DOCKER 101

Why Docker?

Developing apps today requires so much more than writing code. Multiple languages, frameworks, architectures, and discontinuous interfaces between tools for each lifecycle stage creates enormous complexity.

For instance, if an end-to-end application is considered for the variety of tech stack it comprises, the maintenance becomes too difficult as a single tool upgrade might result in series of upgrades to other tool dependencies. This becomes a never ending vicious cycle. That's where Docker comes to the rescue. With docker each tool can be containerised with its very own libraries and dependencies such that others remain unaffected.

Docker simplifies and accelerates your workflow, while giving developers the freedom to innovate with their choice of tools, application stacks, and deployment environments for each project.
Containers are a standardized unit of software that allows developers to isolate their app from its environment, solving the "it works on my machine" headache. For millions of developers today, Docker is the de facto standard to build and share containerized apps - from desktop, to the cloud.

Getting started with Docker…

Docker can be easily installed on MacOS or any Linux based system. Docker host runs on Linux kernel on top of which containers are run.

Whereas when windows are in scope, a Linux virtual machine needs to be installed on which ,docker host is mounted on which containers will run.

Introduction to Images and Containers

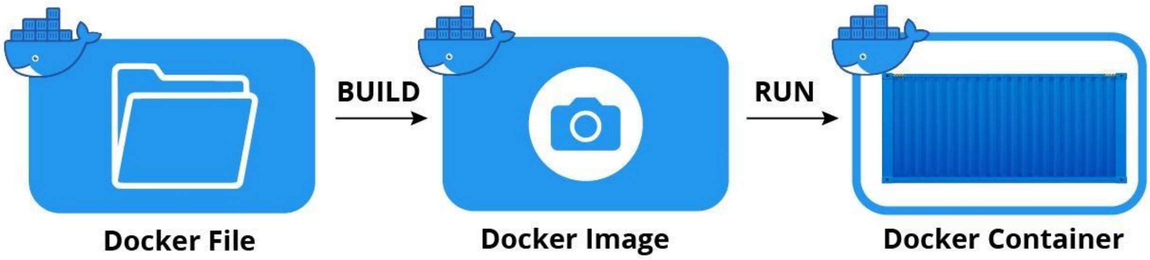


IMAGE:

A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform. It provides a convenient way to package up applications and preconfigured server environments, which you can use for your own private use or share publicly with other Docker users.

A Docker image is made up of a collection of files that bundle together all the essentials – such as installations, application code, and dependencies – required to configure a fully operational container environment.
You can create a Docker image by using one of two methods:
Interactive: By running a container from an existing Docker image, manually changing that container environment through a series of live steps, and saving the resulting state as a new image.
Dockerfile: By constructing a plain-text file, known as a Dockerfile, which provides the specifications for creating a Docker image.

CONTAINER:

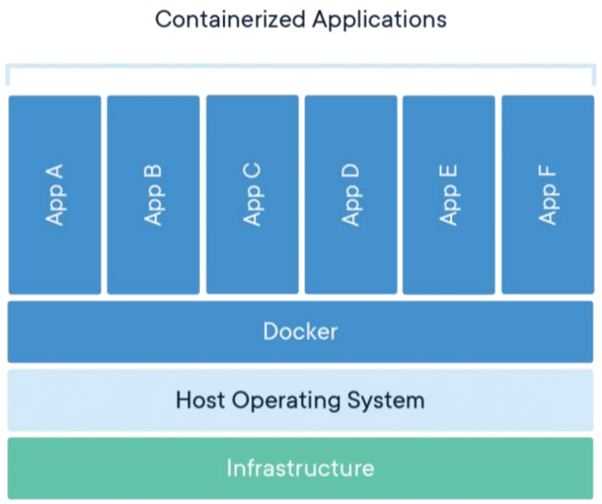
A Docker container is an open source software development platform. Its main benefit is to package applications in containers, allowing them to be portable to any system running a Linux or Windows operating system (OS). A container is created when a docker image is run on docker host. Multiple containers of the same image can be created with different port mapping.

***Please Note: The containers are supposed to host processes but not OS itself. So when the processes within a container complete, the container exits.

Containers vs Virtual Machines:

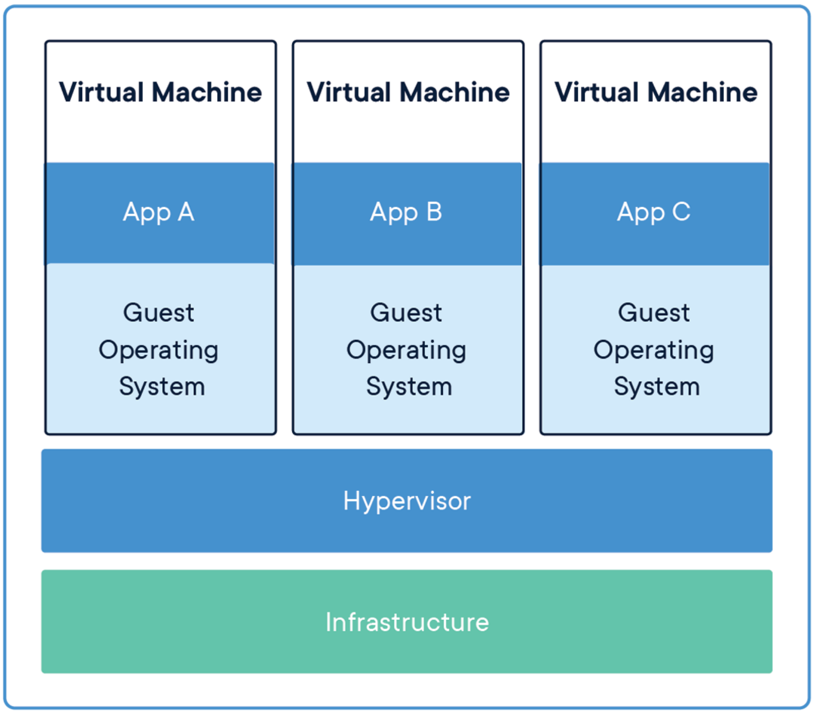
Containers:

Containers basically have the following structure:



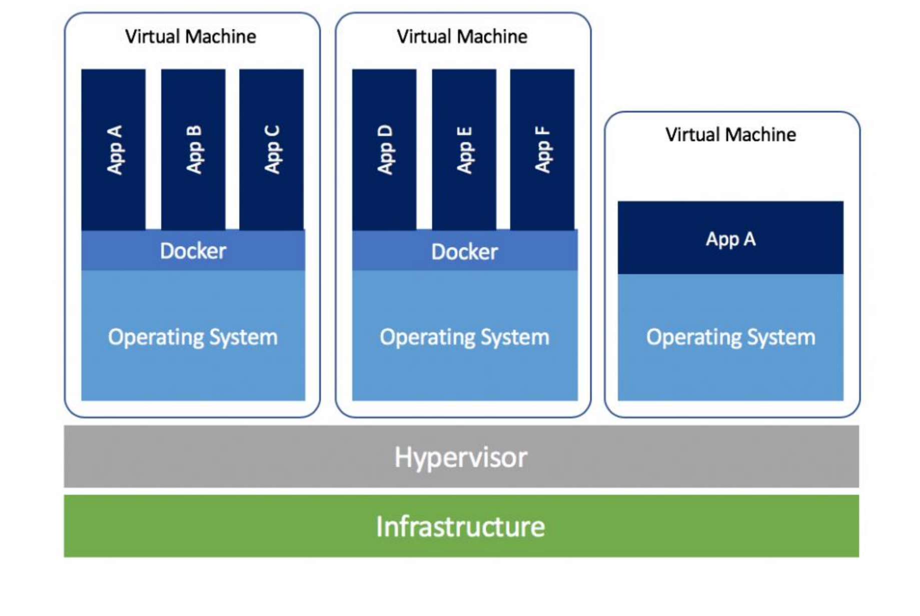
This structure allows less isolation. In other words, the containers share the same OS. This allows easy bootup. The memory too utilized will be comparatively less.

Virtual Machines:



Virtual Machines provide complete isolation to the applications. That is, each application has its own OS and each virtual machine is hosted on Hypervisor that interact with the hardware. This unfortunately puts in a lot of overhead and hence bootup takes time and the memory utilization too is expensive.

Best of both worlds:



In large environments, we can make use of both by integrating the best of both worlds as shown above.

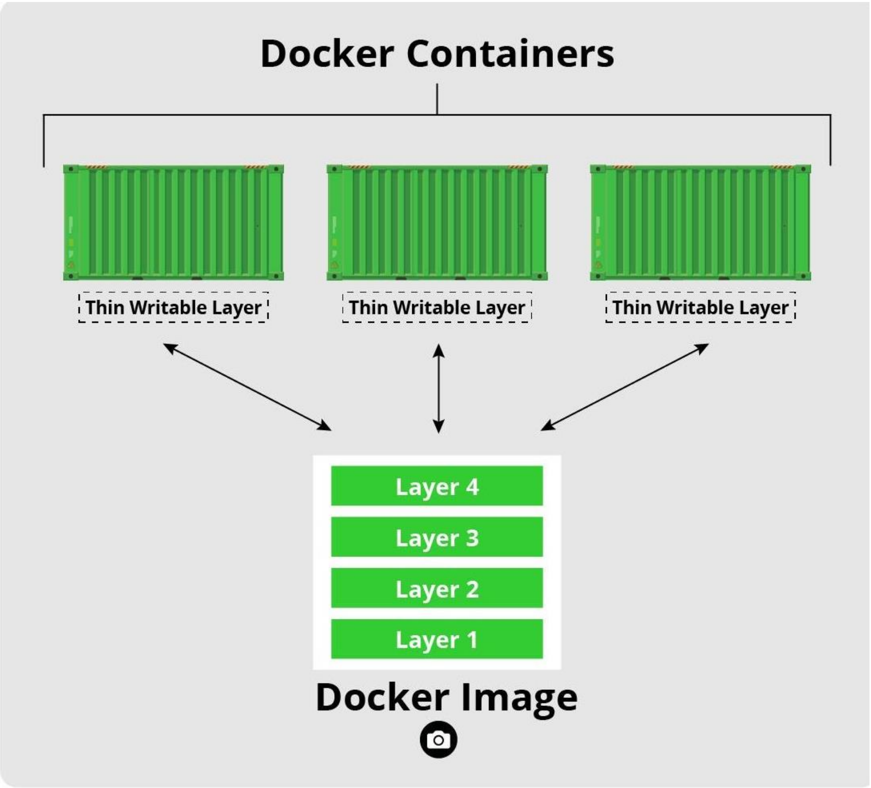
Anatomy of Images and Containers:

IMAGE LAYERS:

Each of the files that make up a Docker image is known as a layer. These layers form a series of intermediate images, built one on top of the other in stages, where each layer is dependent on the layer immediately below it. The hierarchy of your layers is key to efficient lifecycle management of your Docker images. Thus, you should organize layers that change most often as high up the stack as possible. This is because, when you make changes to a layer in your image, Docker not only rebuilds that particular layer, but all layers built from it. Therefore, a change to a layer at the top of a stack involves the least amount of computational work to rebuild the entire image.

CONTAINER LAYER:

Each time Docker launches a container from an image, it adds a thin writable layer, known as the container layer, which stores all changes to the container throughout its runtime. As this layer is the only difference between a live operational container and the source Docker image itself, any number of like-for-like containers can potentially share access to the same underlying image while maintaining their own individual state.



PARENT LAYER/BASE LAYER:

The first layer of a Docker image is known as the parent image. It's the foundation upon which all other layers are built and provides the basic building blocks for your container environments. Typically, parent layer can be an OS Version.

OR,

A base layer of a SQL image on which further layers of configurations can be built.

FILE SYSTEM DIFFERENCES BETWEEN IMAGES AND CONTAINERS

When you create a new container, you add a new writable layer on top of the underlying layers(read-only layers of the image). This layer is often called the "container layer". The major difference between a container and an image is this top writable layer. All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer(diff).

When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged. Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state.

DOCKER COMMANDS:

Docker run :

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

1) DETACHED MODE:

When running a docker image , i.e., while creating container, you must first decide if you want to run the container in the background in a “detached” mode or in the default foreground mode.

which is -d = false

For instance:

```
$ docker run -d ubuntu
```

Ubuntu will run in detached mode.

2) FOREGROUND MODE:

- Interactive mode : -i , attaching STDIN
- Terminal mode : -t, attaching terminal
- Foreground mode: -a, adding STDIN,STDOUT,STDERR

For Instance:

Ubuntu container runs on open terminal as default which makes it to terminate it when the container processes are still running.

Hence,

Command will be like :

```
$ docker run -d ubuntu
```

This will create a container which has a 64 character long SHA-256 Id.

i.e., `f78375b1c487e03c9438c729345e54db9d20cfa2ac1fc3494b6eb60872e74778`

So to identify this container, we can either use the entire string of id or use the first few characters of it or the name of the container

i.e., f78 or f78375b1c487 or “evil_ptolemy”(name that’s generated randomly from docker host during container creation)

Suppose now its required to see the output of the container processes, we can re-attach the container foreground.

i.e.,

```
$ docker run -a f78
```

Or

```
$ docker run -a evil_ptolem
```

In case if we want to interact with the application, we can have an interactive mode along with terminal created:

Like:

```
$ docker run -it ubuntu bash
$ cat /etc/os-release
```

3) CICD SETUP:

Container ID can be passed to a specific file required as follows:

```
$ docker run -d ubuntu –cidfile /tmp/hello-world.cid
```

4) VERSION TAGS OF IMAGE:

A particular version of an image can be pulled from docker hub by specifying the tags

```
$ docker run -d ubuntu:22.04
```

5) NETWORK SETTING OF CONTAINER:

The network of a container can be set up with the following options.
None : the container will be isolated completely.

```
$ docker run --rm -d --network host --name my_nginx nginx
```

Bridge: Connect the container to the bridge. We can have multiple web server containers with bridge network nested on the same docker host, exposed with different port.

```
$ docker network create -d bridge my-bridge

$ docker network ls

NETWORK ID        NAME              DRIVER            SCOPE

326ddef352c5      bridge            bridge            local

1ca18e6b4867      host              host              local

e0fc5f7ff50e      my-bridge         bridge            local

e9530f1fb046      none              null              local

$ docker run -d --name test1 --network my-bridge busybox sh -c "while true;do sleep 3600;done"
```

Host: uses network of that of the host. Hence we can have only single container of a webserver nested on the docker host in this case.
Compared to the default bridge mode, the host mode gives significantly better networking performance since it uses the host’s native networking stack whereas the bridge has to go through one level of virtualization through the docker daemon. It is recommended to run containers in this mode when their networking performance is critical.

```
$ docker run -d --name test3 --net=host centos:7 /bin/bash -c "while true; do sleep 3600; done"
```

- Container: users network of another container specified.

```
$ docker run -d --name redis example/redis --bind 127.0.0.1

$ docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1
```

- User Defined Network:

```
$ docker network create -d bridge my-net

$ docker run --network=my-net -itd --name=container3 busybox
```

6) MANAGING /etc/hosts

Your container will have lines in /etc/hosts which define the hostname of the container itself as well as localhost and a few other common things. The --add-host flag can be used to add additional lines to /etc/hosts.

```
$ docker run -it --add-host db-static:86.75.30.9 ubuntu cat /etc/hosts

172.17.0.22     09d03f76bf2c

fe00::0         ip6-localnet
```

```
ff00::0        ip6-mcastprefix
ff02::1        ip6-allnodes
ff02::2        ip6-allrouters
127.0.0.1        localhost
::1              localhost ip6-localhost ip6-loopback
86.75.30.9      db-static
```

7) REMOVE WHEN EXITS

The container will be removed when the container exits.

```
$ docker run -it --rm --pid=host myhtop
```

8) RESTARTS WHEN EXITS

The container get restarted when exited. This can be mentioned explicitly when the container is created.

By default: --restart = no

Options :

no: default

```
$ docker run --restart=no redis
```

on-failure: restarts when it’s a non-zero exit status and after a count of 10.

```
$ docker run --restart=on-failure:10 redis
```

always: restarts indefinitely

```
$ docker run --restart=always redis
```

unless-stopped: restarts on daemon start up and always regardless of exit status unless explicitly stopped.

```
$ docker run --restart=unless-stopped redis
```

9) OVERRIDING DOCKERFILE IMAGE DEFAULTS

CMD:

Consider a Dockerfile which is as follows:

```
FROM ubuntu
MAINTAINER xyz
RUN apt-get update
ENTRYPOINT ["echo", "Hello"]
CMD ["World"]
```

So when the image is built,

```
$ sudo docker build . -t ourimage
```

and run with the command :

```
$  sudo docker run ourimage
```

it prints :

“Hello World”

This happens as there is no name input given to the docker run command. So ideally it executes according to CMD given in the docker file.

- ENTRYPOINT:

When the docker run command is run with the name parameter:
i.e.,

```
$  sudo docker run ourimage xyz
```

it prints :
"Hello XYZ"
This is executed as per the entrypoint command which expects a name parameter.
So CMD is overridden by ENTRYPOINT command.
Entrypoint command can be executed from the docker run command as well.
i.e.,

```
$ docker run -it ourimage –entrypoint="Hello,xyz!"
```

This will override both ENTRYPOINT and CMD in Dockerfile and will print just "Hello,xyz!"

- EXPOSE:

This includes a few options that can be specified on the docker run command

–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.

```
$ docker run –expose 5000 –name webserver nginx
```

-P

Publishes all ports to the host interfaces.

-p=[]

Publishes port to the docker host so the container is accessible from host/container

```
$ docker run -p 8080:5000 -name webserver nginx
```

–link=""

Helps to link containers to one another.
If we take a classic voting end-to-end application.it might have 2 dbs,one redis container named redis and one postgres container called DB.So the voter api will be connecting to both dbs.So to link these 3 we will have a command like :

```
$ docker run -d --link redis:redis --link db:db —name voter voter-app
```

- ENV:

Docker Automatically sets some environment variables when creating a linux container.
Such as :

HOME:	Set based on the value of USER
HOSTNAME	The hostname associated with the container
PATH	Includes popular directories as :

usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin	
TERM	xterm if the container is allocated a pseudo-TTY

In case of additional env variables, it can be explitly placed in command like :

```
$ export today=Wednesday

$ docker run -e "deep=purple" -e today --rm alpine env
```

- HEALTHCHECK:

Healthcheck is basically used to check health of the containers in intervals.
Such as:

```
$ docker run -d --name db --health-cmd "curl --fail http://localhost:8091/pools || exit 1" --health-interval=5s --
timeout=3s xyz/xyz
```

Multiple options that are available for us to setup the healthcheck for a container are as follows:

--health-cmd	Command to run to check health
--health-interval	Time between running the check
--health-retries	Consecutive failures needed to report unhealthy
--health-timeout	Maximum time to allow one check to run
--health-start-period	Start period for the container to initialize before starting health-retries countdown
--no-healthcheck	Disable any container-specified HEALTHCHECK

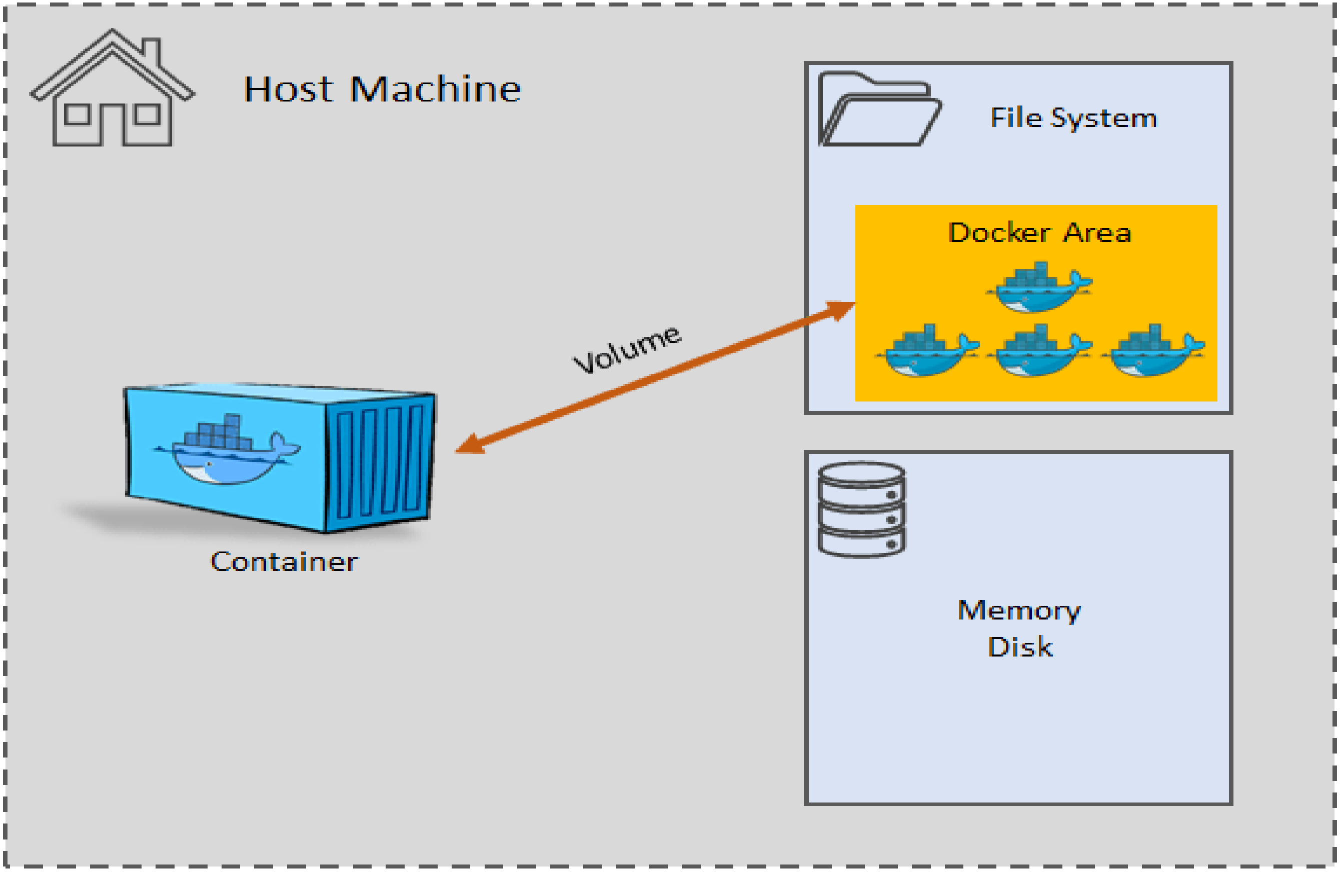
- VOLUMES:

For instance if we consider running a Jenkins container, when stopped the data is lost and the settings done should be redone again.
This is where volumes come into picture. We can store the data so that when it persists even when the container exits.
There are 3 types of volumes present

Docker Volumes
Bind Volumes
Tmpfs Mounts

Docker Volumes:

Volumes are part of the Host filesystem. For example, /var/lib/jenkins is the location that is shared with the docker container, and this location will act as the persistent data in the docker container. Remember this is isolated from the host machine to modify.



Scenarios that hold when using docker volumes:
>> What happens when the container exits? We lose the data
>> What happens when the docker daemon exits? We lose the data

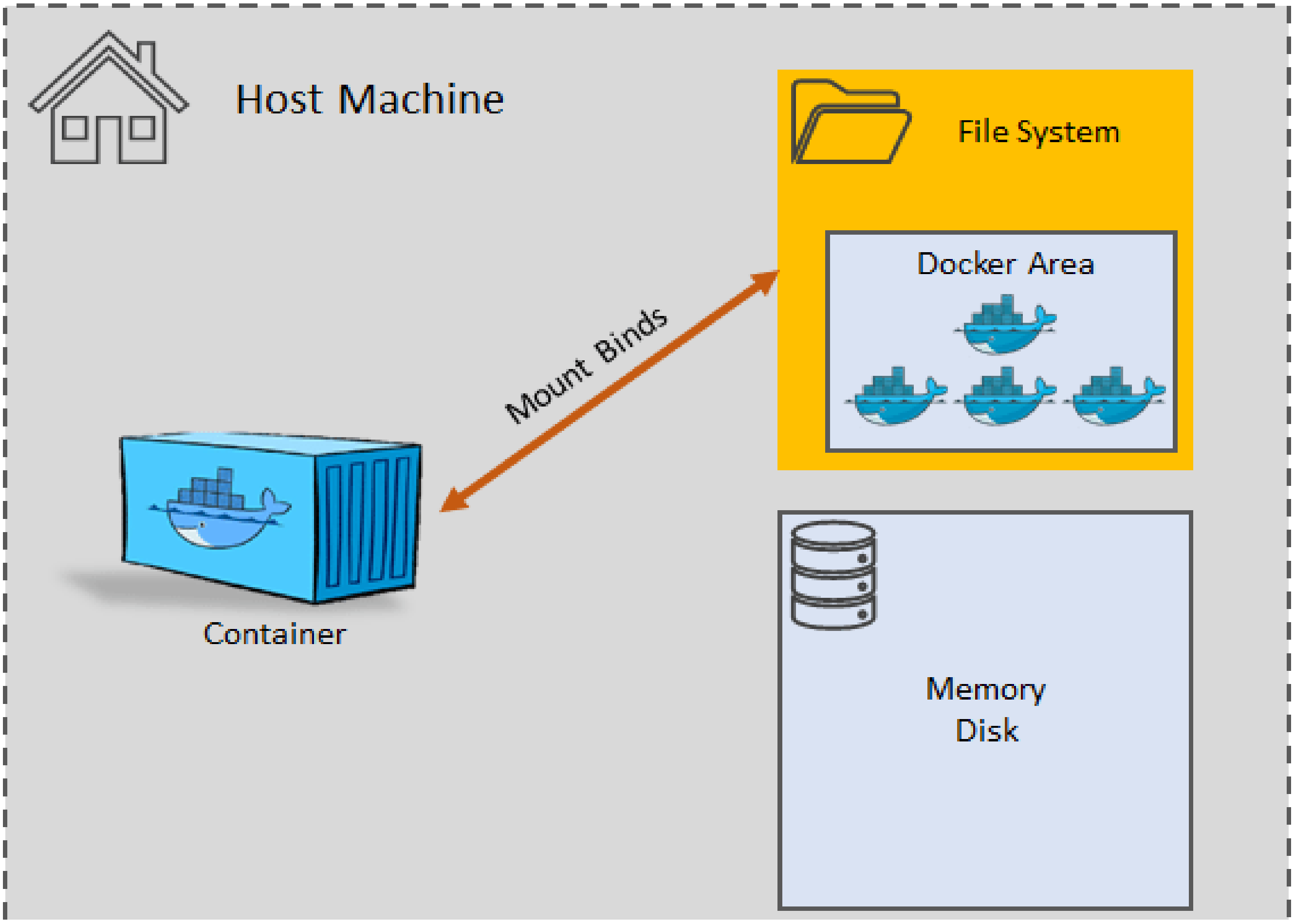
These Docker volumes can be created by command docker volume create command. Or this can be created when a container is created at any means like Dockerfile or Docker compose or docker command line.

Example:

```
$ docker run –name mysql --mount type=volume,src=volumename,dst=/container/path mysql:5.7
```

BIND MOUNTS:

Bind Mounts can be attached at any point at the file system or any disk which can already be used by the Host machine and can be modified from both containers and from the Host. But the limitation with the Bind Mounts are you cannot use the Docker CLI to manage Bind Mounts



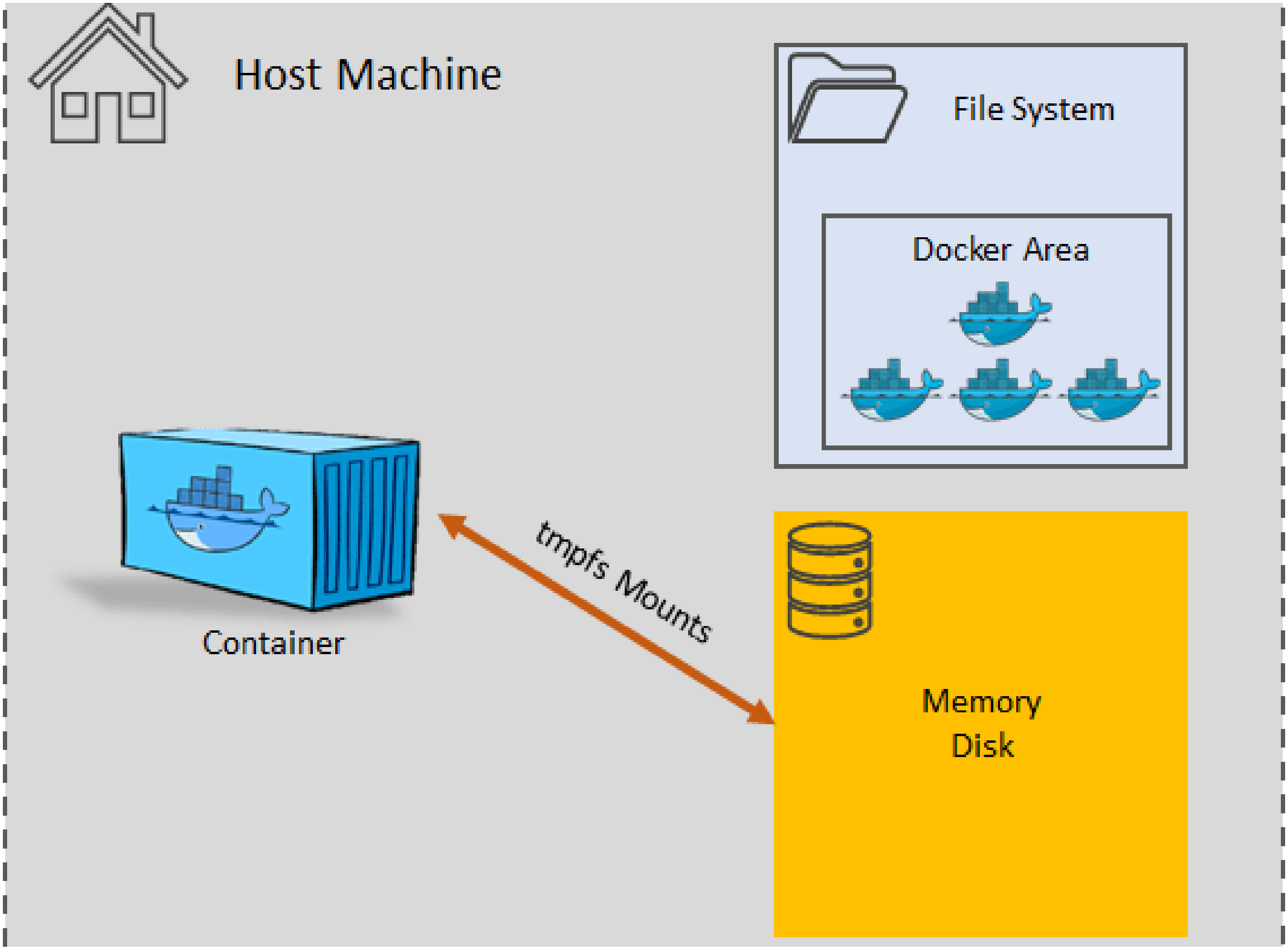
Scenarios that hold when used Bind mounts:
>> What happens when the container exits? We don't lose the data
>> What happens when the docker daemon exits? We lose the data

You can use --mount type=bind, source=/host/path/,target=/container/path to configure the Bind Mounts. Remember, Docker container can modify the file system attached with it and can be harmful if we misuse the containers
Example:

```
$ docker run –name mysql --mount type=bind,src=/host/path/,dst=/container/path mysql:5.7
```

TMPFS MOUNTS:

TMPFS mounts is not the persistent data on the disk. It will be available persistent on Neither Host nor Container filesystem.



Scenario that will hold when used TMPFS mounts:
>> What happens when the container exits? We don’t lose the data
>> What happens when the docker daemon exits? We don’t lose the data

You can create the mount by using --mount type=tmpfs, destination=/app. To create tmpfs mounts, you don’t have to create a file structure in your host file system. You can mention only the destination file path and it will create the directory structure on its own.

Example:

```
$ docker run –name mysql --mount type=tmpfs, dst=/app/path mysql:5.7
```

Docker Attach:

Alternative for:

- docker run -a
- docker run -it

Example:

```
$ docker run -d --name topdemo ubuntu /usr/bin/top -b

$ docker attach topdemo
```

Docker Build:

This command is used to build the Dockerfile to create images of application that can be shipped and run anywhere.

1. Docker Build with URL:

```
$ docker build github.com/creack/docker-firefox
```

This will clone the git repository and use the cloned repository as context.The Dockerfile at the root will be used as “Dockerfile” to build the image.You can specify **git://** or **git@** scheme too.

```
$ docker build -f ctx/Dockerfile http://server/ctx.tar.gz
```

This sends the URL http://server/ctx.tar.gz to the Docker daemon, which downloads and extracts the referenced tarball. The -f ctx/Dockerfile parameter specifies a path inside ctx.tar.gz to the Dockerfile that is used to build the image

2. Docker Build with “-“:

```
docker build - < Dockerfile
```

This will read a Dockerfile from STDIN without context. Due to the lack of a context, no contents of any local directory will be sent to the Docker daemon. Since there is no context, a Dockerfile ADD only works if it refers to a remote URL.

```
$ docker build - < context.tar.gz
```

This will build an image for a compressed context read from STDIN. Supported formats are: bzip2, gzip and xz.

3. Docker Build with .dockerignorefile:

```
$ docker build .
```

docker build searches for a .dockerignore file relative to the Dockerfile name.

For example, running **docker build -f myapp.Dockerfile.** will first look for an ignore file named **myapp.Dockerfile.dockerignore.**
If such a file is not found, the .dockerignore file is used if present.

Using a Dockerfile based .dockerignore is useful if a project contains multiple Dockerfiles that expect to ignore different sets of files.

4. Docker Build with tag “-t”:

```
$ docker build -t vieux/apache:2.0 .
```

This will be building the image with the name provided with -t. We can provide multiple tag names given for an image that’s built.

5. Docker build with “-f”:

```
$ docker build -f Dockerfile.colo .
```

This will use Dockerfile.colo to build the image instead of Dockerfile.

6. Docker build with add-host

This command is equivalent to running the image with bash and adding host into /etc/hosts

```
$ docker build --add-host=docker:10.180.0.1 .
```

7. Docker build with --target

The command allows us to specify an intermediate stage as final stage of the build of an image so that the rest of the stages can be ignored.

```
$ docker build -t mybuildimage --target build-env .
```

8. Docker build with –output

The command helps to specify the output such as a jar or tar file. The docker command usually builds an image. If we wish to build something else out of Dockerfile, we can do so with this option.

```
docker build --output type=tar,dest=out.tar .
```

Docker Commit:

This command is used to create an image out of a container.

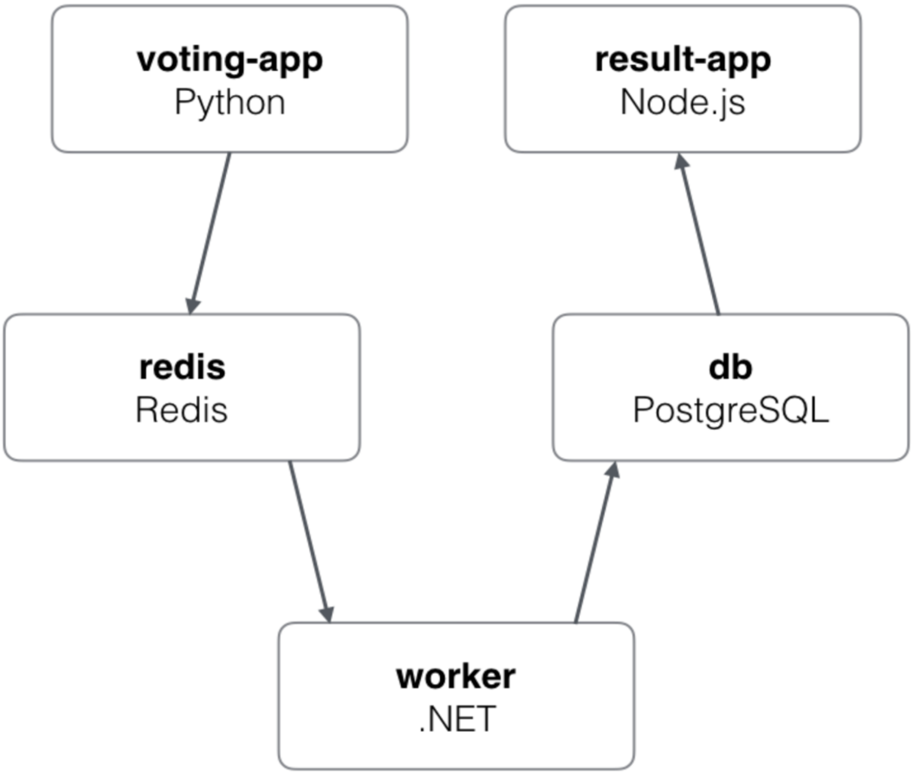
```
$ docker ps
CONTAINER ID       IMAGE              COMMAND            CREATED            STATUS             PORTS              NAMES
c3f279d17e0a       ubuntu:12.04       /bin/bash          7 days ago         Up 25 hours                           desperate_dubinsky
197387f1b436       ubuntu:12.04       /bin/bash          7 days ago         Up 25 hours                           focused_hamilton
$ docker commit c3f279d17e0a  svendowideit/testimage:version3
f5283438590d
$ docker images
REPOSITORY                        TAG               ID                CREATED           SIZE
svendowideit/testimage            version3          f5283438590d      16 seconds ago    335.7 MB
```

DOCKER COMPOSE

Docker compose yaml file is created when you want multiple containers to be brought up at a single time.

If we consider the classic voting application as example, it has multiple components.

- Vote – UI to cast vote
- Redis – in memory DB used as cache/message broker
- Vote worker – the backend that registers the vote and persists into the system.
- db – a classic PostgreSQL used to store the votes
- Result – UI to show the votes casted, to check who is leading/winning.



All these 5 elements are different images and to spun them up the classic way would require us to execute 5 different docker commands.

Instead, we can use docker compose yaml file to spin up all of these images together without hassle.

Docker commands to bring up this end to end application will be :

```
$ Docker run -d --name redis redis
$ Docker run -d –name db -e postgres_user postgres -e postgres_password password -p 5432:5432 postgres:9.4
$ Docker run -d –name vote -p 5000:80 –link redis:redis voting-app
$ Docker run -d –name result -p 5001:80 –link db:db result-app
$ Docker run -d –name worker –link db:db –link redis:redis worker
```

This will be a tedious process if we have more components which should be spun up on the account of an application.

To overcome this issue, we use docker compose.
So the same commands stated above can be decoded as below in docker compose and when executed, all of the components can be spun up implicitly.

DOCKER COMPOSE *(Please check the indentation, you might have to format it)*

```
version :2
    services
        redis
            image:redis
        db
            image:postgres:10.5
            port
                -5432:5432
        environment
            -postgres_user:postgres
            -postgres_password:password
        vote
            image:voter-app
            port
                -5000:80
            link
                -redis
        worker
            image:worker
            links
                -redis
                -db
        result
            image:result
            ports
                -5001:80
            links
                -db
```

Execution of this file is like :

```
$ docker compose up
```

When executed, all the containers will be created and we can have our application accessed at

- http://localhost:5000 which will be to cast vote and
- http://localhost:5001 which will show the results.

There are different versions of docker compose files available in which version 1 is deprecated.

The VERSION is usually specified at the top of the docker compose file.The latest version of docker compose for the same voting-app will comprise of additional bits of information.

The usual parts of docker compose has the following bits:

- VERSION
- SERVICES
- VOLUMES
- NETWORKS

(Please check the indentation, you might have to format it)

```
version: "3.9"

services:

 redis:

  image: redis:alpine

  ports:

   - "6379"

  networks:

   - frontend

  deploy:

   replicas: 2

   update_config:

    parallelism: 2

    delay: 10s

   restart_policy:

    condition: on-failure

 db:

  image: postgres:9.4

  volumes:

   - db-data:/var/lib/postgresql/data

  networks:

   - backend
  deploy:
    placement:
      max_replicas_per_node: 1
      constraints:
        - "node.role==manager"
vote:
  image: dockersamples/examplevotingapp_vote:before
  ports:
    - "5000:80"
  networks:
    - frontend
  depends_on:
    - redis
  deploy:
    replicas: 2
    update_config:
      parallelism: 2
    restart_policy:
      condition: on-failure
result:
  image: dockersamples/examplevotingapp_result:before
  ports:
    - "5001:80"
  networks:
    - backend
  depends_on:
    - db
  deploy:
    replicas: 1
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure
worker:
  image: dockersamples/examplevotingapp_worker
```

```
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints:
          - "node.role==manager"
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints:
          - "node.role==manager"
networks:
  frontend:
  backend:
volumes:
  db-data:
```

BUILD:

- Build: Build can be specified either as a string containing a path to the build context.
- Context: Context is either a path to a directory containing a Dockerfile or a url to a Git repository
- Dockerfile: Dockerfile is the alternative file to build the image with.
- Args: Args will have all the build arguments required during the build process.
- Cache_from: Cache_from is used to the list of the images that engine uses for cache resolution.
- Labels: Add metadata to the resulting image using docker labels.
- Network: Set the network containers connect to the run instructions during the build.
- Shm_size: Set the size of the /dev/shm partition for the build containers.
- Target: Set the targeted stage. Such as dev, pre-prod or prod.
- Command: Command to override the default command.

```
version: "3.9"

services:

  webapp:

    build:

      context: ./dir

      dockerfile: Dockerfile-alternate

      args:

        buildno: 1

      cache_from:

        - alpine: latest

        - corp/web_app: 3.14

      labels:

          com.example.description: "Accounting webapp"

          com.example.department: "Finance"

          com.example.label-with-empty-value: ""

      network: host
      shm_size: '2gb'

      target: prod

      command: bundle exec thin -p 3000
```

CONFIG:

The top-level configs declaration defines or references configs that can be granted to the services in this stack. The source of the config is either file or external.

The configs have

- short syntax: this only specifies the name.
- long syntax.this specifies more granularity.

Thus the docker compose file would become :

```
version: "3.9"

services:

  webapp:

    build:

      context: ./dir

      dockerfile: Dockerfile-alternate

      args:

        buildno: 1

      cache_from:

        - alpine: latest

        - corp/web_app: 3.14

      labels:

          com.example.description: "Accounting webapp"

          com.example.department: "Finance"

          com.example.label-with-empty-value: ""

      network: host

      shm_size: '2gb'

      target: prod

      command: bundle exec thin -p 3000

    configs:

      - source: my_config

        target: /redis_config

        uid: '103'
        gid: '103'
        mode: 0440
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

CONTAINER_NAME

This is to specify a custom container name rather than a generated default random name. Hence Docker compose becomes:

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        - alpine: latest
        - corp/web_app: 3.14
      labels:
          com.example.description: "Accounting webapp"
          com.example.department: "Finance"
          com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
      - source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

DEPENDS_ON:

This option is nothing but –link in docker run command.It will help link the containers.
So,possible docker compose will look like :

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        - alpine: latest
        - corp/web_app: 3.14
      labels:
          com.example.description: "Accounting webapp"
          com.example.department: "Finance"
          com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
      - source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
          - db
          - redis
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

DEPLOY:

Specifies how the container needs to be deployed.

- Endpoint_mode: can be either VIP or dns-rr(DNS-round robin)
- Labels: have labels specified for services.
- Mode: this can be either global/replicated.

  If we have, say 3 components,/containers, for global, all the components will execute different tasks. Whereas, for replicas, each will be a replica of each other serving the same purpose.
- Placement: provides preferences, constraints and max number of replicas per container that it can spin up to.
  - constraints : such as
    1) The role of the user using or creating the container
    2) The type of the OS the container should run on
  - preferences:

    spread: how the replicas should be spread across (zones)
  - max number of replicas
  - replicas: replicas that can be running at any given time.
  - Resources: provides the system limits that a container can have.
    1) CPUs
    2) memory
  - restart policy:
    1) condition: always/on-failure/none
    2) delay: how long to wait before restart
    3) max_attempts: how many times should the docker host try to bring up the container in case of failures.
    4) window: how long to wait before deciding if restart is succeeded or not.
  - Update_config:
    1) Parallelism: how many containers can be updated
    2) Delay: time between an update b/w a group of containers
    3) Failure_action: if we should continue, rollback or pause
    4) Monitor: howe long should we monitor before the next task is rolled out

5) Max_failure_rate: failures to tolerate during an update
6) Order: how to carry the update
   1. stop-first: default;old task is stopped first before starting a new one.
   2. Start-first: start new task and then stop the old one

So the docker compose after adding deploy part will look as follows:

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

ENTRYPOINT:

Override the default entrypoint or command

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

ENV_FILE:

Adding a env variable file either in the same directory path or relative path.

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

ENVIRONMENT:

Adding environment variables to the container in docker compose file

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
```

```
   mode: replicated
   replicas: 6
   endpoint_mode: vip
   labels:
   com.example.description: "This label will appear on the web service"
   constraints:
    node.role: manager
    engine.labels.operatingsystem: ubuntu 18.04
    preferences:
     spread: node.labels.zone
    placement:
     max_replicas_per_node: 2
    update_config:
     parallelism: 2
     delay: 10s
    restart_policy:
     condition: on-failure
   resources:
    limits:
     cpus: '0.50'
     memory: 50M
    reservations:
     cpus: '0.25'
     memory: 20M
   restart_policy:
    condition: on-failure
    delay: 5s
    max_attempts: 3
    window: 120s
   update_config:
    parallelism: 2
    delay: 10s
    order: stop-first
  entrypoint: /code/entrypoint.sh
  env_file:
  - ./common.env
  - ./apps/web.env
  environment:
   RACK_ENV: development
   SHOW: true
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

EXPOSE:

This is used to expose the ports of a container to the host. However these ports will not be accessible outside the container.

This is equivalent to -expose in docker run command

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

EXTERNAL_LINKS:

This is used to link with the containers outside the docker-compose.yml.

So the docker compose will become like :

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

HEALTHCHECK:

Configure the check to run to determine the containers health if they are running as expected or if the container has exited.

- Test: string or an arraylist .If arraylist, first item will be none/cmd/cmd-shell
- Interval
- Timeout
- Retries
- Start_period

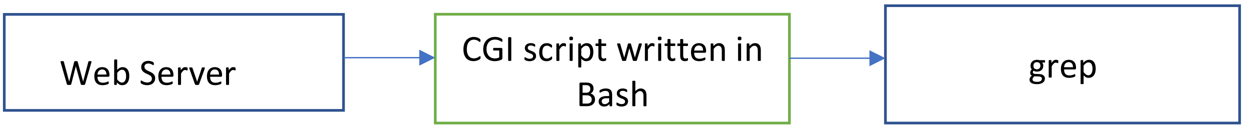
So the docker compose will become like :

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
  redis:
    image: redis
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

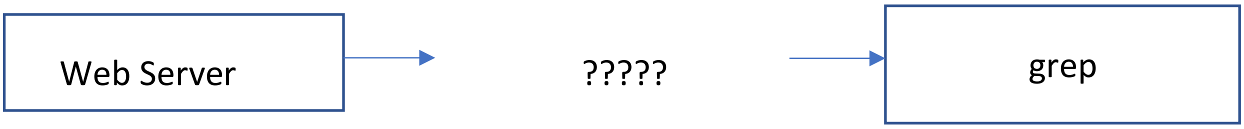
INIT

Run init inside a container that forwards the signals and reaps processes.(equivalent to docker run).This is responsible to start the system, ssh, bash and other daemons.

Why init is required?



Imagine our container runs a webserver that runs a CGI script that’s written in bash.This script calls grep.
Consider an instance where web server decides that this cgi script is taking too long and kills the script.But grep is not affected and it keeps running.



When grep finishes, it becomes a zombie.
The web server doesn’t know about grep and it doesn’t reap it. Hence grep stays in system.

We usually run third party apps/containers. You are running someone else’s code, so you really need to be sure that those apps don’t spawn processes in such a way that become/create zombie processes later. Hence we should run a proper init system to prevent problems.

So the docker compose will become as follows:

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
```

```
    alpine: latest
    corp/web_app: 3.14
  labels:
    com.example.description: "Accounting webapp"
    com.example.department: "Finance"
    com.example.label-with-empty-value: ""
  network: host
  shm_size: '2gb'
  target: prod
  command: bundle exec thin -p 3000
configs:
    source: my_config
    target: /redis_config
    uid: '103'
    gid: '103'
    mode: 0440
container_name: custom_name
depends_on:
    db
    redis
deploy:
  mode: replicated
  replicas: 6
  endpoint_mode: vip
  labels:
  com.example.description: "This label will appear on the web service"
  constraints:
    node.role: manager
    engine.labels.operatingsystem: ubuntu 18.04
    preferences:
      spread: node.labels.zone
    placement:
      max_replicas_per_node: 2
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure
  resources:
    limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
  redis:
    image: redis
    init: true
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

LOGGING:

This gives an option to configure logging for the services.

- Driver : syslog / json-file /awslogs
- Options :
    - Syslog-address
    - Max-size
    - Max-file

```
version: "3.9"

services:

  webapp:

    build:

      context: ./dir

      dockerfile: Dockerfile-alternate

      args:

        buildno: 1

      cache_from:

        alpine: latest

        corp/web_app: 3.14

      labels:

        com.example.description: "Accounting webapp"

        com.example.department: "Finance"

        com.example.label-with-empty-value: ""

      network: host

      shm_size: '2gb'

      target: prod

      command: bundle exec thin -p 3000

    configs:

        source: my_config

        target: /redis_config

        uid: '103'

        gid: '103'

        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
```

```
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
  redis:
    image: redis
    init: true
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

Some of the other options can be like :

```
logging:
      driver: "awslogs"
      options:
      awslogs-region: us-east-1
```

```
logging:
      driver: "gelf"
      options:
            gelf-address: “udp://1.2.3.4:12201”
```

```
logging:
      driver: "gelf"
      options:
            gelf-address: “udp://1.2.3.4:12201”
```

```
logging:
      driver: "syslog"
      options:
            syslog-address: “udp://1.2.3.4:12201”
```

NETWORK_MODE

The options to this command are

- Bridge
- Host
- None
- Container : network_mode: "container:[container name/id]"
- Service : network_mode: "service:[service name]"

This is same as docker run with –network command.

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    network_mode: "service:redis"
  redis:
    image: redis
    init: true
  db:
    image: postgres
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

NETWORKS:

Networks is a section similar to that of services.

- Aliases:
  The container can use the network name given in the networks section or can have a alias described with “aliases”

```
version: "3.9"

services:

  webapp:

    build:

      context: ./dir

      dockerfile: Dockerfile-alternate

      args:

        buildno: 1

      cache_from:

        alpine: latest

        corp/web_app: 3.14

      labels:

        com.example.description: "Accounting webapp"

        com.example.department: "Finance"

        com.example.label-with-empty-value: ""

      network: host

      shm_size: '2gb'

      target: prod

      command: bundle exec thin -p 3000

    configs:

        source: my_config

        target: /redis_config

        uid: '103'

        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
     networks: new
  redis:
    image: redis
    init: true
    networks: legacy
  db:
    image: postgres
    networks:
        new:
            aliases:
            - database
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
networks:
  new:
  legacy:
```

ipv4_address,ipv6_address

Specify ip address for containers for this service when joining the network.

```
version: "3.9"

services:

  webapp:

    build:

      context: ./dir

      dockerfile: Dockerfile-alternate

      args:

        buildno: 1

      cache_from:

        alpine: latest

        corp/web_app: 3.14

      labels:

        com.example.description: "Accounting webapp"

        com.example.department: "Finance"

        com.example.label-with-empty-value: ""

      network: host

      shm_size: '2gb'

      target: prod

      command: bundle exec thin -p 3000

    configs:

        source: my_config

        target: /redis_config

        uid: '103'

        gid: '103'

        mode: 0440

    container_name: custom_name

    depends_on:

        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    networks:
      new:
        ipv4_address: 172.16.238.10
        ipv6_address: 2001:3984:3989::10
  redis:
    image: redis
    init: true
    networks:
      new:
        ipv4_address: 172.16.238.20
        ipv6_address: 2001:3984:3989::20
  db:
    image: postgres
    networks:
      legacy:
        ipv4_address: 172.16.238.45
        ipv6_address: 2001:3984:3989::45
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
networks:
  new:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.0/24"
        - subnet: "2001:3984:3989::/64"
  legacy:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.25/48"
        - subnet: "2001:3984:3989::65/128"
```

PORTS

- Short syntax: this has 3 options
  - Specify both ports – host: container

    ```
    "8000:8000"
    ```

  - Specify container port

    ```
    "3000-3005"
    ```

  - Specify host ip address to bind to AND both ports

    ```
    "127.0.0.1:8001:8001"
    "127.0.0.1:5000-5010:5000-5010"
    "127.0.0.1::5000"
    ```

- Long syntax: This has 4 options
  - Target
  - Published
  - Protocol
  - Mode

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
```

```
    com.example.description: "Accounting webapp"
    com.example.department: "Finance"
    com.example.label-with-empty-value: ""
  network: host
  shm_size: '2gb'
  target: prod
  command: bundle exec thin -p 3000
configs:
    source: my_config
    target: /redis_config
    uid: '103'
    gid: '103'
    mode: 0440
container_name: custom_name
depends_on:
    db
    redis
deploy:
  mode: replicated
  replicas: 6
  endpoint_mode: vip
  labels:
  com.example.description: "This label will appear on the web service"
  constraints:
    node.role: manager
    engine.labels.operatingsystem: ubuntu 18.04
    preferences:
      spread: node.labels.zone
    placement:
      max_replicas_per_node: 2
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure
  resources:
    limits:
      cpus: '0.50'
      memory: 50M
    reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    networks:
      new:
        ipv4_address: 172.16.238.10
        ipv6_address: 2001:3984:3989::10
    ports:
      - "172.16.238.10:5000:80"
  redis:
    image: redis
    init: true
  db:
    image: postgres
    networks:
        new:
          aliases:
            - database
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
networks:
  new:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.0/24"
        - subnet: "2001:3984:3989::/64"
  legacy:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.25/48"
        - subnet: "2001:3984:3989::65/128"
```

PROFILES

Profiles can be used to label the components of an end to end components such as backend, frontend, mid-tier

```
version: "3.9"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    networks:
      new:
        ipv4_address: 172.16.238.10
        ipv6_address: 2001:3984:3989::10
    ports:
      - "172.16.238.10:5000:80"
    profiles: frontend
  redis:
    image: redis
    init: true
    profiles: backend
  db:
    image: postgres
    networks:
        new:
          aliases:
            - database
    profiles: backend
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
networks:
  new:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.0/24"
        - subnet: "2001:3984:3989::/64"
  legacy:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.25/48"
        - subnet: "2001:3984:3989::65/128"
profiles:
  - frontend
  - backend
  - midtier
```

SECRETS

This is for secret/authentication/authorization details configuration.
We can either use short or long syntax for secrets section.
Long syntax has more options that will give more granularity

- Source
- Target
- Uid
- Mode

```
version: "3.9"

services:

  webapp:

    build:

      context: ./dir

      dockerfile: Dockerfile-alternate

      args:

        buildno: 1

      cache_from:

        alpine: latest

        corp/web_app: 3.14

      labels:

        com.example.description: "Accounting webapp"

        com.example.department: "Finance"

        com.example.label-with-empty-value: ""

      network: host

      shm_size: '2gb'

      target: prod

      command: bundle exec thin -p 3000

    configs:

        source: my_config

        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
          max_replicas_per_node: 2
        update_config:
          parallelism: 2
          delay: 10s
        restart_policy:
          condition: on-failure
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    entrypoint: /code/entrypoint.sh
    env_file:
    - ./common.env
    - ./apps/web.env
    environment:
      RACK_ENV: development
      SHOW: true
    expose:
      - "3000"
    external_links:
      -project_db:db_mysql
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m30s
      timeout: 10s
      retries: 3
      start_period: 40s
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    networks:
      new:
        ipv4_address: 172.16.238.10
        ipv6_address: 2001:3984:3989::10
    ports:
      - "172.16.238.10:5000:80"
    profiles: frontend
    secrets:
      - source: my_secret
        target: redis_secret
        uid: '103'
        gid: '103'
        mode: 0440
  redis:
    image: redis
    init: true
    profiles: backend
  db:
    image: postgres
    networks:
        new:
          aliases:
            - database
    profiles: backend
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
networks:
  new:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.0/24"
        - subnet: "2001:3984:3989::/64"
  legacy:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.25/48"
        - subnet: "2001:3984:3989::65/128"
profiles:
  - frontend
  - backend
  - midtier
secrets:
  my_secret:
    file: ./my_secret.txt
  my_other_secret:
    external: true
```

VOLUMES

Mount container,build or named volumes with this option.
This is similar to –mount,-v option in docker run command

```
version: "3.9"
```

```
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
      cache_from:
        alpine: latest
        corp/web_app: 3.14
      labels:
        com.example.description: "Accounting webapp"
        com.example.department: "Finance"
        com.example.label-with-empty-value: ""
      network: host
      shm_size: '2gb'
      target: prod
      command: bundle exec thin -p 3000
    configs:
        source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
        mode: 0440
    container_name: custom_name
    depends_on:
        db
        redis
    deploy:
      mode: replicated
      replicas: 6
      endpoint_mode: vip
      labels:
      com.example.description: "This label will appear on the web service"
      constraints:
        node.role: manager
        engine.labels.operatingsystem: ubuntu 18.04
        preferences:
          spread: node.labels.zone
        placement:
```

```
      max_replicas_per_node: 2
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure
  resources:
    limits:
      cpus: '0.50'
      memory: 50M
    reservations:
      cpus: '0.25'
      memory: 20M
  restart_policy:
    condition: on-failure
    delay: 5s
    max_attempts: 3
    window: 120s
  update_config:
    parallelism: 2
    delay: 10s
    order: stop-first
entrypoint: /code/entrypoint.sh
env_file:
- ./common.env
- ./apps/web.env
environment:
  RACK_ENV: development
  SHOW: true
expose:
  - "3000"
external_links:
  -project_db:db_mysql
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 1m30s
  timeout: 10s
  retries: 3
  start_period: 40s
logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    networks:
      new:
        ipv4_address: 172.16.238.10
        ipv6_address: 2001:3984:3989::10
    ports:
      - "172.16.238.10:5000:80"
    profiles: frontend
    secrets:
      - source: my_secret
        target: redis_secret
        uid: '103'
        gid: '103'
        mode: 0440
    volumes:
      - type: volume
        source: mydata
        target: /data
        volume:
        nocopy: true
      - type: bind
        source: ./static
        target: /opt/app/static
  redis:
    image: redis
    init: true
    profiles: backend
  db:
    image: postgres
    networks:
        new:
          aliases:
            - database
    profiles: backend
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
networks:
  new:
    ipam:
      driver: default
      attachable: true
      config:
        - subnet: "172.16.238.0/24"
        - subnet: "2001:3984:3989::/64"
  legacy:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.25/48"
        - subnet: "2001:3984:3989::65/128"
profiles:
  - frontend
  - backend
  - midtier
secrets:
  my_secret:
    file: ./my_secret.txt
  my_other_secret:
    external: true
volumes:
  mydata:
    driver_opts:
      type: "nfs"
      o: "addr=10.40.0.199,nolock,soft,rw"
      device: ":/docker/example"
  dbdata:
    external: true
```

DOCKER COMPOSE CLIENT COMMANDS

The terminal/cli commands that come in handy to run docker compose are as follows:

```
Commands:
 build         Build or rebuild services
 bundle          Generate a Docker bundle from the Compose file
 config         Validate and view the Compose file
 create          Create services
 down            Stop and remove containers, networks, images, and volumes
 events          Receive real time events from containers
 exec           Execute a command in a running container
 help          Get help on a command
 images          List images
 kill         Kill containers
 logs          View output from containers
 pause           Pause services
 port          Print the public port for a port binding
 ps           List containers
 pull          Pull service images
 push            Push service images
 restart        Restart services
 rm            Remove stopped containers
 run           Run a one-off command
 scale          Set number of containers for a service
 start         Start services
 stop           Stop services
 top           Display the running processes
```

```
unpause          Unpause services
up             Create and start containers
version          Show the Docker-Compose version information
```

Docker diff:

This is used to view all the files that’s changed in the read-write layer of a container

```
$ docker diff 1fdfd1f54c1b
```

Docker History

This is used to see how the image was built.It will have the details of all the layers as to how and when they were built

```
$ docker history docker


IMAGE            CREATED          CREATED BY                              SIZE          COMMENT
3e23a5875458       8 days ago        /bin/sh -c #(nop) ENV LC_ALL=C.UTF-8           0 B
8578938dd170       8 days ago        /bin/sh -c dpkg-reconfigure locales &&    loc   1.245 MB
be51b77efb42      8 days ago        /bin/sh -c apt-get update && apt-get install    338.3 MB
4b137612be55       6 weeks ago        /bin/sh -c #(nop) ADD jessie.tar.xz in /        121 MB
750d58736b4b       6 weeks ago        /bin/sh -c #(nop) MAINTAINER Tianon Gravi <ad   0 B
```

Docker create

This helps to create a writable container layer on the specified image and prepars it for running a specific command

```
$ docker create -v /data --name data ubuntu


240633dfbb98128fa77473d3d9018f6123b99c454b3251427ae190a7d951ad57
```

```
$ docker run --rm --volumes-from data ubuntu ls -la /data


total 8

drwxr-xr-x  2 root root 4096 Dec  5 04:10 .

drwxr-xr-x 48 root root 4096 Dec  5 04:11 ..
```

Docker images

Lists all the images in the system.

```
$ docker images


REPOSITORY            TAG              IMAGE ID         CREATED           SIZE

<none>                <none>           77af4d6b9913       19 hours ago       1.089 GB

committ               latest           b6fa739cedf5       19 hours ago       1.089 GB

<none>                <none>           78a85c484f71       19 hours ago       1.089 GB

docker                latest           30557a29d5ab       20 hours ago       1.089 GB

<none>                <none>           5ed6274db6ce       24 hours ago       1.089 GB

postgres              9                746b819f315e       4 days ago         213.4 MB

postgres              9.3              746b819f315e       4 days ago         213.4 MB

postgres              9.3.5            746b819f315e       4 days ago         213.4 MB

postgres              latest           746b819f315e       4 days ago         213.4 MB
```

We can even list out specific images as :

```
$ docker images java:8
```

Docker inspect

This command helps us inspect almost anything. Be it network, image or container.

```
$ docker inspect c1
```

Where c1 is a container

Docker kill

This is used to kill one or more containers

```
$ docker kill my_container
```

Docker Network

This command is used to work with networks in docker daemon.

- Docker network create : used to create network

  For instance:

```
$  docker network create -d overlay \

  --subnet=192.168.10.0/25 \

  --subnet=192.168.20.0/25 \

  --gateway=192.168.10.100 \

  --gateway=192.168.20.100 \

  --aux-address="my-router=192.168.10.5" --aux-address="my-switch=192.168.10.6" \

  --aux-address="my-printer=192.168.20.5" --aux-address="my-nas=192.168.20.6" \

  my-multihost-network
```

- Docker network connect:used to connect a container or multiple containers to a network

```
$ docker network connect –ip 10.10.36.122 \
--link container1:c1 \
--alias db –alias mysql \
multi-host-network container2
```

- Docker network inspect: used to inspect the features/configuration of a network in docker daemon.
- Docker network ls: used to view all the networks available for us in the system.
- Docker network disconnect :Similar to connect, this option is used to disconnect a container or multiple containers from the network.
- Docker network prune: This will remove all the unused networks
- Docker network rm: This will remove all networks.

Docker node:

In case of orchestration, in docker swarm to be precise, this command is used to promote/demote/update a node.

Options available for us is :

- Docker node promote: promotes a node to become manager
- Docker node demote: demotes a node from being a manager
- Docker node ls: lists all nodes available
- Docker node ps: lists tasks running on one or more nodes
- Docker node update: updates a node
- Docker node rm: removes one or more nodes

Docker ps:

Used to list all the running containers. Along with option “-a”, it lists all the containers both running and exited.

```
$ Docker ps -a
```

Docker pull:

This command is used to pull the docker image from registry but explicitly isn’t run to create a container of that image

```
$ docker pull debian


Using default tag: latest

latest: Pulling from library/debian

fdd5d7827f33: Pull complete

a3ed95caeb02: Pull complete

Digest: sha256:e7d38b3517548a1c71e41bffe9c8ae6d6d29546ce46bf62159837aad072c90aa

Status: Downloaded newer image for debian:latest
```

Docker push:

This is used to push an image to the docker registry.

```
$ docker image push --all-tags registry-host:5000/myname/myimage


The push refers to repository [registry-host:5000/myname/myimage]

195be5f8be1d: Pushed

latest: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527

195be5f8be1d: Layer already exists

v1: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527

195be5f8be1d: Layer already exists

v1.0: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527

195be5f8be1d: Layer already exists

v1.0.1: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527
```

Docker restart:

This command is used to restart one or more containers.

```
$ docker restart 195 fdd
```

Docker rm:

This command is used to remove the containers.If want ot forcefully remove a container, use “-f” along.

```
$ docker rm -v -f --link /webapp/redis
```

Docker rmi:

This is used to remove images form docker daemon

```
$ docker images
REPOSITORY           TAG              IMAGE ID         CREATED           SIZE
test1                latest           fd484f19954f     23 seconds ago     7 B (virtual 4.964 MB)
test                 latest           fd484f19954f     23 seconds ago     7 B (virtual 4.964 MB)
test2                latest           fd484f19954f     23 seconds ago     7 B (virtual 4.964 MB)

$ docker rmi fd484f19954f
Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositories, use -f to force
2013/12/11 05:47:16 Error: failed to remove one or more images

$ docker rmi test1:latest
Untagged: test1:latest

$ docker rmi test2:latest
Untagged: test2:latest

$ docker images
REPOSITORY           TAG              IMAGE ID         CREATED           SIZE
test                 latest           fd484f19954f     23 seconds ago     7 B (virtual 4.964 MB)

$ docker rmi test:latest
Untagged: test:latest
Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
```

Docker Secret:

This command is used to create,inspect,remove secrets in the machine.

Create:

```
$ docker secret create \

  --label env=dev \

  --label rev=20170324 \

  my_secret ./secret.json

eo7jnzguqgtpdah3cm5srfb97
```

Inspect:

```
$ docker secret inspect my_secret

[

    {

        "ID": "eo7jnzguqgtpdah3cm5srfb97",

        "Version": {

            "Index": 17

        },

        "CreatedAt": "2017-03-24T08:15:09.735271783Z",

        "UpdatedAt": "2017-03-24T08:15:09.735271783Z",

        "Spec": {

            "Name": "my_secret",

            "Labels": {

                "env": "dev",

                "rev": "20170324"

            }

        }

    }

]
```

ls:

```
$ docker secret ls


ID                          NAME                        CREATED             UPDATED

6697bflskwj1998km1gnnjr38   q5s5570vtvnimefos1fyeo2u2   6 weeks ago         6 weeks ago
```

```
9u9hk4br2ej0wgngkga6rp4hq   my_secret               5 weeks ago         5 weeks ago
mem02h8n73mybpgqjf0kfi1n0   test_secret             3 seconds ago       3 seconds ago
```

rm:

```
$ docker secret rm secret.json
sapth4csdo5b6wz2p5uimh5xg
```

Docker service:

Docker service is used while orchestration. This is the equivalent to the service section in docker compose file.
All the options will be same of that of the docker compose file.

For instance:

```
$ docker service create \
  --name nginx \
  --replicas 2 \
  --replicas-max-per-node 1 \
  --placement-pref 'spread=node.labels.datacenter' \
  nginx
```

other options available are: create/inspect/ps/ls/rm/update.

Docker start/stop

These commands are used to start or stop a single or multiple containers.

```
$ docker start my_container
$ docker stop my_container
```

Docker swarm:

This is an orchestration tool used to manage multiple containers deployed across multiple host machines.
This is a world within docker. But for now we can go ahead with basic few commands :

Docker swarm ca

This command is used to view the current root CA certificate. In case if any node managers are compromised and no longer can be trusted, this certificate can be rotated with command “Docker swarm ca –rotate”

```
$ docker swarm ca --rotate
desired root digest: sha256:05da740cf2577a25224c53019e2cce99bcc5ba09664ad6bb2a9425d9ebd1b53e
  rotated TLS certificates:  [=====================================================>] 2/2 nodes
  rotated CA certificates:   [=====================================================>] 2/2 nodes
```

```
-----BEGIN CERTIFICATE-----
MIIBazCCARCgAwIBAgIUFynG04h5Rrl4lKyA4/E65tYKg8IwCgYIKoZIzj0EAwIw
EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTcwNTE2MDAxMDAwWhcNMzcwNTExMDAx
MDAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABC2DuNrIETP7C7lfiEPk39tWaaU0I2RumUP4fX4+3m+87j0DU0CsemUaaOG6
+PxHhGu2VXQ4c9pctPHgf7vWeVajQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBSEL02z6mCI3SmMDmITMr12qCRY2jAKBggqhkjO
PQQDAgNJADBGAiEA263Eb52+825EeNQZM0AME+aoH1319Zp9/J5ijILW+6ACIQCg
gyg5u9Iliel99l7SuMhNeLkrU7fXs+Of1nTyyM73ig==
-----END CERTIFICATE-----
```

Along with detach, the progress of the rotation won’t be displayed.

Docker swarm init:

This is used to initialize the swarm orchestration.

```
$ docker swarm init --advertise-addr 192.168.99.121


Swarm initialized: current node (bvz81updecsj6wjz393c09vti) is now a manager.


To add a worker to this swarm, run the following command:


    docker swarm join \
    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-1awxwuwd3z9j1z3puu7rcgdbx \
    172.17.0.2:2377


To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Docker swarm join-token

Join tokens are secrets that allow a node to the swarm.

```
$docker swarm join-token worker
To add a worker to this swarm, run the following command:
    docker swarm join \
    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-1awxwuwd3z9j1z3puu7rcgdbx \
    172.17.0.2:2377

$ docker swarm join-token manager
To add a manager to this swarm, run the following command:
    docker swarm join \
    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-7p73s1dx5in4tatdymyhg9hu2 \
    172.17.0.2:2377


$ docker swarm join-token --rotate worker

Successfully rotated worker join token.

To add a worker to this swarm, run the following command:

$ docker swarm join \

    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-b30ljddcqhef9b9v4rs7mel7t \

    172.17.0.2:2377
```

Docker swarm join:

This is similar to join-token but without the secrets.If required, can be specified using –token.

```
$ docker swarm join --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-
1awxwuwd3z9j1z3puu7rcgdbx 192.168.99.121:2377

This node joined a swarm as a worker.


$ docker node ls

ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS

7ln70fl22uw2dvjn2ft53m3q5    worker2   Ready   Active

dkp8vy1dq1kxleu9g4u78tlag    worker1   Ready   Active        Reachable

dvfxp4zseq4s0rih1selh0d20 *  manager1  Ready   Active        Leader
```

Docker swarm leave:

This is used to leave swarm.

```
$ docker swarm leave

Node left the default swarm.
```

Docker swarm unlock-key:

This command is used to unlock a swarm manager node once the docker daemon is restarted.

```
$ docker swarm unlock-key

To unlock a swarm manager after it restarts, run the `docker swarm unlock`

command and provide the following key:

    SWMKEY-1-fySn8TY4w5lKcWcJPIpKufejh9hxx5KYwx6XZigx3Q4

Please remember to store this key in a password manager, since without it you

will not be able to restart the manager.



$ docker swarm unlock-key --rotate
```

```
Successfully rotated manager unlock key.

To unlock a swarm manager after it restarts, run the `docker swarm unlock`

command and provide the following key:

    SWMKEY-1-7c37Cc8654o6p38HnroywCi19pllOnGtbdZEgtKxZu8

Please remember to store this key in a password manager, since without it you

will not be able to restart the manager.
```

Docker swarm unlock:

This is used to unlock a manager using a user-supplied unlock keys.

```
$ docker swarm unlock

Please enter unlock key:
```

Docker swarm update:

This is used to update swarm with new parameters.

```
$ docker swarm update --cert-expiry 720h
```

Docker Volume

This command is used to create/inspect/prune/remove the volumes.

Create:

```
$ docker volume create --driver local \

    --opt type=tmpfs \

    --opt device=tmpfs \

    --opt o=size=100m,uid=1000 \

    Foo


$ docker volume create --driver local \

    --opt type=btrfs \

    --opt device=/dev/sda2 \

    Foo


docker volume create --driver local \

    --opt type=nfs \

    --opt o=addr=192.168.1.1,rw \

    --opt device=:/path/to/dir \

    foo
```

inspect:

used to inspect configurations of a volume in the system.

```
$ docker volume create myvolume

myvolume


$ docker volume inspect myvolume

[

  {

    "CreatedAt": "2020-04-19T11:00:21Z",

    "Driver": "local",
    "Labels": {},

    "Mountpoint":
"/var/lib/docker/volumes/8140a838303144125b4f54653b47ede0486282c623c3551fbc7f390cdc3e9cf5/_data",

    "Name": "myvolume",

    "Options": {},

    "Scope": "local"

  }

]


$ docker volume inspect --format '{{ .Mountpoint }}' myvolume

/var/lib/docker/volumes/myvolume/_data
```

ls:

Lists all the volumes

```
$ docker volume create rosemary


rosemary


$ docker volume create tyler


tyler


$ docker volume ls


DRIVER              VOLUME NAME

local               rosemary

local               tyler
```

prune:

removes unused volumes

```
$ docker volume prune


WARNING! This will remove all local volumes not used by at least one container.

Are you sure you want to continue? [y/N] y

Deleted Volumes:

07c7bdf3e34ab76d921894c2b834f073721fccfbbcba792aa7648e3a7a664c2e

my-named-vol
```

```
Total reclaimed space: 36 B
```

rm:

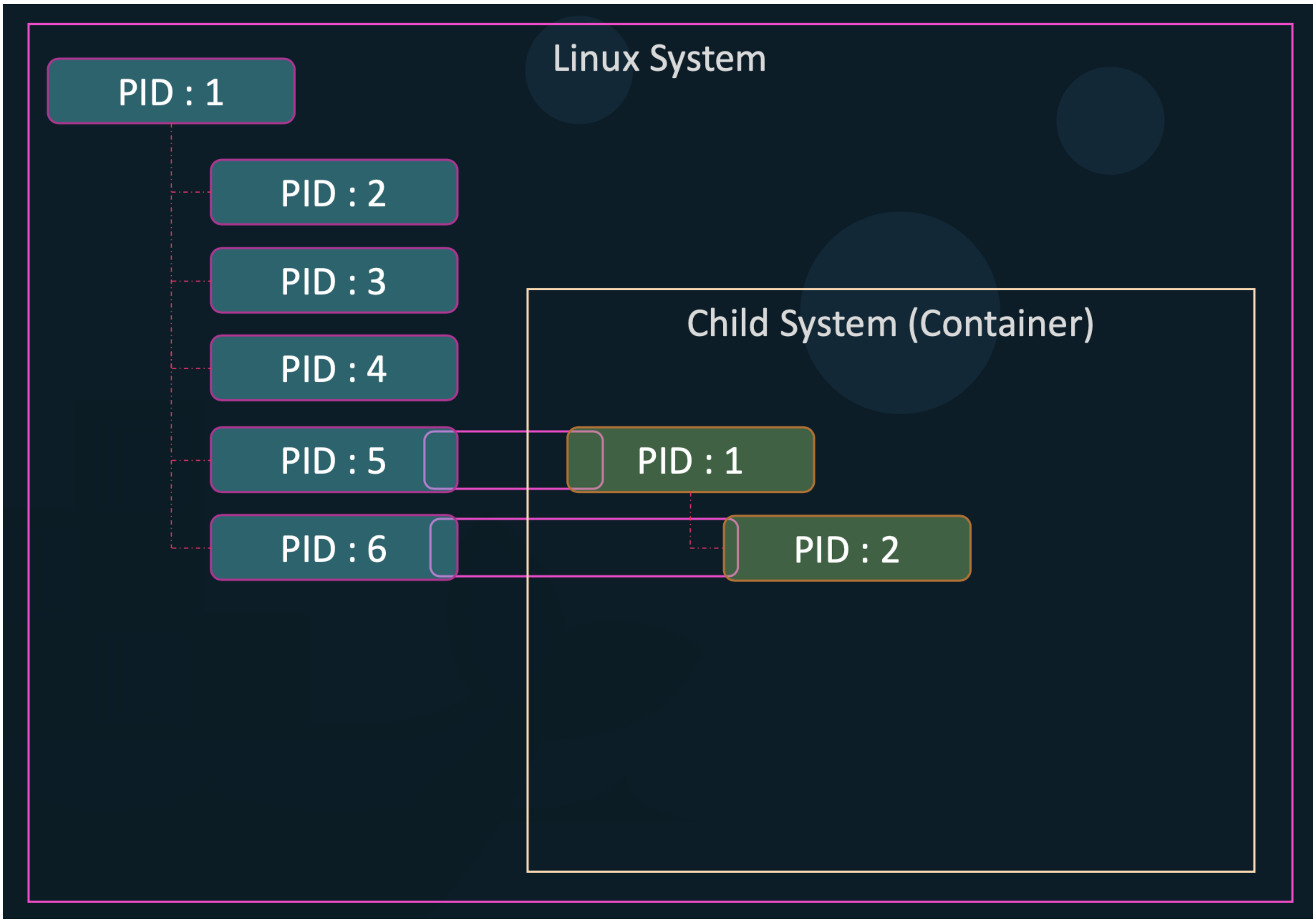
Removes a specific volume or volumes.

```
$ docker volume rm -f hello

hello
```

DOCKER ENGINE:

***Please note : Docker container PID mostly will be different on host to that of the container.


Linux System
PID : 1
PID : 2
PID : 3
PID : 4
PID : 5
PID : 6
Child System (Container)
PID : 1
PID : 2