VMware Tanzu Kubernetes Grid Integrated Edition Validation Guide

vmware®

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

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

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

Contents

About This Guide

The *VMware Tanzu Kubernetes Grid Integrated Edition (TKGI) Validation Guide* provides information about validation of the CNA (Cloud Native Application) solution running in the TKGI environment on vSphere.

Intended Audience

This guide is for experienced Windows and Linux system engineers who are familiar with VMware virtual machine technologies, data center operations, and Tanzu Kubernetes Grid Integrated Edition (TKGI). The user must have familiarity with deploying VC and ESXi, and installing and configuring required software and services.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in the VMware technical documentation, go to https://www.vmware.com/support/pubs/.

VMware Documentation

Related VMware product documentation that might be useful can be found at on the VMware website at http://www.vmware.com/support/pubs.

Kubernetes Documentation

Related Kubernetes product documentation that might be useful can be found at on the Kubernetes website at https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-interactive/ and https://kubernetes.io/blog/2018/05/01/developing-on-kubernetes/.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, submit your feedback through the VMware {code} site (https://code.vmware.com/certifications).

Introduction 1

The TKGI ISV Validation Program provides a pre-configured and ready-to-use validation testbed in the cloud.

The objective of this validation program is to ensure that the Independent Software Vendor (ISV) CNA (Cloud Native Application) solution has passed a detailed evaluation and testing process maintained by VMware for listing on the VMware Marketplace at https://marketplace.vmware.com.

Listing Application on VMware Marketplace as a validated solution provides customer assurance that the application is vetted on TKGI.

This chapter includes the following topics:

Validation Prerequisites

Ensure that the following prerequisites are met before starting the TKGI validation:

- You must be enrolled in the VMware Technology Alliance Partner (TAP) program at any level.
- You must assure that the Application is validated (tested) on the Kubernetes platform.
- Confirm that the Application meets security requirements by filing the *TKGI Validation-in-Cloud - Security Intake Questionnaire*.
- Confirm that the Application Cluster requirements meet the VMware Learning Platform (VLP) lab resources by filing the *TKGI Lab - Kubernetes Environment Intake Questionnaire*.

Validation Process

This section lists the validation process steps that you must perform to get your solution validated.

Procedure

1 Create a DCPN request.

Before validating your application, answer the following questionnaires and submit them by opening a Developer Center Partner Network (DCPN) request:

- TKGI Validation-in-Cloud - Security Intake Questionnaire
- TKGI Lab - Kubernetes Environment Intake Questionnaire

Both questionnaires are available on the **VMware {code}** site > **Certification** > **PKS Partner Application Certification**, under **Documentation and Reference** (https://code.vmware.com/group/cert/1.0/pks-partner-application-certification).

2 VMware reviews and approves the request.

After the documentation from the preceding step is reviewed and approved, you are notified on the DCPN case to go ahead with access to the cloud environment to proceed with your validation.

3 Access the Validation-in-Cloud.

To begin, log in to the VMware {code} site (https://code.vmware.com) using your My VMware or Partner Central credentials. For more information, see Chapter 2 Accessing the Lab.

4 Deploy your Application solution and run the Partner tests.

After deployment, the Application solution is powered on and configured (see Accessing TKGI Environment and Operational Tests). Perform an initial sanity test by logging in to the solution and validating its functionality (see Chapter 6 Partner Validation). Ensure that the IP address and host names are configured appropriately. Before running the VMware Validation Tests, you run a set of tests that are specific to your Application solution. It ensures that your solution operates normally before running the VMware Validation Tests.

5 Run the VMware Validation tests.

After configuring your solution for operation and running specific tests to determine your solution's stability, run the VMware Validation Tests.

6 Collect the logs and submit validation results.

After the validation tests are run successfully, collect the logs (see Chapter 8 Collecting Logs for details), and submit the validation results with the Validation Session ID and *TKGI Validation-in-Cloud - Validation Tests Check List* (see Chapter 9 Submitting Validation Results for details) in the DCPN case.

Ensure that you do not decommission the Validation-in-Cloud test environment for at least 24 hours. VMware updates the DCPN case with test completion status or instructions for any follow-up items.

Also, ensure that the Application solution is deployed and running in a good state after the validation tests are done. VMware might need to verify the Application solution deployment during the validation results review.

If the tests are successful, you are instructed on getting the Partner Ready VMware Tanzu Logo. At this time, the Validation-In-Cloud environment can be decommissioned by clicking the **FINISH** button. If the tests are unsuccessful, rerun the tests and submit new results.

7 Decommission Validation-in-Cloud.

The Validation-in-Cloud environment is automatically decommissioned after the lease expires. All server, networking, and storage configurations associated with the lab that are built on IaaS are deleted.

8 Get Partner Ready VMware Tanzu Logo.

For more information, see Chapter 10 Requesting Partner-Ready VMware Tanzu Logo.

Revalidation Requirements

VMware generally requires revalidation for:

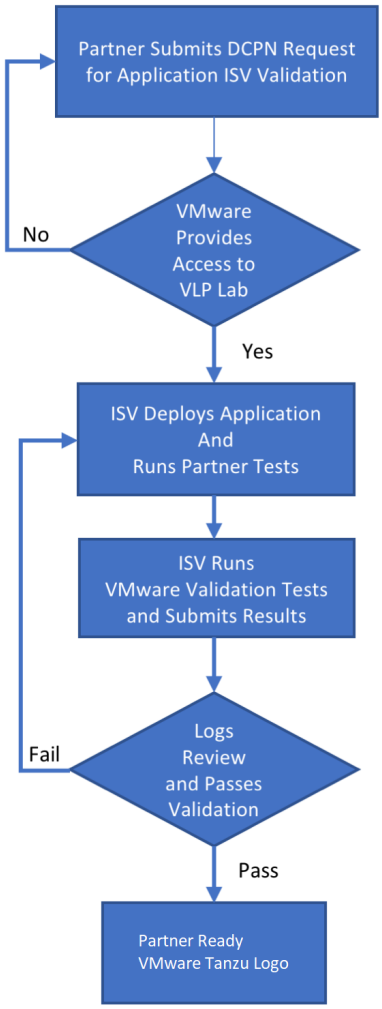
- Every major TKGI version update
- Every major partner application version update

VMware pre-announces TKGI version revalidation requirements as much as possible to allow partner resource planning.

Validation Workflow

The following figure illustrates the TKGI validation workflow:

Figure 1-1. TKGI Validation Workflow



Validation Support

To report issues with the validation environment or to ask questions regarding the validation, create a case using Developer Center Partner Network (DCPN).

Note You must create a new case for every unique issue.

To create a DCPN case, perform the following steps:

1 On the DCPN, navigate to **Cases**.

2 Click **New** and provide the necessary information:

- **Request Type**: `Program Request`
- **Project**: `priv-<company>-tanzu_PR`

  Where <company> is your company name in the DCPN.

- Under **Program Information**:
  - **Summary**: Enter a short descriptive summary about the issue reported.
  - **Description**: Enter a detailed description about the issue or the questions being asked. The description must match the summary of the case as closely as possible.

    Avoid case descriptions that are not descriptive, such as “I cannot do X. Why not?”.
- Under **VMware Details**:
  - **Sub Status**: `Waiting on VMware`.

3 Ensure to attach any supplemental documentation, screenshots, or files to the case (if necessary) in the **Case Files** section, which is located towards the bottom of the case page.

Accessing the Lab

2

This chapter provides information about accessing the VMware Validation Lab from the VMware Integration Validation (VIVa) platform.

After starting the lab, you get access to the control center Windows VM of the lab. The control center VM acts as the jump host to the lab and from here you can access the pre-configured TKGI environment.

This chapter includes the following topics:

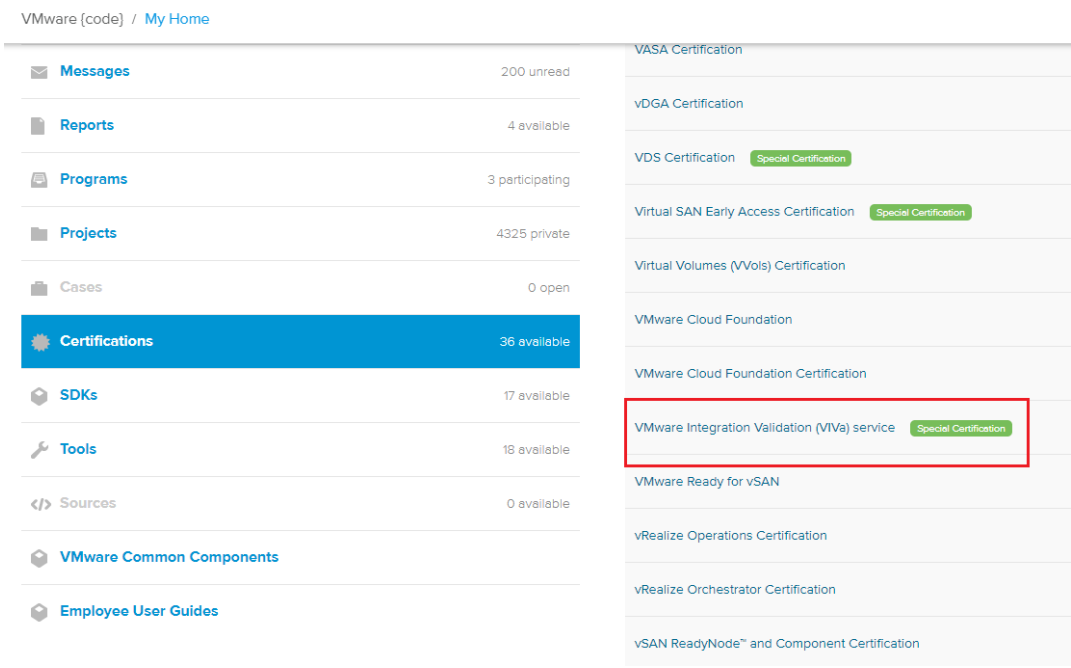
Accessing VMware Integration Validation Service

To access the VMware Integration Validation service on a web browser, perform the following steps:

Procedure

1. Go to the VMware Code site (https://code.vmware.com) and log in using your **My VMware** or **Partner Central** credentials.
2. Click **MY HOME** > **Certifications**.

3 Under **My Certifications**, scroll down and click **VMware Integration Validation (VIVa) service**.



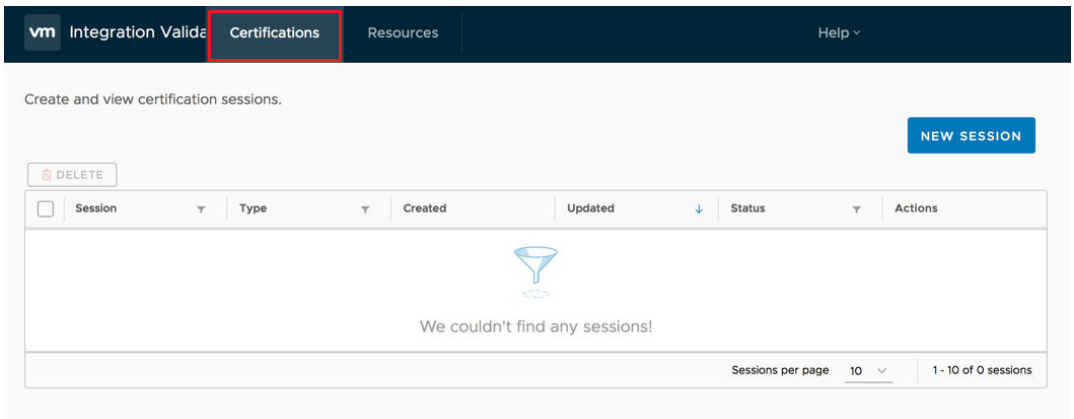
Creating a Certification Test Session

To create a certification test session for TKGI validation, perform the following steps:

Note If you are a first-time user, you must accept the End User License Agreement to proceed.

Procedure

1 On the VMware Integration Validation landing page, click **Certifications**.

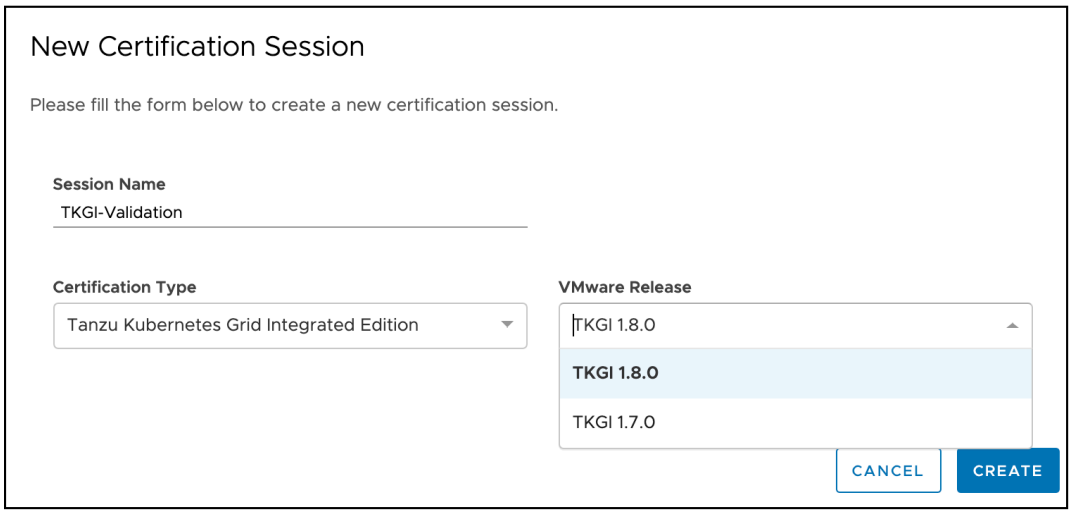


2 Click **NEW SESSION**.

3 From the **Certification Type** drop-down menu, select **Tanzu Kubernetes Grid Integrated Edition**.

4 From the **VMware Release** drop-down menu, select the PKS lab version.

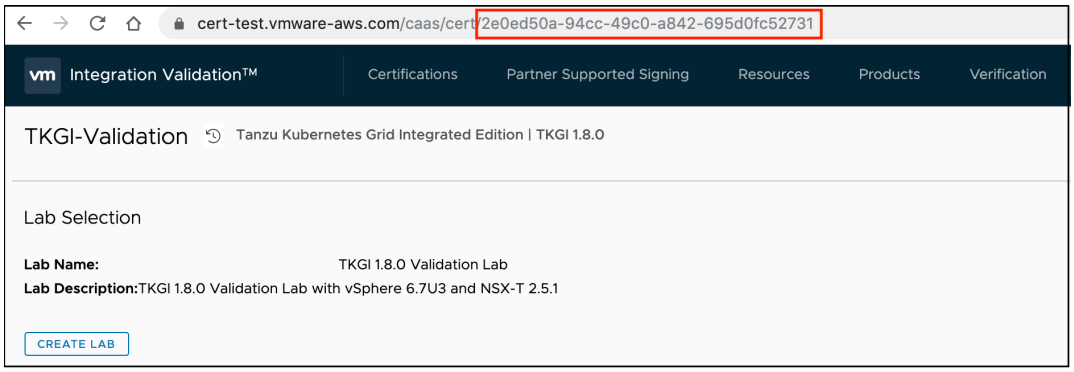
5 Click **CREATE**.



Lab Selection

After creating a TKGI validation session, you must select the lab in the **Lab Selection** page.

Important Make sure that you note down the string in the URL after *https://cert.vmware.com/caas/cert/*. This string is the VIVa Session ID. You must provide this ID in the DCPN case, with the logs after validation is completed.



1 Click **CREATE LAB**.

A cloud-based lab environment is created on the VMware Learning Platform (VLP).

Note The lab environment creation might take some time if the lab resources are not immediately available for the queued request.

2 Click **OPEN LAB**.

A new browser tab opens on the VLP site where the session is automatically authenticated with the VMware {code} user account.

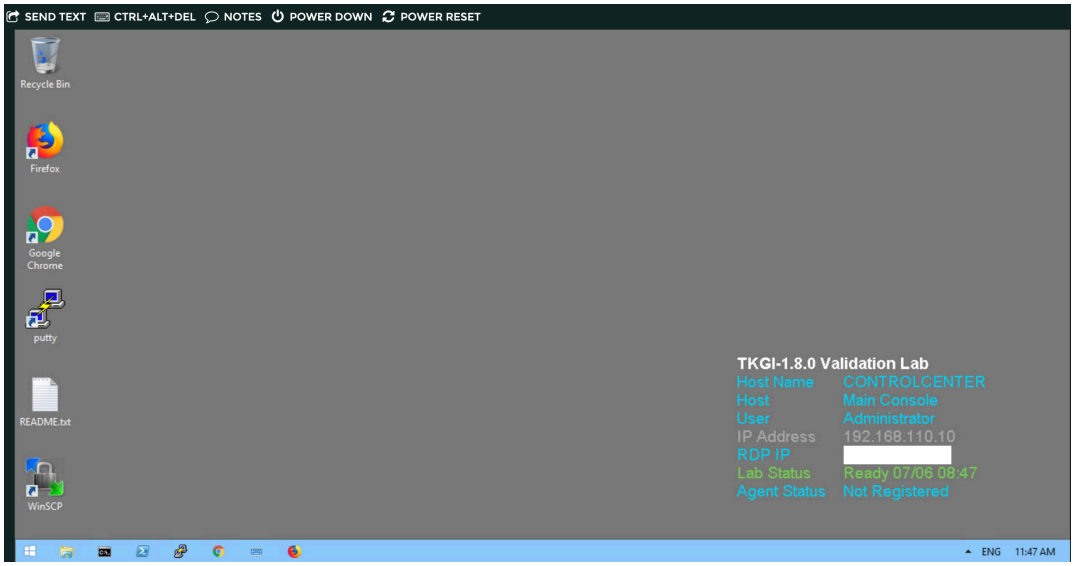
Note Ensure that the pop-up blocker on your browser is disabled.

Main Console

The Main Console is where you start working on the VMware Validation Lab. Ensure that the **Lab Status** is displayed as **Ready** on the desktop of the Main Console before you start.

- The Main Console Windows Client password is *VMware1!*
- Use the Toggle Full-Screen and Maximize options to expand the Main Console window.
- Copy and Paste operations between the console window and your desktop are disabled for security reasons. However, you can use the **SEND TEXT** button on the top left of the console to send text from your desktop to the VM.
- If you are a first-time user of the system, click the Guide icon for a guided tour of the console interface.

Figure 2-1. Lab Main Console

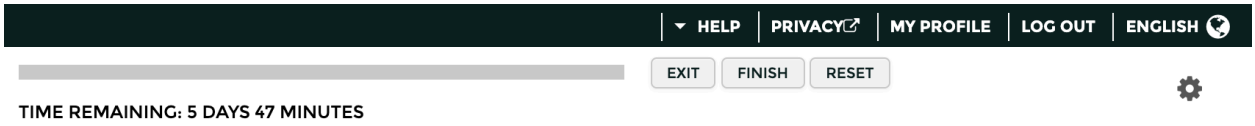


Important After you enroll in the Validation-in-Cloud program, you are provided five days to complete the validation process. If you require additional time to complete the process, you must open a request for an extension of the TKGI lab 24 hours before the Validation-in-Cloud expires.

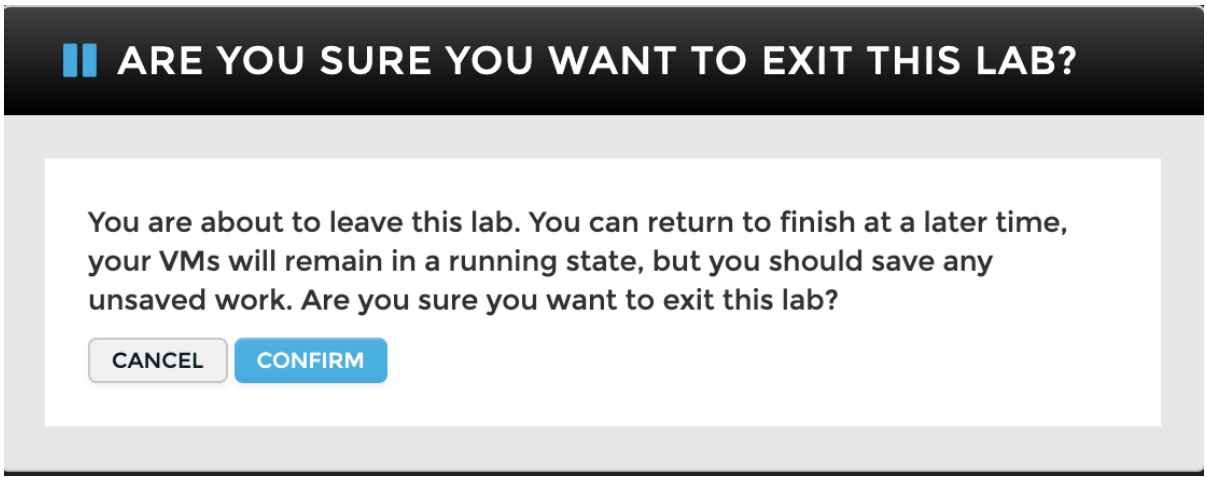
Top Menu Bar

The top menu bar on the console displays the time remaining to complete the validation process.

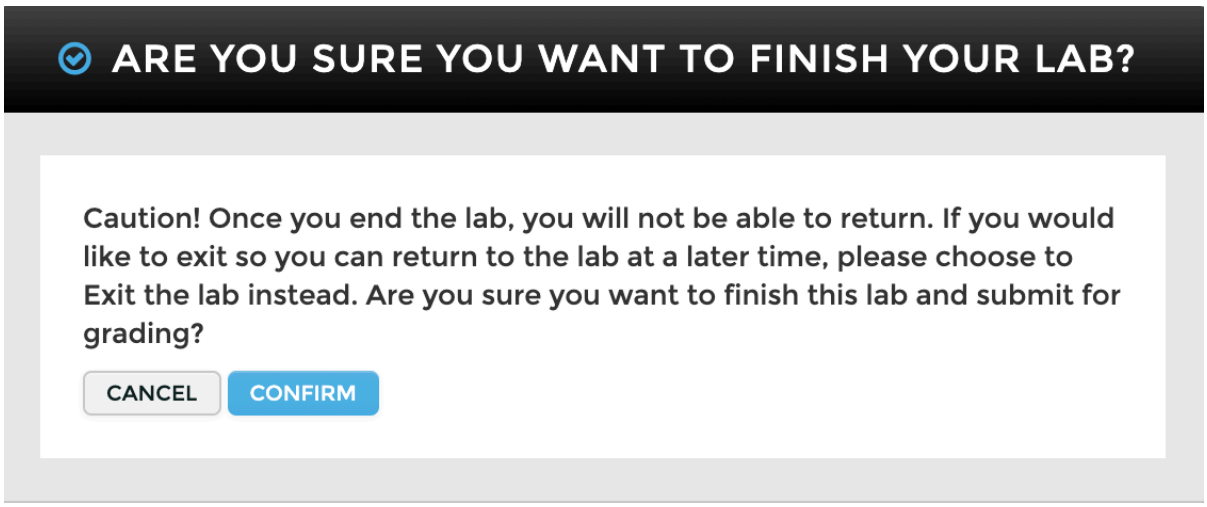
The menu bar has the following buttons:



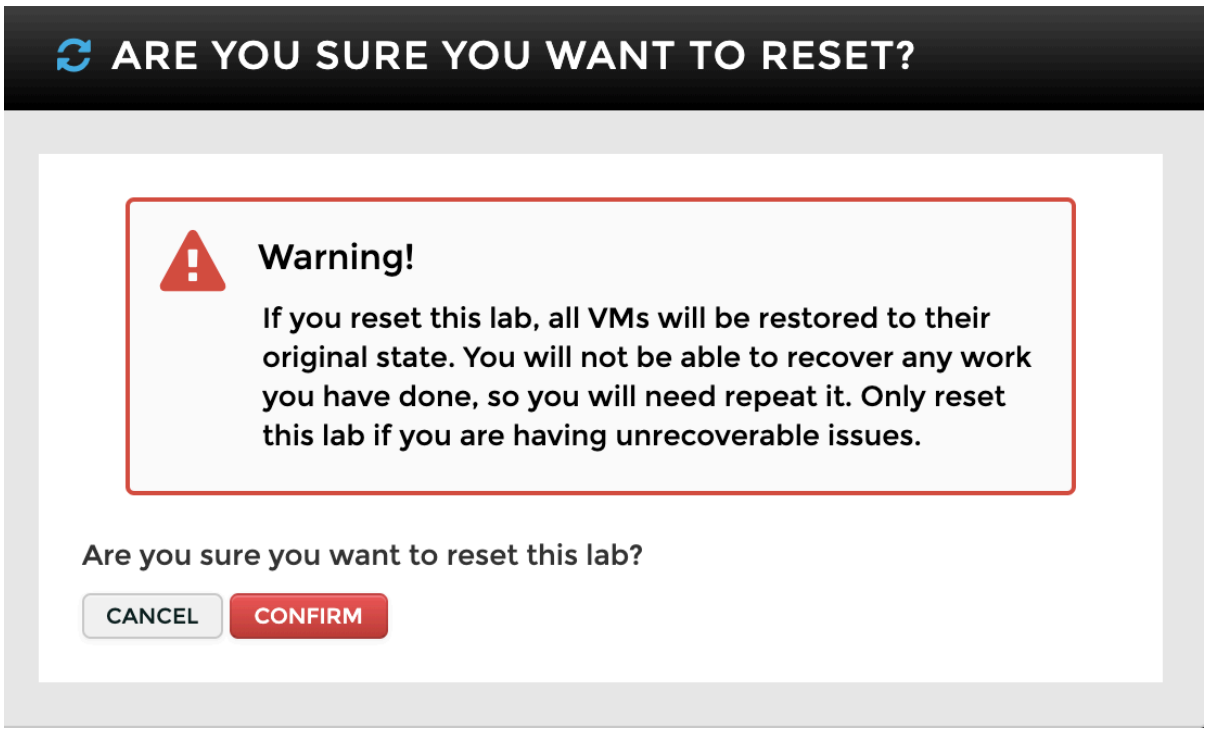
- **EXIT**: Click to exit your lab. A pop-up window appears to confirm your action. If you click **CONFIRM**, the VMware Learning Platform (VLP) considers that you want to exit the lab, and returns to the VMware Learning Platform (VLP) lab console login page. Click **CANCEL** to resume your lab session.



- **FINISH**: Click to complete your lab session. A pop-up window appears to confirm your action. If you click **CONFIRM**, the VMware Learning Platform (VLP) considers that you have completed the lab and returns to the VMware Learning Platform (VLP) lab console login page. You are not allowed to resume your lab after confirming the **FINISH** action. Click **CANCEL** to resume your lab session.



- **RESET**: Click to reset your lab. This action restores all your virtual machines to their original state. You cannot recover any work after you click **CONFIRM**.



For more information about using the lab console effectively, see Chapter 12 Tips and Troubleshooting.

TKGI Validation Tests Overview

3

The TKGI validation tests are categorized into the following test areas:

- Application Packaging and Conformance Validation (Chapter 4 Application Packaging and Conformance Validation)
- Application Life Cycle Management (Chapter 5 Application Life Cycle Management)
- Partner Validation (Chapter 6 Partner Validation)
- TKGI Centric Platform Validation (Chapter 7 TKGI Centric Platform Validation)

Note Google sample images are used as examples in the commands in this guide.

Application Packaging and Conformance Validation

4

This test area focuses on the Harbor push, pull, signing, and scanning operations.

This chapter includes the following topics:

Compliance and Security Tests

The Compliance and Security Tests group has the following test cases:

- PKS.Security.Image.Sign.PushToHarbor
- PKS.Security.Image.Harbor.Scan
- PKS.Security.Image.Harbor.Pull
- PKS.Security.Image.Harbor.Delete
- PKS.Security.Image.Harbor.Modify
- PKS.Security.Image.DockerBenchSecurity
- PKS.Security.Image.Setuid.Setgid
- PKS.Security.SignedImage.Cluster

PKS.Security.Image.Sign.PushToHarbor

This test verifies that the image can be signed and pushed to Harbor with signature recorded in notary. Signing provides protection against any container tampering that can happen in transport.

Note This test is optional.

Optional	
Test ID	PKS.Security.Image.Sign.PushToHarbor
Configuration	Follow the steps detailed in Accessing TKGI Environment.

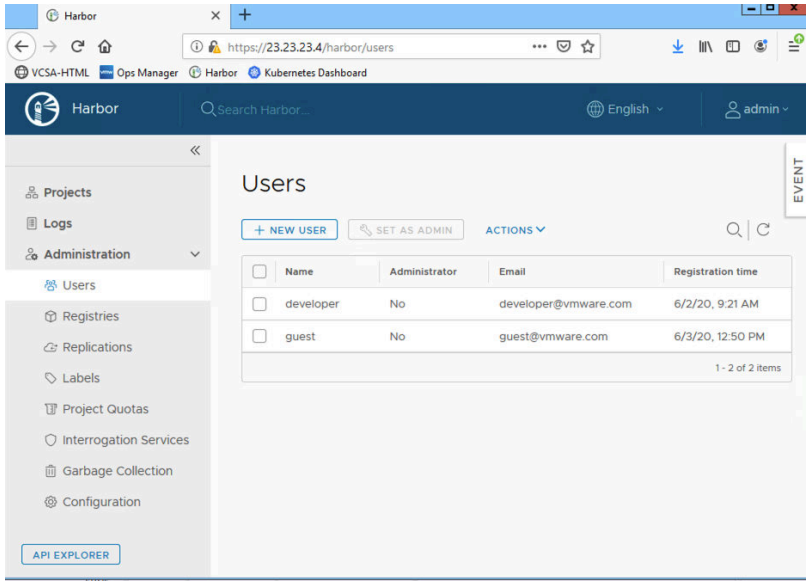
Optional

Test Script

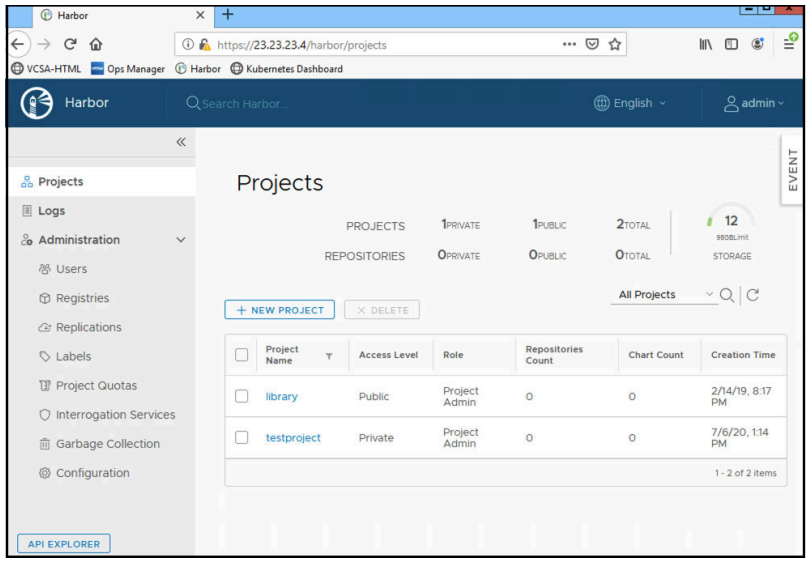
1 Open Firefox and click the **Harbor** bookmark (https://23.23.23.4).

Log in using credentials (user name: *admin* ; password: *VMware1!*).

2 Under the **Users** tab, create two users - *developer* and *guest*.



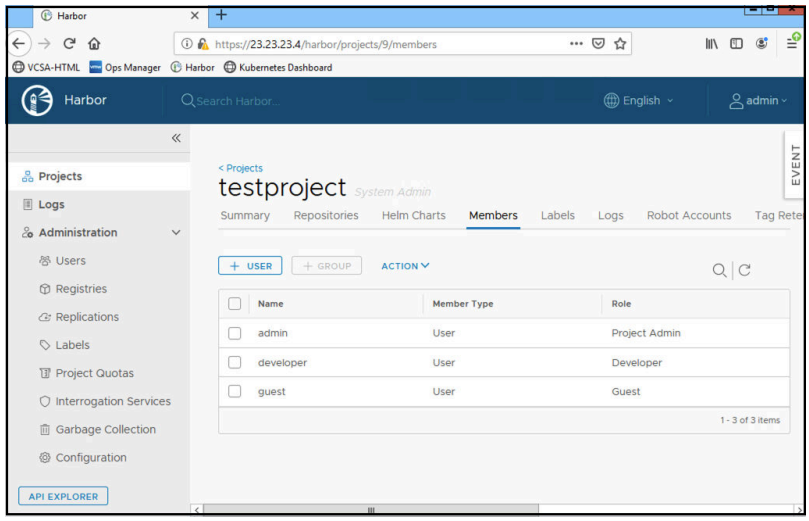
3 Under the **Projects** tab, create a new project - *testproject*.



4 Go to **Projects** > **testproject**. Under the **Members** tab, click the **+User** button and add the *developer* and the *guest* users to the project.

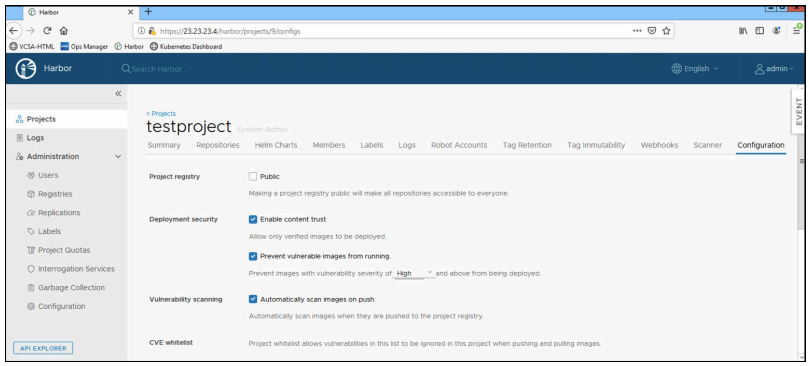
Ensure that the *developer* is assigned the `Developer` role and *guest* is assigned the `Guest` role.

Optional



5 Go to the **Configuration** tab, set the values as follows, and click **SAVE**:

- Select **Enable Content Trust**.
- Select **Prevent vulnerable images from running** and set the severity to "*High*".
- Select **Automatically scan images on push**.



6 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

7 Pull your Application image from your repository (Google sample images are used as examples).

```
root@cli-vm:~# docker pull gcr.io/google-samples/gb-frontend:v4
```

```
root@cli-vm:~# docker pull gcr.io/google_containers/redis:e2e
```

```
root@cli-vm:~# docker pull gcr.io/google_samples/gb-redisslave:v1
```

8 Log in to Harbor registry as *developer*.

```
root@cli-vm:~# docker login harbor.corp.local -u developer
```

9 Set up two environment variables and tag the images (Google sample images are used as examples).

```
root@cli-vm:~# export DOCKER_CONTENT_TRUST=1
```

```
root@cli-vm:~# export DOCKER_CONTENT_TRUST_SERVER=https://harbor.corp.local:4443
```

```
root@cli-vm:~# docker tag gcr.io/google-samples/gb-frontend:v4
harbor.corp.local/testproject/gb-frontend:v4
```

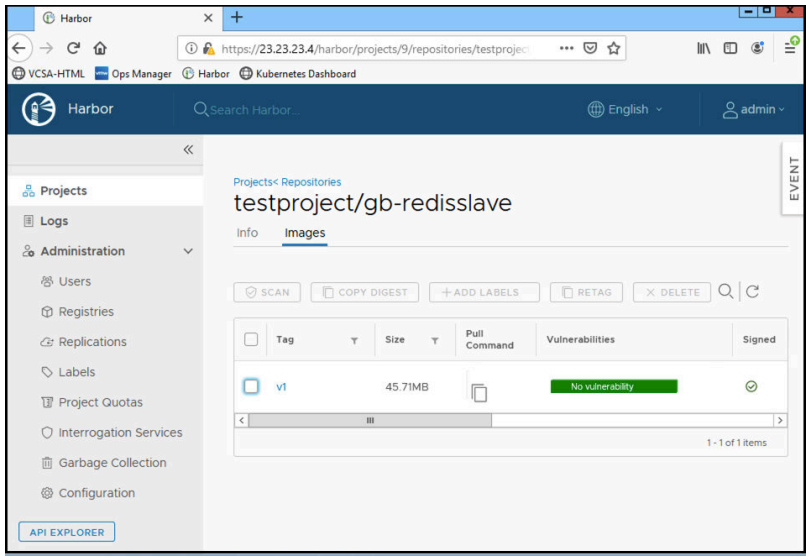
```
root@cli-vm:~# docker tag gcr.io/google_samples/gb-redisslave:v1
harbor.corp.local/testproject/gb-redisslave:v1
```

Optional	

	`root@cli-vm:~# docker tag gcr.io/google_containers/redis:e2e harbor.corp.local/testproject/redis:e2e`
	10 Push the images into the Harbor repository. You are asked to enter a root key passphrase. You must enter the passphrase (*VMware1!*) every time you sign the image.
	`root@cli-vm:~# docker push harbor.corp.local/testproject/gb-frontend:v4`
	`root@cli-vm:~# docker push harbor.corp.local/testproject/gb-redisslave:v1`
	`root@cli-vm:~# docker push harbor.corp.local/testproject/redis:e2e`
	11 Verify that the tags are marked as Signed in the Harbor UI. Go to **Projects** > **Repositories** > **testproject/gb-redisslave** (testproject/gb-frontend, or testproject/redis).

Test Output	1 Images are successfully pushed in to Harbor (see Test script step 10).
	`root@cli-vm:~# docker push harbor.corp.local/testproject/gb-frontend:v4`
	The push refers to repository [`harbor.corp.local/testproject/gb-frontend`]
	`3a31f3bf94a2: Layer already exists`
	`cdc990c9b585: Layer already exists`
	`…`
	`816f1903c60f: Layer already exists`
	`c12ecfd4861d: Layer already exists`
	`v4: digest:`
	`sha256:aaa5b327ef3b4cb705513ab674fa40df66981616950c7de4912a621f9ee03dd4`
	`size: 6968`
	`Signing and pushing trust metadata`
	`Enter passphrase for root key with ID ec1a991: VMware1!`
	`Repeat passphrase for root key with ID ec1a991: VMware1!`
	`Enter passphrase for new repository key with ID 7284e73: VMware1!`
	`Repeat passphrase for new repository key with ID 7284e73: VMware1!`
	`Finished initializing "harbor.corp.local/testproject/gb-frontend"`
	`Successfully signed harbor.corp.local/testproject/gb-frontend:v4`
	`root@cli-vm:~# docker push harbor.corp.local/testproject/gb-redisslave:v1`
	`…`
	`Signing and pushing trust metadata`
	`Enter passphrase for root key with ID ec1a991: VMware1!`
	`Enter passphrase for new repository key with ID 63e9a04: VMware1!`
	`Repeat passphrase for new repository key with ID 63e9a04: VMware1!`
	`Finished initializing "harbor.corp.local/testproject/gb-redisslave"`
	`Successfully signed harbor.corp.local/testproject/gb-redisslave:v1`
	`root@cli-vm:~# docker push harbor.corp.local/testproject/redis:e2e`
	`…`
	`Signing and pushing trust metadata`
	`Enter passphrase for root key with ID ec1a991: VMware1!`
	`Enter passphrase for new repository key with ID e7477bb: VMware1!`
	`Repeat passphrase for new repository key with ID e7477bb: VMware1!`
	`Finished initializing "harbor.corp.local/testproject/redis"`
	`Successfully signed harbor.corp.local/testproject/redis:e2e`
	2 The tags (v4, v1, e2e in the example) are marked as Signed in the Harbor UI. Go to **Projects** > **Repositories** > **testproject/gb-redisslave** (testproject/gb-frontend, or testproject/redis).

Optional

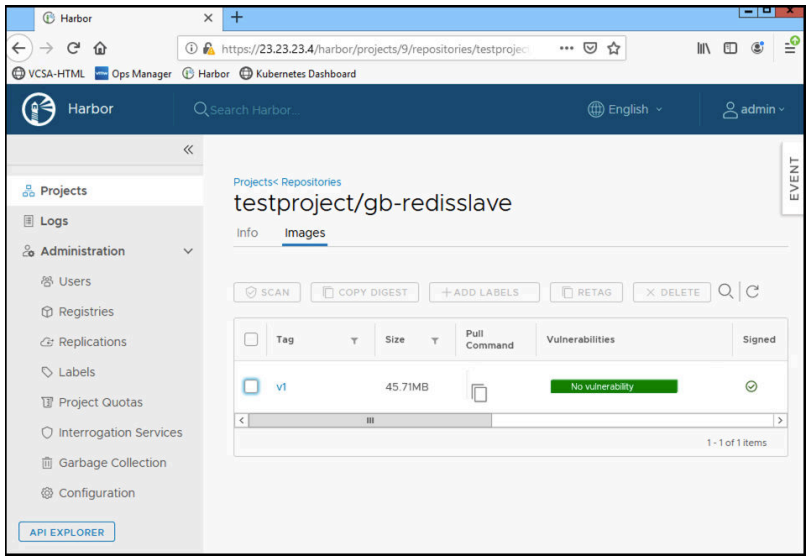


PKS.Security.Image.Harbor.Scan

This test verifies that the image does not have any vulnerability or virus by scanning the image. Vulnerability scanning and Notary services give the confidence to trust container images.

Note This test is optional.

Optional	
Test ID	PKS.Security.Image.Harbor.Scan
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor test.

Optional	
Test Script	1 Select the tag from the Harbor UI. Go to **Projects** > **Repositories** > **testproject/gb-redisslave** (testproject/gb-frontend, or testproject/redis). 2 Click **Scan**. 3 Verify that the **Vulnerability** bar is green.
Test Output	Vulnerability bar is green in the Harbor UI (see the Test script step 2).  **Note** If the Vulnerability bar is not green, fix vulnerability issues and rerun the test (in this case, application image has to be rebuilt and reloaded into Harbor to pass the validation).

Note Harbor uses Clair project for this vulnerability scanning feature. It automatically checks for updates from a few CVE sources. When an update in a CVE database is found, it downloads the data and uses it in the next scan.

PKS.Security.Image.Harbor.Pull

This test verifies that the signed image can be pulled from Harbor using the associated signature.

Note This test is optional.

Optional	
Test ID	PKS.Security.Image.Harbor.Pull
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor and PKS.Security.Image.Harbor.Scan tests.

Optional	

Test Script	1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.
	2 Create K8s secret.
	`root@cli-vm:~# kubectl create secret docker-registry regsecret --docker-server=harbor.corp.local --docker-username=developer --docker-password=VMware1! --docker-email=developer@vmware.com`
	`secret/regsecret created`
	3 Download YAML file from https://github.com/kubernetes/examples/archive/master.zip. This file is used to deploy your Application (K8S examples master.zip is used for Google sample images example).
	4 Modify your Application YAML file locally (example: Google sample YAML) to use the image location in Harbor instead of using remote repository.
	Example:
	https://github.com/kubernetes/examples/blob/master/guestbook/all-in-one/guestbook-all-in-one.yaml
	■ Line 38: replace 'image: k8s.gcr.io/redis:e2e' with 'image: harbor.corp.local/testproject/redis:e2e'
	■ Line 82: replace 'image: gcr.io/google_samples/gb-redisslave:v1' with 'image: harbor.corp.local/testproject/gb- redisslave:v1'
	■ Line 133: replace 'image: gcr.io/google-samples/gb-frontend:v4' with 'image: harbor.corp.local/testproject/gb-frontend:v4'
	Add `imagePullSecrets` with name *regsecret* for `spec` for all images to be used:
	`spec:`
	`  imagePullSecrets:`
	`  - name: regsecret`
	`  containers:`
	`  - name: master`
	`  image: harbor.corp.local/testproject/redis:e2e`
	5 Copy the modified `YAML` file into `/root` on CLI-VM. See Downloading and Transferring Files for details.
	6 Deploy your Application from previously signed Harbor images using modified YAML file (Google sample images are used as examples).
	`root@cli-vm:~# kubectl apply -f ./guestbook-all-in-one.yaml`
	`service/redis-master created`
	`deployment.apps/redis-master created`
	`service/redis-slave created`
	`deployment.apps/redis-slave created`
	`service/frontend created`
	`deployment.apps/frontend created`
	7 Verify that the Application is deployed.
	`root@cli-vm:~# kubectl get all`

Test Output	Application is deployed successfully (see Test script step 7). **Note** It takes up to two minutes to get the pods status to *Running*. At the beginning, the status is *Container creating*.

Optional

```
root@cli-vm:~# kubectl get all
```

```
NAME                                READY      STATUS    RESTARTS      AGE
pod/frontend-7776b5c947-254z6       1/1        Running   0             1m
pod/frontend-7776b5c947-9pv22       1/1        Running   0             1m
pod/frontend-7776b5c947-ccvwx       1/1        Running   0             1m
pod/redis-master-5c95c89d9-488gd    1/1        Running   0             1m
pod/redis-slave-6548b4f8b9-hpcv9    1/1        Running   0             1m
pod/redis-slave-6548b4f8b9-m4hts    1/1        Running   0             1m

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
service/frontend       ClusterIP   10.100.200.133   <none>        80/TCP      1m
service/kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP     3d
service/redis-master   ClusterIP   10.100.200.205   <none>        6379/TCP    1m
service/redis-slave    ClusterIP   10.100.200.29     <none>       6379/TCP    1m

NAME                           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend       3         3         3            3           1m
deployment.apps/redis-master   1         1         1            1           1m
deployment.apps/redis-slave    2         2         2            2           1m

NAME                                      DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-685d76557        3         3         3         1m
replicaset.apps/redis-master-6497dd4b79   1         1         1         1m
replicaset.apps/redis-slave-5d8774ddbc    2         2         2         1m
```

PKS.Security.Image.Harbor.Delete

This test verifies that the signed image can be deleted from Harbor.

Note This test is optional.

Optional	
Test ID	PKS.Security.Image.Harbor.Delete
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor and PKS.Security.Image.Harbor.Scan tests.

Optional

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Install 'notary' CLI client.

```
root@cli-vm# wget https://github.com/theupdateframework/notary/releases/download/
v0.6.0/notary-Linux-amd64
```

```
--2018-11-15 11:45:57--  https://github.com/theupdateframework/notary/releases/
download/v0.6.0/notary-Linux-amd64
Resolving github.com (github.com)... 192.30.255.113, 192.30.255.112
…
HTTP request sent, awaiting response... 200 OK
…
2018-11-15 11:45:59 (9.00 MB/s) - 'notary-Linux-amd64' saved [11109184/11109184]
```

```
root@cli-vm:~# chmod +x notary-Linux-amd64
```

```
root@cli-vm:~# mv ./notary-Linux-amd64 /usr/bin/notary
```

3 Remove tag from previously signed Application image (Google sample images are used as examples).

```
root@cli-vm:~# notary -s https://harbor.corp.local:4443 -d ~/.docker/trust remove
harbor.corp.local/testproject/gb-frontend v4 --tlscacert /etc/docker/certs.d/
harbor.corp.local/ca.crt
```

```
Removal of v4 from harbor.corp.local/testproject/gb-frontend staged for next
publish.
```

4 Verify the status of changes for the previously signed Application image.

```
root@cli-vm:~# notary -s https://harbor.corp.local:4443 -d ~/.docker/trust status
harbor.corp.local/testproject/gb-frontend --tlscacert /etc/docker/certs.d/
harbor.corp.local/ca.crt
```

5 Publish pending changes for the previously signed Application image.

```
root@cli-vm:~# notary -s https://harbor.corp.local:4443 -d ~/.docker/trust publish
harbor.corp.local/testproject/gb-frontend --tlscacert /etc/docker/certs.d/
harbor.corp.local/ca.crt
```

```
Pushing changes to harbor.corp.local/testproject/gb-frontend
Enter username: developer
Enter password: VMware1!
Enter passphrase for targets key with ID 7284e73: VMware1!
Successfully published changes for repository harbor.corp.local/testproject/gb-
frontend
```

6 Select tag (v4 in example) from the Harbor UI: **Projects** > **Repositories** > **testproject/gb-frontend** to make sure that it is shown as unsigned.

7 Select tag (v4 in example) and click **DELETE**. Confirm deletion by clicking **DELETE** in the pop-up dialog-box.

8 Delete (untag) image from Harbor using the Docker command.

```
root@cli-vm:~# docker image rm harbor.corp.local/testproject/gb-frontend:v4
```

```
Untagged: harbor.corp.local/testproject/gb-frontend:v4
Untagged: harbor.corp.local/testproject/
gb-frontend@sha256:12e31f7f4f8fa2f63c338bdf3e1b1fe04e95246ebfc5bacd7fa75125e7255a7e
```

9 Delete the Application image using the Docker command.

Optional

```
root@cli-vm:~# docker image rm gcr.io/google-samples/gb-frontend:v4
```

```
Untagged: gcr.io/google-samples/gb-frontend:v4
Untagged: gcr.io/google-samples/gb-frontend@sha256:
d44e7d7491a537f822e7fe8615437e4a8a08f3a7a1d7d4cb9066b6b
Deleted: sha256:e2b3e8542af735080e6bda06873ce666e2319eea353884a88e45f3c9ef996846
…
Deleted: sha256:c12ecfd4861d454a39fc17d8ef351b183425657e607def95bfa75e482d49fdce
```

10 Verify that the Docker image is deleted.

```
root@cli-vm:~# docker images
```

Test Output

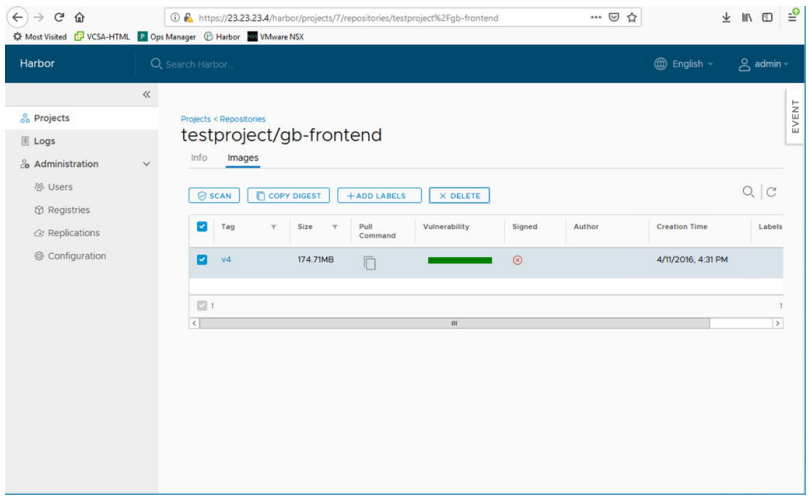
1 Previously signed Application image has unpublished changes (see Test script step 4).

```
root@cli-vm:~# notary -s https://harbor.corp.local:4443 -d ~/.docker/trust status
harbor.corp.local/testproject/gb-frontend --tlscacert /etc/docker/certs.d/
harbor.corp.local/ca.crt
```

```
Unpublished changes for harbor.corp.local/testproject/gb-frontend:

#    ACTION    SCOPE      TYPE      PATH
-    ------    -----      ----      ----
0    delete    targets    target    v4
```

2 Tag (v4 in example) is shown as unsigned in the Harbor UI: **Projects** > **Repositories** > **testproject/gb-frontend** (see Test script step 6).



3 Docker image is deleted (see Test script step 10).

```
root@cli-vm:~# docker images
```

```
REPOSITORY          TAG         IMAGE ID          CREATED          SIZE
```

```
root@cli-vm:~#
```

PKS.Security.Image.Harbor.Modify

This test modifies a container and verifies that the modified image can be created, signed, and pushed into Harbor.

Note This test is optional.

Optional	
Test ID	PKS.Security.Image.Harbor.Modify
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor and PKS.Security.Image.Harbor.Scan tests.

Optional

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Run the container from a previously signed image (Google sample images are used as examples).

```
root@cli-vm:~# docker run -i -t harbor.corp.local/testproject/gb-frontend:v4 /bin/bash
```

```
root@1e395146ecc1:/var/www/html#
```

```
root@1e395146ecc1:/var/www/html# ls -l
```

```
total 12
-rw-r--r-- 1 root root 966 Sep  9  2015 controllers.js
-rw-r--r-- 1 root root 932 Feb 20  2016 guestbook.php
-rw-r--r-- 1 root root 921 Sep  9  2015 index.html
```

3 Modify the container by adding a *testfile*.

```
root@1e395146ecc1:/var/www/html# touch testfile
```

```
root@1e395146ecc1:/var/www/html# ls -l
```

```
total 12
-rw-r--r-- 1 root root 966 Sep  9  2015 controllers.js
-rw-r--r-- 1 root root 932 Feb 20  2016 guestbook.php
-rw-r--r-- 1 root root 921 Sep  9  2015 index.html
-rw-r--r-- 1 root root   0 Nov 29 22:08 testfile
```

4 Exit from container.

```
root@1e395146ecc1:/var/www/html# exit
```

```
exit
```

5 Save changes to the image (use container ID *1e395146ecc1* from the interactive prompt in steps 2–4).

```
root@cli-vm:~#docker commit 1e395146ecc1 harbor.corp.local/testproject/gb-frontend:testfile
```

```
sha256:ac2dc5354fbb73099bf75f1a4ce4b3dc041d8276129ab8d8babb55372ab87298
```

6 Push the modified image into Harbor.

```
root@cli-vm:~# docker push harbor.corp.local/testproject/gb-frontend:testfile
```

```
The push refers to repository [harbor.corp.local/testpoject/gb-frontend]
ec13c8302e63: Pushed
…
testfile: digest:
sha256:ed9cbb8f7366cfc2435bd83b36c07bfc5f113b2ee628d288b1e53d81d0cd5e35
size: 7175
```

7 Run the container from the modified image.

```
root@cli-vm:~# docker run -i -t harbor.corp.local/testproject/gb-frontend:testfile /bin/bash
```

8 Verify that the container includes *tesfile* added to the modified image.

```
root@6974e8f281b9:/var/www/html# ls -l
```

9 Exit from container.

Optional	

	`root@6974e8f281b9:/var/www/html# exit`
	
	`exit`

Test Output	Container includes the *testfile* added to the modified image (see Test script step 8).
	`root@6974e8f281b9:/var/www/html# ls –l`
	
	`total 12`
	`-rw-r--r-- 1 root root 966 Sep  9  2015 controllers.js`
	`-rw-r--r-- 1 root root 932 Feb 20  2016 guestbook.php`
	`-rw-r--r-- 1 root root 921 Sep  9  2015 index.html`
	`-rw-r--r-- 1 root root   0 Nov 29 22:08 testfile`

PKS.Security.Image.DockerBenchSecurity

This test verifies that the Application container runtime does not have any security issues.

Note It is recommended to run this test.

Recommended	
Test ID	PKS.Security.Image.DockerBenchSecurity
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor and PKS.Security.Image.Harbor.Scan tests.

Recommended	

Test Script	1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.
	2 Install Docker Bench Security script.
	This script scans your Docker containers for various aspects and reports the results on which you can act upon. You can perform the following tests using the Docker bench security:
	■ Host configuration
	■ Docker daemon configuration
	■ Container images
	■ Container runtime, and so on.
	You can download the Docker bench script from Git clone by running the following command:
	`root@cli-vm:~# git clone https://github.com/docker/docker-bench-security.git`
	3 Run the container from the Application image stored in Harbor to get the Container Runtime output in step 5.
	`root@cli-vm:~#docker run -i -t harbor.corp.local/testproject/gb-frontend:v4 /bin/bash &`
	`[1] 10380`
	4 Verify that the container is running on background.
	`root@cli-vm:~# docker container ls -a`
	5 Start the container.
	`root@cli-vm:~# docker start 3f423882a7e1`
	`3f423882a7e1`
	6 Verify that the container started.
	`root@cli-vm:~# docker container ls`
	7 Run the docker-bench-security script.
	`root@cli-vm:~# cd docker-bench-security/`
	`root@cli-vm:~/docker-bench-security# ./docker-bench-security.sh`
	8 The docker-bench-security command checks your Application (running on Docker in the PKS environment) for various security aspects. A detailed report is displayed, which shows tests that are passed and tests that have warnings. See Chapter 13 Appendix A: Sample Docker Bench Security Script Output for details.
	Total score calculation in test output:
	[WARN] Scored -1
	[PASS] Scored +1
	Not score 0
	9 Clean up by stopping and removing container.
	`root@cli-vm:~# docker stop 3f423882a7e1`
	`3f423882a7e1`
	`root@cli-vm:~# docker rm 3f423882a7e1`
	`3f423882a7e1`

Recommended	
	10 Check the warnings marked as [WARN]. Address security recommendations to increase the Total score. Rerun and restart the container. Rerun docker-bench-security script and check the report.

Test Output	1 Container is running on background (see Test script step 4).
	
	`root@cli-vm:~# docker container ls -a`
	
	`CONTAINER ID  IMAGE COMMAND  CREATED  STATUS  PORTS  NAMES 3f423882a7e1  harbor.corp.local/testproject/gb-frontend  "/bin/ bash"  14 seconds ago  Created`
	
	2 Container has started (see Test script step 6).
	
	`root@cli-vm:~# docker container ls`
	
	`CONTAINER ID  IMAGE COMMAND  CREATED  STATUS  PORTS NAMES 3f423882a7e1  harbor.corp.local/testproject/gb-frontend  "/bin/ bash"  50 seconds ago  Up 9 seconds  80/tcp competent_feynman`

PKS.Security.Image.Setuid.Setgid

This test verifies that `Setuid` and `Setgid` binaries are removable from the container created from the Application image.

Note This test is mandatory.

Mandatory	
Test ID	PKS.Security.Image.Setuid.Setgid
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor and PKS.Security.Image.Harbor.Scan tests.
Test Script	1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details. 2 `setuid` and `setgid` are UNIX access rights flags that allow you to run an executable file with the permissions of the executable's owner or group respectively, and to change behavior in directories. To get a list of binaries with special permissions in the Application container image, start the container from image stored in Harbor (Google sample images are used as examples). `root@cli-vm:~# docker run harbor.corp.local/testproject/gb-frontend:v4 find / -perm +6000 -type f -exec ls -ld {} \; 2> /dev/null`

Mandatory	
Test Output	Collect the list of binaries with special permissions in the Application container image. `root@cli-vm:~# docker run harbor.corp.local/testproject/gb-frontend:v4 find / -perm +6000 -type f -exec ls -ld {} \; 2> /dev/null` `-rwxr-sr-x 1 root shadow 35408 Jan 28  2016 /sbin/unix_chkpwd` `-rwsr-xr-x 1 root root 39912 Nov 18  2015 /usr/bin/newgrp` `…` `-rwsr-xr-x 1 root root 40000 Mar 29  2015 /bin/mount` `-rwsr-xr-x 1 root root 61392 Oct 28  2014 /bin/ping6` If the test fails, perform the following steps to unset special permission flags (in this case, application image has to be rebuilt and reloaded into Harbor to pass the validation): 1 Unset SUID (special permissions flags): `root@cli-vm:~# docker run -i -t harbor.corp.local/testproject/gb-frontend:v4 /bin/bash` `root@43f4fad7578f:/var/www/html# for i in `find / -perm +6000 -type f`; do chmod a-s $i; done` `find: `/proc/21/task/21/fd/5': No such file or directory` `find: `/proc/21/task/21/fdinfo/5': No such file or directory` `find: `/proc/21/fd/5': No such file or directory` `find: `/proc/21/fdinfo/5': No such file or directory` 2 Verify that SUID is not set. `root@43f4fad7578f:/var/www/html# find / -perm +6000 -type f -exec ls -ld {} \; 2> /dev/null` `root@43f4fad7578f:/var/www/html#` 3 To save changes to the image, exit from container. `root@43f4fad7578f:/var/www/html# exit` `exit` 4 Commit changes into image with the new tag. `root@cli-vm:~#docker commit 43f4fad7578f harbor.corp.local/testproject/gb-frontend:nosuid` `sha256:ac2dc5354fbb73099bf75f1a4ce4b3dc041d8276129ab8d8babb55372ab87298` 5 Push the modified image into Harbor. `root@cli-vm:~# docker push harbor.corp.local/testproject/gb-frontend:nosuid` `The push refers to repository [harbor.corp.local/testpoject/gb-frontend]` `ec13c8302e63: Pushed` `…` `testfile: digest:` `sha256:ed9cbb8f7366cfc2435bd83b36c07bfc5f113b2ee628d288b1e53d81d0cd5e35` `size: 7175` 6 Run the container from the modified image. `root@cli-vm:~# docker run -i -t harbor.corp.local/testproject/gb-frontend:nosuid /bin/bash` `root@ f0154e22dbc4 :/var/www/html#`

Mandatory	

7 Start another PuTTY session to CLI-VM and verify that the container from the modified image is running.

```
root@cli-vm:~# docker container ls
```

```
CONTAINER ID        IMAGE
COMMAND             CREATED             STATUS
PORTS               NAMES
f0154e22dbc4        harbor.corp.local/testproject/gb-frontend   "/bin/
bash"         38 seconds ago      Up 9 seconds        80/tcp
suspicious_tesla
```

PKS.Security.SignedImage.Cluster

This test verifies that the Application image can be used from Harbor in a cluster.

Note This test is mandatory.

Mandatory	
Test ID	PKS.Security.SignedImage.Cluster
Configuration	Run the PKS.Security.Image.Sign.PushToHarbor and PKS.Security.Image.Harbor.Scan tests.

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Verify that the Application image is already pushed into Harbor *testproject* (Google sample images are used as examples.

```
root@cli-vm:~# docker images
```

3 Log in to PKS.

```
root@cli-vm:~# pks login -a uaa.corp.local -u vmware -p VMware1! -k
```

```
API Endpoint: uaa.corp.local
User: vmware
```

4 Create a cluster (test-cluster). See Accessing TKGI Environment for details.

```
root@cli-vm:~# pks create-cluster test-cluster --external-hostname ext-test-cluster.corp.local --plan small --num-nodes 3
```

Note The number of nodes and plan configuration might differ based on your application requirements.

You can skip this step if you have already created a PKS cluster.

5 Configure *test-cluster* to use Harbor.

```
root@cli-vm:~# pks cluster test-cluster
```

- Check output for Kubernetes Master IP(s).
- Edit the /etc/hosts file and add `Kubernetes_Master_IP(s) ext-test-cluster.corp.local` line at the end of the file.

  ```
  root@cli-vm:~# vi /etc/hosts
  ```

- Get credentials for the cluster - *test-cluster*.

  ```
  root@cli-vm:~# pks get-credentials test-cluster
  ```

  ```
  Fetching credentials for cluster test-cluster.

  Context set for cluster test-cluster.

  User can now switch between clusters by using:
                $kubectl config use-context <cluster-name>
  ```

  ```
  root@cli-vm:~# kubectl config use-context test-cluster
  ```

  ```
   Switched to context test-cluster.
  ```

6 Create a YAML file (example: `your-app.yaml`).

```
root@cli-vm~# vi your-app.yaml
```

```
---
apiVersion: v1
kind: Pod
metadata:
  name: your-app
spec:
 containers:
 - name: your-app-container
   image: harbor.corp.local/testproject/your-app-name
 imagePullSecrets:
 - name: regsecret
```

Mandatory

7 Deploy the Application image from Harbor using *test-cluster*.

```
root@cli-vm~# kubectl apply -f ./your-app.yaml
```

```
deployment.apps/your-app created
```

8 Verify the deployment. Check that the pods with Application are up within two minutes.

```
root@cli-vm:~# kubectl get all
```

9 Delete the deployment.

```
root@cli-vm~# kubectl delete deployment.apps/your-app
```

```
deployment.apps “your-app” deleted
```

10 Verify that the Application deployment is deleted.

```
root@cli-vm~# kubectl get all
```

Test Output

1 Application image is pushed into Harbor *testproject* (Google sample images are used as example) - see Test script step 2.

```
root@cli-vm:~# docker images
```

```
REPOSITORY                                    TAG       IMAGE ID
CREATED         SIZE
gcr.io/google-samples/gb-frontend             v4        e2b3e8542af7     2
years ago     512MB
harbor.corp.local/testproject/gb-frontend     v4        e2b3e8542af7     2
years ago     512MB
gcr.io/google_samples/gb-redisslave           v1        5f026ddffa27     3
years ago     110MB
harbor.corp.local/library/gb-redisslave       v1        5f026ddffa27     3
years ago     110MB
gcr.io/google_containers/redis                e2e       e5e67996c442     3
years ago     419MB
harbor.corp.local/library/redis               e2e       e5e67996c442     3
years ago     419MB
```

2 Application is deployed. Pods with your application become available within two minutes (see Test script step 8).

```
root@cli-vm:~# kubectl get all
```

```
NAME                                 READY        STATUS      RESTARTS   AGE
pod/your-app-865974f8d5-xt2vp        1/1          Running       0          2m

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/your-app       ClusterIP   10.100.200.141   <none>        80/TCP
2m
service/kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP
3d

NAME                                       DESIRED  CURRENT   UP-TO-DATE
AVAILABLE   AGE
deployment.apps/your-app                    1         1        1
1           2m
NAME                                       DESIRED   CURRENT       READY
AGE
replicaset.apps/your-app-865974f8d5        1         1
```

3 Verify that the application deployment is deleted (see Test script step 10).

Mandatory	
	root@cli-vm~# kubectl get all NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE service/kubernetes ClusterIP 10.100.200.1 <none> 443/TCP 3d

Application Life Cycle Management

5

This test area focuses on LCM (application install and uninstall operations), and verifies Kubernetes life cycle commands with the Application.

This chapter includes the following topics:

Operational Tests

This group has the following test cases:

PKS.LCM.Image.Build

This test verifies that the Application image can be built and pushed to Harbor.

Note This test is optional.

Optional	
Test ID	PKS.LCM.Image.Build
Configuration	Follow the steps detailed in Accessing TKGI Environment.

Optional	

Test Script	1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.
	2 Clone the Application source from external Git repository. For example:
	`root@cli-vm:~# git clone https://github.com/your-repository-name/your-project-name`
	3 Build the Docker image from the cloned Git source of application. For example: your-project-name/ your-app name has Dockerfile:
	`root@cli-vm:~/your-project-name # docker build -t your-project-name/your-app-name:your_tag`
	`Sending build context to Docker daemon  24.58MB`
	`…`
	`Successfully built f3a1658dd930`
	`Successfully tagged your-project-name/your-app-name:your_tag`
	4 Verify that the Application image is successfully built.
	`root@cli-vm:~/your-project-name # docker images`
	5 Create a project - '*your-project-name*' for your Application solution in Harbor. See step 3 in PKS.Security.Image.Sign.PushToHarbor.
	6 Tag the newly built Docker image.
	`root@cli-vm:~/ your-project-name # docker tag your-project-name/your-app-name:your_tag harbor.corp.local/your-project_name/your-app-name:your_tag`
	7 Push the tagged Docker image into Harbor.
	`root@cli-vm:~/your-app-name # docker push harbor.corp.local/your-project_name/ your-app-name:your_tag`
	`The push refers to repository [harbor.corp.local/your-project_name/your-app-name]`
	`aee035870ae6: Pushed`
	`3e8a55eec1e1: Pushed`
	`bae9908faa30: Pushed`
	`your_tag: digest: sha256:ba23b869f5d813480704f11e52a7a03b3bebf65648b5dbe3bac02c46f6ed5124 size: 1160`
	8 Open Firefox and click the **Harbor** bookmark (https://23.23.23.4).
	Log in using credentials (user name: *admin* ; password: *VMware1!*).
	9 Verify that the Application image is successfully pushed into Harbor.

Test Output	1 Application image is successfully built from Dockerfile (see Test script step 4).
	`root@cli-vm:~/your-project-name # docker images`
	`REPOSITORY  TAG  IMAGE ID  CREATED  SIZE`
	`your-project-name /your-app-name  your_tag  f3a1658dd930  28 seconds ago  45.8MB`
	2 Application image is successfully pushed into Harbor (see Test script step 9).
	Tag *your_tag* is shown in the Harbor UI. Go to **Projects** > **Repositories** > **your-project-name/your-app-name**.

PKS.LCM.App.Install

This test verifies that the Application image can be pulled from Harbor and installed.

Note This test is mandatory.

Mandatory	
Test ID	PKS.LCM.App.Install
Configuration	Application image is uploaded into Harbor.

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Log in to PKS.

```
root@cli-vm:~# pks login -a uaa.corp.local -u vmware -p VMware1! -k
```

```
API Endpoint:uaa.corp.local
User:vmware
```

3 Create a cluster - test-cluster (see Accessing TKGI Environment for details) or use the test-cluster that you created in PKS.Security.SignedImage.Cluster.

```
root@cli-vm:~# pks create-cluster test-cluster --external-hostname ext-test-cluster.corp.local --plan small --num-nodes 3
```

Note The number of nodes and plan configuration might differ based on your application requirements.

You can skip this step if you have already created a PKS cluster.

4 Configure *test-cluster* to use Harbor.

```
root@cli-vm:~# pks cluster test-cluster
```

- Check output for Kubernetes Master IP(s).
- Edit the /etc/hosts file and add `Kubernetes_Master_IP(s) ext-test-cluster.corp.local` line at the end of the file.

```
root@cli-vm:~# vi /etc/hosts
```

- Get credentials for the cluster - *test-cluster*.

```
root@cli-vm:~# pks get-credentials test-cluster
```

```
Fetching credentials for cluster test-cluster.

Context set for cluster test-cluster.

User can now switch between clusters by using:
  $kubectl config use-context <cluster-name>
```

```
root@cli-vm:~# kubectl config use-context test-cluster
```

```
Switched to context "test-cluster".
```

5 Deploy the Application image from Harbor using *test-cluster*.

```
root@cli-vm~# kubectl run your-app --image=harbor.corp.local/ your-project_name/your-app-solution-name:your_tag
```

```
deployment.apps/your-app created
```

6 Verify the deployment. Check that the pods with Application are up within two minutes.

```
root@cli-vm:~# kubectl get all
```

Test Output

Application image is successfully deployed (see Test script step 6).

```
root@cli-vm:~# kubectl get all
```

```
NAME                                READY        STATUS      RESTARTS   AGE
pod/your-app-865974f8d5-xt2vp       1/1          Running     0           2m

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
service/your-app        ClusterIP   10.100.200.141   <none>        80/TCP      2m
```

Mandatory	

```
service/kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP    3d

NAME                           DESIRED  CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/your-app       1        1         1            1           2m

NAME                                    DESIRED   CURRENT  READY     AGE
replicaset.apps/your-app-854d7dcd67     1         1        1         2m
```

PKS.LCM.App.Uninstall

This test verifies that the Application image can be deleted.

Note This test is mandatory.

Mandatory	
Test ID	PKS.LCM.App.Uninstall
Configuration	Run the PKS.LCM.App.Install test.

Mandatory	

Test Script	1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.
	2 Verify existing Application deployment.
	`root@cli-vm:~# kubectl get all`
	3 Delete deployment.
	`root@cli-vm~# kubectl delete deployment.apps/your-app`
	`deployment.apps "your-app" deleted`
	4 Verify that your Application deployment is deleted.
	`root@cli-vm~# kubectl get all`

Test Output	1 Existing application deployment is verified (see Test script step 2).
	`root@cli-vm:~# kubectl get all`
	`NAME READY STATUS RESTARTS AGE`
	`pod/your-app-865974f8d5-xt2vp 1/1 Running 0 1h`
	`NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE`
	`service/your-app ClusterIP 10.100.200.141 <none> 80/TCP 1h`
	`service/kubernetes ClusterIP 10.100.200.1 <none> 443/TCP 3d`
	`NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE`
	`deployment.apps/your-app 1 1 1 1 1h`
	`NAME DESIRED CURRENT READY AGE`
	`replicaset.apps/your-app-854d7dcd67 1 1 1 2m`
	2 Application deployment is successfully deleted (see Test script step 4).
	`root@cli-vm~# kubectl get all`
	`NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE`
	`service/kubernetes ClusterIP 10.100.200.1 <none> 443/TCP 3d`

Kubernetes Life Cycle Tests

This group has the following test cases:

- K8s.Deployment.Create
- K8s.Deployment.Delete.Deploy
- K8s.Service.Deploy
- K8s.Service.Delete.Expose
- K8s.ReplicaSet.Scale

- K8s.Pod.Delete

K8s.Deployment.Create

This test verifies that the Application can be deployed with "*kind: Deployment*" specified in the YAML file.

Note This test is mandatory.

Mandatory	
Test ID	K8s.Deployment.Create
Configuration	NA

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Log in to PKS.

```
root@cli-vm:~# pks login -a uaa.corp.local -u vmware -p VMware1! -k
```

```
API Endpoint: uaa.corp.local
User: vmware
```

3 Create a cluster - *test-cluster* (see Accessing TKGI Environment for details) or use the test-cluster that you created in PKS.Security.SignedImage.Cluster.

```
root@cli-vm:~# pks create-cluster test-cluster --externall-hostname ext-test-cluster.corp.local --plan small --num-nodes 3
```

Note The number of nodes and plan configuration might differ based on your application requirements.

You can skip this step if you have already created a PKS cluster.

4 Configure *test-cluster* to use Harbor.

```
root@cli-vm:~# pks cluster test-cluster
```

- Check output for Kubernetes Master IP(s).
- Edit the /etc/hosts file and add `Kubernetes_Master_IP(s) ext-test-cluster.corp.local` line at the end of the file.

```
root@cli-vm:~# vi /etc/hosts
```

- Get credentials for the cluster - *test-cluster*.

```
root@cli-vm:~# pks get-credentials test-cluster
```

```
Fetching credentials for cluster test-cluster.
Context set for cluster test-cluster.
User can now switch between clusters by using:
               $kubectl config use-context <cluster-name>
```

```
root@cli-vm:~# kubectl config use-context test-cluster
```

```
Switched to context "test-cluster".
```

5 Clone the Application Git repository (Google sample images from K8s are used in examples).

```
root@cli-vm:~# git clone https://github.com/kubernetes/examples.git
```

```
Cloning into 'examples'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 11507 (delta 5), reused 15 (delta 3), pack-reused 11482
Receiving objects: 100% (11507/11507), 16.94 MiB | 8.21 MiB/s, done.
Resolving deltas: 100% (6131/6131), done.
Checking connectivity... done.
```

6 Deploy your Application using the YAML file with "*kind: Deployment*" specified.

```
root@cli-vm:~# cat examples/guestbook/frontend-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
```

Mandatory

```
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

```
root@cli-vm:~# kubectl apply -f examples/guestbook/frontend-deployment.yaml
```

```
deployment.apps/frontend created
```

7 Verify the deployment.

```
root@cli-vm:~# kubectl get all
```

Note Use '`kubectl get all --all-namespaces`', if the Application uses its own namespace.

Test Output

Application is successfully deployed deployment is shown in output as available within two minutes (see Test script step 7).

```
root@cli-vm:~# kubectl get all
```

```
NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     3         3         3            3           2m
```

K8s.Deployment.Delete.Deploy

This test verifies that the Application deployment can be deleted and redeployed.

Note This test is mandatory.

Mandatory

Test ID	K8s.Deployment.Delete.Deploy
Configuration	Run the K8s.Deployment.Create test.

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Verify that the Application deployment is running (Google sample images deployment from K8s are used as examples).

```
root@cli-vm:~# kubectl get all
```

Note Use '`kubectl get all --all-namespaces`', if the Application uses its own namespace.

3 Delete the deployment.

```
root@cli-vm:~# kubectl delete deployment.apps/frontend
```

```
deployment.apps "frontend" deleted
```

4 Verify that the Application deployment is deleted.

```
root@cli-vm:~# kubectl get all
```

Note Use '`kubectl get all --all-namespaces`', if the Application uses its own namespace.

5 Redeploy your Application using the YAML file with "*kind: Deployment*" specified (K8S examples `frontend-deployment.yaml` is used for Google sample images example).

```
root@cli-vm:~# cat examples/guestbook/frontend-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

```
root@cli-vm:~# kubectl apply -f examples/guestbook/frontend-deployment.yaml
```

```
deployment.apps/frontend created
```

6 Verify the deployment.

Mandatory

```
root@cli-vm:~# kubectl get all
```

Note Use '`kubectl get all --all-namespaces`', if the Application uses its own namespace.

Test Output

1 Application deployment is running - *frontend* is shown in the output (see Test script step 2).

```
root@cli-vm:~# kubectl get all
```

```
NAME                            READY     STATUS    RESTARTS   AGE
pod/frontend-56f7975f44-77svf   1/1       Running   0          54m
pod/frontend-56f7975f44-9lwxw   1/1       Running   0          54m
pod/frontend-56f7975f44-qfhsc   1/1       Running   0          54m

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d

NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     3         3         3            3           54m

NAME                                    DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-56f7975f44     3         3         3         54m
```

2 Application deployment is deleted (see Test script step 4).

```
root@cli-vm:~# kubectl get all
```

```
NAME                            READY     STATUS        RESTARTS   AGE
pod/frontend-56f7975f44-77svf   0/1       Terminating   0          54m
pod/frontend-56f7975f44-qfhsc   0/1       Terminating   0          54m

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d

root@cli-vm:~# kubectl get all
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d
```

3 Application is deployed successfully – *deployment* is shown in the output as available within two minutes (see Test script step 6).

```
root@cli-vm:~# kubectl get all
```

```
NAME                            READY     STATUS    RESTARTS   AGE
pod/frontend-56f7975f44-2tbtc   1/1       Running   0          18s
pod/frontend-56f7975f44-ft4kn   1/1       Running   0          18s
pod/frontend-56f7975f44-s8wq5   1/1       Running   0          18s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d

NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     3         3         3            3           19s

NAME                                    DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-56f7975f44     3         3         3         19s
```

K8s.Service.Deploy

This test verifies that the Application service can be deployed with "kind: Service" specified in the YAML file.

Note This test is mandatory.

Mandatory	
Test ID	K8s.Service.Deploy
Configuration	NA

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Log in to PKS.

```
root@cli-vm:~# pks login -a uaa.corp.local -u vmware -p VMware1! -k
```

```
API Endpoint:uaa.corp.local
User:vmware
```

3 Create a cluster - *test-cluster* (see Accessing TKGI Environment for details) or use the test-cluster that you created in PKS.Security.SignedImage.Cluster.

```
root@cli-vm:~# pks create-cluster test-cluster --external-hostname ext-test-cluster.corp.local --plan small --num-nodes 3
```

Note The number of nodes and plan configuration might differ based on your application requirements.

You can skip this step if you have already created a PKS cluster.

4 Configure *test-cluster* to use Harbor.

```
root@cli-vm:~# pks cluster test-cluster
```

- Check output for Kubernetes Master IP(s).
- Edit the /etc/hosts file and add `Kubernetes_Master_IP(s) ext-test-cluster.corp.local` line at the end of the file.

```
root@cli-vm:~# vi /etc/hosts
```

- Get credentials for the cluster - *test-cluster*.

```
root@cli-vm:~# pks get-credentials test-cluster
```

```
Fetching credentials for cluster test-cluster.
Context set for cluster test-cluster.
User can now switch between clusters by using:
               $kubectl config use-context <cluster-name>
```

```
root@cli-vm:~# kubectl config use-context test-cluster
```

```
Switched to context "test-cluster".
```

5 Clone the Application Git repository (Google sample images from K8s are used in examples).

```
root@cli-vm:~# git clone https://github.com/kubernetes/examples.git
```

```
Cloning into 'examples'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 11507 (delta 5), reused 15 (delta 3), pack-reused 11482
Receiving objects: 100% (11507/11507), 16.94 MiB | 8.21 MiB/s, done.
Resolving deltas: 100% (6131/6131), done.
Checking connectivity... done.
```

6 Deploy your Application using the YAML file.

```
root@cli-vm:~# cat service-frontend.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
```

Mandatory	
	labels: app: guestbook tier: frontend spec: # type: LoadBalancer ports: - port: 80 selector: app: guestbook tier: frontend
	root@cli-vm:~# kubectl apply -f ./service-frontend.yaml
	service/frontend createde3c94b6f59a1
	7 Verify that the service is running. root@cli-vm:~# kubectl get svc **Note** Use 'kubectl get svc --all-namespaces', if the Application uses its own namespace.
	8 Verify that the service can be accessed. (This step depends on the application.)
Test Output	1 Service is running - *frontend* is shown in the output (see Test script step 7). root@cli-vm:~# kubectl get svc

```
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
frontend       ClusterIP   10.100.200.98    <none>        80/TCP     17s
kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP    8d
```

	2 Service can be accessed. (This step depends on the application.)

K8s.Service.Delete.Expose

This test verifies that the K8s service can be deleted and redeployed.

Note This test is mandatory.

Mandatory	
Test ID	K8s.Service.Delete.Expose
Configuration	Run the K8s.Service.Deploy test.

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Verify that the previously deployed service is running (Google sample images are used as examples).

```
root@cli-vm:~# kubectl get svc
```

Note Use '`kubectl get svc --all-namespaces`', if the Application uses its own namespace.

3 Delete the service.

```
root@cli-vm:~# kubectl delete service/frontend
```

```
service "frontend" deleted
```

4 Verify that the service is deleted.

```
root@cli-vm:~# kubectl get svc
```

Note Use '`kubectl get svc --all-namespaces`', if the Application uses its own namespace.

5 Expose the service using the YAML file (example: service for Google sample images).

```
root@cli-vm:~# cat service-frontend.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend

root@cli-vm:~# kubectl expose -f ./service-frontend.yaml
service/frontend exposed
```

6 Verify that the service is running.

```
root@cli-vm:~# kubectl get svc
```

Note Use '`kubectl get svc --all-namespaces`', if the Application uses its own namespace.

7 Verify that the service can be accessed. (This step depends on the application.)

Test Output

1 Service is running - *frontend* is shown in the output (see Test script step 2).

```
root@cli-vm:~# kubectl get svc
```

```
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
frontend       ClusterIP   10.100.200.6     <none>        80/TCP     7d
kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP    8d
redis-master   ClusterIP   10.100.200.101   <none>        6379/TCP   7d
redis-slave    ClusterIP   10.100.200.194   <none>        6379/TCP   7d
```

2 Service is deleted - *frontend* is not shown in the output (see Test script step 4).

Mandatory	

```
root@cli-vm:~# kubectl get svc
```

```
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP    8d
redis-master   ClusterIP   10.100.200.101   <none>        6379/TCP   7d
redis-slave    ClusterIP   10.100.200.194   <none>        6379/TCP   7d
```

3 Service is running – *frontend* is shown in the output (see Test script step 6).

```
root@cli-vm:~# kubectl get svc
```

```
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
frontend       ClusterIP   10.100.200.98    <none>        80/TCP     17s
kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP    8d
redis-master   ClusterIP   10.100.200.101   <none>        6379/TCP   7d
redis-slave    ClusterIP   10.100.200.194   <none>        6379/TCP   7d
```

4 Service can be accessed. (This step depends on the application.)

K8s.ReplicaSet.Scale

This test verifies that the Application deployment can be scaled using ReplicaSets.

Note This test is mandatory.

Mandatory	
Test ID	K8s.ReplicaSet.Create.Scale
Configuration	Run the K8s.Deployment.Create test.

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Verify that the Application deployment is running (Google sample images deployment from K8s examples are used) .

```
root@cli-vm:~# kubectl get all
```

3 Scale your Application deployment.

```
root@cli-vm:~# kubectl scale --replicas=5 deployment.apps/frontend
```

```
deployment.apps/frontend scaled
```

4 Verify that the Application deployment is scaled.

```
root@cli-vm:~# kubectl get all
```

5 Rescale your Application deployment.

```
root@cli-vm:~# kubectl scale --replicas=3 deployment.apps/frontend
```

```
deployment.apps/frontend scaled
```

6 Verify that the Application deployment is rescaled.

```
root@cli-vm:~# kubectl get all
```

Test Output

1 Application deployment is running – *frontend* is shown in the output (see Test script step 2).

```
root@cli-vm:~# kubectl get all
```

```
NAME                            READY     STATUS    RESTARTS   AGE
pod/frontend-56f7975f44-7v65j   1/1       Running   0          13s
pod/frontend-56f7975f44-ktkgk   1/1       Running   0          13s
pod/frontend-56f7975f44-v9s5k   1/1       Running   0          13s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d

NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     3         3         3            3           13s

NAME                                    DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-56f7975f44     3         3         3         13s
```

2 Application deployment is successfully scaled (see Test script step 4).

```
root@cli-vm:~# kubectl get all
```

```
NAME                            READY     STATUS    RESTARTS   AGE
pod/frontend-56f7975f44-7v65j   1/1       Running   0          48s
pod/frontend-56f7975f44-j8llt   1/1       Running   0          5s
pod/frontend-56f7975f44-ktkgk   1/1       Running   0          48s
pod/frontend-56f7975f44-m92d6   1/1       Running   0          5s
pod/frontend-56f7975f44-v9s5k   1/1       Running   0          48s
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d
NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     5         5         5            5           48s
NAME                                    DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-56f7975f44     5         5         5         48s
```

3 Application deployment is successfully rescaled (see Test script step 6).

Mandatory	

```
root@cli-vm:~# kubectl get all
```

```
NAME                                 READY     STATUS        RESTARTS   AGE
pod/frontend-56f7975f44-7v65j        1/1       Running       0          1m
pod/frontend-56f7975f44-j8llt        0/1       Terminating   0          33s
pod/frontend-56f7975f44-ktkgk        1/1       Running       0          1m
pod/frontend-56f7975f44-m92d6        0/1       Terminating   0          33s
pod/frontend-56f7975f44-v9s5k        1/1       Running       0          1m
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d
NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     3         3         3            3           1m
NAME                                    DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-56f7975f44     3         3         3         1m
```

K8s.Pod.Delete

This test verifies that K8s pod can be automatically recreated after deletion.

Note This test is mandatory.

Mandatory	
Test ID	K8s.Pod.Delete
Configuration	Run the K8s.Deployment.Create test.

Mandatory

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Verify that the Application deployment is running (Google sample images deployment from K8s examples are used).

```
root@cli-vm:~# kubectl get all
```

3 Delete pod.

```
root@cli-vm:~# kubectl delete pod/frontend-56f7975f44-7v65j
```

```
pod "frontend-56f7975f44-7v65j" deleted
```

4 Verify that the pod is successfully recreated.

```
root@cli-vm:~# kubectl get pods
```

5 Delete another pod.

```
root@cli-vm:~# kubectl delete pod frontend-56f7975f44-ktkgk
```

```
pod "frontend-56f7975f44-ktkgk" deleted
```

6 Verify that the pod is successfully recreated. Also verify that deployments and replicasets are available, and ready.

```
root@cli-vm:~# kubectl get all
```

Test Output

1 Application deployment is running – *frontend* is shown in the output (see Test script step 2).

```
root@cli-vm:~# kubectl get all
```

```
NAME                              READY     STATUS    RESTARTS   AGE
pod/frontend-56f7975f44-7v65j     1/1       Running   0          42m
pod/frontend-56f7975f44-ktkgk     1/1       Running   0          42m
pod/frontend-56f7975f44-v9s5k     1/1       Running   0          42m
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d
NAME                         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend     3         3         3            3           42m
NAME                                    DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-56f7975f44     3         3         3         42m
```

2 The pod is successfully recreated after deletion (see Test script step 4).

```
root@cli-vm:~# kubectl get pods
```

```
NAME                        READY     STATUS    RESTARTS   AGE
frontend-56f7975f44-bxspz   1/1       Running   0          54s
frontend-56f7975f44-ktkgk   1/1       Running   0          44m
frontend-56f7975f44-v9s5k   1/1       Running   0          44m
```

3 The pod is successfully recreated. Deployments and replicasets are available, and ready (see Test script step 6).

```
root@cli-vm:~# kubectl get all
```

```
NAME                              READY     STATUS    RESTARTS   AGE
pod/frontend-56f7975f44-bxspz     1/1       Running   0          1m
pod/frontend-56f7975f44-dwkvf     1/1       Running   0          16s
pod/frontend-56f7975f44-v9s5k     1/1       Running   0          45m

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.200.1   <none>        443/TCP   3d
```

Mandatory	

	`NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE`
	`deployment.apps/frontend 3 3 3 3 45m`
	
	`NAME DESIRED CURRENT READY AGE`
	`replicaset.apps/frontend-56f7975f44 3 3 3 45m`

Partner Validation

6

This test area focuses on partner validation.

This chapter includes the following topics:

Functionality Tests

The Functionality Tests group has the following test case:

App.Functionality (Partner Validation)

This test ensures the container functionality in the TKGI environment.

Note This test is mandatory.

Mandatory	
Test ID	App.Functionality (Partner Validation)
Configuration	PKS cluster must be created.

Mandatory	
Test Script	After deployment, the Application is powered on and configured. Perform an initial sanity test by logging in to the solution and validating its functionality. Ensure that the IP address and host names are configured appropriately. Before running the VMware Validation Tests, you must run a set of tests that are specific to your Application. This ensures that your solution operates normally before running the VMware Validation Tests. **Partner Validation Test Scenario Examples:** Example Scenario 1: Deploy a multi-tier application with a frontend, a database, and a load balancer. For details, see Load Balancers. ■ When the deployment "`app-name/deployment.yaml`" is applied: ■ Current context is set to the cluster "*test-cluster*" and namespace "*app-name*". ■ Deployment *"app-name*" is rolled out within two minutes. ■ Application is reachable at "*http://24.24.24.x*" through the load-balancer "*app-name-loadbalancer*". ■ When the deployment "*app-name/deployment.yaml*" is deleted, the namespace "*app-name*" does not exist. Example Scenario 2: Deleting and recreating workload. ■ When the deployment "`app-name/deployment.yaml`" is applied: ■ Current context is set to the cluster "*test-cluster*" and namespace "*app-name*". ■ Deployment *"app-name*" is resized to five pods. ■ Deployment *"app-name*" is rolled out within two minutes. ■ Application is reachable at "*http://24.24.24.x*" through the load-balancer "*app-name-loadbalancer*". ■ When the deployment "*app-name/deployment.yaml*" is deleted, the namespace "*app-name*" does not exist. ■ When the deployment "`app-name/deployment.yaml`" is applied: ■ Current context is set to the cluster "*test-cluster*" and namespace "*app-name*". ■ Deployment *"app-name*" is resized to five pods. ■ Deployment *"app-name*" is rolled out within two minutes. ■ Application is reachable at "*http://24.24.24.x*" through the load-balancer "*app-name-loadbalancer*". ■ When the deployment "*app-name/deployment.yaml*" is deleted, the namespace "*app-name*" does not exist.
Test Output	Application operates normally before running the VMware Validation Tests.

Note Ensure that the Application solution is deployed and running in a good state after the validation tests are done. VMware might need to verify the Application solution deployment during the validation results review.

TKGI Centric Platform Validation 7

This test area focuses on the cluster K8s operations when the Application is running.

This chapter includes the following topics:

Resilience/Negative Tests

The Resilience/Negative Tests group has the following test cases:

- PKS.Resilience.ClusterExpand
- PKS.Worker.Reboot
- PKS.Master.Reboot
- PKS.Worker.PowerOffOn
- PKS.Master.PowerOffOn

PKS.Resilience.ClusterExpand

This test verifies that the Application can be expanded with the growing K8s resources.

Note This test is optional.

Optional	
Test ID	PKS.Resilience.ClusterExpand
Configuration	PKS cluster is created. Application is running.

Optional

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Check the cluster existence.

```
root@cli-vm:~# pks clusters
```

```
Name               Plan Name            UUID                          Status
Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded
CREATE
```

3 Get credentials for the cluster.

```
root@cli-vm:~# pks get-credentials k8s-cluster-101
```

```
Fetching credentials for cluster k8s-cluster-101.
Context set for cluster k8s-cluster-101.
```

You can now switch between clusters by using:

```
$kubectl config use-context <cluster-name>
```

4 Set `kubectl` to use the context of the cluster.

```
root@cli-vm:~# kubectl config use-context k8s-cluster-101
```

```
Switched to context "k8s-cluster-101".
```

5 Check the cluster nodes status.

```
root@cli-vm:~# kubectl get nodes -o wide
```

```
NAME                                   STATUS    ROLES     AGE  VERSION
INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION
CONTAINER-RUNTIME
afc7bfbd-bea7-4044-8d81-2ed6121eec78   Ready     <none>    25d  v1.11.2
172.23.0.4    172.23.0.4    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
cd48fdbc-5f15-437c-9a1b-f70235c59a62   Ready     <none>    25d  v1.11.2
172.23.0.3    172.23.0.3    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
dc8ca134-93bb-422a-89d0-89a2ba359685   Ready     <none>    25d  v1.11.2
172.23.0.5    172.23.0.5    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
```

6 Resize the cluster by increasing the number of worker nodes ('--num-nodes 5' is only an example, it depends on the initial number of worker nodes in the cluster).

```
root@cli-vm:~# pks resize k8s-cluster-101 --num-nodes 5

Are you sure you want to resize cluster k8s-cluster-101 to 5? (y/n): y
```

Use '*pks cluster k8s-cluster-101*' to monitor the state of your cluster.

```
root@cli-vm:~# pks cluster k8s-cluster-101
```

```
Name:                     k8s-cluster-101
Plan Name:                small
UUID:                     53187740-e5cf-4ed7-9181-ab71bbdc1945
Last Action:              UPDATE
Last Action State:        in progress
Last Action Description:  Instance update in progress
Kubernetes Master Host:   pks-cluster-101.corp.local
```

Optional

```
Kubernetes Master Port:   8443
Worker Nodes:             5
Kubernetes Master IP(s):  24.24.24.10
Network Profile Name:
```

7 Monitor cluster resizing process by running this command:

```
root@cli-vm:~# pks cluster k8s-cluster-101
```

8 Verify that the cluster is successfully resized and number of workers is increased.

```
root@cli-vm:~# pks clusters
```

```
root@cli-vm:~# pks cluster k8s-cluster-101
```

```
root@cli-vm:~# kubectl get nodes -o wide
```

Test Output

Cluster is successfully resized and number of workers is increased (see Test script step 8).

```
root@cli-vm:~# pks clusters
```

```
Name               Plan Name  UUID                                  Status
Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded
UPDATE
```

```
root@cli-vm:~# pks cluster k8s-cluster-101
```

```
Name:                     k8s-cluster-101
Plan Name:                small
UUID:                     53187740-e5cf-4ed7-9181-ab71bbdc1945
Last Action:              UPDATE
Last Action State:        succeeded
Last Action Description:  Instance update completed
Kubernetes Master Host:   pks-cluster-101.corp.local
Kubernetes Master Port:   8443
Worker Nodes:             5
Kubernetes Master IP(s):  24.24.24.10
Network Profile Name:
```

```
root@cli-vm:~# kubectl get nodes -o wide
```

```
NAME                                   STATUS    ROLES     AGE   VERSION
INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-
RUNTIME
40771b7c-9728-4c35-ba88-5407295b45ab   Ready     <none>    16m   v1.11.2
172.23.0.8    172.23.0.8    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
afc7bfbd-bea7-4044-8d81-2ed6121eec78   Ready     <none>    26d   v1.11.2
172.23.0.4    172.23.0.4    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
cd48fdbc-5f15-437c-9a1b-f70235c59a62   Ready     <none>    26d   v1.11.2
172.23.0.3    172.23.0.3    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
dc8ca134-93bb-422a-89d0-89a2ba359685   Ready     <none>    26d   v1.11.2
172.23.0.5    172.23.0.5    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
f1b26c6e-3ae8-4e46-aae6-ec174f3670b0   Ready     <none>    23m   v1.11.2
172.23.0.7    172.23.0.7    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://
17.12.1-ce
```

PKS.Worker.Reboot

This test verifies that the running K8s cluster worker node can be rebooted when the Stateless application is running.

Note This test is mandatory for Stateless applications only.

Mandatory for Stateless Applications Only	
Test ID	PKS.Worker.Reboot
Configuration	PKS cluster is created. Application is running.

Mandatory for Stateless Applications Only

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Check the cluster existence.

```
root@cli-vm:~# pks clusters
```

```
Name               Plan Name            UUID                          Status
Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded
CREATE
```

3 Get credentials for the cluster.

```
root@cli-vm:~# pks get-credentials k8s-cluster-101
```

```
Fetching credentials for cluster k8s-cluster-101.
Context set for cluster k8s-cluster-101.
```

You can now switch between clusters by using:

```
$kubectl config use-context <cluster-name>
```

4 Set `kubectl` to use the context of the cluster.

```
root@cli-vm:~# kubectl config use-context k8s-cluster-101
```

```
Switched to context "k8s-cluster-101".
```

5 Check the cluster nodes status (cluster with six worker nodes for Google sample images is used as application example).

```
root@cli-vm:~# kubectl get nodes -o wide
```

```
NAME                                   STATUS    ROLES     AGE
VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-
VERSION      CONTAINER-RUNTIME
0cc55cd5-ba68-4647-aa30-f0974ee218a5   Ready     <none>    2d
v1.11.6   172.23.2.7    172.23.2.7    Ubuntu 16.04.5 LTS   4.15.0-42-
generic   docker://17.12.1-ce
89f41686-aa51-42b8-8555-7f098e7d7d3e   Ready     <none>    2d
v1.11.6   172.23.2.8    172.23.2.8    Ubuntu 16.04.5 LTS   4.15.0-42-
generic   docker://17.12.1-ce
8bd0e317-fde8-4455-9e2a-d11144bce7db   Ready     <none>    2d
v1.11.6   172.23.2.6    172.23.2.6    Ubuntu 16.04.5 LTS   4.15.0-42-
generic   docker://17.12.1-ce
97a40e95-d49d-420a-8ff2-1f15fb02b247   Ready     <none>    2d
v1.11.6   172.23.2.5    172.23.2.5    Ubuntu 16.04.5 LTS   4.15.0-42-
generic   docker://17.12.1-ce
c3b10035-90e0-4077-a841-38c0db885bdb   Ready     <none>    2d
v1.11.6   172.23.2.4    172.23.2.4    Ubuntu 16.04.5 LTS   4.15.0-42-
generic   docker://17.12.1-ce
e5e041ed-ef25-4ea7-8dc6-9abad079e8f9   Ready     <none>    2d
v1.11.6   172.23.2.3    172.23.2.3    Ubuntu 16.04.5 LTS   4.15.0-42-
generic   docker://17.12.1-ce
```

6 Check pods status.

Mandatory for Stateless Applications Only

```
root@cli-vm:~# kubectl get pods
```

```
NAME                              READY     STATUS    RESTARTS   AGE
frontend-7776b5c947-7clrm         1/1       Running   0          6m
frontend-7776b5c947-96jlh         1/1       Running   0          6m
frontend-7776b5c947-xsrv7         1/1       Running   0          6m
redis-master-5c95c89d9-78ds4      1/1       Running   0          6m
redis-slave-6548b4f8b9-5p92d      1/1       Running   0          6m
redis-slave-6548b4f8b9-x9fd9      1/1       Running   0          6m
```

7 Get the pod IP for worker node under reboot test.

```
root@cli-vm:~# kubectl describe pod frontend-7776b5c947-7clrm | grep IP
```

```
IP:                172.16.7.7
```

8 Get the VM name for the worker node under the reboot test.

```
root@cli-vm:~# source A-BOSH.env

root@cli-vm:~# bosh vms
```

```
…
Deployment 'service-instance_a2a02b92-0ac5-4bfd-9706-fcfeae528d2b'

Instance                                     Process State  AZ
IPs         VM CID                                       VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  running        AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
worker/32ea5716-26fe-47cd-966d-b915af7c854f  running        AZ-COMP
172.23.2.4  vm-4506fbdd-b021-46fe-95ba-eb8197c22482  medium.disk  true
worker/36d21afb-ec09-4c14-8a9c-7e43aac9b375  running        AZ-COMP
172.23.2.5  vm-ad72851e-df0c-4184-a6fe-53d7743242c2  medium.disk  true
worker/9835f1fc-087e-4df9-af50-ff535e42c0c2  running        AZ-COMP
172.23.2.3  vm-edc446f3-3827-424e-b9b7-3136f3af75a4  medium.disk  true
worker/9face359-f852-427f-9faa-649e3f7b7cb5  running        AZ-COMP
172.23.2.6  vm-79f67e08-72c1-46aa-9032-03fd47a0ff2f  medium.disk  true
worker/e218da9d-41d6-4098-a560-56669415b66e  starting       AZ-COMP
172.23.2.7  vm-843f9e36-dcee-4b69-929b-a5355e3bec74  medium.disk  true
worker/f10418ab-3362-4a7d-9968-a6a757a45c78  running        AZ-COMP
172.23.2.8  vm-9163ea29-97c1-494c-b796-fc8a35ef8241  medium.disk  true
```

Pods IP block has CIDR 172.16.0.0/16 defined by NSX-T setup.

Nodes IP block has CIDR 172.23.0.0/16 defined by NSX-T setup.

Pod with IP 172.16.7.7 corresponds to node with IP 172.23.2.7.

vm-843f9e36-dcee-4b69-929b-a5355e3bec74 is VM under test.

9 Access vSphere Client. See Accessing vSphere Client for details.

10 Highlight `vm-843f9e36-dcee-4b69-929b-a5355e3bec74` under `RegionA01-COMP01/PKS-COMP` cluster and select **Power** > **Reset** from the context menu.

11 Go back to the PuTTY session to CLI-VM and monitor the rebooted node status by running the `bosh vms` command. Wait until **Process State** is shown in green as **Running** (it may take up to five minutes).

12 Verify the Application deployment readiness.

```
root@cli-vm:~# kubectl get all
```

Mandatory for Stateless Applications Only

13 Repeat steps 6–12 for each worker node.

Test Output

1 Worker node is successfully rebooted and has **Process State** as **Running** (see Test script step 11). The Process State can be *unresponsive agent*, *starting*, or *failing* during reboot.

```
root@cli-vm:~# bosh vms
```

```
Instance                                     Process State         AZ
IPs         VM CID                           VM Type      Active
worker/e218da9d-41d6-4098-a560-56669415b66e  'unresponsive agent'   AZ-COMP
172.23.2.7  vm-843f9e36-dcee-4b69-929b-a5355e3bec74  medium.disk  true
```

```
root@cli-vm:~# bosh vms
```

```
Instance                                     Process State  AZ
IPs         VM CID                                  VM Type      Active
worker/e218da9d-41d6-4098-a560-56669415b66e  starting       AZ-COMP
172.23.2.7  vm-843f9e36-dcee-4b69-929b-a5355e3bec74  medium.disk  true
```

```
root@cli-vm:~# bosh vms
```

```
Instance                                     Process State  AZ
IPs         VM CID                                  VM Type      Active
worker/e218da9d-41d6-4098-a560-56669415b66e  failing        AZ-COMP
172.23.2.7  vm-843f9e36-dcee-4b69-929b-a5355e3bec74  medium.disk  true
```

```
root@cli-vm:~# bosh vms
```

```
worker/e218da9d-41d6-4098-a560-56669415b66e  running        AZ-COMP
172.23.2.7  vm-843f9e36-dcee-4b69-929b-a5355e3bec74  medium.disk  true
```

2 Application deployment is running and ready.

```
root@cli-vm:~# kubectl get all
```

```
NAME                              READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm     1/1       Running   1          12m
pod/frontend-7776b5c947-96jlh     1/1       Running   0          12m
pod/frontend-7776b5c947-xsrv7     1/1       Running   0          12m
pod/redis-master-5c95c89d9-78ds4  1/1       Running   0          12m
pod/redis-slave-6548b4f8b9-5p92d  1/1       Running   0          12m
pod/redis-slave-6548b4f8b9-x9fd9  1/1       Running   0          12m
```

```
NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend      ClusterIP   10.100.200.152   <none>        80/TCP
12m
service/kubernetes    ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master  ClusterIP   10.100.200.55    <none>        6379/TCP
12m
service/redis-slave   ClusterIP   10.100.200.194   <none>        6379/TCP
12m
```

```
NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend      3         3         3            3
12m
```

Mandatory for Stateless Applications Only

```
deployment.apps/redis-master   1         1         1            1
12m
deployment.apps/redis-slave    2         2         2            2
12m
```

```
NAME                                  DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-7776b5c947   3         3         3         12m
replicaset.apps/redis-master-5c95c89d9   1         1         1         12m
replicaset.apps/redis-slave-6548b4f8b9   2         2         2         12m
```

3 Every worker node is successfully rebooted and has **Process State** as **Running** (see Test script step 13).

PKS.Master.Reboot

This test verifies that the running K8s cluster master node can be rebooted when the Stateless application is running.

Note This test is mandatory for Stateless applications only.

Mandatory for Stateless Applications Only	
Test ID	PKS.Master.Reboot
Configuration	PKS cluster is created. Application is running.

Mandatory for Stateless Applications Only

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Check the cluster existence.

```
root@cli-vm:~# pks clusters
```

```
Name               Plan Name            UUID                          Status
Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded
CREATE
```

3 Get credentials for the cluster.

```
root@cli-vm:~# pks get-credentials k8s-cluster-101
```

```
Fetching credentials for cluster k8s-cluster-101.
Context set for cluster k8s-cluster-101.
```

You can now switch between clusters by using:

```
$kubectl config use-context <cluster-name>
```

4 Set `kubectl` to use the context of the cluster.

```
root@cli-vm:~# kubectl config use-context k8s-cluster-101
```

```
Switched to context "k8s-cluster-101".
```

5 Get the VM name for the master node under reboot test.

```
root@cli-vm:~# source A-BOSH.env
```

```
root@cli-vm:~# bosh vms
```

```
Deployment 'service-instance_a2a02b92-0ac5-4bfd-9706-fcfeae528d2b'

Instance                                     Process State  AZ
IPs         VM CID                                   VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  running        AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
```

6 Access vSphere Client. See Accessing vSphere Client for details.

7 Highlight `vm-04c898ea-6614-4b08-a22a-c50bed645293` under `RegionA01-COMP01/PKS-COMP` cluster and select **Power** > **Reset** from the context menu.

8 Go back to the PuTTY session to CLI-VM and monitor the rebooted master node status by running the `bosh vms` command. Wait until **Process State** is shown in green as **Running** (it might take up to five minutes).

9 Verify the Application deployment readiness.

```
root@cli-vm:~# kubectl get all
```

Test Output

1 Master node is successfully rebooted and has **Process State** as **Running** (see Test script step 8). The Process State can be *unresponsive agent*, *starting*, or *failing* during reboot.

```
root@cli-vm:~# bosh vms
```

```
Instance                                     Process State  AZ
IPs         VM CID                                   VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  'unresponsive agent'       AZ-
COMP  172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
```

Mandatory for Stateless Applications Only

```
root@cli-vm:~# bosh vms
```

```
Instance                                     Process State  AZ
IPs         VM CID                                  VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  starting       AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
```

```
root@cli-vm:~# bosh vms
```

```
Instance                                     Process State  AZ
IPs         VM CID                                  VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  failing      AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
```

```
root@cli-vm:~# bosh vms
```

```
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  running     AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
```

2 Application deployment is running and ready (see Test script step 9).

```
root@cli-vm:~# kubectl get all
```

```
NAME                                READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm       1/1       Running   1          12m
pod/frontend-7776b5c947-96jlh       1/1       Running   0          12m
pod/frontend-7776b5c947-xsrv7       1/1       Running   0          12m
pod/redis-master-5c95c89d9-78ds4    1/1       Running   0          12m
pod/redis-slave-6548b4f8b9-5p92d    1/1       Running   0          12m
pod/redis-slave-6548b4f8b9-x9fd9    1/1       Running   0          12m
```

```
NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend       ClusterIP   10.100.200.152   <none>        80/TCP
12m
service/kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master   ClusterIP   10.100.200.55    <none>        6379/TCP
12m
service/redis-slave    ClusterIP   10.100.200.194   <none>        6379/TCP
12m
```

```
NAME                           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend       3         3         3            3
12m
deployment.apps/redis-master   1         1         1            1
12m
deployment.apps/redis-slave    2         2         2            2
12m
```

```
NAME                                      DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-7776b5c947       3         3         3         12m
replicaset.apps/redis-master-5c95c89d9    1         1         1         12m
replicaset.apps/redis-slave-6548b4f8b9    2         2         2         12m
```

PKS.Worker.PowerOffOn

This test verifies that running K8s cluster worker node can rejoin the cluster after it is powered off/on, when the Stateless application is running.

Note This test is mandatory for Stateless applications only.

Mandatory for Stateless Applications Only	
Test ID	PKS.Worker.PowerOffOn
Configuration	PKS cluster is created. Application is running.

Mandatory for Stateless Applications Only

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Check the cluster existence.

```
root@cli-vm:~# pks clusters
```

```
Name               Plan Name            UUID                          Status
Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded
CREATE
```

3 Get credentials for the cluster.

```
root@cli-vm:~# pks get-credentials k8s-cluster-101
```

```
Fetching credentials for cluster k8s-cluster-101.
Context set for cluster k8s-cluster-101.
```

You can now switch between clusters by using:

```
$kubectl config use-context <cluster-name>
```

4 Set `kubectl` to use the context of the cluster.

```
root@cli-vm:~# kubectl config use-context k8s-cluster-101
```

```
Switched to context "k8s-cluster-101".
```

5 Verify the Application deployment readiness.

```
root@cli-vm:~# kubectl get all
```

6 Check the pod status.

```
root@cli-vm:~# kubectl get pods
```

```
NAME                            READY     STATUS    RESTARTS   AGE
frontend-7776b5c947-7clrm       1/1       Running   0          6m
frontend-7776b5c947-96jlh       1/1       Running   0          6m
frontend-7776b5c947-xsrv7       1/1       Running   0          6m
redis-master-5c95c89d9-78ds4    1/1       Running   0          6m
redis-slave-6548b4f8b9-5p92d    1/1       Running   0          6m
redis-slave-6548b4f8b9-x9fd9    1/1       Running   0          6m
```

7 Get the pod IP for worker node under the power off/on test.

```
root@cli-vm:~# kubectl describe pod frontend-7776b5c947-7clrm | grep IP
```

```
IP:                172.16.7.7
```

8 Get the VM name for the worker node under the power off/on test.

```
root@cli-vm:~# source A-BOSH.env
```

```
root@cli-vm:~# bosh vms
```

```
…
Deployment 'service-instance_a2a02b92-0ac5-4bfd-9706-fcfeae528d2b'

Instance                                    Process State  AZ
IPs         VM CID                                  VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  running        AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
worker/32ea5716-26fe-47cd-966d-b915af7c854f  running        AZ-COMP
```

Mandatory for Stateless Applications Only

```
172.23.2.4  vm-4506fbdd-b021-46fe-95ba-eb8197c22482  medium.disk  true
worker/36d21afb-ec09-4c14-8a9c-7e43aac9b375  running        AZ-COMP
172.23.2.5  vm-ad72851e-df0c-4184-a6fe-53d7743242c2  medium.disk  true
worker/9835f1fc-087e-4df9-af50-ff535e42c0c2  running        AZ-COMP
172.23.2.3  vm-edc446f3-3827-424e-b9b7-3136f3af75a4  medium.disk  true
worker/9face359-f852-427f-9faa-649e3f7b7cb5  running        AZ-COMP
172.23.2.6  vm-79f67e08-72c1-46aa-9032-03fd47a0ff2f  medium.disk  true
worker/e218da9d-41d6-4098-a560-56669415b66e  starting       AZ-COMP
172.23.2.7  vm-843f9e36-dcee-4b69-929b-a5355e3bec74  medium.disk  true
worker/f10418ab-3362-4a7d-9968-a6a757a45c78  running        AZ-COMP
172.23.2.8  vm-9163ea29-97c1-494c-b796-fc8a35ef8241  medium.disk  true
```

Pods IP block has CIDR 172.16.0.0/16 defined by NSX-T setup.

Nodes IP block has CIDR 172.23.0.0/16 defined by NSX-T setup.

Pod with IP 172.16.7.7 corresponds to node with IP 172.23.2.7.

vm-843f9e36-dcee-4b69-929b-a5355e3bec74 is VM under test.

9 Access vSphere Client. See Accessing vSphere Client for details.

10 Highlight `vm-843f9e36-dcee-4b69-929b-a5355e3bec74` under `RegionA01-COMP01/PKS-COMP` cluster and select **Power** > **Power Off** from the context menu.

11 Go back to the PuTTY session to CLI-VM and verify the Application deployment readiness.

```
root@cli-vm:~# kubectl get all
```

12 Access vSphere Client.

13 Highlight `vm-843f9e36-dcee-4b69-929b-a5355e3bec74` under `RegionA01-COMP01/PKS-COMP` cluster and select **Power** > **Power On** from the context menu.

14 Go back to the PuTTY session to CLI-VM and monitor the powered on worker node status by running the `kubectl get all` command. Wait until the worker node is shown as **Available** (it might take up to five minutes).

Test Output

1 Application deployment is running and ready (see Test script step 5).

```
root@cli-vm:~# kubectl get all
```

```
NAME                                READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm       1/1       Running   1          1h
pod/frontend-7776b5c947-96jlh       1/1       Running   0          1h
pod/frontend-7776b5c947-xsrv7       1/1       Running   0          1h
pod/redis-master-5c95c89d9-78ds4    1/1       Running   0          1h
pod/redis-slave-6548b4f8b9-5p92d    1/1       Running   0          1h
pod/redis-slave-6548b4f8b9-x9fd9    1/1       Running   0          1h

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend       ClusterIP   10.100.200.152   <none>        80/TCP
1h
service/kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master   ClusterIP   10.100.200.55    <none>        6379/TCP
1h
service/redis-slave    ClusterIP   10.100.200.194   <none>        6379/TCP
1h

NAME                              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend          3         3         3            3
1h
deployment.apps/redis-master      1         1         1            1
```

Mandatory for Stateless Applications Only

```
1h
deployment.apps/redis-slave    2         2         2            2
1h

NAME                                     DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-7776b5c947      3         3         3         1h
replicaset.apps/redis-master-5c95c89d9   1         1         1         1h
replicaset.apps/redis-slave-6548b4f8b9   2         2         2         1h
```

2 Application deployment is running and ready except for the availability of the powered off node (see Test script step 11).

```
root@cli-vm:~# kubectl get all
```

```
NAME                               READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm      1/1       Running   1          1h
pod/frontend-7776b5c947-96jlh      1/1       Running   0          1h
pod/frontend-7776b5c947-xsrv7      1/1       Running   0          1h
pod/redis-master-5c95c89d9-78ds4   1/1       Running   0          1h
pod/redis-slave-6548b4f8b9-5p92d   1/1       Running   0          1h
pod/redis-slave-6548b4f8b9-x9fd9   1/1       Running   0          1h

NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend      ClusterIP   10.100.200.152   <none>        80/TCP
1h
service/kubernetes    ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master  ClusterIP   10.100.200.55    <none>        6379/TCP
1h
service/redis-slave   ClusterIP   10.100.200.194   <none>        6379/TCP
1h

NAME                           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend       3         3         3            2
1h
deployment.apps/redis-master   1         1         1            1
1h
deployment.apps/redis-slave    2         2         2            2
1h

NAME                                     DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-7776b5c947      3         3         2         1h
replicaset.apps/redis-master-5c95c89d9   1         1         1         1h
replicaset.apps/redis-slave-6548b4f8b9   2         2         2         1h
```

3 Application deployment is running and ready for all nodes (see Test script step 14).

```
root@cli-vm:~# kubectl get all
```

```
NAME                               READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm      1/1       Running   2          1h
pod/frontend-7776b5c947-96jlh      1/1       Running   0          1h
pod/frontend-7776b5c947-xsrv7      1/1       Running   0          1h
pod/redis-master-5c95c89d9-78ds4   1/1       Running   0          1h
pod/redis-slave-6548b4f8b9-5p92d   1/1       Running   0          1h
pod/redis-slave-6548b4f8b9-x9fd9   1/1       Running   0          1h

NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend      ClusterIP   10.100.200.152   <none>        80/TCP
1h
```

Mandatory for Stateless Applications Only

```
service/kubernetes     ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master   ClusterIP   10.100.200.55    <none>        6379/TCP
1h
service/redis-slave    ClusterIP   10.100.200.194   <none>        6379/TCP
1h

NAME                           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend       3         3         3            3
1h
deployment.apps/redis-master   1         1         1            1
1h
deployment.apps/redis-slave    2         2         2            2
1h

NAME                                  DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-7776b5c947   3         3         3         1h
replicaset.apps/redis-master-5c95c89d9   1         1         1         1h
replicaset.apps/redis-slave-6548b4f8b9   2         2         2         1h
```

PKS.Master.PowerOffOn

This test verifies that running K8s cluster master node can rejoin the cluster after it is powered off/on, when the Stateless application is running.

Note This test is mandatory for Stateless applications only.

Mandatory for Stateless Applications Only	
Test ID	PKS.Master.PowerOffOn
Configuration	PKS cluster is created. Application is running.

Mandatory for Stateless Applications Only

Test Script

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

2 Check the cluster existence.

```
root@cli-vm:~# pks clusters
```

```
Name               Plan Name            UUID                          Status
Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded
CREATE
```

3 Get credentials for the cluster.

```
root@cli-vm:~# pks get-credentials k8s-cluster-101
```

```
Fetching credentials for cluster k8s-cluster-101.
Context set for cluster k8s-cluster-101.
```

You can now switch between clusters by using:

```
$kubectl config use-context <cluster-name>
```

4 Set `kubectl` to use the context of the cluster.

```
root@cli-vm:~# kubectl config use-context k8s-cluster-101
```

```
Switched to context "k8s-cluster-101".
```

5 Verify the Application deployment readiness.

```
root@cli-vm:~# kubectl get all
```

6 Get the VM name for the master node under the power off/on test.

```
root@cli-vm:~# source A-BOSH.env
```

```
root@cli-vm:~# bosh vms
```

```
…
Deployment 'service-instance_a2a02b92-0ac5-4bfd-9706-fcfeae528d2b'

Instance                                     Process State  AZ
IPs         VM CID                                   VM Type      Active
master/4a7a6e3c-fcf3-4fa7-950b-7031c68f67ba  running        AZ-COMP
172.23.2.2  vm-04c898ea-6614-4b08-a22a-c50bed645293  medium.disk  true
```

7 Access vSphere Client. See Accessing vSphere Client for details.

8 Highlight `vm-04c898ea-6614-4b08-a22a-c50bed645293` under `RegionA01-COMP01/PKS-COMP` cluster and select **Power** > **Power Off** from the context menu.

9 Go back to the PuTTY session to CLI-VM and verify that the Application deployment cannot be accessed.

```
root@cli-vm:~# kubectl get all
```

10 Access vSphere Client.

11 Highlight `vm-04c898ea-6614-4b08-a22a-c50bed645293` under `RegionA01-COMP01/PKS-COMP` cluster and select **Power** > **Power On** from the context menu.

12 Go back to the PuTTY session to CLI-VM and monitor the application deployment status by running the `kubectl get all` command. Wait until the master node is shown as **Available** (it might take up to five minutes).

Test Output

1 Application deployment is running and ready (see Test script step 5).

Mandatory for Stateless Applications Only

```
root@cli-vm:~# kubectl get all
```

```
NAME                                 READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm        1/1       Running   2          2h
pod/frontend-7776b5c947-96jlh        1/1       Running   0          2h
pod/frontend-7776b5c947-xsrv7        1/1       Running   0          2h
pod/redis-master-5c95c89d9-78ds4     1/1       Running   0          2h
pod/redis-slave-6548b4f8b9-5p92d     1/1       Running   0          2h
pod/redis-slave-6548b4f8b9-x9fd9     1/1       Running   0          2h

NAME                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend         ClusterIP   10.100.200.152   <none>        80/TCP
2h
service/kubernetes       ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master     ClusterIP   10.100.200.55    <none>        6379/TCP
2h
service/redis-slave      ClusterIP   10.100.200.194   <none>        6379/TCP
2h

NAME                             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend         3         3         3            3
2h
deployment.apps/redis-master     1         1         1            1
2h
deployment.apps/redis-slave      2         2         2            2
2h

NAME                                       DESIRED   CURRENT   READY     AGE
replicaset.apps/frontend-7776b5c947        3         3         3         2h
replicaset.apps/redis-master-5c95c89d9     1         1         1         2h
replicaset.apps/redis-slave-6548b4f8b9     2         2         2         2h
```

2 Application deployment cannot be accessed because the master node is down (see Test script step 9).

```
root@cli-vm:~# kubectl get all
```

```
No resources found.
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
Unable to connect to the server: dial tcp 24.24.24.10:8443: i/o timeout
```

3 Application deployment is running and ready for all nodes (see Test script step 12).

```
root@cli-vm:~# kubectl get all
```

```
NAME                                 READY     STATUS    RESTARTS   AGE
pod/frontend-7776b5c947-7clrm        1/1       Running   2          2h
pod/frontend-7776b5c947-96jlh        1/1       Running   0          2h
pod/frontend-7776b5c947-xsrv7        1/1       Running   0          2h
pod/redis-master-5c95c89d9-78ds4     1/1       Running   0          2h
pod/redis-slave-6548b4f8b9-5p92d     1/1       Running   0          2h
```

Mandatory for Stateless Applications Only

```
pod/redis-slave-6548b4f8b9-x9fd9   1/1       Running   0          2h

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/frontend        ClusterIP   10.100.200.152   <none>        80/TCP
2h
service/kubernetes      ClusterIP   10.100.200.1     <none>        443/TCP
2d
service/redis-master    ClusterIP   10.100.200.55    <none>        6379/TCP
2h
service/redis-slave     ClusterIP   10.100.200.194   <none>        6379/TCP
2h

NAME                             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/frontend         3         3         3            3
2h
deployment.apps/redis-master     1         1         1            1
2h
deployment.apps/redis-slave      2         2         2            2
2h
```

Collecting Logs

8

After the validation is completed and all tests are passed, collect the logs from the PKS cluster nodes. Ensure that the Application solution is deployed and running in a good state after the validation tests are done. VMware might need to verify the Application solution deployment during the validation results review.

Procedure

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM.

Run the following commands:

```
root@cli-vm:~# mkdir -p /home/vmware/app-cluster-logs/master
root@cli-vm:~# mkdir -p /home/vmware/app-cluster-logs/worker0
root@cli-vm:~# mkdir -p /home/vmware/app-cluster-logs/worker1
root@cli-vm:~# mkdir -p /home/vmware/app-cluster-logs/worker2
```

2 SSH to the PCF OpsManager VM.

```
root@cli-vm:~# ssh ubuntu@23.23.23.2
```

```
Enter passphrase for key '/root/.ssh/id_rsa': VMware1!
ubuntu@pks-ops-mgr:~#
```

3 To get the PKS cluster deployment ID and node information, run the bosh command:

```
ubuntu@pks-ops-mgr:~$ source A-BOSH.env
ubuntu@pks-ops-mgr:~$ bosh vms
```

```
Using environment '10.1.1.3' as client 'ops_manager'

Deployment 'harbor-container-registry-8d51221a98ebb37124f6'
Instance                                      Process State  AZ       IPs       VM
CID               VM Type      Active
harbor-app/5855fc38-e432-4585-818b-013bd2b839b3  running        AZ-MGMT  10.1.1.5  vm-
bccb123b-3d90-410b-97d3-fe6c011468a5  xlarge.disk  true
1 vms

Deployment 'pivotal-container-service-0b648dc45ece40cd92e1'
Instance                                                      Process State  AZ       IPs
VM CID      VM Type  Active
```

```
pivotal-container-service/0d785096-429e-4267-b6b6-89b5d0295461  running        AZ-MGMT  10.1.1.4
vm-4915010b-b6d7-4ce1-8fda-693efe81ba3e  large    true
1 vms

Deployment 'service-instance_1a385d84-c0cd-4dec-984a-f7735a9500af'
Instance                                     Process State  AZ       IPs         VM
CID                   VM Type      Active
master/43ad8d9c-0469-42eb-b0d1-0ceeb8f99a16  running        AZ-COMP  172.23.1.2
vm-5df9079f-9dbc-4a68-a2da-b0e39552a010  medium.disk  true
worker/899f1f7b-28ff-4b2c-addb-b52fa56d07cb  running        AZ-COMP  172.23.1.3  vm-554d644c-
d8cc-4fd2-8caa-3a1c4937ca9f  medium.disk  true
worker/c7b24def-d3b9-465f-93af-93ea99d1faa1  running        AZ-COMP  172.23.1.4  vm-38f0bbcf-
cef1-4afe-b99d-676b2677f62e  medium.disk  true
worker/ec488113-899d-409c-b1e0-6e8254868618  running        AZ-COMP  172.23.1.5  vm-4e2a6d48-
bcb3-40aa-9048-1ac153a6ec9f  medium.disk  true
4 vms
Succeeded
```

4 From the PCF OpsManager VM, SSH to the PKS cluster master node.

`ubuntu@pks-ops-mgr:~$ bosh -e 10.1.1.3 -d service-instance_1a385d84-c0cd-4dec-984a-f7735a9500af ssh master/0`

```
Using environment '10.1.1.3' as client 'ops_manager'
Using deployment 'service-instance_1a385d84-c0cd-4dec-984a-f7735a9500af'
Task 415084. Done
master/43ad8d9c-0469-42eb-b0d1-0ceeb8f99a16:~$
```

5 Collect the *kube-apiserver* and *kube-controller-manager* logs from the master node by sudo SCP transferring them into `/home/vmware` on the CLI-VM.

`master/43ad8d9c-0469-42eb-b0d1-0ceeb8f99a16:~$ ls -l /var/vcap/sys/log/kube-apiserver/`

```
…
-rw------- 1 vcap vcap 12237564 Dec  8 02:03 kube-apiserver.stderr.log
-rw------- 1 vcap vcap        0 Dec  5 00:43 kube-apiserver.stdout.log
```

`master/43ad8d9c-0469-42eb-b0d1-0ceeb8f99a16:~$ ls -l /var/vcap/sys/log/kube-controller-manager/`

```
…
-rw------- 1 vcap vcap 179295 Dec  8 01:59 kube-controller-manager.stderr.log
-rw------- 1 vcap vcap      0 Dec  5 00:43 kube-controller-manager.stdout.log
```

`master/43ad8d9c-0469-42eb-b0d1-0ceeb8f99a16:~$ exit`

```
logout
Connection to 172.23.1.2 closed.
Succeeded
```

sudo scp command example:

```
ubuntu@pks-ops-mgr:~$ sudo scp -r /var/vcap/sys/log/kube-apiserver/*
root@192.168.110.7:/home/vmware/app-cluster-logs/master/
```

6 From the PCF OpsManager VM, SSH to the PKS cluster worker node.

```
ubuntu@pks-ops-mgr:~$ bosh -d service-instance_1a385d84-c0cd-4dec-984a-
f7735a9500af ssh worker/0
```

```
Using environment '10.1.1.3' as client 'ops_manager'
Using deployment 'service-instance_1a385d84-c0cd-4dec-984a-f7735a9500af'
Task 415703. Done
worker/899f1f7b-28ff-4b2c-addb-b52fa56d07cb:~$
```

7 Collect the *kubelet* logs from the worker node by sudo SCP transferring them into `/home/vmware` on the CLI-VM.

```
worker/899f1f7b-28ff-4b2c-addb-b52fa56d07cb:~$ ls -l /var/vcap/sys/log/kubelet/
```

```
total 380
-rw-r--r-- 1 root root   1605 Dec  5 00:49 kubelet_ctl.stderr.log
-rw-r--r-- 1 root root     81 Dec  5 00:49 kubelet_ctl.stdout.log
-rw-r--r-- 1 root root 368080 Dec  8 02:02 kubelet.stderr.log
-rw-r--r-- 1 root root     31 Dec  5 00:49 kubelet.stdout.log
-rw-r----- 1 root root      0 Dec  5 00:49 post-start.stderr.log
-rw-r----- 1 root root   3180 Dec  5 00:52 post-start.stdout.log
```

sudo scp command example:

```
ubuntu@pks-ops-mgr:~$ sudo scp -r /var/vcap/sys/log/kubelet/*
root@192.168.110.7:/home/vmware/app-cluster-logs/worker0/
```

8 Repeat steps 6 and 7 for all worker nodes.

9 Create a *tgz* bundle with all logs from the master and worker nodes.

```
root@cli-vm:~# cd /home/vmware
```

```
root@cli-vm:/home/vmware# tar cvfz app-cluster-logs.tgz app-cluster-logs/
```

```
app-cluster-logs/
app-cluster-logs/worker0/
app-cluster-logs/worker0/kubelet.stderr.log
app-cluster-logs/worker0/kubelet_ctl.stderr.log
app-cluster-logs/master/
app-cluster-logs/master/kube-apiserver.stderr.log
app-cluster-logs/master/kube-controller-manager.stderr.log
app-cluster-logs/worker1/
app-cluster-logs/worker1/kubelet.stderr.log
app-cluster-logs/worker1/kubelet_ctl.stderr.log
app-cluster-logs/worker2/
app-cluster-logs/worker2/kubelet.stderr.log
app-cluster-logs/worker2/kubelet_ctl.stderr.log
```

10 Use WinSCP to transfer (see Downloading and Transferring Files for details) the *tgz* bundle from CLI-VM to the control center Windows VM.

What to do next

Submit the results.

Submitting Validation Results

9

After collecting the PKS cluster logs, you must submit the validation results to VMware.

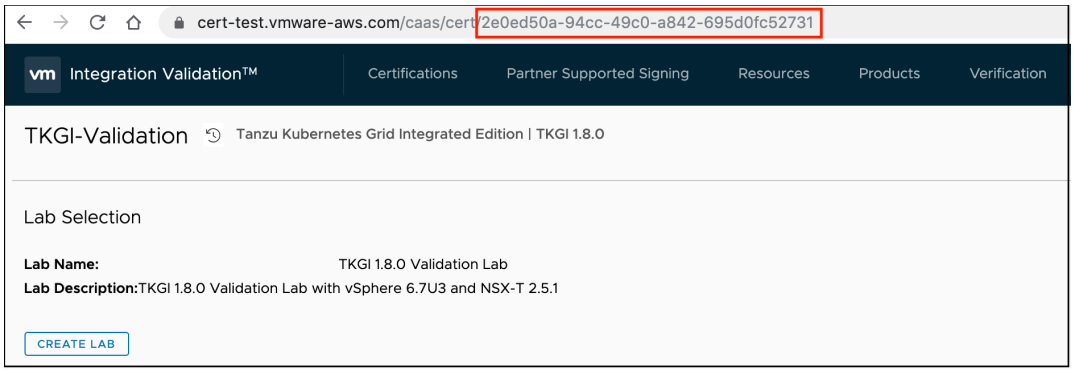
Procedure

1 Collect PKS Cluster logs. See Chapter 8 Collecting Logs for details.

2 Fill the VMware Tanzu Kubernetes Grid Integrated Edition Validation-in-Cloud - Validation Tests Check List.

3 Upload the PKS cluster logs and the VMware Tanzu Kubernetes Grid Integrated Edition Validation-in-Cloud - Validation Tests Check List to the DCPN case.

4 Enter the VIVa Session ID (see Lab Selection) in the DCPN case - comment section to identify the validation session.



5 From the web browser of your Main Console VM, enter the URL: *http://10.148.172.7*. An internal IP address is displayed. Also, an internal IP address is displayed on the Main Console VM desktop. Report this IP address in the DCPN case.

Requesting Partner-Ready VMware Tanzu Logo

10

After the VMware Validation team confirms in the DCPN case that your solution meets the validation requirements, perform the following steps:

Procedure

1 Request for a Partner-Ready logo.

Write to *vsxalliance@vmware.com* requesting for a Partner-Ready VMware Tanzu Logo.

2 Submit a support statement.

a Go to http://vmware-alliances.force.com/SubmitSupport.

b Enter the necessary information in the requested fields.

c In the **Application Category** drop-down menu, select the appropriate application category.

d Click **Add Support Statement**.

e In the Support Statement page, read through the support statement, and click **Submit**.

Working with the Lab

11

This chapter provides general information about VMware Validation Lab environment and ways to work with the lab components.

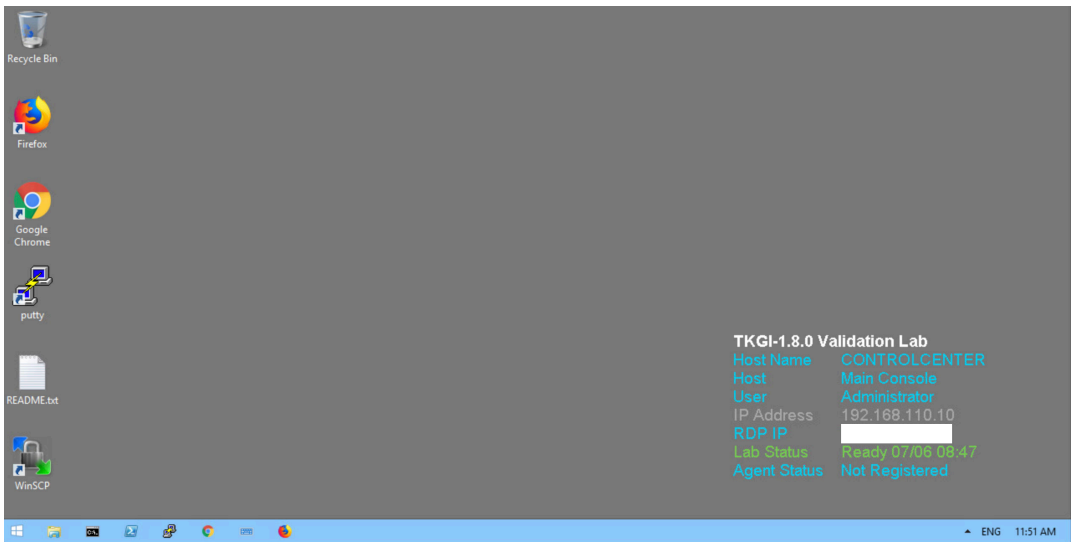
This chapter includes the following topics:

Utilities Installed on the Lab

The following utilities and applications are pre-installed on the control center Windows VM;

- Firefox web browser
- Google Chrome web browser
- PuTTY
- WinSCP
- Notepad++ (go to **Start** > **All Programs**).

See the `README.txt` file on the Desktop for Software BOM and Lab component credentials.



Downloading and Transferring Files

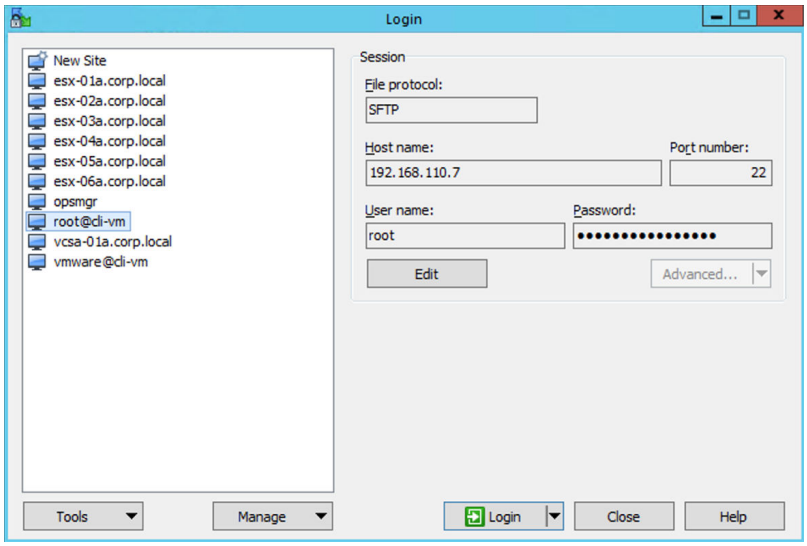
You download files from the Internet as follows:

- If your Application deployment requires downloading files from the Internet, run these commands on the CLI-VM:

```
git clone
wget
curl
```

- If your Application deployment requires downloading files from the Internet using a web interface, use the Firefox browser from the control center Windows VM.

To transfer files from Windows VM to the CLI-VM, use WinSCP from the control center Windows VM. Connect to the CLI-VM 192.168.110.7 (user name: *root* ; password: *VMware1!*) and copy the files into `/root` on CLI-VM.



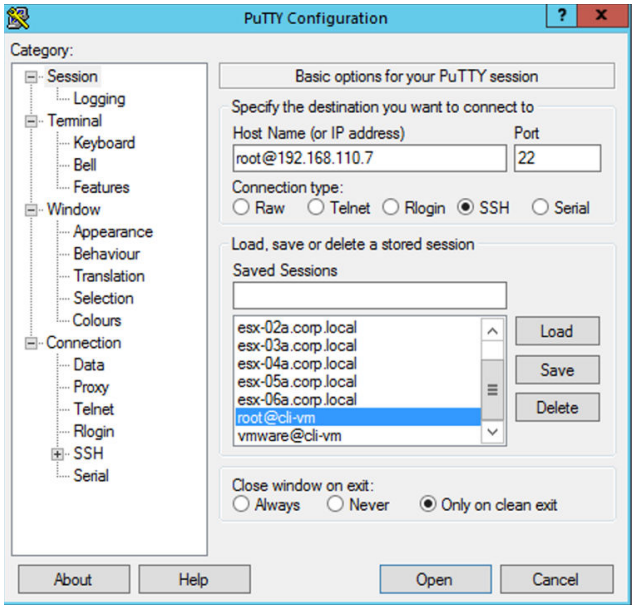
Accessing CLI VM

To access the CLI-VM, perform the following steps:

Procedure

1 Open PuTTY by clicking the PuTTY icon on the Desktop.

2 Select **root@cli-vm** or **Default Settings** in the Stored Sessions list, and click Load.



3 Click Open. A prompt opens requesting to enter the password for `root@192.168.110.7`.

4 Enter the password - *VMware1!*

Results

The Command Prompt for CLI VM opens.

Accessing TKGI Environment

This section provides information about how to create a PKS cluster by accessing the TKGI Lab through the VMware Learning Platform (VLP) control center Windows VM.

To log in to PKS, perform the following steps:

1 Open PuTTY session for the CLI-VM. See Accessing CLI VM.

2 Log in to PKS.

```
root@cli-vm:~# pks login -a uaa.corp.local -u vmware -p VMware1! -k
```

```
API Endpoint: uaa.corp.local
User: vmware
```

Creating a PKS Cluster

The cluster creation can take a significant amount of time to complete (about 45–60 minutes).

You can use the command - `pks create-cluster` and specify:

- one of the predefined cluster plans: --plan [small|medium|large]
- number of workers: --num-nodes <number>

Note Do not use the same name for internal and external cluster names.

For PKS cluster plan details, see Chapter 15 Appendix C: TKGI Cluster Plans.

If the Application requires resources customization for the cluster, plan can be modified from PCF Ops Manager. See the following PKS documentation for details:

- Installing Enterprise PKS on vSphere with NSX-T
- Plans

Run this command:

```
root@cli-vm:~# pks create-cluster k8s-cluster-101 --external-hostname pks-cluster-101.corp.local --plan small --num-nodes 3
```

```
Name:                     k8s-cluster-101
Plan Name:                small
UUID:                     53187740-e5cf-4ed7-9181-ab71bbdc1945
Last Action:              CREATE
Last Action State:        in progress
Last Action Description:  Creating cluster
Kubernetes Master Host:   pks-cluster-101.corp.local
Kubernetes Master Port:   8443
Worker Nodes:             3
Kubernetes Master IP(s):  In Progress
Network Profile Name:
Use pks cluster k8s-cluster-101 to monitor the state of your cluster
```

To check the cluster creation task status (it might take up to 45–60 minutes), run the `pks cluster cluster-name` command.

Checking PKS Cluster Status

After creating the PKS cluster, you must check the cluster name and other details.

Procedure

1 To get the PKS cluster name and status, run this command:

```
root@cli-vm:~# pks clusters
```

```
Name             Plan Name  UUID                                  Status     Action
k8s-cluster-101  small      53187740-e5cf-4ed7-9181-ab71bbdc1945  succeeded  CREATE
```

2 To get PKS cluster details, run this command:

```
root@cli-vm:~# pks cluster k8s-cluster-101
```

```
Name:                     k8s-cluster-101
Plan Name:                small
UUID:                     53187740-e5cf-4ed7-9181-ab71bbdc1945
Last Action:              CREATE
Last Action State:        succeeded
Last Action Description:  Instance provisioning completed
Kubernetes Master Host:   pks-cluster-101.corp.local
Kubernetes Master Port:   8443
Worker Nodes:             3
Kubernetes Master IP(s):  24.24.24.10
Network Profile Name:
```

3 The *pks-cluster-101.corp.local* entry must be added to the `/etc/hosts` file, otherwise *kubectl* fails to communicate with *k8s-cluster-101*.

```
root@cli-vm:~# vi /etc/hosts
```

Add *24.24.24.10 pks-cluster-101.corp.local* line to the end of the `/etc/hosts` file.

4 Fetch the credentials for cluster *k8s-cluster-101*:

```
root@cli-vm:~# pks get-credentials k8s-cluster-101
```

```
Fetching credentials for cluster k8s-cluster-101.
Context set for cluster k8s-cluster-101.
User can now switch between clusters by using:
$kubectl config use-context <cluster-name>
```

```
root@cli-vm:~# kubectl config use-context k8s-cluster-101
```

```
 Switched to context "k8s-cluster-101".
```

Validating Nodes for Cluster

To validate nodes for cluster *k8s-cluster-101*, run the following command:

```
root@cli-vm:~# kubectl get nodes -o wide
```

```
NAME                                   STATUS    ROLES     AGE       VERSION   INTERNAL-IP   EXTERNAL-
IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
afc7bfbd-bea7-4044-8d81-2ed6121eec78   Ready     <none>    24m       v1.11.2   172.23.0.4
172.23.0.4    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://17.12.1-ce
cd48fdbc-5f15-437c-9a1b-f70235c59a62   Ready     <none>    31m       v1.11.2   172.23.0.3
172.23.0.3    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://17.12.1-ce
dc8ca134-93bb-422a-89d0-89a2ba359685   Ready     <none>    17m       v1.11.2   172.23.0.5
172.23.0.5    Ubuntu 16.04.5 LTS   4.15.0-33-generic   docker://17.12.1-ce
```

Installing Helm Package Manager

After the PKS cluster is created, you can install the Helm package manager. Run the following commands:

Procedure

1 `root@cli-vm:~# curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash`

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  7230  100  7230    0     0   4160      0  0:00:01  0:00:01 --:--:--  4159
Downloading https://kubernetes-helm.storage.googleapis.com/helm-v2.12.3-linux-amd64.tar.gz
Preparing to install helm and tiller into /usr/local/bin
helm installed into /usr/local/bin/helm
tiller installed into /usr/local/bin/tiller
Run 'helm init' to configure helm.
```

2 `root@cli-vm:~# kubectl -n kube-system create sa tiller && kubectl create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:tiller`

```
serviceaccount/tiller created
clusterrolebinding.rbac.authorization.k8s.io/tiller created
```

3 `root@cli-vm:~# helm init --skip-refresh --upgrade --service-account tiller`

```
Creating /root/.helm
Creating /root/.helm/repository
…
Creating /root/.helm/cache/archive
Creating /root/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
```

```
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see:
https://docs.helm.sh/using_helm/#securing-your-helm-installation
Happy Helming!
```

4 `root@cli-vm:~# helm version`

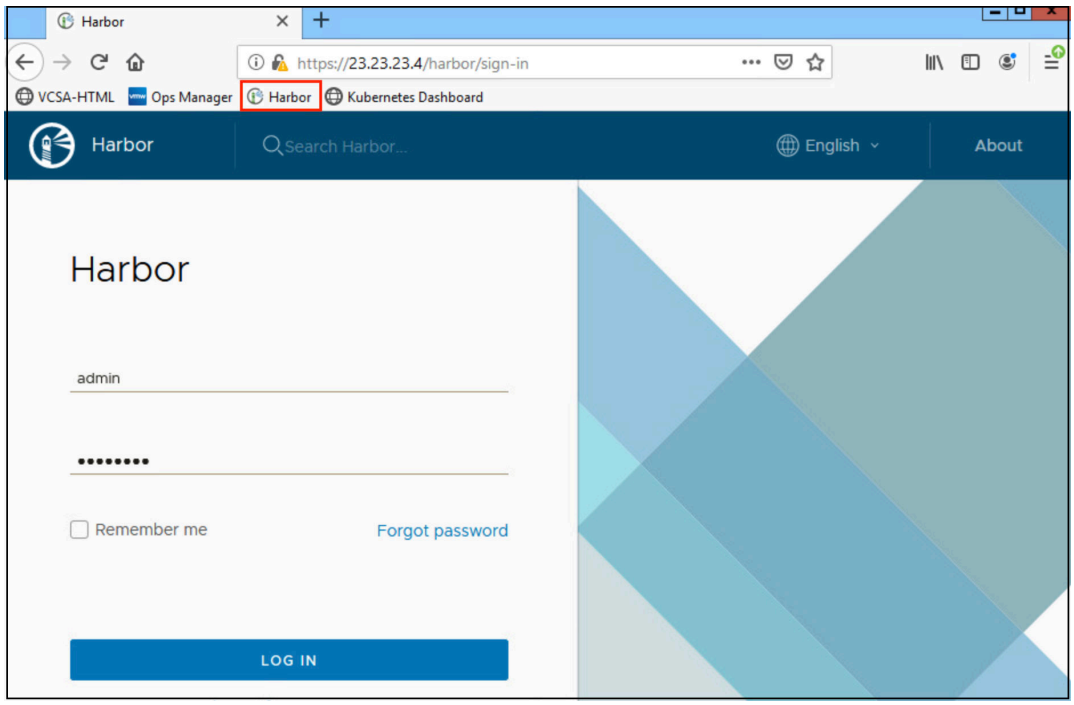
```
Client: &version.Version{SemVer:"v2.12.3", GitCommit:"2e55dbe1fdb5fdb96b75ff144a339489417b146b",
GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.12.3", GitCommit:"2e55dbe1fdb5fdb96b75ff144a339489417b146b",
GitTreeState:"clean"}
```

It might take some time (approximately 10 minutes) before the error message "`Error: could not find a ready tiller pod`" disappears and the server version is displayed.

Accessing Harbor Repository

To access the harbor repository, open the Firefox browser, click the Harbor bookmark, and log in to Harbor.

- URL: *https://23.23.23.4*
- User name: *admin*
- Password: *VMware1!*



Pushing Images into Harbor

To push unsigned images into Harbor from an external repository, perform the following steps:

Procedure

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM.

2 Pull the images from the external repository (Google sample images are used as examples).

```
root@cli-vm:~# docker pull gcr.io/google-samples/gb-frontend:v4
root@cli-vm:~# docker pull gcr.io/google_containers/redis:e2e
root@cli-vm:~# docker pull gcr.io/google_samples/gb-redisslave:v1
```

To check the list of images, run this command:

```
root@cli-vm:~# docker images
```

```
REPOSITORY                          TAG                 IMAGE ID            CREATED
SIZE
gcr.io/google-samples/gb-frontend   v4                  e2b3e8542af7        2 years ago
512MB
gcr.io/google_samples/gb-redisslave v1                  5f026ddffa27        3 years ago
110MB
gcr.io/google_containers/redis      e2e                 e5e67996c442        3 years ago
419MB
```

3 Tag the images.

```
root@cli-vm:~# docker tag gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
library/gb-frontend:v4
root@cli-vm:~# docker tag gcr.io/google_samples/gb-redisslave:v1
harbor.corp.local/library/gb-redisslave:v1
root@cli-vm:~# docker tag gcr.io/google_containers/redis:e2e harbor.corp.local/
library/redis:e2e
```

4 Push the images to Harbor.

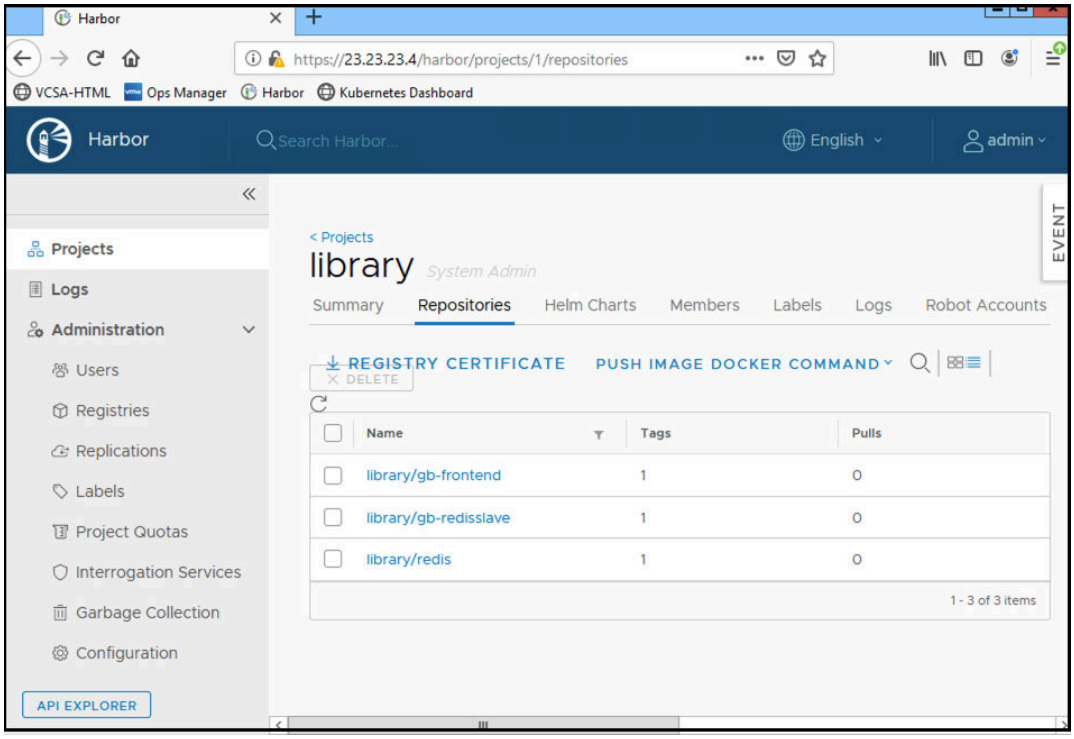
```
root@cli-vm:~# docker push harbor.corp.local/library/gb-frontend:v4
root@cli-vm:~# docker push harbor.corp.local/library/gb-redisslave:v1
root@cli-vm:~# docker push harbor.corp.local/library/redis:e2e
```

5 On the Harbor web UI, verify that the three images are present.

Open the Firefox browser, click the Harbor bookmark (URL: https://23.23.23.4), and log in to Harbor.

- User name: *admin*
- Password: *VMware1!*

Go to **Projects** > **Repositories**.



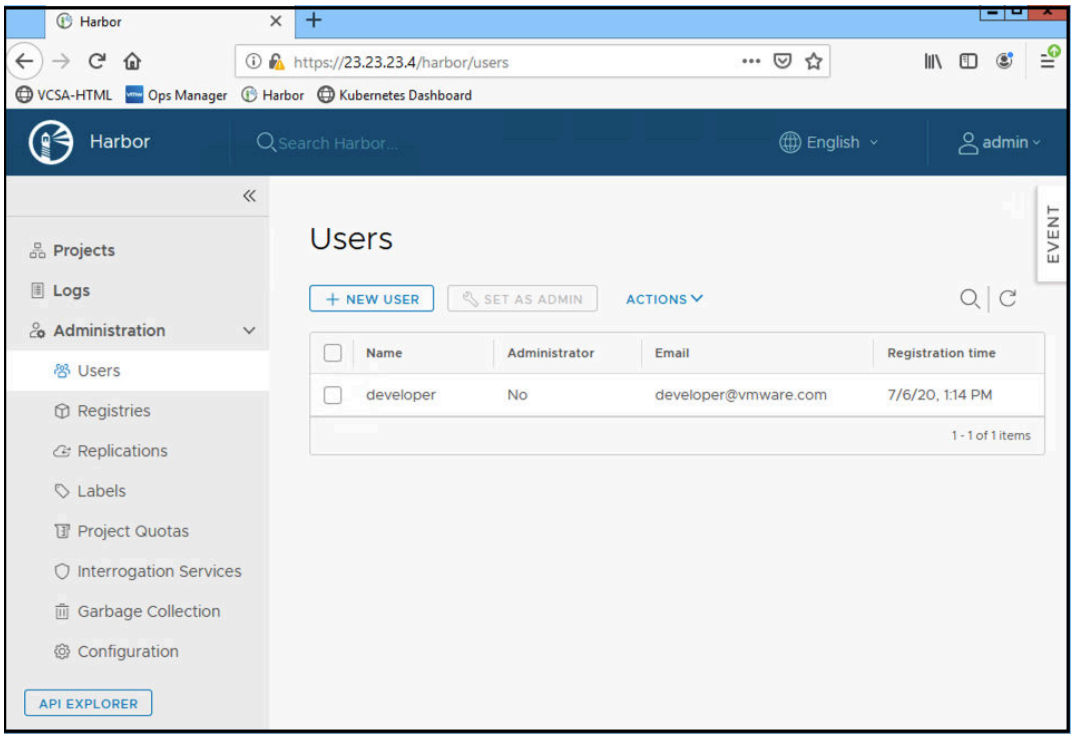
Pushing Securely Signed Images to Harbor

To push securely signed images to Harbor, perform the following steps:

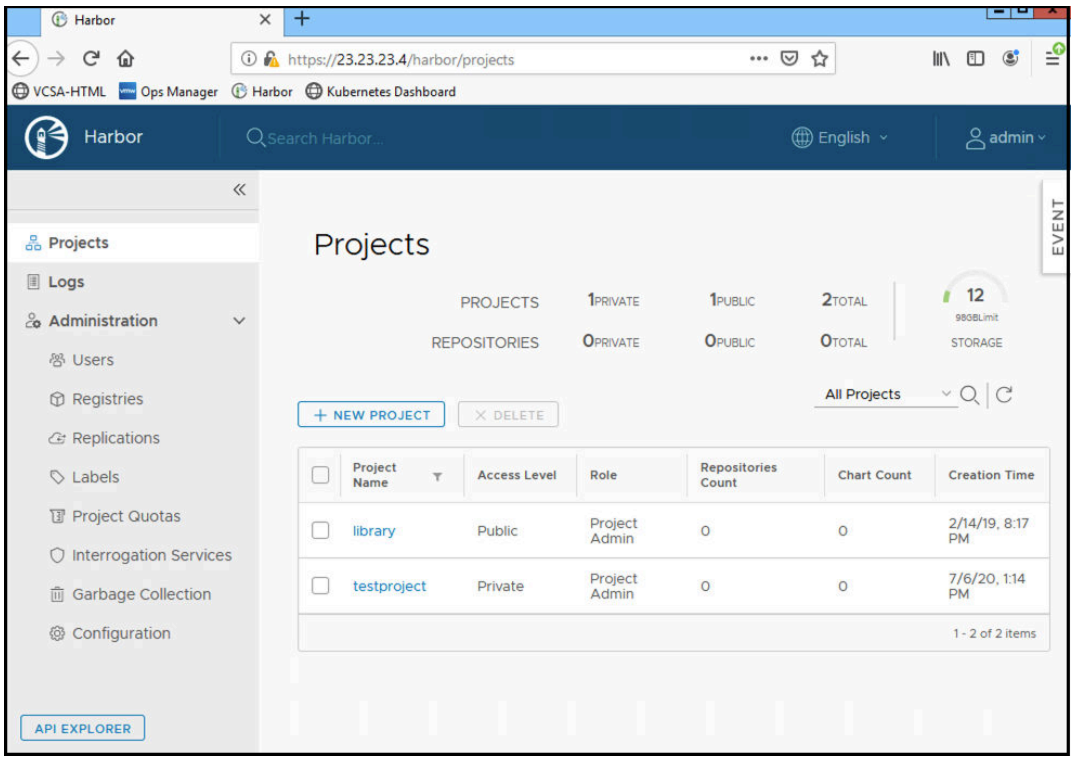
Procedure

1 Log in to Harbor.

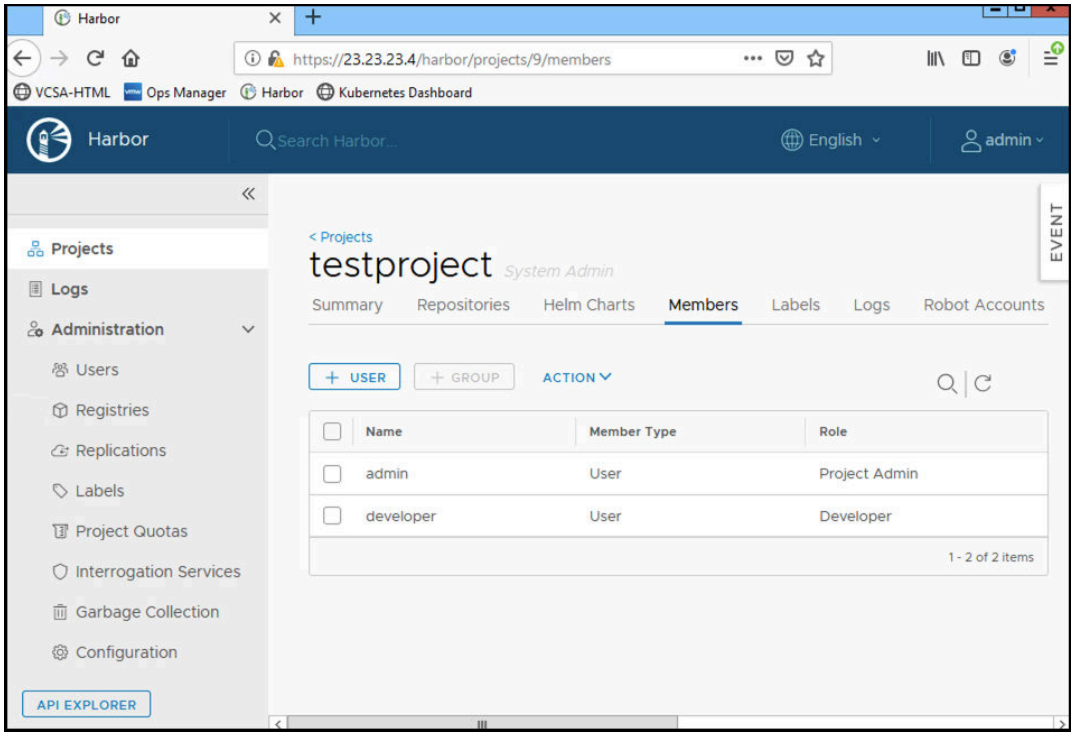
2 Go to **Administration > Users** and click **+NEW USER** (example: *developer*).



3 From the menu on the left, click **Projects** and create a new project by clicking **+NEW Project** (example: *testproject*).

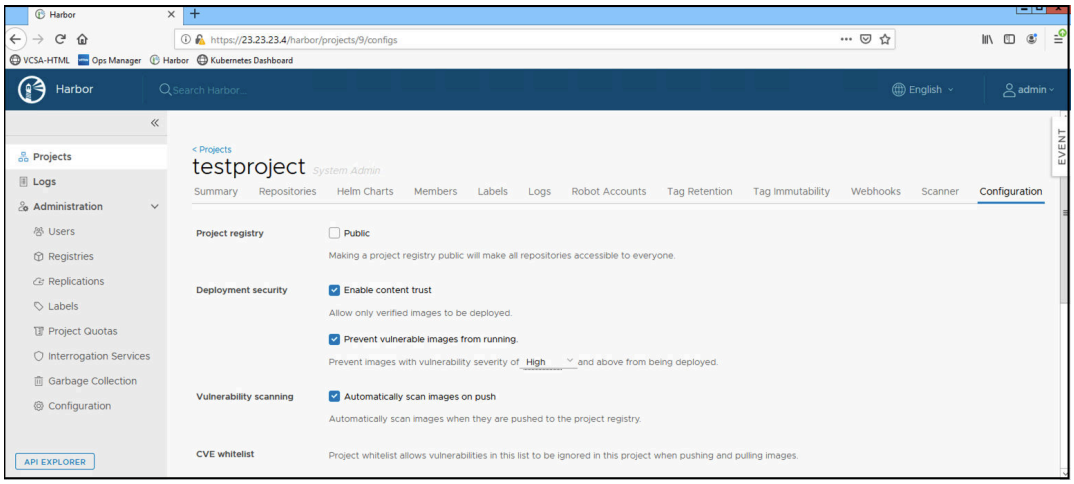


4 Click the project that you created. Under the **Members** tab, click **+User** and add the user that you created in step 1 (*developer*). Ensure that the user is assigned the *Developer* role.



5 Go to the **Configuration** tab, set the values as follows, and click **SAVE**:

- Select **Enable Content Trust**.
- Select **Prevent vulnerable images from running** and set the severity to "*Low*"
- Select **Automatically scan images on push**.



6 Start a PuTTY session to CLI-VM. See Accessing CLI VM for details.

7 Pull the images from the external repository (example: Google sample images).

```
root@cli-vm:~# docker pull gcr.io/google-samples/gb-frontend:v4
```

```
root@cli-vm:~# docker pull gcr.io/google_containers/redis:e2e

root@cli-vm:~# docker pull gcr.io/google_samples/gb-redisslave:v1
```

8 Log in to Harbor registry as *developer*.

```
root@cli-vm:~# docker login harbor.corp.local -u developer
```

9 Set up two environment variables and tag the images.

```
root@cli-vm:~# export DOCKER_CONTENT_TRUST=1

root@cli-vm:~# export DOCKER_CONTENT_TRUST_SERVER=https://harbor.corp.local:4443

root@cli-vm:~# docker tag gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4

root@cli-vm:~# docker tag gcr.io/google_samples/gb-redisslave:v1
harbor.corp.local/testproject/gb-redisslave:v1

root@cli-vm:~# docker tag gcr.io/google_containers/redis:e2e harbor.corp.local/
testproject/redis:e2e
```

10 Push the images into the Harbor repository. You are asked to enter a root key passphrase. You must enter the passphrase (*VMware1!*) every time you sign the image.

```
root@cli-vm:~# docker push harbor.corp.local/testproject/gb-frontend:v4
```

The push refers to repository [harbor.corp.local/testproject/gb-frontend]

```
3a31f3bf94a2: Layer already exists
cdc990c9b585: Layer already exists
…
816f1903c60f: Layer already exists
c12ecfd4861d: Layer already exists
v4: digest: sha256:aaa5b327ef3b4cb705513ab674fa40df66981616950c7de4912a621f9ee03dd4 size: 6968
Signing and pushing trust metadata
Enter passphrase for root key with ID ec1a991: VMware1!
Repeat passphrase for root key with ID ec1a991: VMware1!
Enter passphrase for new repository key with ID 7284e73: VMware1!
Repeat passphrase for new repository key with ID 7284e73: VMware1!
Finished initializing "harbor.corp.local/testproject/gb-frontend"
Successfully signed harbor.corp.local/testproject/gb-frontend:v4
```

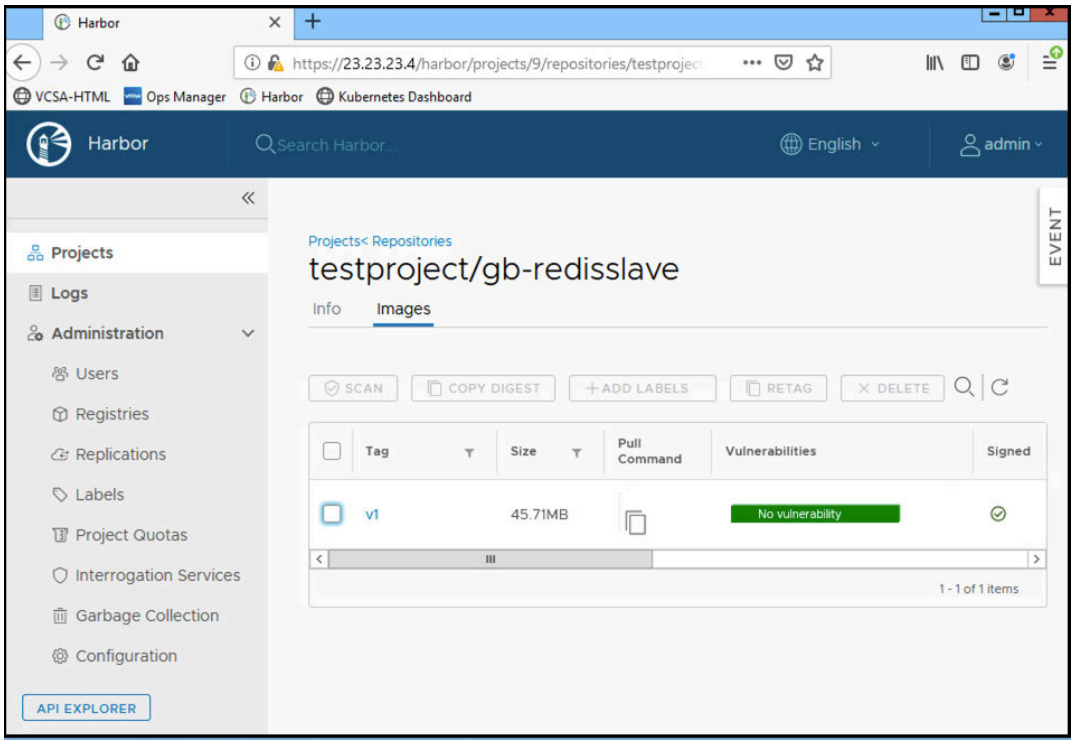
```
root@cli-vm:~# docker push harbor.corp.local/testproject/gb-redisslave:v1
```

```
…
Signing and pushing trust metadata
Enter passphrase for root key with ID ec1a991: VMware1!
Enter passphrase for new repository key with ID 63e9a04: VMware1!
Repeat passphrase for new repository key with ID 63e9a04: VMware1!
Finished initializing "harbor.corp.local/testproject/gb-redisslave"
Successfully signed harbor.corp.local/testproject/gb-redisslave:v1
```

```
root@cli-vm:~# docker push harbor.corp.local/testproject/redis:e2e
```

```
…
Signing and pushing trust metadata
Enter passphrase for root key with ID ec1a991: VMware1!
Enter passphrase for new repository key with ID e7477bb: VMware1!
Repeat passphrase for new repository key with ID e7477bb: VMware1!
Finished initializing "harbor.corp.local/testproject/redis"
Successfully signed harbor.corp.local/testproject/redis:e2e
```

11 The tags (v4, v1, e2e in example) are marked as Signed in the Harbor UI. Go to **Projects** > **Repositories** > **testproject/gb-redisslave** (testproject/gb-frontend, or testproject/redis).



Steps to Add Storage Class

Procedure

1 Start a PuTTY session to CLI-VM. See Accessing CLI VM.

2 Create a `storage-class.yaml` file.

```
root@cli-vm:~# vi storage-class.yaml
```

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard-sc
  annotations:
```

```
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/vsphere-volume
parameters:
    diskformat: thin
```

3 Save and exit the `storage-class.yaml` file.

4 In the Command Prompt, run the following commands:

```
root@cli-vm:~# kubectl apply -f storage-class.yaml
```

```
storageclass.storage.k8s.io/standard-sc created
```

```
root@cli-vm:~# kubectl get sc
```

```
NAME                    PROVISIONER                    AGE
standard-sc (default)   kubernetes.io/vsphere-volume   8s
```

For reference, see:

- Deploy a Kubernetes Stateful Application on PKS - WordPress: https://code.vmware.com/samples/4835/deploy-a-kubernetes-stateful-application-on-pks---wordpress?h=Pivotal%20Container%20Service%20(PKS).
- Storage Classes: https://kubernetes.io/docs/concepts/storage/storage-classes/.

Accessing vSphere Client

To access the vSphere Client, perform the following steps:

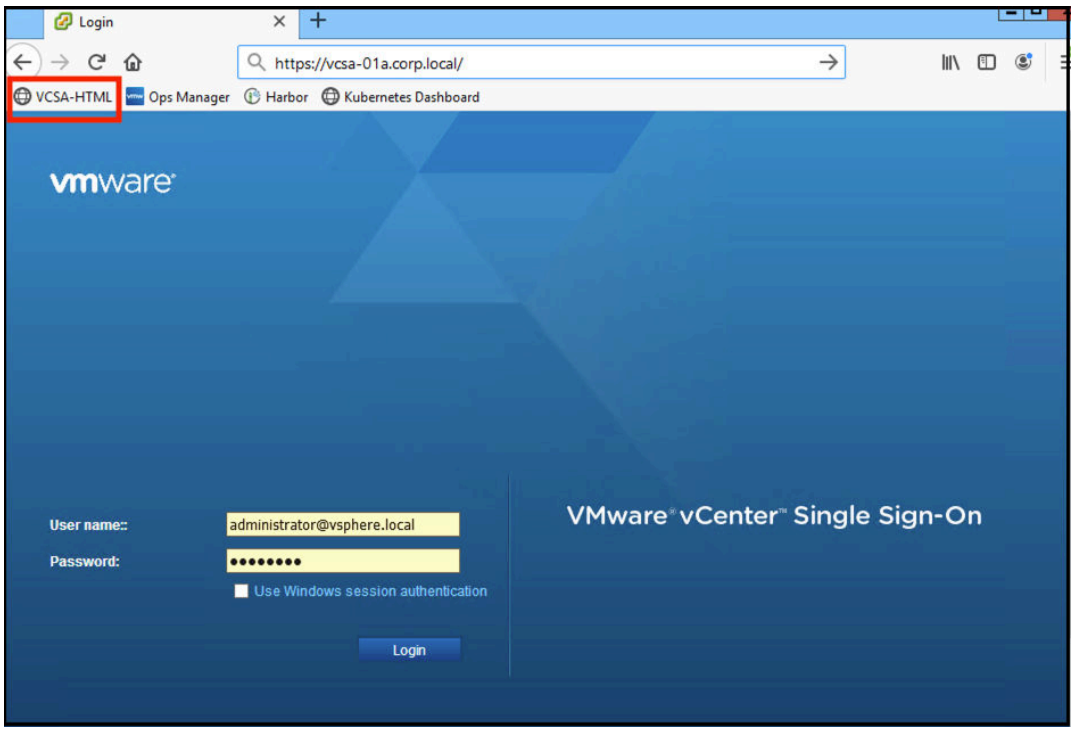
Procedure

1 From the control center Windows VM, open the Firefox web browser.

a Click the VCSA-HTML bookmark. The vCenter Server Appliance page opens.

b Log in to vCenter.

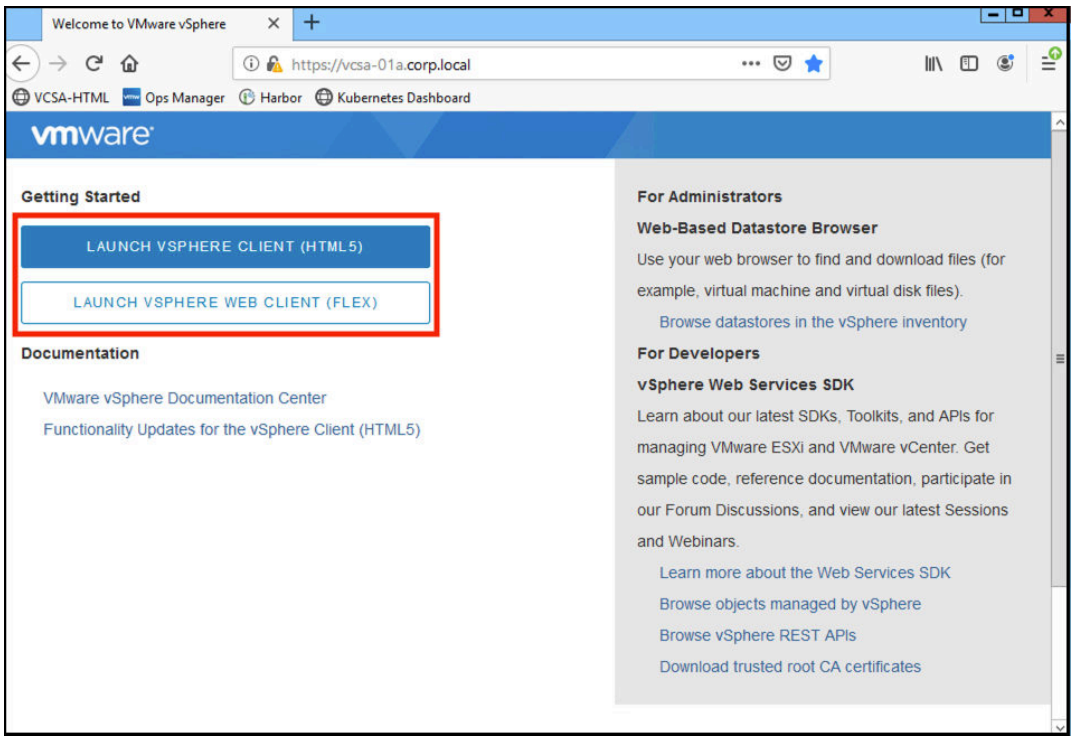
- URL: *https://vcsa-01a.corp.local*
- User name: *administrator@vsphere.local*
- Password: *VMware1!*



2 Click any one of the following links:

a *vSphere Web Client (Flex)*

b *vSphere Client (HTML5)*



Load Balancers

There are no load balancers that are provided in the VMware Validation Lab for TKGI. You may, however, bring your own into the environment.

NSX-T IP pools are used for the associated external IP addresses that are issued to the load balancer service. There might be two IP addresses assigned to the load balancer. If more than one IP address is issued from the IP pools, only one IP address is used to contact the deployed application. The other IP address is used only for communication between the load balancer and the NSX-T router. Ensure that the correct IP address is used.

Note Do not specify any load balancer IP addresses during deployment (in YAML files, Helm charts, or others) as these IP addresses can conflict with the IPs that are issued from the IP pools.

Two IP addresses might be issued from NSX-T:

- 100.64.x.x/31 (This IP address is only accessible from the NSX-T router - DO NOT use this IP address).
- 24.24.24.x/24 (This IP address is the application IP and is routable from the Main Console VM).

Example output from `kubectl` (note down the EXTERNAL-IP field for gateway-external):

```
root@cli-vm:~# kubectl --namespace=test get services
```

```
NAME                      TYPE               CLUSTER-IP       EXTERNAL-IP
PORT(S)          AGE
Service1                  ClusterIP         10.100.200.159   <none>                      9093/
TCP         11m
gateway                  ClusterIP         10.100.200.130   <none>                      8080/
TCP         11m
gateway-external  LoadBalancer 10.100.200.183   100.64.176.7,24.24.24.22   8080:31924/TCP   11m
Service2                  ClusterIP         10.100.200.71    <none>                      4222/
TCP         11m
Service3                  ClusterIP         10.100.200.197   <none>                      9090/
TCP         11m
```

Adding Persistent Storage

The VMware Enterprise PKS environment has limited storage available, which can be used as part of the deployment. The entire amount of available storage can be determined by using vSphere Client, but it will be at least a few 100 GBs.

Adding storage devices using thin provisioning allows the PKS environment to see larger devices and reduces errors in provisioning.

Adding Unmounted Block Devices to the Worker and Master Nodes

To add unmounted block devices to the PKS nodes in your cluster, simply add vSphere disks to the appropriate cluster VMs.

To determine the VM names of the PKS cluster nodes, run the following commands:

```
root@cli-vm:~# source A-BOSH.env
```

```
root@cli-vm:~# bosh vms
```

```
Using environment '10.1.1.3' as client 'ops_manager'

Task 36350
Task 36352
Task 36351
Task 36350 done
Task 36351 done
Task 36352 done
…
…
Deployment 'service-instance_ecd48568-cd51-4cca-a31e-558dc4442853'
Instance                                    Process State  AZ       IPs         VM
CID                                  VM Type      Active
master/4ca975d0-e083-4784-975f-19cd625132e1  running        AZ-COMP  172.23.1.2
vm-6959007d-8fca-4433-9c86-d796a4cdb579  medium.disk  true
worker/029af88e-9436-4c81-b13e-00d32f138a48  running        AZ-COMP  172.23.1.4  vm-95309c86-
f989-4704-8d23-31fcbfd0bdee  medium.disk  true
worker/04bd7e45-885a-485f-877f-2fa3dc3fb103  running        AZ-COMP  172.23.1.3  vm-fdebbec0-
d556-4b69-9641-fa9121f20445  medium.disk  true
3 vms
```

```
root@cli-vm:~#
```

Using vSphere Client, add virtual disks to the appropriate VMs. It is recommended to use thin provisioning with these disks. The VMs can see the block devices, but they are unmounted on the cluster VMs.

To add the additional block device, perform the following steps:

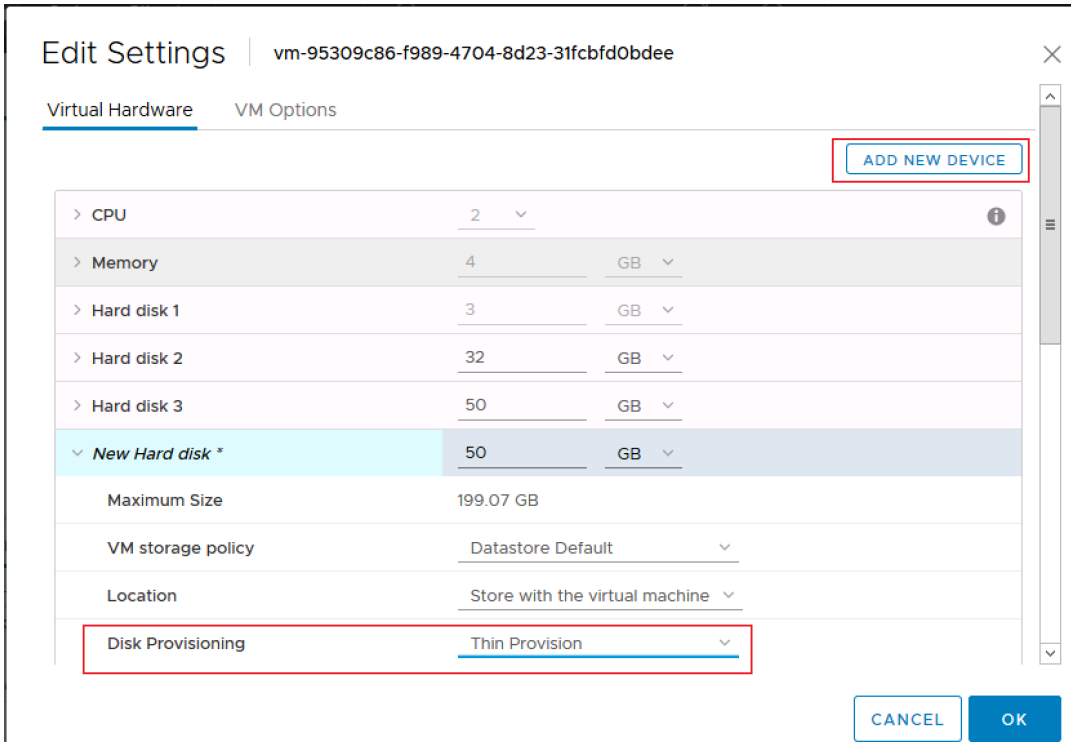
Procedure

1. Right-click the VM and click **Edit Settings**.
2. Click **ADD NEW DEVICE** and select **Hard Disk**.
3. Choose the appropriate size for the virtual device.
4. Expand the options for the device and select **Thin Provision** for **Disk Provisioning**.

   Repeat steps 1–4 for each disk that is to be added to the VM.

5 After adding all the disks, click **OK**.

Repeat steps 1–5 for each VM that requires additional unmounted block devices.



Adding Persistent Storage to a Pod Using Persistent Volume Claim

There are many different ways and different types of storage that can be configured as part of deployments. This section provides information about one statically provisioned storage example and one dynamically provisioned storage example.

The following subsections provide guidance on how to address additional storage requirements in the validation environment.

To configure persistent volumes with your CNA deployment, each YAML file must be applied using `kubectl`:

```
root@cli-vm:~# kubectl apply -f <filename>.yaml
```

Dynamic Persistent Volumes

Perform the following steps for a deployment:

Procedure

1 Create a storage class.

2 Define the Persistent Volume Claims.

3 Define the deployment.

Creating a Storage Class for Dynamically Provisioned Volumes

To dynamically provision storage with your pod, you must use a storage class. The following example shows a YAML manifest file to define a storage class. This file uses the shared datastore in the environment and defines the provisioned disks to be thin-provisioned. The value for the name field (*example-thin-disk*) will be used to reference this storage class when defining the persistent value claims.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: example-thin-disk
provisioner: kubernetes.io/vsphere-volume
parameters:
  datastore: RegionA01-ISCSI01-COMP01
  diskformat: thin
  fstype: ext4
```

Defining Persistent Volume Claim for Deployments

For a dynamically created volume, the previously created storage class must be referenced in the Persistent Volume Claim (PVC) definition.

The following is an example PVC YAML manifest file for a dynamically provisioned volume:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-disk-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: example-thin-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
```

Defining a Deployment

The following is an example of a deployment YAML manifest file that references the PVC (this file focuses on the storage aspect of the deployment). This file can work with either the steps to create a static volume, or the steps to create a dynamic volume.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
spec:
  selector:
  ...
  template:
  ...
    spec:
```

```
      ...
      volumes:
      - name: example-data-disk
        persistentVolumeClaim:
          claimName: example-disk-claim
```

Statically Allocated Persistent Volumes

Statically allocating the virtual disks for persistent volumes can be a straight-forward way to make the virtual disks to be attached to the pods. To use statically allocated volumes, perform the following steps:

Procedure

1. Create the appropriate virtual disks.
2. Define the Persistent Volumes.
3. Define the Persistent Volume Claims.
4. Define the deployment.

Creating Virtual Disks for Statically Provisioned Volumes

Instead of using statically provisioned persistent volumes, you must first create the appropriate virtual disks. The following example assumes that the virtual disks are all created in their own directory on the datastore.

Log in to one of the ESXi hosts that has access to the shared storage for the environment (this example creates one virtual disk of 50 GB, thin provisioned):

```
root@esx-01a:~]  cd /vmfs/volumes/RegionA01-ISCSI01-COMP01
root@esx-01a:~]  mkdir kubestor
root@esx-01a:~]  cd kubestor
root@esx-01a:~]  vmkfstools -c 50G -d thin deployment_disk.vmdk
Create:   100% done.
```

Defining a Persistent Volume for Statically Provisioned Volumes

For static volumes, the next step is to define a persistent volume. It is important to match the capacity of the volume (`spec` field in the following example) to the size of the static disk created.

The following is an example of a Persistent Volume YAML file:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-disk
spec:
  capacity:
    storage: 50Gi
  accessModes:
    - ReadWriteOnce
```

```
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume:
    volumePath: "[RegionA01-ISCSI01-COMP01] kubestor/deployment_disk"
    fsType: ext4
```

Defining Persistent Volume Claim for Deployments

The following is an example Persistent Volume Claim (PVC) YAML manifest file for a statically provisioned volume:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-disk-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
```

Defining a Deployment

The following is an example of a deployment YAML manifest file that references the PVC (this file focuses on the storage aspect of the deployment). This file can work with either the steps to create a static volume, or the steps to create a dynamic volume.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
spec:
  selector:
  ...
  template:
  ...
    spec:
      ...
      volumes:
      - name: example-data-disk
        persistentVolumeClaim:
          claimName: example-disk-claim
```

Tips and Troubleshooting

12

This chapter provides tips and troubleshooting information for using Validation-in-Cloud through the VMware Integration Validation (VIVa) service and the VMware Learning Platform (VLP).

- The VLP hosts the lab and also hosts the VMware Hands-on Labs (HOL). For HOL Online FAQs, see https://communities.vmware.com/docs/DOC-24916.
- After you receive access to the lab, ensure that you validate the infrastructure against your requirement.
- From the left pane, you can access any of the other VMs in the lab through the **Consoles** tab.
- For more screen space, click the **Toggle Full Screen Mode** button on the top right of the console.
- Adjust the screen resolution of the Main Console VM to a higher resolution. The web console matches the resolution output by the VM and fits it within the console window.
- Copy and Paste operations between the console window and your desktop are disabled for security reasons. However, you can use the **Send Text** button on the top left of the console to send text from your desktop to the VM.
- If you are a first-time user of the system, use the Guide icon for a guided tour of the console interface.
- To leave your lab and continue later, close the browser window.
- To delete the current lab instance and begin a new lab instance, click the **Reset** button.
- If you encounter issues with your browser, clear cookies from the browser and try again, or switch to a different browser. You can also use the Incognito/Private mode in the Chrome/Firefox browser.
- You might encounter an issue with the VLP when attempting to resume your validation lab, such as a response of "*Maintenance Mode, please try again later*". In this case, close your browser, wait for a few minutes, and reopen the VLP site in a new browser window.
- For VMware support purposes only: from the Web browser on your Main Console VM, browse to http://10.148.172.7. This will return an internal IP address. Also, an internal IP Address is shown on the Main Console VM desktop. Report this IP address in the DCPN ticket as part of your submission case description.

This chapter includes the following topics:

- Accessing NSX-T Manager
- Accessing Kubernetes Dashboard

Troubleshooting PKS Cluster Creation

This section provides information about how to troubleshoot issues when creating a PKS cluster.

Restart PKS Instance

To make sure that all services started on pivotal-container-service work correctly, restart the PKS instance from Bosh.

Procedure

1 Open PuTTY session to CLI-VM. See Accessing CLI VM.

2 On the Command Prompt, run the following commands:

```
root@cli-vm:~# source A-BOSH.env
```

```
root@cli-vm:~# bosh vms
```

```
Using environment '10.1.1.3' as client 'ops_manager'
 …
Deployment 'harbor-container-registry-8d51221a98ebb37124f6'
harbor-app/5855fc38-e432-4585-818b-013bd2b839b3  running        AZ-MGMT  10.1.1.5  vm-
bccb123b-3d90-410b-97d3-fe6c011468a5  xlarge.disk  true
1 vms

Deployment 'pivotal-container-service-0b648dc45ece40cd92e1'
pivotal-container-service/0d785096-429e-4267-b6b6-89b5d0295461  running        AZ-MGMT  10.1.1.4
vm-4915010b-b6d7-4ce1-8fda-693efe81ba3e  large    true
1 vms
```

3 Restart the PKS instance.

For example, if the service and the service ID are as follows:

pivotal-container-service/0d785096-429e-4267-b6b6-89b5d0295461 instance from deployment '*pivotal-container-service-0b648dc45ece40cd92e1*', run this command:

```
root@cli-vm:~# bosh restart pivotal-container-service/0d785096-429e-4267-
b6b6-89b5d0295461 -d pivotal-container-service-0b648dc45ece40cd92e1
```

```
Using environment '10.1.1.3' as client 'ops_manager'
Using deployment 'pivotal-container-service-0b648dc45ece40cd92e1'

Continue? [yN]: y
Task 79281 | 21:07:07 | Updating instance pivotal-container-service:
pivotal-container-service/0d785096-429e-4267-b6b6-89b5d0295461 (0) (canary) (00:02:41)
```

Get UAAC Token and Log In to PKS

The original UAAC token expires after 24 hours and it might prevent you from logging in to PKS. To get a UAAC token, perform the following steps:

Procedure

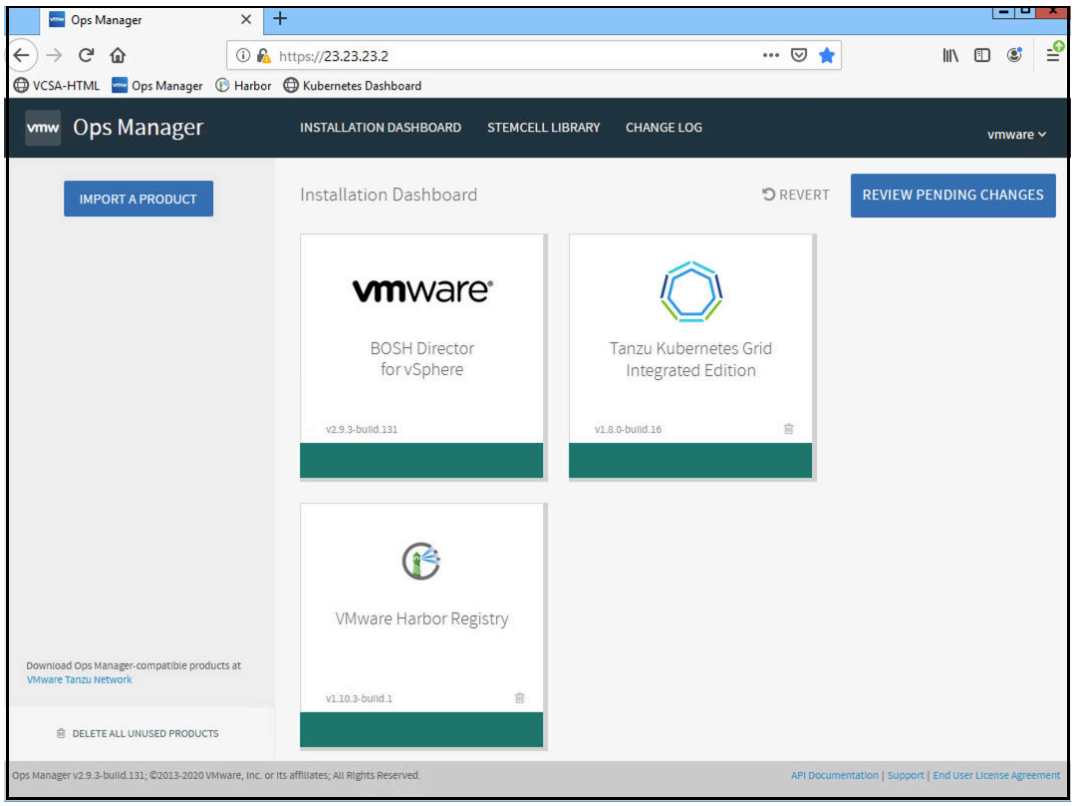
1 Open PuTTY session to CLI-VM. See Accessing CLI VM.

2 Set the UAAC target.

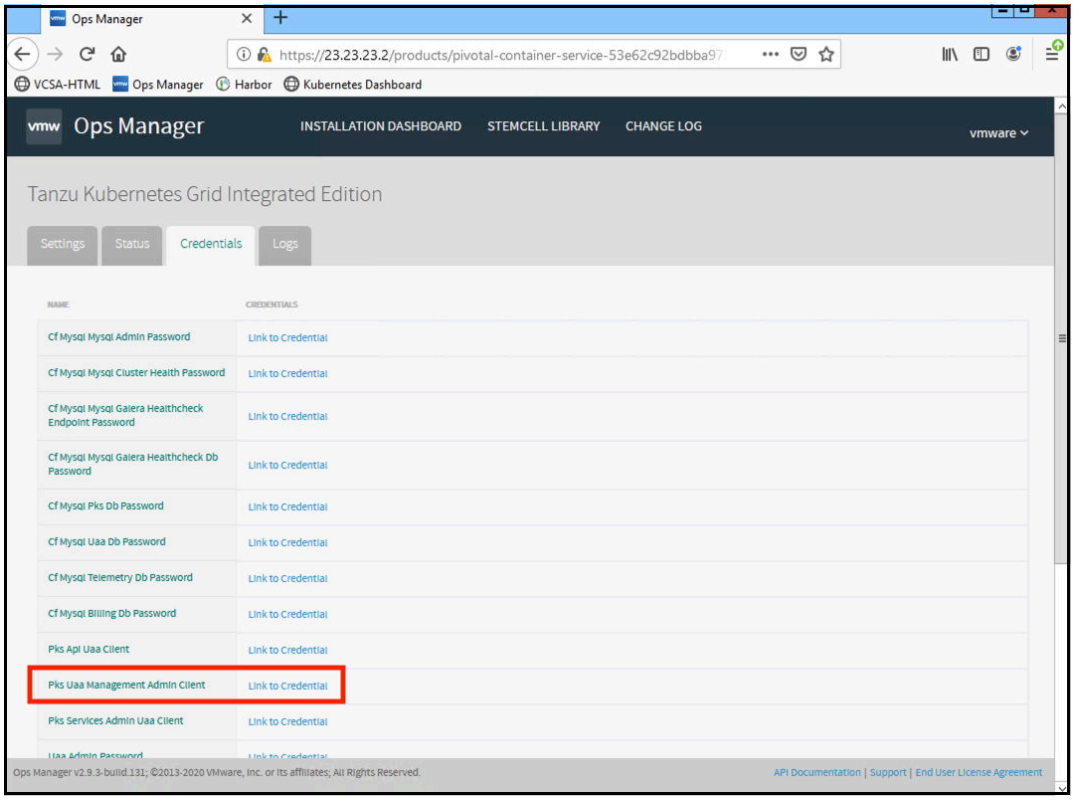
```
root@cli-vm:~# uaac target https://uaa.corp.local:8443 --skip-ssl-validation
```

```
Target: https://uaa.corp.local:8443
Context: admin, from client admin
```

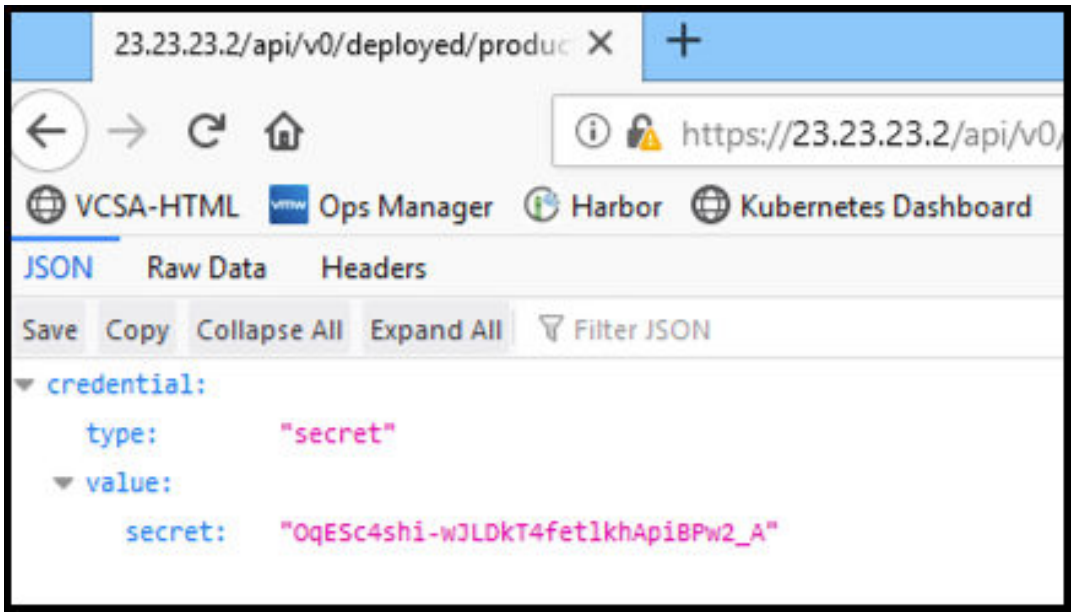
3 Log in to PCF Ops Manager. See Accessing PCF Ops Manager.



4 Go to **Tanzu Kubernetes Grid Integrated Edition > Credentials > Pks Uaa Management Admin Client** and click **Link to Credential**.



5 Copy the `secret` value (*OqESc4shi-wJLDkT4fetlkhApiBPw2_A*) and paste it on a clipboard.



Note The `secret` value depends on the TKGI lab version.

6 Go to the CLI-VM and get the UAAC token by running this command:

```
root@cli-vm:~# uaac token client get admin -s <secret_value>
```

For example:

```
root@cli-vm:~# uaac token client get admin -s OqESc4shi-wJLDkT4fetlkhApiBPw2_A
```

```
Successfully fetched token via client credentials grant.
Target: https://uaa.corp.local:8443
Context: admin, from client admin
```

7 Log in to PKS

```
root@cli-vm:~# pks login -a uaa.corp.local -u vmware -p VMware1! -k
```

```
API Endpoint: uaa.corp.local
User: vmware
```

Accessing PCF Ops Manager

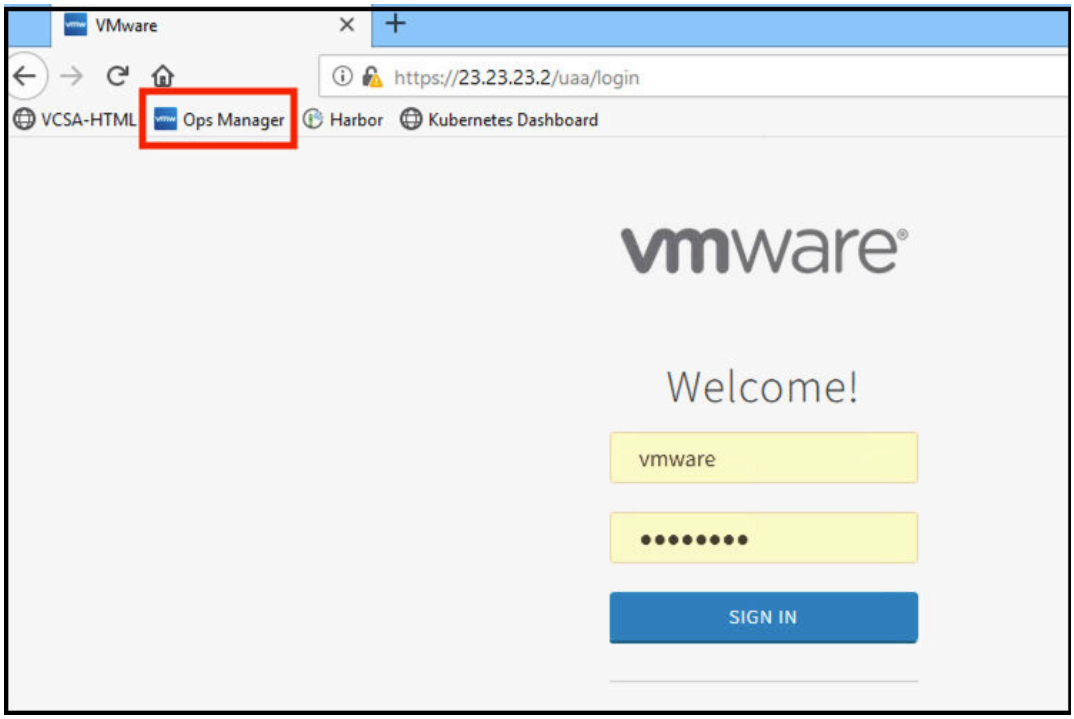
The PCF Ops Manager can be accessed using the following details provided. The IP address provided is constant and will not change.

1 From the control center Windows VM, open Firefox, and click the **Ops Manager** bookmark (https://23.23.23.2/uaa/login).

Login credentials:

- User name: *vmware*
- Password: *VMware1!*

2 If you are asked for passphrase before providing the username or password, enter the passphrase as *VMware1!*

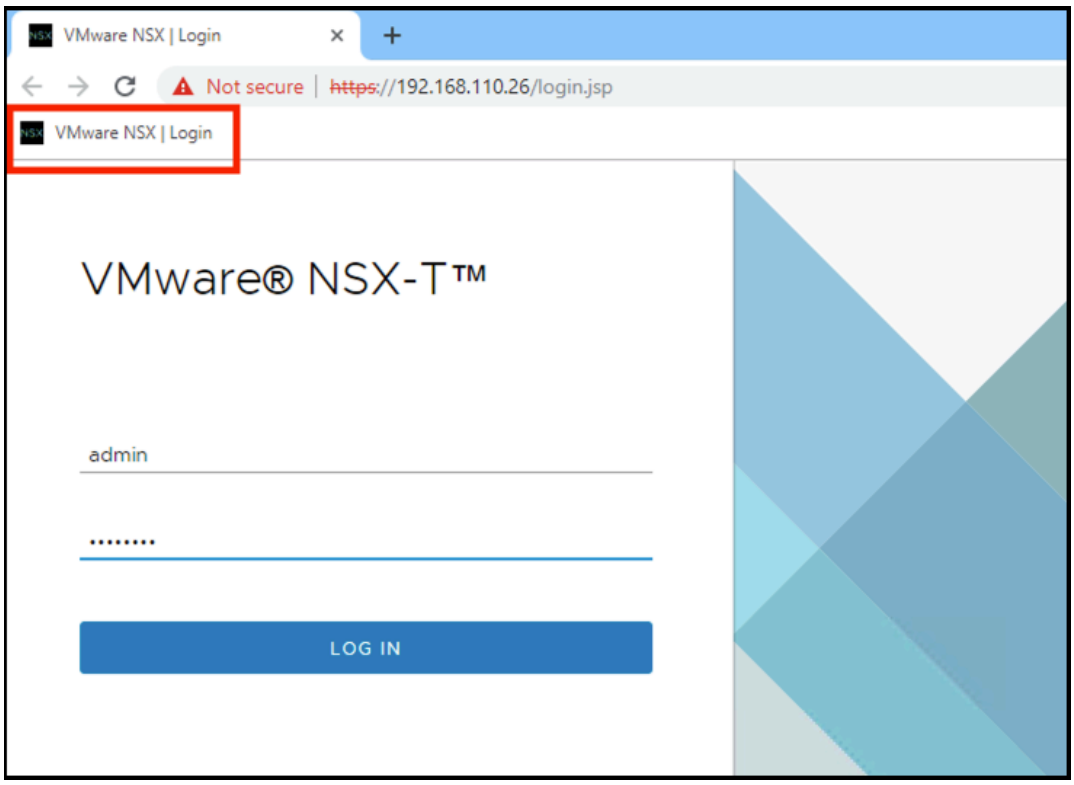


Accessing NSX-T Manager

From the control center Windows VM, open Google Chrome, and click the **NSX-Mgr** bookmark (`https://192.168.110.26`).

Login credentials:

- User name: *admin*
- Password: *VMware1!*



Accessing Kubernetes Dashboard

To access the Kubernetes dashboard, perform the following steps:

Procedure

1 Start a PuTTY session to root@cli-vm.

   a Go to **Expand Connection > SSH > Tunnels**.

   b Add a forwarded port with source port being 8001 and destination being localhost:8001. Click **Add**.

   c Click **Open**. A prompt asking you to enter the password for `root@192.168.110.7` appears. Enter the password (*VMware1!*).

2 Create a cluster (*test-cluster*). For details, see Accessing TKGI Environment and Creating a PKS Cluster sections.

```
root@cli-vm:~# pks create-cluster test-cluster --external-hostname ext-test-cluster.corp.local --plan small --num-nodes 3
```

Note The number of nodes and plan configuration might differ according to your application requirements.

3 Configure the cluster (*test-cluster*) to be used with your application.

```
root@cli-vm:~# pks cluster test-cluster
```

Check output for Kubernetes Master IP(s).

Edit the `/etc/hosts` file and add the line '**`Kubernetes_Master_IP(s) ext-test-cluster.corp.local`**'.

```
root@cli-vm:~# vi /etc/hosts
```

4 Deploy Kubernetes Dashboard.

```
root@cli-vm:~# kubectl apply -f https://raw.githubusercontent.com/kubernetes/
dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml
```

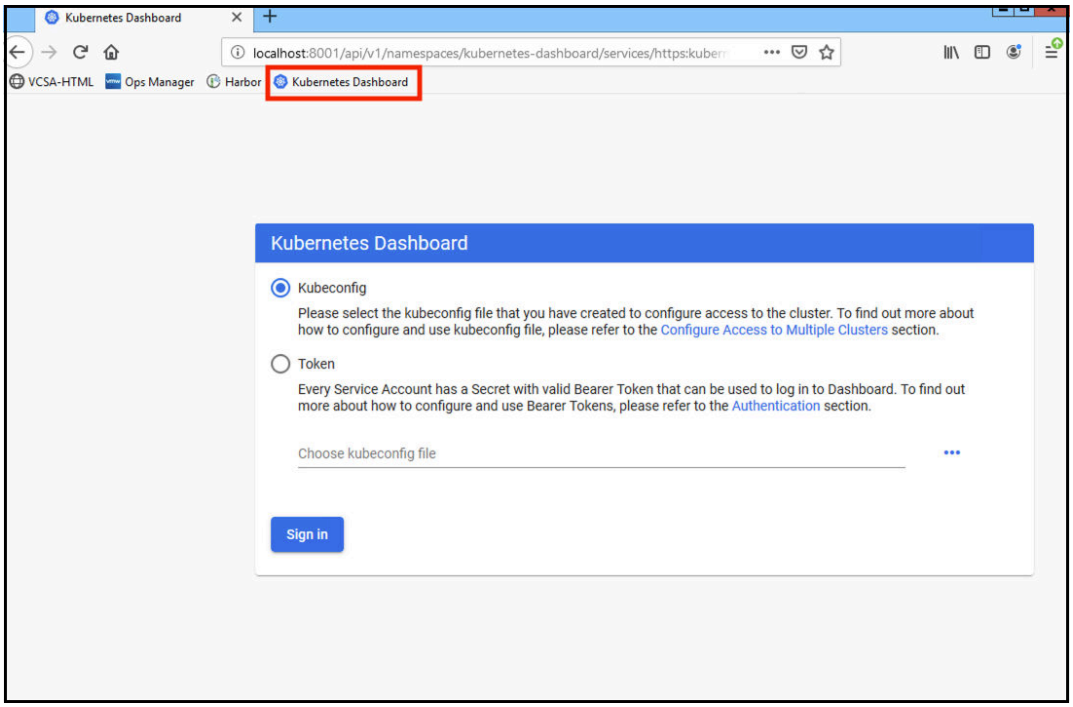
5 Start `kubectl` proxy.

```
root@cli-vm:~# kubectl proxy
```

```
Starting to serve on 127.0.0.1:8001
```

6 From the control center Windows VM, open Firefox, and click the **Kubernetes Dashboard** bookmark.

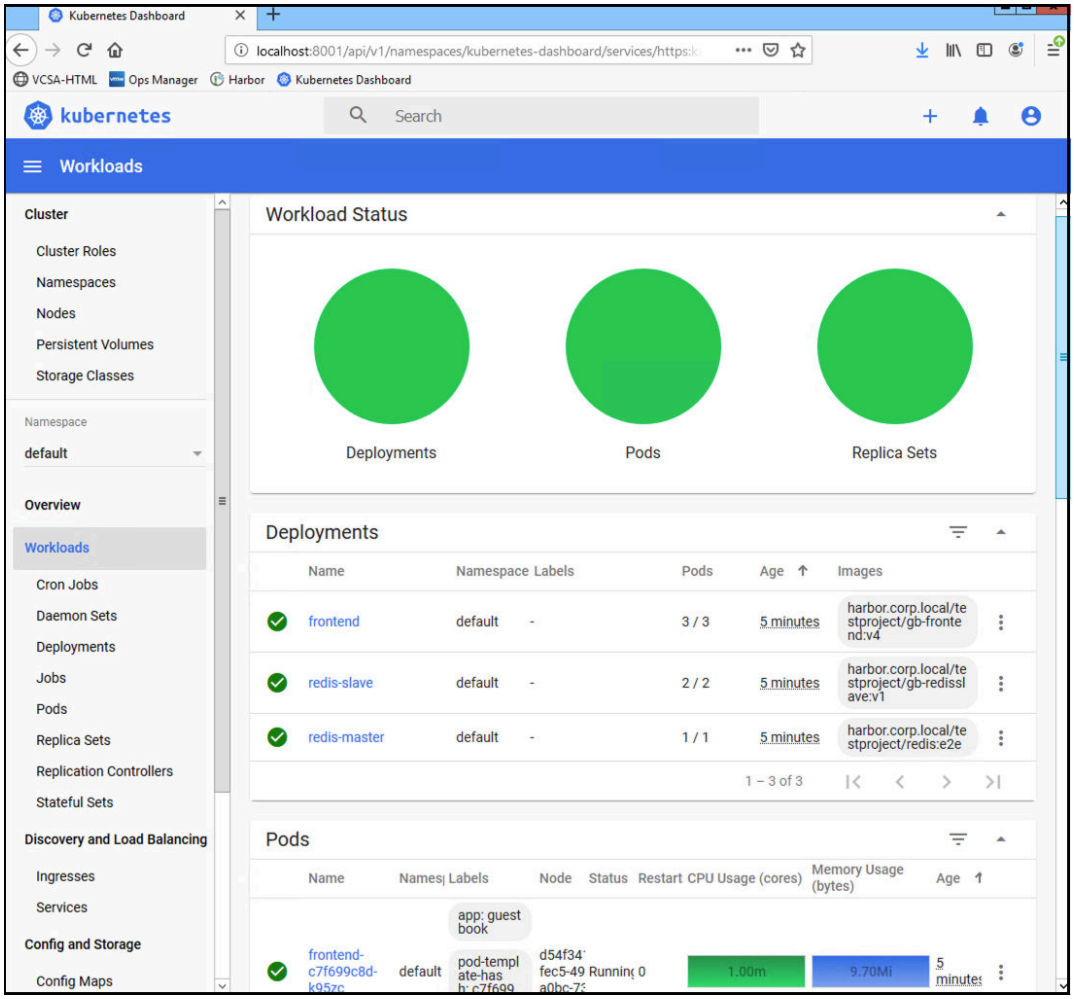
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#!/login



7 From the **Kubernetes Dashboard** selection, select **Kubeconfig**, and provide the `config` file path. The `config` file is located at `/root/.kube/config` on the CLI-VM. Copy the file from the CLI-VM to the control center Windows VM using WinSCP.

8 Click **SIGN IN**.

9 Select your application-related Namespace from the drop-down menu in the Dashboard and check whether your application-related information is shown in the Kubernetes Dashboard correctly in green.



Appendix A: Sample Docker Bench Security Script Output

13

```
# ------------------------------------------------------------------------------
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# ------------------------------------------------------------------------------

Initializing Fri Nov 30 16:25:57 PST 2018



[INFO] 1 - Host Configuration
[WARN] 1.1  - Ensure a separate partition for containers has been created
[NOTE] 1.2  - Ensure the container host has been Hardened
[INFO] 1.3  - Ensure Docker is up to date
[INFO]      * Using 18.06.1, verify is it up to date as deemed necessary
[INFO]      * Your operating system vendor may provide support and security maintenance for Docker
[INFO] 1.4  - Ensure only trusted users are allowed to control Docker daemon
[INFO]      * docker:x:999:
[WARN] 1.5  - Ensure auditing is configured for the Docker daemon
[WARN] 1.6  - Ensure auditing is configured for Docker files and directories - /var/lib/docker
[WARN] 1.7  - Ensure auditing is configured for Docker files and directories - /etc/docker
[WARN] 1.8  - Ensure auditing is configured for Docker files and directories - docker.service
[WARN] 1.9  - Ensure auditing is configured for Docker files and directories - docker.socket
[WARN] 1.10  - Ensure auditing is configured for Docker files and directories - /etc/default/docker
[INFO] 1.11  - Ensure auditing is configured for Docker files and directories - /etc/docker/
daemon.json
[INFO]      * File not found
[WARN] 1.12  - Ensure auditing is configured for Docker files and directories - /usr/bin/docker-
containerd
[WARN] 1.13  - Ensure auditing is configured for Docker files and directories - /usr/bin/docker-runc



[INFO] 2 - Docker daemon configuration
[WARN] 2.1  - Ensure network traffic is restricted between containers on the default bridge
[PASS] 2.2  - Ensure the logging level is set to 'info'
[PASS] 2.3  - Ensure Docker is allowed to make changes to iptables
[PASS] 2.4  - Ensure insecure registries are not used
[PASS] 2.5  - Ensure aufs storage driver is not used
[INFO] 2.6  - Ensure TLS authentication for Docker daemon is configured
```

```
[INFO]      * Docker daemon not listening on TCP
[INFO] 2.7  - Ensure the default ulimit is configured appropriately
[INFO]      * Default ulimit doesn't appear to be set
[WARN] 2.8  - Enable user namespace support
[PASS] 2.9  - Ensure the default cgroup usage has been confirmed
[PASS] 2.10  - Ensure base device size is not changed until needed
[WARN] 2.11  - Ensure that authorization for Docker client commands is enabled
[WARN] 2.12  - Ensure centralized and remote logging is configured
[INFO] 2.13  - Ensure operations on legacy registry (v1) are Disabled (Deprecated)
[WARN] 2.14  - Ensure live restore is Enabled
[WARN] 2.15  - Ensure Userland Proxy is Disabled
[PASS] 2.16  - Ensure daemon-wide custom seccomp profile is applied, if needed
[PASS] 2.17  - Ensure experimental features are avoided in production
[WARN] 2.18  - Ensure containers are restricted from acquiring new privileges


[INFO] 3 - Docker daemon configuration files
[PASS] 3.1  - Ensure that docker.service file ownership is set to root:root
[PASS] 3.2  - Ensure that docker.service file permissions are set to 644 or more restrictive
[PASS] 3.3  - Ensure that docker.socket file ownership is set to root:root
[PASS] 3.4  - Ensure that docker.socket file permissions are set to 644 or more restrictive
[PASS] 3.5  - Ensure that /etc/docker directory ownership is set to root:root
[PASS] 3.6  - Ensure that /etc/docker directory permissions are set to 755 or more restrictive
[PASS] 3.7  - Ensure that registry certificate file ownership is set to root:root
[WARN] 3.8  - Ensure that registry certificate file permissions are set to 444 or more restrictive
[WARN]      * Wrong permissions for /etc/docker/certs.d/
[INFO] 3.9  - Ensure that TLS CA certificate file ownership is set to root:root
[INFO]      * No TLS CA certificate found
[INFO] 3.10  - Ensure that TLS CA certificate file permissions are set to 444 or more restrictive
[INFO]      * No TLS CA certificate found
[INFO] 3.11  - Ensure that Docker server certificate file ownership is set to root:root
[INFO]      * No TLS Server certificate found
[INFO] 3.12  - Ensure that Docker server certificate file permissions are set to 444 or more
restrictive
[INFO]      * No TLS Server certificate found
[INFO] 3.13  - Ensure that Docker server certificate key file ownership is set to root:root
[INFO]      * No TLS Key found
[INFO] 3.14  - Ensure that Docker server certificate key file permissions are set to 400
[INFO]      * No TLS Key found
[PASS] 3.15  - Ensure that Docker socket file ownership is set to root:docker
[PASS] 3.16  - Ensure that Docker socket file permissions are set to 660 or more restrictive
[INFO] 3.17  - Ensure that daemon.json file ownership is set to root:root
[INFO]      * File not found
[INFO] 3.18  - Ensure that daemon.json file permissions are set to 644 or more restrictive
[INFO]      * File not found
[PASS] 3.19  - Ensure that /etc/default/docker file ownership is set to root:root
[PASS] 3.20  - Ensure that /etc/default/docker file permissions are set to 644 or more restrictive


[INFO] 4 - Container Images and Build File
[WARN] 4.1  - Ensure a user for the container has been created
[WARN]      * Running as root: competent_feynman
[NOTE] 4.2  - Ensure that containers use trusted base images
[NOTE] 4.3  - Ensure unnecessary packages are not installed in the container
[NOTE] 4.4  - Ensure images are scanned and rebuilt to include security patches
```

```
[PASS] 4.5  - Ensure Content trust for Docker is Enabled
[WARN] 4.6  - Ensure HEALTHCHECK instructions have been added to the container image
[WARN]      * No Healthcheck found: [gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4]
[WARN]      * No Healthcheck found: [gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4]
[INFO] 4.7  - Ensure update instructions are not use alone in the Dockerfile
[INFO]      * Update instruction found: [gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4]
[INFO]      * Update instruction found: [gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4]
[NOTE] 4.8  - Ensure setuid and setgid permissions are removed in the images
[INFO] 4.9  - Ensure COPY is used instead of ADD in Dockerfile
[INFO]      * ADD in image history: [gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4]
[INFO]      * ADD in image history: [gcr.io/google-samples/gb-frontend:v4 harbor.corp.local/
testproject/gb-frontend:v4]
[NOTE] 4.10  - Ensure secrets are not stored in Dockerfiles
[NOTE] 4.11  - Ensure verified packages are only Installed


[INFO] 5 - Container Runtime
[PASS] 5.1  - Ensure AppArmor Profile is Enabled
[WARN] 5.2  - Ensure SELinux security options are set, if applicable
[WARN]      * No SecurityOptions Found: competent_feynman
[PASS] 5.3  - Ensure Linux Kernel Capabilities are restricted within containers
[PASS] 5.4  - Ensure privileged containers are not used
[PASS] 5.5  - Ensure sensitive host system directories are not mounted on containers
[PASS] 5.6  - Ensure ssh is not run within containers
[PASS] 5.7  - Ensure privileged ports are not mapped within containers
[NOTE] 5.8  - Ensure only needed ports are open on the container
[PASS] 5.9  - Ensure the host's network namespace is not shared
[WARN] 5.10  - Ensure memory usage for container is limited
[WARN]      * Container running without memory restrictions: competent_feynman
[WARN] 5.11  - Ensure CPU priority is set appropriately on the container
[WARN]      * Container running without CPU restrictions: competent_feynman
[WARN] 5.12  - Ensure the container's root filesystem is mounted as read only
[WARN]      * Container running with root FS mounted R/W: competent_feynman
[PASS] 5.13  - Ensure incoming container traffic is binded to a specific host interface
[WARN] 5.14  - Ensure 'on-failure' container restart policy is set to '5'
[WARN]      * MaximumRetryCount is not set to 5: competent_feynman
[PASS] 5.15  - Ensure the host's process namespace is not shared
[PASS] 5.16  - Ensure the host's IPC namespace is not shared
[PASS] 5.17  - Ensure host devices are not directly exposed to containers
[INFO] 5.18  - Ensure the default ulimit is overwritten at runtime, only if needed
[INFO]      * Container no default ulimit override: competent_feynman
[PASS] 5.19  - Ensure mount propagation mode is not set to shared
[PASS] 5.20  - Ensure the host's UTS namespace is not shared
[PASS] 5.21  - Ensure the default seccomp profile is not Disabled
[NOTE] 5.22  - Ensure docker exec commands are not used with privileged option
[NOTE] 5.23  - Ensure docker exec commands are not used with user option
[PASS] 5.24  - Ensure cgroup usage is confirmed
[WARN] 5.25  - Ensure the container is restricted from acquiring additional privileges
[WARN]      * Privileges not restricted: competent_feynman
[WARN] 5.26  - Ensure container health is checked at runtime
```

```
[WARN]      * Health check not set: competent_feynman
[INFO] 5.27  - Ensure docker commands always get the latest version of the image
[WARN] 5.28  - Ensure PIDs cgroup limit is used
[WARN]      * PIDs limit not set: competent_feynman
[INFO] 5.29  - Ensure Docker's default bridge docker0 is not used
[INFO]      * Container in docker0 network: competent_feynman
[PASS] 5.30  - Ensure the host's user namespaces is not shared
[PASS] 5.31  - Ensure the Docker socket is not mounted inside any containers


[INFO] 6 - Docker Security Operations
[INFO] 6.1  - Avoid image sprawl
[INFO]      * There are currently: 1 images
[INFO] 6.2  - Avoid container sprawl
[INFO]      * There are currently a total of 1 containers, with 1 of them currently running


[INFO] 7 - Docker Swarm Configuration
[PASS] 7.1  - Ensure swarm mode is not Enabled, if not needed
[PASS] 7.2  - Ensure the minimum number of manager nodes have been created in a swarm (Swarm mode not
enabled)
[PASS] 7.3  - Ensure swarm services are binded to a specific host interface (Swarm mode not enabled)
[PASS] 7.4  - Ensure data exchanged between containers are encrypted on different nodes on the
overlay network
[PASS] 7.5  - Ensure Docker's secret management commands are used for managing secrets in a Swarm
cluster (Swarm mode not enabled)
[PASS] 7.6  - Ensure swarm manager is run in auto-lock mode (Swarm mode not enabled)
[PASS] 7.7  - Ensure swarm manager auto-lock key is rotated periodically (Swarm mode not enabled)
[PASS] 7.8  - Ensure node certificates are rotated as appropriate (Swarm mode not enabled)
[PASS] 7.9  - Ensure CA certificates are rotated as appropriate (Swarm mode not enabled)
[PASS] 7.10  - Ensure management plane traffic has been separated from data plane traffic (Swarm mode
not enabled)

[INFO] Checks: 105
[INFO] Score: 19
```

Appendix B: TKGI Software BOM

14

This chapter includes the following topics:

TKGI Software BOM – 1.7 Lab

The following table provides the version and build information of different TKGI setup components. This lab is based on the TKGI 1.7 GA version.

Product	Version	Build
vSphere (ESXi)	6.7 U2	ESXi - 13006603
vSphere (VC)	6.7 GA	vCenter - 8217866
NSX-T	2.5.1	2.5.1.0.0.15314288
TKGI	1.7.0	1.7.0-build.26
PCF Ops Manager	2.9.0	v2.9.0-build.106
Harbor	1.10.1	v1.10.1-build.7
Bosh	2.9.0	v2.9.0-build.106
Kubernetes	1.16.7	

TKGI Software BOM – 1.8 Lab

The following table provides the version and build information of different TKGI setup components. This lab is based on the TKGI 1.8 GA version.

Product	Version	Build
vSphere (ESXi)	6.7 U3	ESXi -14320388
vSphere (VC)	6.7 GA	vCenter - 8217866
NSX-T	2.5.1	2.5.1.0.0.15314288
TKGI	1.8.0	1.8.0-build.16
PCF Ops Manager	2.9.3	v2.9.3-build.131

Product	Version	Build
Harbor	1.10.3	v1.10.3-build.1
Bosh	2.9.3	v2.9.3-build.131
Kubernetes	1.17.5	

Appendix C: TKGI Cluster Plans

15

Each master and worker nodes have the following plans.

TKGI Cluster Plans – 1.7 and 1.8 Labs

The following table provides information about the cluster plans for the TKGI 1.7 and 1.8 labs.

Plan Settings Name	Plan	CPU	Memory (GB)	Disk Size (GB)	Number of Nodes	
					Master	Worker
Plan 1	small	2	4	32	1	3
Plan 3	large	8	8	32	1	3
Plan 4	xlarge	4 - master 4 - worker	16 - master 16 - worker	32 - master 128 - worker	1	2
Plan 5	small6workers	2	4	32	1	6
Plan 7	large6workers	8	8	32	1	6
Plan 9	small3masters	2	4	32	3	4
Plan 10	large3masters	8	8	32	3	4