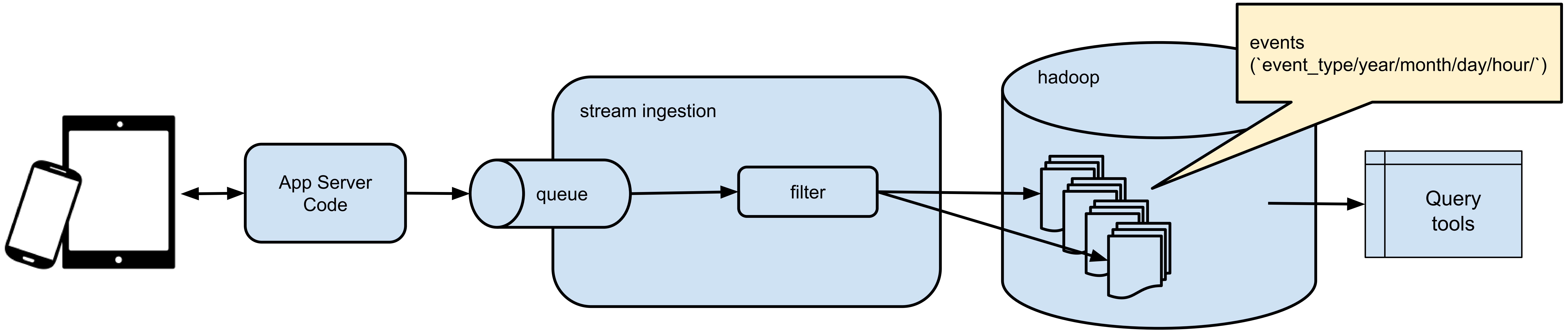
Fundamentals of Data Engineering

Week 12 - sync session

datascience@berkeley

Project Review

- Review your Project 3
- Get ready to share
- `docker pull midsw205/base:latest`
- `git pull` in `~/w205/course-content`


App Server Code
queue
stream ingestion
filter
hadoop
events (`event_type/year/month/day/hour/`)
Query tools

Flask-Kafka-Spark-Hadoop-Presto Part I

Setup

Set up directory, get docker-compose

```
mkdir ~/w205/full-stack/
cd ~/w205/full-stack
cp ~/w205/course-content/12-Querying-Data-II/docker-compose.yml .
cp ~/w205/course-content/12-Querying-Data-II/*.py .
```

The `docker-compose.yml`

Create a `docker-compose.yml` with the following

```yaml
---
version: '2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 32181
      ZOOKEEPER_TICK_TIME: 2000
    expose:
      - "2181"
      - "2888"
      - "32181"
      - "3888"
    extra_hosts:
      - "moby:127.0.0.1"

  kafka:
```

and with no need for a datafile on this one.

Spin up the cluster

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

Create a topic `events`

```
docker-compose exec kafka kafka-topics --create --topic events --part
```

Web-app

- Take our instrumented web-app from before
  `~/w205/full-stack/game_api.py`

```
#!/usr/bin/env python
import json
from kafka import KafkaProducer
from flask import Flask, request

app = Flask(__name__)
producer = KafkaProducer(bootstrap_servers='kafka:29092')


def log_to_kafka(topic, event):
    event.update(request.headers)
    producer.send(topic, json.dumps(event).encode())


@app.route("/")
def default_response():
```

run flask

```
docker-compose exec mids \
  env FLASK_APP=/w205/full-stack/game_api.py \
  flask run --host 0.0.0.0
```

Set up to watch kafka

```
docker-compose exec mids \
  kafkacat -C -b kafka:29092 -t events -o beginning
```

Apache Bench to generate data

```
docker-compose exec mids \
  ab \
    -n 10 \
    -H "Host: user1.comcast.com" \
    http://localhost:5000/
```

```
docker-compose exec mids \
  ab \
    -n 10 \
    -H "Host: user1.comcast.com" \
    http://localhost:5000/purchase_a_sword
```

```
docker-compose exec mids \
  ab \
    -n 10 \
    -H "Host: user2.att.com" \
    http://localhost:5000/
```

```
docker-compose exec mids \
  ab \
    -n 10 \
    -H "Host: user2.att.com" \
    http://localhost:5000/purchase_a_sword
```

More Spark

last time

~/w205/spark-from-files/separate_events.py

```
#!/usr/bin/env python
"""Extract events from kafka and write them to hdfs
"""
import json
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import udf


@udf('string')
def munge_event(event_as_json):
    event = json.loads(event_as_json)
    event['Host'] = "moe"
    event['Cache-Control'] = "no-cache"
    return json.dumps(event)
```

which we ran

```
docker-compose exec spark \
  spark-submit /w205/spark-from-files/separate_events.py
```

what if different event types have different schema?

~/w205/full-stack/just_filtering.py

```python
#!/usr/bin/env python
"""Extract events from kafka and write them to hdfs
"""
import json
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import udf


@udf('boolean')
def is_purchase(event_as_json):
    event = json.loads(event_as_json)
    if event['event_type'] == 'purchase_sword':
        return True
    return False
```

run this

```
docker-compose exec spark \
  spark-submit /w205/full-stack/just_filtering.py
```

we can play with this

add a new event type to the flask app...

```python
@app.route("/purchase_a_knife")
def purchase_a_knife():
    purchase_knife_event = {'event_type': 'purchase_knife',
                            'description': 'very sharp knife'}
    log_to_kafka('events', purchase_knife_event)
    return "Knife Purchased!\n"
```

Write Events

full-stack/filtered_writes.py

```python
#!/usr/bin/env python
"""Extract events from kafka and write them to hdfs
"""
import json
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import udf


@udf('boolean')
def is_purchase(event_as_json):
    event = json.loads(event_as_json)
    if event['event_type'] == 'purchase_sword':
        return True
    return False
```

run this

```
docker-compose exec spark \
  spark-submit /w205/full-stack/filtered_writes.py
```

should see purchases in hdfs

```
docker-compose exec cloudera hadoop fs -ls /tmp/purchases/
```

Queries From Spark

spin up a notebook

```
docker-compose exec spark \
  env \
    PYSPARK_DRIVER_PYTHON=jupyter \
    PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser --port 8888 --i
  pyspark
```

New python3 notebook and play

```
purchases = spark.read.parquet('/tmp/purchases')
purchases.show()
purchases.registerTempTable('purchases')
purchases_by_example2 = spark.sql("select * from purchases where host
purchases_by_example2.show()
newdf = purchases_by_example2.toPandas()
newdf.describe()
```

down

```
docker-compose down
```

SecureShell (SSH)

remote terminal connections

```
ssh science@xxx.xxx.xxx.xxx
```

copying files

On your laptop, run

```
scp some_file science@xxx.xxx.xxx.xxx:
```

or

```
scp some_file science@xxx.xxx.xxx.xxx:/tmp/
```

On your laptop, run

```
scp science@xxx.xxx.xxx.xxx:~/w205/a_file.py .
```

keys

generate a keypair

```
ssh-keygen -t rsa -b 2048
```

datascience@berkeley

this creates

a public key

```
~/.ssh/id_rsa.pub
```

and a secret key

```
~/.ssh/id_rsa
```

add your pubkey to github

verify your pubkey is on github

```
curl https://github.com/<your-gh-id>.keys
```

(note the `https`!)

add pubkey to instance

On your cloud instance, run

```
ssh-import-id-gh <your-gh-id>
```

you should see something like

```
science@smmm-mmm-1:~$ ssh-import-id-gh mmm
2018-04-02 18:09:29,091 INFO Starting new HTTPS connection (1): api.
2018-04-02 18:09:29,285 INFO Authorized key ['4096', 'SHA256:51JGHgl
2018-04-02 18:09:29,287 INFO [1] SSH keys [Authorized]
```

now no more passwords

```
ssh science@xxx.xxx.xxx.xxx
```

Summary

Berkeley SCHOOL OF INFORMATION