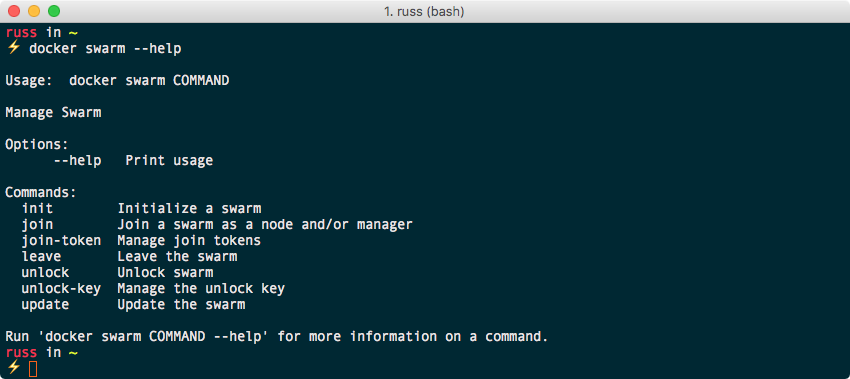
**Installing Docker Swarm**

As we are already running a version of Docker with inbuilt support for Docker Swarm, there isn't anything we need to do to install Docker Swarm; we can verify that Docker Swarm is available on our installation by running the following command:

**$ docker swarm --help**

We should see something that looks like the following Terminal output when running the command:



**Docker Swarm roles**

Which roles are involved with Docker Swarm? The below two roles a host can assume when running within a Docker Swarm cluster:

Swarm manager

Swarm worker

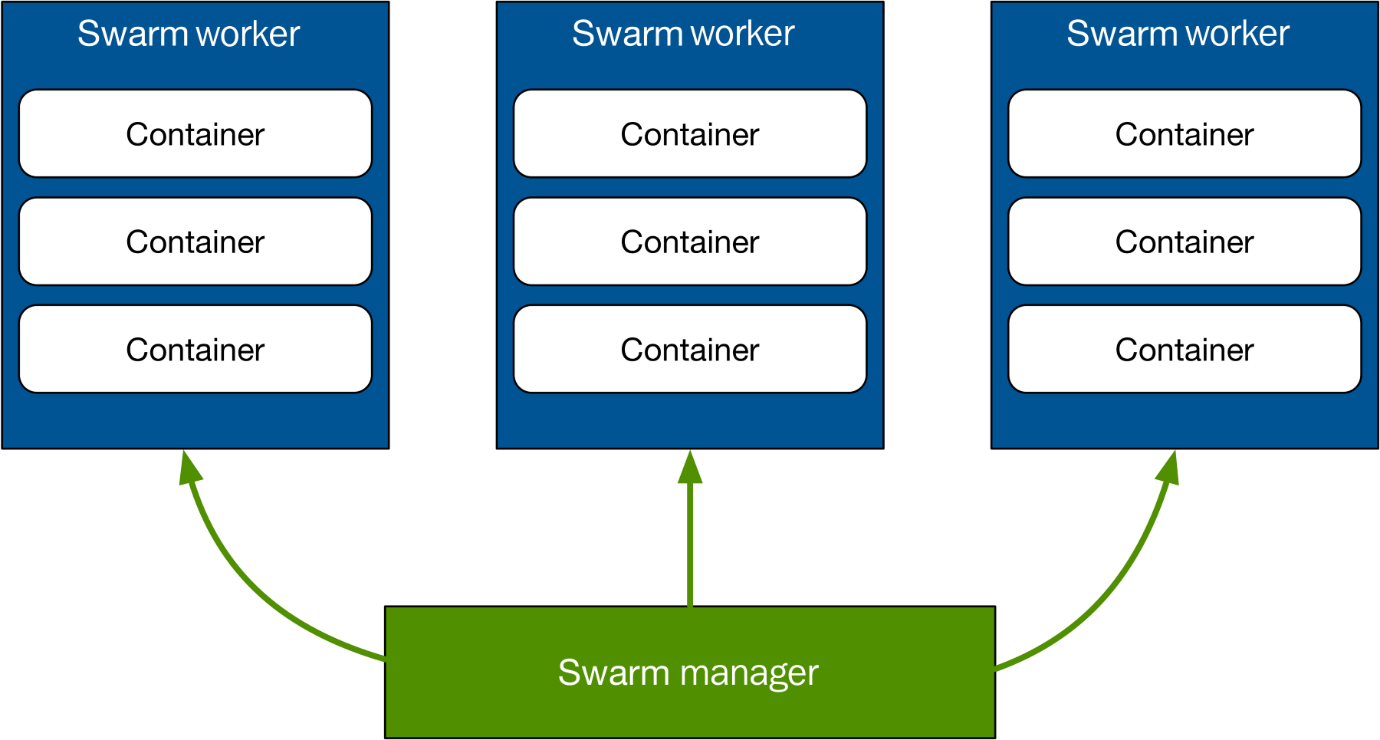
**Swarm manager**

The Swarm manager is the host that is the central management point for all Swarm hosts. The Swarm manager is where we issue all our commands to control those nodes. We can switch between the nodes, join nodes, remove nodes, and manipulate those hosts.

Each cluster can run several Swarm managers. As a general rule, for production, we should run a minimum of five Swarm managers: this would mean that our cluster can take a maximum of two Swarm manager nodes failures before we start to have any errors. Swarm managers use the to maintain a consistent state across all of the manager nodes.

**Swarm worker**

The Swarm workers; also referred to as Docker hosts, are those that run the Docker containers. Swarm workers are managed from the **Swarm manager**.



This is an illustration of all the Docker Swarm components. We see that the Docker **Swarm manager** talks to each Swarm host that is running the Swarm container.

**Using Docker Swarm**

Let's now take a look at using Swarm and how we can perform the following tasks:

* Creating a cluster
* Joining workers
* Listing nodes
* Managing a cluster

**Creating a cluster**

Let's start by creating a cluster, which starts with a Swarm manager. Since we are going to be creating a multi-node cluster on our local machine, we should use Docker Machine to launch a host by running this command:

**$ docker-machine create \**  
 **-d virtualbox \**  
 **swarm-manager**

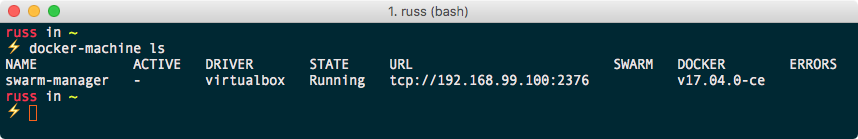
An abridged version of the output we get is shown here:

**(swarm-manager) Creating VirtualBox VM...**  
**(swarm-manager) Creating SSH key...**  
**(swarm-manager) Starting the VM...**  
**Docker is up and running!**  
**To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env swarm-manager**

The **swarm-manager** node is now up running using VirtualBox. we can confirm this by running:

**$ docker-machine ls**

We should see something similar to the following output:



Now let's point Docker Machine at the new Swarm manager. From the preceding output when we created the Swarm manager, we can see it tells us how to point to the node:

**$ docker-machine env swarm-manager**

This will show us the commands needed to configure our local Docker client to talk to our newly launched Docker host: following we can see the configuration we had returned when we ran the command:

**export DOCKER\_TLS\_VERIFY="1"**  
**export DOCKER\_HOST="tcp://192.168.99.100:2376"**  
**export DOCKER\_CERT\_PATH="/Users/russ/.docker/machine/machines/swarm-manager"**  
**export DOCKER\_MACHINE\_NAME="swarm-manager"**  
**# Run this command to configure our shell:**  
**# eval $(docker-machine env swarm-manager)**

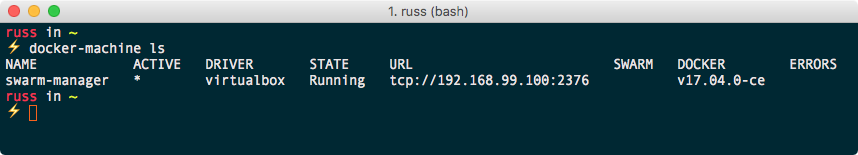
Upon running the previous command, we are told to run the following command to point to the Swarm manager:

**$ eval $(docker-machine env swarm-manager)**

Now if we look at which machines are on our host, we can see that we have the swarm-master host as well as it now set to ACTIVE, which means we can now run commands on it:

**$ docker-machine ls**

It will show us something like the following:



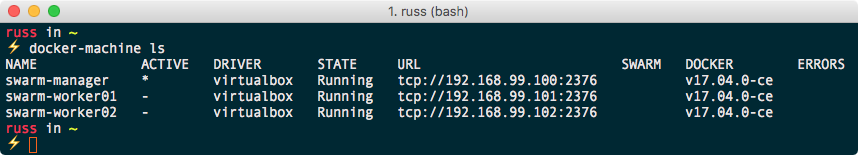
Now that we have the first host up-and-running, we should add the two worker nodes. To do this, simply run the following command to launch two more Docker hosts:

**$ docker-machine create \**  
 **-d virtualbox \**  
 **swarm-worker01  
$ docker-machine create \**  
 **-d virtualbox \**  
 **swarm-worker02**

Once we have launched the two additional hosts, we can get the list of hosts using this command:

**$ docker-machine ls**

It will show something like the following:



It is worth pointing out that, so far, we have not done anything to create our Swarm cluster; we have only launched the hosts it will be running on.

*Please note that one of the columns when running the docker-machine ls command is SWARM. This only contains information if we have launched our Docker hosts using the standalone Docker Swarm command, which is built into Docker Machine.*

Let's bootstrap our Swarm manager. To do this, we will pass the results of a few Docker Machine commands to our host. The command to run to create our manager is:

**$ docker $(docker-machine config swarm-manager) swarm init \**  
 **--advertise-addr $(docker-machine ip swarm-manager):2377 \**  
 **--listen-addr $(docker-machine ip swarm-manager):2377**

We should receive a message similar to this one:

**Swarm initialized: current node (qha7m9bf55wwd8p3e0jiyk7yf) is now a manager.**  
**To add a worker to this swarm, run the following command:**  
  
**docker swarm join \**  
**--token SWMTKN-1-3fxwovyzh24120myqbzolpen303uzk562y26q4z1wgxto5avm3-4aiagep3e2a1nwyaf4yjp7ic5 \**  
**192.168.99.100:2377**  
  
**To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.**

As we can see from the output, once our manager is initialized, we are given a unique token. This token will be needed for the worker nodes to authenticate themselves and join our cluster.

**Using Docker Swarm**

Let's now take a look at using Swarm and how we can perform the following tasks:

* Creating a cluster
* Joining workers
* Listing nodes
* Managing a cluster

**Joining workers**

To add our two workers to the cluster, run the following commands, making sure we replace the token with the one we received when initializing our own manager:

**$ docker $(docker-machine config swarm-worker01) swarm join \**  
 **$(docker-machine ip swarm-manager):2377 \**  
 **--token SWMTKN-1-3fxwovyzh24120myqbzolpen303uzk562y26q4z1wgxto5avm3-4aiagep3e2a1nwyaf4yjp7ic5**

For the second worker, we need to run this:

**docker $(docker-machine config swarm-worker02) swarm join \**  
 **$(docker-machine ip swarm-manager):2377 \**  
 **--token SWMTKN-1-3fxwovyzh24120myqbzolpen303uzk562y26q4z1wgxto5avm3-4aiagep3e2a1nwyaf4yjp7ic5**

Both times, we should get confirmation that our node has joined the cluster:

**This node joined a swarm as a worker.**

**Listing nodes**

We can check the Swarm by running the following command:

**$ docker-machine ls**

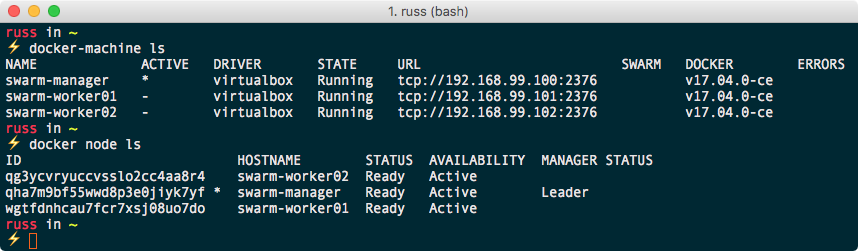
Check that our local Docker client is still configured to connect to the swarm-manager node, and if it isn't, rerun the following command:

**$ eval $(docker-machine env swarm-manager)**

Now that we are connecting to the swarm-manager node, we can run the following:

**$ docker node ls**

This will connect to the swarm-master and query all of the nodes that form our cluster. We should see that all three of our nodes are listed:



**Managing a cluster**

Let's see how we can perform some management of all of these cluster nodes that we are creating.

So there are two ways we can go about managing these Swarm hosts and the containers on each host that we are creating, but first, we need to know some information about them.

As we have already seen, we can list the nodes within the cluster using our local Docker client, as it is already configured to connect to the Swarm manager host. We can simply type this:

**$ docker info**

It will give us lots of information about the host, as we can see from the following output:

**Containers: 0**  
 **Running: 0**  
 **Paused: 0**  
 **Stopped: 0**  
**Images: 0**  
**Server Version: 17.04.0-ce**  
**Storage Driver: aufs**  
**Root Dir: /mnt/sda1/var/lib/docker/aufs**  
**Backing Filesystem: extfs**  
**Dirs: 0**  
**Dirperm1 Supported: true**  
**Logging Driver: json-file**  
**Cgroup Driver: cgroupfs**  
**Plugins:**  
 **Volume: local**  
 **Network: bridge host macvlan null overlay**  
**Swarm: active**  
 **NodeID: qha7m9bf55wwd8p3e0jiyk7yf**  
 **Is Manager: true**  
 **ClusterID: n5akyh6xsnc15qnx5ccp54vrr**  
 **Managers: 1**  
 **Nodes: 3**  
 **Orchestration:**  
 **Task History Retention Limit: 5**  
 **Raft:**  
 **Snapshot Interval: 10000**  
 **Number of Old Snapshots to Retain: 0**  
 **Heartbeat Tick: 1**  
 **Election Tick: 3**  
 **Dispatcher:**  
 **Heartbeat Period: 5 seconds**  
 **CA Configuration:**  
 **Expiry Duration: 3 months**  
 **Node Address: 192.168.99.100**  
 **Manager Addresses:**  
 **192.168.99.100:2377**  
**Runtimes: runc**  
**Default Runtime: runc**  
**Init Binary:**  
**containerd version: 422e31ce907fd9c3833a38d7b8fdd023e5a76e73**  
**runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228**  
**init version: 949e6fa**  
**Security Options:**  
 **seccomp**  
 **Profile: default**  
**Kernel Version: 4.4.59-boot2docker**  
**Operating System: Boot2Docker 17.04.0-ce (TCL 7.2); HEAD : c69677f - Thu Apr 6 16:26:16 UTC 2017**  
**OSType: linux**  
**Architecture: x86\_64**  
**CPUs: 1**  
**Total Memory: 995.8 MiB**  
**Name: swarm-manager**  
**ID: VKLO:MKJK:Y4UD:2IXV:WBA3:LTZE:J4MU:MGAD:VF7Z:QVVI:XNQG:SMAB**  
**Docker Root Dir: /mnt/sda1/var/lib/docker**  
**Debug Mode (client): false**  
**Debug Mode (server): true**  
 **File Descriptors: 32**  
 **Goroutines: 149**  
 **System Time: 2018-12-16T16:56:29.683890515Z**  
 **EventsListeners: 0**  
**Username: russ**  
**Registry: https://index.docker.io/v1/**  
**Labels:**  
 **provider=virtualbox**  
**Experimental: false**  
**Insecure Registries:**  
 **127.0.0.0/8**  
**Live Restore Enabled: false**

As we can see, there is information about the cluster in the Swarm section; however, we are only able to run the docker info command against the host our client is currently configured to communicate with; luckily, the docker node command is cluster aware, so we can use that to get information on each node within our cluster, like this, for example:

**$ docker node inspect swarm-manager --pretty**

*Passing the --pretty flag with the docker node inspect command will render the output in the easy-to-read format we see as follows. If --pretty is left out, Docker will return the raw JSON object containing the results of the query the inspect command runs against the cluster.*

This should provide the following information on our Swarm manager:

**ID: qha7m9bf55wwd8p3e0jiyk7yf**  
**Labels:**  
**Hostname: swarm-manager**  
**Joined at: 2018-12-16 16:56:47.092119605 +0530 utc**  
**Status:**  
 **State: Ready**  
 **Availability: Active**   
 **Address: 192.168.99.100**  
**Manager Status:**  
 **Address: 192.168.99.100:2377**  
 **Raft Status: Reachable**  
 **Leader: Yes**  
**Platform:**  
 **Operating System: linux**  
 **Architecture: x86\_64**  
**Resources:**  
 **CPUs: 1**  
 **Memory: 995.8 MiB**  
**Plugins:**  
 **Network: bridge, host, macvlan, null, overlay**  
 **Volume: local**  
**Engine Version: 17.04.0-ce**  
**Engine Labels:**  
 **- provider = virtualbox**

By Running the same command, but this time targeting one of the worker nodes:

**$ docker node inspect swarm-worker01 --pretty**

It gives us similar information:

**ID: wgtfdnhcau7fcr7xsj08uo7do**  
**Labels:**  
**Hostname: swarm-worker01**  
**Joined at: 2018-12-16 16:57:27.532507218 +0530 utc**  
**Status:**  
 **State: Ready**  
 **Availability: Active**  
 **Address: 192.168.99.101**  
**Platform:**  
 **Operating System: linux**  
 **Architecture: x86\_64**  
**Resources:**  
 **CPUs: 1**  
 **Memory: 995.8 MiB**  
**Plugins:**  
 **Network: bridge, host, macvlan, null, overlay**  
 **Volume: local**  
**Engine Version: 17.04.0-ce**  
**Engine Labels:**  
 **- provider = virtualbox**

The information block is missing the information about the state of the manager functionality. This is because the worker nodes do not need to know about the status of the manager nodes; they just need to know they are allowed to receive instructions from the managers. But we can see the information about this host, such as the number of containers, the numbers of images on the host, and information about the CPU and memory as well as other interesting information about the host.

**Promoting a worker node**

If we wanted to perform some maintenance on our single manager node, but need to maintain the availability of our cluster, we can promote a worker node to a manager node.

While we have our local three-node cluster up and running, let's promote swarm-worker01 to be a new manager. To do this, run the following:

**$ docker node promote swarm-worker01**

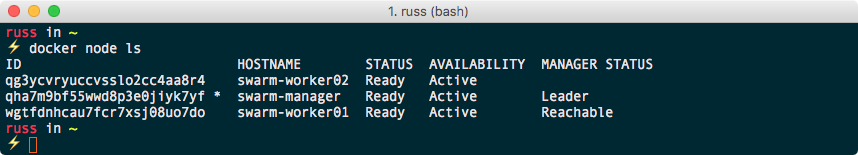
We receive a message confirming that your node has been promoted immediately after executing the command:

**Node swarm-worker01 promoted to a manager in the swarm.**

List the nodes by running this:

**$ docker node ls**

This should show that now there are two nodes that now display something in the MANAGER STATUS column:



Our swarm-manager node is still the primary manager node though. Let's look at doing something about that.

**Demoting a manager node**

To demote a manager node to a worker, we simply need to run this:

**$ docker node demote swarm-manager**

Again, we will receive immediate feedback saying the following:

**Manager swarm-manager demoted in the swarm.**

Now we have demoted our node, and we can check the status of the nodes within the cluster by running this command:

**$ docker node ls**

As our local Docker client is still pointing toward, the newly demoted node, we will receive a message saying the following:

**Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or modify cluster state. Please run this command on a manager node or promote the current node to a manager.**

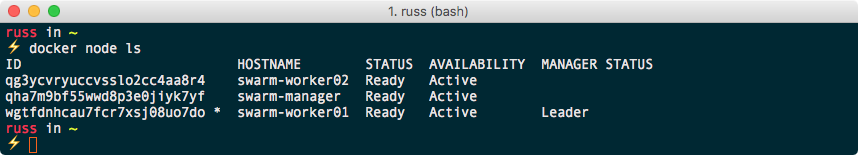
As we have already learned, it is easy to update our local client configuration to communicate with other nodes using Docker Machine. To point our local client to the new manager node, run the following:

**$ eval $(docker-machine env swarm-worker01)**

Now that our client is talking to a manager node again, rerun this:

**$ docker node ls**

It should list the nodes, as expected:



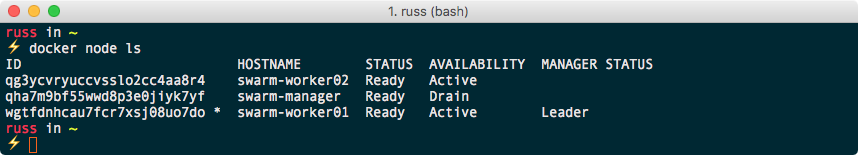
**Draining a node**

To temporarily remove a node from our cluster so that we can perform maintenance, we need to set the status of the node to Drain. Let's look at draining our former manager node. To do this, we need to run the following command:

**$ docker node update --availability drain swarm-manager**

This will stop any new tasks, such as new containers launching or being executed against the node we are draining. Once new tasks have been blocked, all running tasks will be migrated from the node we are draining to nodes with an ACTIVE status.

As can be seen from the following Terminal output, listing the nodes now shows that swarm-manager is listed as Drain in the AVAILABILITY column:



Now that our node is no longer accepting new tasks and all running tasks have been migrated to our two remaining nodes, we can safely perform our maintenance, such as rebooting the host. To reboot swarm-manager, run the following two commands, ensuring that we are connected to the Docker host (we should see the boot2docker banner, like the screenshot after the commands):

**$ docker-machine ssh swarm-manager**  
**$ sudo reboot**



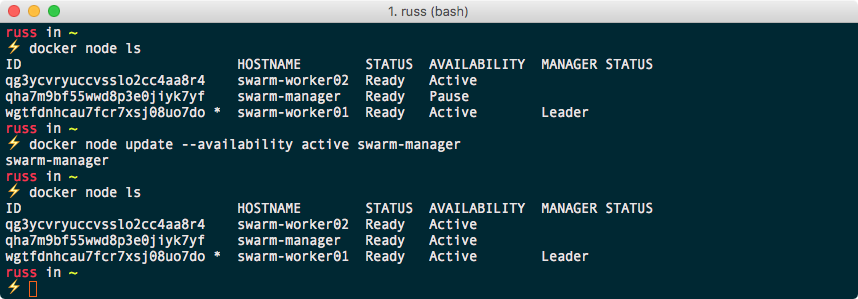
Once the host has been rebooted, run this:

**$ docker node ls**

It should show that the node has an AVAILABILITY of pause. To add the node back into the cluster, simply change the AVAILABILITY to active by running this:

**$ docker node update --availability active swarm-manager**

As we can see from the following Terminal output, our node is now active, meaning new tasks can be executed against it:



Now we completed how to create and manage a Docker Swarm cluster.

**Deleting a Swarm cluster**

If we no longer require it for the next section, we can delete our Swarm cluster by running the following command:

**$ docker-machine rm swarm-manager swarm-worker01 swarm-worker02**

Should we need to relaunch the Swarm cluster for any reason, we need to simply follow the instructions from Page 1 to recreate a cluster.