**Docker Swarm Cluster Creation**

### **Install Docker Engine**

Docker has a shell script we can use to check for any required dependencies and finally install the latest Docker release. It will set the daemon to start on boot. There are three versions we can choose from, which enable different sets of features. We're choosing main but you can also choose experimental to get features like "checkpoints" and "DAB files" that are marked as "not fully baked and may change before release". We'll install the release version do this on both nodes.

Let's take those IP addresses, SSH into both, and do what what we came here to do. Run this on both:

**curl -fsSL https://get.docker.com/ | sh**

Once it's done, run docker info on both and if you get back service information, so we know it worked and docker is running.

### **Initialize Swarm Mode**

Ok that was the hard part. From here it's a little anti-climatic. On the first node, we run:

**docker swarm init**

Note that you may get back an error like this:

Error response from daemon: could not choose an IP address to advertise since this system has multiple addresses on interface eth0 (45.55.87.48 and 10.17.0.7) - specify one with --advertise-addr

This just means docker see's multiple IP's on the host and can't figure out which one to use for servers talking to each other. I'll use the public IP for mine:

**docker swarm init --advertise-addr 45.55.87.48**

We'll get back a random key that we should copy and use to join the 2nd node. This key prevents rogue servers from trying to join our swarm. Notice that this key is to join a worker node to our swarm. The first node we just created is a manager and has powers to control the swarm, unlike a worker that just takes orders. One manager is fine in this config, as losing a manager doesn't kill containers on other nodes, it just means you'll need to bring the manager back online (or create a new swarm) before doing any Swarm activities like changing services. The other reason we don't want two managers here is that Swarm managers use [Raft](https://docs.docker.com/engine/swarm/raft/) to store data about the cluster, and it needs an odd number of managers to [maintain consensus](http://thesecretlivesofdata.com/raft/), so two managers wouldn't provide manager HA, we'd need three for that. This single manager scenario still gives us HA Nginx.

On node 2 let's paste in that command it gives us (your command will have a different token and IP obviously):

**docker swarm join \  
--token SWMTKN-1-1xtnjlu53y6x6hlgbw28yvtpovnydwxa4epe21awtkho9774ir-9x28is0oqn1on1m0nn275mykk \  
45.55.87.48:2377**

We now have a swarm with nodes! Almost magic!

### **Create A Nginx Service**

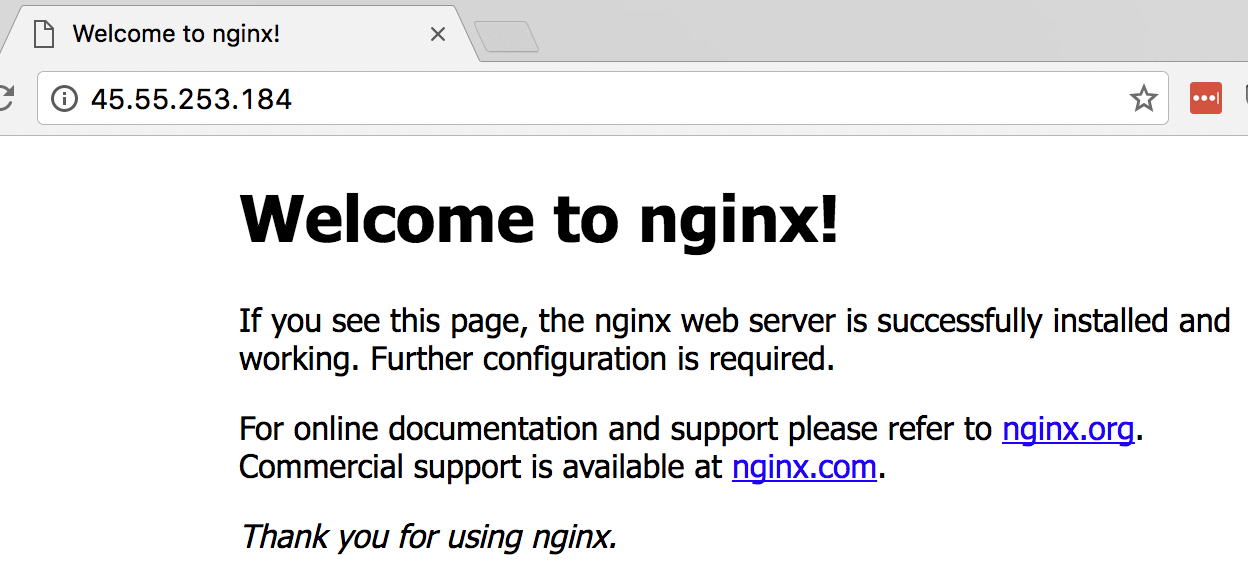
Let's deploy a simple Nginx web server to it, so run this from the manager node:

**docker service create --name nginx --publish 80:80 nginx**

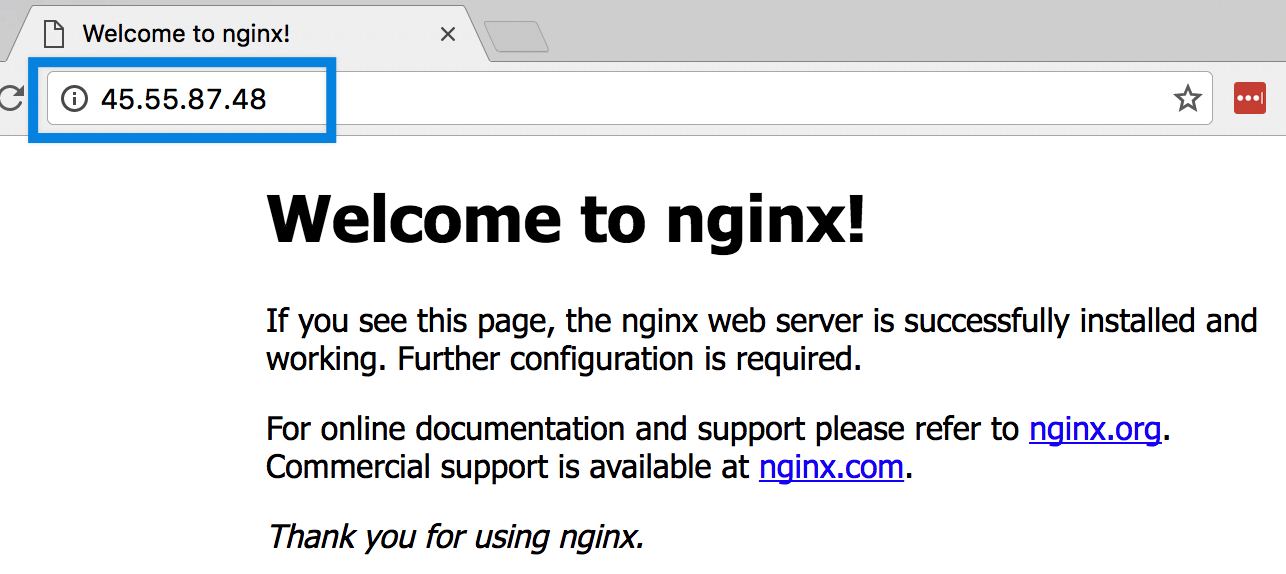
That created a Service, which is a new concept in 1.12 that uses Swarm Mode to control the scheduler. The service create command will ask the scheduler to execute a Task to start a nginx container on one of the nodes. Let's check to see if it's started:

**# docker service ps nginx  
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR   
0iwle3w4hh8r64mys22j89prt nginx.1 nginx docker-swarm-2 Running Running 3 seconds ago**

We should see 1/1 running. Notice at this point we're only running one container on one node. Let's check the public IP of that node where it's running to see if we get the nginx default index page.



Just to confirm only one is running, let's see about the other node.



Did you just see how the node didn't matter? Docker's new Swarm Mode also includes the Routing Mesh, a packet forwarder that causes every node in a Swarm to listen on published ports, and forward packets to the proper node/container. To take advantage of this for a http site, we could use DNS Round Robin to ensure our site stays up even if one container or server fails.

### **Scale Our Nginx Across Servers**

Routing Mesh is cool, but if the host that our Nginx container is running on dies, we still loose our web site. We made this Swarm so our web app could be highly-available, so let's do that. Run this command to spin up a 2nd container from the same image:

**docker service scale nginx=2**

And if we check our task list again, we should see the second container spinning up.

**# docker service ps nginx  
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR   
0iwle3w4hh8r64mys22j89prt nginx.1 nginx docker-swarm-2 Running Running 16 minutes ago   
6ghzany6acm40wu9m0sssu372 nginx.2 nginx docker-swarm-1 Running Running 4 seconds ago**