Assess the Health of an OpenShift Cluster

Objectives

- Query the health of essential cluster services and components.

Query Operator Conditions

Operators are important components of Red Hat OpenShift Container Platform (RHOCP). Operators automate the required tasks to maintain a healthy RHOCP cluster that would otherwise require human intervention. Operators are the preferred method of packaging, deploying, and managing services on the control plane.

Operators integrate with Kubernetes APIs and CLI tools such as `kubectl` and `oc` commands. Operators provide the means of monitoring applications, performing health checks, managing over-the-air (OTA) updates, and ensuring that applications remain in your specified state.

Because CRI-O and the Kubelet run on every node, almost every other cluster function can be managed on the control plane by using Operators. Components that are added to the control plane by using operators include critical networking and credential services.

Operators in RHOCP are managed by two different systems, depending on the purpose of the operator.

Cluster Version Operator (CVO)
Cluster operators perform cluster functions. These operators are installed by default, and the CVO manages them.

Cluster operators use a Kubernetes `kind` value of `clusteroperators`, and thus can be queried via `oc` or `kubectl` commands. As a user with the `cluster-admin` role, use the `oc get clusteroperators` command to list all the cluster operators.

```
[user@host ~]$ oc get clusteroperators
NAME                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
authentication             4.18.6    True        False         False      3d1h
baremetal                  4.18.6    True        False         False      38d
cloud-controller-manager   4.18.6    True        False         False      38d
cloud-credential           4.18.6    True        False         False      38d
cluster-autoscaler         4.18.6    True        False         False      38d
config-operator            4.18.6    True        False         False      38d
console                    4.18.6    True        False         False      38d
...output omitted...
```

For more details about a cluster operator, use the `describe clusteroperators` *`operator-name`* command to view the field values that are associated with the operator, including the current status of the operator. The `describe` command provides a human-readable output format for a resource. As such, the output format might change with an RHOCP version update.

For an output format that is less likely to change with a version update, use one of the `-o` output options of the `get` command. For example, use the following `oc get clusteroperators` command for the YAML-formatted output details for the `dns` operator.

```
[user@host ~]$ oc get clusteroperators dns -o yaml
apiVersion: config.openshift.io/v1
kind: ClusterOperator
metadata:
  annotations:
...output omitted...
status:
  conditions:
  - lastTransitionTime: "2025-07-14T13:55:21Z"
    message: DNS "default" is available.
    reason: AsExpected
    status: "True"
    type: Available
...output omitted...
  relatedObjects:
  - group: ""
    name: openshift-dns-operator
    resource: namespaces
...output omitted...
  versions:
  - name: operator
    version: 4.18.6
...output omitted...
```

Operator Lifecycle Manager (OLM) Operators
Optional add-on operators that the OLM manages can be made accessible for users to run in their applications.

As a user with the `cluster-admin` role, use the `get operators` command to list all the add-on operators.

```
[user@host ~]$ oc get operators
NAME                                AGE
lvms-operator.openshift-storage     44d
metallb-operator.metallb-system     44d
```

You can likewise use the `describe` and `get` commands to query details about the fields that are associated with the add-on operators. Operators use one or more pods to provide cluster services. You can find the namespaces for these pods under the `relatedObjects` section of the detailed output for the operator. As a user with a `cluster-admin` role, use the `-n` *`namespace`* option on the `get pod` command to view the pods. For example, use the following `get pods` command to retrieve the list of pods in the `openshift-dns-operator` namespace.

```
[user@host ~]$ oc get pods -n openshift-dns-operator
NAME                            READY   STATUS    RESTARTS   AGE
dns-operator-74bb989655-vgm7t   2/2     Running   18         56d
```

Use the `-o yaml` or `-o json` output formats to view or analyze more details about the pods. The resource conditions, which are found in the status for the resource, track the current state of the resource object. The following example uses the `jq` processor to extract the `status` values from the JSON output details for the `dns` pod.

```
[user@host ~]$ oc get pod -n openshift-dns-operator \
dns-operator-74bb989655-vgm7t -o json | jq .status
{
  "conditions": [
    {
      "lastProbeTime": null,
      "lastTransitionTime": "2025-07-14T16:04:38Z",
      "status": "True",
      "type": "PodReadyToStartContainers"
    },
...output omitted...
```

In addition to listing the pods of a namespace, you can also use the `--show-labels` option of the `get` command to print the labels used by the pods. The following example retrieves the pods and their labels in the `openshift-etcd` namespace.

```
[user@host ~]$ oc get pods -n openshift-etcd --show-labels
NAME                   READY   STATUS      RESTARTS   AGE   LABELS
etcd-master01          5/5     Running     50         56d   app=etcd,etcd=true,k8s-app=etcd,revision=2
installer-1-master01   0/1     Completed   0          56d   app=installer
installer-2-master01   0/1     Completed   0          56d   app=installer
```

Examining Cluster Metrics

Another way to gauge the health of an RHOCP cluster is to examine the compute resource usage of cluster nodes and pods. The `oc adm top` command provides this information. For example, to list the total memory and CPU usage of all pods in the cluster, you can use the `--sum` option with the command to print the sum of the resource usage.

```
[user@host ~]$ oc adm top pods -A --sum
NAMESPACE                NAME                       CPU(cores)   MEMORY(bytes)
metallb-system           controller-...-fxhh4       2m           43Mi
metallb-system           metallb-...-t9pgn          1m           19Mi
...output omitted...
openshift-storage        topolvm-node-f9rpf         7m           47Mi
openshift-storage        vg-manager-q6zhf           1m           21Mi
                                                    ------       --------
                                                    3121m        9123Mi
```

The `-A` option shows pods from all namespaces. Use the `-n` *`namespace`* option to filter the results to show the pods in a single namespace. Use the `--containers` option to display the resource usage of containers within a pod. For example, use the following command to list the resource usage of the containers in the `etcd-master01` pod in the `openshift-etcd` namespace.

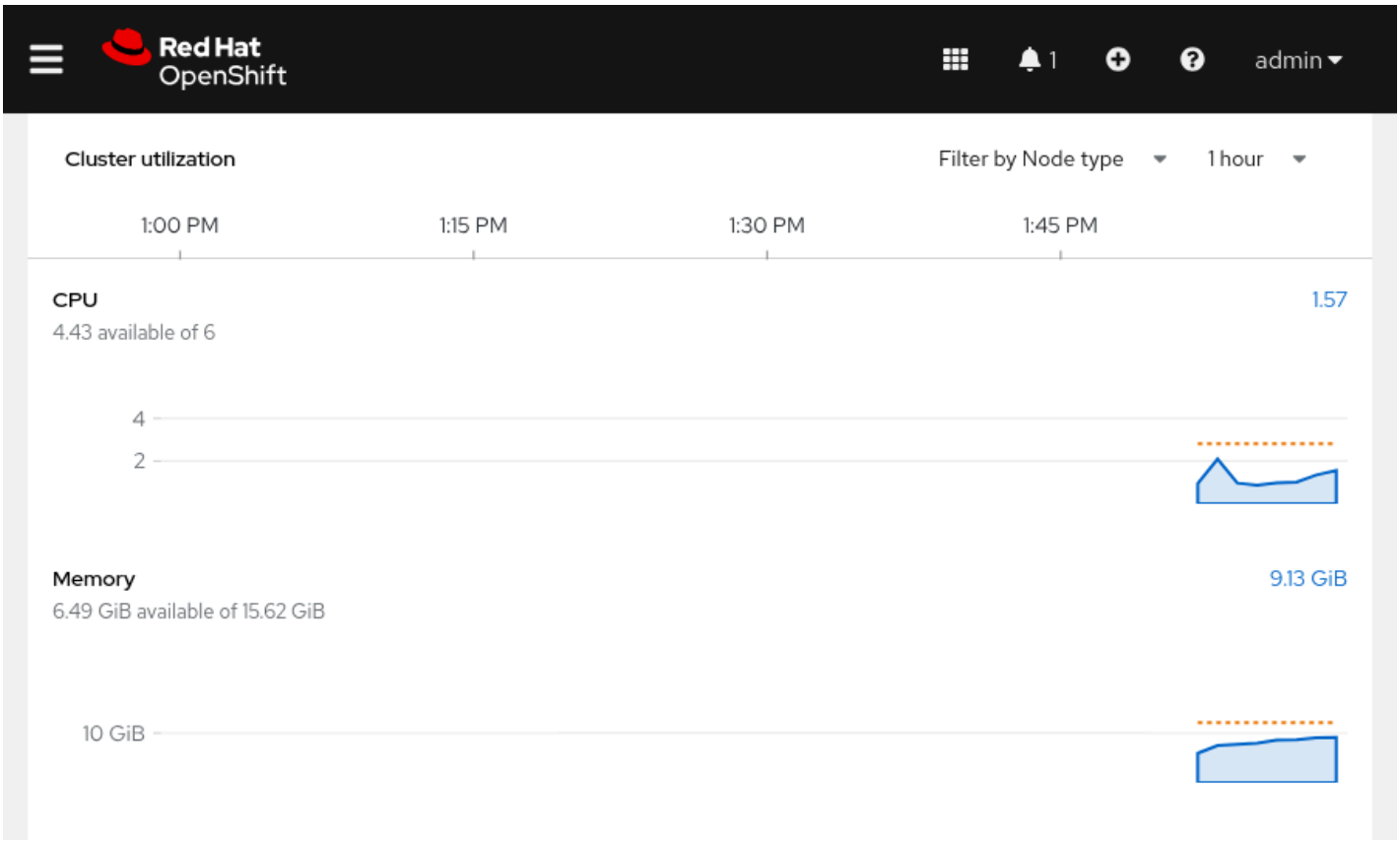
```
[user@host ~]$ oc adm top pods etcd-master01 -n openshift-etcd --containers
POD             NAME           CPU(cores)   MEMORY(bytes)
etcd-master01   etcd           72m          334Mi
etcd-master01   etcd-metrics   8m           31Mi
etcd-master01   etcd-readyz    3m           46Mi
etcd-master01   etcd-rev       1m           33Mi
etcd-master01   etcdctl        0m           0Mi
```

Viewing Cluster Metrics

The OpenShift web console incorporates graphs to visualize cluster and resource analytics. Cluster administrators and users with either the `view` or the `cluster-monitoring-view` cluster role can access the **Home** → **Overview** page. The Overview page displays a collection of cluster-wide metrics, and provides a high-level view of the overall health of the cluster.

The Overview page displays the following metrics:

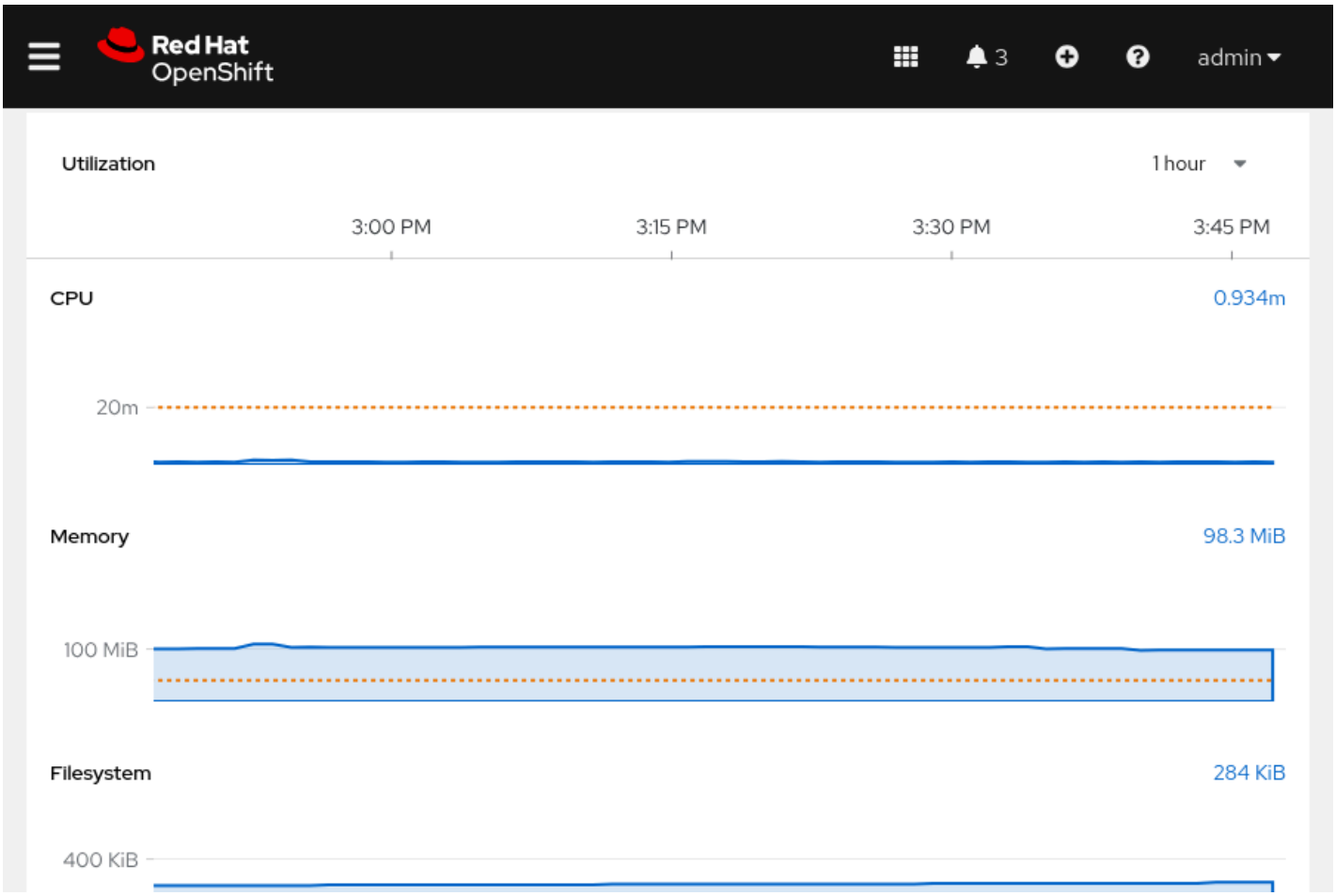
- Current cluster capacity based on CPU, memory, storage, and network usage
- A time-series graph of total CPU, memory, and disk usage
- The ability to display the top consumers of CPU, memory, and storage



For any of the listed resources in the **Cluster Utilization** section, administrators can click the link for current resource usage. The link displays a window with a breakdown of top consumers for that resource. Top consumers can be sorted by project, by pod, or by node. The list of top consumers can be useful for identifying problematic pods or nodes. For example, a pod with an unexpected memory leak might appear at the top of the list.

Viewing Project Metrics

The **Project Details** page displays metrics that provide an overview of the resources that are used within the scope of a specific project. The **Utilization** section displays usage information about resources, such as CPU and memory, along with the ability to display the top consumers for each resource.



All metrics are pulled from Prometheus. Click any graph to go to the **Metrics** page. View the executed query, and inspect the data further.

If a resource quota is created for the project, then the current project request and limits appear on the **Project Details** page.

Viewing Resource Metrics

When troubleshooting, it is often useful to view metrics at a smaller granularity than for the entire cluster or project. The **Metrics** page displays time-series graphs of the CPU, memory, and file-system usage for a specific pod. A sudden change in these critical metrics, such as a CPU spike caused by high load, is visible on this page.

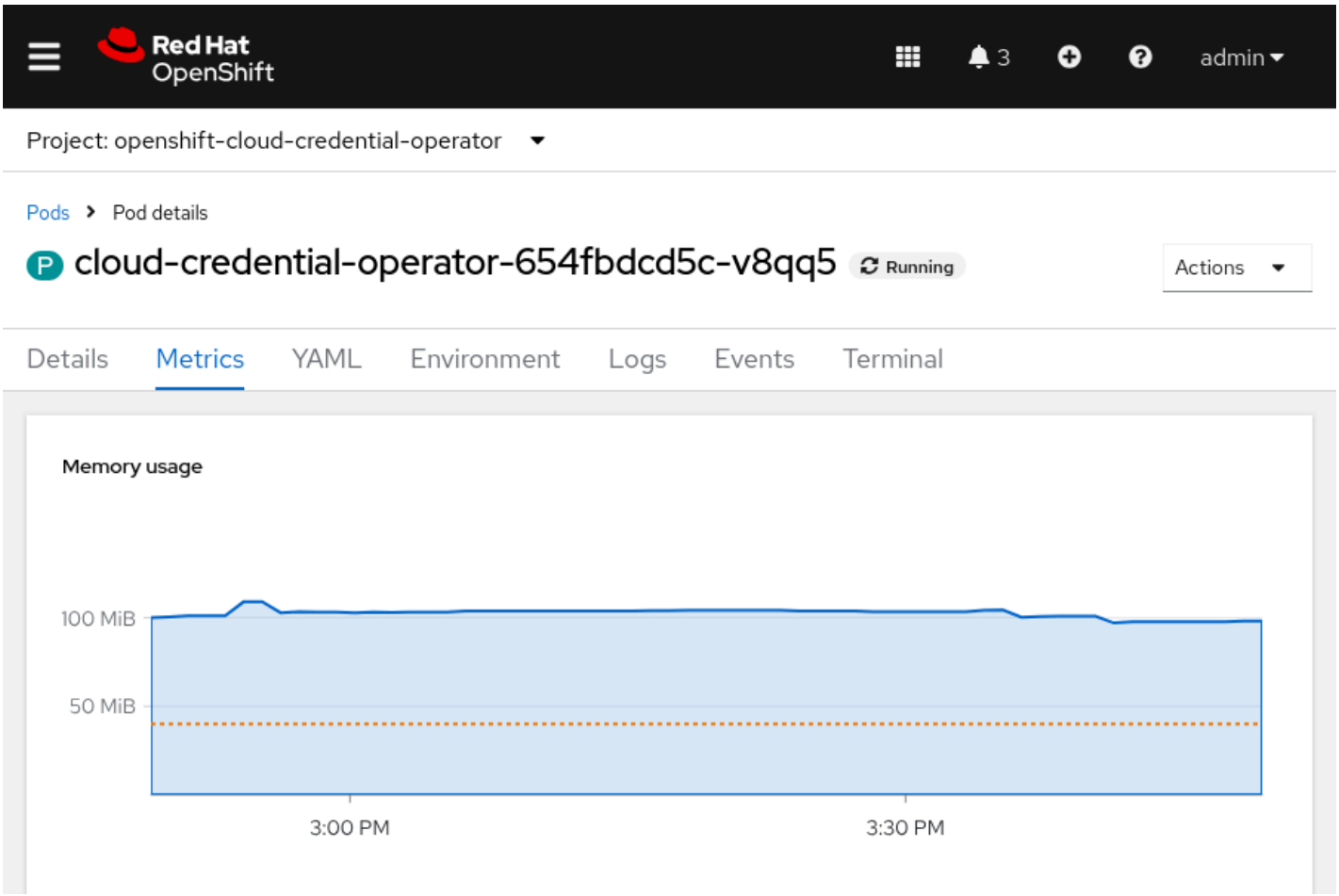


Figure 2.8: Time-series graphs showing various metrics for a pod

Performing Prometheus Queries in the Web Console

The Prometheus UI is a feature-rich tool for visualizing metrics and configuring alerts. The OpenShift web console provides an interface for executing Prometheus queries directly from the web console.

To perform a query, go to **Observe** → **Metrics**, enter a Prometheus Query Language expression in the text field, and click **Run Queries**. The results of the query are displayed as a time-series graph:

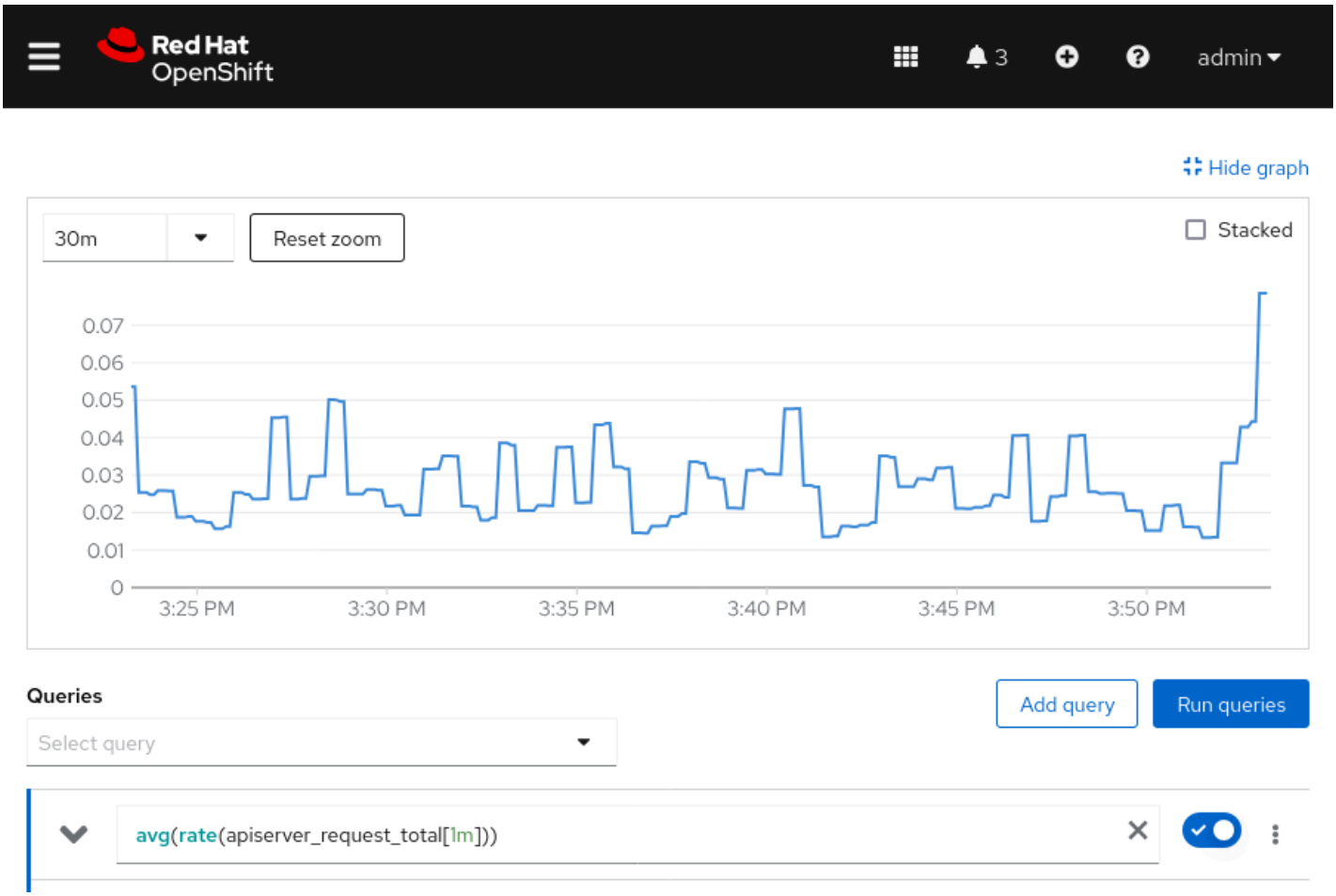


Figure 2.9: Using a Prometheus query to display a time-series graph

> **NOTE**
>
> The Prometheus Query Language is not discussed in detail in this course. Refer to the references section for a link to the official documentation.

Query Cluster Events and Alerts

Some developers consider OpenShift logs to be too low-level, thus making troubleshooting difficult. Fortunately, RHOCP provides a high-level logging and auditing facility called *events*. Kubernetes generates `event` objects in response to state changes in cluster objects, such as nodes, pods, and containers. Events signal significant actions, such as starting a container or destroying a pod.

To read events, use the `get events` command. The command lists the events for the current RHOCP project (namespace). You can display the events for a different project by adding the `-n` *`namespace`* option to the command. To list the events for all the projects, use the `-A` (or `--all-namespaces`) option.

> **NOTE**
>
> To sort the events by time, add the `--sort-by .metadata.creationTimestamp` option to the `oc get events` command.

The following `get events` command prints events in the `openshift-kube-controller-manager` namespace.

```
[user@host ~]$ oc get events -n openshift-kube-controller-manager
LAST SEEN  TYPE    REASON             OBJECT                                MESSAGE
75d        Normal  LeaderElection     lease/cert-recovery-controller-lock   master01_...
...output omitted...
```

You can use the `describe pod` *`pod-name`* command to further narrow the results to a single pod. Refer to the `Events` field from the output of the `oc describe pod` *`pod-name`* command to review events for the specified pod.

Kubernetes Alerts

RHOCP includes a monitoring stack, which is based on the Prometheus open source project. The monitoring stack is configured to monitor the core RHOCP cluster components, by default. You can optionally configure the monitoring stack to also monitor user projects.

The components of the monitoring stack are installed in the `openshift-monitoring` namespace. Use the following `get all` command to display a list of all resources, their status, and their types in the `openshift-monitoring` namespace.

```
[user@host ~]$ oc get all -n openshift-monitoring
NAME                                           READY  STATUS    RESTARTS AGE
pod/alertmanager-main-0                        6/6    Running   48       75d
pod/cluster-monitoring-operator-55b4dbf86-mb4xd 1/1    Running   8        75d
pod/kube-state-metrics-75455b796c-8q28d        3/3    Running   27       75d
pod/prometheus-k8s-0                           6/6    Running   48       75d
...output omitted...
```

The Prometheus Operator in the `openshift-monitoring` namespace creates, configures, and manages the Prometheus and Alertmanager instances. The `prometheus-k8s-0` entry is the Prometheus pod. A cluster administrator can use the following command to get the alerts from the Prometheus API.

```
[user@host ~]$ oc -n openshift-monitoring exec -c prometheus \
prometheus-k8s-0 -- curl -s   'http://localhost:9090/api/v1/alerts' | jq
...output omitted...
```

An Alertmanager pod in the `openshift-monitoring` namespace receives alerts from Prometheus. Alertmanager uses alert receivers to send alerts to external notification systems. The `alertmanager-main-0` pod is the Alertmanager for the cluster. Use the `curl` command to retrieve the fired alerts from the Alertmanager API.

```
[user@host ~]$ oc -n openshift-monitoring exec -c alertmanager \
alertmanager-main-0 -- curl -s  'http://localhost:9093/api/v2/alerts' | jq
...output omitted...
```

Check Node Status

RHOCP clusters can have several components, including at least one control plane and at least one compute node. The two components can occupy a single node. The following `oc` command, or the matching `kubectl` command, can display the overall health of all cluster nodes.

```
[user@host ~]$ oc cluster-info
```

The `oc cluster-info` output is high-level, and can verify that the cluster nodes are running. For a more detailed view into the cluster nodes, use the `get nodes` command.

```
[user@host ~]$ oc get nodes
NAME       STATUS   ROLES                         AGE   VERSION
master01   Ready    control-plane,master,worker   75d   v1.31.6
```

The example shows a single `master01` node with multiple roles. The `STATUS` value of `Ready` means that this node is healthy and can accept new pods. A `STATUS` value of `NotReady` means that a condition triggered the `NotReady` status and the node is not accepting new pods.

As with any other RHOCP resource, you can drill down into further details of the node resource with the `describe node` *`node-name`* command. For parsable output of the same information, use the `-o json` or the `-o yaml` output options with the `get node` *`node-name`* command. For more information about using and parsing these output formats, see the section called " Inspect Kubernetes Resources ".

The output of the `get nodes` *`node-name`* command with the `-o json` or `-o yaml` option is long. The following examples use the `-jsonpath` option or the `jq` processor to parse the `get node` *`node-name`* command output.

```
[user@host ~]$ oc get node master01 -o jsonpath=\
*'{"Allocatable:\n"}{.status.allocatable}{"\n\n"}
{"Capacity:\n"}{.status.capacity}{"\n"}'

Allocatable:
{"cpu":"5500m","ephemeral-storage":"75691125429","hugepages-1Gi":"0",
"hugepages-2Mi":"0","memory":"15225452Ki","pods":"250"}

Capacity:
{"cpu":"6","ephemeral-storage":"83295212Ki","hugepages-1Gi":"0",
"hugepages-2Mi":"0","memory":"16376428Ki","pods":"250"}
```

The JSONPath expression in the previous command extracts the allocatable and capacity measures for the `master01` node. These measures help to understand the available resources on a node.

View the `status` object of a node to understand the current health of the node.

```
[user@host ~]$ oc get node master01 -o json | jq '.status.conditions'
[
  {
    "lastHeartbeatTime": "2025-08-05T15:32:36Z",
    "lastTransitionTime": "2025-05-22T12:12:24Z",
    "message": "kubelet has sufficient memory available",
    "reason": "KubeletHasSufficientMemory",
    "status": "False",
    "type": "MemoryPressure" (1)
  },
  {
    "lastHeartbeatTime": "2025-08-05T15:32:36Z",
    "lastTransitionTime": "2025-05-22T12:12:24Z",
    "message": "kubelet has no disk pressure",
    "reason": "KubeletHasNoDiskPressure",
    "status": "False",
    "type": "DiskPressure" (2)
  },
  {
    "lastHeartbeatTime": "2025-08-05T15:32:36Z",
    "lastTransitionTime": "2025-05-22T12:12:24Z",
    "message": "kubelet has sufficient PID available",
    "reason": "KubeletHasSufficientPID",
    "status": "False",
    "type": "PIDPressure" (3)
  },
  {
    "lastHeartbeatTime": "2025-08-05T15:32:36Z",
    "lastTransitionTime": "2025-08-05T14:41:42Z",
    "message": "kubelet is posting ready status",
    "reason": "KubeletReady",
    "status": "True",
    "type": "Ready" (4)
  }
]
```

(1) If the status of the `MemoryPressure` condition is true, then the node is low on memory.

(2) If the status of the `DiskPressure` condition is true, then the disk capacity of the node is low.

(3) If the status of the `PIDPressure` condition is true, then too many processes are running on the node.

(4) If the status of the `Ready` condition is false, then the node is not healthy and is not accepting pods.

More conditions indicate other potential problems with a node.

Table 2.5. Possible Node Conditions

Condition	Description
`OutOfDisk`	If true, then the node has insufficient free space on the node for adding new pods.
`NetworkUnavailable`	If true, then the network for the node is not correctly configured.
`NotReady`	If true, then one of the underlying components, such as the container runtime or network, is experiencing issues or is not yet configured.
`SchedulingDisabled`	Pods cannot be scheduled for placement on the node.

To gain deeper insight into a given node, you can view the logs of processes that run on the node. A cluster administrator can use the `oc adm node-logs` command to view node logs. Node logs might contain sensitive output, and thus are limited to privileged node administrators. Use `oc adm node-logs` *`node-name`* to filter the logs to a single node.

The `oc adm node-logs` command has other options to further filter the results.

Table 2.6. Filters for `oc adm node-logs`

Option Example	Description
`--role master`	Use the `--role` option to filter the output to nodes with a specified role.
`-u kubelet`	The `-u` option filters the output to a specified unit.
`--path=cron`	The `--path` option filters the output to a specific process under the `/var/logs` directory.
`--tail 1`	Use `--tail x` to limit output to the last `x` log entries.

Use `oc adm node-logs --help` for a complete list of command options.

For example, to retrieve the most recent log entry for the `crio` service on the `master01` node, you can use the following command.

```
[user@host ~]$ oc adm node-logs master01 -u crio --tail 1
-- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Fri 2023-03-17 15:11:43 UTC. --
Aug 05 15:16:09.519642 master01 crio[2783]: time="2025-08-05 15:30:09.519474755Z" level=info msg="Stopped pod sandbox...
...output omitted...
```

When you create a pod with the CLI, the `oc` or `kubectl` command is sent to the `apiserver` service, which then validates the command. The `scheduler` service reads the YAML or JSON pod definition, and then assigns pods to compute nodes. Each compute node runs a `kubelet` service that converts the pod manifest to one or more containers in the CRI-O container runtime.

Each compute node must have an active `kubelet` service and an active `crio` service. To verify the health of these services, first start a debug session on the node by using the `debug` command.

```
[user@host ~]$ oc debug node/node-name
```

Replace the *`node-name`* value with the name of your node.

> **NOTE**
>
> The `debug` command is covered in greater detail in a later section.

Within the debug session, change to the `/host` root directory so that you can run binaries in the host's executable path.

```
sh-4.4# chroot /host
```

Then, use the `systemctl is-active` calls to confirm that the services are active.

```
sh-4.4# for SERVICES in kubelet crio; do echo ---- $SERVICES ---- ;
systemctl is-active $SERVICES ;  echo ""; done
---- kubelet ----
active

---- crio ----
active
```

For more details about the status of a service, use the `systemctl status` command.

```
sh-4.4# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
...output omitted...
```

Check Pod Status

With RHOCP, you can view logs in running containers and pods to ease troubleshooting. When a container starts, RHOCP redirects the container's standard output and standard error to a disk in the container's ephemeral storage. With this redirect, you can view the container logs by using `logs` commands, even after the container stops. However, the pod hosting the container must still exist.

In RHOCP, the following command returns the output for a container within a pod:

```
[user@host ~]$ oc logs pod-name -c container-name
```

Replace *pod-name* with the name of the target pod, and replace *container-name* with the name of the target container. The `-c` *container-name* argument is optional, if the pod has only one container. You must use the `-c` *container-name* argument to connect to a specific container in a multicontainer pod. Otherwise, the command defaults to the only running container and returns the output.

When debugging images and setup problems, it is useful to get an exact copy of a running pod configuration, and then troubleshoot it with a shell. If a pod is failing or does not include a shell, then the `rsh` and `exec` commands might not work. To resolve this issue, the `debug` command creates a copy of the specified pod and starts a shell in that pod.

By default, the `debug` command starts a shell inside the first container of the referenced pod. The debug pod is a copy of your source pod, with some additional modifications. For example, the pod labels are removed. The executed command is also changed to the '/bin/sh' command for Linux containers, or the 'cmd.exe' executable for Windows containers. Additionally, readiness and liveness probes are disabled.

A common problem for containers in pods is security policies that prohibit a container from running as a root user. You can use the `debug` command to test running a pod as a non-root user by using the `--as-user` option. You can also run a non-root pod as the root user with the `--as-root` option.

With the `debug` command, you can invoke other types of objects besides pods. For example, you can use any controller resource that creates a pod, such as a deployment, a build, or a job. The `debug` command also works with nodes, and with resources that can create pods, such as image stream tags. You can also use the `--image=IMAGE` option of the `debug` command to start a shell session by using a specified image.

If you do not include a resource type and name, then the `debug` command starts a shell session into a pod by using the OpenShift tools image.

```
[user@host ~]$ oc debug
```

The next example tests running a job pod as a non-root user.

```
[user@host ~]$ oc debug job/test --as-user=1000000
```

The following example creates a debug session for a node.

```
[user@host ~]$ oc debug node/master01
Starting pod/master01-debug-gdmkp ...
To use host binaries, run chroot /host
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-5.1# chroot /host
sh-5.1#
```

The debug pod is deleted when the remote command completes, or when the user interrupts the shell.

Collect Information for Support Requests

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support. It is recommended that you provide the following information:

- Data gathered by using the `oc adm must-gather` command as a cluster administrator
- The unique cluster ID

The `oc adm must-gather` command collects resource definitions and service logs from your cluster that are most likely needed for debugging issues. This command creates a pod in a temporary namespace on your cluster, and the pod then gathers and downloads debugging information. By default, the `oc adm must-gather` command uses the default plug-in image, and writes into the `./must-gather.local.` directory on your local system. To write to a specific local directory, you can also use the `--dest-dir` option, such as in the following example:

```
[user@host ~]$ oc adm must-gather --dest-dir /home/student/must-gather
```

Then, create a compressed archive file from the `must-gather` directory. For example, on a Linux-based system, you can run the following command:

```
[user@host ~]$ tar cvaf mustgather.tar must-gather/
```

Replace `must-gather/` with the actual directory path.

Then, attach the compressed archive file to your support case in the Red Hat Customer Portal.

Similar to the `oc adm must-gather` command, the `oc adm inspect` command gathers information on a specified resource. For example, the following command collects debugging data for the `openshift-apiserver` and `kube-apiserver` cluster operators.

```
[user@host ~]$ oc adm inspect clusteroperator/openshift-apiserver \
clusteroperator/kube-apiserver
```

The `oc adm inspect` command can also use the `--dest-dir` option to specify a local directory to write the gathered information. The command shows all logs by default. Use the `--since` option to filter the results to logs that are later than a relative duration, such as 5s, 2m, or 3h.

```
[user@host ~]$ oc adm inspect clusteroperator/openshift-apiserver --since 10m
```

REFERENCES

For more information, refer to the *Control Plane Architecture* chapter in the Red Hat OpenShift Container Platform 4.18 *Architecture* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/architecture/index#control-plane

For more information, refer to the Red Hat OpenShift Container Platform 4.18 *Monitoring* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/monitoring/index

For more information, refer to the Red Hat OpenShift Container Platform 4.18 *Nodes* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/nodes/index#nodes-nodes-viewing

Querying Prometheus

For more information, refer to the *Troubleshooting Kubernetes* chapter in the Red Hat Enterprise Linux Atomic Host 7 *Getting Started with Kubernetes* documentation at https://docs.redhat.com/en/documentation/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_kubernetes/index#overview_2

For more information about gathering diagnostic data about your cluster, refer to the *Gathering data about your cluster* chapter in the Red Hat OpenShift Container Platform 4.18 *Support* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/support/index#gathering-cluster-data