DCA: Docker Certified Associate. Guía de estudio.

Sobre el Autor

Mi nombre es Alberto González Mesas y he estado dedicado a la administración de servidores Unix/Linux desde hace una década. Al igual que la mayoría de los administradores de sistemas con cierto bagage, decidí pasar al "mundo" devops hace unos años. Lo interesante de la metodología DevOps es la cantidad de nuevas herramientas que se han puesto a disposición de los técnicos para una administración de servidores y servicios/apps ágil y eficiente.

Actualmente estoy certificado en varias tecnologías y cloud, pero después de trabajar durante un tiempo con Docker, decidí que era el momento de obtener otra más, la "Docker Certified Associate". De aquí surgieron los dos libros que actualmente tengo publicados en Amazon; "Docker para todos 2020: docker, docker-compose y docker swarm" y "Guía de estudio para la DCA (Docker Certified Associate)".

Como gran parte de los "técnologos", soy autodidacta y por ello me decidí a escribir ambos libros; ¿que mejor que prepararme la certificación mientras escribo todo aquello que voy estudiando, investigando y probando? y... ¿y si consigo autofinanciarme la certificación?. Con ambos objetivos en mente, me lancé a escribir.

Espero que el contenidos sea de ayuda y si aún no lo habéis hecho, os recomiendo leer mi primer libro "Docker para todos 2020: docker, docker-compose y docker swarm" con el que os prepararéis de una forma teorico-práctica. Si no es mi libro, podéis comprar cualquer otro que os garantice cierto contenido de calidad.

El Contenido de la guía

¿Que váis a encontrar en este libro?, pues principalmente 340 preguntas, en ocasiones bastante desarrolladas, con el objetivo de no solo saber si la respuesta es la A), B) o C), si no porqué es esa la respuesta.

He dividido el libro en dos partes; por un lado preguntas de apoyo teorico-prácticas (340) que permiten entender mejor la respuesta a la pregunta. Por otro lado he querido añadir un test de exámen completo (150) para que os pongáis a prueba antes de pagar una certificación que si no se aprueba, duele. Además, cuando pagaís la certificación, os dan otro test mas de prueba así que con un libro teórico-práctico, está guía, el test de prueba y algunos meses trabajando con Docker, será mas que suficiente para aprobar el exámen.

El contenido de esta gúia ha sido desarrollado en general a partir de muchas preguntas encontradas por Internet que yo mismo he ido recopilando conforme estudiaba, además de añadir tanto preguntas como respuestas propias acorde a la última guía de estudios que presentó Docker en 2020.

Distribución de contenido en el exámen

Según la guía de estudio oficial, el contenido que se abarca en el exámen está dividido tal que así:

- Orchestration (25%)
- Image Creation, Management, and Registry (20%)

- Installation and Configuration (15%)
- Networking (15%)
- Security (15%)
- Storage and Volumes (10%)

Sin más, comencemos con la primera de las partes, preguntas de apoyo para aprobar la certificación. Lo haremos respetando la división anterior.

Orchestration (25%)

En la última guía de estudios oficial, se han añadido preguntas sobre el orquestador Kubernetes por lo que voy a dividir en dos partes (una por orquestador) el contenido (25%) sobre orquestación de contenedores.

Swarm

Complete the setup of a swarm mode cluster, with managers and worker nodes

1. Which command do you use to create a new swarm?

```
$ docker swarm init --advertise-addr <MANAGER-IP>
```

2. What is this flag --advertise-addr for?

```
This flag configures the IP address for the manager node and The other
nodes in the swarm must be able to access the manager at the IP address.
```

3. How do you know the current status of the swarm?

```
$ docker info
// you can find the info under the swarm section
```

4. Which command do you use to find the information about the nodes in the swarm?

```
$ docker node ls
```

5. How to add another manager to the swarm?

```
$ docker swarm join-token manager
// copy the token and ip:port addr and
$ docker swarm join --token <token>
```

6. How to add another worker node to the swarm?

```
$ docker swarm join-token worker
// copy the token and ip:port addr and
$ docker swarm join --token <token>
```

Describe and demonstrate how to extend the instructions to run individual containers into running services under swarm

7. How to run a simple container?

```
$ docker run <image>
```

8. How to deploy a service in the docker swarm?

```
// for the nginx 1.19.3 version image
docker create service --replicas 3 --name webserver nginx:1.19.3
```

9. How to list the services in the Docker swarm?

```
$ docker service ls
```

10. How to list the tasks of the service in the Docker swarm?

```
$ docker service ps <service name>
```

11. How to inspect the service on the swarm?

```
$ docker service inspect <service name>
```

12. How to inspect the service on the swarm printing information in an easily readable format?

```
$ docker service inspect <service> --pretty
```

13. How to find out which nodes are running the service?

```
$ docker service ps <service>
```

14. How to find out more details of the container running these tasks of the service?

```
// run the following command on the particular node
$ docker ps
```

15. How to get the logs from a service?

```
$ docker service logs <service name>
```

16. How can you tear down a service?

```
$ docker service rm demo
```

Describe the importance of quorum in a swarm cluster.

17. Which algorithm does the docker engine use when it is in swarm mode to manage the global cluster state?

```
Raft Consensus Algorithm
```

18. What is a quorum and why it is important?

```
Quorun ensure that the cluster state stays consistent in the presence of
failures by requiring a majority of nodes to agree on values.
Raft tolerates up to (N-1)/2 failures and requires a majority or quorum of
(N/2)+1 members to agree on values proposed to the cluster.
without quorun swarm wont be able to serve the requests to schedule
additional tasks.
```

Describe the difference between running a container and running a service.

19. What is the difference between running a container and runnning a service?

```
When you run a container you are running an isolated process. When you run
a service in a Swarm, you are creating a kind of slot that Swarm scheduler
will fill by spawning a container, so the container is the instantiation of
the task. If the task fails, the orchestrator removes the task and its
container and then creates a new task to replace it.
```

Interpret the output of “docker inspect” commands

20. How can you see details about Docker resources?

```
With the inspect command. The inspect command let us to see details about
resources, ie services, volumes or networks. We can use --pretty to display
the details in an easily readable format.
```

Convert an application deployment into a stack file using a YAML compose file with "docker stack deploy"

21. If you are running co-related services in the docker swarm, what do you call this?

```
stack
```

22. What is Docker stack?

```
A stack is a group of interrelated services that share dependencies, and
can be orchestrated and scaled together.
```

23. From service to stack

```
From:
$ docker service create --name webserver --replicas 3 -p 8080:80 nginx
$ docker service create --name redis redis
to:

version: 3.8
services:
  webserver:
    image: nginx:1.19.3
    ports:
      - 8080:80
    deploy:
      replicas: 3
  redis:
    image: redis
```

Manipulate a running stack of services

24. Explain the several commands associated with Docker stack?

```
// deploy the new stack or update
$ docker stack deploy -c <compose file>
```

```
// list services in the stack
$ docker stack services
// list the tasks in the stack
$ docker stack ps
// remove the stack
$ docker stack rm
//List stack
$ docker stack ls
```

Describe and demonstrate orchestration activities and how increase the number of replicas

25. What is the autolock feature in the Docker swarm?

```
When Docker restarts, both the TLS key used to encrypt communication among
swarm nodes, and the key used to encrypt and decrypt Raft logs on disk, are
loaded into each manager node’s memory.
Docker 1.13 introduces the ability to protect the mutual TLS encryption key
and the key used to encrypt and decrypt Raft logs at rest, by allowing you
to take ownership of these keys and to require manual unlocking of your
managers. This feature is called autolock.
```

26. How to lock the swarm?

```
// This command produces unlock key. You need to place that in safe place
$ docker swarm init --autolock
```

27. How to unlock the swarm?

```
$ docker swarm unlock
```

28. Are we able to enable autolock feature only when we create a swarm for the first time?

```
No. You can lock the existing swarm as well
```

29. How to enable or disable autolock on the existing swarm?

```
//enable autolock
$ docker swarm update --autolock=true
//disable autolock
$ docker swarm update --autolock=false
```

30. How to view the current unlock key for the running swarm?

```
$ docker swarm unlock-key
```

31. How to rotate the unlock key?

```
$ docker swarm unlock-key --rotate
```

32. If the key was rotated after one of the manager nodes became unavailable and if you don't have access to the previous key you may need to force the manager to leave the swarm and join it back as a new manager. Is this statement correct?

```
Yes
```

33. How to filter the services in the stack?

```
// with the help of --filter flag
docker stack service nginx-web --filter name=web
```

34. How to format the output of the docker stack services command?

```
$ docker stack services --format "{{.ID}}: {{.Mode}} {{.Replicas}}"
```

35. How to increase the number of replicas?

```
// With the scale command
$ docker service scale SERVICE-1=REPLICAS [SERVICE-2=REPLICAS]
// you can also scale with the update command
$ docker service update --replicas=50 frontend
```

36. How to revert the changes for the service configuration?

```
$ docker service rollback my-service
```

Add networks, publish ports

37. What are the existing network drivers?

```
* bridge: The default network driver. Used when your applications run in
standalone containers that need to communicate.
* host: For standalone containers, remove network isolation between the
container and the Docker host, and use the host’s networking directly.
* overlay: connect multiple Docker daemons together and enable swarm
services to communicate with each other.
* macvlan: allow you to assign a MAC address to a container, making it
appear as a physical device on your network. The Docker daemon routes
traffic to containers by their MAC addresses.
* none: disable all networking.
```

38. What are the networks available for the docker services?

```
* verlay networks: manage communications among the Docker daemons
participating in the swarm.You can attach a service to one or more existing
overlay networks as well, to enable service-to-service communication.
* ingress network: is a special overlay network that facilitates load
balancing among a service’s nodes. When any swarm node receives a request
on a published port, it hands that request off to a module called IPVS.
IPVS keeps track of all the IP addresses participating in that service,
selects one of them, and routes the request to it, over the ingress
network.
* docker_gwbridge: is a bridge network that connects the overlay networks
(including the ingress network) to an individual Docker daemon’s physical
network.
```

39. Is the ingress network created automatically when you initialize or join a swarm?

```
Yes
```

40. Is docker_gwbridge network created automatically when you initialize or join a swarm?

```
Yes
```

41. How to create an overlay network?

```
$ docker network create --driver overlay my-network

// you can customize it
docker network create \
 --driver overlay \
 --subnet 10.0.9.0/24 \
 --gateway 10.0.9.99 \
 my-network
```

42. How to inspect the network?

```
$ docker network inspect my-network
```

43. How to attach a service to an overlay network?

```
$ docker service create \
  --replicas 3 \
  --name my-web \
  --network my-network \
  nginx
```

44. Can service containers connected to the overlay network communicate with each other?

```
Yes
```

45. How to find which networks the service is connected to?

```
$ docker network inspect my-network
             or
$ docker service ls // for the name
$ docker service ps <SERVICE> // to list the networks
```

46. Customize the ingress network involves removing and creating a new one and you need to do that before you create any services in the swarm. Is this statement correct?

```
Yes
```

47. How to remove and create an ingress network?

```
$ docker network rm ingress
$ docker network create \
  --driver overlay \
  --ingress \
  --subnet=10.11.0.0/16 \
  --gateway=10.11.0.2 \
  --opt com.docker.network.mtu=1200 \
  my-ingress
```

48. How can you expose ports?

```
you can map ports between docker host and container using the following
flags:

* -p 8080:80 -> Map TCP port 80 in the container to port 8080 on the Docker
host.
* -p 192.168.1.100:8080:80 -> Map TCP port 80 in the container to port 8080
on the Docker host for connections to host IP 192.168.1.100.
* -p 8080:80/udp -> Map UDP port 80 in the container to port 8080 on the
Docker host.
* -p 8080:80/tcp -p 8080:80/udp -> Map TCP port 80 in the container to TCP
port 8080 on the Docker host, and map UDP port 80 in the container to UDP
port 8080 on the Docker host.
```

Mount volumes

49. What is the difference between -v and --mount flags in terms of creating volumes?

```
Originally, the -v or --volume flag was used for standalone containers and
the --mount flag was used for swarm services. However, starting with Docker
17.06, you can also use --mount with standalone containers. In general, --
mount is more explicit and verbose.
```

50. How to create a service with volume?

```
$ docker service create -d \
  --replicas=4 \
  --name devtest-service \
  --mount source=myvol2,target=/app \
  nginx:latest
```

51. Does docker service create command supports -v or — volume flag?

```
No
```

52. What are the volume drivers?

```
When building fault-tolerant applications, you might need to configure
multiple replicas of the same service to have access to the same files.
Volume drivers allow you to abstract the underlying storage system from the
application logic. For example, if your services use a volume with an NFS
driver, you can update the services to use a different driver, as an
example to store data in the cloud, without changing the application logic.
```

53. How to create a volume with the volume driver?

```
$ docker volume create --driver vieux/sshfs \
  -o sshcmd=test@node2:/home/test \
  -o password=testpassword \
  sshvolume
```

54. How to create a service with volume driver?

```
$ docker service create -d \
  --name nfs-service \
  --mount 'type=volume,source=nfsvolume,target=/app,volume-driver=local,volume-opt=type=nfs,volume-opt=device=:/var/docker-nfs,volume-opt=o=addr=10.0.0.10' \
  nginx:latest
```

Describe and demonstrate how to run replicated and global services

55. What is the difference between replicated and global services?

```
For a replicated service, you specify the number of identical tasks you
want to run while in a global service the service runs one task on every
node. Each time you add a node to the swarm, the orchestrator creates a
task and the scheduler assigns the task to the new node.
```

56. I created a deployment that runs exactly one task on every node. which type of service deployment is this?

```
global
```

57. I created a deployment that runs several identical tasks on nodes. which type of service deployment is this?

```
replicated
```

58. How can you deploy a global service?

```
Using a "--mode" option with "global" value:
```

```
$ docker service create --mode <global>
```

59. How can you deploy a replicated service?

```
The replicated service is the default value when you create a new service.
You can force replicated mode with:
$ docker service create --mode replicated
```

Apply node labels to demonstrate placement of tasks

60. How to update metadata about a node?

```
you can use labels to add metadata about the node
```

61. How to add a label to the node?

```
$ docker node update --label-add foo worker1
// add multiple labels
$ docker node update --label-add foo --label-add bar worker1
```

62. How to remove the label from the node?

```
$ docker node update --label-rm foo worker1
```

63. How to set up the service to divide tasks evenly over different categories of nodes?

```
with "--placement-pref" option.

// example: if we have three datacenters 3 replicas will be placed on each
datacenter
docker service create \
  --replicas 9 \
  --name redis_2 \
  --placement-pref 'spread=node.labels.datacenter' \
  redis:3.0.6
```

64. How to limit your service on particular nodes?

```
with "--constraint" option.

// example: the following limits tasks for the redis service to nodes where
```

```
the node type label equals queue
docker service create \
  --name redis_2 \
  --constraint 'node.labels.type == queue' \
  redis:3.0.6
```

Describe and demonstrate how to use templates with “docker service create”

65. What are the supported flags for creating services with templates?

```
--env
--mount
--hostname
// example
service create --name hosttempl \
    --hostname="{{.Node.Hostname}}-{{.Node.ID}}-{{.Service.Name}}"\
      busybox top
```

Identify the steps needed to troubleshoot a service not deploying

66. If you want to troubleshoot the Universal Control Plane (UCP) clusters what is the best method?

```
it's always best practice to use client bundle to troubleshoot UCP clusters
```

67. What is the general flow when troubleshooting services or clusters?

```
$ docker service ls
$ docker service ps <service>
$ docker service inspect <service>
$ docker inspect <task>
$ docker inspect <container>
$ docker logs <container>
```

Kubernetes

Describe how to deploy containerized workloads as Kubernetes pods and deployments

```
All Kubernetes objects can and should be described in manifests called
Kubernetes YAML files. These YAML files describe all the components and
configurations of your Kubernetes app, and can be used to easily create and
destroy your app in any Kubernetes environment.

1. wrote a very basic Kubernetes YAML (ie, in this Kubernetes YAML file, we
have two objects, separated by the "---"):
    apiVersion: apps/v1
    kind: Deployment
    metadata:
    name: bb-demo
    namespace: default
    spec:
    replicas: 1
    selector:
        matchLabels:
        bb: web
    template:
        metadata:
        labels:
            bb: web
        spec:
        containers:
        - name: bb-site
            image: bulletinboard:1.0
    ---
    apiVersion: v1
    kind: Service
    metadata:
    name: bb-entrypoint
    namespace: default
    spec:
    type: NodePort
    selector:
        bb: web
    ports:
    - port: 8080
        targetPort: 8080
        nodePort: 30001

Kubernetes YAM always follows the same pattern:

* The apiVersion, which indicates the Kubernetes API that parses this
object.
* The kind indicating what sort of object this is
* Some metadata applying things like names to your objects
* The spec specifying all the parameters and configurations of your object.

2. Deploy your application to Kubernetes:

$ kubectl apply -f bb.yaml

3. List your deployments:

$ kubectl get deployments

4. Check for your services:

$ kubectl get services
```

```
5. Open a browser and visit your bulletin board at localhost:30001
6. Once satisfied, tear down your application:

$ kubectl delete -f bb.yaml
```

Describe how to provide configuration to Kubernetes pods using configMaps and secrets

```
In this section we will create secret and config MariaDB with ConfigMap.

Kubernetes has two types of objects that can inject configuration data into
a container when it starts up: Secrets and ConfigMaps.

We need to setup a environment variable (MYSQL_ROOT_PASSWORD) and
configuration file into our MariaDB image container. We will use secret in
order to set the environment variable and ConfigMap to share del MariaDB
configuration.
```

68. How can you create a secret in kubernetes?

```
// you can create secrets in kubernetes from two ways:

1. Create a Secret manually

$ echo -n 'KubernetesRocks!' | base64
S3ViZXJuZXRlc1JvY2tzIQ==

$ vi mysql-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mariadb-root-password
type: Opaque
data:
  password: S3ViZXJuZXRlc1JvY2tzIQ==

$ kubectl apply -f mysql-secret.yaml
secret/mariadb-root-password created

$ kubectl describe secret mariadb-root-password

2. Create a Secret using kubectl create secret command

$ kubectl create secret generic mariadb-user-creds \
      --from-literal=MYSQL_USER=kubeuser\
      --from-literal=MYSQL_PASSWORD=kube-still-rocks
secret/mariadb-user-creds created

// Note the --from-literal, which sets the key name and the value all in
one. You can pass as many --from-literal arguments as you need to create
```

```
one or more key/value pairs in the Secret.
```

69. How can you decode a kubernetes secret?

```
// If the password is not encrypted (example two from above question)
$ kubectl get secret mariadb-root-password -o
jsonpath='{.data.MYSQL_PASSWORD}' | base64 --decode -
kube-still-rocks

// if the password is base64 encripted (example one from above question)
# Returns the base64 encoded secret string
$ kubectl get secret mariadb-root-password -o jsonpath='{.data.password}'
S3ViZXJuZXRlc1JvY2tzIQ==

$ kubectl get secret mariadb-root-password -o jsonpath='{.data.password}' |
base64 --decode -
KubernetesRocks!
```

70. How can you create a ConfigMap on kubernetes?

```
// ConfigMaps can be created in the same ways as Secrets. You can write a
YAML representation of the ConfigMap manually and load it into Kubernetes,
or you can use the kubectl create configmap command to create it from the
command line.

1. Create config map from configuration file

vi max_allowed_packet.cnf
[mysqld]
max_allowed_packet = 64M

$ kubectl create configmap mariadb-config --from-
file=max_allowed_packet.cnf
configmap/mariadb-config created

$ kubectl get configmap mariadb-config

Note: You can define a key other than the file name to use in the data
section of your ConfigMap when using the --from-file argument:

$ kubectl create configmap mariadb-config --from-
file=max_pkts_allowed=max_allowed_packet.cnf

2. Create from CLI:

$ kubectl create configmap mariadb-config --from-
literal=mariadb.max_allowed_packet=64M
```

71. How use the secret created on the above questions?

```
// Secrets and ConfigMaps can be mounted as environment variables or as
files within a container. Following our examples, for the MariaDB
container, you will need to mount the Secrets as environment variables and
the ConfigMap as a file.

1. Secrets: add the following code block into MariaDB Manifest YAML file in
order to use the secret created on step one from secret create question:

env:
spec:
   containers:
   - name: mariadb
     image: docker.io/mariadb:10.4
     env:
       - name: MYSQL_ROOT_PASSWORD
         valueFrom:
           secretKeyRef:
             name: mariadb-root-password
             key: password
    ...

2. Secrets: add the following code block into MariaDB Manifest YAML file in
order to use the secrets created on step two from secret create question:

spec:
  containers:
  - name: mariadb
    image: docker.io/mariadb:10.4
    envFrom:
    - secretRef:
        name: mariadb-user-creds
    ...
```

72. How use the ConfigMap created on the above questions?

```
// Add the max_allowed_packet.cnf file to the Deployment as a volumeMount.

// You can add your ConfigMap as a source by adding it to the volume list
and then adding a volumeMount for it to the container definition:

<...>

  volumeMounts:
  - mountPath: /etc/mysql/conf.d
    name: mariadb-config

<...>
```

```
volumes:
- configMap:
    name: mariadb-config
    items:
      - key: max_allowed_packet.cnf
        path: max_allowed_packet.cnf
  name: mariadb-config-volume

<...>
```

Kubernetes administration questions.

73. True or False? In a pod .yaml file, resource limit of cpu: 0.1 is allowed.

```
True.  This can also be written as 100m.
```

74. We have the below key:value pairs in a .yaml. How do we give a pod access to the secret via a volume?

```
metadata.name: user-pid
data.pid-password: cUae83JHes=
apiVersion: v1
kind: Pod
spec:
    volumes:
 - name: secrets
   secret:
     secretName: user-pid
      volumeMounts:
   - name: secrets
     mountPath: /etc/user-pid
```

75. True or False? A secret can be visible to only one container in a pod.

```
True.  This may be done for security reason, such as this example:
https://kubernetes.io/docs/concepts/configuration/secret/#use-case-secret-visible-to-one-container-in-a-pod
```

76. What are four main types of services?

```
* ClusterIP (Expose the service on a cluster-internal IP, not exposed to anything external to Kubernetes cluster)
* NodePort (Expose the service on each Node's IP at a static port. External callers can call the service)
* LoadBalancer (Provision an external IP to act as a load balancer for the
service.  Exposes a service to external callers)
* ExternalName (Maps a service to a DNS name.  The service doesn't change
IP addresses, but it routes traffic to an external service that does have a
dynamic IP)
```

77. What kubectl command will give you information such as what node and IP address a pod is on? And any failure events?

```
$ kubectl describe pod my-nginx
```

78. What are some of the benefits of Deployments?

```
Deployments support zero-downtime updates by creating and destroying
replica sets and provide rollback functionality
```

79. What is the name of the AWS volume type?

```
awsElasticBlockStore
```

80. What command will create three pod replicas?

```
$ kubectl scale deployment my-deployement --replicas=3
```

81. What specifies that data in a storage provider should not be erased if a PVC is deleted?

```
persistentVolumeReclaimPolicy: Retain
```

82. What does the spec.selector.matchLabels key in a Pod .yaml do?

```
Queries for a template with the specified label in order to use that pod
template
```

83. What command creates a ConfigMap from an env file?

```
$ kubectl create configmap [configmap-name] --from-env-file=[path-to-file]
```

84. What is a LimitRange?

```
A LimitRange specifies min and max limits on cpu and memory for pods in a
namespace. This prevents pods from not being given a limit and consuming
too much memory, thus causing other pods to fail on a node.
```

85. What access mode allows only one client (i.e. one pod) to write to a PV?

```
* ReadWriteOnce
```

86. How does Kubernetes accomplish a no downtime deployment?

```
It spins up new pods and routes traffic to them, then subsequently destroys
the old pods that no longer have traffic
```

87. What command can be used to externally expose a port on a clusterIP service?

```
$ kubectl port-forward service/[service-name] 8080
```

88. What are some zero-downtime deployment options that kubernetes can facilitate?

```
Blue-Green and Canary deployments, among others
```

89. If I have the following in a .yaml, how do I access it in a Pod .yaml?

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-1
spec:
  volumes:
  - name: blob1
    persistentVolumeClaim:
      claimName: pvc-1
```

90. What are the two types of Kubernetes probes?

```
Liveness and readiness
```

91. What is the annotations.last-applied-configuration.key in a .yaml file?

```
It gives details of the resource's configurations.  This allows changes to
be made to a Pod using kubectl apply
```

92. What is a StatefulSet?

```
A StatefulSet manages the deployment and scaling of a set of pods
```

93. What happens to a scheduled pod that cannot have its resource requests met by a node?

```
It remains in the PENDING state.
```

94. What is a risk of using a hostPath volume?

```
It is dependent on the host.  If the host dies, the data is inaccessible
and potentially lost.
```

95. What command will show all running pods, replicasets, and deployments?

```
$ kubectl get all
```

96. How are secrets stored on a node?

```
tmpfs
```

97. Will ‘kubectl delete pod [pod-name]’ remove and recreate a pod, or just remove?

```
It will remove and recreate if there is an active deployment
```

98. True or False? A pod can have multiple volumes attached to it?

```
True
```

99. What is gcePersistentDisk fsType?

```
It is the file system type to use for the volume.
```

100. What does Secret type:Opaque signify?

```
The secret may contain unstructured data.  There are no constraints on the
data.
```

101. What field indicates the query by which nodes are selected to create a local storage PV on?

```
nodeAffinity.required.nodeSelectorTerms
```

102. What is the name of the Azure volume type?

```
azureFile
```

103. What is the difference between a memory request (spec.containers[].resources.requests.memory) and a memory limit (spec.containers[].resources.limits.memory) in a pod .yaml?

```
A pod can use more memory than the memory request amount.  However, if the
memory request amount is higher than the available memory on the node, the
pod will throw an Out Of Memory error.  A memory limit is the maximum
amount of memory that a pod will be allowed to use, even if the node has
more available.
```

104. If a pod has a memory request of 512MiB and a memory limit of 1 GiB, how many pods of this type could be run on a node with 2 GiB of avaiable memory?

```
4.  As the docs say: "A Container is guaranteed to have as much memory as
it requests, but is not allowed to use more memory than its limit".
https://kubernetes.io/docs/tasks/configure-pod-container/assign-memory-
resource/
```

105. What field in a StorageClass .yaml determines what volume plugin is used for creating PVs?

```
provisioner
```

106. What command will show you the details of the secret with name: pid-acct?

```
$ kubectl describe secrets/pid-acct
```

107. What kind of volume is useful for sharing transient data between two containers running on a pod?

```
emptyDir.  This directory will be tied to the lifecycle of the pod.
```

108. What command will show you the details of all ConfigMaps?

```
$ kubectl get cm
```

109. What does the command 'kubectl get deployments -l tier=frontend' do?

```
It lists all deployements with label: tier: frontend
```

110. True or False? A ConfigMap can be loaded through a volume?

```
True.  In the pod .yaml file, specify spec.volumes and
spec.spec.containers.volumeMounts to point to the appropriate ConfigMap
```

111. True or False? Information stored as a Secret is available to pods on all nodes whether the pod requests it or not.

```
False.  The pod has to specifically request the Secret.  This reduces the
risk of an attacker getting access to the information contained in a
secret.
```

112. Which of the following is a cluster-wide storage unit provisioned by an administrator and has a lifecycle independent of pods?

```
PersistentVolume.  (A pod uses a PersistentVolumeClaim to connect to the
persistent volume.)
```

113. What flag in the yaml file will deny a container the ability to write to a volume?

```
volumeMounts.readOnly: true
```

114. What is the difference between port, targetPort, and nodePort keys in a NodePort service .yaml?

```
targetPort is the port the container is running on, port is the port the
service is exposed on in the cluster, and nodePort is the port made
avaiable to external consumers of the service.
```

115. What command will show any limits placed on a deployment?

```
$ kubectl describe deployment [deployment-name]
```

116. What two commands can be used to create a service from file my.service.yml?

```
$ kubectl apply -f my.service.yml OR kubectl create -f my.service.yml
```

117. If I have a ConfigMap .yaml with value metadata.name: db-confg, what value in a pod .yaml will link to this ConfigMap?

```
spec.spec.env[].valueFrom.configMapKeyRef.key: db-confg  OR
spec.spec.env[].envFrom.configMapRef.name: db-confg
The first command is paired with a specific variable name to load one
variable.  The second is used to load all variables from a ConfigMap.
```

118. What command will show a pod's .yaml file?

```
$ kubectl get pod [pod-name] -o yaml
```

119. Check that your pod is up and running

```
$ kubectl get pods
```

120. Check that you get the logs you'd expect for a ping process

```
$ kubectl logs demo
```

121. What .yaml key will ensure a pod does NOT get any traffic for X amount of seconds after deployment?

```
minReadySeconds
```

122. What command will delete a service created from my.service.yml?

```
$ kubectl delete -f my.service.yml
```

123. What is needed to allow the docker container to use the docker-socket volume in the following .yaml?

```
    socket.yaml:
    apiVersion: v1
    kind: Pod
    spec:
    volumes:
    — name: docker-socket
    hostPath:
    path: /var/run/docker.socket
    type: Socket
    containers:
    — name: docker
    image: docker
    command: [“sleep”]
    args: [“100000”]
    volumeMounts:
        - name: docker-socket
            mountPath: /var/run/docker.socket
```

124. What is the acceptable naming convention for port names?

```
Port names must only contain lowercase alphanumeric characters and '-'.
Port names must also start and end with an alphanumeric character.
```

125. What is a container MountPath?

```
The directory where the volume storage resides.
```

126. What entity facilitates dynamic provisioning of Persistent Volumes?

```
Storage Classes.  These can be used to provision Persistent Volumes
programatically instead of having an administrator create the PV.
```

127. What is the default binding mode for a StorageClass?

```
Immediate.  This means that volume binding and dynamic provisioning occur
on creating of the PVC
```

128. What flag controls when Kubernetes pulls an image?

```
imagePullPolicy
```

Image Creation, Management, and Registry (20%)

Describe the use of Dockerfile

```
The docker build command builds an image from a Dockerfile and a context.
The build's context is the set of files at a specified location PATH or
URL. The PATH is a directory on your local filesystem. The URL is a Git
repository location.
```

Describe options, such as add, copy, volumes, expose, entrypoint

129. What is the purpose of the LABEL instruction in the Dockerfile?

```
It adds metadata to the Image
```

130. How to check the labels for the current image?

```
docker inspect // Under Labels section
```

131. The EXPOSE instruction actually publish the port. Is this statement correct?

```
No. The EXPOSE instruction informs Docker that the container listens on the
specified network ports at runtime but this port is only reachable from
inside container Network.
```

132. What should you do to actually publish the ports?

```
use -p flag when running a container
```

133. What is the purpose of the ENV instruction in the Dockerfile?

```
an ENV instruction sets the enviroment value to the key and it is available
for the subsequent build steps and in the running container as well. The
syntaxis of this instruction is:

    ENV <key> <value>
```

134. How to change the environment variables when running containers?

```
$ docker run --env <key>=<value>
```

135. What is the difference between ADD and COPY instructions?

```
* The ADD instruction copies new files, directories or remote file URLs
from <src> and adds them to the filesystem of the image at the path <dest>.
The syntaxis is: ADD [--chown=<user>:<group>] <src>... <dest>

* The COPY instruction copies new files or directories from <src> and adds
them to the filesystem of the container at the path <dest>.
```

136. What is ENTRYPOINT instruction in the Dockerfile?

```
An ENTRYPOINT allows you to configure a container that will run as an
executable.
Command line arguments to docker run <image>, will be appended after all
elements in an exec form ENTRYPOINT, and will override all elements
specified using CMD.
```

137. How can you override the ENTRYPOINT instruction?

```
$ docker run --entrypoint
```

138. What is the VOLUME instruction in the Dockerfile?

```
The VOLUME instruction creates a mount point with the specified name and
marks it as holding externally mounted volumes from native host or other
containers.
```

139. What initializes the newly created Volume?

```
docker run -v
```

140. What is the USER instruction in the Dockerfile?

```
The USER instruction sets the user name (or UID) and optionally the user
group (or GID) to use when running the image and for any RUN, CMD and
ENTRYPOINT instructions that follow it in the Dockerfile.
```

141. What is the WORKDIR instruction in the Dockerfile?

```
The WORKDIR instruction sets the working directory for any RUN, CMD,
ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.
```

142. You have specified multiple WORKDIR instructions in the Dockerfile what is the result WORKDIR?

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
result: /a/b/c
```

143. You have specified multiple WORKDIR instructions in the Dockerfile what is the result WORKDIR?

```
WORKDIR /a
WORKDIR /b
WORKDIR c
RUN pwd
result: /b/c
```

144. What is the ONBUILD instruction in the Dockerfile?

```
ONBUILD register advance instructions to run later, during the next build
stage. The instruction does not otherwise affect the current build.

The ONBUILD instruction adds to the image a trigger instruction to be
executed at a later time, when the image is used as the base for another
build.

This is useful if you are building an image which will be used as a base to
build other images.

The ONBUILD instructions can be inspected with the docker inspect command.

An example: if you create a new python builder image in order to build
```

```
python proyects starting from this new python base image, you can set the
next lines in order to be execute in future builds based on this python
base image.

FROM python:3.9.0-buster
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src

$ docker image build -t pythonbuilder:custom -f Dockerfile.yml .

From now on you can use this base image for compiling new pyhon proyect
version:

1. Copy your code/python version into workdir directory (./)
2. Create a new Dockerfile

    FROM pythonbuilder:custom
    CMD ["echo", "The proyect was compiled!"]

3. Build the new image. During the build, two ONBUILD trigger (ADD and RUN)
will be run compiling the new proyect.
```

145. Which instruction sets the system call signal that will be sent to the container to exit?

```
* Dockerfile
    STOPSIGNAL signal

* CLI:
    $ docker kill --signal=SIGHUP my_container
    $ docker kill -s=HUP my_container
    $ docker kill --signal=1 my_container

This signal can be a valid unsigned number that matches a position in the
kernel's syscall table, for instance 9, or a signal name in the format
SIGNAME, for instance SIGKILL.
```

146. Which instruction let Docker daemon know the health of the container?

```
HEALTHCHECK
```

147. What are all the options that can be provided for the HEALTHCHECK instruction?

```
    --interval=DURATION (default: 30s)
    --timeout=DURATION (default: 30s)
    --start-period=DURATION (default: 0s)
    --retries=N (default: 3)
```

148. What is the SHELL instruction in the Dockerfile?

```
The SHELL instruction allows the default shell used for the shell form of
commands to be overridden. The default shell on Linux is ["/bin/sh", "-c"],
and on Windows is ["cmd", "/S", "/C"]. The SHELL instruction must be
written in JSON form in a Dockerfile, ie SHELL ["/bin/bash", "-c"].

The SHELL instruction can appear multiple times. Each SHELL instruction
overrides all previous SHELL instructions, and affects all subsequent
instructions. For example:

    # Executed as powershell -command Write-Host hello
    SHELL ["powershell", "-command"]
    RUN Write-Host hello

    # Executed as cmd /S /C echo hello
    SHELL ["cmd", "/S", "/C"]
    RUN echo hello
```

Identify and display the main parts of a Dockerfile

149. Which instruction sets the base image for the subsequent builds in the Dokcerfile?

```
FROM
```

150. No instruction can precede FROM in the Dockerfile. Is this statement correct?

```
No. ARG is the only instruction can precede FROM
```

151. What is the ARG instruction in the Dockerfile?

```
The ARG instruction defines a variable that users can pass at build-time to
the builder.

    Syntaxis: ARG <name>[=<default value>].

Also you can use the value of ARG with the docker build command using the -
-build-arg <varname>=<value> flag.

FROM instructions support variables that are declared by any ARG
instructions that occur before the first FROM.

    ARG  CODE_VERSION=latest
    FROM base:${CODE_VERSION}
    CMD  /code/run-app
```

```
To use the default value of an ARG declared before the first FROM, use an
ARG instruction without a value inside of a build stage:

    ARG VERSION=latest
    FROM busybox:$VERSION
    ARG VERSION <-- in order to use in the next step, you must use it.
    RUN echo $VERSION > image_version
```

152. What are the two forms for the RUN instruction?

```
shell form: RUN <command>
exec form: RUN ["executable", "param1", "param2"]
```

153. What does the RUN instruction do in the Dockerfile?

```
The RUN instruction will execute any commands in a new layer on top of the
current image and commit the results.
```

154. How many forms that CMD instruction has?

```
// exec from, the preferred form
* CMD ["executable","param1","param2"]

// as default parameters to ENTRYPOINT
* CMD ["param1","param2"]

// shell form
* CMD command param1 param2
```

155. What is the purpose of the CMD instruction in the Dockerfile?

```
The main purpose of a CMD is to provide defaults for an executing
container. These defaults can include an executable, or they can omit the
executable, in which case you must specify an ENTRYPOINT instruction as
well.
```

156. If CMD instruction provides default arguments for the ENTRYPOINT instruction, both should be specified in JSON format. Is this statement correct?

```
Yes
```

157. How to make your container execute the same executable every time?

```
use ENTRYPOINT in combination with CMD
```

Describe and demonstrate how to create an efficient image via a Dockerfile

158. Create ephemeral containers is considered best practice?

```
Yes
```

159. The RUN command normally utilizes cache from the previous build. Which flag should you specify for the build not to use cache?

```
--no-cache
$ docker build --no-cache .
```

160. Is there any other instruction that can invalidate the cache?

```
Yes. ADD
```

161. What should you do if you want to exclude some files while executing the docker build image and don't want to send all the files to Docker daemon?

```
use .dockerignore file
```

162. What is the best way to drastically reduce the size of an image?

```
Multi Stage Builds
```

163. How do you minimize the number of layers while building the image?

```
Only the instructions RUN, COPY, ADD create layers.
Where possible, use multi-stage builds, and only copy the artifacts you
need into the final image.

Also, sort multi line arguments (one layer)

    RUN apt-get update && apt-get install -y \
    bzr \
    cvs \
    git \
    mercurial \
    subversion
```

164. How to leverage (aprovechar) the build cache?

```
Put instructions that likely to change often at the bottom of the
dockerfile.
```

Describe and demonstrate how to use CLI commands to manage images, such as list, delete, prune, rmi

165. How to remove unused dangling images?

```
$ docker image prune
```

166. How to remove all images which are not used by existing containers?

```
//using -a, --all arguments
$ docker image prune -a
```

167. How to remove an image?

```
$ docker rmi <IMAGE ID>
```

168. How to remove image without deleting the untagged (dangling) parent images?

```
$ docker rmi --no-prune <IMAGE ID>
```

169. How to see the history of the image?

```
$ docker image history
```

170. How to see all local images?

```
$ docker image ls
```

Describe and demonstrate how to inspect images and report specific attributes using filter and format

171. How to format the output of the docker inspect command?

```
by using --format flag
//examples
$ docker inspect \
--format='{{range .NetworkSettings.Networks}}{{.MacAddress}}{{end}}'
$INSTANCE_ID

$ docker inspect --format='{{.LogPath}}' $INSTANCE_ID
```

172. How to limit the scope when pruning images?

```
//by using --filter flag
$ docker image prune -a --filter "until=24h"
```

Describe and demonstrate how to tag an image.

173. How to tag an image?

```
// Syntaxis
$ docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

// by id
$ docker image tag 0e5574283393 fedora/httpd:version1.0

// by name
$ docker image tag httpd fedora/httpd:version1.0

// by name and tag
$ docker tag httpd:test fedora/httpd:version1.0.test

// Different default registry
$ docker tag 0e5574283393 myregistryhost:5000/fedora/httpd:version1.0
```

Describe and demonstrate how to apply a file to create a Docker image

174. How to create a Docker image from archive or stdin?

```
$ docker image load
// example
$ docker image load -i example.tar
```

Describe and demonstrate how to display layers of a Docker image

175. How to display the layers of the Docker image?

```
$ docker image inspect //under Layers section
```

176. Each layer is only a set of differences from the layer before it. The layers are stacked on top of each other. Is this statement about the image correct?

```
Yes
```

177. When you create a container It adds one writable layer on top of all the layers of the image. Is this statement about the image correct?

```
yes, The Container Layer or Thin R/W Layer
```

178. What is the copy-on-write (CoW) strategy?

```
Copy-on-write is a strategy of sharing and copying files for maximum
efficiency. If a file or directory exists in a lower layer within the
image, and another layer (including the writable layer) needs read access
to it, it just uses the existing file. The first time another layer needs
to modify the file (when building the image or running the container), the
file is copied into that layer and modified. This minimizes I/O and the
size of each of the subsequent layers.
```

Describe and demonstrate how to modify an image to a single layer (multi-stage build, single layer)

179. How to modify an image to a single layer?

```
$ docker export <container> > single-layer.tar
$ docker import /path/to/single-layer.tar

// check the history
$ docker image history
```

180. How to create a multi-stage Dockerfile? // With multi-stage builds, you use multiple FROM statements in your Dockerfile. Each FROM instruction can use a different base, and each of them begins a new stage of the build.

Dockerfile:

```
FROM golang:1.7.3
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

$ docker build -t alexellis2/href-counter:latest . // The end result is the same tiny production image as before, with a significant reduction in complexity.

Describe and demonstrate registry functions

181. How to copy an image from the Docker hub to a local repository?

```
// pull an image from the Docker Hub
$ docker pull ubuntu:bionic
// tag an image (rename)
$ docker tag ubuntu:bionic localhost:5000/my-ubuntu
// push the image
$ docker push localhost:5000/my-ubuntu
```

182. How to stop and remove a local registry?

```
$ docker container stop registry \
&& docker container rm -v registry
```

183. How to configure a registry?

```
The Registry configuration is based on a YAML file. You can specify a
configuration variable from the environment by passing -e arguments to your
docker run stanza or from within a Dockerfile using the ENV instruction.

// for example, you have a configuration like this for root directory
storage:
  filesystem:
    rootdirectory: /var/lib/registry

// you can create environment variable like this
REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/somewhere
it will change from /var/lib/registry to /somewhere
```

184. What is the location of the registry configuration file?

```
/etc/docker/registry/config.yml
```

185. How to customize an entire config file of registry?

```
$ docker run -d -p 5000:5000 \
    --restart=always --name registry \
    -v `pwd`/config.yml:/etc/docker/registry/config.yml \
    registry:2
```

Deploy a registry

186. How to run a local registry?

```
$ docker run -d -p 5000:5000 \
--restart=always --name registry registry:2
```

187. How to customize the registry while deploying?

```
// customize published port
$ docker run -d \
  -p 5001:5000 \
  --name registry-test \
  registry:2

// If you want to change the port the registry listens on within the
container
$ docker run -d \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:5001 \
  -p 5001:5001 \
  --name registry-test \
  registry:2

// storage customization
$ docker run -d \
  -p 5000:5000 \
  --restart=always \
  --name registry \
  -v /mnt/registry:/var/lib/registry \
  registry:2
```

Log into a registry

188. How to login to a self-hosted registry?

```
$ docker login localhost:5000
```

189. Where do you configure any credential helpers or credentials for the registry to prevent passing every time you log in?

```
/etc/docker/daemon.json
```

Utilize search in a registry

190. How to limit the number of records when docker search?

```
$ docker search nginx --limit=2
```

191. How to format the docker search?

```
$ docker search --format "{{.Name}}: {{.StarCount}}" nginx
```

/->* Push an image to a registry

```
1. Save the new image by finding the container ID (using docker ps) and
then committing it to a new image name. // Optional

$ docker commit c16378f943fe rhel-httpd

Where:
    - c16378f943fe is the container-id
    - rhel-httpd is the image name

2. Tag the image. If the registry is a private registry (ie registry-
host:5000) use the host name or IP address, and the port of the registry.
// Custom registry: registry-host:5000
$ docker tag rhel-httpd registry-host:5000/myadmin/rhel-httpd

// DockerHub (registry by default)
$ docker tag rhel-httpd myadmin/rhel-httpd:1.0.0

Where:
    - rhel-httpd is the name of the image
    - myadmin is the user of the registry
    - rhel-httpd is the name of the image (repository)
    - 1.0.0 is the custom tag for that image
```

```
3. Push the image
$ docker push registry-host:5000/myadmin/rhel-httpd

//or

$ docker push myadmin/rhel-httpd:1.0.0
```

Pull and delete images from a registry

192. How to pull an image from the repository?

```
$ docker pull [OPTIONS] NAME[:TAG|@DIGEST]
// pulling from docker hub by default
$ docker pull debian
// pulling from other repositories
$ docker pull myregistry.local:5000/testing/test-image
```

193. How to pull an image with multiple tags?

```
// using -a or --all-tags arguments
$ docker pull --all-tags <image>
```

194. How to delete an image from the repository?

```
1. login into DTR web UI
2. go to the TAGS section
3 delete the specific TAG

// you can also delete all images by deleting the entire repository
```

Installation and Configuration (15%)

Demonstrate the ability to upgrade the Docker engine

195. How to upgrade docker-engine?

```
//Debian & Ubuntu
$ sudo apt-get update && sudo apt install docker-ce

// CentOS & Fedora
$ yum update docker-ce
```

Complete setup of repo, select a storage driver, and complete installation of Docker engine on multiple platforms

196. What are the ways to install docker?

```
1. using repositories
2. using DEB package
3. using convience scripts
```

197. What is the recommended way of installing Docker

```
1. Set up docker repositories
2. Install from them for the ease of installation and upgrade tasks.
```

198. What are all the release channels that Docker CE supports?

```
* Stable gives you latest releases for general availability.
* Test gives pre-releases that are ready for testing before general
availability.
* Nightly gives you latest builds of work in progress for the next major
release.
```

199. Where are the Docker-CE binaries available?

```
Docker Engine - Community binaries for a release are available on
download.docker.com as packages for the supported operating systems.
```

200. Where are the Docker-EE binaries available?

```
Docker Hub
```

201. What are the recommended storage drivers on different distributions?

```
* Centos: overlay2
* Fedora: overlay2
* Ubuntu supports overlay2, aufs and btrfs storage drivers. Overlay2 is the
default one
* Debian: overlay2, aufs or devicemapper (older versions)
```

202. How to see the Storage Driver currently in use is?

```
// To see what storage driver Docker is currently using, use docker info
and look for the Storage Driver line:

$ docker info

    Containers: 0
    Images: 0
    Storage Driver: overlay2
    Backing Filesystem: xfs
    <output truncated>
```

203. How install docker-ce in any distribution

```
1. Uninstall older versions
2. Install required libs or packages (requeriments)
3. Set or add the official repository
4. Update the repository
5. Install docker-ce package
```

204. How to install Docker CE on Centos?

```
// uninstall older versions
$ sudo yum remove docker \
                docker-client \
                docker-client-latest \
                docker-common \
                docker-latest \
                docker-latest-logrotate \
                docker-logrotate \
                docker-engine

// install required libs
$ sudo yum install -y yum-utils \
  device-mapper-persistent-data \
  lvm2

// set up the stable repo
$ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

// install
$ sudo yum install docker-ce docker-ce-cli containerd.io

// if you want to install specific versions
$ sudo yum install docker-ce-<VERSION_STRING> docker-ce-cli-
<VERSION_STRING> containerd.io
```

```
// start docker
$ sudo systemctl start docker
```

205. How to install Docker CE on Debian?

```
// uninstall older versions
$ sudo apt-get remove \
    docker \
    docker-engine \
    docker.io \
    containerd \
    runc

// update
$ sudo apt-get update

// install required
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common

// add dockers official gpg key
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key
add -

// set up stable repo
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/debian \
   $(lsb_release -cs) \
   stable"

// update and install
$ sudo apt-get update
$ sudo apt-get install docker-ce \
    docker-ce-cli \
    containerd.io

// if you want to install specific versions
$ sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=
<VERSION_STRING> containerd.io
```

206. How to install Docker CE on Fedora?

```
// uninstall old versions
$ sudo dnf remove docker \
                docker-client \
                docker-client-latest \
                docker-common \
                docker-latest \
                docker-latest-logrotate \
                docker-logrotate \
                docker-selinux \
                docker-engine-selinux \
                docker-engine

// install required packages
$ sudo dnf -y install dnf-plugins-core

// add the stable repo
$ sudo dnf config-manager \
    --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo

// install community version
$ sudo dnf install docker-ce \
    docker-ce-cli \
    containerd.io

// if you want specific versions
$ sudo dnf -y install docker-ce-<VERSION_STRING> docker-ce-cli-
<VERSION_STRING> containerd.io

// start docker
$ sudo systemctl start docker
```

207. How to install Docker CE on Ubuntu?

```
// uninstall old versions
$ sudo apt-get remove docker \
    docker-engine \
    docker.io \
    containerd \
    runc

// update and install required packages
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common

// add official gpg key
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -

// stable repo
```

```
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

// update and install
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io

// if you want specific versions
$ sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=
<VERSION_STRING> containerd.io
```

208. How to add the user to the Docker group and use docker as a non-root user?

```
$ sudo usermod -aG docker your-user
```

209. How to uninstall docker?

```
$ sudo apt-get purge docker-ce
$ sudo rm -rf /var/lib/docker
```

210. Are Images, containers, volumes, or customized configuration files on your host are not automatically removed when you uninstall docker?

```
No. You need to explicitly delete those
```

- Configure logging drivers (splunk, journald, etc)

211. What are logging drivers?

```
Docker has multiple mechanisms to get the logging information from running
docker containers and services. These mechanisms are called logging drivers
```

212. How to configure a logging driver for the Docker daemon so that all the containers use it?

```
configure log-driver in /etc/docker/daemon.json
{
  "log-driver": "syslog"
}
```

213. Whats is the default logging driver?

```
json-file
```

214. If you have configurable options for your logging driver how do you specify?

```
use log-opts in the daemon.json file
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3",
    "labels": "production_status",
    "env": "os,customer"
  }
}
```

215. How to find the logging driver for the Docker daemon?

```
$ docker info --format '{{.LoggingDriver}}'
```

216. How to configure a logging driver for a container?

```
$ docker run -it \
    --log-driver json-file \
    --log-opt max-size=10m \
    alpine ash
```

217. What are the available logging drivers for the Docker CE edition?

```
* json-file
* local
* journald
```

Setup swarm, configure managers, add nodes, and setup backup schedule

218. If the swarm loses the quorum of managers it loses the ability to perform management tasks. Is this statement correct?

```
Yes
```

219. If the swarm loses quorum all the existing tasks and services are all deleted. Is this statement correct?

```
No. All the existing tasks will continue to run. But, new nodes cannot be
added and new tasks can't be created.
```

220. We should use a fixed IP address for the advertise address to prevent the swarm from becoming unstable on machine reboot. Is this statement correct?

```
Yes. If the whole swarm restarts and every manager node subsequently gets a
new IP address, there is no way for any node to contact an existing
manager. Therefore the swarm is hung while nodes try to contact one another
at their old IP addresses.
```

221. You should maintain an odd number of managers in the swarm to support manager node failures. Is this statement correct?

```
Yes.
```

222. I have manager nodes 3, 5, 7, 9. How do you distribute these manager nodes on availability zones so that If you suffer a failure in any of those zones, the swarm should maintain the quorum of manager nodes

```
Manager Nodes            Availability Zones
    3                      1-1-1
    5                      2-2-1
    7                      3-2-2
    9                      3-3-3
```

223. How to drain the swarm node

```
$ docker node update --availability drain <NODE>
```

224. How to cleanly rejoin a manager node in the cluster?

```
1. To demote the node to a worker:
    $ run docker node demote <NODE>
2. To remove the node from the swarm
    $ run docker node rm <NODE>
3. Re-join the node to the swarm with a fresh state:
    $ docker swarm join --token <token> <ip-manager:port>
```

225. How to forcibly remove a node?

```
$ docker node rm --force <NODE>
```

226. If you want to remove a manager node you need to demote it to a worker role first. Is this statement correct?

```
Yes. You must ensure that there is a quorum
```

227. What is the location where swarm managers save the swarm state?

```
/var/lib/docker/swarm
```

228. How to backup the swarm?

```
1. If autolock is enabled. You must unlock the swarm
2. stop the docker on the manager node so that you don't have unpredictable
results
3. save the entire contents of /var/lib/docker/swarm
4. start the manager
```

229. How to restore swarm from the backup?

```
1. shut down the docker on the targeted machine
2. Remove the contents of /var/lib/docker/swarm
3. Restore the /var/lib/docker/swarm directory from the backup
4. Start the docker on the node so that it doesn't connect to old ones
    $ docker swarm init --force-new-cluster
5. Verify the state of the swarm
    $ docker service ls
6. rotate the autolock key
7. Add manager and worker nodes for the required capacity
8. backup this swarm
```

Create and manage user and teams

230. What is a team in the DTR?

```
A team defines the permissions a set of users have for a set of
repositories.
```

231. What are all the permission levels that teams could have?

```
* Read Only: View repository and pull images.
* Read Write: View repository, pull and push images.
* Admin: Manage repository and change its settings, pull and push images.
```

232. What are the Docker hub organizations?

```
Docker Hub organizations let you create teams so you can give your team
access to shared image repositories. Organizations are collections of teams
and repositories that can be managed together.
```

233. What are Docker Hub Teams?

```
Teams are groups of Docker Hub users that belong to an organization.
```

234. Can Docker Hub users belong directly to an organization?

```
No. They belong only to teams within an organization
```

235. What is about owners team?

```
The owners team is a special team that has full access to all repositories
in the organization.
```

Interpret errors to troubleshoot installation issues without assistance

236. Where is the Docker daemon directory?

```
* Linux: /var/lib/docker
* Windows: C:\ProgramData\docker
```

237. How to enable the debugging on Docker daemon

```
1. add this flag in /etc/docker/daemon.json
{
  "debug": true
}
```

```
2. Send a HUP signal to the daemon to cause it to reload its configuration.
$ sudo kill -SIGHUP $(pidof dockerd)
```

238. How to check whether Docker is running?

```
// all these can be used depending on the operating system
$ docker info
$ sudo systemctl is-active docker
$ sudo status docker
$ sudo service docker status
```

Outline the sizing requirements prior to installation

239. What are the hardware and software requirements for UCP?

```
Minimum
1. 8GB of RAM for manager nodes or nodes running DTR
2. 4GB of RAM for worker nodes
3. 3GB of free disk space

Recommended
1. 16GB of RAM for manager nodes or nodes running DTR
2. 4 vCPUs for manager nodes or nodes running DTR
3. 25-100GB of free disk space
```

240. What products that Docker EE contains?

```
UCP
DTR
Docker Engine with enterprise-grade support,
```

Understand namespaces, cgroups, and configuration of certificates

241. Define namespaces and cgroups

```
* namespaces: provides the isolated workspace called the container and
layer of isolation. Each aspect of a container runs in a separate namespace
and its access is limited to that namespace.

* cgroups: a cgroup limits an application to a specific set of resources.
Control groups allow Docker Engine to share available hardware resources to
containers and optionally enforce limits and constraints.
```

242. How can you secure Docker daemon over TLS?

```
//Configure Docker Daemon over TLS.
1. Create a CA

    $ openssl genrsa -aes256 -out ca-key.pem 4096
    $ openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem
    Enter pass phrase for ca-key.pem:

2. Create a server key and certificate signing request (CSR)
    $ openssl genrsa -out server-key.pem 4096
    $ openssl req -subj "/CN=$HOST" -sha256 -new -key server-key.pem -out
server.csr //replace $HOST with the DNS name of yur Docker daemon's host.

3. Sign the public key with our CA:
    $ echo subjectAltName = DNS:$HOST,IP:10.10.10.20,IP:127.0.0.1 >>
extfile.cnf //Since TLS connections can be made through IP address as well
as DNS name, the IP addresses need to be specified when creating the
certificate. Put your IPs
    $ echo extendedKeyUsage = serverAuth >> extfile.cnf //Set the Docker
daemon key’s extended usage attributes to be used only for server
authentication

    $ openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey
ca-key.pem -CAcreateserial -out server-cert.pem -extfile extfile.cnf
    Signature ok
    subject=/CN=your.host.com
    Getting CA Private Key
    Enter pass phrase for ca-key.pem:

5. Create client key and certificate signing request:
    $ openssl genrsa -out key.pem 4096
    $ openssl req -subj '/CN=client' -new -key key.pem -out client.csr

6. Make the key suitable for client authentication
    $ echo extendedKeyUsage = clientAuth > extfile-client.cnf

7. Generate the signed certificate:
    $ openssl x509 -req -days 365 -sha256 \
    -in client.csr \
    -CA ca.pem -CAkey ca-key.pem -CAcreateserial \
    -out cert.pem -extfile extfile-client.cnf

8. Remove the two certificate signing requests and extensions config files:
    $ rm -v client.csr server.csr extfile.cnf extfile-client.cnf

9. To protect your keys from accidental damage, remove their write
permissions.
    $ chmod -v 0400 ca-key.pem key.pem server-key.pem

10. You might want to remove write access to prevent accidental damage
    $ chmod -v 0444 ca.pem server-cert.pem cert.pem

11. Make the Docker daemon only accept connections from clients providing a
```

```
certificate trusted by your CA:
    $ dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --
tlskey=server-key.pem \
    -H=0.0.0.0:2376

12. To connect to Docker and validate its certificate, provide your client
keys, certificates and trusted CA:

    $ docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --
tlskey=key.pem \
    -H=$HOST:2376 version //replace $HOST with the DNS name of yur Docker
daemon's host
```

243. What port use Docker over TLS?

```
Docker over TLS should run on TCP port 2376.
```

Use certificate-based client-server authentication to ensure a Docker daemon has the rights to access images on a registry

244. Where is the location of the custom certificates?

```
/etc/docker/certs.d
$ chmod -v 0444 ca.pem server-cert.pem cert.pem
```

Complete configuration of backups for UCP and DTR

245. What are the ports that DTR uses?

```
80/tcp  -   Web app and API client access to DTR.
443/tcp -   Web app and API client access to DTR.
```

246. DTR needs to be installed on a worker node that is being managed by UCP. You can't install DTR on a standalone Docker engine. Is this statement correct?

```
Yes
```

247. How to backup the UCP

```
To create a UCP backup, run the docker/ucp:2.2.22 backup command on a
single UCP manager
```

```
$ docker container run \
  --log-driver none --rm \
  --interactive \
  --name ucp \
  -v /var/run/docker.sock:/var/run/docker.sock \
  docker/ucp:2.2.22 backup \
  --id <ucp-instance-id> \
  --passphrase "secret" > /tmp/backup.tar
```

248. How to restore the UCP

```
$ docker/ucp:2.2.22 restore --passphrase "secret"
$ docker container run --rm -i --name ucp \
    -v /var/run/docker.sock:/var/run/docker.sock  \
    docker/ucp:2.2.22 restore --passphrase "secret" < /tmp/backup.tar
```

249. You need to backup the UCP on the single manager node since Docker maintains the same UCP state in all the manager nodes. Is this statement correct?

```
Yes
```

250. How to backup the DTR?

```
To perform a backup of a DTR node, run the docker/dtr backup command.
```

251. DTR requires that a majority (n/2 + 1) of its replicas are healthy at all times for it to work. So if a majority of replicas are unhealthy or lost, the only way to restore DTR to a working state is by recovering from a backup. Is this statement correct?

```
Yes
```

Configure the Docker daemon to start on boot

252. How to configure Docker to start on boot?

```
$ sudo systemctl enable docker
```

Networking (15%)

Create a Docker bridge network for a developer to use for their containers

253. How to create a user-defined bridge network?

```
$ docker network create \
    -d bridge \
    --subnet 10.10.10.0/24 \ //optional
    my-network
```

254. How to inspect the user-defined network?

```
$ docker network inspect my-network
```

255. What is the default network that the docker creates automatically?

```
Bridge
```

256. How to inspect the default network bridge?

```
$ docker network inspect bridge
```

257. How to connect to the default bridge network when you create a container?

```
// All containers without a --network specified, are attached to the
default bridge network by default.
$ docker run -dit --name alpine1 alpine ash
```

258. The default bridge network is not recommended for production. Is this statement correct?

```
Yes
```

259. How to list the networks on the Docker machine?

```
$ docker netwrok ls
```

260. How can you attach containers to an existing network?

```
// attach new container (creation time)
$ docker container run --name myweb \
    --network my-network \
    -d \
    -p 8080:80 \
    -p 8443:80 \
    nginx

// attach a running container
$ docker network disconnect bridge my-web2 //disconnet container from
default bridge network
$ docker network connect my-network my-web2 //connect to user-define
network
```

261. How to see which network the containers are connected to?

```
//Knowing the name of the network (my-network in this example)
$ docker container ls --filter=network=my-network \
    --format='{{.Names}}'

//or

$ docker network inspect --format='{{range $container := .Containers}}
{{println $container.Name}}{{end}}' my-network

//by container name (myweb in this example)
$ docker container inspect --format='{{range $k,$v :=
.NetworkSettings.Networks}}{{println $k -}}{{end}}' myweb
```

262. What are the differences between default bridge network and bridge user-defined network?

```
* User-defined bridge network provide automatic DNS resolution between
containers. Containers on the default bridge network can only access each
other by IP addresses, unless you use the --link option, which is
considered legacy. On a user-defined bridge network, containers can resolve
each other by name or alias.
* User-defined bridges provide better isolation.
* Containers can be attached and detached from user-defined networks on the
fly. To remove a container from the default bridge network, you need to
stop the container and recreate it with different network options.
* Each user-defined network creates a configurable bridge like as MTU and
iptables rules.
* Linked containers on the default bridge network share environment
variables. However, there are superior ways to share environment variables
between containers attached to user-defined network:
    - Multiple containers can mount a file or directory containing the
shared information, using a Docker volume.
    - Multiple containers can be started together using docker-compose and
the compose file can define the shared variables.
    - You can use swarm services instead of standalone containers, and take
advantage of shared secrets and configs.
```

Troubleshoot container and engine logs to understand a connectivity issue between containers

263. How to troubleshoot a user-defined network?

```
// Manual troubleshooting
1. Determine the names of the affected Docker swarm services, networks, and
hosts.
2. Examining the user-defined network by nicolaka/netshoot
$ docker run -it --rm \
    --network container:<container_name> \
    nicolaka/netshoot

Steps 3 through 5 are executed inside this shell.

3. Look up all backend IP addresses by DNS name
$ nslookup tasks.<backend_service_name>
4. Test listening ports
$ nc -zvw2 <backend_service_ip> <listening_port>
5. do a reverse name lookup of the affected IP to determine its service
name and task id
$ nslookup <IP_address>
6. On a manager, for the set of all failed service names and tasks, collect
the following:
$ docker service ps <service_name>
$ docker inspect --type task <task_id>
7. For tasks that are still present (inspect --type task returns output),
note their Created and Updated times. Collect Docker daemon logs covering
this time frame from the netshoot host, all managers, and hosts of
unresponsive tasks as identified by their HostIP. If your Docker daemon
logs to journald, for example:
$ journalctl -u docker --no-pager --since "YYYY-MM-DD 00:00" --until "YYYY-
MM-DD 00:00"

More and detailed info in
https://success.mirantis.com/article/troubleshooting-container-networking
```

Publish a port so that an application is accessible externally

264. How to publish a port so that it can be accessed externally?

```
$ docker container run --name CONTAINER \
    -p $HOSTPORT:$CONTAINERPORT
    <image>
```

Identify which IP and port a container is externally accessible on

265. How to list port mappings or a specific mapping for the container?

```
//Standalone container
1. Print only mapped ports
$ docker container inspect --format='HostPort    ContainerPort{{ range
$containerport, $hostport := .NetworkSettings.Ports}}
{{range $hostip := $hostport}}{{$hostip.HostPort}} ->  {{println
$containerport}}{{end}}{{end}}' web

2. Print IP and mappped ports
2.1 List the containers
$ docker ps
2.2 Use this command with container name
$ docker port <CONTAINER NAME>
2.3 Use the specific port
$ docker port <CONTAINER NAME> <specific port>

// Swarm mode
1. List services and get the service_name
$ docker service ls
2. Use the service name
$ docker service inspect --format='{{range $port := .Endpoint.Ports}}
{{$port.PublishedPort}}->{{$port.TargetPort}}/{{$port.Protocol}}{{println}}
{{end}}' <service_name>
8091->80/tcp
8092->80/tcp
```

Describe the different types and use cases for the built-in network drivers

266. What are all the different built-in network drivers?

```
* Bridge Network Driver
* Overlay Network Driver
* MACVLAN Driver
* Host
* None
```

267. What are the Bridge network and its use case?

```
* The bridge driver creates a private network internal to the host so
containers on this network can communicate.
* The bridge driver does the service discovery for us automatically if two
containers are on the same network
* The bridge driver is a local scope driver, which means it only provides
service discovery, IPAM, and connectivity on a single host.
```

268. What is the scope of the bridge network?

```
local
```

269. What are the Overlay network and their use case?

```
The built-in Docker overlay network driver radically simplifies many of the
complexities in multi-host networking.
It is a swarm scope driver, which means that it operates across an entire
Swarm or UCP cluster rather than individual hosts.
```

270. What is the scope of the overlay network?

```
swarm
```

271. What are the MACVLAN network and their use case?

```
The macvlan driver is the newest built-in network driver and offers several
unique characteristics.
It’s a very lightweight driver, because rather than using any Linux
bridging or port mapping, it connects container interfaces directly to host
interfaces.
```

272. What is the scope of the macvlan network?

```
local
```

273. What are the Host network and its use case?

```
With the host driver, a container uses the networking stack of the host.
There is no namespace separation, and all interfaces on the host can be
used directly by the container.
```

274. What is the scope of the host network?

```
local
```

275. What are the None network and its use case?

```
The none driver gives a container its own networking stack and network
namespace but does not configure interfaces inside the container. Without
additional configuration, the container is completely isolated from the
host networking stack.
```

276. What is the scope of the None network?

```
local
```

Understand the Container Network Model and how it interfaces with the Docker engine and network and IPAM drivers

277. The Docker networking architecture is built on a set of interfaces called the Container Networking Model (CNM). Is this statement correct?

```
Yes
```

278. What is a sandbox in the CNM model?

```
A Sandbox contains the configuration of a container's network stack. This
includes the management of the container's interfaces, routing table, and
DNS settings. An implementation of a Sandbox could be a Windows HNS or
Linux Network Namespace, a FreeBSD Jail, or other similar concept. A
Sandbox may contain many endpoints from multiple networks.
```

279. What is an endpoint in the CNM model?

```
An Endpoint joins a Sandbox to a Network. The Endpoint construct exists so
the actual connection to the network can be abstracted away from the
application. This helps maintain portability so that a service can use
different types of network drivers without being concerned with how it's
connected to that network.
```

280. What is a network in the CNM model?

```
The CNM does not specify a Network in terms of the OSI model. An
implementation of a Network could be a Linux bridge, a VLAN, etc. A Network
is a collection of endpoints that have connectivity between them. Endpoints
that are not connected to a network do not have connectivity on a network.
```

281. What part of the Docker that provides the actual implementation that makes networks work?

```
Network Drivers
```

282. What is IPAM drivers?

```
Docker has a native IP Address Management Driver that provides default
subnets or IP addresses for the networks and endpoints if they are not
specified.
```

283. Which network which connects the individual Docker daemon to the other daemons participating in the swarm?

```
docker_gwbridge
```

284. What is the default network created when you create a swarm cluster?

```
ingress
```

Configure Docker to use external DNS

285. How to configure docker to use external DNS?

```
$ vim /etc/docker/daemon.json
    {
    "dns": ["10.0.0.2", "8.8.8.8"]
    }
$ sudo systemctl docker restart
```

Use Docker to load balance HTTP/HTTPs traffic to an application (Configure L7 load balancing with Docker EE)

```
Layer 7 load balancing allows traffic going to host domains like acme.com
to be distributed across specific containers in your environment. With
path-based routing, traffic headed to sub-domains within acme.com (eg.
acme.com/app1 or acme.com/app2) can be separately routed to different sets
of containers. This can be especially useful for optimizing application
performance by driving different requests to different groups of
containers.

Layer 7 routing with Swarm is fully Docker native.It is also designed to be
```

```
both scalable and highly available, meeting the needs of production
applications.

//Prerequisites
* You are operating a Docker Enterprise Edition cluster with the Layer 7
Loadbalancing feature enabled
* You have at least 1 service deployed using Interlock labels for Layer 7
Loadbalancing
* You have a UCP client bundle configured in your shell
In Docker EE you nedd enable Routing Mesh option, select the HTTP and HTTPS
ports and your architecture.
```

286. Which network should handles control and data traffic related to swarm services?

```
ingress
```

Understand and describe the types of traffic that flow between the Docker engine, registry, and UCP controllers

```
//Component and architecture
* UCP starts running a globally scheduled service called ucp-agent (ucp-
agent-win on Windows) which monitors the node where it’s running and starts
and stops UCP services. UCP exposes the standard Docker API so you can
continue using the tools you already know. With UCP you can manage a
kubernetes cluster.

Some services that running on manager nodes are:
    - ucp-agent
    - ucp-auth-api: The centralized service for identity and authentication
used by UCP and DTR.
    - ucp-auth-store: Stores authentication configurations and data for
users, organizations, and teams.
    - ucp-controller: The UCP web server.

* Docker Trusted Registry (DTR) is the enterprise-grade image storage
solution from Docker. Docker Trusted Registry (DTR) is a containerized
application that runs on a Docker Universal Control Plane cluster. Each DTR
replica run on each UCP worker node.

Some DTR internal components are:
    - dtr-api-<replica_id>: Executes the DTR business logic. It serves the
DTR web application and API
    - dtr-garant-<replica_id>: Manages DTR authentication
    - dtr-registry-<replica_id>: Implements the functionality for pulling
and pushing Docker images. It also handles how images are stored
    - dtr-rethinkdb-<replica_id>: A database for persisting repository
metadata

* To allow containers to communicate, when installing DTR the following
```

```
network is created: dtr-ol. This network uses overlay driver and Allows DTR
components running on different nodes to communicate, to replicate DTR
data. DTR supports these storage back ends:
    - NFS
    - AWS S3
    - Cleversafe
    - Google Cloud Storage
    - OpenStack Swift
    - Microsoft Azure

* DTR uses these named volumes for persisting data:
    - dtr-ca-<replica_id>: Root key material for the DTR root CA that
issues certificates
    - dtr-notary-<replica_id>: Certificate and keys for the Notary
components
    - dtr-postgres-<replica_id>: Vulnerability scans data
    - dtr-registry-<replica_id>: Docker images data, if DTR is configured
to store images on the local filesystem
    - dtr-rethink-<replica_id>: Repository metadata
    - dtr-nfs-registry-<replica_id>: Docker images data, if DTR is
configured to store images on NFS

* DTR images store: By default, Docker Trusted Registry stores images on
the filesystem of the node where it is running, but you should configure it
to use a centralized storage backend.

* DTR has a web UI where you can manage settings and user permissions. You
can push and pull images using the standard Docker CLI client or other
tools that can interact with a Docker registry.

* Traffic flow:
1. End users login on deployed DTR (as container inside of Universal
Control Plane). DTR is integrated with UCP and UCP has its own built-in
authentication mechanism and integrates with LDAP services. It also has
role-based access control (RBAC), so that you can control who can access
and make changes to your cluster and applications. UCP integrates with
Docker Trusted Registry (DTR) so that you can keep the Docker images you
use for your applications behind your firewall, where they are safe and
can’t be tampered with.
2. Push and pull images to and from DTR.
3. UCP and DTR running like as container inside Docker EE nodes cluster.
```

Deploy a service on a Docker overlay network

287. How to create a user-defined overlay network for communication among services?

```
$ docker network create -d overlay my-overlay
```

288. How to create an overlay network which can be used by swarm services or standalone containers to communicate with other standalone containers running on other Docker daemons?

```
create with --attachable flag
$ docker network create -d overlay --attachable my-attachable-overlay
```

289. All the swarm management data is encrypted by default. Is this statement correct?

```
Yes
```

290. Is application data on the swarm encrypted by default?

```
No
```

291. How to encrypt application data as well on the swarm?

```
// use --opt=encrypted
$ docker network create --opt encrypted --driver overlay --attachable my-
attachable-multi-host-network
```

292. How to deploy a service on a Docker overlay network?

```
$ docker service create --name overoverlay-service \
-p 8080:80 \
-d \
--network my-attachable-overlay // We have created this network few
question above.
nginx
```

Describe the difference between "host" and "ingress" port publishing mode (Host, Ingress)

293. What is the host port publishing mode?

```
* host mode port publishing exposes ports only on the host where specific
service tasks are running. The port is mapped directly to the container on
that host. Only a single task of a given service can run on each host to
prevent port collision. To publish a service’s port directly on the node
where it is running, use the mode=host option to the --publish flag.

$ docker service create --replicas 3 \
--publish mode=host,published=8080,target=80 nginx

There are a few things to keep in mind when you set up the host mode:
    - If you access a node which is not running a service task, the service
```

```
does not listen on that port.
    - If you expect to run multiple service tasks on each node (such as
when you have 5 nodes but run 10 replicas), you cannot specify a static
target port. Either allow Docker to assign a random high-numbered port (by
leaving off the published), or ensure that only a single instance of the
service runs on a given node, by using a global service rather than a
replicated one, or by using placement constraints.
```

294. What is the ingress port publishing mode?

```
* ingress mode enable a routing mesh. In this mode each node in the swarm
to accept connections on published ports for any service running in the
swarm, even if there’s no task running on the node. The routing mesh routes
all incoming requests to published ports on available nodes to an active
container.

To use the ingress network in the swarm, you need to have the following
ports open between the swarm nodes before you enable swarm mode:
    - Port 7946 TCP/UDP for container network discovery.
    - Port 4789 UDP for the container ingress network.

In order to create a routing mesh (ingress port publishing) you shouldn't
configure the mode=host option with the --publish argument:

$ docker service create --replicas 3 \
--publish published=8080,target=80 nginx
```

Security (15%)

Enable Docker Content Trust

295. What is DCT?

```
Through DCT, image publishers can sign their images and image consumers can
ensure that the images they use are signed.
```

296. What is DCT stand for?

```
Docker Content Trust
```

297. How to enable docker content trust in the Docker CLI?

```
Content trust gives you the ability to verify both the integrity and the
publisher of all the data received from a registry over any channel.
```

```
Docker Content Trust (DCT) provides the ability to use digital signatures
for data sent to and received from remote Docker registries. These
signatures allow client-side or runtime verification of the integrity and
publisher of specific image tags.

Through DCT, image publishers can sign their images and image consumers can
ensure that the images they pull are signed.

DCT is associated with the TAG portion of an image. Is the responsibility
of the image publisher to decide if an image tag is signed or not.

Image consumers can enable DCT to ensure that images they use were signed.
If a consumer enables DCT, they can only pull, run, or build with trusted
images.

Alternatively, once the keys have been imported an image (see the next) can
be pushed with the $ docker push command, by exporting the DCT
environmental variable.

export DOCKER_CONTENT_TRUST=1
$ docker push <dtr-domain>/<repository>/<image>:<tag>

* dtr-domain: Docker Trusted Registry is an on-premises registry that
allows enterprises to store and manage their Docker images on-premise or in
their virtual private cloud (VPC) to meet security or regulatory compliance
requirements.
```

298. How to enable docker content trust so that you can sign images automatically when you use docker push?

```
export DOKCER_CONTENT_TRUST=1
```

Describe the process of signing an image

299. How sign images with DCT?

```
Trust for an image tag is managed through the use of signing keys. A key
set is created when an operation using DCT is first invoked. A key set
consists of the following classes of keys:

* An offline key that is the root of DCT for an image tag
* Repository or tagging keys that sign tags
* Server-managed keys such as the timestamp key, which provides freshness *
Security guarantees for your repository

// Delegation key pair. These keys can be generated locally. The private
key is automatically added to the local trust store (~/.docker/trust/).
$ docker trust key generate NAME
```

```
//or if you have an existing key:
$ docker trust key load key.pem --name <name>

//Next we will need to add the delegation public key to the Notary server.
T his is specific to a particular image repository
$ docker trust signer add --key cert.pem <name> <dtr-domain>/<repository>

// Finally, we will use the delegation private key to sign a particular tag
and push it up to the registry.
$ docker trust sign <dtr-domain>/<repository>/<image>:<tag>

Where:
    - dtr-domain is the host and port of the Docker Registry (empty if you
are using DockerHub)
```

300. How to disable Image signing while pushing an image to the repository?

```
$ docker push [OPTIONS] NAME[:TAG]
--disable-content-trust=true
```

301. How to inspect remote trusted data for a tag?

```
$ docker trust inspect IMAGE[:TAG]
```

302. How to remove remote trusted data for a tag?

```
$ docker trust revoke [OPTIONS] IMAGE[:TAG]
```

Demonstrate that an image passes a security scan

```
Docker image security scanning is a process for finding security
vulnerabilities within your Docker image files.

1. Enable DTR security scanning
    - Log in to your DTR > Settings > Security > Enable scanning > on
2. Provide a security database for the scanner
    - Online mode: by default, security scanning is enabled in Online mode.
Your DTR instance will attempt to download a security database from Docker
server (https://dss-cve-updates.docker.com/) and install it.
    - Offline mode: download the .tar file that contains the security
database and upload it.
3. Set repository scanning mode. Two modes are available:
    - Scan on push & Scan manually: the image is re-scanned on each docker
push to the repository or click on the "Start Scan" links or "Scan button".
    - Scan manually: the image is scanned only when a user with write
```

```
access clicks the "Start Scan" links or "Scan button".
4. Update the CVE scanning database. Docker Security Scanning indexes the
components in your DTR images and compares them against a known CVE
database. When new vulnerabilities are reported, Docker Security Scanning
matches the components in new CVE reports to the indexed components in your
images, and quickly generates an updated report.

How does the scan work?

1. The scanner performs a binary scan on each layer of the image,
identifies the software components in each layer, and indexes the SHA of
each component in a bill-of-materials. A binary scan evaluates the
components on a bit-by-bit level, so vulnerable components are discovered
even if they are statically linked or under a different name.

2. The scan then compares the SHA of each component against the US National
Vulnerability Database that is installed on your DTR instance.

3. Results:
    - The results of these scans are reported for each image tag in a
repository. The Tags tab for each repository includes a summary of the most
recent scan results for each image. Users with administrator access to DTR
can check when the CVE database was last updated from the Security tab in
the DTR Settings pages.

Note: Other image scanners, including Anchore and Clair (the same scanner
used by Quay), can be used to scan individual container images even if they
are not part of a repository. Of course, one-off scanning is not efficient
if you have a large-scale container deployment.
```

Identity roles

303. What is the role?

```
A role is a set of permitted API operations that you can assign to a
specific subject and collection by using a grant.
```

304. What are the default UCP Roles?

```
Default UCP Roles: each role has its own set of permissions. A role is
specifically about listing permissions and should not reflect a team of
people, nor a collection of resources.
    - None: any permissions. The resource will not even show up. It will be
like it does not exist.
    - View Only: inspect and view
    - Scheduler:
    - Restricted Control: create, run, restart, stop and delete
    - Full Control: exec, namespaces, kernel access and host-mounted
```

```
volumes
    - Admin: manage users, assign permissions and change UCP settings
```

Configure RBAC in UCP

305. What is a grant?

```
A grant defines who has how much access to set of resources
```

306. What is the subject?

```
A subject can be user, team, organization and is granted a role for set of
resources
```

307. How to configure a RBCA in UCP?

```
1. Create al necessary users in UCP (ie):
    - frank
    - caroline
    - leonard
    - alice
    - albert
    - helen
2. Define the projects to be deployed:
    - Frontend
    - Backend
2. Create organizations and Teams (ie):
    - Organization_1 (developers):
        - Team_1 (frontend):
            - frank
            - alice
        - Team_2 (backend):
            - leonard
            - alice

    - Organization_2 (devops):
        - Team_3 (operators):
            - alberto
            - helen
        - Team_4 (owners):
            - helen
3. Create collections (deployed units):
    - Collection_1: development
        - Projects: frontend/backend
    - Collection_2: staging
        - Projects: frontend/backend
    - Collection_3: production
```

```
        - Projects: frontend/backend
4. Default UCP Roles: each role has its own set of permissions. A role is
specifically about listing permissions and should not reflect a team of
people, nor a collection of resources.
    - None: any permissions. The resource will not even show up. It will be
like it does not exist.
    - View Only: inspect and view
    - Scheduler:
    - Restricted Control: create, run, restart, stop and delete
    - Full Control: exec, namespaces, kernel access and host-mounted
volumes
    - Admin: manage users, assign permissions and change UCP settings
5. Map role permissions. Map to each environment (collections) for each
team:
    - developers-frontend (Team_1)
        - development/frontend: Role = Restricted Control
        - staging/frontend: Role = View
        - production/frontend: Role = View
        - In all other environments: Role = None
    - developers-backend (Team_2)
        - development/backend: Role = Restricted Control
        - staging/backend: Role = View
        - production/backend: Role = View
        - In all other environment: Role = None
    - devops-operators (Team_3)
        - development/frontend: Role = View
        - development/backend: Role = View
        - staging/frontend: Role = Restricted Control
        - staging/backend: Role = Restricted Control
        - production/frontend: Role = None
        - production/backend: Role = None
    - devops-owners (Team_4)
        - production/frontend: Role = Restricted Control
        - production/backend: Role = Restricted Control
        - In all other environments: Role = View

Note: if anyone tries to change a resource that they have only "view"
permissions to, they will get an "access denied" error message. If they
don't have any permissions for a resource at all, the resource will not
even show up. It will be like it does not exist.
```

Integrate UCP with LDAP/AD

```
UCP has its own built-in authentication mechanism and integrates with LDAP
services. It also has role-based access control (RBAC), so that you can
control who can access and make changes to your cluster and applications.
```

Demonstrate creation of UCP client bundles

308. What is a Client Bundle?

```
A client bundle is a group of certificates downloadable directly from the
Docker Universal Control Plane (UCP) user interface within the admin
section for “My Profile”. This allows you to authorize a remote Docker
engine to a specific user account managed in Docker EE, absorbing all
associated RBAC controls in the process. You can now execute docker swarm
commands from your remote machine that take effect on the remote cluster.

In short: add grant to your local Docker machine to run commands in a
remote cluster managed by UCP over you UCP user.
```

309. What is the easiest way to access or control the UCP?

```
Client Bundle
```

310. How to download a client bundle?

```
// UI
1. Go to UCP
2. With admin user, go to "My Profile"
3. Click Client Bundles and click New Client Bundle to download the
certificate bundle.

// CLI (REST API)
1. $ sudo apt-get update && apt-get install curl jq
2. # Create an environment variable with the user security token
$ AUTHTOKEN=$(curl -sk -d '{"username":"<username>","password":"
<password>"}' https://<ucp-ip>/auth/login | jq -r .auth_token)
3. # Download the client certificate bundle
$ curl -k -H "Authorization: Bearer $AUTHTOKEN" https://<ucp-
ip>/api/clientbundle -o bundle.zip
```

311. How to create a new client bundle

```
// UI
1. Go to UCP
2. With admin user, go to "My Profile"
3. Click Client Bundles and click New Client Bundle to download the
certificate bundle.

// CLI (REST API)
1. Log in to the UCP web UI and navigate to your My Profile page.
2. In the left pane, click Client Bundles and click New Client Bundle to
download the certificate bundle.

$ AUTHTOKEN=$(curl -sk -d '{"username":"username","password":"password"}'
https://ucp-url/auth/login | jq -r .auth_token)
```

```
2. To download a client bundle for one's own account make a request to the
/api/clientbundle endpoint directly:

$ curl -k -H "Authorization: Bearer $AUTHTOKEN" https://ucp-
url/api/clientbundle -o bundle.zip

3. An administrative user can download a client bundle for another user by
appending their username to the request, i.e. for the user account janedoe:

$ curl -k -H "Authorization: Bearer $AUTHTOKEN" https://ucp-
url/api/clientbundle/janedoe -o bundle.zip
```

312. How to use a client bundle

```
Once you’ve downloaded a client certificate bundle to your local computer,
you can use it to authenticate your requests.

1. Navigate to the directory where you downloaded the user bundle, and
unzip it. Then source the env.sh script.

$ unzip ucp-bundle-dave.lauper.zip
$ eval $(<env.sh)

The env.sh script updates the DOCKER_HOST environment variable to make your
local Docker CLI communicate with UCP. It also updates the DOCKER_CERT_PATH
environment variables to use the client certificates that are included in
the client bundle you downloaded.

You can now use the Docker CLI to create services, networks, volumes and
other resources on a swarm that’s managed by UCP.
```

Describe default engine security

313. What is the kernel feature that isolates the processes running in different containers?

```
Namespaces provide the first and most straightforward form of isolation:
processes running within a container cannot see, and even less affect,
processes running in another container, or in the host system. Also, each
container also gets its own network stack, meaning that a container doesn’t
get privileged access to the sockets or interfaces of another container.
```

314. Which kernel feature limits the resources used by docker containers?

```
Control Groups. They implement resource accounting and limiting. They
provide many useful metrics, but they also help ensure that each container
gets its fair share of memory, CPU, disk I/O; and, more importantly, that a
```

```
single container cannot bring the system down by exhausting one of those
resources.
```

315. What is the kernel feature that needed extra configuration?

```
user
```

Describe swarm default security

316. Docker swarm should be secure by default?

```
Yes. The nodes in a swarm use mutual Transport Layer Security (TLS) to
authenticate, authorize, and encrypt the communications with other nodes in
the swarm by default.

The manager node also generates two tokens to use when you join additional
nodes to the swarm: one worker token and one manager token. Each token
includes the digest of the root CA’s certificate and a randomly generated
secret.
```

Describe MTLS

```
Mutual authentication refers to two parties authenticating each other at
the same time, being a default mode of authentication in some protocols.

when a new Swarm gets created it generates a self-signed Certificate
Authority (CA) and issues short-lived[1] certificates to every node,
allowing the use of Mutually Authenticated TLS for node-to-node
communications.

    Manager (CA) -> (MTLS) -> Worker
```

Describe the difference between UCP workers and managers

```
If the node is a:

* Manager: the ucp-agent service automatically starts serving
all UCP components, including the UCP web UI and data stores used by UCP.

* Worker: on worker nodes, the ucp-agent service starts serving
a proxy service that ensures only authorized users and other UCP services
can run Docker commands in that node.
```

```
In short: Ucp-agent service automatically starts serving all UCP components
in manager node and only proxy service in worker node.
```

Describe process to use external certificates with UCP and DTR (UCP from cli, from GUI, print the public certificates), DTR)

317. How do I use an external TLS certificate for UCP?

```
1. Install DTR using --dtr-external-url to specify the load-balancer/Public
Address for DTR.
2. Use the load balancer IP address to go to the WebUI.
3. Go to Settings -> Domain and change your Load Balancer/Public Address to
the hostname ( e.g. dtr.example.com) and replace items in "Show TLS
settings" with your certificates

Certificates to use:
* TLS Certificate: Certificate issued by a Certificate Authority. If there
are any intermediate certificates, they should be included here in the
correct order.
* TLS private key: The key you used to generate your request for a TLS
Certificate.
* TLS CA: The CA authority used to create your TLS certificate (root CA).
```

Storage and Volumes (10%)

State which graph driver should be used on which OS

318. What is the preferred storage driver for all Linux distributions which need no extra configuration?

```
Overlay2
```

319. What is the pluggable architecture that Docker supports for the container writable layer storage?

```
Storage Drivers
```

Demonstrate how to configure devicemapper

320. Which device-mapper driver is used for production environments?

```
direct-lvm
```

321. Which device-mapper driver is used for testing environments?

```
loopback-lvm
```

322. How do you check the current storage driver information?

```
$ docker info
```

323. How do you configure device-mapper?

```
// stop docker
sudo systemctl stop docker
// set the device-mapper in /etc/docker/daemon.json file
{
  "storage-driver": "devicemapper"
}

//start docker
$ sudo systemctl start docker
```

324. what is the option that sets the direct-lvm for production device-mapper?

```
dm.directlvm_device
```

325. What do you set the device-mapper and all configurable options in the daemon.json?

```
{
  "storage-driver": "devicemapper",
  "storage-opts": [
    "dm.directlvm_device=/dev/xdf",
    "dm.thinp_percent=95",
    "dm.thinp_metapercent=1",
    "dm.thinp_autoextend_threshold=80",
    "dm.thinp_autoextend_percent=20",
    "dm.directlvm_device_force=false"
  ]
}
```

Compare object storage to block storage, and explain which one is preferable when available

326. What are the different available storage options available for containers?

```
* Block Storage
* FiLE System Storage
* Object Storage
```

Summarize how an application is composed of layers and where those layers reside on the filesystem

327. Do containers create a writable layer on top of Image read-only layers?

```
Yes
```

328. Where is the Docker's local storage area?

```
/var/lib/docker/<storage-driver>
```

Describe how volumes are used with Docker for persistent storage

329. What should we use if we want to persist the data beyond the lifecycle of the containers?

```
Volumes
```

330. What is the difference between bind mounts and volumes?

```
Volumes are completely managed by docker
Bind Mounts are dependent on the host directory structure
```

331. Volumes don't increase the size of the containers. Is this statement correct and why?

```
Yes. Because volumes live outside of containers
```

Identify the steps you would take to clean up unused images on a filesystem, also on DTR. (image prune, system prune and from DTR)

332. How to remove all unused images not just dangling images?

```
$ docker system prune --all
```

Demonstrate how storage can be used across cluster nodes

333. How to create a Volume?

```
$ docker volume create my-volume
```

334. How to list docker volumes?

```
$ docker volume ls
```

335. How to inspect docker volume?

```
$ docker volume inspect my-vol
```

336. How to remove docker volumes?

```
$ docker volume rm my-vol
```

337. If you start a container with a volume that does not yet exist, Docker creates the volume for you. Is this statement correct?

```
Yes
```

338. How to create a volume myvol2 with a docker run?

```
$ docker run -d \
  --name devtest \
  -v myvol2:/app \
  nginx:latest
```

339. How to verify the volume is created with the container?

```
// Look for the mounts section
$ docker inspect devtest
```

340. How to create a volume with the --mount flag?

```
$ docker run -d \
  --name devtest \
  --mount source=myvol2,target=/app \
  nginx:latest
```