Beyond CMS, HTCondor, PaaS... a look to the future challenges

Daniele Spiga

HPC4L - Training. Beirut, Lebanon
21th22th October 2020

Outline

Introduction

DODAS and Big Data:

- Spark use case

Future challenges

- Serverless and Function as a Service

Goal of this last session

1. What we saw these 2 days.
   a. Cloud, Paas, Automation and services orchestration
   b. We saw how DODAS implement the infrastructure as a code paradigm
   c. We saw two main use cases..
      i. HTCodndor batch system
      ii. CMS Site

2. What DODAS can do beyond these use cases
   a. we ‘ll do the spark example

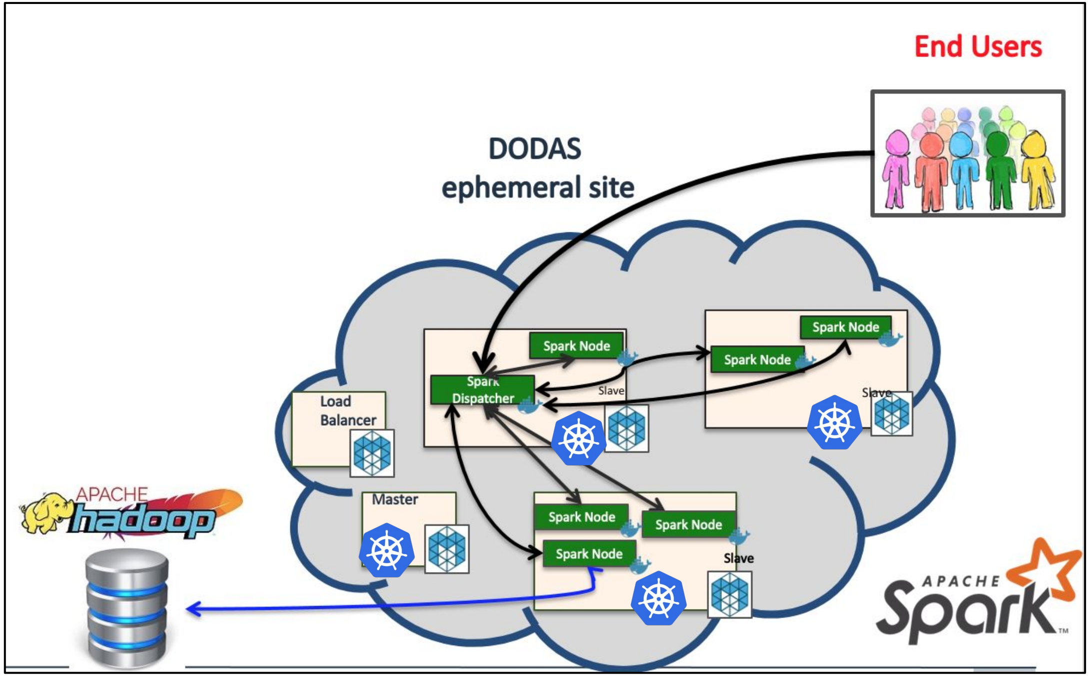
3. And finally what about evolution in cloud ? …
   a. From container / microservices ---> to functions

The Strategy based on a Lego Approach

There is a huge set of tools and solutions available, but there is NOT a one-size-fit-all solution
Open, Standard-based, flexible and extensible building blocks
Each use case can compose and customize
and customize

Let's spoil this talk then

Using DODAS to automatically deploy Spark on a cloud environment

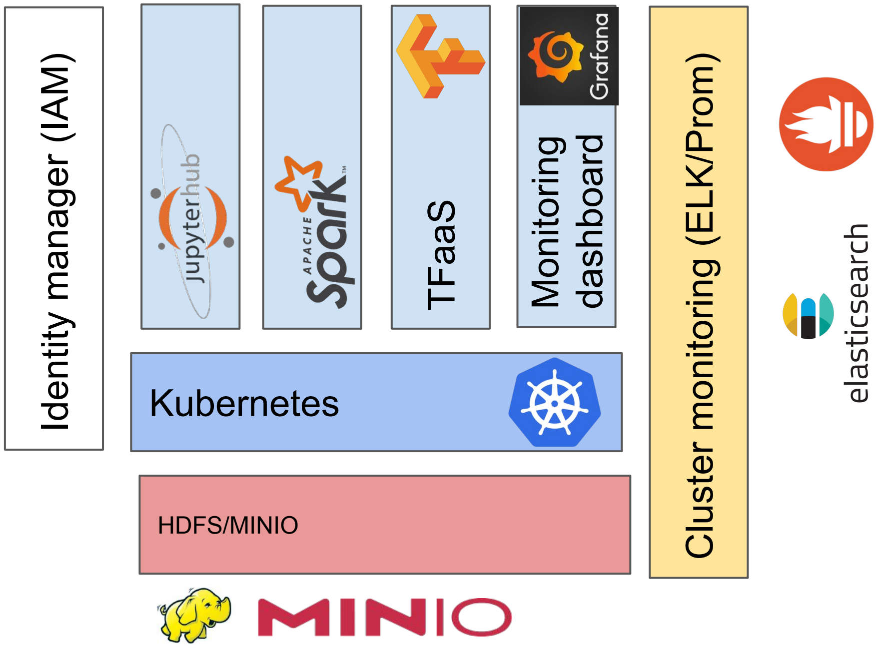


Big data infrastructure

"big data infrastructure entails the tools and agents that collect data, the software systems and physical storage media that store it, the network that transfers it, the application environments that host the analytics tools that analyze it and the backup or archive infrastructure that backs it up after analysis is complete."

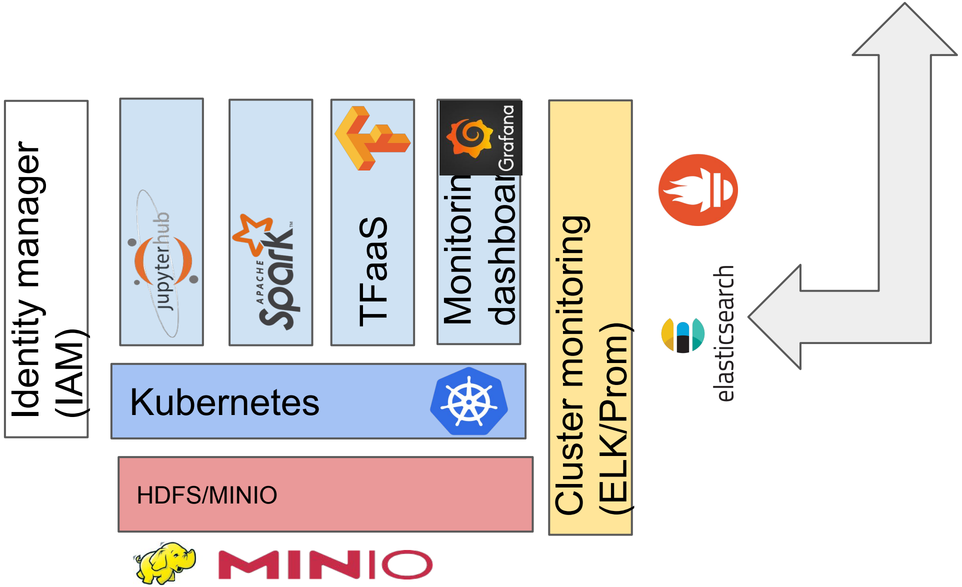
syncsort

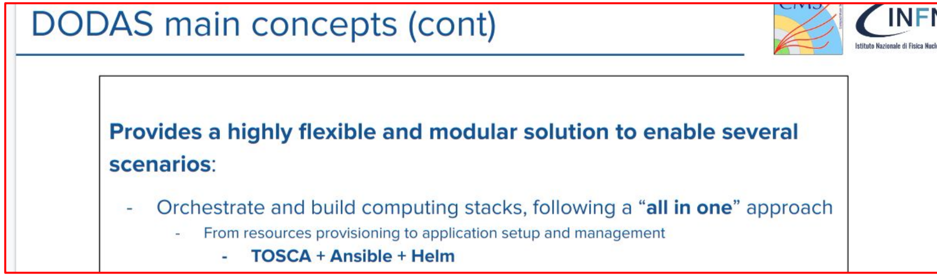
As example: something like this?



Most of what has been discussed this week in term of services and software, components
- plus something I will show in the next…

And how DODAS fits into this ?

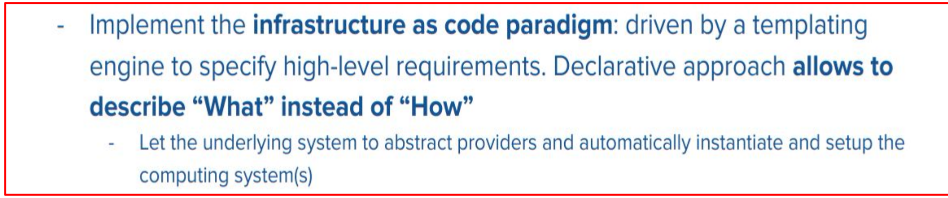




DODAS main concepts (cont)

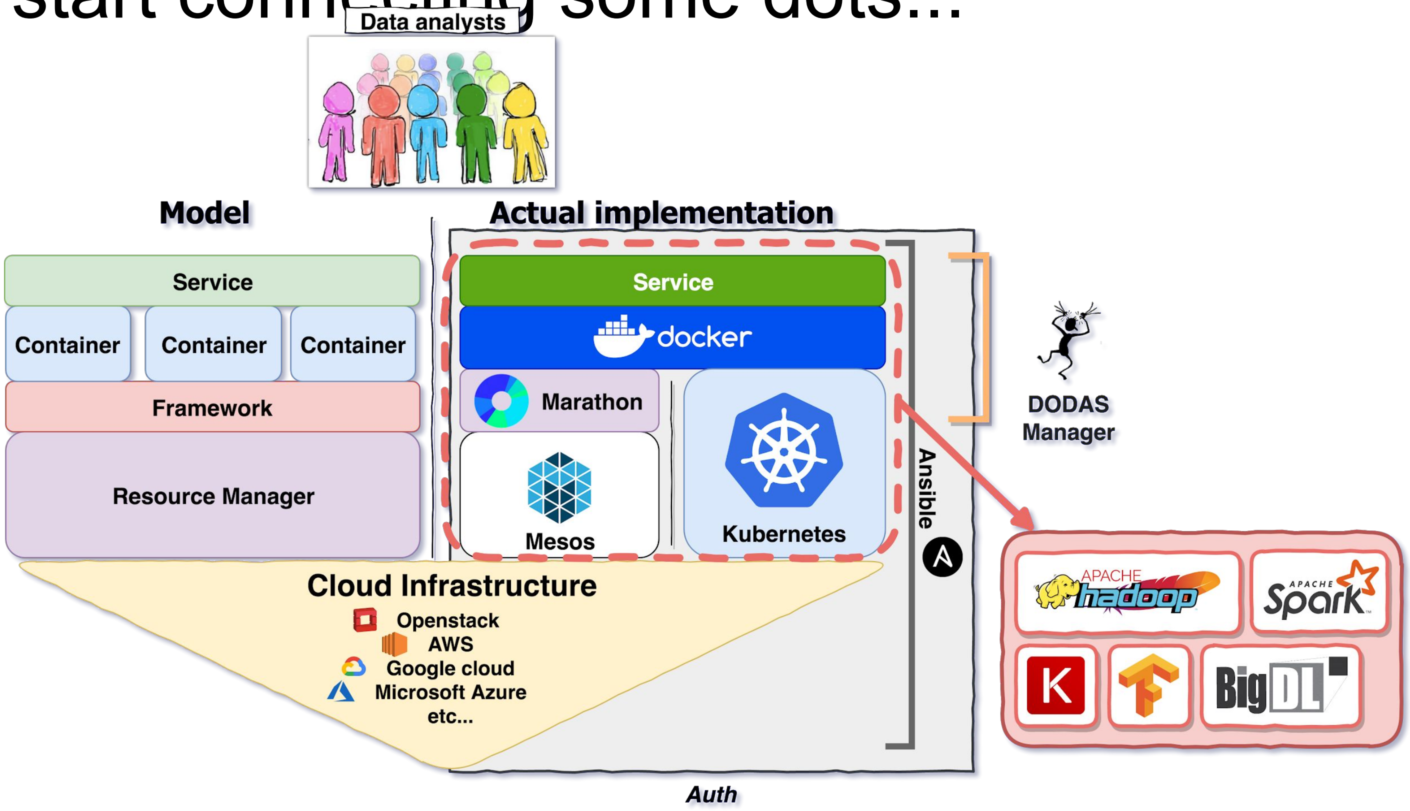
Provides a highly flexible and modular solution to enable several scenarios:

- Orchestrate and build computing stacks, following a "**all in one**" approach
  - From resources provisioning to application setup and management
    - **TOSCA + Ansible + Helm**

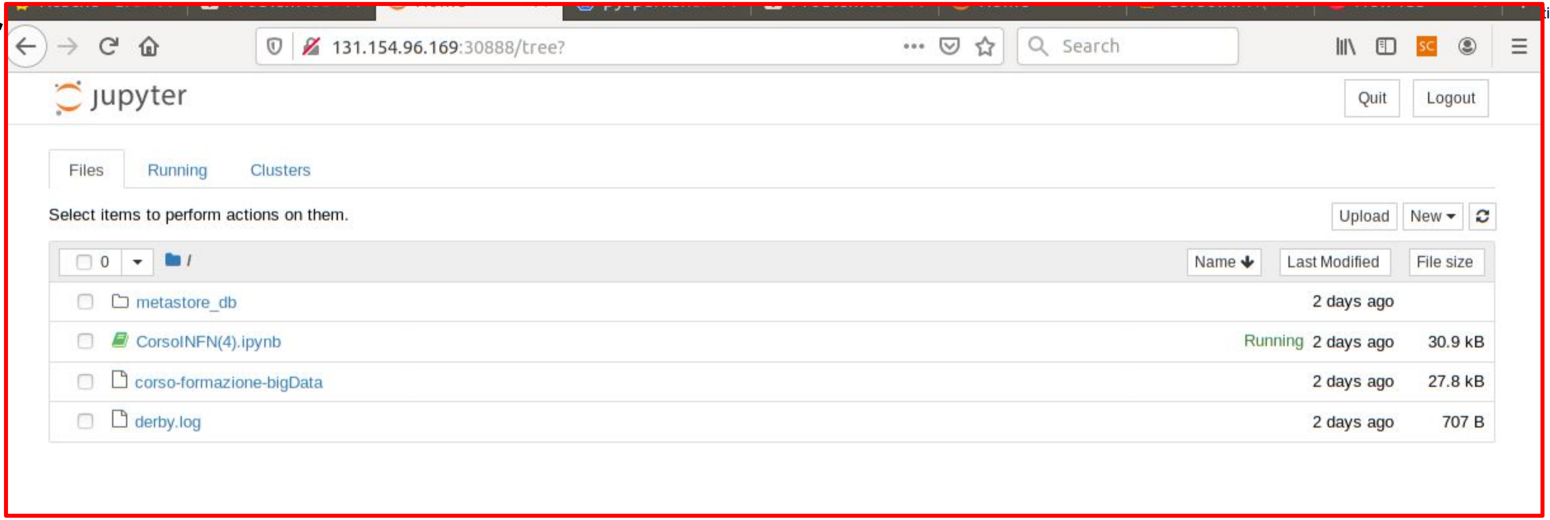


- Implement the **infrastructure as code paradigm**: driven by a templating engine to specify high-level requirements. Declarative approach **allows to describe "What" instead of "How"**
  - Let the underlying system to abstract providers and automatically instantiate and setup the computing system(s)

Let's start connecting some dots...



Fr



Accessed by

Runs on

TOSCA

```yaml
inputs:

  number_of_masters:
    type: integer
    default: 1

  num_cpus_master:
    type: integer
    default: 2

  mem_size_master:
    type: string
    default: "4 GB"

  number_of_slaves:
    type: integer
    default: 1

  num_cpus_slave:
    type: integer
    default: 2

  mem_size_slave:
    type: string
    default: "4 GB"

  server_image_slave:
```

```yaml
    k8s_master:
      type: tosca.nodes.indigo.LRMS.FrontEnd.Kubernetes
      properties:
        admin_token: testme
        kube_version: 1.14.0
        kube_front_end_ip: { get_attribute: [ k8s_master_server, private_address, 0 ] }
      requirements:
        - host: k8s_master_server
```

```yaml
    k8s_wn:
      type: tosca.nodes.indigo.LRMS.WorkerNode.Kubernetes
      properties:
        front_end_ip: { get_attribute: [ k8s_master_server, private_address, 0 ] }
        kube_version: 1.14.0
        nfs_master_ip: { get_attribute: [ k8s_master_server, private_address, 0 ] }
      requirements:
        - host: k8s_slave_server
```

TOSCA (cont)

```
k8s_master_server:
  type: tosca.nodes.indigo.Compute
  capabilities:
    endpoint:
      properties:
        network_name: PUBLIC
        ports:
          kube_port:
            protocol: tcp
            source: 6443
          dashboard_port:
            protocol: tcp
            source: 30443
          web_ui:
            protocol: tcp
            source: 30808
          jupyter:
            protocol: tcp
            source: 30888
    scalable:
      properties:
        count: { get_input: number_of_masters }
    host:
      properties:
        num_cpus: { get_input: num_cpus_master }
        mem_size: { get_input: mem_size_master }
    os:
      properties:
        image: { get_input: server_image }
```

```
    k8s_slave_server:
      type: tosca.nodes.indigo.Compute
      capabilities:
        endpoint:
          properties:
            network_name: PRIVATE
        scalable:
          properties:
            count: { get_input: number_of_slaves }
        host:
          properties:
            num_cpus: { get_input: num_cpus_slave }
            mem_size: { get_input: mem_size_slave }
        os:
          properties:
            image: { get_input: server_image_slave }

```

TOSCA but finally Spark

```
    type: tosca.nodes.indigo.HelmInstall
    properties:
      externalIP: { get_attribute: [ k8s_master_server, public_address,
      name: "spark"
      chart: "cloudpg/spark"
      repos:
          - { name: cloudpg, url: "https://cloud-pg.github.io/charts/" ]
      values_file: { get_input: helm_values }
    requirements:
      - host: k8s_master
      - dependency: k8s_wn
```

Compiling values at runtime and install

Who is doing this?

That’s the last step..
Installing Spark on top of k8s

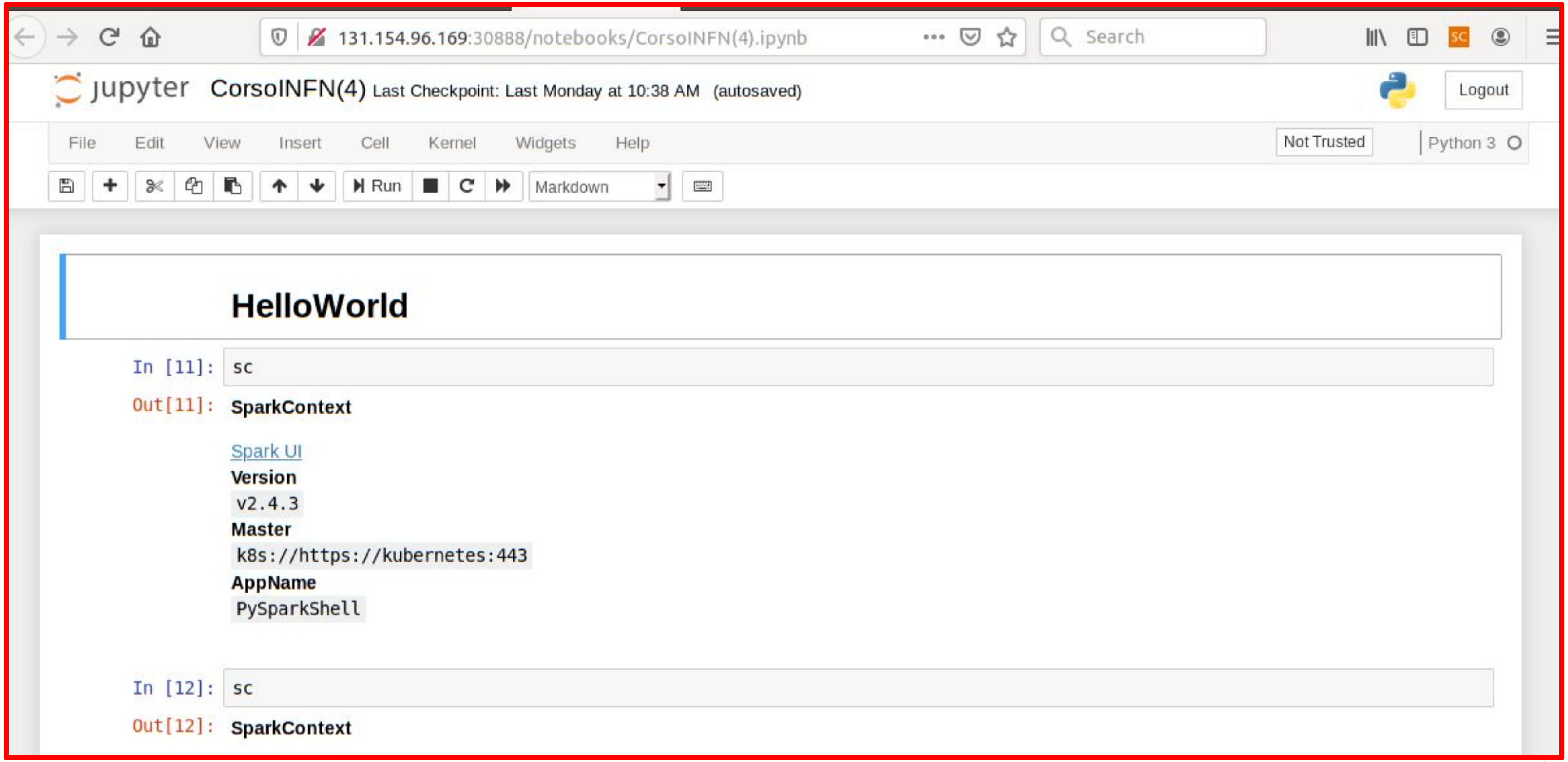
GitHub, Inc. [US] | https://github.com/Cloud-PG/ansible-role-helm/blob/master/tasks/kube.yml

23 lines (18 sloc) | 697 Bytes

Raw | Blame | History

```yaml
---
- name: Helm install cloudpg repo
  command: helm repo add {{ item.name }} {{ item.url }}
  with_items: "{{ repos }}"

# - name: Helm install cloudpg repo
#   command: helm repo add cloudpg https://cloud-pg.github.io/charts/

# - name: Helm install cache repos
#   command: helm repo add cache https://cloud-pg.github.io/CachingOnDemand/

- name: write values
  get_url:
    url: "{{ values_file }}"
    dest:  /tmp/values_{{ name }}-template.yml

- name: compile values
  template:
    src:  /tmp/values_{{ name }}-template.yml
    dest: /tmp/values_{{ name }}.yml

- name: Helm install chart {{ chart }}
  command: "helm install --name {{ name }} -f /tmp/values_{{ name }}.yml {{ chart }}"
```

And the result



Cloud: quick reminder

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

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

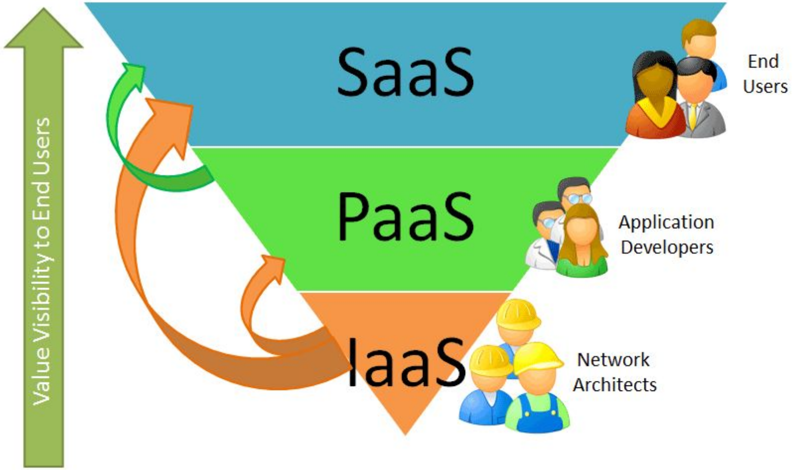
- **IaaS**, the basic building blocks of a data center:
  - **Storage** I want to store data, lots of data, at low cost
  - **Compute** Give me a machine where I can host my services or run my applications
  - **Network** Create a "Software-Defined Network" infrastructure for me

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

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

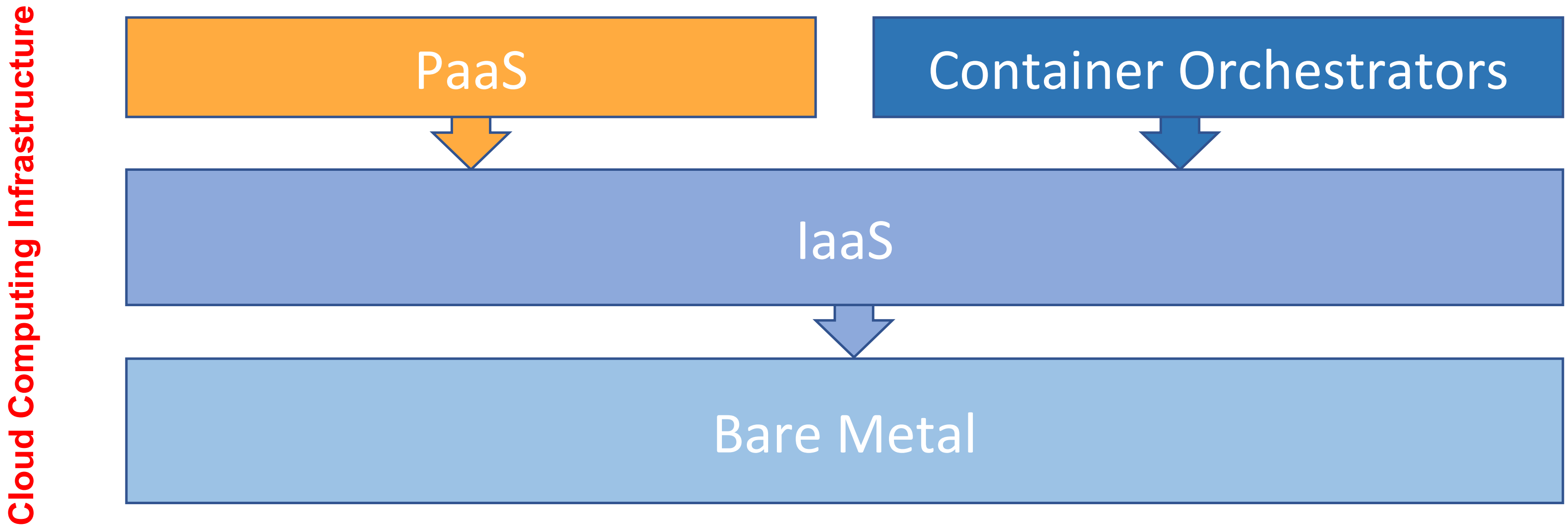
"Wouldn't it be nice if one could **pack the application**, **with all its dependencies**, into a dedicated box and **run it anywhere**? No matter what software dependencies the host system has installed, or where and what the host system actually is?"

- **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…



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**

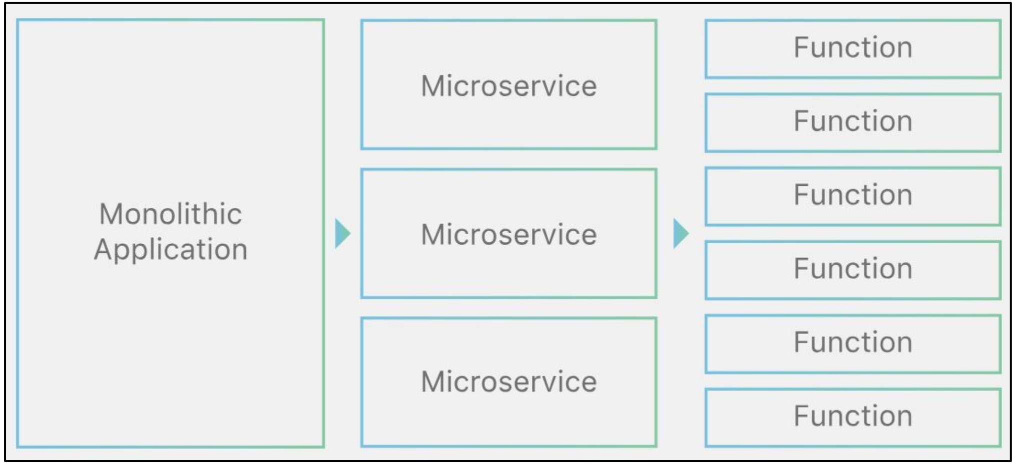
- So the suggestion might be to make them **stateless and smaller**

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

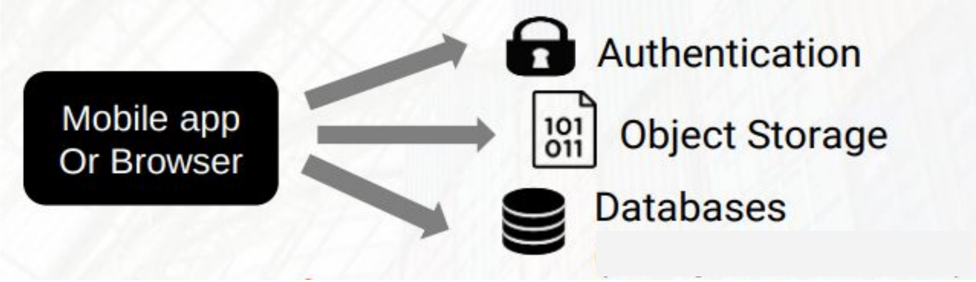
Just code to develop and execute

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

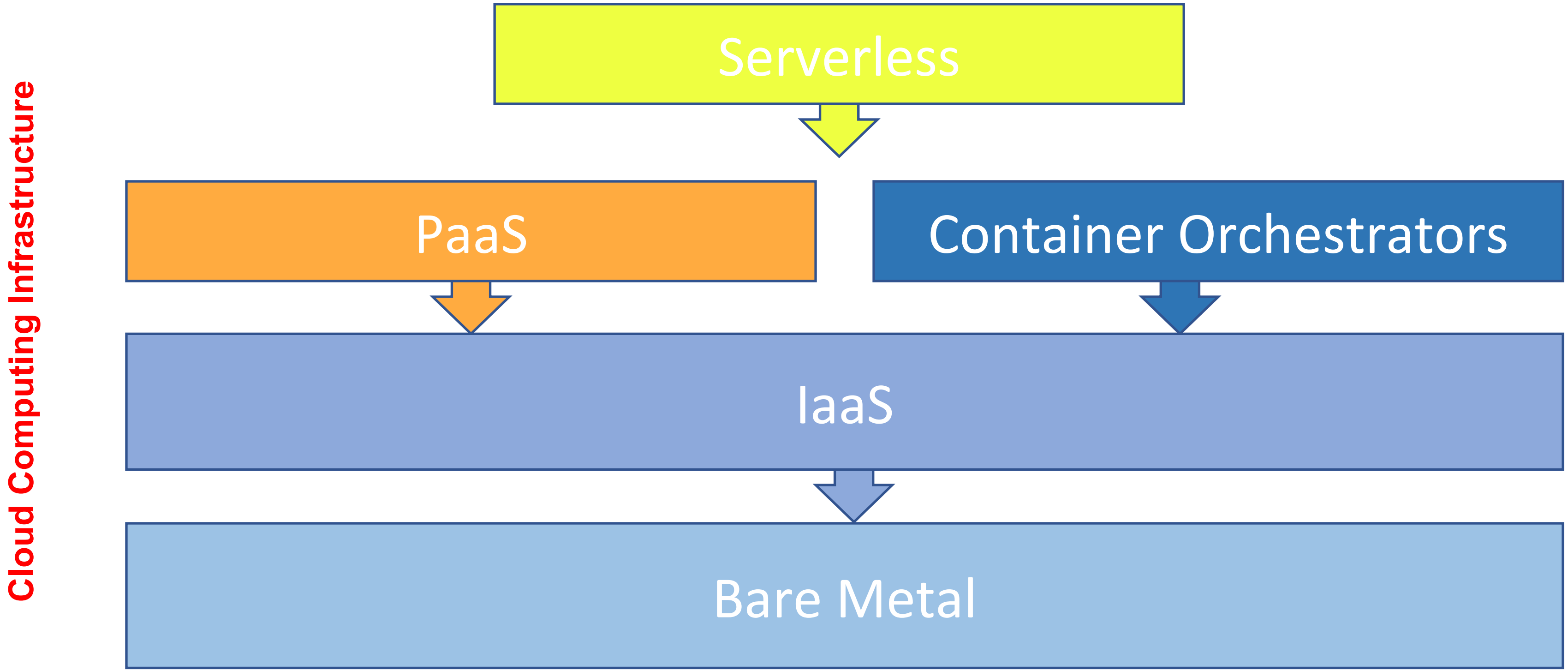
which

scales up and down instantly and automatically

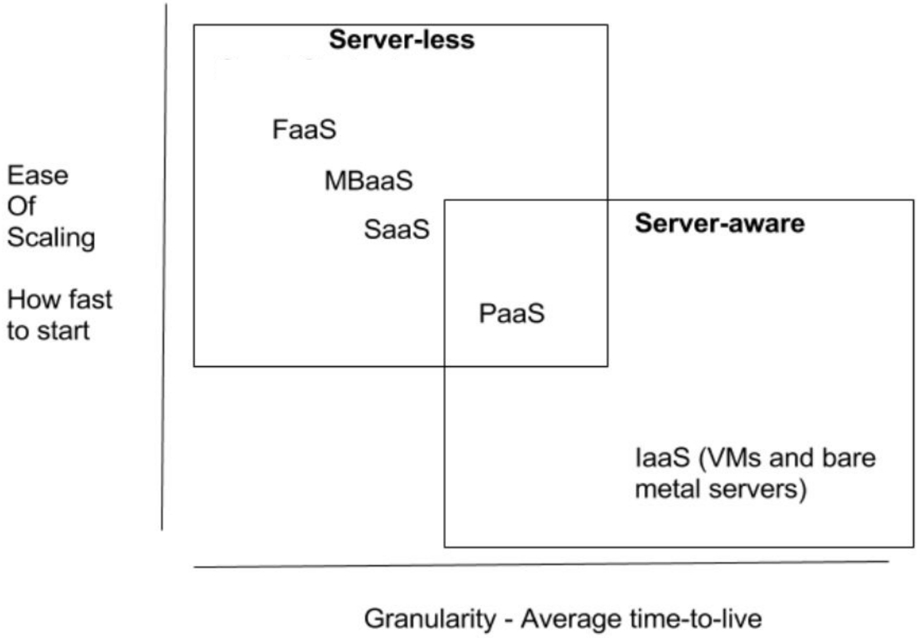
and

(charges for actual usage at a millisecond granularity)

Where this positions ?



Cloud computing: 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)

Why serverless ?

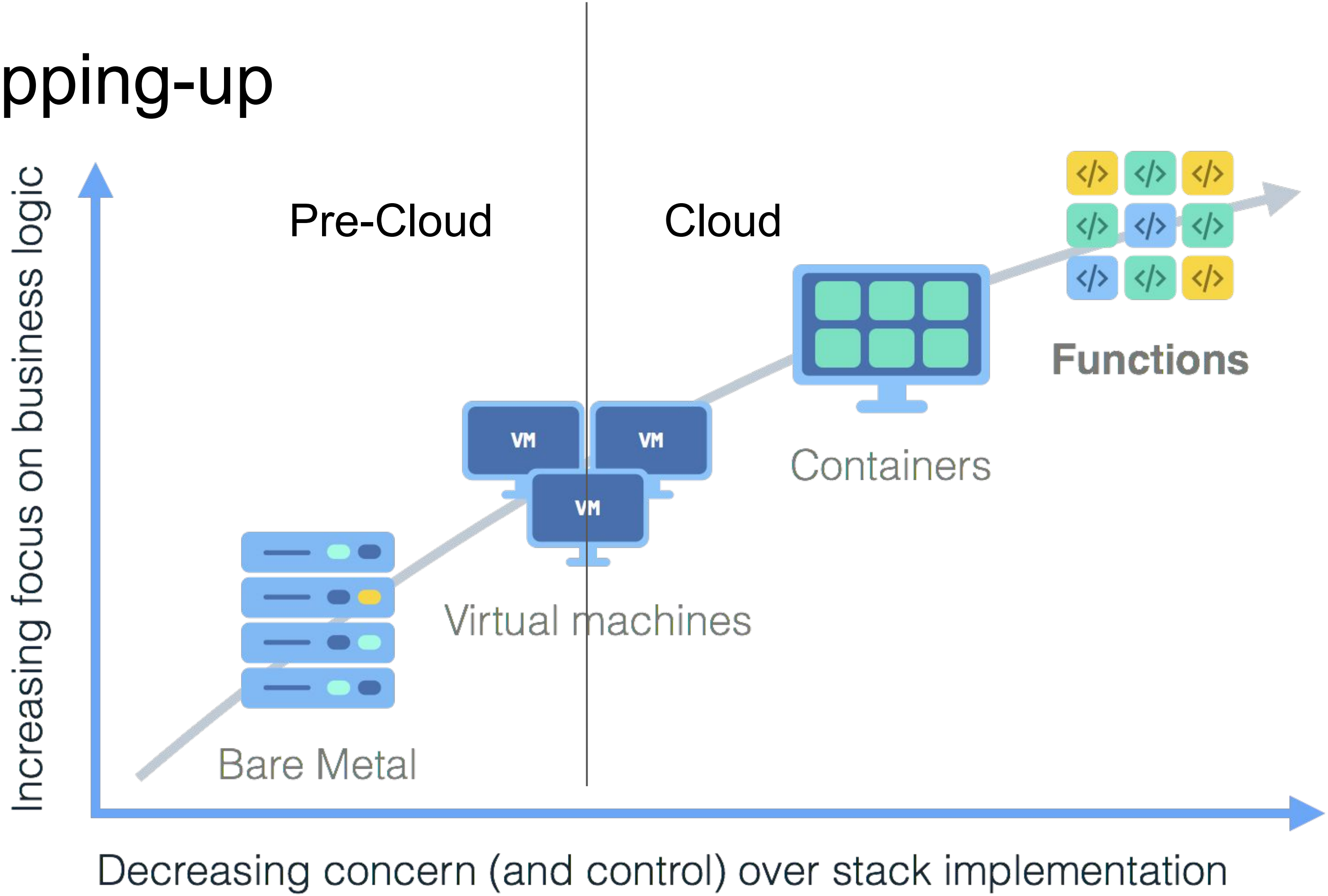
With serverless technologies, we perform another step toward automating and facilitating the use of Cloud resources.

Remember the inverted triangle we saw: what eventually matters are the applications, not the infrastructure.

Recall what happens with traditional Cloud applications, of which we have already seen several examples:

- We need to provision and manage the resources (e.g. VM1, VM2, the disks, the S3 buckets, etc.) for our applications.
- We are charged if we keep the resources up, even if they are doing nothing
- We are responsible to apply all the updates and security patches to our servers

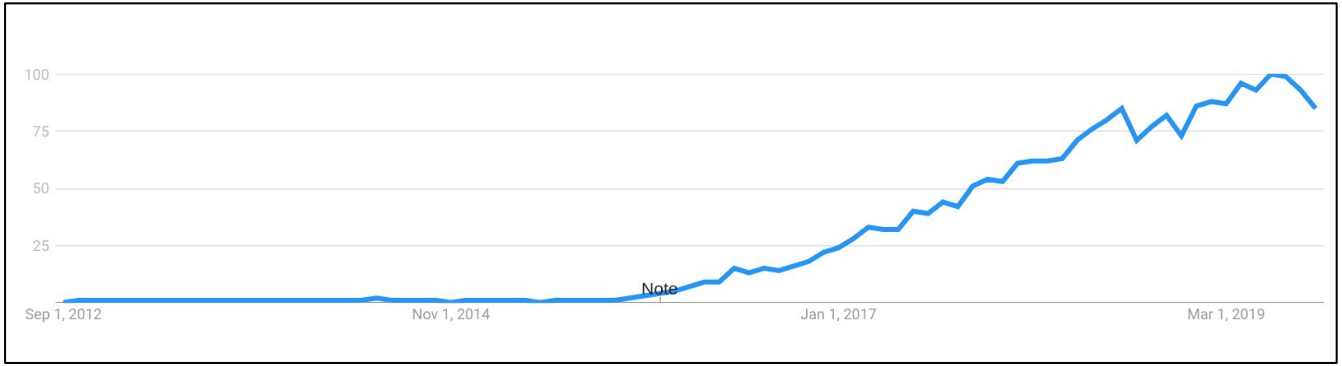
Wrapping-up



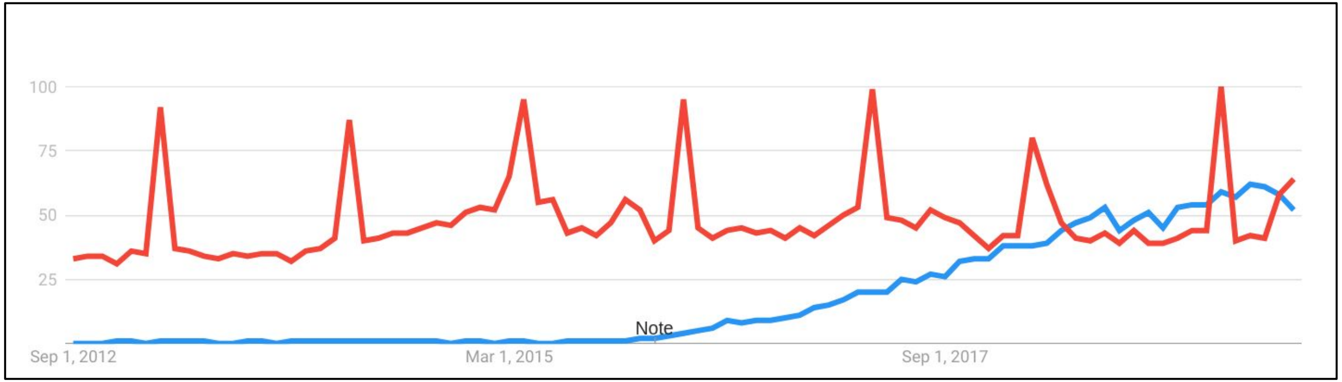
Pre-Cloud
Cloud
Increasing focus on business logic
Bare Metal
VM
VM
VM
Virtual machines
Containers
</>
Functions
Decreasing concern (and control) over stack implementation

Gaining attention…

Google Trends



FaaS



FaaS

vs

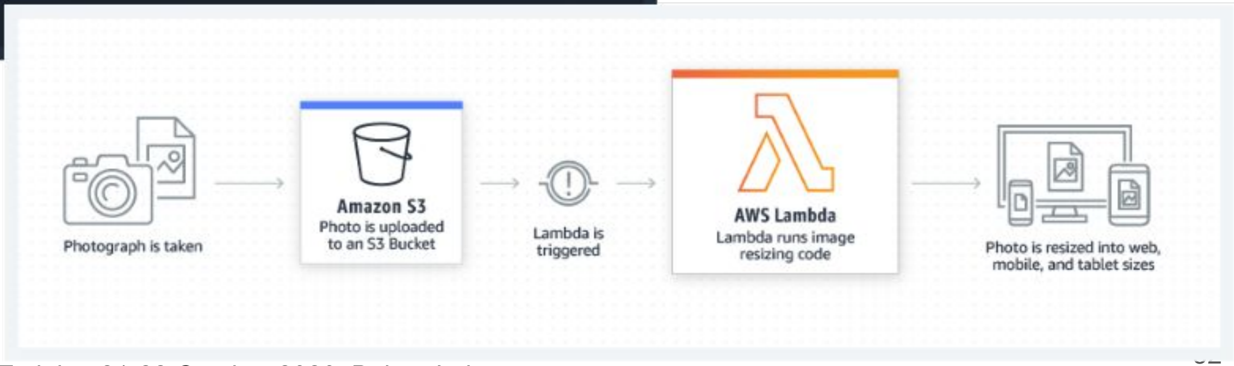
PaaS

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

Some technologies (platforms for serverless)

APACHE OpenWhisk

AWS Lambda

OpenLambda

Iron.io

Azure Functions

fission

Kubernetes

IBM Cloud Functions

Red-Hat

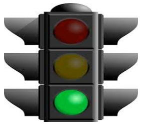
Google Functions

fn

What all of this is good for

good for

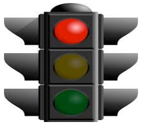
short-running
stateless
event-driven



- 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

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
- …

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