1Z0-1084-21 Q&As

Oracle Cloud Infrastructure Developer 2021 Associate

QUESTION 1

Which Oracle Cloud Infrastructure (OCI) load balancer shape is used by default in OCI container Engineer for Kubernetes?

A. 400 Mbps

B. 8000 Mbps

C. There is no default. The shape has to be specified.

D. 100 Mbps

Correct Answer: D

Specifying Alternative Load Balancer Shapes The shape of an Oracle Cloud Infrastructure load balancer specifies its maximum total bandwidth (that is, ingress plus egress). By default, load balancers are created with a shape of 100Mbps. Other shapes are available, including 400Mbps and 8000Mbps. https://docs.cloud.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingloadbalancer.htm

QUESTION 2

Which pattern can help you minimize the probability of cascading failures in your system during partial loss of connectivity or a complete service failure?

A. Retry pattern

B. Anti-corruption layer pattern

C. Circuit breaker pattern

D. Compensating transaction pattern

Correct Answer: C

A cascading failure is a failure that grows over time as a result of positive feedback. It can occur when a portion of an overall system fails, increasing the probability that other portions of the system fail. the circuit breaker pattern prevents the service from performing an operation that is likely to fail. For example, a client service can use a circuit breaker to prevent further remote calls over the network when a downstream service is not functioning properly. This can also prevent the network from becoming congested by a sudden spike in failed retries by one service to another, and it can also prevent cascading failures. Self-healing circuit breakers check the downstream service at regular intervals and reset the circuit breaker when the downstream service starts functioning properly. https://blogs.oracle.com/developers/gettingstarted-with-microservices-part-three

QUESTION 3

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 includes service B.

Which approach should you take to test service A?

A. Test against production APIs.

B. Test using API mocks.

C. There is no need to explicitly test APIs.

D. Test the APIs in private environments.

Correct Answer: B

Testing using API mocks Developers are frequently tasked with writing code that integrates with other system components via APIs. Unfortunately, it might not always be desirable or even possible to actually access those systems during development. There could be security, performance or maintenance issues that make them unavailable ? or they might simply not have been developed yet. This is where mocking comes in: instead of developing code with actual external dependencies in place, a mock of those dependencies is created and used instead. Depending on your development needs this mock is made "intelligent" enough to allow you to make the calls you need and get similar results back as you would from the actual component, thus enabling development to move forward without being hindered by eventual unavailability of external systems you depend on

QUESTION 4

Which concept is NOT related to Oracle Cloud Infrastructure Resource Manager?

A. Job

B. Stack

C. Queue

D. Plan

Correct Answer: C

https://docs.cloud.oracle.com/en-us/iaas/Content/ResourceManager/Concepts/resourcemanager.htm Following are brief descriptions of key concepts and the main components of Resource Manager. CONFIGURATION Information to codify your infrastructure. A Terraform configuration can be either a solution or a file that you write and upload. JOB Instructions to perform the actions defined in your configuration. Only one job at a time can run on a given stack; further, you can have only one set of Oracle Cloud Infrastructure resources on a given stack. To provision a different set of resources, you must create a separate stack and use a different configuration. Resource Manager provides the following job types: Plan: Parses your Terraform configuration and creates an execution plan for the associated stack. The execution plan lists the sequence of specific actions planned to provision your Oracle Cloud Infrastructure resources. The execution plan is handed off to the apply job, which then executes the instructions. Apply. Applies the execution plan to the associated stack to create (or modify) your Oracle Cloud Infrastructure resources. Depending on the number and type of resources specified, a given apply job can take some time. You can check status while the job runs. Destroy. Releases resources associated with a stack. Released resources are not deleted. For example, terminates a Compute instance controlled by a stack. The stack\\'s job history and state remain after running a destroy job. You can monitor the status and review the results of a destroy job by inspecting the stack\\'s log files. Import State. Sets the provided Terraform state file as the current state of the stack. Use this job to migrate local Terraform environments to Resource Manager. STACK The collection of Oracle Cloud Infrastructure resources corresponding to a given Terraform configuration. Each stack resides in the compartment you specify, in a single region; however, resources on a given stack can be deployed across multiple regions. An OCID is assigned to each stack.

QUESTION 5

As a cloud-native developer, you have written a web service for your company. You have used Oracle Cloud Infrastructure (OCI) API Gateway service to expose the HTTP backend. However, your security team has suggested that your web service should handle Distributed Denial-of-Service (DDoS) attack. You are time-constrained and you need to make sure that this is implemented as soon as possible. What should you do in this scenario?

A. Use OCI virtual cloud network (VCN) segregation to control DDoS.

B. Use a third party service integration to implement a DDoS attack mitigation,

C. Use OCI API Gateway service and configure rate limiting.

D. Re-write your web service and implement rate limiting.

Correct Answer: C

Having created an API gateway and deployed one or more APIs on it, you\\'ll typically want to limit the rate at which front-end clients can make requests to back-end services. For example, to:

- maintain high availability and fair use of resources by protecting back ends from being overwhelmed by too many requests

-prevent denial-of-service attacks

-constrain costs of resource consumption

- restrict usage of APIs by your customers\\' users in order to monetize APIs You apply a rate limit globally to all routes in an API deployment specification. If a request is denied because the rate limit has been exceeded, the response header specifies when the request can be retried. You can add a rate-limiting request policy to an API deployment specification by: using the Console editing a JSON file