upGrad

#LifeKoKaroLift

Introduction to Docker Orchestration

upGrad

Course: Containerisation at Scale

Lecture On: Docker Orchestration

Instructor: Dipesh Garg

upGrad

Prerequisite for this Session

Basic Understanding of Docker

upGrad

Today's Agenda

- Docker basic concepts - recap
- Setting up an application using Docker Compose
- Issues with Docker Compose
- What and why: Docker orchestration
- Introduction to Docker Swarm and its architecture
- Deploying an application in a Swarm cluster
- Basic features of Swarm
- Docker Compose versus Swarm

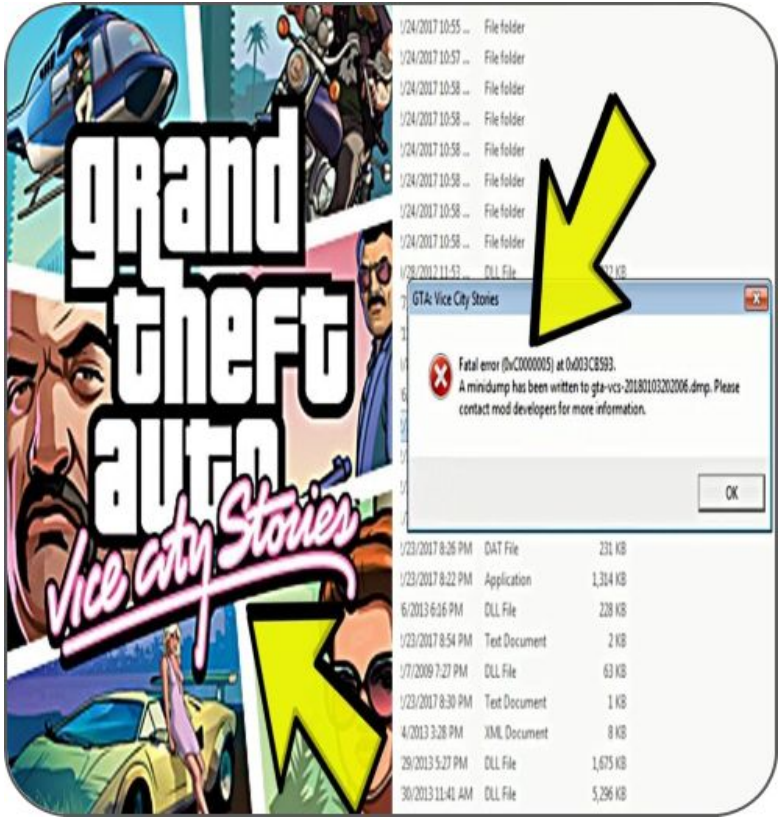
What is Docker?

What is Docker?

upGrad

Rember GTA Vice City Installation?

- Download the installer
- Run the installer
- **Error message during installation**
- Troubleshoot the issue
- Re-run the installer
- **Another issue arises**



https://images.app.goo.gl/cSnrBcWHSRwBnmqk9

What is Docker?

upGrad

- Docker is an open platform. Once we build a Docker container, we can run it anywhere, say, Windows, Linux, a data centre or in a cloud.
- It is a standardised unit, which contains everything that a software application needs to run in any environment, including application code, system libraries and other dependencies.
- It is lightweight, open and secure.

VM Versus Container

upGrad

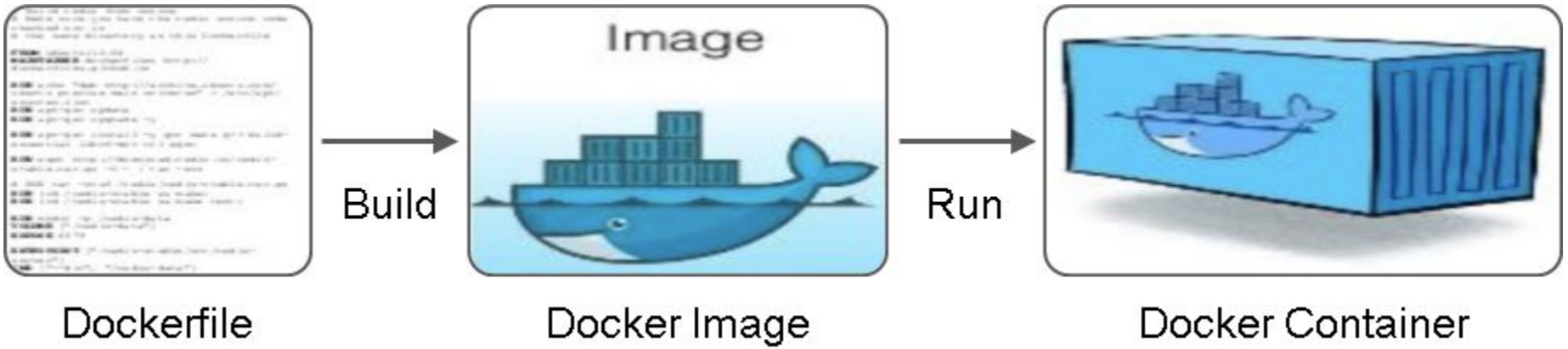
- Virtual Machine
  - It includes multiple applications on a single server.
  - The applications require all the resources and functionalities of the OS.
  - The applications need full isolation and security (since containers provide process-level isolation).
  - Network components that need high performance should be installed on virtual machines (VMs) rather than containers.

- Container
  - It is used to maximise the number of applications running on a server.
  - The applications will be lightweight and the start up time needs to be in milliseconds.
  - The applications need to be infrastructure independent.
  - Containers are more cost efficient than VMs.

Docker File and Image

upGrad

- A Dockerfile is a text document that contains all the commands that a user could call in the command line to assemble an image.
- The commands include FROM, RUN, CMD, ADD, COPY, ENTRYPOINT, VOLUME, USER and many more.



upGrad

Demo 1: Basics of Docker

- Write Dockerfile to create Nginx docker image
- Understanding the use of ADD versus COPY, CMD versus ENTRYPOINT

Poll 1 (15 seconds)

How can you check the list of the space taken by the Docker containers in a system? Like: Images, Containers, Volumes.

A. df -h

B. docker df -h

C. docker system df -h

D. docker system df

Poll 1 (15 seconds)

How can you check the list of the space taken by the Docker containers in a system? Like: Images, Containers, Volumes.

A. df -h

B. docker df -h

C. docker system df -h

D. docker system df

Poll 2 (15 seconds)

What is the docker command for displaying the layers of a Docker image?

A. docker image layers

B. docker history

C. docker layers

D. docker info

upGrad

Poll 2 (15 seconds)

What is the docker command for displaying layers of a Docker image?

A. docker image layers

B. docker history

C. docker layers

D. docker info

Docker Compose

upGrad

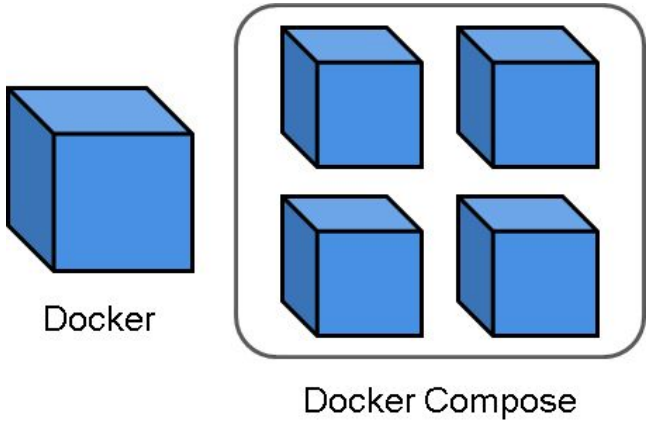
- Compose is a tool for defining and running multi-container Docker applications.
- With compose, you use a YAML file to configure your application's services.
- You can start all the services with a single command: docker-compose up.
- To stop all Docker containers, you can use the command *docker-compose down*.

Features of Docker Compose

upGrad

Here are the features of Docker Compose:

- Multiple isolated environments on a single host
- Preserves volume data when containers are created
- Only recreates containers that have changed



Docker Compose File

upGrad

```
version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Docker Compose Restart Policies

upGrad

- They specify the restart policies for how a container should or should not be restarted on exit.

- **Restart Policies**
  - **no**
    Do not restart the container (default one) automatically
  - **on-failure**
    Restart the container if it exits due to an error
  - **always**
    Always restart the container if it stops
  - **unless-stopped**
    Always restart the container if it stops, except when done manually

Docker Compose ‘Env Variables’ & ‘Depends On’

upGrad

- Environment variables can be set in a service’s containers with the ‘environment’ key.

```
web:
  environment:
    - DEBUG=1
    - KEY=VALUE
```

- **Depends On**
  - It expresses the dependency between services.
  - **docker-compose up** will start the services in the order of dependency. In this example, the ‘db’ container will begin before the ‘web’ container:

```
web:
  depends_on:
    - db
db:
```

Docker Compose Demo

upGrad

- Let’s set up a complete application using Docker Compose and see if it solves all of our problems.

- **Voting application:** https://github.com/dockersamples/example-voting-app

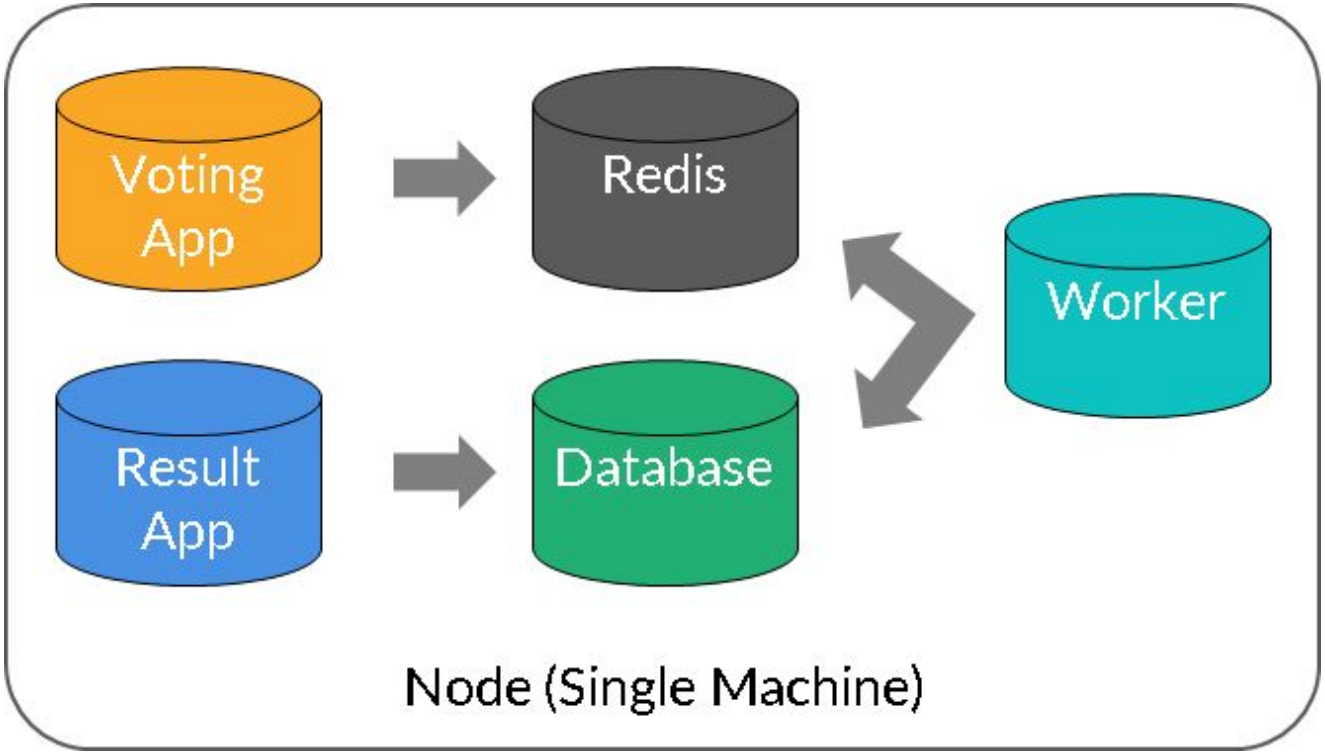
upGrad

Demo 2: Docker Compose

- Docker Compose concepts
- Running multiple containers using Docker Compose
- Deploying a voting application:
  - https://github.com/dockersamples/example-voting-app
- Issues with Docker Compose

Voting App Using Docker Compose

upGrad



upGrad

Poll 3 (15 seconds)

What does the Docker image prune command do?

A. Displays detailed information on one or more images

B. Shows the history of an image

C. Persists a Docker image

D. Removes unused images

upGrad

Poll 3 (15 seconds)

What does the Docker image prune command do?

A. Displays detailed information on one or more images

B. Shows the history of an image

C. Persists a docker image

D. Removes unused images

Poll 4 (15 seconds)

You want to run an Nginx web server container as a background process. You need to make sure the container's port 80 is mapped to port 8080 on the local host machine. As the final output, you should be able to access the default landing page of the Nginx web server. Which of these is the correct command for this?

A. docker run -itd web-server -P 8080:80 nginx

B. docker run -itd web-server -p 80:8080 nginx

C. docker run -itd web-server -P 80:8080 nginx

D. docker run -itd web-server -p 8080:80 nginx

Poll 4 (15 seconds)

You want to run an Nginx web server container as a background process. You need to make sure the container's port 80 is mapped to port 8080 on the local host machine. As the final output, you should be able to access the default landing page of the Nginx web server. Which of these is the correct command for this?

A. docker run -itd web-server -P 8080:80 nginx

B. docker run -itd web-server -p 80:8080 nginx

C. docker run -itd web-server -P 80:8080 nginx

D. docker run -itd web-server -p 8080:80 nginx

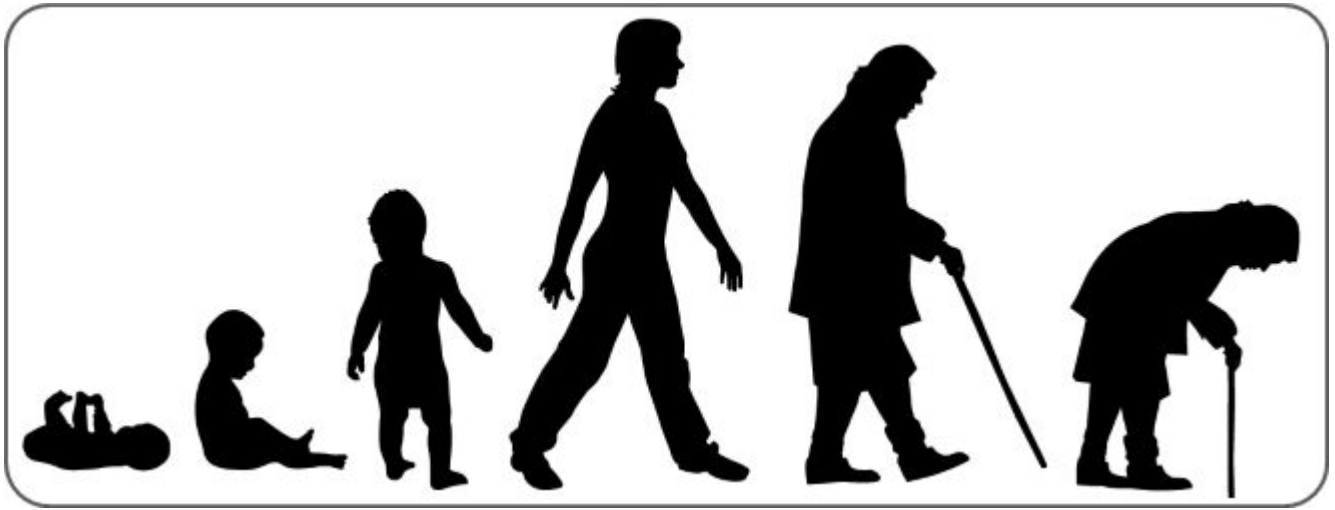
upGrad

Container Orchestration

Container Orchestration

upGrad

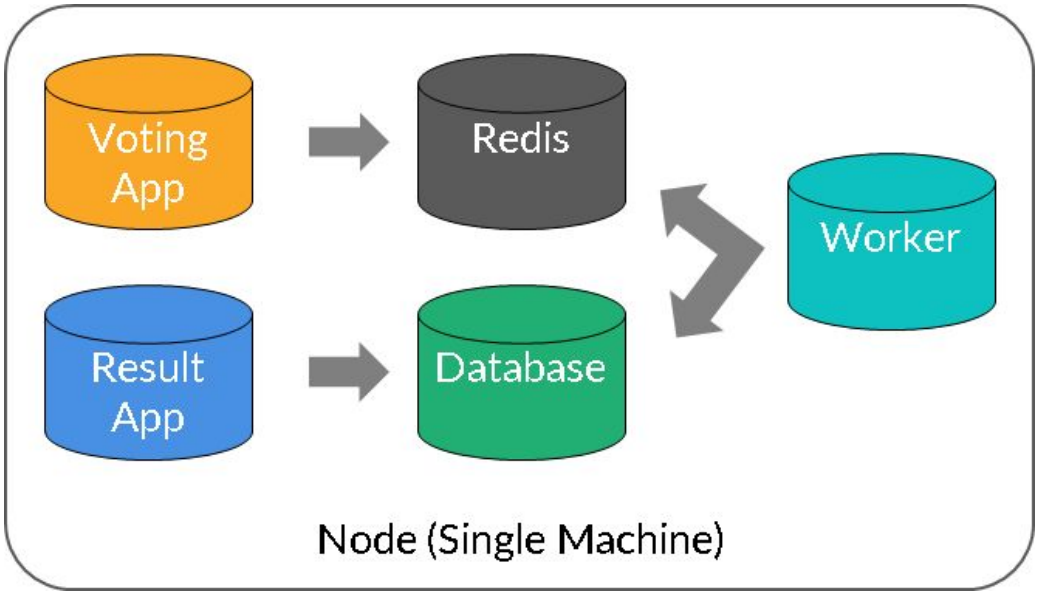
- Container orchestration is all about managing the life cycle of the containers.
- It helps with managing large, dynamic environments.



Voting Application

upGrad

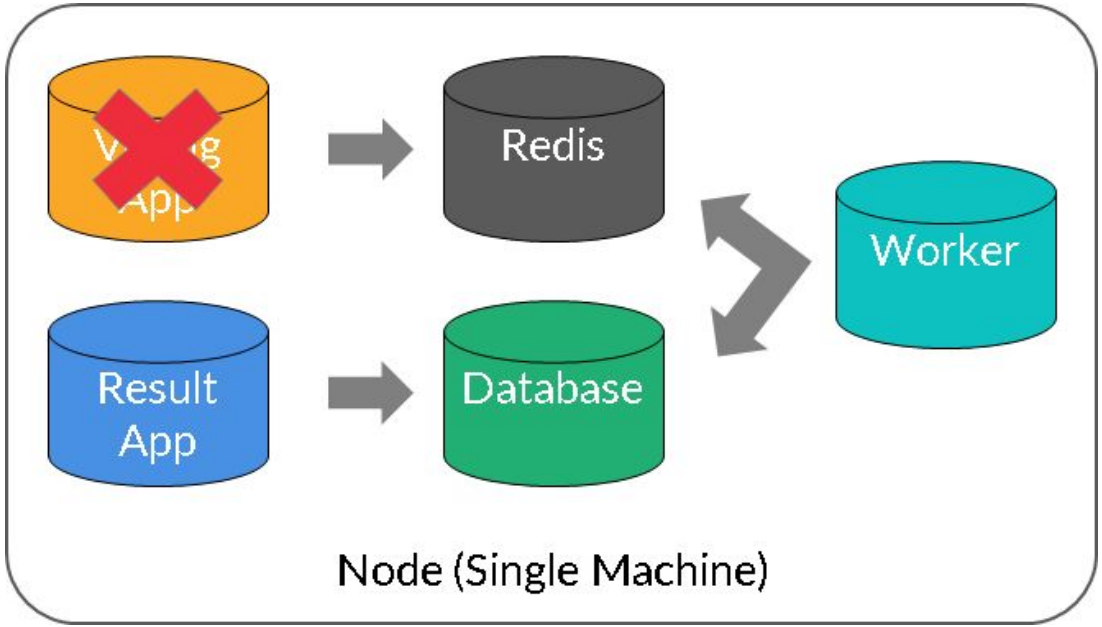
Take the example of the voting application, deployed using Docker Compose on a machine.



Voting App (Availability Challenge)

upGrad

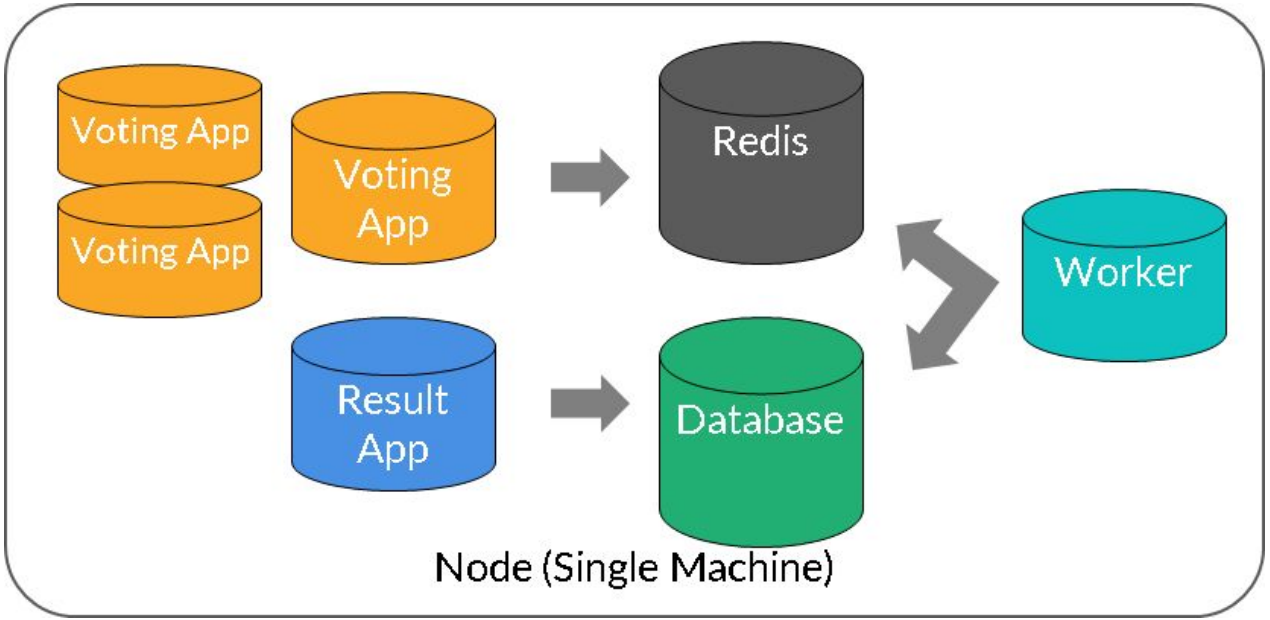
What if one of the applications goes down? How we can ensure availability of the application?



Voting App (Scaling Challenge)

upGrad

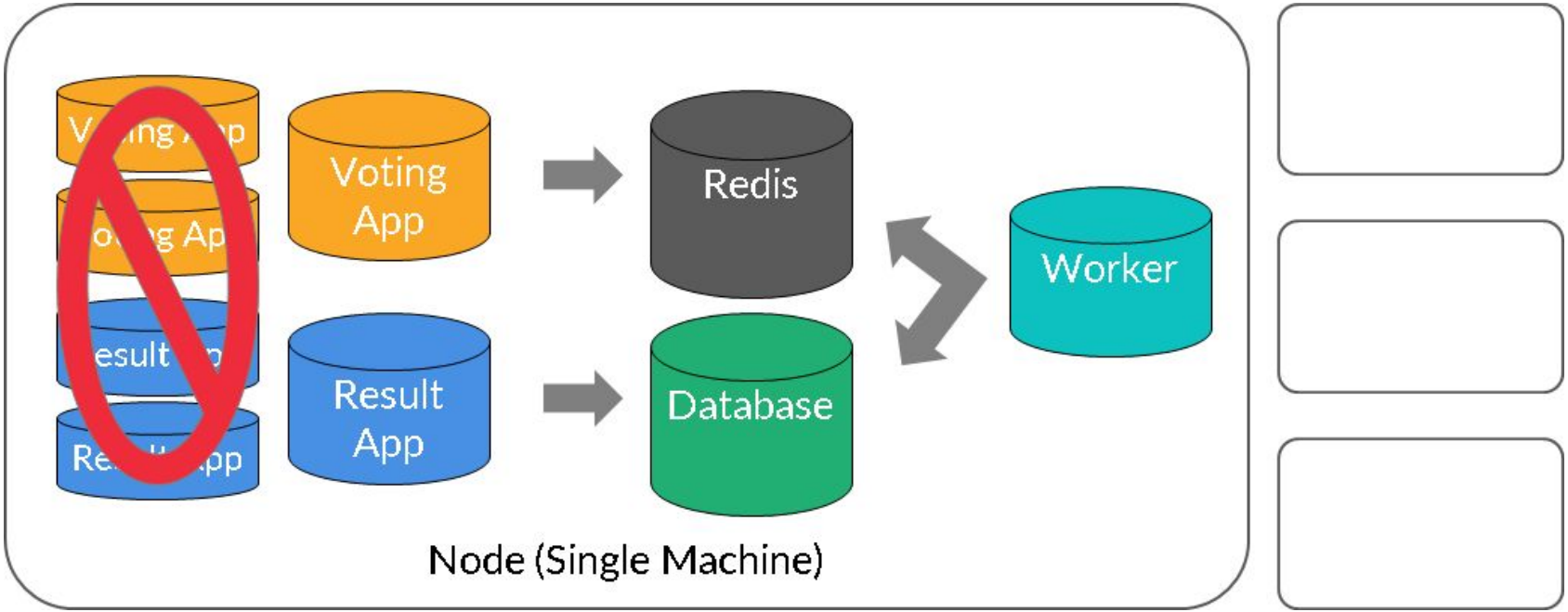
As requests increase, you need to scale your system as well.



Voting App (Multi-Node Challenge)

upGrad

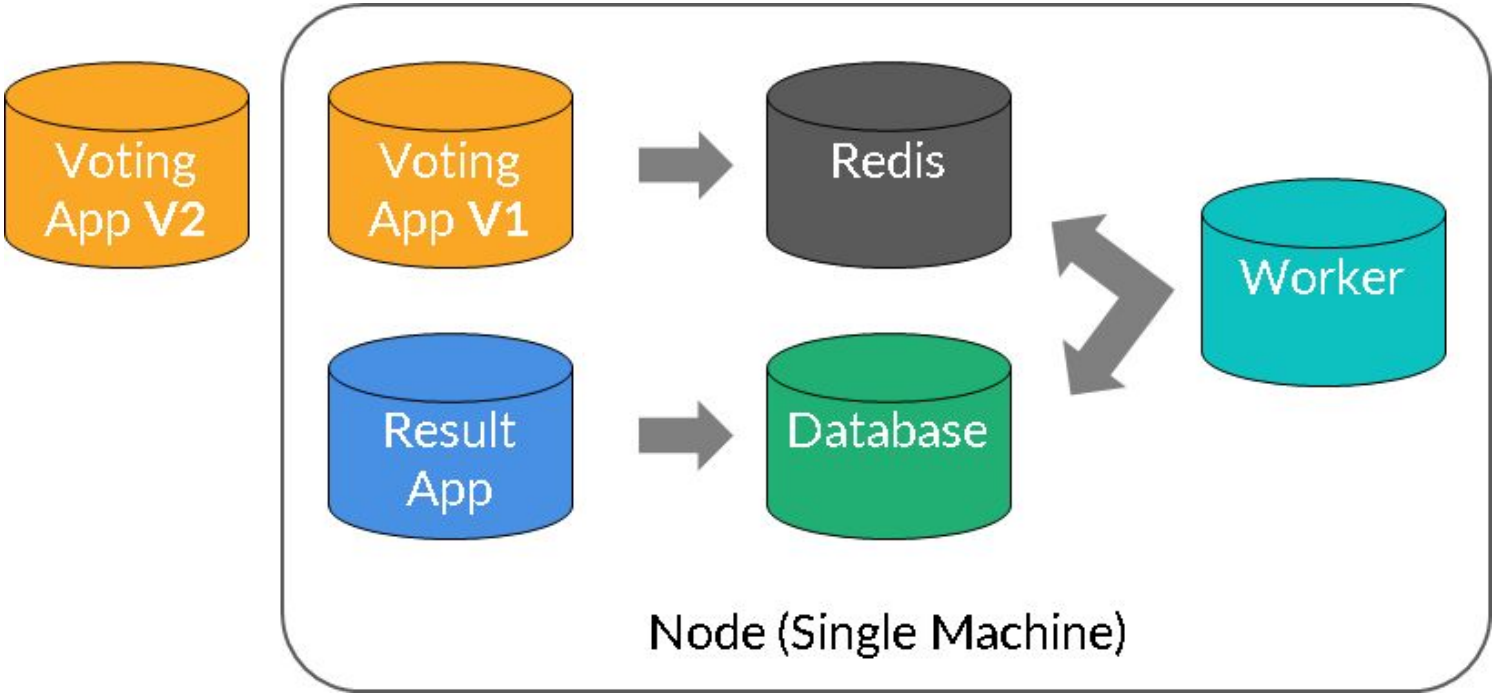
One node is insufficient to place a high number of containers.
But how do we shift or deploy containers on other nodes?



Voting App (Deployment Challenge)

upGrad

How do we replace V1 with V2 without affecting the end user?



Life Cycle of a Container

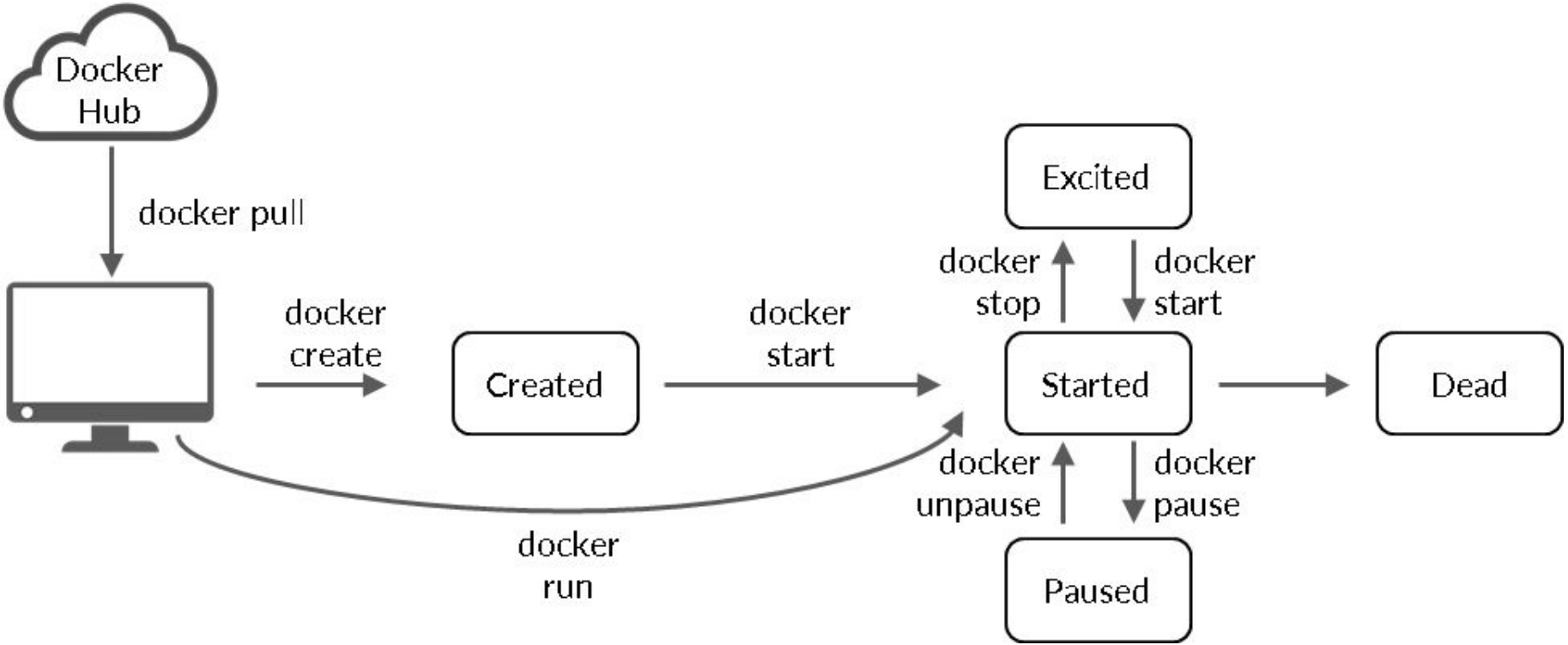
upGrad

The life cycle of a container includes three phases:

- **Starting phase:** Configuration phase - Container name, image name, port mapping, volume, network configuration
- **Running phase:** Scaling, load balancing, availability, health check
- **Deletion phase:** Clean-up of the resources associated with a container

Life Cycle of a Container

upGrad



Container Orchestration

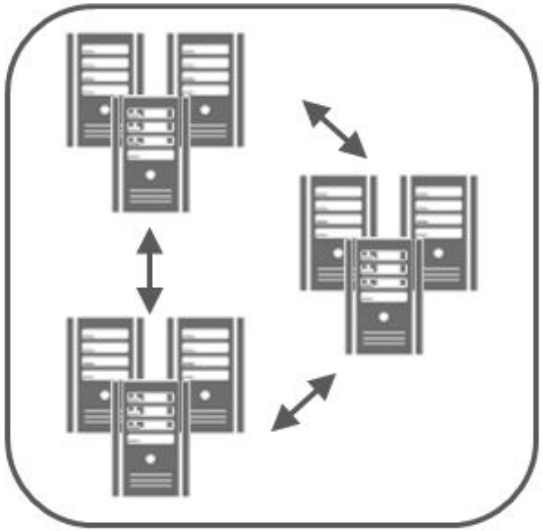
upGrad

Container Orchestration Engine Tool

Automate

Configuration
Availability
Load balancing
Scaling
Health check
Security
Networking

Application environment with multiple servers and containers



Container Orchestration Tools

upGrad

Different Orchestration Tools Available

- Docker Swarm
- AWS managed service - ECS
- K8s - EKS, AKS, GKS
- DigitalOcean Kubernetes Service
- Red Hat OpenShift Online

Docker Swarm

Docker Swarm

upGrad

What is Docker Swarm?

Docker Swarm is a container orchestration tool. It is a part of the Docker engine and helps to create, deploy and manage a cluster of Docker containers as a single virtual system.

It helps to manage:

Creation phase -> Running phase -> Deletion phase

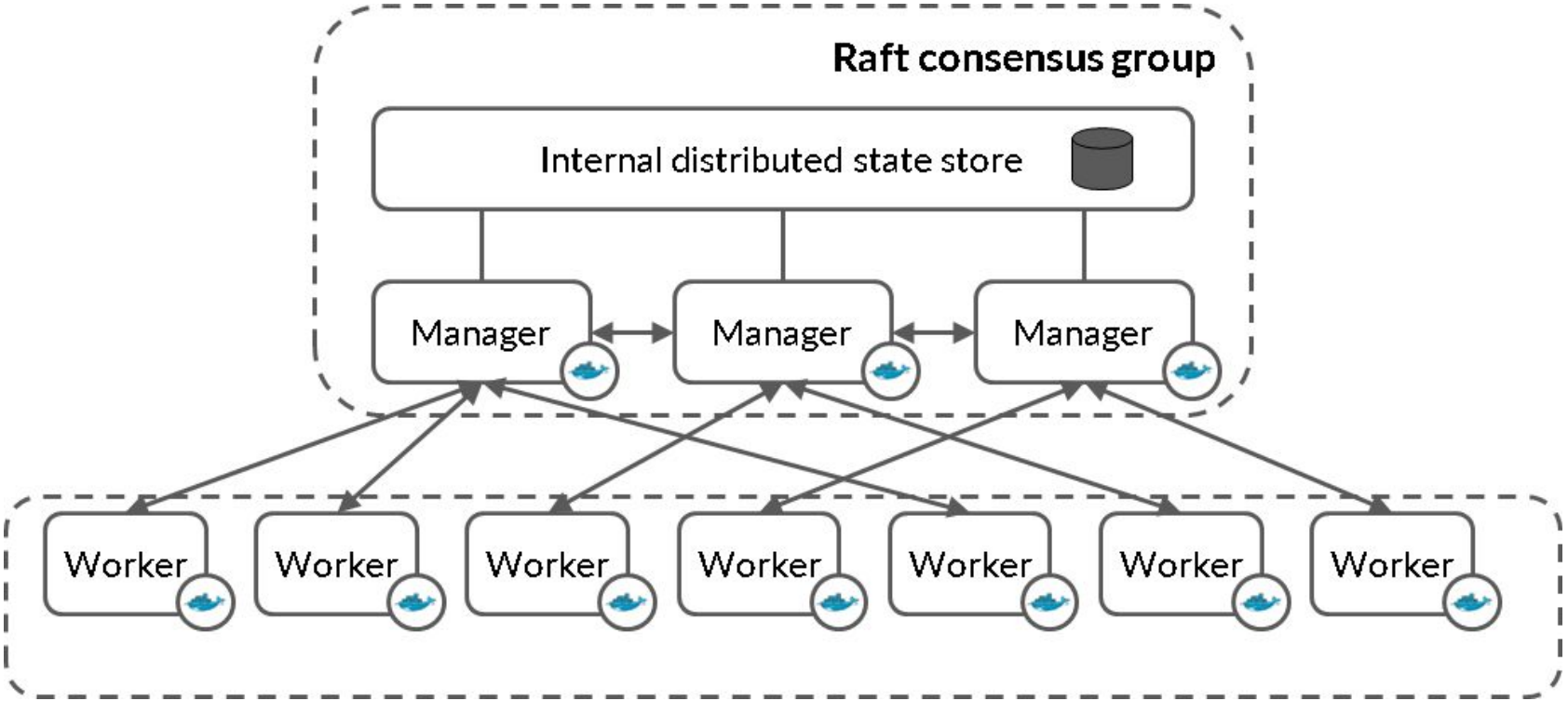
Docker Swarm Architecture

upGrad

- **Nodes -** A node is any machine having a Docker engine installed and participating in a Swarm cluster. We can check all the nodes in a Swarm cluster using the ‘docker node ls’ command.
- **Manager nodes -** The manager node acts as a manager and assigns units of work called tasks to worker nodes. It also manages cluster-management-level tasks.
- **Leader node -** If you have more than one manager node in the Swarm cluster, then the manager nodes elect a single node called the leader to perform orchestration and schedule tasks.
- **Worker nodes -** These are the slave nodes on which the manager schedules the tasks and the Docker containers run.

Docker Swarm Architecture

upGrad



Features of Docker Swarm

upGrad

- Docker engine integrated cluster management
- Scaling and load balancing
- Desired state reconciliation
- Multi-host networking
- Provision for security
- Option for rolling updates

Docker Swarm Initialisation

upGrad

- A Swarm cluster consists of master and worker nodes.
- A node is a machine having the Docker engine installed.
- The command for the manager node is:
  - docker Swarm init --advertise-addr <MANAGER_NODE_IP>
- The command for the worker nodes is:
  - docker Swarm join --token <TOKEN>

Docker Swarm Concepts

upGrad

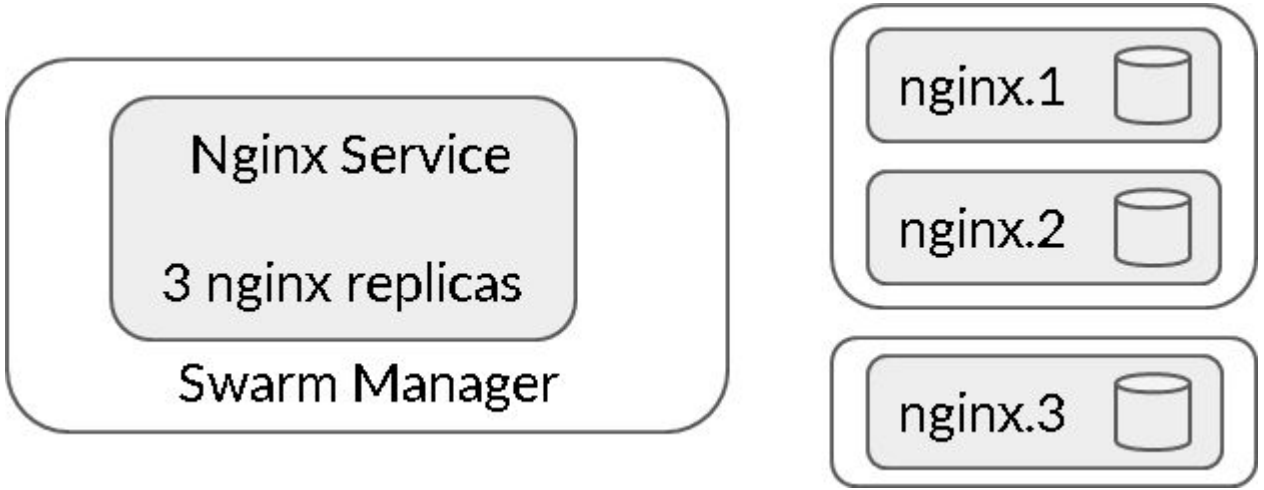
- **Task -** Carries the container and describes the container/blueprint:
  - Assume this to be the “Starting phase of the container”
  - Defines the container name, image, port mapping, logging
  - Contains all other container configurations, such as network, logging and CMD command.

- **Service -** Definition of the task to execute on the manager/worker node:
  - Assume this to be the “Running phase of the container”
  - **Defines:** The number of containers to run of a given task
  - Ensures availability, scaling and load balancing
  - Example: Run three containers of Nginx

docker service create --name webserver --replicas 3 nginx

Docker Swarm Concepts

upGrad



Docker Swarm Basic Commands

upGrad

- Execute these commands in a Swarm cluster and observe the output:
  - docker node ls
  - docker service ls
  - docker service create --name webserver --replicas 2 nginx
  - docker service ps webserver
  - systemctl stop docker in one of the worker nodes
  - docker service ps

```
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID            NAME         IMAGE         NODE              DESIRED STATE   CURRENT STATE            ERROR     PORTS
u2iyjkqgjac2  webserver.1  nginx:latest  ip-172-31-1-160   Running         Running 8 seconds ago
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

Docker Swarm Service Scaling

upGrad

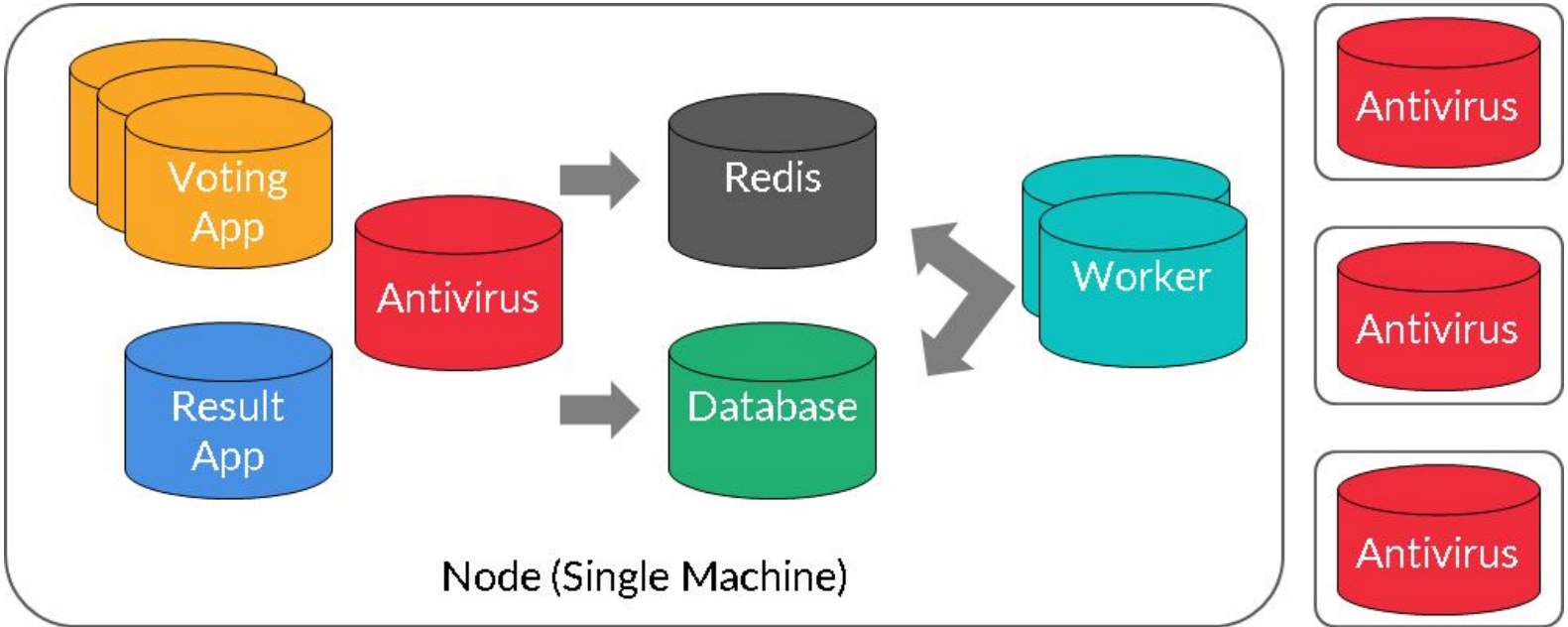
- Execute these commands in a Swarm cluster and observe the output:
  - docker node ls
  - docker service ls
  - docker service create --name webserver --replicas 2 nginx
  - docker service ps webserver
  - docker service scale webserver=4

```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service scale webserver=4
webserver scaled to 4
overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-5-46:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE          PORTS
rws2pzntalqt   webserver   replicated   4/4        nginx:latest
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID             NAME          IMAGE          NODE              DESIRED STATE   CURRENT STATE            ERROR     PORTS
u2iyjkqgjac2   webserver.1   nginx:latest   ip-172-31-1-160   Running         Running 4 minutes ago
vlus60ckr5m3   webserver.2   nginx:latest   ip-172-31-5-46    Running         Running 32 seconds ago
way8dqi4nwd5   webserver.3   nginx:latest   ip-172-31-5-46    Running         Running 32 seconds ago
icapf41xlwlp   webserver.4   nginx:latest   ip-172-31-1-160   Running         Running 35 seconds ago
ubuntu@ip-172-31-5-46:~$
```

Replicated Versus Global Service

upGrad

- **Replicated -** You specify the number of identical tasks that you want to run.
- **Global -** It is a service that runs one task on every node.



Replicated Versus Global Service

upGrad

- **Replicated**
  - docker service create --name replica --replicas 1 nginx
  - docker service ps

```
ubuntu@ip-172-31-5-46:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
ebrsyr16pj9i        replica             replicated          1/1                 nginx:latest
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

- **Global**
  - docker service create --name antivirus --mode global -dt ubuntu
  - docker service ps

```
ubuntu@ip-172-31-5-46:~$ docker service ps antivirus
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
kwjhbto26whp        antivirus.wtf9dy2ivefbldna30oiq9qvo   ubuntu:latest       ip-172-31-1-160     Running             Running 15 seconds ago
zlk47yvpxer6        antivirus.xae2pz1d1o8ho4boxpifwt7ut   ubuntu:latest       ip-172-31-5-46      Running             Running 19 seconds ago
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
na8op9ho2m6a        antivirus           global              2/2                 ubuntu:latest
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

Poll 5 (15 seconds)

Jamie works in the security team of his company, and as part of a log monitoring solution, he wants to push all the logs from the server to a central log-monitoring stack. He has created a container, which can do this automatically. The container should be part of all the servers. The DevOps team is using Swarm as an orchestrator. Which of these is the best way to achieve this?

A. Use *push all* service mode

B. Create replicas equal to the number of nodes

C. Use *global* service mode

D. Use *replicated* service mode

Poll 5 (15 seconds)

Jamie works in the security team of his company, and as part of a log monitoring solution, he wants to push all the logs from the server to a central log-monitoring stack. He has created a container, which can do this automatically. The container should be part of all the servers. The DevOps team is using Swarm as an orchestrator. Which of these is the best way to achieve this?

A. Use *push all* service mode

B. Create replicas equal to the number of nodes

C. Use *global* service mode

D. Use *replicated* service mode

Poll 6 (15 seconds)

Which of these commands is used to get the state of the tasks in a Swarm cluster?

A. docker service ls

B. docker service ps

C. docker status

D. docker stats

Poll 6 (15 seconds)

Which of these commands is used to get the state of the tasks in a Swarm cluster?

A. docker service ls

B. docker service ps

C. docker status

D. docker stats

Updates in Docker Swarm

upGrad

- Replacing a running Docker image version with a new version (replacing V1 with V2)

- During update, ensure that there is no downtime to the application. The end user should not be affected

- Rolling updates in Docker Swarm:
  - Incrementally replace existing containers with containers of a new version of the Docker image
  - **Example**: If four containers are running, first, we can replace two containers with a new version. Once deployed, we can replace the remaining two.

Updates in Docker Swarm

upGrad

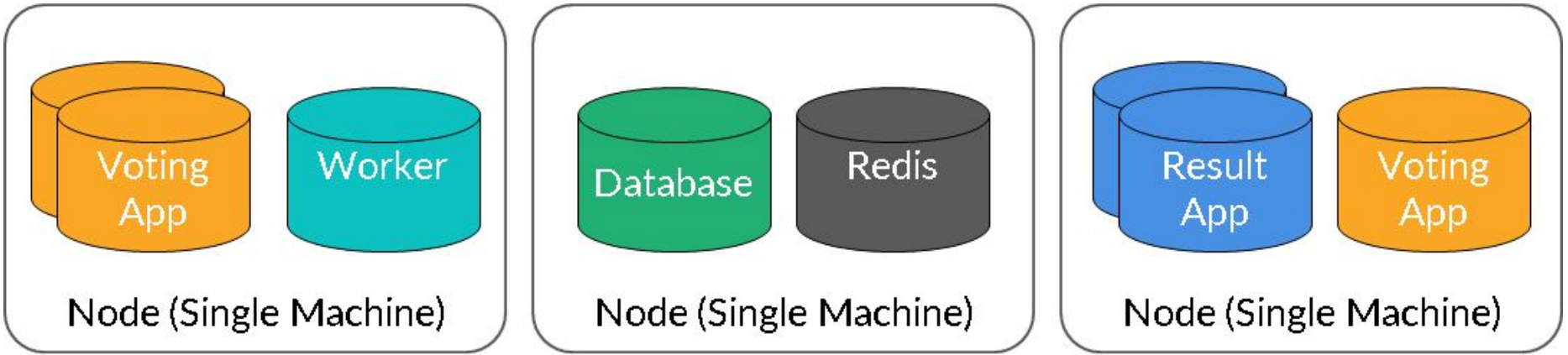
- docker service create --name webserver --replicas 1 nginx
- docker service scale webserver=4
- docker service update \
  --update-parallelism 2 \ *(updates two containers at a time)*
  --update-delay 20s \ *(configures the time delay between updates)*
  --image nginx:1.18-alpine \
  webserver

```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service update \
>       --update-parallelism 2 \
>       --update-delay 20s \
>       --image nginx:1.18-alpine \
>       webserver
webserver
overall progress: 2 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4:
4/4:
```

Bridge/Host Network

upGrad

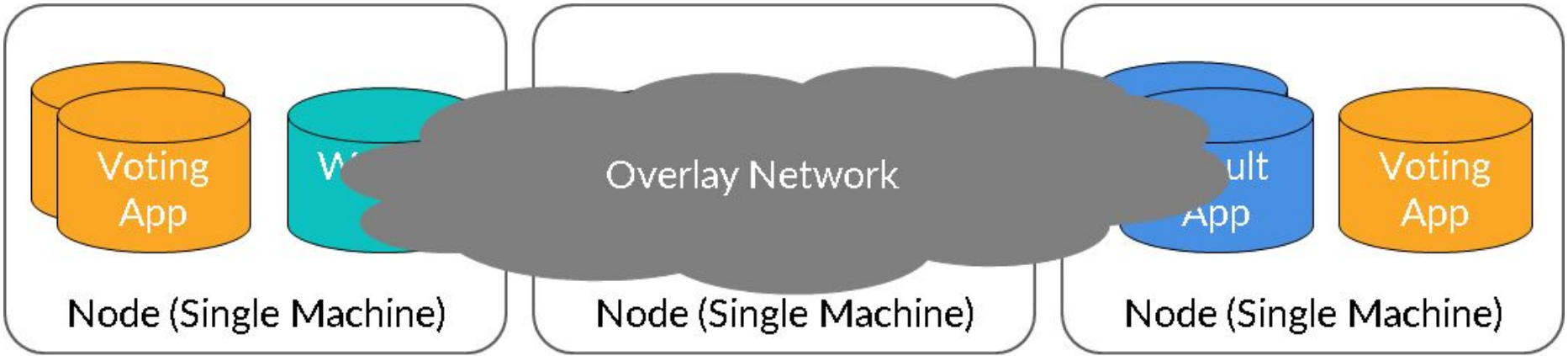
- Using the Bridge/Host network, containers on the same host can communicate with each other.
- How will the containers spread across multiple nodes interact?
- You need a new network to communicate with the other containers, which may be spread across any nodes in the Swarm cluster.



Overlay Network

upGrad

- The overlay network driver creates a distributed network among the multiple nodes present in the Swarm cluster.
- It allows the containers connected to it to communicate securely across the cluster.
- When you initialise a Swarm cluster, the overlay network (called ingress) is created by default.



Custom Overlay Network

upGrad

- You can create user-defined overlay networks using docker network create:
  - docker network create -d overlay my-overlay
  - docker network ls
- You can attach services to the overlay network as shown below:
  - docker service create --name my-overlay-service --network my-overlay --replicas 3 nginx

```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker network create -d overlay my-overlay
9nt9s40dpt4gghnao9m3m03q8
ubuntu@ip-172-31-5-46:~$ docker network ls
NETWORK ID          NAME                DRIVER    SCOPE
6a4aadc36bd9        bridge              bridge    local
daba1f4a5dff        docker_gwbridge     bridge    local
6d8bc3414275        host                host      local
mtc3c1xslwr6        ingress             overlay   swarm
9nt9s40dpt4g        my-overlay          overlay   swarm
9ab0271ffbaf        none                null      local
g1t9z3wyrxbg        webserver_default   overlay   swarm
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

Docker Stack

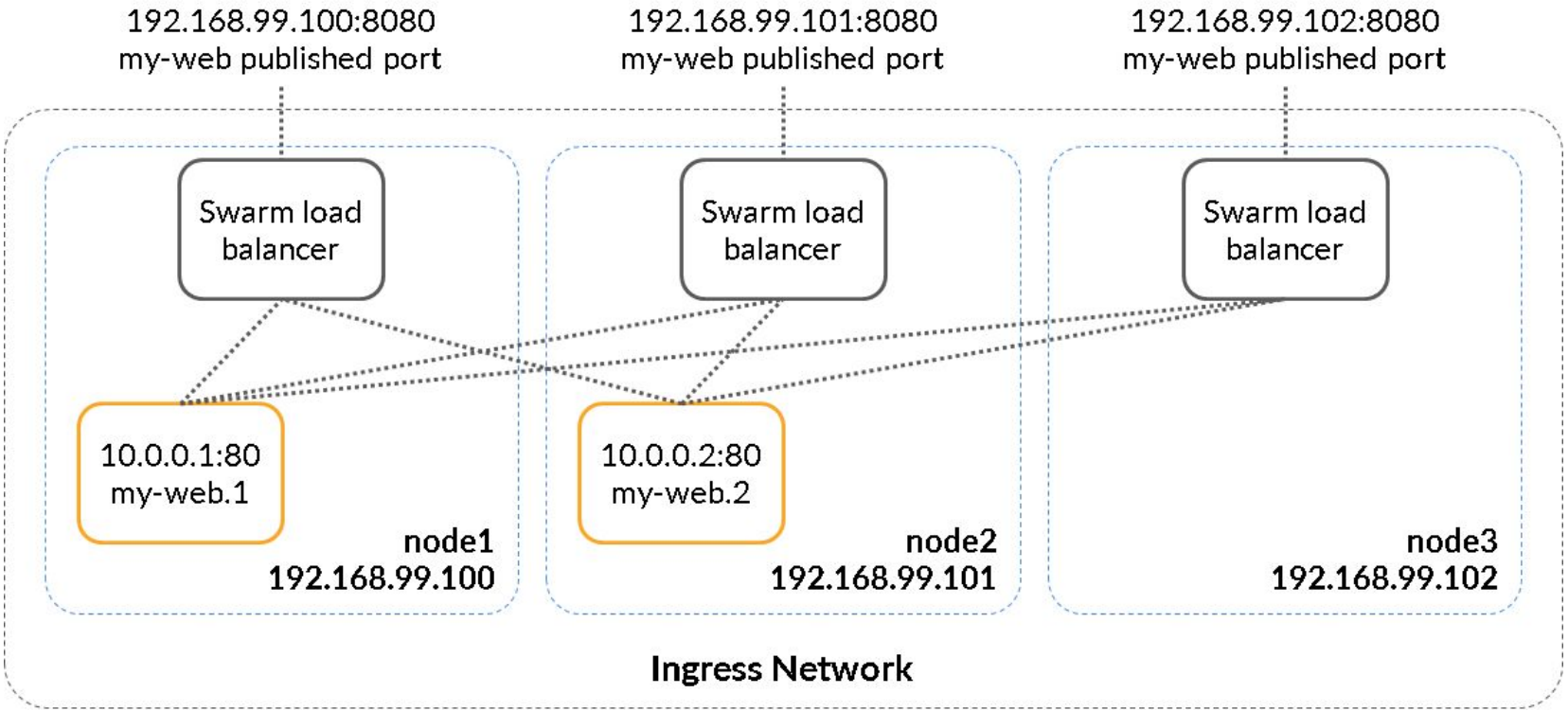
upGrad

- A stack is a group of interrelated services that share the same dependencies and can be orchestrated together.
- Example: Voting application contains Front-End + Back-End + Database.
- A stack can be deployed using a YAML file like the one that we define during docker-compose.
- We can define everything within the YAML file that we might define while creating a docker service.

  docker stack deploy --compose-file docker-compose.yml <NAME>

Publishing a Port in Swarm

upGrad



Docker Swarm Demo

upGrad

- Let’s set up a complete application using Docker Compose and see if it solves all of our problems.
- **Voting application:** https://github.com/dockersamples/example-voting-app

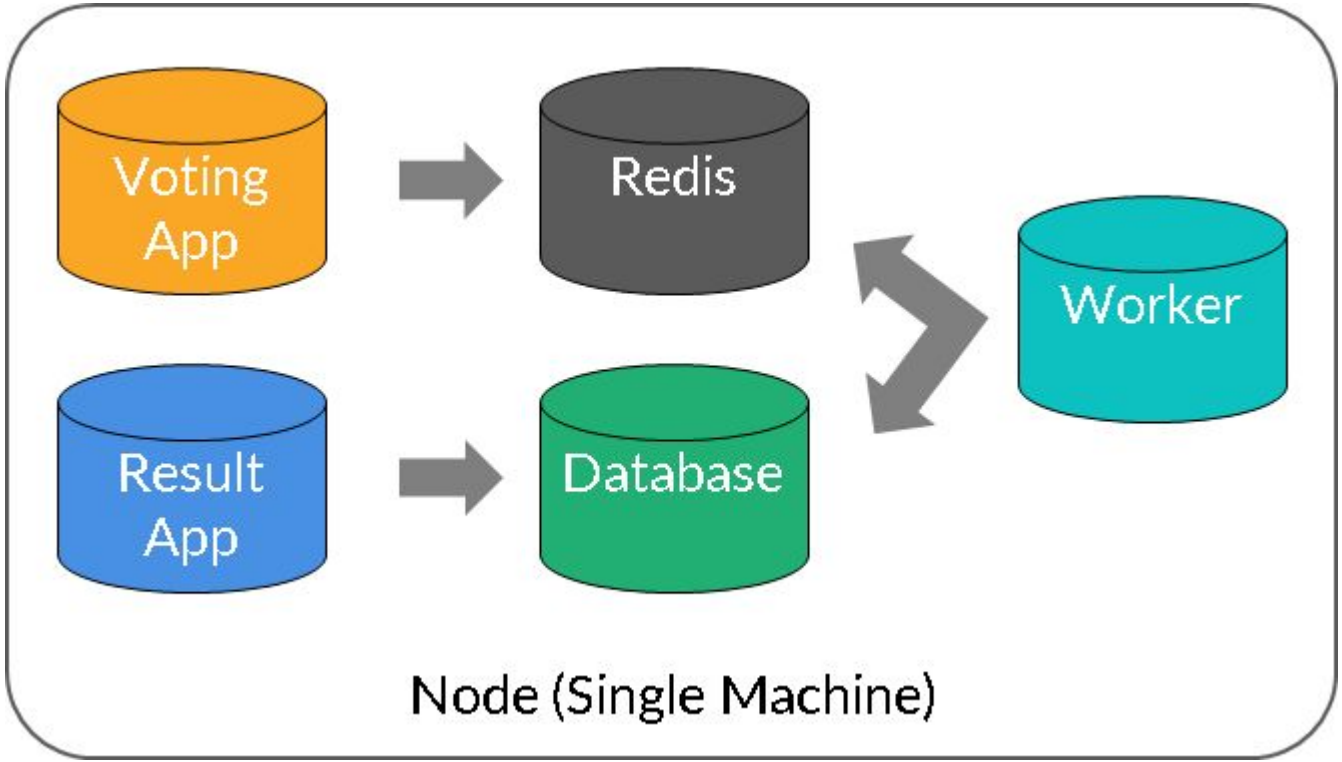
```
      placement:
        constraints: [node.role == manager]
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
```

Demo 3: Docker Swarm

- Setting up a three-node Swarm cluster
- Creating, updating and deleting Swarm services
- Overlay network and routing across multiple nodes
- Setting up a voting application using Swarm:
  - https://github.com/dockersamples/example-voting-app

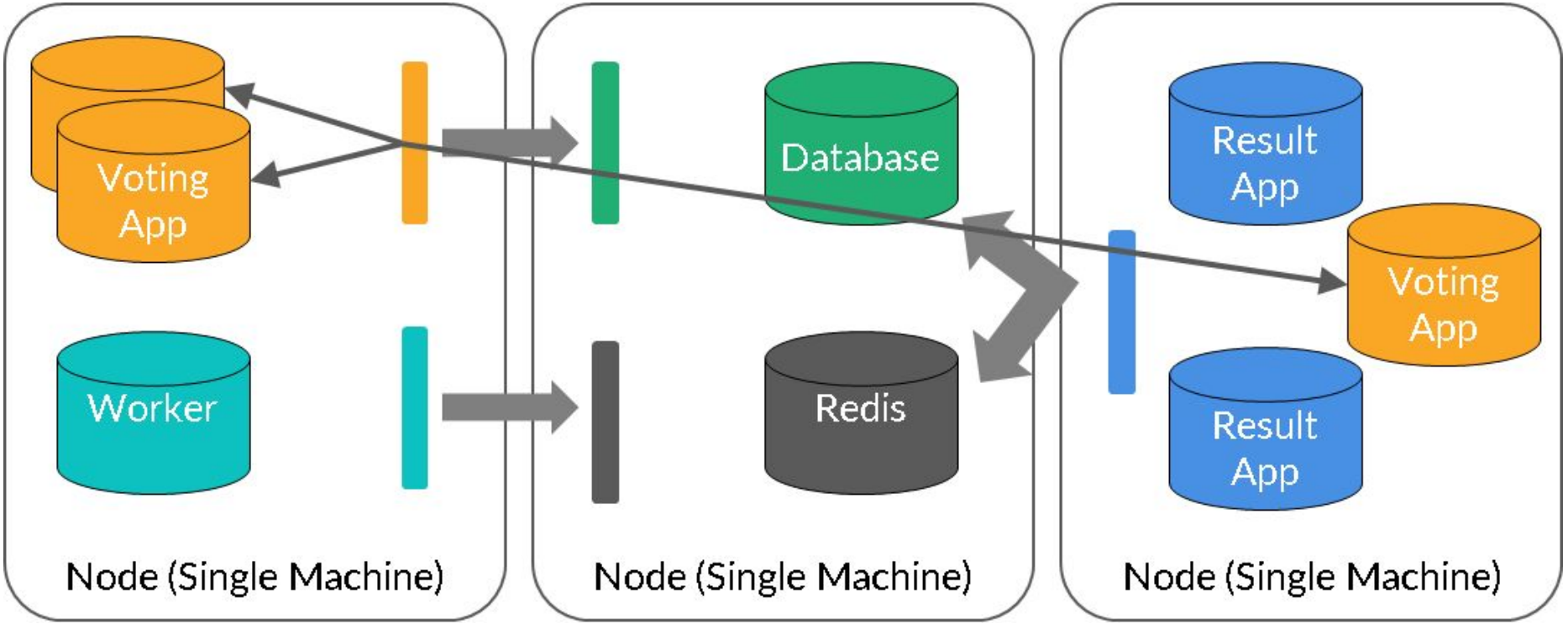
Voting App Using Docker Compose

upGrad



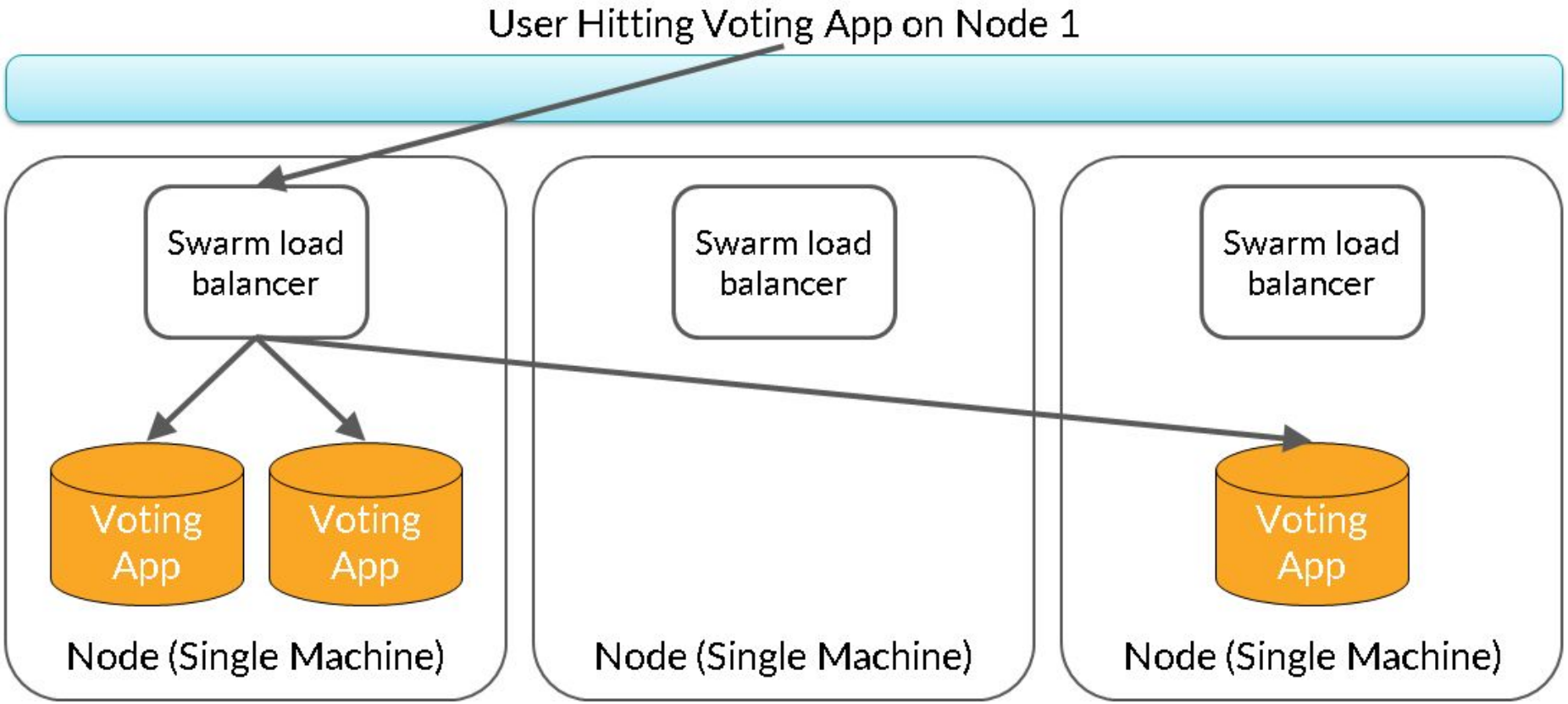
Voting App Using Docker Swarm

upGrad



Voting App Using Docker Swarm

upGrad



Docker Compose Versus Swarm

upGrad

- Swarm provides many features, which include a multi-node setup, scaling, availability, load balancing and many others.

- So, shouldn’t we use use Swarm always?

- Here are some use cases of Docker Compose:
  - In lower environments like dev/test, you can use Docker Compose.
  - In non-critical applications, you can use Docker Compose to save cost.
  - Docker Compose can be used in urgent and temporary situations.

Important Concepts and Questions

Important Questions

upGrad

1. What are the differences between ‘RUN’, ‘CMD’ and ‘ENTRYPOINT’ in Dockerfile?
2. What is the Docker build context?
3. In the Docker Compose port mapping definition, Andrew writes 80:80 whereas James writes only 80 for a web service. Can you tell the differences between them?
4. In which situations should you use Docker Compose?
5. What is Docker Orchestration? Name some of orchestration tools available in the market.
6. Can you explain the Swarm architecture and the differences between a service and a task?
7. Jack, a Front-End developer, asked Chris, a DevOps engineer, to scale his application. Chris is confused between the replicated and global service. Can you help him decide?
8. Explain the features of Swarm over running standalone Docker containers.

upGrad

Doubt Clearance Window

This class has covered the following topics:

upGrad

1. Docker basic concepts - recap
2. Setting up an application using Docker Compose
3. Issues with Docker Compose
4. What and why: Docker orchestration
5. Introduction to Docker Swarm and its architecture
6. Deploying an application in a Swarm cluster
7. Basic features of Swarm
8. Docker Compose versus Swarm

What's Next?

upGrad

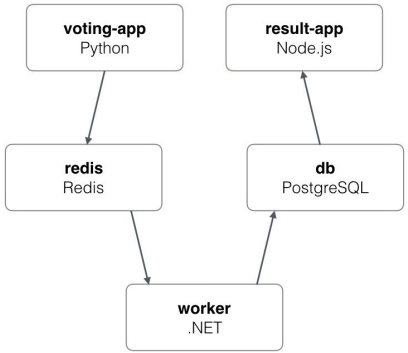
1. Nginx: Reverse proxy versus proxy, Nginx folder structure, virtual site hosting
2. Scaling and load balancing
3. Understand why, when and how to scale an application
4. Horizontal scaling versus vertical scaling
5. Load balancing types
6. Schedule services based on different strategies and constraints
7. Limit downtime with rolling updates.
8. Deploy an application on a Swarm cluster

Homework

upGrad

Voting App: https://github.com/dockersamples/example-voting-app

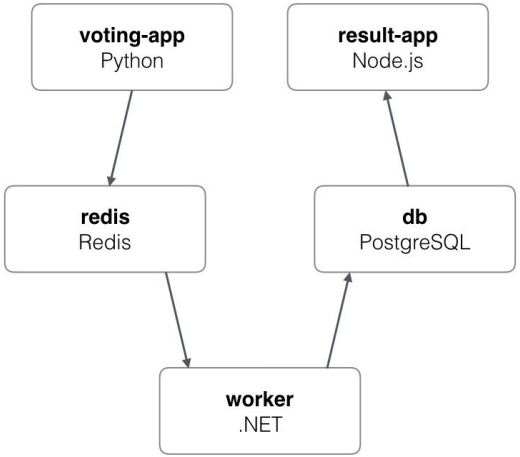
1. Set up the voting application using docker-compose. Deploy two replicas of voting frontend using dynamic port mapping. Add an Nginx service (with a customised configuration) file in compose file such that it load balances the request to two voting frontends.
2. Set up the voting application using docker swarm on a two-node cluster by using compose file as a stack file after making required changes. Create separate overlay networks for the frontend and backend. Attach five services of voting application wisely to these networks. Expose the ports of only the frontend services to the host. Set replica count = 5 for voting app and result app



Understand Voting Application

upGrad

Voting App: https://github.com/dockersamples/example-voting-app



1. **Voting-app :** Front-end for casting the vote

   Image: dockersamples/examplevotingapp_vote:before

   Port to be exposed: 80

2. **Redis:** Db to which voting-app push the votes data

   Image: redis:alpine

3. **Worker:** Checks Redis for anything in the queue and push it to PostgreSQL

   Image: image: dockersamples/examplevotingapp_worker

4. **Db:** Postgresql database

   Image: postgres:9.4 ; environment: POSTGRES_USER: "postgres" , POSTGRES_PASSWORD: "postgres"

5. **result-app :** frontend where one can see the result

   Image: dockersamples/examplevotingapp_result:before ; Port to be exposed: 80

upGrad

#RahoAmbitious

Thank You!