DOCKER

DevOps course by Ali Kahoot - Dice Analytics

EXAMPLE: MYSQL

Go to https://hub.docker.com/_/mysql and walkthrough the examples.

RESTART POLICY

Create a folder `Restarts` and create a shell script `crash.sh` with following content

```
#/bin/bash
sleep 30
exit 1
```

After 30 seconds the container will send a non-zero exit status i.e. a failed response.

Create a Dockerfile

```
FROM ubuntu
ADD crash.sh /
CMD /bin/bash /crash.sh
```

RESTART POLICY

Build the image

```
docker build -t restart .
docker run -d --name test-restart restart
```

Check the status of container

```
docker ps
```

And then check after 30 seconds

```
docker ps -a
docker container rm test-restart
```

RESTART POLICIES

There are 4 restart policy to handle this situation

- No: Never restart the container (Default behavior)
- On-failure: Restart only in case of Failure (Non-zero Exit code)
- Always Always Restart even in success or Failure, even when node restarts
- Unless-stopped: Same as always, but will not restart containers that were manually stopped

```
docker rm test-restart
docker run -d --name test-restart --restart always restart
```

RESTART POLICIES

We will try with On-Failure now.

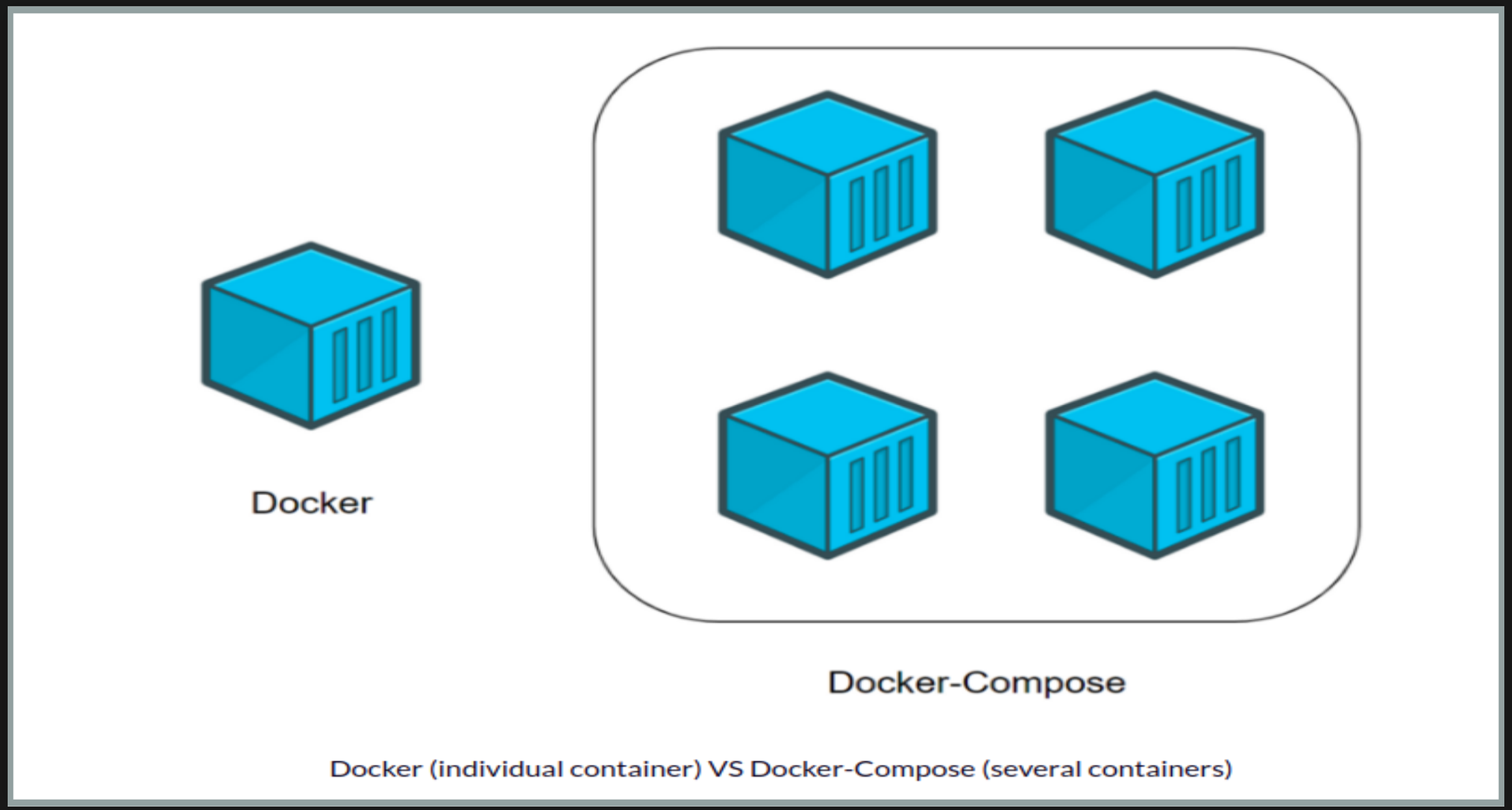
Change the script to

```
#/bin/bash
sleep 30
exit 0
```

Run following commands

```
docker container rm test-restart
docker build -t restart .
docker run -d --name test-restart --restart on-failure:5 restart
```

DOCKER COMPOSE



DOCKER COMPOSE

- Tool for defining and running multi-container - applications..
- Compose contains information how to build and deploy - containers.
- You write a docker-compose.yml file to configure your - applications services.
- Single command to create and start all the services from - your configuration
- It is a YAML file

BENEFITS OF COMPOSE

- Multiple isolated environments on a single host
- Preserve volume data when containers are created
- Only recreate containers that have changed
- Variables and moving a composition between environments
- Infrastructure as Code
- Portability

DOCKER COMPOSE - SYNTAX

- Version: Specifies the Compose file syntax version
- Services: In Docker a service is the name for a - "Container in production".
- Networks: This section is used to configure networking - for your application.
- Volumes: Mounts a linked path on the host machine that - can be used by the container.

DOCKER COMPOSE - SERVICES

- Image: Sets the image that will be used to build the - container.
- Build: Can be used instead of image. Specifies the - location of the Dockerfile that will be used.
- Restart: Tells the container to restart if the system - restarts.
- Volumes: Mounts a linked path on the host machine that - can be used
- Environment: Define environment variables to be passed - in to the Docker run
- Depends_on: Sets a service as a dependency for the - current block-defined container
- Port: Maps a port from the container to the host in the - following manner
- Links: Link this service to any other services in the - Docker Compose file by specifying their names here

DOCKER COMPOSE - SERVICES

```
version: '3'
services:
  cache:
    image: redis:alpine
  db:
    image: mysql:5.7
```

DOCKER COMPOSE - NETWORKS

```yaml
version: '3'
services:
  cache:
    image: redis:alpine
    networks:
     - appnet
  db:
    image: mysql:5.7
    networks:
     - appnet
networks:
  appnet:
    driver: bridge
```

DOCKER COMPOSE - VOLUMES

```yaml
version: '3'
services:
 mysql:
  image: mysql
  container_name: mysql
  volumes:
    - mysql:/var/lib/mysql
volumes:
 mysql:
```

DOCKER COMPOSE - COMMANDS

- docker-compose up: Create and start containers
- docker-compose ps: List containers
- docker compose build: Build or rebuild services
- docker-compose start: Start services
- docker-compose stop: Stop services
- docker-compose restart: Restart services
- docker-compose down: Stop and remove containers, - networks, images, and volumes
- docker-compose pull: Pull service images
- docker-compose scale: Set number of containers for a - service

LAB - DOCKER COMPOSE COMMANDS

- Setup environment
- Create docker-compose file
- Create a compose service
- List containers by compose
- Stopping compose service
- Starting a compose service
- Restarting a compose service
- Deleting compose service

SETUP ENVIRONMENT

```
$ mkdir -p compose/commands
$ cd compose/commands
```

CREATE DOCKER-COMPOSE FILE

Create a file named "docker-compose.yml" with your favorite editor and paste following content in it.

```
version: '3'
services:
  web:
    image: nginx
    ports:
    - "8080:80"
    volumes:
    - nginx_html:/usr/share/nginx/html/
    links:
    - redis
  redis:
    image: redis
volumes:
  nginx_html: {}
```

CREATE A COMPOSE SERVICE

```
$ docker-compose up -d
```

LIST CONTAINERS CREATED BY COMPOSE

```
$ docker-compose ps
```

STOPPING A COMPOSE SERVICE

```
$ docker-compose stop
```

STARTING A COMPOSE SERVICE

```
$ docker-compose start
```

RESTARTING A COMPOSE SERVICE

```
$ docker-compose restart
```

DELETE A COMPOSE SERVICE

```
$ docker-compose down
```

LAB - BUILD IMAGES WITH DOCKER COMPOSE

- Create a simple html file
- Create a Dockerfile
- Create docker-compose.yml
- Building images and launch containers at once

CREATE A SIMPLE HTML FILE

Create an empty project directory and using your favourite editor create a file named index.html

```
<!DOCTYPE html>
<html>
<body>

<p>Hello from the container</p>

</body>
</html>
```

CREATE A DOCKERFILE

Now create a Dockerfile to specify required configurations to create a simple Apache2 container.

Dockerfile

```
FROM centos:latest

RUN yum update -y
RUN yum install -y httpd
ADD ./index.html /var/www/html

ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]

EXPOSE 80
```

CREATE A DOCKER-COMPOSE.YML

We will create docker-compose.yml that will tell docker compose to build images to launch containers. We will define 'web' and 'redis' as the services for which we're building images. In the 'web' we are building an image based on Dockerfile.

docker-compose.yml

```
version: '2'
services:
  web:
    build: .
    ports:
    - "8080:80"
  redis:
    image: "redis:alpine"
```

BUILDING IMAGES AND LAUNCH CONTAINERS AT ONCE

Now we will use docker compose to pull images and run containers from that images. Instead of doing each step manually we can use a single docker compose command.

```
docker-compose up
```

This will perform a build and will start containers based on these images.

EXAMPLE DOCKER-COMPOSE.YML

```yaml
version: "3"
services:
  app:
    image: node:alpine
    volumes:
      - ./:/app
    working_dir: /app
    depends_on:
      - mongo
    environment:
      NODE_ENV: development
    ports:
      - 3000:3000
    command: npm run dev
  mongo:
    image: mongo
    expose:
      - 27017
    volumes:
      - ./data/db:/data/db
```

This docker-compose.yml file creates two services namely app (App's code) and mongo(database). Each service is basically a container. Notice the use of depends_on Command, it tells Compose that in order for app to run it needs mongo to be running.

LAB - BUILD WORDPRESS BLOG

- Create a project directory
- Create docker-compose.yml file
- Build the project
- Complete Setup

CREATE A PROJECT DIRECTORY

Create an empty project directory, and navigate to that directory.

```
$mkdir Wordpress && cd wordpress
```

Create docker-compose.yml

Use your favourite editor and create a file named: docker-compose.yml with the following content. We'll use persistent volume so that changes made to our database are persistent.

```yaml
version: '3.3'
services:
  db:
    image: mysql:5.7
    volumes:
    - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
    - db
    image: wordpress:latest
    ports:
    - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```

BUILD THE PROJECT

Now we'll launch all our containers in detached mode. This will pull all the needed images to build docker containers

```
$ docker-compose up -d
```

COMPLETE SETUP

Paste your host ip address in the browser and complete the wordpress installation steps.

IMAGE BUILD USING COMPOSE

- Using compose is basically a three step process
  - Define application environment with Dockerfile.
  - Define services that make up your application in - docker-compose.yml.
  - Run docker-compose up, it will start your entire - application.

LAB - RUNNING MICROSERVICES USING COMPOSE

We will be using the same application as in the previous weekâ□□s lab, frontend and backend.

Pull the latest changes

```
git pull https://github.com/kahootali/docker-samples master
```

CHECK THE DOCKER COMPOSE FILE

Go to microservices/ folder

```
cd microservices/
```

Check the Docker Compose file

```
version: '3.7'
services:
  backend:
    build:
      context: backend-python/
      dockerfile: Dockerfile
    init: true
    networks:
    - application
  frontend:
    build:
      context: frontend-node/
      dockerfile: Dockerfile
    init: true
    networks:
    - application
    ports:
    - "9091:8080"
    depends_on:
    - backend
networks:
  application:
    name: application
```

Run

```
docker-compose up
```

We are creating two containers, frontend and backend. There are two ways of using an image. You can build the image giving the Dockerfile path

```
backend:
   build:
     context: backend-python/
     dockerfile: Dockerfile
```

Specify the already built image

```
image: frontend-app
```

We have specified Frontend is dependant on backend so it will first run backend and then run frontend container and when shutting down, it will first stop frontend and then backend.

We have also specified the network for both containers so they can communicate with each other.

SCALE MICROSERVICE

Now we will scale a microservice using Docker Compose, in above folder run

```
docker-compose scale frontend=2
```

It will give error of port binding. Now try to scale backend

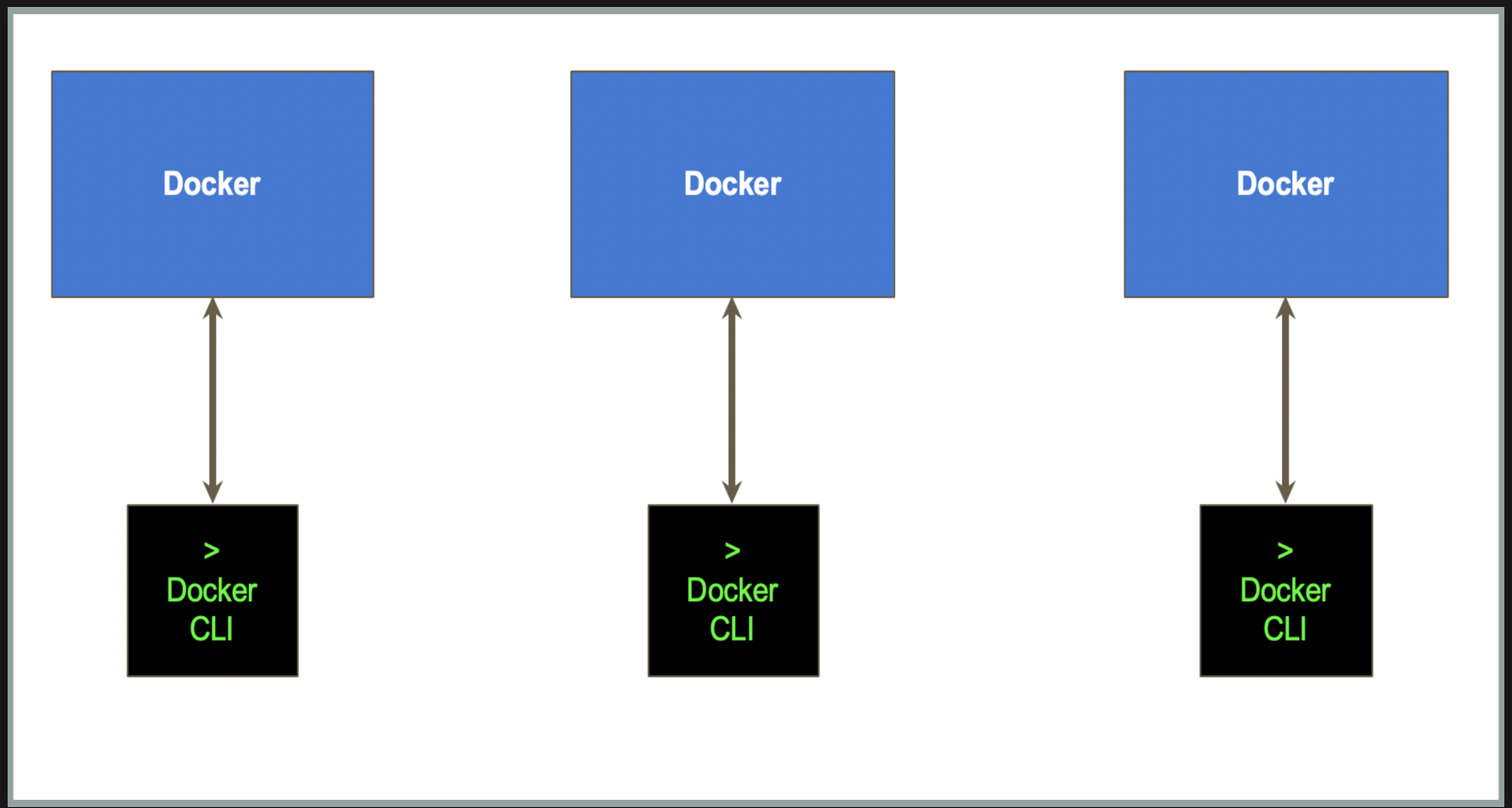
```
docker-compose scale backend=2
```

It will get scaled to 2. Now 2 replicas are running for Backend but 1 replica for Frontend. If we want to scale Frontend, we would have to change the ports section to:

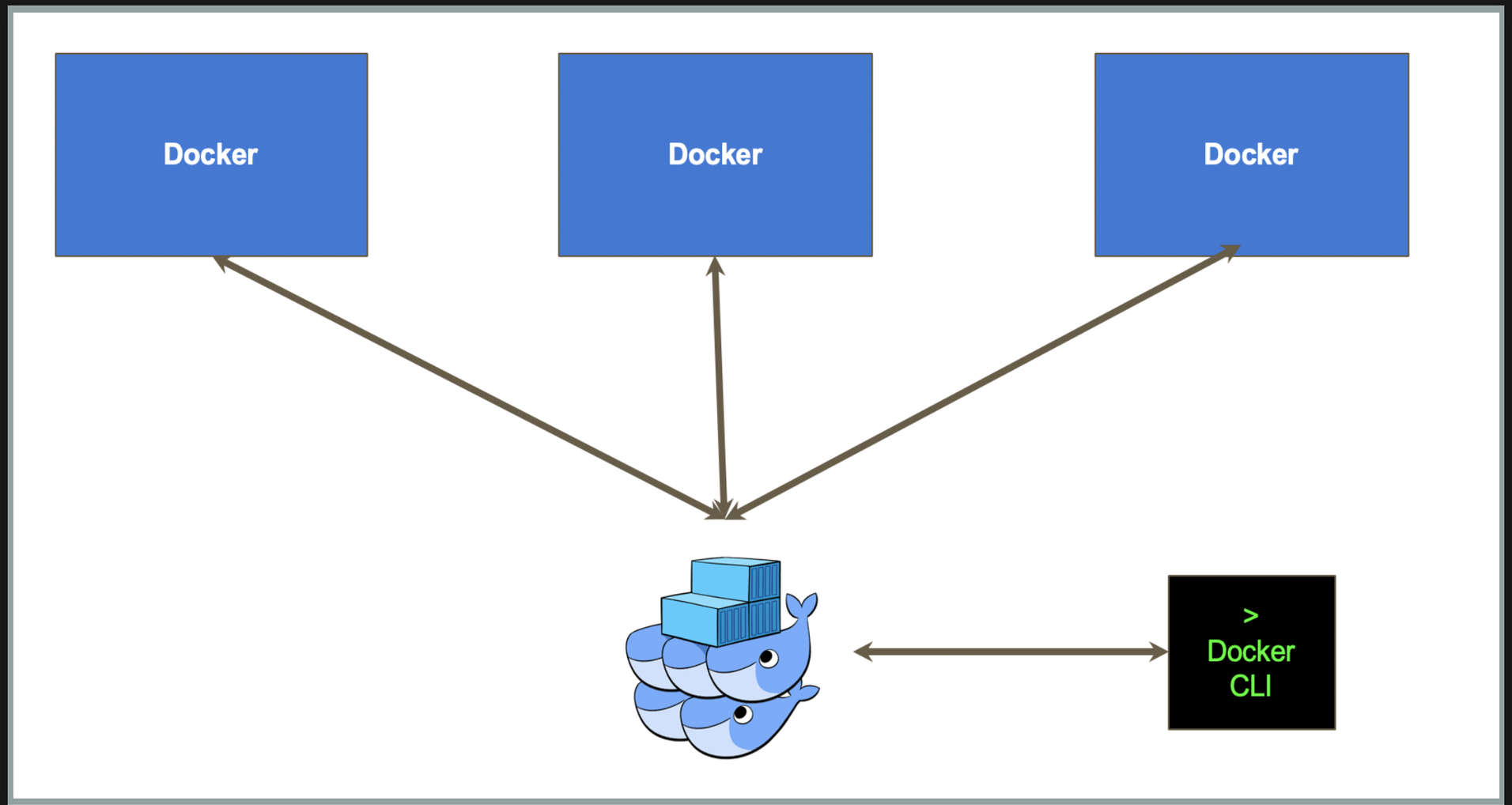
```
    ports:
    - "9091-9095:8080"
```

Now Frontend can use any of ports 9091-9095 when scaled up.

TRADITIONAL MODEL



SWARM



WHAT IS DOCKER SWARM

- Clustering and scheduling tool for containers
- Helps to manage cluster of docker nodes as a single - virtual system.
- Manager and worker nodes
- Cluster of docker engine or nodes
- Two major components of Swarm
  - Cluster (can consist of one or more docker nodes)
  - Orchestration engine

FEATURES

- Cluster management
- Decentralize design
- Scaling
- Declarative service model
- Desired state reconciliation
- Multi Host network
- Load balancing
- Secure by default
- Rolling updates

SWARM CLUSTER

- Cluster consists of one or more docker nodes
- Either manager or a worker
- Manager
  - Manages state of cluster
  - Dispatches tasks to workers
- Worker
  - Accept and execute tasks
- Swarm uses Transport Layer Security (TLS)
  - Encrypted communication
  - Authenticate nodes

ORCHESTRATION

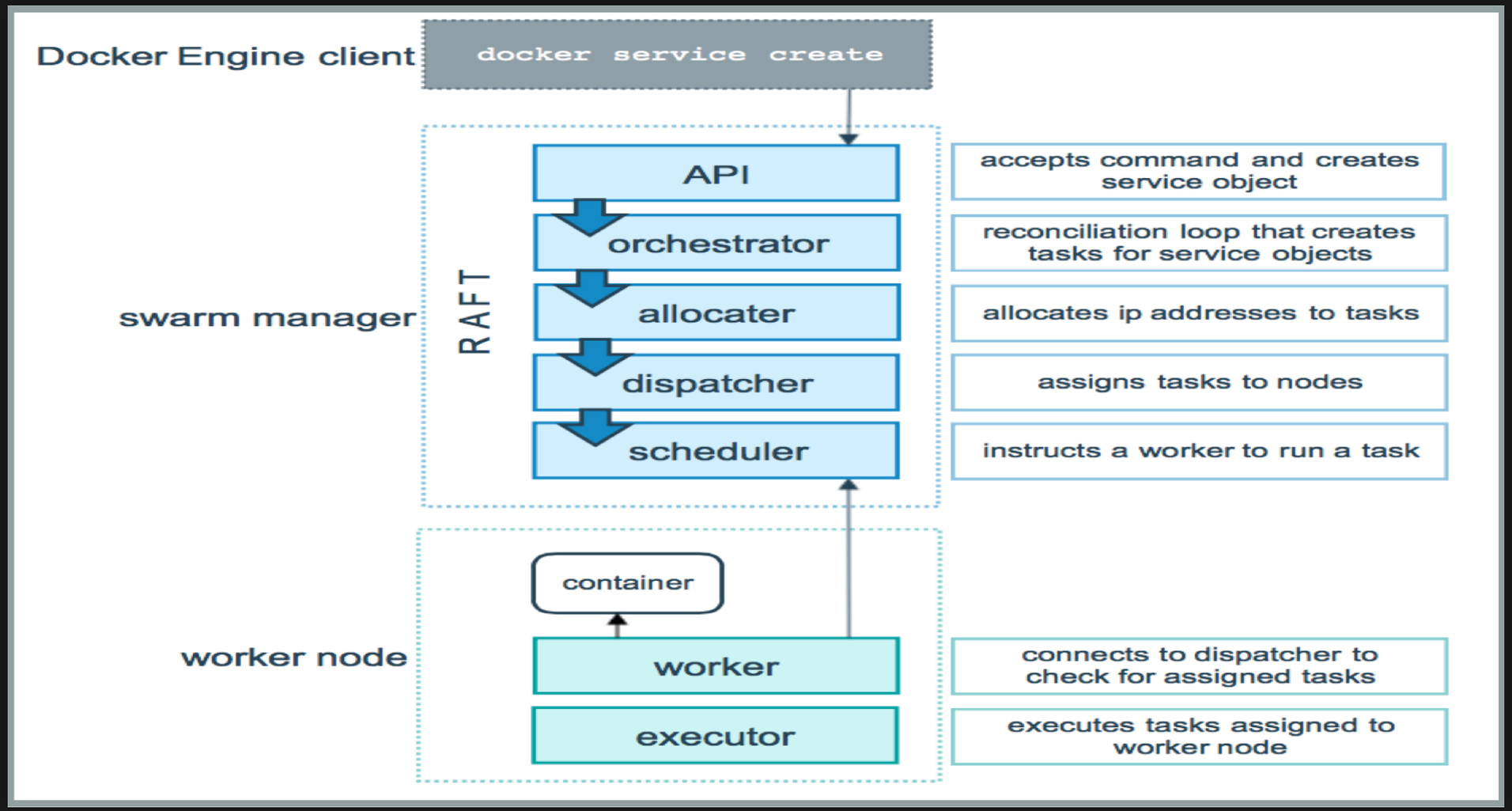
- In Swarm atomic unit of scheduling is the service.
- Service construct adds the following to a container
  - Scaling
  - Rolling update
  - Roll back
- Container wrapped in a service is a task or replica

SWARM CONCEPTS

- Swarm: a swarm consists of multiple Docker hosts which - run in swarm mode and act as managers and workers.
- Task: the swarm manager distributes a specific number - of tasks among the nodes. A task carries a Docker - container and the commands to run inside the container.
- Service: definition of the tasks to execute on the - manager or worker nodes.
- Nodes: a swarm node is an individual Docker Engine - participating in the swarm.

SWARM CONCEPTS

- Manager nodes: dispatches units of work called tasks to - worker nodes. Manager nodes also perform orchestration - and cluster management functions.
- Leader node: manager nodes elect a single leader to - conduct orchestration tasks, using the Raft consensus - algorithm.
- Worker nodes: receive and execute tasks dispatched from - manager nodes.
- Load balancing: the swarm manager uses ingress load - balancing to expose the services running on the Docker - swarm, enabling external access.


Docker Engine client
docker service create
swarm manager
RAFT
API
orchestrator
allocater
dispatcher
scheduler
accepts command and creates service object
reconciliation loop that creates tasks for service objects
allocates ip addresses to tasks
assigns tasks to nodes
instructs a worker to run a task
worker node
container
worker
executor
connects to dispatcher to check for assigned tasks
executes tasks assigned to worker node

SWARM MODE

- Docker engine 1.2 or newer features swarmkit integration.
- Docker CLI commands
  - docker swarm (enable swarm mode, join a swarm, adjust - cluster parameters)
  - docker node (view nodes, promote/demote managers, manage - nodes)
  - docker service (create and manage services)

DOCKER NODE COMMANDS

- demote: demotes one or more nodes from manager
- inspect: Displays detailed information on one or more - nodes
- ls: List nodes in swarm
- promote: promotes one or more nodes to manager in swarm
- ps: list tasks running on one or more nodes
- rm: Removes one or more nodes from swarm
- update: Updates a node in Swarm

DOCKER SWARM COMMANDS

- ca: Displays and rotate the root CA
- init: Initializes a swarm
- join: join as swarm as a node/manager
- join-token: manages join token
- leave: Leaves the swarm
- unlock: unlocks swarm
- unlock-key: Manages the unlock key
- update: Updates the swarm

LAB - RUN DOCKER IN SWARM MODE

DevOps course by Ali Kahoot - Dice Analytics

GET PRIVATE IP ADDRESS OF YOUR HOST MACHINE

Execute ifconfig on your host machine and copy the private IP address of your host machine.

```
$ ifconfig
```

CHECK FOR DOCKER VERSION

Check for your docker version on your host machine, as Docker swarm is not a part of docker version lower than 1.12

```
$ docker --version
```

INITIALISING THE MANAGER

Use swarm init command to advertise the manager, we will use advertise-addr flag and pass the private IP of our host system. This will be advertised to other members of the swarm

```
$ docker swarm init --advertise-addr [PRIVATE_IP]
```

LIST NODES IN THE SWARM

Get a list of nodes in the Swarm. Each of the node reporting back to manager has the state ready.

```
$ docker node ls
```

SWARM SERVICES

- A service is definition of tasks to execute on manager - or worker nodes
- Primary root of user interaction in Swarm
- We need to specify image, and which commands to execute - inside running container at time of creating a service

SWARM SERVICE COMMANDS

- create: Creates a new service
- inspect: Displays detailed information on one or more - services
- logs: Fetch logs of a services/task
- ls: list services
- ps: List tasks of one or more services
- rm : removes one or more services
- rollback: revert changes to a service configuration
- scale: scales one or multiple replicated services
- update: updates a service

LAB - WORKING WITH SWARM SERVICES

- Create a simple service
- List docker swarm services
- Inspecing
- Scaling service

CREATE A SIMPLE SERVICE

We will start nginx service in swarm from nginx image

```
$docker service create -d --name nginx_service -p 8080:80 --replicas 2 nginx:latest
```

LIST DOCKER SWARM SERVICES

Getting a list of services weâ□□ve created in the previous step

```
$ docker service ls
```

INSPECT

Check details of your newly created services.

```
$ docker service inspect nginx_service
```

SCALING SERVICES

We can scale our service to our desired capacity, for nginx weâ□□ve created 2 replicas. Letâ□□s make it 3.

```
$ docker service scale nginx_service=3
```

SWARM STACK

- Used to manage a multi-service application
- Group of services that are deployed together.
- Method of using compose files to run an application

SWARM STACK 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

LAB - CREATING STACK

- Create a compose file
- Deploy the stack
- Complete the setup

CREATE A COMPOSE FILE

Using your favourite editor create a file named: docker-compose.yml and paste the following content in that file

```yaml
version: '3'

services:
  db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     networks:
       mysql_internal:
     environment:
       MYSQL_ROOT_PASSWORD: P4ssw0rd0!
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: P4ssw0rd0!

   blog:
     depends_on:
       - db
     image: wordpress:latest
     networks:
       mysql_internal:
       wordpress_public:
     ports:
       - "80:80"
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: P4ssw0rd0!
volumes:
    db_data:
networks:
  mysql_internal:
    internal: true
  wordpress_public:
```

DEPLOY THE STACK

Finally weâ□□re ready to deploy our stack and as a result weâ□□ll have a wordpress site running.

```
$ docker stack deploy --compose-file docker-compose.yml up
```

COMPLETE THE SETUP

Navigate to IP of your manager node, complete the required steps for wordpress installation.