# Guided Exercise: Create Linux Containers and Kubernetes Pods

Run a base OS container in a pod and compare the environment inside the container with its host node.

**Outcomes**

- Create a pod with a single container, and identify the pod and its container within the container engine of an OpenShift node.
- View the logs of a running container.
- Retrieve information inside a container, such as the operating system (OS) release and running processes.
- 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.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise.

```
[student@workstation ~]$ lab start pods-containers
```

**Instructions**

1. Log in to the OpenShift cluster and create the `pods-containers` project. Determine the UID and GID ranges for pods in the `pods-containers` project.

   Log in to the OpenShift cluster as the `developer` user with the `oc` command.

   ```
   [student@workstation ~]$ oc login -u developer -p developer \
     https://api.ocp4.example.com:6443
   Login successful
   ...output omitted...
   ```

   Create the `pods-containers` project.

   ```
   [student@workstation ~]$ oc new-project pods-containers
   Now using project "pods-containers" on server "https://api.ocp4.example.com:6443".
   ...output omitted...
   ```

   Identify the UID and GID ranges for pods in the `pods-containers` project.

   ```
   [student@workstation ~]$ oc describe project pods-containers
   Name:               pods-containers
   Created:            16 seconds ago
   Labels:             kubernetes.io/metadata.name=pods-containers
                       pod-security.kubernetes.io/audit=restricted
                       pod-security.kubernetes.io/audit-version=latest
                       pod-security.kubernetes.io/warn=restricted
                       pod-security.kubernetes.io/warn-version=latest
   Annotations:        openshift.io/description=
                       openshift.io/display-name=
                       openshift.io/requester=developer
                       openshift.io/sa.scc.mcs=s0:c28,c2
                       openshift.io/sa.scc.supplemental-groups=1000760000/10000
                       openshift.io/sa.scc.uid-range=1000760000/10000
   Display Name:       <none>
   Description:        <none>
   Status:             Active
   Node Selector:      <none>
   Quota:              <none>
   Resource limits:    <none>
   ```

   Your UID and GID range values might differ from the previous output.

2. As the `developer` user, create a pod called `ubi9-user` from a UBI9 base container image. The image is available in the `registry.ocp4.example.com:8443/ubi9/ubi` container registry. Set the restart policy to `Never` and start an interactive session. Configure the pod to execute the `whoami` and `id` commands to determine the UIDs, supplemental groups, and GIDs of the container user in the pod. Delete the pod afterward.

After the `ubi-user` pod is deleted, log in as the `admin` user and then re-create the `ubi9-user` pod. Retrieve the UIDs and GIDs of the container user. Compare the values to the values of the `ubi9-user` pod that the `developer` user created.

Afterward, delete the `ubi9-user` pod.

Use the `oc run` command to create the `ubi9-user` pod. Configure the pod to execute the `whoami` and `id` commands through an interactive bash shell session.

```
[student@workstation ~]$ oc run -it ubi9-user --restart 'Never' \
  --image registry.ocp4.example.com:8443/ubi9/ubi \
  -- /bin/bash -c "whoami && id"
1000760000
uid=1000760000(1000760000) gid=0(root) groups=0(root),1000760000
```

Your values might differ from the previous output.

Notice that the user in the container has the same UID that is identified in the `pods-containers` project. However, the GID of the user in the container is `0`, which means that the user belongs to the `root` group. Any files and directories that the container processes might write to must have read and write permissions by `GID=0` and have the `root` group as the owner.

Although the user in the container belongs to the `root` group, a UID value over `1000` means that the user is an unprivileged account. When a regular OpenShift user, such as the `developer` user, creates a pod, the containers within the pod run as unprivileged accounts.

Delete the pod.

```
[student@workstation ~]$ oc delete pod ubi9-user
pod "ubi9-user" deleted
```

Log in as the `admin` user with the `redhatocp` password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
Login successful.

You have access to 71 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "pods-containers".
```

Re-create the `ubi9-user` pod as the `admin` user. Configure the pod to execute the `whoami` and `id` commands through an interactive bash shell session. Compare the values of the UID and GID for the container user to the values of the `ubi9-user` pod that the `developer` user created.

> **NOTE**
>
> It is safe to ignore pod security warnings when using a cluster-admin user that creates unmanaged pods. The `admin` user can create priviliged pods that the Security Context Constraints controller does not manage.

```
[student@workstation ~]$  oc run -it ubi9-user --restart 'Never' \
  --image registry.ocp4.example.com:8443/ubi9/ubi \
  -- /bin/bash -c "whoami && id"
Warning: would violate PodSecurity "restricted:v1.24": allowPrivilegeEscalation != false (container "ubi9-user" must set s
ecurityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "ubi9-user" must set securityContext.
capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "ubi9-user" must set securityContext.runAsNonRoot=tru
e), seccompProfile (pod or container "ubi9-user" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Loca
lhost")
root
uid=0(root) gid=0(root) groups=0(root)
```

Notice that the value of the UID is `0`, which differs from the UID range value of the `pod-containers` project. The user in the container is the privileged account `root` user and belongs to the `root` group. When a cluster administrator creates a pod, the containers within the pod run as a privileged account by default.

Delete the `ubi9-user` pod.

```
[student@workstation ~]$ oc delete pod ubi9-user
pod "ubi9-user" deleted
```

3. As the `developer` user, use the `oc run` command to create a `ubi9-date` pod from a UBI9 base container image. The image is available in the `registry.ocp4.example.com:8443/ubi9/ubi` container registry. Set the restart policy to `Never`, and configure the pod to execute the `date` command. Retrieve the logs of the `ubi9-date` pod to confirm that the `date` command executed. Delete the pod afterward.

   Log in as the `developer` user with the `developer` password.

   ```
   [student@workstation ~]$ oc login -u developer -p developer
   Login successful.

   You have one project on this server: "pods-containers"

   Using project "pods-containers".
   ```

   Create a pod called `ubi9-date` that executes the `date` command.

   ```
   [student@workstation ~]$ oc run ubi9-date --restart 'Never' \
     --image registry.ocp4.example.com:8443/ubi9/ubi -- date
   pod/ubi9-date created
   ```

   Wait a few moments for the creation of the pod. Then, retrieve the logs of the `ubi9-date` pod.

   ```
   [student@workstation ~]$ oc logs ubi9-date
   Mon Nov 28 15:02:55 UTC 2022
   ```

   Delete the `ubi9-date` pod.

   ```
   [student@workstation ~]$ oc delete pod ubi9-date
   pod "ubi9-date" deleted
   ```

4. Use the `oc run ubi9-command -it` command to create a `ubi9-command` pod with the `registry.ocp4.example.com:8443/ubi9/ubi` container image. Add the `/bin/bash` in the `oc run` command to start an interactive shell. Exit the pods and view the logs for the `ubi9-command` pod with the `oc logs` command. Then, connect to the `ubi9-command` pod with the `oc attach` command, and issue the following command:

   ```
   while true; do echo $(date); sleep 2; done
   ```

   This command executes the `date` and `sleep` commands to generate output to the console every two seconds. Use the `oc logs` command to retrieve the logs of the `ubi9` pod, and confirm that the logs display the executed `date` and `sleep` commands.

   Create a pod called `ubi9-command` and start an interactive shell.

   ```
   [student@workstation ~]$ oc run ubi9-command -it \
     --image registry.ocp4.example.com:8443/ubi9/ubi -- /bin/bash
   If you don't see a command prompt, try pressing enter.
   bash-5.1$
   ```

   Exit the shell session.

   ```
   bash-5.1$ exit
   exit
   Session ended, resume using 'oc attach ubi9-command -c ubi9-command -i -t' command when the pod is running
   ```

   Use the `oc logs` command to view the logs of the `ubi9-command` pod.

   ```
   [student@workstation ~]$ oc logs ubi9-command
   bash-5.1$ [student@workstation ~]$
   ```

   The pod's command prompt is returned. The `oc logs` command displays the pod's current `stdout` and `stderr` output in the console. Because you disconnected from the interactive session, the pod's current `stdout` is the command prompt, and not the commands that you executed previously.

   Use the `oc attach` command to connect to the `ubi9-command` pod again. In the shell, execute the `while true; do echo $(date); sleep 2; done` command to continuously generate `stdout` output.

   ```
   [student@workstation ~]$ oc attach ubi9-command -it
   If you don't see a command prompt, try pressing enter.
   ```

```
bash-5.1$ while true; do echo $(date); sleep 2; done
Mon Nov 28 15:15:16 UTC 2022
Mon Nov 28 15:15:18 UTC 2022
Mon Nov 28 15:15:20 UTC 2022
Mon Nov 28 15:15:22 UTC 2022
...output omitted...
```

Open another terminal window and view the logs for the `ubi9-command` pod with the `oc logs` command. Limit the log output to the last 10 entries with the `--tail` option. Confirm that the logs display the results of the command that you executed in the container.

```
[student@workstation ~]$ oc logs ubi9-command --tail=10
Mon Nov 28 15:15:16 UTC 2022
Mon Nov 28 15:15:18 UTC 2022
Mon Nov 28 15:15:20 UTC 2022
Mon Nov 28 15:15:22 UTC 2022
Mon Nov 28 15:15:24 UTC 2022
Mon Nov 28 15:15:26 UTC 2022
Mon Nov 28 15:15:28 UTC 2022
Mon Nov 28 15:15:30 UTC 2022
Mon Nov 28 15:15:32 UTC 2022
Mon Nov 28 15:15:34 UTC 2022
```

5. Identify the name for the container in the `ubi9-command` pod. Identify the process ID (PID) for the container in the `ubi9-command` pod by using a debug pod for the pod's host node. Use the `crictl` command to identify the PID of the container in the `ubi9-command` pod. Then, retrieve the PID of the container in the debug pod.

   Identify the container name in the `ubi9-command` pod with the `oc get` command. Specify the JSON format for the command output. Parse the JSON output with the `jq` command to retrieve the value of the `.status.containerStatuses[].name` object.

   ```
   [student@workstation ~]$ oc get pod ubi9-command -o json | \
     jq .status.containerStatuses[].name
   "ubi9-command"
   ```

   The `ubi9-command` pod has a single container of the same name.

   Find the host node for the `ubi9-command` pod. Start a debug pod for the host with the `oc debug` command.

   ```
   [student@workstation ~]$ oc get pods ubi9-command -o wide
   NAME           READY STATUS  RESTARTS    AGE  IP         NODE     ...
   ubi9-command   1/1   Running 2 (16m ago) 27m  10.8.0.26  master01 ...
   ```

   ```
   [student@workstation ~]$ oc debug node/master01
   Error from server (Forbidden): nodes "master01" is forbidden: User "developer" cannot get resource "nodes" in API group ""
   at the cluster scope
   ```

   The debug pod fails because the `developer` user does not have the required permission to debug a host node.

   Log in as the `admin` user with the `redhatocp` password. Start a debug pod for the host with the `oc debug` command. After connecting to the debug pod, run the `chroot /host` command to use host binaries, such as the `crictl` command-line tool.

   ```
   [student@workstation ~]$ oc login -u admin -p redhatocp
   Login successful.
   ...output omitted...
   ```

   ```
   [student@workstation ~]$ oc debug node/master01
   Starting pod/master01-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
   ```

   Use the `crictl ps` command to retrieve the `ubi9-command` container ID. Specify the `ubi9-command` container with the `--name` option and use the JSON output format. Parse the JSON output with the `jq -r` command to get the RAW JSON output. Export the container ID as the `$CID` environment variable.

   > **NOTE**
   >
   > When using `jq` without the `-r` flag, the container ID is wrapped in double quotes, which does not work with `crictl` commands. If the `-r` flag is not used, then you can add `| tr -d '"'` to the end of the

command to trim the double quotes.

```
sh-5.1# crictl ps --name ubi9-command -o json | jq -r .containers[0].id
81adbc6222d79ed9ba195af4e9d36309c18bb71bc04b2e8b5612be632220e0d6
```

```
sh-5.1# CID=$(crictl ps --name ubi9-command -o json | jq -r .containers[0].id)
```

```
sh-5.1# echo $CID
81adbc6222d79ed9ba195af4e9d36309c18bb71bc04b2e8b5612be632220e0d6
```

Your container ID value might differ from the previous output.

Use the `crictl inspect` command to find the PID of the `ubi9-command` container. The PID value is in the `.info.pid` object in the `crictl inspect` output. Export the `ubi9-command` container PID as the `$PID` environment variable.

```
sh-5.1# crictl inspect $CID | grep pid
    "pid": 365297,
          "pids": {
            "type": "pid"
...output omitted...
          }
...output omitted...
```

```
sh-5.1# PID=365297
```

Your PID values might differ from the previous output.

6. Use the `lsns` command to list the system namespaces of the `ubi9-command` container. Confirm that the running processes in the container are isolated to different system namespaces.

   View the system namespaces of the `ubi9-command` container with the `lsns` command. Specify the PID with the `-p` option and use the `$PID` environment variable. In the resulting table, the `NS` column contains the namespace values for the container.

   ```
   sh-5.1# lsns -p $PID
           NS TYPE   NPROCS    PID USER       COMMAND
   4026531835 cgroup    540      1 root       /usr/lib/systemd/systemd --swit ...
   4026531837 user      540      1 root       /usr/lib/systemd/systemd --swit ...
   4026536117 uts         1 153168 1000800000 /bin/bash
   4026536118 ipc         1 153168 1000800000 /bin/bash
   4026536120 net         1 153168 1000800000 /bin/bash
   4026537680 mnt         1 153168 1000800000 /bin/bash
   4026537823 pid         1 153168 1000800000 /bin/bash
   ```

   Your namespace values might differ from the previous output.

7. Use the host debug pod to retrieve and compare the operating system (OS) and the GCC support library (`libgcc`) package version of the `ubi9-command` container and the host node.

   Retrieve the OS for the host node with the `cat /etc/redhat-release` command.

   ```
   sh-5.1# cat /etc/redhat-release
   Red Hat Enterprise Linux CoreOS release 4.18
   ```

   Use the `crictl exec` command and the `$CID` container ID variable to retrieve the OS of the `ubi9-command` container. Use the `-it` options to create an interactive terminal to execute the `cat /etc/redhat-release` command.

   ```
   sh-5.1# crictl exec -it $CID cat /etc/redhat-release
   Red Hat Enterprise Linux release 9.6 (Plow)
   ```

   The `ubi9-command` container has a different OS from the host node.

   Use the `rpm -qi libgcc` command to retrieve the `libgcc` package version of the host node.

   ```
   sh-5.1$ rpm -qi libgcc
   Name: libgcc
   Version: 11.4.1
   ...output omitted...
   ```

Use the `crictl exec` command and the `$CID` container ID variable to retrieve the `glibc` package version of the `ubi9-command` container. Use the `-it` options to create an interactive terminal to execute the `rpm -qi libgcc` command.

```
sh-5.1# crictl exec -it $CID rpm -qi libgcc
Name: libgcc
Version: 11.5.0
...output omitted...
```

The `ubi9-command` container might have a different version of the `libgcc` package from its host.

8. Exit the `master01-debug` pod and the `ubi9-command` pod.

   Exit the `master01-debug` pod. You must issue the `exit` command to end the host binary access. Execute the `exit` command again to exit and remove the `master01-debug` pod.

   ```
   sh-5.1# exit
   exit
   ```

   ```
   sh-4.4# exit
   exit

   Removing debug pod ...
   Temporary namespace openshift-debug-bg7kn was removed.
   ```

   Return to the terminal window that is connected to the `ubi9-command` pod. Press **Ctrl**+**C** and then execute the `exit` command. Confirm that the pod is still running.

   ```
   ...output omitted...
   ^C
   bash-5.1$ exit
   exit
   Session ended, resume using 'oc attach ubi9-command -c ubi9-command -i -t' command when the pod is running
   ```

   ```
   [student@workstation ~]$ oc get pods
   NAME           READY   STATUS    RESTARTS     AGE
   ubi9-command   1/1     Running   2 (6s ago)   35m
   ```

### Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish pods-containers
```