Docker Orchestration & Container Storage

- starts at 9:05 pm

AGENDA

- Locking a Swarm Cluster
- Node Labels
- Docker Prune
- Docker Stacks
- AutoScaling vs Autohealing
- Docker in Docker

Locking Swarm Cluster

What we can't do when cluster is locked?

→ Add or remove any nodes from swarm
→ can't update swarm configurations
→ Manage services, tasks or replicas.

What all things can be done.

→ inspect the logs
→ check disk usage.

Locking the swarm cluster

```
docker swarm update --autolock=true
```

Save the unlock key.

Check the lock status

```
docker info | grep -i autolock
```

systemctl restart docker → to enter the locked state

to unlock

docker swarm unlock

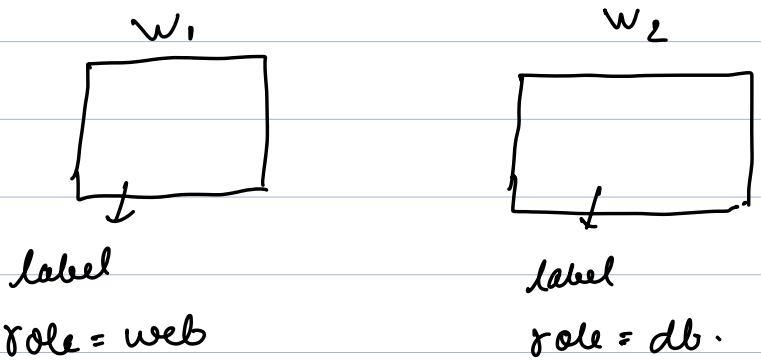
→ enter the key.

```
docker swarm unlock-key
```
→ to print the unlock key (only if the manager is not locked)

→ to revert locking { disabling auto-lock }

```
docker swarm update --autolock=false
```

→ Node Labels



- service.

→ Placement Constraints

-- constraint = node.label.role == web

→ nginx C1 C2 → W1

→ redis

role = db

→ C1 → W2

Label the nodes

docker node update --label-add role=db mvwipfqlgzys9zfawghbfynuy

docker node update --label-add role=web mr4gcd1aud8doy843b7fuyw4m

see the label

→ docker node inspect <node-id> | grep -C 5 -i label

Creating the service

```
docker service create \
  --name nginx-web \
  --replicas 2 \
  --constraint 'node.labels.role == web' \
  nginx:latest
```

role = web

```
docker service create \
  --name redis-db \
  --replicas 1 \
  --constraint 'node.labels.role == db' \
  redis:latest
```

role = db

multiple label service (multiple constraints)

```
docker service create \
  --name app-service \
  --replicas 3 \
```

```
--constraint 'node.labels.role == app' \

--constraint 'node.labels.env == production' \

my-app-image:latest
```

Docker Prune

→ Containers (executed)

→ dangling images

→ unused networks

→ build cache.

docker system prune.

docker container prune

docker volume prune.

docker network prune

docker image prune.

→ Dockerfile.

docker build -t myapp1 .

→ changed your dockerfile (update)

docker build -t myapp1 .

→ docker image prune --all

Important Considerations.

1. **Data Loss**:

- Pruning can result in the loss of data. For example, if you prune volumes that are not used by any containers, you may lose important persistent data stored in those volumes.

2. **Containers**:

- If you have stopped containers that you may want to restart later, avoid pruning them without ensuring they are not needed anymore.

3. **Images**:

- If you prune images, make sure you don't need those images to create new containers. For instance, if an image is

not being used by any containers, it may still be required later, but pruning it will remove it from your system.

4. **Cache and Build Artifacts**:

- Using `docker system prune` removes unused build cache and layers, which can be helpful in freeing up space, but it may also slow down subsequent builds as layers need to be rebuilt.

checking disk usage on docker

docker system df

Break 10:30 pm

Docker Stacks

Collection of services that make-up an application

docker-compose.yml file

deploying a stack

docker stack deploy -c docker-compose.yml my_stack

managing a stack

→ listing

docker stack ls

→ listing services in a stack

docker stack services my_stack

→ listing tasks in a stack

docker stack ps my_stack

updating the stack

→ make changes to docker-compose.yml

docker stack deploy -c docker-compose.yml my_stack

Restart Policy.

→ deploy section.

deploy
restart_policy:
condition
delay
max_attempts
window.

4 things.

① condition.

→ none

→ on-failure (only if they exit with

→ any. (default) non-zero code)

② delay. Time to wait before container is restarted

default → 0s

delay : 5s

③ Max_attempts

max tries to restart a container.

After this number docker won't restart the container.

④ window.

window within which restart attempts are counted for max_attempts.

restart-policy → on-failure

c1
c2 } (service)
web.

c1 → stopped.

C_1 → won't be restarted as it gracefully exited (0)

C_2 → Service.

docker-compose.yml.

replicas → from 2 to 3

C_1 → stopped

C_2

C_3

docker stop --signal=SIGABRT <container_id> (Exit code 137)

↘ restart policy will bring up new container

Check services and placement

docker stack services demo_stack

docker service ps demo_stack_web

docker service ps demo_stack_redis

docker stack rm my_stack → cleanup.

Auto Scaling & Auto healing.

AutoScaling.

Docker compose → no autoscaling.

Docker swarm → NO.

Kubernetes → (HPA) Yes.

Auto healing.

Docker-compose → No.

Docker-Swarm → Yes.

Kubernetes → Yes.

Feature	Docker Compose	Docker Swarm	Kubernetes
Auto Scaling	No native support (manual scaling of replicas)	No native auto-scaling (manual scaling)	Supports Horizontal Pod Autoscaler (HPA)
Auto Healing	No native auto-healing	Automatic container restart upon failure	Automatic pod replacement using ReplicaSets and Deployments
Health Checks	Can define health checks in `docker-compose.yml`	Built-in health checks for tasks	Liveness and Readiness Probes for pods

Docker Compose	Docker Stacks
Primarily used for local development.	Designed for production-grade deployments in a Docker Swarm cluster.
No built-in orchestration; containers are run on a single host.	Manages and deploys distributed applications across multiple nodes in a Swarm.
Controlled using the `docker-compose` CLI tool.	Controlled using the `docker stack` CLI commands.
Commands like `up`, `down`, `start`, `stop` for simple service lifecycle management.	Commands like `deploy`, `rm`, `ls`, `services` to deploy and manage stacks.
Scaling is manual using `docker-compose up --scale`.	Scaling is automatic based on Swarm's orchestration, or can be defined in the YAML.