Fundamentals of Data Engineering

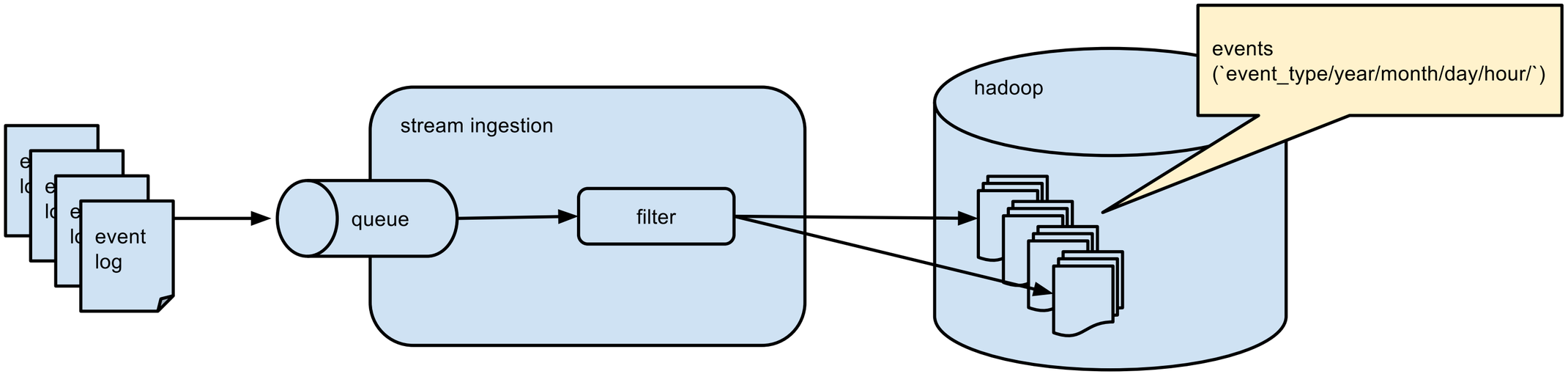
Week 09 - sync session

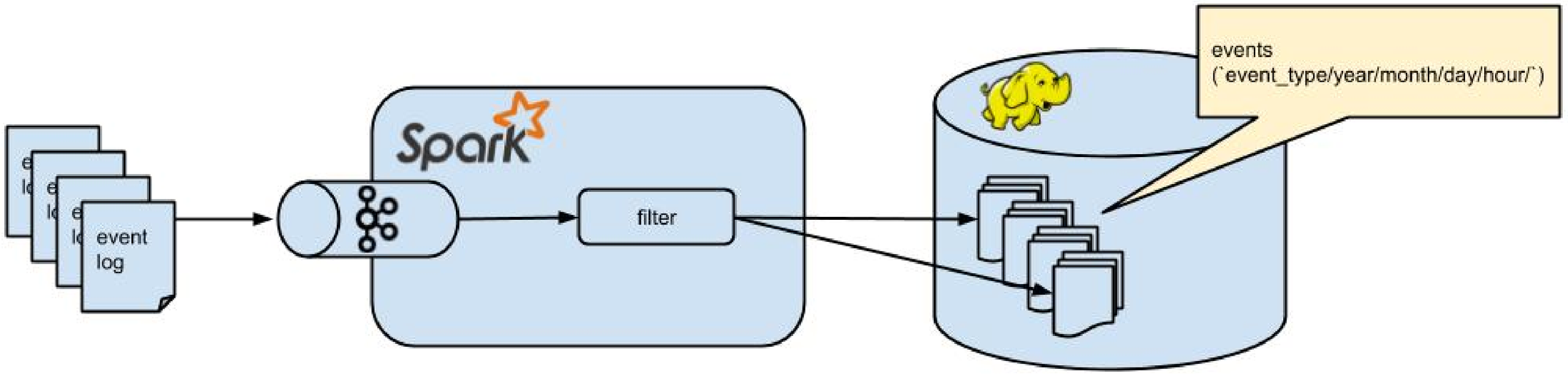
datascience@berkeley

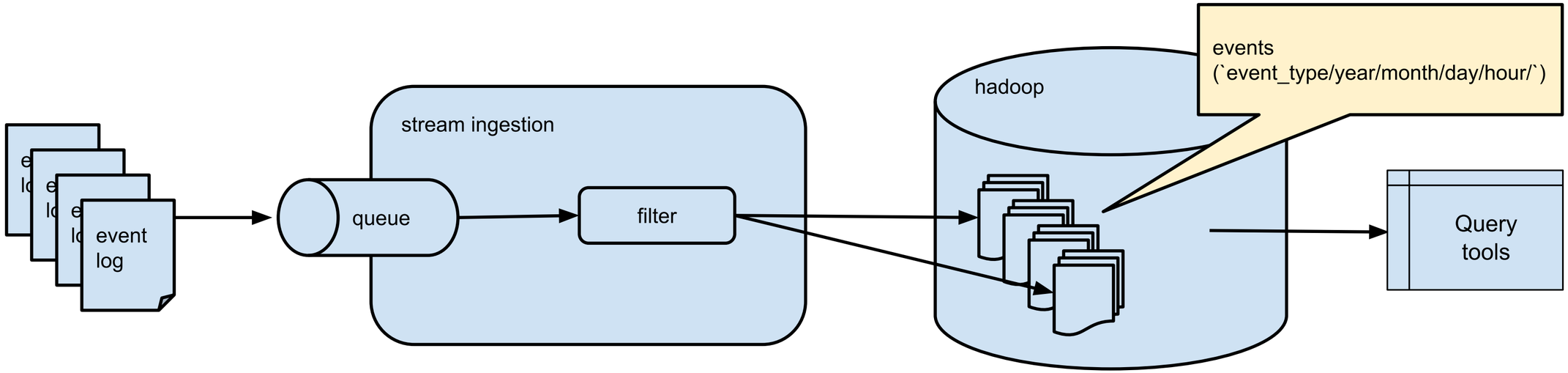
Project Review

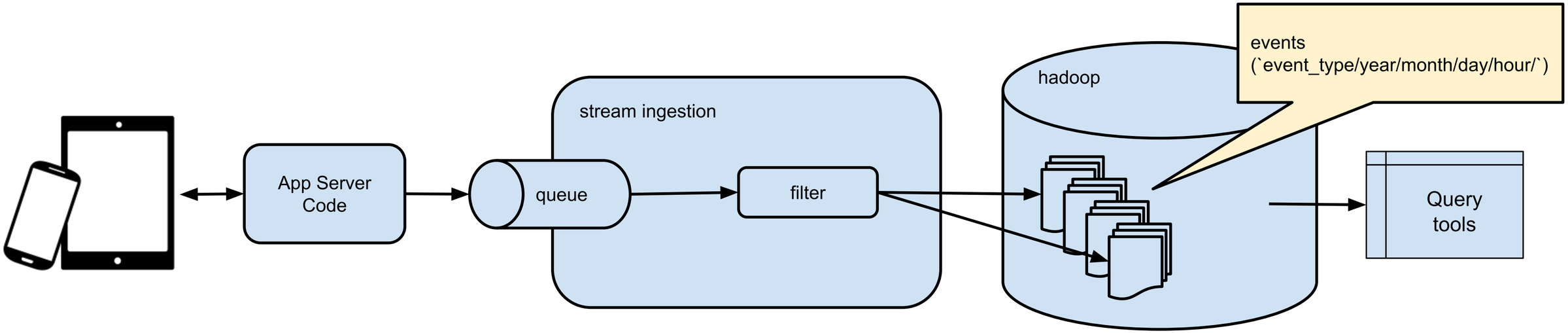
- Review your Project 2
- Get ready to share

Project Transition


event
log
queue
stream ingestion
filter
hadoop
events
(`event_type/year/month/day/hour/`)


event
log
Spark
filter
events
(`event_type/year/month/day/hour/`)


event
log
stream ingestion
queue
filter
hadoop
events
(`event_type/year/month/day/hour/`)
Query
tools


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

Project 3 Setup

- You're a data scientist at a game development company.
- Your latest mobile game has two events you're interested in tracking:
- `buy a sword` & `join guild`...
- Each has metadata

Project 3 Task

- Your task: instrument your API server to catch and analyze these two event types.

Project 3 options

Flask with Kafka

Setup

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

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

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
```

Spin up the cluster

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

Create a topic `events`

```
docker-compose exec kafka \
  kafka-topics \
    --create \
    --topic events \
    --partitions 1 \
    --replication-factor 1 \
    --if-not-exists \
    --zookeeper zookeeper:32181
```

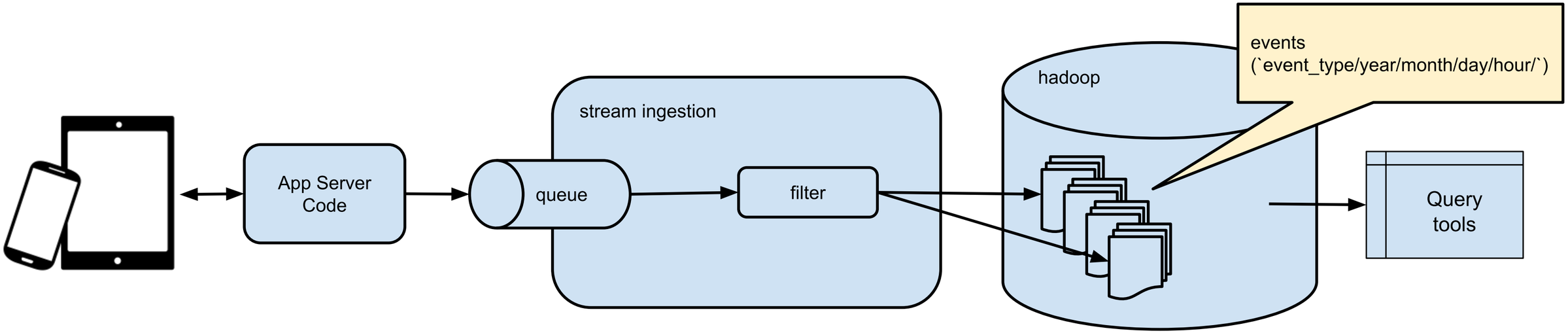
Should show

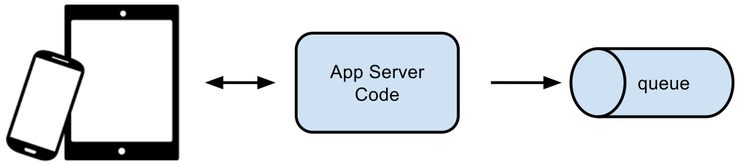
```
Created topic "events".
```

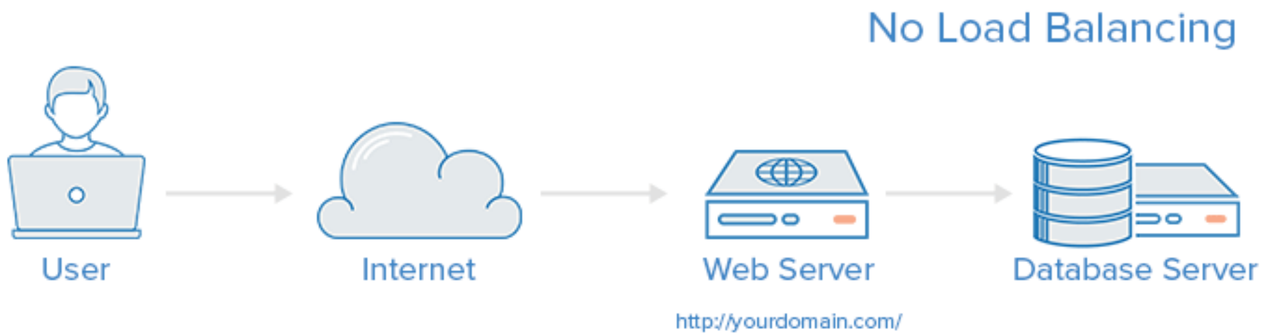
Let’s create a web-based application

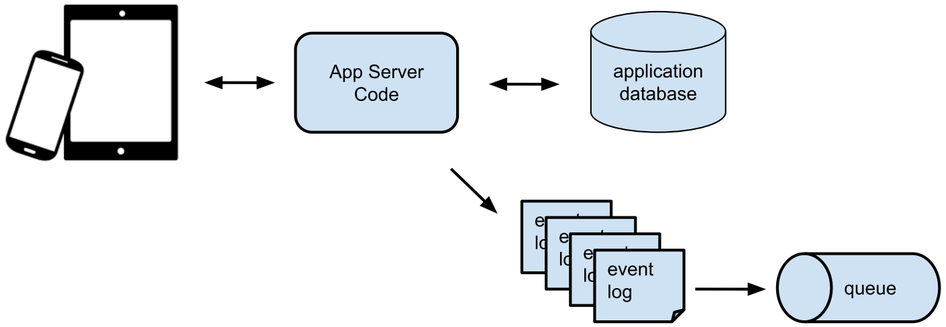
An example scenario

- You’re a mobile game developer. During gameplay, your users perform various actions such as
- purchase a sword
- purchase a knife
- join a guild
- etc
- To process these actions, your mobile app makes API calls to a web-based API-server.


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


App Server
Code
queue


No Load Balancing
User
Internet
Web Server
Database Server
http://yourdomain.com/


App Server
Code
application
database
event
log
queue

An API server - usual case

- User actions map to API endpoints

```
POST /purchase
```

To keep it simple

- Our actions will map to single API calls

```
GET /purchase_a_sword
```

Flask

- Use the python `flask` library to write our simple API server.

```
#!/usr/bin/env python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def default_response():
    return "This is the default response!"

@app.route("/purchase_a_sword")
def purchase_sword():
    # business logic to purchase sword
    return "Sword Purchased!"
```

- Save this as `~/w205/flask-with-kafka/basic_game_api.py` and run it via

```
docker-compose exec mids env FLASK_APP=/w205/flask-with-kafka/basic_g
```

Test it out

```
docker-compose exec mids curl http://localhost:5000/
```

```
docker-compose exec mids curl http://localhost:5000/purchase_a_sword
```

Stop flask

-Kill flask with `ctrl-c`.

Generate events from our webapp

- Let’s add kafka into the mix

```
#!/usr/bin/env python
from kafka import KafkaProducer
from flask import Flask
app = Flask(__name__)
event_logger = KafkaProducer(bootstrap_servers='kafka:29092')
events_topic = 'events'

@app.route("/")
def default_response():
    event_logger.send(events_topic, 'default'.encode())
    return "This is the default response!"

@app.route("/purchase_a_sword")
def purchase_sword():
    # business logic to purchase sword
    # log event to kafka
```

Run that

```
docker-compose exec mids env FLASK_APP=/w205/flask-with-kafka/game_ap
```

Test it out

- Generate events

```
docker-compose exec mids curl http://localhost:5000/
```

```
docker-compose exec mids curl http://localhost:5000/purchase_a_sword
```

read from kafka

- Use kafkacat to consume events from the `events` topic

```
docker-compose exec mids bash -c "kafkacat -C -b kafka:29092 -t event
```

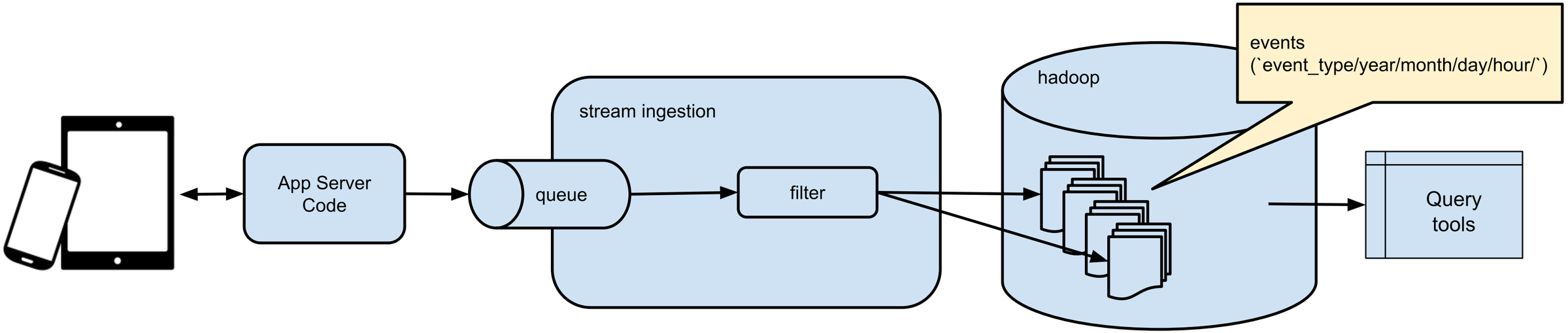
down

```
docker-compose down
```

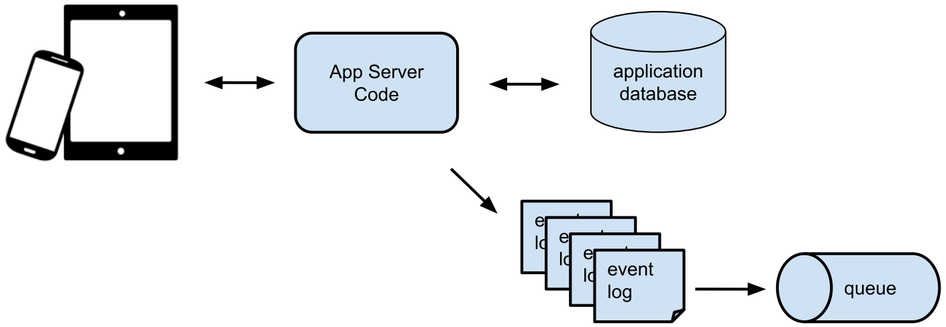
Week 9 Videos

- Basics of brokers, producers, consumers, topics
- Context for using Kafka in a pipeline in reality(aka, at scale, in production)

Summary


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

Summary



Berkeley SCHOOL OF INFORMATION