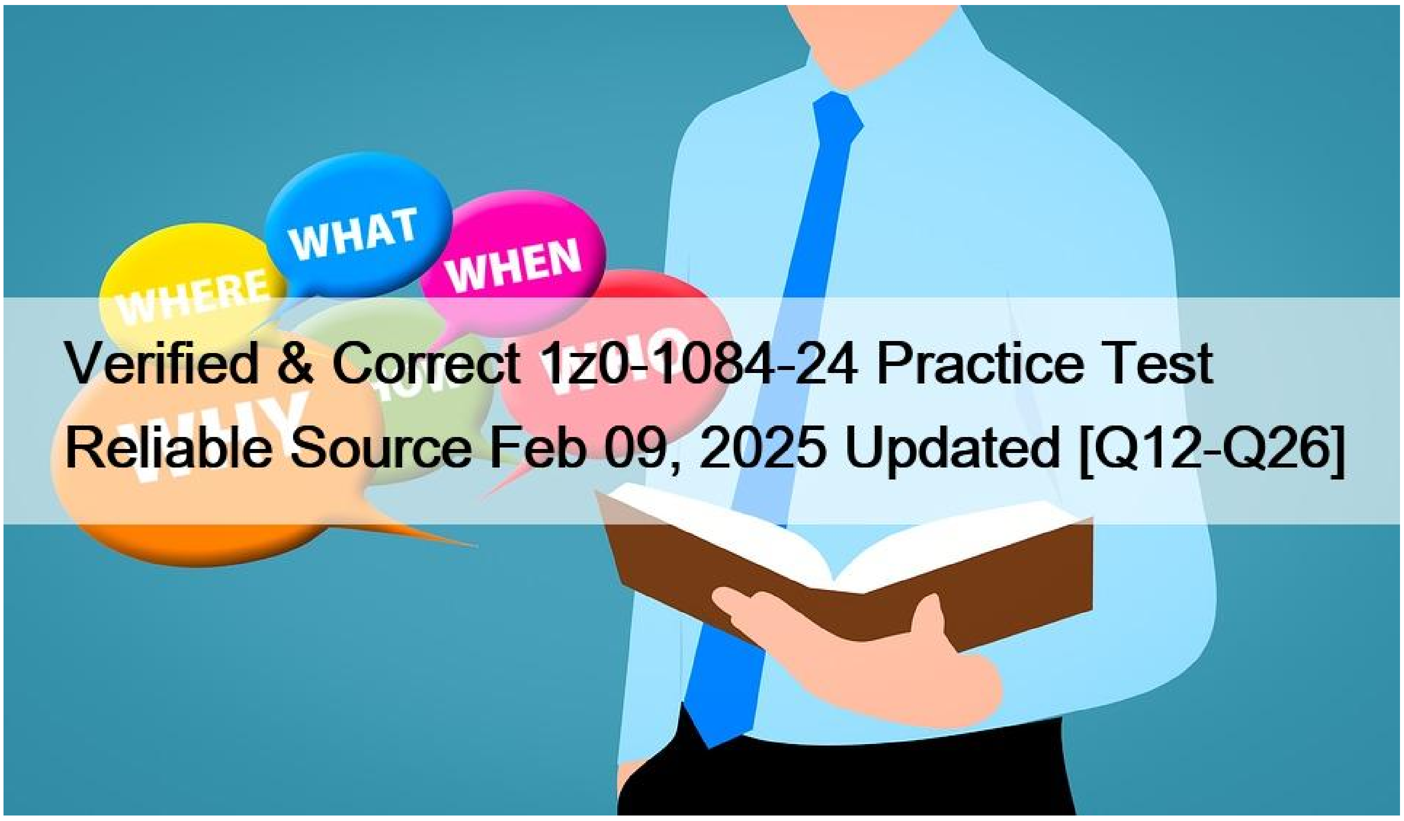
Verified & Correct 1z0-1084-24 Practice Test Reliable Source Feb 09, 2025 Updated [Q12-Q26


WHERE
WHAT
WHEN
WHO
WHY
HOW
Verified & Correct 1z0-1084-24 Practice Test
Reliable Source Feb 09, 2025 Updated [Q12-Q26]

Verified & Correct 1z0-1084-24 Practice Test Reliable Source Feb 09, 2025 Updated Free Oracle 1z0-1084-24 Exam Files Downloaded Instantly Q12. Which kubectl command syntax is valid for implementing a rolling update deployment strategy in Kubernetes? (Choose the best answer.)
* kubectl upgrade -c <container> &#8211;image=image:v2
* kubectl update <deployment-name> &#8211;image=image:v2
* kubectl rolling-update <deployment-name> &#8211;image=image:v2
* kubectl update -c <container> &#8211;iniage=image: v2

The correct syntax for implementing a rolling update deployment strategy in Kubernetes using the kubectl command is: kubectl rolling-update <deployment-name> &#8211;image=image:v2 This command initiates a rolling update of the specified deployment by updating the container image to image:v2. The rolling update strategy ensures that the new version of the application is gradually deployed while maintaining availability and minimizing downtime.

Q13. Which TWO statements accurately describe an Oracle Functions application? (Choose two.)
* A common context to store configuration variables that are available to all functions in the application.

A Docker image containing all the functions that share the same configuration.
* An application based on Oracle Functions, Oracle Cloud Infrastructure (OCI) Events, and OCI API Gateway services.
* A small block of code invoked in response to an OCI Events service.

A logical group of functions.

* A Docker image containing all the functions that share the same configuration.

The correct statements are: A common context to store configuration variables that are available to all functions in the application. A Docker image containing all the functions that share the same configuration. A logical group of functions. Explanation: An Oracle Functions application provides a common context for functions within the application. It allows you to store configuration variables that are accessible by all the functions in the application. Functions within the same application can share the same Docker image, which contains the common configuration and dependencies. An Oracle Functions application serves as a logical group that organizes related functions. Functions within the same application can be managed collectively, and they can interact and share resources within the application context.

Q14. 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?

* fn function deploy app travel-app&#8211;all
* fn app deploy &#8211;app travel-app &#8211;all
* fn app &#8211;app travel-app deploy &#8211;ext java pyljs
* fn deploy&#8211;app travel-app &#8211;all

The correct answer is: fn deploy &#8211;app travel-app &#8211;all Explanation: To build and deploy multiple Oracle Functions as part of a single application named &#8220;travel-app,&#8221; you can use the fn deploy command with the appropriate options. The command fn deploy &#8211;app travel-app &#8211;all is the correct syntax. Here&#8217;s what each part of the command does: fn deploy: This command is used to deploy functions and applications in Oracle Functions. &#8211;app travel-app: This option specifies the application name as &#8220;travel-app,&#8221; indicating that you want to deploy functions to this application. &#8211;all: This option indicates that you want to deploy all the functions within the application. By using fn deploy &#8211;app travel-app &#8211;all, you can build and deploy all the functions in your travel application across different programming languages (Java, Python, and Node.js) to the &#8220;travel-app&#8221; application in Oracle Functions.

Q15. (CHK_4>2) You have a scenario where a DevOps team wants to store secrets in Oracle Cloud Infrastructure (OCI) Vault so that it can inject the secrets into an app&#8217;s environment variables (for example, MYSQL_DB_PASSWD) at deployment time. Which is NOT valid about managing secrets in the OCI Vault service?

* New secret versions automatically expire in 90 days unless you configure an expiry rule.
* You can manually create new secrets as well as new secret versions using the OCI Console:
* A unique OCID is automatically generated for each secret and remains unchanged even when creating a new secret version.
* A secret reuse rule prevents the use of secret contents across different versions of a secret.

The correct answer is: &#8220;A unique OCID is automatically generated for each secret and remains unchanged even when creating a new secret version.&#8221; The statement that is NOT valid about managing secrets in the OCI Vault service is: &#8220;A unique OCID is automatically generated for each secret and remains unchanged even when creating a new secret version.&#8221; In OCI Vault, a secret is identified by its OCID (Oracle Cloud Identifier), which is a unique identifier for each resource in Oracle Cloud Infrastructure. However, when a new secret version is created for an existing secret, the OCID remains the same for the secret itself, but a new OCID is generated for the secret version. This allows you to track and manage different versions of a secret while maintaining a consistent OCID for the secret itself. The other statements mentioned are valid: You can manually create new secrets as well as new secret versions using the OCI Console. This means you have control over creating and managing secrets within the Vault service. A secret reuse rule prevents the use of secret contents across different versions of a secret. This ensures that each secret version maintains its own unique set of contents and avoids accidental reuse or sharing of secrets across versions. By default, new secret versions automatically expire in 90 days unless you configure an expiry rule. This helps enforce good security practices by automatically rotating secrets periodically, reducing the risk of unauthorized access in case of compromise. Therefore, the statement that is NOT valid is the one regarding the uniqueness and consistency of the OCID when creating new secret versions.

Q16. You are developing a real-time monitoring application for a fleet of vehicles, which will be deployed on Oracle Cloud Infrastructure (OCI). You need to choose between using OCI Queue or OCI Streaming to handle the real-time data feeds from the vehicles. Based on the scenario described, which is the most appropriate choice for handling real-time data feeds?

* OCI Streaming, because it is designed for high-volume, continuous ingestion and processing of data, making it the best choice for a fleet of vehicles
* OCI Streaming, because it offers exactly-once message delivery, which is necessary for real-time applications
* OCI Queue, because it is optimized for low-latency messaging and ideal for real-time applications
* OCI Queue, because it provides at-least-once message delivery, which is critical for real-time monitoring applications

OCI Streaming is a fully managed, scalable, and durable messaging solution for ingesting continuous, high- volume streams of data that you can consume and process in real-time1. Streaming is suitable for any use case in which data is produced and processed continually and sequentially in a publish-subscribe messaging model1. Streaming can handle millions of messages per second with low latency2. Therefore, OCI Streaming is the most appropriate choice for handling real-time data feeds from a fleet of vehicles. Verified References: Overview of Streaming, Container Engine for Kubernetes

Q17. You plan to implement logging in your services that will run in Oracle Cloud Infrastructure (OCI) Container Engine for Kubernetes (OKE). Which statement describes the appropriate logging approach?
* All services log to an external logging system.
* All serviceAAs log to a shared log file.
* All services log to standard output only.
* Each service logs to its own log file.

Q18. Which is the smalled unit of Kubernetes architecture?
* Node
* Container
* Cluster
* Pod

The smallest unit of Kubernetes architecture is a Pod. A Pod is a logical grouping of one or more containers that are deployed together on the same host and share the same network namespace, storage, and other resources. It represents the smallest deployable unit in Kubernetes and is used to encapsulate and manage one or more closely related containers. Containers within a Pod are scheduled and deployed together, allowing them to communicate and share resources efficiently.

Q19. You have two microservices, A and B, running in production. Service A relies on APIs from service B. You want to test changes to service A without deploying all of its dependencies, which include service B. Which approach should you take to test service A?
* Test using API mocks.
* Test the APIs in private environments.
* Test against production APIs.
* There is no need to explicitly test APIs.

API mocking is a technique that simulates the behavior of real APIs without requiring the actual implementation or deployment of the dependent services1. API mocking allows you to test changes to service A without deploying all of its dependencies, such as service B, by creating mock responses for the APIs that service A relies on1. API mocking has several benefits, such as1:

* Faster testing: You can test your service A without waiting for service B to be ready or available, which reduces the testing time and feedback loop.

* Isolated testing: You can test your service A in isolation from service B, which eliminates the possibility of external factors affecting the test results or causing errors.

* Controlled testing: You can test your service A with different scenarios and edge cases by creating mock responses that mimic various situations, such as success, failure, timeout, etc.

Q20. What is the difference between blue/green and canary deployment strategies? (Choose the best answer.)
* In blue/green, current applications are slowly replaced with new ones. In canary, the application Is deployed Incrementally to a

select group of people.
* In blue/green, both old and new applications are in production at the same time. In canary, the application Is deployed incrementally to a select group of people.
* In blue/green, current applications are slowly replaced with new ones. In canary, both old and new applications are in production at the same time.
* In blue/green, the application Is deployed In minor Increments to a select group of people. In canary, both old and new applications are simultaneously in production.
The correct answer is: In blue/green deployment, both old and new applications are in production at the same time. In canary deployment, the application is deployed incrementally to a select group of people. In a blue

/green deployment strategy, two identical environments, referred to as blue and green, are set up. The current production environment (blue) continues to serve live traffic while a new version of the application is deployed in the green environment. Once the new version is tested and deemed stable, traffic is routed from the blue environment to the green environment, making it the new production environment. This approach allows for a seamless switch between the old and new versions of the application. On the other hand, in a canary deployment strategy, the new version of the application is deployed incrementally to a small subset of users or a specific group. This allows for testing the new version in a real production environment while minimizing the impact of any potential issues. If the new version performs well and meets the desired criteria, it can be gradually rolled out to a larger audience or the entire user base. In summary, the main difference between blue/green and canary deployment strategies lies in how the deployment is managed. Blue/green involves simultaneous production of both old and new applications, while canary deployment focuses on incremental deployment to a select group of users.

Q21. You are creating an API deployment in Oracle Cloud Infrastructure (OCI) API Gateway and you want to configure request policies to control access. Which is NOT available in OCI API Gateway?
* Controlling access to the backend OCI resources.
* Limiting the number of requests sent to the backend services.
* Enabling Cross-Origin Resource Sharing (CORS) support.
* Providing authentication and authorization.
The correct answer is: Controlling access to the backend OCI resources. OCI API Gateway does not provide direct control over access to backend OCI resources. It primarily focuses on managing and securing access to APIs exposed through the gateway. The gateway acts as a front-end for APIs and provides features such as authentication, authorization, rate limiting, and CORS support. While you can configure authentication and authorization policies, limit the number of requests, and enable CORS support in OCI API Gateway, it does not directly control access to backend OCI resources. Access to backend resources is typically managed through other means, such as IAM policies, network security rules, or resource-specific access controls.

Q22. What can you use to dynamically make Kubernetes resources discoverable to public DNS servers? (Choose the best answer.)
* kubeDNS
* DynDNS
* CoreDNS
* ExternalDNS
To dynamically make Kubernetes resources discoverable to public DNS servers, you can use ExternalDNS.

ExternalDNS is a Kubernetes add-on that automates the management of DNS records for your Kubernetes services and ingresses. It can be configured to monitor the changes in your Kubernetes resources and automatically update DNS records in a supported DNS provider. By integrating ExternalDNS with your Kubernetes cluster, you can ensure that the DNS records for your services and ingresses are automatically created, updated, or deleted based on changes in your Kubernetes resources. This allows your Kubernetes resources to be discoverable by external systems through public DNS servers.

Q23. Which term describes a group formed by a master machine and a worker machine in a Kubernetes architecture?
* Cluster
* Node

* Deployment
* Container
* Pod

The term that describes a group formed by a master machine and a worker machine in a Kubernetes architecture is &#8220;Cluster&#8221;. A cluster in Kubernetes consists of one or more master machines and multiple worker machines (also known as nodes). The master machine manages the overall control plane and orchestrates the deployment and management of containers on the worker nodes. The worker nodes are responsible for running the containers and executing the workloads. The cluster is the fundamental unit of organization and management in Kubernetes, providing the infrastructure and resources to run and manage containerized applications. It ensures high availability, scalability, and fault tolerance for the applications deployed within it.

Q24. Your organization is developing serverless applications with Oracle Functions. Many functions will need to store state data in a database, which will require using appropriate credentials. However, your corporate security standards mandate encryption of secret information, such as database passwords. How would you address this security requirement?

* Use OCI Console to enter the password in the function configuration section in the provided input field.
* Leverage application-level configuration variables to store passwords because they are automatically encrypted by Oracle Functions.
* Use the OCI Vault service to auto-encrypt the password and then set an application-level configuration variable to reference the auto-decrypted password inside your function container.
* Encrypt the password using the OCI Vault service and then decrypt this password in your function code with the generated key.

The best way to store and use secret information, such as database passwords, in Oracle Functions is to use the OCI Vault service. The OCI Vault service provides encryption and decryption capabilities for sensitive data. You can use the OCI Vault service to encrypt the password and store it as an application-level configuration variable. Then, you can use the generated key to decrypt the password in your function code when you need to access the database. Verified References: Oracle Functions: Using Key Management To Encrypt And Decrypt Configuration Variables

Q25. Your company has recently deployed a new web application that uses Oracle Functions. Your manager instructs you to implement monitoring metrics to manage your systems more effectively. You know that Oracle Functions automatically monitors functions on your behalf and reports metrics via Oracle Cloud Infrastructure (OCI) Monitoring. Which TWO metrics are collected and made available by this feature?

(Choose two.)

* Amount of CPU used by a function
* Length of time a function runs
* Number of times a function Is removed
* Amount of RAM used by a function
* Number of times a function is invoked

The correct answers are: Amount of RAM used by a function: Oracle Functions collects and reports the amount of memory (RAM) used by a function during its execution. This metric helps in monitoring and optimizing the resource consumption of functions. Length of time a function runs: Oracle Functions captures and provides the duration of function executions. This metric allows you to track the performance and responsiveness of your functions and identify any potential bottlenecks or delays. These metrics provide valuable insights into the resource utilization and performance of your functions, enabling you to monitor and optimize their behavior in the Oracle Cloud Infrastructure (OCI) environment.

Q26. Which TWO are characteristics of microservices? (Choose two.)

* Microservices communicate over lightweight APIs.
* Microservices can be implemented in limited number of programming languages.
* All microservices share a data store.
* Microservices are hard to test in isolation.
* Microservices can be independently deployed.

The two characteristics of microservices are: Microservices can be independently deployed: One of the key principles of microservices architecture is the ability to independently deploy each microservice. This means that changes or updates to one microservice can be made and deployed without affecting other microservices.

It allows for faster and more frequent deployments, enabling agile development and scalability. Microservices communicate over lightweight APIs: Microservices communicate with each other through lightweight APIs (Application Programming Interfaces). This enables loose coupling between microservices, as they can interact with each other using standard protocols like HTTP/REST or messaging systems like RabbitMQ or Kafka. Lightweight APIs facilitate flexibility and interoperability between microservices, making it easier to develop and maintain complex systems. The remaining statement, &#8220;All microservices share a data store,&#8221; is not a characteristic of microservices. Microservices are designed to be autonomous and have their own data storage or database. Each microservice has its own data store, which promotes the principle of bounded contexts and avoids tight coupling between services. This allows for better scalability and independence of data management within each microservice.