KUBERNETES: PODS, NAMESPACES AND REPLICASETS**Introduction & Learning Objectives**

# Chapter Overview

KUBERNETES: PODS, NAMESPACES AND REPLICASETS Knowledge Check Congratulations on completing Chapter 11 - Kubernetes: Pods, Namespaces and ReplicaSets. Take this quiz to check your understanding of the concepts you've learned about so far. Question 11.1 Compute resources isolation is achieved in Kubernetes by \_\_\_\_\_\_\_\_\_. Fill in the blank. A. Namespaces B. iptables C. Firewall D. Hard disk partitioning

ChatGPT

A. Namespaces

Compute resources isolation in Kubernetes is achieved using Namespaces. Namespaces provide a way to create virtual clusters within a physical Kubernetes cluster. By default, all resources created in Kubernetes are in the "default" namespace, but you can create additional namespaces to logically isolate and separate resources.

Each namespace has its own set of resources, including pods, services, replica sets, and more. This allows different teams or projects to use the same Kubernetes cluster while keeping their resources isolated from each other. Namespaces provide a mechanism for resource quota and access control, ensuring that different namespaces cannot interfere with each other's resources and providing a level of resource isolation.



Question 11.2 Which of the following is a unit of development in Kubernetes? A. Pod B. Containers C. Node D. Deployment

ChatGPT

A. Pod

A Pod is the smallest and simplest unit in the Kubernetes object model. It represents a single instance of a running process in a cluster. A Pod encapsulates one or more containers, storage resources, a unique network IP, and options that govern how the containers should run.

Containers within a Pod share the same network namespace, which means they can communicate with each other using localhost. They can also share the same storage volumes, allowing them to share data.

Pods are used to deploy and manage applications in Kubernetes. They provide a way to group containers together as a single cohesive unit and represent the basic building blocks of applications in the Kubernetes ecosystem.



Question 11.3 Which of the following is not shared between containers in the same pod? A. Network B. Hostname C. Volumes D. PID

ChatGPT

D. PID

The PID (Process ID) namespace is not shared between containers in the same pod.

In Kubernetes, each container in a Pod has its own separate PID namespace. This means that the processes running inside the containers have their unique process IDs, and they cannot see or interact with processes outside their own container or other containers within the same Pod.

On the other hand, the network namespace is shared between containers in the same Pod. This allows containers within the Pod to communicate with each other using the same network interface and IP address, and they can also communicate via **localhost**.

Similarly, containers in the same Pod can share the same hostname, and they can also share volumes, which allows them to share data and files.

The PID namespace isolation ensures that containers in a Pod remain isolated from each other's processes, providing a level of separation and security.



Question 11.4 Two containers in the same pod can be scheduled across different nodes. True or False? A. True B. False

ChatGPT

B. False

Two containers within the same Pod are co-located on the same node. When a Pod is scheduled to run on a specific node in the Kubernetes cluster, all the containers within that Pod will be placed on the same node.

The main reason for this co-location is to allow the containers within the Pod to share the same network namespace and communicate with each other using **localhost**. Additionally, containers in the same Pod can share the same volumes, which enables them to share data and files.

Kubernetes makes sure that all the containers within a Pod are co-located on the same node for efficient communication and resource sharing. If you need to run containers on different nodes, you would create separate Pods for each container, and Kubernetes will schedule them on different nodes as needed.



Question 11.5 Which of the following does not belong to the AKMS group? A. apiVersion B. Metadata C. Spec D. Containers

ChatGPT

D. Containers

In the context of Kubernetes, the "AKMS" group is not a standard terminology or grouping. However, I can break down the options to clarify their relevance in Kubernetes:

A. **apiVersion**: This is a field used in Kubernetes resource definition files to specify the version of the Kubernetes API being used. It is used to ensure compatibility between the resource definition and the Kubernetes API server.

B. **Metadata**: This is a section in Kubernetes resource definition files that contains metadata about the resource, such as the name, labels, annotations, and other identifying information.

C. **Spec**: This is a section in Kubernetes resource definition files that defines the desired state of the resource. It contains the specifications for the resource, such as the desired number of replicas, container images, ports, and other settings.

D. **Containers**: This term is related to Kubernetes but is not part of any specific group like AKMS. Containers are the main building blocks in Kubernetes Pods, and they define the application or services running within the Pod.

In summary, the options A, B, and C are valid concepts in Kubernetes, while option D is not a standard grouping in Kubernetes terminology.



Question 11.6 Replication controller achieves high availability based on labels and selectors. True or False? A. True B. False

ChatGPT

A. True

Yes, that's correct. A Replication Controller (or its successor, the ReplicaSet) in Kubernetes achieves high availability based on labels and selectors.

Here's how it works:

1. **Labels:** Labels are key-value pairs attached to Kubernetes objects like Pods, ReplicaSets, and Services. They are used to identify and organize objects into groups.
2. **Selectors:** A Selector is a way to filter and select objects based on their labels. Replication Controllers (and ReplicaSets) use selectors to determine which Pods they are responsible for managing.
3. **Desired State:** The Replication Controller/ReplicaSet maintains the desired number of replicas of a particular Pod template. It ensures that the actual number of running Pods matches the desired number by creating or deleting Pods as necessary.
4. **High Availability:** By specifying the desired number of replicas and using labels and selectors to manage them, the Replication Controller/ReplicaSet ensures high availability. If a Pod becomes unavailable or fails for any reason, the Replication Controller/ReplicaSet will automatically create a replacement Pod to maintain the desired number of replicas.
5. **Scalability:** Additionally, the Replication Controller/ReplicaSet can also scale the number of replicas up or down to handle increased or decreased demand for the application.

This mechanism allows Kubernetes to maintain a desired level of availability and resilience for applications running in the cluster. If a Pod fails or a node goes down, the Replication Controller/ReplicaSet will create new Pods on available nodes, ensuring that the application continues to run and meet the desired replica count.



Question 11.7 What is a node? A. A machine which runs Kubernetes workload B. A machine which stores Kubernetes cluster state C. Replication of a Kubernetes server-side components D. Kubernetes agent which runs on VMs

ChatGPT

A. A machine which runs Kubernetes workload

Option A is correct. In Kubernetes, a "node" refers to a physical or virtual machine (VM) that is part of the Kubernetes cluster and is capable of running containers. Each node is a worker machine that runs application workloads and other Kubernetes components necessary to manage containers.

Nodes are responsible for running containers and providing the runtime environment for the workloads deployed in the cluster. They are the actual computing resources where the containers are deployed and executed.

A node is a combination of both hardware and software components. The software components include the kubelet (Kubernetes agent running on the node), container runtime (such as Docker or containerd), and other necessary services to communicate with the Kubernetes control plane.

Kubernetes clusters consist of multiple nodes, and the control plane (master components) manages and orchestrates the containers running on these nodes. When you deploy an application to Kubernetes, the pods containing the application's containers are scheduled to run on one of the available nodes in the cluster.



Question 11.8 Which of the following commands should be used to list the pods in a kube-system namespace? A. kubectl get containers -n kube-system B. kubectl get pods C. kubectl get pods -n kube-system D. kubectl list pods -n kube-system

ChatGPT

C. kubectl get pods -n kube-system

The correct command to list the pods in a specific namespace (in this case, the kube-system namespace) is:

shCopy code

kubectl get pods -n kube-system

The **-n** option is used to specify the namespace, and **kube-system** is the namespace where Kubernetes system components and add-ons are typically deployed. Running this command will display a list of all pods running in the **kube-system** namespace.



Question 11.9 How do you run commands inside a running pod? A. kubectl run -itd <POD-NAME> B. kubectl launch -itd <POD-NAME> C. kubectl run -itd <POD-NAME> <COMMAND> D. kubectl exec -it <POD-NAME> <COMMAND>

ChatGPT

D. kubectl exec -it <POD-NAME> <COMMAND>

The correct command to run commands inside a running pod is:

shCopy code

kubectl exec -it <POD-NAME> <COMMAND>

The **exec** command is used to execute a command inside a running container in a pod. The **-it** option is used to make the command interactive by allocating a pseudo-TTY and connecting it to the standard input, output, and error streams of the container.

Replace **<POD-NAME>** with the name of the pod where you want to run the command, and **<COMMAND>** with the command you want to execute inside the container.

For example, to run a shell (e.g., bash) inside a pod named "my-pod," you would use:

shCopy code

kubectl exec -it my-pod bash



Question 11.10 If you delete a pod created without ReplicaSet, it will be created again. True or False? A. True B. False

ChatGPT

B. False

If you delete a pod that was created directly without a ReplicaSet or other controllers managing it, it will not be automatically recreated by Kubernetes. When a pod is created without a controller, Kubernetes does not have a mechanism to ensure its desired state, so it won't automatically recover the pod if it is deleted.

ReplicaSets and Deployments are controllers that manage pods and ensure the desired number of replicas are running. If you use a ReplicaSet or a Deployment to manage your pods, Kubernetes will automatically recreate any deleted pods to maintain the desired replica count.

So, if you want automatic recovery of pods, it's recommended to use a controller like ReplicaSet or Deployment to manage your pods. If a pod managed by a ReplicaSet or Deployment is deleted, the controller will detect the change and create a new pod to replace it, ensuring the desired number of replicas is maintained.



Question 11.11 Scaling is not possible using ReplicaSets. True or False? A. True B. False

ChatGPT

B. False

Scaling is indeed possible using ReplicaSets. A ReplicaSet in Kubernetes is designed to ensure that a specified number of replicas (pods) of a pod template are running at all times. It enables horizontal scaling by automatically increasing or decreasing the number of replicas to match the desired count.

You can scale a ReplicaSet by either increasing or decreasing the number of replicas using the **kubectl scale** command. For example, to scale up a ReplicaSet named "my-replicaset" to have three replicas, you would use the following command:

shCopy code

kubectl scale --replicas=3 replicaset my-replicaset

Similarly, you can scale it down by specifying a lower number of replicas.

ReplicaSets are often used together with Deployments to manage and scale applications in a Kubernetes cluster. Deployments provide a higher-level abstraction for managing ReplicaSets and offer additional features like rolling updates and versioning.



Question 11.12 If you delete a ReplicaSet, the pods that are controlled by this ReplicaSet will also get deleted. True or False? A. True B. False

ChatGPT

A. True

When you delete a ReplicaSet in Kubernetes, the pods that are controlled by that ReplicaSet will also get deleted. This is because the primary purpose of a ReplicaSet is to ensure that a specified number of replicas (pods) of a pod template are running at all times. When the ReplicaSet is deleted, the desired state is no longer being maintained, and Kubernetes will terminate the pods that were managed by that ReplicaSet.

Deleting a ReplicaSet is typically done through the **kubectl delete replicaset <replicaset-name>** command. When you execute this command, Kubernetes will take care of terminating the associated pods.

If you want to delete a ReplicaSet and its pods, you can use the **--cascade=false** option to prevent the associated pods from being deleted. For example:

shCopy code

kubectl delete replicaset <replicaset-name> --cascade=false

In this case, the ReplicaSet will be deleted, but the pods will be left running and will no longer be managed by the ReplicaSet.