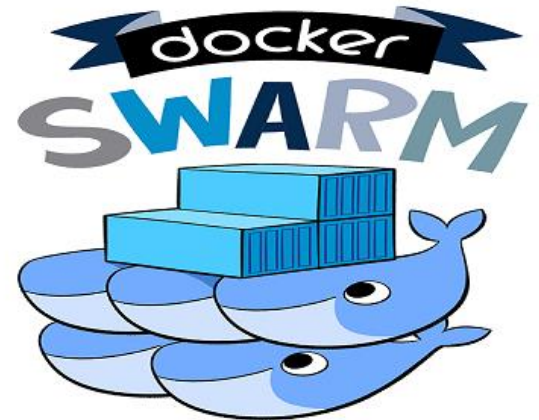


Docker Swarm

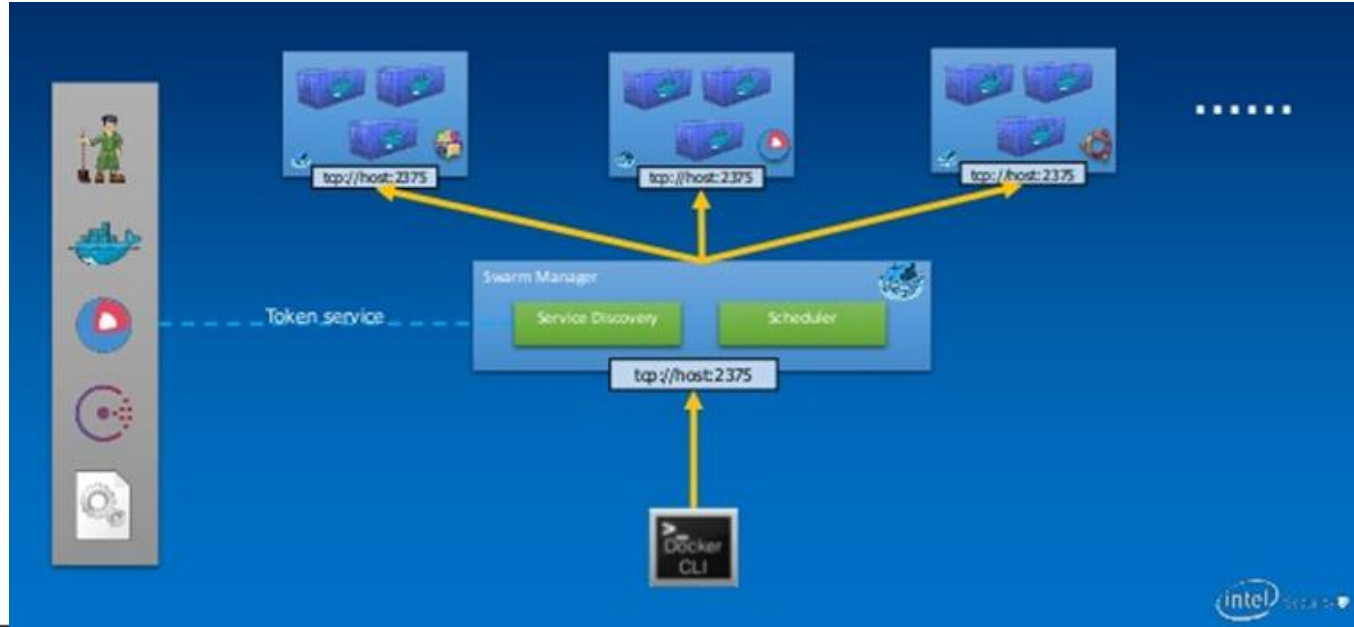
Docker Swarm

- Docker Swarm is clustering for Docker
- It turns several Docker hosts into a single virtual Docker host
- The regular Docker client works transparently with Swarm
- The Swarm is controlled by a Swarm Manager
- Each Docker node communicates with the manager
- It can be installed manually or by using Docker Machine



Swarm Architecture

- The Swarm is controlled by a Swarm Manager
- Each Docker node communicates with the manager
- It can be installed manually or by using Docker Machine



Creating a Swarm

- A machine needs to be designated as a manager
- There can be several managers
- The manager is created by initializing a Swarm
- Swarm must be performed on the manager machine

```
$ docker swarm init
Swarm initialized: current node (5lo6zmzvashepfm8ipnl37) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join \
  --token SWMTKN-1-5licto7l3hfkym98e2cq0rflh6a6hkyqzpxyb8jaid3qzx5kmm-
864t0x9ebw4a2ymsms1kvq9s9 \
  192.168.0.38:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Managing a Swarm

- The manager node is automatically added to the Swarm
- Information about the Swarm can be found using docker info

```
...  
Swarm: active  
  NodeID: 5lo6zmzvashexpfm8ipnl37  
  Is Manager: true  
  ClusterID: e9ff1sv78oxyb1989xa0hvy77  
  Managers: 1  
  Nodes: 1  
...  
  Node Address: 192.168.0.38  
...
```

Joining a Swarm

- Nodes can be added to the Swarm
- Ensure that firewall rules aren't blocking port 2377 on the manager
- The nodes can be listed

```
$ docker swarm join \
  --token SWMTKN-1-51ict07l3hfkym98e2cq0rflh6a6hkyqzpxyb8jaid3qzx5kmm-
864t0x9ebw4a2ymsms1kvq9s9 \
  192.168.0.38:2377
```

```
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
5lo6zmzvashepfm8ipnlui37 *	hp	Ready	Active	Leader
esc6tvcf01tx09zwzzr90no53	localhost.localdomain	Ready	Active	

- A Docker Service is a container
- Services run in Docker Swarm
- They can only be started on a Swarm Manager node
- Multiple copies of services can be run
- The Swarm Manager replicates the service on other nodes in the Swarm

Docker Swarm - Services (Contd.)

- A service can be added to the manager node
- A service is a container
- You can also inspect the service
- It also appears as a running container

```
$ docker service create --replicas 1 --name helloworld alpine  
ping docker.com
```

```
1rw0x7pohbf4mvz9isdiecb2
```

```
$ docker service ls
```

ID	NAME	REPLICAS	IMAGE	COMMAND
1rw0x7pohbf4	helloworld	0/1	alpine	ping docker.com

```
$ docker service inspect -pretty helloworld
```

```
$ docker ps
```


Docker Swarm Services - Scaling

- A service can be scaled
- The service will be duplicated and run on different nodes
- The service can be removed from all nodes

```
$ docker service ps helloworld
$ docker service scale helloworld=2
helloworld scaled to 2
```

```
$ docker service ps helloworld
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
84lhnfmfeob8y4r1tevkvs0d9	helloworld.1	alpine	hp	Running	Running
34u5pzw5yldnessr4radaip26	helloworld.2	alpine	localhost.localdomain	Running	Running

```
$ docker service rm helloworld
```

Applying Rolling Updates

- We will deploy a service based on the Redis 3.0.6 container image
- Then we will upgrade the service to use the Redis 3.0.7 container image using rolling updates
- We configure the rolling update policy at service deployment time
- The `--update-delay` flag configures the time delay between updates to a service task or sets of tasks
- By default the scheduler updates 1 task at a time
- By passing the `--update-parallelism` flag to configure the maximum number of service tasks that the scheduler updates simultaneously.

```
$ docker service create --replicas 3 --name redis --update-delay 10s redis:3.0.6
$ docker service inspect --pretty redis
```

```
$ docker service update --image redis:3.0.7 redis
$ docker service inspect --pretty redis
$ docker service update redis
$ docker service ps redis
```

Docker Swarm Backup

- Select a manager node to do the operation. Try not to choose the leader one in order to avoid a new election inside the

```
docker node ls -f "role=manager" | tail -n+2 | grep -vi leader
```

- ```
ENGINE=$(docker version -f '{{.Server.Version}}')
```

- ```
systemctl stop docker
```

```
tar cvzf "/tmp/swarm-${ENGINE}-${hostname -s}-${date +%s%z}.tgz" /var/lib/docker/swarm/
```

```
systemctl start docker
```

Docker Swarm Recovery

- You have to restore the backup on the same Docker Engine version
- You must use the same IP as the node you made the backup with, this is mandatory
- The command to force the new cluster does not reset the IP in the Swarm data
- Shutdown the Docker Engine on the node where you choose to restore:

```
systemctl stop docker
```

- Remove the contents of the swarm folder /var/lib/docker/swarm
- Restore the swarm folder /var/lib/docker/swarm with the content of the backup
- Restart the new node:

```
systemctl start docker
```

- Re-initialize the Swarm to isolate this node into his own new cluster. Avoid attempting to connect it to nodes that were part of the old swarm, and presumably no longer exist

```
docker swarm init --force-new-cluster
```

- Add the manager and worker nodes to bring your new swarm up to operating capacity

Deploying a Stack to Swarm

- When running Docker Engine in swarm mode, you can use `docker stack deploy` to deploy a complete application stack to the swarm
- The `deploy` command accepts a stack description in the form of a Compose file
- You need:
 - A Docker Engine of version 1.13.0 or later, running in swarm mode
 - Docker Compose version 1.10 or later

Exercise 1 : Docker Compose & Deploying Application Stack on Docker Swarm

- Use lab document



THANK YOU