upGrad
#LifeKoKaroLift

Nginx, Scaling and Load Balancing

A Production Architecture Approach

upGrad

Course: Containerisation at Scale

Lecture On: Nginx and Docker Swarm

Instructor: Dipesh Garg

upGrad

Prerequisite for this Session

Basic understanding of Docker

Basic understanding of Docker Swarm

Poll 1 (15 seconds)

What is the behaviour expected when a service is created with this command in a swarm cluster containing five nodes?

```
docker service create --publish 8080:80  --replicas=3 nginx
```

A. Only those nodes which have these containers will listen on port 8080 and will forward the requests to port 80 of any container inside it.

B. All the nodes will listen on port 80 and will forward the requests to port 8080 of the service.

C. All the nodes will listen on port 80 and will forward the requests to port 8080 of any one of the containers in the cluster.

D. All the nodes will listen on port 8080 and will forward the requests to port 80 of any one of the containers in the cluster.

Poll 1 (15 seconds)

What is the behaviour expected when a service is created with this command in a swarm cluster containing five nodes?

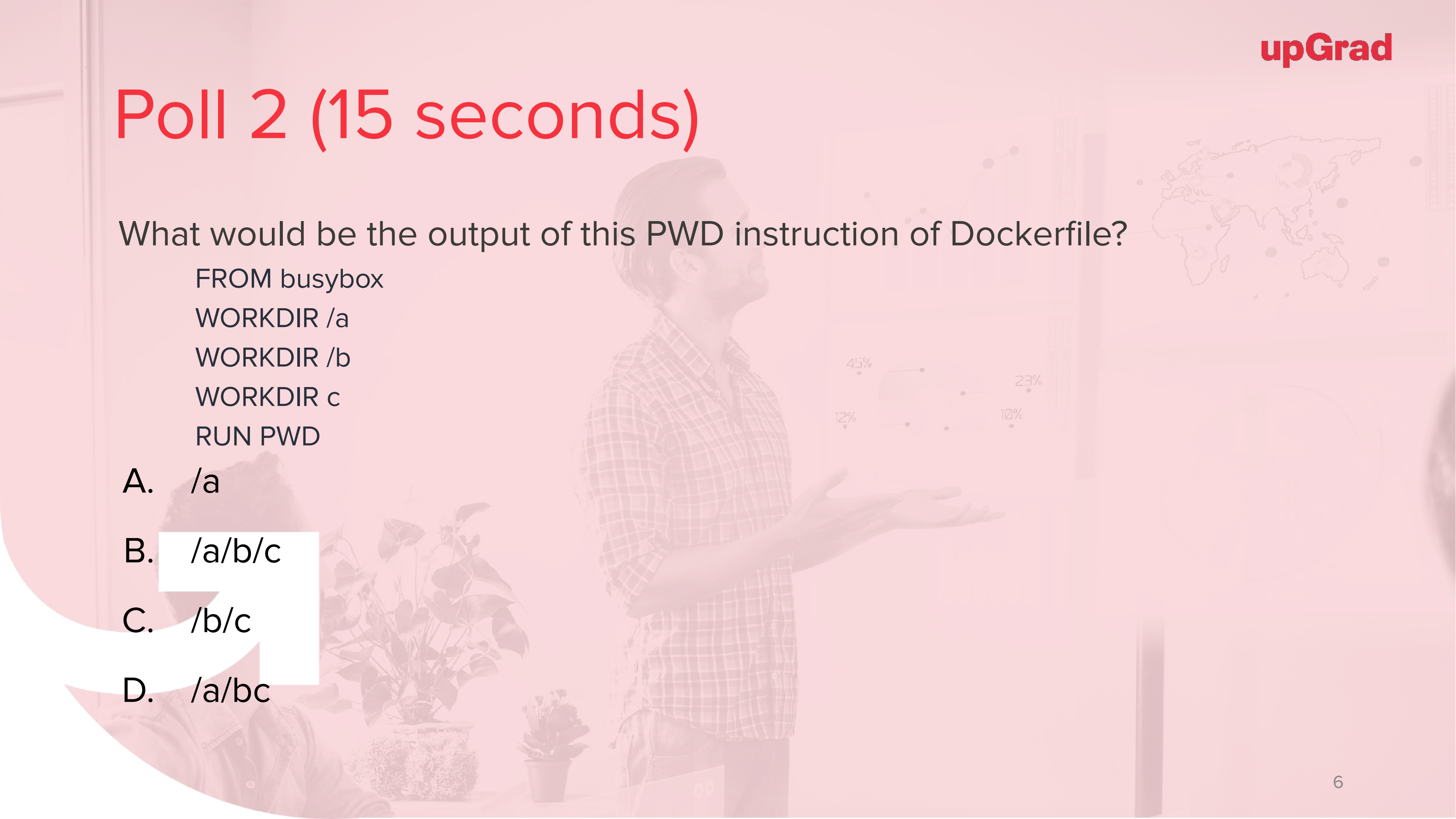
```
docker service create --publish 8080:80  --replicas=3 nginx
```

A. Only those nodes which have these containers will listen on port 8080 and will forward the requests to port 80 of any container inside it.

B. All the nodes will listen on port 80 and will forward the requests to port 8080 of the service.

C. All the nodes will listen on port 80 and will forward the requests to port 8080 of any one of the containers in the cluster.

D. All the nodes will listen on port 8080 and will forward the requests to port 80 of any one of the containers in the cluster.



Poll 2 (15 seconds)

What would be the output of this PWD instruction of Dockerfile?

```
FROM busybox
WORKDIR /a
WORKDIR /b
WORKDIR c
RUN PWD
```

A. /a

B. /a/b/c

C. /b/c

D. /a/bc

upGrad

Poll 2 (15 seconds)

What would be the output of this PWD instruction of Dockerfile?

```
FROM busybox
WORKDIR /a
WORKDIR /b
WORKDIR c
RUN PWD
```

A. /a

B. /a/b/c

C. /b/c

D. /a/bc

Today’s Agenda

- Nginx: Reverse proxy versus proxy, Nginx folder structure, virtual site hosting
- Scaling and load balancing
- Understand why, when and how to scale an application
- Horizontal scaling versus vertical scaling
- Load balancing types
- Schedule services based on different strategies and constraints
- Deploy an application on a Swarm cluster

Nginx

Proxy Server

What is Proxy?

1. In computer networking, a proxy server is a server application or appliance that acts as an intermediary for requests from clients seeking resources from servers that provide those resources.

2. A proxy server thus functions on behalf of the client when requesting service, potentially masking the true origin of the request to the resource server.

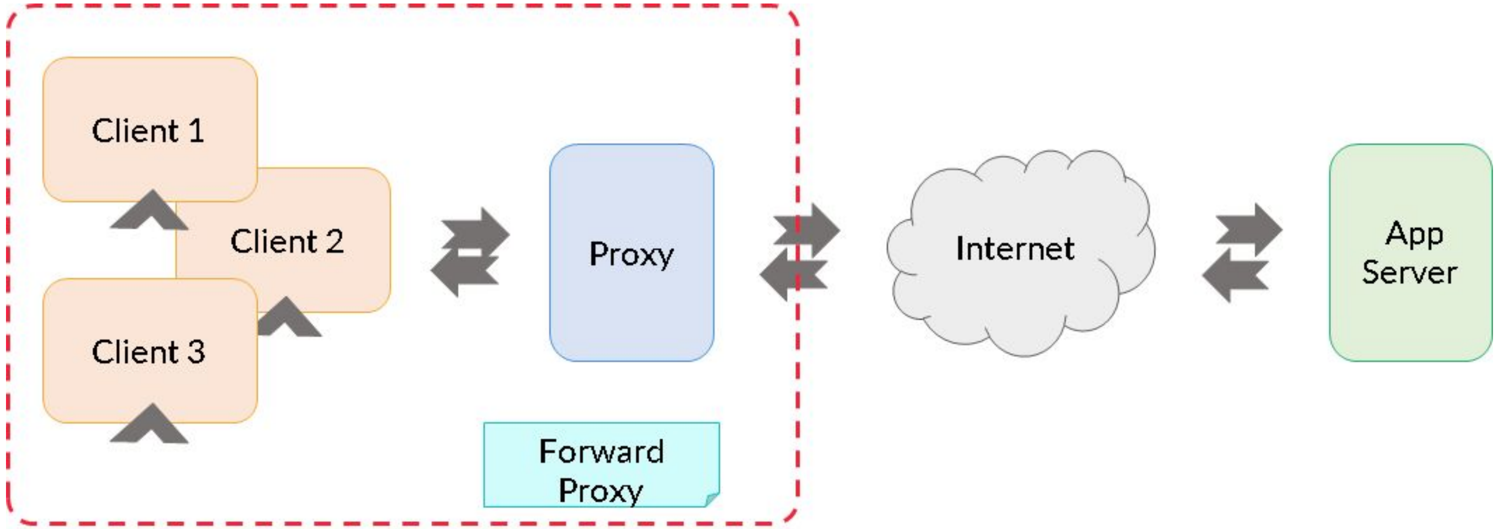
Types of Proxy

1. Forward proxy (or just proxy)
2. Reverse proxy

Proxy : Forward Proxy

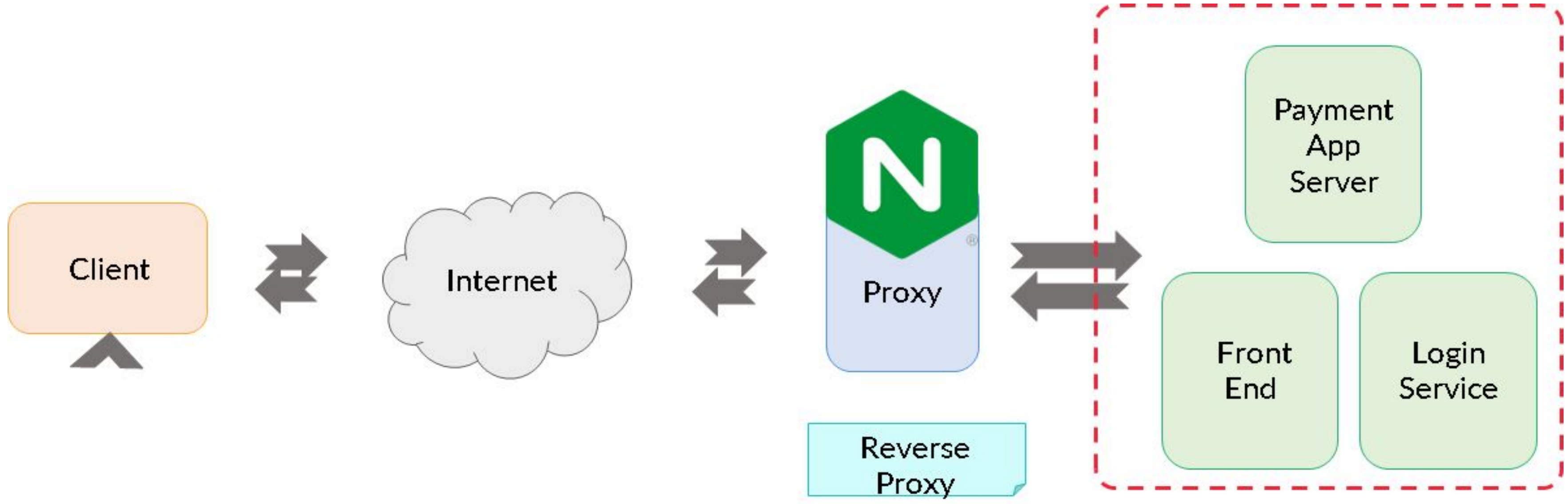
- A forward proxy provides proxy services to a client or a group of clients. Oftentimes, these clients belong to a common internal network

- This server reveals its identity as a proxy server, but does not disclose the originating IP address of the client to the resource server.



Proxy : Reverse Proxy

- A reverse proxy is a proxy server that appears to clients to be an ordinary server. Reverse proxies forward requests to one or more ordinary servers that handle the request.



Proxy vs Reverse Proxy - Use Cases

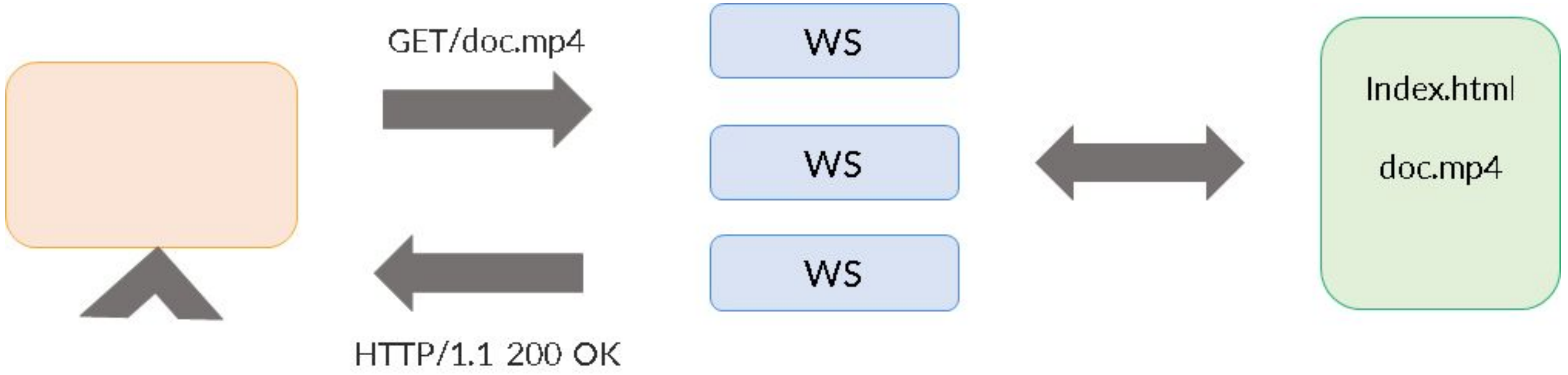
upGrad

- Forward proxy
  - Anonymises the client’s IP
  - Prevents data loss
  - Blocks user from visiting certain websites
  - Improves the user experience by caching external site content

- Reverse proxy
  - Hide the existence of the original back-end servers
  - Load balancing
  - Protect app servers from web based DOS attacks
  - Can provide caching functionality
  - Can optimise content by compressing
  - Combine different websites into a single URL space

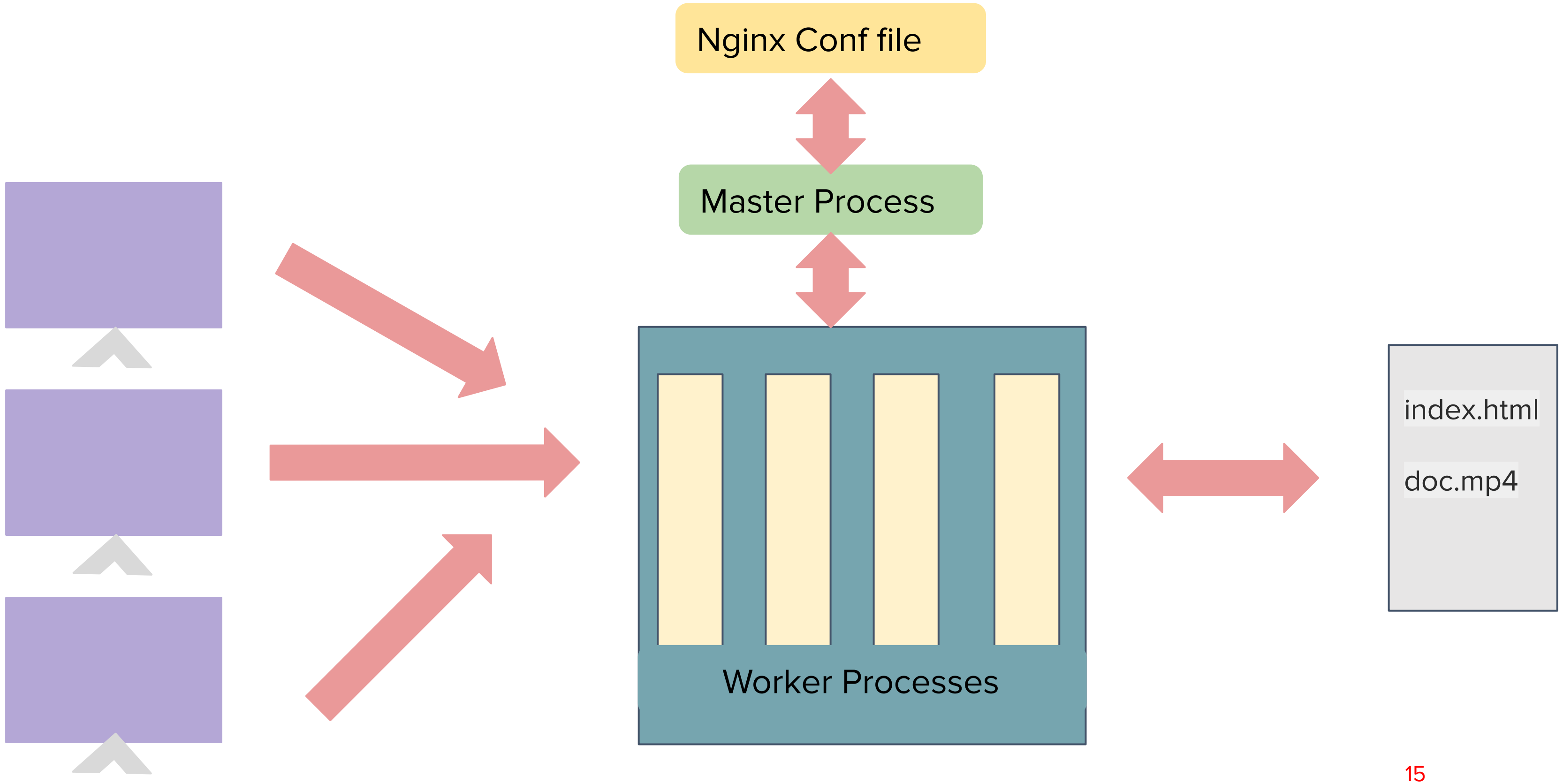
Introduction to Nginx

- Nginx is one of the most popular web servers in the industry.
- A web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve users the files that form web pages, in response to their requests.



Nginx Architecture

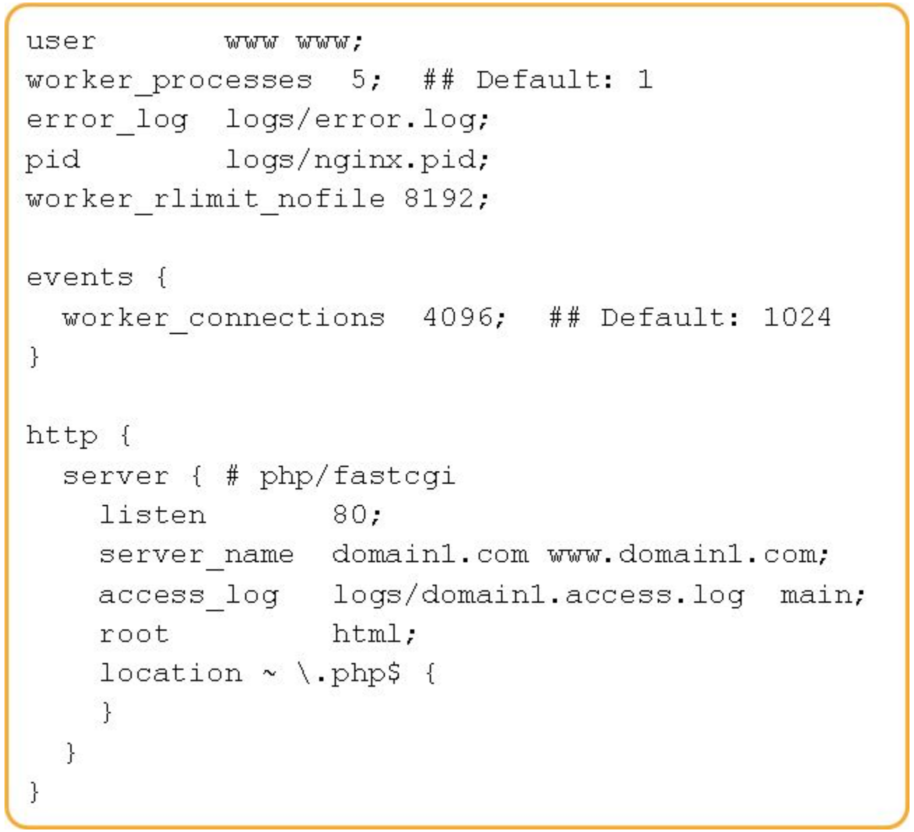
upGrad



DEMO 1 : NGINX

- Let’s take a look at the Nginx folder structure
- Server blocks and custom conf files
- **Host two websites:** Hello upGrad Learners and DevOps is Super Cool
- Serve these websites using upgradlearner.com and devopslearner.com

Nginx configuration file

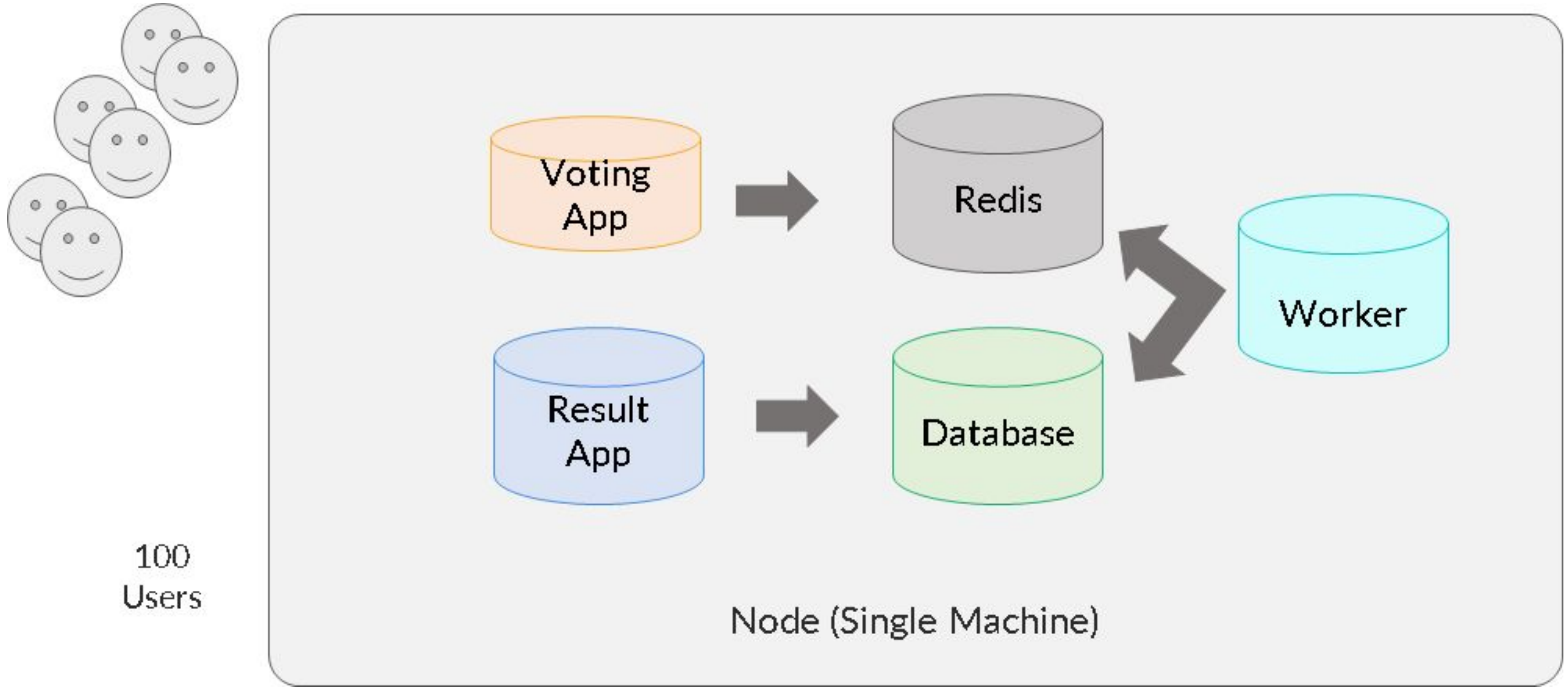


```
user          www www;
worker_processes  5;  ## Default: 1
error_log   logs/error.log;
pid         logs/nginx.pid;
worker_rlimit_nofile 8192;

events {
  worker_connections  4096;  ## Default: 1024
}

http {
  server { # php/fastcgi
    listen        80;
    server_name   domain1.com www.domain1.com;
    access_log    logs/domain1.access.log  main;
    root          html;
    location ~ \.php$ {
    }
  }
}
```

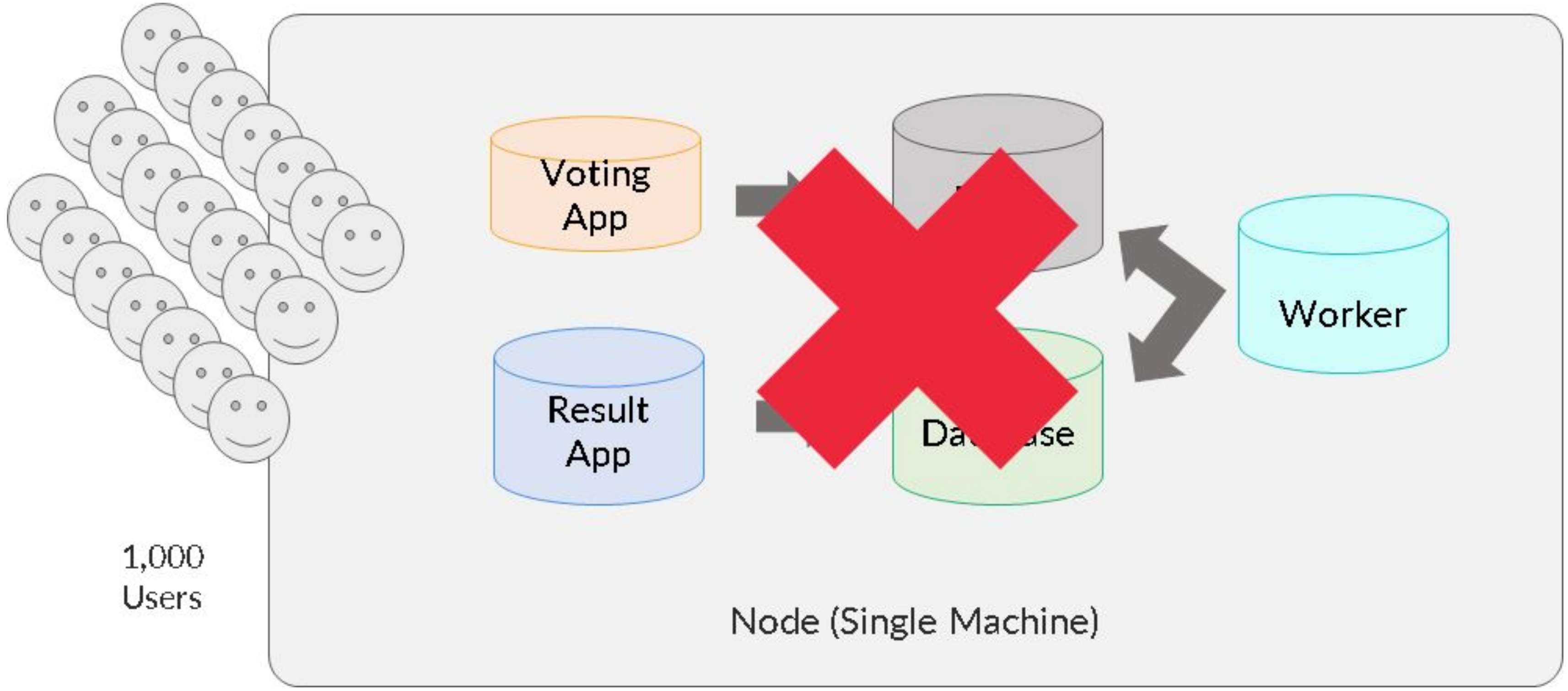
Scaling and Load Balancing

Scaling

upGrad



Scaling


Voting App
Result App
Database
Worker
1,000 Users
Node (Single Machine)

Scaling

- Scalability is the ability to easily increase or decrease compute or storage resources as needed to meet changing demand in cost-effective way.
- Ideally if your app takes t seconds to respond to a user request. It should take the same t seconds to respond to each of the million concurrent user requests on your app
- Cloud has simplified all these scaling problems dramatically.
- Remember, scaling is important for all parts of the application. Ignoring any part can result in downtime.

Scaling Strategies

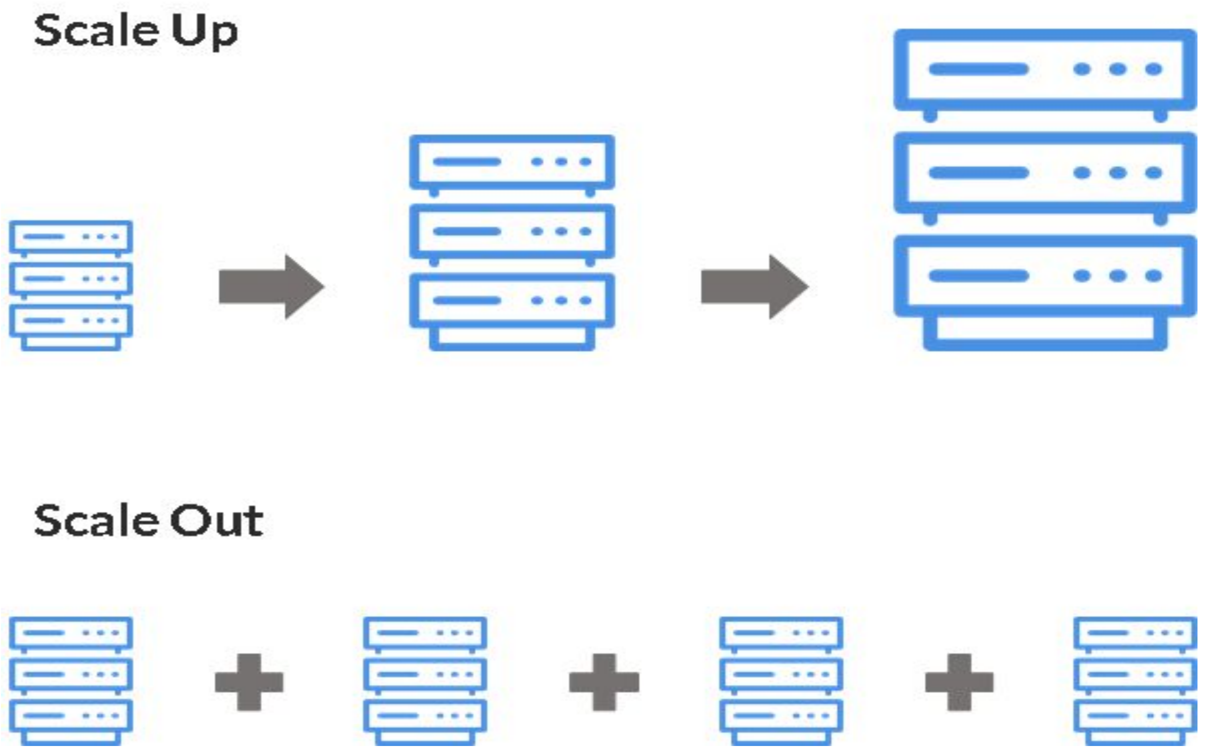
upGrad

- Vertical scaling
  - Adding resources in existing servers
  - Replacing a server with a bigger/more powerful server
  - Example: Changing t2.small to t2.large in AWS

- Horizontal scaling
  - Provision of additional servers to meet the need
  - Execute concurrent workload and handle traffic efficiently
  - Example: Adding 2 more nodes in a cluster of 3 nodes

Scaling Strategies

upGrad



Scaling : Use Cases

upGrad

Vertical Versus Horizontal

- Vertical
  - When data is on a single machine and partitioning is complicated.
  - The resources perform some cron job operations and duplicacy can lead to errors.

- Horizontal
  - It can provide instant and continuous availability.
  - There is no limit to hardware capacity.
  - Applicable when having multiple machines does not cause data duplicacy errors.
  - You want to run only the required number of hardware and choose not to pay for full capacity every time.

Scaling : Pros and Cons

- **Vertical**
  - **Less administrative efforts as you need to manage just one system.**
  - **This involves a downtime.**
  - **Your hardware always runs with full capacity.**
  - **Finite scope of upgradeability in the future.**

- **Horizontal**
  - **No downtime. Dynamically increase or decrease the resources.**
  - **You can choose to run only specific number of hardwares.**
  - **Easier to run fault-tolerance.**
  - **Infinite scope of scaling.**
  - **Architectural design can become highly complicated.**

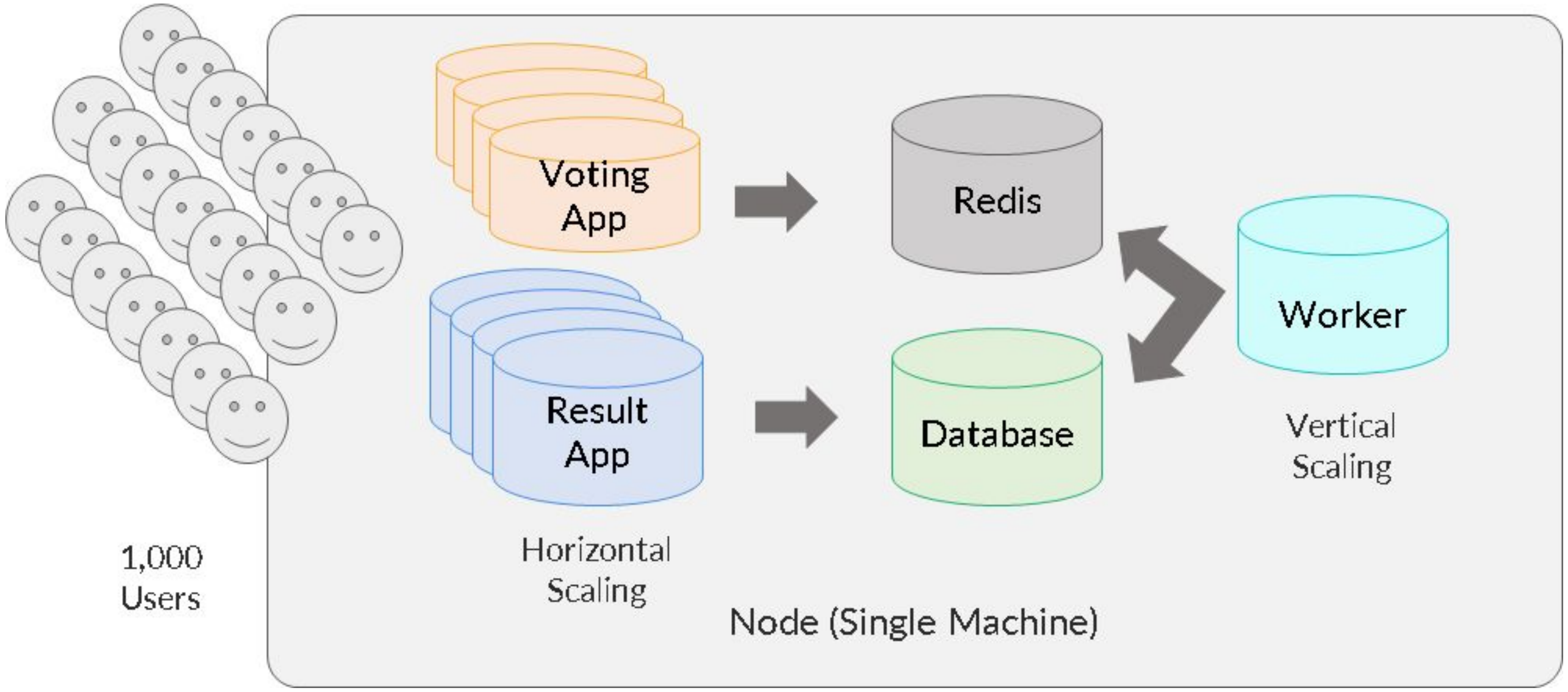
upGrad

DEMO 2 : Scaling

- Let’s see how we can scale application in AWS environment
- Vertical scaling in EC2 & RDS
- Horizontal scaling using ASG (auto scaling groups)

Scaling - What do We Achieve?

upGrad



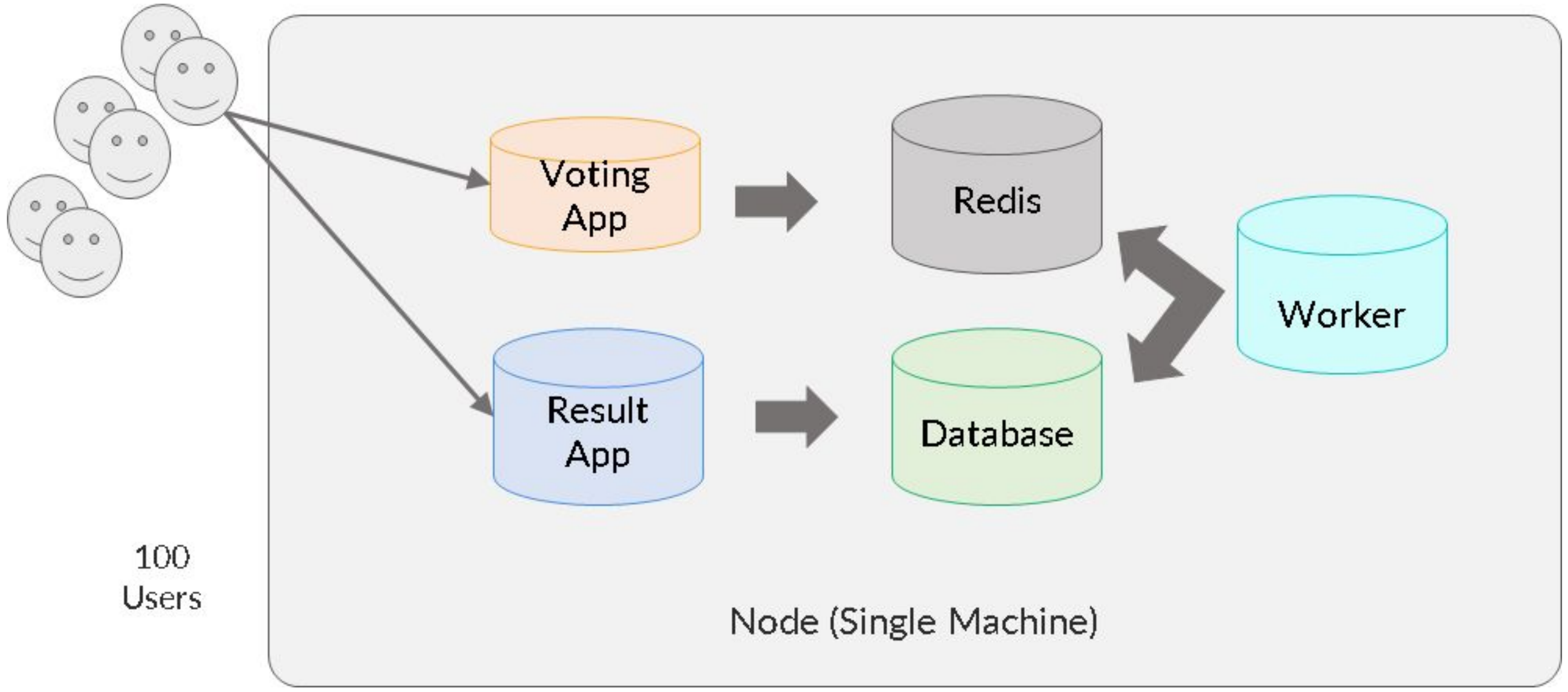
Ways to Scale

upGrad

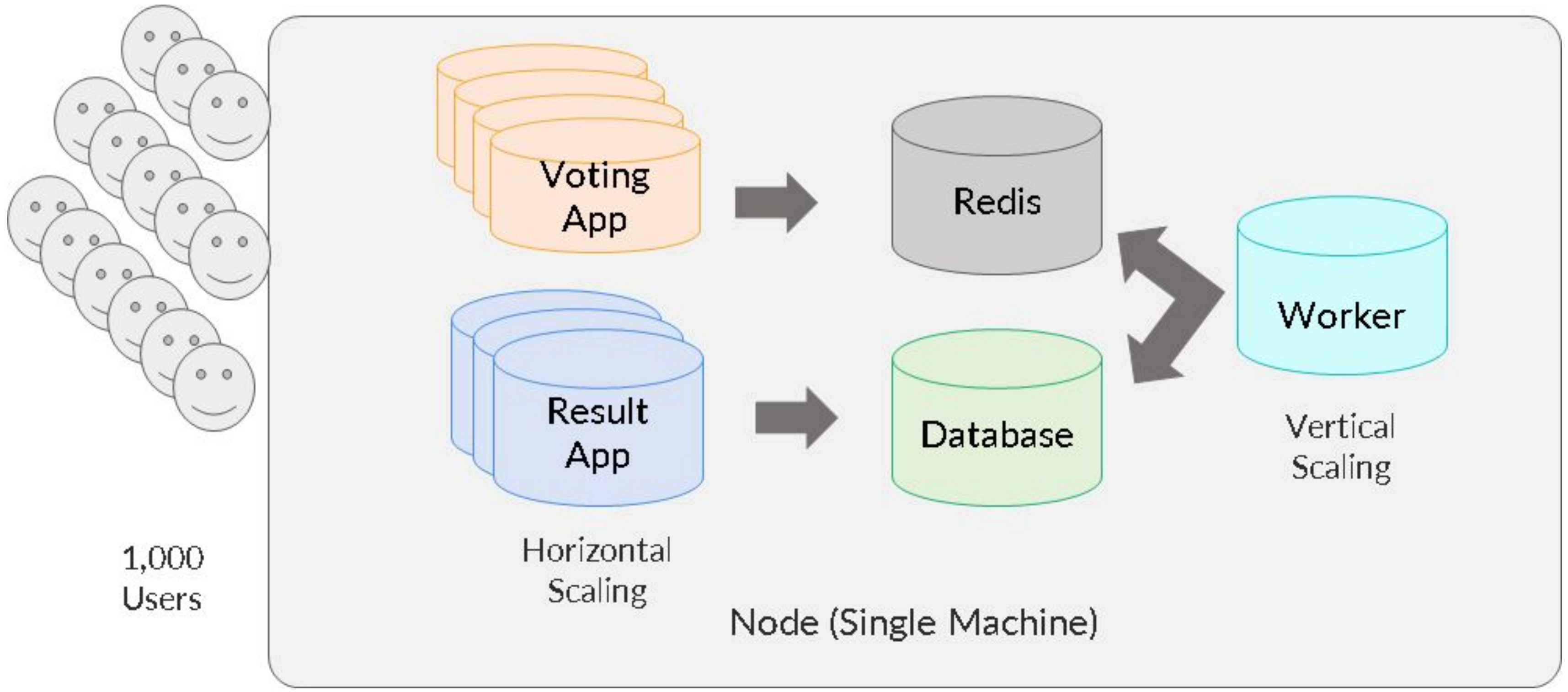
- Manual
  - Requires manual efforts for any scaling operation
  - Not ideal for sudden fluctuations
  - Can lead to human errors
- Scheduled
  - For predictable workloads
  - Effective for non-production system or predictable production traffic
  - Example: Shut down of non production environment in night hours
  - Cost-effective
- Automatic
  - Based on predefined rules and thresholds
  - Makes an application always available
  - Must be configured in dynamic traffic applications

Load Balancing

upGrad

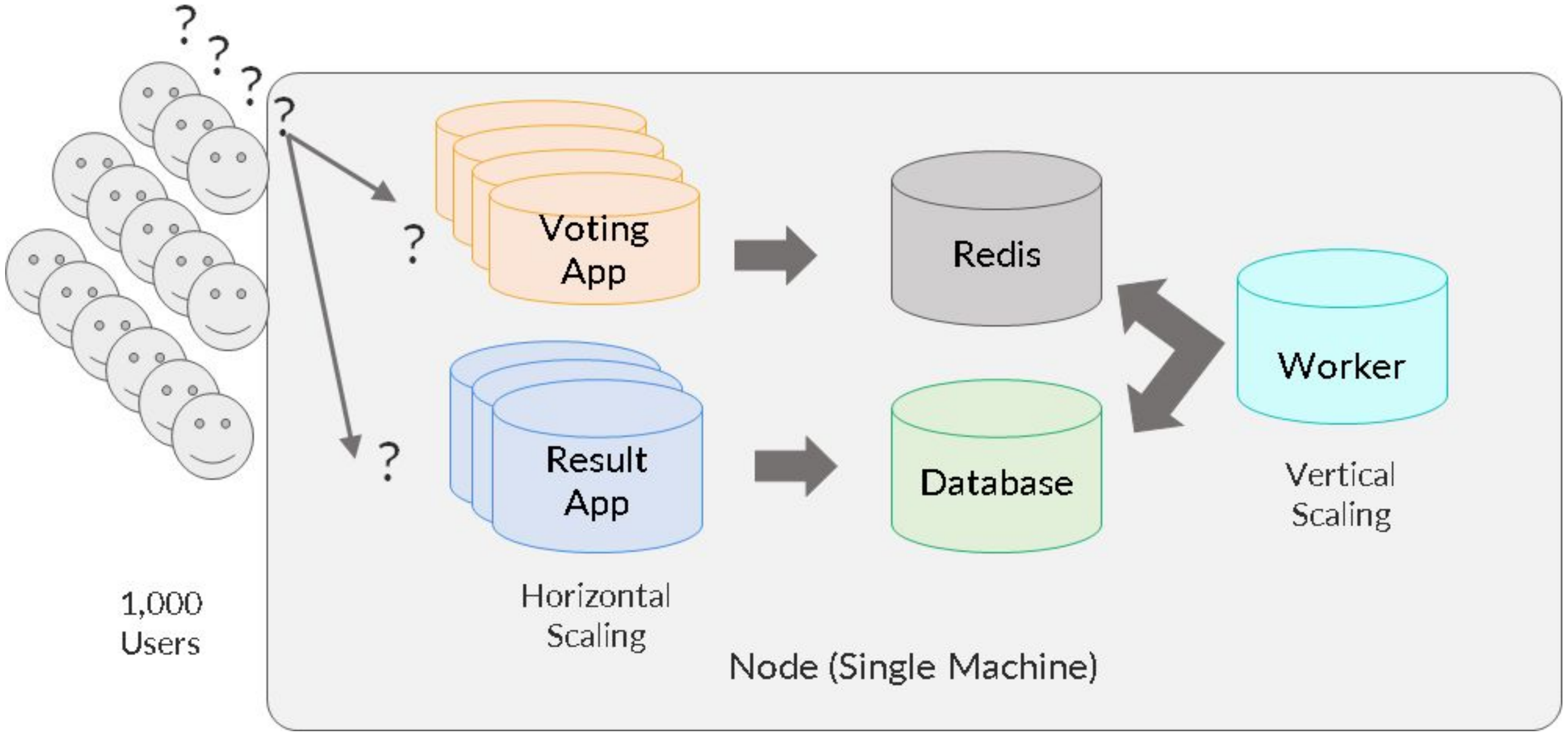


Load Balancing



Load Balancing

upGrad



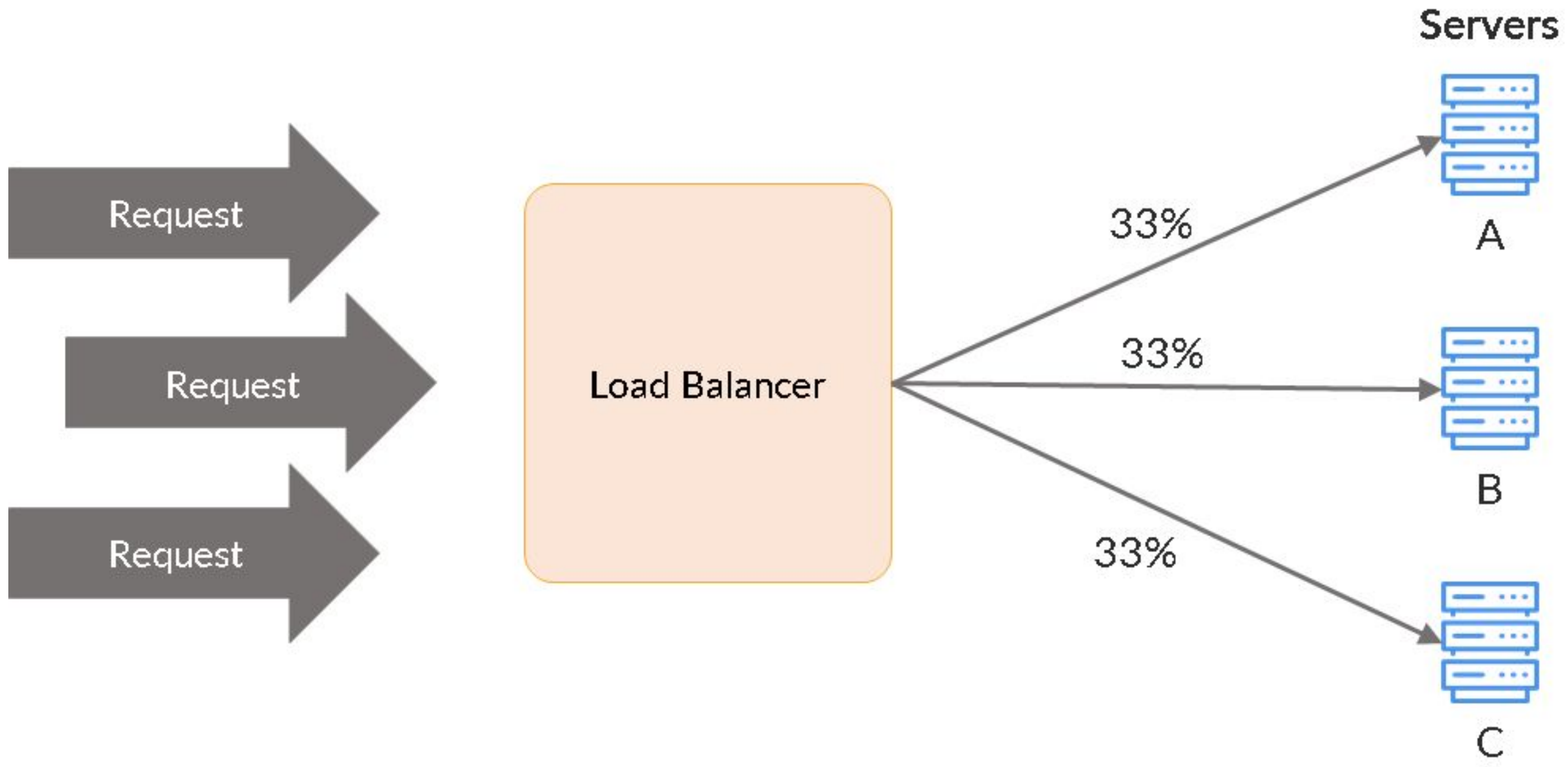
Load Balancing (LB)

upGrad

- Load balancing refers to efficiently distributing application requests across all the healthy back-end servers.
- It is the key component of highly available infrastructure.
- It provides the flexibility to add or remove servers as per the demand.
- The user accesses the LB, which forwards the user's request to a back-end server.
- It saves us from launching any application server in a public subnet. Only the LB needs to be launched in a public subnet.
- It ensures effective use of all the available servers.
- It saves cost.

Load Balancing (LB)

upGrad



Load Balancing Types

upGrad

- **Round robin** - This is the most common type and is widely used. Requests are distributed sequentially [one by one to each server].
- **Least connections** - The server with the fewest connections is awarded the new request.
- **Least time** - It depends on the fastest response time and the fewest active connections.
- **IP hash** - This is used for sticky sessions. The IP address of the client is used to determine which server receives the request.

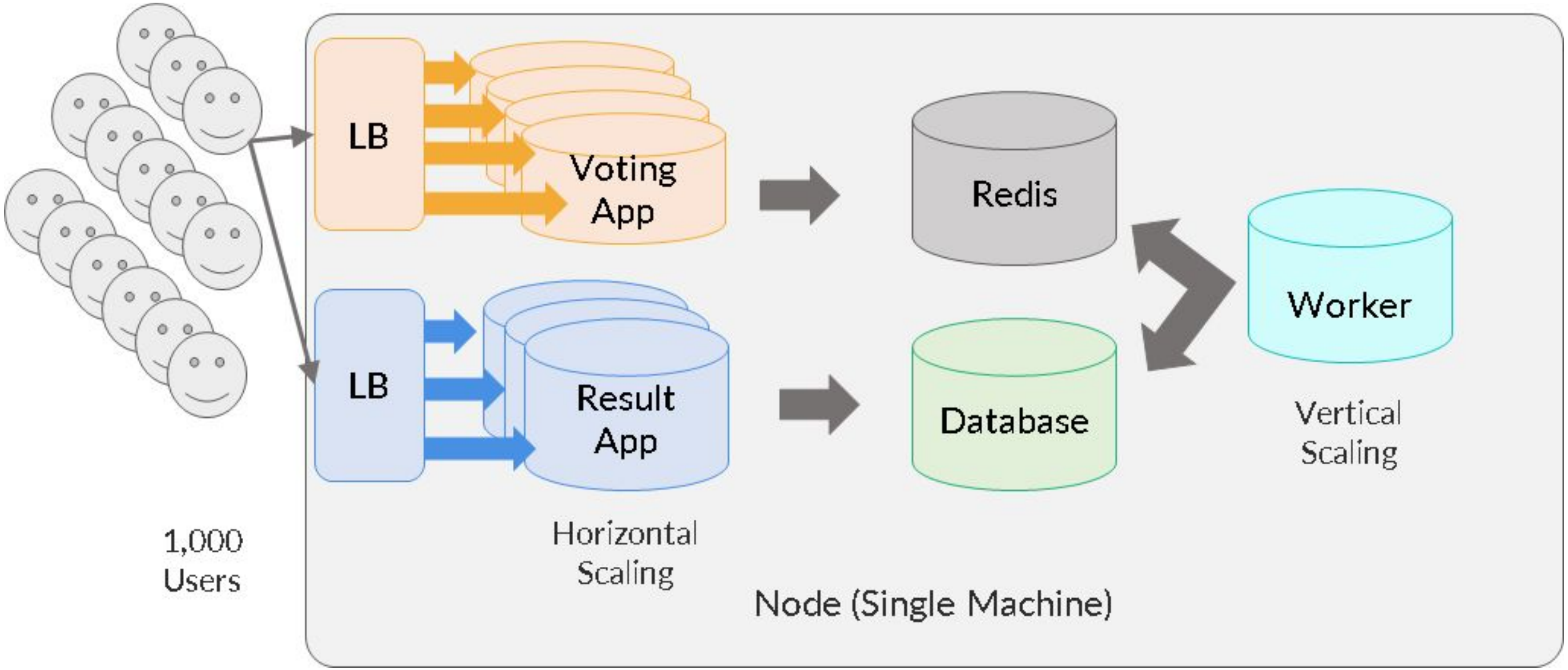
upGrad

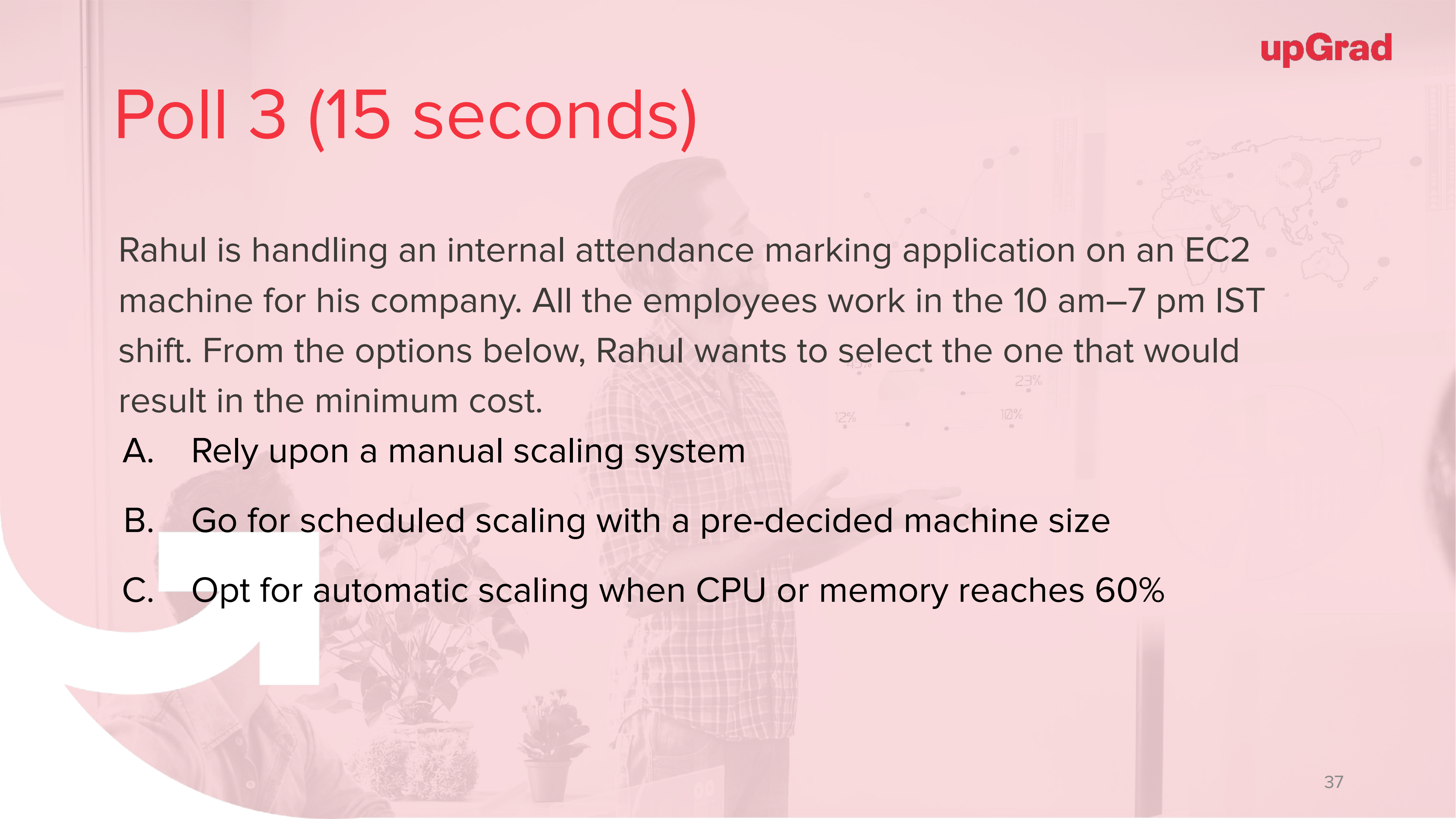
DEMO 3 : Load Balancing

- Use of Nginx for load balancing
- Implementing load balancing types using Nginx
- Load balancing in AWS environment

Load Balancing - What do We Achieve?

upGrad





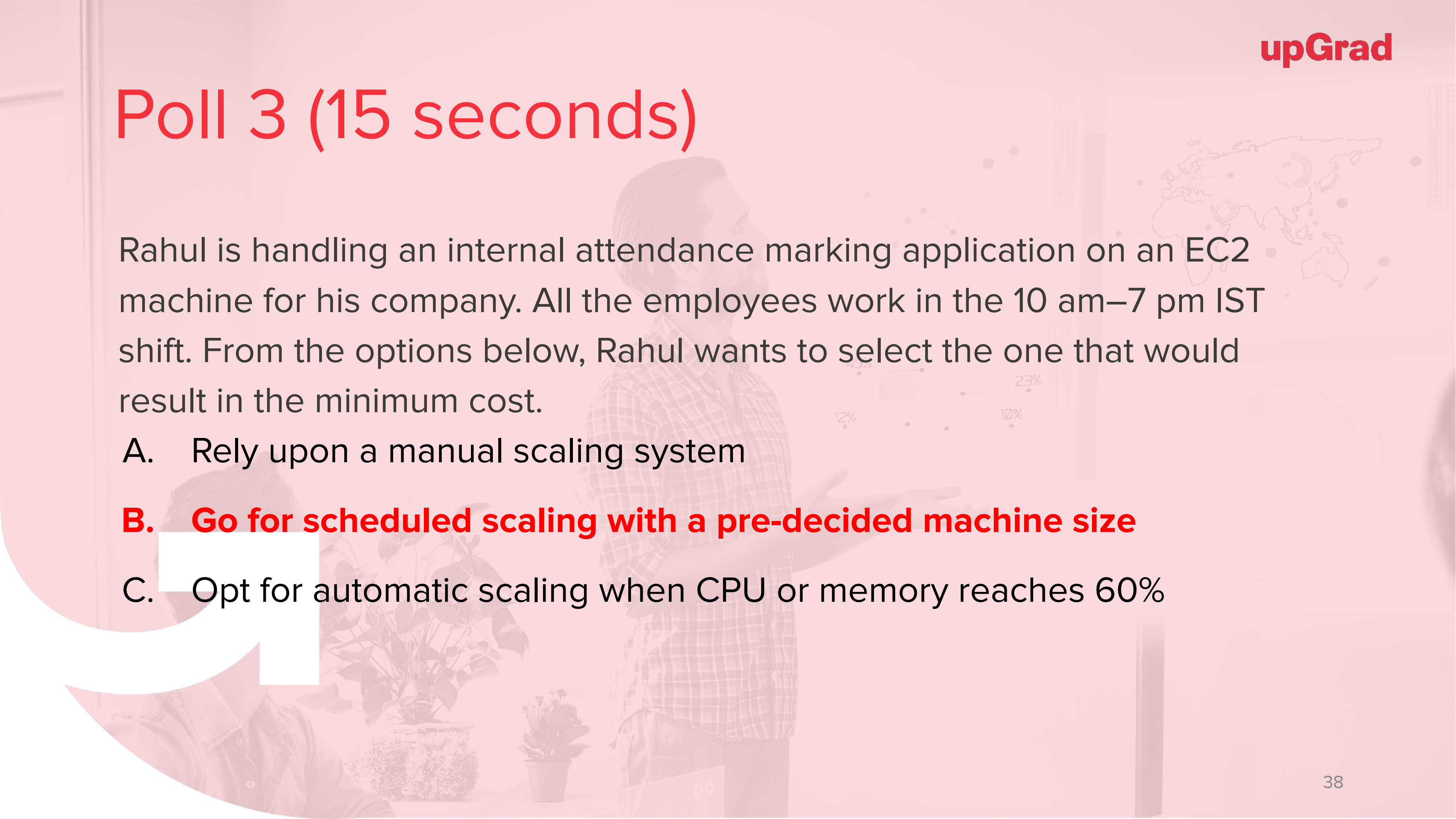
Poll 3 (15 seconds)

Rahul is handling an internal attendance marking application on an EC2 machine for his company. All the employees work in the 10 am–7 pm IST shift. From the options below, Rahul wants to select the one that would result in the minimum cost.

A. Rely upon a manual scaling system

B. Go for scheduled scaling with a pre-decided machine size

C. Opt for automatic scaling when CPU or memory reaches 60%



Poll 3 (15 seconds)

Rahul is handling an internal attendance marking application on an EC2 machine for his company. All the employees work in the 10 am–7 pm IST shift. From the options below, Rahul wants to select the one that would result in the minimum cost.

A. Rely upon a manual scaling system

B. Go for scheduled scaling with a pre-decided machine size

C. Opt for automatic scaling when CPU or memory reaches 60%

Poll 4 (15 seconds)

Jack sets up five back-end services with Nginx for load balancing. Many users in different locations reported latency related issues. Jack checked the status of the system and found that all the systems were underutilised. What he can try in order to resolve the issue?

A. Add two more machines to the cluster

B. Use the IP hash approach for load balancing

C. Use the least connections approach for load balancing

D. Ask the users to check their Internet bandwidth

Poll 4 (15 seconds)

Jack sets up five back-end services with Nginx for load balancing. Many users in different locations reported latency related issues. Jack checked the status of the system and found that all the systems were underutilised. What he can try in order to resolve the issue?

A. Add two more machines to the cluster

B. Use the IP hash approach for load balancing

C. Use the least connections approach for load balancing

D. Ask the users to check their Internet bandwidth

upGrad

Advanced Features of Docker Swarm

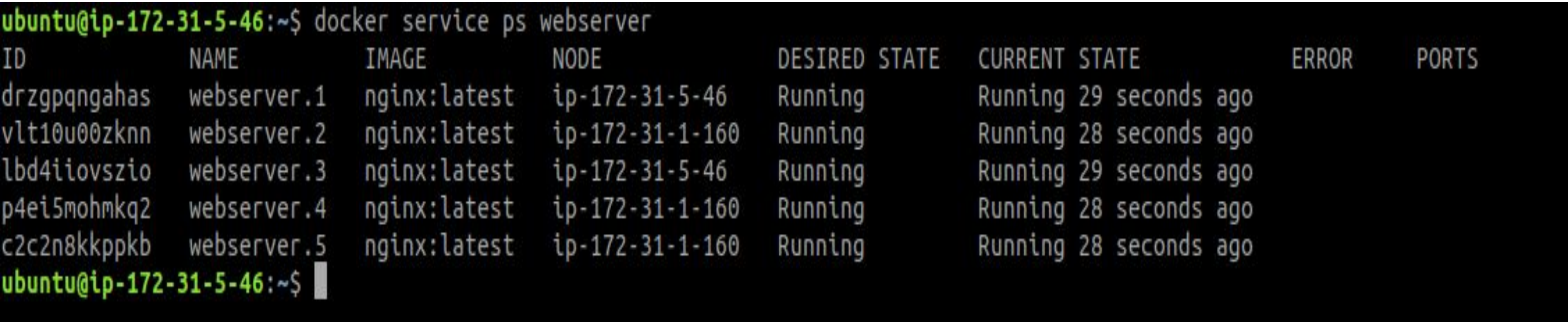
Node Draining

- Prevents a node from receiving new tasks
- Will stop existing tasks running on that node and launch replicas of those tasks on other nodes
- Helps with patching servers and debugging issues

```
docker node update --availability drain worker1

docker node update --availability active worker1
```

Node Draining

- docker service create --name webserver --replicas 5 nginx
- docker service ps webserver
  - See where these 5 tasks have been placed
- Now assume we want to take instance 172.31.1.160 in maintenance mode

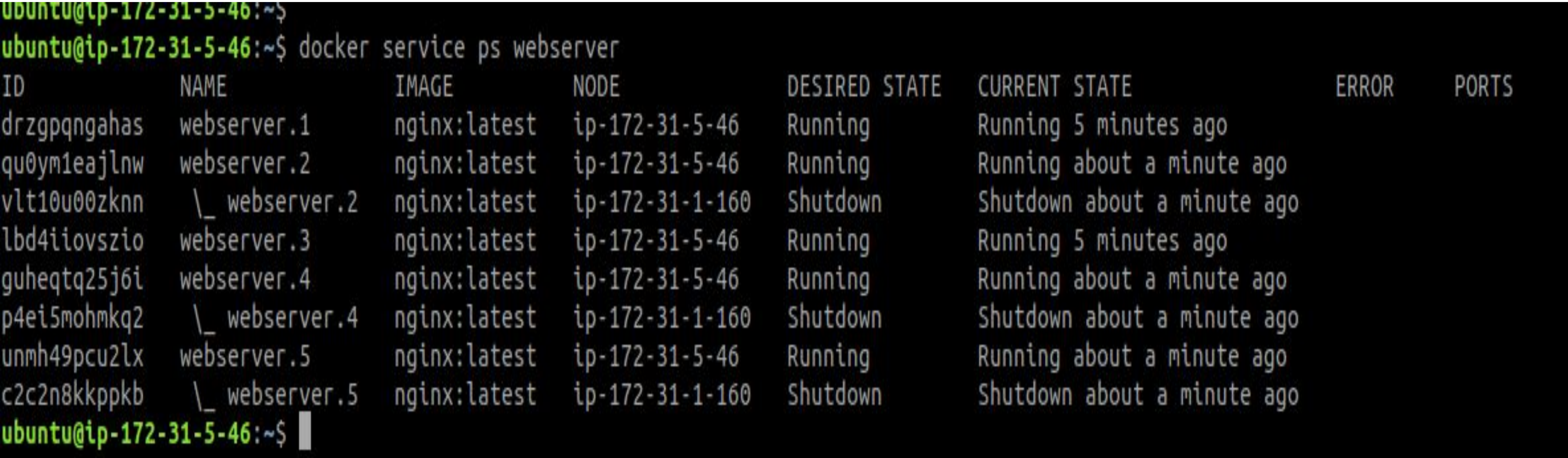


```
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID            NAME         IMAGE         NODE            DESIRED STATE   CURRENT STATE           ERROR     PORTS
drzgpqngahas  webserver.1  nginx:latest  ip-172-31-5-46   Running         Running 29 seconds ago
vlt10u00zknn  webserver.2  nginx:latest  ip-172-31-1-160  Running         Running 28 seconds ago
lbd4iiovszio  webserver.3  nginx:latest  ip-172-31-5-46   Running         Running 29 seconds ago
p4ei5mohmkq2  webserver.4  nginx:latest  ip-172-31-1-160  Running         Running 28 seconds ago
c2c2n8kkppkb  webserver.5  nginx:latest  ip-172-31-1-160  Running         Running 28 seconds ago
ubuntu@ip-172-31-5-46:~$
```

Node Draining

upGrad

- docker node update --availability drain ip-172-31-1-160
- docker service ps webserver
  - Now, See where these 5 tasks have been placed



```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID             NAME              IMAGE          NODE              DESIRED STATE   CURRENT STATE                   ERROR     PORTS
drzgpqngahas   webserver.1       nginx:latest   ip-172-31-5-46    Running         Running 5 minutes ago
qu0ym1eajlnw   webserver.2       nginx:latest   ip-172-31-5-46    Running         Running about a minute ago
vlt10u00zknn    \_ webserver.2   nginx:latest   ip-172-31-1-160   Shutdown        Shutdown about a minute ago
lbd4iiovszio   webserver.3       nginx:latest   ip-172-31-5-46    Running         Running 5 minutes ago
guheqtq25j6i   webserver.4       nginx:latest   ip-172-31-5-46    Running         Running about a minute ago
p4ei5mohmkq2    \_ webserver.4   nginx:latest   ip-172-31-1-160   Shutdown        Shutdown about a minute ago
unmh49pcu2lx   webserver.5       nginx:latest   ip-172-31-5-46    Running         Running about a minute ago
c2c2n8kkppkb    \_ webserver.5   nginx:latest   ip-172-31-1-160   Shutdown        Shutdown about a minute ago
ubuntu@ip-172-31-5-46:~$
```

Inspect Swarm Services and Node

- "**Inspect**" provides detailed information about any service or node.
- By default, it will show the results in a JSON array.
- It helps with debugging an issue (ex- container not getting scheduled or a node not getting registered.)

❑ docker service inspect serviceName

❑ **docker service inspect --pretty serviceName**
(display the details about a service in an easily readable format.)

❑ docker node inspect <nodeId> --pretty

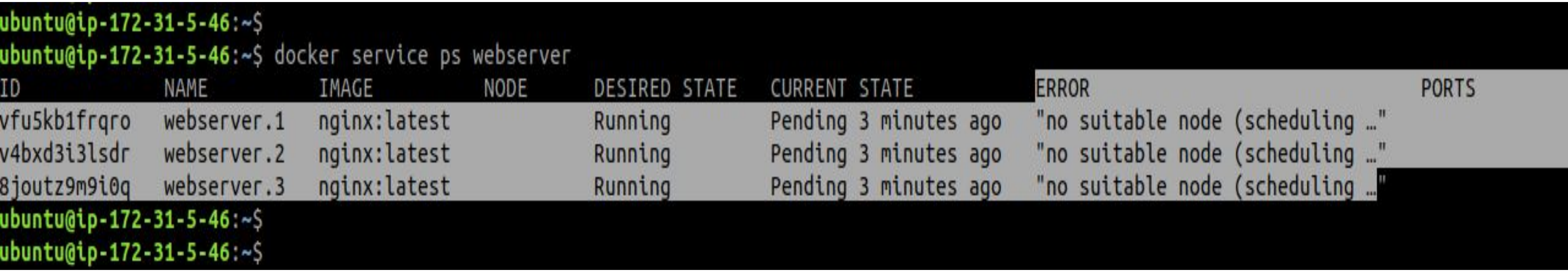
Placement Constraints

- The Swarm service provide a few different ways for you to control scale and placement of services on different nodes:
    - **Replicated** and **global** modes of services
    - **Resource constraints** like requirement of CPU/memory
    - **Placement constraints** (ex- Run container only on a node with label upgrad=true)
    - **Placement preferences** (ex- spread containers evenly across groups of nodes where grouping is based on a label’s value)

Placement Constraints

upGrad

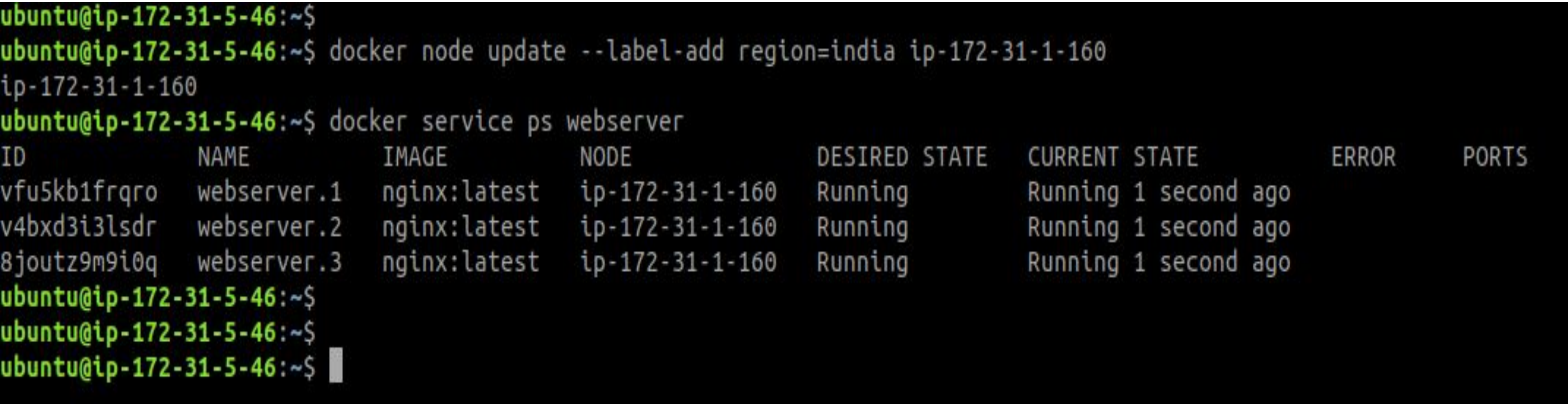
- Let's place a service using placement constraint
  - Constraint : Node should have label => region==india
  - docker service create --name webserver --constraint node.labels.region==india --replicas 3 nginx
  - If no node fulfil this constraint, the containers will go into pending state
  - See the current status using, docker service ps webserver



```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID            NAME         IMAGE         NODE      DESIRED STATE   CURRENT STATE          ERROR                             PORTS
vfu5kb1frqro  webserver.1  nginx:latest            Running         Pending 3 minutes ago  "no suitable node (scheduling …"
v4bxd3i3lsdr  webserver.2  nginx:latest            Running         Pending 3 minutes ago  "no suitable node (scheduling …"
8joutz9m9i0q  webserver.3  nginx:latest            Running         Pending 3 minutes ago  "no suitable node (scheduling …"
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

Placement Constraints

- Set node label to any one node using
  - docker node update --label-add region=india <NODE ID>
- Now, see current status of containers
- All containers will be on same single node to which we applied the label



```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker node update --label-add region=india ip-172-31-1-160
ip-172-31-1-160
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID            NAME          IMAGE         NODE             DESIRED STATE   CURRENT STATE          ERROR      PORTS
vfu5kb1frqro  webserver.1   nginx:latest  ip-172-31-1-160  Running         Running 1 second ago
v4bxd3i3lsdr  webserver.2   nginx:latest  ip-172-31-1-160  Running         Running 1 second ago
8joutz9m9i0q  webserver.3   nginx:latest  ip-172-31-1-160  Running         Running 1 second ago
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

Master Project Set-Up

upGrad

- Let us set up the voting application with all the feature implementations that we have discussed till now:
  - Set up voting application on a three-node cluster
  - Scale the services
  - Apply node constraints on the key value and the resource basis
  - Set up Nginx on the server

upGrad

Demo 4 :

Docker Swarm Advanced Features

- Node draining
- Placement constraints ad placement preferences
- Load balancing in AWS environment

upGrad

Demo 5 :

Docker Voting Application

upGrad

Important Concepts and Questions

Important Questions

upGrad

1. What are the differences between a forward proxy and a reverse proxy?
2. How can you host multiple websites using Nginx?
3. What is scaling? Explain the different strategies and the types of scaling.
4. Explain the different load balancing techniques. Write a practical use case of each.
5. What is the use of draining?
6. Can you explain the different placement constraints present in Docker Swarm?

upGrad

Doubt Clearance Window

Session 1.1 covered these topics:

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

This class covered the following topics:

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 versus vertical scaling
5. Load balancing types
6. Schedule services based on different strategies and constraints
7. Deploy an application on a Swarm cluster

What’s Next??

1. What is ECS, ECS Components
2. Concepts of ECR, EC2 and Fargate; task definition; and services
3. ECS pricing
4. Launch Getting started Cluster, understand things, update services and task definition
5. Launch your own Fargate cluster. Create your own TD and services
6. Logging and monitoring
7. Update services
8. ECS load balancing with ALB
9. Deploy containers in ECS Fargate mode

Homework

upGrad

1. Create a docker file such that when a docker image is build using this file, and a container is run using that image, it should run the Nginx web server inside it and replace the welcome page text with container IP. Now launch three replicas of the same container by docker run command. Launch a new Nginx container and change its configuration file such that it proxy pass the traffic to the other three containers. Try different load balancing types in the configuration file.

2. Host a site yourname.com. Create an index page and a fail-safe page. If a page corresponding to the URI is not available, then the fail-safe page is served.

3. Set up the voting application using Docker Swarm:
   a. Set up the application on a three-node cluster
   b. Scale the voting and result services to 5 replicas
   c. Apply node constraints on the key value and the resource basis
   d. Now add Nginx container in the stack so that voting and result frontend can be accessed using votingapp.com & resultapp.com [Hint: you also have to add two DNS entries in /etc/hosts file in your local system mapped to ec2 IP.
4. Read about the difference between a placement constraint and a placement preference.

Tasks to Complete After Today’s Session

MCQs
Practice Questions
Practice Project

upGrad
#RahoAmbitious

Thank You!