Red Hat OpenShift Administration II: High Availability (DO 380)

# COURSE OVERVIEW

Plan, implement, and manage OpenShift clusters at scale

Red Hat OpenShift Administration III: Scaling Kubernetes Deployments in the Enterprise (DO380) expands upon the skills required to plan, implement, and manage OpenShift® clusters in the enterprise. You will learn how to support a growing number of stakeholders, applications, and users to achieve large-scale deployments.

# Who will benefit from this course?

* Cluster engineers focused on planning, designing, and implementing production-grade OpenShift clusters. They require automation skills to scale their manpower to provision and manage an increasing population of clusters, applications, and users, while ensuring compliance with corporate standards.
* Site reliability engineers focused on keeping OpenShift clusters and applications running without disruption. They are interested in troubleshooting infrastructure and application issues with OpenShift clusters and require automation skills to reduce the time to identify, diagnose, and remediate issues.

# Prerequisites

* Complete Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster (DO280) and become a Red Hat Certified Specialist in OpenShift Administration.
* Complete Red Hat System Administration II (RH134) and become a Red Hat Certified System Administrator.
* Recommended, but not required: Become a Red Hat Certified Systems Engineer or a Red Hat Certified Specialist in Ansible Automation. Basic knowledge about writing and running Ansible playbooks is required.

# Course Objectives

* Build upon the essential skills required to configure and manage an OpenShift 4.x cluster.
* Teach enhanced skills needed to operate production environments at scale, including:
* Automating Day 2 tasks to establish production clusters with higher performance and availability.
* Integrating OpenShift with enterprise authentication, storage, CI/CD, and GitOps systems to improve productivity of IT operations and compliance with organization’s standards.
* Troubleshooting techniques to identify issues with cluster operators and compute capacity.

Prepare your Instance to access OC

To install the OpenShift CLI (oc) on an Ubuntu EC2 instance, follow these steps:

**1. Update the System**

First, ensure your system is up-to-date:

sudo apt update

sudo apt upgrade -y

**2. Install Dependencies**

Install necessary dependencies:

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

**3. Install Docker**

Docker is required for managing containerized applications:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt update

sudo apt install -y docker-ce

Verify Docker installation:

docker --version

**4. Download and Install OpenShift CLI (oc)**

Download the latest version of the OpenShift CLI:

wget https://mirror.openshift.com/pub/openshift-v4/clients/oc/latest/linux/oc.tar.gz

Extract the downloaded archive:

tar -xvf oc.tar.gz

Move the oc binary to /usr/local/bin:

sudo mv oc /usr/local/bin/

Verify the installation:

oc version

**5. Install kubectl (Optional)**

If you need kubectl for managing Kubernetes resources:

sudo snap install kubectl --classic

Verify kubectl installation:

kubectl version --client

**6. Configure OpenShift CLI**

Log in to your OpenShift cluster:

oc login <your-cluster-url>

Replace <your-cluster-url> with the URL of your OpenShift cluster. You will be prompted to enter your credentials.

Demonstrate that OpenShift is Kubernetes by deploying Kubernetes-native applications on OpenShift.

**Move from Kubernetes to OpenShift**

**Prerequisites**

* Access to OpenShift Console and CLI (oc)
* Basic knowledge of Kubernetes
* An example Kubernetes-native app (YAML files or container image)
* OpenShift project/namespace created for deployment

**Lab Guide: Deploying the nginx-ex App on OpenShift**

**Prerequisites**

* Access to OpenShift Console and CLI (oc)
* Basic knowledge of Kubernetes
* OpenShift project/namespace created for deployment

**Step 1: Log in to OpenShift**

Using CLI:

oc login <OpenShift\_API\_URL> --token=<your\_token>

You can get the API URL and token from the OpenShift Web Console (top-right user menu > Copy Login Command).

**Step 2: Create a New Project**

oc new-project my-nginx-ex-app

**Step 3: Deploy the nginx-ex Application**

Use the Source-to-Image (S2I) method to build and deploy the application:

oc new-app centos/nginx-112-centos7~https://github.com/sclorg/nginx-ex

Alternatively, you can deploy using the sample template:

oc new-app -f https://raw.githubusercontent.com/sclorg/nginx-ex/master/openshift/templates/nginx.json

**Step 4: Expose the Application**

OpenShift requires a route for external access:

oc expose svc/nginx-ex

Check the route:

oc get route

You’ll get a public URL to access the application.

**Step 5: Verify the Deployment**

oc get all

Check pods, deployment, service, and route.

To view logs:

oc logs deploy/nginx-ex

To open the app:

oc get route

Use the hostname in a browser.

**Step 6: Cleanup**

oc delete project my-nginx-ex-app

Lab Guide: Automate OpenShift Administration Tasks Using Bash Scripts

# Automating OpenShift Administration with Bash Scripts

This lab guide will walk you through creating and executing Bash scripts to automate common OpenShift administration tasks. It assumes you have:

* Access to an OpenShift cluster with oc CLI tool configured and logged in.
* Basic understanding of Linux command-line interface.
* A text editor (like nano, vim, gedit).

# Lab Objectives:

* Learn to interact with the OpenShift cluster using the oc CLI within Bash scripts.
* Understand how to use variables, loops, and conditional statements in Bash scripts for OpenShift automation.
* Practice automating common OpenShift tasks like checking pod status, scaling deployments, and managing projects.
* Learn basic error handling in Bash scripts for OpenShift operations.

# Prerequisites

* Access to an OpenShift cluster (4.x or newer).
* oc CLI installed and configured (oc login completed).
* Basic knowledge of Bash scripting and OpenShift concepts (Projects, Pods, Deployments, etc.).

# Lab 1: Setup and Basics

## Objective: Validate access and get cluster information using Bash scripts.

Use any of your favorite editor like nano create Bash scripts like nano logindetails.sh and run Bash scripts with command

bash logindetails.sh

## Script 1: Verify Login and Get Cluster Info

#!/bin/bash

# Check if logged in

if ! oc whoami &> /dev/null; then

    echo "You are not logged in. Please run 'oc login' first."

    exit 1

fi

echo "User: $(oc whoami)"

echo "Server: $(oc whoami --show-server)"

echo "Current Project: $(oc project -q)"

# Lab 2: Automating Project Management

## Objective: Create and delete OpenShift projects via script.

## Script 2: Create a New Project

Save this as create\_project.sh:

* Create bash file create\_project.sh
* To create a project: bash create\_project.sh my-test-pro

#!/bin/bash

PROJECT\_NAME=$1

if [ -z "$PROJECT\_NAME" ]; then

  echo "Usage: $0 "

  exit 1

fi

oc new-project "$PROJECT\_NAME"

**Delete a Project**

Save this as delete\_project.sh:

* Create bash file delete\_project.sh
* To delete a project: bash delete\_project.sh my-test-pro

#!/bin/bash

PROJECT\_NAME=$1

if [ -z "$PROJECT\_NAME" ]; then

  echo "Usage: $0 "

  exit 1

fi

oc delete project "$PROJECT\_NAME"

# Lab 3: Automate Resource Deployment

## Objective: Create a Deployment and Service via a script.

## Script 4: Deploy Nginx Pod

#!/bin/bash

# Step 1: Log in to OpenShift

OPENSHIFT\_API\_URL=""

TOKEN=""

oc login $OPENSHIFT\_API\_URL --token=$TOKEN

# Step 2: Create a New Project

PROJECT\_NAME="my-nginx-ex-app"

oc new-project $PROJECT\_NAME

# Step 3: Deploy the nginx-ex Application

# Using Source-to-Image (S2I) method

oc new-app centos/nginx-112-centos7~https://github.com/sclorg/nginx-ex

# Alternatively, using the sample template

# oc new-app -f https://raw.githubusercontent.com/sclorg/nginx-ex/master/openshift/templates/nginx.json

# Step 4: Expose the Application

oc expose svc/nginx-ex

# Step 5: Verify the Deployment

oc get all

# Check logs

oc logs deploy/nginx-ex

# Get the route to open the app

ROUTE=$(oc get route nginx-ex -o jsonpath='{.spec.host}')

echo "Access your application at: http://$ROUTE"

# Step 6: Cleanup

# Uncomment the following line to delete **the project after verification**

**# oc delete project $PROJECT\_NAME**

## Script 5: Expose the Deployment as a Service

#!/bin/bash

DEPLOYMENT\_NAME=nginx-deployment

SERVICE\_NAME=nginx-service

oc expose deployment "$DEPLOYMENT\_NAME" --port=80 --name="$SERVICE\_NAME"

oc expose service "$SERVICE\_NAME"

**Lab 4: Monitor and Manage Pods**

## Objective: Get Pod status and restart pods automatically.

## Script 6: List All Pods and Their Status

#!/bin/bash

echo "Listing all pods in current project:"

oc get pods -o wide

Script 7: Restart All Pods Matching Label

#!/bin/bash

LABEL=$1

if [ -z "$LABEL" ]; then

  echo "Usage: $0 <label-selector>"

  exit 1

fi

PODS=$(oc get pods -l "$LABEL" -o jsonpath='{.items[\*].metadata.name}')

for pod in $PODS; do

  echo "Deleting pod $pod..."

  oc delete pod "$pod"

done

**Lab 5: Automate Backup of Resource Definitions**

## Objective: Backup all YAML manifests for a project.

## Script 8: Backup Resources

#!/bin/bash

PROJECT=$1

BACKUP\_DIR="./backup\_$PROJECT"

if [ -z "$PROJECT" ]; then

echo "Usage: $0 <project-name>"

exit 1

fi

mkdir -p "$BACKUP\_DIR"

RESOURCES=("deployments" "services" "routes" "pods" "configmaps" "secrets")

for res in "${RESOURCES[@]}"; do

echo "Backing up $res..."

oc get "$res" -n "$PROJECT" -o yaml > "$BACKUP\_DIR/${res}.yaml"

done

echo "Backup complete in $BACKUP\_DIR"

**Lab 6: Scheduled CronJob for Cleanup**

## Objective: Create a CronJob to clean up completed pods.

## Script 9: Create Cleanup CronJob YAML

This script is designed to create a **CronJob** in Kubernetes that automatically deletes completed pods every hour

#!/bin/bash

cat <<EOF | oc apply -f -

apiVersion: batch/v1

kind: CronJob

metadata:

  name: cleanup-completed-pods

spec:

  schedule: "0 \* \* \* \*"  # Every hour

  jobTemplate:

    spec:

      template:

        spec:

          containers:

          - name: cleanup

            image: bitnami/kubectl:latest

            command:

            - /bin/sh

            - -c

            - kubectl delete pod --field-selector=status.phase==Succeeded

          restartPolicy: OnFailure

EOF

**Lab 7: Role-Based Access Automation**

## Objective: Assign roles using Bash.

**Script 10: Assign View Role to a User**

#!/bin/bash

USER=$1

PROJECT=$2

if [ -z "$USER" ] || [ -z "$PROJECT" ]; then

  echo "Usage: $0 <user> <project>"

  exit 1

fi

oc policy add-role-to-user view "$USER" -n "$PROJECT"

Tips for Scripting OpenShift

* Use set -euo pipefail at the start of your scripts for safer execution.
* Use oc get <resource> -o jsonpath=... to extract specific data.

OpenShift Operators

OpenShift Operators are powerful tools that extend Kubernetes functionality to manage complex applications more efficiently. This blog dives deep into understanding Operators, their benefits, and practical implementations, including a hands-on exercise on deploying a database using an Operator.

**Understanding OpenShift Operators and Their Benefits**

Operators in OpenShift (and Kubernetes in general) are specialized applications designed to automate the operational tasks of complex, stateful applications. They represent the next evolutionary step in managing cloud-native workloads by encapsulating operational knowledge into Kubernetes-native constructs.

**What Are OpenShift Operators?**

An OpenShift Operator is essentially a method of packaging, deploying, and managing a Kubernetes application. It uses **Custom Resource Definitions (CRDs)** to extend Kubernetes capabilities, enabling administrators and developers to manage applications and infrastructure declaratively. Operators follow the **Controller pattern**, continuously observing the system state and making adjustments to ensure the desired state is maintained.

**Key Concepts:  
1. Custom Resources (CRs):**

* New API objects introduced by the Operator to represent application-specific configurations.
* For example, a PostgreSQL Operator might define a PostgreSQL custom resource that specifies database configurations.

**2. Custom Resource Definitions (CRDs):**

* Define the schema for custom resources.
* CRDs are registered with the Kubernetes API, enabling the cluster to recognize and manage them.

**3. Operator Lifecycle Manager (OLM):**

* A tool for managing the installation and lifecycle of Operators on OpenShift clusters.
* Simplifies updates, dependencies, and permissions for Operators.

**How Do Operators Work?**

Operators monitor custom resources, compare the actual state of the system to the desired state defined in these resources, and take automated actions to reconcile any differences. They encapsulate the logic for managing an application, from installation to upgrades, backup, and recovery.

For example:

* A PostgreSQL Operator can handle database provisioning, scaling, upgrades, and backups without manual intervention.

**Types of Operators**

**1. Basic Operators:**

* Automate simple tasks like creating deployments or setting configurations.

**2. Advanced Operators:**

* Manage stateful applications and handle complex tasks such as data backups, rolling updates, and disaster recovery.

**3. Cluster Operators:**

* Manage infrastructure-level components such as networking, monitoring, or storage.

**Benefits of OpenShift Operators**

Operators bring several advantages to OpenShift environments, making them essential for managing modern applications:

**1. Automation of Operational Tasks**

Operators significantly reduce manual intervention by automating critical operational tasks, such as:

* Application installation and configuration.
* Monitoring, logging, and alerting.
* Scaling applications up or down based on load.
* Performing application upgrades with minimal downtime.

**2. Enhanced Consistency and Reliability**

Operators ensure that applications are deployed and maintained in a consistent state across different environments, reducing human errors and improving system reliability.

**3. Simplified Application Management**

By encapsulating domain-specific operational knowledge, Operators make it easier for developers and administrators to manage complex applications without deep domain expertise.

**4. Scalable Management**

Operators are particularly useful in managing applications across large-scale, distributed environments. They ensure that:

* Resources are provisioned dynamically as needed.
* Applications scale elastically to handle load fluctuations.

**5. Seamless Upgrades**

Operators automate the process of upgrading applications and their dependencies, ensuring compatibility and minimal disruption to services.

**6. Improved DevOps Workflows**

Operators bridge the gap between development and operations by providing tools that automate the entire application lifecycle. This improves collaboration and accelerates the delivery pipeline.

**Real-World Examples of Operator Benefits**

**1. Database Management:**

* Operators like the PostgreSQL Operator can handle provisioning, scaling, and backup of databases without manual intervention, ensuring high availability.

**2. Monitoring and Logging:**

* Prometheus Operator automates the setup and management of Prometheus instances for monitoring Kubernetes clusters.

**3. CI/CD Automation:**

* Jenkins Operator simplifies the setup and management of Jenkins pipelines, streamlining the CI/CD process.

**4. Storage Provisioning:**

* Operators like the Rook Operator manage storage resources dynamically, enabling seamless scaling and recovery.

**Why Choose OpenShift Operators?**

OpenShift adds robust support for Operators, integrating them deeply into the platform through the **OperatorHub** and the **Operator Lifecycle Manager (OLM)**. This makes it easy to discover, install, and manage Operators while ensuring that they meet OpenShift’s high standards of reliability and security.

**Key OpenShift Features for Operators:**

* **OperatorHub:** A marketplace of certified Operators.
* **OLM:** Streamlines Operator deployment, upgrades, and configuration.
* **Monitoring and Alerts:** Operators can integrate with OpenShift monitoring tools to provide actionable insights.

**Installing and Managing Operators**

Operators in OpenShift simplify managing complex applications and infrastructure by automating lifecycle tasks. OpenShift provides a dedicated toolset to easily install, configure, and manage Operators, primarily through the **OperatorHub** and **Operator Lifecycle Manager (OLM)**.

**Installing Operators**

The installation process in OpenShift is user-friendly and primarily handled via the web console or CLI. Operators can be installed cluster-wide or restricted to specific namespaces based on the scope of their functionality.

**Step-by-Step Guide: Installing Operators Using the OpenShift Web Console**

**1. Access the OperatorHub:**

* Log in to the OpenShift Web Console.
* Navigate to **Operators > OperatorHub**.

**2. Search for the Desired Operator:**

* Browse through available Operators or use the search bar to find the specific Operator you need (e.g., PostgreSQL Operator).

**3. Review the Operator Details:**

* Click on the Operator to view its details, including its features, supported versions, and prerequisites.

**4. Install the Operator:**

* Click the **Install** button.
* Choose the installation mode:  
  - **All Namespaces (Cluster-wide):** The Operator is available across the entire cluster.  
  - **Single Namespace:** The Operator is restricted to a specific namespace.
* Select an update strategy:  
  - **Automatic Updates:** Automatically installs updates when available.  
  - **Manual Updates:** Updates must be applied manually.

**5. Monitor Installation:**

* Navigate to **Operators > Installed Operators** to monitor the installation progress.
* Once installed, the Operator will appear in the selected namespace.

**Installing Operators Using the CLI**

1. Search for available Operators:

* oc get packagemanifests -n openshift-marketplace

2. Install the Operator (e.g., PostgreSQL Operator):

* oc apply -f <operator-subscription-file>.yaml

The YAML file should include the subscription details for the Operator.

**Managing Operators**

Once installed, Operators need to be monitored and maintained to ensure they function as expected. This includes tasks like verifying the Operator’s status, managing updates, and troubleshooting issues.

**1. Monitoring Installed Operators**

* Navigate to **Operators > Installed Operators** in the OpenShift Web Console.
* Check the status of the Operator to ensure it is running correctly.
* Review the logs and events associated with the Operator for troubleshooting.

**2. Updating Operators**

Operators can be updated to newer versions using the **Operator Lifecycle Manager (OLM)**.

* **Automatic Updates:** If enabled during installation, the Operator is automatically updated to the latest version.
* **Manual Updates:**
* Go to **Operators > Installed Operators**.
* Select the Operator and click **Upgrade**.
* Choose the desired version and confirm the upgrade.