Kubernetes Security Fundamentals

Pod Security Admissions

Pod security admission in Kubernetes involves the use of mechanisms that enforce security policies on pods at the time of their creation. These mechanisms help ensure that pods comply with the defined security standards and best practices, reducing the risk of security vulnerabilities.

Pod security admissions are enforced by admission controllers, which are plugins that intercept requests to the Kubernetes API Server before an object is persisted. These controllers can validate and mutate pod specifications to ensure compliance with security policies.

RealLife Example

Imagine a bustling city with a strict building code and a team of diligent inspectors (PSA). Just like these inspectors ensure new buildings comply with the code before construction begins, PSA safeguards your Kubernetes cluster by validating new pods against the defined Pod Security Standards (PSS).

Here's how Pod Security Admission (PSA) works:

1. **Pod Creation Request:** When a user or application attempts to create a pod in the cluster, the request is submitted to the Kubernetes API Server. Think of an architect submitting building plans to the city for approval.
2. **PSA Intervention:** The PSA admission controller intercepts the pod creation request. The inspector team halts construction (pod creation) for a mandatory review.
3. **Security Context Analysis:** PSA analyzes the pod's security context, including aspects like:
    - **Resource requests and limits:** CPU, memory, and other resources the pod can utilize.
    - **Capabilities:** Privileges granted to the pod's containers, such as network access or file system mounts.

    - **File System Permissions:** Permissions assigned to volumes mounted within the pod. Imagine the inspectors meticulously reviewing the building plans, checking resource allocation (materials required), permitted actions (construction activities allowed), and access controls (who can enter the construction site).
4. **Compliance Check:** PSA compares the pod's security context with the chosen Pod Security Standard (PSS) policy. The inspectors compare the building plans with the city's building code.
5. **Admission Decision:** Based on the comparison:
    - **Compliant Pod:** If the pod's security context aligns with the PSS policy, PSA grants admission, and the pod creation proceeds. The building plans comply with the code, and construction receives approval.
    - **Non-compliant Pod:** If the pod violates the PSS policy, PSA rejects the pod creation request. The building plans violate the code, and construction is denied.

Benefits of Pod Security Admission (PSA):

- **Enhanced Security:** By enforcing PSS policies, PSA reduces the attack surface and prevents insecure pod configurations from entering the cluster. Strict inspections minimize safety hazards during construction.
- **Improved Consistency:** PSA ensures all pods adhere to a baseline security standard, promoting a consistent security posture across the cluster. Consistent code enforcement across all constructions creates a safer cityscape.
- **Reduced Risk:** Proactive enforcement of security measures by PSA helps mitigate potential security risks associated with misconfigured pods. Early detection of building code violations prevents accidents during construction.

Key Concepts

1. **Admission Controllers**
    - Admission controllers are plugins that govern and enforce policies on how the cluster is used.
    - They can be validating or mutating controllers.

2. **Validating Admission Controllers**
    ◦ These controllers check if the incoming request complies with the predefined policies.
    ◦ They can reject requests that do not meet security standards.
3. **Mutating Admission Controllers**
    ◦ These controllers can modify the incoming request to enforce certain policies.
    ◦ They can add default values or modify configurations to enhance security.
4. **Pod Security Admission Controllers**
    ◦ Specific controllers focused on enforcing pod security standards.
    ◦ Examples: PodSecurityPolicy (PSP), PodSecurity Admission (newer alternative to PSP).

Security Best Practices

1. **Use Pod Security Admission Controllers**
    ◦ Implement PodSecurityPolicy or the newer PodSecurity Admission to enforce security policies.
    ◦ Define policies that restrict pod configurations to enhance security.
2. **Enable Necessary Admission Controllers**
    ◦ Ensure that essential admission controllers are enabled in the API Server configuration.
    ◦ Regularly review and update the list of enabled controllers based on security requirements.
3. **Define Comprehensive Security Policies**
    ◦ Create detailed security policies that cover various aspects of pod security, such as privilege escalation, running as non-root, and read-only file systems.
    ◦ Regularly review and update policies to adapt to evolving security threats.
4. **Audit and Monitor Pod Security**
    ◦ Continuously monitor the effectiveness of pod security admissions.
    ◦ Use logging and monitoring tools to track compliance and detect anomalies.

Lab Exercise: Implementing Pod Security Admissions
Objective
In this lab, you will learn how to implement pod security admissions using admission controllers. You will configure PodSecurityPolicies (PSPs) or the newer PodSecurity Admission, enable necessary admission controllers, and define comprehensive security policies.

Prerequisites

• A running Kubernetes cluster
• kubectl configured to interact with your cluster

Step-by-Step Instructions
Step 1: Enable Necessary Admission Controllers

1. Configure Admission Controllers

◦ Ensure that the desired admission controllers are enabled in the API Server configuration (usually located in /etc/kubernetes/manifests/kube-apiserver.yaml).

```
      -
--enable-admission-plugins=NamespaceLifecycle,LimitRanger,S
erviceAccount,DefaultStorageClass,ResourceQuota,PodSecurity
Policy
```

2. Restart the API Server

◦ Restart the API Server to apply the changes.

Step 2: Implement PodSecurityPolicies (PSPs)

1. Create a PodSecurityPolicy

◦ Define a YAML file for the PodSecurityPolicy.

```
       apiVersion: policy/v1beta1
       kind: PodSecurityPolicy
       metadata:
         name: restricted-psp
       spec:
```

```
    privileged: false
    seLinux:
      rule: 'MustRunAs'
    runAsUser:
      rule: 'MustRunAsNonRoot'
    fsGroup:
      rule: 'MustRunAs'
      ranges:
      - min: 1
        max: 65535
    volumes:
    - 'configMap'
    - 'emptyDir'
    - 'persistentVolumeClaim'
```

2. Apply the PodSecurityPolicy

◦ Apply the policy to the cluster.

```
  kubectl apply -f psp.yaml
```

3. Create a Role and RoleBinding for the PSP

◦ Define a Role that allows using the PSP.

```
  apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    namespace: default
    name: use-psp
  rules:
  - apiGroups: ['policy']
    resources: ['podsecuritypolicies']
    verbs: ['use']
    resourceNames: ['restricted-psp']
```

◦ Define a RoleBinding to bind the Role to a service account.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: default
  name: use-psp
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
roleRef:
  kind: Role
  name: use-psp
  apiGroup: rbac.authorization.k8s.io
```

4. Apply the Role and RoleBinding

◦ Apply the Role and RoleBinding to the cluster.

```
kubectl apply -f role.yaml
kubectl apply -f rolebinding.yaml
```

Step 3: Implement PodSecurity Admission (PSA)

1. Enable PodSecurity Admission

◦ Configure PodSecurity Admission by editing the API Server manifest to include the PodSecurity admission plugin.

```
- --enable-admission-plugins=PodSecurity
```

2. Create a PodSecurity Admission Policy

◦ Define a YAML file to set PodSecurity standards at namespace level.

```
apiVersion: v1
kind: Namespace
metadata:
  name: restricted-ns
```

```
        annotations:
          pod-security.kubernetes.io/enforce: restricted
          pod-security.kubernetes.io/enforce-version: 
v1.22
```

3. **Apply the PodSecurity Admission Policy**
   - Apply the policy to the cluster.

```
      kubectl apply -f namespace.yaml
```

Step 4: Define Comprehensive Security Policies

1. **Create a Security Policy**
   - Define a policy that restricts pod configurations.

```
      apiVersion: v1
      kind: Pod
      metadata:
        name: secure-pod
      spec:
        containers:
        - name: nginx
          image: nginx
          securityContext:
            runAsUser: 1000
            runAsGroup: 3000
            fsGroup: 2000
            readOnlyRootFilesystem: true
            capabilities:
              drop:
              - ALL
```

2. **Apply the Security Policy**
   - Apply the security policy to the cluster.

```
      kubectl apply -f secure-pod.yaml
```

Conclusion
Above exercise is just for techies, you can try it out and sort out the errors or perform debugging and troubleshooting yourself it will not come in exam, as it is a Multiple Choice Exam.

By following these steps, you have implemented pod security admissions in your Kubernetes cluster. You have configured necessary admission controllers, implemented PodSecurityPolicies or PodSecurity Admission, and defined comprehensive security policies. These practices help ensure that pods comply with security standards and reduce the risk of vulnerabilities.