14

Building a Cloud-Native Use Case on a Hybrid Cloud Environment

It has been a wonderful journey so far! We walked through so much content in this book already, from OpenShift architecture to Pipelines, GitOps, and multi-cloud tools! We are now reaching our main goal with this book, which is helping you to make the best decisions and implement a good hybrid/multi-cloud strategy for your OpenShift footprint. To wrap up this book with helpful content, we will make a comprehensive review using a practical approach to building and deploying an application using most features we covered during this book: OpenShift Pipelines (Tekton), OpenShift GitOps (ArgoCD), Advanced Cluster Management, Quay, and Advanced Cluster Security.

Therefore, you will find the following in this chapter:

- Use case description
- Application build using OpenShift Pipelines and S2I
- Application deployment using OpenShift Pipelines and GitOps
- Adding security checks in the building and deployment process
- Provisioning and managing multiple clusters
- Deploying an application into multiple clusters

So, what are we waiting for? Let's play now!

NOTE

The source code used in this chapter is available at **https://github.com/PacktPublishing/OpenShift-Multi-Cluster-Management-Handbook/tree/main/chapter14**.

Use case description

To be a bit closer to what you see in the real world, this time we are going to use a Java application, using **Quarkus**, which is a great option to build modern, cloud-native applications with Java. Look at the references in the *Further reading* section of this chapter for more information about **Quarkus**.

Our application source code was extracted from the *Getting started with Quarkus* sample; see reference for it in the *Further reading* section of this chapter. During this chapter, we will create a CI/CD pipeline that will do the following:

1. Build the application using s2i to generate Java binaries.
2. Push the container image to Quay.
3. Run a security scan on the image using Advanced Cluster Security.
4. Deploy the application on the local cluster using ArgoCD.
5. Deploy the application on multiple remote clusters using ArgoCD and Advanced Cluster Management.

We are going to use Advanced Cluster Management to make all OpenShift clusters compliant with a standard policy we defined for them as well. For the sake of learning and simplicity, we are going to build the pipeline and other objects in sequential phases, like building blocks that are added to build a house.

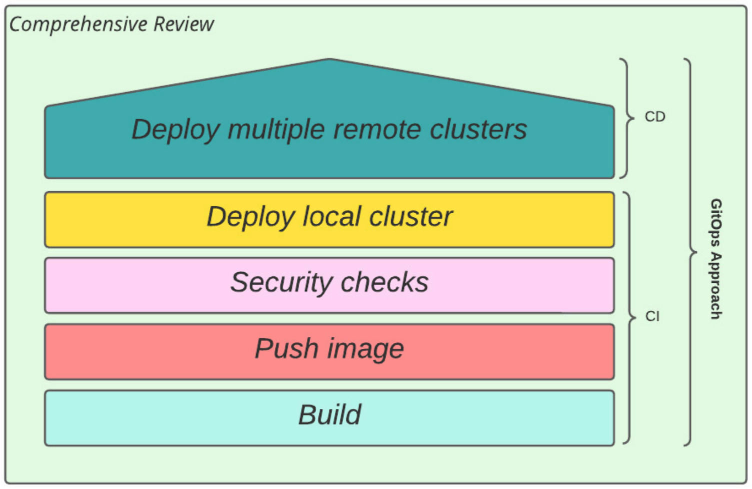


Figure 14.1 – Comprehensive review - Building blocks

We assume that you have access to an OpenShift cluster, which we will call the Hub cluster, with enough resources and with the following tools already installed:

- OpenShift Pipelines
- OpenShift GitOps
- Advanced Cluster Management
- Advanced Cluster Security
- Quay
- The **oc** command line installed and connected to the Hub cluster

We will also deploy additional single node clusters on AWS to be used as managed remote clusters, to exercise the application deployment into multiple clusters. If you haven't installed these tools yet, refer to the installation process of each from *Chapters 9* to *13* of this book.

The source code used in this chapter is available at our GitHub repository: **https://github.com/PacktPublishing/OpenShift-Multi-Cluster-Management-Handbook/tree/main/chapter14**.

Let's start by digging into the first building block: the application build.

Application build using OpenShift Pipelines and S2I

For this step, we are going to use the **quarkus-build** pipeline that you can find in the **chapter14/Build/Pipeline/quarkus-build.yaml** file. This pipeline is very straightforward and explained in the following diagram:

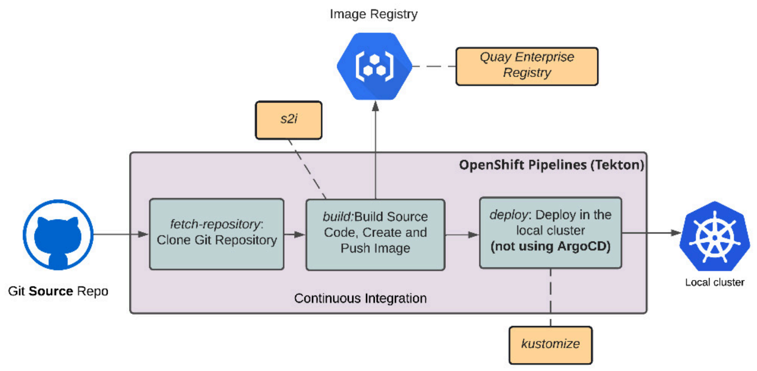


Image Registry
Quay Enterprise Registry
s2i
OpenShift Pipelines (Tekton)
Git Source Repo
fetch-repository: Clone Git Repository
build: Build Source Code, Create and Push Image
deploy: Deploy in the local cluster (not using ArgoCD)
Local cluster
Continuous Integration
kustomize

Figure 14.2 – Pipeline to build a Java Quarkus application

In this pipeline we are using pre-existing ClusterTasks to do all the work:

- **git-clone**: Used to clone the Git repository.
- **s2i-java**: Build the Java source code using S2I and Buildah to generate the image and push it to the Quay registry. S2I is a very convenient way to build code from many different languages, such as Java, Python, Node.js, and others. See the *Further reading* section of this chapter for more information about S2I.
- **openshift-client**: Used to run the manifests that deploy the application. Application manifests use **Kustomize** to declare the Kubernetes manifest. We covered **Kustomize** in ***Chapter 10***, *OpenShift GitOps – ArgoCD*, of this book; if you didn't read it yet, we strongly recommend you to do so now and then get back here to perform the steps in this chapter.

Now let's create and run this pipeline. If you haven't done it yet, fork this repository to your GitHub account: **https://github.com/PacktPublishing/OpenShift-Multi-Cluster-Management-Handbook**. After you forked it, follow the instructions in this section to create and run this pipeline:

1. Clone the repository in your machine:

```
$ GITHUB_USER=<your_user>
```

```
$ git clone https://github.com/PacktPublishing/OpenShift-Multi-Cluster-Management-Handb
```

2. Run the following script and follow the instructions to change the references from the original repository (**PacktPublishing**) to your forked repository:

```
$ cd OpenShift-Multi-Cluster-Management-Handbook/chapter14
```

```
$ ./change-repo-urls.sh
```

```
# Go to the Build folder
```

```
$ cd Build
```

3. Run the following commands to create the namespace and the pipeline:

```
$ oc apply -f Pipeline/namespace.yaml
```

```
$ oc apply -f Pipeline/quarkus-build-pi.yaml
```

4. You should be able to see the pipeline in the OpenShift console, in **Pipelines** | **Pipelines** | **Project: chap14-review-cicd**, as you can see in the following screenshot:

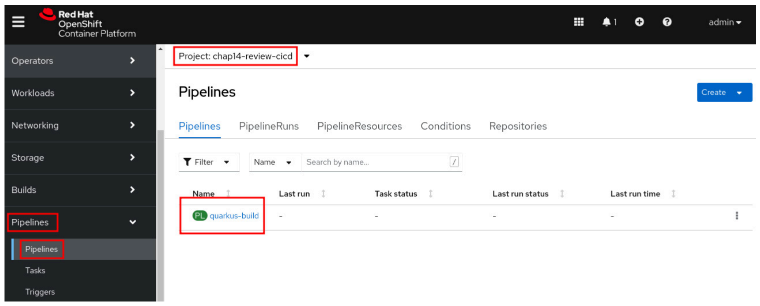


Figure 14.3 – Build pipeline created

5. You can now run the pipeline either using the web interface or through the terminal. To do so from your terminal, run the following command:

```
$ oc create -f PipelineRun/quarkus-build-pr.yaml
```

NOTE ABOUT IMAGE REGISTRY

This pipeline uses an external registry to push the resulting image. To be able to push an image to the registry, you need to link a secret that contains the registry credentials with the `pipeline` *ServiceAccount. If you don't do it before running the pipeline, you will notice that it will fail in the* `build` *task. We are using Quay in this chapter, but you can use any external image registry, such as Nexus, Amazon Elastic Container Registry, Docker Hub, or any other. If you decide to use Quay, you need to create a robot account,* *give it write permissions in the image repository, and import the secret to the namespace. Next, you will find out how to do it.*

See next how to configure your Quay repository and link the credentials to the **`pipeline`** ServiceAccount.

Configuring the image registry

After you have created a new repository on Quay, follow these steps to configure it:

1. Access the **Settings** tab of the repository and access the **Create robot account** link in the **User and Robot Permissions** section:

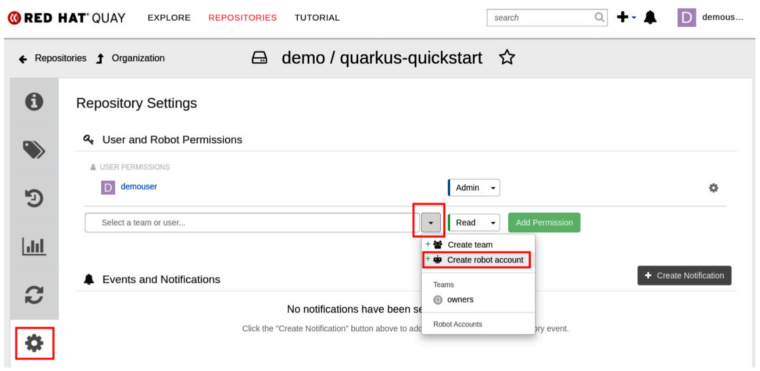
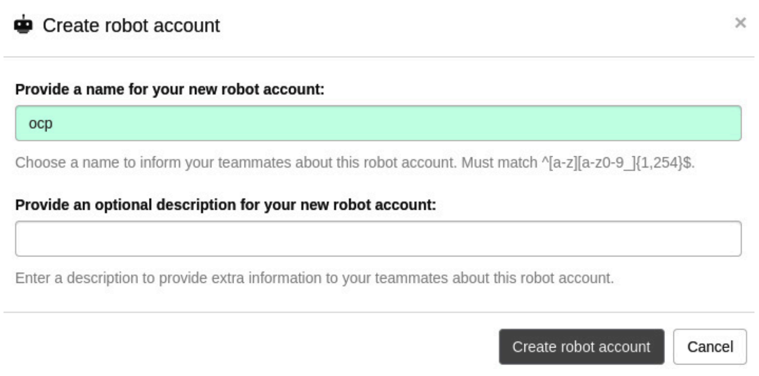


Figure 14.4 – Create a robot account

2. Give it any name and click on the **Create robot account** button:



Create robot account

Provide a name for your new robot account:

ocp

Choose a name to inform your teammates about this robot account. Must match ^[a-z][a-z0-9_]{1,254}$.

Provide an optional description for your new robot account:

Enter a description to provide extra information to your teammates about this robot account.

Create robot account | Cancel

Figure 14.5 – Create a robot account

3. Then, change the permission to **Write** and click on **Add Permission**:

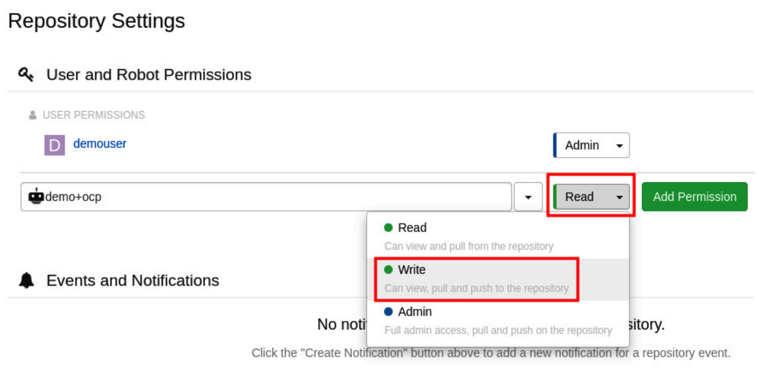

Repository Settings
User and Robot Permissions
USER PERMISSIONS
demouser
Admin
demo+ocp
Read
Add Permission
Read
Can view and pull from the repository
Write
Can view, pull and push to the repository
Admin
Full admin access, pull and push on the repository
Events and Notifications
No noti
sitory.
Click the "Create Notification" button above to add a new notification for a repository event.

Figure 14.6 – Set Write permissions

4. Click on the robot account link to download the secret that we will use to link with the pipeline ServiceAccount:

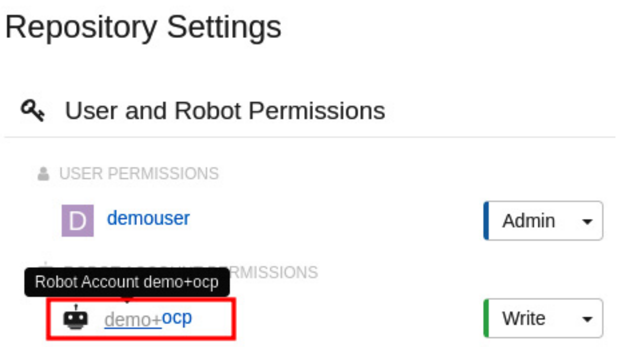
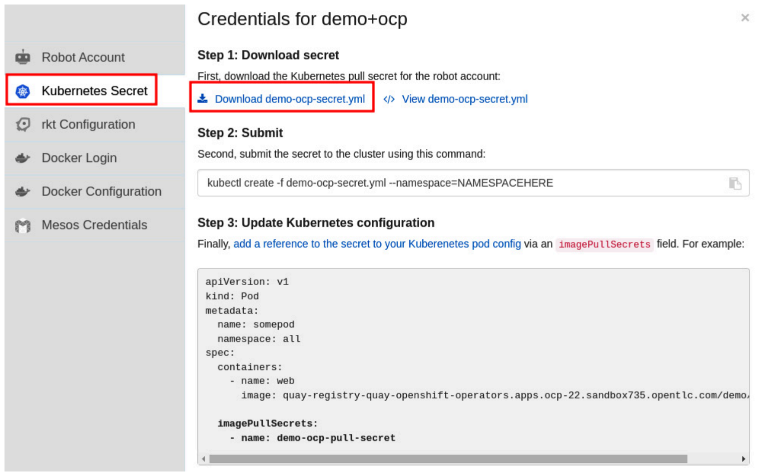


Figure 14.7 – Robot account

5. Download the secret by clicking on the **Download <robot-name>-secret.yml** link in the **Kubernetes Secret** tab:



Credentials for demo+ocp

Step 1: Download secret

First, download the Kubernetes pull secret for the robot account:

Download demo-ocp-secret.yml   View demo-ocp-secret.yml

Step 2: Submit

Second, submit the secret to the cluster using this command:

```
kubectl create -f demo-ocp-secret.yml --namespace=NAMESPACEHERE
```

Step 3: Update Kubernetes configuration

Finally, add a reference to the secret to your Kuberenetes pod config via an `imagePullSecrets` field. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: somepod
  namespace: all
spec:
  containers:
    - name: web
      image: quay-registry-quay-openshift-operators.apps.ocp-22.sandbox735.opentlc.com/demo/

  imagePullSecrets:
    - name: demo-ocp-pull-secret
```

Figure 14.8 – Download Quay credentials

With the secret YAML file in hand, you can proceed with its creation on OpenShift. See next how to do it.

Linking image registry credentials

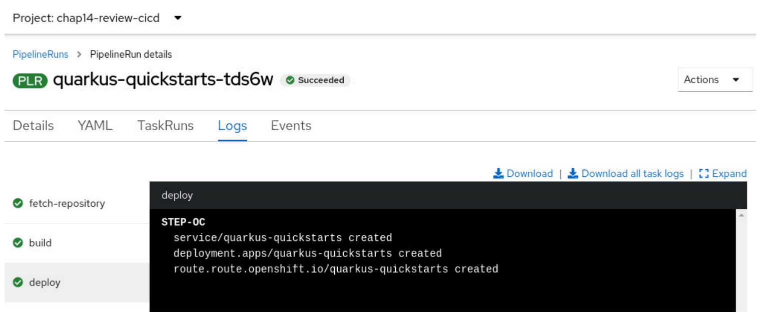
Now that we already have the secret file in our workspace, run the following commands to create the secret and link it to the pipeline ServiceAccount. Alternatively, you can just run the **`link-image-registry-secret.sh`** script from the GitHub repository that we prepared for you, which will do this same process:

```
$ SECRET_FILE=<secret-file-which-contains-image-registry-credentials>
$ SECRET_NAME=<secret-name>
$ oc create -f $SECRET_FILE --namespace=chap14-review-cicd
$ oc patch serviceaccount pipeline -p '{"secrets": [{"name": "$SECRET_NAME"}]}' --namespac
$ oc secrets link default $SECRET_NAME --for=pull -n chap14-review-cicd
```

If you faced the error mentioned in the build task of the pipeline, you can now run it again by running the following command:

```
$ oc create -f PipelineRun/quarkus-build-pr.yaml
```

Now you should see the pipeline finishing successfully, as you can see in the following screenshot:


Project: chap14-review-cicd
PipelineRuns > PipelineRun details
PLR quarkus-quickstarts-tds6w Succeeded
Actions
Details YAML TaskRuns Logs Events
Download | Download all task logs | Expand
fetch-repository
build
deploy

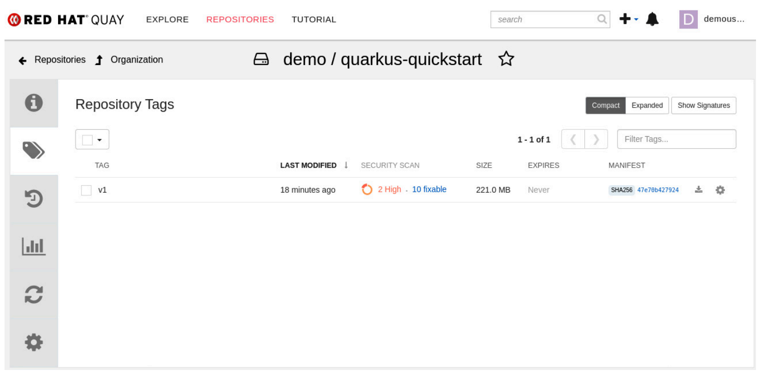
```
deploy
STEP-OC
  service/quarkus-quickstarts created
  deployment.apps/quarkus-quickstarts created
  route.route.openshift.io/quarkus-quickstarts created
```

Figure 14.9 – Build pipeline run successfully

After the pipeline runs successfully, you may want to see what the image looks like on Quay.

Checking the image on Quay

If you are using Quay, at this stage, you should be able to see and inspect the image there:



TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
v1	18 minutes ago	2 High · 10 fixable	221.0 MB	Never	SHA256 47e70b427924

Figure 14.10 – Image on Quay, known vulnerabilities detected automatically

As we can see, Quay detected automatically that this image has some known vulnerabilities. We are going to fix these vulnerabilities further in this chapter, but it is important now that you observe and understand how easy it is to push images and start checking them automatically against known vulnerabilities.

Now that we already have the building pipeline of our application working, let's evolve it to use ArgoCD as the deployment tool, leveraging GitOps practices.

Application deployment using OpenShift Pipelines and GitOps

This time, we are going to use ArgoCD to deploy the application instead of directly running the Kubernetes manifests. The pipeline is basically the same, but now the deploy task will run a YAML file that creates an ArgoCD application and wait until the application becomes healthy.

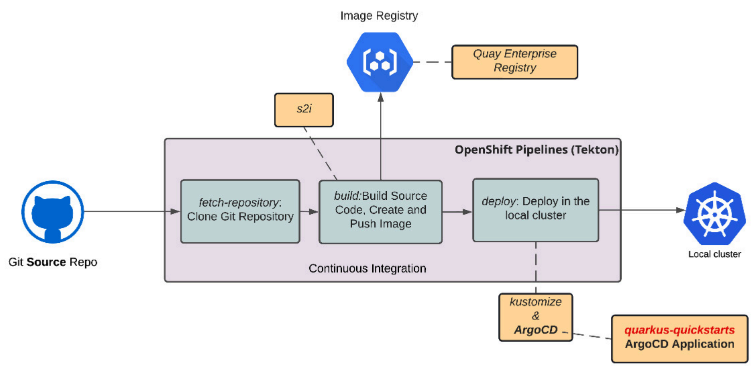


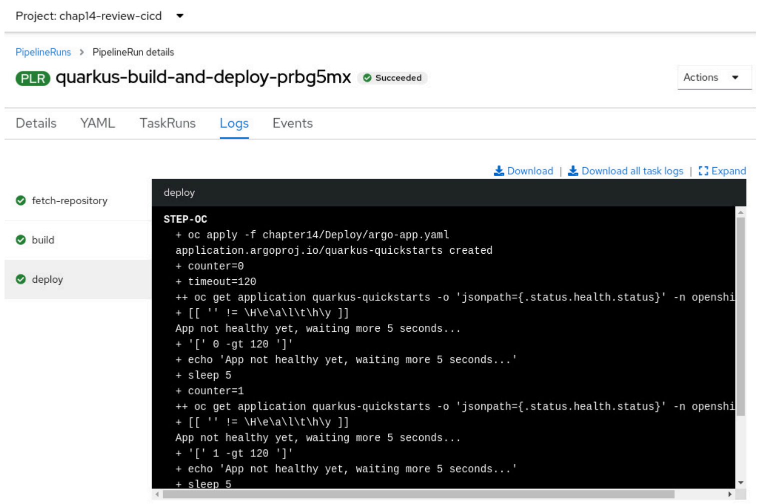
Image Registry
Quay Enterprise Registry
s2i
OpenShift Pipelines (Tekton)
Git Source Repo
fetch-repository: Clone Git Repository
build: Build Source Code, Create and Push Image
deploy: Deploy in the local cluster
Continuous Integration
Local cluster
kustomize & ArgoCD
quarkus-quickstarts ArgoCD Application

Figure 14.11 – Pipeline to build a Java Quarkus application and deploy it using ArgoCD

Run the following command to create and run the pipeline:

```
$ cd OpenShift-Multi-Cluster-Management-Handbook/chapter14/Deploy
$ oc apply -f Rolebindings/ # Permission required for Tekton to create an ArgoCD applicati
$ oc apply -f Pipeline/quarkus-build-and-deploy-pi.yaml
$ oc create -f PipelineRun/quarkus-build-and-deploy-pr.yaml
```

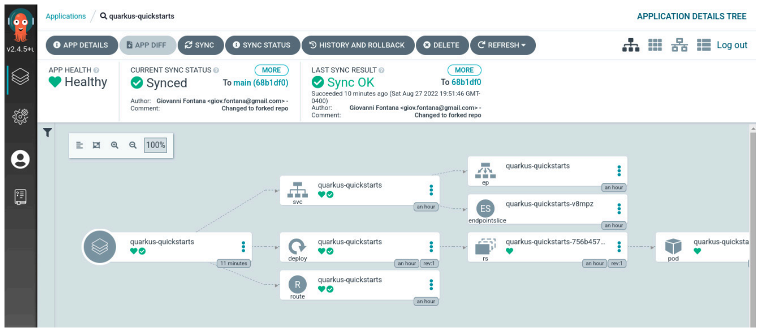
A new **`PipelineRun`** will be created to build the container image and create the ArgoCD application that will deploy the application. You will see the following if everything works well:


Project: chap14-review-cicd
PipelineRuns > PipelineRun details
PLR quarkus-build-and-deploy-prbg5mx
Succeeded
Actions
Details
YAML
TaskRuns
Logs
Events
Download | Download all task logs | Expand
fetch-repository
build
deploy
deploy

```
STEP-OC
  + oc apply -f chapter14/Deploy/argo-app.yaml
  application.argoproj.io/quarkus-quickstarts created
  + counter=0
  + timeout=120
  ++ oc get application quarkus-quickstarts -o 'jsonpath={.status.health.status}' -n openshi
  + [[ '' != \H\e\a\l\t\h\y ]]
  App not healthy yet, waiting more 5 seconds...
  + '[' 0 -gt 120 ']'
  + echo 'App not healthy yet, waiting more 5 seconds...'
  + sleep 5
  + counter=1
  ++ oc get application quarkus-quickstarts -o 'jsonpath={.status.health.status}' -n openshi
  + [[ '' != \H\e\a\l\t\h\y ]]
  App not healthy yet, waiting more 5 seconds...
  + '[' 1 -gt 120 ']'
  + echo 'App not healthy yet, waiting more 5 seconds...'
  + sleep 5
```

Figure 14.12 – Task deployment using ArgoCD

Access the ArgoCD console to check the application deployment from there; you should see something similar to the following screenshot:



APP HEALTH Healthy

CURRENT SYNC STATUS Synced To main (68b1df0)
Author: Giovanni Fontana <giov.fontana@gmail.com> -
Comment: Changed to forked repo

LAST SYNC RESULT Sync OK To 68b1df0
Succeeded 10 minutes ago (Sat Aug 27 2022 19:51:46 GMT-0400)
Author: Giovanni Fontana <giov.fontana@gmail.com> -
Comment: Changed to forked repo

quarkus-quickstarts
quarkus-quickstarts-v8mpz
quarkus-quickstarts-756b457...

Figure 14.13 – Application in ArgoCD

You can find instructions about how to access the ArgoCD console in ***Chapter 10***, *OpenShift GitOps – ArgoCD*. As a reminder, see next the commands to get the ArgoCD URL and admin password:

```
# Get the ArgoCD URL:
$ echo "$(oc get route openshift-gitops-server -n openshift-gitops --template='https://{{.
# Get the Admin password
$ oc extract secret/openshift-gitops-cluster -n openshift-gitops --to=-
```

Now our pipeline already builds the application, pushes it to Quay, and deploys it using ArgoCD. The next step is to bring Advanced Cluster Security to add a security check step in our pipeline. See next how to do it.

Adding security checks in the building and deployment process

This time, we will add a new step to perform a security check in the image that has been built. We are going to use Advanced Cluster Security for that. To successfully use it, you should have Advanced Cluster Security installed and the local cluster configured as a secured cluster. Check ***Chapter 12***, *OpenShift Multi-Cluster Security*, to see how to do it.

See next what our pipeline looks like now:

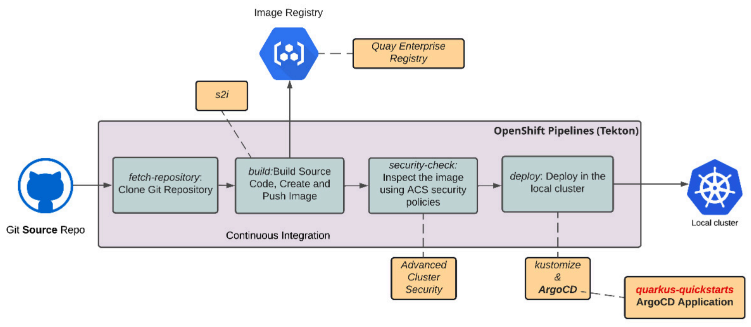


Image Registry
Quay Enterprise Registry
s2i
OpenShift Pipelines (Tekton)
Git Source Repo
fetch-repository: Clone Git Repository
build: Build Source Code, Create and Push Image
security-check: Inspect the image using ACS security policies
deploy: Deploy in the local cluster
Local cluster
Continuous Integration
Advanced Cluster Security
kustomize & ArgoCD
quarkus-quickstarts ArgoCD Application

Figure 14.14 – Pipeline with security checks

Therefore, the following task has been added to the pipeline:

- **`security-check`**: Uses ACS APIs to check the image against existing security policies defined in ACS.

To simulate security issues, we will also use a custom **`s2i-java`** task that uses an old **`ubi-openjdk`** version, which contains many known vulnerabilities. To fix the issues, we will change the build strategy to use a Dockerfile that uses the latest version of the RHEL UBI image and additional security fixes.

Follow the instructions in this section to create and run this pipeline:

1. Before we get into the pipeline, we need to configure the integration between the pipeline and ACS. To do so, access the **Advanced Cluster Security** dashboard and navigate to **Platform Configuration** | **Integrations** | **Authentication Tokens**, and click on **API Token**:

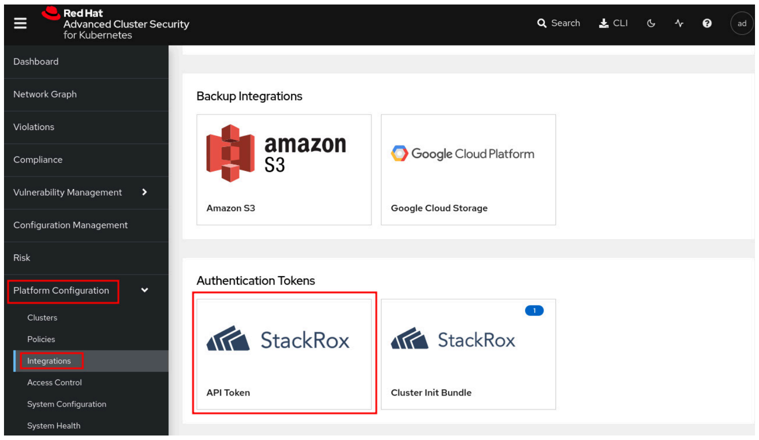
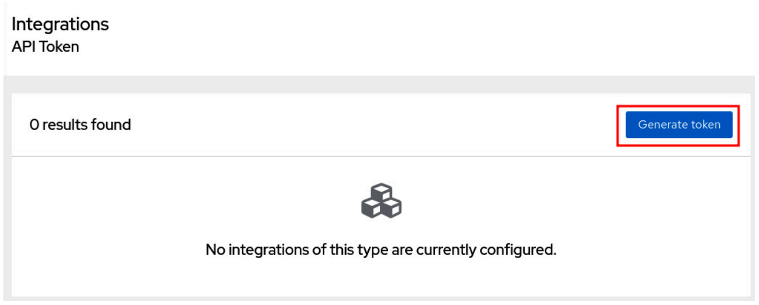


Figure 14.15 – Creating ACS API Token

2. Click on the **Generate token** button:


Integrations
API Token

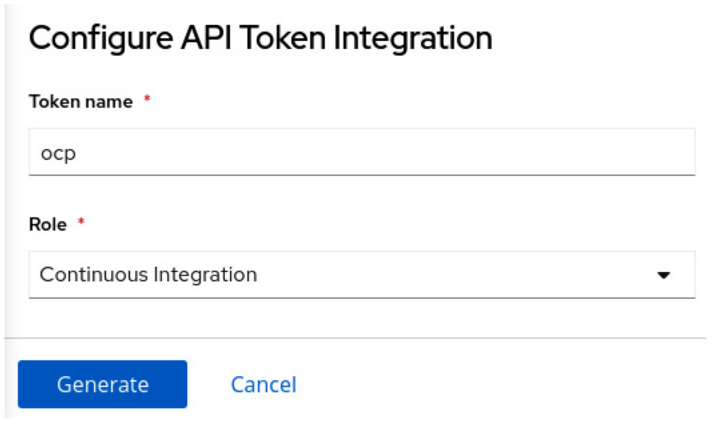
0 results found

Generate token

No integrations of this type are currently configured.

Figure 14.16 – Generate token

3. Fill out a name and select **Continuous Integration** in the **Role** field:


Configure API Token Integration

Token name *

ocp

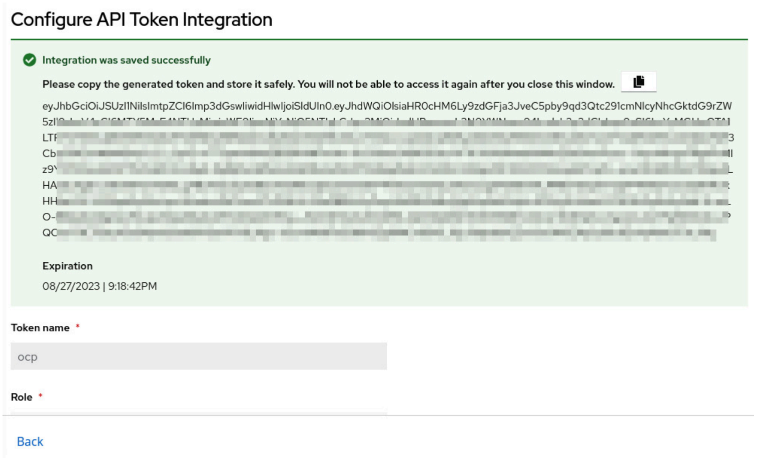
Role *

Continuous Integration

Generate Cancel

Figure 14.17 – Generate token for CI

4. Copy the token that has been generated. We are going to use it in a secret that will be used by the pipeline task to authenticate on ACS APIs:



Configure API Token Integration

Integration was saved successfully

Please copy the generated token and store it safely. You will not be able to access it again after you close this window.

eyJhbGciOiJSUzI1NiIsImtpZCI6Imp3dGswliwidHlwIjoiSldUIn0.eyJhdWQiOlsiaHR0cHM6Ly9zdGFja3JveC5pby9qd3Qtc291cmNlcyNhcGktdG9rZW
5zl[redacted]

Expiration

08/27/2023 | 9:18:42PM

Token name *

ocp

Role *

Figure 14.18 – Copy API Token

5. Now let's create the secret. Run the following command using the token from the previous step and the ACS central endpoint. Do *not* use **`http(s)`** in the **`rox_central_endpoint`** host:

```
oc create secret generic acs-secret --from-literal=rox_api_token='<token from previous
```

6. Now we are all set to create and run our pipeline. Run the following commands:

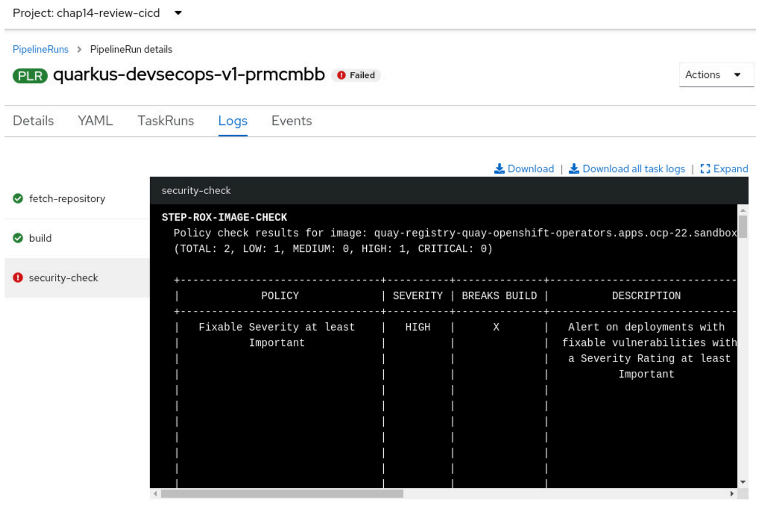
```
$ cd OpenShift-Multi-Cluster-Management-Handbook/chapter14/DevSecOps
```

```
$ oc apply -f Task/ # Create the custom S2I and stackrox-image-check tasks
```

```
$ oc apply -f Pipeline/quarkus-devsecops-v1-pi.yaml
```

```
$ oc create -f PipelineRun/quarkus-devsecops-v1-pr.yaml
```

7. You should see failures in the **`security-check`** task as we are intentionally using an old base image that contains many known vulnerabilities:


Project: chap14-review-cicd
PipelineRuns > PipelineRun details
PLR quarkus-devsecops-v1-prmcmbb Failed
Actions
Details YAML TaskRuns Logs Events
Download | Download all task logs | Expand
fetch-repository
build
security-check

```
security-check
STEP-ROX-IMAGE-CHECK
  Policy check results for image: quay-registry-quay-openshift-operators.apps.ocp-22.sandbox
  (TOTAL: 2, LOW: 1, MEDIUM: 0, HIGH: 1, CRITICAL: 0)

  +-------------------------------+----------+--------------+-----------------------------
  |            POLICY             | SEVERITY | BREAKS BUILD |          DESCRIPTION
  +-------------------------------+----------+--------------+-----------------------------
  |   Fixable Severity at least   |   HIGH   |      X       |  Alert on deployments with
  |          Important            |          |              | fixable vulnerabilities with
  |                               |          |              |  a Severity Rating at least
  |                               |          |              |          Important
  |                               |          |              |
  |                               |          |              |
  |                               |          |              |
  |                               |          |              |
  |                               |          |              |
  |                               |          |              |
```

Figure 14.19 – Security checks failure

Let’s take a look briefly at the errors we have as a result of this task. The policies that have failed are the following:

- **Fixable Severity at least Important**: As expected (remember that we are using now an old base image version), there are several components in the image that have important and critical known CVEs.
- **Red Hat Package Manager in Image**: Our S2I image uses **`ubi-minimal`**, which includes **`microdnf`** as a package manager.

We are going to demonstrate now how to fix these security issues using a Dockerfile that addresses all of them.

Fixing security issues

To fix the issues, we are going to change our pipeline to use a Dockerfile instead of the S2I. To do so we changed the **`build`** task to use the **`buildah`** ClusterTask instead of **`s2i-java`**:

```
    - name: build
      params:
        - name: IMAGE
          value: $(params.IMAGE_NAME)
        - name: CONTEXT
          value: $(params.PATH_CONTEXT)
        - name: DOCKERFILE
          value: src/main/docker/Dockerfile.multistage #[1]
        - name: TLSVERIFY
          value: 'false'
      runAfter:
        - fetch-repository
      taskRef:
        kind: ClusterTask
        name: buildah #[2]
      workspaces:
        - name: source
          workspace: workspace
```

Let’s take a look at what the highlighted numbers mean:

- **[1]**: The path where the Dockerfile with security fixes is located
- **[2]**: **`buildah`** ClusterTasks that will build the application using the given Dockerfile

Let's also take a look at the Dockerfile to understand the security fixes. This file is located at **`quarkus-getting-started/src/main/docker/Dockerfile.multistage`** in our GitHub:

```
## Stage 1: build with maven builder image with native capabilities
FROM quay.io/quarkus/ubi-quarkus-native-image:22.2-java17 AS build
(.. omitted ..)
RUN ./mvnw package -Pnative
FROM registry.access.redhat.com/ubi8/ubi-minimal #[1]
WORKDIR /work/
COPY --from=build /code/target/*-runner /work/application
RUN chmod 775 /work /work/application \
  && chown -R 1001 /work \
  && chmod -R "g+rwX" /work \
  && chown -R 1001:root /work \
  && microdnf update -y \ #[2]
  && rpm -e --nodeps $(rpm -qa '*rpm*' '*dnf*' '*libsolv*' '*hawkey*' 'yum*') #[3]
EXPOSE 8080
USER 1001
CMD ["./application", "-Dquarkus.http.host=0.0.0.0"]
```

Let's take a look at what the highlighted numbers mean:

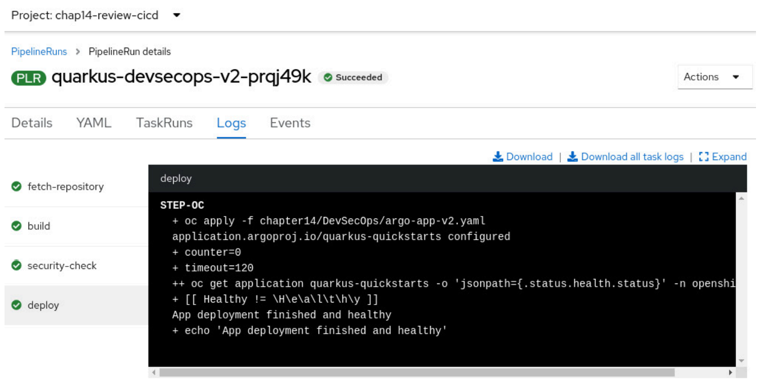
- **[1]**: Use the latest version of **`ubi-minimal`** as the base image.
- **[2]**: Update OS packages to the latest versions.
- **[3]**: Remove the package manager from the image.

The lines highlighted will make sure the most up-to-date components, which contain the most recent security fixes, are in use, and also the package manager is removed from the image before it is packaged.

Now, let's create this new pipeline version and runs it to check whether the security issues have been resolved. To do so, run the following commands:

```
$ oc apply -f Pipeline/quarkus-devsecops-v2-pi.yaml
$ oc create -f PipelineRun/quarkus-devsecops-v2-pr.yaml
```

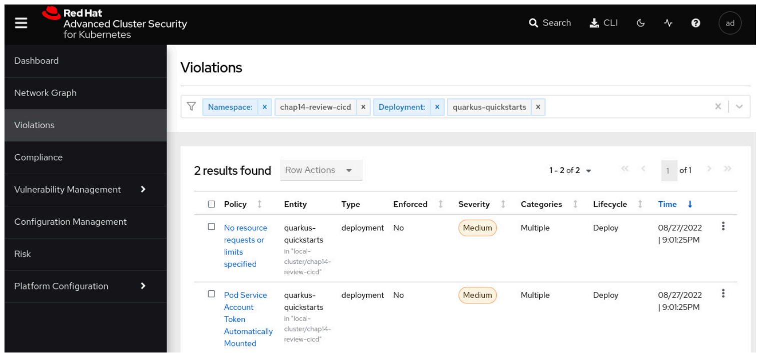
This time, the pipeline should be finished successfully, as there are no security issues detected anymore in our container image:


Project: chap14-review-cicd
PipelineRuns > PipelineRun details
PLR quarkus-devsecops-v2-prqj49k Succeeded
Actions
Details
YAML
TaskRuns
Logs
Events
Download | Download all task logs | Expand
fetch-repository
build
security-check
deploy

```
deploy
STEP-OC
  + oc apply -f chapter14/DevSecOps/argo-app-v2.yaml
  application.argoproj.io/quarkus-quickstarts configured
  + counter=0
  + timeout=120
  ++ oc get application quarkus-quickstarts -o 'jsonpath={.status.health.status}' -n openshi
  + [[ Healthy != \H\e\a\l\t\h\y ]]
  App deployment finished and healthy
  + echo 'App deployment finished and healthy'
```

Figure 14.20 – Security issues fixed

You can optionally check ACS now to investigate whether there are still other violations that may be fixed later. If you want to do so, navigate to the **Violations** feature of ACS and filter by **`Namespace: chap14-review-cicd`** and **`Deployment: quarkus-quickstarts`**. You should still see some minor violations, as follows:

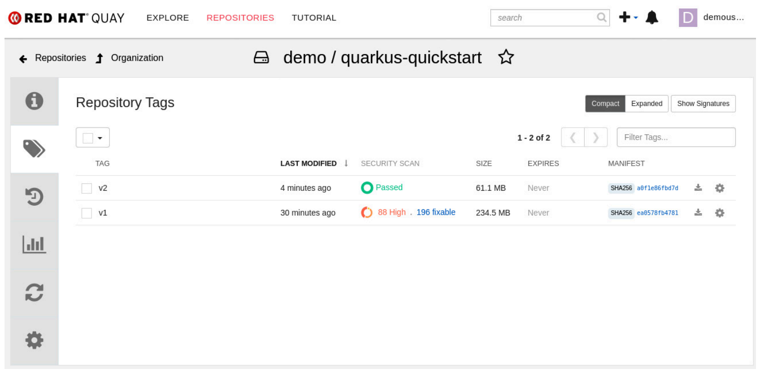


2 results found

Policy	Entity	Type	Enforced	Severity	Categories	Lifecycle	Time
No resource requests or limits specified	quarkus-quickstarts in "local-cluster/chap14-review-cicd"	deployment	No	Medium	Multiple	Deploy	08/27/2022 | 9:01:25PM
Pod Service Account Token Automatically Mounted	quarkus-quickstarts in "local-cluster/chap14-review-cicd"	deployment	No	Medium	Multiple	Deploy	08/27/2022 | 9:01:25PM

Figure 14.21 – ACS Violations

Do you remember that Quay reported some vulnerabilities in our image before? Look at it now to see our new image version:



TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
v2	4 minutes ago	Passed	61.1 MB	Never	SHA256 a0f1e86fbd7d
v1	30 minutes ago	88 High . 196 fixable	234.5 MB	Never	SHA256 ea0578fb4781

Figure 14.22 – Quay security scan

As you can see, the newer image has no security issues detected. In this section, we added a security check in our pipeline and fixed some vulnerabilities detected by this pipeline. In the next section, our pipeline will be able to deploy our application against multiple clusters, using ArgoCD and Advanced Cluster Management.

Provisioning and managing multiple clusters

We haven't touched so far on the hybrid or multi-cluster side of the house. This is what we are going to add now: *deployment into multiple remote clusters*. To do so, we are going to use Advanced Cluster Management to provision new clusters and also help us to deploy the application in them.

Provisioning new clusters

We are going to use AWS to host two new clusters that will be used as remote clusters to exercise our pipeline. For the sake of saving resources, we are going to use single node clusters, so we don't need to get the cost of many servers for this exercise. If you already have clusters available, you can alternatively import the existing clusters, instead of provisioning new ones. You can find, in the *Further reading* section of this chapter, a link that contains instructions about how to import a cluster on Advanced Cluster Management.

To provision a single node cluster using ACM, you need to add the AWS credentials, navigate to the **Credentials** menu, and click on the **Add credential** button:

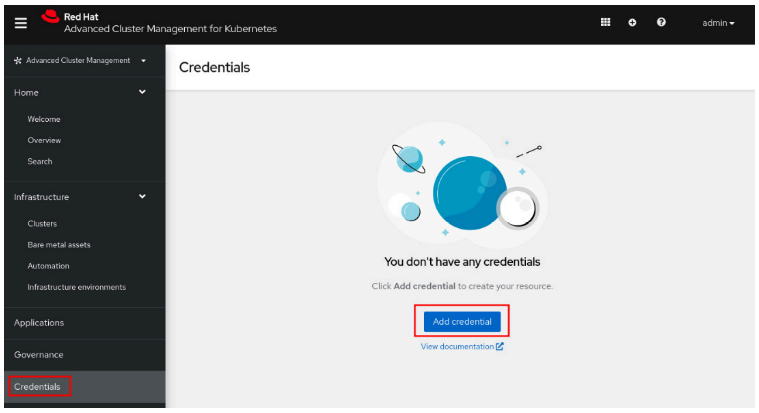
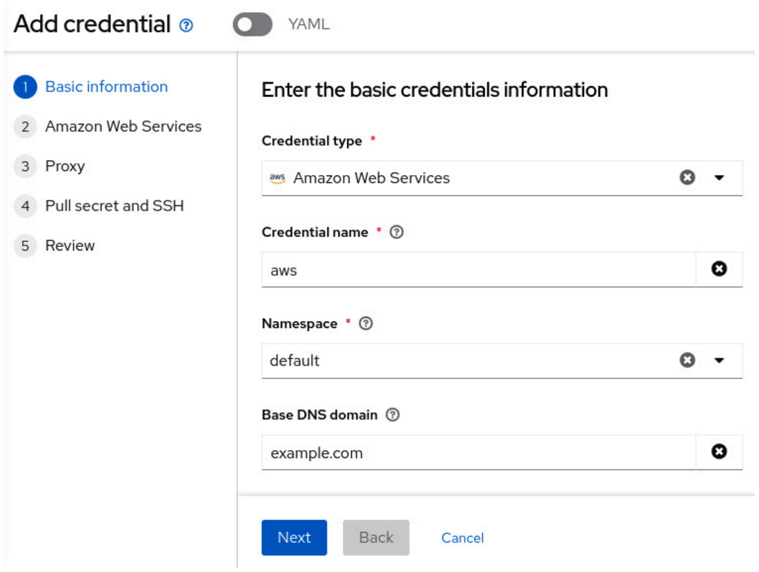

Red Hat
Advanced Cluster Management for Kubernetes
admin
Advanced Cluster Management
Home
Welcome
Overview
Search
Infrastructure
Clusters
Bare metal assets
Automation
Infrastructure environments
Applications
Governance
Credentials
Credentials
You don't have any credentials
Click Add credential to create your resource.
Add credential
View documentation

Figure 14.23 – Adding AWS credentials

Follow the wizard and fill out all required fields. You need to provide your pull secret, which is available at https://console.redhat.com/openshift/downloads:



Enter the basic credentials information

Credential type * Amazon Web Services

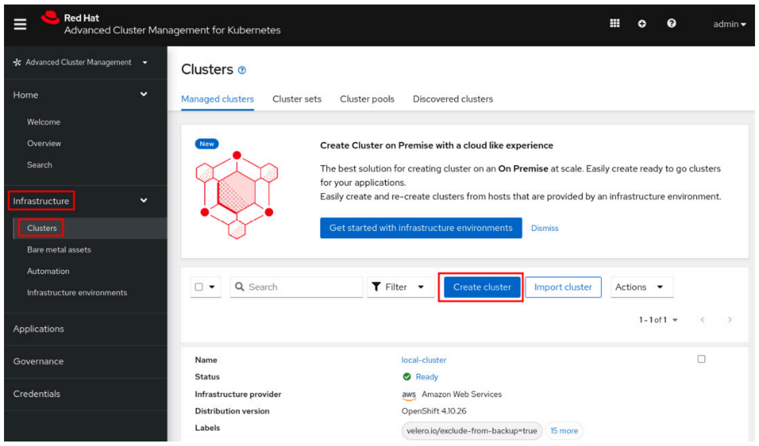
Credential name * aws

Namespace * default

Base DNS domain example.com

Figure 14.24 – Adding a new credential

After you have created the AWS credential, access the **Infrastructure** | **Clusters** feature and click on the **Create cluster** button:



Create Cluster on Premise with a cloud like experience

The best solution for creating cluster on an **On Premise** at scale. Easily create ready to go clusters for your applications.
Easily create and re-create clusters from hosts that are provided by an infrastructure environment.

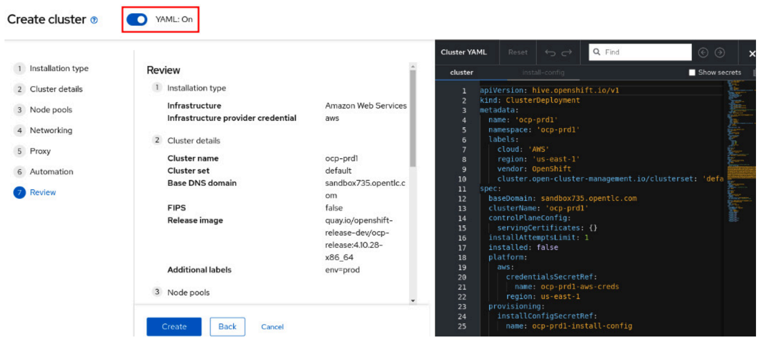
Name	local-cluster
Status	Ready
Infrastructure provider	aws Amazon Web Services
Distribution version	OpenShift 4.10.26
Labels	velero.io/exclude-from-backup=true 15 more

Figure 14.25 – Provisioning a new cluster

Select AWS as the infrastructure provider and fill out the wizard with the following data but do *not* hit the **Create** button in the last step of the wizard:

- **Infrastructure** provider: **Amazon Web Services**
- **Infrastructure provider credential**: **aws** (name of the credential that you created in the previous step)
- **Cluster name**: **`ocp-prd1`**
- **Cluster set**: **default**
- **Base DNS domain**: Your public domain on AWS (for example, example.com)
- **Release image**: Select the most recent
- **Additional labels**: **`env=prod`**
- **Control plane pool**: Instance type: **`m5.2xlarge`**
- **Networking**: Leave as-is
- **Proxy**: Leave unselected
- **Automation**: Leave blank

On the **Review** page, select the **YAML: On** button:



Review

1 Installation type

Infrastructure	Amazon Web Services
Infrastructure provider credential	aws

2 Cluster details

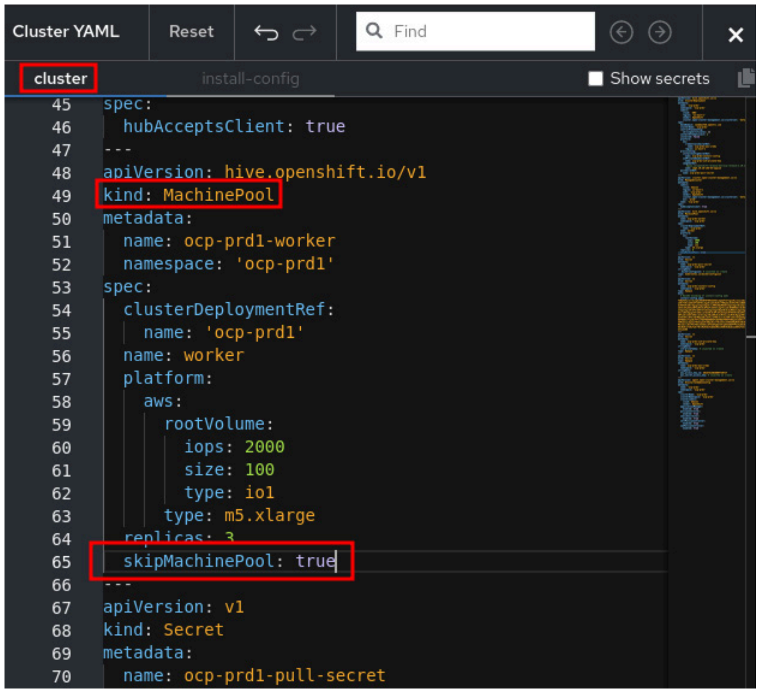
Cluster name	ocp-prd1
Cluster set	default
Base DNS domain	sandbox735.opentlc.com
FIPS	false
Release image	quay.io/openshift-release-dev/ocp-release:4.10.28-x86_64
Additional labels	env=prod

3 Node pools

```yaml
apiVersion: hive.openshift.io/v1
kind: ClusterDeployment
metadata:
  name: 'ocp-prd1'
  namespace: 'ocp-prd1'
  labels:
    cloud: 'AWS'
    region: 'us-east-1'
    vendor: OpenShift
    cluster.open-cluster-management.io/clusterset: 'defa
spec:
  baseDomain: sandbox735.opentlc.com
  clusterName: 'ocp-prd1'
  controlPlaneConfig:
    servingCertificates: {}
  installAttemptsLimit: 1
  installed: false
  platform:
    aws:
      credentialsSecretRef:
        name: ocp-prd1-aws-creds
      region: us-east-1
  provisioning:
    installConfigSecretRef:
      name: ocp-prd1-install-config
```

Figure 14.26 – Edit YAML

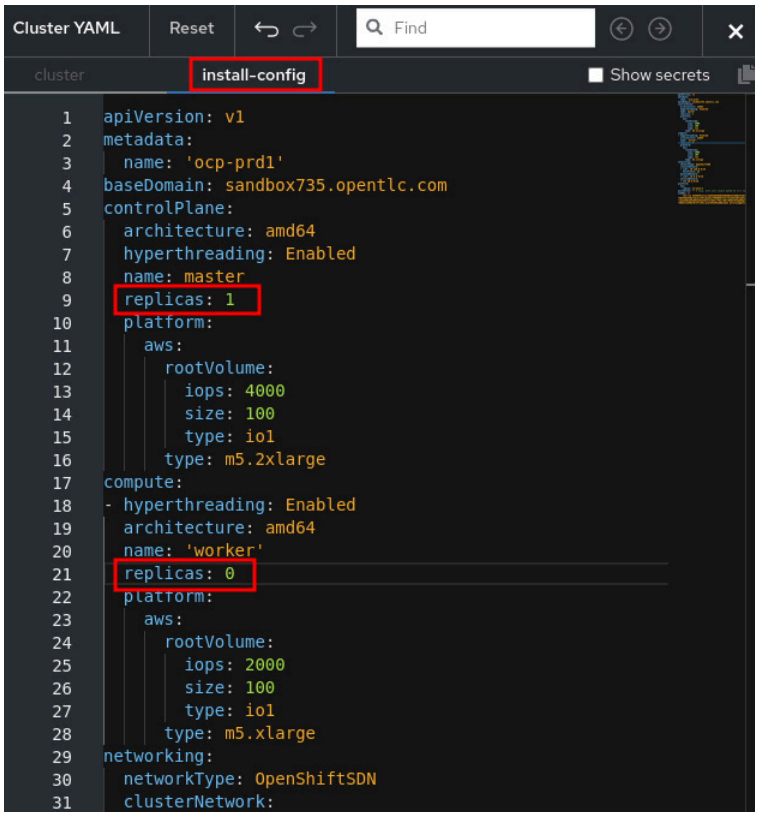
In the YAML file, edit **MachinePool** and add the statement **skipMachinePool: true**, as you can see in the following:



```yaml
spec:
  hubAcceptsClient: true
---
apiVersion: hive.openshift.io/v1
kind: MachinePool
metadata:
  name: ocp-prd1-worker
  namespace: 'ocp-prd1'
spec:
  clusterDeploymentRef:
    name: 'ocp-prd1'
  name: worker
  platform:
    aws:
      rootVolume:
        iops: 2000
        size: 100
        type: io1
      type: m5.xlarge
  replicas: 3
  skipMachinePool: true
---
apiVersion: v1
kind: Secret
metadata:
  name: ocp-prd1-pull-secret
```

Figure 14.27 – Editing MachinePool

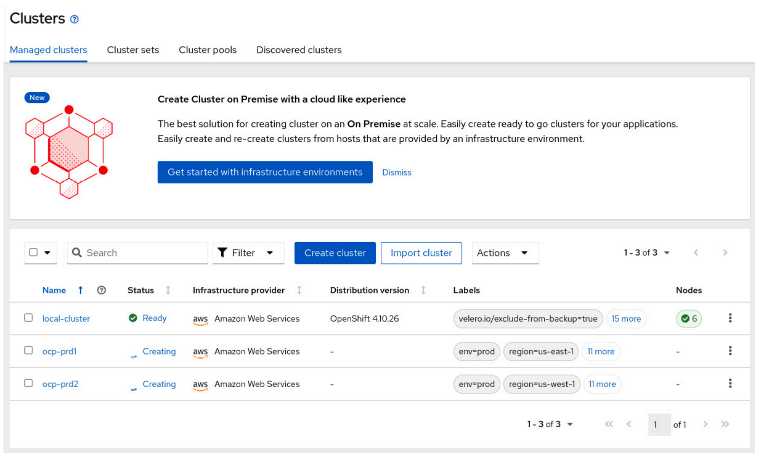
Click in the **install-config** tab and change master replicas to **1** and compute replicas to **0**:



```yaml
apiVersion: v1
metadata:
  name: 'ocp-prd1'
baseDomain: sandbox735.opentlc.com
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  replicas: 1
  platform:
    aws:
      rootVolume:
        iops: 4000
        size: 100
        type: io1
      type: m5.2xlarge
compute:
- hyperthreading: Enabled
  architecture: amd64
  name: 'worker'
  replicas: 0
  platform:
    aws:
      rootVolume:
        iops: 2000
        size: 100
        type: io1
      type: m5.xlarge
networking:
  networkType: OpenShiftSDN
  clusterNetwork:
```

Figure 14.28 – Editing install-config

Now hit the **Create** button. Repeat the steps to create another cluster named **ocp–prd2** with the same parameters used previously. In the end, you should see two clusters being provisioned, as you can see in the following screenshot:



Create Cluster on Premise with a cloud like experience

The best solution for creating cluster on an **On Premise** at scale. Easily create ready to go clusters for your applications. Easily create and re-create clusters from hosts that are provided by an infrastructure environment.

Name	Status	Infrastructure provider	Distribution version	Labels	Nodes
local-cluster	Ready	aws Amazon Web Services	OpenShift 4.10.26	velero.io/exclude-from-backup=true 15 more	6
ocp-prd1	Creating	aws Amazon Web Services	-	env=prod region=us-east-1 11 more	-
ocp-prd2	Creating	aws Amazon Web Services	-	env=prod region=us-west-1 11 more	-

Figure 14.29 – Clusters being created

The provisioning will take about 40 minutes. Continue next when you see both clusters marked as **Ready**.

Cluster governance

One helpful feature that ACM provides is cluster governance using policies. We already covered this feature in ***Chapter 11***, *OpenShift Multi-Cluster GitOps and Management*. If you didn't read it yet, we strongly recommend you check that chapter. We are going to deploy the policy that is in the **`Governance`** folder of our GitHub repository to inform if the etcd keystores of managed clusters are encrypted or not. To do so, run the following command:

```
$ cd OpenShift-Multi-Cluster-Management-Handbook/chapter14/Governance
$ oc apply -k .
```

Wait a few seconds and access the **Governance** feature on ACM to check the compliance of the clusters:

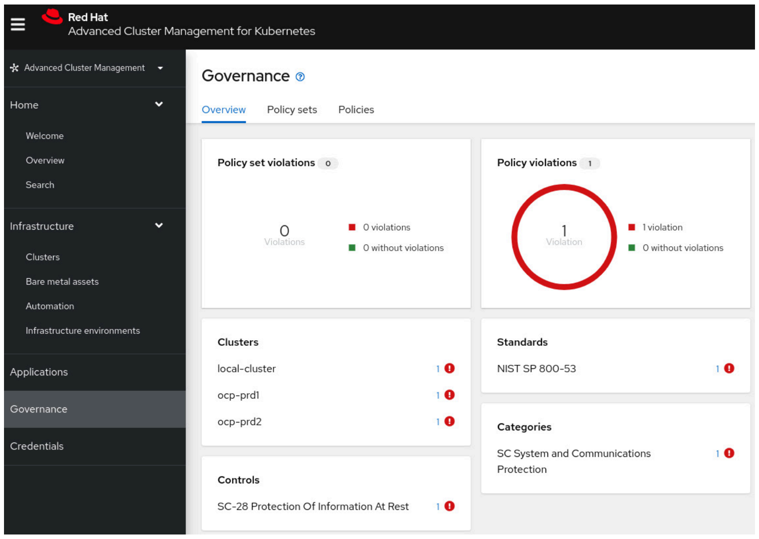


Figure 14.30 – Cluster compliance

Move to the next section to see how to deploy our sample application into multiple remote clusters at once.

Deploying an application into multiple clusters

Now that we already have multiple remote clusters, we can go ahead and use ACM and ArgoCD to make our pipeline able to deploy into all of them at once. We are going to change the deploy task to use an **`ApplicationSet`** object that will be responsible for deploying our application into both OpenShift remote clusters at once.

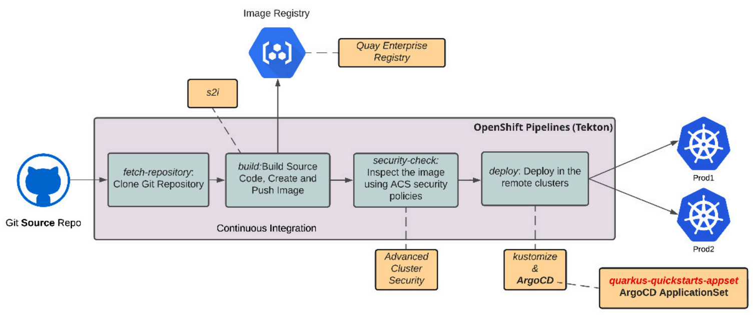


Image Registry
Quay Enterprise Registry
s2i
OpenShift Pipelines (Tekton)
Git Source Repo
fetch-repository: Clone Git Repository
build: Build Source Code, Create and Push Image
security-check: Inspect the image using ACS security policies
deploy: Deploy in the remote clusters
Continuous Integration
Prod1
Prod2
Advanced Cluster Security
kustomize & ArgoCD
quarkus-quickstarts-appset ArgoCD ApplicationSet

Figure 14.31 – Pipeline with deployment into multiple clusters

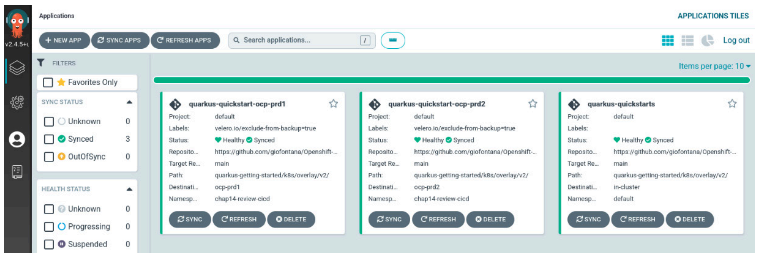
To make ArgoCD aware of the clusters managed by ACM, we first need to create a few objects, such as the **`GitOpsCluster`** Custom Resource. We covered a detailed explanation of these objects in *Chapter 11*, *OpenShift Multi-Cluster GitOps and Management*. Run the following commands to create these objects:

```
$ cd OpenShift-Multi-Cluster-Management-Handbook/chapter14/Multicluster-Deployment
$ oc apply -f GitOpsCluster/
```

Now let's create and run the pipeline, which uses an **`ApplicationSet`** object to deploy the application into the managed clusters that have the **`env=prod`** label. Remember that we used this label in the clusters we provisioned using ACM. If you imported the clusters on ACM, make sure to add the **`env=prod`** label to them:

```
$ oc apply -f Rolebindings/ # Permissions required for pipeline to be able to create an Ap
$ oc apply -f Pipeline/quarkus-multicluster-pi.yaml
$ oc create -f PipelineRun/quarkus-multicluster-pr.yaml
```

When the pipeline finishes, you should now have two new ArgoCD applications automatically created by the **`ApplicationSet`** mechanism:



quarkus-quickstart-ocp-prd1
Project: default
Labels: velero.io/exclude-from-backup=true
Status: Healthy Synced
Reposito... https://github.com/giofontana/Openshift-...
Target Re... main
Path: quarkus-getting-started/k8s/overlay/v2/
Destinati... ocp-prd1
Namesp... chap14-review-cicd

quarkus-quickstart-ocp-prd2
Project: default
Labels: velero.io/exclude-from-backup=true
Status: Healthy Synced
Reposito... https://github.com/giofontana/Openshift-...
Target Re... main
Path: quarkus-getting-started/k8s/overlay/v2/
Destinati... ocp-prd2
Namesp... chap14-review-cicd

quarkus-quickstarts
Project: default
Labels:
Status: Healthy Synced
Reposito... https://github.com/giofontana/Openshift-...
Target Re... main
Path: quarkus-getting-started/k8s/overlay/v2/
Destinati... in-cluster
Namesp... default

Figure 14.32 – ArgoCD and ApplicationSet

That's it, we did it! We started with a simple build pipeline that now performs security checks and deploys into multiple remote clusters at once!

Summary

Thank you for being our partner in this journey! We hope the content of this book was helpful for you and now you have a good understanding of the topics covered in this book. We went through architecture, people, deployment, troubleshooting, multi-cluster administration, usage, and security. So much content that we thought we wouldn't have the ability to write it! And if you are still here, we feel that my mission with this book is accomplished!

There is a quote from *Johann Wolfgang von Goethe* that says the following: "*Knowing is not enough; we must apply. Willing is not enough; we must do*." After reading this book, we hope you not only learned new things but were also able to put them into practice. Following this hybrid cloud journey, you have the opportunity to leap in knowledge with didactic examples and content made with great dedication from us.

We hope that this book becomes one of your handbooks and will be useful to you for planning and executing models suitable for the enterprise, bringing multiple options to use, implementations, and good insights to leverage your knowledge and your career.

To wrap up the content of this chapter, we designed the following diagram to serve as a shortcut to the central themes of each chapter and see the entire journey we have gone through together:

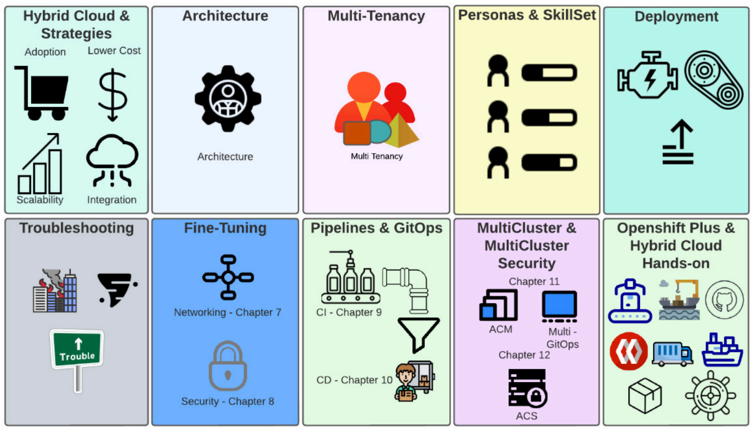


Figure 14.33 – The book journey

We have almost reached the end of this book, but we are not completely done yet. We have prepared for you the last chapter with some suggestions as to where you can go next after this book, to keep learning and growing your OpenShift and Kubernetes skills.

We encourage you to move to the next chapter and look at the training and other content we suggest there.

Further reading

Looking for more information? Check the following references to get more information:

- *Quarkus*:
  - *Main page:* https://quarkus.io/
  - *Quarkus getting started sample*: https://github.com/quarkusio/quarkus-quickstarts/tree/main/getting-started
- *S2i:*
  - *GitHub* page: https://github.com/openshift/source-to-image
  - *How to Create an S2I Builder Image* (blog article): https://cloud.redhat.com/blog/create-s2i-builder-image
  - *Using Source 2 Image build in Tekton* (blog article): https://cloud.redhat.com/blog/guide-to-openshift-pipelines-part-2-using-source-2-image-build-in-tekton
  - *Tekton Hub - S2I*: https://hub.tekton.dev/tekton/task/s2i
- *Advanced Cluster Management – Importing clusters*: https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.5/html/clusters/managing-your-clusters#importing-a-target-managed-cluster-to-the-hub-cluster
- *Advanced Cluster Security – Image check from Tekton Hub*: https://hub.tekton.dev/tekton/task/stackrox-image-check

Previous chapter

< Chapter 13: OpenShift Plus – a Multi-Cluster Enterprise Ready Solution

Next chapter

Part 5 – Continuous Learning >