Aalto University
School of Science

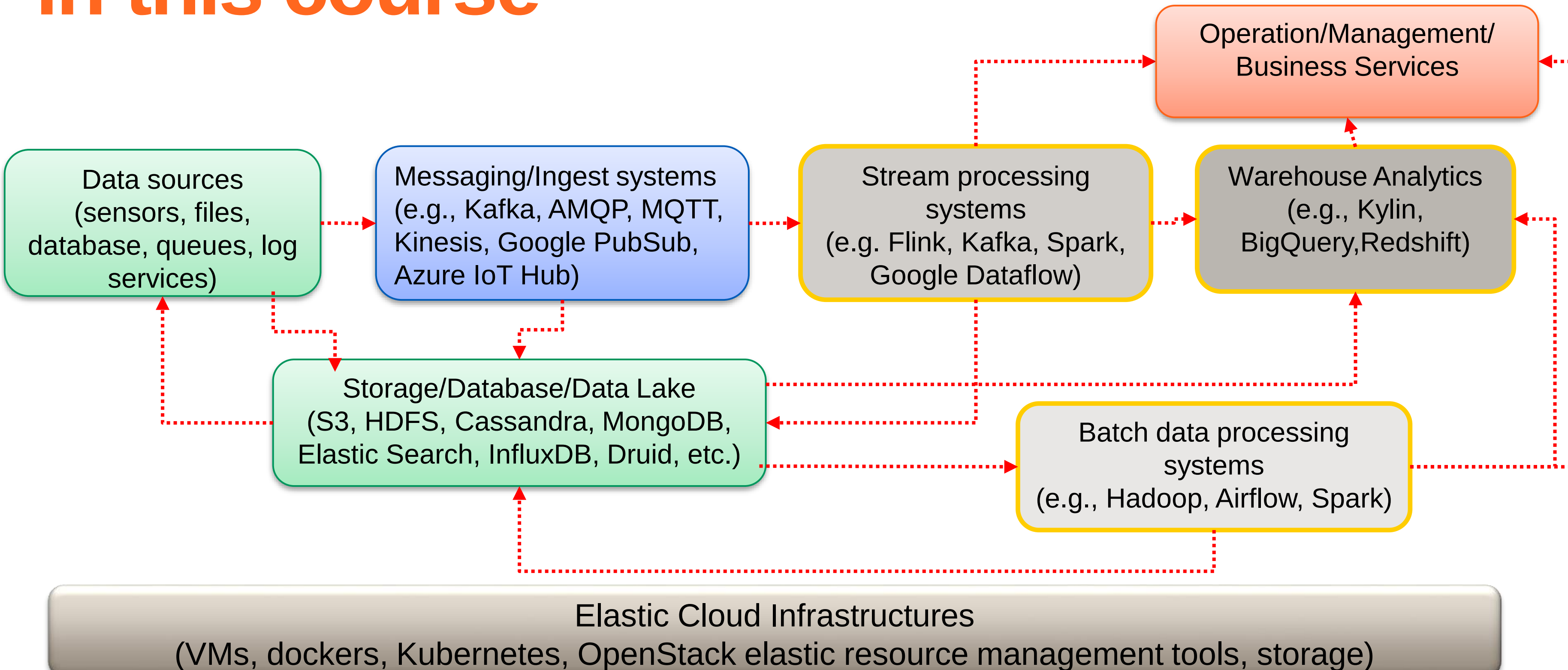
Service and Integration Models in Big Data Platforms

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, *https://rdsea.github.io*

Learning objectives

- **Understand common ways to bring data into platforms**
- **Study service requests and data partition for optimizing integration models**
- **Study MQTT/AMQP for big data platforms**
- **Understand the role of service discovery and consensus**

Big data at large-scale: the big picture in this course



Recall

- **Platforms must facilitate exchanges between many stakeholders centered around data products**
- **Platform services support many types of interactions with different protocols and APIs**
- **Some important aspects of interactions**
  - APIs for encapsulating low-level details
  - protocols for interoperability
  - performance management

Examples of APIs

REST Resource: v2.datasets

Methods	
`delete`	`DELETE /bigquery/v2/projects/{projectId}/datasets/{datasetId}` Deletes the dataset specified by the datasetId value.
`get`	`GET /bigquery/v2/projects/{projectId}/datasets/{datasetId}` Returns the dataset specified by datasetID.
`insert`	`POST /bigquery/v2/projects/{projectId}/datasets` Creates a new empty dataset.
`list`	`GET /bigquery/v2/projects/{projectId}/datasets` Lists all datasets in the specified project to which the user has been granted the READER dataset role.
`patch`	`PATCH /bigquery/v2/projects/{projectId}/datasets/{datasetId}` Updates information in an existing dataset.
`update`	`PUT /bigquery/v2/projects/{projectId}/datasets/{datasetId}` Updates information in an existing dataset.

BigQuery API Client Libraries

Send feedback

This page shows how to get started with the Cloud Client Libraries for the BigQuery API. Read more about the client libraries for Cloud APIs, including the older Google APIs Client Libraries, in Client Libraries Explained.

Installing the client library

C# Go Java Node.js PHP Python Ruby

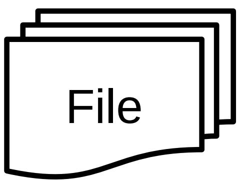
For more information, see Setting Up a C# Development Environment.

```
Install-Package Google.Cloud.BigQuery.V2 -Pre
```

Snapshots from https://cloud.google.com/bigquery/docs/reference/

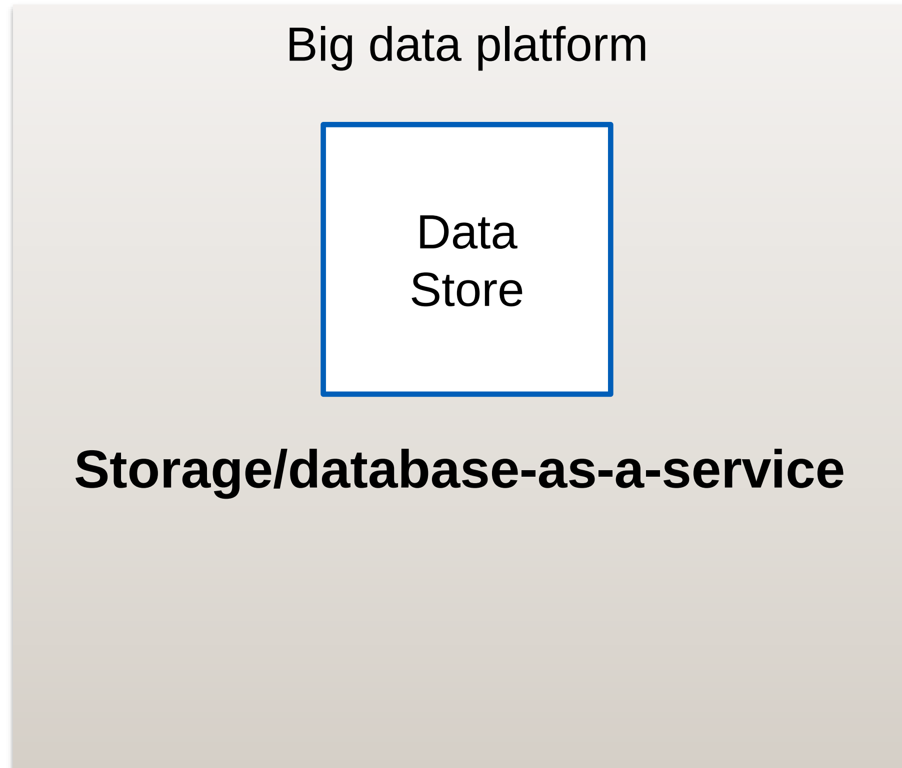
Check other big data platforms: similar approach → **c**ommonly **REST APIs/client libraries** for managing services and for uploading data

Bring big data in files/datasets into platforms

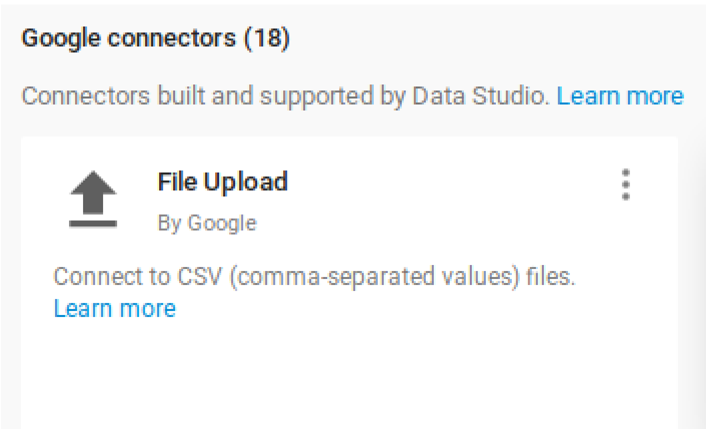


e.g.

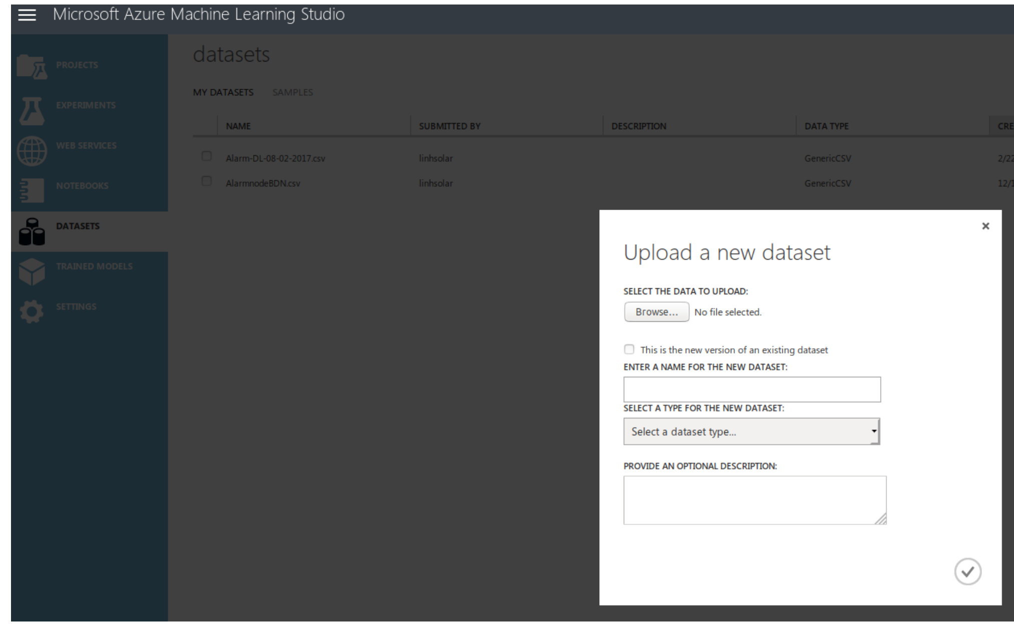
- logs of machines
- sell receipt transaction records
- Images/video



First obstacle: uploading big data into cloud services



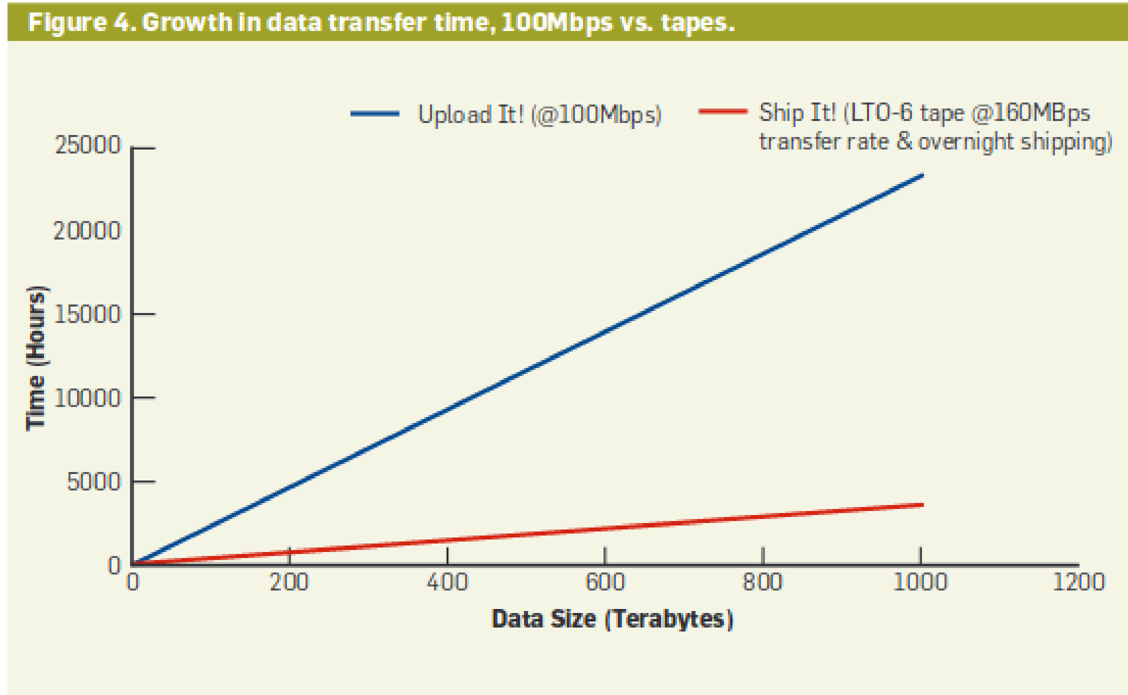
e.g., upload data into the cloud store and run machine learning



Upload or ship big data?

Not upload data in all cases

(assume the uploaded data is correct)



Sachin Date. 2016. Should you upload or ship big data to the cloud?. Commun. ACM 59, 7 (June 2016), 44-51. DOI: https://doi.org/10.1145/2909493

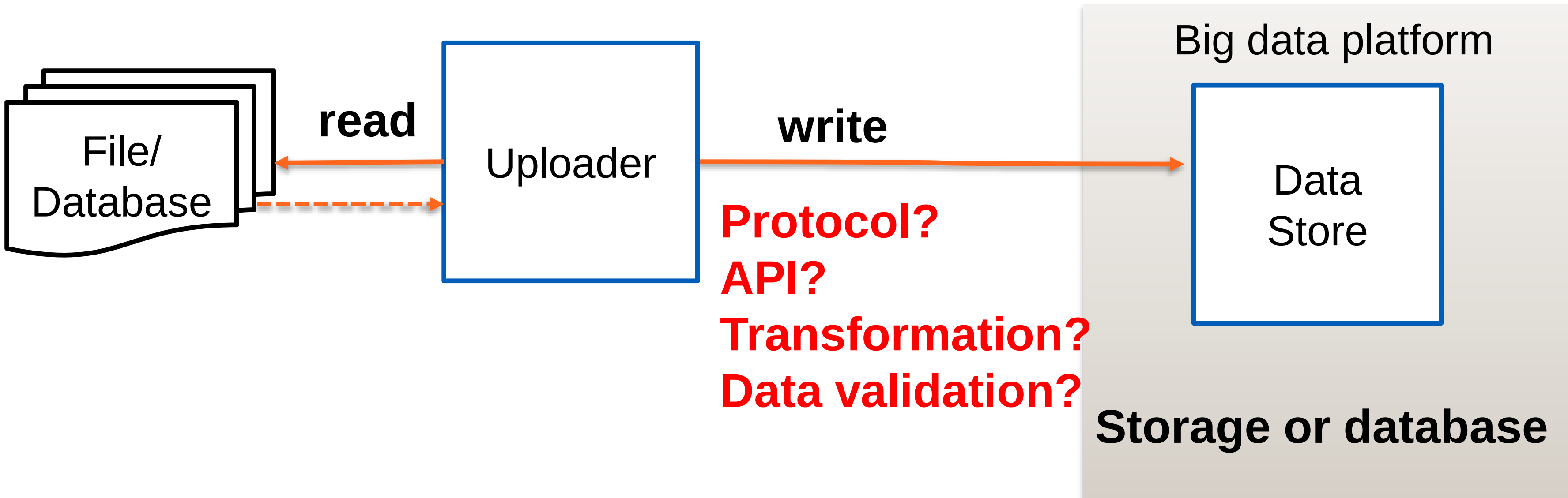
Utilities for big data transfers

Tool	Storage Systems Supported	GUI	Prog. Language	Parallel Transfers	Chunked Uploads
rclone	Multiple (21)	Y	Go 1.6+	Y (param)	Y (param)
cyberduck	Multiple (16)	Y	Java 1.8	Y (param)	N
aws-cli	S3 protocol	N	Python 2.6+	Y (auto)	Y (auto)
gdrive-cli	Google Drive	N	Go 1.5+	N	Y (param)
azure-cli	Azure	N	Python 2.7+	Y (param)	Y (auto)
dbox-cli	Dropbox	N	Go	N	Y (16 MB)

Source: Sergio Rivera, James Griffioen, Zongming Fei, Mami Hayashida, Pinyi Shi, Bhushan Chitre, Jacob Chappell, Yongwook Song, Lowell Pike, Charles Carpenter, and Hussamuddin Nasir. 2018. **Navigating the Unexpected Realities of Big Data Transfers in a Cloud-based World**. In Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18). ACM, New York, NY, USA, Article 22, 8 pages. DOI: https://doi.org/10.1145/3219104.3229276

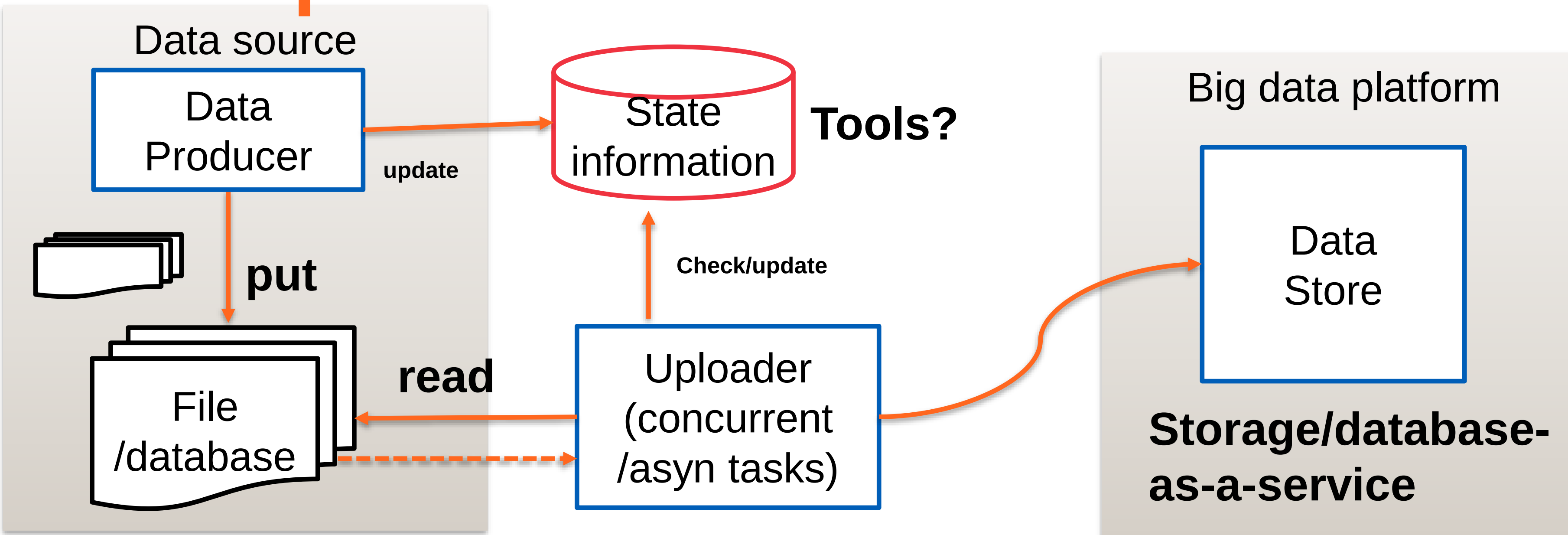
But you may need to design your own utilities? Why? Any ideas?

If you are going to design uploader



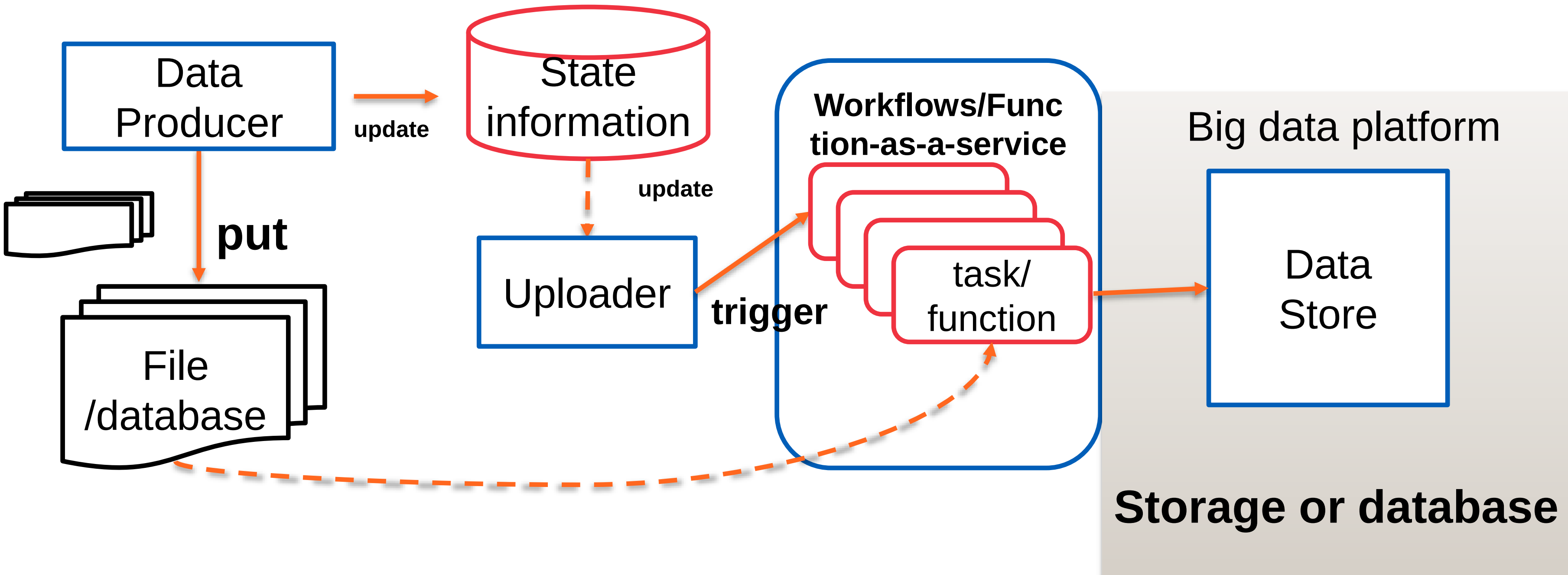
- Practical issues for optimization:
  - What if you have very big files? Or a lot of small files?
  - Any ideas about possible techniques?

How does the data producer inform data uploader



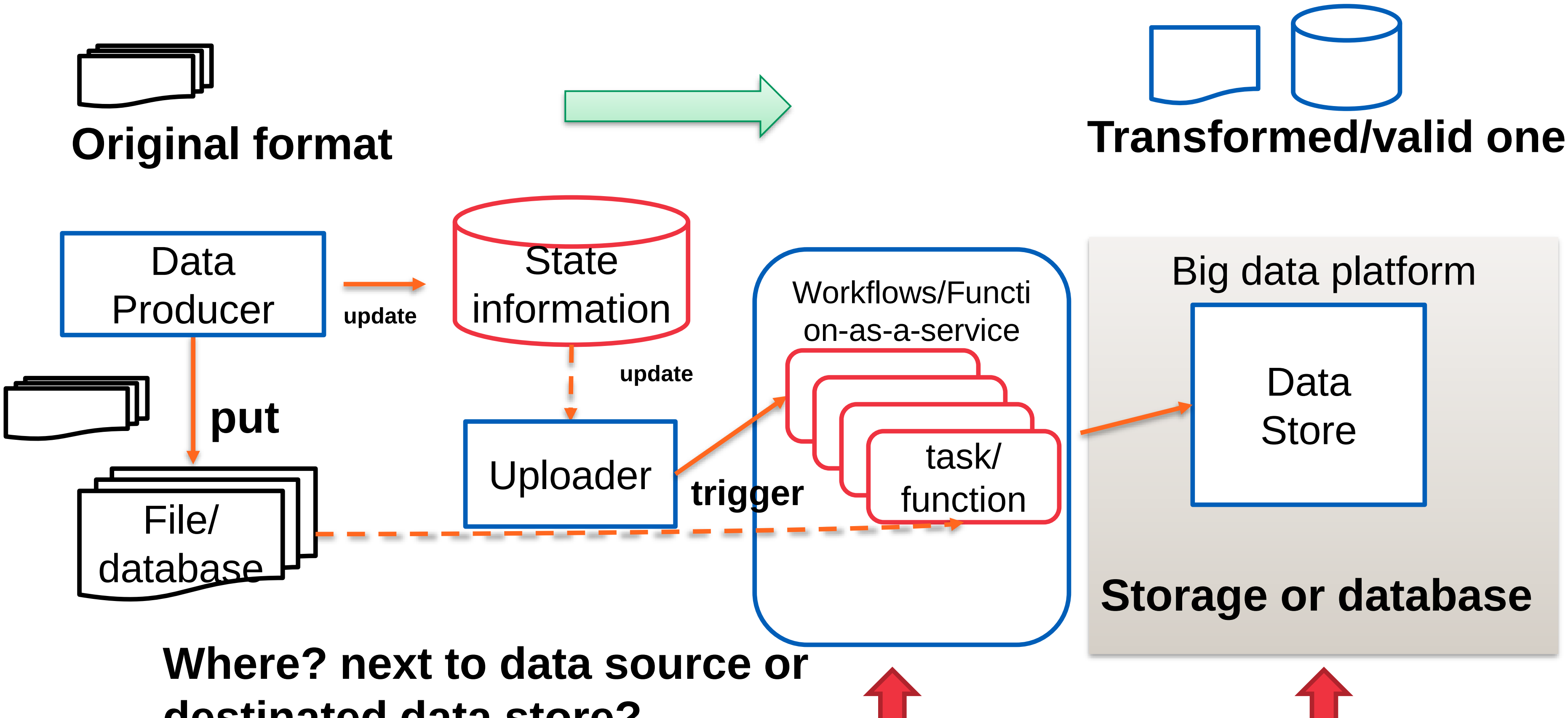
Run as a service
Must know if new files arrive: state management with which techniques?

Uploader as a “scheduler”/”coordinator”



Benefit?

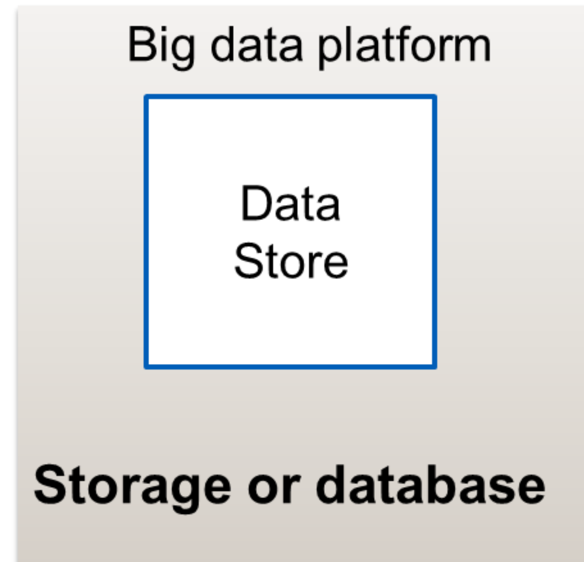
If the transformation/validation is needed?



Where? next to data source or destinated data store?

Integrate streaming data sources into platforms

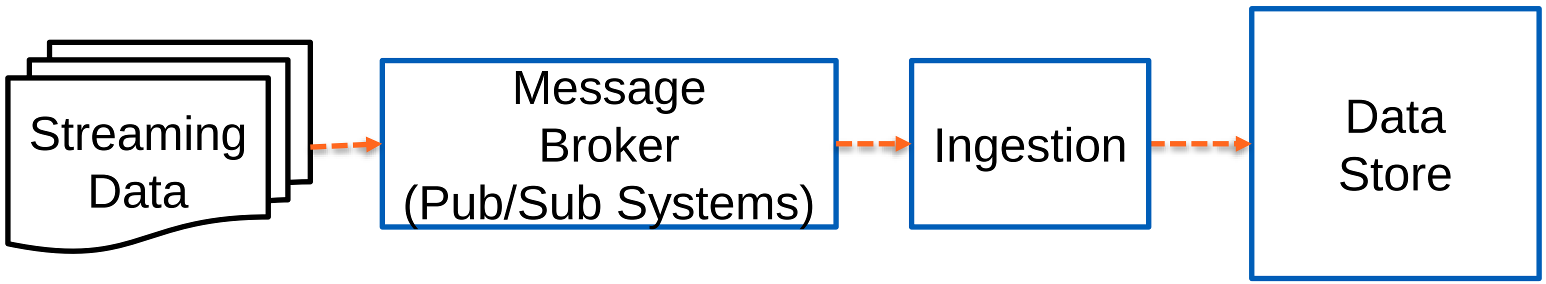
Streaming protocols/frameworks



Recall:

"A big data platform monitors network usage of devices from million+ customers. We have different levels: Sensor/Customer, Node (concentrator of multiple customers), Agent (concentrator of multiple Nodes) and the whole network. In a region, the real operator can generate 1.4 billion records per day ~ 72GB per day"

How do I move streaming data into the cloud?



Protocol?
Data format
Message structure

Real-world technologies

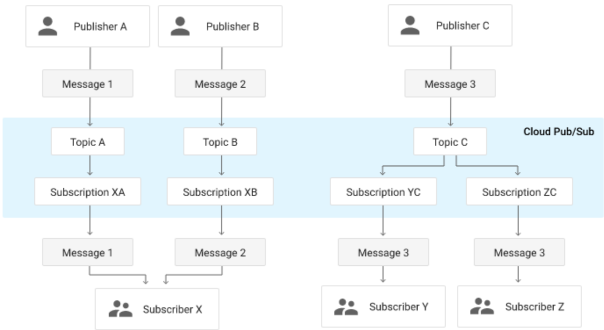


Figure source: https://cloud.google.com/pubsub/docs/overview

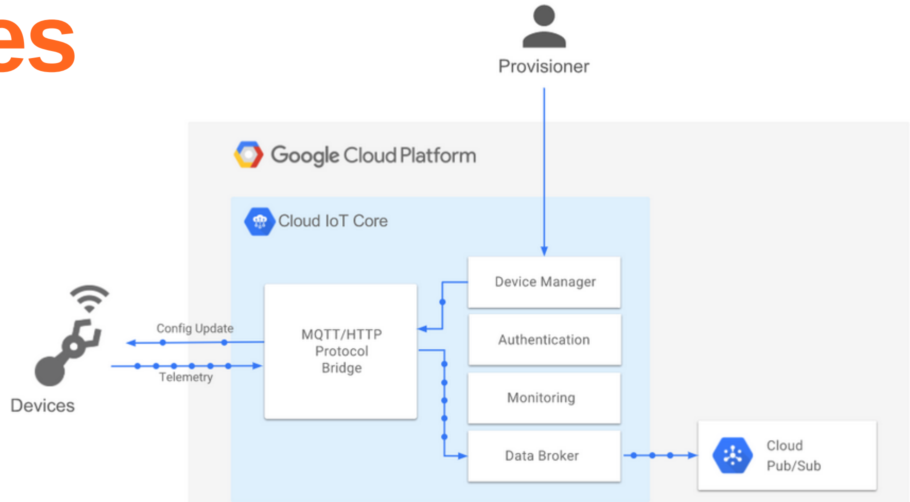


Figure source: https://cloud.google.com/iot/docs/concepts/overview

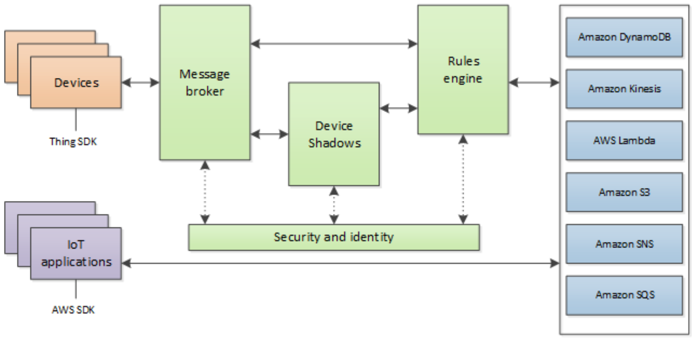


Figure source: https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html

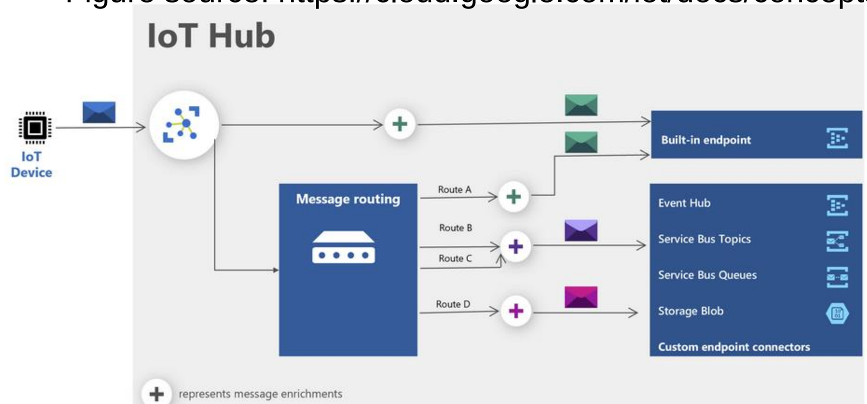


Figure source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-message-enrichments-overview

Do you see common concepts/terms?

Some important protocols

- **Protocols**
  - AMQP, MQTT
- **Systems**
  - Apache Kafka, Apache RocketMQ
- **Distinguish between “protocols” and “specific frameworks”**
  - How would they affect your design?

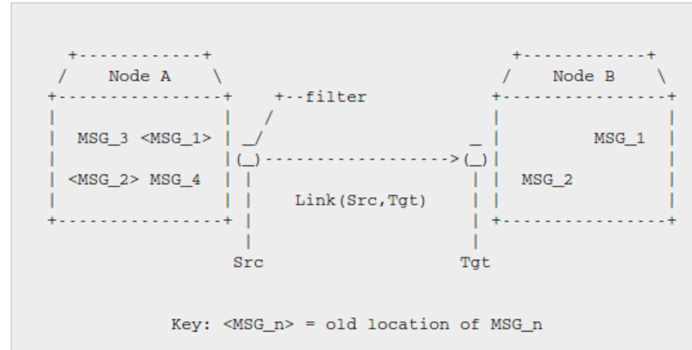
MQTT & AMQP protocols

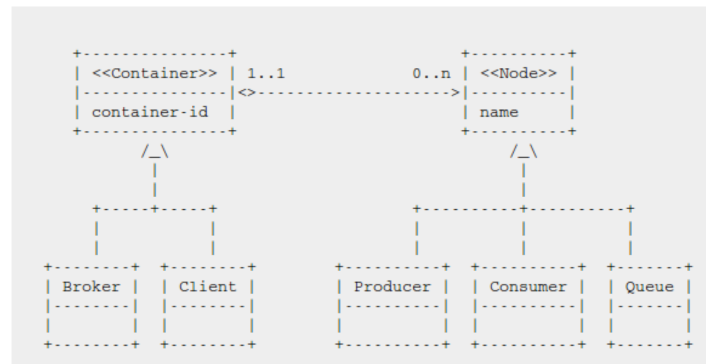
AMQP - Overview

- **Protocol for message-oriented middleware**
  - Not language- or platform- specific
  - For Java, C#, Python, ….
- **Binary wire-level protocol for message exchange, rather than APIs**
- **http://www.amqp.org**

Core concepts – Message/Transport

- **Message representation**
  - Defined based on type systems for interoperability
- **Transport**
  - A network of nodes connected via links
  - Node: message storage, delivery, relay, etc.
  - Container: includes nodes





Figs source: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

Example

- **Get a free instance of RabbitMQ from cloudamqp.com**
  - Or deploy your own RabbitMQ
- **Get some examples from https://www.rabbitmq.com/getstarted.html**

Performance

- **“RabbitMQ Hits One Million Messages Per Second on Google Compute Engine”**
  - https://blog.pivotal.io/pivotal/products/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine
  - https://cloudplatform.googleblog.com/2014/06/rabbitmq-on-google-compute-engine.html
  - Using 32 nodes
- **RabbitMQ is widely used in industries!**

MQTT Overview

- **http://mqtt.org**
- **OASIS Standard**
- **ISO/IEC 20922:2016 (Message Queuing Telemetry Transport (MQTT) v3.1.1)**
- **M2M Connectivity Protocol atop TCP/IP**
- **MQTT brokers enable publish/subscribe messaging systems**
  - Publisher can publish a messge within a topic that can be subscribed by many Subscribers

MQTT Protocol Features

- **Lightweight protocol**
  - Small message size
  - QoS: At most once, at least once and exactly once
  - Few commands/interactions: CONNECT, PUBLISH, SUBSCRIBE, UNSUBRIBE, DISCONNECT
    - *Easy to implement*
- **Small foot-print libary**
- **Low bandwidth, high latency, data limits, and fragile connections**
- **Suitable for IoT (constrained devices/networks)**

Model and Implementation

- **Different programming languages for OS/devices**
- **Implementation example**
  - Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto)

    docker pull eclipse-mosquitto
  - Paho: http://www.eclipse.org/paho/
  - RabbitMQ
  - Cloud providers:
    - *http://cloudmqtt.com (get a free account to learn MQTT)*

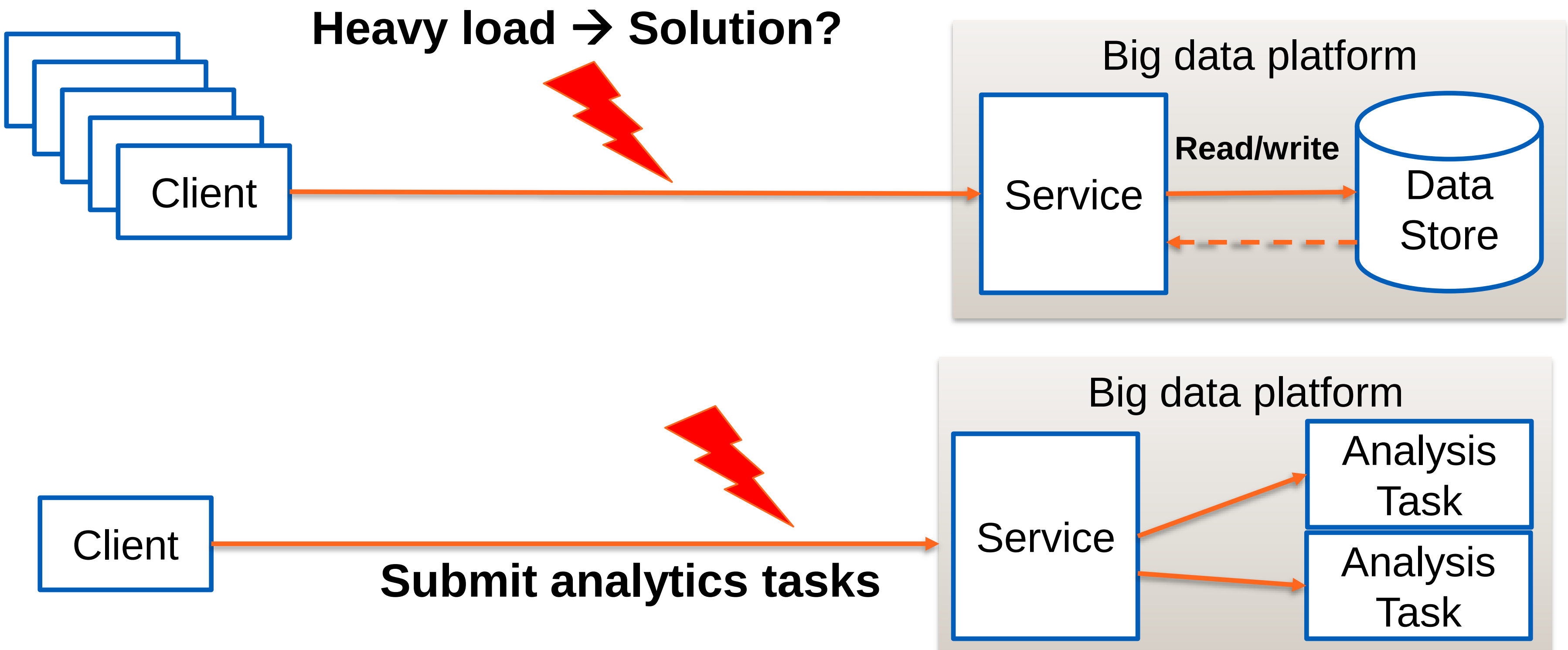
So which one you think is suitable for this?

"A big data platform monitors network usage of devices from million+ customers. We have different levels: Sensor/Customer, Node (concentrator of multiple customers), Agent (concentrator of multiple Nodes) and the whole network. In a region, the real operator can generate 1.4 billion records per day ~ 72GB per day"

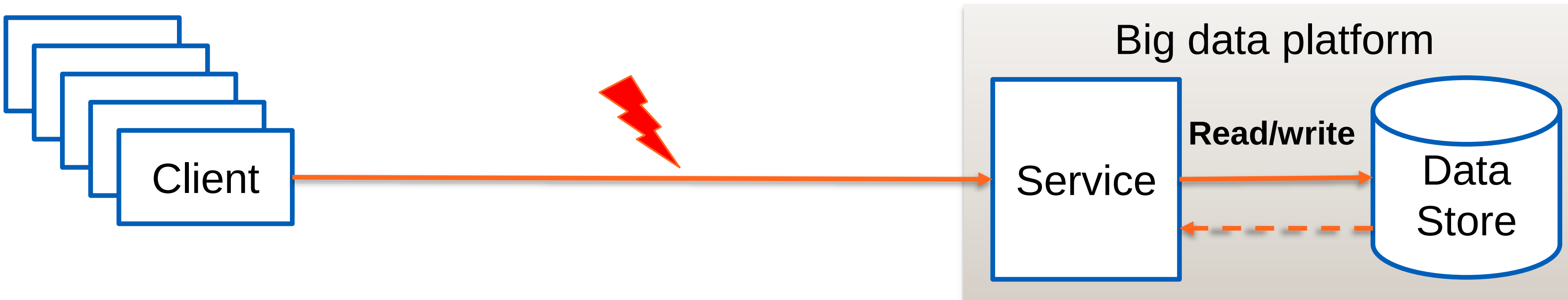
Vote: https://presemo.aalto.fi/bdp

Optimize service requests and functionalities

Concurrent contention

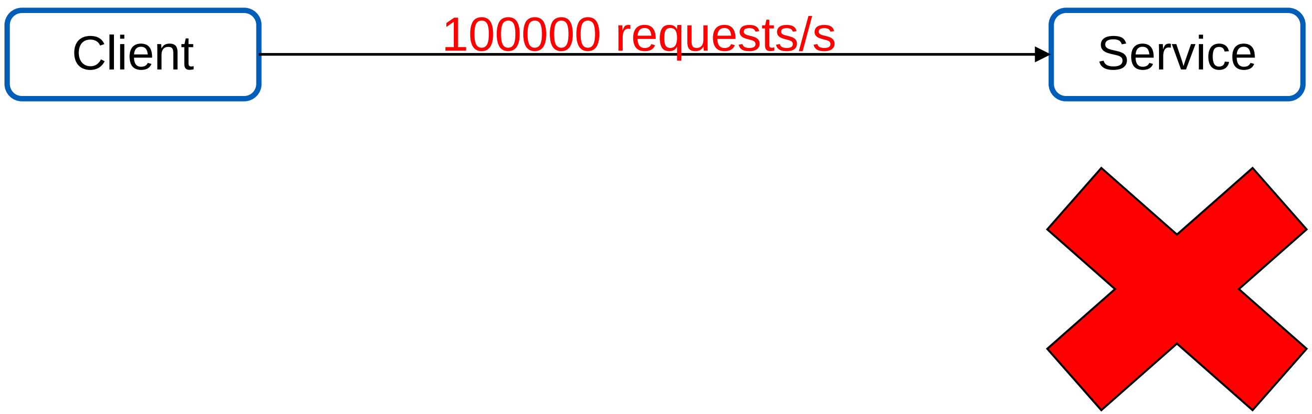


Back-pressure or elasticity



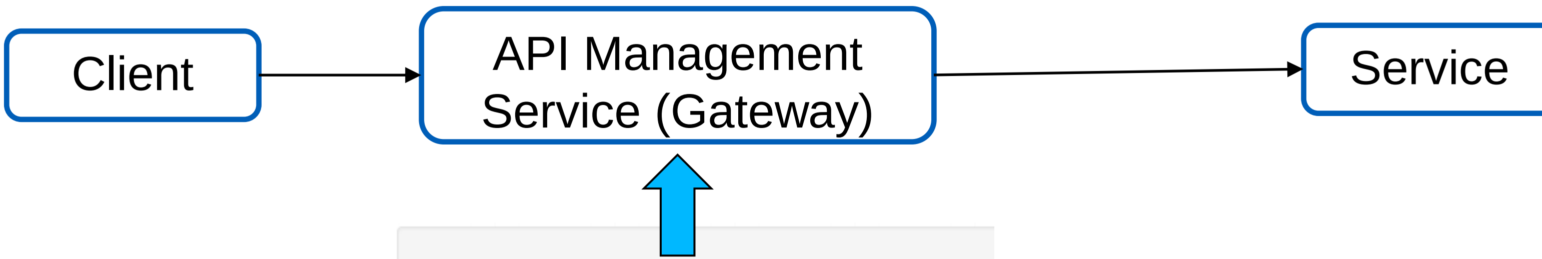
Back-pressure: control, drop, and buffer

Prevent too many accesses?



Throttling

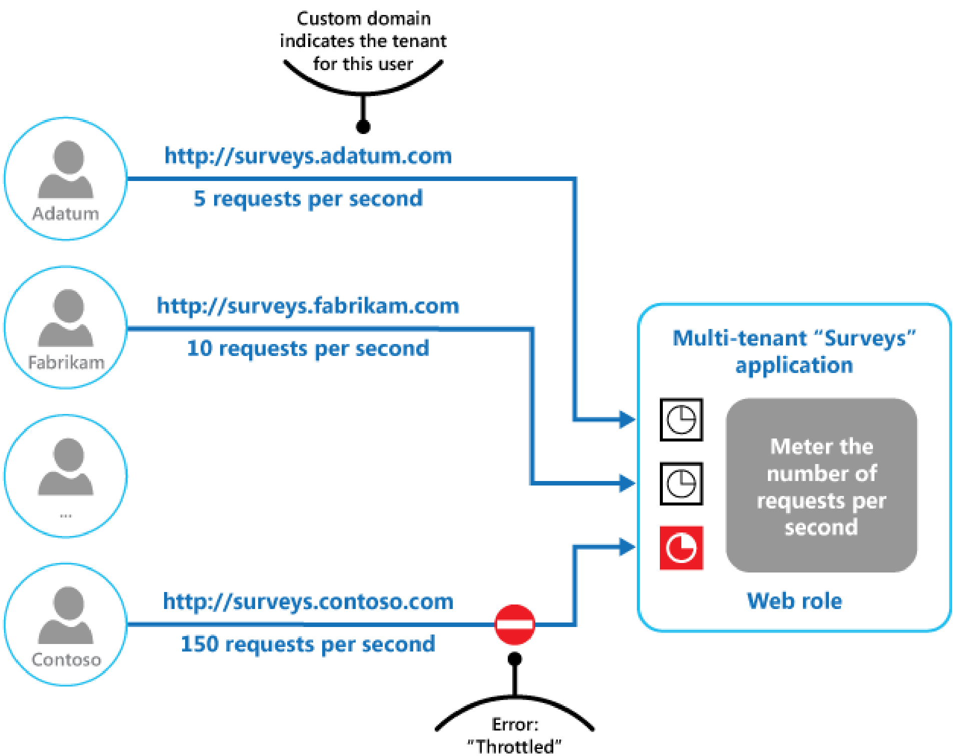
- **Drop strategy: Disable too many access and disable unessential services**
- **E.g., using API Gateway Kong, Kubernetes**



Code: http://www.django-rest-framework.org/api-guide/throttling/#how-throttling-is-determined

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': (
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle'
    ),
    'DEFAULT_THROTTLE_RATES': {
        'anon': '100/day',
        'user': '1000/day'
    }
}
```

Example of throttling based on roles

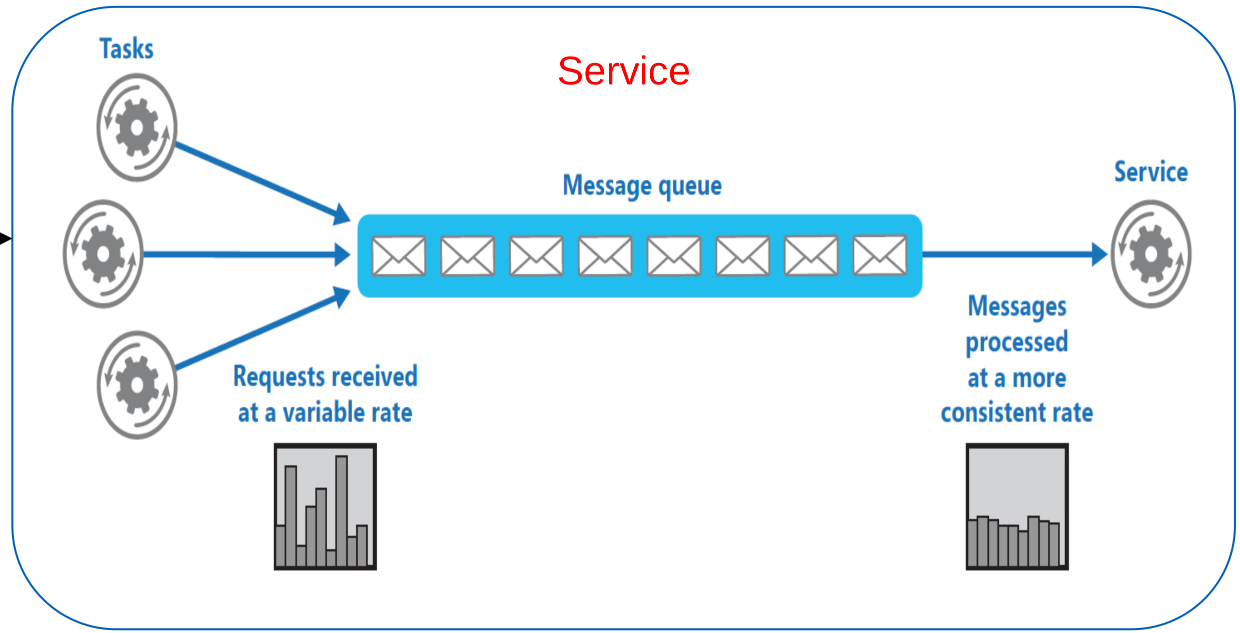


Source: https://msdn.microsoft.com/en-us/library/dn589798.aspx

How this related to your "business service models"/SLA for your platform?

Using tasks and queue-based load leveling pattern

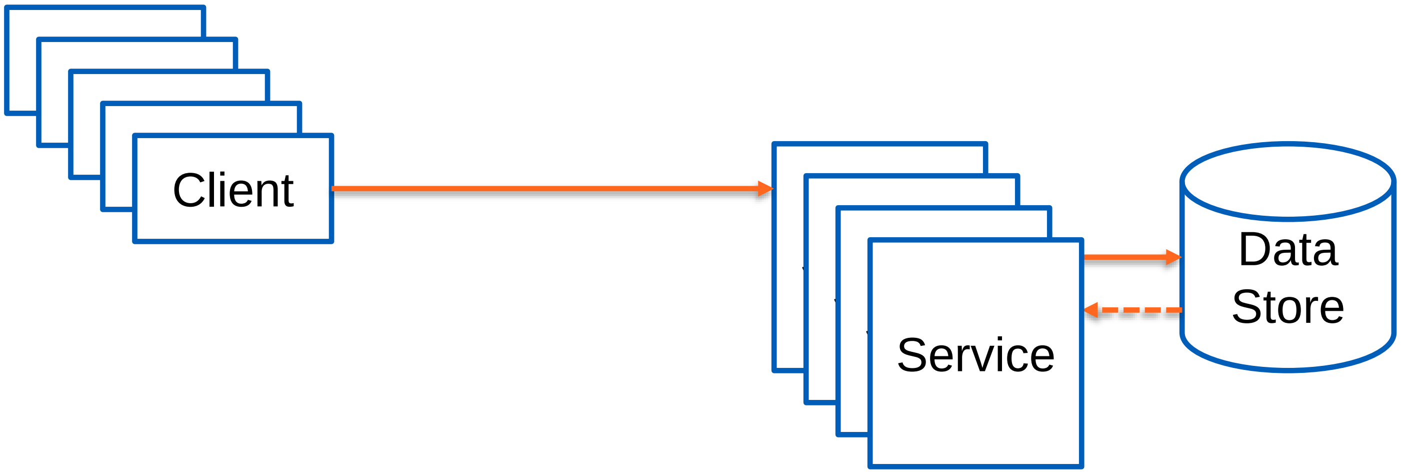
Client



How this affects the internal design of your big service?

Figure source: https://msdn.microsoft.com/en-us/library/dn589783.aspx

Heavy load between service serving request and data store



Elastic solution: scale out or up

Using multiple instances of services and queues

How do we control these instances in an efficient way?

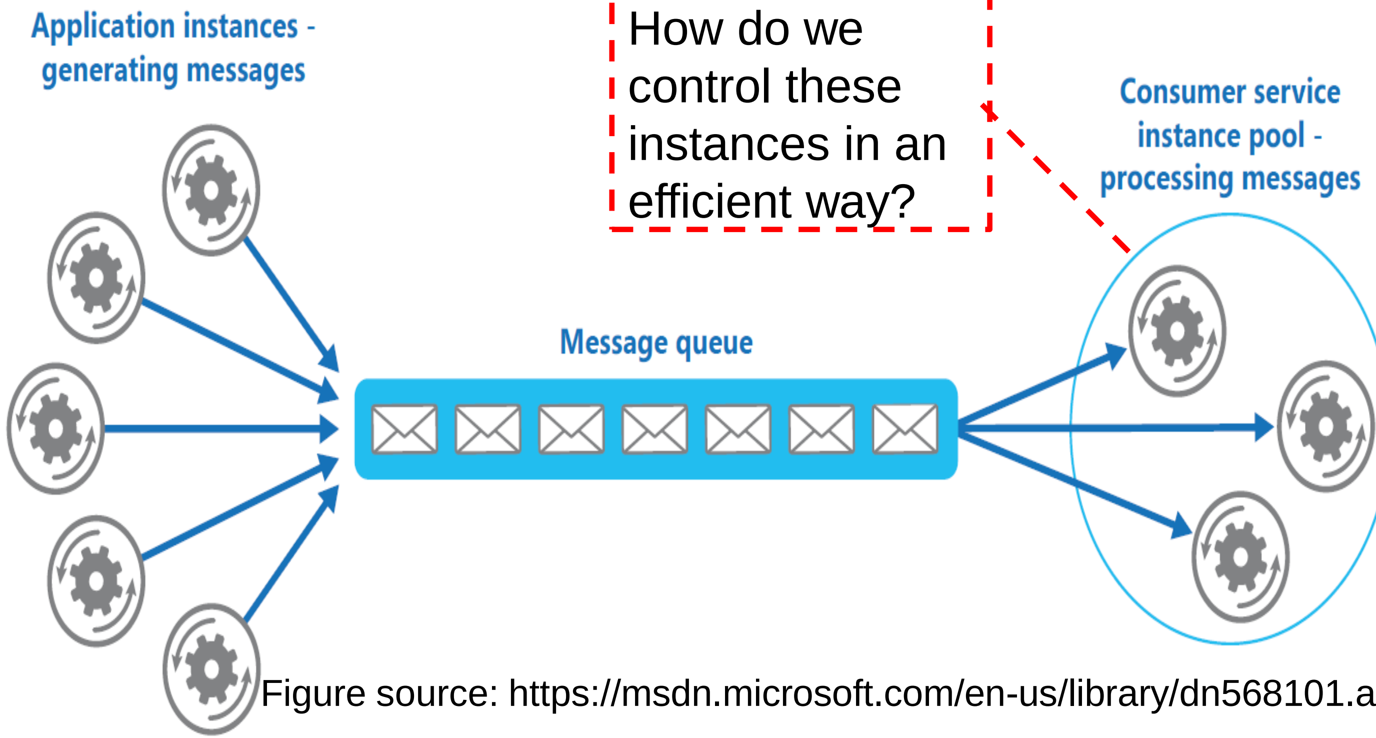
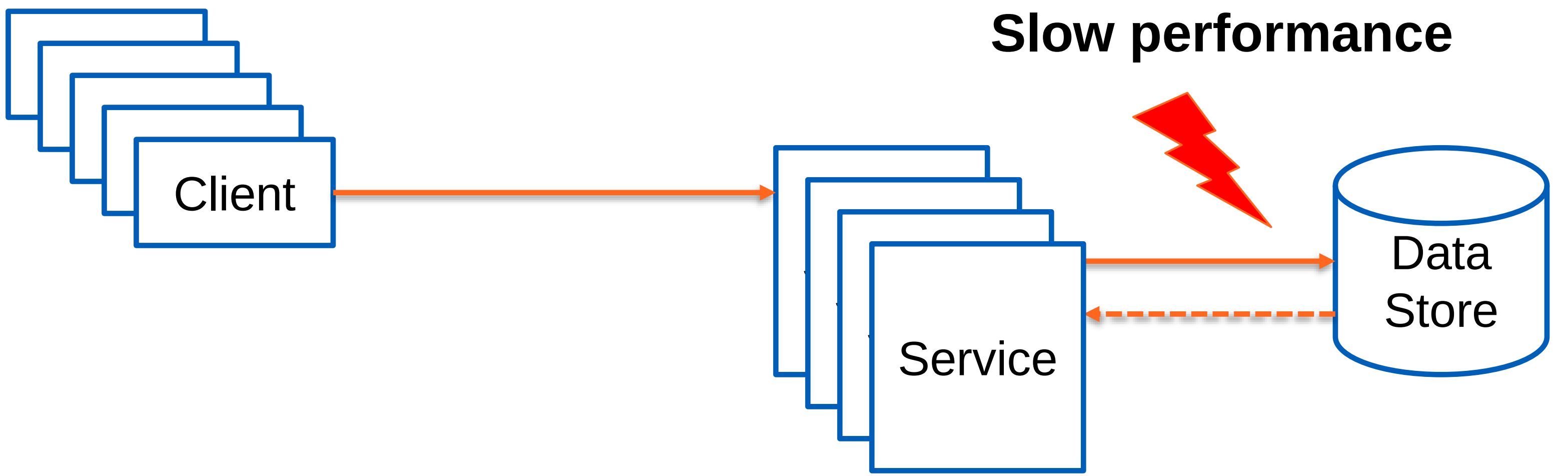


Figure source: https://msdn.microsoft.com/en-us/library/dn568101.aspx

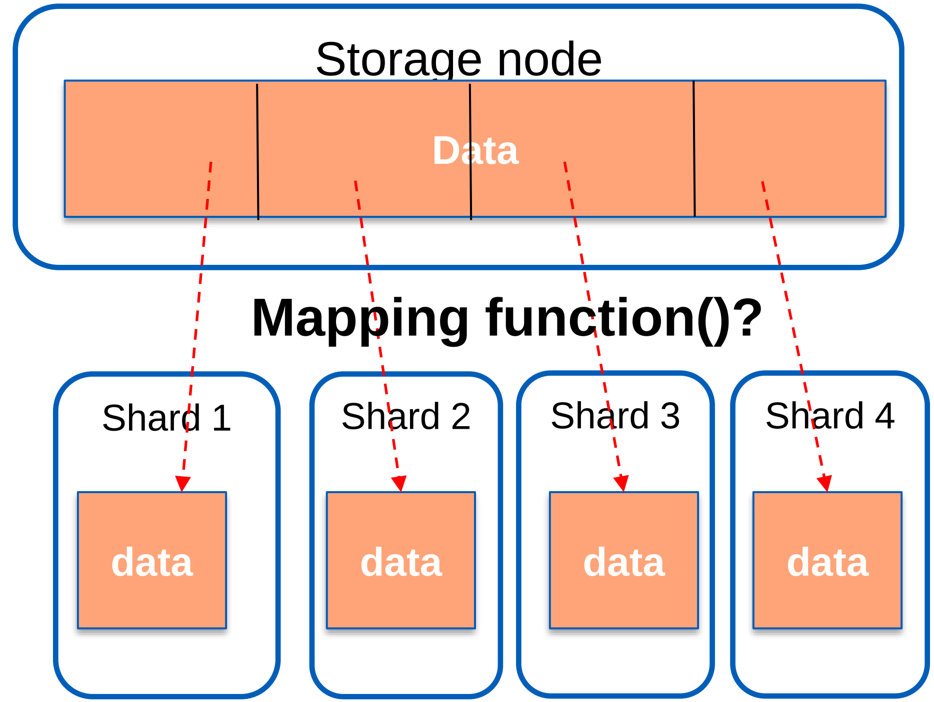
Performance problems between service serving request and data store



- **Big data grows → Data explosion**
- **Concurrent contention, slow read, and slow query**

Principles

- **Partitioning data into different partitions/shards**
- **Making shards in different nodes → Shared nothing, horizonal scaling!**



Sharding Strategies

Key principles

- Determine partitioning attributes associated with data
- Each shard (where the data is stored) has a shard key mapped to partition attributes

Different strategies

- Directory/Lookup: query partitioning attributes based on a lookup table to find a shard
- Range: partitioning attributes are arranged into a range, each shard is responsible for a subrange
- Hash: determined a shard based on the hash of partitioning keys

Sharding patterns/strategies: https://msdn.microsoft.com/en-us/library/dn589797.aspx

Example of strategies in MongoDB

Hash

Range

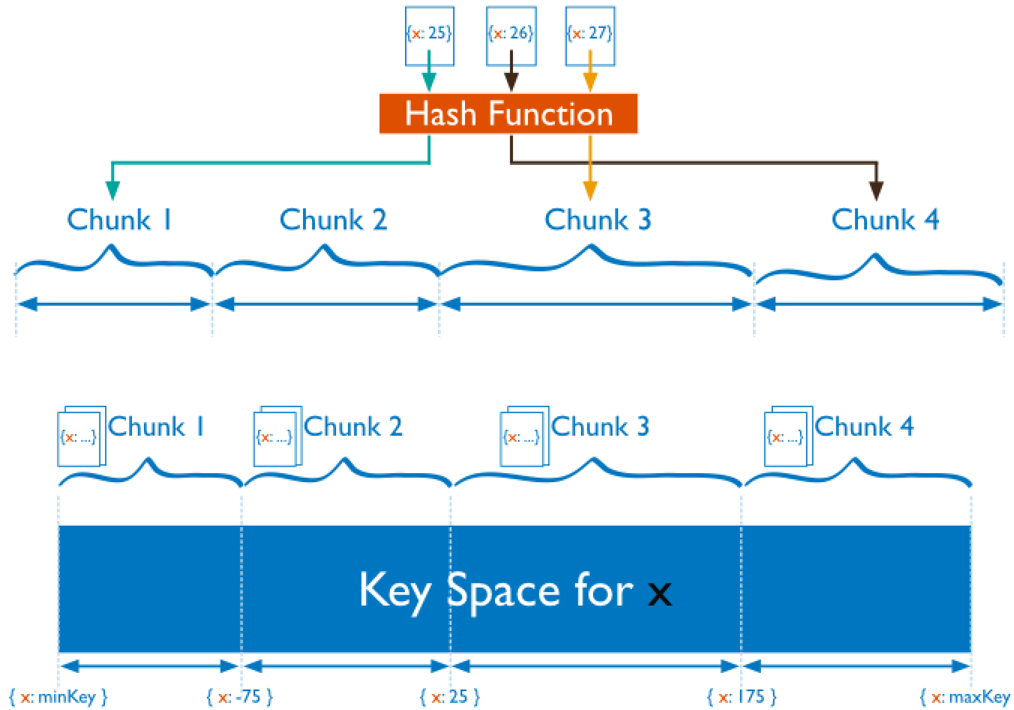
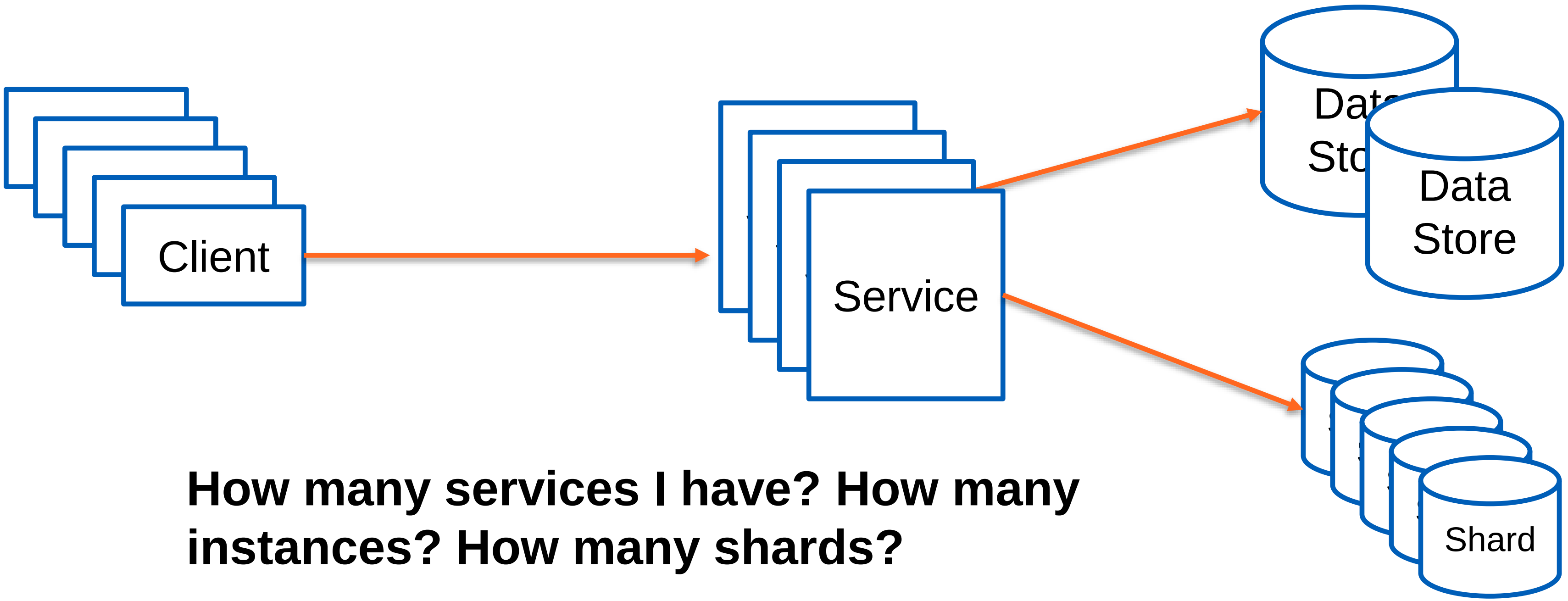


Figure Source: https://docs.mongodb.com/manual/sharding/

Discovery and consensus

We can create a lot of instances or we can create new services

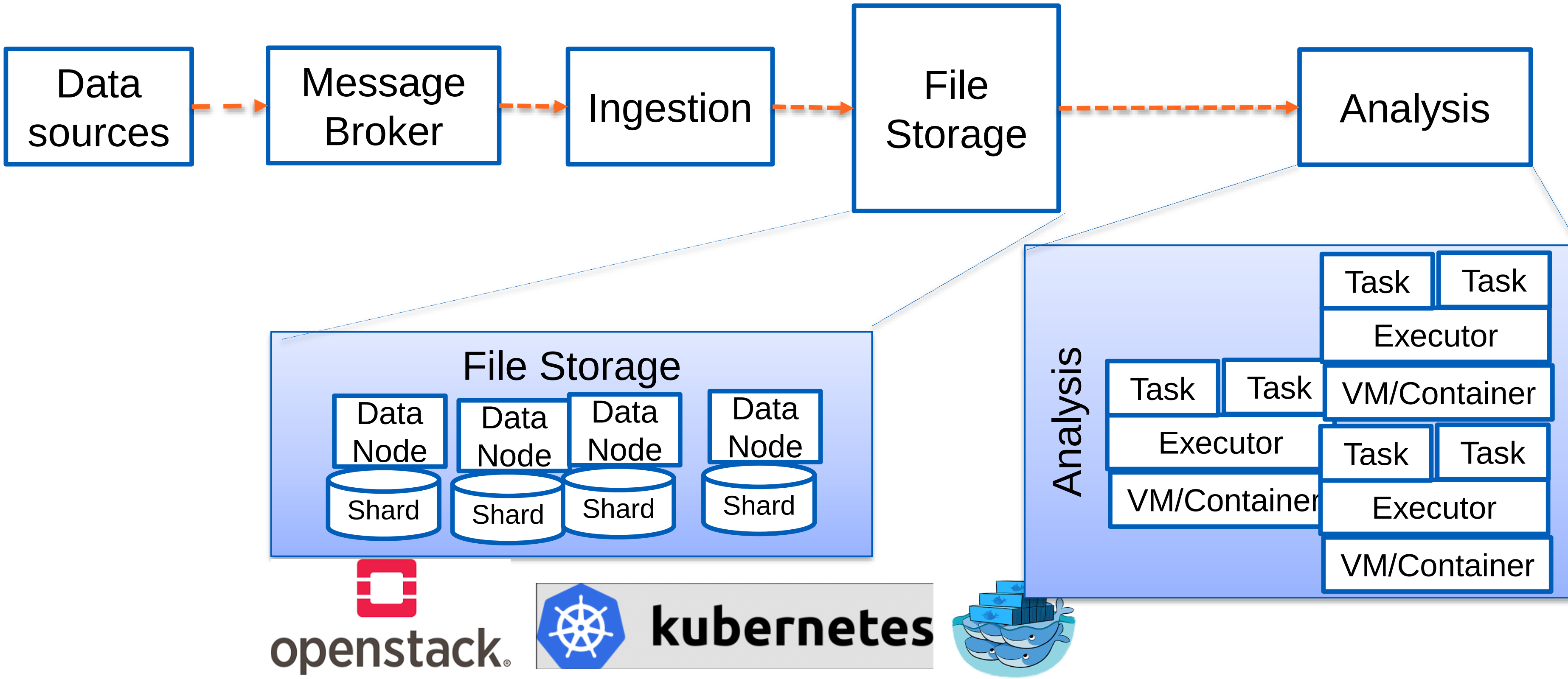


How many services I have? How many instances? How many shards?

Multiple instances

- **A component of big data platforms can have many services and a service can have instances**
  - E.g., for replication and load balancing
  - A database component (e.g. MongoDB) has multiple data nodes, each is a service responsible for a shard/partition
  - A processing engine (e.g., Spark or Airflow) can have many nodes, each executes different tasks of a process
- **The same component can have many deployments**
  - E.g., dedicated deployment of MongoDB for different customers

Runtime view of some components

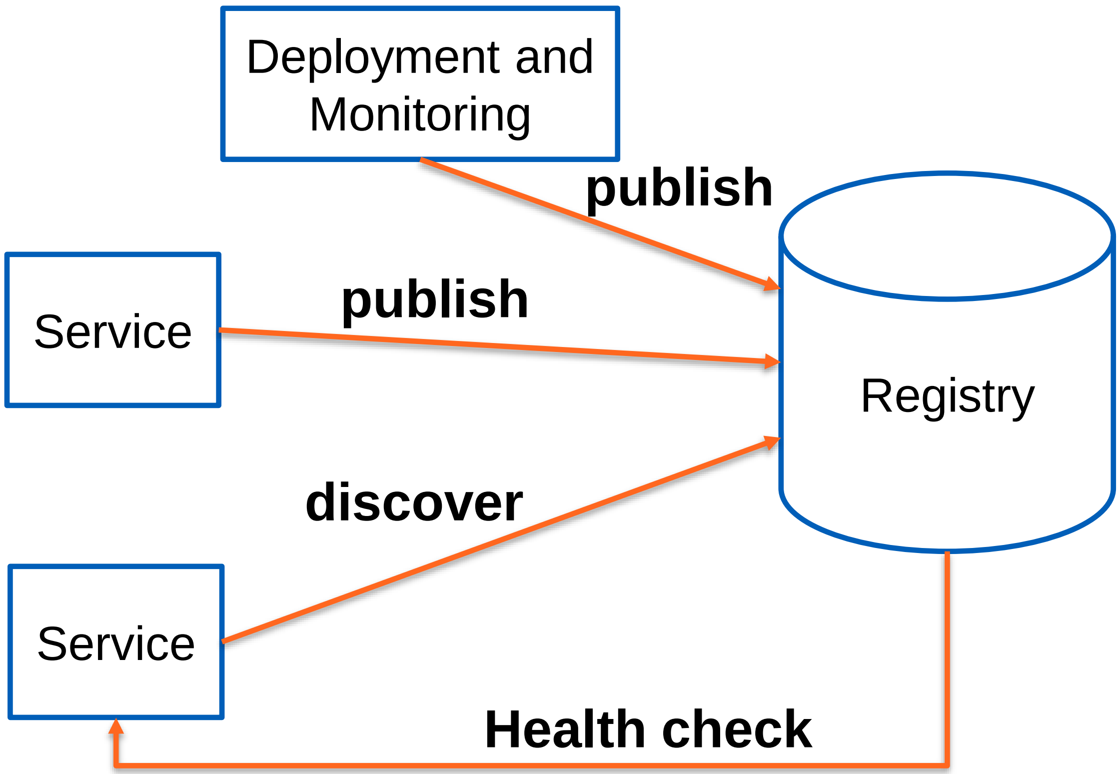


Service state management

- **Service information**
  - Include states and other important configuration information
  - Many instances
  - Cross different infrastructures/data centers
- **Related components**
  - Services themselves
  - Monitoring component, Deployment component, orchestration controllers
- **Lifecycle: very dynamic in elastic environments**
  - *Start, run, shutdown, restart, scale*

Why is it important to know the state of services and what we can do with that?

Service Discovery



- **Key requirements**
  - Fast
  - Consistent
  - Secure
  - Cross data centers
  - Simple APIs

Example:

https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/examples/servicediscovery

Consensus for big data platforms

- **Consensus is about to agree on something**
- **Very important for replication and fault tolerance in big data platforms**
  - Distributed lock, master selection
- **Scope**
  - Platform level and service component levels
  - Single data center or cross-data center
- **We will have to deal with them in several frameworks for big data, e.g. Apache Spark, Hadoop and Kafka**

Distributed Coordination

- **A lot of algorithms, etc.**
  - Paxos family
- **Well-known in the cloud**

Notes from the paper: “server replication (SR), log replication (LR), synchronization service (SS), barrier orchestration (BO), service discovery (SD), group membership (GM), leader election (LE), metadata management (MM) and distributed queues (Q)”

TABLE 4. PATTERNS OF PAXOS USE IN PROJECTS

		Usage Patterns								
Project	**Consensus System**	**SR**	**LR**	**SS**	**BO**	**SD**	**GM**	**LE**	**MM**	**Q**
GFS	Chubby			✓				✓	✓	
Borg	Chubby/Paxos	✓				✓		✓		
Kubernetes	etcd						✓		✓	
Megastore	Paxos		✓							
Spanner	Paxos	✓								
Bigtable	Chubby						✓	✓	✓	
Hadoop/HDFS	ZooKeeper	✓						✓		
HBase	ZooKeeper	✓		✓			✓		✓	
Hive	ZooKeeper			✓					✓	
Configerator	Zeus								✓	
Cassandra	ZooKeeper					✓		✓	✓	
Accumulo	ZooKeeper		✓	✓					✓	
BookKeeper	ZooKeeper						✓		✓	
Hedwig	ZooKeeper						✓		✓	
Kafka	ZooKeeper						✓	✓	✓	
Solr	ZooKeeper							✓	✓	✓
Giraph	ZooKeeper		✓		✓				✓	
Hama	ZooKeeper				✓					
Mesos	ZooKeeper							✓		
CoreOS	etcd					✓				
OpenStack	ZooKeeper					✓				
Neo4j	ZooKeeper			✓				✓		

What if they do not fit into your big data platforms?

Source: Ailidani Ailijiang, Aleksey Charapkoy and Murat Demirbasz , Consensus in the Cloud: Paxos Systems Demystified, http://www.cse.buffalo.edu/tech-reports/2016-02.pdf

Technology choices: ZooKeeper

- **https://zookeeper.apache.org/**
- **Support service discovery, configuration information and distributed synchronization**
- **Centralized registry service**
- **Data is organized into a shared hierarchical name space**
  - Small data size
- **Highly available and reliable**

ZooKeeper Service

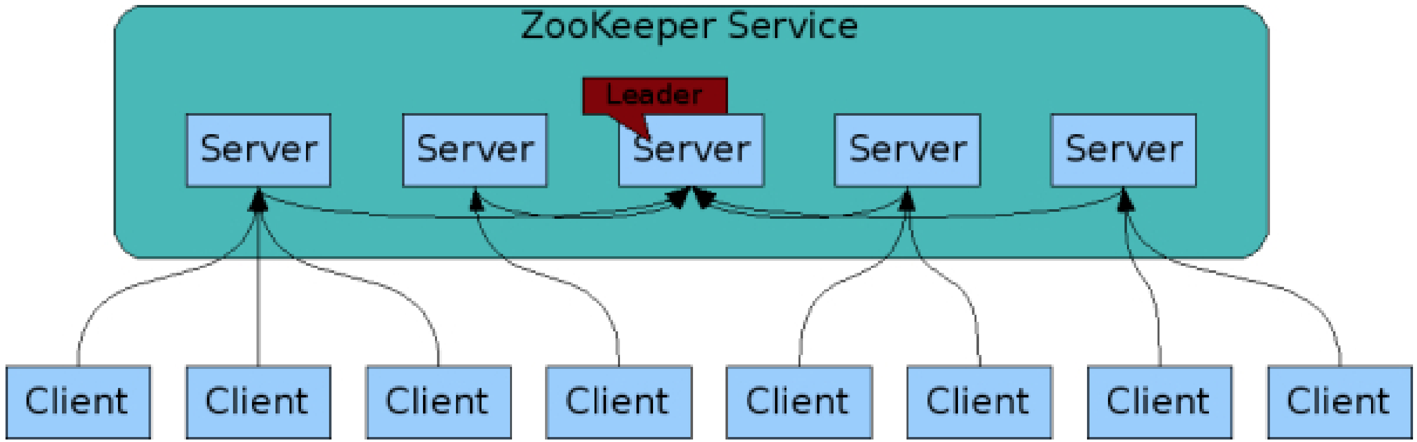


Figure source: https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html

ZooKeeper data -- znodes

- **Data nodes called znodes**
- **Missing data in a znode → problems with the entity that the znode represents**
- **Persistent znode**
  - /path deleted only through a delete call
- **Ephemeral znode, deleted when**
  - The client created it crashed
  - Session expired

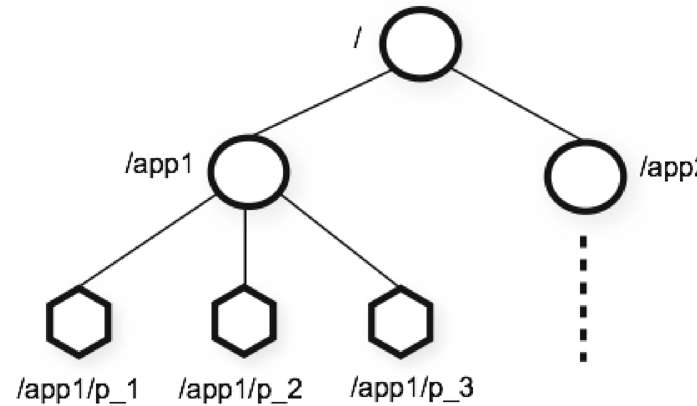


Figure source: https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html

Technology choices: Consul

- **https://www.consul.io**
- **Cross data centers**
- **End-to-end service discovery**
  - Include health check

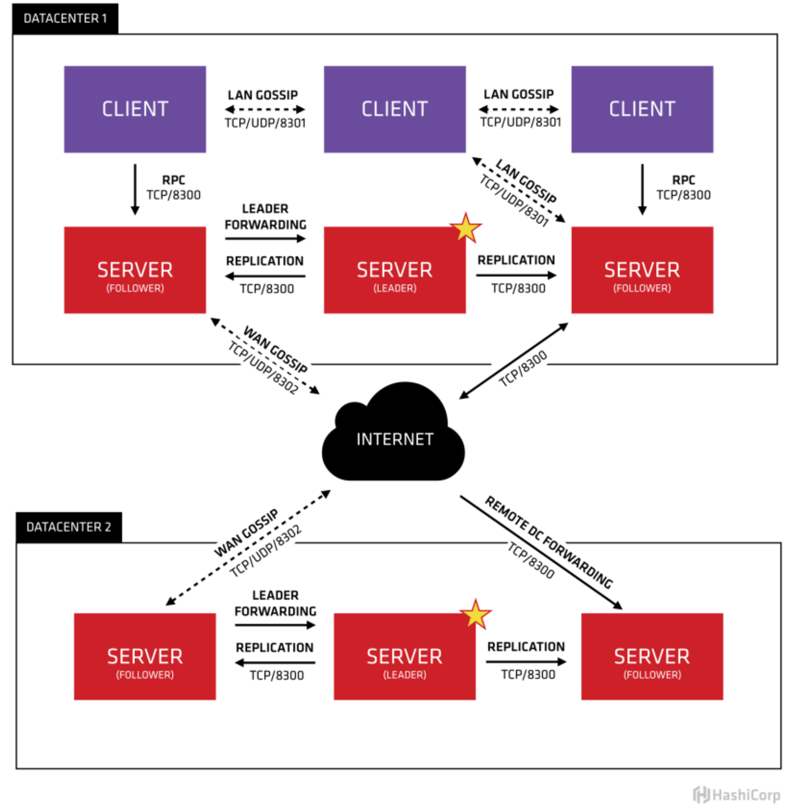


Figure source: https://www.consul.io/docs/internals/architecture.html

Technology choices: etcd

- **Consistent, distributed key-value store**
- **Allow monitor changes of keys/directories**
  - enable reactive actions based on changes
- **Widely used for**
  - service discovery and state/configuration management
  - distributed key locking
  - e.g. in Kubernetes

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io