Moving to the Cloud

The SeismoCloud case

Enrico Bassetti

Cloud Computing course
Sapienza, University of Rome

Roadmap

1. Introduction to SeismoCloud
2. Current architecture
3. Transforming SeismoCloud to be cloud-ready
4. Moving to the Cloud

Introduction to SeismoCloud

Introduction to SeismoCloud

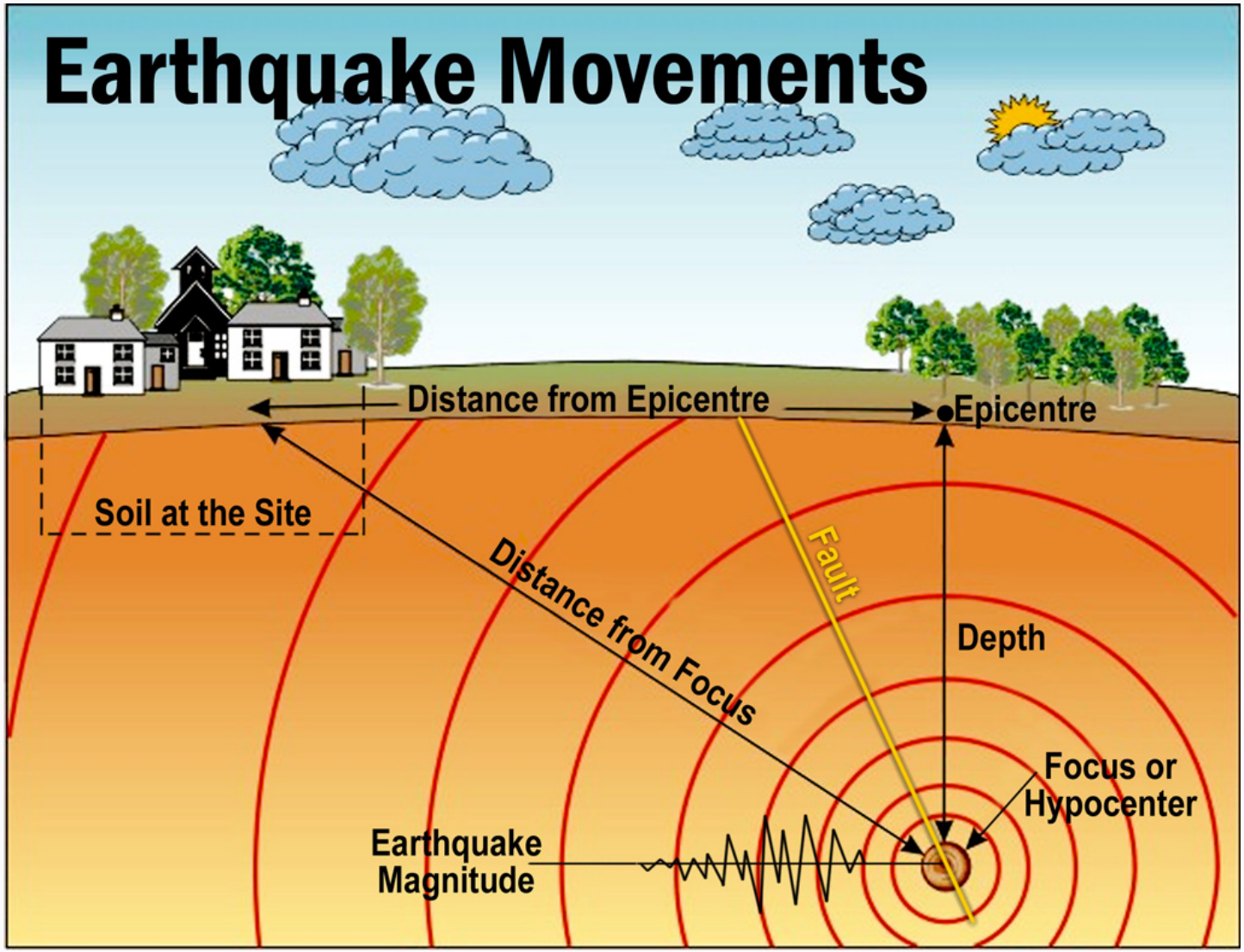
SeismoCloud is an *early warning system* for earthquakes based on a low cost sismometers' network.

The goal is to warn residents about an incoming earthquake **once it emerged on the epicenter**, within a margin of 5 $\sim$ 20 seconds.

DIPARTIMENTO DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

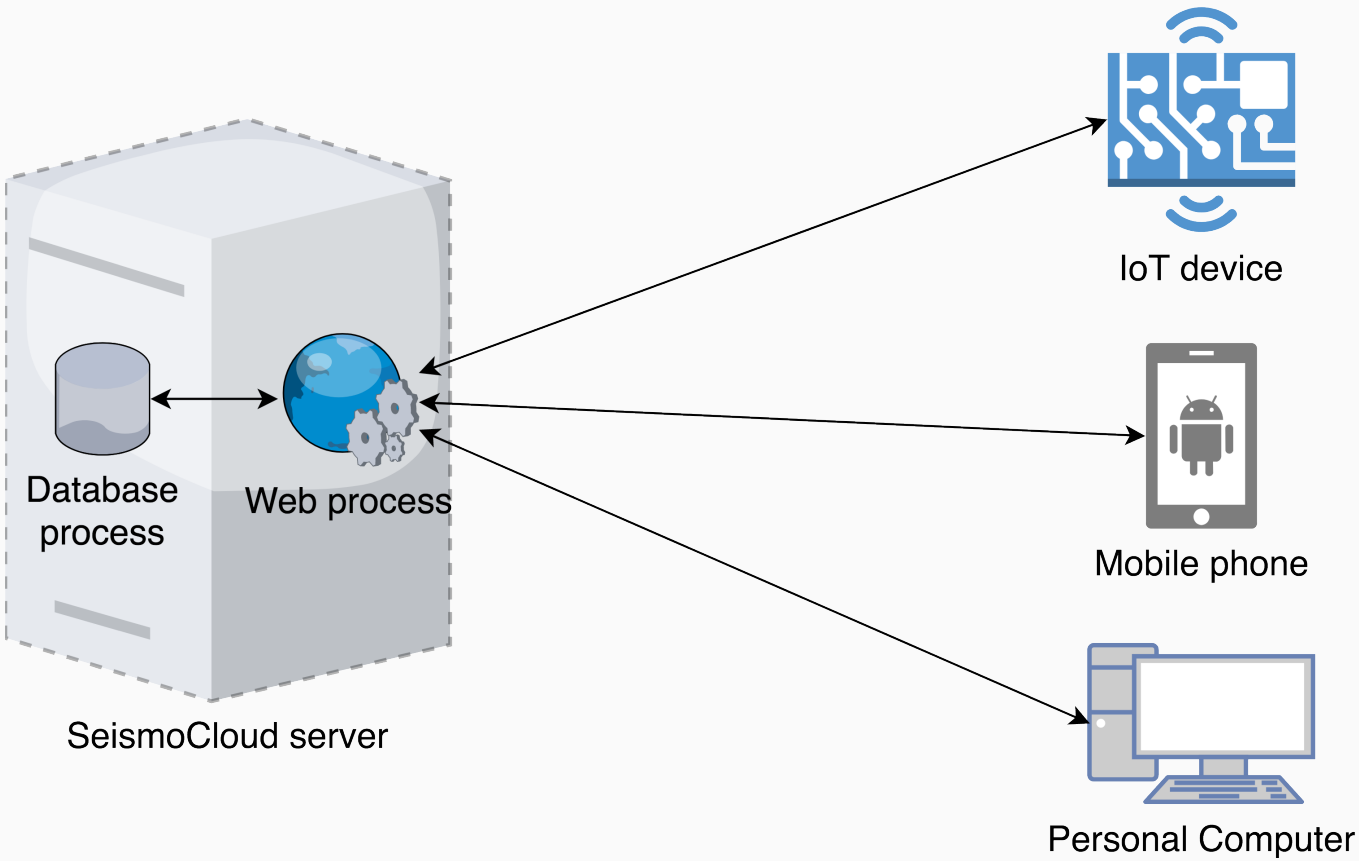
Istituto Nazionale di Geofisica e Vulcanologia

Introduction to SeismoCloud: map



Current architecture

Current SeismoCloud architecture

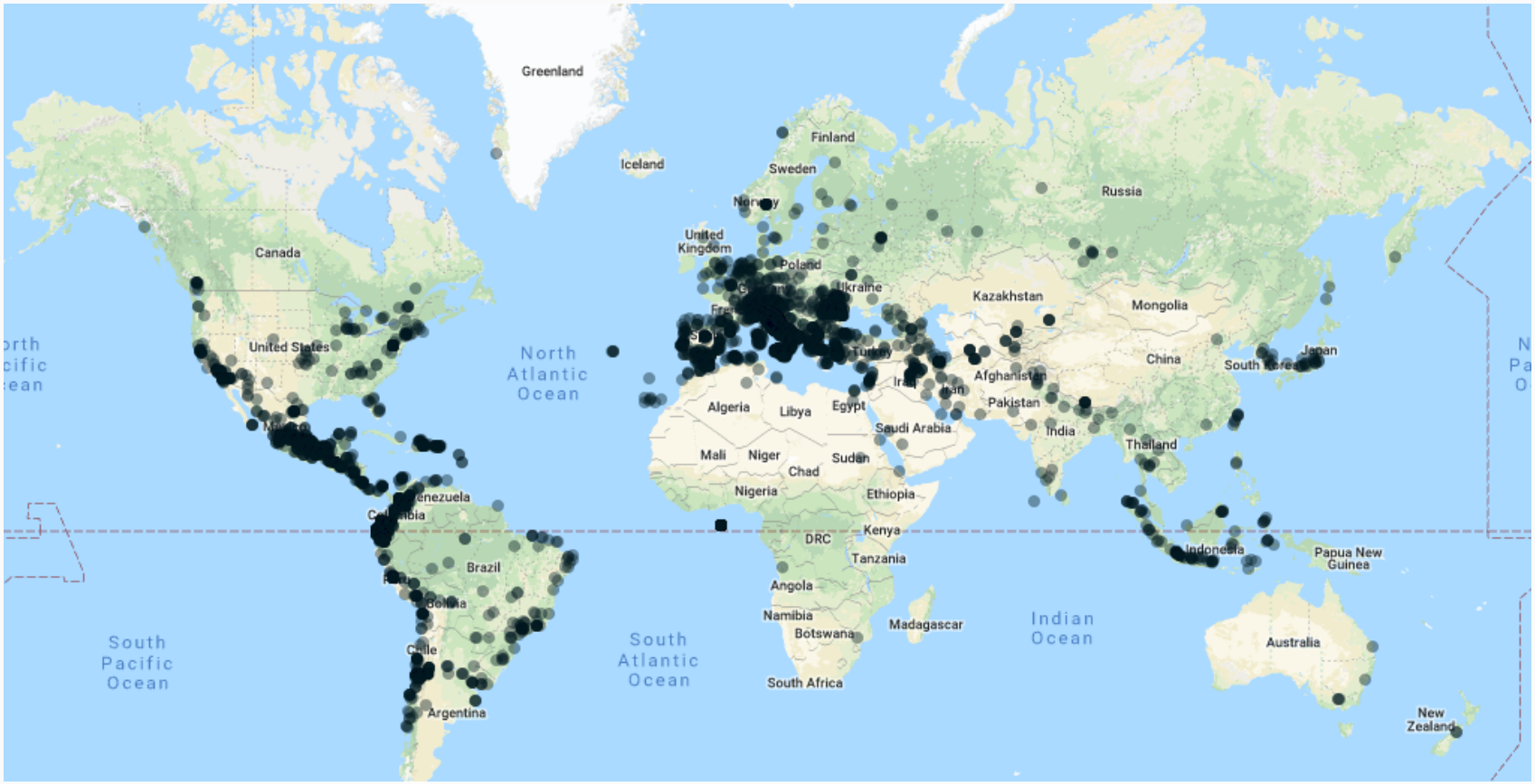


Current SeismoCloud architecture: numbers

- **Users**: $\sim$ 13k devices, nearly 10k users
- **Wide area**: wide adoption in the EU, Mexico, west coast of US and south America
- **Computing power**: 2 x 2.4 GHz Intel CPU, avg load $\sim$ 60%
- **Memory**: 5 GB of RAM, avg load $\sim$ 80%
- **Storage**: +20 GB (both current and historical data)
- **Cost**: $\sim$ 1400 € per year (server maintenance is made by us)

Data collected in a quiet situation - eg. no major earthquakes

Current SeismoCloud architecture: map



Current SeismoCloud architecture: issues

- **Limited scalability**: only vertical scalability. There is no way to scale the system horizontally due design flaws
- **No elasticity**: the system cannot adapt automatically to match workload changes over time
- **No high availability**: only one server manages (nearly) all. If it's down, the project is gone
- **Under provisioning**: the system is highly under-provisioned: necessary resources to handle a big network activity (such as a big earthquake) are missing
- **Over provisioning**: the system is over-provisioned for quiet periods of time (eg. no earthquakes)

Current SeismoCloud architecture: improvements

A recent study highlighted the fact that HTTP(s) is not the right protocol to run a system like SeismoCloud, due many technical facts. The same study suggests to implement a *Message-based, Publish-Subscription* protocol like **MQTT**, Message Queue Transport Protocol.

The MQTT protocol needs a specialized software, named *MQTT Broker*. The *MQTT Broker* software needs to run in a server, and it will become vital for the project.

Also, a gateway between MQTT and HTTP needs to be written from scratch: we named it *SeismoCloud controller*. Even the *SeismoCloud controller* will be part of the vital components.

Transforming SeismoCloud to be cloud-ready

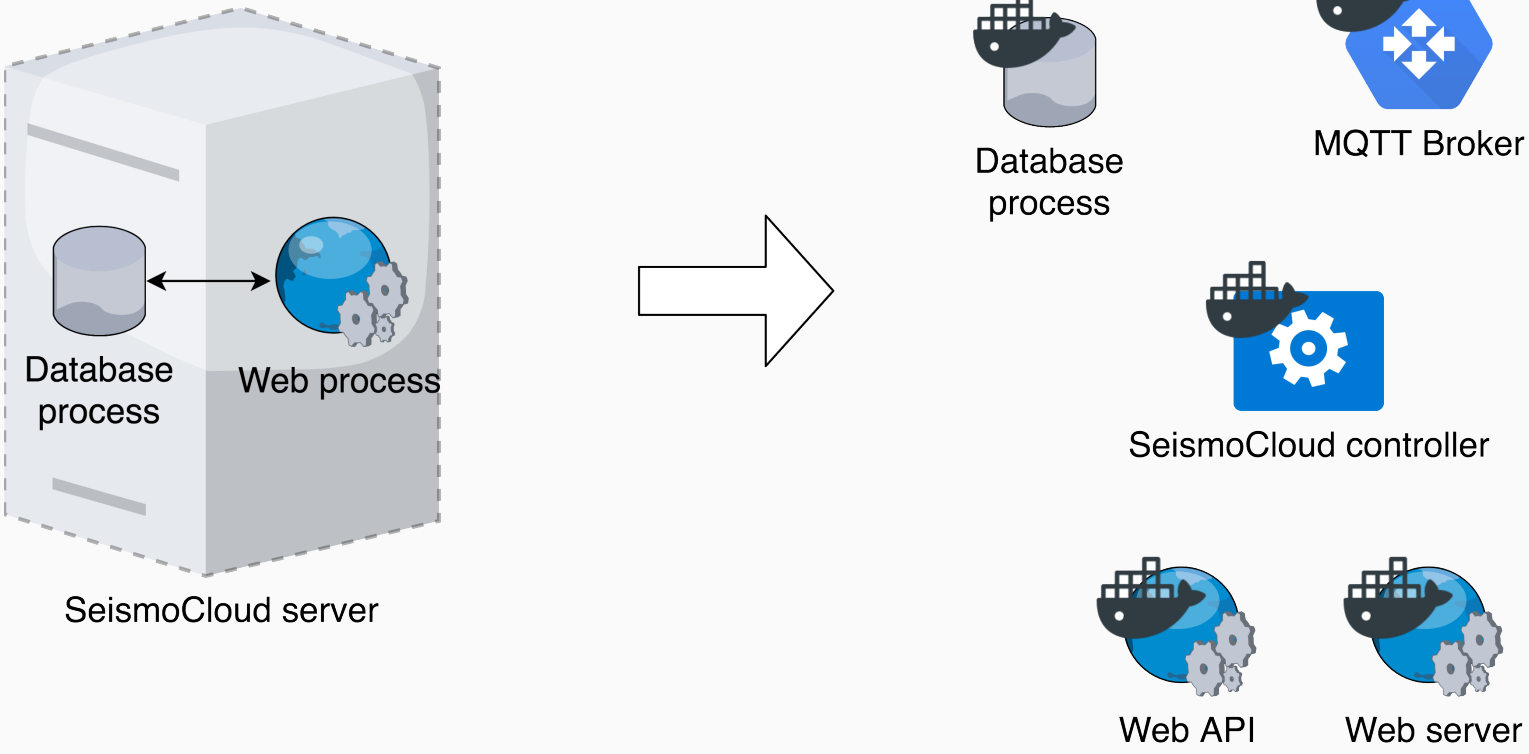
First step: Split components and dockerize

We have three different components with different requirements:

- **SeismoCloud core** (MQTT broker, SeismoCloud controller, database): high availability, high elasticity and geo-distributed, *high priority*
- **API**: high availability and scalability
- **Website**: scalability, *low priority*

They have been separated and re-written as stateless Docker containers. This is a strategic decision in order to simplify the cloud deployment and multi-tier.

First step: Split components and dockerize



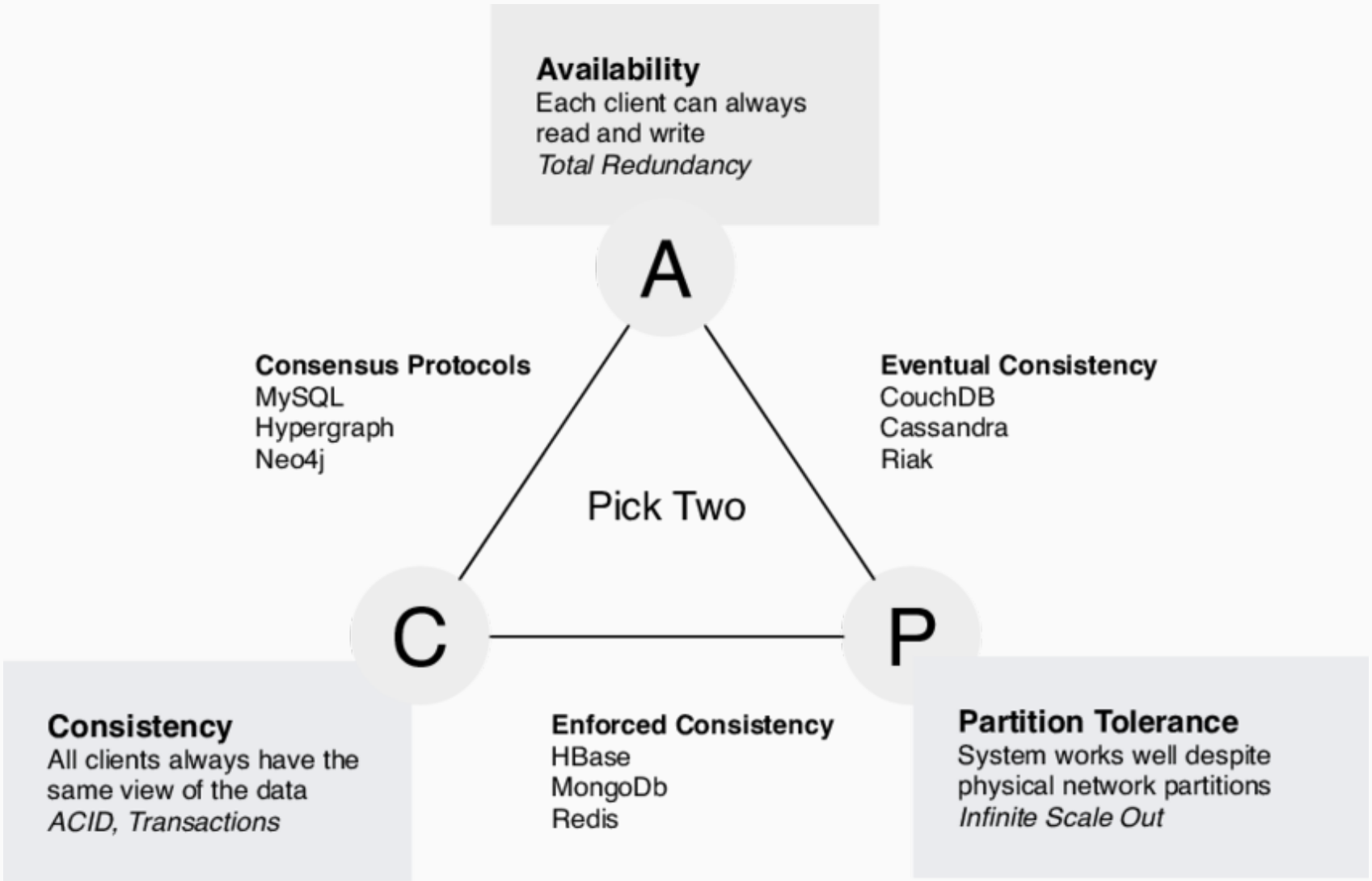
Second step: From MySQL to...

A **core** component of SeismoCloud system is the data storage. We store both technical data (eg. IoT device informations) and sensor data (such as strongness of seismic wave acceleration).

The current system is based on a Relational DMBS named *MySQL*.

As every R-DBMS, its capability to scale is very limited due to CAP theorem: a R-DBMS prioritize *Consistency* over *Availability* during a *Partition*.

CAP theorem


Availability
Each client can always
read and write
Total Redundancy
A
Consensus Protocols
MySQL
Hypergraph
Neo4j
Eventual Consistency
CouchDB
Cassandra
Riak
Pick Two
C
P
Consistency
All clients always have the
same view of the data
ACID, Transactions
Enforced Consistency
HBase
MongoDb
Redis
Partition Tolerance
System works well despite
physical network partitions
Infinite Scale Out

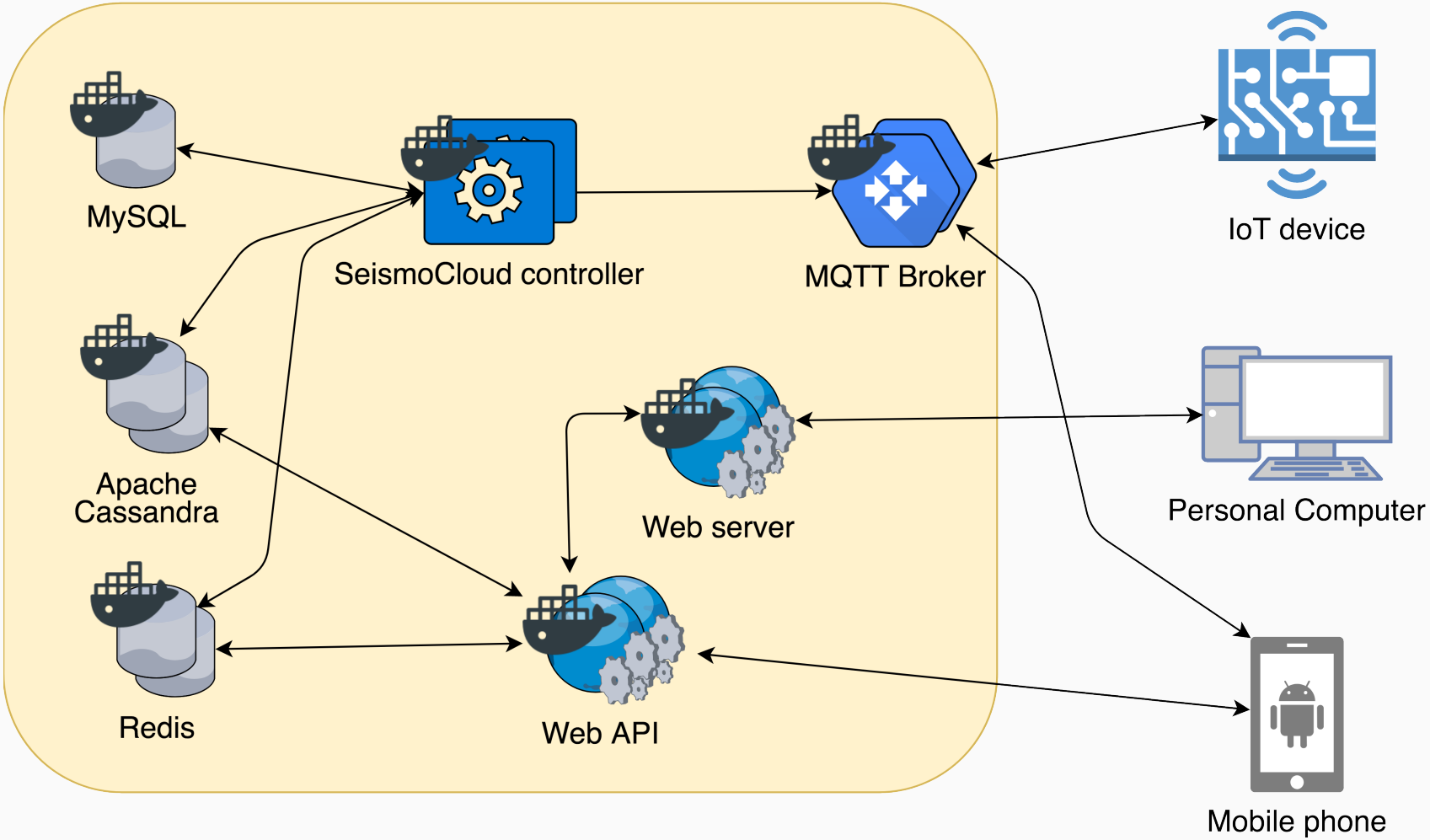
Second step: From MySQL to Apache Cassandra / Redis

For some storage *actions* we need a highly scalable system that doesn't need to be consistent at every moment. Examples of these situation are:

- Store *historical* data about earthquake waves
- Store device performance data
- Handle system and device logging
- Handle volatile data (such as session databases)

This needs are addressed with NoSQL products named *Apache Cassandra* and *Redis*.

Future SeismoCloud architecture



Refactoring costs

The overall refactoring took nearly a month of a 1-person development, and will require an estimated time of 2 week of testing (before going into production).

The total estimated cost for this operation is 7500 € (250 € / workday)

Moving to the Cloud

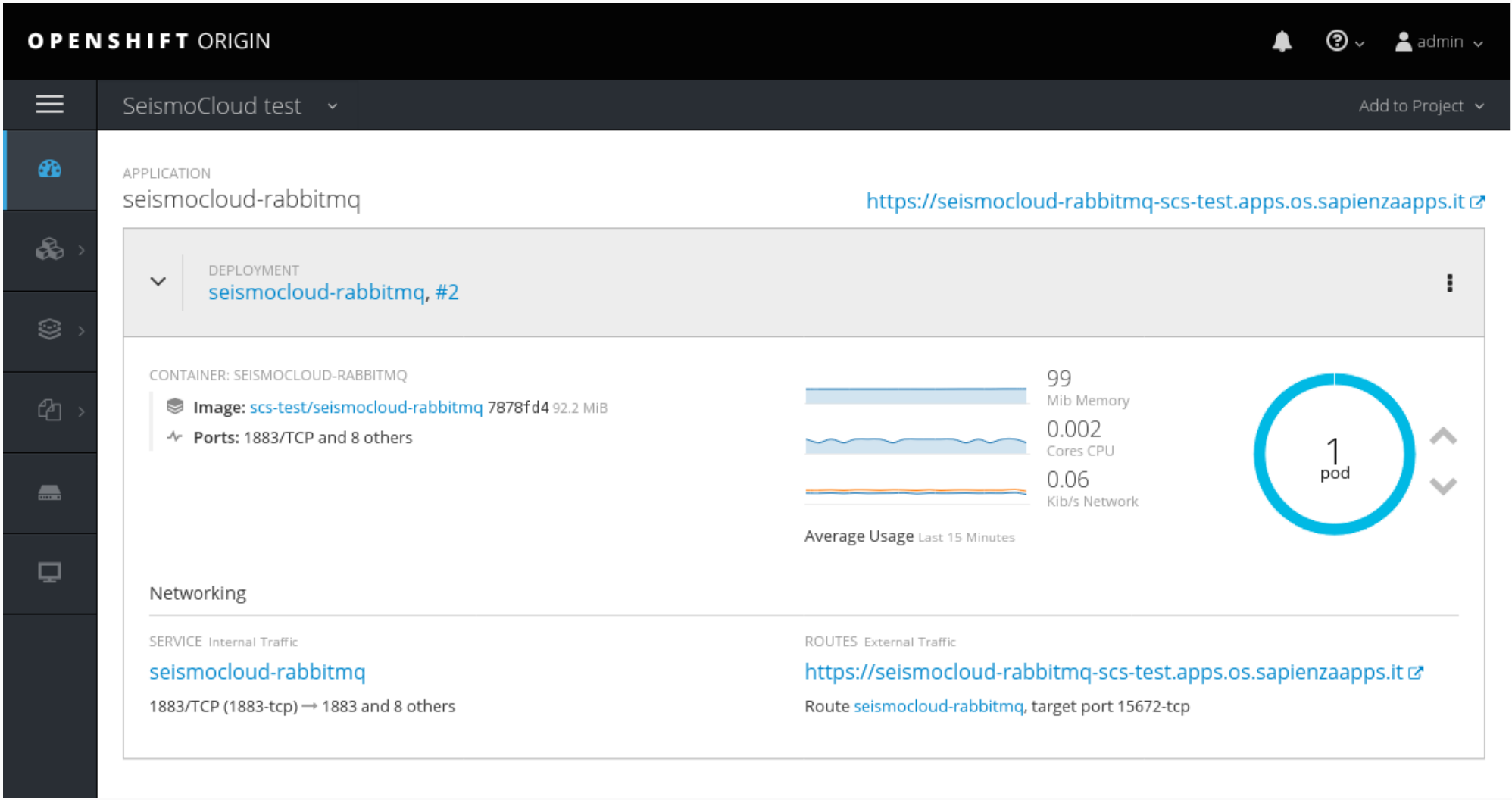
OpenShift

OpenShift is an open-source container system from **Red Hat**. It supports *Docker* containers natively using *Kubernetes* (a container *orchestrator* from Google).

OpenShift can be used both *on-premise* (even for free) or *online* (using Red Hat's own cloud).

OpenShift enables *Platform-as-a-Service* cloud.

OpenShift


OPENSHIFT ORIGIN
admin
SeismoCloud test
Add to Project
APPLICATION
seismocloud-rabbitmq
https://seismocloud-rabbitmq-scs-test.apps.os.sapienzaapps.it
DEPLOYMENT
seismocloud-rabbitmq, #2
CONTAINER: SEISMOCLOUD-RABBITMQ
Image: scs-test/seismocloud-rabbitmq 7878fd4 92.2 MiB
Ports: 1883/TCP and 8 others
99
Mib Memory
0.002
Cores CPU
0.06
Kib/s Network
Average Usage Last 15 Minutes
1
pod
Networking
SERVICE Internal Traffic
seismocloud-rabbitmq
1883/TCP (1883-tcp) → 1883 and 8 others
ROUTES External Traffic
https://seismocloud-rabbitmq-scs-test.apps.os.sapienzaapps.it
Route seismocloud-rabbitmq, target port 15672-tcp

PaaS advantages

By deploying this project on Red Hat's OpenShift cloud, we have:

- **High availability**: Red Hat has many datacenter, and OpenShift is capable to power up pods in different sites around the globe. Also it uses Amazon AWS to load-balacing its own servers (*Hybrid IaaS*).
- **High elasticity**: the underlying Kubernetes is able to scale automatically based on resource usages (eg. CPU/memory) and/or external triggers (such as the application itself)
- **Pay-per-use**: we'll pay per-use, so costs will grow as the user base grow, and we'll be able to react to usage peaks (earthquakes)

PaaS multi-tiering

By using Docker we'll be able to have a **multi-tier** architecture: many cloud providers supports Docker containers (such as Amazon EC2, Google Cloud, IBM Bluemix) as a backup, load balacing or as new main cloud provider in the future, with no changes to the code.

In fact, in this (*testing*) phase, we have an hybrid cloud system: we have both *on-premise* and *on-cloud* OpenShift instances, and some plain Docker containers in the production server **deployed with the same Docker images**.

Old vs PaaS: costs and SLA

Recap of conditions for the current architecture:

- **Costs (yearly)**: $\sim$ 1400 €
- **SLA** (server+net): 99.00% ($\sim$ 7.5 hours per month)
- **Maintenance**: software/O.S. maintenance is our duty
- **No HA, no scalability, no redundancy**

With OpenShift *Platform-as-a-Service* and new architecture:

- **Costs (yearly)**: $\sim$ 1000 € at **idle**, plus **one-time cost** of 7500 €
- **SLA** (PaaS+net): from 99.00% to 99.99% ($\sim$ 4 minutes)
- **Maintenance**: software/O.S. maintenance is offloaded to the PaaS provider - we need to maintain only our own software
- **HA, scalability, elasticity, ...**

Questions?

This presentation is available on GitHub as LaTeX source and compiled PDF:
`https://github.com/Enrico204/cloud-symposium-seismocloud`