DATA DRIVEN RIJNMOND

BUILDING A DATA INGESTION & PROCESSING PIPELINE WITH SPARK & AIRFLOW

Tom Lous
@tomlous

https://cdn.mg.co.za/crop/content/images/2013/07/18/Coffee_Machine_-_Truth_-_Cape_Town_DH_4271_i2e.jpg/1680x1080/

WHO ARE WE?

DATLINQ BY NUMBERS

- 14 countries

  Netherlands, Belgium, France, UK, Germany, Spain, Italy, Luxembourg

- 100 customers across Europe

  Coca-Cola, Unilever, FrieslandCampina, Red Bull, Spa, AB InBev & Heineken

- 2000 active users

  Working with our SaaS, insights & mobile solutions

- 1.2M outlets

  Restaurants, bars, hotels, bakeries, supermarkets, kiosks, hospitals

- 5000 mutations per day

  Out of business, starters, ownership change, name change, etc

- 30 students

  Manually checking many changes

https://www.pexels.com/photo/people-on-dining-chair-inside-the-glass-wall-room-230736/

WHAT DO WE DO?

DATLINQ COLLECTS DATA

- Outlet features

  Address, segmentation, opening hours, contact info, lat/long, terrace, parking, pricing

- Outlet surroundings

  Nearby public transportation, school, going out area, shopping, high traffic

- Demographics

  Median income, age, density, marital status

- Social Media

  Presence, traffic, likes, posts

http://www.wandeltourrotterdam.nl/wp-content/uploads/photo-gallery/1506-Rotterdam-Image-Bank.jpg

WHY DO WE DO IT?

DATLINQ'S MISSION

- Food service market transparent
- Effective marketing & sales
- Which food products are relevant where?

http://frietwinkel.nl/wp-content/uploads/2014/08/Frietwinkel-openingsfoto.jpg

SOLUTION

WHY SPARK?

- Scaling is hard

  Mo' outlets, mo' students
- Data != information

  One-of analysis projects should be done continuously and immediately
- Work smarter not harder

  Have students train AI instead of repetitive work
- Faster results

  Distribute computing allows for increased processing speeds

SOLUTION?

https://s-media-cache-ak0.pinimg.com/originals/7e/63/11/7e6311d96eb4bb28a41ea34378de4182.jpg

SOLUTION

HOW TO START

- Focus on MVP

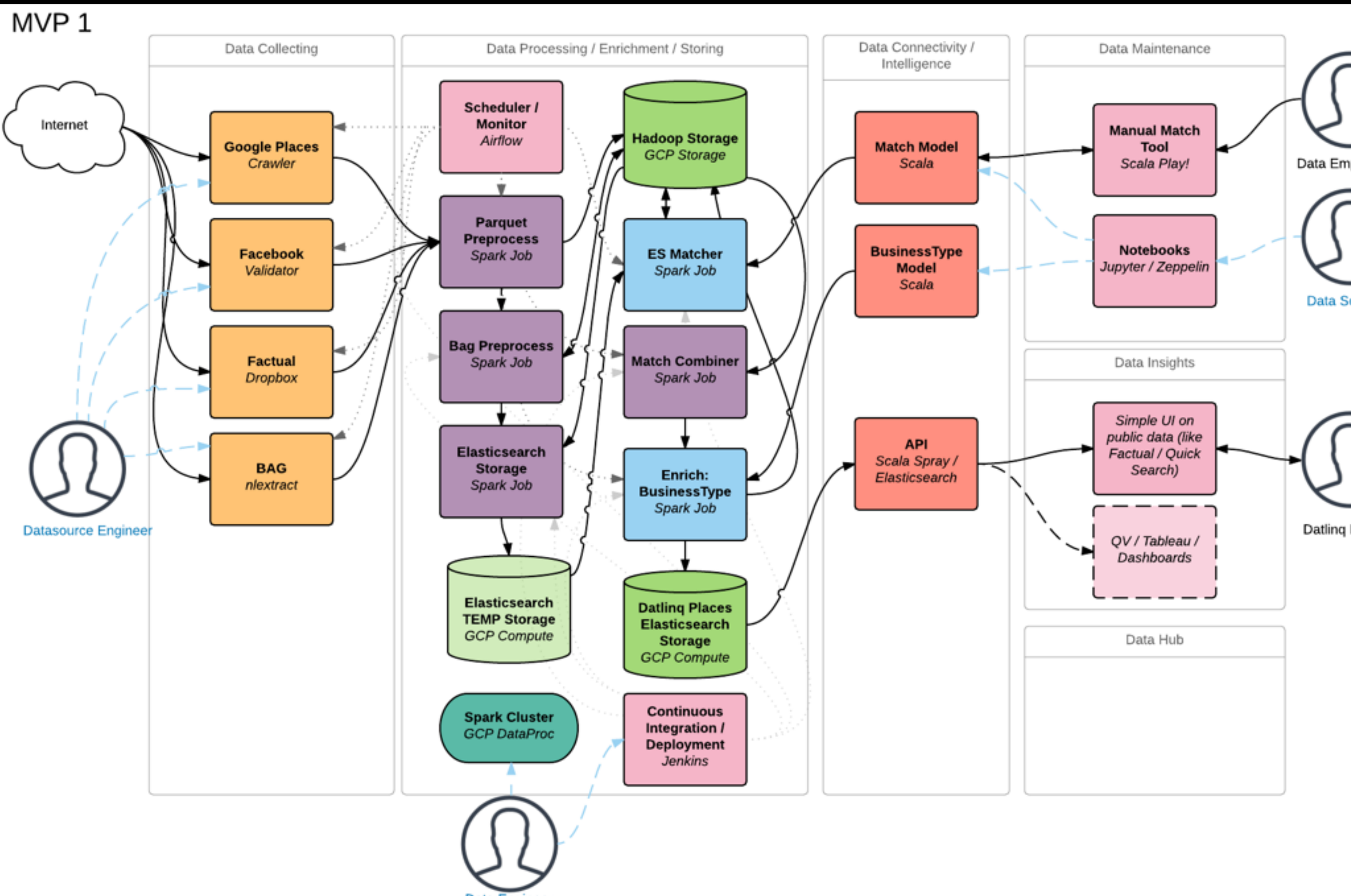
  Minimal Viable Product that acts like a Proof of Concept
- Declare riskiest assumption

  We can automatically match data from different sources
- Prove it

  Small scale ML models

http://realcorfu.com/wp-content/uploads/2013/09/kiosk3.jpg

BUILDING THE PIPELINE



USE THE CLOUD

- Flexible usage

  Pay for what you use
- Scaling out of the box

  The only limit seems to be your credit card
- CLI > UI

  CLI tools are your friend. UI is limited

```
tomlous@~/Development/Scala/datalabs-job/gcloud> gcloud
ERROR: (gcloud) too few arguments
Usage: gcloud [optional flags] <group | command>
  group may be           app | auth | components | compute | config |
                         container | dataflow | dataproc | datastore | debug |
                         deployment-manager | dns | iam | organizations |
                         projects | service-management | source | sql | topic
  command may be         docker | feedback | help | info | init | version

For detailed information on this command and its flags, run:
  gcloud --help
```

https://s-media-cache-ak0.pinimg.com/originals/00/f8/fa/00f8fafea39a7b185de0e3c6c7bbaaec.jpg

BUILDING THE PIPELINE

DEVELOP LOCALLY

- Check versions

  Your cloud provider sets the max version (e.g. Spark 2.0.2 instead of 2.1.0)

- Same code smaller data

  Use samples of your data to test on

- Use at least 2 threads

  Otherwise parallel execution cannot be truly tested

- Exclude libraries from deployment

  Some libraries are provided on the cloud, don't add them to your final build

```
implicit val spark = SparkSession
  .builder()
  .master("local[2]")
  .appName(this.getClass.getSimpleName)
  .config("spark.ui.enabled", "false")
  //    .config("spark.eventLog.dir", "/tmp/spark-events")
  //    .config("spark.eventLog.enabled", "true")
  .getOrCreate()
```

```
val sparkDependencyScope = sys.props.getOrElse("sparkDependencyScope", default = "compile")

// spark
libraryDependencies += "org.apache.spark" %% "spark-core"      % "2.0.2" % sparkDependencyScope
libraryDependencies += "org.apache.spark" %% "spark-sql"       % "2.0.2" % sparkDependencyScope
libraryDependencies += "org.apache.spark" %% "spark-streaming" % "2.0.2" % sparkDependencyScope
```

https://s-media-cache-ak0.pinimg.com/originals/00/f8/fa/00f8fafea39a7b185de0e3c6c7bbaaec.jpg

BUILDING THE PIPELINE

CONTINUOUS INTEGRATION

- Deploy code

  Manual uploads or deployments is asking for trouble.
  Use polling instead of pushing for security
  Create pipeline script to stage builds
  Automated tests

```
node {
    try {
        stage("Checkout") {
            git url: 'git@bitbucket.org:datlinq-datalabs/job.git'
        }

        stage("Build") {
            wrap([$class: 'AnsiColorBuildWrapper', 'colorMapName': 'XTerm']) {
                sh "sbt -DsparkDependencyScope=provided -Dspark.es.cluster.name=eshub -Dspark.es.nodes=192.168.0.2 -Dspark.es.port=9200 -Dspark.es.cluster.port=9300 cle
            }
        }

        stage("Deploy to accept") {
            sh "git rev-parse HEAD > target/_GIT_REVISION"
            sh "gsutil cp target/scala-2.11/job.jar gs://datlinq/accept/jobs/"
            sh "gsutil cp target/_GIT_REVISION gs://datlinq/accept/jobs/"
            slackSend (color: '#4F8A10', message: "SUCCESSFUL: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' (${env.BUILD_URL})")
        }


    } catch (e) {
        currentBuild.result = "FAILED"
        slackSend (color: '#D8000C', message: "FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' (${env.BUILD_URL})")
        throw e
    }
}
```

- Commit config

  Store CI integration

BUILDING THE PIPELINE

SPARK JOBS

- All jobs in one project

  Unless there is a very good reason, keep all your Jobs in one file

- Use Datasets as much as possible

  Avoid RDD's and DataFrames

- Use Case Class (encoders) as much as possible

  Generic Row object will only get you so far

- Verbosity only in main class

  Log messages from nodes won't make it to the master

- Look at the execution plan

  Remove temp files, clean the slate

```
= Optimized Logical Plan ==
roject [address_components#95, adr_address#141, formatted_address#166, formatted_phone_number#191, geometry#99, icon#216, id#101, int
- Project [address_components#95, UDF(adr_address#96) AS adr_address#141, UDF(formatted_address#97) AS formatted_address#166, UDF(fo
  +- LogicalRDD [_id#94, address_components#95, adr_address#96, formatted_address#97, formatted_phone_number#98, geometry#99, icon#1

= Physical Plan ==
Project [address_components#95, adr_address#141, formatted_address#166, formatted_phone_number#191, geometry#99, icon#216, id#101, i
- *Project [address_components#95, UDF(adr_address#96) AS adr_address#141, UDF(formatted_address#97) AS formatted_address#166, UDF(f
  +- Scan ExistingRDD[_id#94,address_components#95,adr_address#96,formatted_address#97,formatted_phone_number#98,geometry#99,icon#10
```

BUILDING THE PIPELINE

PARQUET

- What is it

  Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem

- Compressed

  Small footprint.

- Columnar

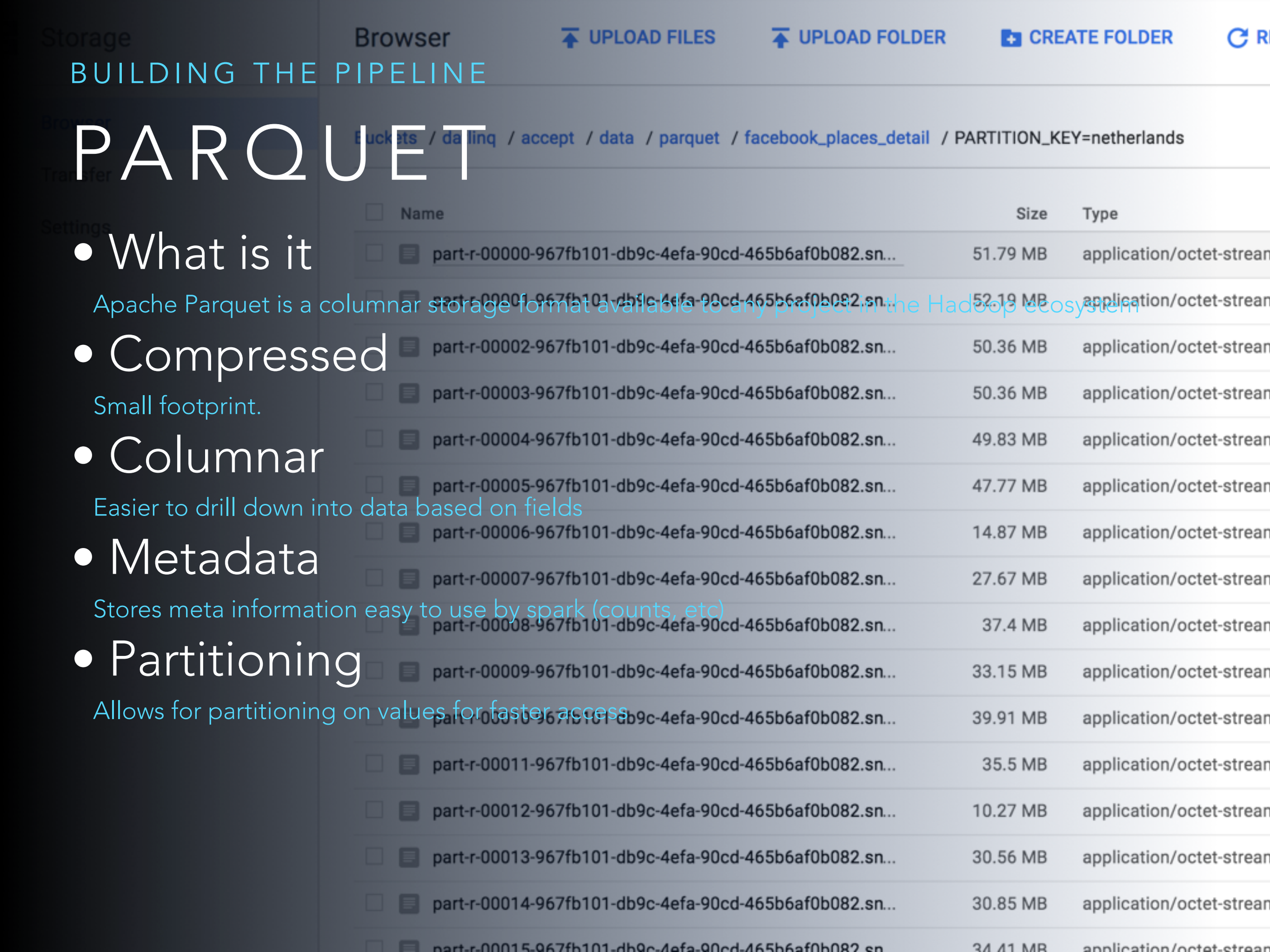
  Easier to drill down into data based on fields

- Metadata

  Stores meta information easy to use by spark (counts, etc)

- Partitioning

  Allows for partitioning on values for faster access



BUILDING THE PIPELINE

ELASTICSEARCH

- NxN matching is computationally expensive
  - Querying Elasticsearch / Lucene / Solr less
  - Fuzzy searching using n-grams and other tokenizers
- Cluster out of the box
  - Auto replication
- Config differs from normal use
  - It's not a search feature for a website.
  - Don't care about update performance
  - Don't load balance queries (_local)
- Discovery Plugins for GC
  - So nodes can find each other on local network without Zen
  - Create VM's with --scopes=compute-rw

BUILDING THE PIPELINE

ANSIBLE

- Automate VM's / agent less

  Elasticsearch cluster

- Dependant on Ansible GC Modules

  gcloud keeps updating. Ansible GC doesn't directly follow suite

```
- name: create instances
  gce:
    instance_names: "{{item.node}}"
    zone: "{{ zone }}"
    machine_type: custom-16-30720
    preemptible: true
    disk_auto_delete: true
    disks:
      - name: "{{ item.node }}-disk"
        mode: READ_WRITE
    state: present
    service_account_email: "{{ service_account_email }}"
    service_account_permissions: "compute-rw"
    credentials_file: "{{ credentials_file }}"
    project_id: "{{ project_id }}"
    tags: "elasticsearch,es5,{{cluster_name}},link"
  register: gce_raw_results
  with_items: "{{nodes}}"
```

BUILDING THE PIPELINE

AIRFLOW

- Create workflows

  Define all tasks and dependencies

- Cron on steroids

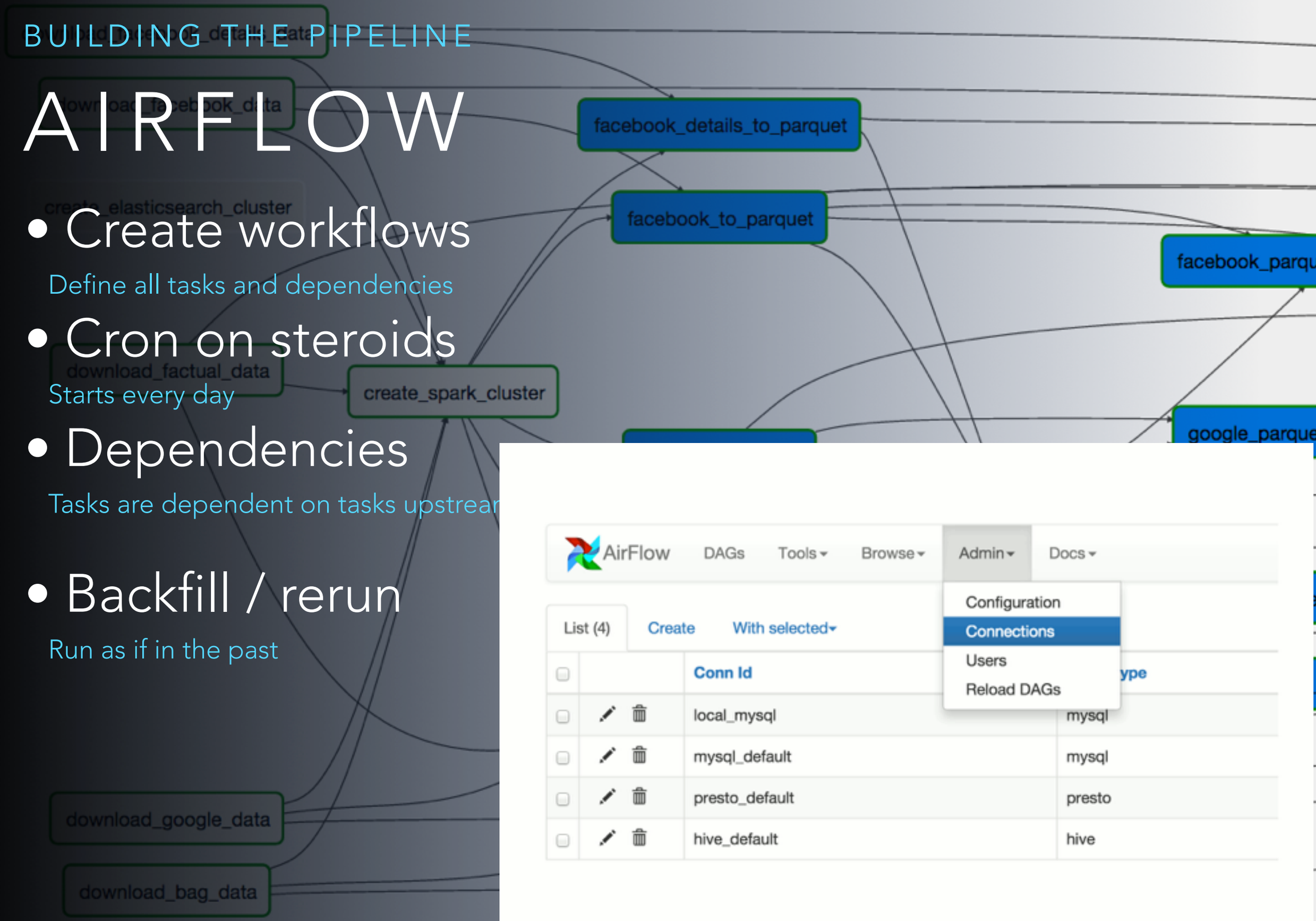
  Starts every day

- Dependencies

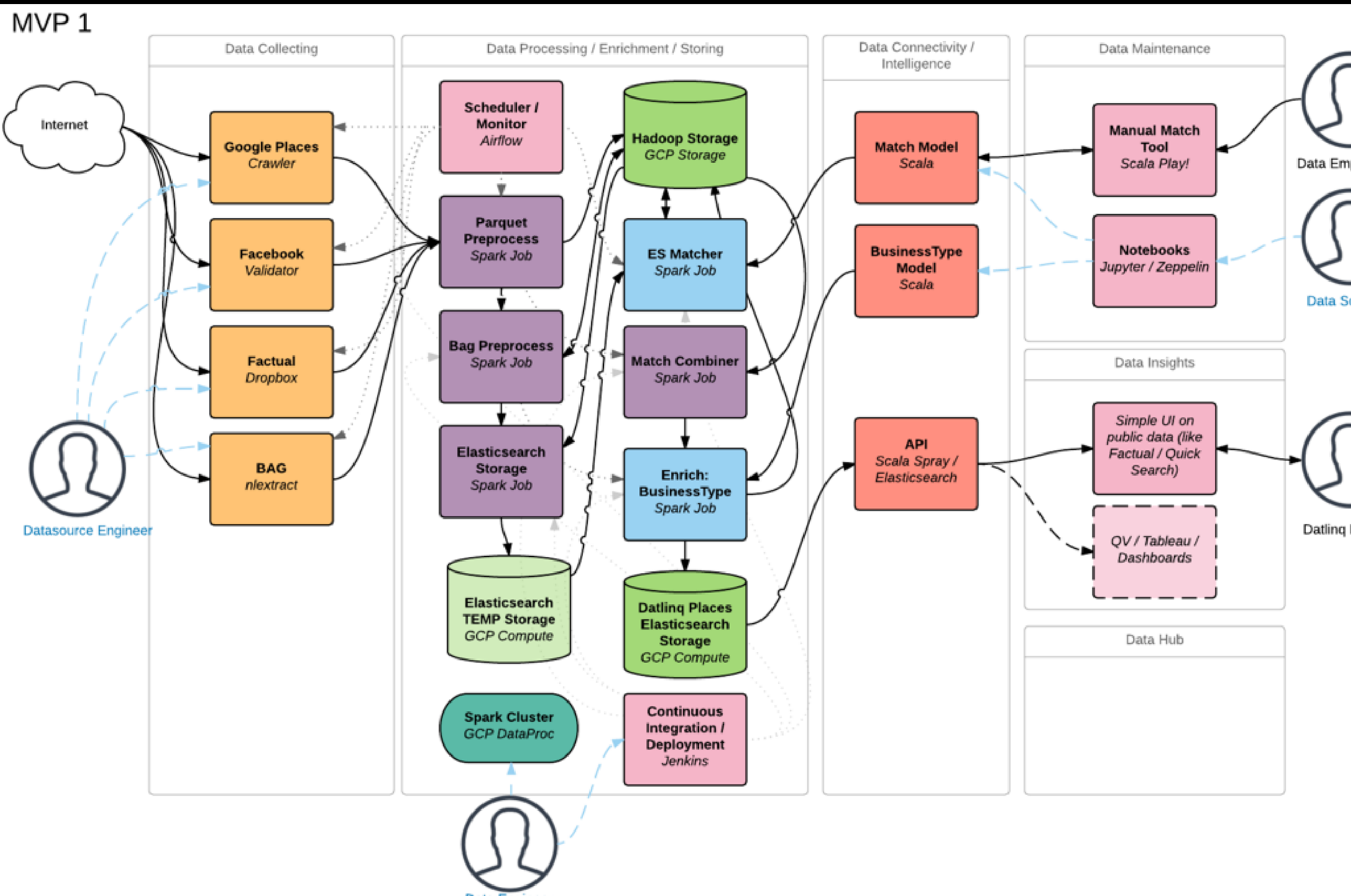
  Tasks are dependent on tasks upstrea

- Backfill / rerun

  Run as if in the past



BUILDING THE PIPELINE



CONCLUSION

FINALLY

- What's next?
  - Improving MVP
  - Adding more machine learning
  - Adding more datasources
  - Adding more countries
  - Improving architecture
- Production
  - Create an API for use in our products
  - Create UI for exploring the data
  - Exporting to standard databases to further use
- Team
  - Hire Data Engineer(s)
  - Hire Data Scientist(s)

https://static.pexels.com/photos/14107/pexels-photo-14107.jpeg

CONCLUSION

THANKS FOR WATCHING

- Questions?

  AMA
- Vacancies!

  Ask me or check our site http://www.datlinq.com/en/vacancies
- Break

  Grab a drink an let's reconvene after a 15 min break