PassQueen
a.k.a dumpsqueen.com

# CloudAssociate (JNCIA-Cloud)

## Juniper JN0-214

**Version Demo**

**Total Demo Questions: 8**

**Total Premium Questions: 86**

**Buy Premium PDF**

https://passqueen.com

support@passqueen.com

# Topic Break Down

| Topic | No. of Questions |
|---|---|
| Topic 1, Cloud Concepts | 45 |
| Topic 2, Cloud Networking | 20 |
| Topic 3, Cloud Security | 4 |
| Topic 4, Cloud Automation and Orchestration | 12 |
| Topic 5, Cloud Operations | 5 |
| Total | 86 |

QUESTION NO: 1

You are asked to run a container in a Kubernetes environment.

What should you do to accomplish this task?

**A.** Create a JINJA2 template for the container and its resources.

**B.** Create a WYSYG definition for the container and its resources.

**C.** Define a YAML manifest for the container and its resources.

**D.** Define an XML configuration for the container and its resources.

**ANSWER: C**

**Explanation:**

Define a YAML manifest for the container and its resources. Kubernetes uses declarative resource definitions: you describe the desired state of an application workload, and the Kubernetes control plane works to create and maintain that state. In practice, a container is normally specified in the `containers` section of a Pod template, most often managed through a workload object such as a Deployment. The manifest identifies the container image and can also define operational settings such as commands, environment variables, ports, volumes, and CPU and memory resource requests or limits.

YAML is the common human-readable representation for Kubernetes API objects, although Kubernetes also accepts JSON. After saving the definition, an administrator can submit it to the cluster with `kubectl apply -f <manifest-file>`. Kubernetes then creates the requested workload and schedules its Pod onto an appropriate node. This declarative approach is fundamental to Kubernetes because the submitted manifest serves as the intended configuration that can be version-controlled, reviewed, reapplied, and updated consistently. The Kubernetes documentation provides a Deployment manifest example and explains how it manages Pods containing application containers: Kubernetes Deployments. For the general format and use of configuration files, see Kubernetes Configuration.

**QUESTION NO: 2**

Which two statements are correct about Kubernetes resources? (Choose two.)

**A.** A ClusterIP type service can only be accessed within a Kubernetes cluster.

**B.** A daemonSet ensures that a replica of a pod is running on all nodes.

**C.** A deploymentConfig is a Kubernetes resource.

**D.** NodePort service exposes the service externally by using a cloud provider load balancer.

**ANSWER: A B**

**Explanation:**

**A ClusterIP type service can only be accessed within a Kubernetes cluster.** is correct because ClusterIP is Kubernetes' default Service type. It assigns the Service an internal virtual IP address that provides stable in-cluster access to a group of matching Pods. Workloads in the cluster can use the Service IP or its DNS name, allowing application components to communicate without depending on individual, changing Pod IP addresses. This makes ClusterIP appropriate for internal application tiers such as databases, APIs, and backend services. Kubernetes documents ClusterIP as exposing a Service on a cluster-internal IP; see the Kubernetes Service documentation.

**A daemonSet ensures that a replica of a pod is running on all nodes.** is correct because a DaemonSet manages Pods so that Kubernetes schedules a copy on every eligible node. As nodes are added, the DaemonSet controller automatically adds the required Pod; when nodes are removed, the corresponding Pod is garbage-collected. Administrators can limit placement to a subset of nodes through node selectors, affinity, tolerations, or related scheduling controls, but its core purpose is one Pod per applicable node. DaemonSets are therefore well suited to node-level operational workloads, including log collectors, monitoring agents, and network plugins. Details are available in the Kubernetes DaemonSet documentation.

## QUESTION NO: 3

You must install a basic Kubernetes cluster.

Which tool would you use in this situation?

**A.** kubeadm

**B.** kubectl apply

**C.** kubectl create

**D.** dashboard

## ANSWER: A

**Explanation:**

**kubeadm** is the appropriate tool because it is Kubernetes' purpose-built utility for creating, or bootstrapping, a cluster. Its `kubeadm init` workflow initializes the control plane, generates the required certificates and kubeconfig files, starts the core control-plane components, and produces a secure join command for additional nodes. A node can then be added to the cluster with `kubeadm join`. This provides a standardized, repeatable foundation for a basic Kubernetes installation while leaving operational choices—such as the container runtime, networking implementation, and high-availability design—under the administrator's control.

After initialization, an administrator installs a Container Network Interface plugin so that workloads can communicate across nodes. This is consistent with a basic kubeadm deployment: kubeadm establishes the Kubernetes cluster components, and the selected network plugin supplies pod networking. The official Kubernetes documentation describes kubeadm as a tool for creating Kubernetes clusters and provides the documented installation and initialization sequence. See the Creating a cluster with kubeadm guide and the kubeadm reference documentation.

## QUESTION NO: 4

Which two statements are correct about interfaces in a software-defined networking architecture? (Choose two.)

**A.** A northbound interface enables applications and orchestration systems to communicate with an SDN controller.

**B.** A southbound interface enables an SDN controller to communicate with network infrastructure.

**C.** A northbound interface is used by switches to install forwarding entries in an SDN controller.

**D.** A southbound interface removes the need for a control plane.

## ANSWER: A B

**Explanation:**

A **northbound interface** allows applications and orchestration systems to communicate intent or policy to an SDN controller. A **southbound interface** is used by the controller to communicate forwarding instructions and configuration to the underlying network infrastructure.

The northbound interface does not normally program switches directly, and the forwarding plane does not replace the controller's role in calculating and distributing network-wide policy.

## QUESTION NO: 5

Which two statements about containers are true? (Choose two.)

**A.** Containers contain executables, libraries, configuration files, and an operating system.

**B.** Containers package the entire runtime environment of an application, including its dependencies.

**C.** Containers can only run on a system with a Type 2 hypervisor.

**D.** Containers share the use of the underlying system's kernel.

**ANSWER: B D**

**Explanation:**

Containers package an application with the components needed for it to execute consistently, such as its runtime, required libraries, binaries, and configuration. This packaging creates a portable unit that can be deployed across compatible environments without requiring the application's dependencies to be separately installed and configured on each target system. It supports consistent application behavior throughout development, testing, and production workflows.

Containers also use operating-system-level virtualization. A container provides an isolated user-space environment for its processes, filesystem view, networking, and resource controls, while using the kernel of its host operating system. Sharing that underlying kernel is a key reason containers are typically smaller and start more quickly than virtual machines, which emulate hardware and run separate guest operating systems. The host kernel's namespaces and control groups help provide the isolation and resource management on which container platforms rely. Docker describes containers as isolated processes that package an application and its dependencies, and Juniper's cloud training materials identify containers as lightweight virtualization based on a shared operating-system kernel. See Docker: What is a container? and Juniper JNCIA-Cloud certification track.

## QUESTION NO: 6

Which two tools are used to deploy a Kubernetes environment for testing and development purposes? (Choose two.)

**A.** OpenStack

**B.** kind

**C.** oc

**D.** minikube

**ANSWER: B D**

**Explanation:**

**kind** and **minikube** are purpose-built tools for creating Kubernetes environments suited to local development, experimentation, and testing. kind, whose name means "Kubernetes in Docker," creates Kubernetes clusters with Docker containers functioning as cluster nodes. This makes it particularly useful for automated testing workflows and for quickly creating and deleting reproducible clusters, including multi-node topologies. The kind project explicitly describes its primary design goal as testing Kubernetes itself, while also supporting local development use cases.

**minikube** deploys a local Kubernetes cluster on a developer workstation using a supported driver, such as Docker or a virtual-machine hypervisor. It provides an approachable way to run Kubernetes locally, test manifests and services, explore cluster features, and develop applications without requiring shared or production infrastructure. Its documentation identifies local Kubernetes application development as a core use case.

Both tools therefore let users provision self-contained Kubernetes clusters with minimal infrastructure overhead, making them appropriate choices for non-production development and test environments. See the kind documentation and the minikube documentation for installation guidance, supported environments, and cluster-creation workflows.

## QUESTION NO: 7

Which type of virtualization provides containerization and uses a microservices architecture?

**A.** hardware-assisted virtualization

**B.** OS-level virtualization

**C.** full virtualization

**D.** paravirtualization

**ANSWER: B**

**Explanation:**

OS-level virtualization is the virtualization model that enables containerization. Rather than emulating hardware and booting a separate guest operating system for every workload, it isolates processes in separate user-space environments while all containers share the host operating system's kernel. Each container can package an application with its required libraries, dependencies, configuration, and filesystem view, allowing it to run consistently across development, test, and production environments.

This lightweight isolation makes OS-level virtualization especially well suited to microservices architectures. Microservices divide an application into small, independently deployable services; containers make those services fast to start, portable, resource-efficient, and easy to scale individually. Container platforms such as Docker use operating-system kernel isolation features, while orchestration systems such as Kubernetes manage deployment, scaling, connectivity, and lifecycle operations for collections of containers. This approach is central to cloud-native application design and is distinct from VM-centric approaches that require a complete guest OS per instance. Docker's overview of containers describes how containers share the host kernel while isolating application processes, and Kubernetes documents its role in managing containerized workloads: Docker overview and Kubernetes concepts overview.

**QUESTION NO: 8**

Regarding the third-party CNI in OpenShift, which statement is correct?

**A.** In OpenShift, you can remove and install a third-party CNI after the cluster has been deployed.

**B.** In OpenShift, you must specify the third-party CNI to be installed during the initial cluster deployment.

**C.** OpenShift does not support third-party CNIs.

**D.** In OpenShift, you can have multiple third-party CNIs installed simultaneously.

**ANSWER: B**

**Explanation:**

In OpenShift, you must specify the third-party CNI to be installed during the initial cluster deployment. The cluster network is a foundational, install-time component: it defines how pods receive addresses, communicate across nodes, and reach cluster services. Consequently, the installation configuration must establish the intended networking model before workloads and normal cluster operations begin. For an installation that uses a third-party CNI, the installer and the subsequent CNI deployment process must be planned together so that the cluster reaches a supported, consistent network state from the outset.

This requirement protects core control-plane and node connectivity. The Cluster Network Operator manages the configured cluster network provider, and OpenShift documentation treats changing the primary cluster network provider after installation as unsupported. In practical terms, administrators should decide whether the environment requires a third-party CNI's particular networking, security, or policy capabilities during cluster design, then include that decision in the installation procedure. This avoids an in-place replacement of the cluster's foundational pod-network implementation after workloads, routes, and network-dependent platform components are already active. See Red Hat's OpenShift cluster networking documentation and its installation documentation.