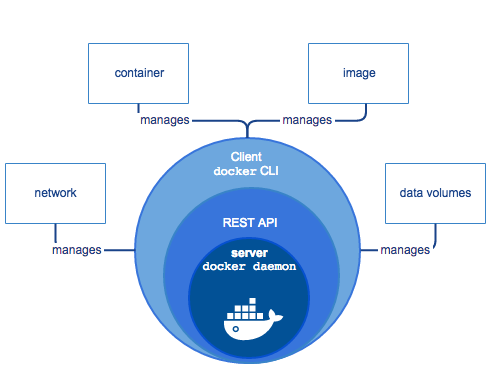
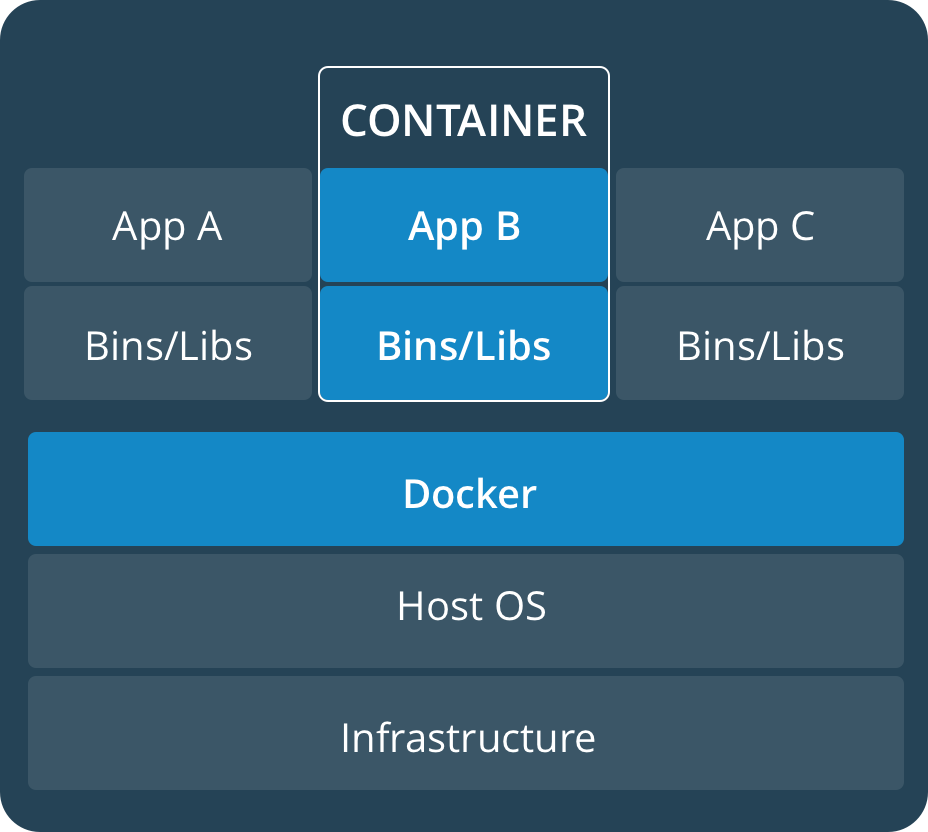
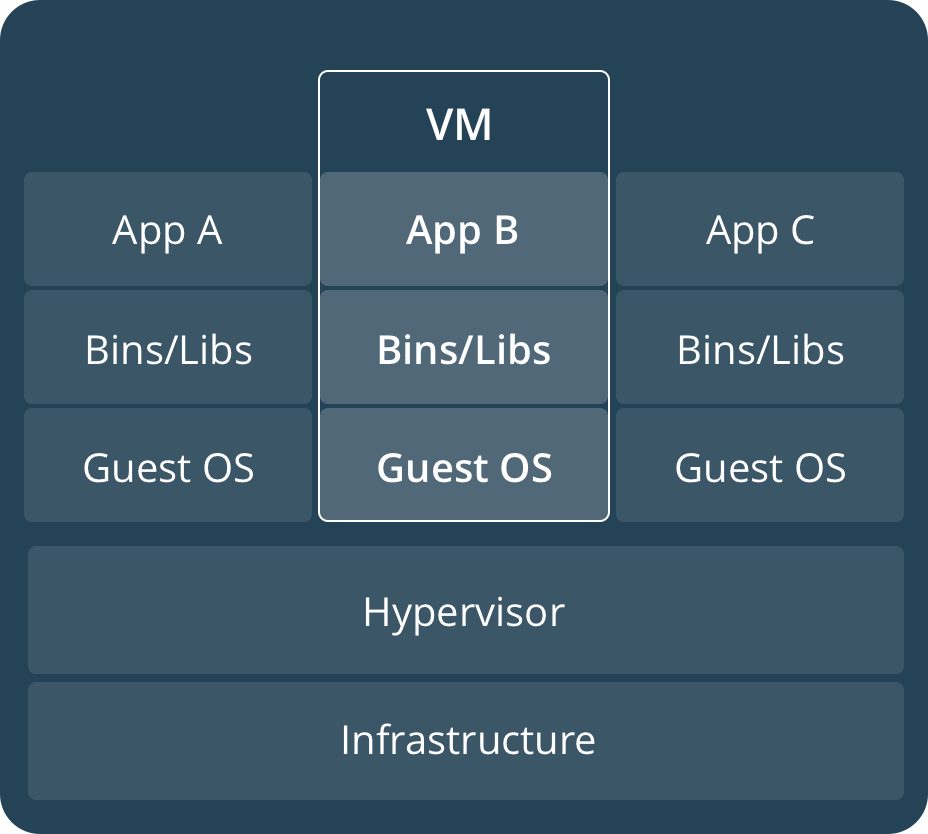
**Docker**

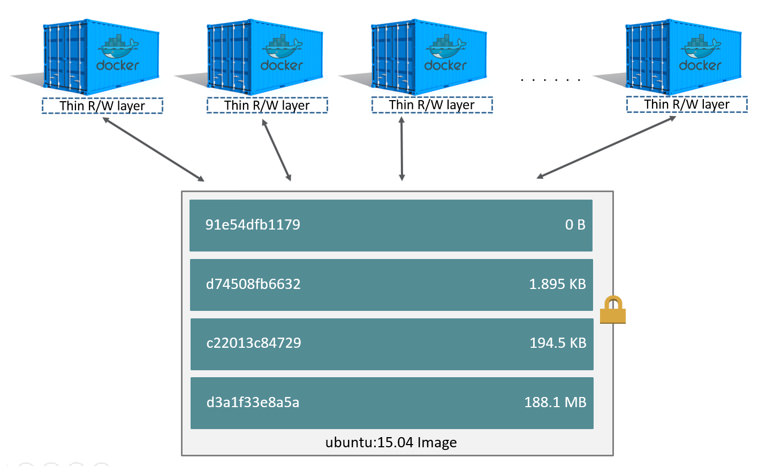
Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

Containers doesn’t have an OS to run the applications like Physical servers/VMs, then? It uses the lib/binr of the OS to run the application which makes you container lightweight.



****

****

****

Architecture Overview

Docker architecture:

* Client-server architecture
* Client talks to the Docker daemon
* The Docker daemon handles:
  + Building Running
  + Distributing
* Both communicate using a REST API:
  + UNIX sockets
  + Network interface

The Docker daemon (dockerd):

* Listens for Docker API requests and manages Docker objects:
  + Images
  + Containers
  + Networks
  + Volumes

The Docker client (docker):

* Is how users interact with Docker
* Sends commands to dockerd

Docker registries:

* Stores Docker images
* Public registry such as Docker Hub
* Let you run your own private registry

Docker objects:

* Images:
  + Read-only template with instructions for creating a Docker container
  + Image is based on another image
  + Create your own images
  + Use images published to a registry
  + Use a Docker file to build images
* Containers:
  + Runnable instance of an image
  + Connect a container to networks
  + Attach storage
  + Create a new image based on its current state
  + Isolated from other containers and the host machine
* Services
  + Scale containers across multiple Docker daemons
  + Docker Swarm
  + Define the desired state
  + Service is load-balanced

Docker Swarm:

* Multiple Docker daemons (Master and Workers)
* The daemons all communicate using the Docker API
* Supported in Docker 1.12 and higher

Running Containers

**docker container run -it --name <NAME> <IMAGE>:<TAG>**

Creating a container:

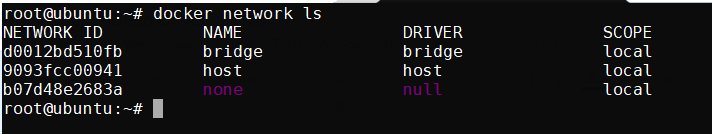
* CLI use for executing a command
* Docker client uses the appropriate API payload
* POSTs to the correct API endpoint
* Docker deamon receives instructions
* Docker deamon calls containerd to start a new container
* Docker daemon uses gRPC (a CRUD style API)
* containerd creates an OCI bundle from the Docker image
* Tells runc to create a container using the OCI bundle
* runc interfaces with the OS kernel to get the constructs needed to create a container
  + This includes namespaces, cgroups, etc.
* Container process starts as a child process
* runc exits once the container starts
* Process is complete, and container is running

**Docker network:**

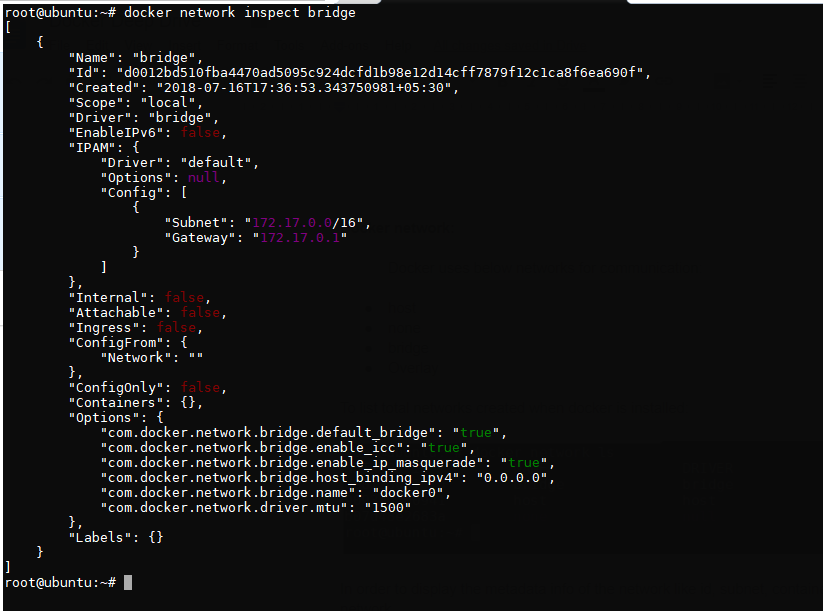
Docker uses below networks for communication:

* host
* none
* bridge
* Overlay

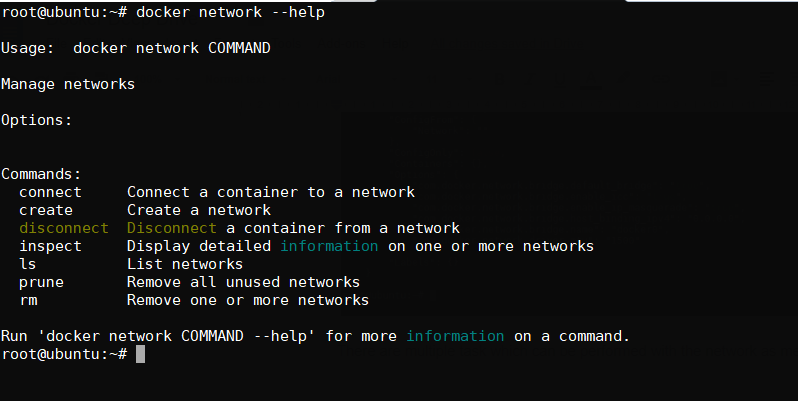
To list total networks created when docker is installed:



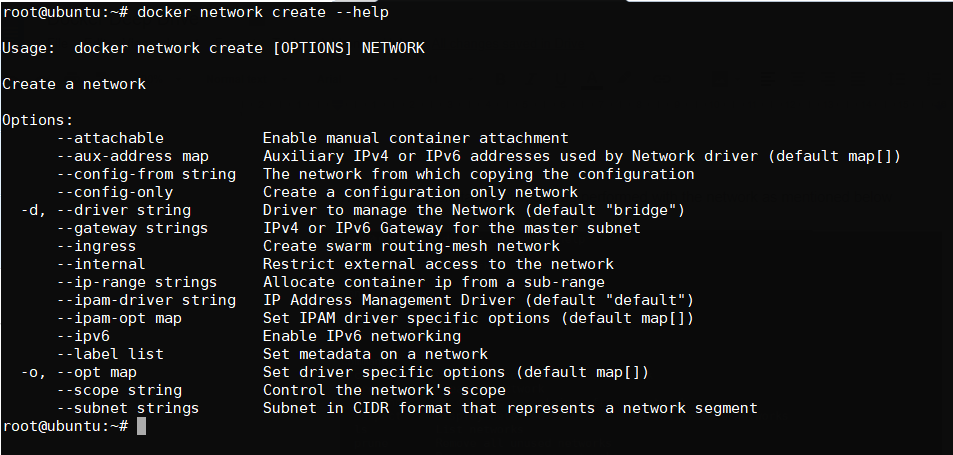
In order to display the metadata info of the network like id, subnet, containers binded to the network



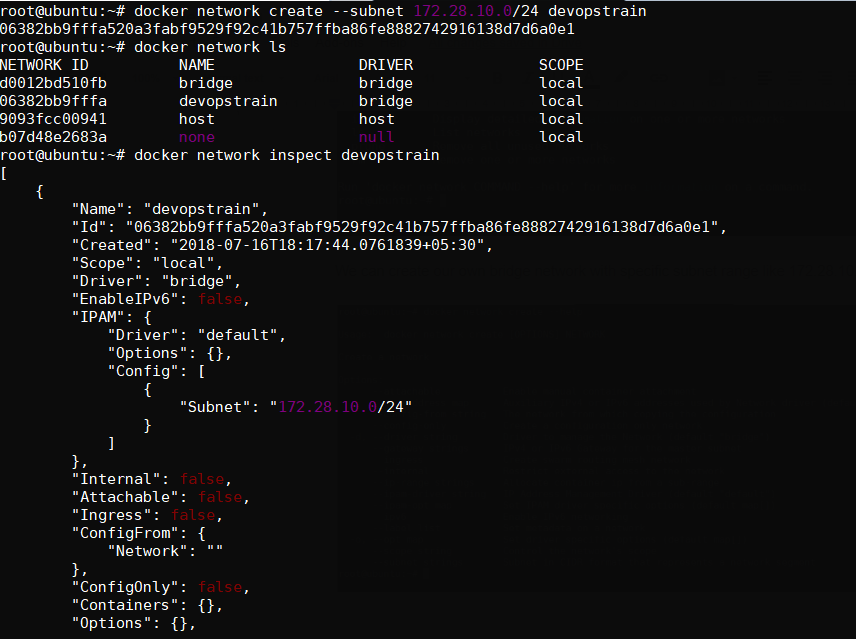
There are multiple task which can be performed with the network as mentioned below



We can create our own bridge network with specific subnet range like 172.28.10.0/24

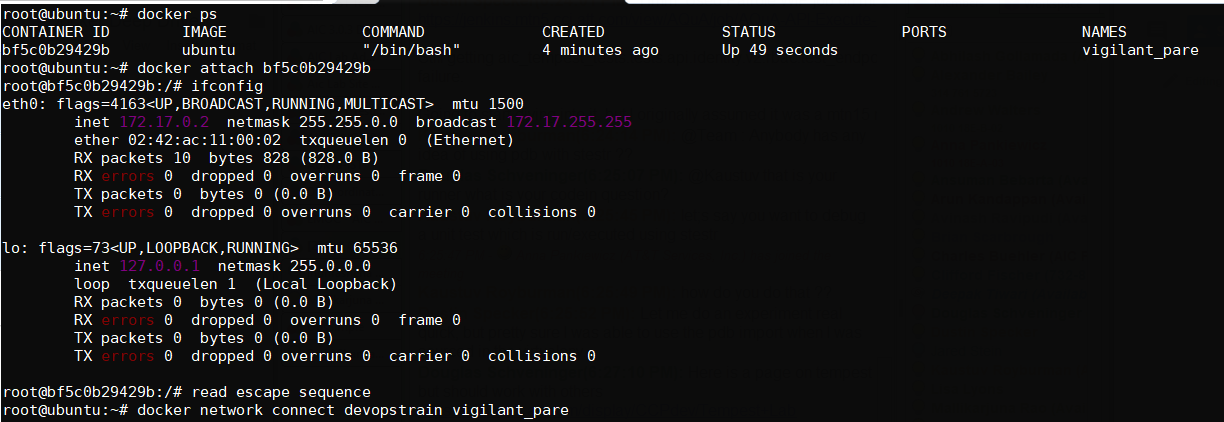


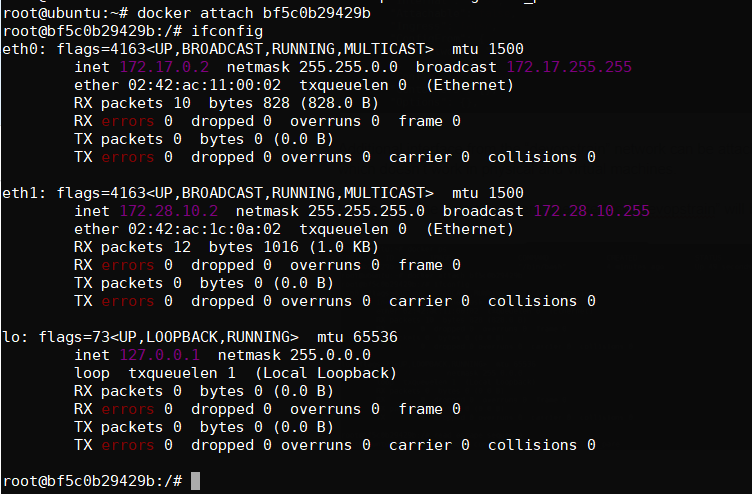
Here we are not passing the driver type bcz we are creating bridge network by default and gateway is optional which is not required in this case.



Additional interface from the “devopstrain” network can be attached to the running container, which doesn’t work in physical and virtual machines.

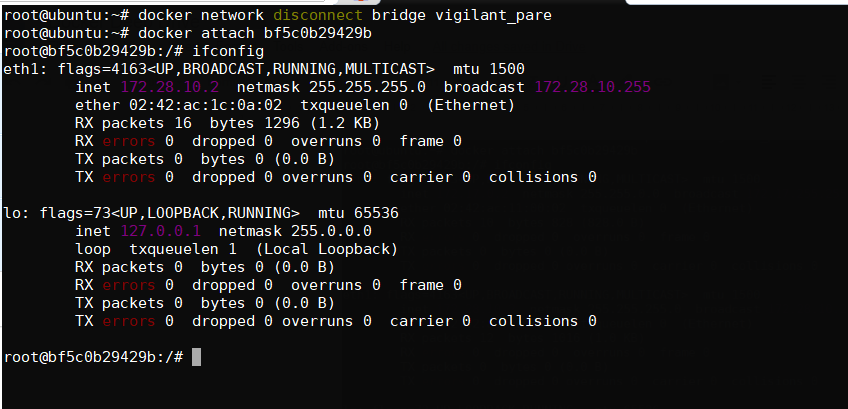
Let us take one scenario where one interface “devopstrain” will be attached to the conatiner:



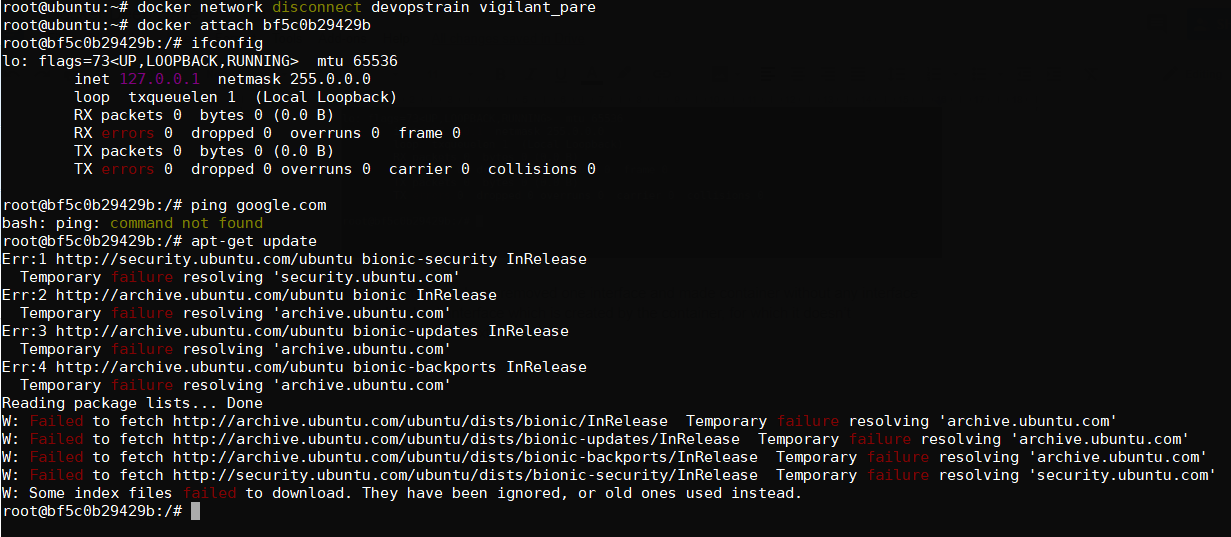


How to remove an interface from the running container?

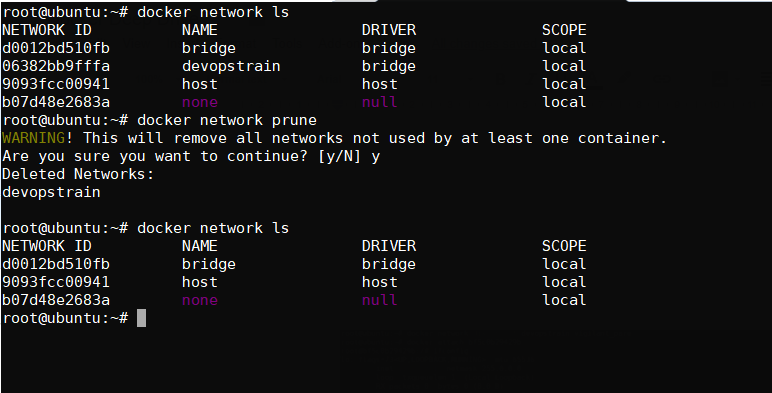
In the below example we have removed the “bridge network” from the container



Below example we have removed one interface and made container without any interface except loopback interface which is created by the container, for which it doesn’t communicate to the external world

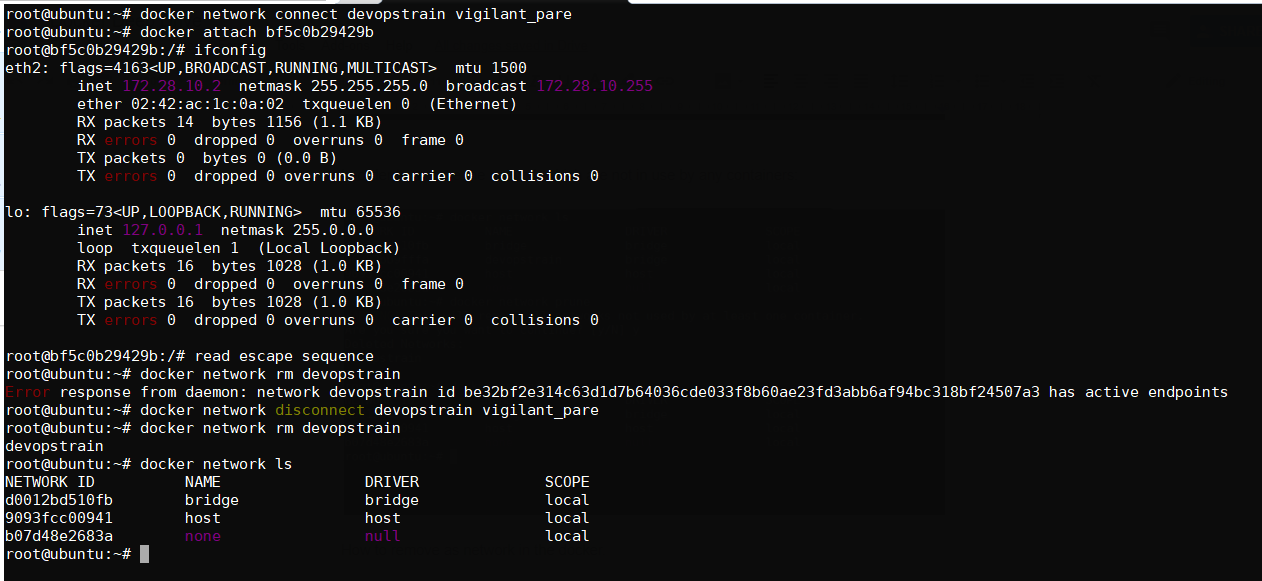


In order to remove the network which are not in use by any containers:

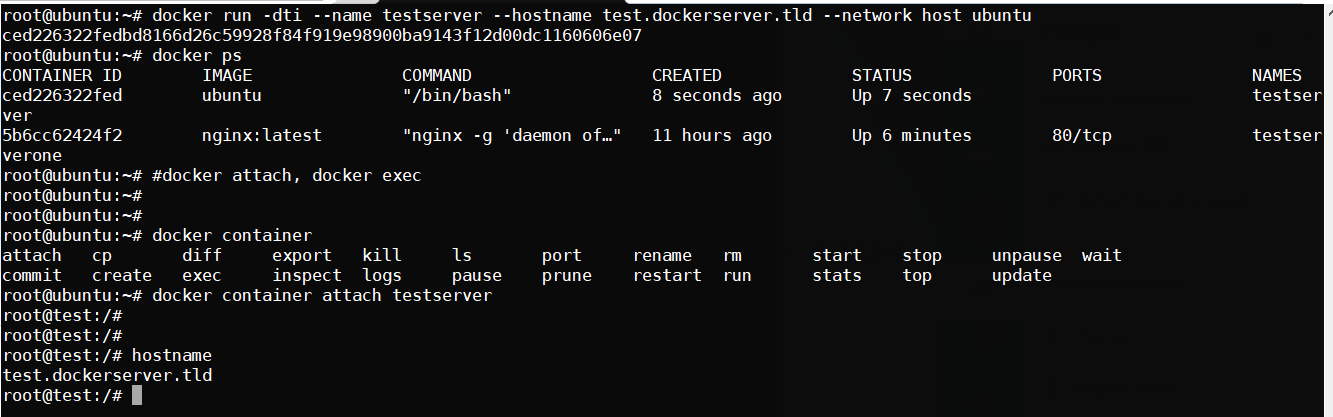


How to remove as network in the docker.

Note: Before we delete the network, we need to ensure that none of the container is running on that network. (Either stop the container and disconnect the network)



Host network can be assigned to the container, which is mentioned below and we can access the application with the public ip address it self.



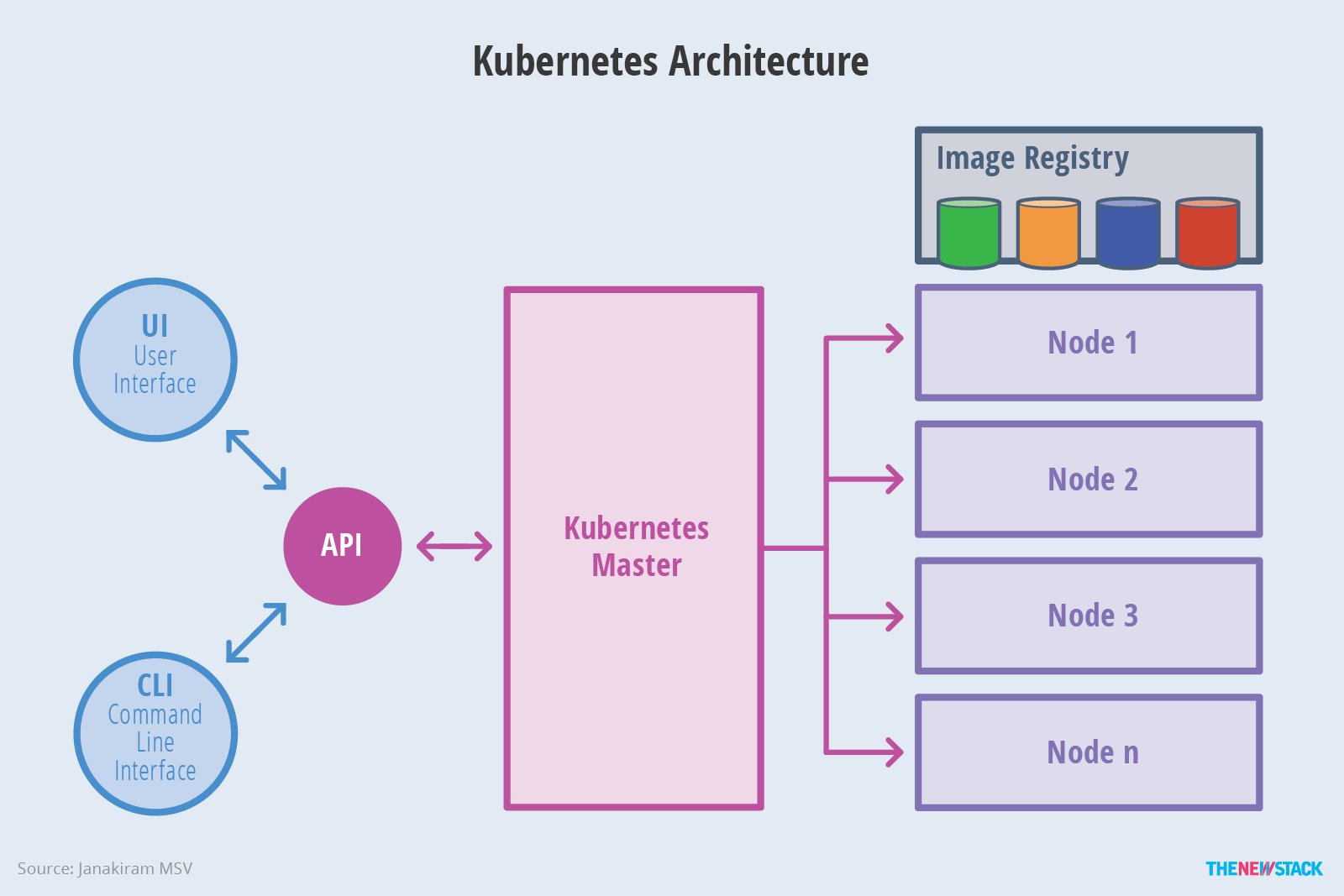
Note: overlay network will be covered in docker swarm or Kubernetes cluster

**Kubernetes:**

-----------------

Kubernetes is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation

Kubernetes works in server-client setup, where it has a master providing centralized control for a number of minions. We will be deploying a Kubernetes master with two minions in this scenario.



**Kubernetes has several components:**

------------------------------------------------------

**etcd** - A highly available key-value store for shared configuration and service discovery.

**flannel/calico** - An etcd backed network fabric for containers.

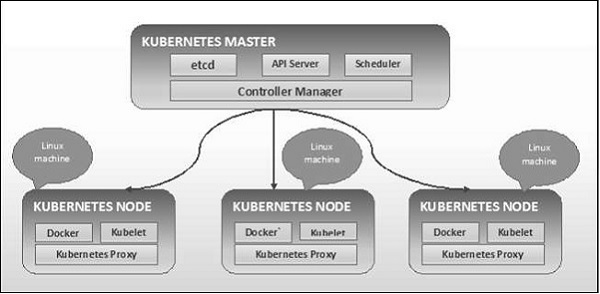
**kube-apiserver** - Provides the API for Kubernetes orchestration.

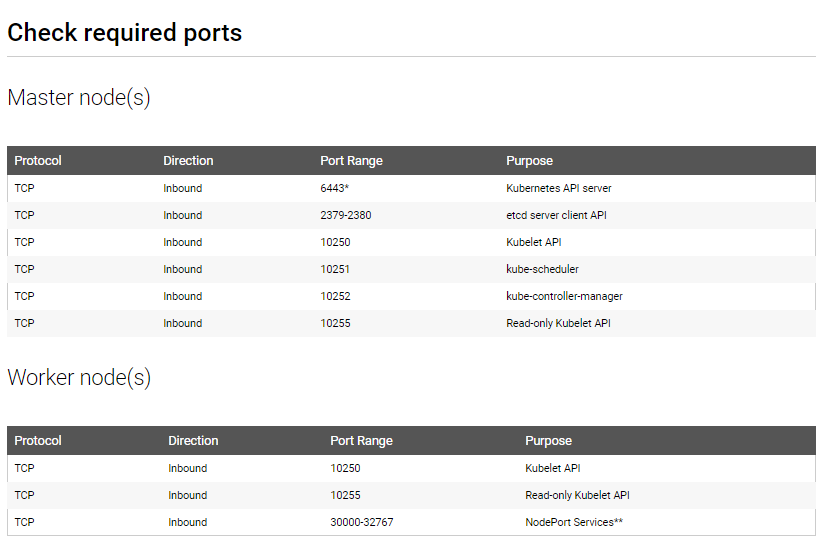
**kube-controller-manager** - Enforces Kubernetes services.

**kube-scheduler** - Schedules containers on hosts.

**kubelet** - Processes a container manifest so the containers are launched according to how they are described.

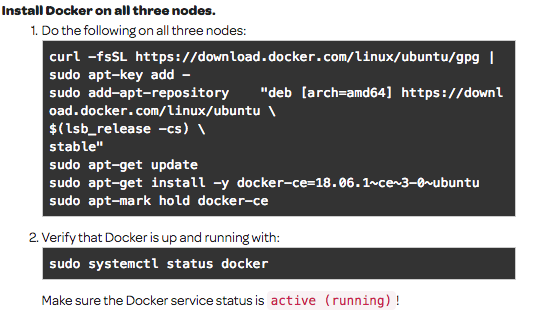
**kube-proxy** - Provides network proxy services.

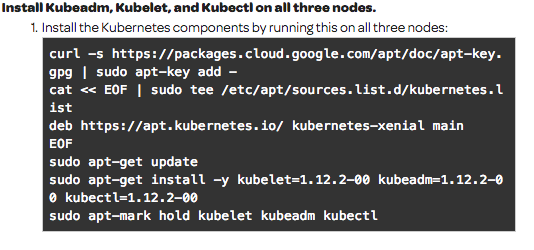


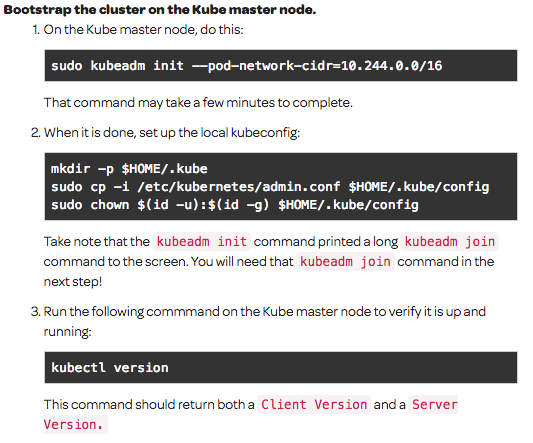


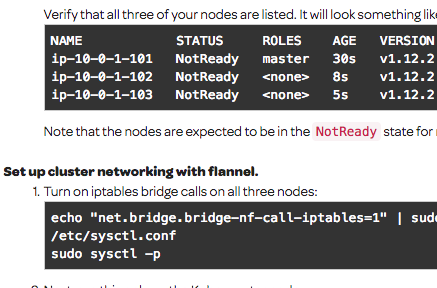
**Prerequisites for Kubernetes Cluster:**

On Ubuntu 16.04, Install docker version 1.12, 1.13 with native ubuntu repo as mentioned below:

****

****

****

****