KCNA Q&As

Kubernetes and Cloud Native Associate (KCNA)

Pass Linux Foundation KCNA Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.geekcert.com/kcna.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Linux Foundation Official Exam Center

QUESTION 1

There are three Nodes in a cluster, and want to run exactly one replica of a Pod on each Node. Pre-fer to

automatically create a replica on any new Nodes when they are added.

Which Kubernetes re-source should you use?

A. DaemonSet

B. ReplicaSet

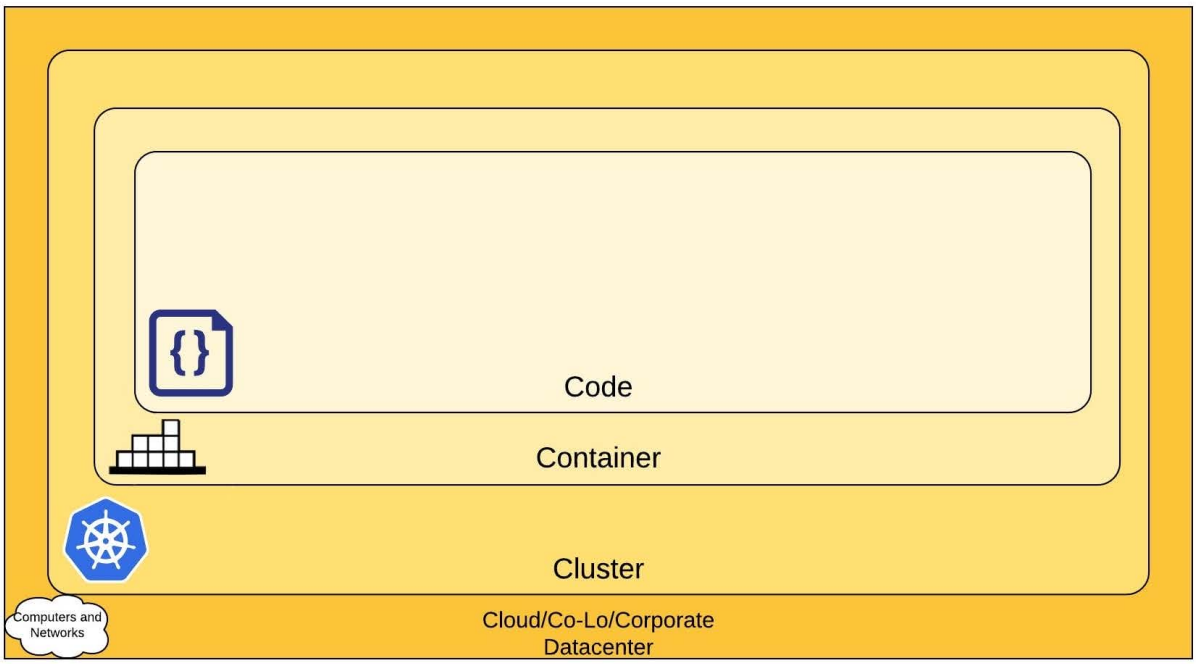
C. NodeSet

D. StatefulSet

E. Deployment

Correct Answer: A

https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/ A DaemonSet runs replicas on all (or just some) Nodes in the cluster.


Code
Container
Cluster
Computers and Networks
Cloud/Co-Lo/Corporate Datacenter

QUESTION 2

The Kubernetes rolling update is used for __.

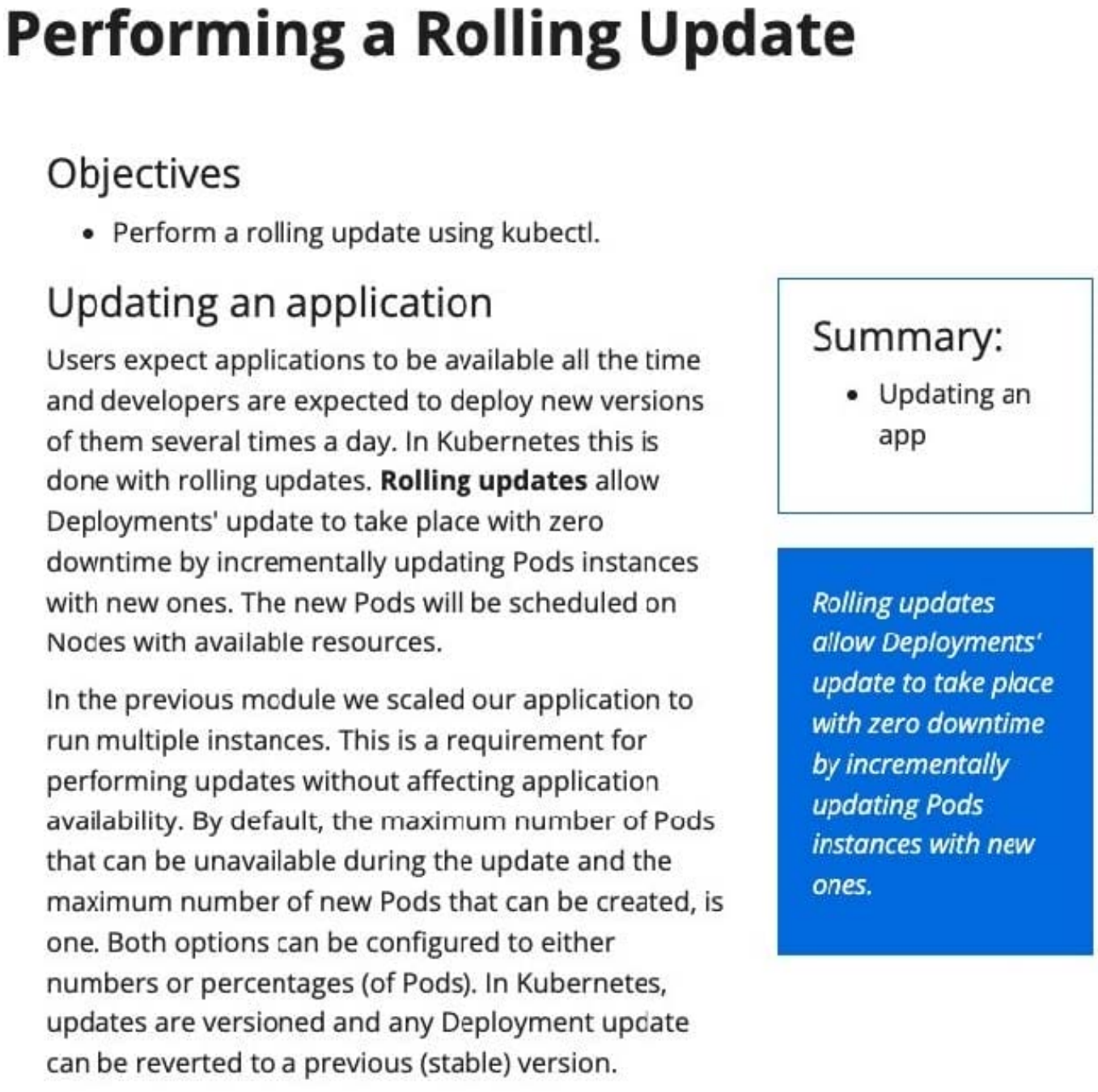
A. Updating a service

B. Scaling an application

C. Updating a deployment

Correct Answer: C

Explanation: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/



QUESTION 3

Which statement is true about Pod Networking?

A. All pod requires an external DNS server to get the hostname

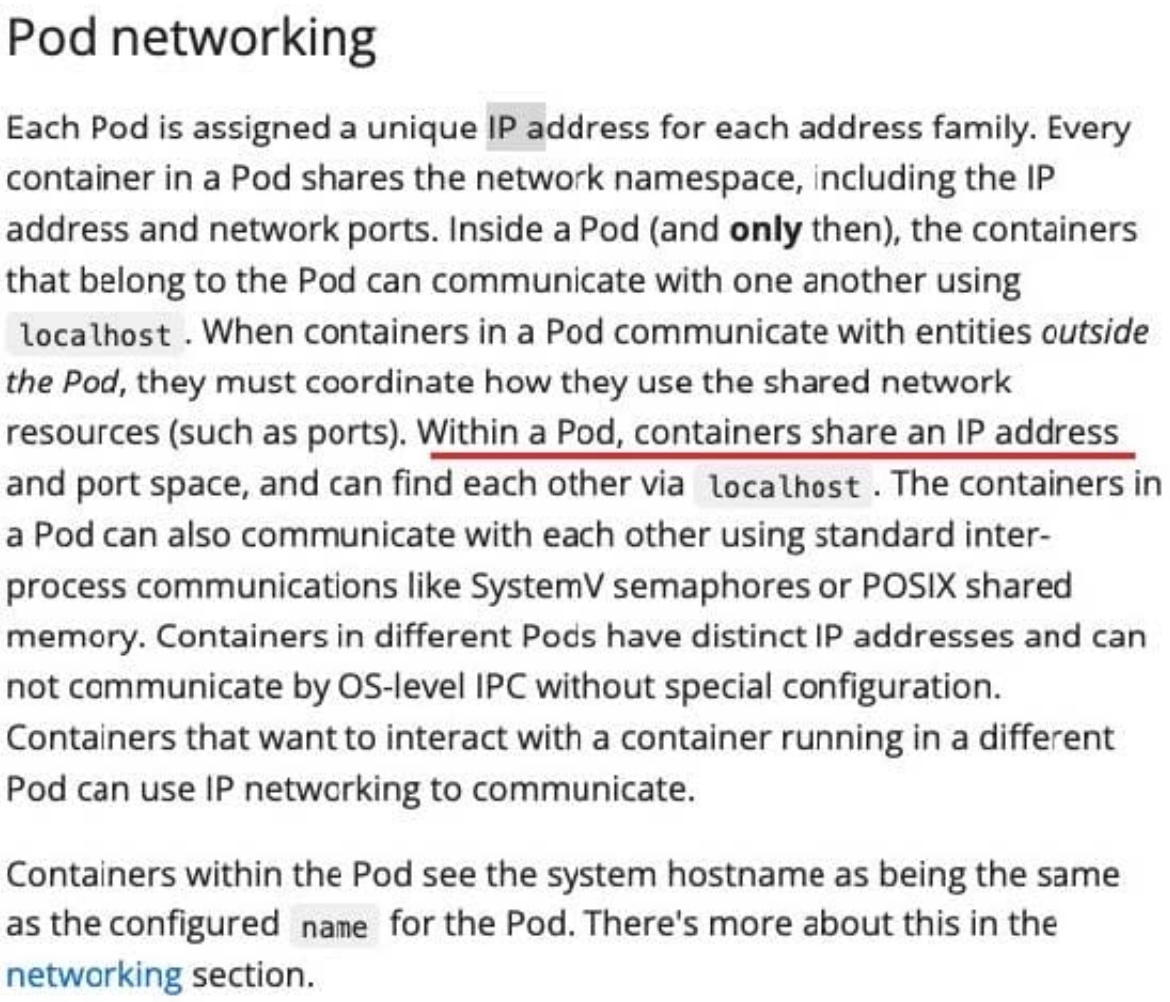
B. All containers in a pod get a unique IP address

C. All containers in a pod share a single IP address

D. All pod requires NAT to get a unique IP address.

Correct Answer: C

Explanation: https://kubernetes.io/docs/concepts/workloads/pods/#pod-networking



Pod networking

Each Pod is assigned a unique IP address for each address family. Every container in a Pod shares the network namespace, including the IP address and network ports. Inside a Pod (and **only** then), the containers that belong to the Pod can communicate with one another using `localhost`. When containers in a Pod communicate with entities *outside the Pod*, they must coordinate how they use the shared network resources (such as ports). Within a Pod, containers share an IP address and port space, and can find each other via `localhost`. The containers in a Pod can also communicate with each other using standard inter-process communications like SystemV semaphores or POSIX shared memory. Containers in different Pods have distinct IP addresses and can not communicate by OS-level IPC without special configuration. Containers that want to interact with a container running in a different Pod can use IP networking to communicate.

Containers within the Pod see the system hostname as being the same as the configured `name` for the Pod. There's more about this in the networking section.

QUESTION 4

What is the name for a service that has no clusterIp address?

A. Headless

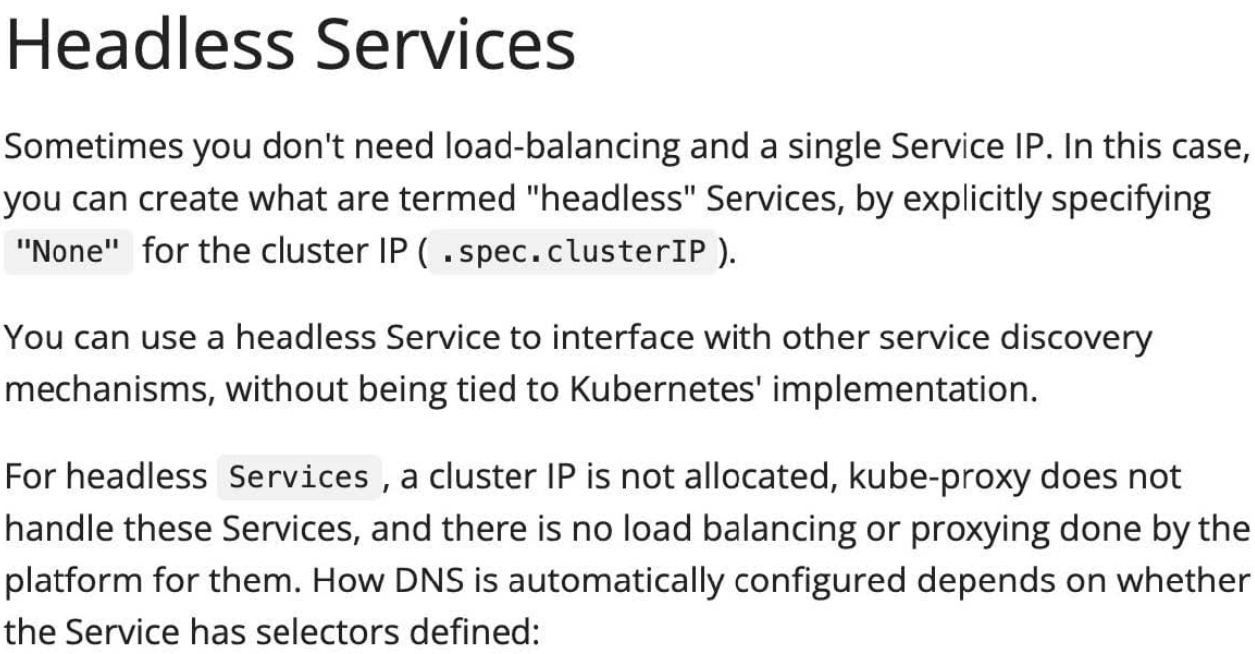
B. NodePort

C. ClusterIP

D. LoadBalancer

Correct Answer: A

Explanation: https://kubernetes.io/docs/concepts/services-networking/service/#headless- services



QUESTION 5

What is horizontal scaling?

A. Creating a Deployment

B. Adding resources to existing apps and servers

C. Moving workloads from one server to another

D. Adding additional replicas of apps and servers

Correct Answer: D

Explanation: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

In Kubernetes, a *HorizontalPodAutoscaler* automatically updates a workload resource (such as a Deployment or StatefulSet), with the aim of automatically scaling the workload to match demand.

Horizontal scaling means that the response to increased load is to deploy more Pods. This is different from *vertical* scaling, which for Kubernetes would mean assigning more resources (for example: memory or CPU) to the Pods that are already running for the workload.

If the load decreases, and the number of Pods is above the configured minimum, the HorizontalPodAutoscaler instructs the workload resource (the Deployment, StatefulSet, or other similar resource) to scale back down.

Horizontal pod autoscaling does not apply to objects that can't be scaled (for example: a DaemonSet.)

The HorizontalPodAutoscaler is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The horizontal pod autoscaling controller, running within the Kubernetes control plane, periodically adjusts the desired scale of its target (for example, a Deployment) to match observed metrics such as average CPU utilization, average memory utilization, or any other custom metric you specify.