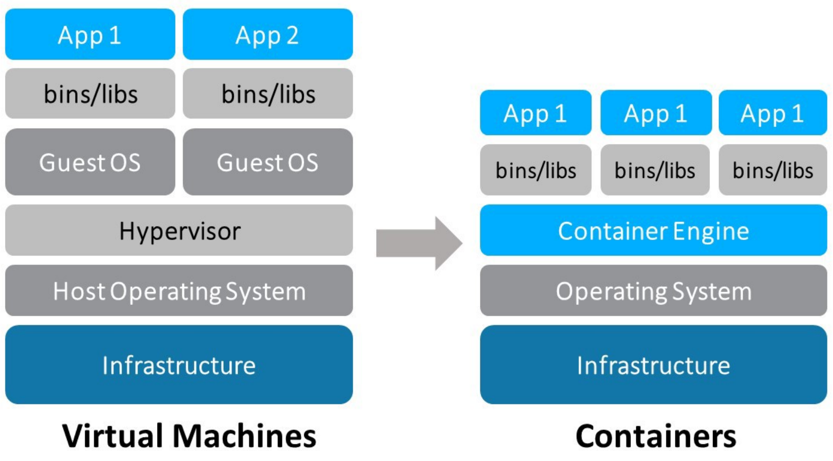
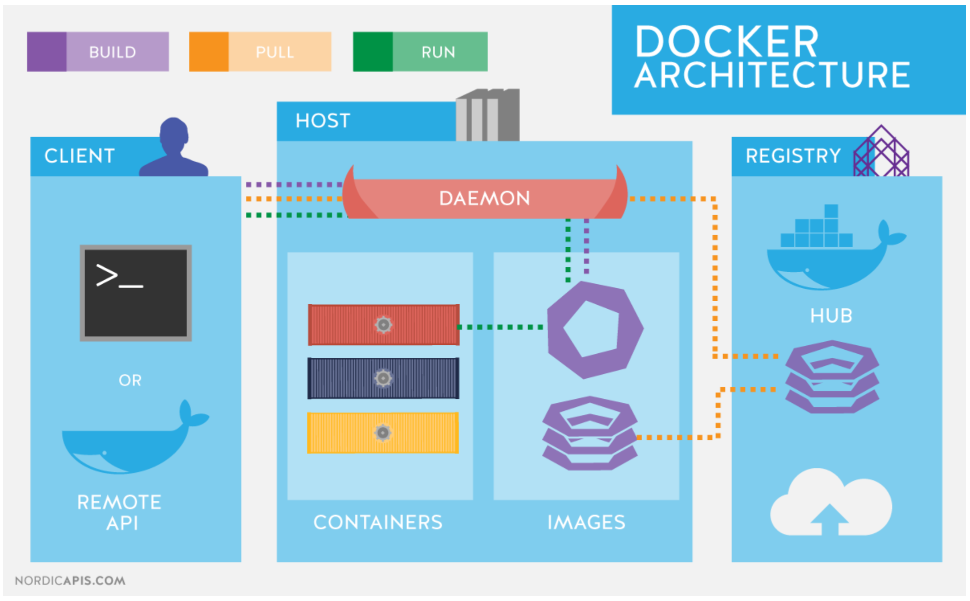
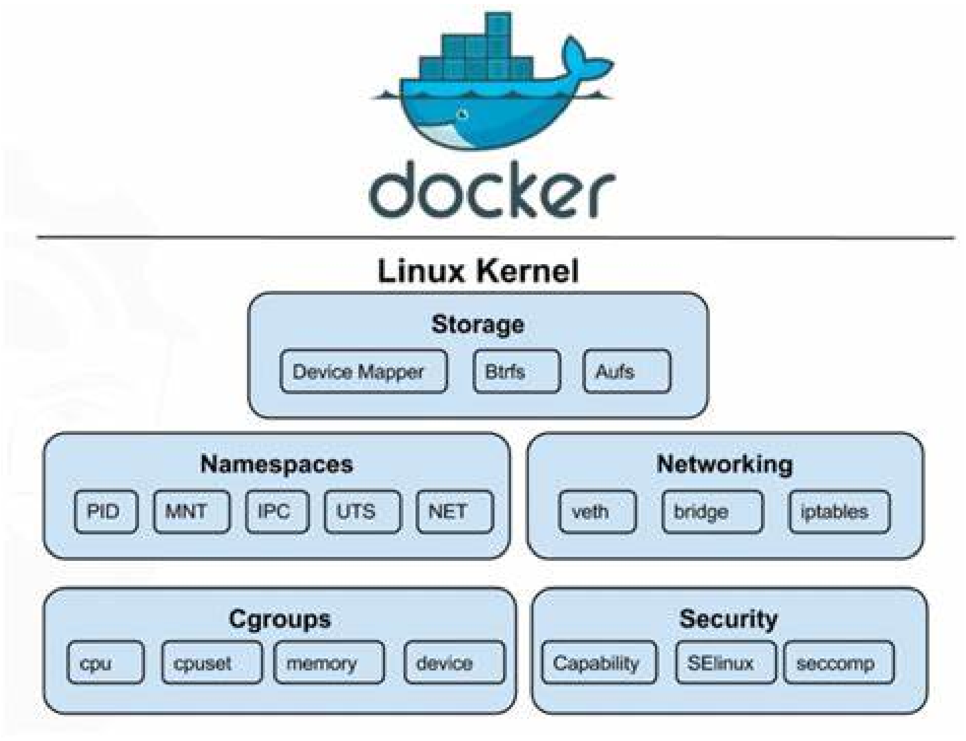
Memo - DevOps - Docker

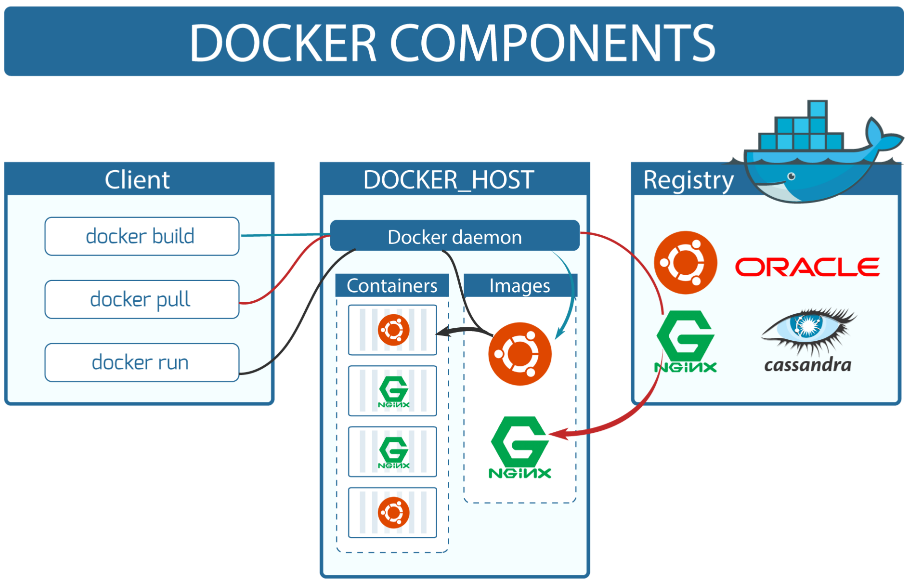
Docker Containers vs Virtual Machines



Docker Architecture



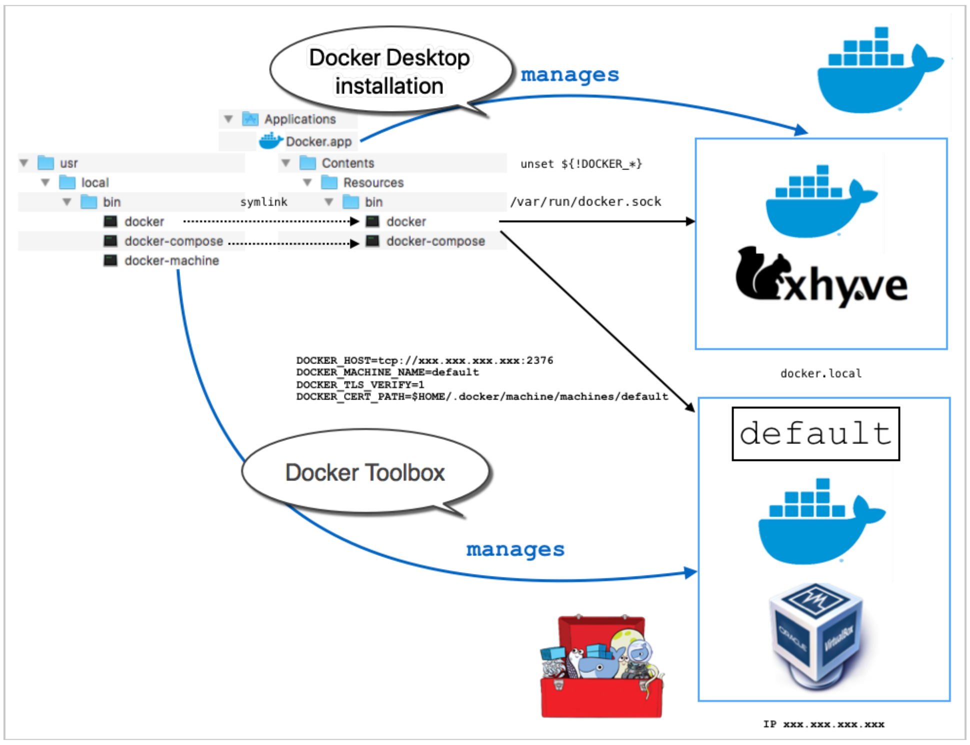




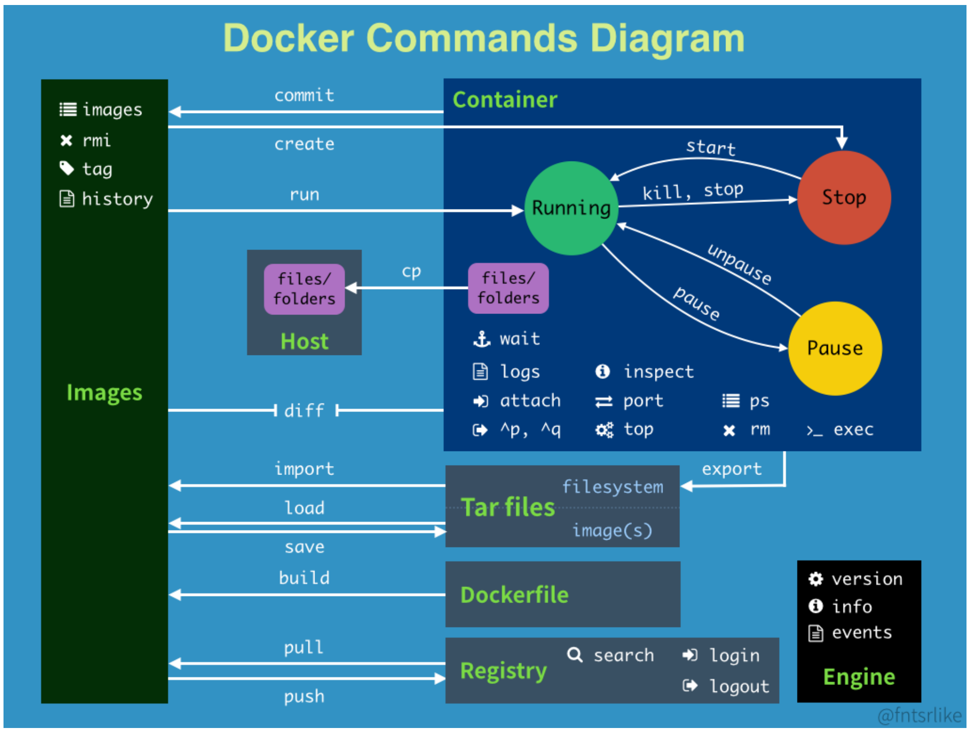
Docker Installation

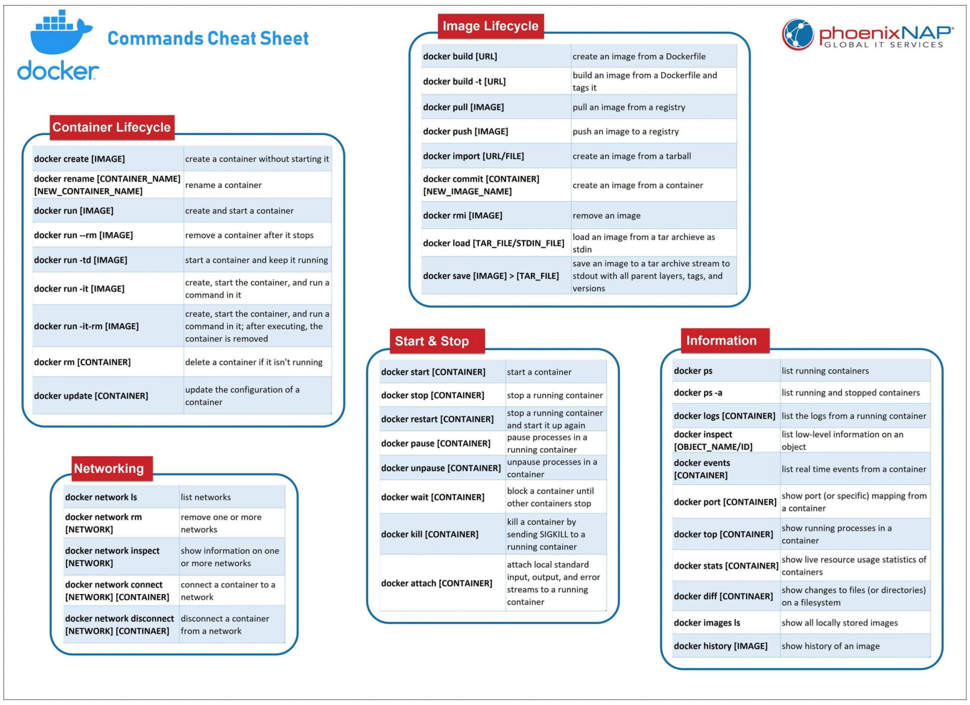
- Pre-requisiters
- `pip install docker`

-> CMD: docker --version
-> CMD: docker --help
-> CMD: docker run hello-world



Docker Commands




docker
phoenixNAP GLOBAL IT SERVICES

Commands Cheat Sheet

Container Lifecycle

docker create [IMAGE]	create a container without starting it
docker rename [CONTAINER_NAME] [NEW_CONTAINER_NAME]	rename a container
docker run [IMAGE]	create and start a container
docker run --rm [IMAGE]	remove a container after it stops
docker run -td [IMAGE]	start a container and keep it running
docker run -it [IMAGE]	create, start the container, and run a command in it
docker run -it-rm [IMAGE]	create, start the container, and run a command in it; after executing, the container is removed
docker rm [CONTAINER]	delete a container if it isn't running
docker update [CONTAINER]	update the configuration of a container

Image Lifecycle

docker build [URL]	create an image from a Dockerfile
docker build -t [URL]	build an image from a Dockerfile and tags it
docker pull [IMAGE]	pull an image from a registry
docker push [IMAGE]	push an image to a registry
docker import [URL/FILE]	create an image from a tarball
docker commit [CONTAINER] [NEW_IMAGE_NAME]	create an image from a container
docker rmi [IMAGE]	remove an image
docker load [TAR_FILE/STDIN_FILE]	load an image from a tar archieve as stdin
docker save [IMAGE] > [TAR_FILE]	save an image to a tar archive stream to stdout with all parent layers, tags, and versions

Start & Stop

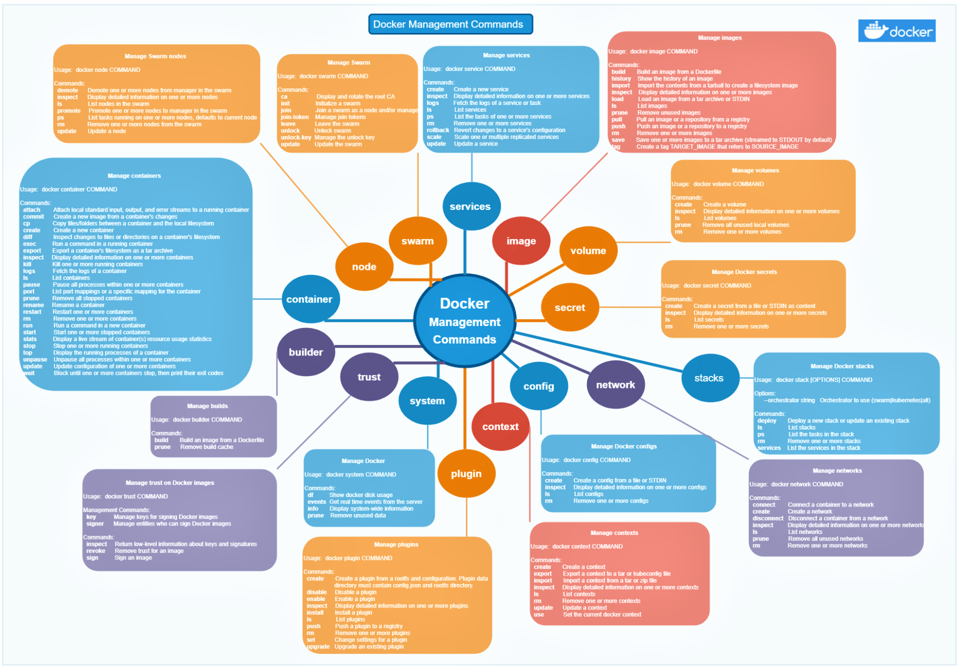
docker start [CONTAINER]	start a container
docker stop [CONTAINER]	stop a running container
docker restart [CONTAINER]	stop a running container and start it up again
docker pause [CONTAINER]	pause processes in a running container
docker unpause [CONTAINER]	unpause processes in a container
docker wait [CONTAINER]	block a container until other containers stop
docker kill [CONTAINER]	kill a container by sending SIGKILL to a running container
docker attach [CONTAINER]	attach local standard input, output, and error streams to a running container

Information

docker ps	list running containers
docker ps -a	list running and stopped containers
docker logs [CONTAINER]	list the logs from a running container
docker inspect [OBJECT_NAME/ID]	list low-level information on an object
docker events [CONTAINER]	list real time events from a container
docker port [CONTAINER]	show port (or specific) mapping from a container
docker top [CONTAINER]	show running processes in a container
docker stats [CONTAINER]	show live resource usage statistics of containers
docker diff [CONTAINER]	show changes to files (or directories) on a filesystem
docker images ls	show all locally stored images
docker history [IMAGE]	show history of an image

Networking

docker network ls	list networks
docker network rm [NETWORK]	remove one or more networks
docker network inspect [NETWORK]	show information on one or more networks
docker network connect [NETWORK] [CONTAINER]	connect a container to a network
docker network disconnect [NETWORK] [CONTAINER]	disconnect a container from a network



Docker Management Commands

```
Manage Swarm nodes

Usage:  docker node COMMAND

Commands:
  demote      Demote one or more nodes from manager in the swarm
  inspect     Display detailed information on one or more nodes
  ls          List nodes in the swarm
  promote     Promote one or more nodes to manager in the swarm
  ps          List tasks running on one or more nodes, defaults to current node
  rm          Remove one or more nodes from the swarm
  update      Update a node
```

```
Manage Swarm

Usage:  docker swarm COMMAND

Commands:
  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm
```

```
Manage services

Usage:  docker service COMMAND

Commands:
  create      Create a new service
  inspect     Display detailed information on one or more services
  logs        Fetch the logs of a service or task
  ls          List services
  ps          List the tasks of one or more services
  rm          Remove one or more services
  rollback    Revert changes to a service's configuration
  scale       Scale one or multiple replicated services
  update      Update a service
```

```
Manage images

Usage:  docker image COMMAND

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

```
Manage containers

Usage:  docker container COMMAND

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes
```

```
Manage volumes

Usage:  docker volume COMMAND

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

```
Manage Docker secrets

Usage:  docker secret COMMAND

Commands:
  create      Create a secret from a file or STDIN as content
  inspect     Display detailed information on one or more secrets
  ls          List secrets
  rm          Remove one or more secrets
```

```
Manage Docker stacks

Usage:  docker stack [OPTIONS] COMMAND

Options:
    --orchestrator string   Orchestrator to use (swarm|kubernetes|all)

Commands:
  deploy      Deploy a new stack or update an existing stack
  ls          List stacks
  ps          List the tasks in the stack
  rm          Remove one or more stacks
  services    List the services in the stack
```

```
Manage builds

Usage:  docker builder COMMAND

Commands:
  build       Build an image from a Dockerfile
  prune       Remove build cache
```

```
Manage trust on Docker images

Usage:  docker trust COMMAND

Management Commands:
  key         Manage keys for signing Docker images
  signer      Manage entities who can sign Docker images

Commands:
  inspect     Return low-level information about keys and signatures
  revoke      Remove trust for an image
  sign        Sign an image
```

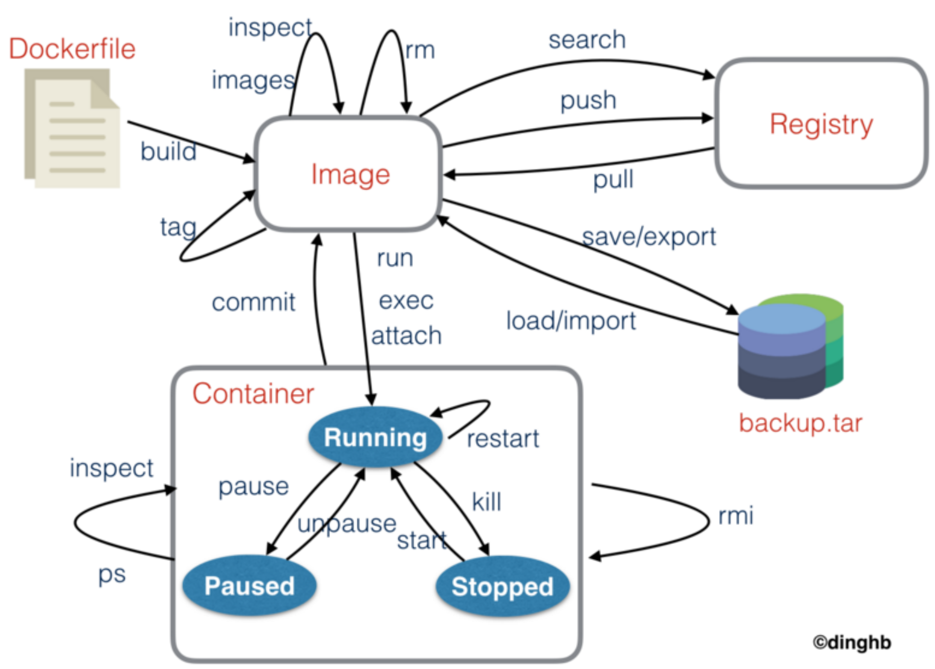
```
Manage Docker

Usage:  docker system COMMAND

Commands:
  df          Show docker disk usage
  events      Get real time events from the server
  info        Display system-wide information
  prune       Remove unused data
```

```
Manage Docker configs

Usage:  docker config COMMAND

Commands:
  create      Create a config from a file or STDIN
  inspect     Display detailed information on one or more configs
  ls          List configs
  rm          Remove one or more configs
```

```
Manage networks

Usage:  docker network COMMAND

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks
```

```
Manage plugins

Usage:  docker plugin COMMAND

Commands:
  create      Create a plugin from a rootfs and configuration. Plugin data
              directory must contain config.json and rootfs directory.
  disable     Disable a plugin
  enable      Enable a plugin
  inspect     Display detailed information on one or more plugins
  install     Install a plugin
  ls          List plugins
  push        Push a plugin to a registry
  rm          Remove one or more plugins
  set         Change settings for a plugin
  upgrade     Upgrade an existing plugin
```

```
Manage contexts

Usage:  docker context COMMAND

Commands:
  create      Create a context
  export      Export a context to a tar or kubeconfig file
  import      Import a context from a tar or zip file
  inspect     Display detailed information on one or more contexts
  ls          List contexts
  rm          Remove one or more contexts
  update      Update a context
  use         Set the current docker context
```



Docker Image Commands

- docker search `images` (from docker hub)
- docker pull `images`

-> docker images

-> docker image ls -> docker image history

-> docker rmi `Repository`

-> docker rmi `images_id`

-> docker rmi `images_id` --force

-> docker pull ubuntu

-> docker images

-> docker create ubuntu

-> docker ps

-> docker ps -a

-> docker rm `CONTAINER_ID`

-> docker start `CONTAINER_NAME`

-> docker images

-> docker rmi ubuntu

Docker Container Commands

- docker ps: `list running containers`
- docker ps -a : `list running containers and stopped containers`
- docker ps -aq : `list running containers ID and stopped containers ID`

- Format

> ```
> -> docker ps -- format=
> "ID\t{{.ID}}\nNAME\t{{.Names}}\nIMAGE\t{{.Image}}\nPORTS\t{{.Ports}}\n
> COMMAND\t{{.Command}}\nCREATE\t{{.CreatedAt}}\nSTATUS\t{{.Status}}\n"
> ```

> -> docker rm `CONTAINER_ID`
> -> docker rm `CONTAINER_NAME`
> -> docker rm $(docker ps -aq)
>
> -> docker stop `CONTAINER_ID`
> -> docker stop `CONTAINER_NAME`
>
> -> docker start `CONTAINER_ID`
> -> docker start `CONTAINER_NAME`

> ## run
>
> -> docker images
> -> docker run ubuntu (`pull image` + `create contain` + `start contain`)
> -> docker images
> -> docker ps -a
> -> docker run ubuntu ls (ls in the container ubuntu)
> -> docker ps -a
>
> ```
> -d, --detach             Run container in background and print
> container ID
> -e, --env list           Set environment variables
> -h, --hostname string    Container host name
> -i, --interactive        Keep STDIN open even if not attached
> -p, --publish list       Publish a container's port(s) to the
> host
> -P, --publish-all        Publish all exposed ports to random
> ports
> -t, --tty                Allocate a pseudo-TTY
> -v, --volume list        Bind mount a volume
>     --volumes-from list  Mount volumes from the specified
> container(s)
> ```

> ## Naming
>
> -> docker run --name postgres1010 -d -p 6003:5432 postgres:10.10

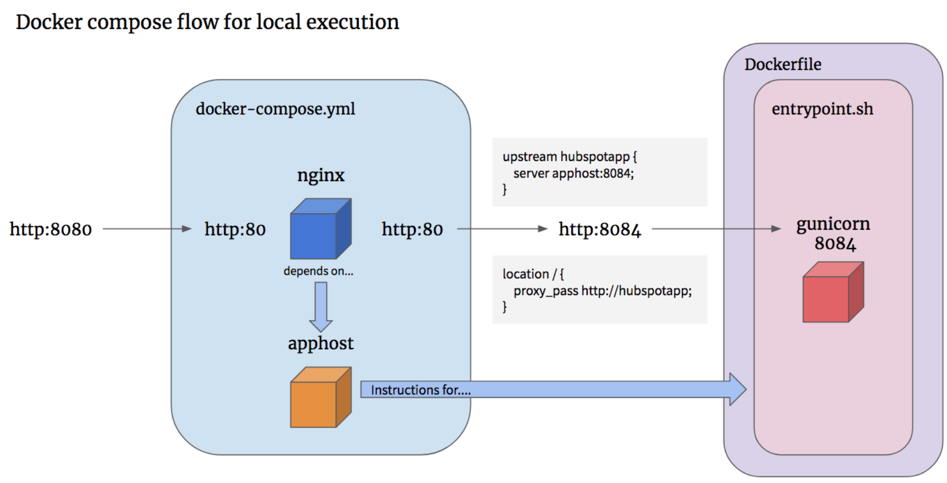
> ## Docker Exec
>
> -> docker run -it `REPOSITORY:TAG` bin/sh
> -> #
>
> -> docker run -it -d -p 9000:80 `REPOSITORY:TAG` bin/sh

-> docker exec -it `contain_id` bash
-> docker exec -it `contain_id` /bin/sh
-> ~ # ls
-> ~ # ls -al
-> ~ # pwd
-> ~ # env
-> ~ # exit
-> ls

Docker Logs

-> docker logs `contain_id`
-> docker logs `CONTAINER_NAMES`
-> docker logs -f `contain_id` (follow log outputs)

Run with Shared Port



```
upstream hubspotapp {
  server apphost:8084;
}
```

```
location / {
  proxy_pass http://hubspotapp;
}
```

-> docker run -it ubuntu
-> # apt-get update
-> # apt-get install nginx
-> docker inspect `CONTAINER_ID` (find the IPAddress)
-> docker ps -a (container run)
-> # exit
-> docker ps -a (container exited)

-> docker run -it -p 9000:80 ubuntu
-> # apt-get update && apt-get install nginx -y
-> # nginx -v
-> # sevice nginx start
-> Chrome: localhost:9000

> -> # cd /var/www/html
> -> # ll

> -> # apt-get install vim
> -> # vim index.nginx-debian.html
> -> `vim` : i for insert, : for commnad lind, wq for save and exit()

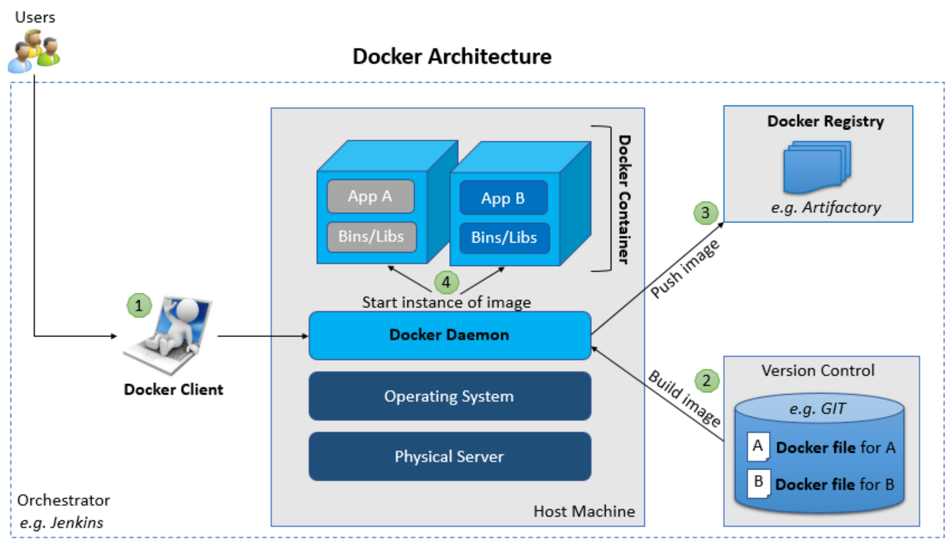
> -> docker run -it ubuntu
> -> # apt-get update
> -> # apt-get install nginx
> -> docker inspect `CONTAINER_ID` (find the IPAddress)
> -> docker ps -a (container run)
> -> # exit
> -> docker ps -a (container exited)

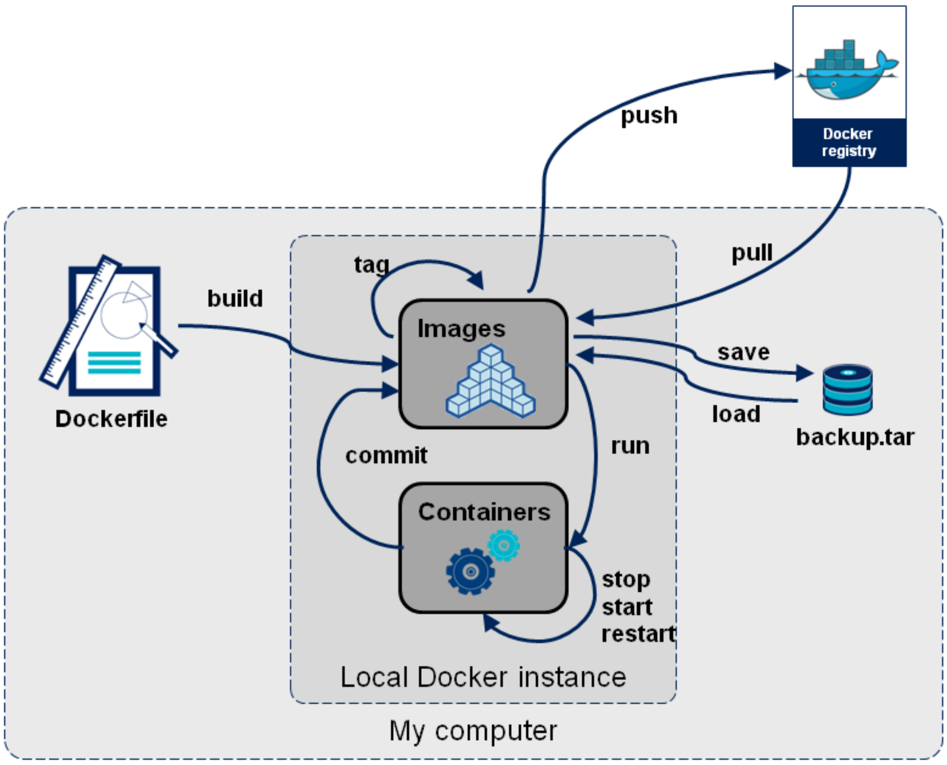
> -> docker run -it -p 9000:80 ubuntu
> -> # apt-get update && apt-get install nginx -y
> -> # nginx -v
> -> # sevice nginx start
> -> Chrome: localhost:9000

> -> # cd /var/www/html
> -> # ll

> -> # apt-get install vim
> -> # vim index.nginx-debian.html
> -> `vim` : i for insert, : for commnad lind, wq for save and exit()

Docker Workflow





Ceate Dockerfile

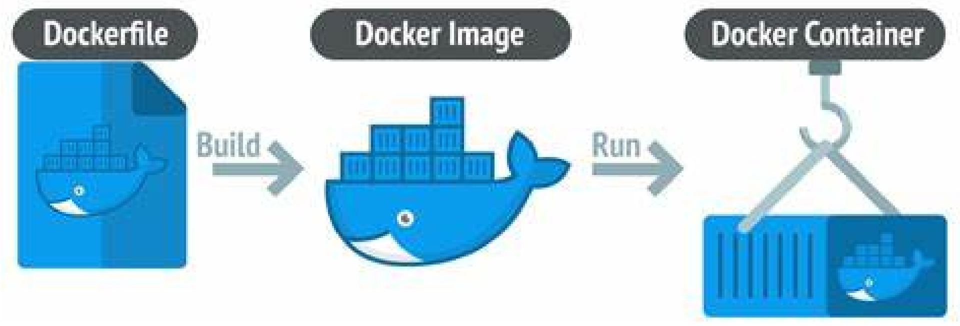
```
FROM ubuntu
RUN apt-get update
RUN apt-get install nginx -y
```

`CMD vs ENTRYPOINT`

Create Image

- docker build -t `REPOSITORY:TAG` .

Create Container

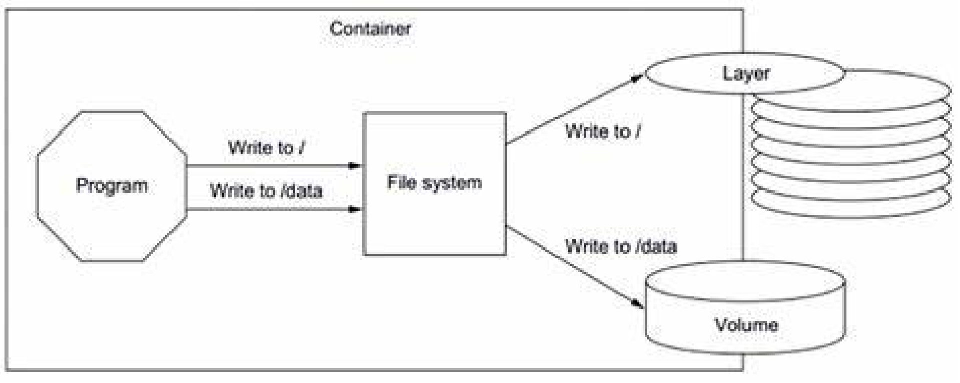


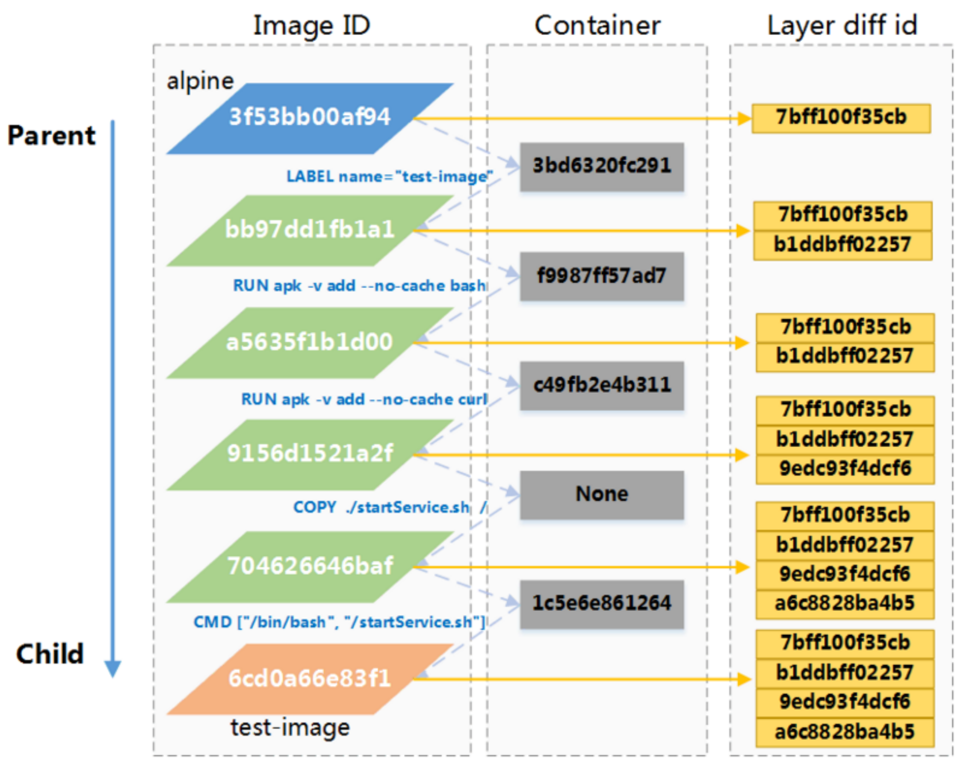
- docker run --name `container` `REPOSITORY:TAG` .
- daemon off

```
FROM ubuntu
RUN apt-get update
RUN apt-get install nginx -y
CMD ["nginx", "-g", "deamon off;"]
```

-> docker run -it -d -p 9000:80 `REPOSITORY:TAG`

Caching and Layers





Start/Stop Container

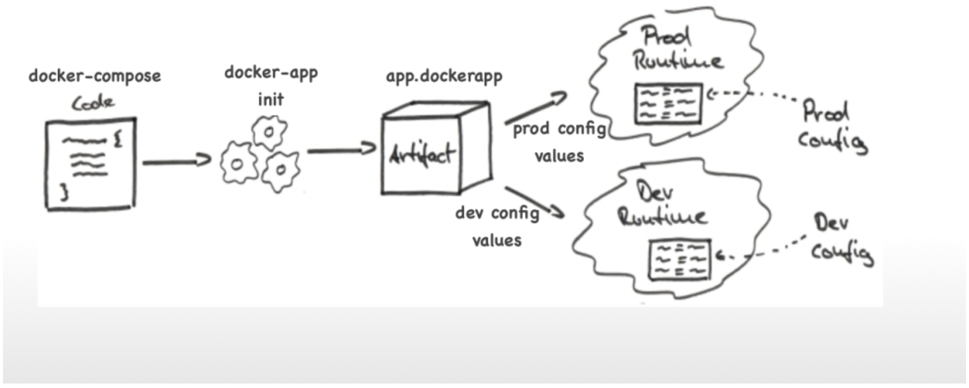
-> docker stop `CONTAINER_ID`

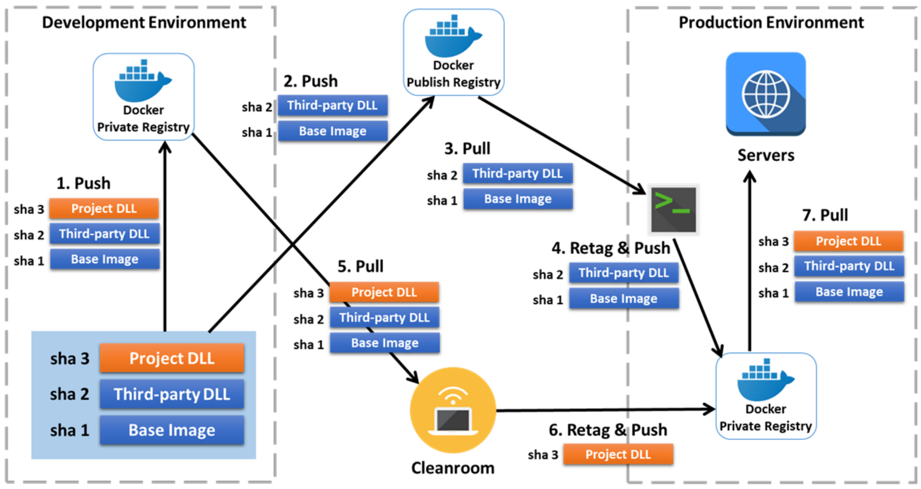
-> docker start `CONTAINER_ID`

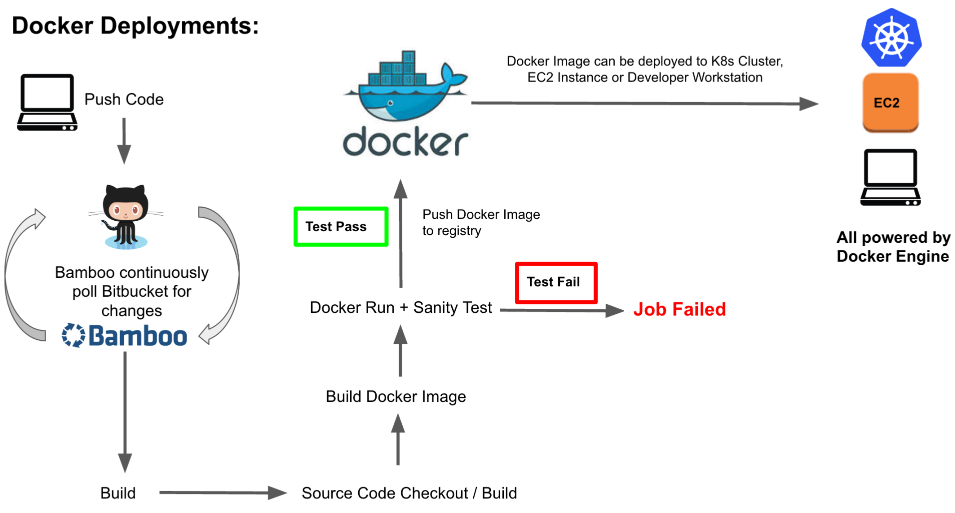
Ceate docker-compose

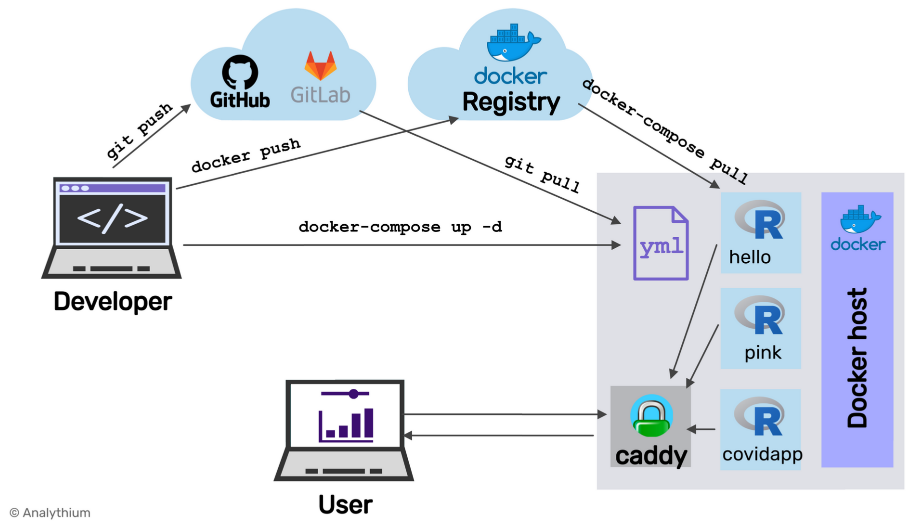
- docker-compose version
- Create `docker-compose.yml`

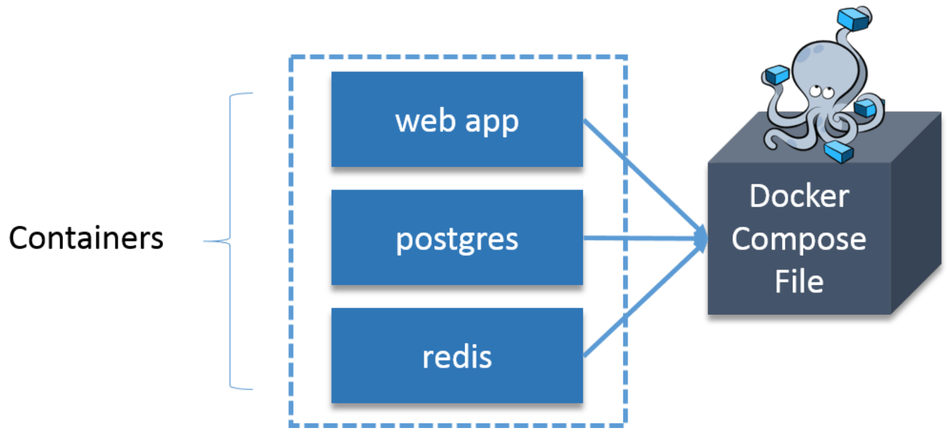
- docker-compose up
- docker-compose down

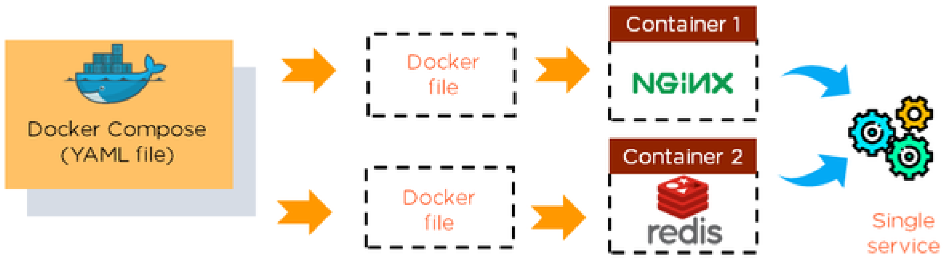


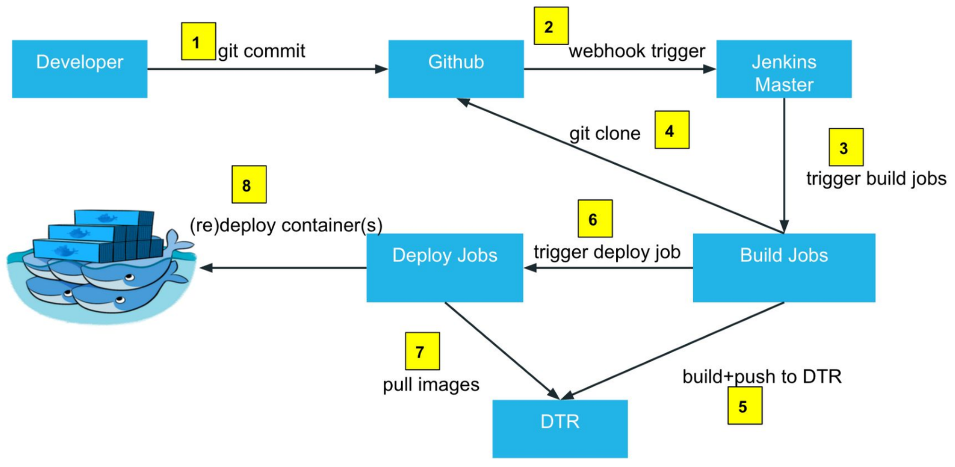



Docker Deployments:
Push Code
Bamboo continuously poll Bitbucket for changes
Bamboo
Build
Source Code Checkout / Build
Build Docker Image
Docker Run + Sanity Test
Test Pass
Push Docker Image to registry
Test Fail
Job Failed
docker
Docker Image can be deployed to K8s Cluster, EC2 Instance or Developer Workstation
EC2
All powered by Docker Engine

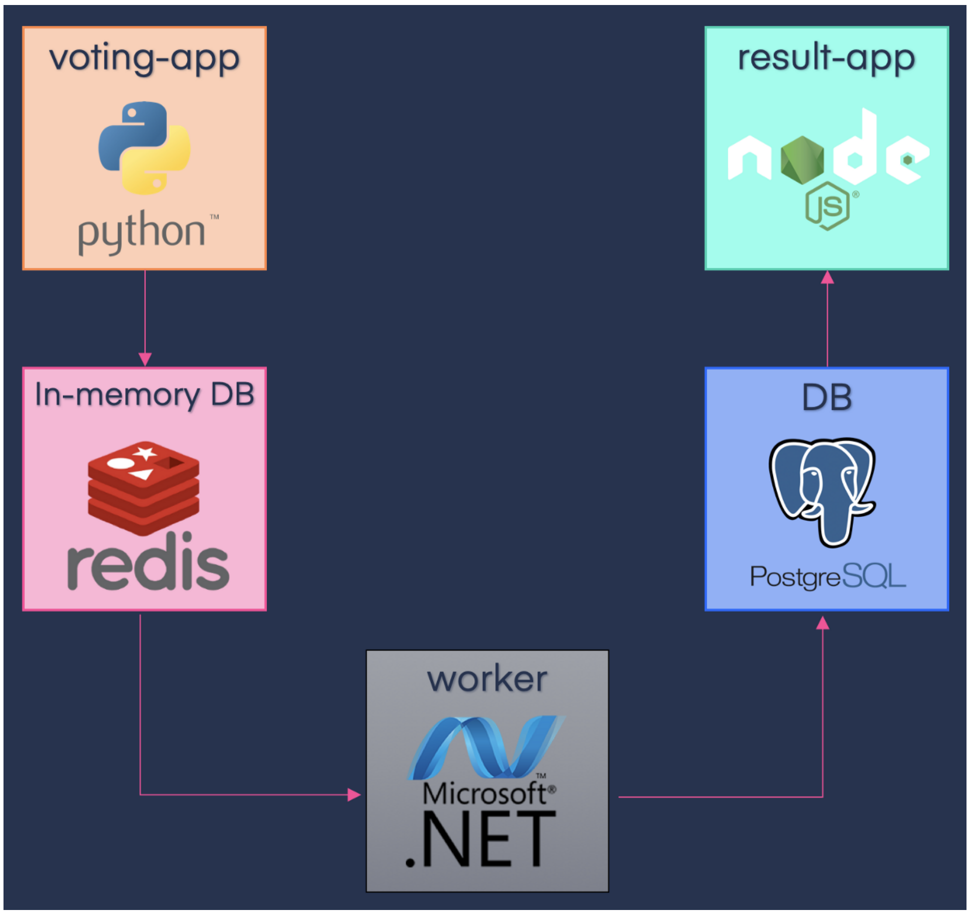








Example - voting app



- docker containers

```
docker run -d --name=rdeis rdis
docker run -d --name=db --link db:db postgres:9.4
docker run -d --name=vote -p 5000:80 --link redis:redis
voting-app
docker run -d --name=result -p 5001:80 result-app
docker run -d --name=worker --link db:db redis:redis worker
```

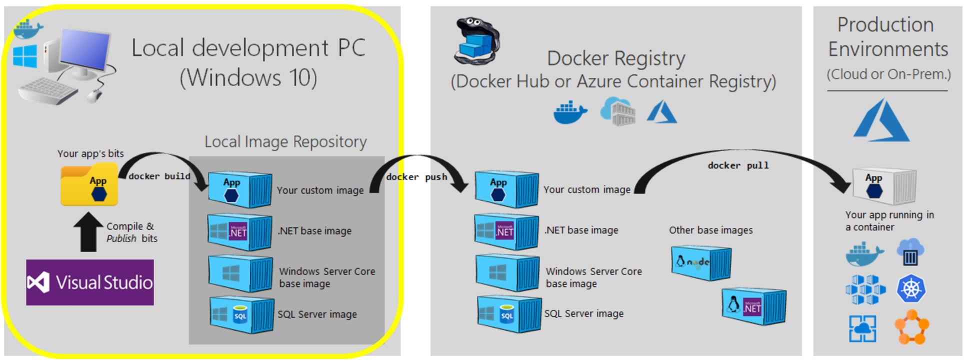
- Create docker-compsose.yml

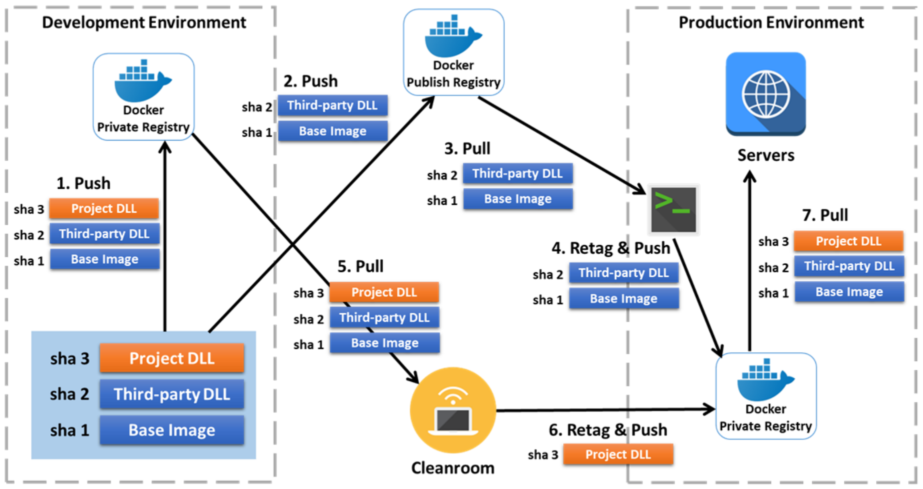
```
Service:
  redis:
    image: redis
    networks:
     back-end
  db:
    image: postgres:9.4
    networks:
     back-end
  vote:
    image: voting-app
    ports:
      - 5000:80
    links:
      - redis
    networks:
     front-end
     back-end
  result:
    image: result-app
    ports:
     - 5001:80
    links:
     - db
    networks:
      front-end
      back-end
 worker:
    image: worker
    links:
     - redis
     - db
networks:
  front-end:
  back-end:
```

> ### - docker-compsose.yml build

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  build: ./vote
  ports:
    - 5000:80
  links:
    - redis
result:
  build: ./result
  ports:
   - 5001:80
 links:
   - db
worker:
  build: ./worker
  links:
    - redis
    - db
```

Docker Registry


Local development PC (Windows 10)
Local Image Repository
Your app's bits
docker build
Compile & Publish bits
Visual Studio
Your custom image
.NET base image
Windows Server Core base image
SQL Server image
docker push
Docker Registry (Docker Hub or Azure Container Registry)
Other base images
docker pull
Production Environments (Cloud or On-Prem.)
Your app running in a container



- Docker Hub

Docker Repositoies

- Create Repository

`docker tag local-image:tagname new-repo:tagname`

`docker push new-repo:tagname`

- Push Repository

-> docker tag website:copylearning ericarthuang/website:copylearning
-> docker images
-> docker push ericarthuang/website:copylearning

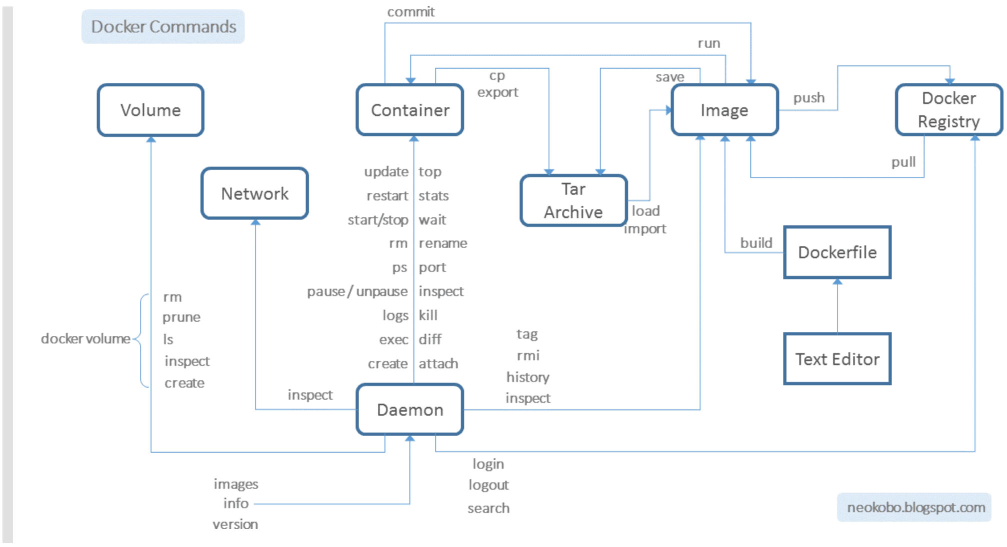
- Pull Image from Repository

-> docker pull ericarthuang/website:copylearning

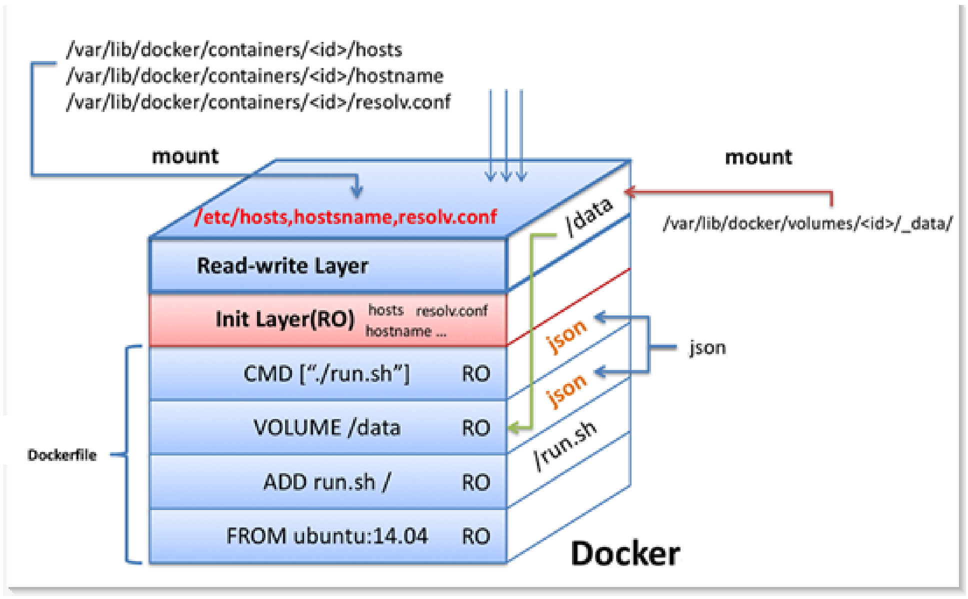
- Create Container

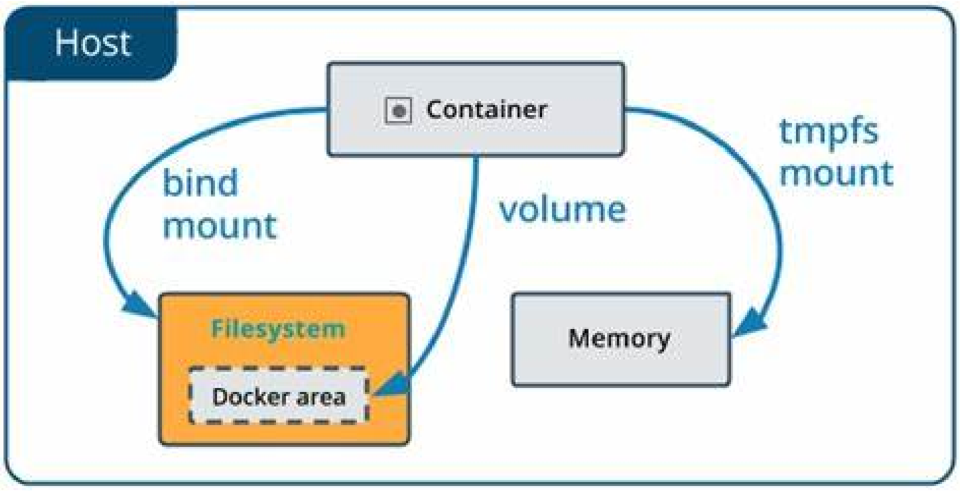
-> docker run --name website_copylearning -d -p 8080:80 ericarthuang/website:copylearning
->
47ec4e1f3d011c9401712196358e27c9b7e6f1e4d355480b641433f90d47e5b7

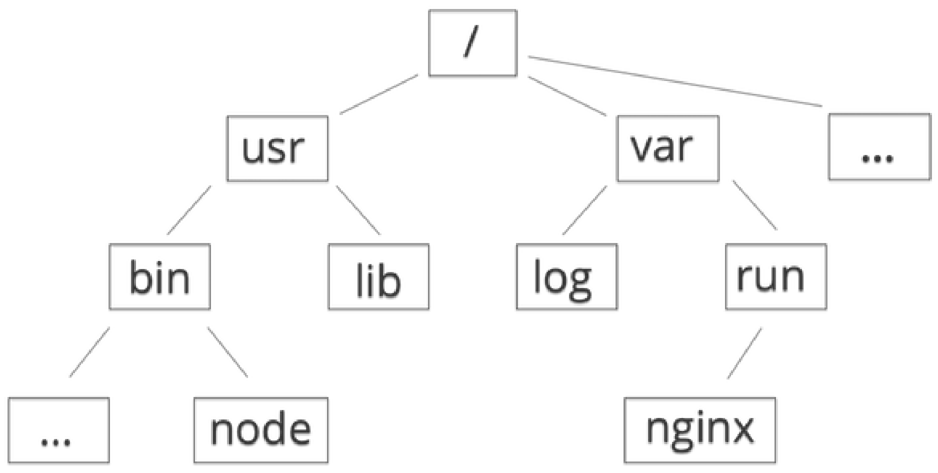
Docker Volumes



- Volume Mounting vs Bind Mounting


/var/lib/docker/containers/<id>/hosts
/var/lib/docker/containers/<id>/hostname
/var/lib/docker/containers/<id>/resolv.conf
mount
mount
/etc/hosts,hostname,resolv.conf
/data
/var/lib/docker/volumes/<id>/_data/
Read-write Layer
Init Layer(RO)
hosts resolv.conf
hostname ...
json
json
json
CMD ["./run.sh"] RO
VOLUME /data RO
/run.sh
Dockerfile
ADD run.sh / RO
FROM ubuntu:14.04 RO
Docker





- Volume Mounting

Create volume

-> docker volume create demo-volume

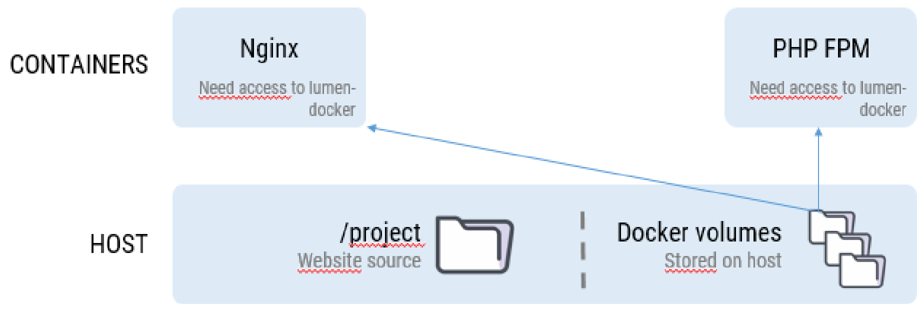
-> docker volume ls

-> docker volume inspect demo-volume

```
[
    {
        "CreatedAt": "2022-11-02T16:31:38Z",
       "Driver": "local",
       "Labels": {},
       "Mountpoint": "/var/lib/docker/volumes/demo-
volume/_data",
       "Name": "demo-volume",
       "Options": {},
       "Scope": "local"
   }
]
```

-> docker volume ls

-> docker prune

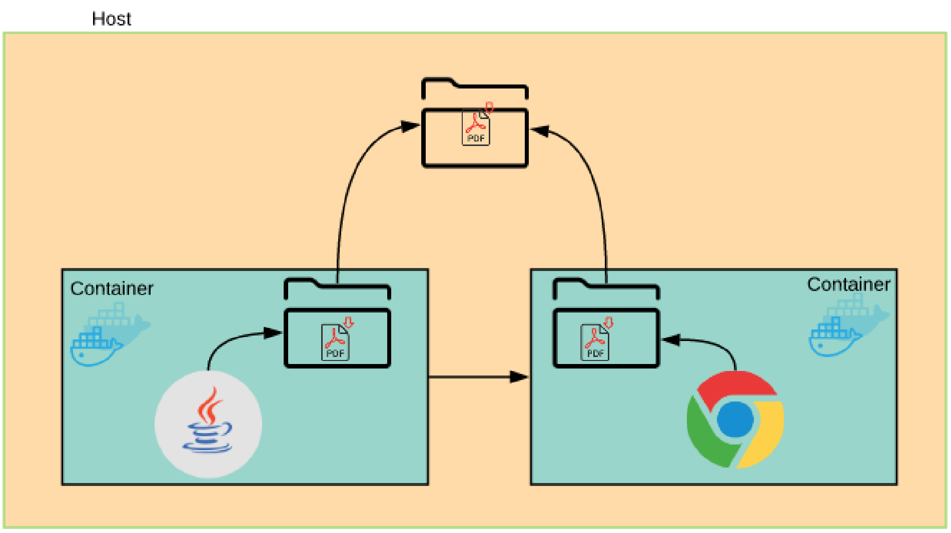


- Sharing Volumes Between Containers

- docker run

-> docker run --name website-copy `--volumes-from` website -d -p 9001:80 nginx
-> 7b6a3ffa6cc4a0fd00430241ba2dfc1731e569ddf57150e37e94bff83b926de5

```
-v, --volume list              Bind mount a volume
    --volume-driver string     Optional volume driver for the
container
    --volumes-from list        Mount volumes from the specified
```

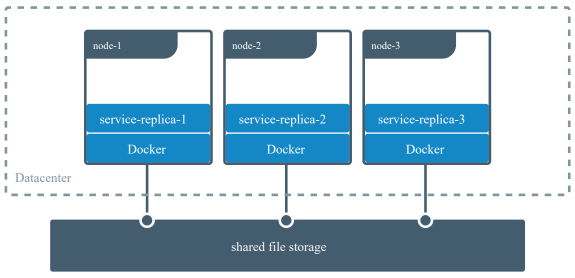


- Bind Mounting: Run with Shared Volumes

```
-> docker run -it -d -p 9001:80 ubuntu
->
c9e1a90ab4432c80e1952e9966f7079b3cdb020f9c84f462ef0e8ed686a138db
```
-> docker exec c9 apt-get update
-> docker exec c9 apt-get install nignx -y
-> docker exec c9 service nginx start
-> docker exec c9 ls
-> docker exec c9 ls /var/www/html

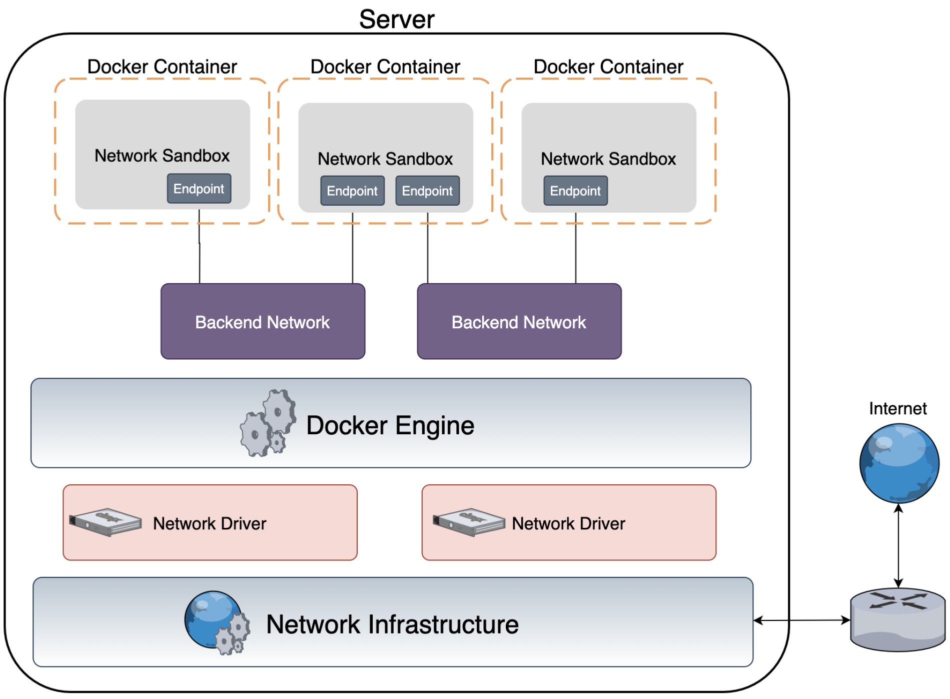
docker run --name website -v
e:/CS54/CS_CICD_GitHub_Docker/Docker_Amigoscode/demo-volumes:/usr/share/nginx/html
-d -p 9000:80 nginx
10b8c11e4ed283584bd789ebc7d3ec6c4697d5d4dd16eaf5164f987c44509794

- Storage Driver



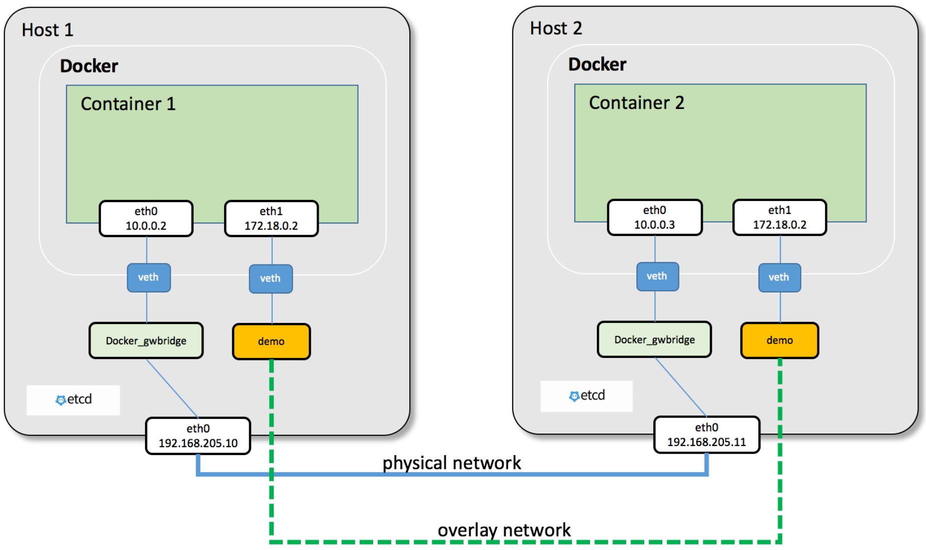
Docker Inspect

- docker inspect `container_id`



```
docker exec -it 10 bash
root@10b8c11e4ed2:/# ls
root@10b8c11e4ed2:~# cd /usr/share/nginx/html
root@10b8c11e4ed2:/usr/share/nginx/html# ls -al
total 4
drwxrwxrwx 1 root root  512 Nov  4 07:54 .
drwxr-xr-x 3 root root 4096 Oct 25 10:23 ..
-rwxrwxrwx 1 root root  313 Nov  3 17:22 index.html
```

renew the index.html and go to review the website



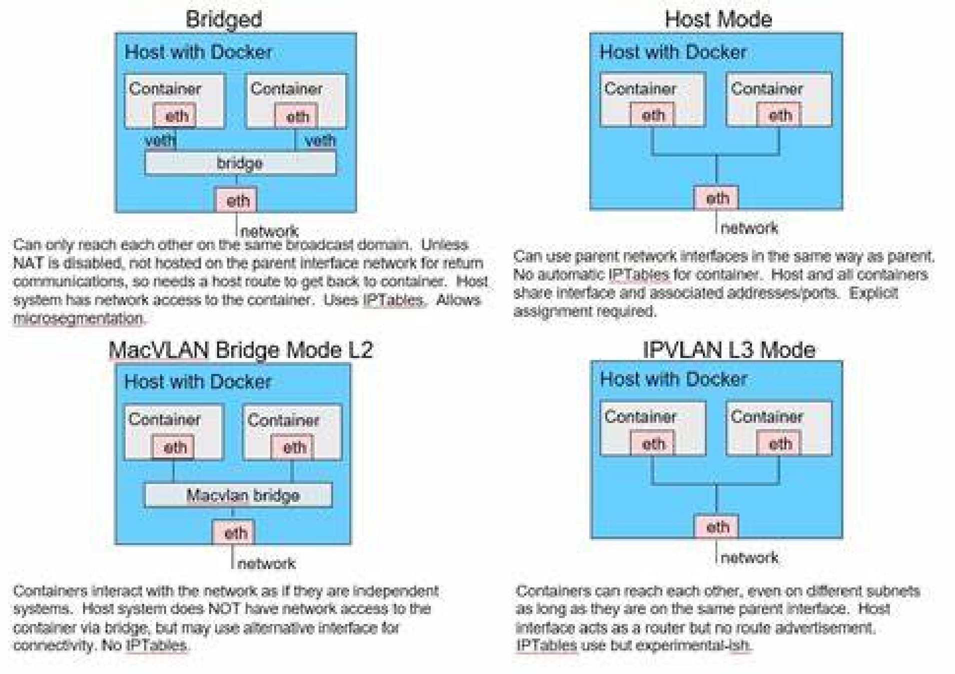
- Docker Network

- docker network 'inspect' NETWORK_ID

```
[
   {
       "Name": "bridge",
       "Id":
"da0aea13af084d57f98167d99a3b2005394cbe0a37a15cdf74504aef24f92a12",
       "Created": "2022-10-31T09:42:42.5528269Z",
       "Scope": "local",
       "Driver": "bridge",
       "EnableIPv6": false,
       "IPAM": {
           "Driver": "default",
           "Options": null,
           "Config": [
               {
                   "Subnet": "172.17.0.0/16",
                   "Gateway": "172.17.0.1"
               }
           ]
       },
       "Internal": false,
       "Attachable": false,
       "Ingress": false,
       "ConfigFrom": {
           "Network": ""
       },
       "ConfigOnly": false,
       "Containers": {

"4b3c92396c0d49116208654d2b90b915361bdef24259423b4628150fb6e9839a":
{
               "Name": "elated_robinson",
               "EndpointID":
"09450082d763b7e7129719da81e5439ec7b4c7285e3f0c559e8dd7b53b15b5a1",
               "MacAddress": "02:42:ac:11:00:02",
               "IPv4Address": "172.17.0.2/16",
               "IPv6Address": ""
           }
       },
       "Options": {
           "com.docker.network.bridge.default_bridge": "true",
           "com.docker.network.bridge.enable_icc": "true",
           "com.docker.network.bridge.enable_ip_masquerade":
"true",
           "com.docker.network.bridge.host_binding_ipv4":
"0.0.0.0",
           "com.docker.network.bridge.name": "docker0",
           "com.docker.network.driver.mtu": "1500"
       },
       "Labels": {}
   }
]
```

> - Docker Network Create
>
> -> docker network create `newwork name`
> -> docker network create `-d bridge` `newwork name`
> -> docker inspect `newtwork name`

> - Docker Network Connect
>
> -> docker network connect `newtwork name` `container name`
> -> docker run -it --network= `newtwork name` `image_name` bash
> -> docker inspect `container`

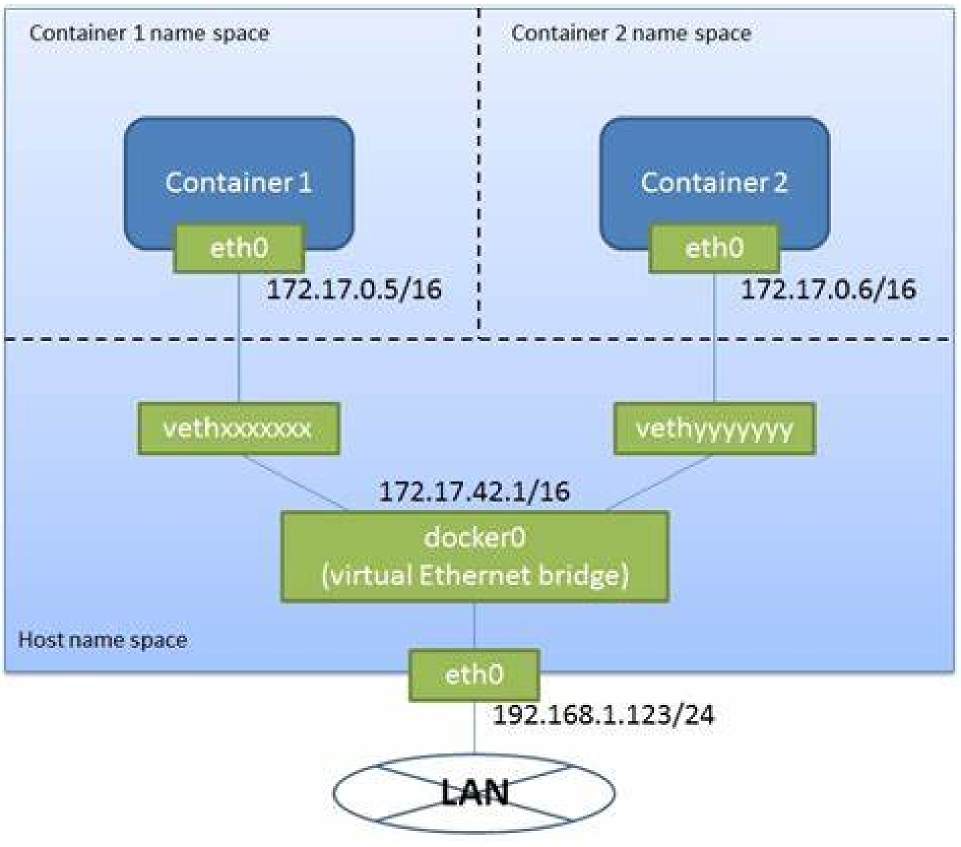


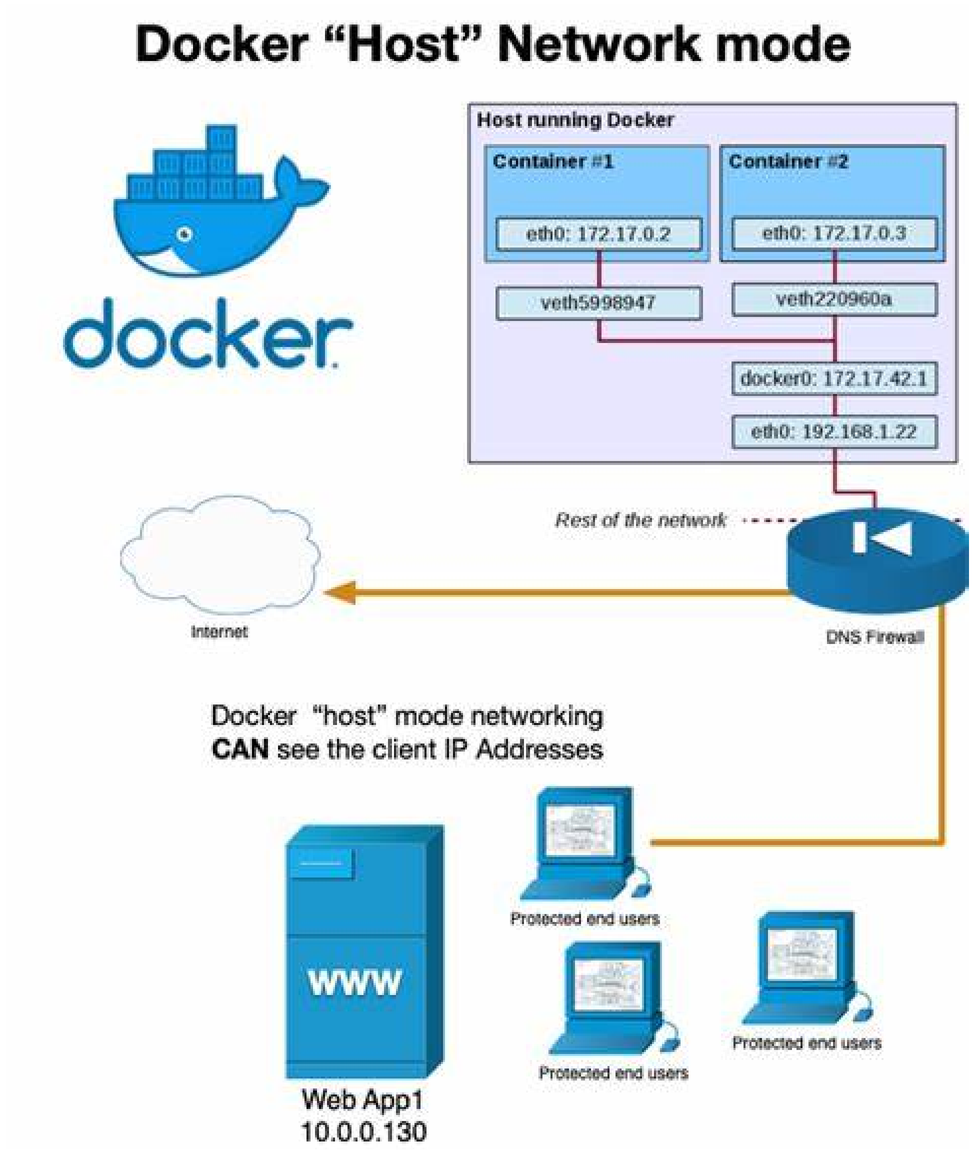
Can only reach each other on the same broadcast domain. Unless NAT is disabled, not hosted on the parent interface network for return communications, so needs a host route to get back to container. Host system has network access to the container. Uses IPTables. Allows microsegmentation.

Can use parent network interfaces in the same way as parent. No automatic IPTables for container. Host and all containers share interface and associated addresses/ports. Explicit assignment required.

Containers interact with the network as if they are independent systems. Host system does NOT have network access to the container via bridge, but may use alternative interface for connectivity. No IPTables.

Containers can reach each other, even on different subnets as long as they are on the same parent interface. Host interface acts as a router but no route advertisement. IPTables use but experimental-ish.


Container 1 name space
Container 2 name space
Container 1
eth0
172.17.0.5/16
Container 2
eth0
172.17.0.6/16
vethxxxxxxx
vethyyyyyyy
172.17.42.1/16
docker0
(virtual Ethernet bridge)
Host name space
eth0
192.168.1.123/24
LAN



Containerize Projects

Containerize Python - Web Scraping Project

> - Create VENV
>
> -> CMD: `python -m venv venv`
> -> CMD: venv\Scripts\activate.bat

> - Create main.py

> - Create Dockerfile

```
FROM python:3.10.8
ADD main.py .
RUN pip install requests beautifulsoup4 lxml
CMD ["python", "./main.py"]
```

> - Build image and Run image

> -> CMD: docker build -t python-imdb .
>
> -> CMD: docker run python-imdb

> ### Web Scraping Containerize with interaction
>
> - CMD: docker build -t python-imdb-active .
> - CMD: docker run -t -i python-imdb-active

Containerize Python - Flask

> - Create Registry
>
> Docker Repositoies

> - Create VENV
> - CMD: `python -m venv venv`
>
> -> CMD: venv\Scripts\activate.bat

> - Pre-requisiters
>
> -> `pip install flask`
>
> -> `pip freeze > requirements.txt`

> - Create Dockerfile
>
> ```
> FROM python:3.10.8
> WORKDIR /flask-app
> COPY requirements.txt .
> RUN python -m pip install --upgrade pip && \
>     pip install -r requirements.txt
> ADD . .
> CMD ["python", "app.py"]
> ```

> - Build Image
>
> -> docker build -t flaskninja:jobs .
>
> -> docker images
>
> -> docker tag flaskninja:jobs ericarthuang/websiteflask:jobs
>
> -> docker images
>
> -> docker push ericarthaung/websiteflask:jobs
>
> -> docker rmi ericarthuang/websiteflask:jobs
>
> -> docker pull ericarthuang/websiteflask:jobs

> - Create Container
>
> -> docker run --name ninjaflask -d -p 5000:80 ericarthuang/websiteflask:jobs
>
> -> docker ps -a

> go to `localhost:5000`

Containerize Python - Diango

> - Create Registry
>
> Docker Repositoies

> - Create VENV
>
> -> CMD: `python -m venv venv`
> -> CMD: venv\Scripts\activate.bat

> - Pre-requisiters
>
> -> `pip install django`
> -> `pip freeze > requirements.txt`

> - Create Dockerfile

```
FROM python:3.10.8
WORKDIR /django-app
COPY ./Django_Blog_Project ./
RUN pip install -r requirements.txt && \
    python -m pip install --upgrade pip
CMD ["python", "./Django_Blog_Project/manage.py", "runserver"]
```

> - Build Image
>
> -> docker build -t djangoblog:copylearning .
> -> docker images
>
> -> docker tag djangoblog:copylearning ericarthuang/djangoblog:copylearning
> -> docker images
> -> docker push ericarthuang/djangoblog:copylearning
>
> -> docker rmi ericarthuang/djangoblog:copylearning
> -> docker pull ericarthaung/djangoblog:copylearning

> - Create Container
>
> -> docker run --name djangoblog -d -p 8000:8000 ericarthuang/djangoblog:copylearning
> -> docker ps -a

> go to `localhost:8000`

Containerize Python - FastAPI

> -Create VENV
> -> CMD: `python -m venv venv`
> -> CMD: venv\Scripts\activate.bat

> - Pre-requisiters
>
> -> `pip install fastapi`
>
> -> `pip install uvicorn`
>
> -> `pip freeze > requirements.txt`

> - Create main.py and Execute main.py - 1
> - create `app` folder
> - create `__init__.py` in `app` folder
> - create `main.py` in `app` folder
> - CMD: uvicorn app.main:app --reload

> - Create main.py and Execute main.py -2
> - create `app` folder
> - create `__init__.py` in `app` folder
> - create `main.py` in `app` folder
>
> ```
> import uvicron
>
> if __name__=='__main__':
>      uvicorn.run(app, port=8000, host="0.0.0.0")
> ```
>
> - CMD: cd app
> - CMD: python main.py

> - Create `Dockerfile` in root directory
>
> ```
> FROM python:3.10.8
> WORKDIR /fastapi-app
> COPY requirements.txt .
> RUN pip install -r requirements.txt && \
>     python -m pip install --upgrade pip
> COPY ./app ./app
> CMD ["python", "./app/main.py"]
> ```

> - Create `Dockerfile` in root directory
>
> ```
> FROM python:3.10.8
> WORKDIR /fastapi-app
> COPY requirements.txt .
> RUN pip install -r requirements.txt && \
>     python -m pip install --upgrade pip
> COPY ./app ./app
> CMD ["python", "./app/main.py"]
> ```

> - Build Image
>
> -> docker build -t python-fastapi .

> - Create Container
>
> -> docker run `-p 8000:8000` python-fastapi

> - Review the container in terminal

> - CMD: docker ps
>
> -> `CONTAINER_ID`
>
> - CMD: docker exec -it `CONTAINER_ID` /bin/sh
>
> -> # ls
> -> # cd ..
> -> # ls
> -> # cd fastapi-app
> -> # app
> -> # ls
> -> # pwd
> -> # env
> -> # exit
> -> # ls `folder`
>
> - CMD: docker run `REPOSITORY:TAG`
>
> -> `pull image` + `create contain` + `start contain`
>
> - CMD: docker run -it `REPOSITORY:TAG` bin/sh
>
> -> #
>
> - CMD: docker run -it -d -p 9000:80 `REPOSITORY:TAG` bin/sh
> - CMD: docker exec `CONTAINER_ID` ...

Containerize User-Service-API

> - Pre-requisiters
> - Create `package.json`
>
> -> CMD: npm init

> - Create `index.js`

```
const express = require('express')
const app = express()
const port = 3000
app.get('/', (req, res) => res.json([
 {
   name: 'Bob',
    email: 'bob@gmail.com'
 },
 {
   name: 'Alice',
   email: 'Alice@gmail.com'
 },
 {
     name: 'Mario',
     email: 'Mario@gmail.com'
   },
]))
```

> ```
> app.listen(port, () => {
>   console.log(`Example app listening on port ${port}`)
> })
> ```

> - Go to localhost:3000
>
> -> CMD: node index.js

> - Create Dockerfile
>
> ```
> FROM node:alpine
> WORKDIR /app
> ADD package*.json ./
> RUN npm install
> ADD . .
> CMD ["node", "index.js"]
> ```

> - Create Dockerfile
>
> ```
> FROM node:alpine
> WORKDIR /app
> ADD package*.json ./
> RUN npm install
> ADD . .
> CMD ["node", "index.js"]
> ```

> - Build Image
>
> -> docker build -t user-service-api:latest .
>
> -> docker images
>
> - Push Repository
>
> -> docker tag website:copylearning ericarthuang/website:copylearning
>
> -> docker images
>
> -> docker push ericarthuang/website:copylearning
>
> - Pull Image from Repository
>
> -> docker pull ericarthuang/website:copylearning

> - Create Container
>
> -> docker run --name website_copylearning -d -p 8080:80 ericarthuang/website:copylearning
>
> -> docker ps -a

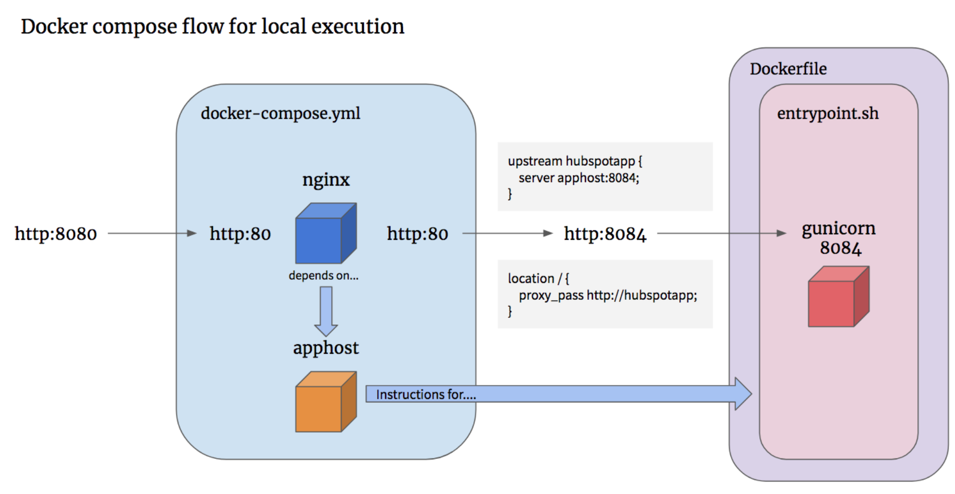
> - go to `localhost:5000`

> - Review the container in terminal
>
> -> docker exec -it eb bash
>
> root@eb7fbecb6365:/app# ls
>
> Dockerfile index.js node_modules package-lock.json package.json

> - .dockerignore

```
node-modules
Dockerfile
.git
```

Project - Using Docker Compose to Deploy a Django App



Create Registry

Docker Repositoies

Create VENV

- CMD: `python -m venv venv`

-> CMD: venv\Scripts\activate.bat

Pre-requiries

- `pip install django`
- `pip freeze > requirements.txt`

- Create `app` folder
- Create .gitignore
- Create .dockerignore

Create Docker File

```
FROM python:3.10-alpine
LABEL maintainer="londonappdeveloper.com"
ENV PYTHONUNBUFFERED 1
WORKDIR /app
EXPOSE 8000
COPY ./requirements.txt /requirements.txt
COPY ./app /app
RUN python -m venv /py && \
   /py/bin/pip install --upgrade pip && \
   /py/bin/pip install -r /requirements.txt && \
   adduser --disabled-password --no-create-home app
ENV PATH="/py/bin:$PATH"
USER app
```

Create docker-compose.yml

-> CMD: docker-compose version

```
version: "3.10"
services:
  app:
    build:
      context: .
    ports:
      - 8000:8000
    volumes:
      - ./app:/app
```

Build Image

-> docker-compose build

-> docker images

Use Image to Create Django Project

-> docker-compose run --rm app sh -c "django-admin startproject app ."

-> docker ps -a

- can find the `app/app`

Config settings.py

-> add `import os`

-> SECRET_KEY = os.environ.get('SECRET_KEY')

-> DEBUG = (os.environ.get('DEBUG') == 'True')

-> ALLOWED_HOSTS

-> INSTALLED_APPS

```
INSTALLED_APPS = [
    'app',
]
```

> ## Add ENV Variables into `docker-compose.yml`
>
> ```
> services:
>   app:
>     environments:
>        - SECRET_KEY=devsecretkey
>        - DEBUG=True
> ```

> ## Add ENV Variables into `docker-compose.yml`
>
> ```
> services:
>   app:
>     environments:
>        - SECRET_KEY=devsecretkey
>        - DEBUG=True
> ```

> ## Link `app` with `db` in `docker-compose.yml`
>
> ```
> services:
>   app:
>     environment:
>        - SECRET_KEY=devsecretkey
>        - DEBUG=True
>        - DB_HOST=db
>        - DB_NAME=devdb
>        - DB_USER=devuser
>        - DB_PASS=changeme
>     depends_on:
>        - db
> ```

> ## Add Postgres Drive into Django Application
>
> - Install some packages into `Dockerfile`
>
> ```
> RUN python -m venv /py && \
>     /py/bin/pip install --upgrade pip && \
>     # apk: alpine package manager
>     apk add --update --no-cache postgresql-client && \
>     apk add --update --no-cache --virtual .tmp-deps \
>     build-base postgresql-dev musl-dev && \
>     /py/bin/pip install -r /requirements.txt && \
>     apk del .tmp-deps && \
>     adduser --disabled-password --no-create-home app
> ```
>
> - modify `requirements.txt`

> -> `psysopq2>=2.9.5`

> ## Config `DATABASES` in settings.py

> ## Create New Application `core` and Container

> -> docker-compose build
> -> docker-compose run --rm app sh -c "python manage.py startapp core"

> Can find `app/core` folder
> -> `Config settings.py`
> -> INSTALLED_APPS

> -> docker ps -a
> container_names: docker_djangotoec2-db-1

> ## Create Testing Models

> - Create models
>
> -> `app/app/core/models/py`
> -> create Class Sample(models.Model)

> - Register Models in Admin Site
>
> -> `app/app/core/admin.py`
> -> from core.models import Sample
> -> admin.site.register(Sample)

> - Create Migrations
>
> -> docker-compose run --rm app sh -c \

"python manage.py makemigrations"

```
Migrations for 'core':
  core/migrations/0001_initial.py
    - Create model Sample
```

> - add `wait for db command` for connecting postgresql
>
> -> Create `management` folder in `app/app/core`
> -> Create `__init__.py` in `app/app/core/management`
> -> Create `commands` folder in `app/app/core/management`
> -> Create `__init__py` in `app/app/core/management/commands`
> -> Create `wait_for_db.py`

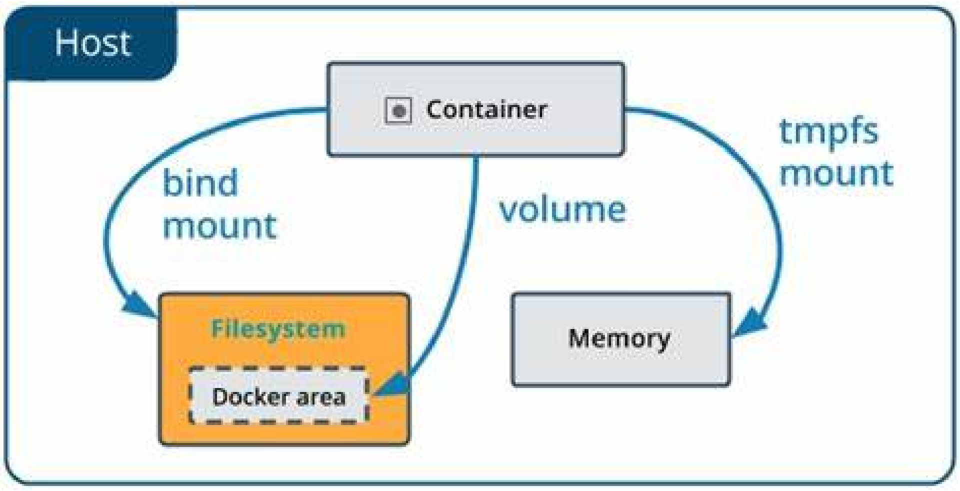
> - Update Docker Compose file to handle migrations

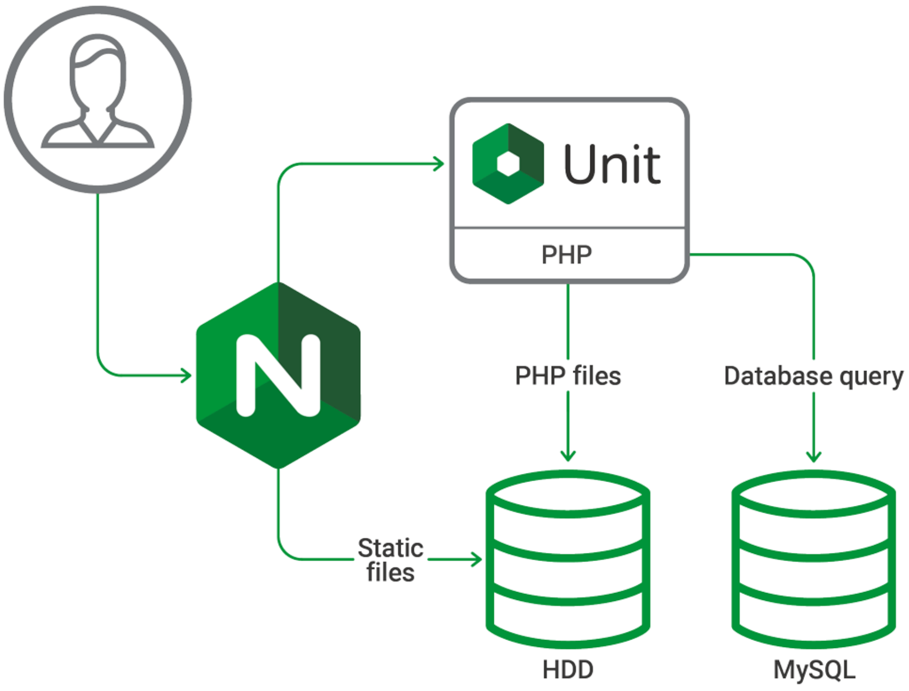
```
services:
  app:
command: >
      sh -c "python manage.py wait_for_db &&
            python manage.py makemigrations &&
            python manage.py migrate &&
            python manage.py runserver 0.0.0.0:8000"
```

- Start the app

-> docker-compose build
-> docker-compose up
-> docker-compose down

Handle static and media files





- go to `Dockerfile`

```
RUN:
    mkdir -p /vol/web/static && \
    mkdir -p /vol/web/mdeia && \
    chown -R app:app /vol && \
    chmod -R 755 /vor
```

- go to `docker-compose.yml`

```
services:
  app:
    volumes:
      - ./data/web:/vol/web
```

- Config `settings.py` for static and media files

```
STATIC_URL = 'static/static/'
MEDIA_URL = 'static/media/'

STATIC_ROOT = '/vol/web/static'
MEDIA_ROOT = '/vol/web/media'
```

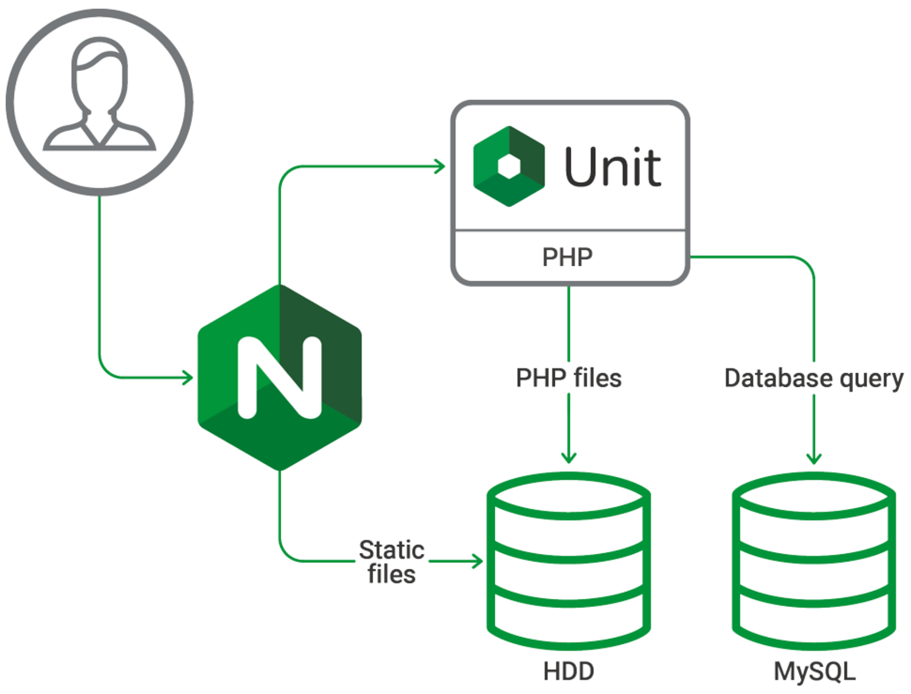
config `urls.py` in `app/app` for static and media files

-> from django.conf.urls.static import static

-> from django.conf import settings

```
if settings.DEBUG:
    urlpatterns += static(
        settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT,
    )
```

Handle static and media files



- go to `Dockerfile`

```
RUN:
    mkdir -p /vol/web/static && \
    mkdir -p /vol/web/mdeia && \
    chown -R app:app /vol && \
    chmod -R 755 /vor
```

> - go to `docker-compose.yml`
>
> ```
> services:
>   app:
>     volumes:
>       - ./data/web:/vol/web
> ```

> - Config `settings.py` for static and media files
>
> ```
> STATIC_URL = 'static/static/'
> MEDIA_URL = 'static/media/'
>
> STATIC_ROOT = '/vol/web/static'
> MEDIA_ROOT = '/vol/web/media'
> ```

> config `urls.py` in `app/app` for static and media files
>
> -> from django.conf.urls.static import static
>
> -> from django.conf import settings
>
> ```
> if settings.DEBUG:
>     urlpatterns += static(
>         settings.MEDIA_URL,
>         document_root=settings.MEDIA_ROOT,
>     )
> ```

Reverse Proxy to Handle Static and Media Files

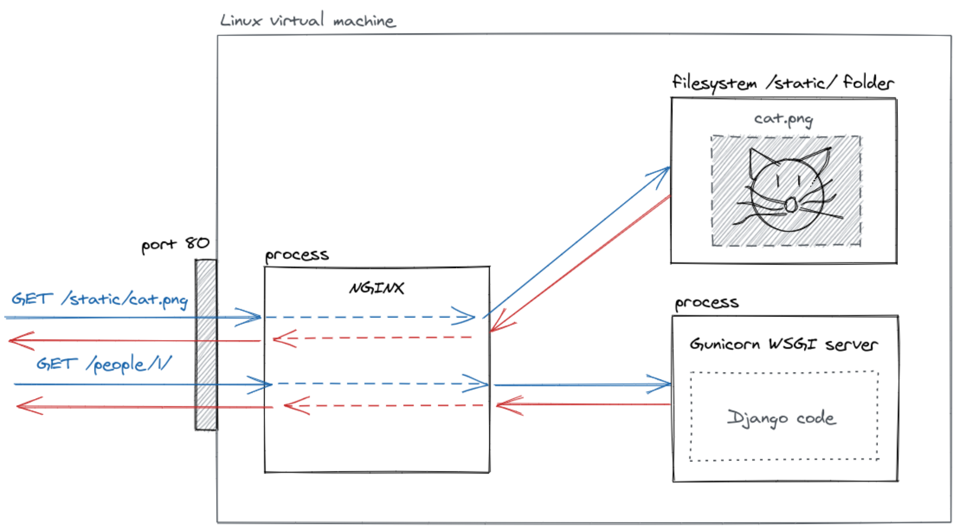
> - Create `proxy` folder in root directory

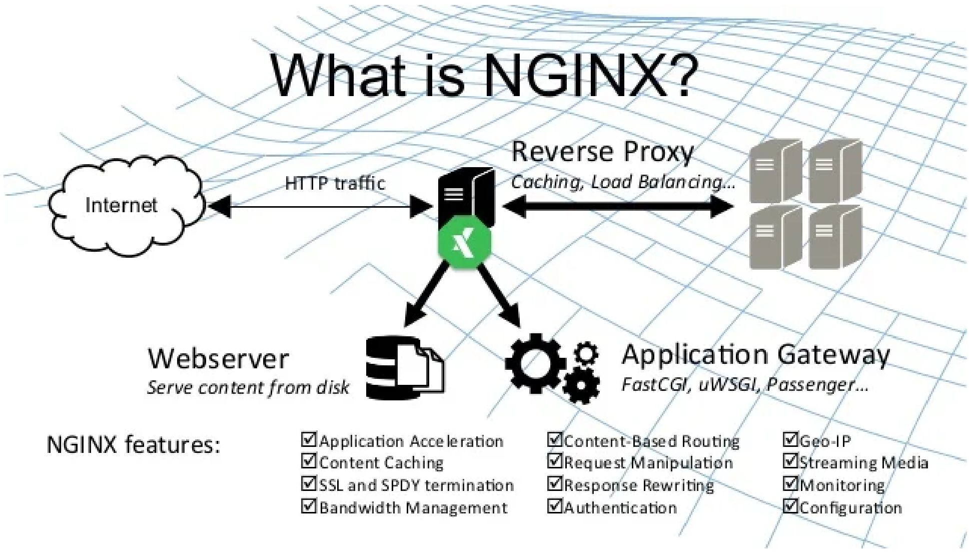
> - Create `uwsgi_params` in `proxy` folder

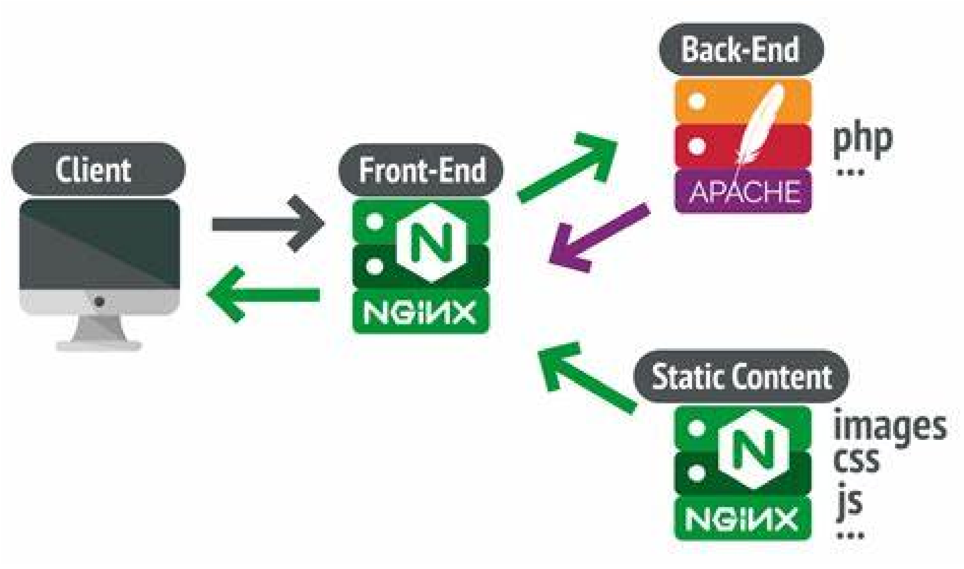
> - Create `default_conf_tpl` in `proxy` folder

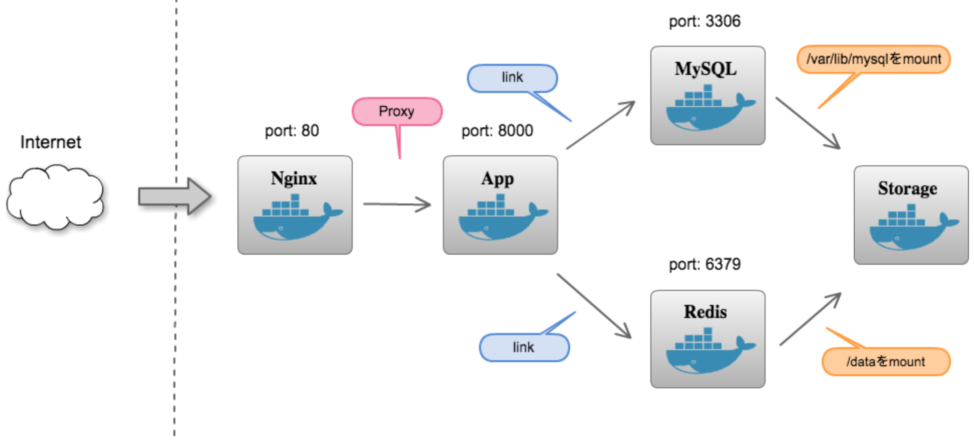
> - Create `run.sh` in `proxy` folder
>
> ```
> set -e
> envsubst < /etc/nginx/default.conf.tpl >
> /etc/nginx/conf.d/default.conf
> nginx -g 'daemon off;'
> ```

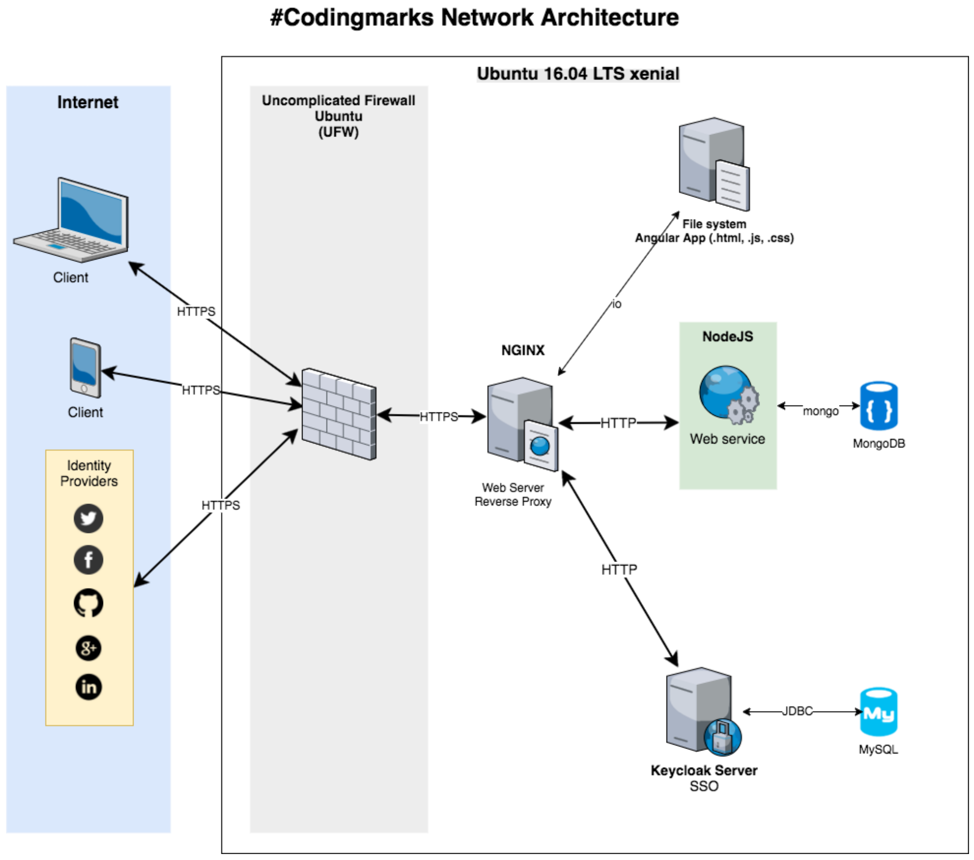
> - Create `Dockerfile` in `proxy` folder`


Linux virtual machine
filesystem /static/ folder
cat.png
port 80
process
NGINX
GET /static/cat.png
GET /people/1/
process
Gunicorn WSGI server
Django code

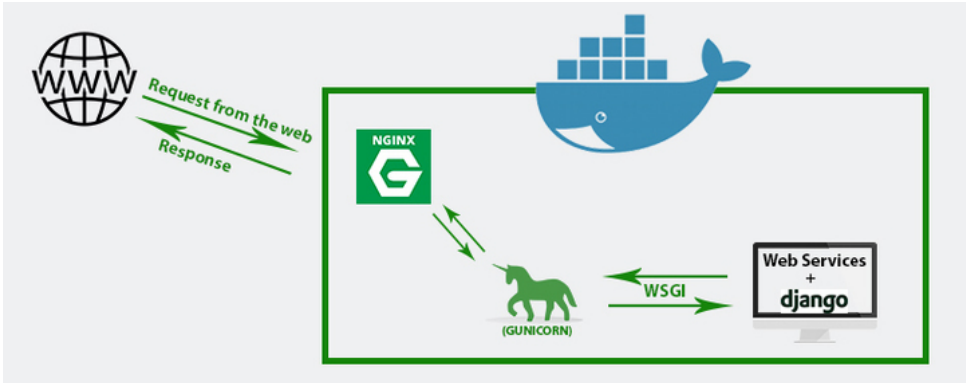

What is NGINX?
Internet
HTTP traffic
Reverse Proxy
Caching, Load Balancing...
Webserver
Serve content from disk
Application Gateway
FastCGI, uWSGI, Passenger...
NGINX features:
☑Application Acceleration
☑Content Caching
☑SSL and SPDY termination
☑Bandwidth Management
☑Content-Based Routing
☑Request Manipulation
☑Response Rewriting
☑Authentication
☑Geo-IP
☑Streaming Media
☑Monitoring
☑Configuration

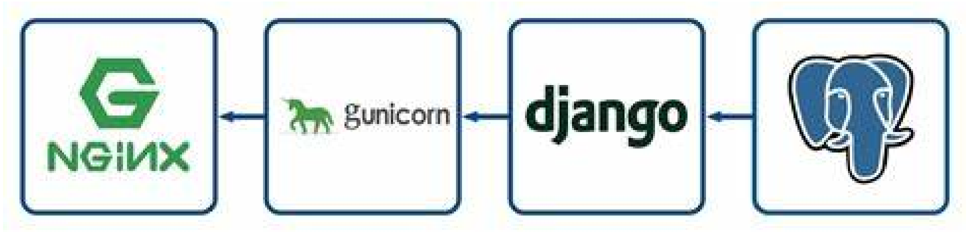

Client
Front-End
NGINX
Back-End
APACHE
php
...
Static Content
NGINX
images
css
js
...





Configure Django app to run as a uWSGI service





- Create `scripts` folder in root directroy

- Create `run.sh` in `scripts`
- add `uWSGI>=2.0.19.1,<2.1` into `requirements.txt`

- modify `Dockerfile`

```
COPY ./scripts /scripts
RUN apk add --update --no-cache --virtual .tmp-deps \
    build-base postgresql-dev musl-dev linux-headers && \
    chmod -R +x /scripts
ENV PATH="/scripts:/py/bin:$PATH"
CMD ["run.sh"]
```

- Create **`docker-compose-deploy.yml`** in root directory

- docker-compose -f docker-compose-deploy.yml down --volumes

- docker-compose -f docker-compose-deploy.yml build

- docker-compose -f docker-compose-deploy.yml up

Test uploading images in production mode

-> docker compose -f docker-compose-deploy.yml run --rm app sh -c "python manage.py createsuperuser

Future Plan

- **`Integrate with GitHub Actions and AWS`**

-- Memo End --