OPENSHIFT

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

OpenShift is a cloud development Platform as a Service (PaaS) developed by Red Hat. It is an open source development platform, which enables the developers to develop and deploy their applications on cloud infrastructure. It is very helpful in developing cloud-enabled services.

This tutorial will help you understand OpenShift and how it can be used in the existing infrastructure. All the examples and code snippets used in this tutorial are tested and working code, which can be simply used in any OpenShift setup by changing the current defined names and variables.

Audience

This tutorial has been prepared for those who want to understand the features and functionalities of OpenShift and learn how it can help in building cloud-enabled services and applications.

After completing this tutorial, readers will be at a moderate level of understanding of OpenShift and its key building block. It will also give a fair idea on how to configure OpenShift in a preconfigured infrastructure and use it.

Prerequisites

Readers who want to understand and learn OpenShift should have a basic knowledge of Docker and Kubernetes. Readers also need to have some understanding of system administration, infrastructure, and network protocol communication.

Copyright & Disclaimer

Table of Contents

1. OpenShift - Overview

OpenShift is a cloud development Platform as a Service (PaaS) hosted by Red Hat. It's an open source cloud-based user-friendly platform used to create, test, and run applications, and finally deploy them on cloud.

OpenShift is capable of managing applications written in different languages, such as Node.js, Ruby, Python, Perl, and Java. One of the key features of OpenShift is it is extensible, which helps the users support the application written in other languages.

OpenShift comes with various concepts of virtualization as its abstraction layer. The underlying concept behind OpenShift is based on virtualization.

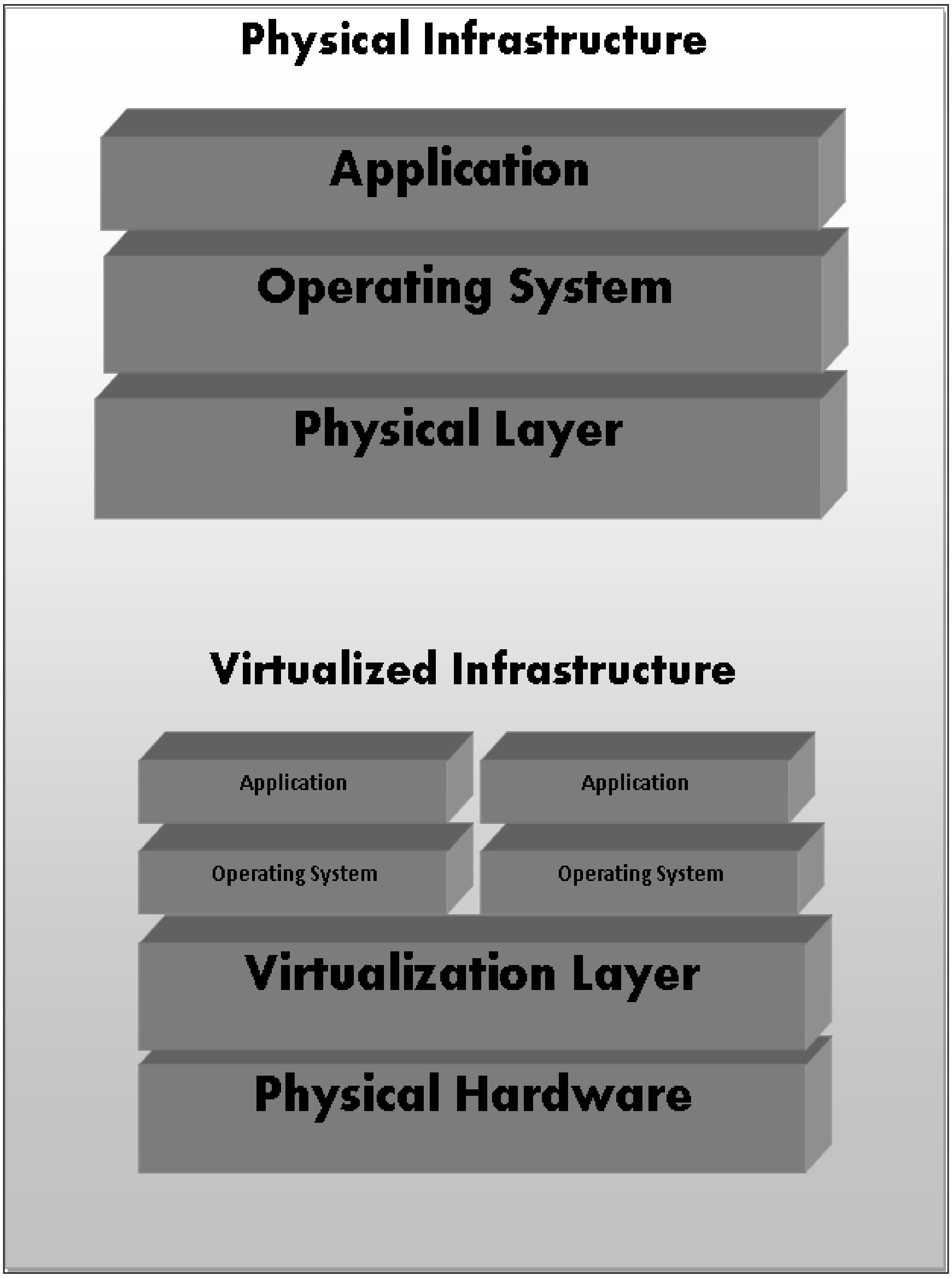
Virtualization

In general, virtualization can be defined as the creation of a virtual system rather than physical or actual version of anything starting from system, storage, or an operating system. The main goal of virtualization is to make the IT infrastructure more scalable and reliable. The concept of virtualization has been in existence from decades and with the evolution of IT industry today, it can be applied to a wide range of layers starting from System level, Hardware level, to Server level virtualization.

How It Works

It can be described as a technology in which any application or operating system is abstracted from its actual physical layer. One key use of the virtualization technology is server virtualization, which uses a software called hypervisor to abstract the layer from the underlying hardware. The performance of an operating system running on virtualization is as good as when it is running on the physical hardware. However, the concept of virtualization is popular as most of the system and application running do not require the use of the underlying hardware.

Physical vs Virtual Architecture



Types of Virtualization

Application Virtualization: In this method, the application is abstracted from the underlying operating system. This method is very useful in which the application can be run in isolation without being dependent on the operating system underneath.

Desktop Virtualization: This method is used to reduce the workstation load in which one can access the desktop remotely, using a thin client at the desk. In this method, the desktops are mostly run in a datacenter. A classic example can be a Virtual Desktop Image (VDI) which is used in most of the organizations.

Data Virtualization: It is a method of abstracting and getting away from traditional method of data and data management.

Server Virtualization: In this method, server-related resources are virtualized which includes the physical server, process, and operating system. The software which enables this abstraction is often referred to as the hypervisor.

Storage Virtualization: It is the process of pooling in multiple storage devices into a single storage device that is managed from a single central console.

Network Virtualization: It is the method in which all available network resources are combined by splitting up the available bandwidth and channels, each of which is independent of each other.

OpenShift

OpenShift is a cloud-enabled application Platform as a Service (PaaS). It's an open source technology which helps organizations move their traditional application infrastructure and platform from physical, virtual mediums to the cloud.

OpenShift supports a very large variety of applications, which can be easily developed and deployed on OpenShift cloud platform. OpenShift basically supports three kinds of platforms for the developers and users.

Infrastructure as a Service (IaaS)

In this format, the service provider provides hardware level virtual machines with some pre-defined virtual hardware configuration. There are multiple competitors in this space starting from AWS Google cloud, Rackspace, and many more.

The main drawback of having IaaS after a long procedure of setup and investment is that, one is still responsible for installing and maintaining the operating system and server packages, managing the network of infrastructure, and taking care of the basic system administration.

Software as a Service (SaaS)

With SaaS, one has the least worry about the underlying infrastructure. It is as simple as plug and play, wherein the user just has to sign up for the services and start using it. The main drawback with this setup is, one can only perform minimal amount of customization, which is allowed by the service provider. One of the most common example of SaaS is Gmail, where the user just needs to login and start using it. The user can also make some minor modifications to his account. However, it is not very useful from the developer's point of view.

Platform as a Service (PaaS)

It can be considered as a middle layer between SaaS and IaaS. The primary target of PaaS evaluation is for developers in which the development environment can be spin up with a few commands. These environments are designed in such a way that they can satisfy all the development needs, right from having a web application server with a database. To do this, you just require a single command and the service provider does the stuff for you.

Why Use OpenShift?

OpenShift provides a common platform for enterprise units to host their applications on cloud without worrying about the underlying operating system. This makes it very easy to use, develop, and deploy applications on cloud. One of the key features is, it provides managed hardware and network resources for all kinds of development and testing. With OpenShift, PaaS developer has the freedom to design their required environment with specifications.

OpenShift provides different kind of service level agreement when it comes to service plans.

Free: This plan is limited to three gears with 1GB space for each.

Bronze: This plan includes 3 gears and expands up to 16 gears with 1GB space per gear.

Sliver: This is 16-gear plan of bronze, however, has a storage capacity of 6GB with no additional cost.

Other than the above features, OpenShift also offers on-premises version known as OpenShift Enterprise. In OpenShift, developers have the leverage to design scalable and non-scalable applications and these designs are implemented using HAproxy servers.

Features

There are multiple features supported by OpenShift. Few of them are -

- Multiple Language Support
- Multiple Database Support
- Extensible Cartridge System
- Source Code Version Management
- One-Click Deployment
- Multi Environment Support
- Standardized Developers' workflow

- Dependency and Build Management
- Automatic Application Scaling
- Responsive Web Console
- Rich Command-line Toolset
- Remote SSH Login to Applications
- Rest API Support
- Self-service On Demand Application Stack
- Built-in Database Services
- Continuous Integration and Release Management
- IDE Integration
- Remote Debugging of Applications

2. OpenShift - Types

OpenShift came into existence from its base named OpenShift V2, which was mainly based on the concept of gear and cartridges, where each component has its specifications starting from machine creation till application deployment, right from building to deploying the application.

Cartridges: They were the focal point of building a new application starting from the type of application the environment requires to run them and all the dependencies satisfied in this section.

Gear: It can be defined as the bear metal machine or server with certain specifications regarding the resources, memory, and CPU. They were considered as a fundamental unit for running an application.

Application: These simply refer to the application or any integration application that will get deployed and run on OpenShift environment.

As we go deeper in the section, we will discuss on different formats and offerings of OpenShift. In the earlier days, OpenShift had three major versions.

OpenShift Origin: This was the community addition or open source version of OpenShift. It was also known as upstream project for other two versions.

OpenShift Online: It is a pubic PaaS as a service hosted on AWS.

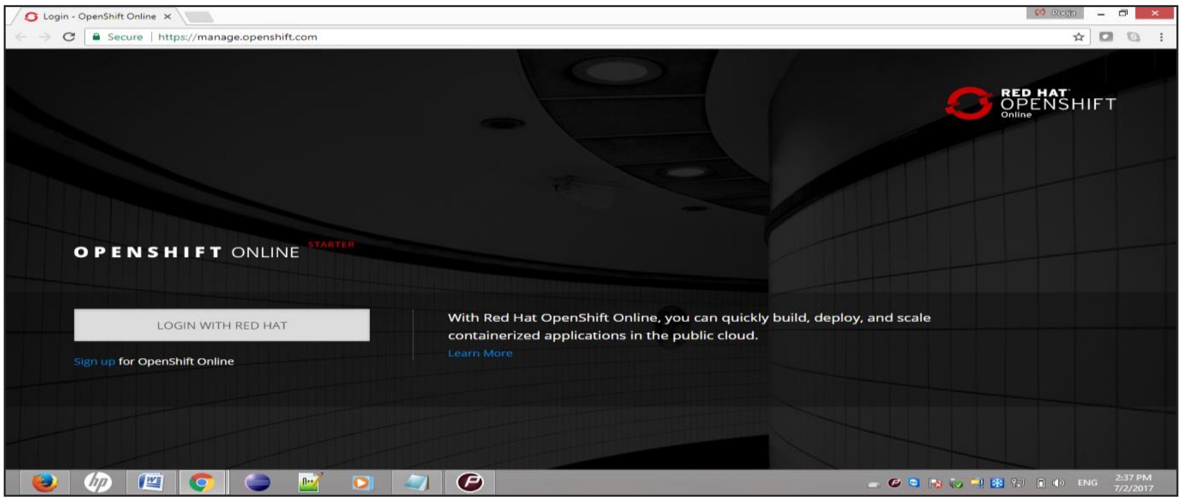
OpenShift Enterprise: This is the hardened version of OpenShift with ISV and vendor licenses.

OpenShift Online

OpenShift online is an offering of OpenShift community using which one can quickly build, deploy, and scale containerized applications on the public cloud. It is Red Hat's public cloud application development and hosting platform, which enables automated provisioning, management and scaling of application which helps the developer focus on writing application logic.

Setting Up Account on Red Hat OpenShift Online

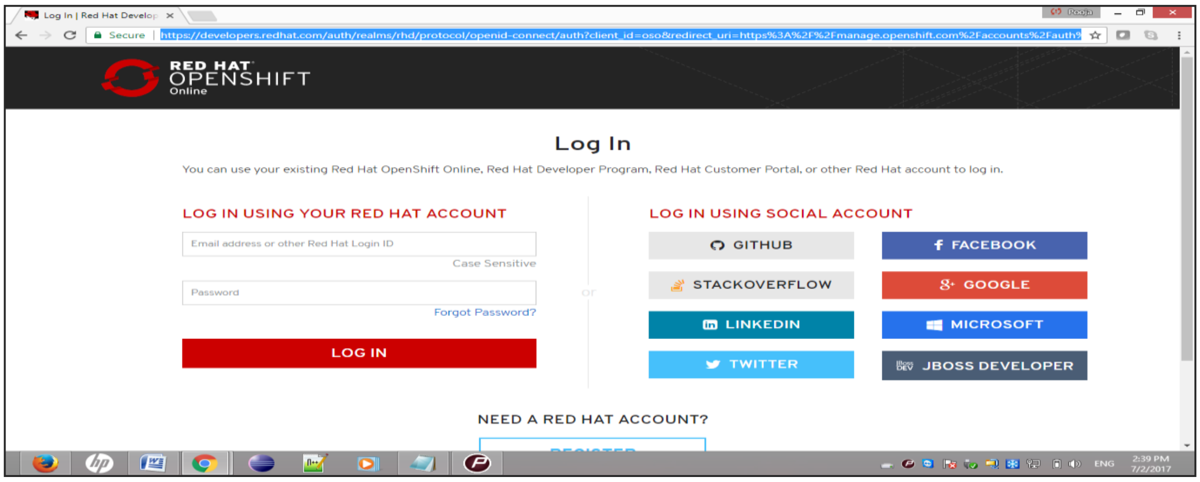
Step 1: Go to browser and visit the site https://manage.openshift.com/



Sign up for OpenShift Online

With Red Hat OpenShift Online, you can quickly build, deploy, and scale containerized applications in the public cloud.

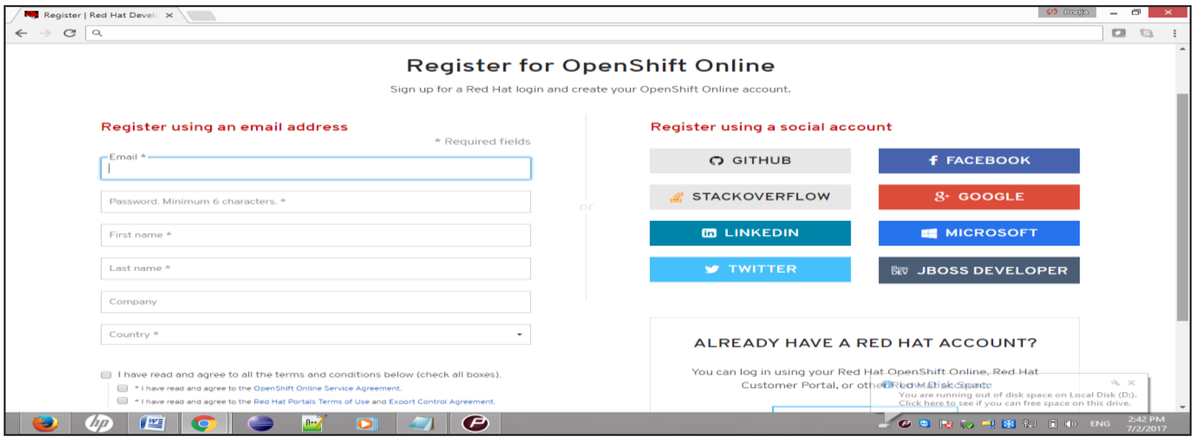
Step 2: If you have a Red Hat account, login to OpenShift account using the Red Hat login ID and password using the following URL. https://developers.redhat.com/auth/realms/rhd/protocol/openid-connect/auth?client_id=oso&redirect_uri=https%3A%2F%2Fmanage.openshift.com%2Faccounts%2Fauth%2Fkeycloak%2Fcallback&response_type=code&scope=openid+profile+email&state=b73466d00a5b3b4028ca95eac867e2dd



You can use your existing Red Hat OpenShift Online, Red Hat Developer Program, Red Hat Customer Portal, or other Red Hat account to log in.

Step 3: If you do not have a Red Hat account login, then sign up for OpenShift online service using the following link.

https://developers.redhat.com/auth/realms/rhd/login-actions/registration?code=G4w-myLd3GCH_QZCqMUmIOQlU7DIf_gfIvGu38nnzZQ.cb229a9d-3cff-4c58-b7f6-7b2c9eb17926



Register for OpenShift Online

Sign up for a Red Hat login and create your OpenShift Online account.

Register using an email address

* Required fields

Email *

Password. Minimum 6 characters. *

First name *

Last name *

Company

Country *

I have read and agree to all the terms and conditions below (check all boxes).

* I have read and agree to the OpenShift Online Service Agreement.

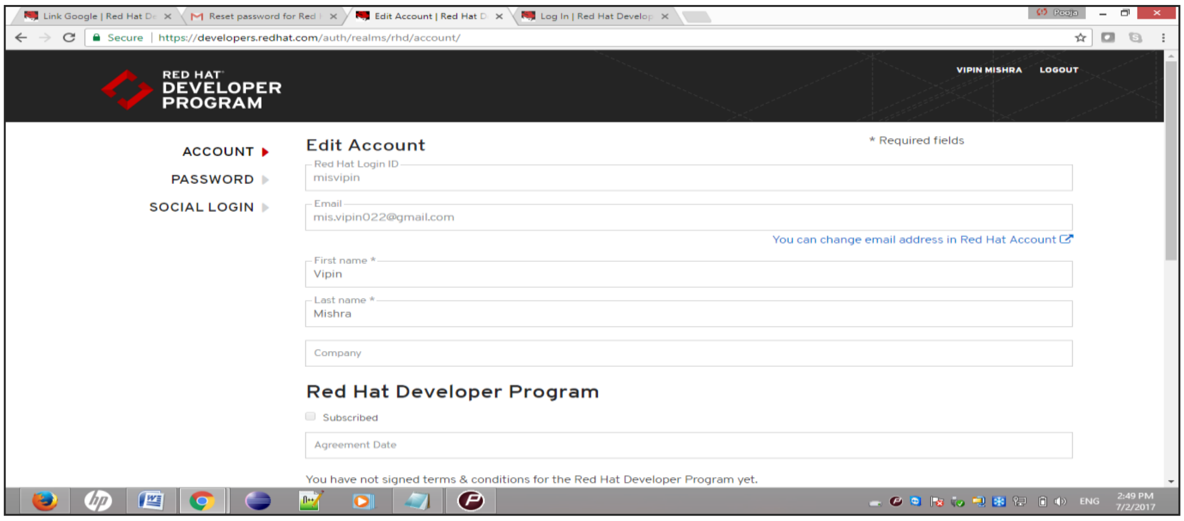
* I have read and agree to the Red Hat Portals Terms of Use and Export Control Agreement.

Register using a social account

ALREADY HAVE A RED HAT ACCOUNT?

You can log in using your Red Hat OpenShift Online, Red Hat Customer Portal, or othe

After login, you will see the following page.



https://developers.redhat.com/auth/realms/rhd/account/

Edit Account

* Required fields

Red Hat Login ID: misvipin

Email: mis.vipin022@gmail.com

You can change email address in Red Hat Account

First name *: Vipin

Last name *: Mishra

Company

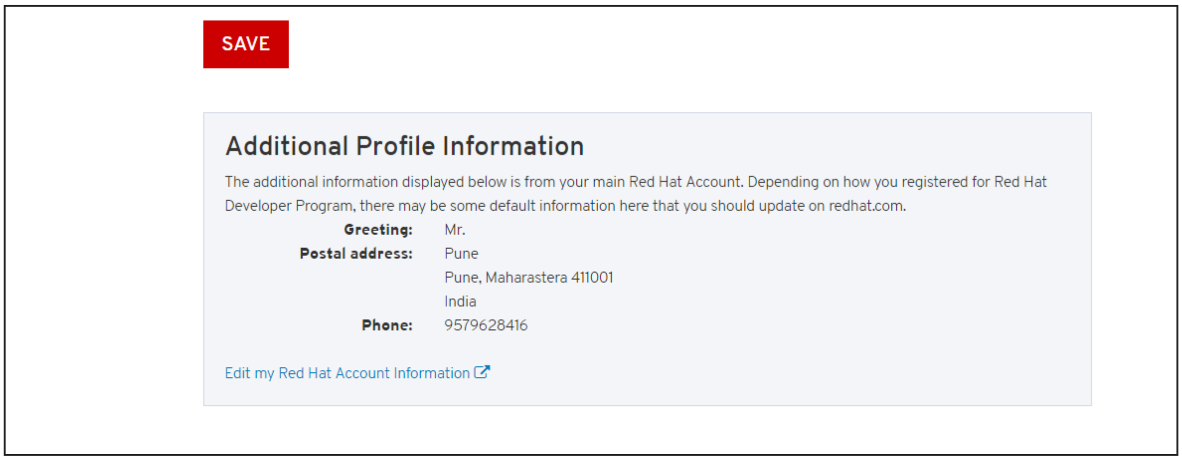
Red Hat Developer Program

Subscribed

Agreement Date

You have not signed terms & conditions for the Red Hat Developer Program yet.

Once you have all the things in place, Red Hat will show some basic account details as shown in the following screenshot.



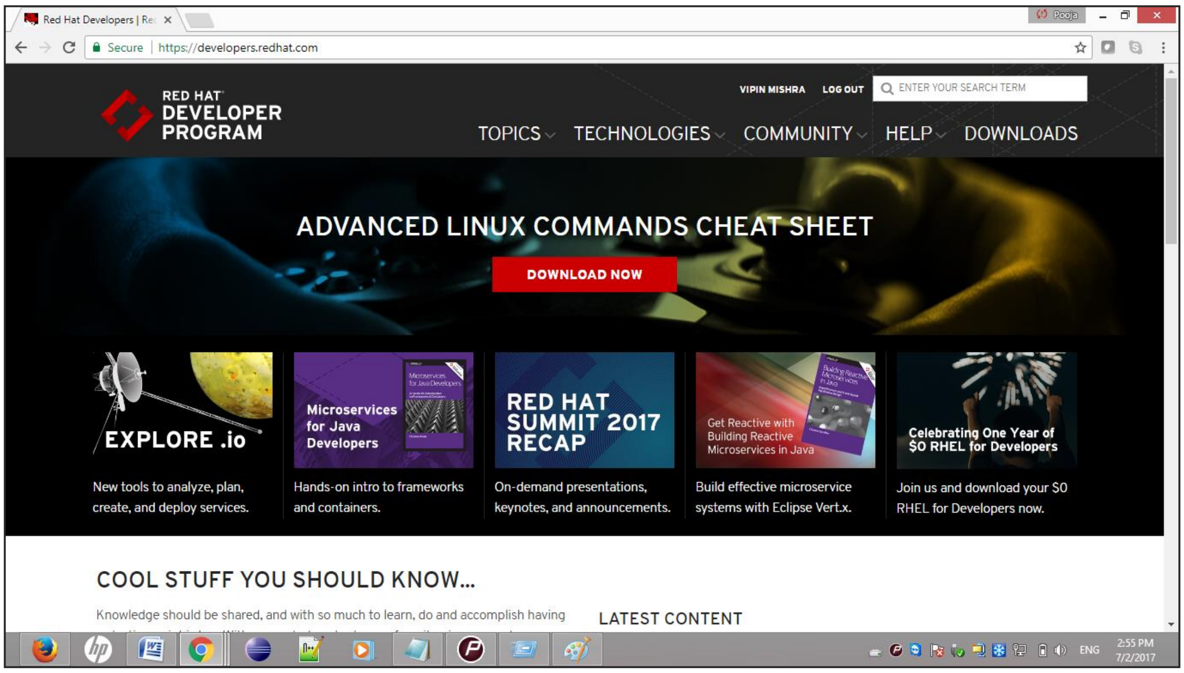
Additional Profile Information

The additional information displayed below is from your main Red Hat Account. Depending on how you registered for Red Hat Developer Program, there may be some default information here that you should update on redhat.com.

Greeting:	Mr.
Postal address:	Pune Pune, Maharastera 411001 India
Phone:	9579628416

Edit my Red Hat Account Information

Finally, when you are logged in, you will see the following page.


Red Hat Developers | Re

Secure | https://developers.redhat.com

RED HAT DEVELOPER PROGRAM

VIPIN MISHRA LOG OUT ENTER YOUR SEARCH TERM

TOPICS TECHNOLOGIES COMMUNITY HELP DOWNLOADS

ADVANCED LINUX COMMANDS CHEAT SHEET

DOWNLOAD NOW

EXPLORE .io
New tools to analyze, plan, create, and deploy services.

Microservices for Java Developers
Hands-on intro to frameworks and containers.

RED HAT SUMMIT 2017 RECAP
On-demand presentations, keynotes, and announcements.

Get Reactive with Building Reactive Microservices in Java
Build effective microservice systems with Eclipse Vert.x.

Celebrating One Year of $0 RHEL for Developers
Join us and download your $0 RHEL for Developers now.

COOL STUFF YOU SHOULD KNOW...

Knowledge should be shared, and with so much to learn, do and accomplish having

LATEST CONTENT

2:55 PM 7/2/2017

OpenShift Container Platform

OpenShift container platform is an enterprise platform which helps multiple teams such as development and IT operations team to build and deploy containerized infrastructure. All the containers built in OpenShift uses a very reliable Docker containerization technology, which can be deployed on any data center of publically hosted cloud platforms.

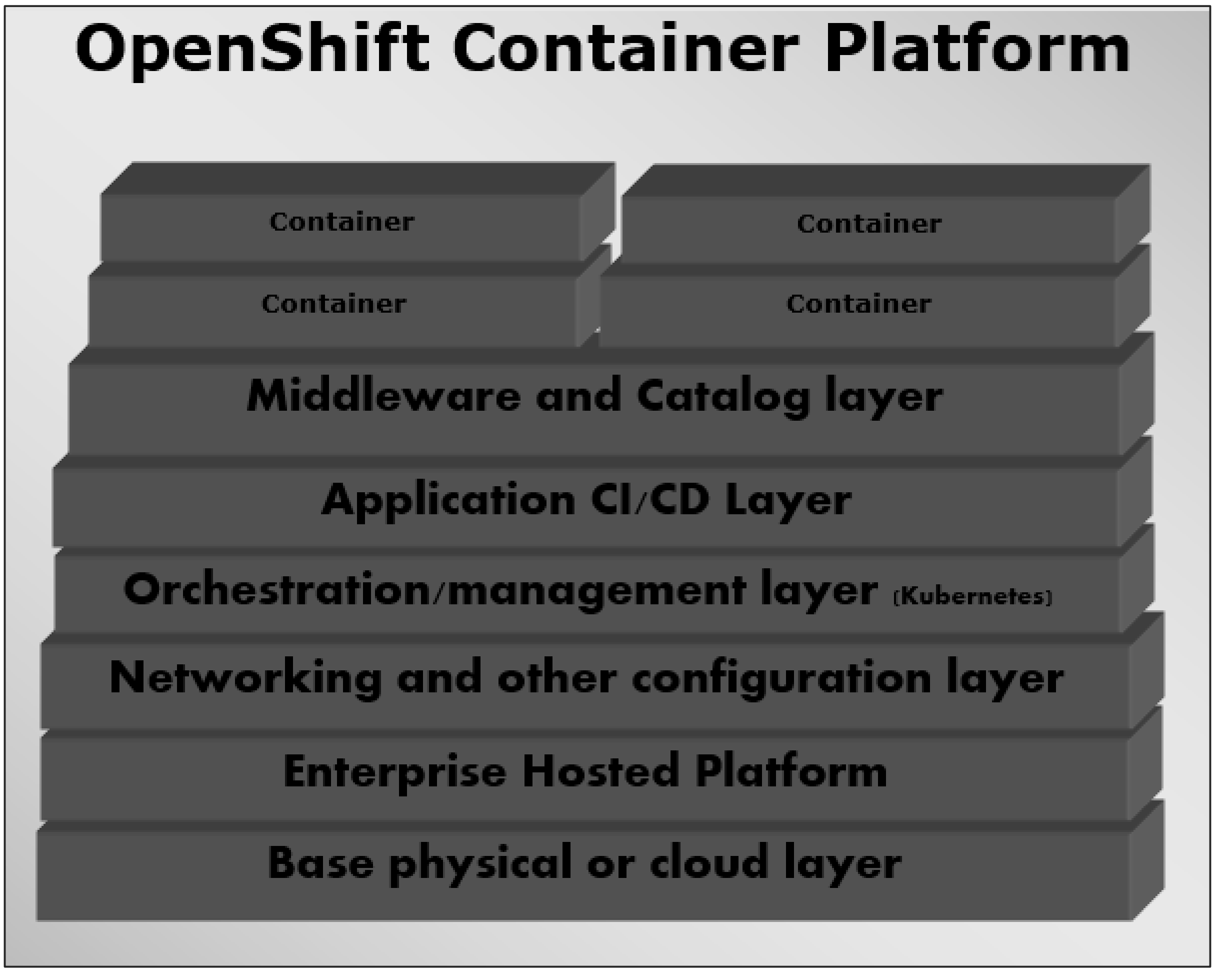
OpenShift container platform was formally known as OpenShift Enterprises. It is a Red Hat on-premise private platform as service, built on the core concept of application containers powered by Docker, where orchestration and administration is managed by Kubernetes.

In other words, OpenShift brings Docker and Kubernetes together to the enterprise level. It is a container platform software for enterprise units to deploy and manage applicants in an infrastructure of own choice. For example, hosting OpenShift instances on AWS instances.

OpenShift container platform is available in **two package levels**.

OpenShift Container Local: This is for those developers who wish to deploy and test applications on the local machine. This package is mainly used by development teams for developing and testing applications.

OpenShift Container Lab: This is designed for extended evaluation of application starting from development till deployment to pre-prod environment.


OpenShift Container Platform
Container
Container
Container
Container
Middleware and Catalog layer
Application CI/CD Layer
Orchestration/management layer (Kubernetes)
Networking and other configuration layer
Enterprise Hosted Platform
Base physical or cloud layer

OpenShift Dedicated

This is another offering added to the portfolio of OpenShift, wherein there is a customer choice of hosting a containerized platform on any of the public cloud of their choice. This gives the end user a true sense of multi-cloud offering, where they can use OpenShift on any cloud which satisfies their needs.

This is one of the newest offering of Red Hat where the end user can use OpenShift to build test deploy and run their application on OpenShift which is hosted on cloud.

Features of OpenShift Dedicated

OpenShift dedicated offers customized solution application platform on public cloud and it is inherited from OpenShift 3 technology.

- **Extensible and Open**: This is built on the open concept of Docker and deployed on cloud because of which it is can expend itself as and when required.
- **Portability**: As it is built using Docker, the applications running on Docker can easily be shipped from one place to the other, where Docker is supported.
- **Orchestration**: With OpenShift 3, one of the key features of container orchestration and cluster management is supported using Kubernetes which came into offering with OpenShift version 3.
- **Automation**: This version of OpenShift is enabled with the feature of source code management, build automation, and deployment automation which makes it very popular in the market as a Platform as a Service provider.

Competitors of OpenShift

Google App Engine: This is Google's free platform for developing and hosting web applications. Google's app engine offers fast development and deployment platform.

Microsoft Azure: Azure cloud is hosted by Microsoft on their data centers.

Amazon Elastic Cloud Compute: They are built-in services provided by Amazon, which help in developing and hosting scalable web applications on cloud.

Cloud Foundry: It is an open source PaaS platform for Java, Ruby, Python, and Node.js applications.

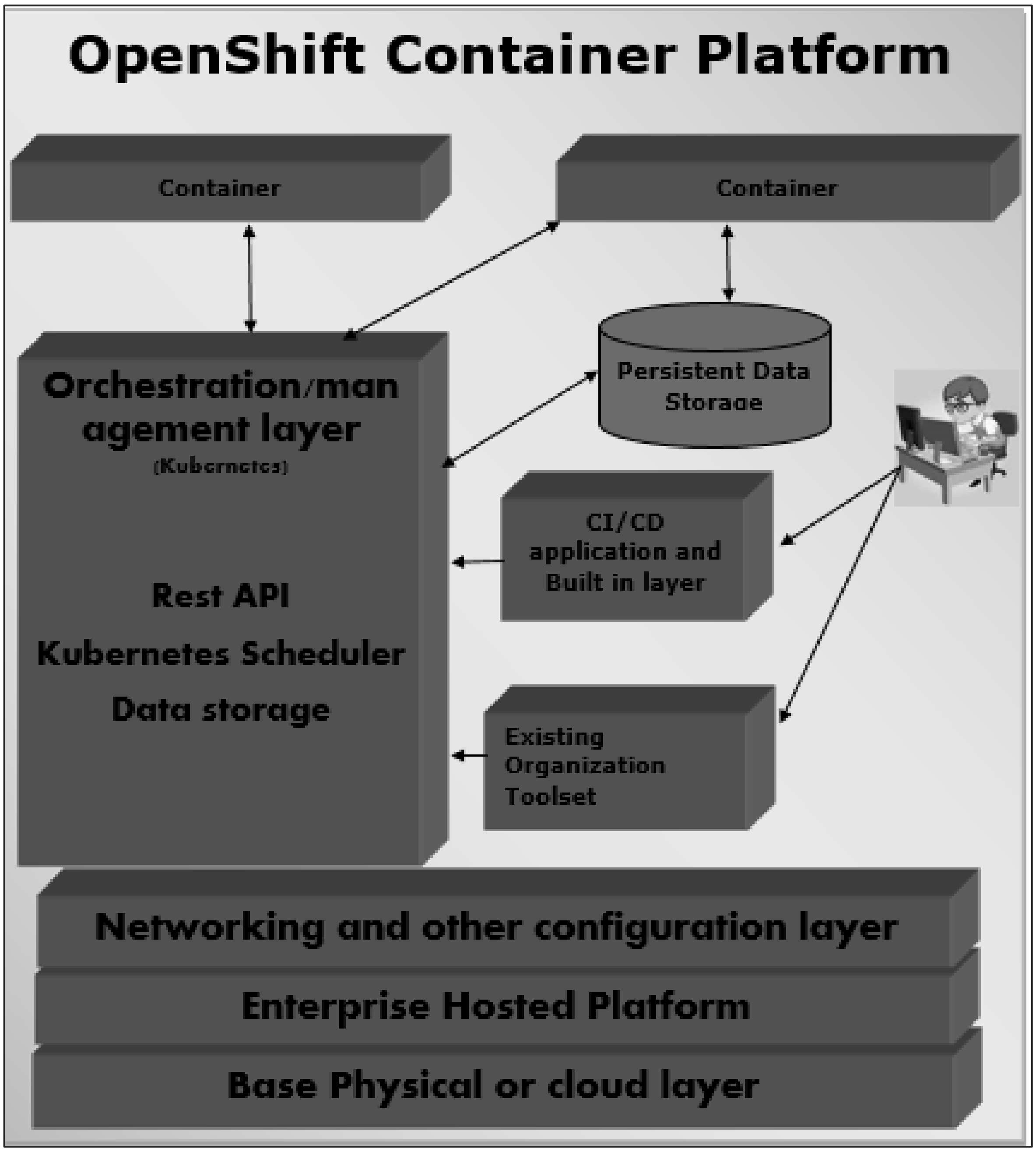
CloudStack: Apache's CloudStack is a project developed by Citrix and is designed to become a direct competitor of OpenShift and OpenStack.

OpenStack: Another cloud technology provided by Red Hat for cloud computing.

Kubernetes: It is a direct orchestration and cluster management technology built to manage Docker container.

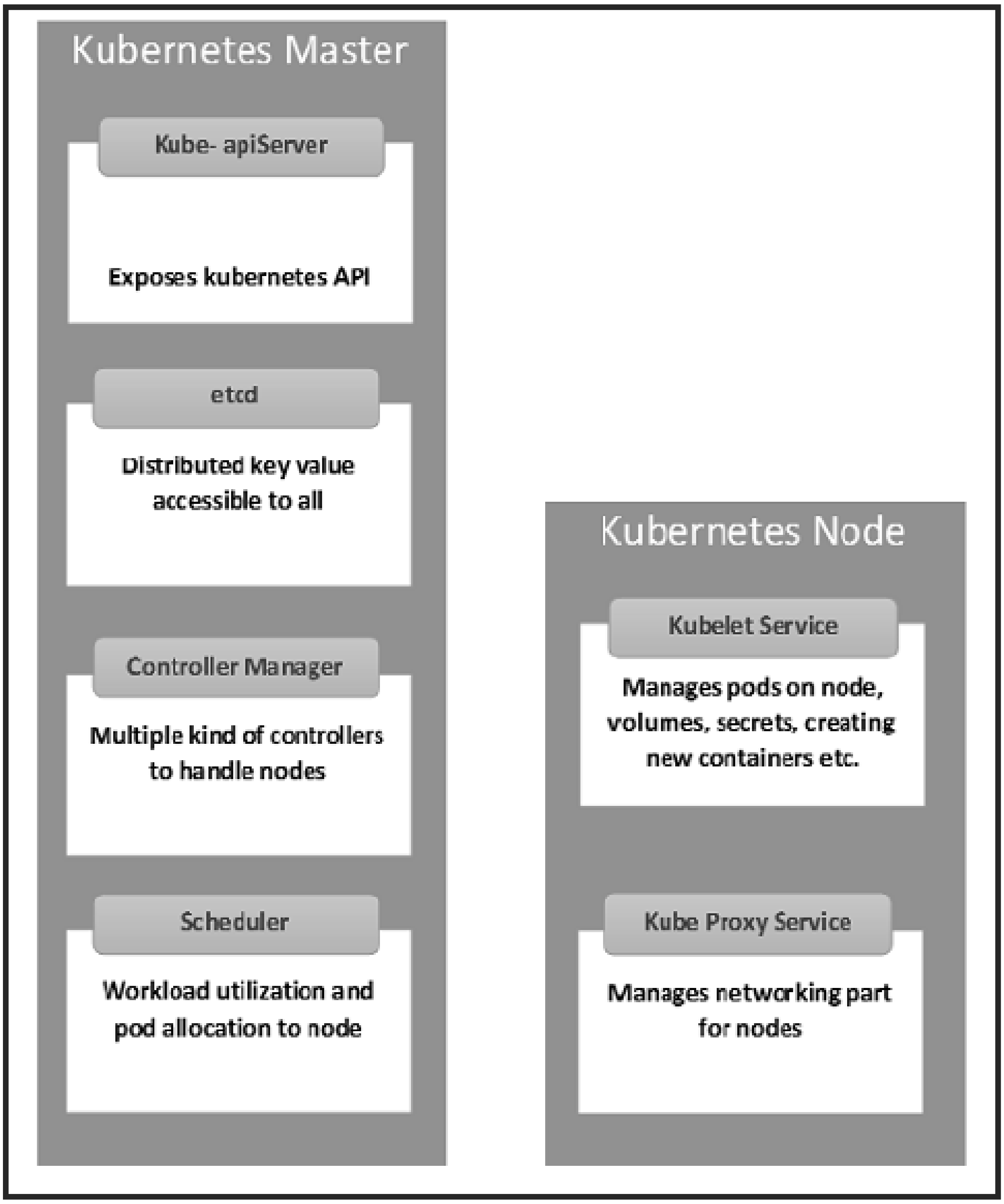
3. OpenShift - Architecture

OpenShift is a layered system wherein each layer is tightly bound with the other layer using Kubernetes and Docker cluster. The architecture of OpenShift is designed in such a way that it can support and manage Docker containers, which are hosted on top of all the layers using Kubernetes. Unlike the earlier version of OpenShift V2, the new version of OpenShift V3 supports containerized infrastructure. In this model, Docker helps in creation of lightweight Linux-based containers and Kubernetes supports the task of orchestrating and managing containers on multiple hosts.


OpenShift Container Platform
Container
Container
Persistent Data Storage
Orchestration/management layer
(Kubernetes)
Rest API
Kubernetes Scheduler
Data storage
CI/CD application and Built in layer
Existing Organization Toolset
Networking and other configuration layer
Enterprise Hosted Platform
Base Physical or cloud layer

Components of OpenShift

One of the key components of OpenShift architecture is to manage containerized infrastructure in Kubernetes. Kubernetes is responsible for Deployment and Management of infrastructure. In any Kubernetes cluster, we can have more than one master and multiple nodes, which ensures there is no point of failure in the setup.


Kubernetes Master
Kube- apiServer
Exposes kubernetes API
etcd
Distributed key value accessible to all
Controller Manager
Multiple kind of controllers to handle nodes
Scheduler
Workload utilization and pod allocation to node
Kubernetes Node
Kubelet Service
Manages pods on node, volumes, secrets, creating new containers etc.
Kube Proxy Service
Manages networking part for nodes

Kubernetes Master Machine Components

Etcd: It stores the configuration information, which can be used by each of the nodes in the cluster. It is a high availability key value store that can be distributed among multiple nodes. It should only be accessible by Kubernetes API server as it may have sensitive information. It is a distributed key value Store which is accessible to all.

API Server: Kubernetes is an API server which provides all the operation on cluster using the API. API server implements an interface which means different tools and libraries can readily communicate with it. A kubeconfig is a package along with the server side tools that can be used for communication. It exposes Kubernetes API".

Controller Manager: This component is responsible for most of the collectors that regulate the state of the cluster and perform a task. It can be considered as a daemon which runs in a non-terminating loop and is responsible for collecting and sending information to API server. It works towards getting the shared state of the cluster and then make changes to bring the current status of the server to a desired state. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. The controller manager runs different kind of controllers to handle nodes, endpoint, etc.

Scheduler: It is a key component of Kubernetes master. It is a service in master which is responsible for distributing the workload. It is responsible for tracking the utilization of working load on cluster nodes and then placing the workload on which resources are available and accepting the workload. In other words, this is the mechanism responsible for allocating pods to available nodes. The scheduler is responsible for workload utilization and allocating a pod to a new node.

Kubernetes Node Components

Following are the key components of the Node server, which are necessary to communicate with the Kubernetes master.

Docker: The first requirement of each node is Docker which helps in running the encapsulated application containers in a relatively isolated but lightweight operating environment.

Kubelet Service: This is a small service in each node, which is responsible for relaying information to and from the control plane service. It interacts with etcd store to read the configuration details and Wright values. This communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

Kubernetes Proxy Service: This is a proxy service which runs on each node and helps in making the services available to the external host. It helps in forwarding the request to correct containers. Kubernetes Proxy Service is capable of carrying out primitive load balancing. It makes sure that the networking environment is predictable and accessible but at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers health checkup, etc.

Integrated OpenShift Container Registry

OpenShift container registry is an inbuilt storage unit of Red Hat, which is used for storing Docker images. With the latest integrated version of OpenShift, it has come up with a user interface to view images in OpenShift internal storage. These registries are capable of holding images with specified tags, which are later used to build containers out of it.

Frequently Used Terms

Image: Kubernetes (Docker) images are the key building blocks of Containerized Infrastructure. As of now, Kubernetes only supports Docker images. Each container in a pod has its Docker image running inside it. When configuring a pod, the image property in the configuration file has the same syntax as the Docker command.

Project: They can be defined as the renamed version of the domain which was present in the earlier version of OpenShift V2.

Container: They are the ones which are created after the image is deployed on a Kubernetes cluster node.

Node: A node is a working machine in Kubernetes cluster, which is also known as minion for master. They are working units which can a physical, VM, or a cloud instance.

Pod: A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it. For example, keeping the database container and web server container inside the pod.

4. OpenShift - Environment Setup

In this chapter, we will learn about the environment setup of OpenShift.

System Requirement

In order to set up enterprise OpenShift, one needs to have an active Red Hat account. As OpenShift works on Kubernetes master and node architecture, we need to set up both of them on separate machines, wherein one machine acts as a master and other works on the node. In order to set up both, there are minimum system requirements.

Master Machine Configuration

Following are the minimum system requirements for master machine configuration.

- A base machine hosted either on physical, virtual, or on any of the cloud environment.
- At least Linux 7 with the required packages on that instance.
- 2 CPU core.
- At least 8 GB RAM.
- 30 GB of internal hard disk memory.

Node Machine Configuration

- Physical or virtual base image as given for the master machine.
- At least Linux 7 on the machine.
- Docker installed with not below than 1.6 version.
- 1 CPU core.
- 8 GB RAM.
- 15 GB hard disk for hosting images and 15 GB for storing images.

Step by Step Guide to OpenShift Setup

In the following description, we are going to set up OpenShift lab environment, which can be later extended to a bigger cluster. As OpenShift requires master and node setup, we would need at least two machines hosted on either cloud, physical, or virtual machines.

Step 1: First install Linux on both the machines, where the Linux 7 should be the least version. This can be done using the following commands if one has an active Red Hat subscription.

```
# subscription-manager repos --disable="*"
```

```
# subscription-manager repos --enable="rhel-7-server-rpms"
```

```
# subscription-manager repos --enable="rhel-7-server-extras-rpms"
```

```
 # subscription-manager repos --enable="rhel-7-server-optional-rpms"
```

```
# subscription-manager repos --enable="rhel-7-server-ose-3.0-rpms"
```

```
# yum install wget git net-tools bind-utils iptables-services bridge-utils
```

```
# yum install wget git net-tools bind-utils iptables-services bridge-utils
```

```
# yum install python-virtualenv
```

```
# yum install gcc
```

```
# yum install httpd-tools
# yum install docker
```

```
# yum update
```

Once we have all the above base packages installed in both of the machines, the next step would be to set up Docker on the respective machines.

Step 2: Configure Docker so that it should allow insecure communication on the local network only. For this, edit the Docker file inside /etc/sysconfig. If the file is not present then you need to create it manually.

```
# vi  /etc/sysconfig/docker
OPTIONS=--selinux-enabled --insecure-registry 192.168.122.0/24
```

After configuring the Docker on the master machine, we need to set up a password-less communication between both the machines. For this, we will use public and private key authentication.

Step 3: Generate keys on the master machine and then copy the id_rsa.pub key to the authorized key file of the node machine, which can be done using the following command.

```
# ssh-keygen
```

```
# ssh-copy-id -i .ssh/id_rsa.pub root@ose3-node.test.com
```

Once you have all of the above setup in place, next is to set up OpenShift version 3 on the master machine.

Step 4: From the master machine, run the following curl command.

```
 # sh <(curl -s https://install.openshift.com/ose)
```

The above command will put the setup in place for OSV3. The next step would be to configure OpenShift V3 on the machine.

If you cannot download from the Internet directly, then it could be downloaded from https://install.openshift.com/portable/oo-install-ose.tgz as a tar package from which the installer can run on the local master machine.

Once we have the setup ready, then we need to start with the actual configuration of OSV3 on the machines. This setup is very specific to test the environment for actual production, we have LDAP and other things in place.

Step 5: On the master machine, configure the following code located under /etc/openshift/master/master-config.yaml

```
# vi /etc/openshift/master/master-config.yaml
identityProviders:
- name: my_htpasswd_provider
challenge: true
login: true
provider:
apiVersion: v1
kind: HTPasswdPasswordIdentityProvider
file: /root/users.htpasswd
routingConfig:
 subdomain: testing.com
```

Next, create a standard user for default administration.

```
# htpasswd -c /root/users.htpasswd admin
```

Step 6: As OpenShift uses Docker registry for configuring images, we need to configure Docker registry. This is used for creating and storing the Docker images after build.

Create a directory on the OpenShift node machine using the following command.

```
# mkdir  /images
```

Next, login to the master machine using the default admin credentials, which gets created while setting up the registry.

```
# oc login
Username: system:admin
```

Switch to the default created project.

```
# oc project default
```

Step 7: Create a Docker Registry.

```
#echo
'{"kind":"ServiceAccount","apiVersion":"v1","metadata":{"name":"registry"}}' |
oc create -f -
```

Edit the user privileges.

```
#oc edit scc privileged

users:

- system:serviceaccount:openshift-infra:build-controller

- system:serviceaccount:default:registry
```

Create and edit the image registry.

```
#oadm registry --service-account=registry --
config=/etc/openshift/master/admin.kubeconfig  --
credentials=/etc/openshift/master/openshift-registry.kubeconfig --
images='registry.access.redhat.com/openshift3/ose-${component}:${version}' --
mount-host=/images
```

Step 8: Create a default routing.

By default, OpenShift uses OpenVswitch as software network. Use the following command to create a default routing. This is used for load balancing and proxy routing. The router is similar to the Docker registry and also runs in a registry.

```
# echo
'{"kind":"ServiceAccount","apiVersion":"v1","metadata":{"name":"router"}}' | oc
create -f -
```

Next, edit the privileges of the user.

```
#oc edit scc privileged
users:
 - system:serviceaccount:openshift-infra:build-controller
 - system:serviceaccount:default:registry
 - system:serviceaccount:default:router


#oadm router router-1 --replicas=1 --
credentials='/etc/openshift/master/openshift-router.kubeconfig' --
images='registry.access.redhat.com/openshift3/ose-${component}:${version}'
```

Step 9: Configure the DNS.

In order to handle URL request, OpenShift needs a working DNS environment. This DNS configuration is required to create a wild card, which is required to create DNS wild card that points to a router.

```
# yum install bind-utils bind
```

```
# systemctl start named
```

```
# systemctl enable named
```

```
vi /etc/named.conf
options {listen-on port 53 { 10.123.55.111; };
forwarders {
10.38.55.13;
;
};
zone "lab.com" IN {
 type master;
 file "/var/named/dynamic/test.com.zone";
 allow-update { none; };
};
```

Step 10: The final step would be to set up github server on OpenShift V3 master machine, which is optional. This can be done easily using the following sequence of commands.

```
#yum install curl openssh-server
```

```
#systemctl enable sshd
```

```
# systemctl start sshd
```

```
# firewall-cmd --permanent --add-service=http
```

```
# systemctl reload firewalld
```

```
#curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-
```

```
#yum install gitlab-ce
```

```
# gitlab-ctl reconfigure
```

Once the above setup is complete, you can verify by test and deploy applications, which we will know more about in the subsequent chapters.

5. OpenShift - Basic Concept

Before beginning with the actual setup and deployment of applications, we need to understand some basic terms and concepts used in OpenShift V3.

Containers and Images

Images

These are the basic building blocks of OpenShift, which are formed out of Docker images. In each pod on OpenShift, the cluster has its own images running inside it. When we configure a pod, we have a field which will get pooled from the registry. This configuration file will pull the image and deploy it on the cluster node.

```
apiVersion: v1
kind: pod
metadata:
    name: Tesing_for_Image_pull ----------->  Name of Pod
       spec:
containers:
- name: neo4j-server ------------------------> Name of the image
image: <Name of the Docker image>----------> Image to be pulled
imagePullPolicy: Always ------------->Image pull policy
command: [“echo”, “SUCCESS”] -------------------> Massage after image pull
```

In order to pull and create an image out of it, run the following command. OC is the client to communicate with OpenShift environment after login.

```
$ oc create –f Tesing_for_Image_pull
```

Container

This gets created when the Docker image gets deployed on the OpenShift cluster. While defining any configuration, we define the container section in the configuration file. One container can have multiple images running inside and all the containers running on cluster node are managed by OpenShift Kubernetes.

```
spec:
   containers:
   - name: py ------------------------> Name of the container
   image: python----------> Image going to get deployed on container
   command: [“python”, “SUCCESS”]
   restartPocliy: Never --------> Restart policy of container
```

Following are the specifications for defining a container having multiple images running inside it.

```
apiVersion: v1
kind: Pod
metadata:
    name: Tomcat
spec:
   containers:
   - name: Tomcat
   image: tomcat: 8.0
   ports:
   - containerPort: 7500
     imagePullPolicy: Always
      -name: Database
      Image: mongoDB
      Ports:
      - containerPort: 7501
imagePullPolicy: Always
```

In the above configuration, we have defined a multi-container pod with two images of Tomcat and MongoDB inside it.

Pods and Services

Pods

Pod can be defined as a collection of container and its storage inside a node of OpenShift (Kubernetes) cluster. In general, we have two types of pod starting from a single container pod to multi-container pod.

Single Container Pod: These can be easily created with OC command or by a basic configuration yml file.

```
$ oc run <name of pod> --image=<name of the image from registry>
```

Create it with a simple yaml file as follows.

```
apiVersion: v1
kind: Pod
metadata:
      name: apache
spec:
      containers:
        - name: apache
        image: apache: 8.0
        ports:
             - containerPort: 7500
imagePullPolicy: Always
```

Once the above file is created, it will generate a pod with the following command.

```
$ oc create –f apache.yml
```

Multi-Container Pod: Multi-container pods are those in which we have more than one container running inside it. They are created using yaml files as follows.

```
apiVersion: v1
kind: Pod
metadata:
     name: Tomcat
spec:
      containers:
        - name: Tomcat
        image: tomcat: 8.0
        ports:
             -  containerPort: 7500
imagePullPolicy: Always
  -name: Database
   Image: mongoDB
   Ports:
         - containerPort: 7501
imagePullPolicy: Always
```

After creating these files, we can simply use the same method as above to create a container.

Service: As we have a set of containers running inside a pod, in the same way we have a service that can be defined as a logical set of pods. It's an abstracted layer on top of the pod, which provides a single IP and DNS name through which pods can be accessed. Service helps in managing the load balancing configuration and to scale the pod very easily. In OpenShift, a service is a REST object whose deification can be posted to apiService on OpenShift master to create a new instance.

```
apiVersion: v1
kind: Service
metadata:
      name: Tutorial_point_service
spec:
   ports:
   - port: 8080
      targetPort: 31999
```

Builds and Streams

Builds

In OpenShift, build is a process of transforming images into containers. It is the processing which converts the source code to an image. This build process works on pre-defined strategy of building source code to image.

The build processes multiple strategies and sources.

Build Strategies

- **Source to Image**: This is basically a tool, which helps in building reproducible images. These images are always in a ready stage to run using the Docker run command.

- **Docker Build**: This is the process in which the images are built using Docker file by running simple Docker build command.

- **Custom Build**: These are the builds which are used for creating base Docker images.

Build Sources

Git: This source is used when the git repository is used for building images. The Dockerfile is optional. The configurations from the source code looks like the following.

```
source:
  type: "Git"
  git:
    uri: "https://github.com/vipin/testing.git"
    ref: "master"
  contextDir: "app/dir"
  dockerfile: "FROM openshift/ruby-22-centos7\nUSER example"
```

Dockerfile: The Dockerfile is used as an input in the configuration file.

```
source:
  type: "Dockerfile"
  dockerfile: "FROM ubuntu: latest
  RUN yum install -y httpd"
```

Image Streams: Image streams are created after pulling the images. The advantage of an image stream is that it looks for updates on the new version of an image. This is used to compare any number of Docker formatted container images identified by tags.

Image streams can automatically perform an action when a new image is created. All the builds and deployments can watch for image action and perform an action accordingly. Following is how we define a build a stream.

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  generation: 1
  labels:
    app: ruby-sample-build
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample
  tags:
  - items:
    - created: 2016-01-29T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-apache-sample
      generation: 1
      image: vklnld908.int.clsa.com/vipin/test
    tag: latest
```

Routes and Templates

Routes

In OpenShift, routing is a method of exposing the service to the external world by creating and configuring externally reachable hostname. Routes and endpoints are used to expose the service to the external world, from where the user can use the name connectivity (DNS) to access defined application.

In OpenShift, routes are created by using routers which are deployed by OpenShift admin on the cluster. Routers are used to bind HTTP (80) and https (443) ports to external applications.

Following are the different kinds of protocol supported by routes:

- HTTP
- HTTPS
- TSL and web socket

When configuring the service, selectors are used to configure the service and find the endpoint using that service. Following is an example of how we create a service and the routing for that service by using an appropriate protocol.

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "Openshift-Rservice"
  },
  "spec": {
    "selector": {
      "name":"RService-openshift"
    },
    "ports": [
      {
        "protocol": "TCP",
        "port": 8888,
        "targetPort": 8080
      }
    ]
  }
}
```

Next, run the following command and the service is created.

```
$ oc create -f ~/training/content/Openshift-Rservice.json
```

This is how the service looks like after creation.

```
$ oc describe service Openshift-Rservice
Name:                    Openshift-Rservice
Labels:                    <none>
Selector:                 name= RService-openshift
Type:                       ClusterIP
IP:                            172.30.42.80
Port:                        <unnamed>       8080/TCP
Endpoints:               <none>
Session Affinity:        None
No events.
```

Create a routing for service using the following code.

```
{
  "kind": "Route",
  "apiVersion": "v1",
  "metadata": {
    "name": "Openshift-service-route"
  },
  "spec": {
    "host": "hello-openshift.cloudapps.example.com",
    "to": {
      "kind": "Service",
      "name": "OpenShift-route-service"
    },
    "tls": {
      "termination": "edge"
    }
  }
}
```

When OC command is used to create a route, a new instance of route resource is created.

Templates

Templates are defined as a standard object in OpenShift which can be used multiple times. It is parameterized with a list of placeholders which are used to create multiple objects. This can be used to create anything, starting from a pod to networking, for which users have authorization to create. A list of objects can be created, if the template from CLI or GUI interface in the image is uploaded to the project directory.

```
apiVersion: v1
kind: Template
metadata:
  name: <Name of template>
  annotations:
    description: <Description of Tag>
    iconClass: "icon-redis"
    tags: <Tages of image>
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: <Object Specification>
  spec:
    containers:
      image: <Image Name>
      name: master
      ports:
      - containerPort: <Container port number>
        protocol: <Protocol>
labels:
  redis: <Communication Type>
```