Docker Swarm

Docker Swarm is a container orchestration tool that allows you to manage a cluster of Docker nodes as a single logical system. It provides several benefits, such as scalability, high availability, load balancing, and simplified deployment. Here are some use cases and examples of how Docker Swarm can be utilized:

1. High Availability Web Application

Use Case: Deploying a web application that requires high availability and redundancy.

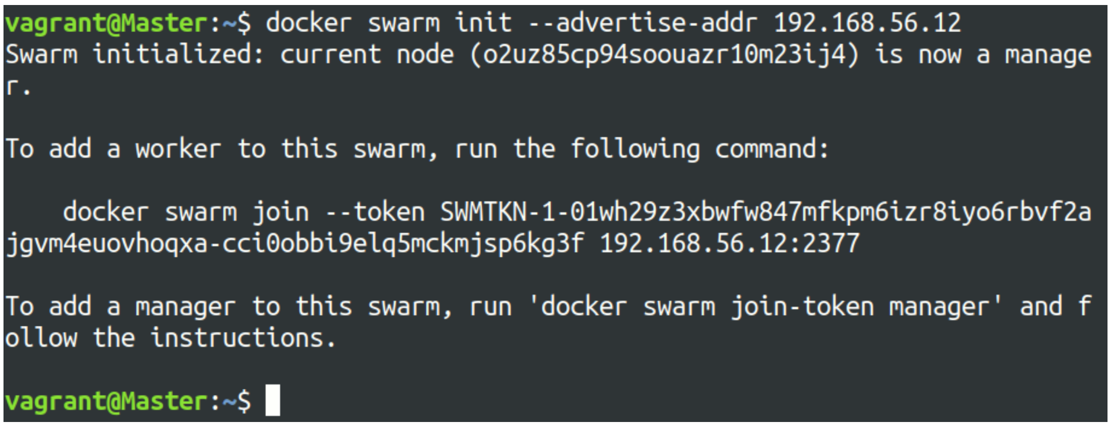
Example:

- Create a Swarm cluster with multiple manager and worker nodes.
- Deploy a replicated service for the web application.
- Docker Swarm ensures that if one node fails, another node takes over, maintaining the application's availability.

Steps:

Initialize Swarm:

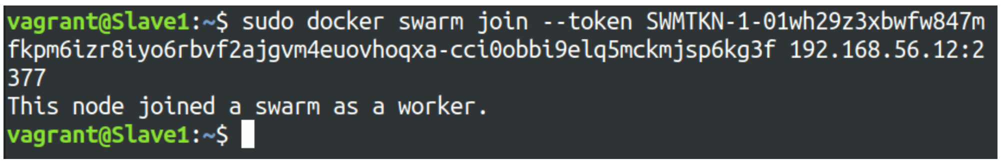
```
docker swarm init --advertise-addr <MANAGER-IP>
```



```
vagrant@Master:~$ docker swarm init --advertise-addr 192.168.56.12
Swarm initialized: current node (o2uz85cp94soouazr10m23ij4) is now a manage
r.

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

    docker swarm join --token SWMTKN-1-01wh29z3xbwfw847mfkpm6izr8iyo6rbvf2a
jgvm4euovhoqxa-cci0obbi9elq5mckmjsp6kg3f 192.168.56.12:2377

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

vagrant@Master:~$
```

Add Worker Nodes: On each worker node:

```
docker swarm join --token <WORKER-TOKEN> <MANAGER-IP>:2377
```



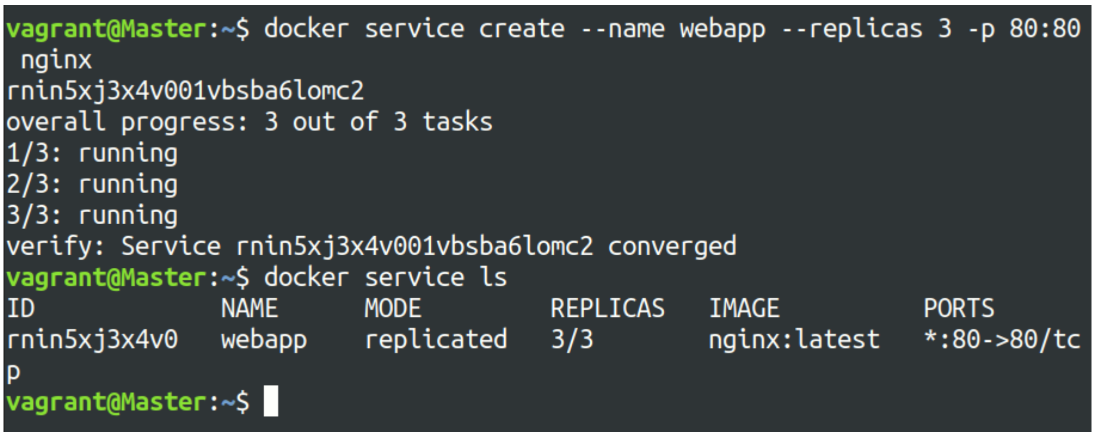
```
vagrant@Slave1:~$ sudo docker swarm join --token SWMTKN-1-01wh29z3xbwfw847m
fkpm6izr8iyo6rbvf2ajgvm4euovhoqxa-cci0obbi9elq5mckmjsp6kg3f 192.168.56.12:2
377
This node joined a swarm as a worker.
vagrant@Slave1:~$
```

Deploy a Web Application:

```
docker service create --name webapp --replicas 3 -p 80:80 nginx
```

Check Service Status:

```
docker service ls
```



```
vagrant@Master:~$ docker service create --name webapp --replicas 3 -p 80:80
 nginx
rnin5xj3x4v001vbsba6lomc2
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service rnin5xj3x4v001vbsba6lomc2 converged
vagrant@Master:~$ docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
rnin5xj3x4v0   webapp    replicated   3/3        nginx:latest   *:80->80/tc
p
vagrant@Master:~$ 
```

2. Continuous Integration/Continuous Deployment (CI/CD) Pipeline

Use Case: Automating the deployment of applications with a CI/CD pipeline.

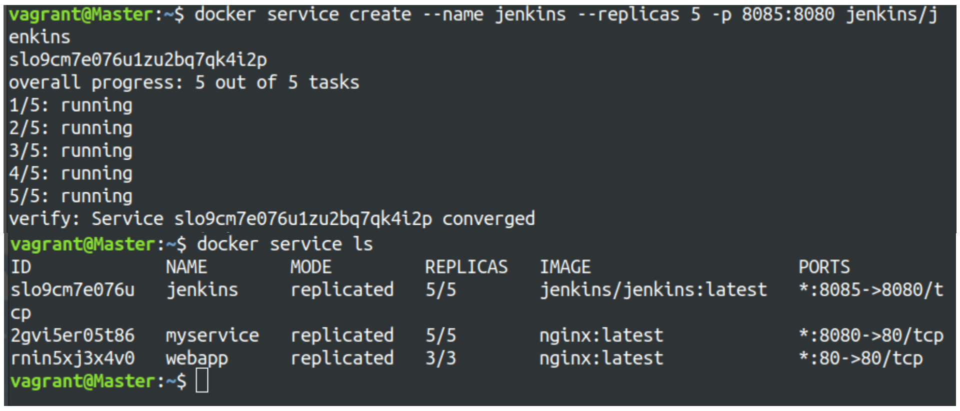
Example:

- Use Docker Swarm to deploy applications automatically when new code is committed.
- Integrate with CI/CD tools like Jenkins, GitLab CI, or GitHub Actions.

Steps:

Initialize Swarm and Deploy Jenkins:

```
docker service create --name jenkins --replicas 1 -p 8080:8080
jenkins/jenkins
```



```
vagrant@Master:~$ docker service create --name jenkins --replicas 5 -p 8085:8080 jenkins/j
enkins
slo9cm7e076u1zu2bq7qk4i2p
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service slo9cm7e076u1zu2bq7qk4i2p converged
vagrant@Master:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE                     PORTS
slo9cm7e076u   jenkins     replicated   5/5        jenkins/jenkins:latest    *:8085->8080/t
cp
2gvi5er05t86   myservice   replicated   5/5        nginx:latest              *:8080->80/tcp
rnin5xj3x4v0   webapp      replicated   3/3        nginx:latest              *:80->80/tcp
vagrant@Master:~$ 
```

1. **Configure Jenkins to Deploy to Swarm:**
   - Set up Jenkins with necessary plugins for Docker and Docker Swarm.
   - Create a Jenkins pipeline that builds Docker images and deploys them to the Swarm cluster.

2. **Automate Deployment:**
    - Configure Jenkins to trigger builds and deployments on code changes.

3. Load Balancing and Scaling Services

Use Case: Distributing traffic across multiple instances of a service for load balancing and scaling.

Example:

- Deploy a service with multiple replicas.
- Docker Swarm automatically load balances requests across these replicas.

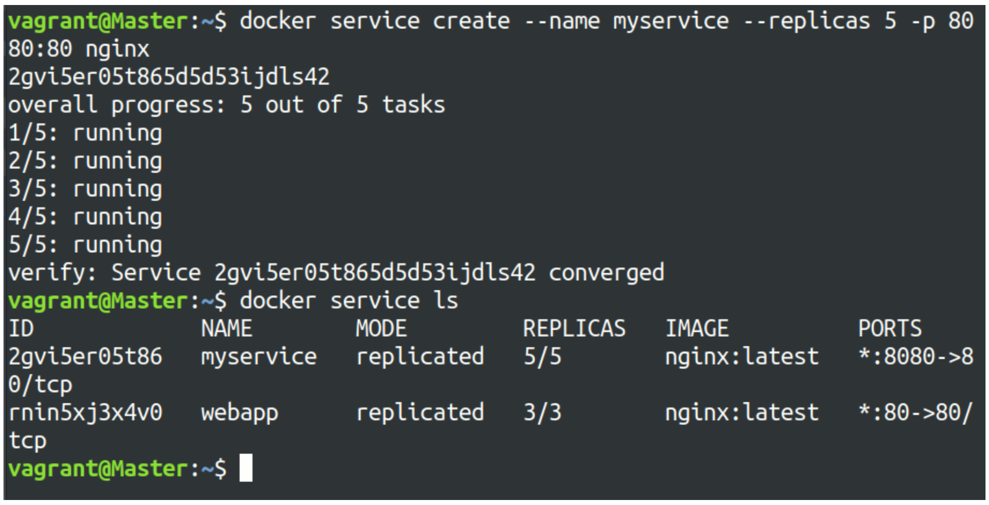
Steps:

Initialize Swarm:

```
docker swarm init
```

Deploy a Service with Load Balancing:

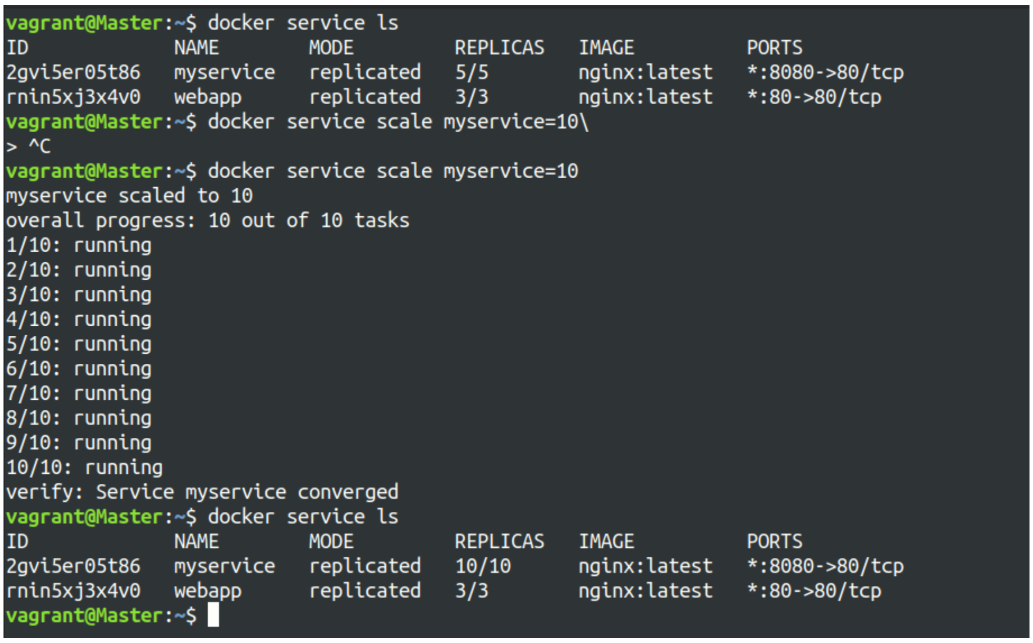
```
docker service create --name myservice --replicas 5 -p 8080:80 nginx
```



```
vagrant@Master:~$ docker service create --name myservice --replicas 5 -p 80
80:80 nginx
2gvi5er05t865d5d53ijdls42
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service 2gvi5er05t865d5d53ijdls42 converged
vagrant@Master:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE          PORTS
2gvi5er05t86   myservice   replicated   5/5        nginx:latest   *:8080->8
0/tcp
rnin5xj3x4v0   webapp      replicated   3/3        nginx:latest   *:80->80/
tcp
vagrant@Master:~$
```

Scale the Service:

```
docker service scale myservice=10
```



```
vagrant@Master:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE          PORTS
2gvi5er05t86   myservice   replicated   5/5        nginx:latest   *:8080->80/tcp
rnin5xj3x4v0   webapp      replicated   3/3        nginx:latest   *:80->80/tcp
vagrant@Master:~$ docker service scale myservice=10\
> ^C
vagrant@Master:~$ docker service scale myservice=10
myservice scaled to 10
overall progress: 10 out of 10 tasks
1/10: running
2/10: running
3/10: running
4/10: running
5/10: running
6/10: running
7/10: running
8/10: running
9/10: running
10/10: running
verify: Service myservice converged
vagrant@Master:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE          PORTS
2gvi5er05t86   myservice   replicated   10/10      nginx:latest   *:8080->80/tcp
rnin5xj3x4v0   webapp      replicated   3/3        nginx:latest   *:80->80/tcp
vagrant@Master:~$ 
```

4. Microservices Architecture

Use Case: Deploying a microservices-based application with multiple interdependent services.

Example:

- Use Docker Swarm to manage the deployment and scaling of each microservice.
- Ensure communication between services through the Swarm network.

Steps:

Initialize Swarm:

```
docker swarm init
```

Deploy Microservices:

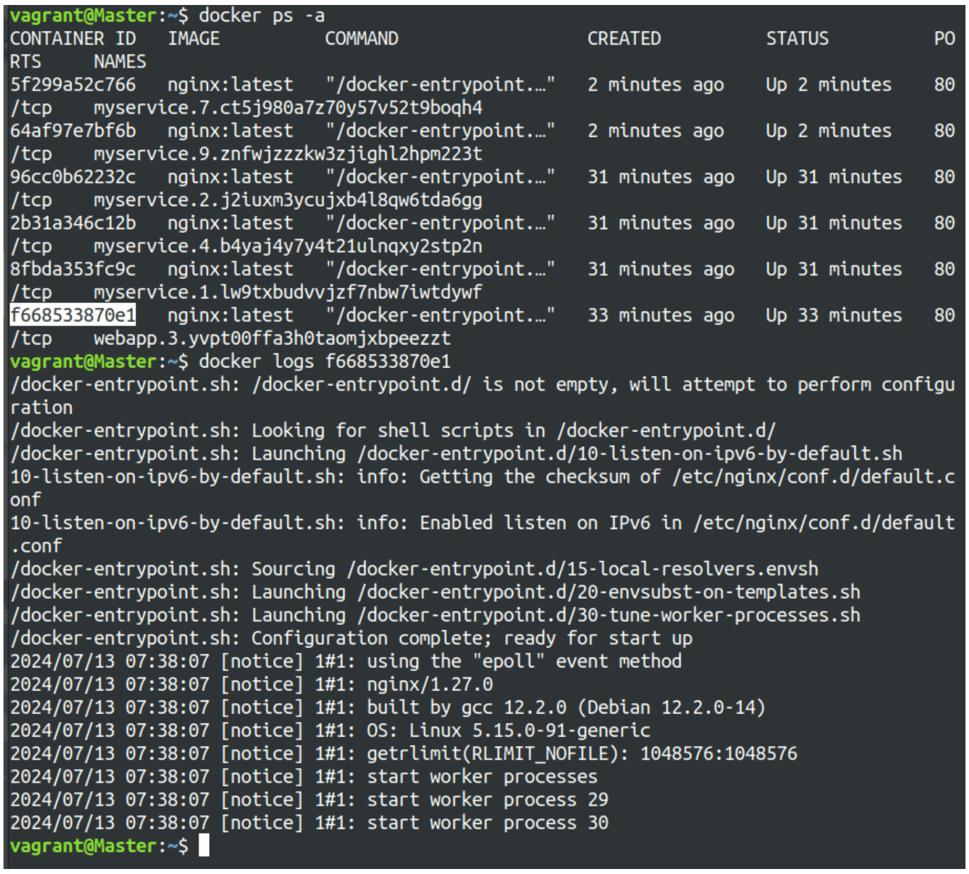
```
docker service create --name service1 --replicas 3 -p 5000:5000
my_microservice1

docker service create --name service2 --replicas 2 -p 5001:5001
my_microservice2
```

1. **Ensure Services Communicate:**
   - Use Docker Swarm's service discovery to enable communication between services using their service names.

Docker Logs

To view the logs of a container, you can use the following command:

```
docker logs <container_name_or_id>
```



```
vagrant@Master:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PO
RTS     NAMES
5f299a52c766   nginx:latest   "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    80
/tcp    myservice.7.ct5j980a7z70y57v52t9boqh4
64af97e7bf6b   nginx:latest   "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    80
/tcp    myservice.9.znfwjzzzkw3zjighl2hpm223t
96cc0b62232c   nginx:latest   "/docker-entrypoint.…"   31 minutes ago   Up 31 minutes   80
/tcp    myservice.2.j2iuxm3ycujxb4l8qw6tda6gg
2b31a346c12b   nginx:latest   "/docker-entrypoint.…"   31 minutes ago   Up 31 minutes   80
/tcp    myservice.4.b4yaj4y7y4t21ulnqxy2stp2n
8fbda353fc9c   nginx:latest   "/docker-entrypoint.…"   31 minutes ago   Up 31 minutes   80
/tcp    myservice.1.lw9txbudvvjzf7nbw7iwtdywf
f668533870e1   nginx:latest   "/docker-entrypoint.…"   33 minutes ago   Up 33 minutes   80
/tcp    webapp.3.yvpt00ffa3h0taomjxbpeezzt
vagrant@Master:~$ docker logs f668533870e1
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configu
ration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.c
onf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default
.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/07/13 07:38:07 [notice] 1#1: using the "epoll" event method
2024/07/13 07:38:07 [notice] 1#1: nginx/1.27.0
2024/07/13 07:38:07 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/07/13 07:38:07 [notice] 1#1: OS: Linux 5.15.0-91-generic
2024/07/13 07:38:07 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/07/13 07:38:07 [notice] 1#1: start worker processes
2024/07/13 07:38:07 [notice] 1#1: start worker process 29
2024/07/13 07:38:07 [notice] 1#1: start worker process 30
vagrant@Master:~$
```

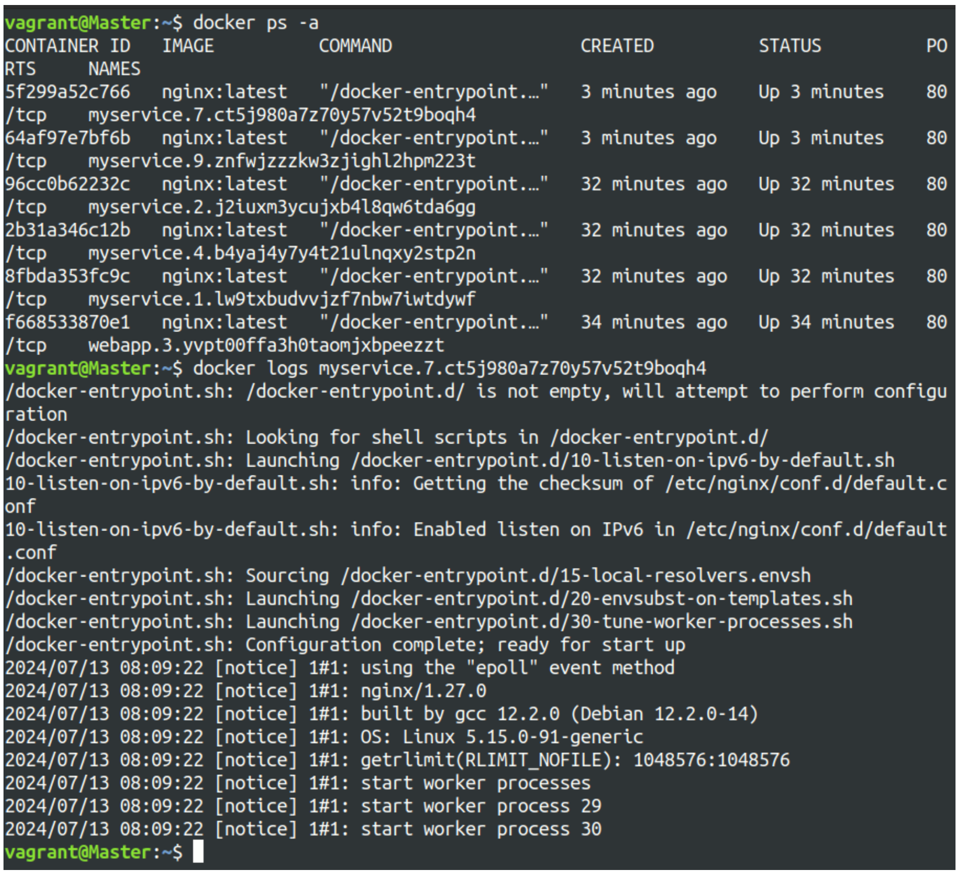
Options

Here are some useful options for the `docker logs` command:

- **-f, --follow**: Follow log output (similar to `tail -f`).
- **--tail**: Show only the last `N` lines of log output.
- **-t, --timestamps**: Show timestamps for each log entry.
- **--since**: Show logs since a specific time (e.g., `2022-07-01T13:23:37` or `10m` for last 10 minutes).
- **--until**: Show logs up until a specific time.

1. Viewing Logs of a Container

```
docker logs my_container
```



```
vagrant@Master:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PO
RTS      NAMES
5f299a52c766   nginx:latest   "/docker-entrypoint.…"   3 minutes ago    Up 3 minutes    80
/tcp     myservice.7.ct5j980a7z70y57v52t9boqh4
64af97e7bf6b   nginx:latest   "/docker-entrypoint.…"   3 minutes ago    Up 3 minutes    80
/tcp     myservice.9.znfwjzzzkw3zjighl2hpm223t
96cc0b62232c   nginx:latest   "/docker-entrypoint.…"   32 minutes ago   Up 32 minutes   80
/tcp     myservice.2.j2iuxm3ycujxb4l8qw6tda6gg
2b31a346c12b   nginx:latest   "/docker-entrypoint.…"   32 minutes ago   Up 32 minutes   80
/tcp     myservice.4.b4yaj4y7y4t21ulnqxy2stp2n
8fbda353fc9c   nginx:latest   "/docker-entrypoint.…"   32 minutes ago   Up 32 minutes   80
/tcp     myservice.1.lw9txbudvvjzf7nbw7iwtdywf
f668533870e1   nginx:latest   "/docker-entrypoint.…"   34 minutes ago   Up 34 minutes   80
/tcp     webapp.3.yvpt00ffa3h0taomjxbpeezzt
vagrant@Master:~$ docker logs myservice.7.ct5j980a7z70y57v52t9boqh4
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configu
ration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.c
onf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default
.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/07/13 08:09:22 [notice] 1#1: using the "epoll" event method
2024/07/13 08:09:22 [notice] 1#1: nginx/1.27.0
2024/07/13 08:09:22 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/07/13 08:09:22 [notice] 1#1: OS: Linux 5.15.0-91-generic
2024/07/13 08:09:22 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/07/13 08:09:22 [notice] 1#1: start worker processes
2024/07/13 08:09:22 [notice] 1#1: start worker process 29
2024/07/13 08:09:22 [notice] 1#1: start worker process 30
vagrant@Master:~$
```

2. Following Logs in Real-Time

```
docker logs -f my_container
```

3. Showing the Last 10 Lines of Logs

```
docker logs --tail 10 my_container
```

4. Showing Logs with Timestamps

```
docker logs -t my_container
```

5. Showing Logs Since a Specific Time

```
docker logs --since "2023-07-11T15:00:00" my_container
```

6. Combining Options

```
docker logs -f --tail 10 --since "10m" my_container
```