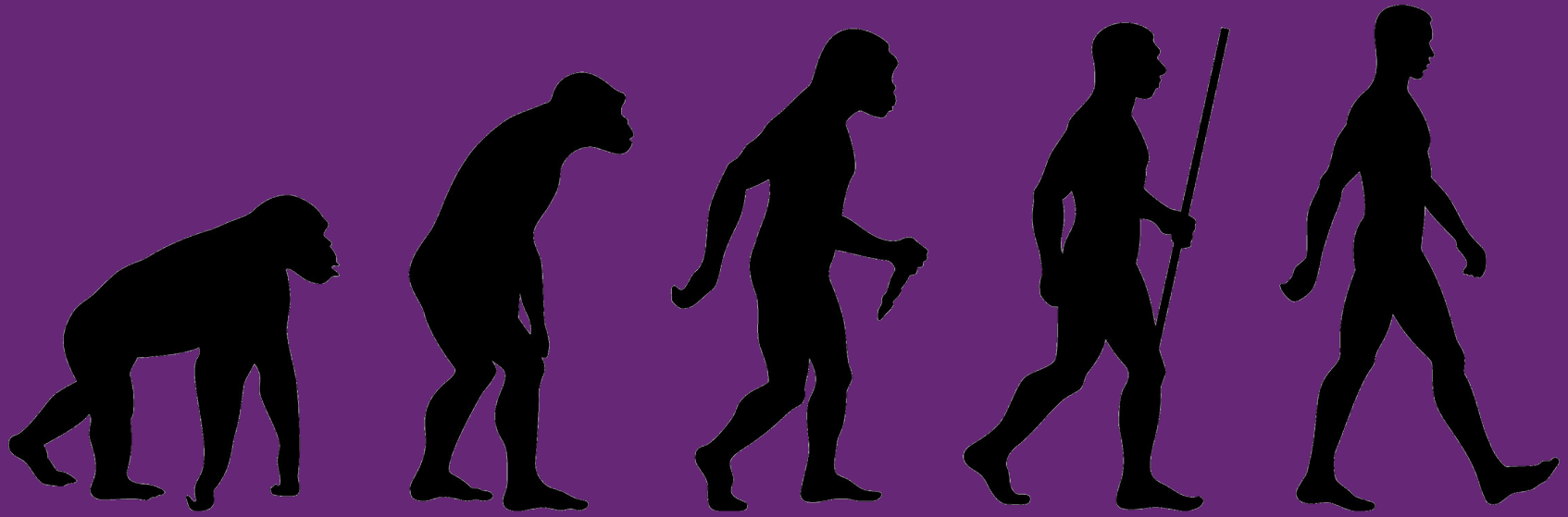


Microservices as an Evolutionary Architecture: Lessons learned

Claudio Eduardo de Oliveira & Luram Archanjo

Who am I?

- Software Engineer @Sensedia
- MBA in Java projects
- Java and Microservice enthusiast

Who am I?

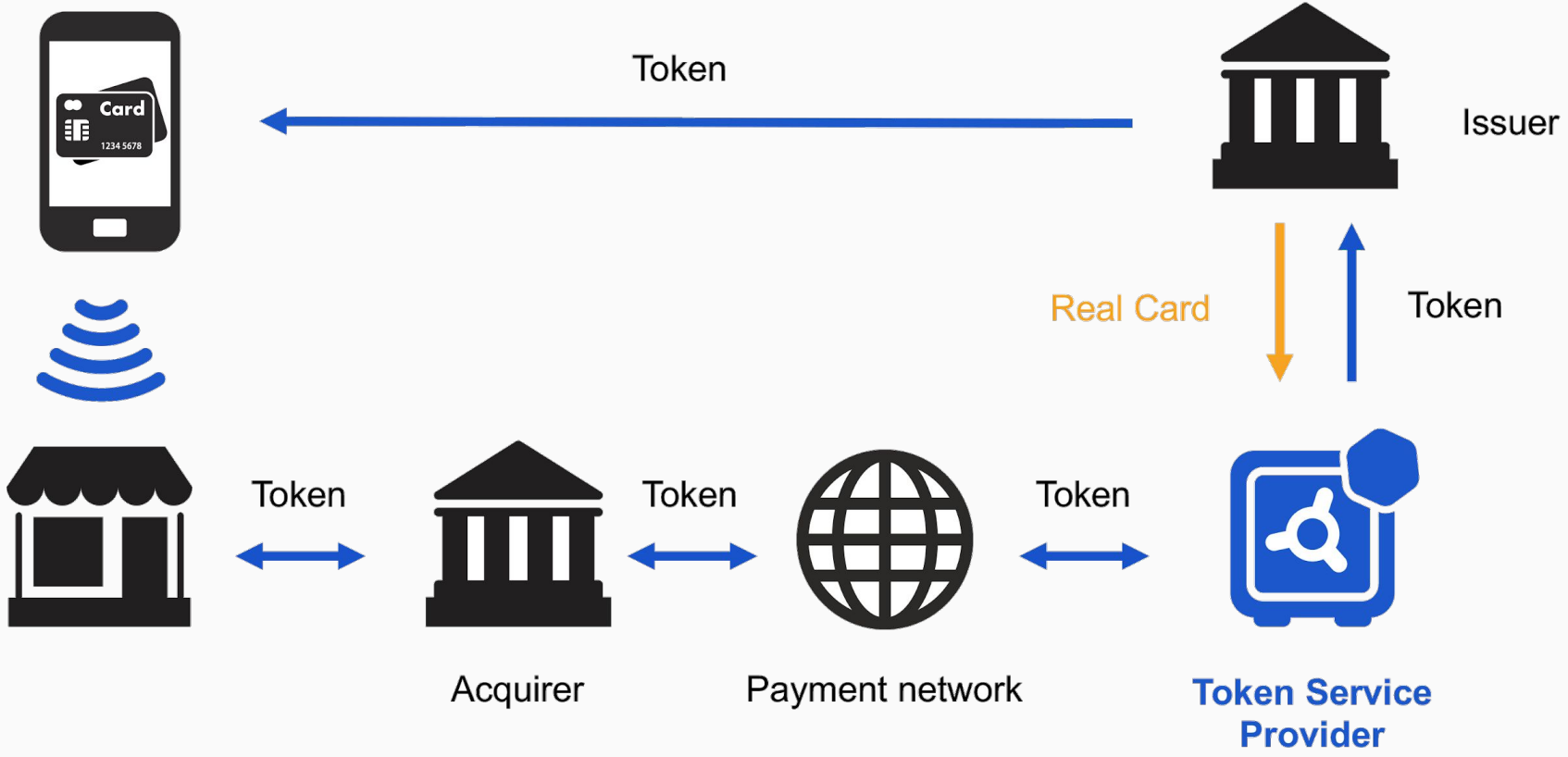
- Lead Solutions Architect @Sensedia
- Book Author
- Spring, Java and Cloud Native enthusiast

Agenda

- Use Case
- Microservices & Evolutionary Architecture
- Domain Driven Design (DDD)
- Communication Patterns
- Concurrency Patterns
- Observability
- Questions

Use case

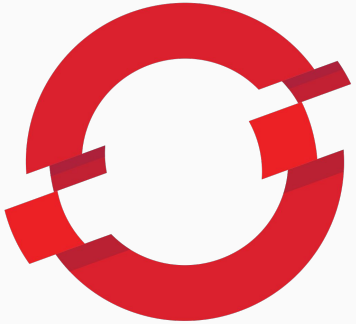
Use case


Card
1234 5678
Token
Issuer
Real Card
Token
Token
Token
Token
Acquirer
Payment network
Token Service Provider

THE CHALLENGE

Response time

Throughput



OPENSHIFT

sensedia

API PLATFORM

In the beginning

As is

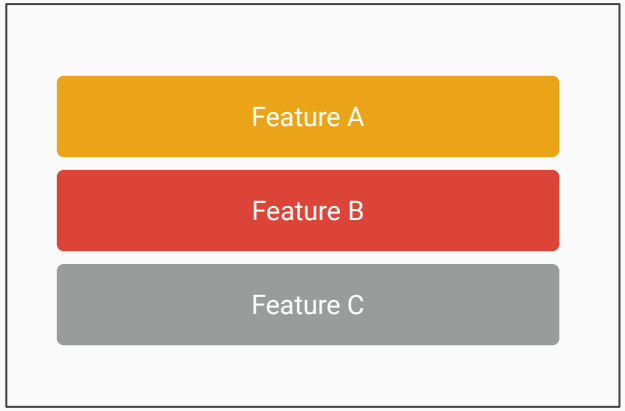
- Single communication pattern
- Java & Spring everywhere
- Synchronous process everywhere
- Traffic east-west in the API Gateway
- 50 - 60% of our commits during the sprint were in the same repository
- Service with different workload requirements (Crypto and Business)

We were in the wrong way to adopt microservices architecture!

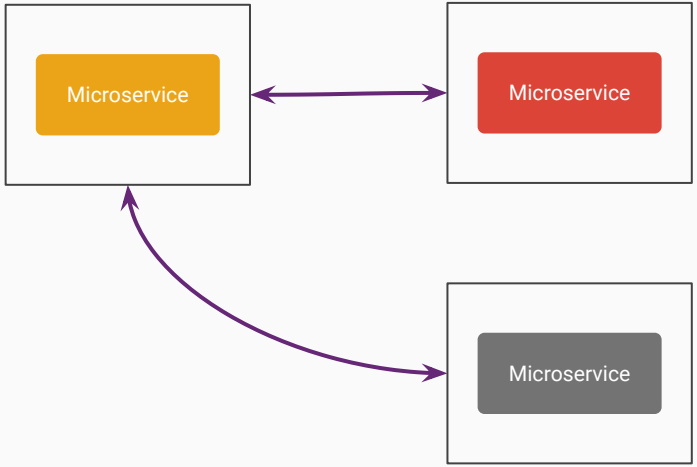
Let's recap what is the microservices architecture!

Moving to Microservices

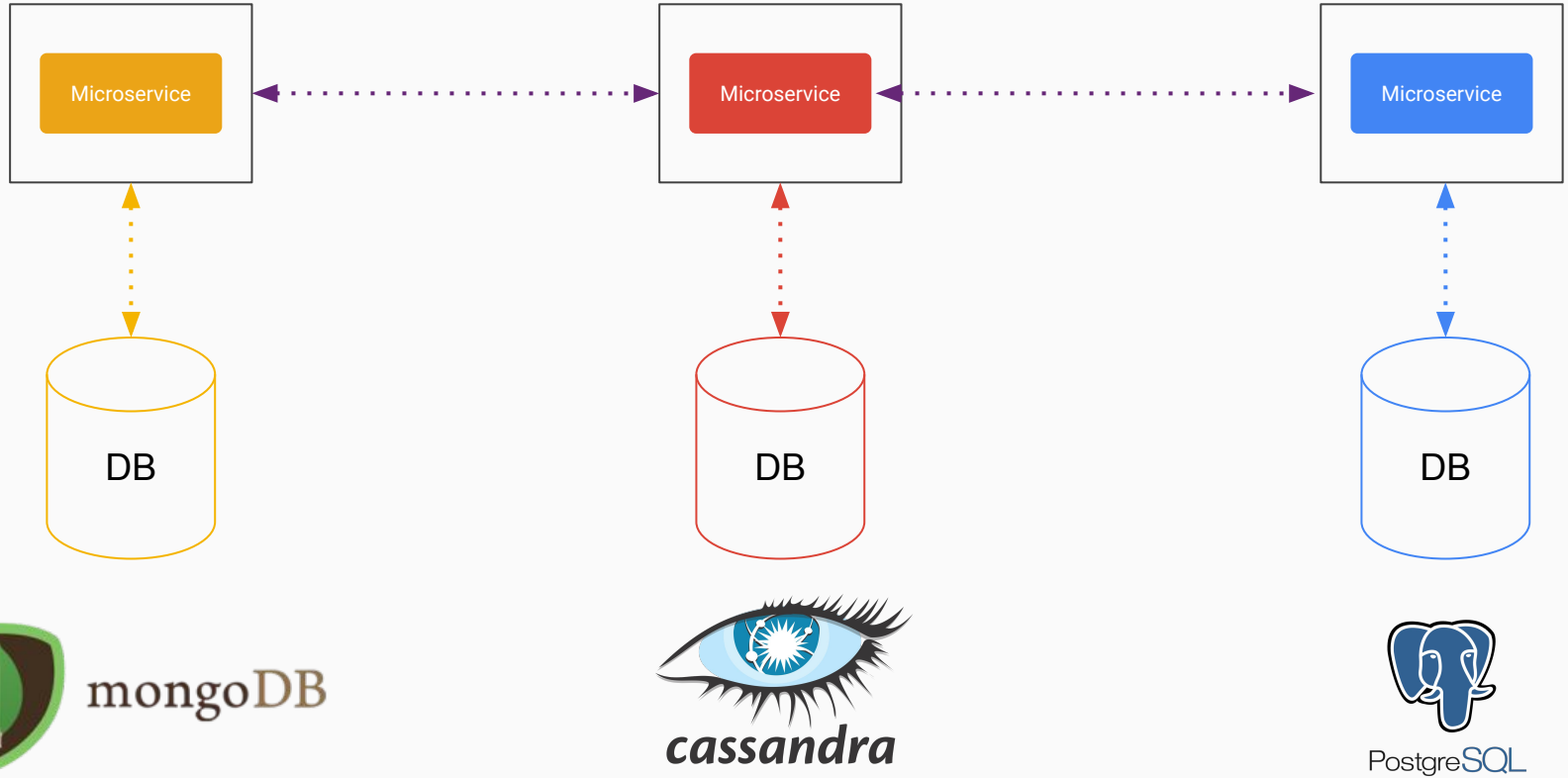
Monolith



Microservices



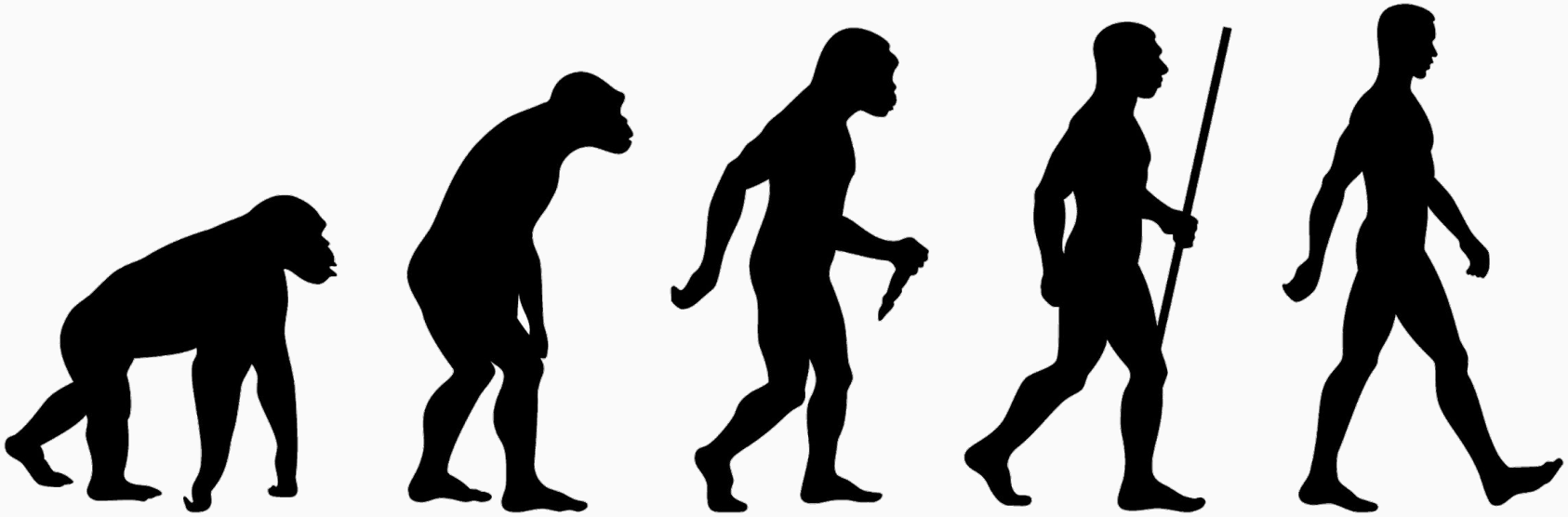
Technological Heterogeneity



Evolutionary Architecture

Evolutionary Architecture

An evolutionary architecture supports guided, incremental change as a first principle across multiple dimensions.



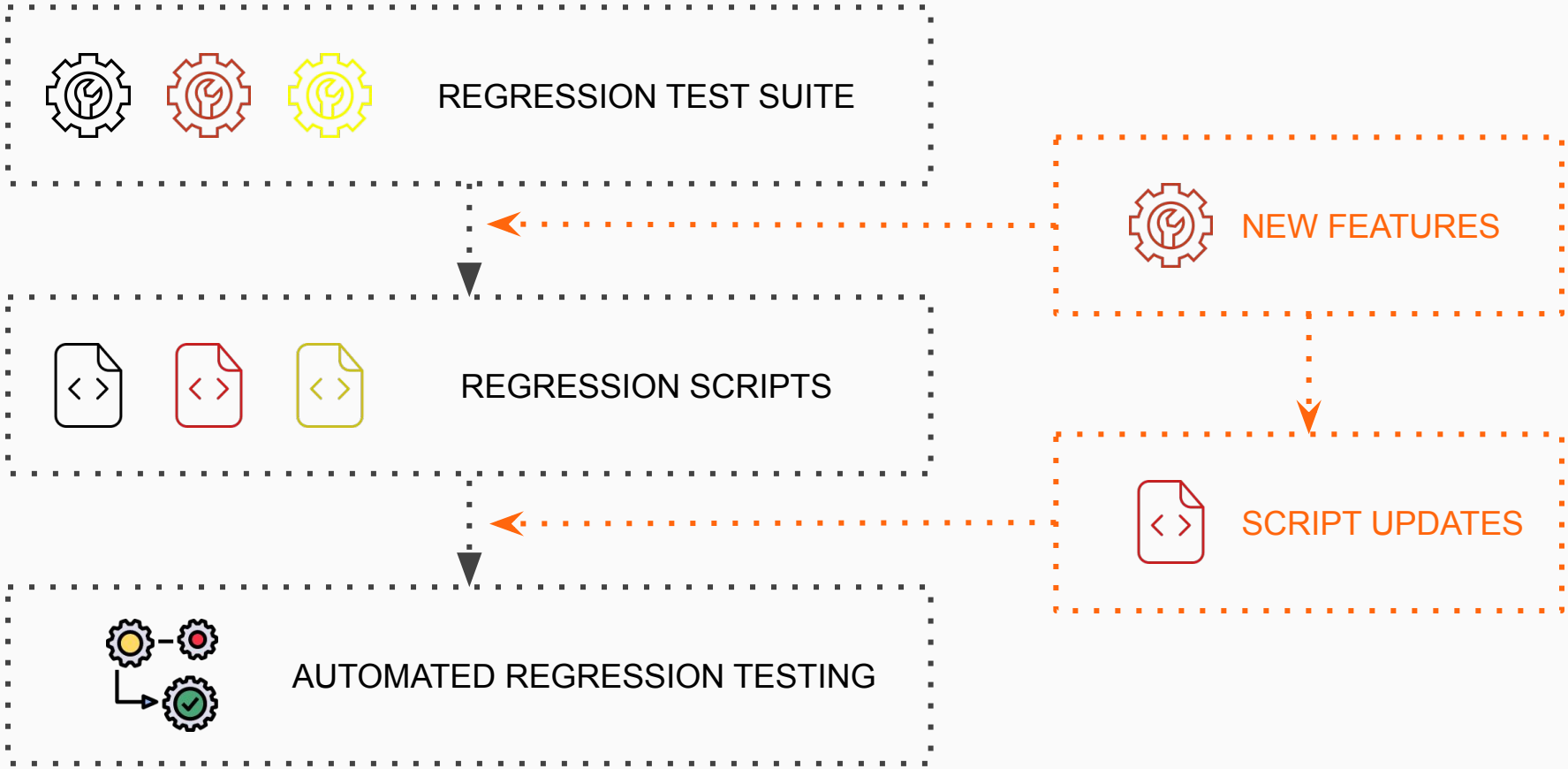
Source: https://www.thoughtworks.com/books/building-evolutionary-architectures

Before make a architectural decision, we need to be confident!

How?

Regression Tests

Regression Tests



We have tests coverage and now?

Domain Driven Design (DDD)

Domain Driven Design (DDD)

Domain-driven design (DDD) is an approach to developing software for complex needs by deeply connecting the implementation to an evolving model of the core **business concepts**.

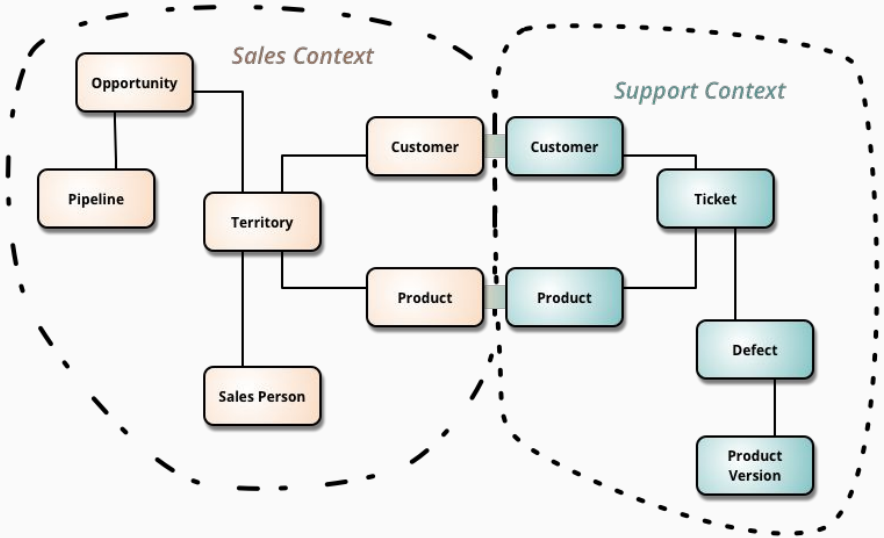
- Place the project's primary focus on the **core domain** and **domain logic**
- Base **complex designs** on a model
- Initiate a creative collaboration between **technical** and **domain experts** to iteratively cut ever closer to the conceptual heart of the problem.

Source: https://dddcommunity.org/learning-ddd/what_is_ddd/

We are trying to figure out the technique to divide services, what part of DDD can help us?

Domain Driven Design - Bounded Context

Bounded Context is a **central** pattern in Domain-Driven Design. It is the focus of DDD's strategic design section which is all about dealing with **large models** and **teams**. DDD deals with large models by dividing them into different Bounded Contexts and being **explicit** about their interrelationships.



Source: https://martinfowler.com/bliki/BoundedContext.html

We separated the responsibilities
and now?

Communication Patterns


REST API
LIGHTYEAR
REST API, EVERYWHERE

Advanced Message Queuing Protocol (AMQP)

Advanced Message Queuing Protocol (AMQP)

- Introducing Message-Driven Programming with Low Level Events
- Fire & Forget principle
- Durable Messages (avoid Circuit Breakers & Retries)
- Better Scale strategies & Application Decoupling
- Fits well for 3rd partners integrations

Advanced Message Queuing Protocol (AMQP) - Thoughts

- Keep track of events otherwise you will have problems
- We used retries to put message in the queue
- Take care of the data into the message (security concerns)

RabbitMQ

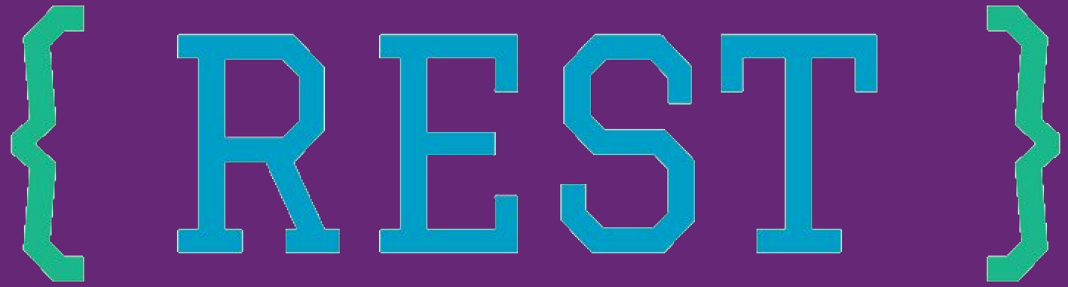

GRPC

HTTP / gRPC

- HTTP Connections are stateful (avoid open & close)
- Static Typed & Well Defined Contracts
- Reduce serialization complexity & improve efficiency
- Easy Peasy to integrate (there is code generation based in .proto)
- Traffic is binary reduce bandwidth usage

HTTP / gRPC - Thoughts

- Load Balancing and Client Load Balancing
- We used in the high throughput solution
- By definition in general we use for internal communication (a.k.a East-West)
- Not for humans, designed by machine communications


{ REST }

HTTP / REST

- Well documented Style
- “De-Facto” Pattern for Microservices Architecture
- “*Toolability*”
- Security

HTTP / REST - Thoughts

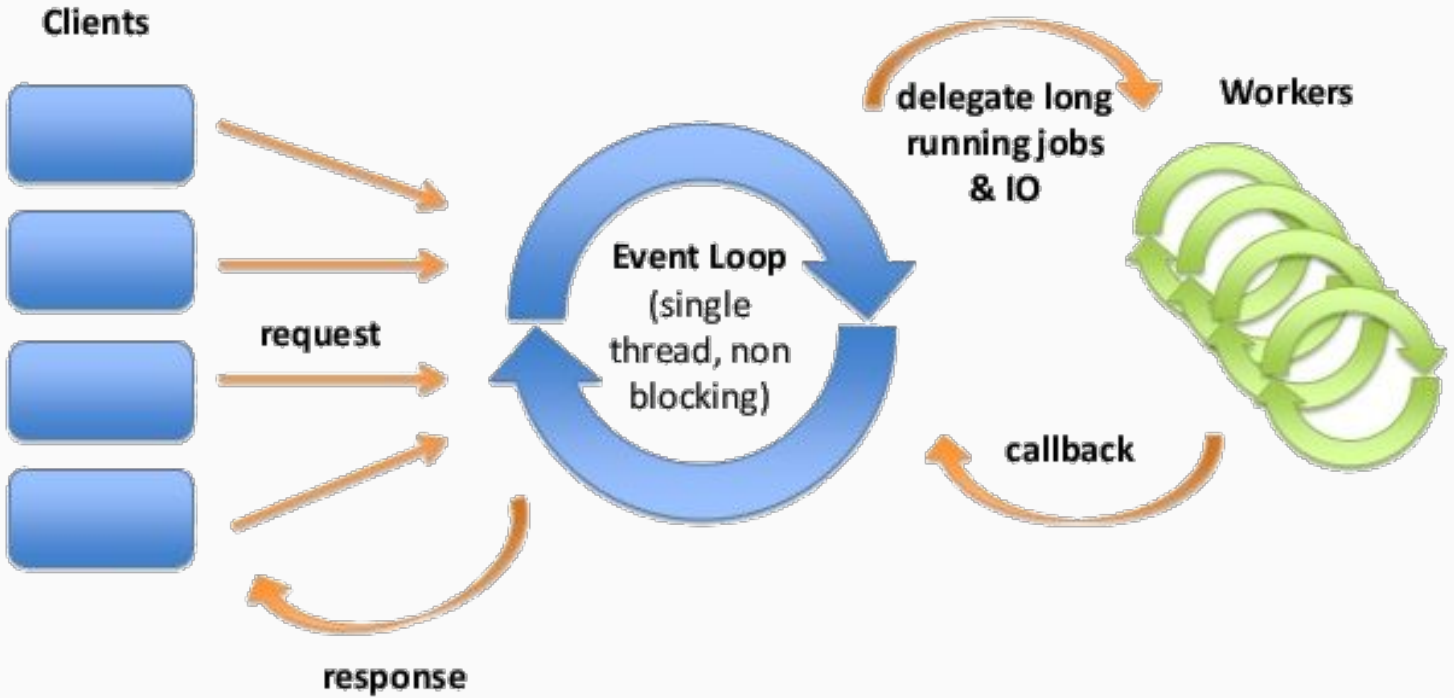
- Well documented Style
- For Humans, fits very well for External APIs (API Strategy)
- API Management provided by Sensedia offers interesting features to troubleshoot

We have tests, responsibilities and communication!

It is possible to improve more?

Reactor Pattern & Reactive Programming

Reactor Pattern & Reactive Programming



Reactive Programming

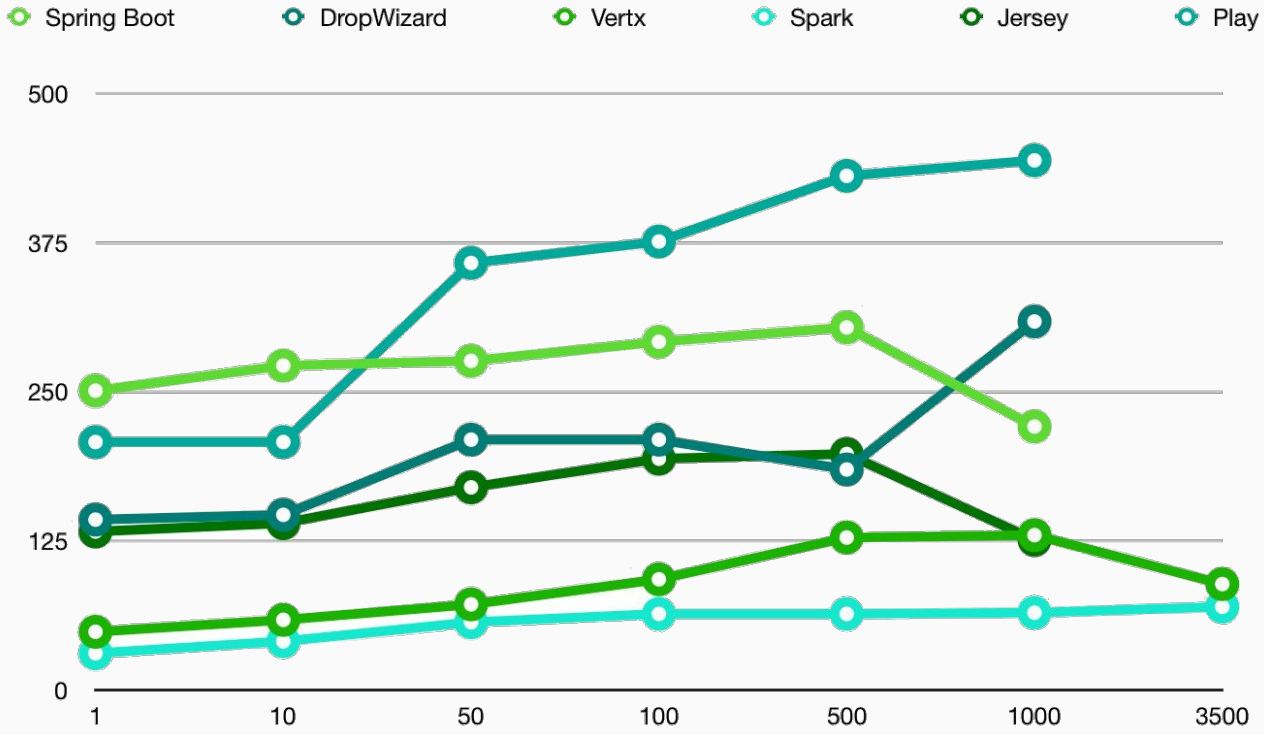
Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change.

- Easier to read, maintain and evolute
- Low memory footprint
- Scalability

VERT.X

Source: https://en.wikipedia.org/wiki/Reactive_programming

Reactive Programming



Source: https://craftsmen.nl/memory-usage-6-popular-rest-server-frameworks-compared/

Reactive Programming - Thoughts

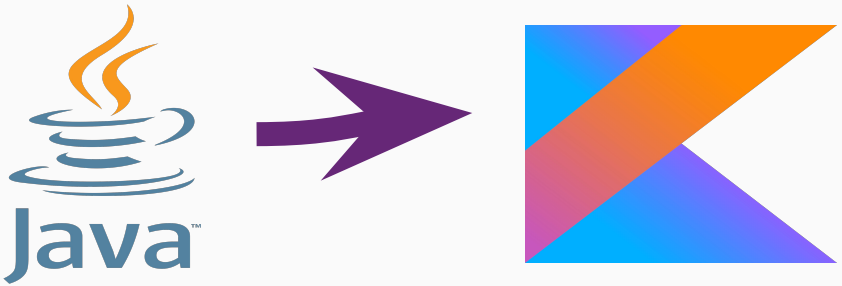
- Never blocks the event loop
- Callback hell
- Don’t look memory, look the event loop size

Language

Language

Kotlin is fully compatible with all Java-based frameworks, which lets you stay on your familiar technology stack while reaping the benefits of a more modern language.

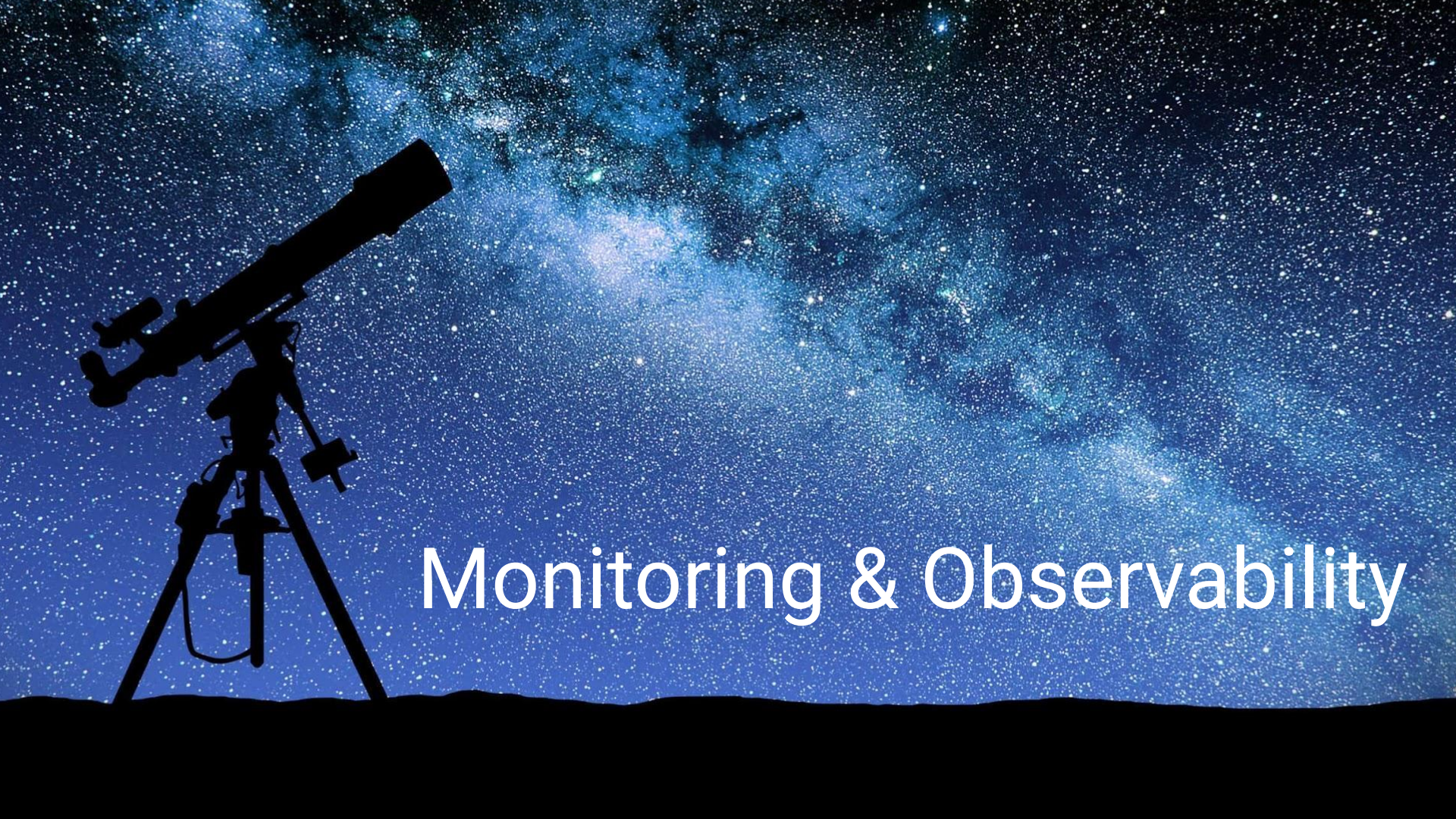
- 40% less code = Easier to read, maintain and evolute
- Fail fast principle = Time to Market



Source: https://kotlinlang.org/docs/reference/server-overview.html

Language - Thoughts

- Expertise in Java Frameworks
- Learning curve, sprint in progress


Monitoring & Observability

Monitoring is the practice of collecting signals, telemetry, traces, etc and aggregating them, and matching them against some pre-defined criteria of system states we should carefully watch. When we find that one of our signals has crossed a threshold and may be heading toward a known bad state, we take action to remedy the system

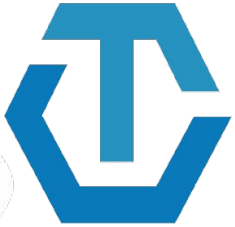
- Christian Posta

Monitoring is a subset of Observability

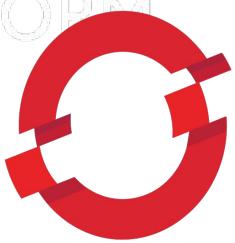
Observability on the other hand supposes up front that our systems are highly unpredictable and we cannot know all of the possible failure modes up front

We need to collect much more data, even high-cardinality data like userIDs, requestIDs, source IPs, etc where the entire set could be exponentially large





sensedia
API PLATFORM


Everything have been possible, because we formed a great Team Mindset

Team Mindset

In a fixed mindset, people believe their **basic qualities**, like their intelligence or talent, are simply **fixed traits**. Contrarily, in a growth mindset, people believe that their most basic abilities can be **developed** through dedication and hard work.

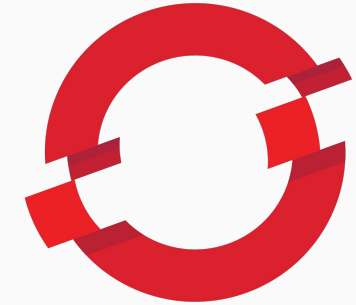
Source: https://mindsetonline.com/whatisit/about/

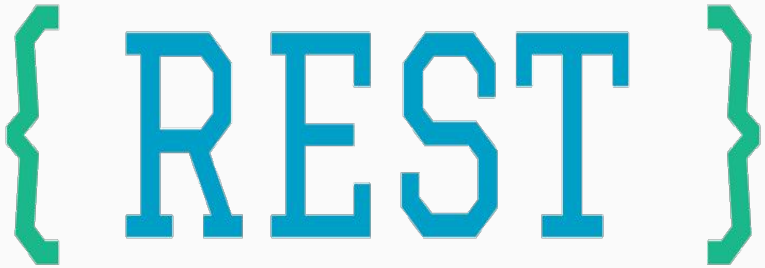
Team Mindset - Thoughts

- Hands-on
- Tech talks
- Lightning talks
- Events
  - The Developer’s Conference (TDC)
  - APIX
  - Meetups

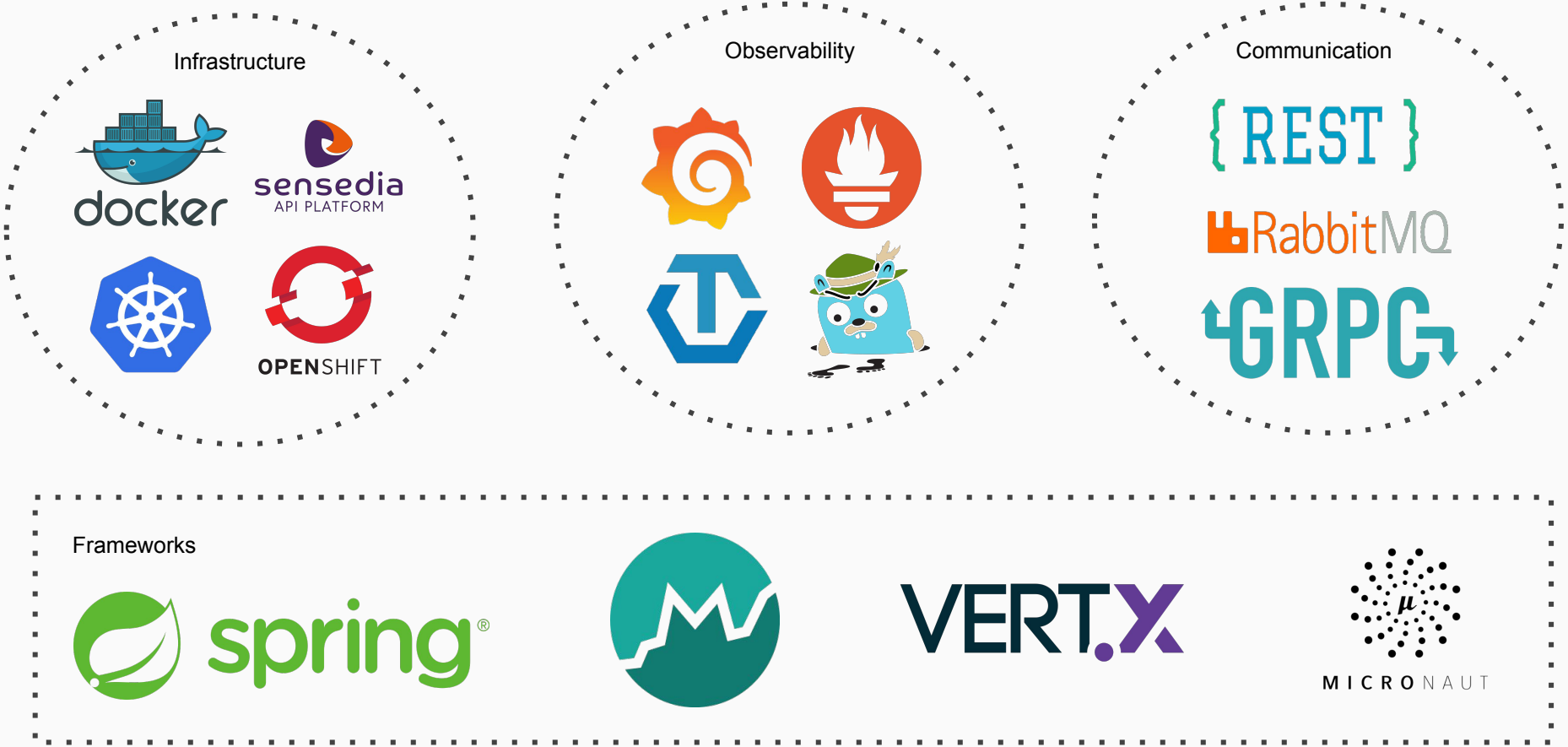
Before & After

Before

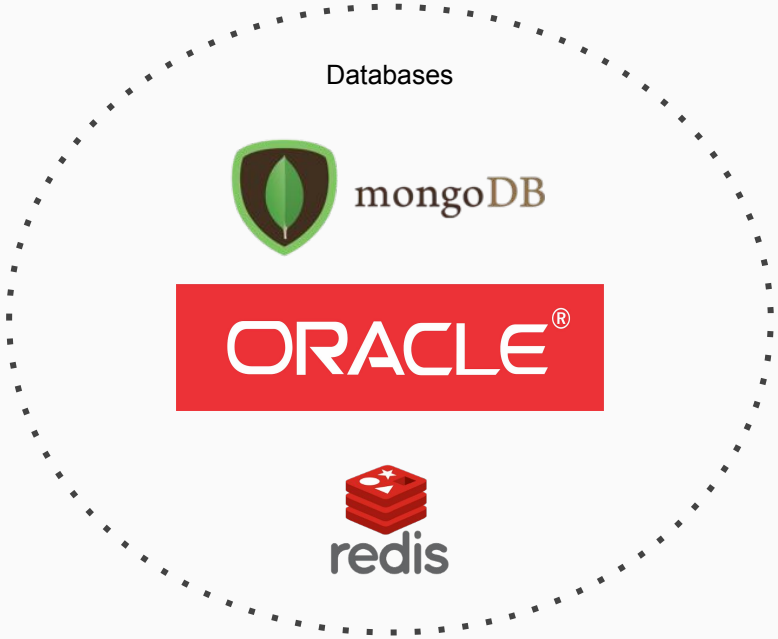




After


Infrastructure
docker
sensedia
API PLATFORM
OPENSHIFT
Observability
Communication
{ REST }
RabbitMQ
GRPC
Frameworks
spring®
VERT.X
MICRONAUT

After



Summary

2º Place

Team mindset

- Commitment
- Opportunity to grow and learn
- Feedbacks

1º Place

Last responsible moment

- Decisions
- Experience
- Maturity

3º Place

Technologies

- Understand the technologies concepts
- Business value

Thanks a million!

Questions?

 /claudioed

 /claudioed

 /larchanjo

 /luram-archanjo