CS 310 : Scalable Software Architectures

Class session on Tuesday, November 12th

NOVEMBER 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

www.a-printable-calendar.com

Northwestern University

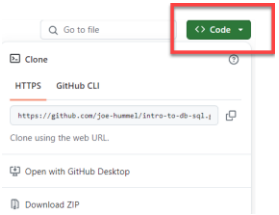
Notes:

- ***Focus this week***:
  - *Software execution and packaging*
  - *Docker*
  - *Async architectures, Distributed systems*
  - *CAP theorem*
- ***Midterms** have been graded; details in email*
- ***Project 03** due Friday (Sunday late)*
  - *Serverless, event-driven PDF analysis*
  - *Gradescope will open today*
- ***No class session Thursday** --- office hours instead*
- ***No class Tuesday 11/26** --- thanksgiving week*

Getting the necessary software

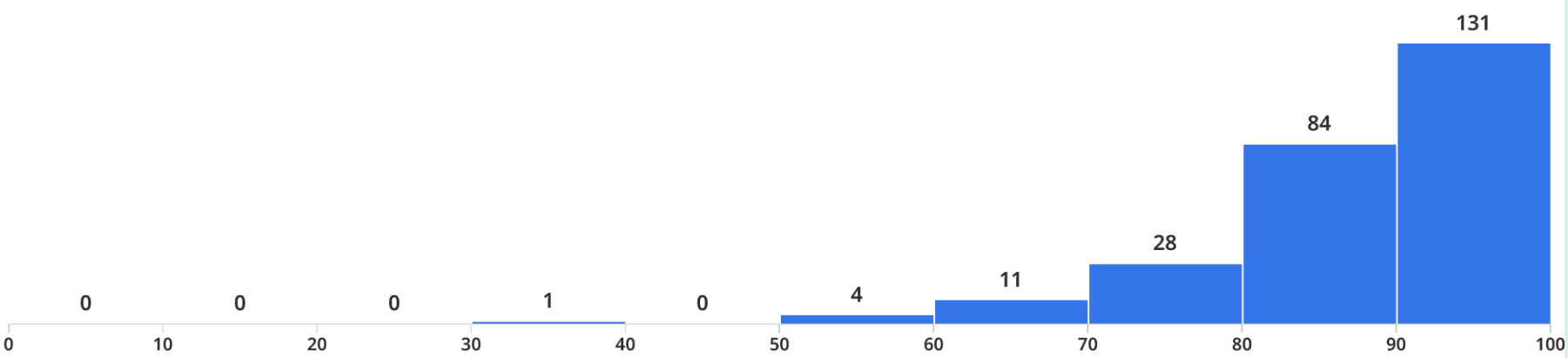
1. **Make sure Docker Desktop is running**

2. **Download files you need for today**
   - https://github.com/joe-hummel/docker-lambda-movies



- **Install desktop version of Postman app**
  - visit https://www.postman.com/
  - download desktop app (click link in lower-left corner)

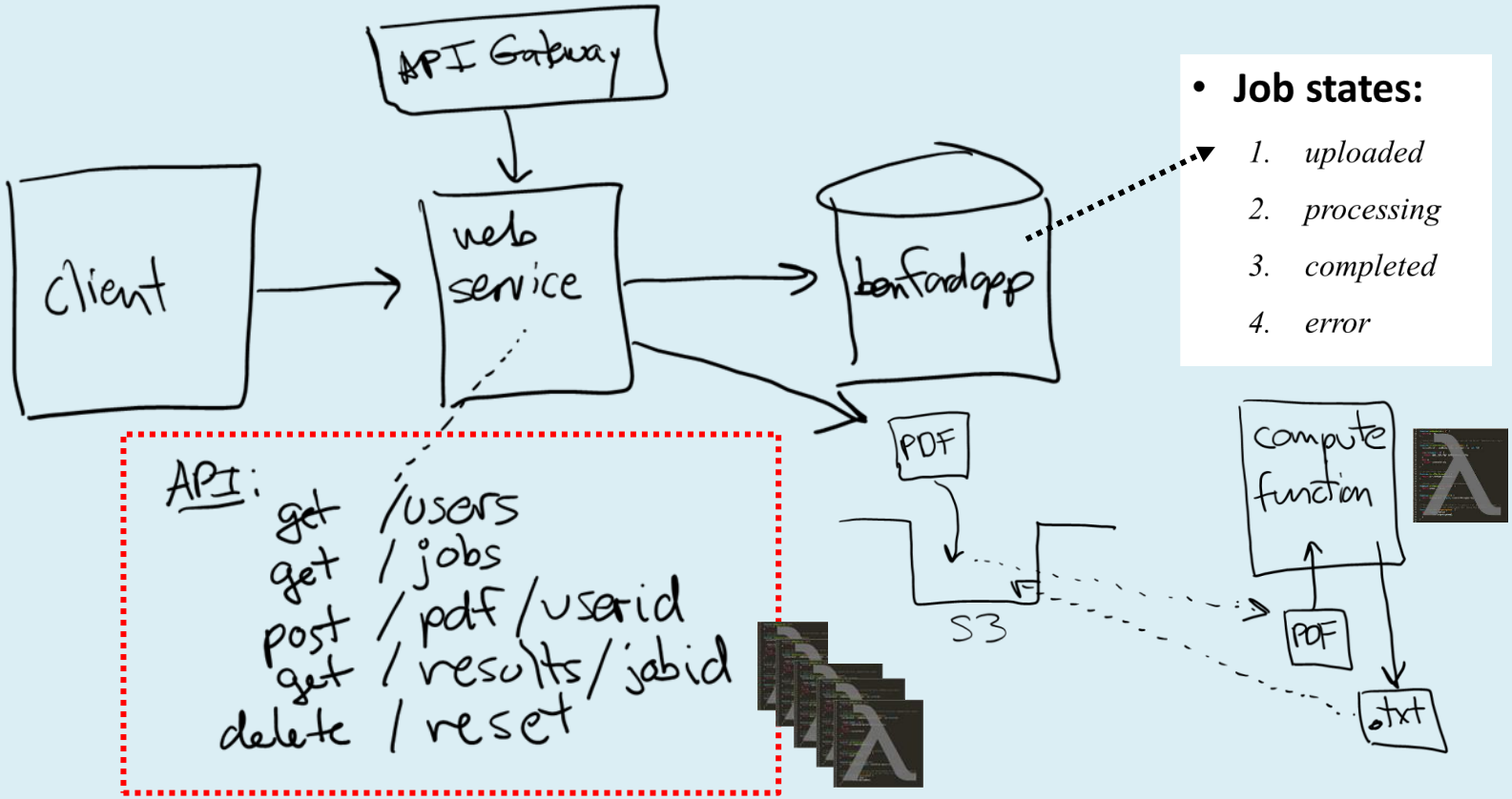
Midterm



Minimum	Median	Maximum	Mean	Std Dev
30.0	90.0	100.0	86.83	9.99

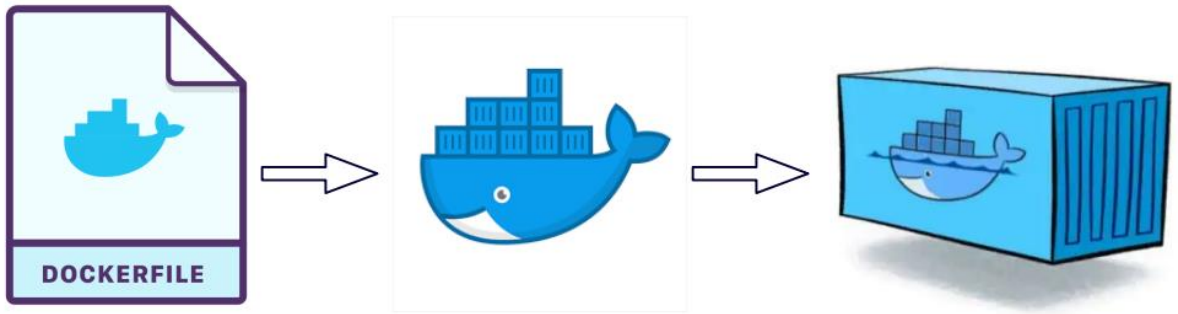
Project 03

- **Project 03 uses a different architectural approach:**
  1. *Event-driven*
  2. *Serverless*
  3. *Asynchronous API*



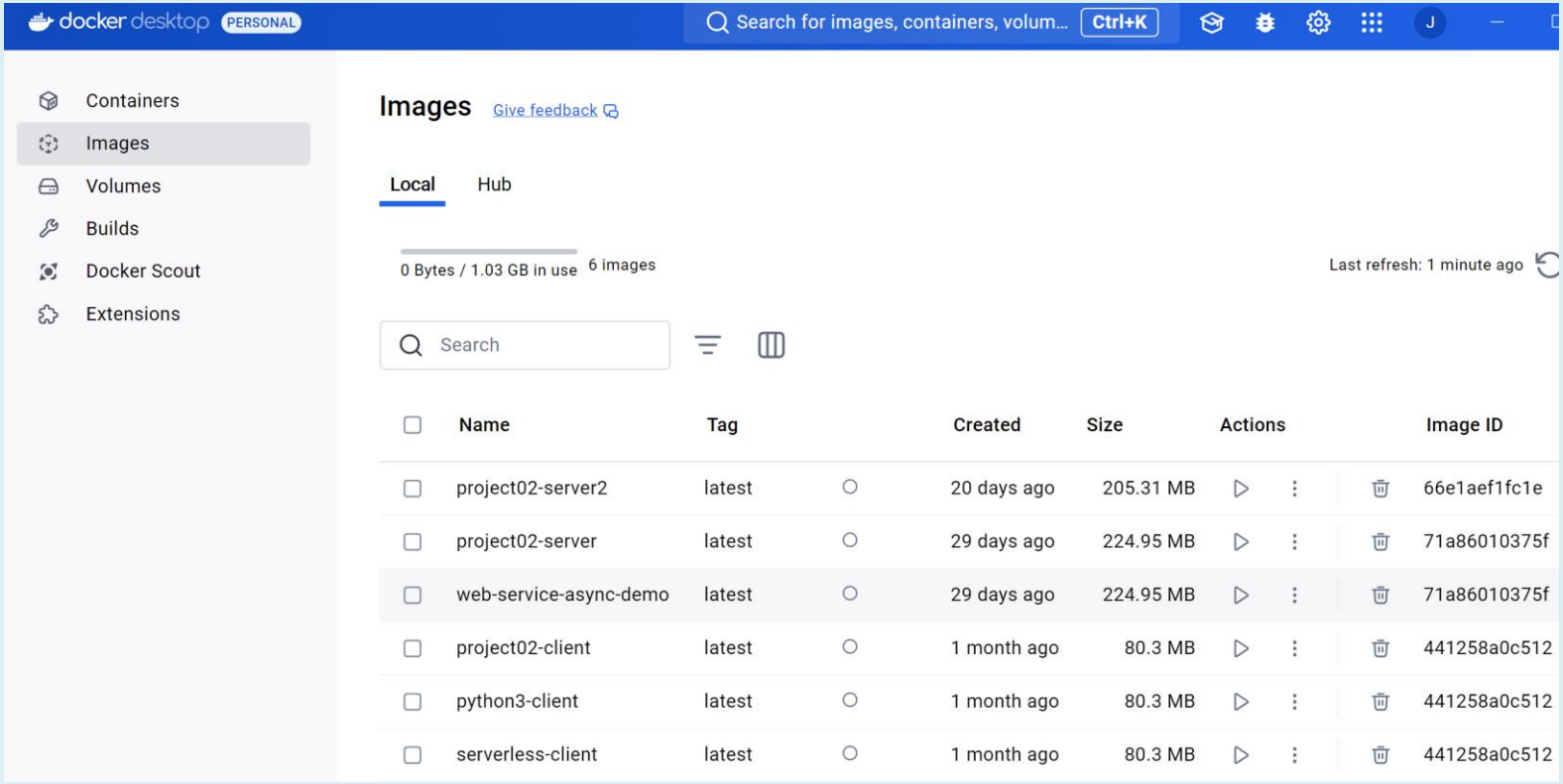
Overview

- A **Dockerfile** is a text document that contains the commands needed to build a **docker image**
- Think of a **docker image** as a snapshot of the software – a large file that can be stored and shared
  - ***Docker Hub*** *is the app store for docker images*
- A **docker container** runs a docker image



Images vs. Containers

- **Docker desktop** will show your images and containers…

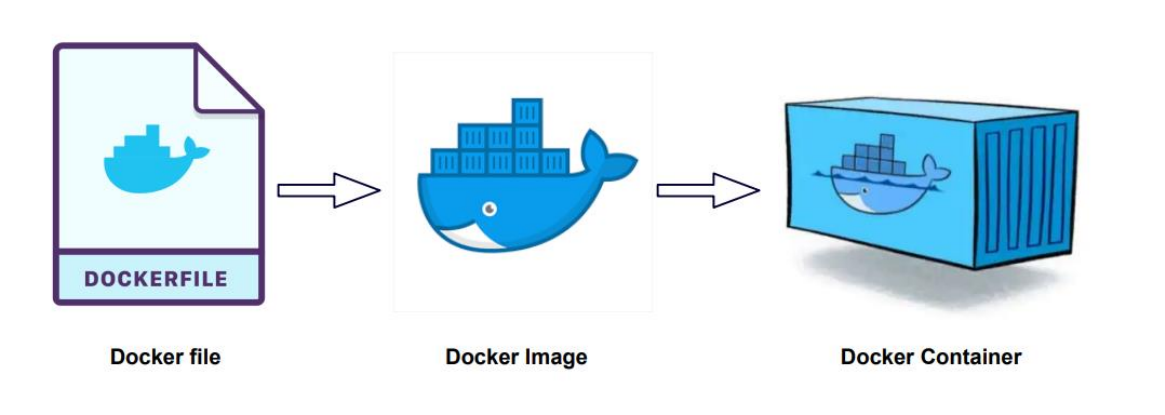


Docker Commands

Command	Usage	Example
docker run	Start a new container	`docker run image_name`
docker build	Build a Docker image from a Dockerfile	`docker build –t image_name .`
docker pull	Pull an image or repository from Docker registry	`docker pull ubuntu`
docker push	Push a local Docker image to a registry	`docker push my_username/image_name`
docker images	List all the Docker images stored locally	`docker images`
docker ps	Shows all running containers. Flag -a shows all containers (running & stopped).	`docker ps -a`
docker stop	Stops one or more running containers	`docker stop container_name`
docker rm	Remove a specific stopped container	`docker rm container_id`
docker rmi	Remove one or more Docker images	`docker rmi image_name`

Example #1

- **Let's run MySQL on your local machine using Docker…**



Step #1

1. Create the Dockerfile

Dockerfile

```
1  FROM mysql:latest
2  #
3  # set root password for local execution:
4  #
5  ENV MYSQL_ROOT_PASSWORD=abc123
6  #
7  # NOTE: changing to port 3307 since I already have MySQL installed
8  # and running locally on its own. So this docker image is a second
9  # version.
10 #
11 ENV MYSQL_TCP_PORT=3307
12 #
13 # expose the port needed to connect to MySQL server:
14 #
15 EXPOSE 3307
16 #
17 # start server when container runs:
18 #
19 CMD ["mysqld"]
20
```

Step #2

2. Build the docker image “mysql”

-t *tag the image so easier to refer to*

. *refer to Dockerfile in current directory*

```
docker build -t mysql .

docker images
```

Step #3

3. Run the docker image => docker container

-d detach from terminal window to run in background

-p map local port to container port

--name container so easier to refer to when running

--rm to remove container when it stops

```
docker run -d –p 3307:3307 --name mysql --rm mysql

docker ps
```

Step #4

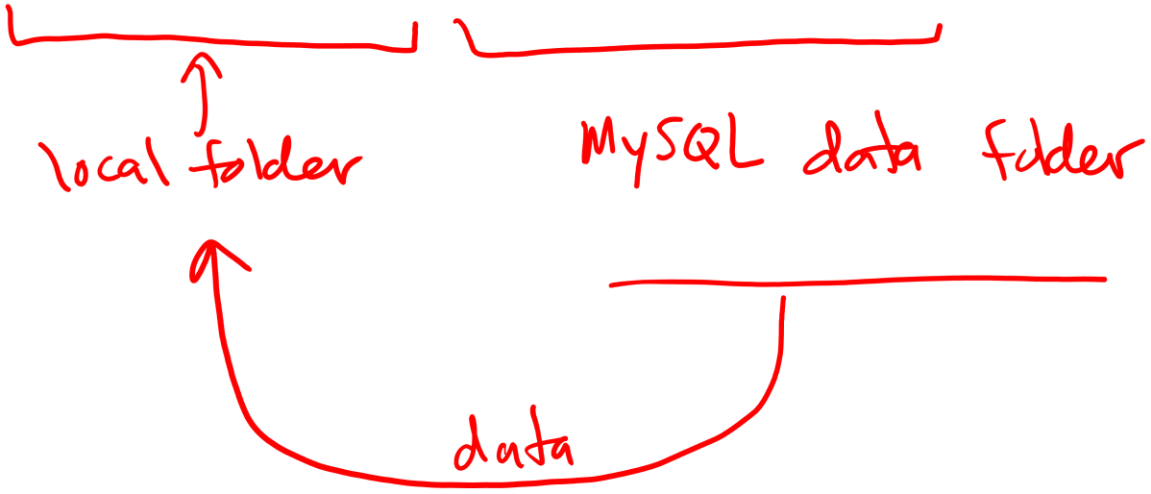
4. Did it work? Let's interact using cmd-line tool…

```
docker exec -it mysql bash

mysql –u root –p
abc123

show databases;

exit
exit

docker stop mysql
docker ps
```

Persisting data in the database?

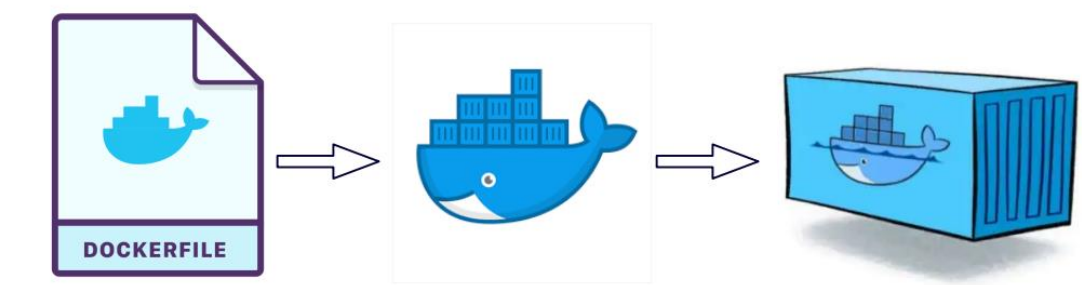
- **By default, docker images are stateless --- you do not save data within the docker image**
- **Instead, you mount a local folder ("volume") and save the data to your local machine with -v option**

```
docker run -d -v /data/mysql:/var/lib/mysql ...
```



Example #2

- **Let's implement a Lambda function in a Docker container to retrieve movies from MovieLens DB running in AWS…**



Step #1

1. Create the Dockerfile

Dockerfile

```
1  FROM public.ecr.aws/lambda/python:3.12
2
3  COPY *.py ${LAMBDA_TASK_ROOT}/
4  COPY *.ini ${LAMBDA_TASK_ROOT}/
5
6  RUN pip3 install pymysql
7
8  CMD [ "lambda_function.lambda_handler" ]
```

Step #2

2. Build the docker image “lambda-movies”

```
docker build -t lambda-movies .

docker images
```

Step #3

3. Run the docker image => docker container

-d option is NOT specified so we can see output from lambda

```
docker run -p 8080:8080 --rm lambda-movies
```

Step #4

4. Test locally using POSTMAN desktop app

POST

http://localhost:8080/2015-03-31/functions/function/invocations

BODY raw JSON

```
{
    "pathParameters":
    {
        "limit":  5,
        "offset": 50
    }
}
```


```
{"statusCode": 200, "body": "[[86, \"2006-02-10T00:00:00\", 105, \"de\", 6000000.0, 0.0, \"The
    Elementary Particles\"], [87, \"1984-05-23T00:00:00\", 118, \"en\", 28000000.0, 333000000.0,
    \"Indiana Jones and the Temple of Doom\"], [88, \"1987-08-21T00:00:00\", 100, \"en\", 6000000.0,
    213954000.0, \"Dirty Dancing\"], [89, \"1989-05-24T00:00:00\", 127, \"en\", 48000000.0,
    474172000.0, \"Indiana Jones and the Last Crusade\"], [90, \"1984-11-29T00:00:00\", 105, \"en\",
    15000000.0, 316360000.0, \"Beverly Hills Cop\"]]"}
```

Step #5

5. **Deploy to AWS ECR, create lambda function from ECR image, test within Lambda console…**

Push to ECR, load into Lambda, test

- Switch over to AWS:
  1. *Search for ECR, open ECR console*
  2. *Create a repository with same name as container image*
  3. *Select repository, view "Push command"*
  4. *Open terminal window on your laptop (AWS CLI must be configured --- we did this in project 01)*
  5. *Execute each of the "push" commands to push image --- skip the command that builds the image, we did that*
  6. *Switch to Lambda console*
  7. *Create new lambda function*
  8. *Select "Create from Image", and select image from ECR drop-down*
  9. *Name function and create*
  10. *Configuration tab: increase timeout*
  11. *Deploy*
  12. *Test --- logs are available from CloudWatch*

That's it, thank you!