Application Service Adapter for VMware Tanzu Application Platform v1.0

Application Service Adapter for VMware Tanzu Application Platform 1.0

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

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

Contents

Application Service Adapter for VMware Tanzu Application Platform

Application Service Adapter provides compatibility with CF client interfaces while running on top of K8s and integrating with Tanzu Application Platform. Learn more in this topic.

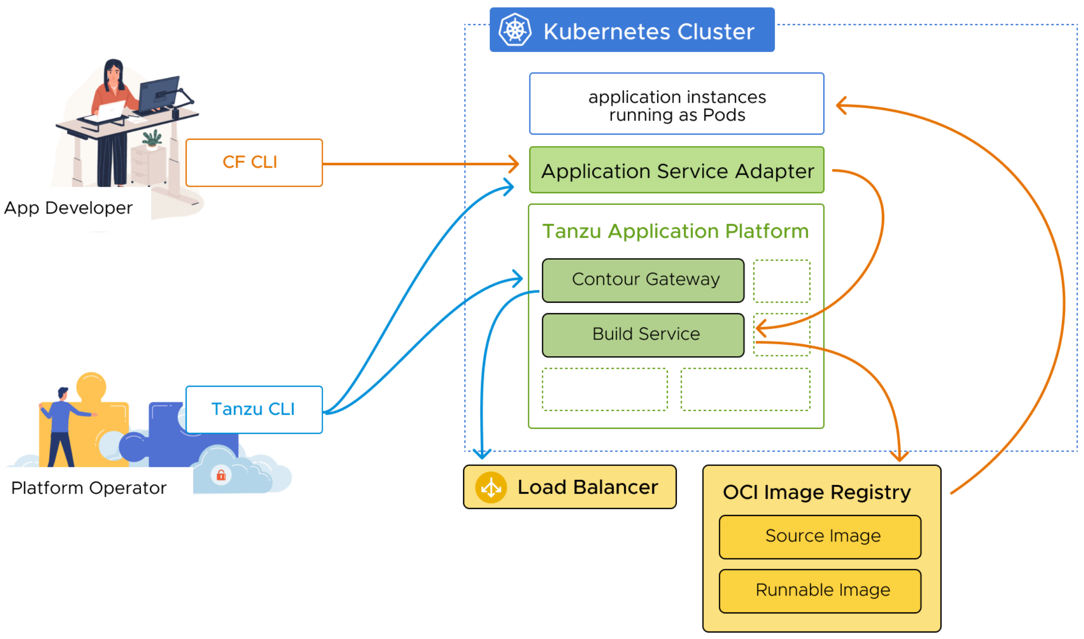
With Application Service Adapter, development teams using Tanzu Application Service for VMs tooling, such as the Cloud Foundry command-line interface (cf CLI) and other clients of the Cloud Foundry API (CAPI), can maintain their familiar workflows while their platform teams transition their infrastructure and deployments to Kubernetes.

For more information, see:

- Cloud Foundry
- Tanzu Application Service for VMs
- Tanzu Application Platform

Application Service Adapter Overview

The following diagram shows a high-level architecture of Application Service Adapter with user flows:



Application Service Adapter is distributed as a Carvel package for platform teams to configure and install to a Kubernetes cluster with the Tanzu CLI tool. The key user personas of Application Service

Adapter remain the same as the user personas of Tanzu Application Serivce: the platform operator and the app developer.

Platform teams create Cloud Foundry orgs and spaces in the installation, which the Application Service Adapter backs with separate Kubernetes namespaces. Application developers then use these orgs and spaces to organize their apps as they do today with Tanzu Application Service for VMs.

Application developers log in to their Application Service Adapter installation with credentials for the underlying Kubernetes cluster, but then use the cf CLI and CAPI to push apps and to map routes to them.

Application Service Adapter integrates with an existing installation of Tanzu Build Service component of Tanzu Application Platform to build container images for app code deployed to the platform. It also integrates with Contour to realize ingress routes to running apps.

To learn about Kubernetes cluster requirements and to plan your installation, see Reference architecture.

To install Application Service Adapter and its dependencies, go to Installing prerequisites.

Notice of telemetry collection for Application Service Adapter

Application Service Adapter for VMware Tanzu Application Platform participates in the VMware Customer Experience Improvement Program (CEIP). As part of CEIP, VMware collects technical information about your organization’s use of VMware products and services in association with your organization’s VMware license keys. For information about CEIP, see the Trust & Assurance Center. You may join or leave CEIP at any time. The CEIP Standard Participation Level provides VMware with information to improve its products and services, identify and fix problems, and advise you on how to best deploy and use VMware products. For example, this information can enable a proactive product deployment discussion with your VMware account team or VMware support team to help resolve your issues. This information cannot directly identify any individual.

You must acknowledge that you have read the VMware CEIP policy before you can proceed with the installation. For more information, see Configure the installation settings in *Installing Application Service Adapter*. To opt out of telemetry participation after installation, see Opting out of telemetry reporting in *Installing Application Service Adapter*.

Release notes for Application Service Adapter

You can view the release notes for v1.0.0 through v1.0.4 of Application Service Adapter for VMware Tanzu Application Platform on this page.

v1.0.4 Release

Release date: July 18, 2023

Changelog

- Updated package versions and associated dependencies.

v1.0.3 Release

Release date: June 7, 2023

Changelog

- Updated package versions and associated dependencies.

v1.0.2 Release

Release date: February 8, 2023

Security fixes

- System components in this release have been rebuilt with Go 1.19.5 to fix CVE-2022-41717 and CVE-2022-41720.

v1.0.1 Release

Release date: December 13, 2022

Resolved issues

- Application instance pods now have the `CF_INSTANCE_GUID`, `CF_INSTANCE_INTERNAL_IP`, `CF_INSTANCE_IP`, or `POD_NAME` environment variables set when Application Service Adapter is configured to use the experimental integration with the Supply Chain Choreographer.
- The output of `cf logs` no longer contains blank log lines interleaved with the log content from the appplication instances.

- The values schema published in the Tanzu package for Application Service Adapter now matches the values that the package installation recognizes.

Components

This release contains the following components:

- cartographer-builder-runner @ 9ae3d6c
- Korifi @ v0.3.0
- tas-adapter-telemetry-controller @ 0a2e7ba

Known Issues

- If you push an application with a specific buildpack set with the `buildpacks` property in the application manifest or with the `-b` flag, that application will fail to build with an error that only autodetection of buildpacks is supported. As a workaround, set `buildpacks: ~` in the application manifest or `-b null` on `cf push` to reset the app to use buildpack autodetection. If you only remove the field from the manifest or the flag from the `cf push` command, the app will continue to fail to build.
- If you change the application code so that the build process generates a different start command for the app, the app's start command is not updated, and the app may fail to start correctly. As a workaround, you can manually override the start command with the `command` property in the application manifest or with the `-c` flag on `cf push`.
- The organization manager role does not have permissions to create Cloud Foundry spaces. As a workaround, instead use the Cloud Foundry admin role to create spaces in organizations.

v1.0.0 Release

Release date: November 10, 2022

Features

Application management

- Application developers can use `cf push` to create, build, and run applications from source code.
- Application developers can use `cf stop` to stop an application without deleting it.
- Application developers can use `cf start` to start a stopped application.
- Application developers can use `cf restart` to restart an application.
- Application developers can use `cf restage` to restage an application.
- Application developers can use `cf delete` to delete an application.
- Application developers can use `cf apps` to list the applications in a Cloud Foundry space.
- Application developers can use `cf app` to describe an application, including the status and CPU and memory usage metrics for its instances.

- Application developers can use `cf buildpacks` to list the system buildpacks.
- Application developers can use `cf get-health-check` to get the health-check type of an application's process types.
- Application developers can use `cf set-health-check` to set the health-check type of an application's process types.
- Application processes with a route mapped to them have a port-based health-check by default.
- Application developers can use `cf env` to list the environment variables on an application.
- Application developers can use `cf set-env` to set an environment variable on an application.
- Application developers can use `cf unset-env` to remove an environment variable on an application.
- Application environment variables are included in the environment variables set during the application build process.
- Application developers can use `cf set-label` to set a label on an application.
- Application developers can use `cf unset-label` to unset a label on an application.
- Application developers can use `cf labels` to read labels on an application.
- Application developers can use CF API endpoints to read, set, and unset annotations on an application.
- Applications have a default memory limit of 1024M.

Application routes and domains

- Application developers can use `cf create-route` to create a route.
- Application developers can use `cf delete-route` to delete a route.
- Application developers can use `cf map-route` to map a route from an application.
- Application developers can use `cf unmap-route` to unmap a route from an application.
- Application developers can use `cf domains` to list the shared domains available for routes.
- Applications receive a HTTP route on `cf push` unless opting out with the `--no-route` flag.
- Application developers can obtain a randomly generated route for their app using the `--random-route` flag on `cf push` or the `random-route` key in the application manifest.
- Application Service Adapter validates that domains do not overlap.
- Application Service Adapter validates that each route has a non-empty hostname and has a fully qualified domain name that is a valid DNS name.
- Application Service Adapter validates that paths of routes are well-formed and less than 128 characters in length.

Application logs

- Application developers can see application staging logs during `cf push`.
- Application developers can use `cf logs` to stream logs from running applications.
- Application developers can use `cf logs --recent` to retrieve recent build and runtime logs for applications.

Application tasks

- Application developers can use `cf run-task` to run a one-off task for an application.
- Application developers can use `cf tasks` to list the one-off tasks for an application.
- Application developers can use `cf terminate-task` to cancel a running one-off task for an application.

Service management

- Application developers can use `cf create-user-provided-service` to create a user-provided service instance.
- Application developers can use `cf services` to list the user-provided service instances in the current CF space.
- Application developers can use `cf service` to describe a user-provided service instance.
- Application developers can use `cf delete-service` to delete a user-provided service instance.
- Application developers can use `cf bind-service` to bind a user-provided service instance to an application. After the app is restarted or restaged, the contains the credentials for the bound service.
- Application Service Adapter presents service credentials to bound applications both in the application's `VCAP_SERVICES` environment variable as filesystem projections under the service binding root directory.
- Application developers can use `cf unbind-service` to unbind a user-provided service instance from an application.

User authentication and authorization

- Application developers can use `cf login` to log into the Application Service Adapter with authentication information from their local kubeconfig file.
- The Application Service Adapter enforces authorization rules for API resources. A user must have an admin role or a Space Developer role to push apps and map routes.

System installation

- System operators can use the `tanzu` CLI to install Application Service Adapter as a Tanzu package.
- System operators can configure the Application Service Adapter system to trust container registries that have certificates signed by private certificate authorities.

- System operators can set replica counts and resource limits for each Application Service Adapter system component.
- System operators can set the expiry duration beyond which users of the Application Service Adapter API are notified to use shorter-lived certificates for authentication.
- System operators must specify a domain to use as the default shared domain for application routes.
- System operators can change the default shared domain for application routes after an initial installation.
- System operators can specify the name of the kpack ClusterBuilder to use to build application images, defaulting to `default`.
- System operators can configure Application Service Adapter to work with Kubernetes authentication proxies such as Pinniped.
- System operators can configure the certificate/private key pairs for the Application Service Adapter API and app ingress TLS using Kubernetes secrets instead of certificate and key contents.
- System operators can opt out of telemetry collection by creating a particular ConfigMap in the underlying Kubernetes cluster, identical to the one used to opt out of telemetry collection for Tanzu Application Platform.

Org and space management

- System operators can use `cf create-org` to create a Cloud Foundry org.
- System operators can use `cf create-space` to create a Cloud Foundry space.
- System operators can also create `CFOrg` and `CFSpace` custom resources in the Kubernetes API to create orgs and spaces.
- System operators can use `cf delete-space` to delete a Cloud Foundry space.
- System operators can use `cf delete-org` to delete a Cloud Foundry org.
- System operators can use `cf set-label` to set a label on an org or a space.
- System operators can use `cf unset-label` to unset a label on an org or a space.
- System operators can use `cf labels` to read labels on an org or a space.
- System operators can use CF API endpoints to read, set, and unset annotations on an org or a space.

System security

- App traffic runs over HTTPS instead of HTTP, with the terminating TLS certificate and private key specified at installation time.

Supply Chain Choreographer integration (experimental)

- **EXPERIMENTAL**: Platform operators can configure the Application Service Adapter system to build and run applications using the Workload resource from Tanzu Application Platform.

Note that this configuration is experimental and applies system-wide.

Components

This release contains the following components:

- cartographer-builder-runner @ 9ae3d6c
- Korifi @ v0.3.0
- tas-adapter-telemetry-controller @ 0a2e7ba

Known Issues

- If you push an application with a specific buildpack set with the `buildpacks` property in the application manifest or with the `-b` flag, that application will fail to build with an error that only autodetection of buildpacks is supported. As a workaround, set `buildpacks: ~` in the application manifest or `-b null` on `cf push` to reset the app to use buildpack autodetection. If you only remove the field from the manifest or the flag from the `cf push` command, the app will continue to fail to build.
- When deleting CF spaces or uninstalling Application Service Adapter, the underlying Kubernetes namespaces may not be deleted due to an issue with `ServiceBindingProjection` resource cleanup. As a workaround, you can manually remove the `finalizers` from the `ServiceBindingProjections` to allow namespace deletion to complete.
- If you change the application code so that the build process generates a different start command for the app, the app's start command is not updated, and the app may fail to start correctly. As a workaround, you can manually override the start command with the `command` property in the application manifest or with the `-c` flag on `cf push`.
- Application instance pods do not have the `CF_INSTANCE_GUID`, `CF_INSTANCE_INTERNAL_IP`, `CF_INSTANCE_IP`, or `POD_NAME` environment variables set when Application Service Adapter is configured to use the experimental integration with the Supply Chain Choreographer.
- The output of `cf logs` contains blank log lines interleaved with the log content from the appplication instances.
- The organization manager role does not have permissions to create Cloud Foundry spaces. As a workaround, instead use the Cloud Foundry admin role to create spaces in organizations.
- The values schema published in the Tanzu package for Application Service Adapter does not precisely match the values that the package installation recognizes. The published schema contained obsolete parameters for scaling the `kube_rbac_proxy` component and for enabling telemetry collection and omitted the `api_ingress.port` parameter.

Application Service Adapter reference architecture

The Application Service Adapter architecture described here gives you a path to create a production deployment of Application Service Adapter.

Your use case may warrant a different architecture, but the design decisions described here exemplify the main design issues in planning your Application Service Adapter environment. Understanding these decisions can help provide a rationale for any necessary deviation from this architecture.

- Tanzu Application Platform installation
- Adapter system requirements
- Application workload requirements

Tanzu Application Platform installation

Application Service Adapter is installed on top of the Tanzu Application Platform, and uses Tanzu Application Platform components to build and deploy workloads. Because both "build" and "run" steps are executed on the same cluster, you must install the required Tanzu Application Platform packages to a single cluster before installing the Application Service Adapter. This type of installation maps most closely to the "iterate cluster" described in the Tanzu Application Platform reference architecture.

Some packages included in an "iterate cluster" installation are optional when preparing a cluster for Application Service Adapter. Required packages are as follows:

```
buildservice.tanzu.vmware.com
cert-manager.tanzu.vmware.com
contour.tanzu.vmware.com
service-bindings.labs.vmware.com
tap.tanzu.vmware.com
tap-telemetry.tanzu.vmware.com
```

If you plan to enable the experimental Cartographer integration in the Application Service Adapter, the following packages are also required:

```
tekton.tanzu.vmware.com
cartographer.tanzu.vmware.com
ootb-templates.tanzu.vmware.com
controller.source.apps.tanzu.vmware.com
```

Adapter system requirements

Application Service Adapter installs a Cloud Foundry API server and a set of controller runtime components on top of the base Tanzu Application Platform installation. Although these components consume cluster resources, the recommended minimum sizing for a Tanzu Application Platform "iterate cluster" are sufficient to accommodate Application Service Adapter system components running with modest load.

Specifically, VMware recommends the following for an "iterate cluster":

- LoadBalancer for ingress controller (2 external IP addresses).
- Default storage class.
- At least 16 GB available memory that is allocatable across clusters, with at least 8 GB per node.
- Logging is enabled and targets the desired application logging platform.
- Monitoring is enabled and targets the desired application observability platform.
- Spread across three AZs for high availability.

Note

With increased API traffic or deployed object counts, Application Service Adapter components might need to be scaled as described in the scaling topic and might require additional cluster resources.

Application workload requirements

Because `cf push`ed applications run as pods on the same cluster as the Application Service Adapter itself, plan the capacity of your cluster to accommodate those applications.

Planning capacity for application workloads depends on the applications you plan to deploy and how they are scaled. Default memory, CPU, and disk allocations for each application instance are as follows:

```
limits:
  ephemeral-storage: 1Gi
  memory: 1Gi
requests:
  cpu: 100m
  ephemeral-storage: 1Gi
  memory: 1Gi
```

Planning total capacity on your cluster requires understanding the size of your planned applications, and then adding up those resources to ensure that you have sufficient resources on your cluster.

Prerequisites to install Application Service Adapter

To install Application Service Adapter, you need to first complete the prerequisites listed here.

Application Service Adapter requires using Cloud Foundry command-line interface (cf CLI) v8.1.0 or later. VMware recommends using v8.5.0 or later. For more information, see the cf CLI repository on GitHub.

Kubernetes cluster and container image registry

To install the Application Service Adapter, you need:

- Admin access to a Kubernetes cluster that meets the same requirements as the version of Tanzu Application Platform you have installed, either v1.2, v1.3, or v1.4. See Kubernetes cluster requirements under *Prerequisites*.
    - As of Tanzu Application Platform v1.3, Kubernetes v1.22, v1.23, or v1.24 is required. There are additional requirements for some implementations. For example, Amazon Elastic Kubernetes Service (EKS) requires containerd as the Container Runtime Interface (CRI) among with other requirements.
- A container image registry. See VMware Tanzu Network and container image registry requirements in *Prerequisites*. The Application Service Adapter does not support Amazon's Elastic Container Registry (ECR).

Required installation tools

The following tools must be installed in the workstation environment in which you intend to perform the installation:

- The Kubernetes CLI (kubectl) v1.22 or v1.23.
- Tanzu CLI and its plug-ins. See Install or update the Tanzu CLI and plugins in *Accepting Tanzu Application Platform EULAs and installing Tanzu CLI*.

After you install the Tanzu CLI, run `tanzu plugin list` to verify that the required `package` and `secret` plug-ins are installed. To install these plug-ins, run:

```
tanzu plugin install --local cli package
tanzu plugin install --local cli secret
```

Required components

The following dependencies must be installed to the target Kubernetes cluster before installing Application Service Adapter. If you completed an installation of the `full` profile or the `iterate` profile of Tanzu Application Platform v1.2 in your target Kubernetes cluster, these dependencies are already present.

- Cluster Essentials. See Deploying Cluster Essentials v1.3.

  > **Note**
  >
  > If you are operating a Tanzu Kubernetes Grid cluster, the Cluster Essentials are already installed.

- cert-manager v1.5.3 or later for managing internal certificates.
  - If you installed Tanzu Application Platform v1.3 with a profile, this package is installed.
  - If you installed Tanzu Application Platform v1.3 without using a profile, see Install cert-manager in *Install cert-manager, Contour*.
- Contour v1.18.2 or later for ingress control.
  - If you installed Tanzu Application Platform v1.3 with a profile, this package is installed.
  - If you installed Tanzu Application Platform v1.3 without using a profile, see Install Contour in *Install cert-manager, Contour*.

  > **Note**
  >
  > You must configure Contour's ingress to provision a LoadBalancer. See the default configuration in Install your Tanzu Application Platform profile in *Installing Tanzu Application Platform Package and Profiles*.

- Service Bindings v0.7.2 or later.
  - If you installed Tanzu Application Platform v1.3 with the `full`, `iterate`, or `run` profile, this package is installed.
  - If you installed Tanzu Application Platform v1.3 without using a profile, see Install Service Bindings.
- Tanzu Build Service v1.6.1 or later for building images.
  - If you installed Tanzu Application Platform v1.3 with the `full`, `iterate`, or `build` profile, this package is installed.
  - If you installed Tanzu Application Platform v1.3 without using a profile, see Installing Tanzu Build Service.

Required components for experimental Cartographer integration

The following dependencies are required to be installed to the target Kubernetes cluster to opt into using the experimental Cartographer integration. If you installed Tanzu Application Platform v1.3 with the `full`, `iterate`, or `build` profile, these dependencies are already present.

- Out of the Box Templates v0.8.1 or later.
  - If you installed Tanzu Application Platform v1.3 without using a profile, see Install Out of the Box Templates.
- Supply Chain Choreographer v0.4.0 or later.
  - If you installed Tanzu Application Platform v1.2 without using a profile, see Install Supply Chain Choreographer.
- Source Controller v0.4.1 or later.
  - If you installed Tanzu Application Platform v1.2 without using a profile, see Install Source Controller.
- Tekton v0.33.5 or later.
  - If you installed Tanzu Application Platform v1.2 without using a profile, see Install Tekton.

Recommended components

VMware recommends installing the following dependencies to the target Kubernetes cluster.

- Kubernetes Metrics Server v0.4.0 or later for app instance resource metrics.

  Note

  Many Kubernetes distributions automatically come with the Metrics Server installed. If the API resources in your target cluster include the `PodMetrics` Kind in the `metrics.k8s.io` API group, the Metrics Server is already present.

After you installed these prerequisites, proceed to Installing Application Service Adapter.

Installing Application Service Adapter

You can install and configure Application Service Adapter for VMware Tanzu Application Platform by following the steps in this topic.

After you complete the steps in Installing Prerequisites, set the Kubernetes context to the cluster where you installed kpack and Contour.

Install the package repository

To install Application Service Adapter:

1. Set up environment variables for the installation:

   ```
   export TAS_ADAPTER_VERSION=VERSION-NUMBER
   ```

   Where `VERSION-NUMBER` is the version of Application Service Adapter you want to install. For example, `1.0.0`.

2. Verify that the `tap-install` namespace exists in your cluster.

   ```
   kubectl get ns tap-install
   ```

   The output lists the status of the `tap-install` namespace:

   ```
   NAME          STATUS   AGE
   tap-install   Active   2d
   ```

3. Create a registry secret to store your VMware Tanzu Network credentials in the `tap-install` namespace. These are required so that the Kubernetes cluster can pull images for the Application Service Adapter system from the VMware Tanzu Network registry.

   ```
   tanzu secret registry add tanzunet-tas-adapter-registry \
     --username "TANZU-NET-USERNAME" \
     --password "TANZU-NET-PASSWORD" \
     --server registry.tanzu.vmware.com \
     --export-to-all-namespaces \
     --yes \
     --namespace tap-install
   ```

   Where:

   - `TANZU-NET-USERNAME` is your user name on VMware Tanzu Network.
   - `TANZU-NET-PASSWORD` the password for your user name on VMware Tanzu Network.

4. Add the Application Service Adapter package repository to the cluster.

```
tanzu package repository add tas-adapter-repository \
  --url registry.tanzu.vmware.com/app-service-adapter/tas-adapter-package-rep
o:${TAS_ADAPTER_VERSION} \
  --namespace tap-install
```

5. Verify that the package repository contains the Application Service Adapter package.

   ```
   tanzu package available list \
     --namespace tap-install
   ```

   The output includes the Application Service Adapter package:

   ```
   NAME                                        DISPLAY-NAME                 SHOR
   T-DESCRIPTION                                                          LATEST-VERSION
   ...
   application-service-adapter.tanzu.vmware.com  Application Service Adapter  Appl
   ication Service Adapter for VMware Tanzu® Application Platform  1.0.0
   ...
   ```

6. List the installation settings for the `application-service-adapter` package.

   ```
   tanzu package available get application-service-adapter.tanzu.vmware.com/${TAS_
   ADAPTER_VERSION} --values-schema --namespace tap-install
   ```

   It should output a list of settings similar to:

   ```
   | Retrieving package details for application-service-adapter.tanzu.vmware.com/
   1.0.0...
     KEY                            DEFAULT  TYPE     DESCRIPTION
     api_auth_proxy.ca_cert.data             string   TLS CA certificate of your clus
   ter's auth proxy
     api_auth_proxy.host                     string   FQDN of your cluster's auth pro
   xy
     api_ingress.fqdn                        string   FQDN used to access the CF API
     api_ingress.tls.secret_name             string   Name of the secret containing t
   he TLS certificate for the CF API (PEM format)
     api_ingress.tls.namespace               string   Namespace containing the CF API
   TLS secret
     app_ingress.default_domain              string   Default application domain
     app_ingress.tls.secret_name             string   Name of the secret containing t
   he TLS certificate for the default application domain (PEM format)
     app_ingress.tls.namespace               string   Namespace containing the defaul
   t application domain TLS secret
     app_registry.path.droplets              string   Container registry repository w
   here staged, runnable app images (Droplets) will be stored
     app_registry.path.packages              string   Container registry repository w
   here uploaded app source code (Packages) will be stored
     kpack_clusterbuilder_name  default  string   Name of the kpack cluster builde
   r to use for staging
     ...
   ```

Configure the installation settings

To configure the installation settings:

1. If you do not already have a secret containing a certificate and private key pair for HTTPS ingress to the Application Service Adapter API:

   - If you have a certificate and private key pair, create a secret containing them:

     ```
     kubectl create namespace API-TLS-SECRET-NAMESPACE
     kubectl create secret tls API-TLS-SECRET-NAME \
        --cert=tls.crt \
        --key=tls.key \
        --namespace API-TLS-SECRET-NAMESPACE
     ```

   - If you do not have a certificate and private key pair, you can use cert-manager to generate a secret containing a self-signed certificate in the cert-manager documentation:

     ```
     kubectl apply -f - <<EOF
     ---
     apiVersion: cert-manager.io/v1
     kind: Issuer
     metadata:
        name: selfsigned-issuer
        namespace: API-TLS-SECRET-NAMESPACE
     spec:
        selfSigned: {}
     ---
     apiVersion: cert-manager.io/v1
     kind: Certificate
     metadata:
        name: api-selfsigned-certificate
        namespace: API-TLS-SECRET-NAMESPACE
     spec:
        commonName: API-FQDN
        dnsNames:
          - API-FQDN
        issuerRef:
          name: selfsigned-issuer
        privateKey:
          algorithm: RSA
        secretName: API-TLS-SECRET-NAME
        usages:
        - server auth
        - client auth
     EOF
     ```

2. If you do not already have a secret containing a wildcard certificate and private key pair for HTTPS application ingress:

   - If you have a wildcard certificate and private key pair, create a secret containing them:

     ```
     kubectl create namespace APP-TLS-SECRET-NAMESPACE
     kubectl create secret tls APP-TLS-SECRET-NAME \
        --cert=tls.crt \
        --key=tls.key \
        --namespace APP-TLS-SECRET-NAMESPACE
     ```

- If you do not have a wildcard certificate and private key pair, you can use cert-manager to generate a Secret containing a self-signed wildcard certificate in the cert-manager documentation:

  ```
  kubectl apply -f - <<EOF
  ---
  apiVersion: cert-manager.io/v1
  kind: Issuer
  metadata:
     name: selfsigned-issuer
     namespace: APP-TLS-SECRET-NAMESPACE
  spec:
     selfSigned: {}
  ---
  apiVersion: cert-manager.io/v1
  kind: Certificate
  metadata:
     name: app-domain-selfsigned-certificate
     namespace: APP-TLS-SECRET-NAMESPACE
  spec:
     commonName: *.DEFAULT-APP-DOMAIN
     dnsNames:
       - *.DEFAULT-APP-DOMAIN
     issuerRef:
       name: selfsigned-issuer
     privateKey:
       algorithm: RSA
     secretName: APP-TLS-SECRET-NAME
     usages:
     - server auth
     - client auth
  EOF
  ```

3. If you do not already have a secret containing the host name, user name, and password for your application image registry, create one:

   ```
   kubectl create namespace APP-REGISTRY-CREDENTIALS-SECRET-NAMESPACE
   kubectl create secret docker-registry APP-REGISTRY-CREDENTIALS-SECRET-NAME \
     --docker-server=APP-REGISTRY-SERVER \
     --docker-username=APP-REGISTRY-USERNAME \
     --docker-password=$(cat /path/to/APP-REGISTRY-PASSWORD) \
     --namespace APP-REGISTRY-CREDENTIALS-SECRET-NAMESPACE
   ```

   Where:

   - `APP-REGISTRY-SERVER` is the address of the registry used for app packages and droplets. This value is the same as the server name in a `dockerconfigjson` Kubernetes secret. For example:
     - Harbor has the form `my-harbor.io`.
     - Docker Hub the form `https://index.docker.io/v1/`.
     - Google Container Registry has the form `gcr.io`.

4. Create a `SecretExport` to allow Application Service Adapter to copy the application image registry credentials secret into the `cf` namespace.

```
kubectl apply -f - <<EOF
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: APP-REGISTRY-CREDENTIALS-SECRET-NAME
  namespace: APP-REGISTRY-CREDENTIALS-SECRET-NAMESPACE
spec:
  toNamespace: cf
EOF
```

5. Create a `tas-adapter-values.yaml` file with the installation settings that you want, following the schema specified for the package.

   The following values are required:

```
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not
set to the boolean value true. Not a string.
api_ingress:
  fqdn: "API-FQDN"
  tls:
    secret_name: API-TLS-SECRET-NAME
    namespace: API-TLS-SECRET-NAMESPACE
app_ingress:
  default_domain: "DEFAULT-APP-DOMAIN"
  tls:
    secret_name: APP-TLS-SECRET-NAME
    namespace: APP-TLS-SECRET-NAMESPACE
app_registry:
  credentials:
    secret_name: "APP-REGISTRY-CREDENTIALS-SECRET-NAME"
    namespace: "APP-REGISTRY-CREDENTIALS-SECRET-NAMESPACE"
  hostname: "APP-REGISTRY-HOSTNAME"
  path:
    droplets: "APP-REGISTRY-PATH-DROPLETS"
    packages: "APP-REGISTRY-PATH-PACKAGES"
```

   Where:

   - `API-FQDN` is the fully qualified domain name (FQDN) that you want to use for the Application Service Adapter API. Example: `api.example.com`
   - `API-TLS-SECRET-NAME` is the `kubernetes.io/tls` secret containing the PEM-encoded public certificate for the Application Service Adapter API.
   - `API-TLS-SECRET-NAMESPACE` is the namespace containing the API TLS secret.
   - `DEFAULT-APP-DOMAIN` is the domain that you want to use for automatically configured application routes. Example: `apps.example.com`.
   - `APP-TLS-SECRET-NAME` is the `kubernetes.io/tls` secret containing the PEM-encoded public certificate for applications deployed using the Application Service Adapter.
   - `APP-TLS-SECRET-NAMESPACE` is the namespace containing the application TLS secret.
   - `APP-REGISTRY-CREDENTIALS-SECRET-NAME` is the `kubernetes.io/dockerconfigjson` secret containing the host, user name, and password for the application image registry.

- `APP-REGISTRY-CREDENTIALS-SECRET-NAMESPACE` is the namespace containing the application image registry secret.
- `APP-REGISTRY-HOSTNAME` is the host name of the registry used for app packages and droplets. For example:
  - Harbor has the form `hostname: "my-harbor.io"`
  - Docker Hub has the form `hostname: "index.docker.io"`
  - Google Container Registry has the form `hostname: "gcr.io"`
- `APP-REGISTRY-PATH-DROPLETS` is the path to the directory or project in the app registry where Application Service Adapter uploads droplets, such as runnable application images. This value does not include the registry host name itself. Examples:
  - Harbor has the form `droplets: "project-name/my-repo-name"`
  - Docker Hub has the form `droplets: "my-dockerhub-username"`
  - Google Container Registry has the form `droplets: "project-id/my-repo-name"`
- `APP-REGISTRY-PATH-PACKAGES` is the is the path to the directory or project in the app registry where Application Service Adapter uploads packages, such as application source code. This value does not include the registry host name itself. Examples:
  - Harbor has the form `packages: "project-name/my-repo-name"`
  - Docker Hub has the form `packages: "my-dockerhub-username"`
  - Google Container Registry has the form `packages: "project-id/my-repo-name"`

The following values are optional but recommended:

```
admin:
  users:
  - "ADMIN-USERNAME"
  ...
```

Where:

- `ADMIN-USERNAME` is the name of an existing user in the Kubernetes cluster to whom to grant system admin privileges. You can specify as many users as you want, one per line. These names are identifiers for Kubernetes user accounts, not Kubernetes service accounts.
  - For Amazon EKS, see the AWS IAM user management for EKS section of the User Management topic for information on additional required cluster configuration to map AWS IAM users and roles to Kubernetes roles.
  - For clusters configured to use authentication proxies such as Pinniped, you can authenticate to the cluster and use the output of `cf curl /whoami` to see the user account name to provide.

Note

These user names are the ones that Kubernetes recognizes as user identifiers in the subject section of its RBAC resources, such as `RoleBindings`, and may differ from the names of the user entries in your local Kubeconfig file. If you are not certain of this user name, you can leave this entry empty for the initial installation. After completing the installation and logging in with the cf CLI, use the `cf curl /whoami` command to confirm the user name and then update the installation with the correct name value. For more information about user subject names in Kubernetes, see the Referring to subjects section of *Using RBAC Authorization* and the Authenticating topic in the Kubernetes project documentation.

See additional optional values in the following example. For more information on optional values, see the Tanzu CLI output.

```
api_auth_proxy:
  ca_cert:
    data: |
      API-AUTH-PROXY-TLS-CRT
  host: "API-AUTH-PROXY-FQDN"
api_ingress:
  port: "API-PORT"
app_registry:
  ca_cert:
    data: |
      PEM-ENCODED-CERTIFICATE-CONTENTS
experimental_use_cartographer: FALSE-OR-TRUE-VALUE
kpack_clusterbuilder_name: "KPACK-CLUSTER-BUILDER-NAME"
scaling:
  korifi_api:
    limits:
      cpu: "API-CPU-LIMIT"
      memory: "API-MEMORY-LIMIT"
    requests:
      cpu: "API-CPU-REQUEST"
      memory: "API-MEMORY-REQUEST"
    replicas: API-REPLICA-COUNT
  korifi_controllers:
    ... #! scaling keys are the same as above
  korifi_job_task_runner:
    ... #! scaling keys are the same as above
  korifi_kpack_image_builder:
    ... #! scaling keys are the same as above
  korifi_statefulset_runner:
    ... #! scaling keys are the same as above
  cartographer_builder_runner:
    ... #! scaling keys are the same as above
  telemetry_informer:
    ... #! scaling keys are the same as above
shared:
  kubernetes_distribution: KUBERNETES-DISTRIBUTION
telemetry:
  heartbeat_interval: TELEMETRY-HEARTBEAT-INTERVAL
user_certificate_expiration_warning_duration: "USER-CERT-EXPIRY-WARNING-DURATIO
N"
```

Where:

- `API-AUTH-PROXY-TLS-CRT` is the CA certificate from the authentication proxy running along side your Kubernetes cluster.
- `API-AUTH-PROXY-FQDN` is the FQDN for the authentication proxy running along side your Kubernetes cluster.
- `API-PORT` is the port number which clients should use to connect to the Application Service Adapter API, and which the API will include in URLs that direct back to itself. When set to `0` or left unset, no port is included in those URLs, and clients should connect to port 443, the standard port for HTTPS traffic.
- `PEM-ENCODED-CERTIFICATE-CONTENTS` is a PEM encoded multiline string containing the certificate authority (CA) certificate.
  - The value must be inserted into your values file as a YAML multiline string with a block scalar literal.
- `KPACK-CLUSTER-BUILDER-NAME` is the name of the kpack cluster builder to use for staging. Tanzu Build Service provides two cluster builders named `base` and `default`. To create your own builder, see Managing Builders in the Tanzu Build Service documentation, and update this setting with the corresponding builder name.
- `USER-CERT-EXPIRY-WARNING-DURATION` is the recommended duration beyond which user are warned to use short-lived certificates for authentication. Default is 168 hours.
- `API-CPU-LIMIT` is the CPU resource limit for the pods that you want in the specified deployment. Default is 1 CPU.
- `API-MEMORY-LIMIT` is the memory resource limit that you want for the pods in the specified deployment. Default is 1000Mi.
- `API-CPU-REQUEST` is the CPU resource request that you want for the pods in the specified deployment. Default is 50m.
- `API-MEMORY-REQUEST` is the memory resource request that you want for the pods in the specified deployment. Default is 100Mi.
- `API-REPLICA-COUNT` is the number of replicas that you want for the specified deployment. Default is 1.
- `KUBERNETES-DISTRIBUTION` is the name of the Kubernetes distribution. Defaults to "".
  - Set "openshift" as the value when installing on an OpenShift environment.
  - Leave it unset for all other distributions.
- `TELEMETRY-HEARTBEAT-INTERVAL` is how often telemetry data is sent to VMware. Default is every 24 hours.

The `requests` and `limits` text boxes map directly to the resource requests and limits text boxes on the Kubernetes containers for these system components. For more information, see Resource requests and limits of Pod and container in the Kubernetes documentation.

Opting out of telemetry reporting

By default, when you install Application Service Adapter, you opt into telemetry collection. To deactivate telemetry collection, complete the following instructions:

1. Ensure that your Kubernetes context is pointing to the cluster where Application Service Adapter is installed.
2. Run the following kubectl command:

```
kubectl apply -f - <<EOF
---
apiVersion: v1
kind: Namespace
metadata:
  name: vmware-system-telemetry
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: vmware-system-telemetry
  name: vmware-telemetry-cluster-ceip
data:
  level: disabled
EOF
```

Your Application Service Adapter deployment no longer emits telemetry, and you are opted out of the VMware Customer Experience Improvement Program.

(Optional) Configure a Registry With a Custom Certificate Authority

Your Kubernetes cluster nodes and the Tanzu Build Service component of Tanzu Application Platform must also both be configured to trust this Certificate Authority for the registry.

To configure Application Service Adapter to trust a registry that has a custom or self-signed certificate authority:

1. Set the value of the `app_registry_credentials.ca_cert_data` property in the `tas-adapter-values.yaml` file with the PEM encoded certificate for the registry's Certificate Authority.

(Optional) Configure the Experimental Cartographer Integration

Opting into the experimental Cartographer integration requires a larger set of Tanzu Application Platform packages to be installed. See Required components for experimental Cartographer integration in *Install Prerequisites*.

To configure the experimental Cartographer integration:

1. Set the value of the `experimental_use_cartographer` property in the `tas-adapter-values.yaml file.

Install Application Service Adapter

To install Application Service Adapter:

1. Install Application Service Adapter to the cluster.

```
tanzu package install tas-adapter \
  -p application-service-adapter.tanzu.vmware.com \
  --version "${TAS_ADAPTER_VERSION}" \
  --values-file tas-adapter-values.yaml \
  --namespace tap-install
```

2. Verify that the package install was successful.

```
tanzu package installed get tas-adapter \
  --namespace tap-install
```

The following is an example output:

```
| Retrieving installation details for tas-adapter...
NAME:                    tas-adapter
PACKAGE-NAME:            application-service-adapter.tanzu.vmware.com
PACKAGE-VERSION:         1.0.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Configure DNS for Application Service Adapter

To configure DNS for Application Service Adapter:

1. Determine the external IP address to use for ingress to your cluster. This step varies depending on the IaaS used to provision your cluster.

   For clusters that support LoadBalancer services, you can obtain the external IP address of the LoadBalancer Service that is associated with Contour's Envoy proxy. The Namespace for this service is typically either `tanzu-system-ingress` or `projectcontour` depending on how Contour was installed.

```
kubectl -n tanzu-system-ingress get service envoy -ojsonpath='{.status.loadBala
ncer.ingress[*].ip}'
```

> **Note**
>
> If you are using a cluster deployed on AWS, your LoadBalancer has a DNS name instead of an IP address.

2. Create an A record in your DNS zone that resolves the configured API FQDN to the external IP address from step 1. This step varies depending on your DNS provider.

> **Note**
>
> If you are using a cluster deployed on AWS, create a CNAME record that resolves to the DNS name of the load balancer instead of an A record.

3. Create a wildcard A record in your DNS zone that resolves all sub-domains of the configured application domain to the external IP address from step 1. This step varies depending on your DNS provider.

4. Verify that the Contour HTTPProxy for the API endpoint is valid.

   ```
   kubectl -n tas-adapter-system get httpproxy korifi-api-proxy
   ```

   The following is an example output:

   ```
   NAME                FQDN       TLS SECRET                STATUS    STATUS DESCRIP
   TION
   korifi-api-proxy    API-FQDN   korifi-api-ingress-cert   valid     Valid HTTPProx
   y
   ```

Log in with a system admin user

After you install the Cloud Foundry command-line interface (cf CLI), log in to Application Service Adapter with one of the system admin users you configured in the `admin.users` value:

1. Target the cf CLI at the API endpoint.

   ```
   cf api API-FQDN --skip-ssl-validation
   ```

   Where `API-FQDN` is the fully qualified domain name (FQDN) for the Application Service Adapter API.

   Note

   If you configured the Application Service Adapter with a globally trusted certificate during installation, you can omit the `--skip-ssl-validation` flag.

2. Log in with the cf CLI.

   ```
   cf login
   ```

   The cf CLI detects the user authentication entries in your local Kubeconfig file and presents them for you to select one interactively. Select an entry corresponding to one of the users you configured in the list in the `admin.users` value.

3. Use the `cf curl` command to verify the subject name of the logged-in user.

   ```
   cf curl /whoami
   ```

   The output looks like the following:

   ```
   {"name":"my_user@example.com","kind":"User"}
   ```

   The value of the `name` text box in the response is the subject name of the user, and matches the name configured in `admin.users`.

The `kind` text box in the output must have the value `User`. If it is some other value, such as `ServiceAccount`, log in to the Application Service Adapter with an account for a user in the Kubernetes cluster.

To test Application Service Adapter, continue to Getting Started.

Install Application Service Adapter to air-gapped environments

You can install Application Service Adapter to a K8s cluster and registry that is air-gapped from outside traffic. This topic has the steps you need to take.

For instructions on how to install prerequisites in air-gapped environments, see:

- Tanzu Application Platform
- Tanzu Build Service

Relocate images to a registry (air-gapped)

This procedure relocates images from VMware Tanzu Network registry to an internal container image registry that is available to the air-gapped environment through a local machine. The local machine must have access to the air-gapped environment.

1. Set up environment variables for the installation:

   ```
   export TAS_ADAPTER_VERSION=VERSION-NUMBER
   ```

   Where `VERSION-NUMBER` is the version of Application Service Adapter you want to install. For example, `1.0.0`.

2. Log in to VMware Tanzu Network registry with your VMware Tanzu Network credentials:

   ```
   docker login registry.tanzu.vmware.com
   ```

3. Copy the Application Service Adapter bundle to a tarball with the Carvel `imgpkg` tool by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/app-service-adapter/tas-adapter-packa
   ge-repo:${TAS_ADAPTER_VERSION} --to-tar tas-adapter-package-repo.tar
   ```

4. Move the tarball file `tas-adapter-package-repo.tar` to the local machine that has access to the air-gapped environment.

5. Log in to the internal image registry from the local machine:

   ```
   docker login INTERNAL-REGISTRY
   ```

   Where `INTERNAL-REGISTRY` is the name of your internal image registry.

6. Unpackage the images from the tarball to the internal registry:

```
 imgpkg copy --tar tas-adapter-package-repo.tar --to-repo=INTERNAL-REGISTRY /ta
s-adapter-package-repo
```

Install the package repository

After the images are relocated:

1. Verify that the `tap-install` namespace exists in your cluster.

   ```
   kubectl get ns tap-install
   ```

   The output lists the status of the `tap-install` namespace:

   ```
   NAME          STATUS   AGE
   tap-install   Active   2d
   ```

2. Create a registry secret to store your registry credentials in the `tap-install` namespace. These are required so that the Kubernetes cluster can pull images for the Application Service Adapter system from the internal registry.

   ```
   tanzu secret registry add internal-tas-adapter-registry \
     --username INTERNAL-REGISTRY-USERNAME \
     --password INTERNAL-REGISTRY-PASSWORD \
     --server INTERNAL-REGISTRY \
     --export-to-all-namespaces \
     --yes \
     --namespace tap-install
   ```

   Where `INTERNAL-REGISTRY-USERNAME` and `INTERNAL-REGISTRY-PASSWORD` are your credentials for `INTERNAL-REGISTRY`.

3. Add the Application Service Adapter package repository to the cluster.

   ```
   tanzu package repository add tas-adapter-repository \
     --url <INTERNAL-REGISTRY>/tas-adapter-package-repo:${TAS_ADAPTER_VERSION} \
     --namespace tap-install
   ```

4. Verify that the package repository contains the Application Service Adapter package.

   ```
   tanzu package available list \
     --namespace tap-install
   ```

   The output includes the Application Service Adapter package:

   ```
   NAME                                           DISPLAY-NAME                 SHOR
   T-DESCRIPTION                                                  LATEST-VERSION
   ...
   application-service-adapter.tanzu.vmware.com   Application Service Adapter   Appl
   ication Service Adapter for VMware Tanzu Application Platform  1.0.0
   ...
   ```

5. List the installation settings for the `application-service-adapter` package.

```
tanzu package available get application-service-adapter.tanzu.vmware.com/${TAS_
ADAPTER_VERSION} --values-schema --namespace tap-install
```

It should output a list of settings similar to:

```
| Retrieving package details for application-service-adapter.tanzu.vmware.com/
1.0.0...
  KEY                          DEFAULT  TYPE     DESCRIPTION
  api_auth_proxy.ca_cert.data           string   TLS CA certificate of your clus
ter's auth proxy
  api_auth_proxy.host                   string   FQDN of your cluster's auth pro
xy
  api_ingress.fqdn                      string   FQDN used to access the CF API
  api_ingress.tls.crt                   string   TLS certificate for the CF API
(PEM format)
  api_ingress.tls.key                   string   TLS private key for the CF API
(PEM format)
  app_ingress.default_domain            string   Default application domain
  app_ingress.tls.crt                   string   TLS certificate for the default
application domain (PEM format)
  app_ingress.tls.key                   string   TLS private key for the default
application domain (PEM format)
  app_registry.path.droplets            string   Container registry repository w
here staged, runnable app images (Droplets) will be stored
  app_registry.path.packages            string   Container registry repository w
here uploaded app source code (Packages) will be stored
  kpack_clusterbuilder_name  default    string   Name of the kpack cluster build
er to use for staging
  ...
```

For installation and configuring instructions, see the install guide.

Get started with Application Service Adapter

This get-started walkthrough gives you hands-on experience of Application Service Adapter: creating an org and space, deploying an app, and more.

Create orgs and spaces

You can use `cf create-org` and `cf create-space` the same way that you do with Tanzu Application Service for VMs. Under the hood, these commands create a Kubernetes namespace for each org and each space and propagate required resources into the namespace.

To create orgs and spaces:

1. Create the Cloud Foundry org and space.

   ```
   cf create-org ORG-NAME
   cf target -o ORG-NAME
   cf create-space SPACE-NAME
   cf target -s SPACE-NAME
   ```

   Where:

   - `ORG-NAME` is the name of the org you want to create.
   - `SPACE-NAME` is the name of the space you want to create.

2. (Optional) Assign the SpaceDeveloper role to other users in the Kubernetes cluster.

   ```
   cf set-space-role USER-NAME ORG-NAME SPACE-NAME SpaceDeveloper
   ```

   Where `USER-NAME` is the name of another user in the Kubernetes cluster.

Deploy a sample app

Use the cf CLI to deploy a sample app to the Application Service Adapter installation.

```
cf push APP-NAME
```

Where `APP-NAME` is the name of your app.

Route to an app

Applications automatically receive a default HTTP route unless pushed with the `--no-route` flag. This default route uses the name of the app as the route host name. To configure additional routes for the app that you pushed, use the cf CLI to map a route to your app.

```
cf map-route APP-NAME apps.example.com --hostname my-app
```

Create and bind to a user-provided service instance

Service credentials are provided to apps through user-provided service instances. See User-Provided Service Instances in the Cloud Foundry documentation.

To create and bind user-provided service instances, do the following:

1. Create a user-provided service instance containing the credentials necessary for accessing your service:

   ```
   cf create-user-provided-service SERVICE-INSTANCE-NAME -p '{"credential-name":
   "credential-value"}'
   ```

   Where `SERVICE-INSTANCE-NAME` is the name of your service instance.
2. Bind the service instance to your app:

   ```
   cf bind-service APP-NAME SERVICE-INSTANCE-NAME
   ```
3. Restart (or restage if a buildpack relies on the service) the app to make the service credentials available:

   ```
   cf restart APP-NAME
   ```

User-provided service instance credentials is provided to the app and staging tasks in two ways to support both existing TAS applications and next-generation frameworks, such as Spring Cloud Bindings:

- As part of the traditional Cloud Foundry VCAP_SERVICES environment variable
- As volume mounted secrets in accordance with the Service Bindings for Kubernetes specification
  - This workload projection handles the Service Bindings Package from Tanzu Application Platform

Use Application Service Adapter to push the spring-music app

This how-to uses Application Service Adapter to push the spring-music app and bind it to a MySQL-compatible database represented as a CF service instance.

See spring-music sample Java application.

Prerequisites

You must have a Java version between 8 and 17 installed to your local workstation.

Clone and prepare the application locally

1. From the command line, clone the spring-music sample application Git repository to a directory on your local workstation:

   ```
   git clone https://github.com/cloudfoundry-samples/spring-music
   ```

2. Change into the root directory of the cloned repository:

   ```
   cd spring-music
   ```

3. Build a runnable Spring Boot JAR file for the application:

   ```
   ./gradlew clean assemble
   ```

Push the app without persistent storage

Ensure you are logged into the Application Service Adapter environment you intend to use and that you have targeted the desired Cloud Foundry org and space. You can use `cf target` to verify this context.

1. Use the cf CLI to push the application to your Application Service Adapter environment:

   ```
   cf push
   ```

   You should see output similar to the following:

   ```
   Pushing app spring-music to org o / space s as cf-admin...
   Applying manifest file /Users/tanzu/workspace/spring-music/manifest.yml...

   Updating with these attributes...
     applications:
     - name: spring-music
       path: /Users/tanzu/workspace/spring-music/build/libs/spring-music-1.0.jar
       memory: 1G
       random-route: true
       env:
         JBP_CONFIG_SPRING_AUTO_RECONFIGURATION: '{enabled: false}'
         SPRING_PROFILES_ACTIVE: http2
   Manifest applied
   Packaging files to upload...
   Uploading files...
    52.64 MiB / 52.64 MiB [==========================================] 100.00% 31s

   Waiting for API to complete processing files...

   Staging app and tracing logs...

      Build reason(s): CONFIG
   ...

   Waiting for app spring-music to start...

   Instances starting...
   ```

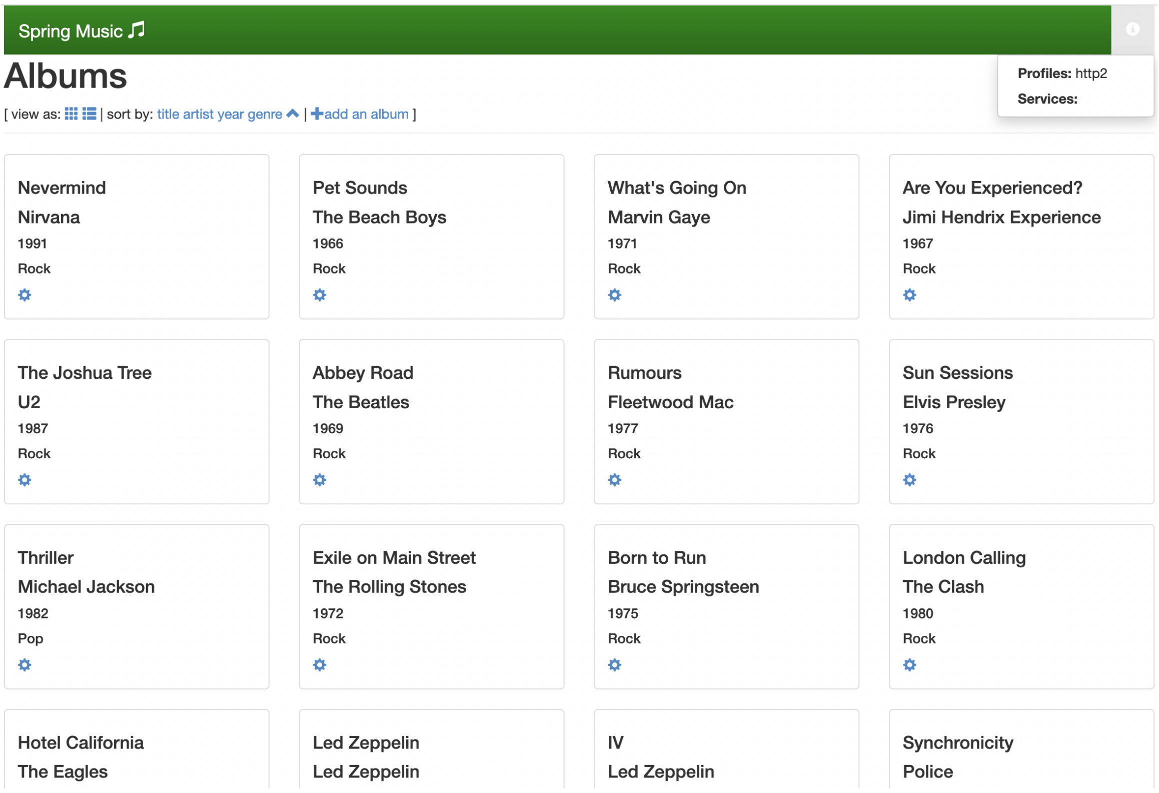
```
name:              spring-music
requested state:   started
routes:            spring-music-stellar-sitatunga-cz.apps.example.com
last uploaded:     Tue 08 Nov 16:24:36 PST 2022
stack:             io.buildpacks.stacks.bionic
buildpacks:

type:            web
sidecars:
instances:       1/1
memory usage:    1024M
start command:   java "org.springframework.boot.loader.JarLauncher"
     state     since                  cpu    memory   disk     logging      det
ails
#0   running   2022-11-09T00:25:33Z   0.0%   0 of 0   0 of 0   0/s of 0/s

type:            executable-jar
sidecars:
instances:       0/0
memory usage:    1024M
start command:   java "org.springframework.boot.loader.JarLauncher"
There are no running instances of this process.

type:            task
sidecars:
instances:       0/0
memory usage:    1024M
start command:   java "org.springframework.boot.loader.JarLauncher"
There are no running instances of this process.
```

2. The manifest for the spring-music application assigns it a random route by default, which is present in the `routes` field of the `cf push` output. Navigate to the route URL in your browser and verify that it serves requests correctly. You should see a webpage similar to the following:


Spring Music

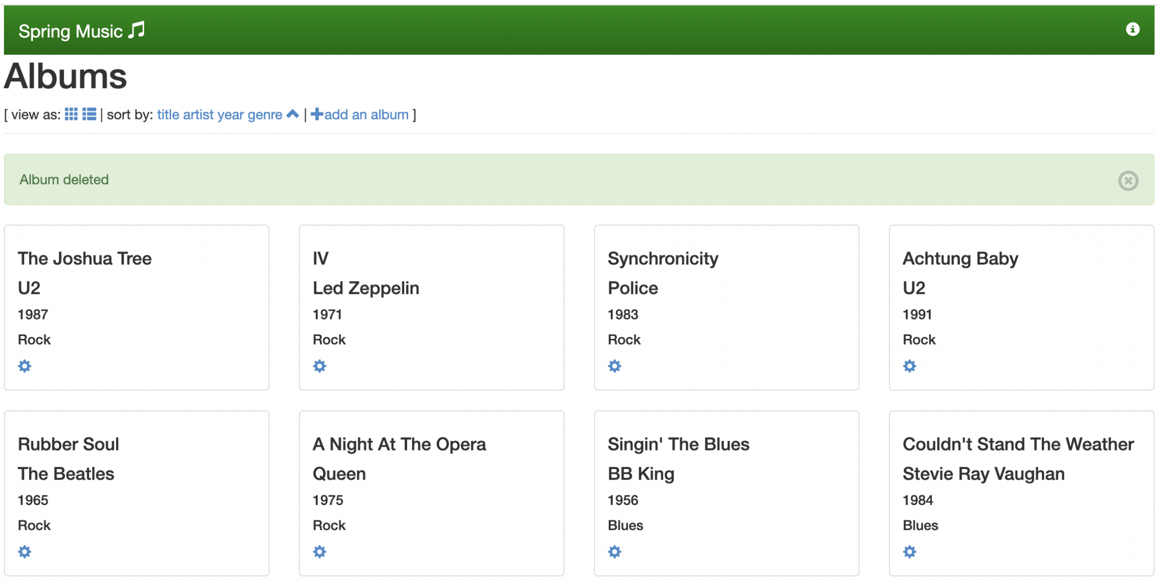
Albums

[view as: | sort by: title artist year genre | add an album]

Profiles: http2
Services:

Nevermind Nirvana 1991 Rock	Pet Sounds The Beach Boys 1966 Rock	What's Going On Marvin Gaye 1971 Rock	Are You Experienced? Jimi Hendrix Experience 1967 Rock
The Joshua Tree U2 1987 Rock	Abbey Road The Beatles 1969 Rock	Rumours Fleetwood Mac 1977 Rock	Sun Sessions Elvis Presley 1976 Rock
Thriller Michael Jackson 1982 Pop	Exile on Main Street The Rolling Stones 1972 Rock	Born to Run Bruce Springsteen 1975 Rock	London Calling The Clash 1980 Rock
Hotel California The Eagles	Led Zeppelin Led Zeppelin	IV Led Zeppelin	Synchronicity Police

3. Make a few changes to the music catalog: use the gear icon on each entry to modify or delete some of the albums, and use the "add an album" link at the top of the page to add some albums.


Spring Music

Albums

[view as: | sort by: title artist year genre | add an album]

Album deleted

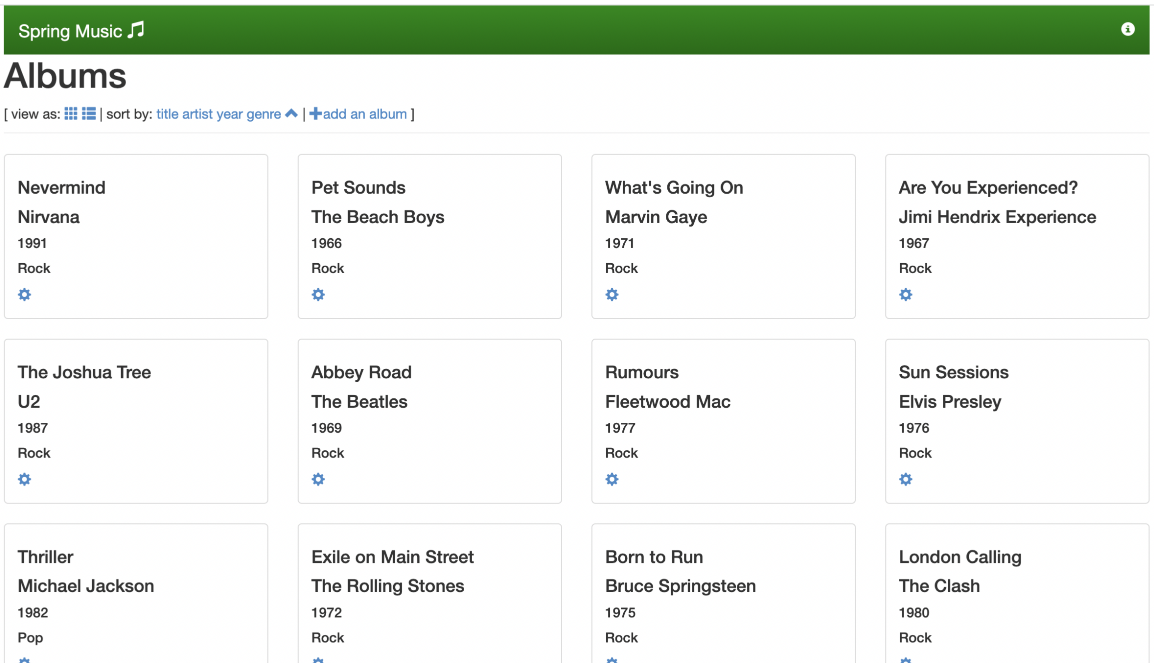
The Joshua Tree U2 1987 Rock	IV Led Zeppelin 1971 Rock	Synchronicity Police 1983 Rock	Achtung Baby U2 1991 Rock
Rubber Soul The Beatles 1965 Rock	A Night At The Opera Queen 1975 Rock	Singin' The Blues BB King 1956 Blues	Couldn't Stand The Weather Stevie Ray Vaughan 1984 Blues

4. Restart the spring-music application:

```
cf restart spring-music
```

The application is unavailable for a short period of time while it restarts.

5. After it has finished restarting, refresh your browser window and observe that your modifications have not been preserved.



Create a database service for persistent storage

Caution

The following instructions provide an easy way to create a MySQL database on the same Kubernetes cluster as the Application Service Adapter installation, but this database deployment is not suitable for production use. If you require a production-grade database on Kubernetes, consider using VMware Tanzu SQL with MySQL for Kubernetes or VMware Tanzu SQL with Postgres for Kubernetes.

1. Create a separate namespace to run a containerized MySQL database:

```
kubectl create namespace service-instances
```

2. Create a Kubernetes Deployment of a MySQL-compatible database in this namespace, along with a Kubernetes Service that exposes it inside the cluster and a Kubernetes Secret with credentials:

```
kubectl apply -n service-instances -f - <<EOF
---
apiVersion: v1
kind: Secret
metadata:
  name: spring-music-db
type: servicebinding.io/mysql
stringData:
  type: mysql
  provider: mariadb
  host: spring-music-db.service-instances.svc
  port: "3306"
  database: default
  # demo credentials
  username: user
  password: pass

---
apiVersion: v1
kind: Service
metadata:
  name: spring-music-db
spec:
  ports:
  - port: 3306
  selector:
    app: spring-music-db

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-music-db
  labels:
    app: spring-music-db
spec:
  selector:
    matchLabels:
      app: spring-music-db
  template:
    metadata:
      labels:
        app: spring-music-db
    spec:
      # no persistance configured, the database will be reset when the pod term
inates
      containers:
      - image: mariadb:10.5
        name: mysql
        env:
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              name: spring-music-db
              key: username
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: spring-music-db
              key: password
        - name: MYSQL_DATABASE
          valueFrom:
            secretKeyRef:
              name: spring-music-db
              key: database
        - name: MYSQL_ROOT_PASSWORD
          value: root
        ports:
        - containerPort: 3306
          name: mysql
        livenessProbe:
          tcpSocket:
            port: mysql
        readinessProbe:
          tcpSocket:
            port: mysql
        startupProbe:
          tcpSocket:
            port: mysql
EOF
```

3. Verify that the Deployment for the database is ready:

```
kubectl get deploy -n service-instances spring-music-db
```

   You should see output similar to the following:

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
spring-music-db    1/1     1            1           4m
```

   If the `READY` column does not display `1/1`, wait a few moments and inspect the Deployment again.

Bind the database to the application

1. Create a user-provided service instance in the Cloud Foundry space containing the JSON-formatted contents of the `spring-music-db` Secret:

```
MYSQL_CRED_JSON=$(kubectl -n service-instances get secret spring-music-db -ojso
n | jq '.data | map_values(@base64d)')
cf create-user-provided-service spring-music-db -p "$MYSQL_CRED_JSON"
```

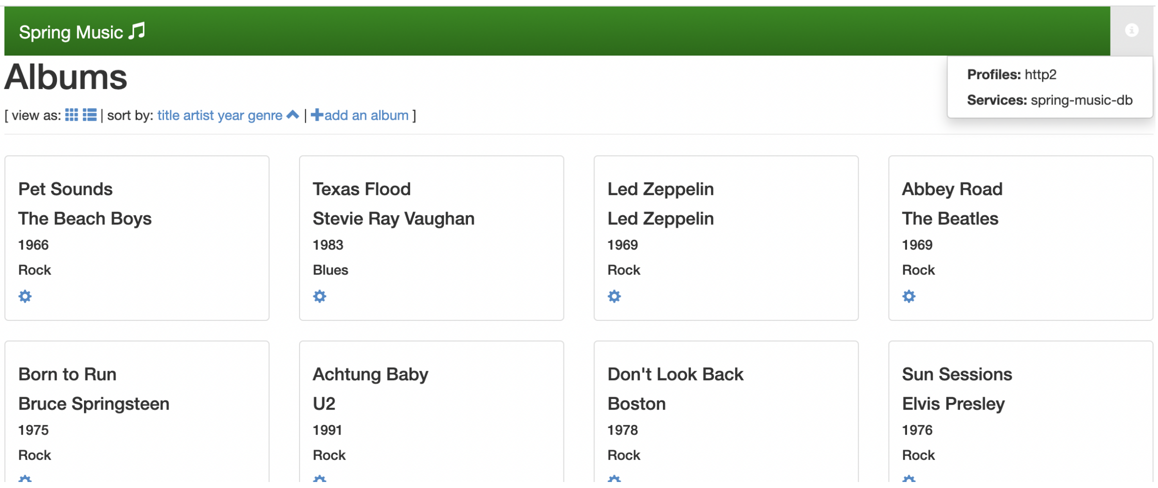
2. Bind the service instance to the spring-music application:

```
cf bind-service spring-music spring-music-db
```

3. Restage the application:

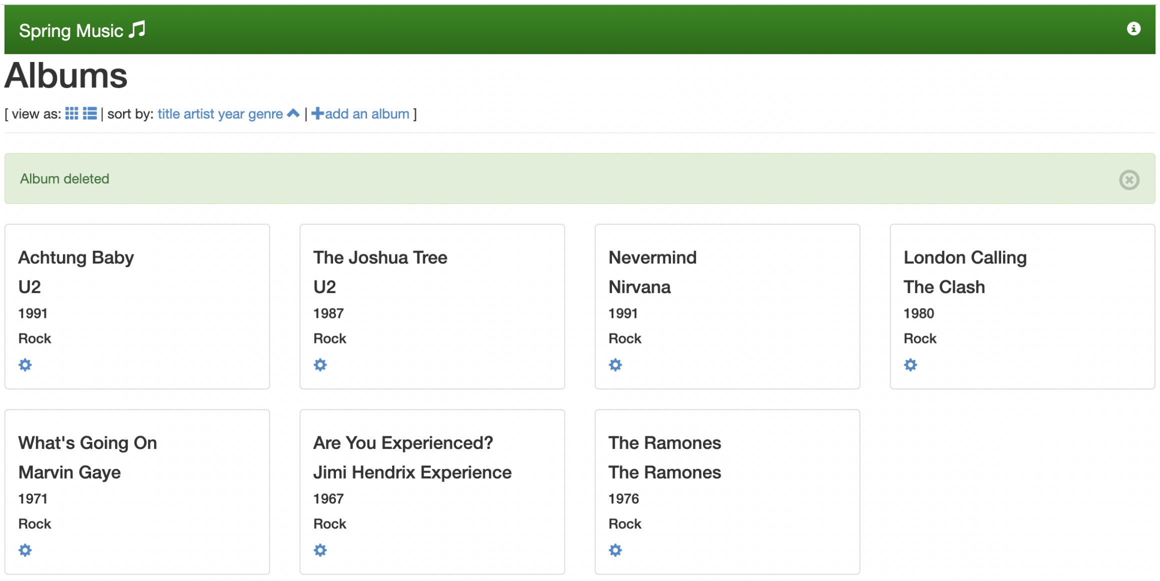
```
cf restage spring-music
```

4. After the application finishes restaging and restarting, refresh the browser window with the spring-music application and click on the app information icon in the upper right corner. It should indicate that the application is bound to the spring-music-db service:


Spring Music
Albums
[view as: | sort by: title artist year genre | add an album]
Profiles: http2
Services: spring-music-db

Pet Sounds The Beach Boys 1966 Rock	Texas Flood Stevie Ray Vaughan 1983 Blues	Led Zeppelin Led Zeppelin 1969 Rock	Abbey Road The Beatles 1969 Rock
Born to Run Bruce Springsteen 1975 Rock	Achtung Baby U2 1991 Rock	Don't Look Back Boston 1978 Rock	Sun Sessions Elvis Presley 1976 Rock

5. Make modifications to the album catalog again:


Spring Music
Albums
[view as: | sort by: title artist year genre | add an album]
Album deleted

Achtung Baby U2 1991 Rock	The Joshua Tree U2 1987 Rock	Nevermind Nirvana 1991 Rock	London Calling The Clash 1980 Rock
What's Going On Marvin Gaye 1971 Rock	Are You Experienced? Jimi Hendrix Experience 1967 Rock	The Ramones The Ramones 1976 Rock	

6. Restart the spring-music application again:

```
cf restart spring-music
```

7. Refresh the browser window and observe that the changes you made now persist through application restarts.

Administering Application Service Adapter

You can learn all about administering Application Service Adapter for VMware Tanzu Application Platform here:

- Disaster recovery
- Failover and redundancy
- Rotating certificates
- System logs and metrics
- Scaling Application Service Adapter

Disaster recovery with Application Service Adapter

You have a number of options for disaster recovery with Application Service Adapter. This topic describes your disaster recovery options.

Disaster recovery overview

You have a range of approaches for ensuring you can recover your Application Service Adapter deployment, apps, and data in case of a disaster. These approaches fall into two categories:

1. Backing up cluster and container image registry state and restoring from backups.
2. Re-creating the data in the deployment by automating the creation of state. This is achieved using scripted CI automation of the cf CLI and applying declarative Application Service Adapter resources to the cluster.

This topic focuses on the first approach by outlying where Application Service Adapter stores state and suggestions for backing up.

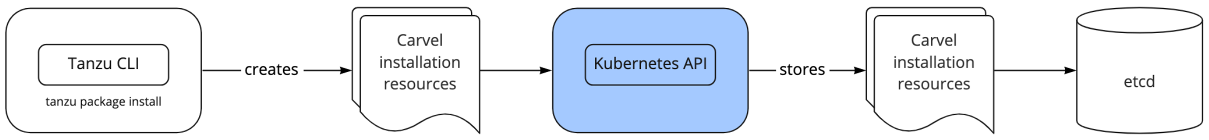
State storage

In contrast to TAS for VMs, Application Service Adapter has no dedicated databases or blobstore. Instead, state is stored in two places:

1. As Kubernetes custom resources in the Kubernetes API. These custom resources are persisted in the cluster's etcd datastore.
2. As container images in an OCI compatible registry, such as a self-hosted Harbor, DockerHub, or an IaaS provided registry.

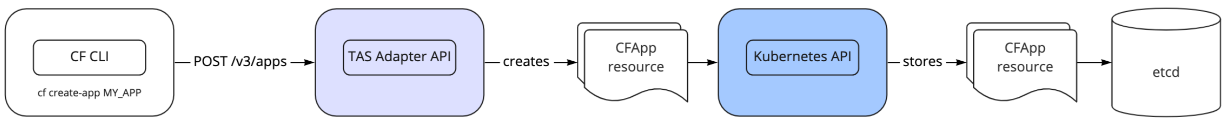
To help illustrate this, consider the following scenarios.

Application Service Adapter installation resources



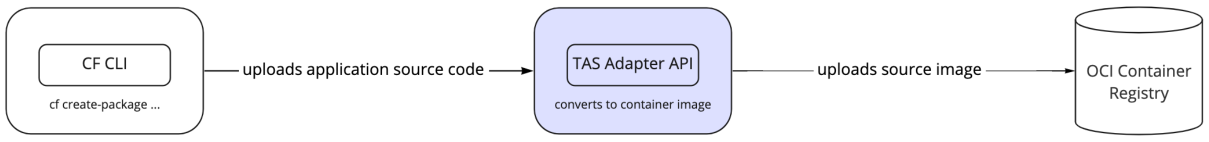
TAS operators are familiar with backing up their Ops Manager and BOSH Director databases to safeguard their installation configuration. For Application Service Adapter this works differently. Application Service Adapter is installed using the `tanzu package install` command. This causes the creation of a variety of Carvel package installation resources and `ConfigMaps` on the cluster which are managed by `kapp-controller`. These installation resources are stored in the Kubernetes API as custom resources which are ultimately stored in etcd.

Application Service Adapter Cloud Foundry API resources



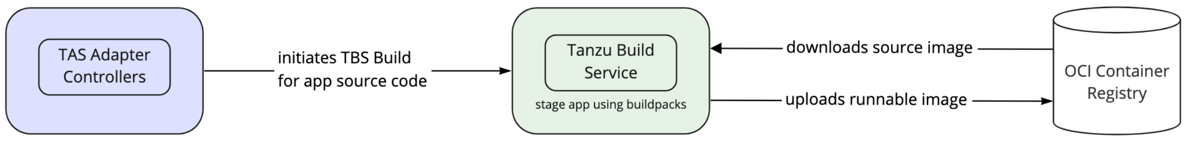
TAS for VMs stores CF API state and application configuration across a number of databases, with the majority of it held within the Cloud Controller API's CCDB database. Application Service Adapter does not maintain its own datastore for this purpose, but instead represents all Cloud Foundry resources as Kubernetes custom resources. These custom resources are managed by the Kubernetes API and stored in etcd.

Application source code



Application Service Adapter converts application source code into single-layer OCI images which are stored in the registry specified at installation.

Runnable application artifacts



TAS for VMs operators can be familiar with the concept of droplets, or TAR files representing staged apps that are ready to run on the platform. Application Service Adapter, using Tanzu Build Service, produces runnable container images instead of TAS-style droplets which are stored in the registry specified at installation.

Backup and restore

VMware recommends that operators take frequent backups of both the Kubernetes cluster's etcd and the registry using open source tools such as Velero or the native backup function provided by their infrastructure platform.

Failover and redundancy with Application Service Adapter

This topic covers failover characteristics and redundancy of Application Service Adapter components and applications pushed with Application Service Adapter.

For instructions to edit the scaling characteristics of your Application Service Adapter installation, see Scaling Application Service Adapter.

Cloud Foundry-compatible API

Application Service Adapter deploys two replicas of a v3 Cloud Foundry-compatible API (Korifi API) that clients communicate with. This API is stateless and is horizontally scaled for increased availability and performance. See Scaling the Application Service Adapter API.

Controllers and webhooks

Application Service Adapter deploys multiple components known as controllers to the cluster. For more information about controllers, see the Kubernetes documentation. These components watch and update state on the cluster in what is known as a "control loop," and over time, they ensure that the state in the cluster is consistent. Additionally, these components run admission webhooks that validate and update Application Service Adapter resources. For more information about dynamic admission control and admission webhooks, see the Kubernetes documentation.

Application Service Adapter controllers are effectively singletons. They have leader election turned on by default so only a single controller instance is active at a time. All instances can serve the webhooks. They are scaled horizontally for faster failover and for higher-availability of the webhooks. In the event that a controller fails, it is automatically restarted by the platform. Application Service Adapter controllers are idempotent and the newly restarted instance carries on where the failed one left off.

Applications

The failover characteristics and redundancy recommendations for applications that are pushed with the adapter depend on the application itself. However, there are some common recommendations that are provided for all applications.

1. Ensure that all applications have at least two instances by using `cf scale` to scale up the application or by declaring multiple instances in the app's manifest. A single-instance

application incurs downtime during cluster upgrades and maintenance. When an application is configured to run with two or more instances, the Kubernetes pod scheduler attempts to balance the instances across nodes and minimize downtime. Additionally, Application Service Adapter creates a `PodDisruptionBudget` for multi-instance applications that sets the minimum available instances for an app to be 50% of the total instances needed to maintain availability during these events. For more information about Protecting an Application with a `PodDisruptionBudget`, see the Kubernetes documentation.

2. Ensure that all applications have the appropriate health checks configured to accurately verify the readiness and liveness of your apps. For more information about app health checks, see the Cloud Foundry documentation.

   Application Service Adapter represents Cloud Foundry app health checks using `startupProbes` and `livenessProbes` on the underlying pods running the application. By default, a `port` health check is set to verify whether the app can accept TCP connections, but you can configure more advanced `http` health checks to better detect readiness of the application.

Rotate Application Service Adapter certificates

You may need to manually rotate your Application Service Adapter system certificates. This topic tells you how.

Note

Certificate rotation does not result in downtime.

Rotating ingress certificates

1. Set up environment variables for the installation:

   ```
   export TAS_ADAPTER_VERSION=VERSION-NUMBER
   ```

   Where `VERSION-NUMBER` is the version of Application Service Adapter you want to install. For example, `1.0.0`.

2. Update your `tas-adapter-values.yaml` file with new API and App Ingress TLS certificates, such as crt and key.

   The following values are updated:

   ```
   api_ingress:
     tls:
       secret_name: NEW-API-TLS-SECRET-NAME
       namespace: NEW-API-TLS-SECRET-NAMESPACE
   app_ingress:
     tls:
       secret_name: NEW-APP-TLS-SECRET-NAME
       namespace: NEW-APP-TLS-SECRET-NAMESPACE
   ```

   Where:

  - `NEW-API-TLS-SECRET-NAME` is the kubernetes.io/tls secret containing the PEM-encoded public certificate for the Application Service Adapter API.
  - `NEW-API-TLS-SECRET-NAMESPACE` is namespace containing the Application Service Adapter API secret.
  - `NEW-APP-TLS-SECRET-NAME` is the kubernetes.io/tls secret containing the PEM-encoded public certificate for applications deployed using the Application Service Adapter.
  - `NEW-APP-TLS-SECRET-NAMESPACE` is the namespace containing the Application Service Adapter applications secret.

3. Install the Application Service Adapter to the cluster by running:

```
tanzu package install tas-adapter \
  -p application-service-adapter.tanzu.vmware.com \
  --version "${TAS_ADAPTER_VERSION}" \
  --values-file tas-adapter-values.yaml \
  --namespace tap-install
```

4. Verify that the package install was successful. Run:

```
tanzu package installed get tas-adapter \
  --namespace tap-install
```

The following is an example output:

```
| Retrieving installation details for tas-adapter...
NAME:                    tas-adapter
PACKAGE-NAME:            application-service-adapter.tanzu.vmware.com
PACKAGE-VERSION:         1.0.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Rotating internal certificates

Internal certificates are managed by Certificate Manager with a self-signed Certificate Authority. The default Certificate Manager configuration provides certificates that are valid for 90 days. Certificates are renewed 30 days before expiry. For more information, see the cert-manager documentation.

System logs and metrics for Application Service Adapter

You can access Application Service Adapter system component logs and query metrics by following the steps in this topic.

- Gathering system logs
- Querying performance metrics

Gathering system logs

Logs for Application Service Adapter system components flow through standard Kubernetes logging channels. You can retrieve them by using the tooling of your choice. For ad-hoc log retrieval, VMware recommends the `kapp logs` command in the kapp cli as the most convenient method to gather the logs from pods in one or more deployments. See the Carvel documentation.

To fetch recent logs from all components of an Application Service Adapter installation:

```
kapp logs --app tas-adapter.app -n tap-install
```

To fetch recent logs from a specific deployment:

```
kapp logs --app tas-adapter.app -n tap-install --pod-name DEPLOYMENT-NAME%
```

Where `DEPLOYMENT-NAME` is the name of the Kubernetes Deployment, for example, `korifi-api-deployment`.

To stream logs instead of fetching the most recent logs, add the `--follow` flag to the earlier `kapp logs` commands. For additional details and options, see the `kapp logs --help` help text.

Querying performance metrics

All of the controller-managers deployed by Application Service Adapter provide standard Prometheus performance metrics generated by the Kubernetes controller-runtime library. For more information, see the Kubebuilder documentation.

Application Service Adapter controller managers are deployed with annotations that makes them discoverable by a Prometheus server deployed to the same cluster.

After you have Prometheus deployed, you can query any of the available metrics. See Default Exported Metrics References in the Kubebuilder documentation. In particular, queries for `workqueue_depth` or `workqueue_queue_duration_seconds` can help to indicate when a controller manager is resource constrained and must be scaled up.

To query the queue depth for the CFApp controller:

```
workqueue_depth{name="cfapp"}
```

To list the queue depths for all Application Service Adapter controllers:

```
workqueue_depth{namespace="tas-adapter-system"}
```

To query the average queue time for CFApp objects before reconciliation:

```
workqueue_queue_duration_seconds_sum{name="cfapp"}/workqueue_queue_duration_seconds_co
unt
```

Scale Application Service Adapter

Application Service Adapter gives you options for scaling components for performance or availability. Read this topic for scaling guidance.

- Notes on system performance

- Scaling the Application Service Adapter controller managers
- Scaling the Application Service Adapter API

Notes on system performance

The Application Service Adapter runs on a Kubernetes cluster and makes extensive use of the Kubernetes API to store and manage resources. For these reasons, the performance of the Application Service Adapter is very dependent on the underlying Kubernetes control plane. If you observe slow performance or timeouts when using the Application Service Adapter, a good first step is to ensure that the control plane for your Kubernetes cluster is properly resourced.

For managed clusters that come from a public-cloud IaaS, you generally won't have direct control over how the control plane is provisioned, so you won't be able to scale up control place resources. If you are managing your own Kubernetes clusters, you must ensure you have provisioned adequate capacity for the `apiserver` and `etcd` components. See the Kubernetes documentation as a starting place for understanding production cluster concerns.

Scaling the Application Service Adapter controller managers

Vertical scaling

The Application Service Adapter deploys a set of controller managers responsible for reconciling various custom resources into running pods on your Kubernetes cluster. If these controller managers are resource constrained, scaling them up vertically may improve performance. In particular, controller managers maintain an in-memory informer cache of the events that they are responsible for tracking, so it is common for controller managers on busy systems with many objects to require more memory.

Controller manager deployments in the `tas-adapter-system` namespace are:

- `cartographer-builder-runner-controller-manager`
- `korifi-controllers-controller-manager`
- `korifi-job-task-runner-controller-manager`
- `korifi-kpack-build-controller-manager`
- `korifi-statefulset-runner-controller-manager`

To determine whether your controller managers face CPU or memory pressure, you can check the Prometheus metrics for your controllers as described in the metrics topic.

You can also inspect the running pods for these deployments using the `kubectl top` command.

If you determine that one or more of your controller managers is resource constrained, you can increase the available resources by setting the optional scaling parameters as described in the configure installation settings instructions, and then re-running the install step.

Horizontal scaling

As described in the failover and redundancy topic, controller managers use leader election to designate a single active instance, so horizontal scaling is not an effective strategy to improve performance of a slow controller. Since the same deployments also serve webhook requests, you may choose to add horizontal scale to improve webhook performance. You may also choose to scale your deployments to improve controller availability. Multi-instance deployments allow rolling deployments so that controllers and webhooks don't become unresponsive during upgrades. Horizontal scaling can be achieved by using the `replicas` scaling configuration.

Scaling the Application Service Adapter API

If you have confirmed that the Kubernetes control plane and the controller managers both have sufficient resources, but you are still observing slow response times or timeouts from the Cloud Foundry API, scaling up the `korifi-api-deployment` in the `tas-adapter-system` namespace may help.

Unlike the controller managers, requests to the pods in this deployment are load balanced, so either horizontal or vertical scaling should improve performance. By default, `korifi-api-deployment` runs with two replicas, so it should already maintain availability during upgrades unless it is scaled down.

Similar to the controller managers, both vertical (memory and CPU) and horizontal (replicas) scaling can be achieved by changing the installation settings, and then re-running the install step.

Upgrade Application Service Adapter

This topic gives you the steps to upgrade Application Service Adapter for VMware Tanzu Application Platform.

You can perform a fresh installation of the Application Service Adapter by following the instructions in Installing Application Service Adapter.

When upgrading to a new major or minor version of Application Service Adapter, please see that versions' documentation for version-specific configuration and upgrade instructions.

You can find new patch versions of Application Service Adapter on the TanzuNet product page directly, or sign up to receive email alerts when the product is updated.

Note: Upgrades are not currently supported if you have enabled the experimental Cartographer integration. Existing application workloads will not behave correctly after upgrading from v1.0 to v1.1.

Prerequisites

Before you upgrade Application Service Adapter:

- Verify that you meet all the prerequisites of the target Tanzu Application Service Adapter for TAP version. If the target Tanzu Application version does not support your existing Kubernetes version, VMware recommends upgrading to a supported version before proceeding with the upgrade.
- Install or update the Tanzu CLI and plugins

- For information about installing or updating the Tanzu CLI and plug-ins, see Install or update the Tanzu CLI and plugins.
- Verify all packages are reconciled by running `tanzu package installed list -A`.
- It is strongly recommended to upgrade the Tanzu Application Platform version to the latest patch version of the currently installed major-minor (for example, 1.2).

Update the new package repository

Follow these steps to update the new package repository:

1. Add the target version of the Application Service Adapter package repository:

```
tanzu package repository add tas-adapter-repository \
  --url registry.tanzu.vmware.com/app-service-adapter/tas-adapter-package-rep
o:${TAS_ADAPTER_VERSION} \
  --namespace tap-install
```

2. Verify you have added the new package repository by running:

```
tanzu package available list \
  --namespace tap-install
```

Upgrade Application Service Adapter

To upgrade, run:

```
tanzu package installed update tas-adapter \
  -p application-service-adapter.tanzu.vmware.com \
  --version "${TAS_ADAPTER_VERSION}" \
  --values-file tas-adapter-values.yaml \
  --namespace tap-install
```

Where `TAS_ADAPTER_VERSION` is the target revision of Application Service Adapter you are migrating to.

Verify the upgrade

Important

Run the following command in the directory where the `tas-adapter-values.yaml` file resides.

To verify the versions of packages after the upgrade, run:

```
tanzu package installed list --namespace tap-install
```

Your output should be similar, but probably not identical, to the following example output:

```
  Retrieving installed packages...
  NAME                         PACKAGE-NAME                                 PACKAGE-VER
SION  STATUS
  api-auto-registration        apis.apps.tanzu.vmware.com                   0.1.1
Reconcile succeeded
  appsso                       sso.apps.tanzu.vmware.com                    2.0.0
Reconcile succeeded
  buildservice                 buildservice.tanzu.vmware.com                1.7.2
Reconcile succeeded
  cartographer                 cartographer.tanzu.vmware.com                0.5.3
Reconcile succeeded
  cert-manager                 cert-manager.tanzu.vmware.com                1.7.2+tap.1
Reconcile succeeded
  contour                      contour.tanzu.vmware.com                     1.22.0+tap.
4     Reconcile succeeded
  eventing                     eventing.tanzu.vmware.com                    2.0.1
Reconcile succeeded
  ootb-templates               ootb-templates.tanzu.vmware.com              0.10.2
Reconcile succeeded
  policy-controller            policy.apps.tanzu.vmware.com                 1.1.2
Reconcile succeeded
  service-bindings             service-bindings.labs.vmware.com             0.8.0
Reconcile succeeded
  source-controller            controller.source.apps.tanzu.vmware.com      0.5.0
Reconcile succeeded
  tap                          tap.tanzu.vmware.com                         1.3.0
Reconcile succeeded
  tap-telemetry                tap-telemetry.tanzu.vmware.com               0.3.1
Reconcile succeeded
  application-service-adapter application-service-adapter.tanzu.vmware.com 1.0.1
Reconcile succeeded
  tekton-pipelines             tekton.tanzu.vmware.com                      0.39.0+tap.
2     Reconcile succeeded
```

Uninstall Application Service Adapter

This topic gives you the steps to uninstall Application Service Adapter for VMware Tanzu Application Platform.

To uninstall Application Service Adapter:

1. Uninstall the Application Service Adapter package from your cluster.

   ```
   tanzu package installed delete tas-adapter \
     --namespace tap-install
   ```

2. Remove the repository from your cluster.

   ```
   tanzu package repository delete tas-adapter-repository \
     --namespace tap-install
   ```

3. Delete the image pull secret.

   ```
   tanzu secret registry delete tanzunet-tas-adapter-registry \
     --namespace tap-install
   ```

Learn more about Application Service Adapter

You can learn more about Application Service Adapter for VMware Tanzu Application Platform in the topics listed here:

- Technical architecture
- User authentication

Application Service Adapter Architecture

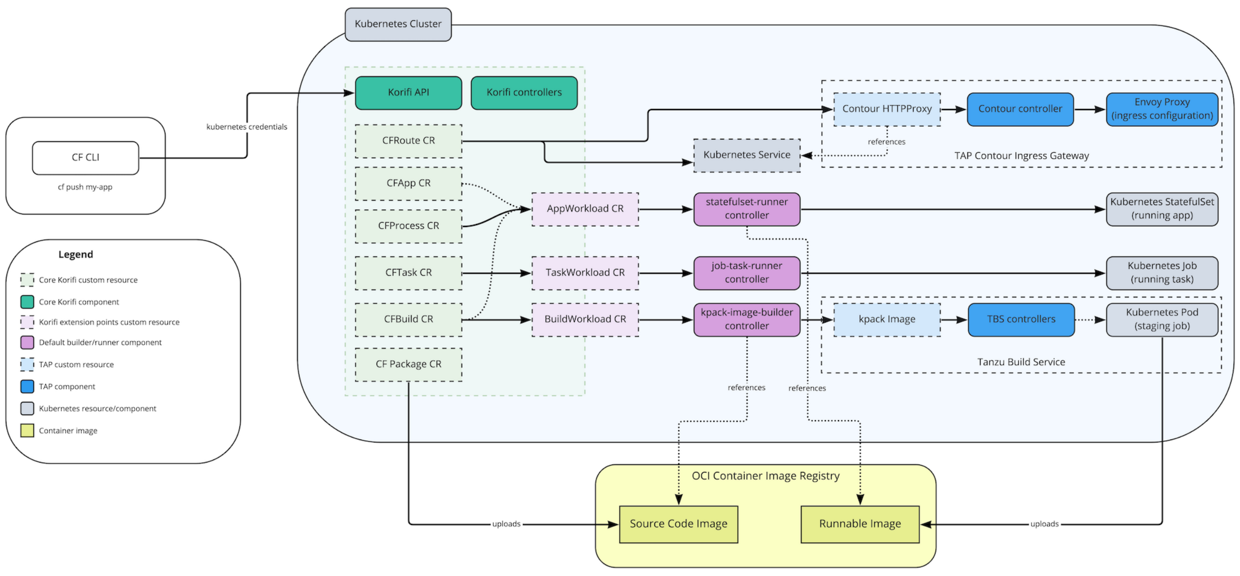
You can read about the technical architecture of Application Service Adapter and its subsystems in this topic.

- Overview
- High-level architecture
- Experimental Cartographer integration
- Authentication and authorization
- Organization and Space management
- Building (staging) applications
- Services
- Routing
- App Logging and Metrics

Overview

Application Service Adapter implements a subset of the v3 Cloud Foundry APIs in order to support common Cloud Foundry developer workflows. Application Service Adapter is installed directly onto a Kubernetes cluster than has Tanzu Application Platform (TAP) installed and provides a CF API translation layer that converts CF API calls into underlying and Kubernetes resources. In addition to this API, Application Service Adapter also provides a set of Kubernetes custom resources, controllers, and admission webhooks.

High-level architecture



Components

Application Service Adapter includes components from the open source Cloud Foundry Korifi project as well as a number of proprietary components for deeper integration with Tanzu Application Platform.

The Korifi components are primarily tasked with implementing the Cloud Foundry data model and APIs. They are:

- **Korifi CRDs:** A set of Kubernetes custom resources under the korifi.cloudfoundry.org API Group that implement the core V3 Cloud Foundry resources. These custom resources replicate the Cloud Foundry data model and are used to store persistent state for the CF API translation layer. These resources serve as an extension point for further interoperability and are accessible to Kubernetes users using existing Kubernetes tooling (e.g. the kubectl CLI).
- **Korifi API deployment:** A Golang implementation of a core set of V3 Cloud Foundry APIs that is backed by the Korifi CRDs. Existing Cloud Foundry API clients (such as the cf CLI) can target the Korifi API and continue to use their existing CF developer workflows.
- **Korifi Controllers deployment:** A set of Kubernetes controllers that implement CF subsystems by orchestrating and reconciling the Korifi CRDs into consolidated intermediate resources that contain the information necessary to build and run an application. This deployment also runs admission webhooks that validate and normalize Korifi resources.

Application Service Adapter converts these intermediate resources using its default builder/runner controller implementations into TAP and Kubernetes resources such as Tanzu Build Service (kpack) Images for building applications and Kubernetes StatefulSets for running them. These intermediate resources and default implementations are:

- **BuildWorkload resource**: A custom resource that serves as an interface to the underlying build system used for staging applications. This resource contains all the information needed to stage an app, and controller implementations communicate back via its status. The `kpack-image-builder` component is the default implementation for application staging that uses Tanzu Build Service and Tanzu Buildpacks. When the experimental Cartographer integration is activated, this resource is reconciled by the `cartographer-builder-runner` component. For more details on the application building process, see the build (staging) applications section below.

- **AppWorkload resource**: A custom resource that serves as an interface to the underlying runtime. This resource contains all the information needed to run an app, and controller implementations communicate back via its status. The `statefulset-runner` component is the default implementation that runs apps using Kubernetes `StatefulSets`. `StatefulSets` allow Application Service Adapter to support Cloud Foundry features such as the `CF_INSTANCE_INDEX` (an ordered numeric index for each container) environment variable and APIs. When the experimental Cartographer integration is activated, this resource is reconciled by the `cartographer-builder-runner` component.
- **TaskWorkload resource**: A custom resource that serves as an interface to the underlying runtime. This resource contains all the information needed to run a Cloud Foundry task, and controller implementations communicate back via its status. The `job-task-runner` component is the default implementation that runs tasks via Kubernetes `Jobs`.

Experimental Cartographer integration

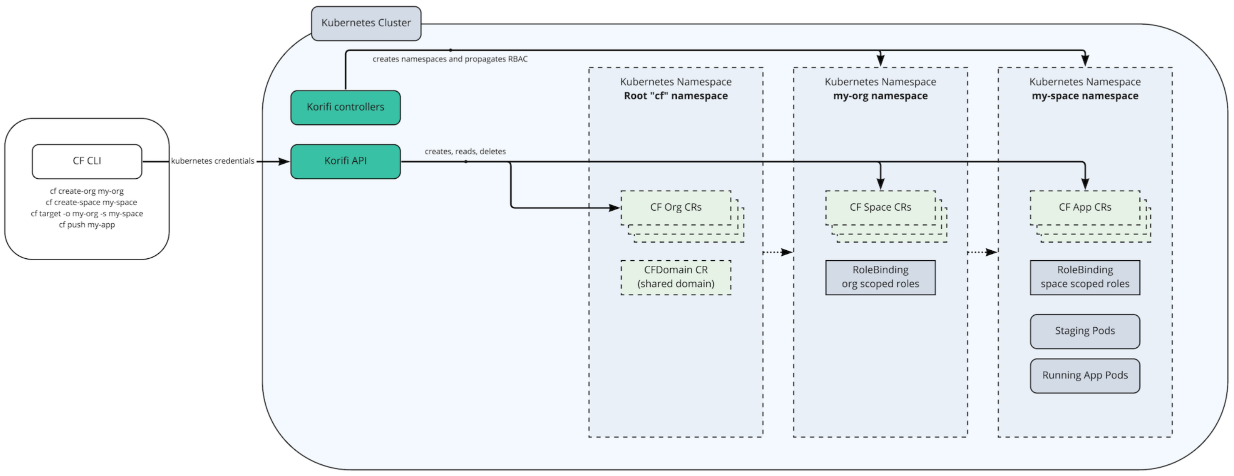
Application Service Adapter includes an (optional) experimental builder and runner implementation (`cartographer-builder-runner`) that utilizes TAP's Cartographer Supply Chains to manage the build and run stages of the application lifecycle.

Authentication and authorization

Application Service Adapter does not include its own user management or permission system and instead uses the user's Kubernetes credentials to interact directly with the Kubernetes API. Cloud Foundry roles are implemented using Kubernetes RBAC resources and users can be assigned to these roles using the existing cf CLI role assignment commands. Internally this is represented as namespace-scoped `RoleBindings` to `ClusterRoles` that represent common CF roles such as `SpaceDeveloper`.

For more details, see the dedicated [User Authentication Overview] docs.

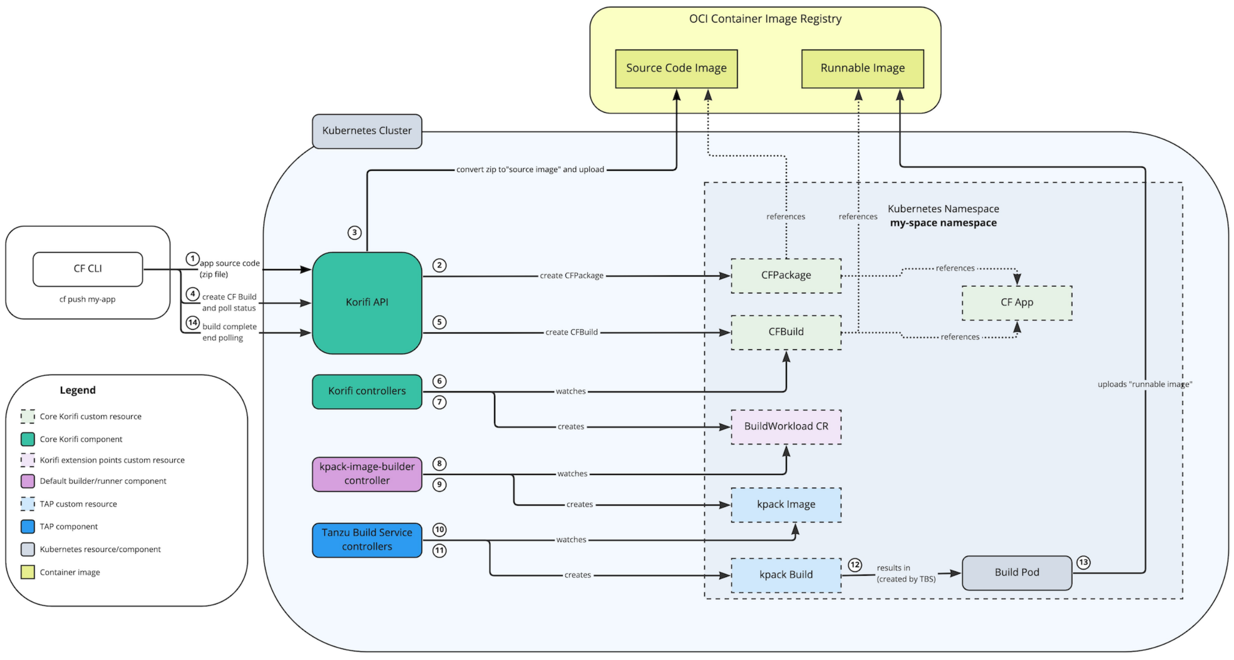
Organization and space management



Cloud Foundry has a tiered tenancy system consisting of the cluster or "foundation" level, organization level, and space level. A Cloud Foundry installation will contain one or more organizations which will themselves contain one or more spaces. CF roles typically allow for read/write access in these various areas. For example, a "CF Admin" user can make shared domains for the entire CF installation as well as interact with apps within an individual space, while a "Space Developer" user will typically be able to view certain resources within their org as well as push apps within their assigned space.

Application Service Adapter models CF organizations and spaces using Kubernetes Namespaces. There is a root "cf" namespace that can contain multiple `CFOrg` custom resources. These initiate the creation of namespaces for each org which themselves will contain `CFSpace` resources that point to additional namespaces for each space. This maps closely to the CF tenancy model in terms of app isolation and user permissions on Kubernetes. All organization-scoped CF resources live within the corresponding "organization namespace" and space-scoped CF resources (apps, routes, builds, etc.) live within the relevant "space namespace." Kubernetes RBAC resources are made in the each namespace to control access. For example, an Org Manager has a `RoleBinding` to the "OrgManager" `ClusterRole` in each namespace they manage. Likewise, Space Developers have a `RoleBinding` to the "SpaceDeveloper" `ClusterRole` in the space namespaces.

Building (staging) applications



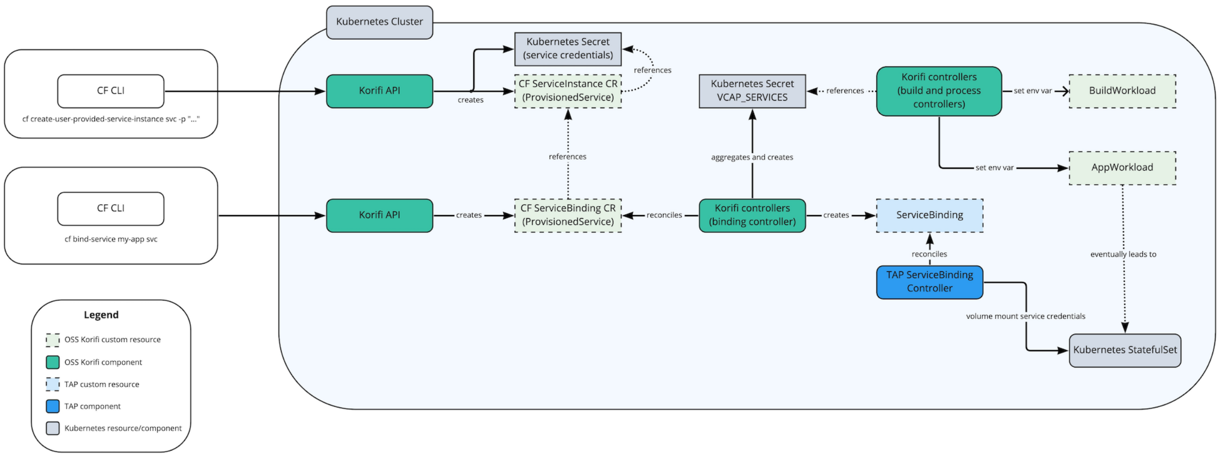
Application Service Adapter uses Tanzu Build Service and Tanzu Buildpacks to build applications.

Application source code is packaged and transmitted by CF clients to the Korifi API where it is converted into a single-layer container image, or "source image", that can be used by Tanzu Build Service. When CF clients create a Cloud Foundry Build through the Korifi API this is translated into a "BuildWorkload" custom resource by Korifi. By default, Application Service Adapter will use its `kpack-image-builder` controller to translate this "BuildWorkload" directly into Tanzu Build Service resources and a "build Pod" will be scheduled to build the app using Tanzu Buildpacks. Tanzu Buildpacks are Cloud Native Buildpacks and can be thought of as an evolution of the Cloud Foundry buildpacks that Cloud Foundry operators may be familiar with. They differ mainly in that there are a few configuration differences and that they produce OCI container images instead of Cloud Foundry droplets.

Note on blobstores

Operators of Tanzu Application Service may be familiar with the platform's "blobstore," or object storage for application source code and staged droplets. Application Service Adapter does not rely on a blobstore and instead uses the image registry that is configured at installation time for storing source code and runnable app images.

Services

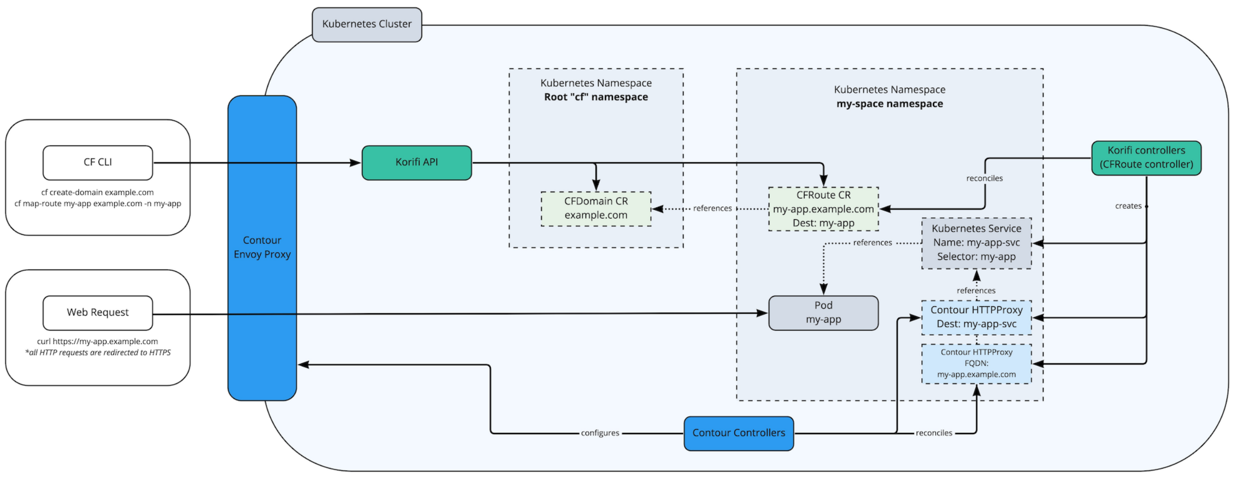

Kubernetes Cluster
CF CLI
cf create-user-provided-service-instance svc -p "..."
CF CLI
cf bind-service my-app svc
Korifi API
Korifi API
creates
creates
Kubernetes Secret (service credentials)
references
CF ServiceInstance CR (ProvisionedService)
references
CF ServiceBinding CR (ProvisionedService)
reconciles
Kubernetes Secret VCAP_SERVICES
aggregates and creates
references
Korifi controllers (build and process controllers)
set env var
BuildWorkload
set env var
AppWorkload
Korifi controllers (binding controller)
creates
ServiceBinding
reconciles
TAP ServiceBinding Controller
volume mount service credentials
eventually leads to
Kubernetes StatefulSet
Legend
OSS Korifi custom resource
OSS Korifi component
TAP custom resource
TAP component
Kubernetes resource/component

Application Service Adapter has support for user-provided service instances through the CFServiceInstance and CFServiceBinding custom resources. These resources primarily exist to power the CF APIs and store additional state that isn't relevant downstream. They also implement the `ProvisionedService` "duck type" from the Service Binding for Kubernetes specification which allow them to interoperate directly with other projects in the service bindings ecosystem (e.g. kpack, ServiceBinding reconcilers, etc.).

Application developers can provide service credentials through user-provided service instances and Application Service Adapter will store them in Kubernetes `Secrets`. It then aggregates them into a single "VCAP_SERVICES" secret that is provided to app containers as the VCAP_SERVICES environment variable to maintain compatibility with existing Cloud Foundry-aware applications and frameworks.

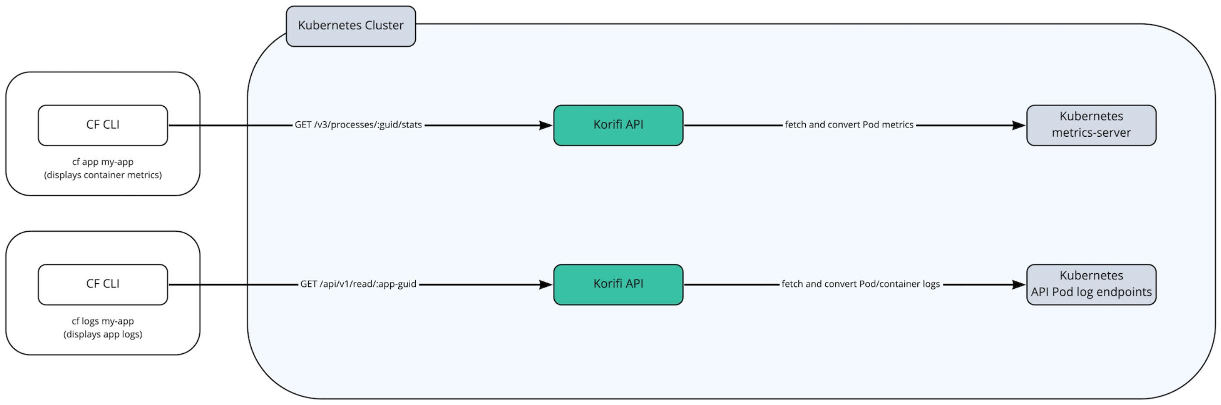
Additionally, Application Service Adapter integrates with the TAP Service Bindings reconciler through the ServiceBinding resource to volume mount these credentials on to workload Pods. This enables apps to use updated frameworks, such as Spring Cloud Bindings, that are aware of this form of credential projection.

Routing



Application Service Adapter uses the Tanzu Application Platform installation's Contour to implement ingress routing for both the Korifi API and app workloads. The `CFRoute` custom resource backs the relevant Cloud Foundry route management APIs and is converted by the Korifi Controllers component into Contour `HTTPProxy` and Kubernetes `Service` resources. A validating admission webhook applies validation rules to the routes (e.g. no duplicate routes, route has a valid `CFDomain`, etc).

App logging and metrics


Kubernetes Cluster
CF CLI
cf app my-app
(displays container metrics)
GET /v3/processes/:guid/stats
Korifi API
fetch and convert Pod metrics
Kubernetes metrics-server
CF CLI
cf logs my-app
(displays app logs)
GET /api/v1/read/:app-guid
Korifi API
fetch and convert Pod/container logs
Kubernetes API Pod log endpoints

Application Service Adapter supports best-effort access to the latest logs and metrics for apps through the "cf app", "cf logs", and "cf push" cf CLI commands. The Korifi API implements the relevant Cloud Foundry APIs for querying these resources and translates the request into requests to the Kubernetes metrics-server (for Pod cpu/memory metrics) and the Kubernetes Pod log API (application/staging logs).

For longer term storage and more reliable access to logs and metrics, we suggest following these recommendations around app observability and exporting logs and metrics to an external service such as VMware Aria Operations for Applications.

User authentication with Application Service Adapter

In this topic, learn how Application Service Adapter authenticates and authorizes users by using the Kubernetes API server.

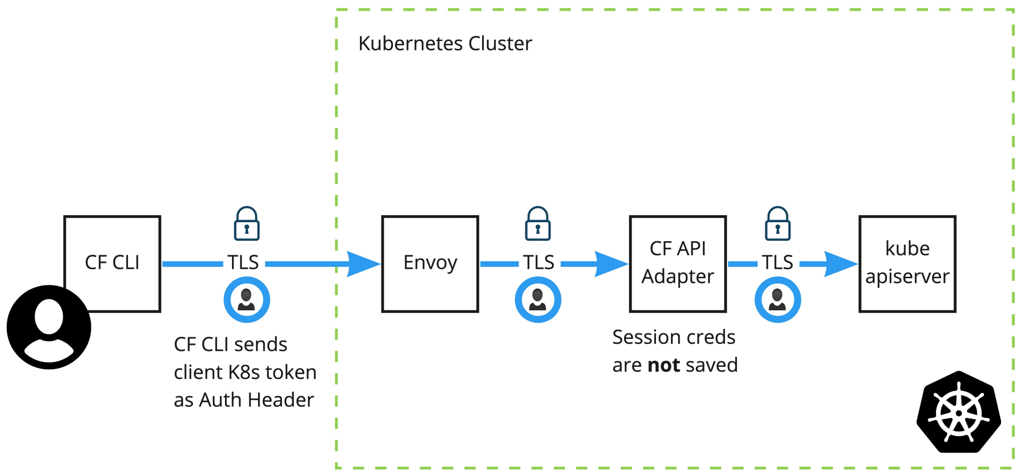
Background

Traditionally, the cf CLI authenticates with the Cloud Foundry User Account and Authentication (UAA) server, which acts as an OAuth2 provider. In this model, the Cloud Foundry API server validates the user's token and authorizes user actions based on its own set of user role assignments.

The Application Service Adapter takes a different approach to user authentication and authorization. Instead of requiring UAA as a separate account and authentication service, the Application Service Adapter delegates this responsibility to the Kubernetes API server. The cf CLI now recognizes when it targets a Kubernetes-backed CAPI server such as the Application Service Adapter, uses user information from the local kubeconfig file to authenticate with the underlying Kubernetes API, and extracts the user token or client certificate or key pair from the authentication response. When it makes a request to the CAPI server, it then sends that credential in the Authorization header. The Cloud Foundry API server uses that credential to perform requests on behalf of the end user and retains it in memory only for the duration of the user's API request.

The Application Service Adapter relies on core Kubernetes role-based access control (RBAC) resources such as `ClusterRole` and `RoleBinding` to configure user authorization rules. Platform operators can create these RBAC resources either using the Cloud Foundry role API endpoints or directly in the Kubernetes API.

Architecture


Kubernetes Cluster
CF CLI
TLS
Envoy
TLS
CF API Adapter
TLS
kube apiserver
CF CLI sends client K8s token as Auth Header
Session creds are **not** saved

The Application Service Adapter API requires that users connect to it using HTTPS because the Authorization header contains the user's authentication token or client certificate or key pair. The API translates the CAPI request into Kubernetes API requests using the provided credentials.

Note

> The user is authenticated through their Kubernetes token or client certificate or key for each request to the Adapter's API. There is no persistent session data stored between requests.

VMware recommends using short-lived tokens or certificates to authenticate with the Application Service Adapter. The Application Service Adapter warns users if their certificate is still valid in one week.

Application Service Adapter reference documentation

You can view all the Application Service Adapter reference documentation here:

- Application Service Adapter environment variables
- Buildpacks used by Application Service Adapter
- cf CLI commands supported by Application Service Adapter
- Troubleshoot Application Service Adapter
- User management

Application Service Adapter environment variables

Application Service Adapter sets environment variables when it builds and runs an application. Here's what you should know about environment variables.

Environment variables overview

Environment variables are the means by which Application Service Adapter communicates with a deployed app about its environment.

For information about setting your own app-specific environment variables, see the Environment Variable section of the *Deploying with App Manifests* topic.

View environment variables

Using the Cloud Foundry Command Line Interface (cf CLI), you can run the `cf env` command to view the Application Service Adapter environment variables for your app. The `cf env` command displays the following environment variables:

- The user-provided variables set using the `cf set-env` command

For more information about the `cf env` command, see env in the cf CLI documentation. For more information about the `cf set-env` command, see set-env in the cf CLI documentation.

The following example demonstrates the environment variables `cf env` displays:

```
$ cf env my-app
Getting env variables for app my-app in org my-org / space my-space as
admin...
No system-provided env variables have been set
```

```
User-Provided:
MY_DRAIN: http://drain.example.com
MY_ENV_VARIABLE: 100

No running env variables have been set

No staging env variables have been set
```

App-specific system variables

This section describes the environment variables that Application Service Adapter makes available to your app container. Some of these variables are the same across instances of a single app, and some vary from instance to instance.

You can access environment variables programmatically, including variables defined by the buildpack.

The table below lists the app-specific system environment variables available to your app container. See App-Specific System Variables in *TAS for VMs Environment Variables* for more information on each environment variable.

Environment Variable	Running	Staging
`CF_INSTANCE_ADDR`		
`CF_INSTANCE_GUID`	x	
`CF_INSTANCE_INDEX`	x	
`CF_INSTANCE_INTERNAL_IP`	x	
`CF_INSTANCE_IP`	x	
`CF_INSTANCE_PORT`		
`CF_INSTANCE_PORTS`		
`CF_STACK`		
`DATABASE_URL`		
`HOME`	x	
`INSTANCE_GUID`		
`INSTANCE_INDEX`		
`LANG`		
`MEMORY_LIMIT`		
`PATH`	x	
`PORT`	x	
`PWD`	x	
`TMPDIR`		

Environment Variable	Running	Staging
`USER`		
`VCAP_APP_HOST`	x	
`VCAP_APP_PORT`	x	
`VCAP_APPLICATION`		
`VCAP_SERVICES`	x	

CF_INSTANCE_GUID

The UUID of the app instance.

For example: `CF_INSTANCE_GUID=41653aa4-3a3a-486a-4431-ef258b39f042`

CF_INSTANCE_INDEX

The index number of the app instance.

For example: `CF_INSTANCE_INDEX=0`

CF_INSTANCE_INTERNAL_IP

The internal IP address of the container running the app instance.

For example: `CF_INSTANCE_INTERNAL_IP=5.6.7.8`

CF_INSTANCE_IP

The external IP address of the host running the app instance.

For example: `CF_INSTANCE_IP=1.2.3.4`

HOME

The root folder for the deployed app.

For example: `HOME=/home/cnb`

PORT

The port on which the app should listen for requests. Application Service Adapter allocates a port dynamically for each instance of the app, so code that obtains or uses the app port should refer to it using the `PORT` environment variable.

For example: `PORT=8080`

PWD

The present working directory where the buildpack that processed the app ran.

For example: `PWD=/workspace`

VCAP_APP_HOST

Deprecated. Always set to `0.0.0.0`.

VCAP_APP_PORT

Deprecated name for the `PORT` variable.

VCAP_SERVICES

Application Service Adapter has support for user-provided service instances and adds their binding details to the `VCAP_SERVICES` environment variable. For more information, see User-Provided Service Instances in the Cloud Foundry documentation.

Application Service Adapter returns the results as a JSON document that contains an object for each service for which one or more instances are bound to the app. The service object contains a child object for each instance of the service that is bound to the app.

The table below defines the attributes that describe a bound service. The key for each service in the JSON document is the same as the value of the `label` attribute.

Note

Application Service Adapter does not support managed services, so the `label` for a user-provided service instance is always `user-provided`. VMware recommends that apps find connection details through the user-settable `tags` field when parsing `VCAP_SERVICES`.

Attribute	Description
`binding_guid`	The GUID of the service binding.
`binding_name`	The name assigned to the service binding by the user.
`instance_guid`	The GUID of the service instance.
`instance_name`	The name assigned to the service instance by the user.
`name`	The `binding_name`, if it exists. Otherwise, the `instance_name`.
`label`	The name of the service offering.
`tags`	An array of strings an app can use to identify a service instance.
`credentials`	A JSON object containing the service-specific credentials needed to access the service instance.
`syslog_drain_url`	Not supported.
`volume_mounts`	Not supported.

The following example shows the value of the `VCAP_SERVICES` environment variable for bound instances of user-provided service instances.

```
VCAP_SERVICES=
{
  "user-provided": [
    {
      "binding_guid": "65ec345e-4f19-4499-ae70-a32b55c7f1cf",
      "binding_name": null,
      "credentials": {
        "hostname": "ca6efe83-8a9b-4395-98d0-124145d4e97a.mysql.service.internal",
        "jdbcUrl": "jdbc:mysql://ca6efe83-8a9b-4395-98d0-124145d4e97a.mysql.service.in
ternal:3306/service_instance_db?user=441123dedf5d4b7ab988d7fae43bc452&password=P4$$W0R
D&useSSL=false",
        "name": "service_instance_db",
        "password": "P4$$W0RD",
        "port": "3306",
        "type": "user-provided",
        "uri": "mysql://441123dedf5d4b7ab988d7fae43bc452:P4$$W0RD@ca6efe83-8a9b-4395-9
8d0-124145d4e97a.mysql.service.internal:3306/service_instance_db?reconnect=true",
        "username": "441123dedf5d4b7ab988d7fae43bc452"
      },
      "instance_guid": "0622de7e-2437-4a39-8048-a7df324c35df",
      "instance_name": "mysql",
      "label": "user-provided",
      "name": "mysql",
      "syslog_drain_url": null,
      "tags": [
        "p.mysql",
        "mysql",
        "database"
      ],
      "volume_mounts": []
    },
    {
      "binding_guid": "be7aba4d-a465-4e9d-9c01-9ce9861e68e7",
      "binding_name": "custom-binding-name",
      "credentials": {
        "some-credential": "some-value",
        "type": "user-provided"
      },
      "instance_guid": "f97f96d7-62f2-43db-866a-175f5a8e95bc",
      "instance_name": "custom-user-provided-service",
      "label": "user-provided",
      "name": "custom-binding-name",
      "syslog_drain_url": null,
      "tags": [
        "user-defined",
        "arbitrary",
        "tags"
      ],
      "volume_mounts": []
    }
  ]
}
```

Buildpacks used by Application Service Adapter

Application Service Adapter uses different buildpacks than TAS for VMs, which you need to know when moving applications between the two platforms. This topic describes the differences between the two buildpack systems.

Differences between the buildpack systems

Application platforms such as TAS for VMs and Tanzu Application Platform use buildpacks to transform application source code and other assets into a self-contained artifact, such as a Cloud Foundry droplet or an Open Container Initiative container image. The platform then deploys this artifact to run instances of the application.

TAS for VMs uses Cloud Foundry buildpacks and provides a collection of system buildpacks that process many common languages and frameworks for cloud-native web applications. For more information about Cloud Foundry buildpacks, see How Buildpacks Work in the TAS for VMs documentation.

Tanzu Application Platform instead uses Tanzu Buildpacks, which implement the later Cloud Native Buildpack specification to process application source code. For more information about Cloud Native Buildpacks, see the Cloud Native Buildpacks project website.

An installation of Tanzu Application Platform provides a default collection of Tanzu Buildpacks, and Application Service Adapter uses this same set of Tanzu Buildpacks to stage Cloud Foundry applications. Although this collection of Tanzu Buildpacks processes many of the same application languages and frameworks that the TAS for VMs system buildpacks do, there are some differences in the set of languages it can process, as well as differences in how certain buildpacks detect application code or accept configuration parameters.

System buildpack comparison

TAS for VMs provides the following system buildpacks for Linux-based applications:

- Binary buildpack
- Go buildpack
- Java buildpack
- .NET Core buildpack
- NGINX buildpack
- Node.js buildpack
- PHP buildpack
- Python buildpack
- R buildpack
- Ruby buildpack
- Staticfile buildpack

For many applications using these buildpacks on TAS for VMs, the suite of Tanzu Buildpacks contains a cloud-native buildpack that can process its language, frameworks, and build configuration. Tanzu Buildpacks are not currently available for R applications, although community-based solutions might be available.

Binary buildpack

Applications that use the binary buildpack in TAS for VMs should use the Tanzu Procfile Buildpack with Application Service Adapter. Applications must include a Procfile that defines the process types and their start commands if they do not already.

Go buildpack

Applications that use the Go buildpack in TAS for VMs should use the Tanzu Go Buildpack with Application Service Adapter.

Java buildpack

Applications that use the Java buildpack in TAS for VMs should use the Tanzu Java Buildpack or the Tanzu Java Native Image Buildpack with Application Service Adapter.

.NET Core buildpack

Applications that use the .NET Core buildpack in TAS for VMs should use the Tanzu .NET Core Buildpack with Application Service Adapter. For more information about migrating an application to use this buildpack, see Migrating to the Tanzu .NET Core Buildpack in the Tanzu Buildpacks documentation.

NGINX buildpack

Applications that use the NGINX buildpack in TAS for VMs should use the Tanzu NGINX Buildpack with Application Service Adapter.

Node.js buildpack

Applications that use the Node.js buildpack in TAS for VMs should use the Tanzu Node.js Buildpack with Application Service Adapter. For more information about migrating an application to use this buildpack, see Migrating to the Tanzu Node.js Buildpack in the Tanzu Buildpacks documentation.

PHP buildpack

Applications that use the PHP buildpack in TAS for VMs should use the Tanzu PHP Buildpack with Application Service Adapter. For more information about migrating an application to use this buildpack, see Migrating to the Tanzu PHP Buildpack in the Tanzu Buildpacks documentation.

Python buildpack

Applications that use the Python buildpack in TAS for VMs should use the Tanzu Python Buildpack with Application Service Adapter.

Ruby buildpack

Applications that use the Ruby buildpack in TAS for VMs should use the Tanzu Ruby Buildpack with Application Service Adapter.

Staticfile buildpack

Applications that use the Staticfile buildpack in TAS for VMs should use the Tanzu Web Servers Buildpack with Application Service Adapter.

cf CLI commands supported by Application Service Adapter

In this topic, you can learn about the Cloud Foundry command-line interface (cf CLI) commands and options that are supported by Application Service Adapter.

Application Service Adapter supports the following cf CLI commands by providing a subset of the endpoints the v3 CF API through its API server.

Getting started

Command	Supported?	Notes
`cf login`	Y	Using the `--sso` flag is not supported.
`cf logout`	Y	
`cf target`	Y	
`cf passwd`	N	
`cf api`	Y	
`cf auth`	Y	Use the name of a user authentication info entry from the local Kubeconfig file as the username argument. The password field is ignored.

cf push

Application Service Adapter supports the basic use of `cf push APP-NAME`, where `APP-NAME` is the name of your app.

Flag	Supported?	Notes
`--app-start-timeout, -t`	Y	
`--buildpack, -b`	N	Automatic buildpack detection is supported. Omit the flag or set to `null` or `default`. User-specified buildpacks are not supported.
`--disk, -k`	Y	
`--docker-image, -o`	N	
`--docker-username`	N	
`--droplet`	N	
`--endpoint`	N	
`--health-check-type, -u`	Y	
`--instances, -i`	Y	
`--manifest, -f`	Y	Some manifest configurations described in App Manifest Attribute Reference are supported. See the Supported manifest configuration section.
`--memory, -m`	Y	

Flag	Supported?	Notes
`--no-manifest`	Y	
`--no-route`	Y	
`--no-start`	Y	
`--no-wait`	N	
`--path, -p`	Y	
`--random-route`	Y	
`--stack, -s`	N	
`--start-command, -c`	Y	
`--strategy`	N	
`--task`	Y	
`--var`	Y	
`--vars-file`	Y	

Supported manifest configuration

The Application Service Adapter supports a subset of the Cloud Foundry manifest schema. For more information, see The manifest schema in the Cloud Foundry API documentation.

Supported app-level configuration

Attribute	Supported?	Notes
`buildpacks`	N	
`command`	Y	
`default-route`	Y	
`disk_quota`	Y	
`docker`	N	
`env`	Y	
`health-check-http-endpoint`	Y	
`health-check-invocation-timeout`	Y	
`health-check-type`	N	
`instances`	Y	
`memory`	Y	
`metadata`	N	
`no-route`	Y	

Attribute	Supported?	Notes
`path`	N	
`processes`	Y	See the Supported process-level configuration section.
`random-route`	Y	
`routes`	Y	See the Supported route-level configuration section.
`services`	N	
`sidecars`	N	
`stack`	N	
`timeout`	Y	

Supported process-level configuration

Attribute	Supported?	Notes
`type`	Y	
`command`	Y	
`disk_quota`	Y	
`docker`	N	
`health-check-http-endpoint`	Y	
`health-check-invocation-timeout`	Y	
`health-check-type`	Y	
`instances`	Y	
`memory`	Y	
`timeout`	Y	

Supported route-level configuration

Attribute	Supported?	Notes
`route`	Y	
`protocol`	N	Only `http1` routes are supported.

App operations

Command	Supported?	Notes
`cf apps`	Y	The default usage of `cf apps` is supported. Using `--labels` to filter apps is not supported.
`cf app`	Y	

Command	Supported?	Notes
`cf create-app`	Y	
`cf scale`	Y	
`cf delete`	Y	
`cf rename`	N	
`cf cancel-deployment`	N	
`cf start`	Y	
`cf stop`	Y	
`cf restart`	Y	
`cf stage-package`	Y	
`cf restage`	Y	Using `--strategy` flag is not supported.
`cf restart-app-instance`	N	
`cf run-task`	Y	Only the `-c` parameter is supported.
`cf tasks`	Y	
`cf terminate-task`	Y	
`cf packages`	Y	
`cf create-package`	Y	
`cf droplets`	N	
`cf set-droplet`	Y	
`cf download-droplet`	N	
`cf events`	N	
`cf logs`	Y	
`cf env`	Y	Fetching `system-provided`, `running`, and `staging` environment variables are not supported.
`cf set-env`	Y	
`cf unset-env`	Y	
`cf stacks`	N	
`cf stack`	N	
`cf copy-source`	N	
`cf create-app-manifest`	N	
`cf get-health-check`	Y	
`cf set-health-check`	Y	

Command	Supported?	Notes
`cf enable-ssh`	N	
`cf disable-ssh`	N	
`cf ssh-enabled`	N	
`cf ssh`	N	

Org and space operations

This section describes the org and space operations that Application Service Adapter supports.

Org operations

Command	Supported?	Notes
`cf orgs`	Y	
`cf org`	Y	Using the `--guid` flag to retrieve the GUID of the org is supported. Using the command without the `--guid` flag is not supported.
`cf create-org`	Y	
`cf delete-org`	Y	
`cf rename-org`	N	

Space operations

Command	Supported?	Notes
`cf spaces`	Y	
`cf space`	Y	Using the `--guid` flag to retrieve the GUID of the space is supported. Using the command without the `--guid` flag is not supported.
`cf create-space`	Y	
`cf delete-space`	Y	
`cf rename-space`	N	
`cf apply-manifest`	Y	
`cf allow-space-ssh`	N	
`cf disallow-space-ssh`	N	
`cf space-ssh-allowed`	N	

Route and domain operations

This section describes the route and domain operations that Application Service Adapter supports.

Route operations

Command	Supported?	Notes
`cf routes`	Y	
`cf route`	Y	
`cf create-route`	Y	
`cf check-route`	N	
`cf map-route`	Y	
`cf unmap-route`	Y	
`cf delete-route`	Y	
`cf delete-orphaned-routes`	N	

Domain operations

Command	Supported?	Notes
`cf domains`	Y	
`cf create-private-domain`	N	
`cf delete-private-domain`	N	
`cf create-shared-domain`	N	
`cf delete-shared-domain`	N	
`cf router-groups`	N	

Service operations

This section describes the service operations that Application Service Adapter supports.

Command	Supported?	Notes
`cf marketplace`	N	
`cf services`	Y	
`cf service`	Y	
`cf create-service`	N	
`cf update-service`	N	
`cf upgrade-service`	N	
`cf delete-service`	Y	
`cf rename-service`	N	
`cf create-service-key`	N	
`cf bind-service`	Y	
`cf unbind-service`	Y	

Command	Supported?	Notes
`cf delete-service-key`	N	
`cf bind-route-service`	N	
`cf unbind-route-service`	N	
`cf create-user-provided-service`	Y	
`cf update-user-provided-service`	N	
`cf share-service`	N	
`cf unshare-service`	N	

Metadata operations

This section describes the metadata operations that Application Service Adapter supports.

Command	Supported?	Notes
`cf labels`	Y	The `app`, `org`, and `space` resources are supported.
`cf set-label`	Y	The `app`, `org`, and `space` resources are supported.
`cf unset-label`	Y	The `app`, `org`, and `space` resources are supported.

Troubleshoot Application Service Adapter

Here you'll find generic techniques for diagnosing Application Service Adapter system health and troubleshooting steps in common failure scenarios.

Generic troubleshooting techniques

This section contains generic techniques that can be used to gather information about the status of the Application Service Adapter to aid in troubleshooting. We recommend keeping up to date with the latest version of the Application Service Adapter to get access to the latest bug fixes and stability improvements.

Application Service Adapter logs

There are several Application Service Adapter deployments whose logs can be queried to gather information on failures that occurred within the system.

To fetch recent logs from all components of an Application Service Adapter installation:

~~~
```bash
kapp logs --app tas-adapter.app -n tap-install
```
~~~

To fetch recent logs from a specific deployment:

~~~
```bash
kapp logs --app tas-adapter.app -n tap-install --pod-name DEPLOYMENT-NAME%
~~~

~~~
```
Where DEPLOYMENT-NAME is the name of the Kubernetes Deployment (e.g.
"korifi-api-deployment").
~~~

To stream logs instead of fetching the most recent logs, add the `--follow` flag to the above `kapp logs` commands. For additional details and options, refer to the `kapp logs --help` help text.

Deployments

The following is a brief description of the specific Application Service Adapter deployments and what their main responsibilities are to help isolate which logs to query when troubleshooting.

1. The `korifi-api-deployment` Deployment is tasked with responding to API requests sent to the Application Service Adapter. Logs for failures related to the image registry may also show up in this Deployment's logs.
2. The `korifi-controllers-controller-manager` Deployment is tasked with processing commands for the Application Service Adapter. This Deployment's logs is an excellent starting point when debugging failures as it is where most commands flow through before being processed by more specific components (The exception being API commands that fail before getting to the controller manager).
3. The `tas-adapter-telemetry-informer` Deployment is tasked with handling all outgoing telemetry.
4. The `korifi-kpack-build-controller-manager` Deployment is tasked with building Images when using the default builder/runner flow.
5. The `korifi-statefulset-runner-controller-manager` Deployment is tasked with generating StatefulSets for apps when using the default builder/runner flow.
6. The `korifi-job-task-runner-controller-manager` Deployment is tasked with handling all task related functionality.
7. The `cartographer-builder-runner-controller-manager` Deployment is tasked with creating Cartographer Workloads for apps when using the experimental Cartographer builder/runner flow.

Tanzu Application Platform logs

While not directly part of the Application Service Adapter, there are several Tanzu Application Platform deployments that are utilized by the Application Service Adapter and can also provide further debug information on failures that occur.

To fetch recent logs from a given Tanzu Application Platform application:

~~~
```bash
kapp logs --app APP-NAME -n tap-install
```

Where APP-NAME is the name of the Tanzu Application Platform application. For example,
"buildservice.app".
~~~

To stream logs instead of fetching the most recent logs, add the `--follow` flag to the above `kapp logs` command. For additional details and options, refer to the `kapp logs --help` help text.

Deployments

The following is a brief description of the specific Tanzu Application Platform deployments used by the Application Service Adapter and what their main responsibilities are to help determine which logs to query when troubleshooting.

1. The `buildservice.app` application is tasked with processing any Tanzu Build Service (kpack) commands. If any failures in the build image process occurs in Tanzu Build Service itself, this application's logs can provide further information.
2. The `contour.app` application is tasked with creating an ingress into the system. If a failure to connect to the Application Service Adapter or an Application occurs, this application's logs can provide further information.
3. The `cartographer.app` application is tasked with processing Cartographer ClusterSupplyChains when using the experimental Cartographer builder/runner flow. If a failure to create an Image, Build, ConfigMap, or StatefulSet occurs, this application's logs can provide further information.

CFApp Logs

For CFApps that have been staged, the cf logs command can be used to get more information in the event of run failures:

```
cf logs <CFApp name>
```

Kubernetes System Events

Kubernetes system events can provide debug information in cases where system restrictions prevent objects from being created. These events may be caused by insufficient resources or evicted pods, for example. The logs for the system can be queried with the following command:

```
kubectl get events -A
```

System Object Types

There are system objects created/used by Application Service Adapter that can be useful in debugging failures. Describing the objects can show error messages in their status fields. Whether objects are present can also be useful in determining where in the command process failures have occurred and which logs should be queried for more information.

For example, if an object in the image build chain is missing, the component in the "Created By" column can be queried for logs to see if any useful error messages are present. If no useful logs are found there, identify the previous object in the image build chain and check the logs of the component listed in the "Reconciled By" column to see if there are any error messages.

Instructions for querying logs can be found for Application Service Adapter and Tanzu Application Platform.

Some Kubernetes objects exist within a CFOrg or CFSpace namespace. To find the corresponding namespace of a CFOrg or CFSpace, you can run:

```
cf org --guid <CFOrg name>
cf space --guid <CFSpace name>
```

Installed Object Types

Kind	Component	Namespace or Scope	Notes
ClusterBuilder	kpack	Cluster	ClusterBuilders are Tanzu Build Service (kpack) components installed by Tanzu Application Platform that are used to build images. If they are not present, image builds may not complete successfully.
Deployment	kubernetes	`tas-adapter-system`	All of the Application Service Adapter Deployment's `Ready` counts should be listed as `1/1` except for `korifi-api-deployment` which should be `2/2`. Should this not be the case, the system is not functioning correctly and the status/events fields of the inoperable Deployment should be analyzed. This issue can sometimes be related to RBAC.
ClusterRole	kubernetes	Cluster	ClusterRoles are created to manage system privileges at the cluster scope.
ClusterRoleBinding	kubernetes	Cluster	ClusterRoleBindings are created to manage system privileges at the cluster scope.
Service Account	kubernetes	`cf`	ServiceAccounts are created to manage CF permissions. Some ServiceAccounts are designed to be propagated to CFOrg/CFSpace namespaces. These objects will be designated with the `cloudfoundry.org/propagate-service-account: "true"` annotation.
RoleBinding	kubernetes	`cf`	RoleBindings are created to manage CF permissions. Some RoleBindings are designed to be propagated to CFOrg/CFSpace namespaces. These objects will be designated with the `cloudfoundry.org/propagate-cf-role: "true"` annotation.
Security Context Constraint	openshift	Cluster	SecurityContextConstraints are specific to OpenShift and are used to grant required permissions to Application Service Adapter components in that environment. These CRDs do not exist in non-OpenShift environments. An associated ClusterRole, ClusterRoleBinding, and RoleBinding will also be installed if the `kubernetes_distribution` shared setting is set to `openshift`.
ClusterSupplyChain	Cartographer	Cluster	When using the experimental Cartographer builder/runner flow, a ClusterSupplyChain is created during Application Service Adapter installation at the cluster scope. This is what the `cartographer-ctrl` uses to creates the Image, Build, ConfigMap, and StatefulSets in the builder/runner flow.

CFOrg/CFSpace Object Types

Kind	Component	Namespace or Scope	Created By	Reconciled By	Notes
CFOrg	CF	`cf`	`korifi-api-deployment`	`korifi-controllers-controller-manager`	CFOrg resources are created by the `cf create-org` command. A corresponding namespace should also be created by the `korifi-controllers-controller-manager`.

Kind	Component	Namespace or Scope	Created By	Reconciled By	Notes
CFSpace	CF	CFOrg Namespace	`korifi-api-deployment`	`korifi-controllers-controller-manager`	CFSpace resources are created by the `cf create-space` command. A corresponding namespace should also be created by the `korifi-controllers-controller-manager`.

Image Build Object Types (Default Builder/Runner)

All image build object types are located within the targeted CFSpace Namespace.

Kind	Component	Created By	Reconciled By	Notes
CFApp	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 1st object in the image build chain that's created when you run the `cf push` command. The CFApp resource represents the application.
CFPackage	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 2nd object in the image build chain that's created when you run the `cf push` command. The CFPackage resource contains a reference to the uploaded app source code.
CFBuild	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 3rd object in the image build chain that's created when you run the `cf push` command. The CFBuild resource initiates the build and eventually contains a reference to the runnable app image.
BuildWorkload	Cloud Foundry	`korifi-controllers-controller-manager`	`korifi-controllers-controller-manager`	The 4th object in the image build chain that's created when you run the `cf push` command. The BuildWorkload is an intermediate resource that contains information about the build.
Image	kpack	`korifi-controllers-controller-manager`	`buildservice-ctrl`	The 5th object in the image build chain that's created when you run the `cf push` command. The full object path for image objects are kpack.io/v1alpha2/images. This may need to be specified when querying in certain environments like OpenShift since there are other resources with the same name. The image is a kpack resource that provides configuration to build and maintains a docker image utilizing Cloud Native Buildpacks.
Build	kpack	`buildservice-ctrl`	NA	The 6th object in the image build chain that's created when you run the `cf push` command. The full object path for build objects are kpack.io/v1alpha2/builds. This may need to be specified when querying in certain environments like OpenShift since there are other resources with the same name. The build is a kpack resource that schedules and runs a single Cloud Native Buildpacks build. The build object spawns a build pod.

Run App Object Types (Default Builder/Runner)

All run app object types are located within the targeted CFSpace Namespace.

Kind	Component	Created By	Reconciled By	Notes
CFProcess	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 1st object in the app run chain that's created when you run the `cf push` command. The CFProcess resource represents a single process.
AppWorkload	Cloud Foundry	`korifi-controllers-controller-manager`	`korifi-controllers-controller-manager`	The 2nd object in the app run chain that's created when you run the `cf push` command. The AppWorkload is an intermediate resource that contains information about the application/process.
StatefulSet	Kubernetes	`korifi-controllers-controller-manager`	NA	The 3rd object in the app run chain that's created when you run the `cf push` command. The StatefulSet is a Kubernetes resource that represents a single process for an application. There must be a StatefulSet for each CFProcess.
ReplicaSet	Kubernetes	Kubernetes	NA	The 4th object in the app run chain that's created when you run the `cf push` command. The ReplicaSet is created by Kubernetes to maintain a stable set of replica pods for the StatefulSet.
Pod	Kubernetes	Kubernetes	NA	The app process pods are last in the app run chain and Kubernetes creates them based on the ReplicaSet. The number of pods are based on the ReplicaSet replicas parameter.
CFRoute	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	CFRoutes are created to reach running apps. They can be created as part of the `cf push` command if a default or random route is specified, defined in a manifest, or as part of the `cf create-route` command.
HTTPProxy	contour	`korifi-controllers-controller-manager`	NA	The HTTPProxy is a Contour resource created by the `korifi-controllers-controller-manager` to reach running apps.

Image Build Object Types (Cartographer Builder/Runner)

All image build object types are located within the targeted CFSpace Namespace.

Kind	Component	Created By	Reconciled By	Notes
CFApp	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 1st object in the image build chain that's created when you run the `cf push` command. The CFApp resource represents the application.
CFPackage	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 2nd object in the image build chain that's created when you run the `cf push` command. The CFPackage resource contains a reference to the uploaded app source code.
CFBuild	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 3rd object in the image build chain that's created when the `cf push` command is run. The CFBuild resource initiates the build and eventually contains a reference to the runnable app image.

Kind	Component	Created By	Reconciled By	Notes
BuildWorkload	Cloud Foundry	`korifi-controllers-controller-manager`	`cartographer-builder-runner-controller-manager`	The 4th object in the image build chain that's created when you run the `cf push` command. The BuildWorkload is an intermediate resource that contains information about the build.
Workload	Cartographer	`cartographer-builder-runner-controller-manager`	`cartographer-ctrl`	The 5th object in the image build chain that's created when you run the `cf push` command. The workload is a Cartographer resource that contains information used by the `buildservice-ctrl` in conjunction with the SupplyChain to create an image and later a ConfigMap containing the StatefulSet definition.
Image	kpack	`cartographer-ctrl`	`buildservice-ctrl`	The 6th object in the image build chain that's created when you run the `cf push` command. The full object path for image objects are kpack.io/v1alpha2/images. This may need to be specified when querying in certain environments like OpenShift since there are other resources with the same name. The image is a kpack resource that provides configuration to build and maintain a docker image utilizing Cloud Native Buildpacks.
Build	kpack	`buildservice-ctrl`	NA	The 7th object in the image build chain that's created when you run the `cf push` command. The full object path for build objects are kpack.io/v1alpha2/builds. This may need to be specified when querying in certain environments like OpenShift since there are other resources with the same name. The build is a kpack resource that schedules and runs a single Cloud Native Buildpacks build. The build object spawns a build pod.

Run App Object Types (Cartographer Builder/Runner)

All run app object types are located within the targeted CFSpace Namespace.

Kind	Component	Created By	Reconciled By	Notes
CFProcess	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 1st object in the app run chain that's created when you run the `cf push` command. The CFProcess resource represents a single process.
AppWorkload	Cloud Foundry	`korifi-controllers-controller-manager`	`cartographer-builder-runner-controller-manager`	The 2nd object in the app run chain that's created when you run the `cf push` command. The AppWorkload is an intermediate resource that contains information about the application/process. `cartographer-builder-runner-controller-manager` edits the Cartographer workload created during the image build process with the information from the AppWorkload.
PodIntent	Cartographer	App Operator	`conventions-controller-manager`	An optional object in the app run chain. Convention Service enables app operators to consistently apply desired runtime configurations to fleets of workloads.

Kind	Component	Created By	Reconciled By	Notes
ConfigMap	Kubernetes	`cartographer-ctrl`	NA	The 3rd object in the app run chain that's created when you run the `cf push` command. A Kubernetes ConfigMap resource containing the StatefulSet definition is created that `cartographer-ctrl` uses to spawns a StatefulSet by means of kapp.
StatefulSet	Kubernetes	`cartographer-ctrl`	NA	The 4th object in the app run chain that's created when you run the `cf push` command. The StatefulSet is a Kubernetes resource that represents a single process for an application. There should be a StatefulSet for each CFProcess.
ReplicaSet	Kubernetes	Kubernetes	NA	The 5th object in the app run chain that's created when you run the `cf push` command. Kubernetes creates the ReplicaSet to maintain a stable set of replica pods for the StatefulSet.
Pod	Kubernetes	Kubernetes	NA	The app process pods are last in the app run chain and Kubernetes creates them based on the ReplicaSet. The number of pods are based on the ReplicaSet replicas parameter.
CFRoute	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	CFRoutes are created to be able to reach running apps. They can be created as part of the `cf push` command if a default or random route is specified, defined in a manifest, or as part of the `cf create-route` command.
HTTPProxy	contour	`korifi-controllers-controller-manager`	NA	The HTTPProxy is a Contour resource that is created by the `korifi-controllers-controller-manager` to reach running apps.

Run Task Object Types

All run task object types are located within the targeted CFSpace Namespace.

Kind	Component	Created By	Reconciled By	Notes
CFTask	Cloud Foundry	`korifi-api-deployment`	`korifi-controllers-controller-manager`	The 1st object in the run task chain that's created when you run the `cf run-task` command.
TaskWorkload	Cloud Foundry	`korifi-controllers-controller-manager`	`korifi-controllers-controller-manager`	The 2nd object in the run task chain that's created when you run the `cf run-task` command.
Job	Kubernetes	`korifi-controllers-controller-manager`	NA	The 3rd object in the run task chain that's created when you run the `cf run-task` command.

Kind	Component	Created By	Reconciled By	Notes
Pod	Kubernetes	Kubernetes	NA	The running job pods are last in the run task chain, and Kubernetes creates them based on the job. The job pod completes when its command is run.

Common Failure Scenarios

This section contains common failure scenarios and describes the appropriate troubleshooting techniques that can be used to gather further information and solve the issue.

Organization not found when creating an Org.

Symptom

When I run `cf create org my-org`, the following error message is returned:

```
Organization 'my-org' not found
```

Possible Causes

The user creating the CFOrg is not bound to the `korifi-controllers-admin` ClusterRole.

Troubleshooting Steps/Potential Solutions

1. Verify the user in the `cf create org` command output.

```
Creating org my-org as cf-admin...
```

1. Verify that there is a RoleBinding in the `cf` namespace binding the `korifi-controllers-admin` ClusterRole to the given user and that it has the `cloudfoundry.org/propagate-cf-role: "true"` annotation.
2. If that RoleBinding does not exist, either switch to a user that is bound to the `korifi-controllers-admin` ClusterRole or create a RoleBinding as a cluster admin.

Pushing an app fails to build an image

Symptom

When I run `cf push`, the CF CLI does not respond with a built image.

Possible causes

1. ClusterBuilder is not ready.
2. A failure occurred in one of the Tanzu Application Platform or Application Service Adapter components.

Troubleshooting Steps/Potential Solutions

1. Check the status of the ClusterBuilders by running `kubectl get clusterbuilder`. The `Ready` column should show `True`.
2. If the ClusterBuilder's `Ready` state is not `True`, run `kubectl describe clusterbuilder` and check the status section for more information.
3. Walk through the list of image build object types and check if any objects are missing. For any missing objects, check the logs of the Tanzu Application Platform or Application Service Adapter component listed in the "Created By" column to see if there are any error messages.
4. If the component listed in the "Created By" column does not have any helpful error messages, identify the previous object in the image build chain and check the logs of the component listed in the "Reconciled By" column to see if there are any error messages.

Pushing an app fails to upload an image to the image registry

Symptom

When I run `cf push`, the following output/error message is returned:

```
    Unexpected Response
    Response Code: 500
    Code: 0, Title: , Detail: {"errors":[{"detail":"An unknown error occurred.","titl
e":"UnknownError","code":10001}]}
```

Possible Causes

The image registry is unavailable/unauthorized.

Troubleshooting Steps/Potential Solutions

1. Check the `korifi-api-deployment` logs for registry related failures like `failed to upload image`.
2. Determine the cause of the upload failure like connection refused, credentials failed, certificate untrusted, etc.
3. Check the image registry settings provided to the Application Service Adapter installation related to the failure and reinstall Application Service Adapter with the correct image registry settings.

Pushing an app fails to start

Symptom

When I run `cf push`, my app stages correctly, but fails to start or become healthy.

Possible Causes

1. There is an issue with the application's code.
2. The Kubernetes cluster cannot schedule the application.
3. The Application Service Adapter cannot turn the app into a StatefulSet.

Troubleshooting Steps/Potential Solutions

1. To debug the application's code, check the application logs
2. To debug applications that fail to be scheduled, check the Kubernetes System Logs and the StatefulSet status in the CFSpace namespace.
3. Walk through the list of run app object types and check if any objects are missing. For any missing objects, check the logs of the Tanzu Application Platform or Application Service Adapter component listed in the "Created By" column to see if there are any error messages.
4. If the component listed in the "Created By" column does not have any helpful error messages, identify the previous object in the run app chain and check the logs of the component listed in the "Reconciled By" column to see if there are any error messages.

Deployed apps fail to become routable

Symptom

When I run `cf push`, my app stages and runs, but fails to be routable.

Possible Causes

1. There is an issue with the Application Service Adapter Deployments
2. There is an issue with the cluster's routing.

Troubleshooting Steps/Potential Solutions

1. Check the logs for the `korifi-controllers-controller-manager` and see if there is any additional information.
2. Check the status of the CFRoute corresponding to the app's route in the CFSpace namespace and see if there are any error messages.
3. Check the status of the HTTPProxy corresponding to the app's route in the CFSpace namespace and see if there are any error messages.

OpenShift Failure Scenarios

This section contains common failure scenarios specific to operation on the OpenShift platform and describes the appropriate troubleshooting techniques that can be used to gather further information and solve the issue.

OpenShift Setting

The OpenShift installation setting for Application Service Adapter can be set in the `tas-adapter-values.yml` as described here.

Installing the Application Service Adapter on a non-OpenShift Kubernetes distribution fails

Symptom

When I install the Application Service Adapter on a non-OpenShift cluster, I get the following error message:

```
kapp: Error: Expected to find kind 'security.openshift.io/v1/SecurityContextConstraint
s', but did not:
- Kubernetes API server did not have matching apiVersion + kind
- No matching CRD was found in given configuration
```

Possible Causes

The OpenShift setting is enabled.

Troubleshooting Steps/Potential Solutions

Creating the SecurityContextConstraint will fail on a non-OpenShift cluster because the SecurityContextConstraint CRD is only present on OpenShift. Reinstall Application Service Adapter with the OpenShift setting turned off.

Installing the Application Service Adapter on an OpenShift Kubernetes distribution fails

Symptom

When I install the Application Service Adapter on an OpenShift cluster, I get the following error message:

```
1:41:32PM:  ^ Retryable error: Creating resource cfdomain/apps.openshift-aro.k8s-dev.r
elint.rocks (korifi.cloudfoundry.org/v1alpha1) namespace: cf: API server says: Interna
l error occurred: failed calling webhook "vcfdomain.korifi.cloudfoundry.org": failed t
o call webhook: Post "https://korifi-controllers-webhook-service.tas-adapter-system.sv
c:443/validate-korifi-cloudfoundry-org-v1alpha1-cfdomain?timeout=10s": no endpoints av
ailable for service "korifi-controllers-webhook-service" (reason: InternalError)
```

Possible Causes

The OpenShift setting is not enabled.

Troubleshooting Steps/Potential Solutions

Without the SecurityContextConstraint/ClusterRole/ClusterRoleBinding/RoleBinding, the Application Service Adapter deployments will not be able to deploy Pods.

1. Query the Application Service Adapter Deployments and check if their `Ready` counts are listed as `0/0`.
2. Reinstall Application Service Adapter with the OpenShift setting enabled.

Pushing an app on an OpenShift cluster fails to start

Symptom

When I run `cf push` on an OpenShift cluster, my app stages correctly, but fails to start or become healthy.

Possible Causes

The ServiceAccount being used by the StatefulSets does not have permission to create pods.

Troubleshooting Steps/Potential Solutions

1. Query the StatefulSet and check if their `Ready` counts are listed as `0/0`.
2. Verify that the `ServiceAccount` field is set to `korifi-app`.
3. Verify that there is a RoleBinding in the CFSpace namespace that links `system:tas-adapter:scc:tas-adapter-scc` to `korifi-app`.
4. If any of those objects are not present, reinstall Application Service Adapter with the OpenShift setting enabled.

Cartographer Failure Scenarios

This section contains common failure scenarios specific to the use of the experimental optional Cartographer feature and describes the appropriate troubleshooting techniques that can be used to gather further information and solve the issue.

Cartographer setting

The experimental Cartographer installation setting for Application Service Adapter can be set in the `tas-adapter-values.yaml`, as described in Install Application Service Adapter.

Pushing an app fails to start

Symptom

When I run `cf push`, my app stages correctly, but fails to start or become healthy.

Possible Causes

There is a failure in the `cartographer-builder-runner-controller-manager`, the SupplyChain, or the `cartographer-ctrl`.

Troubleshooting Steps/Potential Solutions

1. Check the status of the Workload and see if there are any error messages.
2. Check the status of the BuildWorkload and see if there are any error messages.
3. Check the status of the AppWorkload and see if there are any error messages.
4. Check the logs for the `cartographer-builder-runner-controller-manager` and see if there is any additional information.
5. Verify that there is a cluster-scoped SupplyChain present.
6. Check the logs for the `cartographer-ctrl` and see if there is any additional information.

User management with Application Service Adapter

Application Service Adapter allows you to manage users. This topic tells you how.

Application Service Adapter users are user identifiers that Kubernetes recognizes in the subject section of its role-based access control (RBAC) resources, such as RoleBindings. For more information about user subject names in Kubernetes, see the Referring to subjects section of *Using RBAC Authorization* and the Authenticating topic in the Kubernetes project documentation.

Users can be assigned Cloud Foundry roles using the role management commands of the cf CLI or by directly creating RoleBinding resources through the Kubernetes API.

AWS IAM user management for EKS

To configure an AWS IAM user for an Elastic Kubernetes Service (EKS) cluster, you must configure the `aws-auth` ConfigMap on the EKS cluster to map IAM resources by ARN to the cluster. Follow the AWS IAM user and role access documentation for more information.

Note

The AWS documentation recommends using `eksctl` to edit the ConfigMap.