SOSC 2019 Third International School on Open Science Cloud

Cloud Computing resources abstraction in *Function as a Service* environment

Daniele Spiga INFN-PG
spiga@infn.it

Outline

- Introduction to cloud computing evolution
  - A new “as a Service”: Function as a Service (FaaS)
  - FaaS & Serverless
- FaaS architecture
  - Basic concepts and features
  - What make it attractive
- Data Pre-Processing with FaaS frameworks
- Summary and challenges

spiga@infn.it

Intro

Engineering Effort for Effective ML

- From "Hidden Technical Debt in Machine Learning Systems", D. Sculley at al. (Google), paper at NIPS 2015

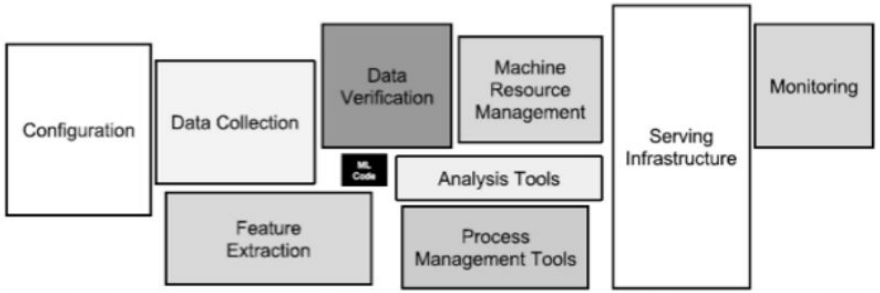


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

29

You need to know, cont'd

- Platforms: AWS, Microsoft Azure, Google Cloud, etc.
- Cloud infrastructures: Docker, Kubernetes, etc.
- BigData tools: Hadoop, Spark, HDFS, HDF5, etc.
- Databases: ORACLE, MySQL, SQLite, NoSQL, GraphDB, MongoDB, CouchDB, etc.
- Monitoring: ElasticSearch, Kibana, Grafana, Prometheus, etc.
- Streaming: Spark, Kafka, Storm, etc.
- Collaboration: Jupyter, Zeppelin, Anaconda, SWAN, etc.
- Search: ElasticSearch, Lucene, Solr, etc.
- Lexical analysis & NLP: lexer, tokenizer, scanner, etc.
- Read, write, and ask questions about everything

10

Cloud: quick reminder

https://csrc.nist.gov/publications/detail/sp/800-145/final

- **Infrastructure (IaaS → Infrastructure as a Service)**

- **IaaS**, the basic building blocks of a data center:
  - **Storage** I want to store data, lots of data, at low cost
  - **Compute** Give me a machine where I can host my services or run my applications
  - **Network** Create a “Software-Defined Network” infrastructure for me

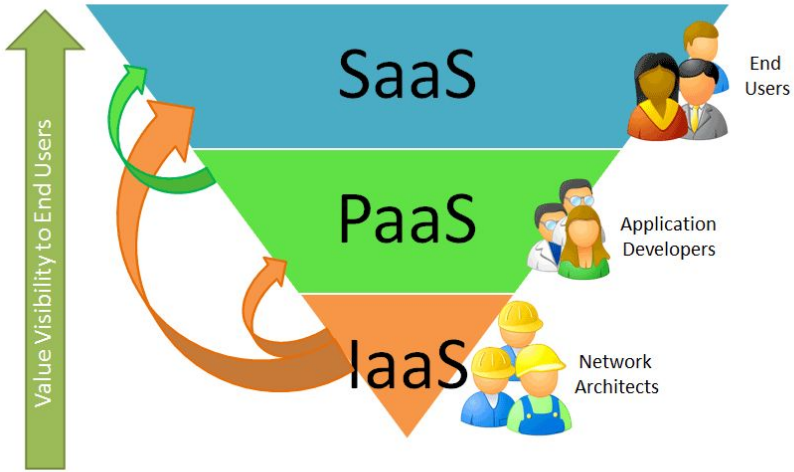
→ No need to know details, no need to contacts administrators to install something

spiga@infn.it

Cloud definition (cont)

- **Platform (PaaS → Platform as a Service)**
  - **PaaS**, a computing platform providing you with several building blocks or components that you can request programmatically or statically. For example:
    - A cluster of systems with operating system and an entire execution environment installed and configured.
    - A web server (or a clusters) with database(s), virtual storage, load balancers…
- **Software (SaaS → Software as a Service)**
  - With **SaaS**, you are directly given access to some application software. You don't have to worry about the installation, setup and running of the application. You typically access SaaS apps via a web browser.
    - For example: gmail, social media such as
    - Facebook, Twitter, etc.

What matters at the end…
are the applications.



Do we need something else?

While cloud environments made it convenient to build large-scale applications, there is still the downside of manual administration and operational interventions, such as:

- *Are the latest security fixes installed?"*
- *"When should we scale down/up?"*
- *"How many more servers do we need?*

Ideally we would avoid all those administrative tasks, and we would like to simply **focus on applications and related business value**.

And thus yes: There is a digital transformation driven by the need for greater agility and scalability

- You saw containers as building blocks for Microservices as evolution of monolithic.
- We'll see now what come later

Containers

“Wouldn’t it be nice if one could **pack the application**, **with all its dependencies**, into a dedicated box and **run it anywhere**? No matter what software dependencies the host system has installed, or where and what the host system actually is?”

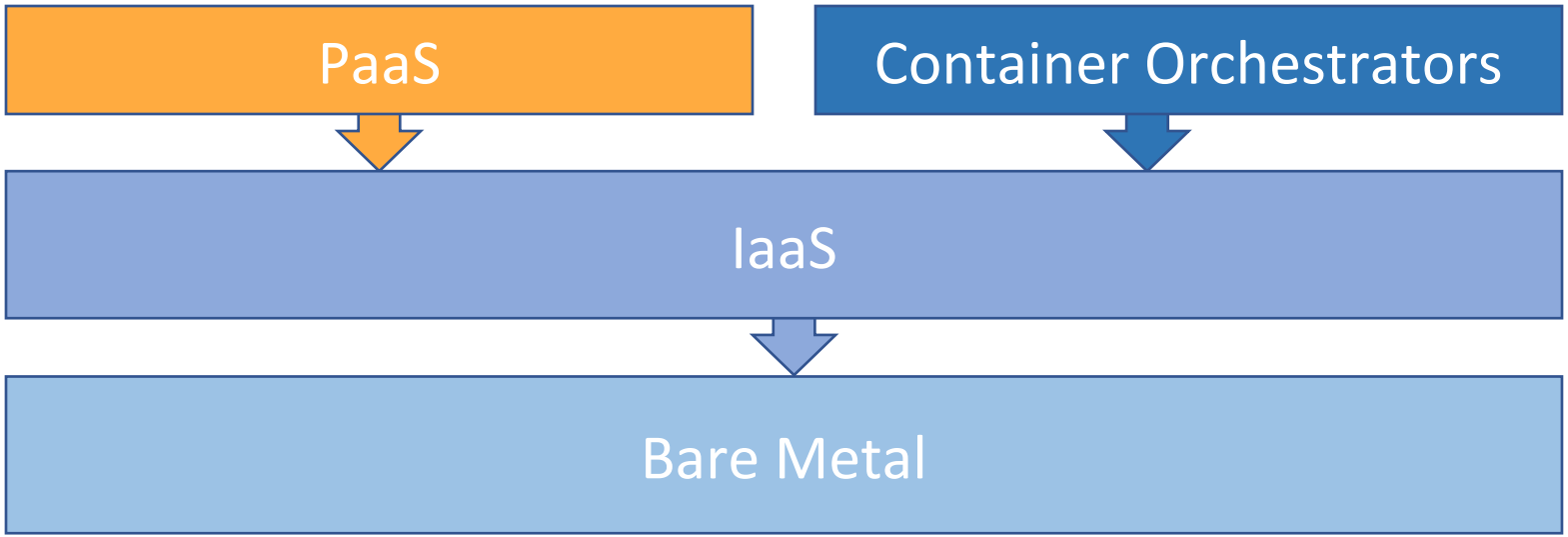
- **Yes it is and that is the idea of containerization allows all of this**.

Also, we know that containers are key pillars of microservices

Microservices architecture emerged as a **key method of providing development teams with flexibility** and other benefits, such as the ability **to deliver applications at warp speed** using infrastructure as a service (IaaS) and platform as a service (PaaS) environments.

Where we are so far…

Cloud Computing Infrastructure



Do we need something else?

So, is it all done?

- probably no, we need something else...

Wouldn't be nice if we could

- **divide our work into smaller pieces**
- **let the platform worry about manageability and autoscaling**

Great Ideally, but it's hard for the platform **to scale and manage the services**

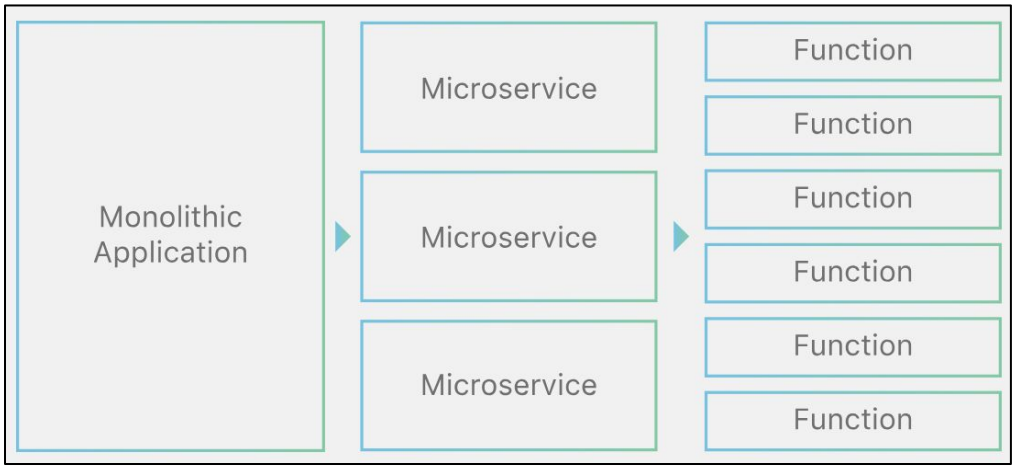
- So the suggestion might be to make them **stateless and smaller**

What is then something smaller than a "piece of application" running in a container?

Functions

With respect containers, **the basic idea of functions is to take the step further by making an application more granular** to the level of functions (**and events**).

Developers: difference here is to focus on a single function or module rather than a service with a large surface area like in the application runtime.



We've gone from monoliths to microservices to functions

Function as a Service (FaaS)

Extending the as a service model already presented we can define FaaS (Function as a Service) as

- **the ability to take a function and run it somewhere in the cloud**

Or maybe as

- the **“compute”** part of the **serverless** stack where you bring your own code.

The function contains a bespoke logic block. It is then called via some kind of registry like an API gateway, or it is scheduled or triggered by a cloud-related event (i.e., data written to Data Storage).

In other words: **FaaS is a form of serverless computing**, where you execute certain functions of your application in a **abstracted computing environment**.

Defining Server-less

Does server-less **means no servers**?

No, it is about deployment & operations model and **means worry-less about server operations and management,**

No servers to manage

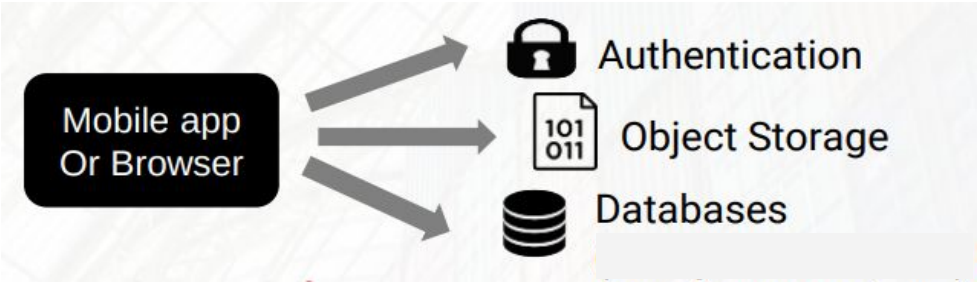
Just code to develop and execute

Runs code **only** on-demand on a per-request basis over transparent & dynamically provisioned resources

Serverless vs FaaS

Let's consider serverless as an **amalgamation of two distinct points** as follows:

❑ **MBaaS, aka Mobile Backend-as-a-Service**:
The use of 3rd party services/applications (in the cloud) to handle the server-side logic and state.



❑ **FaaS, Functions-as-a-Service**: the use of 3rd party stateless compute containers to handle the server-side logic. These containers are event-triggered and may last for only one invocation i.e. ephemeral.

At its core, serverless computing provides runtimes to execute code, which is also known as function as a service (FaaS) platforms.

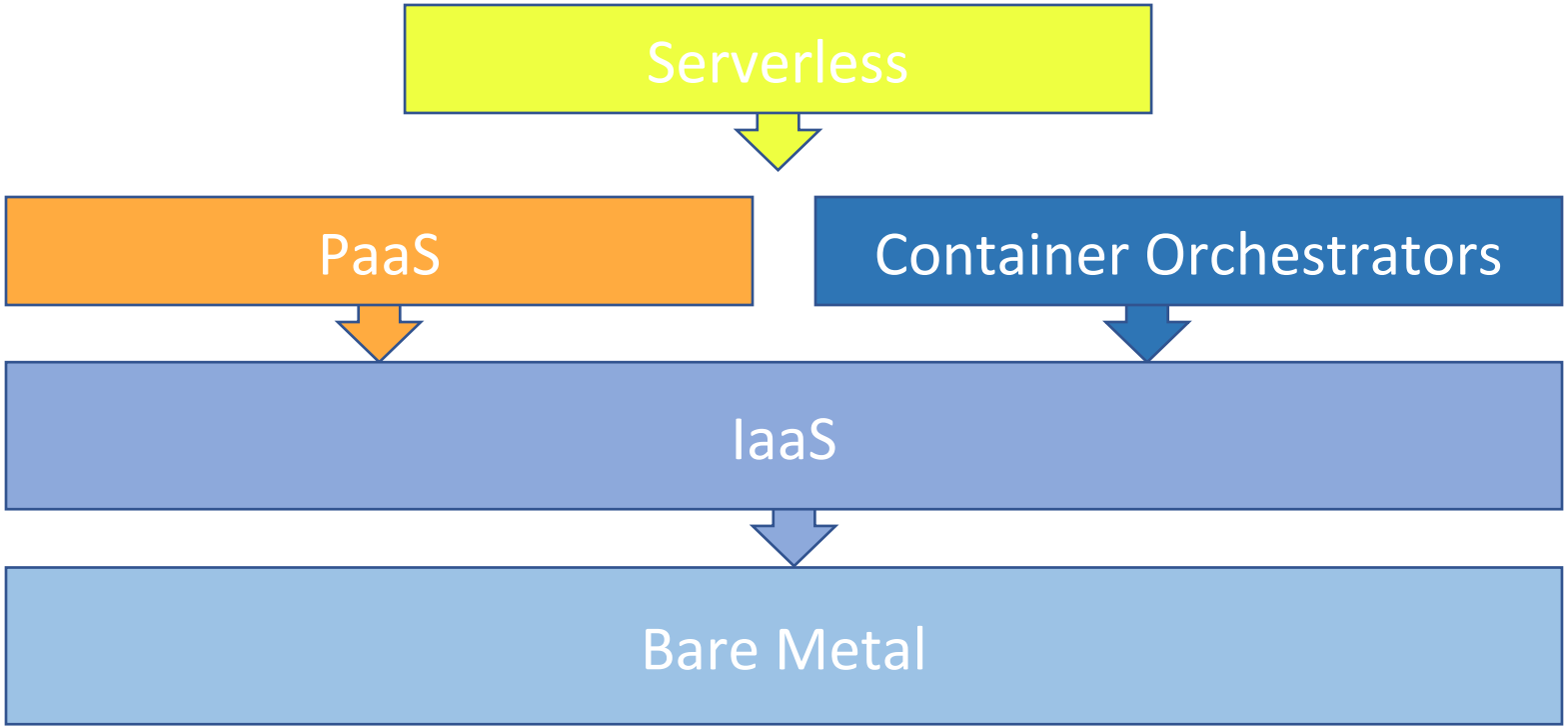
Ok, but in the end

What is serverless? Or better how we intend it in this lecture?

A cloud-native platform

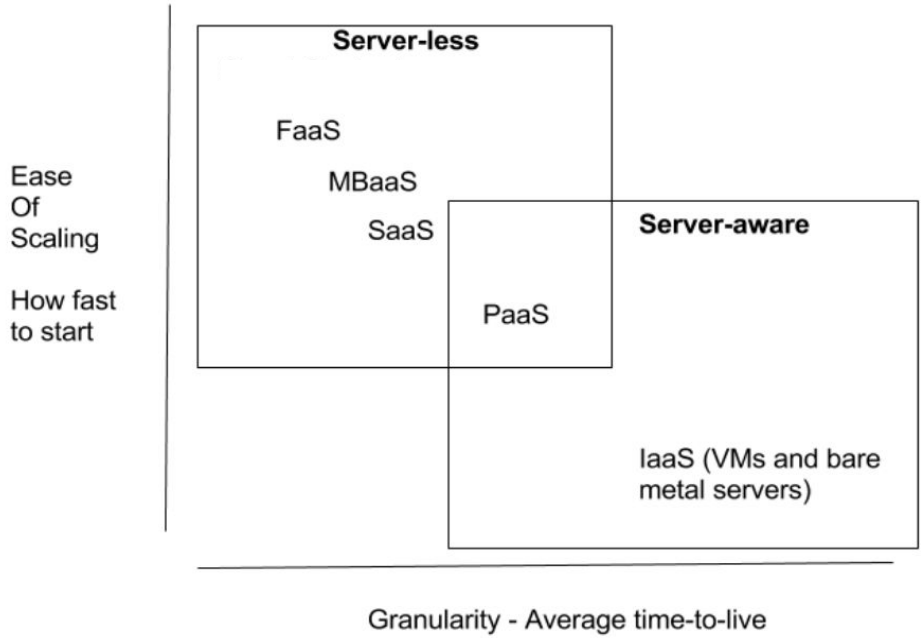
for
short-running, **stateless** computation
and
event-driven applications
which
scales up and down instantly and automatically
and
(charges for actual usage at a millisecond granularity)

Where this positions ?



Cloud Computing Infrastructure

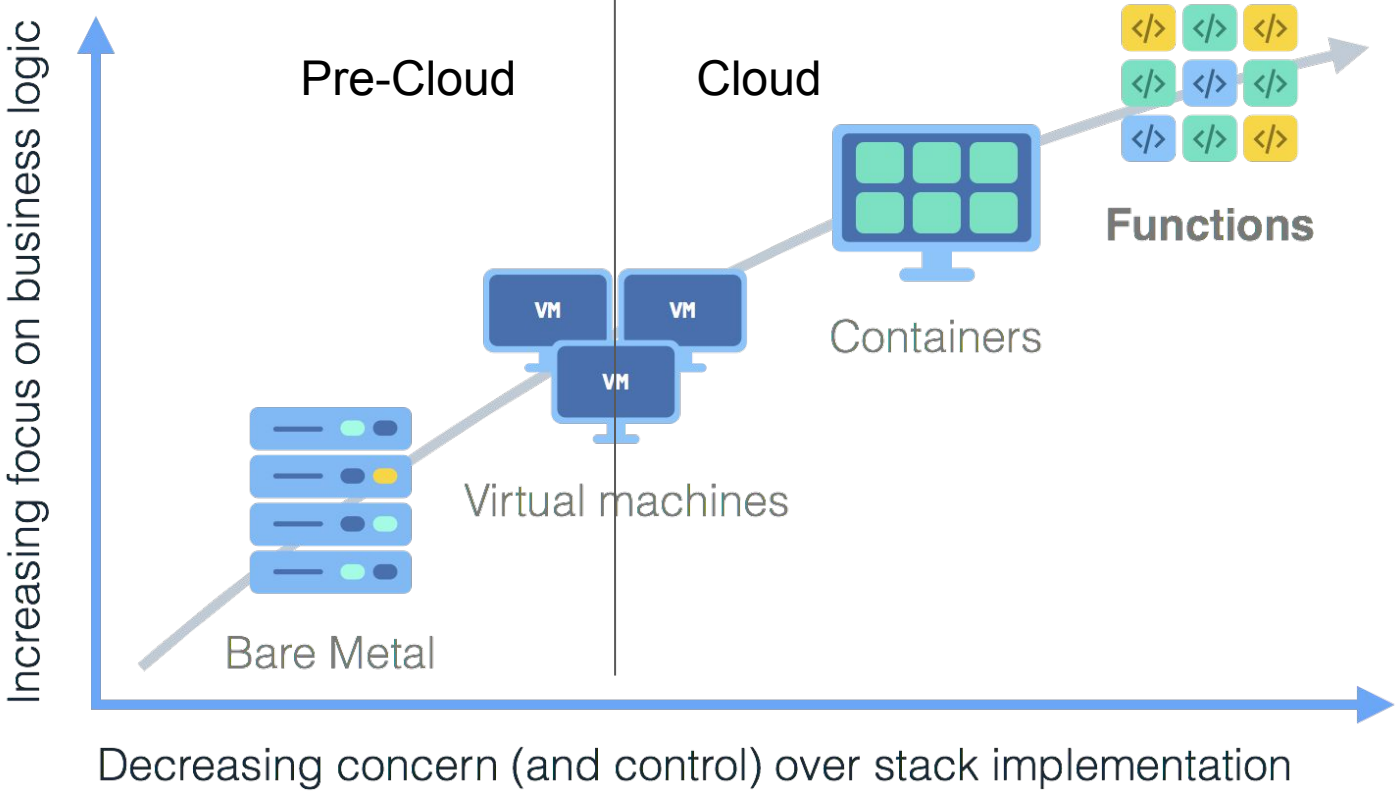
Cloud computing: server-less vs server-aware



If your PaaS can efficiently start instances in 20ms that run for half a second, then call it serverless."

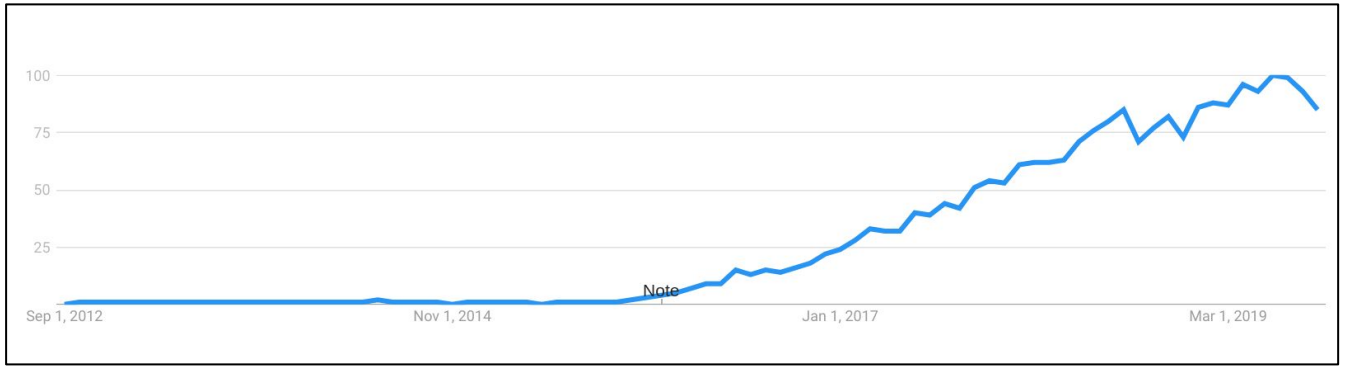
- Adrian Cockroft (2016)

Wrapping-up

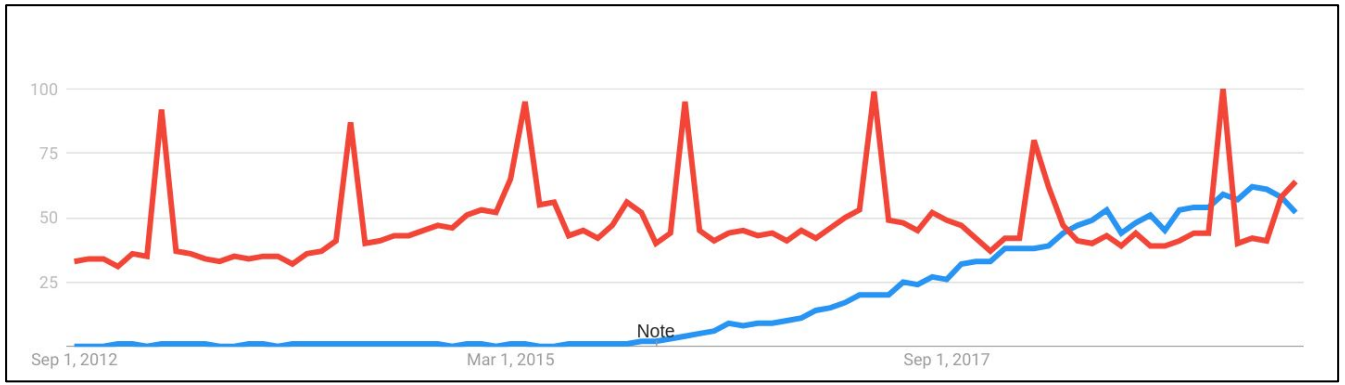

Pre-Cloud
Cloud
Increasing focus on business logic
Functions
Containers
VM
VM
VM
Virtual machines
Bare Metal
Decreasing concern (and control) over stack implementation

Gaining attention…

Google Trends



FaaS



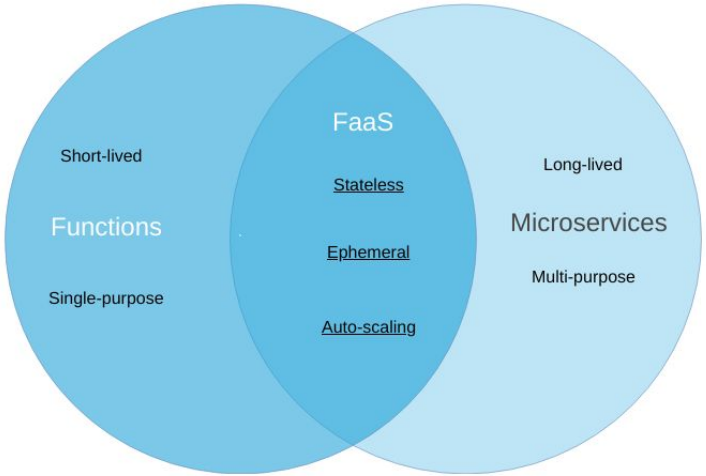
FaaS

vs

PaaS

A final question is

But still, there will be space for both **microservices and FaaS to co-exist**?

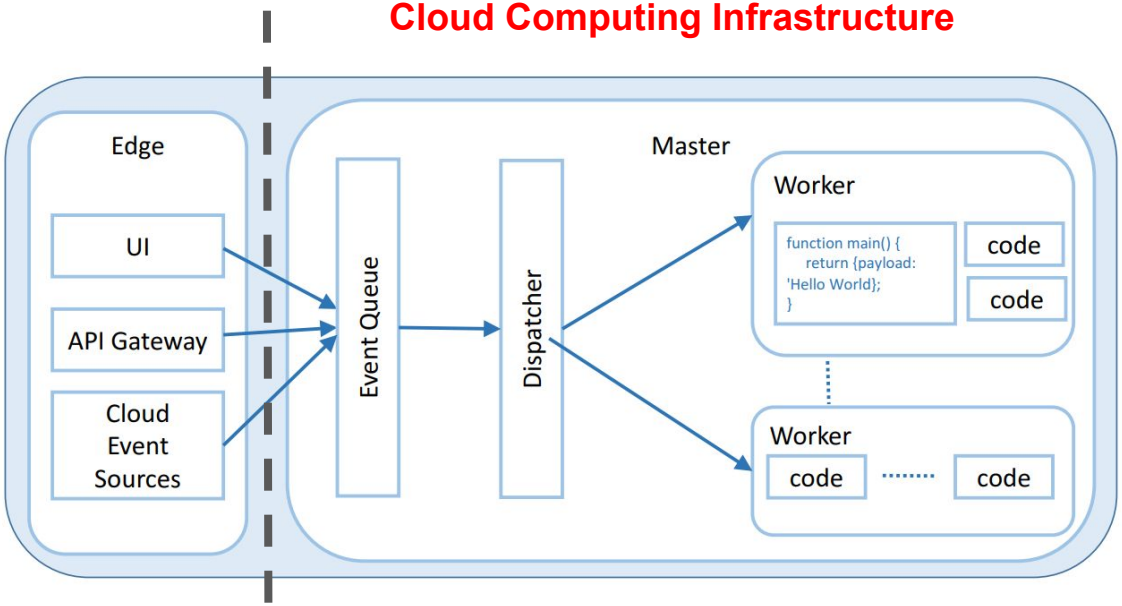


This high-level flow remains the same as the traditional approach. The key difference is that, in case of a function, the container is created and destroyed by algorithms used in FaaS platforms and the operational team have no control over that.

FaaS Architecture: Basic Concepts

Events:
In the context of functions are things that happen within your computing environment, that you might want to take action on.

Triggers:
is what must be associate with functions so that they will execute when an event is fired.



Data: When an event triggers the execution of your Cloud Function, data associated with the event is passed via the function's parameters

Triggers

A trigger defines how a function is invoked and a function must have exactly one trigger.

- Triggers have associated data, which is often provided as the payload of the function.

Triggers might be:

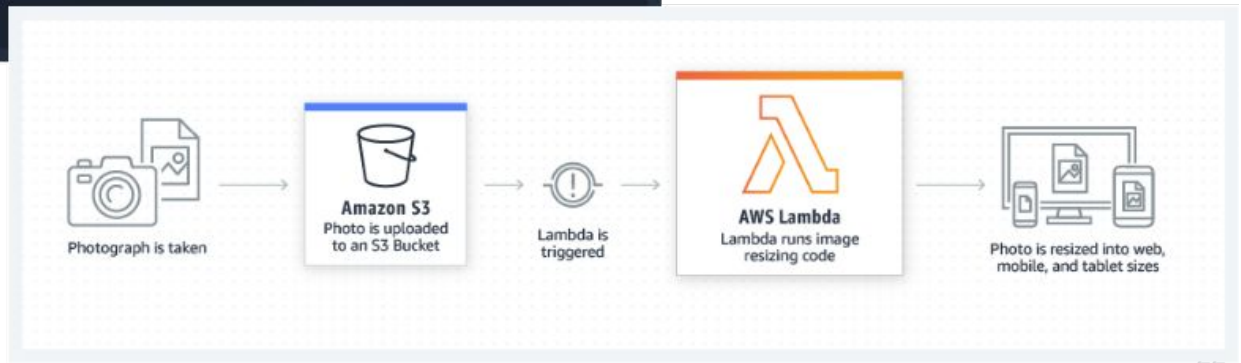
- changes to data in a database,
- files added to a storage system
- Any cloud event basically can be a trigger
  - a new virtual machine instance being created
- Cron, to implement scheduled processing
- Any HTTP (cli, browser)

Several FaaS FW allows also to concatenate functions whose trigger theirself

Example

In the Amazon world, serverless computing is called AWS Lambda.
This is how it works (picture from Amazon):

A simple AWS Lambda example:



Hands-on will provide more details

Binding of triggers to functions

FIX ME

Functions and triggers are bound to each other on a many-to-one basis.

- You cannot bind the same function to more than a single trigger at a time.
- however, have the same trigger cause multiple functions to execute by simply deploying two different functions with the same trigger.

Binding to a function is a way of declaratively connecting another resource to the function; bindings may be connected as *input bindings*, *output bindings*, or both.

It is possible to mix and match different bindings to suit your needs. Bindings are optional and a function might have one or multiple input and/or output bindings.

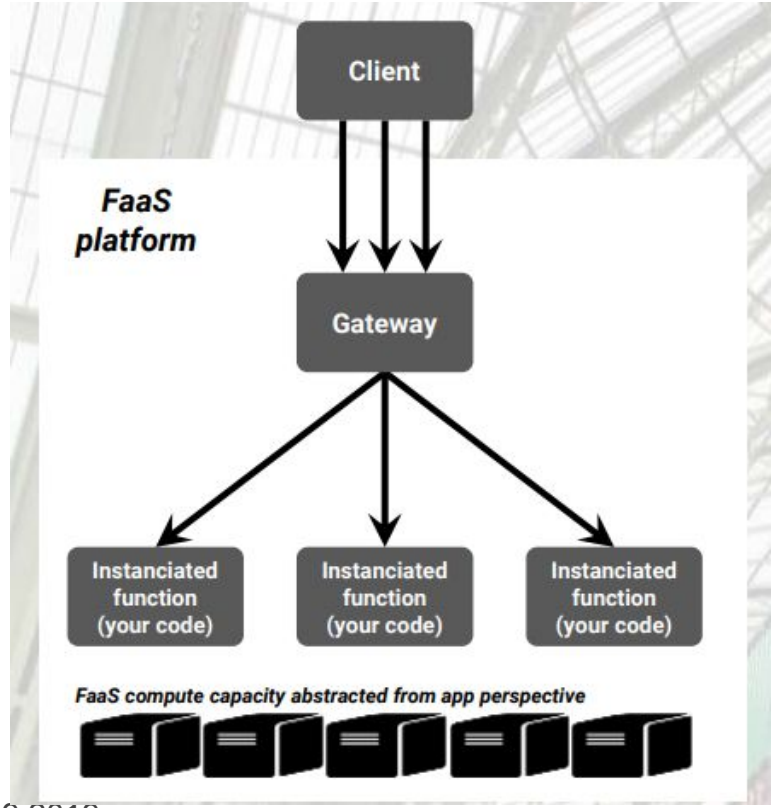
Triggers and bindings let you avoid hardcoding access to other services. Your function receives data (for example, the content of a queue message) in function parameters. You send data (for example, to create a queue message) by using the return value of the function.

Computing infrastructure perspectives

❏ **Ephemeral**: platform waits requests and triggers function on demand, which “lives” the time to deliver the result: your code is not always-on, waiting calls!

❏ **Dynamic scalability & resilience provided** by the platform: more calls, more instances

❏ if you are on public cloud: **Extremely fine grained pay-per-use**… per-call costs



Finally about operations perspectives

Traditionally (**DevOps**):

- **The development team tests their program** in an isolated development environment for quality assurance (QA)
  - if requirements are met
- **The operations team deploys and maintains the program** from that point on
  - Each group assume some of the responsibilities of the other team

It become now intuitive that serverless paradigm is a way to evolve towards NoOps (no operations)

- the concept that an IT environment can become so automated and abstracted from the underlying infrastructure that there is no need for a dedicated team to manage software in-house.

(However, in the real life a completely automated deployment, monitoring and management of applications and the infrastructure on which they run is probably still a dream)

What all of this is good for

good for

short-running
stateless
event-driven

- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Modest Stream Processing
- Service integration

not good for

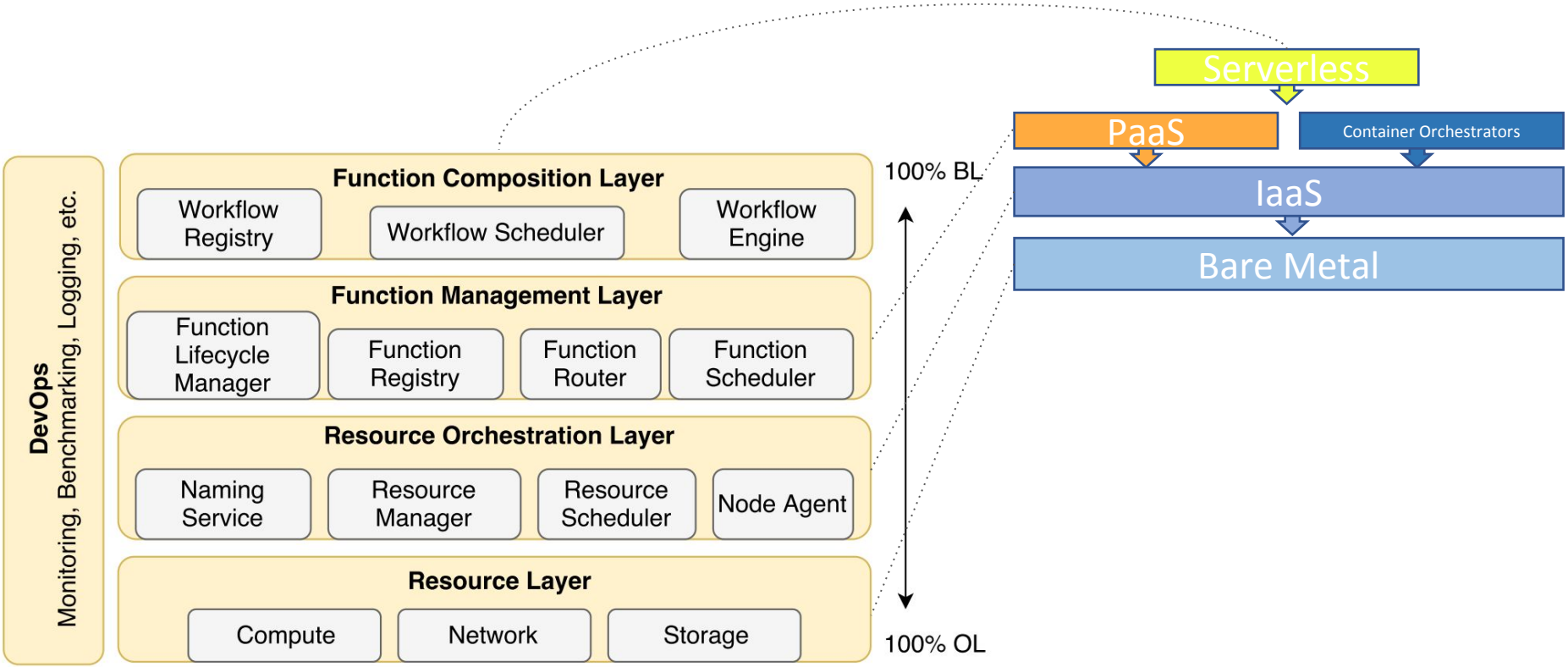
long-running
stateful
number crunching

- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Numerical Simulation
- Video Streaming

Example Use cases

- **Executing logic in response to database changes** (insert, update, trigger, delete).
- **Performing analytics on IoT sensor** input messages, for example, as Message Queuing Telemetry Transport (MQTT) messages.
- **Handling stream processing** (analyzing or modifying data in motion).
- Managing single time extract, transform, and load **jobs that require a great deal of processing for a short time**.
- Providing **cognitive computing** via a chat bot interface (asynchronous, but correlated).
- Scheduling **tasks performed for a short time** (e.g., cron or batch style invocations).
- Serving **machine learning and AI models** (retrieving one or more data elements such as tables or images and matching against a pre-learned data model to identify text, faces, anomalies, etc.).
- Continuous integration pipelines that provision resources for build jobs on-demand, instead of keeping a pool of build slave hosts waiting for jobs to be dispatched.

Putting (almost) everything together



But….

Why are we talking about all this stuff?

SOSC 2019 Intelligent Infrastructures 24

In the end we want to process data...

...Or most probably Big Data to implement Real-time Analytical

Reminder: when we are In the word of BigData:

- we know that we cannot define a fixed number of resources for our platform
- we never know that when the velocity/size of data can change.
- We know that dataset are “never” closed… keep growing cause data sources push data continuously

A BigData computing platform must be able to tackle any these situations:

- Serverless architecture is **A** key element to provide solutions to these problems

Workflow example (serverless stream processing)

Data Sources push data continuously, so we need:

A unified Data Collection Layer is need to write it to Real-time

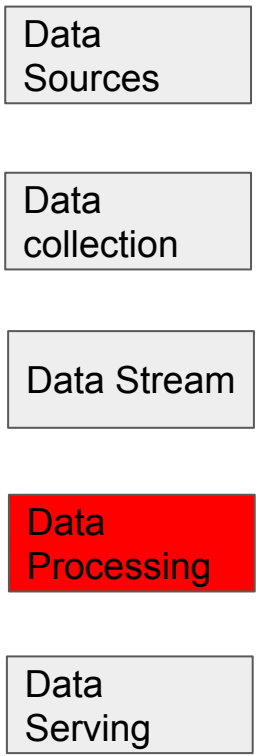
A real-time storage system ables to scale up and down depending on data rate

- To Stream collected data which can be further processed by Data Processing Engines.

A layer where to do some **Data preprocessing** (Data Cleaning, Data Validation, Data Transformations etc).

- This is the layer where we also perform real-time analytics on incoming streaming data by using the window of last 5 or 10 minutes..
- Data Processing Platform need to process any amount of data with consistent throughput

Data Serving Layer to write processed data for further user analysis



Examples

Mobile and Internet-of-Things applications

FaaS is often adopted for Data Processing and Transformation Service in which continuously consuming data from streams and perform the Data Cleaning, Transformations and Enrichment on the data and store

Real-time Log Monitoring & Alerting

Real-time log monitoring can be enabled using FaaS in which one can keep on consuming the log events Then, After doing some parsing of logs, one can monitoring the metrics and check for any critical event and generate alerts to a notification platforms (e.g. Slack, Email, etc.)

Wrapping up:

How to build analytics solutions in **public clouds**

And what about using Open Source...

… and Private/Hybrid Clouds?

As a scientist I might want/need/must build my open source intelligent computing infrastructure, possible customized, to execute my analytics to process my (detector) data…

No worry (at least not too much)

- There are several open source solutions available in the market which can be adopted/customized/deployed in any cloud infrastructure
- And, of course, a key to the success in this respect is to learn how to code your infrastructure reusing available components.
    - Now we are referring to **Infrastructure as a code**… You will see this more this afternoon

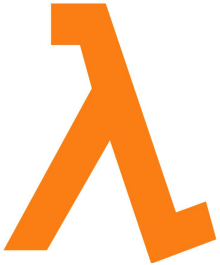
Recap of some key messages in this talk

- We tasted how/why the computing infrastructure has a central role while building solutions for intelligent systems
  - Data handling and management, processing and serving

- We discussed objectives and motivations for abstracting computing infrastructures and knobs provided by emerging technologies
  - Dev: allows to focus on application/business logic
  - Ops: reduce the effort in maintaining complex system

- We discussed FaaS as a cloud computing framework
    - Trying to relate FaaS and Serverless (sometime misnomer)

Some technologies (platforms for serverless)



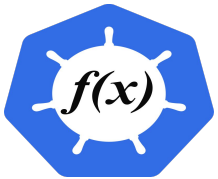
OpenLambda

Azure Functions

AWS
Lambda

Kubernetes

IBM Cloud Functions



Red-Hat

Google Functions

One size fits all solutions?

No, generally speaking there is no a generic solutions which covers all the use cases…

This is true in general and for FaaS frameworks in particular

- Open problems
- Research challenges
- Questions
- …

spiga@infn.it

Challenges ahead of us: in a nutshell

- Can different cloud computing service models be mixed?
- Monitoring and debugging
  - Debugging is much different if instead of having one artifact (a micro-service or traditional monolithic app) developers need to deal with a myriad of smaller pieces of code …
- Can legacy code be made running on serverless?
  - Hybrid model?
  - To what degree existing legacy code can be automatically or semi-automatically decomposed into smaller-granularity pieces to take advantage of these new economics?
- Is serverless fundamentally stateless?
  - Can there be serverless services that have stateful support built-in