# Build a Swarm cluster for production

## Prerequisites

- An Amazon Web Services (AWS) account
- Familiarity with AWS features and tools, such as:
    - Elastic Cloud (EC2) Dashboard
    - Virtual Private Cloud (VPC) Dashboard
    - VPC Security groups
    - Connecting to an EC2 instance using SSH

## Step 1. Add network security rules

AWS uses a “security group” to allow specific types of network traffic on your VPC network. The default security group’s initial set of rules deny all inbound traffic, allow all outbound traffic, and allow all traffic between instances.

You’re going to add a couple of rules to allow inbound SSH connections and inbound container images. This set of rules somewhat protects the Engine, Swarm, and Consul ports. For a production environment, you would apply more restrictive security measures. Do not leave Docker Engine ports unprotected.

From your AWS home console, do the following:

1. Click VPC - Isolated Cloud Resources.
   The VPC Dashboard opens.
2. Navigate to Security Groups.
3. Select the default security group that’s associated with your default VPC.
4. Add the following two rules.

| Type | Protocol | Port Range | Source |
|---|---|---|---|
| SSH | TCP | 22 | 0.0.0.0/0 |
| HTTP | TCP | 80 | 0.0.0.0/0 |

The SSH connection allows you to connect to the host while the HTTP is for container images.

## Step 2. Create your instances

In this step, you create five Linux hosts that are part of your default security group. When complete, the example deployment contains three types of nodes:

| Node Description | Name |
|---|---|
| Swarm primary and secondary managers | `manager0`, `manager1` |
| Swarm node | `node0`, `node1` |
| Discovery backend | `consul0` |

To create the instances do the following:

1. Open the EC2 Dashboard and launch five EC2 instances, one at a time.
    - During Step 1: Choose an Amazon Machine Image (AMI), pick the *Amazon Linux AMI*.
    - During Step 5: Tag Instance, under Value, give each instance one of these names:
        - `manager0`
        - `manager1`
        - `consul0`
        - `node0`
        - `node1`
    - During Step 6: Configure Security Group, choose Select an existing security group and pick the “default” security group.
2. Review and launch your instances.

## Step 3. Install Engine on each node

In this step, you install Docker Engine on each node. By installing Engine, you enable the Swarm manager to address the nodes via the Engine CLI and API.

SSH to each node in turn and do the following.

1. Update the yum packages.
   Keep an eye out for the “y/n/abort” prompt:

```
$ sudo yum update
```

2. Run the installation script.

```
3.$ curl -sSL https://get.docker.com/ | sh
```

4. Configure and start Engine so it listens for Swarm nodes on port `2375`.

```
5.$ sudo docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock
```

6. Verify that Docker Engine is installed correctly:

```
7.$ sudo docker run hello-world
```

   The output should display a “Hello World” message and other text without any error messages.

8. Give the `ec2-user` root privileges:

```
9.$ sudo usermod -aG docker ec2-user
```

10. Enter `logout`.

### Troubleshooting

- If entering a `docker` command produces a message asking whether docker is available on this host, it may be because the user doesn’t have root privileges. If so, use `sudo` or give the user root privileges.
- For this example, don’t create an AMI image from one of your instances running Docker Engine and then re-use it to create the other instances. Doing so will produce errors.
- If your host cannot reach Docker Hub, the `docker run` commands that pull container images may fail. In that case, check that your VPC is associated with a security group with a rule that allows inbound traffic (e.g., HTTP/TCP/80/0.0.0.0/0). Also Check the Docker Hub status page for service availability.

## Step 4. Set up a discovery backend

Here, you’re going to create a minimalist discovery backend. The Swarm managers and nodes use this backend to authenticate themselves as members of the cluster. The Swarm managers also use this information to identify which nodes are available to run containers.

To keep things simple, you are going to run a single consul daemon on the same host as one of the Swarm managers.

1. Use SSH to connect to the `consul0` instance.

```
2.$ ifconfig
```

3. From the output, copy the `eth0` IP address from `inet addr`.
4. Paste the launch command into the command line:

```
5.$ docker run -d -p 8500:8500 --name=consul progrium/consul -server -bootstrap
```

6. Enter `docker ps`.
   From the output, verify that a consul container is running. Then, disconnect from the `consul0` instance.

Your Consul node is up and running, providing your cluster with a discovery backend. To increase its reliability, you can create a high-availability cluster using a trio of consul nodes using the link mentioned at the end of this page. (Before creating a cluster of consul nodes, update the VPC security group with rules to allow inbound traffic on the required port numbers.)

## Step 5. Create Swarm cluster

After creating the discovery backend, you can create the Swarm managers. In this step, you are going to create two Swarm managers in a high-availability configuration. The first manager you run becomes the Swarm'*sprimary manager*. Some documentation still refers to a primary manager as a "master", but that term has been superseded. The second manager you run serves as a *replica*. If the primary manager becomes unavailable, the cluster elects the replica as the primary manager.

1. Use SSH to connect to the `manager0` instance and use `ifconfig` to get its IP address.

```
2.$ ifconfig
```

3. To create the primary manager in a high-availability Swarm cluster, use the following syntax:

```
4.$ docker run -d -p 4000:4000 swarm manage -H :4000 --replication --advertise
    <manager0_ip>:4000 consul://<consul0_ip>:8500
```

   Replacing `<manager0_ip>` and `<consul0_ip>` with the IP address from the previous command, for example:

```
$ docker run -d -p 4000:4000 swarm manage -H :4000 --replication --advertise
172.30.0.125:4000 consul://172.30.0.161:8500
```

5. Enter `docker ps`.
   From the output, verify that a Swarm cluster container is running. Then, disconnect from the `manager0`instance.
6. Connect to the `manager1` node and use `ifconfig` to get its IP address.

```
7.$ ifconfig
```

8. Start the secondary Swarm manager using following command.
   Replacing `<manager1_ip>` with the IP address from the previous command, for example:

```
$ docker run -d -p 4000:4000 swarm manage -H :4000 --replication --advertise
<manager1_ip>:4000 consul://172.30.0.161:8500
```

9. Enter `docker ps`to verify that a Swarm container is running. Then disconnect from the `manager1` instance.
10. Connect to `node0` and `node1` in turn and join them to the cluster.
    a. Get the node IP addresses with the `ifconfig` command.
    b. Start a Swarm container each using the following syntax:

```
docker run -d swarm join --advertise=<node_ip>:2375 consul://<consul0_ip>:85
00
```

For example:

```
$ docker run -d swarm join --advertise=172.30.0.69:2375 consul://172.30.0.16
1:8500
```

c. Enter `docker ps` to verify that the Swarm cluster container started from the previous command is running.

Your small Swarm cluster is up and running on multiple hosts, providing you with a high-availability virtual Docker Engine. To increase its reliability and capacity, you can add more Swarm managers, nodes, and a high-availability discovery backend.

## Step 6. Communicate with the Swarm

You can communicate with the Swarm to get information about the managers and nodes using the Swarm API, which is nearly the same as the standard Docker API. In this example, you use SSL to connect to `manager0` and`consul0` host again. Then, you address commands to the Swarm manager.

1. Get information about the manager and nodes in the cluster:

```
2.$ docker -H :4000 info
```

   The output gives the manager's role as primary (`Role: primary`) and information about each of the nodes.

3. Run an application on the Swarm:

```
4.$ docker -H :4000 run hello-world
```

5. Check which Swarm node ran the application:

```
6.$ docker -H :4000 ps
```

## Step 7. Test Swarm failover

To see the replica instance take over, you're going to shut down the primary manager. Doing so kicks off an election, and the replica becomes the primary manager. When you start the manager you shut down earlier, it becomes the replica.

1. SSH connection to the `manager0` instance.
2. Get the container id or name of the `swarm` container:

```
3.$ docker ps
```

4. Shut down the primary manager, replacing `<id_name>` with the container's id or name (for example, "8862717fe6d3" or "trusting_lamarr").

```
5.docker rm -f <id_name>
```

6. Start the Swarm manager. For example:

```
7.$ docker run -d -p 4000:4000 swarm manage -H :4000 --replication --advertise
    172.30.0.161:4000 consul://172.30.0.161:8500
```

8. Review the Engine's daemon logs the logs, replacing `<id_name>` with the new container's id or name:

```
9.$ sudo docker logs <id_name>
```

   The output shows will show two entries like these ones:

```
time="2016-02-02T02:12:32Z" level=info msg="Leader Election: Cluster leaders
hip lost"
time="2016-02-02T02:12:32Z" level=info msg="New leader elected: 172.30.0.160
:4000"
```

10. To get information about the manager and nodes in the cluster, enter:

```
11. $ docker -H :4000 info
```

You can connect to the `manager1` node and run the `info` and `logs` commands. They will display corresponding entries for the change in leadership.