synch_05

UCB MIDS W205 Summer 2018 - Kevin Crook's agenda for Synchronous Session #5

Update docker images (before class)

Run these command in your droplet (but **NOT** in a docker container):

```
docker pull midsw205/base:latest
docker pull redis:latest
docker pull confluentinc/cp-zookeeper:latest
docker pull confluentinc/cp-kafka:latest
```

Update the course-content repo in your docker container in your droplet (before class)

See instructions in previous synchronous sessions.

Assignment 5

Go over how to start a Jupyter Notebook for Python 3 in a docker container in your droplet and run the assignment 5 jupyter notebook.

Create a docker cluster with a container for redis (key-value data store) and a container for mids base ubuntu

Outside of docker, create a directory called redis that will hold our yml file for this docker cluster:

```
mkdir ~/w205/redis
```

Change to redis directory:

```
cd ~/w205/redis
```

Use vi to create a docker-compose.yml file with the content below.

```
vi docker-compose.yml
```

```
---
version: '2'
services:
  redis:
    image: redis:latest
    expose:
      - "6379"
    extra_hosts:
      - "moby:127.0.0.1"

  mids:
    image: midsw205/base:latest
```

```
    stdin_open: true
    tty: true
    extra_hosts:
      - "moby:127.0.0.1"
```

Startup the docker cluster:

```
docker-compose up -d
```

Verify the docker cluster is running properly:

```
docker-compose ps
```

Output should be similar to this:

```
           Name                         Command               State    Ports
-------------------------------------------------------------------------------
redisexample_midsbase_1   /bin/bash                          Up      8888/tcp
redisexample_redis_1      docker-entrypoint.sh redis ...     Up      6379/tcp
```

Look at the logs of the redis container in the cluster:

```
docker-compose logs redis
```

The last of the output should be similar to this to let us know that redis it ready to accept connections:

```
Ready to accept connections
```

We are running our cluster containers "headless", that is without an interactive connection. What if we want to interact with a container, such as running a bash shell with standard input and standard output piped to and from our command prompt? We can exec processes in the containter, including bash shells:

```
docker-compose exec mids bash
```

We now have a bash shell running in our mids container inside our docker cluster.

We now want to run ipython (interactive python) which is a command line utility similar to jupyter notebook. It actually predates jupyter notebook and was the basis on which jupter notebooks was designed and built.

```
ipython
```

In our ipython command line, we will try out the redis database. Redis is a key-value pair, NoSQL database. Check TCP port 6379 against the specification in the yml file.

```
import redis
r = redis.Redis(host='redis', port='6379')
r.keys()
exit
```

Exit our bash shell which was exec'd in the container.

```
exit
```

Tear down our docker cluster

```
docker-compose down
```

Verify the docker cluster is down

```
docker-compose ps
```

```
docker ps -a
```

Adding Jupyter Notebooks to the mids container in our docker cluster

We will change our docker-compose.yml file to allow jupyter notebooks to be run in the mids container. We will expose the correct ports to the outside world so we can connect from a Chrome browser on our laptops to jupyter notebook running in our mids container in our droplet.

Using vi, edit our docker-compose.yml file to match the following:

```
---
version: '2'
services:
  redis:
    image: redis:latest
    expose:
      - "6379"
    extra_hosts:
      - "moby:127.0.0.1"

  mids:
    image: midsw205/base:latest
    stdin_open: true
    tty: true
    expose:
      - "8888"
    ports:
      - "8888:8888"
    extra_hosts:
      - "moby:127.0.0.1"
```

Bring up our docker cluster, "detached" or "headless".

```
docker-compose up -d
```

Exec a jupyter notebook in our mids container in our docker cluster. Compare port 888 to our specfication in the yml file.

```
docker-compose exec mids jupyter notebook --no-browser --port 8888 --ip 0.0.0.0 --allow-root
```

On your laptop, open a Chrome browser and connect to jupyter notebook running in the mids container of your docker cluster in your droplet. Please note that outgoing TCP port 8888 must not be blocked between your laptop and your droplet.

```
http://xxx.xxx.xxx.xxx:8888/?token=xxx
```

Exit and logout of jupyter notebook in your browser.

Tear down your docker cluster.

```
docker-compose down
```

Verify the docker cluster is down. (This is the last time I'll put verify instruction from now on, always verify that a docker cluster tears down properly)

```
docker-compose ps
```

```
docker ps -a
```

Breakout - students discuss Query Project

We will go into breakout for students to discuss the Query Project. If students are having trouble with getting their cluster from the previous to work, or their jupyter notebook to work, the instructor can work with them one-on-one to get caught up.

Automatically start Jupyter Notebook in the mids container of your docker cluster

Let's change our mids container to automatically start the jupyter notebook so we don't have to manually start it each time by updating the docker-compose.yml file as follows:

```
---
version: '2'
services:
  redis:
    image: redis:latest
    expose:
      - "6379"
    extra_hosts:
      - "moby:127.0.0.1"

  mids:
    image: midsw205/base:latest
    stdin_open: true
    tty: true
    expose:
      - "8888"
    ports:
      - "8888:8888"
    extra_hosts:
      - "moby:127.0.0.1"
    command: jupyter notebook --no-browser --port 8888 --ip 0.0.0.0 --allow-root
```

Start up our docker cluster:

```
docker-compose up -d
```

Last time, we manually started jupyter notebook from the command line, so we were able to see the url with token string. Since we had docker automatically start this for us, we will need to pull it from the logs. This is very common when running headless, important information is written to log files instead of the command line.

```
docker-compose logs mids
```

As we did previously, open a Chrome browser on your laptop and surf the jupyter notebook:

```
http://xxx.xxx.xxx.xxx:8888/?token=xxx
```

Inside jupyter notebook, create a new python 3 notebook, and create and execute code cells.

First we will import the redis module and connect from our mids container to our redis container using TCP port 6379 over the virtual network. Note that we can use the container name in the yml file as our hostname.

```
import redis
r = redis.Redis(host='redis', port='6379')
r.keys()
```

Add some key-value pairs to our redis NoSQL database. Try some more key value pairs. What happens if we have an existing key value pair and try to add a new key value pair with the same key? Try it.

```
r.set('key1', 'value1')
value = r.get('key1')
print(value)
```

Tear down our cluster:

```
docker-compose down
```

Load and query real data into our redis key-value NoSQL database

We will load and query a subset of our bike share data into redis.

Download our bike share trips subset data into a csv file.

```
cd ~/w205/
curl -L -o trips.csv https://goo.gl/XygbZp
```

We will now add our volume mount to our docker-compose.yml file as follows to mount /home/science/w205 in our droplet to /w205 in the mids container of our docker cluster:

```
---
version: '2'
services:
  redis:
    image: redis:latest
    expose:
      - "6379"
    extra_hosts:
      - "moby:127.0.0.1"

  mids:
    image: midsw205/base:latest
    stdin_open: true
    tty: true
    volumes:
      - /home/science/w205:/w205
    expose:
      - "8888"
    ports:
      - "8888:8888"
    extra_hosts:
      - "moby:127.0.0.1"
    command: jupyter notebook --no-browser --port 8888 --ip 0.0.0.0 --allow-root
```

Startup our docker cluster:

```
docker-compose up -d
```

Get the token to run jupyter notebook from our mids container logs:

```
docker-compose logs mids
```

As we did previously, open a Chrome browser on your laptop and surf the jupyter notebook:

```
http://xxx.xxx.xxx.xxx:8888/?token=xxx
```

Inside jupyter notebook, create a new python 3 notebook, and create and execute code cells.

```
import redis
import pandas as pd
```

Using pandas, read in our trips file in csv format into a pandas data frame. Sort by end date.

```
trips=pd.read_csv('trips.csv')

date_sorted_trips = trips.sort_values(by='end_date')

date_sorted_trips.head()
```

Loop through the data frame and pretty print the data.

```
for trip in date_sorted_trips.itertuples():
  print(trip.end_date, '', trip.bike_number, '', trip.end_station_name)
```

Connect to our redis container and look at our keys(). What should their value be at this point?

```
current_bike_locations = redis.Redis(host='redis', port='6379')
current_bike_locations.keys()
```

Loop through the data fram and insert each record into the redis database. Since keys most be unique, what happens when we have an existing key value pair and we have a second key value pair with the same key?

```
for trip in date_sorted_trips.itertuples():
  current_bike_locations.set(trip.bike_number, trip.end_station_name)

current_bike_locations.keys()
```

Look up the value for a specific bike. In this case 92. Play around with different bikes to see if you can verify the overwrite.

```
current_bike_locations.get('92')
```

Tear down the cluster

```
docker-compose down
```

Breakout - discuss key value NoSQL database

What kinds of applications would be good for key value NoSQL database?

- If I had a social media website, and to bring up a user's home page, and it requires querying 30 or more tables in a tradional SQL based relational database. How could I use a key value NoSQL database to speed this up? Speed up would the advantage, but what would be the big disadvantage?
- If I had a credit card company with 200 million customers, with customer data stored in a tradition SQL based relational database. Suppose I created a key value store where the key is the customer id and the value has all information about the customer and their transactions.

- How would that make analytics on a specific customer easier?
- Would we see similar problems to the social media in regard to staleness of data? How would staleness of data differ in an analytics environment versus a production transcation processing environment?
- Suppose I wanted to do analytics on transactions across all customers, what would be the performance? How could I fix it?