

Access Services

Exposing the Results

Target == Dashboard == Bad Pipeline



- ☐ **Data Scientist would focus on the dashboard/report instead of content**
- ☐ **Breaks reusability of data**
- ☐ **Time wasted on learning viz instead of increasing accuracy (or velocity)**
- ☐ **Monolithic instead of service oriented**

Include intermediate Service

The data science pipeline has now to include the way the results can be consumed by the third parties (consumers, customers, ...)

The question is what are the results of a data science pipeline?

Actually, not only the model (machine learning) **needs to be exposed but also all the intermediate results that are worth considering persistent** (f.i. features creation, cleaning, ...)

Having services for those, say, views allows us to abstract the way they are created to the way they are used.

External scalability

The other clear advantage of services as part of the pipeline allows the team to separate the resources needed to serve the data to those necessary to compute them!

Example:

Let's say the dashboard is built directly using the streaming technology and the same cluster used to compute.

Then the computing resources will have to **also follow the consuming patterns.**

So, the more users the more computing power while the data may not have changed...

Horizontal Scalability

In the context of Big or Fast Data, it must be clear for everyone that views need to be created on the data for each use case
(or a small subgroup of use cases)

This gives us another incentive to introduce micro services into the architecture, each of them will have the responsibilities to expose one of these views.

So we will be able to scale independently each feature of the data science product that we were in charge to produce, as a team.

Access Service

Akka, again

HTTP server

**HTTP client/server
implementation using Akka as
the core engine**

**Providing DSLs for creating
routing, utils for un/marshalling
and more**

```
val route =  
  get {  
    path(Segment / Segment) { (pathSegment1, pathSegment2) =>  
      parameters("param1".as[Long], "param2".as[String]) {  
        (param1, param2) = complete()  
      }  
    } ~  
    path("path1") {  
      complete(...)  
    }  
  }  
  
Http().bindAndHandle(route, "localhost", 1111)
```

Request / Response (HTTP)

```
val http = Http(context.system)
val HttpResponse(StatusCodes.OK, headers, entity, _) = http.singleRequest(HttpRequest(uri = url))
val responseAsString = entity.dataBytes.runFold(ByteString(""))(_ ++ _)
```

Http client uses Akka system (dispatcher) to create a non blocking/reactive client.

The client can issue a single request that will return an entity which streams the response as the server sends the bytes

This stream can be consumed reactively using the `runFold` method

Serve

Expose and Predict

Writing the service

Use notebook: 04_Access Service.snb

We'll expose the data from Cassandra in JSON as a timeseries ("all" and range)

We'll deploy the trained model to predict in real time

Resource Manager

Optimize Hardware Usage

Cluster

The distributed data science pipeline wouldn't be complete without considering the hardware on which each piece will have to run on

This shouldn't be neglected of course and it's necessary to use it efficiently. Each tools can be scaled independently which is the strength of our pipeline, on the other side, we need to provision the infrastructure to do so.

We need to find a way to avoid having a dedicated infrastructure for each tool! However, we have to mutualize the available resources

Resource Manager

A classical way to do resource management was to use virtual machines that will run components

However, the management of such virtual machines can become quickly a hassle

That's, rather than managing machines we shall focus on manage resources independently... here we talk about CPUs, Memory, bandwidth and so on.

Orchestration

The orchestration of the tasks running on the system is necessary to ensure:

- **that the cluster is used efficiently**
- **the connected tools are provisioned enough to work**

This kind of orchestration needs to either keep track of the overall state or react on state changes.

Example: how to deploy, scale and maintain a bag micro service instances?

Mesos

Two Levels Scheduler

Mesos

Cluster Resource negotiator

Scalable to 10,000s of nodes

Fault-tolerant, battle-tested

An SDK for distributed apps

Native Docker support

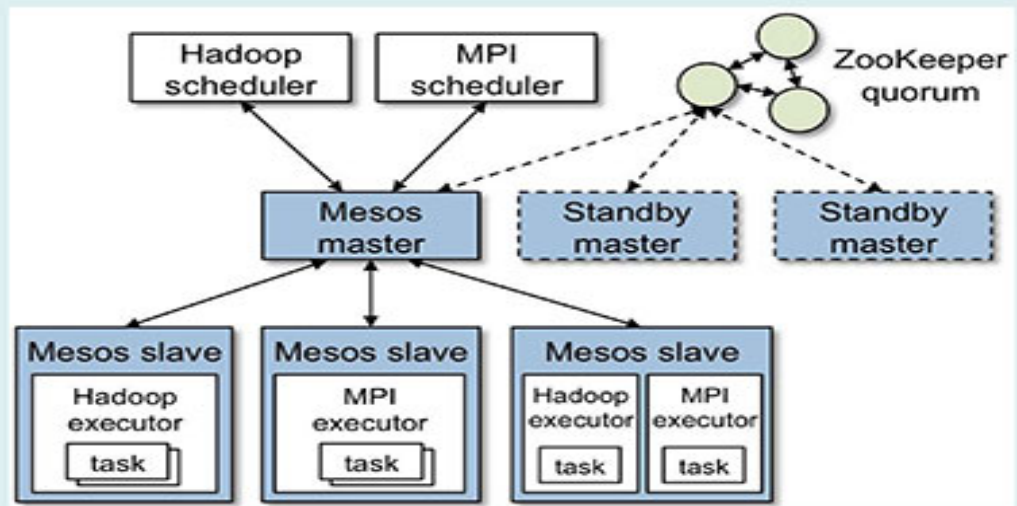
80% usage instead of 15%!

Architecture

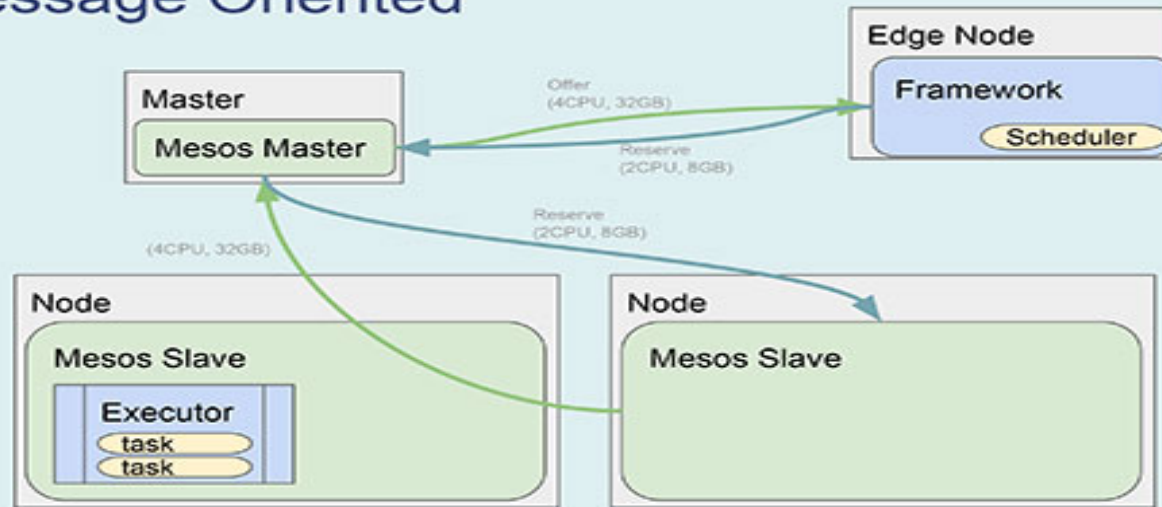
HA

2 levels schedulers!

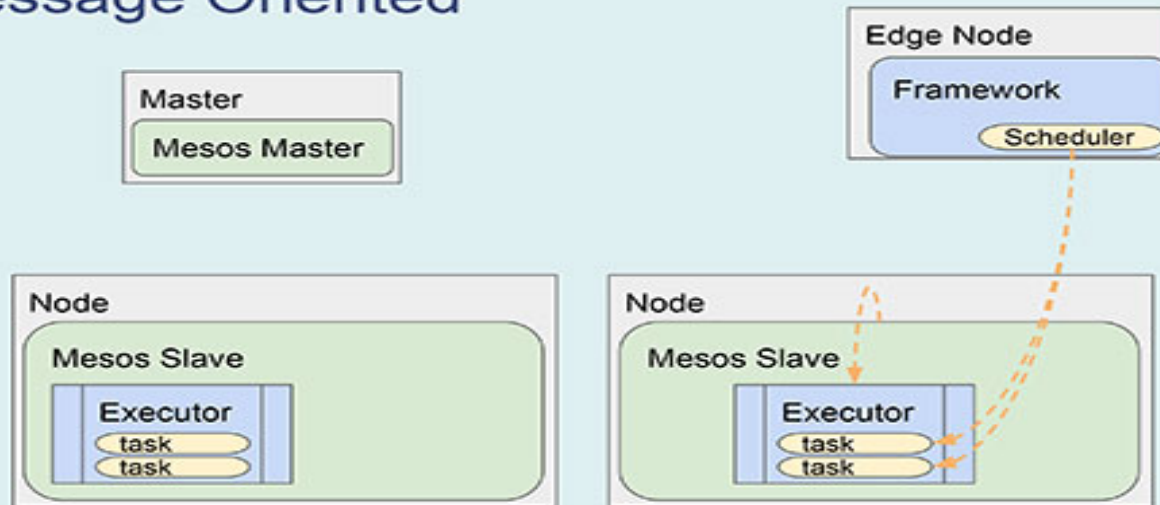
- masters
- frameworks



Message Oriented



Message Oriented



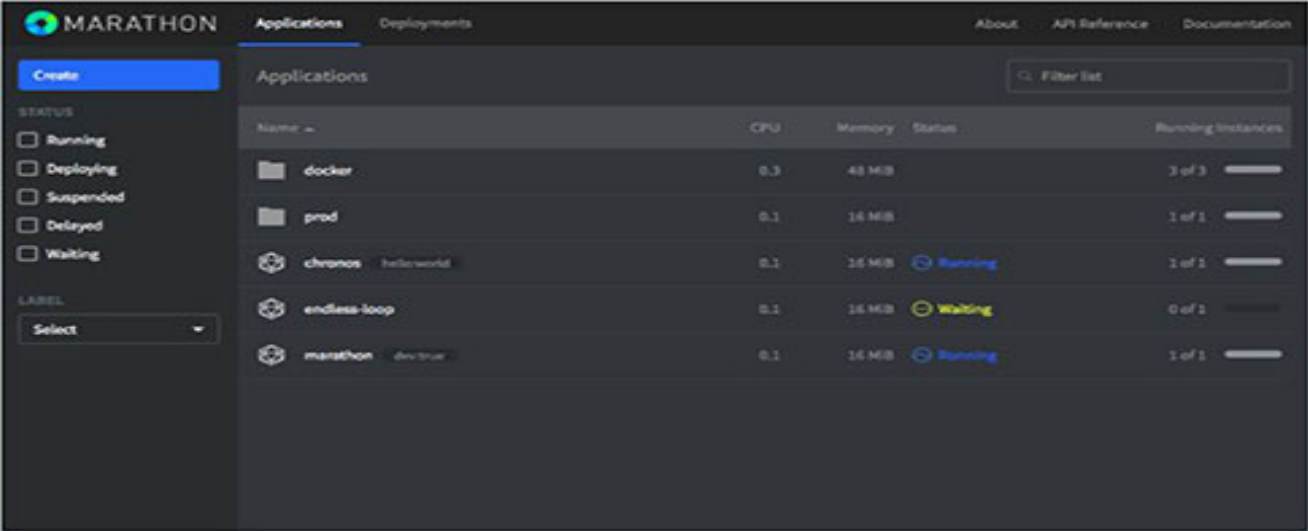
Useful Frameworks

For Mesos

Marathon

- **HA run any number of Marathon schedulers, but only one gets elected as leader**
- **Constraints - e.g., only one instance of an application per rack, node, etc.**
- **Service Discovery & Load Balancing via HAProxy or the events API.**
- **Health Checks: check your application's health via HTTP or TCP checks.**
- **JSON/REST API for easy integration and scriptability**
- **Native Docker support**

Marathon



The screenshot shows the Marathon web interface. At the top, there's a navigation bar with the Marathon logo, tabs for 'Applications' and 'Deployments', and links for 'About', 'API Reference', and 'Documentation'. On the left sidebar, there's a 'Create' button and a 'STATUS' filter section with checkboxes for Running, Deploying, Suspended, Delayed, and Waiting. Below that is a 'LABEL' section with a 'Select' dropdown. The main area displays a table of applications. The table has columns for Name, CPU, Memory, Status, and Running Instances. The applications listed are 'docker', 'prod', 'chronos' (with a 'helloworld' label), 'endless-loop', and 'marathon' (with a 'dev:prod' label). The 'endless-loop' application is highlighted in yellow and has a 'Waiting' status.

Name	CPU	Memory	Status	Running Instances
docker	0.3	48 MB		3 of 3
prod	0.1	16 MB		1 of 1
chronos <small>helloworld</small>	0.1	16 MB	Running	1 of 1
endless-loop	0.1	16 MB	Waiting	0 of 1
marathon <small>dev:prod</small>	0.1	16 MB	Running	1 of 1

Chronos

- **8601 Repeating Interval Notation**
- **Handles dependencies**
- **Job Stats (e.g. 50th, 75th, 95th and 99th percentile timing, failure/success)**
- **Job History (e.g. job duration, start time, end time, failure/success)**
- **Fault Tolerance (Hot Master)**

Chronos

The screenshot displays the Chronos web interface. On the left, a sidebar shows a list of tasks, each with a status indicator (e.g., 'SUCCESS', 'FAILURE'). The main panel on the right shows the details of a selected task. The task name is 'daily_gibson-default_data_cooked_table_imp...'. The command is 'cd /srv/datachef && . /srv/scr/scr.profile && /bin/bash -x ./import_mysql/import_tables.bash'. A calendar pop-up is overlaid on the task details, showing the date '2012-03-19' selected. The calendar is for March 2012, with the 19th highlighted in blue. The task status is 'FAILURE'.

GRAPH LAST

NAME

COMMAND

PARENTS

OWNER

SCHEDULE

EPILON

PT15M

EXECUTOR

daily_gibson-default_data_cooked_table_imp...

COMMAND

cd /srv/datachef && . /srv/scr/scr.profile && /bin/bash -x ./import_mysql/import_tables.bash

2012-03-19 T 10:04:37 Z/P T24h

March 2012

Sun	Mon	Tue	Wed	Thu	Fri	Sat
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	31				

Today Clear

Assessment #3

Extend Service Access

Extending Access Layer

Use the notebook provided in the slack room: Assessment 3.snb

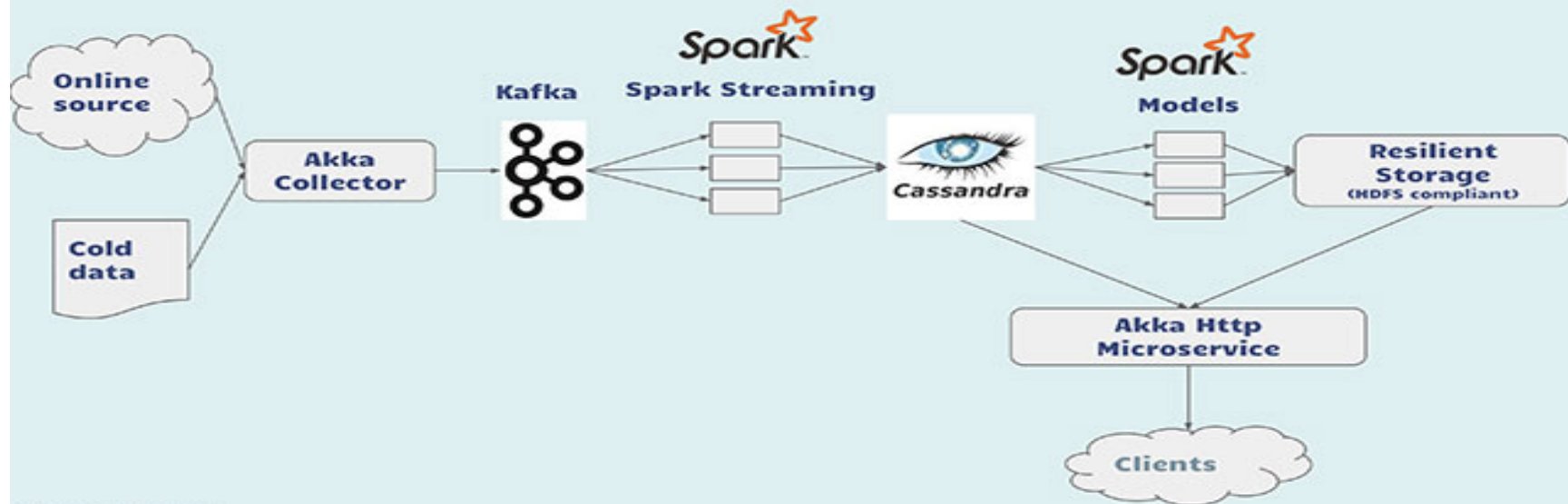
The task is to extend the access layer with a new micro service that will have the responsibility to return the last quote for JPM.

This service used in combination with the original prediction one allows to compare and plot the error. Check the Client Example.snb

Wrap Up

From Big Data to Micro Service

Distributed Data Science Pipeline



Follow Up

Use Mesos

Productize notebooks using SBT, Docker or similar

Face the challenge to build an enterprise ready, sustainable and maintainable solution...

By first introducing schemas at each step!

Don't get stuck with existing implementations, new models are created and developed to leverage the power of distributed systems and the amount of available data!

DATA FELLAS

<http://data-fellas.guru>

Helping businesses to get ready for
Enterprise Data Science at any Scale

We are hiring!

Data Engineers, Scientists, Analysts, ...

Help us: put a star on the [Spark Notebook](#) 