KPLABS Course

Docker Certified Associate 2020

Important Pointers for Exams

ISSUED BY
Zeal Vora

REPRESENTATIVE
instructors@kplabs.in

Module 1: Important CLI Commands

Use-Case	Important Commands
Scale Swarm Service	docker service create --name webserver --replicas 1 nginx
Draining Swarm Node	docker node update --availability drain swarm03 docker node update --availability active swarm03
Docker Stack CLI	docker stack deploy --compose-file docker-compose.yml
Adding Label To Nodes	docker node update --label-add region=mumbai worker-node-id
Placement Constraints	docker service create --name webserver --constraint node.label.region==blr nginx docker service create --name webserver --constraint node.label.region!=blr nginx
Adding Label to Node	docker node update --label-add region=mumbai worker-node-id
Initialize Swarm	docker swarm init
Updating Image of Swarm Service	docker swarm update --image myimage:tag servicename

Module 2: Replicated vs Global Service

Replicated service will have N number of containers defined with the --replica flag

Global service will create 1 container in every node of the cluster.

Module 3: Multiple approaches to scale swarm service

docker service scale allows us to specify multiple services in the same command.

docker service update command only allows us to specify one service per command.

- docker service scale mywebserver=5
- docker service update --replicas 5 mywebserver

Module 4: Overlay Networks & Security

An overlay network allows containers to spread across different servers to communicate.

The communication between containers can be made secured with IPSec tunnels.

docker network create --opt encrypted --driver overlay my-overlay-secure-network

Module 5: Be Aware of Quorum

Cluster Size	Majority	Fault Tolerance
1	0	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

Module 6: Troubleshooting Aspect

- **docker system events** provide you real-time even information from your server.
- Service Deployment would be in a pending state if the node is drained.
- Service deployments can also be in a pending state due to placement constraints.
- You can further inspect the task to see more information.

Module 7: Docker Stack Commands

Child Commands	Description
docker stack deploy	Deploy a new stack or update an existing stack
docker stack ls	List stacks
docker stack ps	List the tasks in the stack
docker stack services	List the services in the stack

Module 8: Docker Service Commands

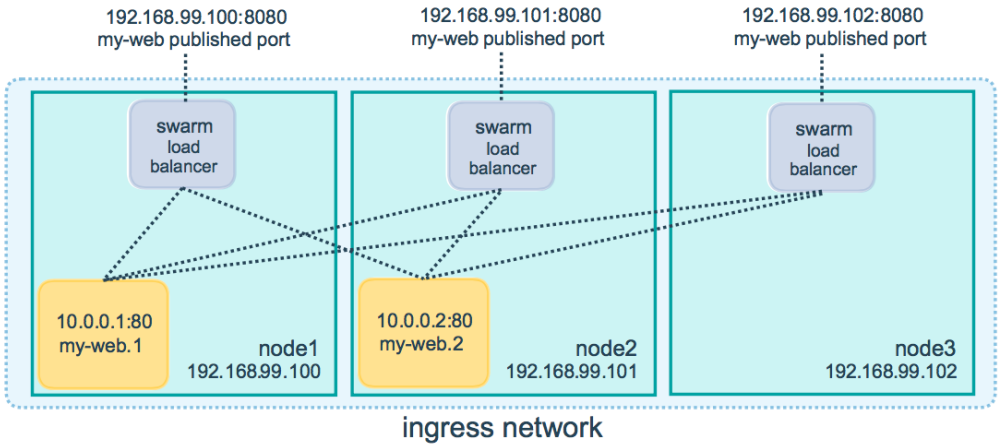
Child Commands	Description
docker service create	Create a new service
docker service inspect	Display detailed information on one or more services
docker service ps	List the tasks of one or more services
docker service update	Update a service
docker service scale	Scale one or multiple replicated services

Module 9: Swarm Routing Mesh

Routing mesh enables 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.

docker service create --name my_web --replicas 3 --publish published=8080,target=80 nginx

Port 8080 traffic is directed to 80



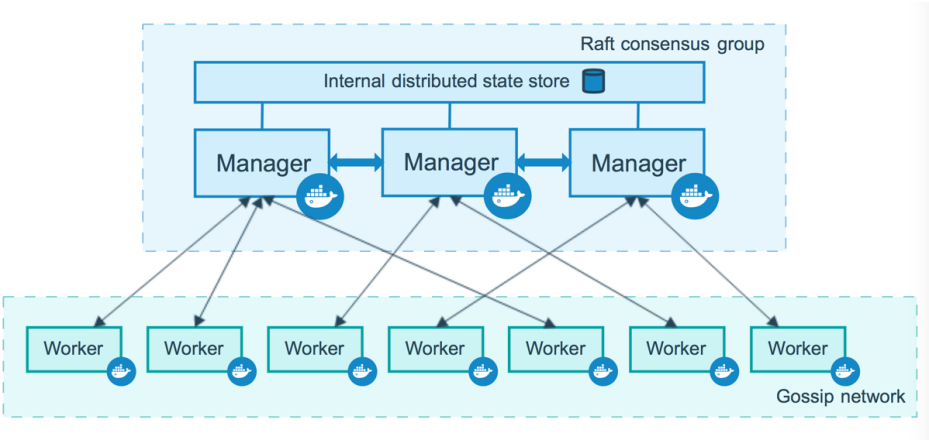
Module 10: Manager and Worker Nodes

Manager Nodes handles cluster management tasks like scheduling services.

Raft implementation is used to maintain a consistent internal state.

The sole purpose of worker nodes is to execute containers.

To prevent the scheduler from placing tasks on manager nodes, set the availability for the manager node to Drain



Module 11: Join Tokens in Swarm

Join tokens are secrets that allow a node to join the swarm.

There are two different join tokens available, one for the worker role and one for the manager role.

docker swarm join-token worker

Module 12: Leaving Swarm Cluster

If a worker intends to leave swarm, he can run the following command:

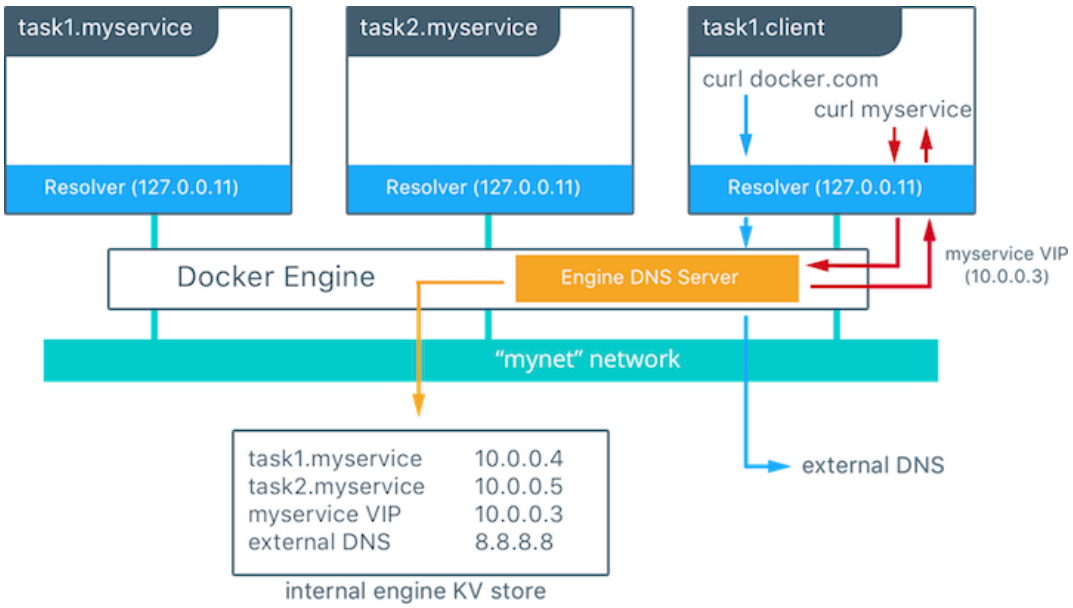
docker swarm leave

You can use the **--force** option on a manager to remove it from the swarm.

Only use --force in situations where the swarm will no longer be used after the manager leaves, such as in a single-node swarm.

Module 13: Service Discovery in Swarm

Docker uses embedded DNS to provide service discovery for containers running on a single Docker engine and tasks running in a Docker swarm



Module 14: Diskspace Stats in Docker

The **docker system df** command displays information regarding the amount of disk space used by the docker daemon.

```
C:\Users\Zeal Vora>docker system df
TYPE                TOTAL               ACTIVE              SIZE                RECLAIMABLE
Images              15                  7                   6.092GB             2.575GB (42%)
Containers          18                  1                   23.21GB             18.33GB (78%)
Local Volumes       88                  3                   32.09GB             30.85GB (96%)
Build Cache         0                   0                   0B                  0B
```

Module 15: System Events in Docker

Use **docker system events** to get real-time events from the server. These events differ per Docker object type.

```
$ docker system events --filter 'event=stop'

2017-01-05T00:40:22.880175420+08:00 container stop 0fdb...ff37 (image=alpine:latest, name=test)
2017-01-05T00:41:17.888104182+08:00 container stop 2a8f...4e78 (image=alpine, name=kickass_brattain)

$ docker system events --filter 'image=alpine'

2017-01-05T00:41:55.784240236+08:00 container create d9cd...4d70 (image=alpine, name=happy_meitner)
2017-01-05T00:41:55.913156783+08:00 container start d9cd...4d70 (image=alpine, name=happy_meitner)
2017-01-05T00:42:01.106875249+08:00 container kill d9cd...4d70 (image=alpine, name=happy_meitner, signal=15)
2017-01-05T00:42:11.111934041+08:00 container kill d9cd...4d70 (image=alpine, name=happy_meitner, signal=9)
2017-01-05T00:42:11.119578204+08:00 container die d9cd...4d70 (exitCode=137, image=alpine, name=happy_meitner)
2017-01-05T00:42:11.173276611+08:00 container stop d9cd...4d70 (image=alpine, name=happy_meitner)
```

Module 16: Inspecting Container

docker container inspect displays detailed information on one or more containers.

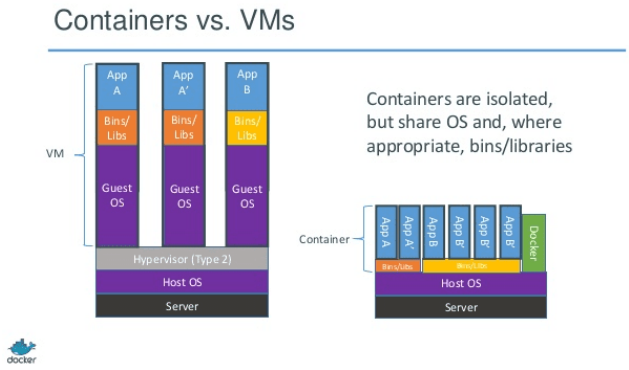
It can also show you a list of volumes that are attached to the container.

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/host_mnt/d/containers/git",
        "Destination": "/gitplace",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

Module 17: Container vs Virtual Machines

Virtual Machine contains the entire Operating System.

The container uses the resource of the host operating system (primarily the kernel)



Module 18: Miscellaneous Pointer

dockerd is a persistent process that manages containers. Docker uses different binaries for the daemon and client. To run the daemon you type dockerd

For specifying the configuration options, you can make use of daemon.json file.

The default location of the configuration file on Linux is /etc/docker/daemon.json

The default location of the configuration file on Windows is %programdata%\docker\config\daemon.json

Module 19: DockerFile Instructions

Have a thorough understanding of various Dockerfile instructions.

- ADD
- COPY
- RUN
- ENTRYPOINT
- WORKDIR
- ENV
- VOLUMES
- CMD
- HEALTHCHECK

Module 20: Difference ADD vs COPY

- COPY allows us to copy files from a local source to a destination.
- ADD allows the same in addition to using URL & extraction capabilities of TAR files.

Module 21: Difference between CMD and ENTRYPOINT

- CMD can be overridden.
- ENTRYPOINT cannot be overridden.

Module 22: Use-Case: Add vs WGET/CURL

Because image size matters, using ADD to fetch packages from remote URLs is strongly discouraged; you should use curl or wget instead.

```
ADD http://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things
RUN make -C /usr/src/things all
```

```
RUN mkdir -p /usr/src/things \
    && curl -SL http://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
     && make -C /usr/src/things all
```

Module 23: WORKDIR Instruction

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

The WORKDIR instruction can be used multiple times in a Dockerfile

Sample Snippet:

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

Output = /a/b/c

Module 24: Format Option

The **format** option allows us to format the output based on various criteria that we have defined with the command

```
[root@ip-172-31-45-184 ~]# docker images --format "{{.ID}}: {{.Repository}}"
87159635da2d: aml-ssh
d131e0fa2585: ubuntu
eaa8f22fecc5: registry.aquasec.com/enforcer
27a188018e18: nginx
af2f74c517aa: busybox
01da4f8f9748: amazonlinux
```

Module 25: Filter Option

The **filter** option allows us to filter output based on the condition provided.

Sample Use-Case:

- Show all the dangling images
- Show all the images created after the N image.
- Show all the containers which are in the exited stage.

Module 26: Accessing in-secure Registries

By default, the docker will not allow you to perform the operation with an insecure registry.
You can override by adding the following stanza within the /etc/docker/daemon.json file

```
{
  "insecure-registries" : ["myregistrydomain.com:5000"]
}
```

Module 27: Pushing an image to a private repository

Description	Commands
Push Image to Private Repository with DNS name of example.com	docker tag ubuntu:latest example.com/myrepo:ubuntu
Login to Private Repository	docker login example.com
Searchability Aspects of Images	docker search nginx docker search nginx --filter “is-official=true”
Save Container State to Image	docker commit c3f279d17e0a container-image:v2
Save Container to TAR Archive (Data is also flattened)	docker export my-container > container.tar

Module 28: Moving Images Across Hosts

The docker save command will save one or more images to a tar archive

docker save busybox > busybox.tar

The docker load command will load an image from a tar archive

docker load < busybox.tar

Module 29: ENV Instruction

The ENV instruction sets the environment variable <key> to the value <value>.

ENV NGINX 1.2
RUN curl -SL http://example.com/web-$NGINX.tar.xz
RUN tar -xzvf web-$NGINX.tar.xz

You can use the -e, --env, and --env-file flags to set simple environment variables in the container you’re running or overwrite variables that are defined in the Dockerfile of the image you’re running.

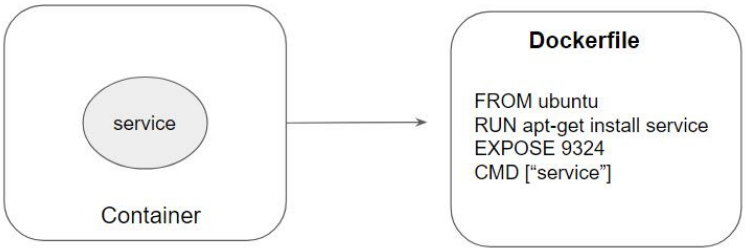
docker run --env VAR1=value1 --env VAR2=value2 ubuntu env | grep VAR

Module 30: EXPOSE Instruction

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.

The EXPOSE instruction does not actually publish the port.

It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.



Module 31: HEALTHCHECK Instruction

HEALTHCHECK instruction in Docker allows us to tell the platform on how to test that our application is healthy.

HEALTHCHECK CMD curl --fail http://localhost || exit 1

That uses the curl command to make an HTTP request inside the container, which checks that the web app in the container does respond.

It exits with a 0 if the response is good, or a 1 if not - which tells Docker the container is unhealthy.

Module 32: Tagging Docker Images

Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Syntax:

docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

Example Snippet:

docker tag httpd fedora/httpd:version1.0

Module 33: Pruning Docker Images

docker image prune command allows us to clean up unused images.

By default, the above command will only clean up dangling images.

Dangling Images = Image without Tags and Image not referenced by any container

Module 34: Downloading Images from Private Repository

If your image is available on a private registry which requires login, use the --with-registry-auth flag with docker service create, after logging in.

docker login registry.example.com

docker service create --with-registry-auth \--name my_service
registry.example.com/acme/my_image:latest

This passes the login token from your local client to the swarm nodes where the service is deployed, using the encrypted WAL logs. With this information, the nodes are able to log into the registry and pull the image.

Module 35: Build Cache

Docker creates container images using layers.

Each command that is found in a Dockerfile creates a new layer.

Docker uses a layer cache to optimize the process of building Docker images and make it faster.

```
[root@swarm02 build-cache]# docker build -t demo2
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM python:3.7-slim-buster
 ---> 87b1022604d5
Step 2/4 : COPY . .
 ---> Using cache
 ---> fe355c27a8ff
```

Module 36: DTR Backup Process

When we backup DTR, there are certain things that are not backed.

User/Organization backup should be taken from UCP.

Data	Description	Backed up
Raft keys	Used to encrypt communication among Swarm nodes and to encrypt and decrypt Raft logs	yes
membership	List of the nodes in the cluster	yes
Services	Stacks and services stored in Swarm-mode	yes
(overlay) networks	The overlay networks created on the cluster	yes
configs	The configs created in the cluster	yes
secrets	Secrets saved in the cluster	yes
Swarm unlock key	**Must be saved on a password manager !**	no

Module 37: Namespaces

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

User namespace is not enabled by default

Module 38: Control Groups (cgroups)

Control Groups (cgroups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

Approaches	Description
--cpus=<value>	If host has 2 CPUs and if you set --cpus=1, than container is guaranteed at most one CPU. You can even specify --cpus=0.5
--cpuset-cpus	Limit the specific CPUs or cores a container can use.A comma-separated list or hyphen-separated range of CPUs a container can use. 1st CPU is numbered 0 2nd CPU is numbered 1. Value of 0-3 means usage of first, second, third and fourth CPU. Value of 1,3 means second and fourth CPU.

Module 39: Reservation and Limits

The limit is a hard limit.

Reservation is a soft limit.

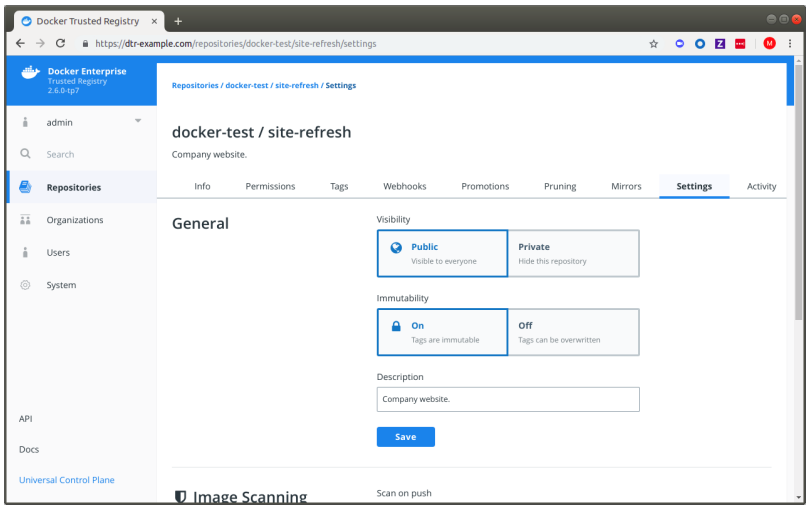
docker container run -dt --name container01 --memory-reservation 250m ubuntu
docker container run -dt --name container02 -m 500m ubuntu

Denoted by --memory and --memory-reservation

Module 40: Immutable Tags in DTR

By default, users with read and write access can overwrite tags.

To prevent tags from being overwritten, we can configure the repository to be immutable.

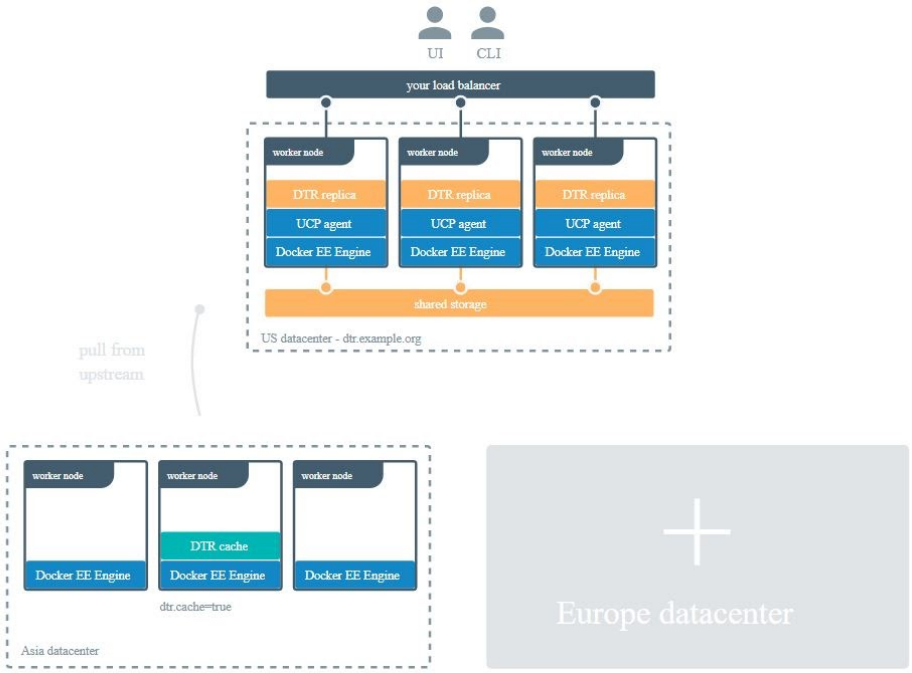


Module 41: DTR Cache

To decrease the time to pull an image, you can deploy DTR caches geographically closer to users.

Caches are transparent to users since users still log in and pull images using the DTR URL address. DTR checks if users are authorized to pull the image, and redirects the request to the cache.

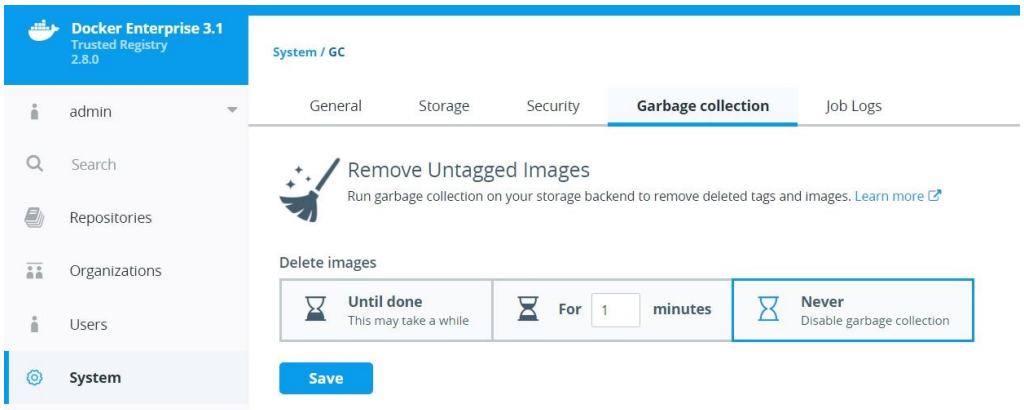
Module 42: DTR Architecture



Module 43: DTR Garbage Collection

You can configure the Docker Trusted Registry (DTR) to automatically delete unused image layers, thus saving you disk space.

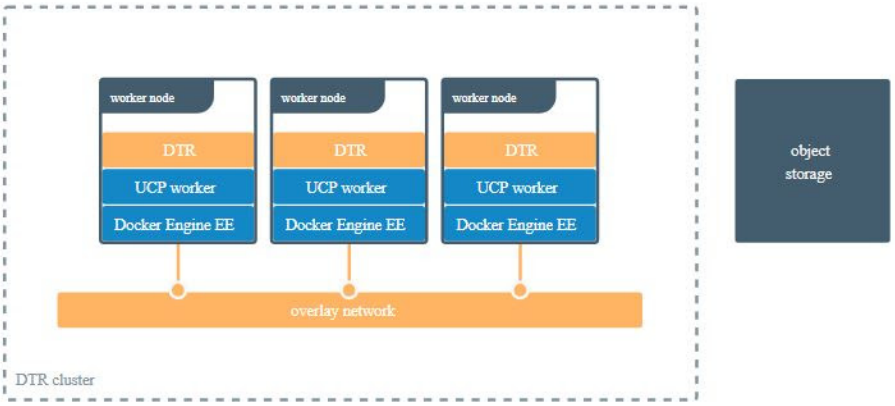
This process is also known as garbage collection.



Module 44: DTR High-Availability

Docker Trusted Registry is designed to scale horizontally as your usage increases. You can add more replicas to make the DTR scale to your demand and for high availability.

If your DTR deployment has multiple replicas, for high availability, you need to ensure all replicas are using the same storage backend.



Module 45: HA of UCP And DTR

To have high-availability on UCP and DTR, you need a minimum of:

- 3 dedicated nodes to install UCP with high availability,
- 3 dedicated nodes to install DTR with high availability,

You can monitor the status of UCP by using the web UI or the CLI. You can also use the _ping endpoint to build monitoring automation.

Module 46: Orchestrator Types in UCP

Docker UCP supports both Swarm and Kubernetes.

When you install Docker Enterprise, new nodes are managed by Docker Swarm, but you can change the default orchestrator to Kubernetes in the administrator settings.

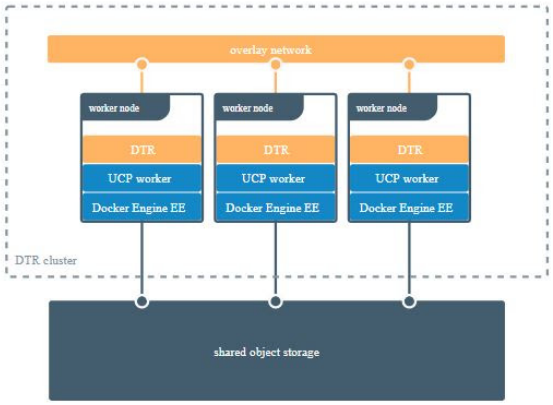
To change the orchestrator type for a node from Swarm to Kubernetes:

docker node update --label-add com.docker.ucp.orchestrator.kubernetes=true <node-id>

Module 47: Storage Driver in DTR

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.

You can configure DTR to use an external storage backend, for improved performance or high availability.



The following are the supported storage system:

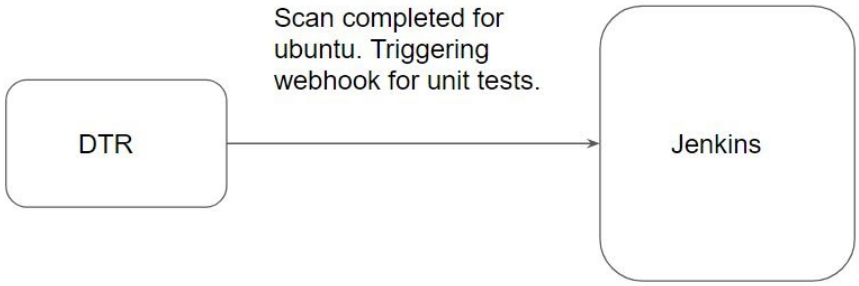
Some of the supported storage systems in DTR are:

- Local:
- NFS
- Bind Mount
- Volume

- Cloud Storage Provider:
- AWS S3
- Azure
- Google Cloud

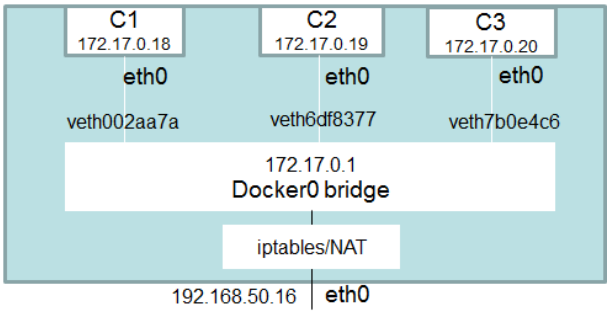
Module 48: DTR WebHooks

You can configure DTR to automatically post-event notifications to a webhook URL of your choosing



Module 49: Bridge Network

The bridge is the default network driver for Docker.



Module 50: Host Network

This driver removes the network isolation between the docker host and the docker containers to use the host's networking directly.

Module 51: Overlay Network

- Default in Swarm.
- Allows containers across host to communicate with each other.
- Communication can be encrypted with --opt encrypted option.
- Do not confuse, -o is same as --opt

You don't need to create the overlay network on the other nodes, because it will be automatically created when one of those nodes starts running a service task which requires it.

Module 52: Difference between -p and -P

- Publish List (-p) will publish a list of ports that you define. [-p 80:80]
- Publish All (-P) will assign random ports for all exposed ports of the container.
- -P will map the container port to a random port above 32768

Module 53: None Network

If you want to completely disable the networking stack on a container, you can use the none network.

This mode will not configure any IP for the container and doesn't have any access to the external network as well as for other containers.

Module 54: Configuring Docker for External DNS

Docker Container's DNS configuration is taken from the host's /etc/resolv.conf DNS settings for containers be customized via daemon.json file.

```
{
    "dns": ["8.8.8.8", "172.31.0.2"]
}
```

Module 55:Inspecting Network Details

To Fetch information about a specific network, you can run the following command:

docker network inspect <network-name>

```
C:\Users\Zeal Vora>docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "9631036a79ff6cabfdb7a70e68de32aa4f3fc4cc1b1ecc5ecfa4dbfc07a5114f",
        "Created": "2020-09-17T05:49:51.250002649Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
```

Module 56: UCP Client Bundles

Client bundles are a group of certificates and files downloaded from UCP.

Depending on the permission associated with the user, you can now execute docker swarm commands from your remote machine that take effect on the remote cluster.

Module 57: Docker Content Trust

Used for interacting with only trusted images (signed images)

Enabled with export DOCKER_CONTENT_TRUST=1

Example Dockerfile:

FROM myubuntu:latest
RUN apt-get install net-tools
CMD[“bash”]

If DCT is enabled, the following actions will be blocked:

- docker container run of an unsigned or altered image.
- docker pull of an unsigned or altered image.
- docker build where the FROM image is not signed or is not scratch.

Module 58: Docker Secrets

To create a new secret, docker secret create command can be used.

This is a cluster management command, and must be executed on a swarm manager node.

Remember that you cannot update or rename a secret.

--secret-add and --secret-rm flags for docker service update

Module 59: Secrets Mount Path

When you grant a newly-created or running service access to a secret, the decrypted secret is mounted into the container in the following path:

/run/secrets/<secret_name>

We can also specify a custom location for the secret.

docker service create --name redis --secret source=mysecret,target=/root/secretdata

Module 60: Swarm Auto-Lock

If your Swarm is compromised and if data is stored in plain-text, an attack can get all the sensitive information.

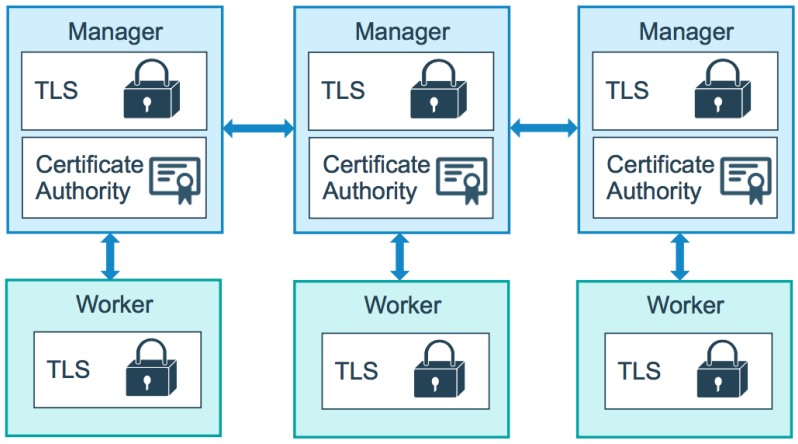
Docker Lock allows us to have control over the keys.

docker swarm update --autolock=true

Module 61: Mutual TLS Authentication

When you create a swarm by running **docker swarm init**, Docker designates itself as a manager node.

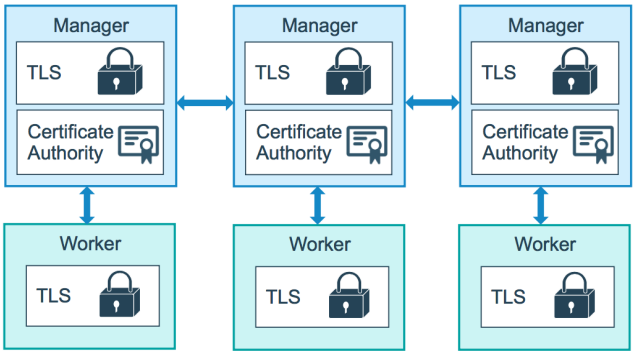
By default, the manager node generates a new root Certificate Authority (CA) along with a key pair, which are used to secure communications



Module 62: Certificates for Each Node

Each time a new node joins the swarm, the manager issues a certificate to the node.

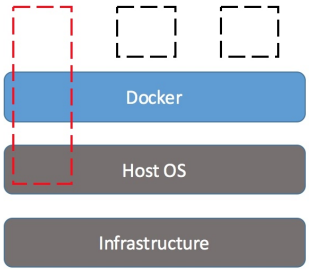
By default, each node in the swarm renews its certificate every three months.



Module 63: Privileged Containers

By default, Docker containers are “unprivileged”

When the operator executes docker run --privileged, Docker will enable access to all devices on the to allow the container nearly all the same access to the host as processes running outside containers on the host.



Module 64: Roles in UCP

Built-in role	Description
None	Users have no access to Swarm or Kubernetes resources. Maps to No Access role in UCP 2.1.x.
View Only	Users can view resources but can't create them.
Restricted Control	Users can view and edit resources but can't run a service or container in a way that affects the node where it's running. Users cannot mount a node directory, exec into containers, or run containers in privileged mode or with additional kernel capabilities
Scheduler	Users can view nodes (worker and manager) and schedule (not view) workloads on these nodes. By default, all users are granted the Scheduler role against the /Shared collection
Full Control	Users can view and edit all granted resources. They can create containers without any restriction, but can't see the containers of other users.

Module 65: Mounting Volumes Inside Container

docker container run -dt --name webserver -v myvolume:/etc busybox sh
docker container run -dt --name webserver --mount source=myvolume,target=/etc busybox sh

You can even mount the same volume to multiple containers.

Also know what bind mounts are all about
How you can map a directory from host to container.

Module 66: Remove Volume on Container Exit

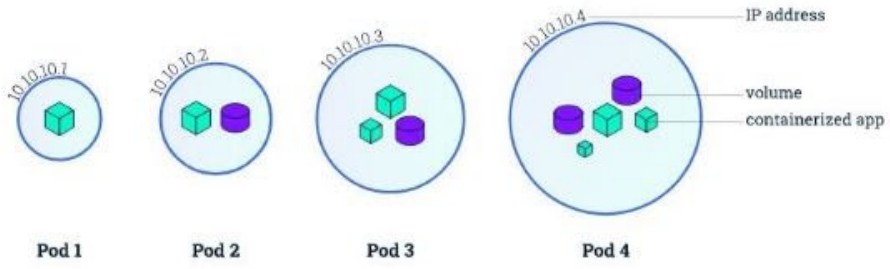
If you have not named the volume, then when you specify --rm option, the volumes associated will also be deleted when the container is deleted.

Module 67: Device Mapper

- loop-lvm mode is for testing purposes only.
- direct-lvm mode can be used in production.

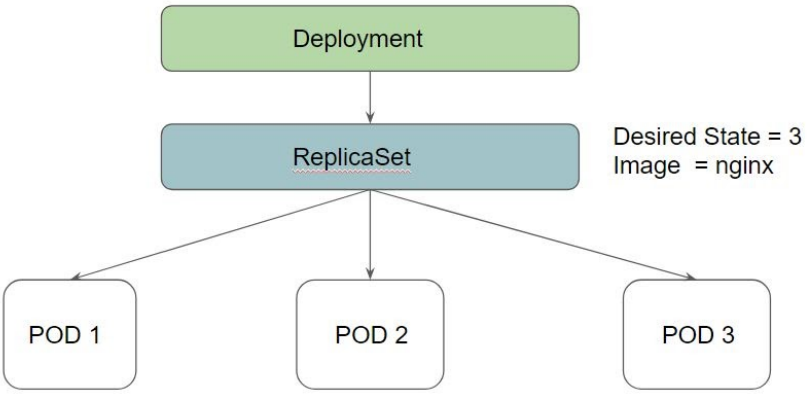
Module 68: K8s Pods

A Pod in Kubernetes represents a group of one or more application containers and some shared resources for those containers.



Module 69: Deployments

Deployments provide replication functionality with the help of ReplicaSets, along with various additional capability like rolling out of changes, rollback changes if required.



Module 70: Rolling Update - Deployment

The Deployment updates Pods in a rolling update fashion

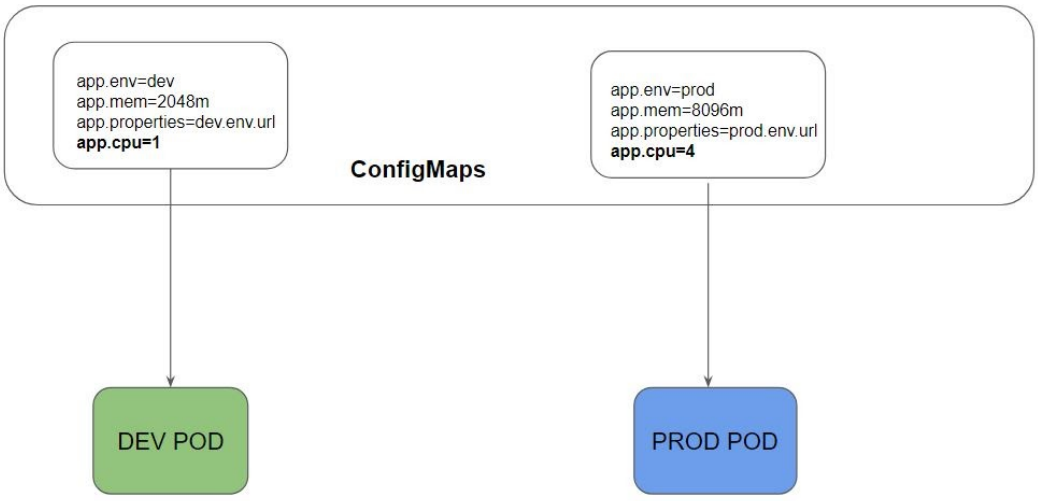
Example:

If you update the image of deployment from nginx to apache, not all of the pods in deployments are brought down together.

You can specify maxUnavailable and maxSurge to control the rolling update process

Module 71: ConfigMaps

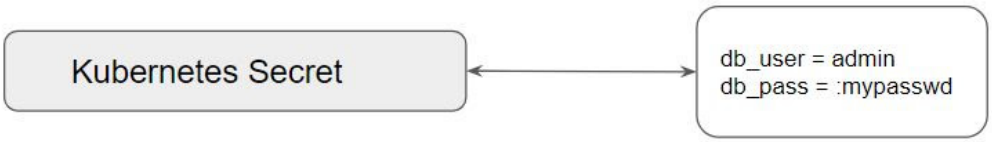
ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.



Module 72: K8s Secrets

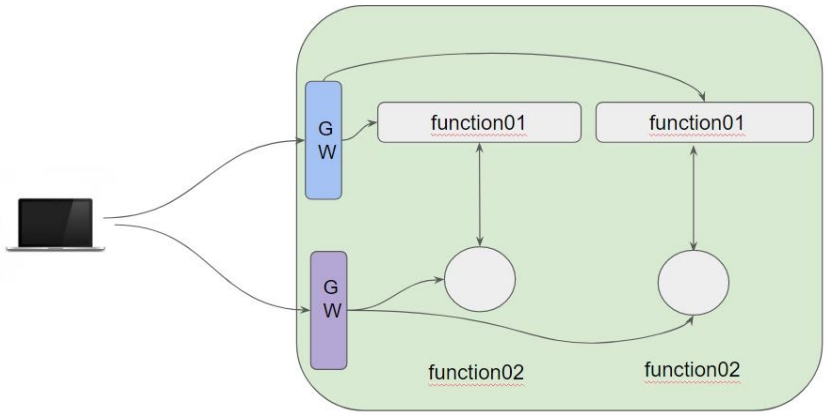
A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.

- Allows customers to store secrets centrally to reduce the risk of exposure.
- Stored in the ETCD database.



Module 73: Kubernetes Service

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them



Module 74: Service Type - Cluster IP

Whenever the service type is ClusterIP, an internal cluster IP address is assigned to the service.

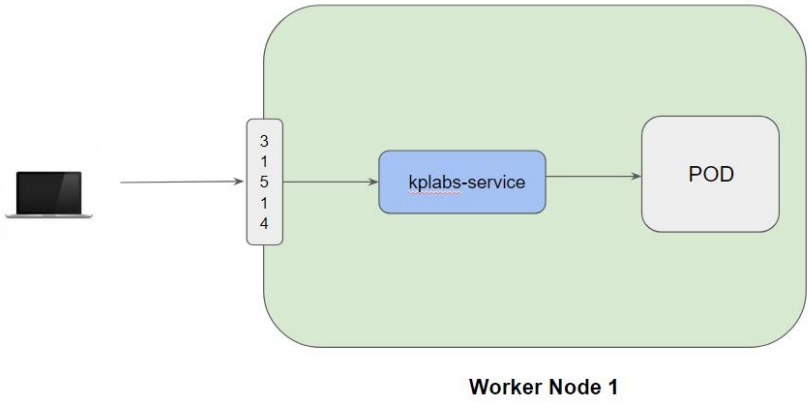
Since an internal cluster IP is assigned, it can only be reachable from within the cluster.

This is a default ServiceType.

Module 75: Service Type - NodePort

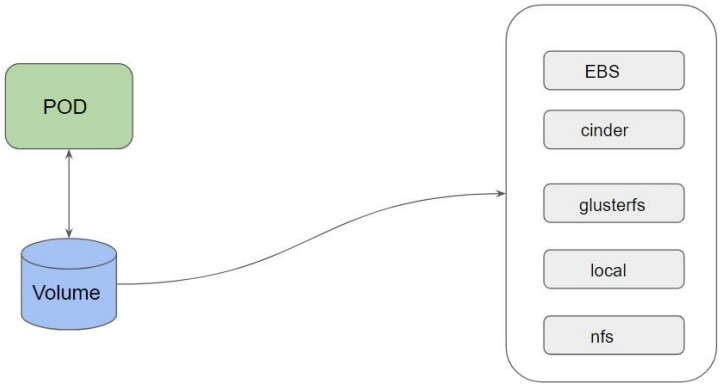
If the service type is NodePort, then Kubernetes will allocate a port (default: 30000-32767) on every worker node.

Each node will proxy that port into your service.



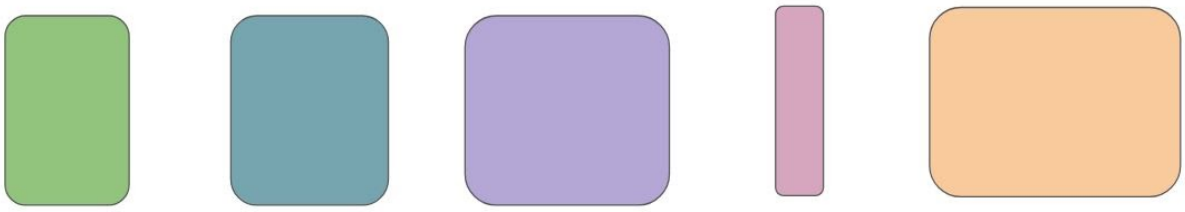
Module 76: Volume in Kubernetes

One of the benefits of Kubernetes is that it supports multiple types of volumes.



Module 77: Persistent Volumes

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes



Volume 1
Size: Small
Speed: Fast

Volume 2
Size: Medium
Speed: Fast

Volume 3
Size: Big
Speed: Slow

Volume 4
Size: VSmall
Speed: Ultra Fast

Volume 5
Size: Very Big
Speed: Very Slow

Module 78: Dynamic Volume Provisioning

Dynamic volume provisioning allows storage volumes to be created on-demand

The dynamic provisioning feature eliminates the need for cluster administrators to pre-provision storage. Instead, it automatically provisions storage when it is requested by users.

Module 79: Persistent Volume Claim

A PersistentVolumeClaim is a request for the storage by a user.

Within the claim, the user needs to specify the size of the volume along with access mode.

Developer:

I want a volume of size 10 GB which has a speed of Fast for my pod.

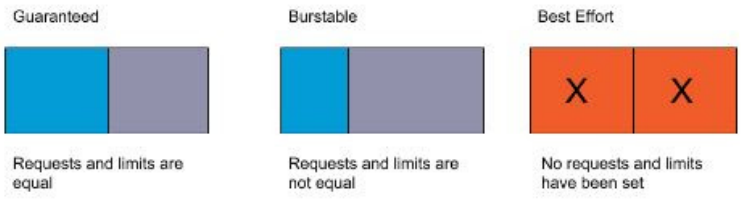
Module 80: Requests & Limits

Requests and Limits are two ways in which we can control the amount of resource that can be assigned to a pod (resource like CPU and Memory)

Requests: Guaranteed to get.
Limits: Makes sure that the container does not take node resources above a specific value.

Kubernetes Scheduler decides the ideal node to run the pod depending on the requests and limits.

If your POD requires 8GB of RAM, however, there are no nodes within your cluster which has 8GB RAM, then your pod will never get scheduled.



Module 81: Kubernetes Labels

Labels are key/value pairs that are attached to objects, such as pods.

List nodes within the K8s cluster along with labels:

kubectl get nodes --show-labels

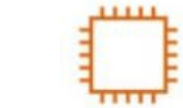
```
NAME      STATUS    ROLES    AGE     VERSION       LABELS
worker0   Ready     <none>   1d      v1.13.0       ...,kubernetes.io/hostname=worker0
worker1   Ready     <none>   1d      v1.13.0       ...,kubernetes.io/hostname=worker1
worker2   Ready     <none>   1d      v1.13.0       ...,kubernetes.io/hostname=worker2
```

Module 82: Kubernetes Selectors

Selectors allow us to filter objects based on labels.

Show me all the pods which have a label where env: production

kubectl get pods -l env=production



name: kplabs-gateway
env: production

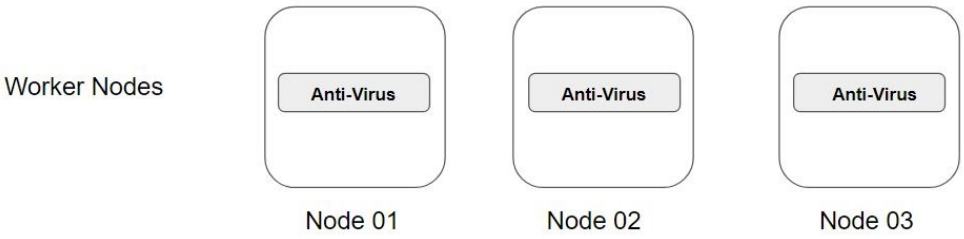
name: kplabs-db
env: production

name: kplabs-elb
env: production

Module 83: Daemonsets

A DaemonSet can ensure that all Nodes run a copy of a Pod.

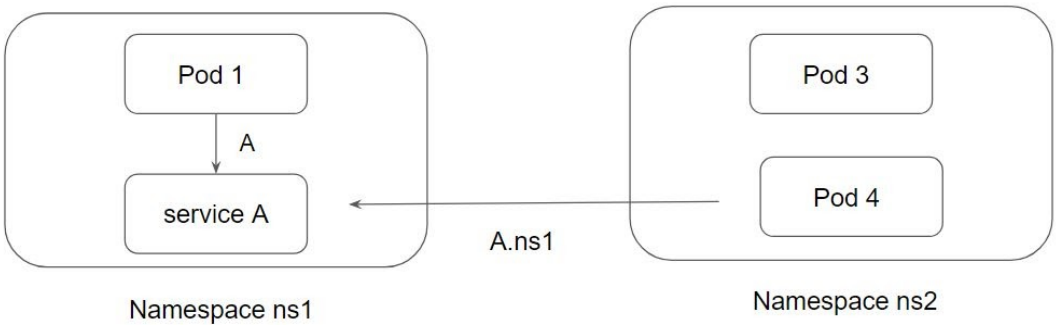
As nodes are added to the cluster, Pods are added to them.



Module 84: DNS Service for Pods

In Kubernetes, communication can happen via DNS.

For SVC in ns1, POD in the same namespace can communicate with just A. Pod in ns2 namespace can make use of a.ns1 <service.namespace>



Module 85: Taints and Tolerations

Taints are used to repel the pods from a specific node.

In order to enter the taint worker node, you need a special pass. This is referred to as Toleration.

Taints	Description
NoSchedule	No pod will be able to schedule onto node1 unless it has a matching toleration. Pods that are already running will continue to run.
NoExecute	Pod is evicted from the node if it is already running on the node

Module 86: Readiness Probe

It can happen that an application is running but temporarily unavailable to serve traffic.

For example, the application is running but it is still loading it's large configuration files from external vendors.

In such a case, we don't want to kill the container however we also do not want it to serve the traffic.

Module 87: Reclaim Policy

PersistentVolumes can have various reclaim policies, including “Retain”, “Recycle”, and “Delete”.

Reclaim Policy	Description
Retain	If a user deletes a PersistentVolumeClaim, the corresponding PersistentVolume is not be deleted. Instead, it is moved to the Released phase, where all of its data can be manually recovered.
Recycle	If supported by the underlying volume plugin, the Recycle reclaim policy performs a basic scrub (rm -rf /thevolume/*) on the volume and makes it available again for a new claim.
Delete	Dynamically provisioned volume is automatically deleted when a user deletes the corresponding PersistentVolumeClaim.

Module 88: Expanding PVC

You can only expand a PVC if its storage class allowVolumeExpansion field is set to true.

To request a larger volume for a PVC, edit the PVC object and specify a larger size. This triggers the expansion of the volume that backs the underlying PersistentVolume

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gluster-vol-default
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://192.168.10.100:8080"
  restuser: ""
  secretNamespace: ""
  secretName: ""
allowVolumeExpansion: true
```

Module 89: Storage Classes

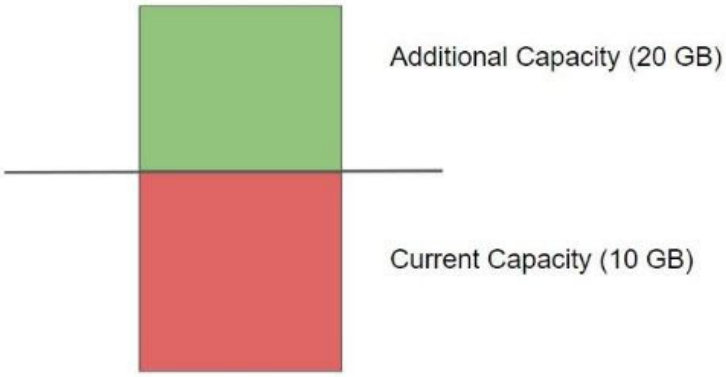
A StorageClass provides a way for administrators to describe the "classes" of storage they offer.

Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class needs to be dynamically provisioned.

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

Module 90: Volume Expansion Steps

1. Enable Volume Expansion in Storage Class (allowVolumeExpansion: true)
2. Resize the PersistentVolumeClaim
3. Restart the POD.



Module 91: Retail Reclaim Policy

When PVC is deleted, the PersistentVolume still exists and the volume is considered "released".

It is not yet available for another claim because the previous claimant's data remains on the volume.

You should know the basic steps for reclamation.

```
bash-4.2# kubectl get pv
kNAME                                     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM
 STORAGECLASS       REASON   AGE
pvc-41872ac3-55b7-412c-ac2c-a44e0a04fe33   5Gi        RWO            Retain           Released   default/kplabs-pvc
```

Module 92: Volume vs Persistent Volume

There are two primary types of storage abstractions:

- Volume
- Persistent Volume
-

Kubernetes volume exists only while the containing pod exists. Once the pod is deleted, the associated volume is also deleted. Does not exists outside of the pod's lifecycle. Used for storing temporary data.

Kubernetes persistent volumes remain available outside of the pod lifecycle. PV will remain even after the pod is deleted

Module 93: HEALTHCHECK Instruction Options

HEALTHCHECK Instruction is combined with various options to get the desired results.

Options	Default Values
--interval=DURATION	30 seconds
--timeout=DURATION	30 seconds
--start-period=DURATION	0 seconds
--retries=N	3

Which one among these will be able to determine health check within 1 minute?

--interval=30s

--interval=10s --timeout=5s retries=3

Module 94: K8s Namespaces and Selectors

To list the PODS from all the namespaces in Kubernetes, the following commands needs to be used:

kubectl get pods --all-namespaces -l env=development

Here -l is the selector.

Module 95: Multi-Stage Builds

Multi-Stage builds allow users to use artifacts from one image to another.

```
FROM golang:1.14

RUN git clone https://github.com/coredns/coredns.git /coredns
RUN cd /coredns && make

FROM scratch
COPY --from=0 /coredns/coredns /coredns

EXPOSE 53 53/udp
CMD ["/coredns"]
```