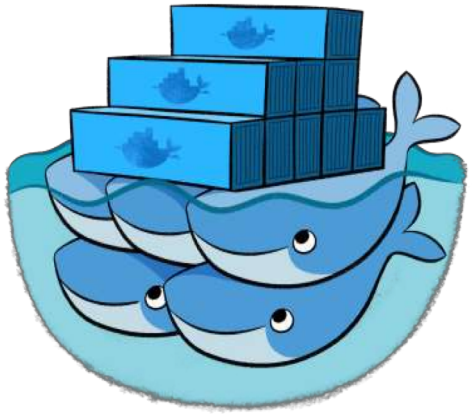
# Docker Swarm

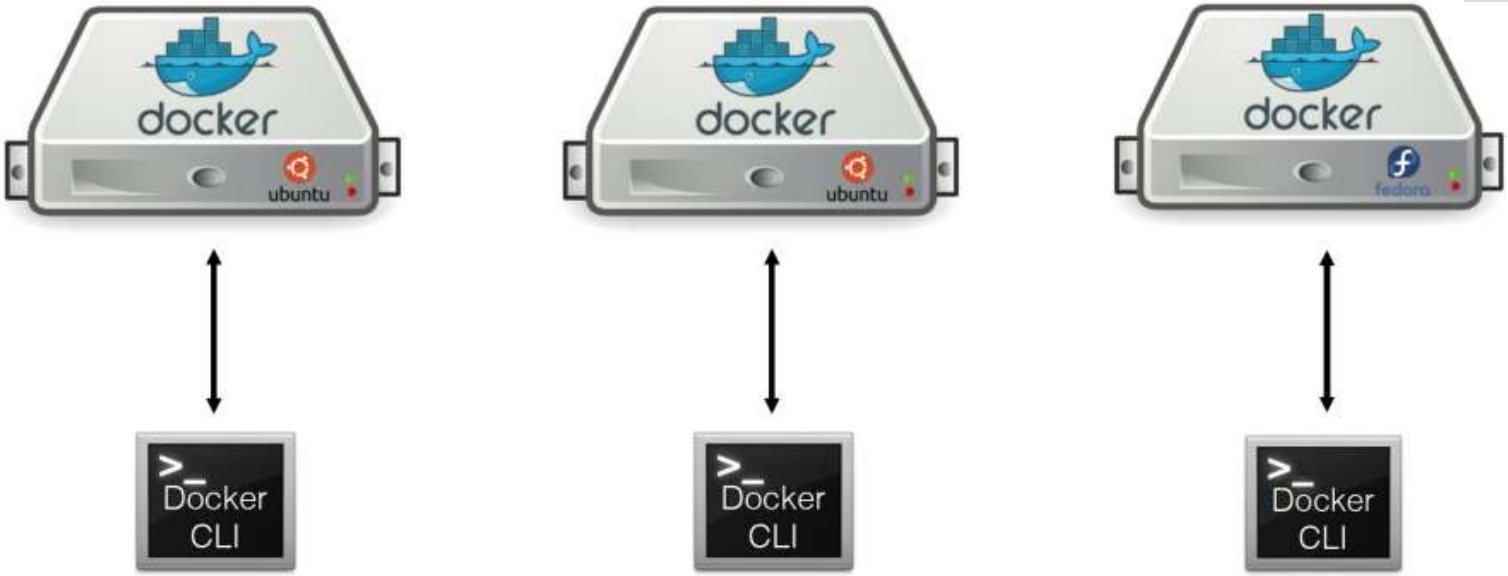



Mithun Technologies
devopstrainingblr@gmail.com
+91-9980923226

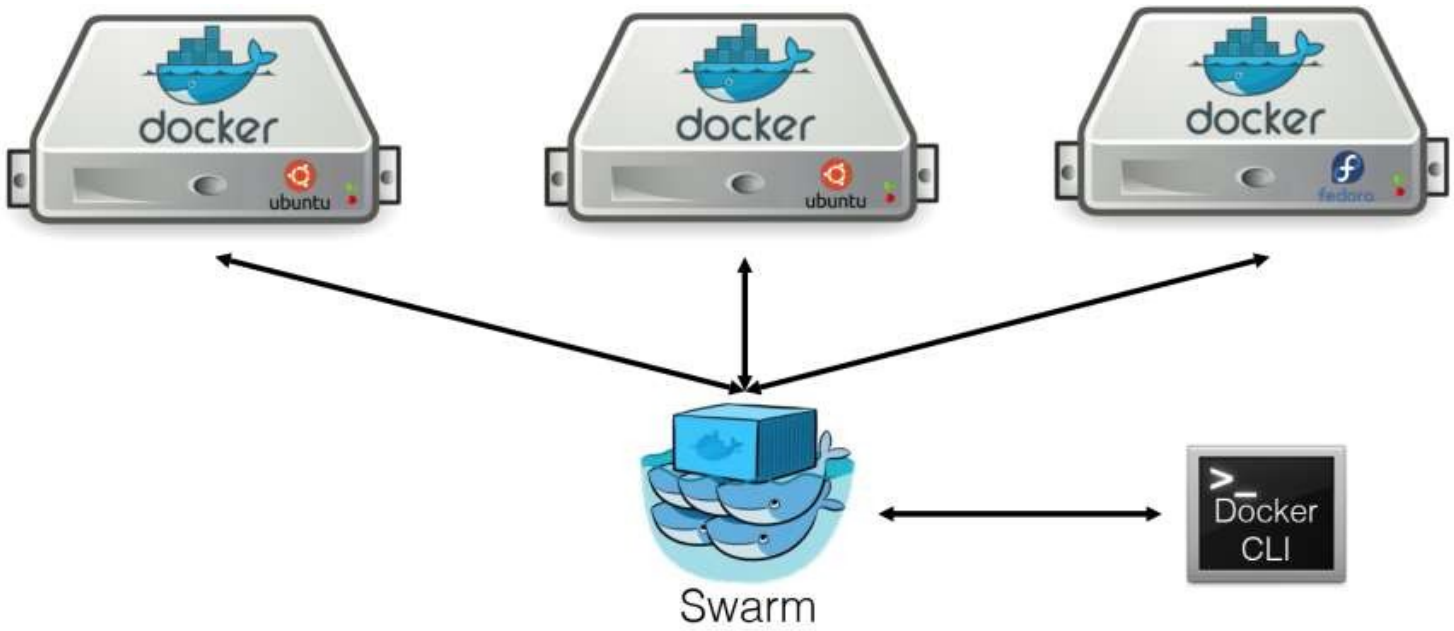
# Roadmap

- Key concepts
- Initializing a cluster of Docker Engines in swarm mode
- Adding nodes to the swarm
- Deploying services to the swarm
- Rolling updates
- Draining Nodes

# Traditional Model

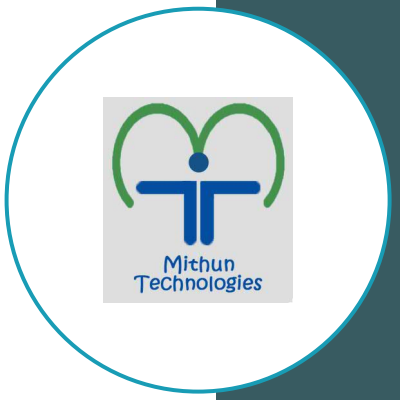


# Swarm


docker
ubuntu
docker
ubuntu
docker
fedora
Swarm
Docker
CLI

# What?

- Cluster management and orchestration feature embedded in Docker engine
- Cluster of Docker engines or nodes.
- One manager, and rest workers



# Swarm Features

- Cluster Management
- Decentralize design
- Declarative service model
- Scaling
- Desired state reconciliation
- Multi-host network
- Service discovery
- Load balancing
- Secure by default
- Rolling updates

# Pre-requisites

- As part of this session Docker Swarm will be explained in AWS systems.

- 1) AWS Account
- 2) Create three different ubuntu machines using **in AWS.**
- 3) Install Docker in all the 3 machines using below
- command.
- curl -fsSL get.docksal.io | bash
-

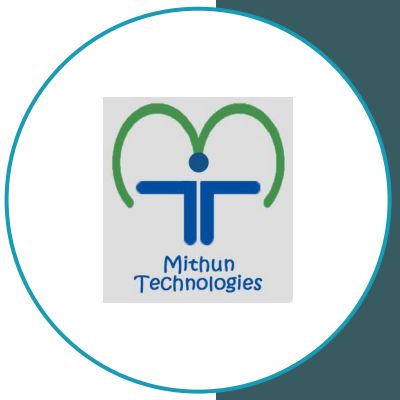
# Additional Network Checks

- The following ports must be available. On some systems, these ports are open by default.
  - **TCP port 2377** for cluster management communications
  - **TCP** and **UDP port 7946** for communication among nodes
  - **UDP port 4789** for overlay network traffic
- If you are planning on creating an overlay network with encryption (--opt encrypted), you will also need to ensure **ip protocol 50** (**ESP**) traffic is allowed.
- Enable all the ports in AWS security groups.

# Create a Swarm Manager

- Make sure the Docker Engine daemon is started on
- the hostmachines.
  - Open a terminal and ssh into the machine where you want to run your manager node.

ssh -i “mithuntechnologies.pem” ubuntu@ec2-13-232-190.ap-south-1.compute.amazon.com



# Create a Swarm Manager


1) Run the following command to create a new swarm

docker swarm init --listen-addr=eth0 --advertise-addr $(curl http://169.254.169.254/latest/meta-data/public-ipv4)

```
ubuntu@ip-172-31-11-139:~$ docker swarm init  --listen-addr=eth0 --advertise-addr $(curl http://169.254.169.254/latest/
meta-data/public-ipv4)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    14  100    14    0     0   3500      0 --:--:-- --:--:-- --:--:--  3500
Swarm initialized: current node (n0w1vfwd8o10ijf3octbde7j9) is now a manager.

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

    docker swarm join --token SWMTKN-1-150nke1tn9dy4b8rv0uilw479xczkr240v4l6l0pk0dsdfvz8o-ditefrekljiqsk29095yqk4sf 13.
232.238.190:2377

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

ubuntu@ip-172-31-11-139:~$
```

# Create a Swarm Manager


2) Run the docker node ls command to view information about nodes:

```
ubuntu@ip-172-31-11-139:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VE
RSION
n0w1vfwd8o10ijf3octbde7j9 *   ip-172-31-11-139    Ready               Active              Leader              18.03.1-c
e
ubuntu@ip-172-31-11-139:~$
```

The * next to the node ID indicates that you're currently connected on this node.

Docker Engine swarm mode automatically names the node for the machine host name.

# Add Nodes to Swarm

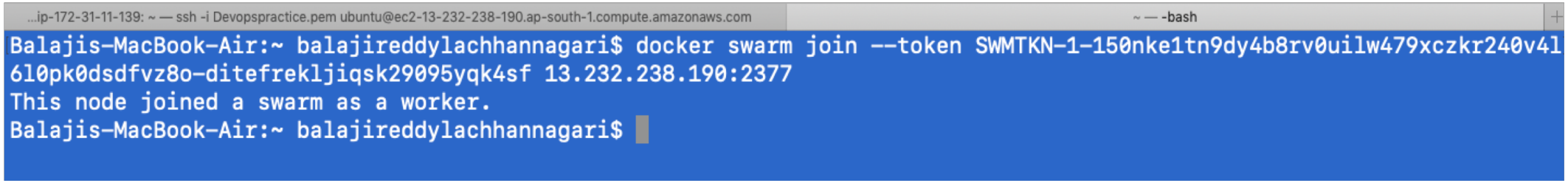
Make sure the Docker Engine daemon is started on the hostmachines.

1. Open a terminal and ssh into the machine where you want to add node (worker1). If you use Docker Machine, you can connect to it via SSHusing the following command:

```
ssh -i "mithuntechnologies.pem" ubuntu@ec2-13-232-190.ap-south-1.compute.amazon.com
```

# Add Nodes to Swarm

2. Run the command produced by the docker swarm init output from the create a swarm step 2 to create a worker node joined to the existing swarm:




Mithun Technologies

# Add Nodes to Swarm

3. If you don't have the command available, you can run the following command on a manager node to retrieve the join command for a worker:

```
...ip-172-31-11-139: ~ — ssh -i Devopspractice.pem ubuntu@ec2-13-232-238-190.ap-south-1.compute.amazonaws.com    ~ — -bash
ubuntu@ip-172-31-11-139:~$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-150nke1tn9dy4b8rv0uilw479xczkr240v4l6l0pk0dsdfvz8o-ditefrekljiqsk29095yqk4sf 13.
232.238.190:2377
```

# Add Nodes to Swarm

4. Repeat Step 1, and Step2 if you want to add more nodes

5. Open a terminal and ssh into the machine where the manager node runs and run the docker node ls command to see the worker nodes:

```
ubuntu@ip-172-31-11-139:~$ docker node ls
ID                            HOSTNAME                STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
n0w1vfwd8o10ijf3octbde7j9 *   ip-172-31-11-139        Ready               Active              Leader              18.03.1-ce
tkk7u3t8d18v8yx6etttdzlz7     linuxkit-025000000001   Ready               Active                                  18.06.1-ce
ubuntu@ip-172-31-11-139:~$
```

The MANAGER column identifies the manager nodes in the swarm. The empty status in this column for worker1 and worker2 identifies them as worker nodes.

Swarm management commands like Docker node ls only work on manager nodes.

# Deploy a service to Swarm

1. Open a terminal and ssh into the machine where you run your manager node. For example, use a machine named manager1. (i.e. docker-machine env manager1)
2. Run the following command:

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

9uk4639qpg7npwf3fn2aasksr
```

3. Run Docker service ls to see the list of running services:

```
$ docker service ls

ID            NAME        SCALE  IMAGE   COMMAND
9uk4639qpg7n  helloworld  1/1    alpine  ping docker.com
```

# Inspect the service on the Swarm (1/2)

1. If you haven't already, open a terminal and ssh into the machine where you run your manager node. For example, use a machine named manager1.
2. Run docker service inspect --pretty <SERVICE-ID> to display the details about a service in an easily readable

```
$ docker service inspect --pretty helloworld

ID:             9uk4639qpg7npwf3fn2aasksr
Name:           helloworld
Service Mode:   REPLICATED
 Replicas:              1
Placement:
UpdateConfig:
 Parallelism:   1
ContainerSpec:
 Image:         alpine
 Args:  ping docker.com
Resources:
Endpoint Mode:  vip
```

# Inspect the service on the Swarm (2/2)

3. Run docker service ps helloworld to see which nodes are running the service:

```
$ docker service ps helloworld

NAME                                    IMAGE   NODE     DESIRED STATE  LAST STATE
helloworld.1.8p1vev3fq5zm0mi8g0as41w35  alpine  worker2  Running        Running 3 minutes
```

4. Run docker ps on the node where the task is running to see details about the container for the task.

```
$docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
   PORTS               NAMES
e609dde94e47        alpine:latest       "ping docker.com"   3 minutes ago       Up 3 minutes
                       helloworld.1.8p1vev3fq5zm0mi8g0as41w35
```

# Scale the service in the Swarm(1/2)

1. If you haven't already, open a terminal and ssh into the machine where you run your manager node. For example, the tutorial uses a machine named manager1.
2. Run the following command to change the desired scale of the service running in the swarm:

```
$ docker service scale helloworld=5

helloworld scaled to 5
```

3. Run docker service ps helloworld to see the updated task list:

```
$ docker service ps helloworld

NAME                                    IMAGE   NODE      DESIRED STATE  CURRENT STATE
helloworld.1.8p1vev3fq5zm0mi8g0as41w35  alpine  worker2   Running        Running 7 minutes
helloworld.2.c7a7tcdq5s0uk3qr88mf8xco6  alpine  worker1   Running        Running 24 seconds
helloworld.3.6crl09vdcalvtfehfh69ogfb1  alpine  worker1   Running        Running 24 seconds
helloworld.4.auky6trawmdlcne8ad8phb0f1  alpine  manager1  Running        Running 24 seconds
helloworld.5.ba19kca06l18zujfwxyc5lkyn  alpine  worker2   Running        Running 24 seconds
```

# Scale the service in the Swarm(2/2)

4. Run docker ps to see the containers running on the node where you're connected.

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
   PORTS               NAMES
528d68040f95        alpine:latest       "ping docker.com"   About a minute ago   Up About a minut
e                       helloworld.4.auky6trawmdlcne8ad8phb0f1
```

# Delete service running on swarm(1/2)

1. If you haven't already, open a terminal and ssh into the machine where you run your manager node. For example, use a machine named manager1.
2. Run docker service rm helloworld to remove the helloworld service.

```
$ docker service rm helloworld

helloworld
```

3. Run docker service inspect <SERVICE-ID> to verify that the swarm manager removed the service. The CLI returns a message that the service is not found:

```
$ docker service inspect helloworld
[]
Error: no such service: helloworld
```

# Delete service running on Swarm(2/2)

4. Even though the service no longer exists, the task containers take a few seconds to clean up. You can use docker ps to verify when they are gone.

```
$ docker ps

    CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
            PORTS               NAMES
    db1651f50347        alpine:latest       "ping docker.com"        44 minutes ago      Up 46 se
conds                           helloworld.5.9lkmos2beppihw95vdwxy1j3w
    43bf6e532a92        alpine:latest       "ping docker.com"        44 minutes ago      Up 46 se
conds                           helloworld.3.a71i8rp6fua79ad43ycocl4t2
    5a0fb65d8fa7        alpine:latest       "ping docker.com"        44 minutes ago      Up 45 se
conds                           helloworld.2.2jpgensh7d935qdc857pxulfr
    afb0ba67076f        alpine:latest       "ping docker.com"        44 minutes ago      Up 46 se
conds                           helloworld.4.1c47o7tluz7drve4vkm2m5olx
    688172d3bfaa        alpine:latest       "ping docker.com"        45 minutes ago      Up About
 a minute                       helloworld.1.74nbhb3fhud8jfrhigd7s29we

$ docker ps
   CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
      PORTS
```

# Rolling Update (1/6)

1. If you haven't already, open a terminal and ssh into the machine where you run your manager node. For example, use a machine named manager1.
2. Deploy Redis 3.0.6 to the swarm and configure the swarm with a 10 second update delay:

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

0u6a4s31ybk7yw2wyvtikmu50
```

# Rolling Update (2/6)

3. Inspect the redis service:

```
$ docker service inspect --pretty redis

ID:             0u6a4s31ybk7yw2wyvtikmu50
Name:           redis
Service Mode:   Replicated
 Replicas:      3
Placement:
 Strategy:          Spread
UpdateConfig:
 Parallelism:   1
 Delay:         10s
ContainerSpec:
 Image:         redis:3.0.6
Resources:
Endpoint Mode:  vip
```

# Rolling Update (3/6)

4. Now you can update the container image for redis. The swarm manager applies the update to nodes:

```
$ docker service update --image redis:3.0.7 redis
redis
```

The scheduler applies rolling updates as follows by default:

- Stop the first task.
- Schedule update for the stopped task.
- Start the container for the updated task.
- If the update to a task returns RUNNING, wait for the specified delay period then start the next task.
- If, at any time during the update, a task returns FAILED, pause the update.

# Rolling Update (4/6)

5. Run docker service inspect --pretty redis to see the new image in the desired state:

```
$ docker service inspect --pretty redis

ID:             0u6a4s31ybk7yw2wyvtikmu50
Name:           redis
Service Mode:   Replicated
 Replicas:      3
Placement:
 Strategy:          Spread
UpdateConfig:
 Parallelism:   1
 Delay:         10s
ContainerSpec:
 Image:         redis:3.0.7
Resources:
Endpoint Mode:  vip
```

# Rolling Update (5/6)

6. The output of service inspect shows if your update paused due to failure:

```
$ docker service inspect --pretty redis

ID:             0u6a4s31ybk7yw2wyvtikmu50
Name:           redis
...snip...
Update status:
 State:      paused
 Started:    11 seconds ago
 Message:    update paused due to failure or early termination of task 9p7ith557h8ndf0ui9s0q951b
...snip...
```

7. To restart a paused update run docker service update <SERVICE-ID>. For example:

```
docker service update redis
```

# Rolling Update (6/6)


8. Run docker service ps <SERVICE-ID> to watch the rolling update:

```
$ docker service ps redis

NAME                                    IMAGE        NODE       DESIRED STATE  CURRENT STATE
      ERROR
redis.1.dos1zffgeofhagnve8w864fco       redis:3.0.7  worker1    Running        Running 37 seconds
 \_ redis.1.88rdo6pa52ki8oqx6dogf04fh   redis:3.0.6  worker2    Shutdown       Shutdown 56 seconds
 ago
redis.2.9l3i4j85517skba5o7tn5m8g0       redis:3.0.7  worker2    Running        Running About a min
ute
 \_ redis.2.66k185wilg8ele7ntu8f6nj6i   redis:3.0.6  worker1    Shutdown       Shutdown 2 minutes
 ago
redis.3.egiuiqpzrdbxks3wxgn8qib1g       redis:3.0.7  worker1    Running        Running 48 seconds
 \_ redis.3.ctzktfddb2tepkr45qcmqln04   redis:3.0.6  mmanager1  Shutdown       Shutdown 2 minutes
 ago
```

# Drain a Node on the Swarm(1/5)

1. If you haven't already, open a terminal and ssh into the machine where you run your manager node. For example, use a machine named manager1.
2. Verify that all your nodes are actively available.

```
$ docker node ls

ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
1bcef6utixb0l0ca7gxuivsj0    worker2   Ready   Active
38ciaotwjuritcdtn9npbnkuz    worker1   Ready   Active
e216jshn25ckzbvmwlnh5jr3g *  manager1  Ready   Active        Leader
```

3. Create a radis service with 3 replicas and update delay of 10 seconds

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

c5uo6kdmzpon37mgj9mwglcfw
```

# Drain a Node on the Swarm(2/5)

4. Run docker service ps redis to see how the swarm manager assigned the tasks to different nodes:

```
$ docker service ps redis

NAME                                    IMAGE        NODE     DESIRED STATE  CURRENT STATE
redis.1.7q92v0nr1hcgts2amcjyqg3pq  redis:3.0.6  manager1 Running        Running 26 seconds
redis.2.7h2l8h3q3wqy5f66hlv9ddmi6  redis:3.0.6  worker1  Running        Running 26 seconds
redis.3.9bg7cezvedmkgg6c8yzvbhwsd  redis:3.0.6  worker2  Running        Running 26 seconds
```

In this case the swarm manager distributed one task to each node. You may see the tasks distributed differently among the nodes in your environment.

5. Run docker node update --availability drain <NODE-ID> to drain a node that had a task assigned to it:

```
docker node update --availability drain worker1

worker1
```

Mithun Technologies

# Drain a Node on the Swarm (3/5)

6. Inspect the node to check its availability:

```
$ docker node inspect --pretty worker1

ID:                     38ciaotwjuritcdtn9npbnkuz
Hostname:               worker1
Status:
 State:                 Ready
 Availability:          Drain
...snip...
```

# Drain a Node on the Swarm (4/5)

7. Run docker service ps redis to see how the swarm manager updated the task assignments for the redis service:

```
$ docker service ps redis

NAME                                    IMAGE        NODE      DESIRED STATE  CURRENT STATE
     ERROR
redis.1.7q92v0nr1hcgts2amcjyqg3pq       redis:3.0.6  manager1  Running        Running 4 minutes
redis.2.b4hovzed7id8irg1to42egue8       redis:3.0.6  worker2   Running        Running About a min
ute
 \_ redis.2.7h2l8h3q3wqy5f66hlv9ddmi6   redis:3.0.6  worker1   Shutdown       Shutdown 2 minutes
 ago
redis.3.9bg7cezvedmkgg6c8yzvbhwsd       redis:3.0.6  worker2   Running        Running 4 minutes
```

# Drain a Node on the Swarm(5/5)

8. Run docker node update --availability active <NODE-ID> to return the drained node to an active state:

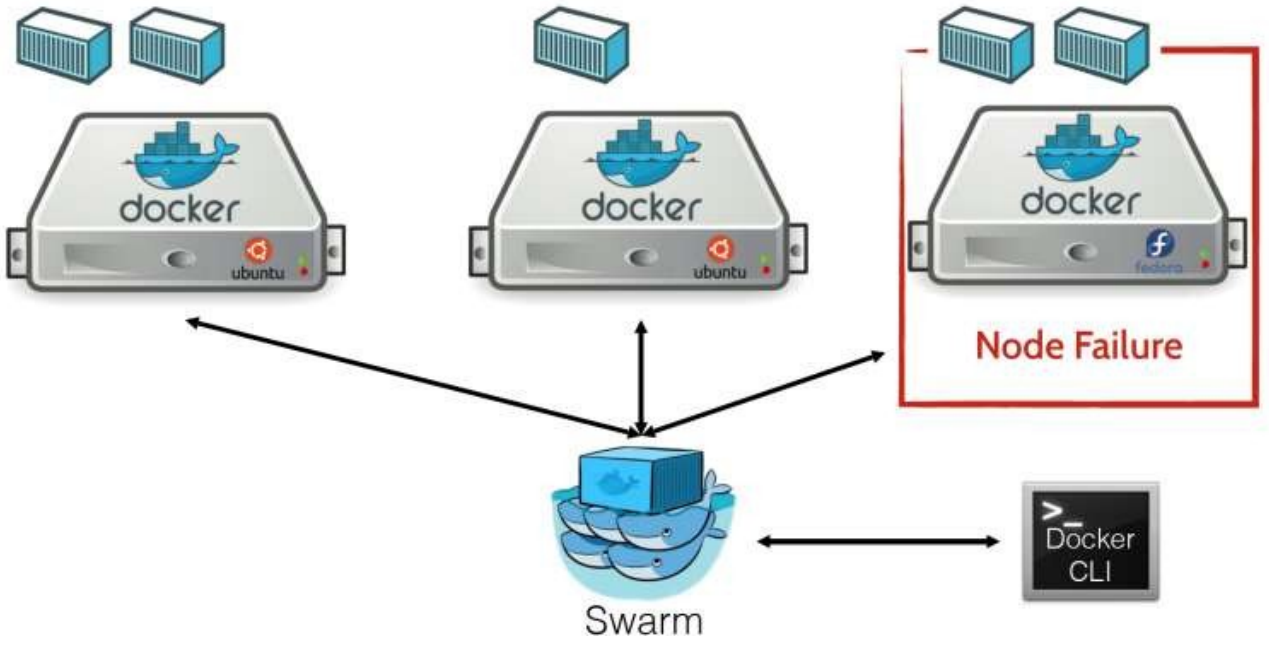
```
$ docker node update --availability active worker1

worker1
```

9. Inspect the node to see the updated state

When you set the node back to Active availability, it can receive new tasks:

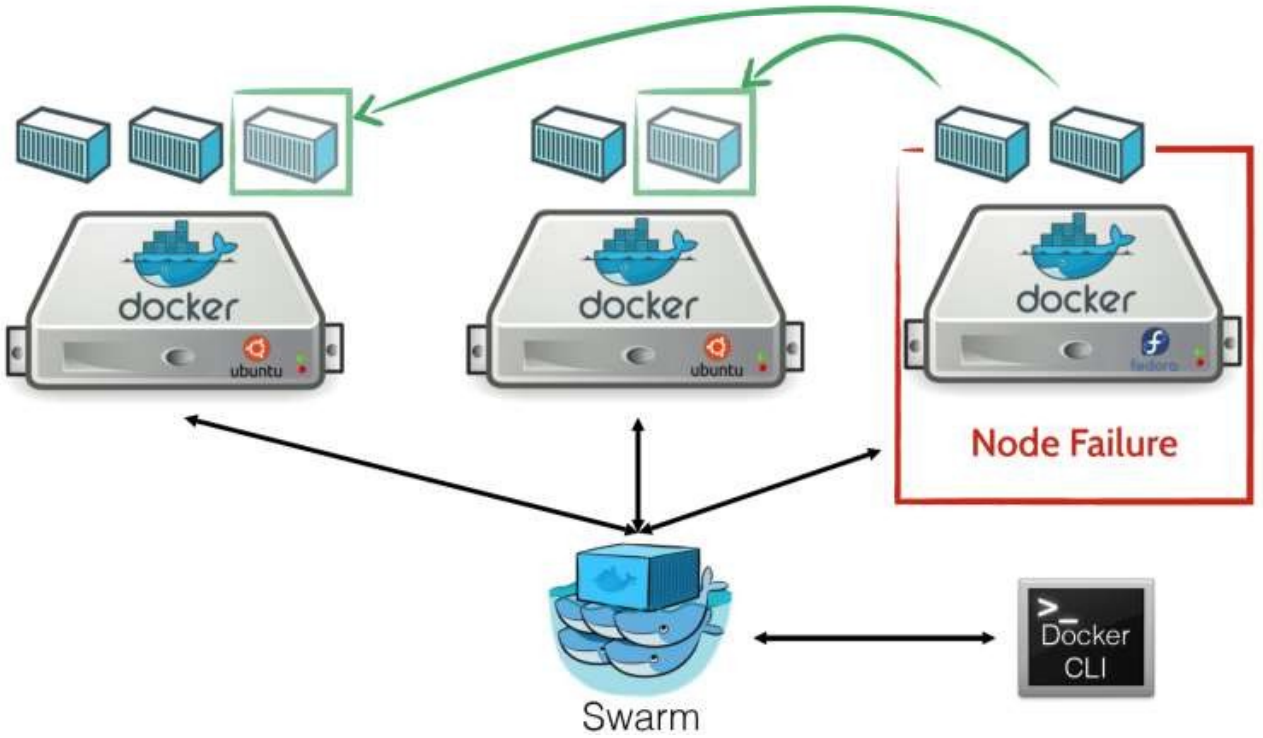
- during a service update to scale up
- during a rolling update
- when you set another node to Drain availability
- when a task fails on another active node

# On Failure (Node)



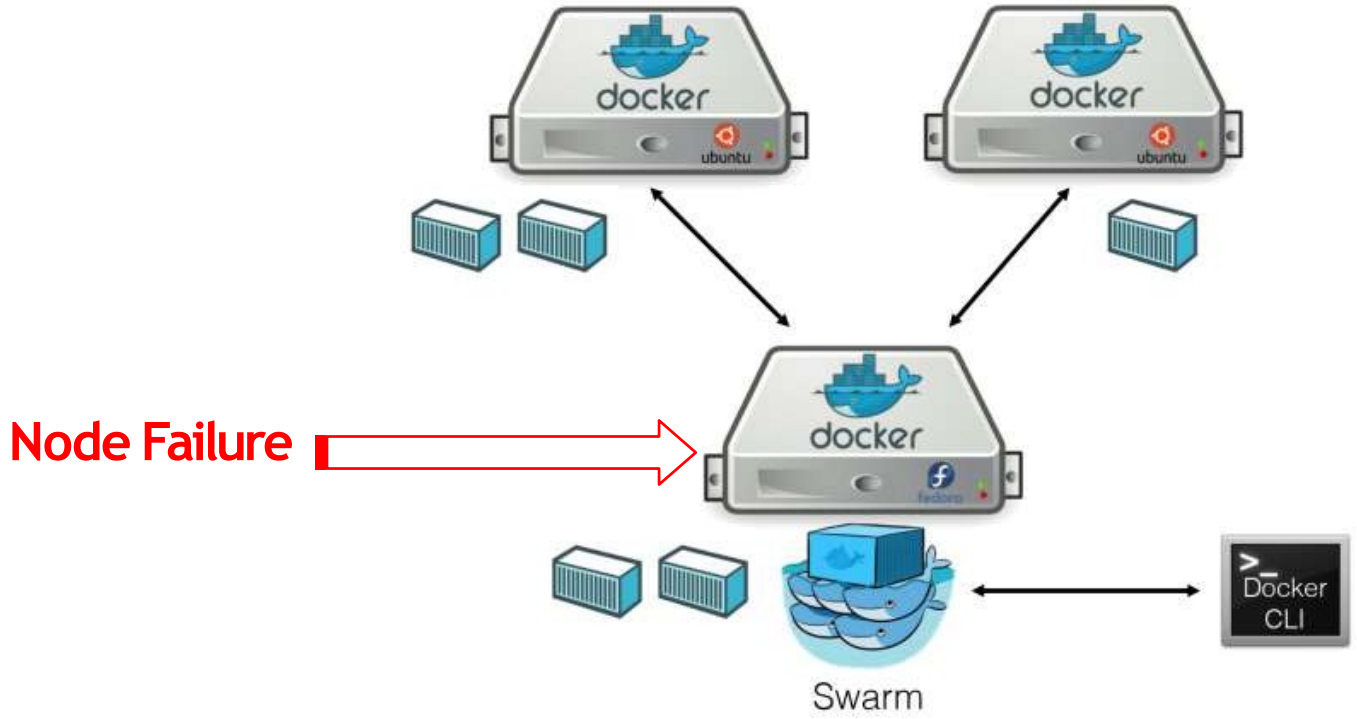

Mithun Technologies

# On Failure (Node) - Rescheduling



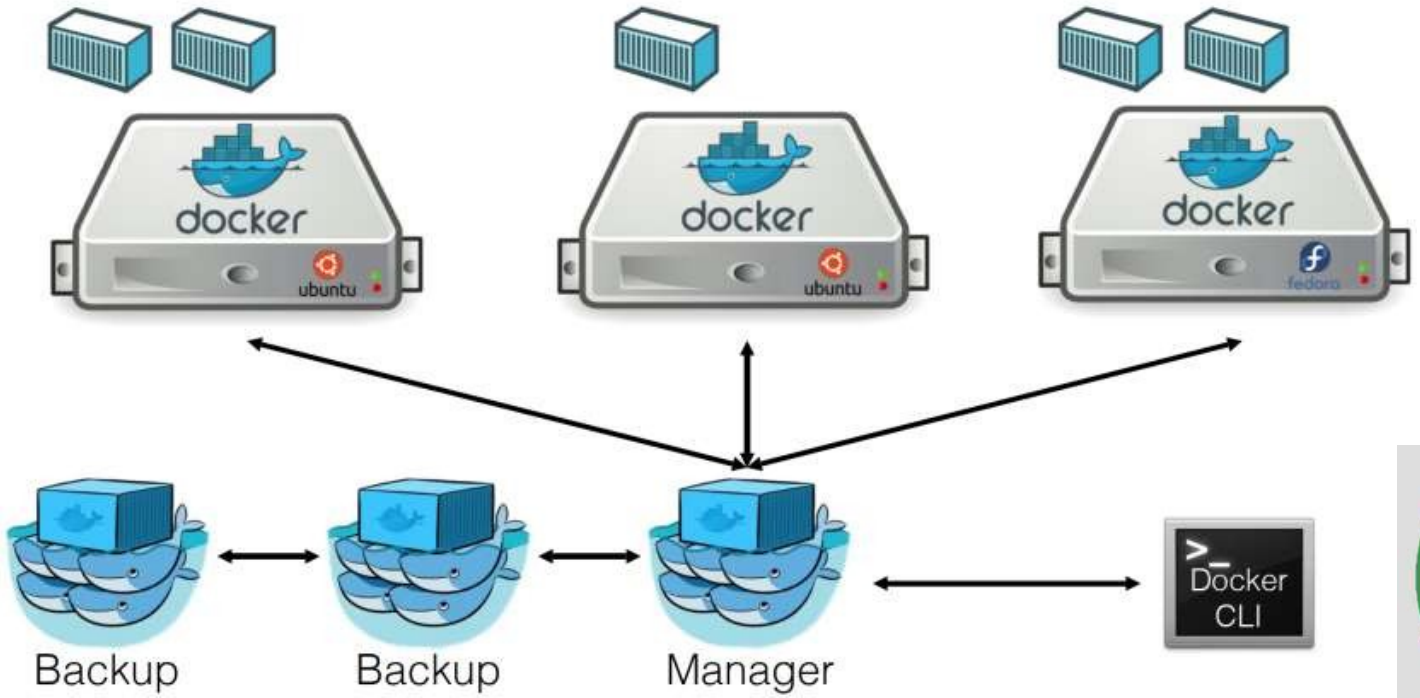

Mithun Technologies

# On Failure (Manager)



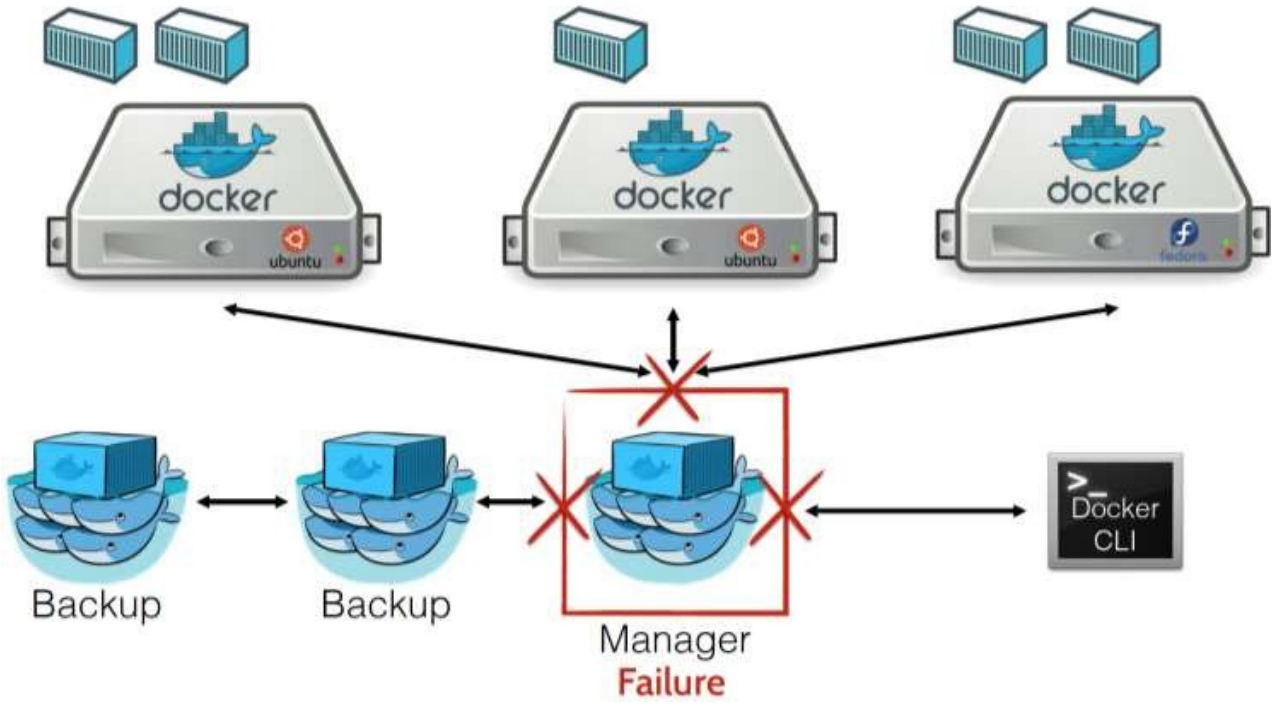

Mithun Technologies

# On Failure (Manager) – Backup Managers



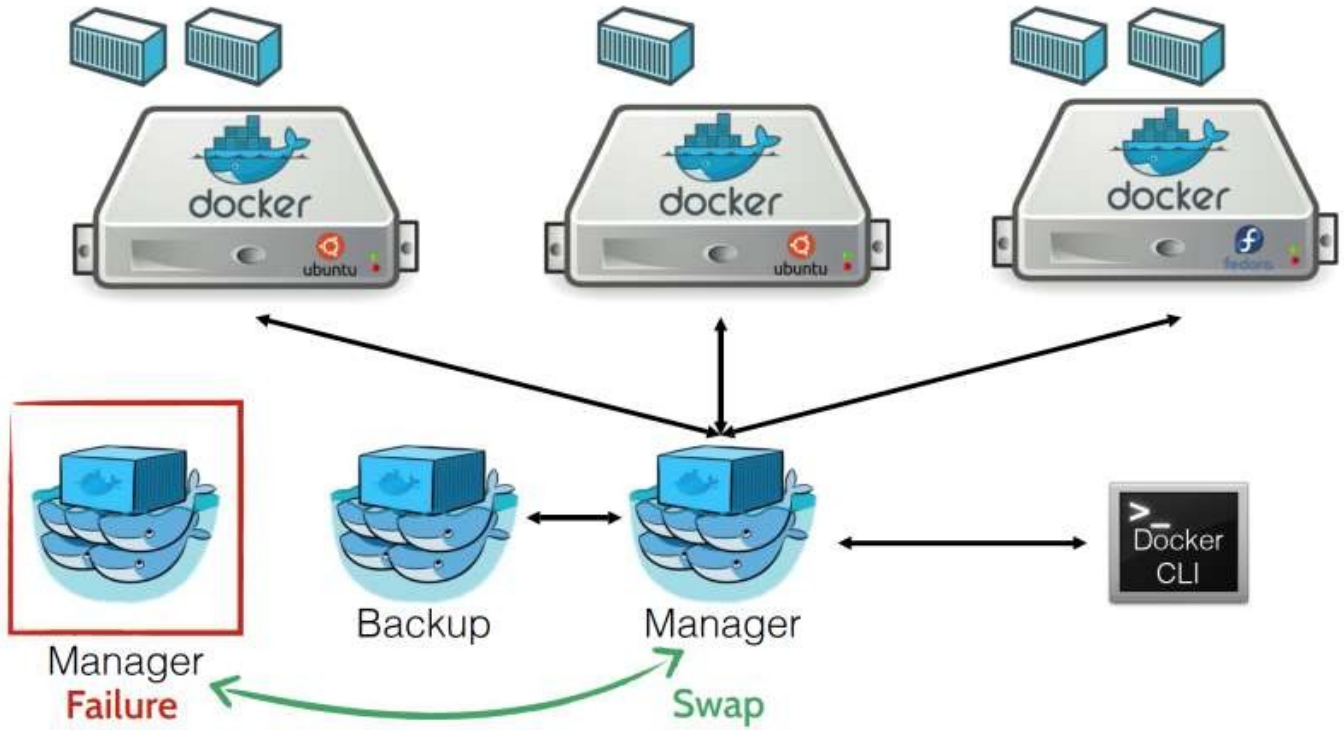

Mithun Technologies

# On Failure (Manager) - Failed Manager




Mithun Technologies

# On Failed (Manager) – Swap Manager




Mithun Technologies

Questions ?

Thank You

Mithun Technologies
+91-9980923226
devopstrainingblr@gmail.com