PassQueen
a.k.a dumpsqueen.com

Oracle Cloud Infrastructure 2026 Developer Professional

Oracle 1z0-1084-25

Version Demo

Total Demo Questions: 10

Total Premium Questions: 105

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

Which of the following step is NOT required for setting up the Container Engine for Kubernetes (OKE) cluster access using a local installation of kubectl?

A. Generate Auth token from the OCI console to access the OKE cluster using kubectl.

B. Install and configure the Oracle Cloud Infrastructure (OCI) CLI.

C. Set up the kubeconfig file.

D. Generate an API signing key pair (if you do not already have one) and upload the public key of the API signing key pair.

ANSWER: A

Explanation:

Generate Auth token from the OCI console to access the OKE cluster using kubectl. is not required for standard local `kubectl` access to an OKE cluster. OKE uses a kubeconfig file that invokes the OCI CLI authentication mechanism to obtain the authentication token needed by the Kubernetes API server. The kubeconfig is generated with the OCI CLI, typically using the `oci ce cluster create-kubeconfig` command. When the CLI is configured with a user profile and API signing key, it can authenticate requests to OCI and support this token-generation flow.

An OCI auth token has a different purpose: it is a credential commonly used for services that require a password-like token, such as authenticating to OCI Registry with Docker. It is not the credential that must be created for the normal OCI CLI-based OKE kubeconfig workflow. Therefore, local cluster administration requires a suitable `kubectl` installation, a configured OCI CLI environment, the permissions to access the cluster, and the generated kubeconfig, but not an auth token created in the Console. Oracle's OKE documentation describes generating kubeconfig with the OCI CLI and the supported authentication prerequisites. See Accessing Clusters Using kubectl and Creating a kubeconfig File for a Cluster.

QUESTION NO: 2

A worker application receives messages from an Oracle Cloud Infrastructure (OCI) Queue. Processing each message can take up to five minutes. If a worker fails before deleting a received message, the message must become available for another worker to process. Which Queue setting should be configured to support this requirement?

A. Message retention period

B. Delivery delay

C. Visibility timeout

D. Dead-letter queue retention period

ANSWER: C

Explanation:

The **visibility timeout** controls how long a message received from an OCI Queue remains unavailable to other consumers. A worker should receive the message with a visibility timeout long enough to complete normal processing. If the worker successfully finishes, it deletes the message. If it fails or does not delete the message before the timeout expires, the message becomes visible again and can be received by another worker.

Message retention controls how long Queue stores a message overall, rather than temporary exclusivity after receipt. A delivery delay postpones the initial availability of a newly enqueued message, and a dead-letter queue is used for messages that repeatedly fail processing.

QUESTION NO: 3

You are building a cloud native serverless travel application with multiple Oracle Functions in Java, Python, and Node.js. You need to build and deploy these functions to a single application named travel-app. Which command will help you complete this task successfully?

A. fn function deploy app travel-app--all

B. fn app deploy --app travel-app --all

C. fn app --app travel-app deploy --ext java pyljs

D. fn deploy --app travel-app --all

ANSWER: D

Explanation:

`fn deploy --app travel-app --all` is the appropriate Fn Project CLI command for building and deploying all functions found under the current project directory to the Oracle Functions application named `travel-app`. The `fn deploy` operation builds a function's container image when necessary, pushes that image to the configured registry, and deploys the function configuration to Oracle Functions. Supplying `--app travel-app` explicitly targets the named application rather than relying on the current context or a value in `func.yaml`. The `--all` flag makes the command operate on every deployable function in the project structure, which is useful when the travel solution contains separate Java, Python, and Node.js functions. Each function can retain its own runtime-specific source and `func.yaml` metadata while being deployed into the same logical Oracle Functions application. Before deployment, the developer should have configured the Fn CLI context, registry, compartment, and authentication as required for the OCI tenancy. Oracle's Functions documentation describes deploying functions with the Fn CLI and the application targeting model in Deploying Functions. The available CLI command syntax and deployment behavior are documented in the Fn Project documentation.

QUESTION NO: 4

In the shared responsibility model, who should perform patching, upgrading, and maintaining of the worker nodes in provisioned Oracle Container Engine for Kubernetes (OKE) clusters?

A. Oracle Support does it.

B. It is the responsibility of the customer.

C. It is an automated process.

ANSWER: B

Explanation:

It is the responsibility of the customer. In a provisioned OKE cluster, Oracle operates and maintains the managed Kubernetes control plane, including its availability and underlying control-plane infrastructure. The worker nodes, however, are part of the customer-managed data plane. Because these compute instances run the customer's workloads, the customer is responsible for their node operating system lifecycle, Kubernetes worker-node version upgrades, configuration, security patching, capacity, and ongoing maintenance. This responsibility also includes planning upgrades in a way that maintains application availability, such as creating replacement nodes or node pools, draining workloads from nodes, and validating workload compatibility with the intended Kubernetes version. OKE provides managed cluster capabilities and tools to support these activities, but those capabilities do not transfer ownership of worker-node lifecycle operations to Oracle for provisioned worker-node clusters. This division is consistent with OCI's shared-responsibility approach: Oracle secures and manages the cloud services and infrastructure it operates, while customers manage the resources, operating systems, configurations, and applications they deploy. See the Oracle Container Engine for Kubernetes overview and the OKE cluster upgrade documentation.

QUESTION NO: 5

Which two " Action Type " options are NOT available in an Oracle Cloud Infrastructure (OCI) Events rule definition? (Choose two.)

A. Email

B. Streaming

C. Slack

D. Functions

E. Notifications

ANSWER: A C

Explanation:

Email and Slack are not native action types that can be selected when defining an OCI Events rule. An Events rule evaluates incoming OCI events against its condition, then routes matching events to a supported target. The OCI Events service provides target actions for invoking an OCI Function, publishing to an OCI Notifications topic, or writing to an OCI Streaming stream. Therefore, notifications are delivered indirectly rather than by choosing an individual email address or a Slack workspace as the Events action itself.

For email delivery, an Events rule can publish to an OCI Notifications topic. The topic can have an email subscription, and OCI Notifications handles delivery to that subscribed email endpoint. This preserves the Events rule's event-routing model while allowing recipients to receive messages by email. Similarly, Slack is not an Events action type. A Slack integration must be implemented through an intermediary integration pattern, such as invoking a function or another service that formats and sends the message to Slack's webhook API. Oracle's Events documentation lists the supported action targets, while the Notifications documentation describes topic subscriptions and supported delivery protocols. See OCI Events Overview and OCI Notifications Overview.

QUESTION NO: 6

As a developer, you have been tasked with implementing a microservices-based application. Which THREE technologies are best suited to accomplish the task? (Choose three.)

A. Terraform

B. Big Data

C. Anomaly Detection

D. Service Mesh

E. Docker

F. Kubemetes

ANSWER: D E F

Explanation:

Service Mesh, Docker, and Kubemetes are the technologies most directly aligned with implementing and operating a microservices-based application. Docker provides container packaging: each independently deployable service can include its application code, runtime, libraries, and configuration dependencies in a consistent image. This portability supports repeatable local development, testing, CI/CD, and deployment across environments.

Kubemetes refers to Kubernetes, the container-orchestration platform commonly used to run microservices at scale. Kubernetes manages the desired state of containerized workloads and supplies core operational capabilities such as service discovery, internal load balancing, rolling updates, health-based recovery, and horizontal scaling. Oracle Kubernetes Engine applies these Kubernetes capabilities as a managed OCI service, reducing the operational burden of maintaining the control plane.

A service mesh complements Kubernetes by handling service-to-service communication through a dedicated infrastructure layer. It can provide traffic management, mutual TLS, policy enforcement, telemetry, tracing, and observability without requiring every service team to implement these cross-cutting concerns independently. Together, containers provide the unit of deployment, Kubernetes provides orchestration, and a service mesh provides consistent communication and security controls across the distributed application. See Oracle's Oracle Kubernetes Engine overview and OCI Service Mesh documentation.

QUESTION NO: 7

When developing microservices, each one can be developed in the language of choice. Which term describes this type of development? (Choose the best answer.)

A. Agile

B. DevOps

C. Distributed

D. Polyglot

ANSWER: D

Explanation:

Polyglot describes an architecture in which individual microservices may be implemented using different programming languages, frameworks, or technology stacks according to the needs of each service. Because a microservice is independently developed, deployed, and operated, a team can select a language that is especially well suited to that service's responsibilities—for example, a language and framework optimized for high-throughput APIs, data processing, or a particular integration. This is commonly called *polyglot programming*, and in microservice environments it is often discussed as polyglot microservices or polyglot architecture. The essential characteristic is technology diversity at the service level while services remain interoperable through well-defined network interfaces, such as HTTP-based APIs, messaging, or gRPC. Oracle's cloud-native guidance recognizes that microservices are independently deployable services that communicate through APIs, a design that enables teams to evolve services separately and choose appropriate implementation technologies. See Oracle Cloud Native—Microservices and Martin Fowler's overview of Polyglot Programming.

QUESTION NO: 8

A development team uses OCI DevOps Code Repositories. Every commit pushed to the `main` branch must automatically start a build that runs the repository build specification. Which TWO OCI DevOps components are required for this design? (Choose two.)

A. A build pipeline containing a managed build stage.

B. A build pipeline trigger filtered for commits to the main branch.

C. A deployment pipeline containing an OKE deployment stage.

D. A Manual Approval stage in a deployment pipeline.

E. An OCI Events rule that invokes the build worker directly.

ANSWER: A B

Explanation:

An OCI DevOps build pipeline is required to define the managed build stage that retrieves the source and runs the build specification. The build pipeline is responsible for executing the build, test, and artifact-producing steps.

A build pipeline trigger is also required to start the build automatically when a qualifying commit is pushed to the selected repository branch. The trigger can be configured to filter the source connection and branch so that commits to `main` initiate the pipeline.

A deployment pipeline deploys an already produced artifact and does not build source code. A Manual Approval stage pauses a deployment workflow and does not initiate a build. OCI Events is not required when the OCI DevOps build trigger provides the source-code event integration.

QUESTION NO: 9

Which TWO statements are true for serverless computing and serverless architectures? (Choose two.)

A. Serverless function execution is fully managed by third party.

B. Applications running on a FaaS (Functions as a Service) platform.

C. Long running tasks are perfectly suited for serverless.

D. Application DevOps team is responsible for scaling.

E. Serverless function state should never be stored externally.

ANSWER: A B

Explanation:

Serverless function execution is fully managed by third party is correct because serverless computing shifts responsibility for the underlying servers, operating system management, capacity provisioning, patching, and much of the operational scaling to the cloud provider. Developers focus on writing and deploying code, configuring triggers, and defining the required permissions rather than administering the infrastructure that executes the code. In Oracle Cloud Infrastructure, Functions is a fully managed serverless platform: OCI manages the infrastructure, while customers deploy functions and invoke them through events, HTTP calls, or other integrations.

Applications running on a FaaS (Functions as a Service) platform is also correct because Function as a Service is a central serverless architecture pattern. Application logic is packaged as small functions that execute in response to an invocation or event. This model supports event-driven designs, where functions can be connected to services such as API Gateway, Object Storage, Streaming, and Events. OCI Functions is built on the open-source Fn Project and provides the managed runtime environment for this FaaS approach. See the OCI Functions overview and Oracle's Oracle Cloud Infrastructure Functions page.

QUESTION NO: 10

Which TWO are part of the Cloud Native Computing Foundation (CNCF) container runtime? (Choose two.)

A. rkt-o

B. runc

C. getcd

D. containerd

ANSWER: B D

Explanation:

runc and **containerd** are CNCF graduated projects that provide complementary container-runtime functionality. runc is a lightweight command-line tool for spawning and running containers according to the Open Container Initiative Runtime Specification. It performs the low-level work of creating an isolated container process using operating-system primitives such as namespaces and cgroups. containerd is an industry-standard container runtime that manages the complete container lifecycle, including image transfer and storage, container execution, snapshotting, and supervision. In typical

implementations, containerd invokes runc as its default low-level runtime to create and run OCI containers. This layered architecture is central to the cloud-native container ecosystem: higher-level platforms can use containerd to manage workloads while runc supplies standards-based container execution. CNCF lists both projects as graduated, reflecting their maturity, broad adoption, and ongoing community governance. See the CNCF project pages for containerd and runc.