Julian Kunkel

The Apache Ecosystem and Beyond

Outline

Hortonworks (2019 bought by Cloudera)

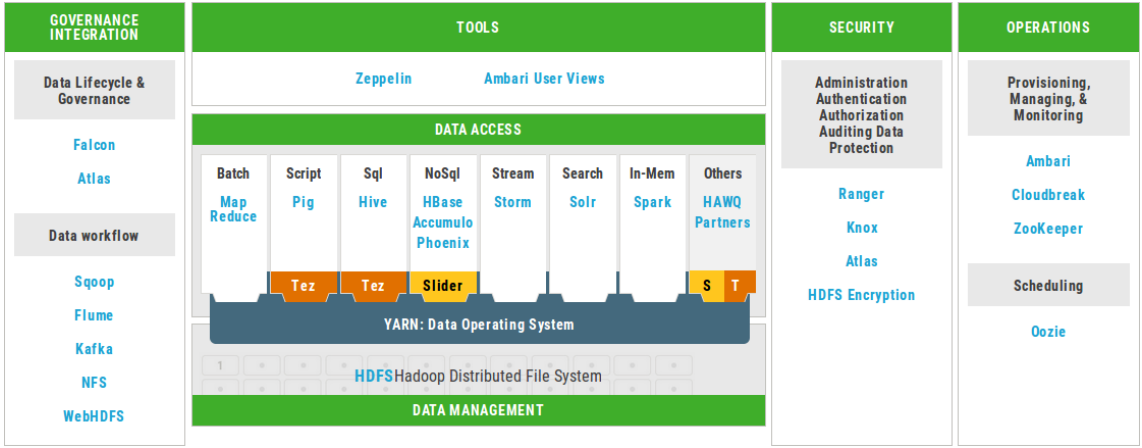


Figure: Screenshot from [40]

- Additionally: Hortonworks offers support, service
- Build with open-source

Cloudera Enterprise Data Hub [25]

- Cloudera offers support, services and tools around Hadoop
- Unified architecture: common infrastructure and data pool for tools
- Build with open-source tools, some own tools for management
- Cloudera has various other products (e.g., cloud-ready available)

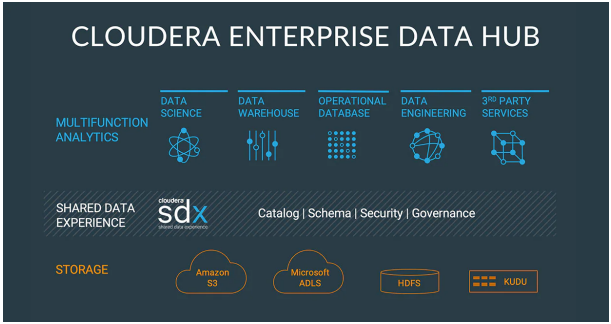


Figure: Source: [25]

Supporting Tools[43]

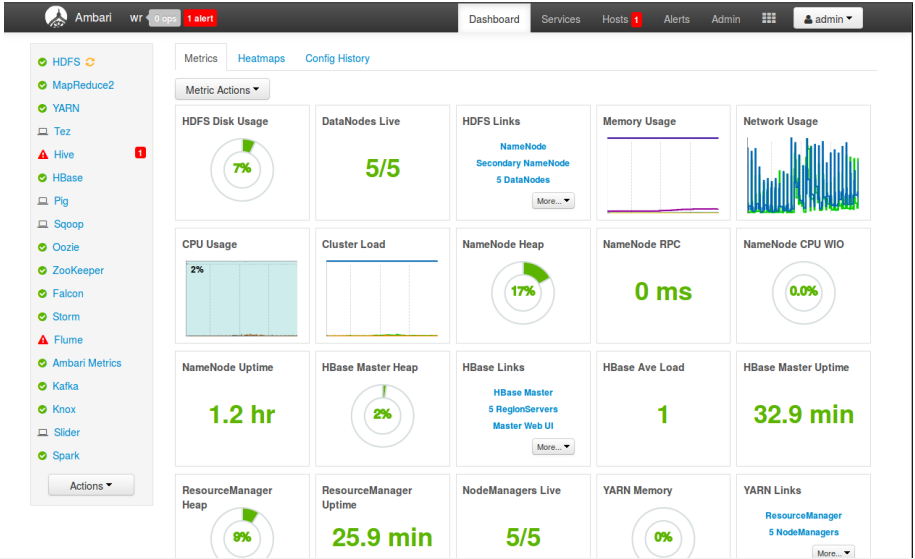
- Ambari: A Tool for Managing Hadoop Clusters
- Hue: Manage „BigData“ projects in a browser
- ZooKeeper: coordination/configuration service for services
- Oozie: Workflow scheduler (schedules/triggers workflows)
- Falcon: Data governance engine for data pipelines
- Flume: collecting, aggregating and moving large streaming event data
- Kafka: publish-subscribe distributed messaging system
- knox: REST API gateway (for all services)
- Ranger: Integrate ACL permissions into Hadoop (ecosystem)
- Slider: YARN application supporting monitoring and dynamic scaling of non-YARN apps

[43] https://hadoop.apache.org/

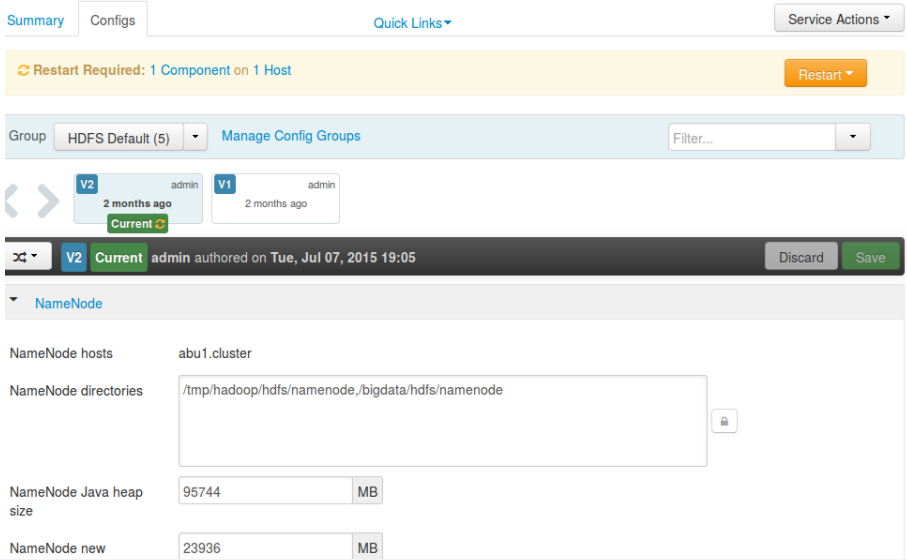
Ambari: A Tool for Managing Hadoop Clusters

- Convenient tool managing 10+ Apache tools
- Supports installation and management
  - Dealing with data dependencies
  - Service startup
  - Monitoring of health and performance
  - (Re)configuration of services
  - Web UI with REST API

Management with Ambari: Dashboard

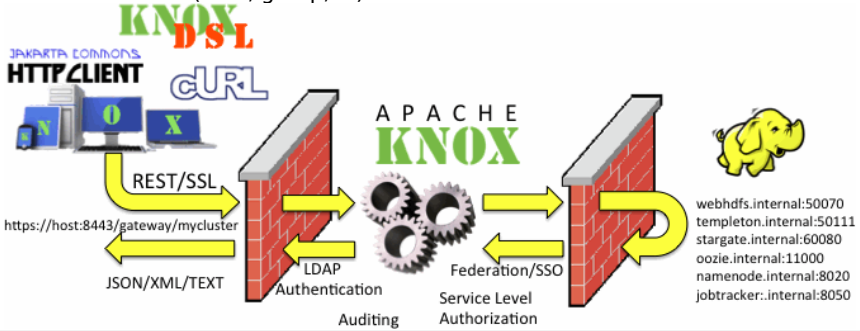


Management with Ambari: Configuration



Knox: Security for Hadoop [22]

- REST API Gateway for Hadoop ecosystem services
  - Supports: HDFS, Hcatalog, HBase, Oozie, Hive, Yarn, Storm
  - Supports multiple clusters
- Provides authentication, federation/SSO, authorization, auditing
- Enhances security providing central control and protection
  - SSL encryption
  - Authentication: LDAP, Active Directory, Kerberos
  - Authorization: ACL's (user, group, IP) on service level[44]



Example Accesses via the API [22]

List a HDFS directory

```
curl -i -k -u guest:guest-password -X GET 'https://localhost:8443/gateway/sandbox/webhdfs/v1/?op=LISTSTATUS'
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 450
Server: Jetty(6.1.26)

{"FileStatuses":{"FileStatus":[
{"accessTime":0,"blockSize":0,"group":"hdfs","length":0, "modificationTime":1350595859762, "owner":"hdfs",
    ↪ "pathSuffix":"apps", "permission":"755", "replication":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0, "group":"mapred","length":0, "modificationTime":1350595874024,
    ↪ "owner":"mapred","pathSuffix":"mapred", "permission":"755", "replication":0,"type":"DIRECTORY"},
]}}
```

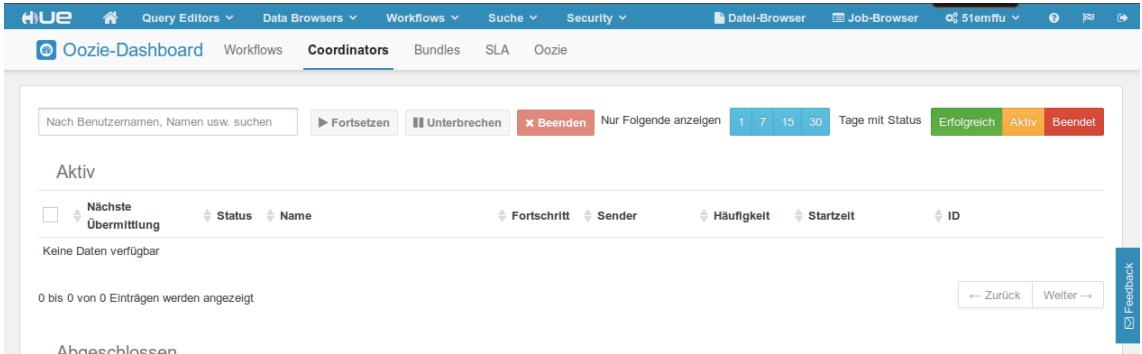
Hue [12]: Lightweight Web Server for Hadoop

- Manage BigData projects in a browser
- Supports: Hadoop ecosystem
  - HDFS, Pig, Hive, MapReduce, Spark, Flink, ...

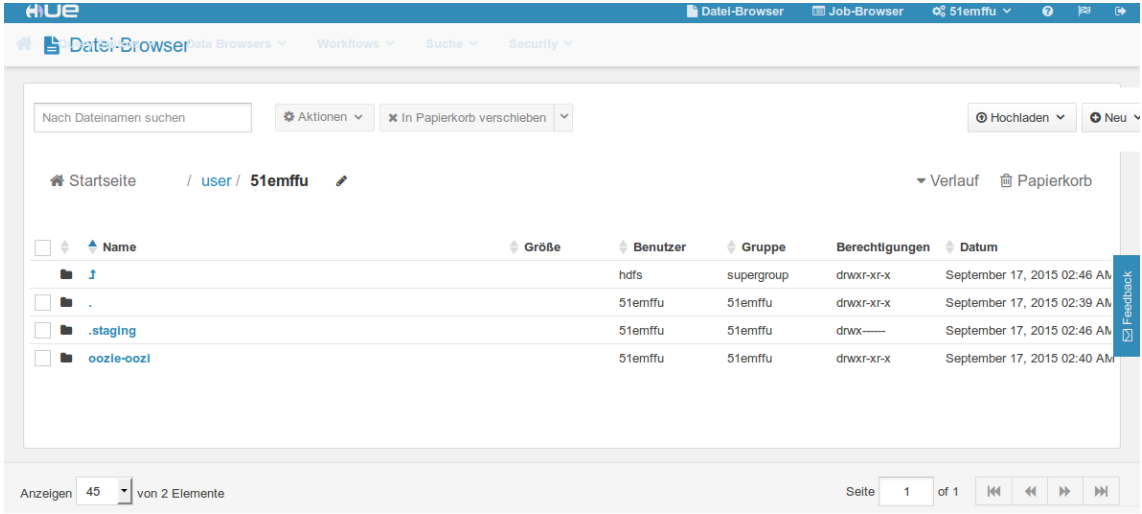
Features

- Data upload/download
- Management of HCatalog tables
- Query editor (Hive, Pig, Impala)
- Starting and monitoring of jobs

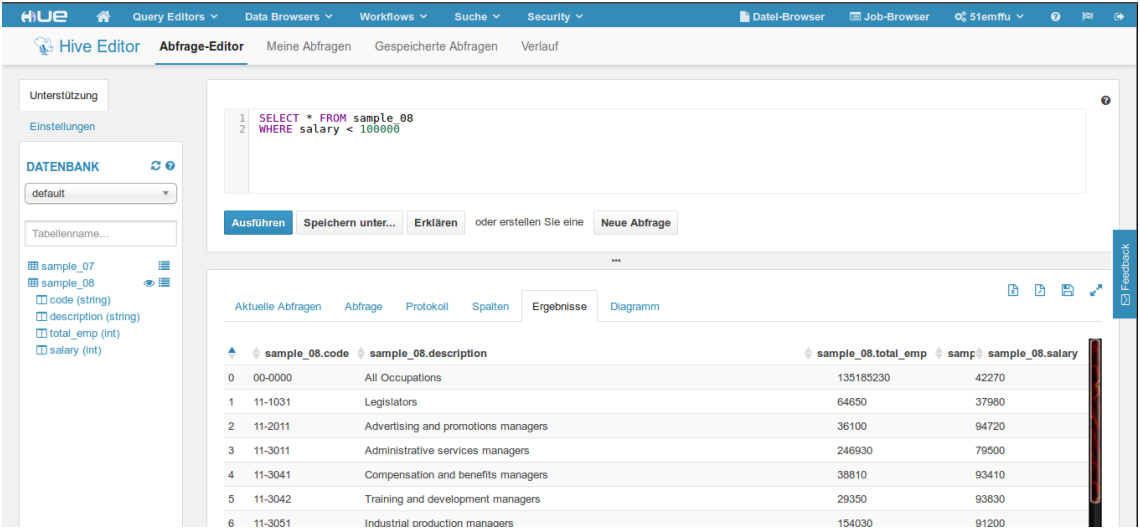
Hue: Lightweight Web Server for Hadoop



Hue: Lightweight Web Server for Hadoop



Hue: Lightweight Web Server for Hadoop



	sample_08.code	sample_08.description	sample_08.total_emp	sample_08.salary
0	00-0000	All Occupations	135185230	42270
1	11-1031	Legislators	64650	37980
2	11-2011	Advertising and promotions managers	36100	94720
3	11-3011	Administrative services managers	246930	79500
4	11-3041	Compensation and benefits managers	38810	93410
5	11-3042	Training and development managers	29350	93830
6	11-3051	Industrial production managers	154030	91200

Hue: Lightweight Web Server for Hadoop

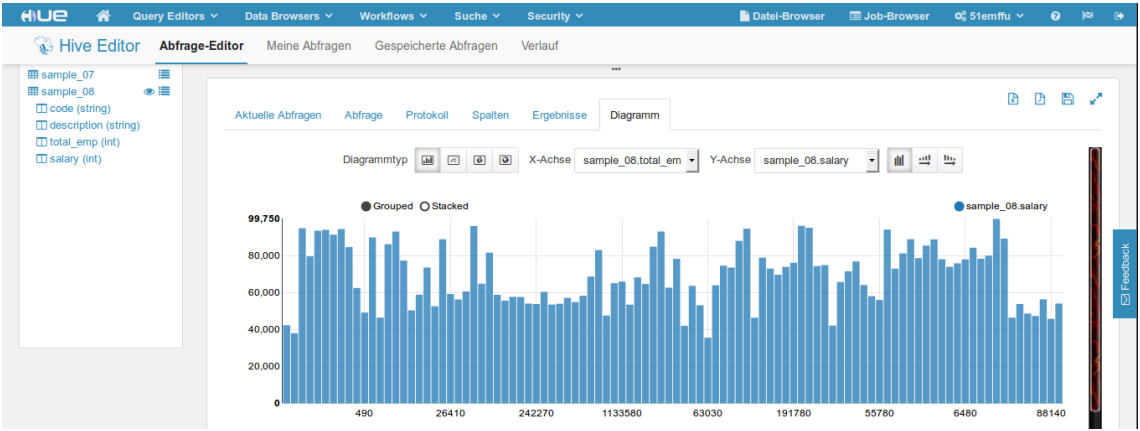


Figure: Visualizing query results in diagrams (Live system on `gethue.com`)

Zeppelin [39]

- Web-based notebook for interactive data analytics
  - Add code snippets
  - Arrange them
  - Execute them
  - Visualizes results
- Supports Spark, Scala, Pig, SQL, Python, R, Hive, Shell, ...
- Collaborative environment for multiple users
- Can export paragraph links for embedding into a webpage
- Similarities to Jupyter but has some improvements

Zeppelin Notebook Interpreter Connected

Zeppelin Tutorial

default

```
%md
## Welcome to Zeppelin.
##### This is a live tutorial, you can run the code yourself. (Shift-Enter to Run)
```

Took 1 seconds (outdated) FINISHED

Load Data Into Table

ERROR

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or SqlContext)
// So you don't need create them manually

// load bank data
val bankText = sc.parallelize(
    IOUtils.toString(
        new URL("https://s3.amazonaws.com/apache-zeppelin/tutorial/bank/bank.csv"),
        Charset.forName("utf8")).split("\n"))

case class Bank(age: Integer, job: String, marital: String, education: String, balance: Integer)

val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
    s => Bank(s(0).toInt,
            s(1).replaceAll("\"", ""),
            s(2).replaceAll("\"", ""),
            s(3).replaceAll("\"", ""),
            s(5).replaceAll("\"", "").toInt
        )
).toDF()
bank.registerTempTable("bank")
```

Too many open files

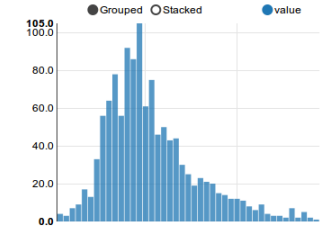
Took 0 seconds (outdated)

```
%sql
select age, count(1) value
from bank
where marital="${marital=single,single|divorced|married}"
group by age
order by age
```

FINISHED

marital

single




```
%sql
select age, count(1) value
from bank
```

ERROR

```
%sql
select age, count(1) value
from bank
```

ERROR

Outline

Oozie [15, 16]

- Scalable, reliable and extensible workflow scheduler
- Jobs are DAGs of actions specified in XML workflows
- Actions: Map-reduce, Pig, Hive, Spark, Shell actions
- Workflows can be parameterized
  - Triggers notifications via HTTP GET upon start/end of a node/job
  - Automatic user-retry to repeat actions when fixable errors occur
  - Monitors a few runtime metrics upon execution
- Interfaces: command line tools, web-service and Java APIs
- Integrates with HCatalog
- Coordinator jobs trigger start of jobs
  - By time schedules
  - When data becomes available
    - Requires polling of HDFS (1-10 min intervals)
    - With HCatalog's publish-subscribe, jobs can be started immediately
  - Can record events for service level agreement

Workflows [16]

- A workflow application is a ZIP file to be uploaded
  - Includes workflow definition and coordinator job
  - Bundles scripts, JARs, libraries needed for execution
- Workflow definition is a DAG with control flow and action nodes
  - Control flow: start, end, decision, fork, join
  - Action nodes: whatever to execute
- Variables/Parameters[45]
  - Default values can be defined in a config-default.xml in the ZIP
- Expression language functions help in parameterization[1]
  - Basic functions: `timestamp(), trim(), concat(s1, s2)`
  - Workflow functions: wf:errorCode(< action node >)
  - Action specific functions: hadoop:counters("mr-node")["FileSystemCounters"]["FILE_BYTES_READ"]
- Coordinator job is also an XML file

[45] They are used with with ${NAME/FUNCTION}, e.g., ${timestamp()}

Coordinator Jobs [17]

App which periodically starts a workflow (every 60 min)

```
<coordinator-app name="MY_APP" frequency="60" start="2009-01-01T05:00Z" end="2009-01-01T06:00Z" timezone="UTC"
       ↪ xmlns="uri:oozie:coordinator:0.1">
   <action>
      <workflow> <!-- here the workflow is not further defined -->
         <app-path>hdfs://localhost:9000/tmp/workflows</app-path>
      </workflow>
   </action>
</coordinator-app>
```

Every 24h check if dependencies for a workflow are met, then run it

```
<coordinator-app name="MY_APP" frequency="1440" start="2009-02-01T00:00Z" end="2009-02-07T00:00Z" ...>
   <datasets> <-- define a dataset, that is checked for existence -->
      <dataset name="input1" frequency="60" initial-instance="2009-01-01T00:00Z" timezone="UTC">
         <uri-template>hdfs://localhost:9000/tmp/revenue_feed/${YEAR}/${MONTH}/${DAY}/${HOUR}</uri-template>
      </dataset>
   </datasets>
   <input-events> <-- we depend on the last 24 hours input data -->
      <data-in name="coordInput1" dataset="input1">
          <start-instance>${coord:current(-23)}</start-instance>
          <end-instance>${coord:current(0)}</end-instance>
      </data-in>
   </input-events>
   <action>
      <workflow>
         <app-path>hdfs://localhost:9000/tmp/workflows</app-path>
      </workflow>
   </action>
</coordinator-app>
```

Example Oozie Workflow [13]

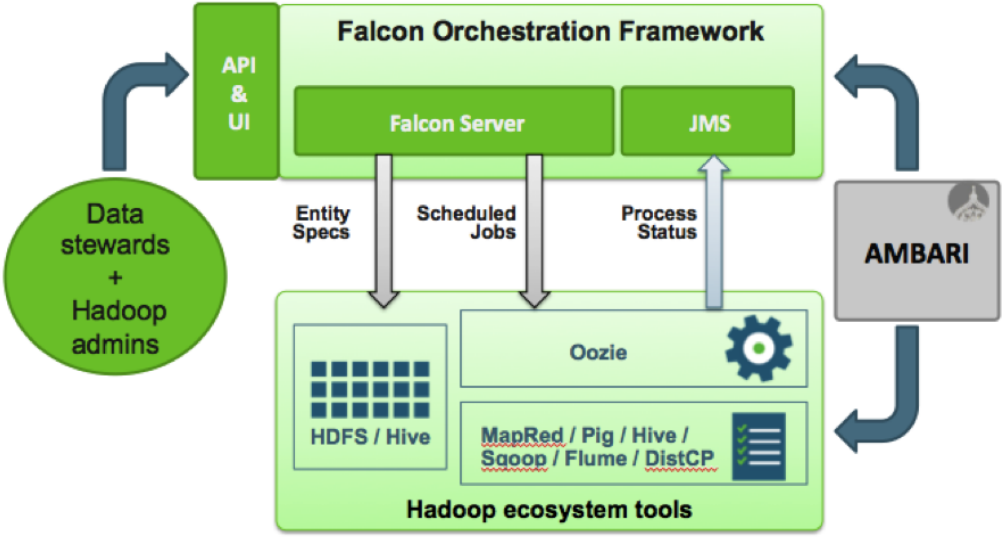
Three actions: Execute pig script, concatenate reducer files, upload files remotely via ssh

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="sample-wf">
  <start to="pig" />
  <action name="pig">
    <pig><job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <prepare><delete path="${output}"/></prepare>
      <configuration>
        <property> <name>mapred.job.queue.name</name><value>${queueName}</value></property>
        <property> <name>mapreduce.fileoutputcommitter.marksuccessfuljobs</name><value>true</value></property>
      </configuration>
      <script>${nameNode}/projects/bootcamp/workflow/script.pig</script>
      <param>input=${input}</param>
      <param>output=${output}</param>
      <file>lib/dependent.jar</file>
    </pig><ok to="concatenator" /><error to="fail" /> <-- the concatenator action is not shown here -->
  </action>

  <action name="fileupload">
  <ssh><host>localhost</host>
    <command>/tmp/fileupload.sh</command>
    <args>${nameNode}/projects/bootcamp/concat/data-${fileTimestamp}.csv</args><args>${wf:conf("ssh.host")}</args>
    <capture-output/></ssh>
  <ok to="fileUploadDecision" /><error to="fail"/>
  </action>

  <decision name="fileUploadDecision"> <-- check the exit status of the file upload -->
    <switch><case to="end">${wf:actionData('fileupload')['output'] == '0'}</case><default to="fail"/> </switch>
  </decision>
  <kill name="fail"><message>Workflow failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message></kill>
  <end name="end" />
</workflow-app>
```

Falcon [11,13]

- Feed (data set) management and processing system
- Simplifies dealing with many Oozie jobs
- Supports data governance
  - Define and run data pipelines (management policies)
  - Monitor data pipelines
  - Trace pipelines to identify dependencies and perform audits
- Data model defines entities describing policies and pipelines
  - Clusters define resources and interfaces to use
  - Feeds define frequency, data retention, input, outputs, retry and use clusters (multiple for replication)
  - Process: processing task, i.e., Oozie workflow, Hive or Pig script
- Features
  - Supports reuse of entities for different workflows
  - Enables replication across clusters and data archival
  - Supports HCatalog
  - Notification of users upon availability of feed groups

Falcon: High-level Architecture



Falcon: Example Pipeline

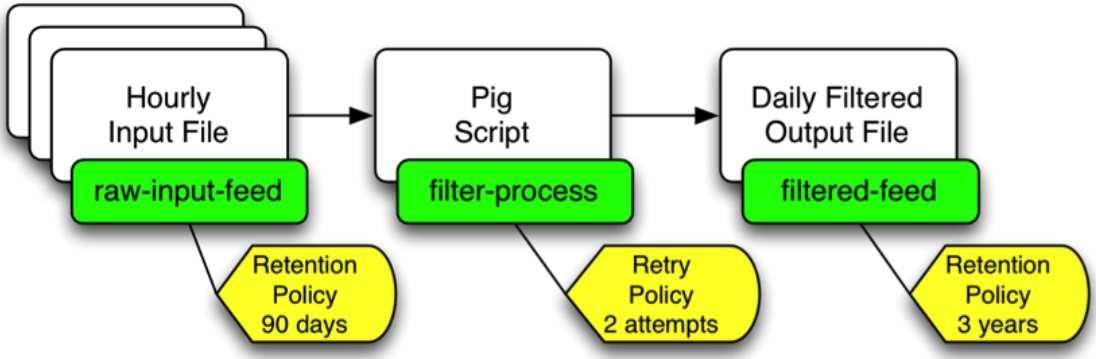


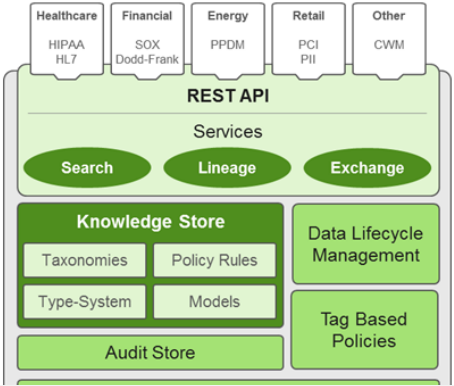
Figure: Source: [11]

Falcon: Example Process Definition [11, 14]

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Sample process. Runs at 6th hour every day. Input: last day hourly data. Output: for yesterday  -->
<process name="SampleProcess">
    <cluster name="wr" />
    <frequency>days(1)</frequency>

    <validity start="2015-04-03T06:00Z" end="2022-12-30T00:00Z" timezone="UTC" />

    <inputs>
        <input name="input" feed="SampleInput" start="yesterday(0,0)" end="today(-1,0)" />
    </inputs>

    <outputs>
            <output name="output" feed="SampleOutput" instance="yesterday(0,0)" />
    </outputs>

    <properties>
        <property name="queueName" value="reports" />
        <property name="ssh.host" value="host.com" />
        <property name="fileTimestamp" value="${coord:formatTime(coord:nominalTime(), 'yyyy-MM-dd')}" />
    </properties>

    <workflow engine="oozie" path="/projects/bootcamp/workflow" />

    <retry policy="backoff" delay="minutes(5)" attempts="3" />

    <-- How to check and handle late arrival of input data-->
    <late-process policy="exp-backoff" delay="hours(1)">
        <late-input input="input" workflow-path="/projects/bootcamp/workflow/lateinput" />
    </late-process>
</process>
```

Atlas [23]

- A framework for platform-agnostic data governance and metadata management
- Gather, process and preserve metadata; exchange with other tools
- Dynamic creation of tags for classification
- Audit operations, explore data linage (history) of data and metadata
- Support lifecycle management workflows (Falcon) and access control (Ranger)



Source: [48]

Atlas: Metadata Fields

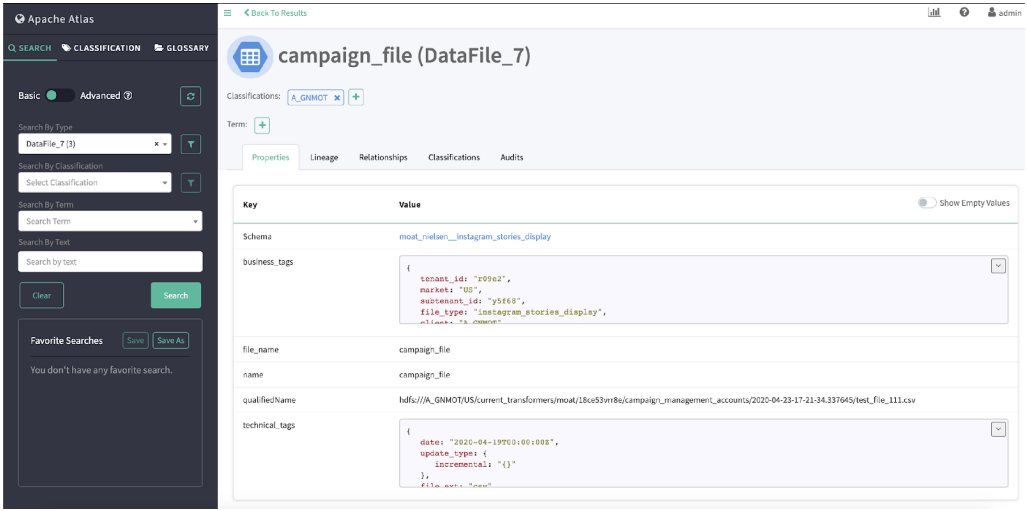
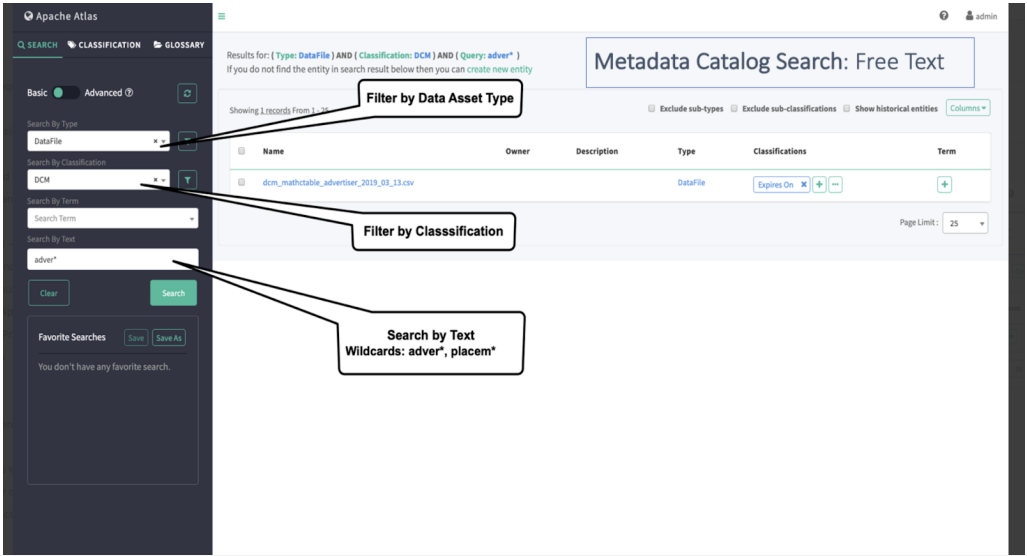


Figure: Source: [49]

Atlas: Metadata Search



Atlas: Data Linage Visualization

Sqoop (A retired tool replaced by Flume...) [18, 19]

- Transfers bulk data between Hadoop and RDBMS, either
  - One/multiple tables (preserving their schema)
  - Results of a free-form SQL query
- Uses MapReduce to execute import/export jobs
  - Parallelism is based on splitting one column's value
- Validate data transfer (comparing row counts) for full tables
- Save jobs for repeated invocation
- Main command line tool sqoop, more specific tools sqoop*

Features [19]

Import Features

- Incremental import (scan and add only newer rows)
- File formats: CSV, SequenceFiles, Avro, Parquet
  - Compression support
- Outsource large BLOBS/TEXT into additional files
- Import into Hive (and HBase)
- Can create the table schema in HCatalog automatically
  - With HCatalog, only CSV can be imported

Export Features

- Bulk insert: 100 records per statement
- Periodic commit after 100 statements

Import Process [19]

- Read the schema of the source table
- Create a Java class representing a row of the table
  - This class can be used later to work with the data
- Start MapReduce to load data parallel into multiple files
  - The number of mappers can be configured
  - Mappers work on different values of the splitting column
  - The default splitting column is the primary key
    - Determines min and max value of the key
    - Distributes fixed chunks to mappers
- Output status information to the MapReduce job tracker

Example Imports [19]

```
# Import columns from "foo" into HDFS to /home/x/foo (table name is appended)
# When not specifying any columns, all columns will be imported.
$ sqoop import --connect jdbc:mysql://localhost/db --username foo --table TEST --columns "matrikel,name"
    ↪ --warehouse-dir /home/x --validate

# We'll use a free-form query, it is parallelized on the split-by column
# The value is set into the magic $CONDITIONS variable
$ sqoop import --query 'SELECT a.*, b.* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS'  --split-by a.id
    ↪ --target-dir  /user/foo/joinresults

# To create the HCatalog table use --hcatalog-table or --hive-import
# See [19] for details of the available options
```

Slider [20]

- Is a YARN application that manages non-YARN apps on a cluster

⇒ Utilize YARN for resource management

- Enables installation, execution, monitoring and dynamic scaling
- Command line tool `slider`
- Apps are installed and run from a package
  - Tarball with well-defined structure [21]
  - Scripts for installing, starting, status, ...
- Example packages: jmemcached, HBase
- Slider is currently extended to deploy Docker images (Tech preview)

Apache Airflow™: What is Airflow?

- An open-source platform for batch-oriented workflow orchestration.
- Workflow is referred as DAGs (Direct Acyclic Graphs)
- Mainly, used for:
  - Developing,
  - Scheduling, and
  - Monitoring DAGs
- It requires a Linux server and has its own backend interface.
- Its user interface provides in-depth views of two things:
  - Pipelines, and
  - Tasks

Apache Airflow™: Graphical View

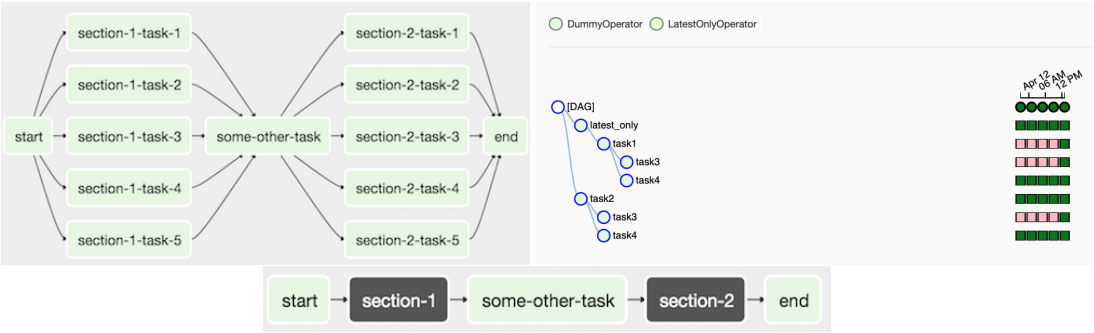


Figure: This example demonstrates different aspects of the Airflow DAG.[46]

46 Source: https://airflow.apache.org/docs/apache-airflow/stable/index.html

Apache Airflow™: Purpose

Airflow is for?

- Programmable workflows as:
  - Dynamic:
    - Pipelines are configured as Python code, allowing for dynamic generation.
  - Extensible:
    - The framework contains operators to connect with numerous technologies, and
    - Are easily adjustable to custom environments.
  - Flexible:
    - Workflow parameterization is built-in leveraging the Jinja templating engine.
    - Workflows built for finite batch and are triggerable through CLI and REST API.

Airflow is not for?

- It is not a streaming solution such as Apache Kafka.
- It was not built for infinitely running event-based workflows.

Apache Airflow™: Architecture

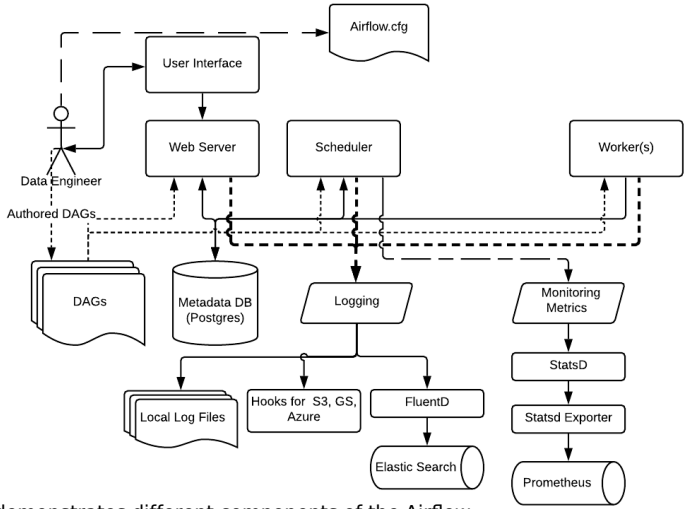


Figure: This example demonstrates different components of the Airflow.
Source: https://airflow.apache.org/docs/apache-airflow/stable/index.html

Apache Airflow™: Hello World

```python
from datetime import datetime

from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:
# Tasks are represented as operators
hello = BashOperator(task_id="hello", bash_command="echo hello")

@task()
def airflow():
print("airflow")

# Set dependencies between tasks
hello >> airflow()
```

Apache Airflow™: Why Airflow?

- Airflow is the tool for workflows defined as Python code which means:
  - Workflows can be stored in version control,
  - Workflows can be developed by multiple users simultaneously,
  - Tests can be written to validate the functionality,
  - Its components are reusable and extensible to a wide range.
  - Installable from PyPI:

```
pip install "apache-airflow[celery]==2.8.0" --constraint
"https://raw.githubusercontent.com/apache/airflow/constraints-2.8.0/constraints-3.8.txt"
```

  - Support for Python and Kubernetes versions[47].

47 More details: https://airflow.apache.org/docs/apache-airflow/stable/installation/supported-versions.html

Apache Airflow™: Security

- Airflow Security Model
  - Model to help users make decisions on how to deploy and manage Airflow.
- Airflow's Security Policy[48]
  - To know how to report security vulnerabilities and how to handle them.
- Secrets[49]
  - Variables configurations used that contain particularly sensitive information,
  - These variable and connection configurations are kept secret during operation.
  - These follows:
    - Encryption at rest,
    - Using external secret stores, and
    - Masking of sensitive data.

48 More details: https://github.com/apache/airflow/security/policy

49 More details: https://airflow.apache.org/docs/apache-airflow/stable/security/secrets/index.html

Apache Airflow™: How to define a DAG?

1. Create a DAG definition file.
   - Import modules[50] (same as in Python program),
2. Default arguments "default_args"[51]
   - A dictionary of default parameters that is used when creating tasks.
3. Instantiate a DAG
   - It is a DAG object to nest tasks into.
4. Operators
   - An operator defines a unit of work for Airflow to complete.
   - This helps to visualize task dependencies in DAG code.
5. Tasks
   - Tasks determines how to execute the operator's work within a DAG.
   - A task must be instantiated to use an operator in a DAG.

50 Modules: https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/modules_management.html

51 Arguments: https://airflow.apache.org/docs/apache-airflow/stable/_modules/airflow/example_dags/tutorial.html

Apache Airflow™: Example of a DAG?

```python
import textwrap; from datetime import datetime, timedelta;  ## Modules

from airflow.models.dag import DAG  ## The DAG object; needed to instantiate a DAG

from airflow.operators.bash import BashOperator ## Operators; needed this to operate!

with DAG(
"tutorial", ## DAG name
# These args get passed on to each operator(Op) \& can be overridden on a per-task basis during Op
     ↪ initialization
default_args={"depends_on_past": False, "email": ["airflow@example.com"], "email_on_failure": False, ...},
description="A simple tutorial DAG", schedule=timedelta(days=1), start_date=datetime(2021, 1, 1),
     ↪ catchup=False,
tags=["example"],
) as dag:

## t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator( task_id="print_date", bash_command="date", )
t2 = BashOperator( task_id="sleep", depends_on_past=False, bash_command="sleep 5", retries=3, )
t3 = BashOperator( task_id="templated", depends_on_past=False, bash_command=templated_command, )
# Task dependency
t1 >> [t2, t3]
```

Apache Airflow™: Templating with Jinja

```
templated_command = textwrap.dedent(
"""
{% for i in range(5) %}
echo "{{ ds }}"
echo "{{ macros.ds_add(ds, 7)}}"
{% endfor %}
"""
)

t3 = BashOperator(
task_id="templated",
depends_on_past=False,
bash_command=templated_command,
)
```

Apache Airflow™: Adding documentation

```
t1.doc_md = textwrap.dedent(
"""\
#### Task Documentation
You can document your task using the attributes ‘doc_md‘ (markdown),
‘doc‘ (plain text), ‘doc_rst‘, ‘doc_json‘, ‘doc_yaml‘ which gets
rendered in the UI’s Task Instance Details page.
![img](http://montcs.bloomu.edu/~bobmon/Semesters/2012-01/491/import%20soul.png)
**Image Credit:** Randall Munroe, [XKCD](https://xkcd.com/license.html)
"""
)

dag.doc_md = __doc__  # providing that you have a docstring at the beginning of the DAG; OR
dag.doc_md = """
This is a documentation placed anywhere
"""  # otherwise, type it like this
```

Extra: Snakemake Workflow

- It is a console based tool designed mainly for HPC clusters.
- Like Apache Airflow, Snakemake is another open sourced workflow management tool.
- Below is a simple example of it, where it needs rule, input and output parameters only.

```
1   rule all:
2   input:
3   "final_output.txt"
4
5   rule process_data:
6   input:
7   "raw_data.txt"
8   output:
9   "processed_data.txt"
10  shell:
11  "python process_data.py {input} > {output}"
```

Extra: Apache Airflow™ Vs Bash Snakemake

A few information about **Apache Airflow** Vs **Bash Snakemake**[52] Workflow

1. Workflow Definition
   - Airflow
     - Uses Python code to define workflows,
     - Provides more programmatic flexibility.
   - Snakemake
     - Uses a declarative DSL in a Snakefile.
     - Designed for HPC clusters.
2. Execution Model
   - Airflow
     - Emphasizes dynamic scheduling, monitoring, and orchestration.
   - Snakemake
     - Focuses on data dependencies and parallelization.

52 source: https://snakemake.readthedocs.io/en/stable/

Extra: Apache Airflow™ Vs Bash Snakemake Contd..

A few information about **Apache Airflow** Vs **Bash Snakemake** Workflow

3. User Interface:
   - Airflow
     - Provides a web-based UI for DAG visualization, monitoring, and management.
   - Snakemake
     - Doesn't include a built-in web-based UI.
4. Use Cases:
   - Airflow
     - Versatile and suitable for various use cases,
     - including ETL processes and workflow orchestration.
   - Snakemake
     - Often preferred for data analysis workflows,
     - Especially in bioinformatics.

Outline

Drill [10, 29, 30]

- Software framework for data-intensive distributed applications
- Data model: relational (ANSI SQL !) + schema-free JSON
- Analyse data in-situ without data movement
  - Execute one query against multiple NoSQL datastores
  - Datastores: HBase, MongoDB, HDFS, S3, Swift, local files
- Features
  - REST APIs
  - Columnar execution engine supporting complex data
  - Locality-aware execution
  - Cost-based optimizer pushing processing into datastore
  - Runtime compilation of queries

```
# Different datastores, localstorage, mongodb and s3
SELECT * FROM dfs.root.‘/logs‘;
SELECT country, count(*) FROM mongodb.web.users GROUP BY country;
SELECT timestamp FROM s3.root.‘users.json‘ WHERE user_id = ’max’;

# Query JSON: access the first students age from private data (a map)
SELECT student[0].private.AGE, FROM dfs.’students.json’;
```

Cloudera Impala [25, 26]

- Enterprise analytic database
  - Utilizes HDFS, HBase and Amazon S3
  - Based on Google Dremel like Apache Drill
- Written in C++, Java
- Massively-parallel SQL engine
  - Supports HiveQL and subset of ANSI-92 SQL
- Uses LLVM to generate efficient code for queries

Apache Metamodel [43]

- Provides a Java based SQL-alike interface to various data sources
  - CSV, SQL dbs, JSON, HBase, MongoDB

Query [43]

```
DataContext dataContext = DataContextFactory.create[TypeOfDatastore](...);
DataSet dataSet = dataContext.query()    .from("libraries").select("name").where("language").eq("Java").and("enhances_data_access").eq(true).execute();
```

Update [43]

```
dataContext.executeUpdate(new UpdateScript() {
    public void run(UpdateCallback callback) {
        // CREATE a table
        Table table = callback.createTable("contributors")
            .withColumn("id").ofType(INTEGER).withColumn("name").ofType(VARCHAR).execute();

        // INSERT INTO table
        callback.insertInto(table).value("id", 1).value("name", "John Doe").execute();
        callback.insertInto(table).value("name", "Jane D.").execute();

        // UPDATE table
        callback.update(table).value("name","Jane Doe").where("id").eq(2).execute();

        // DELETE FROM table
        callback.deleteFrom(table).where("id").eq(1).execute();
    }
});
```

Outline

Kafka [41]

- **Publish-subscribe** distributed messaging system
  - Producer publishes message for a given **topic**
  - Consumer subscribes to topic and receives data
  - Simple: Consumer has to remember its read position (**offset**)
- A data source for Storm, HBase, Spark, ...
- Use cases – support data ingestion:
  - GPS data from truck fleet, sensor data
  - Error logs from cluster nodes, web server activity
- Features
  - Parallel, fault-tolerant server system (a server is called **broker**)

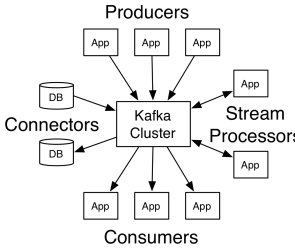


Figure: Source: [42]

Solr [10, 31]

- Full-text search and indexing platform
- REST API: index documents and query via HTTP
  - Query response in JSON, XML, CSV, binary
- Features
  - Data can be stored on HDFS
  - High-availability, scalable and fault tolerant
  - Distributed search
  - Faceted classification: organize knowledge into a systematic order using (general or subject-specific) semantic categories that can be combined per classification entry [10]
    - Examples: country, color, size, product type
  - Geo-spatial search
  - Caching of queries, filters and documents
- Uses lucene library for search
- Very similar: Elasticsearch [33], http://solr-vs-elasticsearch.com/

Example Query [32]

Identifying available facets terms and number of docs for each

```
curl http://localhost:8983/solr/gettingstarted/select?wt=json&indent=true&q=*:*&rows=0&facet=true& facet.field=manu_id_s
```

Response

```
{
  "responseHeader":{
    "status":0,
    "QTime":3,
    "params":{ /* Parameters of the query */
      "facet":"true", "indent":"true", "q":"*:*", "facet.field":"manu_id_s", "wt":"json",
      "rows":"0"}},
  "response":{"numFound":2990,"start":0,"docs":[]}, /* number of documents found */
  "facet_counts":{
    "facet_queries":{},
    "facet_fields":{ /* the available facets and number of documents */
    "manu_id_s":["corsair",3, "belkin",2, "canon",2, "apple",1, "asus",1, "ati",1, "boa",1, "dell",1, "eu",1, "maxtor",1, "nor",1, "uk",1, "viewsonic",1,
          ↪ "samsung",0]},
    "facet_dates":{},
    "facet_ranges":{},
    "facet_intervals":{}}
}
```

Outline

Mahout [34]

- Framework for scalable machine learning
  - Collbarorative filtering
  - Classification
  - Clustering
  - Dimensionality reduction
  - Recommender
    - history: user purchases + all purchases ⇒recommendations (user)
- Computation on Spark, MapReduce, H2O engines [36]
  - Can also use a single machine without Hadoop
  - Algorithm availability depends on the backend
- Bindings for Scala language [35]
  - Provide distributed BLAS, Row Matrix (DRM)
  - R-like DSL embedded in Scala
  - Algebraic optimizer

Recommender Architecture

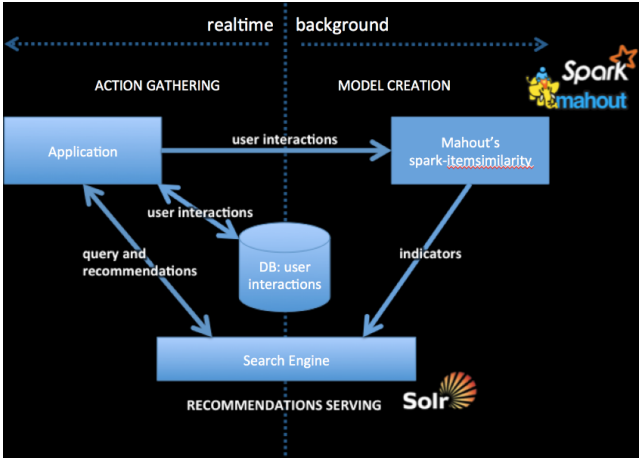


Figure: Source: [36]

1. Collect user interactions n x (user-id, item-id)
2. Learning:
   1. Itemsimilarity creates item, list-of-similar-items
   2. Store those tuples in the search engine
3. Query search engine with n latest user interactions
4. If they occur in the list-of-similar-items, recommend item

TensorFlow [44]

- Platform independent opens-source library for machine learning
- GPU accelerated
- Developed by GoogleBrain
- Tensor: Multidimensional array with rank dimensions
- Workflow
  - Define tensors (or placeholders)
  - Build execution graph based on equations and optimizers (math)
  - Start execution (bind dummy variables)
- Big community
- PyTorch alternative from Facebook with dynamic graph creation
- Nowadays typically used with Keras that provides high-level NN-architectures

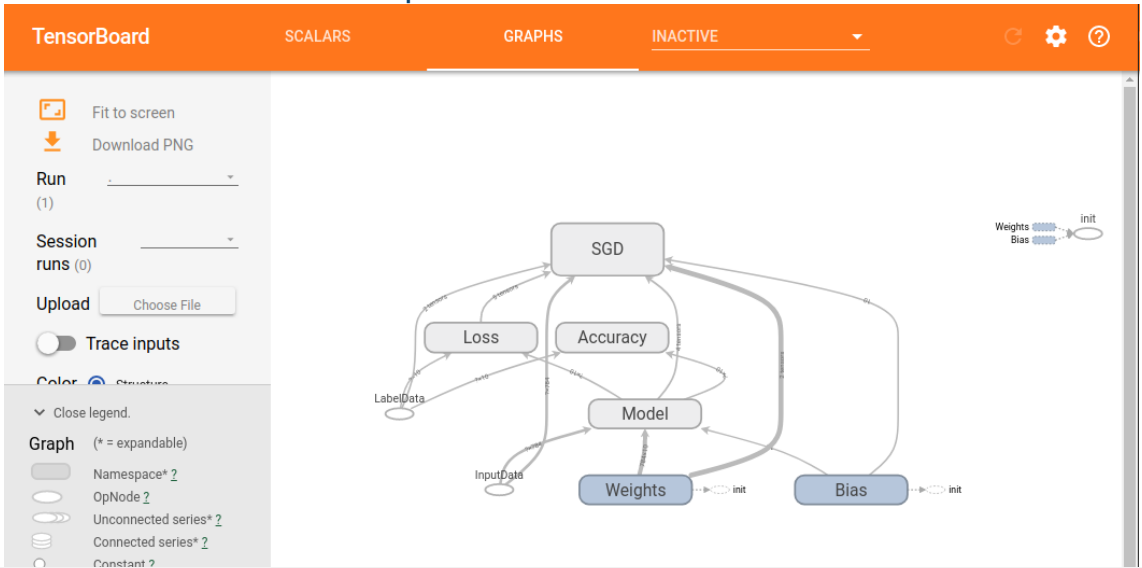
TensorFlow 1.X: Example for MNIST Dataset [47]

```
# Here build a neural network of a linear classifier to estimate classes
import tensorflow as tf
# Create two tensors holding features and labels for MNIST
# 28x28 images, number is unknown (None)
x = tf.placeholder(tf.float32, [None, 784], name='Feature') # name is useful in the GUI
# Create tensor to recognize 0-9 => 10 classes
y = tf.placeholder(tf.float32, [None, 10],  name='Label')

# Values to compute for a linear classifier, for each pixel compute chance it is class 0-9
W = tf.Variable(tf.zeros([784, 10]),        name='Weights')
b = tf.Variable(tf.zeros([10]),             name='Bias')

# Formulate solution with tensors
# Softmax returns a probability for each class that sums up to 1; here 10 classes
pred = tf.nn.softmax( tf.matmul(x, W) + b )
# Cost function for gradient descent; here is the cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(pred), reduction_indices=1))
# Learning algorithm to train the network
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Start execution, assume features and labels have been loaded
sess.run([optimizer, cost], feed_dict={x: features, y: labels})
```

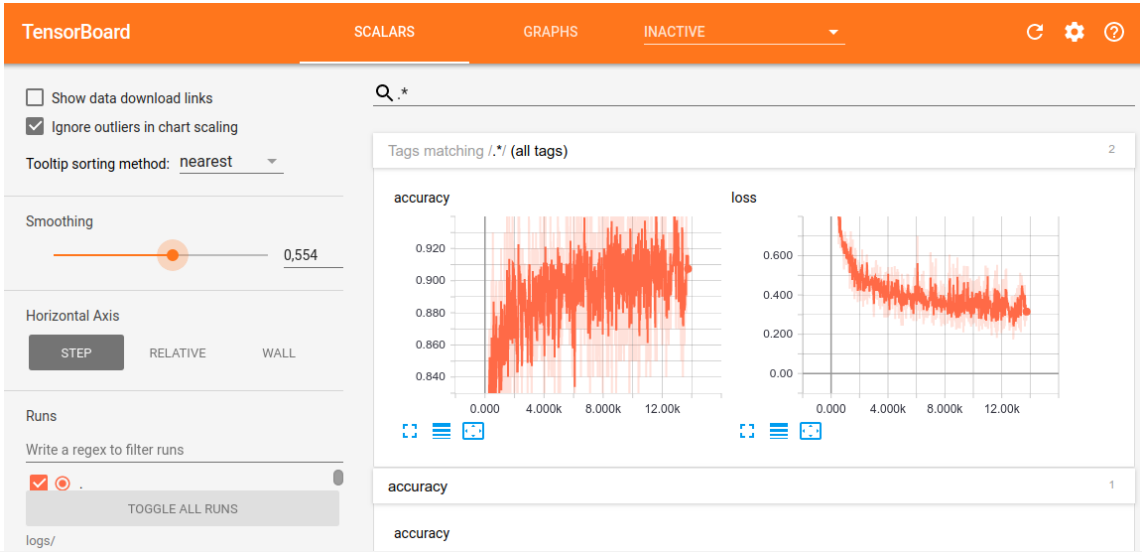
TensorBoard [45]

- Browser visualization for Tensorflow
  - Model graph
    - Operations can be given names
    - Name scopes organize operations together
  - Results: scalar, events, histogram, multi-dimensional, audio, images, ...
  - e.g., visualize accuracy over training steps
  - Embeddings: a mapping from discrete objects, e.g., words, to vectors of real numbers
- Reads (binary) output from a log directory, can be multiple runs!
  - Uses protocol buffers for serialization
  - Analyze hyperparameter search

TensorBoard: Model Graph Visualization



TensorBoard: Result Visualization



Summary

- The Apache community is very active
- Software responsibilities:
  - Hadoop deployment and cluster management
  - Data management and provenance
  - Security
  - Analysis
  - Automation (scheduling, data ingestion)
- One goal: simple usage
  - Alternative user interfaces
  - Research of domain-specific languages (XML based or language embedded)
- Many software packages are used but still in Apache incubator (beta)

Bibliography I

10 Wikipedia

11 http://hortonworks.com/blog/introduction-apache-falcon-hadoop/

12 http://gethue.com/

13 http://falcon.apache.org/

14 http://falcon.apache.org/0.7/EntitySpecification.html

15 http://oozie.apache.org/

16 http://oozie.apache.org/docs/4.2.0/index.html

17 https://github.com/yahoo/oozie/wiki/Oozie-Coord-Use-Cases

18 http://sqoop.apache.org/

19 http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

20 http://slider.incubator.apache.org/

21 http://slider.incubator.apache.org/docs/slider_specs/application_package.html

22 https://knox.apache.org/

23 http://hortonworks.com/blog/apache-atlas-project-proposed-for-hadoop-governance/

24 http://ranger.incubator.apache.org/

25 https://www.cloudera.com/products/enterprise-data-hub.html

26 http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise.html

27 https://github.com/cloudera/impala

29 http://incubator.apache.org/drill/

30 https://drill.apache.org/docs/json-data-model/

31 http://lucene.apache.org/solr/features.html

Bibliography II

32 http://lucene.apache.org/solr/quickstart.html

33 http://solr-vs-elasticsearch.com/

34 http://mahout.apache.org/

35 http://www.scala-lang.org/

36 http://h2o.ai/

37 https://mahout.apache.org/users/recommender/quickstart.html

38 Free Ebook: https://www.mapr.com/practical-machine-learning

39 https://zeppelin.incubator.apache.org/

40 http://hortonworks.com/products/data-center/hdp/

41 http://hortonworks.com/apache/kafka/

42 http://kafka.apache.org/intro

43 http://metamodel.apache.org/

44 https://www.tensorflow.org/

45 https://github.com/tensorflow/tensorboard

46 https://www.tensorflow.org/get_started/mnist/beginners

47 https://www.tensorflow.org/get_started/get_started

48 https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.0/atlas-overview/content/apache_atlas_features.html

49 https://blog.dotmodus.com/what-is-apache-atlas-and-why-is-it-important/