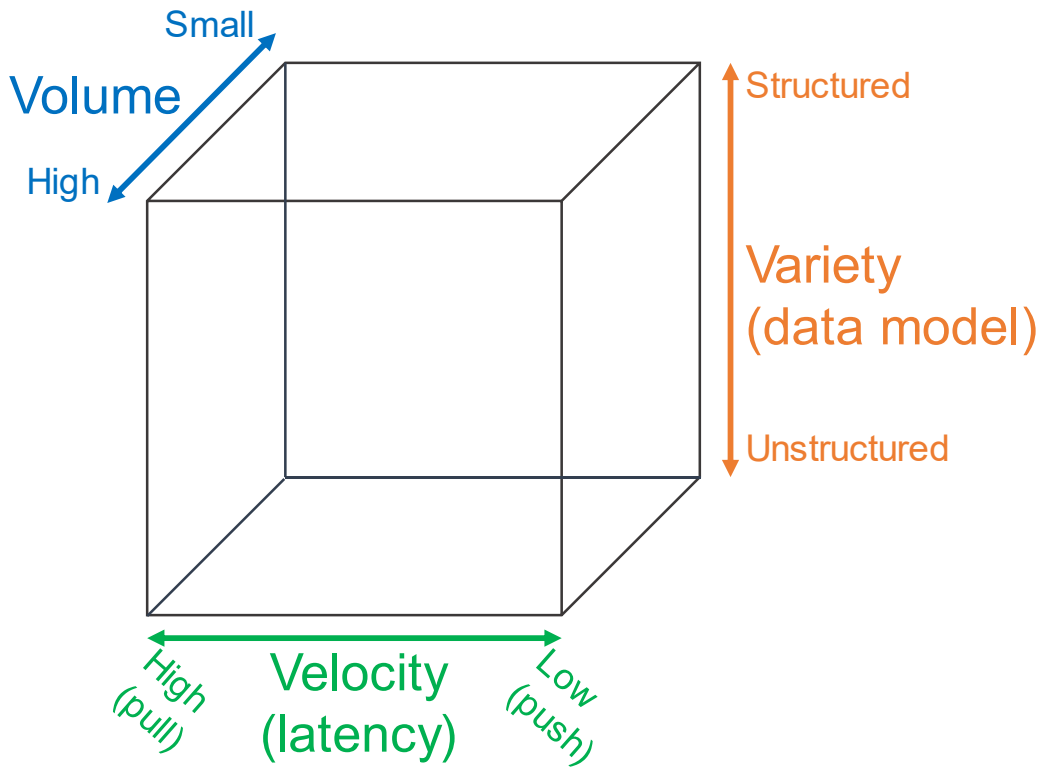
BIG DATA

Building data pipelines

Reference scenario

The big-data cube

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

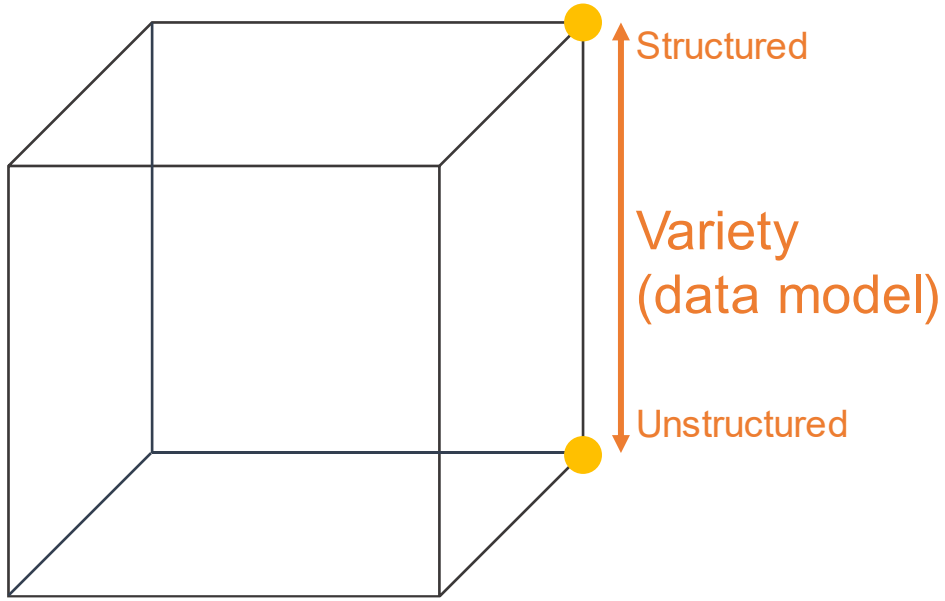


Meijer, Erik. "Your mouse is a database." *Communications of the ACM* 55.5 (2012): 66-73.

Reference scenario

Variety

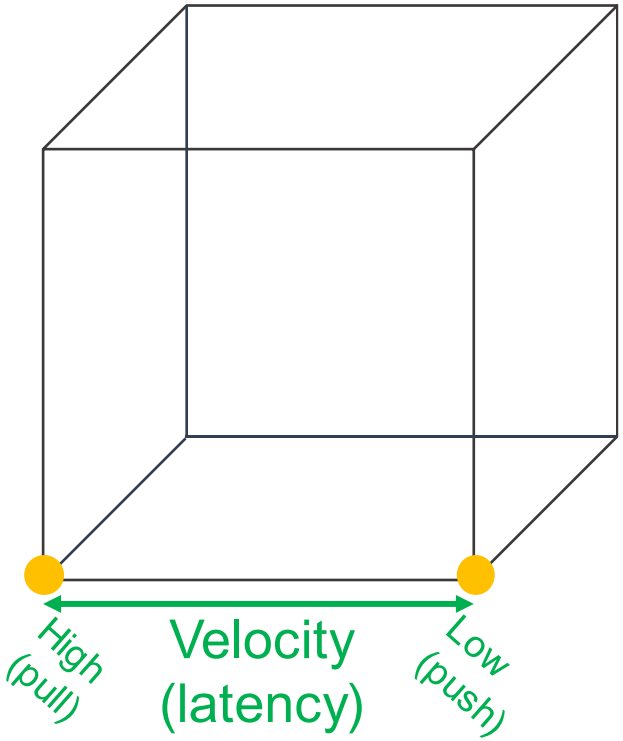
- **Structured**: relational tuples with
  - FK/PK relationships
- **Unstructured**
  - Key-value
  - Columnar
  - Document-based
  - Graph
  - ...



Reference scenario

Velocity (latency)

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

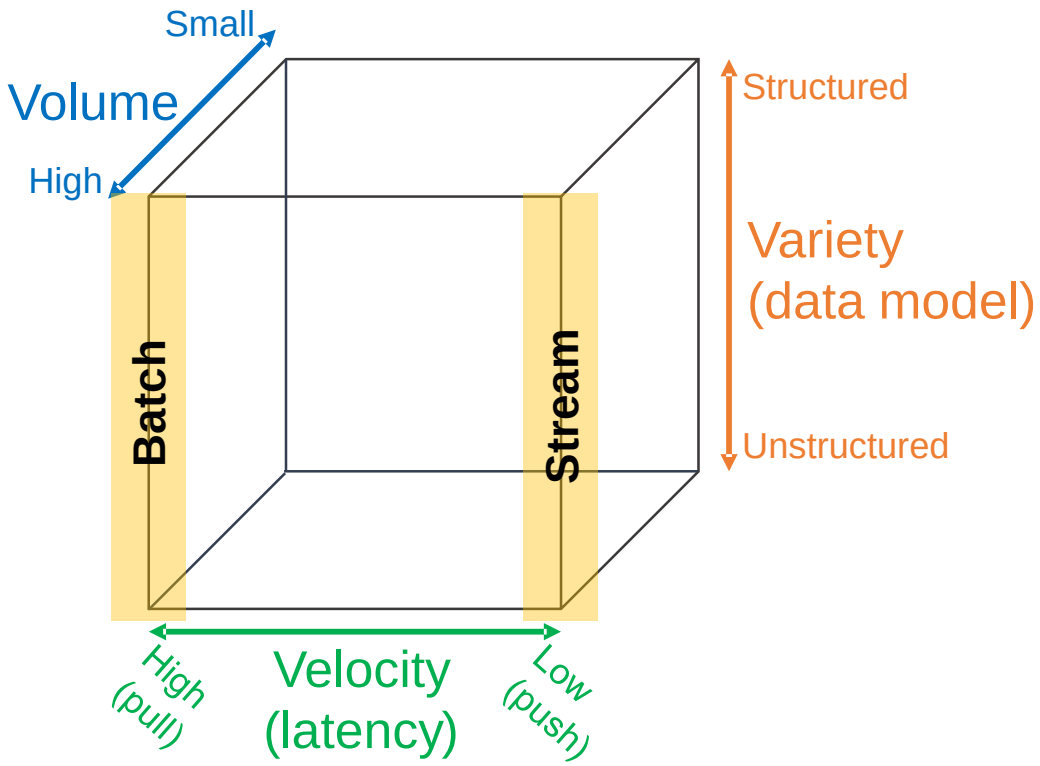


Reference scenario

Our focus

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

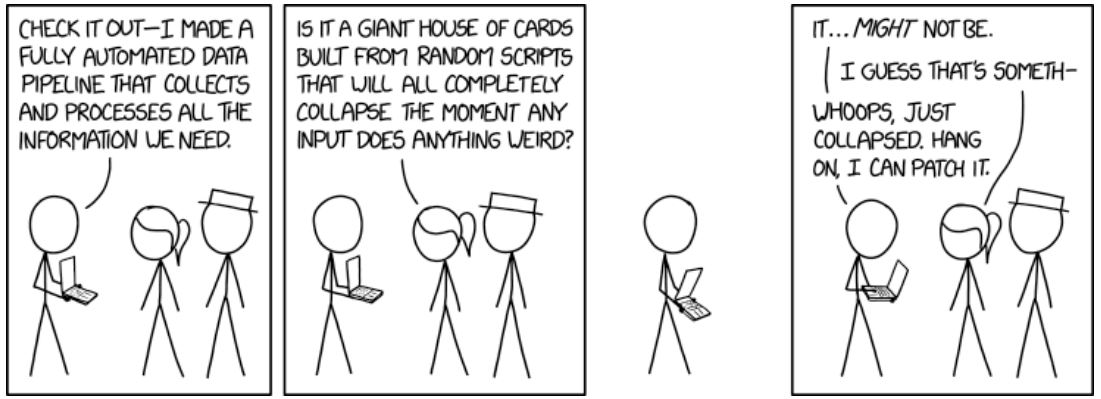
Goal: keep in mind the cube to categorize the services



Data pipeline

Data pipeline

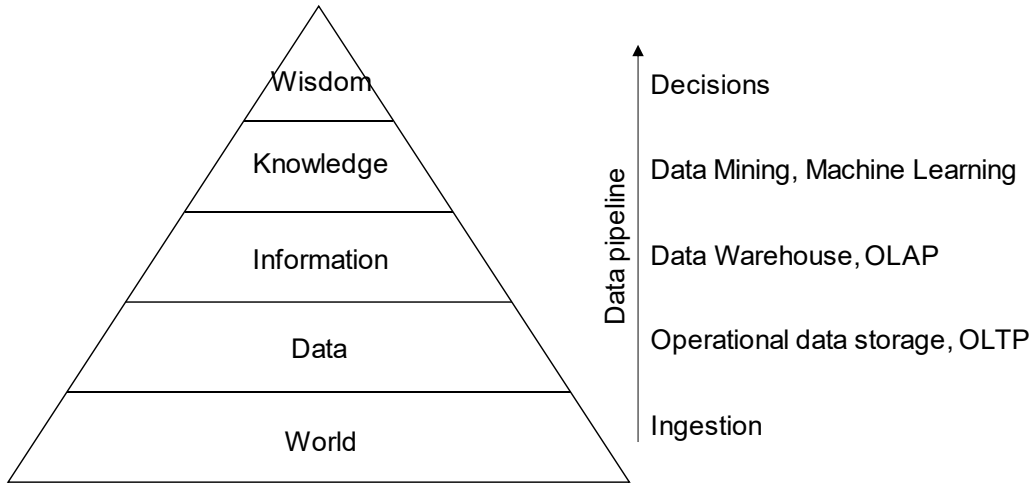
"A succession of operations to transform and consume raw data"



https://xkcd.com/2054/

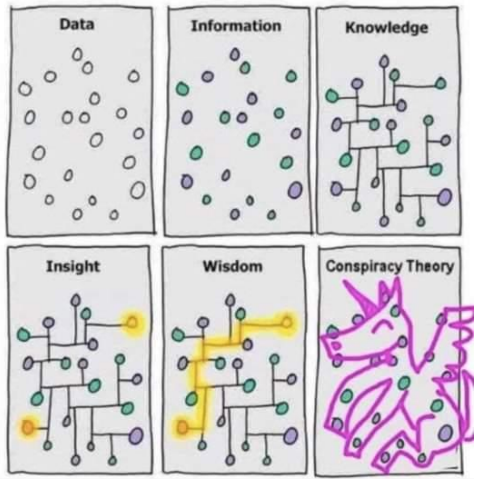
Quemy, Alexandre. "Data Pipeline Selection and Optimization." *DOLAP*. 2019.

Data pipeline



DIKW hierarchy

- Layers representing structural relationships between data, information, knowledge, and wisdom



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

Date pipelines on cloud

The pyramid abstracts tons of techniques, algorithms, etc.

To provide them as services, architecting data pipelines on cloud requires

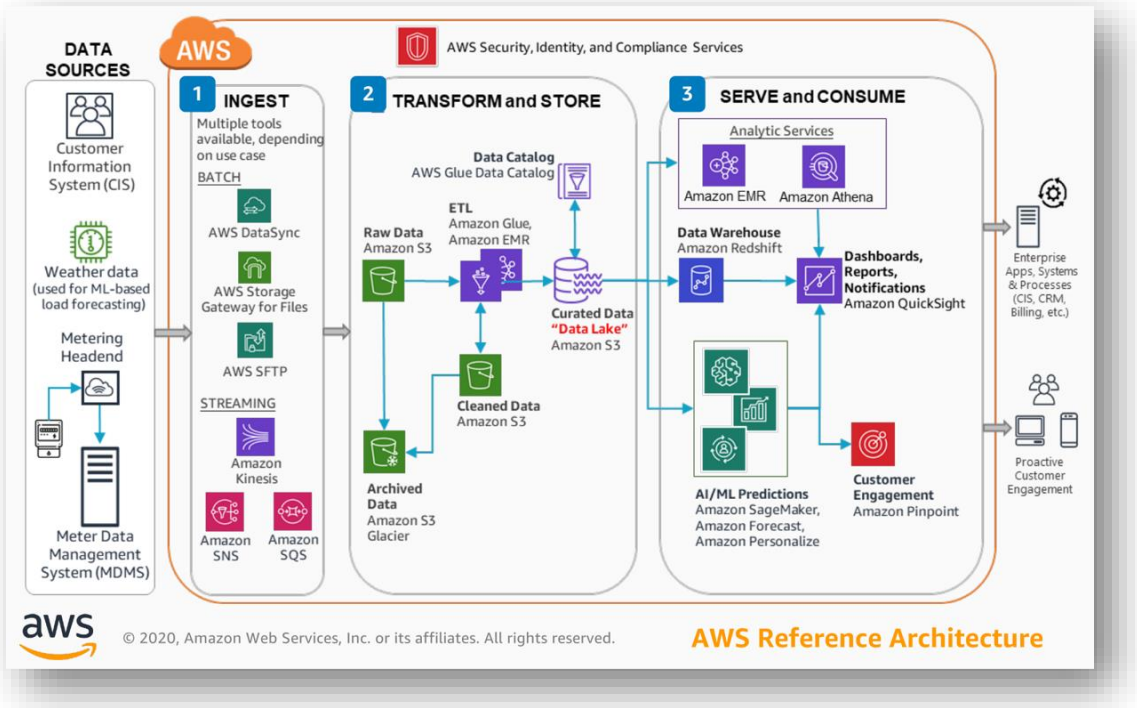
- Standardization (of common services)
- Integration
- Orchestration
- Accessibility through simple APIs

Let us look to data pipelines on different cloud services providers

Data pipeline - AWS

Three main categories

- Ingest
  - Gateway, DataSync (batch)
  - Kinesis, SNS, SQS (stream)
- Transform and store
  - S3 and Glacier (storage)
  - Glue (ETL)
- Serve and consume
  - EMR (Hadoop-like cluster)
  - Athena (serverless query service to analyze data in Amazon S3)
  - (Many) Machine learning services

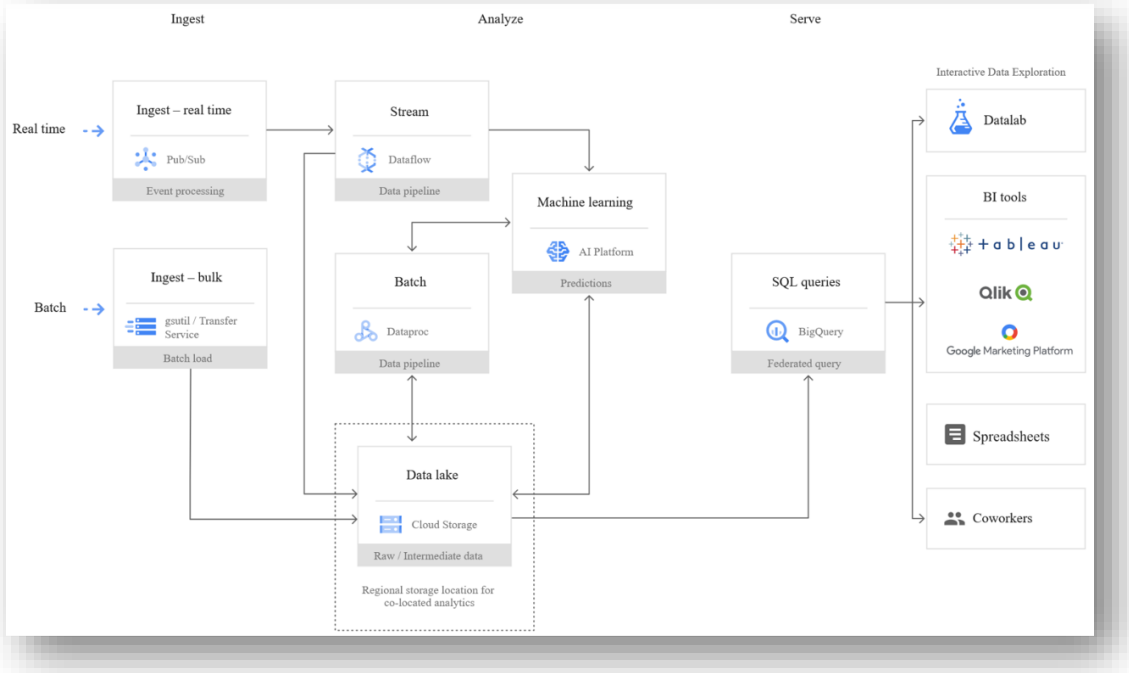


https://console.aws.amazon.com/console

Data pipeline - Google cloud

Three main categories

- Ingest
  - Transfer service (batch)
  - Pub/Sub (stream)
- Analyze
  - Dataproc (batch)
  - Dataflow (stream)
  - Cloud storage (storage)
  - Machine learning services
- Serve
  - BigQuery (query service)



A tentative organization

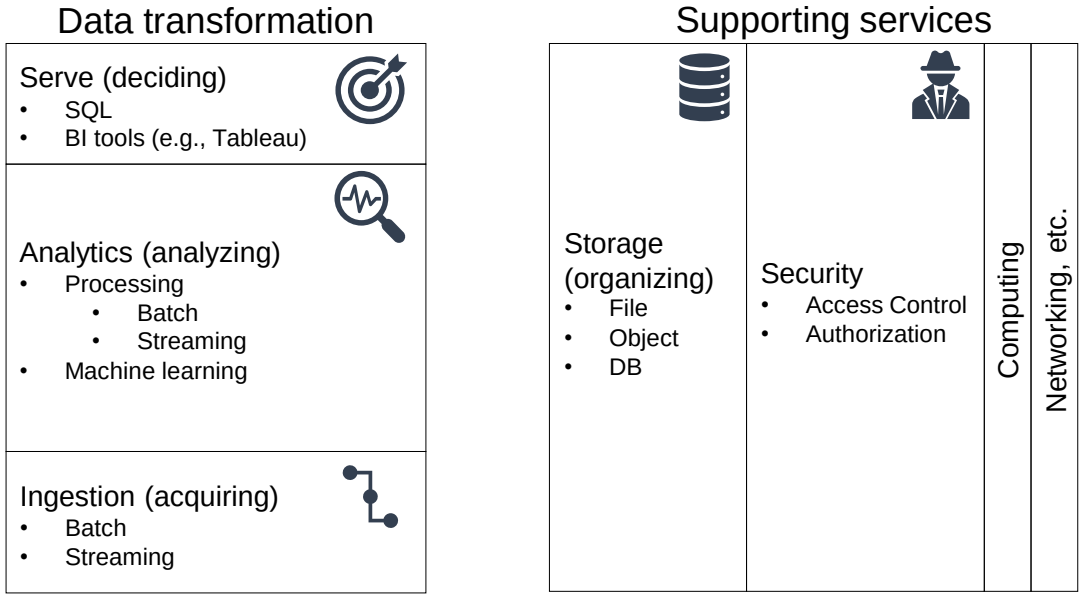
Common points

We have services

- Which transforms data
  - Recall the DIKW pyramid
- Which supports data transformation

Data pipelines are based on

- Ingesting data
- Analyzing data
- Serving data

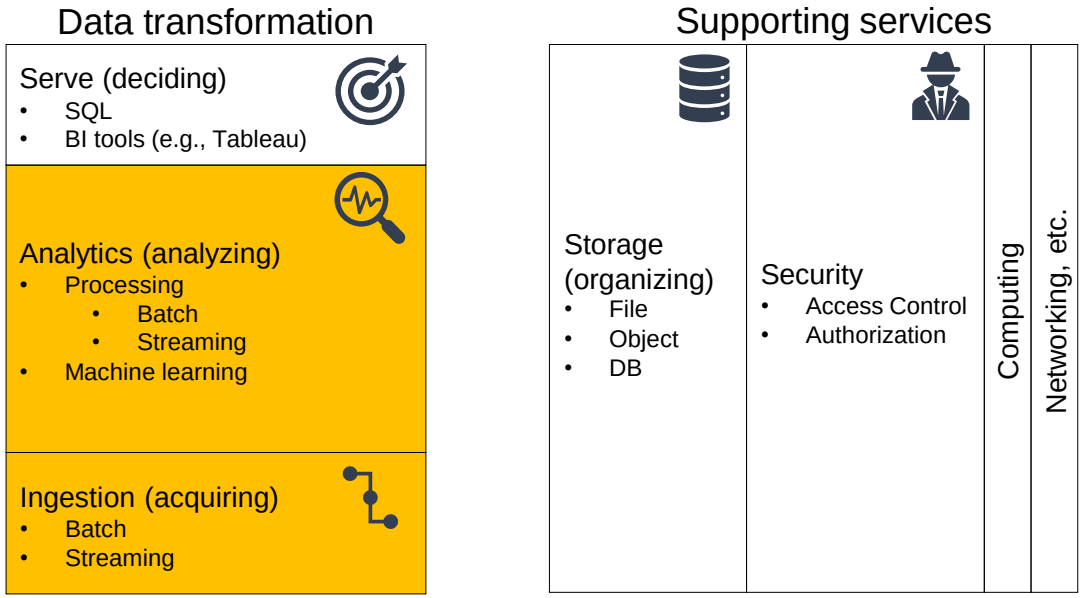


A tentative organization

This is not a sharp taxonomy

Ingestion vs Analytics

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

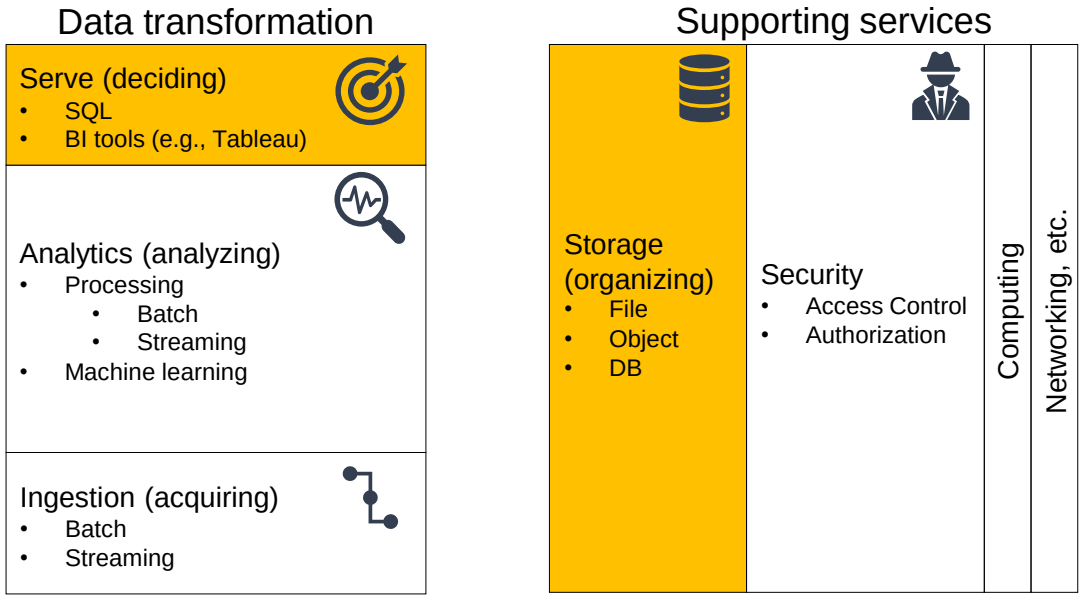


A tentative organization

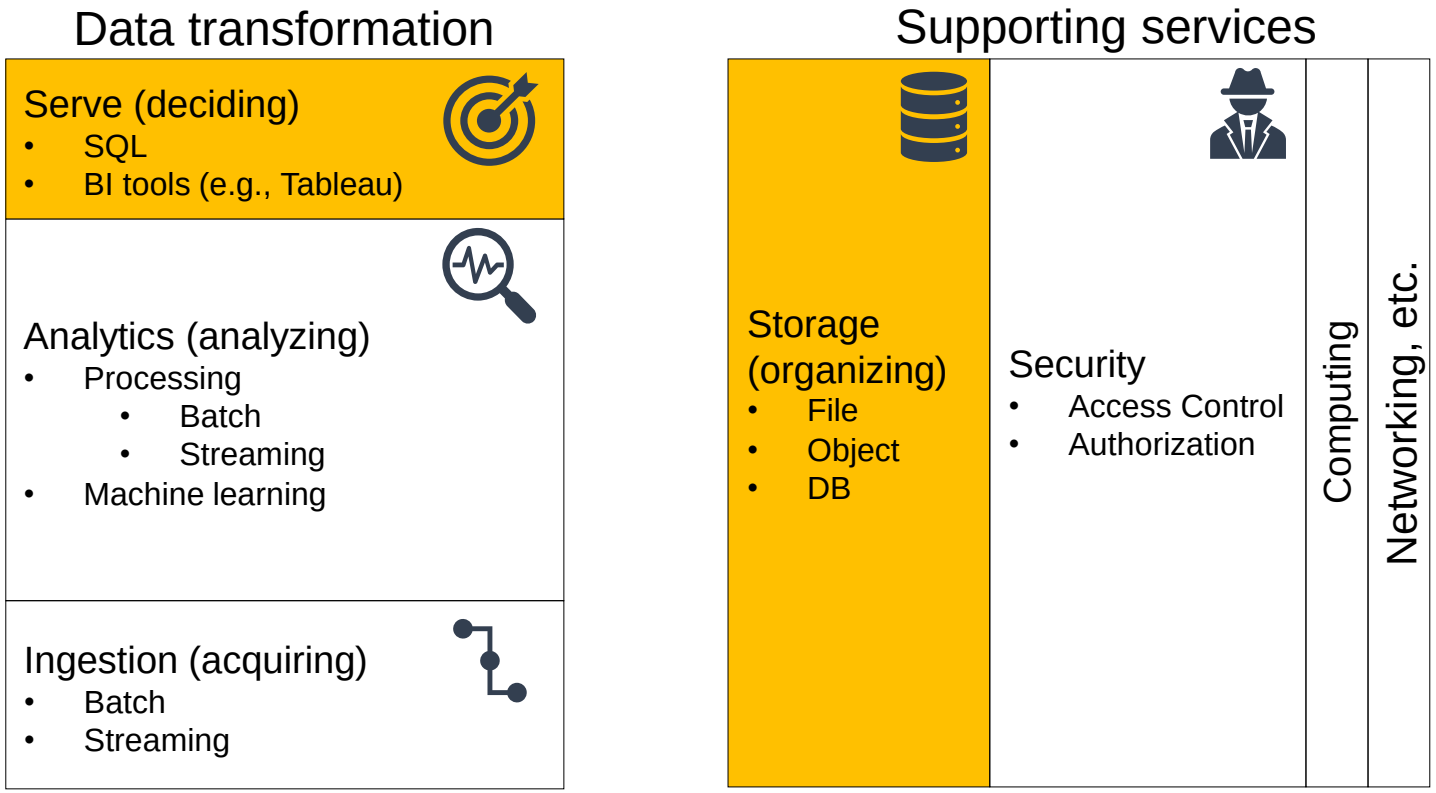
This is not a sharp taxonomy

Storage vs Serving

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



A tentative organization



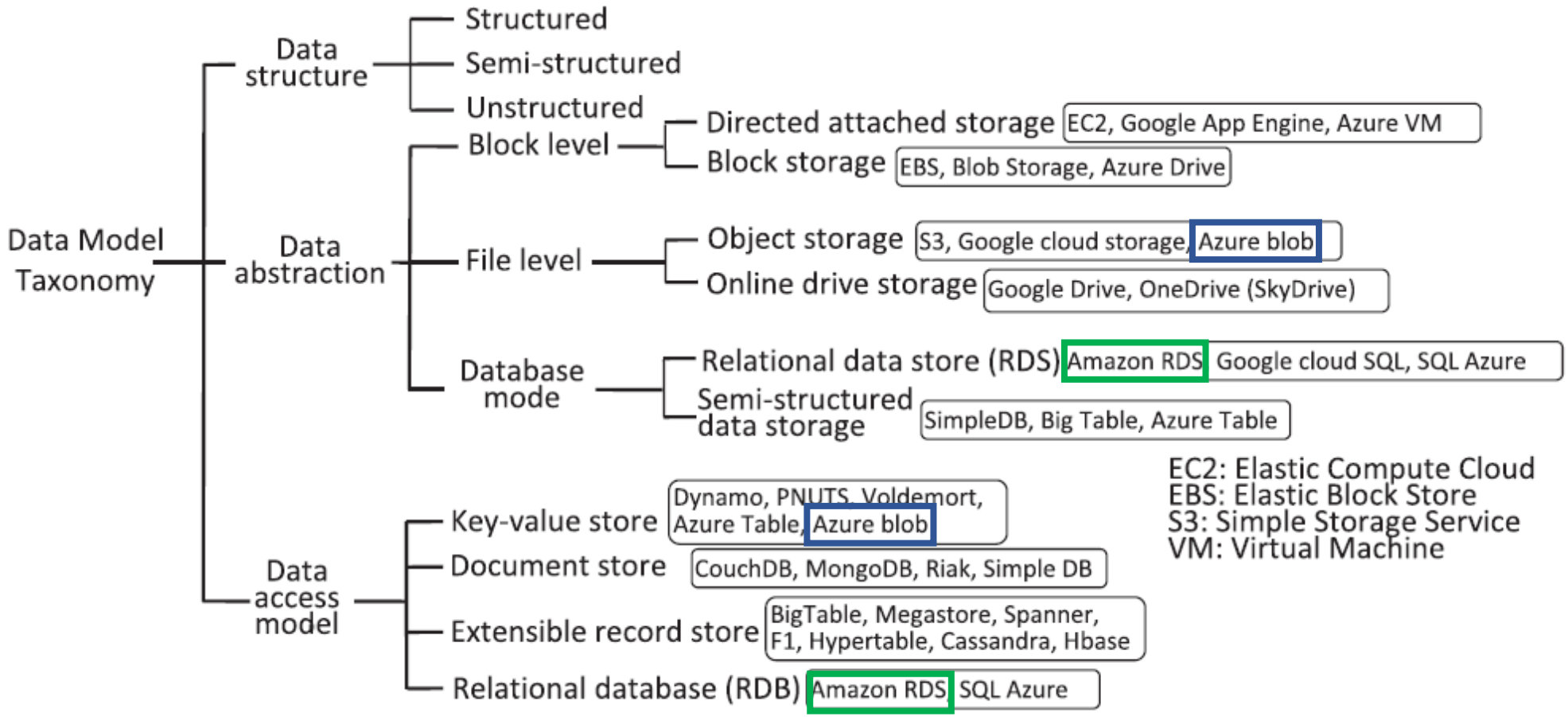
Storage

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 (AWS)

Data structure: structured

Data abstraction: database

Data access model: relational

Relational

- Store data with predefined schemas and relationships between them
- Support ACID transactions
- 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 (AWS)

Data structure: *

Data abstraction: database

Data access model: *

- **Key/value:** store and retrieve large volumes of data
- **Document :** store semi-structured data as JSON-like documents
- **Columnar:** use tables but unlike a relational database, columns can vary from row to row in the same table
- **Graph:** navigate and query relationships between highly connected datasets
- **... 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 models (Google Cloud)

	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 (AWS)

Data structure: unstructured

Data abstraction: file (or database)

Data access model: key-value

File system (EFS), **object storage** (S3) (or **DB K-V**; e.g., 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, latency, etc.

Storage: access frequency (AWS)

Object storage (AWS S3) **classes**

- **Standard**: general purpose
- **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
- **Intelligent-Tiering**: move objects between access tiers when access patterns change

	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 (AWS)

Lifecycle configuration

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

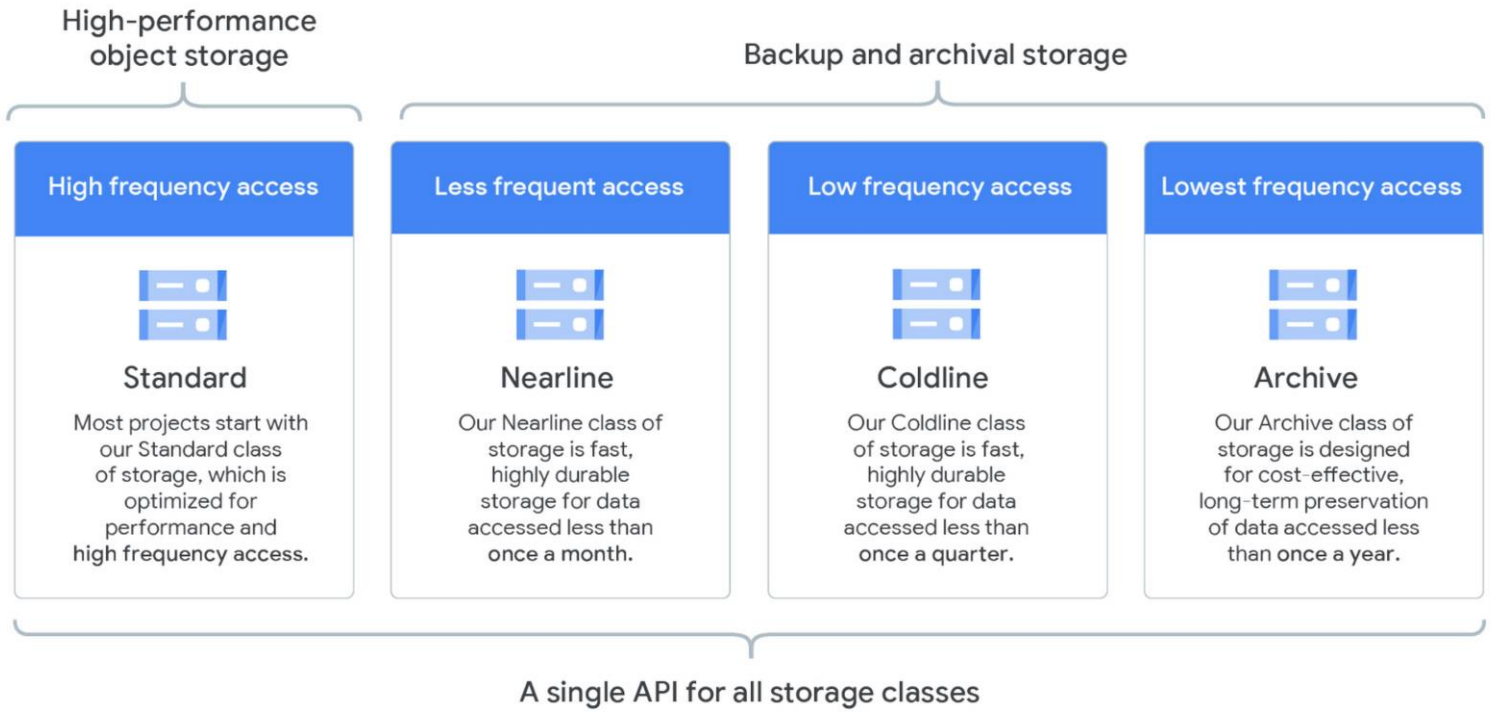
Two types of actions:

- **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

	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)	(11 9's)	(11 9's)	(11 9's)	(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 (Google Cloud)



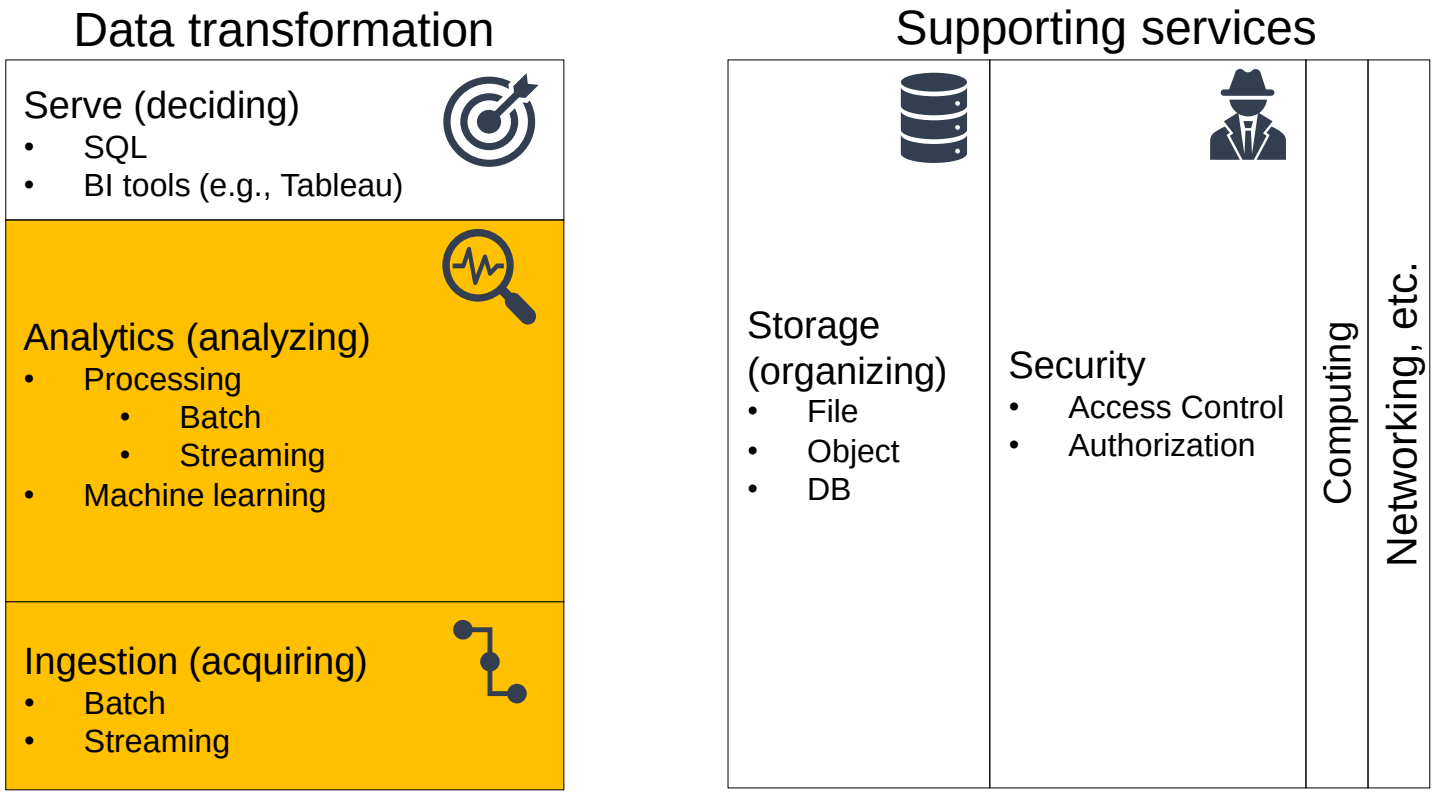
Retention Period

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?

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)

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

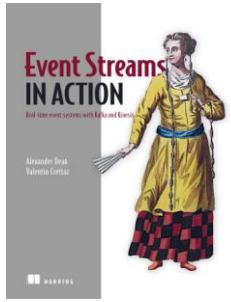
Event: anything that we can observe occurring at a particular point in time

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

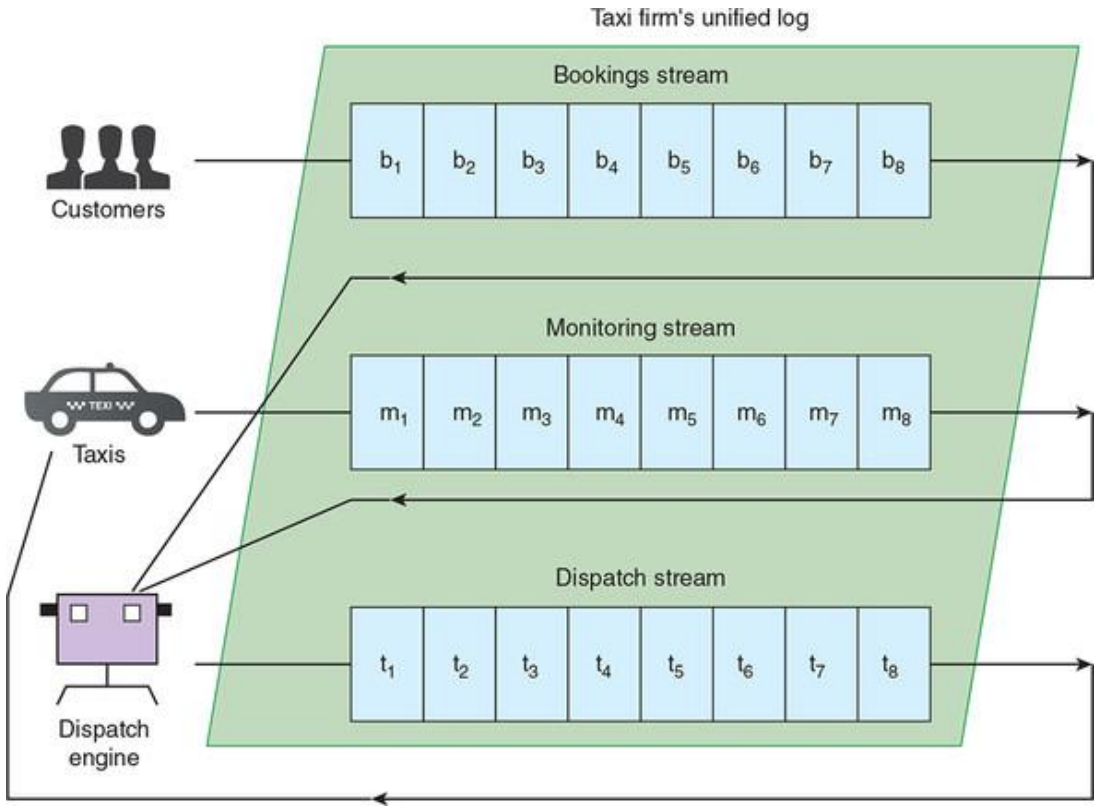
Ingestion: stream

General idea:

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

Unified log

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



Ingestion: stream

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

- Log serves as central data backbone
  - It 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

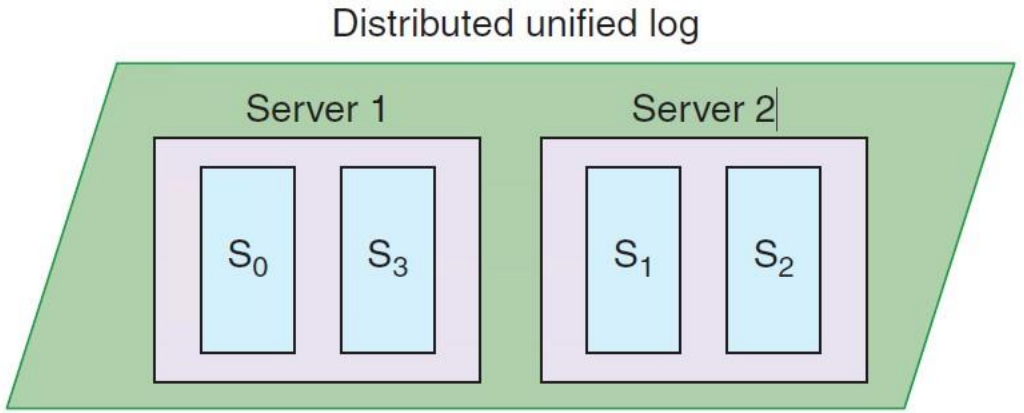
Ingestion: stream

Distributed: the unified log lives across a cluster of machines

- Optionally divide events into shards (i.e., partitions)
  Still, the log is unified since we have a single (conceptual) log

Distribution ensures

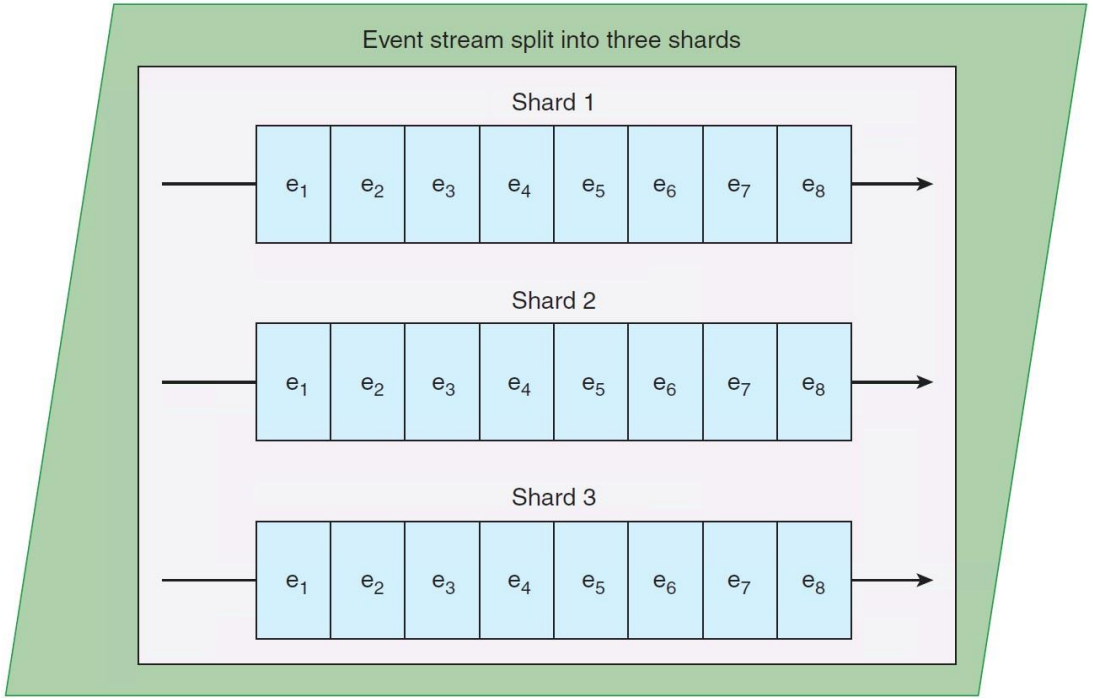
- Scalability: work with streams larger than the capacity of single machines
- Durability: replicate all events within the cluster to overcome data loss



Ingestion: stream

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

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

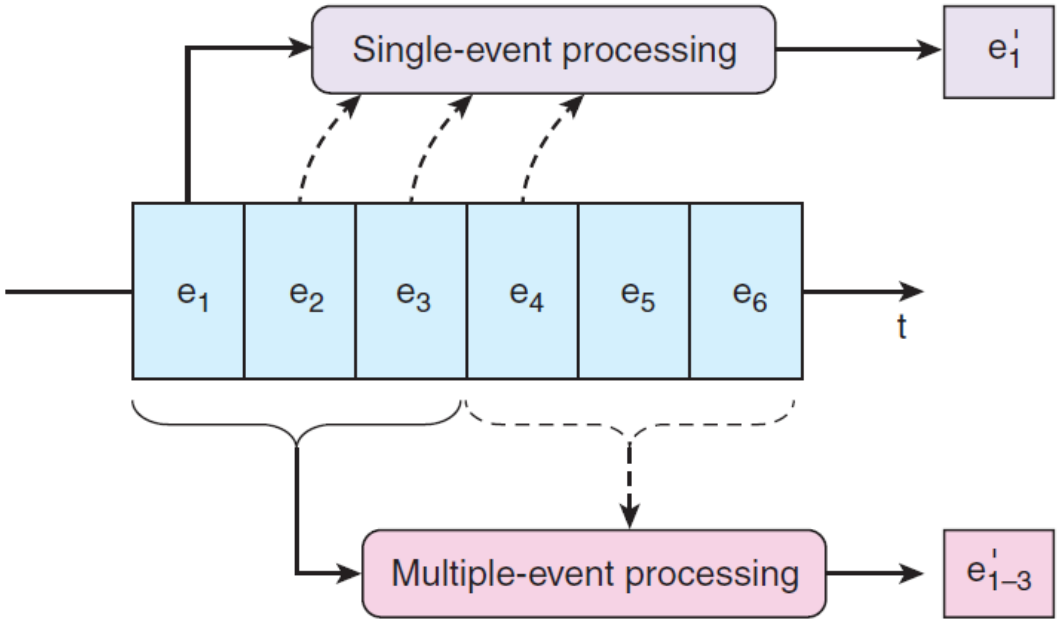


Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." *Concurrency: the Works of Leslie Lamport*. 2019. 179-196.

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

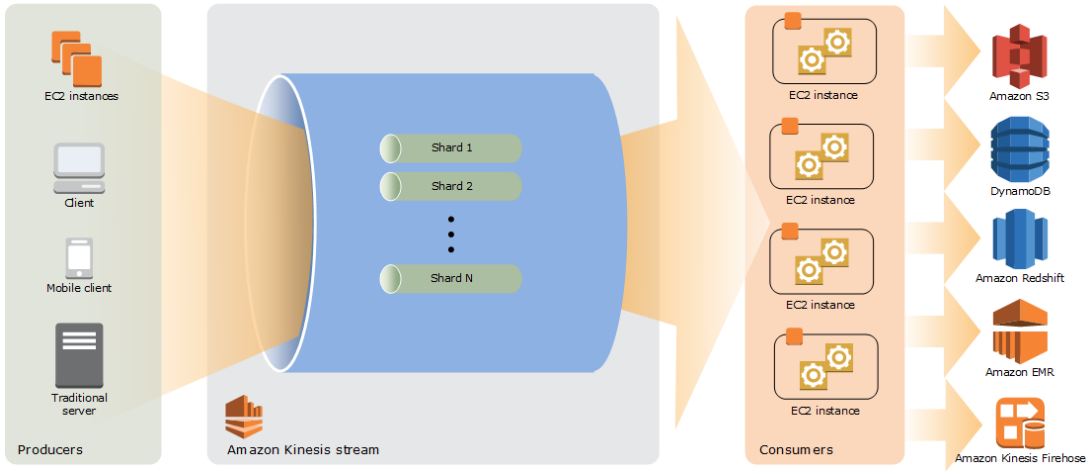


Ingestion: stream (AWS)

Amazon Kinesis Data Streams

- Created and provisioned by shard
  - Each shard provides 1 MBps and 1000 data puts per second
- A data record consists of
  - User-supplied partition key to balance records across shards
  - Incremental sequence number added by the shard
  - A data blob
- Consumers get records by shard
  - Records are sorted by partition key and sequence number
  - Ordering is not guaranteed across shards
- Records are retained for 7 days at maximum

https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html



Ingestion: stream (AWS)

Resharding (i.e., scaling)

- 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

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

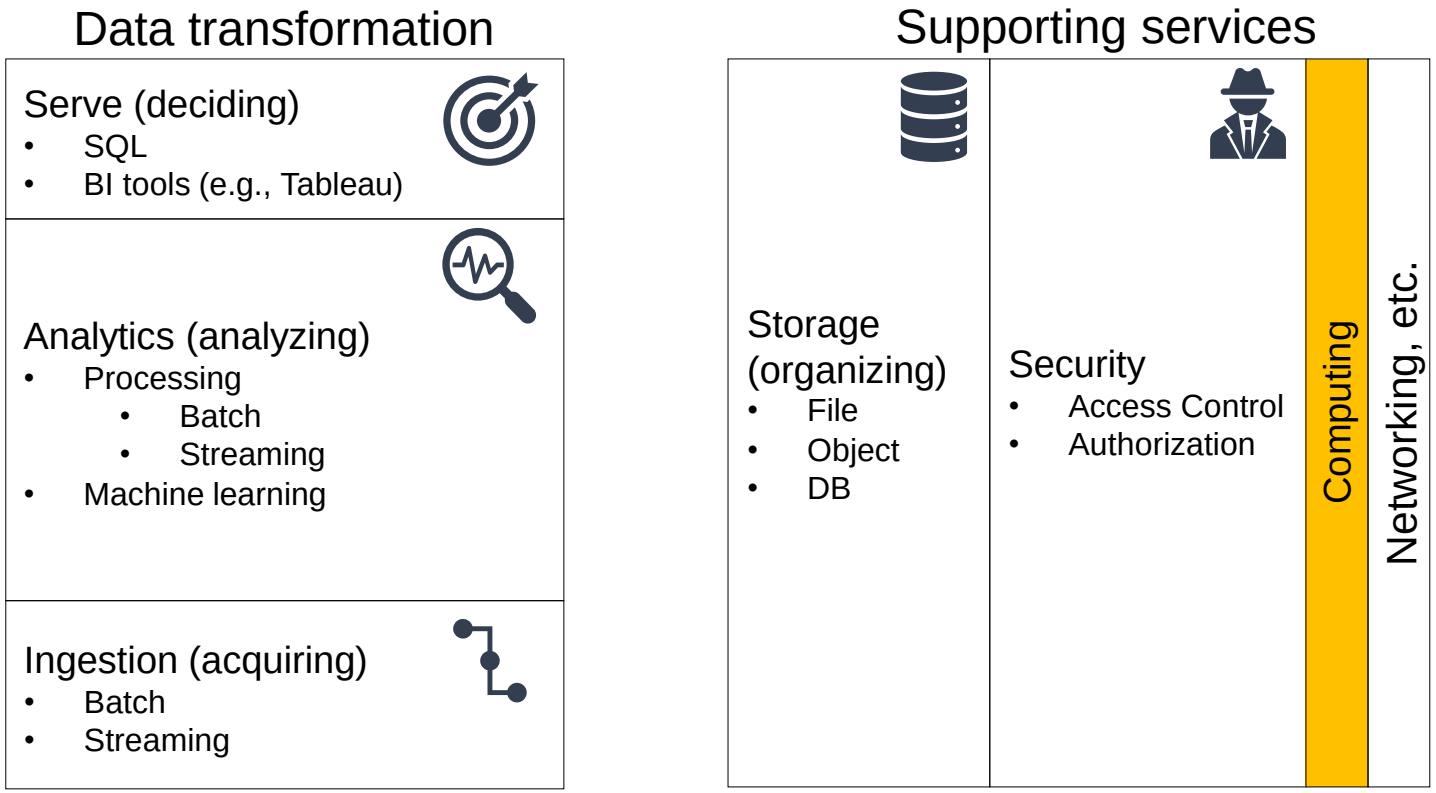
- Priced by shard hour, data volume, and data retention period
- Pay for resources you provision (even if not used)

https://aws.amazon.com/cloudwatch/
https://aws.amazon.com/kinesis/data-firehose

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



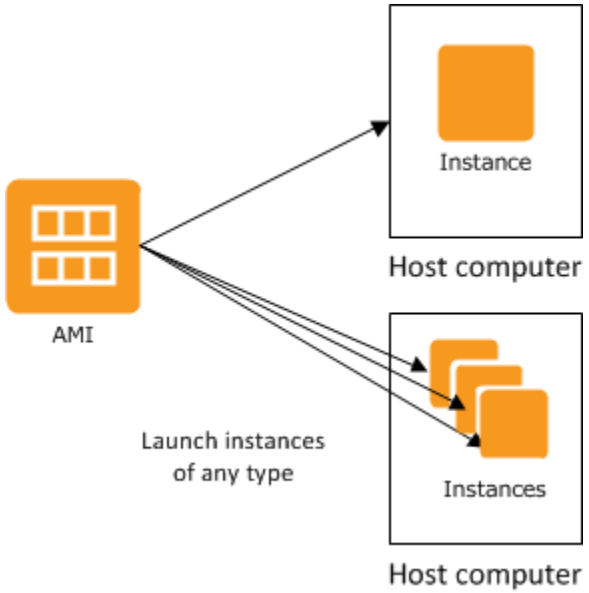
Single instance: AWS EC2

Amazon Elastic Compute Cloud

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

Amazon Machine Image is a template of a software configuration

- E.g., an operating system, an application server, and applications
- From an AMI, you launch (multiple) instances running as a virtual servers



https://aws.amazon.com/ec2/
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/compute-optimized-instances.html (example of AMI)

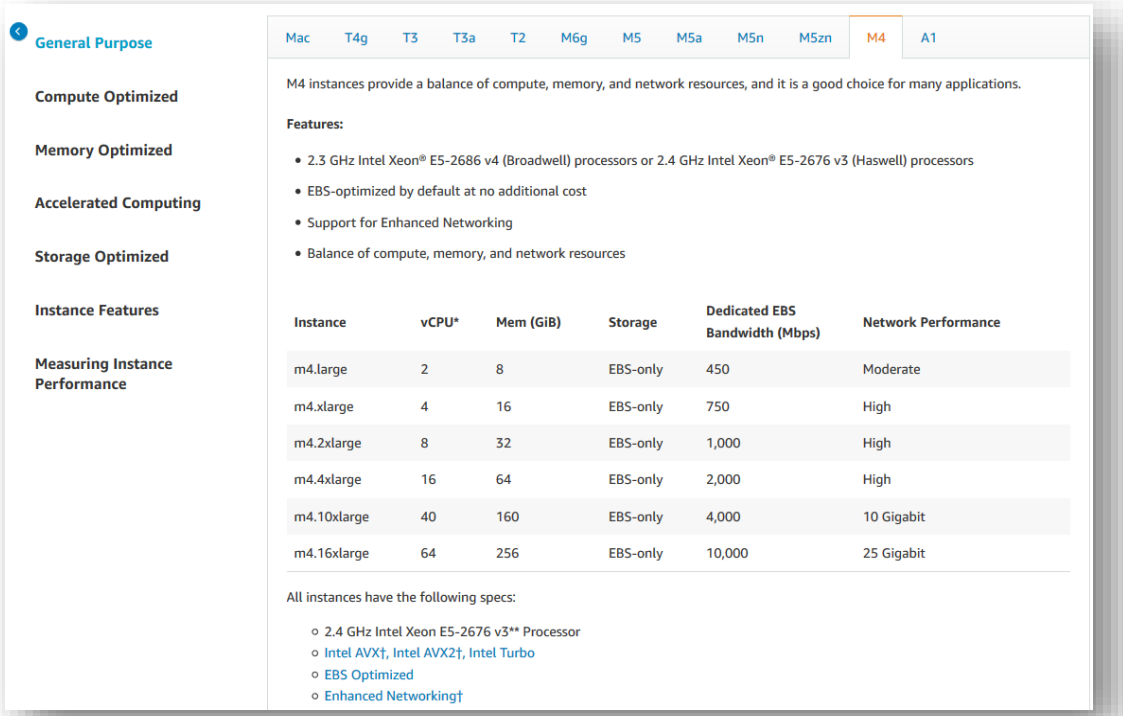
Single instance: AWS EC2

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 with any computer
- You have complete control of your instances
  - E.g., `sudo` to run commands

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

Single instance: AWS 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.

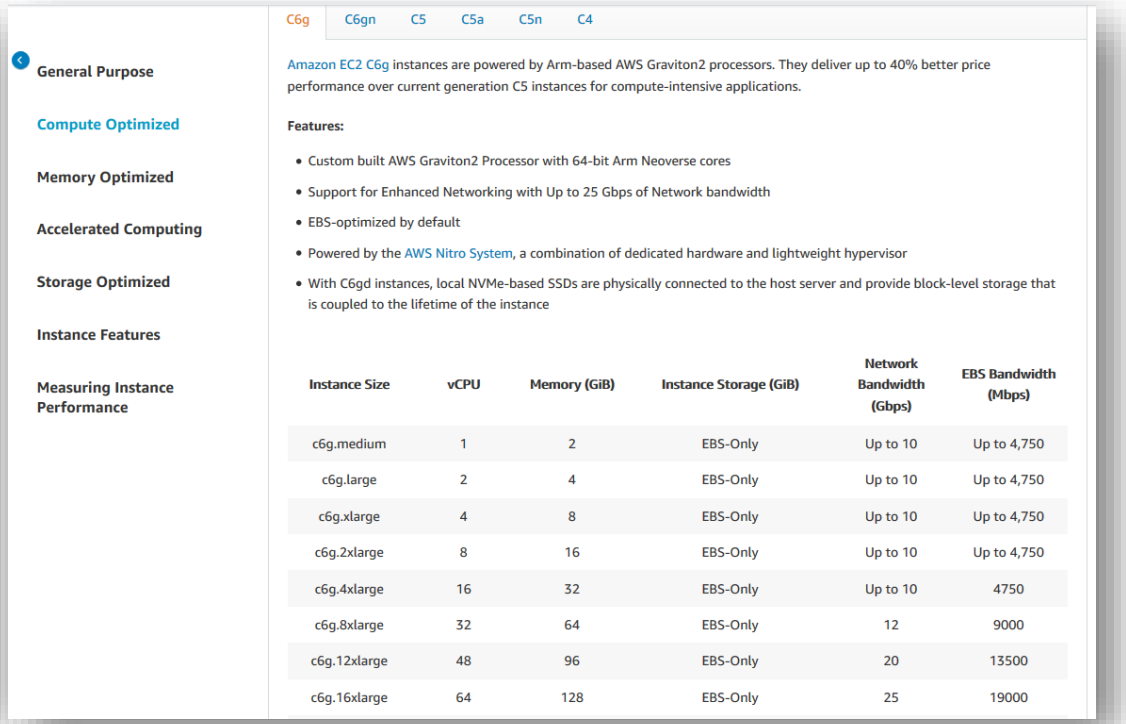
Features:

- 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: AWS EMR

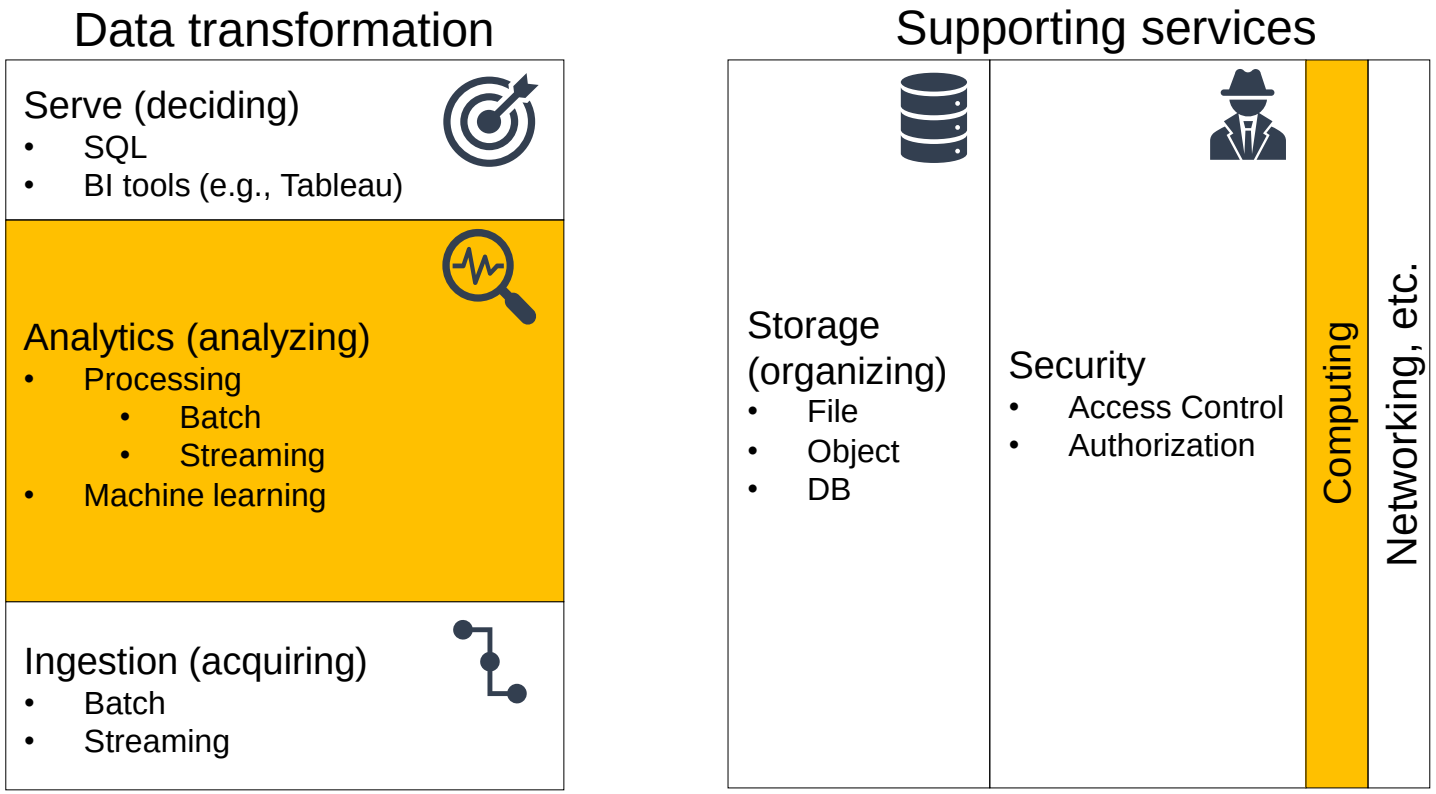
Amazon EMR is a data platform based on the Hadoop stack

- 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



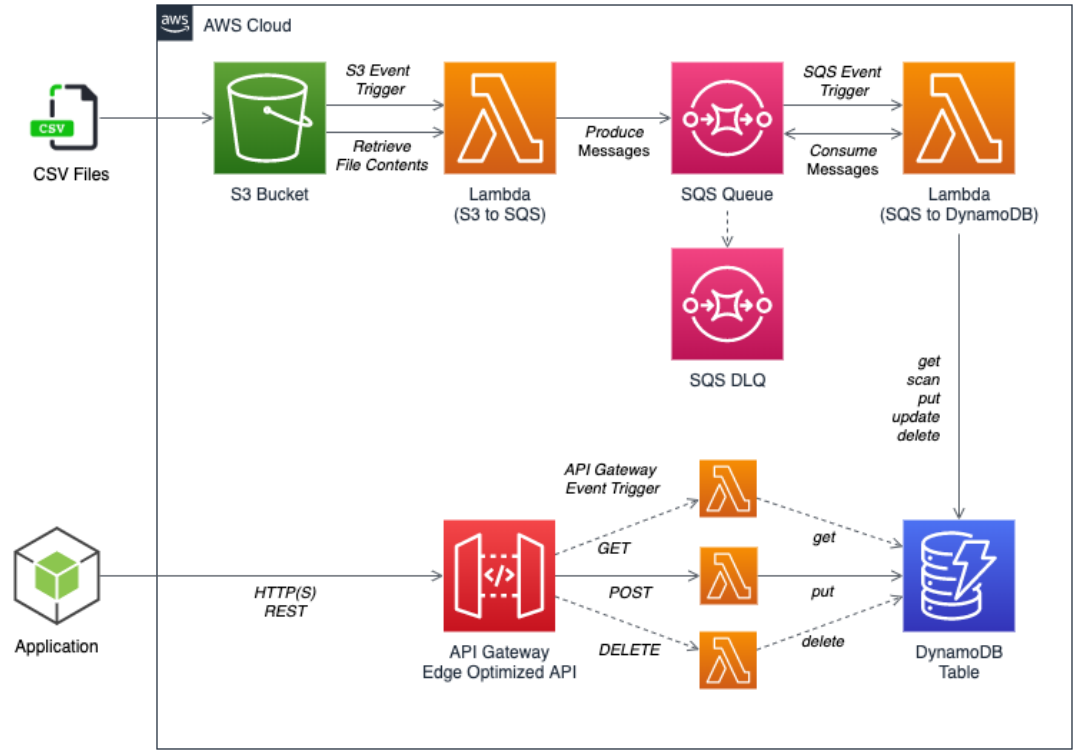
Serverless computing/processing

AWS Lambda: compose code functions in a loose orchestration

- Build modular back-end systems
- Event-driven and push-based pipelines

With Lambda, you are responsible only for your code (i.e., a Lambda function)

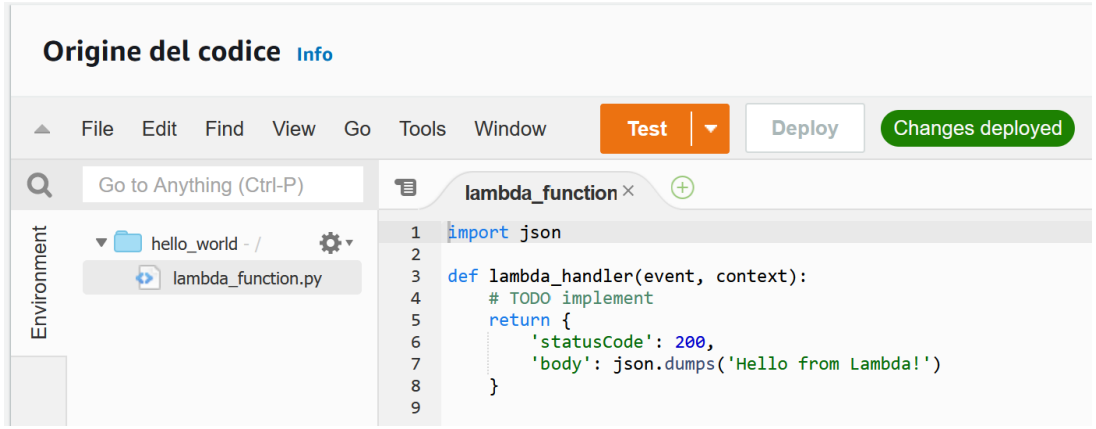
- Lambda manages the compute fleet that offers a balance of memory and CPU
- Lambda performs operational and administrative activities on your behalf
  - Provisioning capacity, monitoring fleet health, applying security patches, deploying your code



Serverless computing (AWS Lambda)

AWS Lambda

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



Serverless computing (FaaS)

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

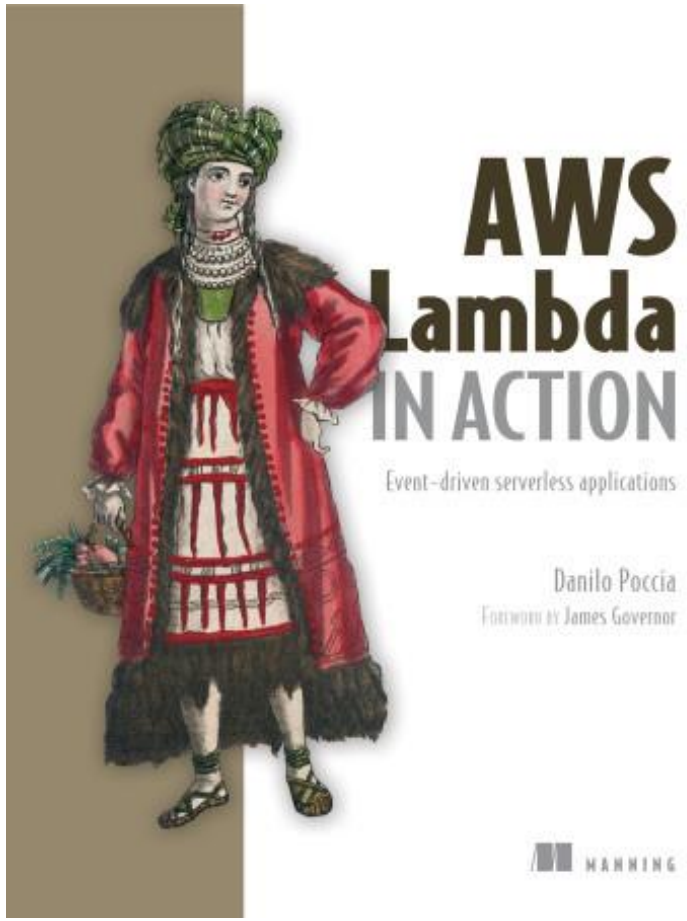
Patterns for data pipelines

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

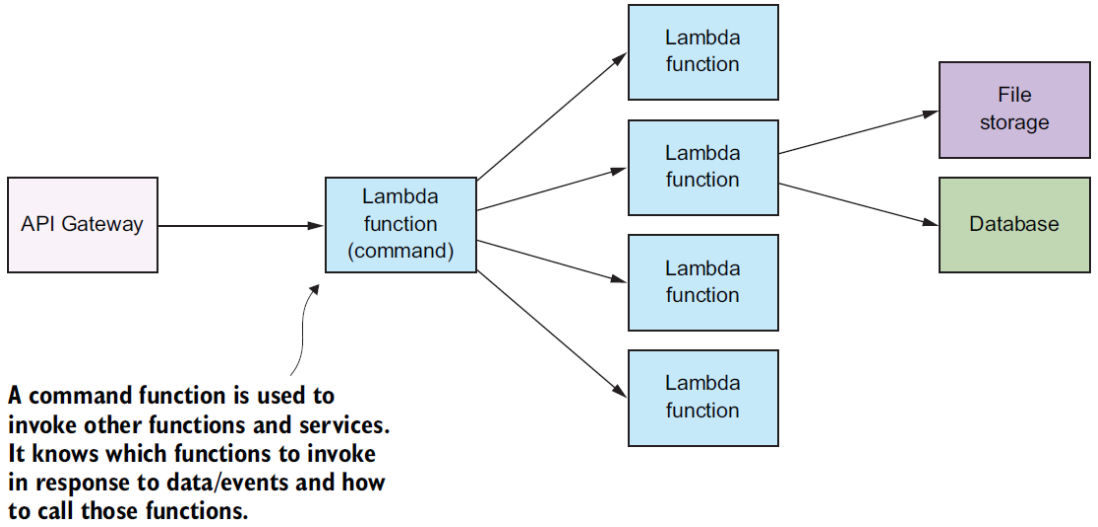
Command pattern

Command pattern

- A behavioral design pattern in which an object is used to encapsulate the 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

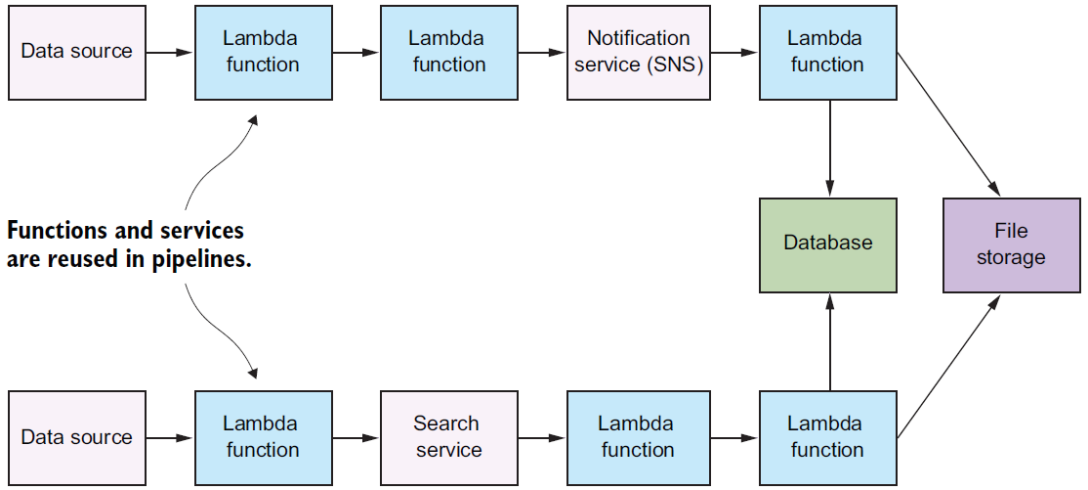


https://aws.amazon.com/api-gateway

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



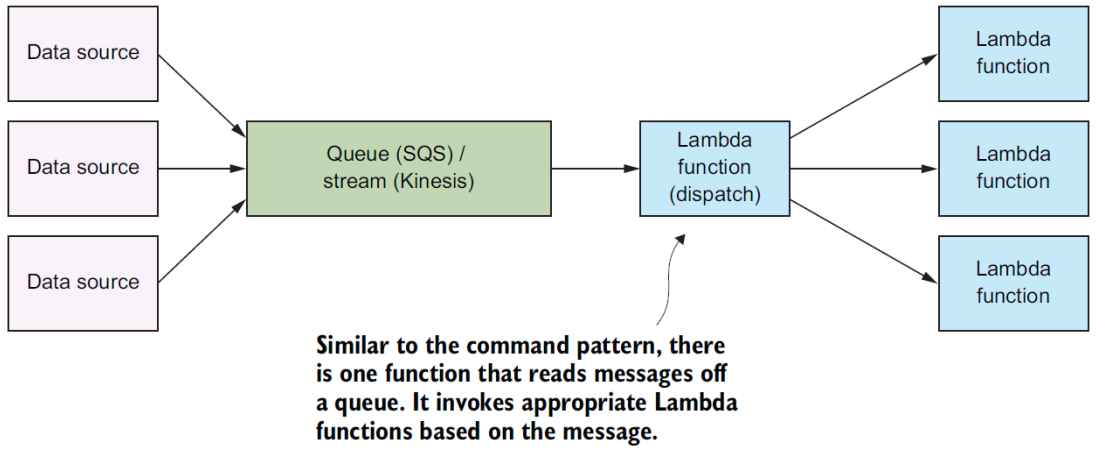
Messaging pattern

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