Swam:

In this Swarm cluster we made the choice that Node1 will be the leader and that the Swarm Cluster leader will listen on the private network:

```
# docker swarm init –listen-addr 192.168.0.161
```

# docker swarm join --token SWMTKN-1-5n9u1wqz5p4un6dv3vgrackvh0lij5p4f6dcm38ak64f5hx9i5-983zx6dv5ahdeqv3ipjer6416 192.168.0.161:2377

The choice of using private networking is important for your security especially that Docker Swarm is in its beta version.

Anyway, type the last generated command on the other nodes (node2 & node3 & etc).

Well once you run the same command on required nodes says that "This node joined a Swarm as a worker.". Let's go back to the master and verify it:

```
# docker node ls
 ID                           HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS
44o38p57llcnw2qv9258yba3n *  node1     Ready   Active         Leader
ae58smy84cj4jxbpncqd8r4it    node2     Ready   Active
```

Using Docker node command we can list all the nodes that have already joined the cluster.

##############################

To deploy an application image when Docker Engine is in swarm mode, you create a service. Frequently a service will be the image for a microservice within the context of some larger application. Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment. For more: Docker

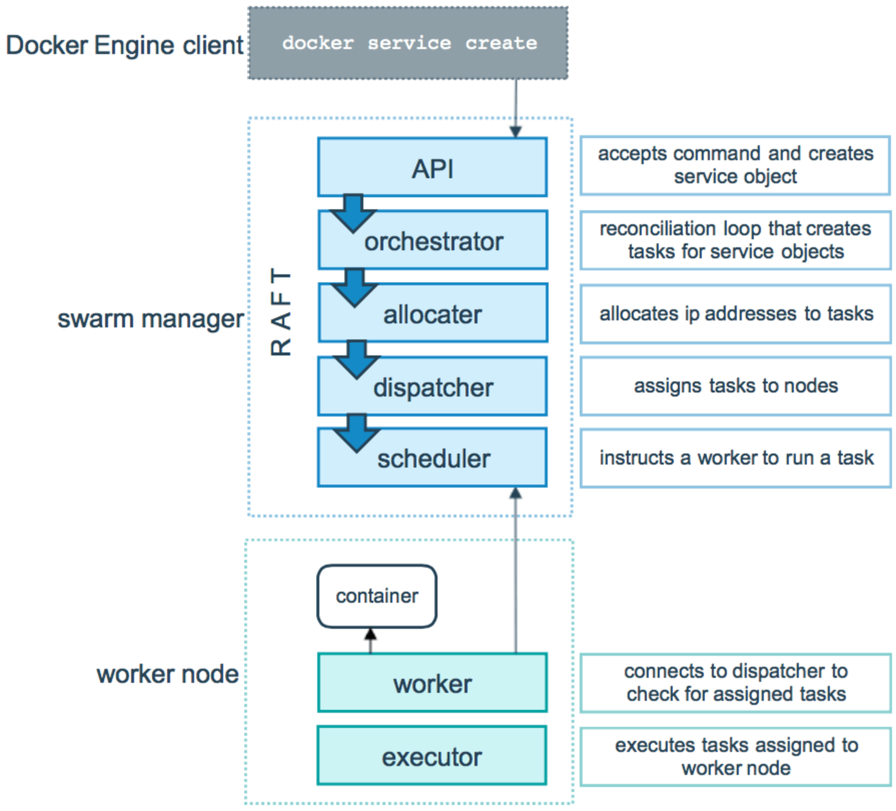

Docker Engine client
docker service create
swarm manager
RAFT
API
accepts command and creates service object
orchestrator
reconciliation loop that creates tasks for service objects
allocater
allocates ip addresses to tasks
dispatcher
assigns tasks to nodes
scheduler
instructs a worker to run a task
worker node
container
worker
connects to dispatcher to check for assigned tasks
executor
executes tasks assigned to worker node

Image Source: docker.com

When you create a service, you specify which container image to use and which commands to execute inside running containers. You also define options for the service including:

- the port where the swarm will make the service available outside the swarm
- an overlay network for the service to connect to other services in the swarm

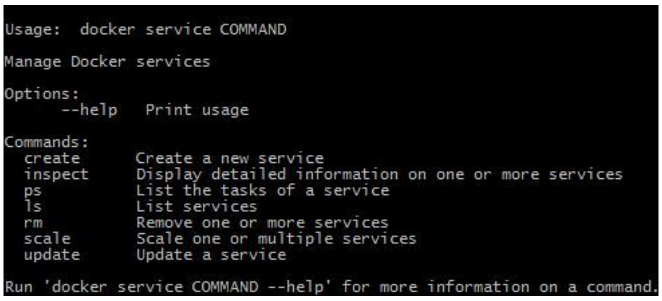
- CPU and memory limits and reservations
- a rolling update policy
- the number of replicas of the image to run in the swarm

Starting with a service

Let’s pick a service or a long running container. This could be the sleep 500 command, but we can just as easily make it ping docker.com.

Have a look at the new docker service command’s help page:

# docker service



```
Usage:  docker service COMMAND

Manage Docker services

Options:
      --help   Print usage

Commands:
  create      Create a new service
  inspect     Display detailed information on one or more services
  ps          List the tasks of a service
  ls          List services
  rm          Remove one or more services
  scale       Scale one or multiple services
  update      Update a service

Run 'docker service COMMAND --help' for more information on a command.
```

Let’s go one by one docker service command.

docker service create

docker service create is synonymous with docker run, so anything we can do with docker run should work the same.

```
# docker run busybox ping google.com
PING google.com (172.217.6.78): 56 data bytes
64 bytes from 172.217.6.78: seq=0 ttl=46 time=76.507 ms
64 bytes from 172.217.6.78: seq=1 ttl=46 time=76.387 ms
64 bytes from 172.217.6.78: seq=2 ttl=46 time=76.376 ms
64 bytes from 172.217.6.78: seq=3 ttl=46 time=76.325 ms
```

Here’s the equivalent command with docker service create:

```
# docker service create --name ping busybox ping google.com
3uyw3cqsdwizqenk7jcgv1sbq
```

We have create the service with name, which we can refer it easily in future for our reference. This help us to refresh anytime our service and also its generated GUID of assigned a name to the service so that we can reference it easily, but Docker has also generated a GUID of 3uyw3cqsdwizqenk7jcgv1sbq.

docker service ls

Check which services we have running/created type in docker service ls as below:

```
# docker service ls
ID            NAME  REPLICAS  IMAGE    COMMAND
3uyw3cqsdwiz  ping  1/1       busybox  ping google.com
```

docker service scale

Scaling this has never been easier, we just type in the name of the service and how many replicas we want.

```
# docker service scale ping=4
```

this command make ping service scaled to 4

We will now see that the docker service ls command gives 4/4 replicas as running and docker ps will show 4 unique containers running, too.

```
# docker service ls
ID                       NAME  REPLICAS  IMAGE     COMMAND
3uyw3cqsdwiz             ping  4/4       busybox   ping google.com
```

docker scale remove

Once you have finished your work either scale back to 0 or remove the service with docker service rm ping

```
# docker service rm ping
```

Update a Service

You can change almost everything about an existing service using the docker service update command. When you update a service, Docker stops its containers and restarts them with the new configuration.

Since Nginx is a web service, it will work much better if you publish port 80 to clients outside the swarm. You can specify this when you create the service, using the -p or –publish flag. When updating an existing service, the flag is –publish-add. There is also a –publish-rm flag to remove a port that was previously published.

Assuming that the **webserver** service from the previous section still exists, use the following command to update it to publish port 80.

```
# docker service update --publish-add 80 webserver
```

To verify that it worked, use docker service ls:

```
# docker service ls
ID            NAME       REPLICAS  IMAGE                  COMMAND
0nbjrzb080mf  webserver  1/1       motoskia/apache-php

# docker ps
```

Remove a Service

To remove a service, use the docker service remove command. You can remove a service by its ID or name, as shown in the output of the docker service ls command. The following command removes the ping service.

```
# docker service remove ping
```

Compose with swarm:

https://docs.docker.com/engine/swarm/stack-deploy/#push-the-generated-image-to-the-registry

Create the stack with `docker stack deploy`

$ docker stack deploy --compose-file docker-compose.yml stackdemo

```
docker stack services stackdemo

docker stack rm stackdemo
```

################################

Installing dependencies kubernetes

The first thing you must do is install the necessary dependencies. This will be done on all machines that will join the Kubernetes cluster. The first piece to be install is apt-transport-https (a package that allows using https as well as http in apt repository sources). This can be installed with the following command:

```
# apt-get update && apt-get install -y apt-transport-https
```

Our next dependency is Docker. Our Kubernetes installation will depend upon this, so install it with:

```
# apt install docker.io
```

Once that completes, start and enable the Docker service with the commands

```
# systemctl start docker
# systemctl enable docker
```

Installing Kubernetes

Installing the necessary components for Kubernetes is simple. Again, what we're going to install below must be installed on all machines that will be joining the cluster.

Our first step is to download and add the key for the Kubernetes install. Back at the terminal, issue the following command:

```
# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add
```

Next add a repository by creating the file

```
# vim /etc/apt/sources.list.d/kubernetes.list          ((add the following content.))
deb http://apt.kubernetes.io/ kubernetes-xenial main
```

Save and close that file. Install Kubernetes with the following commands:

```
# apt-get update
# apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

Initialize your master

With everything installed, go to the machine that will serve as the Kubernetes master and issue the command:

```
# kubeadm init --node-name master
```

```
kubeadm init --apiserver-advertise-address=172.31.18.61
```

When this completes, you'll be presented with the exact command you need to join the nodes to the master. This command gives Node joining details. make note of it.

Before you join a node, you need to issue the following commands in master (as a regular user):

```
# mkdir -p $HOME/.kube
# cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# chown $(id -u):$(id -g) $HOME/.kube/config
```

Deploying a pod network

You must deploy a pod network before anything will actually function properly. I'll demonstrate this by installing the Flannel pod network. This can be done with two commands (run on the master):

```
# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/
kube-flannel.yml
# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/
k8s-manifests/kube-flannel-rbac.yml
```

Issue the command

```
#  kubectl get pods --all-namespaces     ##to see that the pod network has been deployed
```

Joining a node

With everything in place, you are ready to join the node to the master. To do this, go to the node's terminal and issue the command:

```
# kubeadm join --token TOKEN SERVER_MASTER_IP:6443
```

Where TOKEN is the token you were presented after initializing the master and SERVER_MASTER_IP is the IP address of the master.

Once the node has joined, go back to the master and issue the command

```
#  kubectl get nodes                      ## to see the nodes has successfully joined.
```

Deploying a service

At this point, you are ready to deploy a service on your Kubernetes cluster. To deploy an NGINX service (and expose the service on port 80), run the following commands (from the master):

```
# kubectl run --image=nginx nginx-app --port=80 --env="DOMAIN=cluster"
# kubectl expose deployment nginx-app --port=80 --name=nginx-http
```

If you go to your node and issue the command

```
#  docker ps                                     ## you should see the service listed
```