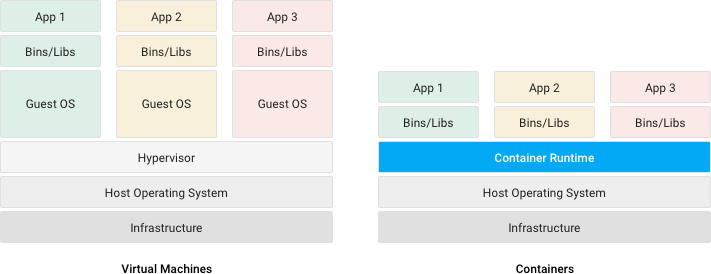
What are containers?

Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer’s personal laptop. Containerization provides a clean separation of concerns, as developers focus on their application logic and dependencies, while IT operations teams can focus on deployment and management without bothering with application details such as specific software versions and configurations specific to the app.

For those coming from virtualized environments, containers are often compared with virtual machines (VMs). You might already be familiar with VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with virtualized access to the underlying hardware. Like virtual machines, containers allow you to package your application together with libraries and other dependencies, providing isolated environments for running your software services. As you’ll see below however, the similarities end here as containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.



Why Containers?

Instead of virtualizing the hardware stack as with the virtual machines approach, containers virtualize at the operating system level, with multiple containers running atop the OS kernel directly. This means that containers are far more lightweight: they share the OS kernel, start much faster, and use a fraction of the memory compared to booting an entire OS.

There are many container formats available. Docker is a popular, open-source container format that is supported on Google Cloud Platform and by Google Kubernetes Engine.

Consistent Environment

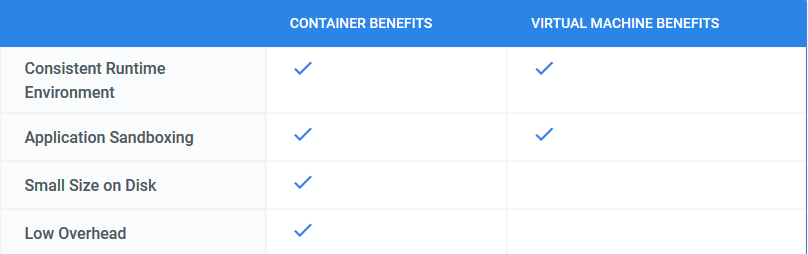
Containers give developers the ability to create predictable environments that are isolated from other applications. Containers can also include software dependencies needed by the application, such as specific versions of programming language runtimes and other software libraries. From the developer’s perspective, all this is guaranteed to be consistent no matter where the application is ultimately deployed. All this translates to productivity: developers and IT Ops teams spend less time debugging and diagnosing differences in environments, and more time shipping new functionality for users. And it means fewer bugs since developers can now make assumptions in dev and test environments they can be sure will hold true in production.

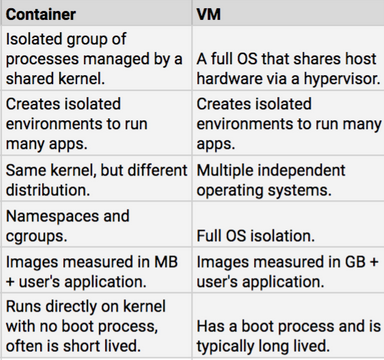
Run Anywhere

Containers are able to run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal; on a developer’s machine or in data centers on-premises; and of course, in the public cloud. The widespread popularity of the Docker image format for containers further helps with portability. Wherever you want to run your software, you can use containers.

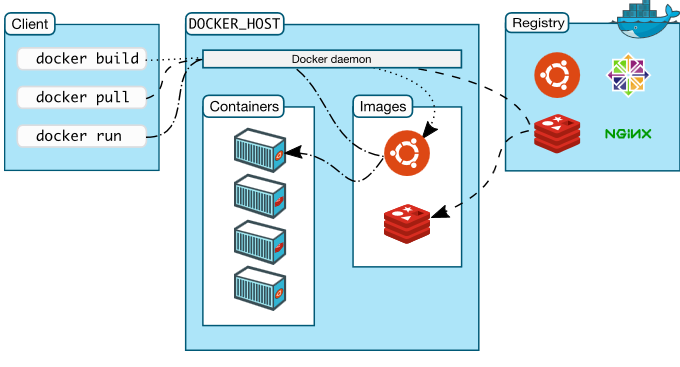
Isolation

Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications.





Docker Architecture



Docker hub is nothing but the remote registry to store images (public/private) if user want local one then they have to user registory2 where you can store their own images (private).

Example : Openjdk public image -

https://hub.docker.com/\_/openjdk

Just run - docker pull openjdk

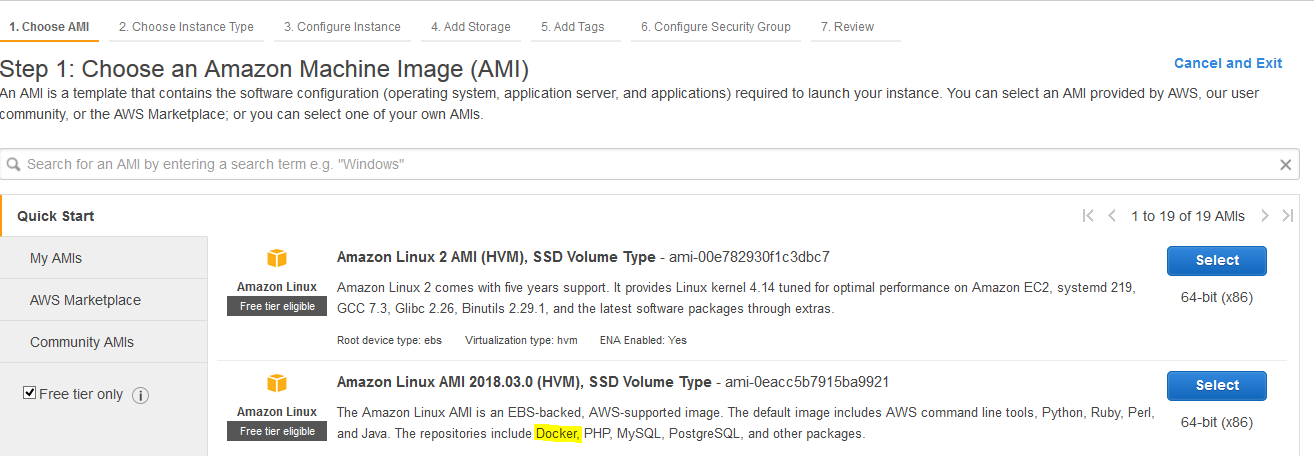
Here we are pulling this image and run as container where no need to install anyting.

- Every container have its own hierarchy .

**DOCKER ON AWS**

1. Login on your aws account and select free tier AMI with docker.

2. Select all default setting and until security group.



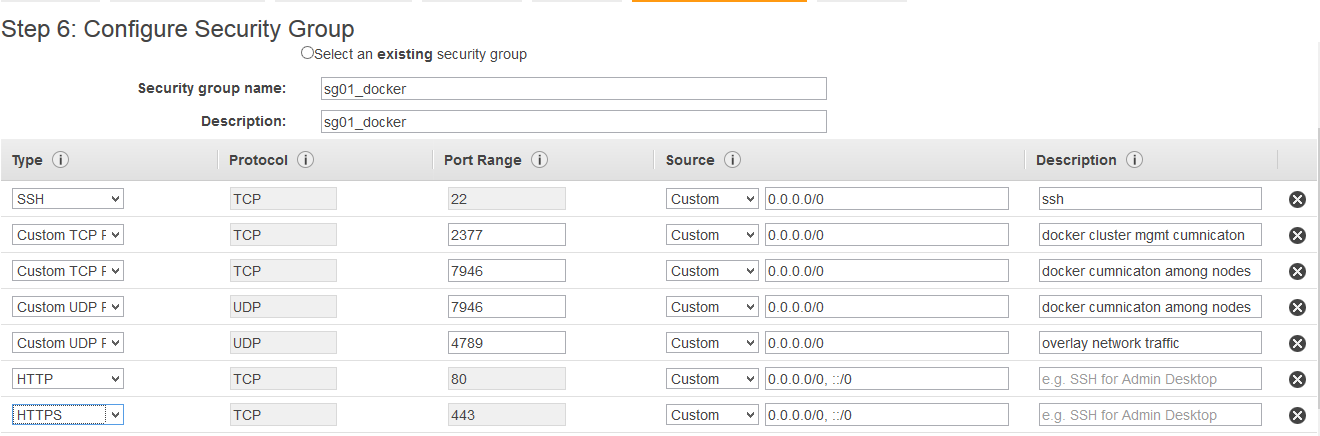
3. Create security group -

The following ports must be available. On some systems, these ports are open by default.

TCP port 2377 for cluster management communications

TCP and UDP port 7946 for communication among nodes

UDP port 4789 for overlay network traffic



https://docs.docker.com/engine/swarm/swarm-tutorial/

3. Select key and launch the instance.

If its your new setup then create new key and download it. To use these keys on putty have to convert from .pem to .ppk.

a. Download Putty and puttygen.

b. Use puttygen to convert .PEM file to .PPK file.

c. Start puttygen and select “Load”

d. Select your .PEM file.

e. Putty will convert the .PEM format to .PPK format.

=====================================================================================

1. **Update installed package on you instance.**

sudo yum update -y



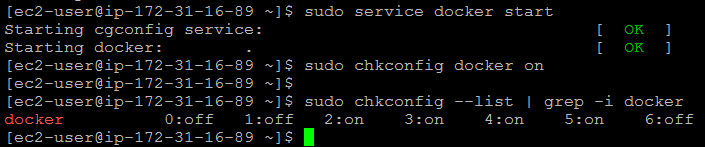
1. **Install docker**

sudo yum install docker -y



1. **Start docker services**

sudo service docker start; sudo chkconfig docker on; sudo chkconfig --list | grep -i docker

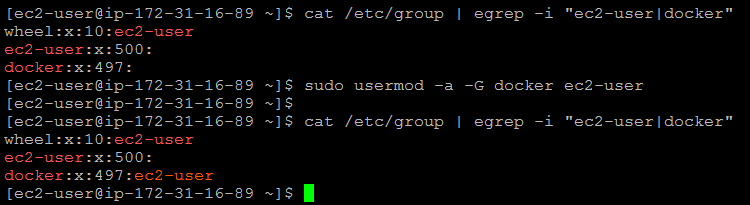


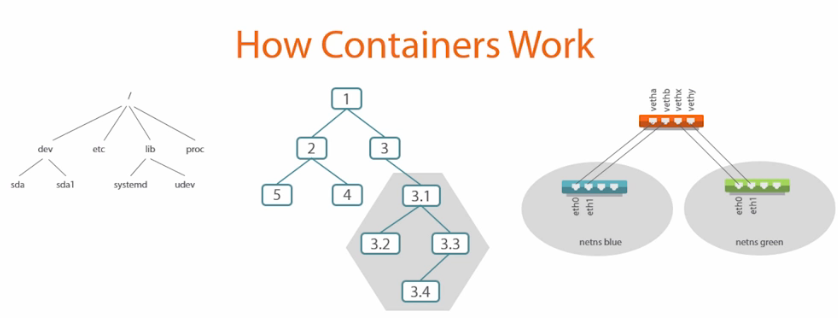
1. **Add ec2-user in docker group so we can run docker commands without sudo.**

cat /etc/group | egrep -i "ec2-user|docker"

sudo usermod -a -G docker ec2-user

cat /etc/group | egrep -i "ec2-user|docker"





Simpleway:

1. Containers are isolated but share OS and appropriate bin/libraries but vm can't.

2. Docker hub is nothing but the remote registry to store images if user want local one then they have to user registory2 where you can store their own images.

3.

Links and details -

1. Create local repository

https://code-maze.com/docker-hub-vs-creating-docker-registry/

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-18-04