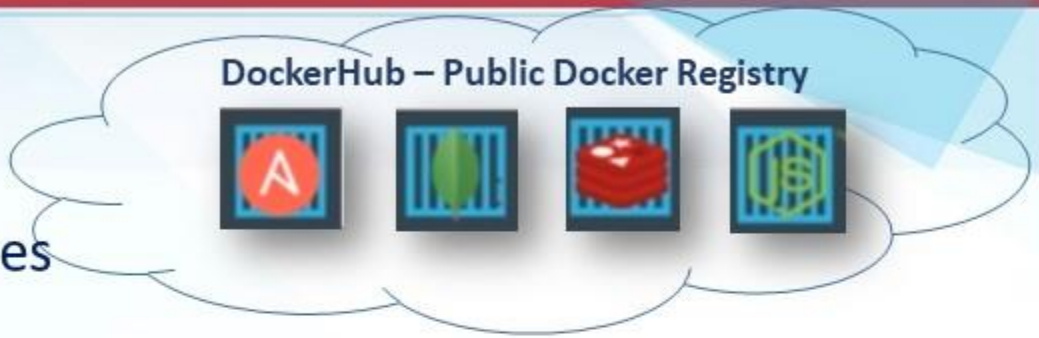
SkillAssure

# Docker Compose

SkillAssure

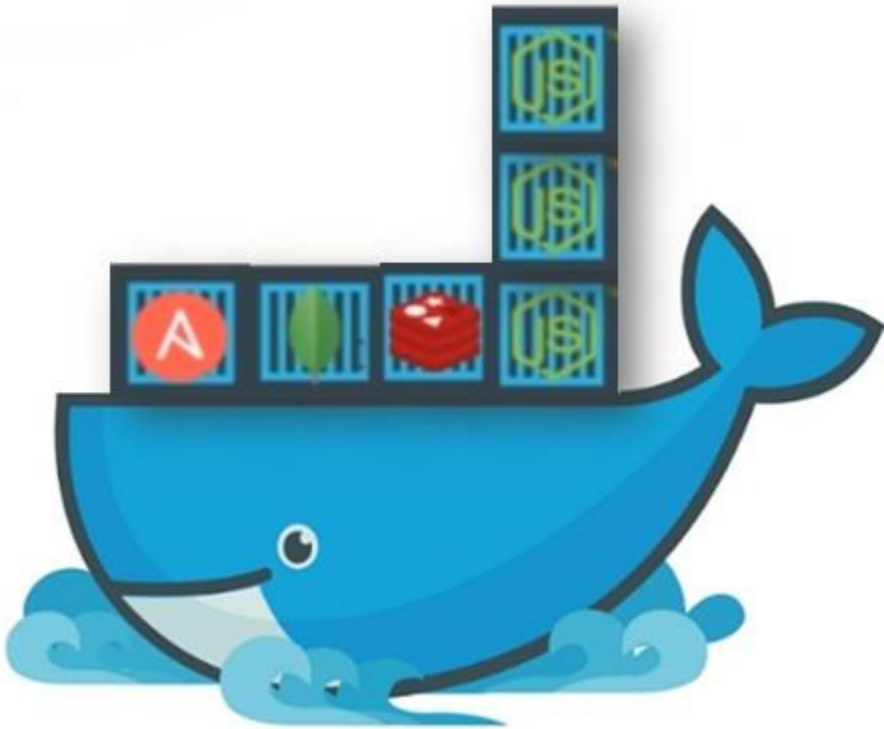
# Docker Compose




- What is a better way to run a complex application that involves multiple services.

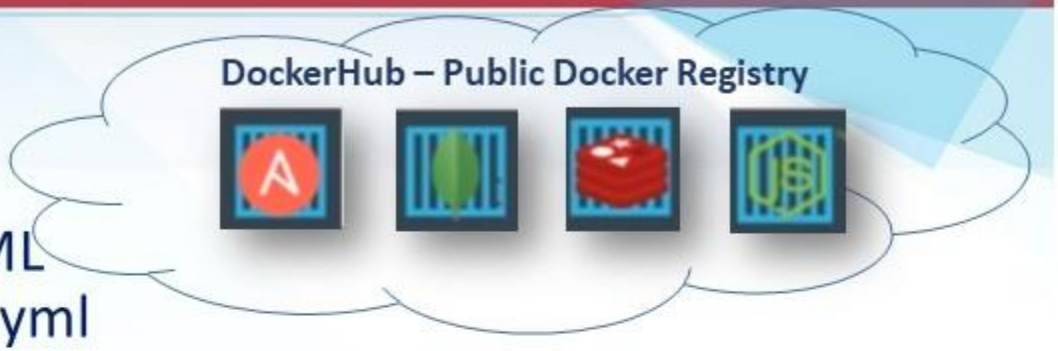
```
docker run prabhav/simple-webapp
docker run mongodb
docker run redis
docker run ansible
```

- **Docker Compose**



SkillAssure

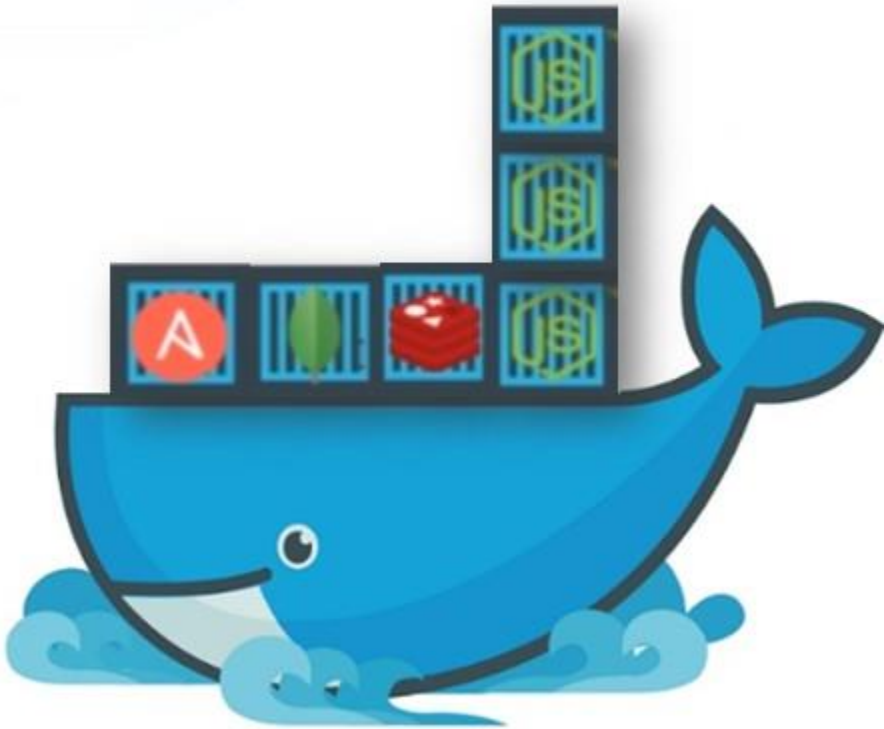
# Docker Compose




- With Docker compose we can create a configuration file in YML format called Docker-compose.yml
- We can specify all services and options for running docker containers

docker-compose.yml

```
services:
  web:
    image: "prabhav/simple-webapp"
  database:
    image: "mongodb"
  messaging:
    image: "redis"
  configmgmt:
    image: "ansible"
```



SkillAssure

# Docker Compose

- To bring up the entire application stack, we need to run

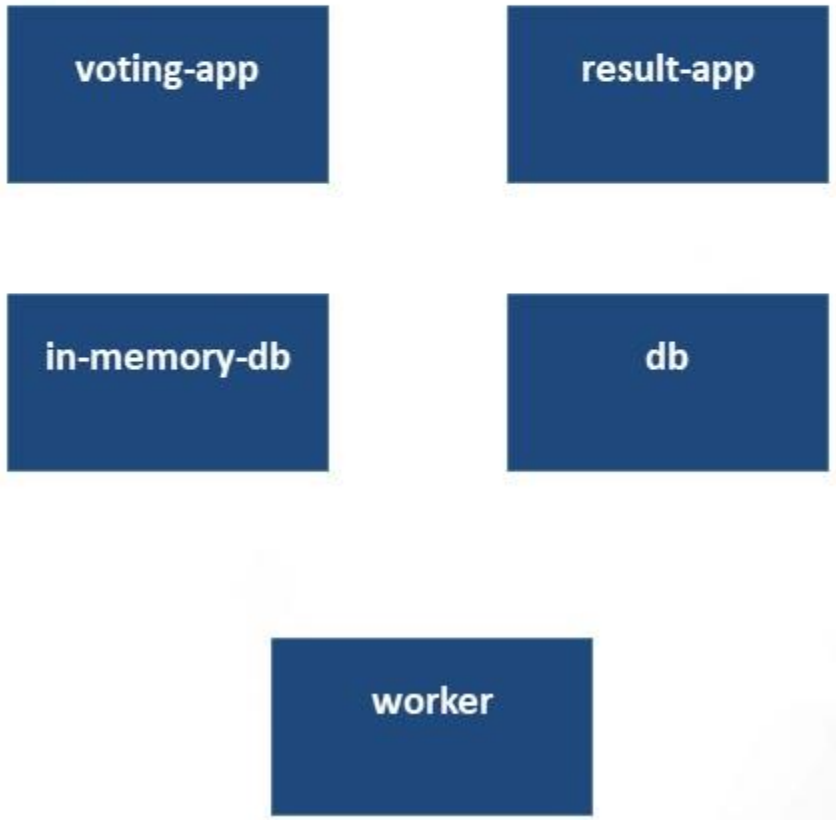
  **docker compose up**

- This is easier to implement, run and maintain.
- All changes are stored in the docker-compose file and can even be placed in SCM Tool.
- Only applicable to run containers on a single Docker Host.

SkillAssure

# Sample Application – Voting App

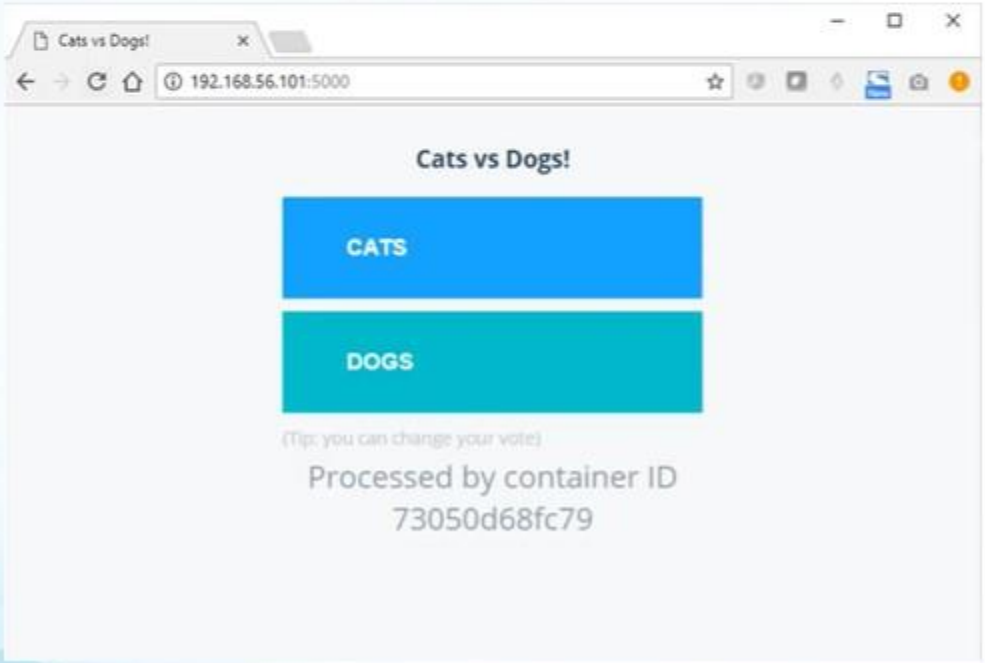
- It's a simple application developed by Docker to demonstrate the various features available in running an application stack on Docker
- This is a sample voting application which provides an interface for a user to vote and another interface to show the results.
- The application consists of various components



SkillAssure

# Sample Application – Voting App

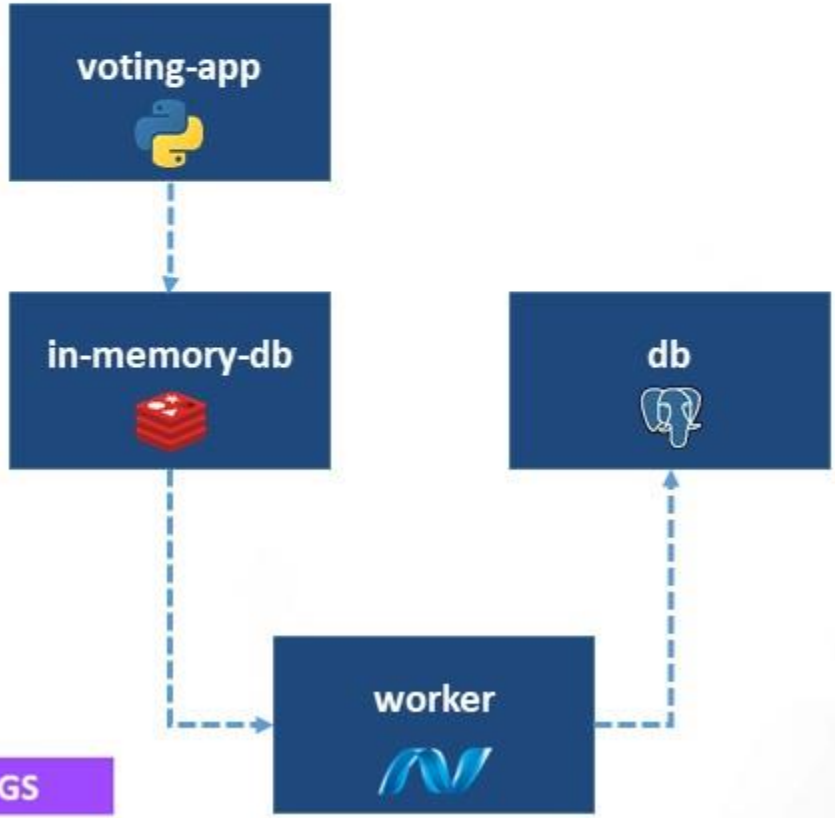
- Voting app which is a web application developed in Python.
- It provides the user with an interface to choose between two options, i.e. a cat and a dog.

SkillAssure

# Sample Application – Voting App

- When you make a selection the vote is stored in **redis**
- Redis serves as in memory database
- This vote is then processed by the worker which is an application written in .net
- The worker application takes the new vote and updates the persistent database which is a PostgreSQL
- The PostgresSQL simply has a table with the number of votes for each category i.e. cats and a dogs

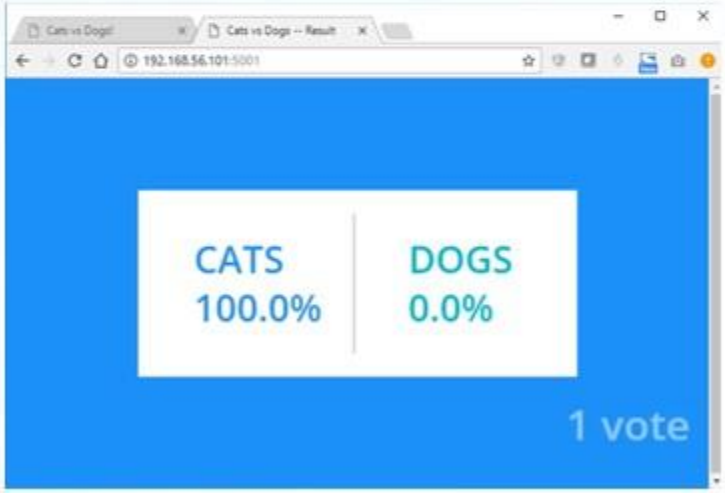


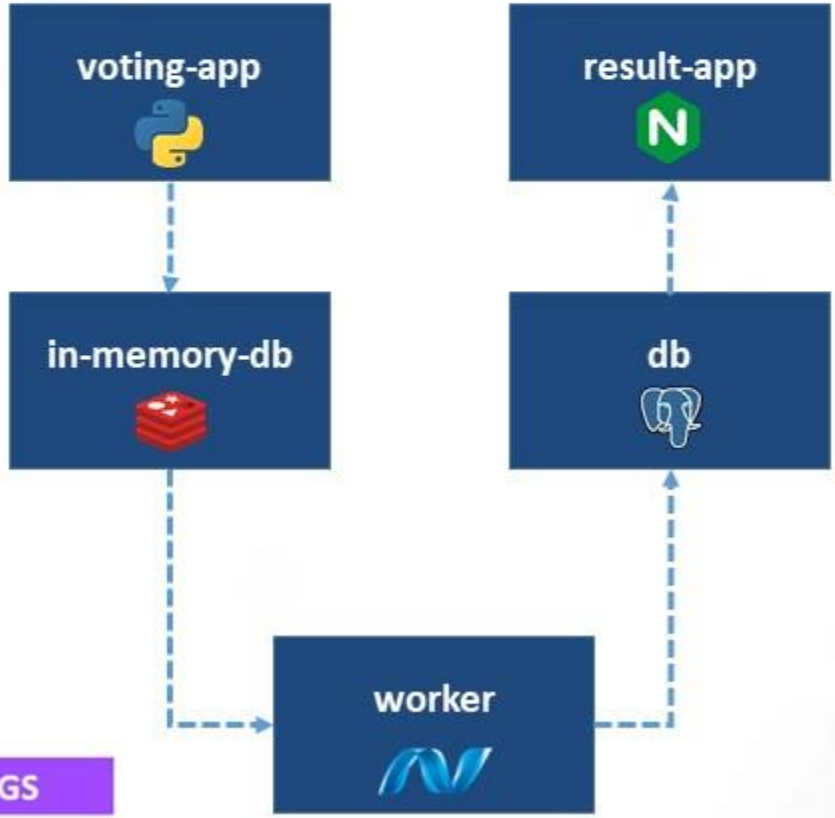

| CATS | DOGS |
|---|---|
| 1 | 0 |

SkillAssure

# Sample Application – Voting App

- The result of the vote is displayed in a web interface which is another web application developed in nodeJS
- This resulting application reads the count of votes from the PostgreSQL database and displays it to the user




| CATS | DOGS |
|---|---|
| 1 | 0 |

SkillAssure

# Sample Application – Voting App

- **Setup Data Layer**
- Run a redis container with latest image in detach mode and name it “redis”

```
docker run -d --name=redis redis
```

- Run a postgres container with postgres:9.4 image in detach mode and name it “db”

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trustpostgres:9.4
```

SkillAssure

# Sample Application – Voting App

- **Setup Application Layer**
- Run the voting app container in detach mode and name it as “vote”. Also map host port 5000 to container port 80

```
docker run -d --name=vote -p 5000:80 vote-app
```

- Run the result app container in detach mode and name it as “result”. Also map host port 5001 to container port 80

```
docker run -d --name=result -p 5001:80 result-app
```

- **Setup Worker**
- Run the worker app container in detach mode and name it “worker”

```
docker run -d --name=worker worker-app
```

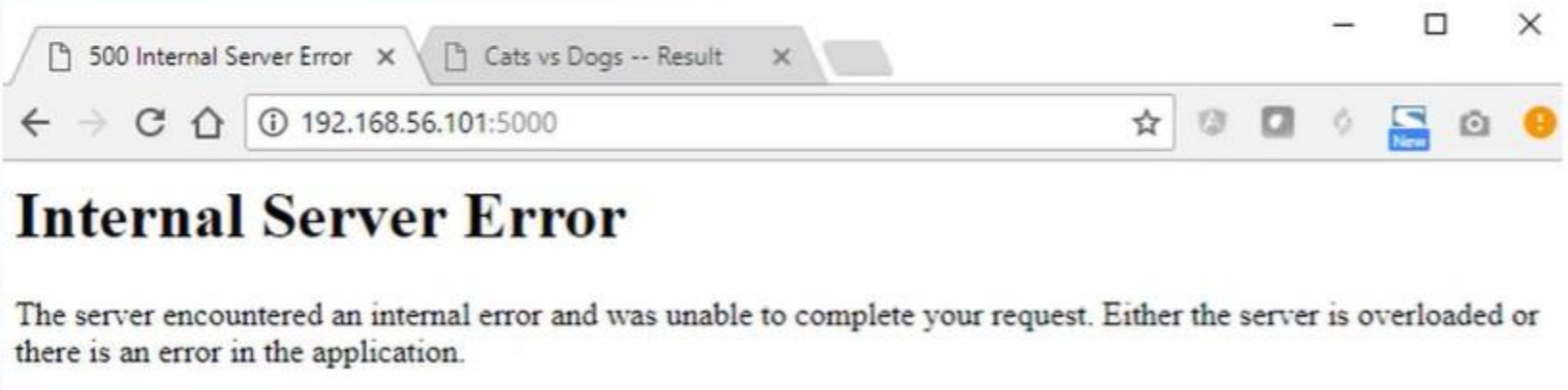
SkillAssure

# Sample Application – Voting App

- List all running containers

```
docker ps
```

- Try opening the voting app.
- You will not that there is some problem. The application doesn't work and gives internal server error



SkillAssure

# Sample Application – Voting App

- We have successfully run all the containers, then what is the problem?
- We haven't linked containers together.
- e.g. We haven't told the voting web application to use this particular redis instance.

- **Link** is a command line option which can be used to link two containers together

- The voting app web service is dependent on the redis service.
- When the webserver starts, it looks for a “redis” service running on host “redis”
- But the voting app container cannot resolve a host by the name “redis”.

```
def get_redis():
    if not hasattr(g, 'redis'):
        g.redis = Redis(host="redis", db=0, socket_timeout=5)
    return g.redis
```

SkillAssure

# Sample Application – Voting App

- To make the voting app aware of the “redis” service, we add a link option while running the voting app container

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
<Name-Of-Redis-Container>:<Name-Of-Redis-Host>
```

- Internally, it creates an entry into the /etc/hosts file on the voting app container adding an entry with the hostname redis with an internal IP of the redis container

```
/app # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      redis 89cd8eb563da
172.17.0.3      ebcae9eb46bf
```

SkillAssure

# Sample Application – Voting App

- To make the voting app aware of the “redis” service, we add a link option while running the voting app container

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
<Name-Of-Redis-Container>:<Name-Of-Redis-Host>
```

- Internally, it creates an entry into the /etc/hosts file on the voting app container adding an entry with the hostname redis with an internal IP of the redis container

```
/app # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      redis 89cd8eb563da
172.17.0.3      ebcae9eb46bf
```

SkillAssure

# Sample Application – Voting App

- Add a link for the result app by the name "db" to communicate with the database.

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

- Add links for the worker app to communicate with both redis & database.

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Note: Using links this way is deprecated. Advanced and newer concepts in Docker swarm and networking supports better ways of achieving this.

SkillAssure

# Sample Application – Voting App

- Let's now look at all docker run commands.

```
docker run -d --name=redis redis
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Test if the application is works fine.

SkillAssure

# Docker compose

- Docker run commands

```
docker run -d --name=redis redis
```

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
```

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- We will refer these commands to generate our docker compose file.

- Let's start simple!
- Create a dictionary of container names. Use the same name as we used in the docker run commands

docker-compose.yml

SkillAssure

# Docker compose

- Docker run commands

```
docker run -d --name=redis redis
```

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
```

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Create a key and value under each item. The key is the image and the value is the name of the image

docker-compose.yml

```
redis:

db:

vote:




result:




worker:
```

SkillAssure

# Docker compose

- Docker run commands

```
docker run -d --name=redis redis
```

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
```

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Publish the ports for respective containers. Create a property called ports and list all the ports that you want to publish under that

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  image: vote-app


result:
  image: result-app


worker:
  image: worker-app
```

SkillAssure

# Docker compose

- Docker run commands

```
docker run -d --name=redis redis
```

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
```

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Link the containers. Whichever container requires the link, create a property under it called links and provide an array of links such as redis or db

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  image: vote-app
  ports:
    - 5000:80


result:
  image: result-app
  ports:
    - 5001:80


worker:
  image: worker-app
```

SkillAssure

# Docker compose

- Docker run commands

```
docker run -d --name=redis redis
```

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
```

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Finally, add the environment variable

docker-compose.yml

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

SkillAssure

# Docker compose

- Docker run commands

```
docker run -d --name=redis redis
```

```
docker run -d --name=db -e POSTGRES_HOST_AUTH_METHOD=trust postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis vote-app
```

```
docker run -d --name=result -p 5001:80 --link db:db --link redis:redis result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker-app
```

- Now our docker-compose file is ready. Let's bring the entire stack up which is very easy now.

```
docker-compose up
```

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
  environment:
    - POSTGRES_HOST_AUTH_METHOD=trust
vote:
  image: vote-app
  ports:
    - 5000:80
  links:
    - redis:redis
result:
  image: result-app
  ports:
    - 5001:80
  links:
    - redis:redis
    - db:db
worker:
  image: worker-app
  links:
    - redis:redis
    - db:db
```

SkillAssure

# Docker compose

- How to build docker images instead of pulling image from docker registry using docker compose?
- Replace the image line with a build line and specify the location of a directory which contains the application code and a dockerfile with instructions to build the Docker image.
- Now when we run the docker compose up command it will first build the images give a temporary name for it and then use those images to run containers using the specified options.

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
  environment:
    -
POSTGRES_HOST_AUTH_METHOD:
"trust"
vote:
  image: vote-app
  ports:
    - 5000:80
  links:
    - redis:redis
result:
  image: result-app
  ports:
    - 5001:80
  links:
    - redis:redis
    - db:db
worker:
  image: worker-app
  links:
    - redis:redis
    - db:db
```

docker-compose.yml

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

SkillAssure

# Exercise

- Fork the git repository to your account
- Modify the docker-compose file to build docker images while running docker-compose up command.

SkillAssure

# Docker compose - versions

- Docker compose evolved over time and have different versions.
- Version 1 a number of limitations
  - Deploying containers on a different network other than the default bridged network.
  - Specify dependency or start up order of some kind, e.g. database container must come up first and then the voting app

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  image: voting-app
  ports:
    - 5000:80
  links:
    - redis:redis
```

**version: 1**

SkillAssure

# Docker compose - versions

- In version 2, all stack information is encapsulated in the services section
- How does docker-compose know what version of the file you're using.
- You must specify the version of docker-compose file at the top of the file.
- Version 1 docker-compose attaches all the containers to the default bridged network and then use links to enable communication between the containers
- Version 2 docker-compose automatically creates a dedicated bridge network for this application and then attaches all containers to the new network
- All containers communicate to each other using each other's service name. So you basically don't need to use links in version 2 of docker-compose.

docker-compose.yml

```
version: "2"
services:
  redis:
    image: redis
  db:
    image: postgres:9.4
  vote:
    image: voting-app
    ports:
      - 5000:80
    links:
      - redis:redis
```

**version: 2**

SkillAssure

# Docker compose - versions

- Version 2 also introduces a depends on feature if you wish to specify a start up order.
- e.g. The voting web application is dependent on the redis service. So you need to ensure that redis container it started before the voting web application

docker-compose.yml

```
version: "2"
services:
  redis:
    image: redis
  db:
    image: postgres:9.4
  vote:
    image: voting-app
    ports:
      - 5000:80
    depends_on:
      - redis
```

**version: 2**

SkillAssure

# Exercise

- Convert the docker-compose file to version 2
  - Vote container should start after redis
  - Worker should start after redis & db

SkillAssure

# Docker compose - versions

- Version 3 is the latest as of today
- Version 3 is similar to version 2 in the structure
- It comes with support for Docker Swarm

docker-compose.yml

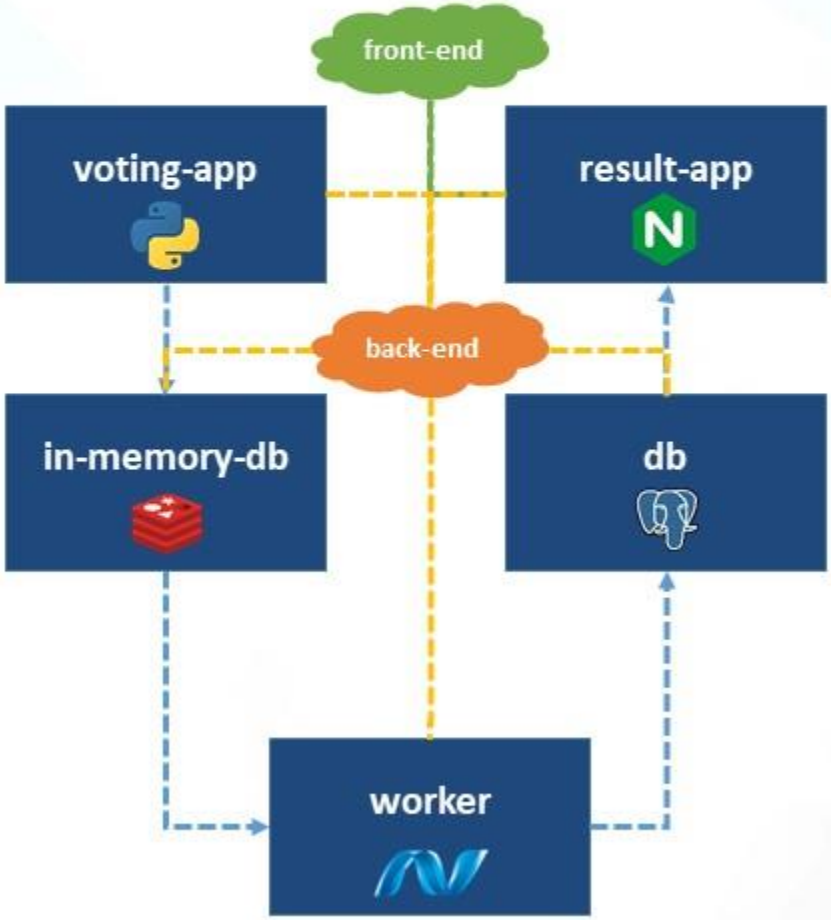
```
version: "3"
services:
  redis:
    image: redis
  db:
    image: postgres:9.4
  vote:
    image: voting-app
    ports:
      - 5000:80
    depends_on:
      - redis
```

**version: 2**

SkillAssure

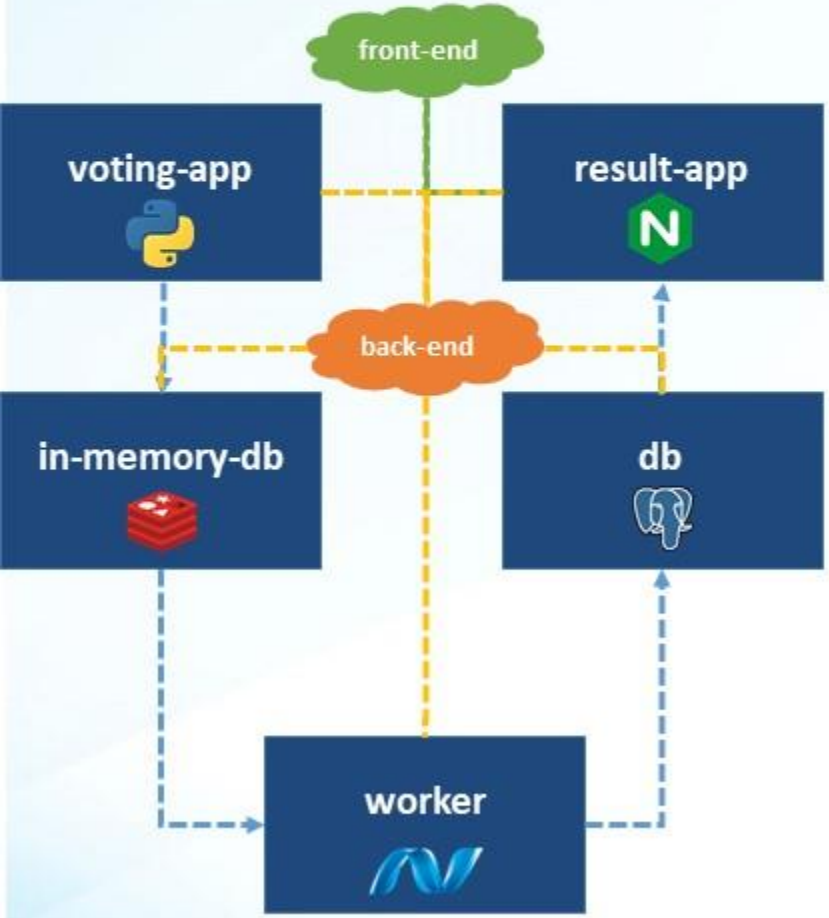
# Sample Application – Voting App

- Let's say we would like to separate the user generated traffic from the applications internal traffic
- We can create two dedicated networks
  - front-end for traffic from users
  - back-end for traffic within the application.
- Then we connect the user facing applications, i.e. voting app and the result app to the frontend network
- Connect all other component to an internal backend network



SkillAssure

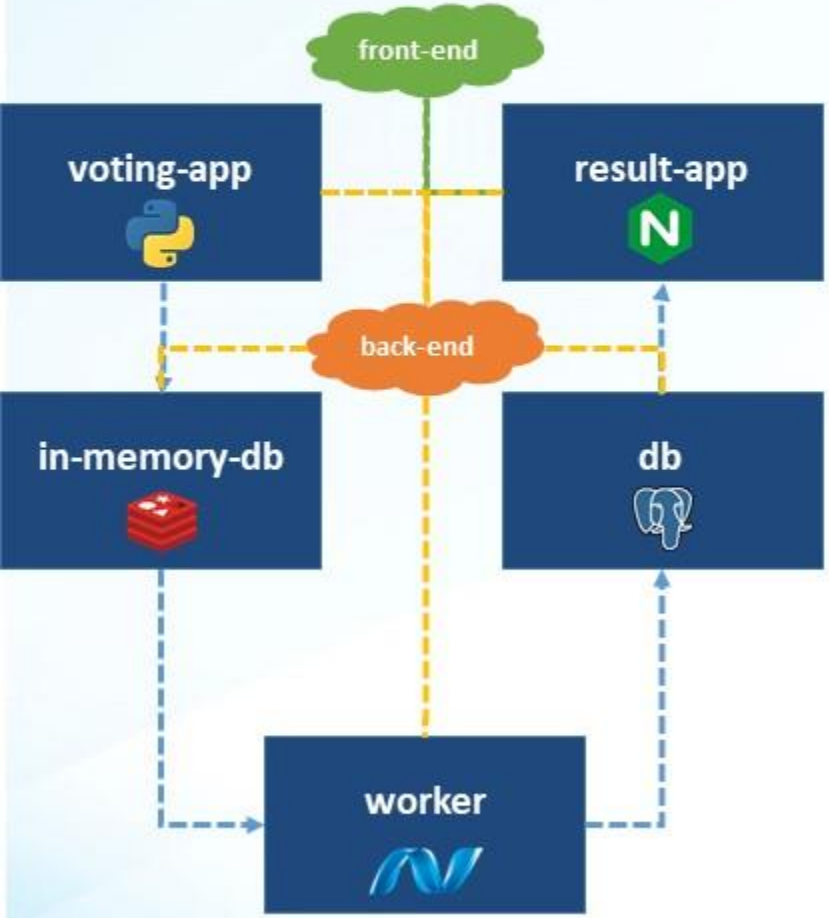
# Sample Application – Voting App




- Define the two networks, i.e. front-end and back-end
- **Create a new property called networks at the roots level and add both networks.**

SkillAssure

# Sample Application – Voting App

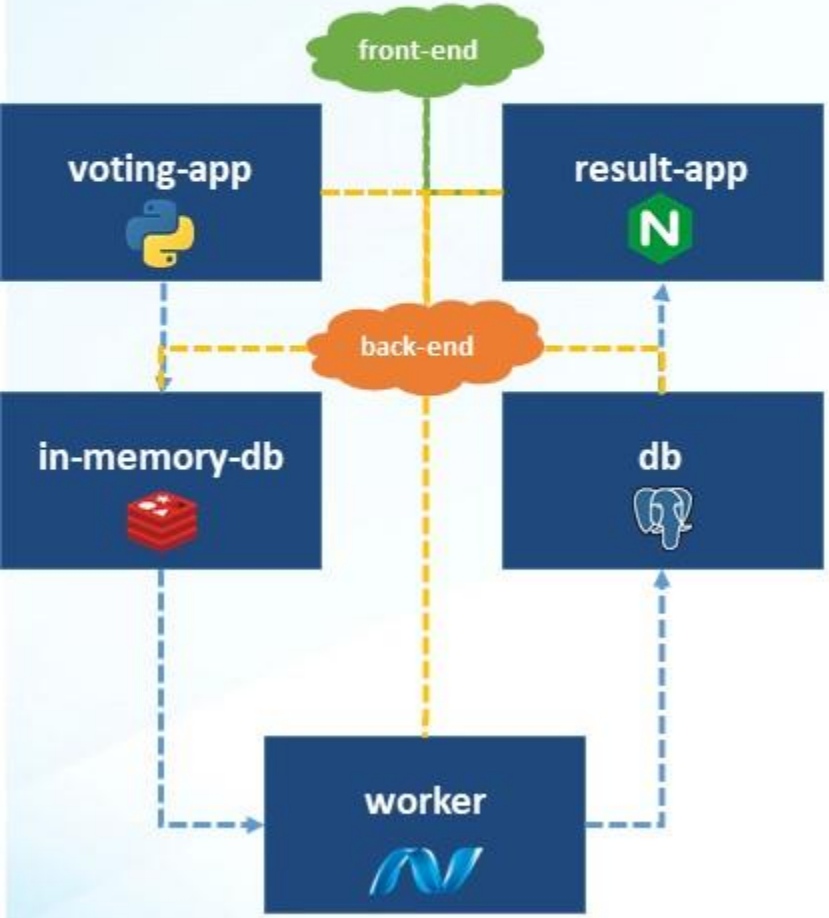



docker-compose.yml

```
version: 2
services:
  redis:
    image: redis

  db:
    image: postgres:9.4

  vote:
    image: voting-app

  result:
    image: voting-app

networks:
  front-end:
  back-end:
```

SkillAssure

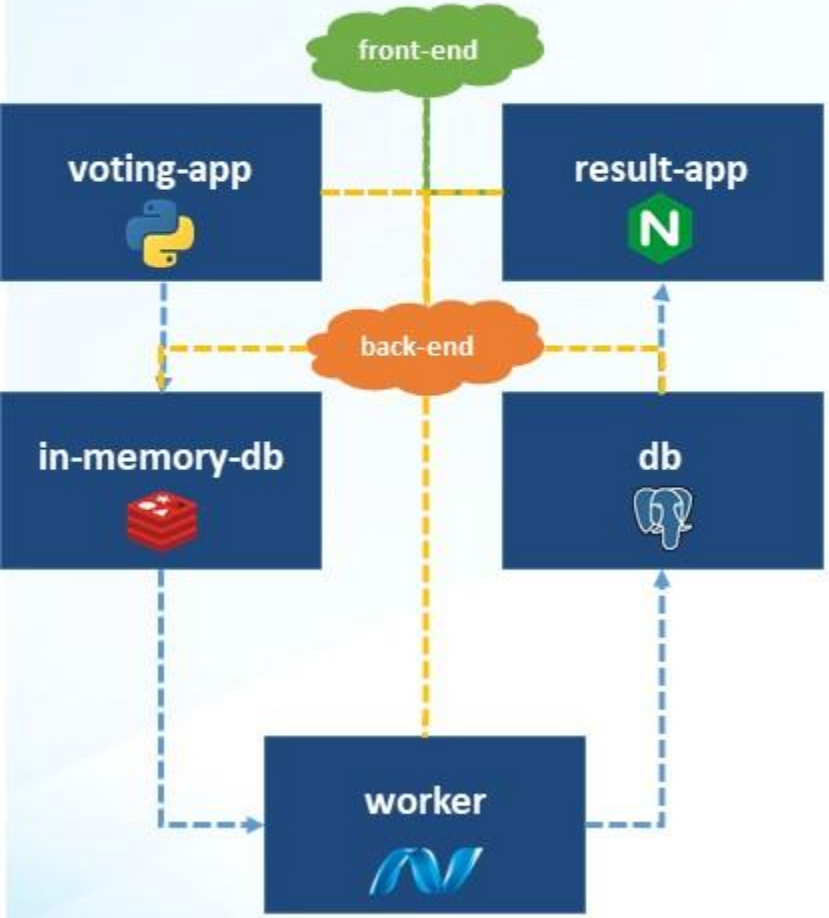
# Sample Application – Voting App




- **Under as each service create a networks property and provide a list of networks that service must be attached to**
- redis, worker & db
  - back-end
- Voting app & Result app
  - back-end
  - front-end

SkillAssure

# Sample Application – Voting App



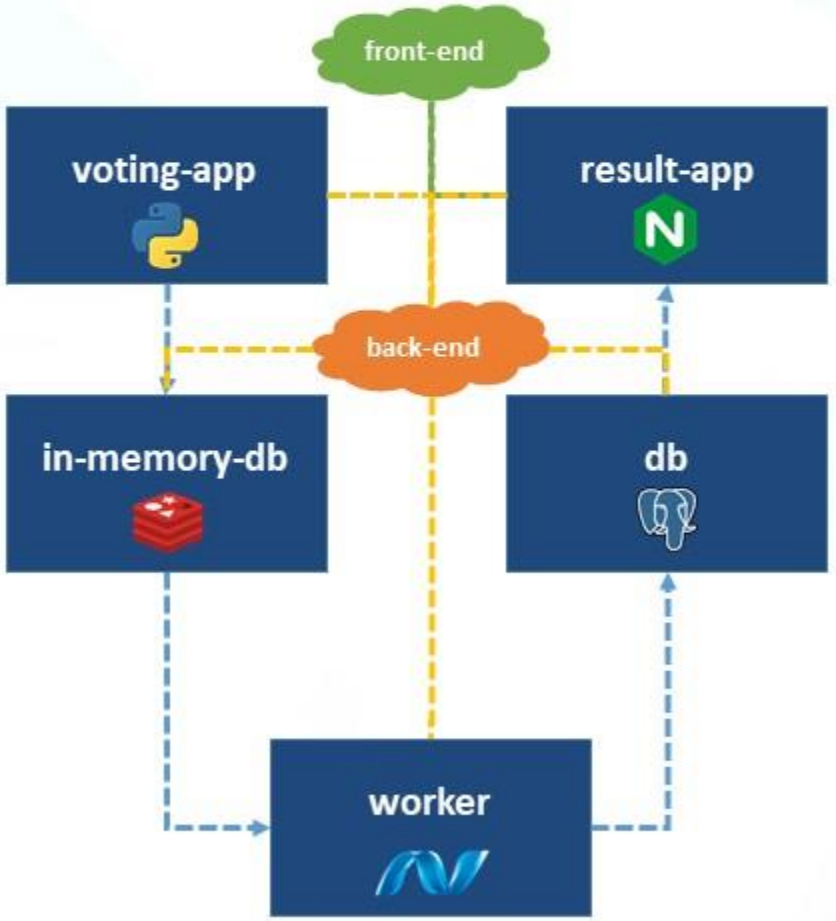

docker-compose.yml

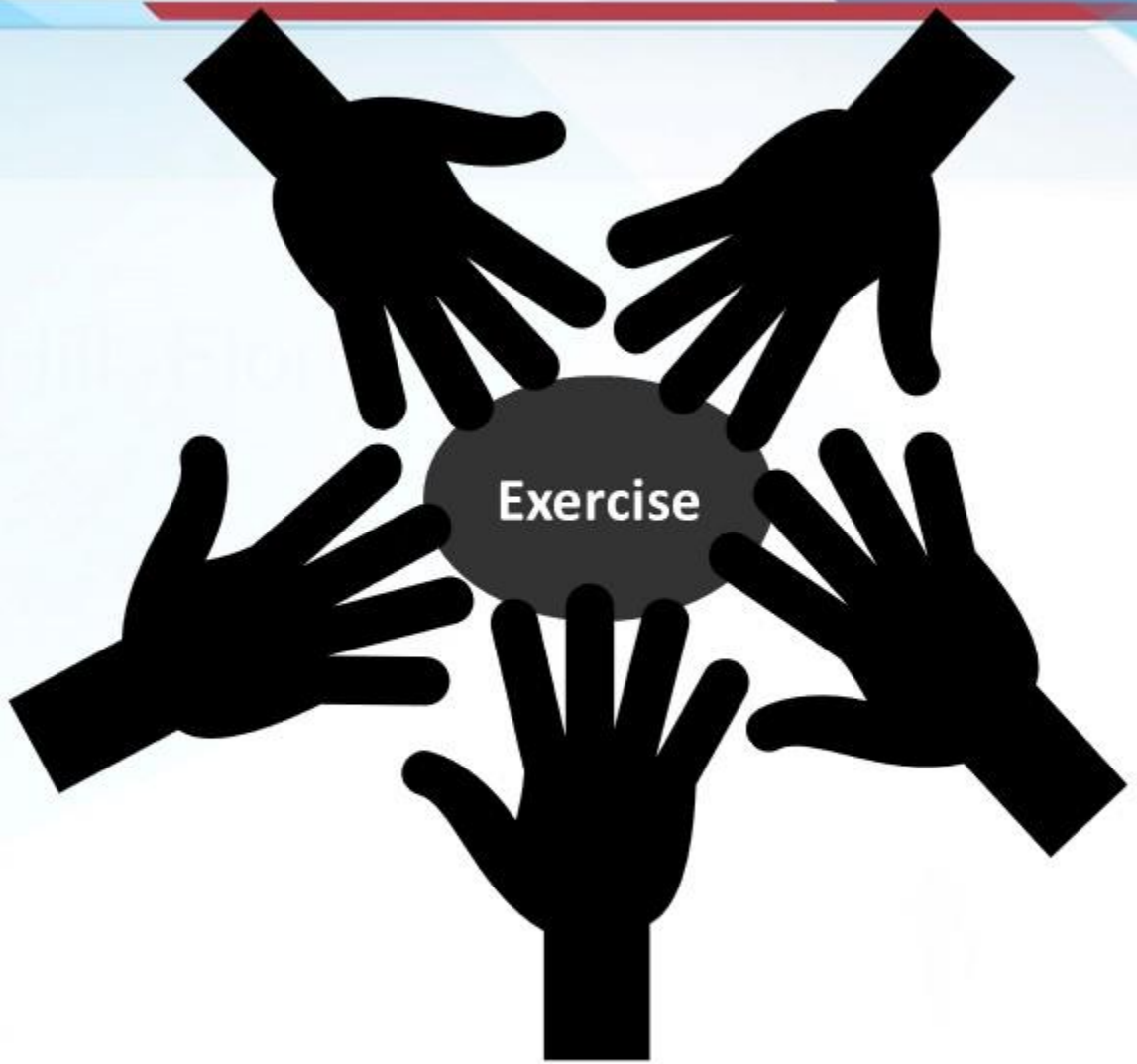
```
version: 2
services:
  redis:
    image: redis
    networks:
      - back-end
  db:
    image: postgres:9.4
    networks:
      - back-end
  vote:
    image: voting-app
    networks:
      - front-end
      - back-end
  result:
    image: voting-app
    networks:
      - front-end
      - back-end
networks:
  front-end:
  back-end:
```

SkillAssure

# Exercise

- Modify the docker-compose file to add network information as per the give diagram.




Exercise