

1. Environment
Overview

2. Installation and
Verification

3. Deploying
Container Native
Storage

4. Application
Management Basics

5. Project Template,
Quota, and Limits

6. Using External
Authentication
Providers (LDAP)

7. Infrastructure
Management Basics

8. Container-native
Storage
Management

Infrastructure Management, Metrics and Logging

7

In this lab you will explore various aspects of managing cluster infrastructure. This includes the extending the OpenShift cluster and installation of the Logging and Metrics components, all automated by the installer. It also includes some maintenance of nodes, as well as manipulating the multitenant network.

Extending the cluster

Log in as cluster administrator:

```
oc login -u system:admin
```

Verify your cluster currently consists of 5 nodes, of which one is marked as *Not Schedulable* (the Master).

```
oc get nodes
```

You should see:

9. Skipping modules

NAME	STATUS
infra.internal.aws.testdrive.openshift.com	Ready
master.internal.aws.testdrive.openshift.com	Ready, SchedulingDisal
node01.internal.aws.testdrive.openshift.com	Ready
node02.internal.aws.testdrive.openshift.com	Ready
node03.internal.aws.testdrive.openshift.com	Ready

Use the label specified as the default node selector for pods in the inventory file `/etc/ansible/hosts` to verify that your cluster currently has 3 application nodes:

```
oc get nodes -l region=apps
```

You should see:

NAME	STATUS
node01.internal.aws.testdrive.openshift.com	Ready
node02.internal.aws.testdrive.openshift.com	Ready
node03.internal.aws.testdrive.openshift.com	Ready

Extending the cluster is easy. Simply add a new set of hosts to an Ansible group called `new_nodes` in the `openshift-ansible` installer's inventory. Then, run the `scaleup` playbook.

Configure the Installer

Your environment already has 3 additional nodes, but you have not used them so far. They are already configured in the inventory file but commented out.

Edit the file `/etc/ansible/hosts/` in an editor of your choice with `sudo` privileges, e.g.

```
sudo vim /etc/ansible/hosts
```

There two places where `#scaleup_` appears:

1. in front of the `new_nodes` in the `OSEv3:children` group
 - a. removing the comment tells the installer to include that subgroup.
2. in the actual `new_nodes` group, for each node's definition

When finished, your inventory file should look like the following:

/etc/ansible/hosts

```
[OSEv3:children]
```

```
masters
nodes
etcd
new_nodes
```

```
...
```

```
[new_nodes]
```

```
node04.internal.aws.testdrive.openshift.com openshift_node
```

```
node05.internal.aws.testdrive.openshift.com openshift_node
node06.internal.aws.testdrive.openshift.com openshift_node

...
```

Now that these hosts are properly defined (uncommented), you can use Ansible to verify that they are, in fact, online:

```
ansible new_nodes -m ping
```

You will see:

```
node05.internal.aws.testdrive.openshift.com | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
node04.internal.aws.testdrive.openshift.com | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
node06.internal.aws.testdrive.openshift.com | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Much like when you installed OpenShift originally, these new hosts have all of the [prerequisites](#) already taken care of.

Run the Playbook to Extend the Cluster

To extend your cluster run the following playbook:

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/byo
```

The playbook takes 1-2 minutes to complete. When done verify the amount of application nodes known to OpenShift has been increased from 3 to 6:

```
oc get nodes -l region=apps
```

You will see:

NAME	STATUS
node01.internal.aws.testdrive.openshift.com	Ready
node02.internal.aws.testdrive.openshift.com	Ready
node03.internal.aws.testdrive.openshift.com	Ready
node04.internal.aws.testdrive.openshift.com	Ready
node05.internal.aws.testdrive.openshift.com	Ready
node06.internal.aws.testdrive.openshift.com	Ready



When deploying a highly-available multi-master OpenShift environment, it is also possible to add new master nodes. There is a similar playbook to run. For more information on multi-master and HA setups, please refer to the

OpenShift Metrics

Metrics in OpenShift refers to the continuous collection of performance and utilization data of pods in the cluster. It allows for centralized monitoring in the OpenShift UI and automated horizontal scaling of pods based on utilization.

The metrics implementation is based on [Hawkular](#), a metrics collection system running on OpenShift persisting data in a Cassandra database.

In your environment metrics is not yet deployed. Configuration is done by customizing the Ansible inventory file `/etc/ansible/hosts` and deployment is facilitated by running a specific playbook that is part of the `openshift-ansible` installer. You could have chosen to install the metrics solution when the cluster was initially installed.

Configure the Installer

Using your favourite editor, open the `/etc/ansible/hosts` file with `sudo` privileges, e.g.

```
sudo vim /etc/ansible/hosts
```

In the `[OSEv3:vars]` section, you will find some directives that begin with `openshift_metrics`. Several are commented out, with the prefix `#metrics_` (similar to the comments used for extending the cluster).

`openshift_metrics_install_metrics=false` tells the installer **not** to install the metrics solution when it runs. Make sure that you delete that

line. Then, remove all of the comments, so that the section in your file looks like the following:

/etc/ansible/hosts

```
...  
[OSEv3:vars]  
...  
openshift_metrics_install_metrics=true  
openshift_metrics_cassandra_storage_type=pv  
openshift_metrics_cassandra_pvc_size=10Gi  
openshift_metrics_hawkular_hostname=metrics.apps.193183262  
...
```

Install Metrics

There is a specific playbook included with the installer that will handle metrics. It can be run like so:

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/byo
```

This will deploy the metric collection and visualization stack on OpenShift. All resources will be stood up in the **openshift-infra Project**. As part of the deployment, persistent storage will automatically be used for storing the metrics information.



In this environment you will end up using CNS as the persistent storage for the Cassandra database. While functional, at this time CNS is not a fully supported back-end storage solution for Cassandra. Full support for the metrics solution on CNS should come in a future release.

Once the installation playbook has completed, you can then verify that the metrics components are running in the **openshift-infra** Project:

```
oc login -u system:admin -n openshift-infra
oc get pods -o wide
```

You will see something like:

NAME	READY	STATUS	RESTARTS	AGE
hawkular-cassandra-1-6gv0d	1/1	Running	0	3m
hawkular-metrics-zkp0h	1/1	Running	0	3m
heapster-r2l2v	1/1	Running	0	3m



In this lab environment it can take up to 2-3 minutes after the metrics playbook finishes for the metrics stack to finish initialization and for all pods to reach the *Ready* state.

In the **node** column you will notice that the **Pods** for metrics are distributed throughout the environment. As we discussed **nodeSelectors** in the app management exercises, it may be desirable to force the metrics

components to run on specific **Nodes** in the cluster that user workload cannot run on. The configuration options for metrics support this, and those options look like the following:

```
openshift_metrics_hawkular_nodeselector={"region":"infra"}
openshift_metrics_cassandra_nodeselector={"region":"infra"}
openshift_metrics_heapster_nodeselector={"region":"infra"}
```

Explore the Metrics UI

If you don't have it open, return to the OpenShift web console:

<https://openshift.193183262213.aws.testdrive.openshift.com/console>

You will want to be sure you are logged in as **fancyuser1** with the password **openshift**, who is a **cluster-reader** and can see interesting **Projects**.



At this point the OpenShift UI will display an error message, stating that the metrics URL could not be reached:



An error occurred getting metrics.

This is because OpenShift generated a self-signed certificate for the Hawkular API. Go ahead and click the metrics URL <https://metrics.apps.193183262213.aws.testdrive.openshift.com/> to access Hawkular and accept the untrusted certificate. Then,

return to the OpenShift web console and refresh the page, and the metrics should begin to display.

When working properly, it looks like this:

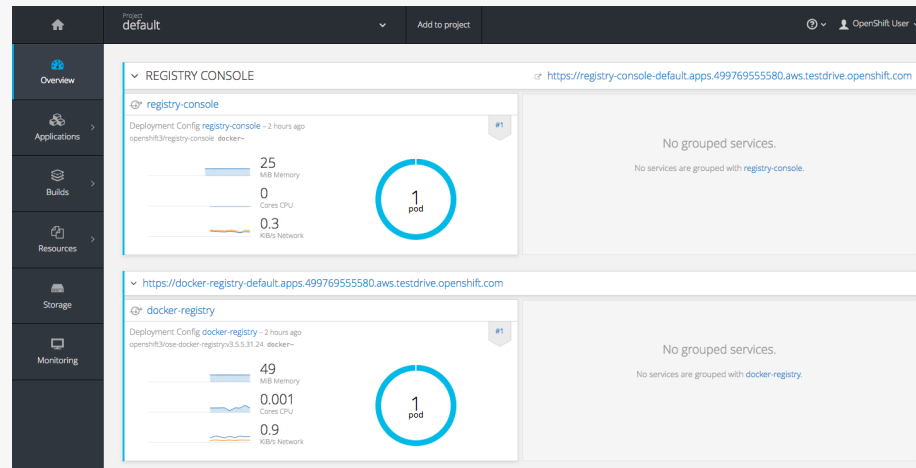
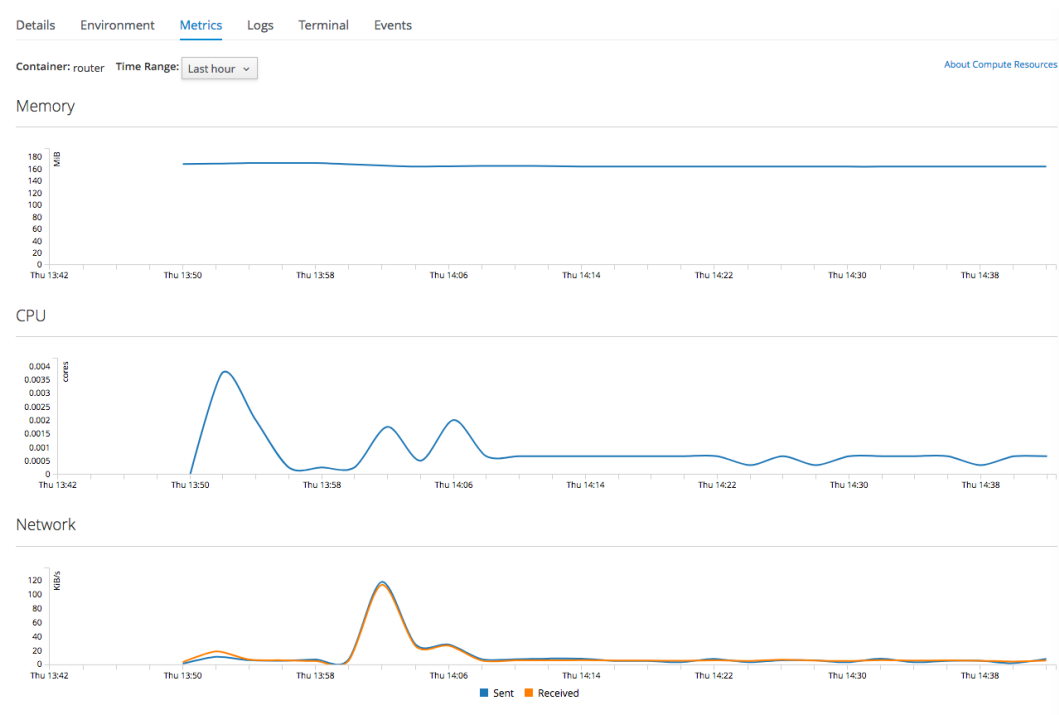


Figure 1. The OpenShift UI will show history metrics for applications

In the context of a specific **Pod**, the *Metrics* tab in the UI will show CPU, memory and network throughput for this particular **Pod** with a configurable time-range. Also optionally a *donut* chart next to a resource appears if the pods was given a consumption limit on this resource (e.g. RAM).



If you want to see interesting metrics, explore the **Project** for metrics itself, [openshift-infra](#).

OpenShift Logging

Equally important to performance metrics is collecting and aggregating logs from the environments and the application pods it is running. OpenShift ships with an elastic log aggregation solution: **EFK**. + **ElasticSearch**, **Fluentd** and **Kibana** forms a configuration where logs from all nodes and applications are consolidated (Fluentd) in a central place (ElasticSearch) on top of which rich queries can be made from a single UI (Kibana). Administrators can see and search through all logs, application owners and developers can allow access logs that belong to their projects. + Like metrics the EFK stack runs on top of OpenShift.

Configuring the Inventory

To configure the installation of EFK edit (update or insert) the Ansible inventory file just like you did for metrics. In the `/etc/ansible/hosts` file, make the following changes:

- remove the line `openshift_logging_install_logging=false`
- remove the comments beginning with `#logging_`

Your resulting file should look like the following:

/etc/ansible/hosts

```
...  
  
[OSEv3:vars]  
...  
openshift_logging_install_logging=true  
openshift_logging_namespace=logging  
openshift_logging_es_pvc_size=10Gi  
openshift_logging_kibana_hostname=kibana.apps.193183262213  
openshift_logging_public_master_url=https://kibana.apps.19  
...
```

Install Logging

With these settings in place executing the `openshift-logging` Ansible playbook that ships as part of the `openshift-ansible` installer:

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/by
```

Once the installation finishes, log in as the cluster administrator, using the **logging Project**:

```
oc login -u system:admin -n logging
```

Verify the logging stack components are up and running:

```
oc get pods -o wide
```

You will see something like:

NAME	READY	STATUS	RESTARTS	AGE
logging-curator-1-cnpt8	1/1	Running	0	5m
logging-es-yeqpfrm5-1-l9k0t	1/1	Running	0	5m
logging-fluentd-2ptb2	1/1	Running	0	4m
logging-fluentd-38lvq	1/1	Running	0	4m
logging-fluentd-9m6rs	1/1	Running	0	4m
logging-fluentd-gstc4	1/1	Running	0	4m
logging-fluentd-h5zjz	1/1	Running	0	4m
logging-fluentd-kkmb	1/1	Running	0	4m
logging-fluentd-twsjg	1/1	Running	0	4m
logging-fluentd-xgh11	1/1	Running	0	5m
logging-kibana-1-dfl8p	2/2	Running	0	5m

The *Fluentd* **Pods** are deployed as part of a **DaemonSet**, which is a mechanism to ensure that specific **Pods** run on specific **Nodes** in the cluster at all times:

```
oc get daemonset
```

You will see something like:

NAME	DESIRED	CURRENT	READY	NODE-SELECTOR
logging-fluentd	5	5	5	logging-infra-flue

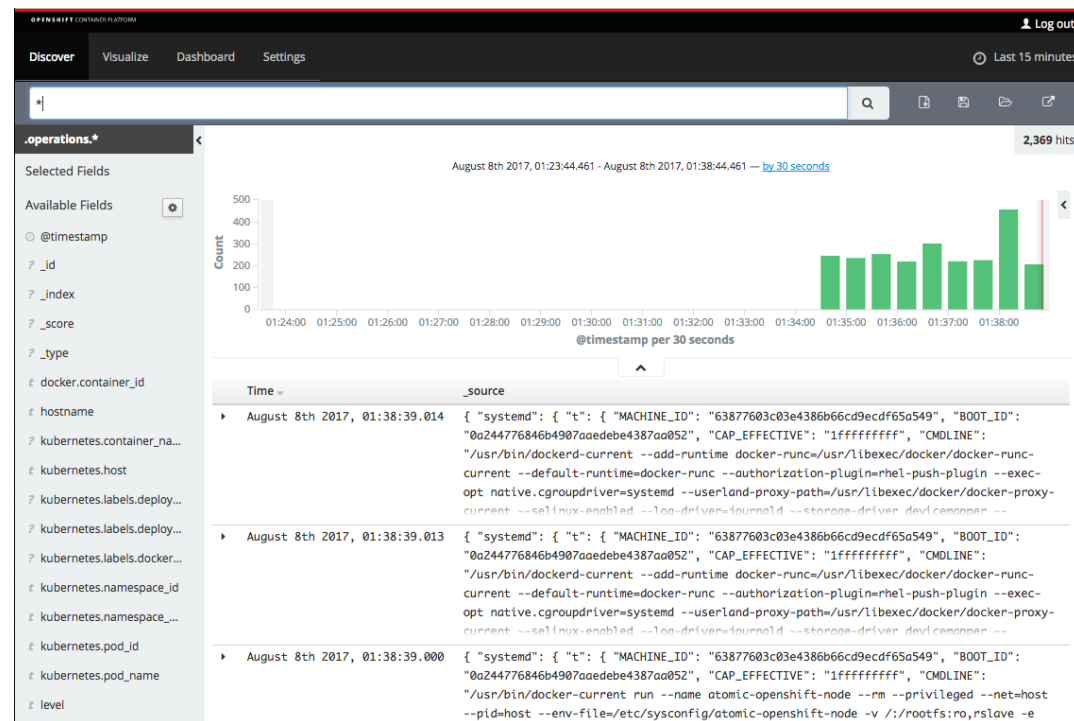
To reach the *Kibana* user interface, first determine its public access URL by querying the **Route** that got set up to expose Kibana's **Service**:

```
oc get route/logging-kibana
```

You will see something like:

NAME	HOST/PORT
logging-kibana	kibana.apps.apps.193183262213.aws.testdrive.opensl

You can click the link (<https://kibana.apps.193183262213.aws.testdrive.openshift.com>) to open the Kibana interface. There is a special authentication proxy that is configured as part of the EFK installation that results in Kibana requiring OpenShift credentials for access. You should login to Kibana as the **fancyuser1** user to be able to see all of the cluster's logs. Kibana utilizes the same RBAC underpinning OpenShift to ensure that users can only see the logs they should have access to.




OpenShift Multitenant Networking

OpenShift has a software defined network (SDN) inside the platform that is based on Open vSwitch. This SDN is used to provide connectivity between application components inside of the OpenShift environment. It comes with

default network ranges pre-configured, although you can make changes to these should they conflict with your existing infrastructure, or for whatever other reason you may have.

When you installed OpenShift, there was an option set in the installer's configuration to enable the multitenant network plugin:

```
os_sdn_network_plugin_name=redhat/openshift-ovs-multitenan
```



The OpenShift Multitenant SDN plug-in enables a true isolated multi-tenant network infrastructure inside OpenShift's software defined network. While you have seen projects isolate resources through OpenShift's RBAC, the multitenant SDN plugin isolates projects using separate virtual network IDs within Open vSwitch.

The multitenant network plugin was introduced in OpenShift 3.1, and more information about it and its configuration can be found in the [networking documentation](#). Additionally, other vendors are working with the upstream Kubernetes community to implement their own SDN plugins, and several of these are supported by the vendors for use with OpenShift. These plugin implementations make use of appc/CNI, which is outside the scope of this lab.

Execute the Creation Script

Only users with cluster administration privileges can manipulate **Project** networks. First, make sure you are logged in as the cluster administrator:


```
oc login -u system:admin
```

Then, execute a script that we have prepared for you. It will create two **Projects** and then deploy a **DeploymentConfig** with a **Pod** for you:

```
bash /opt/lab/support/net-proj.sh
```

Examine Network Namespaces

Two **Projects** were created for you, **netproj-a** and **netproj-b**. Execute the following command to see the network namespaces:

```
oc get netnamespaces
```

You will see something like the following:

NAME	NETID
default	0
kube- system	8046473
logging	2245491
management-infra	693975
netproj-a	8708998
netproj-b	8295735
openshift	10626031
openshift-infra	1151705
...	

Note that each project has its own network namespace with a unique ID. The **default** project is a special exception. Its network ID is 0. This network is a global network. It is joined (not isolated) to all other networks in the SDN by default. If you remember from earlier exercises, the OpenShift router and the image registry are both in the **default** project. This means that **Pods** in all other projects can access them. That's good, because the router needs to be able to proxy traffic to the **Pods** to make them accessible from outside of OpenShift.

Test Connectivity

Now that you have some networks and pods, you will need to find the IP address of the pod in the **netproj-b Project**. The following command will show you the IP address:

```
bash /opt/lab/support/podbip.sh
```

The output will simply be the IP address of the pod in the **netproj-b** project. The everyday way to do this would be with a combination of the **get** and **describe** verbs. Feel free to do the following to verify what the script did:

```
oc get pod -n netproj-b  
oc describe pod ose-1-f0deb
```

Make sure to substitute the correct pod name in the describe command.

describe will show you a lot of information about the pod, including its IP address on the software defined network. Either way, make note of the IP address you found above. It will look something like *10.1.4.12*.

Export the IP address of your pod into a shell variable like so:

```
export POD_B_IP=10.1.4.12
```

Make sure to use the correct IP address that you saw earlier in the command output.

The OpenShift command-line tool and the web console provide mechanisms to execute commands inside **Pods** running in the environment. This is a useful feature for both developers as well as for cluster and application operators/administrators. You will use that feature in order to test network connectivity between the two **Pods** you created.

Get the name of the **Pod** running in the **netproj-a Project**:

```
oc get pods -n netproj-a
```

Then, export the **Pod** ID as a shell variable:

```
export POD_A_NAME=ose-1-q9mt5
```

Be sure to use the name that you saw in the output of your command.

Now, go ahead and **exec** a **ping** command inside **Pod A**, trying to reach **Pod B**:

```
oc exec -n netproj-a $POD_A_NAME -- ping -c1 -W1 $POD_B_IP
```

Your **ping** output should look like the following:

```
PING 10.129.0.10 (10.129.0.10) 56(84) bytes of data.  
  
--- 10.129.0.10 ping statistics ---  
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

You will see 100% packet loss (your **ping** command sends 1 packet, waits 1 second, and gets no response). This is because the networks are not connected to one another. Now simply execute the following:

```
ping -c1 -W1 $POD_B_IP
```

You will see a successful ping. This is because the master (the system you are on) is also a node attached to the SDN. At the host level you are able to reach across all networks, virtual or otherwise. This is important to keep in mind when you consider the overall network-level security of your cluster. Someone logged in to an OpenShift host can "see" and touch everything on the SDN.

Join the Networks

Now it's time to join the networks. Execute the following:

```
oc get netnamespace
```

Take note of the network IDs for **netproj-a** and **netproj-b**. Then:

```
oc adm pod-network join-projects netproj-a --to=netproj-b
```

And then look at the network IDs again:

```
oc get netnamespace
```

You should see that the network IDs of the two projects are now the same.

Retest Connectivity

Now that the projects are joined, your **ping** between the pods should work. Execute the original **ping** test again:

```
oc exec -n netproj-a $POD_A_NAME -- ping -c1 -W1 $POD_B_IP
```

This time, your packet should reach its destination:

```
PING 10.129.0.10 (10.129.0.10) 56(84) bytes of data.  
64 bytes from 10.129.0.10: icmp_seq=1 ttl=64 time=1.07 ms  
  
--- 10.129.0.10 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 1.075/1.075/1.075/0.000 ms
```

Isolate Projects

Now, go ahead and isolate (unjoin) the projects, and then run your **ping** again:

```
oc adm pod-network isolate-projects netproj-a  
oc exec -n netproj-a $POD_A_NAME -- ping -c1 -W1 $POD_B_IP
```

You should see that your **ping** fails again.

Network multitenancy is a bit of a blunt tool. You can either give total access between two projects, or completely restrict access. Don't fret, though. If you need finer-grained control of inter-**Pod** and **Service** communication, there is a Tech Preview network implementation called **NetworkPolicy**. You can learn more about it in the [product documentation](#).

Node Maintenance

It is possible to put any node of the OpenShift environment into maintenance by marking it as non-schedulable following by *evacuation* of all pods on the node.

These operations require elevated privileges. Ensure you are logged in as cluster admin:

```
oc login -u system:admin
```

You will see by now that there are pods running on almost all of your nodes:

```
oc get pods --all-namespaces -o wide
```

When a node needs to undergo maintenance like replacing degraded hardware components or updating packages you can temporarily remove it from OpenShift like so:

Mark node `node02.internal.aws.testdrive.openshift.com` as non-schedulable to prevent the schedulers in the system to place any new workloads on it:

```
oc adm manage-node node02.internal.aws.testdrive.openshift.com --s
```

The output of the command will show that the node is now not schedulable:

NAME	STATUS
node02.internal.aws.testdrive.openshift.com	Ready,SchedulingDisal

Marking the node out like this did not impact the pods it is running. List those pods:

```
oc adm manage-node node02.internal.aws.testdrive.openshift.com --l
```

Other than a **Pod** for Container Native Storage and a Fluentd instance (there is one on every node), there may or may not be other **Pods** running on this node.

The next step is to evacuate the **Pods** to other nodes in the cluster. You can first simulate what actions the system would perform during evacuation with the following command:

```
oc adm manage-node node02.internal.aws.testdrive.openshift.com --e
```



Pods running on the node as part of a **DaemonSet** like those associated to Logging or CNS will **not** be evacuated. They will

not be accessible anymore through OpenShift, but will continue to run as containers on the nodes until the local OpenShift services are stopped and/or the node is shutdown. This is not a problem since software like CNS or the OpenShift Metrics stack is designed to handle such situations transparently.

Start the evacuation process like this:

```
oadm manage-node node02.internal.aws.testdrive.openshift.com --evac
```

After a few moments, all of the **Pods**, except those for Fluentd and Container Native Storage, previously running on `node02.internal.aws.testdrive.openshift.com` should have terminated and been launched elsewhere.

```
oc get pods --all-namespaces -o wide
```

The node `node02.internal.aws.testdrive.openshift.com` is now ready for an administrator to start maintenance operations. If those include a reboot of the system or upgrading OpenShift components, the **Pods** associated with CNS and logging will come back up automatically.

Now that our maintenance is complete, the node is still non-schedulable. Let's fix that:

```
oc adm manage-node node02.internal.aws.testdrive.openshift.com --s
```

Now the node will be able to have workload scheduled on it again:

NAME	STATUS	AGE
node02.internal.aws.testdrive.openshift.com	Ready	4h

[Go to previous module](#)[Go to next module](#)