# Practical Distributed Systems Introduction

Piotr Jaczewski
RTB House

http://mimuw.rtbhouse.com/slides/1.pdf

# RTB House

This course is loosely based on the experiences derived from the construction of the RTB advertising platform developed by RTB House.

Some technical details:

- Up to 10 million requests per second.
- 2500+ physical hosts.
- 5 datacenters on 3 continents.
- 10+ petabytes of data.
- 3000+ customers.
- 20000+ active ad campaigns.

# What this course will be about?

During this course we will discuss the broad topic of data intensive distributed systems.

The course will be focused on:

- The engineering aspects of distributed systems.
- The design and architecture of distributed systems components.
- The operational aspects of running a distributed system.
- The performance considerations for a distributed system.

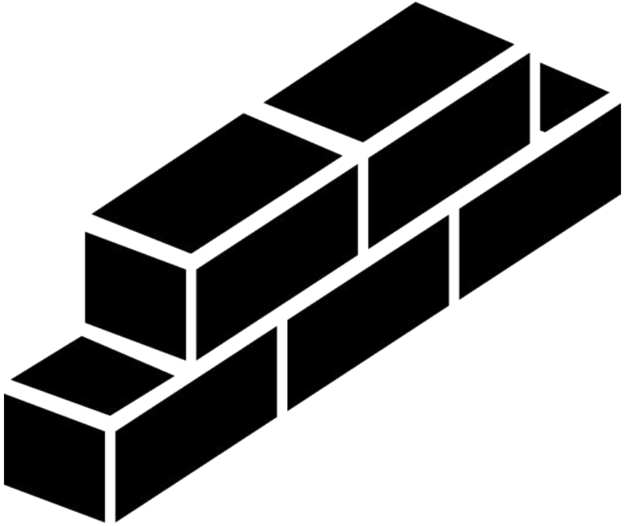
# Motivation

- We need to build business and applications around modern web.
- We must handle ever increasing traffic of thousands and millions of requests.
- We want to maintain high availability and resilience of the system.
- We want to be able to perform maintenance of the system without causing harm.

Under these conditions we want our systems to be **scalable**, **reliable** and reasonably **maintainable**.

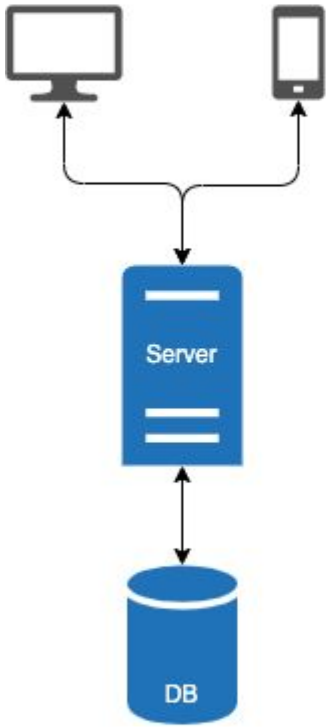
# Architecture

During this lecture we will try to gradually build a generic architecture for a modern scalable distributed system and to identify its components.



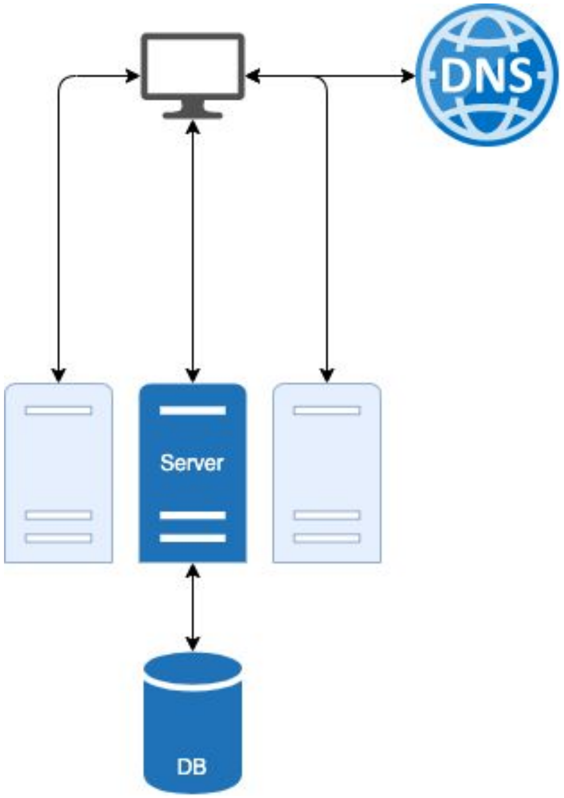
# Single server

- The basic first approach to the development of any web based system.
- Mostly represents the Proof of Concept for a system or early stages of development.
- Simple deployment.
- Suitable for very small applications.



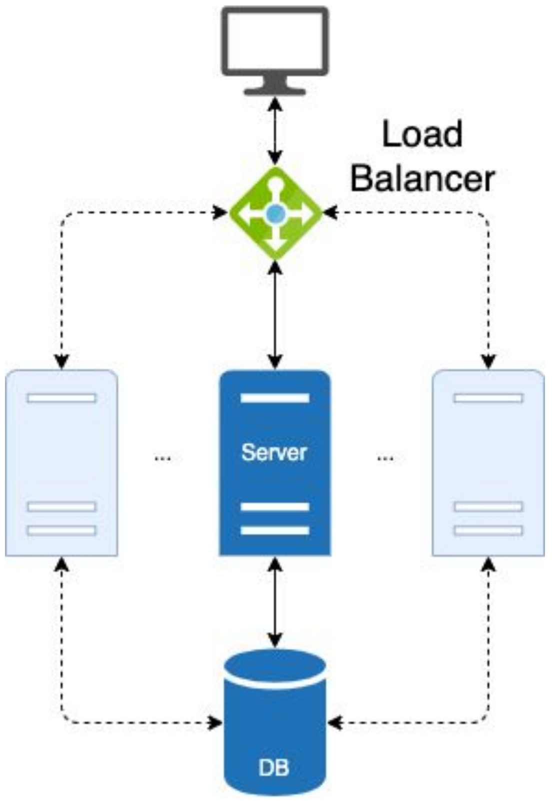
# Multiple servers

- We want to increase the number of servers to handle more a little bit more traffic.
- We assume that the architecture of the system is monolithic and each server handles the same core functionality.
- The servers have the same hardware capabilities.
- We want to distribute the traffic between the servers evenly.
- The DNS host name is mapped to multiple addresses.



# Load balancing

- We add a dedicated load balancing component.
- We want to have more control over the request distribution.
- We want to implement the custom request distribution policy, eg distribute more traffic to the more capable servers.
- We want to implement simple health checking functionality which enables the dynamic addition and removal of machines.

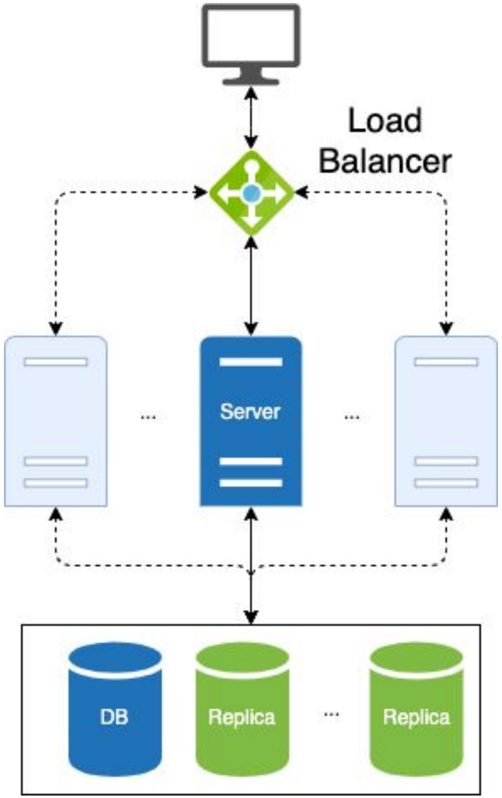


# Data Redundancy

- We come to the point where data redundancy is required to prevent data losses.

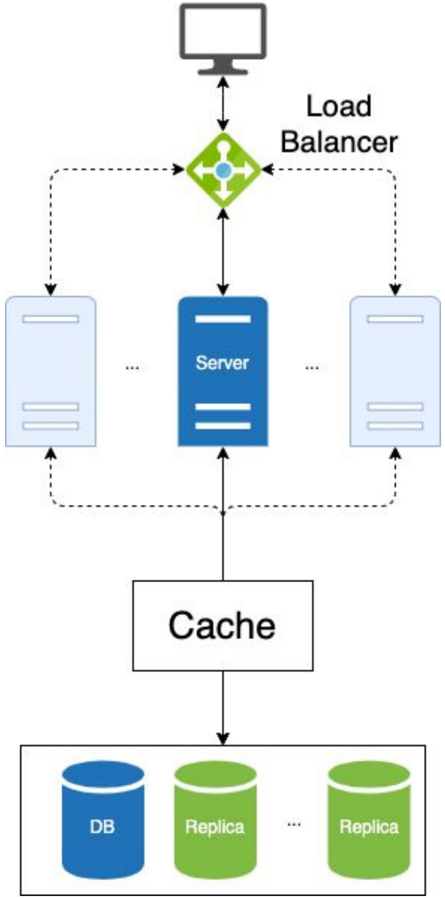
- If the master database comes down we want to easily promote a new master.

- We want to reduce the burden on the master server by allowing the read requests to be handled on replica nodes.



# Caching

- As performance reasons become more important the obvious solution is to introduce caching.
- Caching might be a separate component like a memcached server or caching may be performed by a server application itself.
- Various caching side effects must be taken into account, like consistency issues, so adequate caching strategies must be used not to disturb normal operations.



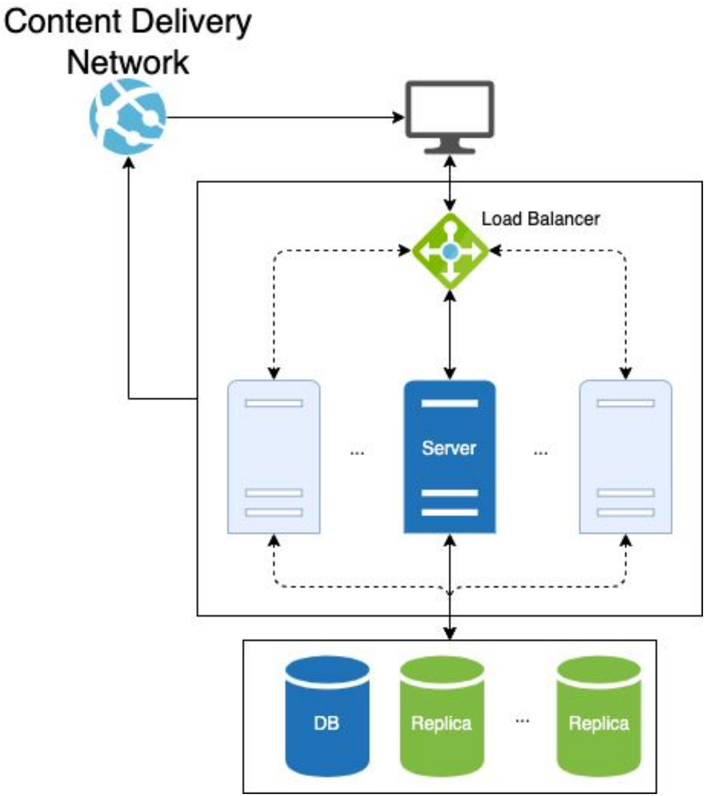
# Caching strategies

- **Cache-Aside** - data missing in cache upon read is fetched  from source and stored in cache by the application.
- **Read-Through** - data is read from cache layer which synchronously reads and stores data from db if missing.
- **Write-Through -** data is always written to the cache layer which synchronously stores it in db.
- **Write-Back -** data is always written to the cache layer which asynchronously stores it in db.

# Content Delivery Networks

The aim of CDN is to reduce web tier load by storing static media on a dedicated caching service.

1. The client sees static media URL referring to the CDN service.
2. The CDN service caches the media content from the server if not already present.
3. The CDN service returns the media content to the client.



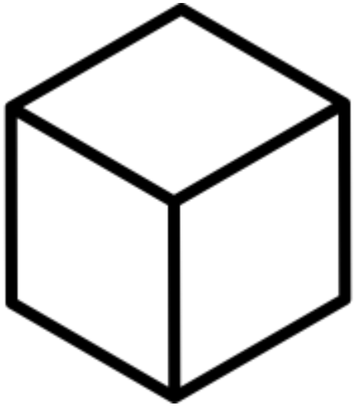
# Monolithic architecture

A state when most of the system functionality is encapsulated within a single deployable application:

- Easier to develop and test at the beginning.
- Straightforward to deploy.
- Easier to scale if application  can be run in multiple instances.

But inevitably leads to “monolithic hell”:

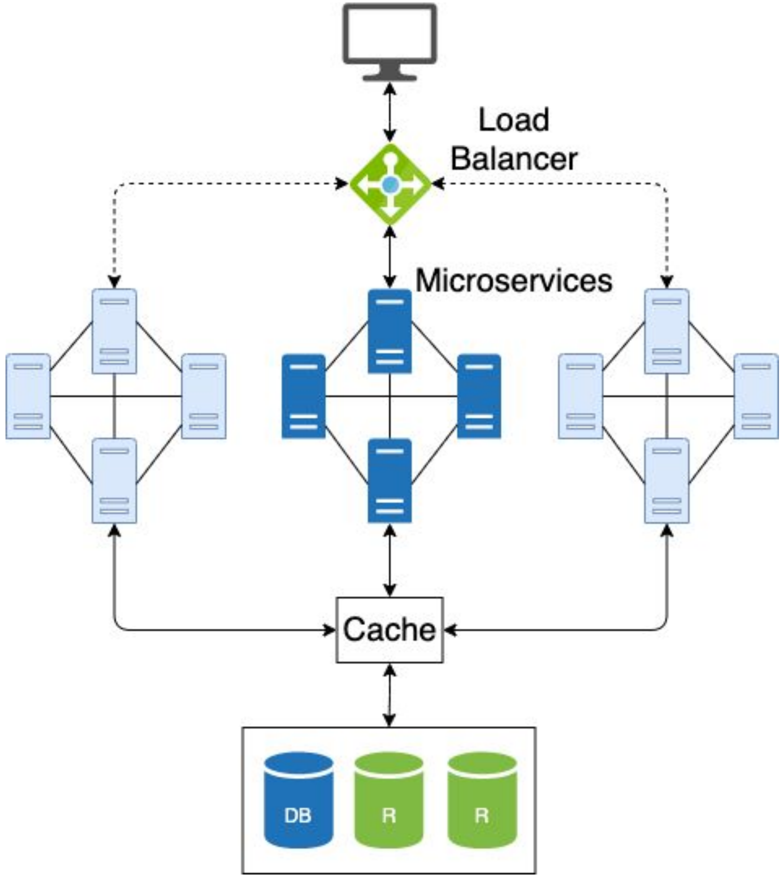
- Convoluted codebase with tightly coupled modules - “big ball of mud”.
- Slow and painful development and difficult testing.



# Microservices

- At some point it may be reasonable to abandon monolithic architecture of the system applications and start transition to a microservices architecture.

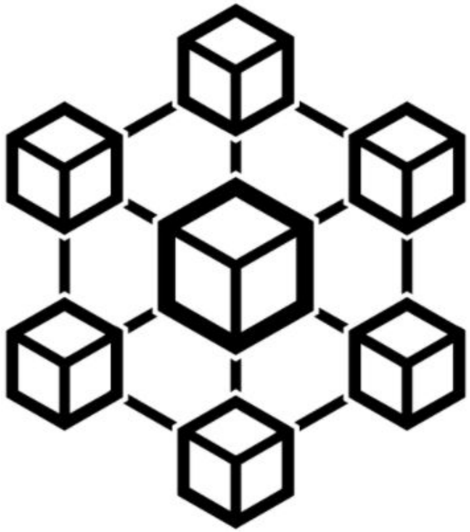
- Previous monolithic design is gradually replaced by a set of well defined loosely coupled services communicating via APIs.



# Microservices architecture

The benefits of Microservice architecture:

- Services are smaller and easier to maintain.
- Services are independently deployable.
- Services are independently scalable.
- Better fault isolation.

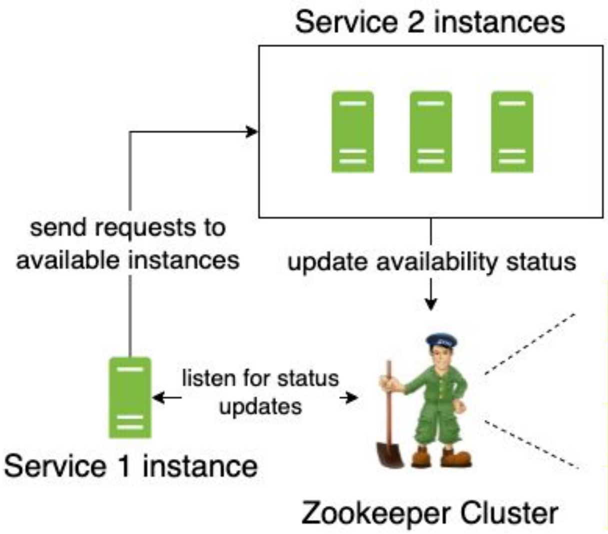


The drawbacks of Microservice architecture:

- It’s difficult to find right services division.
- Deployment and testing may be more complicated.
- Complex features that span multiple services may require increased coordination.

# Service discovery

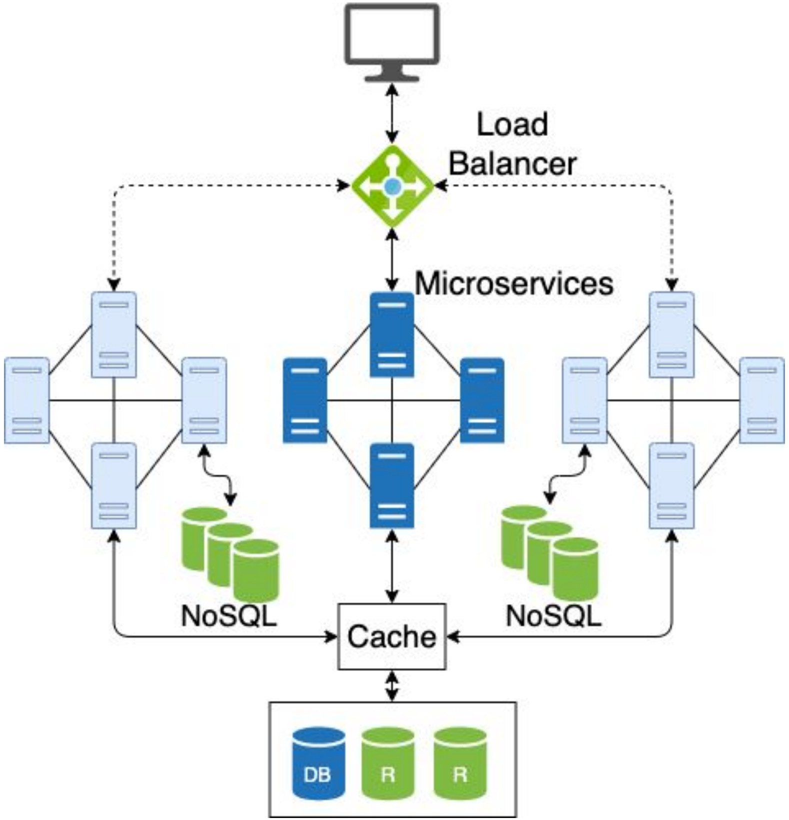
- Independently scaled must be able to know where to find instances of dependencies.
- Coordination components like ZooKeeper may implement service discovery pattern.
- Registered instances are given individual ephemeral nodes.
- Ephemeral nodes disappear on instance deregistration, lost connection, etc.




| Path | Value |
|---|---|
| /service2/192.168.1.100 | {"port": 8080} |
| /service2/192.168.1.101 | {"port": 8080} |
| /service2/192.168.1.102 | {"port": 9080} |

# Alternative storage

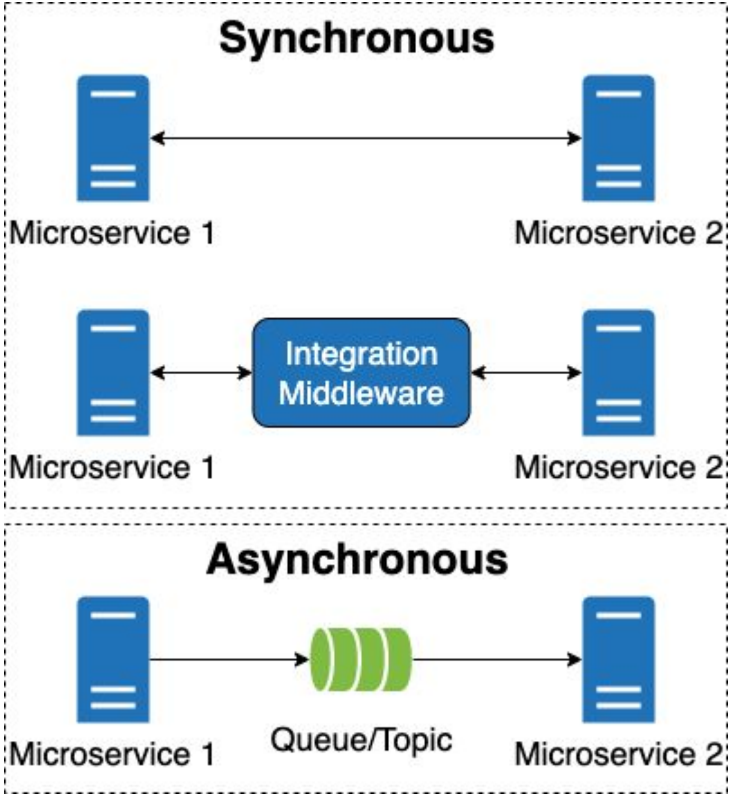
- The introduction of loosely coupled services enables the use of dedicated data stores for individual services.
- These stores can be optimized for a particular purpose of a particular service.
- Stores may be either dedicated RDMBses or NoSQL databases.



# Communication

Splitting monolithic architecture into microservices requires communication between services:

- Direct synchronous RPC calls:
    - REST, SOAP
    - gRPC, Thrift, Avro
- Synchronous calls via RPC based integration middleware.
- Asynchronous communication via message-oriented middleware:
    - ActiveMQ, RabbitMQ
    - ZeroMQ
    - Apache Kafka - events streaming platform

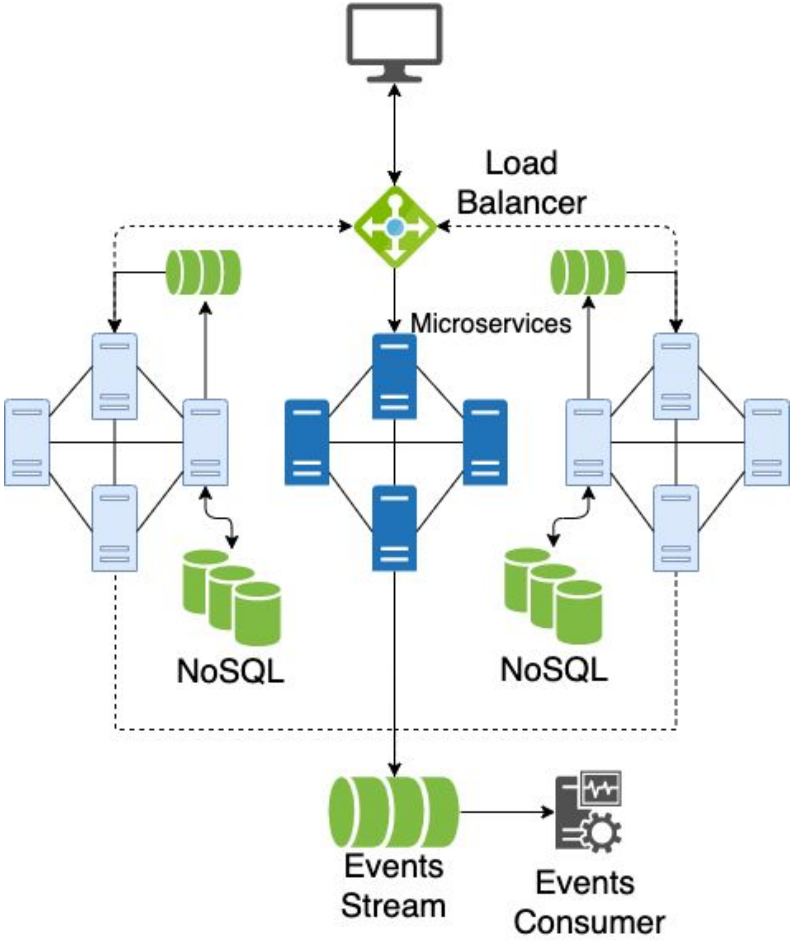


# Event Driven Architecture

**Event** - immutable statement of fact that something happened in the past.

**Event Driven Architecture** - services can publish an event message that another service can use to perform one or more actions in turn.

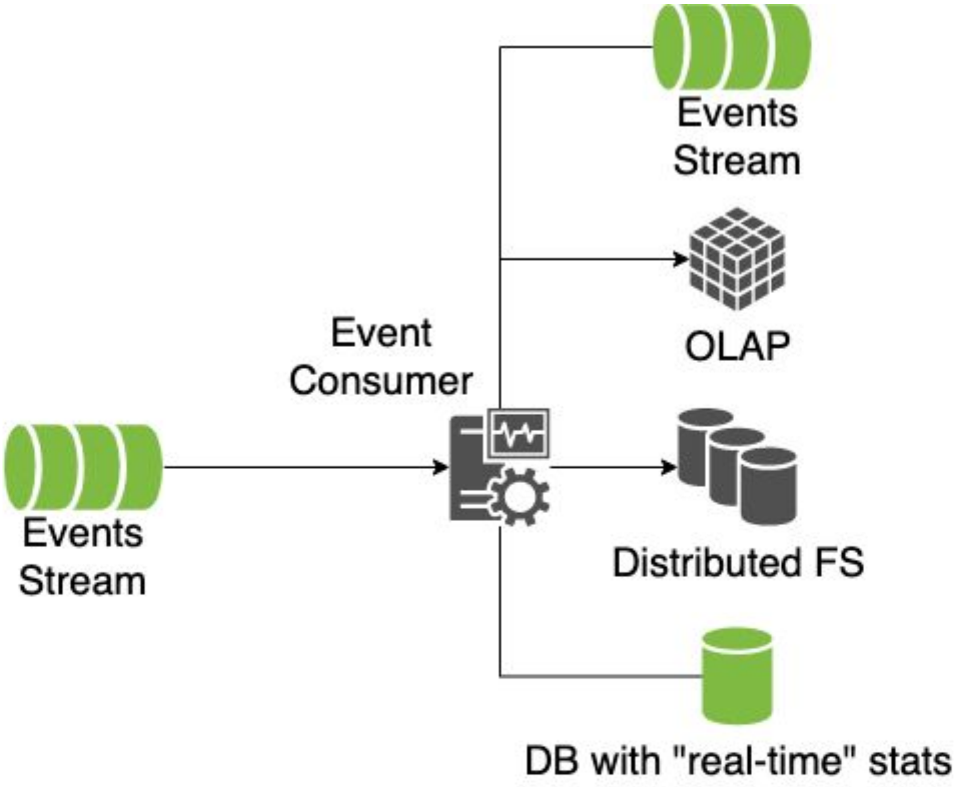
**Event streaming -** services can publish streams of events to a broker. Consumers can access each stream and consume their preferred events, and those events are then retained by the broker.




Practical Distributed Systems, 2022

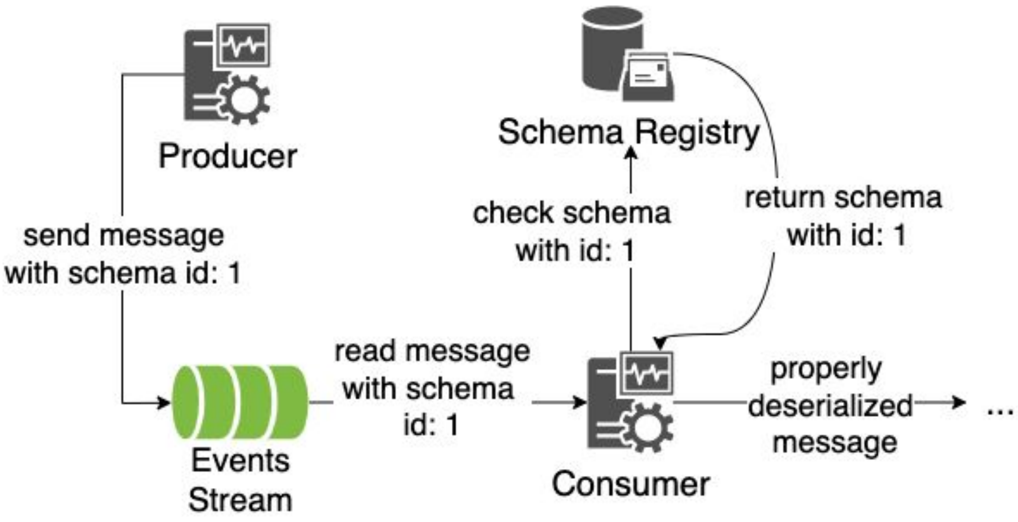
# Stream processing

- Consumed events may result in emission of a different kind of event that may be consumed by separate consumer.
- Events may be stored in a variety of datastores for warehousing purposes.
- Or may provide source data for various kinds of real-time analytics.



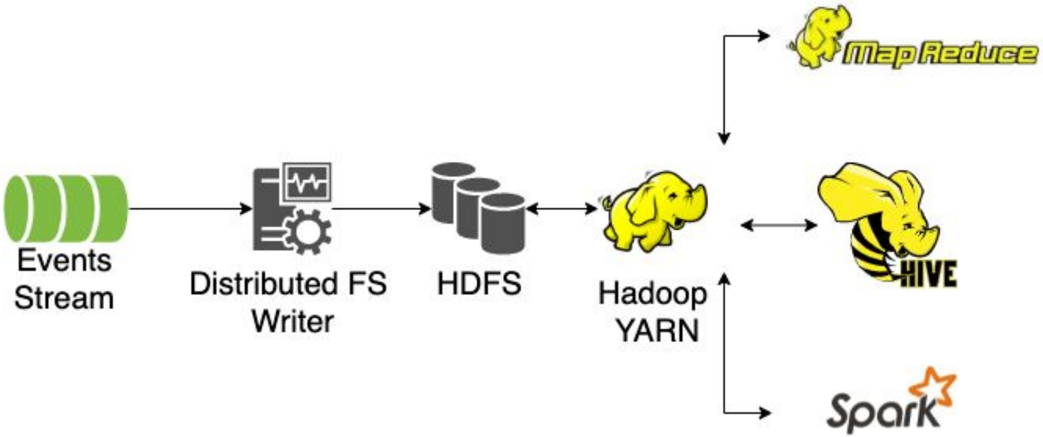
# Event schema

- In some scenarios event messages may be defined with a formal schema.
- Event schema may evolve over time.
- Consumers need to know how to deserialize messages according to the schema in which they were published.
- Schema registry serves information about  available event schemas.



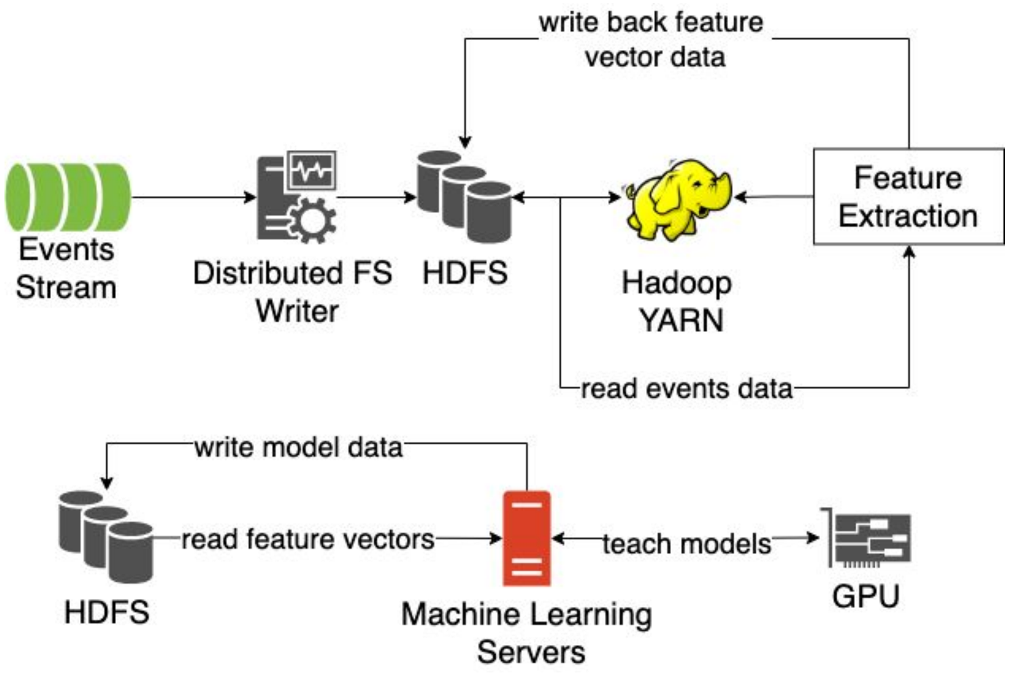
# Big Data processing

- One of the typical destinations for streamed events data is to store it on a distributed FS like HDFS.
- Data can be further processed by various tools available within the Hadoop ecosystem.
- The concept of storing large amount of data in raw format on a distributed file system is sometimes called a “data lake”.



# Machine learning

- The raw data on HDFS has to be transformed by the feature extraction workflows and stored back for Machine Learning purposes.
- Usually distinct high end machines with GPUs are used for Machine Learning.
- The feature sets are copied to machine learning servers for performance reasons.

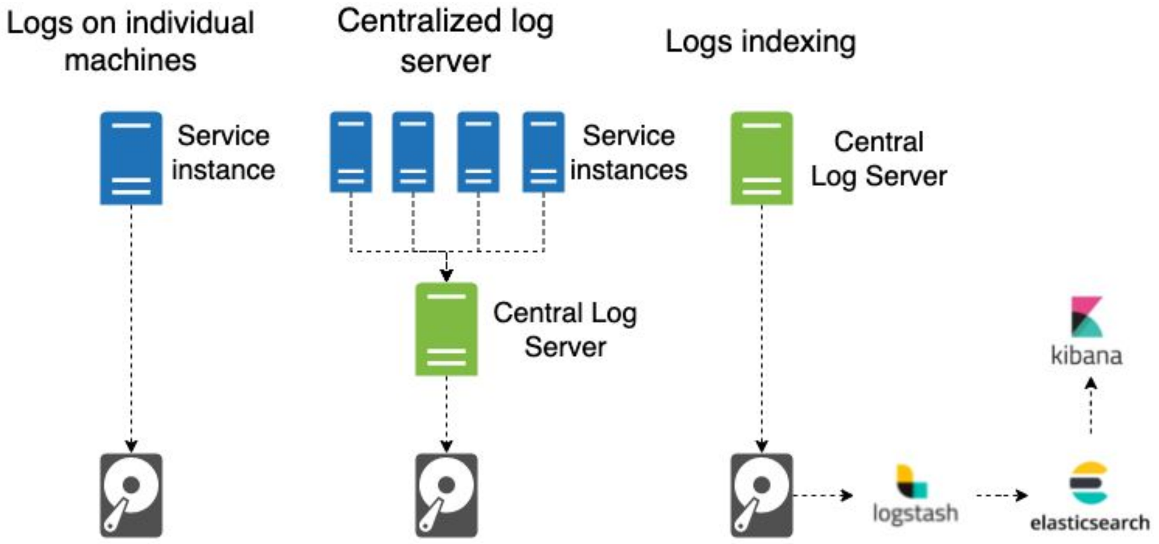


# Streaming to Cloud

- Alternatively the events data may be streamed to the Cloud.
- Fast Querying over large datasets - BigQuery, Redshift
- Storage - Google Cloud Storage, Amazon S3

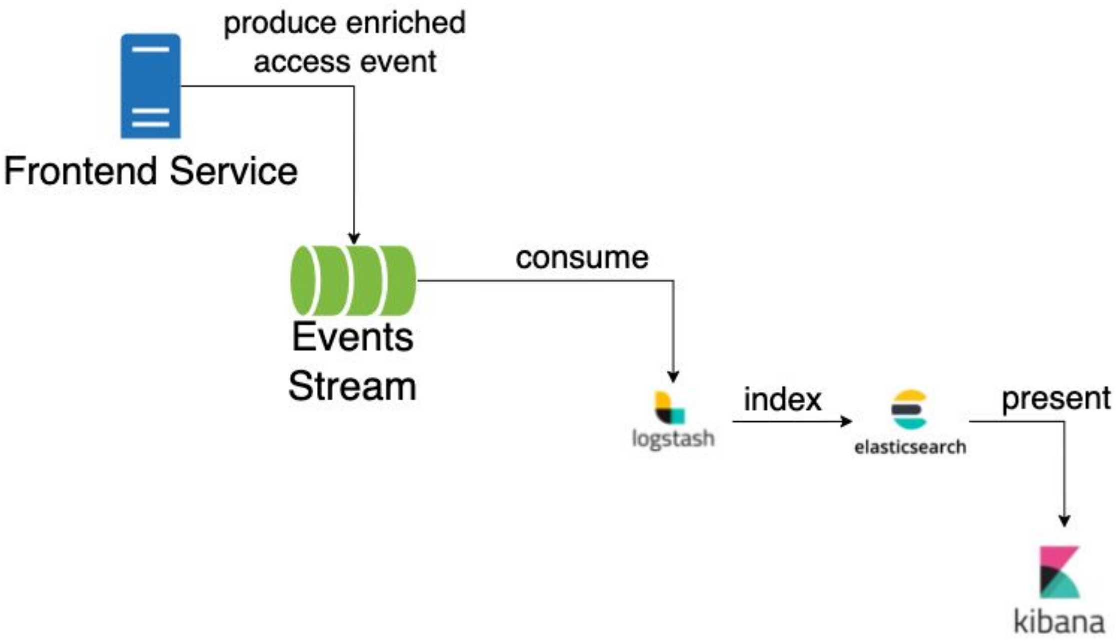
# Distributed logging

- Services write application logs on the machines where the are installed.
- It is also useful to stream logging messages to the central server (eg. via rsyslogd).
- The logs from the central server are ingested, indexed and presented using tools from ELK stack or similar.



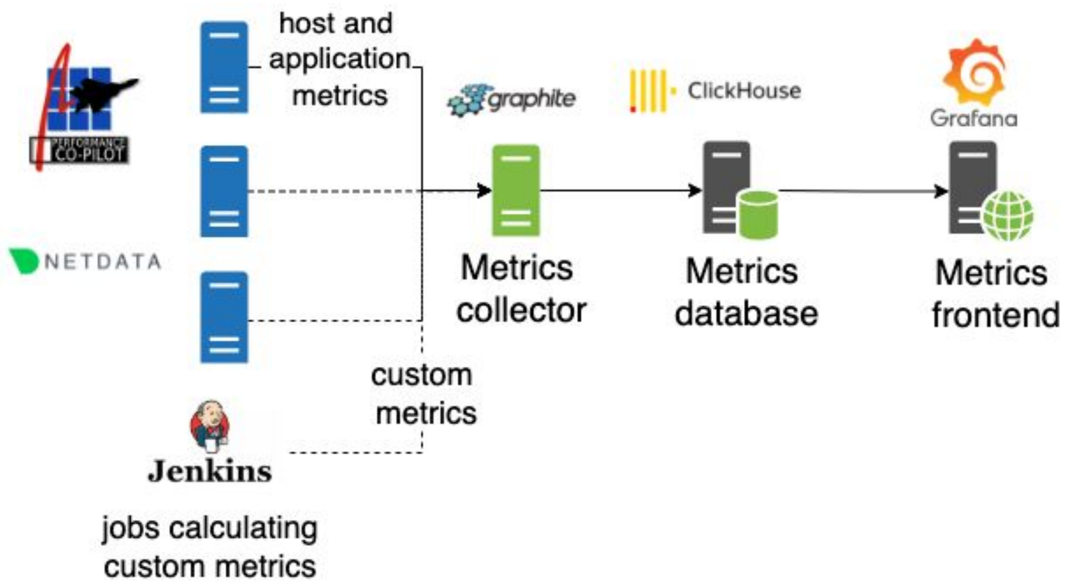
# Access logging

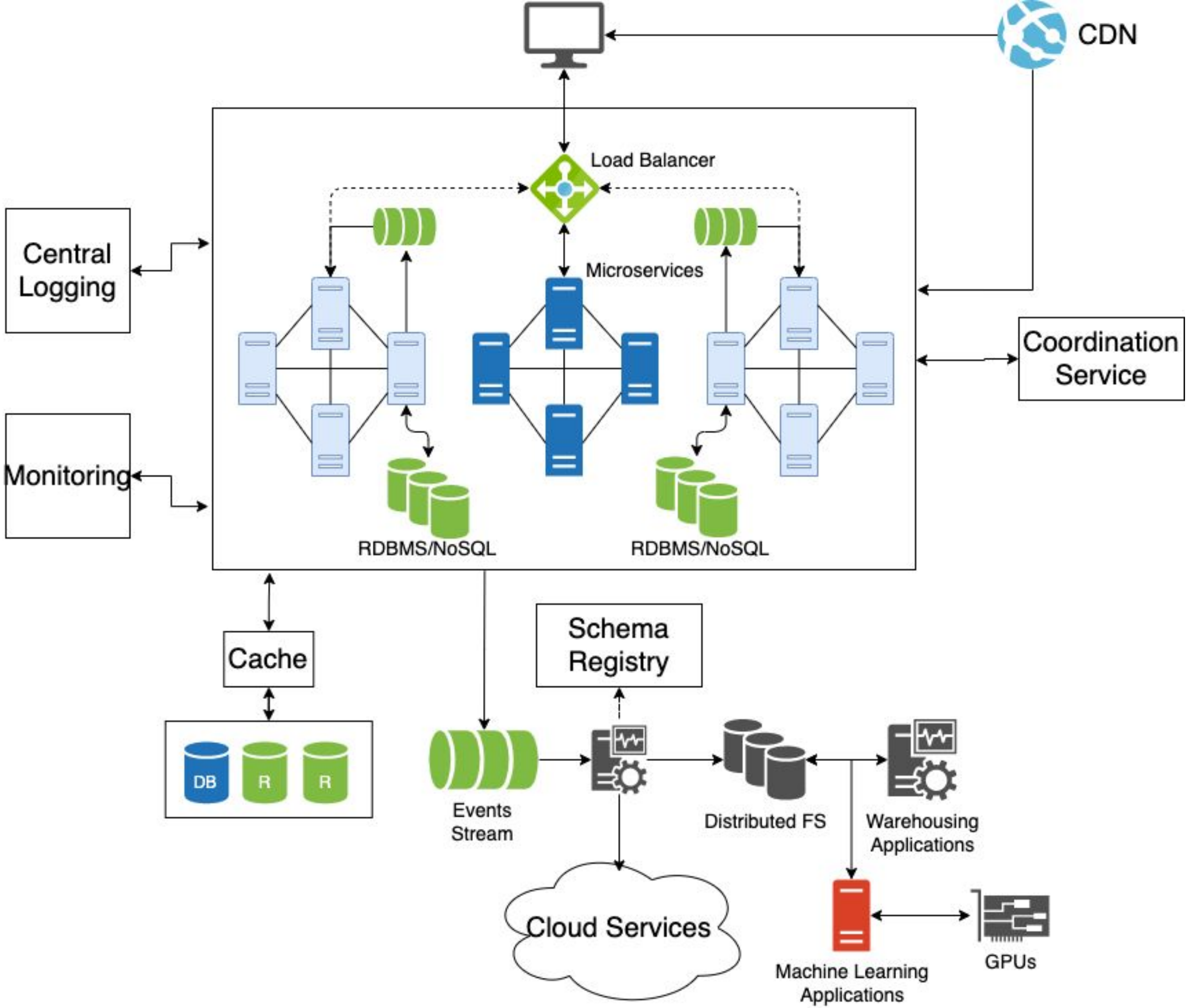
- Standard web server access logs.
- Enriched access events asynchronously sent to events stream.
- Indexing access logs via ELK stack enables the in depth analysis of the structure of incoming traffic.



# System metrics

- Host metrics - technical hardware/OS.
- Application metrics - application software, either technical or business oriented.
- Custom metrics - calculated when standard host or application metrics are not available.
- Metrics are also useful for resource usage predictions and thus preventive maintenance.

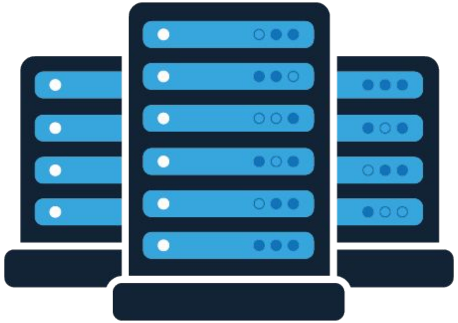





Practical Distributed Systems, 2022

# Datacenters

- On premise - within organization premises:
  - full control
  - high customizability, low flexibility
  - very high costs
  - no SLA guarantees
- Colocation center - leased equipment in a rented facility:
  - acceptable control
  - acceptable customizability, moderate flexibility
  - moderate costs
  - some SLA guarantees
- Public Cloud infrastructure:
  - low control
  - low customizability, high flexibility
  - costs depending on usage
  - very low downtime risk



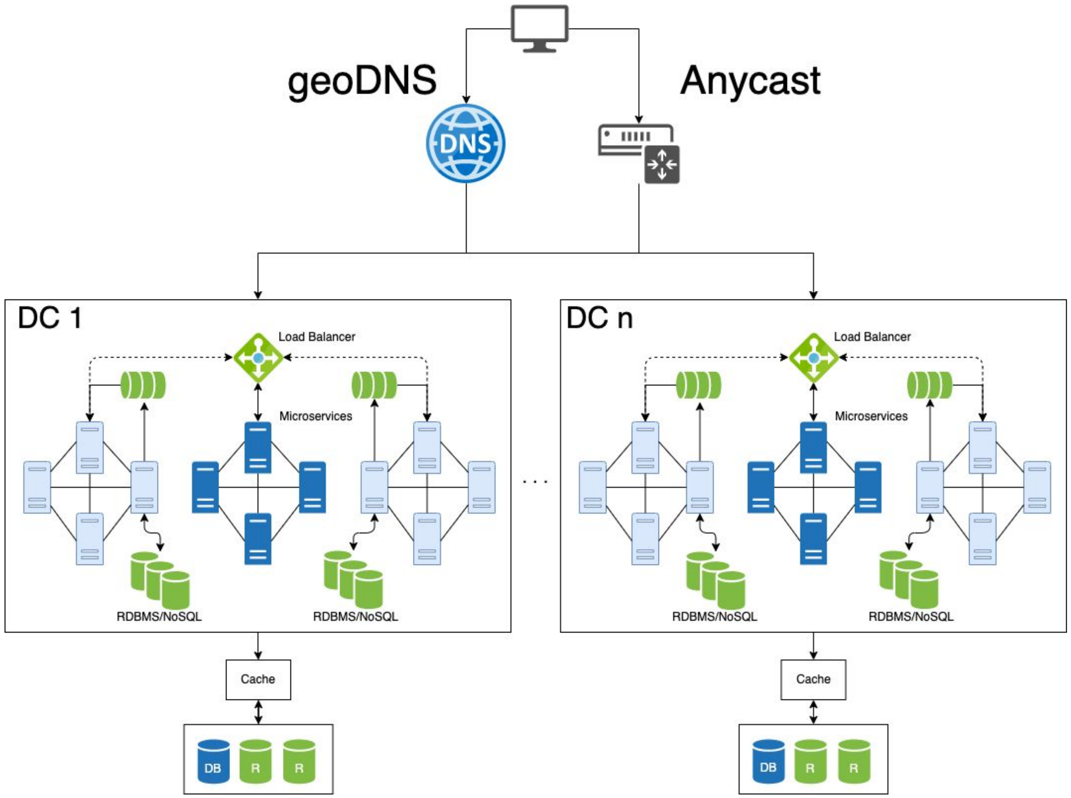
# Cloud considerations

Why just don't move everything into a public Cloud?

- Limited possibility of low level optimizations.
- Shared and unpredictable networking.
- Relatively high costs for high-performance virtual machines and egress network traffic.
- Tedious process of going through customer support in case of non-trivial issues.

# Datacenter traffic distribution

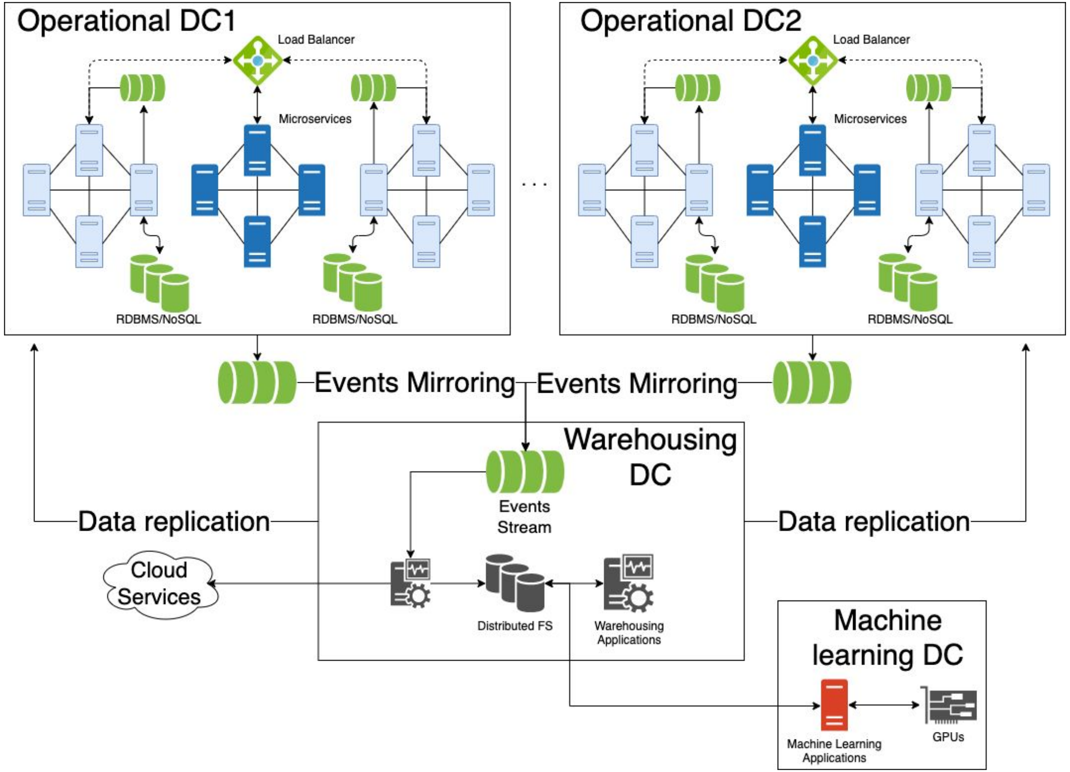
- Traffic affinity - traffic is pretargeted to a particular DC.
- geoDNS - dedicated DNS service redirects traffic to the nearest datacenter based on the approximate client location.
- Anycast - IP addresses are shared between DCs and traffic is redirected based on network router decision for the shortest path.




Practical Distributed Systems, 2022

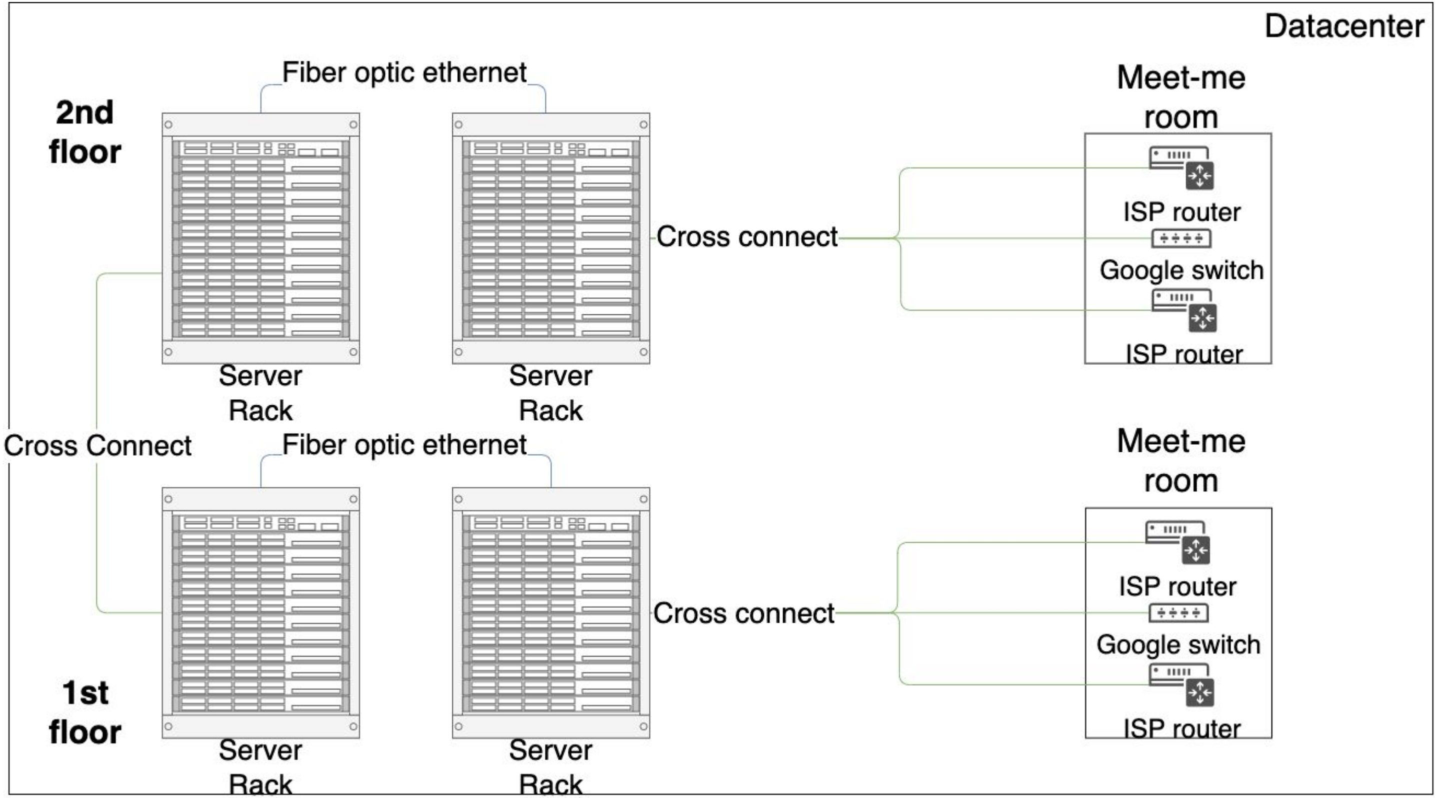
# Datacenter specialization

- Individual DCs don't have to be equivalent.
- Apart from resources size, DCs may also differ in dedicated purpose.
- Operational DCs with frontend services.
- Warehousing and analytical DCs with events stream processing components.
- Machine learning oriented DCs with GPUs.

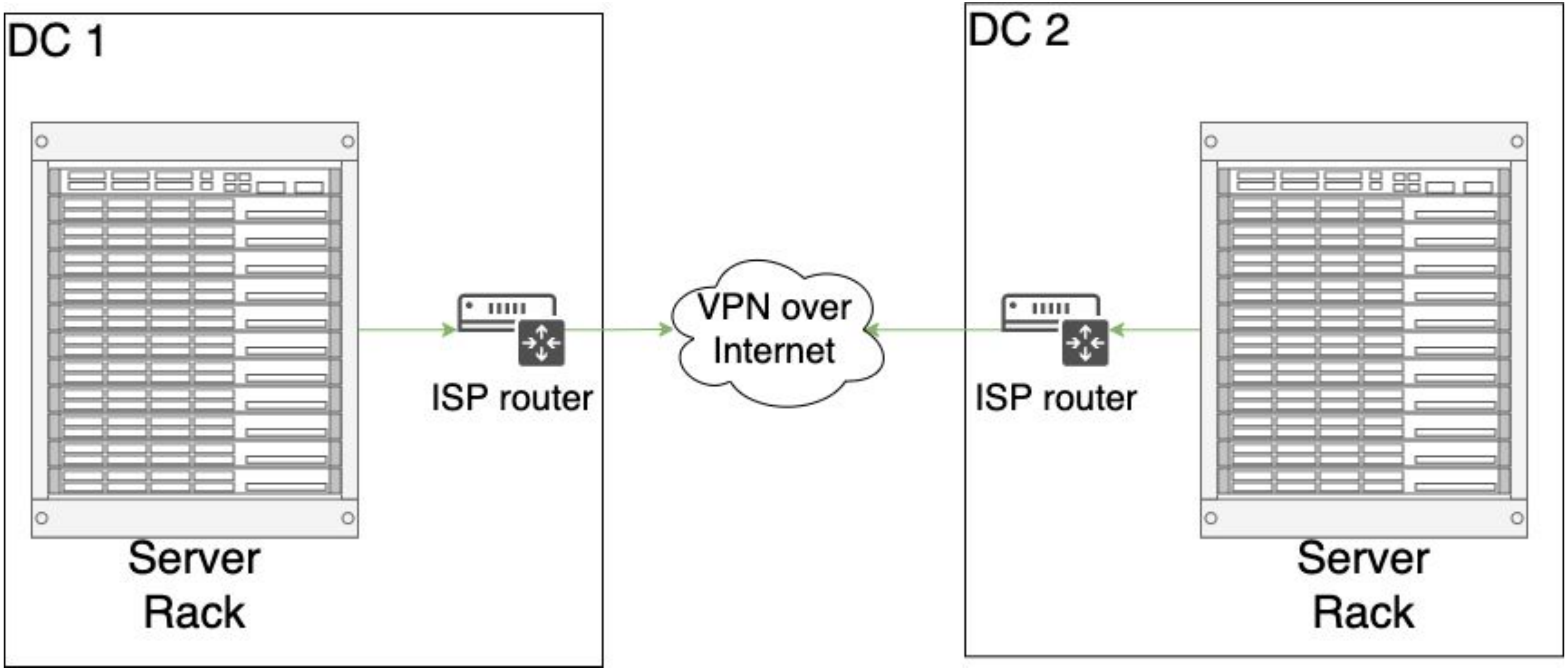



Practical Distributed Systems, 2022

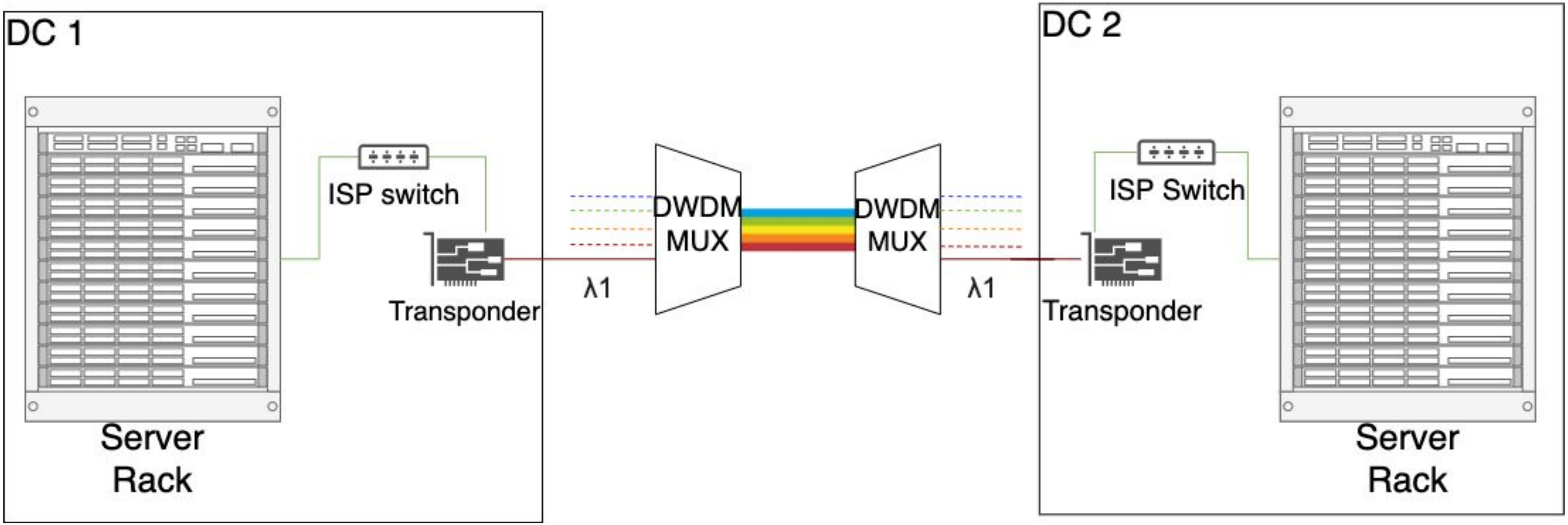
# Datacenter networking



# Datacenter networking



# Datacenter networking

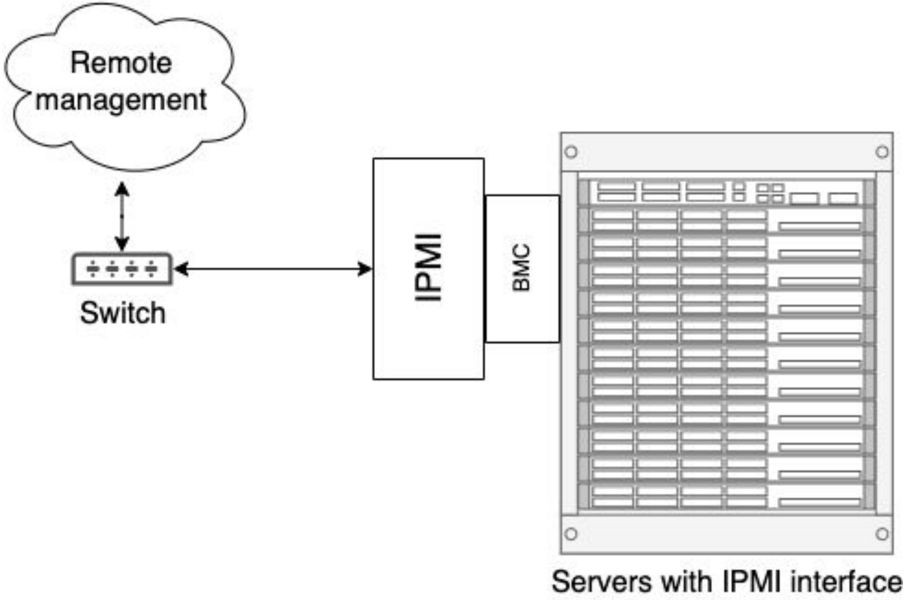



DWDM - Dense Wavelength Division Multiplexing

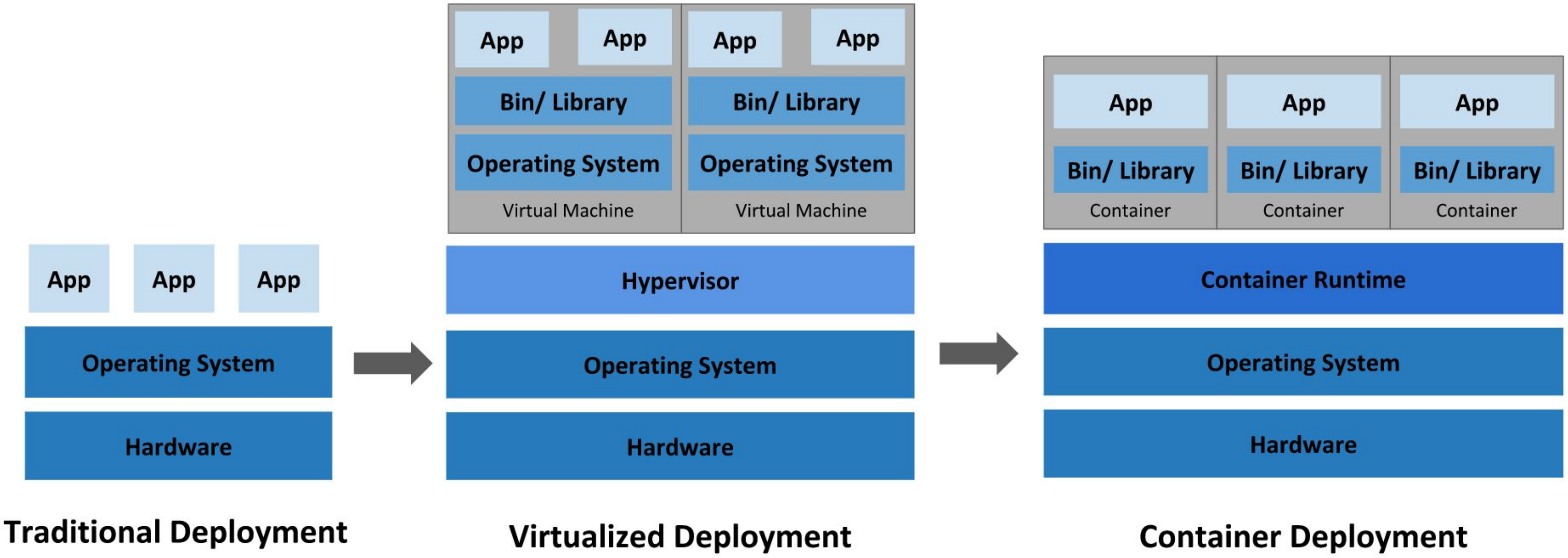
# Datacenter operations

BMC - Baseboard Management Controller

- “Remote hands” - onsite personnel to perform physical infrastructure management.
- Remote management via IPMI (Intelligent Platform Management Interface).
- Remote management via infrastructure automation software - Puppet, Ansible, etc.

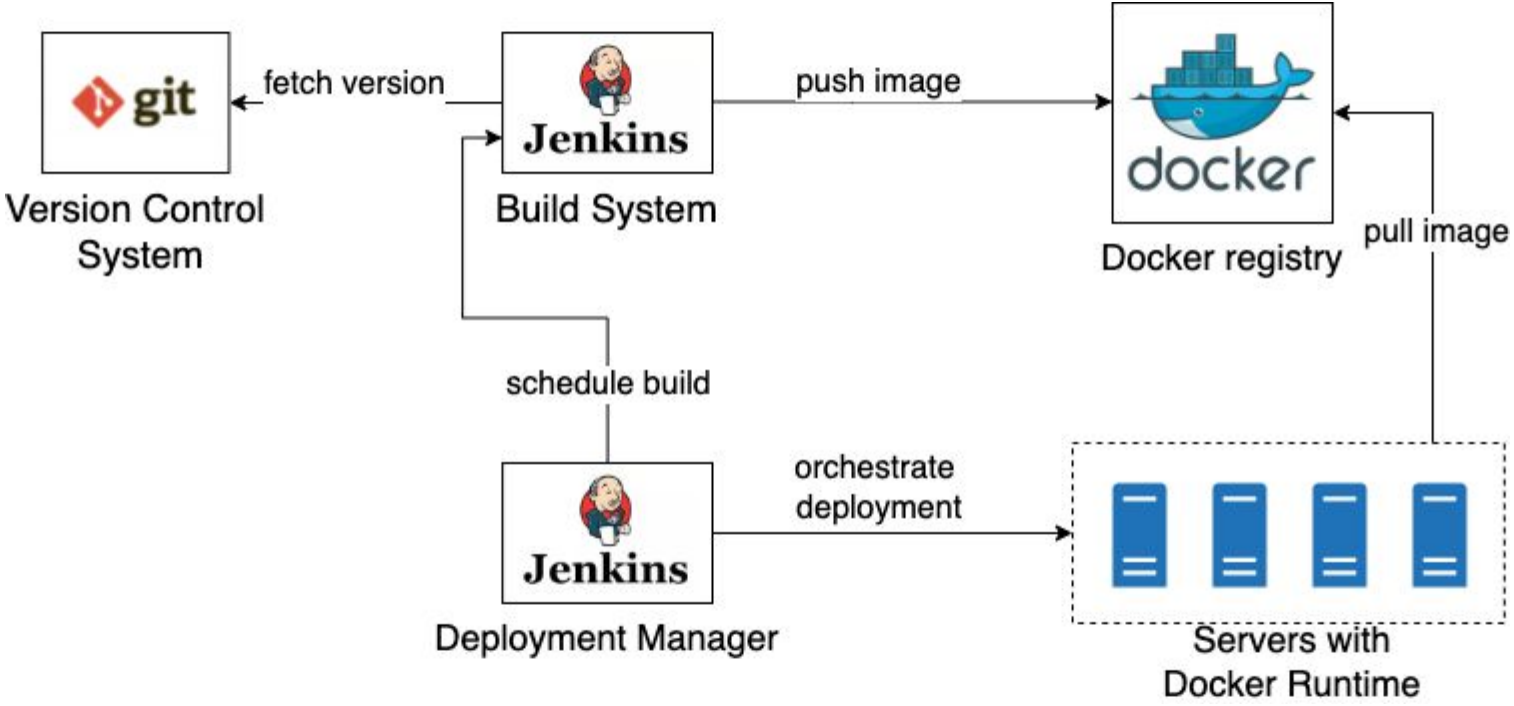


# Application deployment

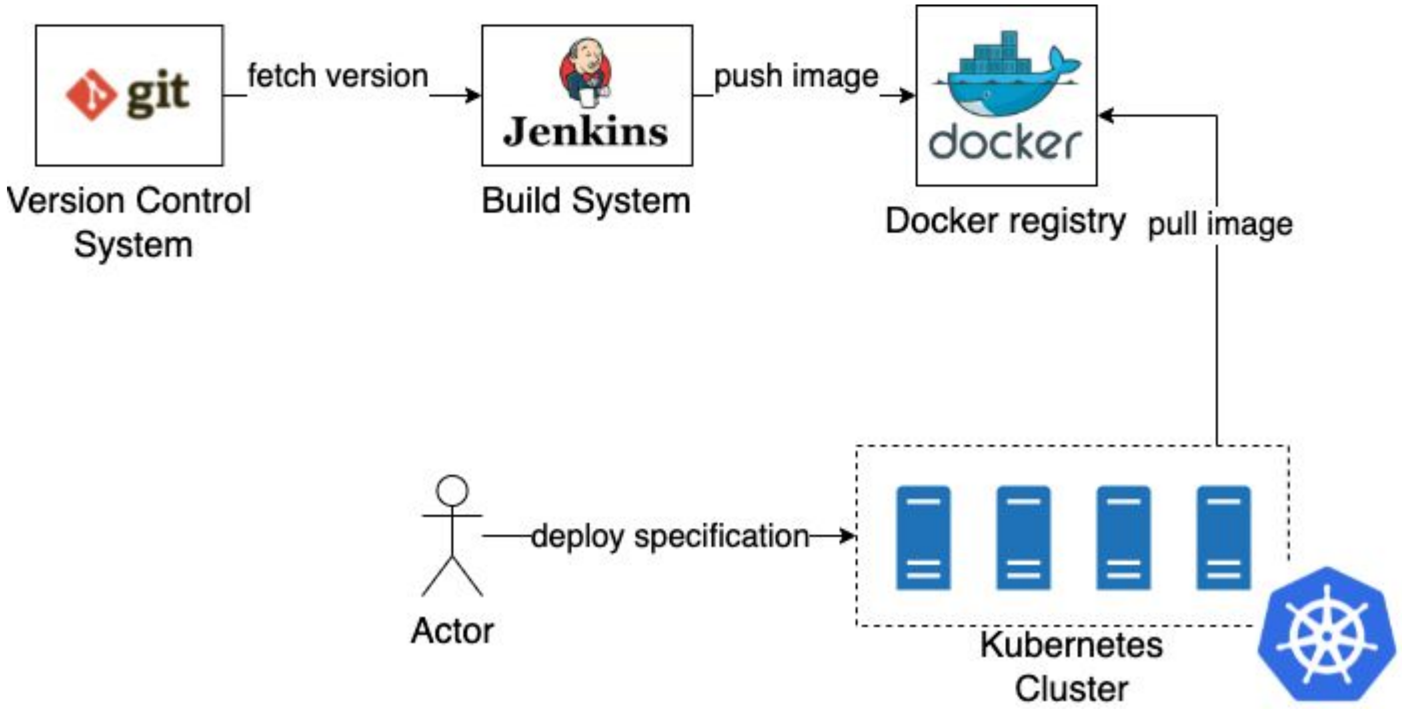



Source: kubernetes.io

# Application deployment



# Application deployment



# Performance considerations

Hardware optimizations:

- Fast and reliable networking in Data Centers.
- High-end servers with NVMe storage.

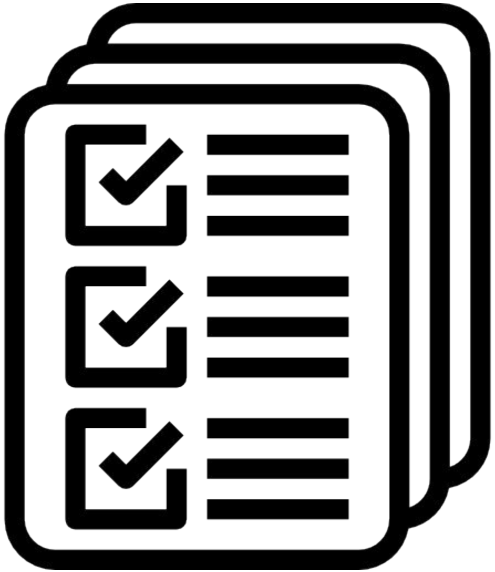
Software optimizations:

- Caching as much as possible.
- Garbage Collector optimization.
- Reduce malicious or low quality traffic.
- Using compression.
- Mechanical sympathy.

# Summary

We have discussed:

- The overview of a generic architecture of a web based modern distributed system and its components.
- The basics of various maintenance related aspects of distributed systems like logging, metrics collection and application deployment.
- The basics of Data Center design, operations and networking.
- Some performance considerations.



# Evaluation

Evaluation of the course will be solely based on the project:

- Will have to run on infrastructure shared by RTB House.
- The assessment of the project will be mostly focused on a distributed systems features like scalability, reliability and maintainability.
- Can be done in pairs.
- Full specification will be revealed in April.