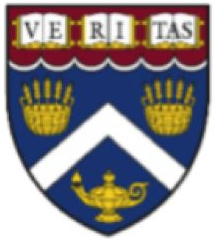
CSCI E-88 Principles Of Big Data Processing

Harvard University Extension, Fall 2019

Marina Popova



Lecture 13 Presentation Tier and Monitoring

Agenda

Odds and Ends :)

- Final Project heads up and What's next
- Presentation Tier architecture
- Big Data Visualization options
- Monitoring at Scale
- Pipeline orchestration/ scheduling

Admin Info

Final Project Milestones

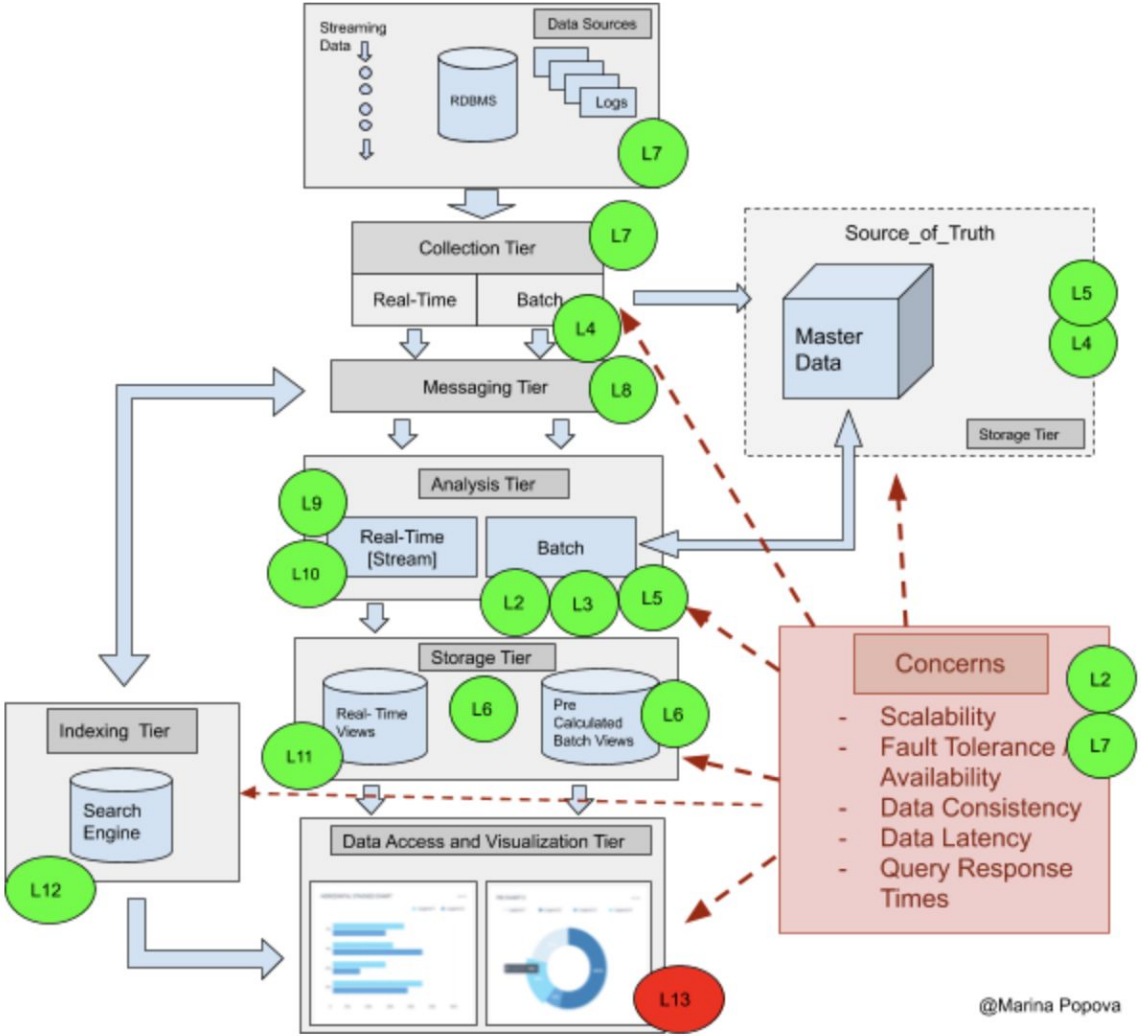
- Final Project Proposal: Due: Sun, Dec 1, noon EST; **Points: 50** - **no Late Days or extensions!**
- Final Project: Due: Sat, Dec 14, midnight, EST - **no Late Days, no extensions!**
- Final Projects Presentations: Dec 77

Final Project Deliverables:

- Final Project Proposal
- 3 min YouTube video
- Final Project Document
- Full source code/ config artifacts/ samples of input data

Final Project Q&A Lab: TBD

Where Are We?


Streaming Data
Data Sources
RDBMS
Logs
L7
Collection Tier
L7
Real-Time
Batch
L4
Source_of_Truth
Master Data
L5
L4
Storage Tier
Messaging Tier
L8
Analysis Tier
L9
Real-Time [Stream]
Batch
L10
L2
L3
L5
Storage Tier
Real- Time Views
L6
Pre Calculated Batch Views
L6
L11
Indexing Tier
Search Engine
L12
Data Access and Visualization Tier
L13
Concerns
L2
L7
- Scalability
- Fault Tolerance / Availability
- Data Consistency
- Data Latency
- Query Response Times
@Marina Popova

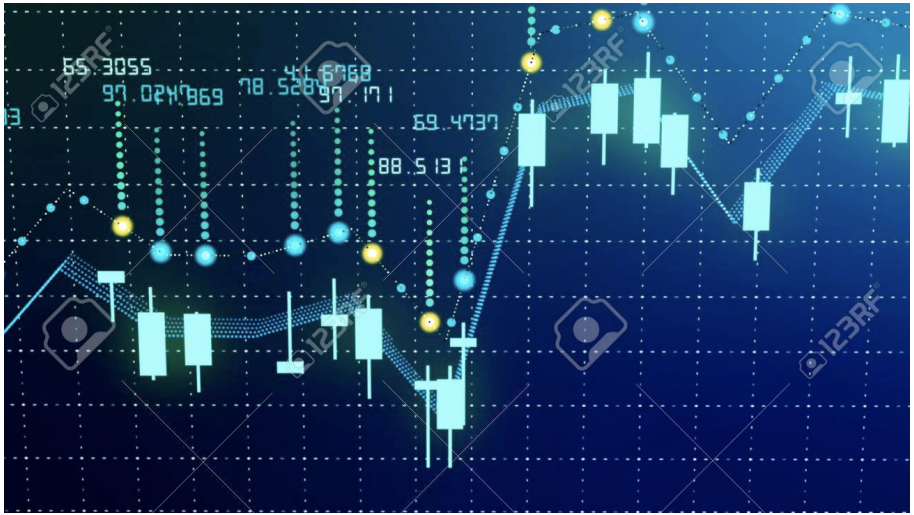
Where Are We?

What have we learned so far ?

- How to collect data
- How to model/store data in the Master Datastore
- How to process streaming and batch data and do analyses on "windowed" batches
- How to store results of the windowed analyses in RT data storage
- How to use specialized Search systems for complex analyses of the indexed data
- How to send/index the data into the Search systems

What's Next?

- How do I use the data I've collected/analysed/indexed so far?
- who will be visualizing/presenting my data?
- how complex this presentation has to be?



This is usually the responsibility of the Presentation Tier

Presentation Tier Architecture

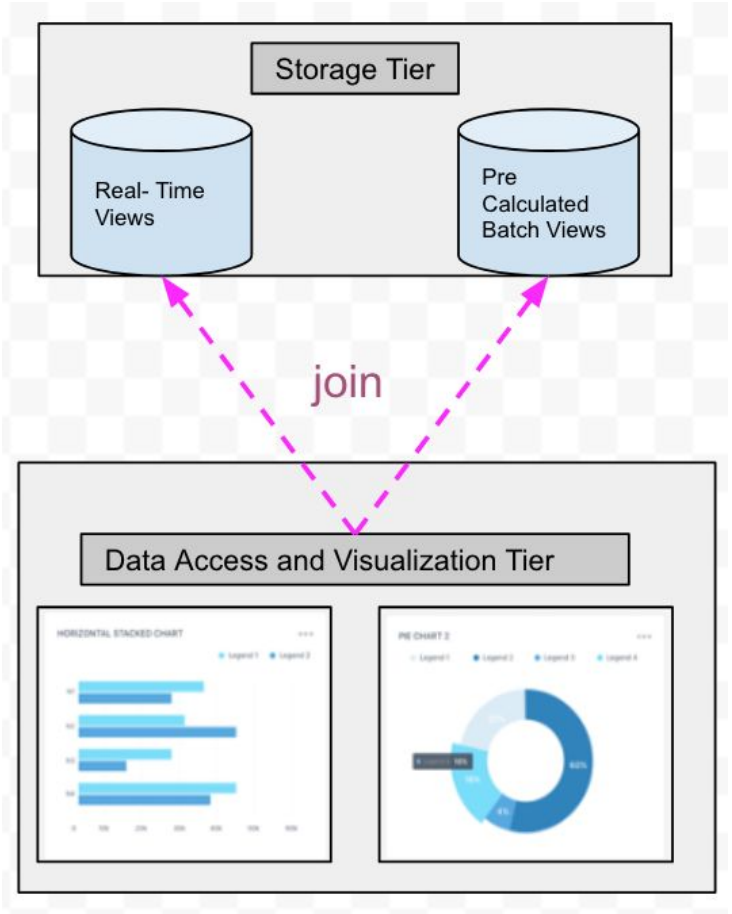
How do I use the data I've collected/analysed so far?

- In the case of Lambda-like Architecture - you have to combine Real-Time and Batch Views
- In Stream-only (Kappa-like) Architectures - you may use Real-Time Views directly

If your presentation requirements are simple and can be fulfilled by directly querying the RT/Batch views (aggregated data):

- a custom app with UI can query the RT+Batch storage and present the data
- for example, you can use something like HighCharts:

https://www.highcharts.com/docs/getting-started/your-first-chart



Presentation Tier Architecture

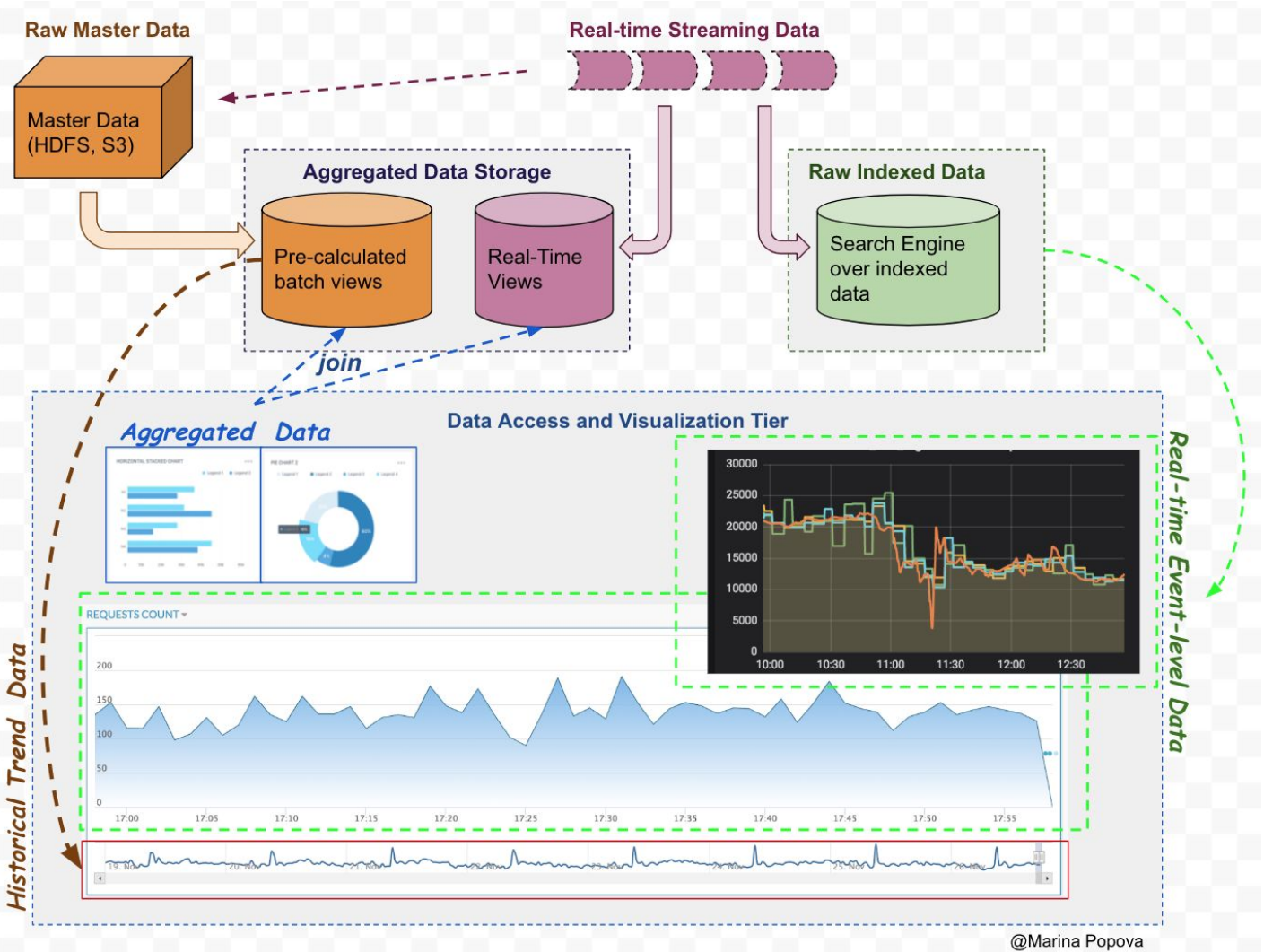
If you have more complex requirements for your presentation layer:

- Ad-hoc queries over the latest real-time data
- Real-time interaction with data (visualization)

---> have to use a full-blown Search system like ES or Solr

Most often, it is a combination of all or some of the above!

Presentation Tier Architecture



Presentation Tier Architecture

Indexed Data - main considerations:

- ingestion/indexing speed - has to keep up with the incoming data rate
- Required transformations of data for the Search System (for example, convert text events into JSON for ES)
- Amount of data that a Search System can store
- cost!

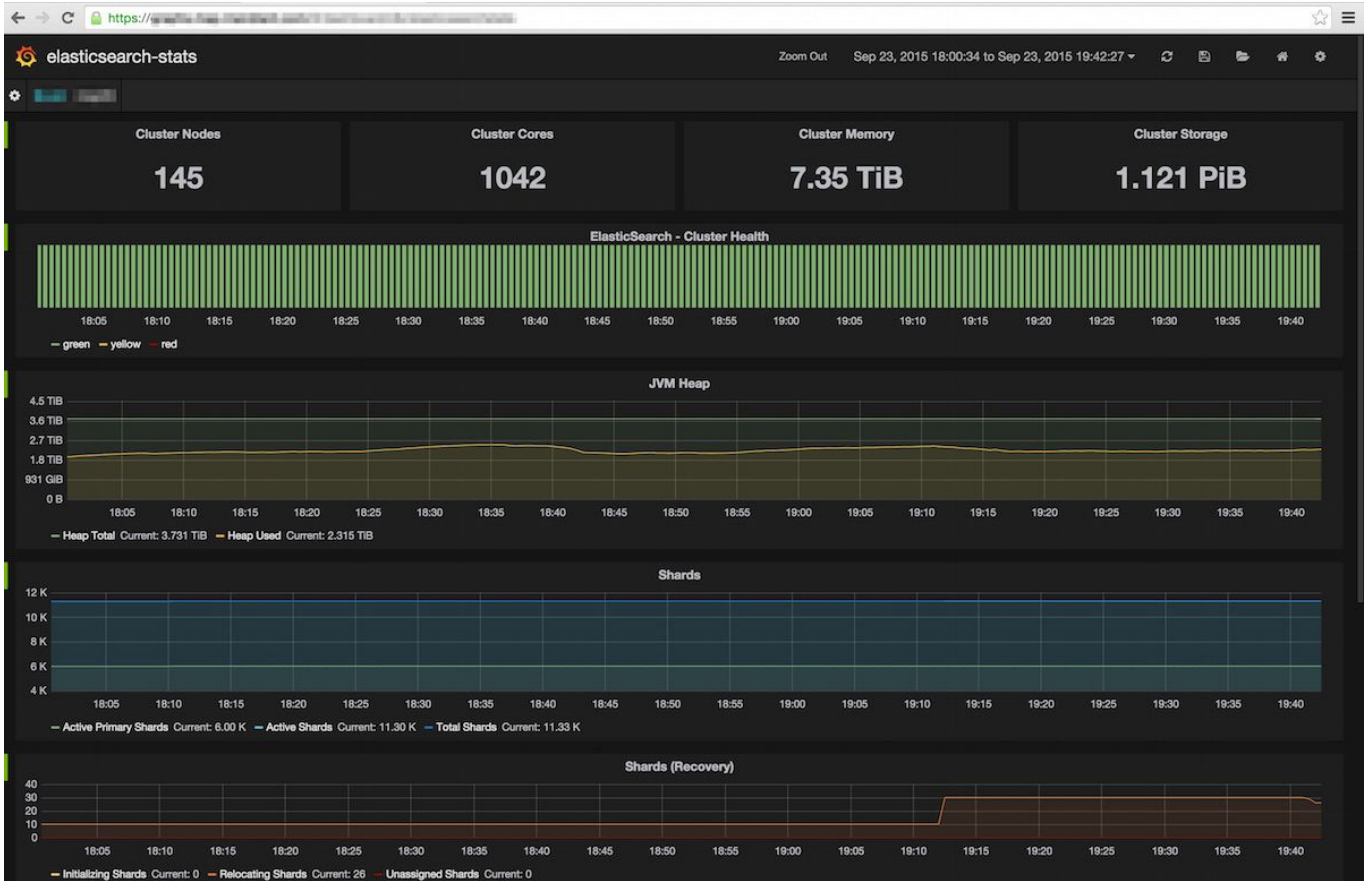
Why not store ALL data in the Indexed Storage?

- scaling/sizing/cost challenges

For example, for ElasticSearch:

- 1 shard: < ~ 5M of 1K events
- 1 node: < ~ 500-800 shards

Great post on scaling issues and solutions when scaling ES cluster to handle Petabyte-sized data: https://grey-boundary.io/field-notes-elasticsearch-at-petabyte-scale-on-aws/



One more Real-life use case of scaling ES to 500+ nodes:

https://underthehood.meltwater.com/blog/2018/02/06/running-a-400+-node-es-cluster/
https://underthehood.meltwater.com/blog/2018/11/05/optimal-shard-placement-in-a-petabyte-scale-elasticsearch-cluster/

The biggest issues:

- uneven/unpredictable shard allocations across servers
  "*Elasticsearch effectively assigned shards at random, and with 500+ nodes in a cluster it suddenly became very likely that too many of the 'hot' shards were placed on a single node, and such nodes quickly got overwhelmed.*"
- continuous cluster state changes
  - *New indices created and old indices dropping.*
  - *Disk watermark triggers due to indexing and other shard movements.*
  - *Elasticsearch randomly deciding that a node has too few/too many shards compared to the cluster average.*
  - *Hardware and OS-level failures causing new AWS instances to spin up and join the cluster. With 500+ nodes this happens several times a week on average.*
  - *New nodes added, almost every week because of normal data growth*
- *"cascading shard relocation storms"*

Presentation Tier - Visualization

How can all this data be visualized? - Many options!

- Custom applications (HigCharts/HighMap, other JS-based frameworks)
- Specialized indexed data - Kibana (for ES), Banana for Solr
- Tableau
- Graphana/ Prometheus
- Google Charts (nice integration with google Maps)
- D3.js - JS-based library
- R, Jupyter notebooks

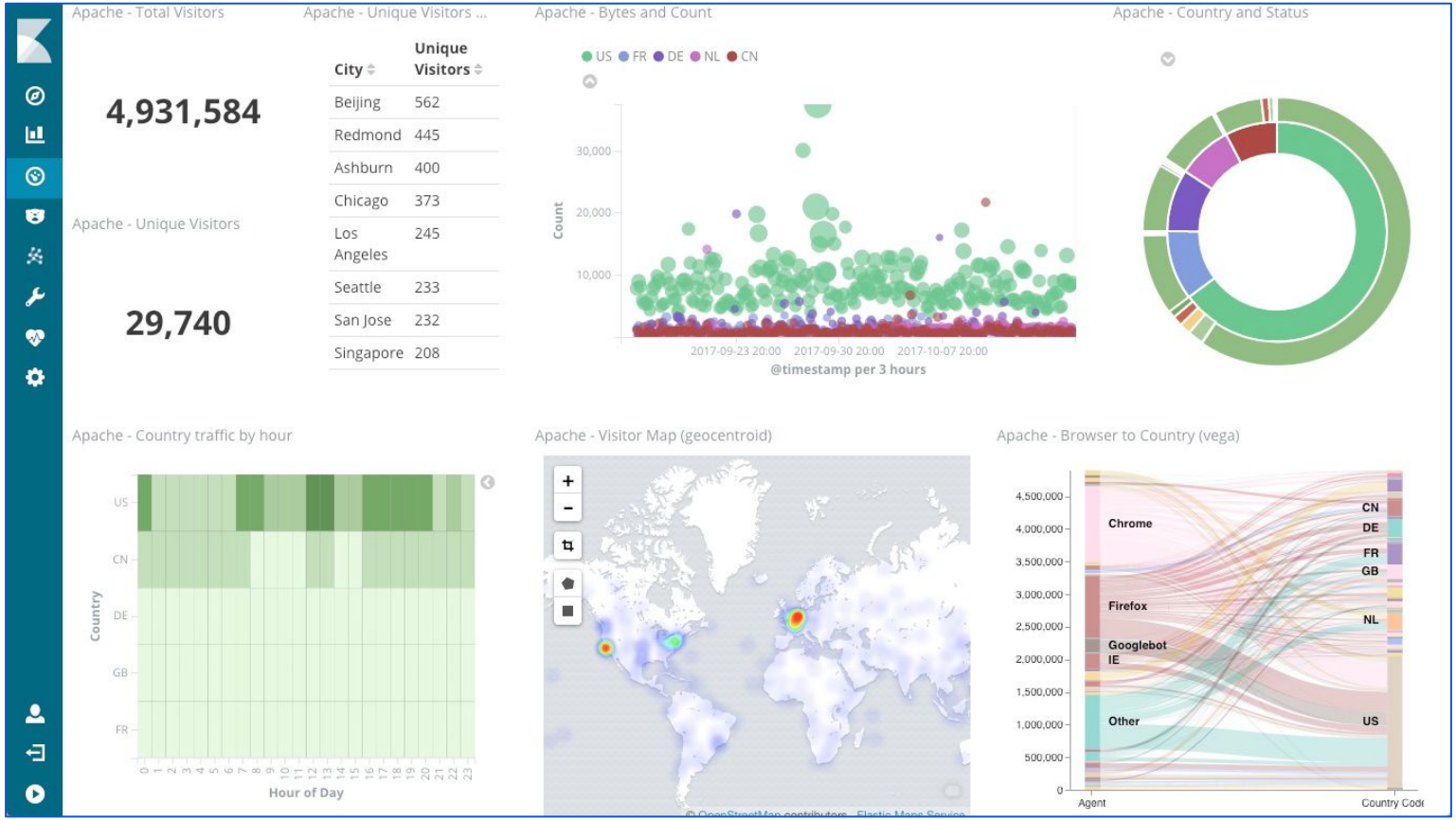
Presentation Tier - Visualization

How do you pick one?

- How much control over interactions/visualization do you need?
  - Build vs. Buy
- Integrated with your own apps or stand-alone?
- Supported data source
  - JSON/CSV, NoSQL, AWS S3 , GDP , HDFS ?
- Intended Audience and Platforms
  - Mobile/simple stats vs. full-featured complex analysis with dashboards
- Types of widgets and dynamics of data
  - static/relationship vs time-series
- User interaction
  - Ad-hoc parameter specification vs. pre-defined limited interactions
  - Access control

Presentation Tier: Visualization: Kibana

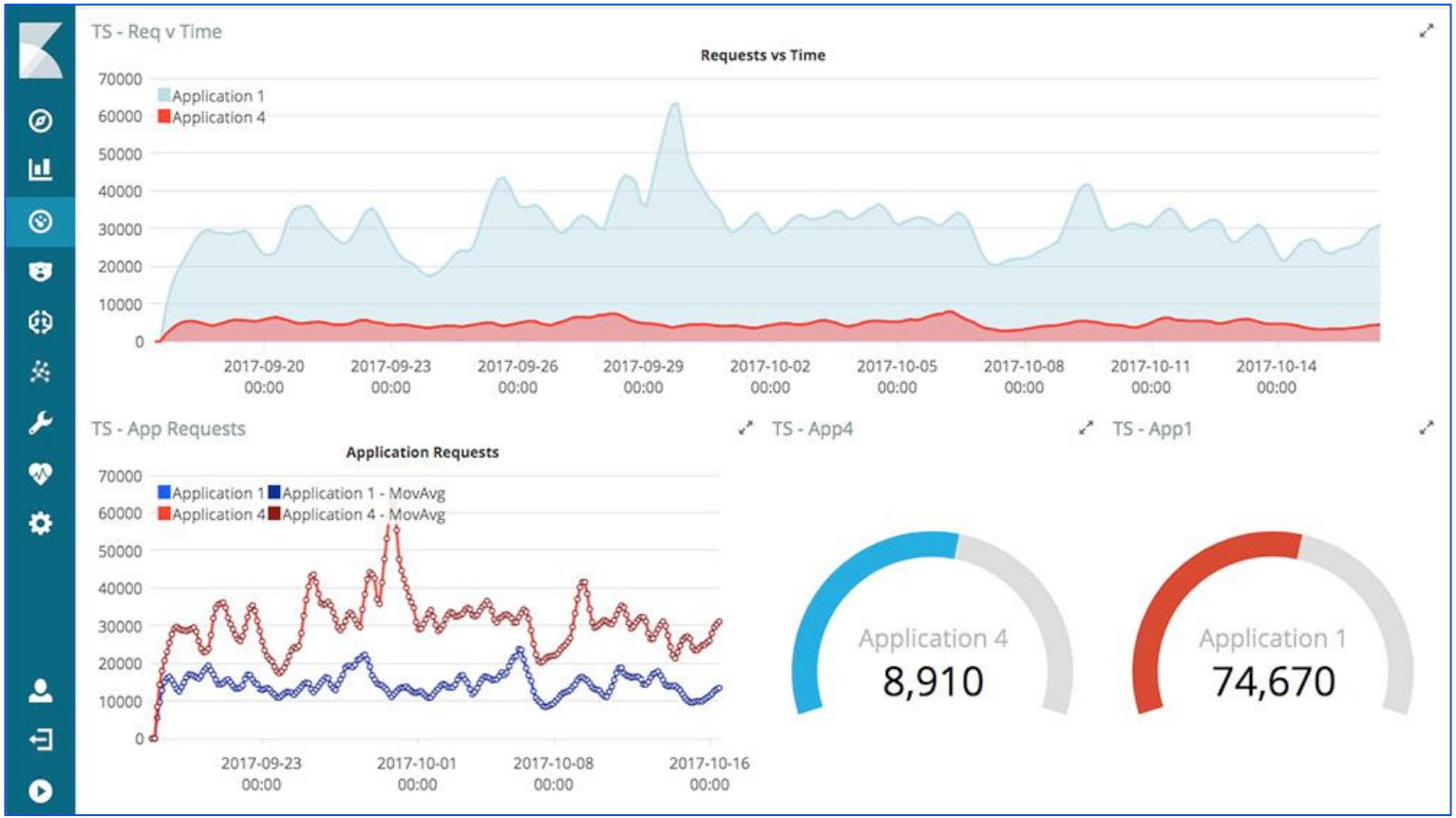
#1 choice for Elastic Stack https://www.elastic.co/products/kibana



@Marina Popova

Presentation Tier: Visualization: Kibana

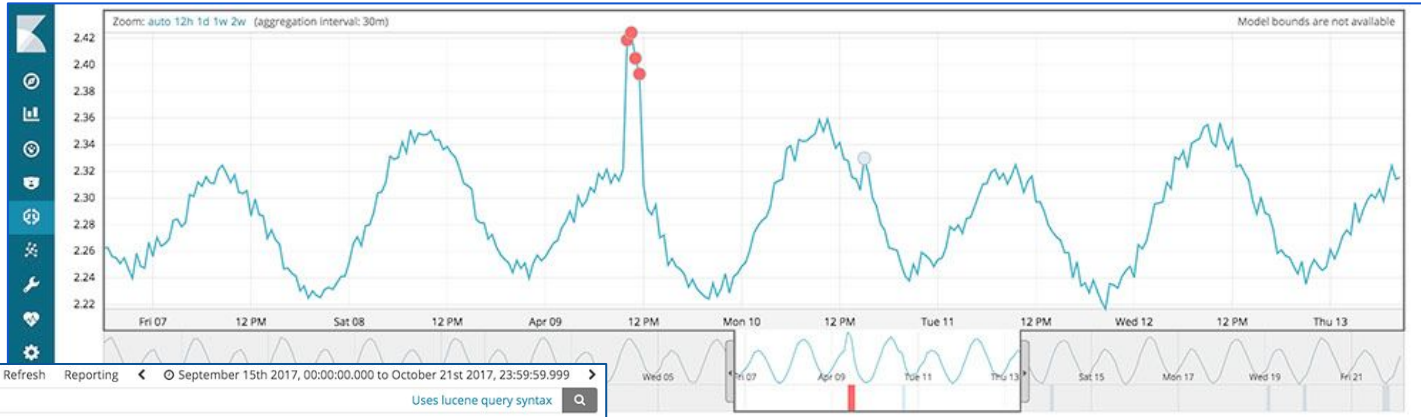
Time-series graphs and analytics:

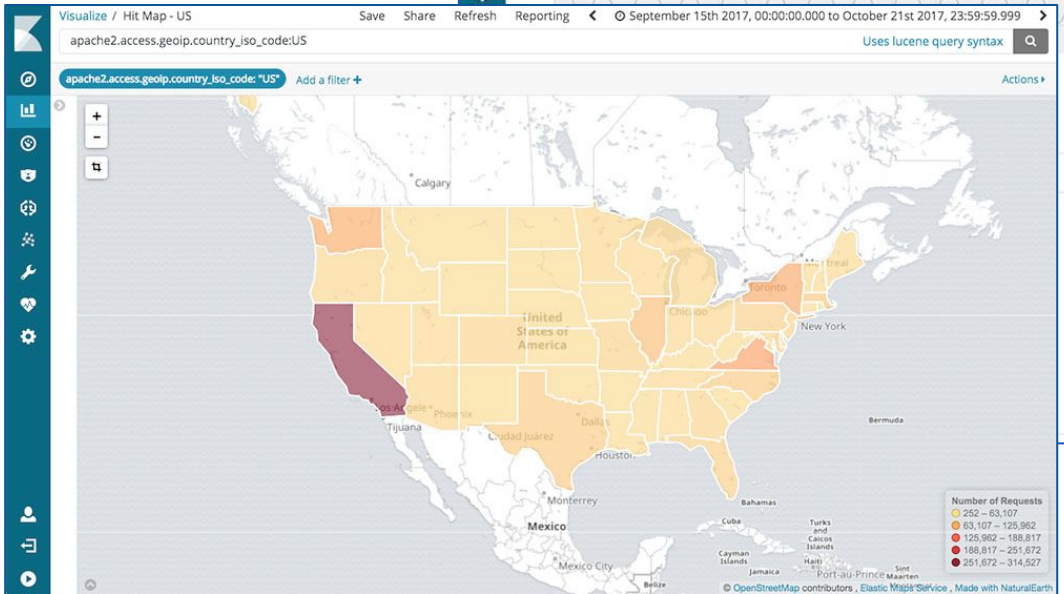


@Marina Popova

Presentation Tier: Visualization: Kibana

Geo, Alerts:





found for	influenced by	actual	typical	description	job ID
server_3	host: server_3 service: app_5	2.42113	2.31031	Unusually high	mm-job-3

@Marina Popova

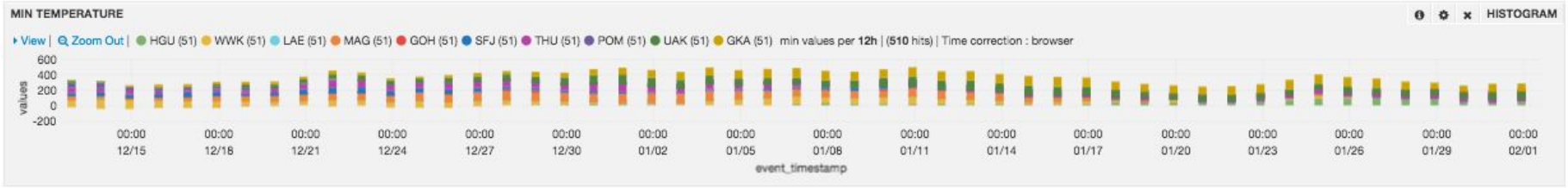
Presentation Tier: Visualization: Banana

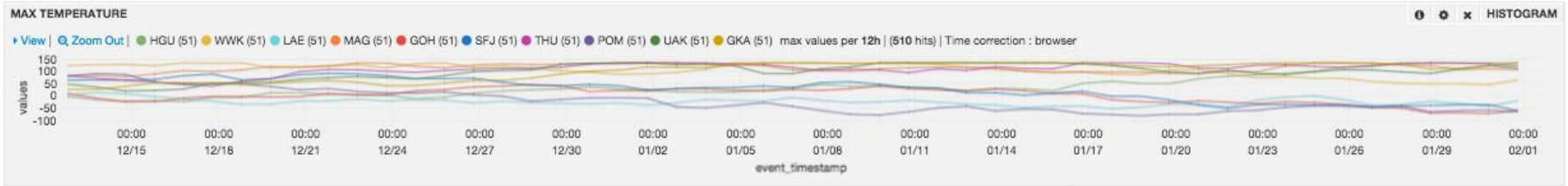
Banana - visualization of Solr-based data
https://github.com/lucidworks/banana

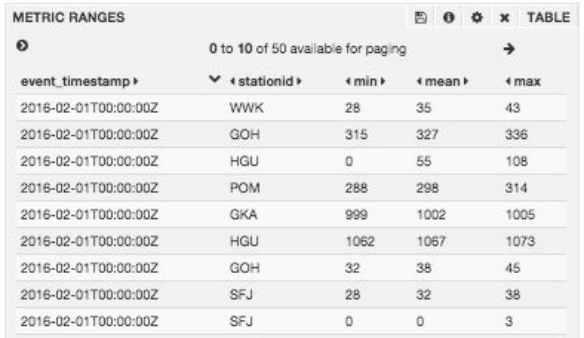
- was forked from Kibana
- has many panels and visual capabilities of Kibana (but not all)
- leverages D3.js for some panels

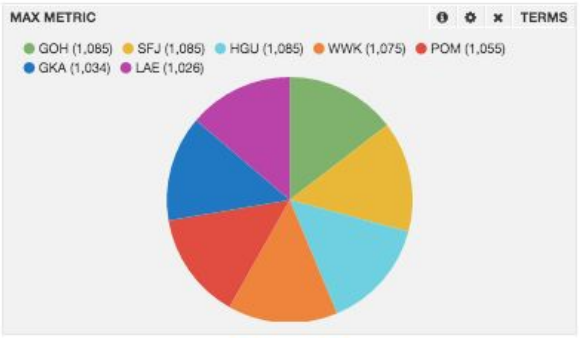
Example of using Cassandra and Solr together to add visualization to data stored in Cassandra:
https://medium.com/@carolinerg/visualizing-cassandra-solr-data-with-banana-b54bf9dd24c

Presentation Tier: Visualization: Banana









@Marina Popova

Presentation Tier: Visualization: Tableau

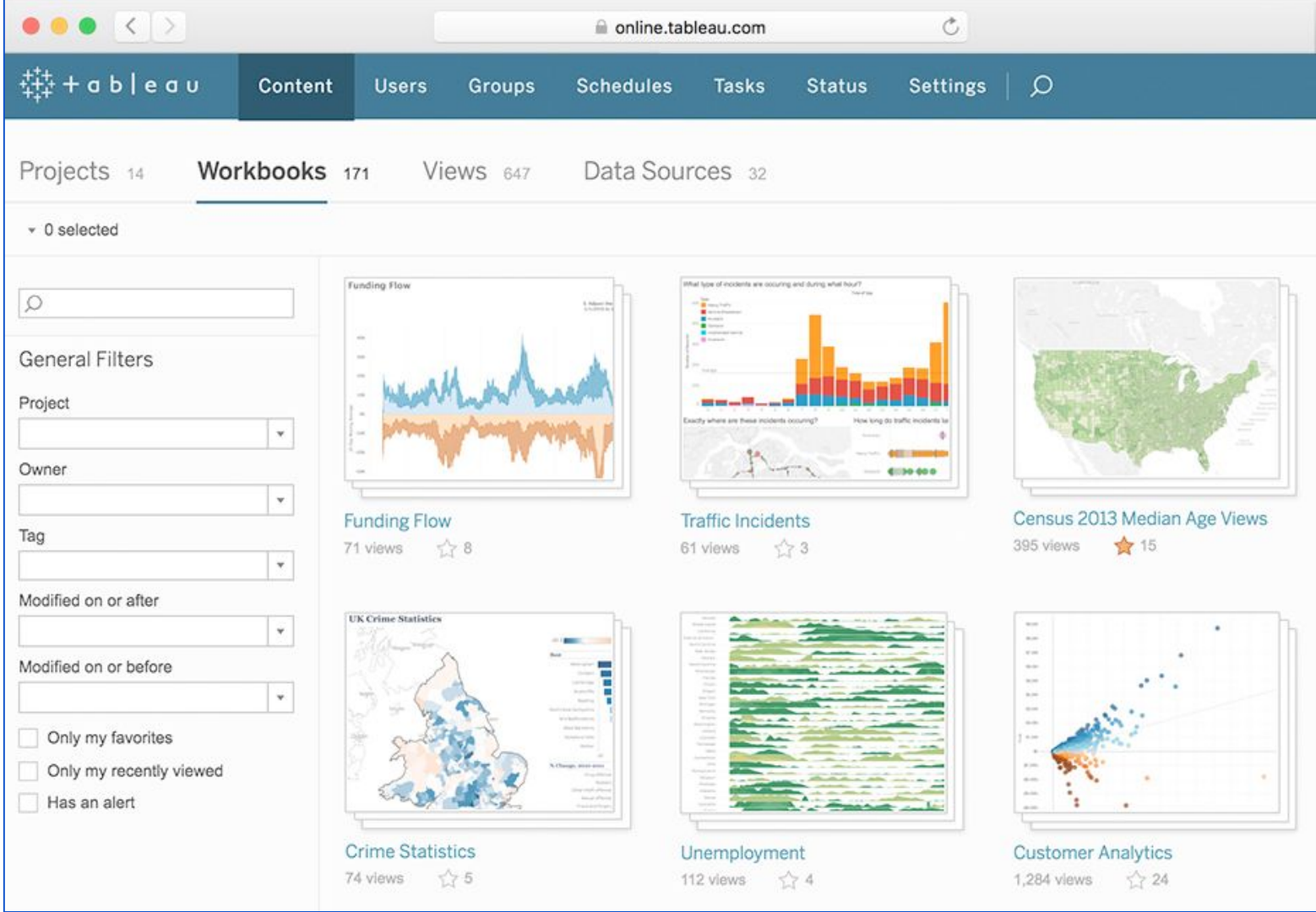
Tableau is a Data Visualisation tool that is widely used for Business Intelligence

- multiple profiles/ platform support
  - Desktop/ Mobile/ Cloud
  - Embedded (via JS API or REST calls)
  - Public Edition (limited, free)
- millions of visualization and interaction options !
- access control
- multiple deployment options:
  - AWS
  - on-premises
  - Google Cloud
  - Azure

- many data sources: Cloudera Hadoop, Oracle, AWS Redshift, cubes, Teradata, Microsoft SQL Server, and more

@Marina Popova

Tableau



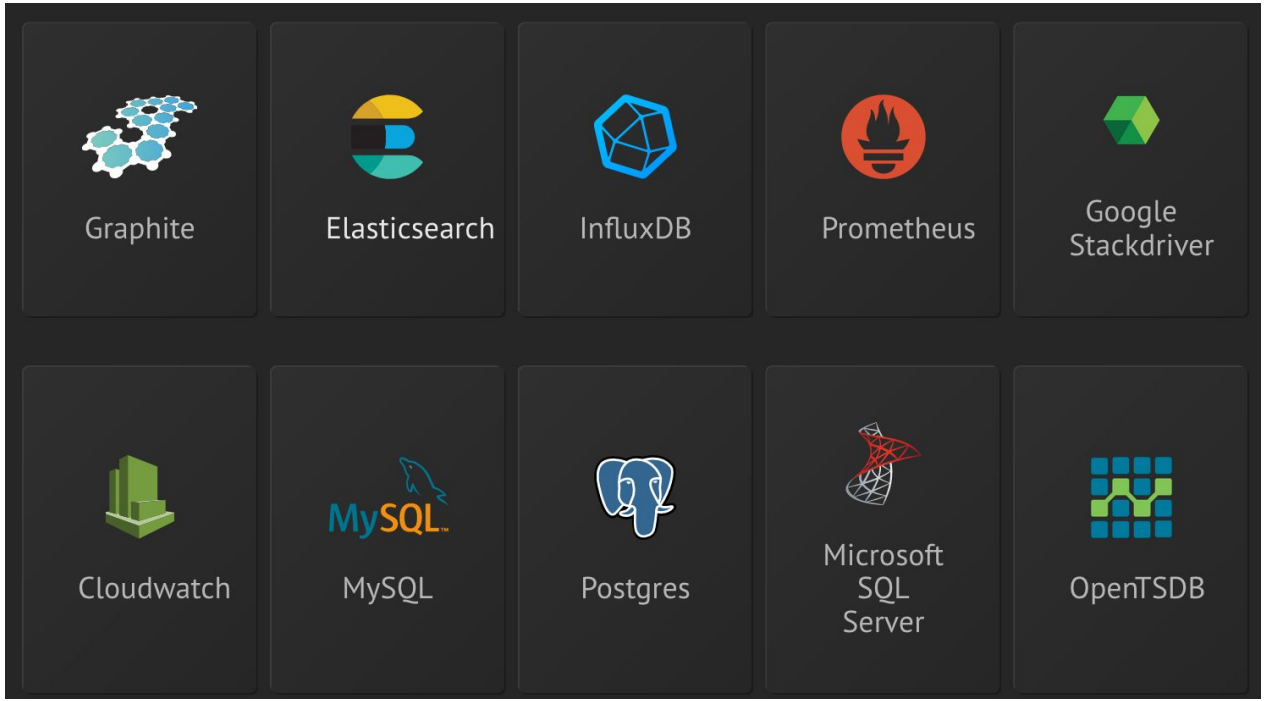
@Marina Popova

Presentation Tier: Visualization: Graphana

Grafana https://grafana.com/grafana – a professional data visualization and analytic tool that support up to 30 data sources, including AWS, Elasticsearch and Prometheus.

Presentation Tier: Visualization: Graphana

Data Sources:



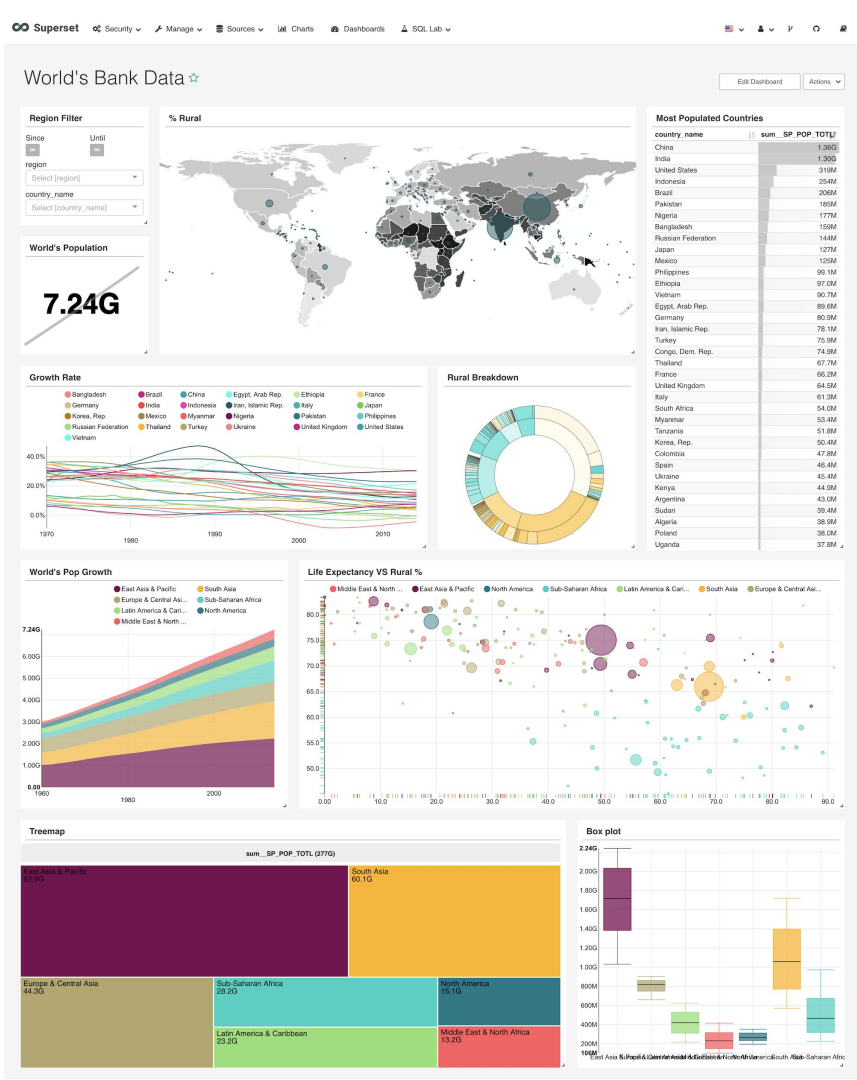
AirBnb/Apache Superset

Superset

http://superset.apache.org/index.html

Superset is a data exploration and visualization platform designed to be visual, intuitive and interactive. It consists of two primary interfaces:

- A Rich SQL IDE (Interactive Development Environment) enabling fast and flexible access of data
- A Data Exploration Interface that converts data tables into rich visual insights
- rich dashboarding options



@Marina Popova

AirBnb/Apache Superset

Main Features:

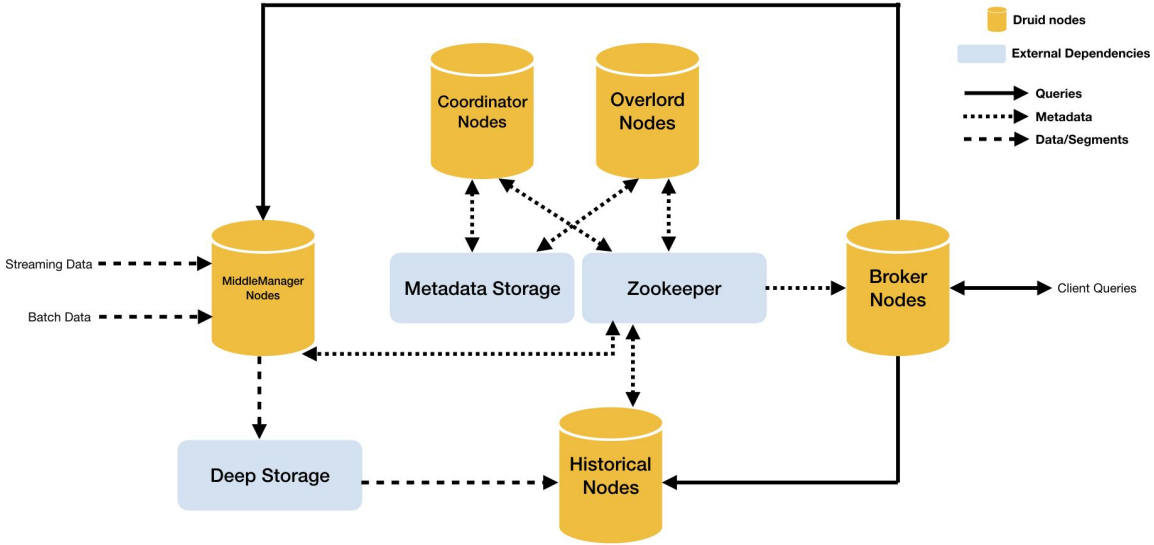
- Enterprise-ready authentication with integration with major authentication providers (database, OpenID, LDAP, OAuth & REMOTE_USER through Flask AppBuilder)
- An extensible, high-granularity security/permission model allowing intricate rules on who can access individual features and the dataset
- A simple semantic layer, allowing users to control how data sources are displayed in the UI by defining which fields should show up in which drop-down and which aggregation and function metrics are made available to the user
- Integration with most SQL-speaking RDBMS through SQLAlchemy
- Deep integration with Druid.io

supported DBs

- Amazon Athena
- Amazon Redshift
- Apache Drill
- Apache Druid
- Apache Hive
- Apache Impala
- Apache Kylin
- Apache Pinot
- Apache Spark SQL
- BigQuery
- ClickHouse
- Google Sheets
- Greenplum
- IBM Db2
- MySQL
- Oracle
- PostgreSQL
- Presto
- Snowflake
- SQLite
- SQL Server
- Teradata
- Vertica

Druid: Recap

Druid: Real-time Distributed Column Data Store



When querying Druid, Superset can query humongous amounts of data on top of real time dataset

Ingestion/Indexing Details

- incoming data is parsed and stored into Deep Storage (S3 or HDFS) in segments
- uses proprietary storage format
- columnar storage
- compression - optimized for columns
- segments are indexed (using Druid-specific indexing tecniques)
- meta-data info about each index is stored in the Metadata Storage (location of the segment, time range, etc.)

http://druid.io/docs/latest/design/index.html#architecture

@Marina Popova

When nothing existing works - DIY !

Netflix
https://medium.com/netflix-techblog/lumen-custom-self-service-dashboarding-for-netflix-8c56b541548c

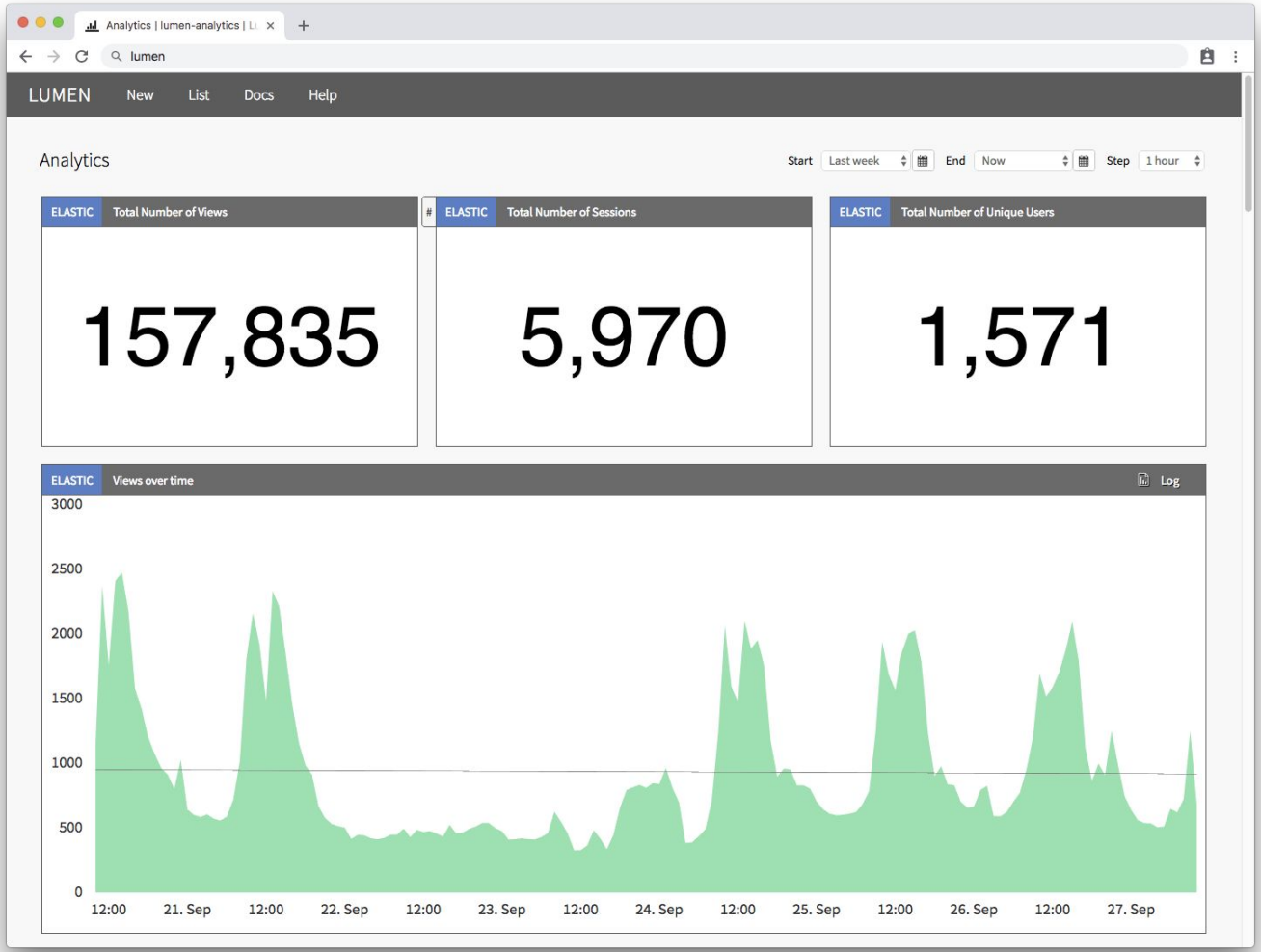
Challenges:

- many different [and huge - P-sized] data sources with different volumes/ response times/ latencies
- Users need to be able to construct dynamic dashboards on their own
- Must support different transport methods, such as streaming data over WebSockets, long-running queries that redirect to a cache upon completion, and standard RESTful APIs

Solution: custom Dashboarding solution - Lumen

"Lumen is a dashboarding platform that allows users to define JSON configuration files that are parsed at runtime in the browser to generate custom dashboards"

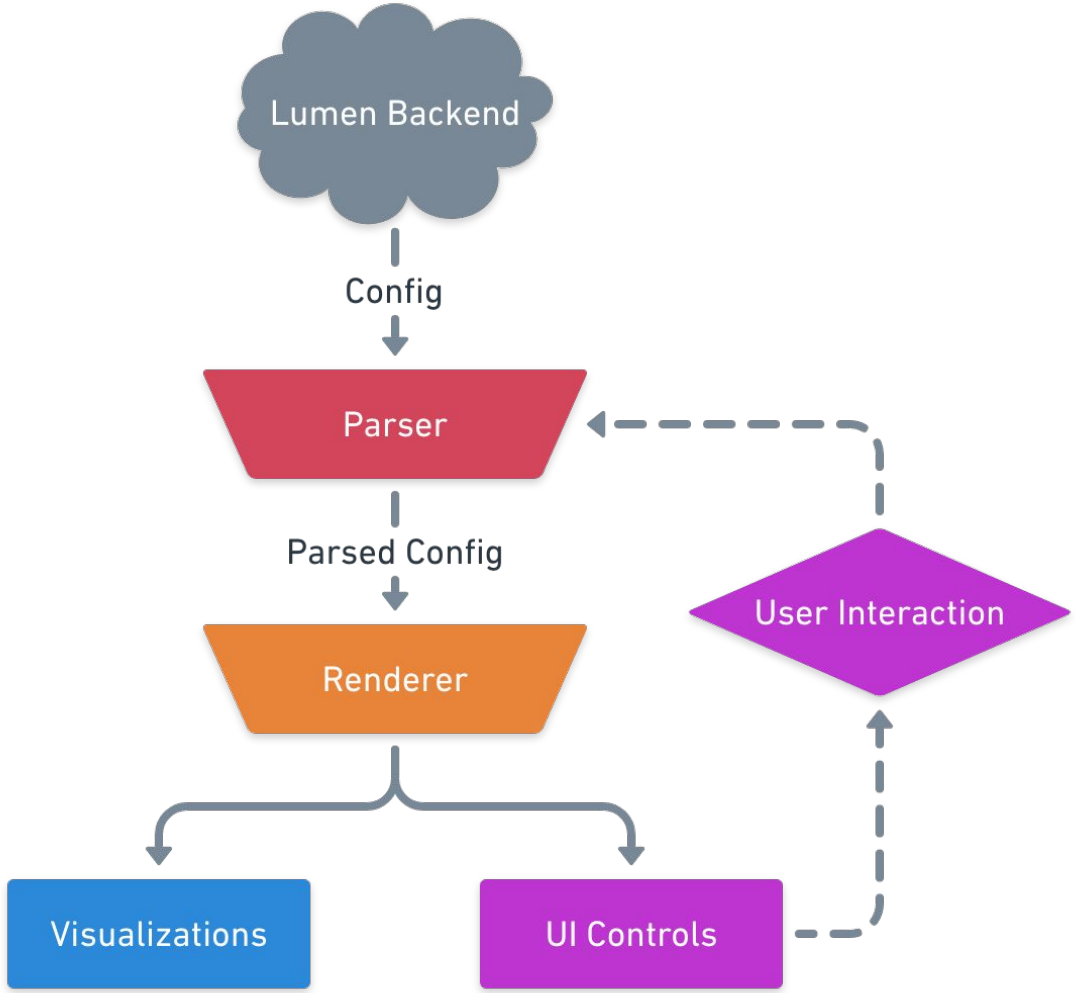
Netflix: Lumen



@Marina Popova

Netflix: Lumen

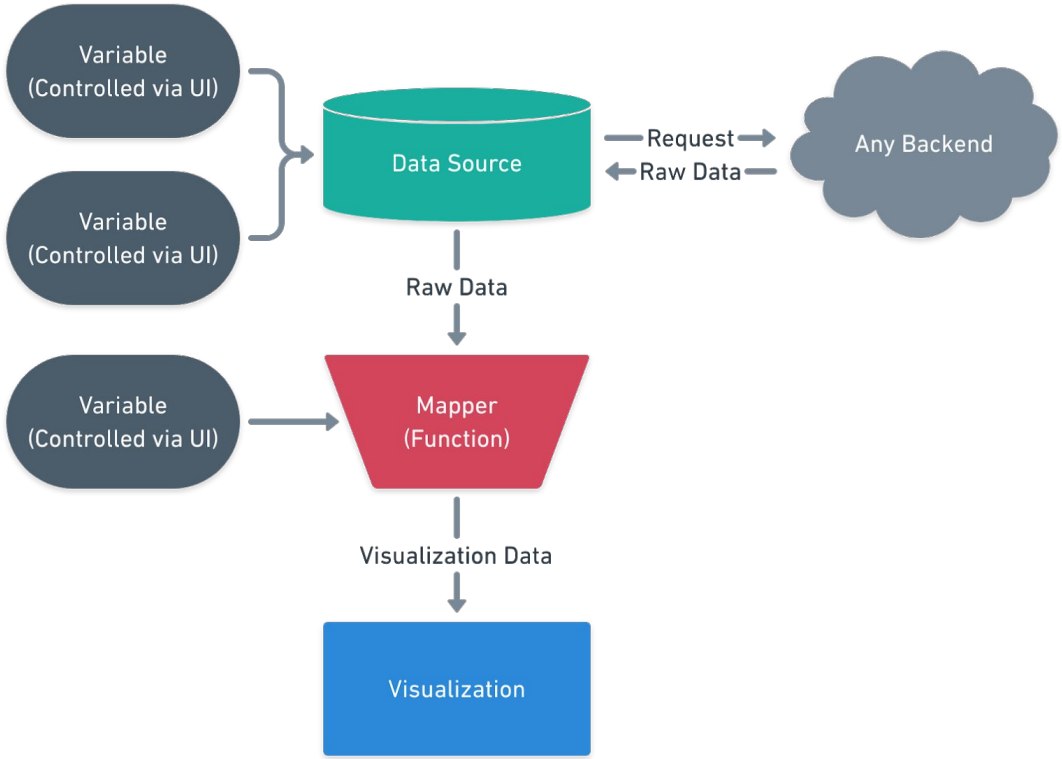
- Config is loaded from the backend store and then parsed into an internal data structure
- That data structure is then passed to a renderer, which generates the visualizations and UI controls for the dashboard
- Users can configure most aspects of their dashboards, including the visualizations shown, what data sources are used, and which UI controls to display



Netflix: Lumen

Lumene Dashboard is composed of the following components:

- **Data Source**: define how to load a certain type of data into the dashboard (REST , specific APIs, ...)
- **Visualization**: define how to visualized specific datasets
- **Mapper**: define how to transform the payload from he Data Source into Lumene's format
- **Variables**: can be substituted with dynamic values from config or user input



@Marina Popova

Moving On ...

Next: Monitoring !

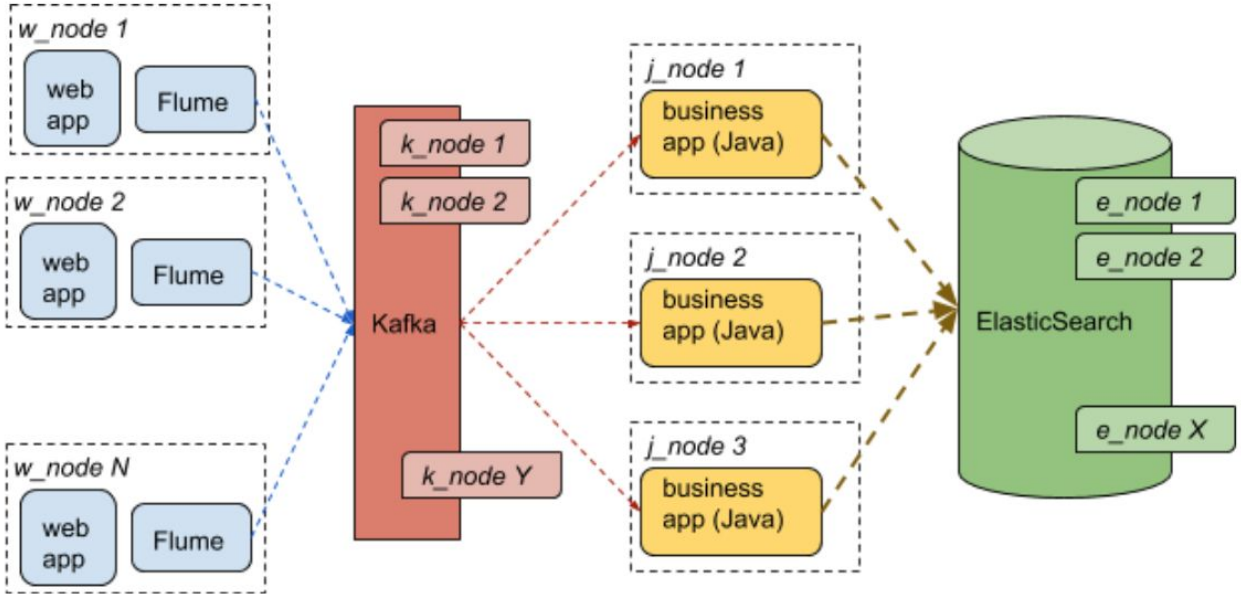
@Marina Popova

Monitoring of Big Data Processing Pipelines

Why monitoring ??

Lets consider a simple pipeline:

layer	# of nodes	# of apps
collection	10	20
kafka	5	5
Java apps	3	3
ES	15	15
Total	**33**	**43**



Monitoring of Big Data Processing Pipelines

what could possibly go wrong ??

@Marina Popova

Monitoring of Big Data Processing Pipelines

What could possibly go wrong ?? - Many things ....

Hardware/ Server side:

- any node can go down
- out of disk space or disk corruption
- memory utilization grows too high
- CPU utilization grows too high
- # of open files / TCP sockets/ etc. exceeds limits
- high IOPs

Application side:

- application process (JVM) dies
- out-of-memory or close to be
- connection leaking
- some (or all) threads within app die
- too many failures (event parsing, read/writes to DBs, etc.)

layer	# of nodes	# of apps
collection	10	20
kafka	5	5
Java apps	3	3
ES	15	15
Total	**33**	**43**

Monitoring of Big Data Processing Pipelines

How do we monitor ? By collecting and visualizing relevant metrics

There are a number of open-source and commercial tools such Amazon CloudWatch, Nagios, New Relic, Prometheus and others

Differentiating Factors:

- which metrics and from what stack (hardware/ software/ networking) they can collect
- operational modes: cloud-based/ on-premise/ SaaS
- supported data formats and communication protocols
- system overhead of the collection agents
- deployment automation
- visualization and alerting capabilities

Monitoring of Big Data Processing Pipelines

Hardware/ Server side:

Usually have to use some kind of a collection agent to collect and send metrics to a centralized server

Examples:

- **CollectD**: open-source; daemon, which collects basic system performance statistics over time and stores the data it collects in multiple formats; often used together with Cacti - to visualize the results
- **Ganglia** - a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It uses XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization; Ganglia is a BSD-licensed open-source project that grew out of the **University of California, Berkeley**
- **Nagios** - basic metrics collection
- **AWS CloudWatch**: uses CWAgent - can collect metrics from both EC2/AWS-managed servers as well as on-premises servers

Monitoring of Big Data Processing Pipelines

Application side:

- general process metrics - CPU, RAM, threads, I/O, swap
- application-specific exposed metrics:
  - JMX-based
  - instrumented using special agent libraries (like client libraries in Prometheus)

Prometheus:

- very versatile and widely used
- works with both server- and application- level metrics

Often used with Graphana for data visualization

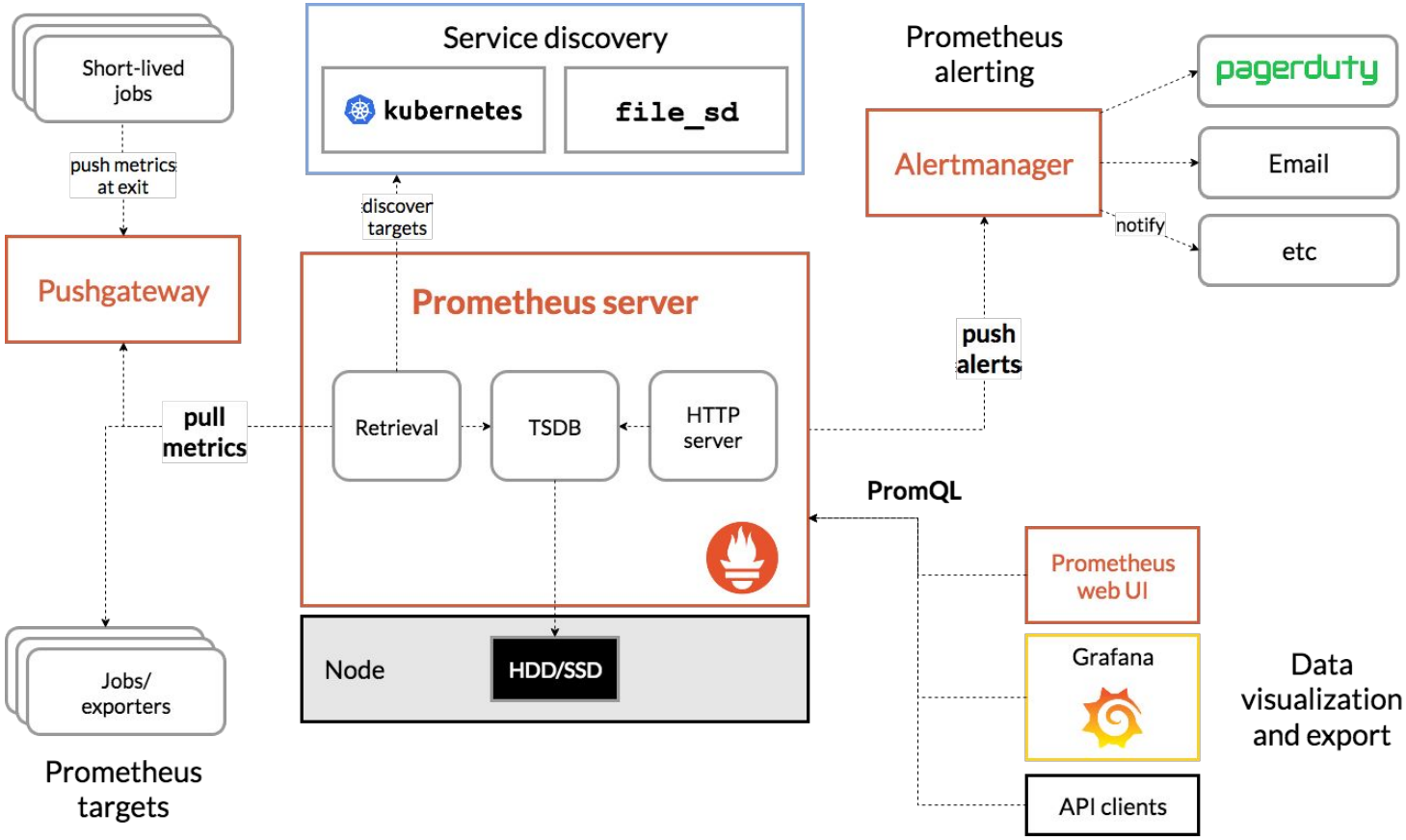
Monitoring: Prometheus

https://prometheus.io/docs/introduction/overview/

Main Features:

- main objective: collect pure numeric time-series data
- a **multi-dimensional data model** with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- no reliance on distributed storage; single server nodes are autonomous
    a. this means it is NOT a distributed and fault-tolerant data storage system!
- time series collection happens via a pull model over HTTP
- pushing time series is supported via an intermediary gateway
- targets are discovered via service discovery or static configuration
- multiple modes of graphing and dashboarding support

Prometheus: architecture



Prometheus: storage

Prometheus includes a local on-disk time series database, but also optionally integrates with remote storage systems

Local storage:

- not clustered or replicated
- data stored on local disk in a TSDB format: https://github.com/prometheus/tsdb/tree/master/docs/format

TSDB format

- Index
- Chunks
- Tombstones
- Wal

The directory structure of a Prometheus server's data directory will look something like this:

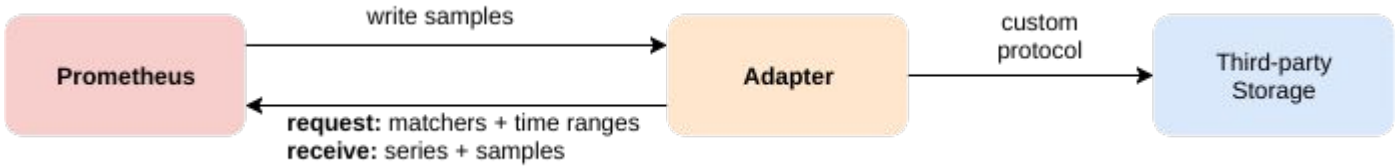
```
./data/01BKGV7JBM69T2G1BGBGM6KB12
./data/01BKGV7JBM69T2G1BGBGM6KB12/meta.json
./data/01BKGV7JBM69T2G1BGBGM6KB12/wal
./data/01BKGV7JBM69T2G1BGBGM6KB12/wal/000002
./data/01BKGV7JBM69T2G1BGBGM6KB12/wal/000001
./data/01BKGTZQ1SYQJTR4PB43C8PD98
./data/01BKGTZQ1SYQJTR4PB43C8PD98/meta.json
./data/01BKGTZQ1SYQJTR4PB43C8PD98/index
./data/01BKGTZQ1SYQJTR4PB43C8PD98/chunks
./data/01BKGTZQ1SYQJTR4PB43C8PD98/chunks/000001
./data/01BKGTZQ1SYQJTR4PB43C8PD98/tombstones
```

Prometheus: storage

Remote Storage:

Prometheus can use external third-party storage via integration:

- can write samples that it ingests to a remote URL in a standardized format
- can read (back) sample data from a remote URL in a standardized format
- The read and write protocols both use a snappy-compressed protocol buffer encoding over HTTP



Prometheus: storage

Remote Storage - Integration options:

- AppOptics: write
- Chronix: write
- Cortex: read and write
- CrateDB: read and write
- Elasticsearch: write
- Gnocchi: write
- Graphite: write
- InfluxDB: read and write
- IRONdb: read and write
- Kafka: write
- M3DB: read and write
- OpenTSDB: write
- PostgreSQL/TimescaleDB: read and write
- SignalFx: write
- VictoriaMetrics: write

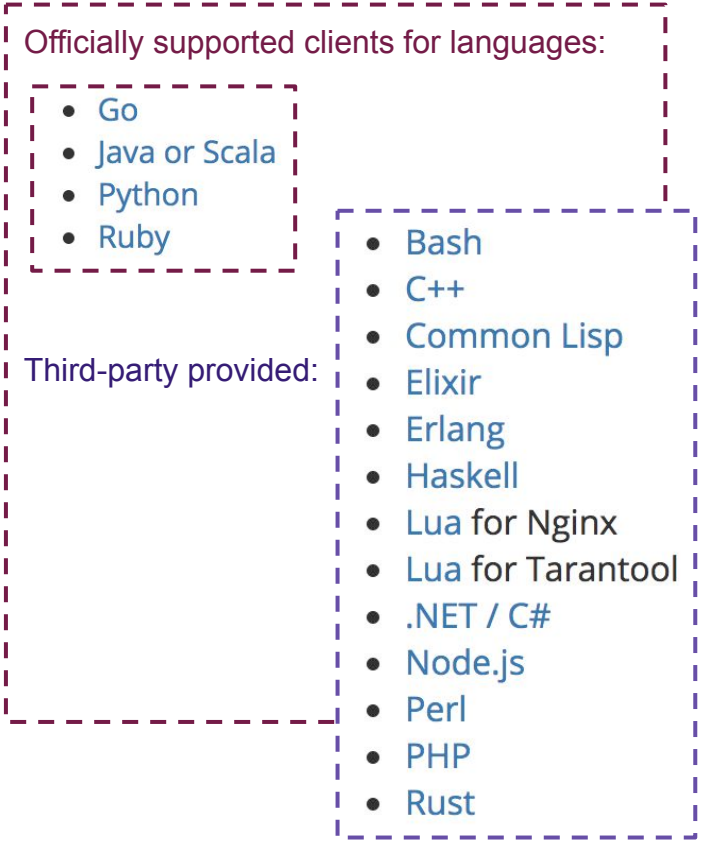
Prometheus: client libraries

Client libraries:

- to monitor your own services/applications, you need to add instrumentation to their code via one of the Prometheus client libraries that implement the Prometheus metric types.
- in your app: use a Prometheus client library that matches the language in which your application is written
- define and expose internal metrics via an HTTP endpoint

for JVM-based applications:

- can expose metrics via standard JMX interface
- use JMX Exporter - Prometheus export agent



Prometheus: exporters

Exporters:

- libraries and servers which help in exporting existing metrics from third-party systems as Prometheus metrics
- This is useful for cases where it is not feasible to instrument a given system with Prometheus metrics directly (for example, HAProxy or Linux system stats)
- Official exporters - maintained in the Prometheus GitHub projects:
- Third-party exporters - many other community-contributed ones

Examples:

- **Node Exporter** exposes a wide variety of hardware- and kernel-related metrics
- JMX exporter can export from a wide variety of JVM-based applications, for example Kafka and Cassandra.

Prometheus: supported exporters

Huge list!!! https://prometheus.io/docs/instrumenting/exporters/

Messaging systems

- Beanstalkd exporter
- Gearman exporter
- Kafka exporter
- NATS exporter
- NSQ exporter
- Mirth Connect exporter
- MQTT blackbox exporter
- RabbitMQ exporter
- RabbitMQ Management Plugin exporter

and many more...

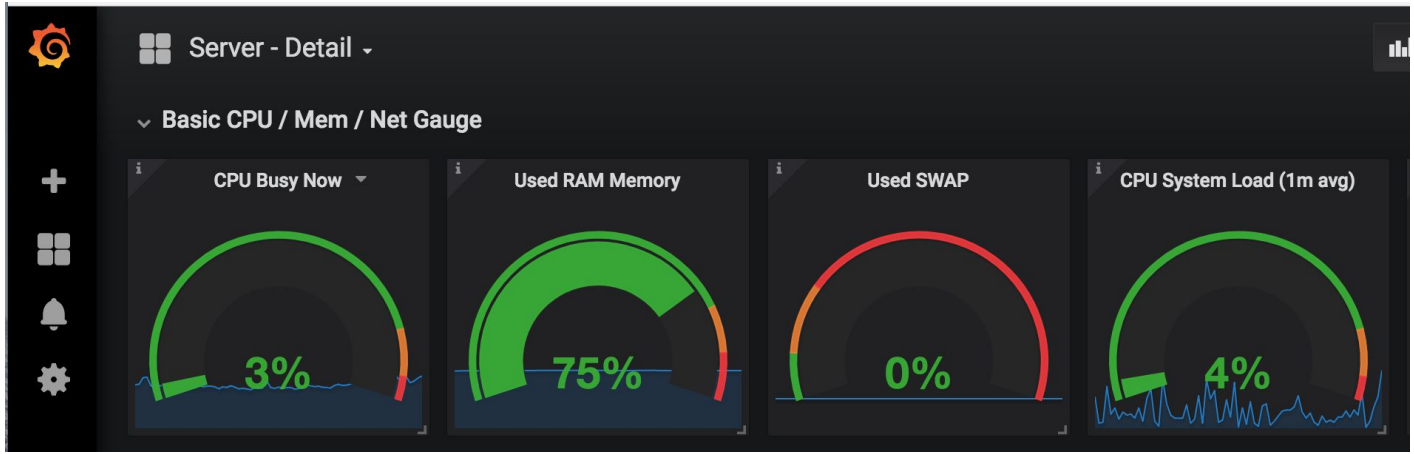
Other monitoring systems

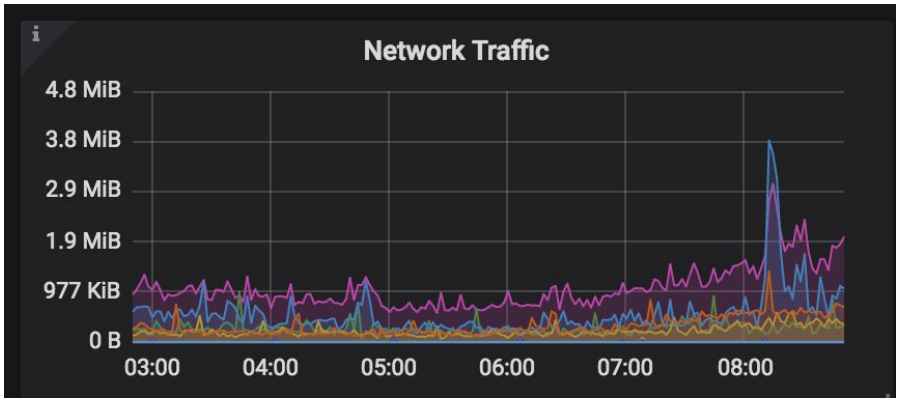
- Akamai Cloudmonitor exporter
- AWS CloudWatch exporter (official)
- Cloud Foundry Firehose exporter
- Collectd exporter (official)
- Google Stackdriver exporter
- Graphite exporter (official)
- Heka dashboard exporter
- Heka exporter
- InfluxDB exporter (official)
- JavaMelody exporter
- JMX exporter (official)
- Munin exporter
- Nagios / Naemon exporter
- New Relic exporter
- NRPE exporter
- Osquery exporter
- OTC CloudEye exporter
- Pingdom exporter
- scollector exporter
- Sensu exporter
- SNMP exporter (official)
- StatsD exporter (official)

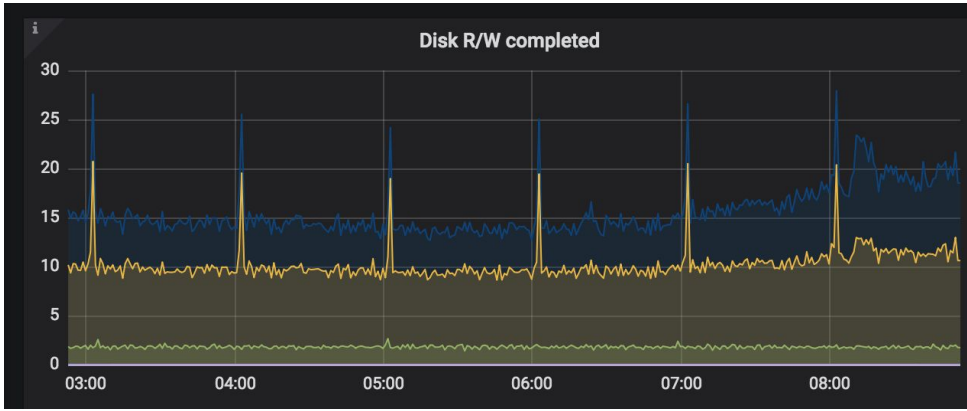
Databases

- Aerospike exporter
- ClickHouse exporter
- Consul exporter (official)
- Couchbase exporter
- CouchDB exporter
- ElasticSearch exporter
- EventStore exporter
- Memcached exporter (official)
- MongoDB exporter
- MSSQL server exporter
- MySQL server exporter (official)
- OpenTSDB Exporter
- Oracle DB Exporter
- PgBouncer exporter
- PostgreSQL exporter
- ProxySQL exporter
- RavenDB exporter
- Redis exporter
- RethinkDB exporter
- SQL exporter
- Tarantool metric library

Prometheus + Graphana

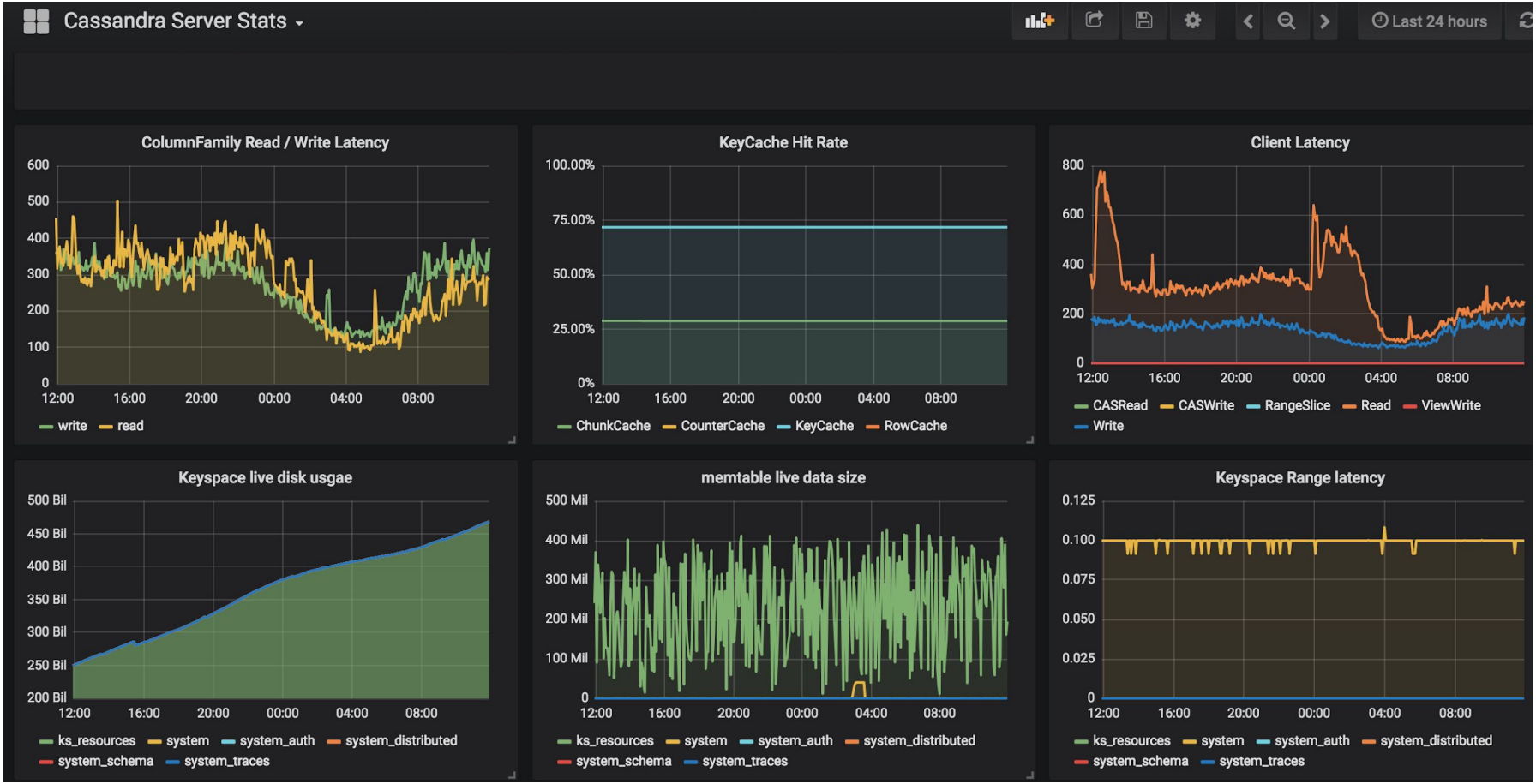






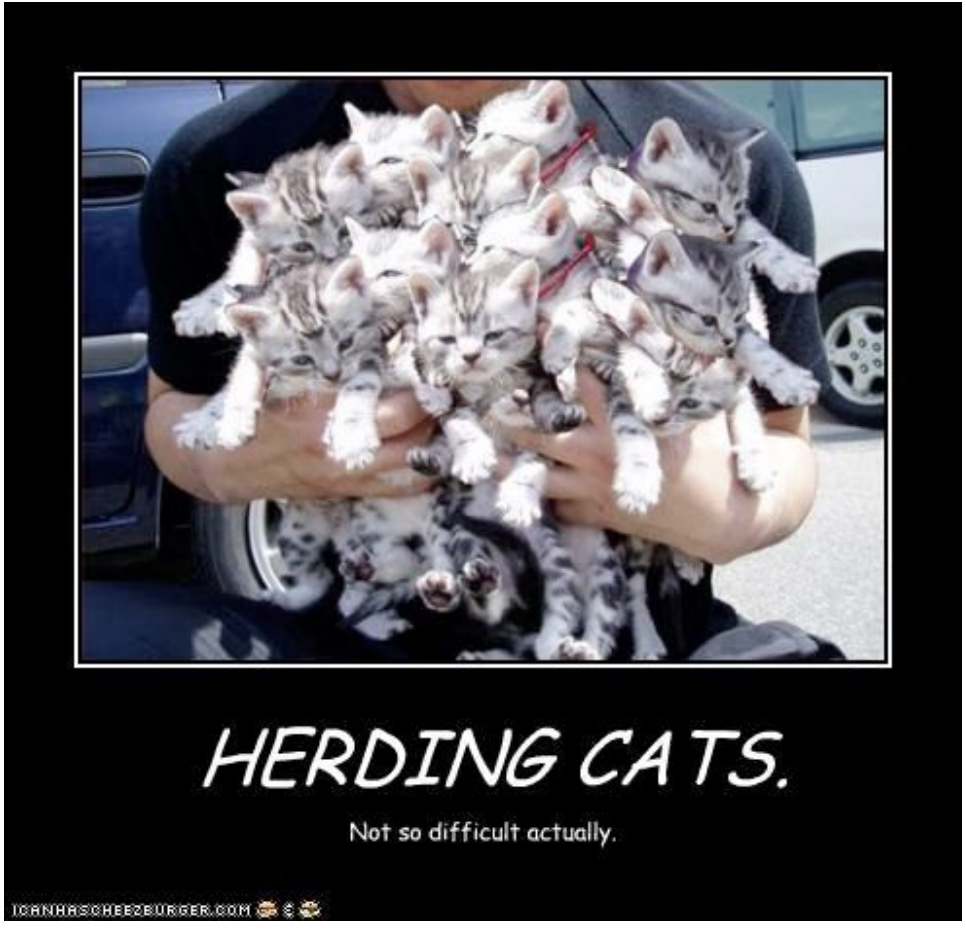
@Marina Popova

Prometheus + Graphana: Cassandra monitoring



@Marina Popova

Moving On: Controlling the Chaos ...



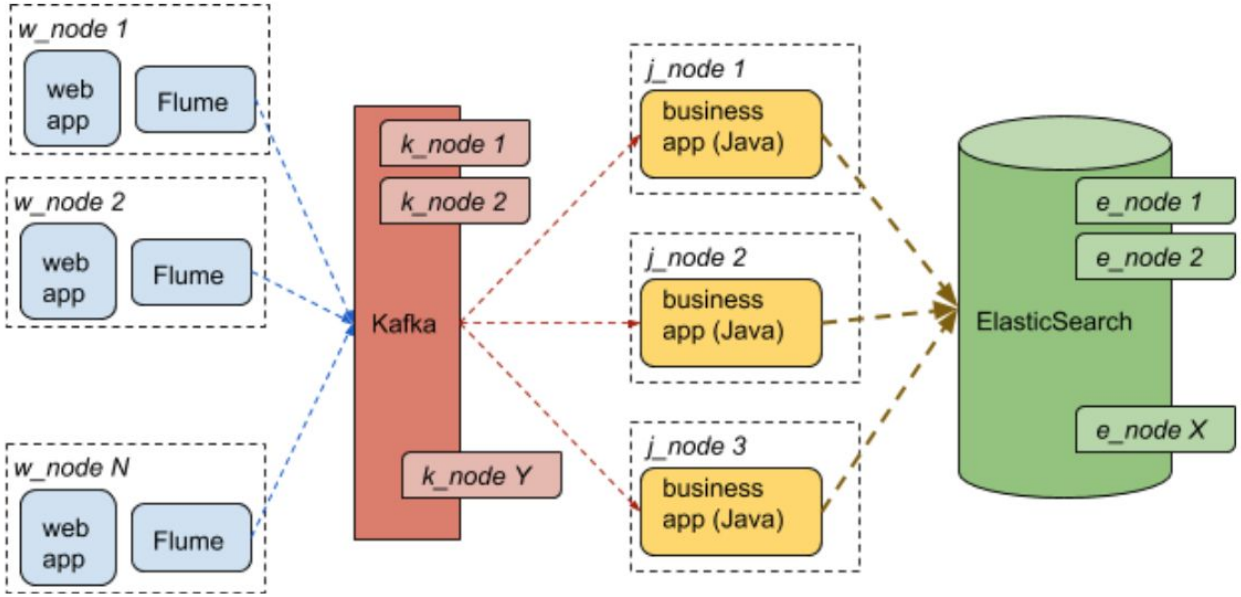
@Marina Popova

Orchestrating Big Data Processing Pipelines

Why orchestration ??

Same simple pipeline:

layer	# of nodes	# of apps
collection	10	20
kafka	5	5
Java apps	3	3
ES	15	15
Total	**33**	**43**



How do you control starting/stopping all services, order of execution, failed nodes detection/replacement, ... ??

Workflow Scheduling: Options

- tried and true Unix CRON jobs
- DevOps approach: Puppet/Ansible/Chef/Saltstack/Terraform/...
- Hadoop ecosystem: YARN, Oozi
- specialized workflow management systems:
  - Airflow (AirBnb)
  - Luigi (Spotify)

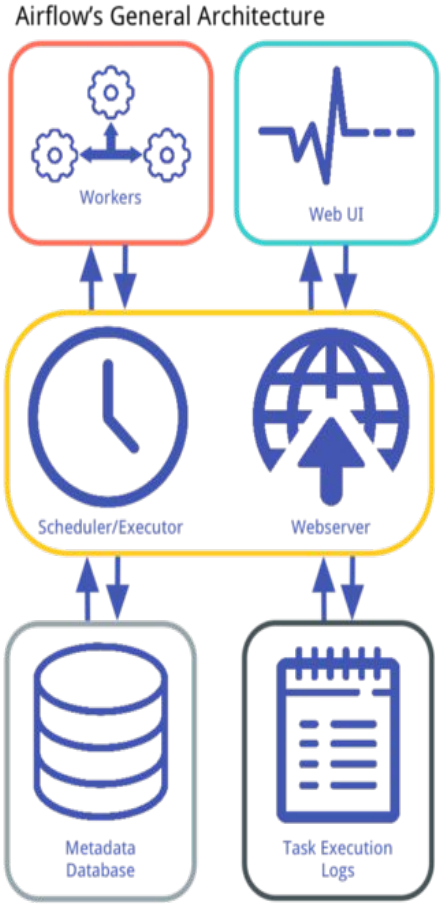
Airflow

Airflow is a platform to programmatically author, schedule and monitor workflows

https://airflow.apache.org/start.html

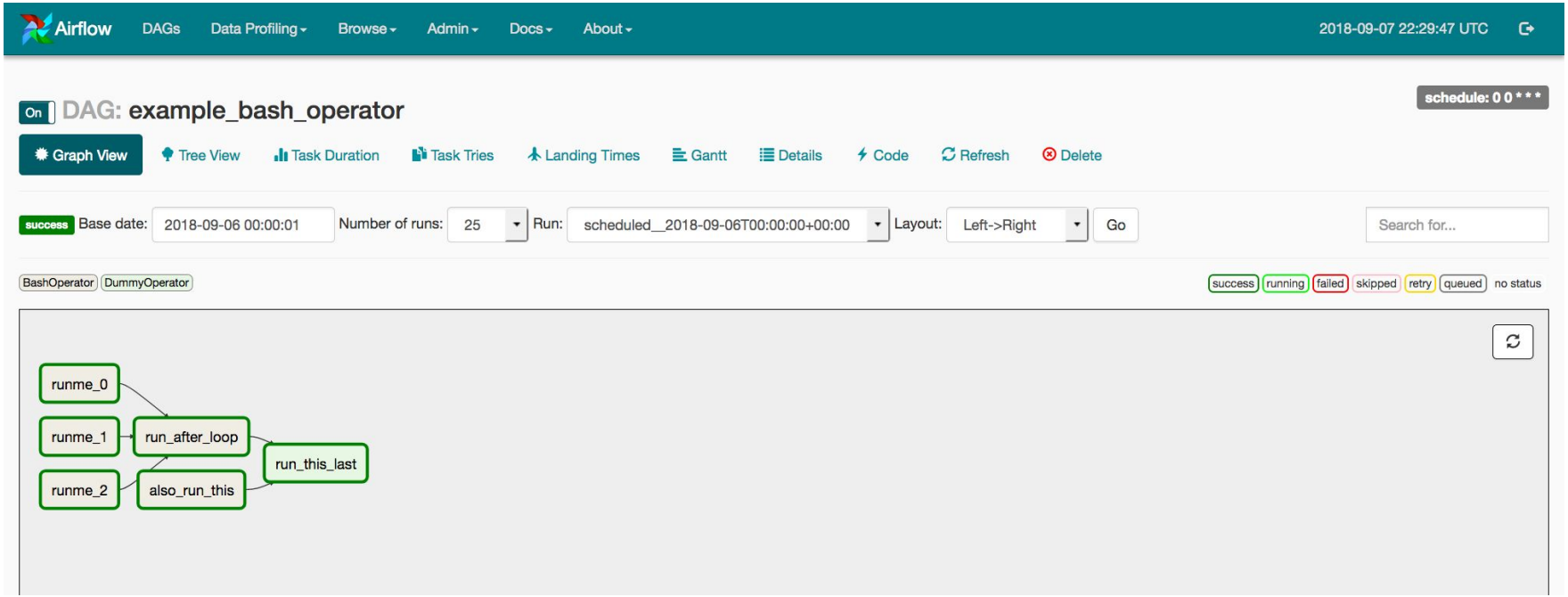
Main components and concepts:

- DAG - workflow definition as a set of Tasks, with directed dependences
- Task - Unit of work to be executed, can be of types:
  - Sensors: check status of tasks/data structures
  - Operator: execute some operation (like bash script, Python code ...)
- WebUI - a portal to monitor status of the DAGs
- Metadata DB - the metastore of Airflow for storing job status, task instance status, etc.
- Scheduler - a multi-process which parses the DAG bag, creates a DAG object and triggers executor to execute correct tasks
- Executor - a **message queuing process** that orchestrates worker processes to execute tasks - usually Celery



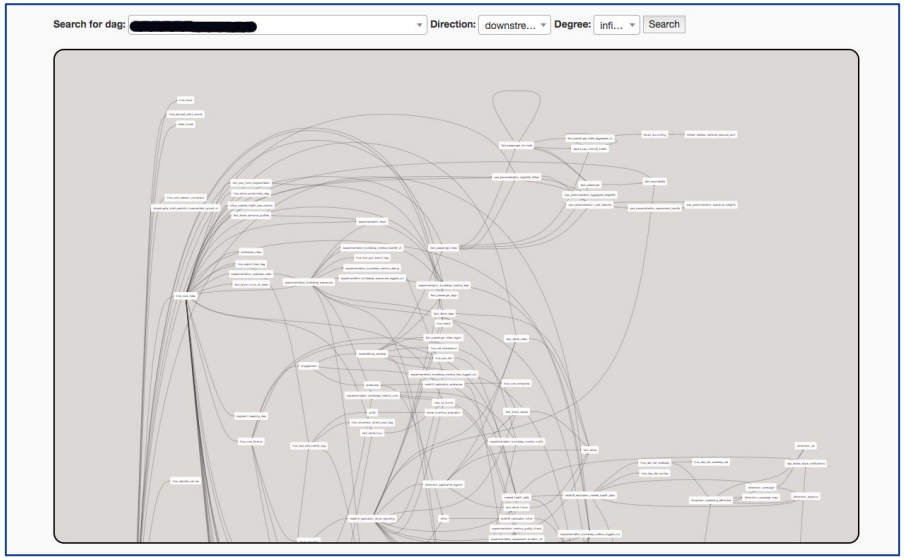
Airflow

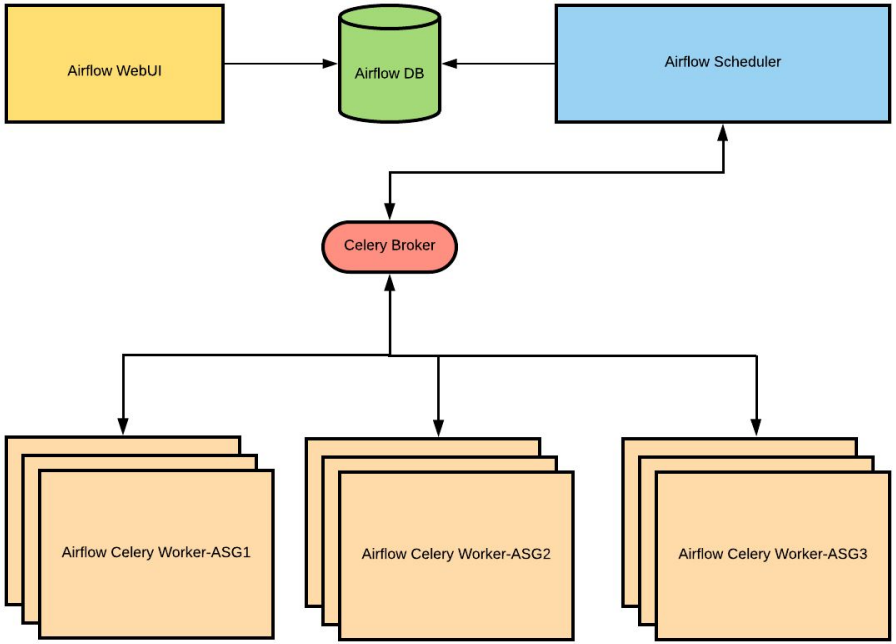
The Airflow UI makes it easy to monitor and troubleshoot your data pipelines



Airflow: Use Cases

Lyft: https://eng.lyft.com/running-apache-airflow-at-lyft-6e53bb8fccff





@Marina Popova