

Kubernetes Concepts Quiz (100 Questions)

DevOps Learning Module

This quiz covers fundamental concepts related to Kubernetes (K8s) orchestration. Choose the best answer for each question.

1. What is Kubernetes (K8s)?

- A. A container runtime (like Docker).
- B. An open-source container orchestration platform.
- C. A configuration management tool (like Ansible).
- D. A cloud provider (like AWS).

Answer: B

Explanation: Kubernetes is a platform for automating the deployment, scaling, and management of containerized applications.

2. What is the smallest deployable unit in Kubernetes?

- A. A Container
- B. A Pod
- C. A Service
- D. A Node

Answer: B

Explanation: A Pod is the smallest unit. It represents a single instance of an application and can contain one or more tightly coupled containers.

3. What is a "Node" in Kubernetes?

- A. A master machine that runs the control plane.
- B. A worker machine (VM or physical) in a cluster where Pods are run.
- C. A single Pod.
- D. The entire cluster.

Answer: B

Explanation: A Node is a worker machine that runs the `kubelet` and a container runtime, and is responsible for running Pods.

4. What is the "Control Plane" in Kubernetes?

- A. The set of all worker Nodes.

- B. The networking layer of the cluster.
- C. The set of components that manage the cluster (e.g., API server, `etcd`, scheduler).
- D. A single Pod.

Answer: C

Explanation: The Control Plane is the "brain" of the cluster. It makes global decisions (like scheduling) and responds to cluster events.

5. What is the primary component of the Control Plane, which exposes the Kubernetes API?
- A. `etcd`
 - B. `kube-scheduler`
 - C. `kube-apiserver`
 - D. `kubelet`

Answer: C

Explanation: The `kube-apiserver` is the frontend for the control plane. It's the only component that `kubectl` and other components talk to directly.

6. What is `etcd`?
- A. A container runtime.
 - B. A networking plugin.
 - C. A consistent and highly-available key-value store used as the cluster's database.
 - D. A DNS server.

Answer: C

Explanation: `etcd` is the database for Kubernetes. It stores all cluster data (Pod definitions, Services, Deployments, etc.).

7. What is the role of the `kube-scheduler`?
- A. It watches for new Pods and assigns them to a specific Node to run on.
 - B. It runs Pods on the Nodes.
 - C. It stores the cluster state.
 - D. It manages network traffic.

Answer: A

Explanation: The scheduler is responsible for the critical task of deciding which Node is best suited to run a newly created Pod based on resources, constraints, etc.

8. What is the `kubelet`?
- A. A component of the Control Plane.
 - B. An agent that runs on each Node and ensures that containers are running in a Pod.
 - C. The command-line tool (CLI) for Kubernetes.

D. The cluster's DNS service.

Answer: B

Explanation: The `kubelet` is the primary agent on each Node. It receives Pod definitions from the API server and manages their lifecycle on that Node.

9. What is the `kube-proxy`?

A. A proxy for the API server.

B. A network proxy that runs on each Node, maintaining network rules to allow communication to Pods.

C. A proxy for `etcd`.

D. A security component.

Answer: B

Explanation: `kube-proxy` is responsible for implementing the Kubernetes `Service` concept on each Node, typically by managing `iptables` rules.

10. What is the command-line tool for interacting with a Kubernetes cluster?

A. `docker`

B. `k8s-cli`

C. `kubectl`

D. `kube-admin`

Answer: C

Explanation: `kubectl` (kube-control) is the primary CLI tool used by administrators and users to interact with the Kubernetes API.

11. What is a Kubernetes "Service"?

A. A single, long-running Pod.

B. A set of instructions for building a container.

C. An abstract way to expose an application running on a set of Pods as a network service.

D. A worker Node.

Answer: C

Explanation: Pods are ephemeral (they can be destroyed and replaced). A `Service` provides a stable, single IP address and DNS name to access a group of Pods.

12. How does a `Service` find the Pods it should route traffic to?

A. By using `NodePort`.

B. By using `Labels` and `Selectors`.

C. By manually listing Pod IPs.

D. By using an `Ingress`.

Answer: B

Explanation: A `Service` definition includes a `selector` (e.g., `app: my-app`). It automatically finds all Pods that have a matching `label` (e.g., `app: my-app`).

13. What is a "Deployment"?

- A. The act of running `kubectl apply`.
- B. A resource object that provides declarative updates for Pods and `ReplicaSets`.
- C. A single Pod.
- D. A component of the Control Plane.

Answer: B

Explanation: A `Deployment` is a "controller" that manages a `ReplicaSet`. You declare your desired state (e.g., "I want 3 replicas of `nginx:1.21`"), and the `Deployment` makes it happen.

14. What is a "ReplicaSet"?

- A. A set of worker Nodes.
- B. A resource that ensures a specified number of Pod "replicas" are running at any given time.
- C. A backup of `etcd`.
- D. A type of `Service`.

Answer: B

Explanation: A `ReplicaSet` is a simple controller whose only job is to maintain a stable set of Pods. You almost never create them directly; `Deployments` manage them for you.

15. What is the recommended way to manage application scaling and rolling updates?

- A. By creating Pods manually.
- B. By creating `ReplicaSets` manually.
- C. By creating a `Deployment`.
- D. By creating a `Service`.

Answer: C

Explanation: `Deployments` are the standard, high-level resource for managing stateless applications. They handle rolling updates, rollbacks, and scaling.

16. What is a "Rolling Update"?

- A. Restarting all Pods at once.
- B. The process of slowly replacing old Pods with new ones, one by one or in batches.
- C. Updating the worker Nodes.
- D. Updating `etcd`.

Answer: B

Explanation: This is a key feature of **Deployments**. It ensures zero-downtime updates by gradually introducing new Pods while gradually terminating old ones.

17. What `kubectl` command is used to apply a configuration file?

- A. `kubectl run -f deployment.yml`
- B. `kubectl create -f deployment.yml`
- C. `kubectl apply -f deployment.yml`
- D. `kubectl set -f deployment.yml`

Answer: C

Explanation: `kubectl apply` is the declarative, idempotent command. It creates the resource if it doesn't exist or **updates** it if it does. `kubectl create` will fail if it already exists.

18. What `kubectl` command is used to get a list of all Pods in the current namespace?

- A. `kubectl list pods`
- B. `kubectl get pods`
- C. `kubectl ps`
- D. `kubectl get all`

Answer: B

Explanation: `kubectl get` is the standard command for retrieving information about resources.

19. What `kubectl` command shows detailed information about a **specific** Pod?

- A. `kubectl get pod <pod-name>`
- B. `kubectl logs <pod-name>`
- C. `kubectl describe pod <pod-name>`
- D. `kubectl inspect pod <pod-name>`

Answer: C

Explanation: `kubectl describe` provides a human-readable, detailed view, including its current state, events, and configuration.

20. What `kubectl` command is used to view the logs of a running Pod?

- A. `kubectl describe pod <pod-name>`
- B. `kubectl exec <pod-name> - cat /var/log/app.log`
- C. `kubectl logs <pod-name>`
- D. `kubectl get logs <pod-name>`

Answer: C

Explanation: `kubectl logs` streams the `stdout/stderr` from the container(s) in the Pod. Use `-f` to "follow" the logs.

21. What `kubectl` command allows you to get a shell **inside** a running container?

- A. `kubectl attach <pod-name>`
- B. `kubectl ssh <pod-name>`
- C. `kubectl exec -it <pod-name> - /bin/bash`
- D. `kubectl run shell <pod-name>`

Answer: C

Explanation: `kubectl exec` executes a command **inside** the container. `-it` makes it interactive and allocates a terminal (TTY).

22. What is a "Namespace"?

- A. A physical cluster.
- B. A virtual cluster inside a physical cluster, used to partition resources.
- C. A type of Pod.
- D. A network policy.

Answer: B

Explanation: Namespaces are a way to divide cluster resources between multiple users or teams (e.g., `dev`, `staging`, `prod`).

23. What command lists all Pods in **all** namespaces?

- A. `kubectl get pods -all`
- B. `kubectl get pods -all-namespaces`
- C. `kubectl get pods -A`
- D. Both B and C.

Answer: D

Explanation: `-A` is the shorthand for the `-all-namespaces` flag.

24. What is the default namespace for objects created without specifying one?

- A. `kube-system`
- B. `default`
- C. `kube-public`
- D. `none`

Answer: B

Explanation: The `default` namespace is used unless another is specified. `kube-system` is reserved for Control Plane components.

25. What type of **Service** exposes the service on each Node's IP at a static port?

- A. **ClusterIP**
- B. **NodePort**
- C. **LoadBalancer**
- D. **ExternalName**

Answer: B

Explanation: A **NodePort** service makes the service accessible from outside the cluster (for testing, etc.) by opening a specific port on **all** Nodes.

26. What is the default **Service** type if none is specified?

- A. **ClusterIP**
- B. **NodePort**
- C. **LoadBalancer**
- D. **None**

Answer: A

Explanation: **ClusterIP** exposes the service on an internal-only IP. This is the most common type for internal, cluster-only communication.

27. What **Service** type provisions an external load balancer (e.g., an ELB on AWS) from a cloud provider?

- A. **ClusterIP**
- B. **NodePort**
- C. **LoadBalancer**
- D. **ExternalName**

Answer: C

Explanation: This is the standard way to expose a service to the public internet on a cloud platform. It automatically provisions and configures a cloud load balancer.

28. What is an "Ingress"?

- A. A type of **Service**.
- B. An API object that manages external access (HTTP/HTTPS) to services in a cluster, typically handling hostnames and paths.
- C. A networking plugin (CNI).
- D. A security policy.

Answer: B

Explanation: An **Ingress** is a "level 7" (HTTP) load balancer. It allows you to define rules like "traffic for **foo.com** goes to **service-foo**, traffic for **bar.com** goes to **service-bar**".

29. What component is required to make an **Ingress** resource work?

- A. An **Ingress Controller** (e.g., NGINX, Traefik).
- B. A **LoadBalancer** service.
- C. A **NodePort** service.
- D. kube-proxy.

Answer: A

Explanation: The **Ingress** *resource* is just a definition. You must also install an **Ingress Controller**, which is the actual software (Pod) that reads those definitions and does the proxying.

30. What is a "ConfigMap"?

- A. A resource for storing secret data.
- B. A resource for storing non-confidential configuration data as key-value pairs.
- C. A map of the cluster's network.
- D. A resource for defining a Deployment.

Answer: B

Explanation: **ConfigMaps** are used to decouple configuration from your Pods. You can inject the data into Pods as environment variables or as files.

31. What is a "Secret"?

- A. The same as a **ConfigMap**, but for confidential data.
- B. A resource for storing data (like passwords, API keys) in a base64-encoded format.
- C. A special type of **Service**.
- D. Both A and B.

Answer: D

Explanation: **Secrets** are functionally very similar to **ConfigMaps** but are intended for sensitive data. Kubernetes stores them as base64 (which is encoding, *not* encryption).

32. How can you inject **ConfigMap** or **Secret** data into a Pod?

- A. As environment variables.
- B. As mounted files (as a volume).
- C. As command-line arguments.
- D. Both A and B.

Answer: D

Explanation: You can inject keys as environment variables (**envFrom**) or mount the entire **ConfigMap/Secret** as a directory of files (**volumes**).

33. What is a "Volume" in Kubernetes?

- A. A Docker Volume.

- B. A directory of data that exists **outside** the container's filesystem and can be mounted into a Pod.
- C. A block storage device.
- D. An image layer.

Answer: B

Explanation: A K8s **Volume** is a storage abstraction. Unlike a Docker volume, it is tied to the **Pod's** lifecycle, not the container's.

34. What is the lifecycle of a **Volume** defined within a Pod spec?

- A. It is destroyed when the container restarts.
- B. It is destroyed when the Pod is deleted.
- C. It persists forever, even after the Pod is deleted.
- D. It is destroyed when the Node reboots.

Answer: B

Explanation: A volume (like **emptyDir**) lives as long as the Pod. If a container in the Pod restarts, the volume's data is preserved.

35. What is a "PersistentVolume" (PV)?

- A. A volume that is defined inside a Pod.
- B. A piece of storage in the cluster (like an EBS volume or NFS share) that has been provisioned by an administrator.
- C. A volume that is persistent.
- D. A backup of **etcd**.

Answer: B

Explanation: A PV is a "cluster resource" for storage, just like a Node is a "cluster resource" for CPU/memory. It exists independently of any Pod.

36. What is a "PersistentVolumeClaim" (PVC)?

- A. A request for storage by a user (a Pod).
- B. The same as a **PersistentVolume**.
- C. A claim on a specific Pod.
- D. A claim on a specific Node.

Answer: A

Explanation: A user/Pod creates a PVC (e.g., "I need 10Gi of fast storage"). Kubernetes then **binds** this claim to a matching, available PV.

37. What is the relationship between PV, PVC, and Pod?

- A. Pod -> mounts -> PV -> requests -> PVC
- B. PV -> claims -> PVC -> mounts -> Pod

C. Pod -> claims -> PVC -> binds to -> PV

D. PV -> binds to -> Pod -> claims -> PVC

Answer: C

Explanation: The Pod claims a PVC. The PVC is then bound to a PV by the control plane. The Pod then mounts the PVC (which is backed by the PV).

38. What is a "StatefulSet"?

A. A **Deployment** with a PV.

B. A controller used to manage stateful applications (like databases) that require stable, unique network identifiers and stable, persistent storage.

C. A Pod that cannot be deleted.

D. A set of worker Nodes.

Answer: B

Explanation: **StatefulSets** provide guarantees (e.g., stable hostnames like db-0, db-1) that **Deployments** do not, which is critical for clustered applications.

39. What is a "DaemonSet"?

A. A Pod that runs as **root**.

B. A controller that ensures a copy of a Pod runs on **all** (or some) Nodes in the cluster.

C. A **StatefulSet** with only one replica.

D. A Control Plane component.

Answer: B

Explanation: This is used for cluster-level services like log collectors (e.g., Fluentd) or monitoring agents (e.g., Prometheus Node Exporter) that **must** run on every Node.

40. What is a "Job"?

A. A **Deployment** that has finished.

B. A resource that creates one or more Pods and ensures they run to completion (and **do not** restart).

C. A **kubelet** task.

D. A monitoring alert.

Answer: B

Explanation: A **Job** is for "batch" work. It runs a task once. A **Deployment** is for "services" that should run forever.

41. What is a "CronJob"?

A. A **Job** that runs inside a **DaemonSet**.

B. A resource that creates **Jobs** on a repeating schedule (like a cron syntax).

C. A **Job** that has failed.

D. A system-level cron.

Answer: B

Explanation: A `CronJob` is the K8s-native way to run scheduled tasks (e.g., "run a backup job at 1 AM every night").

42. What is a "Label"?

- A. Key/value pairs attached to objects (like Pods) used for identification and grouping.
- B. A metadata tag.
- C. A network address.
- D. A volume name.

Answer: A

Explanation: Labels are the core grouping mechanism. They are non-unique and are used by `Selectors` (e.g., `app: nginx, env: prod`).

43. What is a "Selector"?

- A. A way to choose a Node.
- B. A query used by controllers (like `Services` or `ReplicaSets`) to find objects with matching `Labels`.
- C. A way to choose a `ConfigMap`.
- D. A user.

Answer: B

Explanation: A `Service` "selects" its Pods using a Label Selector. This decouples the `Service` from the Pods themselves.

44. What is an "Annotation"?

- A. The same as a `Label`.
- B. A comment in a YAML file.
- C. Key/value pairs used to attach arbitrary, non-identifying metadata to objects.
- D. A security policy.

Answer: C

Explanation: Annotations are for "extra" information for tools or humans (e.g., `prometheus.io/scrape: 'true'`). Unlike Labels, they cannot be used by Selectors.

45. What is a "Taint"?

- A. A property applied to a `*Node*` that "repels" Pods.
- B. A property applied to a `*Pod*` that "repels" Nodes.
- C. A failed Pod.
- D. A security vulnerability.

Answer: A

Explanation: A **Taint** (e.g., `gpu=true:NoSchedule`) is placed on a Node. By default, no Pod will be scheduled on that Node.

46. What is a "Toleration"?

- A. A property applied to a `*Node*` that "attracts" Pods.
- B. A property applied to a `*Pod*` that allows it to "tolerate" (and be scheduled on) a Node with a matching **Taint**.
- C. A setting for resource limits.
- D. A successful health check.

Answer: B

Explanation: Taints and Tolerations work together. A Pod must have a **Toleration** for a Node's **Taint** to be scheduled there.

47. What is "Node Affinity"?

- A. A set of rules on a `*Pod*` that "prefers" or "requires" it to run on Nodes with certain **Labels**.
- B. A set of rules on a `*Node*` that "prefers" or "requires" certain Pods.
- C. The same as Taints/Tolerations.
- D. A networking rule.

Answer: A

Explanation: This is the "attraction" mechanism. `nodeAffinity` on a Pod says "I **want** to run on a Node with the label `disk: ssd`".

48. What is the difference between `requiredDuringScheduling...` and `preferredDuringScheduling...` affinity?

- A. **required** is a hard rule (must match); **preferred** is a soft rule (a suggestion).
- B. **preferred** is a hard rule; **required** is a soft rule.
- C. **required** is for Taints; **preferred** is for Affinity.
- D. There is no difference.

Answer: A

Explanation: **required** (hard) means the scheduler **will not** schedule the Pod unless a matching Node is found. **preferred** (soft) means the scheduler will **try** to find a matching Node, but will schedule it elsewhere if it can't.

49. What is "Pod Anti-Affinity"?

- A. A rule that prevents a Pod from running on a Node.
- B. A rule on a `*Pod*` that tells the scheduler to **avoid** scheduling it on a Node where other specific Pods are already running.
- C. A rule to delete Pods.

D. A Taint.

Answer: B

Explanation: This is critical for high availability. You use anti-affinity to ensure your 3 web server replicas don't all get scheduled onto the same single Node.

50. What is a "Liveness Probe"?

- A. A check to see if a Pod is ready to accept traffic.
- B. A check to see if a container is still "alive" (healthy). If it fails, the `kubelet` will restart the container.
- C. A check to see if a Node is alive.
- D. A security scan.

Answer: B

Explanation: The Liveness Probe (e.g., an HTTP GET) answers the question: "Is my application still running correctly?" If it fails, K8s assumes it's deadlocked and restarts it.

51. What is a "Readiness Probe"?

- A. A check to see if a container is alive.
- B. A check to see if a container is "ready" to start serving traffic. If it fails, the `Service` will **not** send traffic to this Pod.
- C. A check that runs before the container starts.
- D. A check to see if the Node is ready.

Answer: B

Explanation: The Readiness Probe answers: "Is my app ready for new connections?" This is used to prevent traffic from being sent to a Pod that is still starting up.

52. What is a "Startup Probe"?

- A. The same as a Liveness Probe.
- B. A check used for slow-starting containers. It "disables" the Liveness Probe until the app has successfully started one time.
- C. A check that runs during the `docker build`.
- D. A check that runs before the Pod is scheduled.

Answer: B

Explanation: This probe is essential for applications that take a long time (e.g., 2 minutes) to start, to prevent the Liveness Probe from restarting them prematurely.

53. What is "Helm"?

- A. The Kubernetes Control Plane.
- B. A package manager for Kubernetes.
- C. A CNI plugin.

D. A storage class.

Answer: B

Explanation: Helm is a tool that helps you manage complex K8s applications. It packages a set of resources (Deployments, Services, etc.) into a "Chart" that can be easily installed and configured.

54. What is a "Helm Chart"?

A. A Kubernetes dashboard.

B. A collection of templated YAML files that describe a related set of Kubernetes resources.

C. A single `ConfigMap`.

D. A type of `Service`.

Answer: B

Explanation: A Chart is the "package" in Helm. It contains all the YAML needed to run an application (e.g., the "wordpress" chart contains a Deployment, a Service, a PVC, etc.).

55. What is a "StorageClass"?

A. A type of PV.

B. A type of PVC.

C. A resource that defines a "class" of storage (e.g., `fast-ssd`, `slow-hdd`) and allows for *dynamic provisioning* of PVs.

D. A `ConfigMap` for storage.

Answer: C

Explanation: With a `StorageClass`, you don't need to pre-create PVs. A user just creates a PVC asking for the `fast-ssd` class, and the cloud provider *automatically* provisions a new PV for it.

56. What is a "CNI" (Container Network Interface)?

A. The Kubernetes API.

B. A standard (and plugin system) for how containers get network connectivity in Kubernetes.

C. The `kube-proxy`.

D. A `Service` type.

Answer: B

Explanation: CNI is the interface that lets different networking solutions (like Calico, Flannel, Weave) plug into Kubernetes to provide Pod networking.

57. What is "RBAC" (Role-Based Access Control)?

A. A networking policy.

- B. A resource quota.
- C. A mechanism for controlling **who** (users, groups) can perform **what** (verbs) on **which** (resources) in the cluster.
- D. A storage class.

Answer: C

Explanation: RBAC is the standard security model for K8s. It uses `Roles` and `RoleBindings` (or `ClusterRoles` and `ClusterRoleBindings`).

58. What is the difference between a `Role` and a `ClusterRole`?

- A. A `Role` is for users; a `ClusterRole` is for machines.
- B. A `Role` is namespaced (e.g., "read Pods in `default`"); a `ClusterRole` is cluster-wide (e.g., "read Nodes").
- C. A `ClusterRole` is more powerful.
- D. There is no difference.

Answer: B

Explanation: `Roles` and `RoleBindings` are always **inside** a `Namespace`. `ClusterRoles` and `ClusterRoleBindings` apply to the entire cluster.

59. What does a `RoleBinding` or `ClusterRoleBinding` do?

- A. It defines permissions (the "what").
- B. It defines a user (the "who").
- C. It **binds** a "who" (User, Group, or ServiceAccount) to a "what" (`Role` or `ClusterRole`).
- D. It defines a network.

Answer: C

Explanation: The `Role` defines the "verbs" (get, list, watch) and "resources" (pods). The `RoleBinding` connects a "subject" (user) to that `Role`.

60. What is a "ServiceAccount"?

- A. A user account for a human.
- B. An identity for **processes running inside a Pod** to authenticate with the API server.
- C. An account for `kubectl`.
- D. An admin account.

Answer: B

Explanation: If a Pod needs to talk to the K8s API (e.g., to list other Pods), it uses its `ServiceAccount` to authenticate.

61. What is a "NetworkPolicy"?

- A. A `Service` type.

- B. A resource that defines how Pods are allowed to communicate with each other (a firewall).
- C. A CNI plugin.
- D. An **Ingress** resource.

Answer: B

Explanation: By default, all Pods can talk to all other Pods. **NetworkPolicy** allows you to "lock down" traffic (e.g., "Only **frontend** Pods can talk to **backend** Pods on port 8080").

62. What is `kubectl port-forward`?

- A. It's the same as a **NodePort** service.
- B. It's a command to forward traffic from your local machine to a Pod in the cluster.
- C. It's a way to configure **kube-proxy**.
- D. It's a way to set up an **Ingress**.

Answer: B

Explanation: This is a common debugging tool. `kubectl port-forward pod/my-db 5432:5432` lets you connect to the database in the Pod on your `localhost:5432`.

63. What is a "sidecar" container?

- A. A Pod with only one container.
- B. A secondary container that runs in a Pod to provide a helper function (e.g., logging, proxying) to the main container.
- C. A container that runs on the Node.
- D. A container that has failed.

Answer: B

Explanation: Since all containers in a Pod share the same network namespace, a "sidecar" can sit at `localhost` and proxy, log, or secure traffic for the main app container.

64. What is an "init container"?

- A. A sidecar container.
- B. A container that runs **before** the main app containers in a Pod, and must run to completion.
- C. The first container listed in the Pod spec.
- D. The **pause** container.

Answer: B

Explanation: **InitContainers** are used for setup tasks, like waiting for a database to be ready or pre-populating a volume, before the main application starts.

65. What is the "pause" container?

- A. A container that is stopped.

- B. A container that runs a `sleep` command.
- C. A special, hidden container created by K8s in every Pod to "hold" the Pod's network namespace.
- D. A debugging container.

Answer: C

Explanation: The "pause" container is the first container started in a Pod. It acquires the IP and network namespace, and then all other "real" containers join that namespace.

66. What are "resource requests" in a Pod spec?

- A. The *minimum* amount of CPU/Memory that the scheduler *must* guarantee for the Pod.
- B. The *maximum* amount of CPU/Memory the Pod can use.
- C. The same as a PVC.
- D. The current usage of the Pod.

Answer: A

Explanation: `requests` (e.g., `cpu: "100m"`) are used by the Scheduler. A Node must have at least "100m" CPU free to be eligible to run this Pod.

67. What are "resource limits" in a Pod spec?

- A. The *minimum* amount of CPU/Memory for the Pod.
- B. The *hard maximum* amount of CPU/Memory the container is allowed to use.
- C. A `NetworkPolicy`.
- D. The total resources on the Node.

Answer: B

Explanation: `limits` (e.g., `memory: "512Mi"`) are enforced by the `kubelet`. If a container exceeds its memory limit, it will be killed (`OOMKilled`).

68. What is "OOMKilled"?

- A. A Pod status meaning "Out of Money".
- B. A container status meaning "Out of Memory". The container was killed because it exceeded its memory `limit`.
- C. A Node status.
- D. A failed Job.

Answer: B

Explanation: This is a common failure mode. `kubectl describe pod` will show the `Reason` as `OOMKilled`.

69. What is a "HorizontalPodAutoscaler" (HPA)?

- A. A tool for scaling Nodes.

- B. A controller that automatically scales the number of Pods in a `Deployment` or `StatefulSet` based on metrics (like CPU usage).
- C. A manual scaling command.
- D. A type of `Service`.

Answer: B

Explanation: The HPA is the component responsible for auto-scaling. You create an HPA resource (e.g., "scale `my-app` between 3 and 10 replicas, targeting 80% CPU").

70. What is the "Metrics Server"?

- A. A full monitoring solution like Prometheus.
- B. A lightweight, cluster-wide component that collects resource usage metrics (CPU/RAM) and serves them to the HPA and `kubectl top`.
- C. A logging tool.
- D. The `kube-scheduler`.

Answer: B

Explanation: The Metrics Server is a required component for both `kubectl top` (to show current usage) and the HPA (to make scaling decisions).

71. What is `kubectl top node`?

- A. It lists the most important Nodes.
- B. It shows the current CPU and Memory usage of all Nodes in the cluster.
- C. It re-schedules Pods on the top Node.
- D. It describes the Nodes.

Answer: B

Explanation: This command (along with `kubectl top pod`) gets its data from the Metrics Server.

72. What is a "PodDisruptionBudget" (PDB)?

- A. A resource limit.
- B. A resource that specifies the *minimum* number of replicas that must remain available during a *voluntary disruption* (like a Node drain).
- C. A budget for cloud spending.
- D. A network policy.

Answer: B

Explanation: A PDB tells K8s "You can drain this Node, but only if it doesn't cause my `my-app` deployment to drop below 3 replicas."

73. What is the "Cluster Autoscaler"?

- A. The same as the HorizontalPodAutoscaler (HPA).

- B. A tool that automatically adds or removes *Nodes* from the cluster based on Pod scheduling demands.
- C. A tool for scaling `etcd`.
- D. A tool for scaling `Services`.

Answer: B

Explanation: HPA scales Pods. If there aren't enough Nodes to run the new Pods, the Cluster Autoscaler will provision a new Node (e.g., a new EC2 instance).

74. What is "Kustomize"?

- A. A package manager like Helm.
- B. A template-free way to customize (overlay) Kubernetes YAML files.
- C. A CNI plugin.
- D. A security tool.

Answer: B

Explanation: Kustomize lets you take a "base" set of YAML files and apply "overlays" (e.g., for `dev` vs `prod`) to patch or modify them without using string-based templates. It is built into `kubectl`.

75. How do you use Kustomize with `kubectl`?

- A. `kubectl apply -k <directory>`
- B. `kubectl apply -f <directory>`
- C. `kustomize build | kubectl apply -f -`
- D. Both A and C.

Answer: A

Explanation: The `-k` flag tells `kubectl apply` to look for a `kustomization.yml` file in the specified directory and use it to build the final YAMLs.

76. What is the `apiVersion` field in a YAML file?

- A. The version of your application.
- B. The version of the Kubernetes cluster.
- C. The version of the Kubernetes API that defines this object (e.g., `v1`, `apps/v1`).
- D. The version of `kubectl`.

Answer: C

Explanation: This (along with `kind`) is how the API server knows what kind of object you are trying to create. A Pod is `v1`, while a Deployment is `apps/v1`.

77. What is the `kind` field in a YAML file?

- A. The type of object you are creating (e.g., Pod, Service, Deployment).
- B. The kind of application (e.g., `web`, `db`).

- C. The kind of Node.
- D. The `apiVersion`.

Answer: A

Explanation: The `kind` field is the primary identifier for the resource type.

78. What is `minikube`?

- A. A production-grade Kubernetes installer.
- B. A tool for running a single-node Kubernetes cluster locally on your machine for development.
- C. A lightweight version of `kubectl`.
- D. A cloud provider.

Answer: B

Explanation: `minikube` (along with tools like `kind` or Docker Desktop) is the standard way for developers to run and test against a "real" K8s cluster.

79. What is a "Headless" Service?

- A. A `Service` that has no Pods.
- B. A `Service` with `clusterIP: None`. It does **not** get a stable IP, but instead returns the IPs of **all** its backing Pods.
- C. A `Service` of type `ExternalName`.
- D. A `Service` managed by an `Ingress`.

Answer: B

Explanation: Headless services are used when you want to discover the Pods directly, without a proxy. This is used heavily by `StatefulSets`.

80. What is `kubeadm`?

- A. A tool for administering `etcd`.
- B. A tool for creating and managing Kubernetes clusters (bootstrapping).
- C. The `kubectl` alias.
- D. A CNI plugin.

Answer: B

Explanation: `kubeadm` is a command-line tool that provides the "fast path" for creating a best-practice, production-viable Kubernetes cluster.

81. What is a "Container Runtime Interface" (CRI)?

- A. The CNI.
- B. The interface (API) that the `kubelet` uses to talk to the container runtime (e.g., `containerd`, `cri-o`).
- C. The Docker API.

D. A `NetworkPolicy`.

Answer: B

Explanation: This interface allows Kubernetes to be runtime-agnostic. It doesn't talk to "Docker" directly; it talks to a CRI-compliant runtime.

82. What is `containerd`?

- A. A K8s control plane component.
- B. An industry-standard, CRI-compliant container runtime (the one Docker itself uses).
- C. A tool for building images.
- D. A `Service` type.

Answer: B

Explanation: `containerd` is the container runtime that actually manages starting, stopping, and pulling images. It is the default runtime for most K8s clusters.

83. What does "declarative" mean in the context of Kubernetes?

- A. You tell K8s **how** to do something.
- B. You tell K8s **what** you want (the "desired state"), and K8s figures out how to achieve it.
- C. You run `kubectl create`.
- D. You use YAML.

Answer: B

Explanation: This is the core principle. You `kubectl apply` a file saying "I want 3 replicas." You don't tell it "check if 2, if so, create 1." K8s "reconciles" the "actual state" with your "desired state."

84. What is a "reconciliation loop"?

- A. A `Job` that has failed.
- B. The core pattern of a K8s controller: 1. Get desired state, 2. Get actual state, 3. If (actual != desired), take action, 4. Repeat.
- C. A network loop.
- D. A failed `kubectl apply`.

Answer: B

Explanation: Every controller (Deployment controller, ReplicaSet controller, etc.) runs this "reconciliation loop" to enforce the desired state.

85. What is a "CustomResourceDefinition" (CRD)?

- A. A `ConfigMap` with a custom name.
- B. A way to **extend** the Kubernetes API by adding your own new object types (e.g., `kind: Database`).
- C. A custom `kubectl` command.

D. A `NetworkPolicy`.

Answer: B

Explanation: CRDs are the foundation of the "Operator" pattern. They let you define new K8s-native resources.

86. What is an "Operator"?

A. A human administrator.

B. A piece of software (a controller) that uses a `CRD` to manage a complex application as if it were a native K8s resource.

C. A `ServiceAccount`.

D. A `ClusterRole`.

Answer: B

Explanation: An Operator encodes human operational knowledge. For example, the "Prometheus Operator" lets you create a `kind: Prometheus` object, and the operator knows how to provision the necessary `Deployments`, `ConfigMaps`, etc.

87. What is a "PodSecurityPolicy" (PSP)? (Note: Deprecated)

A. A `NetworkPolicy`.

B. A (now deprecated) cluster-level resource for controlling security-sensitive aspects of Pod creation (e.g., "don't allow `root` containers").

C. A `Secret`.

D. A `ServiceAccount`.

Answer: B

Explanation: PSPs were complex and have been replaced by "Pod Security Admission," which is a built-in admission controller.

88. What is a "SecurityContext"?

A. A `Secret`.

B. A field in the `Pod` or `Container` spec that defines privilege and access control settings (e.g., `runAsUser`, `readOnlyRootFilesystem`).

C. A `NetworkPolicy`.

D. An RBAC `Role`.

Answer: B

Explanation: This is the modern, standard way to secure a Pod by defining its security settings directly in its own manifest.

89. What is the purpose of `runAsUser: 1000` in a `SecurityContext`?

A. It ensures the container runs as the `root` user.

B. It ensures the container's main process runs with User ID 1000 (a non-root user).

C. It assigns 1000m (1 CPU) to the Pod.

D. It gives the Pod 1000 permissions.

Answer: B

Explanation: This is a critical security setting to enforce running as a non-privileged user inside the container.

90. What is `kubectl drain <node-name>`?

A. It deletes the Node.

B. It stops all Pods on the Node.

C. It safely evicts all Pods from a Node (respecting PDBs) and marks the Node as unschedulable.

D. It deletes all Jobs.

Answer: C

Explanation: This is the command used to safely remove a Node from the cluster for maintenance (e.g., a kernel update).

91. What is `kubectl uncordon <node-name>`?

A. It deletes a Node.

B. It marks a Node (that was previously "cordoned" or "drained") as schedulable again.

C. It adds a new Node.

D. It evicts all Pods from a Node.

Answer: B

Explanation: `kubectl cordon` marks a Node as unschedulable (but doesn't evict existing Pods). `uncordon` removes that mark.

92. What is a "Taint" and "Toleration" (again)?

A. Taint (on Node) "repels" Pods; Toleration (on Pod) "allows" it to be scheduled.

B. Taint (on Pod) "repels" Nodes; Toleration (on Node) "allows" it.

C. They are for network security.

D. They are for storage.

Answer: A

Explanation: This is a key scheduling concept. Taints are used to "reserve" nodes (e.g., for the control plane) and Pods must "tolerate" that taint to run there.

93. What is a "Service Mesh"? (e.g., Istio, Linkerd)

A. A CNI plugin.

B. A dedicated infrastructure layer for managing service-to-service communication (e.g., mTLS, traffic shifting, metrics).

C. A Service type.

D. A NetworkPolicy.

Answer: B

Explanation: A Service Mesh typically works by injecting a "sidecar" proxy (like Envoy) into every Pod to control all network traffic.

94. What is "Prometheus"?

- A. A logging tool.
- B. A service mesh.
- C. An open-source monitoring and alerting system, very popular in the K8s ecosystem.
- D. A CNI plugin.

Answer: C

Explanation: Prometheus is the de-facto standard for metrics-based monitoring in Kubernetes.

95. What is "GitOps"?

- A. A way of using `git` inside a Pod.
- B. A practice of using a Git repository as the *single source of truth* for the desired state of the cluster.
- C. A CI/CD tool.
- D. A backup strategy for `etcd`.

Answer: B

Explanation: In GitOps, you don't run `kubectl apply` yourself. You push a change to Git, and an agent in the cluster (like Argo CD) sees the change and applies it.

96. What is "Argo CD"?

- A. A CI (Continuous Integration) tool.
- B. A popular "GitOps" tool (a CD controller) that continuously reconciles the cluster state with a Git repository.
- C. A service mesh.
- D. A CNI plugin.

Answer: B

Explanation: Argo CD is an agent you install in your cluster. You point it at a Git repo, and it automatically applies any YAMLs it finds there, ensuring the cluster matches the repo.

97. What is `kubectl config get-contexts`?

- A. It gets all `ConfigMaps`.
- B. It lists all the cluster "contexts" (cluster + user + namespace) defined in your `kubeconfig` file.
- C. It describes the current context.
- D. It gets all `SecurityContexts`.

Answer: B

Explanation: This is used to list all the different clusters your `kubectl` is configured to talk to.

98. What is `kubectl config use-context <context-name>`?

- A. It deletes a context.
- B. It creates a new context.
- C. It switches your `kubectl` tool to talk to a different cluster (or user/namespace).
- D. It applies a `ConfigMap`.

Answer: C

Explanation: This is the command you run to switch between your `dev`, `staging`, and `prod` clusters.

99. What is an "Admission Controller"?

- A. A `NetworkPolicy`.
- B. A piece of code (webhook) that intercepts requests to the API server *after* authentication. It can validate or mutate the request.
- C. A user.
- D. The `kube-scheduler`.

Answer: B

Explanation: Admission Controllers are a powerful K8s feature. They are "gatekeepers" that can enforce policies (e.g., "All Pods *must* have a `team` label").

100. What is the `-dry-run=client` flag used for in `kubectl`?

- A. It applies the change to the server but doesn't save it.
- B. It runs the command but does nothing.
- C. It *doesn't* send the request to the server. It just prints the object that *would* be sent.
- D. It deletes the object.

Answer: C

Explanation: This is incredibly useful for testing. `kubectl create deployment ... -dry-run=client -o yaml` will *print* the YAML for a Deployment without creating it.