Microservice-Based Agile Architectures:

An Opportunity for Specialized Niche Technologies

UNIVERSITÀ DEGLI STUDI DI PADOVA

Stefano Munari, Sebastiano Valle, Tullio Vardanega
University of Padua, Department of Mathematics

Ada-Europe 2018

June 21st, 2018

Outline

1) Introduction

2) Requirements

3) Solutions

4) Evaluation

5) Lessons learned

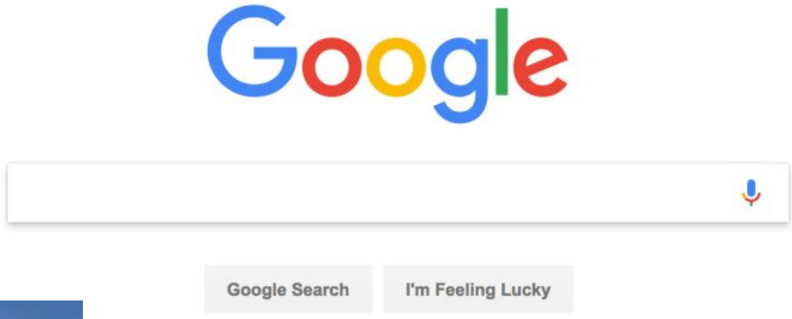
Rationale

- Software systems shape our world
  - Reliability is a major concern

Mission-critical Systems

- More and more systems are assuming **critical** traits
  - Being unavailable may incur major losses (finance, reputation, trust)

amazon web services



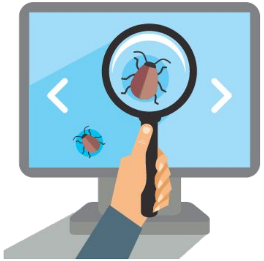
Principles

- Reliable systems can not ignore
  - **Agility**
    - **Sustain increments without giving up stability**
  - **Versatility**
    - **Facilitate reuse**
  - **Scalability**
    - **Cope with variable amounts of demand**
  - **Simplicity**
    - **Design easy-to-change systems**

1+1 = 2

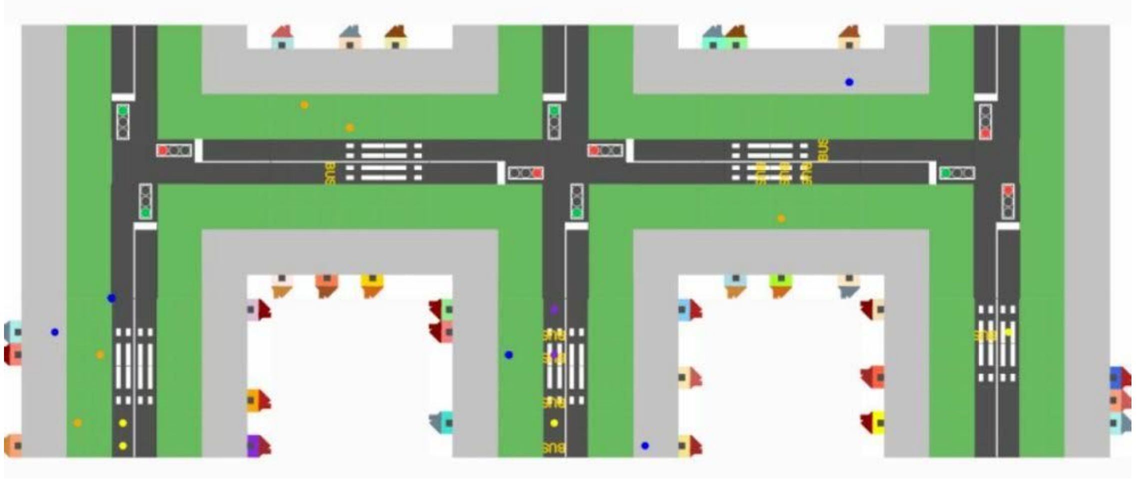
Technology requirements

- Silver bullets are long gone
- But convenient technologies should come with
  - **High-level abstractions**
    - **Inversion of Control**
  - **A modern testing framework**
    - **Automatically verify software properties**
  - **Interoperability**
    - **Connect heterogeneous technologies**
  - **Efficiency**
    - **Bear technical debt to stay on schedule**



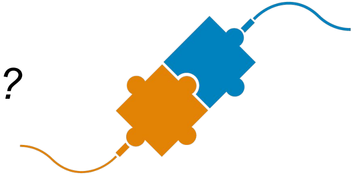
Our Simulator

- We design a system which should come with
  - Live streaming
    - Continuous state frontend
  - High availability
    - Network delays
  - Testability
    - Heterogeneous distribution
  - Malleably stable
    - Scalability, adaptation, evolution
  - Consistency
    - Discrete distributed state w/ real-time requirements



Challenges

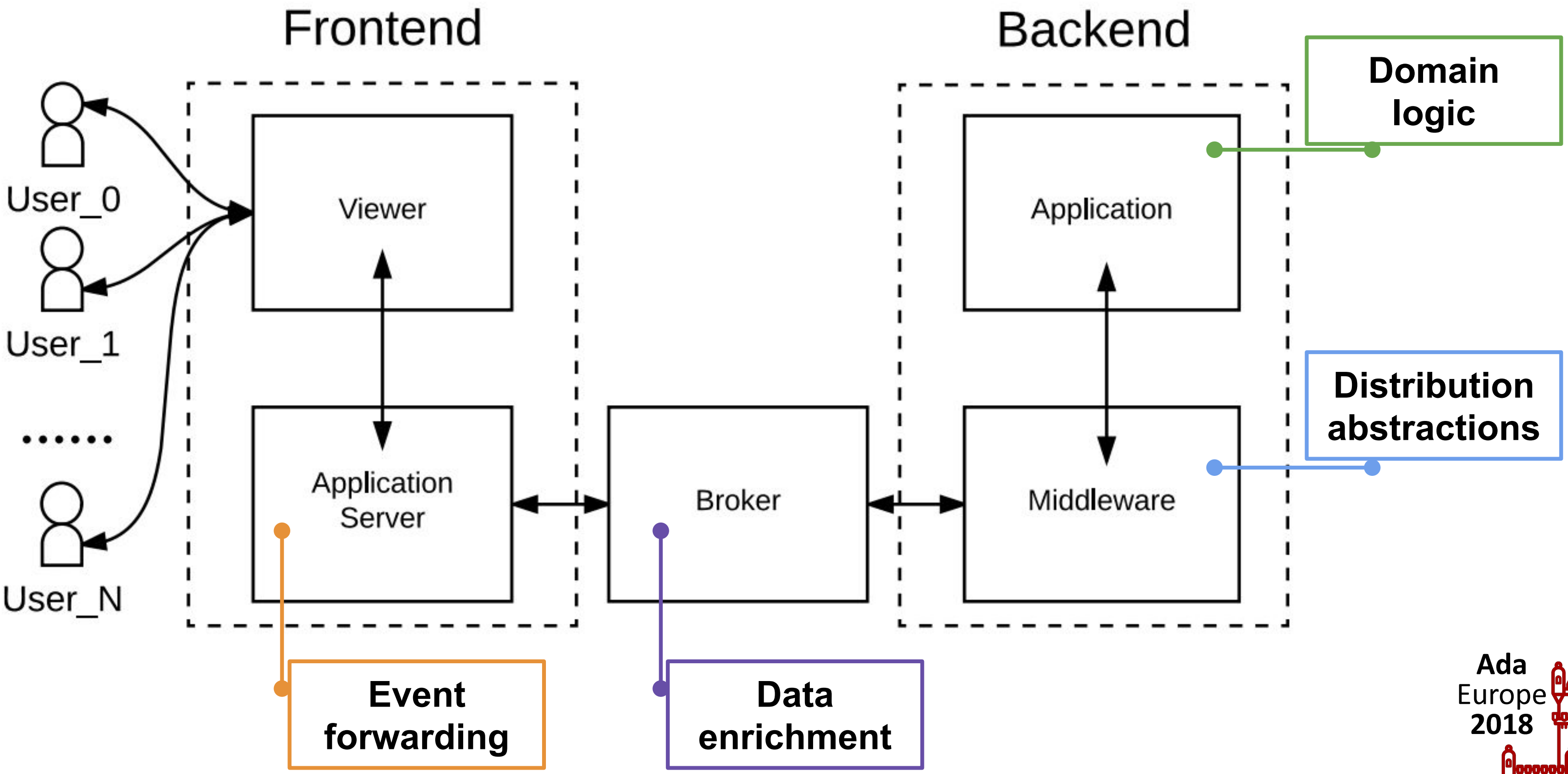
- Synchronous vs Asynchronous
  - *How to integrate a **discrete** backend with a **continuous** frontend?*



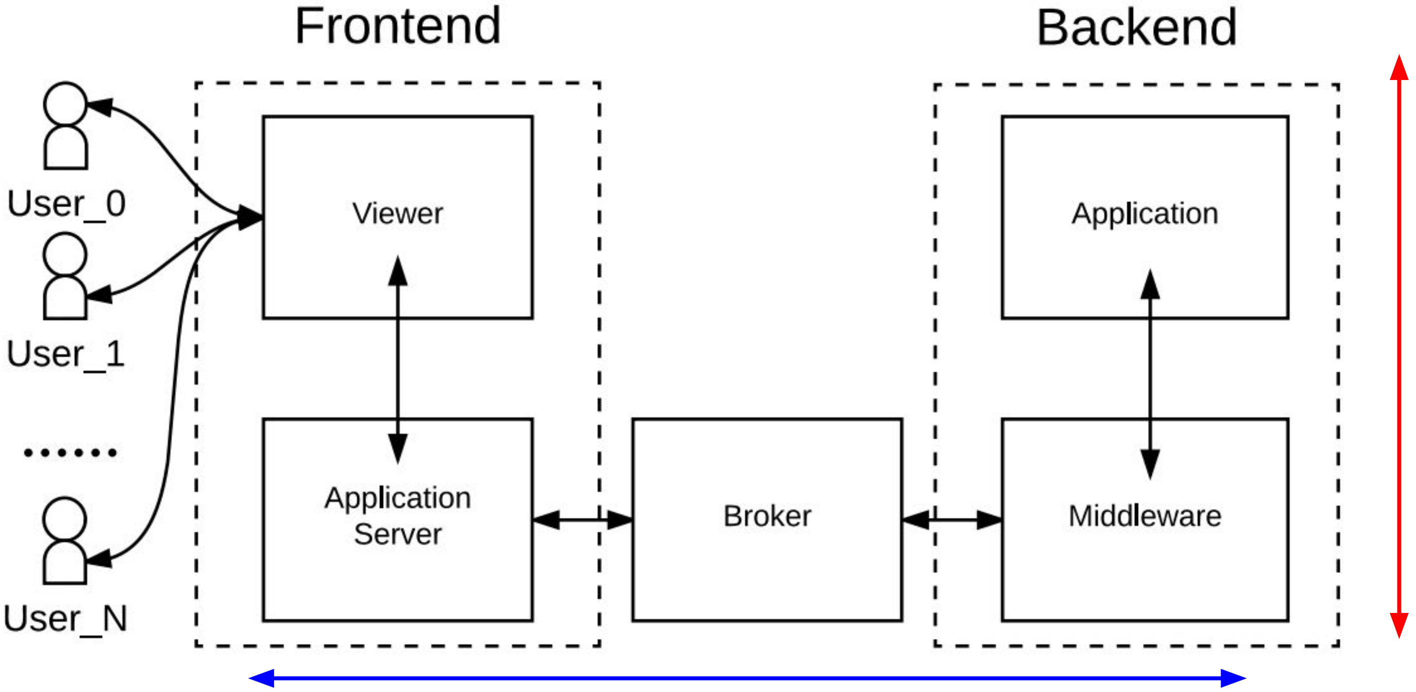
- Going (*soft*) real-time
  - *How **not** to lose consistency?*

- Being flexible and reliable enough
  - *How to guarantee **stability** when you have to **change** frequently?*

Our System Architecture



Architectural Patterns (1/2)



Layering

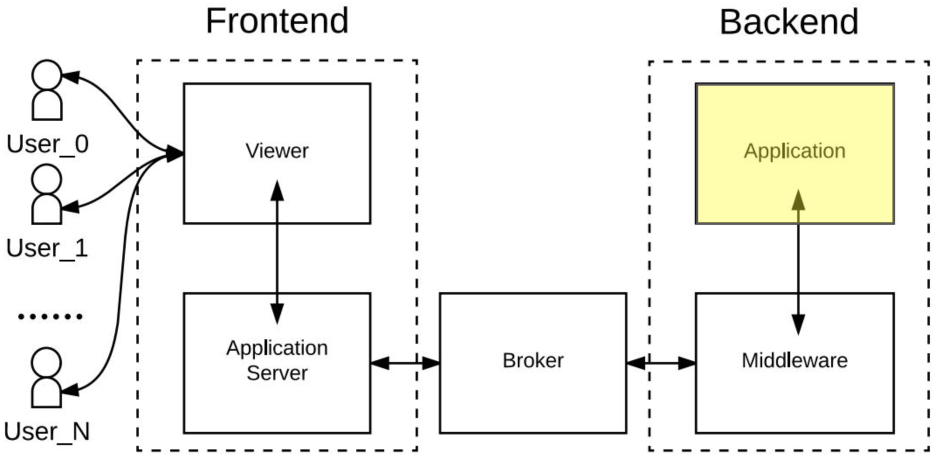
1+1 = 2 **Pipe-N-Filter**

Architectural Patterns (2/2)

- These are two of the most powerful design patterns we used
  - Layering recalls the Single Responsibility Principle
    - Simple interfaces regulate the communication between layers
    - Each layer could be seen as a *decoupled* microservice
    - Decoupling yields **versatility**
  - Pipe-N-Filter helps in building high available systems
    - Staged functionalities are encapsulated
    - Each stage can therefore be regarded as a *replicable* microservice
    - Replication of individual components leans towards **scalability**

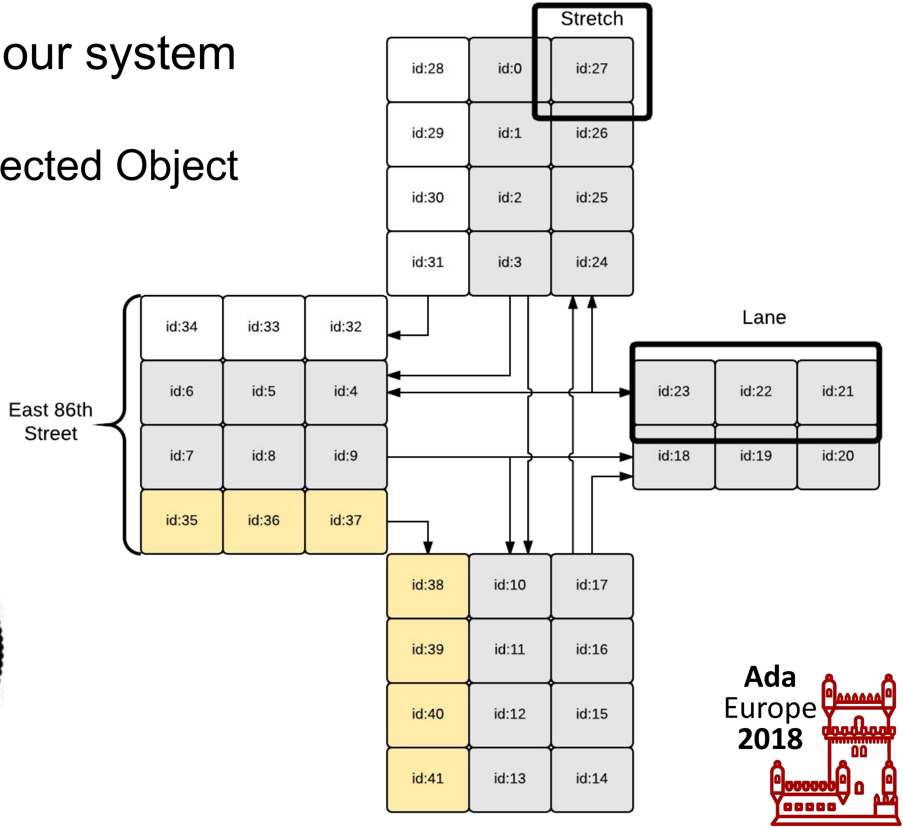
Application Layer

- We use Ada as a general-purpose programming language
  - Powerful high-level semantics for **concurrency**
  - **OOP** features are essential when building a large system



Concurrency vs. Parallelism

- Ada allowed us to boost concurrency in our system
  - Each street segment was a distinct Protected Object
  - Locks are kept only on segments
  - **High potential parallelism**



Stretch

id:28 id:0 id:27

id:29 id:1 id:26

id:30 id:2 id:25

id:31 id:3 id:24

Lane

East 86th Street

id:34 id:33 id:32

id:6 id:5 id:4

id:7 id:8 id:9

id:35 id:36 id:37

id:23 id:22 id:21

id:18 id:19 id:20

id:38 id:10 id:17

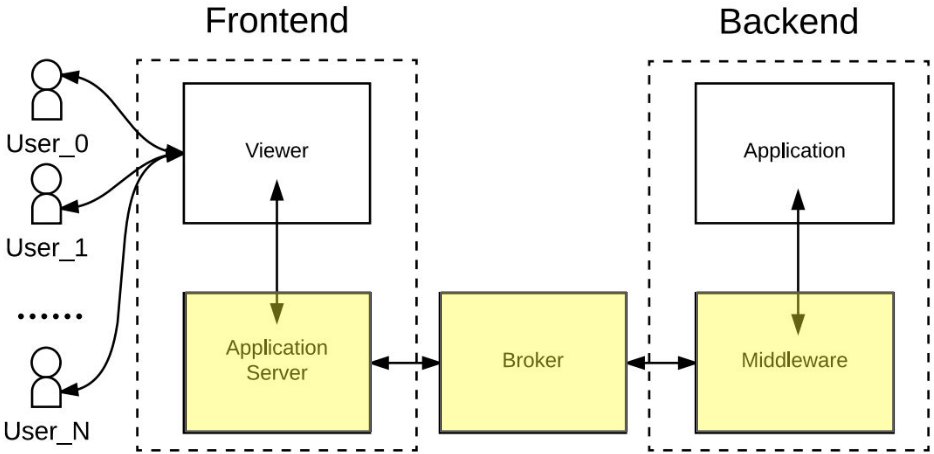
id:39 id:11 id:16

id:40 id:12 id:15

id:41 id:13 id:14

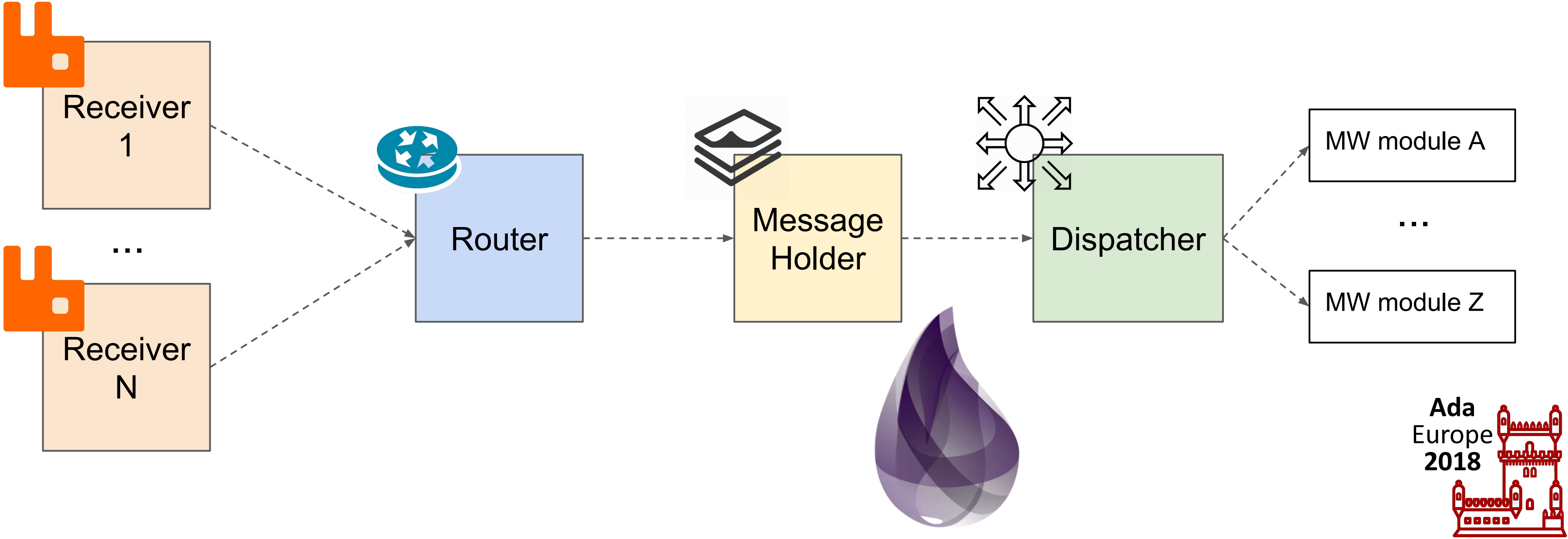
Distribution

- Elixir is used in our middleware and in our custom broker
  - Thought for distribution and fault tolerance
  - Very flexible and easy to test



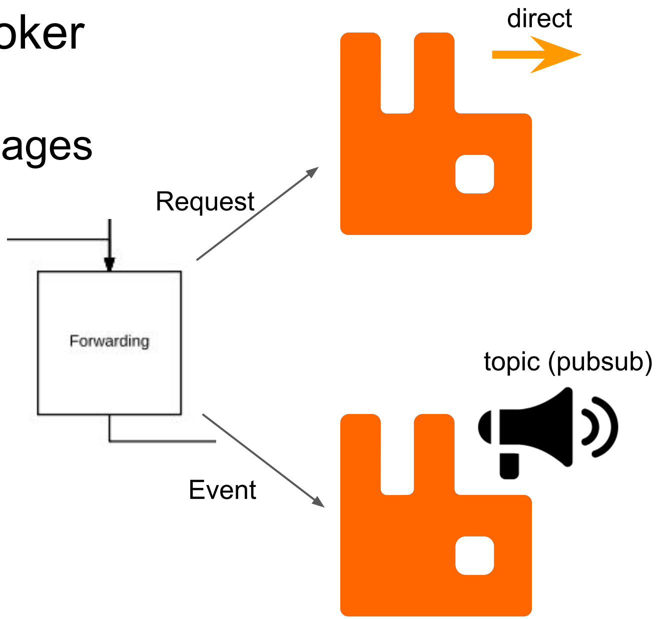
A Language Implementation for Pipe-N-Filter

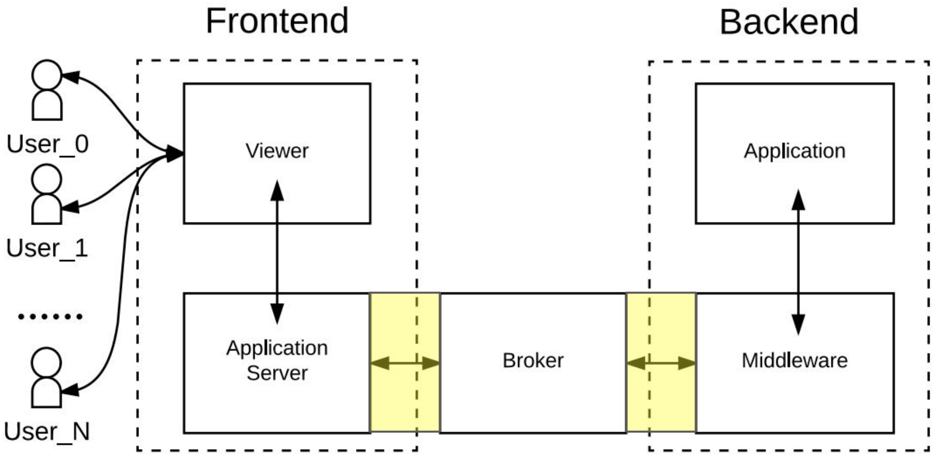
- Elixir 1.4 comes with **GenStage**
    - *"GenStage is a new Elixir behaviour for exchanging events with back-pressure between Elixir processes"*



Message Broker

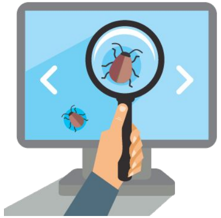
- RabbitMQ is a lightweight and simple message broker
  - **Easy-to-integrate** with *most* of programming languages
  - Clear distinction between requests and events





Operations

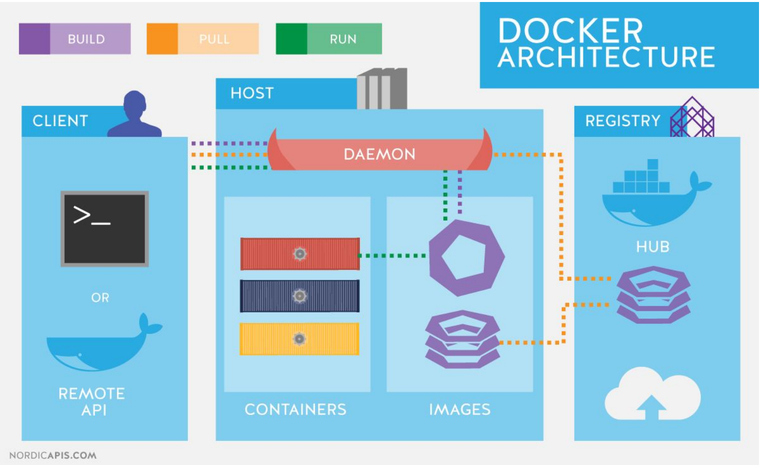
- Future complex systems will be more and more **heterogeneous**
  - → Distributed verification techniques
  - → Distributed deployments



- We tested our system continuously thanks to **xUnit**
  - ○ *AUnit* and *ExUnit*
- The productivity of a technology cannot be undervalued in the industry world
  - ○ Elixir’s Mix tool made us gain a lot of time

Integration and Deployment

- Docker helps to build microservice-based architectures
    - **Containers** isolate dependencies
    - Easy-to-manage by means of *orchestrators*
        - We used **Docker Swarm** owing to lower learning curve

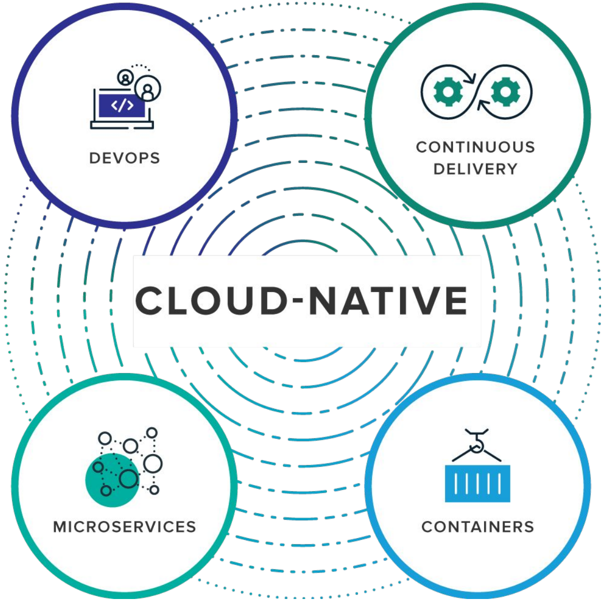


Technologies – Strengths and Weaknesses

Technology	*Ada*	*Elixir, Phoenix*	*RabbitMQ*	*Docker*	*Docker Swarm*
Pros	Good abstractions Type safety IoC	Performance Productivity IoC	Simplicity Monitoring Interoperability	Agility Productivity Isolation Simplicity	Docker-native Simplicity Easy scaling
Cons	Hard-to-integrate Poor productivity	Error reporting Loose typing	Too simple?	Not (yet) mature Low flexibility	Instability Features Adoption

Summary

- Reliability is a primary concern
  - The world is moving forward and adopting cloud natively
- Microservices can pull out the best from each technology
  - But technologies must be able to interoperate!
- We looked at different technologies during our project
  - Here is what we found out for Ada…

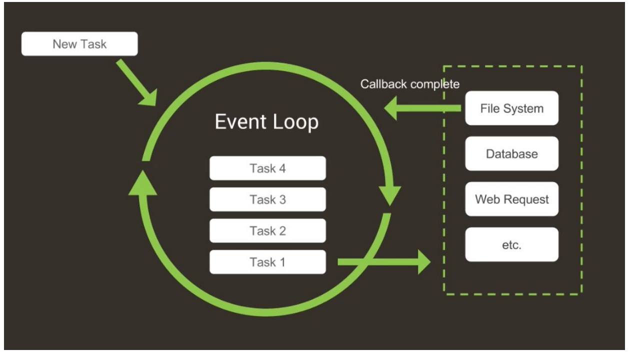


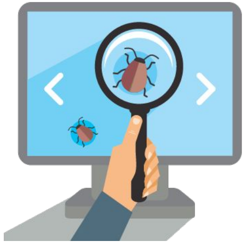
Wishes from Lessons Learned

- RESTful Ada

GET PUT POST DELETE

- Event-driven Ada
- More libraries for general purpose programming
- *Friendly* (helpful) linking errors
- Improve AUnit
    - Mocks
    - Generic packages
- Productivity
    - Syntax
    - Tools





Ada Europe 2018