Free Questions for DCA by actualtestdumps

Shared by Bush on 24-05-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

The Kubernetes yaml shown below describes a networkPolicy.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: dca
  namespace: default
spec:
  podSelector:
    matchLabels:
      tier: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tier: api
...
```

Will the networkPolicy BLOCK this trafftc?

Solution. a request issued from a pod bearing only the tier: frontend label, to a pod bearing the tier: backend label

Options:

A- Yes

B- No

Answer:

A

Explanation:

The provided Kubernetes NetworkPolicy YAML configuration indicates that the policy applies to pods with the label tier: backend in the default namespace1. The ingress rule allows traffic from pods with the label tier: api1. Therefore, a request issued from a pod bearing only the tier: frontend label to a pod bearing the tier: backend label will be blocked by this networkPolicy1. This is because the networkPolicy does not have a rule allowing ingress from pods with the tier: frontend label1. For more information on Kubernetes NetworkPolicies, you can refer to the Kubernetes Documentation on Network Policies.

Question 2

Question Type: MultipleChoice

The Kubernetes yaml shown below describes a networkPolicy.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: dca
  namespace: default
spec:
  podSelector:
    matchLabels:
      tier: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tier: api
...
```

Will the networkPolicy BLOCK this trafftc?

Solution. a request issued from a pod bearing the tier: api label, to a pod bearing the tier: backend label

Options:

A- Yes

B- No

Answer:

B

Explanation:

The provided Kubernetes NetworkPolicy YAML configuration indicates that the policy applies to pods with the label tier: backend in the default namespace1. The ingress rule allows traffic from pods with the label tier: api1. Therefore, a request issued from a pod bearing the tier: api label to a pod bearing the tier: backend label will not be blocked by this networkPolicy1. This is because the networkPolicy explicitly allows ingress from pods with the tier: api label1. For more information on Kubernetes Network

Question 3

Question Type: MultipleChoice

Will this configuration achieve fault tolerance for managers in a swarm?

Solution: one manager node for two worker nodes

Options:

A- Yes

B- No

Answer:

B

Explanation:

Docker Swarm requires more than one manager node to achieve fault tolerance12. A single manager node is not fault tolerant because if it goes down, the entire cluster goes down3. For a swarm to be fault-tolerant, it needs to have an odd number of manager nodes2. For example, a three-manager swarm tolerates a maximum loss of one manager2. Therefore, a configuration with one manager node for two worker nodes will not achieve fault tolerance for managers in a swarm12.

Question 4

Question Type: MultipleChoice

You want to create a container that is reachable from its host's network.

Does this action accomplish this?

Solution: Use network attach to access the container on the bridge network.

Options:

A- Yes

B- No

Answer:

A

Explanation:

Docker's bridge network allows containers connected to the same bridge network to communicate, while providing isolation from containers that aren't connected to that bridge network1. By using the network attach command, you can connect a container to the bridge network, making it reachable from the host's network12. However, it's important to note that containers on the default bridge network can only access each other by IP addresses, unless you use the --link option, which is considered legacy2. For better isolation and automatic DNS resolution between containers, it's recommended to use user-defined bridge networks12.

Question 5

Question Type: MultipleChoice

In Docker Trusted Registry, is this how a user can prevent an image, such as 'nginx:latest', from being overwritten by another user with push access to the repository?

Solution: Remove push access from all other users.

Options:

A- Yes

B- No

Answer:

B

Explanation:

While removing push access from all other users can prevent an image from being overwritten, it's not the only way and might not be the most efficient or practical solution, especially in a collaborative environment. Docker Trusted Registry (DTR) provides a feature called 'Immutable Tags' which can be used to prevent an image, such as 'nginx:latest', from being overwritten. Once a tag is marked as immutable, DTR will prevent any user from pushing the same tag to the repository, thus preserving the image. This allows for better version control and prevents accidental overwrites. Therefore, the solution to prevent an image from being overwritten is not just to remove push access from all other users, but to use the features provided by DTR like 'Immutable Tags'.

Question 6

Question Type: MultipleChoice

Can this set of commands identify the published port(s) for a container?

Solution: `docker network inspect', `docker port'

Options:

A- Yes

B- No

Answer:

A

Explanation:

Yes, the docker port command can be used to identify the published ports for a running container. It shows the mapping between the host ports and the container's exposed ports. The docker network inspect command can also provide information about the network settings of the container, including port mappings. However, it's important to note that docker network inspect requires the network's name or ID as an argument, not the container's. Therefore, to get the network details of a specific container, you would first need to identify the network the container is connected to. These commands, when used appropriately, can help you identify the published ports for a container.

Question 7

Question Type: MultipleChoice

An application image runs in multiple environments, with each environment using different certificates and ports. Is this a way to provision configuration to containers at runtime?

Solution: Create a Dockerfile for each environment, specifying ports and ENV variables for certificates.

Options:

A- Yes

B- No

Answer:

B

Explanation:

While creating a Dockerfile for each environment is a possible solution, it is not the most efficient or scalable way to provision configuration to containers at runtime. Docker provides several mechanisms to inject configuration into containers at runtime, such as environment variables, command line arguments, Docker secrets for sensitive data, or even configuration files mounted as volumes. These methods allow the same Docker image to be used across multiple environments, promoting immutability and consistency across your deployments. Creating a separate Dockerfile for each environment would mean maintaining multiple versions of the Dockerfile, which could lead to inconsistencies and is generally not a recommended practice.

Question 8

Question Type: **MultipleChoice**

Is this a way to configure the Docker engine to use a registry without a trusted TLS certificate?

Solution: List insecure registries in the 'daemon.json configuration file under the \insecure-registries' key.

Options:

A- Yes

B- No

Answer:

A

Explanation:

Docker allows the use of insecure registries through a specific configuration in the Docker daemon. By listing the insecure registries in the 'daemon.json' configuration file under the 'insecure-registries' key, Docker can interact with these registries even without a trusted TLS certificate1. This is particularly useful when setting up a private Docker registry1. However, it's important to note that this configuration bypasses the security provided by TLS, and should be used with caution1.

Question 9

Question Type: MultipleChoice

You created a new service named 'http' and discover it is not registering as healthy. Will this command enable you to view the list of historical tasks for this service?

Solution: 'docker ps http'

Options:

A- Yes

B- No

Answer:

B

Explanation:

The command 'docker ps http' is not the correct command to view the list of historical tasks for a service in Docker1. The 'docker ps' command is used to list containers1. If you want to view the list of historical tasks for a service, you should use the 'docker service ps' command2. This command lists the tasks that are running as part of the specified services and also shows the task history2. Therefore, to view the list of historical tasks for the 'http' service, you should use the command 'docker service ps http'2.

Question 10

Question Type: MultipleChoice

A company's security policy specifies that development and production containers must run on separate nodes in a given Swarm cluster.

Can this be used to schedule containers to meet the security policy requirements?

Solution: node affinities

Options:

A- Yes

B- No

Answer:

A

Explanation:

They provide granular control over where pods (or in this case, containers) are scheduled, based on the labels of the nodes1. In the context of Docker Swarm, this means that you could use node affinities to ensure that development and production containers are scheduled on separate nodes, thus meeting the company's security policy requirements12345.

Question 11

Question Type: MultipleChoice

In Kubernetes, to mount external storage to a filesystem path in a container within a pod, you would use a volume in the pod specification. This volume is populated with a persistentVolumeClaim that is bound to an existing persistentVolume. The persistentVolume is defined and managed by the storageClass which provides dynamic or static provisioning of the volume and determines what type of storage will be provided1. Reference:

* Dynamic Volume Provisioning | Kubernetes

Is this a supported user authentication method for Universal Control Plane?

Solution: Docker ID

Options:

A- Yes

B- No

Answer:

B

Explanation:

Docker Universal Control Plane (UCP) has its own built-in authentication mechanism and integrates with LDAP services1. It also has role-based access control (RBAC), so that you can control who can access and make changes to your cluster and applications1. However, there is no mention of Docker ID being a supported user authentication method for UCP in the resources provided1234.