SERVICE ORIENTED ARCHITECTURES

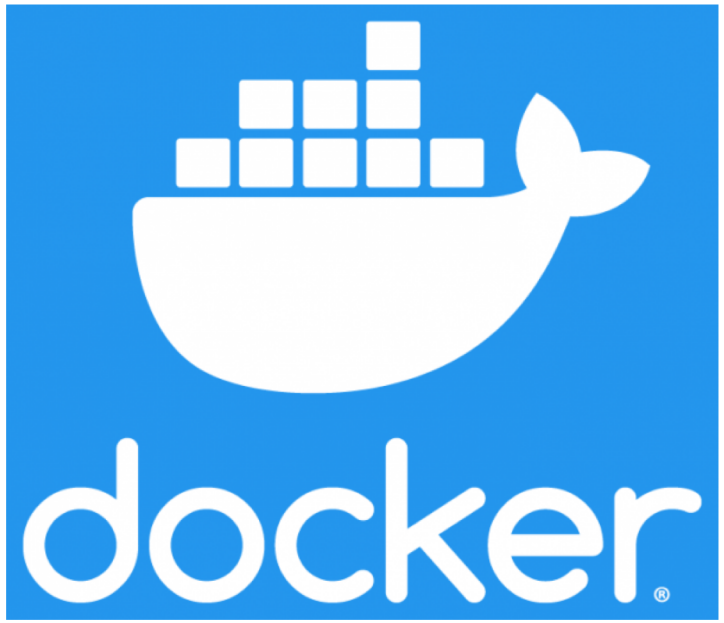
ACIT3855 – WINTER 2024

AGENDA

- Quick Review – Docker and Docker Compose, Virtual Environments
- Quiz 6
- Midterm Review Quiz
- Topics:
  - Containerizing Your Services for a Test Environment
- Lab 6B – Demos
- Lab 7 – Containerization with Docker and Docker Compose

REVIEW

- Docker
- Docker Compose



REVIEW

- Virtual Environments
- requirements.txt

virtualenv

connexion==2.7.0
requests==2.25.1
APscheduler==3.7.0

connexion==2.7.0
pykafka==2.8.0

QUIZ 6

- Quiz is on the Learning Hub
- Open book, do your own work
- You have <15 minutes to complete it

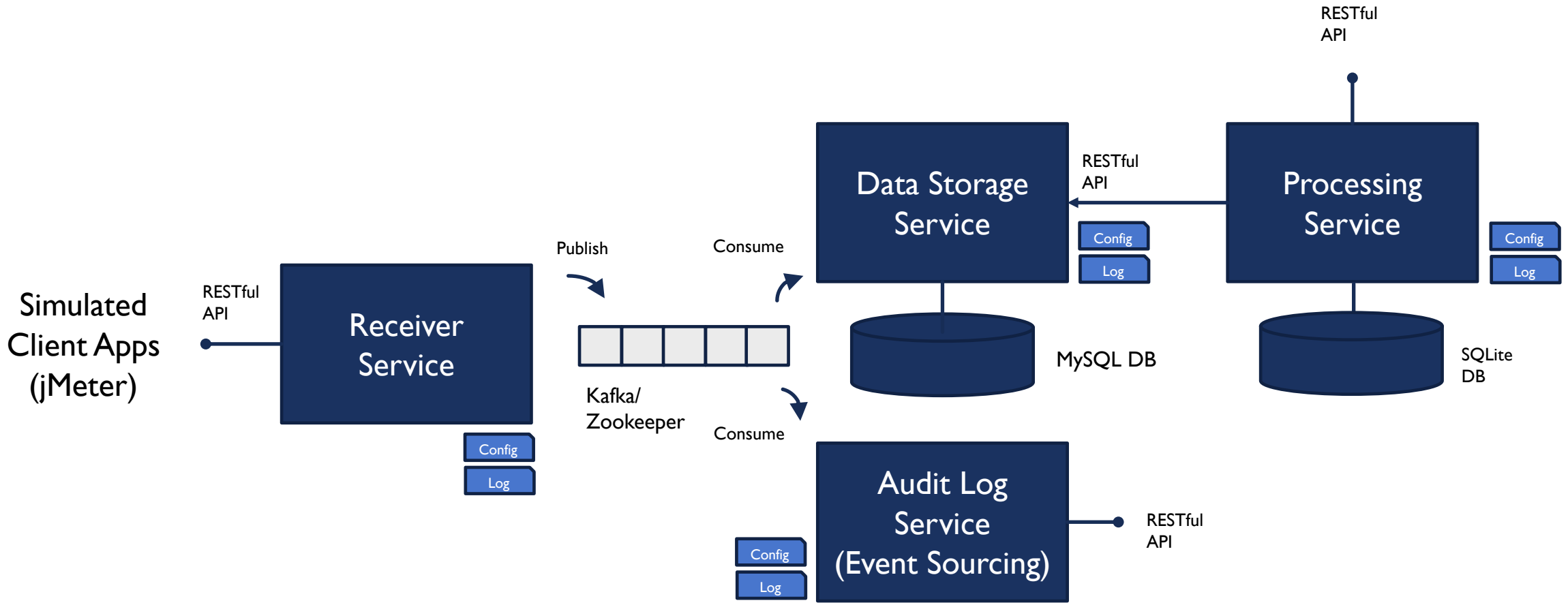
COURSE SCHEDULE

Week	Topics	Notes
1	• Services Based Architecture Overview • RESTful APIs Review	Lab 1
2	• Microservices Overview • Edge Service	Lab 2, Quiz 1
3	• Database Per Service • Storage Service (SQLite)	Lab 3, Quiz 2
4	• Logging, Debugging and Configuration • Storage Service (MySQL)	Lab 4, Quiz 3
5	• RESTful API Specification (OpenAPI) • Processing Service	Lab 5, Quiz 4
6	• Synchronous vs Asynchronous Communication • Message Broker Setup, Messaging and Event Sourcing	Lab 6, Quiz 5
7	• Deployment - Containerization of Services *Note: At home lab for Monday Set*	Lab 7, Quiz 6 (Sets A and B) , Assignment 1 Due
8	• Midterm Week	Midterm Review Quiz
9	• Dashboard UI and CORS	Lab 8, Quiz 6 (Set C), Quiz 7
10	• Spring Break	No Class
11	• Issues and Technical Debt	Lab 9, Quiz 8
12	• Deployment – Centralized Configuration and Logging	Lab 10, Quiz 9
13	• Deployment – Load Balancing and Scaling *Note: At home lab for Monday Set*	Lab 11, Quiz 10 (Sets A and B)
14	• Final Exam Preview	Quiz 10 (Set C), Assignment 2 Due
15	• Final Exam	

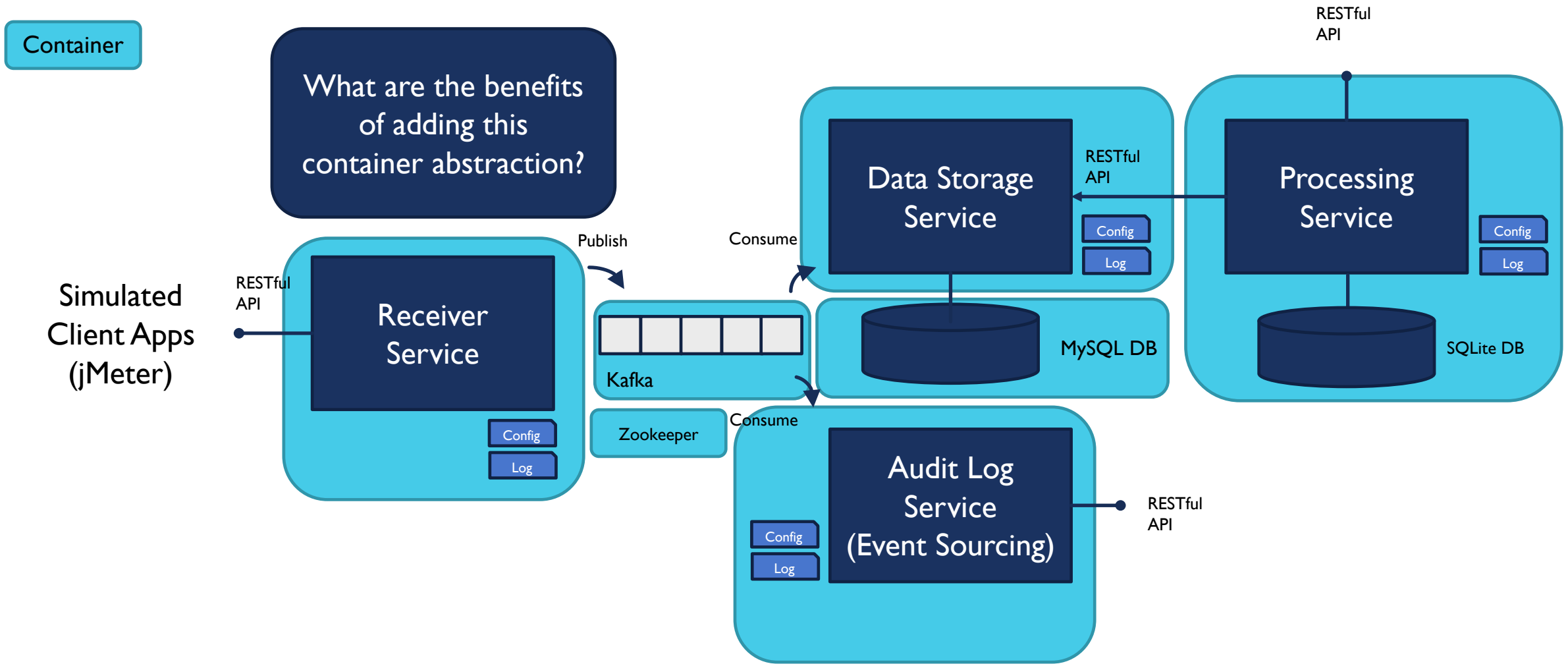
MIDTERM REVIEW QUIZ

- Next Week, Online
- During approximately the first hour of scheduled class time
- Covers topics from Week 1 to Week 7
- Includes multiple choice, multiple select, T/F and written response type questions

OUR SAMPLE APPLICATION – LAB 6B



OUR SAMPLE APPLICATION – LAB 7 – CONTAINERIZATION

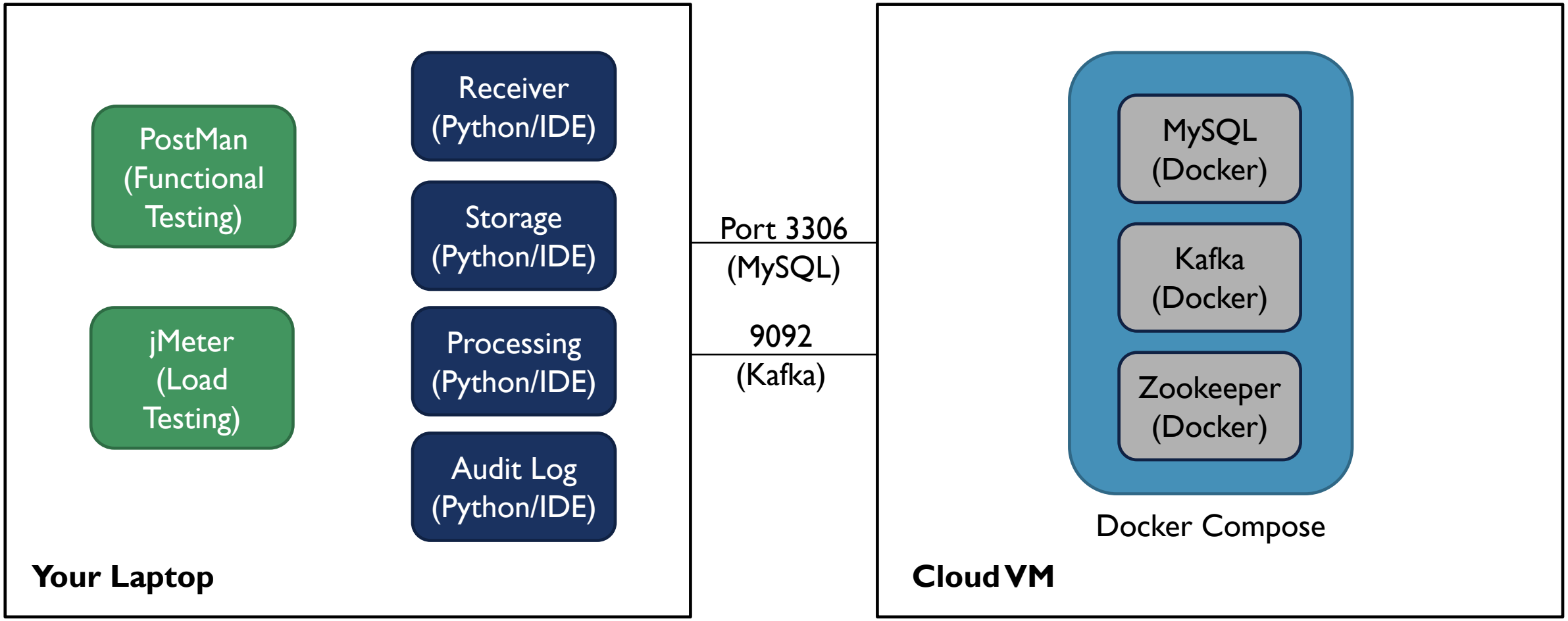


DOCKER - BENEFITS

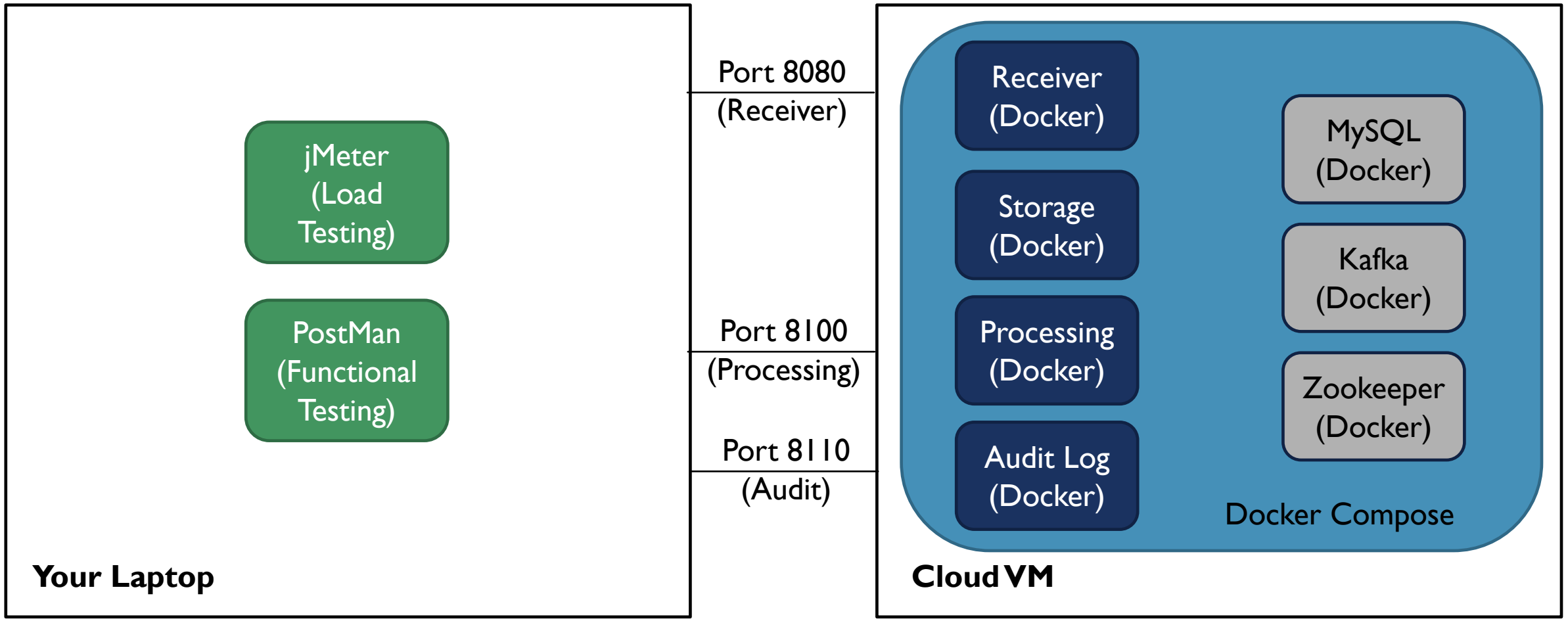
What are some benefits of using Docker?

- Known/standardized configurations – the base image and any additional installed software/configurations are defined in a Dockerfile
- Can start and stop as needed. So can scale up and down as well
- Deployment platforms are built on top of Docker images (i.e., Kubernetes – K8s, OKD)

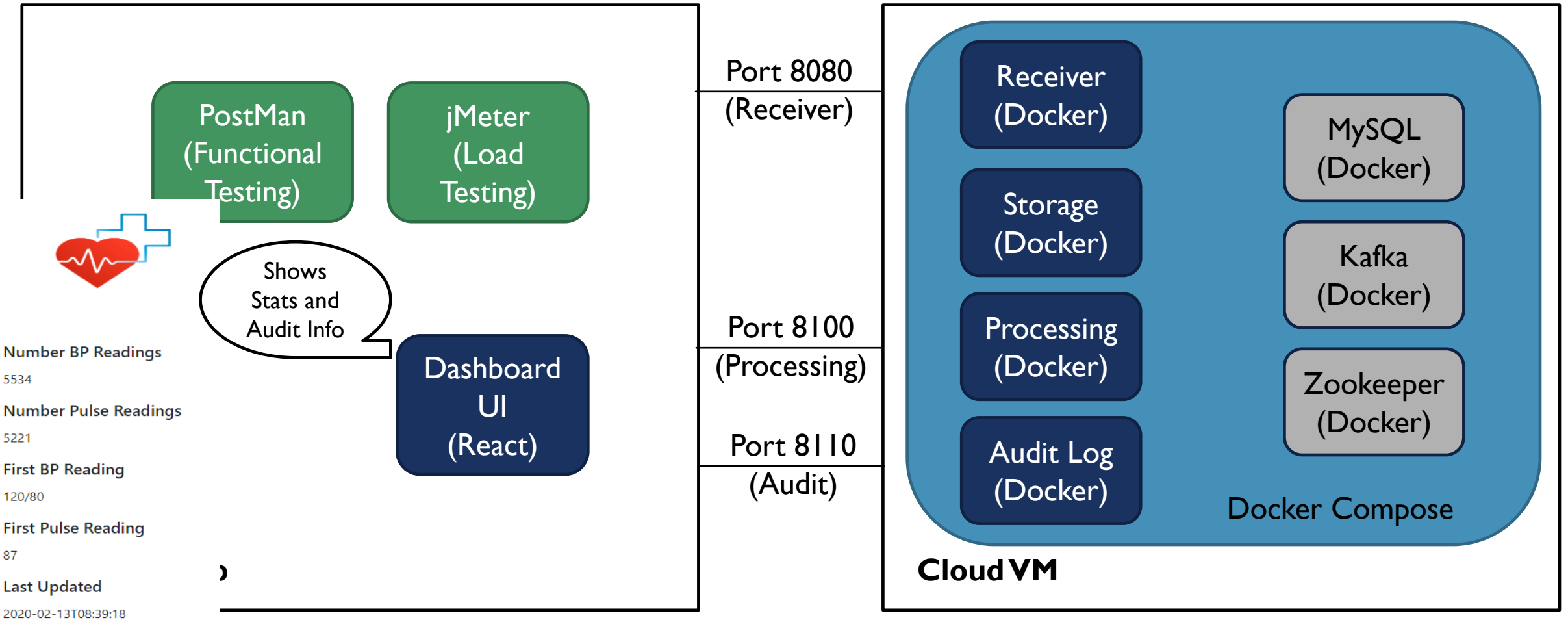
CURRENT DEVELOPMENT ENVIRONMENT



LAB 7 TEST ENVIRONMENT



LAB 8 TEST ENVIRONMENT



DOCKERFILE – EXAMPLE FOR CONNEXION APP

```
FROM ubuntu:18.04

LABEL maintainer="mmulder10@bcit.ca"

RUN apt-get update -y && \
    apt-get install -y python3 python3-pip

# We copy just the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip3 install -r requirements.txt

COPY . /app

ENTRYPOINT [ "python3" ]

CMD [ "app.py" ]
```

Base Image

Image Maintainer

Run Command to Install Software on Our Image

Copy our requirements.txt into an app directory

Default Dir for Container Runtime

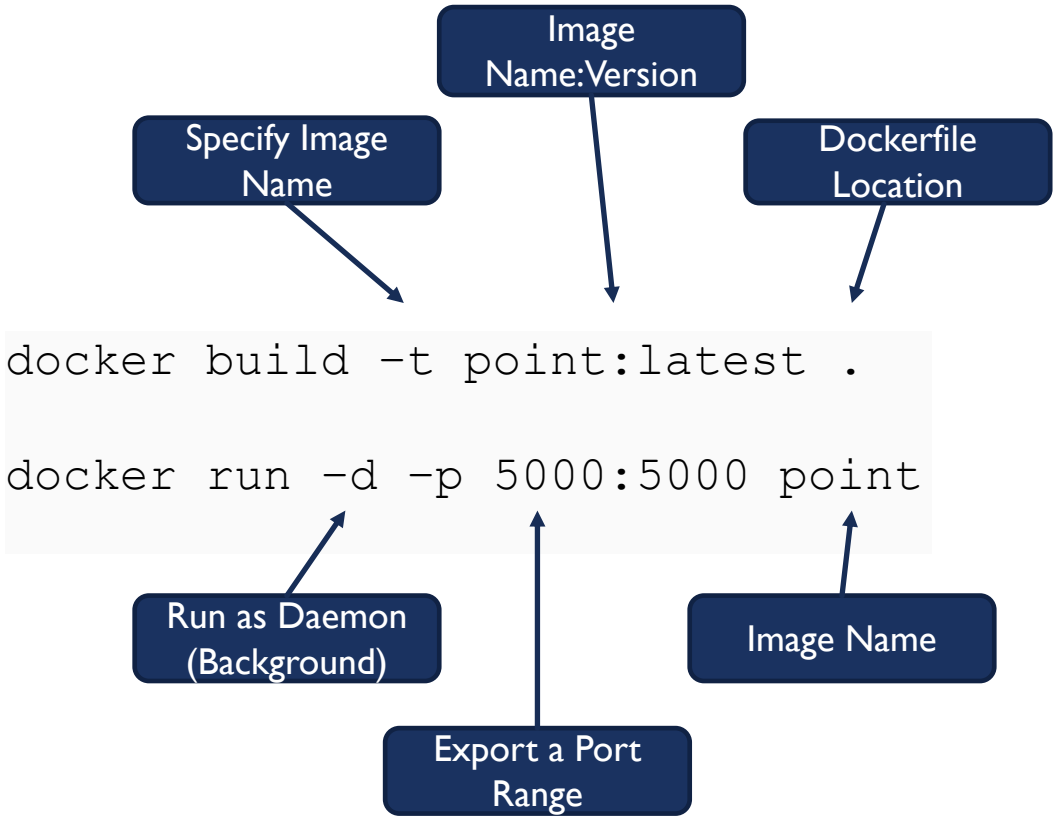
Run Command to Install Python Dependencies

Copy our code into the app directory

Entrypoint – Run when Container Started

CMD – Argument to ENTRYPOINT

DOCKER – BUILDING AND IMAGE AND RUNNING A CONTAINER



DOCKER – RELEVANT COMMANDS

- docker build – Used to build an image from a Dockerfile
- docker images – Used to list the current set of built images
- docker run – Used to run an image as a container
- docker stop – Used to stop a running image
- docker ps – Used to list the running images

Here is a Docker cheat sheet of commands: https://devhints.io/docker

DOCKER COMPOSE

Even with Docker, there still are a lot of steps to build and start/stop services when you have many services and/or the services have dependencies (i.e., the order in which the services are started/stopped is important).

Docker Compose provides an abstraction on top of Docker that lets you define service groupings, including 3rd party services (i.e., messaging, DB) in a single file (docker-compose.yml).

Operations then can be applied to the entire service grouping – such as start all (docker-compose up) and stop all (docker-compose down).

Regular Docker commands can still be used even when you use Docker Compose.

DOCKER COMPOSE FILE - SIMPLE

```
version: '3.3'

services:

  web:
    build: .
    ports:
      - "5000:5000"

  redis:
    image: "redis:alpine"
```

Version of Docker Compose (Currently 3.X)

List of Services

Service Definition (with Dockerfile):
- Path to Dockerfile to Build
- Ports to Expose
- …

Service Definition (from Registry):
- Name of image
- …

DOCKER COMPOSE FILE - COMPLEX

```
version: '3.3'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181"
    hostname: zookeeper
  kafka:
    image: wurstmeister/kafka
    command: [start-kafka.sh]
    ports:
      - "9092:9092"
    hostname: kafka
    environment:
      KAFKA_CREATE_TOPICS: "events:1:1" # topic:partition:replicas
      KAFKA_ADVERTISED_HOST_NAME: acit38500-demo.azure.com  # docker-machine ip
      KAFKA_LISTENERS: INSIDE://:29092,OUTSIDE://:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INSIDE
      KAFKA_ADVERTISED_LISTENERS: INSIDE://kafka:29092,OUTSIDE://acit38500-demo.azure.com:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - "zookeeper"
```

DOCKER COMPOSE – RELEVANT COMMANDS

- docker compose — Lists out all available commands
- docker compose up –d — Runs all the services defined in the docker-compose.yml (in the current directory)
- docker compose down — Stops all the services defined in the docker-compose.yml (in the current directory)

Docker Compose Reference: https://docs.docker.com/compose/compose-file/

DOCKER SUMMARY

Docker	Core for building images and running containers.	Use for a small number of independent services.
Docker Compose	Build on top of Docker and part of the Docker Ecosystem.	Use for groupings of related services where the number and type are static (i.e., no scaling). Good for development and test environments, and simple production environments.
Container Orchestration	Typically 3rd party software for deployment of Docker containers, often over a cluster of VMs. Docker does have the built-in "Docker Swarm".	Use for large number of services and/or when services need to be automatically scaled up and down.

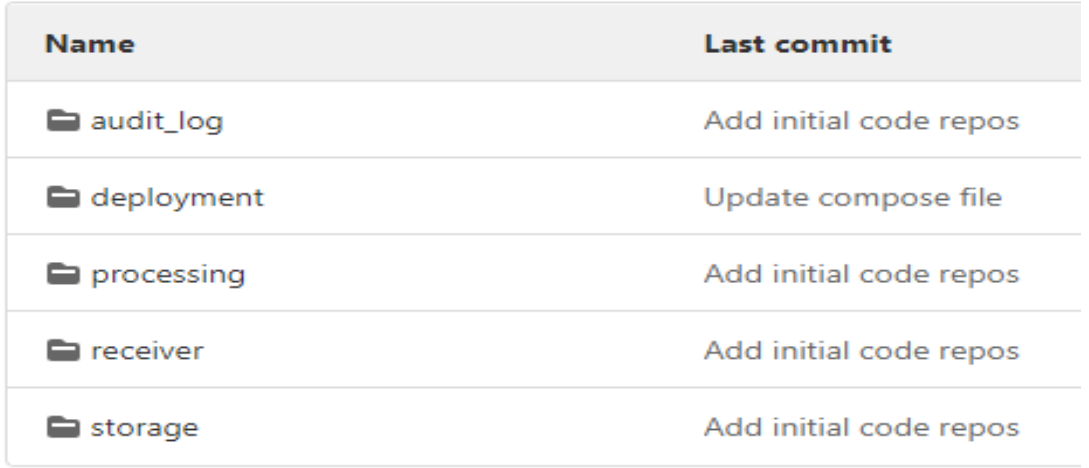
DOCKER SUMMARY

Docker	Core for building images and running containers.	Use for a small number of independent services.
Docker Compose	Build on top of Docker and part of the Docker Ecosystem.	Use for groupings of related services where the number and type are static (i.e., no scaling). Good for development and test environments, and simple production environments.
Container Orchestration	Typically 3rd party software for deployment of Docker containers, often over a cluster of VMs. Docker does have the built-in "Docker Swarm".	Use for large number of services and/or when services need to be automatically scaled up and down.

SOURCE CODE MANAGEMENT - REPOSITORIES

Best Practices for Microservices:

- One Repository Per Service
  - Provides separation between services to prevent dependencies
  - Harder to manage with many services or smaller services
  - Better for larger teams and/or larger services
- One Repository, Multiple Folders
  - Easier to manage than multiple repositories
  - Still provides some separation between services
  - Better for smaller teams and/or smaller services



You will be transitioning your code to a single Git Repo (with subfolders) to make it easier to share code between the development and test environments.

You can use a public GitHub or GitLab repo for this.

PYTHON 3RD PARTY DEPENDENCIES

- Currently we install them using PIP in our Python environment and/or our IDE
  - It can be hard to keep track of which dependencies you need for a particular project!
- Best practice in Python is to create a requirements.txt file with each 3rd party dependency defined with the version

Example requirements.txt for a Connexion App:

```
connexion==2.7.0
swagger-ui-bundle==0.0.8
pykakfa==2.8.0
```

Run the following to install the dependencies in a test environment:

```
pip3 install -r requirements.txt
```

WORKING IN YOUR GIT REPOS - BEST PRACTICE

- If you are making code or configuration changes, make those changes in the cloned Git repo on your laptop
- Push those changes to the corresponding Git repo on GitLab/GitHub
- Pull those changes on your VM
- Only specific configuration values (i.e., URLs, passwords) may be changed (but NOT pushed to the repo) on the VM

Otherwise it can get confusing which version of the code is up-to-date between your laptop and VM. It can also result in the dreaded merge conflicts.

LAB 7 - TROUBLESHOOTING

- These two commands are very helpful when troubleshooting your Docker containers:
  - docker logs <container name or id> # Shows the full logs for the container
  - docker logs <container name or id> -f # Follows the logs as they are updated
- Also:
  - docker compose logs -f # But the output gets a little busy
- We'll demo these in a running installation next slide…

DOCKER LOGS

Running Normally

```
[azureuser@Acit3855:~/acit3855/receiver$ docker logs c2929e0f0d00
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
2023-10-19 11:58:02,459 - werkzeug - WARNING -  * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
2023-10-19 11:58:02,459 - werkzeug - INFO -  * Running on http://172.17.0.2:8080/ (Press CTRL+C to qui
t)
```

After An Error

```
[azureuser@Acit3855:~/acit3855/receiver$ docker logs c2929e0f0d00
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
2023-10-19 11:58:02,459 - werkzeug - WARNING -  * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
2023-10-19 11:58:02,459 - werkzeug - INFO -  * Running on http://172.17.0.2:8080/ (Press CTRL+C to qui
t)
2023-10-19 11:58:52,939 - basicLogger - INFO - Received event Blood Pressure request with a unique id
of d290f1ee-6c54-4b01-90e6-d701748f0851
2023-10-19 11:58:52,960 - app - ERROR - Exception on /blood-pressure [POST]
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 2073, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1518, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1516, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.6/dist-packages/flask/app.py", line 1502, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
  File "/usr/local/lib/python3.6/dist-packages/connexion/decorators/decorator.py", line 48, in wrapper
    response = function(request)
  File "/usr/local/lib/python3.6/dist-packages/connexion/decorators/uri_parsing.py", line 144, in wrap
per
    response = function(request)
  File "/usr/local/lib/python3.6/dist-packages/connexion/decorators/validation.py", line 184, in wrapp
er
    response = function(request)
  File "/usr/local/lib/python3.6/dist-packages/connexion/decorators/response.py", line 103, in wrapper
    response = function(request)
  File "/usr/local/lib/python3.6/dist-packages/connexion/decorators/parameter.py", line 121, in wrappe
r
    return function(**kwargs)
  File "/app/app.py", line 28, in report_blood_pressure_reading
    client = KafkaClient(hosts=hostname)
  File "/usr/local/lib/python3.6/dist-packages/pykafka/client.py", line 94, in __init__
    ssl_config=ssl_config)
  File "/usr/local/lib/python3.6/dist-packages/pykafka/cluster.py", line 199, in __init__
    self.update()
  File "/usr/local/lib/python3.6/dist-packages/pykafka/cluster.py", line 413, in update
    metadata = self._get_metadata()
  File "/usr/local/lib/python3.6/dist-packages/pykafka/cluster.py", line 276, in _get_metadata
    'Unable to connect to a broker to fetch metadata. See logs.')
pykafka.exceptions.NoBrokersAvailableError: Unable to connect to a broker to fetch metadata. See logs.
2023-10-19 11:58:52,962 - werkzeug - INFO - 192.157.124.16 - - [19/Oct/2023 11:58:52] "POST /blood-pre
ssure HTTP/1.1" 500 -
```

docker logs lets your see the console logs and any stack traces (typically associated with runtime errors). It contains the STDOUT and STDERR output from your application.

This should be your starting point for troubleshooting your services.

LAB 7

- Five Parts:
  - Add your code to Source Code Management (Git)
  - Add requirements.txt and Dockerfile to each service
  - Build and run Docker Images for each service
  - Test out your running Docker containers on your Cloud VM
  - Add your Dockerized services to your docker-compose.yml
    - Test your services, running through Docker Compose on your Cloud VM with your jMeter script

TODAY'S LAB

Today you will:

- Demo your Lab 6B, if you haven't already done so
- Work on Lab 7 – It is due via demo next class