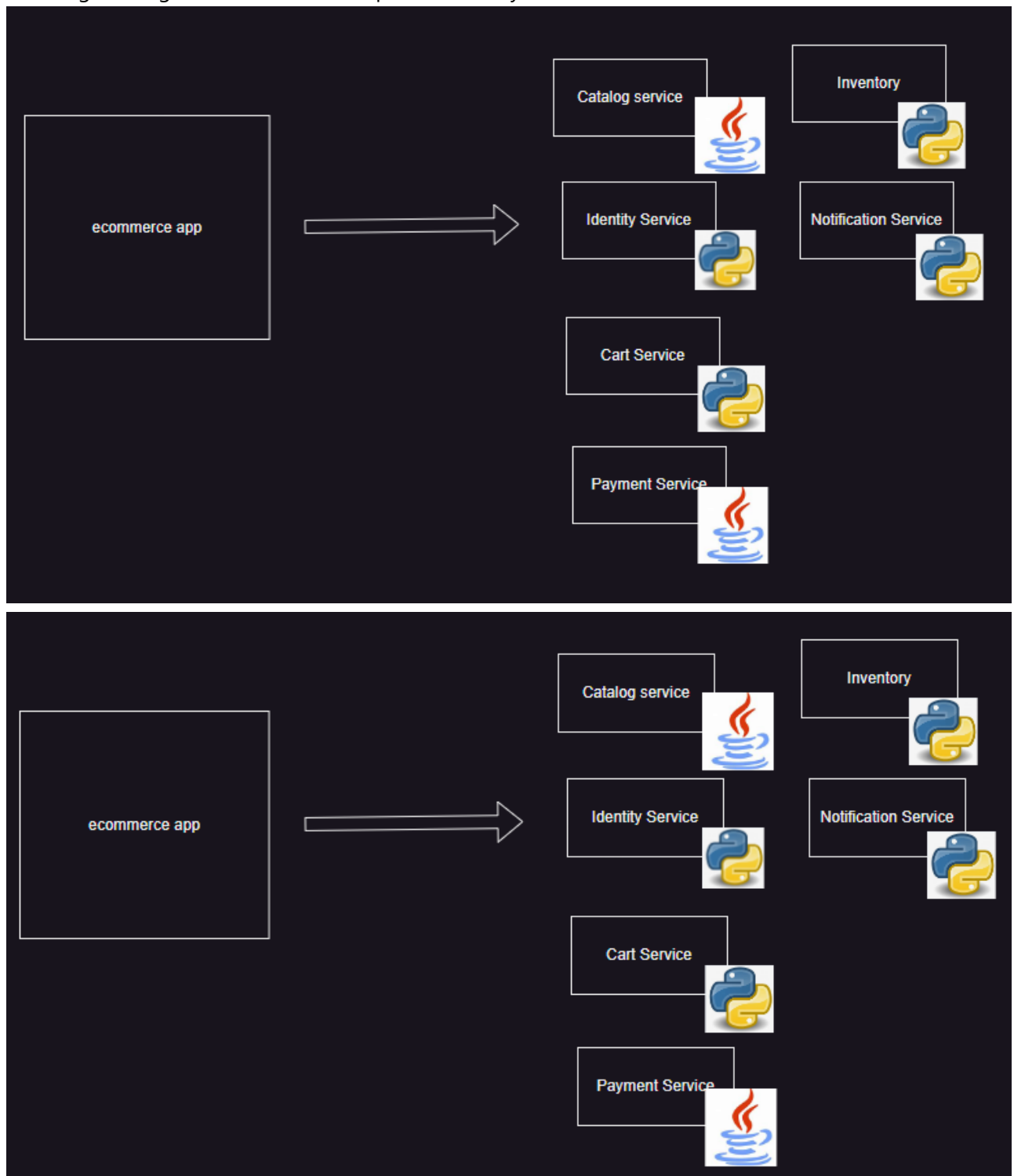# Microservices at a 10,000 feet overview

- Breaking the large monolith into multiple individually runnable services

- Advantages
  - Each micro service can be in a best suite technology
  - Microservcies can be reused across projects
  - Microservies can be upgraded or deployed without impacting other services (Zero down time deployments are easier)
- Disadvantages:
  - Handling these many services require an orchestrator

Sample Applications to be containerized

- Java (Spring boot):
    - spring petclinic [Refer Here](#)
- Dotnet (aspnet.core):
    - monolith application: nopcommerce [Refer Here](#)
- Python (Fast API):
    - A Fast api based web api

Ways to containerize

- Figure out the below manually
    - what is required to run the application
    - how to deploy application
    - What command to execute to start the application.
    - on which port is application accessible
- Try configuring application manually once
    - Virtual machine
    - container
- Dockerfile

Spring pet clinic

- Requires:
    - java 17
    - link to download spring petclinic [Refer Here](#)
    - Command to run the application `java -jar <path to spring petclinic jar>`
    - It runs on port 8080

Setup on a virtual machine

- create a linux vm on azure/aws
- This applicaton runs on 8080 port so open 8080 in security groups
- setup java

```
sudo apt update
sudo apt install openjdk-17-jdk -y
```
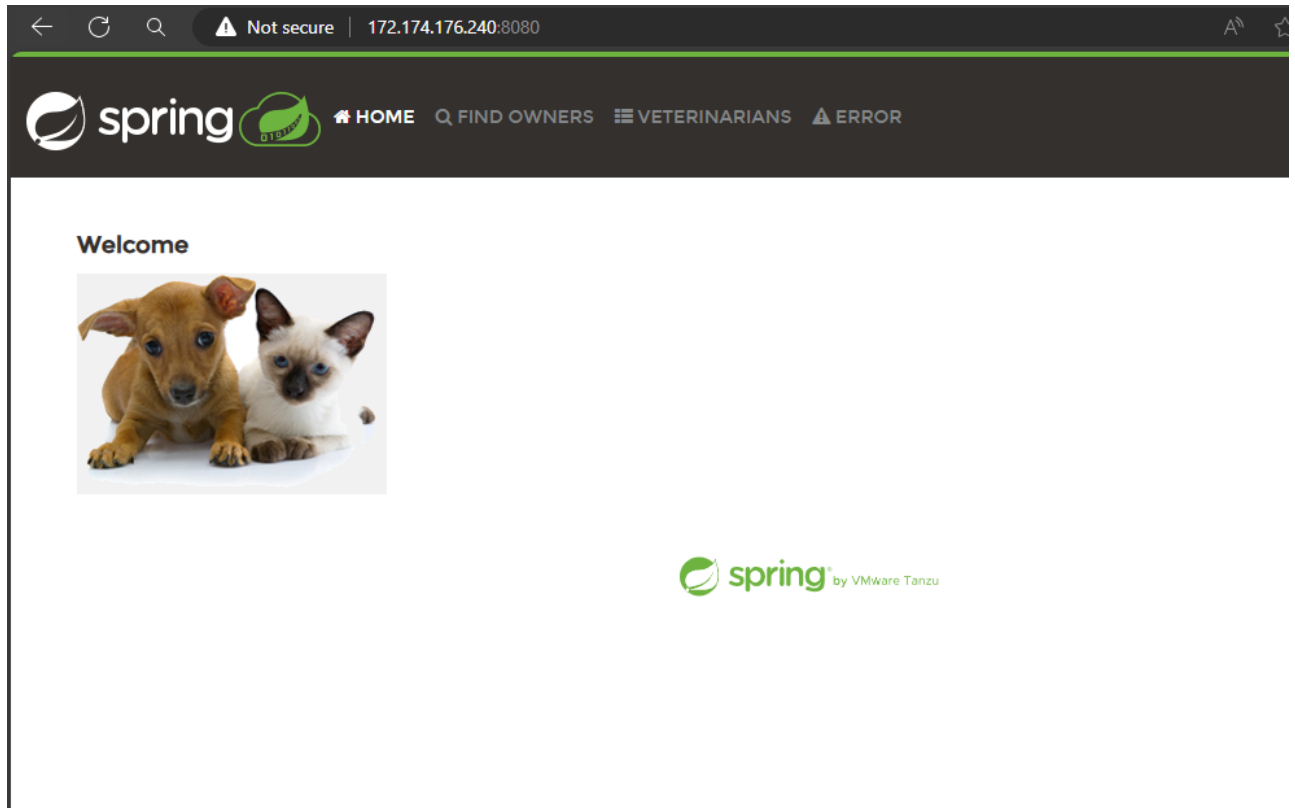
- Download spring petclinic jar

```
# cd into home directory
cd ~
wget https://khajareferenceapps.s3.ap-south-1.amazonaws.com/spring-petclinic-3.2.0-SNAPSHOT.jar
```

- Run the application

```
java -jar spring-petclinic-3.2.0-SNAPSHOT.jar
```
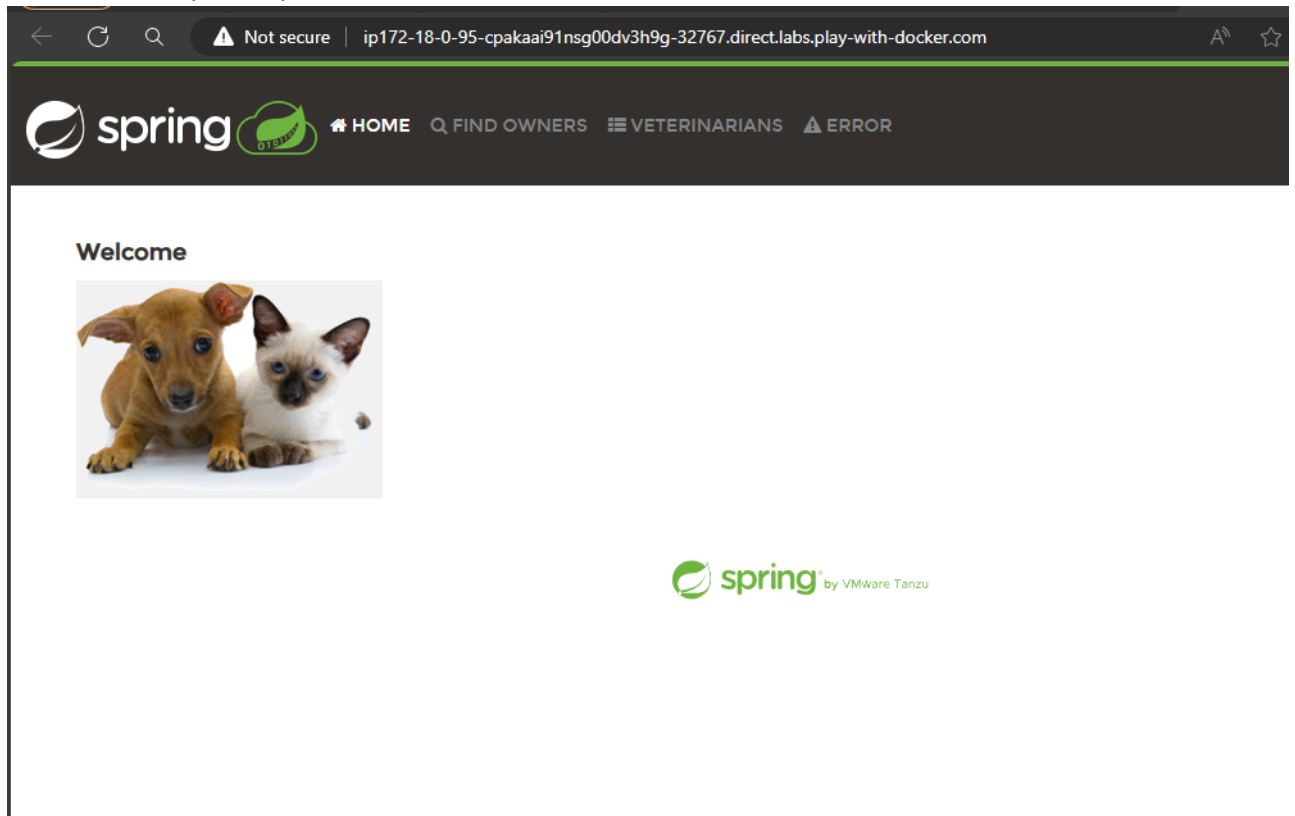
- Now access the application on `http://<public-ip>:8080`

Setup manually in a container

- java 17 is required to run the spring petclinic
- For jdk 17 lets use `amazoncorretto:17`

```
docker container run --name manualspc -p 32767:8080 -it amazoncorretto:17
/bin/bash
```

- Access the vm public ip `http://<publicip>:32767`

Next Steps

- Containerizing the application using Dockerfiles.