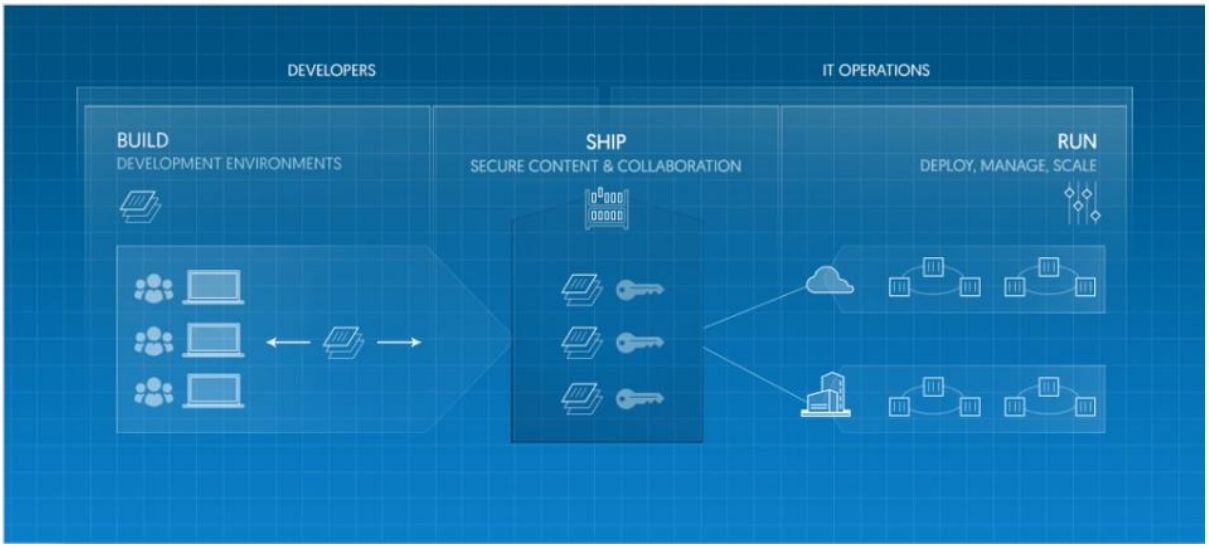
Docker Compose and Docker Swarm

Docker Inc. came up with a tool called 'Fig' which allowed you to use a single YAML file to orchestrate all your Docker containers and configurations. **Later the 'Fig' source code was used to create the tool called Docker-Compose.**

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.

What do we want to achieve?



Using Compose is basically a three-step process.

1. Define your app's environment with a **`Dockerfile`** so it can be reproduced anywhere.
2. **Great for running / Testing application in Dev / staging / QA and CI environments.**
3. Define the services that make up your app in **`docker-compose.yml`** so they can be run together in an isolated environment.
4. Lastly, run **`docker-compose`** `up` and Compose will start and run your entire app.

Sample `docker-compose.yml` file,

```
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
```

```
      - logvolume01:/var/log
      links:
      - redis
    redis:
      image: redis
  volumes:
    logvolume01: {}

-------------------------
  version: '3'
  services:
    wordpress:
      image: wordpress:latest
      ports:
        -  8080:80
       environment:
             WORDPRESS_DB_PASSWORD: abc123

    mysql:
      image: mysql:latest
      environment:
        MYSQL_ROOT_PASSWORD: abd123$
```

To get the Docker compose installed on Ubuntu/ CentOS, follow below steps.

```
$ sudo curl -o /usr/local/bin/docker-compose -L
https://github.com/docker/compose/releases/download/1.11.
2/docker-compose-$(uname -s)-$(uname -m)
```

Set the permissions.

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Check the docker compose version by running the command,

```
$ docker-compose -v
```

Once you have the docker-compose installed, create the .yml file and start creating multiple containers that are linked together.

Create a directory (e.g. dockerproject) and a file names as `docker-compose.yml`

The .yml file would have details about the image to be used, volumes to be mounted, links to be established, ports to be exposed, if a Dockerfile is used for container creation etc. and all diff images to be used for creating containers.

Commands:

build	Build or rebuild services
bundle	Generate a Docker bundle from the Compose file

config	Validate and view the compose file
create	Create services
down	Stop and remove containers, networks, images, and volumes
events	Receive real time events from containers
exec	Execute a command in a running container
help	Get help on a command
kill	Kill containers
logs	View output from containers
pause	Pause services
port	Print the public port for a port binding
ps	List containers
pull	Pull service images
push	Push service images
restart	Restart services
rm	Remove stopped containers
run	Run a one-off command
scale	Set number of containers for a service
start	Start services
stop	Stop services
top	Display the running processes
unpause	Unpause services
up	Create and start containers
version	Show the Docker-Compose version information

In docker-compose we can create an override file named as "`docker-compose.override.yml`" file. In such case when one runs the `docker-compose up -d` command the settings / instructions written in docker-compose file would be overridden by the settings written in "`docker-compose.override`" file.

Few more docker-compose examples,

Here's an example for spinning up a container for 'db' service and a container for 'Jboss wildfly'.

'db' is '**state full'** container, where the data written by DB service to the volume remains intact even if the container dies.

```
version: "2"
services:
   db:
     image: couchbase
     volume:
          - ~/couchbase: /opt/couchbase/var
     ports:
          - 8091:8091
          - 8092:8092
          - 8093:8093
          - 11210:11210
    web:
       image: arungupta/wildfly
       environment:
          - COUCHBASE_URI=db
       ports:
          - 8080:8080
          - 9990:9990
```

Here a JavaEE application is deployed on 'Jboss Wildfly'. The app would query 'db' container and display the results.
In Jboss the war file would be deployed into the location as, `/opt/jboss/wildfly/standalone/deployments/abc.war` inside the jboss container.

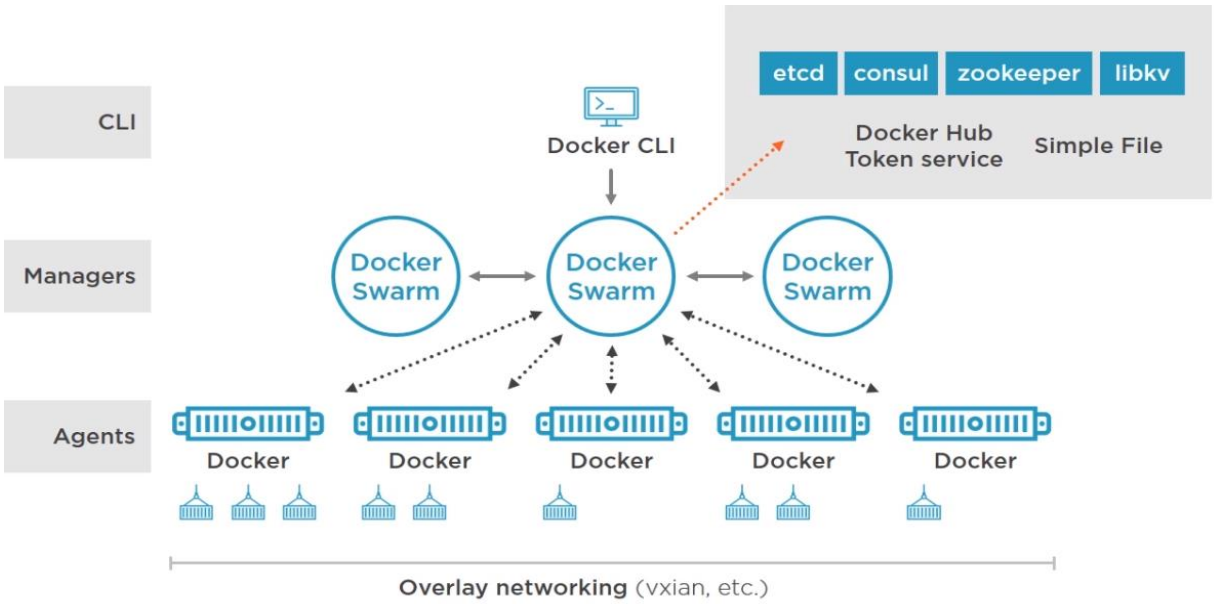
Now in Order to start the multi container app on multiple hosts so that there are multiple instances of the application running inside a cluster of more than one host.

Docker Swarm:

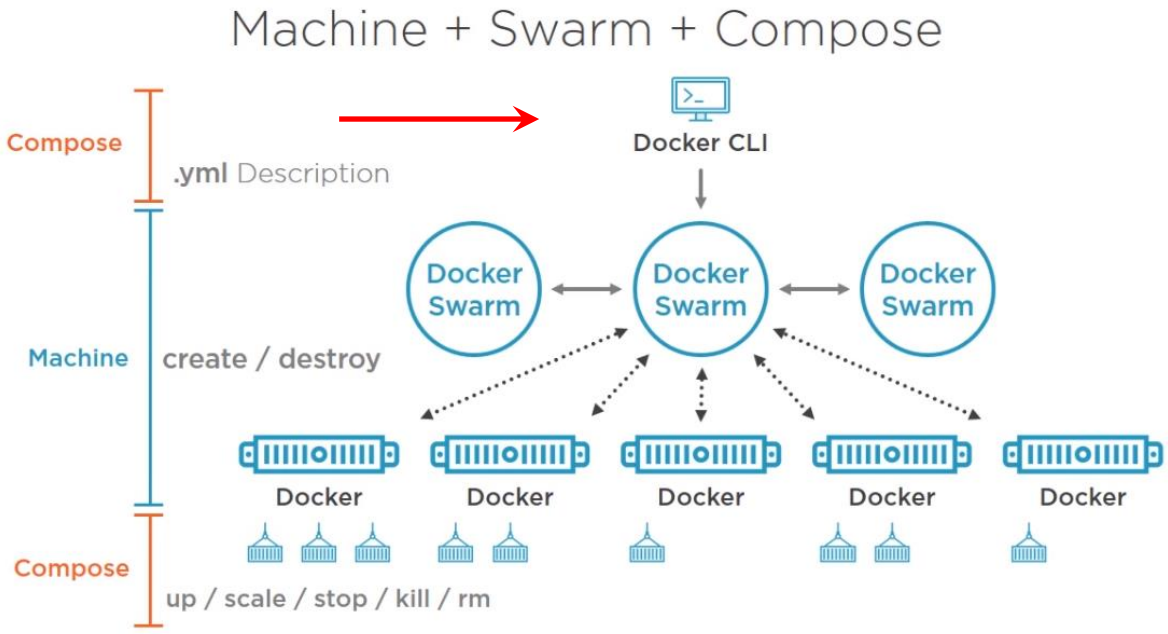
Let's look at **Docker Swarm:**

1. Native clustering for Docker
2. Provides a unified interface to a pool of Docker hosts.
3. Fully integrated with Docker machine and Docker compose
4. Swarm servers standard docker API
5. Version 1.2.4 and later,
    a. Reschedule containers when a node fails.
    b. Better node management

Docker Engine 1.12 introduced a new **swarm mode** for natively managing a cluster of Docker Engines called a **swarm**. Docker **swarm mode** implements Raft Consensus Algorithm and does not require using external key value store anymore, such as Consul or etcd.

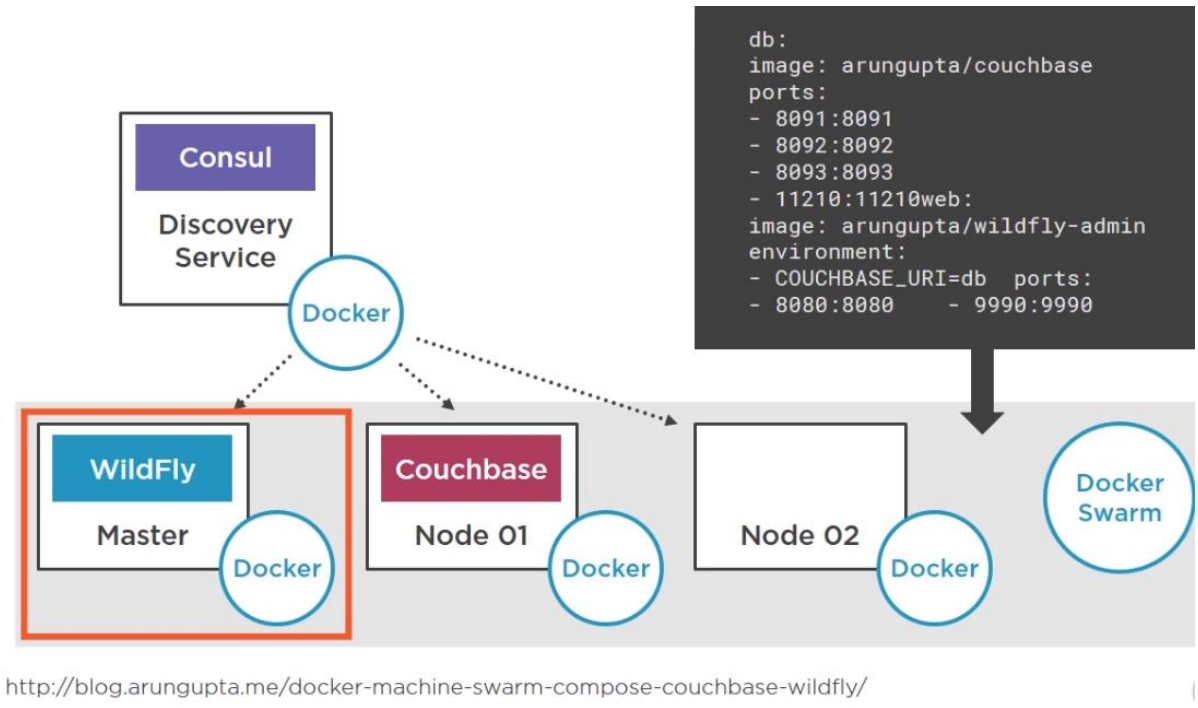


This can then be used as below..



To deploy an application to a Docker Swarm Cluster, first,

1) Create a discovery service with Docker machine and Consul as a key-value pair for authentication.
2) Start one more Docker machine registered with using Consul token service and working as part of swarm cluster as Manager.
3) Start one more node; add it to the cluster using Consul working and as Worker Node.
4) Start one more node working as a worker node and add it to the cluster. So this becomes a three node cluster with one manager and two worker nodes.
5) Now we will use the docker-compose file and push it to the Docker Swarm cluster to start two applications, one 'db' and 'web' with respective docker images.
6) We can use the distribution strategy called as '***spread***'. Using this we can spread the container across worker nodes or, we can use strategy as '***binpack***' to start all container on one node only, and allow may be bigger container on other worker nodes. Also, there's one more strategy called as, '***Random***' so as to create container randomly without even distribution across the cluster.



http://blog.arungupta.me/docker-machine-swarm-compose-couchbase-wildfly/

With Docker 1.12, **Docker swarm is integrated with docker machine**. This means that we can use docker CLI to create a swarm cluster, deploy apps to cluster and manage swarm.
This new version use **swarm kit** for working with cluster.
The new version also has a feature of scaling apps in cluster.
Swarm Kit is used for **Orchestrating Distributed system** for cluster.

RAFT Protocol introduced to reach to a consensus on which would be primary and secondary leader, so if primary goes down, it automatically decides on who will be the next leader.

Swarm Node – either a worker of manager

- Worker node comes with Docker Container executer
- Tasks run on Worker nodes
- Tasks are organized in services
- Each service has desired state, e.g. how many replicas.
- Resource / strategy aware – **Spread** strategy is default.
- Matches the desired and actual state of cluster.. – **reconciles the state to match it to the desired state.**
- Secured with TLS authentication.

To setup a swarm Mode cluster, we will have to open certain ports on VMs.

1) Port 2377 for cluster management communication
2) Port 7946 for communication among nodes
3) Port 4789 for overlay network traffic. To setup network across nodes.

To start with setting up the cluster let's begin with below steps.

- On google cloud or AWS, create 4 default (micro / standard in case of GC) instances (VMs) with either CentOS or Ubuntu OS.
- Edit the security group / Firewall rule with above ports opened for TCP protocol.
- We will have one manager node and two Worker nodes setup initially. We will later add one more node as Manager to add redundancy to the cluster in case of failure of first manager node.

Setting up Swarm Manager:

From the list of VMs, we will be making one VMs as a Master by initializing Swarm on it, thus making is a Master Node. This is done by running the command,

```
$ docker swarm init  --listen-addr <master-1-ip-address>:2377
```

Setting up Swarm Worker nodes

```
$ docker swarm join --token <autogenerated-token> <manager-ip-address>:2377
```

If a token is not available, we can retrieve with below command,

```
$ docker swarm join-token worker
```

To know the nodes that are part of the cluster, we can run the command as,

```
$ docker node ls
```

Adding secondary master … master-2 joining to cluster as master with master-1 as primary.
In this case while worker nodes are going to communicate to the master, they can talk to any master, but the request would then be forwarded to the primary master.

```
$ docker swarm join --manager --listen-addr $(hostname -
i):2377 $FIRST_MANAGER_IP:2377
```

We can also use docker-swarm in scaling mode in cluster.

In case of any constraints that we want to include in compose we can have a compose file as below,

```
version: "2"
services:
  foo:
    image: foo
    volumes_from: ["bar"]
    network_mode: "service:baz"
    environment:
      - "constraint:node==node-1"
  bar:
    image: bar
    environment:
      - "constraint:node==node-1"
  baz:
    image: baz
    environment:
      - "constraint:node==node-1"
```

Now once we have the cluster up and running we can create services inside the cluster by using below command.

```
$ docker service create --replicas=4 --name=webapp
ganeshhp/maven-petclinic-project
```

To check the service running on a node,

```
$ docker service ls
```

To inspect a running service inside a cluster, we can use

```
$ docker service inspect  --pretty <service-id>
```

Also,

```
$ docker service ps <service-id>
```

Now that we have a service running in the cluster, we should see to scale the service running in the cluster.

```
$ docker service scale <SERVICE-ID>=<number>
```

To delete a service running inside the cluster, use,

```
$ docker service rm <service-id>
```

Applying rolling updates,

```
$ docker service create --replicas 3 --name=webapp -update-
parallelism 2 --update-delay 10s ganeshhp/maven-pet-
clinic:latest
```

To drain a running node in the cluster..

```
$ docker node update --availability drain <Node-Id>
```

To make the node active again after maintenance.

```
$ docker node update --availability active <NODE-ID>
```

In case of anode is having problem is want to remove temporarily from the cluster, we can use the command,

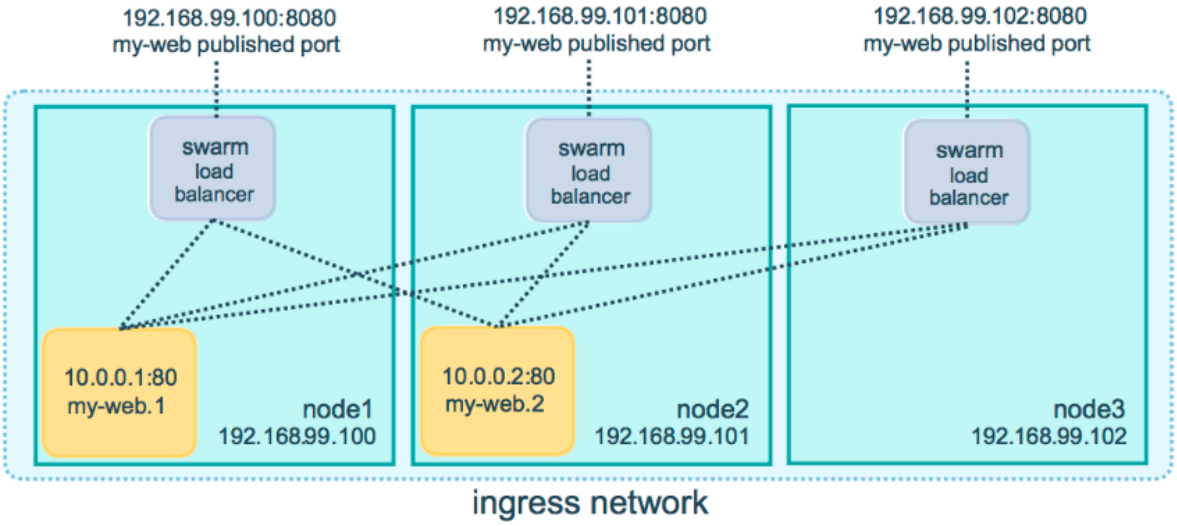
```
$ docker node update --availability passive <nodename>
```

Once the node is repaired, we can take it back to active mode by running command as,

```
$ docker node update --availability active <nodename>
```

To use swarm Mode routing mesh

```
$ docker service create --name my-web --publish
published=8080,target=80 --replicas 2 ganeshhp/petclinic-
maven-project
```



- Check below command as well to deploy using docker-compose.

```
$ docker deploy --compose-file docker-compose.yml
```

Container Monitoring:

Few important commands..

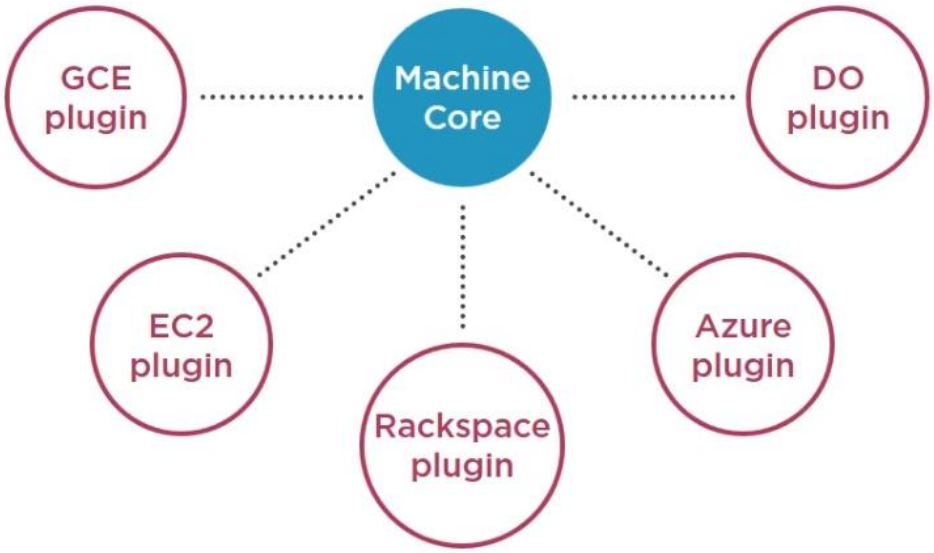
Run `$ docker stats` command with options such as `–LogEntries.`

Also we can use tools such as,

1) Docker Remote API:/ container/{container-name|CID}/stats
2) Docker Universal Control Plane
3) cAdvisor … tool from Google. This tools can be combined with Promethius and Kibana to display the dashboard with some analysis002E
4) NewRelic … can be used for very detailed Container monitoring and also apps running inside the container.
5) ELK … Elastic search + Logstash + Kibana

Few interesting references and points:

1) Eclipse, IntelliJ, NetBeans…. IDEs have support for containers. We can connect to Docker Host from these IDEs and can run, Test Docker images and applications.
2) Docker for Java.. tutorials, https://github.com/docker/labs/tree/master/java
3) While working with docker machine on MAC or Windows, we first have to create a host VM and this can be done using below command, `$ docker-machine create --driver=virtualbox myhost` … this command will create a VM Host named ‘myhost’ locally using VirtualBox as hypervisor. One can change the driver to AWS, Google, vagrant and similar to create a VM host.



If you are not running on Windows10, but are on Windows7, you will need the Docker Toolbox which comes with, docker client, docker machine, docker compose, Docker kitematic, Boot2Docker ISO and VirtualBox.