Training KCNA Solutions, Latest KCNA Exam Notes

P.S. Free 2026 Linux Foundation KCNA dumps are available on Google Drive shared by ExamCost: https://drive.google.com/open?id=1XgHu7zjZl7Kh5BCBwOkLhvFdkZ4et8TO

If you have interests with our KCNA practice materials, we prefer to tell that we have contacted with many former buyers of our KCNA exam questions and they all talked about the importance of effective KCNA learning prep playing a crucial role in your preparation process. Our practice materials keep exam candidates motivated and efficient with useful content based wholly on the real KCNA Guide materials.

Linux Foundation KCNA certification is an excellent way for individuals to validate their skills and knowledge in Kubernetes and cloud-native technologies. Kubernetes and Cloud Native Associate certification provides a solid foundation in these technologies and is recognized globally. With the growing adoption of cloud-native technologies, the demand for certified professionals is on the rise, and the KCNA Certification is an excellent way to stand out in a competitive job market.

>> Training KCNA Solutions <<

2026 100% Free KCNA –Reliable 100% Free Training Solutions | Latest Kubernetes and Cloud Native Associate Exam Notes

When preparing for the test KCNA certification, most clients choose our products because our KCNA learning file enjoys high reputation and boost high passing rate. Our products are the masterpiece of our company and designed especially for the certification. Our KCNA latest study question has gone through strict analysis and verification by the industry experts and senior published authors. The clients trust our products and treat our products as the first choice. So the total amounts of the clients and the sales volume of our KCNA learning file is constantly increasing.

Linux Foundation KCNA (Kubernetes and Cloud Native Associate) Exam is an industry-recognized certification that validates the skills and knowledge of professionals in cloud computing and Kubernetes. Kubernetes and Cloud Native Associate certification is designed for individuals who want to demonstrate their proficiency in cloud-native technologies and Kubernetes, the popular open-source container orchestration platform.

Linux Foundation Kubernetes and Cloud Native Associate Sample Questions (Q183-Q188):

NEW QUESTION # 183
What is the API that exposes resource metrics from the metrics-server?

- A. cadvisor.k8s.io
- B. metrics.k8s.io
- C. custom.k8s.io
- D. resources.k8s.io

Answer: B

Explanation:
The correct answer is C: metrics.k8s.io. Kubernetes' metrics-server is the standard component that provides resource metrics (primarily CPU and memory) for nodes and pods. It aggregates this information (sourced from kubelet/cAdvisor) and serves it through the Kubernetes aggregated API under the group metrics.k8s.io.
This is what enables commands like kubectl top nodes and kubectl top pods, and it is also a key data source for autoscaling with the Horizontal Pod Autoscaler (HPA) when scaling on CPU/memory utilization.
Why the other options are wrong:
* custom.k8s.io is not the standard API group for metrics-server resource metrics. Custom metrics are typically served through the custom metrics API (commonly custom.metrics.k8s.io) via adapters (e.g., Prometheus Adapter), not metrics-server.
* resources.k8s.io is not the metrics-server API group.
* cadvisor.k8s.io is not exposed as a Kubernetes aggregated metrics API. cAdvisor is a component integrated into kubelet that provides container stats, but metrics-server is the thing that exposes the aggregated Kubernetes metrics API, and the canonical group is metrics.k8s.io.
Operationally, it's important to understand the boundary: metrics-server provides basic resource metrics suitable for core autoscaling and "top" views, but it is not a full observability system (it does not store long- term metrics history like Prometheus). For richer metrics (SLOs, application metrics, long-term trending), teams typically deploy Prometheus or a managed monitoring backend. Still, when the question asks specifically which API exposes metrics-server data, the answer is definitively metrics.k8s.io.
==========

NEW QUESTION # 184
A CronJob is scheduled to run by a user every one hour. What happens in the cluster when it's time for this CronJob to run?

- A. Kubelet watches API Server for CronJob objects. When it's time for a Job to run, it runs the Pod directly.
- B. CronJob controller component creates a Job. Then the Job controller creates a Pod and waits until it finishes to run.
- C. CronJob controller component creates a Pod and waits until it finishes to run.
- D. Kube-scheduler watches API Server for CronJob objects, and this is why it's called kube-scheduler.

Answer: B

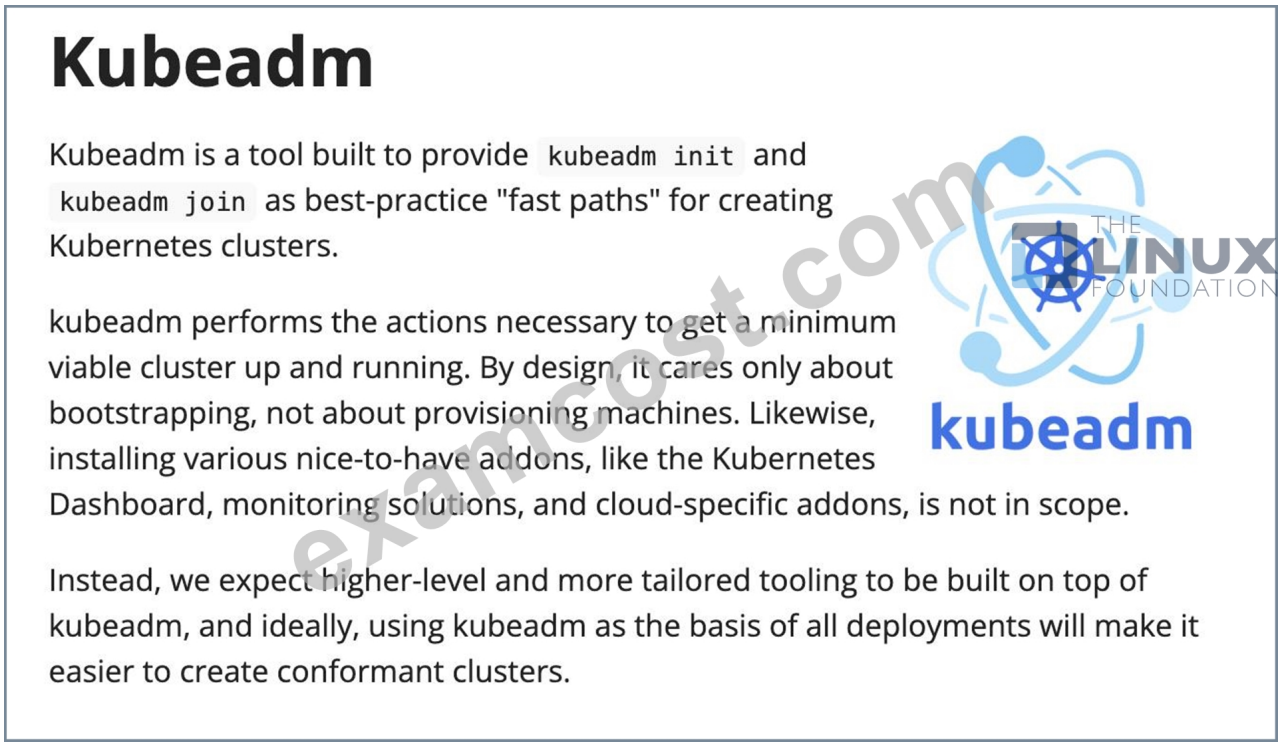
Explanation:
CronJobs are implemented through Kubernetes controllers that reconcile desired state. When the scheduled time arrives, the CronJob controller (part of the controller-manager set of control plane controllers) evaluates the CronJob object's schedule and determines whether a run should be started. Importantly, CronJob does not create Pods directly as its primary mechanism. Instead, it creates a Job object for each scheduled execution. That Job object then becomes the responsibility of the Job controller, which creates one or more Pods to complete the Job's work and monitors them until completion. This separation of concerns is why option D is correct.
This design has practical benefits. Jobs encapsulate "run-to-completion" semantics: retries, backoff limits, completion counts, and tracking whether the work has succeeded. CronJob focuses on the temporal triggering aspect (schedule, concurrency policy, starting deadlines, history limits), while Job focuses on the execution aspect (create Pods, ensure completion, retry on failure).
Option A is incorrect because kubelet is a node agent; it does not watch CronJob objects and doesn't decide when a schedule triggers. Kubelet reacts to Pods assigned to its node and ensures containers run there. Option B is incorrect because kube-scheduler schedules Pods to nodes after they exist (or are created by controllers); it does not trigger CronJobs. Option C is incorrect because CronJob does not usually create a Pod and wait directly; it delegates via a Job, which then manages Pods and completion.
So, at runtime: CronJob controller creates a Job; Job controller creates the Pod(s); scheduler assigns those Pods to nodes; kubelet runs them; Job controller observes success/failure and updates status; CronJob controller manages run history and concurrency rules.
==========

NEW QUESTION # 185
kubeadm is an administrative dashboard for kubernetes

- A. True
- B. False

Answer: B

Explanation:
https://kubernetes.io/docs/reference/setup-tools/kubeadm/



NEW QUESTION # 186
Which of the following statements is correct concerning Open Policy Agent (OPA)?

- A. The policies must be written in Python language.
- B. It cannot be used outside Kubernetes.
- C. Kubernetes can use it to validate requests and apply policies.
- D. Policies can only be tested when published.

Answer: C

Explanation:
Open Policy Agent (OPA) is a general-purpose policy engine used to define and enforce policy across different systems. In Kubernetes, OPA is commonly integrated through admission control (often via Gatekeeper or custom admission webhooks) to validate and/or mutate requests before they are persisted in the cluster. This makes B correct: Kubernetes can use OPA to validate API requests and apply policy decisions.
Kubernetes' admission chain is where policy enforcement naturally fits. When a user or controller submits a request (for example, to create a Pod), the API server can call external admission webhooks. Those webhooks can evaluate the request against policy-such as "no privileged containers," "images must come from approved registries," "labels must include cost-center," or "Ingress must enforce TLS." OPA's policy language (Rego) allows expressing these rules in a declarative form, and the decision ("allow/deny" and sometimes patches) is returned to the API server. This enforces governance consistently and centrally.
Option A is incorrect because OPA policies are written in Rego, not Python. Option C is incorrect because policies can be tested locally and in CI pipelines before deployment; in fact, testability is a key advantage. Option D is incorrect because OPA is designed to be platform-agnostic-it can be used with APIs, microservices, CI/CD pipelines, service meshes, and infrastructure tools, not only Kubernetes.
From a Kubernetes fundamentals view, OPA complements RBAC: RBAC answers "who can do what to which resources," while OPA-style admission policies answer "even if you can create this resource, does it meet our organizational rules?" Together they help implement defense in depth: authentication + authorization + policy admission + runtime security controls. That is why OPA is widely used to enforce security and compliance requirements in Kubernetes environments.

NEW QUESTION # 187
What is the default deployment strategy in Kubernetes?

- A. Blue/Green deployment
- B. Canary deployment

- C. Rolling update
- D. Recreate deployment

Answer: C

Explanation:
For Kubernetes Deployments, the default update strategy is RollingUpdate, which corresponds to "Rolling update" in option A. Rolling updates replace old Pods with new Pods gradually, aiming to maintain availability during the rollout. Kubernetes does this by creating a new ReplicaSet for the updated Pod template and then scaling the new ReplicaSet up while scaling the old one down. The pace and safety of a rolling update are controlled by parameters like maxUnavailable and maxSurge. maxUnavailable limits how many replicas can be unavailable during the update, protecting availability. maxSurge controls how many extra replicas can be created temporarily above the desired count, helping speed up rollouts while maintaining capacity. If readiness probes fail, Kubernetes will pause progression because new Pods aren't becoming Ready, helping prevent a bad version from fully replacing a good one.
Options B (Blue/Green) and C (Canary) are popular progressive delivery patterns, but they are not the default built-in Deployment strategy. They are typically implemented using additional tooling (service mesh routing, traffic splitting controllers, or specialized rollout controllers) or by operating multiple Deployments/Services. Option D (Recreate) is a valid strategy but not the default; it terminates all old Pods before creating new ones, causing downtime unless you have external buffering or multi-tier redundancy. From an application delivery perspective, RollingUpdate aligns with Kubernetes' declarative model: you update the desired Pod template and let the controller converge safely. kubectl rollout status is commonly used to monitor progress. Rollbacks are also supported because the Deployment tracks history. Therefore, the verified correct answer is A: Rolling update.

NEW QUESTION # 188
.....