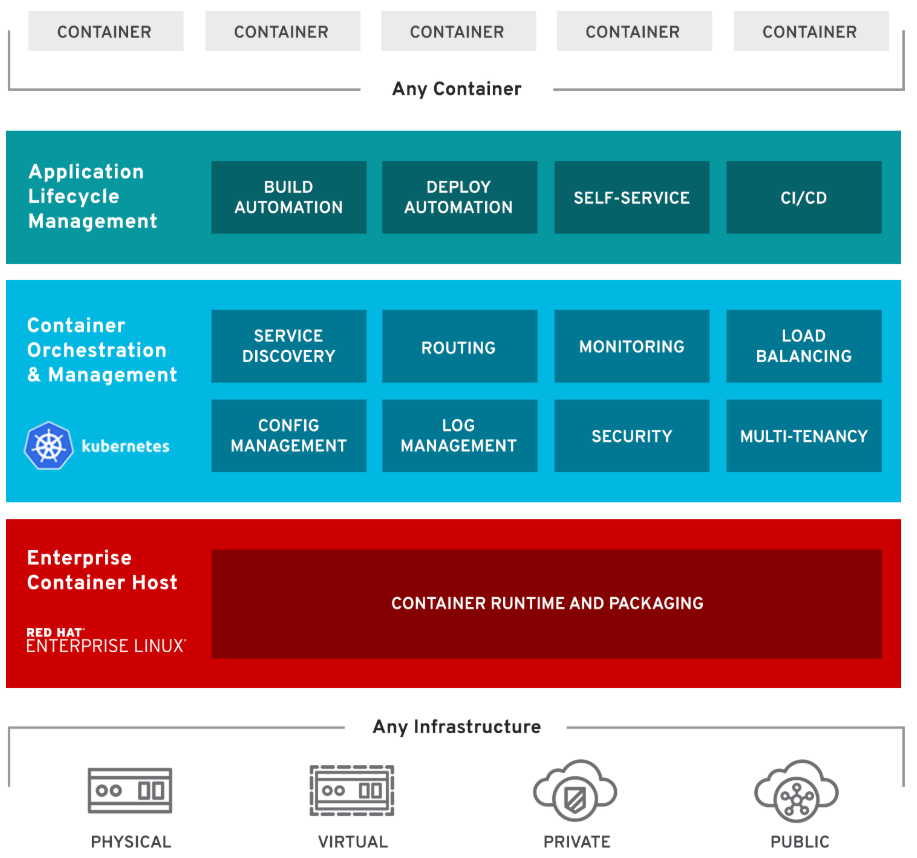
# Deploying OpenShift OKD Cluster setup

# OpenShift Layered Structure­



OpenShift Features

1. **Traditional and Cloud-Native**: Handles cloud-native and traditional applications on a single platform
2. **Multi-tenancy**: Gives your teams self-service to consistent environments across your organization, from development through production.
3. **Hybrid Infra Support**: Offers trusted, proven Kubernetes on any infrastructure.
4. **Security**: Integrated throughout OpenShift, from infrastructure to services, and throughout the operations and application lifecycle
5. **Service on demand**: Seamlessly configure, deploy and consume private and public cloud services with OpenShift Service Catalog.
6. **Built-in Automation**: Remove routine tasks from application teams to focus on business value.
7. **Pluggable Architecture**: Your choice of supported, container runtimes, networking, storage, CI/CD solutions.

# OpenShift Components:

Server roles

There are two types of roles a server can play in OpenShift:

* **Master**: Holding the key services, managing nodes, load distribution, management of OpenShift platform, orchestration. Typically, a master server has the API server, etcd, controller manager, and HAProxy (load balance)
* **Node**: Handles the actual load, act as compute provider. Provides runtime environment for containers. Typically, a node has docker service, kubelet, network proxy services.
* **ETCD**: The **openshift** Master, **openshift** provides a REST endpoint for interacting with the system. An **etcd** server, **Openshift** uses **etcd** to store system configuration and state. Controllers: Controllers are the components that run with the masters that makes sure the running system matches the desired state as stored in **etcd**.

## Services

**API Service**:

* Validates and configures the data for pods, services, and replication controllers.
* Synchronizes pod information with service configuration

**Etcd service**: Stores the persistent master state

**Controller Manager service**: Watches etcd for changes to replication controller objects and then uses the API to enforce the desired state.

**HAProxy service**: Provides load balancing for master servers.

**Kubelet**: Updates the node as specified by a container manifest

**Network proxy**: Reflects the services defined in the API on that node.

## Important Components

**OCR**: Provides users with a built-in location for their application builds to push the resulting images

**Web console**: A user interface accessible from a web browser

**OC CLI**: A CLI-based interface to manage OpenShift and use services.

**Pod**:

* Pods are the rough equivalent of a machine instance (physical or virtual) to a container.
* Each pod is allocated its own internal IP address, therefore owning its entire port space.
* Containers within pods can share their local storage and networking.
* Pod is a smallest compute unit that can be defined, deployed, and managed.

## Users

Users are those who usage and consumed the services.

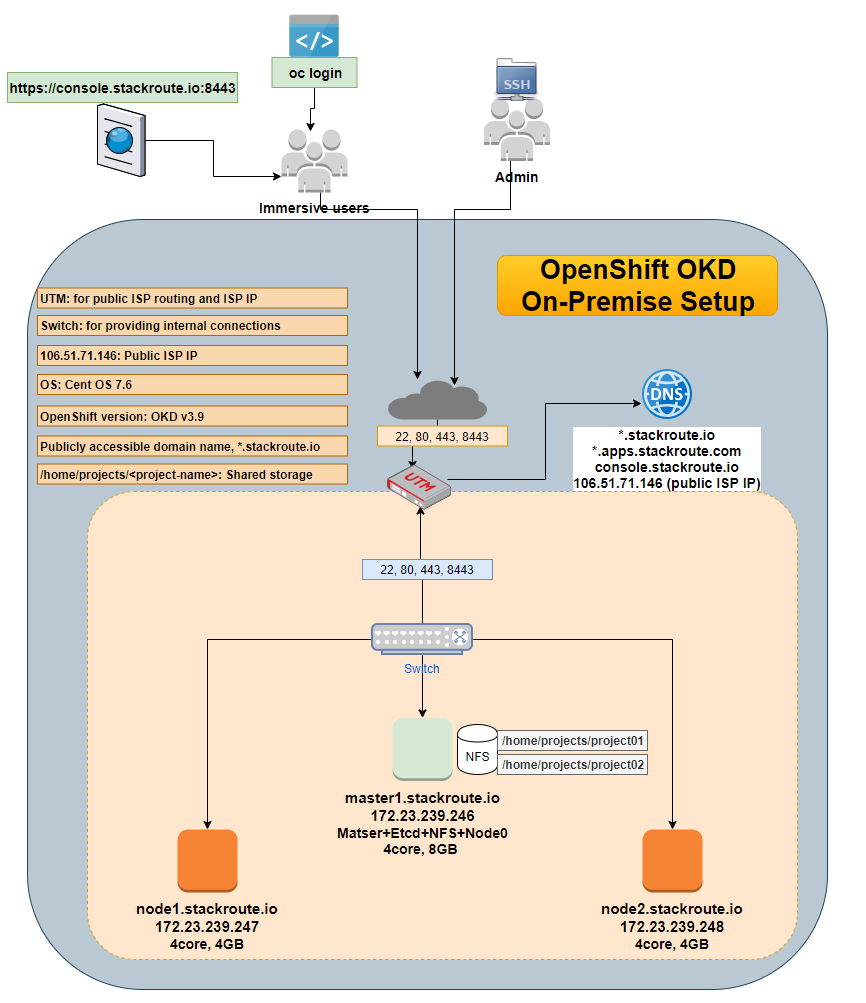
1. **Regular users** can login and can management their applications as per need.
2. **System users** are created automatically for the internal services usage, authentication and platform management.

## Setup Requirements

1. Operating System: Red Hat Enterprise Linux 7 (64-bit)
2. Compatibility: ESXi 6.5 and later (VM version 13)
3. VMware Tools: Running, version:10309 (Guest Managed)
4. Master: 2core and 16 GB
5. Nodes: Two nodes with 2 core and 4GB RAM.
6. Domain name: console.Hobbylobby.com, \*.apps.Hobbylobby.com
7. Private IP Scheme: Any private scheme except 172.16.x.x/16
8. Connectivity: Private IP connectivity with password-less SSH access from master to all servers.

Note: Whatever commands and values are mentioned with italic and in yellow color need to be replaced with your own requirements.

## Setup Design



## On master and all nodes servers

Once the setup is ready with basic requirements, proceed for the below steps. The following commands need to be executed on all master and node servers included in your cluster.

1. Switch to root user using the following command.

sudo su -

cd

1. Update the yum packages using the following command.

yum update

reboot

1. Install the prerequisites with basic tools using the following command.

yum install -y wget git zile nano net-tools docker-1.13.1 bind-utils iptables-services bridge-utils bash-completion kexec-tools sos psacct openssl-devel httpd-tools NetworkManager python-cryptography python2-pip python-devel python-passlib java-1.8.0-openjdk-headless "@Development Tools"

1. Install the latest RHEL 7 epel repository using the following command.

yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

1. Enable the added RHEL 7 epel repository using the following command.

sed -i -e "s/^enabled=1/enabled=0/" /etc/yum.repos.d/epel.repo

1. Start the network manager service

systemctl start NetworkManager

systemctl enable NetworkManager

## Only on Master server or Ansible Admin server

The following steps only need to be performed on your server you deiced to act as ansible deployment server.

1. Installing ansible on master server using the following command.

yum -y --enablerepo=epel install ansible pyOpenSSL

1. Create the project directory using the following command.

*mkdir openshift*

*cd openshift*

1. Cloning the OpenShift ansible git repository using the following command.

git clone https://github.com/openshift/openshift-ansible.git

1. Check out the desired stable version ( e.g v3.9) using the following command.

cd openshift-ansible && git fetch && git checkout release-3.9 && cd ..

## On all master and nodes servers

OpenShift OKD needs a working DNS server that can resolve the master and node servers FQDNs. If your OpenShift OKD also need to be accessed from public network, then you must have proper domain name registered with the following records:

<Public ip of master server> <record type> <public domain name>

<x.x.x.x> A <master.domian.com>

<x.x.x.x> A <console.domian.com>

<x.x.x.x> A <\*.domian.com>

<x.x.x.x> A <\*apps.domian.com>

<x.x.x.x> A <node01.domian.com>

<x.x.x.x> A <node02.domian.com>

Alternatively, you can add all host entries in host file of all servers. However, \*.domain.com does not work for host file so either you must have proper DNS domain name, or you must register each project’s public domain name in your host file.

1. Make the appropriate changes in the /etc/hosts file on all servers.

vi /etc/hosts

*192.168.137.101 master master.stackroute.io*

*192.168.137.101 console.stackroute.io*

*192.168.137.102 node01 node01.stackroute.io*

*192.168.137.103 node02 node02.stackroute.io*

Note: You need to replace the appropriate IPs and Domain names as per your setup.

## Only on master server.

1. Restart and enable the docker service on all servers

systemctl stop docker && systemctl restart docker && systemctl enable docker

1. Generating the ssh keys and coping the public key to all servers for ansible connection from master.

ssh-keygen

for host in *192.168.137.101* \

*192.168.137.102* \

*192.168.137.103*; \

do ssh-copy-id -i ~/.ssh/id\_rsa.pub $host; \

done

1. Creating an inventory file for ansible playbook, paste the content of inventory.ini file in this file.

vi inventory.ini

The sample of inventory.ini file can be found here.

<https://github.com/vipin-k/Openshift-Cluster/blob/master/inventory.ini>

1. Execute the prerequisites ansible playbook using the following command.

cd openshift

ansible-playbook -i inventory.ini openshift-ansible/playbooks/prerequisites.yml

Note: Make sure the ansible user mentioned in the inventory file is able to access all master and node servers before to execute playbook.

1. Execute the OpenShift cluster deploy playbook using the following command.

ansible-playbook -i inventory.ini openshift-ansible/playbooks/deploy\_cluster.yml

Note: in inventory file you have two options: either have two node including master acting as one node, or have three sperate nodes if you don’t consider master as node.

1. If you get an error due to previous attempt, you can run the following command to remove the existing setup.

ansible-playbook -i inventory.ini openshift-ansible/playbooks/adhoc/uninstall.yml

1. Generating the OpenShift OKD admin password using the following command.

htpasswd -c /etc/origin/master/htpasswd admin

1. Login to **oc cluster** with the system admin user using the following command.

oc login -u system:admin

If you get oc command not found, then you may need to set the path using below command “export PATH=/usr/local/bin/”

1. Adding admin user as cluster admin role using the following command.

oc adm policy add-cluster-role-to-user cluster-admin admin

1. Listing the current nodes in cluster using the following command.

oc get nodes

1. Listing the current projects using the following command.

oc projects

1. Login to web console using the following web link.

[*https://console.stackroute.io:8443*](https://console.stackroute.io:8443)

Note: Make sure you open the port 8443 on your server, network and firewall. Also replace the appropriate domain URL registered for as per your setup.

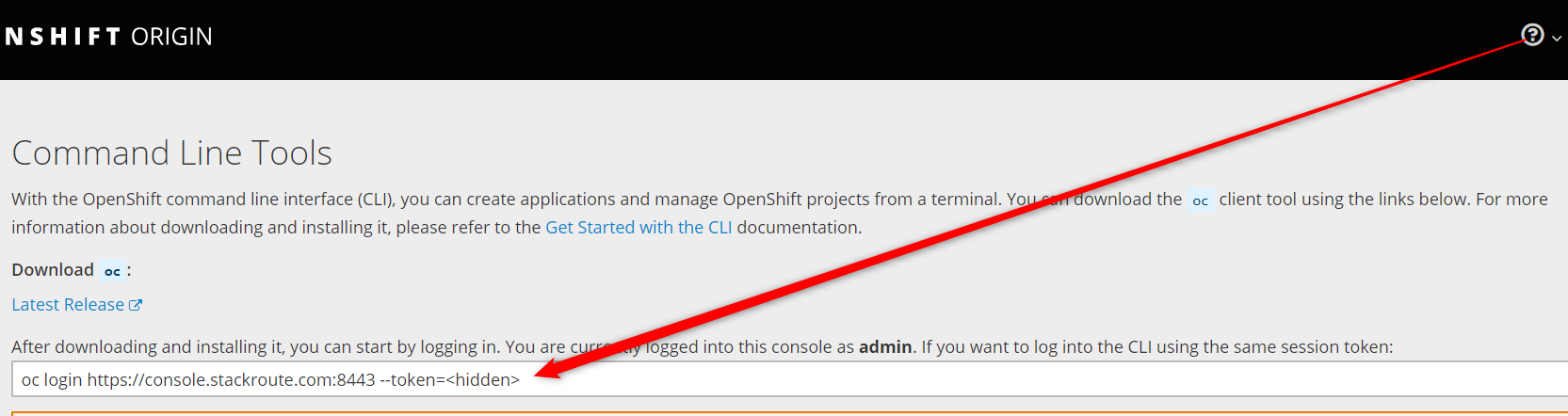
You should be able to login to OpenShift OKD cluster console. Now, you can proceed to explore the OpenShift OKD platform for creation of projects, building, deploying and scaling your apps. For more details how to use OpenShift OKD platform, please visit the following link and explore the functions, features, and options of OpenShift OKD platform.

<https://learn.openshift.com>

## Working with API Token

API token can be used for the REST API action instead of user name and password.

1. To get the API token login to Web console and copy the text as highlighted in the below figure.



1. Paste the copied login URL on master server to login using API token.

oc login https://console.stackroute.com:8443 --token=<you-token-key-here>

1. To get the API token, execute the following command:

oc whoami -t

1. Let’s see an example how to use API token to get the user details. For this make sure logged in with API token using above command and then create the following script.

vi get-user-details.sh

#!/bin/sh

SERVER=`oc whoami --show-server`

TOKEN=`oc whoami --show-token`

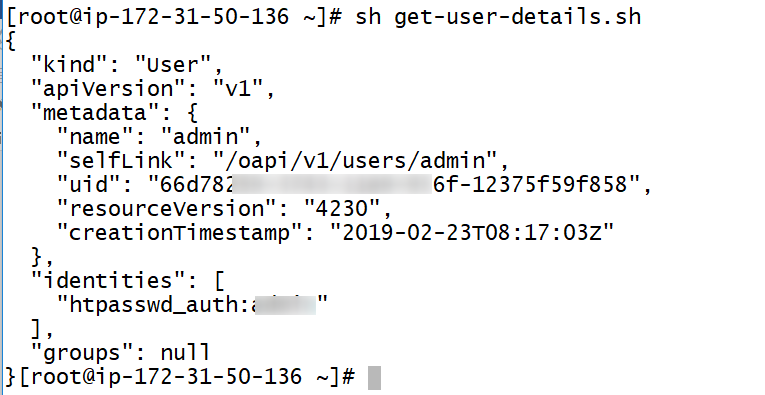
URL="$SERVER/oapi/v1/users/~"

curl -H "Authorization: Bearer $TOKEN" $URL

1. Now execute the script to verify whether token is working or not.

sh get-user-details.sh

1. You should get the result similar like shown below.



## Working with OpenShift Project

Before you can deploy any application, though, you first need to create a project in the OpenShift cluster to contain your applications. Whenever you work with OpenShift, you will work within the context of a project. An OpenShift project is a walled namespace used to hold everything related to a set of applications.

## Projects

* A project allows a group of users to organize and manage their content/applications in isolation from others.
* Access to projects is managed by administrator.
* Regular users automatically have access to their own projects.
* Cluster administrators can create projects and delegate administrative rights for the project to any member of the users.
* Project name you choose must be unique across the whole OpenShift cluster.
* Project name can only include lowercase letters, numbers, and the dash character.

1. Creating project

oc new-project stackroute-project --display-name 'Stackroute Project'

1. Viewing project

oc projects

1. Start using project

oc project stackroute-project

## Adding a Collaborator

More than one users can work on the same OpenShift project for collaboration. There are three collaborator roles:

1. **Admin**: A project manager with full control on project.
2. **Edit**: A user that can modify most objects in a project but does not have the power to view or modify roles or bindings.
3. **View**: A user who cannot make any modifications but can see most objects in a project.

### Adding users in OpenShift:

1. Use the below command to add a user:

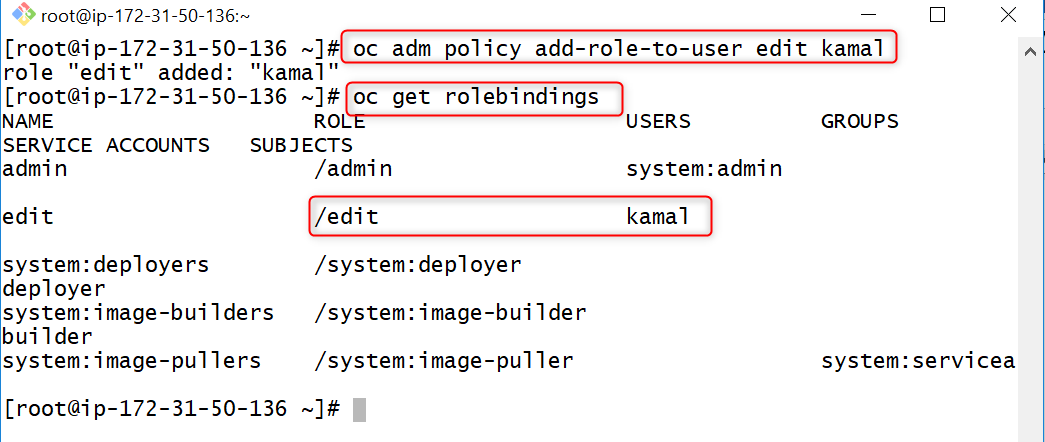
htpasswd -c /etc/origin/master/htpasswd <username>

1. To add a user to a project, run

oc adm policy add-role-to-user edit <collaborator>

1. To get a list of the users who have access to a project and their roles, run:

oc get rolebindings



1. To remove a user from a project, run:

oc adm policy remove-role-from-user edit <collaborator>

## Deploying Applications

The main methods for deploying an application are:

* From an existing container image hosted on an image registry located outside the OpenShift cluster.
* From an existing container image that has been imported into the image registry running inside the OpenShift cluster.
* From application source code in a Git repository hosting service using an **S2I builder**.
* From image source code in a Git repository hosting service using **Dockerfile**.
* From application source code pushed into OpenShift from a local filesystem using the command-line oc client.

The first image we will deploy is stored on Docker Hub and is named openshiftkatacoda/blog-django-py. The application in the image implements a simple blog site.

1. Use the following command to deploy a sample blog.

oc new-app openshiftkatacoda/blog-django-py --name stackroute-blog

1. You can check on the status of the overall project using the oc status command:

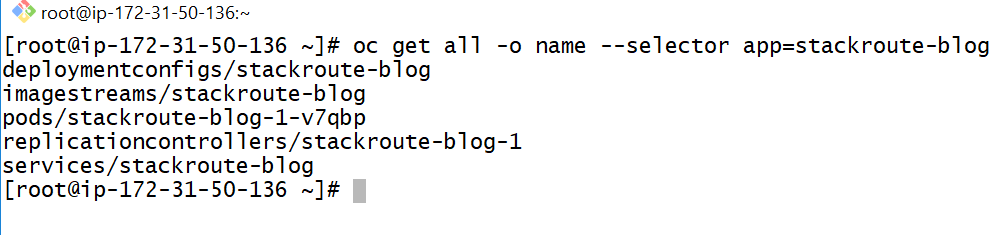
oc status

1. To get a list of the instances of the application that were deployed you can use the command oc get pods:

oc get pods

1. To get a list of the resource objects created for the application you can use the oc get all command:

oc get all -o name --selector app=stackroute-blog



* **deploymentconfig**: captures the details of how the deployment should be done.
* **imagestream**: is a record of the image you want deployed.
* **pods**: consists of a group of containers for an application.
* **replicationcontrollers**: responsible for replication between containers included for a pod.
* **service**: maintains a mapping to instances of your application so it can be accessed.

## Service, Networking, and Routing

1. Execute the following commands to get running pods:

oc get pods

1. Execute the following command to see the more details of specific pod:

oc describe *pod stackroute-blog-5-scnn8*

1. Execute the following command to see what is running inside a pod.

oc rsh *stackroute-blog-5-scnn8*

1. To see a list of both the pods and services for an application, you can run oc get pods,services and provide a label selector which matches that used by the application:

oc get pods,services --selector app=*stackroute-blog*

1. To see a list of the pod IP addresses associated with a service, you can use the oc get endpoints command:

oc get endpoints *stackroute-blog*

1. To expose a service for a web application so it can be accessed externally by a user, you can run the oc expose service command

oc expose service/*stackroute-blog*

oc describe route/*stackroute-blog*

**The route has the following syntax:**

**<route-name>-<project-name>.apps.<domain.com>**

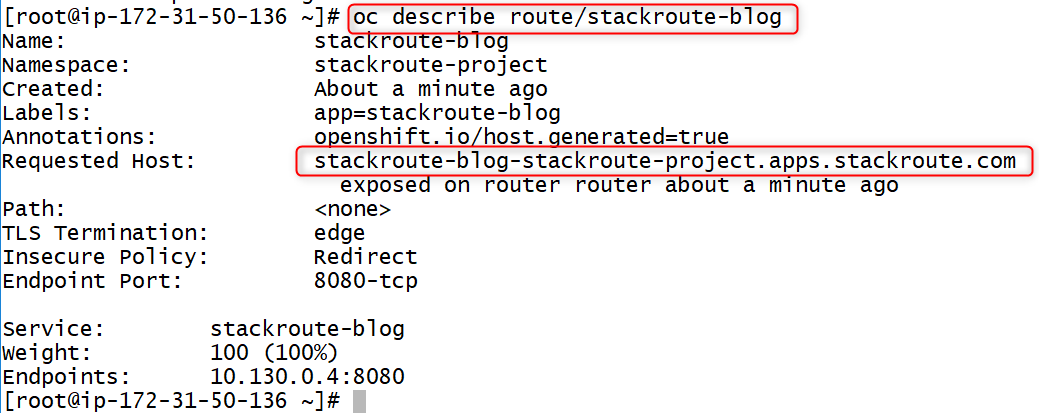
**Example: myapp-myproject.apps.stackroute.io**

1. The route created using oc expose only supports requests using the HTTP protocol. If you want clients to use a secure connection, you will need to create the route using the following command.

oc create route edge *blog-secure* --service *stackroute-blog*

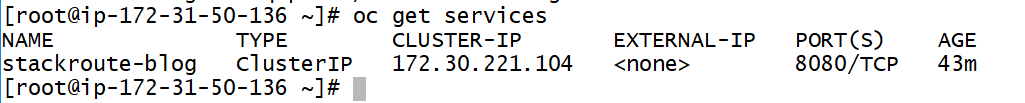
1. If you want users who attempt to use an insecure connection to be redirected so that a secure connection is used, run:

oc create route edge *stackroute-blog* --service *stackroute-blog* --insecure-policy Redirect



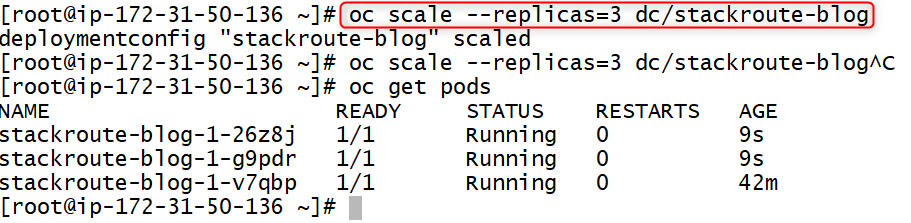
1. You can see what ports a service is advertised as using by running.

oc get services



1. You scale up and down your service using the following command.

oc scale --replicas=3 dc/stackroute-blog

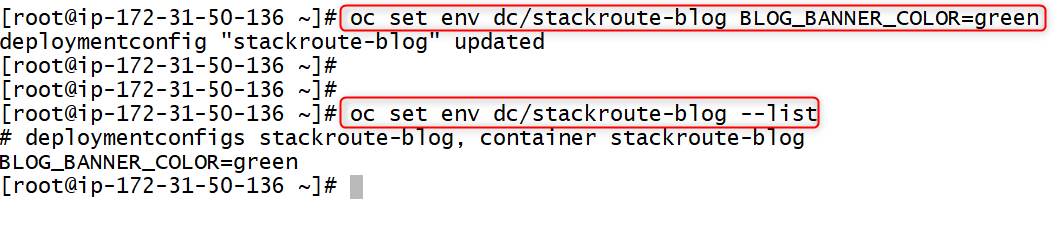


## Updating variables

Variables for an application can be either set before deploying app or after the application deployed. You can use following commands to update and verify variables.

oc set env dc/stackroute-blog BLOG\_BANNER\_COLOR=green

oc set env dc/stackroute-blog --list

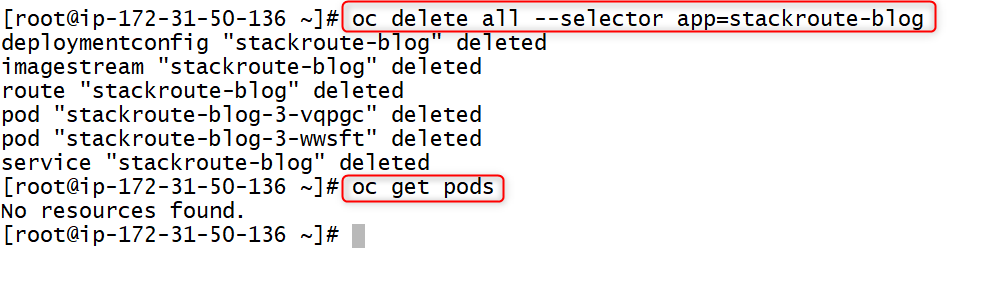


Once the variables are updated, the application will get redeployed automatically. Just refresh the browser to see the changes.

## Deleting an application using command

Use the following command to delete the deployed application and its associated service components.

oc delete all --selector app=stackroute-blog



## Building and Deploying from Source

OpenShift provides four different build strategies:

* **Source**: This uses Source-to-Image to produce ready-to-run images by injecting application source (or other assets) into a builder image.
* **Docker**: This uses docker build to take a Dockerfile and associated source files and create a runnable image.
* **Pipeline**: This uses Jenkins and a workflow defined by a Jenkinsfile to create a pipeline for building a runnable image.
* **Custom**: This uses your own custom image to control the build process for creating the runnable image.

1. The following command will use the S2I method to deploy a blog code hosted in GitHub repository.

oc new-app --name stackroute-blog python:3.5~https://github.com/openshift-katacoda/blog-django-py

1. To monitor the building of the application image as it occurs, you can run the command:

oc logs -f bc/stackroute-blog

1. A new build can be triggered using the oc start-build command:

oc get bc

oc start-build bc/stackroute-blog

1. Building from a local source is triggered using oc start-build, with the location of the local source directory specified using the --from-dir option:

oc start-build bc/stackroute-blog --from-dir=.

## Persistent Storage

OpenShift supports several underlying storage technologies including NFS, GlusterFS, Ceph RBD, OpenStack Cinder, AWS Elastic Block Storage, GCE Persistent Storage, Azure Disk, Azure File, iSCSI, Fibre Channel, and VMware vSphere.

Access modes for persistent storage are:

* ReadWriteOnce (RWO): The volume can be mounted as read/write by a single node.
* ReadOnlyMany (ROX): The volume can be mounted as read-only by many nodes.
* ReadWriteMany (RWX): The volume can be mounted as read/write by many nodes.

## Using NFS Server with OpenShift

Using NFS with OpenShift has been explained in the below article. For concept and more details please visit this link.

<https://docs.openshift.com/container-platform/3.3/install_config/persistent_storage/persistent_storage_nfs.html#install-config-persistent-storage-persistent-storage-nfs>

Before you could use NFS server with OpenShift, make sure you have done the following prerequisites.

1. Install and configure NFS server.
2. Export the volumes in /etc/exportfs file with all nodes which are going to use these NFS shares.
3. Open necessary ports and set read and write permissions for all shared volumes.

Once your NFS server is ready, proceed with the following steps on master.

1. Create a PV Object Definition file with appropriate settings.

sudo vi nfs-pv.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv0001

spec:

capacity:

storage: 5Gi

accessModes:

- ReadWriteOnce

nfs:

path: /tmp

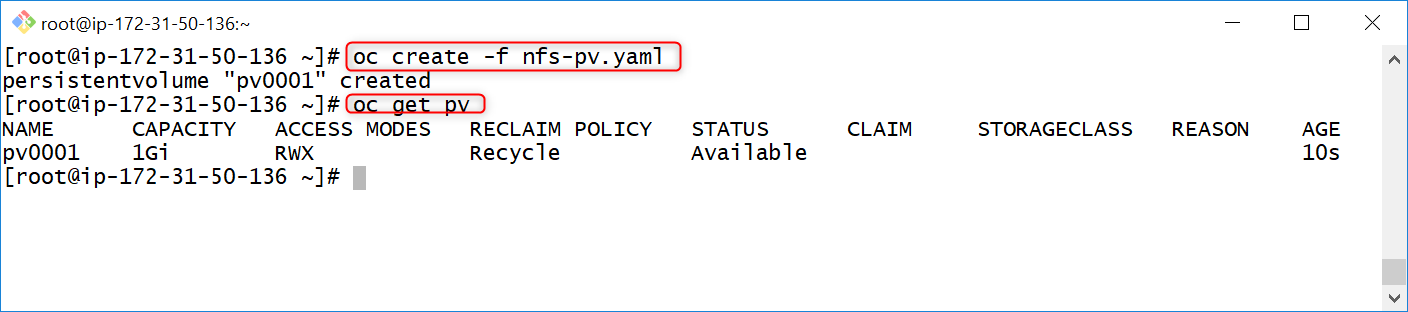
server: 172.17.0.2

persistentVolumeReclaimPolicy: Recycle

1. Save the file and execute the following command to create a persistent volume.

oc create -f nfs-pv.yaml

oc get pv



1. The next step can be to create a PVC, which binds to the new PV:

sudo vi nfs-claim.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: nfs-claim1

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

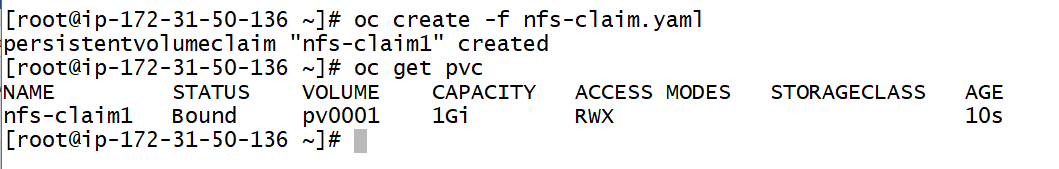
storage: 1Gi

volumeName: "pv0001"

1. Save the file and execute the following command to create the claimed storage.

oc create -f nfs-claim.yaml

oc get pvc



Now you can mount this claimed volume to any pod. When you need persistent storage for an application, you need to make a persistent volume claim. When making the claim, you must specify the size of the persistent volume you want.

1. The following example create and claim 1Gigabit volume for blog pod.

oc set volume dc/stackroute-blog --add --type=pvc --claim-size=1Gi --claim-mode=ReadWriteMany --claim-name nfs-claim1 --name nfs-data --mount-path /opt/app-root/src/media –overwrite

1. If you have single NFS volume and want to use this volume for multiple application yet to keep data separate for reach application, you can create sub directories and mount it using sub-path option.

oc set volume dc/stackroute-blog --add --type=pvc --claim-size=1Gi --claim-mode=ReadWriteMany --claim-name nfs-claim-new --name new-data --mount-path /opt/app-root/src/data --path /nfs-data --sub-path stackroute-blog --overwrite

1. To list the persistent volumes that have been added against an application, run oc set volume against the deployment configuration without any additional arguments:

oc set volume dc/stackroute-blog

1. To stop using a persistent volume with an application, you can use the **oc set volume --remove** command. You must supply the name used to identify the volume mount in the deployment configuration:

oc set volume dc/stackroute-blog --remove --name nfs-data

1. If you no longer need the persistent volume, you can release it by running the **oc delete pvc** command on the persistent volume claim:

oc delete pvc/*nfs-data*

1. The reclaim policy of a persistent volume will usually be Recycle, meaning that as soon as you delete the persistent volume, its contents will be deleted, and the persistent volume will be returned to the pool of available persistent volumes. You can then run **oc rsync** to copy the directory.

mkdir -p /tmp/images

cd /tmp/images

touch kamal.jpg

oc rsync /tmp/images stackroute-blog-20-ttw4h:/opt/app-root/src/media --no-perms

oc rsh stackroute-blog-20-ttw4h

cd media

ls

You should be able to see kamal.jpg inside this container directory.

## Quota and Limits

### Quota

Quotas are used to manage two categories of resources.

1. The first category is resource objects: No. of projects, applications, persistent volumes you can use.
2. The second category is compute resources: RAM, CPU

When quotas are applied on a per-project basis, you can view them with the **oc describe quota** command.

If the quota is across multiple projects, you would instead view them with the **oc describe appliedclusterresourcequota** command

### Limits

How many compute resources an individual application can consume is controlled by a limit range. A limit range is applied on a per-project basis.

The limit ranges can be viewed from the command line using **oc describe** limits.

1. To modify the resource requirements for your application, you can run the **oc set resources** command:

oc set resources dc/blog --limits memory=512Mi

oc set resources dc/kamal-nginx --limits cpu=500m

1. To specify quota and limit while running application from image, use the following command.

oc run ruby-hello-world --image=ruby-hello-world --limits=cpu=200m,memory=400Mi --requests=cpu=100m,memory=200Mi

### Additional reference link:

<https://docs.openshift.com/container-platform/3.6/dev_guide/compute_resources.html>

## Deployment Strategy

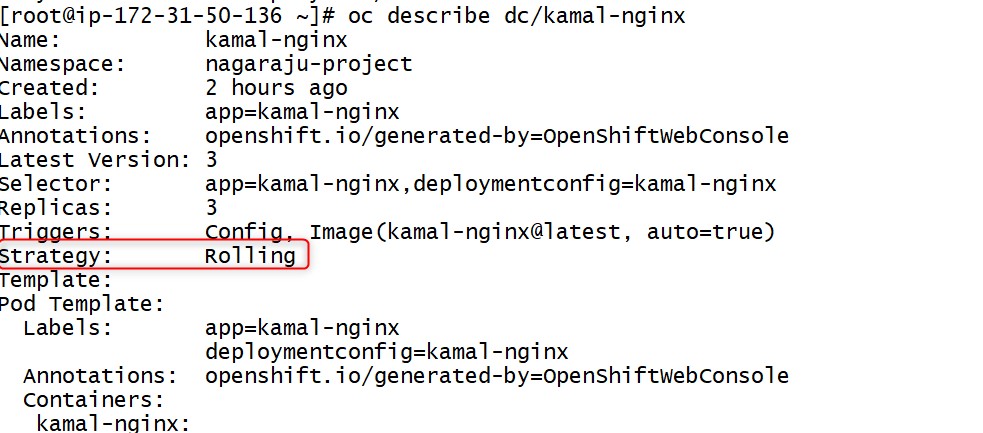
A deployment strategy defines the process by which a new version of your application is started, and the existing instances shut down.

OpenShift implements two basic deployment strategies. The default deployment strategy is a Rolling deployment.

The second deployment strategy is Recreate. This is used when you cannot have multiple instances, or versions, of your application running at the same time.

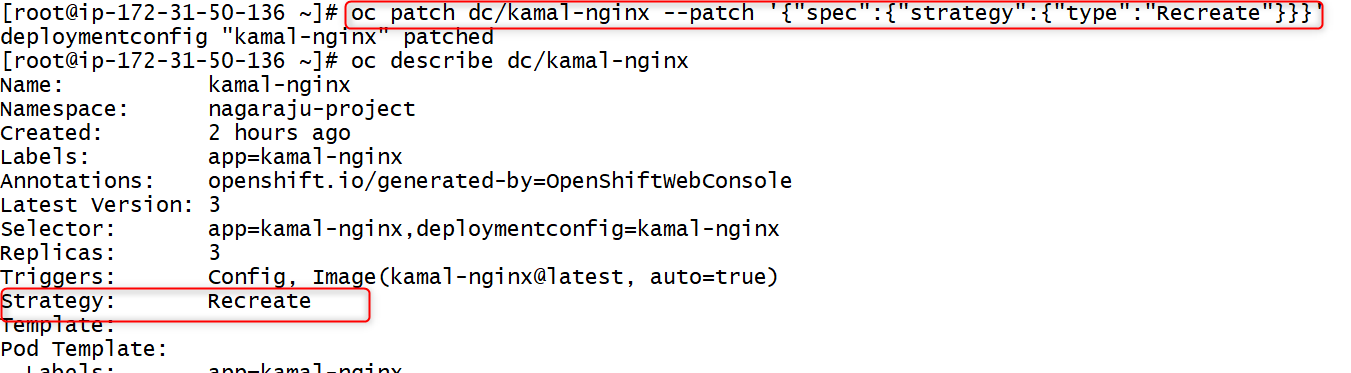
1. To view what strategy a deployment is using, run the **oc describe** command

oc describe dc/<app-name>



1. To change the Deployment strategy, use the following command.

oc patch dc/kamal-nginx --patch '{"spec":{"strategy":{"type":"Recreate"}}}'



### Hooks

To enable additional actions when a deployment is occurring, you can define two types of lifecycle hooks:

1. **Pre**: Executed before the first new instance of your application is created for the new deployment.
2. **Post**: Executed after all instances of your application for a deployment have been successfully started and the old instances have been shut down.
3. **Mid**: This hook is executed after all old instances of your application have been shut down, but before any new instances of your application have been started.

An example of a command from a pre-lifecycle hook: enable a database flag to put an application in read-only mode. When the deployment had completed successfully, the post lifecycle hook could disable the flag. The mid hook can be used to safely run any database migrations, as your application will not be running at that point.

## Logging and Debugging

1. To view the build logs for the last build run, you can run the oc logs command against the build configuration for your application:

oc logs bc/blog

1. You could also run oc get pods to see the list of all pods and run oc logs on the build pod:

oc logs pods/blog-1-build

1. If you want to monitor a set of resource objects over time, you can pass the --watch option to oc get.

oc get pods --watch

1. To monitor what OpenShift is doing for applications running in your project, you can use oc get events, passing the --watch option

oc get events --watch

1. You can gain access to the container and run an interactive shell. To do this, run oc rsh against the name of the pod:

oc rsh blog-4-d6xbx

1. To debug a container that will not start you can use the oc debug command, running it against the deployment configuration for your application:

oc debug dc/blog