

[Menu](#)[Browse Library](#) ▾

Training Library / Mastering Kubernetes Pod Configuration: Security Contexts

# Configuring Pod Security Contexts

37m 51s left

## Open Cloud Environment

✓  
100%

Setup completed  
Average setup time: 2m 28s

## Open Code Environment

✓  
100%

Setup completed

## Credentials

Account ID ⓘ

632743795 Copy

Username ⓘ

student Copy

Password ⓘ

Ca1\_eIUh7 Copy

Region ⓘ

US West 2 Copy

PEM ⓘ

PPK ⓘ

Download Download

Bastion Public Ip ⓘ

54.245.18 Copy

Cluster SSH ⓘ

ssh ubuntu

## Introduction

A security context allows you to set access control for Pods, as well as containers and volumes in Pods, when applicable. Examples of access controls that can be set with security contexts include:

- The user ID and group IDs of the first process running in a container
- The group ID of volumes
- If a container's root file system is read-only
- Security-Enhanced Linux (SELinux) options
- The privileged status of containers, which allows the container to do almost everything root can do on the host if enabled
- Whether or not privilege escalation, where child processes can have more privileges than their parent, is allowed

When a security context field is configured for a Pod and one of the Pod's containers, the container's setting takes precedence. Configuring the security context of Pods and containers can greatly reduce the security risk posed by using third-party images.

In this lab step, you will review the available options for configuring security contexts. You will then create multiple Pods with differing security contexts to observe the effects of each.

## Instructions

1. Explain the available Pod-level security context fields:

Copy code

```
1 | kubectl explain pod.spec.securityContext | more
```

Support

[Skip to content](#)

Press + to open this menu



## Lab Steps

1

Connecting to the  
Kubernetes Cluster

2

Configuring Pod  
Security Contexts

Need help? Contact our  
support team

owned by the pod:

1. The owning GID will be the FSGroup 2. The setgid bit is set (new files created in the volume will be owned by FSGroup) 3. The permission bits are OR'd with rw-rw----

If unset, the Kubelet will not modify the ownership and permissions of any volume. Note that this field cannot be set when spec.os.name is windows.

**fsGroupChangePolicy** <string>

fsGroupChangePolicy defines behavior of changing ownership and permission of the volume before being exposed inside Pod. This field will only apply to volume types which support fsGroup based ownership (and permissions). It will have no effect on ephemeral volume types such as: secret, configmaps and emptydir. Valid values are "OnRootMismatch" and "Always". If not specified, "Always" is used. Note that this field cannot be set when spec.os.name is windows.

Possible enum values:

- "Always" indicates that volume's ownership and permissions should always be changed whenever volume is mounted inside a Pod. This the default behavior.
- "OnRootMismatch" indicates that volume's ownership and permissions will be changed only when permission and ownership of root directory does not match with expected permissions on the volume. This can help shorten the time it takes to change ownership and permissions of a volume.

**runAsGroup** <integer>

The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in SecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence for that container. Note that this field cannot be set when spec.os.name is windows.

Briefly read through each of the fields to get an understanding of what each can be used for.

2. Explain the available container-level security context fields:

 Copy code

```
1 | kubectl explain pod.spec.containers.securityContext | more
```



no\_new\_privs flag will be set on the container process.

AllowPrivilegeEscalation is true always when the container is: 1) run as Privileged 2) has CAP\_SYS\_ADMIN Note that this field cannot be set when spec.os.name is windows.

capabilities <Object>

The capabilities to add/drop when running containers. Defaults to the default set of capabilities granted by the container runtime. Note that this field cannot be set when spec.os.name is windows.

privileged <boolean>

Run container in privileged mode. Processes in privileged containers are essentially equivalent to root on the host. Defaults to false. Note that this field cannot be set when spec.os.name is windows.

procMount <string>

procMount denotes the type of proc mount to use for the containers. The default is DefaultProcMount which uses the container runtime defaults for readonly paths and masked paths. This requires the ProcMountType feature flag to be enabled. Note that this field cannot be set when spec.os.name is windows.

Possible enum values:

- "Default" uses the container runtime defaults for readonly and masked paths for /proc. Most container runtimes mask certain paths in /proc to avoid accidental security exposure of special devices or information.
- "Unmasked" bypasses the default masking behavior of the container runtime and ensures the newly created /proc the container stays in tact with no modifications.

readOnlyRootFilesystem <boolean>

Whether this container has a read-only root filesystem. Default is false. Note that this field cannot be set when spec.os.name is windows.

runAsGroup <integer>

The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

Briefly read through each of the fields to get an understanding of what each can be used for.

3. Create the following Pod manifest file:

 Copy code

```
1 cat << EOF > pod-no-security-context.yaml
2 apiVersion: v1
3 kind: Pod
4 metadata:
5   name: security-context-test-1
6 spec:
7   containers:
8     - image: busybox:1.30.1
9       name: busybox
10      args:
11        - sleep
12        - "3600"
13 EOF
```

The pod simply runs a container that sleeps.

4. Create the Pod and use exec to list the available devices in the container:

Skip to content

Press  +  to open this menu seconds before

running the command again



```
ubuntu@ip-10-0-128-5:~$  
core  
fd  
full  
mqueue  
null  
ptmx  
pts  
random  
shm  
stderr  
stdin  
stdout  
termination-log  
tty  
urandom  
zero
```

There are only a minimal number of devices available in the container and none that can do any harm. For the sake of what you will do next, notice there are no block devices. In particular, there is no `nvme0n1p1` device that is the host's file system disk.

5. Delete the previous Pod and create a similar Pod that has a privileged container:

[Copy code](#)

```
1  kubectl delete -f pod-no-security-context.yaml  
2  cat > pod-privileged.yaml <<EOF  
3  apiVersion: v1  
4  kind: Pod  
5  metadata:  
6    name: security-context-test-2  
7  spec:  
8    containers:  
9    - image: busybox:1.30.1  
10      name: busybox  
11      args:  
12      - sleep  
13      - "3600"  
14      securityContext:  
15        privileged: true  
16  EOF  
17  kubectl create -f pod-privileged.yaml
```

Note the `securityContext` field included in the spec.

6. List the devices available in the container:

[Copy code](#)

```
1  kubectl exec security-context-test-2 -- ls /dev
```



```
cpu_dma_latency
cuse
dma_heap
ecryptfs
fd
full
fuse
hpet
hwrng
input
kmsg
loop-control
loop0
loop1
loop2
loop3
loop4
loop5
loop6
loop7
mapper
mcelog
```

All of the host devices are available including the host file system disk **nvme0n1p1**. This could be a major security breach and shows the importance of carefully considering if you should ever use a privileged container.

8. Create another pod that includes a Pod security context as well as a container security context:

[Copy code](#)

```
1  kubectl delete -f pod-privileged.yaml
2  cat << EOF > pod-runas.yaml
3  apiVersion: v1
4  kind: Pod
5  metadata:
6    name: security-context-test-3
7  spec:
8    securityContext:
9      runAsNonRoot: true
10     runAsUser: 1000
11     runAsGroup: 1000
12   containers:
13   - image: busybox:1.30.1
14     name: busybox
15     args:
16     - sleep
17     - "3600"
18     securityContext:
19       runAsUser: 2000
20       readOnlyRootFilesystem: true
21 EOF
22 kubectl create -f pod-runas.yaml
```

The Pod security context enforces that container processes do not run as root (**runAsNonRoot**) and sets the user ID of the container process to **1000**. The container **securityContext** sets the container process' user ID to **2000** and sets the root file system to read-only.

```
1 | kubectl exec security-context-test-3 -it -- /bin/sh
```

```
/ $
```

Notice that the shell prompt is **\$** and not **#** indicating that you are not the root user.

10. List the running processes in the container:

[Copy code](#)

```
1 | ps
```

```
PID    USER     TIME    COMMAND
   1   2000      0:00   sleep 3600
   5   2000      0:00   /bin/sh
  26   2000      0:00   ps
```

The **USER** ID is 2000 illustrating that it is not root and that the container security context overrides the setting in the Pod security context when both security contexts include the same field. Whenever possible you should not run as root.

11. Attempt to create a file in the /tmp directory:

[Copy code](#)

```
1 | touch /tmp/test-file
2 | exit
```

```
touch: /tmp/test-file: Read-only file system
```

The attempt fails due to the **Read-only file system**. When possible, it is best to use read-only root file systems to harden your container environments. A best practice is to use volumes to mount any files that require modification, allowing the root file system to be read-only.

12. Delete the pod resource:

[Copy code](#)

In this lab step, you understood the use and capabilities of Pod and container security contexts. You also learned about potential risks and how to protect your cluster from them.

Did you like this step?

[✕ End Lab](#)[< Back](#)[Submit](#)

## ABOUT US

[About Cloud Academy](#)[About QA](#)[About Circus Street](#)

## COMMUNITY

[Join Discord Channel](#)

## HELP

[Help Center](#)

Copyright © 2024 Cloud Academy Inc. All rights reserved.

[Terms and Conditions](#)[Privacy Policy](#)[Sitemap](#)[System Status](#)[Manage your cookies](#)