
Tech Data
INFINITY


Code
Build
Test
Release
Deploy
Operate
Monitor
DEV
OPS

Docker


Tech Data
INFINITY

5. Docker

What are Microservices?

Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.

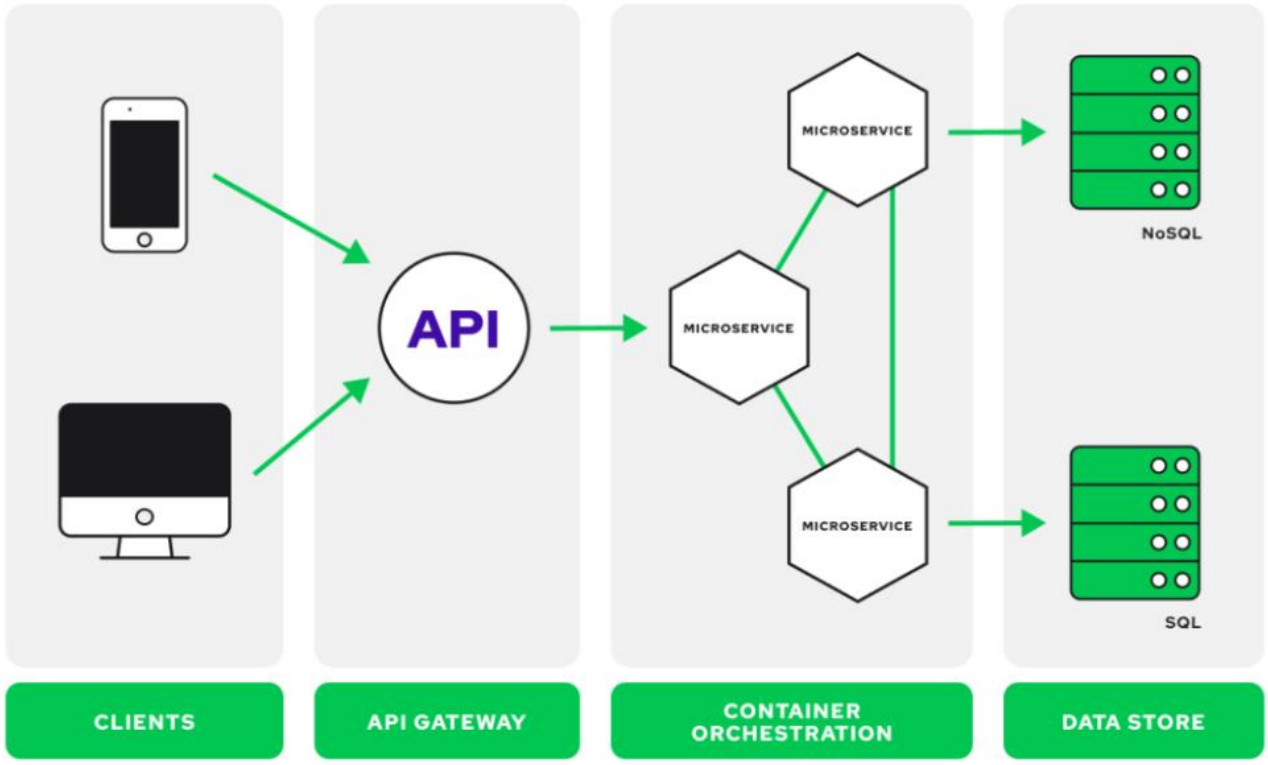
Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features. Microservices are:-

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

Benefits of Microservices-

1. **Agility** – Microservices foster an organization of small, independent teams that take ownership of their services.
2. **Flexible Scaling** – Microservices allow each service to be independently scaled to meet demand for the application feature it supports.
3. **Easy Deployment** – Microservices enable continuous integration and continuous delivery, making it easy to try out new ideas and to roll back if something doesn't work.
4. **Technological Freedom** – Microservices architectures don't follow a "one size fits all" approach. Teams have the freedom to choose the best tool to solve their specific problems.
5. **Reusable code** – Dividing software into small, well-defined modules enables teams to use functions for multiple purposes. A service written for a certain function can be used as a building block for another feature.
6. **Resilience** – Service independence increases an application's resistance to failure. In a monolithic architecture, if a single component fails, it can cause the entire application to fail.

TechData-Infinity-Devops with MultiCloud



What is Virtualisation?

- Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware.
- It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments.
- Virtualization is a technique of importing a Guest OS on the top of the host OS.
- This technique was a revelation at the beginning as it allowed developers to run multiple OS in different VMs on the same physical host.

Benefits of Virtualisation

1. More flexible and efficient allocation of resources.
2. Enhance development productivity.
3. It lowers the cost of IT infrastructure.
4. Remote access and rapid scalability.
5. High availability and disaster recovery.
6. Pay peruse of the IT infrastructure on demand.
7. Enables running multiple operating systems.

Disadvantages of Hypervisor:

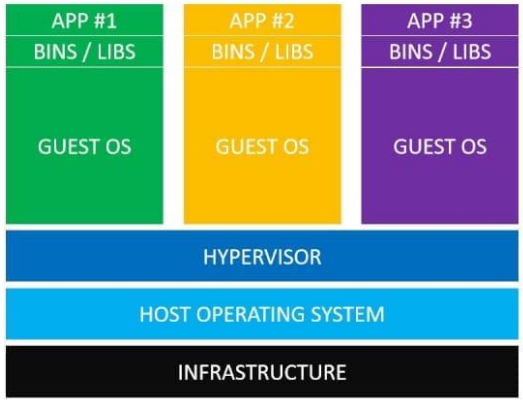
- Running multiple VMs leads to unstable performance.
- Hypervisors are not as efficient as the host OS.
- Boot up process is long and takes time.

Containerization:

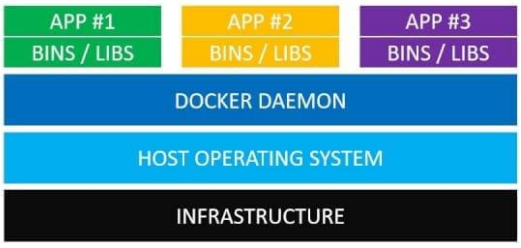
- Containerization is a form of virtualization where applications run in isolated user spaces, called containers, while using the same shared operating system (OS).
- A container is essentially a fully packaged and portable computing environment:
- Everything an application needs to run—its binaries, libraries, configuration files, and dependencies—is encapsulated and isolated in its container.
- The container itself is abstracted away from the host OS, with only limited access to underlying resources—much like a lightweight virtual machine (VM).

What is Docker?

- Docker is an open source centralized platform designed to create deploy and run actions.
- Docker uses container on the host OS to run applications. It allows applications to use the same Linux Kernel as a system on the host computer rather than creating a whole virtual OS.
- We can install docker on any OS but docker engine runs natively on Linux distribution.
- Docker is a tool that performs OS level virtualization also known as containerization.
- Before docker many users faces the problem that a particular code is running in developers system but not in the uses system.
- Docker is a set of platform as a service that uses OS level virtualization where VMware uses hardware level virtualization.



Virtual Machines



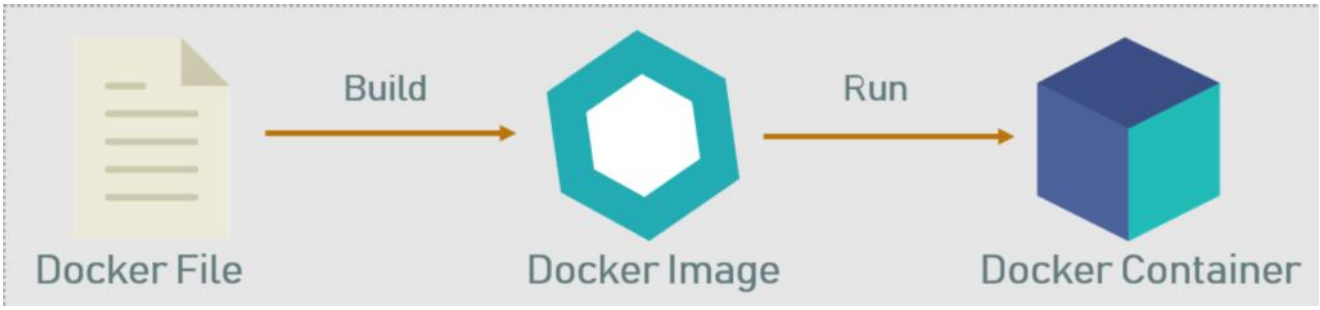
Docker Containers

Advantages of Docker

- No pre allocation of RAM.
- CI efficiency- docker enables you to build a container image and use that same image across every step of the deployment process.
- Less cost.
- It is light weight.
- It can run on physical H/W virtual H/W or on cloud.
- You can reuse the image.
- It took very less time to create container.

Disadvantages of Docker

- Docker is not a good solution for application that requires rich GUI.
- Difficult to manage large amount of containers
- Docker does not provide cross platform compatibility means if and application is designed to run in docker container on Windows than it can't run on Linux or vice versa.
- Docker is suitable when the developer OS and testing OS are same if the OS is different we should use VM.
- No solution for data recovery and backup.



Docker Architecture:

Docker Daemon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
- A daemon can also communicate with other daemons to manage Docker services.
- Docker daemon runs on the host OS.
- It is responsible for running containers to manage docker services.
- Docker Daemon can communicate with other daemons.

Docker client

- Docker users can interact with docker daemon through a client (CLI).
- Docker client uses commands and rest API to communicate with the doctor daemon
- When a client runs any server command on the doctor client terminal, the client terminal sends this docker commands to the docker daemon.
- It is possible for docker client to communicate with more than one day one daemon.
- The Docker client (docker) is the primary way that many Docker users interact with Docker.
- When you use commands such as docker run, the client sends these commands to dockerd, which carries them out.
- The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

TechData-Infinity-Devops with MultiCloud

Docker host

- Docker host is used to provide an environmental execute and run applications it contains the docker daemon images, containers, networks and storages.

Docker Hub/Registry

- Docker registry manages and stores the docker images.
- There are two types of registries in the docker.

1. Public registry- public registry is also called as docker hub.
2. Private registry- it is used to share images within the Enterprise.

- A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.
- When you use the docker pull or docker run commands, the required images are pulled from your configured registry.
- When you use the docker push command, your image is pushed to your configured registry.

Docker images

- Docker images are the read only binary templates used to create docker containers.

Or

Single file with all dependency and configuration required to run a program.

- Ways to create an images-

1. Take image from docker hub
2. Create image from docker file
3. Create image from existing docker containers.

Docker Objects

- When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects.
- These we will discuss one by one.

Objects

1. Images

- Images are nothing but a read-only binary template that can build containers.
- They also contain metadata that describe the container's capabilities and needs.
- Images are used to store and ship applications.
- An image can be used on its own to build a container or customized to add additional elements to extend the current configuration.

- You might create your own images or you might only use those created by others and published in a registry.
- To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it.
- Each instruction in a Dockerfile creates a layer in the image.
- When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt.
- This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Docker Container

- Container hold the entire packages that is needed to run the application.
  Or
  In other words we can say that the image is template and the container is a copy of that template.

- Container is like a virtual machine.
- Images becomes container when they run on docker engine.
- To see all images present in your local machine
  # docker images

- To find out images in docker hub
  # docker search Jenkins

- To download image from docker hub to local machine.
  # docker pull Jenkins

- To give name to container
  #docker run -it –name container_name ubuntu bin/ bash

- To check service is start or not
  #service docker status

- To start container
  #docker start container_name

- To go inside container
  #docker attach container_name

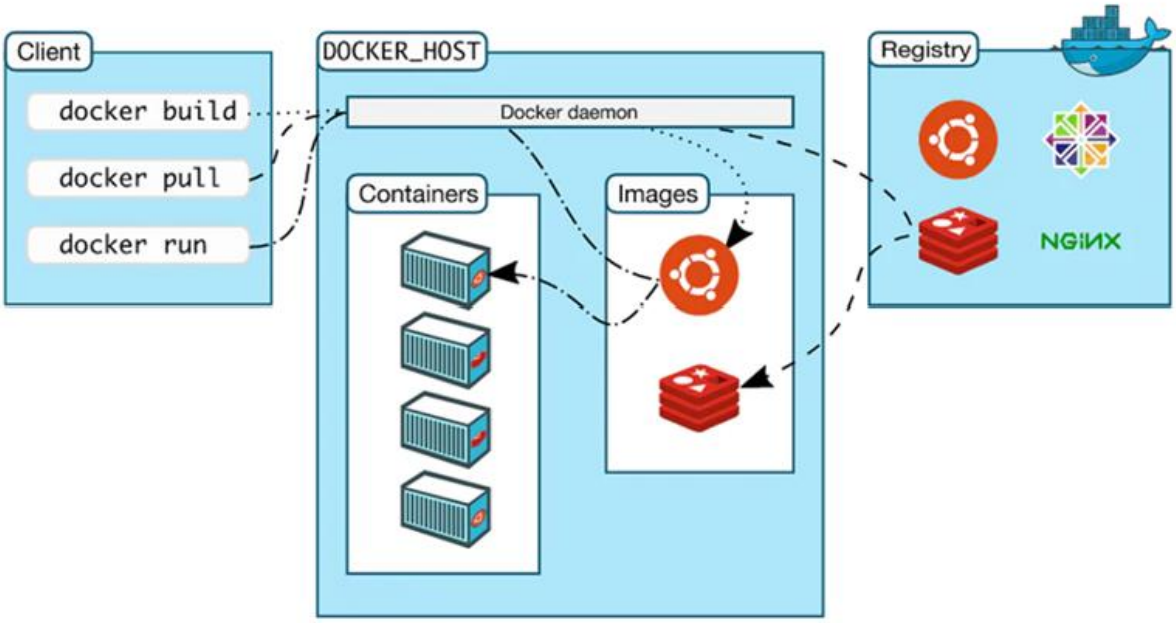
- To see all containers
  #docker ps -a

- To see only running containers
  #docker ps

- To stop container
  #docker stop container_name

- To delete container
  #docker rm container_name

TechData-Infinity-Devops with MultiCloud



Dockerfile Components & Commands

Therefore, create one container first.

#docker run -it –name container_name ubuntu /bin/bash

Dockerfile

- Docker file is basically a text file it contains set of instruction.
- Automation of docker image creation

Docker Components

- **FROM-** Every dockerfile starts with from command for Base image this command must be on top of the docker file. ex. FROM Ubuntu

- **RUN-** The run command is used to run any shell commands these commands will run on top of the base image layer. run commands executed during the build time. you can use this command any number of times in dockerfile. ex. RUN apt-get update && apt-get install -y nginx

- **MAINTAINER**– Author/ Owner /description ex. MAINTAINER “ygminds73@gmail.com”

- **COPY**– It will copy the files and directories from the host machine to the container. if the destination does not exist. it will create the directory. ex. COPY . /usr/src/app

- **ADD**– it will copy the files and directories from the host machine to the container and its support copy the files from remote URLs. ex. ADD <SRC> <DEST>

- **EXPOSE**– to expose points such as port 8080 for tomcat port 80 nginx etc, ex. EXPOSE 80

- **WORKDIR**– It is just like Linux cd command. If you mention any path after workdir the shell will be changed into this directory. The next mentioned commands like RUN, CMD, ENTRYPOINT commands will be executed in this directory.

ex. WORKDIR /devops

RUN npm install

- **CMD**– The CMD command doesn't execute during the build time it will execute after the creation of the container. there can be only one CMD command in Dockerfile. if you add more than one CMD commands the last one will be executed and remaining all will be skipped. whatever you are mentioning commands with CMD command in Dockefile can be overwritten with docker run command. if there is ENTRYPOINT command in the Dockerfile the CMD command will be executed after entry point command. with CMD command you can pass arguments to ENTRYPOINT command. CMD command you can use in two types one is as executable for and another one is parameters form
  - ex. **Parameters form:**

CMD ["param1″,"param2"] this param1 and param2 will be passed to ENTRYPOINT command as a arguments.

ENTRYPOINT ["/db.sh"]

CMD ["postgress"]

Executable form:

CMD ["python", "app.py"]

python app.py will be run in the container after the creation of the container.

- **ENTRYPOINT**–

The first command that will execute in the container is ENTRYPOINT command. ENTRYPOINT command will be executed before the CMD command. If you mentioned more than one, only the last one will be executed and remaining all will be skipped. CMD command paraments are arguments to ENTRYPOINT command.

ex.

ENTRYPOINT ["/db.sh"]

CMD ["postgress"]

above commands will run as /db.sh postgress

here postgress is the first argument to script db.sh

- **ENV**– This command is used to set environment variables in the container.

DockerFile

1. Create a file name docker file

2. Add instruction in docker file

3. Build docker file to create image

4. Run image to create container

#vim dockerfile

```
# Dockerfile Maintainer
MAINTAINER "techdatainfinity111@gmail.com"

# Install nginx and adjust nginx config to stay in foreground
RUN apt-get update && apt-get install -y nginx; \
 echo "daemon off;" >> /etc/nginx/nginx.conf

# Expose HTTP
EXPOSE 80

# Start nginx
CMD ["/usr/sbin/nginx"]
```

To create image out of docker file

#docker build -t myimage.

#docker ps -a

#docker images

- **Now create container from the above image**

#docker run -it –name mycontainer myimage /bin/bash

#cat /tmp/testfile

Docker Volume & how to share it

- Volume is simply a directory inside our containers
- Firstly we have to declare this directory as a volume and then share volume
- Even if we stop container still we can access volume
- Volume will be created in Once container
- You will declare a directory as a volume only while creating container
- You can’t create volume from existing container
- You can share one volume across any number of containers
- Volume will not be included when you update an image

You can map volume in the two ways

1. Container – container

2. Host – container

Benefits of volume

- Decoupling container from storage
- Share volume among different containers
- Attach volume to containers
- On deleting container volume does not delete

Creating volume from Dockerfile

- Create a docker file and write

Then create image from this docker file

#docker build -t myimage

- **Now create a container from this image and run**
  - #docker run -it–name cont1 myimage/bin/bash
  - Create files under volume
  - #docker run -it –name cont2 –privileged=true –volumes-from cont1 ubuntu /bin/bash

- **Now try to create volume by using command**
  - #docker run -it –name cont3 -v /volume2 ubuntu /bin/bash
  - Create files in the containers
  - Now create one more container and share volume 2
  - #docker run -it –name cont4 –privileged=true –volumes-from cont3 ubuntu /bin/bash

- **Volumes (Host-container)**

Create/decide a directory which we want to be treat as a volume **/home/ec2 user**

docker run -it –name hostcont -v /home/ec2-user:/volume –privileged=true ubuntu /bin/bash

cd volume/

check and add files

-

Docker Attach Vs Docker Exec

Docker exec creates a new process in the containers environment while docker attach just connect this standard input or output of the main process inside the container to corresponding standard input or output error of current terminal. Docker exec is specifically for running new things in a already started container, be it a shell or some other process

#docker attach jenkins

#docker exec -i -t jenkins /bin/bash

Docker exec is specifically for running new things in a already started container, be it a shell or some other process

TechData-Infinity-Devops with MultiCloud

Docker Port Forwarding–

Port forwarding or port mapping **redirects a communication request from one IP address and port number combination to another**. Through port forwarding, services are exposed to the applications residing outside of the host's internal network.

#docker run -d –name jenkins -p 8080:8080 jenkins/jenkins

Difference between expose and publish

Basically you have three options

1. Neither specify expose nor -P

If you specify neither expose not-p the service in the container will only be accessible from inside the container itself.

2. Only specify expose

If you expose a port, the service in the container is not accessible from outside docker but from inside other docker containers so this is good for inter container communication.

3. Specify expose and –P

If you expose and -P a port, the service in the container is accessible from anywhere even outside docker

If you do -p but do not expose docker does an implicit expose this is because if a port is open to the public, it is automatically also open to the other docker containers.

hence '-P' includes expose

Docker Networks:

- Docker networking is a passage through which all the isolated container communicate. There are mainly five network drivers in docker:
  - **Bridge**: It is the default network driver for a container. You use this network when your application is running on standalone containers, i.e. multiple containers communicating with the same docker host.
  - **Host**: This driver removes the network isolation between docker containers and docker host. You can use it when you don't need any network isolation between host and container.
  - **Overlay**: This network enables swarm services to communicate with each other. You use it when you want the containers to run on different Docker hosts or when you want to form swarm services by multiple applications.
  - **None**: This driver disables all the networking.
  - **macvlan**: This driver assigns mac address to containers to make them look like physical devices. It routes the traffic between containers through their mac addresses. You use this network when you want the containers to look like a physical device, for example, while migrating a VM setup.

Docker Compose:

- Docker Compose is a tool that was developed to help define and share multi-container applications.
- Docker compose is a tool for defining and running multi-container docker applications
- With compose, you can use a compose file to configure your application services. Then you use a single command, you create and start all the services from your configuration.
- Compose have multiple commands so as to start, stop, rebuild, view the status of running services.

TechData-Infinity-Devops with MultiCloud

- To install docker compose – https://www.techgeekbuzz.com/tutorial/docker/how-to-install-docker-compose-in-linux/

Docker Compose File:

- Docker compose uses a yaml file where you define all the steps.
- Default file Is docker-compose.yml or docker-compose.yaml

YAML:

- YAML stands for YAML Ain't Markup Language.
- Previously, it stood for Yet Another Markup Language.
- YAML is a very simple, text-based, human-readable language used to exchange data between people and computers.
- YAML is not a programming language. It is mostly used for storing configuration information.
- YAML files store data, and they do not include commands and instructions.
- YAML is a data serialization language similar to XML or JSON.

Basic Syntax of YAML:

- YAML is used to store key-value pairs, or mappings, and lists, or sequences of elements.
- <key>: <value>
- where <key> represents name and <value> represents data separated by : (the space is mandatory).
- In yaml file whatever data we will give is line by line.
- In yaml any key can also have a child key.
- Sometime we may also need to group parent keys, where we will having something called as sections.

Docker Compose File:

- Service:- A service definition contains configuration that is applied to each container started for that service, much like passing command-line parameters to docker run.
- In simple words collection of everything that i want to tell docker-compose to create a container or multiple containers i.e what is the image you want how many containers etc.
- Here in compose we are not going to refer as a container but as a service.
- Docker compose will not create container it will create service, service will include image, containers, ports etc.
- Compose handles everything as a service.

TechData-Infinity-Devops with MultiCloud

docker-compose.yaml

```yaml
version: "3.9"
services:
  # Service "web"
  web:
    # Build Dockerfile in current dir
    build: .
    # Expose app port 5000 on machine as 8000
    ports:
      - "8000:5000"
    # mount the project directory (current directory) on the host to /code inside the container
    # modify the code on the fly, without having to rebuild the image.
    volumes:
      - .:/code
    # tell flask run to run in development mode and reload the code on change
    environment:
      FLASK_ENV: development
  # Service "redis" from image redis:alpine
  redis:
    image: "redis:alpine"
```

Commands:

- To run compose-file:- docker-compose –f <compose-file> options servicename
- To create/start container:- docker-compose up
- To start in detach mode:- docker-compose up –d
- To start specific service:- docker-compose up –f <servicename>
- To list container:- docker-compose ps <service>
- To stop-remove container:- docker-compose down
- List images:- docker-compose images
- Build services:- docker-compose build servicename
- View output of container:- docker-compose logs –f service
- Stop services :- docker-compose stop sevicename
- Remove:- docker-compose rm servicename
- Execute commands in a running container:- docker-compose exec service command
- Execute commands in a running container detach mode:- docker-compose exec –d
- Scaling container:- docker-compose up –d –scale service=num

Docker Commands

DOCKER VOLUME	
docker volume ls	List all Volumes
docker volume create <volume name>	Create a Volume
docker volume rm <volume name>	Remove Volume

docker volume prune	It removed all unused docker volume
docker volume inspect <volume name>	Full info about volume
DOCKER INFORMATION	
docker ps	Running container
docker ps -a	Show all container
docker ps –format “{{.Names}}”	Get docker container names
docker inspect container	Return low-level information on Docker objects
docker rename	Rename a container
docker pause	Pause all processes within one or more containers
docker logs	Fetch the logs of a container
docker logs –until 10m <container_id>	logs until 10min
docker logs –tail 100 <container_id>	latest 100 lines logs
journalctl CONTAINER_NAME=mycontainer	In order to inspect logs sent to journald
docker container inspect <container name>	Full info about container
docker exec -it <container> /bin/bash	execute commands on running containers

TechData-Infinity-Devops with MultiCloud

dcoker events <container name>	List real time events from container
docker port <container name>	Show port mapping from container
docker top <container name>	Show running process in container
dcoker stats <container name>	Show live resources usage
docker rm <container name>	Remove specific container
docker rm $(docker ps -a -q)	Remove all containers
docker commit	Create a new image from a container's changes
docker cp	Copy files/folders between a container & host
docker kill <container name>	Kill one or more running containers
docker diff	Show changes to file and file system
dcoker update <container name>	To update the configuration of container
docker import {url/file}	Create a image from tarball
docker save [image] > [Tar file]	Save an image to tar file
DOCKER IMAGES	
docker images ls	show all imgaes
docker rmi <image>	Remove image
docker rmi $(sudo docker images -q)	remove all images
DOCKER ACTIONS	

TechData-Infinity-Devops with MultiCloud

docker start	Start container
docker stop	Stop container
docker restart	Restart container
dcoker pause	Pause container
docker unpause	Unpause container
docker wait	Wait container
docker kill	Kill container
docker attach	Attach container /Access of container
DOCKER HUB & SYSTEM	
docker login	Log in to a Docker registry
docker pull	Pull an image or a repository from a registry
docker push	Push an image or a repository to a registry
docker system df	Show docker disk usage.
docker system events	Get real-time events from the server
docker system info	Display system-wide information.
docker system prune	Remove unused data.
DOCKER NETWORKING	
docker network ls	List Network
docker netowrk rm <network>	Remove one or more network
docker netowork inspect <netowrk>	Network information

docker netowrk connect <network><container>	Connect container to a network
docker netowrk disconnect <network><container>	Disconnect container to a network