DevOps – cloud native

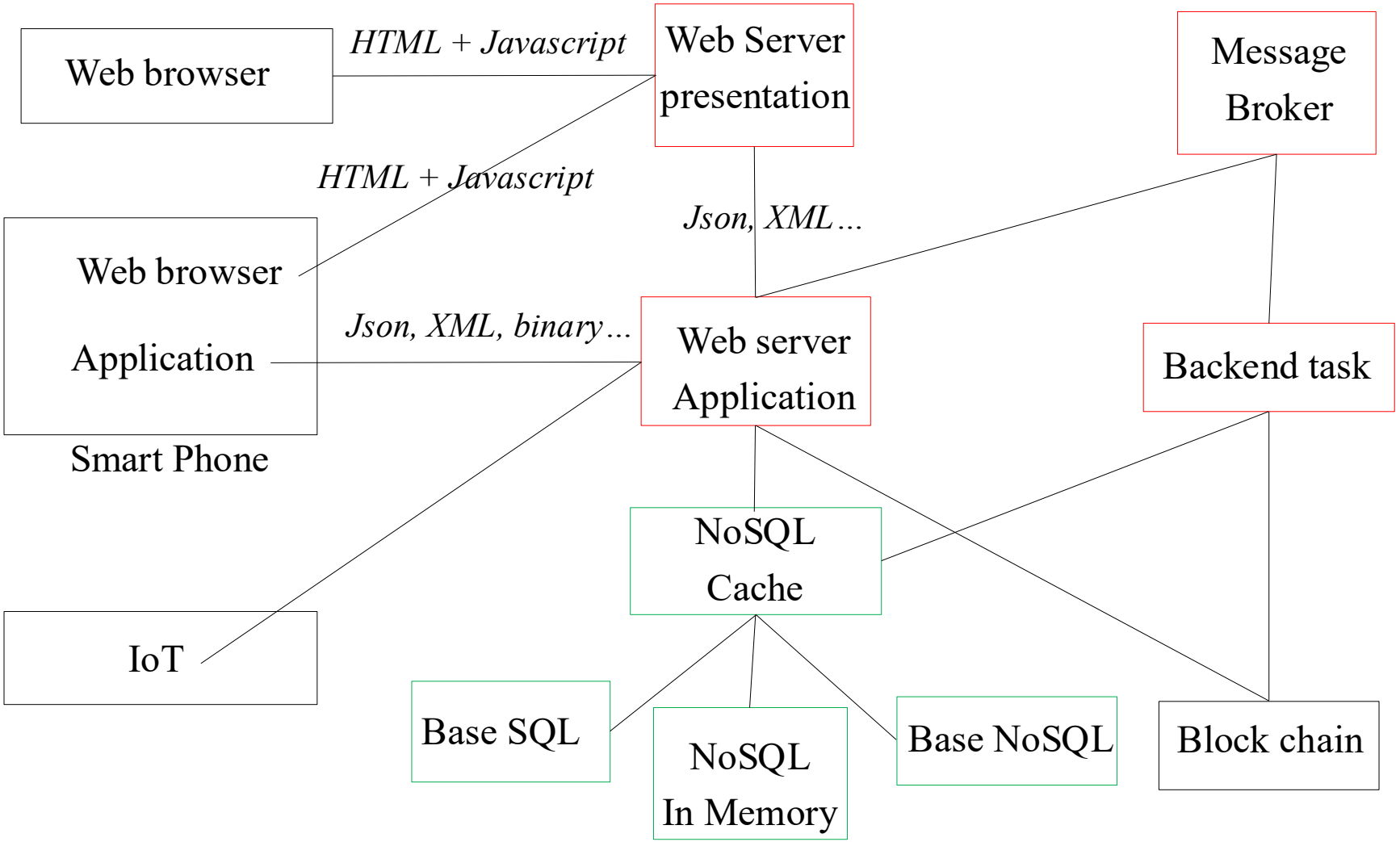
Benoît CHARROUX

Summary

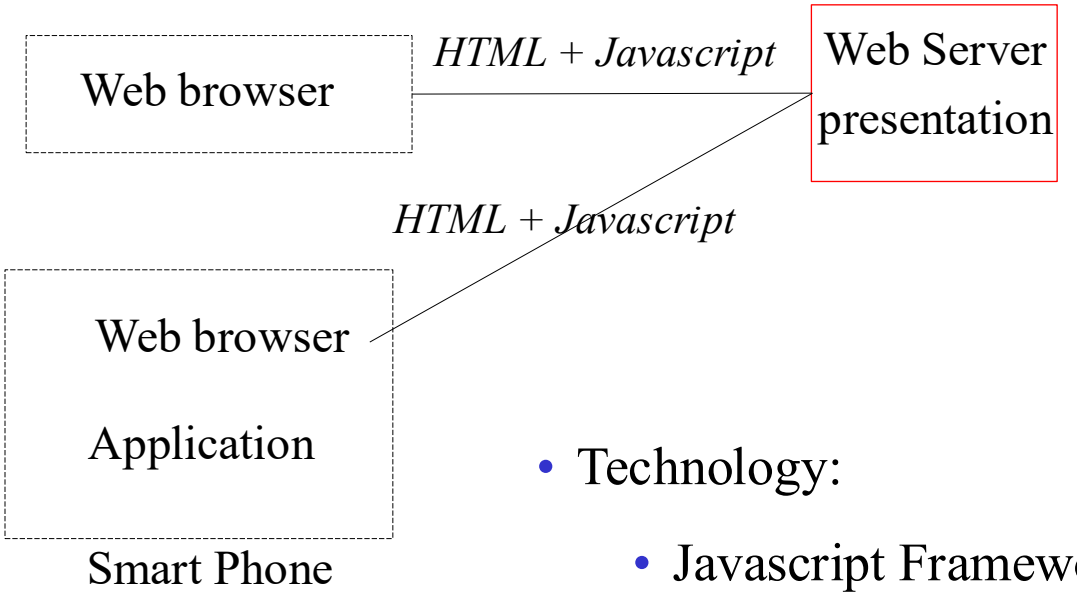
- The big picture of the web services
- Web services
- Microservices
- Applications cloud native

The big picture of the Web services

Form the end user to the database



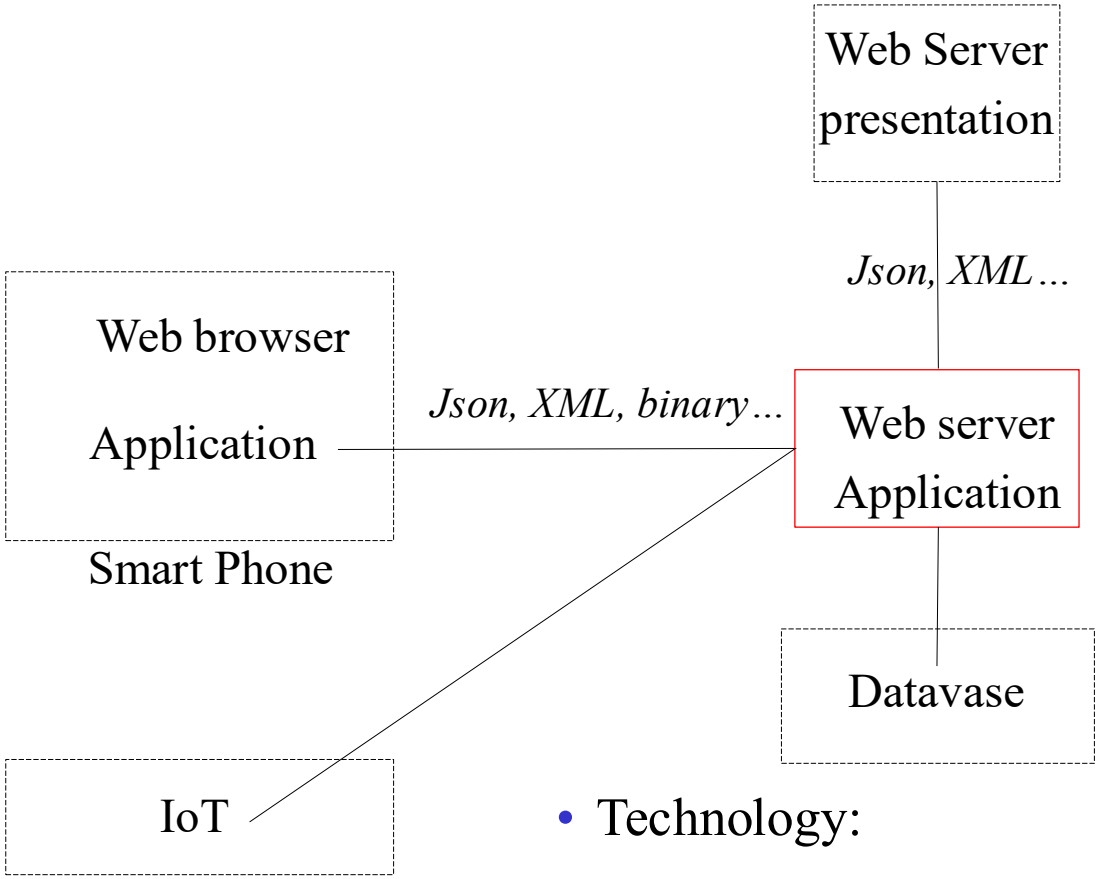
Presentation server



- Technology:
  - Javascript Framework (Angular, React, VueJS…)
  - Example angular:

```
<div *ngFor="let item of items" [item]="item"></div>
```

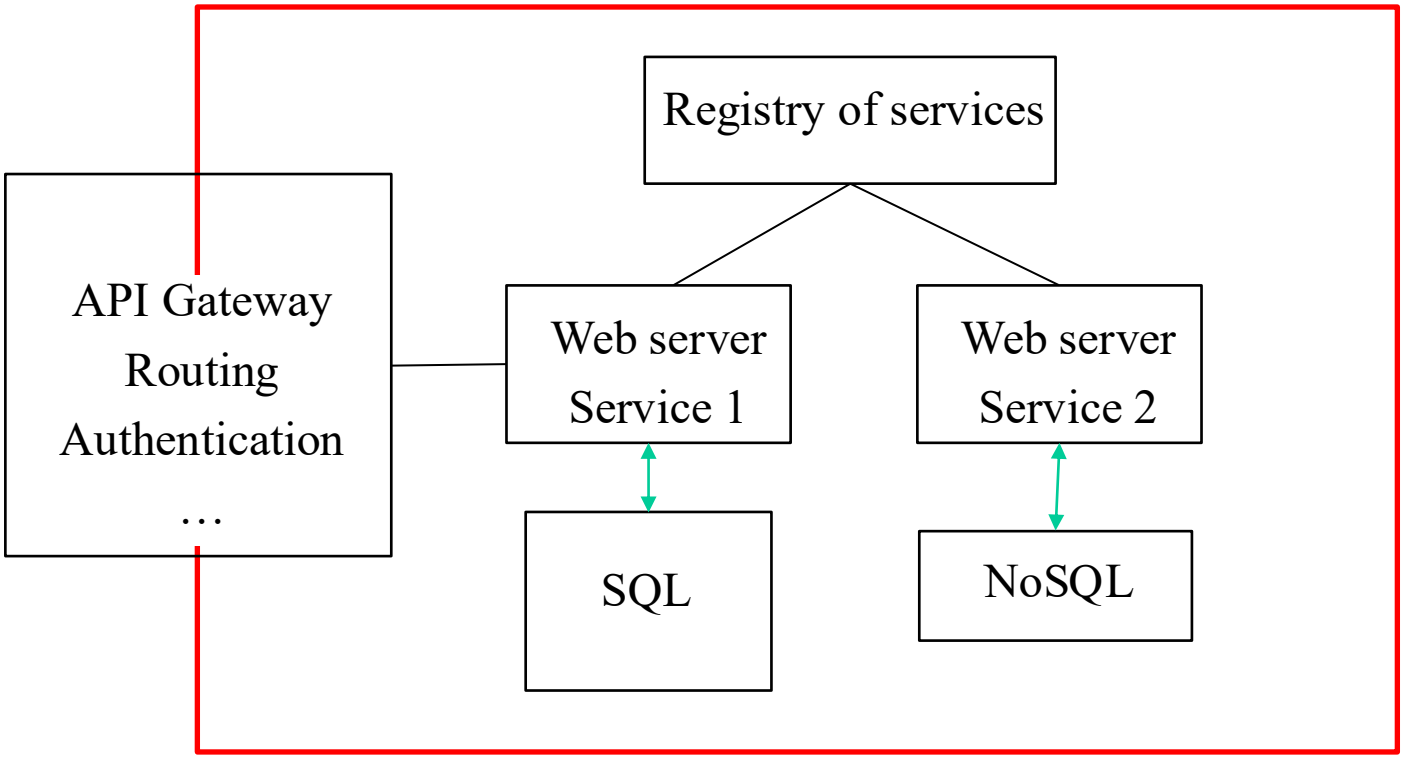
Service




```
{
"firstName": "John",
"age": 27,
}
```

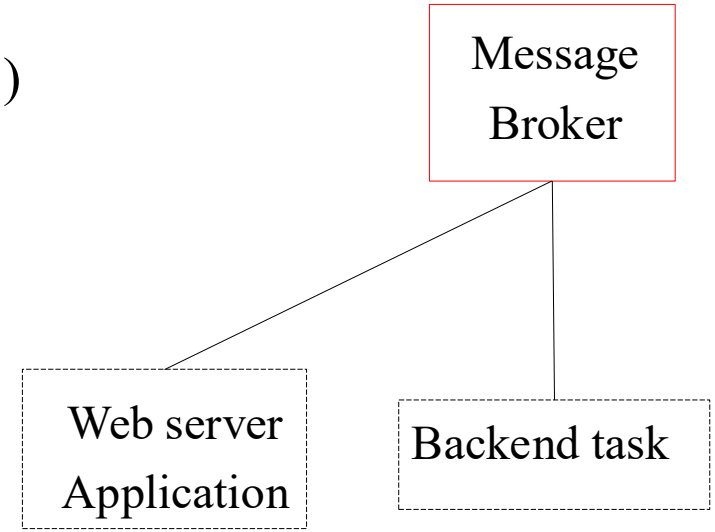
- Technology:
  - Web Service Rest
  - Remote Procedure Call
- Multiple languages: Java, Node.js, Python…

Microservices



Asynchronous microservices

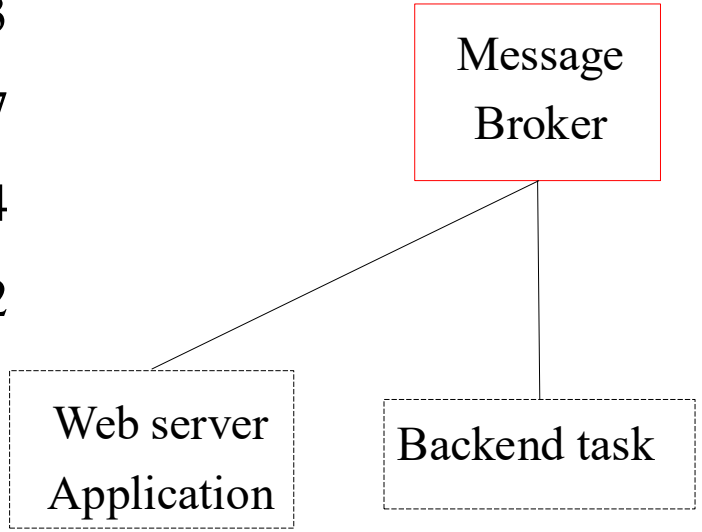
- Technology:
  - Message brokers (Kafka, RabbitMQ…)
- Properties:
  - Asynchronous
  - Message Driven, Event oriented
  - Low latency
  - High availability
  - Fault tolerance
  - Durable
  - Trillions of messages par day
  - Multi languages: Java, Python…



Event Driven Systems

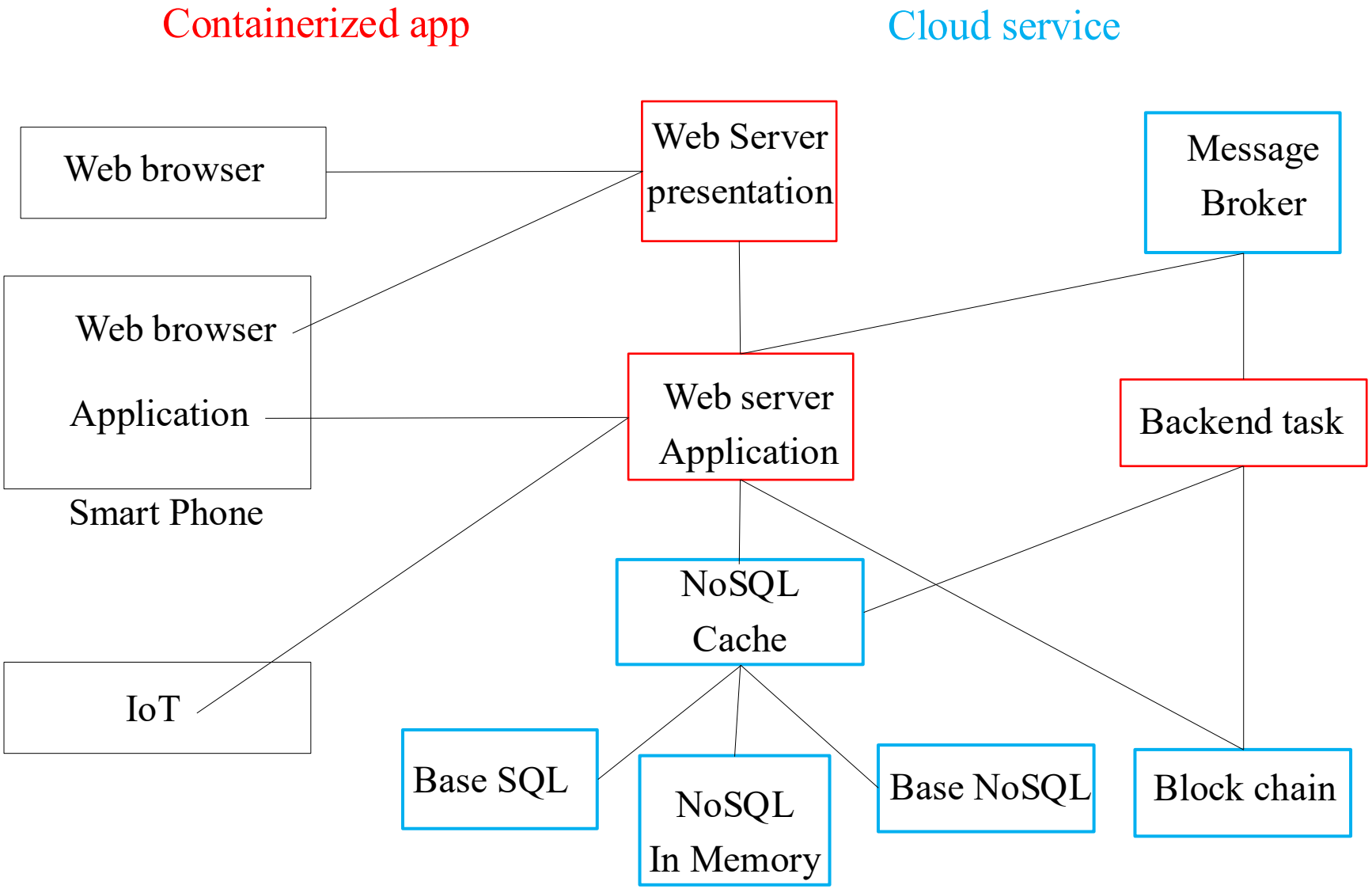
- Messages contain events

Paul	added	3 pants	12:13
Emilie	added	1 T-shirt	12:27
Paul	removed	1 pants	13:04
Emilie	added	1 hat	14:12



- Stream processing

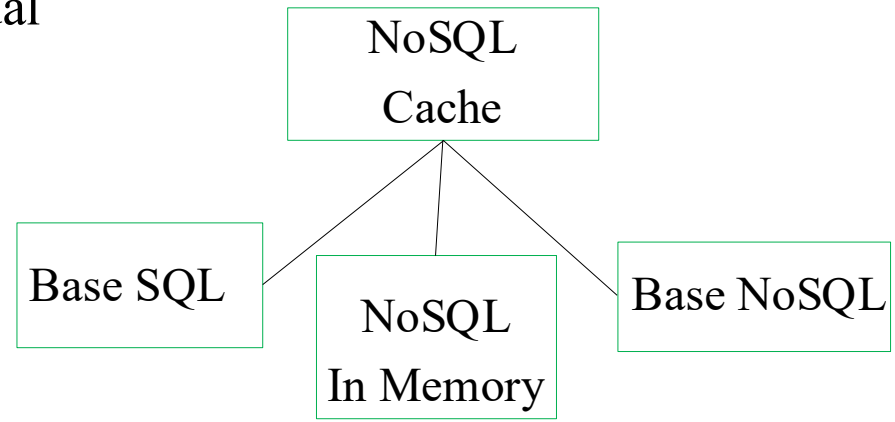
Cloud native



Database

- SQL:
  - Standard query language
  - Low but ACID
- NoSQL:
  - Supports heterogeneous data
  - Simpler horizontal scaling to cluster
  - Speed but eventual consistency

- NoSQL:
  - Key-value store => DynamoDB
  - Document store => MongoDB
  - Object base => CouchDB
  - Graph => Neo4J
- Cache:
  - Key-value store => Redis



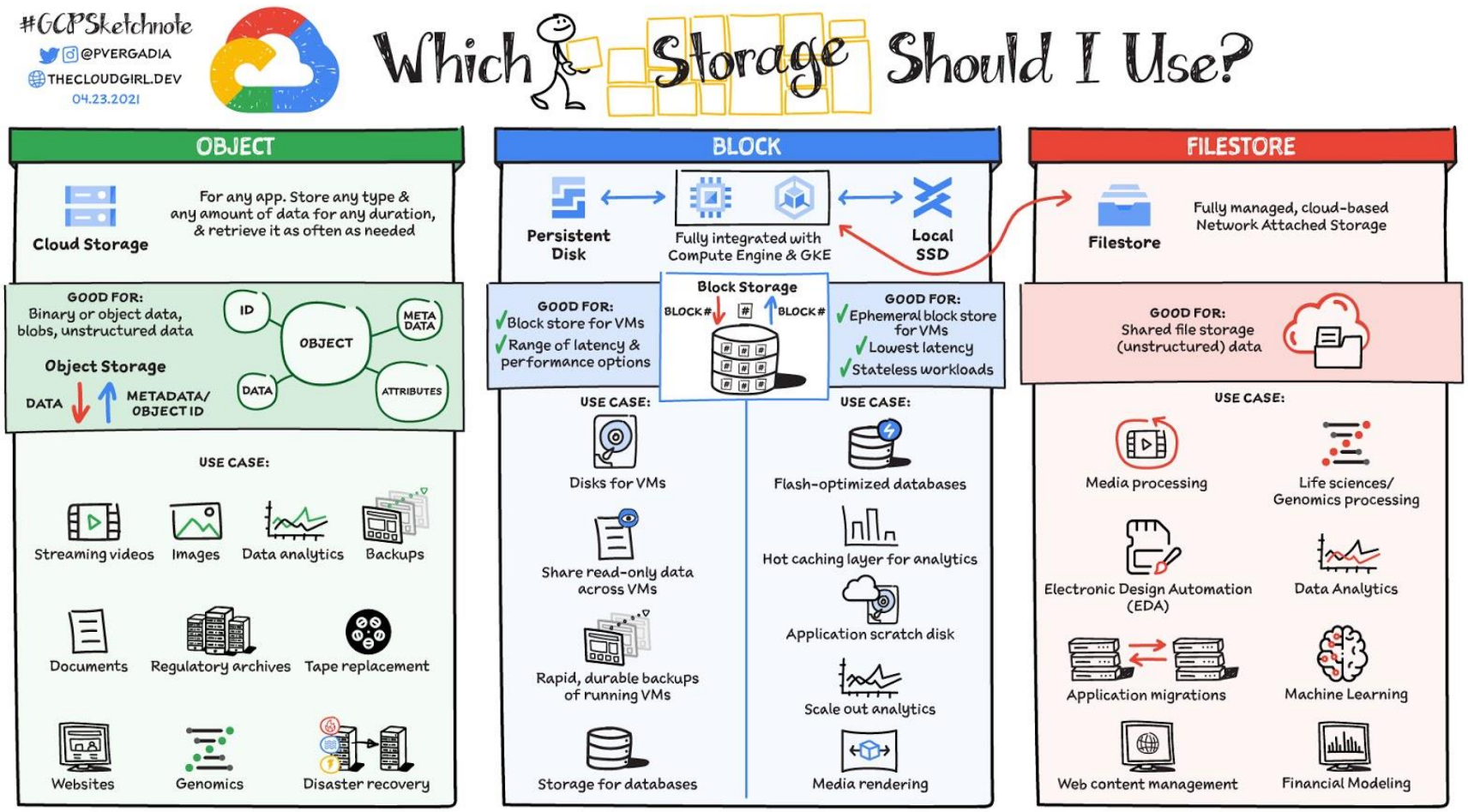
Database and Cloud providers

	AWS	Azure	Google Cloud Platform
Relational Database	RDS Instance de DB relationnelle (MySQL, PostgreSQL, Oracle, SQL Server, MariaDB) RDS on VMware	Azure Database (MySQL, PostGreSQL, SQL Server, MariaDB) **SQL Edge**	Cloud SQL (MySQL, PostGreSQL, SQL Server), BareMetal
NoSQL	DynamoDB / Keyspaces NoSQL DB / Cassandra	Table Storage / Azure Managed Instance for Apache Cassandra NoSQL DB / Cassandra	Firebase Realtime Database
Cache	Elasticache		MemoryStore
Warehouse	Redshift	Datalake gen2	BigQuery Looker
High performance database	Aurora & Aurora Serverless		Cloud Spanner
Other	Neptune (BD de graphes), Quantum Ledger (transaction sécurisée), Timestreams (time series), DocumentDB (MongoDB)	Cosmos DB (NoSQL), **Azure confidential ledger (preview)**	BigTable (colonne), Firestore (Document)
In memory	MemoryDB for Redis	Azure Cache for Redis	Memory Store

Storage

	File	Block	Object
Accessibility	File server via OS/protocole	Low level interface	HTTP (REST)
Examples	NFS, SMB… Hadoop FS…	SAN iSCSI, FC… Amazon Elastic Block Storage, Openstack Cinder, Ceph RADOS…	Amazon S3, Openstack Swift, Ceph Storage…

Use case for storage



Service providers offers

	AWS	Azure	Google Cloud Platform
Object storage	Amazon Simple Storage Service	Blob	Cloud Storage
Block storage	Elastic Block Store	Disk	Persistent Disk **Local SSD (éphémère)**
File storage	Elastic File System, FSx for Lustre, FSx for Windows File Server, NetApp ONTAP, **OpenZFS**	File (SMB), Netapp Files	File Storage (NFS)
On-premises gataway	AWS Storage gateway	StorSimple	
Backup	AWS Backup	Azure Backup	Actifio
Cold storage	Aamazon S3 Glacier (storage classes)	Archive Storage	Archive Storage
Disaster Recovery	CloudEndure	Azure Site recovery	Actifio

Summary

- The big picture of the web services
- **Web services**
- Microservices
- Applications cloud native

Web Services

Restful architecture

Remote Procedure Call

Definition

- OASIS defines a service as "a mechanism to enable access to one or more capabilities, where the access is provided using an interface and is exercised consistent with constraints and policies as specified by the service description.

Definition

- A Web Service is a program allowing communication and data exchange between applications and heterogeneous systems in a distributed environment:
  - Representational State Transfert (REST):
    - Simple and standard
  - Remote Procedure Call:
    - SOAP:
      - Complex and for legacy systems
    - gRPC:
      - HTTP/2 : asynchronous (temporal recoupling)

REST Web Services

What is REST ?

- **Representational State Transfert** is an abstraction of the architecture of the World Wide Web :
  - States are represented in Json, XML…
  - Transfert through HTTP
- Proposed by Roy Fielind in 2000 !

RESTful architecture

- The 4 rules:
  - Each resource has a unique URI
  - Resource are represented in Json, XML, Atom…
  - Communicate using the HTTP protocol (GET, PUT, POST, DELETE):
    - GET: send back data to the client from the server
    - PUT: update data to the server
    - POST: create data on the server side
    - DELETE: delete data on the server
  - Hypertext links to futures states and resources.

A car rental service

- The resources: cars
- Resources URI: http://www.rental.com/
- Protocole:
  - Get the list of cars to be rented:
    - URI: http://www.rental.com / cars
    - http: GET
    - Json response: [{"plaque" : "22AA33"}, {..}]
  - Get the features of a car:
    - URI: http://www.rental.com / cars / 22AA33
    - http: GET
    - Json response: {"plaque" : "22AA33"}

Design a Rest application: a car rental service

- Protocole:
  - Rent a car:
    - URI: http://www. rental.com / cars / 22AA33 ? rent=true
    - http: PUT
    - Send date: { "begin" : "…" }
  - Get back the car:
    - URI: http://www.rental.com /cars / 22AA33 ? rent=false
    - http: PUT

HATEOAS

- Hypermedia as the Engine of Application State:
  - All subsequent requests are discovered inside the responses to each request.
  - Rent a car returns links to get back the car:
    - Response:

```
{
        "links": {
                "getBackCar": "/plateNumber?rent=false"
        }
}
```

Properties of REST

- Stateless:
  - each request contains all information to understand the request.
  - The session management must be:
    - At the client side or
    - In a database
  - Every server can serve any client at any time.
- Benefit: scaling
- Clear contract is optional (unlike gRPC):
  - Swagger

The lockdown problem

- User A => GET /car/AA11BB
- User B => GET /car/AA11BB
- User A => PUT /car/AA11BB
- User B => PUT /car/AA11BB
- User A => GET /car/AA11BB => incoherent state

- Pessimistic locking: User A locks the resource.
- Drawback: how long user A keep the resource?

- Optimistic locking: Any user can get the resource but only user A can update it (Mongo DB can mark a resource with a version id).

Rest web service implementation

https://github.com/charroux/servicemesh

gRPC

HTTP/1.1

- Textual (non-binary) protocol.
- Ordered and blocking
- Requires multiple connections for parallelism.

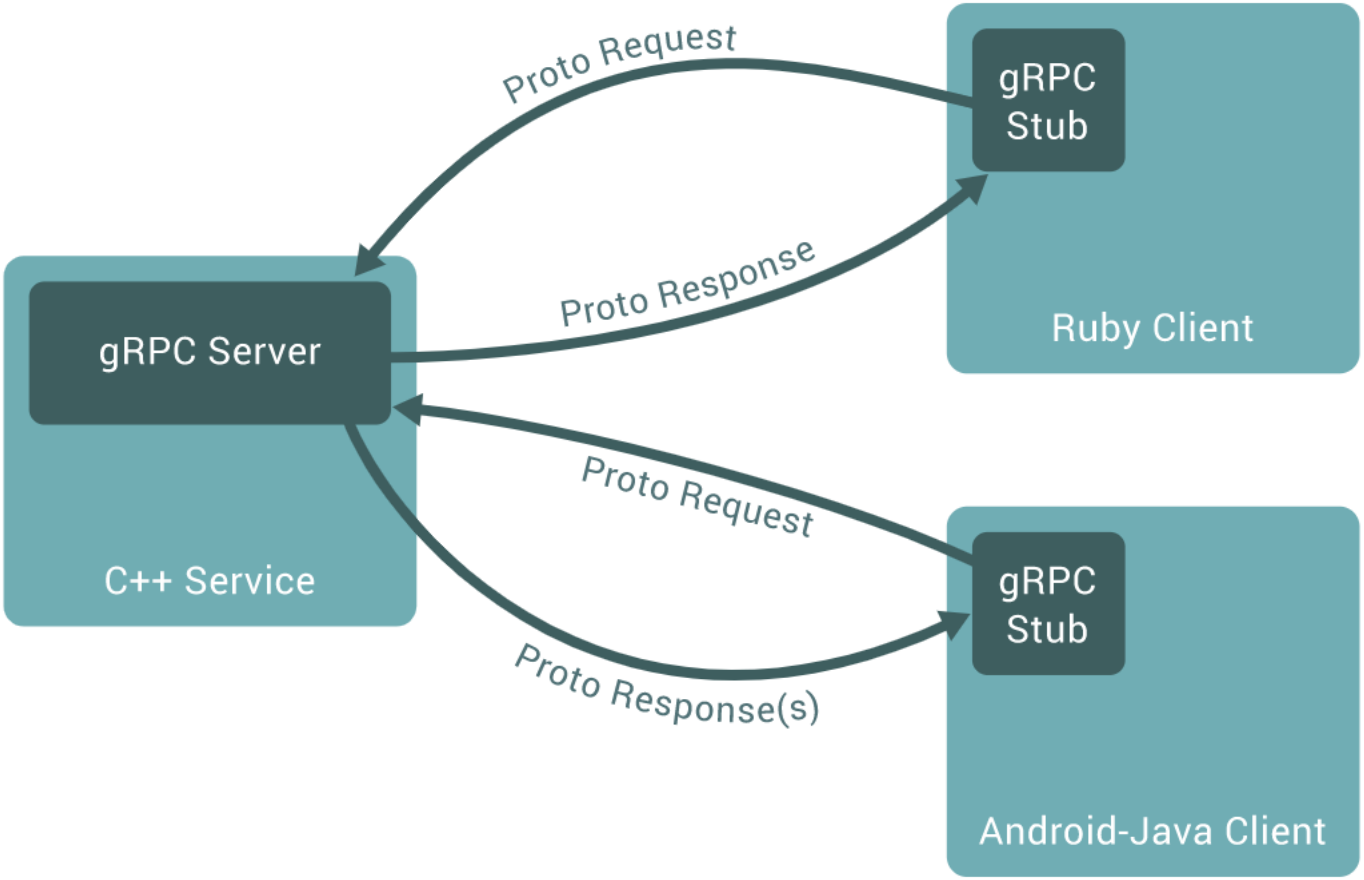
JSON

- Human readable.
- Not secure (clear text).
- Not strongly typed => errors.
- Requires manual (de)serialisation.

HTTP/2

- Single TCP connection per client-server.
- Requests Multiplexing.
- Header compression.
- Bidirectional streaming.
- Server push.

Remote Procedure Call



Protocole Buffer

Protocole Buffers

```
service CarRentalService {
  rpc rentCars(stream Car) returns (stream Invoice) {}
}

message Car {
  string plateNumber = 1;
  string brand = 2;
  uint32 price = 3;
}

message Invoice {
  uint32 price = 1;
}
```

Service method

- Single request => single response.
- Single request => response as a stream (sequence of messages).
- Stream => single response.
- Bidirectional streaming RPC (both sides send a sequence of messages).

Synchronous or asynchronous call

- Synchronous => calls that block until a response.
- Asynchronous => Never block.

Languages

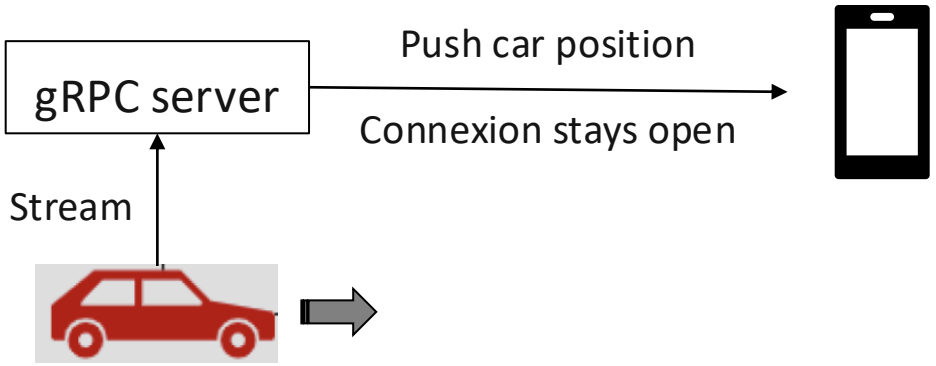
- C# / .NET
- C++
- Dart
- Go
- Java
- Kotlin
- Node
- Objective-C
- PHP
- Python
- Ruby

Platforms

- Android
- Web
- Coming soon:
  - IoS
  - Flutter: mobile, Web, desktop

Use cases

- Temporal unifies workflow activities written in several programming languages:
  - one developer code activity in PHP
  - another one code in Go.
  - Each activity sends its mission-critical data to a gRPC client that forwards it to a gRPC server.
- Lyft uses gRPC to transmit the location of a vehicle in a continuous stream of gRPC messages



PRO / CONS

Pro	Cons
Machine Readale	Not human readable
Handles (de)serialization	
Code generator for many languages	Lot a boilerplate code generator at compilation time
Based on a schema to code	
Binary data representation – Less expansive for big payload - Fast	Not browser consumable natively. Java script client is possible.
Strongly typed	Client and server need to access the generated code
Structures can be extended	

Thrift an alternative to gPRC

- Apache fundation
- Multiple serialization formats
- more flexibility in terms of message types and service definition
- wide range of transport protocols such as TCP, HTTP, …
- Performance: gRPC is generally considered to be faster due to its use of HTTP/2 as a transport layer

Thrift an alternative to gPRC

- JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments
- SSE (Server-Sent Events) est un protocole de communication unidirectionnel entre les clients et les serveurs basé sur le protocole HTTP. Il permet aux serveurs de pousser des données aux clients en temps réel en utilisant des flux d'événements
- The Streamable HTTP transport allows MCP servers to operate as independent processes that can handle multiple client connections using HTTP POST and GET requests, with optional Server-Sent Events (SSE) streaming for multiple server messages. It replaces the SSE transport.

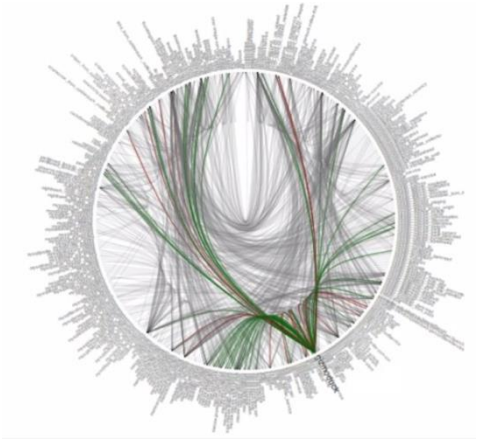
Implementation web service

https://github.com/charroux/servicemesh

Summary

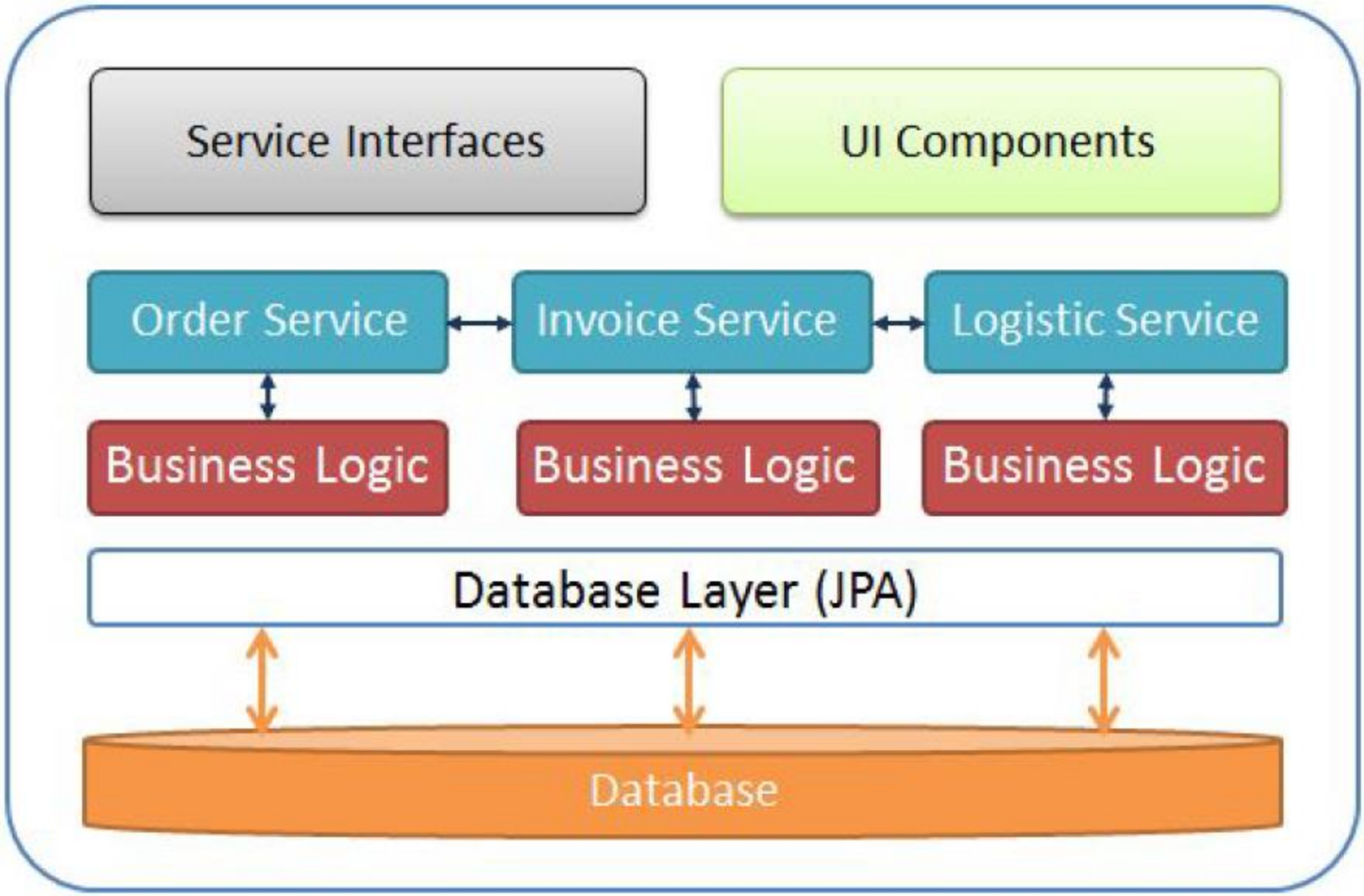
- The big picture of the web services
- Web services
- Microservices
- Applications cloud native

Microservices



The monolithic architecture

Monolithic Enterprise Application



Drawbacks

- A small change => rebuilt and deployed the monolith.
- Refactoring => hard to keep a good modular structure.
- Scaling => scaling of the entire application.

Microservice

Definition

Microservices are small building blocks, highly decoupled and focused on doing a small task, facilitating a modular approach to system-building.

Characteristics of a microservice

- Relatively small
- Managed by a small team (DevOps).
- Dev: can use his own technology
- Ops :
  - can be deployed independently
  - can be scaled independently
  - can be isolated in case of failure

Characteristics of a microservice

- Relatively small => Domain Driver Design
- Managed by a small team (DevOps)
- Dev: can use his own technology
- Ops :
  - can be deployed independently => CI/CD / Docker-Kubernetes
  - can be scaled independently => Kubernetes
  - can be isolated in case of failure => Service mesh

Drawback

- Complex inter-service communication mechanism:
  - Discovery is complex
  - Synchronism leads to bad performance
  - Distributed transactions are never ACID
- How implementing use cases that span multiple services ?

- Testing is more difficult

Solutions?

- Complex inter-service communication mechanism:
  - Discovery is complex => DNS / pub/sub
  - Synchronism leads to bad performance. => Reactive systems
  - Distributed transactions are never ACID. => Saga pattern
- How implementing use cases that span multiple services ?

  => API management, GraphQL
- Testing is more difficult => CI

Are they able to support an information system?

- Information Systems are:
  - heterogenous
  - decentralized
  - distributed
- Need for integration

Are they able to support an information system?

- Information Systems are:
  - heterogenous
  - decentralized
  - distributed
- Need for integration => Enterprise Integration Patterns, message broker

- Microservice architecture is just the most cloud compatible and fastest option
- Microservice architecture ≠ Service Oriented Architecture
- Domain Driven Design is thinking in terms of code
- API approach can bridge the gap between the users and the microservices
- Event Driven Architectures brings decoupling and asynchronous exchanges

Are microservices cloud native?

Virtualize microservices
=> Containers (CaaS)

Cloud compatible app by design => 15 factor app

Fast delivery of new releases => Continuous Integration / Continuous Delivery

Yes they can

The gateway to the cloud => API management / API gateway

Deploy from scratch => IaC

Monitoring and Control pane => Service mesh

Manage containers in a cluster => Kubernetes

Summary

- The big picture of the web services
- Web services
- Microservices
- Applications cloud native

Application cloud native

Cloud native ?

The term Cloud Native is used to:

- Describe application architectures adapted to the Cloud deployment model
- Describe how applications are developed and deployed

  - The applications are distributed in container format
  - Applications are developed on a micro-services model to exploit the elasticity properties of the infrastructure
  - Applications are developed using agile methods and deployed on the infrastructure using a DevOps process
  - The DevOps process is based on CI/CD pipeline technologies
    Applications are designed for failure: design for failure

https://github.com/cncf/foundation/blob/main/charter.md

Virtualize microservices
=> Containers (CaaS)

Cloud compatible app by design => 15 factor app

Fast delivery of new releases => Continuous Integration / Continuous Delivery

Life cycle

The gateway to the cloud => API management / API gateway

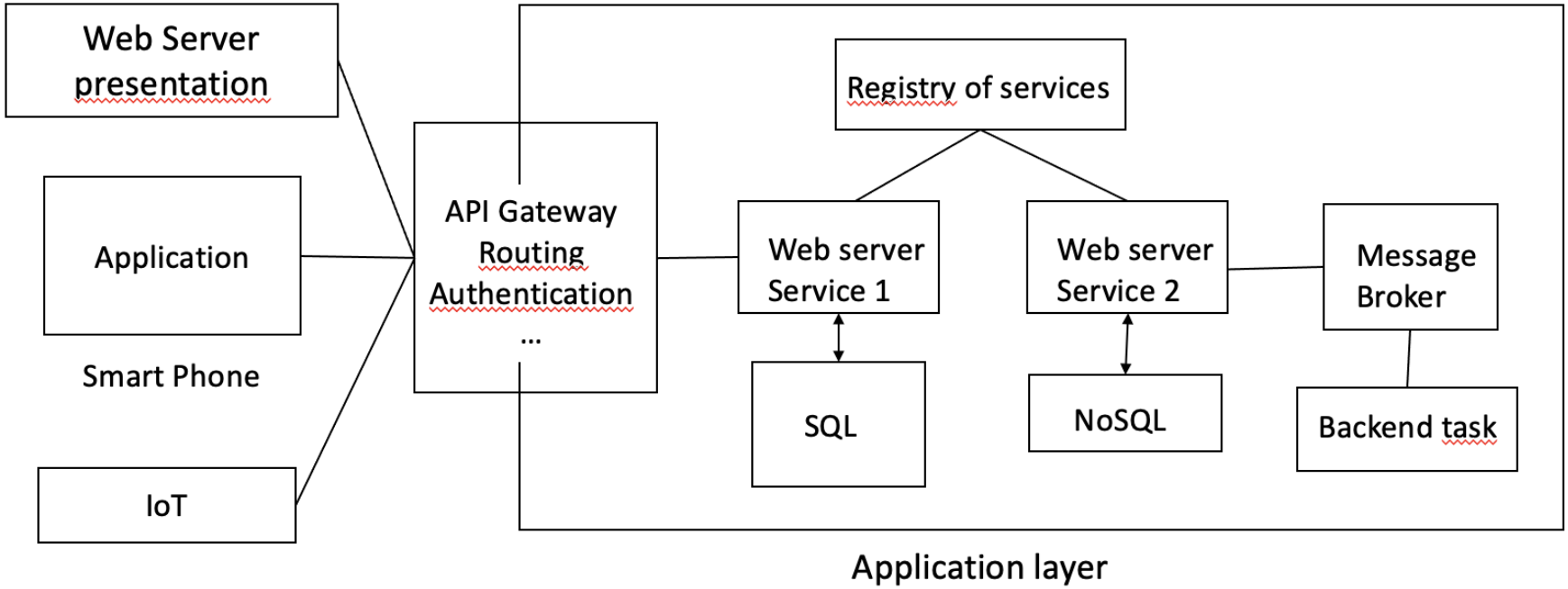
Deploy from scratch => IaC

Monitoring and Control pane => Service mesh

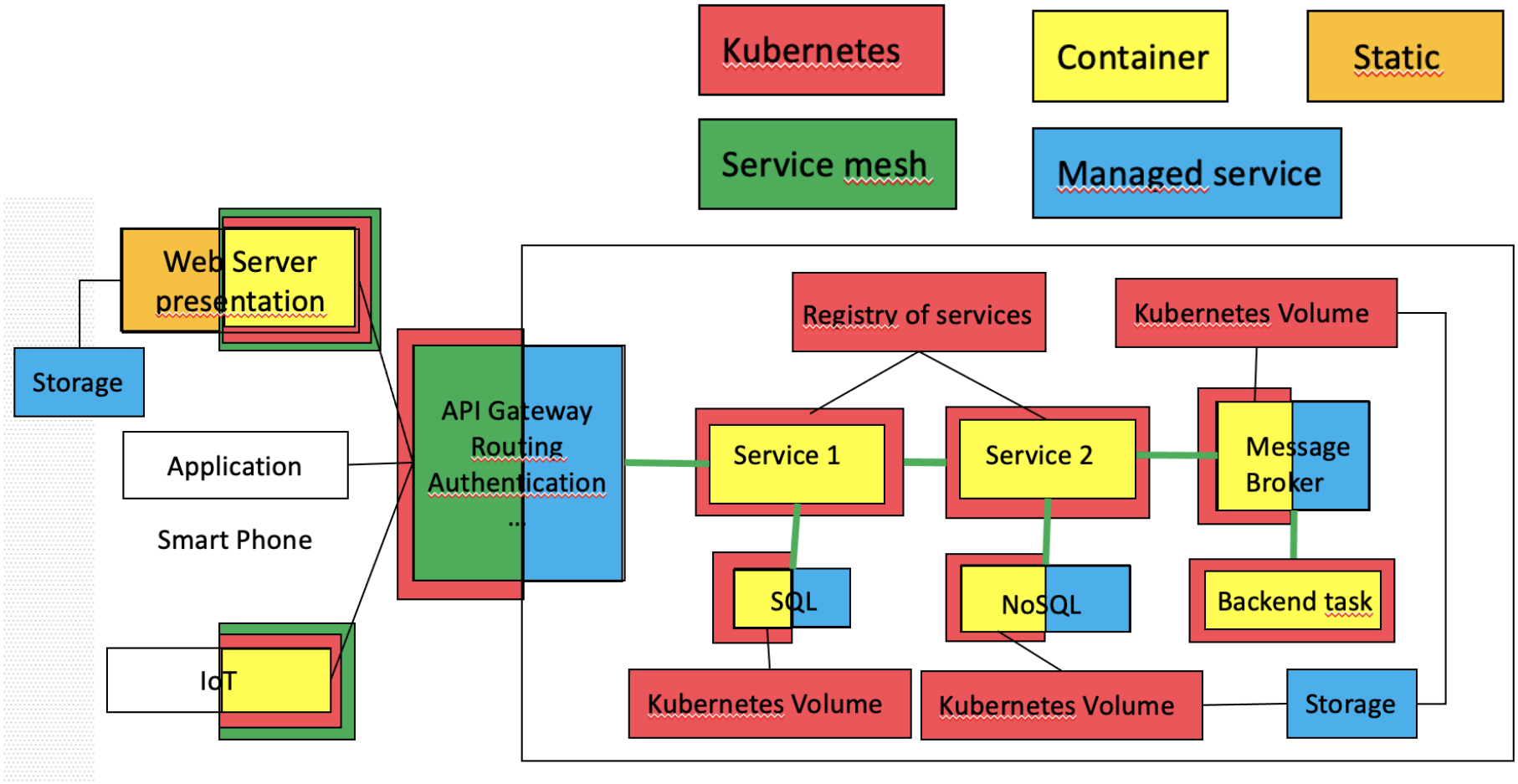
Manage containers in a cluster => Kubernetes

Container as a Service

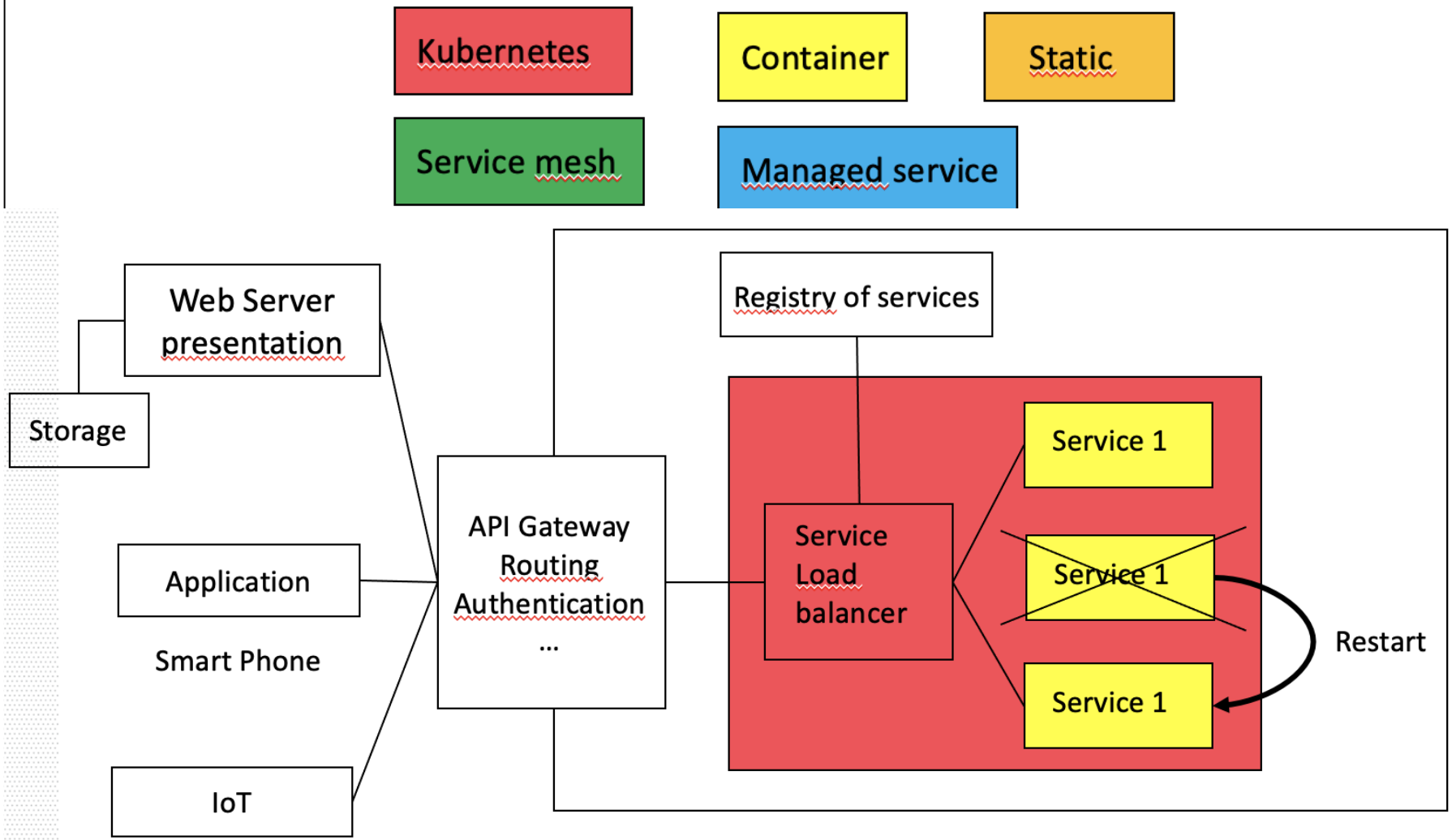
Microservices



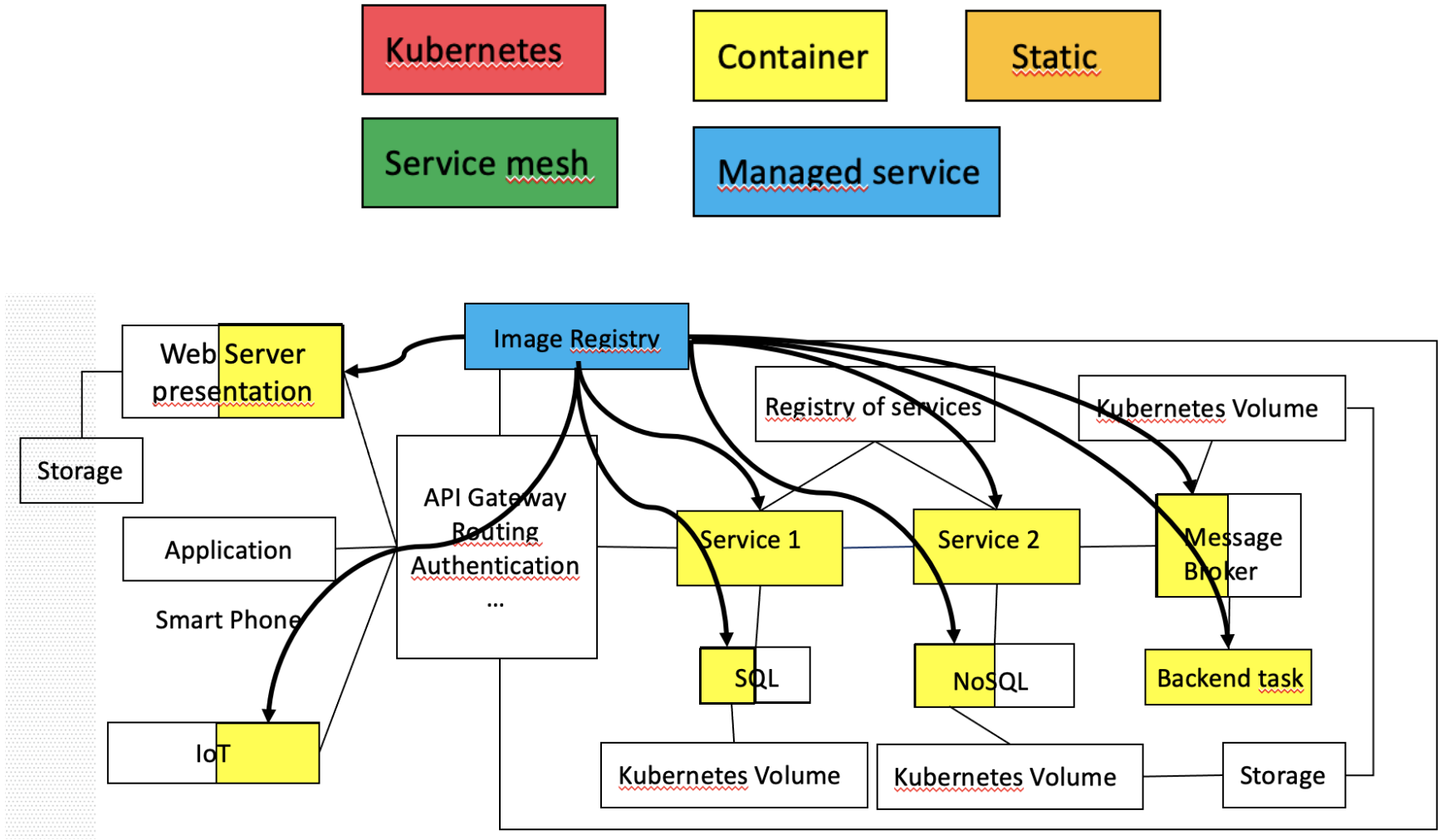
CaaS


Kubernetes
Container
Static
Service mesh
Managed service
Web Server
presentation
Storage
Application
Smart Phone
IoT
API Gateway
Routing
Authentication
...
Registry of services
Kubernetes Volume
Service 1
Service 2
Message
Broker
SQL
NoSQL
Backend task
Kubernetes Volume
Kubernetes Volume
Storage

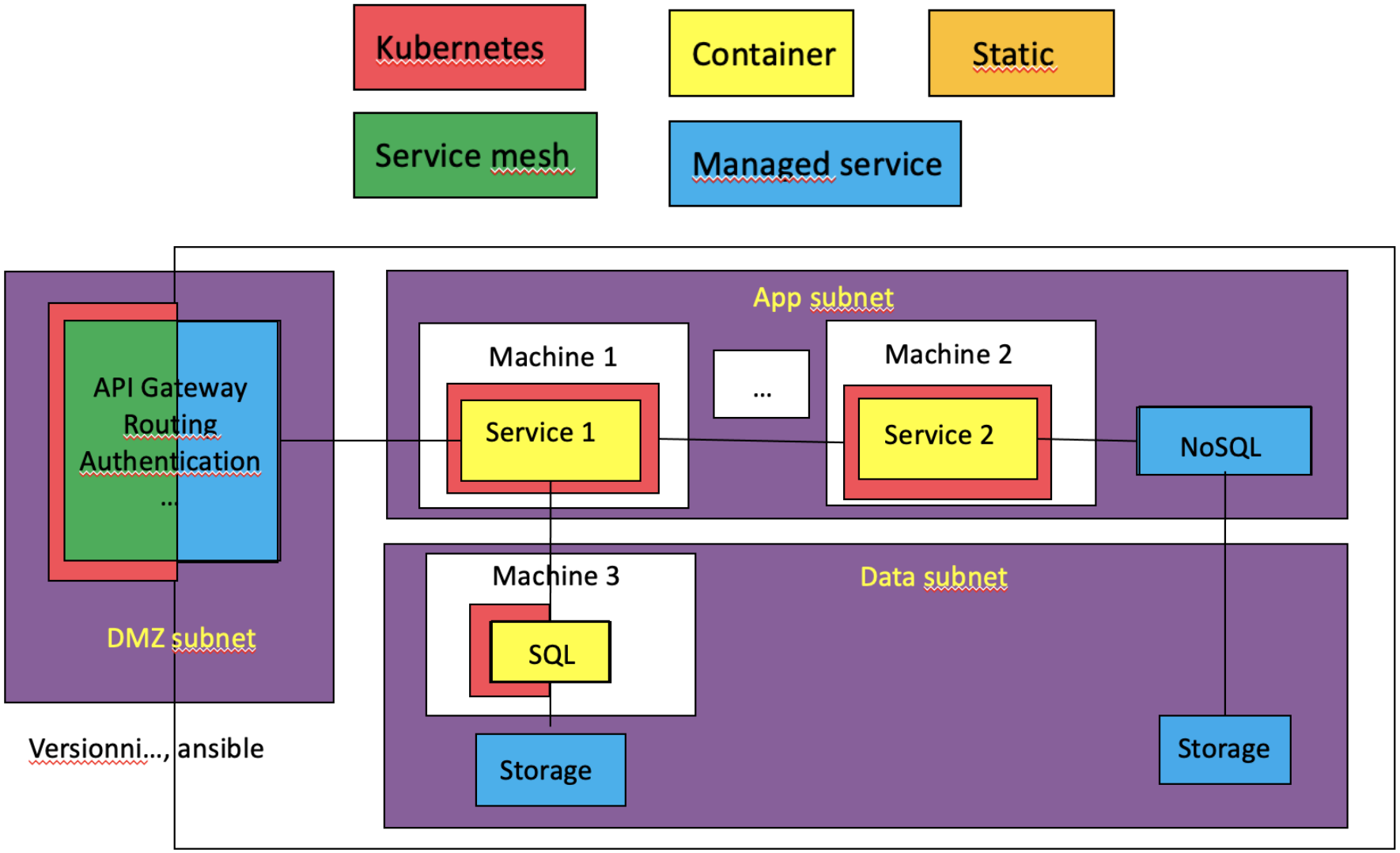
Scalling and fault tolerance



Images management

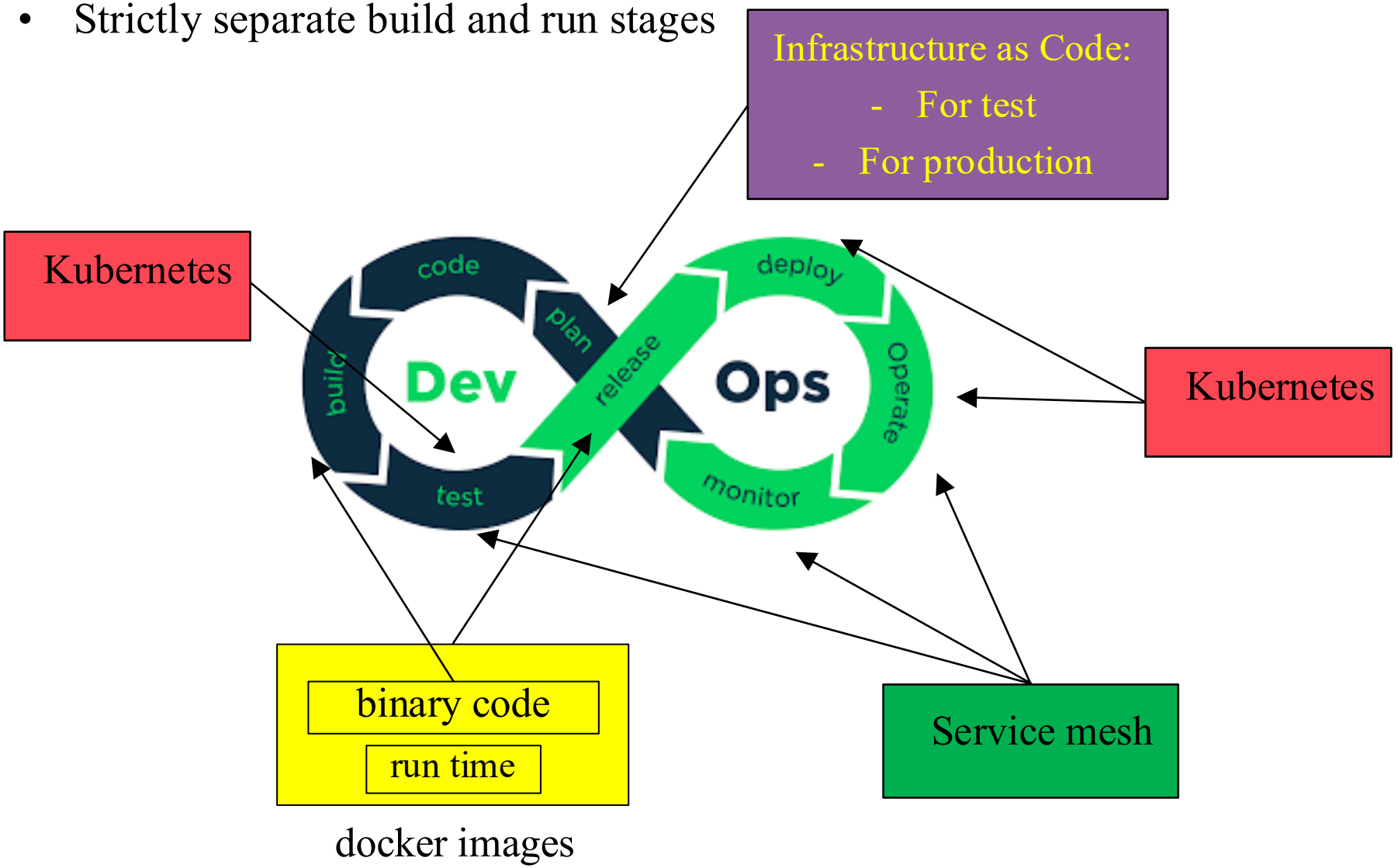


Infrastructure as Code



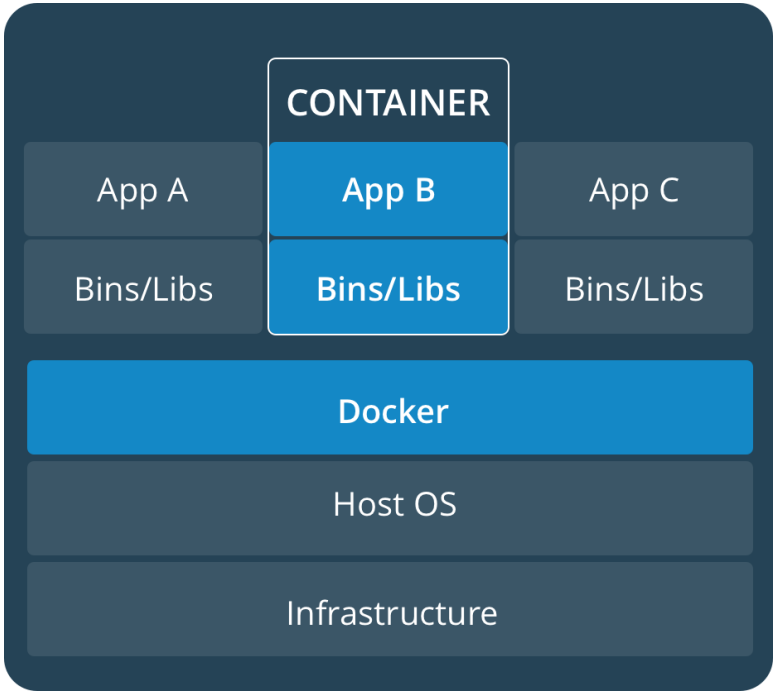
Dev Ops

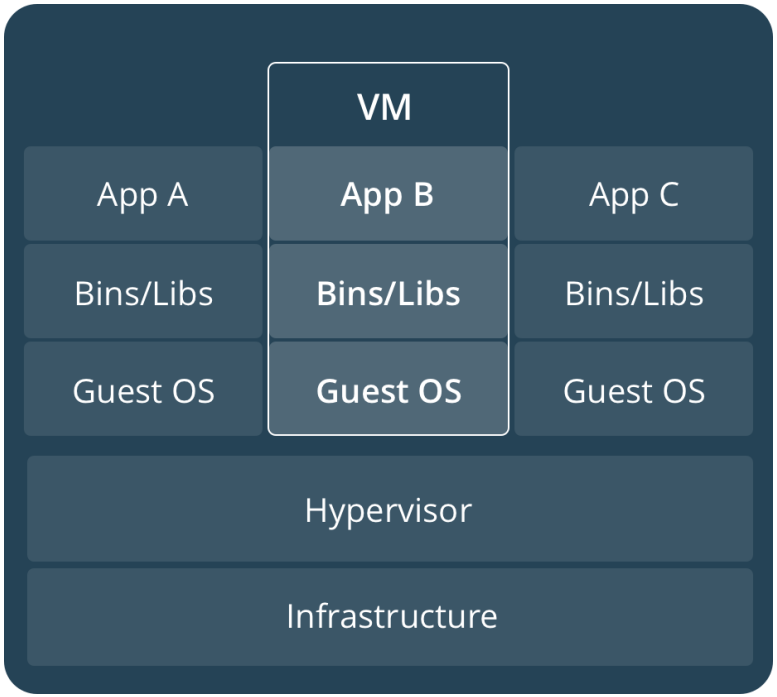
- Strictly separate build and run stages



What is Kubernetes?

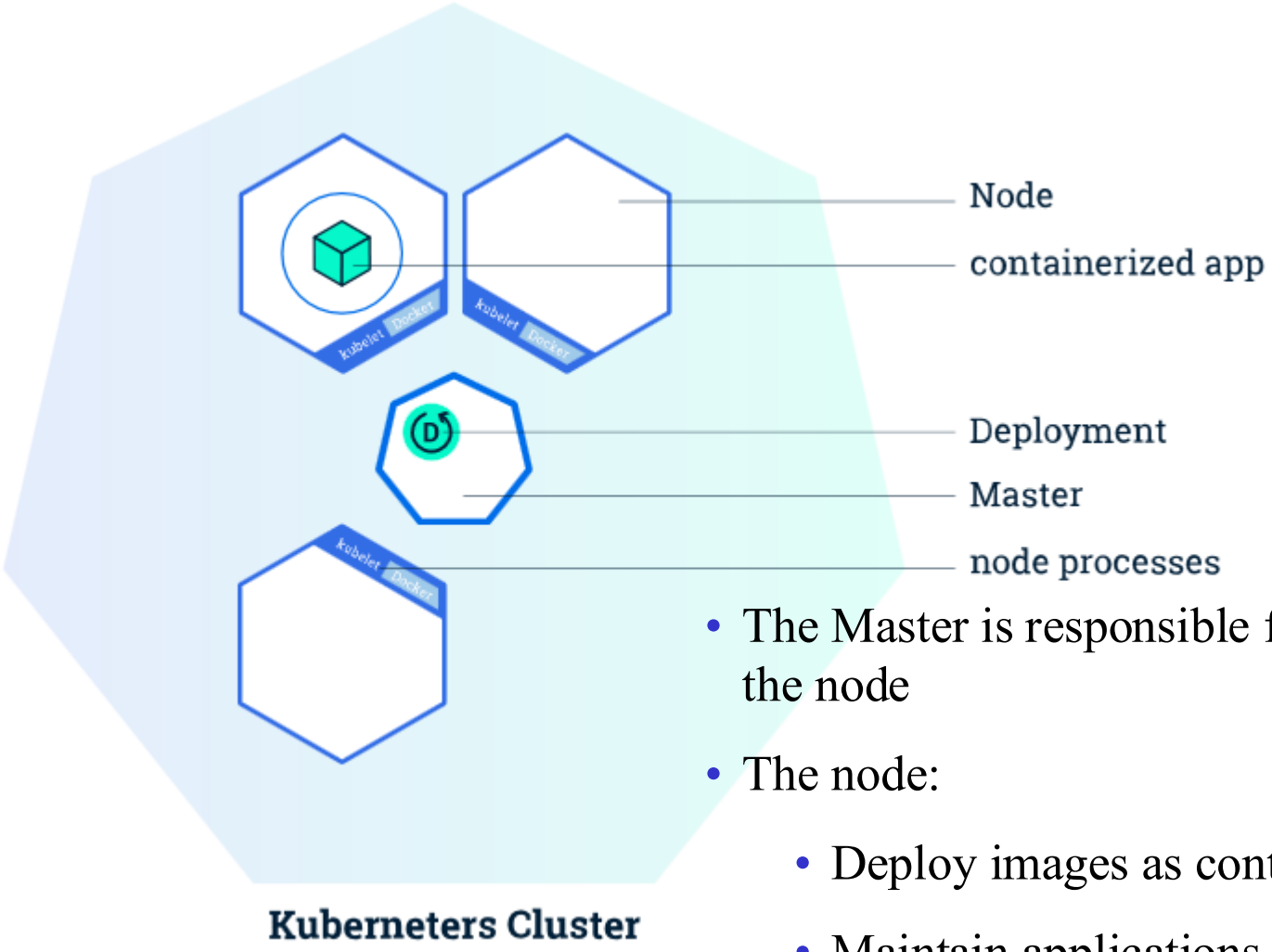
Docker containers vs Virtual Machines





- Multiple containers share the OS kernel.
- Containers take less space than VMs.
- Containers start almost instantly.

The Kubernetes cluster

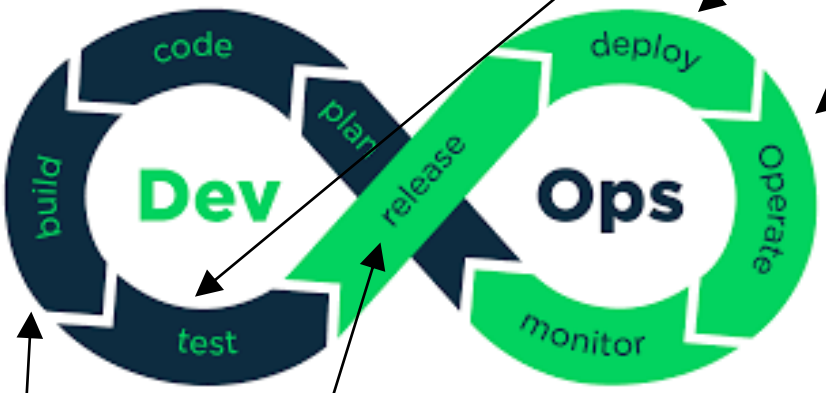


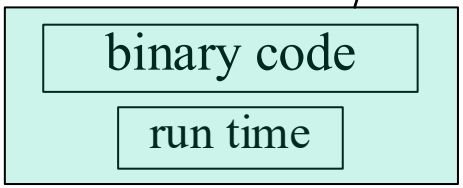
- The Master is responsible for selecting the node
- The node:
  - Deploy images as containers
  - Maintain applications state
  - Scaling applications

Kubernetes and DevOps

- 24/7 availability
- Continuous Integration / Continuous Delivery

- **Kubernetes**:
  - Deploy docker images
  - Restart failed containers
  - Scaling applications





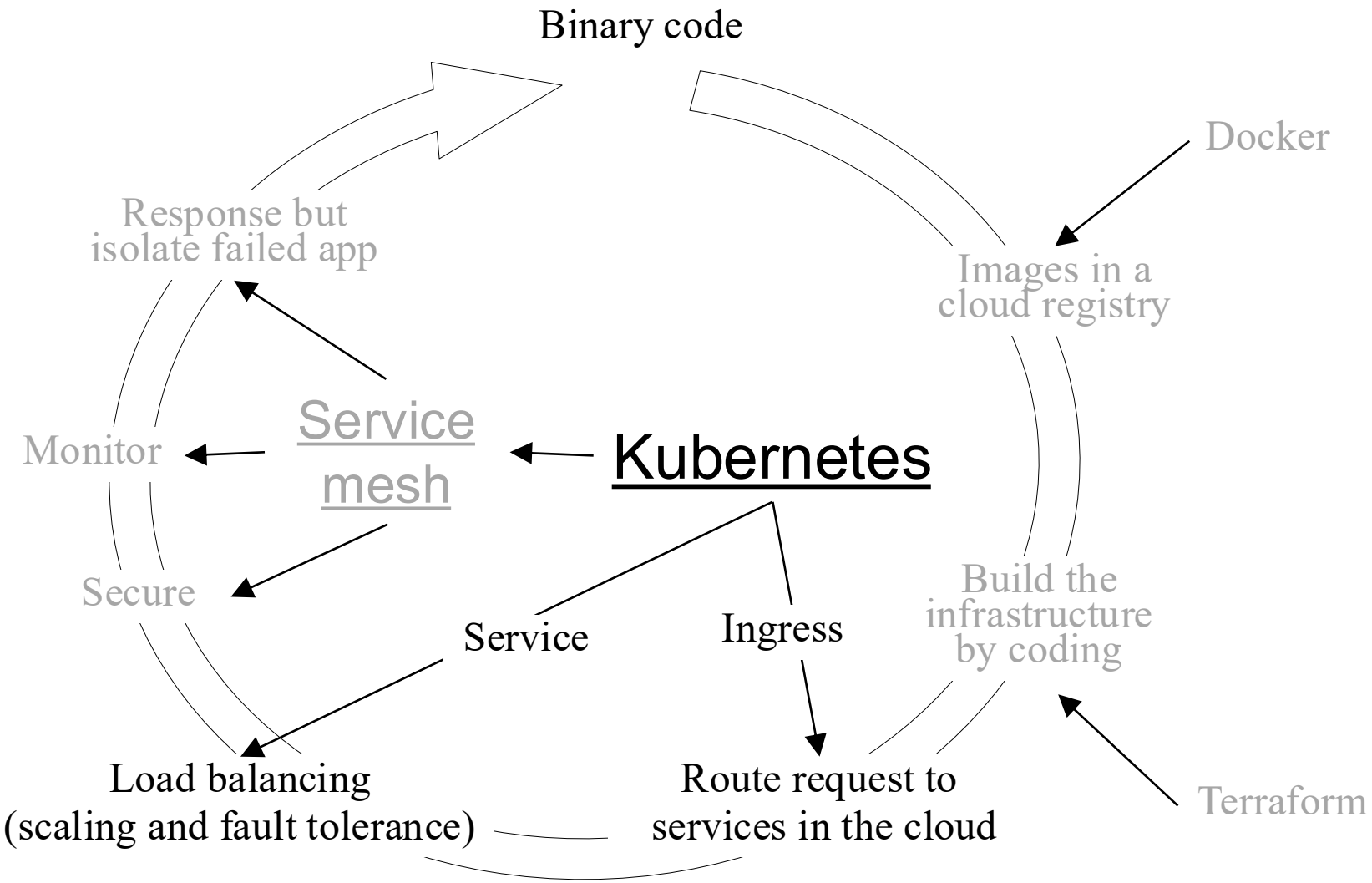
docker images

Service mesh

Kubernetes

- Kubernetes is open source
- Versions:
  - k8s
  - k3s: ligthweight solution for edge computing, IoT, Raspberry Pi…
- Turnkey solutions (IaaS providers with a few command):
  - Google cloud
  - AWS
  - …

Cloud native applications



Best practise for databases and containers

- Keep the current mode of operation for relational databases (HA, replication, backup, etc.)
- Integrate stateless parts of applications on your CaaS while maintaining current database environments
- Create containers with the database engine when relevant (low performance, low volume of data, non-production environment, micro-services development, etc.) and place the data on persistent volumes
- Segment the BDDs during the development in micro-service of the associated functionality
- Upgrade to Cloud Native database engines (sharding, etc.)
- Depending on the level of adoption of IaaS or public cloud technologies, evaluate DBaas solutions

Helm

- An ideal tool for managing Kubernetes applications should:
  - Support YAML file templating:
    - Possible to customize the files by Variables => useful for Update, RollBack and Scaling
  - Support dependency management:
    - Allow the tool to install the target application, and all these dependencies simply
    - Ease of publishing applications and their dependencies
  - Offer a repeatable and consistent deployment model

Kubernetes in practice

https://github.com/charroux/servicemesh

Service Mesh

Why service mesh?

The more Kubernetes applications grow,

the more service to service communication is difficult.

What is service mesh?

Service mesh

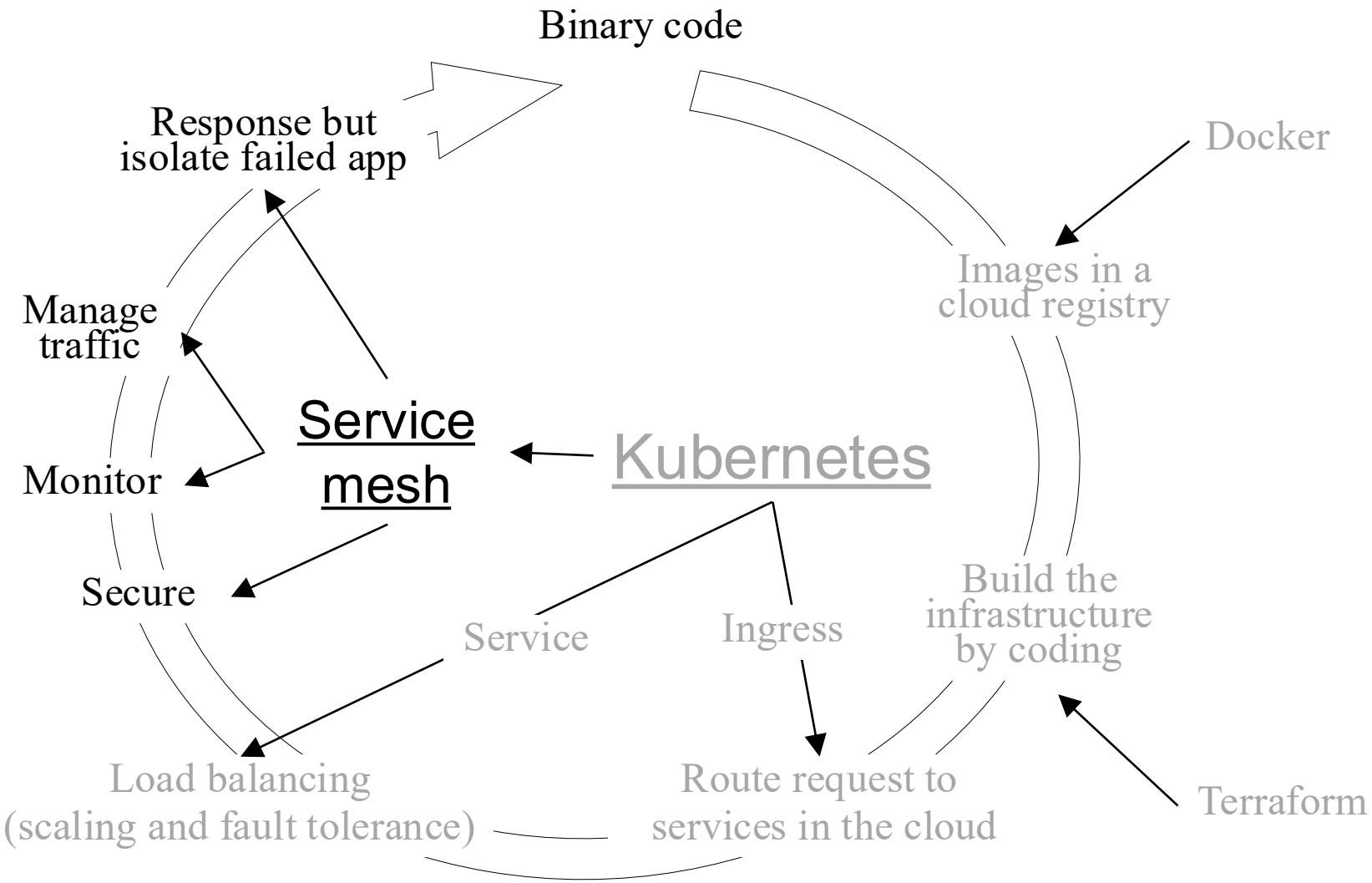
- Service mesh manages network traffic between services at the platform layer instead of the application layer.

Open service mesh

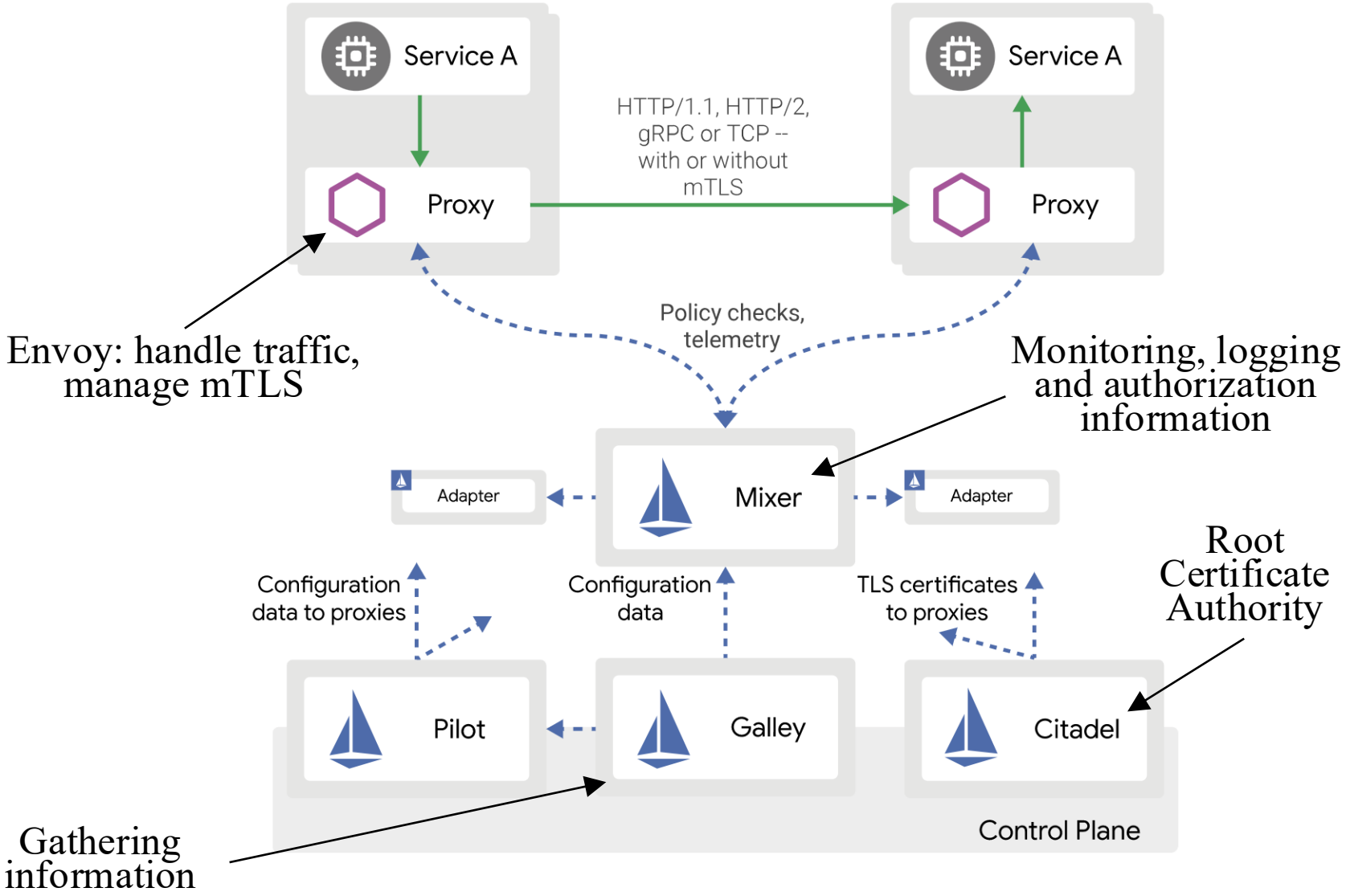
AWS App Mesh

Anthos Service Mesh

Cloud native applications



The example of Istio



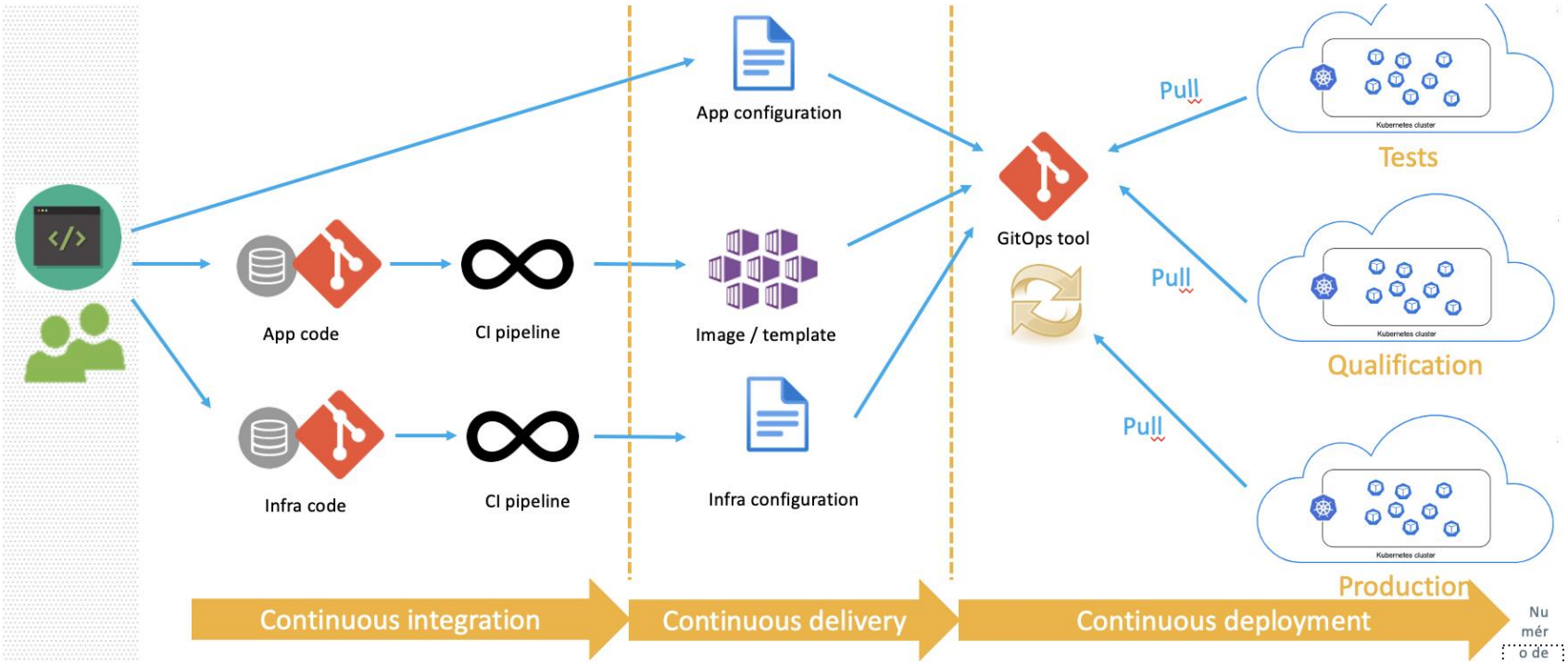
Service mesh in practice

https://github.com/charroux/servicemesh

Continuous Integration with Kubernetes

https://github.com/charroux/servicemesh#continuous-integration-with-github-actions

Continuous Delivery / Deployment with GitOps



The twelve-factor app

The twelve-factor app

- Use declarative formats for setup automation, to minimize time and cost for new developers joining the project => Terraform, Kubernetes and Servicemesh are declaratives.
- Have a clean contract with the underlying operating system, offering maximum portability between execution environments => Docker images
- Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration => Terraform, Kubernetes and Service mesh are supported by all clouds
- Minimize divergence between development and production, enabling continuous deployment for maximum agility => CI/CD pipeline, containers vs managed services
- And can scale up without significant changes to tooling, architecture, or development practices => Kubernetes

The twelve-factor app

- One codebase tracked in revision control, many deploys: Git
- Explicitly declare and isolate dependencies: package manager
- Store config in the environment (environment variables) not in the code, this includes:
  - Resource handles to the database, Memcached, …
  - Credentials to external services such as Amazon S3 or Twitter
  - Per-deploy values such as the canonical hostname for the deploy

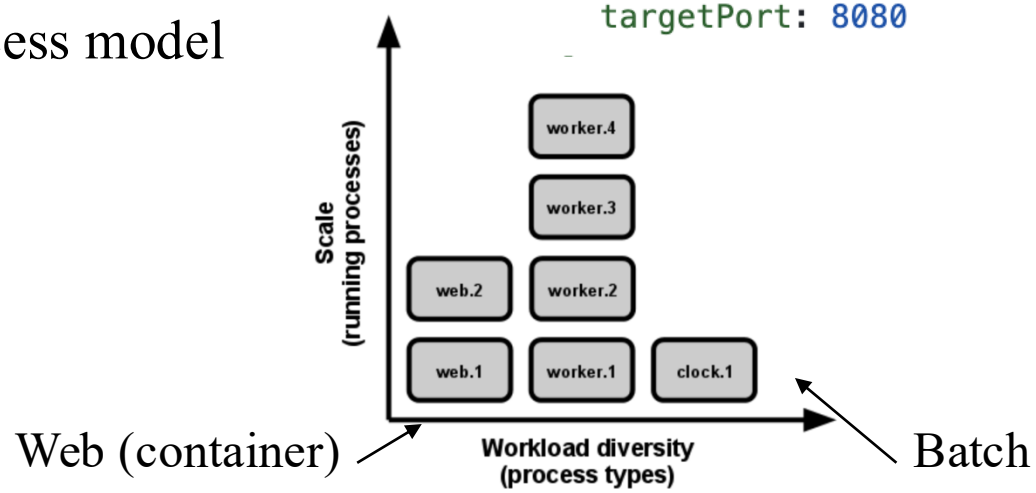
  => Kubernetes
- Treat backing services as attached resources
  - Swap out a local MySQL database or services with one managed by a third party without coding => Kubernetes volumes
- Execute the app as one or more stateless processes
  - processes are stateless and share-nothing. Any data that needs to persist must be stored in a stateful backing service, typically a database ????

The twelve-factor app

- Export services via port binding

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    run: carstat
  name: carstat
spec:
  ports:
    - name: grpc-web
      port: 8080
      targetPort: 8080
```

- Scale out via the process model



The twelve-factor app

- Maximize robustness with fast startup and graceful shutdown
  - Docker containers start instantly because they don't contain a full OS
  - Kubernetes restrats failed containers
- Keep development, staging, and production as similar as possible
  - Kubernetes can be used as virtual (Dev) or real machines (production)
  - Kubernetes claim can search for volumes at starting (local, NTFS, cloud…)
  - Kubernetes can change containerized database (for dev) to cloud managed database (for prod)
- Treat logs as event streams (Sequence of business events for indexing and analysis):
  - During local development, the developer view the logs in stdout
  - In production, each process' log is routed to a long-term archival.
- Run admin/management tasks as one-off processes (running database migrations…)
  - Admin code must ship with application code to avoid synchronization issues.
  - The same dependency isolation techniques should be used on all process types.

3 additional factors

- Telemetry and real-time app monitoring
  - Monitoring, tracing, logging
  - Service mesh for example
- Authentication and authorization
  - Service mesh:
    - Authentication throw the gateway
    - mTLS
    - …
  - Dev**Sec**Ops
  - Secret management
- API first