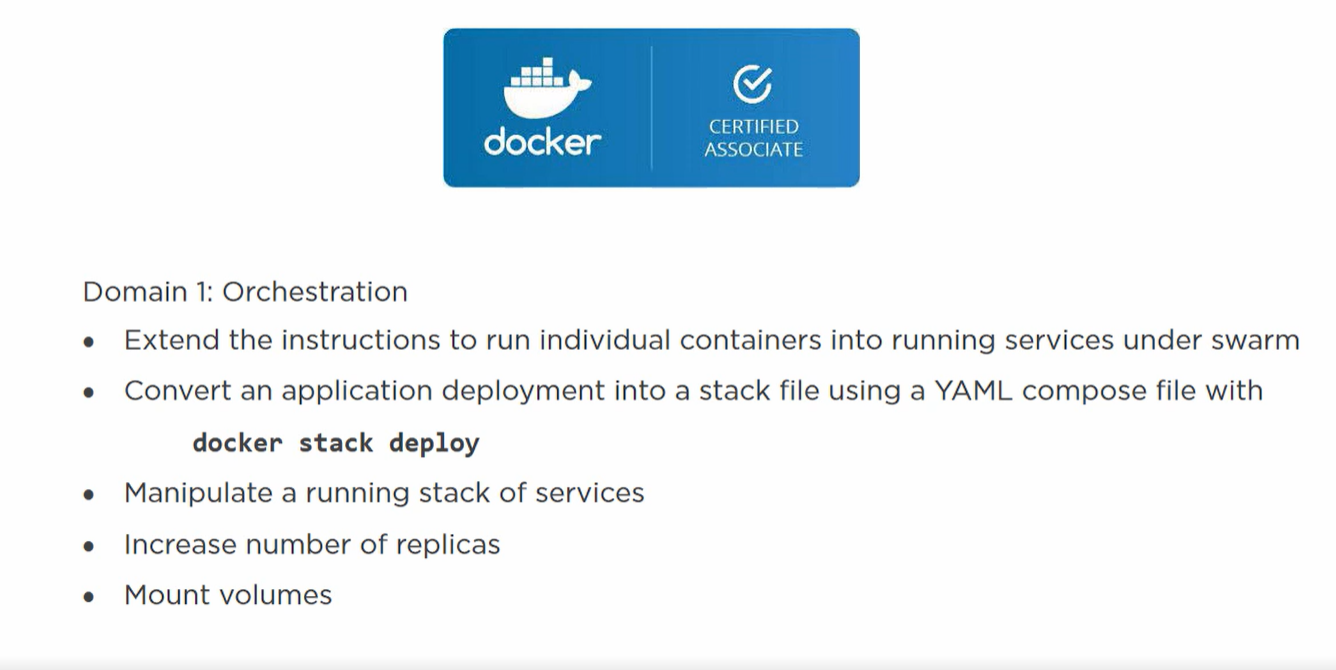
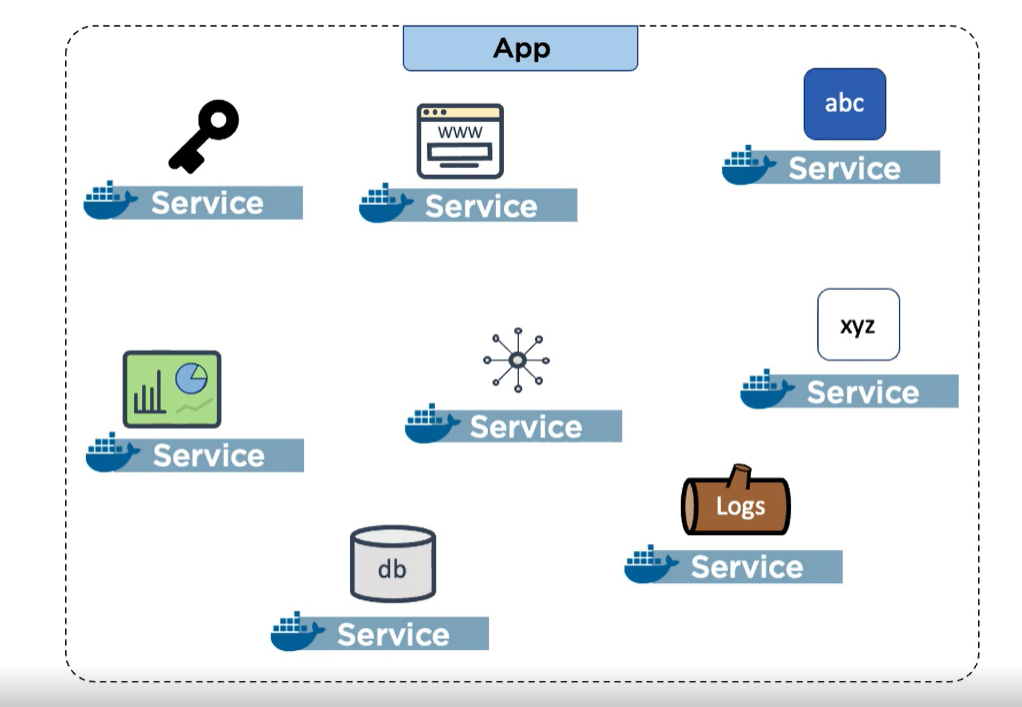
**SERVICES AND STACKS**



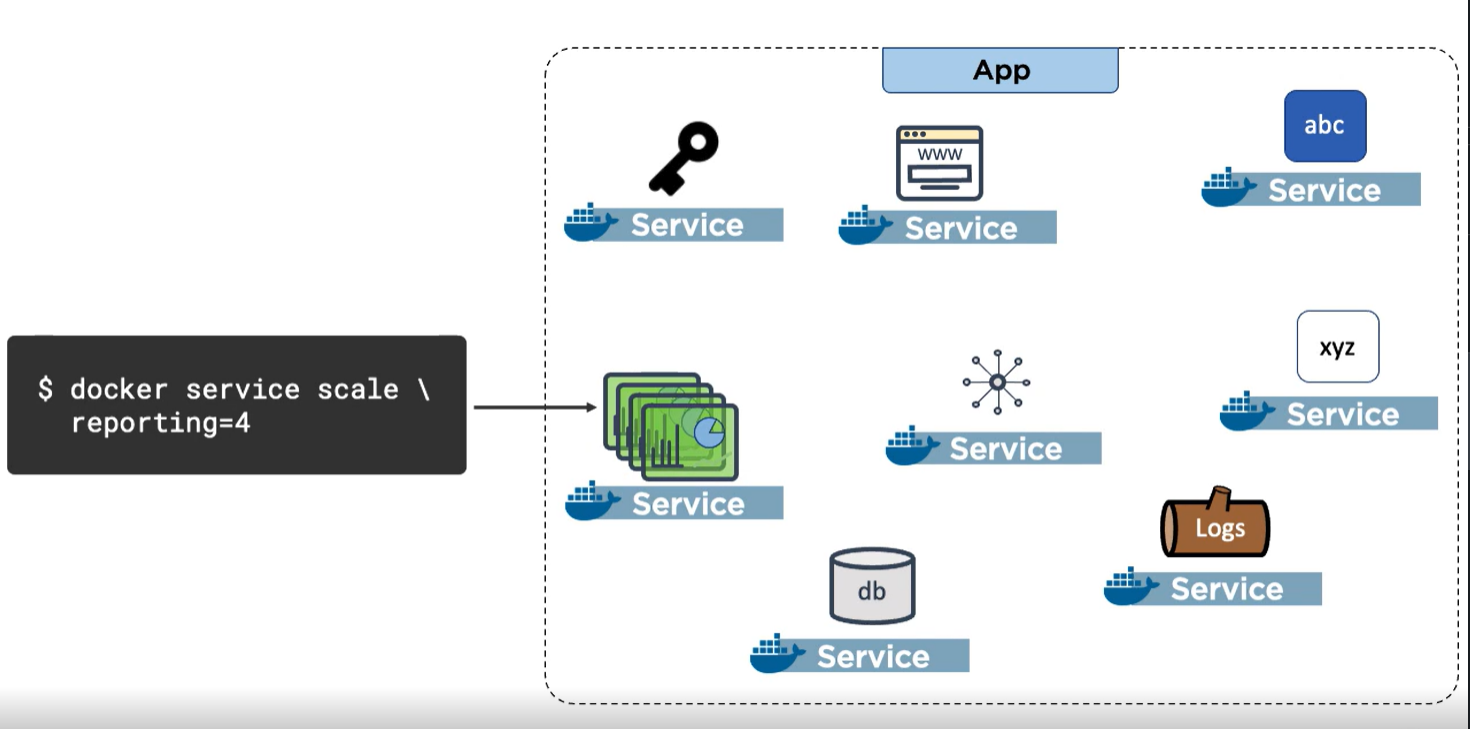
Docker swarm unlocks a some features like **service object** and **docker stack**.

Docker service object maps with an individual service in microservice app.



Each service in a microservice is implemented through individual docker service and we deploy and manipulate through each docker service object.

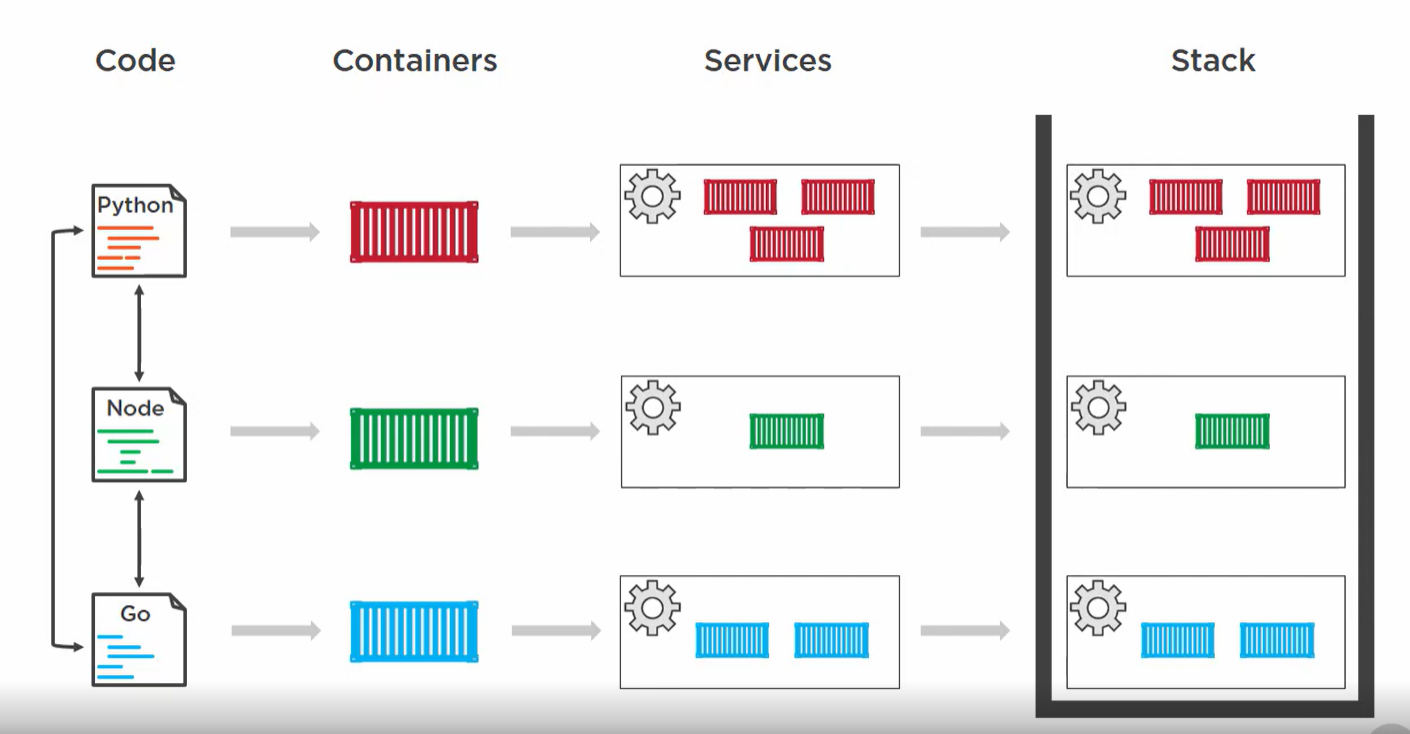
If we have to scale an individual microservice we do that by altering that docker service.



We have 8 microservices here so we will 8 docker service.

Multiple small services working together to make an app. Each of these micro-services has one service associated with a **service object** in a docker engine.

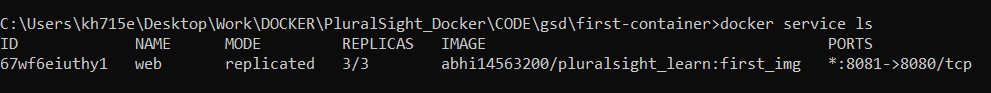
Managing bunch of services is done by **Stack.**

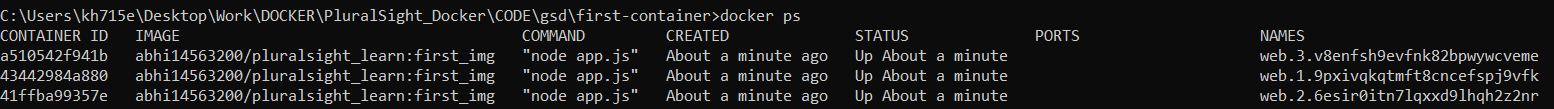


* Our application is made up of bunch of services that talks to each other.
* We create separate container for each service.
* But in order to scale them we deploy them as **Services.**
* Deploy and manage each service separately is not ideal. So we group service as a **Stack** (Highest layer of docker application service hierarchy).

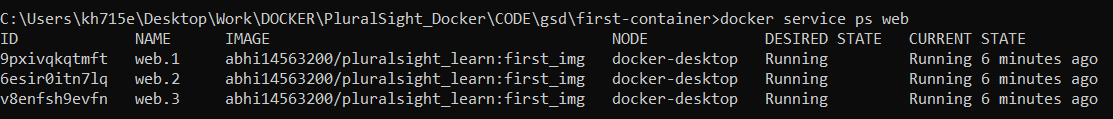
**Imperative way is to use docker services**

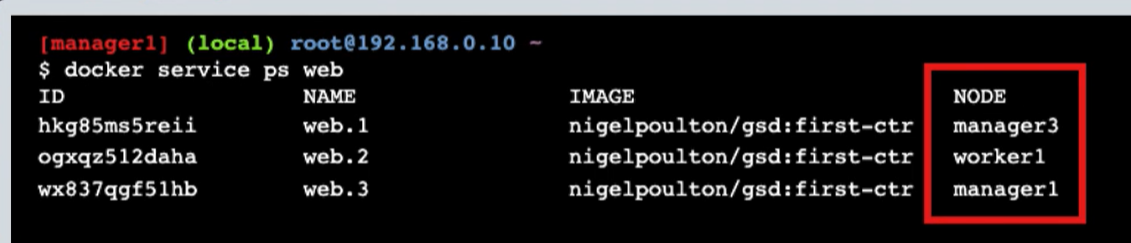
1. **Initialize the swarm** – docker swarm init
2. docker service create --name web -p 8081:8080 --replicas 3 abhi14563200/pluralsight\_learn:first\_img
3. Above command means we are asking to create 3 service replica (means 3 container) of a microservice.

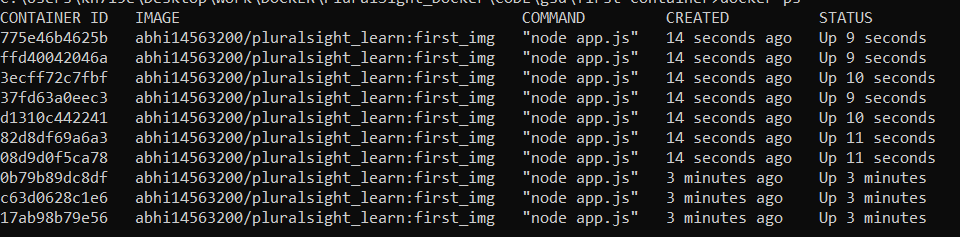




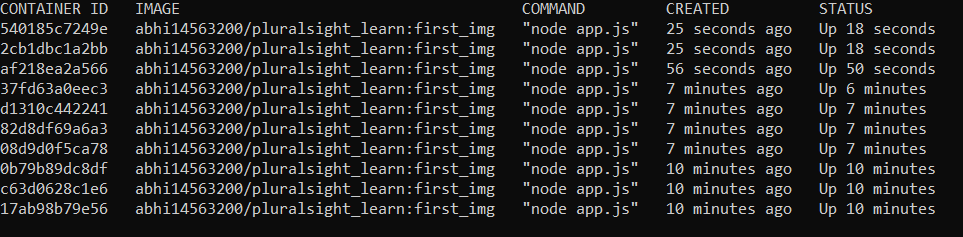
1. When we do docker service ps web



1. Here node is important as we are running on a single node of docker desktop then all replica is on same node but if we create on multi node swarm then container can be on different nodes.
2. In an actual multi node cluster replicas load balanced among all nodes in a cluster.
3. When we run the service in localhost we see every time we refresh the service is switch between 3 containers.
4. We can scale the service using **docker service scale web=10**
5. If we do **docker ps** then we have 10 containers.

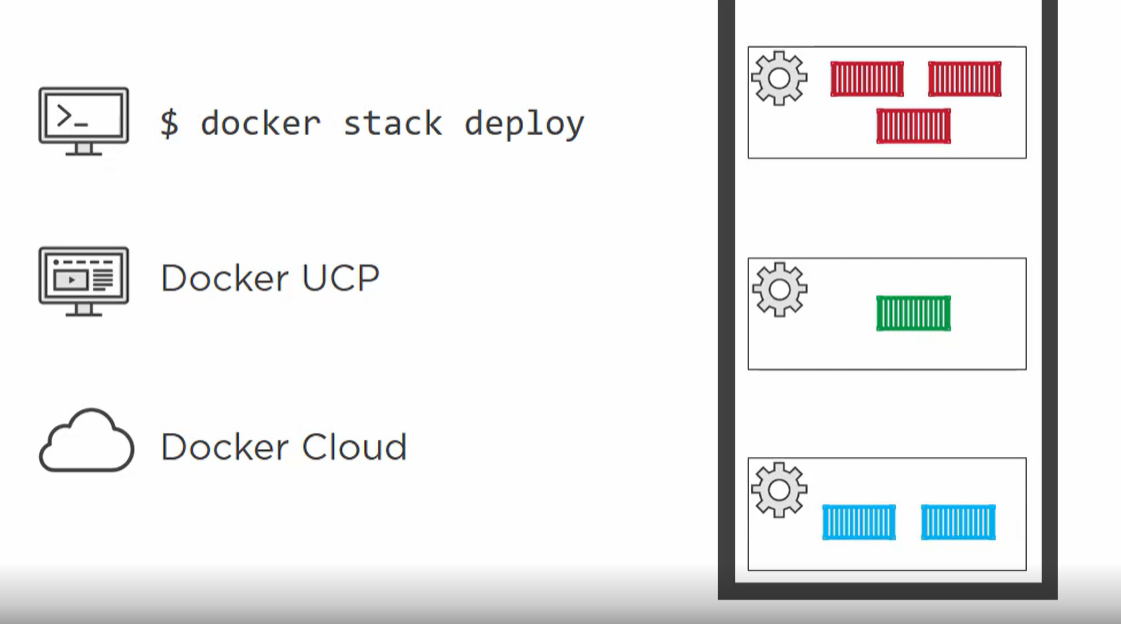


1. But if we go **side door** and destroy first 3 containers then docker will remove and then immediately create 3 new one



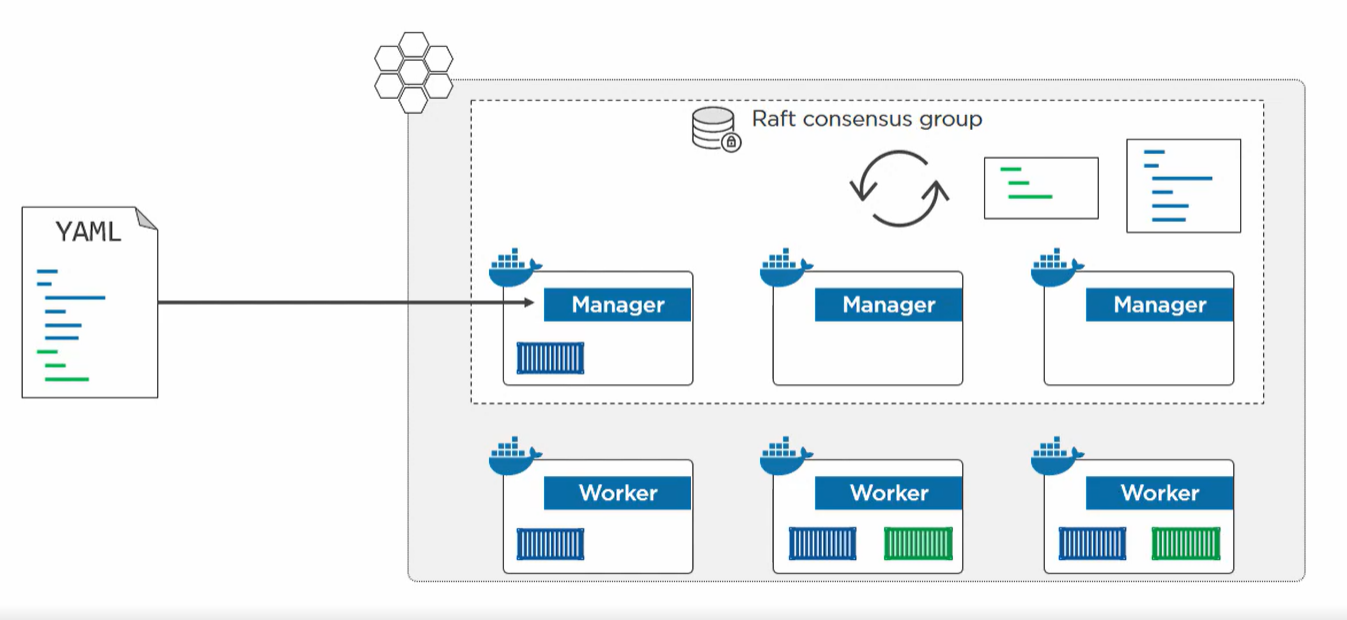
1. Docker knows we asked for 10 so it record that as our “**desired state**” and then we go side door and delete 3 of them (CRUDE WAY TO SIMULATE FAILURE)
2. So docker thought we desired 10 and due to some reason 3 is failed so it automatically created 3 new ones.
3. **A loop is constantly watching the state of the cluster and comparing the desired and observed state. It aims to equate desired and observed state every time. This is called ORCHESTRATION**

**STACKS**



**Stack file is basically a compose.yml** **file**.

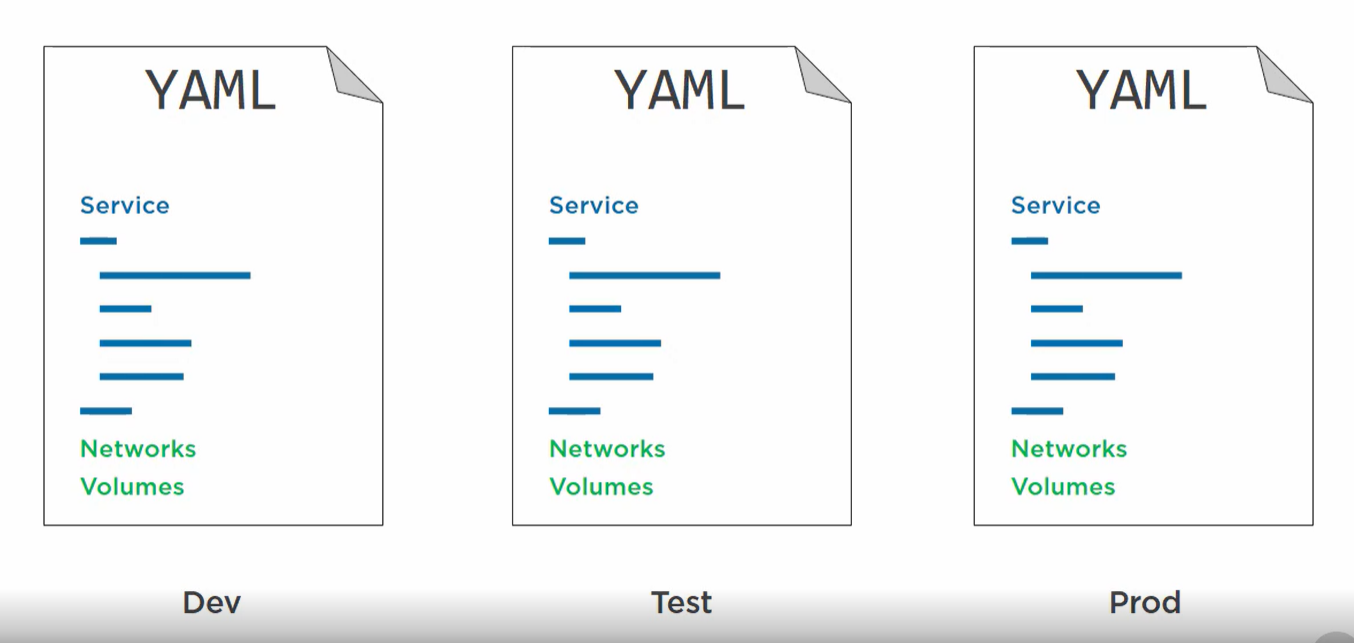
* In a stack file we define the services , replicas , network , volume etc.
* We feed all of them to a swarm. Swarm deploys it and manages it.
* Swarm records the specification of each service in a cluster.
* It then runs a reconciliation loop and watch the desired and actual state. IF any node fails and bring down replicas then it brings another one back.



**Stack file is a great way of documenting your application.**

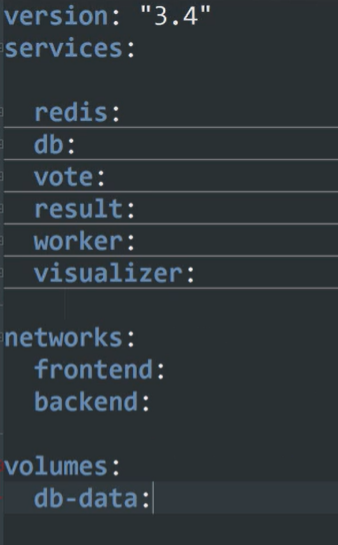
A developer define a the app in a stack file and hands over to Ops team. Ops team gets a great description of application what services, replicas, network and volumes to use.

We can have different version of stack file for dev, test and prod.



In different env we can have different no of replicas or a different secret.

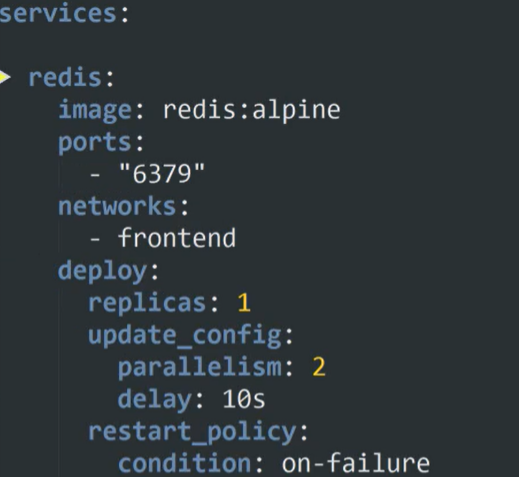
**Stack file are compose file but we need to call version: 3.4 cause deploy key is supported in version after 3.4**



**We have 4 top level key**

* Version
* Services
* Network
* Volumes

**Each service has its own JSON dictionary with its keys.**



**We have a redis service which will use**

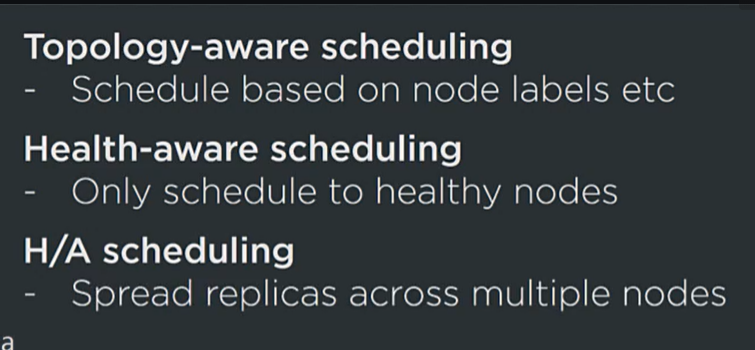
1. Image – redis
2. Port – within a container
3. Network – frontend
4. Deploy(actual stack stuff) –

* replicas
* parallelism – update 2 replicas
* condition – restart service on fail

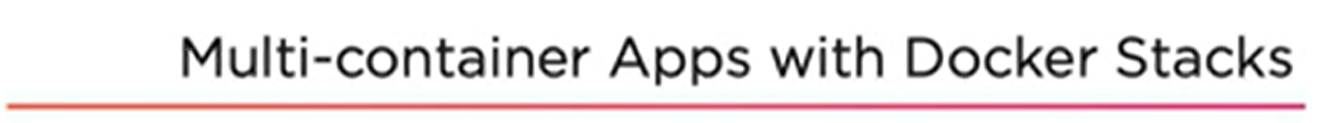


Placement policy – to run only on managers

Swarm does placement using below policy



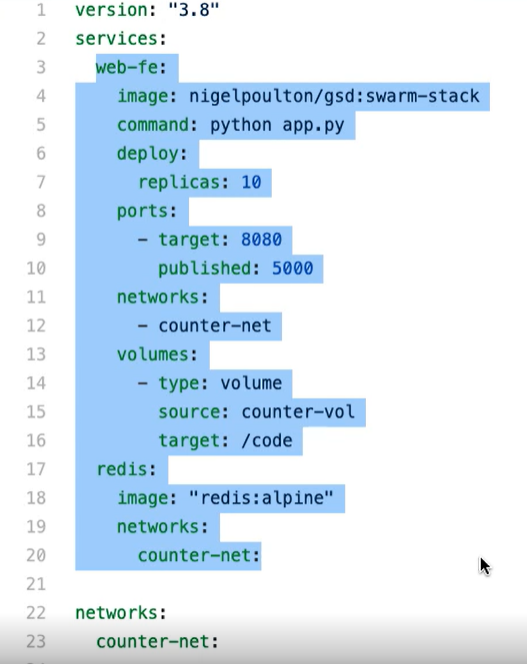
**DECLARATIVE WAY IS DOCKER STACK**



Declarative way means instead of us writing all those commands to scale and create service by our own. Instead we will create a configuration yml file and define the services.

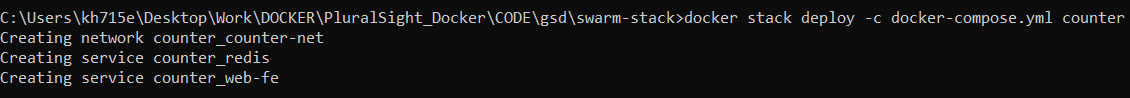
Advantage of yml file:

1. We have our apps documented. Benefit ops people.
2. For dev by just looking at yml we can get refresh of application.

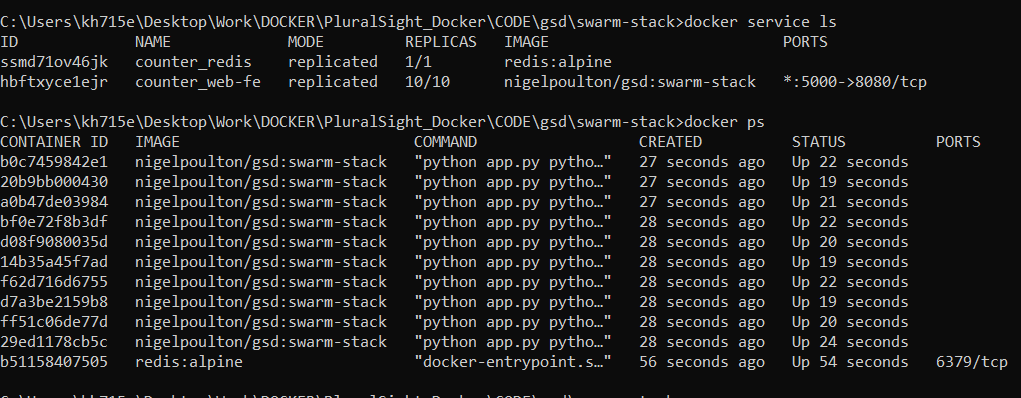


Same as earlier compose.yml file. We have 2 services one is the web frontend and other is redis cache. As we are running in docker swarm mode we call this app a stack.

* Stack on a swarm do not support building images on a fly like in compose so we have to build image first and then declare here.
* So once image is created we need to push to a registry so that every node in a cluster can access it.
* Now we can run the stack –
* **Docker stack deploy -c docker-compose.yml counter**
* -c = tells stack is deployed using compose.yml
* Counter = name of stack



So it creates a network and 2 services and 11 containers



**When we run this app using localhost:5000 we see the new request are mapped among 10 web containers.**

If we need to scale further we need to update the yml file. And run docker stack deploy again. So we need to keep yml file as part of our version control.