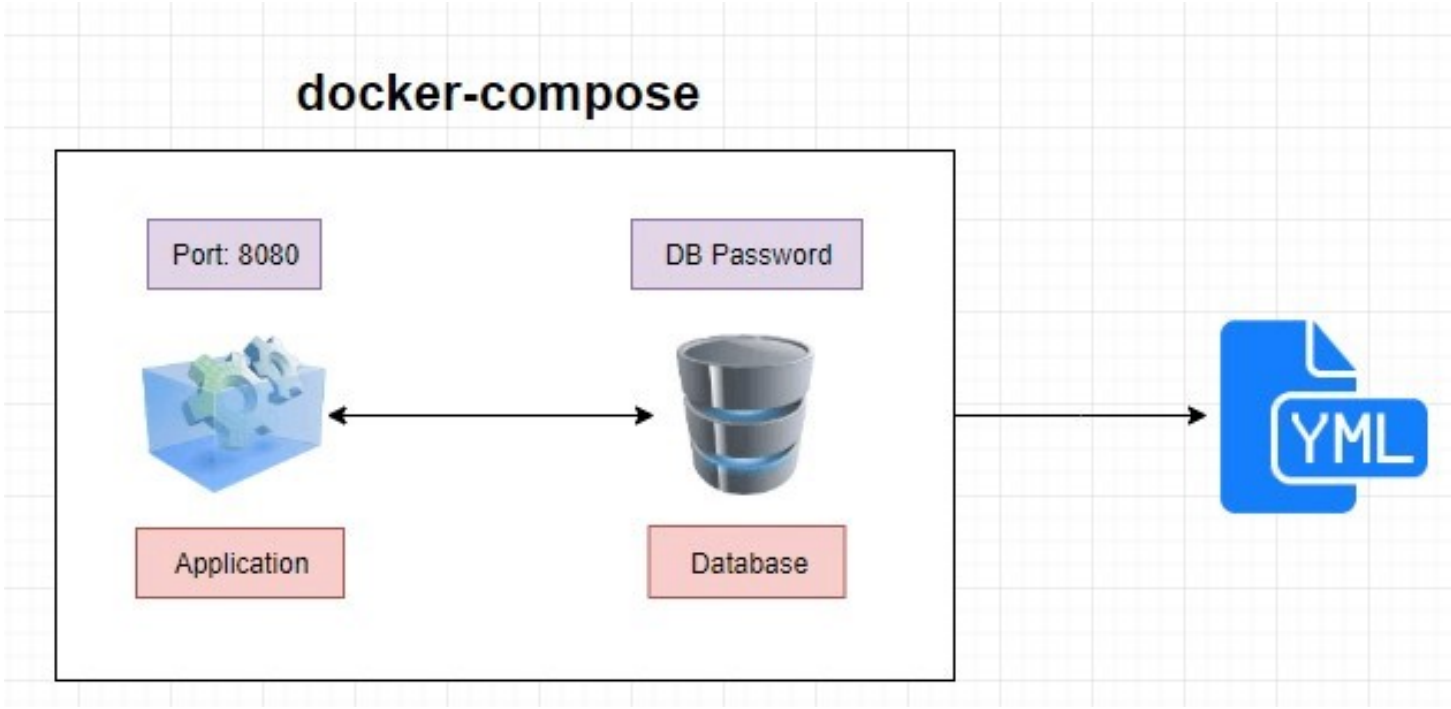
Docker – Day 3

Docker Compose

- A tool for defining and running multi-container docker applications.
- With Compose, we use a YAML file to configure our application's services.
- Then we create and start all the services from the configuration with a single command.



Docker Compose

- A three-step process:
  - Define application environment with a Dockerfile.
  - Define the services that make up the app in docker-compose.yml so they can be run together in an isolated environment.
  - Finally, run docker-compose up command and Compose will start and run your entire application.

Getting started with Docker Compose

- docker-compose –version

```
root@CPDockerTEST:/home/ubuntu/DevOps# ll
total 12
drwxr-xr-x  2 root   root   4096 Nov 11 08:01 ./
drwxr-xr-x 37 ubuntu ubuntu 4096 Nov 11 07:10 ../
-rw-r--r--  1 root   root     75 Nov 11 07:58 docker-compose.yaml
root@CPDockerTEST:/home/ubuntu/DevOps# cat docker-compose.yaml
version: '3'
services:
  web:
    image: nginx
  databse:
    image: redis
root@CPDockerTEST:/home/ubuntu/DevOps#
```

- docker-compose config
  - To check the validity of the file
- docker-compose up -d

Getting started with Docker Compose

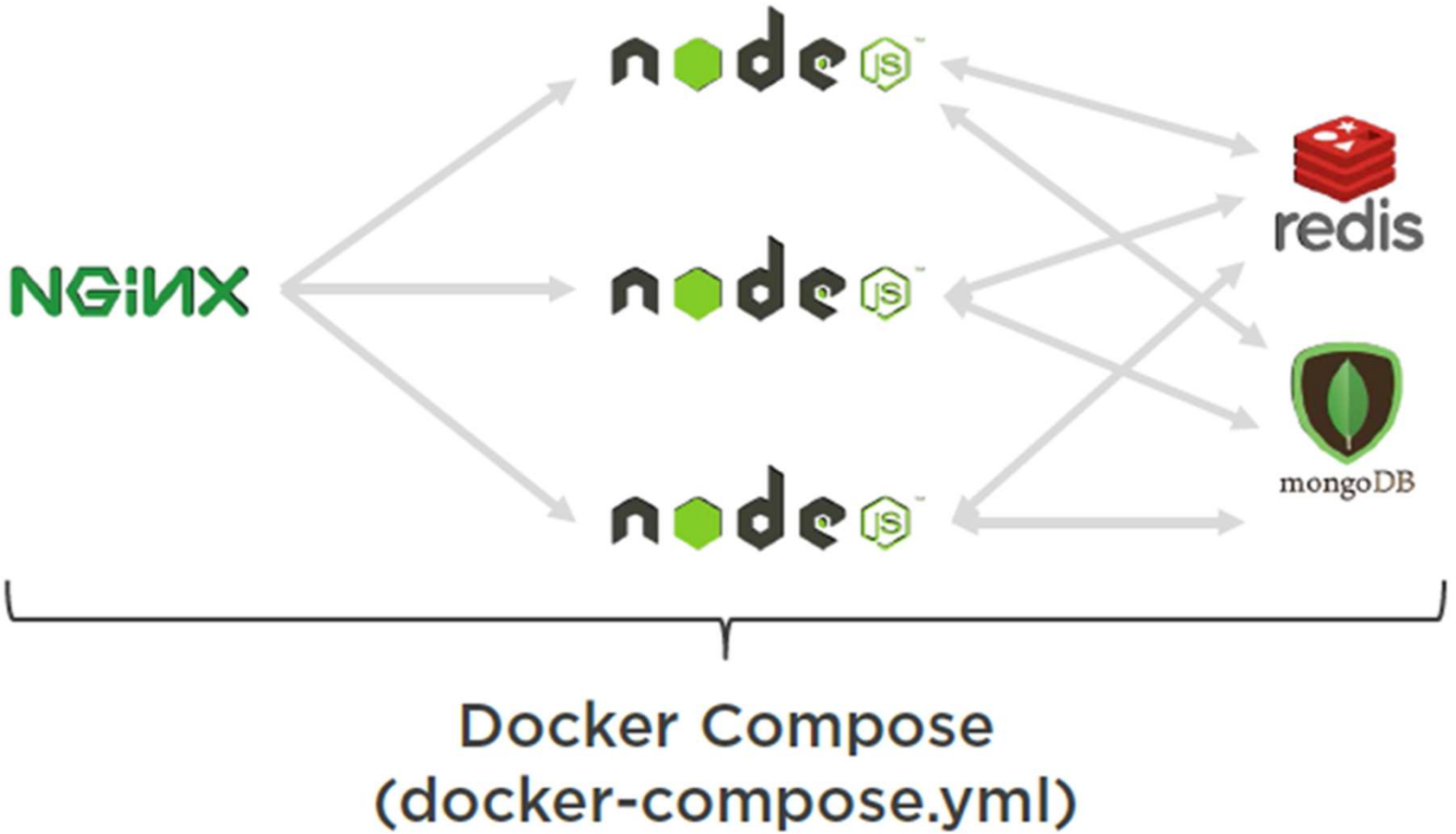
- docker-compose ps
- docker-compose down

```
root@CPDockerTEST:/home/ubuntu/DevOps# cat docker-compose.yaml
version: '3'
services:
  web:
    image: nginx
    ports:
    - 8181:80/tcp
  databse:
    image: redis
root@CPDockerTEST:/home/ubuntu/DevOps#
```

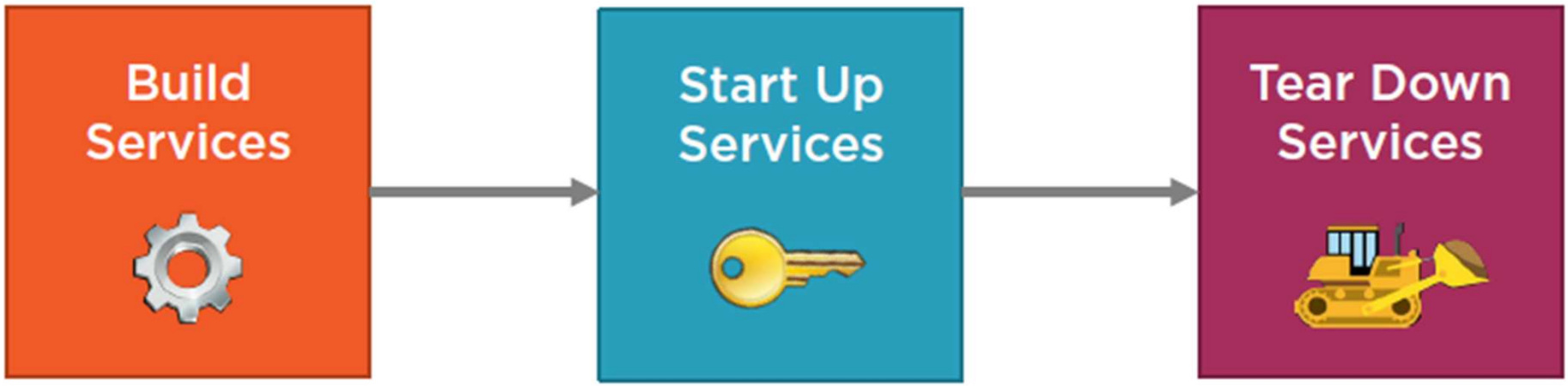
Docker Compose

- Manages the whole application lifecycle:
  - Start, stop and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service

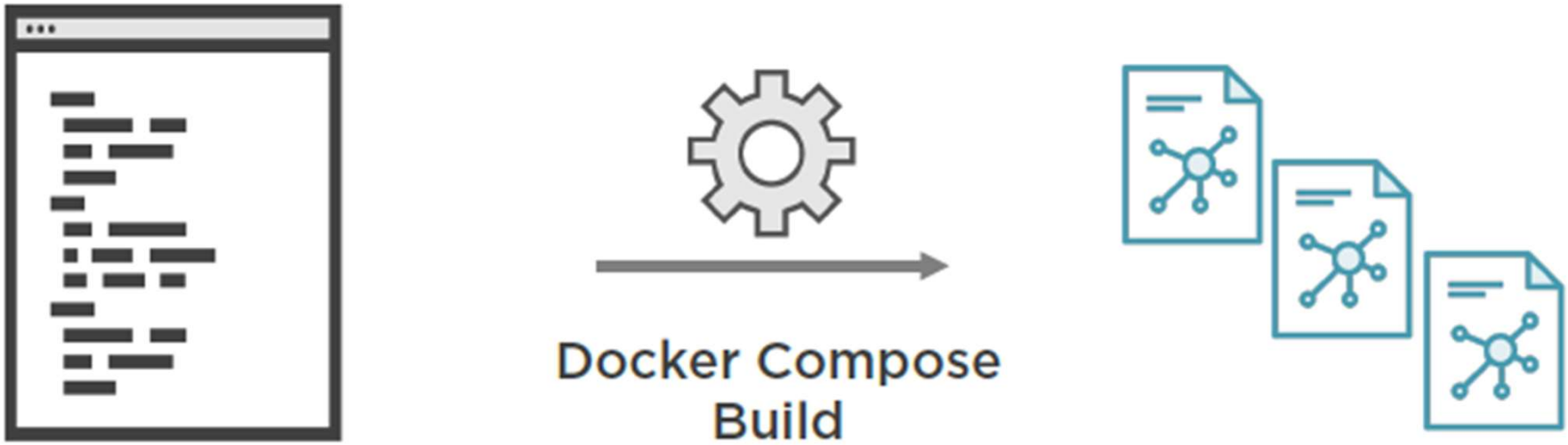
The need for Docker Compose



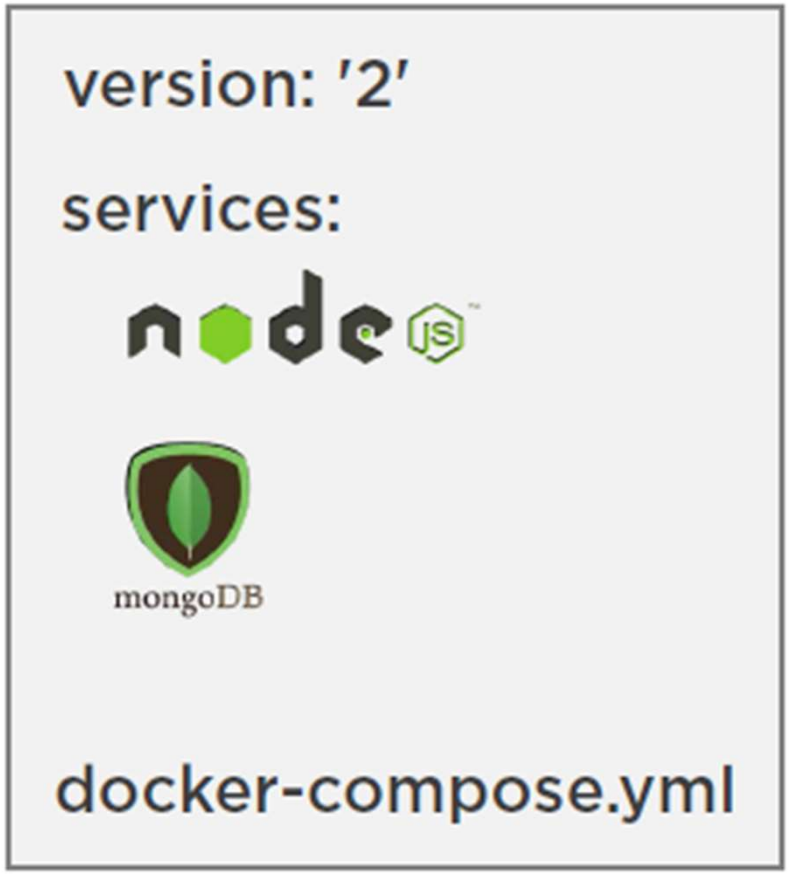
Docker Compose Workflow



The Role of the DockerCompose File



Docker Compose and Services



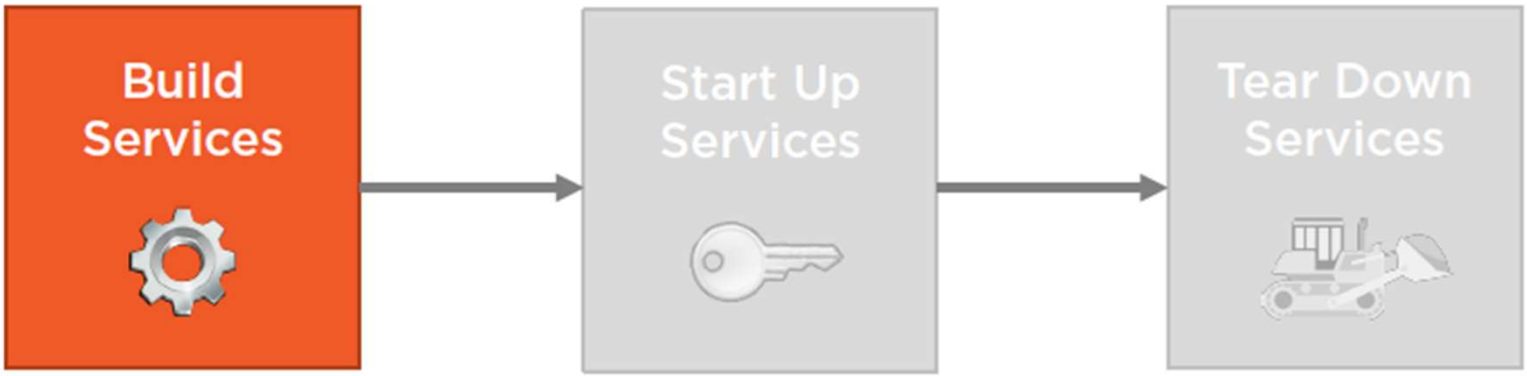
docker-compose.yml Example

- version: '2‘
- services:
  - node:
    - build:
      - context: .
      - dockerfile: node.dockerfile
    - networks:
      - -nodeapp-network
  - mongodb:
    - image: mongo
    - networks:
      - -nodeapp-network
- networks:
  - nodeapp-network
    - driver: bridge

Key Docker Compose Commands

- docker-compose build
- docker-compose up
- docker-compose down
- docker-compose logs
- docker-compose ps
- docker-compose stop
- docker-compose start
- docker-compose rm

Building Services



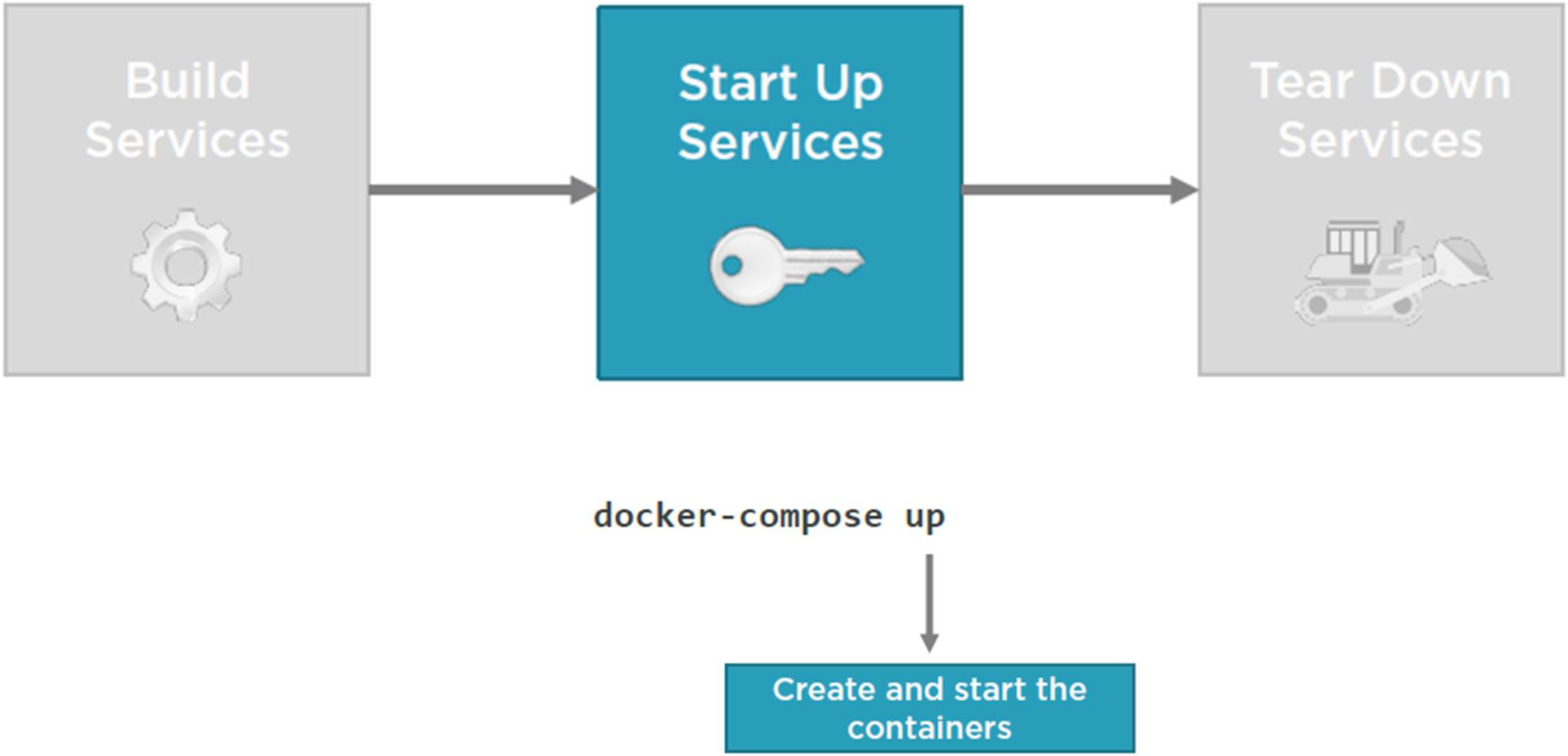

```
docker-compose build
```

Build or rebuild services defined in docker-compose.yml

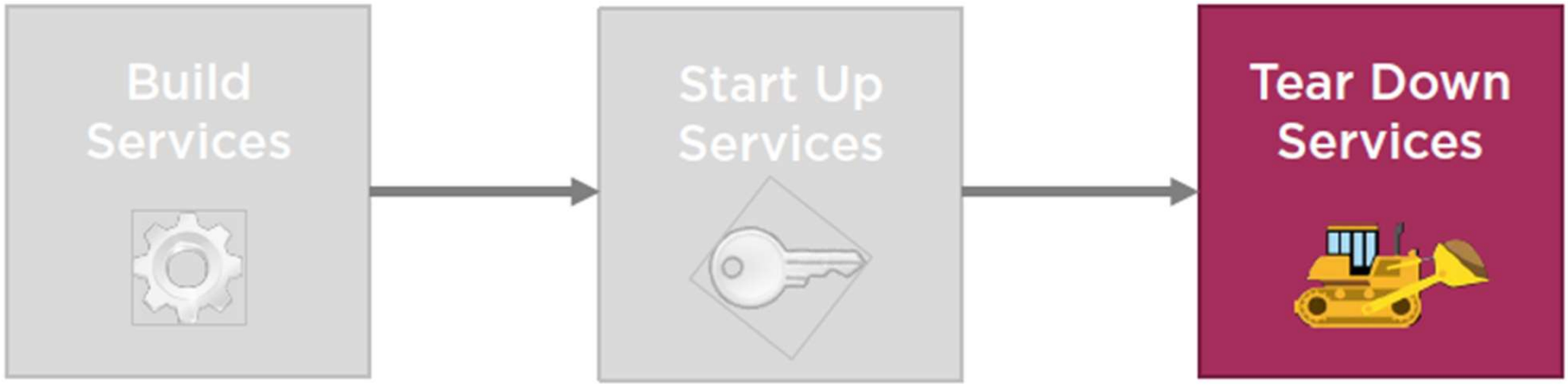
Building Specific Services

```
docker-compose build mongo
```

Starting Services Up



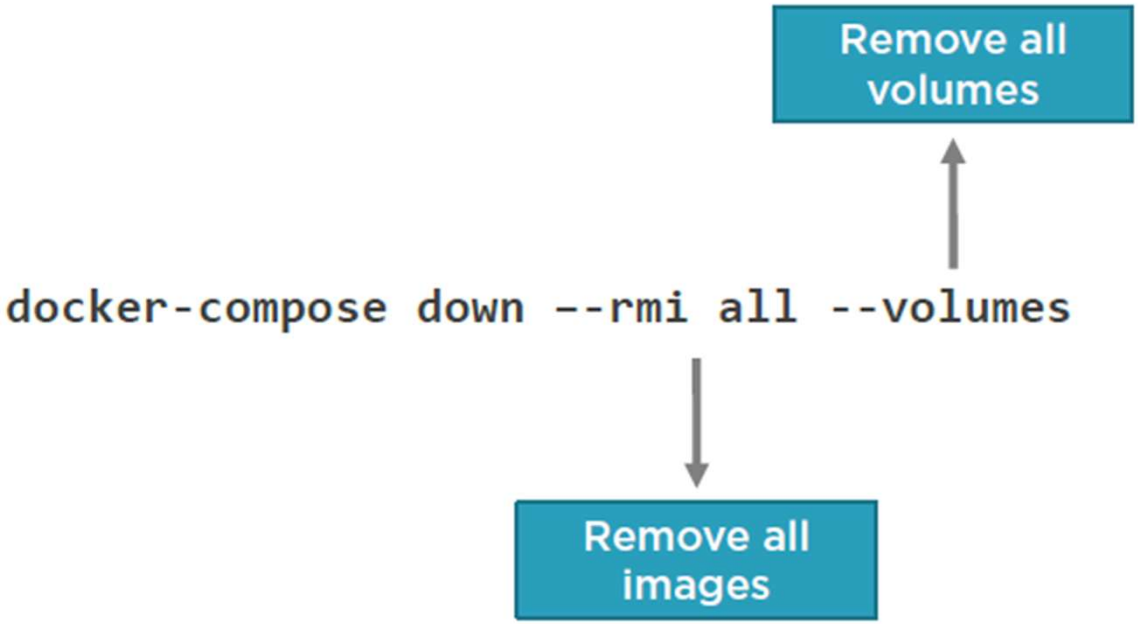
Tearing Down Services




```
docker-compose down
```

Take all of the containers down (stop and remove)

Stop and Remove Containers, Images, Volumes



Scale

- docker-compose up -d --scale database=3

```
root@CPDockerTEST:/home/ubuntu/DevOps# cat docker-compose.yaml
version: '3'
services:
  web:
    image: nginx
    ports:
    - 8181:80/tcp
  database:
    image: redis
root@CPDockerTEST:/home/ubuntu/DevOps# docker-compose up -d --scale database=3
WARNING: Found orphan containers (devops_databse_1) for this project. If you removed
 with the --remove-orphans flag to clean it up.
devops_web_1 is up-to-date
Stopping and removing devops_database_4 ... done
Starting devops_database_1              ... done
Starting devops_database_2              ... done
Starting devops_database_3              ... done
root@CPDockerTEST:/home/ubuntu/DevOps# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
devops_database_1   docker-entrypoint.sh redis ...   Up      6379/tcp
devops_database_2   docker-entrypoint.sh redis ...   Up      6379/tcp
devops_database_3   docker-entrypoint.sh redis ...   Up      6379/tcp
devops_web_1        nginx -g daemon off;             Up      0.0.0.0:8181->80/tcp
root@CPDockerTEST:/home/ubuntu/DevOps#
```

display running services

- docker-compose top

```
root@CPDockerTEST:/home/ubuntu/DevOps# docker-compose top
devops_database_1
UID    PID    PPID    C   STIME   TTY      TIME             CMD
------------------------------------------------------------------------
omi    37533   37494   0   Nov15   ?      00:01:20   redis-server *:6379

devops_web_1
  UID        PID    PPID    C   STIME   TTY      TIME                        CMD
------------------------------------------------------------------------------------------
root       37561   37504   0   Nov15   ?      00:00:00   nginx: master process nginx -g daemon off;
systemd+   37658   37561   0   Nov15   ?      00:00:00   nginx: worker process
root@CPDockerTEST:/home/ubuntu/DevOps#
```

logs

- docker-compose logs

```
root@CPDockerTEST:/home/ubuntu/DevOps# docker-compose logs
Attaching to devops_database_1, devops_web_1
database_1  | 1:C 15 Nov 2019 03:48:34.981 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
database_1  | 1:C 15 Nov 2019 03:48:34.981 # Redis version=5.0.6, bits=64, commit=00000000, modified=0,
database_1  | 1:C 15 Nov 2019 03:48:34.981 # Warning: no config file specified, using the default confi
r /path/to/redis.conf
database_1  | 1:M 15 Nov 2019 03:48:34.984 * Running mode=standalone, port=6379.
database_1  | 1:M 15 Nov 2019 03:48:34.984 # WARNING: The TCP backlog setting of 511 cannot be enforced
lower value of 128.
database_1  | 1:M 15 Nov 2019 03:48:34.984 # Server initialized
database_1  | 1:M 15 Nov 2019 03:48:34.984 * Ready to accept connections
database_1  | 1:signal-handler (1573789733) Received SIGTERM scheduling shutdown...
database_1  | 1:M 15 Nov 2019 03:48:53.930 # User requested shutdown...
database_1  | 1:M 15 Nov 2019 03:48:53.930 * Saving the final RDB snapshot before exiting.
database_1  | 1:M 15 Nov 2019 03:48:53.937 * DB saved on disk
database_1  | 1:M 15 Nov 2019 03:48:53.937 # Redis is now ready to exit, bye bye...
database_1  | 1:C 15 Nov 2019 03:50:49.730 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
database_1  | 1:C 15 Nov 2019 03:50:49.730 # Redis version=5.0.6, bits=64, commit=00000000, modified=0,
database_1  | 1:C 15 Nov 2019 03:50:49.730 # Warning: no config file specified, using the default confi
r /path/to/redis.conf
database_1  | 1:M 15 Nov 2019 03:50:49.736 * Running mode=standalone, port=6379.
database_1  | 1:M 15 Nov 2019 03:50:49.736 # WARNING: The TCP backlog setting of 511 cannot be enforced
lower value of 128.
database_1  | 1:M 15 Nov 2019 03:50:49.736 # Server initialized
```

Docker Compose Example

- mkdir ag_dockercompose && cd ag_dockercompose
- mkdir webapp
- echo "<h2>It Works</h2>" > webapp/index.html
- vim webapp/Dockerfile
  - FROM tecadmin/ubuntu-ssh:16.04
  - RUN apt-get update \
  - && apt-get install -y apache2
  - COPY index.html /var/www/html/
  - WORKDIR /var/www/html
  - CMD ["apachectl", "-D", "FOREGROUND"]
  - EXPOSE 80

Docker Compose Example

- vim docker-compose.yml
  - version: '3'
  - services:
  - db:
  - image: mysql
  - container_name: mysql_db
  - restart: always
  - environment:
  - - MYSQL_ROOT_PASSWORD="secret"
  - web:
  - image: apache
  - build: ./webapp
  - depends_on:
  - - db
  - container_name: apache_web
  - restart: always
  - ports:
  - - "8085:80"

Docker Compose Example

- docker-compose build
- docker-compose up -d
- You can access your web application running on the apache_web container by accessing your docker host on port 8085.

- echo "Welcome to Docker Compose Tutorial" >> webapp/index.html
- docker-compose build
- docker-compose up –d

Docker Security

- Prefer minimal base images
  - By preferring minimal images that bundle only the necessary system tools and libraries required to run your project, you are also minimizing the attack surface for attackers and ensuring that you ship a secure OS.
- Least privileged user
  - When a Dockerfile doesn’t specify a USER, it defaults to executing the container using the root user
  - To minimize exposure, opt-in to create a dedicated user
    - FROM ubuntu
    - RUN mkdir /app
    - RUN groupadd -r lirantal && useradd -r -s /bin/false -g lirantal lirantal
    - WORKDIR /app
    - COPY . /app
    - RUN chown -R lirantal:lirantal /app
    - USER lirantal
    - CMD node index.js

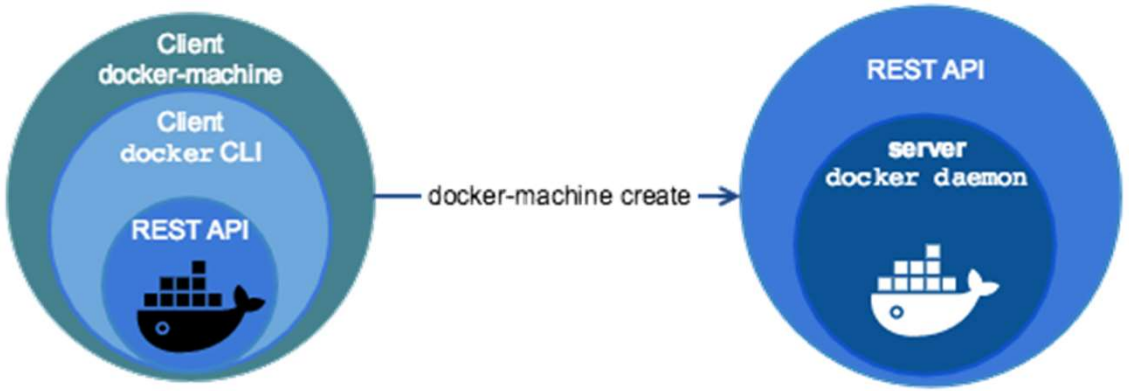
Docker Security

- Use only verified images
  - Authenticity of Docker images is a challenge.
  - We put a lot of trust into these images as we are literally using them as the container that runs our code in production.
  - Therefore, it is critical to make sure the image we pull is the one that is pushed by the publisher
- Don’t leak sensitive information to Docker images
- Beware of recursive copy
  - Use .dockerignore
- Don’t hard code credentials in images
- Store sensitive data only in volumes, never in a container
- Don’t use the default bridge network
- Mount volumes as read-only when you only need to read from them.

Docker Machine

- By "Docker" we mean Docker Engine which is a client-server application
- Its made up of
  - The Docker daemon,
  - REST API and
  - CLI client
- Docker Machine is a tool for provisioning and managing your Dockerized hosts (hosts with Docker Engine on them)

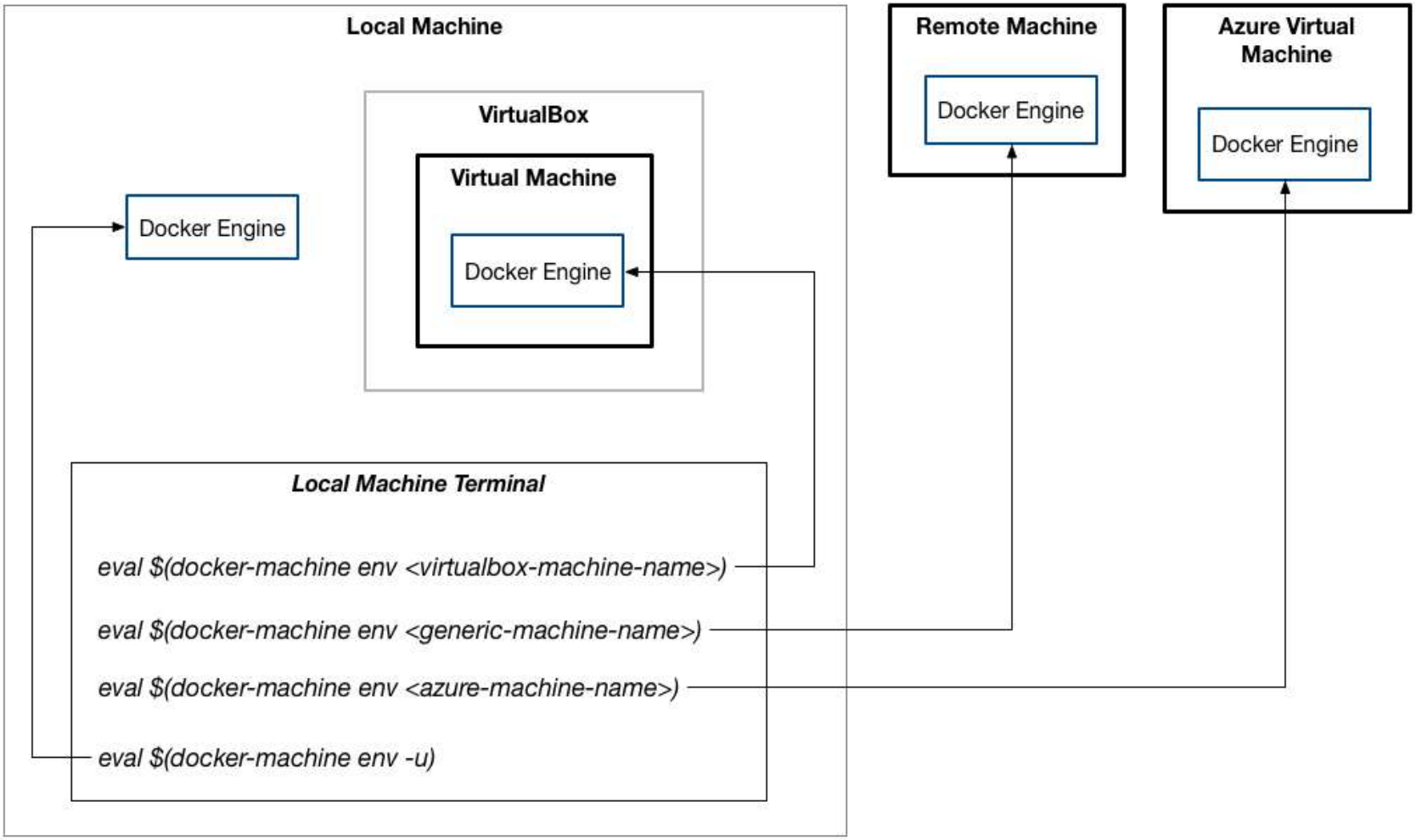
Docker Machine



Docker Machine

- It’s important to be able to easily deploy container app in the cloud
- Enables to create a remote virtual machine (VM) easily and manage those containers.
- Allows you to control the docker engine of a VM created using docker-machine remotely.
- Reason:
  - Create a deployment environment for application and
  - Manage all the micro-services/containers running on it
- Easily have a development, staging and production environment accessible from your own machine.

Docker Machine



Install and configure Docker Machine

- curl -L https://github.com/docker/machine/releases/download/v0.13.0/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine && \
- chmod +x /tmp/docker-machine && \
- sudo cp /tmp/docker-machine /usr/local/bin/docker-machine

- docker-machine version

Create a docker machine - Azure

- #!/usr/bin/env bash
- set -e
- MACHINE_NAME="VIRTUAL MACHINE NAME"
- RESOURCE_GROUP="RESOURCE GROUP NAME"
- SUBSCRIPTION="YOUR AZURE SUBSCRIPTION ID"
- AZURE_LOCATION="eastus"
- AZURE_VNET_NAME="VNET NAME"
- docker-machine create --driver azure
- --azure-availability-set="MACHINE_NAME-as"
- --azure-subscription-id="${SUBSCRIPTION}"
- --azure-location "${AZURE_LOCATION}"
- --azure-open-port 80
- --azure-open-port 443
- --azure-size "${AZURE_MACHINE_SIZE}"
- --azure-subnet "${AZURE_VNET_NAME}-subnet"
- --azure-vnet "${AZURE_VNET_NAME}"
- --azure-resource-group "${RESOURCE_GROUP}"
- ${MACHINE_NAME}

Delete docker machine - Azure

- docker-machine rm <machine-name>
  - This command will DELETE the Azure virtual machine and all related resources from your subscription! Use it with care.

Create a docker machine - Virtualbox

- MACHINE_NAME="agm1"
- sudo docker-machine create --driver virtualbox ${MACHINE_NAME}

- docker-machine ls
- docker-machine stop agm1
- docker-machine start agm1
- docker-machine restart agm1

Create a docker machine - Generic

- MACHINE_IP="MACHINE IP"
- MACHINE_NAME="MACHINE NAME"
- SSH_USER="MACHINE USERNAME"
- SSH_PUBLIC_KEY="MACHINE USERNAME PUBLIC KEY PATH"
- # If you did an ssh-copy-id to the machine: ~/.ssh/id_rsa
- docker-machine create --driver generic --generic-ip-address=${MACHINE_IP} --generic-ssh-key ${SSH_PUBLIC_KEY} --generic-ssh-user ${SSH_USER} ${MACHINE_NAME}

Docker machines – list, stop, start, restart

- docker-machine ls
- docker-machine restart <machine-name>
- docker-machine stop <machine-name>
- docker-machine start <machine-name>

Deploy containers to a remote host

- eval $(docker-machine env demo-machine) # demo-machine is machine-name
- To validate which docker-machine you point to, use this command:
  - $ docker-machine active
- docker-compose up –d # Deployed on the remote machine
- curl $(docker-machine ip demo-machine):80
- # ssh into remote machine
  - docker-machine ssh demo-machine
- # Use SCP command to send/receive files to/from the machine.
  - docker-machine scp ~/localfile.txt demo-machine:~/
  - docker-machine scp demo-machine:~/removefile.txt ~/
- go back to your local instance
  - eval $(docker-machine env -u)
  - docker-machine active

Setting up a private registry

- docker run -d -p 5000:5000 --restart=always --name registry registry:2
- docker ps
- # Push a custom Docker image to a remote private registry
- docker tag hello-world <ipaddress>:5000/hello-world # Replace with your IP/domain
- docker images
- docker push <ipaddress>:5000/hello-world
- # if above command does not work then
- vim /etc/docker/daemon.json
- {
- "insecure-registries" : ["my_registry_address:5000"]
- }

Swarm

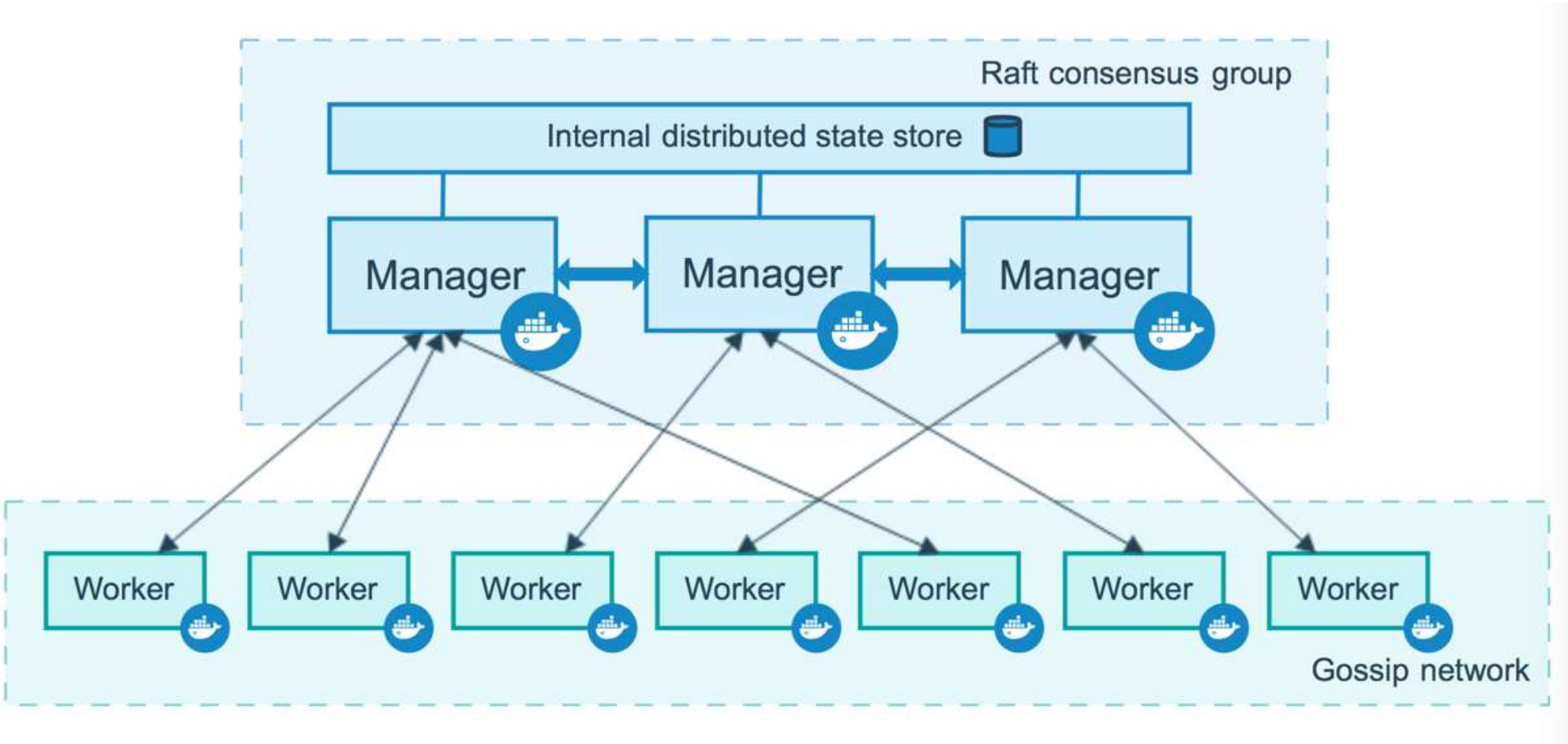
Docker Swarm

- Clustering and scheduling tool.
- IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.

Use cases

- Imagine that you had to run hundreds of containers.
- Now there are multiple features that you will need from a management angle to make sure that the cluster is up and running.
  - Health Checks on the Containers
  - Launching a fixed set of Containers for a particular Docker image
  - Scaling the number of Containers up and down depending on the load
  - Performing rolling update of software across containers

Swarn Managers



Manager nodes

- Manager nodes handle cluster management tasks:
  - maintaining cluster state
  - scheduling services
  - serving swarm mode HTTP API endpoints

- A three-manager swarm tolerates a maximum loss of one manager.
- A five-manager swarm tolerates a maximum simultaneous loss of two manager nodes.
- An N manager cluster tolerates the loss of at most (N-1)/2 managers.

Worker nodes

- Sole purpose is to execute containers
- You can create a swarm of one manager node, but you cannot have a worker node without at least one manager node.
- By default, all managers are also workers.
- You can promote a worker node to be a manager by running docker node promote

Setup Docker Swarm

- Create Docker Machines
  - Create a set of Docker machines that will act as nodes in our Docker Swarm
    - docker-machine create --driver virtualbox manager1
    - docker-machine create --driver virtualbox worker1
    - docker-machine create --driver virtualbox worker2
    - docker-machine create --driver virtualbox worker3
    - docker-machine create --driver virtualbox worker4
    - docker-machine create --driver virtualbox worker5
  - docker-machine ls
  - docker-machine ip manager1 # Note the IP address for later usage
  - docker-machine ssh manager1
    - docker swarm init --advertise-addr MANAGER_IP
    - docker node ls
    - docker swarm join-token worker
    - docker swarm join-token manager

Adding Worker Nodes to our Swarm

- docker-machine ssh worker1
  - docker swarm join \
  - — token SWMTKN-1–5mgyf6ehuc5pfbmar00njd3oxv8nmjhteejaald3yzbef7osl1-ad7b1k8k3bl3aa3k3q13zivqd \
  - 192.168.1.8:2377
- Do the same on other machines
- Run the below command on manager1 node:
  - docker node ls
    - ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS
    - 1ndqsslh7fpquc7fi35leig54 worker4 Ready Active
    - 1qh4aat24nts5izo3cgsboy77 worker5 Ready Active
    - 25nwmw5eg7a5ms4ch93aw0k03 worker3 Ready Active
    - 5oof62fetd4gry7o09jd9e0kf * manager1 Ready Active Leader
    - 5pm9f2pzr8ndijqkkblkgqbsf worker2 Ready Active
    - 9yq4lcmfg0382p39euk8lj9p4 worker1 Ready Active

Swarm details

- Ssh to manager1
- docker info
  - Swarm: active
  - NodeID: 5oof62fetd4gry7o09jd9e0kf
  - Is Manager: true
  - ClusterID: 6z3sqr1aqank2uimyzijzapz3
  - Managers: 1
  - Nodes: 6
  - Orchestration:
  - Task History Retention Limit: 5
  - Raft:
  - Snapshot Interval: 10000
  - Heartbeat Tick: 1
  - Election Tick: 3
  - Dispatcher:
  - Heartbeat Period: 5 seconds
  - CA Configuration:
  - Expiry Duration: 3 months
  - Node Address: 192.168.1.8

Create a Service

- docker service create --replicas 5 -p 80:80 --name web nginx
- docker service ls
- docker service ps web
- docker service ls
- docker service ls
- docker service ps web
- docker ps # nginx daemon has been launched

Accessing the Service

- Hit – http://<manager1-ip>
- Hit – http://<worker1-ip>
- Hit – http://<worker2-ip>
- Hit – http://<worker3-ip>
- Hit – http://<worker4-ip>

Scaling up and Scaling down

- docker service scale web=8
- docker service ls
- docker service ps web
- docker service ls

Inspecting nodes

- docker node inspect self
- docker node inspect worker1

Remove the Service

- docker service rm web

Applying Rolling Updates

- In case you have an updated Docker image to roll out to the nodes, all you need to do is fire an service update command.
  - docker service update --image <imagename>:<version> web

Docker Basic Troubleshooting

Configure and troubleshoot the Docker daemon

- dockerd

- dockerd --debug \
- --tls=true \
- --tlscert=/var/docker/server.pem \
- --tlskey=/var/docker/serverkey.pem \
- --host tcp://192.168.59.3:2376

- dockerd --help

Docker daemon directory

- /var/lib/docker on Linux.
- C:\ProgramData\docker on Windows.

Docker status

- docker info

- sudo systemctl is-active docker or
- sudo status docker or
- sudo service docker status

Read the Logs

Platform	File Location
Ubuntu	/var/log/upstart/docker.log
Debian[1]	/var/log/daemon.log
Windows 10	%APPDATA%\Local\Docker\log.txt
Windows Server 2016	Windows Application Event Log

Troubleshooting containers

- Use the docker logs command to see logs for a container.
- docker logs <containerid> -–tail 50

- docker cp <container_id>:/path/to/useful/file /local-path

Docker IPTABLES

- By default, docker daemon appends iptables rules for forwarding. For this, it uses a filter chain named DOCKER
  - sudo iptables --list
- When you tell docker to expose a port of a container, it exposes it to the entire world.
- docker run --name some-nginx -d -p 9090:80 nginx
  - behind the scene is adding an iptables rule to the DOCKER filter chain.
    - Chain FORWARD (policy DROP)
    - target prot opt source destination
    - DOCKER all -- 0.0.0.0/0 0.0.0.0/0
    - ...
    - Chain DOCKER (1 references)
    - target prot opt source destination
    - ACCEPT tcp -- 0.0.0.0/0 172.17.0.2 tcp dpt:9090 <-- this was added when running the container

Docker IPTABLES

- Now port 9090 is available from the entire world. Why?
  - Because we're listening 9090 on any IP addresses (*) and because of the forwarding rules that are dynamically added in the DOCKER filter chain
- You probably don't want that.
- docker run --name some-nginx -d -p 127.0.0.1:9090:80 nginx
  - # BEFORE
  - netstat -an | grep 9090
  - tcp6 0 0 :::9090 :::* LISTEN
  - # AFTER
  - netstat -an | grep 9090
  - tcp 0 0 127.0.0.1:9090 0.0.0.0:* LISTEN

Misc Queries

- How to clear the cache?
- docker build --no-cache -t u12_core .
-
- How to check where the volumes are mounted?
- docker ps -a --filter volume=u80_vol

Thanks