BIG DATA [MODULE 2]

Cloud computing

whoami

Matteo Francia

- Research fellow @ UniBO

Research topics

- Big data / database
- Geo-spatial analytics

Thesis proposals

- https://big.csr.unibo.it/teaching/

https://big.csr.unibo.it/

Exam

We will see how (big) data pipelines can be done in the cloud

- We will cover some "theory" as well as get our hands dirty
- This module is part of the oral examination
- (If agreed) The project can be extended to the cloud to get more points
  - E.g., rethink a part of the project on cloud

Roadmap

Introduction to cloud and cloud service providers

Cloud services (in AWS)

Hands-on cloud services

- Orchestrating (small) data pipelines
- Migrating a cluster to the cloud
- Migration pricing

Set up

You (will) have an AWS Educate account

- A coupon of 150$ to test AWS cloud services

The content of these slides refers to this repo

- https://github.com/w4bo/bigdata-aws/
- Options
  - Work on your pc: check the `README.md` to install the necessary software tools
    - Mainly: git, IntelliJ IDEA (Community Edition), docker, AWS CLI, AWS SAM CLI
  - Download a pre-configured (VMware) VM with the installed software

https://aws.amazon.com/education/awseducate/

Set up

Download a pre-configured (VMware) VM with the installed software

1. Connect to a PC-lab using Guacamole (click here https://csi-rlab.campusfc.unibo.it/)
   a) Enter your credentials
   b) Choose `LabCEZ`
   c) Click on the proper lab
2. Once logged in, download the VM from http://big.csr.unibo.it/downloads/ubuntu-64-bit.zip
3. Unzip the content and launch `VMware Workstation Pro` (from the desktop)
4. `File` > `Open...` > `path/to/the/unzipped/vm/*.vmx` file
5. Launch the VM (user: `bigdata`, pwd: `bigdata`)

So far

You have practiced with **on-premises** solutions

- You were given a working hardware cluster
- ... to deploy software applications on Hadoop-based stack

Let us guess, how would you start from scratch?

- How would you do that?
- How much time would it take?

So far

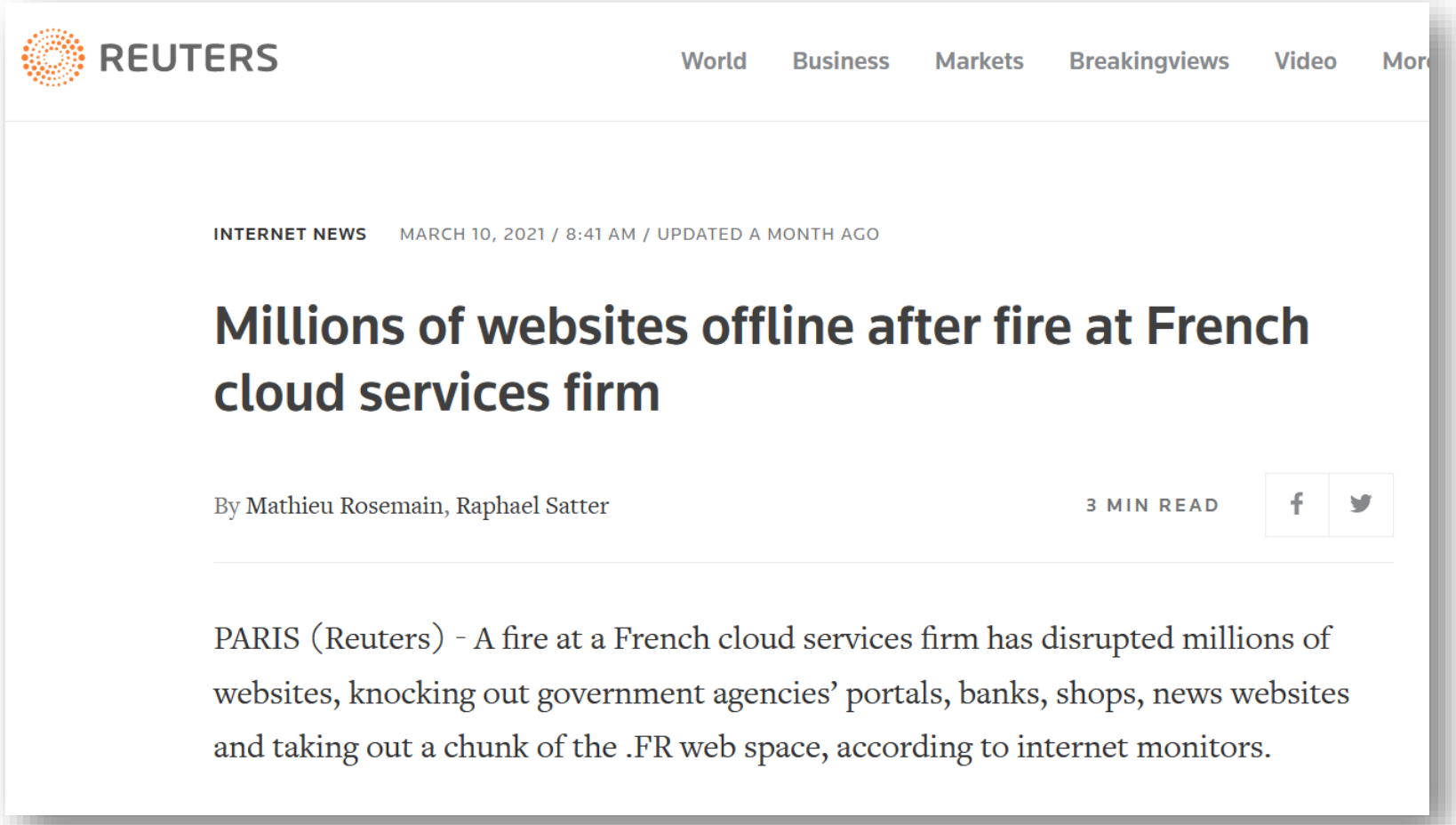
No easy answers

Big-data (distributed) architectures require a lot of skills

- **Installation**: how do I set up a new machine?
- **Networking**: how do I cable dozens of machines?
- **Management**: how do I replace a broken disk?
- **Upgrade**: how do I extend the cluster with new services/machines?
- (energy and cooling, software licenses, insurance...)

https://aws.amazon.com/compliance/data-center/data-centers/

So far



REUTERS

World Business Markets Breakingviews Video Mor

INTERNET NEWS MARCH 10, 2021 / 8:41 AM / UPDATED A MONTH AGO

Millions of websites offline after fire at French cloud services firm

By Mathieu Rosemain, Raphael Satter 3 MIN READ

PARIS (Reuters) - A fire at a French cloud services firm has disrupted millions of websites, knocking out government agencies' portals, banks, shops, news websites and taking out a chunk of the .FR web space, according to internet monitors.

https://www.reuters.com/article/us-france-ovh-fire-idUSKBN2B20NU

So far

Technological perspective

- How do we configure a distributed environment?
  - How do we set up independent services?
  - How do we integrate such services?
  - How do control resource accesses (e.g., storage)?
- How do we orchestrate data flows?
- It depends on your (team) skills (not only software engineering)

Business perspective

- No free lunch, each choice has cost/benefit
- How much time does it take to master a technology?
- How many people do I need?

So far

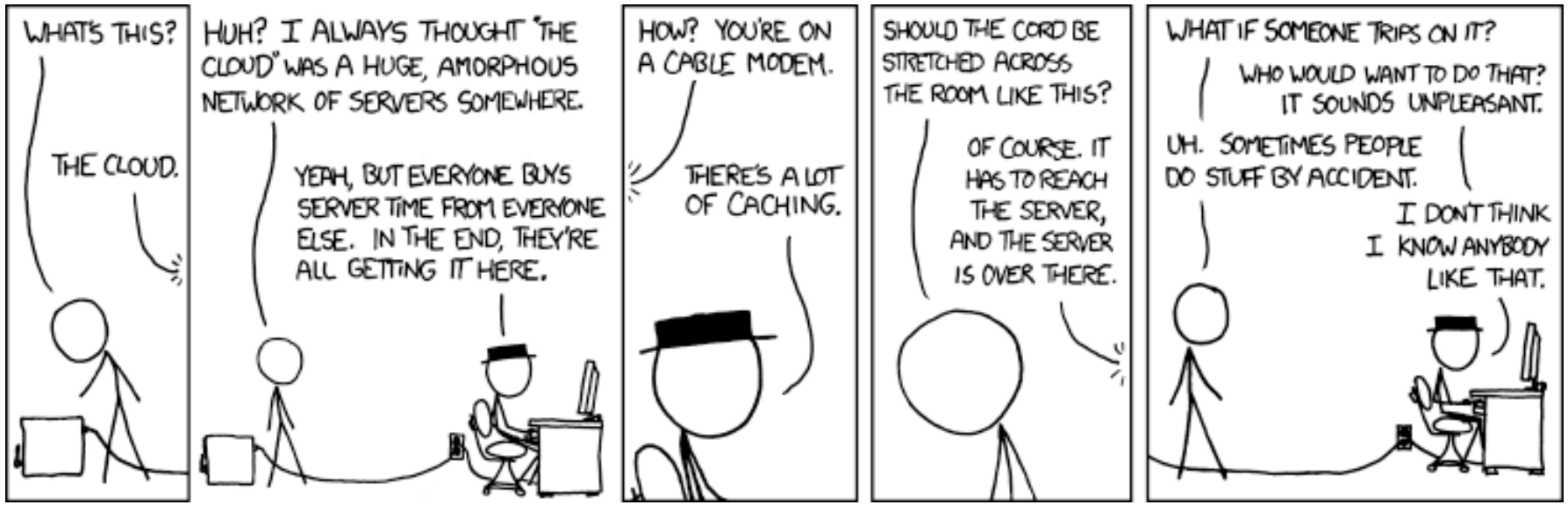
Can we afford to spend resources on tasks that are not mission oriented?

- Mission: a statement used by a company to explain its purpose(s)

How can I build a working application/data platform?

BIG DATA

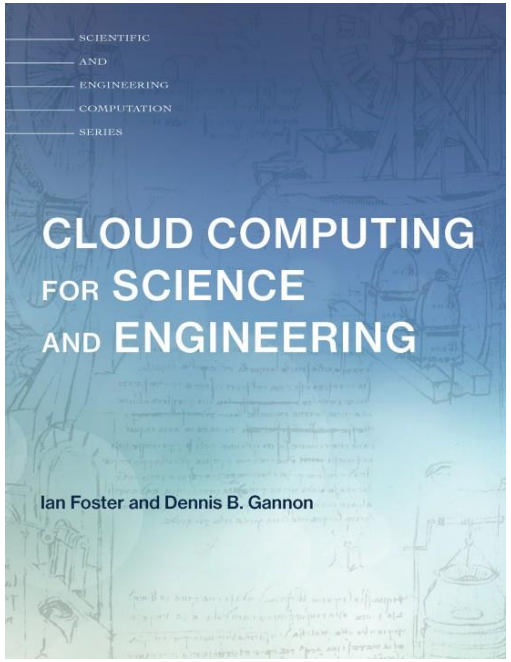
Why going cloud?

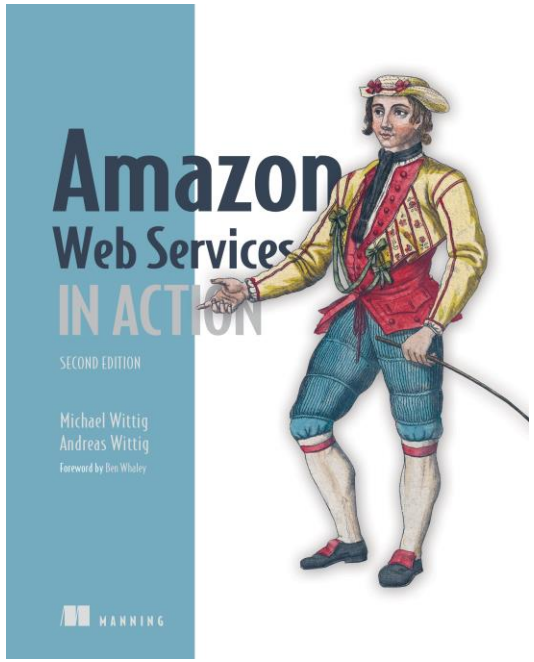


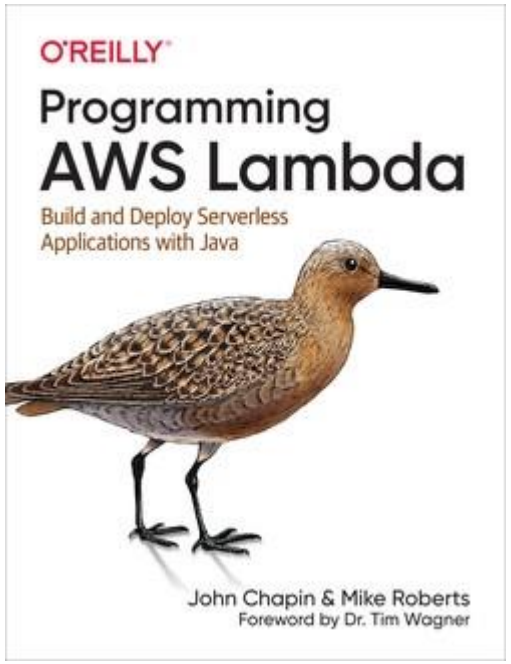
https://xkcd.com/908/

Teaching material

Books

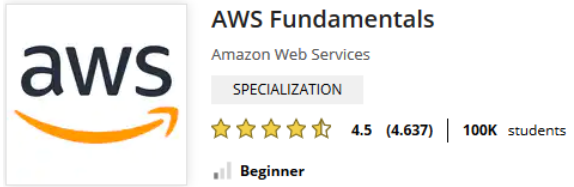


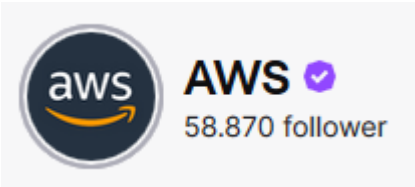




Generic ⟶ Specific

Web content





Why going cloud?

Cloud computing (National Institute of Standards and Technology)

"A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

- On-demand self-service (consume services when you want)
- Broad network access (consume services from anywhere)
- Resource pooling (infrastructure, virtual platforms, and applications)
- Rapid elasticity (enable horizontal scalability)
- Measured service (pay for the service you consume as you consume)

Why going cloud?

Goal: adjusts capacity to have predictable performance at the lowest cost

Scalability that is not possible on premises

- Scale from one to thousands of servers

Elasticity

- Automatically scale resources in response to run-time conditions
- Core justification for the adoption of cloud

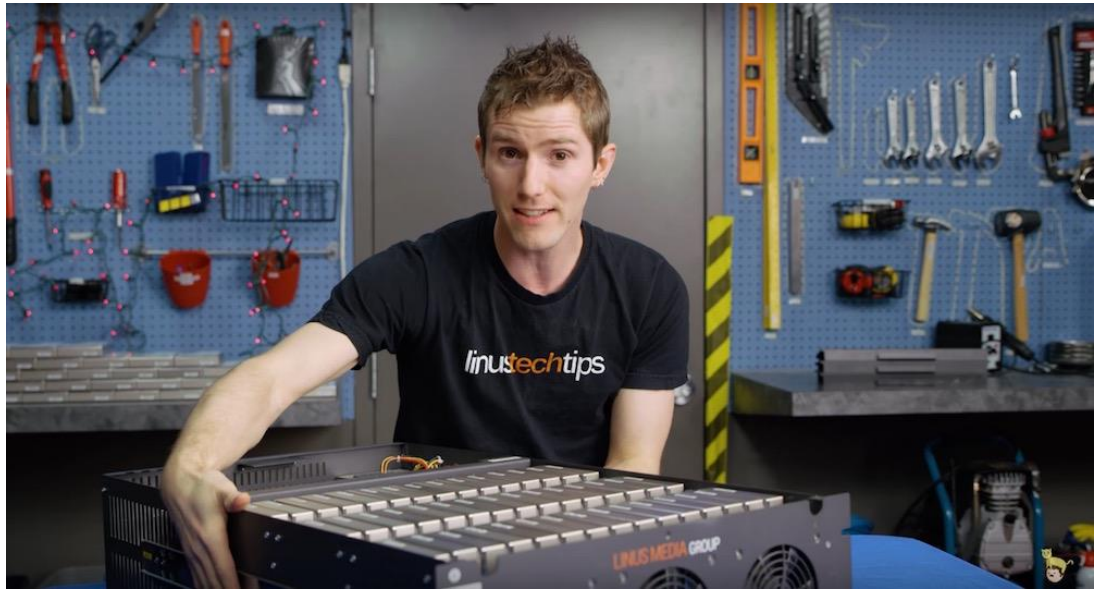
Why going cloud

Hardware scalability

- No longer think about rack space, switches, and power supplies, etc.

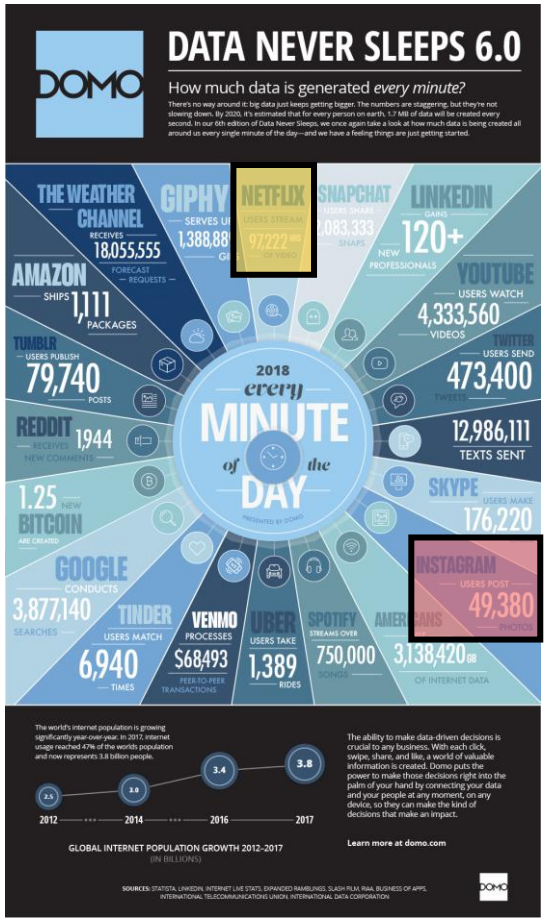
Grow storage from GBs to PBs

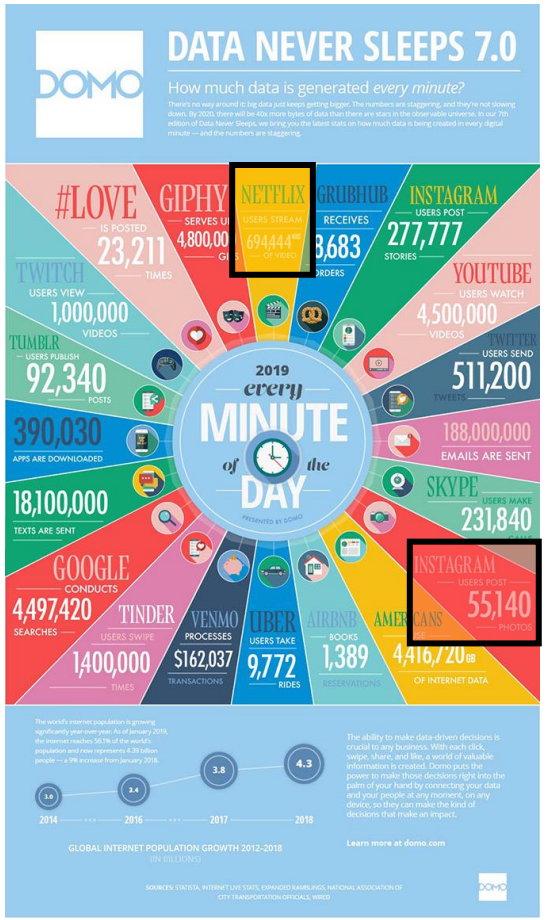
- One hundred of our 10TB Enterprise Capacity 3.5 HDD hard drives

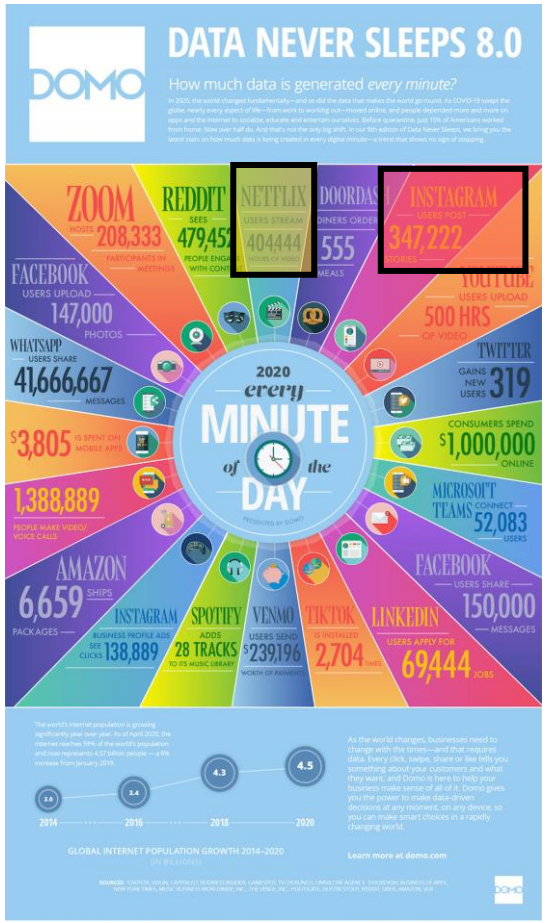


https://blog.seagate.com/business/linus-tech-tips-want-petabyte-system/

Why going cloud?







https://www.domo.com/learn/data-never-sleeps-8

Why going cloud?

Resource pooling

- Enable a resource to serve different consumers
- Dynamically reassigned according to demands
- Economy of scale

Reliability

- Built to handle failures
- Fault-tolerant or highly available

Worldwide **deployment**

- Deploy applications as close to customers as possible
  - E.g., to reduce network latency
- Improve data locality
- Compliant to privacy regulations (e.g., GDPR)

Why going cloud?

User perspective

- Eliminate repetitive tasks to focus on strategic ones
- Abstract the underlying architecture
- Adapt infrastructure to requirements, create (test) environments on demand

Service **integration**

- Do not reinvent the wheel
- Use services that solve common problems (e.g., load balancing, queuing)

Service integration and abstraction are drivers of change

- From databases to data platforms
- From on-premises hardware to serverless architectures
- From custom to standard data pipelines

BIG DATA

From databases to data platforms

Data platform

Companies are collecting tons of data to enable advanced analytics

- Data are more and more heterogeneous and complex
- Raw data are difficult to obtain, interpret, describe, and maintain
- There is a need for describing/curating the data to make them consumable

Databases/warehouses are no longer ideal data hubs for integration/analysis

- Getting value from data is not only a matter of storage
- Need integrated and multilevel analytical skills and techniques

Data platform

Database

"A database is a structured and persistent collection of information about some aspect of the real world organized and stored in a way that facilitates efficient retrieval and modification. The structure of a database is determined by an abstract data model. Primarily, it is this structure that differentiates a database from a data file. The most popular data model is relational that represents data as a set of tables."

Data Warehouse

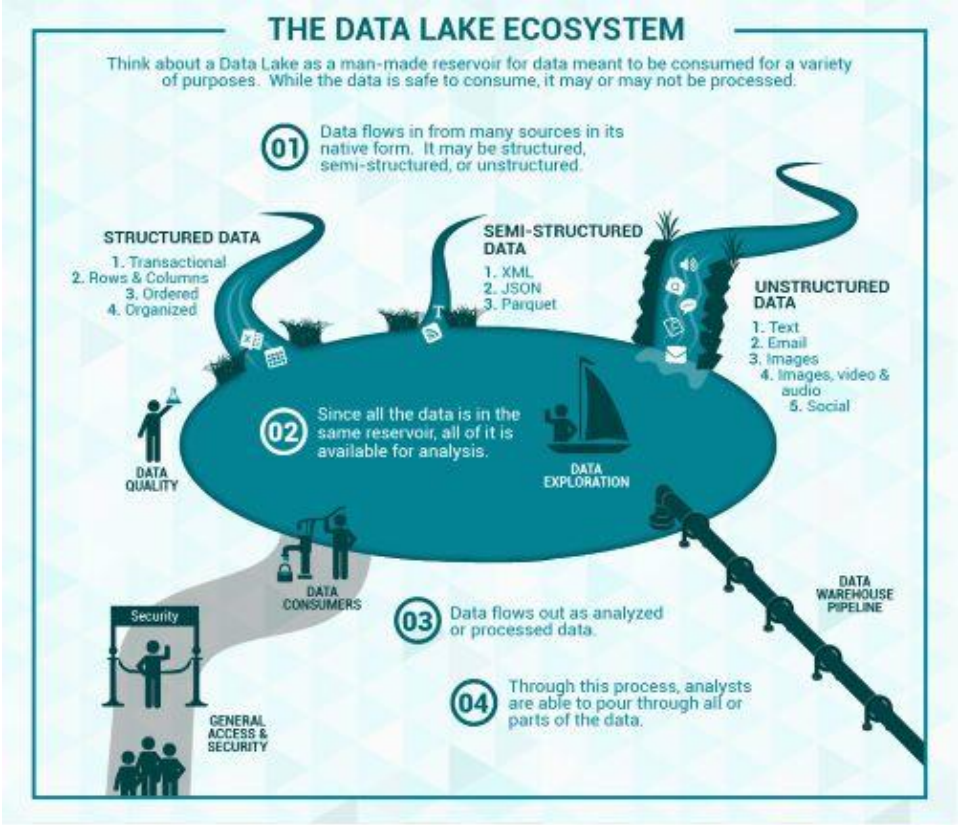
"A collection of data that supports decision-making processes. It provides the following features: subject-oriented, integrated and consistent, not volatile."

Özsu M.T. (2018) Database. In: Encyclopedia of Database Systems. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-8265-9_80734
Matteo Golfarelli and Stefano Rizzi. *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, Inc., 2009.

Data platform

Data lake

Couto et al.: *“A DL is a central repository system for storage, processing, and analysis of raw data, in which the data is kept in its original format and is processed to be queried only when needed. It can store a varied amount of formats in big data ecosystems, from unstructured, semi-structured, to structured data sources”*



Couto, Julia, et al. "A Mapping Study about Data Lakes: An Improved Definition and Possible Architectures." *SEKE*. 2019.
https://dunnsolutions.com/business-analytics/big-data-analytics/data-lake-consulting

Data platform

Data lakes have increasingly taken the role of data hubs

- Eliminate up-front costs of ingestion since data are stored in original format
- Once in DL, data are available for analysis by everyone in the organization

Drawing a sharp line been storage/computation/analysis is hard

- Is a database just storage?
- What about SQL/?

Blurring of the architectural borderlines

- DL is often replaced by “data platform” or “data ecosystem”
- Encompass systems supporting data-intensive storage, computation, analysis

Data platform

Blurring of the architectural borderlines

- DL is often replaced by “data platform” or “data ecosystem”
- Encompass systems supporting data-intensive storage, computation, analysis

Data platform (e.g., on Google Cloud and Amazon AWS)

- Rationale: relieve users from complexity of administration and provision
  - Not only technological skills, but also privacy, access control, etc.
  - Only focus on functional aspects

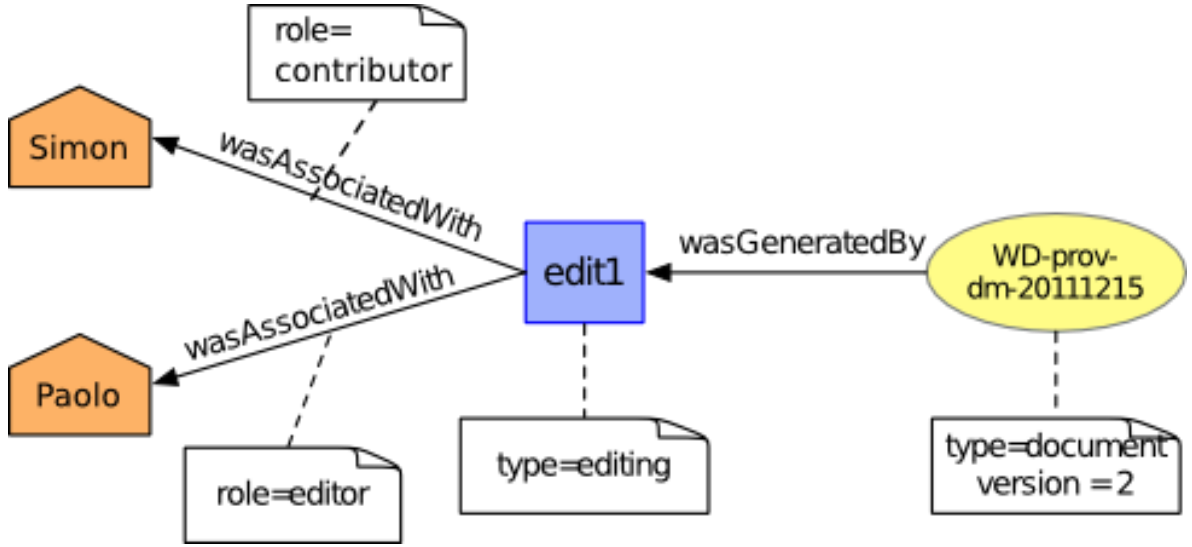
Are we done? No!

- Lacking smart support to govern the complexity of data and transformations
- Data transformations must be governed to prevent DP turns into a swamp
  - Amplified in data science, with data scientists prevailing data architects
  - Leverage descriptive metadata and maintenance to keep control over data

Data platform

Data provenance

- Metadata pertaining to the history of a data item
- Pipeline including the origin of objects and operations they are subjected to
- We have a standard: https://www.w3.org/TR/prov-dm/



https://www.w3.org/TR/prov-dm/

Data provenance

Entity

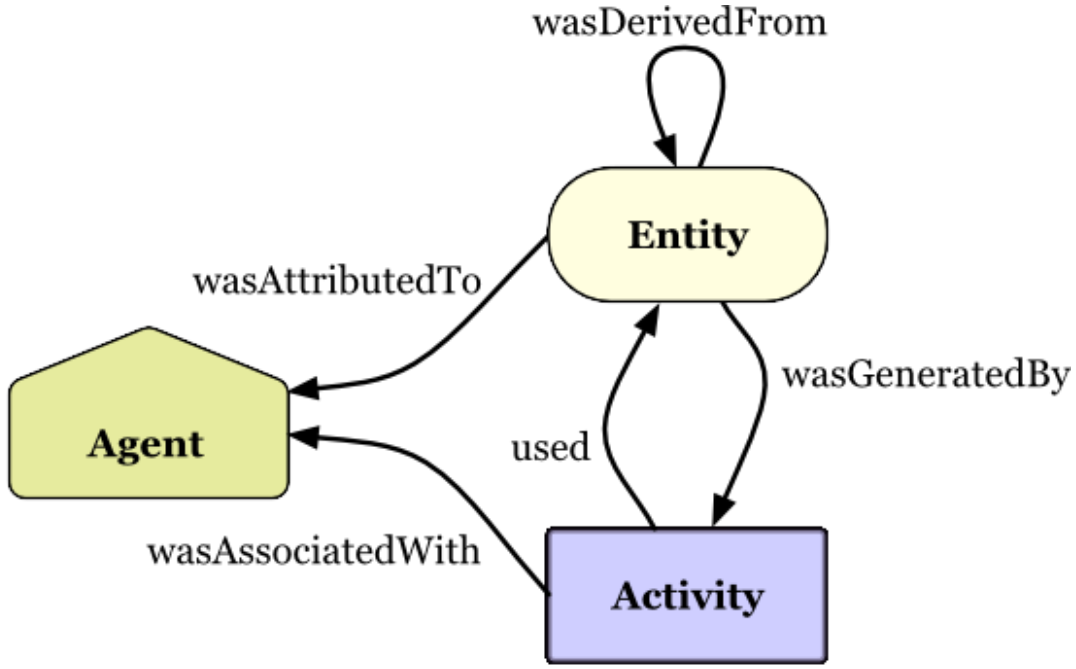
- Physical/conceptual things

Activity

- Dynamic aspects of the world, such as actions
- How entities come into existence, often making use of previously existing entities

Agent

- A person, a piece of software
- Takes a role in an activity such that the agent can be assigned some degree of responsibility for the activity taking place



https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/

Data provenance

Data quality

- Monitoring of the quality (e.g., accuracy) of the objects produced
- Notify when a transformation pipeline is not behaving as expected

Debugging

- Inferring the cause of pipeline failures is challenging
- Store inputs of each operation with versions and environmental settings (RAM, CPUs, etc.)

And so on...

Data platform

Are we done? No!

- Metadata can become bigger than data themselves

We need meta meta-data (or models)...

- ... chasing our own tails

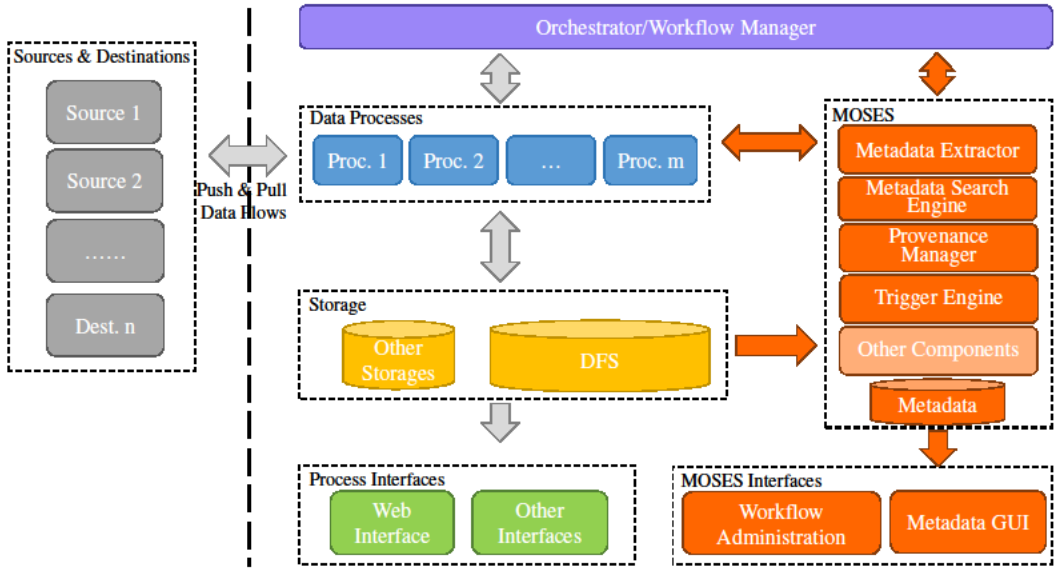
Data management is still a (research) issue in data platforms

Data platform

MOSES: making data platform smarter

Functional architecture

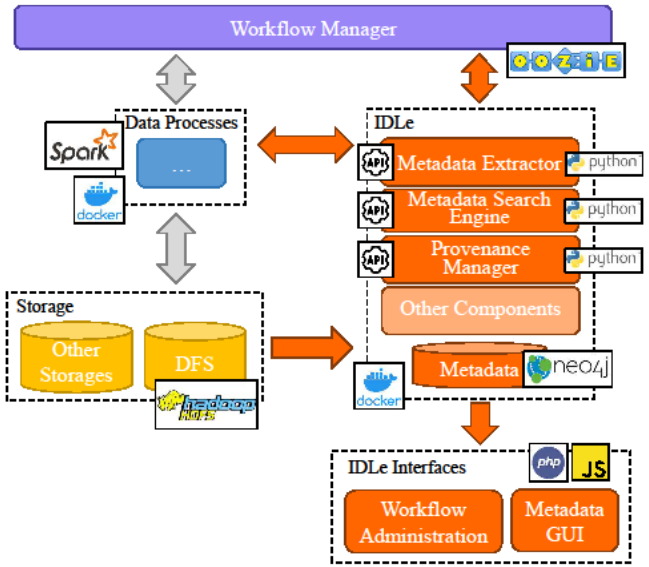
- Components of MOSES are in orange
- The others are standard components in charge of producing/consuming, processing, storing, and visualizing data
- The orchestrator (e.g., Oozie in the Hadoop ecosystem) manages (e.g., schedules) the data transformation processes



Data platform

MOSES is used within a DP that builds atop the Hadoop ecosystem

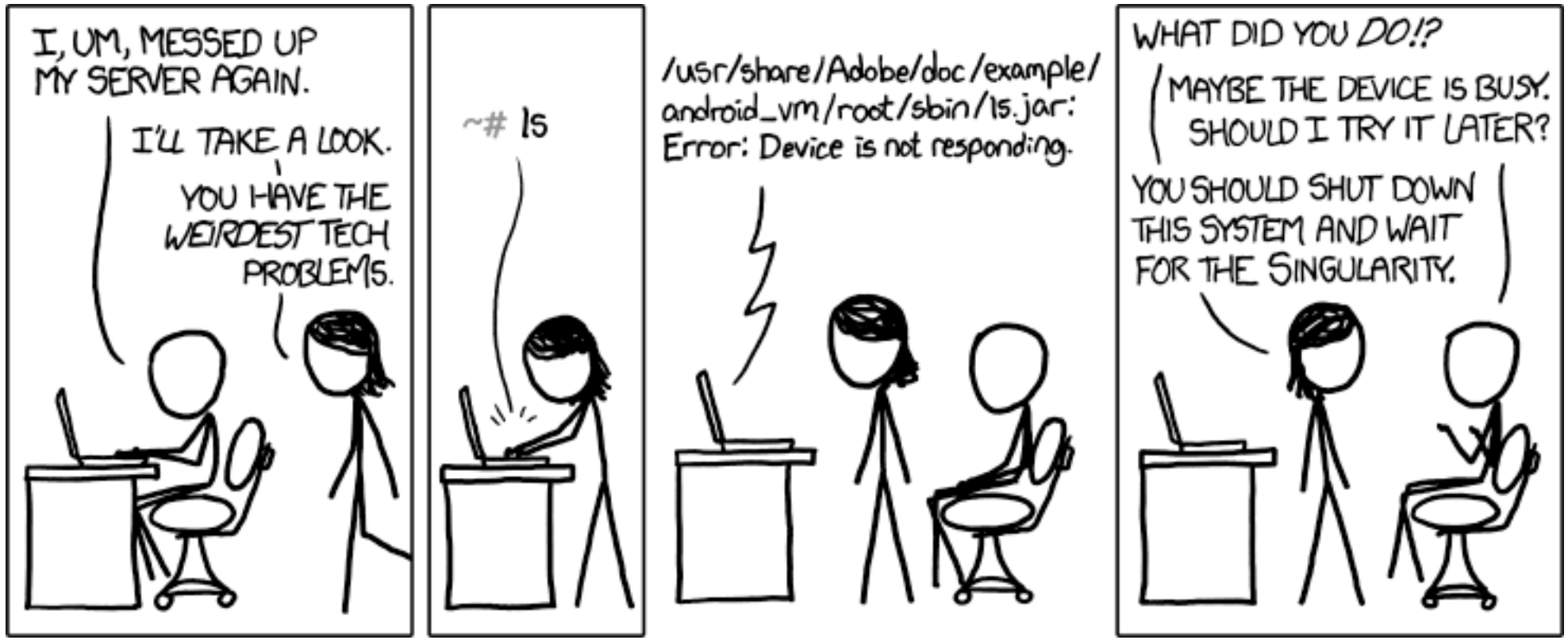
- The cluster runs the Cloudera Distribution for Apache Hadoop 6.2.0 and Docker
- The metadata collected and manipulated are stored in a graph database
  - The graph data model favors the modeling of highly interconnected data in the absence of fixed schemas
- Service decoupling: functional components are accessible using RESTful APIs



Leoni Anna Giulia. "Gestione di un data lake strutturato attraverso il riconoscimento semantico dei dati acquisiti."

BIG DATA

From PaaS to FaaS (serverless)



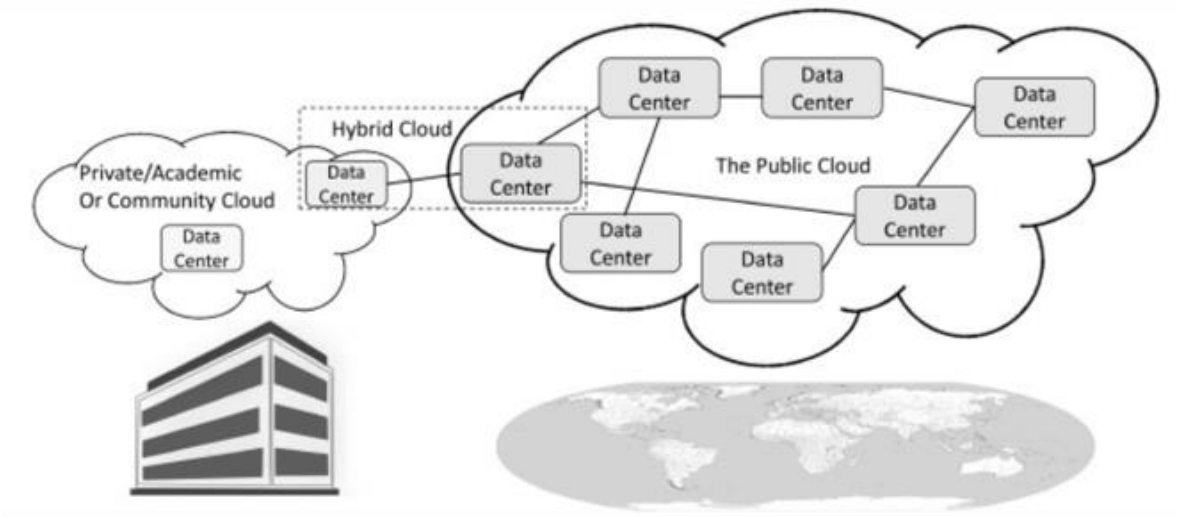
https://xkcd.com/1084/

From PaaS to FaaS

Public: accessible to anyone willing to pay (e.g., Microsoft, AWS, Google)

Private: accessible by individuals within an institution

Hybrid: a mix of the previous



From PaaS to FaaS

Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services

https://www.gartner.com/en/research/methodologies/magic-quadrants-research

Gartner Magic Quadrant

- Understanding the technology providers to consider for an investment
- **Leaders** execute well and are well positioned for tomorrow
- **Visionaries** understand where the market is going but do not yet execute well
- **Niche Players** focus successfully on a small segment, or are unfocused and do not out-innovate or outperform others
- **Challengers** execute well but do not demonstrate an understanding of market direction
- Focusing on leaders isn't always the best
  - A niche player may support needs better than a market leader. It depends on how the provider aligns with business goals

From PaaS to FaaS

Cloud services are hosted in multiple locations worldwide

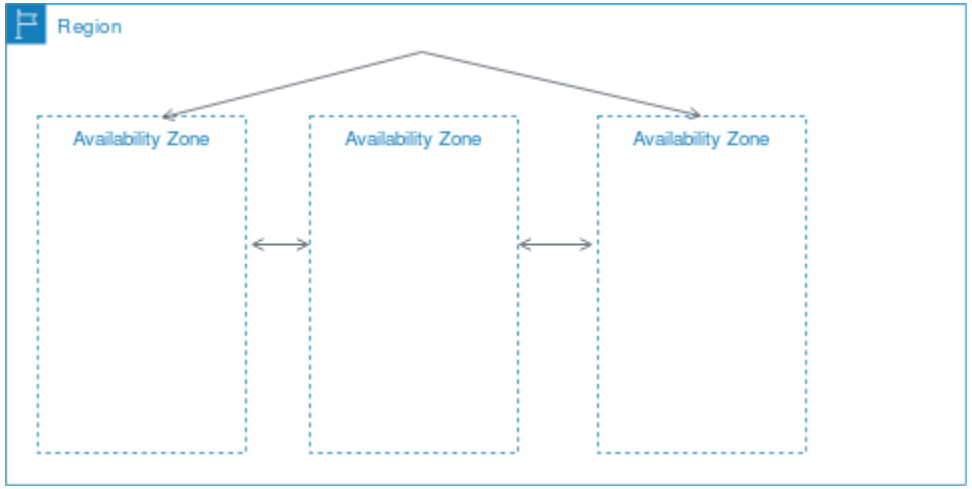
- Locations are composed of **regions** and **availability zones**

Region (e.g., us-east-1)

- Is an independent geographical area
- Has availability zones

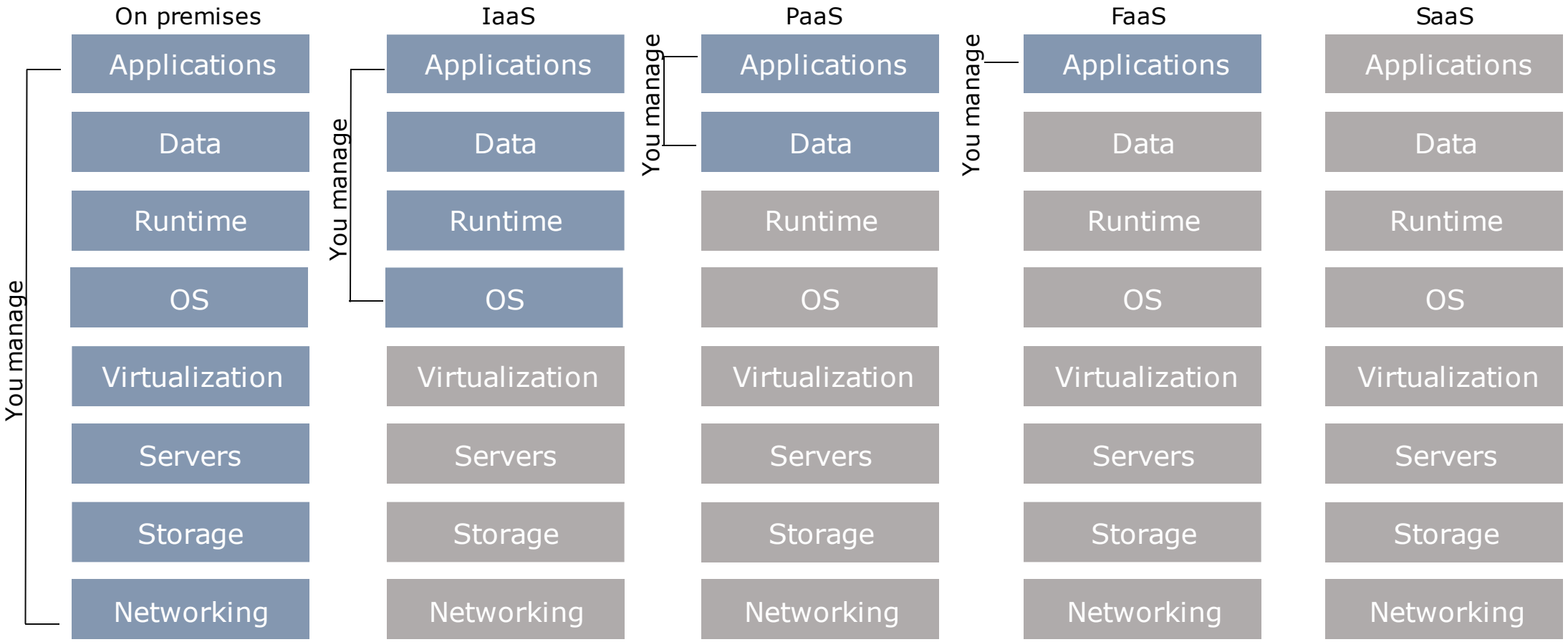
Availability zones in a region

- Are connected through low-latency links
- Resources are usually replicated across zones but not regions



https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

From PaaS to FaaS



From PaaS to FaaS

Understanding architectures is paramount to successful software systems

- Good architectures help to scale
- Poor architectures cause issues that necessitate a costly rewrite

On-premises

- Provisioning, managing, and patching servers is time-consuming
- Require dedicated operations people
- A non-trivial environment is hard to set up and operate effectively
- Infrastructure and hardware are often a distraction from strategic tasks

From PaaS to FaaS

Infrastructure as a service (IaaS)

- A computing infrastructure provisioned and managed over the internet (e.g., AWS EC2)
- Avoid expense/complexity of buying/managing physical servers/data-centers
- IaaS overcomes issues on-premises
- Possibly requires to manage many environments

Platform as a Service (PaaS)

- A development and deployment environment in the cloud (e.g., AWS Elastic Beanstalk)
- Support complete application life-cycle: building, testing, deploying, etc.
- Avoid expense/complexity of managing licenses and application infrastructure

From PaaS to FaaS

PaaS and **containers** are potential solutions to inconsistent infrastructures

PaaS provides a platform for users to run their software

- Developers write software targeting features/capabilities of the platform

Containerization isolates an application with its own environment

- Lightweight alternative to full virtualization
- Containers are isolated but need to be deployed to (public/private) server
- Excellent solution when dependencies are in play
- Housekeeping challenges and complexities

From PaaS to FaaS

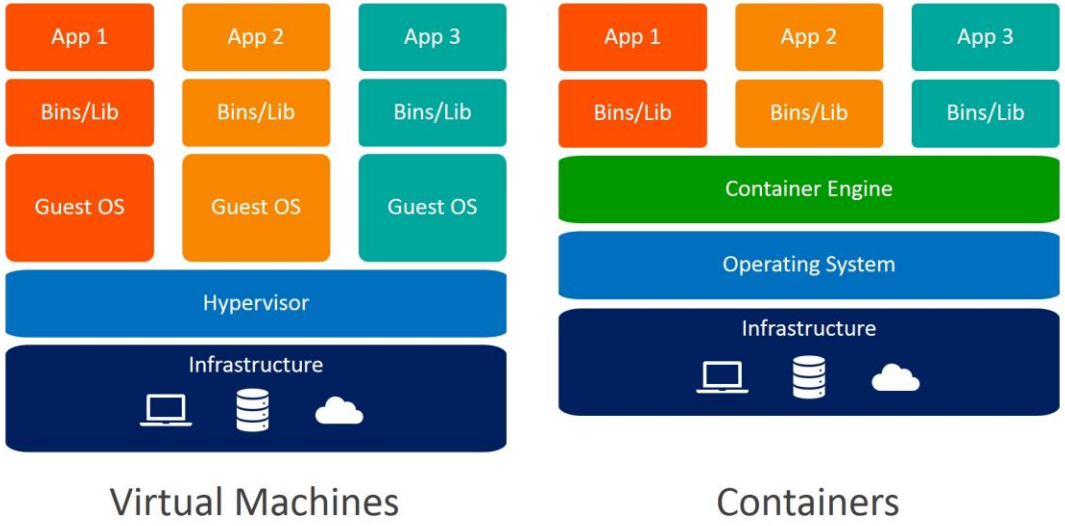
Containers and virtual machines are packaged computing environments

Containers

- On top of physical server and its host OS
- Share the host OS kernel
- Shared components are read-only
- Containers are “light” and take just seconds to start

Virtual machines

- Emulate a hardware/software system
- On top of a hypervisor (VM monitor)



From PaaS to FaaS

Function as a Service (FaaS)

- A coding environment, cloud provider provisions platform to run the code (e.g., AWS Lambda)
- Infrastructure provisioning and management are invisible to the developer

Software as a service (SaaS)

- An application environment
- Access cloud-based apps over the Internet (e.g., email, Microsoft Office 365)

From PaaS to FaaS

Serverless

- A software architecture that does not rely on direct access to a server
- Embodies principles from microservices
  - Small, standalone, fully independent services built for a specific purpose
  - Cloud provider is responsible for integration

Principles of FaaS architectures

- FaaS is based on a serverless approach
  - Use a compute service to execute code on demand (no servers/containers)
- Every function could be considered as a standalone service
- Write single-purpose stateless functions

From PaaS to FaaS

Functions react to events

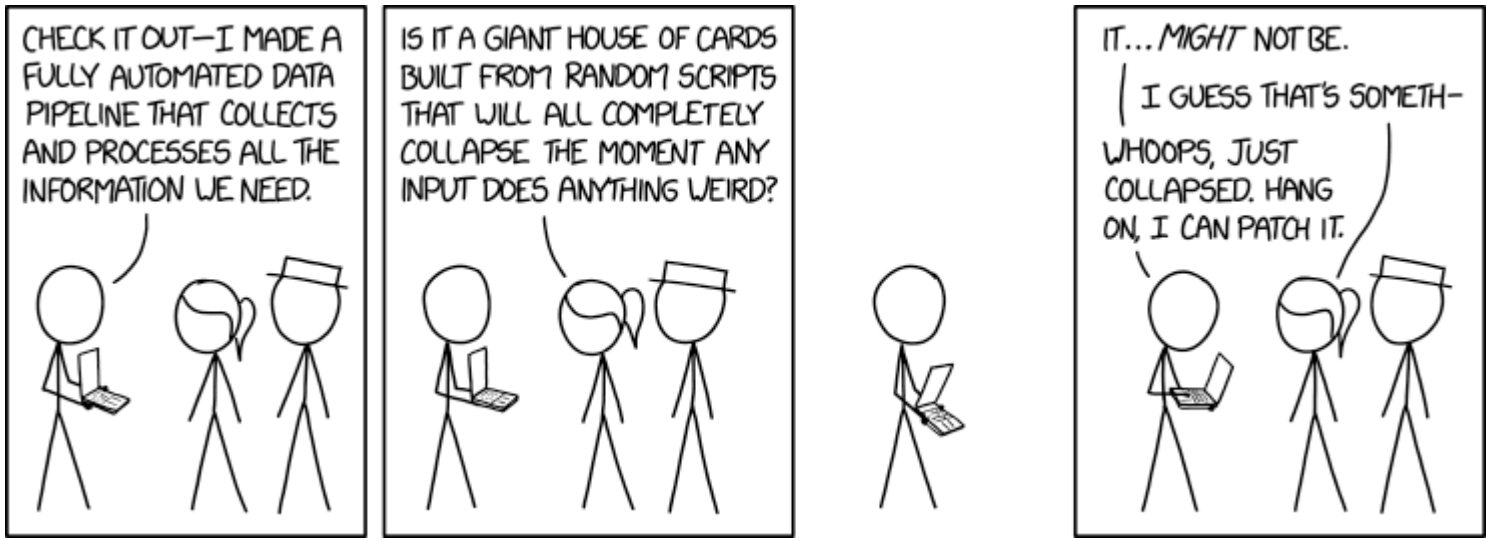
- Design push-based, event-driven pipelines
- Create thicker, more powerful front ends
- Embrace third-party services (e.g., security)

FaaS is not a silver bullet

- Not appropriate for latency-sensitive applications
- Strict specific service-level agreements
- Migration costs
- Vendor lock-in can be an issue

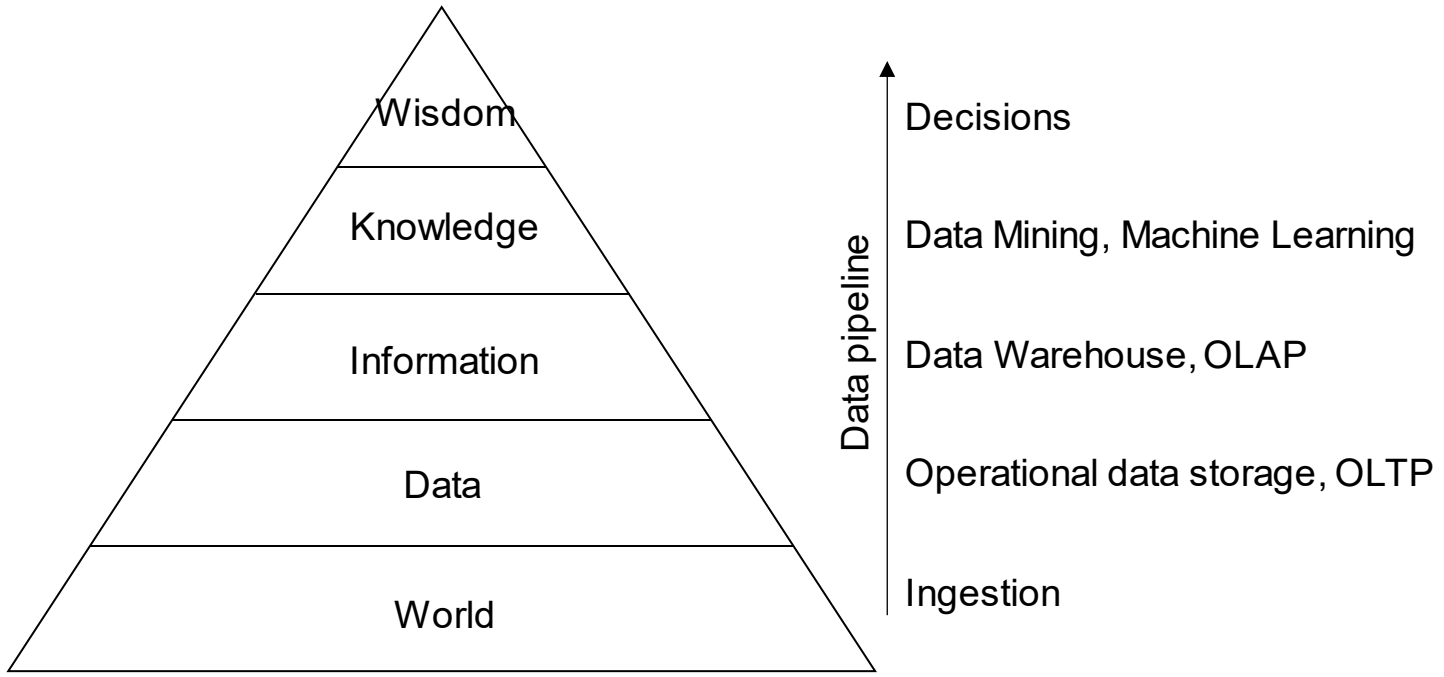
BIG DATA

Building data pipelines



https://xkcd.com/2054/

Data pipeline



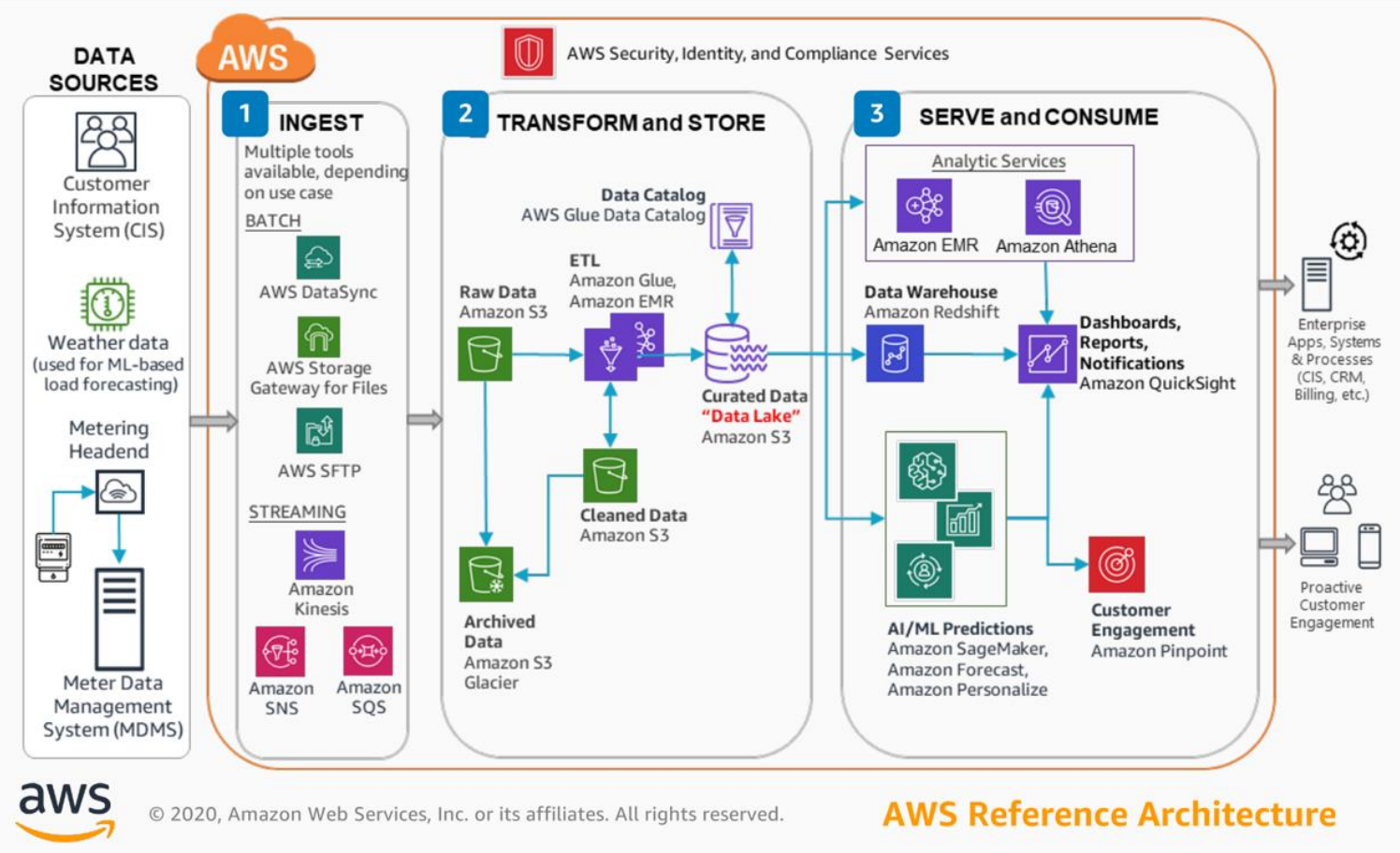
Ackoff, Russell L. "From data to wisdom." *Journal of applied systems analysis* 16.1 (1989): 3-9.

Date pipelines on cloud

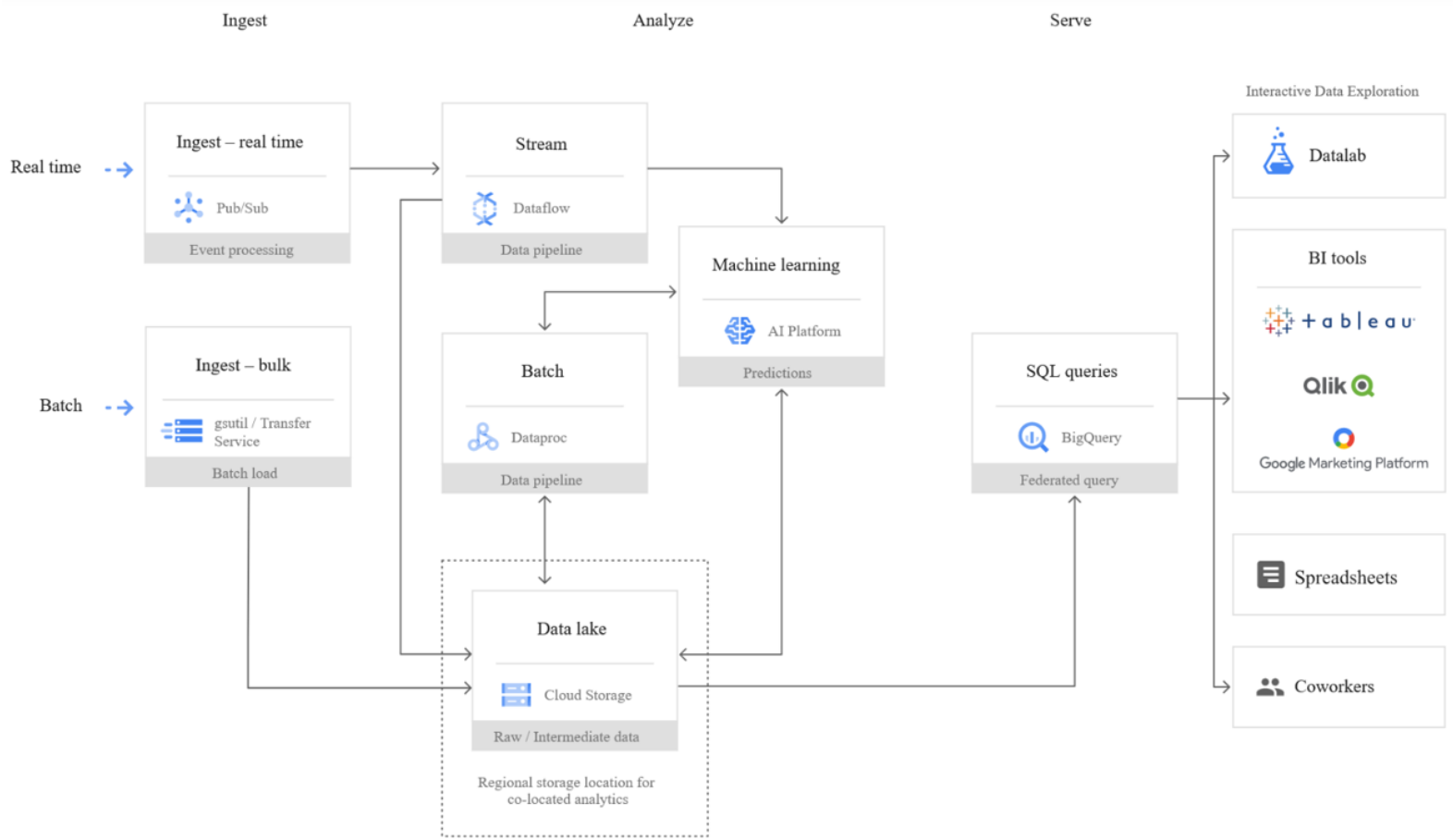
Architecting data pipelines on cloud requires to standardize services

- Make them available through simple portals
- Track usage/cost with billing mechanism
- Measure availability
- Orchestrate to meet demand
- Provide a security framework

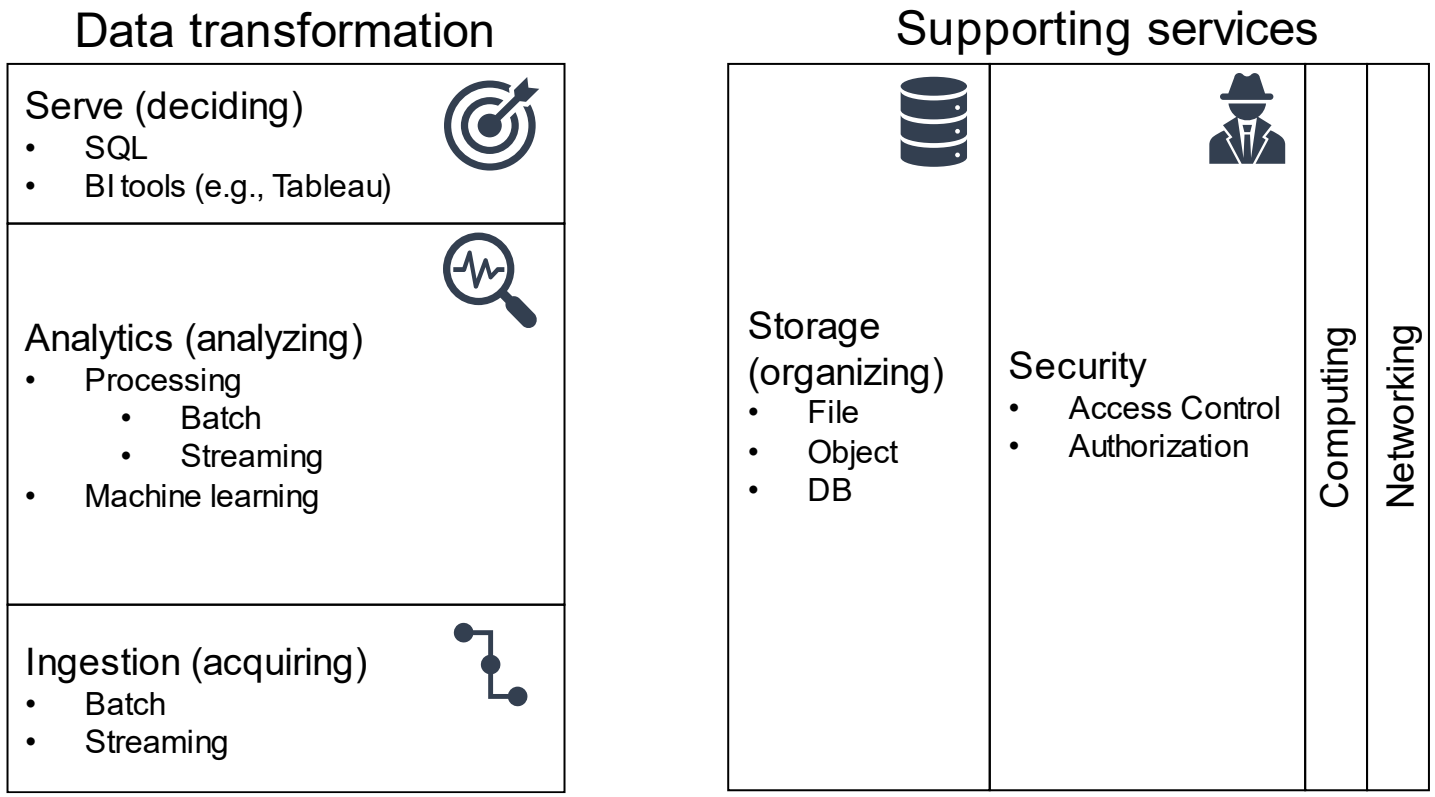
Data pipeline - AWS



Data pipeline - Google cloud



A tentative organization



A tentative organization

This is not a sharp taxonomy

Ingestion vs Analytics

- Data streams are used for ingestion
- ... and (event) processing

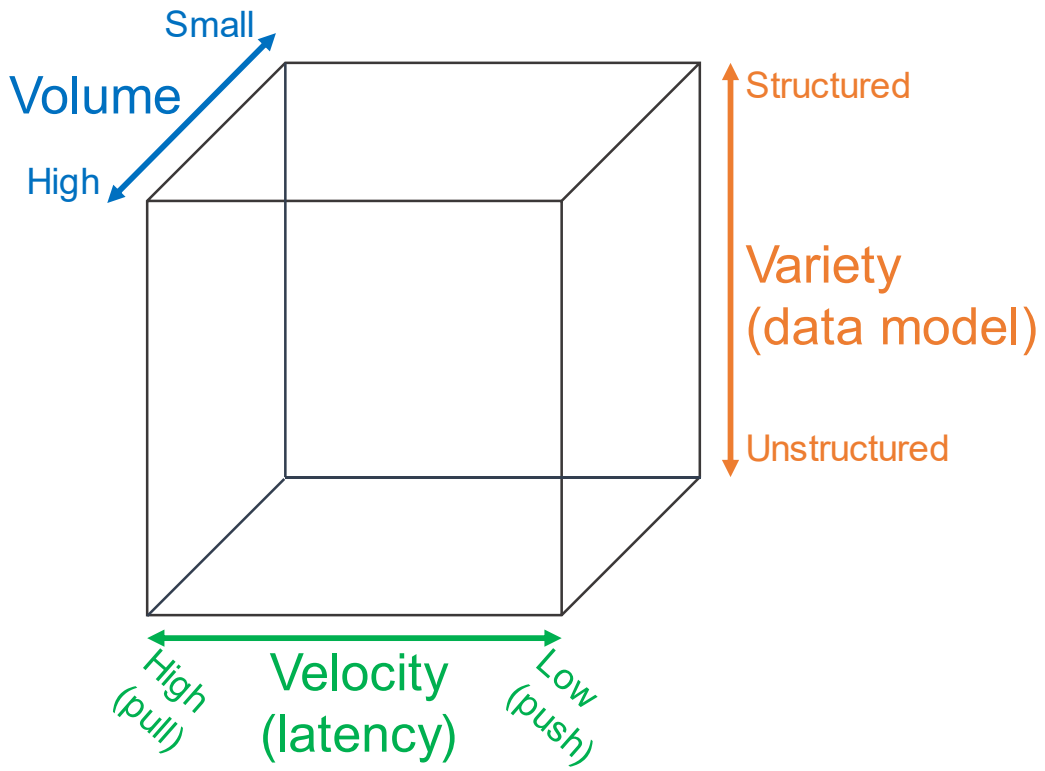
Storage vs Database

- Databases are storage
- ... with processing capability
- ... and with serving capability

A tentative organization

The big-data cube

- Volume: small to high
- Variety: structure to unstructured
- Velocity: pull to push

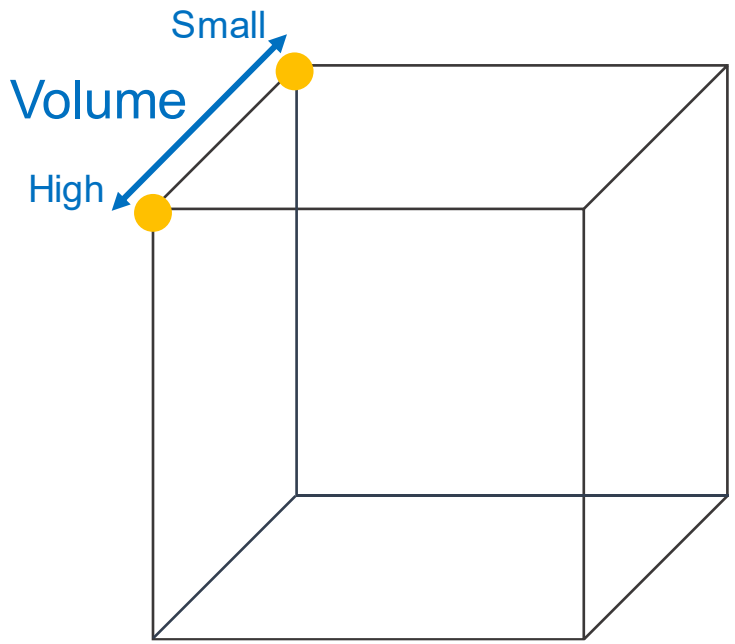


Meijer, Erik. "Your mouse is a database." *Communications of the ACM* 55.5 (2012): 66-73.

A tentative organization

Volume

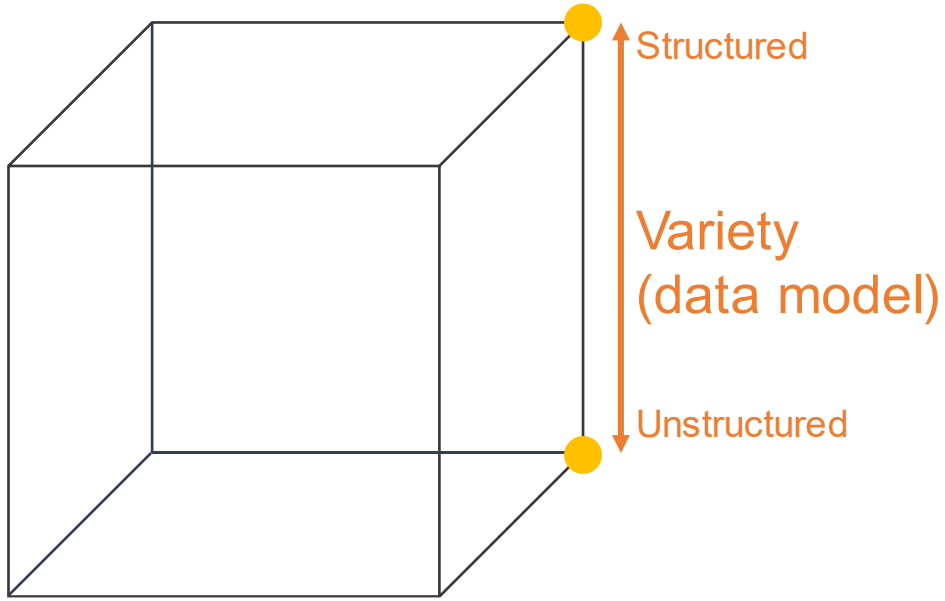
- **Small**: small relational DBs
- **High**: TBs o PBs of data



A tentative organization

Variety

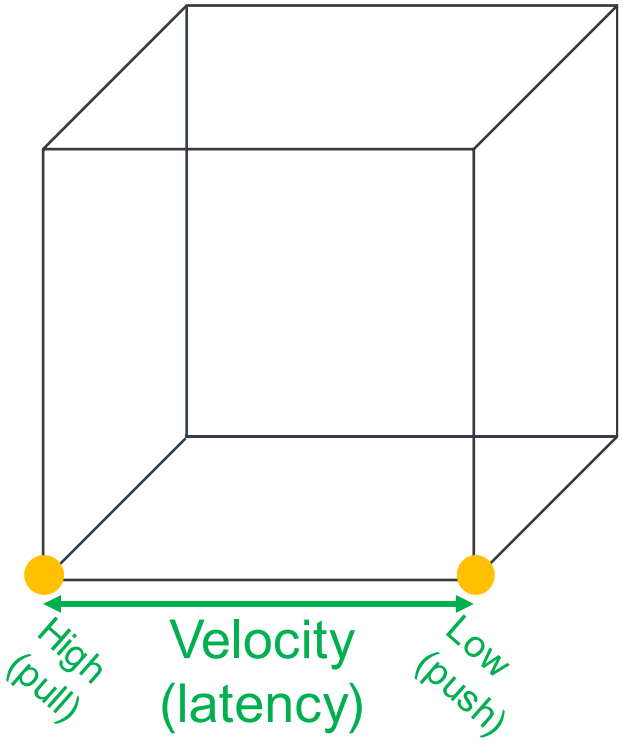
- **Structured**: relational tuples with
  - FK/PK relationships
- **Unstructured**
  - Key-value
  - Columnar
  - Document-based
  - Graph
  - ...



A tentative organization

Velocity (latency)

- **High**: clients synchronously pulling
  - Data from sources
- **Low**: sources asynchronously
  - Pushing data to clients

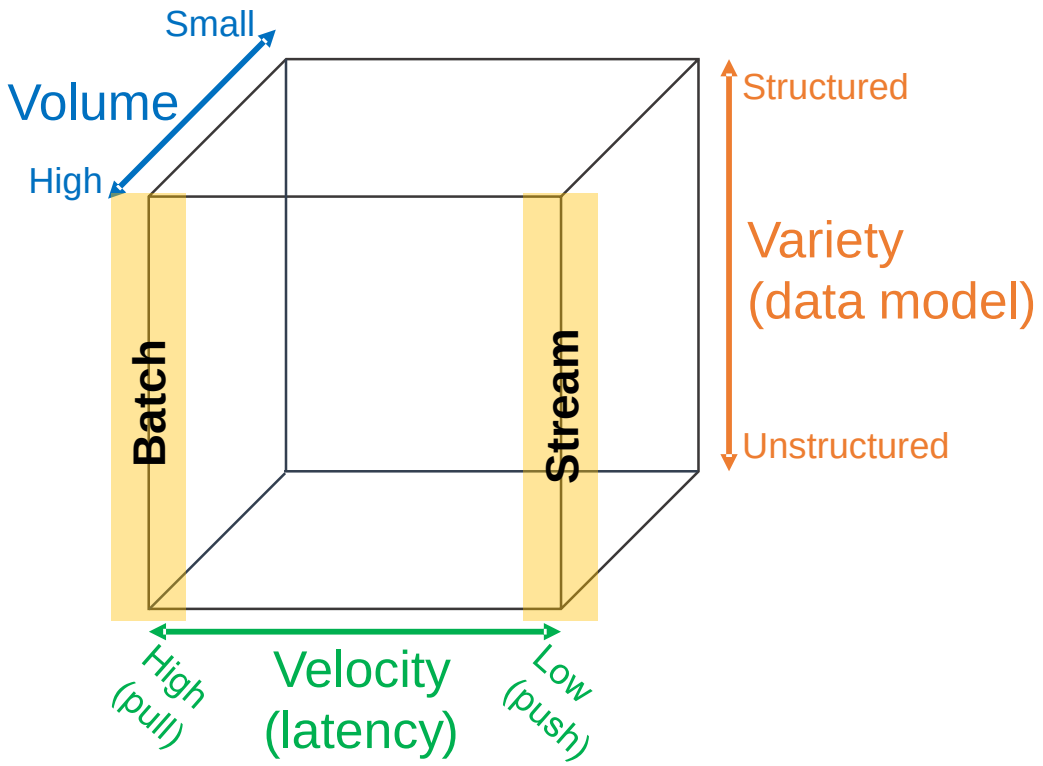


A tentative organization

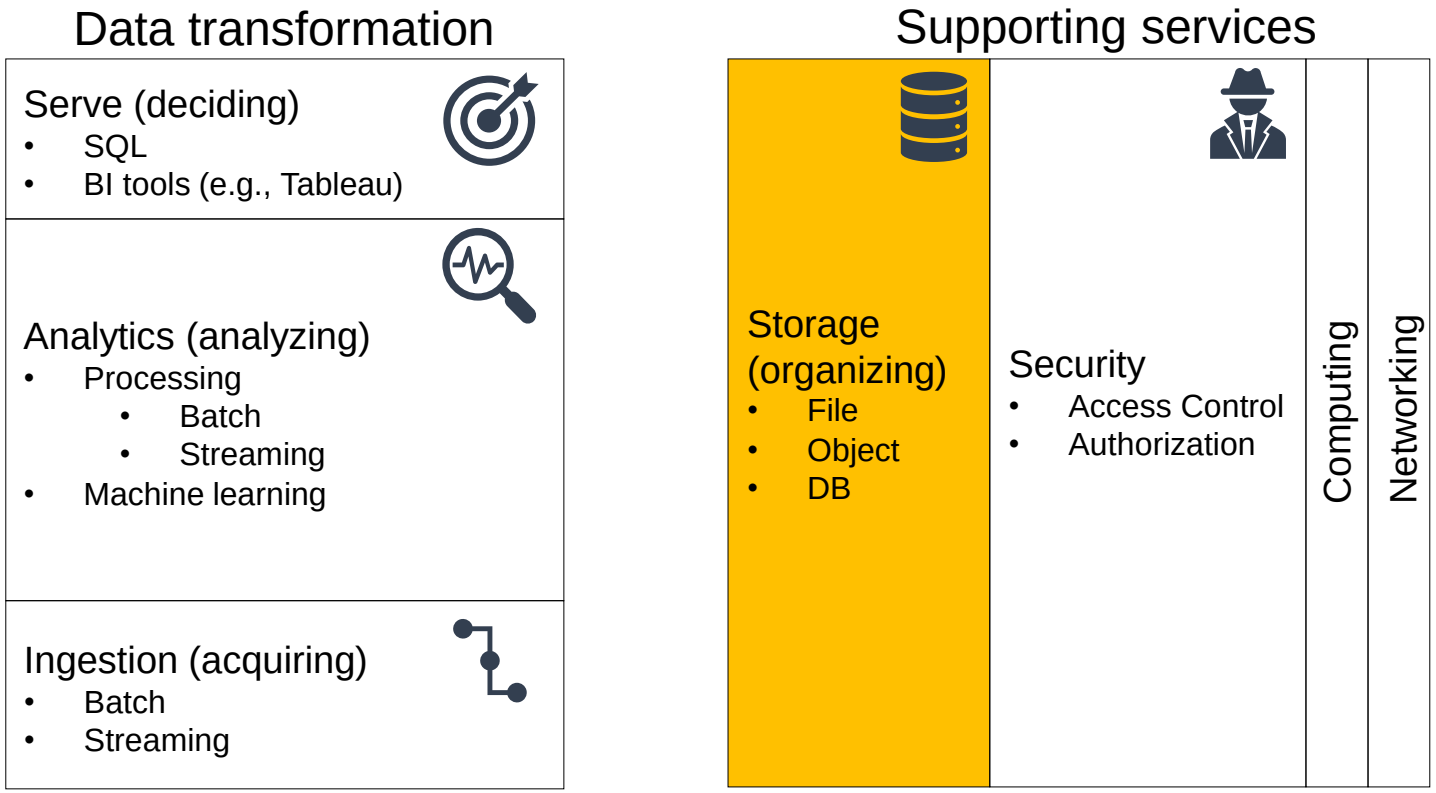
Our focus

- (Un)Structured big-data batch
- (Un)Structured big-data streams

Goal: keep in mind the cube to categorize the following services



A tentative organization



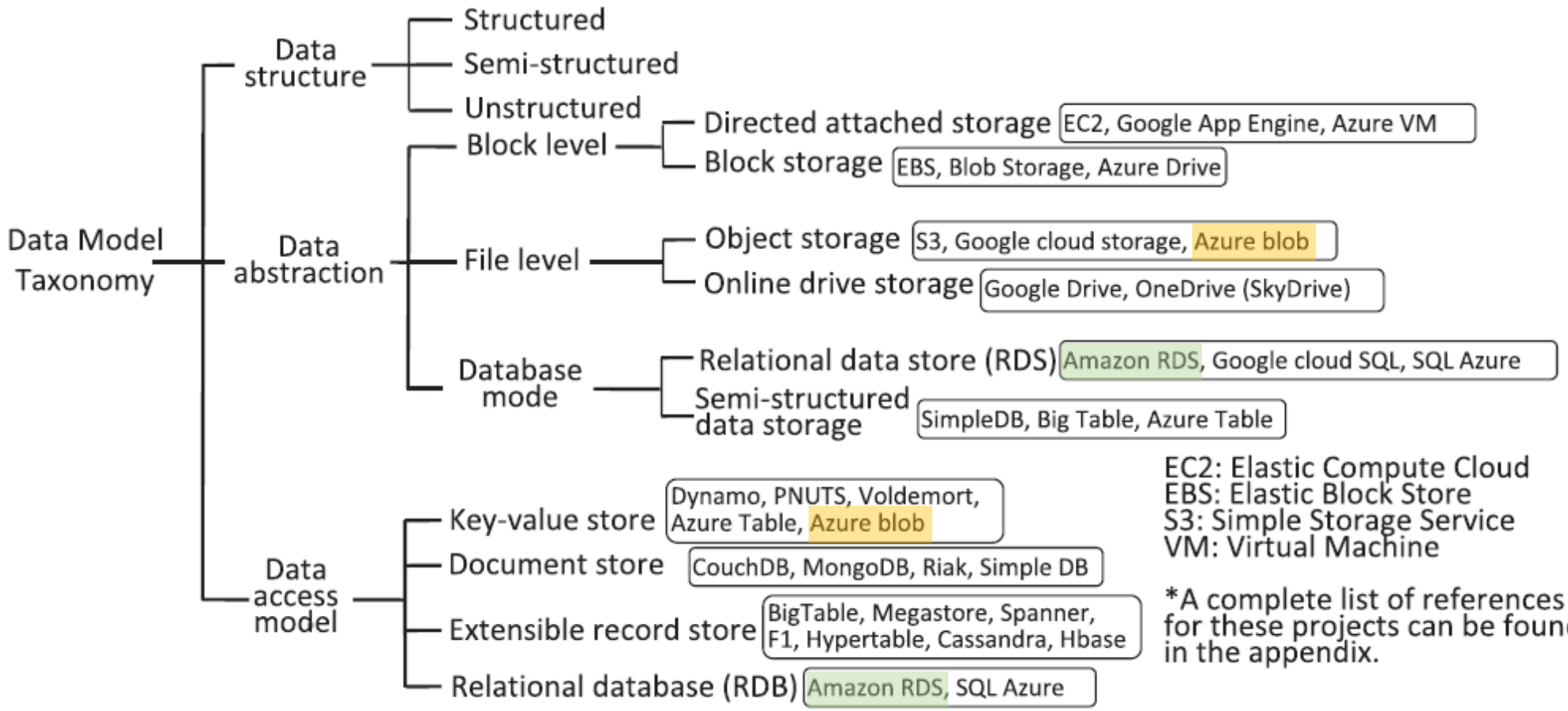
Storage

Goal: persisting data

Which storage do we choose?

- **Storage model** (or data model) ~= variety
  - How data are organized/accessed in a storage system
    - Structured vs unstructured
    - Data access model (key-value, column, etc.)
- Access **frequency**
- **Analyses** to be performed

Storage models



Mansouri, Yaser, Adel Nadjaran Toosi, and Rajkumar Buyya. "Data storage management in cloud environments: Taxonomy, survey, and future directions." ACM Computing Surveys (CSUR) 50.6 (2017): 1-51.

Storage models

Resemble the notion of **variety**

- **Relational** store data with predefined schemas and relationships between them. Support ACID transactions and maintain referential integrity

Database type	Use cases	AWS service
Relational	Traditional applications, ERP, CRM, e-commerce	Amazon Aurora, Amazon RDS, Amazon Redshift
Key-value	High-traffic web apps, e-commerce systems, gaming applications	Amazon DynamoDB
In-memory	Caching, session management, gaming leaderboards, geospatial applications	Amazon ElastiCache for Memcached, Amazon ElastiCache for Redis
Document	Content management, catalogs, user profiles	Amazon DocumentDB (with MongoDB compatibility)
Wide column	High scale industrial apps for equipment maintenance, fleet management, and route optimization	Amazon Keyspaces (for Apache Cassandra)
Graph	Fraud detection, social networking, recommendation engines	Amazon Neptune
Time series	IoT applications, DevOps, industrial telemetry	Amazon Timestream
Ledger	Systems of record, supply chain, registrations, banking transactions	Amazon QLDB

https://aws.amazon.com/products/databases/

Storage models

Resemble the notion of **variety**

- **Key/value** optimized for common access patterns, typically to store and retrieve large volumes of data
- **Document** store semi-structured data as JSON-like documents
- **Columnar** it uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table
- **Graph** to navigate and query millions of relationships between highly connected datasets with ms latency at large scale
- **... and more**

Database type	Use cases	AWS service
Relational	Traditional applications, ERP, CRM, e-commerce	Amazon Aurora, Amazon RDS, Amazon Redshift
Key-value	High-traffic web apps, e-commerce systems, gaming applications	Amazon DynamoDB
In-memory	Caching, session management, gaming leaderboards, geospatial applications	Amazon ElastiCache for Memcached, Amazon ElastiCache for Redis
Document	Content management, catalogs, user profiles	Amazon DocumentDB (with MongoDB compatibility)
Wide column	High scale industrial apps for equipment maintenance, fleet management, and route optimization	Amazon Keyspaces (for Apache Cassandra)
Graph	Fraud detection, social networking, recommendation engines	Amazon Neptune
Time series	IoT applications, DevOps, industrial telemetry	Amazon Timestream
Ledger	Systems of record, supply chain, registrations, banking transactions	Amazon QLDB

https://aws.amazon.com/products/databases/

Storage: analysis type

	Cloud Datastore	Bigtable	Cloud Storage	Cloud SQL	Cloud Spanner	BigQuery
Type	NoSQL document	NoSQL wide column	Blobstore	Relational SQL for OLTP	Relational SQL for OLTP	Relational SQL for OLAP
Transactions	Yes	Single-row	No	Yes	Yes	No
Complex queries	No	No	No	Yes	Yes	Yes
Capacity	Terabytes+	Petabytes+	Petabytes+	Terabytes	Petabytes	Petabytes+
Unit size	1 MB/entity	~10 MB/cell ~100 MB/row	5 TB/object	Determined by DB engine	10,240 MiB/ row	10 MB/row

	Cloud Datastore	Cloud Bigtable	Cloud Storage	Cloud SQL	Cloud Spanner	BigQuery
Type	NoSQL document	NoSQL wide column	Blobstore	Relational SQL for OLTP	Relational SQL for OLTP	Relational SQL for OLAP
Best for	Semi-structured application data, durable key-value data	"Flat" data, Heavy read/write, events, analytical data	Structured and unstructured binary or object data	Web frameworks, existing applications	Large-scale database applications (> ~2 TB)	Interactive querying, offline analytics
Use cases	Getting started, App Engine applications	AdTech, Financial and IoT data	Images, large media files, backups	User credentials, customer orders	Whenever high I/O, global consistency is needed	Data warehousing

https://cloud.google.com/products/databases

Storage models

File system (EFS), **object storage** (S3), **key-value** (DynamoDB)

- Handle unstructured data
- ... organized as files (or blob)
- ... accessed using a key-value

Differ in the supported features

- E.g., maximum item size
  - DynamoDB: 400KB
  - S3: 5TB
- E.g., indexes, querying mechanisms, etc.

Storage: access frequency

Object storage (S3) **classes**

- **Standard**: general purpose
- **Intelligent-Tiering**: move objects between access tiers when access patterns change
- **Infrequent** (rapid) **access**
- **One Zone-IA**: lower-cost option for infrequently accessed data that do not require high availability and resilience
- **Glacier**: low-cost storage class for data archiving, three retrieval options that range from a few minutes to hours
- **Deep Glacier**: long-term retention for data accessed once or twice in a year. E.g., retain data sets for 10 years or longer

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive
Designed for durability	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%
Availability Zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved
First byte latency	milliseconds	milliseconds	milliseconds	milliseconds	select minutes or hours	select hours
Storage type	Object	Object	Object	Object	Object	Object
Lifecycle transitions	Yes	Yes	Yes	Yes	Yes	Yes

https://aws.amazon.com/s3/storage-classes/

Storage: access frequency

Lifecycle configuration

- A set of rules that define actions that Amazon S3 applies to a group of objects

Two types of actions:

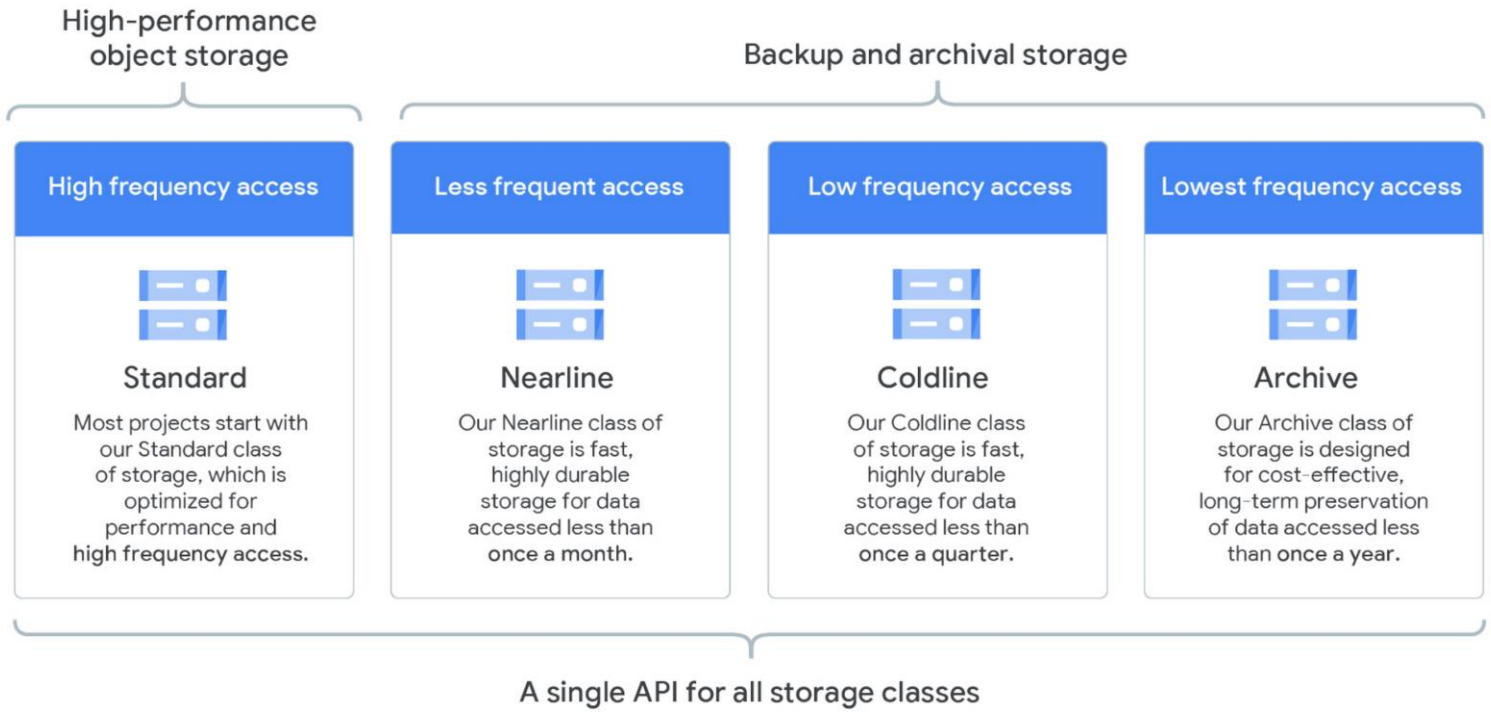
- **Transition:** when objects transition to another storage class. E.g., archive objects to the S3 Glacier storage class one year after creating them
- **Expiration**: when objects expire. Amazon S3 deletes expired objects on your behalf

Transition

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive
Designed for durability	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%
Availability Zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved
First byte latency	milliseconds	milliseconds	milliseconds	milliseconds	select minutes or hours	select hours
Storage type	Object	Object	Object	Object	Object	Object
Lifecycle transitions	Yes	Yes	Yes	Yes	Yes	Yes

https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lifecycle-mgmt.html

Storage: access frequency

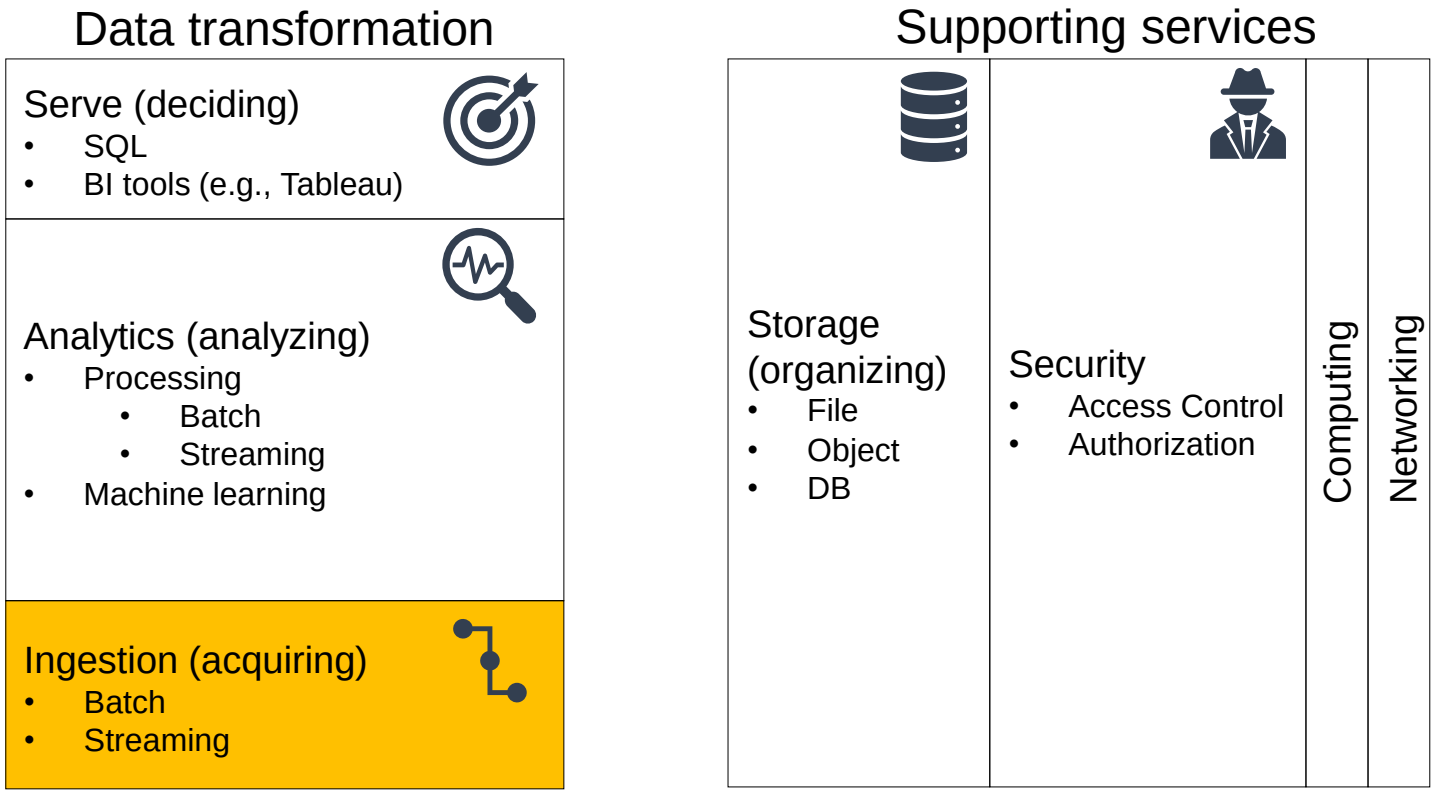


Retention Period

Access Frequency	<1 mo	1–3 mo	3–12 mo	>12 mo
>12/yr	Standard	Standard	Standard	Standard
4–12/yr	Standard	Nearline	Nearline	Nearline
1–4/yr	Standard	Nearline	Coldline	Coldline
<1/yr	Standard	Nearline	Coldline	Archive

https://cloud.google.com/blog/products/storage-data-transfer/archive-storage-class-for-coldest-data-now-available

A tentative organization



Ingestion: bulk

Goal: moving data to the cloud

Moving data to the cloud

- 80TB of data to move,
- 1Gbps connection to the internet

How many days?

- 80000GB / (1Gbps / 8) / 60 / 60 / 24 ~= a week without internet

Ingestion: bulk

Batch/Bulk: move data from on-premises storage

Workflow

- Receive shipment
- Set up
- Transfer data
- Ship back (shipping carrier)

Ingestion: bulk

AWS Snowball

- 50TB (North America only) and 80TB versions
- Not rack-mountable

Throughput

- 1 Gbps or 10 Gbps using an RJ-45 connection
- 10 Gbps using a fiber optic connection

https://aws.amazon.com/snowball/


amazon
web services

Ingestion: stream

Stream: real-time streaming data

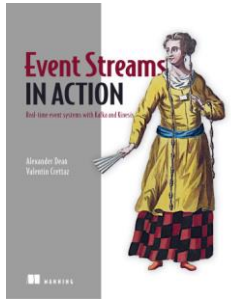
Event: anything that we can observe occurring at a particular point in time

Continuous streaming

- Illimited succession of individual events
- Ordered by the point in time at which each event occurred

Publish/subscribe (pub/sub): a way of communicating messages

- Senders publish messages associated with one or more **topics**
- Receivers subscribe to specific topics, receive all messages with that topic
- Messages are events



https://www.manning.com/books/event-streams-in-action

Ingestion: stream

Unified log

- Unified, append-only, ordered, distributed log that allows the centralization of event streams

General idea:

- Collect events from disparate source systems
- Store them in a unified log
- Enable data processing applications to operate on these event streams

Ingestion: stream

Unified: a single log in a company with applications sending/reading events

- Log serves as central data backbone
- Unified log can contain many distinct continuous streams of events
- Not all events are sent to the same event stream

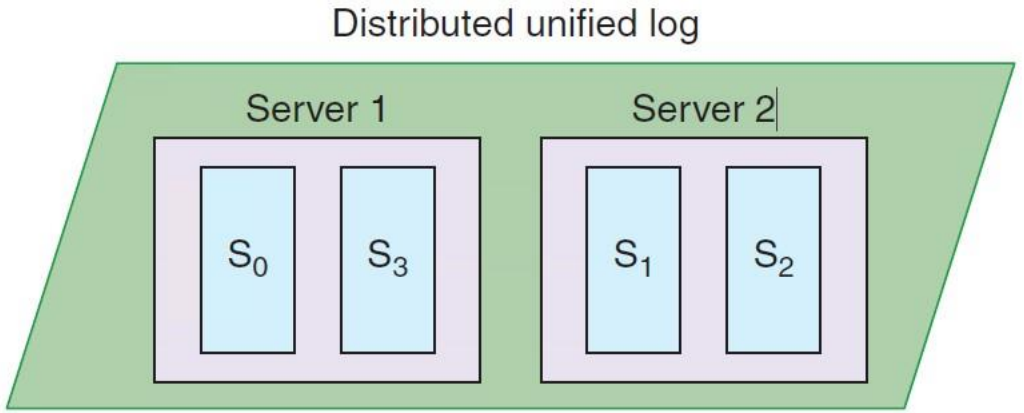
Append-only: new events are appended to the unified log

- Existing events are never updated in place
- If read the event #10, never look at events 1 through 10 again
- Events are automatically deleted from the unified log when they age
  - E.g., automatically remove events after 7 days

Ingestion: stream

Distributed: the unified log lives across a cluster of machines

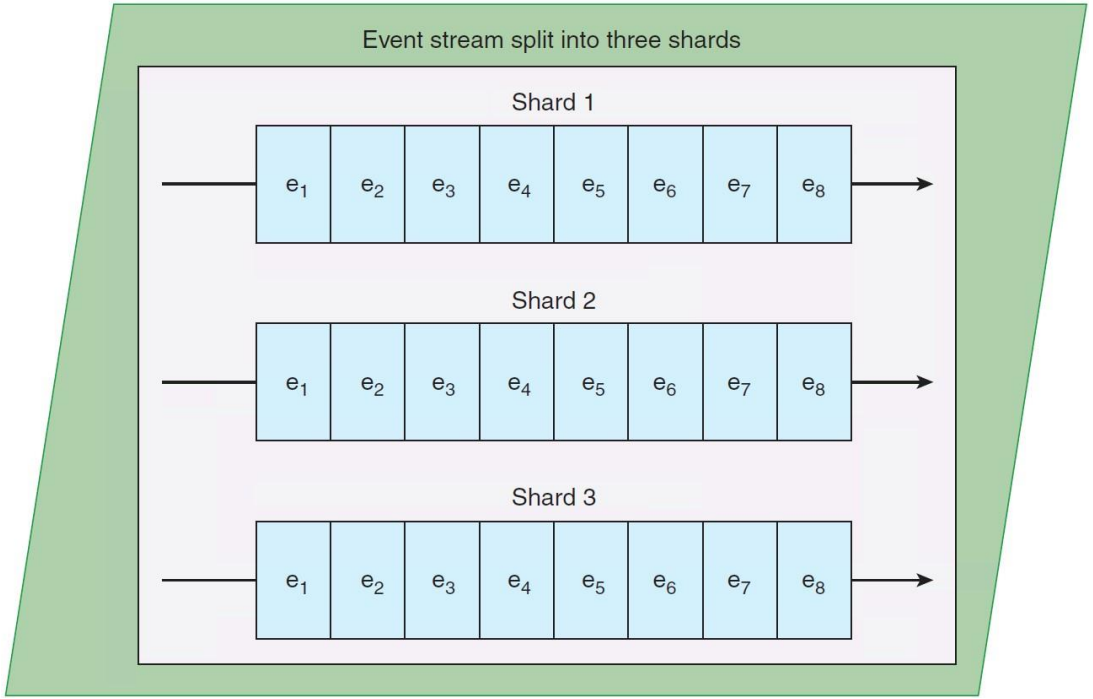
- Still, the log is unified since we have a single (conceptual) log
- Divide events into shards (partitions)
- Scalability: work with streams larger than the capacity of single machines
- Durability: replicate all events within the cluster to overcome data loss



Ingestion: stream

Ordered: events in a shard have a sequential IDs (unique in a shard)

- Local ordering keeps things much simpler
- Applications maintain their own cursor for each shard

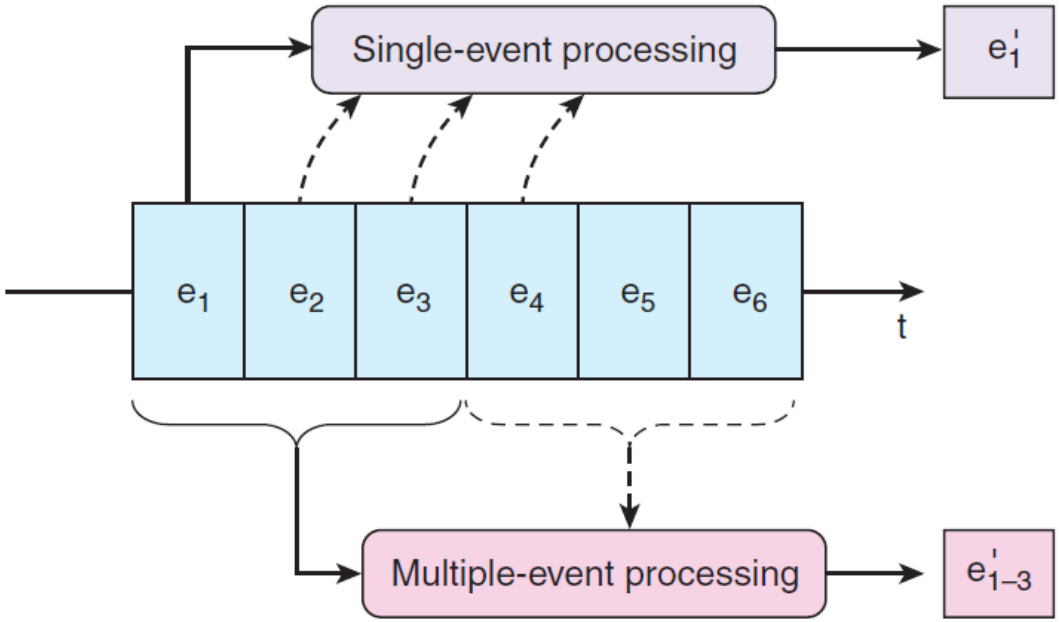


Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." *Concurrency: the Works of Leslie Lamport*. 2019. 179-196.

Ingestion: stream

Two types of processing

- **Single-event**, a single event produces zero or more events
  - Validating “Does this event contain all the required fields?”
  - Enriching “Where is this IP address located?”
  - Filtering “Is this error critical?”
- **Multiple-event**, multiple events collectively produce zero or more events
  - Aggregating, functions such as minimum, maximum, sum
  - Pattern matching, looking for patterns or co-occurence
  - Reordering events based on a sort key



Ingestion: stream

Amazon Kinesis Data Streams

- Producers send data to a stream that you create and provision by shard
- This data is stored in data records that consist of
  - An incremental sequence number
  - A user-supplied partition key
    - Load-balance records across shards
  - A data blob
- Data order
  - Maintains order through partition key and sequence number
  - Producer provides a partition key that determines the shard
  - The shard adds an incremental sequence number to the record
  - Consumers get records by shard, records ordered by sequence number
  - Ordering is not guaranteed for requests across shards
- Each shard provides a maximum of 1 MBps and 1000 data puts per second
- By default, records are retained for 24 hours (maximum of 7 days)

Ingestion: stream

Operations

- Scaling process is called resharding
  - Split a shard into two, or merge two shards
  - Users must scale shards up and down manually
  - Monitor usage with Amazon CloudWatch and modify scale as needed
- Avoid shard management by using Kinesis Data Firehose
- Automate management to aggregating data into S3/Redshift

Kinesis is a regional service, with streams scoped to specific regions

- All ingested data must travel to the region in which the stream is defined

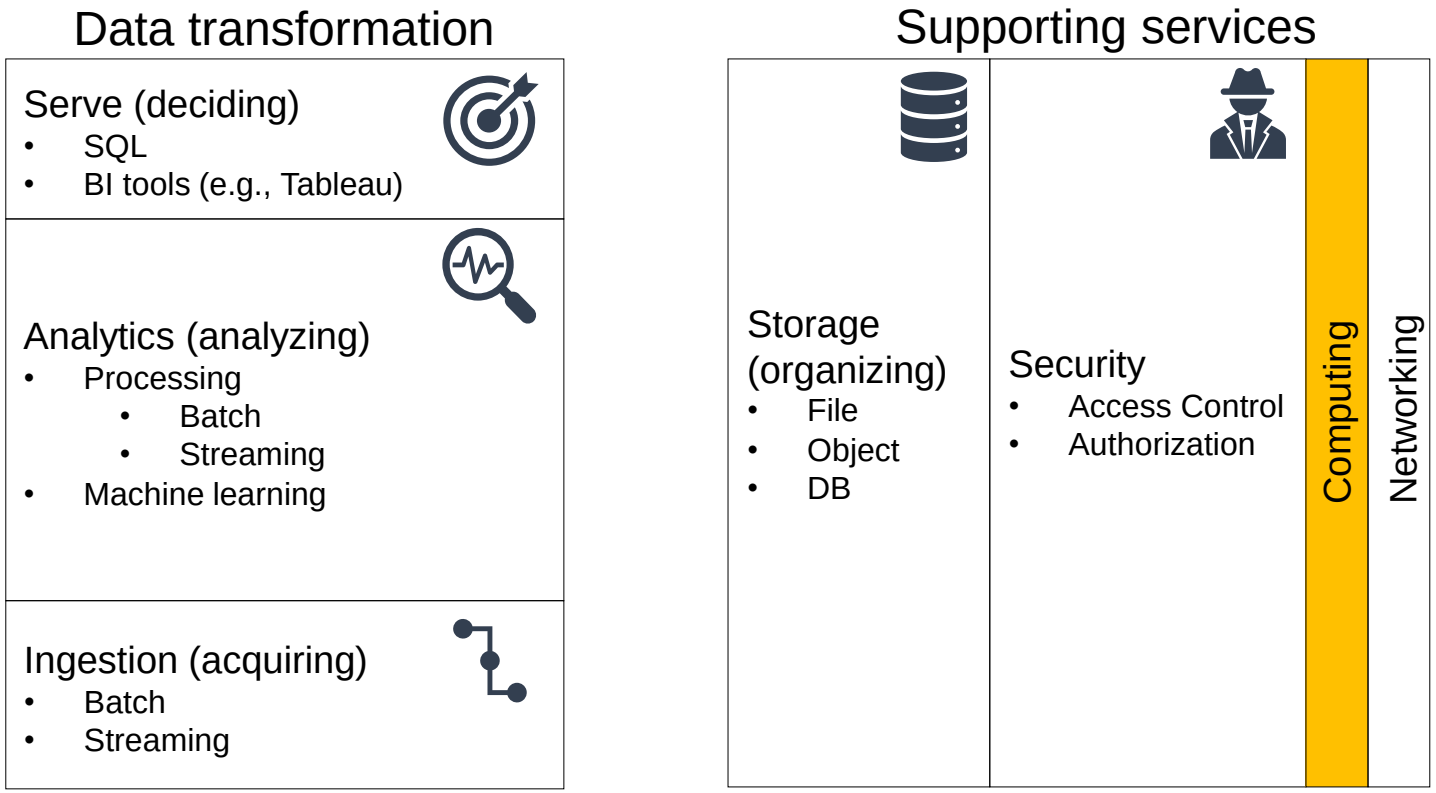
Costs

- Priced by shard hour, data volume, and data retention period
- Pay for resources you provision (even if not used)
- Amazon Kinesis Data Firehose is priced by data volume

Ingestion: stream

Feature	AWS Kinesis	Google Pub/Sub
Unit of deployment	Stream	Topic
Unit of provisioning	Shard	N/A (fully managed)
Data unit	Record	Message
Data producer/destination	Producer/Consumer	Publisher/Subscriber
Data partitioning	User-supplied partition key	N/A (fully managed)
Retention period	Up to 7 days	Up to 7 days
Pricing	Per shard-hour, PUT payload units, and optional data retention	Message ingestion and delivery, and optional message retention

A tentative organization



Single instance: EC2

Amazon Elastic Compute Cloud (Amazon EC2)

- A web service that provides secure, resizable compute capacity
- Complete control of your computing resources
- Processor, storage, networking, operating system, and purchase model

Amazon Machine Image is a template that contains a software configuration

- E.g., an operating system, an application server, and applications
- From an AMI, you launch an instance, which is a copy of the AMI running as a virtual server
- You can launch multiple instances of an AMI

https://aws.amazon.com/ec2/

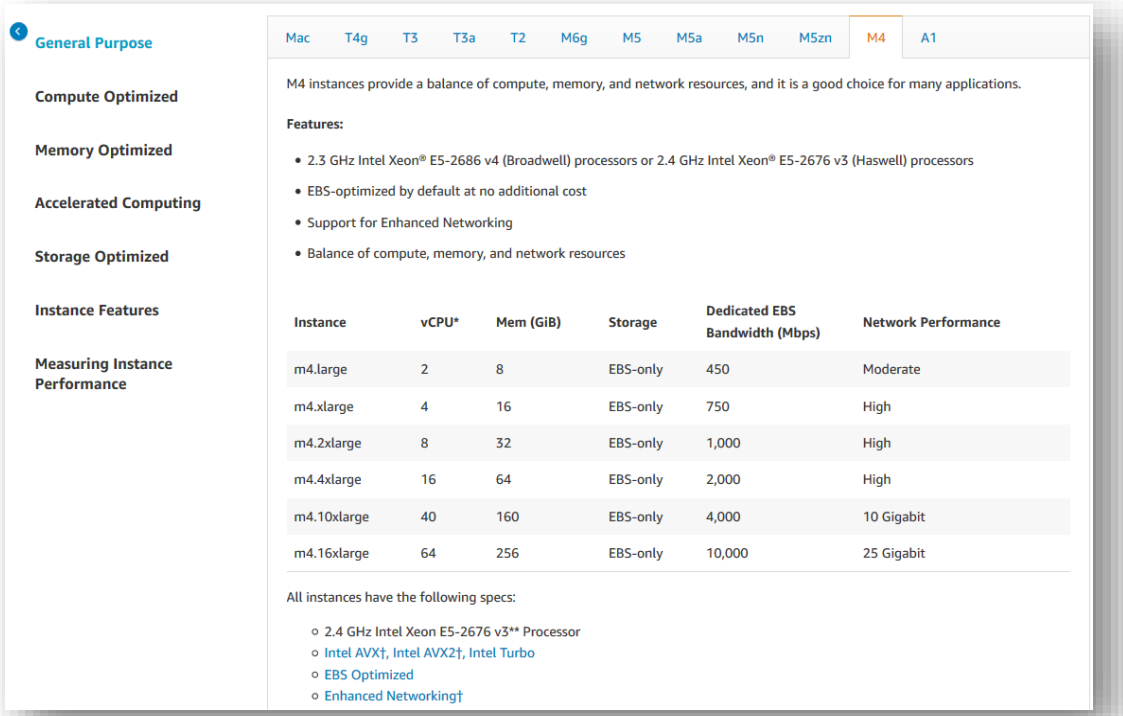
Single instance: EC2

An instance type determines the hardware of the host computer

- Each instance type offers different compute and memory capabilities
- After launch, you can interact with it as you would any computer
- You have complete control of your instances
  - E.g., `sudo` to run commands

https://aws.amazon.com/ec2/instance-types/

Single instance: EC2



General Purpose
Compute Optimized
Memory Optimized
Accelerated Computing
Storage Optimized
Instance Features
Measuring Instance Performance

Mac | T4g | T3 | T3a | T2 | M6g | M5 | M5a | M5n | M5zn | M4 | A1

M4 instances provide a balance of compute, memory, and network resources, and it is a good choice for many applications.

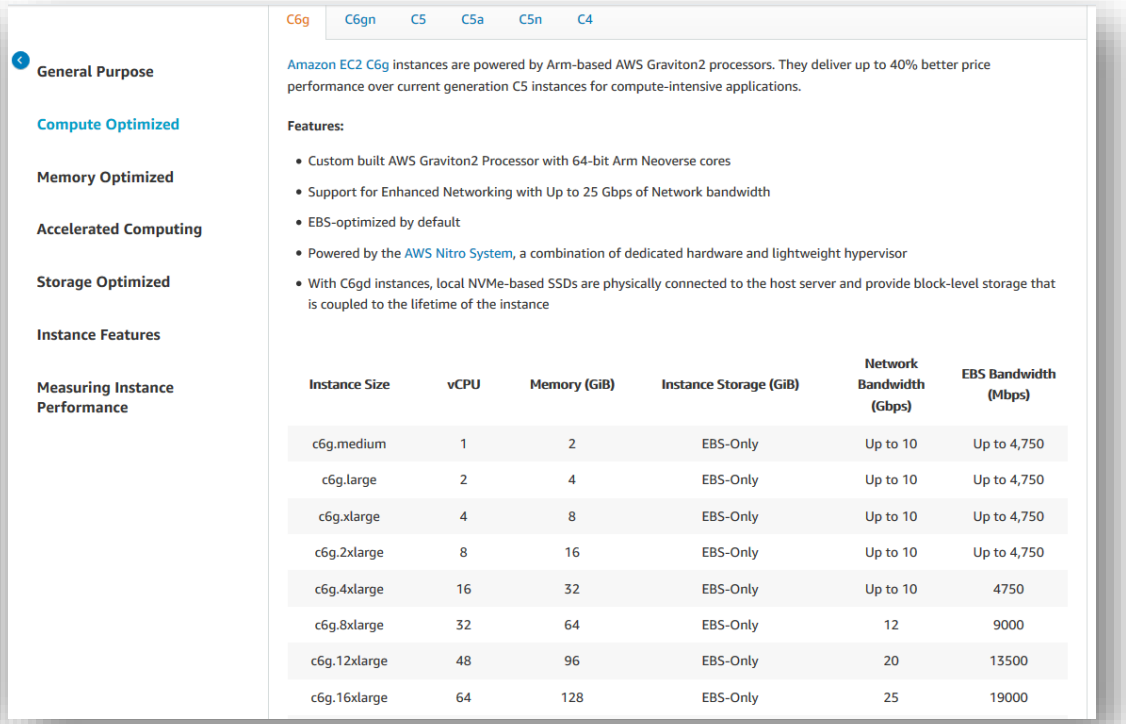
Features:

- 2.3 GHz Intel Xeon® E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors
- EBS-optimized by default at no additional cost
- Support for Enhanced Networking
- Balance of compute, memory, and network resources

Instance	vCPU*	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)	Network Performance
m4.large	2	8	EBS-only	450	Moderate
m4.xlarge	4	16	EBS-only	750	High
m4.2xlarge	8	32	EBS-only	1,000	High
m4.4xlarge	16	64	EBS-only	2,000	High
m4.10xlarge	40	160	EBS-only	4,000	10 Gigabit
m4.16xlarge	64	256	EBS-only	10,000	25 Gigabit

All instances have the following specs:

- 2.4 GHz Intel Xeon E5-2676 v3** Processor
- Intel AVX†, Intel AVX2†, Intel Turbo
- EBS Optimized
- Enhanced Networking†



General Purpose
Compute Optimized
Memory Optimized
Accelerated Computing
Storage Optimized
Instance Features
Measuring Instance Performance

C6g | C6gn | C5 | C5a | C5n | C4

Amazon EC2 C6g instances are powered by Arm-based AWS Graviton2 processors. They deliver up to 40% better price performance over current generation C5 instances for compute-intensive applications.

Features:

- Custom built AWS Graviton2 Processor with 64-bit Arm Neoverse cores
- Support for Enhanced Networking with Up to 25 Gbps of Network bandwidth
- EBS-optimized by default
- Powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor
- With C6gd instances, local NVMe-based SSDs are physically connected to the host server and provide block-level storage that is coupled to the lifetime of the instance

Instance Size	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)	EBS Bandwidth (Mbps)
c6g.medium	1	2	EBS-Only	Up to 10	Up to 4,750
c6g.large	2	4	EBS-Only	Up to 10	Up to 4,750
c6g.xlarge	4	8	EBS-Only	Up to 10	Up to 4,750
c6g.2xlarge	8	16	EBS-Only	Up to 10	Up to 4,750
c6g.4xlarge	16	32	EBS-Only	Up to 10	4750
c6g.8xlarge	32	64	EBS-Only	12	9000
c6g.12xlarge	48	96	EBS-Only	20	13500
c6g.16xlarge	64	128	EBS-Only	25	19000

https://aws.amazon.com/ec2/instance-types/

Cluster: EMR

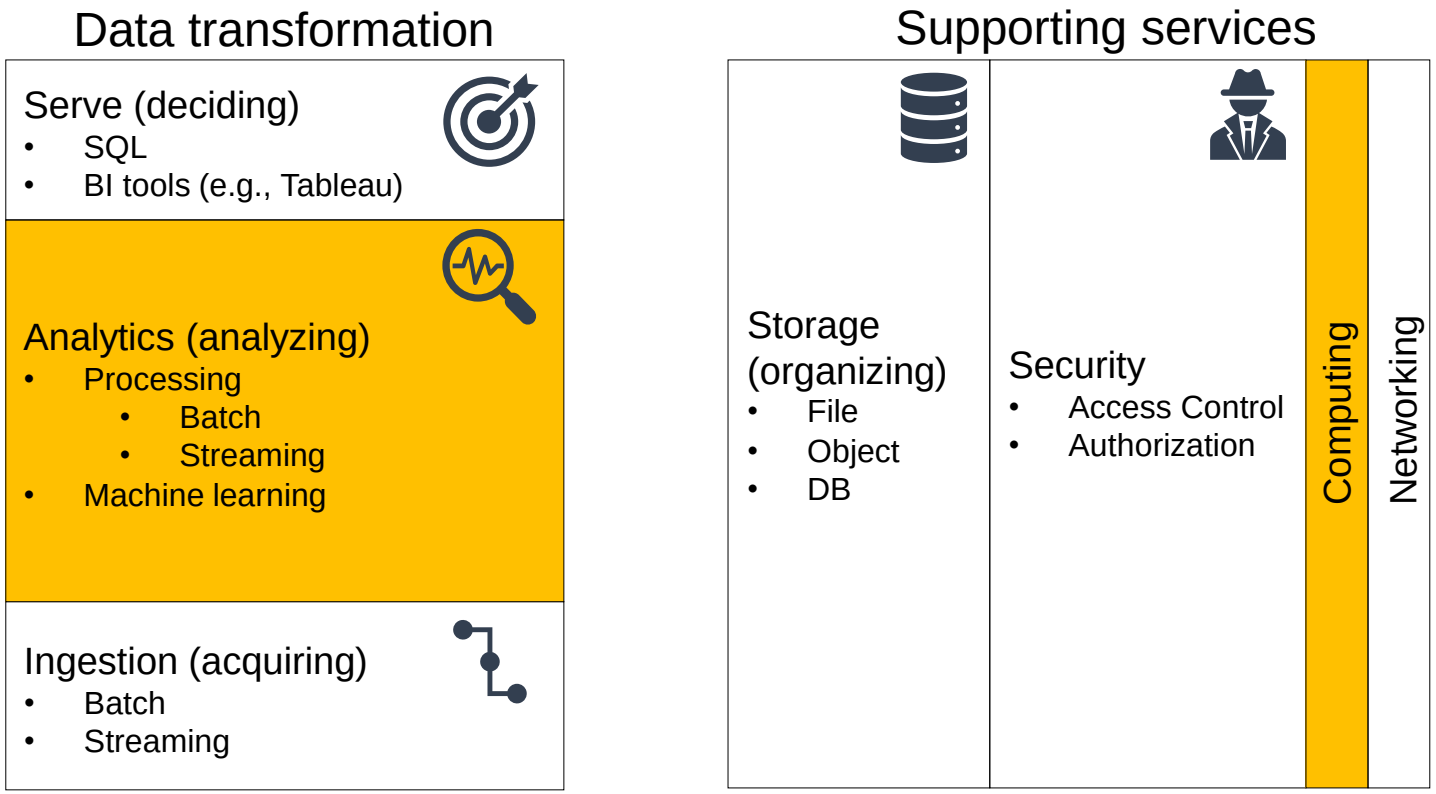
Amazon EMR is a data platform for processing data using the Hadoop stack

- Apache Spark, Apache Hive, Apache HBase, etc.
- You can run workloads on
  - Amazon EC2 instances
  - Amazon Elastic Kubernetes Service (EKS) clusters
  - On-premises using EMR on AWS Outposts

Example of workload

- Upload input data into Amazon S3
- EMR launches EC2 instances that you specified
- EMR begins the execution while pulling the input data from S3 into the launched instances
- Once the cluster is finished, EMR transfers output data to Amazon S3

A tentative organization



Serverless processing

AWS Lambda: compose code functions in a loose orchestration

- Build complex but understandable back-end systems
- Event-driven and push-based pipelines

When using Lambda, you are responsible only for your code

- Lambda manages the compute fleet that offers a balance of memory and CPU
- This is in exchange for flexibility, you cannot log in to compute/customize instances
- Lambda performs operational and administrative activities on your behalf
  - Provisioning capacity, monitoring fleet health, applying security patches, deploying your code

Serverless processing

Amazon AWS Lambda as a case study

- Execute code in a massively parallelized way in response to events
- See also Microsoft Azure Functions, IBM Bluemix, Google Cloud Functions

AWS Lambda

- The Lambda runtime invokes a lambda function multiple times in parallel
- Compute service that executes code written in JavaScript/Python/C#/Java
- Lambda function: code + configuration + dependencies
  - Source code (JARs or DLLs) is zipped up and deployed to a container
  - Every Lambda function is a granular service
- Invocation supports push/pull event models
- Elastic Compute Cloud (EC2) servers run the code
  - E.g., Linux server with distribution Amazon Linux

Serverless processing (FaaS)

Write single-purpose stateless functions

- Keep the single responsibility principle in mind
- A function that does just one thing is more testable and robust
- A function with a well-defined interface is also more likely to be reused
- Code should be created in a stateless style
  - Statelessness allows scalability
  - Local resources or processes will not survive along sessions
- Functions that terminate sooner are cheaper
  - E.g., pricing is based on #requests, execution time, and allocated memory

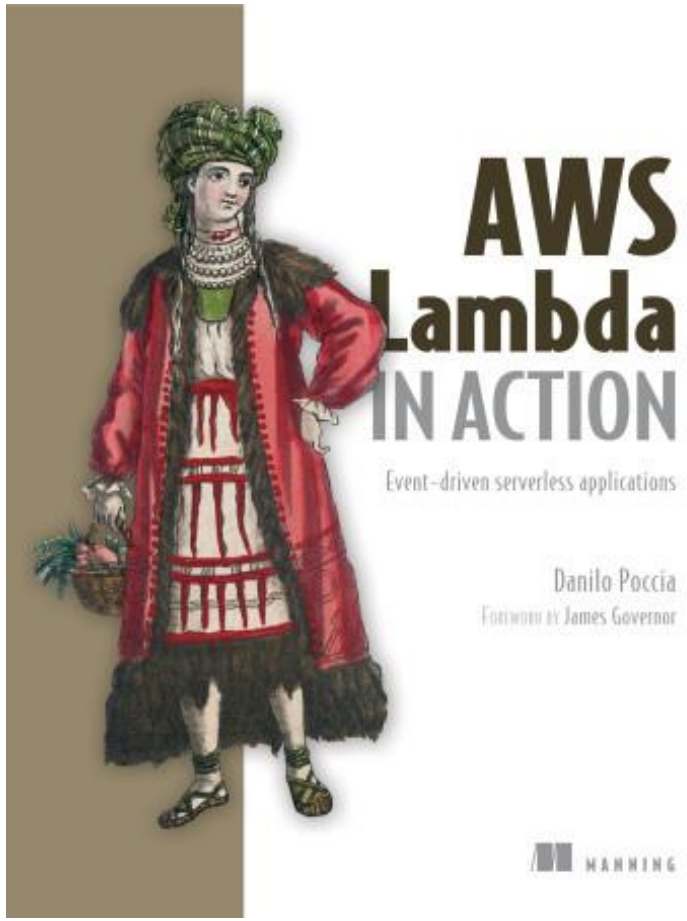
Patterns for data pipelines

Patterns are architectural solutions to problems in software design

- A (design) pattern is a general, best-practice reusable solution to a commonly occurring problem within a given context in software design
- It is a template for how to solve a problem in many different situations

Patterns for serverless data pipelines

- Command pattern
- Messaging pattern
- Priority queue pattern
- Pipes and filters pattern



https://www.manning.com/books/aws-lambda-in-action

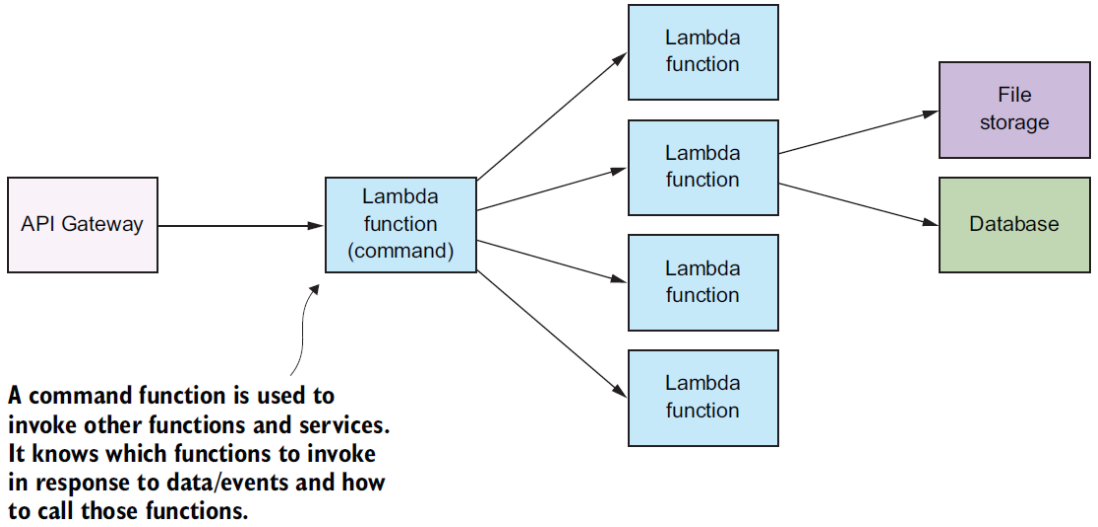
Command pattern

Command pattern

- A behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event

Encapsulate a request as an object

- Issue requests to objects without knowing anything about the operation being requested or the receiver

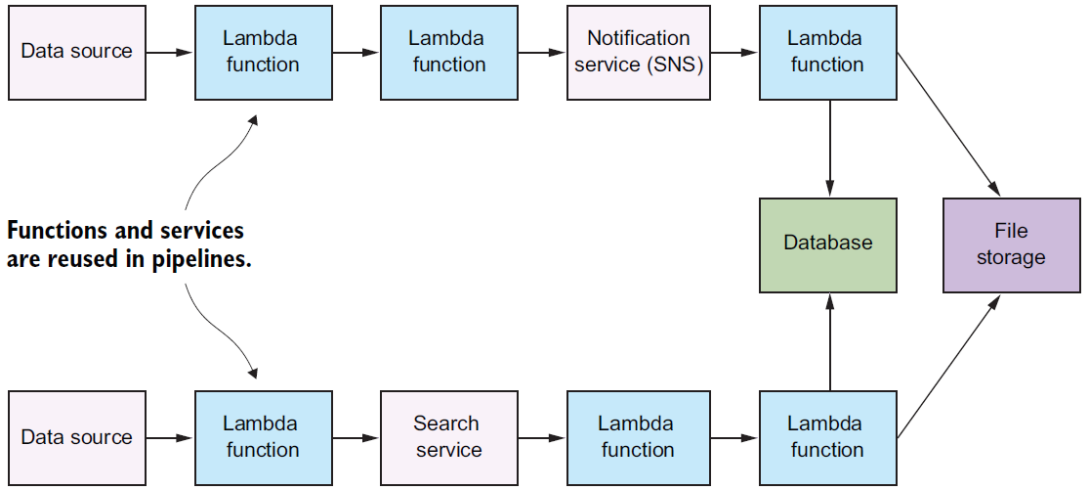


https://aws.amazon.com/api-gateway

Pipes and filters pattern

Decompose a complex processing task into a sequence of manageable services

- Components designed to transform data are referred to as filters
- Connectors that pass data between components are referred to as pipes



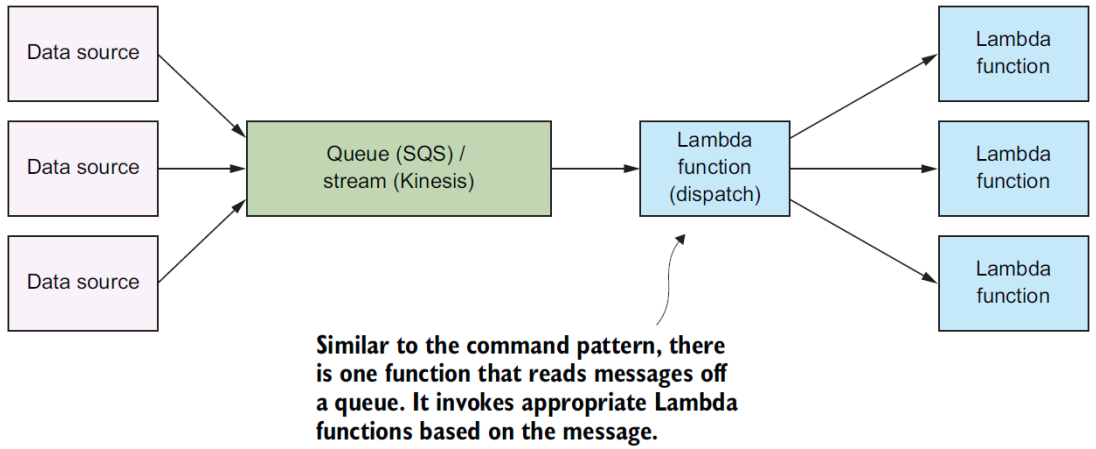
Messaging pattern

Messaging pattern

- Describes how two different parts of a message passing system connect and communicate with each other

Decouple services from direct dependence and allow storage of events in a queue

- Reliability: if the consuming service goes offline, messages are retained in the queue and can still be processed
- A message queue can have a single sender/receiver or multiple senders/receivers



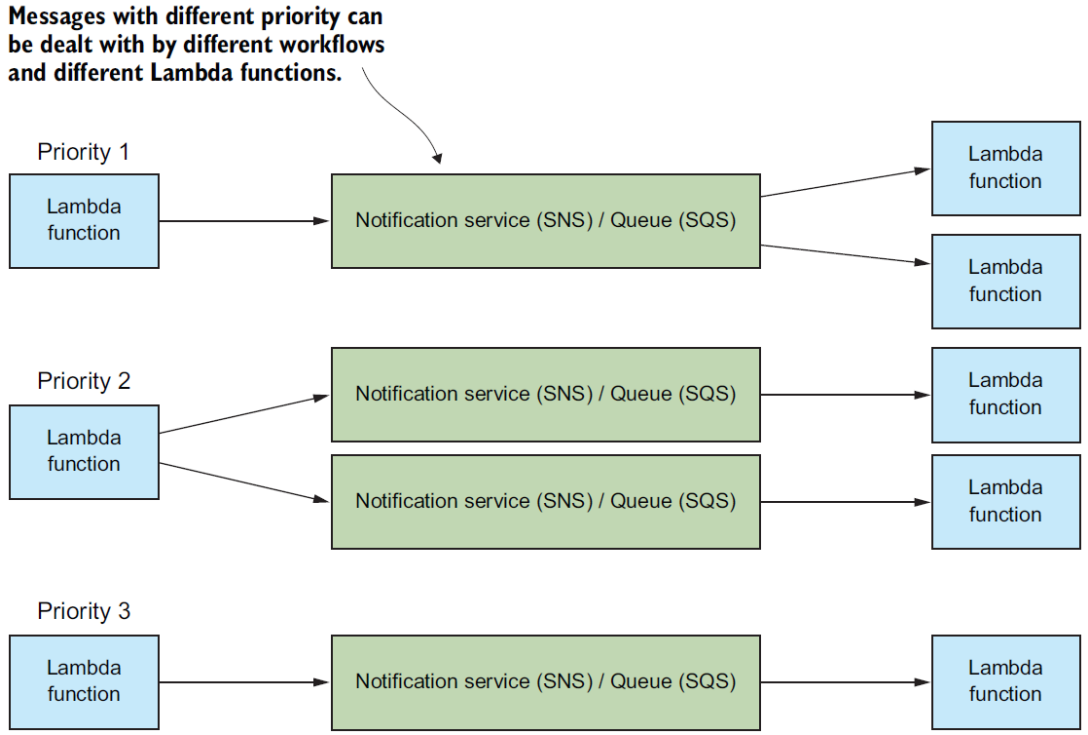
Priority queue pattern

Decouple and prioritize requests sent to services

- Requests with a higher priority are received and processed more quickly than those with a lower priority
- Useful in applications that offer different service level guarantees

Control how and when messages are dealt with

- Different queues, topics, or streams to feed messages to your functions
- High-priority messages go through expensive services with more capacity



BIG DATA

Data pipeline on AWS

AWS services

Amazon Web Services (AWS) is a public-cloud platform of web services

- Computing, storing, and networking, at different abstraction layers
- Web service means services controlled via a (visual) web interface
  - IAM: identity and Access Management
  - EC2: virtual servers
  - S3: (object) storage capacity

Most cloud services can be accessed in multiple ways

- Web GUI: intuitive point and click access without any programming
  - Intuitive interfaces is part of the attraction of cloud services
  - Tedious if the same or similar actions must be performed repeatedly
- (REST) Application programming interface (API)
  - Permits requests to be transmitted via Hypertext Transfer Protocol (HTTPS)
- Software development kits (SDKs) that you install on your computer
  - Access from programming languages such as Python, Java, etc.

Identity and Access Management

Identity and Access Management (IAM)

- Web service that controls fine-grained access to AWS resources
- IAM controls who is authenticated and authorized to use resources

User: unique identity recognized by AWS services and applications

- Individual, system, or application accessing AWS services
- Similar to user in an operating system like Windows or UNIX

Identity and Access Management

After the AWS account creation

- Begin with a single sign-in identity that has complete access to all AWS services
  - I.e., a root user
  - Do not use the root user for your everyday tasks, even the administrative ones
  - Use the root user only to create your first IAM user
- Specify permissions to control which operations a user can perform

What can a user do?

- If permitted, a user has access to all of the resources under the AWS account
- Users can make requests to AWS services using security credentials
- Explicit permissions govern a user's ability to call AWS services
  - Place requests to web services such as Amazon S3 and Amazon EC2

Identity and Access Management

IAM role

- Set of policies for making AWS service requests
- Trusted entities (e.g., such as IAM users) assume roles
  - Not associated with a specific user or group
  - Delegate access with defined permissions to trusted entities without having to share keys
- There is no limit to the number of IAM roles you can assume

Role vs user

- User has permanent long-term credentials and is used to directly interact with AWS services
- Role does not have any credentials and cannot make direct requests to AWS services
- IAM roles are meant to be assumed by authorized entities, such as IAM users, applications, or an AWS service such as EC2

Identity and Access Management

Alice is person (i.e., a IAM user)

- Although she is the same person (user) with or without her turnout gear
- When, as a firewoman, she speeds to a house fire and passes a police officer, he isn't going to give her a ticket
  - In her role as a *firewoman*, she is allowed to speed to the house fire
- When she gets back to the station, off duty, if she speeds past that same police officer, he's going to give her a ticket
  - In her role as a *private citizen*, she is not allowed to speed

Identity and Access Management

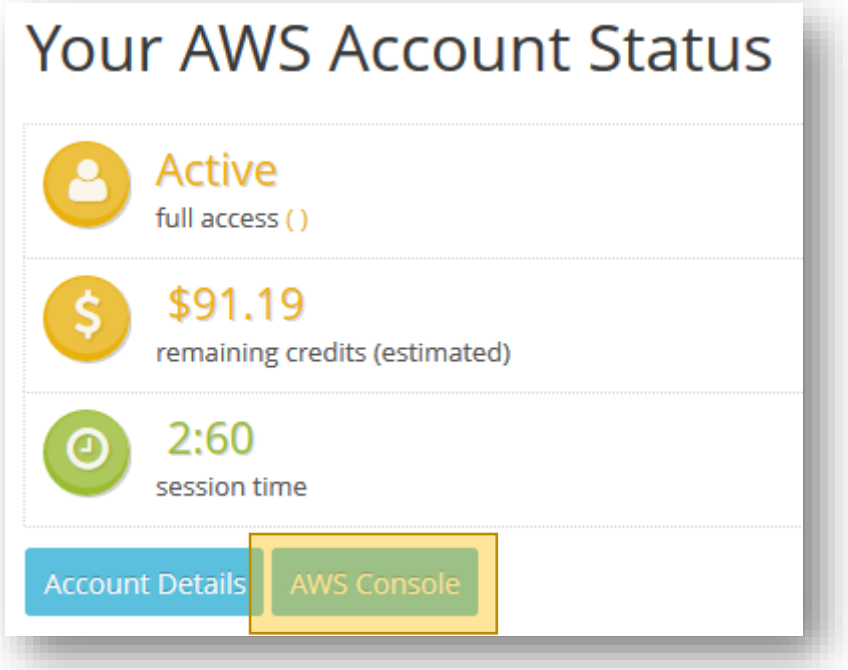
Policy

- An object in AWS that, when associated with an identity or resource, defines their permissions
  - Permissions in the policies determine whether the request is allowed or denied
- AWS evaluates these policies when an IAM principal (user or role) makes a request
- You manage access in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles)
- Six types of policies (listed in order of frequency)
  - Identity-based policies, grant permissions to an identity
  - Resource-based policies, E.g., resource-based policies are Amazon S3 bucket policies
  - Permissions boundaries, maximum permissions that the identity-based policies can grant to an entity
  - Service control policy (SCP), maximum permissions for members of a unit
  - Access control lists (ACLs), which accounts can access the resource to which the ACL is attached
  - Session policies, limit the permissions that the role or identity-based policies grant to the session

AWS Web console

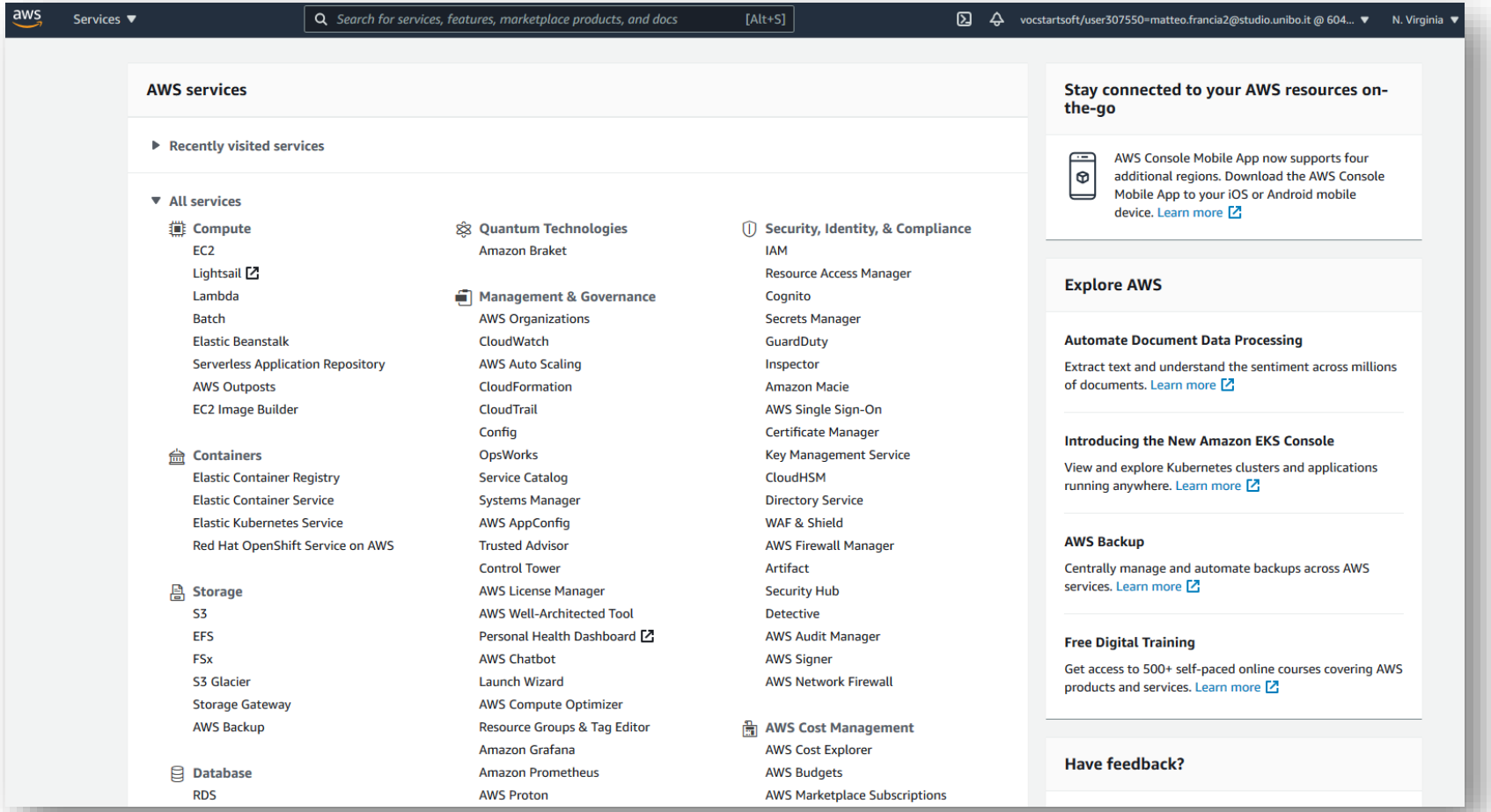
We use the AWS Educate program

- Login with the provided account
- You got 150$ to work on AWS services
- Provisioned services charge even if not used



https://www.awseducate.com/signin/SiteLogin

AWS Web console



AWS CLI

CLI interface

- Necessary to install the CLI (version 2)
- See https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html

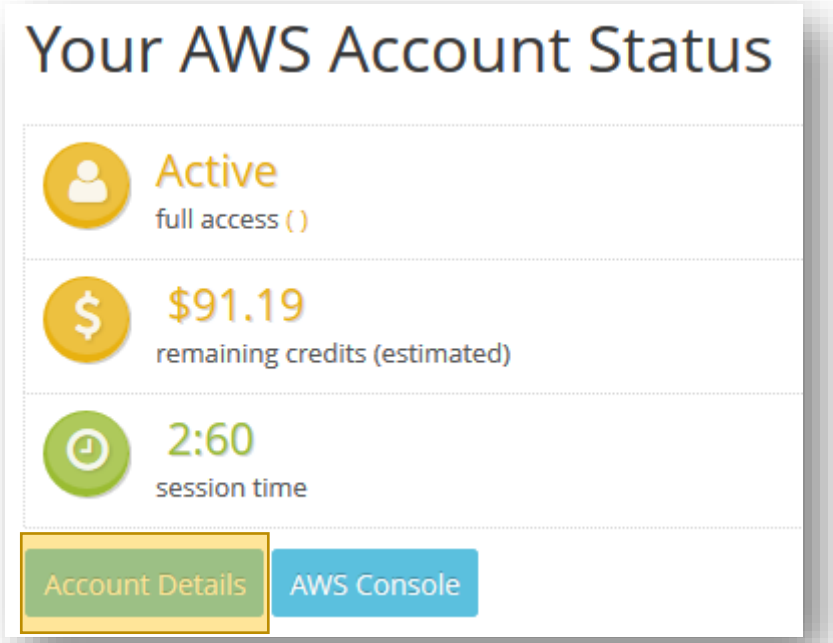
```
Synopsis
********
aws [options] <command> <subcommand> [parameters]


Description
***********
Aunifiedtool to manage your AWS services.
```

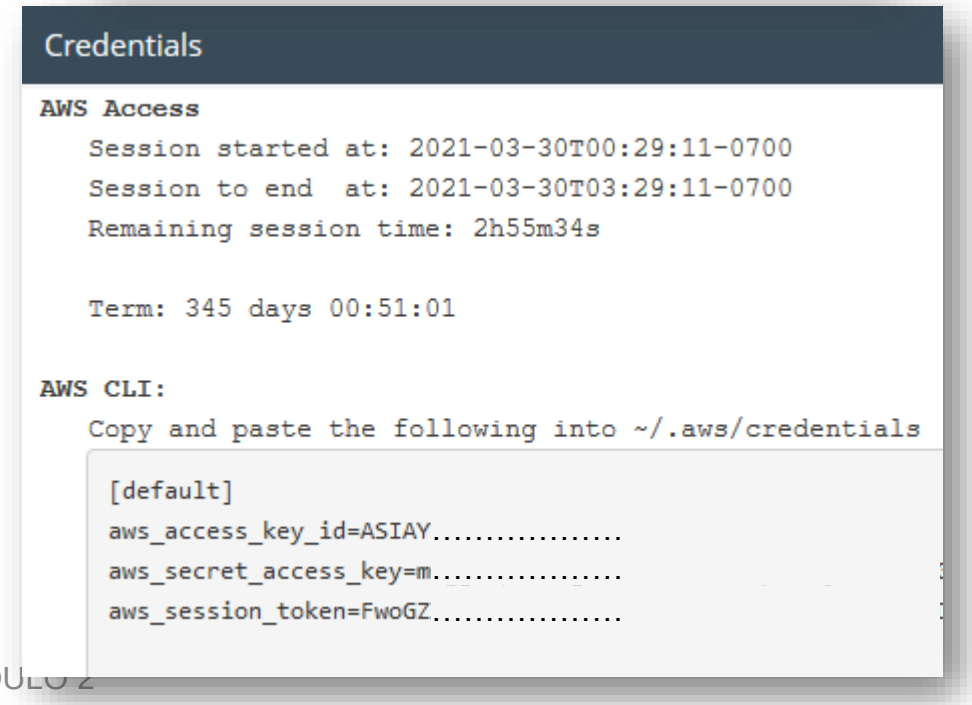
https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html

AWS CLI

CLI needs credentials to work

- Go back to AWS Educate
- Click on “Account Details”
- Copy the content into the file `~/.aws/credentials`
- Later, we assume that you have setup your credentials in the credentials file
- Credentials expire after some time; we need a manually refresh





AWS CLI

Run `aws configure`

- Confirm AWS Access Key ID (press enter)
- Confirm AWS Secret Access Key (press enter)
- Set Default region name to `us-east-1`
- Set Default output format to `json`

It is also possible to configure an AWS profile

- A (named) profile is a collection of settings and credentials
- If profile is specified, its settings and credentials are used to run a command
- When no profile is explicitly referenced, use `default`
  - We stick to `default`

AWS SAM CLI

The AWS Serverless Application Model (AWS SAM) is an open-source framework that you can use to build serverless applications

- A serverless application is a combination of Lambda functions, event sources, and other resources that work together to perform tasks
- Install AWS SAM CLI (on Linux)

```
sudo group add docker
sudo usermod –aG docker $USER
newgrp docker
sudo chmod 666 /var/run/docker.sock
wget https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-linux-x86_64.zip
unzip aws-sam-cli-linux-x86_64.zip -d sam-installation
sudo ./sam-installation/install
sam --version
```

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html

Case study

Given a dataset of sales per customer

find the products frequently bought together

```
Dataset sample
%%%%%%%%%%%%

[ { customerName: Alice, products: [Pizza, Beer, Diaper] },
  { customerName: Bob, products: [Pizza, Beer, Diaper] },
  { customerName: Charlie, products: [Pizza, Cola] } ]
```

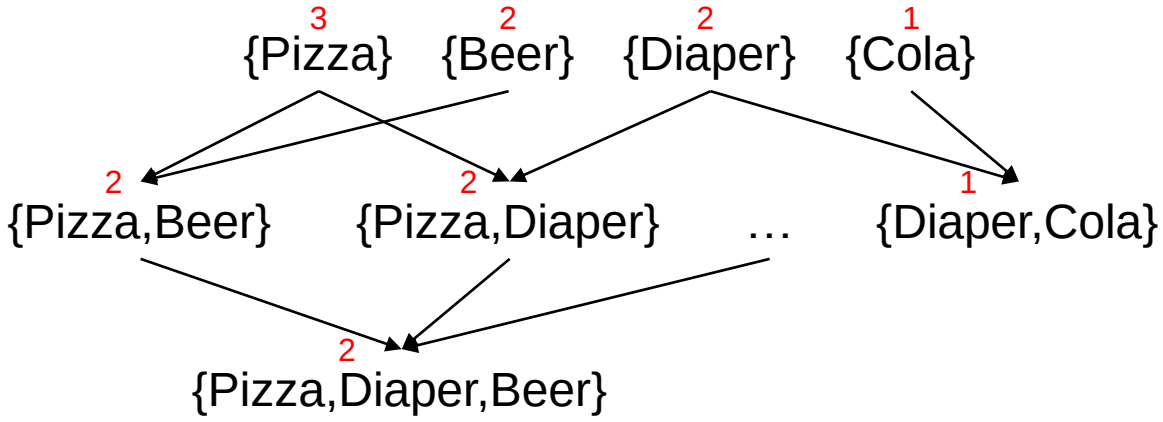
Frequent itemset mining

Find sets of items (i.e., itemsets) frequently appearing together

- **Item**: a product
- **Itemset**: a set of products
- **Frequently**: support above threshold
- **Support**: number of clients buying a set of products

Complexity: $O(2^{|items|})$

```
Dataset sample
%%%%%%%%%%%%%

[ { customerName: Alice, products: [Pizza, Beer, Diaper] },
  { customerName: Bob, products: [Pizza, Beer, Diaper] },
  { customerName: Charlie, products: [Pizza, Cola] } ]
```



Case study

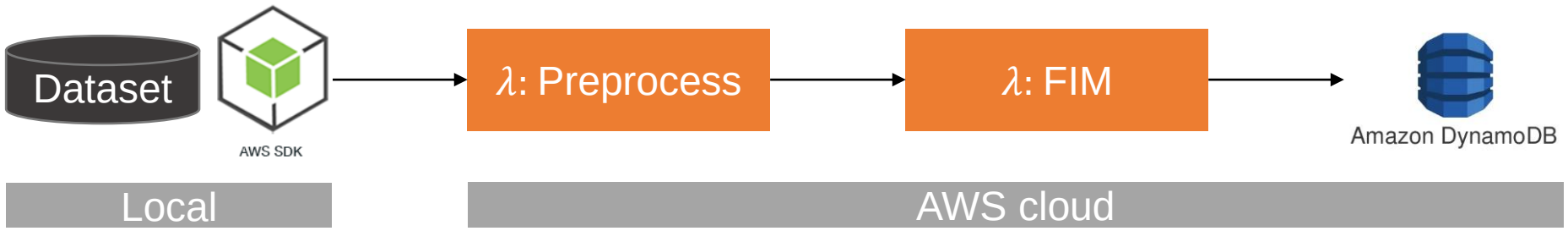
The pipeline involves a single transformation

- FIM

However

- Data is provided as a nested JSON object
- We need a preprocessing step

Reference pipeline



Object storage: S3

Create S3 bucket, the following rules apply for naming buckets

- Must be between 3 and 63 characters long
- Can consist only of lowercase letters, numbers, dots (.), and hyphens (-)
- Must be unique within a partition (i.e., a group of regions)

```
$ git clone https://github.com/w4bo/bigdata-aws/
$ cd bigdata-aws/lab01-lambda
$ aws s3api create-bucket --bucket aws-bucket-bigdata2021
$ aws s3 cp datasets/inferno.txt s3://aws-bucket-bigdata2021/inferno.txt
$ aws s3api list-objects --bucket aws-bucket-bigdata2021
```

https://s3.console.aws.amazon.com/s3/home?region=us-east-1#

NOSQL storage: DynamoDB

Basic DynamoDB components: tables and items

Tables, collection of (data) items

Items, a group of attributes that is uniquely identifiable

- Each table contains zero or more items
- No limit to the number of items you can store in a table
- Each item in the table has a unique identifier, or primary key
- E.g., in the table `people`, each item represents a `person`
  - The primary key consists of one attribute (`fiscalCode`)

NOSQL storage: DynamoDB

Attributes

- A fundamental data element that is not broken down any further
  - E.g., an item in the `people` table contains attributes `fiscalCode` and `lastName`
- Most of the attributes are scalar (have only one value)
- Some of the items have a nested attribute (`address`) up to 32 levels deep

Schemaless

- Other than the primary key, a table is schemaless
  - Neither the attributes nor their data types need to be defined beforehand
  - Each item can have its own distinct attributes

NOSQL storage: DynamoDB

Primary Key

- To create a table, you must specify the primary key of the table
- No two items can have the same key

Two types of primary keys

- Partition key: a simple primary key composed of one attribute (partition key)
  - Keys are inputs to an internal hash function
  - The hash function determines the physical partition in which the item will be stored
  - E.g., access any item in the `people` table directly by providing the `fiscalCode`
- Composite primary key: partition key and sort key (two attributes)
  - First attribute is the partition key
  - Second attribute is the sort key
  - Items in same partition key value are stored together and sorted by sort key

NOSQL storage: DynamoDB

Secondary Indexes

- One or more secondary indexes per table
- Indexes are automatically maintained on add, update, or delete
- Query data using an alternate key (additionally to queries against primary key)

Two types of indexes

- Global secondary has partition and sort keys different from those on table
- Local secondary has the same partition key but a different sort key
- Each table has a limited quota of 20 global and 5 local indexes

How do we shape the schema?

- https://cloud.google.com/bigtable/docs/schema-design

NOSQL storage: DynamoDB

Create a table `frequent-sales` with a composite key

- `dataset`: String
- `timestamp`: String

```
$ aws dynamodb create-table \
    --table-name frequent-sales \
    --attribute-definitions AttributeName=dataset,AttributeType=S AttributeName=timestamp,AttributeType=S \
    --key-schema AttributeName=dataset,KeyType=HASH AttributeName=timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1


$ aws dynamodb delete-table --table-name frequent-sales


$ aws dynamodb list-tables
```

NOSQL storage: DynamoDB

When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write

Eventually Consistent Reads (default)

- Response might include stale data. After short time, the response should return the latest data

Strongly Consistent Reads

- DynamoDB returns a response with the most up-to-date data
- A strongly consistent read might not be available if there is a network delay or outage
  - In this case, DynamoDB may return a server error (HTTP 500)
- Strongly consistent reads may have higher latency than eventually consistent reads
- Strongly consistent reads are not supported on global secondary indexes

NOSQL storage: DynamoDB

Provisioned mode: specify the #reads and #writes per second is good if

- You have predictable application traffic or traffic ramps gradually
- You can forecast capacity requirements to control costs

One read capacity unit

- One strongly consistent read per second
- Or two eventually consistent reads per second
- RCUs also depend on the item size (a read is up to 4 KB in size), if item size is 8 KB
  - 2 RCUs to sustain one strongly consistent read per second
  - 1 RCU if you choose eventually consistent reads

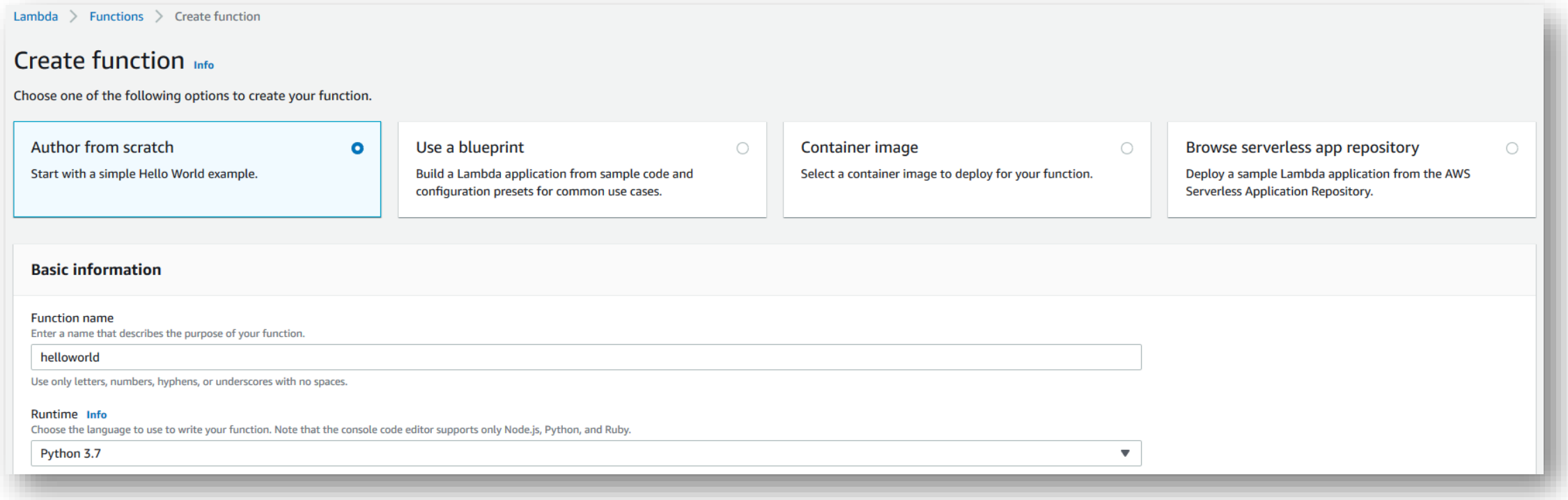
One write capacity unit represents one write per second for an item up to 1 KB in size

NOSQL storage: DynamoDB

Put a new item and get it back

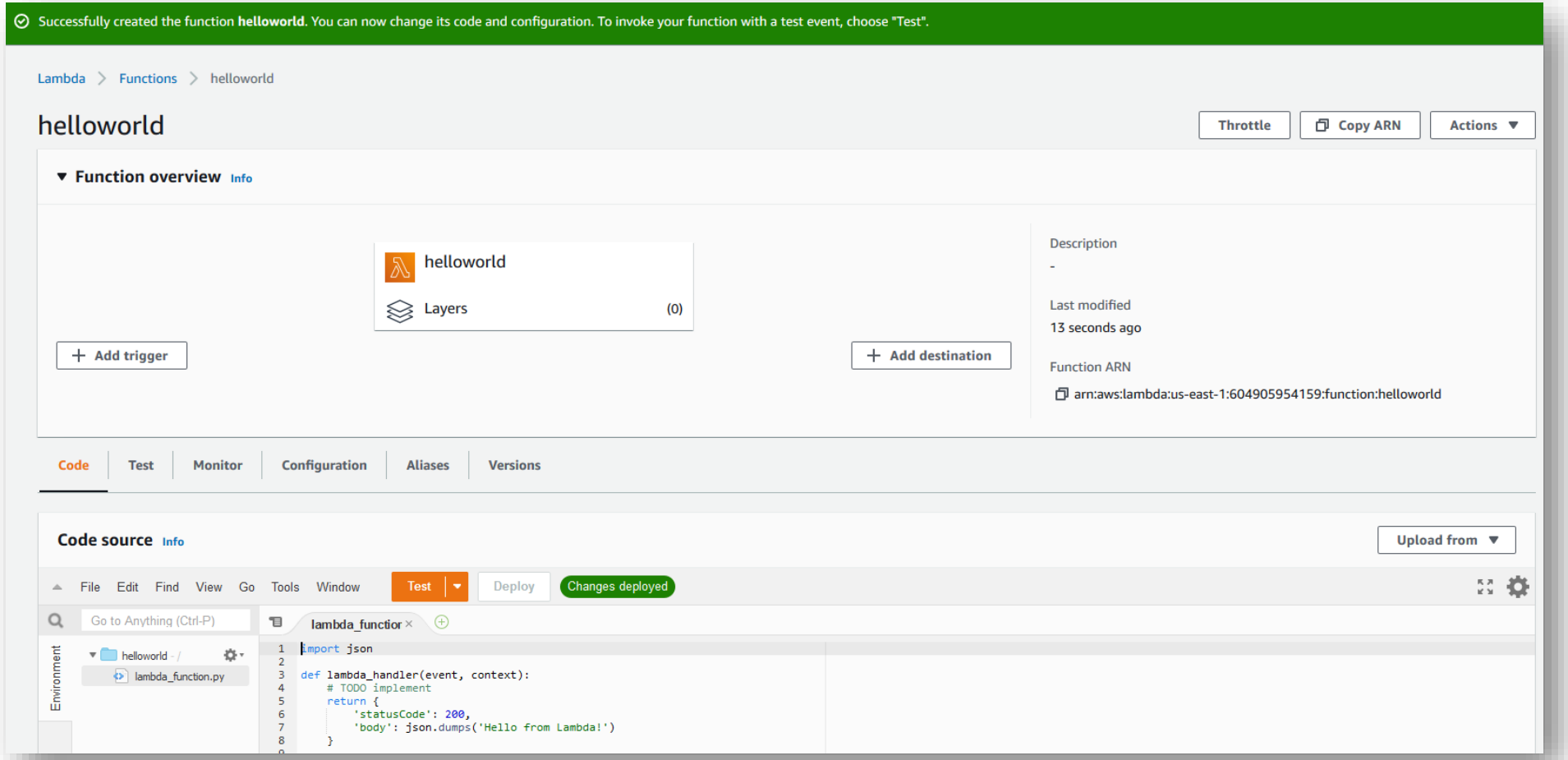
```
$ aws dynamodb put-item
    --table-name frequent-sales
    --item '{"dataset": {"S": "sales"}, "timestamp": {"S": "1611226870"}, "bar": {"S": "foobar"}}'


$ aws dynamodb query
    --table-name frequent-sales
    --key-condition-expression " dataset = :n"
    --expression-attribute-values '{":n":{"S":"sales"}}'
```

Lambda: create a function



https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions

Lambda: create a function



https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions

Lambda: create a function

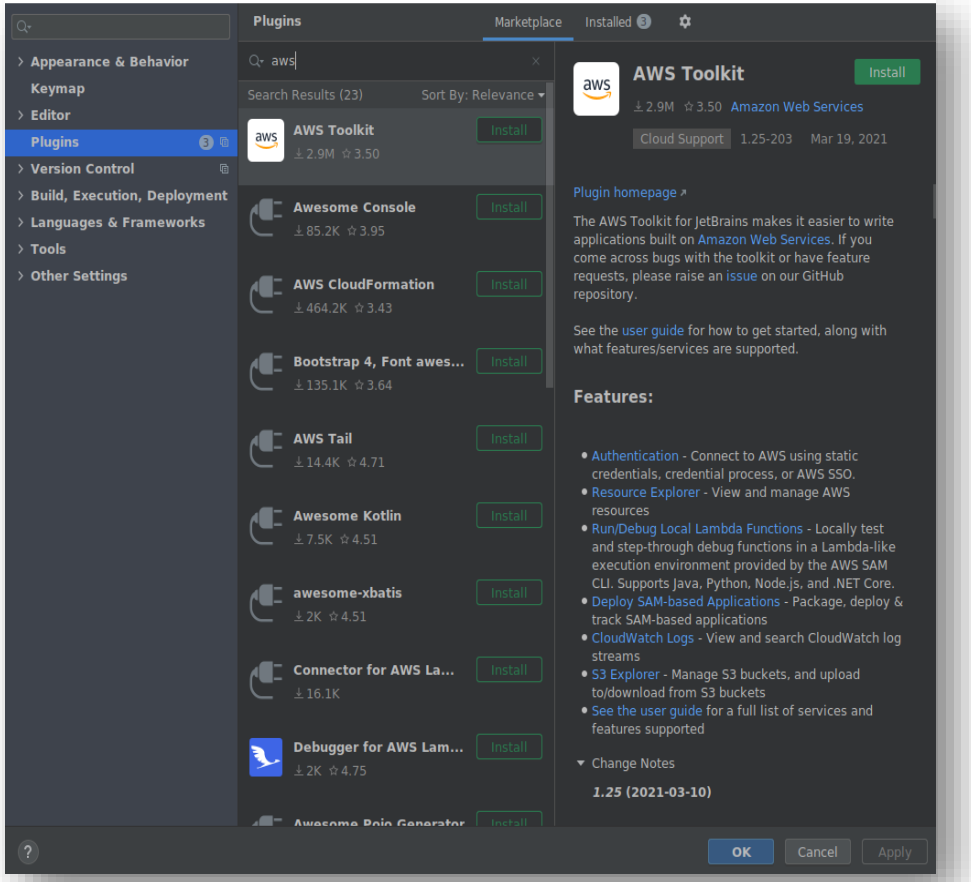
Manually creating the functions is cumbersome

- We must copy and paste code
- No automatic testing
- No debugging
- No IDE support (and not all languages are supported)

Switch to IntelliJ IDEA + AWS Toolkit

AWS Toolkit

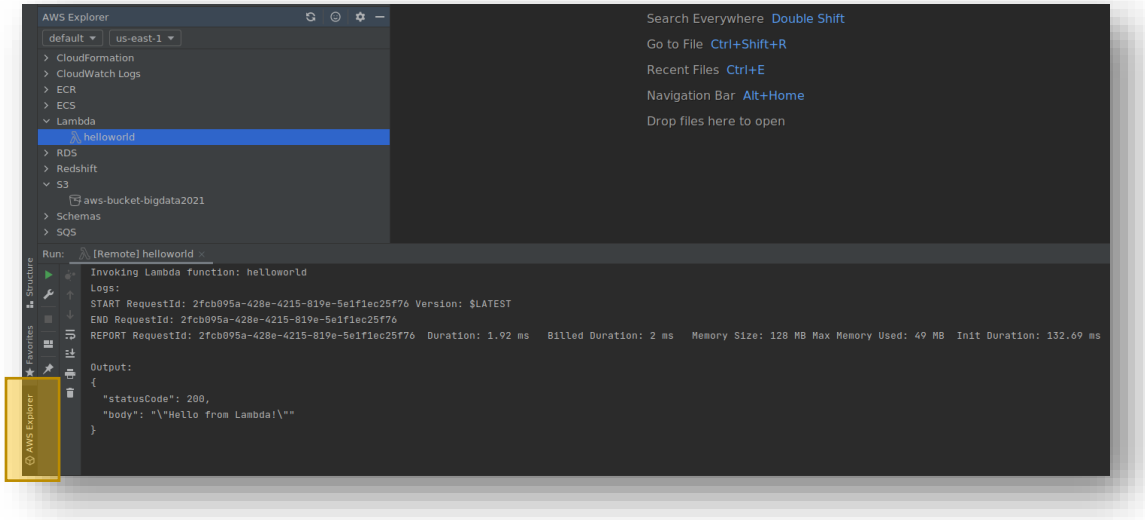
- Get the latest IntelliJ IDEA
- Install the `AWS Toolkit`
- Copy the credentials
  `cp ~/.aws/credentials ~/.aws/config`
- Clone the repo
  `git clone https://github.com/w4bo/bigdata-aws/`
- Import `lab01-lamba` as a Gradle project
- Verify that the project builds
  `./gradlew`



AWS Toolkit

Click on `AWS Explorer`

- You can see the `helloworld` function
- Plus `CloudWatch Logs` and `S3`



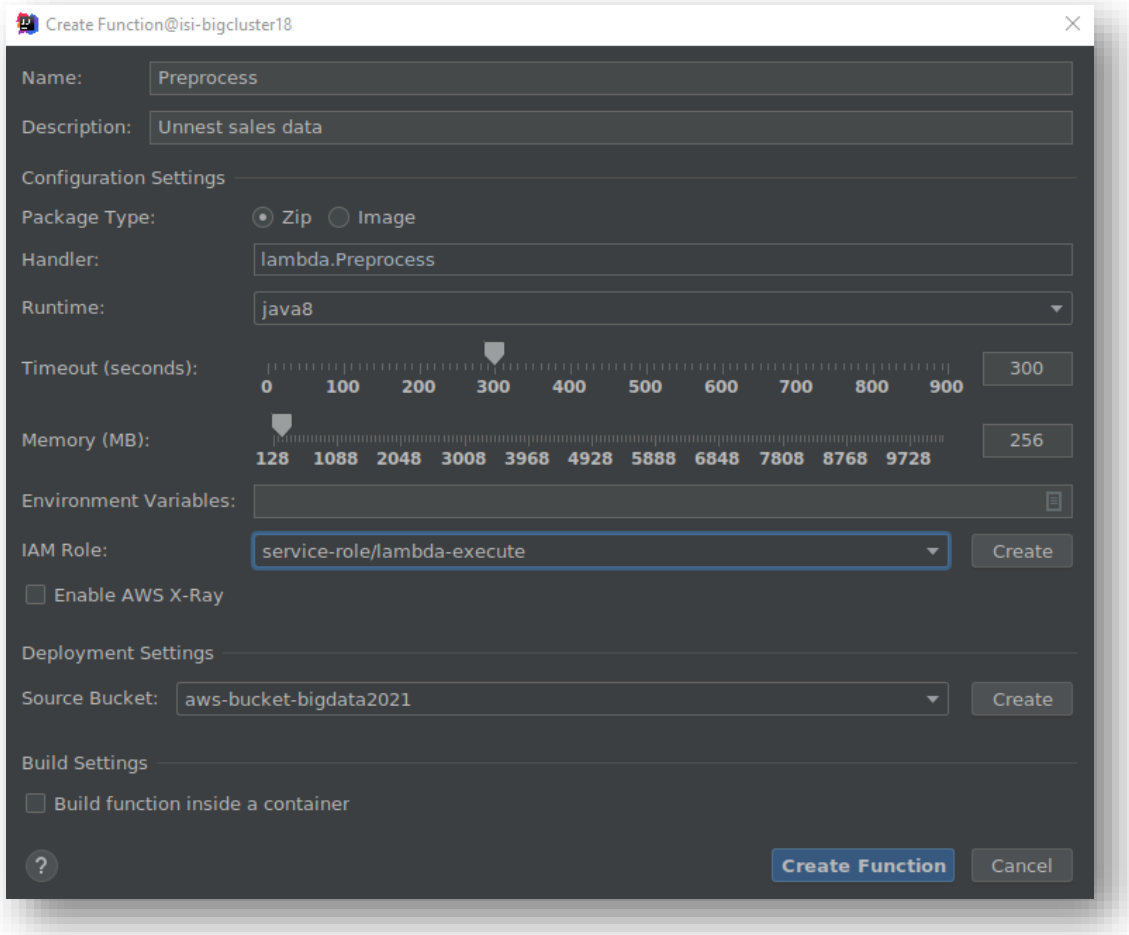
AWS Toolkit

Test the existing code locally

- With Gradle
- Or with local Lambda execution

Deploy a new Lambda function from the existing code

- Right click on AWS Explorer > Lambda
- Select `Create new AWS Lambda…`
- Populate the settings
- `Create the function`

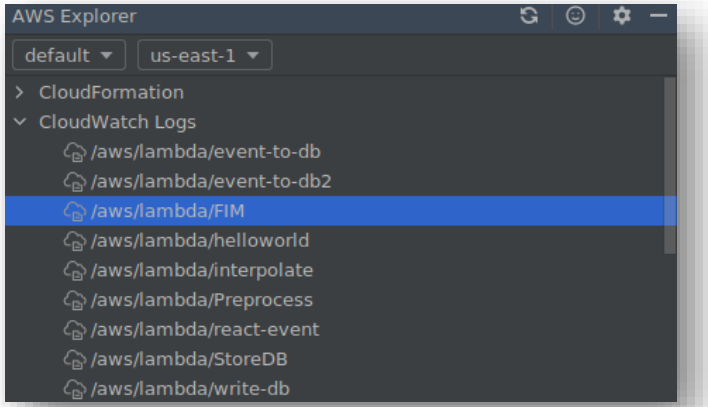


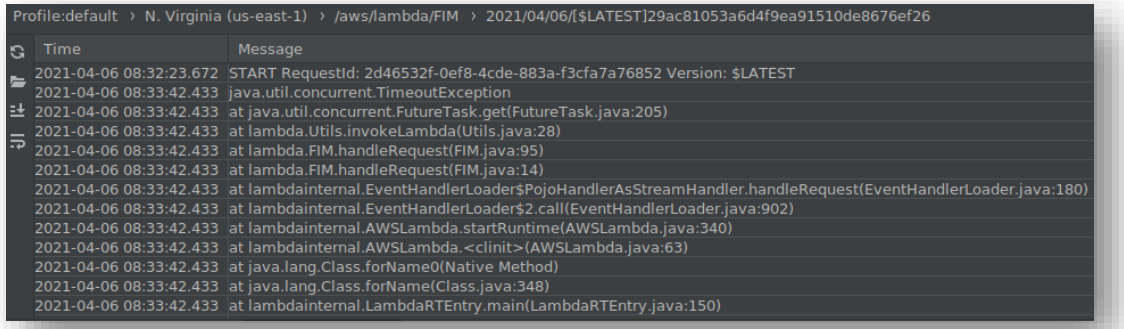
https://aws.amazon.com/lambda/pricing/

AWS Toolkit

Check the log for errors and pricing

- AWS Toolkit > CloudWatch Logs
- Double click on the function name
- Double click on the log entry





Data pipeline

Deploy and execute the HelloWorld.java lambda function

Given the created storage: S3 and DynamoDB

- Deploy the function `FIM`
- Deploy the function `Preprocess`
- Run ReadDataset.java
- Check that the table `frequent-sales` has the FIs for the dataset `sales`

Some hints

- Function names are case sensitive
- Some function need more than 128MB of memory
  - Behold! The higher the RAM, the higher the costs

AWS services

AWS Educate (and AWS console)

- https://aws.amazon.com/it/education/awseducate/
- https://console.aws.amazon.com/console/home?region=us-east-1

IAM (authentication)

- https://docs.aws.amazon.com/IAM/latest/UserGuide/iam-ug.pdf

SDK (software API)

- https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home.html

Lambda (serverless computing and processing)

- https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html
- https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions

DynamoDB (key-value database)

- https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction

S3 (object storage)

- https://s3.console.aws.amazon.com/s3/home?region=us-east-1

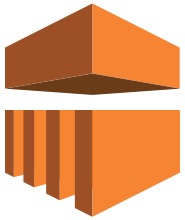
BIG DATA

Migration - Amazon EMR

Motivation

Amazon EMR (Elastic Map Reduce)

- Provides a managed Hadoop framework
- Quickly and cost-effectively process vast amounts of data
- Run other popular distributed frameworks such as Spark



Some features

- Low Hourly Pricing
- Amazon EC2 Spot Integration
- Amazon EC2 Reserved Instance Integration Elasticity
- Amazon S3 Integration

https://aws.amazon.com/emr

EMR Cluster

The central component of Amazon EMR is the **cluster**

- A collection of **Amazon Elastic Compute Cloud (Amazon EC2)** instances
- Each instance is called a **node**

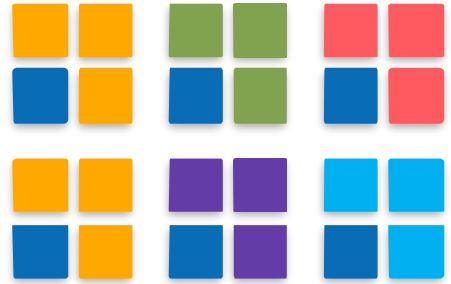
The **node type** identifies the role within the cluster

- **Master** node coordinates the distribution of data and tasks among other nodes
  - Tracks the status of tasks and monitors the health of the cluster
  - Every cluster has (at least) a master node
    - It is possible to create a single-node cluster with only the master node
- **Core** node runs tasks and store data in the Hadoop Distributed File System (HDFS)
  - Multi-node clusters have at least one core node
- **Task** node only runs tasks and does not store data in HDFS
  - Task nodes are optional

EMR Cluster

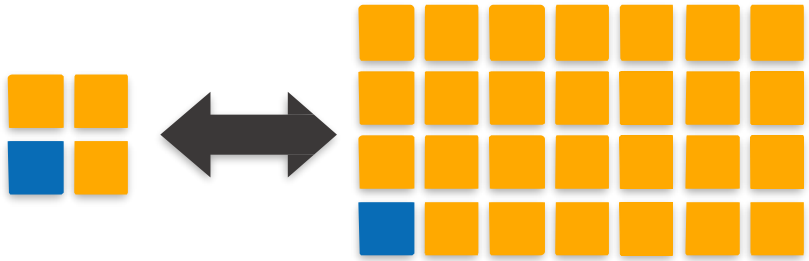
Provision as much capacity as you need

Deploy Multiple Clusters



Add or remove capacity at any time

Resize a Running Cluster



EMR Cluster

Start an EMR cluster

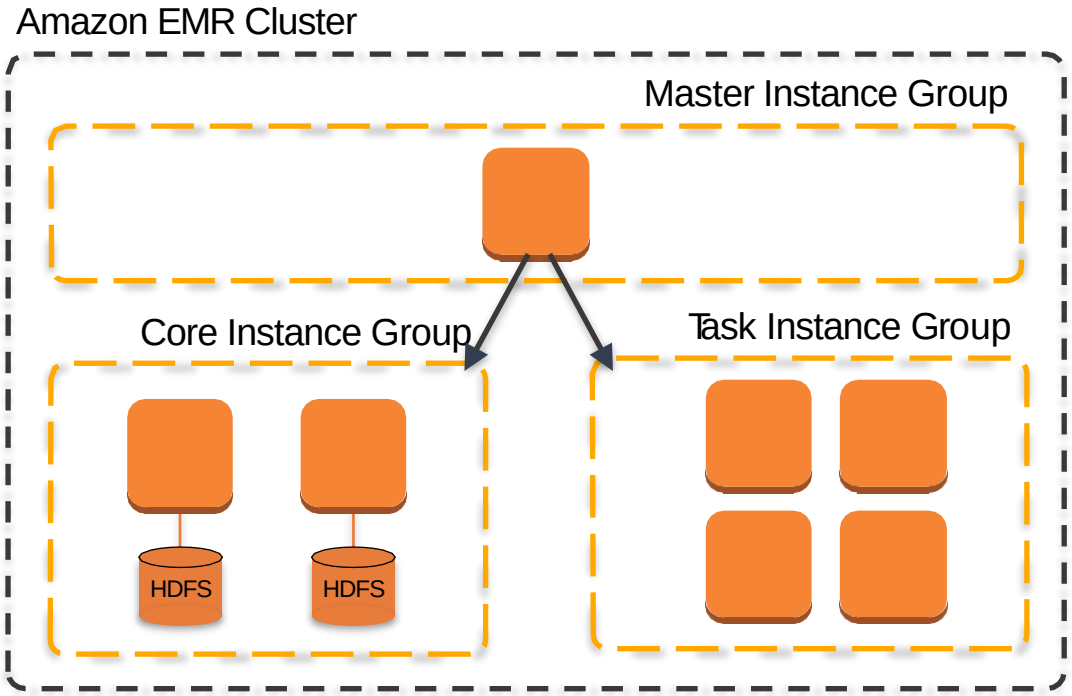
Master group controls the cluster

- Master node coordinates distribution of work and manages cluster state

Core group created for life of cluster

- Core instances run **DataNode** and **TaskTracker** daemons

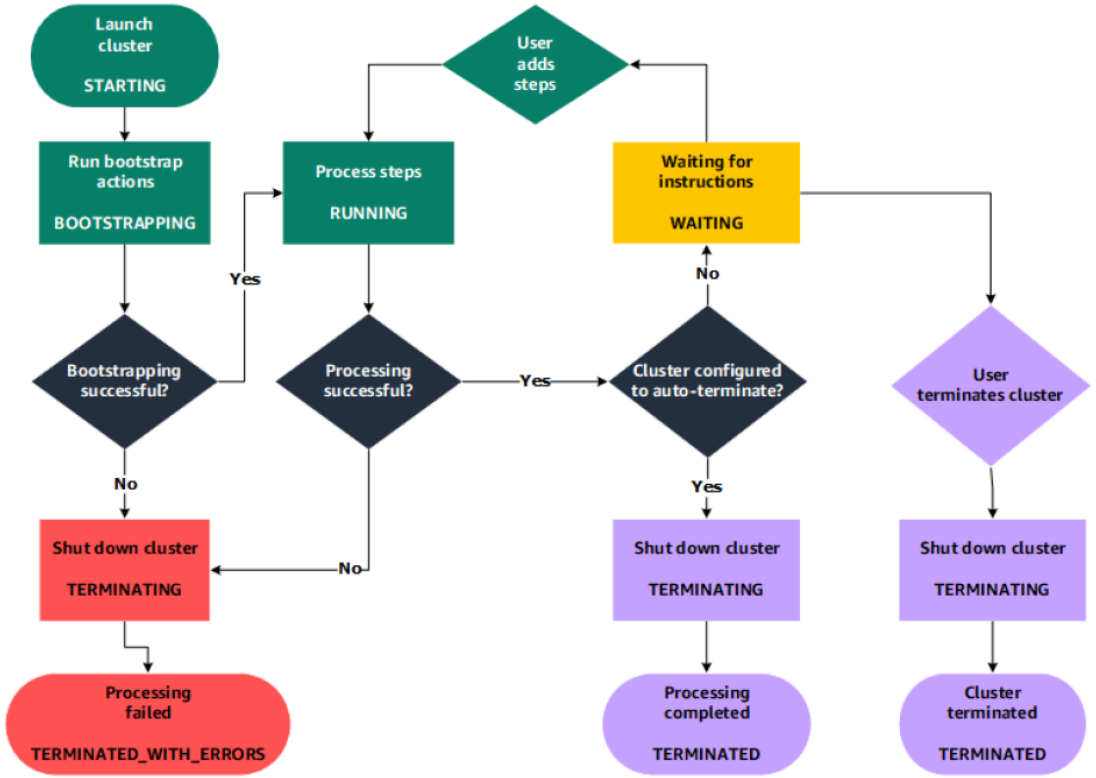
Optional task instances can be added or subtracted



Cluster lifecycle

Creating a cluster (it takes ~10 minutes)

- A cluster cannot be stopped
- It can only be terminated



Cluster lifecycle

`STARTING`: EMR provisions EC2 instances for each required instance

`BOOTSTRAPPING`: EMR runs actions that you specify on each instance

- E.g., install custom applications and perform customizations

Amazon EMR installs the native applications

- E.g., Hive, Hadoop, Spark, and so on.

`RUNNING`: a step for the cluster is currently being run

- Cluster sequentially runs any steps that you specified when you created the cluster

`WAITING`: after steps run successfully

`TERMINATING`: After you manually shut down the cluster

- If a cluster terminates because of a failure, any data stored on the cluster is deleted

Cluster: EMR

A **step** is a user-defined unit of processing

- E.g., one algorithm that manipulates the data

Step states

- `PENDING:` The step is waiting to be run
- `RUNNING:` The step is currently running
- `COMPLETED:` The step completed successfully
- `CANCELLED:` The step was cancelled before running because an earlier step failed
- `FAILED:` The step failed while running

Hands on

Choose the frameworks and applications to install

To process data in your Amazon EMR cluster

- Submit jobs or queries directly to installed applications
- Run steps in the cluster

Directly submit jobs

- Connect to the master node over a secure connection
- Access the interfaces and tools that are available on your cluster

EC2

AWS uses public-key cryptography to secure the login

You can create one using the Amazon EC2 console

- Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/
- In the navigation pane, choose `Key Pairs`
- Choose `Create key pair`
- For `Name`, enter a descriptive name for the key pair
- For `File format`, choose the format in which to save the private key
  - OpenSSH, choose `pem` (`chmod 400 *my-key-pair*.pem`)
  - PuTTY, choose `ppk`
- Choose `Create key pair`
- The private key file is automatically downloaded by your browser

Creating the cluster

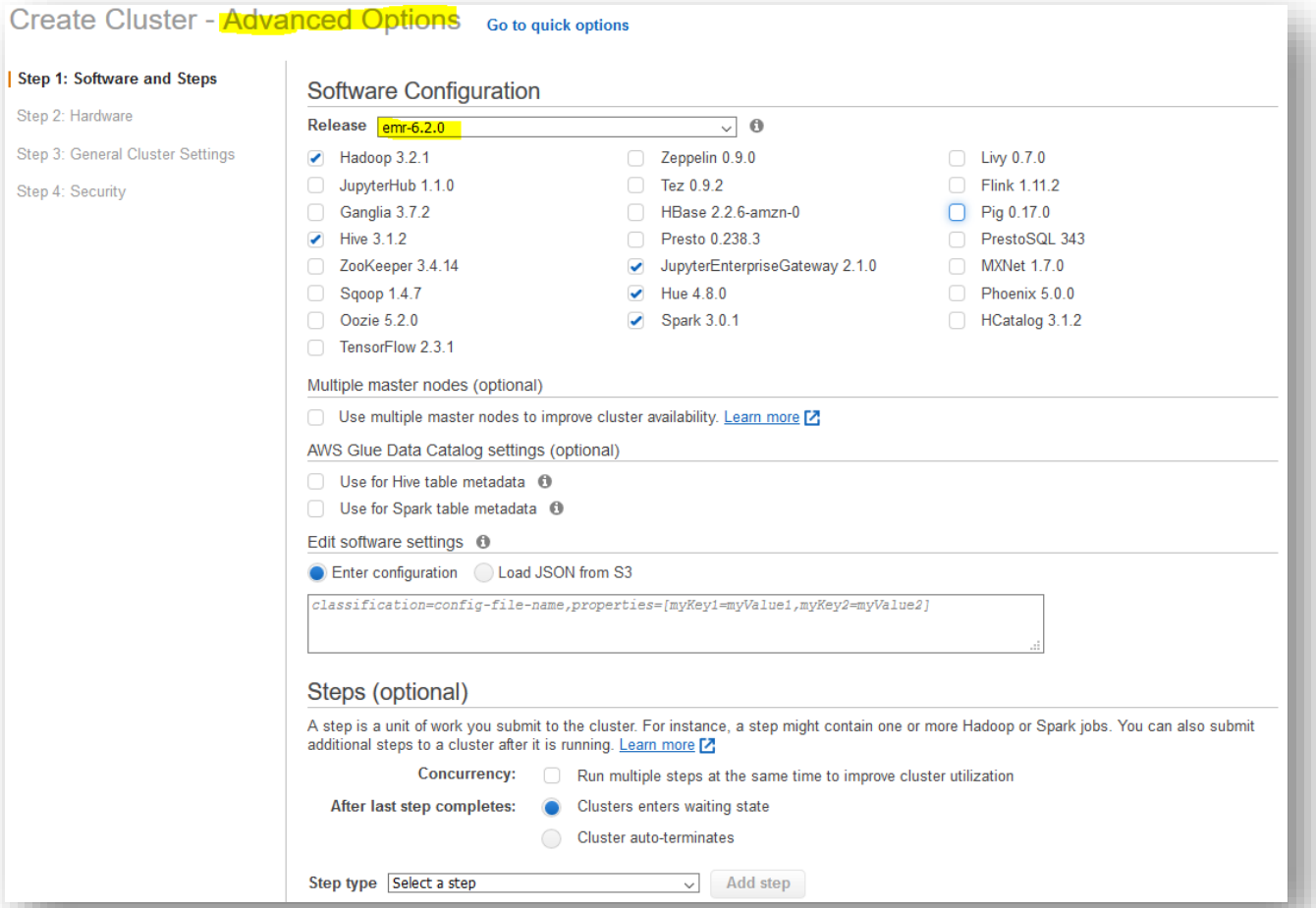
Using CLI (command line interface)

```
aws emr create-cluster \
    --name "My First EMR Cluster" \
    --release-label emr-5.32.0 \
    --applications Name=Spark \
    --ec2-attributes KeyName=myEMRKeyPairName \
    --instance-type m5.xlarge \
    --instance-count 3 \
    --use-default-roles
```

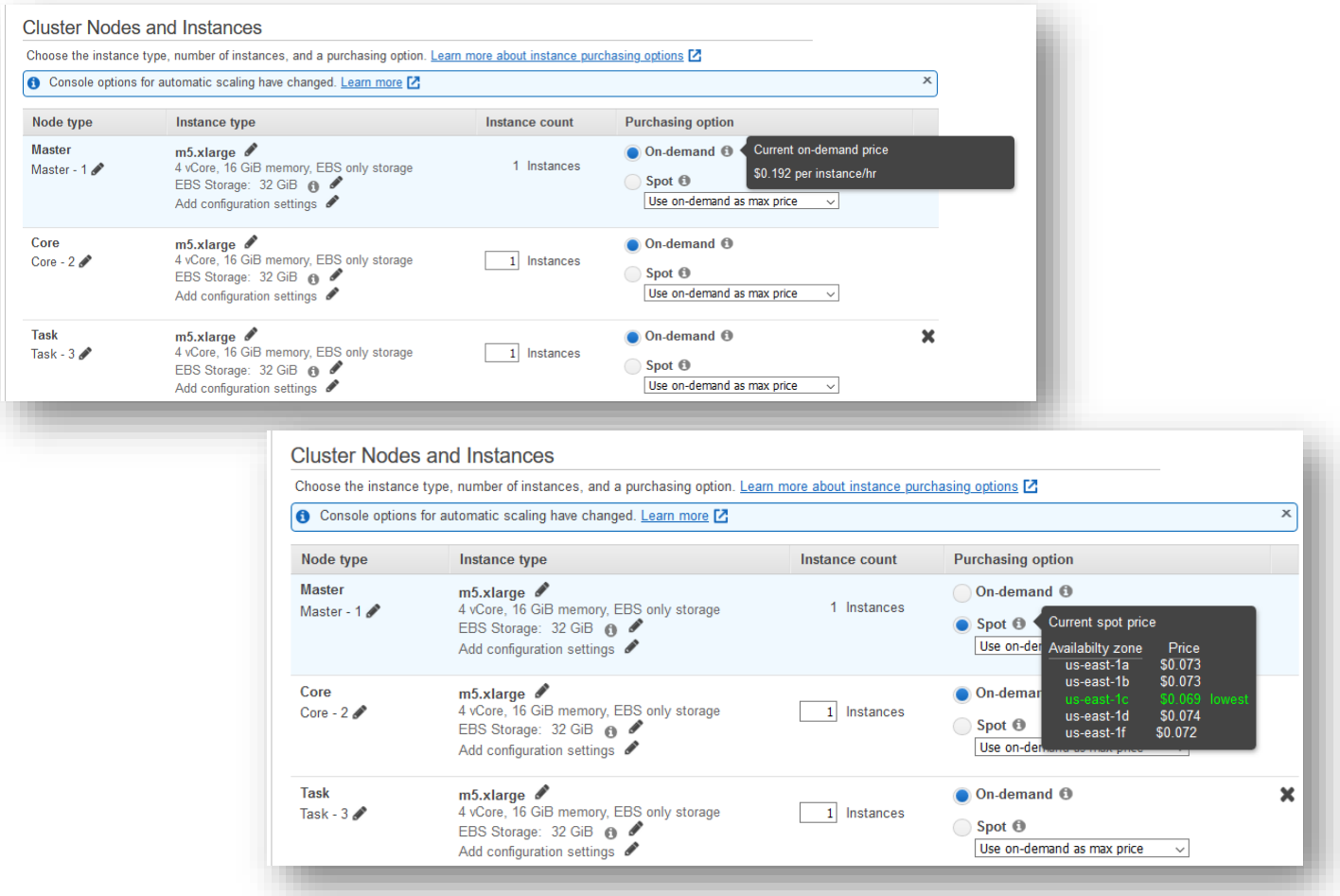
This is more pragmatic, but there are many options to explore

- Let's stick to AWS Console
- https://console.aws.amazon.com/elasticmapreduce/

Creating the cluster



Creating the cluster



EMR

In EMR you can choose to launch **master**, **core**, or **task** on Spot Instances

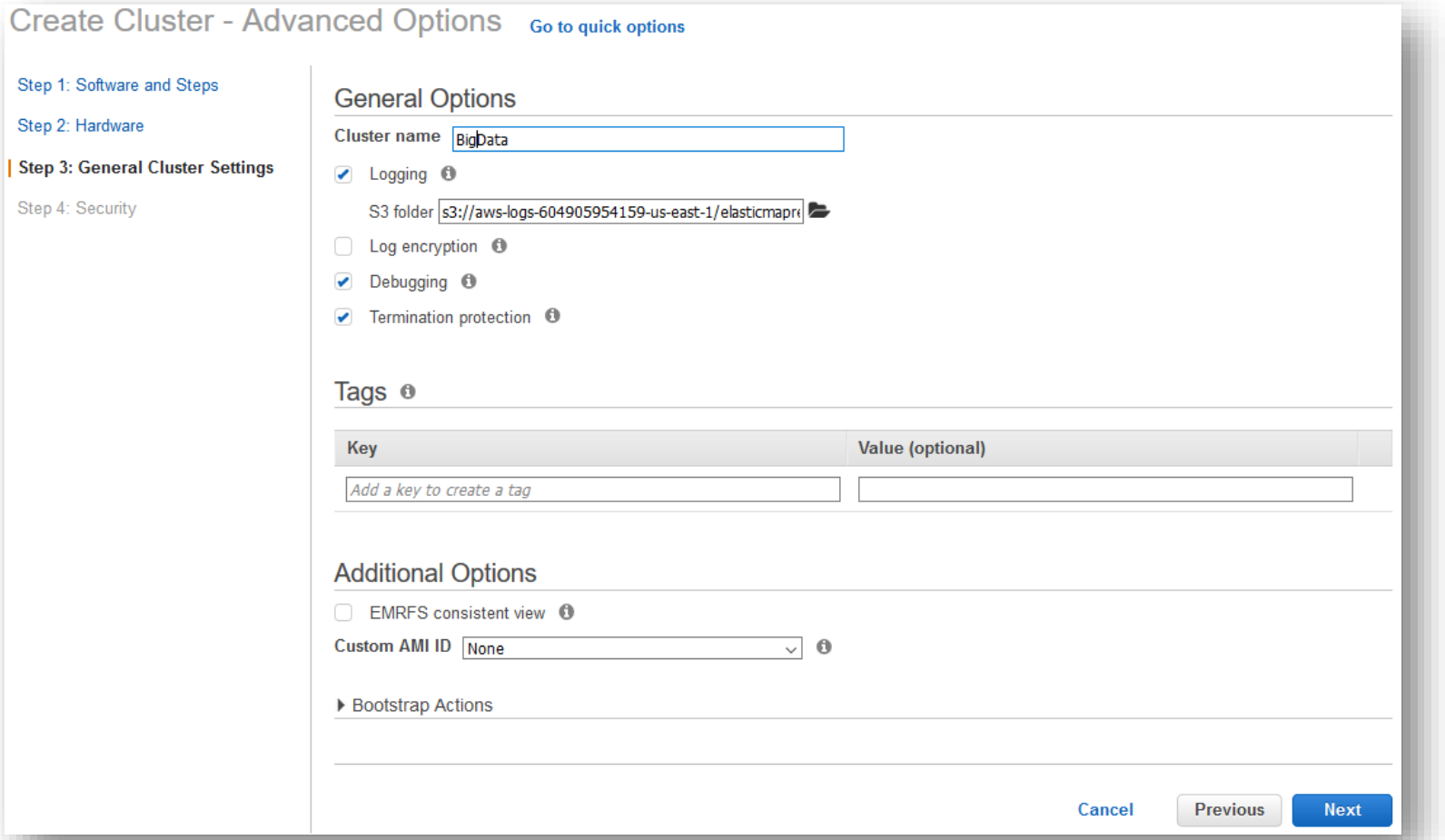
- The **master** node controls and directs the cluster. When it terminates, the cluster ends, so you should only launch the master node as a Spot Instance if you are running a cluster where sudden termination is acceptable.
- Core nodes process data and store information using HDFS. Terminating a core instance risks data loss. For this reason, you should only run core nodes on Spot Instances when partial HDFS data loss is tolerable.
- The task nodes process data but do not hold persistent data in HDFS. If they terminate because the Spot price has risen above your maximum Spot price, no data is lost and the effect on your cluster is minimal

https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-instances-guidelines.html

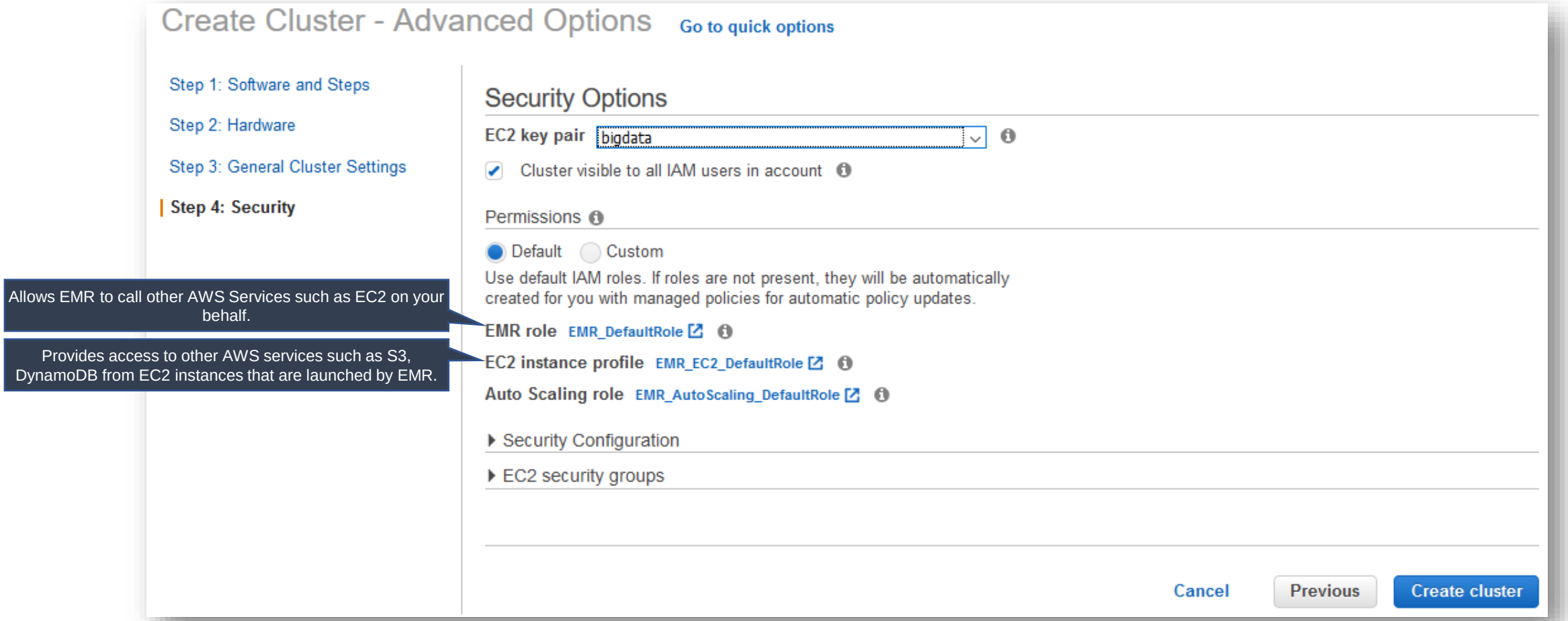
EMR

Application Scenario	Master Node Purchasing Option	Core Nodes Purchasing Option	Task Nodes Purchasing Option
Long-Running Clusters and Data Warehouses	On-Demand	On-Demand or instance-fleet mix	Spot or instance-fleet mix
Cost-Driven Workloads	Spot	Spot	Spot
Data-Critical Workloads	On-Demand	On-Demand	Spot or instance-fleet mix
Application Testing	Spot	Spot	Spot

Creating the cluster



Creating the cluster

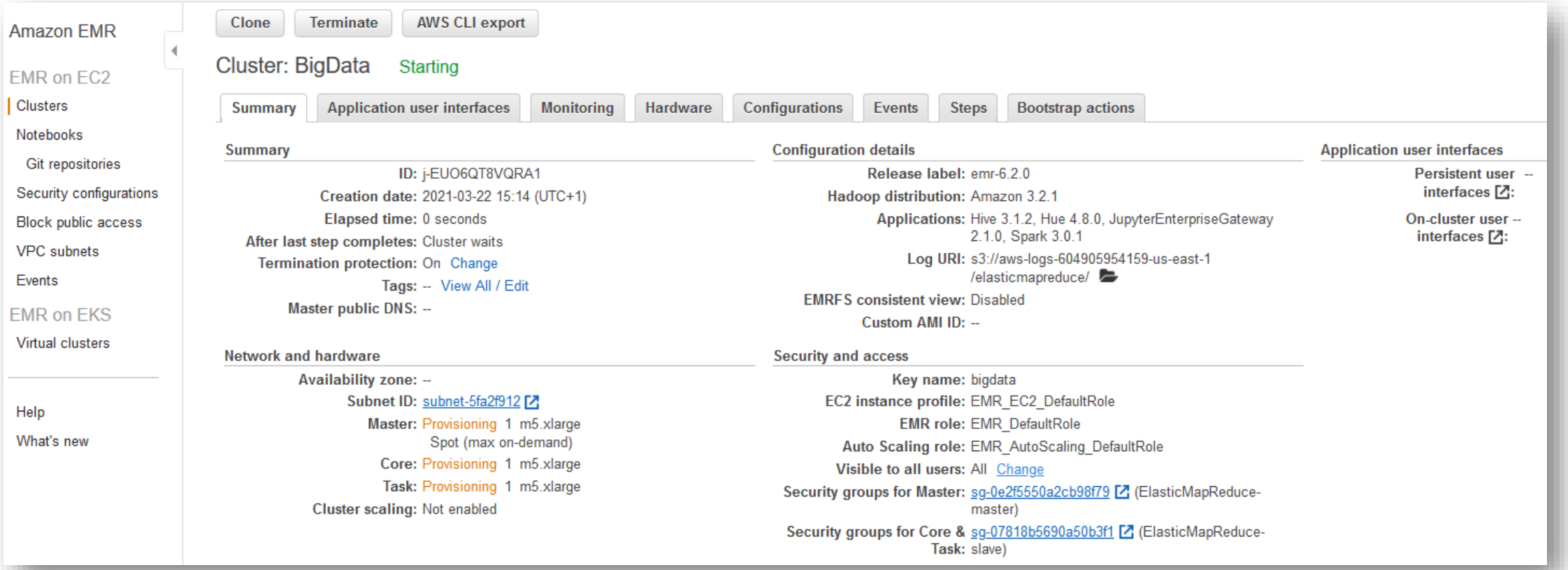


Creating the cluster

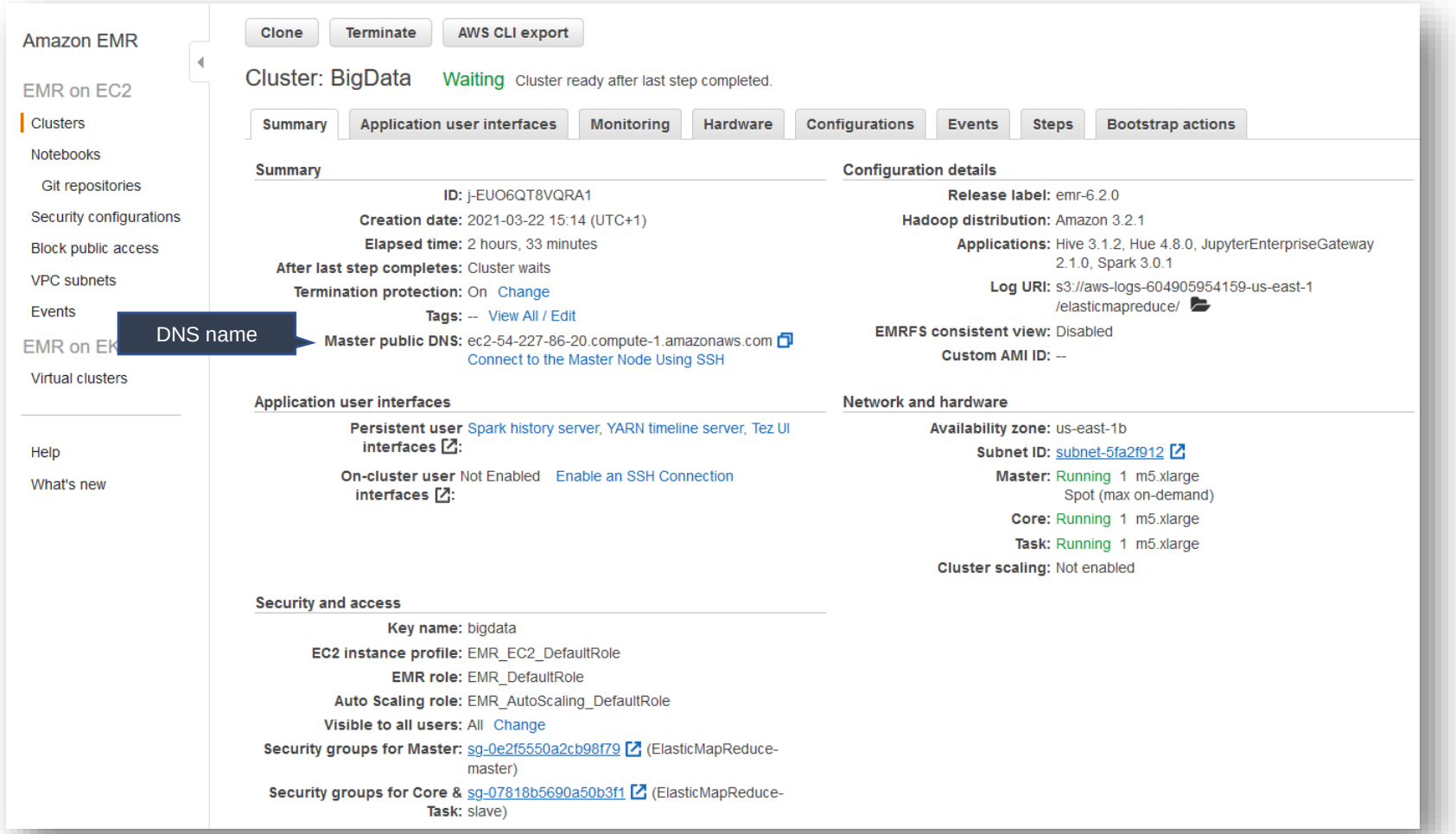
Using CLI (command line interface)

```
aws emr create-cluster --auto-scaling-role EMR_AutoScaling_DefaultRole --termination-protected --
applications Name=Hadoop Name=Hive Name=Hue Name=JupyterEnterpriseGateway Name=Spark --ebs-root-volume-
size 10 --ec2-attributes
'{"KeyName":"bigdata","InstanceProfile":"EMR_EC2_DefaultRole","SubnetId":"subnet-
5fa2f912","EmrManagedSlaveSecurityGroup":"sg-07818b5690a50b3f1","EmrManagedMasterSecurityGroup":"sg-
0e2f5550a2cb98f79"}' --service-role EMR_DefaultRole --enable-debugging --release-label emr-6.2.0 --log-
uri 's3n://aws-logs-604905954159-us-east-1/elasticmapreduce/' --name 'BigData' --instance-groups
'[{"InstanceCount":1,"BidPrice":"OnDemandPrice","EbsConfiguration":{"EbsBlockDeviceConfigs":[{"VolumeSpe
cification":{"SizeInGB":32,"VolumeType":"gp2"},"VolumesPerInstance":2}]},"InstanceGroupType":"MASTER","I
nstanceType":"m4.xlarge","Name":"Master -
1"},{"InstanceCount":1,"BidPrice":"OnDemandPrice","EbsConfiguration":{"EbsBlockDeviceConfigs":[{"VolumeS
pecification":{"SizeInGB":32,"VolumeType":"gp2"},"VolumesPerInstance":2}]},"InstanceGroupType":"CORE","I
nstanceType":"m4.xlarge","Name":"Core - 2"}]' --scale-down-behavior TERMINATE_AT_TASK_COMPLETION --
region us-east-1
```

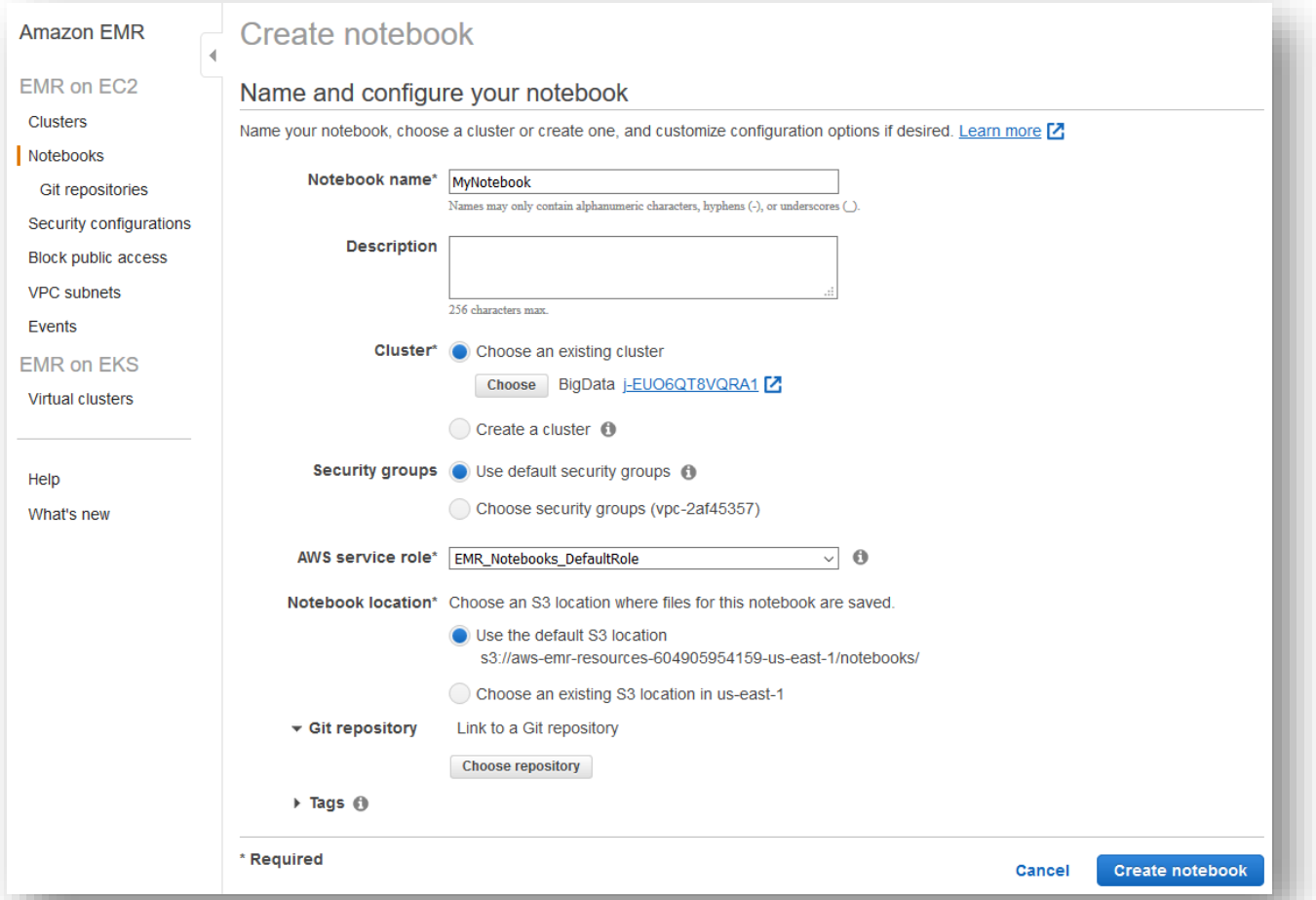
Running the cluster



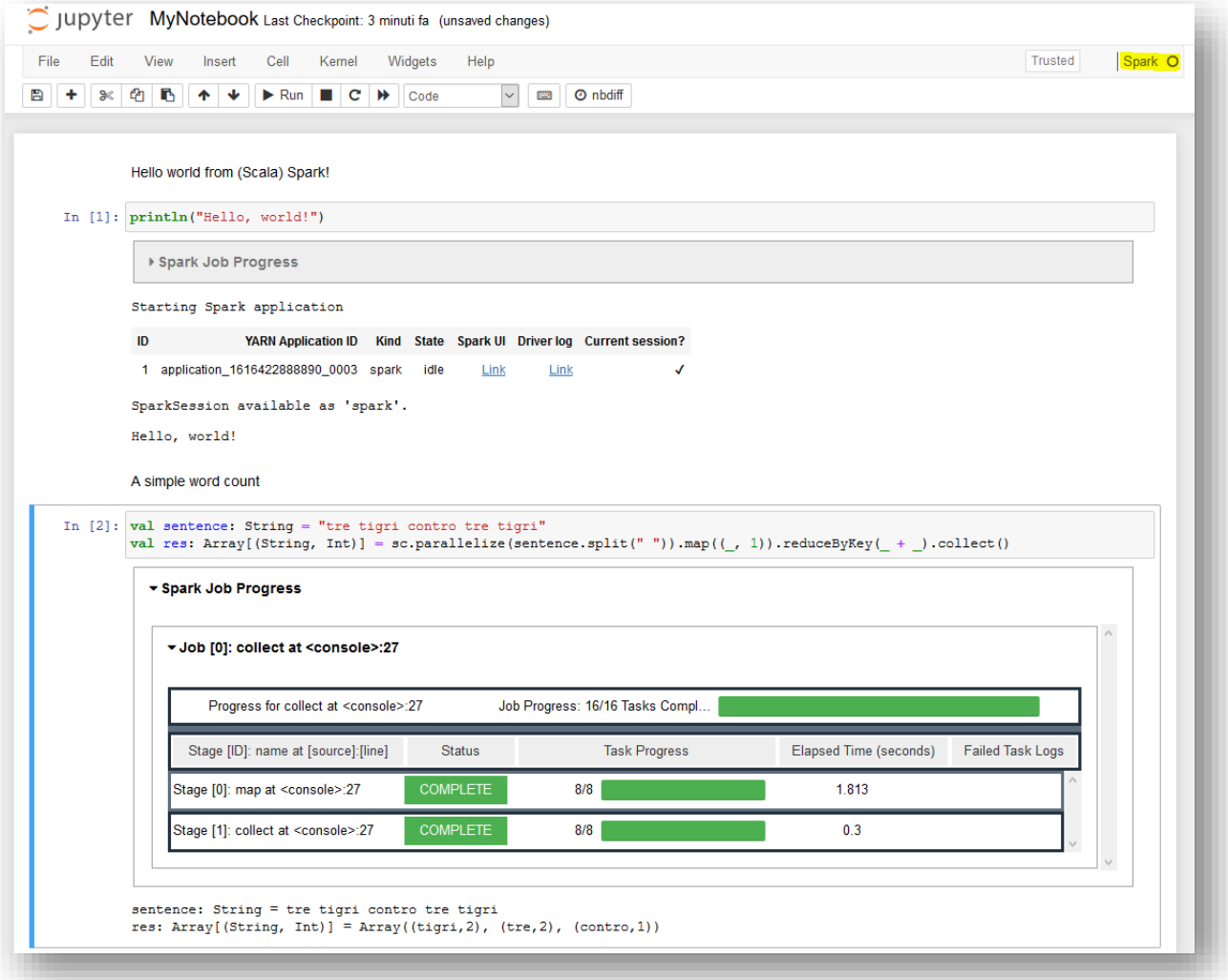
Running the cluster



Creating a notebook



Hello, world!

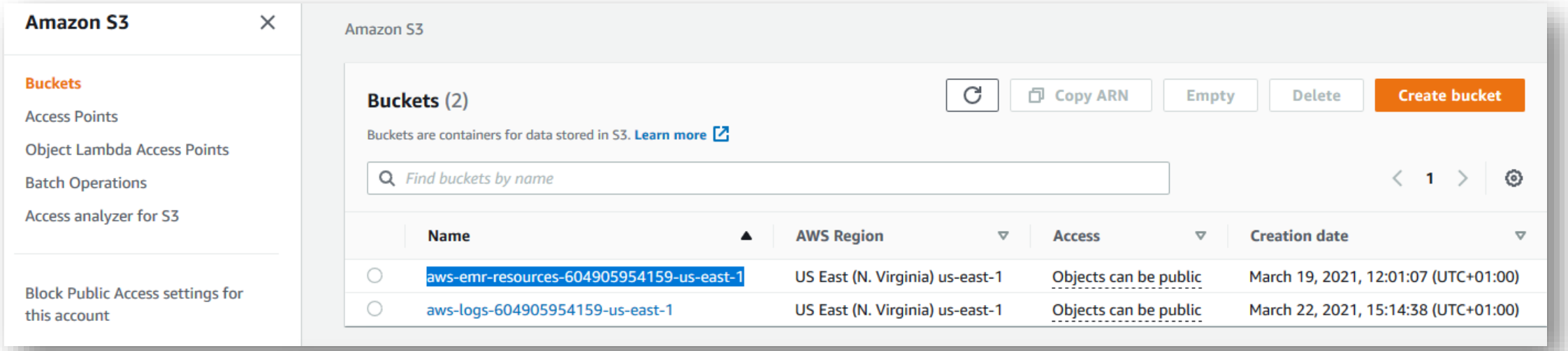

jupyter MyNotebook Last Checkpoint: 3 minuti fa (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Spark
Run
Code
nbdiff
Hello world from (Scala) Spark!
In [1]:
println("Hello, world!")
Spark Job Progress
Starting Spark application
ID YARN Application ID Kind State Spark UI Driver log Current session?
1 application_1616422888890_0003 spark idle Link Link ✓
SparkSession available as 'spark'.
Hello, world!
A simple word count
In [2]:
val sentence: String = "tre tigri contro tre tigri"
val res: Array[(String, Int)] = sc.parallelize(sentence.split(" ")).map((_, 1)).reduceByKey(_ + _).collect()
Spark Job Progress
Job [0]: collect at <console>:27
Progress for collect at <console>:27
Job Progress: 16/16 Tasks Compl...
Stage [ID]: name at [source]:[line]
Status
Task Progress
Elapsed Time (seconds)
Failed Task Logs
Stage [0]: map at <console>:27
COMPLETE
8/8
1.813
Stage [1]: collect at <console>:27
COMPLETE
8/8
0.3
sentence: String = tre tigri contro tre tigri
res: Array[(String, Int)] = Array((tigri,2), (tre,2), (contro,1))

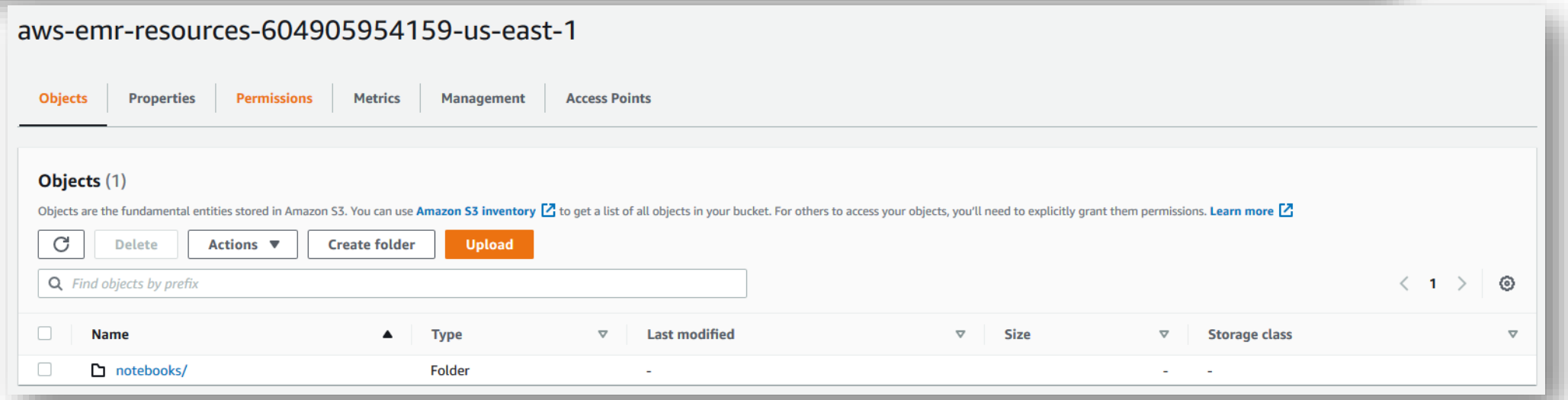
Add some storage

Amazon EMR and Hadoop provide a variety of file systems

- **HDFS** and **EMRFS** are the two main file systems used with Amazon EMR
  - Specify which file system to use by the prefix
- `hdfs://path (or just `path`)`
  - HDFS is used by the master and core nodes
  - Is fast, best used for caching the results produced by intermediate job-flow steps
    - **Why?**
  - It's ephemeral storage which is reclaimed when the cluster ends
- `s3://DOC-EXAMPLE-BUCKET1/path` (EMRFS)
  - An implementation of the Hadoop file system used for reading and writing regular files to Amazon S3
  - Storing persistent data in Amazon S3
  - https://s3.console.aws.amazon.com/s3

Add some storage





Add some storage

Save the result to HDFS

```
import org.apache.hadoop.fs.{FileSystem, Path}
val fs = FileSystem.get(sc.hadoopConfiguration) // get the file system
val outPutPath = new Path(path)
if (fs.exists(outPutPath)) { // delete the HDFS folder if exists
    fs.delete(outPutPath, true)
}

val hdfspath: String = "wordcount" // HDFS path
def writeandread(path: String) = {
    sc.parallelize(res).saveAsTextFile(path) // save the RDD
    val rdd = sc.textFile(path) // read it back
    rdd.collect() // print it
}

writeandread(hdfspath)
```

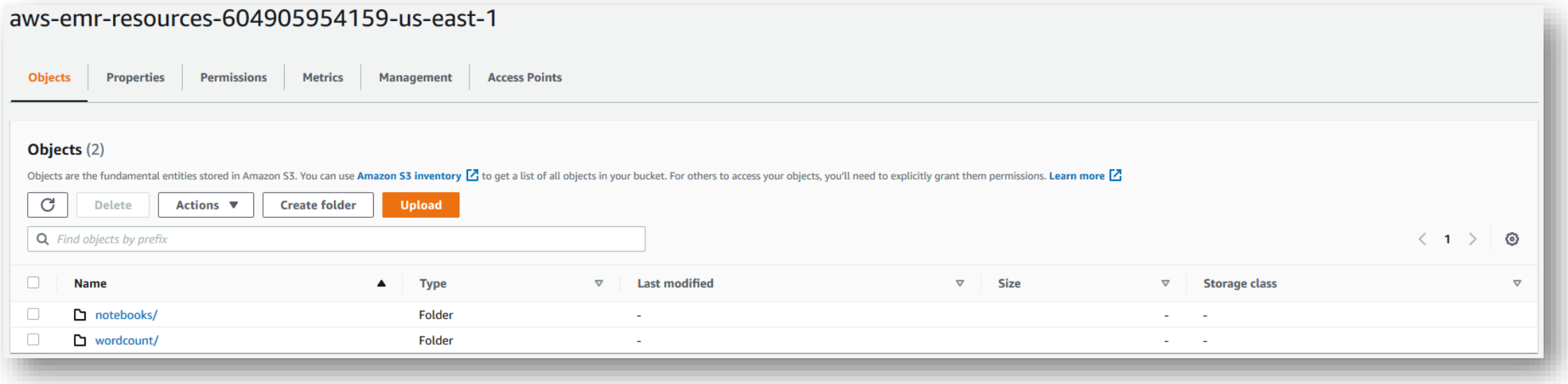
▸ Spark Job Progress

```
import org.apache.hadoop.fs.{FileSystem, Path}
fs: org.apache.hadoop.fs.FileSystem = DFS[DFSClient[clientName=DFSClient_NONMAPREDUCE_1600703682_22, ugi=livy (auth:SIMPL
E)]]
outPutPath: org.apache.hadoop.fs.Path = wordcount
res28: AnyVal = true
hdfspath: String = wordcount
writeandread: (path: String)Array[String]
res32: Array[String] = Array((tigri,2), (tre,2), (contro,1))
```

... and to S3 as well

```
val s3bucket: String = "s3://aws-emr-resources-604905954159-us-east-1/wordcount"
writeandread(s3bucket)
```

Add some storage



Other services: HUE

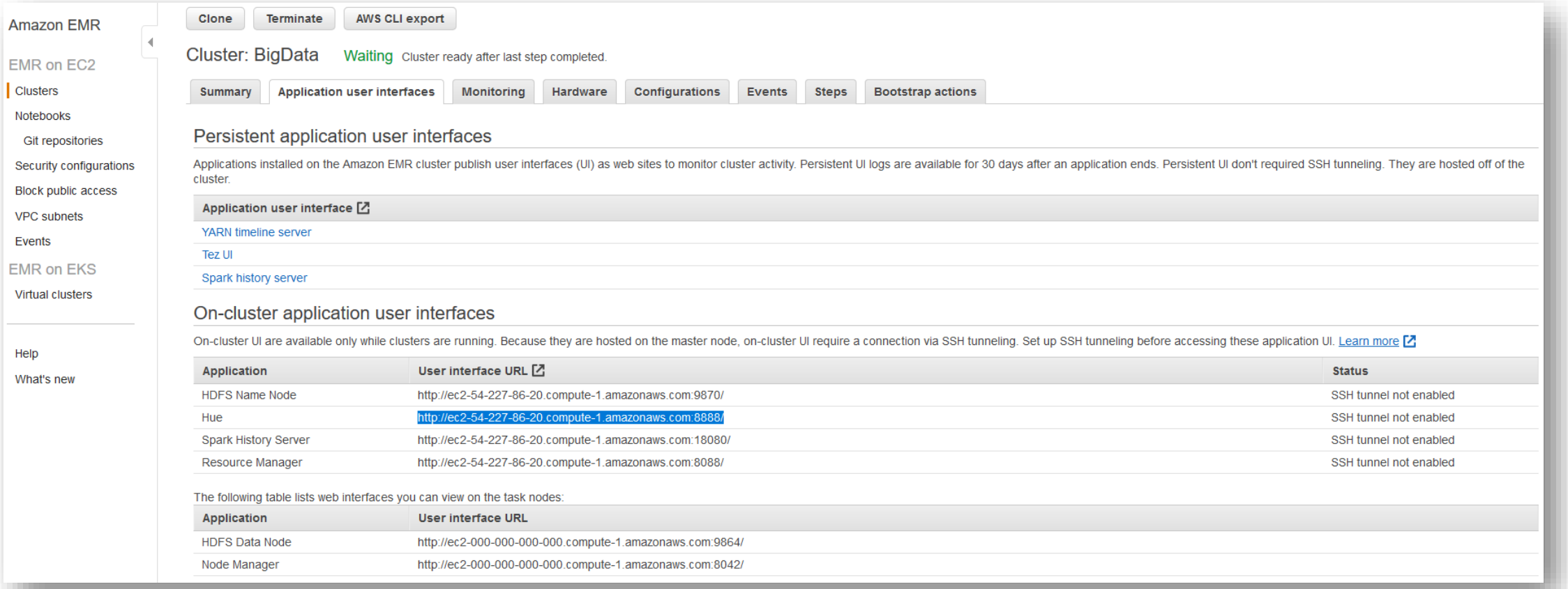
Connecting to Hue

- I.e., connecting to any HTTP interface hosted on the master node of a cluster

To view the Hue web user interface

- Set Up an SSH Tunnel to the Master Node Using Dynamic Port Fowarding
- Type the following address in your browser to open the Hue web interface
  - http://master-public-DNS:8888
  - Where master-public-DNS is the public DNS name of the master node
- If you are the administrator logging in for the first time
  - Enter a user name and password to create your Hue superuser account
  - Otherwise, type your username and password and select Create account

Other services: HUE



Set Up an SSH Tunnel

Security and access

Key name:	bigdata
EC2 instance profile:	EMR_EC2_DefaultRole
EMR role:	EMR_DefaultRole
Auto Scaling role:	EMR_AutoScaling_DefaultRole
Visible to all users:	All Change
Security groups for Master:	sg-0e2f5550a2cb98f79 (ElasticMapReduce-master)
Security groups for Core & Task:	sg-07818b5690a50b3f1 (ElasticMapReduce-slave)

Name	Security group ID	Security group name	VPC ID
–	sg-07818b5690a50b3f1	ElasticMapReduce-slave	vpc-2af45357
–	sg-0e2f5550a2cb98f79	ElasticMapReduce-master	vpc-2af45357

Inbound rules | Outbound rules | Tags

Inbound rules (6) — Edit inbound rules

Type	Protocol	Port range	Source
All TCP	TCP	0 - 65535	0.0.0.0/0
All TCP	TCP	0 - 65535	::/0
All UDP	UDP	0 - 65535	sg-07818b5690a50b3f1 / ElasticMapReduce-slave
All UDP	UDP	0 - 65535	sg-0e2f5550a2cb98f79 / ElasticMapReduce-master
All ICMP - IPv4	ICMP	All	sg-07818b5690a50b3f1 / ElasticMapReduce-slave
All ICMP - IPv4	ICMP	All	sg-0e2f5550a2cb98f79 / ElasticMapReduce-master

Connect to HUE

Application user interfaces

Persistent user interfaces: Spark history server, YARN timeline server, Tez UI

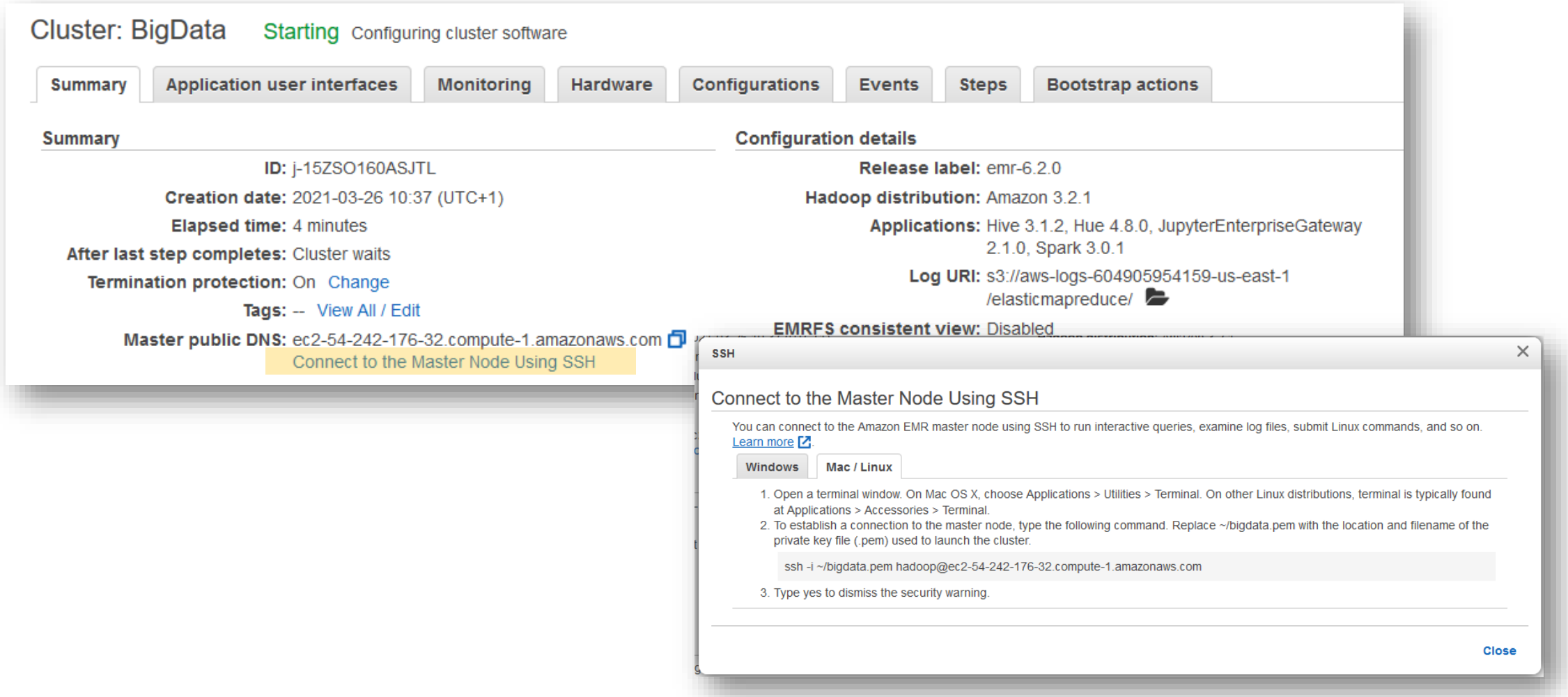
On-cluster user interfaces: HDFS Name Node, Hue, Spark History Server, Resource Manager

On-cluster application user interfaces

On-cluster UI are available only while clusters are running. Because they are hosted on the master node, on-cluster UI require a connection via SSH tunneling. Set up SSH tunneling before accessing these application UI. Learn more

Application	User interface URL	Status
HDFS Name Node	http://ec2-54-242-176-32.compute-1.amazonaws.com:9870/	Available
Hue	http://ec2-54-242-176-32.compute-1.amazonaws.com:8888/	Available
Spark History Server	http://ec2-54-242-176-32.compute-1.amazonaws.com:18080/	Available
Resource Manager	http://ec2-54-242-176-32.compute-1.amazonaws.com:8088/	Available

Connect using SSH



Running a Spark Job

Connect using SSH

Install git

Clone & build the project

```
ssh -i ~/bigdata.pem hadoop@ec2-54-242-176-32.compute-1.amazonaws.com

sudo yum install git -y

git clone https://github.com/w4bo/spark-word-count.git

cd spark-word-count

./gradlew

spark-submit --class it.unibo.big.WordCount build/libs/WordCount-all.jar
                    s3://aws-emr-resources-604905954159-us-east-1/inferno.txt
```

BIG DATA

Migration - Costs

Migration

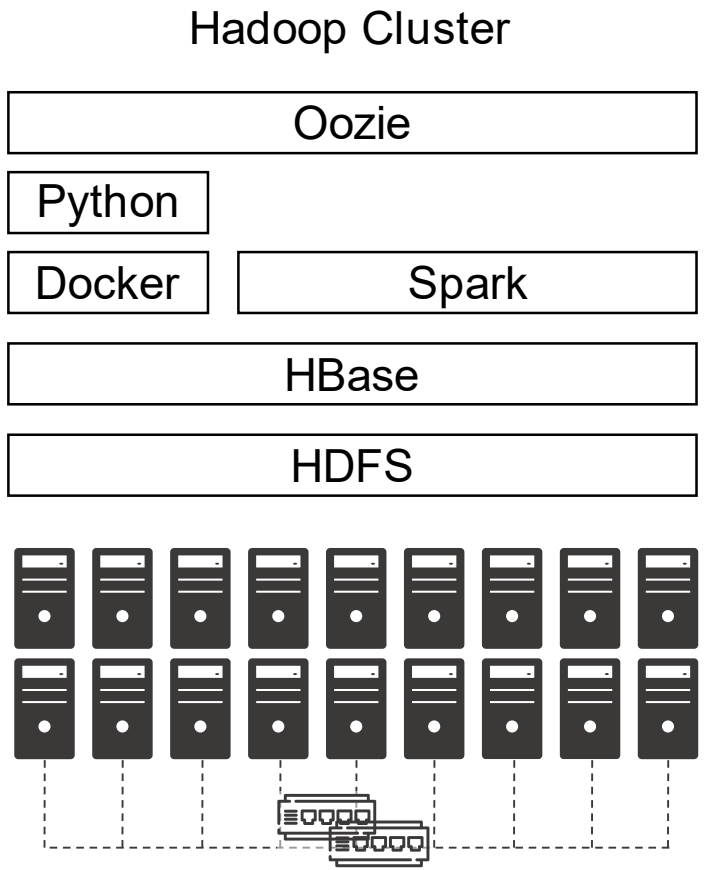
Goals

- Evaluating the costs for a cloud/on-premises data platform
- Real-world case study
- Fill in this table

Cost	**On-premises**	**On cloud**
Hardware	?	?
Software	?	?

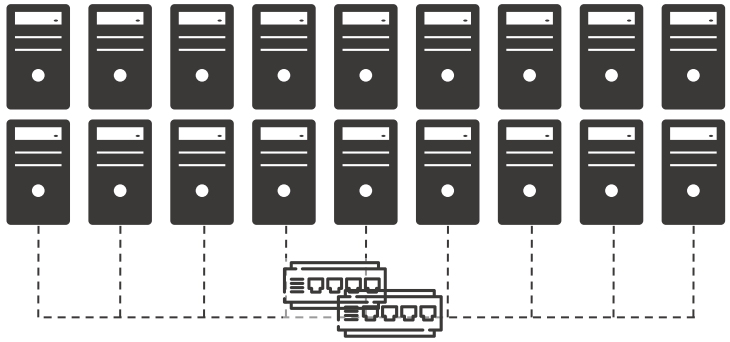
Migration

Reference architecture



Migration

Hardware requirement



8 CPUs (144 total)
- Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz

32GB RAM (576GB total)
- 2x 16GB DIMM DDR4 2666 MHz

12TB HDD Disk (216TB total)
- 4x 4TB ST4000DM004-2CV1

```
lshw -short -C cpu
lshw -short -C memory
lshw -short -C disk
```

Software stack

- Classic Hadoop stack

Migration

SOL_{onprem}	On-premises	On cloud
Hardware	?	?
Software	?	?

Hardware cost: ?

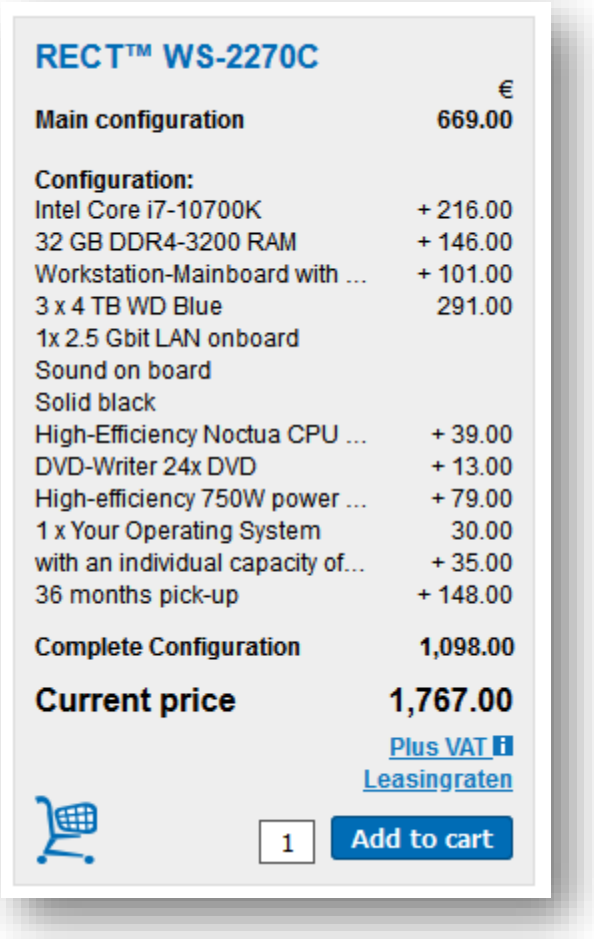
- Refer to https://www.rect.coreto-europe.com/

Migration

SOL_{onprem}	On-premises	On cloud
Hardware	10602€/year	?
Software	?	?

Hardware cost (up to Mar 05, 2021):
1767€ x 18 = 31806€

- Refer to https://www.rect.coreto-europe.com/
- Amortization over 3 years (i.e., 10602€/year)



Migration

SOL_{onprem}	On-premises	On cloud
Hardware	10602€/year	?
Software	0€	?

Software cost: ?

Migration

SOL_{onprem}	On-premises	On cloud
Hardware	10602€/year	?
Software	0€	?

Software cost (up to 2020): 0€

- Free Cloudera Management System
- No software licensing (for research purpose)

Migration

SOL_{onprem}	On-premises	On cloud
Hardware	10602€/year	?
Software	180000€/year	?

Software cost (up to Mar 05, 2021): 10000€/year x 18 = 180000€/year

- Cloudera is no more free, 10K€ per node
- https://www.cloudera.com/products/pricing.html#private-cloud-services
- https://www.cloudera.com/products/pricing/product-features.html
- No license for research purpose

"Houston we've had a problem!"

- We cannot update/extend the cluster anymore
- What about migrating to the cloud? (we only consider AWS)

Migration

Moving a Hadoop cluster to the cloud (we only consider AWS)

- AWS price calculator https://calculator.aws/#/estimate

How do we start?

- We need architectural/application requirements
  - We already defined the hardware and the software stack
- Start with coarse tuning, identify the dominating costs first
  - Is it computing, storage, or processing?
- Identify a suitable budget, implement, refine later
  - Wrong refinements can do a lot of damage

Migration

SOL_{cloud1}	On-premises	On cloud
Hardware	10602€/year	?
Software	180000€/year	?

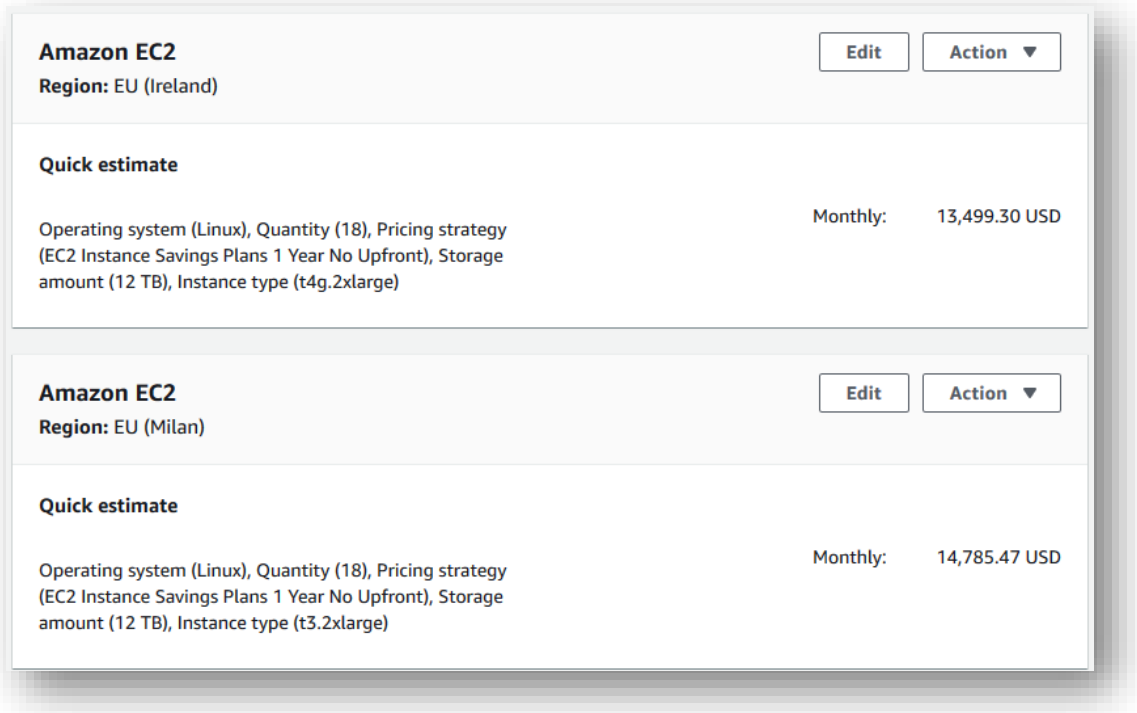
Migrating the cluster as-is: ?

Migration

SOL_{cloud1}	On-premises	On cloud
Hardware	10602€/year	162000$/year
Software	180000€/year	?

SOL_{cloud1} migrating the cluster as-is:
13500$/month = 162000$/year

- 18 EC2 instances (t4g.2xlarge) with 12TB EBS storage each machine
- Still, we have no software stack configuration



Migration

It makes no sense to move the cluster as-is

- More machines ensure better scalability but higher costs
- Let’s think about some optimization

We need have some minimum software requirements

- How many machines do we need at minimum?

Migration

How do we proceed with the migration?

Try to achieve the smallest migration impact

- Find the most similar cloud-based solution to a Hadoop cluster
- Rethink applications when you got the know-how

Rethinking cloud-based applications (business processes) means

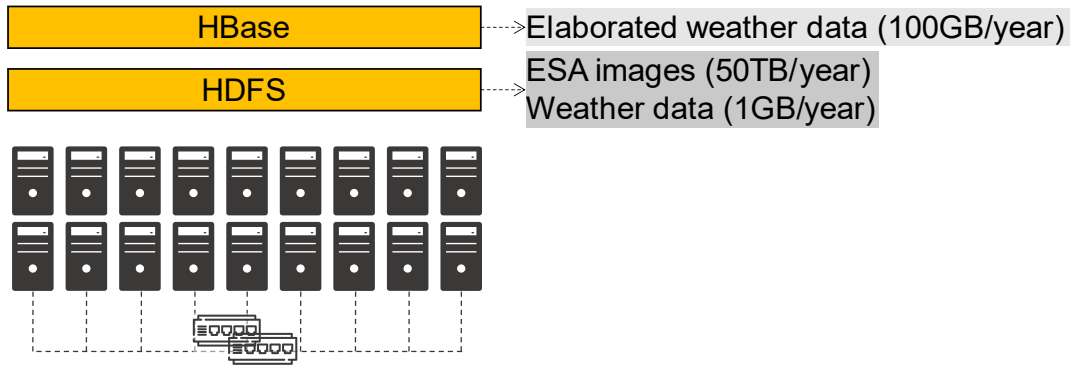
- Understand the application requirements of each process
- Rethink all the applications in a cluster fashion
- Much harder

Migration

Storage

HDFS

- How much durability do we need?
- We can think about a durability decreasing over time
  - HP_1: two replicas for fresh data
  - HP_2: move cold data to glacier or delete id
  - Is it cheaper to store data or (re)process them?



HBase has marginal effects on the pricing (100GB << 50TB)

- For simplicity, we can omit it

Overall: 50TB storage/year

Migration

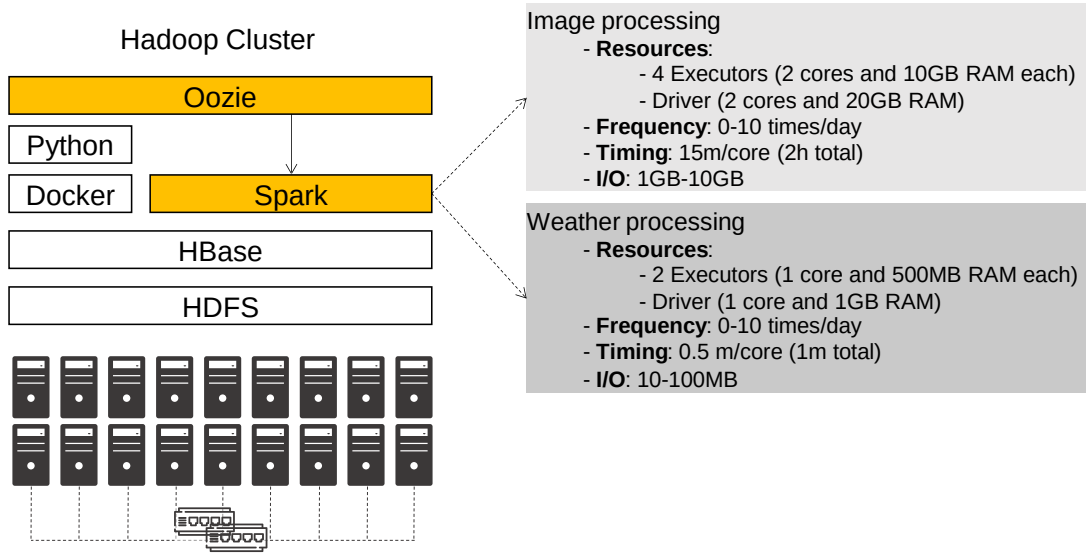
Processing takes place each time that ESA provides a satellite image

- Some days no images are available
- Some days up to 10 images are available
- Each image is about 1GB
- Processing produces new images with about the same size
- Spark jobs are always executed with the same parameters

Image processing

- 4 machines, 2 cores, 10GB RAM at least

Weather processing is negligible



Migration

	On-premises	On cloud
Hardware	2356$/year	16404$/year
Software	100000$/year	?

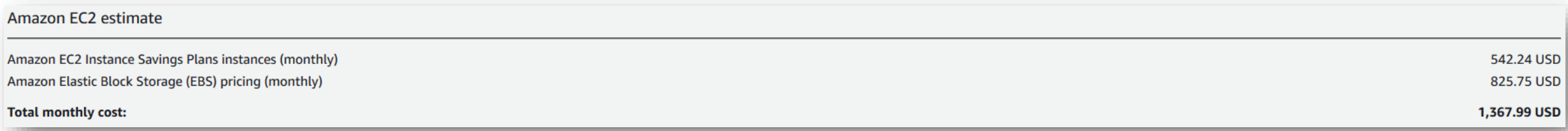
Compare 4 machines on-premises vs on cloud

On-premises

- 4 machines: 196$/month = 2356$/year
- Cloudera requires at least 10 nodes: 100000$/year

Migrate the cluster with minimal requirements: 1367$/month = 16404$/year

- 4 EC2 instances (t4g.2xlarge), 12TB EBS storage each machine
- Still, we have no software stack configuration



Migration

Amazon EMR provides a Hadoop stack similar to Cloudera

- Create/delete a cluster at need
- Computational nodes (based on EC2)
  - Master node, manages the cluster (always active)
  - Core nodes, computation + data
    - Include the HDFS demon (this cannot be turned off)
    - Hard to scale
- Task nodes: core nodes without data demon
  - Write to S3, not to HDFS
  - Easy to (auto)scale
  - Decoupling processing and storage, we loose data locality

Migration

	On-premises	On cloud
Hardware	2356$/year	?
Software	100000$/year	

Migrating cluster to EMR: ?

Migration

	On-premises	On cloud
Hardware	2356$/year	14710€/year
Software	100000$/year	

Migrating cluster to EMR: 14710€/year

- S3 Infrequent Access storage (50 TB per month): 640€
- 1 x Master EMR nodes, EC2 (m4.xlarge), Utilization (75 h/month): 4.5€
  - 75 h/month = 15min/task x 10task/day x 30day/month / 60min/hour
- 1 x Core EMR nodes, EC2 (m4.xlarge), Utilization (75 h/month): 4.5€
- 4 x Task EMR nodes, EC2 (m4.4xlarge), Utilization (75 h/month): 72€
- 4 x EC2 on demand (task node): 174.83€
  - Storage amount (30 GB)
  - Workload (Daily, Duration of peak: 0 Hr 15 Min)
  - Instance type (m4.xlarge)
- 2 x EC2 on demand (master and core nodes): 330€
  - Storage amount (30 GB)
  - Instance type (m4.xlarge)

Migration

Can we do better?

On-Demand Instance

- Pay for compute capacity by the hour (minimum of 60 seconds)
- No long-term commitments

Spot Instance

- Unused EC2 instance that is available for less than the on-demand price
- Hourly price is called a spot price
- Adjusted based on long-term supply and demand for spot instances
- Runs when capacity is available and price lower than threshold
- When data-center resources are low, spot instances are dropped
- Only suitable for batch workloads

Migration

Spot Instance cost strategies

Capacity-optimized strategy

- Allocated instances into the most available pools
- Look at real-time capacity data, predict which are the most available
- Works well for workloads such as big data and analytics
- Works well when we have high cost of interruption

Lowest-price strategy

- Allocates instances in pools with lowest price at time of fulfillment

Migration

	On-premises	On cloud
Hardware	2356$/year	13445€/year
Software	100000$/year	

Migrating cluster to EMR: 13445€/year

- S3 Infrequent Access storage (50 TB per month): 640€
- 1 Master EMR nodes, EC2 (m4.xlarge), Utilization (75 h/month): 4.5€
- 1 Core EMR nodes, EC2 (m4.xlarge), Utilization (75 h/month): 4.5€
- 4 Task EMR nodes, EC2 (m4.4xlarge), Utilization (75 h/month): 72€
- 4 x EC2 spot (task node): 69.55€
  - Storage amount (30 GB)
  - Workload (Daily, Duration of peak: 0 Hr 15 Min)
  - Instance type (m4.xlarge)
- 2 x EC2 on demand (master and core nodes): 330€
  - Storage amount (30 GB)
  - Instance type (m4.xlarge)

Migration

Can we do better?

- Pick ad-hoc cloud services
- E.g., AWS Lambda e AWS Batch (container Docker)
- ... to re-build the applications (food for thoughts)