5

OpenShift Deployment

In this chapter, we will navigate through a deployment process for Red Hat OpenShift. A successful implementation case will be only possible when you understand the needs from architecture, understand the branding technology applied to the underlying infrastructure, and the workloads you are supposed to have on top of OpenShift.

As you have seen in the first chapter there are a lot of options to this universe of multi-cluster. So, what are the best options for you? How can you choose? You can get a lot of confusion if you start to prepare the requisites of the deployment without properly preparing yourself for that. That said, we reinforce the importance to be aligned with the architecture chosen to ensure the expected deployment will succeed.

Changes during the execution time and non-planned gaps can cause unexpected deployed solutions and can create a messed-up platform that cannot work as expected and is full of patched holes. Making a comparison with the real world, it is like a ship that cannot be smoothly loaded with containers because it is always being repaired and, as such, is not ready to perform long trips or even to depart.

So, how to prepare this ship to navigate with low maintenance? We will guide you through reviewing some important questions and getting the answers that are the best scenario to start, maintain, scale, update the cluster and get a robust platform that can host a lot of applications on top of it.

Considering the explanation of the first part of this book, you have a lot of options to deploy OpenShift clusters, let's start making some questions in which you will build up the options to deploy the right choice for you. Keep in mind, there are no right or wrong answers, but it will have the answers that make you decide which is the best for you.

This chapter covers:

* Requirements
* OpenShift Installation Prerequisites
* Prepare Installers
* Installation
* What's Next

Let’s get started then!

Requirements

This is a practical chapter in which you will deploy an OpenShift cluster using what we consider as the most complex deployment procedure: the UPI/Agnostic installation. As we already covered in this book, there are many different types of installations and supported providers, and it is almost impossible to cover every combination of them – neither is our intention, there is plenty of documentation, tutorials, and great references over the internet that will guide you through all types of installations.

That said, we understand that the most added-value deployment procedure we can bring to you is the UPI/Agnostic one – when you make it, for sure you will be able to understand and execute easily the other types of installations. The reason for this is simple, with the UPI/Agnostic installation you are responsible to provide all pre-requisites an OpenShift requires, while with the IPI deployment the installer itself will provision the pre-requisites for you automatically within the underlying provider.

OpenShift Checklist Opt-In

Have you seen the checklist we brought to you in this book in Chapter 2, Architecture Overview and Definitions, and Chapter 3, Multi-tenant Considerations? If you are reading this chapter to implement an OpenShift cluster in an enterprise and you didn’t read those chapters yet, we strongly recommend you go back there and check the important aspects you need to think and consider first, before deploying a cluster.

Lab requisites

To follow the labs in this chapter, you will need the following:

* A hypervisor or cloud provider in which you can spin up the instances. You can use Baremetal servers also if you have them available.
* This is the minimum required for the VMs:
  + 1 temporary server for Bootstrap node.
  + 3 servers for Masters with 2 vCPU, 8 GB RAM, and 50 GB of disk (minimum)
  + 2 servers for Workers with 2 vCPU, 8 GB RAM, and 50 GB of disk (minimum)
  + 1 server for the Bastion with 2 vCPU, 4 GB RAM, and 20 GB of disk (minimum)

If you don´t have enough resources available in your environment you can also use a 3-nodes cluster, in which masters and workers are co-located in the same nodes.

Important Note

The requirements listed above are valid only for a lab. Refer to Chapter 2, Architecture Overview and Definitions, to have a reliable sizing for an enterprise installation.

OpenShift Installation Pre-requisites

Before starting our journey through OpenShift deployment you must observe several pre-requisites. First, we will explore the options according to the decision that you previously choose in the opt-in form.

As explained in Chapter 1, OpenShift has 3 installation modes: IPI (Installer Provisioned Infrastructure), UPI (User Provisioned Infrastructure), and Agnostic (aka Bare Metal Installer). It is very important to consider that no option will work well for every case, although the best option is the one that best fits into the architecture designed for you previously.

This chapter is very focused on deployment and all stuff related to spanning a cluster by yourself, so keep that in mind to make your own lab and enjoy the tips and materials on our GitHub which will be useful and a Swiss knife for you.

Based on the following opt-in form you are going to see which installation methods you have, according to the provider chosen (at the time this book was written).

|  |  |  |  |
| --- | --- | --- | --- |
| Provider | IPI | UPI | Agnostic |
| Vmware Esxi 6.5+ | Recommended | Possible |  |
| Nutanix AHV | Not Available | Recommended |  |
| Red Hat Openstack | Recommended | Possible |  |
| Microsoft Hyper-V | Not Available | Recommended |  |
| Oracle Virtualization | Not Available | Recommended |  |
| Red Hat Virtualization | Recommended | Possible |  |
| Amazon Web Services (AWS) | Recommended | Possible |  |
| Microsoft Azure | Recommended | Possible |  |
| Google Cloud Platform | Recommended | Possible |  |
| IBM Cloud | Not Available | Recommended |  |
| IBM Z w/VM - LinuONE | Not Available | Recommended |  |
| IBM Power Systems | Not Available | Not Available | Recommended |
| Bare Metal servers | Possible | Not Available | Recommended |

Regarding the terms in the previous table, we classified some of the available options for each infrastructure provider to give you an overview of the current possibilities (by the time when this book was written). When we say Recommended, we are not only giving our perspective but we are trying to say this is a common and best choice for that scenario, as well, Possible indicates you a valid option, but you will have some penalties, like losing some great automation features the product brings out-of-the-box – for that reason we classified them as Possible, but not as the best choice. Not available is self-explanatory and does not need any further connotation.

As the prerequisites must be different according to the installation method, we prepared a matrix that helps you start the preparation of the underlying infrastructure to start the cluster deployment.

|  |  |  |
| --- | --- | --- |
| Installation Method | Prerequisites Needed | Common Requisites |
| UPI Installation | * DHCP * DNS * Web Server * Load Balancer * Firewall Rules | * Pull secret * Openshift Installer * Openshift Client * SSH Key |
| Agnostic Installation | * All UPI Prereqs + PXE Server (optional, you can boot from ISO instead) |
| IPI Installation | * Provider properly configured (Account permissions and limits) |

User-Provisioned Infrastructure / Agnostic Installer

Any OpenShift installation must have a valid Red Hat Subscription, the OpenShift Installer binaries, a pullsecret file, a public ssh key, and the resources available according to each provider.

In this section, we will guide you through a feasible and reliable cluster installation whatever provider you have decided. We are going to set up the pre-requisites needed as well using practical examples – feel free to use those configurations as many as you need.

So, let's start with the pre-requisites systems that are not part of the OpenShift itself, but are indispensable to ensure that everything will work fine during the deployment process.

DNS

In the following lines, we will discuss the Domain Name System requirements to provision an OpenShift cluster. For demonstrations purposes, we will bring in this chapter a minimum configuration to make everything work, for in-depth settings check the references we left in the last chapter of this book.

For our lab, we will use the BIND tool running in a Red Hat Enterprise Linux 8 VM, however, you can use any other DNS server on top of Windows or your preferred Linux distribution. We will refer to this Linux VM from now on with the name bastion, which is a kind of convention with Red Hat architectures. If you want to follow strictly the instructions in this chapter, we recommend you to use a fresh install of Red Hat Enterprise Linux 8, using the minimum install.

An OpenShift cluster requires a dedicated subdomain. To facilitate your understanding, we will use a hypothetical Hybrid Cloud Company that uses hybridcloud.com as your main public domain. The complete subdomain for the OpenShift cluster will be as follows:

ocp.hybridcloud.com

To install BIND, run the following commands on your bastion VM:

$ sudo yum install bind bind-utils -y

$ sudo systemctl enable --now named

$ sudo firewall-cmd --permanent --add-port=53/tcp

$ sudo firewall-cmd --permanent --add-port=53/udp

$ sudo firewall-cmd --reload

Now we are going to configure the DNS server to be used with the OpenShift installation and applications. Perform the following steps to accomplish that:

1. Create subdomain zone at named.conf file:

$ sudo cat << EOF > /etc/named.conf

zone "ocp.hybridcloud.com" IN {

type master;

file "/var/named/ocp.hybridcloud.com.db";

allow-query { any; };

allow-transfer { none; };

allow-update { none; };

};

zone "1.168.192.in-addr.arpa" IN {

type master;

file "/var/named/ocp.hybridcloud.com.zone";

allow-update { none; };

};

EOF

1. Create a forward zone file at /var/named/ocp.hybridcloud.com.db

$ sudo cat <<EOF > /var/named/ocp.hybridcloud.com

;[1] Begin Common Header Definition

$TTL 86400

@ IN SOA bastion.ocp.hybridcloud.com. root.ocp.hybridcloud.com. (

202201010001 ;Serial

21600 ;Refresh

3600 ;Retry

604800 ;Expire

86400 ;Minimum TTL

)

;End Common Header Definition

;Name Server Information [2]

IN NS bastion.ocp.hybridcloud.com.

;IP address of Name Server [3]

bastion IN A 192.168.1.200

;api internal and external purposes [4]

api IN A 192.168.1.5

api-int IN A 192.168.1.5

;wildcard application [5]

\*.apps IN A 192.168.1.6

;bootstrap node to start cluster install only [6]

bootstrap IN A 192.168.1.90

;master nodes [7]

master1 IN A 192.168.1.91

master2 IN A 192.168.1.92

master3 IN A 192.168.1.93

;worker nodes [8]

worker1 IN A 192.168.1.101

worker2 IN A 192.168.1.102

EOF

[1] Common DNS Zone Header

[2] The nameserver will be the own BASTION server

[3] The IP Address from nameserver (Bastion IP)

[4] These records are mandatory need to point to the VIP that will be used for the OpenShift API functions

[5] Wildcard VIP record used for the applications that run on OpenShift.

[6] Boostrap Node Ip record used only for the cluster installation and can be removed after it.

[7] Master Nodes Ip records, where the control plane objects will be hosted.

[8] Worker Nodes Ip records, where the workloads will run. If you go for a 3-nodes cluster, disregard the worker hosts

1. Create a reverse zone file at /var/named/1.168.192.arpa-in.addr

$ sudo cat <<EOF > /var/named/1.168.192.in-addr.arpa

$TTL 1W @ IN SOA bastion.ocp.hybridcloud.com.root (

2019070700 ; serial

3H ; refresh (3 hours)

30M ; retry (30 minutes)

2W ; expiry (2 weeks)

1W ) ; minimum (1 week)

5.1.168.192.in-addr.arpa. IN PTR api.ocp.hybridcloud.com.;   
5.1.168.192.in-addr.arpa. IN PTR api-int.ocp.hybridcloud.com.; 90.1.168.192.in-addr.arpa. IN PTR bootstrap.ocp.hybridcloud.com.; 91.1.168.192.in-addr.arpa. IN PTR master1.ocp.hybridcloud.com.; 92.1.168.192.in-addr.arpa. IN PTR master2.ocp.hybridcloud.com.; 93.1.168.192.in-addr.arpa. IN PTR master3.ocp.hybridcloud.com.; 101.1.168.192.in-addr.arpa. IN PTR worker1.ocp. hybridcloud.com.; 102.1.168.192.in-addr.arpa. IN PTR worker2.ocp. hybridcloud.com.;

EOF

Important Notes

1. Do NOT create a reverse zone record for the applications wildcard VIP, that will lead to wrong DNS resolution.

2. If you for a 3-nodes cluster, disregard the worker A and PTR records.

1. Restart named service

sudo systemctl restart named

1. Validate the DNS to ensure that all DNS records are set up appropriately using dig commands:

Validate OpenShift API:

dig +short @192.168.1.200 api.ocp.hybridcloud.com

dig +short @192.168.1.200 api-int.ocp.hybridcloud.com

For the BIND samples we described in this section, the output MUST be:

192.168.1.5

192.168.1.5

Validate the applications wildcard:

dig +short @192.1681.1.200 joedoe.apps.ocp.hybridcloud.com

dig +short @192.1681.1.200 whatever.apps.ocp.hybridcloud.com

All results MUST point to ingress applications wildcard VIP:

192.168.1.6

192.168.1.6

Validate the nodes:

dig +short @192.1681.1.200 boostrap.ocp.hybridcloud.com

dig +short @192.1681.1.200 master1.ocp.hybridcloud.com

dig +short @192.1681.1.200 master2.ocp.hybridcloud.com

dig +short @192.1681.1.200 master3.ocp.hybridcloud.com

dig +short @192.1681.1.200 worker1.ocp.hybridcloud.com

dig +short @192.1681.1.200 worker2.ocp.hybridcloud.com

The answer must be the following:

192.168.1.90

192.168.1.91

192.168.1.92

192.168.1.93

192.168.1.101

192.168.1.102

Finally, let’s validate the reverse records:

dig +short @192.168.1.200 -x 192.168.1.5

dig +short @192.168.1.200 -x 192.168.1.90

dig +short @192.168.1.200 -x 192.168.1.91

dig +short @192.168.1.200 -x 192.168.1.92

dig +short @192.168.1.200 -x 192.168.1.93

dig +short @192.168.1.200 -x 192.168.1.94

dig +short @192.168.1.200 -x 192.168.1.95

The results look similar to:

api-int.ocp.hybridcloud.com.

api.ocp.hybridcloud.com.

bootstrap.ocp.hybridcloud.com.

master1.ocp.hybridcloud.com.

master2.ocp.hybridcloud.com.

master3.ocp.hybridcloud.com.

worker1.ocp.hybridcloud.com.

worker2.ocp.hybridcloud.com.

Well done! If your DNS server is properly resolving names you gave a big step in preparing the prerequisites. Now let’s move to another important piece of an OpenShift installation using the UPI method: the DHCP.

DHCP

The Dynamic Host Configuration Protocol is used to provide IP addresses to the OpenShift nodes. In UPI or Agnostic installation nodes, the IP needs to be set using static configuration on DHCP (fixed-address parameter).

Make sure that the IP and Hostname set in the DNS and DHCP for the nodes match between them – each IP and hostname in the DNS and DHCP need to be equal. In this subsection of prerequisites, we are focusing on creating a simple DHCP setup for your use during your study and laboratory use. As previously stated DHCP will be configured to provide static IPs, under 192.168.1.x subnet, so, this configuration uses the MAC Address of each node’s Ethernet interfaces.

1. Installing the DHCP on your Bastion VM:

$ sudo yum install dhcp -y

$ sudo systemctl enable --now dhcpd

1. Configuring the dhcpd.conf file according to the hostnames and IPs used with the DNS:

cat <<EOF > /etc/dhcp/dhcpd.conf

# DHCP Server Configuration file.

#[1]

ddns-update-style interim;

ignore client-updates;

authoritative;

allow booting;

allow bootp;

allow unknown-clients;

default-lease-time 3600;

default-lease-time 900;

max-lease-time 7200;

#[2]

subnet 192.168.1.0 netmask 255.255.255.0 {

option routers 192.168.1.254;

option domain-name-servers 192.168.1.200;

option ntp-servers 192.168.1.200;

next-server 192.168.1.200; #[2.1]

#filename "pxelinux.0";#[2.2]  
#[3]

group {

host bootstrap {

hardware ethernet 50:6b:8d:aa:aa:aa;

fixed-address 192.168.1.90;

option host-name "bootstrap.ocp.hybridcloud.com";

allow booting;

}

host master1 {

hardware ethernet 50:6b:8d:bb:bb:bb;

fixed-address 192.168.1.91;

option host-name "master1.ocp.hybridcloud.com";

allow booting;

}

host master2 {

hardware ethernet 50:6b:8d:cc:cc:cc;

fixed-address 192.168.1.92 ;

option host-name "master2.ocp.hybridcloud.com";

allow booting;

}

host master3 {

hardware ethernet 50:6b:8d:dd:dd:dd;

fixed-address 192.168.1.93 ;

option host-name "master3.ocp.hybridcloud.com";

allow booting;

}

host worker1 {

hardware ethernet 50:6b:8d:11:11:11;

fixed-address 192.168.1.101;

option host-name "worker1.ocp.hybridcloud.com";

allow booting;

}

host worker2 {

hardware ethernet 50:6b:8d:22:22:22;

fixed-address 192.168.1.102;

option host-name "worker2.ocp.hybridcloud.com";

allow booting;

}

}

}

EOF

[1] Common settings to define DHCP as authoritative into that subnet and times of IP lease

[2] Scope subnet definition

[2.1] and [2.2] must be defined when using a PXE Server, helpful for Baremetal installations. In this lab, we are going to use VMs and, as such, that will not be used – therefore leave it commented ("#" char in the begging of the line).

[3] A group with all nodes to lease IP addresses. If you go for a 3-nodes cluster, disregard the worker hosts.

Important Note

After you create the VMs in your hypervisor, update the dhcpd.conf accordingly with the MAC addresses you get from the network interfaces - otherwise, no IP address will be given into this subnet.

Webserver

A web server is used to serve the OS Image to install nodes and also provide the ignition files (ignition files are manifest files encoded on base64). To our scenario, we will install and configure an Apache web server, which is a very simple way to provide all the necessary stuff for the cluster installation to run adequately.

Check the quick list to accomplish this task:

1. Install httpd server:

$ sudo yum install httpd –y

* Configure /etc/httpd/conf.d/httpd.conf to change "Listen" directive:

$ sudo sed –i 's/80/81/g' /etc/httpd/conf.d/httpd.conf

* Apply SELINUX to change default httpd port.

$ sudo semanage port -a -t http\_port\_t -p tcp 81

* Create a rule on firewalld to allow port 81.

$ sudo firewall-cmd --add-port 81/tcp

$ sudo firewall-cmd --add-port 81/tcp --permanent

$ sudo firewall-cmd --reload

* Create a directory for the OS image and ignition files and a file to test the connectivity.

$ sudo mkdir –p /var/www/html/images

$ sudo mkdir –p /var/www/html/ignition

$ sudo touch /var/www/html/images/imageFileToTest.txt

$ sudo touch /var/www/html/ignition/ignitionFileToTest.txt

* Set permission and owner to files.

$ sudo chown –R apache. /var/www/html/

$ sudo chmod 744 –R /var/www/html/

* Start and Enable Apache Webserver

$ sudo systemctl enable --now httpd

* Test connectivity using curl

$ curl –O http://192.168.1.200/images/imageFileToTest.txt

$ curl –O http://192.168.1.200/ignition/ignitionFileToTest.txt

If you can see the file downloaded to your current folder it means you have Apache web server properly configured to serve OpenShift installation.

Load Balancer

A load balancer is another important element in an OpenShift cluster architecture. It is responsible to balance the connection to the pool member in a set of nodes. For performance and resilience reasons, it is recommended, for production environments, that you use dedicated load balancing appliance hardware.

For our lab, we will be using an HAProxy in our Bastion VM to perform the Load Balancing function, which is a powerful, lightweight, and easy-to-use software load balancing.

That being said, before we start to configure it, it is important you understand the basics of load balancing methods and the best practices that fit best with the OpenShift platform.

Imagine that load balancer is like a drop of water filling equally distributed into some cups, each drop must fall into a cup, then the next drop must fill the next cup, and so on. Nonetheless, the cups are periodically dumped avoiding wastes or overload.

So, there are some ways to perform the tasks, these ways are known as balancing methods, see the following table for an explanation about the scenarios which OpenShift will lay out.

|  |  |  |
| --- | --- | --- |
| Method | Overview | OSI Model |
| Round Robin | Distribute the requests among all available servers on a cyclical basis. It is commonly the default setup for load balancers. | Layer 4 (Transport) |
| Least Connection | New connection to least member in the pool or members with least number of active connections. | Layer 4 (Transport), only |
| Fastest | Least number of current outstanding sessions. | Layer 4 (Transport) and Layer 7 (Application) |

A typical Load Balancer configuration is composed of 4 pairs of front and back-end configurations that will balance the different types of requests and give a reliable fault tolerance to the platform.

The first pool members are the master nodes, they are a group of 3 members and it should work with Least Connection with sourced address setup. This setting assures that during internal API calls to the load balancer, the request will be handled from the same node that started the call requisition and, as such, gives the proper callback and asynchronous functioning.

You can find the HAProxy front and back-end configurations in the sample below (frontend openshift-api-server and backend openshift-api-server).

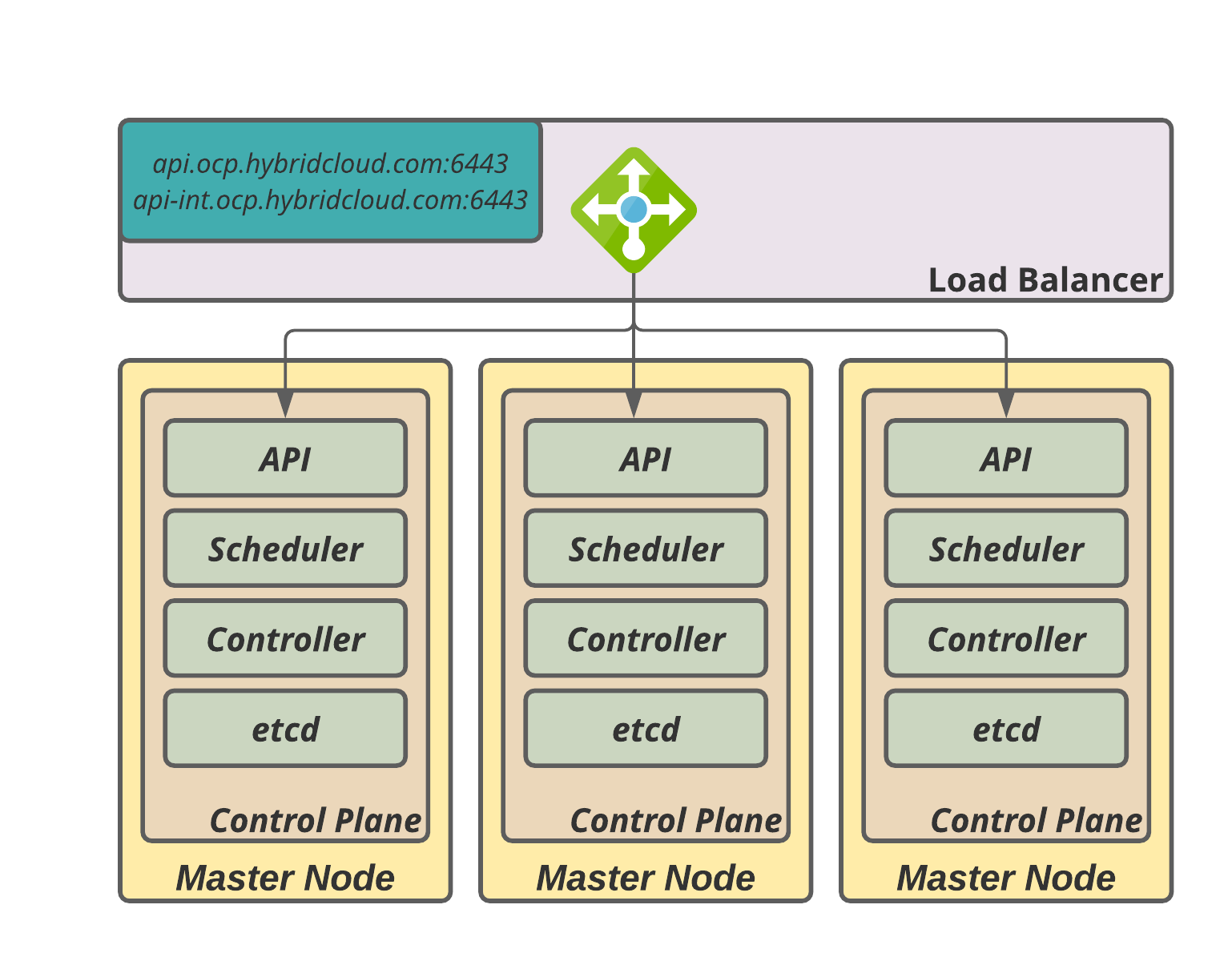


Fig. 6.1 – Master Node Load Balancer

The second group of load balancer configurations is also used with the master nodes for the OpenShift machine-config-server API, which will be explained better during chapter X of this book. See the HAProxy configuration for the machine-config-server API in the frontend machine-config-server and backend machine-config-server.

The third and fourth groups of load balancing should be a pool of at least 2 nodes [worker nodes] that will be responsible for the traffic routing from outside to application distributed on the worker nodes of the cluster (one for HTTP and another for HTTPS).

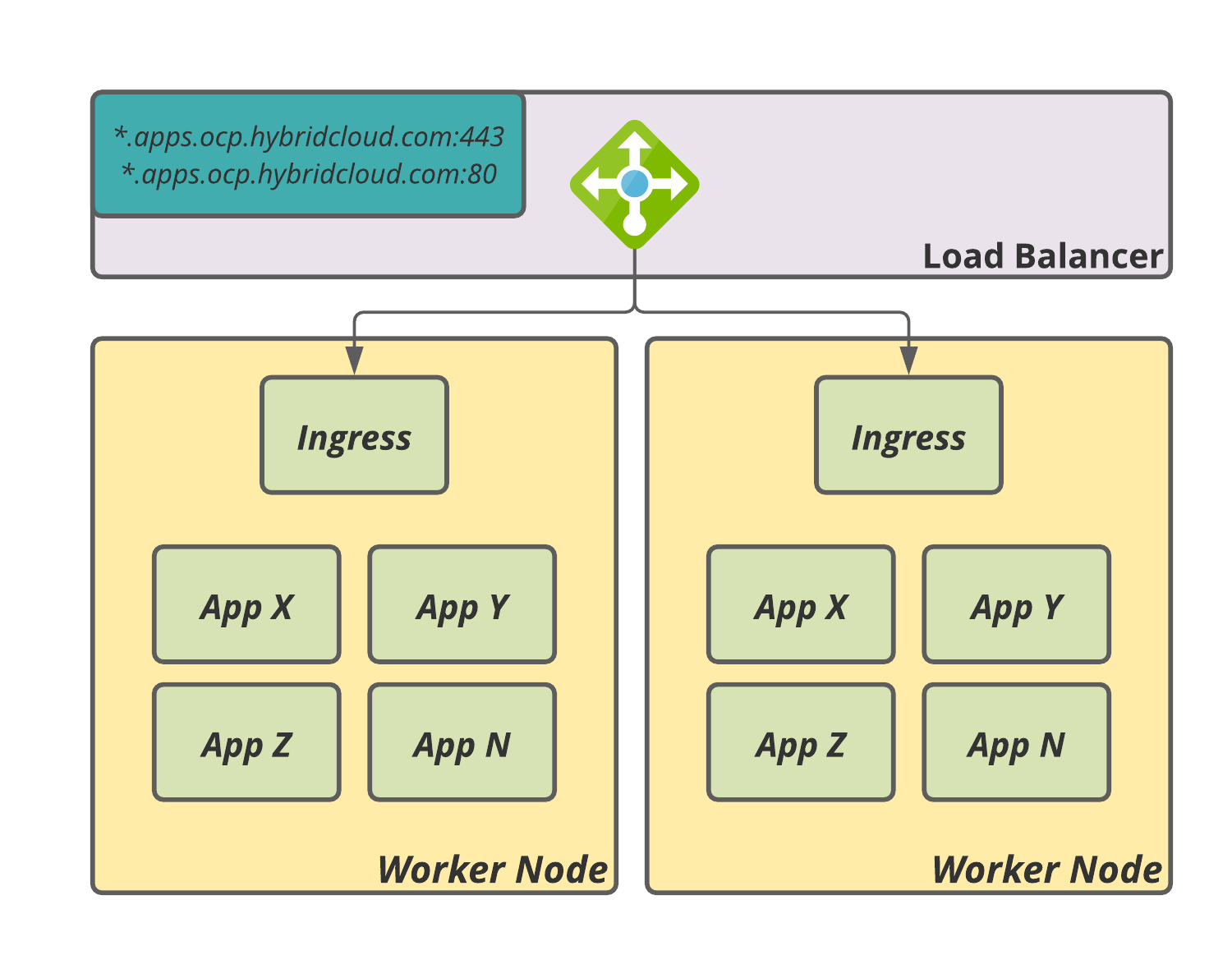


Fig. 6.2 – Ingress Load Balancer

Regularly OpenShift recommends the least connections with source addresses running on the transport layer (layer 4) which commonly gives a good performance to routing applications. However, when using certificates signed by a public CA in the load balancer, instead of the OpenShift Ingress Controller, you must eventually set up this pool to work on the application layer (layer 7).

Important Note

We strongly recommend disabling the SSL inspection at the load balancer/firewall layers to avoid Digital Certificate issues and malfunctioning of the OpenShift cluster. It occurs because the enterprise load balancer/firewall solution when using SSL inspection mode is enabled, decrypts every TCP payload, and re-incapsulates them with a new SSL header. OpenShift interprets it as a certificate error, causing incorrect source/destiny TCP packets and hanging up to TLS termination.

In a nutshell the complete haproxy.cfg will be like the following:

$ sudo cat <<EOF > /etc/haproxy/haproxy.cfg

# Global settings

global

maxconn 20000

log /dev/log local0 info

chroot /var/lib/haproxy

pidfile /var/run/haproxy.pid

user haproxy

group haproxy

daemon

# turn on stats unix socket

stats socket /var/lib/haproxy/stats

defaults

mode http

log global

option httplog

option dontlognull

option forwardfor except 127.0.0.0/8

option redispatch

retries 3

timeout http-request 10s

timeout queue 1m

timeout connect 10s

timeout client 300s

timeout server 300s

timeout http-keep-alive 10s

timeout check 10s

maxconn 20000

# Enable haproxy status endpoint

listen stats

bind :9000

mode http

stats enable

stats uri /

# OpenShift API (port 6443)

frontend openshift-api-server

bind \*:6443

default\_backend openshift-api-server

mode tcp

option tcplog

backend openshift-api-server

balance source

mode tcp

# bootstrap line below can be removed after the cluster is deployed

server bootstrap 192.168.1.90:6443 check

server master1 192.168.1.91:6443 check

server master2 192.168.1.92:6443 check

server master3 192.168.1.93:6443 check

# machine-config-server API (port 22623)

frontend machine-config-server

bind \*:22623

default\_backend machine-config-server

mode tcp

option tcplog

backend machine-config-server

balance source

mode tcp

# bootstrap line below can be removed after the cluster is deployed

server bootstrap 192.168.1.90:22623 check

server master1 192.168.1.91:22623 check

server master2 192.168.1.92:22623 check

server master3 192.168.1.93:22623 check

# Applications HTTP (port 80)

frontend ingress-http

bind \*:80

default\_backend ingress-http

mode tcp

option tcplog

backend ingress-http

balance source

mode tcp

server worker1 192.168.1.101:80 check # [1]

server worker2 192.168.1.102:80 check # [1]

# Applications HTTPS (port 443)

frontend ingress-https

bind \*:443

default\_backend ingress-https

mode tcp

option tcplog

backend ingress-https

balance source

mode tcp

server worker0 192.168.1.101:443 check # [1]

server worker1 192,168.1.102:443 check # [1]

EOF

[1] If you go for a 3-nodes cluster you should point to the master nodes here also.

After that, apply the haproxy configuration by starting and enabling the service.

$ sudo systemctl enable --now haproxy

After it starts, your load balancer can receive incoming connections and give the redirect to one member of the pool.

Note

As soon as the installation of the OpenShift control plane is finished, you will receive a message that is safe to remove bootstrap from load balancing, then you can open the haproxy.cfg file and comment or remove the lines that are referencing the bootstrap server and restart the haproxy server to apply the configuration.

Firewall

As the cluster has a lot of components involved, it is critical to think about security between Red Hat OpenShift and all other systems it integrates to/from. Unless you are working with a disconnected environment (refer to Further reading of this chapter for more details about it), you will need to grant access to ports 80 and 443 from some URLs. These URLs are needed to download required container images and others. Therefore, whitelist the following URLs in your network firewall:

|  |  |
| --- | --- |
| URL | Required for |
| registry.redhat.io | Core container images |
| quay.io | Core container images |
| \*.quay.io | Core container images |
| sso.redhat.com | Red Hat SSO authentication URL |
| \*.openshiftapps.com | Red Hat Enterprise Linux CoreOS (RHCOS) images |
| cert-api.access.redhat.com | Cluster Telemetry and Insights |
| api.access.redhat.com | Cluster Telemetry and Insights |
| infogw.api.openshift.com | Cluster Telemetry and Insights |
| console.redhat.com/api/ingress | Cluster Telemetry and Insights |
| mirror.openshift.com | Access mirrored installation content and images and release image signatures |
| storage.googleapis.com/openshift-release | Source of release image signatures |
| \*.apps.<cluster\_name>.<base\_domain> | Default cluster routes to an ingress wildcard. It depends on the cluster and domain name chosen, as the purpose of this book, we are using \*.apps.ocp.hybridcloud.com |
| quay-registry.s3.amazonaws.com | Quay image content in AWS |
| api.openshift.com | Cluster token and to check if updates are available for the cluster. |
| art-rhcos-ci.s3.amazonaws.com | Repository to download Red Hat Enterprise Linux CoreOS (RHCOS) images. |
| console.redhat.com/openshift | Cluster token |
| registry.access.redhat.com | Used by odo CLI tool |

Reference

Refer to this link for the latest set of URLs: https://docs.openshift.com/container-platform/4.9/installing/install\_config/configuring-firewall.html

PXE Server

A PXE Server is a component to easily allows the boot process to look for the installation files. Throughout PXE configuration, you can create a simple GRUB (Grand Unified Bootloader) menu that works exactly like an operating system installer with all kernel parameters desired.

We will deploy some packages to install the PXE server, create directories to store the configuration files, and develop a simple and useful boot-start menu.  
Now, ssh to your bastion server and:

Install syslinux packages:

yum install –y syslinux-tftpboot

yum install –y syslinux-nonlinux

yum install –y syslinux

yum install –y tftp-server

Create directories:

/var/lib/tftpboot/networkboot/coreOS

/var/lib/tftpboot/pxelinux.cfg

Copy required PXE Server files:

cp –r /usr/share/syslinux /var/lib/tftpboot

Copy Red Hat CoreOS image files:

├── networkboot

│ └── coreOS

│ ├── rhcos-4.x.y-x86\_64-live-initramfs.x86\_64.img

│ ├── rhcos-4.x.y-x86\_64-live-kernel-x86\_64

│ └── rhcos-4.x.y-x86\_64-live-rootfs.x86\_64.img

Finally, create the BootLoader Menu to assist you on installation

$ sudo cat <<EOF > /etc/haproxy/haproxy.cfg

UI vesamenu.c32

MENU COLOR sel 4 #ffffff std

MENU COLOR title 0 #ffffff

TIMEOUT 120

PROMPT 0

MENU TITLE OPENSHIFT 4.X AGNOSTIC PXE MENU

LABEL BOOTSTRAP NODE

KERNEL networkboot/coreOS/rhcos-4.x.y-x86\_64-live-kernel-x86\_64 # [1]

APPEND initrd=networkboot/coreOS/rhcos-4.x.y-x86\_64-live-initramfs.x86\_64.img,networkboot/coreOS/rhcos-4.x.y-x86\_64-live-rootfs.x86\_64.img coreos.inst.install\_dev=/dev/sda coreos.inst.ignition\_url=http://192.168.1.200:81/ignition/bootstrap.ign # [1]

LABEL MASTER NODE

KERNEL networkboot/coreOS/rhcos-4.x.y-x86\_64-live-kernel-x86\_64 # [1]

APPEND initrd=networkboot/coreOS/rhcos-4.x.y-x86\_64-live-initramfs.x86\_64.img,networkboot/coreOS/rhcos-4.x.y-x86\_64-live-rootfs.x86\_64.img coreos.inst.install\_dev=/dev/sdacoreos.inst.ignition\_url=http://192.168.1.200:81/ignition/master.ign # [1]

LABEL WORKER NODE

KERNEL networkboot/coreOS/rhcos-4.x.y-x86\_64-live-kernel-x86\_64 # [1]

APPEND initrd=networkboot/coreOS/rhcos-4.x.y-x86\_64-live-initramfs.x86\_64.img,networkboot/coreOS/rhcos-4.x.y-x86\_64-live-rootfs.x86\_64.img coreos.inst.install\_dev=/dev/sdacoreos.inst.ignition\_url=http://192.168.1.200:81/ignition/worker.ign # [1]

EOF

[1] Replace to proper minor version according to image files downloaded.

Now that we have all the pre-requisites components correctly set, we can start the installation using the UPI or Agnostic installation method! So, go ahead and start your engines!

Installer-Provisioned Infrastructure

Even we tried to bring you a smoothly diving showing how to create all the pre-requisites systems and servers, perhaps it may look like an exhaustive process to get all the prerequisites done. Otherwise, it is important to emphasize that it can be exhaustive when prepare during studies all alone, but in huge enterprises, it is very common that those infrastructures commonly are already working and needs only some settings to reach the necessary state.

You must be sweating a lot after preparing all the previous steps for UPI Installer. By now, the good news is that the IPI installer is much easier than everything you already read until now. You probably compared all the things needed on the table Preparation Stuff Table, so to get accomplished the task using IPI, you should have only your cloud credentials, ensure the object limits of your cloud provider is enough for the minimum required from OpenShift, choose the size of cloud instances that best fit your needs, create the install-config.yaml file properly, and run Openshift-install binary to spawn your cluster.

This easy process is due to the high level of automation that OpenShift has behind the hood, which uses the cloud’s APIs to create all the prerequisites for you, according to the parameters you set in the install-config.yaml. Obviously, there are some changes from cloud to cloud, in the following few lines we demonstrate 2 excerpts that basically change in install-config.yaml when preparing your file for AWS, Azure, and Google Cloud Platform.

AWS install-config sample:

apiVersion: v1

baseDomain: hybridcloud.com

credentialsMode: Mint

controlPlane:

hyperthreading: Enabled

name: master

platform:

aws:

zones:

- us-west-2a

- us-west-2b

rootVolume:

iops: 4000

size: 500

type: io1

type: m5.xlarge

replicas: 3

compute:

- hyperthreading: Enabled

name: worker

platform:

aws:

rootVolume:

iops: 2000

size: 500

type: io1

type: c5.4xlarge

zones:

- us-west-2c

replicas: 3

metadata:

name: test-cluster

networking:

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

machineNetwork:

- cidr: 10.0.0.0/16

networkType: OpenShiftSDN

serviceNetwork:

- 172.30.0.0/16

platform:

aws:

region: us-west-2

userTags:

adminContact: jdoe

costCenter: 7536

amiID: ami-96c6f8f7

serviceEndpoints:

- name: ec2

url: <https://vpce-id.ec2.us-west-2.vpce.amazonaws.com>

fips: false

sshKey: ssh-ed25519 AAAA...

pullSecret: '{"auths": ...}'

Azure install-config sample:

apiVersion: v1

baseDomain: hybridcloud.com

controlPlane:

hyperthreading: Enabled

name: master

platform:

azure:

osDisk:

diskSizeGB: 1024

type: Standard\_D8s\_v3

replicas: 3

compute:

- hyperthreading: Enabled

name: worker

platform:

azure:

type: Standard\_D2s\_v3

osDisk: diskSizeGB: 512

zones:

- "1"

- "2"

- "3"

replicas: 5

metadata:

name: test-cluster

networking:

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

machineNetwork:

- cidr: 10.0.0.0/16

networkType: OpenShiftSDN

serviceNetwork:

- 172.30.0.0/16

platform:

azure:

BaseDomainResourceGroupName: resource\_group

region: centralus

resourceGroupName: existing\_resource\_group

outboundType: Loadbalancer

cloudName: AzurePublicCloud

pullSecret: '{"auths": ...}'

Google Cloud Platform install-config sample:

apiVersion: v1

baseDomain: hybridcloud.com

controlPlane:

hyperthreading: Enabled

name: master

platform:

gcp:

type: n2-standard-4

zones:

- us-central1-a

- us-central1-c

osDisk:

diskType: pd-ssd

diskSizeGB: 1024

encryptionKey:

kmsKey:

name: worker-key

keyRing: test-machine-keys

location: global

projectID: project-id

replicas: 3

compute:

- hyperthreading: Enabled

name: worker

platform:

gcp:

type: n2-standard-4

zones:

- us-central1-a

- us-central1-c

osDisk:

diskType: pd-standard

diskSizeGB: 128

encryptionKey:

kmsKey:

name: worker-key

keyRing: test-machine-keys

location: global

projectID: project-id

replicas: 3

metadata:

name: test-cluster

networking:

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

machineNetwork:

- cidr: 10.0.0.0/16

networkType: OpenShiftSDN

serviceNetwork:

- 172.30.0.0/16

platform:

gcp:

projectID: openshift-production

region: us-central1

pullSecret: '{"auths": ...}'

fips: false

sshKey: ssh-ed25519 AAAA...

Well done! Now you have also the proper install-config.yaml files to use according to your cloud provider. Proceed with the installation to start your Openshift using your preferred installation method.

Prepare for the installation

As you have seen in the previous sections, the pre-requisites are very important and any error on that could be an Achilles' heel for the OpenShift cluster installation and functioning. That happens because failure in preparing the pre-requisites will cause errors during the cluster deployment that are not always easy to troubleshoot and find the root cause. That said, we would like to reassure the importance of preparing and validating the pre-requisites correctly before starting the cluster deployment.

To start the installation using the UPI method, you will need:

* An SSH key pair
* A pull secret for the cluster, which you can generate by accessing the <https://console.redhat.com/openshift/install> with a valid user subscription.
* OpenShift Installer binary.
* OpenShift command-line tools.
* Installation Configuration File (install-config.yaml).

In the following lines, we will detail all of those steps.

SSH Key-Pair

Starting from OpenShift version 4, Red Hat begins to use Red Hat CoreOS as the main operating system due to the container and immutable nature. Red Hat CoreOS needs some ignition files to provision the OS based on that configurations. This process leads to a secure and reliable manner to provision OpenShift nodes, allowing a standard and zero-touch provisioning process.

The SSH is used to access the nodes directly and only throughout a pair of keys assigned for user coreos (it is not possible to access the nodes using a simple user/password). That said, it is very important to maintain a copy of the ssh key-pair used during the cluster deployment – in case of an incident with your cluster, this is the only way to directly access the nodes to collect logs and try to troubleshoot them. Also, SSH key pair will make a part of ignition files and the public key pair are distributed across all nodes of the cluster.

We are going to use our Bastion to create an SSH key-pair, by using the following command:

$ ssh-keygen -t ecdsa -N '' -f ~/.ssh/clusterOCP\_key

You will use a public key in the next steps, for example, clusterOCP\_key.pub.

Important Note

Never expose or share the SSH private key, any malicious person with the private key could get root access to the nodes and, with some knowledge, escalate privileges as OpenShift cluster-admin user.

Pull Secret

A pull secret is a file that contains a collection of users and passwords encoded in Base64 used to authenticate with required image registries, like quay.io and registry.redhat.io. You need to have a valid user at console.redhat.com to download or copy the pull secret.

To do so, follow the 3-step mode below:

1. Access <https://console.redhat.com/openshift/create> and access Downloads in the side menu.

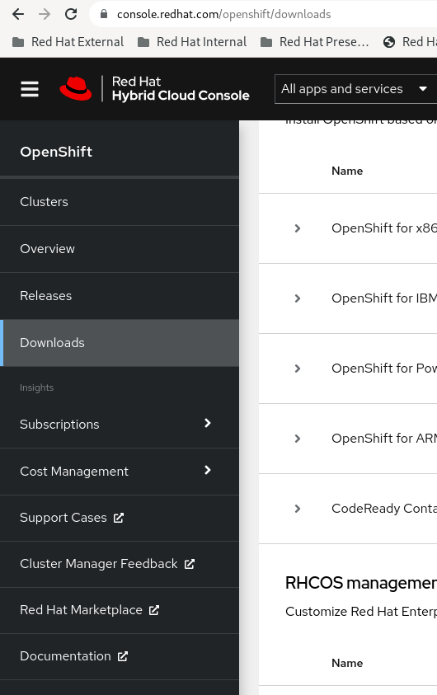


Fig. 5.1 – Pull secret – Downloads menu

1. Scroll down to the section Tokens and click on the Copy or Download button to get the pull secret.

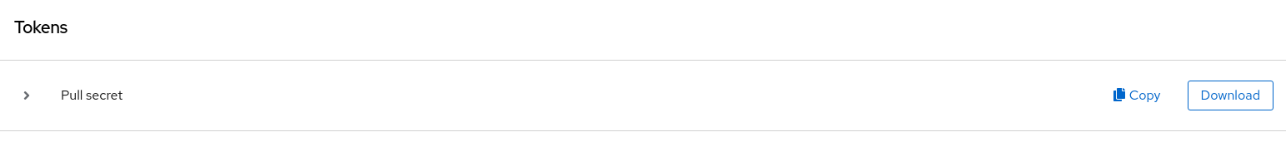


Fig. 5.2 – Download or copy the pull secret

On this page, you will also find the command-line, developer tools, and the installer binaries to download.

OpenShift Installer binary

To install the cluster, you should download the installer binary which can be downloaded on Red Hat Cloud Console as mentioned previously, or you can browse directly the public repository: <https://mirror.openshift.com/pub/openshift-v4/x86_64/clients/ocp/latest/>

OpenShift Command-Line Tools

Like the installer binary, you can download the command-line tools under the same public repository mentioned previously or thru Red Hat Cloud Console where you downloaded the pull secret.

Command-line tools are composed of "oc" and "kubectl" CLI which you will use to manage and run commands on the cluster, as soon as it has been spawned.

Installation Configuration File (install-config.yaml)

The last step before starting the installation is to create a manifest file that is called install-config.yaml. Essentially, this file is composed of the control plane, worker, and network definitions, along with some metadata like the pull secret and the public ssh key.

Based on our previous configuration, below is a sample of the install-config file which can be used with the installation.

$ mkdir ~/ocp

$ cat <<EOF > ~/ocp/install-config.yaml

apiVersion: v1

baseDomain: hybridcloud.com # [1]

compute:

- hyperthreading: Enabled

name: worker

replicas: 2 # [2]

controlPlane:

hyperthreading: Enabled

name: master

replicas: 3 # [3]

metadata:

name: ocp # [4]

networking:

clusterNetwork:

- cidr: 10.148.0.0/14 # [5]

hostPrefix: 23

networkType: OpenShiftSDN # [6]

serviceNetwork:

- 10.153.0.0/16 # [7]

platform:

none: {} # [8]

fips: false

pullSecret: '<YOUR-PULL-SECRET>' # [9]

sshKey: '<YOUR-SSH-KEY>' # [10]

EOF

[1] The base domain for your cluster. Needs to match with the one you configured in your DNS in the previous section.

[2] The initial number of worker nodes you will be deploying with the cluster. If you go for a 3-nodes cluster, this value must be zero.

[3] The initial number of master nodes you will be deploying with the cluster. For a highly available cluster use 3 replicas always.

[4] Cluster name. It also needs to match with what you specified in your DNS.

[5] Block of IP Addresses that will be used internally for pods within OpenShift SDN. We explored the concept of this in Chapter 2, Architecture Overview and Definitions, of this book.

[6] Type of SDN used. Valid values are OpenShiftSDN or OVNKubernetes.

[7] Block of IP Addresses that will be used internally for network services within OpenShift SDN. We explored the concept of this in Chapter 2, Architecture Overview and Definitions, of this book.

[8] Specific data about the underlying platform. That will be different depending on the platform your cluster will be hosted (AWS, Azure, GCP, Vmware, etc.). For agnostic installation use none.

Note

You can use also the openshift-installer binary to generate a sample install-config.yaml file for the provider you are going to work with. Use this command for that: ./openshift-install create install-config

After this file is created you can proceed to the installation steps in the following section.

Installation

Now, some further steps should be performed before deploying the cluster. The first step in the installation is to create the Manifest files. We strongly recommend you create a backup of the install-config.yaml file before running the command below – this command removes the original file and you will need to create it again from scratch if you need to retry the installation.

$ ./openshift-install create manifests --dir= ~/ocp/

Open the file ~/ocp/manifests/cluster-scheduler-02-config.yml in the editor of your preference and change the mastersSchedulable parameter to false if you go for a regular cluster or true if you decided to provision a 3-nodes cluster.

You should now generate the Ignition files by running the following command:

$ ./openshift-install create ignition-configs --dir=~/ocp/

After the previous command you might have 3 new ignition files: bootstrap.ign, master.ign and worker.ign.

Copy these 3 files to the HTTP Server you prepared in the previous section:

$ sudo cp ~/ocp/\*.ign /var/www/html/ignition

Finally, you can proceed with the nodes provisioning.

Phase 1 – Provisioning Servers

At this point, you need to provision the servers. That will vary depending on the underlying infrastructure but in general, the process for virtualized environments (Vmware vSphere, RHV, etc.) is:

1. Import the Red Hat CoreOS template to the hypervisor.
2. Clone it and configure the VM parameters according to the provider.

On the other hand, this process for Baremetal or Agnostic is performed either by booting using the Red Hat CoreOS ISO or using a PXE.

In our lab, we are going to boot using the Red Hat CoreOS ISO. Follow the steps below to do it:

1. Download the ISO file from console.redhat.com, as mentioned previously or directly through this link: <https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/latest/latest/rhcos-live.x86_64.iso>
2. In the Bastion VM run the following command to get the SHA512 digest of the ignition files (this will be used after the boot from ISO to validate the authenticity of the file):

$ sha512sum ~/ocp/bootstrap.ign

$ sha512sum ~/ocp/master.ign

$ sha512sum ~/ocp/worker.ign

Example of the output:

a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf0116e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b installation\_directory/bootstrap.ign

1. Boot using the ISO image and don’t specify any options until you see a shell prompt.
2. Run the following coreos-installer command to start the ignition process and, consequently, the OS provisioning:

$ sudo coreos-installer install --ignition-url= http://192.168.1.200/ignition/bootstrap.ign /dev/sda --ignition-hash=SHA512- a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf0116e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b

1. Repeat the same process above for each server, always respecting the following format for the coreos-intaller command:

$ sudo coreos-installer install --ignition-url=http://192.168.1.200/<node\_type>.ign <device> --ignition-hash=SHA512-<digest>

Where <node\_type> will be bootstrap.ign, master.ign or worker.ign, <device> is the disk to be used to install the OS (e.g.: /dev/sda) and <digest> is the result of the sha512sum command mentioned above.

After booting the Bootstrap and Master nodes using the procedure listed above, you can go to the next step to monitor the progress of the installation.

Phase 2 – Bootstrap and Control Plane

At this phase, bootstrap will download the container images to provision the control plane components. As soon as the containers in each master node are running, the control plane components will start configuring themselves until the etcd cluster, API, and controllers from OpenShift are synchronized.

Run the command below from the Bastion VM to monitor the progress of the Bootstrap and Control Plan deployment:

./openshift-install wait-for bootstrap-complete --dir= /home/user/ocp/ --log-level=debug

Right after the command is triggered, you will see some log messages on the console, similar to the sample below:

INFO Waiting up to 30m0s for the Kubernetes API at https://api.ocp.hybridcloud.com:6443... INFO API v1.22.1 up INFO Waiting up to 30m0s for bootstrapping to complete... INFO It is now safe to remove the bootstrap resources

After that, you must remove the bootstrap from the load balancer and restart the haproxy service.

Note

Remember, the bootstrap server is a one-shot use only, therefore, you can destroy the bootstrap server completely from the infrastructure provider because it will not be used anymore, even if something goes wrong during the cluster installation.

Phase 3 – Check for Certificates to Sign – For UPI and AGNOSTIC install only

When bootstrap finishes its process, the Red Hat OpenShift Container Platform creates a series of certificate signing requests (CSR) for each of the nodes. During our planning, we attempt to provision two worker nodes, so we must accept the certificates to join the worker into the cluster.

We need to use the oc client to approve the certificates. To do so, run the following command to export the kubeadmin credentials and get access to the cluster:

$ export KUBECONFIG=~/ocp/auth/kubeconfig

A simple command can list the pending certificates and the approving them until no pending certificates are showing.

$ oc get csr | grep –i Pending

NAME AGE REQUESTOR CONDITION

csr-bfd72 5m26s system:node:worker0.ocp.hybridcloud.com Pending

csr-c57lv 5m26s system:node:worker1.ocp.hybridcloud.com Pending

...

Then, to approve the certificates run the following command:

$ oc get csr -o name | xargs oc adm certificate approve

certificatesigningrequest.certificates.k8s.io/csr-bfd72 approved

certificatesigningrequest.certificates.k8s.io/csr-c57lv approved

To confirm that everything worked fine, run the below commands until all nodes remain Ready

$ oc get nodes

NAME STATUS ROLES AGE VERSION

ocp-7m9wx-master-0 Ready master 77d v1.21.1+9807387

ocp-7m9wx-master-1 Ready master 77d v1.21.1+9807387

ocp-7m9wx-master-2 Ready master 77d v1.21.1+9807387

ocp-7m9wx-worker-jds5s Ready worker 77d v1.21.1+9807387

ocp-7m9wx-worker-kfr4d Ready worker 77d v1.21.1+9807387

Phase 4 – Finishing the installation

We are almost reaching the end of our UPI/Agnostic installation! Now we must check the cluster operators to ensure that all of them are available.

Using the following command you will be able to monitor the cluster operators deployment progress.

./openshift-install wait-for install-complete --dir= /home/user/ocp/ --log-level=debug

INFO Waiting up to 30m0s for the cluster to initialize...

When it finishes you will get the kubeadmin password to finally get access to your OpenShift cluster.

Important Note

The kubeadmin is a temporary user with cluster-admin privileges. It is highly recommended that you remove the kubeadmin user as soon as you set up a new identity provider and give proper cluster-admin privileges to the cluster administrators.

Now, you can access the OpenShift Console GUI using your preferred browser. To do so, browse to <https://console-openshift-console.apps.ocp.hybridcloud.com> and insert kubeadmin credentials:

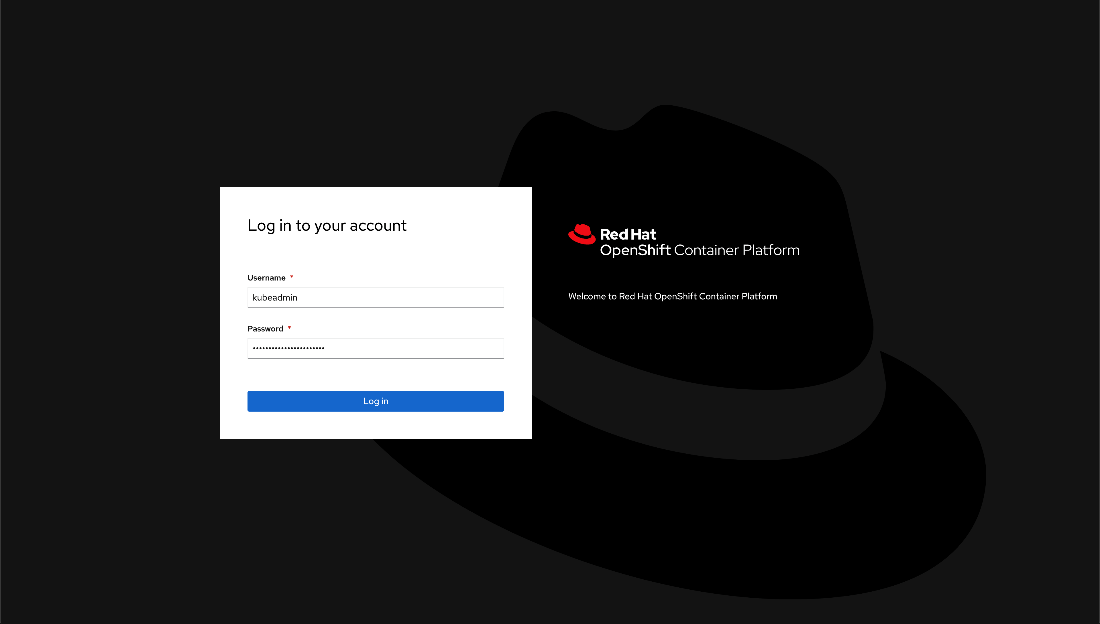


Fig. 5.3 – Access the console UI

... and enjoy:

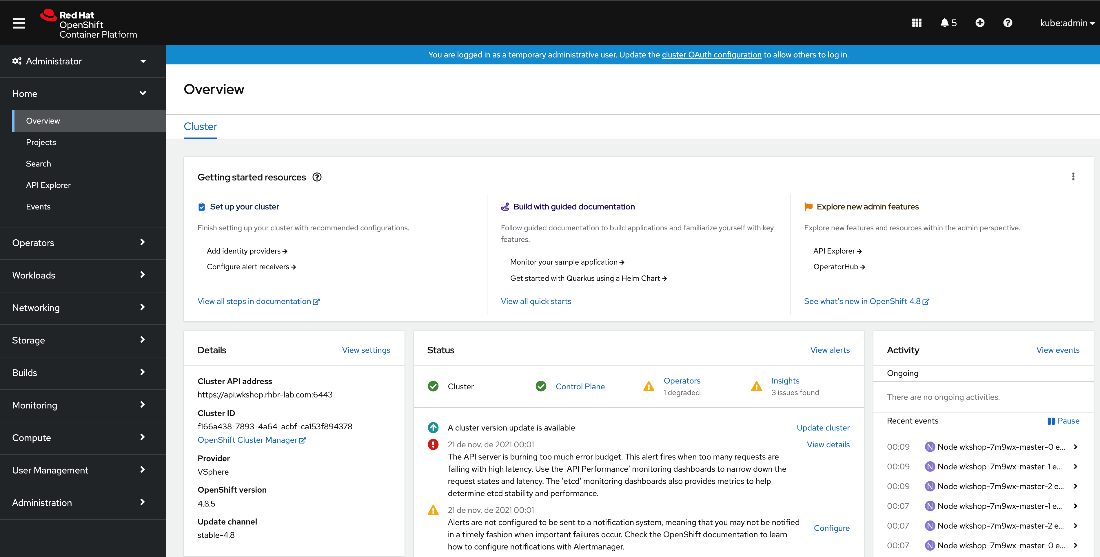


Fig. 5.4 – Console UI

Congratulations! You successfully deployed an OpenShift cluster! Celebrate your great success, but remember that your journey is just starting!

What's Next?

Now OpenShift is minimally functional, meaning that you have a control plane that can schedule pods, handle API calls and Controllers, and an etcd cluster, which is the key/value storage of objects of the cluster. You also have some worker nodes fully functional that already can host some workloads.

But that's not all, now the activities that demand configurations begin – also known as day 2 or post-installation activities. In the next few chapters, we will look at ingress configurations, networks, registry configurations, views for local volumes, persistent volumes, and so on.

We will talk also about taints and tolerations, security, and best practices. All it takes for you to go from zero to the hero and start acting on more complex implementations.

Summary

We have seen in this chapter some options to install and configure your OpenShift container platform solution. From public cloud to on-premises, you had been navigated inside UPI/IPI and Agnostic kind of implementation method.

Also, now you know the public clouds that have fully supported implementation and documentation to start your cluster.

By now, you are invited to deep dive into turning your OpenShift cluster stronger, reliable, and at the most secure as possible. We encourage you to continue to the next chapter and learn with us to go deeper and beyond.

Further reading

If you want to look at more information related to the concepts we covered in this chapter, check the following references:

* The installation process for disconnected installations: <https://docs.openshift.com/container-platform/latest/installing/installing-mirroring-installation-images.html>
* OpenShift Container Platform 4.x Tested Integrations by Red Hat and partners: https://access.redhat.com/articles/4128421