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*

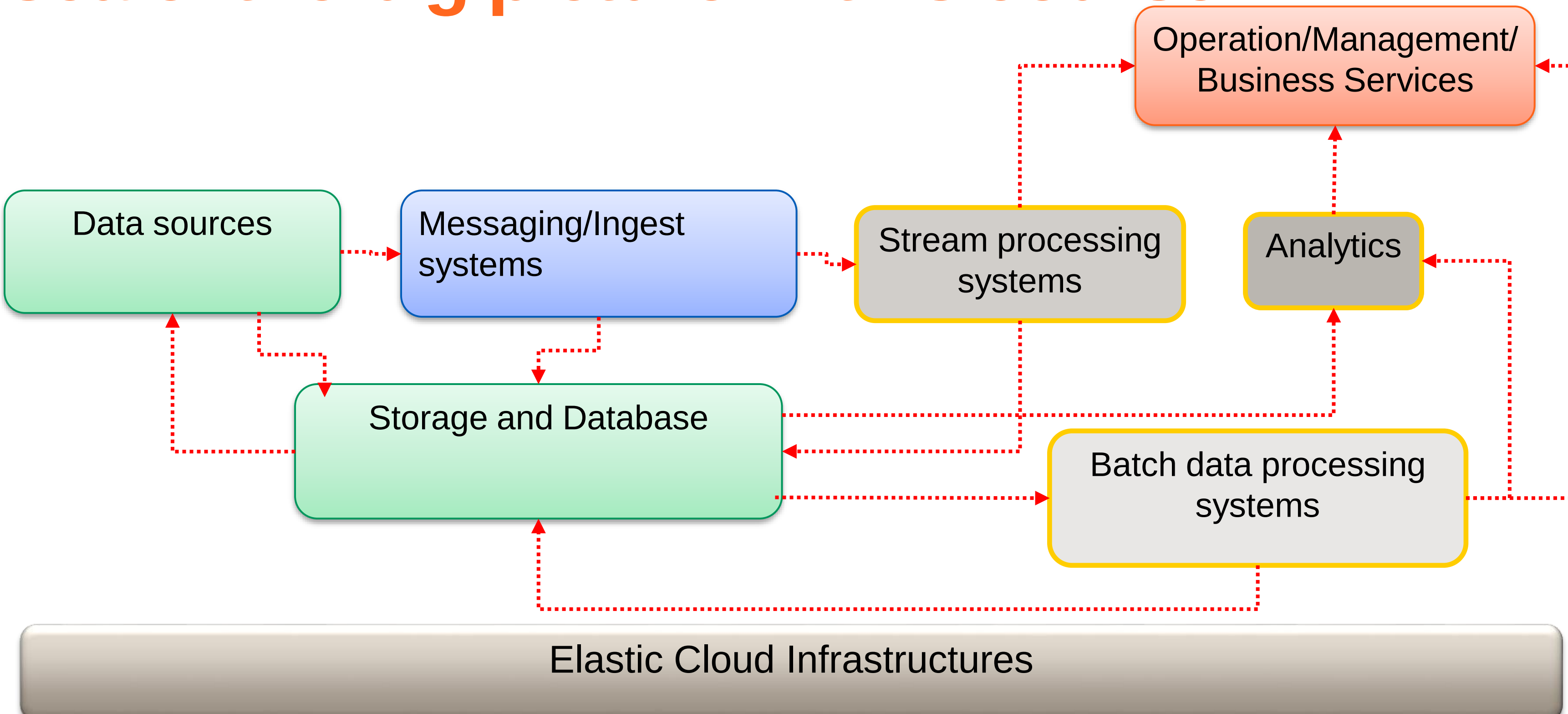
# Learning objectives

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

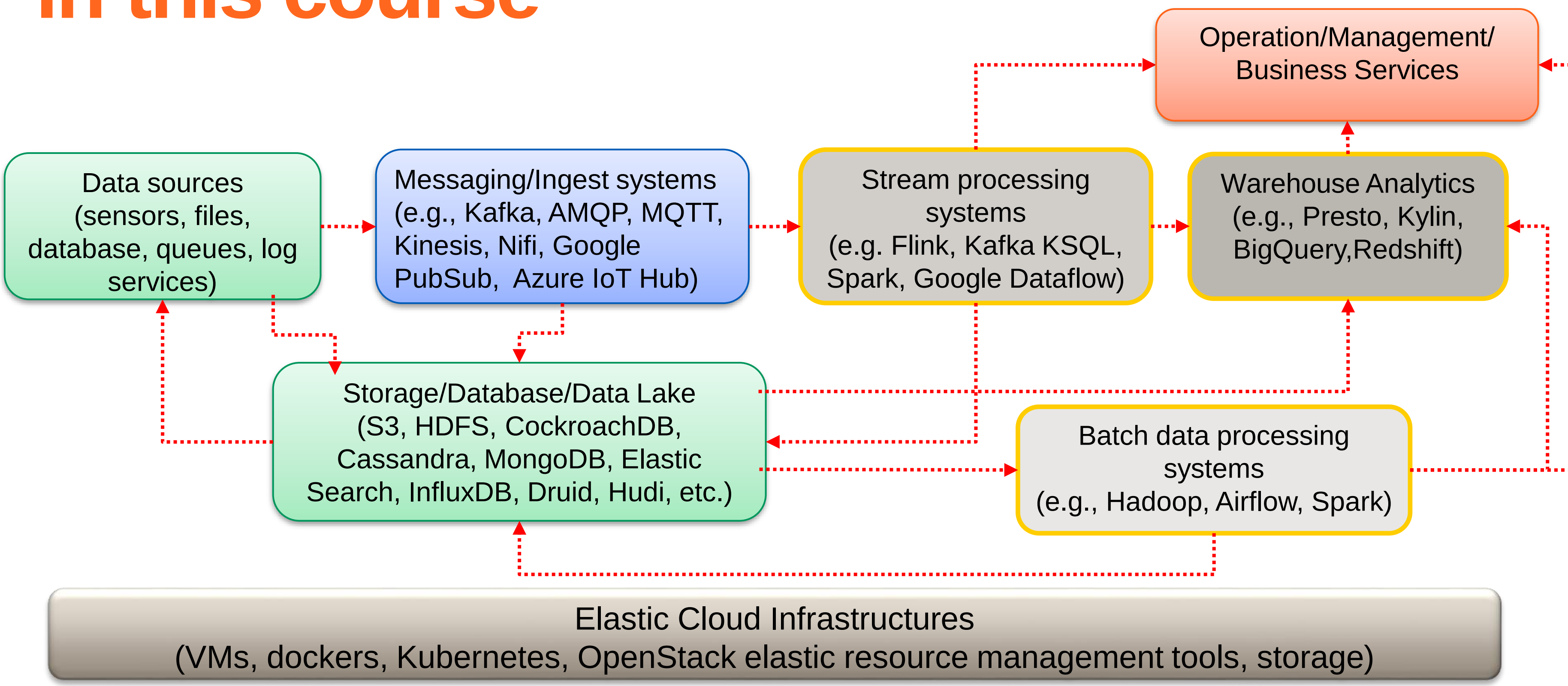
# Recall

- **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

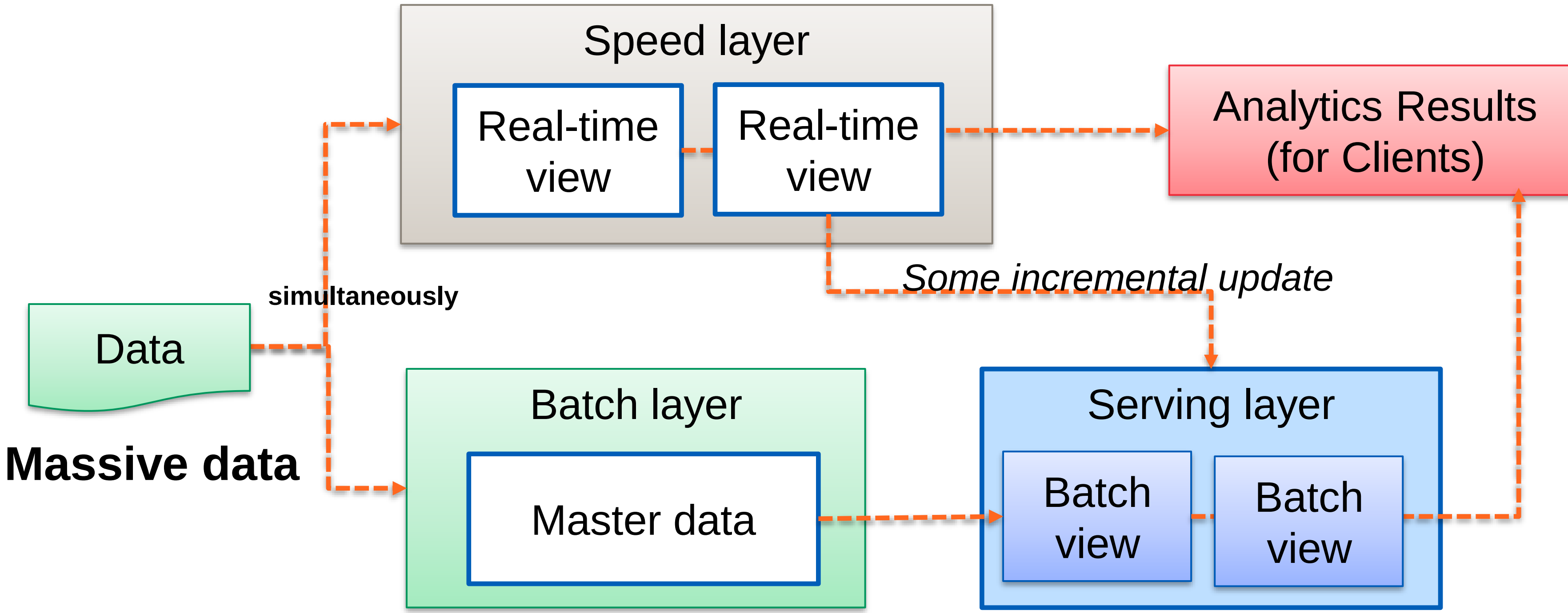
# Basic building blocks for big data at large-scale: the big picture in this course



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

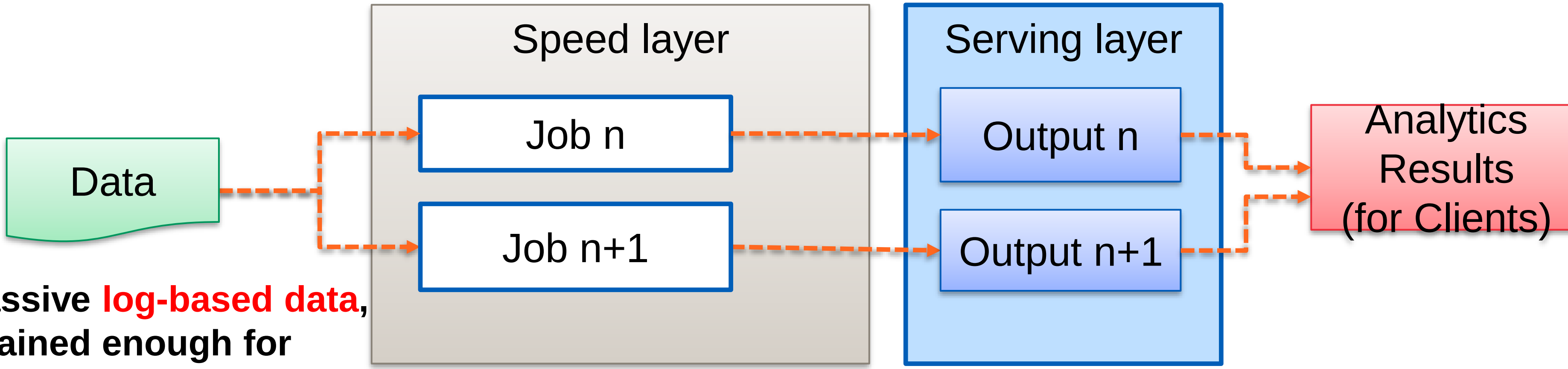


# Lambda architectural style




**Check: http://lambda-architecture.net/**

# Kappa architectural style




**Massive log-based data, retained enough for different analytics needed**

**Job n: analytics**
**Switch from version n to version n+1**

**Check: https://milinda.pathirage.org/kappa-architecture.com/ & http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html**

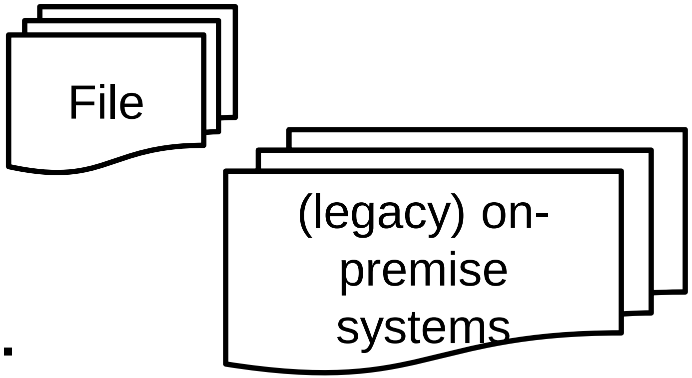
# Quick check

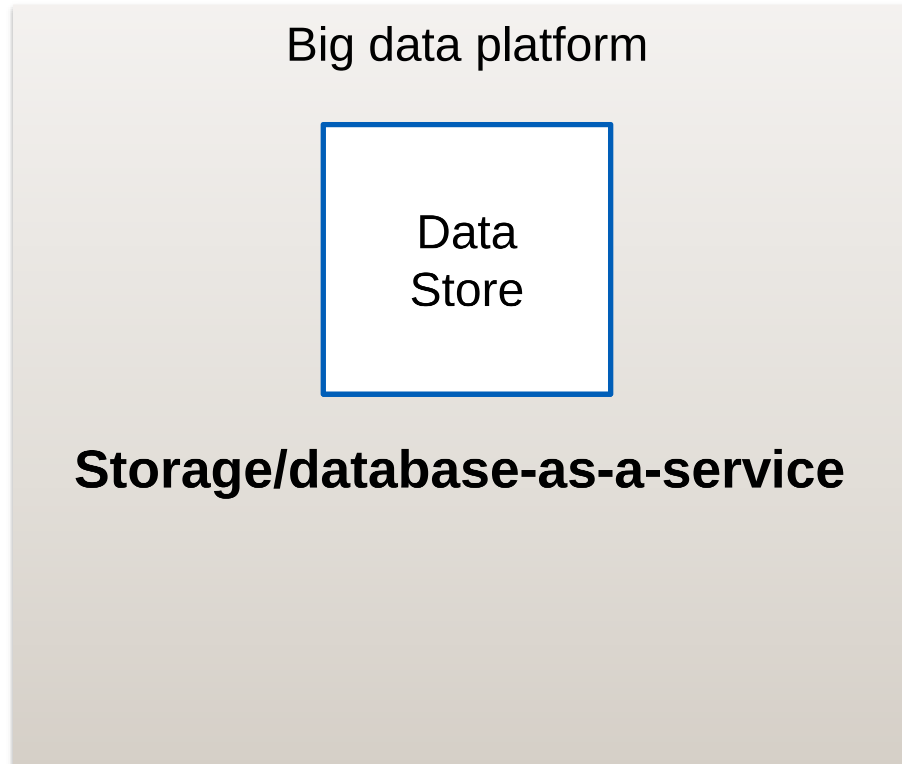
**"A big data platform monitors network usage of devices from million+ customers. We have different levels: Sensor/Customer, Node (concentrator of multiple customers), Agent (concentrator of multiple Nodes) and the whole network. In a region, the real operator can generate 1.4 billion records per day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

Aalto University
School of Science

# Basics of data upload

# Bring big data in files/datasets into platforms

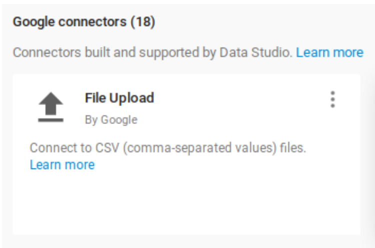


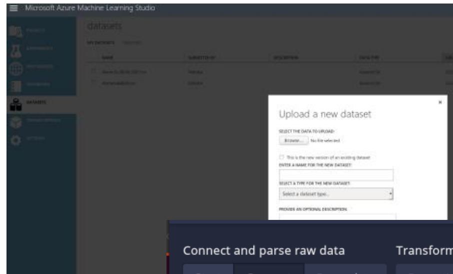




**e.g.**

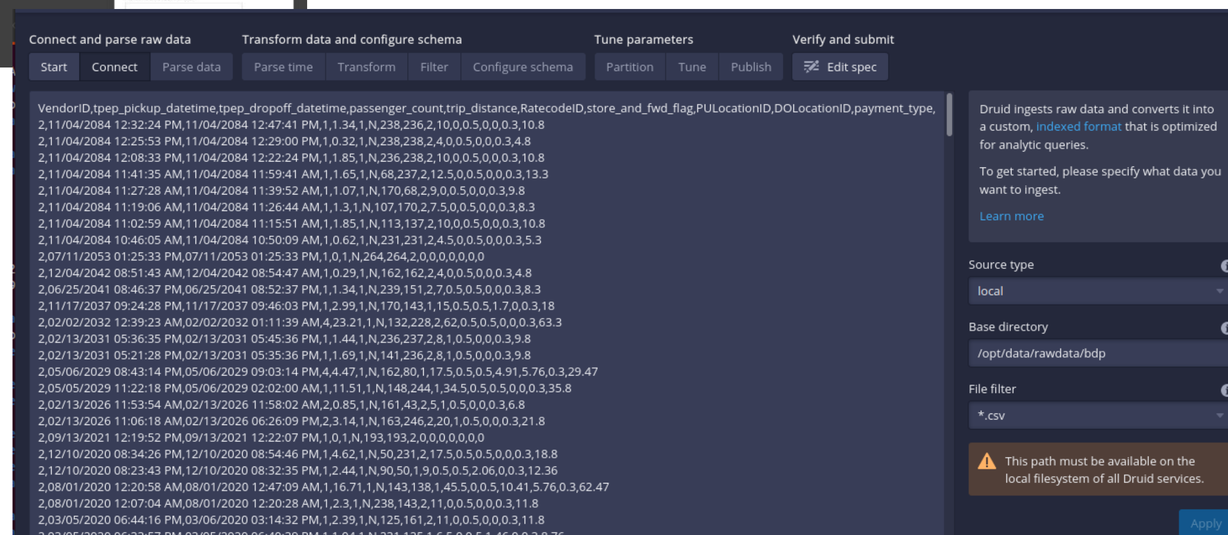
- **logs of machines**
- **sell receipt transaction records**
- **Images/video**

# First obstacle: uploading big data into cloud data storage/database services



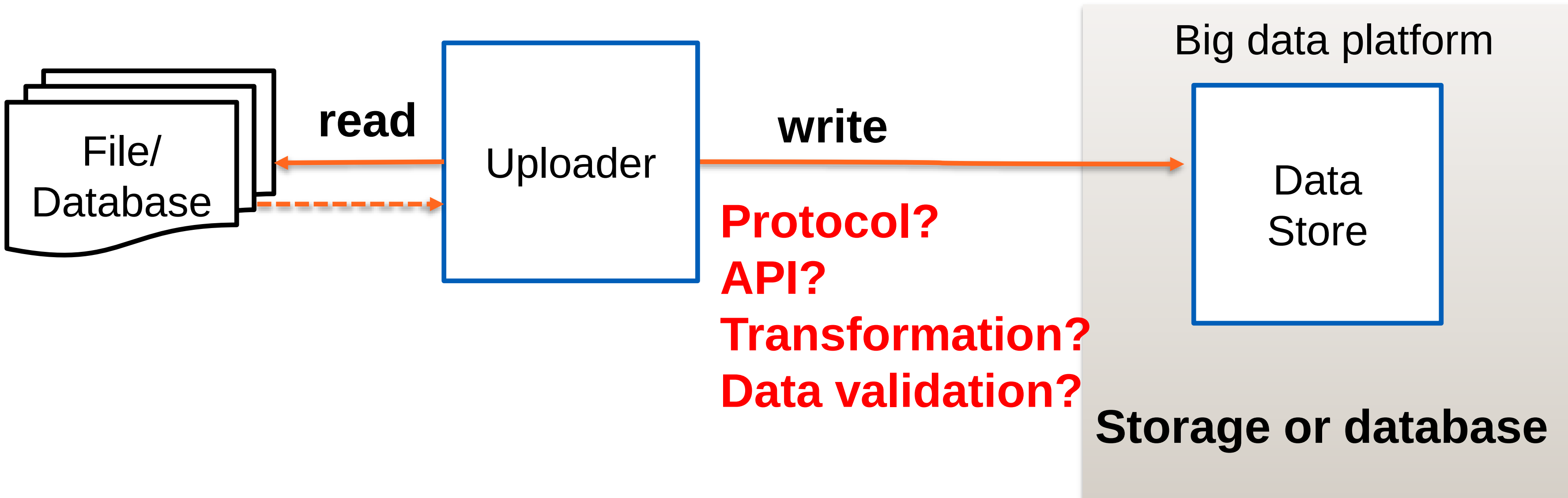




**What would be a good way? Using Flask REST API?**



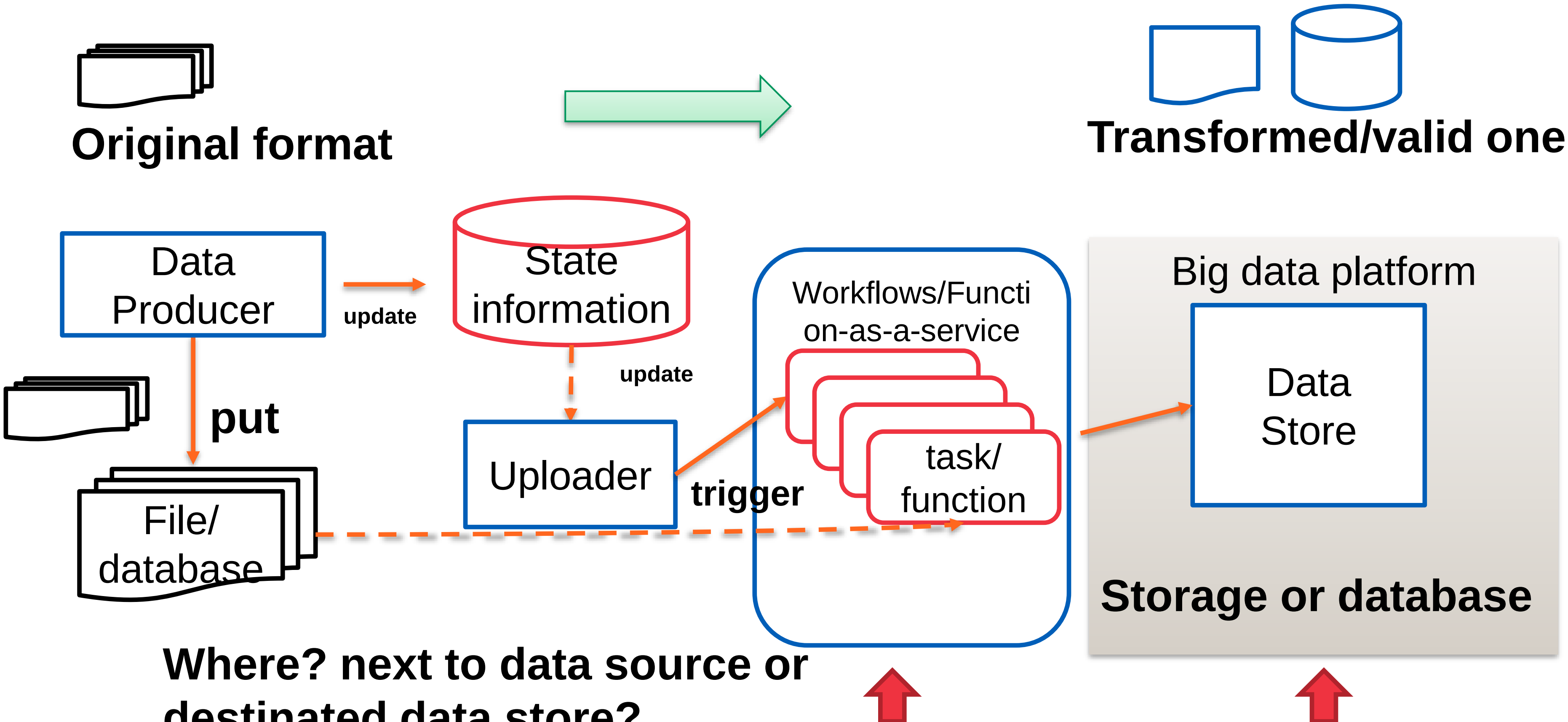

**e.g., upload data into the cloud store and run machine learning**

# 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?**

# If the transformation/validation is needed?



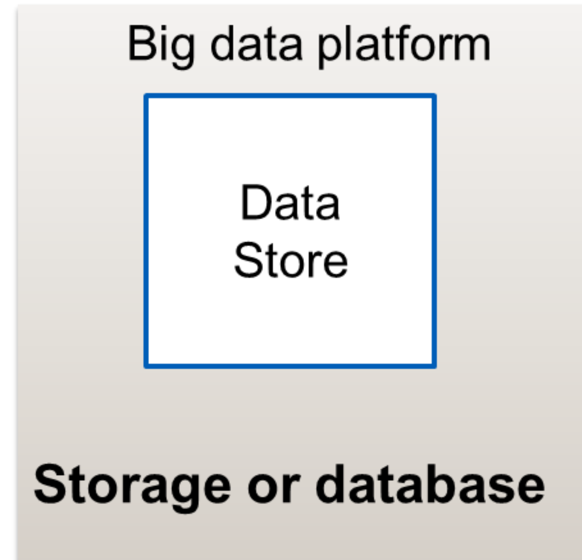

**Where? next to data source or destinated data store?**

# Check the simple example in

## https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/queuebaseddataingestion

# Integrate streaming data sources into platforms

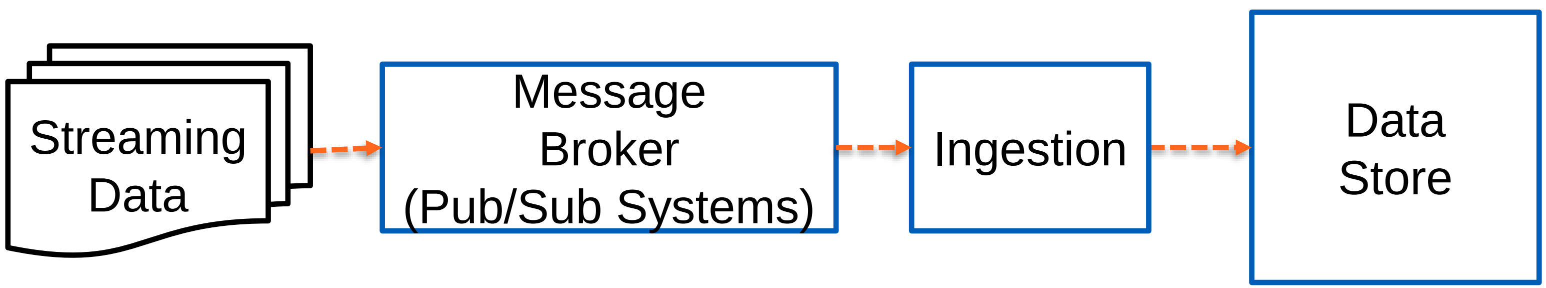

**Streaming protocols/frameworks**



# 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

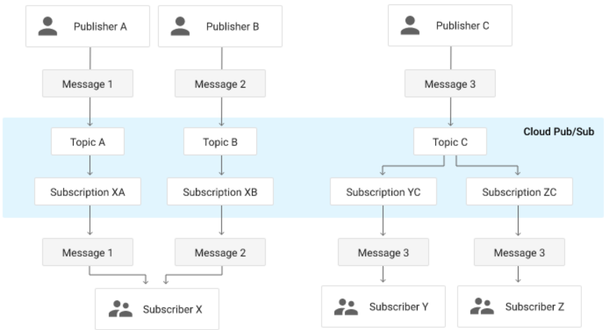



Figure source: https://cloud.google.com/pubsub/docs/overview

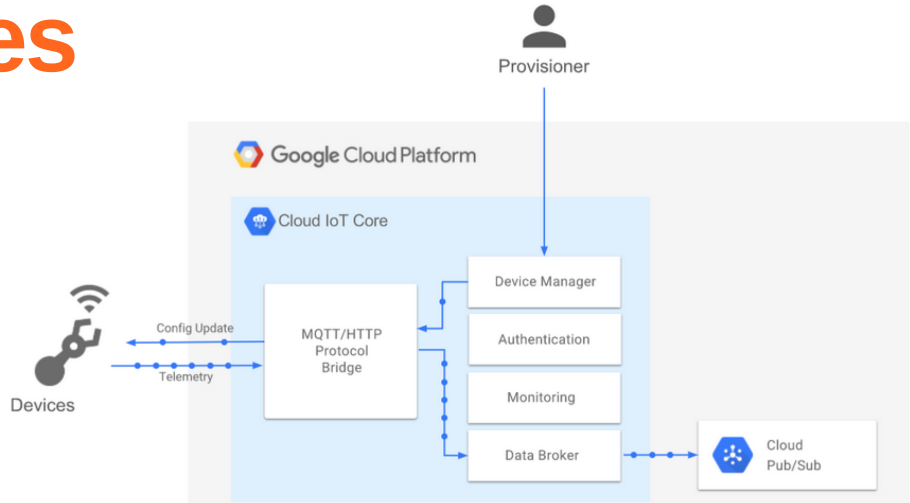



Figure source: https://cloud.google.com/iot/docs/concepts/overview

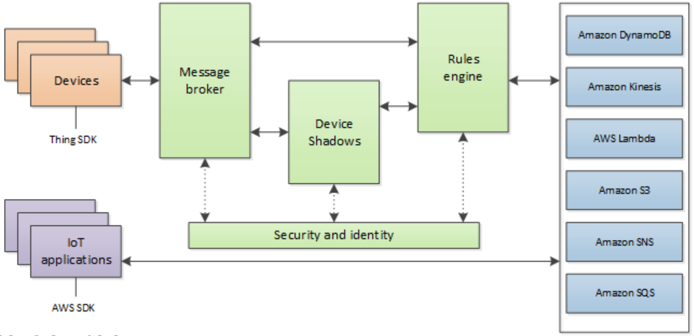



Figure source: https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html

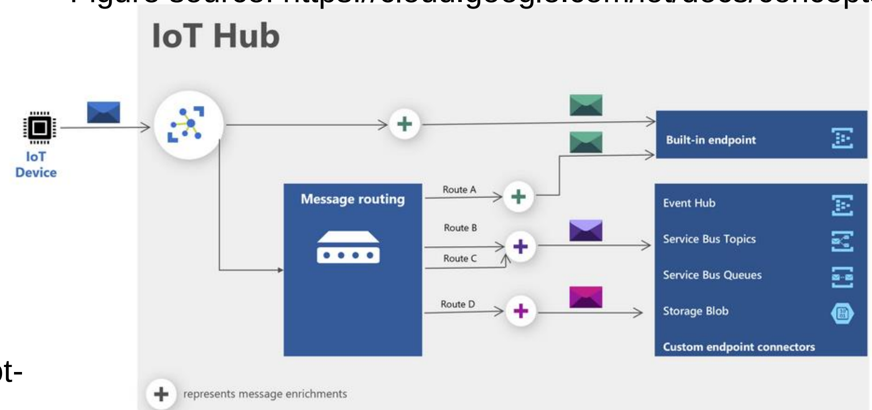



Figure source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-message-enrichments-overview

**Do you see common concepts/terms?**

Aalto University
School of Science

# Some important protocols

- **Protocols**
  - AMQP, MQTT, NATS (https://nats.io/)
- **Systems**
  - Apache Kafka,  Apache Pulsar, Apache RocketMQ
- **Distinguish between “protocols” and “specific frameworks”**
  - How would they affect your design?

# MQTT & AMQP protocols

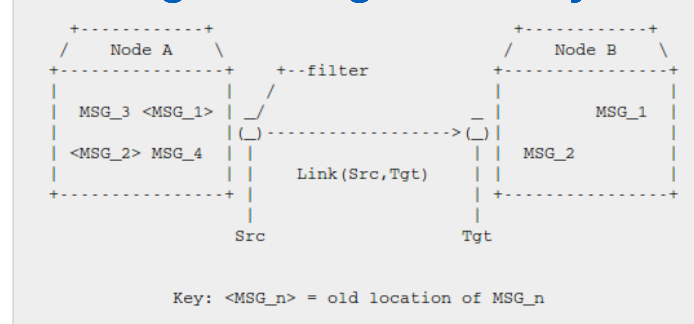
# AMQP - Overview

- **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**
- **We use it for *big data movement and tasks coordination***

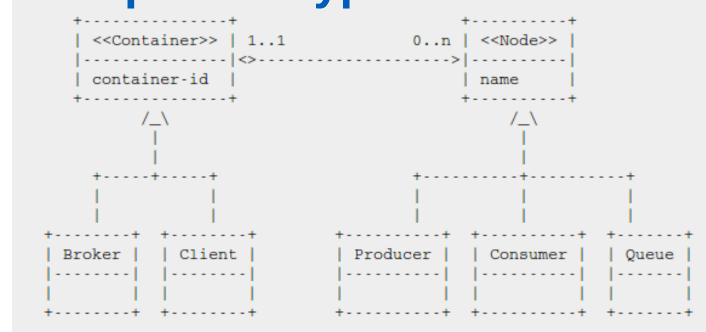
# Core concepts – Message/Transport

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

## Message storage/delivery




## Component types




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

# 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!**

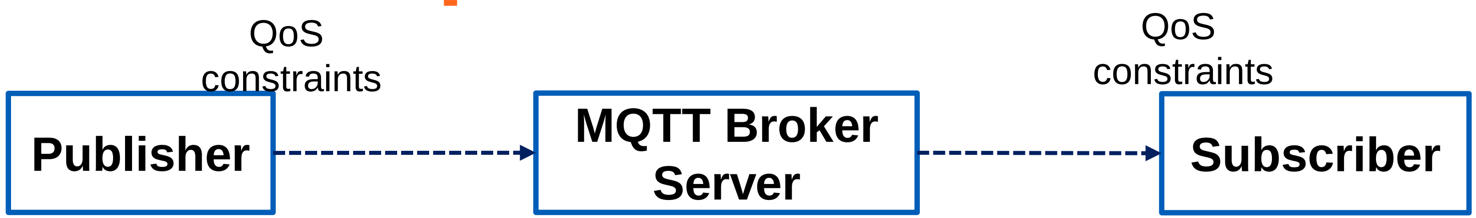
# MQTT Overview

- **http://mqtt.org**
- **OASIS Standard**
  - ISO/IEC 20922:2016 (Message Queuing Telemetry Transport (MQTT) v3.1.1)
- **IoT/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
- **We use it mostly for *big data movement***

# MQTT Protocol Features

- **Lightweight protocol**
  - Small message size
  - QoS for message delivery: 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 examples**
  - Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto)

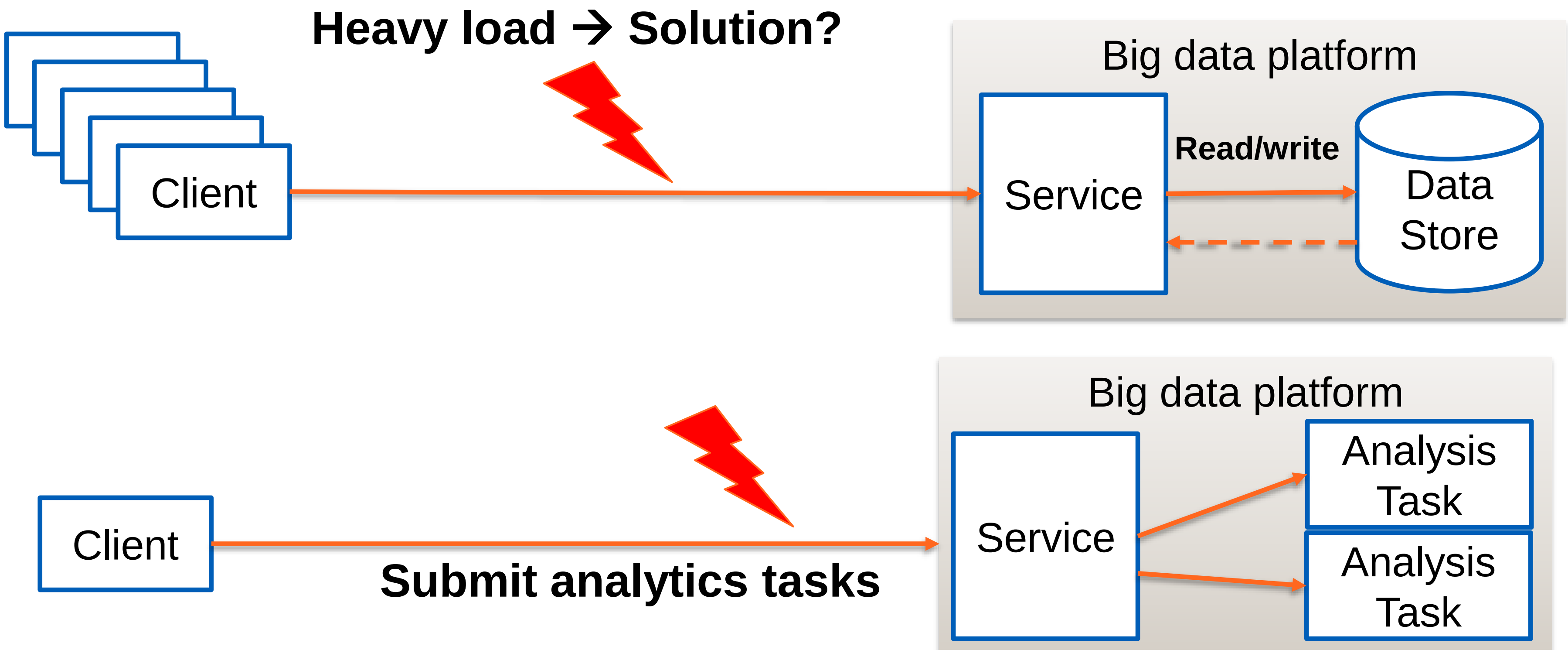
    *docker pull eclipse-mosquitto*
  - Paho: http://www.eclipse.org/paho/
  - RabbitMQ: https://www.rabbitmq.com/
  - Cloud providers: http://cloudmqtt.com (offer a free instance)
  - Cluster of MQTT brokers: VerneMQ (https://vernemq.com/), EMQ (https://www.emqx.io/)

# So which one you think is suitable for this?

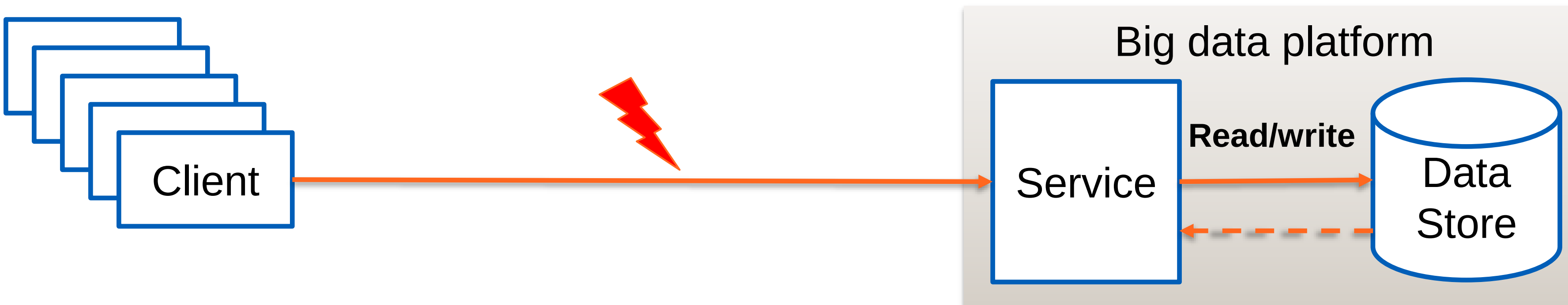
**“A big data platform monitors network usage of devices from million+ customers. We have different levels: Sensor/Customer, Node (concentrator of multiple customers), Agent (concentrator of multiple Nodes) and the whole network. In a region, the real operator can generate 1.4 billion records per day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

# Optimize service requests and functionalities

# Concurrent contention

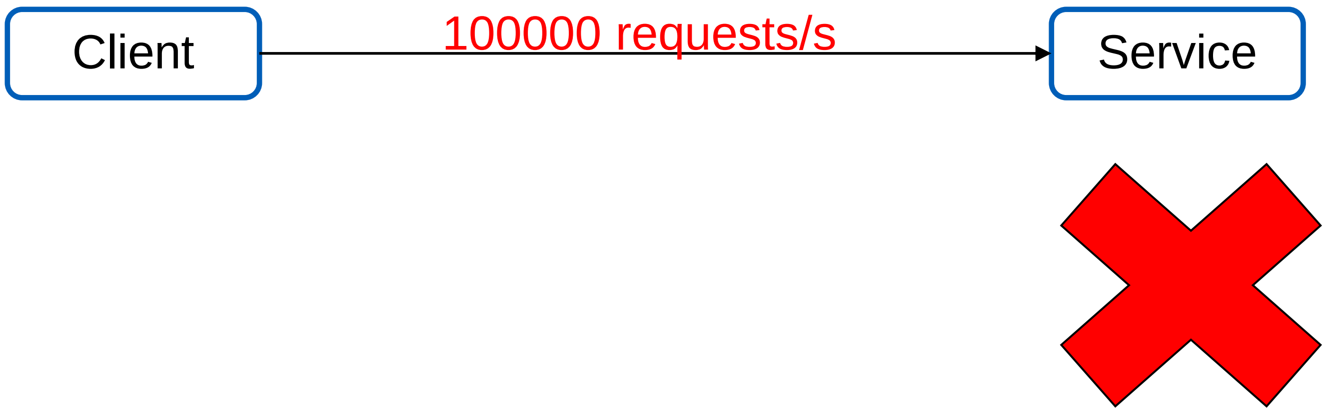


# Back-pressure or elasticity



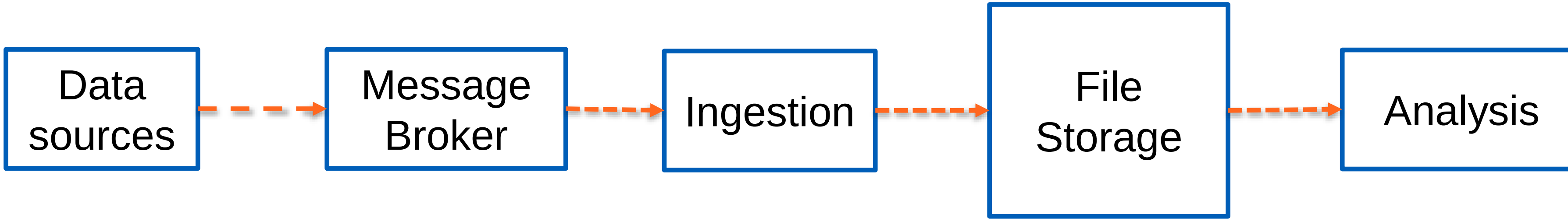

**Back-pressure: control, drop, and buffer**

# Prevent too many accesses?




**A related situation:** prevents clients to retry a (failed) operation
http://martinfowler.com/bliki/CircuitBreaker.html
https://msdn.microsoft.com/en-us/library/dn589784.aspx

# Scaling in every place of big data pipelines

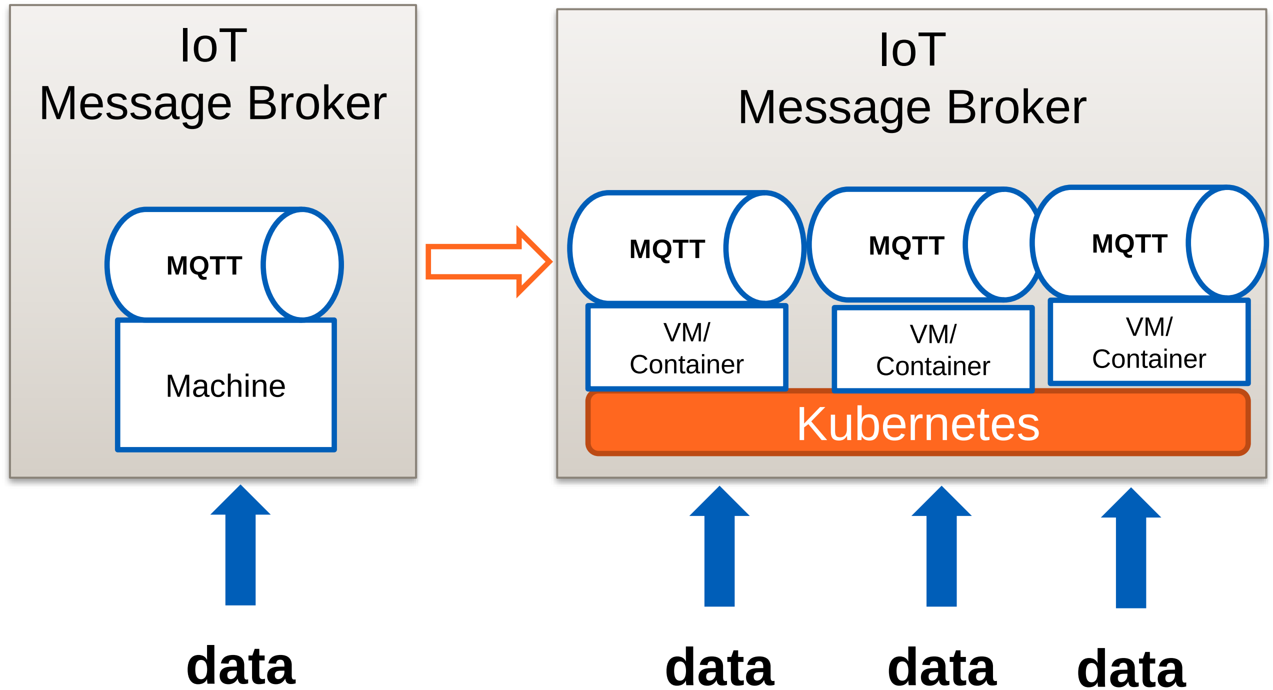



- **Scaling**
  - disk spaces for file storage
  - resources for data ingestion
  - resources for data analysis

Happen at different times and location

# Scaling middleware nodes

- Increase the number of brokers when more data arrive
- Provide dedicated brokers on-demand



# Example: scaling compute nodes for data analysis

Monitoring | Jobs | VM Instances | Configuration | Web Interfaces

| Name | Role | |
|---|---|---|
| thebasecluster-m | Master | SSH |
| thebasecluster-w-0 | Worker | |
| thebasecluster-w-1 | Worker | |
| thebasecluster-w-2 | Worker | |
| thebasecluster-w-3 | Worker | |

Equivalent REST

**4 nodes**

**On-demand change**

| | |
|---|---|
| Name | thebasecluster |
| Region | europe-north1 |
| Zone | europe-north1-a |
| Autoscaling | Off |
| Scheduled deletion | Off |
| Enhanced flexibility mode | Off |
| Master node | Standard (1 master, N workers) |
| Machine type | n1-standard-2 (2 vCPU, 7.50 GB memory) |
| Primary disk type | pd-standard |
| Primary disk size | 500 GB |
| Worker nodes | 6 |
| Machine type | n1-standard-1 (1 vCPU, 3.75 GB memory) |

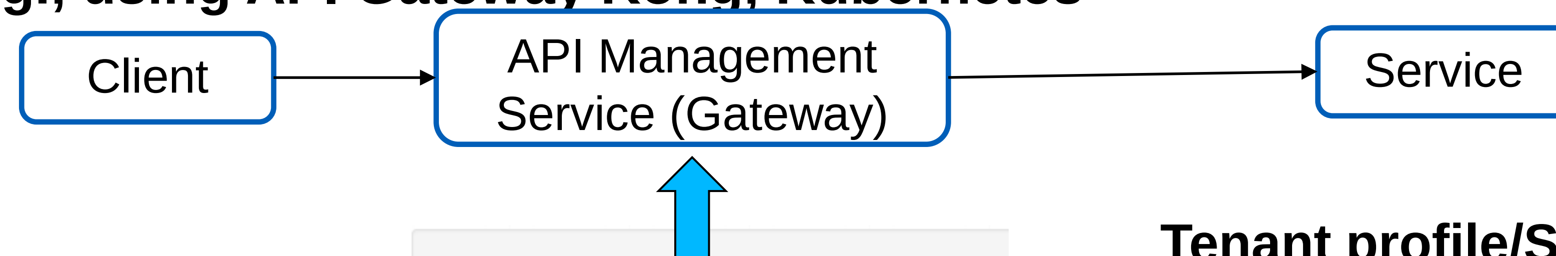
Monitoring | Jobs | VM Instances | Configuration | Web Interfaces

| Name | Role | |
|---|---|---|
| thebasecluster-m | Master | SSH |
| thebasecluster-w-0 | Worker | |
| thebasecluster-w-1 | Worker | |
| thebasecluster-w-2 | Worker | |
| thebasecluster-w-3 | Worker | |
| thebasecluster-w-4 | Worker | |
| thebasecluster-w-5 | Worker | |

**6 nodes**

# Throttling

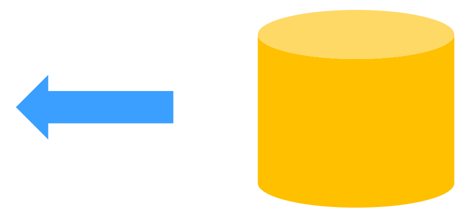
- **Drop strategy: Disable too many accesses and disable unessential services**
  - Dynamic vs static configuration
- **E.g., using API Gateway Kong, Kubernetes**



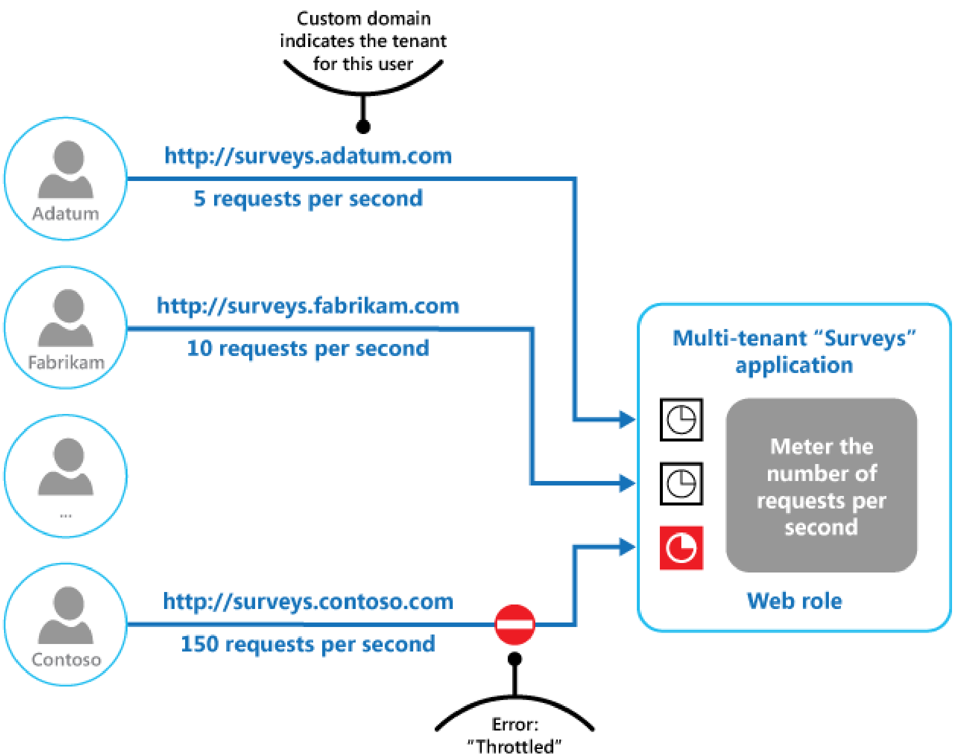

Code: http://www.django-rest-framework.org/api-guide/throttling/#how-throttling-is-determined

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

**Tenant profile/SLA**



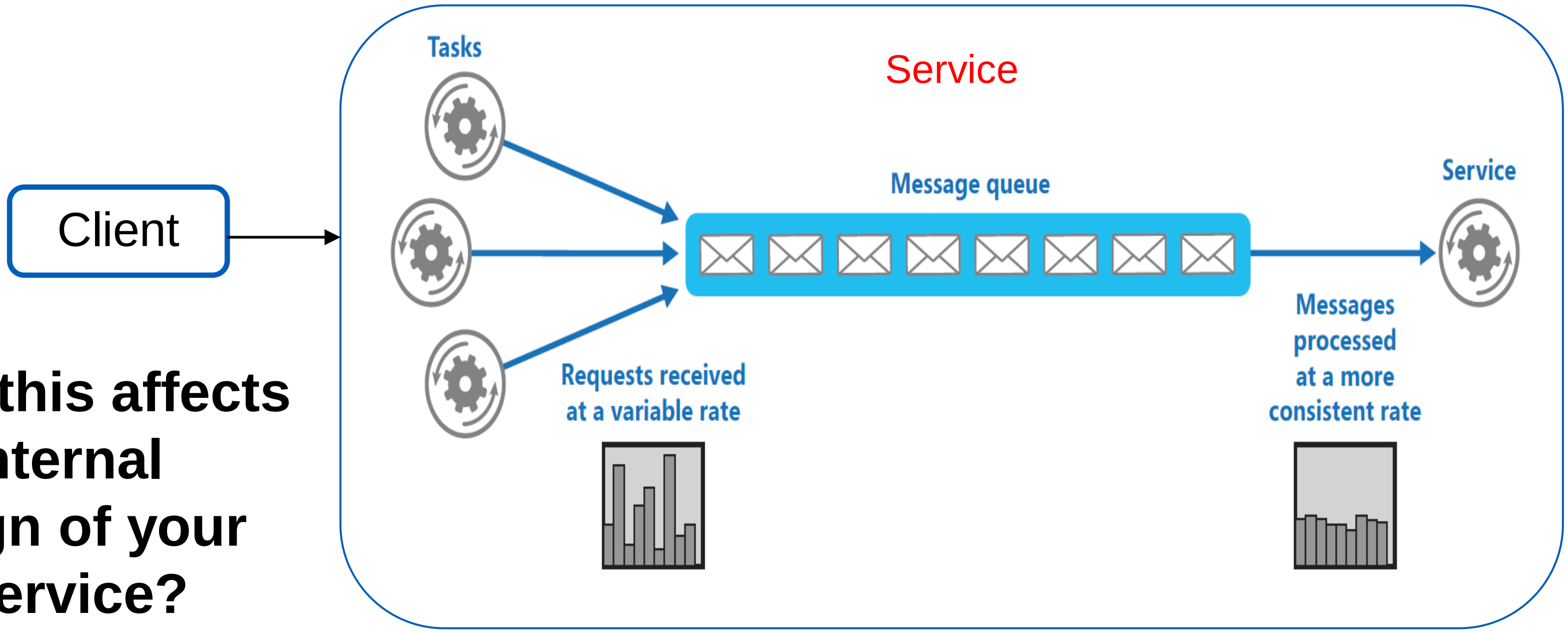
# Example of  throttling based on roles




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

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

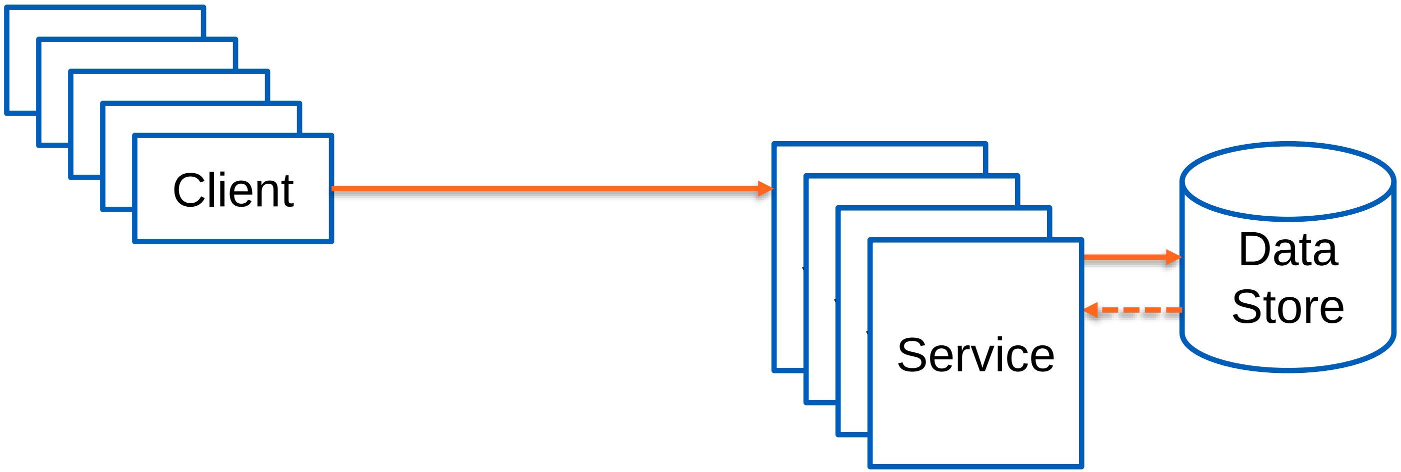
# Using tasks and queue-based load leveling pattern




**How this affects the internal design of your big service?**

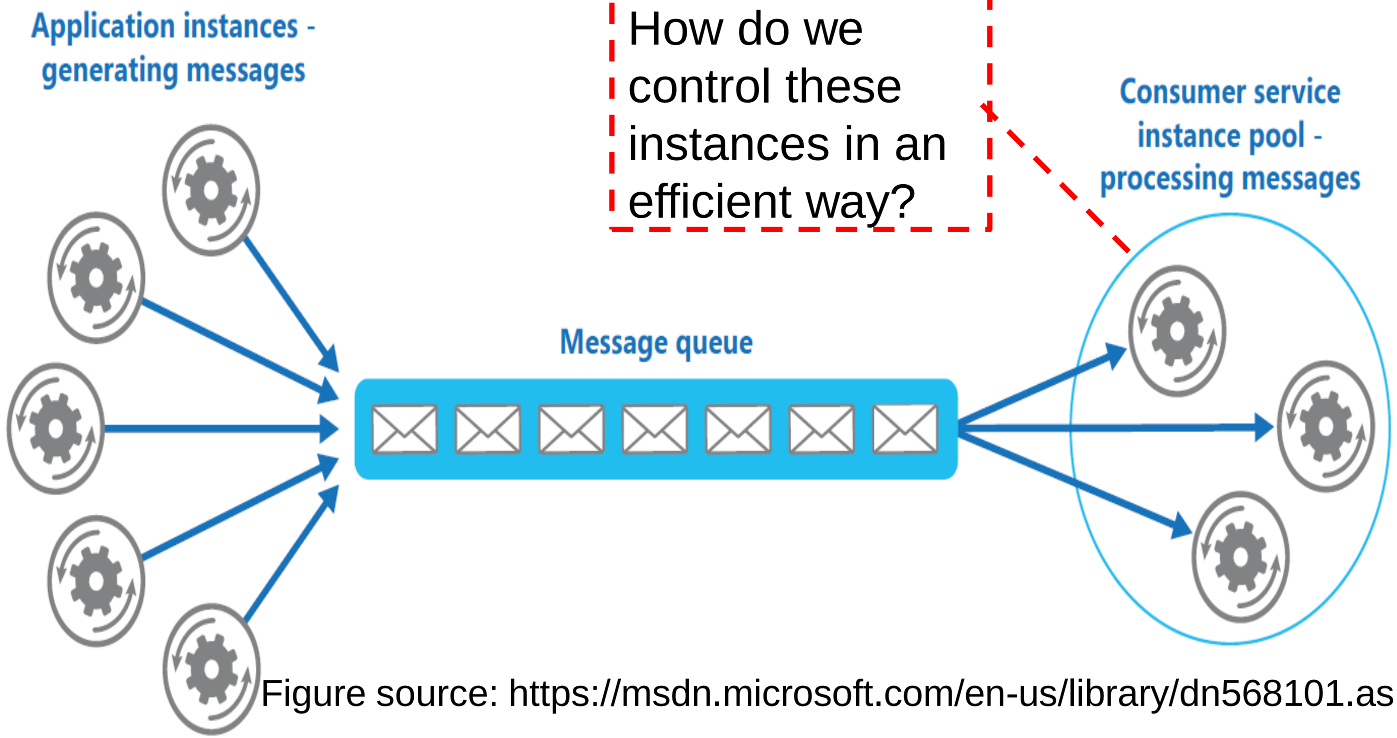
Figure source: https://msdn.microsoft.com/en-us/library/dn589783.aspx

# Heavy load between service serving request and data store




**Elastic solution: scale out or up**

# Using multiple instances of services and queues

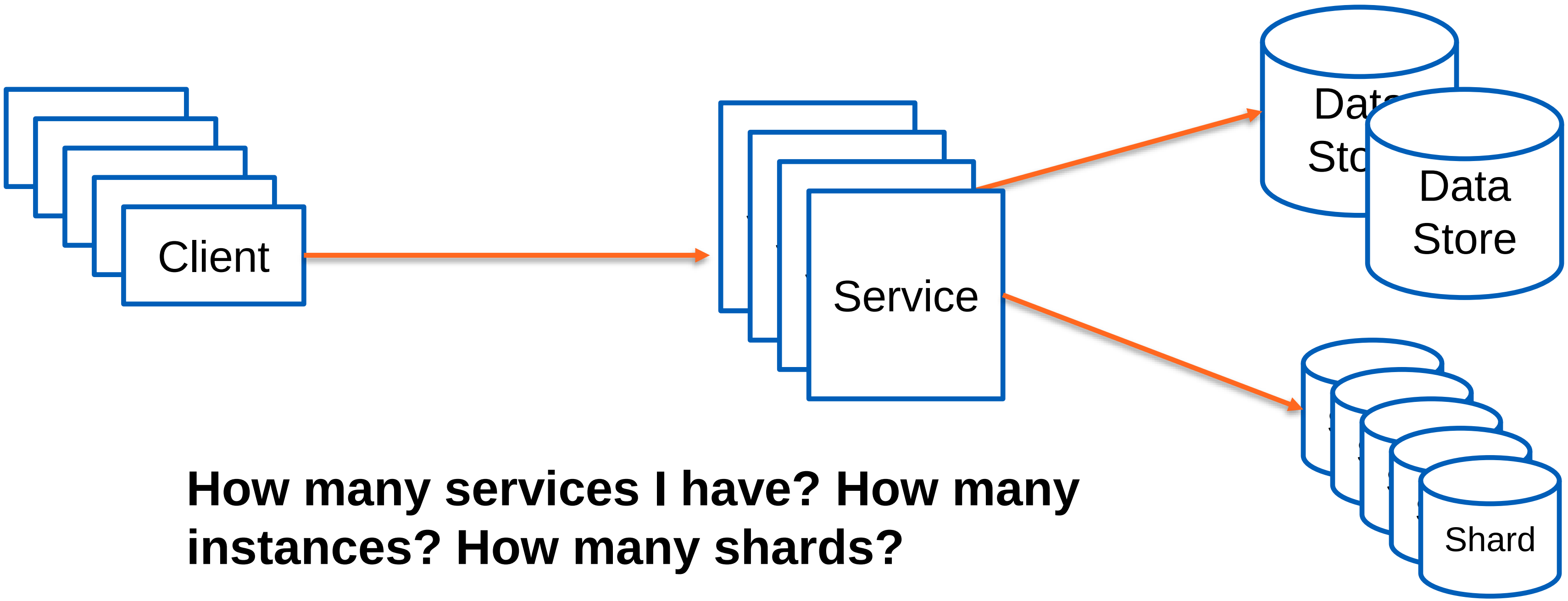



How do we control these instances in an efficient way?

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

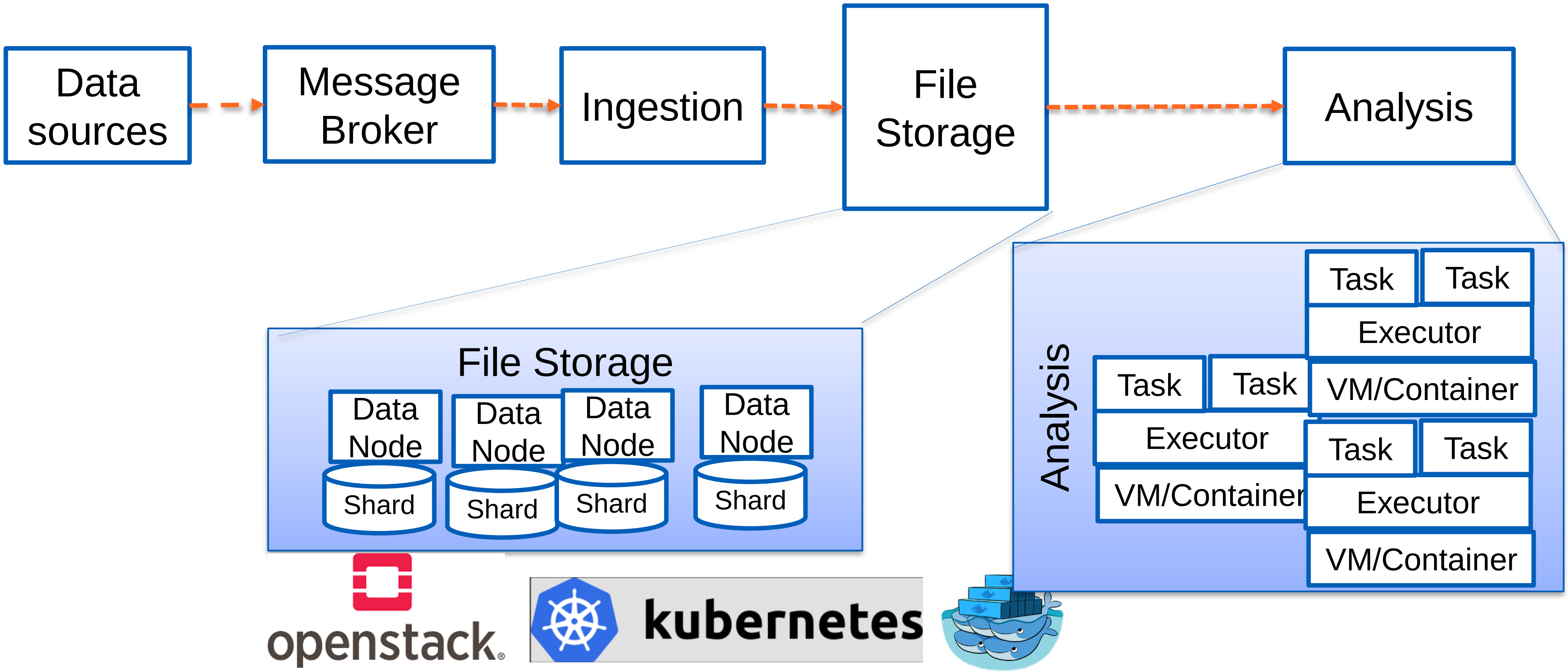
# Discovery and consensus

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




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

# Runtime view of some components



# Multiple instances

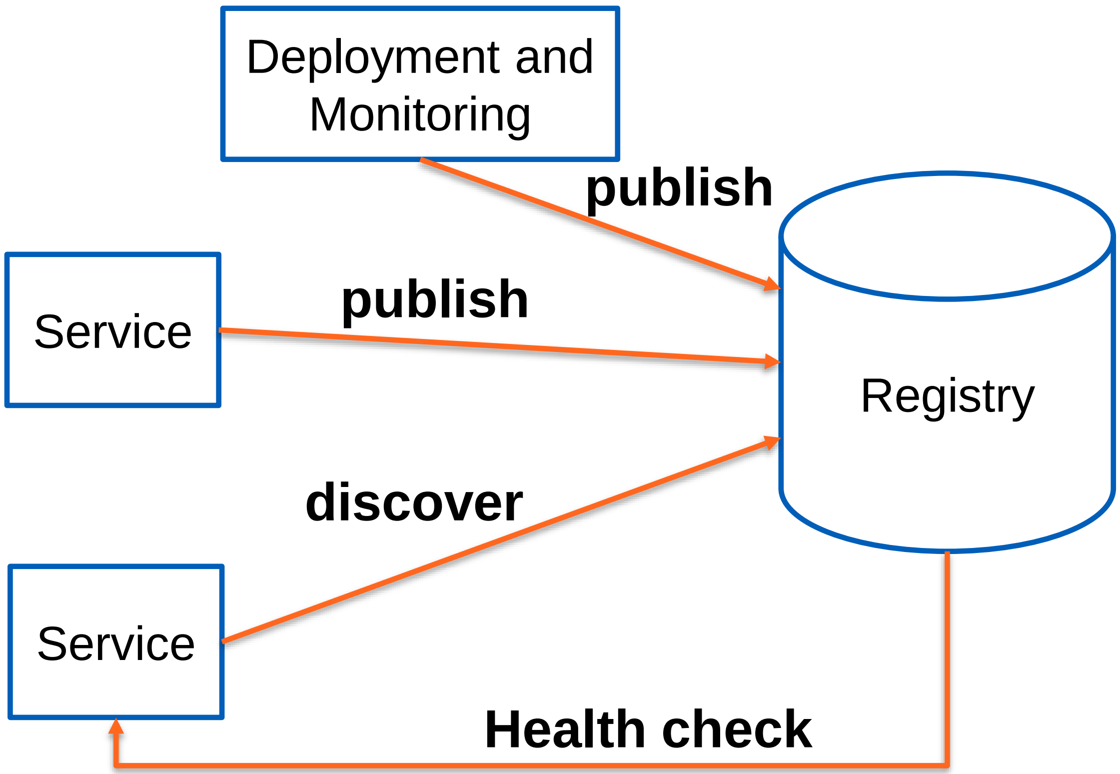
- **A building block of big data platforms can have many services and a service can have many instances**
  - E.g., for replication and load balancing
  - A database service (e.g. MongoDB) has multiple data nodes, each is a service responsible for a shard/partition
  - A processing engine (e.g., Spark 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

# 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**

# How do we know where are available services and data?

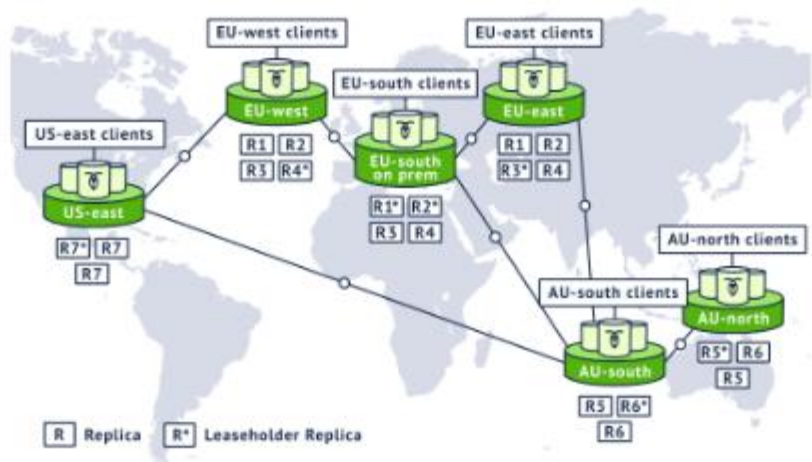



Figure 1: A global CockroachDB cluster

Figure source: *Taft et al., CockroachDB: The Resilient Geo-Distributed SQL Databasettps://dl.acm.org/doi/pdf/10.1145/3318464.3386134*

"At the time of writing, our largest Druid cluster deployment uses more than 100 nodes for Historical processes and about 75 nodes for MiddleManager processes. We ingest over three million events per second and respond to over 250 queries per second. We keep seven days of queryable data in Druid Historical nodes and two years of data retention in S3 deep storage."

Source: *November 8, 2021, https://www.confluent.io/blog/scaling-apache-druid-for-real-time-cloud-analytics-at-confluent/*

# Distributed Coordination

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

TABLE 4. PATTERNS OF PAXOS USE IN PROJECTS

| Project | Consensus System | Usage Patterns | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 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 | | | ✓ | | | | ✓ | | |

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)”

**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

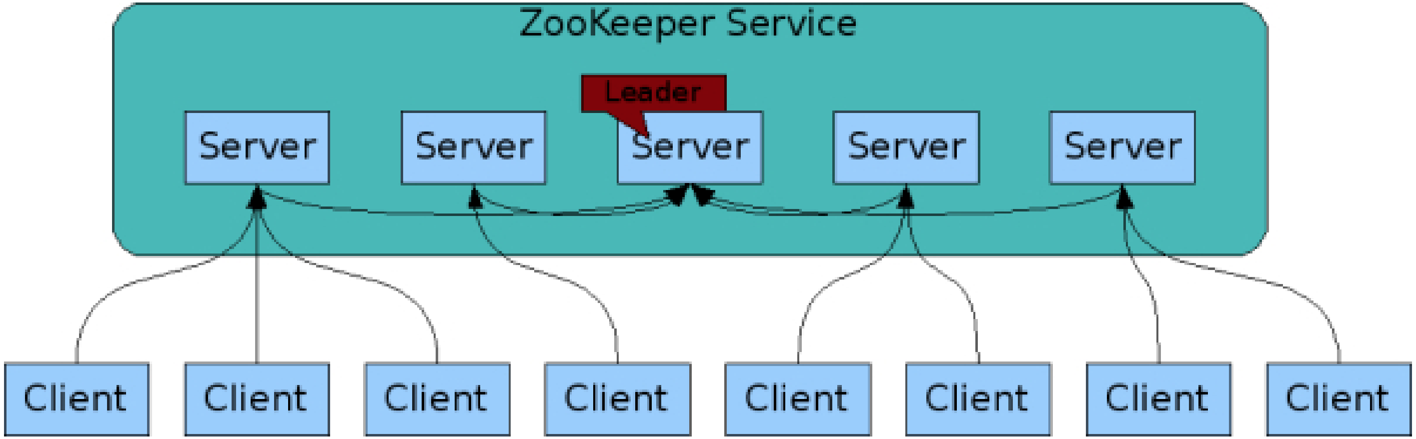



Figure source: https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html

# 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

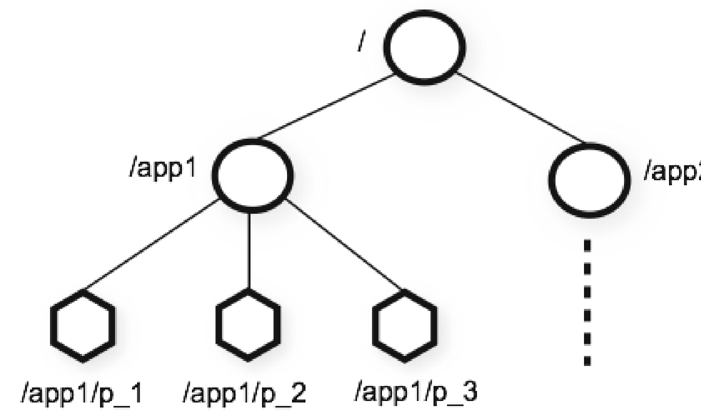



Figure source: https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html

# Technology choices: Consul

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

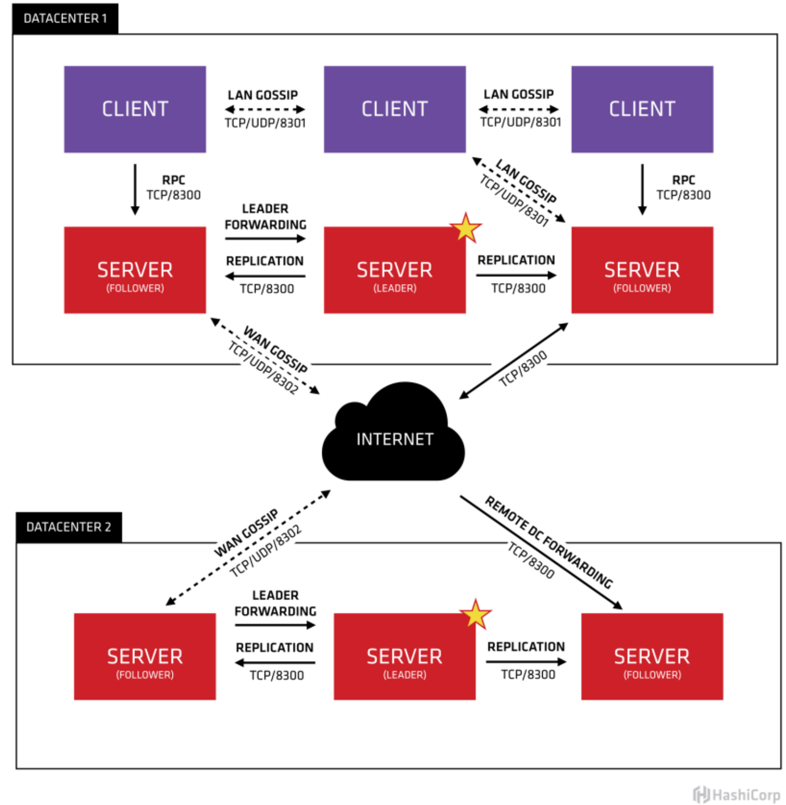



Figure source: https://www.consul.io/docs/internals/architecture.html

# 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

# What you should do this week

- Look at the list of data sources and start think which data sources you will use for your study
- Lambda and Kappa architecture styles
- Check and play with basic ingestion: simple queue, MQTT/AMQP (from the cloud background)
- Brush up patterns for scaling and failure handling
- Look at how service discovery and consensus are implemented in big data systems

***Note:*** *materials/links are in our git and slides*

# Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io