CloudButton

Serverless FaaS

Pedro Garcia Lopez

Table of Contents

CLOUDLAB

Who am I

UNIVERSITAT ROVIRA i VIRGILI

- Associate Professor at Universitat Rovira i Virgili
- Head of the Cloud and Distributed Systems Lab (2003-2019)
- My research interests and experience: Distributed Systems, Middleware, Storage systems, Cloud Computing, Data Analytics
- Principal Investigator of large research projects (>10)
- 20 years teaching distributed systems and Advanced Programming Techniques

CLOUDLAB

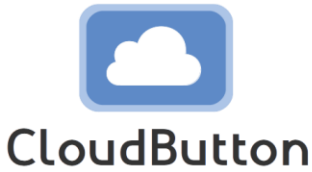
CloudLab: Cloud and Distributed Systems Research group

- **Professors**: Pedro Garcia Lopez, Marc Sanchez Artigas, Carlos Molina Clemente, Carlos Aliagas Castells
- **Postdocs**: Josep Sampé Domenech
- **Senior Research Engineer**: Gerard Paris
- **PhDs**: Daniel Barcelona, Ammar Okran, Alvaro Ruiz
- **Master Students**: Amanda Gómez

- Projects:
  - H2020 CloudButton,
  - Software Defined Edge Clouds

CLOUDLAB

Serverless Data Analytics



- 4.4M€ European research project
- cloudbutton.eu
- Coordinated by URV
- 2019-2021

Why Serverless

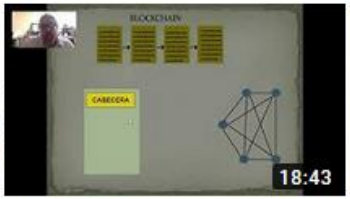
WHY SERVERLESS

Blockchains or Cloud ?

- **When P2P declined this was an important question**
- **BitCoin**: From 0 to 20K$ to 3K$
- Ethereum, ICOs, Smart Contracts
- Bram Cohen sold Bittorrent.com to Tron for 140M$

primeras compras de criptomonedas

198 visualizaciones • Hace 3 meses



QUE ES Y COMO FUNCIONA BLOCKCHAIN

320 visualizaciones • Hace 3 meses

AL PRINCIPIO ERA LA NADA

384 visualizaciones • Hace 4 meses

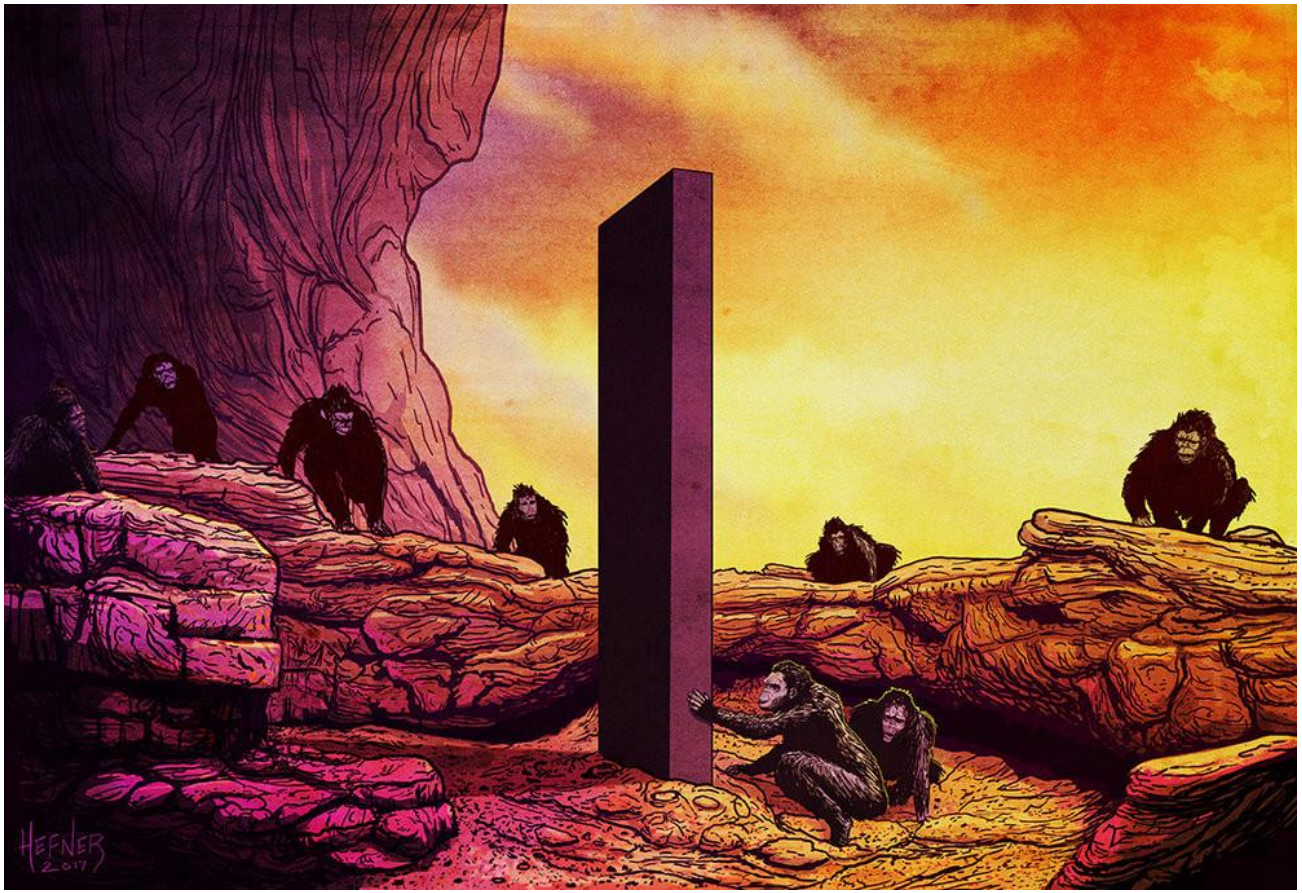
SEGUNDA PRESENTACION

637 visualizaciones • Hace 4 meses

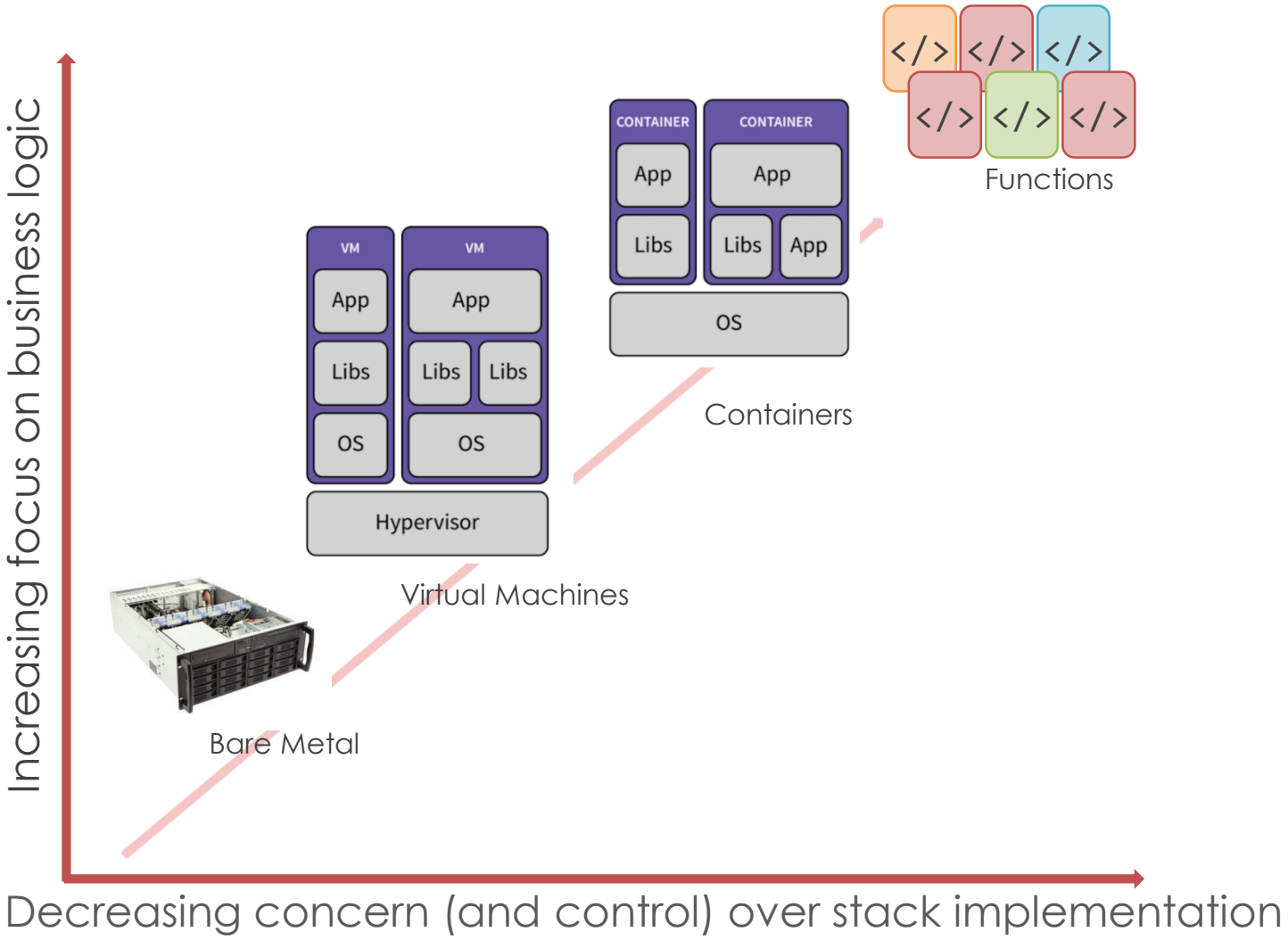
MUNDO FINANCIERO Y CRIPTOMONEDAS

265 visualizaciones • Hace 4 meses

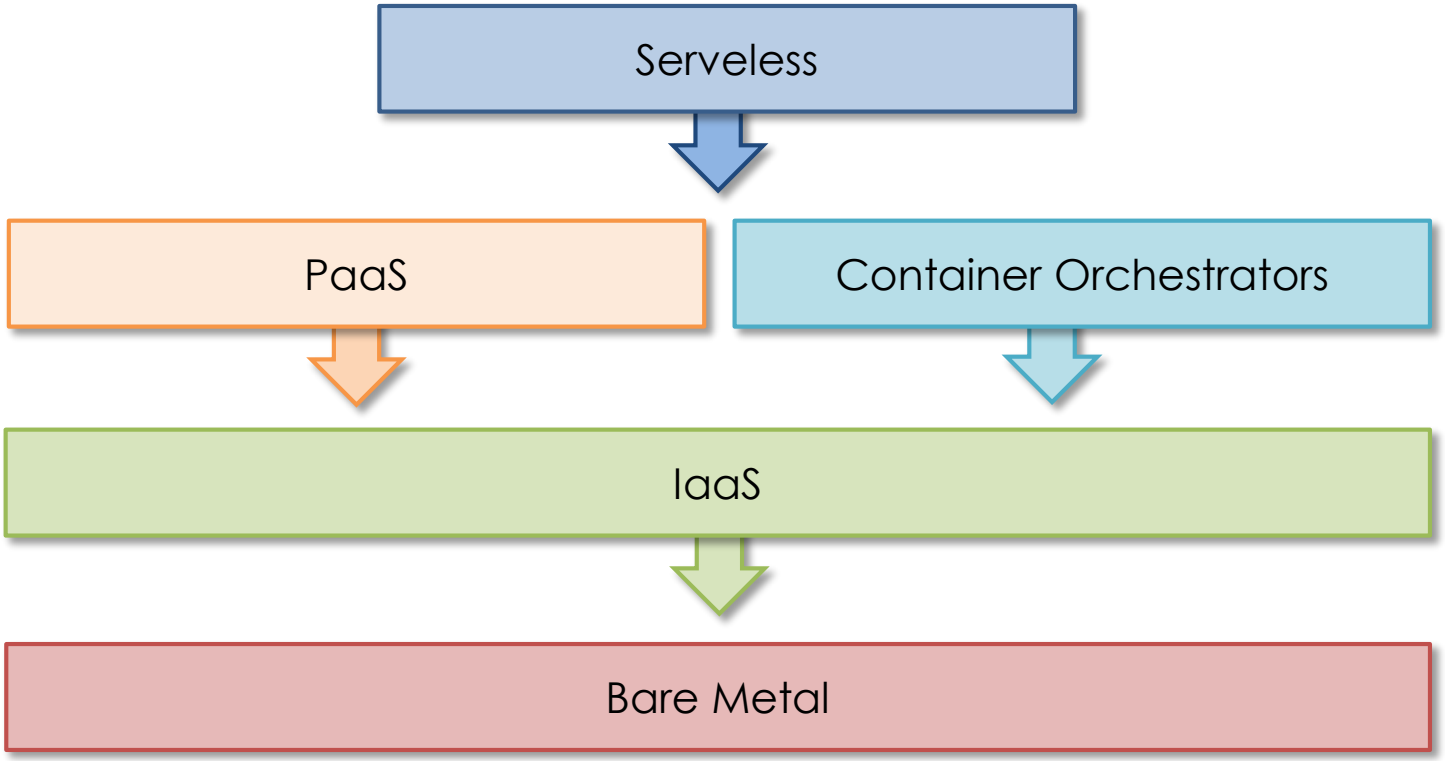
The Monolith



Cloud Computing Evolution



Enter Serverless

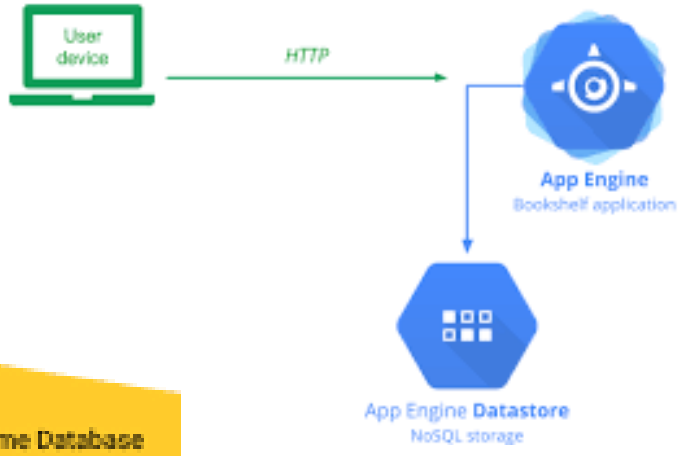


Source: Paul Castro, Serverless Computing Tutorial for WoSC2.

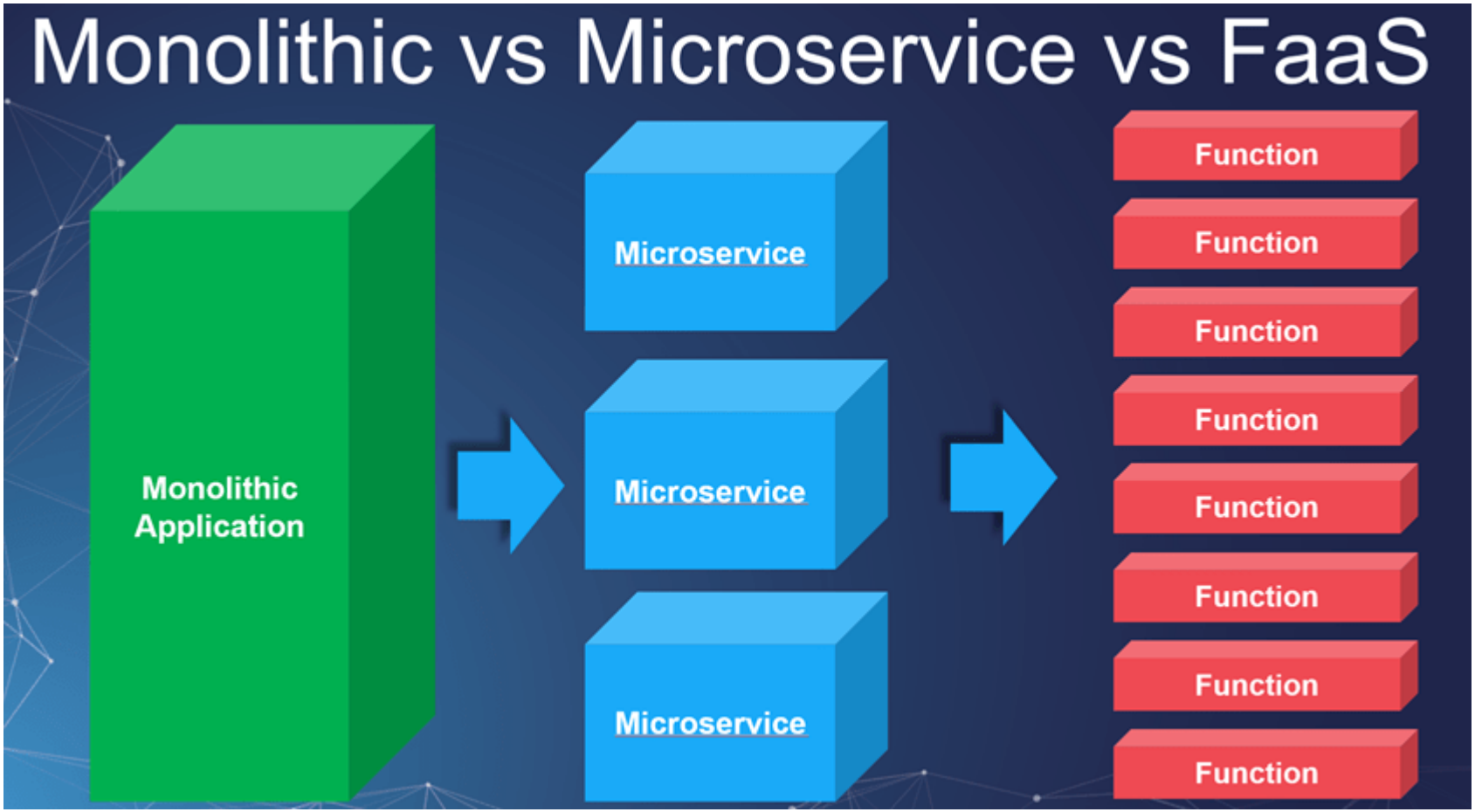
Serverless origins

Parse

- Google App Engine PaaS vs Amazon IaaS
- **Parse.com**
- **Backend as a Service**
  - **Firebase**



Cloud Computing Evolution



SERVERLESS COMPUTING

What is Serverless?

As FAAS (Function-as-a-Service):

a cloud-native platform
FOR
short-running, stateless computation
AND
event- and (data-driven) applications
WHICH
scales up and down instantly and automatically
AND
charges for actual usage at a millisecond granularity

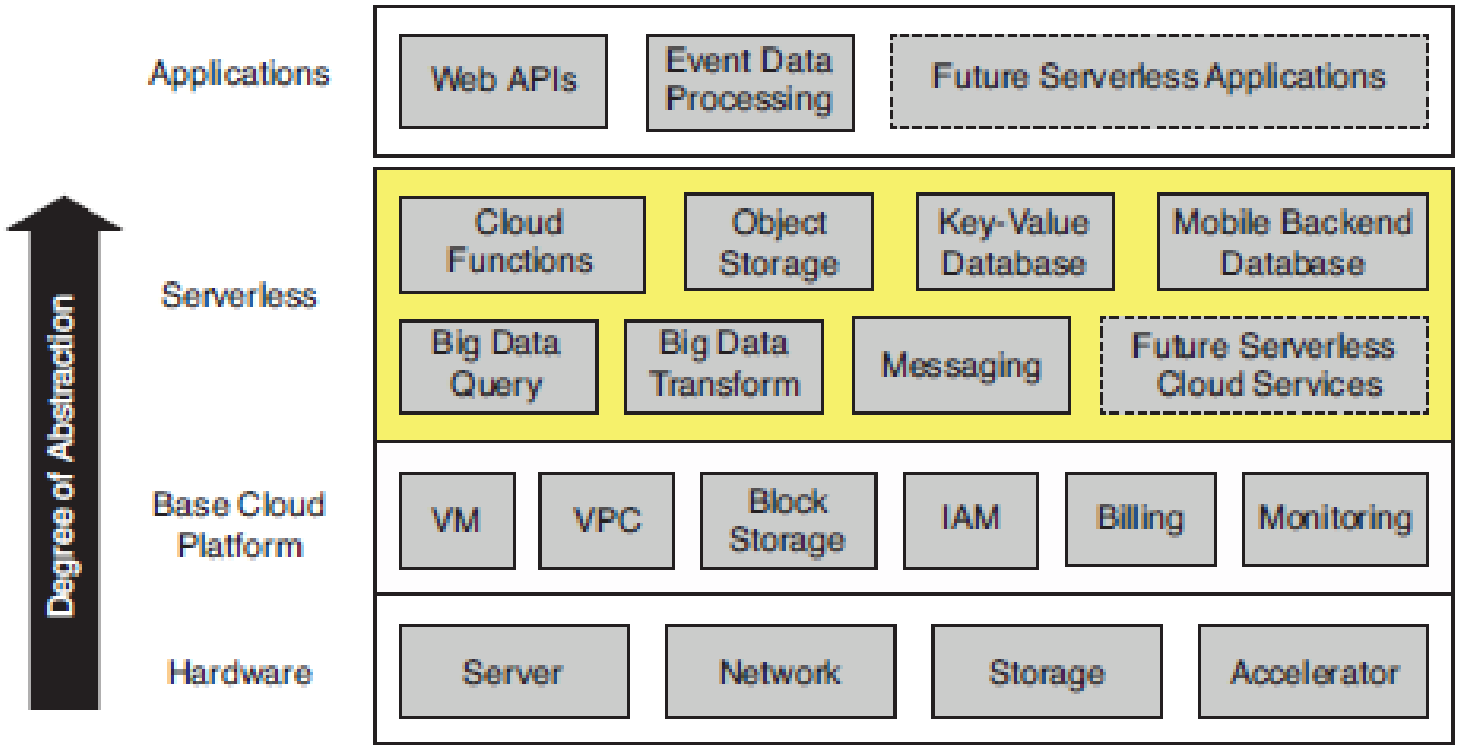
Source: Paul Castro, Serverless Computing Tutorial for WoSC2.

What is Serverless?

	Characteristic	*AWS Serverless Cloud*	*AWS Serverful Cloud*
PROGRAMMER	When the program is run	On event selected by Cloud user	Continuously until explicitly stopped
	Programming Language	JavaScript, Python, Java, Go, C#, etc.[4]	Any
	Program State	Kept in storage (stateless)	Anywhere (stateful or stateless)
	Maximum Memory Size	0.125 - 3 GiB (Cloud user selects)	0.5 - 1952 GiB (Cloud user selects)
	Maximum Local Storage	0.5 GiB	0 - 3600 GiB (Cloud user selects)
	Maximum Run Time	900 seconds	None
	Minimum Accounting Unit	0.1 seconds	60 seconds
	Price per Accounting Unit	$0.000002 (assuming 0.125 GiB)	$0.0000867 - $0.4080000
	Operating System & Libraries	Cloud provider selects[5]	Cloud user selects
SYSADMIN	Server Instance	Cloud provider selects	Cloud user selects
	Scaling[6]	Cloud provider responsible	Cloud user responsible
	Deployment	Cloud provider responsible	Cloud user responsible
	Fault Tolerance	Cloud provider responsible	Cloud user responsible
	Monitoring	Cloud provider responsible	Cloud user responsible
	Logging	Cloud provider responsible	Cloud user responsible

Source: Cloud Programming Simplified: A Berkeley view on Serverless Computing

What is Serverless?



Source: Cloud Programming Simplified: A Berkeley view on Serverless Computing

What is Serverless?

- Function ("Action")
  - Containerized custom-written application code
  - Should include bundled dependencies & binaries
  - Memory & execution time limits
- Triggers ("Events")
  - Causes function execution
  - Can be another function
  - Examples:
    - Upload of a video or image
    - Git commit to a repository
    - ...
- Resources
  - External BaaS/PaaS/FaaS services (object storage, queueing, elastic cache, etc.)

What is Serverless?

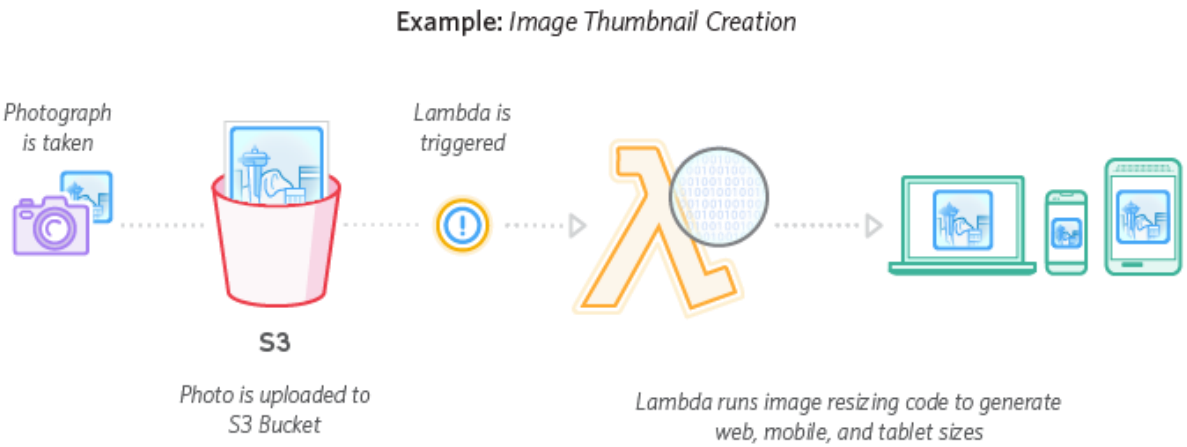
- Function example
  (pandas and numpy are dependencies)

```
import pandas as pd
import numpy as np

def main(args):
    dates = pd.date_range('20130101', periods=2)
    df = pd.DataFrame(np.random.randn(2,2), index=dates,
columns=list('AB'))
    print(df)
    return df.to_dict('split')
```

```
In  [12]: df
Out[12]:
                   A        B
2013-01-01  0.468173  0.64710
2013-01-02 -0.297858 -0.07476
```

Serverless Pattern

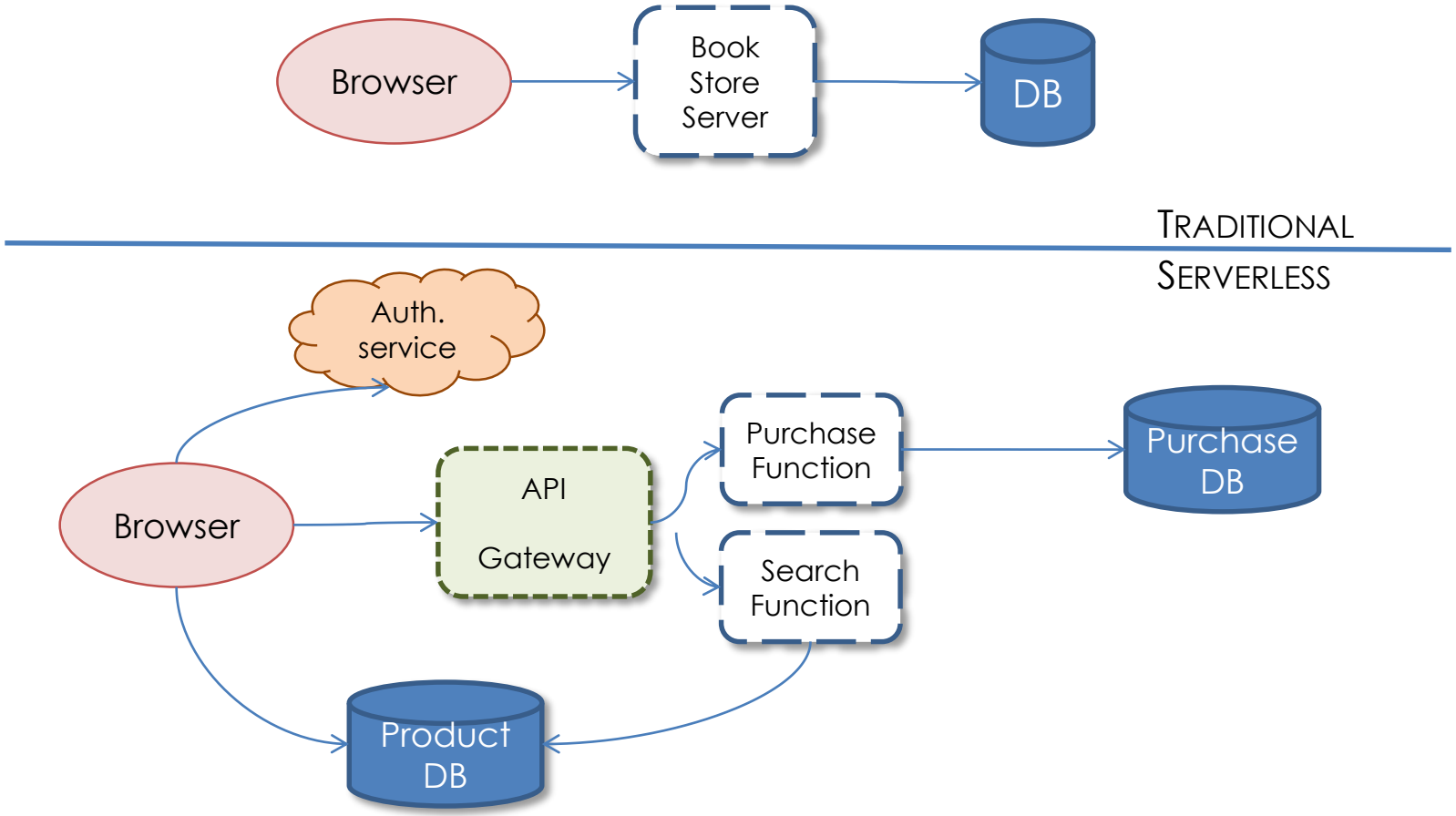
- An application is architected as a set of business logic functions, triggered by discrete events or requests



- Good for microservices, IoT, modest stream processing, ML inferencing, etc.

Serverless Pattern

- Another good example is a typical e-commerce app



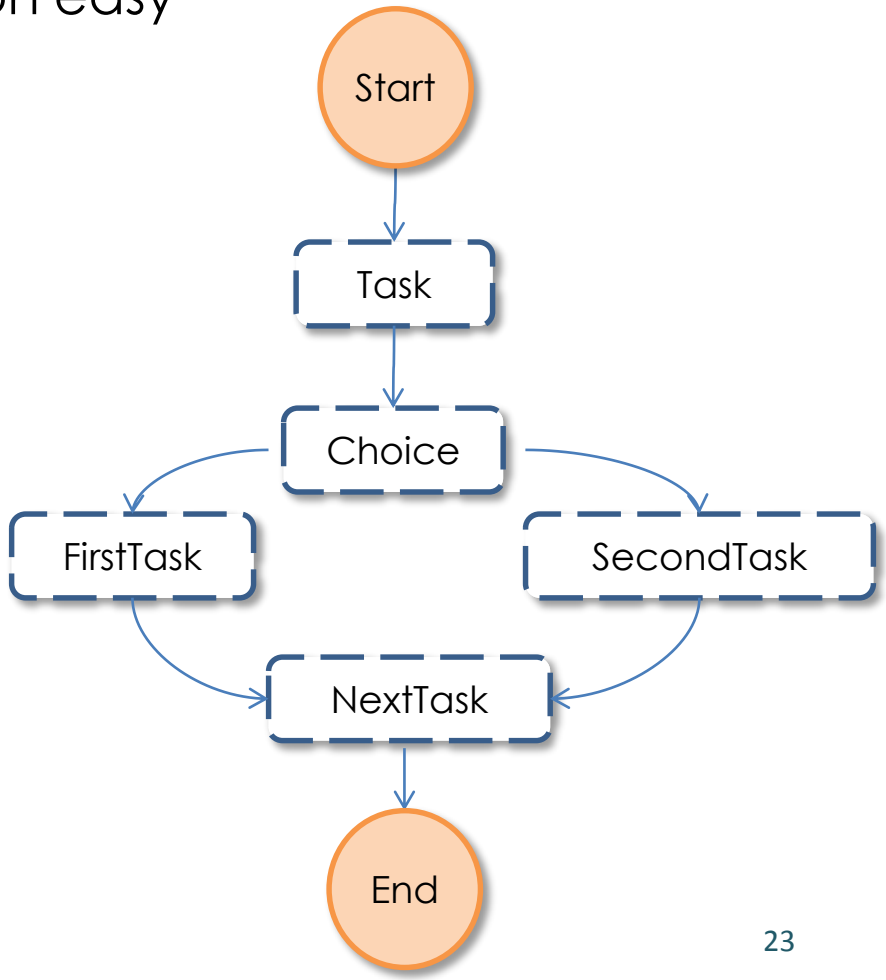
Example from https://martinfowler.com/articles/serverless.html

Serverless Pattern

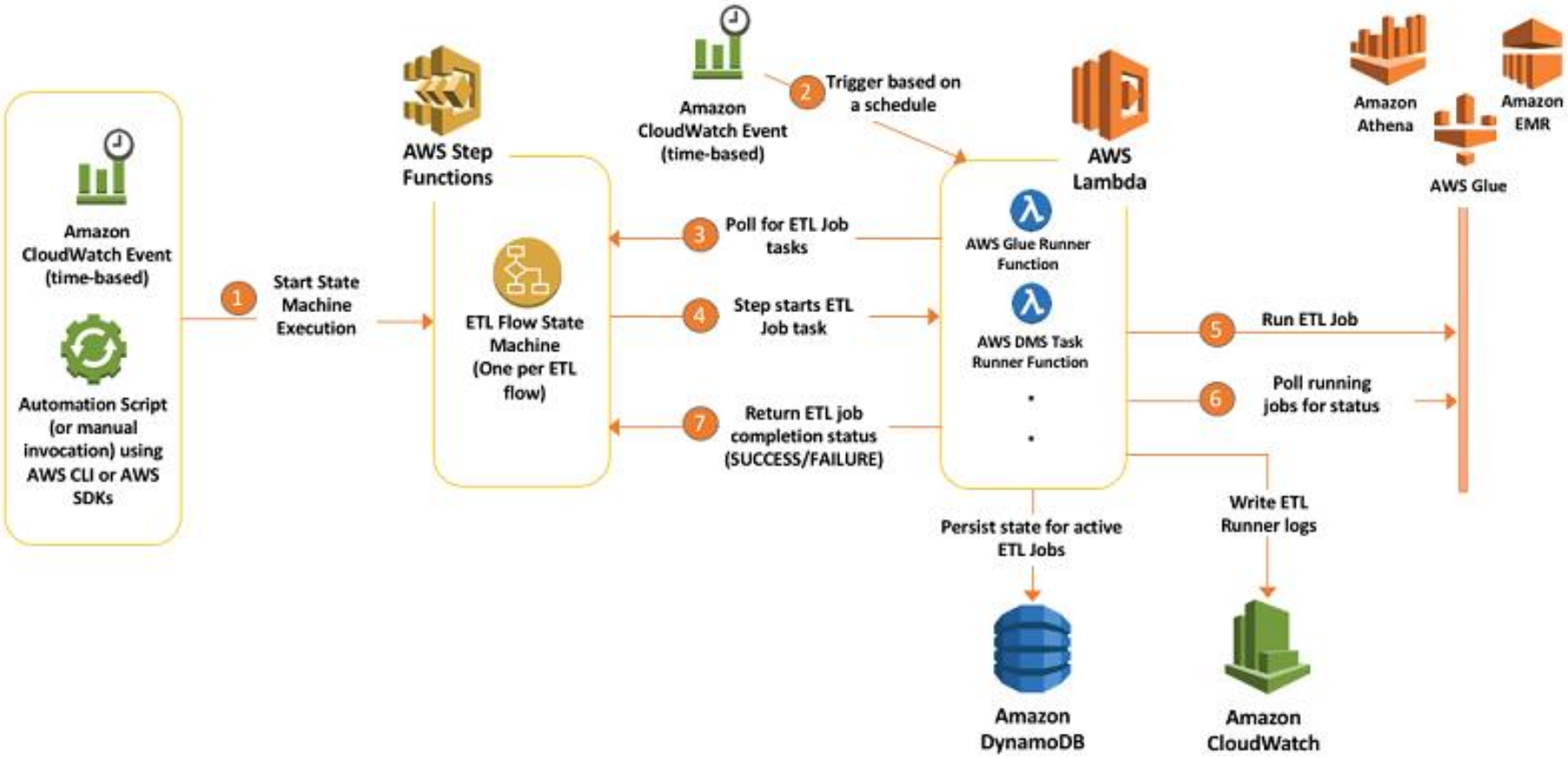
- Also, there is tools such as AWS Step Functions that make function and workflow orchestration easy

- State Example: Choice

```
"Choice": {
    "Type": "Choice",
    "Choices": [
        {
            "Variable": "$.foo",
            "NumericEquals": 1,
            "Next": "FirstTask"
        },
        {
            "Variable": "$.foo",
            "NumericEquals": 2,
            "Next": "SecondTask"
        }
    ]}
```



Serverless Pattern



Applications

Percent	*Use Case*
32%	Web and API serving
21%	Data Processing, e.g., batch ETL (database Extract, Transform, and Load)
17%	Integrating 3rd Party Services
16%	Internal tooling
8%	Chat bots e.g., Alexa Skills (SDK for Alexa AI Assistant)
6%	Internet of Things

Table 4: Popularity of serverless computing use cases according to a 2018 survey [22].

Source: Cloud Programming Simplified: A Berkeley view on Serverless Computing

Why is Serverless (Un)attractive?

	On-premise	VMs	Containers	Serverless
Time to provision	Weeks-Months	Minutes	Seconds-minutes	Milliseconds
Utilization	Low	High	Higher	Highest
Charging granularity	CapEx	Hours	Minutes	Interval of milliseconds

- The Good
  - Removal of the need for a traditional always-on servers
  - Making app development dramatically faster, cheaper, easier
  - Highly available and scalable apps with zero administration
- The Bad
  - No in-server state for serverless functions
  - Limited computation times and memory can entail app refactoring
  - Functions are not directly network-addressable

Table source: Jason McGee, IBM; Serverless Conference 2017.

Why is Serverless Unattractive?

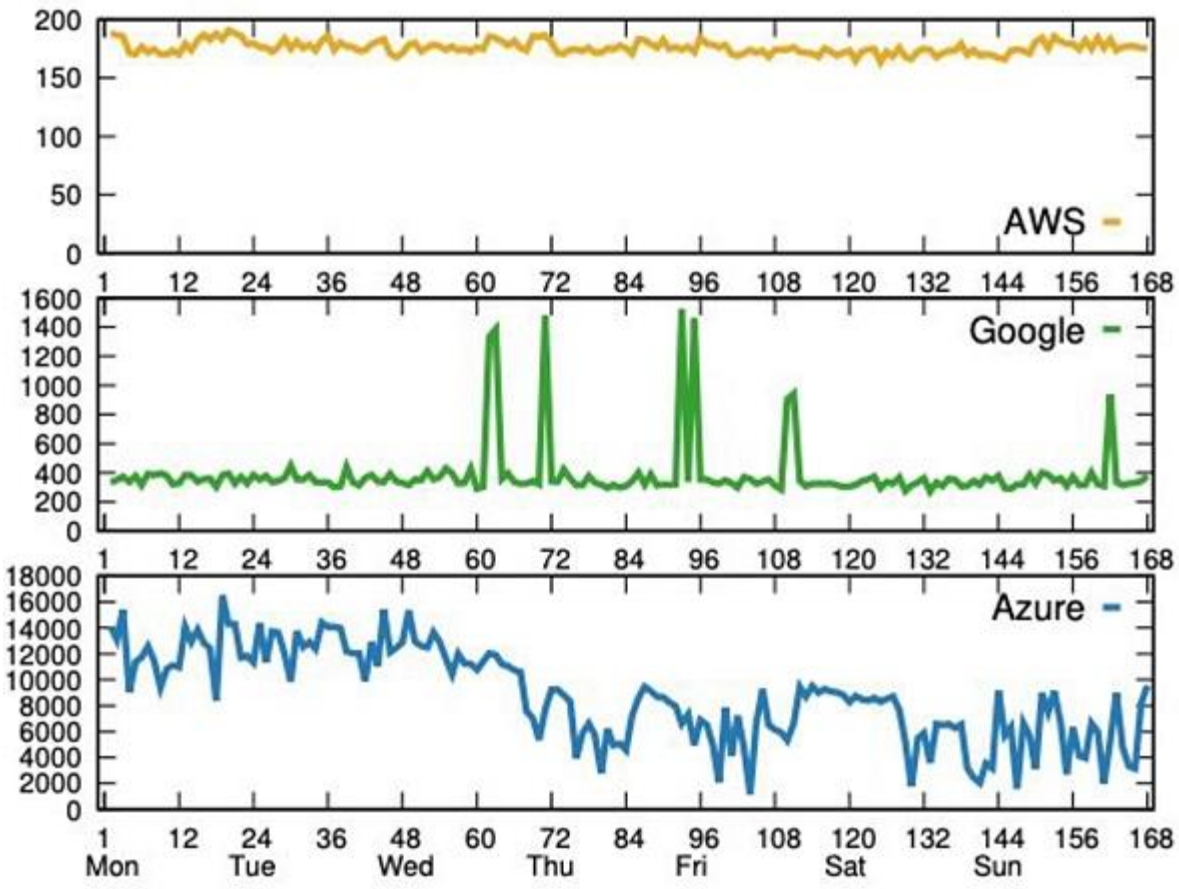


Table source: Behinds the Curtains of Serverless Platforms

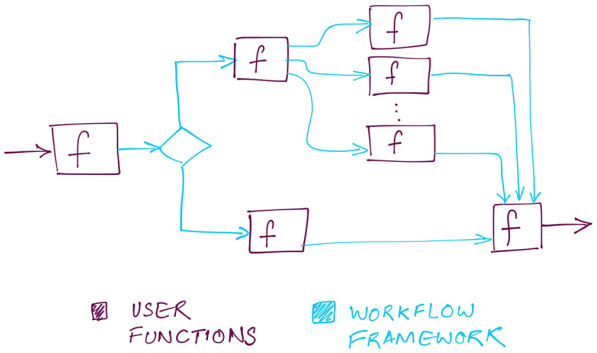
Why is Serverless Unattractive?

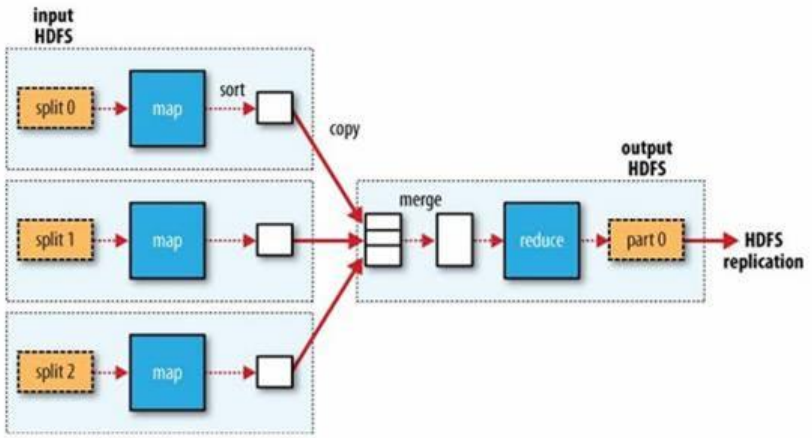
- FaaS is a Data-Shipping Architecture
- FaaS Stymies Distributed Computing (addressability, communication)
- FaaS stymies hardware-accelerated software innovation
- FaaS discourages Open Source service innovation.
- Limited Lifetimes
- Communication Through Slow Storage
- No guarantees, no QoS
- It can be expensive
- Cold starts !

Table source: Serverless Computing: One step forward, two steps back

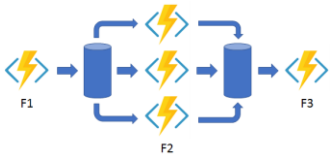
FAAS ORCHESTRATION

Creating Serverless Workflows





AWS Step Functions

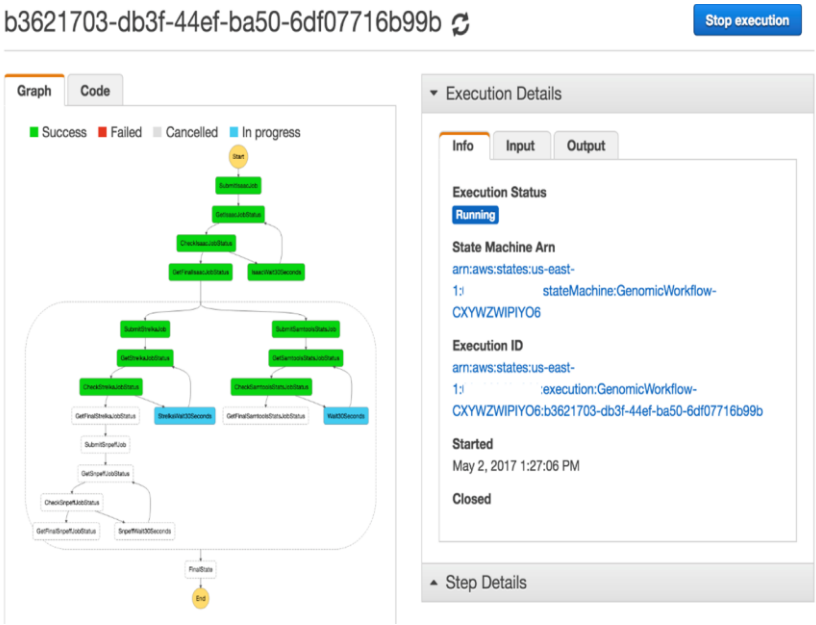


Azure Durable Functions

IBM Function Composer

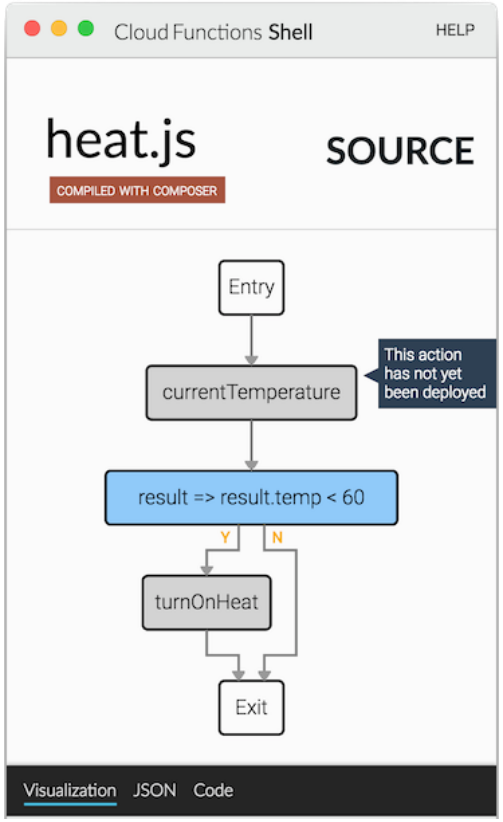
Amazon Step Functions


```
StateMachine.Builder stateMachineBuilder =
  stateMachine()
  .comment("A state machine with par. states.")
  .startAt("Parallel");

  Branch.Builder[] branchBuilders =
    new Branch.Builder[NSTEPS];

  for (int i = 0; i < NSTEPS; i++) {
    branchBuilders[i] = branch()
      .startAt(String.valueOf(i + 1))
      .state(String.valueOf(i + 1),
        taskState()
        .resource(arnTask).transition(end()));
  }

  stateMachineBuilder.state("Parallel",
    parallelState().branches(branchBuilders)
    .transition(end()));
  final StateMachine stateMachine =
    stateMachineBuilder.build();
```

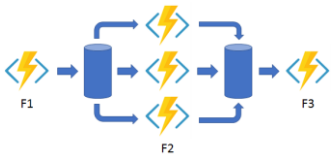


IBM Composer

```
composer.sequence(
// programmatic composition
  ‘currentTemperature’,
// call cloud function or API
  composer.if(
// conditional control flow
    result => result.temp < 60,
// mix inline JavaScript
    ‘turnOnHeat’)
// interface to 3rd party services
)
```



Azure Durable Functions




```
public static async Task Run(DurableOrchestrationContext ctx)
{
    var parallelTasks = new List<Task<int>>();

    // get a list of N work items to process in parallel
    object[] workBatch = await ctx.CallActivityAsync<object[]>("F1");
    for (int i = 0; i < workBatch.Length; i++)
    {
        Task<int> task = ctx.CallActivityAsync<int>("F2", workBatch[i]);
        parallelTasks.Add(task);
    }

    await Task.WhenAll(parallelTasks);

    // aggregate all N outputs and send result to F3
    int sum = parallelTasks.Sum(t => t.Result);
    await ctx.CallActivityAsync("F3", sum);
}
```

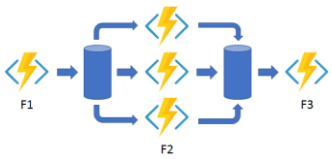
Sequences

```
StateMachine.Builder stateMachineBuilder =
  stateMachine()
  .comment("A Sequence state machine")
  .startAt("1");
for (int i = 1; i <= NSTEPS; i++) {
  stateMachineBuilder.state(String.valueOf(i),
    taskState().resource(arnTask)
      .transition((i != NSTEPS) ?
        next(String.valueOf(i + 1)) : end()));
}
StateMachine stateMachine =
  stateMachineBuilder.build();
```

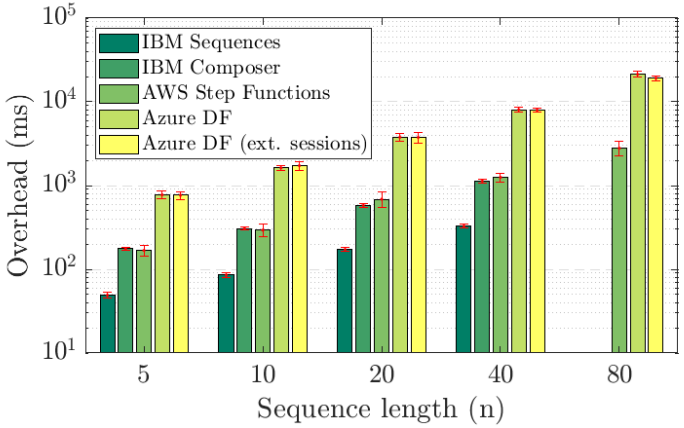
AWS Step Functions

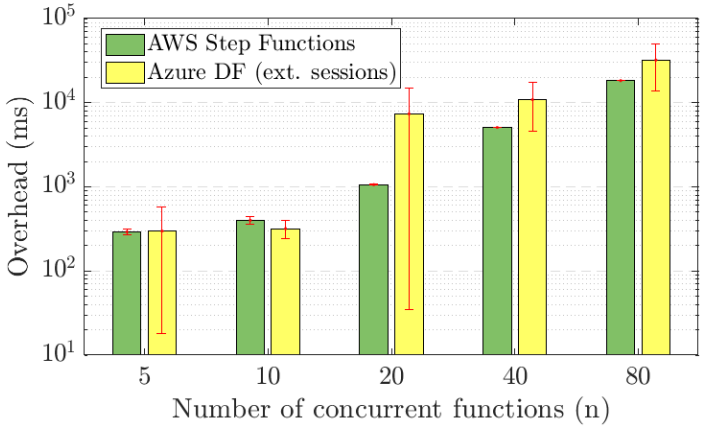
```
composer.repeat(40, 'sleepAction')
```

```
for (int i = 0; i < NSTEPS; i++) {
  await context.
    CallActivityAsync("sleepAction", null);
}
```



Sequences


IBM Sequences
IBM Composer
AWS Step Functions
Azure DF
Azure DF (ext. sessions)
Overhead (ms)
10^5
10^4
10^3
10^2
10^1
5
10
20
40
80
Sequence length (n)


AWS Step Functions
Azure DF (ext. sessions)
Overhead (ms)
10^5
10^4
10^3
10^2
10^1
5
10
20
40
80
Number of concurrent functions (n)

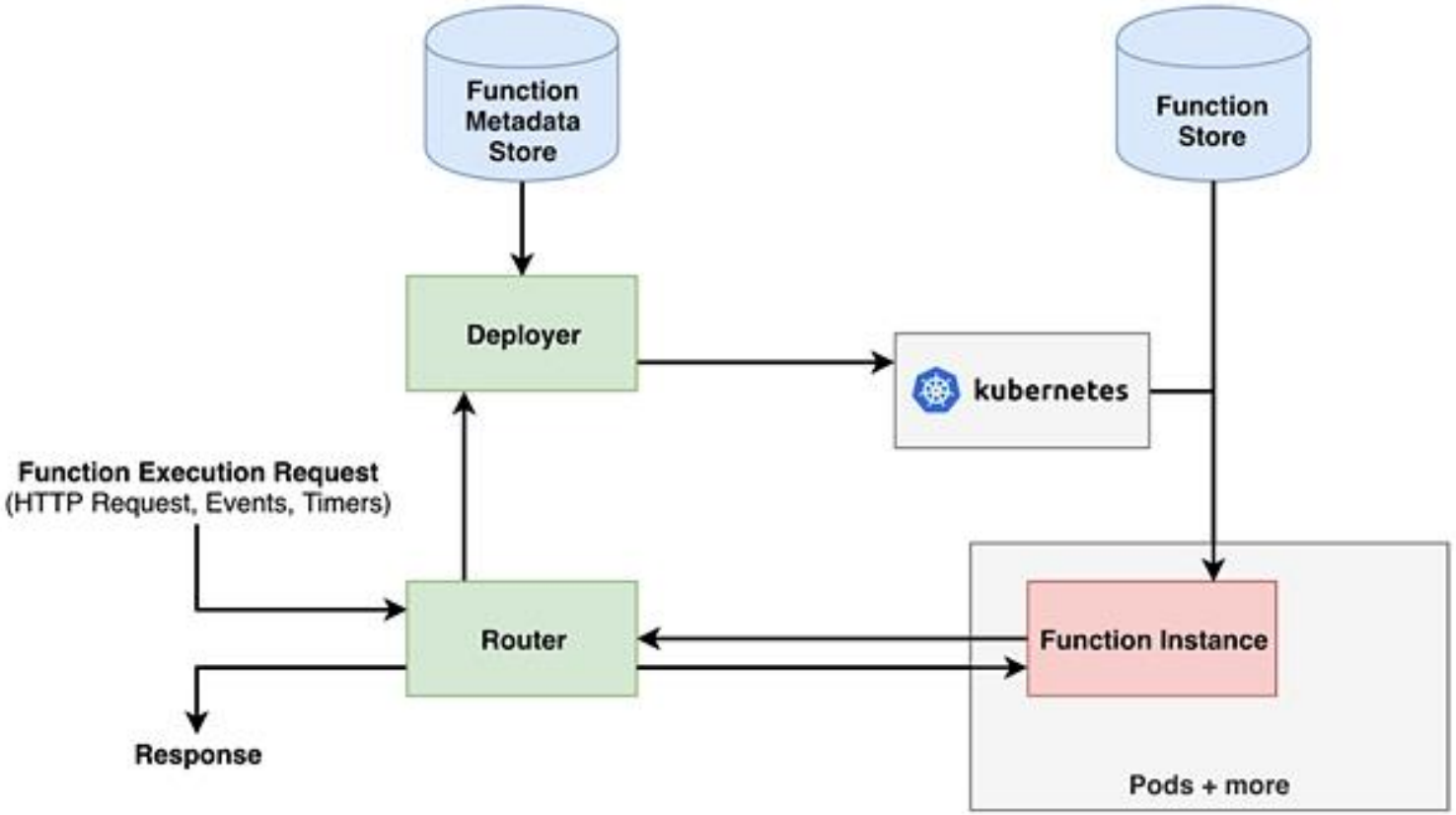
Comparison

- Amazon Step Functions is the most mature project
- Microsoft ADF is the more advanced in programmability, IBM Composer wins in simplicity
- None of them support parallel tasks efficiently
- Orchestration must have a cost if it is fault-tolerant
- Early immature projects with high potential for the future

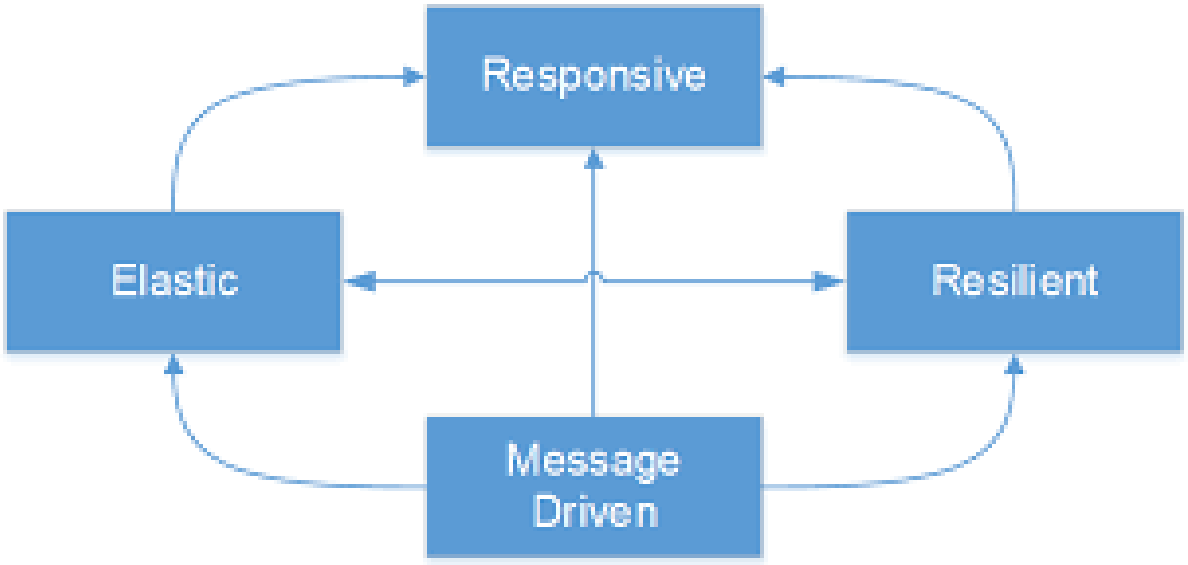
Table source: Comparison of FaaS Orchestration Systems

FAAS ARCHITECTURE

Architecture

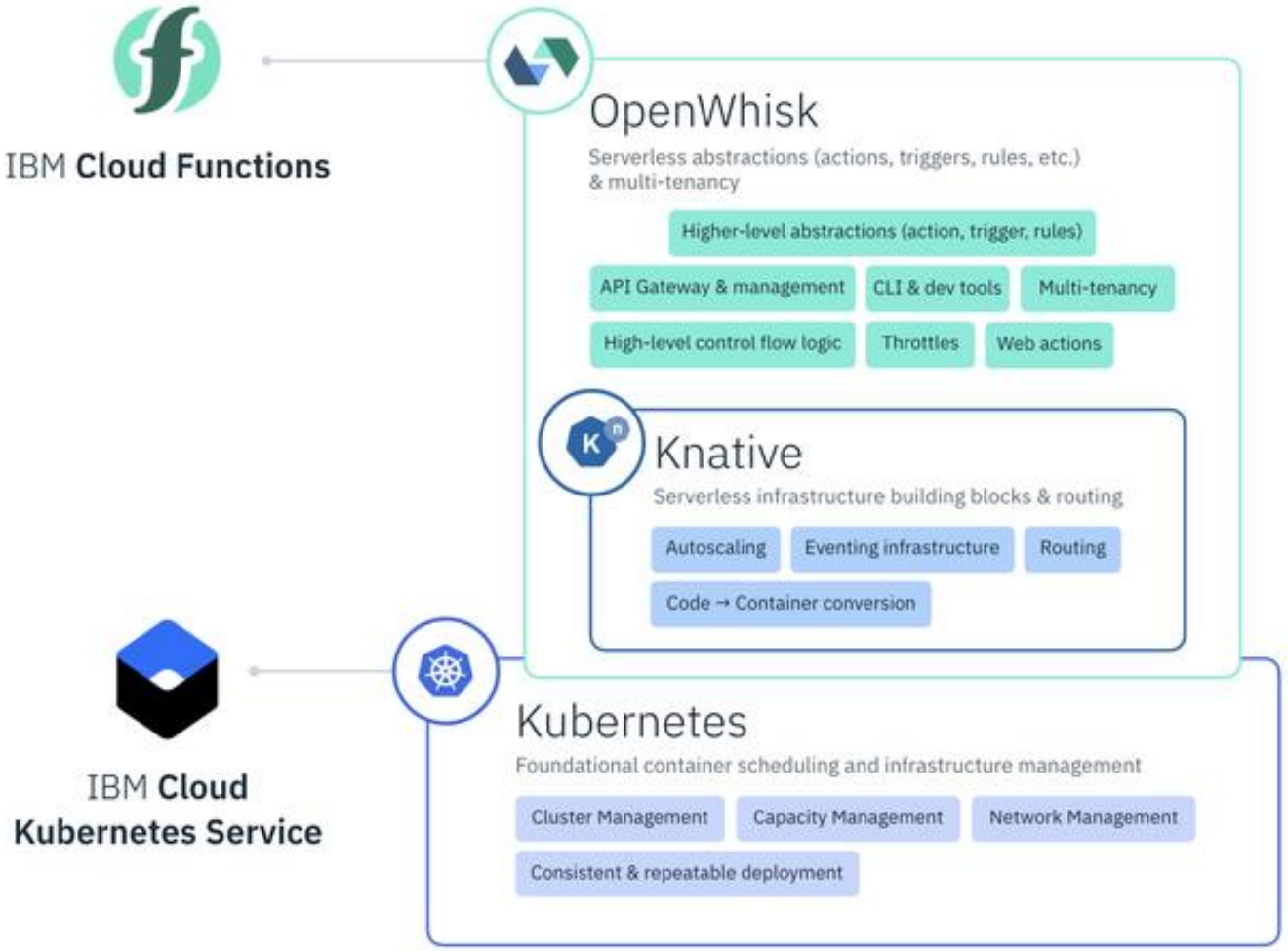


Reactive Core



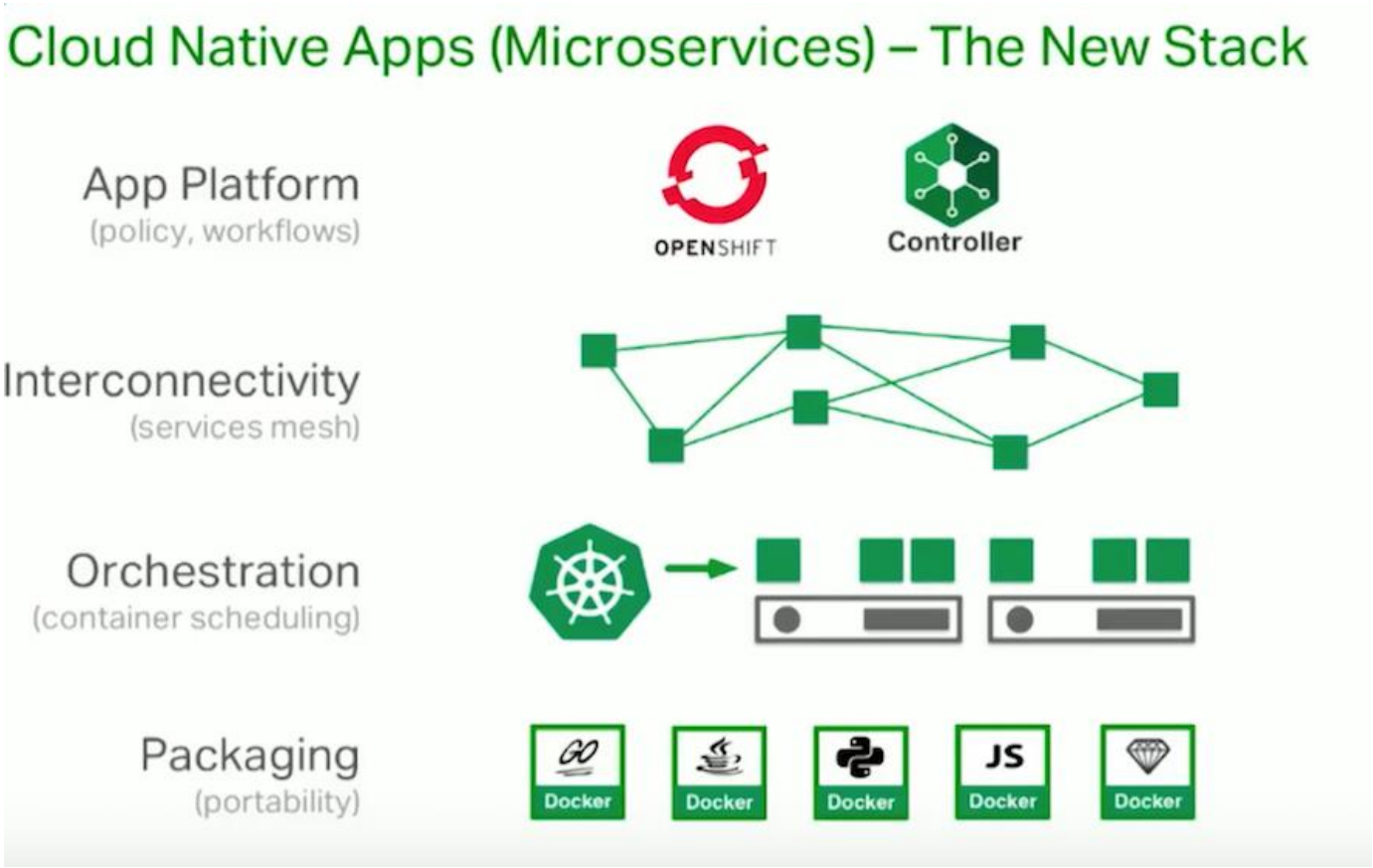
Knative

CLOUD NATIVE COMPUTING FOUNDATION



Service mesh

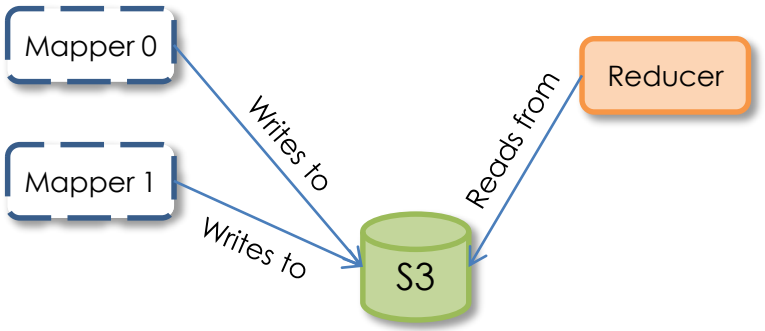
CLOUD NATIVE COMPUTING FOUNDATION



SERVERLESS DATA ANALYTICS

Serverless Data Analytics?

- Abide by the functional programming paradigm:
  - Embarrassingly parallel functions
  - Immutable data through "slow" storage (e.g., S3)
  - PyWren† and ExCamera‡ research projects show that functions can perform a wider variety of such "map" functions
  - PyWren†'s word count job on 83M items is only 17% slower than PySpark running on dedicated servers



† Occupy the Cloud: Distributed Computing for the 99%. ACM SOCC 2017

‡ Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. USENIX NSDI 2017

Serverless Data Analytics?

- One can do a lot of things with a map(function, data)

```
def addone(x):
    return x + 1

wrenexec= pywren.default_executor()
data = range(1, 10)
futures = wrenexec.map(addone, data)

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- Functional, declarative programming models simplify consistency and fault tolerance
- Domain experts tend to write imperative programs
  - Java, Matlab, C++, R, Python, Fortran, ...
- Mismatch between experts' coding skills and analytics

Why CloudButton ?

"Our proposal in this paper was motivated by a professor of computer graphics at UC Berkeley asking us "Why is there no **cloud button**?" He outlined how his students simply wish they could easily "push a button" and have theircode – existing, optimized, single-machine code – running on the cloud."

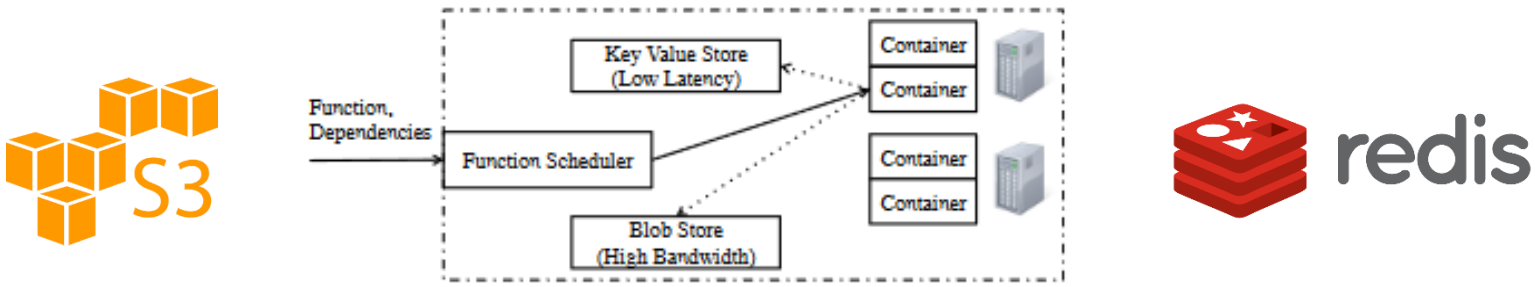


Figure 1: System architecture for stateless functions

Occupy the Cloud: Distributed Computing for the 99%. SOCC 2017.

Impact

- Serverless Data Analytics:
  - Much simpler for data scientists than analytics frameworks
  - Data lakes and FaaS
  - Run "single threaded" code in parallel
  - Use Cases: Data lakes, Data pipelines, Data flows, IoT

- In-Memory Computing:
  - In the industry: Apache Spark ecosystem
  - Much better than classical MapReduce for iterative processing
  - Operate on datasets in memory to run faster computations
  - Use cases: Sorting, clustering, regressions, interactive queries,…

Objectives

- The main goal is to create CloudButton: a Serverless Data Analytics Platform. CloudButton will "democratize big data" by overly simplifying the overall life cycle and programming model thanks to serverless technologies.

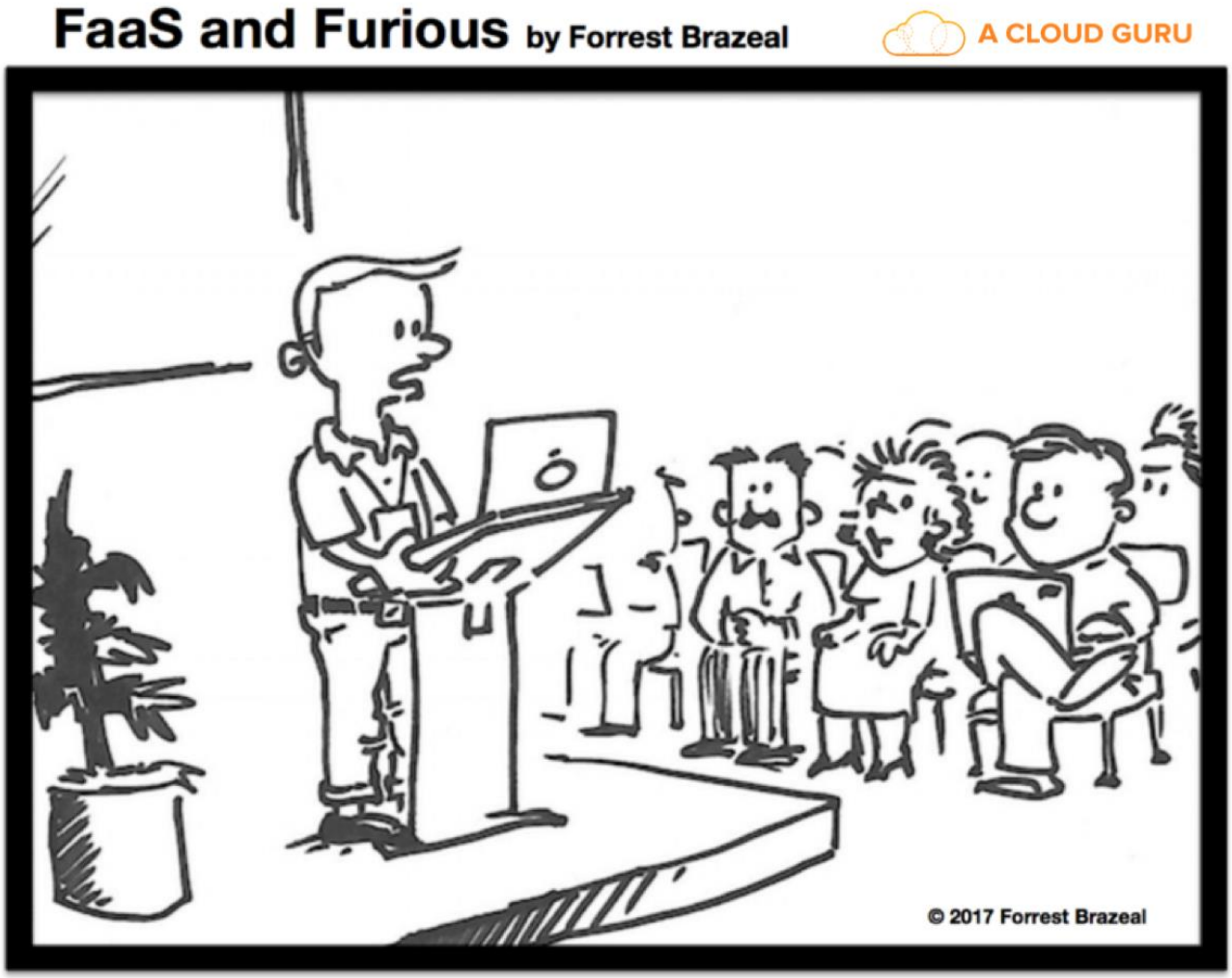
- To demonstrate the impact of the project, we target two settings with large data volumes: bioinformatics (genomics, metabolomics) and geospatial data (LiDAR, satellital).

CloudButton

THANK YOU!

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825184.

What is next ?



"Hello, and welcome to my recently retitled breakout session: 'My Cool Product That Became Irrelevant When AWS Released a New Service Right Before This Breakout Session'."