In Kubernetes, Pod Security Policies (PSPs) are used to define and enforce security policies for pods running in a cluster. PSPs can be used to control which users or groups are allowed to create or modify pods, which capabilities or volumes a pod can use, and other security-related settings.

PSPs define a set of rules that must be met by a pod's security context in order for the pod to be admitted to the cluster. When a pod is created, the Kubernetes API server checks the PSPs to make sure the pod's security context matches the rules defined in the policies. If the pod's security context violates any of the policies, the pod is rejected and an error message is returned.

PSPs can be used to enforce a wide range of security policies, from basic restrictions on privilege escalation and network access to more advanced policies such as mandatory access control and seccomp profiles.

The example YAML file in a previous question includes annotations that specify pod security policies for the namespace, which are used to enforce and audit certain security policies when pods are created within the namespace. The policies specified in the annotations enforce a baseline level of security for pods created within the namespace and restrict certain privileges, such as host network access and privilege escalation.

In Kubernetes, Pod Security allows you to declare different security profiles for Pods. These security profiles are known as Pod Security Standards and are applied at the namespace level. Pod Security Standards are a collection of security-sensitive fields in a Pod specification (including but not limited to SecurityContext) and their associated values. There are three different standards that range from restricted to permissive. The idea is that you can apply a general security posture to all Pods in a given namespace.

In this lab, you will experience the effects of a Pod Security Standard during Pod creation.

Prerequisite Skills
Comfort building and deploying server-based applications
Familiarity with concepts like load balancers and network storage will be useful, though not required
Experience with Linux, containers, and container runtimes such as Docker Engine

Apply Pod Security Standards
Pod Security Standards are applied to a namespace using labels. The labels are as follows:
pod-security.kubernetes.io/<MODE>: <LEVEL>: REQUIRED
pod-security.kubernetes.io/<MODE>-version: <VERSION>: OPTIONAL (defaults to latest)

The three Pod Security Standards are as follows:

Baseline: Most common privilege escalation while enabling easier onboarding.
Restricted: Highly restricted covering security best practices. May cause workloads to break.
Privileged: Open and unrestricted.

Each standard is applied to a namespace using a given mode. There are three different modes a policy may be applied, which are as follows:

Enforce: Any Pods that violate the policy will be denied.
Warn: Any Pods that violate the policy will be allowed and a warning message will be displayed to the user.
Audit: Any Pods that violate the policy will generate an audit message in the audit log.

The following namespace illustrates how you may use multiple modes to enforce at one standard (baseline in this example) and audit and warn at another (restricted). Using multiple modes allows you to deploy a policy with a lower security posture and audit which workloads violate a standard with a more restricted policy. You can then remediate the policy violations before enforcing the more restricted standard. You can also pin a mode to a specific version (e.g., v1.22). This allows the policy standards to change with each Kubernetes release and allow you to pin a specific version. In the following example, we are enforcing the baseline standard and both warning and auditing the restricted standard. All modes are pinned to the v1.22 version of the standard.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: baseline-ns
  labels:
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/enforce-version: v1.22

    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/audit-version: v1.22
    pod-security.kubernetes.io/warn: restricted
    pod-security.kubernetes.io/warn-version: v1.22
```

In Kubernetes, a namespace is a virtual cluster that allows multiple teams or applications to share the same physical cluster while providing isolation and resource allocation. Namespaces are a way to divide cluster resources between multiple users, teams, or projects.

The "kind" field in Kubernetes YAML files is used to specify the type of Kubernetes object being defined. In this case, the "kind" field is set to "Namespace", indicating that this YAML file defines a Kubernetes Namespace object.

Therefore, the combination of "kind: Namespace" and the YAML configuration provided in the example file will create a Kubernetes namespace with the specified metadata and pod security policies.

This is a Kubernetes manifest file written in YAML format that defines a Kubernetes namespace named "baseline-ns" with certain labels and pod security policies.

The "metadata" section of the manifest file contains information about the namespace, such as its name and labels. The "labels" section specifies some labels that are associated with the namespace, which can be used to select or filter resources that belong to this namespace.

The "pod-security.kubernetes.io" annotations are used to specify pod security policies for the namespace. These annotations enforce and audit certain security policies when pods are created within the namespace.

Specifically, the "pod-security.kubernetes.io/enforce" annotation with a value of "baseline" indicates that the baseline pod security policies should be enforced for all pods created within this namespace. The "pod-security.kubernetes.io/enforce-version" annotation with a value of "v1.22" specifies the version of the baseline policies to use.

The "pod-security.kubernetes.io/audit" annotation with a value of "restricted" indicates that restricted audit policies should be enforced for pods created within this namespace. The "pod-security.kubernetes.io/audit-version" annotation with a value of "v1.22" specifies the version of the audit policies to use.

The "pod-security.kubernetes.io/warn" annotation with a value of "restricted" indicates that warning messages should be generated when pods created within this namespace violate the restricted policies. The "pod-security.kubernetes.io/warn-version" annotation with a value of "v1.22" specifies the version of the warning policies to use.

Create the namespace using the following command:

kubectl apply -f baseline-ns.yaml

```
$ kubectl apply -f baseline-ns.yaml
namespace/baseline-ns created
$
```

So we have applied security policies at namespace level .

Creating a Pod
We'll create a new Pod to see the effects of the Pod Security Standard. Create a file called kuard-pod.yaml with the following content:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: kuard
  labels:
    app: kuard
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:blue
      name: kuard
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
```

In Kubernetes, a Pod is the smallest deployable unit that represents a single instance of a running process in a cluster. A Pod can contain one or more containers that share the same network namespace and can communicate with each other using localhost.

The "kind" field in Kubernetes YAML files is used to specify the type of Kubernetes object being defined. In this case, the "kind" field is set to "Pod", indicating that this YAML file defines a Kubernetes Pod object.

Therefore, the combination of "kind: Pod" and the YAML configuration provided in the example file will create a Kubernetes Pod object named "kuard" with one container running the specified image and listening on the specified port. The label "app:kuard" is also associated with the Pod, which can be used for identifying or selecting it for various purposes.

This is a Kubernetes manifest file written in YAML format that defines a Kubernetes Pod object named "kuard" with one container.

The "metadata" section of the manifest file contains information about the Pod, such as its name and labels. The "labels" section specifies a label named "app" with a value of "kuard", which can be used to select or filter resources that belong to this Pod.

The "spec" section of the manifest file describes the desired state of the Pod, including its container(s), image, and networking configuration. In this case, the Pod has a single container specified by the "containers" section.

The container is defined by its "image" which is set to "gcr.io/kuar-demo/kuard-amd64:blue", the "name" field which is set to "kuard", and the "ports" section which specifies that the container will listen on port 8080 using the TCP protocol.

Overall, this YAML file will create a Pod with the specified container configuration and name, running the "kuard-amd64" image from the Google Container Registry.

Create the Pod and review the output with the following command:

```
$ kubectl apply -f baseline-ns.yaml
namespace/baseline-ns created
$ kubectl apply -f kuard-pod.yaml --namespace baseline-ns
Warning: would violate PodSecurity "restricted:v1.22": allowPrivilegeEscalation != false (container "kuard" must set securityContext.allowPrivilegeEscalation=false), unres
tricted capabilities (container "kuard" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "kuard" must set securityContext.runAsN
onRoot=true), seccompProfile (pod or container "kuard" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
pod/kuard created
$
```

As we can see when we create pod , security standard is applied .
In the output you can see that the Pod was successfully created; however, it violated the restricted Pod Security Standard. The details of the violations are provided in the output so that you can remediate.

Pod Security is a great way to manage the security posture of your workloads by applying policy at the namespace level and only allowing Pods to be created if they don't violate the policy. It's flexible and offers different prebuilt policies from permissive to restricted, along with tooling to easily roll out policy changes without the risk of breaking workloads.