Pod Security Standards

**[Pod Security Concepts](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss" \l "pod-security-concepts)**

When considering Pod Security, there are several settings that can be used to control what containers, within pods, can do.

* Linux Capabilities
* Security Context
* Seccomp profiles
* AppArmor profiles
* Host Access (Network, PID)

**[Kubernetes Security Context Elements](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss" \l "kubernetes-security-context-elements)**

Security Context Elements are applied to Kubernetes pods and containers to apply specific security settings. They can be applied at pod and container levels. Container-level Security Context elements are more granular than there pod counterparts, and override pod-level settings.

A example Pod *Security Context* can be seen below. The settings therein apply to every container within the pod, unless overridden by a container-level *Security Context*.

securityContext:

runAsUser: 1000

runAsGroup: 3000

fsGroup: 2000

fsGroupChangePolicy: "OnRootMismatch"

The following is a container-level *Security Context* element. As you can see, it is more granular.

*This example container Security Context Element follows the official*[Kubernetes documentation](https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-a-container)*.*

securityContext:

allowPrivilegeEscalation: false

runAsUser: 1000

readOnlyRootFilesystem: true

runAsNonRoot: true

capabilities:

drop: ["ALL"]

seccompProfile:

type: "RuntimeDefault"

*This example is based on known good configurations, as part of*[Pod Security Admission](https://kubernetes.io/docs/concepts/security/pod-security-admission/)*and*[Pod Security Standards](https://kubernetes.io/docs/concepts/security/pod-security-standards/)*and*[testing](https://github.com/aws-samples/k8s-psa-pss-testing#testing-setup-and-execution)*performed by the Amazon EKS team.*

**[Pod Security Policies are Deprecated](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss" \l "pod-security-policies-are-deprecated)**

To control pod security, Kubernetes included Pod Security Policy (PSP) resources and admission controllers just prior to Kubernetes version 1.13. PSPs specify a set of security settings that pods must meet before they can be created or updated in a cluster. However, PSPs were deprecated in Kubernetes version 1.21, and they were removed in Kubernetes version 1.25.

The Kubernetes project [documented](https://kubernetes.io/blog/2021/04/06/podsecuritypolicy-deprecation-past-present-and-future/) why PSPs were deprecated. Simply put, PSPs were confusing to the majority of users. This confusion resulted in many misconfigurations; clusters were impaired or left unprotected by overly-restrictive or overly-permissive settings.

How PSPs were applied was not obvious to many users. PSPs lacked certain capabilities that would have made them easier to add to existing clusters while gauging cluster impact, such as dry-run and audit modes.

Finally, because of PSP implementation details, it wasn’t possible to enable PSPs by default. All this precipitated the need for a new, more user-friendly, and deterministic solution for pod security. It also had to remain built-in to Kubernetes.

In the next section we will explore Kubernetes [Pod Security Standards](https://kubernetes.io/docs/concepts/security/pod-security-standards/) and how they are replacing PSPs.

Pod Security Standards(PSS)

**[Introduction](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-standards" \l "introduction)**

In Kubernetes, PSPs are replaced with [Pod Security Admission (PSA)](https://kubernetes.io/docs/concepts/security/pod-security-admission/), a built-in admission controller that implements the security controls outlined in the [Pod Security Standards (PSS)](https://kubernetes.io/docs/concepts/security/pod-security-standards/). PSS was introduced into Kubernetes in 2020, prior to Kubernetes version 1.21. PSA reached a beta state in Kubernetes version 1.23, and was enabled in Amazon EKS version 1.23 by default.

*Kubernetes users can move to PSA and PSS prior to Kubernetes version 1.25, and before they replace PSP; both solutions can coexist in the same cluster. It’s considered a best practice to ease adoption and migration by using PSA/PSS, until PSPs are removed from clusters. For additional guidance on migrating from PSPs to PSA, you should review the*[Kubernetes documentation](https://kubernetes.io/docs/tasks/configure-pod-container/migrate-from-psp/)*on this topic.*

The below *kubectl* snippet can be used to identify pods in clusters that are annotated to use PSP.

1

kubectl get pod -A -o jsonpath='{range .items[?(@.metadata.annotations.kubernetes\.io/psp)]}{.metadata.name}{"\t"}{.metadata.annotations.kubernetes\.io/psp}{"\t"}{.metadata.namespace}{"\n"}'

**[Pod Security Standards (PSS) Profiles](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-standards" \l "pod-security-standards-(pss)-profiles)**

The security settings prescribed by PSS were derived from the experiences of the Kubernetes community with PSP.

According to the Kubernetes PSS documentation, the PSS “define three different profiles to broadly cover the security spectrum. These profiles are cumulative and range from highly-permissive to highly-restrictive.”

The policy levels are defined in the Kubernetes documentation as:

* **Privileged:** Unrestricted policy, providing the widest possible level of permissions. This policy allows for known privilege escalations.
* **Baseline:** Minimally restrictive policy which prevents known privilege escalations. Allows the default (minimally specified) pod configuration.
* **Restricted:** Heavily restricted policy, following current pod hardening best practices.

*The PSS Restricted profile includes the new pod.spec.os.name field. This field is used to enable/disable OS-specific PSS settings. For example, restrictions on the following controls are only required if .spec.os.name is not windows: Privilege Escalation, Seccomp and Linux Capabilities.*

Now that we explored PSS, let's see how we can enforce the related security profiles with the Kubernetes Pod Security Admission (PSA) controller.

Pod Security Admission (PSA)

**[Pod Security Admission (PSA) Modes](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-admision" \l "pod-security-admission-(psa)-modes)**

Pod Security Admission (PSA) went Beta in Kubernetes version 1.23, and consequently became available in Amazon EKS 1.23.

PSA is a Kubernetes in-tree admission controller to enforce:

*"…requirements on a Pod's Security Context and other related fields according to the three levels defined by the Pod Security Standards” – Kubernetes Documentation*

The PSA admission controller implements the controls, outlined by the PSS profiles, via three modes of operation:

* **enforce:** Policy violations will cause the pod to be rejected.
* **audit:** Policy violations trigger the addition of an audit annotation to the event recorded in the audit log, but are otherwise allowed.
* **warn:** Policy violations will trigger a user-facing warning, but are otherwise allowed.

*For Kubernetes (and Amazon EKS) versions prior to 1.23, Kubernetes Dynamic Admission Controller version can be used.*

**[Default PSA and PSS settings](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-admision" \l "default-psa-and-pss-settings)**

The default (cluster-wide) settings for PSA and PSS are seen below.

***Note:****These settings can not be changed (customized) at the Kubernetes API server for Amazon EKS.*

defaults:

enforce: "privileged"

enforce-version: "latest"

audit: "privileged"

audit-version: "latest"

warn: "privileged"

warn-version: "latest"

exemptions:

# Array of authenticated usernames to exempt.

usernames: []

# Array of runtime class names to exempt.

runtimeClasses: []

# Array of namespaces to exempt.

namespaces: []

The above settings configure the following cluster-wide scenario:

* No PSA exemptions are configured at Kubernetes API server startup.
* The Privileged PSS profile is configured by default for all PSA modes, and set to latest versions.
* Namespaces are opted into more restrictive PSS policies via labels.

**[Namespaces opt-in to PSA/PSS settings](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-admision" \l "namespaces-opt-in-to-psapss-settings)**

Given the above default PSA/PSS configuration, you must configure specific PSA modes and PSS profiles at the Kubernetes Namespace level, to opt Namespaces into more restrictive pod security provided by PSA and PSS. In this way, you can configure Namespaces to define the admission control mode you want to use for pod security.

With Kubernetes labels, you can choose which of the predefined PSS levels you want to use for pods in a given Namespace. The labels you select define what action the PSA takes if a potential violation is detected. As seen in the following code, you configure any or all modes, or even set a different level for different modes. For each mode, there are two possible labels that determine the policy used.

# The per-mode level label indicates which policy level to apply for the mode.

#

# MODE must be one of `enforce`, `audit`, or `warn`.

# LEVEL must be one of `privileged`, `baseline`, or `restricted`.

pod-security.kubernetes.io/<MODE>: <LEVEL>

# Optional: per-mode version label that can be used to pin the policy to the

# version that shipped with a given Kubernetes minor version (for example v1.24).

#

# MODE must be one of `enforce`, `audit`, or `warn`.

# VERSION must be a valid Kubernetes minor version, or `latest`.

pod-security.kubernetes.io/<MODE>-version: <VERSION>

Below is an example of PSA and PSS Namespace configurations that can be used for testing.

***Note:****The optional PSA mode-version label is not included. The cluster-wide setting, latest, configured by default, is used. By uncommenting the desired labels (in the following code), you can enable the PSA modes and PSS profiles you need for your respective Namespaces.*

apiVersion: v1

kind: Namespace

metadata:

name: <NAMESPACE\_NAME>

labels:

# pod-security.kubernetes.io/enforce: privileged

# pod-security.kubernetes.io/audit: privileged

# pod-security.kubernetes.io/warn: privileged

# pod-security.kubernetes.io/enforce: baseline

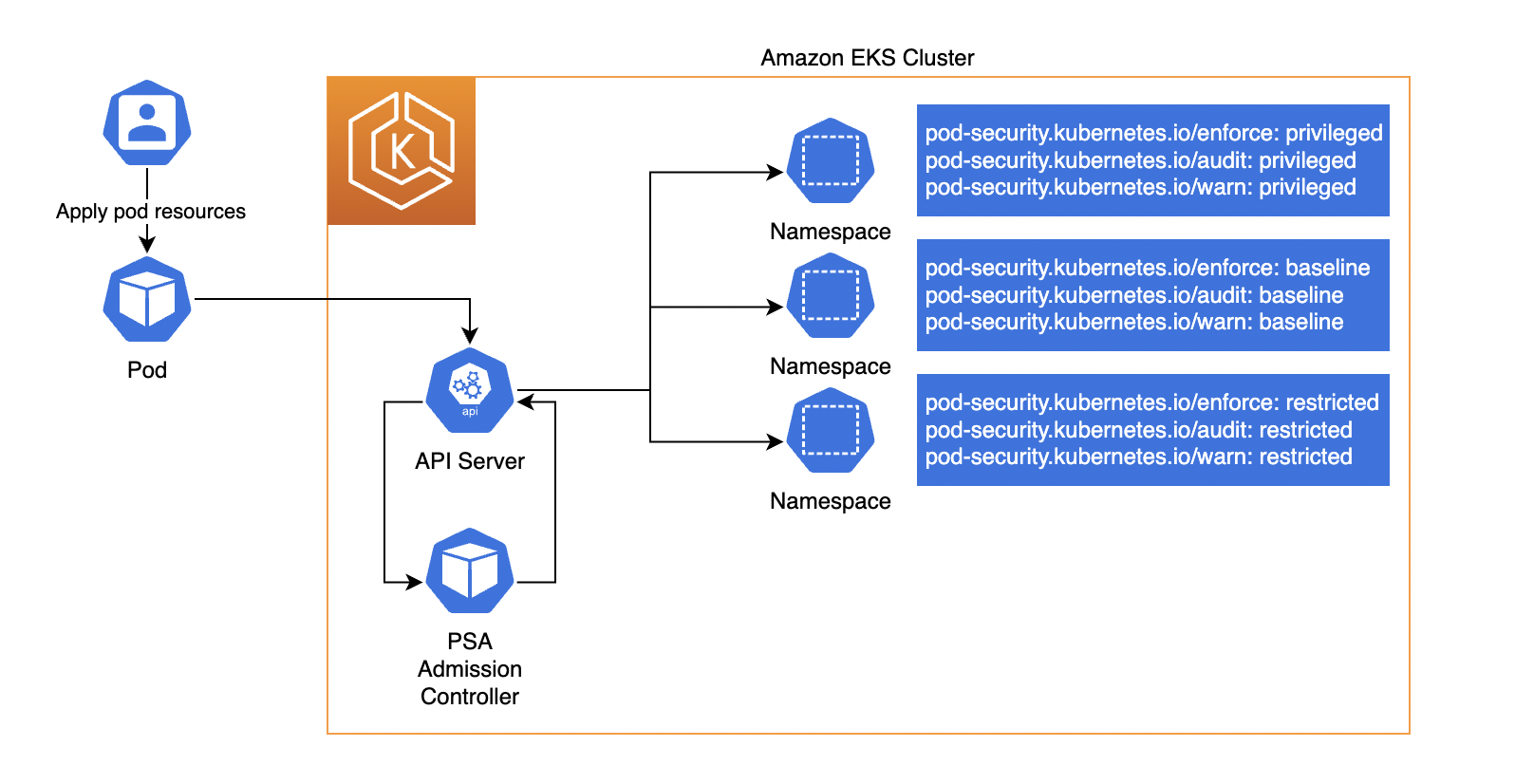
# pod-security.kubernetes.io/audit: baseline

# pod-security.kubernetes.io/warn: baseline

# pod-security.kubernetes.io/enforce: restricted

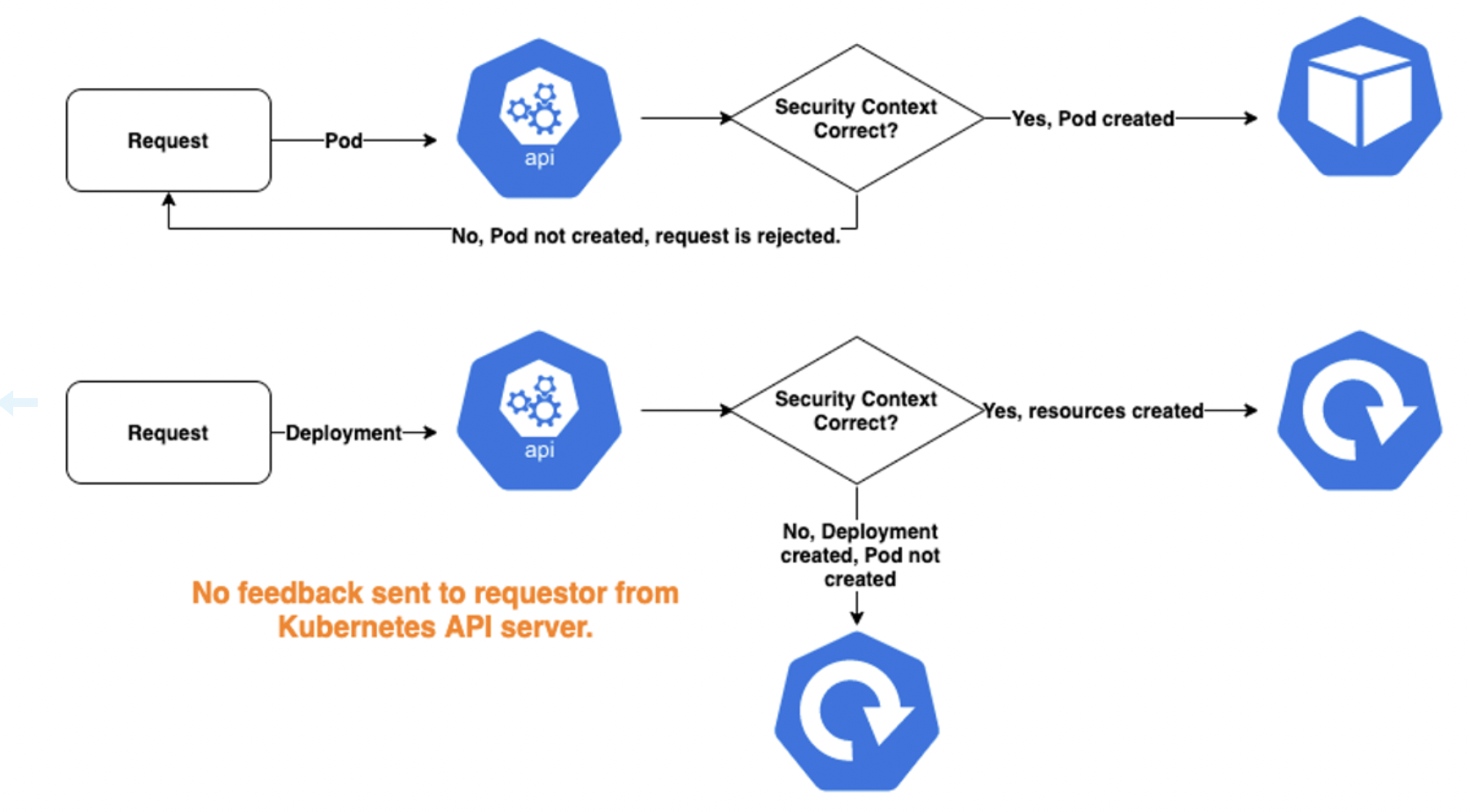
# pod-security.kubernetes.io/audit: restricted

# pod-security.kubernetes.io/warn: restricted



**[PSA enforce mode user experience (UX) issues](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-admision" \l "psa-enforce-mode-user-experience-(ux)-issues)**

When used independently, the PSA modes have different responses that result in different user experiences. The enforce mode prevents Pods from being created if the respective Pod specs violate the configured PSS profile. However, in this mode, non-Pod Kubernetes objects that create Pods, such as Deployments, won’t be prevented from being applied to the cluster, even if the Pod spec therein violates the applied PSS profile. In this case, the Deployment is applied while the Pods are prevented from being applied.



In some scenarios, this is a difficult user experience, as there is no immediate indication that the successfully applied Deployment object belies failed Pod creation. The offending Pod specs won’t create Pods. Inspecting the Deployment resource with kubectl get deployments.apps <DEPLOYMENT\_NAME> -o=jsonpath='{.status}' will expose the message from the failed Pod(s) in the Deployment .status.conditions element.

In both the audit and warn PSA modes, the Pod restrictions don’t prevent violating Pods from being created and started. However, in these modes audit annotations on API server audit log events and warnings to API server clients (e.g., kubectl) are triggered, respectively. This occurs when Pods, as well as objects that create Pods, contain Pod specs with PSS violations. A kubectl Warning message is seen in the following output.

deployment.apps/test created

Warning: would violate PodSecurity "restricted:latest": allowPrivilegeEscalation !=

false (container "test" must set securityContext.allowPrivilegeEscalation=false),

unrestricted capabilities (container "test" must set

securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container

"test" must set securityContext.runAsNonRoot=true), seccompProfile (pod or

container "test" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")

While the Deployment was created, the Pod was not. It’s clear that a best practice would be to use *warn* and *audit* modes at all times, for a better user experience.

**[Mixed PSA modes and PSS profiles](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-admision" \l "mixed-psa-modes-and-pss-profiles)**

PSA modes can be mixed for a customized solution. For example, when you want to measure the impact of new restrictions, before the *enforce* PSA mode is enabled, you could use the following PSA/PSS settings to enforce the *Baseline* PSS profile, yet *audit* and *warn* on the *Restricted* PSS profile.

apiVersion: v1

kind: Namespace

metadata:

name: <NAMESPACE\_NAME>

labels:

pod-security.kubernetes.io/enforce: baseline

pod-security.kubernetes.io/audit: restricted

pod-security.kubernetes.io/warn: restricted

Lab: Amazon EKS Pod Security

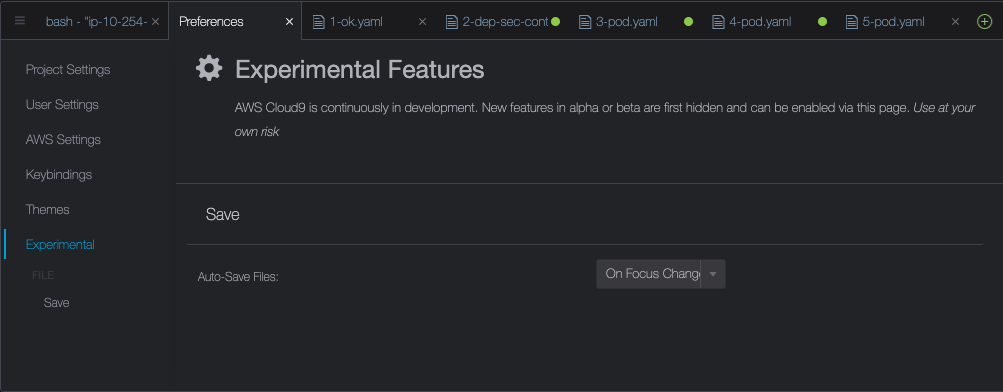
**[Introduction](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-lab" \l "introduction)**

The following hands-on lab is based on the Amazon EKS testing done with PSA and PSS. This testing can be reviewed in the [AWS Samples OSS project on GitHub](https://github.com/aws-samples/k8s-psa-pss-testing).

**[Lab assumptions](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-lab" \l "lab-assumptions)**

Amazon EKS 1.23 (or later version) is provisioned and kubectl CLI is installed and connected to the cluster

In this lab we are going to update many times files in the Cloud9 environment. As Auto-save is not enabled by default, we recommand you activate the option in Cloud9 **On focus Change**



**[Steps](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-lab" \l "steps)**

1. Clone the GitHub repo: <https://github.com/aws-samples/k8s-psa-pss-testing>
2. Review the [Testing Summary](https://github.com/aws-samples/k8s-psa-pss-testing#testing-summary), [Testing Outcomes](https://github.com/aws-samples/k8s-psa-pss-testing#testing-outcomes), [Testing Assumptions](https://github.com/aws-samples/k8s-psa-pss-testing#testing-assumptions), and [Testing Setup and Execution](https://github.com/aws-samples/k8s-psa-pss-testing#testing-setup-and-execution) before getting started.
3. Change directory into the [tests](https://github.com/aws-samples/k8s-psa-pss-testing/tree/main/tests) directory.
4. Modify the policy-test Kubernetes Namespace in the 0-ns.yaml file to set the desired PSA mode and PSS profile settings.
5. Execute the tests.sh script to apply the Kubernetes resources. The outcomes of the tests should match the README.md file.

*Use the*[Testing Scenarios](https://github.com/aws-samples/k8s-psa-pss-testing#testing-scenarios)*to guide your testing.*

1. Between each test run, execute the clean.sh script to delete the resources created by the tests.sh script.
2. Change the policy-test Namespace settings and run the next test scenario.

***Note:****If you have time and are so inclined, feel free to mix PSA modes and PSS profiles to go beyond the documented test scenarios.*

**[Summary](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/1-psa-pss/pod-security-lab" \l "summary)**

PSA and PSS are the native Kubernetes replacement for PSP; moreover, PSA and PSS can coexist with PSP in the same cluster, to facilitate PSP replacement. Testing of PSA and PSS with Amazon EKS 1.23 is documented in this [AWS Samples OSS project](https://github.com/aws-samples/k8s-psa-pss-testing).

The default configurations of PSA and PSS are part of Amazon EKS 1.23+, and Kubernetes Namespaces can be configured with labels to opt into Pod security defined by PSS and implemented by PSA. With appropriate policies you can successfully replace PSP.