ExamsIndex
Practice Materials & Tests

DEMO VERSION

Linux Foundation

KCNA Exam

Kubernetes and Cloud Native Associate

Exam Latest Version: 11.0

Question 1. (Single Select)

What native runtime is Open Container Initiative (OCI) compliant?

A: runC
B: runV
C: kata-containers
D: gvisor

Correct Answer: A

Explanation:

The Open Container Initiative (OCI) publishes open specifications for container images and container runtimes so that tools across the ecosystem remain interoperable. When a runtime is "OCI-compliant," it means it implements the OCI Runtime Specification (how to run a container from a filesystem bundle and configuration) and/or works cleanly with OCI image formats through the usual layers (image ↓ unpack ↓ runtime). runC is the reference implementation of the OCI runtime specification and is the low-level runtime underneath many higher-level systems. In Kubernetes, you typically interact with a higher-level container runtime (such as containerd or CRI-O) through the Container Runtime Interface (CRI). That higher-level runtime then uses a low-level OCI runtime to actually create Linux namespaces/cgroups, set up the container process, and start it. In many default installations, containerd delegates to runC for this low-level "create/start" work.

The other options are related but differ in what they are: Kata Containers uses lightweight VMs to provide stronger isolation while still presenting a container-like workflow; gVisor provides a user-space kernel for sandboxing containers; these can be used with Kubernetes via compatible integrations, but the canonical "native OCI runtime" answer in most curricula is runC. Finally, "runV" is not a common modern Kubernetes runtime choice in typical OCI discussions. So the most correct, standards-based answer here is A (runC) because it directly implements the OCI runtime spec and is commonly used as the default low-level runtime behind CRI implementations.

=========

Question 2. (Single Select)

Let’s assume that an organization needs to process large amounts of data in bursts, on a cloud-based Kubernetes cluster. For instance: each Monday morning, they need to run a batch of 1000 compute jobs of 1 hour each, and these jobs must be completed by Monday night. What’s going to be the most cost-effective method?

A: Run a group of nodes with the exact required size to complete the batch on time, and use a combination of taints, tolerations, and nodeSelectors to reserve these nodes to the batch jobs.
B: Leverage the Kubernetes Cluster Autoscaler to automatically start and stop nodes as they’re needed.
C: Commit to a specific level of spending to get discounted prices (with e.g. “reserved instances” or similar mechanisms).
D: Use PriorityClasses so that the weekly batch job gets priority over other workloads running on the cluster, and can be completed on time.

Correct Answer: B

Explanation:

Burst workloads are a classic elasticity problem: you need large capacity for a short window, then very little capacity the rest of the week. The most cost-effective approach in a cloud-based Kubernetes environment is to scale infrastructure dynamically, matching node count to current demand. That’s exactly what Cluster Autoscaler is designed for: it adds nodes when Pods cannot be scheduled due to insufficient resources and removes nodes when they become underutilized and can be drained safely. Therefore B is correct.

Option A can work operationally, but it commonly results in paying for a reserved “standing army” of nodes that sit idle most of the week—wasteful for bursty patterns unless the nodes are repurposed for other workloads. Taints/tolerations and nodeSelectors are placement tools; they don’t reduce cost by themselves and may increase waste if they isolate nodes. Option D (PriorityClasses) affects which Pods get scheduled first given available capacity, but it does not create capacity. If the cluster doesn’t have enough nodes, high priority Pods will still remain Pending. Option C (reserved instances or committed-use discounts) can reduce unit price, but it assumes relatively predictable baseline usage. For true bursts, you usually want a smaller baseline plus autoscaling, and optionally combine it with discounted capacity types if your cloud supports them.

In Kubernetes terms, the control loop is: batch Jobs create Pods !' if many Pods are Pending due to insufficient CPU/memory, Clus and increases the node group size !' new nodes join and kube-sc jobs finish and nodes become empty, Cluster Autoscaler drains and removes nodes. This matches cloud-native principles: elasticity, pay-for-what-you-use, and automation. It minimizes idle capacity while still meeting the completion deadline.

=========

Question 3. (Single Select)

What is a Kubernetes service with no cluster IP address called?

A: Headless Service
B: Nodeless Service
C: IPLess Service
D: Specless Service

Correct Answer: A

Explanation:

A Kubernetes Service normally provides a stable virtual IP (ClusterIP) and a DNS name that load-balances traffic across matching Pods. A headless Service is a special type of Service where Kubernetes does not allocate a ClusterIP. Instead, the Service's DNS returns individual Pod IPs (or other endpoint records), allowing clients to connect directly to specific backends rather than through a single virtual IP. That is why the correct answer is A (Headless Service).

Headless Services are created by setting spec.clusterIP: None. When you do this, kube-proxy does not program load-balancing rules for a virtual IP because there isn't one. Instead, service discovery is handled via DNS records that point to the actual endpoints. This behavior is especially important for stateful or identity-sensitive systems where clients must talk to a particular replica (for example, databases, leader/follower clusters, or StatefulSet members).

This is also why headless Services pair naturally with StatefulSets. StatefulSets provide stable network identities (pod-0, pod-1, etc.) and stable DNS names. The headless Service provides

the DNS domain that resolves each Pod’s stable hostname to its IP, enabling peer discovery and consistent addressing even as Pods move between nodes.

The other options are distractors: “Nodeless,” “IPLess,” and “Specless” are not Kubernetes Service types. In the core API, the Service “types” are things like ClusterIP, NodePort, LoadBalancer, and ExternalName; “headless” is a behavioral mode achieved through the ClusterIP field.

In short: a headless Service removes the virtual IP abstraction and exposes endpoint-level discovery. It’s a deliberate design choice when load-balancing is not desired or when the application itself handles routing, membership, or sharding.

==========

Question 4. (Single Select)

CI/CD stands for:

A: Continuous Information / Continuous Development
B: Continuous Integration / Continuous Development
C: Cloud Integration / Cloud Development
D: Continuous Integration / Continuous Deployment

Correct Answer: D

Explanation:

CI/CD is a foundational practice for delivering software rapidly and reliably, and it maps strongly to cloud native delivery workflows commonly used with Kubernetes. CI stands for Continuous Integration: developers merge code changes frequently into a shared repository, and automated systems build and test those changes to detect issues early. CD is commonly used to mean Continuous Delivery or Continuous Deployment depending on how far automation goes. In many certification contexts and simplified definitions like this question, CD is interpreted as Continuous Deployment, meaning every change that passes the automated pipeline is automatically released to production. That matches option D.

In a Kubernetes context, CI typically produces artifacts such as container images (built from Dockerfiles or similar build definitions), runs unit/integration tests, scans dependencies, and pushes images to a registry. CD then promotes those images into environments by updating Kubernetes manifests (Deployments, Helm charts, Kustomize overlays, etc.). Progressive delivery patterns (rolling updates, canary, blue/green) often use Kubernetes-native controllers and Service routing to reduce risk.

Why the other options are incorrect: “Continuous Development” isn’t the standard “D” term; it’s ambiguous and not the established acronym expansion. “Cloud Integration/Cloud Development” is unrelated. Continuous Delivery (in the stricter sense) means changes are always in a deployable state and releases may still require a manual approval step, while Continuous Deployment removes that final manual gate. But because the option set explicitly includes “Continuous Deployment,” and that is one of the accepted canonical expansions for CD, D is the correct selection here.

Practically, CI/CD complements Kubernetes’ declarative model: pipelines update desired state (Git or manifests), and Kubernetes reconciles it. This combination enables frequent releases, repeatability, reduced human error, and faster recovery through automated rollbacks and controlled rollout strategies.

=========

Question 5. (Single Select)

What default level of protection is applied to the data in Secrets in the Kubernetes API?

A: The values use AES symmetric encryption
B: The values are stored in plain text
C: The values are encoded with SHA256 hashes
D: The values are base64 encoded

Correct Answer: D

Explanation:

Kubernetes Secrets are designed to store sensitive data such as tokens, passwords, or

certificates and make them available to Pods in controlled ways (as environment variables or mounted files). However, the default protection applied to Secret values in the Kubernetes API is base64 encoding, not encryption. That is why D is correct. Base64 is an encoding scheme that converts binary data into ASCII text; it is reversible and does not provide confidentiality.

By default, Secret objects are stored in the cluster's backing datastore (commonly etcd) as base64-encoded strings inside the Secret manifest. Unless the cluster is configured for encryption at rest, those values are effectively stored unencrypted in etcd and may be visible to anyone who can read etcd directly or who has API permissions to read Secrets. This distinction is critical for security: base64 can prevent accidental issues with special characters in YAML/JSON, but it does not protect against attackers.

Option A is only correct if encryption at rest is explicitly configured on the API server using an EncryptionConfiguration (for example, AES-CBC or AES-GCM providers). Many managed Kubernetes offerings enable encryption at rest for etcd as an option or by default, but that is a deployment choice, not the universal Kubernetes default. Option C is incorrect because hashing is used for verification, not for secret retrieval; you typically need to recover the original value, so hashing isn't suitable for Secrets. Option B ("plain text") is misleading: the stored representation is base64-encoded, but because base64 is reversible, the security outcome is close to plain text unless encryption at rest and strict RBAC are in place.

The correct operational stance is: treat Kubernetes Secrets as sensitive; lock down access with RBAC, enable encryption at rest, avoid broad Secret read permissions, and consider external secret managers when appropriate. But strictly for the question's wording—default level of protection—base64 encoding is the right answer.

=========

ExamsIndex

Demo PDF Complete

Your KCNA Demo (5 Questions)

Get the Complete Version

Full Questions with Detailed Explanations

Interactive Web-Based Exams Available

To get 30% off, use Coupon Code: OFF30

https://examsindex.com/exam/kcna