Exam Linux Foundation KCNA Registration - Regualer KCNA Update

BONUS!!! Download part of ExamBoosts KCNA dumps for free: https://drive.google.com/open?id=1-WwEc7iBMOuzQQY0GdClnley-p2WLuO0

Now we can say that with the KCNA Exam Dumps you will get the updated and verified Linux Foundation KCNA exam practice Test all the time. With the Kubernetes and Cloud Native Associate KCNA Exam Questions, you will get the opportunity to download the updated and real Kubernetes and Cloud Native Associate KCNA exam practice questions.

The KCNA certification is highly regarded in the IT industry and is recognized by many organizations as a benchmark for cloud-native expertise. Kubernetes and Cloud Native Associate certification is ideal for IT professionals who want to enhance their career prospects and demonstrate their knowledge and skills in the field of cloud computing and containerization. Kubernetes and Cloud Native Associate certification is also suitable for organizations that want to validate the skills of their employees in Kubernetes and other cloud-native technologies.

The KCNA Exam is ideal for IT professionals who are looking to advance their careers in cloud-native computing. KCNA exam covers a wide range of topics, including containerization, Kubernetes architecture, deployment, and maintenance. It also covers other important topics such as networking, storage, security, and troubleshooting.

>> Exam Linux Foundation KCNA Registration <<

Prepare with ExamBoosts and Achieve Linux Foundation KCNA Exam Success

To provide our users with the Kubernetes and Cloud Native Associate (KCNA) latest questions based on the sections of the actual exam quesions, we regularly update our KCNA study material. Also, ExamBoosts provides free updates of Linux Foundation KCNA Exam Questions for up to 365 days. For customers who don't crack the Linux Foundation KCNA test after using our product, ExamBoosts will provides them a refund guarantee according to terms and conditions.

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 (Q155-Q160):

NEW QUESTION # 155
You have a Kubernetes deployment running on a cluster that includes multiple nodes. You need to ensure that Pods from this deployment are scheduled only on nodes with a specific label 'gpu=true'. How can you achieve this?

- A. By using a daemonset to run the deployment only on nodes with the 'gpu=true' label.

- B. By applying a NodeAffinity to the deployments pod template.
- C. By using a ReplicaSet to control the deployments replicas on nodes with the 'gpu=true' label.
- D. By using a PodDisruptionBudget to ensure that only nodes with the 'gpu=true' label are considered for scheduling.
- E. By creating a custom resource definition that defines the GPU constraint and applying it to the deployment.

Answer: B

Explanation:
Applying a NodeAffinity to the deployment's pod template is the correct way to schedule Pods on nodes with a specific label. NodeAffinity allows you to specify preferences or requirements for node selection based on labels, taints, and other criteria. Option 'B' is incorrect as PodDisruptionBudget deals with graceful pod termination during node maintenance. Options 'C' and 'D' are not the appropriate approach in this situation. Option 'E' is also incorrect because a ReplicaSet is used for managing replicas, not for node scheduling.

NEW QUESTION # 156
What is a sidecar container?

- A. A container that runs next to another container within the same Pod.
- B. A container that runs next to another Pod within the same namespace.
- C. A Pod that runs next to another Pod within the same namespace.
- D. A Pod that runs next to another container within the same Pod.

Answer: A

Explanation:
A sidecar container is an additional container that runs alongside the main application container within the same Pod, sharing network and storage context. That matches option C, so C is correct. The sidecar pattern is used to add supporting capabilities to an application without modifying the application code. Because both containers are in the same Pod, the sidecar can communicate with the main container over localhost and share volumes for files, sockets, or logs.
Common sidecar examples include: log forwarders that tail application logs and ship them to a logging system, proxies (service mesh sidecars like Envoy) that handle mTLS and routing policy, config reloaders that watch ConfigMaps and signal the main process, and local caching agents. Sidecars are especially powerful in cloud-native systems because they standardize cross-cutting concerns-security, observability, traffic policy-across many workloads.
Options A and D incorrectly describe "a Pod running next to ..." which is not how sidecars work; sidecars are containers, not separate Pods. Running separate Pods "next to" each other in a namespace does not give the same shared network namespace and tightly coupled lifecycle. Option B is also incorrect for the same reason: a sidecar is not a separate Pod; it is a container in the same Pod.
Operationally, sidecars share the Pod lifecycle: they are scheduled together, scaled together, and generally terminated together. This is both a benefit (co-location guarantees) and a responsibility (resource requests/limits should include the sidecar's needs, and failure modes should be understood). Kubernetes is increasingly formalizing sidecar behavior (e.g., sidecar containers with ordered startup semantics), but the core definition remains: a helper container in the same Pod.

NEW QUESTION # 157
How does Horizontal Pod autoscaling work in Kubernetes?

- A. The Horizontal Pod Autoscaler controller adds more CPU or memory to the pods when the load is above the configured threshold, and reduces CPU or memory when the load is below.
- B. The Horizontal Pod Autoscaler controller adds more pods when the load is above the configured threshold, and reduces the number of pods when the load is below.
- C. The Horizontal Pod Autoscaler controller adds more pods to the specified DaemonSet when the load is above the configured threshold, and reduces the number of pods when the load is below.
- D. The Horizontal Pod Autoscaler controller adds more pods when the load is above the configured threshold, but does not reduce the number of pods when the load is below.

Answer: B

Explanation:
Horizontal Pod Autoscaling (HPA) adjusts the number of Pod replicas for a workload controller (most commonly a Deployment) based on observed metrics, increasing replicas when load is high and decreasing when load drops. That matches D, so D is correct.

HPA does not add CPU or memory to existing Pods-that would be vertical scaling (VPA). Instead, HPA changes spec.replicas on the target resource, and the controller then creates or removes Pods accordingly.
HPA commonly scales based on CPU utilization and memory (resource metrics), and it can also scale using custom or external metrics if those are exposed via the appropriate Kubernetes metrics APIs.
Option A is vertical scaling behavior, not HPA. Option B is incorrect because HPA can scale down as well as up (subject to stabilization windows and configuration), so it's not "scale up only." Option C is incorrect because HPA does not scale DaemonSets in the usual model; DaemonSets are designed to run one Pod per node (or per selected nodes) rather than a replica count. HPA targets resources like Deployments, ReplicaSets (via Deployment), and StatefulSets in typical usage, where replica count is a meaningful knob.
Operationally, HPA works as a control loop: it periodically reads metrics (for example, via metrics-server for CPU/memory, or via adapters for custom metrics), compares the current value to the desired target, and calculates a desired replica count within min/max bounds. To avoid flapping, HPA includes stabilization behavior and cooldown logic so it doesn't scale too aggressively in response to short spikes or dips. You can configure minimum and maximum replicas and behavior policies to tune responsiveness.
In cloud-native systems, HPA is a key elasticity mechanism: it enables services to handle variable traffic while controlling cost by scaling down during low demand. Therefore, the verified correct answer is D.
==========

NEW QUESTION # 158
You are managing a large Kubernetes cluster with multiple namespaces. You want to control access to resources within different namespaces. Which of the following mechanisms can be used to achieve fine-grained access control?

- A. Service Accounts
- B. Network policies
- C. Network Segmentation
- D. Pod Security Policies
- E. Role-Based Access Control (RBAC)

Answer: D,E

Explanation:
Both Role-Based Access Control (RBAC) and Pod Security Policies (PSP) are used for managing access to resources within Kubernetes. RBAC provides fine-grained permissions based on roles and users, while PSPs define security constraints for Pods, limiting their capabilities and access to resources.

NEW QUESTION # 159
In which framework do the developers no longer have to deal with capacity, deployments, scaling and fault tolerance, and OS?

- A. Serverless
- B. Mesos
- C. Kubernetes
- D. Docker Swarm

Answer: A

Explanation:
Serverless is the model where developers most directly avoid managing server capacity, OS operations, and much of the deployment/scaling/fault-tolerance mechanics, which is why D is correct. In serverless computing (commonly Function-as-a-Service, FaaS, and managed serverless container platforms), the provider abstracts away the underlying servers. You typically deploy code (functions) or a container image, define triggers (HTTP events, queues, schedules), and the platform automatically provisions the required compute, scales it based on demand, and handles much of the availability and fault tolerance behind the scenes.
It's important to compare this to Kubernetes: Kubernetes does automate scheduling, self-healing, rolling updates, and scaling, but it still requires you (or your platform team) to design and operate cluster capacity, node pools, upgrades, runtime configuration, networking, and baseline reliability controls. Even in managed Kubernetes services, you still choose node sizes, scale policies, and operational configuration. Kubernetes reduces toil, but it does not eliminate infrastructure concerns in the same way serverless does.
Docker Swarm and Mesos are orchestration platforms that schedule workloads, but they also require managing the underlying capacity and OS-level aspects. They are not "no longer have to deal with capacity and OS" frameworks.
From a cloud native viewpoint, serverless is about consuming compute as an on-demand utility. Kubernetes can be a foundation for a serverless experience (for example, with event-driven autoscaling or serverless frameworks), but the pure framework that removes the most operational burden from developers is serverless.

NEW QUESTION # 160

......

Regualer KCNA Update: https://www.examboosts.com/Linux-Foundation/KCNA-practice-exam-dumps.html

P.S. Free & New KCNA dumps are available on Google Drive shared by ExamBoosts: https://drive.google.com/open?id=1-WwEc7iBMOuzQQY0GdClnley-p2WLuO0