DUMPSARENA

Kubernetes and Cloud Native Security Associate ()

Linux Foundation KCSA

Version Demo

Total Demo Questions: 6

Total Premium Questions: 69

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Overview of Cloud Native Security	1
Topic 2, Kubernetes Cluster Component Security	29
Topic 3, Kubernetes Security Fundamentals	25
Topic 4, Kubernetes Threat Model	5
Topic 5, Platform Security	5
Topic 6, Compliance and Security Frameworks	4
Total	69

QUESTION NO: 1

How do Kubernetes namespaces impact the application of policies when using Pod Security Admission?

A. Namespaces are ignored; Pod Security Admission policies apply cluster-wide only.

B. Different policies can be applied to specific namespaces.

C. Each namespace can have only one active policy.

D. The default namespace enforces the strictest security policies by default.

ANSWER: B

Explanation:

Different policies can be applied to specific namespaces. Pod Security Admission uses namespace labels to determine which Pod Security Standards level applies to Pods created in that namespace. The available policy levels are `privileged`, `baseline`, and `restricted`, allowing an organization to apply controls appropriate to each workload's security requirements. For example, a namespace hosting a tightly controlled application can use the `restricted` standard, while a namespace for infrastructure components that require broader host access can use a different standard where justified.

Administrators configure this behavior with labels such as `pod-security.kubernetes.io/enforce`, optionally including a version label to pin evaluation to a particular Kubernetes version. The same namespace can also use the `warn` and `audit` modes, helping teams identify noncompliant Pod specifications before enforcing a policy. This namespace-scoped design supports incremental adoption: policies can be tested or strengthened independently without imposing an identical standard on every namespace. Kubernetes documents that Pod Security Admission is configured through labels on namespaces and evaluates Pods against the selected Pod Security Standards profile. See the Kubernetes Pod Security Admission documentation and the Pod Security Standards documentation.

QUESTION NO: 2

What is the purpose of the Supplier Assessments and Reviews control in the NIST 800-53 Rev. 5 set of controls for Supply Chain Risk Management?

A. To evaluate and monitor existing suppliers for adherence to security requirements.

B. To conduct regular audits of suppliers' financial performance.

C. To establish contractual agreements with suppliers.

D. To identify potential suppliers for the organization.

ANSWER: A

Explanation:

To evaluate and monitor existing suppliers for adherence to security requirements is correct because the Supplier Assessments and Reviews control, SR-6, requires organizations to assess and review supply-chain-related risks associated with suppliers, contractors, and system or component developers. These activities provide evidence that a supplier can satisfy the organization's security and supply-chain risk-management requirements throughout the relationship, rather than relying solely on a one-time preselection decision. Assessments and reviews can consider such matters as the supplier's security practices, development and delivery processes, provenance practices, and ability to protect organizational systems and information. The results are used to inform the organization's risk response, including decisions about whether risks are acceptable and what monitoring or mitigation is needed. NIST also notes that organizations can use a variety of assessment approaches, including independent assessments, supplier-provided evidence, and relevant external information, with the rigor tailored to the criticality of the supplied service or component. See the NIST SP 800-53 Release 5.1 publication and NIST's SP 800-53 control catalog for the Supply Chain Risk Management control family and SR-6.

QUESTION NO: 3

What is Grafana?

A. A cloud-native distributed tracing system for monitoring microservices architectures.

B. A container orchestration platform for managing and scaling applications.

C. A platform for monitoring and visualizing time-series data.

D. A cloud-native security tool for scanning and detecting vulnerabilities in Kubernetes clusters.

ANSWER: C

Explanation:

Grafana is an open-source observability platform used to query, visualize, explore, and alert on data from many sources. It is particularly well known for building dashboards from time-series metrics, including metrics collected by Prometheus, but it can also work with logs, traces, databases, and other data sources through integrations. In a Kubernetes or cloud-native environment, teams commonly use Grafana to create dashboards that show cluster, node, workload, application, and service-health metrics over time. Its visualization capabilities make operational data easier to interpret, helping users identify trends, investigate incidents, and monitor the health and performance of systems. Grafana Alerting can also evaluate queries and send notifications when configured conditions are met. Therefore, “A platform for monitoring and visualizing time-series data.” is the best description among the available choices. Although Grafana supports a broader observability use case than time-series data alone, monitoring and visualizing time-series metrics is a core and widely recognized function. Grafana’s documentation describes it as an open-source platform for monitoring and observability, and its data-source documentation explains that Grafana can query and visualize data from connected sources. See the Grafana introduction and Grafana data sources documentation.

QUESTION NO: 4

A cluster administrator wants to enforce the use of a different container runtime depending on the application a workload belongs to.

A. By manually modifying the container runtime for each workload after it has been created.

B. By modifying the kube-apiserver configuration file to specify the desired container runtime for each application.

C. By configuring a**validating admission controller**webhook that verifies the container runtime based on the application label and rejects requests that do not comply.

D. By configuring a**mutating admission controller**webhook that intercepts new workload creation requests and modifies the container runtime based on the application label.

ANSWER: D

Explanation:

By configuring a **mutating admission controller** webhook that intercepts new workload creation requests and modifies the container runtime based on the application label is correct because Kubernetes provides workload-specific runtime selection through the `RuntimeClass` API. A RuntimeClass maps a named runtime handler, configured by the cluster’s container runtime interface implementation, to Pods that specify that class using `spec.runtimeClassName`. This makes it possible for a cluster to support distinct runtime environments, such as a sandboxed runtime for selected applications and the standard runtime for others.

A mutating admission webhook can apply this policy automatically at admission time. It can inspect an application-identifying label and inject the appropriate `runtimeClassName` into the incoming Pod specification or into the Pod template of a supported workload resource. The API server then persists the mutated object, and the scheduler and kubelet use the selected RuntimeClass when the Pod is scheduled and started. This provides centralized, consistent enforcement without requiring application teams to remember runtime settings in every manifest. Kubernetes documents RuntimeClass and its use with `runtimeClassName` in the RuntimeClass documentation; webhook mutation behavior is described in the dynamic admission control documentation.

QUESTION NO: 5

A Kubernetes cluster tenant can launch privileged Pods in contravention of the**restricted Pod Security Standard**mandated for cluster tenants and enforced by the built-in**PodSecurity admission controller**.

The tenant has full CRUD permissions on the namespace object and the namespaced resources. How did the tenant achieve this?

A. The scope of the tenant role means privilege escalation is impossible.

B. By tampering with the namespace labels.

C. By deleting the PodSecurity admission controller deployment running in their namespace.

D. By using higher-level access credentials obtained reading secrets from another namespace.

ANSWER: B

Explanation:

By tampering with the namespace labels is correct because the built-in PodSecurity admission controller determines which Pod Security Standards to apply to a namespace from labels on that Namespace object. In particular, labels such as `pod-security.kubernetes.io/enforce: restricted` select the enforced policy level. If a tenant has permission to update, patch, or otherwise modify its Namespace, it can alter or remove those Pod Security admission labels. Subsequent Pod creation requests are then evaluated using the changed label configuration rather than the cluster operator's intended `restricted` profile. For example, changing the enforced level to `privileged`, or removing the enforcement label where no stricter default applies, can permit a privileged Pod specification that would otherwise be rejected. This is why namespace modification is security-sensitive: Namespace labels are not merely informational metadata when Pod Security Admission is enabled; they are policy configuration consumed at admission time. Kubernetes documentation explicitly states that Pod Security Admission uses namespace labels to configure enforce, audit, and warn modes and their policy versions. Cluster administrators should restrict tenant write access to Namespace objects or use an authorization policy that prevents changes to the `pod-security.kubernetes.io/*` labels. See the Kubernetes Pod Security Admission documentation and the Pod Security Standards documentation.

QUESTION NO: 6

An application team wants to allow an external deployment system to update Deployments in one namespace, but it must not be able to read or modify any Secret objects.

Which RBAC design best follows the principle of least privilege?

A. Create a Role for Deployment operations in the namespace and bind it to the deployment system with a RoleBinding.

B. Create a ClusterRole that grants all verbs on Deployments and Secrets, then use a ClusterRoleBinding.

C. Grant the deployment system the `cluster-admin` ClusterRole for the duration of each deployment.

D. Store the deployment system credential in a Secret and grant it read access to all Secrets in the namespace.

ANSWER: A

Explanation:

A namespaced Role containing only the required verbs on the `deployments` resource, granted through a RoleBinding in that namespace, limits the deployment system to its stated function. RBAC permissions are granted explicitly; access to Deployments does not require Secret read access. A ClusterRoleBinding would unnecessarily extend the scope of the permissions across the cluster, and granting Secret access increases the impact of a compromised deployment-system credential.