Docker:

* Docker is a tool that encapsulates the process of creation of distributable artifact and deploying it to scale it into any environment.
* Benefits of docket workflow.
  + It makes architectural decisions simpler, because all the applications look the same from the hosting machine.
  + Docker wraps up all the requirements in a single file that is defined in a single file.
  + Bundles the application software and the OS requirements in a single package.
  + Same packaged artifact can be tested and delivered across all environments.
  + Unlike VM’s which use a dedicated amount of the resources from the host machine. Docker containers work like a process and talk to the Linux kernel, and use up the resources until the system quota is reached.
* Architecture of docker is simple client/server model, with only one executable serving on both ends depending upon on how the docker command is invoked.
  + **Client/Server Model:** Docker consists primarily of two parts docker server and docker client and optional third component called as registry which consists of image files.
  + Server does the ongoing work of running and maintaining containers and client is used to tell the server on what needs to be done.
  + Docker daemon can run on any number of servers, clients drive all the communication but docker servers talk directly to the registry whenever instructed by the client.

.

Docker Registry

Docker Client

Docker Server

* Docker is a little different from traditional client server model, instead of having different components for both client and server; it uses the same binary executable for both.
* **Docker Command line tool** is the interface available to users to interact with the docker engine and containers. Some of the tasks that can be achieved by docker command line are creating containers, starting containers; retrieve docker logs from remote server, start command line shell on a container running on remote server.
* **Container Networking:** Docker containers behave like hosts on a private network. Docker server behaves like a host and containers are like clients behind it. The implementation is done in a way that each container has its own virtual Ethernet interface connected to Docker Bridge. Each container has its own IP address allocated and docker lets you bind the ports of the containers to the host so the outside world can talk to them. Apart from the default Docker Virtual Bridge, there are many other ways in which the docker networking can be achieved.
* **Installation Of Docker:** the best way to install docker on a machine is going through the official documentation available on docker pages  
  **https://docs.docker.com/install/linux/docker-ce/centos/** Alternative to install on linux environments , you can use the script available onsudo curl -sSL https://get.docker.io/ | sh
  + **Once docker is installed**check the versions by running :

docker –version

docker info

* + - restart docker service : sudo service docker restart

|  |
| --- |
| **Application** |
| **Dependencies** |
| **Software** |
| **Base Image** |
| **Kernel** |

* **Docker Images:** docker image is a collection of files that make up and executable software application. Collection includes the application plus the libraries and the dependencies required to run an application. Files are in read only mode, which is they are in a layer, if we wish to alter or modify any existing files a new layer has to be added on top existing one. Below diagram represents the layered architecture.  
    
  + Base image is nothing but minimalistic replica of an OS i.e. all the required libraries and files for running an application. A base image does not have a kernel. An image continuously gets enriched by addition of modules (in the form of layers).
  + Any docker image comes from a base image. A new Image can be created by modifying existing images.
* **Docker Containers:**
  + Containers are just wrappers around UNIX processes and not as standalone virtual machines.
  + Containers are light weight, creating a container takes very little space, because the container only has reference for a layered file system and metadata of the configuration.
  + Containers are isolated from each other. Container share resources from the host system as per the requirement.
  + Containers are created with the images and Images define the behavior of the container. When a container is spun a writable layer is added on top of the image.
* **Docker Registry:**
  + Docker registry is a storage place where the images are stored where they can be accessed publicly or privately.
* **Working With Docker:**
  + Once docker is installed, we are good to start working with the images and the containers and dwell into the land of awesomeness.
  + For starters the Docker commands work in the syntax as follows  
    docker <command> <sub\_command> - <options\_for\_sub\_command>
  + Type in docker on the cli and you will see list of commands that can be used, you will two sections stating management commands and commands, so to end the confusion the start why did I put in the syntax as command and sub-command, it is because this helped me in the long run.
  + What you see in the commands on your screen are some subcommands that can be used without invoking any management command. For example

docker ps and docker container ls work in the same way of displaying the running containers.

* + **Image commands:** whenever we request a container to be spun up , the docker client looks up for the base image being used on the local machine , if it’s not there it goes and pulls it from the docker registry.
    - Pulling Image: Below command pulls down the image from central repository and saves it on the local.  
      docker pull <image\_name>
    - Try and look up the subcommands associated with image, type in docker <main\_command> --help (this way you don’t have to memorize any of the commands, just have in mind what operation you want to perform and call docker to help with the syntax)  
      docker image -–help
    - As you can see from the output of the above command, our image command also has a sub command pull which works in similar fashion as docker pull. Instead of getting confused pick a way you would like to use. I prefer main\_command and sub\_command makes my life easier.
    - To list available images on the local type in  
      docker image ls or docker images
  + **Container Commands:** like we did in the case of image, we can check for the sub commands available to use with containers by using
    - docker container -–help
    - the output of the above should look like

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu latest f975c5035748 4 weeks ago 112MB

centos latest 2d194b392dd1 4 weeks ago 119MB

REPOSITORY – this is the name of the image.

TAG – TAG of the IMAGE given at the time of creating the image, this can also be the version

IMAGE ID – Unique Identifier for the Image

CREATED – Time line at which the image was created.

SIZE – Size of Image on the disk