BUILDING APACHE SPARK APPLICATION PIPELINES FOR THE KUBERNETES ECOSYSTEM

Michael McCune

14 November 2016

INTRODUCTION

A little about me

- Embedded to Orchestration
- Red Hat emerging technologies
- OpenStack Sahara
- Oshinko project for OpenShift

OVERVIEW

Building Application Pipelines

Case Study: Ophicleide

Demonstration

Lessons Learned

Next Steps

INSPIRATION

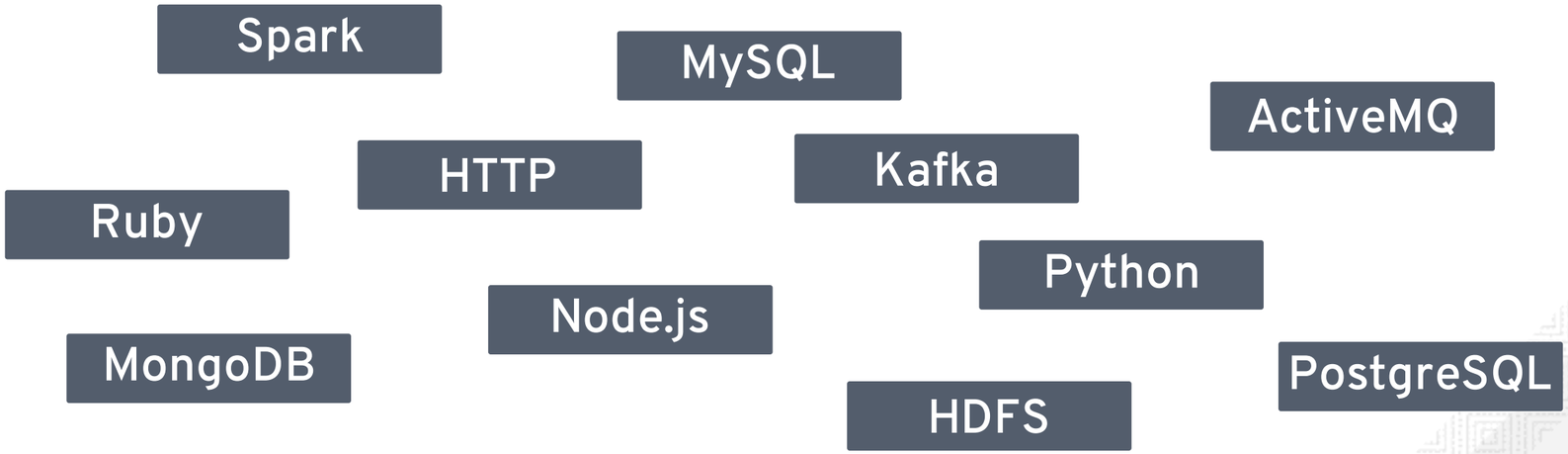
Larger themes

- Developer empowerment
- Improved collaboration
- Operational freedom

CLOUD APPLICATIONS

What are we talking about?

- Multiple disparate components
- Require deployment flexibility
- Challenging to debug



PLANNING

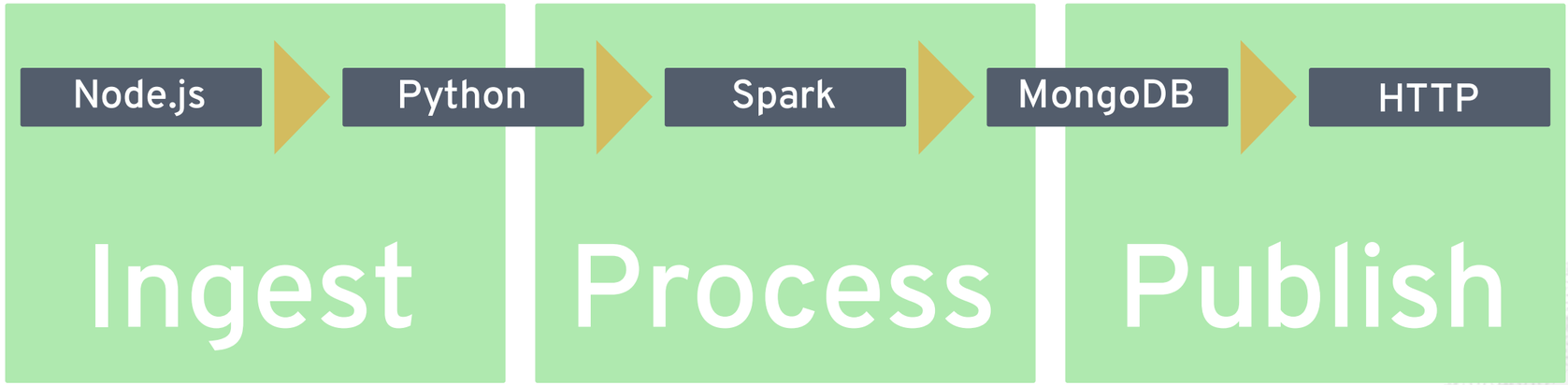
Before you begin engineering

- Identify moving pieces
- Storyboard the data flow
- Visualize success *and* failure

PLANNING

Insightful analytics

- What dataset?
- How to process?
- Where are the results?



BUILDING

Decompose application components

- Natural breakpoints
- Build for modularity
- Stateless versus stateful

BUILDING

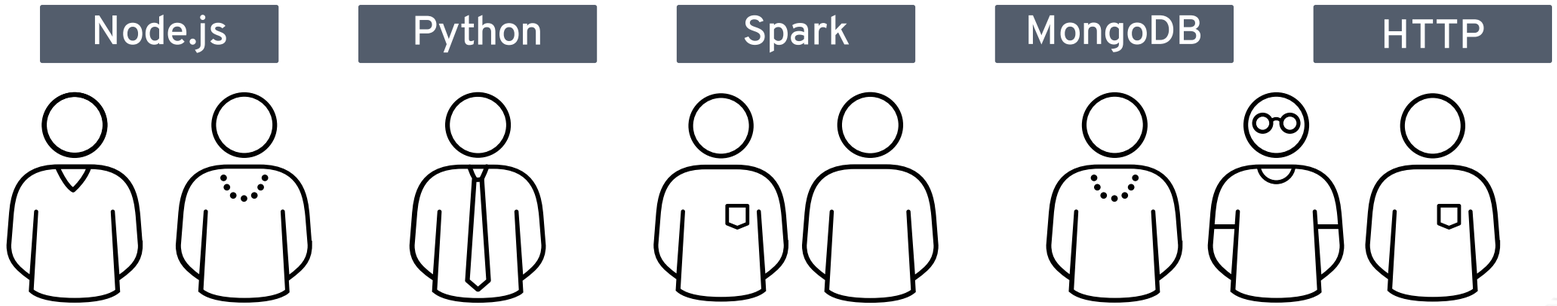
Focus on the communication

- Coordinate in the middle
- Network resiliency
- Kubernetes DNS

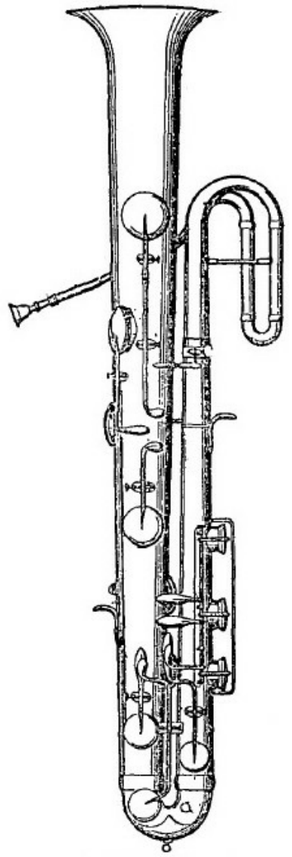
COLLABORATING

Building as a team

- The right tools
- Modular projects
- Iterative improvements
- Coordinating actions



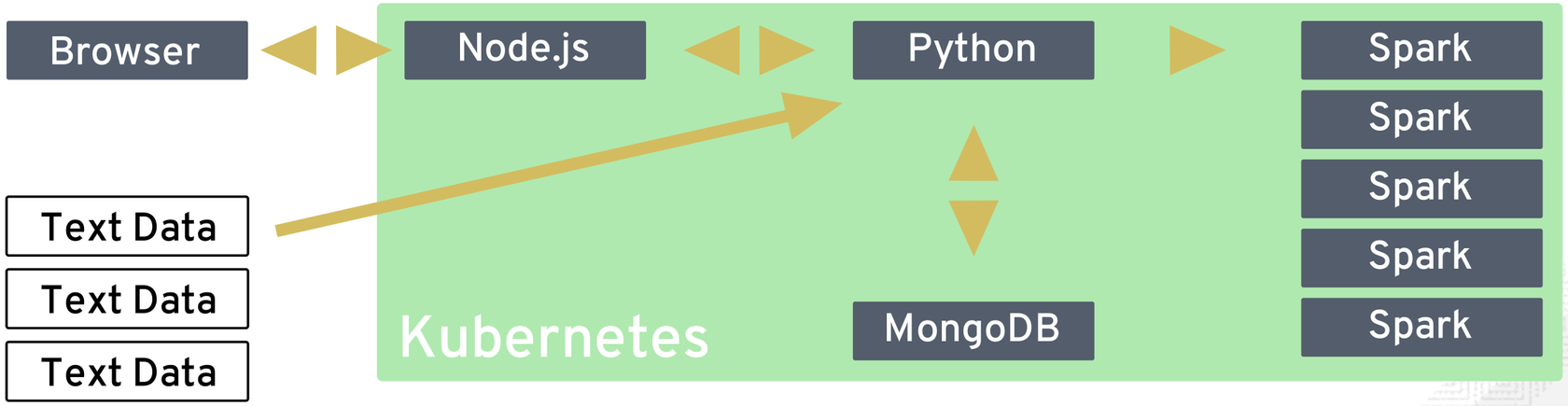
CASE STUDY: OPHICLEIDE



CASE STUDY: OPHICLEIDE

What does it do?

- Word2Vec models
- HTTP available data
- Similarity queries



CASE STUDY: OPHICLEIDE

Building blocks

- Apache Spark
- Word2Vec
- Kubernetes
- OpenShift
- Node.js
- Flask
- MongoDB
- OpenAPI

DEEP DIVE

OpenAPI

- Schema for REST APIs
- Wealth of tooling
- Central discussion point

OPENAPI

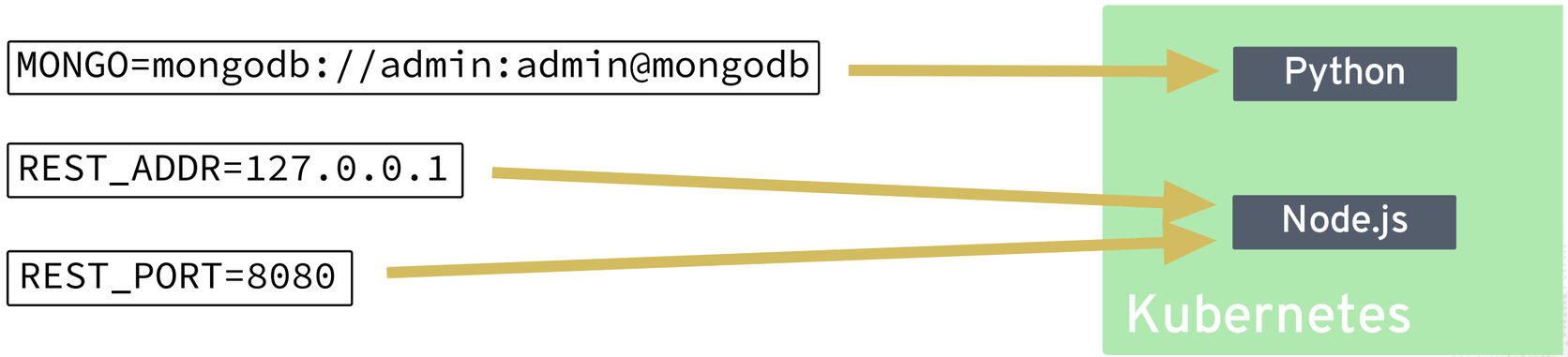
```
paths:
  /:
    get:
      description: |-
        Returns information about the server version
      responses:
        "200":
          description: |-
            Valid server info response
          schema:
```

```
app = connexion.App(__name__, specification_dir='./swagger/')
app.add_api('swagger.yaml',
            arguments={'title':
                       'The REST API for the Ophicleide '
                       'Word2Vec server'})
app.run(port=8080)
```

DEEP DIVE

Configuration Data

- What is needed?
- How to deliver?



CONFIGURATION DATA

```
spec:
  containers:
    - name: ${WEBNAME}
      image: ${WEBIMAGE}
      env:
        - name: OPH_TRAINING_ADDR
          value: ${OPH_ADDR}
        - name: OPH_TRAINING_PORT
          value: ${OPH_PORT}
        - name: OPH_WEB_PORT
          value: "8081"
      ports:
        - containerPort: 8081
          protocol: TCP
```

CONFIGURATION DATA

```
var training_addr = process.env.OPH_TRAINING_ADDR || '127.0.0.1';
var training_port = process.env.OPH_TRAINING_PORT || '8080';
var web_port = process.env.OPH_WEB_PORT || 8080;

app.get("/api/models", function(req, res) {
  var url = `http://${training_addr}:${training_port}/models`;
  request.get(url).pipe(res);
});

app.get("/api/queries", function(req, res) {
  var url = `http://${training_addr}:${training_port}/queries`;
  request.get(url).pipe(res);
});

app.listen(ophicleide_web_port, function() {
  console.log(`ophicleide-web listening on ${web_port}`);
});
```

SECRETS

Not used in Ophicleide, but worth mentioning

```
volumes:
    - name: mongo-secret-volume
      secret:
        secretName: mongo-secret
containers:
  - name: shiny-squirrel
    image: elmiko/shiny_squirrel
    args: ["mongodb"]
    volumeMounts:
      - name: mongo-secret-volume
        mountPath: /etc/mongo-secret
        readOnly: true
```

SECRETS

Each secret exposed as a file in the container

```
MONGO_USER=$(cat /etc/mongo-secret/username)
MONGO_PASS=$(cat /etc/mongo-secret/password)

/usr/bin/python /opt/shiny_squirrel/shiny_squirrel.py \
    --mongo \
    mongodb://${MONGO_USER}:${MONGO_PASS}@${MONGO_HOST_PORT}
```

DEEP DIVE

Spark processing

- Read text from URL
- Split words
- Create vectors

SPARK PROCESSING

```
def workloop(master, inq, outq, dburl):
    sconf = SparkConf().setAppName(
        "ophicleide-worker").setMaster(master)
    sc = SparkContext(conf=sconf)

    if dburl is not None:
        db = pymongo.MongoClient(dburl).ophicleide

    outq.put("ready")

    while True:
        job = inq.get()
        urls = job["urls"]
        mid = job["_id"]
        model = train(sc, urls)

        items = model.getVectors().items()
        words, vecs = zip(*[(w, list(v)) for w, v in items])
```

SPARK PROCESSING

```
def train(sc, urls):
    w2v = Word2Vec()
    rdds = reduce(lambda a, b: a.union(b),
                  [url2rdd(sc, url) for url in urls])
    return w2v.fit(rdds)
```

```
def url2rdd(sc, url):
    response = urlopen(url)
    corpus_bytes = response.read()
    text = str(
        corpus_bytes).replace("\\r", "\r").replace("\\n", "\n")
    rdd = sc.parallelize(text.split("\r\n\r\n"))
    rdd.map(lambda l: l.replace("\r\n", " ").split(" "))
    return rdd.map(lambda l: cleanstr(l).split(" "))
```

SPARK PROCESSING

```
def create_query(newQuery) -> str:
    mid = newQuery["model"]
    word = newQuery["word"]
    model = model_cache_find(mid)
    if model is None:
        msg = (("no trained model with ID %r available; " % mid)
               + "check /models to see when one is ready")
        return json_error("Not Found", 404, msg)
    else:
        # XXX
        w2v = model["w2v"]
        qid = uuid4()
        try:
            syns = w2v.findSynonyms(word, 5)
            q = {
                "_id": qid, "word": word, "results": syns,
                 "modelName": model["name"], "model": mid
            }
            (query_collection()).insert_one(q)
```

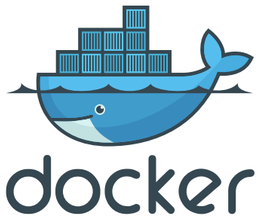
DEMONSTRATION

see a demo at https://vimeo.com/189710503

LESSONS LEARNED

Things that went smoothly

- OpenAPI
- Dockerfiles
- Kuberenetes templates



LESSONS LEARNED

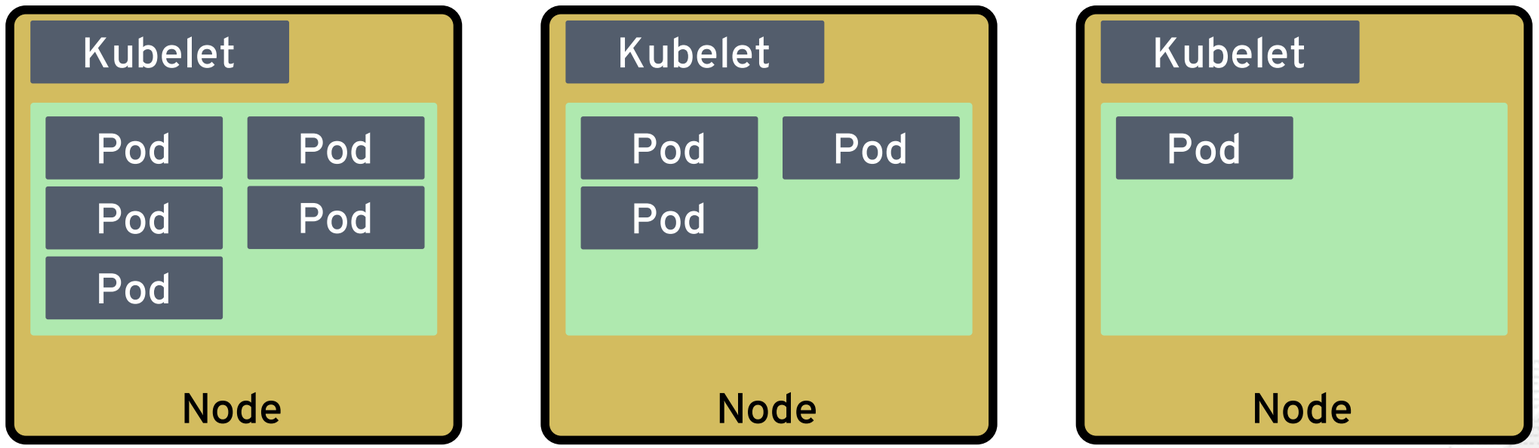
Things that require greater coordination

- API coordination
- Compute resources
- Persistent storage
- Spark configurations

LESSONS LEARNED

Compute resources

- CPU and memory constraints
- Label selectors



NEXT STEPS

Where to take this project?

- More Spark!
- Separate query service
- Development versus production

PROJECT LINKS

Ophicleide

- https://github.com/ophicleide

Apache Spark

- https://spark.apache.org

Kubernetes

- https://kubernetes.io

OpenShift

- https://openshift.org

THANKS!


elmiko


@FOSSjunkie

https://elmiko.github.io