다시 시작
- 컨테이너 배포 오류 문제 해결
- 컨테이너 레지스트리 오류 문제 해결
- 추가적인 신뢰할 수 있는 CA 오류 문제 해결

감독자의 TKG 클러스터 문제 해결을 위해 로그 끌어오기

감독자의 TKG 클러스터 문제를 해결하기 위해 감독자 지원 번들, 워크로드 관리 로그 및 CAPI, CAPV, VM 운영자, TKG 컨트롤러 관리자 로그를 비롯한 다양한 로그를 끌어오려면 이 항목을 참조하십시오.

감독자에 대한 지원 번들 수집

TKG 클러스터 오류를 해결하기 위해 감독자 로그를 내보낼 수 있습니다. 일반적으로 이러한 로그 검토는 VMware 지원팀과 협의하여 수행됩니다.

1 vSphere Client를 사용하여 vSphere IaaS control plane 환경에 로그인합니다.

2 **메뉴 > 워크로드 관리**를 선택합니다.

3 **감독자** 탭을 선택합니다.

4 대상 **감독자** 인스턴스를 선택합니다.

5 **로그 내보내기**를 선택합니다.

지원 번들을 수집한 후에는 KB 문서: 보안 FTP 포털을 통해 VMware에 대한 진단 정보 업로드(http://kb.vmware.com/kb/2069559)를 참조하십시오. vSpehre with Tanzu에 대한 로그 수집도 참조하십시오.

TKG 클러스터에 대한 지원 번들 수집

TKC 지원 번들러 유틸리티를 사용하여 TKG 클러스터 로그 파일을 수집하고 문제를 해결할 수 있습니다.

TKC 지원 번들 유틸리티를 가져오고 사용하려면 VMware 지원 기술 자료 문서 vSpehre with Tanzu에 대한 로그 수집을 참조하십시오.

워크로드 관리 로그 파일에 tail 명령 사용

WCP(워크로드 제어부) 로그 파일에 tail 명령을 사용하면 감독자 및 TKG 클러스터 오류를 해결하는 데 도움이 될 수 있습니다.

1 vCenter Server Appliance에 대한 SSH 연결을 설정합니다.

2 `root` 사용자로 로그인합니다.

3 `shell` 명령을 실행합니다.

다음 내용이 보입니다

```
Shell access is granted to root
root@localhost [ ~ ]#
```

4 다음 명령을 실행하여 WCP 로그 파일의 마지막 부분을 출력니다.

```
tail -f /var/log/vmware/wcp/wcpsvc.log
```

감독자에서 TKG별 로그 수집

감독자는 TKG 2.0에 인프라를 제공하는 여러 Kubernetes 포드를 실행합니다.

```
kubectl -n vmware-system-capw get deployments.apps
NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
capi-controller-manager                         2/2     2            2           18h
capi-kubeadm-bootstrap-controller-manager       2/2     2            2           18h
capi-kubeadm-control-plane-controller-manager   2/2     2            2           18h
capv-controller-manager                         2/2     2            2           10h
capw-controller-manager                         2/2     2            2           18h
capw-webhook                                    2/2     2            2           18h
```

인프라 포드는 복제본을 실행하는 배포입니다. 어떤 복제본이 리더인지 확인하고 리더의 최신 로그를 확인해야 합니다. 리더가 아니면 일반적으로 리스 획득 시도에 대한 로그를 기록한 후 중지합니다.

감독자에 로그인하고 감독자 vSphere 네임스페이스를 사용하여 이러한 포드를 확인해야 합니다.

레이블 선택기를 사용하는 로그는 작동하지 않으므로 포드 이름 끝에 추가되는 임의의 문자열을 구체화해야 할 수 있습니다. 출력을 grep 'error' 또는 grep -i 'error'로 파이핑하여 시작하는 것이 유용한 경우도 있습니다. 예: `kubectl logs <args> | grep error`.

CAPI 로그

클러스터 API 제공자:

```
kubectl logs -n vmware-system-capw -c manager vmware-system-capw-capi-controller-manager-
<id>
```

CAPV 로그

vSphere 제공자에 대한 클러스터 API:

```
kubectl logs -n vmware-system-capv -c manager vmware-system-capw-v1alpha3-vmware-system-
capv-v1alpha3-controller-manager-<id>
```

VM 운영자 로그

VM Operator:

```
kubectl logs -n vmware-system-vmop -c manager vmware-system-vmop-controller-manager-<id>
```

TKG 컨트롤러 관리자 로그

GCM 컨트롤러 관리자

```
kubectl logs -n vmware-system-tkg -c manager vmware-system-tkg-controller-manager-<id>
```

감독자에서 TKG 구성 요소의 상태 확인

이 항목에서는 TKG 구성 요소와 관련하여 감독자의 상태를 확인하는 다양한 기법을 참조할 수 있습니다.

감독자 포드의 상태 확인

감독자 포드는 TKG 인프라 구성 요소를 실행합니다.

감독자의 모든 포드가 [실행 중] 상태인지 확인하십시오.

```
kubectl get pods -A | grep "Running"
```

참고 `grep -v "Running"`을 사용하여 [실행 중]이 아닌 포드를 반환할 수도 있습니다.

예:

```
NAMESPACE
NAME                                                              READY   STATUS
RESTARTS        AGE
```

```
kube-system
coredns-855c5b4cfd-8w4hp                                        1/1     Running
0              27d
kube-system                                 coredns-855c5b4cfd-
bx2hk                                          1/1     Running     0              27d
kube-system                                 coredns-855c5b4cfd-
rrb5n                                          1/1     Running     0              27d
kube-system                                 docker-
registry-423f01b9b30c727e9c237a0031999b14              1/1     Running
0              27d
kube-system                                 docker-
registry-423f568f75dcb48725b0d768b7e4bdf5              1/1     Running
0              27d
kube-system                                 docker-
registry-423f930ca2413d96beef34526c2e61b4              1/1     Running
0              27d
kube-system
etcd-423f01b9b30c727e9c237a0031999b14                           1/1     Running     1 (27d
ago)     27d
kube-system
etcd-423f568f75dcb48725b0d768b7e4bdf5                           1/1     Running     1 (27d
ago)     27d
kube-system
etcd-423f930ca2413d96beef34526c2e61b4                           1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-
apiserver-423f01b9b30c727e9c237a0031999b14                1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-
apiserver-423f568f75dcb48725b0d768b7e4bdf5                1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-
apiserver-423f930ca2413d96beef34526c2e61b4                1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-controller-
manager-423f01b9b30c727e9c237a0031999b14        1/1     Running     0              27d
kube-system                                 kube-controller-
manager-423f568f75dcb48725b0d768b7e4bdf5        1/1     Running     0              27d
kube-system                                 kube-controller-
manager-423f930ca2413d96beef34526c2e61b4        1/1     Running     0              27d
kube-system                                 kube-
proxy-8h499                                               1/1     Running
0              27d
kube-system                                 kube-proxy-
bm7qt                                                  1/1     Running     0              27d
kube-system                                 kube-proxy-
dnmq2                                                  1/1     Running     0              27d
kube-system                                 kube-
scheduler-423f01b9b30c727e9c237a0031999b14                2/2     Running     13 (25d
ago)    27d
kube-system                                 kube-
scheduler-423f568f75dcb48725b0d768b7e4bdf5                2/2     Running
0              27d
kube-system                                 kube-
scheduler-423f930ca2413d96beef34526c2e61b4                2/2     Running
```

```
0              27d
kube-system                                 kubectl-plugin-
vsphere-423f01b9b30c727e9c237a0031999b14          1/1     Running     3 (27d ago)     27d
kube-system                                 kubectl-plugin-
vsphere-423f568f75dcb48725b0d768b7e4bdf5          1/1     Running     3 (27d ago)     27d
kube-system                                 kubectl-plugin-
vsphere-423f930ca2413d96beef34526c2e61b4          1/1     Running     3 (27d ago)     27d
kube-system                                 wcp-
authproxy-423f01b9b30c727e9c237a0031999b14                  1/1     Running
0              27d
kube-system                                 wcp-
authproxy-423f568f75dcb48725b0d768b7e4bdf5                  1/1     Running
0              27d
kube-system                                 wcp-
authproxy-423f930ca2413d96beef34526c2e61b4                  1/1     Running
0              27d
kube-system                                 wcp-
fip-423f01b9b30c727e9c237a0031999b14                        1/1     Running
0              27d
kube-system                                 wcp-
fip-423f568f75dcb48725b0d768b7e4bdf5                        1/1     Running
0              27d
kube-system                                 wcp-
fip-423f930ca2413d96beef34526c2e61b4                        1/1     Running
0              27d
svc-tmc-c63                                 agent-updater-69f6598bcd-
zrkwq                                    1/1     Running     0              27d
svc-tmc-c63                                 agentupdater-workload-27696934--1-
vz5sg                           0/1     Completed   0              35s
svc-tmc-c63                                 cluster-health-
extension-68948f657-4gpcd                         1/1     Running     0              27d
svc-tmc-c63                                 extension-manager-f8886bfb7-
vdsm9                                  1/1     Running     0              27d
svc-tmc-c63                                 extension-updater-79b4787cf6-
bwssn                                  1/1     Running     0              27d
svc-tmc-c63                                 intent-agent-66576db5bd-
lj2gk                                     1/1     Running     0              5d6h
svc-tmc-c63                                 sync-agent-
f9c68cc58-6zddj                                     1/1     Running     0              6d
svc-tmc-c63                                 tmc-agent-installer-27696934--1-
jgwvw                             0/1     Completed   0              35s
svc-tmc-c63                                 tmc-auto-attach-6488b9cd8b-
xdfzz                                   1/1     Running     0              18h
svc-tmc-c63                                 vsphere-resource-
retriever-58985c99cb-68h6v                       1/1     Running     0              18h
vmware-system-appplatform-operator-system   vmware-system-appplatform-operator-
mgr-0                          1/1     Running     0              27d
vmware-system-appplatform-operator-system   vmware-system-psp-operator-mgr-587f66646d-
xxvmr                  1/1     Running     0              27d
vmware-system-capw                          capi-controller-
manager-766c6fc449-4qqvf                        2/2     Running     423 (26d ago)   27d
vmware-system-capw                          capi-controller-manager-766c6fc449-
bcpdq                          2/2     Running     410 (26d ago)   27d
vmware-system-capw                          capi-controller-manager-766c6fc449-
rnznx                          2/2     Running     0              26d
```

```
vmware-system-capw                          capi-kubeadm-bootstrap-controller-
manager-58fd767b49-585f2        2/2     Running     402 (25d ago)   27d
vmware-system-capw                          capi-kubeadm-bootstrap-controller-
manager-58fd767b49-96q6m        2/2     Running     398 (25d ago)   27d
vmware-system-capw                          capi-kubeadm-bootstrap-controller-
manager-58fd767b49-nssgq        2/2     Running     407 (25d ago)   27d
vmware-system-capw                          capi-kubeadm-control-plane-controller-
manager-559df997b-762jr     2/2     Running     193 (26d ago)   27d
vmware-system-capw                          capi-kubeadm-control-plane-controller-
manager-559df997b-bb42s     2/2     Running     189 (26d ago)   27d
vmware-system-capw                          capi-kubeadm-control-plane-controller-
manager-559df997b-wxhqv     2/2     Running     199 (26d ago)   27d
vmware-system-capw                          capw-controller-
manager-6dd47d75b-6ncxk                           2/2     Running     400 (25d ago)   27d
vmware-system-capw                          capw-controller-manager-6dd47d75b-
k2ph4                           2/2     Running     399 (25d ago)   27d
vmware-system-capw                          capw-controller-manager-6dd47d75b-
np9sg                           2/2     Running     403 (25d ago)   27d
vmware-system-capw                          capw-
webhook-5484757c7-2pkbt                                       2/2     Running
0               27d
vmware-system-capw                          capw-webhook-5484757c7-
fkt7z                                     2/2     Running     0               27d
vmware-system-capw                          capw-webhook-5484757c7-
r85kw                                     2/2     Running     0               27d
vmware-system-cert-manager                  cert-manager-6ccbcfcd57-
lppgn                                   1/1     Running     1 (27d ago)     27d
vmware-system-cert-manager                  cert-manager-
cainjector-796f7b74db-5qvgn                         1/1     Running     3 (27d ago)     27d
vmware-system-cert-manager                  cert-manager-webhook-586948846f-
b584m                             1/1     Running     0               27d
vmware-system-csi                           vsphere-csi-
controller-6d8cfd75cd-66zbj                           6/6     Running     0               27d
vmware-system-csi                           vsphere-csi-controller-6d8cfd75cd-
b4nhz                           6/6     Running     1 (27d ago)     27d
vmware-system-csi                           vsphere-csi-controller-6d8cfd75cd-
v6hlf                           6/6     Running     0               27d
vmware-system-kubeimage                     image-controller-ff79fb5fc-
kd6ts                                 1/1     Running     0               27d
vmware-system-license-operator              vmware-system-license-operator-controller-
manager-7d555768bnxjb   1/1     Running     0               25d
vmware-system-license-operator              vmware-system-license-operator-controller-
manager-7d555768j2sb8   1/1     Running     0               25d
vmware-system-license-operator              vmware-system-license-operator-controller-
manager-7d555768w7v77   1/1     Running     0               25d
vmware-system-logging                       fluentbit-
p24gk                                               1/1     Running     0
27d
vmware-system-logging                       fluentbit-
rj2t8                                               1/1     Running     0
27d
vmware-system-logging                       fluentbit-
xx2lk                                               1/1     Running     0
27d
vmware-system-nsop                          vmware-system-nsop-controller-
```

```
manager-65b8445959-66msw            1/1     Running     0               27d
vmware-system-nsop                          vmware-system-nsop-controller-manager-65b8445959-
nm6xh            1/1     Running     0              27d
vmware-system-nsop                          vmware-system-nsop-controller-manager-65b8445959-
sv5w7            1/1     Running     0              27d
vmware-system-nsx                           nsx-ncp-6f989c9c67-
vb4x6                                          1/1     Running     5 (27d ago)     27d
vmware-system-registry                      vmware-registry-controller-
manager-7f49485b9-72kh7               2/2     Running     0               27d
vmware-system-tkg                           masterproxy-tkgs-
plugin-8npzx                                    1/1     Running     0               27d
vmware-system-tkg                           masterproxy-tkgs-plugin-
bjtsz                                      1/1     Running     0               27d
vmware-system-tkg                           masterproxy-tkgs-plugin-
v92gt                                      1/1     Running     0               27d
vmware-system-tkg                           tkgs-plugin-server-5fc4c985c7-
bz8jh                              1/1     Running     0               27d
vmware-system-tkg                           tkgs-plugin-server-5fc4c985c7-
r9wj5                              1/1     Running     0               27d
vmware-system-tkg                           tkgs-plugin-server-5fc4c985c7-
sdr55                              1/1     Running     0               27d
vmware-system-tkg                           vmware-system-tkg-controller-manager-7ffcc55df5-
dqkkm            2/2     Running     0              25d
vmware-system-tkg                           vmware-system-tkg-controller-manager-7ffcc55df5-
hkvx9            2/2     Running     0              25d
vmware-system-tkg                           vmware-system-tkg-controller-manager-7ffcc55df5-
txxrf            2/2     Running     0              25d
vmware-system-tkg                           vmware-system-tkg-state-
metrics-5bbb6d668c-7c5vt                  2/2     Running     238 (26d ago)   27d
vmware-system-tkg                           vmware-system-tkg-state-metrics-5bbb6d668c-
c87zs                  2/2     Running     237 (26d ago)   27d
vmware-system-tkg                           vmware-system-tkg-state-metrics-5bbb6d668c-
wc46p                  2/2     Running     237 (26d ago)   27d
vmware-system-tkg                           vmware-system-tkg-
webhook-567f9fd68c-425xs                       2/2     Running     0               25d
vmware-system-tkg                           vmware-system-tkg-
webhook-567f9fd68c-97d6z                       2/2     Running     0               25d
vmware-system-tkg                           vmware-system-tkg-webhook-567f9fd68c-
dnkgt                        2/2     Running     0              25d
vmware-system-ucs                           upgrade-compatibility-service-5745846d58-
tpk67                    1/1     Running     0              27d
vmware-system-ucs                           upgrade-compatibility-service-5745846d58-
twxkt                    1/1     Running     0              27d
vmware-system-ucs                           upgrade-compatibility-service-5745846d58-
wzl8x                    1/1     Running     0              27d
vmware-system-vmop                          vmware-system-vmop-controller-manager-
c8499b9df-5h6f9             2/2     Running     0              27d
vmware-system-vmop                          vmware-system-vmop-controller-manager-
c8499b9df-6wgr7             2/2     Running     0              27d
vmware-system-vmop                          vmware-system-vmop-controller-manager-c8499b9df-
tvbg6            2/2     Running     0              27d
vmware-system-vmop                          vmware-system-vmop-hostvalidator-8498cc5f4d-
vqhnk                1/1     Running     0              27d
```

감독자의 포드가 [실행 중] 상태가 아닌 경우 다음 명령을 사용하여 해당 포드를 검사합니다.

```
kubectl describe pod <POD Name> -n <Namespace>
```

감독자 리소스의 상태 확인

TKG 컨트롤러 리소스:

```
kubectl get tkc
```

클러스터 API 리소스(CAPI, CABPK, CAPW, CAPV):

```
kubectl get cluster-api
```

VM 운영자 리소스:

```
kubectl get virtualmachines,virtualmachineservices,virtualmachinesetresourcepolicies
```

VM 운영자 리소스(클러스터 범위 지정 및 컨텐츠 라이브러리에서 동기화됨):

```
kubectl get virtualmachineimages
```

스토리지 리소스:

```
kubectl get persistentvolumeclaims,cnsnodevmattachment,cnsvolumemetadatas
```

네트워킹 리소스(NSX 관련):

```
kubectl get service,lb,lbm,vnet,vnetif,nsxerrors,nsxnetworkinterfaces
```

모든 감독자 리소스를 가져와서 파일에 쓰기:

```
kubectl api-resources --namespaced -o name | paste -d',' -s | xargs kubectl get -n
<namespace> > resources_in_namespace.txt
```

Cluster API 배포가 있는지 확인

CAPI, CAPW, CAPV 배포가 있는지 확인합니다.

```
kubectl -n vmware-system-capw get deployments.apps
NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
capi-controller-manager                     2/2     2            2           18h
capi-kubeadm-bootstrap-controller-manager   2/2     2            2           18h
capi-kubeadm-control-plane-controller-manager   2/2     2            2           18h
capv-controller-manager                     2/2     2            2           10h
capw-controller-manager                     2/2     2            2           18h
capw-webhook                                2/2     2            2           18h
```

지원 번들 파일 확인

감독자에 대한 지원 번들 수집의 `commands/` 폴더에는 WCP 시작 프로세스 중에 발생한 작업에 대한 세부 정보를 제공하는 journalctl 로그가 있습니다.

`kubectl_describe_virtualmachine.txt`

`kubectl_describe_tanzukubernetescluster.txt`

`kubectl_describe_kubeadmconfig.txt`

`kubectl-describe-pod_kube-system.txt`

`kubectl-describe-pod_vmware-system-capw.txt`

`kubectl-describe-pod_vmware-system-tkg.txt`

`kubectl-describe-pod_vmware-system-ucs.txt`

`kubectl-describe-pod_vmware-system-vmop.txt`

`kubectl_describe_cluster_resource_virtualmachineimages.txt`

`docker_images.txt`

TKG 클러스터의 상태 확인

모든 클러스터 노드(VM)가 [준비] 상태인지 확인합니다.

```
kubectl get nodes -o wide
```

```
NAME                                                    STATUS   ROLES
AGE   VERSION            INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                 KERNEL-
VERSION   CONTAINER-RUNTIME
tkgs-cluster-13-control-plane-dpmjj                     Ready    control-plane,master
12d   v1.22.9+vmware.1   10.244.0.25   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-control-plane-nb5r6                     Ready    control-plane,master
12d   v1.22.9+vmware.1   10.244.0.18   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-control-plane-zpcgs                     Ready    control-plane,master
12d   v1.22.9+vmware.1   10.244.0.26   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a1-gq458-9d6458d6f-c7t8c    Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.24   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a1-gq458-9d6458d6f-slzvn    Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.19   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a1-gq458-9d6458d6f-vzrsd    Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.22   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a2-tw99z-7b547b7f85-k5h4s   Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.20   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a2-tw99z-7b547b7f85-lkmdx   Ready    <none>
```

```
12d   v1.22.9+vmware.1   10.244.0.21   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a2-tw99z-7b547b7f85-qwv98   Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.23   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
```

모든 포드가 [실행 중] 상태인지 확인합니다.

```
kubectl get pods -A
```

```
NAMESPACE
NAME                                                                READY
STATUS      RESTARTS      AGE
kube-system                     antrea-
agent-58hv7                                                  2/2     Running
0             12d
kube-system                     antrea-
agent-6x897                                                  2/2     Running
0             12d
kube-system                     antrea-
agent-7d99k                                                  2/2     Running
0             12d
kube-system                     antrea-agent-
b7vdv                                                   2/2     Running
0             12d
kube-system                     antrea-agent-
dhdlg                                                   2/2     Running
0             12d
kube-system                     antrea-agent-
mj4wx                                                   2/2     Running
0             12d
kube-system                     antrea-agent-
v7vtv                                                   2/2     Running
0             12d
kube-system                     antrea-agent-
x49gz                                                   2/2     Running     1 (12d
ago)   12d
kube-system                     antrea-agent-
z2gth                                                   2/2     Running
0             12d
kube-system                     antrea-controller-bb59f5fbf-
t6cm9                                       1/1     Running     0             12d
kube-system                     antrea-resource-
init-65b586c9db-2cbxx                                  1/1     Running     0
12d
kube-system
coredns-5f64c4fff8-2gsqn                                                1/1
Running     0             12d
kube-system                     coredns-5f64c4fff8-
hvkg9                                                1/1     Running     0             12d
kube-system                     docker-registry-tkgs-cluster-13-control-plane-
dpmjj                          1/1     Running     0             12d
kube-system                     docker-registry-tkgs-cluster-13-control-plane-
nb5r6                          1/1     Running     0             12d
```

```
kube-system                    docker-registry-tkgs-cluster-13-control-plane-
zpcgs                         1/1     Running     0             12d
kube-system                    docker-registry-tkgs-cluster-13-worker-nodepool-a1-
gq458-9d6458d6f-c7t8c    1/1     Running     0             12d
kube-system                    docker-registry-tkgs-cluster-13-worker-nodepool-a1-
gq458-9d6458d6f-slzvn    1/1     Running     0             12d
kube-system                    docker-registry-tkgs-cluster-13-worker-nodepool-a1-
gq458-9d6458d6f-vzrsd    1/1     Running     0             12d
kube-system                    docker-registry-tkgs-cluster-13-worker-nodepool-a2-
tw99z-7b547b7f85-k5h4s   1/1     Running     0             12d
kube-system                    docker-registry-tkgs-cluster-13-worker-nodepool-a2-
tw99z-7b547b7f85-lkmdx   1/1     Running     0             12d
kube-system                    docker-registry-tkgs-cluster-13-worker-nodepool-a2-
tw99z-7b547b7f85-qwv98   1/1     Running     0             12d
kube-system                    etcd-tkgs-cluster-13-control-plane-
dpmjj                                  1/1     Running     0             12d
kube-system                    etcd-tkgs-cluster-13-control-plane-
nb5r6                                  1/1     Running     0             12d
kube-system                    etcd-tkgs-cluster-13-control-plane-
zpcgs                                  1/1     Running     0             12d
kube-system                    kube-apiserver-tkgs-cluster-13-control-plane-
dpmjj                          1/1     Running     0             12d
kube-system                    kube-apiserver-tkgs-cluster-13-control-plane-
nb5r6                          1/1     Running     0             12d
kube-system                    kube-apiserver-tkgs-cluster-13-control-plane-
zpcgs                          1/1     Running     0             12d
kube-system                    kube-controller-manager-tkgs-cluster-13-control-plane-
dpmjj                 1/1     Running     0             12d
kube-system                    kube-controller-manager-tkgs-cluster-13-control-plane-
nb5r6                 1/1     Running     1 (12d ago)   12d
kube-system                    kube-controller-manager-tkgs-cluster-13-control-plane-
zpcgs                 1/1     Running     0             12d
kube-system                    kube-
proxy-4kp57                                                      1/1     Running
0             12d
kube-system                    kube-
proxy-5q8pw                                                      1/1     Running
0             12d
kube-system                    kube-
proxy-5th6p                                                      1/1     Running
0             12d
kube-system                    kube-
proxy-8m6mx                                                      1/1     Running
0             12d
kube-system                    kube-proxy-
dn5lp                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
qgmcg                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
vbq27                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
xhnws                                                       1/1     Running
```

```
0            12d
kube-system                    kube-proxy-
zgfvn                                                1/1     Running
0            12d
kube-system                    kube-scheduler-tkgs-cluster-13-control-plane-
dpmjj                          1/1     Running     0             12d
kube-system                    kube-scheduler-tkgs-cluster-13-control-plane-
nb5r6                          1/1     Running     1 (12d ago)   12d
kube-system                    kube-scheduler-tkgs-cluster-13-control-plane-
zpcgs                          1/1     Running     0             12d
kube-system                    metrics-server-774bc4dc99-
qp7tb                                    1/1     Running     0             12d
vmware-system-auth             guest-cluster-auth-
svc-6m6cd                                          1/1     Running     0             12d
vmware-system-auth             guest-cluster-auth-svc-
h44xf                                     1/1     Running     0             12d
vmware-system-auth             guest-cluster-auth-svc-
l968n                                     1/1     Running     0             12d
vmware-system-cloud-provider   guest-cluster-cloud-provider-5f87d5d7d8-
rmd78                               1/1     Running     1 (12d ago)   12d
vmware-system-csi              vsphere-csi-controller-7d858778bd-
h7zhg                                6/6     Running     4 (12d ago)   12d
vmware-system-csi              vsphere-csi-controller-7d858778bd-
rkl98                                6/6     Running     0             12d
vmware-system-csi              vsphere-csi-controller-7d858778bd-
snmk7                                6/6     Running     0             12d
vmware-system-csi              vsphere-csi-
node-22fnt                                           3/3     Running     1 (12d
ago)   12d
vmware-system-csi              vsphere-csi-
node-5jtbr                                           3/3     Running
0            12d
vmware-system-csi              vsphere-csi-
node-87lz6                                           3/3     Running
0            12d
vmware-system-csi              vsphere-csi-node-
gp9sf                                          3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
k2psv                                          3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
mg8bw                                          3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
pctmv                                          3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
sslrl                                          3/3     Running     1 (12d ago)
12d
vmware-system-csi              vsphere-csi-node-
zbqbq                                          3/3     Running     0
12d
```

TKG 클러스터 상태를 가져오고 describe 명령을 실행합니다.

```
kubectl get tkc <clustername>
```

```
kubectl describe tkc <clustername>
```

TKG 컨트롤러 관리자 상태 확인

TKG 컨트롤러 관리자 상태를 확인합니다.

```
kubectl get deployments -n vmware-system-tkg vmware-system-tkg-controller-manager -o yaml
```

VM 운영자 상태 확인

포드가 실행 중이어야 합니다.

```
kubectl get pods -n vmware-system-vmop
NAME                                                    READY   STATUS    RESTARTS   AGE
vmware-system-vmop-controller-manager-c8499b9df-5h6f9   2/2     Running   0          27d
vmware-system-vmop-controller-manager-c8499b9df-6wgr7   2/2     Running   0          27d
vmware-system-vmop-controller-manager-c8499b9df-tvbg6   2/2     Running   0          27d
vmware-system-vmop-hostvalidator-8498cc5f4d-vqhnk       1/1     Running   0          27d
```

VM 운영자는 `VirtualNetworkInterface`를 생성하고 해당 상태를 확인합니다. 노드 VM이 IP를 얻지 못하는 경우 이것이 가장 먼저 확인해야 할 영역입니다. 가상 시스템 생성이 이 단계를 통과했습니까?

또한 VM 운영자는 `VirtualMachineService`를 조정하고 해당 상태를 업데이트하는 작업을 담당합니다. 외부 IP로 TKG 클러스터 Kubernetes API에 액세스할 수 없는 경우 VM 운영자 로그를 확인하십시오.

예를 들어 VM 운영자 포드 중 하나를 선택하고 네임스페이스를 지정한 다음 관리자 컨테이너를 지정합니다. (`logs` 명령은 컨테이너용입니다. 컨트롤러 포드 내부에는 로그를 확인할 수 있는 관리자 컨테이너가 있습니다.)

```
kubectl logs -f vmware-system-vmop-controller-manager-c8499b9df-5h6f9 -n vmware-system-vmop
manager
```

TKG 클러스터 연결 문제 해결 및 로그온 오류

이 섹션을 사용하여 TKG 클러스터 연결 문제를 해결하고 오류를 로그온합니다.

사용 권한 부족 오류

vSphere 네임스페이스에 대한 충분한 사용 권한이 없으면 감독자 또는 TKG 클러스터에 vCenter Single Sign-On 사용자로 연결할 수 없습니다.

vCenter Single Sign-On 사용자로 감독자 또는 TKG 클러스터에 연결하려고 하면 kubectl용 vSphere 플러그인에서 `Error from server (Forbidden)` 오류 메시지를 반환합니다.

vSphere 네임스페이스에 대한 역할 사용 권한이 충분하지 않거나 사용자 계정에 액세스 권한이 부여되지 않았습니다.

클러스터를 운영하는 DevOps 엔지니어인 경우 vSphere 네임스페이스에 대한 **편집** 권한이 부여되었는지 vSphere 관리자에게 확인하십시오. 클러스터를 사용하여 워크로드를 배포하는 개발자인 경우 클러스터 액세스 권한이 부여되었는지 관리자에게 확인하십시오.

Kubectl vSphere 로그인 오류

kubectl용 vSphere 플러그인을 사용하여 감독자 또는 TKG 클러스터에 로그인하려고 할 때 다음 오류가 발생하면 로그인 오류 때문일 수 있습니다.

```
Failed to get available workloads, response from the server was invalid.
```

로그인 오류 문제를 해결하려면 `-v=10`을 사용하여 더 많은 세부 로그 출력을 가져옵니다.

```
kubectl vsphere login --server=10.110.150.56 --vsphere-username user@vsphere.local -v=10
```

예를 들어 다음에서는 세부 정보 표시 출력을 사용하여 `invalid or missing credentials` 오류를 확인합니다.

```
DEBU[0000] User passed verbosity level: 10
DEBU[0000] Setting verbosity level: 10
DEBU[0000] Setting request timeout:
DEBU[0000] login called as: /usr/local/bin/kubectl-vsphere login --server=10.110.150.56 --
vsphere-username user@vpshere.local -v=10
DEBU[0000] Creating wcp.Client for --server=10.110.150.56.
INFO[0000] Does not appear to be a vCenter or ESXi address.
DEBU[0000] Got response:
INFO[0000] Using user@vpshere.local as username.
DEBU[0000] Env variable KUBECTL_VSPHERE_PASSWORD is present
DEBU[0000] Error while getting list of workloads: invalid or missing credentials
FATA[0000] Failed to get available workloads, response from the server was invalid.
```

감독자에 SSH 사용

로그인 오류 문제를 해결하기 위해 감독자에 SSH를 사용해야 할 수 있습니다.

경고 감독자 제어부 노드에 SSH로 연결하면 감독자 클러스터를 영구적으로 손상시킬 수 있는 사용 권한이 있습니다. 고객이 감독자 제어부 노드에서 감독자 구성 요소를 변경했다는 증거를 VMware 지원팀이 발견하면 VMware 지원팀은 감독자 클러스터를 지원되지 않는 것으로 표시하고 vSphere IaaS control plane 솔루션을 다시 배포하도록 요구할 수 있습니다. 이 세션은 네트워크를 테스트하고, 로그를 확인하고, kubectl logs/get/describe 명령을 실행할 때만 사용하십시오. KB 또는 VMware 지원의 명시적인 사용 권한 없이는 이 세션에서 아무것도 배포, 삭제 또는 편집하지 마십시오.

SSH를 통해 감독자 제어부 노드에 연결하려면 다음 단계를 완료합니다.

1 루트 사용자 계정을 사용하여 vCenter에 로그인합니다.

2 `dcli +i`를 입력하여 대화형 모드에서 데이터 센터 CLI를 사용합니다.

3 명령 `namespacemanagement software clusters list`를 실행하여 감독자의 상태를 반환합니다.

4 `exit`를 입력하여 `dcli` 셸을 종료합니다.

5 `shell`을 입력하여 Bash 셸 모드로 전환합니다.

6 `/usr/lib/vmware-wcp/decyptK8Pwd.py`를 입력하여 감독자에 대한 IP 주소 및 암호를 가져옵니다.

7 `ssh 10.100.150.56`을 입력하여 감독자에 SSH를 사용합니다. 여기서 예제 IP 주소를 이전 명령에서 반환된 IP 주소로 바꿉니다.

컨텐츠 라이브러리 오류 문제 해결

Tanzu Kubernetes 릴리스 컨텐츠 라이브러리 오류 문제를 해결하려면 이 항목의 팁을 참조하십시오.

TKR 리소스를 찾을 수 없음

vSphere 관리자가 컨텐츠 라이브러리를 생성하고 지원되는 Tanzu Kubernetes 릴리스와 동기화했습니다. TKG 클러스터를 배포하는 vSphere 네임스페이스에 컨텐츠 라이브러리를 할당했습니다. 감독자에 로그인하고 컨텍스트를 vSphere 네임스페이스로 전환했습니다.

다음 명령을 실행하면 `No resources found`가 반환됩니다.

```
kubectl get tanzukubernetesreleases
```

```
kubectl get tkr
```

문제를 해결하려면 다음 명령을 실행합니다.

```
kubectl get virtualmachineimages -A
```

```
kubectl get vmimage -o wide
```

컨텐츠 라이브러리가 있고 네임스페이스에 등록되어 있는지 확인합니다.

```
kubectl get contentsources
```

```
 kubectl get contentsourcebindings -A
```

이 문제를 해결하려면 vCenter Server 관리 인터페이스에 로그인합니다. **서비스**로 이동하고 **컨텐츠 라이브러리 서비스**를 선택한 후 **다시 시작**을 클릭합니다.

이렇게 해도 문제가 해결되지 않으면 네임스페이스에서 컨텐츠 라이브러리를 제거해야 할 수 있습니다. 이렇게 하려면 새 컨텐츠 라이브러리를 생성하여 네임스페이스에 추가하고 이전 컨텐츠 라이브러리를 제거합니다.

라이브러리 항목 가져오기 실패

TKG 클러스터를 프로비저닝하려고 할 때 vSphere 네임스페이스와 동기화되고 연결된 구독 컨텐츠 라이브러리에서 항목을 가져올 수 없습니다.

다음 오류가 표시됩니다.

```
Internal error occurred: get library items failed for.
```

구독 컨텐츠 라이브러리가 스토리지 용량 제한에 도달하면 TKG 클러스터를 프로비저닝할 수 없습니다. 컨텐츠 라이브러리는 연결된 스토리지를 통해 지원됩니다. 시간이 경과하면 Kubernetes 버전이 더 많이 릴리스되고 OVA 파일을 라이브러리에 동기화되면 스토리지 용량이 가득 찰 수 있습니다.

TKR을 자동으로 동기화하는 경우 수동 동기화로 전환하는 것을 고려하고 필요한 TKR 이미지만 로컬에 저장합니다. 주문형 동기화를 이미 사용 중인 경우 라이브러리에서 더 이상 필요하지 않은 이미지를 정리합니다. 또는 새 컨텐츠 라이브러리로 마이그레이션할 수 있습니다.

로컬 컨텐츠 라이브러리에서 TKR을 찾을 수 없음

로컬 컨텐츠 라이브러리는 인터넷이 제한된 환경에서 사용할 수 있습니다.

로컬 컨텐츠 라이브러리를 생성할 때 라이브러리에 보안 정책을 적용하는 옵션이 있습니다. Tanzu Kubernetes 릴리스가 라이브러리에 업로드된 경우에도 다음 조건 중 하나에 해당하는 경우 TKG 클러스터에서 이 릴리스를 사용할 수 없습니다.

- 컨텐츠 라이브러리의 OVF 패키지가 서명되지 않았습니다.
- OVF 패키지가 잘못된 인증서를 사용하여 서명되었습니다.
- OVF 패키지가 로컬 컨텐츠 라이브러리가 구성된 vCenter Server에서 신뢰하지 않는 인증서를 사용하여 서명되었습니다.

OVA 및 VMDK 파일을 컨텐츠 라이브러리에 업로드할 때 매니페스트 파일이 파일을 업로드하는 홈 디렉토리에 인증서가 있는지 확인합니다.

VM 클래스 오류 문제 해결

가상 시스템 클래스는 TKG 서비스 클러스터를 프로비저닝하는 vSphere 네임스페이스에 연결되어야 합니다.

가상 시스템 클래스 바인딩 오류

대상 vSphere 네임스페이스에 추가하지 않은 하나 이상의 VM 클래스를 사용하여 TKGS 클러스터를 프로비저닝하려고 하면 `VirtualMachineClassBindingNotFound` 오류가 발생하며, 그 예는 아래에 나와 있습니다.

```
conditions:
  - lastTransitionTime: "2021-04-25T02:50:58Z"
    message: 1 of 2 completed
    reason: VirtualMachineClassBindingNotFound @ Machine/test-cluster
    severity: Error
    status: "False"
    type: ControlPlaneReady
  - lastTransitionTime: "2021-04-25T02:49:21Z"
    message: 0/1 Control Plane Node(s) healthy. 0/2 Worker Node(s) healthy
```

```
    reason: WaitingForNodesHealthy
    severity: Info
    status: "False"
    type: NodesHealthy
```

오류를 수정하려면 TKG 서비스 클러스터에 사용할 VM 클래스로 vSphere 네임스페이스를 구성합니다. `kubectl get virtualmachineclass` 명령을 실행하여 vSphere 네임스페이스와 연결된 VM 클래스를 봅니다.

고지 사항 `kubectl get virtualmachineclassbindings` 명령은 vSphere 8 U3에서 더 이상 사용되지 않습니다. 올바른 명령은 `virtualmachineclass`입니다.

경고 `kubectl get virtualmachineclasses` 명령은 감독자에서 사용 가능한 모든 VM 클래스를 반환합니다. 그러나 클러스터를 프로비저닝할 때 대상 vSphere 네임스페이스에 연결된 VM 클래스만 사용 가능하므로 복수형 명사는 정보 제공만을 위한 것이며 프로비저닝 시 사용할 수 없습니다.

TKGS 클러스터 프로비저닝 오류 문제 해결

TKGS 클러스터를 프로비저닝할 수 없는 경우 이 일반 오류 목록을 검토하여 문제를 해결합니다.

클러스터 API 로그 확인

TKG 클러스터를 생성할 수 없는 경우 CAPW/V가 작동하는지 확인합니다.

CAPW/V 컨트롤러는클러스터 API의 인프라별 구현입니다. CAPW/V는 감독자를 통해 사용하도록 설정됩니다. CAPW/V는 TKG의 구성 요소이며 TKG 클러스터의 수명 주기 관리를 담당합니다.

CAPW/V는 VirtualNetwork의 생성 및 업데이트를 담당합니다. VirtualNetwork가 준비된 경우에만 클러스터 노드 생성을 진행할 수 있습니다. 클러스터 생성 워크플로가 이 단계를 통과했습니까?

CAPW/V는 VirtualMachineService 생성 및 업데이트를 담당합니다. VirtualMachineService가 생성되었습니까? 외부 IP를 얻었습니까? 클러스터 생성 워크플로가 이 단계를 통과했습니까?

이러한 질문에 답하려면 다음과 같이 Cluster API 로그를 확인하십시오.

```
kubectl config use-context tkg-cluster-ns
```

```
kubectl get pods -n vmware-system-capw  | grep capv-controller
```

```
kubectl logs -n vmware-system-capw -c manager capv-controller-manager-...
```

클러스터 규격 검증 오류

YAML 규격에 따라 공백 문자를 키 이름에 사용할 수 있습니다. 공백을 포함하는 스칼라 문자열이며 따옴표가 필요하지 않습니다.

그러나 TKGS 유효성 검사에서는 키 이름에 공백 문자를 사용할 수 없습니다. TKGS에서 유효한 키 이름은 영숫자, 대시(예: `key-name`), 밑줄(예: `KEY_NAME`) 또는 점(예: `key.name`)으로만 구성되어야 합니다.

클러스터 규격의 키 이름에 공백 문자를 사용하는 경우 TKGS 클러스터가 배포되지 않습니다. vmware-system-tkg-controller-manager 로그에 다음 오류 메시지가 표시됩니다.

```
Invalid value: \"Key Name\": a valid config key must consist of alphanumeric
characters, '-', '_' or '.' (e.g. 'key.name', or 'KEY_NAME', or 'key-name', regex used
for validation is '[-._a-zA-Z0-9]+')"
```

오류를 수정하려면 공백 문자를 완전히 제거하거나 지원되는 문자로 교체합니다.

TKG 클러스터 YAML 적용 시 오류 발생

TKG 클러스터 YAML을 적용할 때 오류가 발생하면 다음과 같이 문제를 해결합니다.

클러스터 네트워크가 올바른 상태가 아님

TKG 클러스터 프로비저닝 워크플로 이해:

- CAPV는 각 TKG 클러스터 네트워크에 대한 VirtualNetwork 개체를 생성합니다.
- 감독자가 NSX 네트워킹으로 구성된 경우 NCP는 VirtualNetwork 개체를 감시하고 각 VirtualNetwork에 대해 NSX Tier-1 라우터와 NSX 세그먼트를 생성합니다.
- CAPV는 VirtualNetwork의 상태를 확인하고 준비가 되면 워크플로의 다음 단계로 진행합니다.

VM 서비스 컨트롤러는 CAPV에 의해 생성된 사용자 지정 개체를 감시하고 이러한 규격을 사용하여 TKG 클러스터를 구성하는 VM을 생성하고 구성합니다.

NCP(NSX Container Plugin)는 Kubernetes API를 통해 etcd에 추가된 네트워크 리소스를 감시하고 NSX에서 해당 개체의 생성을 오케스트레이션하는 컨트롤러입니다.

이러한 각 컨트롤러는 감독자 제어부에서 Kubernetes 포드로 실행됩니다. 네트워크 문제를 해결하려면 CAPV 컨트롤러 로그, VM 서비스 로그 및 NCP 로그를 확인하십시오.

컨테이너 로그를 확인합니다. 여기서 `name-XXXX`는 실행할 때 고유한 포드 이름입니다.

```
kubectl get pods -A
kubectl logs pod/name-XXXXX -c pod-name -n namesapce
```

제어부 노드 수가 잘못됨

감독자의 TKG 클러스터는 1개 또는 3개의 제어부 노드를 지원합니다. 다른 수의 복제본을 입력하면 클러스터 프로비저닝이 실패합니다.

제어부/작업자 VM에 대한 스토리지 클래스가 잘못됨

다음 명령을 실행합니다.

```
kubectl describe ns <tkg-clsuter-namesapce>
```

TKG 클러스터를 생성하려는 네임스페이스에 스토리지 클래스가 할당되었는지 확인합니다. 해당 스토리지 클래스를 참조하는 vSphere 네임스페이스에 ResourceQuota가 있어야 하며 스토리지 클래스가 감독자에 존재해야 합니다.

이름이 감독자에 있는 스토리지 클래스와 일치하는지 확인합니다. `kubectl get storageclasses` 감독자를 vSphere 관리자 권한으로 실행합니다. WCP는 스토리지 프로파일을 감독자에 적용할 때 이름을 변환할 수 있습니다(예: 하이픈이 밑줄이 됨).

잘못된 VM 클래스

클러스터 YAML에 제공된 값이 `kubectl get virtualmachineclass`에서 반환된 VM 클래스 중 하나와 일치하는지 확인합니다. 바인딩된 클래스만 TKG 클러스터에서 사용할 수 있습니다. VM 클래스는 vSphere 네임스페이스에 추가할 때 바인딩됩니다.

`kubectl get virtualmachineclasses` 명령은 감독자의 모든 VM 클래스를 반환하지만 바인딩된 클래스만 사용할 수 있습니다.

TKR 배포를 찾을 수 없음

다음과 유사한 오류가 표시되는 경우:

```
"Error from server (unable to find Kubernetes distributions):
admission webhook "version.mutating.tanzukubernetescluster.run.tanzu.vmware.com"
denied the request: unable to find Kubernetes distributions"
```

컨텐츠 라이브러리 문제 때문일 수 있습니다. 사용 가능한 항목을 나열하려면 `kubectl get virtualmachineimages -A` 명령을 사용합니다. 결과는 컨텐츠 라이브러리에서 사용 가능하고 동기화되거나 업데이트되는 항목입니다.

감독자의 TKG에는 새 TKR API와 호환되는 새 TKR 이름이 있습니다. 컨텐츠 라이브러리에서 각 TKR의 이름을 올바르게 지정해야 합니다.

컨텐츠 라이브러리의 이름: `photon-3-amd64-vmi-k8s-v1.23.8---vmware.2-tkg.1-zshippable`

TKG 클러스터 규격의 해당 이름: `version: v1.23.8+vmware.2-tkg.1-zshippable`

TKG YAML이 적용되지만 VM이 생성되지 않음

TKG 2.0 클러스터 YAML이 유효하고 적용되지만 노드 VM이 생성되지 않은 경우 다음과 같이 문제를 해결합니다.

CAPI/CAPV 리소스 확인

TKG가 CAPI/CAPV 수준 리소스를 생성했는지 확인합니다.

- CAPV가 VirtualMachine 리소스를 생성했는지 확인합니다.
- VM Operator 로그를 확인하여 VM이 생성되지 않은 이유를 확인합니다. 예를 들어 ESX 호스트의 리소스가 부족하여 OVF 배포가 실패했을 수 있습니다.
- CAPV 및 VM Operator 로그를 확인합니다.

- NCP 로그를 확인합니다. NCP는 제어부에 대한 VirtualNetwork, VirtualNetworkInterface, LoadBalancer 인식을 담당합니다. 이러한 리소스와 관련된 오류가 있는 경우 문제가 될 수 있습니다.

가상 시스템 서비스 오류

가상 시스템 서비스 오류

- 네임스페이스에서 `kubectl get virtualmachineservices` 명령을 실행합니다.
- 가상 시스템 서비스가 생성되었습니까?
- 네임스페이스에서 `kubectl describe virtualmachineservices` 명령을 실행합니다.
- 가상 시스템 서비스에 보고된 오류가 있습니까?

가상 네트워크 오류

네임스페이스에서 `kubectl get virtualnetwork` 명령을 실행합니다.

이 클러스터에 대해 가상 네트워크가 생성되었습니까?

네임스페이스에서 `kubectl describe virtual network` 명령을 실행합니다.

VM에 대한 가상 네트워크 인터페이스가 생성되었습니까?

TKG 클러스터 제어부가 실행되지 않음

TKG 제어부가 실행되지 않으면 오류가 발생했을 때 리소스가 준비되었는지 확인합니다. 조인 노드 제어부가 작동하지 않습니까? 아니면 Init 노드가 작동하지 않습니까? 또한 노드 개체에 제공자 ID가 설정되어 있지 않은지 확인합니다.

오류가 발생했을 때 리소스가 준비되었는지 확인

로그를 찾아보는 것 외에도 관련 개체의 상태 ControlPlaneLoadBalancer를 확인하면 오류가 발생했을 때 리소스가 준비되었는지 이해하는 데 도움이 됩니다. 네트워크 문제 해결을 참조하십시오.

조인 노드 제어부가 작동하지 않습니까? 아니면 Init 노드가 작동하지 않습니까?

노드 조인이 제대로 작동하지 않는 경우가 있습니다. 특정 VM에 대한 노드 로그를 확인합니다. Init 노드가 나타나지 않으면 클러스터에 작업자 및 제어부 노드가 없을 수 있습니다.

노드 개체에 제공자 ID가 설정되지 않음

VM이 생성되면 IP가 있는지 확인한 다음 cloud-init 로그를 확인합니다(kubeadm 명령이 제대로 실행됨).

CAPI 컨트롤러 로그를 확인하여 문제가 있는지 확인합니다. TKG 클러스터에서 `kubectl get nodes` 명령을 사용하여 확인한 다음 노드 개체에 제공자 ID가 있는지 확인할 수 있습니다.

TKG 작업자 노드가 생성되지 않음

TKG 클러스터 및 제어부 VM이 생성되었지만 작업자가 생성되지 않았거나 다른 가상 시스템 개체가 생성되지 않은 경우 다음을 시도합니다.

```
kubectl describe cluster CLUSTER-NAME
```

네임스페이스에서 가상 시스템 리소스를 확인합니다. 다른 리소스가 생성되었습니까?

그렇지 않은 경우 CAPV 로그를 확인하여 사용할 수 없는 다른 가상 시스템 개체 부트스트랩 데이터가 생성되지 않는 이유를 확인합니다.

CAPI가 로드 밸런서(노드 VM IP가 있는 NSX 또는 외부 로드 밸런서가 있는 VDS)를 통해 TKG 클러스터 제어부와 통신할 수 없는 경우, 네임스페이스의 암호를 사용하여 TKG 클러스터 kubeconfig를 가져옵니다.

네임스페이스의 암호를 사용하여 TKG 클러스터 kubeconfig를 가져옵니다.

```
kubectl get secret -n <namespace> <tkg-cluster-name>-kubeconfig -o jsonpath='{.data.value}' |
base64 -d
> tkg-cluster-kubeconfig; kubectl --kubeconfig tkg-cluster-kubeconfig get pods -A
```

'연결이 거부됨'으로 인해 실패하면 제어부가 제대로 초기화되지 않았을 수 있습니다. I/O 시간 초과가 있는 경우 kubeconfig에서 IP 주소에 대한 연결을 확인합니다.

내장된 로드 밸런서가 있는 NSX:

- 제어부 LB가 작동 중이고 연결 가능한지 확인합니다.
- LB에 IP가 없는 경우 NCP 로그를 확인하고 NSX-T UI를 확인하여 관련 구성 요소가 올바른 상태인지 확인합니다. (NSX-T LB, VirtualServer, ServerPool은 모두 정상 상태여야 합니다.)
- LB에 IP가 있지만 연결할 수 없는 경우 `curl -k https://<LB- VIP>:6443/healthz` 명령은 승인되지 않음 오류를 반환해야 합니다.

  LoadBalancer 서비스 유형 외부 IP가 "보류 중" 상태인 경우 TKG 클러스터가 감독자 LB VIP를 통해 감독자 Kubernetes API와 통신할 수 있는지 확인합니다. 겹치는 IP 주소가 없는지 확인합니다.

TKG 제어부 노드가 정상 상태인지 확인합니다.

- TKG 클러스터 제어부가 오류를 보고하는지 확인합니다(예: 제공자 ID로 노드를 만들 수 없음).
- TKG 클러스터 클라우드 제공자가 노드를 올바른 제공자 ID로 표시하지 않았으므로 CAPI는 게스트 클러스터 노드의 제공자 ID와 감독자 클러스터의 시스템 리소스를 비교하여 확인할 수 없습니다.

제어부 VM에 SSH를 통해 연결하거나 TKG 클러스터 kubeconfig를 사용하여 TKG Cloud Provider Pod가 실행 중이거나 기록된 오류가 있는지 확인합니다. Kubernetes 관리자 및 시스템 사용자로 TKG 서비스 클러스터에 연결의 내용을 참조하십시오.

```
kubectl get po -n vmware-system-cloud-provider
```

```
kubectl logs -n vmware-system-cloud-provider <pod name>
```

VMOP가 VirtualMachineService를 성공적으로 조정하지 못한 경우 VM Operator 로그를 확인합니다.

NCP에서 NSX-T 리소스를 생성하는 데 문제가 있는 경우 NCP 로그를 확인합니다.

제어부가 제대로 초기화되지 않은 경우 VM IP를 확인합니다. 상태에 VM IP가 포함되어야 합니다.

```
kubectl get virtualmachine -n <namespace> <TKC-name>-control-plane-0 -o yaml
```

```
ssh vmware-system-user@<vm-ip> -i tkc-cluster-ssh
```

kubeadm이 오류를 기록했는지 확인합니다.

```
cat /var/log/cloud-init-output.log | less
```

프로비저닝된 TKG 클러스터가 "생성" 단계에서 멈춤

다음 명령을 실행하여 클러스터 상태를 확인합니다.

```
kubectl get tkc -n <namespace>
```

```
kubectl get cluster -n <namespace>
```

```
kubectl get machines -n <namespace>
```

KubeadmConfig가 있었지만 CAPI에서 찾을 수 없었습니다. vmware-system-capv의 토큰에 kubeadmconfig를 쿼리할 수 있는 올바른 사용 권한이 있는지 확인했습니다.

```
$kubectl --token=__TOKEN__ auth can-i get kubeadmconfig
yes
```

controller-runtime 캐시가 업데이트되지 않았을 수 있습니다. CAPI 감시 캐시가 오래되어 새 개체를 선택하지 않을 수 있습니다. 필요한 경우 capi-controller-manager를 다시 시작하여 문제를 해결합니다.

```
kubectl rollout restart deployment capi-controller-manager -n vmware-system-capv
```

vSphere 네임스페이스가 "종료" 단계에서 멈춤

TKR, 감독자 및 vCenter가 버전 호환성 관점에서 동기화되어 있는지 확인합니다.

네임스페이스는 네임스페이스 아래의 모든 리소스가 차례로 삭제된 경우에만 삭제할 수 있습니다.

```
kubectl describe namespace NAME
```

다음 오류가 발견되었습니다. "서버 오류(Kubernetes 배포를 찾을 수 없음): admission webhook “version.mutating.tanzukubernetescluster.run.tanzu.vmware.com” denied the request: Kubernetes 배포를 찾을 수 없음"

vCenter에서 가상 시스템 이미지를 확인합니다.

```
kubectl get virtualmachineimages -A
```

TKG 서비스 클러스터 노드 오류 문제 해결

TKGS 클러스터를 프로비저닝했으며 노드가 생성되었지만 하나 이상의 가상 시스템이 시작되지 않거나 오류가 있는 경우 다음 문제 해결 팁을 시도해보십시오.

관련 CRD 확인

VM이 생성되면 관련 CRD를 생성해야 합니다. 관련 CRD가 생성되어 존재하는지 확인합니다 (machinedeployments, virtualmachines).

시스템 배포 확인:

```
kubectl get machinedeployments -A -o wide
```

가상 시스템 확인:

```
kubectl get virtualmachines -A
```

노드 크기 확인

TKG 클러스터를 프로비저닝했습니다. 시스템에서 제어부 가상 시스템의 전원을 켜려고 시도하는데, 오류가 발생하고 다음 메시지가 표시됩니다.

```
The host does not have sufficient CPU resources to satisfy the reservation.
```

가상 시스템 크기 또는 클래스가 클러스터 배포에 충분하지 않습니다. 가상 시스템 유형 또는 클래스를 변경합니다. 제어부 및 작업자 노드 모두에 대해 초소형 및 소형 VM 클래스 유형을 사용하지 마십시오.

TKG 서비스 클러스터 네트워킹 오류 문제 해결

TKGS 클러스터 네트워킹 오류 문제를 해결하려면 이 섹션의 팁을 참조하십시오.

노드 네트워킹 확인

각 TKG 클러스터에는 다음과 같은 네트워크 리소스가 있어야 합니다.

네트워크 개체	네트워크 리소스	설명	문제 해결	명령
VirtualNetwork	Tier-1 라우터 및 연결된 세그먼트	클러스터에 대한 노드 네트워크	SNAT IP가 할당되었는지 확인	`kubectl get virtualnetwork -n NS-NAME`
VirtualNetworkInterface	세그먼트의 논리적 포트	클러스터 노드에 대한 노드 네트워크 인터페이스	각 VirtualMachine에 IP 주소가 있는지 확인	`kubectl get virtualmachines -n NS-NAME NODE-NAME`

제어부에 대한 로드 밸런서 확인

TKG 클러스터 제어부에 대한 로드 밸런서는 Kubernetes API 서버에 대한 액세스를 제공합니다. 이 로드 밸런서는 클러스터 생성 중에 시스템에 의해 자동으로 프로비저닝됩니다. 다음 리소스가 있어야 합니다.

제어부 로드 밸런서의 상태를 확인하면 오류가 발생했을 때 리소스가 준비되었는지 파악하는 데 도움이 될 수 있습니다. 일반적으로 감독자 클러스터에 대해 다음 명령으로 이 로드 밸런서 찾기를 사용하여 이 로드 밸런서를 찾습니다. `kubectl get services -A | grep control-plane-service`

네트워크 개체	네트워크 리소스	설명	문제 해결	명령
VirtualMachineService	해당 없음	VirtualMachineService가 생성되고 k8s 서비스로 변환됩니다.	상태가 업데이트되고 로드 밸런서 VIP(가상 IP)가 포함되어 있는지 확인합니다.	`kubectl get virtualmachines ervices -n NS-NAME SERVICE-NAME`

서비스	VirtualServer 인스턴스 및 연결된 서버 풀(멤버 풀)이 있는 로드 밸런서 서버	TKG 클러스터 API 서버에 액세스하기 위해 Load Balancer 유형의 Kubernetes 서비스가 생성됩니다.	외부 IP가 할당되었는지 확인합니다. LB 서비스의 외부 IP를 통해 TKG 클러스터 API에 액세스할 수 있는지 확인합니다.	감독자 네임스페이스:
				`kubectl get services -A | grep control-plane-service`
				클러스터 네임스페이스:
				`kubectl get services -n NS-NAME`
				둘 중 하나의 네임스페이스
				`curl -k https://EXTERNAL-IP:PORT/healthz`

끝점	끝점 멤버(TKG 클러스터 제어부 노드)는 멤버 풀에 있어야 합니다.	모든 TKG 클러스터 제어부 노드를 포함하도록 끝점이 생성됩니다.		`kubectl get endpoints -n NS-NAME SERVICE-NAME`

작업자 노드에서 로드 밸런서 서비스 확인

TKG 클러스터 작업자 노드에 대한 로드 밸런서 인스턴스는 LoadBalancer 유형의 Kubernetes 서비스가 생성될 때 사용자가 생성합니다.

첫 번째 단계는 클라우드 제공자가 TKG 클러스터에서 실행되고 있는지 확인하는 것입니다.

```
kubectl get pods -n vmware-system-cloud-provider
```

관련 Kubernetes 개체가 생성되었고 올바른 상태에 있는지 확인합니다.

네트워크 개체	네트워크 리소스	설명	명령
감독자의 VirtualMachineService	해당 없음	감독자에서 VirtualMachineService가 생성되고 감독자에서 Kubernetes 서비스로 변환됨	`kubectl get virtualmachineservice -n NS-NAME SVC-NAME`
감독자의 로드 밸런서 서비스	TKG 클러스터 로드 밸런서의 VirtualServer 및 연결된 멤버 풀.	이 LB 유형의 서비스에 액세스하기 위해 감독자에서 로드 밸런서 서비스가 생성됨	`kubectl get services -n NS-NAME SVC-NAME`
감독자의 끝점	끝점 멤버(TKG 클러스터 작업자 노드)는 NSX의 멤버 풀에 있어야 합니다.	모든 TKG 클러스터 작업자 노드를 포함하도록 끝점이 생성됩니다.	`# kubectl get endpoints -n NS-NAME SVC-NAME`
TKG 클러스터의 로드 밸런서 서비스	해당 없음	사용자가 배포한 TKG 클러스터의 로드 밸런서 서비스의 상태가 로드 밸런서 IP로 업데이트되어야 합니다.	`kubectl get services`

감독자 NSX 네트워킹 스택 확인

컨트롤러 포드에서 실행되는 Kubernetes API 서버, NCP 포드 및 관리자 컨테이너는 인프라 네트워킹 문제를 확인하기 위한 기본 시작점입니다.

다음 오류 메시지는 ESXi 호스트 NIC가 연결되어 있는 물리적 포트 그룹을 포함하여 네트워크 패브릭의 모든 지점에서 라우팅 또는 MTU 문제를 나타낼 수 있습니다.

```
{"log":"I0126 19:40:15.347154 1 log.go:172] http: TLS handshake error from
100.64.128.1:4102: EOF\n","stream":"stderr","time":"2021-01-26T19:40:15.347256146Z"}
```

문제를 해결하려면 SSH를 통해 ESXi 호스트에 연결하고 다음 명령을 실행합니다.

```
esxcli network ip interface ipv4 get
```

이 명령은 호스트의 모든 vmkernel 인터페이스를 나열합니다. 단일 TEP 인터페이스가 있는 경우 항상 vmk10입니다. 2번째 또는 3번째 TEP 인터페이스가 있는 경우 vmk11 및 vmk12 등이 됩니다. 생성된 TEP 인터페이스의 수량은 업링크 프로파일의 TEP에 할당한 업링크 수에 따라 다릅니다. 업링크 간 TEP에 대해 "로드 공유"를 선택한 경우 업링크별로 TEP 인터페이스가 생성됩니다.

기본 TEP-TEP ping 명령에는 다음과 같은 구문이 있습니다.

```
vmkping ++netstack=vxlan -s 1572 -d -I vmk10 10.218.60.66
```

여기서

- `-s`는 패킷 크기입니다.
- `-d`는 조각화하지 않음을 의미합니다.
- `-I`는 vmk10의 링크를 소스로 지정합니다.
- `IP address`는 ping하는 다른 ESXi 호스트 또는 NSX Edge의 TEP 인터페이스입니다.

MTU가 1600으로 설정되면 1573을 초과하는 패킷 크기는 실패합니다(1500보다 큰 MTU만 필요). MTU가 1500으로 설정되면 1473을 초과하는 항목은 실패합니다. ping을 소싱하려는 추가 TEP 인터페이스가 있는 경우 vmk11로 변경할 수 있습니다.

실패한 TKG 클러스터 업그레이드 다시 시작

TKG 클러스터의 업데이트가 실패한 경우 업데이트 작업을 다시 시작하고 업데이트를 다시 시도할 수 있습니다.

문제

TKG 클러스터의 업데이트가 실패하고 클러스터 상태가 `upgradefailed`가 됩니다.

원인

클러스터 업데이트 실패에는 스토리지 부족과 같은 몇 가지 이유가 있을 수 있습니다. 실패한 업데이트 작업을 다시 시작하고 클러스터 업데이트를 다시 시도하려면 다음 절차를 완료하십시오.

해결책

1 감독자에 관리자로 로그인합니다.

2 `update_job_name`을 조회합니다.

```
kubectl get jobs -n vmware-system-tkg -l "run.tanzu.vmware.com/cluster-namespace=$
{cluster_namespace},cluster.x-k8s.io/cluster-name=${cluster_name}"
```

3 `curl`을 사용하여 요청을 실행할 수 있도록 `kubectl proxy`를 실행합니다.

```
kubectl proxy &
```

`Starting to serve on 127.0.0.1:8001`이 표시되어야 합니다.

참고 `kubectl`을 사용하여 리소스의 `.status`를 패치하거나 업데이트할 수 없습니다.

4 `curl`을 사용하면 다음 패치 명령을 실행하여 `.spec.backoffLimit`을 높입니다.

```
curl -H "Accept: application/json" -H "Content-Type: application/json-patch+json"
--request PATCH --data '[{"op": "replace", "path": "/spec/backoffLimit", "value": 8}]'
http://127.0.0.1:8001/apis/batch/v1/namespaces/vmware-system-tkg/jobs/${update_job_name}
```

5 `curl`을 사용하면 다음 패치 명령을 실행하여 작업 컨트롤러가 새 포드를 생성할 수 있도록 `.status.conditions`를 지웁니다.

```
$ curl -H "Accept: application/json" -H "Content-Type: application/json-patch+json"
--request PATCH --data '[{"op": "remove", "path": "/status/conditions"}]'
http://127.0.0.1:8001/apis/batch/v1/namespaces/vmware-system-tkg/jobs/${update_job_name}/
status
```

컨테이너 배포 오류 문제 해결

인증된 사용자에 대해 포드 보안 정책 및 역할 기반 액세스 제어가 구성되지 않은 경우 컨테이너 배포 오류가 발생할 수 있습니다.

문제

TKG 2.0 클러스터에 컨테이너 워크로드를 배포했지만 워크로드가 시작되지 않습니다. 다음과 유사한 오류가 표시됩니다.

```
Error: container has runAsNonRoot and image will run as root.
```

원인

TKG 클러스터는 PodSecurityPolicy 승인 컨트롤러를 사용하도록 설정된 상태로 프로비저닝됩니다. 클러스터 관리자가 PodSecurityPolicy를 인증된 사용자에게 바인딩할 때까지 인증된 사용자는 권한이 있거나 없는 포드를 생성할 수 없습니다.

해결책

TKR 1.24 이하를 사용하는 경우 기본 PodSecurityPolicy에 대한 적절한 바인딩을 생성하거나 사용자 지정 PodSecurityPolicy를 정의합니다. TKR 1.25 이상을 사용하는 경우 포드 보안 승인을 구성합니다. 장 18 TKG 서비스 클러스터에 대한 보안 관리의 내용을 참조하십시오.

컨테이너 레지스트리 오류 문제 해결

TKG 클러스터 포드에서 외부 컨테이너 레지스트리를 사용할 수 있습니다.

컨테이너 레지스트리에서 이미지 끌어오기 오류 문제 해결

신뢰할 수 있는 인증서로 TKG를 구성하고 자체 서명된 인증서를 kubeconfig 클러스터에 추가하면 자체 서명된 인증서를 사용하는 개인 레지스트리에서 컨테이너 이미지를 성공적으로 끌어올 수 있습니다.

다음 명령은 포드 워크로드에 대해 컨테이너 이미지를 성공적으로 끌어왔는지 확인하는 데 유용할 수 있습니다.

```
kubectl describe pod PODNAME
```

이 명령은 주어진 포드에 대한 자세한 상태 및 오류 메시지를 보여줍니다. 클러스터에 사용자 지정 인증서를 추가하기 전에 이미지 끌어오기를 시도하는 예:

```
Events:
  Type     Reason                          Age               From               Message
  ----     ------                          ----              ----               -------
  Normal   Scheduled                       33s               default-scheduler  ...
  Normal   Image                           32s               image-controller   ...
  Normal   Image                           15s               image-controller   ...
  Normal   SuccessfulRealizeNSXResource  7s (x4 over 31s)  nsx-container-ncp  ...
  Normal   Pulling                         7s                kubelet            Waiting test-gc-
e2e-demo-ns/testimage-8862e32f68d66f727d1baf13f7eddef5a5e64bbd-v10612
  Warning  Failed                          4s                kubelet            failed to get
images: ... Error: ... x509: certificate signed by unknown authority
```

다음 명령을 실행하는 경우:

```
kubectl get pods
```

전체 포드 상태 보기에서도 `ErrImagePull` 오류를 볼 수 있습니다.

```
NAME                                          READY   STATUS         RESTARTS   AGE
testimage-nginx-deployment-89d4fcff8-2d9pz   0/1     Pending        0          17s
testimage-nginx-deployment-89d4fcff8-7kp9d   0/1     ErrImagePull   0          79s
testimage-nginx-deployment-89d4fcff8-7mpkj   0/1     Pending        0          21s
testimage-nginx-deployment-89d4fcff8-fszth   0/1     ErrImagePull   0          50s
testimage-nginx-deployment-89d4fcff8-sjnjw   0/1     ErrImagePull   0          48s
testimage-nginx-deployment-89d4fcff8-xr5kg   0/1     ErrImagePull   0          79s
```

"x509: certificate signed by unknown authority" 및 "ErrImagePull" 오류는 클러스터가 개인 컨테이너 레지스트리에 연결할 수 있는 올바른 인증서로 구성되지 않았음을 나타냅니다. 인증서가 누락되었거나 잘못 구성되었습니다.

인증서를 구성한 후 개인 레지스트리에 연결하는 데 오류가 발생하는 경우 구성에 적용된 인증서가 클러스터에 적용되었는지 확인할 수 있습니다. SSH를 사용하여 구성에 적용했던 인증서가 제대로 적용되었는지 여부를 확인할 수 있습니다.

SSH를 통해 작업자 노드에 연결하여 두 가지 조사 단계를 수행할 수 있습니다.

1 `/etc/ssl/certs/` 폴더에서 `tkg-<cert_name>.pem`이라는 파일을 확인합니다. 여기서 `<cert_name>`은 `TkgServiceConfiguration`에 추가된 인증서의 "name" 속성입니다. 인증서가 `TkgServiceConfiguration`에 있는 인증서와 일치하고 개인 레지스트리 사용이 여전히 작동하지 않으면 다음 단계를 수행하여 추가 진단을 진행합니다.

2 `openssl s_client -connect hostname:port_num` 명령을 실행하여 자체 서명된 인증서를 사용하는 대상 서버에 다음 openssl 연결 테스트를 실행합니다. 여기서 `hostname`은 자체 서명된 인증서를 사용하는 개인 레지스트리의 호스트 이름/DNS 이름이고 `port_num`은 서비스가 실행되고 있는 포트 번호(일반적으로 HTTPS의 경우 443임)입니다.

   자체 서명된 인증서를 사용하는 끝점에 연결하려고 할 때 openssl이 정확히 어떤 오류를 반환하는지 확인하고, 거기서부터 상황을 해결할 수 있습니다(예를 들어 `TkgServiceConfiguration`에 올바른 인증서를 추가하여). TKG 클러스터에 잘못된 인증서가 내장되어 있는 경우 올바른 인증서로 구성을 업데이트하고, TKG 클러스터를 삭제한 다음 올바른 인증서가 포함된 구성을 사용하여 다시 생성해야 합니다.

3 암호 데이터 맵의 컨텐츠가 이중 base64로 인코딩되어 있는지 확인합니다. 이중 base64 인코딩은 필수입니다. 데이터 맵 값의 컨텐츠가 base64로 이중으로 인코딩되지 않으면 결과 PEM 파일을 처리할 수 없습니다.

추가적인 신뢰할 수 있는 CA 오류 문제 해결

TKG 클러스터에 신뢰할 수 있는 CA 인증서를 더 추가하는 데 문제가 있는 경우 이 항목을 참조하십시오.

추가적인 신뢰할 수 있는 CA 오류 문제 해결

v1alpha3 API 또는 v1beta1 API를 사용하여 추가적인 신뢰할 수 있는 CA 인증서에 대한 값이 포함된 `trust` 변수를 포함할 수 있습니다. 일반적인 사용 사례는 개인 컨테이너 레지스트리 인증서를 클러스터에 추가하는 것입니다. TKG 서비스 클러스터를 개인 컨테이너 레지스트리와 통합의 내용을 참조하십시오.

예를 들어 v1beta1 API를 사용합니다.

```
topology:
  variables:
    - name: trust
      value:
        additionalTrustedCAs:
          - name: my-ca
```

이때, 암호는 다음과 같습니다.

```
apiVersion: v1
data:
  my-ca: # Double Base64 encoded CA certificate
kind: Secret
metadata:
  name: CLUSTER_NAME-user-trusted-ca-secret
  namespace: tap
type: Opaque
```

TKG 클러스터 규격에 `trust.additionalTrustedCAs` stanza를 추가하면 감독자가 TKG 서비스 클러스터를 위한 롤링 업데이트 모델 이해 방식으로 클러스터 노드를 업데이트합니다. 하지만 `trust` 값을 잘못 입력하면 시스템이 제대로 시작되지 않고 클러스터에 가입하지 못합니다.

v1beta1 API를 사용하는 경우 인증서의 컨텐츠가 이중 base64로 인코딩되어야 합니다. 인증서의 컨텐츠가 이중 base64로 인코딩되지 않은 경우 다음 오류가 표시될 수 있습니다.

```
ls cannot access '/var/tmp/_var_ib_containerd': No such file or directory
```

v1alpha3 API(또는 v1alpha2 API)를 사용하는 경우 인증서의 컨텐츠가 단일 base64로 인코딩되어야 합니다. 인증서의 컨텐츠가 base64로 인코딩되지 않은 경우 다음 오류가 표시될 수 있습니다.

```
"default.validating.tanzukubernetescluster.run.tanzu.vmware.com" denied the request:
Invalid certificate internalharbor, Error decoding PEM block for internalharbor in the
TanzuKubernetesCluster spec's trust configuration
```

적절한 인코딩을 사용하지 않으면 시스템 노드가 시작되지 않고 위의 오류가 표시됩니다. 이 문제를 해결하려면 인증서의 컨텐츠를 올바르게 인코딩하고 해당 값을 암호의 데이터 맵에 추가합니다.

'kubectl replace -f/tmp/kubectl-edit-2005376329.yaml'을 실행하여 이 업데이트를 다시 시도할 수 있습니다.