

VMware Tanzu Build Service 1.6 Documentation

Tanzu Build Service 1.6

You can find the most up-to-date technical documentation on the VMware website at:
<https://docs.vmware.com/>

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

Copyright © 2022 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

Tanzu Build Service 1.6 Documentation	10
Overview	10
Build Service Concepts	10
Image Resource	10
Builder	10
ClusterStore	10
ClusterStack	10
Build Service components	11
Build Service Dependencies	11
Buildpacks	11
Stacks	11
Descriptors	11
Updating Build Service Dependencies	11
Troubleshooting	11
TBS Release Notes	13
v1.6.1	13
New Features	13
Breaking Changes	13
Product Snapshot	13
Security Scanning	13
Product Dependencies	13
Upgrade Path	13
v1.6.0	13
New Features	13
Breaking Changes	13
Product Snapshot	14
Security Scanning	14
Product Dependencies	14
Upgrade Path	14
Installing Tanzu Build Service	15
Prerequisites	15
Install Tanzu Build Service	16
Relocate images to a registry	17

Installation	18
Using Secrets for credentials	19
Additional configuration	20
Using AWS IAM for Authentication with ECR	20
Install Tanzu Build Service with AWS IAM for Authentication with ECR	20
Configure the build namespace for IAM Auth with ECR	21
Upgrading Tanzu Build Service	21
Installing Tanzu Build Service (air-gapped)	22
Relocate images to a registry (air-gapped)	22
Installation (air-gapped)	23
Additional configuration	24
Updating Build Service Dependencies (air-gapped)	25
Upgrading Tanzu Build Service (air-gapped)	25
Uninstalling Tanzu Build Service	25
Ensuring access to cluster builders	25
Ensuring the run image is readable	25
Next steps	26
Minimum AWS IAM permissions for ECR IAM auth	26
Installing Windows Components (Beta)	27
Getting Started with Tanzu Build Service	27
Prerequisites	28
Assumptions	28
Installation	29
Relocate Images to a Registry	29
Install Tanzu Build Service	29
Verify Installation	30
Create an Image	30
Installing Tanzu Build Service without kapp controller	30
Prerequisites	31
Installing	32
Relocate images to a registry	32
Install Tanzu Build Service	34
Install Tanzu Build Service Public Registry	34
Installing with a CA certificate for internal registry	35
Additional Configuration	36
Use AWS IAM for Authentication with ECR	36
Install Tanzu Build Service with AWS IAM for Authentication with ECR	36

Configure the build namespace for IAM Auth with ECR	37
Configuring TKGI as an OIDC Provider	38
Installation to Air-Gapped Environment	38
Relocate Images to a Registry (Air-Gapped)	39
Installing (Air-Gapped)	40
Additional Configuration	41
Import Tanzu Build Service Dependencies (Air-Gapped)	41
Relocate Tanzu Build Service Dependency Images (Air-Gapped)	41
Import Tanzu Build Service Dependency Resources (Air-Gapped)	42
Verify Installation	42
Upgrading Tanzu Build Service	42
Uninstalling Tanzu Build Service	42
Updating Build Service Dependencies	43
Ensuring Access to Cluster Builders	43
Ensuring the Run Image is Readable	43
Next Steps	43
Kubernetes Permissions for Installation	43
Minimum AWS IAM permissions for ECR IAM auth	46
Accessing Tanzu Build Service	48
Updating Build Service Dependencies	49
Updating dependencies	49
Automatically update dependencies	49
Bulk update	50
Cluster stacks update	51
Cluster store update	52
Updating buildpacks From Tanzu Network	52
Offline update of dependencies	53
Managing Secrets	55
Overview	55
Create Secrets	55
Create a Docker Hub registry secret	55
Create a GCR registry secret	56
Create an Artifactory, Harbor, or ACR registry secret	56
Create a Git SSH secret	57
Create a Git basic auth secret	57
List secrets	58
Delete secrets	58

Encrypting secrets at rest	59
Using SecretGen controller Secrets to use private cluster builders	59
When to use Synchronized Secrets	59
Installing the Carvel secret-gen-controller	59
Managing secret synchronization	60
Create a synchronized secret	60
Update a synchronized secret	60
Stop synchronizing a secret	60
Managing Image Resources and Builds	61
Image Resources	62
Creating Image Resources	62
Source Code	62
Builders	62
Creating an image resource with source code in a Git repository	62
Create an image resource with source code in a blob store	63
Creating an image resource with local source code	64
Buildpack configuration	65
Buildpack configuration use cases	65
Buildpack configuration documentation	65
Buildpack configuration in image resources	65
Patching image resources	66
Saving image resources	67
Listing images	67
Filter image resources	68
Image resource rebuilds	68
Trigger an image resource rebuild	69
Viewing the status of an image resource	69
Deleting an image resource	70
Managing image resources with YAML	70
Image resource additional tags	70
Using a registry for caching	70
Using secrets	70
Debugging with image resource status	70
Image resource service bindings	71
Creating an image resource with service bindings	71
Builds	72

Listing builds	72
Viewing build details for an image	73
Image resource status shows ImagePullBackOff	75
Getting build logs	75
Viewing bill of materials	77
Offline builds	78
Image signing with cosign	78
Cosign signing secret	78
Adding cosign annotations	78
Push cosign signature to a different location	79
Cosign legacy Docker media types	79
Managing ClusterStacks	81
Create a ClusterStack	82
Update a ClusterStack	82
Save a ClusterStack	83
Get ClusterStack status	83
Delete a ClusterStack	83
List all ClusterStacks	83
How to update an image for Stack updates only?	84
Managing Stores	85
Creating buildpacks and buildpackages	85
Listing ClusterStores	85
Creating a ClusterStore	86
Saving a ClusterStore	86
Adding buildpackages to a ClusterStore	86
Adding buildpackages to a ClusterStore from Tanzu Network	87
Offline Adding Buildpackages to a ClusterStore from Tanzu Network	87
Removing buildpackages from a ClusterStore	88
Get ClusterStore status	89
Migrating buildpacks	90
Corresponding kpack resource	90
Descriptors	91
About descriptors	91
Lite descriptor	91
Full descriptor	91
Descriptor comparison	91

Managing Builders	93
Creating a Builder	94
Patching a builder	95
Saving Builders	96
Deleting Builders	97
Retrieving Builder Details	97
Listing Builders	98
Corresponding kpack Resources	98
Pinning Buildpack versions	98
Update Lifecycle	99
Managing Custom Stacks	100
Creating a CustomStack	100
Source configuration	101
Destination Configuration	101
Example CustomStack from registry images	101
Example CustomStack from ClusterStack	103
Debugging CustomStacks	104
RBAC in Tanzu Build Service	105
RBAC support in Tanzu Build Service	105
Build Service user role	105
Build Service admin role	105
Using Tanzu Build Service in CI	107
Using Tanzu Build Service in CI/CD	107
Frequently Asked Questions	113
How do Cloud Native Buildpacks (CNBs), kpack, and Tanzu Build Service overlap and differ?	113
Why do I see two images in the image registry after a successful build?	113
How does TBS work in air gapped environments?	113
Is there documentation on supported Tanzu Buildpacks?	114
Why do I get an X509 error from Build Service when trying to create an image in my registry?	114
How do I configure a secret to publish images to Dockerhub?	114
How can I configure an image resource to pull from a private GitHub repository?	114
Why do some builds fail with "Error: could not read run image: *"?	114
Why don't my image builds appear in my Harbor v1.X.X registry?	115
How do I fix "unsupported status code 500" when creating a builder on my Harbor v2.X.X registry?	115

How do I configure credentials for using gcr as my installation registry?	115
Can I configure a proxy for my Tanzu Build Service?	115
How do I build my app locally using kpack builders?	116
What can I do with the kp --dry-run and --output flags?	116
Does TBS support Azure Devops for git repositories	117
Why do I get a "repository does not exist" error when I use ECR Registry?	117
How do I troubleshoot a failed build?	117
How do I troubleshoot an UNAUTHENTICATED error?	118
Why does TBS leave behind pods after builds on my Cluster?	119
How do I check what version of TBS I am using?	119
How does TBS use windows-based images?	120
What is the relationship between a kpack image resource and an OCI image?	120
Pinning the Tanzu Net Updater	120
Additional resources for Tanzu Build Service	122
Concourse Kpack resource	122
Helpful Articles	122
Helpful Videos	122
Helpful Repositories	122

Tanzu Build Service 1.6 Documentation

This topic provides an overview of Tanzu Build Service.

Overview

Tanzu Build Service uses the open-source [Cloud Native Buildpacks](#) project to turn application source code into [container images](#). Build Service executes reproducible builds that align with modern container standards and keeps image resources up to date. It does so by leveraging Kubernetes infrastructure with [kpack](#), a Cloud Native Buildpacks Platform, to orchestrate the image lifecycle. The kpack CLI tool, `kp` can aid in managing kpack resources.

Build Service helps you develop and automate containerized software workflows securely and at scale.

Build Service Concepts

Build Service reduces operational overhead and improves security by automating the building of application images. It relies on [Image](#), [Builder](#), [ClusterStore](#), and [ClusterStack](#) to achieve these results.

Image Resource

An [Image](#) resource defines the source of the application, build time environment and registry destination. This source code could reside in git, a blobstore, or as code on a workstation.

For more information see the [Managing Images and Builds](#) page.

Builder

A [Builder](#) references the Stack and Buildpacks used in the process of building source code. They provide the Buildpacks that run against the application and the operating system (OS) images upon which the application is built and run.

For more information, see the [Managing Builders](#) page.

ClusterStore

A [ClusterStore](#) serves as a repository for Cloud Native Buildpacks available for use in Builders. You can populate a store with Buildpacks you [create](#) and [package](#).

For more information, see the [Managing ClusterStores](#) page.

ClusterStack

A `ClusterStack` defines a pair of build and run OS images. Critical security vulnerabilities are addressed by building apps on the most up-to-date stack. The stacks used by Build Service to build applications are referenced in the Builders.

For more information, see the [Managing ClusterStacks](#) page.

Build Service components

Tanzu Build Service ships with the following components:

- [kpack](#)
- [kpack CLI \(kp\)](#)
- [CNB lifecycle](#)

Build Service Dependencies

Buildpacks

Tanzu Build Service utilize [Tanzu Buildpacks](#).

Stacks

Stack Documentation is available on the [Tanzu Buildpacks documentation](#).

The following Stacks and their updates can be found on the [Tanzu Build Service Dependencies](#) page.

Name	ID
tiny	io.paketo.stacks.tiny
base	io.buildpacks.stacks.bionic
full	io.buildpacks.stacks.bionic

Descriptors

Tanzu Build Service descriptors are curated sets of dependencies, including stacks and buildpacks, that are continuously released on [VMware Tanzu Network](#) to resolve all workload Critical and High CVEs. Descriptors are imported into Tanzu Build Service to update the entire cluster.

There are two types of descriptor, `lite` and `full`. The different descriptors can apply to different use cases and workload types. You can configure which descriptor is imported when installing Tanzu Build Service.

For more information, see [Descriptors](#).

Updating Build Service Dependencies

Build Service allows the user to update Buildpacks and Stacks via the `kp` CLI. You can learn more about updating Build Service Dependencies [here](#).

Troubleshooting

For troubleshooting failed builds, check the [FAQ section](#) of our docs.

If you are unable to resolve your problem, please contact [Tanzu VMware Support](#).

TBS Release Notes

v1.6.1

Release Date: August 9, 2022

New Features

- Update stacks

Breaking Changes

None

Product Snapshot

Tanzu Build Service 1.6.1 ships with the following components:

- [kpack 0.6.1](#)
- [kpack cli v0.6.0](#)
- [CNB lifecycle v0.14.1](#)

Tanzu Build Service supports and utilizes [Tanzu Buildpacks](#).

Security Scanning

No HIGH or CRITICAL CVEs at time of release

Product Dependencies

Build Service can be installed on any Kubernetes cluster (v1.19 or later).

Upgrade Path

v1.5.* can be upgraded to v1.6.*. Please follow the [install](#) process to upgrade.

v1.6.0

Release Date: July 11, 2022

New Features

- Support for using AWS IAM for authentication with ECR on EKS clusters
 - ◊ This mitigates issues with needing to recreate credentials when using ECR

Breaking Changes

None

Product Snapshot

Tanzu Build Service 1.6.0 ships with the following components:

- [kpack 0.6.0](#)
- [kpack cli v0.6.0](#)
- [CNB lifecycle v0.14.1](#)

Tanzu Build Service supports and utilizes [Tanzu Buildpacks](#).

Security Scanning

No HIGH or CRITICAL CVEs at time of release

Product Dependencies

Build Service can be installed on any Kubernetes cluster (v1.19 or later).

Upgrade Path

v1.5.* can be upgraded to v1.6.*. Please follow the [install](#) process to upgrade.

Installing Tanzu Build Service

This topic describes how to install and configure Tanzu Build Service.

VMware recommends installing Tanzu Build Service with the Tanzu CLI and kapp controller. If you want to view the previous method of installation, see [Installing without kapp controller](#).

Reasons to use the previous installation method include:

- You do not want to install kapp controller on your cluster.
- You want to make custom changes to the Tanzu Build Service installation through `ytt` templating or overlays.

Prerequisites

Before you install Build Service, you must:

- Be on Kubernetes cluster v1.19 or later.
- Ensure that all worker nodes have at least 50 GB of ephemeral storage allocated to them.
 - ◊ To do this on TKGs, mount a 50GB volume at `/var/lib/containerd` to the worker nodes in the `TanzuKubernetesCluster` resource that corresponds to your TKGs cluster. For information about configuring storage on worker nodes, see [Cluster with Separate Disks and Storage Parameters](#).
- Have access to a container registry to install Tanzu Build Service and store application images.
 - ◊ Although the documentation references specific registries for the purpose of providing examples, any registry that adheres to the Docker Registry HTTP API V2 is supported.
 - ◊ If installing using the `lite` descriptor, VMware recommends 1 GB of registry storage.
 - ◊ If installing using the `full` descriptor, which is intended for production use and offline environments, VMware recommends 10 GB of available storage.
 - ◊ This registry space suggestion does not include the space used for application images built by Tanzu Build Service.
- Ensure your Kubernetes cluster is configured with a default `StorageClass`. Tanzu Build Service defaults to using 2 GB of cache if a default `StorageClass` is defined. Build Service utilizes `PersistentVolumeClaims` to cache build artifacts, which reduces the time of subsequent builds.

For more information, see [Persistent Volumes](#) in the Kubernetes documentation. For information about defining a default `StorageClass`, see [Changing the default StorageClass](#).

- Download Carvel [imgpkg](#). Version 0.12.0 or higher is required.
- Install the [Tanzu CLI](#).
 - ◊ Only the [package](#) and [secret](#) plug-ins are required.
- Install Carvel [kapp controller](#) and [secret gen controller](#) on your cluster.
- Navigate to the following pages in Tanzu Network and accept all EULAs highlighted in yellow.
 - ◊ [Tanzu Build Service](#)
 - ◊ [Tanzu Build Service Dependencies](#)
 - ◊ [Buildpacks for VMware Tanzu](#)
 - ◊ [Stacks for VMware Tanzu](#)
- Find the latest Tanzu Build Service version by checking the [Tanzu Build Service](#) page on Tanzu Network. Just knowing the version is sufficient.
- Download the [kp](#) CLI for your operating system from the [Tanzu Build Service](#) page on Tanzu Network. See the [kp](#) CLI [help text](#).
 - ◊ These docs assume `kp cli v0.6.*` from Tanzu Build Service release `v1.6.*`. If a feature is not working, you may need to upgrade your cli.
- Download either the [docker](#) CLI or [crane](#) CLI for authentication with registries.

Note: Clusters running with Containerd 1.4.1, 1.5.6, and 1.5.7 are not compatible with Tanzu Build Service. Notably, TKG 1.2.1 and TKGi 1.13.0 & 1.13.1 use these versions of Containerd, so a different version must be used.

Note: TKGs clusters running Kubernetes 1.20.0-1.20.6 are not compatible with Tanzu Build Service.

Install Tanzu Build Service

1. Set up environment variables for use during the installation.

```
export INSTALL_REGISTRY_HOSTNAME=<IMAGE-REGISTRY>
export INSTALL_REPOSITORY=<IMAGE-REPOSITORY>
export INSTALL_REGISTRY_USERNAME=<REGISTRY-USERNAME>
export INSTALL_REGISTRY_PASSWORD=<REGISTRY-PASSWORD>
export TANZUNET_REGISTRY_USERNAME=<TANZUNET_REGISTRY_USERNAME>
export TANZUNET_REGISTRY_PASSWORD=<TANZUNET_REGISTRY_PASSWORD>
export TBS_VERSION=<LATEST-TBS-VERSION>
```

Where:

- ◊ `IMAGE-REGISTRY` is the host name of the registry.
- ◊ `IMAGE-REPOSITORY` is the repository in your registry that you want to relocate images to.
 - Dockerhub has the form `my-dockerhub-username/build-service` or `index.docker.io/my-dockerhub-username/build-service`.
 - gcr.io has the form `gcr.io/my-project/build-service`.
 - Harbor has the form `my-harbor.io/my-project/build-service`.

- ✦ `REGISTRY-USERNAME` is the user name of the registry. You must be able to write to `IMAGE-REPOSITORY` with this credential. `gcr.io` expects `_json_key`.
- ✦ `REGISTRY-PASSWORD` is the password of the registry. You must be able to write to `IMAGE-REPOSITORY` with this credential.
- ✦ `TANZUNET_REGISTRY_USERNAME` is the user name you use to access [TanzuNet](#).
- ✦ `TANZUNET_REGISTRY_PASSWORD` is the password you use to access [TanzuNet](#).
 - For Google Cloud Registry, use the contents of the service account JSON key.
- ✦ `LATEST-TBS-VERSION` is from the [Tanzu Network Release](#) page.

Relocate images to a registry

This procedure relocates images from the Tanzu Network registry to an internal image registry through a local machine.

The local machine must have write access to the install registry.

1. Log in to the Tanzu Network registry with your Tanzu Network credentials:

- ✦ If using the `docker` CLI, run:

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

- ✦ If using the `crane` CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

2. Log in to the image registry where you want to store the images:

- ✦ If using the `docker` CLI, run:

```
docker login ${INSTALL_REGISTRY_HOSTNAME}
```

- ✦ If using the `crane` CLI, run:

```
crane auth login ${INSTALL_REGISTRY_HOSTNAME}
```

3. Copy the Tanzu Build Service package repository to your registry with the [Carvel](#) tool `imgpkg` by running:

```
imgpkg copy -b registry.tanzu.vmware.com/build-service/package-repo:$TBS_VERSION --to-repo=${INSTALL_REPOSITORY}
```

For example:

- ✦ Dockerhub `imgpkg copy -b registry.tanzu.vmware.com/build-service/package-repo:$TBS_VERSION --to-repo=my-dockerhub-account/build-service`
- ✦ GCR `imgpkg copy -b registry.tanzu.vmware.com/build-service/package-repo:$TBS_VERSION --to-repo=gcr.io/my-project/build-service`
- ✦ Artifactory `imgpkg copy -b registry.tanzu.vmware.com/build-service/package-`

```
repo:$TBS_VERSION --to-repo=artifactory.com/my-project/build-service
```

- ◊ Harbor `imgpkg copy -b registry.tanzu.vmware.com/build-service/package-repo:$TBS_VERSION --to-repo=harbor.io/my-project/build-service`

Installation

1. Create a namespace called `tbs-install` for deploying the package by running:

```
kubectl create ns tbs-install
```

This namespace keeps the installation objects grouped together logically.

2. Create a secret to pull in the package repository:

```
tanzu secret registry add tbs-install-registry \
  --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWORD} \
  --server ${INSTALL_REGISTRY_HOSTNAME} \
  --export-to-all-namespaces --yes --namespace tbs-install
```

3. Add the Tanzu Build Service package repository to the cluster by running:

```
tanzu package repository add tbs-repository \
  --url "${INSTALL_REPOSITORY}:${TBS_VERSION}" \
  --namespace tbs-install
```

4. You should be able to get the status of the package repository and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tbs-repository --namespace tbs-install
```

5. Create a `tbs-values.yml` file by using the following sample as a guide. This file should be kept for future use.

```
---
kp_default_repository: <INSTALL_REPOSITORY>
kp_default_repository_username: <INSTALL_REGISTRY_USERNAME>
kp_default_repository_password: <INSTALL_REGISTRY_PASSWORD>
pull_from_kp_default_repo: true
tanzunet_username: <TANZUNET_REGISTRY_USERNAME>
tanzunet_password: <TANZUNET_REGISTRY_PASSWORD>
descriptor_name: <DESCRIPTOR_NAME>
enable_automatic_dependency_updates: true
ca_cert_data: <CA_CERT_CONTENTS> (optional)
```

Where:

- ◊ `INSTALL_REPOSITORY` is a writable repository in your registry. Tanzu Build Service Dependencies are written to this location. Same value as used during relocation.
- ◊ `INSTALL_REGISTRY_USERNAME` is the registry username. Same value as used during relocation.
- ◊ `INSTALL_REGISTRY_PASSWORD` is the registry password. Same value as used during

relocation.

- ◊ `TANZUNET_REGISTRY_USERNAME` is used to pull dependencies from tanzu network. Same value used during relocation
- ◊ `TANZUNET_REGISTRY_PASSWORD` is used to pull dependencies from tanzu network. Same value used during relocation
- ◊ `DESCRIPTOR-NAME` is the name of the descriptor to import automatically. For more information about which descriptor to choose for your workload and use case, see [Descriptors](#). Available options:
 - `full` contains all dependencies.
 - `lite` smaller footprint used for speeding up installs. Requires Internet access on the cluster.
- ◊ `CA_CERT_CONTENTS` *must be provided when using registry that is signed by a Custom Cert*. This should be the value of the PEM-encoded CA certificate.

6. Install the package by running:

```
tanzu package install tbs -p buildservice.tanzu.vmware.com -v $TBS_VERSION -n tbs-install -f tbs-values.yml --poll-timeout 30m
```

Note: Installing with Tanzu Network credentials automatically relocates buildpack dependencies to your registry. This install process can take some time and the `--poll-timeout` flag increases the timeout duration. Using the `lite` descriptor speeds this up significantly. If the command times out, periodically run the installation verification step provided in the following optional step. Image relocation continues in the background.

Using Secrets for credentials

You may not want to install TBS with passwords saved in plaintext in the `tbs-values.yml`.

You can store these credentials in [Secrets](#) and reference them in the `tbs-values.yml` using the following process:

1. Create the secret for the `kp-default-repository` credentials using the `tanzu cli`:

```
tanzu secret registry add kp-default-repository-creds \
  --username "${INSTALL_REGISTRY_USERNAME}" \
  --password "${INSTALL_REGISTRY_PASSWORD}" \
  --server "${INSTALL_REGISTRY_HOSTNAME}" \
  --namespace tbs-install
```

2. When leveraging automatic updates, create the secret for the Tanzu Network credentials using the `tanzu cli`:

```
tanzu secret registry add tanzunet-creds \
  --server "registry.tanzu.vmware.com" \
  --username "${TANZUNET_REGISTRY_USERNAME}" \
  --password "${TANZUNET_REGISTRY_PASSWORD}" \
  --namespace tbs-install
```

3. Create a `tbs-values.yml` with secret references instead of plaintext credentials:

```

---
kp_default_repository: <INSTALL_REPOSITORY>
kp_default_repository_secret:
  name: kp-default-repository-creds
  namespace: tbs-install
pull_from_kp_default_repo: true
tanzunet_secret:
  name: tanzunet-creds
  namespace: tbs-install
descriptor_name: <DESCRIPTOR_NAME>
enable_automatic_dependency_updates: true
ca_cert_data: <CA_CERT_CONTENTS> (optional)

```

Additional configuration

- You can add additional fields to the `tbs-values.yml`:
 - `admin_users` is a comma-separated list of users granted admin privileges on Build Service.
 - `admin_groups` is a comma-separated list of groups granted admin privileges on Build Service.
 - `http_proxy` is the HTTP proxy to use for network traffic.
 - `https_proxy` is the HTTPS proxy to use for network traffic.
 - `no_proxy` is a comma-separated list of host names, IP addresses, or IP address ranges in CIDR format. These must not use a proxy.
- You can see the full values schema by running:

```

tanzu package available get buildservice.tanzu.vmware.com/$TBS_VERSION --values
-schema --namespace tbs-install

```

Using AWS IAM for Authentication with ECR

You can install Tanzu Build Service using [AWS IAM roles](#) for authentication with Amazon Elastic Container Registry (ECR) on Amazon Elastic Kubernetes Service (EKS) clusters.

There are two additional steps required:

- [Install Tanzu Build Service with AWS IAM for Authentication with ECR](#)
- [Configure the build namespace for IAM Auth with ECR](#)

Install Tanzu Build Service with AWS IAM for Authentication with ECR

To install with AWS IAM for Authentication with ECR:

- Before installing Tanzu Build Service, create a repository in ECR to use for the `kp_default_repository`.
- Configure an AWS IAM role that has read and write access to the `kp_default_repository` and a trust policy for the EKS cluster. The minimum required permissions can be found [here](#).

- Follow the Tanzu Build Service installation steps until the `tbs-values.yml` file.
- In the `tbs-values.yml` file, replace the `kp_default_repository_username` and `kp_default_repository_secret` with the following line:

```
...
kp_default_repository_aws_iam_role_arn: "INSTALL-IAM-ROLE-ARN"
...
```

Where:

- ◆ `INSTALL-IAM-ROLE-ARN` is the AWS IAM role Amazon Resource Name (ARN) for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-install-role`.

Configure the build namespace for IAM Auth with ECR

Similar to configuring a secret in the build namespace for registry authentication, the build namespace requires configuration for Tanzu Build Service to use AWS IAM authentication for ECR in builds.

To configure the build namespace for IAM Auth with ECR:

- Configure an AWS IAM role that has read and write access to the registry location to store build images.
- For the service account used for `Image` resources, add an annotation including the role ARN configured earlier.

```
kubectl annotate serviceaccount -n BUILD-NAMESPACE SERVICE-ACCOUNT-NAME \
eks.amazonaws.com/role-arn=${BUILD-IAM-ROLE-ARN}
```

Where:

- ◆ `BUILD-NAMESPACE` is the name of your build namespace.
 - ◆ `SERVICE-ACCOUNT-NAME` is the service account used for `Image` resources. This is `default` if unset.
 - ◆ `BUILD-IAM-ROLE-ARN` is the AWS IAM role ARN for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-build-role`.
- To create an `Image` resource with this service account, run:

```
kp image save my-ecr-image --tag 12345.ecr.com/my-repo --service-account=SERVICE-ACCOUNT-NAME -n BUILD-NAMESPACE
```

Upgrading Tanzu Build Service

To upgrade Tanzu Build Service to a newer version, take the following steps.

- Relocate the new package repository:

```
imgpkg copy -b registry.tanzu.vmware.com/build-service/package-repo:$NEW_TBS_VERSION --to-repo=${INSTALL_REPOSITORY}
```

- Add the Tanzu Build Service package repository to the cluster by running:

```
tanzu package repository add tbs-repository \
  --url "${INSTALL_REPOSITORY}:${NEW_TBS_VERSION}" \
  --namespace tbs-install
```

3. Install the package with the same `tbs-values.yml` file used during initial installation by running:

```
tanzu package installed update tbs -p buildservice.tanzu.vmware.com -v $NEW_TBS_VERSION -n tbs-install -f tbs-values.yml --poll-timeout 30m
```

Installing Tanzu Build Service (air-gapped)

Tanzu Build Service can be installed to a Kubernetes cluster and registry that are air-gapped from external traffic.

An air-gapped environment often uses an internal registry with a self-signed CA certificate. You need access to this CA certificate file to install Tanzu Build Service.

Note: If you are using a CA certificate that is trusted (for example, Lets Encrypt), you do not need the CA certificate file.

1. Set up environment variables for use during the installation.

```
export INSTALL_REGISTRY_HOSTNAME=<IMAGE-REGISTRY>
export INSTALL_REPOSITORY=<IMAGE-REPOSITORY>
export INSTALL_REGISTRY_USERNAME=<REGISTRY-USERNAME>
export INSTALL_REGISTRY_PASSWORD=<REGISTRY-PASSWORD>
export TBS_VERSION=<LATEST-TBS-VERSION>
```

Where:

- ◊ `IMAGE-REGISTRY` is the host name of the private registry.
- ◊ `IMAGE-REPOSITORY` is the repository in your registry that you want to relocate images to.
 - Harbor has the form `my-harbor.io/my-project/build-service`
- ◊ `REGISTRY-USERNAME` is the user name of the private registry. You must be able to write to `IMAGE-REPOSITORY` with this credential.
- ◊ `REGISTRY-PASSWORD` is the password of the private registry. You must be able to write to `IMAGE-REPOSITORY` with this credential.
- ◊ `LATEST-TBS-VERSION` is from the [Tanzu Network Release](#) page.

Note: The `IMAGE-REPOSITORY` must be the `IMAGE-REGISTRY` appended with the destination repository for the images. For example, `IMAGE-REGISTRY/some-repo/build-service`.

Relocate images to a registry (air-gapped)

This procedure relocates images from the Tanzu Network registry to an internal image registry through local machine(s).

1. Log in to the Tanzu Network registry with your Tanzu Network credentials:

- ◆ If using the `docker` CLI, run:

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

- ◆ If using the `crane` CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

2. Copy the Tanzu Build Service package repository to your local machine as a tar with the `Carvel` tool `imgpkg` by running:

```
imgpkg copy -b registry.tanzu.vmware.com/build-service/package-repo:$TBS_VERSION --to-tar=/tmp/tanzu-build-service.tar
```

3. Move the output file `tanzu-build-service.tar` to a machine that has access to the air-gapped environment. The machine must have write access to the internal registry.
4. Log in to the image registry where you want to store the images by running:

- ◆ If using the `docker` CLI, run:

```
docker login ${INSTALL_REGISTRY_HOSTNAME}
```

- ◆ If using the `crane` CLI, run:

```
crane auth login ${INSTALL_REGISTRY_HOSTNAME}
```

5. Copy the images from your local machine to the internal registry:

```
imgpkg copy --tar /tmp/tanzu-build-service.tar \
  --to-repo=${INSTALL_REPOSITORY} \
  --registry-ca-cert-path <PATH-TO-CA>
```

Where:

- ◆ `PATH-TO-CA` is the path to the registry CA certificate file.

For example:

- ◆ `Artifactory` `imgpkg copy --tar /tmp/tanzu-build-service.tar --to-repo=artifactory.com/my-project/build-service --registry-ca-cert-path ca.crt`
- ◆ `Harbor` `imgpkg copy --tar /tmp/tanzu-build-service.tar --to-repo=harbor.io/my-project/build-service --registry-ca-cert-path ca.crt`

Installation (air-gapped)

1. Create a namespace called `tbs-install` for deploying the package by running:

```
kubectl create ns tbs-install
```

This namespace keeps the installation objects grouped together logically.

2. Add the Tanzu Build Service package repository to the cluster by running:

```
tanzu package repository add tbs-repository \
  --url "${IMAGE-REPOSITORY}:${TBS_VERSION}" \
  --namespace tbs-install
```

3. You should be able to get the status of the package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tbs-repository --namespace tbs-install
```

4. Create a `tbs-values.yml` file by using the following sample as a guide. This file should be kept for future use.

```
---
kp_default_repository: $INSTALL_REPOSITORY
kp_default_repository_username: $INSTALL_REGISTRY_USERNAME
kp_default_repository_password: $INSTALL_REGISTRY_PASSWORD
pull_from_kp_default_repo: true
exclude_dependencies: true
ca_cert_data: <CA_CERT_CONTENTS>
```

Where:

- ◆ `INSTALL_REPOSITORY` is a writable repository in your internal registry. Tanzu Build Service Dependencies are written to this location. Same value as used during relocation.
- ◆ `INSTALL_REGISTRY_USERNAME` is the internal registry username. Same value as used during relocation.
- ◆ `INSTALL_REGISTRY_PASSWORD` is the internal registry password. Same value as used during relocation.
- ◆ `CA_CERT_CONTENTS` *must be provided when using registry that is signed by a Custom Cert*. This should be the value of the PEM-encoded CA certificate.

Additional configuration

1. Additional fields can be added to `tbs-values.yml`.
 - ◆ `admin_users` is a comma-separated list of users who are granted admin privileges on Build Service.
 - ◆ `admin_groups` is a comma-separated list of groups that are granted admin privileges on Build Service.
 - ◆ `http_proxy` is the HTTP proxy to use for network traffic.
 - ◆ `https_proxy` is the HTTPS proxy to use for network traffic.
 - ◆ `no_proxy` is a comma-separated list of hostnames, IP addresses, or IP ranges in CIDR format that should not use a proxy.
2. You can see the full values schema by running:


```
tanzu package available get buildservice.tanzu.vmware.com/$TBS_VERSION --values
-schema --namespace tbs-install
```

3. Install the package by running:

```
tanzu package install tbs -p buildservice.tanzu.vmware.com -v $TBS_VERSION -n t
bs-install -f tbs-values.yml
```

Updating Build Service Dependencies (air-gapped)

When running in an air-gapped environment, Tanzu Build Service cannot pull dependencies automatically from external Internet. Therefore, dependencies must be imported manually as a part of installation for Tanzu Build Service to work.

Tanzu Build Service dependencies must be kept up to date manually or in a CI/CD automated way in order to keep application images patched. To learn more about keeping dependencies up to date in an offline environment, see [Updating Build Service Dependencies](#).

Upgrading Tanzu Build Service (air-gapped)

1. To upgrade Tanzu Build Service to a newer version in an air-gapped environment, the same relocation steps listed [here](#) must be followed with a new `TBS_VERSION`.
2. Then the same installation steps can be followed by using the same `tbs-values.yml` file used for initial installation.

Re-importing dependencies is not required for upgrading Tanzu Build Service.

Uninstalling Tanzu Build Service

1. To uninstall Tanzu Build Service, uninstall the package using the `tanzu cli`:

```
tanzu package installed delete tbs -n tbs-install -y
```

Note: All Tanzu Build Service resources are deleted. Registry images created by Tanzu Build Service are not deleted.

2. To delete the Tanzu Build Service package repository:

```
tanzu package repository delete tbs-repository --namespace tbs-install
```

Ensuring access to cluster builders

To use cluster builders, such as the ones installed with Tanzu Build Service, VMware suggests installing Tanzu Build Service to a repository accessible by the nodes in the Kubernetes cluster without credentials.

If you do not want to do this, see [When to use Synchronized Secrets](#).

Ensuring the run image is readable

Build Service relies on the run image being publicly readable or readable with the registry

credentials configured in a project/namespace for the builds to be executed successfully.

The location of the run image can be identified by running:

```
kp clusterstack status <stack-name>
```

If the cluster stack run image is not public, you might need to create a registry secret in any namespace where images or builds are used. For more details on secrets in Tanzu Build Service, see [Managing Secrets](#)

You can do this with the `kp` CLI:

```
kp secret create my-registry-creds --registry example-registry.io --registry-user my-registry-user --namespace build-namespace
```

Next steps

Visit the [Managing Images and Builds](#) page to learn how to create and manage a new image.

Minimum AWS IAM permissions for ECR IAM auth

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:PutImageTagMutability",
        "ecr:StartImageScan",
        "ecr:DescribeImageReplicationStatus",
        "ecr:ListTagsForResource",
        "ecr:UploadLayerPart",
        "ecr:BatchDeleteImage",
        "ecr:ListImages",
        "ecr:BatchGetRepositoryScanningConfiguration",
        "ecr>DeleteRepository",
        "ecr:CompleteLayerUpload",
        "ecr:TagResource",
        "ecr:DescribeRepositories",
        "ecr:BatchCheckLayerAvailability",
        "ecr:ReplicateImage",
        "ecr:GetLifecyclePolicy",
        "ecr:PutLifecyclePolicy",
        "ecr:DescribeImageScanFindings",
        "ecr:GetLifecyclePolicyPreview",
        "ecr:PutImageScanningConfiguration",
        "ecr:GetDownloadUrlForLayer",
        "ecr>DeleteLifecyclePolicy",
        "ecr:PutImage",
        "ecr:UntagResource",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:StartLifecyclePolicyPreview",
        "ecr:InitiateLayerUpload",
        "ecr:GetRepositoryPolicy"
      ]
    }
  ]
}
```

```

    ],
    "Resource": "<your repository>"
  },
  {
    "Sid": "VisualEditor1",
    "Effect": "Allow",
    "Action": [
      "ecr:GetRegistryPolicy",
      "ecr:BatchImportUpstreamImage",
      "ecr:CreateRepository",
      "ecr:DescribeRegistry",
      "ecr:DescribePullThroughCacheRules",
      "ecr:GetAuthorizationToken",
      "ecr:PutRegistryScanningConfiguration",
      "ecr:CreatePullThroughCacheRule",
      "ecr>DeletePullThroughCacheRule",
      "ecr:GetRegistryScanningConfiguration",
      "ecr:PutReplicationConfiguration"
    ],
    "Resource": "<your repository>"
  }
]
}

```

Installing Windows Components (Beta)

Important: This feature is in beta because of its limitations. Beta features might undergo changes before the end of the beta stage. This beta feature receives full VMware support.

Note: Tanzu Build Service on Windows does not currently support self-signed registry certificates. Please use a public registry or a non-self-signed cert.

Tanzu Build Service supports building [.NET Framework](#) application images. Building .NET Framework images require a Kubernetes Cluster with Windows nodes provisioned.

After the Windows nodes are provisioned, the Tanzu Build Service Windows Dependencies (Stacks, Buildpacks, Builders, and so on) can be used to build .NET Framework applications and keep them patched. These must be imported with the `kp` cli and the Dependency Descriptor (`windows-descriptor-<version>.yaml`) file from the [Tanzu Build Service Dependencies for Microsoft Windows](#) page:

```
kp import -f /tmp/windows-descriptor-<version>.yaml
```

The following features are not yet supported on Windows nodes of Tanzu Build Service:

- Caching of build artifacts (which reduces the time of subsequent builds)
- Preloading of ClusterBuilder images
- Self-signed registry certificate

Getting Started with Tanzu Build Service

This topic describes how to get started with a typical installation of Tanzu Build Service and create an Image.

This page is meant to serve as a quick-start guide and may not include some configurations required for your specific environment. For more details on installation, see [Installing Tanzu Build Service](#).

Prerequisites

Before you install Build Service, you must:

- Have access to the Kubernetes cluster satisfying the [minimum required permissions](#).
- Users must navigate to the following dependencies pages in Tanzu Network and accept all EULAs highlighted in yellow.
 1. [Tanzu Build Service Dependencies](#)
 2. [Buildpacks for VMware Tanzu](#)
 3. [Stacks for VMware Tanzu](#)
- Ensure your Kubernetes cluster is configured with default `StorageClass`. Tanzu Build Service defaults to using 2G of cache if a default `StorageClass` is defined. Build Service utilizes `PersistentVolumeClaims` to cache build artifacts, which reduces the time of subsequent builds.

For more information, see [Persistent Volumes](#) in the Kubernetes documentation. And for information on defining a default `StorageClass`, see [Changing the default StorageClass](#)

- Download three [Carvel](#) CLIs for your operating system. These tools facilitate the installation of Tanzu Build Service on your cluster. They can be found on their respective Tanzu Network pages:
 - ◊ [kapp](#) is a deployment tool that allows users to manage Kubernetes resources in bulk.
 - ◊ [ytt](#) is a templating tool that understands YAML structure. Version 0.35.0 or higher is required.
 - ◊ [kblid](#) is needed to map relocated images to k8s config.
 - ◊ [imgpkg](#) is tool that relocates container images and pulls the release configuration files. Version 0.12.0 or higher is required.
- Find the latest Tanzu Build Service version by checking the [Tanzu Build Service](#) page on Tanzu Network. Just knowing the version is sufficient.
- Download the [kp](#) CLI for your operating system from the [Tanzu Build Service](#) page on Tanzu Network. The [kp](#) CLI help text is published [here](#).
 - ◊ These docs assume `kp cli v0.6.*` from TBS release `v1.6.*`. If a feature is not working, you may need to upgrade your cli.
- Download either the [docker](#) CLI or [crane](#) CLI for authentication with registries.

Assumptions

For this example setup, we make the following assumptions:

- You are installing TBS 1.6.0 (This is the latest version at the time of writing. Go to the [Tanzu Build Service](#) page to find the most up-to-date version).

- You are using a registry named `my.registry.io` with credentials
 - ◊ Username: `my-user`
 - ◊ Password: `my-password`
- Your registry uses a self-signed CA certificate and you have access to the cert in a file `/tmp/ca.crt`
 - ◊ The nodes on your cluster must also be configured to trust this CA certificate so they can pull in images. Configuration for this depends on the cluster provider
- You are using an "online" environment that has access to the internet

Installation

Relocate Images to a Registry

This procedure relocates images from the Tanzu Network registry to your registry.

1. Log in to your image registry:

- ◊ If using the `docker` CLI, run:

```
docker login my.registry.io --tlscacert /tmp/ca.crt
```

- ◊ If using the `crane` CLI, run:

```
crane auth login my.registry.io --tlscacert /tmp/ca.crt
```

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

- ◊ If using the `docker` CLI, run:

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

- ◊ If using the `crane` CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

3. Relocate the images with the `Carvel` tool `imgpkg` by running:

```
imgpkg copy -b "registry.tanzu.vmware.com/build-service/bundle:1.6.0" --to-repo my.registry.io/some-repo/tbs --registry-ca-cert-path /tmp/ca.crt
```

4. Pull the Tanzu Build Service bundle locally using `imgpkg`:

```
imgpkg pull -b "my.registry.io/tbs:1.6.0" -o /tmp/bundle
```

Install Tanzu Build Service

Use the `Carvel` tools `kapp`, `ytt`, and `kblid` to install Build Service and define the required Build Service parameters by running:

```
ytt -f /tmp/bundle/config/ \
```

```
-f /tmp/ca.crt \
-v kp_default_repository='my.registry.io/tbs' \
-v kp_default_repository_username='my-user' \
-v kp_default_repository_password='my-password' \
--data-value-yaml pull_from_kp_default_repo=true \
-v tanzunet_username='tanzunet-username' \
-v tanzunet_password='tanzunet-password' \
-v descriptor_name='lite' \
--data-value-yaml enable_automatic_dependency_updates=true \
| kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
| kapp deploy -a tanzu-build-service -f- -y
```

Verify Installation

To verify your Build Service installation:

List the cluster builders available in your installation:

```
kp clusterbuilder list
```

You should see an output that looks as follows:

NAME	READY	STACK	IMAGE
base	true	io.buildpacks.stacks.bionic	<image@sha256:digest>
default	true	io.buildpacks.stacks.bionic	<image@sha256:digest>
full	true	io.buildpacks.stacks.bionic	<image@sha256:digest>
tiny	true	io.paketo.stacks.tiny	<image@sha256:digest>

Create an Image

You can now create a Tanzu Build Service Image to start building your app and keep it patched with the latest Stack and Buildpack Dependencies.

We assume you are using the `default` namespace, use `-n` when using `kp` to set a specific namespace.

1. Create a Kubernetes Secret that allows your Builds to push to the desired registry with the `kp` cli:

```
kp secret create my-registry-creds --registry my.registry.io --registry-user my-user
```

You are prompted for your password (`my-password`).

2. Create the Tanzu Build Service Image:

We use a [sample java-maven app](#):

```
kp image create my-image --tag my.registry.io/tbs/test-app --git https://github.com/buildpacks/samples --sub-path ./apps/java-maven --wait
```

Installing Tanzu Build Service without kapp controller

This topic describes the method of installing Tanzu Build Service without kapp controller. The

recommended method uses the [tanzu cli](#) and kapp controller and can be found [here](#).

Reasons to use the previous installation method include:

- You do not want to install kapp controller on your cluster
- You want to make custom changes to the Tanzu Build Service installation via [ytt](#) templating or overlays

Prerequisites

Before you install Build Service, you must:

- Be on Kubernetes cluster v1.19 or later
- Have access to the Kubernetes cluster satisfying the [minimum required permissions](#).
- Ensure that all worker nodes have at least 50 GB of ephemeral storage allocated to them.
 - ◊ To do this on TKGs, mount a 50GB volume at `/var/lib/containerd` to the worker nodes in the `TanzuKubernetesCluster` resource that corresponds to your TKGs cluster. [These instructions](#) show how to configure storage on worker nodes.
- Have access to a container registry to install Tanzu Build Service and store the application images created.
 - ◊ Although the documentation references specific registries for the purpose of providing examples, any registry that adheres to the Docker Registry HTTP API V2 is supported.
 - ◊ If installing using the `lite` descriptor, VMware recommends 1 GB of registry storage.
 - ◊ If installing using the `full` descriptor, which is intended for production use and offline environments, VMware recommends 10 GB of available storage.
 - ◊ This registry space suggestion does not include the space used for application images Tanzu Build Service builds.
- Ensure your Kubernetes cluster is configured with default `StorageClass`. Tanzu Build Service defaults to using 2G of cache if a default `StorageClass` is defined. Build Service utilizes `PersistentVolumeClaims` to cache build artifacts, which reduces the time of subsequent builds.

For more information, see [Persistent Volumes](#) in the Kubernetes documentation. And for information on defining a default `StorageClass`, see [Changing the default StorageClass](#)

- Download four [Carvel](#) CLIs for your operating system. These tools facilitate the installation of Tanzu Build Service on your cluster. They can be found on their respective Tanzu Network pages:
 - ◊ [kapp](#) is a deployment tool that allows users to manage Kubernetes resources in bulk.
 - ◊ [ytt](#) is a templating tool that understands YAML structure.
 - ◊ [kblid](#) is needed to map relocated images to k8s config.
 - ◊ [imgpkg](#) is tool that relocates container images and pulls the release configuration files. Note: `imgpkg 0.12.0` or higher is required for installation. If it is not available on

TanzuNet, it can be found [here](#)

- Navigate to the following pages in Tanzu Network and accept all EULAs highlighted in yellow.
 - ◊ [Tanzu Build Service](#)
 - ◊ [Tanzu Build Service Dependencies](#)
 - ◊ [Buildpacks for VMware Tanzu](#)
 - ◊ [Stacks for VMware Tanzu](#)
- Find the latest Tanzu Build Service version by checking the [Tanzu Build Service](#) page on Tanzu Network. Just knowing the version is sufficient.
- Download the `kp` CLI for your operating system from the [Tanzu Build Service](#) page on Tanzu Network. The `kp` CLI help text is published [here](#).
 - ◊ These docs assume `kp cli v0.6.*` from Tanzu Build Service release `v1.6.*`. If a feature is not working, you may need to upgrade your cli.
- Download either the `docker` CLI or `crane` CLI for authentication with registries.
- Download the Dependency Descriptor file (`descriptor-<version>.yaml`) from the latest release on the [Tanzu Build Service Dependencies] ([https://network.tanzu.vmware.com/products/Tanzu Build Service-dependencies/](https://network.tanzu.vmware.com/products/Tanzu%20Build%20Service-dependencies/)) page on Tanzu Network. This file contains paths to images that contain dependency resources Tanzu Build Service needs to execute image builds.

Note: Clusters running with Containerd 1.4.1, 1.5.6, and 1.5.7 are not compatible with Tanzu Build Service. Notably, TKG 1.2.1 and TKGi 1.13.0 & 1.13.1 use these versions of Containerd, a different version must be used.

Note: TKGs clusters running Kubernetes 1.20.0-1.20.6 are not compatible with Tanzu Build Service.

Installing

Create a kubernetes cluster where you would like to install build service and target the cluster as follows:

```
kubectl config use-context <CONTEXT-NAME>
```

Relocate images to a registry

This procedure relocates images from the Tanzu Network registry to an internal image registry.

1. Log in to the image registry where you want to store the images by running:

- ◊ If using the `docker` CLI, run:

```
docker login <IMAGE-REGISTRY>
```

Where `IMAGE-REGISTRY` is the name of the image registry where you want to store the images.

- ◊ If using the `crane` CLI, run:


```
crane auth login <IMAGE-REGISTRY>
```

Where `IMAGE-REGISTRY` is the name of the image registry where you want to store the images.

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

- ◆ If using the `docker` CLI, run:

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

- ◆ If using the `crane` CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

3. Relocate the images with the `Carvel` tool `imgpkg` by running:

```
imgpkg copy -b "registry.tanzu.vmware.com/build-service/bundle:<TBS-VERSION>" -  
-to-repo <IMAGE-REPOSITORY>
```

Where:

- ◆ `TBS-VERSION` is the version full version (1.6.x) of Tanzu Build Service that you want to install.
- ◆ `IMAGE-REPOSITORY` is the repository in your registry that you want to relocate images to.

Note: When relocating, the `IMAGE-REPOSITORY` must be the `IMAGE-REGISTRY` appended with the destination repository for the images. For example, `IMAGE-REGISTRY/some-repo/build-service`.

Exception: When relocating to Dockerhub, you must provide the Dockerhub username and a repository name that `imgpkg` uses for relocation. For example, `my-dockerhub-account/build-service`.

For example:

- ◆ Dockerhub `imgpkg copy -b "registry.tanzu.vmware.com/build-service/bundle:<TBS-VERSION>" --to-repo my-dockerhub-account/build-service`
- ◆ GCR `imgpkg copy -b "registry.tanzu.vmware.com/build-service/bundle:<TBS-VERSION>" --to-repo gcr.io/my-project/build-service`
- ◆ Artifactory `imgpkg copy -b "registry.tanzu.vmware.com/build-service/bundle:<TBS-VERSION>" --to-repo artifactory.com/my-project/build-service`
- ◆ Harbor `imgpkg copy -b "registry.tanzu.vmware.com/build-service/bundle:<TBS-VERSION>" --to-repo harbor.io/my-project/build-service`

Note: During relocation, `imgpkg` reports the following:

Skipped layer due to it being non-distributable. If you would like to include non-distributable layers, use the `--include-non-distributable` flag.

This is due to windows-based images shipped with Tanzu Build Service and can be ignored. For more details see the [FAQ](#).

Install Tanzu Build Service

There are two ways to install Tanzu Build Service:

1. Using a public registry (eg. GCR, Dockerhub) or an internal registry that uses a trusted certificate (eg. Lets Encrypt)
2. Using an internal registry that uses a self-signed CA certificate (eg. Harbor, Artifactory)

Install Tanzu Build Service Public Registry

1. Pull the Tanzu Build Service bundle image locally using `imgpkg`:

```
imgpkg pull -b "<IMAGE-REPOSITORY>:<TBS-VERSION>" -o /tmp/bundle
```

Where `TBS-VERSION` and `IMAGE-REPOSITORY` are the same values used during relocation.

2. Use the `Carvel` tools `kapp`, `ytt`, and `kbld` to install Build Service and define the required Build Service parameters:

Tanzu Build Service ships with a dependency updater that can update ClusterStacks, ClusterStores, ClusterBuilders, and the CNB Lifecycle from TanzuNet automatically. Enabling this feature keeps Images up to date with the latest security patches and fixes, and is *highly recommended*. To enable this feature, include your TanzuNet credentials, `descriptor_name`, and `enable_automatic_dependency_updates` when running the install command below:

```
ytt -f /tmp/bundle/config/ \
  -v kp_default_repository='<IMAGE-REPOSITORY>' \
  -v kp_default_repository_username='<REGISTRY-USERNAME>' \
  -v kp_default_repository_password='<REGISTRY-PASSWORD>' \
  --data-value-yaml pull_from_kp_default_repo=true \
  -v tanzunet_username='<TANZUNET-USERNAME>' \
  -v tanzunet_password='<TANZUNET-PASSWORD>' \
  -v descriptor_name='<DESCRIPTOR-NAME>' \
  --data-value-yaml enable_automatic_dependency_updates=true \
  | kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
  | kapp deploy -a tanzu-build-service -f- -y
```

You can check the status of the DependencyUpdater by running:

```
kubectl -n build-service get TanzuNetDependencyUpdater dependency-updater -o ya
ml
```

Alternatively, if you prefer to manage dependencies manually, leave the TanzuNet credentials, `descriptor_name`, and `enable_automatic_dependency_updates` out of the install.

```
ytt -f /tmp/bundle/config/ \
  -v kp_default_repository='<IMAGE-REPOSITORY>' \
  -v kp_default_repository_username='<REGISTRY-USERNAME>' \
  -v kp_default_repository_password='<REGISTRY-PASSWORD>' \
  --data-value-yaml pull_from_kp_default_repo=true \
  | kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
  | kapp deploy -a tanzu-build-service -f- -y
```

Where:

- ◆ `IMAGE-REPOSITORY` is the image repository where Tanzu Build Service images exist.

Note: This is identical to the `IMAGE-REPOSITORY` argument provided during `imgpkg` relocation command.

- ◆ `REGISTRY-USERNAME` is the username you use to access the registry. `gcr.io` expects `_json_key` as the username when using JSON key file authentication.
- ◆ `REGISTRY-PASSWORD` is the password you use to access the registry.

Note: [Managing Secrets](managing-secrets.html) for more information about how the registry username and password are used in Tanzu Build Service.

- ◆ `TANZUNET-USERNAME` is the username you use to access [TanzuNet](#)
- ◆ `TANZUNET-PASSWORD` is the password you use to access [TanzuNet](#)
- ◆ `DESCRIPTOR-NAME` is the name of the descriptor to import automatically. For more information about which descriptor to choose for your workload and use case, see [Descriptors](#). Available options:
 - `full` contains all dependencies.
 - `lite` smaller footprint used for speeding up installs. Requires Internet access on the cluster.

Note: You may want to pin your Tanzu Build Service to a specific descriptor version and temporarily pause the automatic update of dependencies. For details, see the FAQ section "[Pinning the Tanzu Net Updater](#)"

Installing with a CA certificate for internal registry

To install Tanzu Build Service with an internal registry that requires providing a CA certificate such as Harbor, use the normal installation command with the CA certificate file passed in with a `-f` flag:

```
ytt -f /tmp/bundle/config/ \
  -f <PATH-TO-CA> \
  -v kp_default_repository='<IMAGE-REPOSITORY>' \
  -v kp_default_repository_username='<REGISTRY-USERNAME>' \
  -v kp_default_repository_password='<REGISTRY-PASSWORD>' \
  --data-value-yaml pull_from_kp_default_repo=true \
  -v tanzunet_username='<TANZUNET-USERNAME>' \
  -v tanzunet_password='<TANZUNET-PASSWORD>' \
  -v descriptor_name='<DESCRIPTOR-NAME>' \
  --data-value-yaml enable_automatic_dependency_updates=true \
  | kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
  | kapp deploy -a tanzu-build-service -f- -y
```

Where:

- `PATH-TO-CA` is the path to the registry root CA. This CA is required to enable Build Service to interact with internally deployed registries. This is the CA that was used while deploying the registry.
- `IMAGE-REPOSITORY` is the image repository where Tanzu Build Service images exist.

Note: This is identical to the `IMAGE-REPOSITORY` argument provided during `imgpkg`

relocation command.

Exception: When using Dockerhub as your registry target, only use your DockerHub account for this value. For example, my-dockerhub-account (without /build-service). Otherwise, you encounter an error similar to: `Error: invalid credentials, ensure registry credentials for 'index.docker.io/my-dockerhub-account/build-service/tanzu-buildpacks_go' are available locally`

- `REGISTRY-USERNAME` is the username you use to access the registry. `gcr.io` expects `_json_key` as the username when using JSON key file authentication.
- `REGISTRY-PASSWORD` is the password you use to access the registry.

Note: [Managing Secrets](managing-secrets.html) for more information about how the registry username and password are used in Tanzu Build Service.

- `TANZUNET-USERNAME` is the username you use to access [TanzuNet](#)
- `TANZUNET-PASSWORD` is the password you use to access [TanzuNet](#)

Additional Configuration

You can add other optional parameters using the `-v` flag:

- `admin_users`: A comma-separated list of users who are granted admin privileges on Build Service.
- `admin_groups`: A comma-separated list of groups that are granted admin privileges on Build Service.
- `http_proxy`: The HTTP proxy to use for network traffic.
- `https_proxy`: The HTTPS proxy to use for network traffic.
- `no_proxy`: A comma-separated list of host names, IP addresses, or IP address ranges in CIDR format. These must not use a proxy.

Note: When proxy server is enabled using `http_proxy` or `https_proxy`, traffic to the Kubernetes API server also flows through the proxy server. You can circumvent this known limitation by using `no_proxy` to specify the Kubernetes API server.

Use AWS IAM for Authentication with ECR

You can install Tanzu Build Service using [AWS IAM roles](#) for authentication with Amazon Elastic Container Registry (ECR) on Amazon Elastic Kubernetes Service (EKS) clusters.

There are two additional steps required:

1. [Install Tanzu Build Service with AWS IAM for Authentication with ECR](#)
2. [Configure the build namespace for IAM Auth with ECR](#)

Install Tanzu Build Service with AWS IAM for Authentication with ECR

To install with AWS IAM for Authentication with ECR:

1. Before installing Tanzu Build Service, create a repository in ECR to use for the `kp_default_repository`.
2. Configure an AWS IAM role that has read and write access to the `kp_default_repository` and a trust policy for the EKS cluster. The minimum required permissions can be found [here](#).
3. Follow the Tanzu Build Service installation steps until the `ytt` command.
4. Install Tanzu Build Service the following `ytt` command, which replaces `kp_default_repository_username/password` with `kp_default_repository_aws_iam_role_arn`:

```
ytt -f /tmp/bundle/config/ \
-v kp_default_repository='IMAGE-REPOSITORY' \
-v kp_default_repository_aws_iam_role_arn='INSTALL-IAM-ROLE-ARN' \
-v tanzunet_username='TANZUNET-USERNAME' \
-v tanzunet_password='TANZUNET-PASSWORD' \
-v descriptor_name='DESCRIPTOR-NAME' \
--data-value-yaml enable_automatic_dependency_updates=true \
| kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
| kapp deploy -a tanzu-build-service -f- -y
```

Where:

- ◆ `IMAGE-REPOSITORY` is the image repository where Tanzu Build Service images are located.
- ◆ `INSTALL-IAM-ROLE-ARN` is the AWS IAM role Amazon Resource Name (ARN) for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-install-role`.
- ◆ `TANZUNET-USERNAME` is the username you use to access VMware Tanzu Network.
- ◆ `TANZUNET-PASSWORD` is the password you use to access VMware Tanzu Network.
- ◆ `DESCRIPTOR-NAME` is the name of the descriptor to import automatically. For more information about which descriptor to choose for your workload and use case, see [Descriptors](#). Available options:
 - `full` contains all dependencies.
 - `lite` smaller footprint used for speeding up installs. Requires Internet access on the cluster.

Configure the build namespace for IAM Auth with ECR

Similar to configuring a secret in the build namespace for registry authentication, the build namespace requires configuration for Tanzu Build Service to use AWS IAM authentication for ECR in builds.

To configure the build namespace for IAM Auth with ECR:

1. Configure an AWS IAM role that has read and write access to the ECR registry location to store build images. This can be the same repository used during installation, in which case, you can use the same AWS ARN.
2. For the service account used for `Image` resources, add an annotation including the role ARN configured earlier.

```
kubectl annotate serviceaccount -n BUILD-NAMESPACE SERVICE-ACCOUNT-NAME \
eks.amazonaws.com/role-arn=${BUILD-IAM-ROLE-ARN}
```

Where:

- ◆ `BUILD-NAMESPACE` is the name of your build namespace.
- ◆ `SERVICE-ACCOUNT-NAME` is the service account used for `Image` resources. This is `default` if unset.
- ◆ `BUILD-IAM-ROLE-ARN` is the AWS IAM role ARN for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-build-role`.

3. To create an `Image` resource with this service account, run:

```
kp image save my-ecr-image --tag 12345.ecr.com/my-repo --service-account=SERVICE-ACCOUNT-NAME -n BUILD-NAMESPACE
```

Configuring TKGI as an OIDC Provider

The authentication and authorization processes for Build Service use a combination of RBAC rules and third-party authentication, including OpenID Connect (OIDC). You may configure UAA as an OIDC provider for your TKGI deployment to provide authentication for Build Service.

To configure UAA as an OIDC provider for your TKGI deployment:

1. Navigate to the OpsManager Installation Dashboard.
2. Click the TKGI tile.
3. Select **UAA**.
4. Under **Configure created clusters to use UAA as the OIDC provider**, select **Enable**.
5. Ensure the values in the **UAA OIDC Groups Prefix** and **UAA OIDC Username Prefix** fields are the same and record them. For example, `"oidc:"`. You need these values during the installation of Build Service.

Note: Ensure you add a `:` at the end of the desired prefix.

6. Click **Save**.
7. In the OpsManager Installation Dashboard, click **Review Pending Changes**, then **Apply Changes**.

Installation to Air-Gapped Environment

Note: The `TanzuNetDependencyUpdater` cannot be used in air-gapped environments. Do not include Tanzu Net credentials for air-gapped installations.

Tanzu Build Service can be installed to a Kubernetes Cluster and registry that are air-gapped from external traffic.

An air-gapped environment often uses an internal registry with a self-signed CA certificate and you need access to this CA certificate file to install Tanzu Build Service.

Note: If you are using a CA certificate that is trusted (eg. Lets Encrypt) you do not need the CA certificate file.

Relocate Images to a Registry (Air-Gapped)

This procedure relocates images from the Tanzu Network registry to an internal image registry via a local machine.

The local machine must have write access to the internal registry.

1. Log in to the image registry where you want to store the images by running:

- ✦ If using the `docker` CLI, run:

```
docker login <IMAGE-REGISTRY>
```

Where `IMAGE-REGISTRY` is the name of the image registry where you want to store the images.

- ✦ If using the `crane` CLI, run:

```
crane auth login <IMAGE-REGISTRY>
```

Where `IMAGE-REGISTRY` is the name of the image registry where you want to store the images.

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

- ✦ If using the `docker` CLI, run:

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

- ✦ If using the `crane` CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

3. Copy the Tanzu Build Service bundle to your local machine as a tar with the [Carvel](#) tool `imgpkg` by running:

```
imgpkg copy -b registry.tanzu.vmware.com/build-service/bundle:<TBS-VERSION> --to-tar=/tmp/tanzu-build-service.tar
```

Where `TBS-VERSION` is the version of Tanzu Build Service you want to install.

4. Move the output file `tanzu-build-service.tar` to a machine that has access to the air-gapped environment.
5. Unpackage the images from your local machine to the internal registry:

```
imgpkg copy --tar /tmp/tanzu-build-service.tar \
  --to-repo=<IMAGE-REPOSITORY> \
  --registry-ca-cert-path <PATH-TO-CA>
```

Where:

- ✦ `IMAGE-REPOSITORY` is the repository in your registry that you want to relocate images to.

- ◆ `PATH-TO-CA` is the path to the registry CA certificate file.

Note: The `IMAGE-REPOSITORY` must be the `IMAGE-REGISTRY` appended with the destination repository for the images. For example, `IMAGE-REGISTRY/some-repo/build-service`.

Exception: When relocating to Dockerhub, you must provide the Dockerhub username and an image name that `imgpkg` uses for relocation. For example, `my-dockerhub-account/build-service`.

For example:

- ◆ Dockerhub `imgpkg copy --tar /tmp/tanzu-build-service.tar --to-repo=my-dockerhub-account/build-service --registry-ca-cert-path ca.crt`
- ◆ GCR `imgpkg copy --tar /tmp/tanzu-build-service.tar --to-repo=gcr.io/my-project/build-service --registry-ca-cert-path ca.crt`
- ◆ Artifactory `imgpkg copy --tar /tmp/tanzu-build-service.tar --to-repo=artifactory.com/my-project/build-service --registry-ca-cert-path ca.crt`
- ◆ Harbor `imgpkg copy --tar /tmp/tanzu-build-service.tar --to-repo=harbor.io/my-project/build-service --registry-ca-cert-path ca.crt`

Installing (Air-Gapped)

Note: The `TanzuNetDependencyUpdater` cannot be used in air-gapped environments. Do not include Tanzu Net credentials for air-gapped installations.

Once the images have been relocated, installation is the same as a regular install.

1. Pull the Tanzu Build Service bundle image locally using `imgpkg`:

```
imgpkg pull -b "<IMAGE-REPOSITORY>:<TBS-VERSION>" -o /tmp/bundle
```

Where `TBS-VERSION` and `IMAGE-REPOSITORY` are the same values used during relocation.

2. Use the [Carvel](#) tools `kapp`, `ytt`, and `kbld` to install Build Service and define the required Build Service parameters by running:

```
ytt -f /tmp/bundle/config/ \
  -f <PATH-TO-CA> \
  -v kp_default_repository='<IMAGE-REPOSITORY>' \
  -v kp_default_repository_username='<REGISTRY-USERNAME>' \
  -v kp_default_repository_password='<REGISTRY-PASSWORD>' \
  --data-value-yaml pull_from_kp_default_repo=true \
  --data-value-yaml exclude_dependencies=true \
  | kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
  | kapp deploy -a tanzu-build-service -f- -y
```

Where:

- ◆ `PATH-TO-CA` is the path to the registry root CA. This CA is required to enable Build Service to interact with internally deployed registries. This is the CA that was used while deploying the registry.

- ◆ `IMAGE-REPOSITORY` is the image repository where Tanzu Build Service images exist.
Note: This is identical to the `IMAGE-REPOSITORY` argument provided during `imgpkg` relocation command.
- ◆ `REGISTRY-USERNAME` is the username you use to access the registry. `gcr.io` expects `_json_key` as the username when using JSON key file authentication.
- ◆ `REGISTRY-PASSWORD` is the password you use to access the registry.
Note: [Managing Secrets](managing-secrets.html) for more information about how the registry username and password are used in Tanzu Build Service.

Additional Configuration

You can add other optional parameters using the `-v` flag:

- `admin_users`: A comma-separated list of users who are granted admin privileges on Build Service.
- `admin_groups`: A comma separated list of groups that are granted admin privileges on Build Service.

Import Tanzu Build Service Dependencies (Air-Gapped)

The Tanzu Build Service Dependencies (Stacks, Buildpacks, Builders, etc.) are used to build applications and keep them patched.

For air-gapped environments, dependencies must be imported with the `kp` cli and the Dependency Descriptor bundle image (`registry.tanzu.vmware.com/tbs-dependencies/full`) from the [Tanzu Build Service Dependencies](#) page.

Relocate Tanzu Build Service Dependency Images (Air-Gapped)

1. First, find the latest version of the descriptor from the [Tanzu Build Service Dependencies](#) page.
2. To import these dependencies into an air-gapped environment, they must first be pulled locally. Use `imgpkg` and the `<VERSION>` from the previous step:

```
imgpkg copy -b registry.tanzu.vmware.com/tbs-dependencies/full:<VERSION> \
  --to-tar=tbs-dependencies.tar
```

Note: You must be logged in locally to the Tanzu Network registry.

3. Move the output file `tbs-dependencies.tar` to a machine that has access to the air-gapped environment.
4. Upload the dependencies to the Tanzu Build Service registry by running:

```
imgpkg copy --tar=tbs-dependencies.tar \
  --to-repo <IMAGE-REPOSITORY> \
  --registry-ca-cert-path <PATH-TO-CA>
```

Where:

- ◆ `IMAGE-REPOSITORY` is the internal image repository where dependency images are relocated. This should be the same as `kp_default_repository` from installation.
- ◆ `PATH-TO-CA` is the path to the registry CA certificate file.

Note: You must be logged in locally to the registry used for `IMAGE-REPOSITORY`.

Import Tanzu Build Service Dependency Resources (Air-Gapped)

After the dependency images are uploaded to the internal registry, you can successfully import these images and create the corresponding Tanzu Build Service resources.

Run the following commands with `imgpkg`, `kbld`, and the `kp` CLI:

```
imgpkg pull -b <IMAGE-REPOSITORY>:<VERSION> \
  -o /tmp/descriptor-bundle \
  --registry-ca-cert-path <PATH-TO-CA>

kbld -f /tmp/descriptor-bundle/.imgpkg/images.yml \
  -f /tmp/descriptor-bundle/tanzu.descriptor.v1alpha3/descriptor-<VERSION>.yaml \
  | kp import -f - --registry-ca-cert-path <PATH-TO-CA>
```

Verify Installation

Verify your Build Service installation by first targeting the cluster Build Service has been installed on.

To verify your Build Service installation:

1. Download the `kp` binary from the [Tanzu Build Service](#) page on Tanzu Network.
2. List the cluster builders available in your installation:

```
kp clusterbuilder list
```

You should see an output that looks as follows:

NAME	READY	STACK	IMAGE
base	true	io.buildpacks.stacks.bionic	<image@sha256:digest>
default	true	io.buildpacks.stacks.bionic	<image@sha256:digest>
full	true	io.buildpacks.stacks.bionic	<image@sha256:digest>
tiny	true	io.paketo.stacks.tiny	<image@sha256:digest>

Upgrading Tanzu Build Service

To upgrade Tanzu Build Service to a newer version, run the same commands as [installation](#), `kapp` updates resources if they already exist. Re-importing dependencies is not required for upgrading Tanzu Build Service.

Uninstalling Tanzu Build Service

To uninstall Tanzu Build Service simply run the following `kapp` command:

```
kapp delete -a tanzu-build-service
```

Note: All Tanzu Build Service resources are deleted. Registry images created by Tanzu Build Service is not be deleted.

Updating Build Service Dependencies

Note: If you enabled the `TanzuNetDependencyUpdater` during install (This can be verified by running `kubectl get TanzuNetDependencyUpdater -A`` you do not need to do anything to manage your Tanzu Build Service dependencies

Use the following documentation to keep applications patched and up-to-date with Tanzu Build Service:

To keep dependencies up-to-date, see [Updating Build Service Dependencies](#)

To manage Stacks, see [Managing Stacks](#)

To manage Buildpack Stores, see [Managing Stores](#)

Ensuring Access to Cluster Builders

In order to use Cluster Builders, such as the ones installed with Tanzu Build Service, we suggest to install Tanzu Build Service to a repository that is accessible by the nodes in the kubernetes cluster without credentials.

If this is not desired, see [When to use Synchronized Secrets](#).

Ensuring the Run Image is Readable

Build Service relies on the run-image being publicly readable or readable with the registry credentials configured in a project/namespace for the builds to be executed successfully.

The location of the run image can be identified by running the following command:

```
kp clusterstack status <stack-name>
```

If the cluster stack run image is not public, you may need to create a registry secret in any namespace where Images or Builds are used. For more details on secrets in Tanzu Build Service, see [Managing Secrets](#)

This can be done with the `kp` CLI:

```
kp secret create my-registry-creds --registry example-registry.io --registry-user my-registry-user --namespace build-namespace
```

Next Steps

Visit the [Managing Images and Builds](#) page to learn how to create and manage a new image.

Kubernetes Permissions for Installation

The minimum Kubernetes RBAC permissions required to install Tanzu Build Service are as follows.

This includes the namespaces required for the Kubernetes Roles:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: build-service
---
apiVersion: v1
kind: Namespace
metadata:
  name: kpack
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: build-service-install-cluster-role
rules:
- apiGroups:
  - "admissionregistration.k8s.io"
  resources:
  - mutatingwebhookconfigurations
  - validatingwebhookconfigurations
  verbs:
  - '*'
- apiGroups:
  - "rbac.authorization.k8s.io"
  resources:
  - clusterroles
  - clusterrolebindings
  verbs:
  - '*'
- apiGroups:
  - "apiextensions.k8s.io"
  resources:
  - customresourcedefinitions
  verbs:
  - '*'
- apiGroups:
  - "storage.k8s.io"
  resources:
  - storageclasses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - kpack.io
  resources:
  - builds
  - builds/status
  - builds/finalizers
  - images
  - images/status
  - images/finalizers
  - builders
  - builders/status
  - clusterbuilders
  - clusterbuilders/status

```

```

    - clusterstores
    - clusterstores/status
    - clusterstacks
    - clusterstacks/status
    - sourceresolvers
    - sourceresolvers/status
  verbs:
    - '*'
- apiGroups:
  - "projects.vmware.com"
  resources:
    - projects
  verbs:
    - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: build-service-install-role
  namespace: build-service
rules:
- apiGroups:
  - ""
  resources:
    - configmaps
    - secrets
    - serviceaccounts
    - services
    - namespaces
  verbs:
    - '*'
- apiGroups:
  - "rbac.authorization.k8s.io"
  resources:
    - roles
    - rolebindings
  verbs:
    - '*'
- apiGroups:
  - apps
  resources:
    - deployments
    - daemonsets
  verbs:
    - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kpack-install-role
  namespace: kpack
rules:
- apiGroups:
  - ""
  resources:
    - services
    - serviceaccounts
    - namespaces
    - secrets

```

```

    - configmaps
  verbs:
    - '*'
- apiGroups:
  - "rbac.authorization.k8s.io"
  resources:
    - roles
    - rolebindings
  verbs:
    - '*'
- apiGroups:
  - apps
  resources:
    - deployments
    - daemonsets
  verbs:
    - '*'

```

The `kapp` command used to install Tanzu Build Service requires access to ConfigMaps in the namespace used to run `kapp`:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kapp-role
  namespace: <my-kapp-namespace>
rules:
  - apiGroups:
    - ""
    resources:
    - configmaps
    verbs:
    - '*'

```

Where the namespace `<my-kapp-namespace>` must be the namespace of the Kubernetes context that `kapp` is run in.

Minimum AWS IAM permissions for ECR IAM auth

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:PutImageTagMutability",
        "ecr:StartImageScan",
        "ecr:DescribeImageReplicationStatus",
        "ecr:ListTagsForResource",
        "ecr:UploadLayerPart",
        "ecr:BatchDeleteImage",
        "ecr:ListImages",
        "ecr:BatchGetRepositoryScanningConfiguration",
        "ecr>DeleteRepository",
        "ecr:CompleteLayerUpload",
        "ecr:TagResource",

```

```

        "ecr:DescribeRepositories",
        "ecr:BatchCheckLayerAvailability",
        "ecr:ReplicateImage",
        "ecr:GetLifecyclePolicy",
        "ecr:PutLifecyclePolicy",
        "ecr:DescribeImageScanFindings",
        "ecr:GetLifecyclePolicyPreview",
        "ecr:PutImageScanningConfiguration",
        "ecr:GetDownloadUrlForLayer",
        "ecr>DeleteLifecyclePolicy",
        "ecr:PutImage",
        "ecr:UntagResource",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:StartLifecyclePolicyPreview",
        "ecr:InitiateLayerUpload",
        "ecr:GetRepositoryPolicy"
    ],
    "Resource": "<your repository>"
},
{
    "Sid": "VisualEditor1",
    "Effect": "Allow",
    "Action": [
        "ecr:GetRegistryPolicy",
        "ecr:BatchImportUpstreamImage",
        "ecr:CreateRepository",
        "ecr:DescribeRegistry",
        "ecr:DescribePullThroughCacheRules",
        "ecr:GetAuthorizationToken",
        "ecr:PutRegistryScanningConfiguration",
        "ecr:CreatePullThroughCacheRule",
        "ecr>DeletePullThroughCacheRule",
        "ecr:GetRegistryScanningConfiguration",
        "ecr:PutReplicationConfiguration"
    ],
    "Resource": "<your repository>"
}
]
}

```

Accessing Tanzu Build Service

To use your Build Service installation, gain kubeconfig access to the Kubernetes Cluster that has the Build Service installed. For example, if you are using TKGI (formerly PKS):

```
tkgi login -a <tkg-api-url> -u <username> -p <password>
tkgi get-credentials <clustername> -a <tkg-api-url> -u <username> -p <password>
```

You can use the `kp` CLI, downloaded as part of the [installation](#) to interact with Build Service. The `kp` CLI uses the local `KUBECONFIG` utilized by `kubectl`. All operations are performed on Kubernetes current-context namespace.

The `kp` CLI help text is published [here](#).

```
$ kp
kp controls the kpack installation on Kubernetes.

kpack extends Kubernetes and utilizes unprivileged kubernetes primitives to provide
builds of OCI images as a platform implementation of Cloud Native Buildpacks (CNB).
Learn more about kpack @ https://github.com/pivotal/kpack

Usage:
  kp [command]

Available Commands:
  build           Build Commands
  builder        Builder Commands
  clusterbuilder Cluster Builder Commands
  clusterstack   Cluster Stack Commands
  clusterstore   Cluster Store Commands
  completion     Generate completion script
  help           Help about any command
  image          Image commands
  import         Import dependencies for stores, stacks, and cluster builders
  secret        Secret Commands
  version        Display kp version

Flags:
  -h, --help  help for kp

Use "kp [command] --help" for more information about a command.
```

Note: These docs assume `kp` cli v0.6.* from TBS release v1.6.*. If a feature is not working, you may need to upgrade your cli.

Updating Build Service Dependencies

Keeping applications up to date with the latest dependency patches is a core feature of Tanzu Build Service. Updates to dependencies are propagated to application images. The resources that account for these patches are:

- **ClusterStacks** - Update a ClusterStack to patch operating system packages.
- **ClusterStores** - Update a ClusterStore to patch the Cloud Native Buildpacks used to build your applications.

You can use the `kp` CLI to update any resource. See the `kpack-cli help text` on GitHub.

Note: These docs assume `kp cli v0.6.*` from Tanzu Build Service release `v1.6.*`. If a feature is not working, you may need to upgrade your cli.

Updating dependencies

Automatically update dependencies

Tanzu Build Service ships with a dependency updater that can update ClusterStacks, ClusterStores, ClusterBuilders, and the CNB Lifecycle from TanzuNet automatically. Enabling this feature keeps images up to date with the latest security patches and fixes.

You can run `kubectl get TanzuNetDependencyUpdater -A` to check if you have a `TanzuNetDependencyUpdater` set up already. If you have one, there is nothing you need to do to manage your dependencies in Tanzu Build Service.

If you would like to enable this feature after install, you can create the following resources:

1. A secret with your TanzuNet credentials (`kp secret create dependency-updater-secret --registry registry.tanzu.vmware.com --registry-user <TANZUNET_USERNAME>`) in the namespace where you would like your dependency updater to be.
2. A service account that contains that secret. (If the secret was created using `kp`, it is automatically added to the `default` service account in that namespace.)
3. A `TanzuNetDependencyUpdater` resource:

```
---
apiVersion: buildservice.tanzu.vmware.com/v1alpha1
kind: TanzuNetDependencyUpdater
metadata:
  name: dependency-updater
  namespace: <NAMESPACE>
spec:
  serviceAccountName: <SERVICE-ACCOUNT>
  productSlug: tbs-dependencies
```

```

checkEvery: 1m
descriptorName: <DESCRIPTOR-NAME>
descriptorVersion: <DESCRIPTOR-VERSION>

```

- The `productSlug` field corresponds to the product name in TanzuNet.
- The `checkEvery` field is the frequency that the updater checks for new descriptor file releases.
- The `serviceAccountName` field is the name of the service account from step 2.
- The `descriptorName` field is the name of the descriptor to import automatically. Available options can be found on the [Tanzu Network Build Service Dependencies](#) page. Currently available options at time of release:
 - `full` contains all dependencies - for production use.
 - `lite` smaller footprint used for speeding up installs. Requires Internet access on the cluster.
- The `descriptorVersion` (optional) field can be used to pin to a specific version of the descriptor. This is only recommended for use to protect from breaking changes. This can usually be left blank.

Bulk update

Note: If you want to be alerted when a new descriptor file is published, we recommend using an RSS reader and watching the Tanzu Build Service Dependencies Tanzu Network feed for updates <https://network.tanzu.vmware.com/rss>.

The bulk update workflow can update all dependencies in Tanzu Build Service, such as ClusterStacks, ClusterStores, and ClusterBuilders, using the `kp import` command.

1. Download the dependency descriptor file (`descriptor-<version>.yaml`) from the latest release on the [Tanzu Build Service Dependencies](#) page on Tanzu Network.

Note: You can see all of the buildpackages versions that are imported by looking at the `buildpackage-versions-.yaml` file from the [Tanzu Build Service Dependencies](#) release.

2. Log in to the Tanzu Network registry, using both URLs.

- If using the `docker` CLI, run:

```

docker login registry.tanzu.vmware.com
docker login registry.pivotal.io

```

- If using the `crane` CLI, run:

```

crane auth login registry.tanzu.vmware.com
crant auth login registry.pivotal.io

```

3. Use the `kp` CLI.

Warning: Tanzu Build Service ships with a automatic dependency updater. If you have enabled this feature during install by passing in your TanzuNet credentials, you **must** skip this step. To check if you have a `TanzuNetDependencyUpdater` in your cluster, run: `kubectl get TanzuNetDependencyUpdaters -A`

```
kp import -f descriptor-<version>.yaml
```

The following ClusterStacks are updated with the latest operating system patches: [base](#), [default](#), [full](#), and [tiny](#).

The following ClusterStore is updated with the latest Cloud Native Buildpacks: [default](#).

Using the `--show-changes` flag gives a summary of the resource changes for the import. You are also asked to confirm the import. Confirmation can be skipped with `--force`.

Cluster stacks update

This section describes how to update individual cluster stacks. This provides a more fine-grained way to patch operating system packages.

New stack versions are provided on the [Tanzu Build Service Dependencies](#) page on Tanzu Network.

To update specific cluster stacks, go to the latest release of the [Tanzu Build Service Dependencies](#) page on Tanzu Network to find the image references and their `<sha256>` sums. Example commands are provided on this page.

Use the following `kp` CLI commands to update the desired stack:

```
kp clusterstack update base \
  --build-image registry.tanzu.vmware.com/tbs-dependencies/build-base@<sha256> \
  --run-image registry.tanzu.vmware.com/tbs-dependencies/run-base@<sha256>

kp clusterstack update default \
  --build-image registry.tanzu.vmware.com/tbs-dependencies/build-full@<sha256> \
  --run-image registry.tanzu.vmware.com/tbs-dependencies/run-full@<sha256>

kp clusterstack update full \
  --build-image registry.tanzu.vmware.com/tbs-dependencies/build-full@<sha256> \
  --run-image registry.tanzu.vmware.com/tbs-dependencies/run-full@<sha256>

kp clusterstack update tiny \
  --build-image registry.tanzu.vmware.com/tbs-dependencies/build-tiny@<sha256> \
  --run-image registry.tanzu.vmware.com/tbs-dependencies/run-tiny@<sha256>
```

Note: Both build and run images need to be provided to update the stack.

To view the updated ClusterStack, run:

```
kp clusterstack status <stack-name>
```

Example output:

```
$ kp clusterstack status tiny
Status:      Ready
Id:         io.paketo.stacks.tiny
Run Image:   gcr.io/build-service-dev/test/run@sha256:34b01fd9a3745fcaa345f89939382
91c931f7977cc2bee78ed377da2edc55e3d
Build Image: gcr.io/build-service-dev/test/build@sha256:5288d9c5b7cf7068d07b5a184f3
ec2f124fbc5842401b8b23c74485c4d2ba23a
```

Cluster store update

ClusterStores contain all of the buildpackages (one or more packaged Cloud Native Buildpacks) to be used by builders to build application images.

You can update Cloud Native Buildpacks in Tanzu Build Service by adding new buildpackage versions to the store.

To list the buildpackages available in a store:

```
kp clusterstore status <store-name>
```

Example output:

```
$ kp clusterstore status default
Status:    Ready

BUILDPACKAGE ID          VERSION   HOMEPAGE
paketo-buildpacks/procfile  1.4.0    https://github.com/paketo-buildpacks/procfile
tanzu-buildpacks/dotnet-core 0.0.3
tanzu-buildpacks/go        1.0.5
tanzu-buildpacks/httpd    0.0.38
tanzu-buildpacks/java     2.5.0    https://github.com/pivotal-cf/tanzu-java
tanzu-buildpacks/nginx    0.0.45
tanzu-buildpacks/nodejs   1.1.0
tanzu-buildpacks/php      0.0.3
```

To show a complete list of all buildpacks available in a store, run:

```
kp clusterstore status <store-name> --verbose
```

To update a store with one or more buildpackages, run:

```
kp clusterstore add <store-name> -b <buildpackage-image1> -b <buildpackage-image2>
```

Note: Any number of buildpackages can be added to a store at a time with multiple `-b` flags.`

Updating buildpacks From Tanzu Network

New Cloud Native Buildpacks (packaged as buildpackages) are available on [Tanzu Network](#) and can be uploaded to a Tanzu Build Service to keep application images patched.

New versions of the Java, NodeJS, and Go buildpacks are released on their respective Tanzu Network pages:

- [Java](#)
- [NodeJS](#)
- [Go](#)

New versions of all other buildpacks are released on the [Tanzu Build Service Dependencies](#) page.

Here is a list of how to update each buildpack that is included with Tanzu Build Service by default:

```
kp clusterstore add default -b registry.tanzu.vmware.com/tanzu-java-buildpack/java:<ve
```

```

rsion>
kp clusterstore add default -b registry.tanzu.vmware.com/tanzu-nodejs-buildpack/nodejs
:<version>
kp clusterstore add default -b registry.tanzu.vmware.com/tanzu-go-buildpack/go:<versio
n>
kp clusterstore add default -b registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildp
acks_dotnet-core:<version>
kp clusterstore add default -b registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildp
acks_php:<version>
kp clusterstore add default -b registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildp
acks_nginx:<version>
kp clusterstore add default -b registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildp
acks_httpd:<version>
kp clusterstore add default -b registry.tanzu.vmware.com/tbs-dependencies/paketo-build
packs_procfile:<version>

```

Additionally, multiple buildpackages can be added to Build Service by passing multiple image references:

```

kp clusterstore add <store-name> \
  -b registry.tanzu.vmware.com/buildpackage-1 \
  -b registry.tanzu.vmware.com/buildpackage-2 \
  -b registry.tanzu.vmware.com/buildpackage-3

```

Offline update of dependencies

If your Tanzu Build Service installation is in an offline/air-gapped environment, you can update stores with the following offline workflow:

1. Find the latest version of the Dependency Descriptor bundle image (registry.tanzu.vmware.com/tbs-dependencies/full) from the latest release on the [Tanzu Build Service Dependencies](#) page on Tanzu Network.
2. Download the following CLIs for your operating system:
 - ◆ [kp](#)
 - ◆ [imgpkg](#)
 - ◆ [kbld](#)
3. Log in to the Tanzu Network registry with your Tanzu Network credentials:

- ◆ If using the `docker` CLI, run:

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

- ◆ If using the `crane` CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

4. Download the dependency images for Tanzu Build Service to your local machine with `imgpkg` using the `VERSION` found from Tanzu Network in step 1:

```
imgpkg copy -b registry.tanzu.vmware.com/tbs-dependencies/full:<VERSION> \
  --to-tar=tbs-dependencies.tar
```

5. Move the output file `tbs-dependencies.tar` to a machine that has access to the "offline" environment.
6. Log in to the image registry used to deploy Tanzu Build Service:

- If using the `docker` CLI, run:

```
docker login <build-service-registry>
```

- If using the `crane` CLI, run:

```
crane auth login <build-service-registry>
```

7. Upload the dependency images to the registry used to deploy Tanzu Build Service by running:

```
imgpkg copy --tar=tbs-dependencies.tar \
  --to-repo <IMAGE-REPOSITORY>
```

Where `IMAGE-REPOSITORY` is the repository used to install Tanzu Build Service. This should be the same value as `IMAGE-REPOSITORY` used in the [Installation Steps](#).

8. Now that you have relocated the dependencies to the internal registry, update the resources by running:

```
imgpkg pull -b <IMAGE-REPOSITORY>:<VERSION> \
  -o /tmp/descriptor-bundle \
  --registry-ca-cert-path <PATH-TO-CA>

kblld -f /tmp/descriptor-bundle/.imgpkg/images.yml \
  -f /tmp/descriptor-bundle/tanzu.descriptor.v1alpha3/descriptor-<VERSION>.yaml \
  | kp import -f -
```

Managing Secrets

Overview

VMware Tanzu Build Service uses Kubernetes secrets to manage credentials.

- To publish images to a Registry, you must use a Registry secret.
- To use source code stored in a private Git repository, you must use a Git secret.

Secrets are namespaced and can only be used for image configurations that exist in the same namespace. For more information about Kubernetes secrets, see [Secrets](#) in the Kubernetes documentation.

For more information about secret synchronization, see the [Secret Synchronization](#) page.

You can manage secrets with the `kp` CLI. For more information, see the [help text](#) on GitHub.

```
$ kp secret
Secret Commands

Usage:
  kp secret [command]

Available Commands:
  create      Create a secret configuration
  delete      Delete secret
  list        List secrets

Flags:
  -h, --help  help for secret

Use "kp secret [command] --help" for more information about a command.
```

Note: These docs assume `kp cli v0.6.*` from Tanzu Build Service release `v1.6.*`. If a feature is not working, you may need to upgrade your cli.

Create Secrets

You can create secrets using the `kp` CLI and script them with environment variables.

Secrets are created in the Kubernetes `current-context` namespace, unless you specify a different namespace using the `--namespace` or `-n` flag. Kubernetes automatically adds these secrets to the `default` service account in the same namespace.

Note: The `kp` CLI does not validate the secret against the specified registry or Git at the time of secret creation. Incorrect credentials are reported as they are used during an image build.

Create a Docker Hub registry secret

You can create a Docker Hub registry secret using the `--dockerhub` flag.

```
kp secret create SECRET-NAME --dockerhub DOCKER-HUB-ID
```

Where:

- `SECRET-NAME` is the name you give your secret.
- `DOCKER-HUB-ID` is your Docker Hub user ID.

When prompted, enter your Docker Hub password. Alternatively, you can use the `DOCKER_PASSWORD` environment variable to bypass the password prompt.

The Docker Hub registry secret is stored as a `kubernetes.io/dockerconfigjson` secret.

For example:

```
$ kp secret create secret1 --dockerhub my-dockerhub-id
dockerhub password:
"secret1" created

$ DOCKER_PASSWORD="my-password" kp secret create secret2 --dockerhub my-dockerhub-id
"secret2" created
```

Create a GCR registry secret

You can create a GCR registry secret by using the `--gcr` flag.

```
kp secret create SECRET-NAME --gcr GCR-SERVICE-ACCOUNT-PATH
```

Where:

- `SECRET-NAME` is the name you give your secret.
- `GCR-SERVICE-ACCOUNT-PATH` is the path to your GCR service account JSON file.

Alternatively, use the `GCR_SERVICE_ACCOUNT_PATH` environment variable instead of the `--gcr` flag.

The GCR registry secret is stored as a `kubernetes.io/dockerconfigjson` secret.

For example:

```
$ kp secret create secret1 --gcr /tmp/my-gcr-service-account.json
"secret1" created

$ GCR_SERVICE_ACCOUNT_PATH="/tmp/my-gcr-service-account.json" kp secret create secret2
"secret2" created
```

Create an Artifactory, Harbor, or ACR registry secret

You can create an Artifactory, Harbor, or ACR secret using the `--registry` and `--registry-user` flags:

```
kp secret create SECRET-NAME --registry REGISTRY-URL --registry-user REGISTRY-USER-ID
```


Where:

- `SECRET-NAME` is the name you give your secret.
- `REGISTRY-URL` is the URL of the registry. This should only be the domain for the registry and should not contain folders or projects. Example: `registry.io` and not `registry.io/project`.
- `REGISTRY-USER-ID` is your registry user ID.

When prompted, enter your registry password. Alternatively, you can use the `REGISTRY_PASSWORD` environment variable to bypass the password prompt.

The Artifactory, Harbor, or ACR registry secret is stored as a `kubernetes.io/dockerconfigjson` secret.

For example:

```
$ kp secret create secret1 \
  --registry registry.tanzu.vmware.com \
  --registry-user someuser@pivotal.io
registry password:
"secret1" created

$ REGISTRY_PASSWORD="my-password" kp secret create secret2 \
  --registry registry.tanzu.vmware.com \
  --registry-user someuser@pivotal.io
"secret2" created
```

Create a Git SSH secret

You can create a Git SSH secret by specifying a Git SSH URL and private SSH key.

```
kp secret create SECRET-NAME --git-url GIT-SSH-URL --git-ssh-key PRIVATE-SSH-KEY-PATH
```

Where:

- `SECRET-NAME` is the name you give your secret.
- `GIT-SSH-URL` is the Git SSH domain URL. This is not the full repository URL. For example, value should be `git@github.com` for GitHub.
- `PRIVATE-SSH-KEY-PATH` is the path to your private SSH key.

Alternatively, use the `GIT_SSH_KEY_PATH` environment variable instead of the `--git-ssh-key` flag.

The Git SSH secret is stored as a `kubernetes.io/ssh-auth` secret.

For example:

```
$ kp secret create secret1 \
  --git-url git@github.com \
  --git-ssh-key /tmp/private-repo-git-deploy-key
"secret1" created

$ GIT_SSH_KEY_PATH="/tmp/private-repo-git-deploy-key" kp secret create secret2 \
  --git-url git@github.com \
  "secret2" created
```

Create a Git basic auth secret

You can create a Git basic auth secret by providing your Git username and password:

```
kp secret create SECRET-NAME --git-url GIT-DOMAIN-URL --git-user GIT-USERNAME
```

Where:

- `SECRET-NAME` is the name you give your secret.
- `GIT-DOMAIN-URL` is the Git domain url. This is not the full repository url. For example, value should be `https://github.com` for GitHub.
- `GIT-USERNAME` is your Git username.

When prompted, enter your Git password. Alternatively, you can use the `GIT_PASSWORD` environment variable to bypass the password prompt.

The Git basic auth secret is stored as a `kubernetes.io/basic-auth` secret.

For example:

```
$ kp secret create secret1 \
  --git-url https://github.com \
  --git-user someone@vmware.com
git password:
"secret1" created

$ GIT_PASSWORD="my-password" kp secret create secret2 \
  --git-url https://github.com \
  --git-user someone@vmware.com
"secret2" created
```

List secrets

To list the names and the targets for your secrets:

```
kp secret list
```

Unless you specify a namespace using the `--namespace` or `-n` flag, running the `kp secret list` command lists secrets for the Kubernetes `current-context` namespace.

For example:

```
$ kp secret list
NAME                TARGET
default-token-qrdbr
docker-hub-creds    https://index.docker.io/v1/
gcr-creds           gcr.io
git-creds           https://github.com
git-ssh-creds       git@github.com
harbor-creds        registry.tanzu.vmware.com
```

The `default-token-xxxxxx` secret is automatically added to the `default` service account by Kubernetes.

Delete secrets

To delete secrets:

```
kp secret delete SECRET-NAME
```

Where:

- `SECRET-NAME` is the name of the secret you want to delete.

Unless you specify a namespace using the `--namespace` or `-n` flag, secrets are deleted from the Kubernetes `current-context` namespace. There is no confirmation required from the user.

Encrypting secrets at rest

Because Tanzu Build Service uses standard Kubernetes secrets, admins may configure the cluster to encrypt secrets at rest. For more information, see [Encrypting Secret Data at Rest](#).

Using SecretGen controller Secrets to use private cluster builders

Note: As of TBS 1.6, Synced Secrets are disabled by default. To enable the synced secrets feature use `enable_synced_secrets: true` during installation.

When to use Synchronized Secrets

To enable the use of cluster builders from private registries, a secret with registry credentials must exist in the namespace of the image using that cluster builder.

You can configure this secret manually for each namespace, but Tanzu Build Service in tandem with the Carvel [secret-gen-controller](#) provides functionality to synchronize secrets across namespaces to simplify this process.

This feature is applicable in the following cases:

- You have installed Tanzu Build Service to a private registry and do not wish to make cluster builders imported by `kp` publicly readable.
- You have used `kp` to create a cluster builder in a private registry and do not wish to make it publicly readable.

Synchronized secrets are attached to build pods as `imagePullSecrets` so that the cluster builder image can be pulled at build time.

Note: Synchronizing secrets with write access is not recommended. Instead, create and synchronize read-only secrets. A synced secret is not created during installation because the credentials provided for installation must be writable.

Installing the Carvel secret-gen-controller

In order to manage secrets across namespaces, Tanzu Build Service utilizes the carvel secret-gen-controller. Tanzu Build Service automatically creates placeholder secrets in namespaces that contain

Build resources and then the secret-gen-controller populates these placeholders across namespaces with the corresponding sync-secret in the secret-syncer namespace.

Please follow the secret-gen-controller [install docs](#) to install the controller.

Managing secret synchronization

Currently, the `kp` CLI does not support adding and removing synchronized secrets. However, this may be achieved by using the `kubect1` CLI.

Create a synchronized secret

To start synchronizing a secret to all namespaces with builds, first create the secret in the build-service namespace by using the `kubect1` cli.

Example secret:

```
apiVersion: v1
data:
  .dockerconfigjson: <SECRET DATA>
kind: Secret
metadata:
  name: my-synced-secret
  namespace: build-service
type: kubernetes.io/dockerconfigjson
```

Create a secretExport resource

Example secretExport resource:

```
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: my-synced-secret
  namespace: build-service
spec:
  toNamespace: "*"

```

Once the Tanzu Build Service secret-syncer controller creates the placeholder, this secret is automatically copied to the build namespace.

For more detailed information on the carvel secret-gen-controller, please review the [carvel docs](#)

Update a synchronized secret

To update a secret and roll out those changes to all namespaces that use builds, update the secret(s) that have a corresponding secretExport resource located in the `build-service` namespace. After this secret is updated, the change is rolled out across namespaces.

Stop synchronizing a secret

To stop synchronizing a secret, delete the secret from the `build-service` namespace or remove the corresponding secretExport resource from the secret located in the `build-service` namespace.

Managing Image Resources and Builds

This topic contains the following sections:

- [Image Resources](#)
- [Builds](#)

The `kp` CLI can be used to manage image resources and builds. The help text is published in the [kpack-cli documentation](#) on GitHub.

```
$ kp image
Image commands

Usage:
  kp image [command]

Aliases:
  image-resource, images, imgs, img

Available Commands:
  create      Create an image resource
  delete      Delete an image resource
  list        List image resources
  patch       Patch an existing image resource
  save        Create or patch an image resource
  status      Display status for an image resource
  trigger     Trigger an image resource build

Flags:
  -h, --help  help for image

Use "kp image [command] --help" for more information about a command.
```

```
$ kp build
Build Commands

Usage:
  kp build [command]

Aliases:
  build, builds, blds, bld

Available Commands:
  list      List builds for an image resource
  logs      Tails logs for an image resource build
  status    Display status for an image resource build

Flags:
  -h, --help  help for build
```

```
Use "kp build [command] --help" for more information about a command.
```

Note: These docs assume `kp cli v0.6.*` from TBS release `v1.6.*`. If a feature is not working, you might need to upgrade your cli.

Image Resources

Image resources provide a configuration for Tanzu Build Service to build and maintain a Docker image utilizing Tanzu, Paketo, and custom [Cloud Native Buildpacks](#).

Build Service monitors the inputs to the image resource to rebuild the image when the underlying source or buildpacks have changed.

The following procedures describe how to create and manage image resources in Build Service with the `kp` CLI.

Creating Image Resources

Prerequisites:

- [Access to a cluster](#) running Build Service.
- Configured write [secrets](#) for your Docker registry.

Source Code

The `kp` CLI supports creating Image Resources using source code from the following locations:

- [Git based source](#)
- [Blob store](#)
- [Local machine](#)

You can specify only one location for app source code.

Builders

Users can select a Builder (namespaced-scoped) or a Cluster Builder (cluster-scoped) used to create image resource builds. You can use any of the available Builders or Cluster Builders with any of the source types (git, blob, or local).

If you do not use the `--builder` or `--cluster-builder` flags, the `default` Cluster Builder is used.

For more information on Builders, see [Managing Builders](#).

Creating an image resource with source code in a Git repository

To create an image resource using source code from a git repository, run:

```

kp image create <name> \
  --tag <tag> \
  [--builder <builder> or --cluster-builder <cluster-builder>] \
  --namespace <namespace> \
  --env <env> \
  --sub-path <sub-path> \
  --wait \
  --git <git-repo> \
  --git-revision <git-revision>

```

Where:

- `name` is the name of the image resource.
- `tag` is the registry location where the image is created.
- `builder` (optional) is the builder name used in the image resource. Cannot be used with `cluster-builder`.
- `cluster-builder` (optional) is the Cluster Builder name to be used in the image resource. Defaults to `default` when `builder` is not set. Cannot be used with `builder`.
- `namespace` (optional) is the Kubernetes namespace for the image resource. Defaults to the local Kubernetes current-context namespace.
- `env` (optional) is the image resource environment variable configuration as key=val pairs (`env_var=env_val`). The `--env` flag can be specified multiple times.
- `sub-path` (optional) is the build code at the subpath located within the source code directory.
- `cache-size` (optional) is the cache size used for subsequent builds. Must be a valid kubernetes quantity (default 2G).
- `wait` flag (optional) waits for image create to be reconciled and tails resulting build logs.
- `git-repo` is the Git repository URL of the source code.
- `git-revision` (optional) is the Git revision of the code that the image is built against. Can be either a `branch`, `tag` or a commit `sha`. When you target the image resource against a branch, Build Service triggers a build for every new commit. Defaults to `master`.

Note: If the `git-repo` is a private repository, you must configure the Git credentials. For more information, see [Create Secrets](#).

Create an image resource with source code in a blob store

Users can specify source code in a blob store saved as a compressed file (`zip`, `tar.gz`, `.tar`) or a `.jar` file.

To create an image resource using source code from blob store:

```

kp image create <name> \
  --tag <tag> \
  [--builder <builder> or --cluster-builder <cluster-builder>] \
  --namespace <namespace> \
  --env <env> \
  --sub-path <sub-path> \

```

```
--wait \  
--blob <blob-url>
```

Where:

- `name` is the name of the image resource.
- `tag` is the registry location where the image is created.
- `builder` (optional) is the builder name used in the image resource. Cannot be used with `cluster-builder`.
- `cluster-builder` (optional) is the Cluster Builder name to be used in the image resource. Defaults to `default` when `builder` is not set. Cannot be used with `builder`.
- `namespace` (optional) is the Kubernetes namespace for the image resource. Defaults to the local Kubernetes current-context namespace.
- `env` (optional) is the image resource environment variable configuration as key=val pairs (`env_var=env_val`). The `--env` flag can be specified multiple times.
- `sub-path` (optional) is the build code at the subpath located within the source code directory.
- `cache-size` (optional) is the cache size used for subsequent builds. Must be a valid kubernetes quantity (default 2G).
- `wait` flag (optional) waits for image create to be reconciled and tails resulting build logs.
- `blob-url` is the URL of the source code blob file.

Note: The source code file in the blob store must be publicly viewable or the `blob-url` must contain the basic authentication credentials.

Creating an image resource with local source code

Users can apply local source code from a directory, compressed source code (`zip`, `tar.gz`, `.tar`), or a `.jar` file.

To create an image resource using source code from a local machine, run:

```
kp image create <name> \  
--tag <tag> \  
--local-path <source-path> \  
[--builder <builder> or --cluster-builder <cluster-builder>] \  
--namespace <namespace> \  
--env <env> \  
--cache \  
--registry-ca-cert-path <path-to-ca-cert> \  
--registry-verify-certs
```

Where:

- `name` is the name of the image resource.
- `tag` is the registry location where the image is created.
- `source-path` is the path to local source code.
- `builder` (optional) is the builder name used in the image resource. Cannot be used with

`cluster-builder`.

- `cluster-builder` (optional) is the Cluster Builder name used in the image resource. Defaults to `default` when `builder` is not set. Cannot be used with `builder`.
- `namespace` (optional) is the Kubernetes namespace for the image resource. Defaults to the local Kubernetes current-context namespace.
- `env` (optional) is the image resource environment variable configuration as key=val pairs (`env_var=env_val`). The `--env` flag can be specified multiple times.
- `cache-size` (optional) is the cache size used for subsequent builds. Must be a valid kubernetes quantity (default 2G).
- `--wait` flag (optional) waits for image create to be reconciled and tails resulting build logs.
- `registry-ca-cert-path` (optional) adds CA certificate for registry API.
- `registry-verify-certs` (optional) sets whether to verify server's certificate chain and host name (default true).

Buildpack configuration

Image resources use buildpacks to build application images in a registry. The buildpacks contain the dependencies needed for these builds and you can add buildpack configuration to Tanzu Build Service image resources.

Buildpack configuration use cases

Common use cases for setting buildpack configuration include:

- Selecting a specific version or version line of a dependency (Go 1.15.*, Java 1.8)
- Language-specific configuration (Go build target)
- Buildpack-specific configuration

Buildpack configuration documentation

Buildpack configuration details can be found in the documentation for that specific buildpack.

Use `kp clusterstore status <store-name> --verbose` to find the homepage of the desired buildpack.

Buildpack configuration in image resources

Buildpack configuration, including manually selecting buildpacks to use, can be set in two ways in Tanzu Build Service Image Resources. The configuration depends on the specific buildpack, find buildpack details in [Buildpack Configuration Documentation](#).

1. Creating a `buildpack.yml` file at the root of the application source code.

Example `buildpack.yml` for a Go app to use the latest Go 1.15 version and build with the path `./cmd/package`:

```
go:
```

```
version: 1.15.*
targets: ["/cmd/package"]
```

2. Setting environment variables on an image resource.

Tanzu Build Service image resources can have environment variables configured that are set in all Builds and in the final exported registry image. These can be used for buildpack configuration.

Example `kp` command to create an image resource for a Go app to build with the path `./cmd/package`:

```
kp image create my-image \
  --tag registry.io/my-repo \
  --git https://github.com/my-go-app \
  --env BP_GO_TARGETS="./cmd/package"
```

Patching image resources

Users can patch their existing image resources with the `kp` CLI. Running a patch triggers a new build of the image resource if any of the build inputs are changed.

Patch image resources with the following commands:

- With source code in a Git repository

```
kp image patch <name> \
  [--builder <builder> or --cluster-builder <cluster-builder>] \
  --namespace <namespace> \
  --env <env> \
  --wait \
  --git <git-repo> \
  --git-revision <git-revision>
```

- With source code in a blob store

```
kp image patch <name> \
  [--builder <builder> or --cluster-builder <cluster-builder>] \
  --namespace <namespace> \
  --env <env> \
  --wait \
  --blob <blob-url>
```

- With local source code

```
kp image patch <name> \
  [--builder <builder> or --cluster-builder <cluster-builder>] \
  --namespace <namespace> \
  --env <env> \
  --wait \
  --local-path <source-path>
```

Where:

- `name` is the name of the image resource to patch.
- `namespace` (optional) is the Kubernetes namespace for the image resource. Defaults to the

local Kubernetes current-context namespace.

- `env` (optional) is the image resource environment variable configuration as key=val pairs (`env_var=env_val`). The `--env` flag can be specified multiple times.
- `cache-size` (optional) is the cache size used for subsequent builds. Must be a valid Kubernetes quantity (default 2G).
- `git-repo` is the Git repository URL of the source code. Must select one of `git-repo`, `blob-url`, or `source-path`
- `git-revision` (optional) is the Git revision of the code that the image is built against. Can be either a `branch`, `tag` or a commit `sha`. When you target the image resource against a branch, Build Service triggers a build for every new commit. Defaults to `master`.
- `blob-url` is the URL of the source code blob file. Must select one of `git-repo`, `blob-url`, or `source-path`
- `source-path` is the path to local source code. Must select one of `git-repo`, `blob-url`, or `source-path`

Note: If the `git-repo` is a private repository, you must configure the Git credentials. For more information, see [Create Secrets](#).

Note: The `tag` location in a registry and `name` of an image resource cannot be modified. To change these fields, you must create a new image resource.

Saving image resources

Users can create or patch an image resource using the `save` command. The `kp image save` command is used exactly the same as `kp image create` or `kp image patch`, but it determines whether an image resource needs to be created or patched.

Note:For handling source code changes in the Tanzu Build Service process, VMware recommends utilizing the `kp image save --wait`` command within a CI/CD pipeline to update the source code referenced in the image resource.

You can do this by updating the `--git-revision`` field with a new commit ID. This commit ID references source code that has undergone unit testing, so the resulting image can be deployed or promoted to higher level environments.

```
kp image save my-image \
  --tag my-registry.com/my-repo \
  --git https://my-repo.com/my-app.git \
  --git-revision my-branch
```

Listing images

To list all the image resources in a Kubernetes namespace:

```
kp image list --namespace <namespace>
```

For example:

```
$ kp image list -n example1
```

NAME	READY	LATEST REASON	LATEST IMAGE	NAMESPACE
test-image1	True	CONFIG	first/image:sha	example1
test-image2	False	BUILDPACK	second/image:sha	example1

To list all the image resources across all Kubernetes namespaces:

```
kp image list --all-namespaces
```

For example:

```
$ kp image list -A
```

NAME	READY	LATEST REASON	LATEST IMAGE	NAMESPACE
test-image1	True	CONFIG	first/image:sha	example1
test-image2	True	BUILDPACK	second/image:sha	example1
test-image3	True	BUILDPACK	third/image:sha	example2
test-image4	False	CONFIG	fourth/image:sha	example2

Filter image resources

Users can further filter the list of image resources by applying the `--filter` flag and specifying a filter and value. This command is useful for traversing large number of image resources by narrowing the list to only display image resources that possess certain attributes.

```
$ kp image list --filter ready=false -A
```

NAME	READY	LATEST REASON	LATEST IMAGE	NAMESPACE
test-image2	False	BUILDPACK	second/image:sha	example1
test-image4	False	CONFIG	fourth/image:sha	example2

See below for the currently supported filters and values:

```
builder=string
clusterbuilder=string
latest-reason=commit,trigger,config,stack,buildpack
ready=true,false,unknown
```

Image resource rebuilds

Rebuilds happen in three ways:

1. An imperative rebuild occurs when you patch an image resource with `kp image patch`.
2. An automatic rebuild occurs when build inputs change (source code, stack, or buildpacks).
3. A user can trigger a rebuild manually.

An imperative rebuild is initiated if any of the following changes are made to an image resource:

- An update to the commit, branch, Git repository, or other arguments to `kp image patch`.
- You upload a new copy of the local source code by running `kp image patch --local-path <source-path>`, where `<source-path>` is the source code path.

For more information, see [Patching image resources](#).

Build Service auto-rebuilds image resources when one or more of the following build inputs change:

- New buildpack versions are made available through updates to a Cluster Store.
 - ◊ New Buildpack versions are made available on [Tanzu Network](#).
 - ◊ To update buildpacks, you must add new buildpack versions from Tanzu Network to a Cluster Store. See [Updating Build Service Dependencies](#) for more details.
- There is a new commit on a branch or tag Tanzu Build Service is tracking.
- There is a new Cluster Stack (that is, base OS image) available, such as [full](#), [tiny](#), or [base](#).
 - ◊ New Stack versions are made available on the [Tanzu Build Service Dependencies](#) page on Tanzu Network.
 - ◊ You can get updates to Stacks from the Tanzu Network Registry by using the `kp` CLI. See [Updating Build Service Dependencies](#) for more details.

Trigger an image resource rebuild

You can initiate a manual rebuild using `kp`:

```
kp image trigger <image-name> --namespace <namespace>
```

This is useful for debugging image resource builds.

Viewing the status of an image resource

When a user creates an image resource using the above workflow, they are configuring Tanzu Build Service to start creating builds of the image resource that create container images to be pushed to a registry.

If a particular build associated with an image resource fails, check the status of the image resource by running:

```
kp image status <image-name> --namespace <namespace>
```

Where `image-name` is the name of the image resource. See [Listing image resources](#) to get image names.

The following is an example output of this command:

```
Status:          Not Ready
Message:         --
LatestImage:     gcr.io/myapp@sha256:9d7b1fbf7f5cb0f8efe797f30e598b5e38bb1c08ada143d4c
96e4f78111a9239

Last Successful Build
Id:              1
Reason:         CONFIG

Last Failed Build
Id:              2
Reason:         COMMIT
```

Deleting an image resource

This procedure describes how to delete a Build Service image resource with the `kp` CLI.

Warning: Deleting an image resource deletes the image resource and all the builds that the image resource owns. It does not delete the app images generated by those builds from the registry.

To delete an image resource:

```
kp image delete <image> --namespace <namespace>
```

Where `image` is the name of the image resource.

When you successfully delete an image resource, you see this message:

```
"<image>" deleted
```

Managing image resources with YAML

Build Services image resources can be created by applying the [kpack image resources](#) to cluster through `kubectl`.

Use the `default` service account for Build Service registry and Git secrets.

Image resource additional tags

With the addition of `kpack apiVersion kpack.io/v1alpha2`, additional tags can be specified on image resources. Additional tags are not currently configurable through the `kp` cli, applying YAML configuration with `kubectl` is required. See [kpack docs](#) for details.

Using a registry for caching

Tanzu Build Service image resource can be configured to use a registry as the build cache instead of a persistent volume. Currently, configuring the registry cache is not supported with `kp` and `kubectl` must be used.

For more details, see the [kpack image config](#) docs for how to set the cache tag.

Using secrets

Use the `default` service account for Build Service registry and Git secrets. `kpack` will default to the `default` service account if no service account is specified.

Debugging with image resource status

Using `kubectl` is a good way to debug image resources.

When an image resource has successfully built with its current configuration, its status reports the up-to-date fully qualified built image reference.

This information is available with `kubectl get image <image-name> -o yaml`.

```

status:
  conditions:
  - lastTransitionTime: "2020-01-17T16:16:36Z"
    status: "True"
    type: Succeeded
  - lastTransitionTime: "2020-01-17T16:16:36Z"
    status: "True"
    type: BuilderReady
  latestImage: index.docker.io/sample/image@sha256:d3eb15a6fd25cb79039594294419de2328f
  14b443fa0546fa9e16f5214d61686
  ...

```

When a build fails, the image resource status reports the condition Succeeded=False. The image resource status also includes the status of the builder used by the image resource. If the builder is not ready, you may want to inspect that builder. More details in [Managing Builders](#).

```

status:
  conditions:
  - lastTransitionTime: "2020-01-17T16:13:48Z"
    status: "False"
    type: Succeeded
    message: "Some error occurred"
  - lastTransitionTime: "2020-01-17T16:16:36Z"
    status: "False"
    type: BuilderReady
    message: "Some builder error occurred"
  ...

```

If further debugging is required, inspect the image resource's latest Build status discussed in [Viewing Build Details for an Image Resource](#).

Image resource service bindings

Tanzu Build Service supports application service bindings as described in the Kubernetes [Service Bindings specification](#).

The `kp` CLI does not currently support creating service bindings, you should use `kubectl`.

Creating an image resource with service bindings

For more information, see [Service Bindings](#) in the kpack documentation on GitHub.

To create a service binding in your application image, you must create *either* of the following:

- A [Provisioned service](#)
- A Kubernetes secret that follows the guidelines provided in [Well-known Secret Entry](#)

Note: Check the desired buildpack documentation for details on the service bindings it supports. You can access these docs on [Language Family Buildpacks for VMware Tanzu] (<https://docs.vmware.com/en/VMware-Tanzu-Buildpacks/services/tanzu-buildpacks/GUID-index.html>).

The following is an example you can use with `kubectl apply`. It creates a `production-db` service binding for a maven app.

Example:

```

apiVersion: kpack.io/v1alpha2
kind: Image
metadata:
  name: sample-binding-with-secret
spec:
  tag: my-registry.com/repo
  builder:
    kind: ClusterBuilder
    name: default
  source:
    git:
      url: https://github.com/buildpack/sample-java-app.git
      revision: 0eccc6c2f01d9f055087ebbf03526ed0623e014a
  build:
    services:
      - name: production-db-secret
        kind: Secret
---
apiVersion: v1
kind: Secret
metadata:
  name: production-db-secret
type: servicebinding.io/mysql
stringData:
  type: mysql
  provider: bitnami
  host: localhost
  port: 3306
  username: root
  password: root

```

Builds

The procedures in this section describe how to view information and logs for image resource builds using the `kp` CLI.

Listing builds

Build Service stores the ten most recent successful builds and the ten most recent failed builds.

To see a the list of builds for an image resource, run:

```
kp build list <image-name> --namespace <namespace>
```

If the `namespace` is not specified, it defaults to the Kubernetes current-context namespace. If the `image-name` is not specified, the builds for all the image resources in your namespace are listed.

The following is an example of the output for this command:

BUILD	STATUS	IMAGE	REASON
1	SUCCESS	gcr.io/myapp@sha256:some-sha1	CONFIG
2	SUCCESS	gcr.io/myapp@sha256:some-sha2	COMMIT
3	SUCCESS	gcr.io/myapp@sha256:some-sha3	STACK
4	FAILURE	gcr.io/myapp@sha256:some-sha4	CONFIG+
5	BUILDING	gcr.io/myapp@sha256:some-sha5	BUILDPACK

The following describes the fields in the example output:

- **BUILD** describes the index of builds in the order they were built.
- **STATUS** describes the status of a previous build image.
- **IMAGE** is the full image reference for the app image produced by the build.
- **REASON** describes why an image rebuild occurred. These reasons include:
 - ◊ **CONFIG**: Occurs when a change is made to commit, branch, Git repository, or build fields on the image's configuration file and you run `kp image apply`.
 - ◊ **COMMIT**: Occurs when new source code is committed to a branch or tag that Build Service is monitoring for changes.
 - ◊ **BUILDPACK**: Occurs when new buildpack versions are made available through an updated builder.
 - ◊ **STACK**: Occurs when a new base OS image, called a `run image`, is available.
 - ◊ **TRIGGER**: Occurs when a new build is manually triggered.

Note: A rebuild can occur for more than one reason. When there are multiple reasons for a rebuild, the `kp` CLI output shows the primary `Reason` and appends a `+` sign to the `Reason` field. The priority rank for the `Reason`, from highest to lowest, is `CONFIG`, `COMMIT`, `BUILDPACK`, `STACK`, and `TRIGGER`.

Viewing build details for an image

To display retrieve a detailed bill of materials for a particular build:

```
kp build status <image> -b <build-number>
```

Where:

- `image-name` is the name of the image resource the build is associated with
- `build-name` (optional) is the index of the build from [listing builds](#). Defaults to latest build.

The following is an example of the output for this command:

```
Image:      gcr.io/myapp@sha256:f87b614257af05c3301c1554c4f15131793caec3adf55e45d2c612
e90445765a
Status:     SUCCESS
Reason:     CONFIG
            resources
            - source: {}
            + source:
            +   git:
            +   revision: 948b2eff6a21580a44a0f4d8c609a2af45359d41
            +   url: https://github.com/paketo-buildpacks/samples
```

```

+   subPath: go/mod

Started:    2021-02-02 18:34:33
Finished:   2021-02-02 18:41:03

Pod Name:   build-pod-xyz

Builder:    gcr.io/my-builder:base@sha256:grtewwads0asdvf09asdf
Run Image:  gcr.io/base-image:run@sha256:asdas098asdas

Source:     Git
Url:        http://github.com/myapp
Revision:   ad123ad

BUILDPACK ID      BUILDPACK VERSION
io.java.etc       123
io.kotlin.etc     321

```

The following describes the fields in the example output:

- **Image** is the full image reference for the app image produced by the build.
- **Status** describes the status of a previous build image.
- **Reason** describes why an image resource build occurred and the change diff. The reason could be one or more of these:
 - ◊ **CONFIG**: Occurs when a change is made to commit, branch, Git repository, or build fields on the image's configuration file and you run `kp image apply`.
 - ◊ **COMMIT**: Occurs when new source code is committed to a branch or tag that Build Service is monitoring for changes.
 - ◊ **BUILDPACK**: Occurs when new buildpack versions are made available through an updated builder.
 - ◊ **STACK**: Occurs when a new base OS image (called a `run image`) is available.
 - ◊ **TRIGGER**: Occurs when a new build is manually triggered.
- **Started** is when a build started.
- **Finished** is when a build finished.
- **Pod Name** is the name of the Pod being used for the Build.
- **Builder** is the full image tag for the builder image used by the build.
- **Run Image** is the full image tag for the run image used by the app.
- **Source** describes where the source code used to build the image is coming from. Can be `Git`, `Blob`, or `Local Source`.
- **Url** is the Git repository URL for `Git` source, the Blob file URL for `Blob` source. Unset for `Local Source`.
- **Revision** is the Git commit sha of the source code used to create the build for `Git` source.
- **BUILDPACK ID** is a list of buildpack ids the build used.
- **BUILDPACK VERSION** is a list of buildpack versions the build used.

Image resource status shows ImagePullBackOff

If the Build is currently waiting for a container, the Build status shows details in the output of `kp build status`.

Here is an example output:

```
Image:          --
Status:        BUILDING
Reason:        CONFIG
Status Reason: ImagePullBackOff
Status Message: A container image currently cannot be pulled: Back-off pulling image
                "gcr.io/my-builder:base@sha256:grtewwads0asdvf09asdf"

Pod Name:      build-pod-xyz

Builder:       gcr.io/my-builder:base@sha256:grtewwads0asdvf09asdf
Run Image:     gcr.io/base-image:run@sha256:asd098asd098

Source:        Git
Url:           http://github.com/myapp
Revision:      ad123ad

BUILDPACK ID      BUILDPACK VERSION
```

If you see this error and you are using a Cluster Builder, you might need to configure a Synced Secret. See [When to use Synchronized Secrets](#).

Getting build logs

An image resource that a user creates causes builds to be initiated for that image. Builds are where Cloud Native Buildpacks are run and apps get built into images.

Build logs are a good way to debug issues and to get information about how your app is being built.

If you get logs of a build in progress, the logs are tailed and terminate when the build completes.

To get logs from a build run:

```
kp build logs <image> --build <build-number> --namespace <namespace>
```

Where:

- `image-name` is the name of the image resource the build is associated with.
- `build-name` (optional) is the index of the build from [listing builds](#). Defaults to latest build.

The following is an example of the output of the command:

```
====> PREPARE
Build reason(s): CONFIG
CONFIG:
  resources: {}
- source: {}
+ source:
+   git:
```

```

+     revision: 446dbda043ca103d33e2cad389d43f289e63f647
+     url: https://github.com/some-org/some-repo
Loading secret for "gcr.io" from secret "gcr" at location "/var/build-secrets/gcr"
Cloning "https://github.com/some-org/some-repo" @ "446dbda043ca103d33e2cad389d43f289e63f647"...
Successfully cloned "https://github.com/some-org/some-repo" @ "446dbda043ca103d33e2cad389d43f289e63f647" in path "/workspace"
===> DETECT
tanzu-buildpacks/node-engine 0.1.2
tanzu-buildpacks/npm-install 0.1.1
tanzu-buildpacks/npm-start   0.0.2
===> ANALYZE
Previous image with name "gcr.io/test-app" not found
===> RESTORE
===> BUILD
Tanzu Node Engine Buildpack 0.1.2
  Resolving Node Engine version
  Candidate version sources (in priority order):
    -> ""
    <unknown> -> "*"

  Selected Node Engine version (using ): 14.15.1

  Executing build process
  Installing Node Engine 14.15.1
  Completed in 2.495s

  Configuring environment
  NODE_ENV      -> "production"
  NODE_HOME     -> "/layers/tanzu-buildpacks_node-engine/node"
  NODE_VERBOSE -> "false"

  Writing profile.d/0_memory_available.sh
  Calculates available memory based on container limits at launch time.
  Made available in the MEMORY_AVAILABLE environment variable.

Tanzu NPM Install Buildpack 0.1.1
  Resolving installation process
  Process inputs:
    node_modules      -> "Not found"
    npm-cache         -> "Not found"
    package-lock.json -> "Not found"

  Selected NPM build process: 'npm install'

  Executing build process
  Running 'npm install --unsafe-perm --cache /layers/tanzu-buildpacks_npm-install/npm-cache'
  Completed in 3.591s

  Configuring environment
  NPM_CONFIG_LOGLEVEL -> "error"
  NPM_CONFIG_PRODUCTION -> "true"
  PATH                -> "$PATH:/layers/tanzu-buildpacks_npm-install/modules/node_modules/.bin"

Tanzu NPM Start Buildpack 0.0.2
  Assigning launch processes
  web: node server.js

```

```

====> EXPORT
Adding layer 'tanzu-buildpacks/node-engine:node'
Adding layer 'tanzu-buildpacks/npm-install:modules'
Adding layer 'tanzu-buildpacks/npm-install:npm-cache'
Adding 1/1 app layer(s)
Adding layer 'launcher'
Adding layer 'config'
Adding label 'io.buildpacks.lifecycle.metadata'
Adding label 'io.buildpacks.build.metadata'
Adding label 'io.buildpacks.project.metadata'
*** Images (sha256:0abdbaf1f25c3c13cdb918d06906670b84dd531bc7301177b11284dac68bdb9c) :
    gcr.io/test-app
    gcr.io/test-app:b1.20210203.225422
Adding cache layer 'tanzu-buildpacks/node-engine:node'
Adding cache layer 'tanzu-buildpacks/npm-install:modules'
Adding cache layer 'tanzu-buildpacks/npm-install:npm-cache'
====> COMPLETION
Build successful

```

Viewing bill of materials

The `kp cli` allows you to view the bill of materials in an image built by a Build.

```
kp build status <image-name> --bom
```

To generate the bill of materials, the `kp CLI` reads metadata from the image (generated by the build) in the registry.

Note: You must have credentials to access the image registry on your machine.

For example:

```

$ kp build status --bom my-app-image | jq
[
  {
    "buildpack": {
      "id": "tanzu-buildpacks/node-engine",
      "version": "0.1.2"
    },
    "metadata": {
      "licenses": [],
      "name": "Node Engine",
      "sha256": "b981046a0ea3d5594a7f04fae3afdfa1983bc65f4e26e768b38a2d67057ac75c",
      "stacks": [
        "io.buildpacks.stacks.bionic",
        "org.cloudfoundry.stacks.cflinuxfs3"
      ],
      "uri": "file:///dependencies/b981046a0ea3d5594a7f04fae3afdfa1983bc65f4e26e768b38a2d67057ac75c",
      "version": "14.15.1"
    },
    "name": "node",
    "version": "14.15.1"
  },
  {
    "buildpack": {
      "id": "tanzu-buildpacks/npm-install",

```

```

    "version": "0.1.1"
  },
  "metadata": {
    "launch": true
  },
  "name": "node_modules"
}
]

```

Offline builds

Tanzu Build Service supports offline/air-gapped builds with Tanzu Buildpacks. Offline builds use pre-packaged dependencies and do not need to download from anywhere off-cluster to create application images.

When using Tanzu Buildpacks the build will execute as an offline build. For details on how to configure buildpacks, see [Buildpack configuration in images](#).

Note: Offline builds only ensure buildpack dependencies are offline. The application build and custom configuration must also not reach off-cluster to be completely offline.

Image signing with cosign

Tanzu Build Service supports [cosign](#) image signing.

Images signed with cosign require using `kubect1` instead of `kp`.

Cosign signing secret

Images can be signed with cosign when a cosign formatted secret is added to the service account used to build the image. The secret can be added using the cosign CLI or manually.

To create a cosign signing secret through the cosign CLI, when targetted to the Kubernetes cluster, use:

```
cosign generate-key-pair k8s://[NAMESPACE]/[NAME]
```

Alternatively, create the cosign secret and provide your own cosign key files manually to Kubernetes by running the following command:

```
% kubectl create secret generic <secret-name> --from-literal=cosign.password=<password>
> --from-file=</path/to/cosign.key>
```

- `<secret-name>` is the name of the secret. Ensure that the secret is created in the same namespace as the eventual image resource.
- `<password>` is the password provided to encrypt the private key. If not present, an empty password will be used.
- `</path/to/cosign.key>` is the cosign private key file generated with `cosign generate-key-pair`.

After adding the cosign secret, the secret must be added to the list of `secrets` attached to the service account resource that is building the image.

Adding cosign annotations

By default, the build number and build timestamp information is added to the cosign signing annotations. Users can specify additional cosign annotations under the spec key.

```
cosign:
  annotations:
  - name: "annotationName"
    value: "annotationValue"
```

One way these annotations can be viewed is through verifying cosign signatures. The annotations are under the `optional` key in the verified JSON response. For example, you can do this by running:

```
% cosign verify -key /path/to/cosign.pub registry.example.com/project/image@sha256:<DIGEST>
```

Which provides a JSON response similar to:

```
{
  "critical": {
    "identity": {
      "docker-reference": "registry.example.com/project/image"
    }, "image": {
      "docker-manifest-digest": "sha256:<DIGEST>"
    }, "type": "cosign container image signature"
  }, "optional": {
    "buildNumber": "1",
    "buildTimestamp": "20210827.175240",
    "annotationName": "annotationValue"
  }
}
```

Push cosign signature to a different location

Cosign signatures can be pushed to a different registry from where the image is located. To enable this, add the corresponding annotation to the cosign secret resource:

```
metadata:
  name: ...
  namespace: ...
  annotations:
    kpack.io/cosign.repository: other.registry.com/project/image
data:
  cosign.key: ...
  cosign.password: ...
```

This is equivalent to setting `COSIGN_REPOSITORY` as specified in cosign [Specifying Registry](#) on GitHub.

The same service account that has that cosign secret attached, and is used for signing and building the image, requires that the registry credentials for this other repository be placed under the listed `secrets`. It is not required to be listed in `imagePullSecrets`. To push the signatures to the same registry but to a different path from the image, the credential used must have access to both paths. You cannot use two separate credentials for the same registry with different paths.

Cosign legacy Docker media types

To sign images in a registry that does not fully support OCI media types, you can use legacy equivalents by adding the corresponding annotation to the cosign secret resource:

```
metadata:
  name: ...
  namespace: ...
  annotations:
    kpack.io/cosign.docker-media-types: "1"
data:
  cosign.key: ...
  cosign.password: ...
```

This is equivalent to setting `COSIGN_DOCKER_MEDIA_TYPES=1` as specified in the cosign [registry-support](#).

Managing ClusterStacks

A ClusterStack is a cluster scoped resource that provides the build and run images for the [Cloud Native Buildpack stack](#) that will be used in a [Builder](#).

Most users automatically configure three ClusterStack resources through the Tanzu Build Service installation process. These ClusterStacks are referenced in three corresponding ClusterBuilder resources.

For more information about security and patching cadence for these stacks and their ideal use cases, see [Stacks](#). More detailed release notes for the stacks can be accessed by following the links:

Name	ID
tiny	io.paketo.stacks.tiny
base	io.buildpacks.stacks.bionic
full	io.buildpacks.stacks.bionic

The `kp` CLI can be used to manage clusterstack. See the `kp`-cli [help text](#).

```

$ kp clusterstack
Cluster Stack Commands

Usage:
  kp clusterstack [command]

Aliases:
  clusterstack, csk

Available Commands:
  create      Create a cluster stack
  delete     Delete a cluster stack
  list       List cluster stacks
  save       Create or update a cluster stack
  status     Display cluster stack status
  update     Update a cluster stack

Flags:
  -h, --help  help for clusterstack

Use "kp clusterstack [command] --help" for more information about a command.
```

Note: These docs assume `kp` cli v0.6.* from Tanzu Build Service release v1.6.*. If a feature is not working, you may need to upgrade your cli.

Note: Only Build Service Admins (that is, users with the `pb-admin-role` kubernetes ClusterRole) can perform clusterstack commands.

Create a ClusterStack

Users can create a clusterstack by using build and run images from a Docker registry or the local machine. The run and build images provided during clusterstack creation are uploaded to the `canonical repository`, which is the `docker-repository` specified during Tanzu Build Service install.

- If you use a Docker registry for the stack images:

```
kp clusterstack create <clusterstack-name> \
  --build-image <location of build-image> \
  --run-image <location of run-image>
```

Note: The user must have read access to the source Docker registry and write access to the canonical registry on the local machine.

For example:

```
kp csk create my-clusterstack \
  -b gcr.io/test/stack/run:latest
  -r gcr.io/test/stack/build:latest
```

- If you use local stack images created with `docker save`:

```
kp clusterstack create <clusterstack-name> \
  --build-image <path to build-image>.tar \
  --run-image <path to run-image>.tar
```

Note: The user must have write access to the canonical registry on the local machine.

For example:

```
kp csk create my-clusterstack \
  -b ./local-build-image.tar \
  -r ./local-run-image.tar
```

Update a ClusterStack

Users can update a stack by using build and run images from a Docker registry or the local machine. The run and build images provided during clusterstack update are uploaded to the `canonical repository`, which is the `docker-repository` specified during Tanzu Build Service install.

- If using a Docker registry:

```
kp clusterstack update <stack-name> \
  --build-image <location of build-image> \
  --run-image <location of run-image>
```

Note: The user must have read access to the source Docker registry and write access to the canonical registry on the local machine.

For example:

```
kp csk update my-clusterstack \
  -b gcr.io/test/stack/run:latest
```

```
-r gcr.io/test/stack/build:latest
```

- If using local stack images created with `docker save`:

```
kp clusterstack update <stack-name> \
  --build-image <path to build-image>.tar \
  --run-image <path to run-image>.tar
```

Note: The user must have write access to the canonical registry on the local machine.

For example:

```
kp csk update my-clusterstack \
  -b ./local-build-image.tar \
  -r ./local-run-image.tar
```

Save a ClusterStack

Users can create or update a ClusterStack by using the `save` command. The `kp clusterstack save` command is used exactly the same as `kp clusterstack create` and `kp clusterstack update`, but it determines whether a clusterstack needs to be created or updated.

Get ClusterStack status

Users can get the current status of a clusterstack by running:

```
kp clusterstack status <stack-name>
```

The following is an example of the output for this command:

```
Status:      Ready
ID:          org.cloudfoundry.stacks.cflinuxfs3
Run Image:   paketo/run:full-cnb
Build Image: paketo/build:full-cnb
```

Delete a ClusterStack

Users can delete an existing clusterstack by running:

```
kp clusterstack delete <stack-name>
```

Note: The user is not asked for a confirmation before deletion.

List all ClusterStacks

Users can view the list of all ClusterStacks created:

The following is an example of the output for this command:

NAME	READY	ID
base	True	io.buildpacks.stacks.bionic
default	True	io.buildpacks.stacks.bionic

```
full      True      org.cloudfoundry.stacks.cflinuxfs3
tiny     True      io.paketo.stacks.tiny
```

How to update an image for Stack updates only?

To achieve Stack only updates for an Image, you can [pin the Buildpack versions](#) in the Builder used for creating the image.

Managing Stores

A Store is a cluster-level resource that provides a collection of buildpacks that can be utilized by Builders. Buildpacks are distributed and added to a store in buildpackages, which are docker images containing one or more buildpacks.

Build Service ships with a curated collection of Tanzu buildpacks for Java, Nodejs, Go, PHP, nginx, and httpd and Paketo buildpacks for profile, and .NET Core. Detailed documentation about the buildpacks installed with Tanzu Build Service can be found [here](#). It is important to keep these buildpacks up-to-date. Updates to these buildpacks are provided on [Tanzu Network](#).

In addition to supported Tanzu and Paketo buildpacks, custom buildpackages can be uploaded to Build Service stores.

The `kp` CLI can be used to manage clusterstores. See the `kpack-cli` [help text](#) on GitHub.

```
$ kp clusterstore
ClusterStore Commands

Usage:
  kp clusterstore [command]

Aliases:
  clusterstore, clusterstores, clstrcsrs, clstrcsr, csrs, csr

Available Commands:
  add          Add buildpackage(s) to cluster store
  create       Create a cluster store
  delete       Delete a cluster store
  list         List cluster stores
  remove       Remove buildpackage(s) from cluster store
  save         Create or update a cluster store
  status       Display cluster store status

Flags:
  -h, --help  help for clusterstore
```

Note: These docs assume `kp cli v0.6.*` from Tanzu Build Service release `v1.6.*`. If a feature is not working, you may need to upgrade your cli.

Creating buildpacks and buildpackages

For information about creating buildpacks, see [Create a buildpack](#) on the Buildpacks.io website.

For information about packaging buildpacks, see [Package a buildpack](#) on the Buildpacks.io website.

Note: Only Build Service admins can perform store commands.

Listing ClusterStores

Users can view the existing stores by running:

```
kp clusterstore list
```

Creating a ClusterStore

Tanzu Build Service ships with a `default` store containing all of the supported buildpacks. Users can create additional stores with:

```
kp clusterstore create <store-name> -b <buildpackage-1> -b <buildpackage-2>
```

For example:

```
kp clusterstore create my-store -b my-registry.com/my-buildpackage
kp clusterstore create my-store -b my-registry.com/my-buildpackage -b my-registry.com/
my-other-buildpackage
kp clusterstore create my-store -b ../path/to/my-local-buildpackage.cnb
```

Buildpackages are uploaded to the registry used during installation.

Note: The user must have read access to the source Docker registry and write access to the registry used for installation on the local machine.

Saving a ClusterStore

Users can create or update a ClusterStore using the `save` command. The `kp clusterstore save` command is used exactly the same as `kp clusterstore create`, but it determines whether a clusterstore needs to be created or updated.

```
kp clusterstore save <store-name> -b <buildpackage-1> -b <buildpackage-2>
```

Adding buildpackages to a ClusterStore

Users can add multiple buildpackages at a time from a registry or from a file on the local machine.

This command is useful for users that want to only consume certain buildpacks rather than update all dependencies with `kp import`.

- If using a Docker registry:

```
kp clusterstore add <store-name> -b <buildpackage-1> -b <buildpackage-2> ...
```

Note: The user must have read access to the source Docker registry and write access to the registry used for installation on the local machine.

- If using local `.cnb` buildpackage files created as described in the [buildpackages docs](#):

```
kp clusterstore add <store-name> -b <path-to-buildpackage-1>.cnb -b <path-to-buildpackage-2>.cnb ...
```

Adding buildpackages to a ClusterStore from Tanzu Network

Updated versions of all supported Buildpacks are available on [Tanzu Network](#) as registry images. Updated Buildpacks are found in the following locations:

- [Java](#)
- [NodeJS](#)
- [Go](#)
- [PHP, .NET Core, nginx, httpd, procfile](#)

Here is a list of how to update each buildpack included with Tanzu Build Service by default:

```
kp clusterstore add default registry.tanzu.vmware.com/tanzu-java-buildpack/java:<version>
kp clusterstore add default registry.tanzu.vmware.com/tanzu-nodejs-buildpack/nodejs:<version>
kp clusterstore add default registry.tanzu.vmware.com/tanzu-go-buildpack/go:<version>
kp clusterstore add default registry.tanzu.vmware.com/tbs-dependencies/paketo-buildpacks_dotnet-core:<version>
kp clusterstore add default registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildpacks_php:<version>
kp clusterstore add default registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildpacks_nginx:<version>
kp clusterstore add default registry.tanzu.vmware.com/tbs-dependencies/tanzu-buildpacks_httpd:<version>
kp clusterstore add default registry.tanzu.vmware.com/tbs-dependencies/paketo-buildpacks_procfile:<version>
```

Offline Adding Buildpackages to a ClusterStore from Tanzu Network

If your Tanzu Build Service installation is in an offline/air-gapped environment, you can update stores with the following offline workflow:

1. Find the latest version of the Dependency Descriptor bundle image ([registry.tanzu.vmware.com/tbs-dependencies/full](#)) from the latest release on the [Tanzu Build Service Dependencies](#) page on Tanzu Network.
2. Download the following CLIs for your operating system:
 - ◆ [kp](#).
 - ◆ [imgpkg](#)
 - ◆ [kblid](#)
3. Log in to the Tanzu Network registry with your Tanzu Network credentials:

- ◆ If you are using the [docker](#) CLI, run:

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

- ◆ If you are using the [crane](#) CLI, run:

```
crane auth login registry.tanzu.vmware.com
```

- Download the dependency images for Tanzu Build Service to your local machine with `imgpkg` by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tbs-dependencies/full:<VERSION> \
  --to-tar=tbs-dependencies.tar
```

- Move the output file `tbs-dependencies.tar` to a machine that has access to the "offline" environment.
- Log in to the image registry used to deploy Tanzu Build Service:

- If you are using the `docker` CLI, run:

```
docker login <build-service-registry>
```

- If you are using the `crane` CLI, run:

```
crane auth login <build-service-registry>
```

- Upload the dependency images to the registry used to deploy Tanzu Build Service by running:

```
imgpkg copy --tar=tbs-dependencies.tar \
  --to-repo <IMAGE-REPOSITORY>
```

Where:

- `IMAGE-REPOSITORY` is the repository used to install Tanzu Build Service. This should be the same value as `IMAGE-REPOSITORY` used in the [Installation Steps](#).

- Now that dependencies are relocated to the internal registry, you can use the following commands to update the necessary resources:

```
imgpkg pull -b <IMAGE-REPOSITORY>:<VERSION> \
  -o /tmp/descriptor-bundle \
  --registry-ca-cert-path <PATH-TO-CA>

kblid -f /tmp/descriptor-bundle/.imgpkg/images.yml \
  -f /tmp/descriptor-bundle/tanzu.descriptor.v1alpha3/descriptor-<VERSION>.yaml \
  | kp import -f -
```

Removing buildpackages from a ClusterStore

Users can remove a buildpackage from a ClusterStore by referencing the buildpackage Id and version.

```
kp clusterstore remove <store> -b <buildpackage-id>@<buildpackage-version>
```

For example:

```
kp clusterstore remove my-store -b buildpackage@1.0.0
kp clusterstore remove my-store -b buildpackage@1.0.0 -b other-buildpackage@2.0.0
```


The ClusterStore status shows the list of buildpackage ID and version

Get ClusterStore status

Users can use the `kp` CLI to get details about a store including buildpackages and their buildpacks, as well as meta-buildpacks. [Meta-buildpacks](#) are buildpacks that indicate the order that other buildpacks run:

To view the buildpackages in a store, run:

```
kp clusterstore status <store-name>
```

For example:

```
$ kp clusterstore status default

Status:    Ready

BUILDPACKAGE ID          VERSION    HOMEPAGE
paketo-buildpacks/go     0.1.3     https://github.com/paketo-buildpacks/
go
paketo-buildpacks/procfile 2.0.2     https://github.com/paketo-buildpacks/
procfile
paketo-buildpacks/procfile 3.0.0     https://github.com/paketo-buildpacks/
procfile
tanzu-buildpacks/dotnet-core 0.0.4
tanzu-buildpacks/dotnet-core 0.0.7
tanzu-buildpacks/dotnet-core 0.0.6
tanzu-buildpacks/go      1.0.6
tanzu-buildpacks/go      1.0.7
tanzu-buildpacks/go      1.0.9
tanzu-buildpacks/go      1.0.5
tanzu-buildpacks/httpd   0.0.38
tanzu-buildpacks/httpd   0.0.39
tanzu-buildpacks/httpd   0.0.40
tanzu-buildpacks/java    3.8.0     https://github.com/pivotal-cf/tanzu-j
ava
tanzu-buildpacks/java    3.5.0     https://github.com/pivotal-cf/tanzu-j
ava
tanzu-buildpacks/java    4.1.0     https://github.com/pivotal-cf/tanzu-j
ava
tanzu-buildpacks/java    4.0.0     https://github.com/pivotal-cf/tanzu-j
ava
tanzu-buildpacks/java-native-image 3.6.0     https://github.com/pivotal-cf/tanzu-j
ava-native-image
tanzu-buildpacks/java-native-image 3.9.0     https://github.com/pivotal-cf/tanzu-j
ava-native-image
tanzu-buildpacks/java-native-image 3.4.2     https://github.com/pivotal-cf/tanzu-j
ava-native-image
tanzu-buildpacks/java-native-image 3.10.0    https://github.com/pivotal-cf/tanzu-j
ava-native-image
tanzu-buildpacks/nginx   0.0.48
tanzu-buildpacks/nginx   0.0.46
tanzu-buildpacks/nodejs  1.1.0
tanzu-buildpacks/nodejs  1.2.3
tanzu-buildpacks/nodejs  1.2.2
tanzu-buildpacks/php     0.0.3
```

```
tanzu-buildpacks/php
```

```
0.0.5
```

To view buildpackages & their individual buildpacks, and display the order of meta-buildpacks, use the `--verbose` flag:

```
kp clusterstore status <store-name> --verbose
```

Migrating buildpacks

Build Service never automatically removea buildpackages from the store unless you explicitly remove them. In this way, users can continue to use older buildpacks until the operator is ready to migrate them.

How you migrate is entirely dependent on the configuration of your Builder resources:

- Builders that do not provide a buildpack version automatically update to the latest buildpack version if it is available.
- Builders that explicitly specify a buildpack version do not update automatically.

With the above in mind, migrating buildpackages in the store is as simple as `kp clusterstore adding` newer buildpackages and `kp clusterstore removeing` older buildpackages as necessary.

For fine-grained control over buildpack updates, you can create multiple stores to manage buildpack versions. Then you can point individual builders at the desired store. Each store can be updated as needed without affecting other builders or fanning out large, sweeping changes.

Corresponding kpack resource

All Build Service builders utilize cluster-scoped [Store resources](#).

Descriptors

This topic describes the descriptors that are available so you can choose which option to configure depending on your use case.

About descriptors

Tanzu Build Service descriptors are curated sets of dependencies, including stacks and buildpacks, that are continuously released on VMware Tanzu Network to resolve all workload Critical and High CVEs. Descriptors are imported into Tanzu Build Service to update the entire cluster.

There are two types of descriptor, `lite` and `full`, available on the [Tanzu Network Build Service Dependencies](#) page. The different descriptors can apply to different use cases and workload types. For the differences between the descriptors, see [Descriptor comparison](#).

You configure which descriptor is imported when installing Tanzu Build Service.

Lite descriptor

The Tanzu Build Service `lite` descriptor is the default descriptor selected if none is configured.

It contains a smaller footprint to speed up installation time. However, it does not support all workload types. For example, the `lite` descriptor does not contain the PHP buildpack.

The `lite` descriptor only contains the `base` stack. The `default` stack is installed, but is identical to the `base` stack. For more information, see [Stacks](#).

Full descriptor

The Tanzu Build Service `full` descriptor contains more dependencies, which allows for more workload types.

The dependencies are pre-packaged so builds don't have to download them from the Internet. This can speed up build times and allows builds to occur in airgapped environments.

The `full` descriptor contains the following stacks, which support different use cases:

- `base`
- `default` (identical to `base`)
- `full`
- `tiny`

For more information, see [Stacks](#). Due to the larger footprint of `full`, installations might take longer.

Descriptor comparison

Both `lite` and `full` descriptors are suitable for production environments.

	lite	full
Faster installation time	Yes	No
Dependencies pre-packaged	No	Yes
Contains base stack	Yes	Yes
Contains full stack	No	Yes
Contains tiny stack	No	Yes
Supports Java workloads	Yes	Yes
Supports Node.js workloads	Yes	Yes
Supports Go workloads	Yes	Yes
Supports Python workloads	Yes	Yes
Supports .NET Core workloads	Yes	Yes
Supports PHP workloads	No	Yes
Supports static workloads	Yes	Yes
Supports binary workloads	Yes	Yes

Managing Builders

A Builder is a Tanzu Build Service resource used to manage [Cloud Native Buildpack builders](#).

Builders contain a set of buildpacks and a stack used to create images.

There are two types of Builders:

- Cluster Builders: Cluster-scoped Builders
- Builders: Namespace-scoped Builders

Note: Only Build Service Admins can manage Cluster Builders.

The `kp` CLI can be used to manage builders and clusterbuilders. The help text is published in the [kpack-cli documentation](#) on GitHub.

```
$ kp builder
Builder Commands

Usage:
  kp builder [command]

Aliases:
  builder, builders, bldrs, bldr

Available Commands:
  create      Create a builder
  delete      Delete a builder
  list        List available builders
  patch       Patch an existing builder configuration
  save        Create or patch a builder
  status      Display status of a builder

Flags:
  -h, --help  help for builder

Use "kp builder [command] --help" for more information about a command.
```

```
$ kp clusterbuilder
ClusterBuilder Commands

Usage:
  kp clusterbuilder [command]

Aliases:
  clusterbuilder, clusterbuilders, clstrbldrs, clstrbldr, cbldrs, cblldr, cbs, cb

Available Commands:
  create      Create a cluster builder
  delete      Delete a cluster builder
```

```
list      List available cluster builders
patch    Patch an existing cluster builder configuration
save     Create or patch a cluster builder
status   Display cluster builder status
```

Flags:

```
-h, --help  help for clusterbuilder
```

Note: These docs assume kp cli v0.6.* from TBS release v1.6.*. If a feature is not working, you might need to upgrade your cli.

Creating a Builder

Use the `kp` cli to create a Builder:

- Cluster Builder:

```
kp clusterbuilder create <name> --tag <tag> --order <order> --stack <stack> --store <store>
```

```
kp clusterbuilder create <name> --tag <tag> --stack <stack> --store <store> --buildpack <buildpack>
```

- Builder:

```
kp builder create <name> --tag <tag> --order <order> --stack <stack> --store <store> --namespace <namespace>
```

```
kp builder create <name> --tag <tag> --stack <stack> --store <store> --namespace <namespace> --buildpack <buildpack>
```

Where:

- `name`: The name of the builder.
- `tag`: The registry location where the builder is created.
- `stack`: The name of the stack used by the builder.
- `store`: The name of the store containing the buildpacks used by the builder.
- `namespace`: The kubernetes namespace for the builder (for Builders only).
- `order`: The local path to the buildpack order YAML that the builder uses. Sample order YAML files are available on the [VMware Tanzu Build Service Dependencies](#) page on Tanzu Network. For more information about listing buildpacks in groups in the order YAML, see [builder.toml](#) in the Buildpacks.io documentation.

Example order YAML file that used by a builder designed to build NodeJS and Java apps:

```
- group:
  - id: tanzu-buildpacks/nodejs
- group:
  - id: tanzu-buildpacks/java
```

- `buildpack`: Buildpack id and optional version in the form of either '@' or '. Repeat for each

buildpack in order, or supply once with comma-separated list. This cannot be combined with `--order`. All supplied buildpacks are in the same group.

Patching a builder

You can update a builder resource using the `kp` cli. To update a builder given a `name`, run:

- Cluster Builder:

```
kp clusterbuilder patch <name> --order <order> --stack <stack> --store <store>
```

```
kp clusterbuilder patch <name> --stack <stack> --store <store> --buildpack <buildpack>
```

- Builder:

```
kp builder patch <name> --order <order> --stack <stack> --store <store> --namespace <namespace>
```

```
kp builder patch <name> --stack <stack> --store <store> --namespace <namespace> --buildpack <buildpack>
```

`kp ccb patch` and `kp cb patch` are respective aliases.

Where:

- `name`: The name of the builder.
- `stack`: The name of the stack used by the builder.
- `store`: The name of the store containing the buildpacks used by the builder.
- `namespace`: The Kubernetes namespace for the builder (for Builders only).
- `order`: The local path to the buildpack order YAML that the builder uses. Sample order YAML files are available on the [VMware Tanzu Build Service Dependencies](#) page on Tanzu Network. For more information about listing buildpacks in groups in the order YAML, see [builder.toml](#) in the Buildpacks.io documentation.

Example order YAML file used by a builder designed to build NodeJS and Java apps:

```
---
- group:
  - id: paketo-buildpacks/bellsoft-liberica
  - id: paketo-buildpacks/gradle
- group:
  - id: paketo-buildpacks/nodejs
```

- `buildpack`: Buildpack ID and optional version in the form of either '@' or '. Repeat for each buildpack in order, or supply once with comma-separated list. This cannot be combined with `--order`. All supplied buildpacks are in the same group.

Note: The `tag` (location in a registry) of a builder cannot be modified. To change this field, you must create a new builder.

Saving Builders

Users can create or update a Builder/ClusterBuilder by using the `save` command. The `kp builder/clusterbuilder save` command is used exactly the same as `kp builder/clusterbuilder create` and `kp builder/clusterbuilder update`, but it determines whether a builder/clusterbuilder needs to be created or updated.

To save a Builder/ClusterBuilder:

- Cluster Builder:

```
kp clusterbuilder save <name> --tag <tag> --order <order> --stack <stack> --store <store>
```

```
kp clusterbuilder save <name> --tag <tag> --stack <stack> --store <store> --buildpack <buildpack>
```

- Builder:

```
kp builder save <name> --tag <tag> --order <order> --stack <stack> --store <store> --namespace <namespace>
```

```
kp builder save <name> --tag <tag> --stack <stack> --store <store> --namespace <namespace> --buildpack <buildpack>
```

Where:

- `name`: The name of the builder.
- `tag`: The registry location where the builder is created.
- `stack`: The name of the stack used by the builder.
- `store`: The name of the store containing the buildpacks used by the builder.
- `namespace`: The Kubernetes namespace for the builder (for Builders only).
- `order`: The local path to the buildpack order YAML that the builder uses. Sample order YAML files are available on the [VMware Tanzu Build Service Dependencies](#) page on Tanzu Network. For more information about listing buildpacks in groups in the order YAML, see [builder.toml](#) in the Buildpacks.io documentation.

Example order YAML file used by a builder designed to build NodeJS and Java apps:

```
---
- group:
  - id: paketo-buildpacks/bellsoft-liberica
  - id: paketo-buildpacks/gradle
- group:
  - id: paketo-buildpacks/nodejs
```

- `buildpack`: Buildpack ID and optional version in the form of either '@' or ". Repeat for each buildpack in order, or supply once with comma-separated list. This cannot be combined with `--order`. All supplied buildpacks are in the same group.

Deleting Builders

To delete a builder:

- Cluster Builder:

```
kp clusterbuilder delete <builder name>
```

- Builder:

```
kp builder delete <builder name> --namespace <namespace>
```

Warning: Deleting a builder prevents image configs that reference that builder from successfully building again.

Retrieving Builder Details

To get builder details:

- Cluster Builder:

```
kp clusterbuilder status <builder-name>
```

- Builder:

```
kp builder status <builder-name> --namespace <namespace>
```

Example:

```
$ kp clusterbuilder status tiny

Status:      Ready
Image:       gcr.io/my-repo/tiny@sha256:07d94db2e3e9f43cba67c389f1c83e4eac821aa83084a
88136ed8d431b37f008
Stack:       io.paketo.stacks.tiny
Run Image:   gcr.io/cf-build-service-dev-219913/ssuresh/install/run@sha256:e9159f0ef2
3c28b943cfb1b5d5be9638b67211f6ff0bd3fae35ff4b499136152

BUILDPACK ID          VERSION    HOMEPAGE
paketo-buildpacks/graalvm 4.0.0     https://github.com/paketo-bui
ldpacks/graalvm
tanzu-buildpacks/go-dist 0.1.3
paketo-buildpacks/gradle 3.5.0     https://github.com/paketo-bui
ldpacks/gradle
paketo-buildpacks/sbt    3.6.0     https://github.com/paketo-bui
ldpacks/sbt
paketo-buildpacks/maven  3.2.1     https://github.com/paketo-bui
ldpacks/maven
tanzu-buildpacks/dep     0.0.10
paketo-buildpacks/spring-boot 3.5.0     https://github.com/paketo-bui
ldpacks/spring-boot
paketo-buildpacks/leiningen 1.2.1     https://github.com/paketo-bui
ldpacks/leiningen
paketo-buildpacks/spring-boot-native-image 2.0.0     https://github.com/paketo-bui
ldpacks/spring-boot-native-image
```

```

paketo-buildpacks/executable-jar 3.1.3 https://github.com/paketo-bui
ldpacks/executable-jar
tanzu-buildpacks/go-build 0.0.23
paketo-buildpacks/environment-variables 2.1.2 https://github.com/paketo-bui
ldpacks/environment-variables
paketo-buildpacks/procfile 3.0.0 https://github.com/paketo-bui
ldpacks/procfile
paketo-buildpacks/image-labels 2.0.6 https://github.com/paketo-bui
ldpacks/image-labels
tanzu-buildpacks/dep-ensure 0.0.29
tanzu-buildpacks/go-mod-vendor 0.0.26
tanzu-buildpacks/java-native-image 3.10.0 https://github.com/pivotal-cf
/tanzu-java-native-image
tanzu-buildpacks/go 1.0.9

DETECTION ORDER
Group #1
  tanzu-buildpacks/go@1.0.9
Group #2
  tanzu-buildpacks/java-native-image@3.10.0
Group #3
  paketo-buildpacks/procfile@3.0.0

```

Listing Builders

To list all builders available to the current user:

- Cluster Builder:

```
kp clusterbuilder list
```

- Builder:

```
kp builder list --namespace <namespace>
```

Corresponding kpack Resources

All Build Service Builders are represented as kpack resources.

- [Builder](#)
- [ClusterBuilder](#)

Pinning Buildpack versions

You can pin buildpack versions by specifying the version for buildpacks in the order file.

As an **example**, consider the clusterbuilder created below:

```

kp cb create pinned \
  --tag my-registry.io/example/pinned \
  --order order.yaml

```

Where the contents of order.yaml file is:

```
- group:  
  - id: tanzu-buildpacks/php  
    version: 0.0.5  
- group:  
  - id: tanzu-buildpacks/nodejs  
    version: 1.3.0
```

Note: When a buildpack version is pinned, Images that use the Builder do not initiate new Builds due to new Buildpack versions. As a best practice, only pin a buildpack version when necessary.

Update Lifecycle

All builders make use of a lifecycle. A lifecycle orchestrates buildpack execution, then assembles the resulting artifacts into a final app image. Within Build Service, it is uploaded to the canonical registry, which is the docker-repository specified during Tanzu Build Service install. More information on lifecycles can be found in [Lifecycle](#) on the Buildpacks.io website.

To update the lifecycle to be used by builders:

```
```\nkp lifecycle update --image <image-tag>\n```\n
```

**Note:** You must have credentials to access the registry on your machine.

# Managing Custom Stacks

A CustomStack is a resource that allows users to create a customized [ClusterStack](#) from Ubuntu 18.04 (Bionic Beaver) and UBI7/UBI8 non-minimal-based OCI images.

**Note:** CustomStacks created with UBI7/UBI8 images do not currently support adding packages, mixins, or caCerts. Currently, only the Java Tanzu Buildpacks support CustomStacks that use UBI images.

**Note:** Private registries which require a custom CA certificate are not currently supported as publishing destinations for CustomStack images.

CustomStacks can be used to:

- Convert a pre-existing base image that you'd like to use with Tanzu Build Service into a ClusterStack resource.
- Add required stack metadata to base images.
- Add CA certificates to build and/or run image.
- Add packages and [mixin labels](#) to build and/or run image.
- Set CNB user and group IDs.

## Creating a CustomStack

**Note:** CustomStacks make changes to a base image that require root permissions such as adding OS packages, setting the user or group id, or adding CA certificates to the OS trusted store. This means that CustomStack pods must run as privileged. However, TBS does NOT require any privileged containers to install or run – only when using CustomStacks is a privileged container ever created.

A CustomStack is created by running `kubectl apply` with a resource configuration file. The following defines the relevant fields of the CustomStack resource spec in more detail:

- `source`: The location of base images used for building the stack. See more info in [Source Configuration](#).
- `destination`: The location to publish built images and optional ClusterStack. See more info in [Destination Configuration](#).
- `caCerts`: References to config maps of CA certificates to add to one or both of the stack images.
- `packages`: List of packages to install on one or both of the stack images. A list of all available packages can be found in [Software Packages in "bionic"](#) on the ubuntu packages web page.
- `mixins`: List of mixin labels to add to one or both of the stack images. Information on the mixins concept can be found in [Stack](#) on the Buildpacks.io website.

- `service-account-name`: Name of service account with secret containing credentials to push to registry.
- `user`: User and group ID of the CNB user.
  - ◊ Not required if the user is already present in metadata.
  - ◊ If the user and/or group ID do not exist on the image, they are created.

## Source configuration

The `source` field describes the base images for the CustomStack. It can be configured in exactly one of the following ways:

- Registry Images

```
source:
 registryImages:
 build:
 image: <build-base-image>
 run:
 image: <run-base-image>
```

- ◊ `build-base-image`: The fully qualified reference of the build base image.
- ◊ `run-base-image`: The fully qualified reference of the run base image.

- Stack

```
stack:
 name: <cluster-stack-name>
 kind: ClusterStack
```

- ◊ `cluster-stack-name`: Name of ClusterStack to base CustomStack images on.

## Destination Configuration

The `destination` field describes where the built images are published and whether a ClusterStack should be created.

```
destination:
 build:
 tag: <output-build-image-tag>
 run:
 tag: <output-run-image-tag>
 stack: # Optional
 name: <output-cluster-stack-name>
 kind: ClusterStack
```

- `output-build-image-tag`: The registry location where the build image is created.
- `output-run-image-tag`: The registry location where the run image is created.
- `output-cluster-stack-name`: Name of ClusterStack to create with CustomStack images.

## Example CustomStack from registry images

```

apiVersion: v1
kind: ConfigMap
metadata:
 name: build-ca-certs
data:
 cert-1: |
 -----BEGIN CERTIFICATE-----
 ...
 -----END CERTIFICATE-----
 cert-2: |
 -----BEGIN CERTIFICATE-----
 ...
 -----END CERTIFICATE-----

apiVersion: v1
kind: ConfigMap
metadata:
 name: run-ca-certs
data:
 cert-3: |
 -----BEGIN CERTIFICATE-----
 ...
 -----END CERTIFICATE-----

apiVersion: stacks.stacks-operator.tanzu.vmware.com/v1alpha1
kind: CustomStack
metadata:
 name: stack-sample
spec:
 source:
 registryImages:
 build:
 image: paketobuildpacks/build@sha256:ae88191cc5bfd0dcd2938954f20d5df5060a562af
6e3d65a92a815612054537c
 run:
 image: paketobuildpacks/run@sha256:48f67dcb3f2b27403de80193e34abd3172b3fbdfdd8
7e452721aba90ea68fc66
 destination:
 build:
 tag: my.registry.io/final-build-image
 run:
 tag: my.registry.io/final-run-image
 stack: # Optional
 name: stack-sample-cluster-stack
 kind: ClusterStack
 caCerts: # Optional
 buildRef: # Optional
 name: build-ca-certs
 runRef: # Optional
 name: run-ca-certs
 packages: # Optional
 - name: cowsay
 - name: cowsay-off
 - name: fortune
 phase: build
 - name: rolldice
 phase: run
 mixins: # Optional
 - name: set=build-utils

```

```

 phase: build
 - name: set=run-utils
 phase: run
 - name: set=shared-utils
serviceAccountName: default
user: # Optional
 userID: 1000
 groupID: 1000

```

## Example CustomStack from ClusterStack

```

apiVersion: v1
kind: ConfigMap
metadata:
 name: build-ca-certs
data:
 cert-1: |
 -----BEGIN CERTIFICATE-----
 ...
 -----END CERTIFICATE-----
 cert-2: |
 -----BEGIN CERTIFICATE-----
 ...
 -----END CERTIFICATE-----

apiVersion: v1
kind: ConfigMap
metadata:
 name: run-ca-certs
data:
 cert-3: |
 -----BEGIN CERTIFICATE-----
 ...
 -----END CERTIFICATE-----

apiVersion: stacks.stacks-operator.tanzu.vmware.com/v1alpha1
kind: CustomStack
metadata:
 name: stack-sample
spec:
 source:
 stack:
 name: stack-sample-cluster-stack
 kind: ClusterStack
 destination:
 build:
 tag: my.registry.io/final-build-image
 run:
 tag: my.registry.io/final-run-image
 stack: # Optional
 name: final-stack-sample-cluster-stack
 kind: ClusterStack
 caCerts: # Optional
 buildRef: # Optional
 name: build-ca-certs
 runRef: # Optional
 name: run-ca-certs
 packages: # Optional

```

```

- name: cowsay
- name: cowsay-off
- name: fortune
 phase: build # Optional
- name: rolldice
 phase: run # Optional
mixins: # Optional
- name: set=build-utils
 phase: build # Optional
- name: set=run-utils
 phase: run # Optional
- name: set=shared-utils
serviceAccountName: default
user: # Optional
 userID: 1000 # Optional
 groupID: 1000 # Optional

```

## Debugging CustomStacks

When a CustomStack is created, a pod is created in the same namespace which modifies the base image and pushes the resulting stack image to the registry. The pod is named `stack-pod-  
<customstack-name>-<number>`

Where:

- `customstack-name` is the name of your CustomStack.
- `number` is the revision of your CustomStack. This is incremented by one each time a new spec is applied.

The ten latest pods are kept around for debugging purposes. To debug a failing CustomStack, check the logs of the corresponding pod: `kubectl logs <pod-name> -c <create-build-image/create-run-image>`

Where:

- `pod-name` is the name of the pod.
- `create-build-image/create-run-image` is the container whose logs you want to see.
  - ◊ `create-build-image` for logs related to creating the build image.
  - ◊ `create-run-image` for logs related to create the run image.



# RBAC in Tanzu Build Service

Given that Tanzu Build Service supports functionality most customers would likely want to restrict to only certain users, we encourage utilization of role-based access control (RBAC) as a best practice if Tanzu Build Service is to be broadly deployed to many users.

## RBAC support in Tanzu Build Service

Tanzu Build Service is installed with 2 Kubernetes [ClusterRoles](#) that can be used for RBAC for Build Service users and admins:

- `build-service-user-role`
- `build-service-admin-role`

### Build Service user role

This should be used for users who create images and builds.

To view the configuration for this role, run:

```
kubectl get clusterrole build-service-user-role -o yaml
```

To use this ClusterRole, create a [RoleBinding](#) with an existing user.

For example:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: my-build-service-user-role-binding
 namespace: my-build-namespace
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: build-service-user-role
subjects:
- kind: User
 name: my-user
```

### Build Service admin role

Use this role for admin users who operate Tanzu Build Service.

To view the configuration for this role, run:

```
kubectl get clusterrole build-service-admin-role -o yaml
```

---

To use this ClusterRole, create a [RoleBinding](#) or [ClusterRoleBinding](#) with an existing user.

For example:

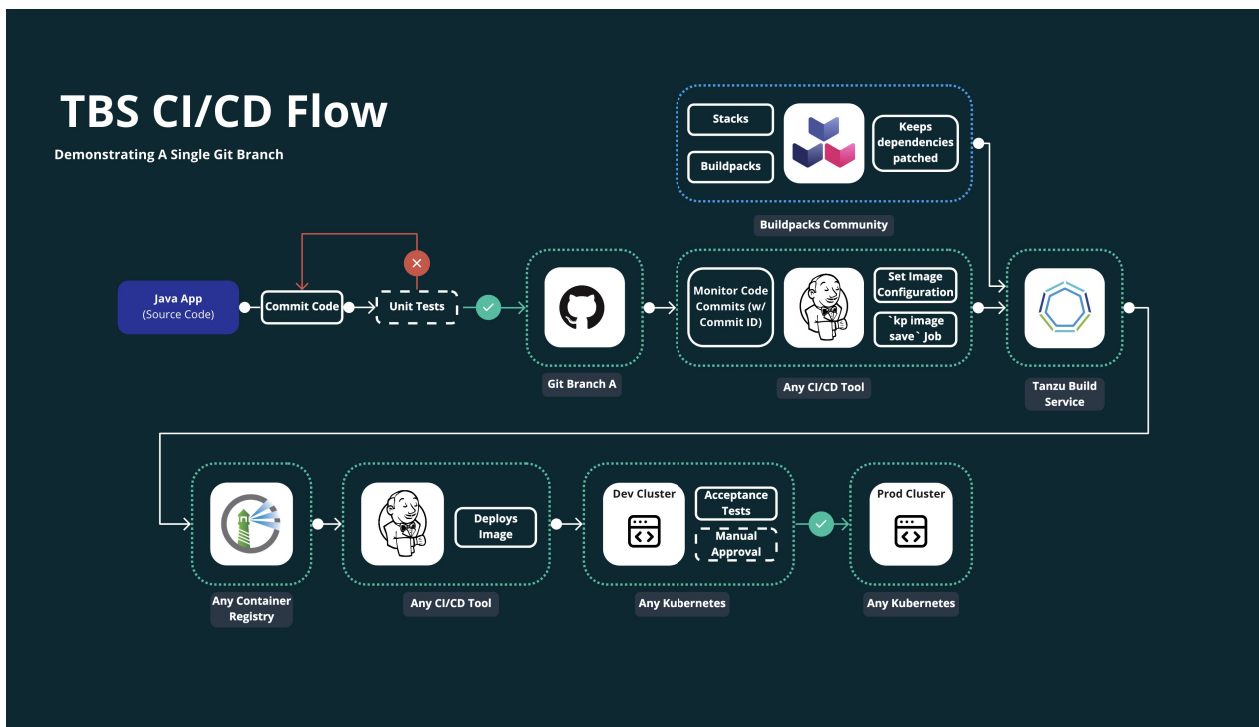
```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: my-build-service-admin-role-binding
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: build-service-admin-role
subjects:
- kind: User
 name: my-cluster-wide-admin-user
```

# Using Tanzu Build Service in CI

This topic describes how to best leverage Tanzu Build Service in a continuous integration context to build applications and keep them up to date at scale.

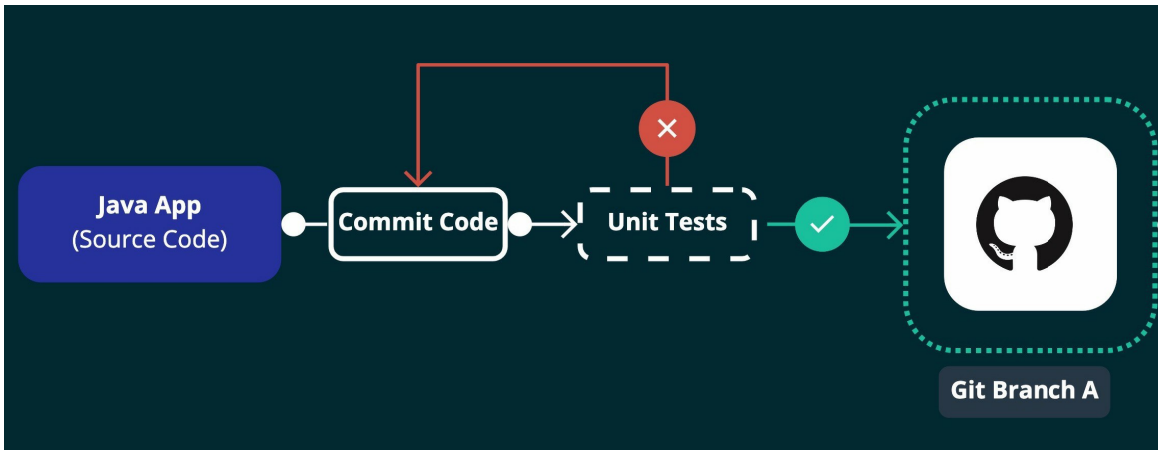
## Using Tanzu Build Service in CI/CD



This example shows using an image resource with Git source in a development-to-production continuous integration/continuous delivery (CI/CD) pipeline flow.

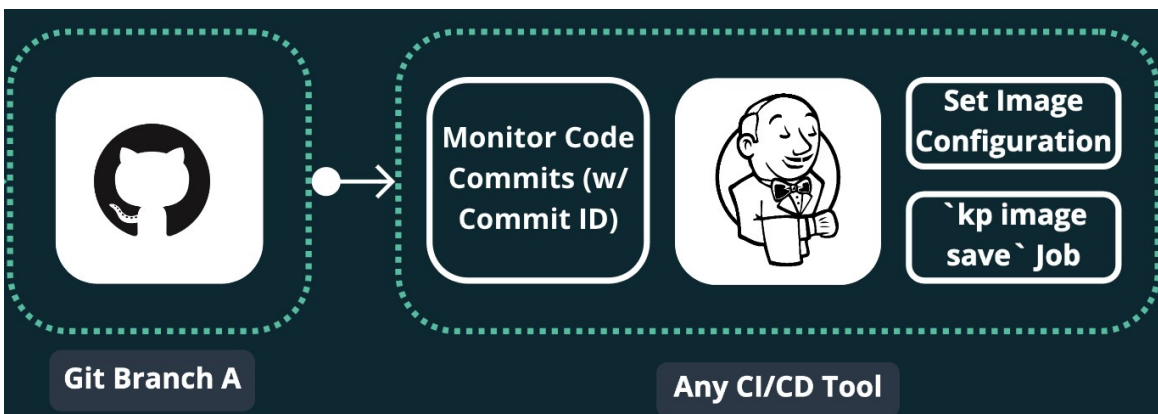
Let's split this up into each step.

1. Run unit tests & merge to branch:



This step shows a typical initial unit testing CI flow.

1. Developer pushes code to feature branch.
  2. CI/CD runs unit tests on that branch.
  3. Once tests have passed, the feature branch is merged to release branch (Git Branch A).
2. Update Tanzu Build Service Image Configuration in CI/CD:



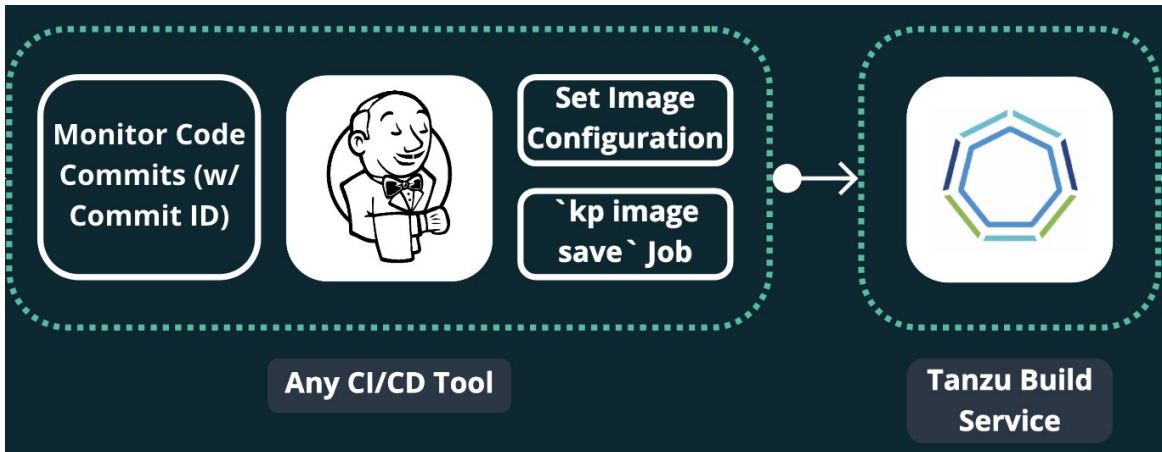
After unit tests pass, CI/CD must tell Tanzu Build Service to build the registry image using the Git commit that passed tests.

For example:

Jenkins job that runs the following after unit tests with the successful `<git-commit>`:

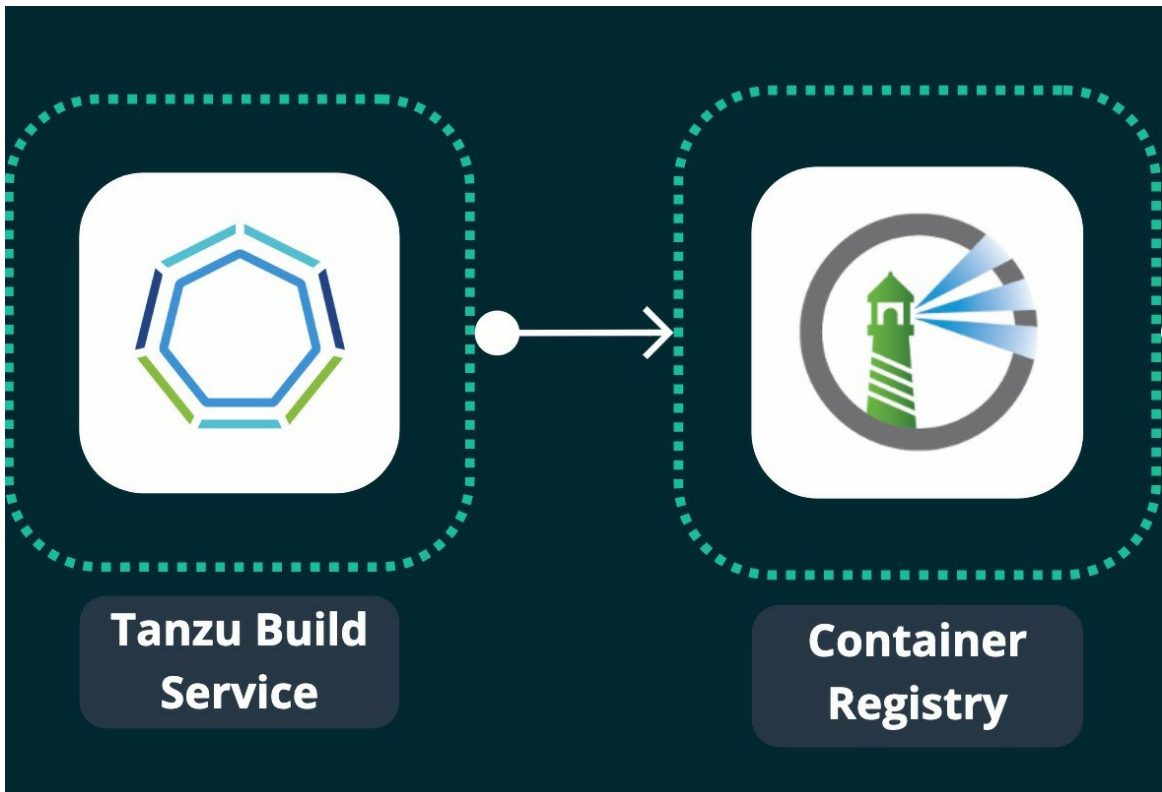
```
kp image save my-image --git-revision <git-commit>
```

3. Tanzu Build Service builds the OCI registry image using the Git commit:



Here Tanzu Build Service works its magic and builds a new registry image using the Git commit set in the previous step and the latest app dependencies (stacks and buildpacks).

4. Tanzu Build Service pushes the built image to your registry:



After the build finishes, Tanzu Build Service writes the resulting image to a container registry such as Harbor.

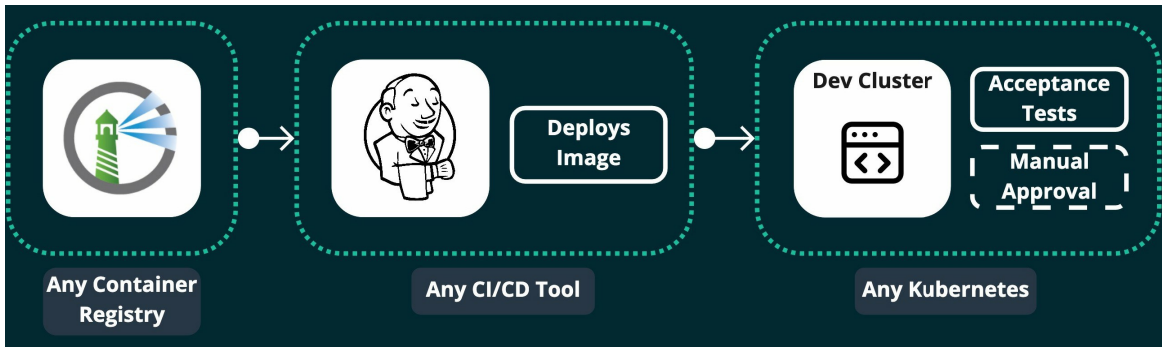
This image reference can be found with:

```
kp image status <image-name>
```

or

```
kp build status <image-name> -b <build-number>
```

5. Using CI/CD, deploy the built image to a Dev/QA Kubernetes cluster:

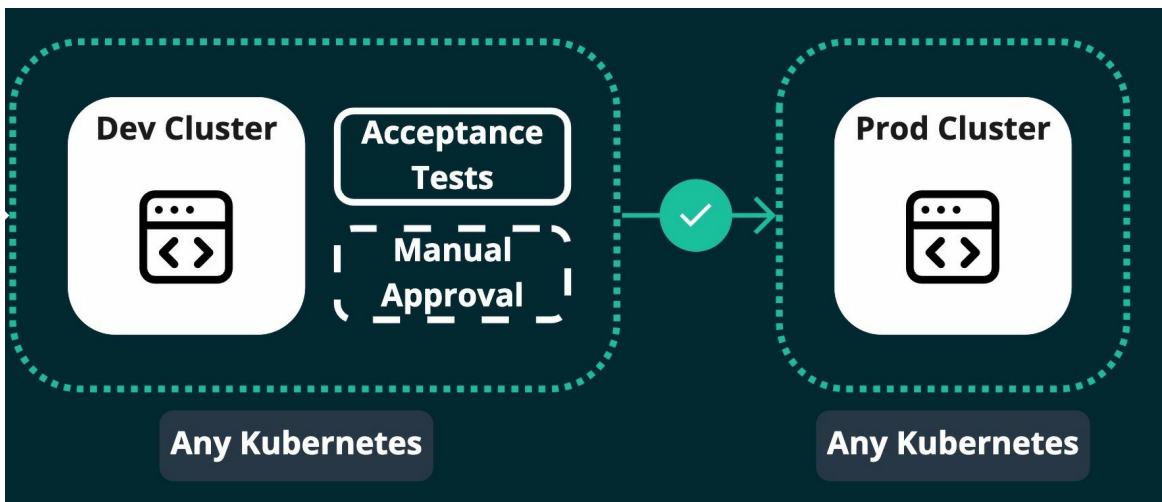


Now that the image is available in your registry, it can be deployed to any Kubernetes cluster. In this example, it is deployed to a Dev cluster for acceptance testing and QA/manual approval.

There are a couple of ways to trigger this job:

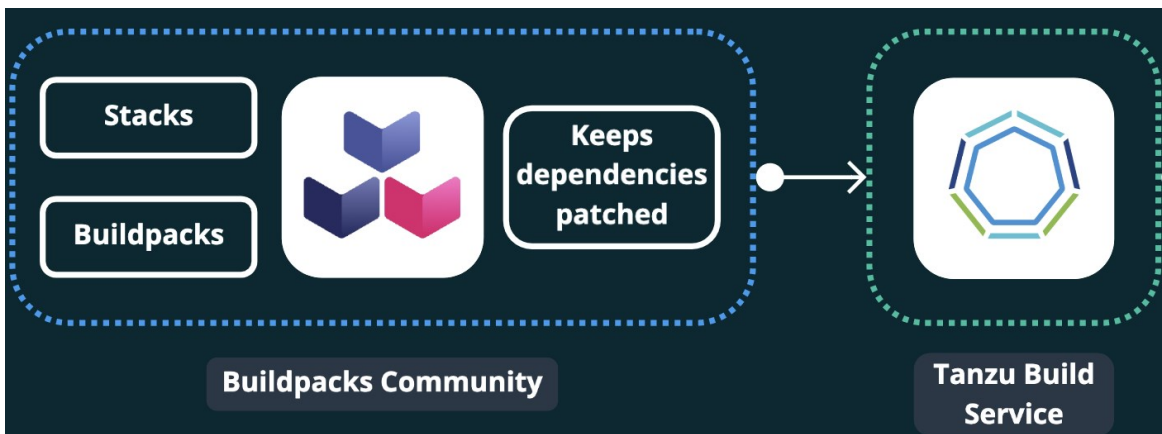
- ◊ Using registry webhooks (such as Harbor's) to trigger a CI/CD job.
- ◊ If you are using Concourse CI: the [Concourse kpack Resource](#).
- ◊ Write your own polling mechanism to check for new images in your registry.

6. Once the app has been vetted, deploy to production:



The same way the image was deployed to the Dev cluster, the image can be pushed to production.

7. Bonus: dependencies are kept up to date for secure app images:



Images are kept up to date with the latest dependencies provided through stacks and buildpacks from the Cloud Native Buildpacks community, which are released for Tanzu Build Service as Tanzu Buildpacks on Tanzu Network.

As of Tanzu Build Service 1.2, these dependencies are automatically updated. You can also update these dependencies with the `kp` cli in CI/CD by running:

```
kp import -f descriptor.yaml
```

When dependencies are updated, affected apps are rebuilt to be promoted by using steps 5 and 6.





# Frequently Asked Questions

## How do Cloud Native Buildpacks (CNBs), kpack, and Tanzu Build Service overlap and differ?

**CNBs** are build tools that adhere to the [CNB v3 Specification](#) and transform source code into an OCI compliant runnable image. The v3 specification, lifecycle, and local CLI ([pack](#)) are governed by the open source [Cloud Native Buildpacks project](#).

[kpack](#) is a collection of open source resource controllers that together function as a Kubernetes native build service. The product provides a declarative image type that builds an image and schedules image rebuilds when dependencies of the image change. kpack is a platform implementation of CNBs in that it utilizes CNBs and the v3 lifecycle to execute image builds.

**Tanzu Build Service** is a commercial product owned and operated by VMware that utilizes kpack and CNBs. Build Service provides additional abstractions intended to ease the use of the above technologies in Enterprise settings. These abstractions are covered in detail throughout the documentation on this site. Additionally, customers of Build Service are entitled to support and VMware Tanzu buildpacks.

---

## Why do I see two images in the image registry after a successful build?

By default Build Service will tag each built image twice. The first tag is the configured image tag. The second tag is a unique tag with the build number and build timestamp. The second tag is added to ensure that previous images are not deleted on registries that garbage collect untagged images.

---

## How does TBS work in air gapped environments?

Build Service is installed and deployed using [Carvel](#) tools. Therefore, the `imgpkg copy` command can create a `.tar` file composed of the Kubernetes config and images required to successfully install Build Service. The `imgpkg copy` command also ensures that all the images can be relocated to air-gapped registries. By providing the credentials to the air-gapped registry when executing the `kapp install` command, Build Service can then use that secret to pull images from the registry and so work in air-gapped environments.

Currently, `kbld package` and `kbld unpackage` must be used to import dependencies to an air-gapped environment.

For more details on air-gapped installation, see [Installation to Air-Gapped Environment](#).

For more details on air-gapped builds, see [Offline Builds](#).

---

## Is there documentation on supported Tanzu Buildpacks?

Yes, documentation is available on [Tanzu Buildpacks Documentation](#).

---

## Why do I get an X509 error from Build Service when trying to create an image in my registry?

When interacting with a registry or a Git repo that has been deployed using a self-signed certificate, Build Service must be provided with the certificate during install time. You will either need to target a registry that does not have self-signed certificates or re-install Build Service to work with this registry.

---

## How do I configure a secret to publish images to Dockerhub?

1. Create a dockerhub secret with the `kp` cli:

```
kp secret create my-dockerhub-creds --dockerhub DOCKERHUB-USERNAME
```

Where `DOCKERHUB-USERNAME` is your dockerhub username. You will be prompted for your dockerhub password.

---

## How can I configure an image resource to pull from a private GitHub repository?

1. Create a github secret with the `kp` cli:

Using a [git ssh key](#)

```
kp secret create my-git-ssh-cred --git git@github.com --git-ssh-key PATH-TO-GIT
HUB-PRIVATE-KEY
```

Where `PATH-TO-GITHUB-PRIVATE-KEY` is the absolute local path to the github ssh private key.

Or with a basic auth github username and password.

```
kp secret create my-git-cred --git https://github.com --git-user GITHUB-USERNA
ME
```

Where `GITHUB-USERNAME` is your github username. You will be prompted for your github password.

---

## Why do some builds fail with "Error: could not read run image: \*"?

The run image must be publicly readable or readable with the registry credentials configured in a project/namespace.

To see where the build service run image is located, run: `kp stack status STACK-NAME`.

If you cannot make the run image publicly readable, you must `kp` to create a registry secret within the namespace where your builds reside. This can be accomplished by using `kp secret create`.

## Why don't my image builds appear in my Harbor v1.X.X registry?

There is a known bug in Harbor that, at times, prevents the UI from showing images. If you are unable to see a recently built image in the Harbor UI, try pulling it using the `docker` or `crane` CLI to verify that it exists.

## How do I fix "unsupported status code 500" when creating a builder on my Harbor v2.X.X registry?

Some builders are very large and can overwhelm Harbor's default database connection. You can remediate this issue by increasing the `database.maxOpenConns` setting in the helm `values.yaml` file. Increase this value from 100 to 300. The exact setting can be found in the [harbor-helm values.yaml file](#) on GitHub.

## How do I configure credentials for using gcr as my installation registry?

You can use Google Container Registry for your Tanzu Build Service installation registry.

If you have trouble configuring the registry credentials for gcr when following the [install docs](#), use the following to set the gcr credentials:

```
registry_name="_json_key"
registry_password="$(cat /path/to/gcp/service/account/key.json) "
ytt -f /tmp/bundle/config/ \
 -v kp_default_repository='<IMAGE-REPOSITORY>' \
 -v kp_default_repository_username='<REGISTRY-USERNAME>' \
 -v kp_default_repository_password='<REGISTRY-PASSWORD>' \
 --data-value-yaml pull_from_kp_default_repo=true \
 | kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
 | kapp deploy -a tanzu-build-service -f- -y
```

## Can I configure a proxy for my Tanzu Build Service?

TBS can be configured with a proxy at [installation](#) time by specifying additional parameters:

- `http_proxy`: The HTTP proxy to use for network traffic.
- `https_proxy`: The HTTPS proxy to use for network traffic.
- `no_proxy`: A comma-separated list of hostnames, IP addresses, or IP ranges in CIDR format that should not use a proxy.

**Note:** When proxy server is enabled using `http_proxy` and/or `https_proxy`, traffic to the kubernetes API server will also flow through the proxy server. This is a known limitation and can be

circumvented by using `no_proxy` to specify the kubernetes API server.

```
ytt -f /tmp/bundle/config/ \
 -v kp_default_repository='<IMAGE-REPOSITORY>' \
 -v kp_default_repository_username='<REGISTRY-USERNAME>' \
 -v kp_default_repository_password='<REGISTRY-PASSWORD>' \
 --data-value-yaml pull_from_kp_default_repo=true \
 -v http_proxy='<HTTP-PROXY-URL>' \
 -v https_proxy='<HTTPS-PROXY-URL>' \
 -v no_proxy='<KUBERNETES-API-SERVER-URL>' \
 | kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
 | kapp deploy -a tanzu-build-service -f- -y
```

## How do I build my app locally using kpack builders?

You can use the `pack cli` with your kpack builders to test them locally before checking in your code. By using your kpack builder locally, you can guarantee that the buildpacks, stacks, and lifecycle used to build the image config will also be used by the pack CLI, resulting in a container image that is the exact same, whether it is built by `kpack` or `pack`.

**Note:** Make sure that you `docker login` or `crane auth login` to the image repository containing your kpack builder.

```
pack build my-app --path ~/workspace/my-app --builder gcr.io/my-project/my-image:lates
t --trust-builder
```

## What can I do with the `kp --dry-run` and `--output` flags?

From `kp CLI v1.0.3+` the `--dry-run` and `--output` flags are made available to `kp` commands that create or update any kpack Kubernetes resources.

The `--dry-run` flag lets you perform a quick validation with no side-effects as no objects are sent to the server. And the `--output` flag lets you view the resource in `yaml` or `json` format.

The `--dry-run-with-image-upload` flag is similar to the `--dry-run` flag in that no kpack Kubernetes resources are updated. This flag is provided as a convenience for `kp` commands that can output Kubernetes resource with generated container image references.

For example, consider the command below

```
$ kp clusterstack create test-stack \
 --dry-run \
 --output yaml \
 --build-image gcr.io/paketo-buildpacks/build@sha256:f550ab24b72586cb26215817b874b9e9e
c2ca615ede03206833286934779ab5d \
 --run-image gcr.io/paketo-buildpacks/run@sha256:21c1fb65033ae5a765a1fb44bfefdea37024c
eac86ac6098202b891d27b8671f

Creating ClusterStack... (dry run)
Uploading to 'gcr.io/my-project/my-repo'... (dry run)
Skipping 'gcr.io/my-project/my-repo/build@sha256:f550ab24b72586cb26215817b874b
9e9ec2ca615ede03206833286934779ab5d'
Skipping 'gcr.io/my-project/my-repo/run@sha256:21c1fb65033ae5a765a1fb44bfefdea
```

```

37024ceac86ac6098202b891d27b8671f'
apiVersion: kpack.io/v1alpha1
kind: ClusterStack
metadata:
 creationTimestamp: null
 name: test-stack
spec:
 buildImage:
 image: gcr.io/my-project/my-repo/build@sha256:f550ab24b72586cb26215817b874b9e9ec2c
a615ede03206833286934779ab5d
 id: io.buildpacks.stacks.bionic
 runImage:
 image: gcr.io/my-project/my-repo/run@sha256:21c1fb65033ae5a765a1fb44bfefdea37024c
ac86ac6098202b891d27b8671f
status:
 buildImage: {}
 runImage: {}

```

The resource yaml output above has the relocated build and run image urls. However, the images were never uploaded.

If you now apply the resource output using `kubectl apply -f` as shown below, then the resource will be created but will be faulty since the referenced images do not exist.

```

$ kp clusterstack create test-stack \
 --dry-run \
 --output yaml \
 --build-image gcr.io/paketo-buildpacks/build@sha256:f550ab24b72586cb26215817b874b9e9e
c2ca615ede03206833286934779ab5d \
 --run-image gcr.io/paketo-buildpacks/run@sha256:21c1fb65033ae5a765a1fb44bfefdea37024c
eac86ac6098202b891d27b8671f \
 | kubectl apply -f -
Creating ClusterStack... (dry run)
Uploading to 'gcr.io/my-project/my-repo'... (dry run)
 Skipping 'gcr.io/my-project/my-repo/build@sha256:f550ab24b72586cb26215817b874b
9e9ec2ca615ede03206833286934779ab5d'
 Skipping 'gcr.io/my-project/my-repo/run@sha256:21c1fb65033ae5a765a1fb44bfefdea
37024ceac86ac6098202b891d27b8671f'
clusterstack.kpack.io/test-stack created

```

Running the same command above with the `--dry-run-with-image-upload` flag (instead of `--dry-run`) ensures the created resource refers to images exist.

## Does TBS support Azure Devops for git repositories

Yes! Azure DevOps Git is fully supported as of TBS 1.2

## Why do I get a "repository does not exist" error when I use ECR Registry?

ECR is supported but requires manually creating each repository that TBS will use. With other registries, the repositories will be created automatically.

## How do I troubleshoot a failed build?

Like many Kubernetes native products, operating TBS involves orchestrating resources that depend on each other to function. If a resource is in a "not ready" state it is likely that there is a problem with one of the resources it depends on.

If you are encountering a not ready `Image`, check and see which builder it uses and then check the status of that builder for additional information that could help you troubleshoot the problem.

```
$ kp image status <image-name>

$ kp clusterbuilder status <clusterbuilder-name>
```

Similarly, if a builder resource is in a "not ready" state, it is possible that there is a problem with the `clusterstack` or `clusterstore` resources it is referencing.

```
$ kp clusterstack status <clusterstack-name> --verbose

$ kp clusterstore status <clusterstore-name> --verbose
```

All Build Service concepts are also Kubernetes resources. Therefore, customers can interact with them using the `kubectl` CLI to see all the information that can be provided by the Kubernetes API.

```
$ kubectl describe image <image-name>

$ kubectl describe clusterbuilder <clusterbuilder-name>
```

## How do I troubleshoot an UNAUTHENTICATED error?

During `imgpkg copy`

1. Ensure you are logged in locally to both registries with:

- If using the `docker` CLI, run:

```
docker logout registry.tanzu.vmware.com && docker login registry.tanzu.v
ware.com
docker logout <tbs-registry> && docker login <tbs-registry>
```

- If using the `crane` CLI, run:

```
rm ~/.docker/config.json
crane auth login <tbs-registry>
crane auth login registry.tanzu.vmware.com
```

2. On linux, if you have installed `docker` with `snap` you will need to copy `/root/snap/docker/471/.docker/config.json` to `~/.docker/config.json` which is where `imgpkg` is looking for the `docker` credentials.
3. Ensure your credentials have write access to your registry. This is the same repository used during installation with the `ytt/kapp` command.

- For the `docker` CLI:

```
docker push <tbs-registry>/<build-service-repository>
```

- ◆ For the `crane` CLI:

```
crane pull alpine alpine.tar && crane push alpine.tar <tbs-registry>/<build-service-repository>
```

### During `kp import`

1. Ensure you are logged in locally to both registries with:

- ◆ If using the `docker` CLI, run:

```
docker logout registry.tanzu.vmware.com && docker login registry.tanzu.vmware.com
docker logout <tbs-registry> && docker login <tbs-registry>
```

- ◆ If using the `crane` CLI, run:

```
rm ~/.docker/config.json
crane auth login <tbs-registry>
crane auth login registry.tanzu.vmware.com
```

2. Ensure the credentials used to install TBS have write access to your registry as they sometimes differ from local credentials:

- ◆ Run `docker login <tbs-registry>` or `crane auth login <tbs-registry>` using the credentials used to install TBS with `ytt/kapp`.
- ◆ Try to write to the image repository used during installation with the `ytt/kapp` command:

- For the `docker` CLI:

```
docker push <tbs-registry>/<build-service-repository>
```

- For the `crane` CLI:

```
crane pull alpine alpine.tar && crane push alpine.tar <tbs-registry>/<build-service-repository>
```

## Why does TBS leave behind pods after builds on my Cluster?

All TBS builds happen in pods. By default, TBS will not delete the last ten successful builds and the last ten failed builds for the purpose of providing historical logging and debugging. If this behavior is not desired, users can configure the number of stored build pods by modifying the `failedBuildHistoryLimit` and `successBuildHistoryLimit` on the Image resource. This is not currently supported in the `kp` CLI, but users can apply yaml configuration using `kubectl` to update these fields. Follow [this link](#) for documentation.

## How do I check what version of TBS I am using?

**After successfully installing tanzu-build-service** In terminal run the command `kubectl describe configMap build-service-version -n build-service`

Under the `data` field you will see the version of TBS you are currently using. EX:

```
data:
 version: 1.3.0
```

**Note:** This will only work for TBS versions 1.2 and above

## How does TBS use windows-based images?

When running `imgpkg copy`, the command will output the following message:

```
Skipped layer due to it being non-distributable. If you would like to include non-dist
ributable layers, use the --include-non-distributable flag
```

This is because TBS ships with windows images to support windows builds. Windows images contain "foreign layers" that are references to proprietary windows layers that cannot be distributed without proper Microsoft licensing.

By default, `imgpkg` will not relocate the proprietary windows layers to your registry. TBS also will not pull any windows layers to the cluster unless windows builds are being run so if you do not need windows this message can be ignored.

## What is the relationship between a *kpack image resource* and an *OCI image*?

"Image resource" describes a kubernetes custom resource that produces OCI images by way of build resources. This resource will continue producing new builds that, when successful, output container images to the registry as configured by the image resource.

## Pinning the Tanzu Net Updater

When you configure the Tanzu Network Auto Updater during installation, dependencies are pulled in from `network.tanzu.vmware.com` as they are released to keep the all app OCI images up-to-date and patched.

Using the steps from the installation docs will result in all dependencies to stay up-to-date with the latest buildpacks.

To pin to a TBS "descriptor" version, find the desired version of the descriptor from the [TBS Dependencies Tanzu Network](#) page. Run the following command to re-install TBS:

```
ytt -f /tmp/bundle/config/ \
-v kp_default_repository='<IMAGE-REPOSITORY>' \
-v kp_default_repository_username='<REGISTRY-USERNAME>' \
-v kp_default_repository_password='<REGISTRY-PASSWORD>' \
--data-value-yaml pull_from_kp_default_repo=true \
-v tanzunet_username='<TANZUNET-USERNAME>' \
-v tanzunet_password='<TANZUNET-PASSWORD>' \
-v descriptor_name='<DESCRIPTOR-NAME>' \
-v descriptor_version='<DESCRIPTOR-VERSION>' \
| kbld -f /tmp/bundle/.imgpkg/images.yml -f- \
| kapp deploy -a tanzu-build-service -f- -y
```



Where:

- `DESCRIPTOR-VERSION` is the desired descriptor version from the [TBS Dependencies Tanzu Network](#) page.

# Additional resources for Tanzu Build Service

## Concourse Kpack resource

The [Concourse Kpack resource](#) helps in the integration of Kpack in a Concourse based CI/CD pipeline. This Concourse resource is capable of triggering Image builds based on a commit SHA. The [Git repo](#) for the Concourse Kpack resource provides guidance on usage within a pipeline.

**Note:** The Kpack Image must be created within a TBS cluster before referring to it within a pipeline using the Concourse Kpack resource

**Note:** The Concourse Kpack resource currently only supports GKE and TKGI clusters

## Helpful Articles

- [Getting Started with VMware Tanzu Build Service 1.0](#)

(September 03, 2020 - Tony Vetter)

This covers installation of Tanzu Build Service on local Kubernetes cluster (using Docker Desktop) and demonstrates the auto build of app images for Code and OS updates.

- [VMware Tanzu Build Service, a Kubernetes-Native Way to Build Containers, Is Now GA](#)

(September 03, 2020 - Brad Bock)

A big picture overview of Tanzu Build Service, integration with CI/CD and links on getting started.

## Helpful Videos

- [Introduction to Tanzu Build Service 1.0](#)

(September 22, 2020 - Tony Vetter)

This covers the different components of TBS, the benefits it offers, and a demo of how TBS can auto update your application images for different reasons - Code update, Config change or Stack update.

## Helpful Repositories

- kpdemo - <https://github.com/matthewmcnew/kpdemo>

A tool to visualize and demo kpack.

Demos include auto Image creation for Stack and Buildpack updates.