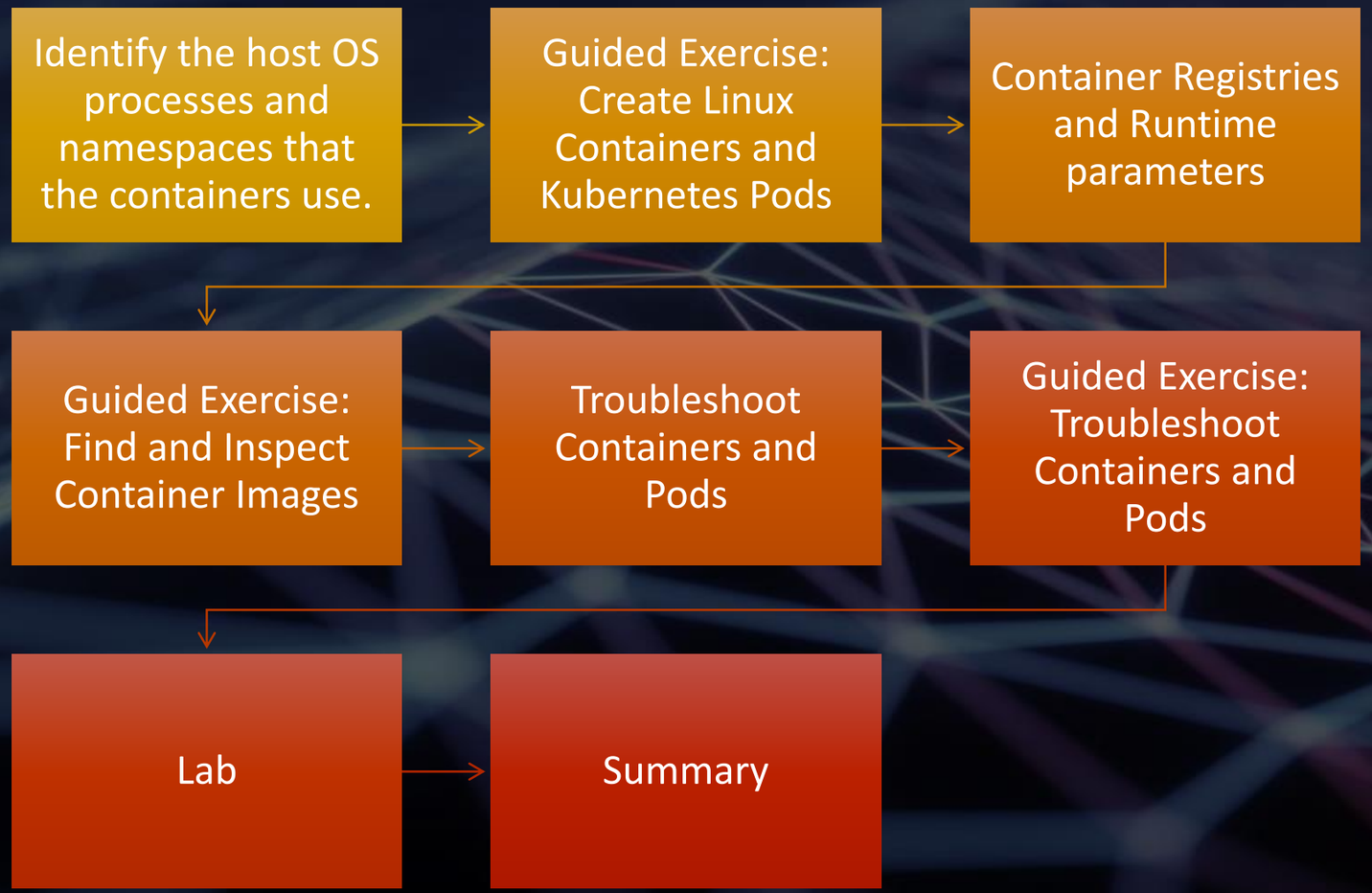
Chapter 3: Run Applications as Containers and Pods

Chapter objectives



Lesson 1 : Create Linux Containers and Kubernetes Pods

- Run containers inside pods and identify the host OS processes and namespaces that the containers use

Creating Containers and Pods

- Syntax

oc run *RESOURCE/NAME* --image IMAGE [options] or

kubectl run *RESOURCE/NAME* --image IMAGE [options]

- Example

oc run web-server --image registry.access.redhat.com/ubi8/httpd-24

kubectl run my-app --image registry.access.redhat.com/ubi9/ubi

Runtime Parameters / Options

- --command
- -i / --stdin
- -t / --tty
- --restart
- --env
- --rm

Use the --command option

- Syntax

```
oc run RESOURCE/NAME --image IMAGE --command -- cmd arg1 ... argN

kubectl run RESOURCE/NAME --image IMAGE --command -- cmd arg1 ... argN
```

- Example

```
oc run web-server --image registry.access.redhat.com/ubi8/httpd-24 --command -- sleep 500

kubectl run my-app --image registry.access.redhat.com/ubi9/ubi --command -- date
```

Use the --rm and -i -t option

- -i option : Interactive mode
- -t option : open a TTY session
- --rm option : delete pod after it exits
- --restart : restart pod
- Example 1

```
[user@host ~]$ oc run -it my-app \
--image registry.access.redhat.com/ubi9/ubi \
--restart Never --command -- date
Mon Feb 20 22:36:55 UTC 2023
```

- Example 2

```
[user@host ~]$ kubectl run -it my-app --rm \
--image registry.access.redhat.com/ubi9/ubi \
--restart Never --command -- date
Mon Feb 20 22:38:50 UTC 2023
pod "date" deleted
```

The --restart option

Accepted Values	Description
--restart Always	Cluster continuously tries to restart a successfully exited container, for up to five minutes. The default pod restart policy is Always.
--restart OnFailure	Setting the pod restart policy to OnFailure tells the cluster to restart only failed containers in the pod, for up to five minutes
--restart Never	If the restart policy is set to Never, then the cluster does not try to restart exited or failed containers in a pod. Instead, the pods immediately fail and exit.

The --env option

- Start a database system using MySQL image. At same time set the root password

```
[user@host ~]$ oc run mysql \
--image registry.redhat.io/rhel9/mysql-80 \
--env MYSQL_ROOT_PASSWORD=myP@$$123
pod/mysql created
```

User and Group IDs Assignment

- Defined at project/namespace level
- Determine UID and GID for pods and containers
- Default security policies
  - rootless container cannot choose USER or UID
  - OpenShift assigns user in container to UID and GID from range
  - ignore USER instruction in container image
  - rootfull container can choose USER - but not recommended
- Red Hat recommendations:
  - Always run containers as rootless
  - Use unprivileged user with necessary privileges to specific commands

The UID and supplemental GID range

```
[user@host ~]$ oc describe project my-app
Name:   my-app
...output omitted...
Annotations:   openshift.io/description=
      openshift.io/display-name=
      openshift.io/requester=developer
      openshift.io/sa.scc.mcs=s0:c27,c4
      openshift.io/sa.scc.supplemental-groups=1000710000/10000
      openshift.io/sa.scc.uid-range=1000710000/10000
...output omitted...
```

- The UID and GID range follow the format
  <first_id>/<id_pool_size> or <first_id>-<last_id>

- Assign distinct range of UIDs and GIDs for each project
  - ensures applications do not run as the same UID and GID
  - if two containers is the same, processes in one container can access resources in the other

Pod Security

- Security Context Constraints (SCC)
- *Grant* or *Deny* OS-level privileges in pods
- Undefined security context - issue warning
- Safely ignore pod security warnings in course exercises
- Discussed in more details in DO280 course.

Execute Commands in Running Containers

- Use the **oc exec** or **kubectl exec** command
- The syntax

```
oc exec RESOURCE/NAME -- COMMAND [args...] [options]
```

- Example: Execute date command on running my-app

```
[user@host ~]$ oc exec my-app -- date
Tue Feb 21 20:43:53 UTC 2023
```

- Example: For multiple container in pod, use the -c option

```
[user@host ~]$ kubectl exec my-app -c ruby-container -- date
Tue Feb 21 20:46:50 UTC 2023
```

Execute Commands in Running Containers

- Combine with -i and -t options

```
[user@host ~]$ oc exec my-app -c ruby-container -it -- bash -il
[1000780000@ruby-container /]$
```

- Exit command to terminate the interactive session

```
[user@host ~]$ kubectl exec my-app -c ruby-container -it -- bash -il
[1000780000@ruby-container /]$ date
Tue Feb 21 21:16:00 UTC 2023
[1000780000@ruby-container] exit
```

Attach to a running container

- is to gain access to the input/output stream of a container.

```
# Get output from running pod mypod; use the 'oc.kubernetes.io/default-container' annotation
# for selecting the container to be attached or the first container in the pod will be chosen
oc attach mypod

# Get output from ruby-container from pod mypod
oc attach mypod -c ruby-container

# Switch to raw terminal mode; sends stdin to 'bash' in ruby-container from pod mypod
# and sends stdout/stderr from 'bash' back to the client
oc attach mypod -c ruby-container -i -t

# Get output from the first pod of a replica set named nginx
oc attach rs/nginx
```

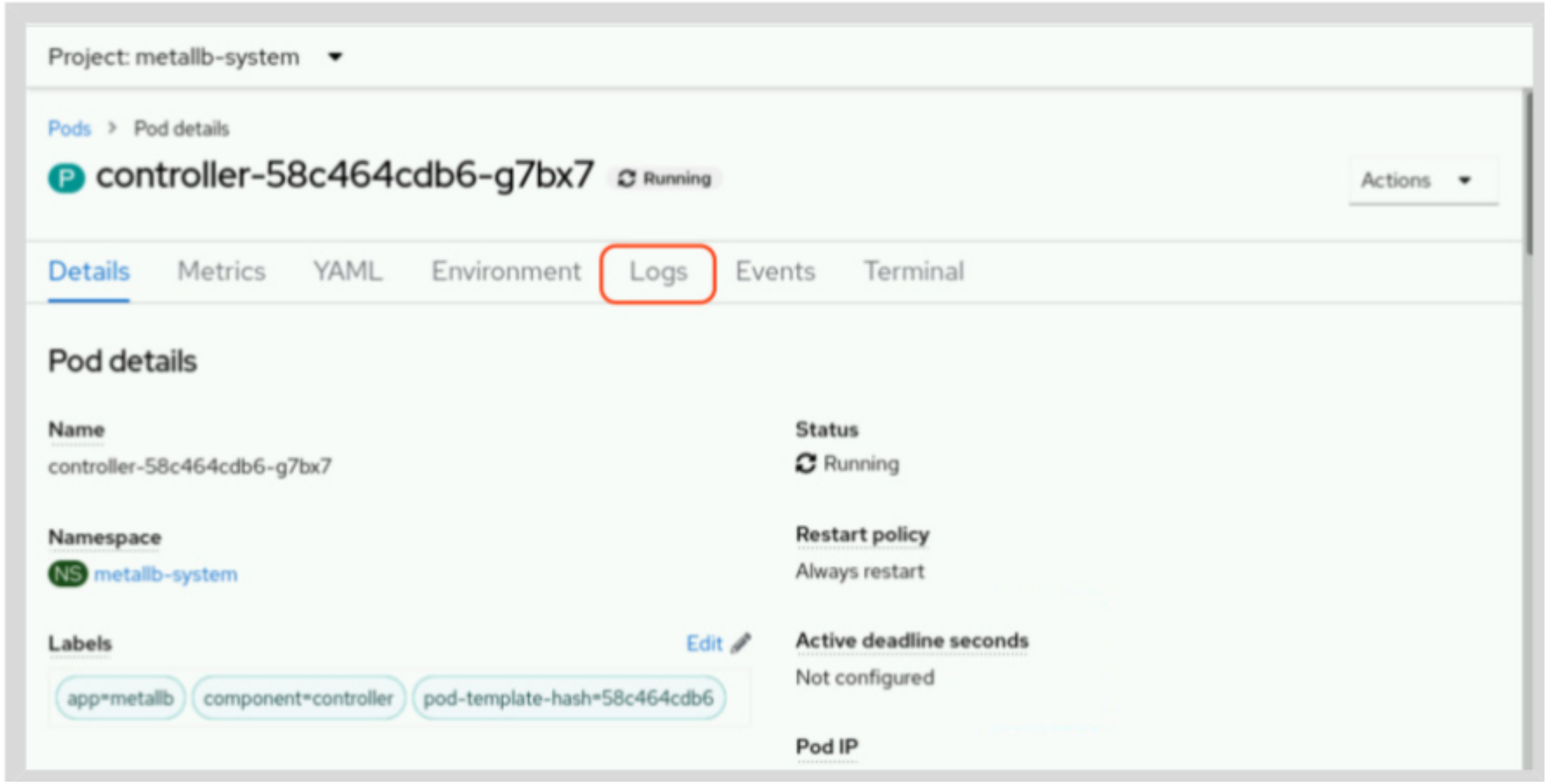
Access Container Logs using command

- Standard output (stdout) and standard error (stderr)
- Retrieve log using **oc logs** command, use **--tail=10** to limit lines of output

```
[user@host ~]$ oc logs postgresql-1-jw89j --tail=10
 done
server stopped
Starting server...
2023-01-04 22:00:16.945 UTC [1] LOG:  starting PostgreSQL 12.11 on x86_64-redhat-
linux-gnu, compiled by gcc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-10), 64-bit
2023-01-04 22:00:16.946 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port
 5432
2023-01-04 22:00:16.946 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2023-01-04 22:00:16.953 UTC [1] LOG:  listening on Unix socket "/var/run/
postgresql/.s.PGSQL.5432"
2023-01-04 22:00:16.960 UTC [1] LOG:  listening on Unix socket "/
tmp/.s.PGSQL.5432"
2023-01-04 22:00:16.968 UTC [1] LOG:  redirecting log output to logging collector
 process
2023-01-04 22:00:16.968 UTC [1] HINT:  Future log output will appear in directory
 "log".
```

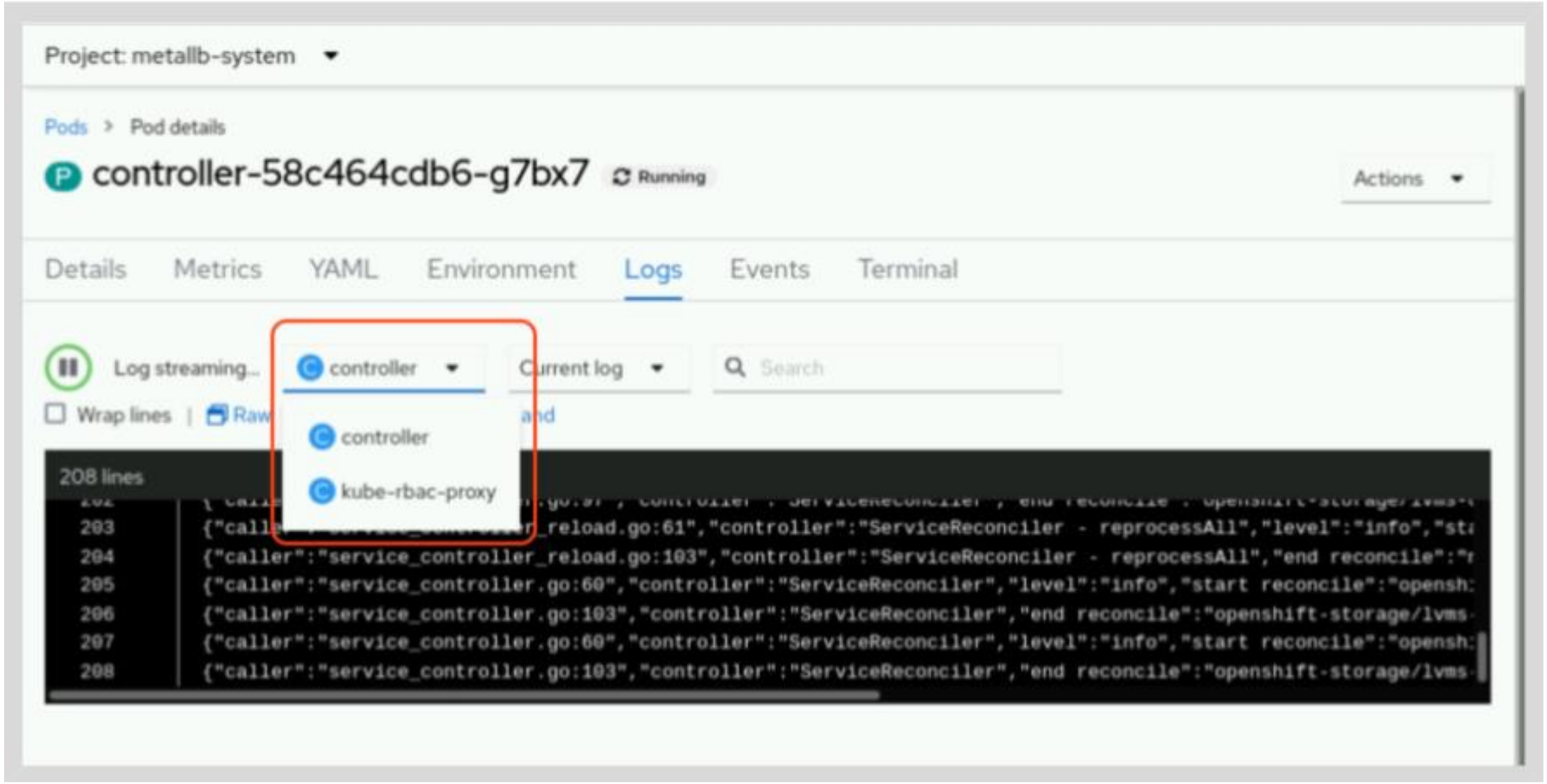
Access Container Logs using web console

- Access Pod then click on Logs tab



Access Container Logs using web console

- If there are multiple container, choose from drop down list
- Perform search, pause the stream



Deleting Resources

- Delete by resource type

```
[user@host ~]$ oc delete pod php-app
```

- Delete by label

```
[user@host ~]$ kubectl delete pod -l app=my-app
pod "php-app" deleted
pod "mysql-db" deleted
```

- Delete by resource type and a given json file

```
[user@host ~]$ oc delete pod -f ~/php-app.json
pod "php-app" deleted
```

```
[user@host ~]$ cat ~/php-app.json | kubectl delete -f -
pod "php-app" deleted
```

Deleting Resources

- Specify graceful period

```
[user@host ~]$ oc delete pod php-app --grace-period=10
```

- To shutdown pod immediately

```
[user@host ~]$ oc delete pod php-app --grace-period=10
```

- or use --now option

```
[user@host ~]$ oc delete pod php-app --now
```

- Forcibly delete pod

```
[user@host ~]$ kubectl delete pod php-app --force
```

- Delete all pods in project/namespace

```
[user@host ~]$ kubectl delete pods --all
pod "php-app" deleted
pod "mysql-db" deleted
```

Deleting Project

- Will remove all resources within project

```
[user@host ~]$ oc delete project my-app
project.project.openshift.io "my-app" deleted
```

- --timeout =#s : specify length of time to wait before giving up on delete.
- --now : resources are signaled for immediate shutdown
- --force : no graceful period. Immediate deletion of some resources and may result in inconsistency or data lose

The CRI-O Container Engine

- is required to run containers
- Used in Worker and Control plane nodes in OpenShift
- Designed and optimized to run containers in Kubernetes cluster
- Meets Kubernetes CRI standards
  - can integrate resources with each other
  - such as networking and storage plug-ins

CRI-O management command

`crictl pods`
Lists all pods on a node.

`crictl image`
Lists all images on a node.

`crictl inspect`
Retrieve the status of one or more containers.

`crictl exec`
Run a command in a running container.

`crictl logs`
Retrieve the logs of a container.

`crictl ps`
List running containers on a node.

CRI-O management command - setup

- Find out which node did the pod runs

```
[user@host ~]$ kubectl get pods -o wide
NAME                READY STATUS    RESTARTS AGE IP         NODE
postgresql-1-8lzf2  1/1   Running   0        20m 10.8.0.64 master01
postgresql-1-deploy 0/1   Completed 0        21m 10.8.0.63 master01
```

- Create debug pod to access the node

```
[user@host ~]$ oc debug node/worker01
Starting pod/worker01-debug ...
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-4.4# chroot /host
```

CRI-O management command - diagnose

- Find the container ID

```
sh-4.4# crictl ps --name postgresql
CONTAINER     IMAGE         CREATED STATE   NAME       ATTEMPT POD ID      POD
27943ae4f3024 image...7104 5...ago Running postgresql 0       5768...f015
 postgresql-1...

sh-4.4# crictl ps --name postgresql -o json | jq .containers[0].id
"2794...29a4"
```

- Retrieve PID of running container

```
sh-4.4# crictl inspect -o json 27943ae4f3024 | jq .info.pid
43453
sh-4.4# crictl inspect 27943ae4f3024 | grep pid
    "pid": 43453,
...output omitted...
```

CRI-O management command - diagnose

- List the system namespaces of the container

```
sh-4.4# lsns -p 43453
        NS TYPE   NPROCS   PID USER       COMMAND
4026531835 cgroup    530     1 root       /usr/lib/systemd/systemd --switched-root
 --system --deserialize 17
4026531837 user      530     1 root       /usr/lib/systemd/systemd --switched-root
 --system --deserialize 17
4026537853 uts         8 43453 1000690000 postgres
4026537854 ipc         8 43453 1000690000 postgres
4026537856 net         8 43453 1000690000 postgres
4026538013 mnt         8 43453 1000690000 postgres
4026538014 pid         8 43453 1000690000 postgres
```

Troubleshooting a latency-sensitive application in container - use of nsenter

- Use nsenter to execute a command within specified namespace of running container

```
sh-4.4# nsenter -t 43453 -p -r ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
1000690+       1       0  0 18:49 ?        00:00:00 postgres
1000690+      58       1  0 18:49 ?        00:00:00 postgres: logger
1000690+      60       1  0 18:49 ?        00:00:00 postgres: checkpointer
1000690+      61       1  0 18:49 ?        00:00:00 postgres: background writer
1000690+      62       1  0 18:49 ?        00:00:00 postgres: walwriter
1000690+      63       1  0 18:49 ?        00:00:00 postgres: autovacuum launcher
1000690+      64       1  0 18:49 ?        00:00:00 postgres: stats collector
1000690+      65       1  0 18:49 ?        00:00:00 postgres: logical replication
 launcher
root        7414       0  0 20:14 ?        00:00:00 ps -ef
```

Troubleshooting a latency-sensitive application in container - use of nsenter

- Use of -a option to execute a command in all the container's namespaces

```
sh-4.4# nsenter -t 43453 -a ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
1000690+       1       0  0 18:49 ?        00:00:00 postgres
1000690+      58       1  0 18:49 ?        00:00:00 postgres: logger
1000690+      60       1  0 18:49 ?        00:00:00 postgres: checkpointer
1000690+      61       1  0 18:49 ?        00:00:00 postgres: background writer
1000690+      62       1  0 18:49 ?        00:00:00 postgres: walwriter
1000690+      63       1  0 18:49 ?        00:00:00 postgres: autovacuum launcher
1000690+      64       1  0 18:49 ?        00:00:00 postgres: stats collector
1000690+      65       1  0 18:49 ?        00:00:00 postgres: logical replication
 launcher
root       10058       0  0 20:45 ?        00:00:00 ps -ef
```

Guided Exercise: Create Linux Containers and Kubernetes Pods

You should be able to:

- Create a pod
- View the logs of a running container.
- Retrieve information inside a container,
- Identify the process ID (PID) and namespaces for a container.
- Identify the User ID (UID) and supplemental group ID (GID) ranges of a project.
- Compare the namespaces of containers in one pod versus in another pod.
- Inspect a pod with multiple containers, and identify the purpose of each container.

In this chapter, you learned:

The OpenShift web console provides a GUI for visualizing and managing OpenShift resources.

Some resources feature a specialized page that makes creating and editing resources more convenient than writing YAML by hand, such as the Edit Key/Value Secret editor, which automatically handles Base64 encoding and decoding.

You can install partner and community operators from the embedded OperatorHub page.

Cluster-wide metrics such as CPU, memory, and storage usage are displayed on the Dashboards page.

Project Details pages display metrics specific to the project, such as the top ten memory consumers by pod and the current resource quota usage.