

Serverless / Function-as-a-Service

Assane Wade

Francisco Mendonça

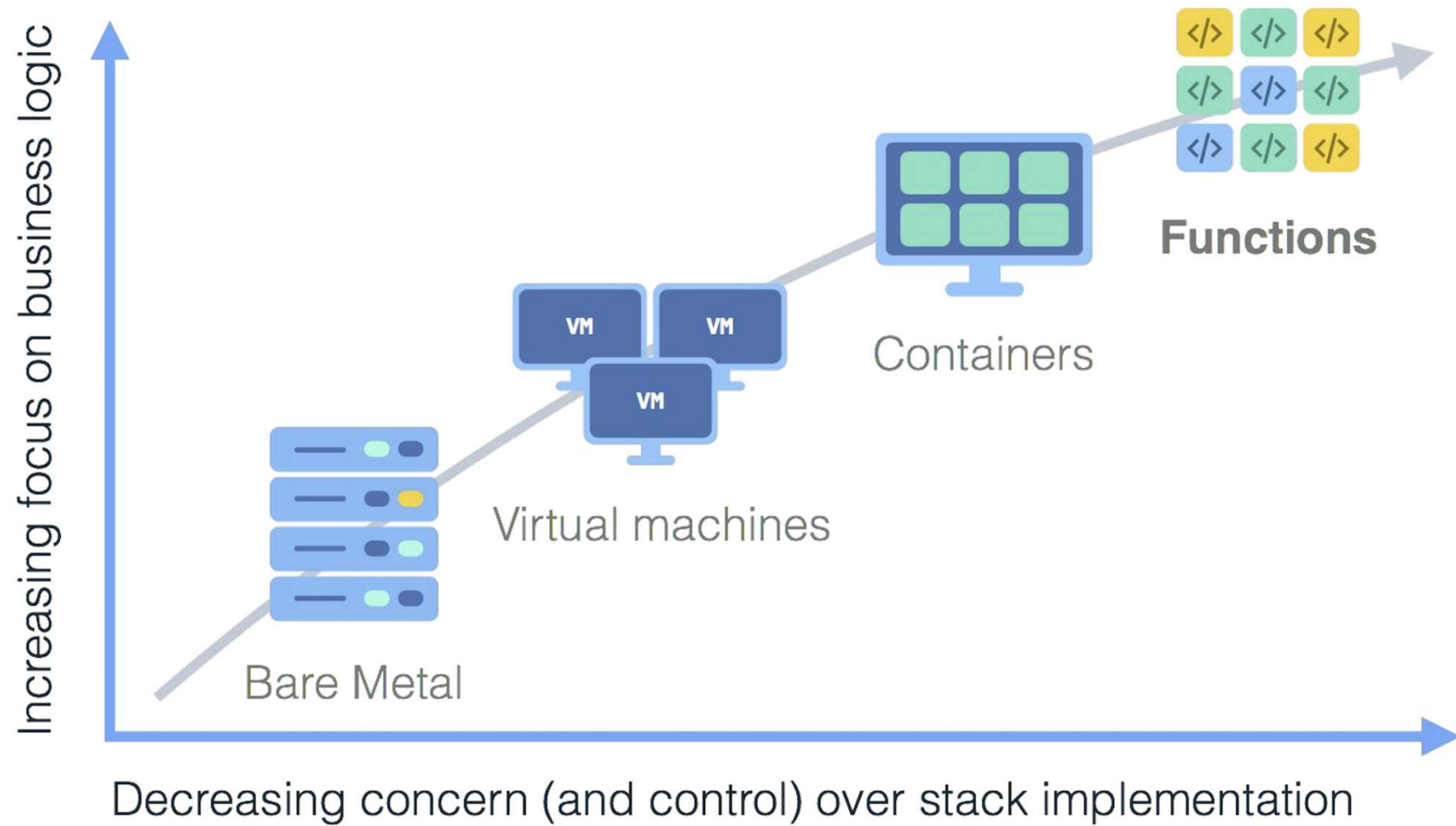
Gabriel Strano

Nabil Abdennadher

Overview

- The concepts *Serverless Computing* and *Function-as-a-Service*
- Motivation for FaaS
- AWS Lambda
- FaaS pricing model
- From three-Tier architecture to FaaS architecture (a use-case)
- Open-Source software for building FaaS platforms

Serverless computing



Serverless Computing

- Making devOps faster, cheaper and easier

	On-prem	VMs	Containers	Serverless
Time to provision	Weeks-months	Minutes	Seconds-Minutes	Milliseconds
Utilization	Low	High	Higher	Highest
Charging granularity	CapEx	Hours	Minutes	Blocks of milliseconds

Serverless Computing

- Serverless is a cloud native development model that allows developers to build and run applications without having to manage servers. “Cloud Native Computing Foundation”.
- Servers still exist but are abstracted away from the application development process.

Serverless Computing

Benefits for developers:

- Zero server ops
 - No provisioning, updating, and managing server infrastructure
 - Fully automated scalability — no need to pre-plan capacity or configure rules for autoscaling
- No compute cost when idle — no charge for idle VMs or containers

Serverless Computing

Serverless is **good** for
short-running
stateless
event-driven



Microservices



Mobile Backends



Bots, ML Inferencing



IoT



Modest Stream Processing



Service integration

Serverless is **not good** for
long-running
stateful
number crunching



Databases



Deep Learning Training



Heavy-Duty Stream Analytics



Spark/Hadoop Analytics



Numerical Simulation



Video Streaming

Serverless - Tools

- AWS Serverless Application Model (SAM) :is an open-source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings.
- Azure serverless : Build, deploy, and operate serverless apps on an end-to-end platform
- Google Cloud Workflows

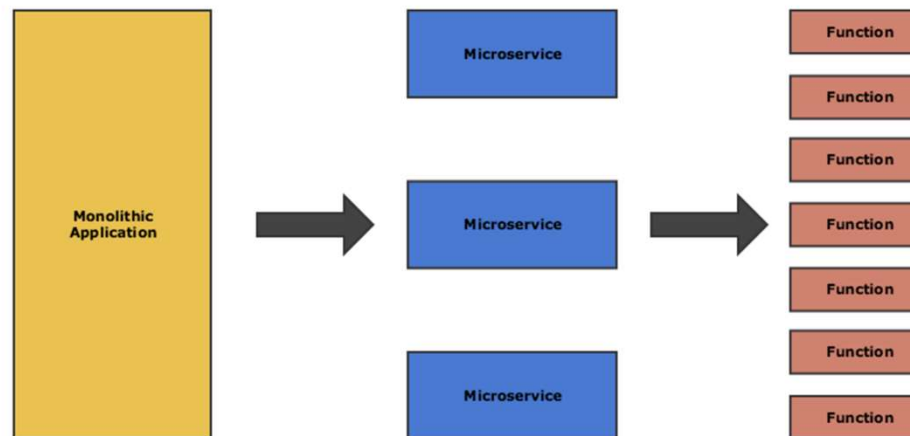
Function as a Service (FaaS)

- Function-as-a-Service, or FaaS, is a serverless way to run functions in any cloud environment.
- FaaS builds/runs applications without caring about provisioning, scaling, and managing any servers.
- With FaaS, it may not be running at all until the function needs to be executed. It starts the function within the needed time and then shuts it down.
- A new business model
- **Eg: AWS Lambda, Azure Functions, Cloud Functions, Manta, Openwhisk etc.**

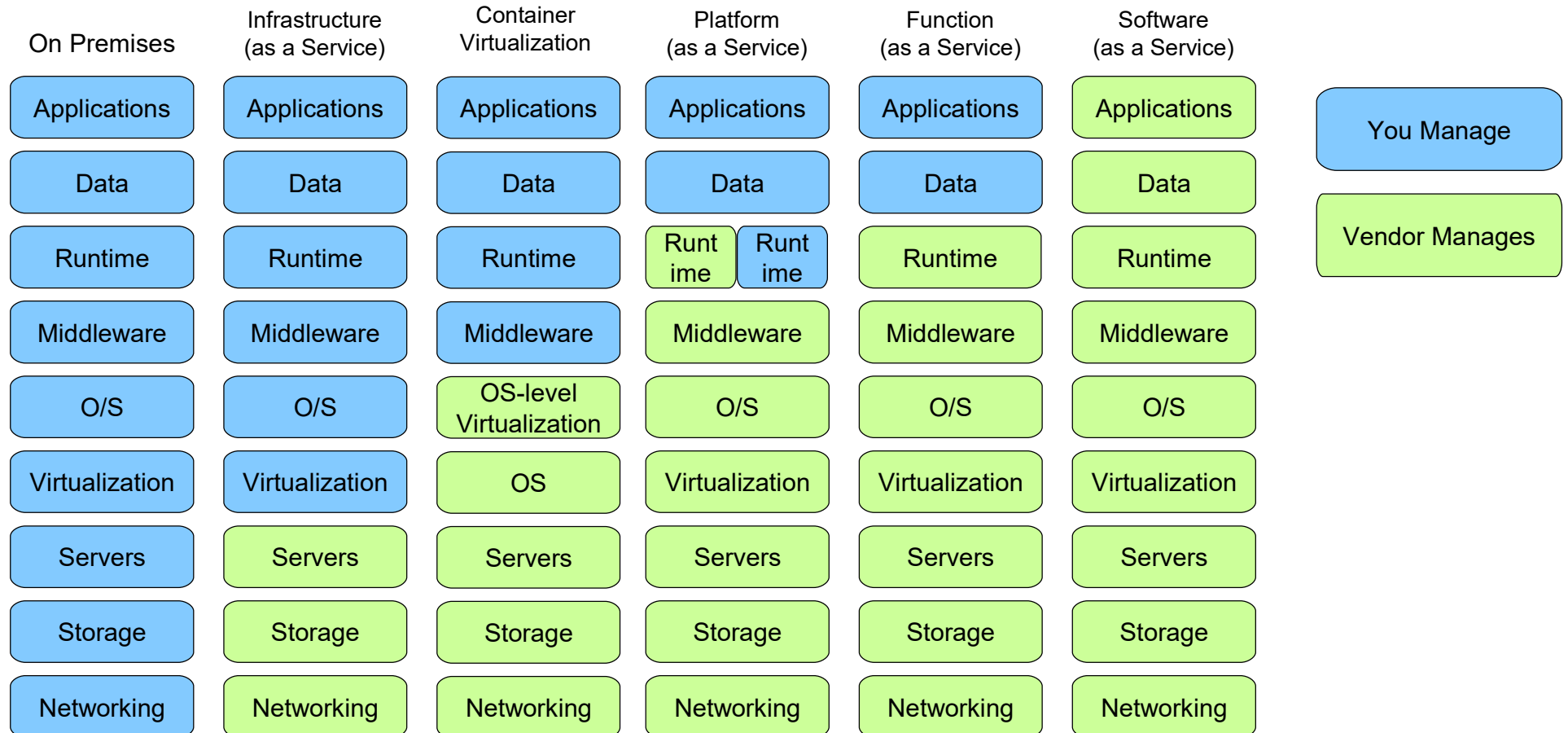
Function as a Service in a nutshell

- Run your code without supplying or administrating servers.
- No charge is applicable when your code is not running
- Scalability: Scaling up the IaaS infrastructure according to the needs
- Stateless functions: use external resources to manage the state of the application so that the state can be shared.

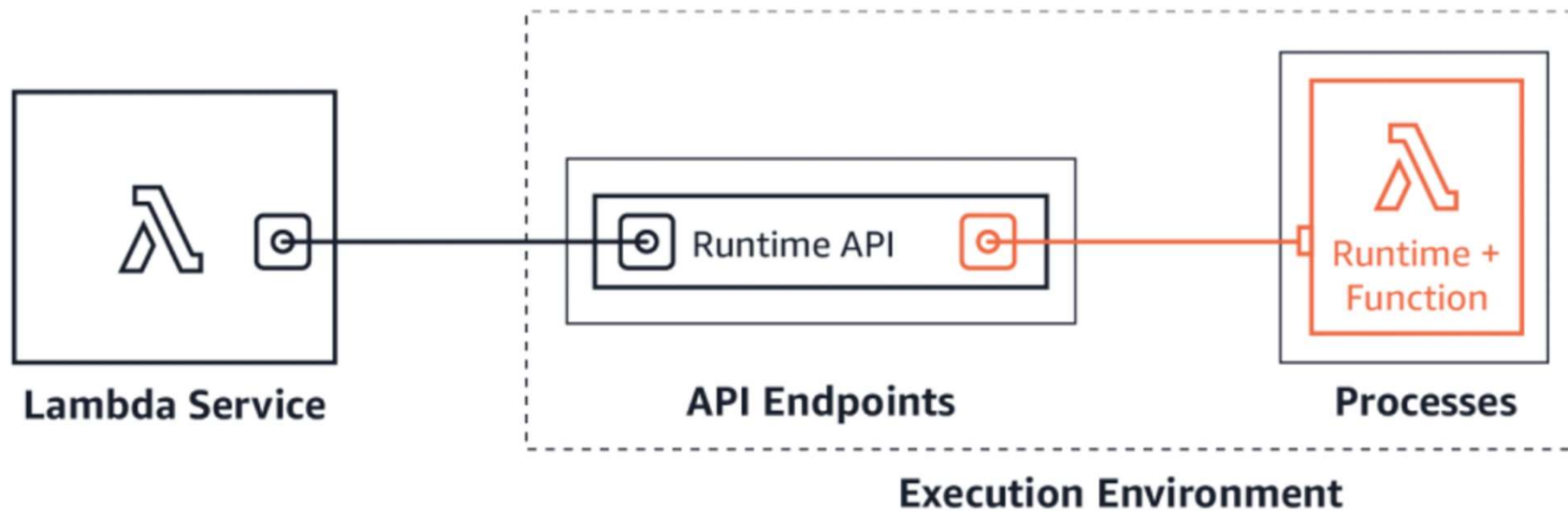
Monolithic vs Microservice vs FaaS



Cloud Service Models



AWS Lambda in a nutshell



[Code](#)[Test](#)[Monitor](#)[Configuration](#)[Aliases](#)[Versions](#)

Code source [Info](#)

[Upload from](#) ▼

File Edit Find View Go Tools Window

Test

Deploy

Changes deployed



Go to Anything (⌘ P)

Environment

- getSensorData - /
 - a3db.py
 - config.json
 - lambda_function.py
 - sensor.py
 - sensors_with_addresses.



lambda_function ×

config.json ×

Execution results ×

sensor.py ×

a3db.py ×



```
5 from sensor import Sensor
6
7 from flask import json, Response
8
9 import os
10
11 def lambda_handler(event, context):
12
13     db = a3db()
14     CONFIG = json.loads(open("config.json").read())
15
16
17     deveui = event["deveui"]
18
19     print("New Sensor: {}".format(deveui))
20     sensor = Sensor(deveui, CONFIG, db)
21
22     if sensor.empty():
23         return json.dumps({'error': 'Sensor exists but no data could be retrieved'})
24
25     json_df = sensor.df.drop(columns=["good_sample"]).to_json(orient='index', date_format='iso')
26     j = json.loads(json_df)
27
28     return j
29
```

handler

FaaS programming model

The *programming model* defines the interface between the developer's code and the runtime:

- The developer tells the runtime which method to run by defining a **handler** in the function configuration.
- When calling the handler, the runtime passes in two objects: Event & Context
 - Event: a JSON-formatted document that contains data for a Lambda function to process. The Lambda runtime converts the event to an object and passes it to the function code.

```
https://0wm4lhk445.execute-api.us-east-
```

```
1.amazonaws.com/StageName/NabilFunction?TableName=Music
```

```
{
```

```
  "TableName": "Music"
```

```
}
```

event

- Context object. A context object is passed to the function by Lambda at runtime. This object provides methods and properties that provide information about the invocation, function, and runtime environment.

FaaS programming model (Context)

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:", context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time remaining in get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:", context.get_remaining_time_in_millis())
```

For Lambda context object in Python, see this [link](#)

Lambda concept (Environment variables)

- Adjust your function's behaviour without updating code
- Environment variables are not evaluated before the function invocation.
- A key value pair

Example:


















```
import os  
region = os.environ['AWS_REGION']
```

ENVIRONMENT	DEVELOPMENT	Remove
databaseHost	lambdadb	Remove
databaseName	rd1owwlydynnm5.cuovuayfg087	Remove
Key	Value	Remove

Lambda concept (Trigger)

Trigger: a resource or configuration that invokes a Lambda function. Triggers include AWS services that you can configure to invoke a function

Examples of
function triggers
(AWS services)

 API Gateway api application-services aws serverless	 Kinesis analytics aws streaming
 AWS IoT aws devices iot	 MQ aws messaging multi-protocol
 Alexa Skills Kit alexa iot	 MSK aws cluster
 Alexa Smart Home alexa iot	 S3 aws storage
 Application Load Balancer aws load-balancing	 SNS aws messaging notifications pub-sub push
 CloudFront aws cdn edge	 SQS aws queue
 CloudWatch Logs aws logging management-tools	
 CodeCommit aws developer-tools git	
 Cognito Sync Trigger authentication aws identity mobile-services sync	
 DynamoDB aws database nosql	
 EventBridge (CloudWatch Events) aws events management-tools	

Lambda concept (Trigger)

Examples of function triggers (third parties)

Partner event sources (powered by Amazon EventBridge)



Atlassian - Opsgenie

Opsgenie is a modern incident management platform for operating always-on services, empowering DevOps teams to stay in control during incidents



Auth0

Auth0, the identity platform for application builders, provides developers and enterprises with the building blocks they need to secure their applications



BUIDLhub

BUIDLHub offers products and services that simplify Blockchain integration with traditional and decentralized web infrastructure



Blitline

Blitline is the industry leader in enterprise level, massively parallel, image and document processing services



Buildkite

Buildkite is a continuous integration and deployment platform, helping you automate and scale the testing and delivery of all your software projects



CleverTap

CleverTap is a customer retention platform that helps consumer brands to maximize user lifetime value and fuel rapid growth



Datadog

Datadog is the essential monitoring platform for cloud applications



Epsagon

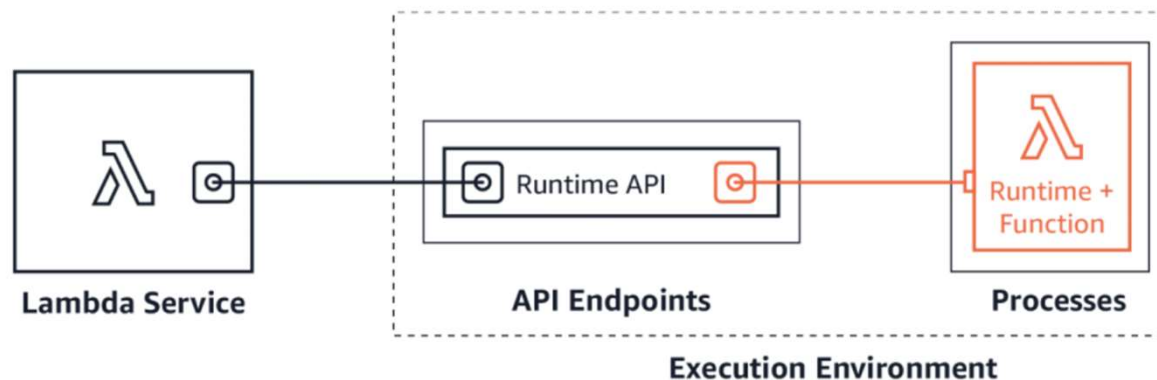
Epsagon delivers automated, cloud-native application performance monitoring and troubleshooting for modern applications, including microservices and serverless

Lambda concept (Runtime)

- Provides a language-specific environment that runs in an execution environment.
- The user can use runtimes that Lambda provides (.zip), or build his own (container image).
- For a container image, you include the runtime when you build the image.
- The runtime converts the event to an object and passes it to the function code.
- Runtimes: Node.js, Python, Ruby, Java, .NET, Go

Lambda concept (Execution environment)

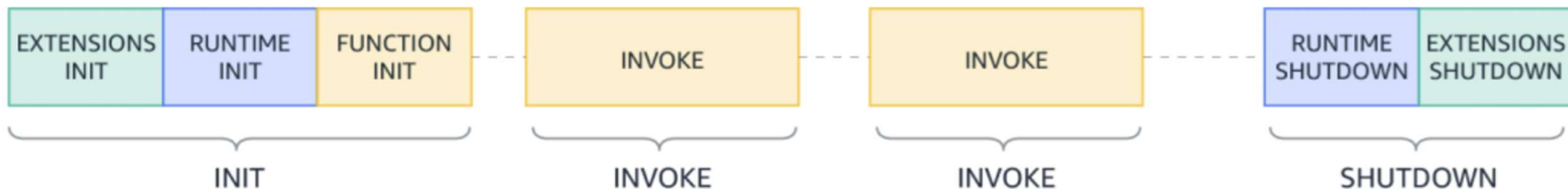
- Secure and isolated runtime environment
- Lambda uses configuration information (memory, execution time, etc.) to set up the execution environment.
- After the function runs, the Lambda service freezes the execution environment and maintains it for some time in anticipation of another function invocation.



Lambda concept (Execution environment)

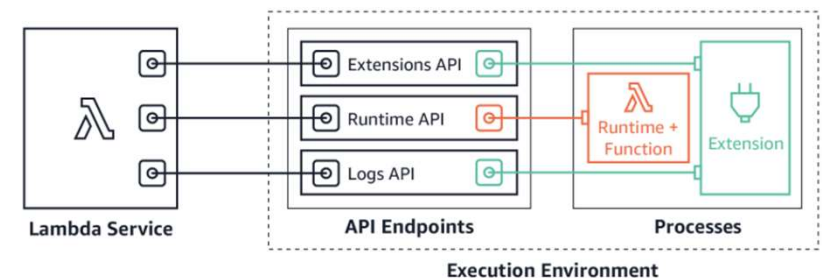
The runtime process exposes three distinct phases in the lifecycle of the Lambda execution environment: *Init*, *Invoke* and *Shutdown*.

- **Init:** The Init phase happens either during the first invocation, or in advance of function invocations if you have enabled provisioned concurrency. The Init phase is split into three sub-phases: Extension init, Runtime init, and Function init.
- **Invoke:** In this phase, Lambda invokes the function handler. After the function runs to completion, Lambda prepares to handle another function invocation.
- **Shutdown:** triggered if the Lambda function does not receive any invocations for a period of time.



Lambda concept (Extensions)

- Used to solve a common request: How to smoothly integrate existing tools with Lambda?
- Extension use-cases:
 - capturing diagnostic information before, during, and after function invocation
 - fetching configuration settings or secrets before the function invocation
- Lambda supports external and internal extensions:
 - An external extension runs as an independent process in the execution environment and continues to run after the function invocation is fully processed. Because extensions run as separate processes, you can write them in a different language than the function.
 - An internal extension runs as part of the runtime process



Lambda concept (Layer)

- A Lambda *layer* is a .zip file archive that can contain additional code or other content
- Extensions are deployed as Lambda layers
- Layers provide a convenient way to package libraries and other dependencies used by the Lambda functions.
- Functions deployed as a container image do not use layers. Instead, you package your runtime, libraries, and other dependencies into the container image when you build the image.

Lambda concept (Deployment package)

Two types of deployment packages:

- A .zip file (**up to 250 MB**)
 - Contains the function code and its dependencies.
 - Lambda provides the operating system and runtime for the function.
 - The .zip file can be uploaded from Amazon Simple Storage Service (Amazon S3) or the local machine
- A container image (**up to 10 GB**)
 - Contains the function code and dependencies to the image, the operating system and a Lambda runtime.
 - Lambda provides a set of open-source base images that you can use to build the container image
 - The container image is stored in the Amazon Elastic Container Registry (Amazon ECR). To deploy the image, the user must specify the Amazon ECR image URL.

Lambda concept (Qualifier)

- When you invoke or view a function, you can include a qualifier to specify a version or alias.
- A version is an immutable snapshot of a function's code and configuration that has a numerical qualifier. For example, my-function:1.
- An alias is a pointer to a version that you can update to map to a different version, or split traffic between two versions. For example, my-function:BLUE.
- You can use versions and aliases together to provide a stable interface for clients to invoke your function.

<https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html>

Lambda concept (Destination)

- A *destination* is an AWS resource where Lambda can send events from an asynchronous invocation.
- You can configure a destination for events that fail processing.
- Some services also support a destination for events that are successfully processed.

<https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html>

Lambda concept (Concurrency)

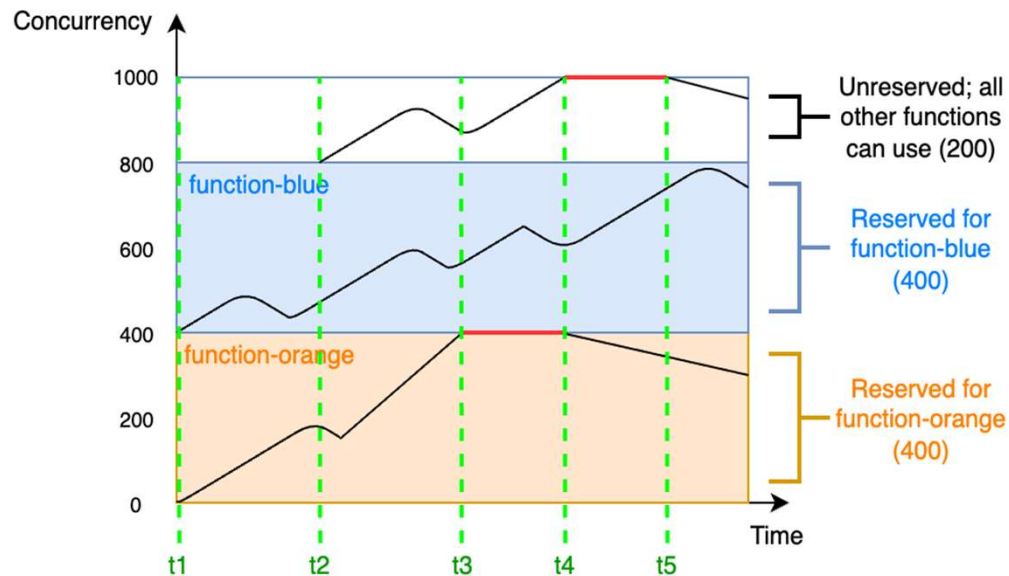
- The number of requests that your function is serving at any given time
- The total concurrency for all of the functions in your account is subject to a per-region quota
- Two types of concurrency controls: Reserved concurrency and Provisioned concurrency

<https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html>

Lambda concept (Concurrency)

Reserved concurrency (no charge)

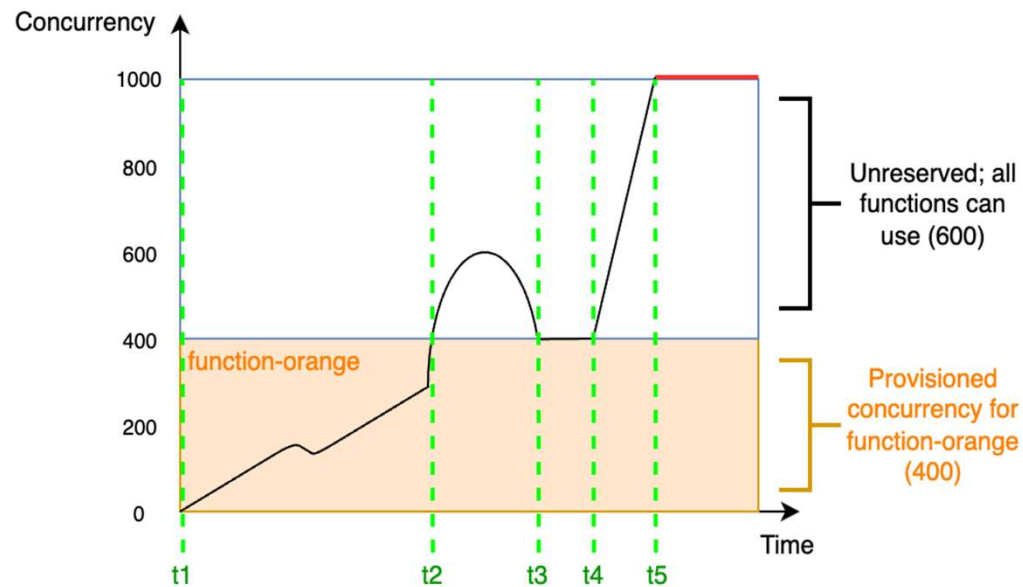
- Guarantees the maximum number of concurrent instances for the function.
- When a function has reserved concurrency, no other function can use that concurrency.



Lambda concept (Concurrency)

Provisioned concurrency (incurs charges to the AWS account)

- initializes a requested number of execution environments so that they are prepared to respond immediately to your function's invocations.



The cold start problem

- Cold start is about the delay between the execution of a function after someone invokes it.
- Cold start is about the time it takes to bring up a new container instance when there are no warm containers available for the request.

FaaS pricing model

- Typical FaaS pricing models have two components:
 - Billing by resources consumed (execution time x memory)
 - Billing by number of invocations
 - Ingress/egress network traffic is billed separately
- This is very different from IaaS or PaaS pricing that bills for VM/instance allocation time

FaaS pricing model

Pricing examples

	AWS Lambda	Azure Functions	Google Cloud Functions
1M invocations	US\$ 0.20	US\$ 0.20	US\$ 0.40
Execution time granularity	100 ms	100 ms	100 ms
1Ms execution time with 1024 MB memory	US\$ 16.7	US\$ 16	US\$ 16.5
Free tier (per month)	1 million invocations, 400'000 GB-s	1 million invocations, 400'000 GB-s	2 million invocations, 400'000 GB-s

Aws Step Functions

- AWS Step Functions is a visual workflow service that helps developers use AWS services to build distributed applications, automate processes, orchestrate microservices, and create data and machine learning (ML) pipelines.
- Same As Workflows in Google Cloud or Azure Logic Apps

Aws Step Functions



Software for building FaaS platforms

Open Source software for FaaS platforms:

- OpenFaaS — Independent project, funded through donations.
- Fn Project
- Fission
- OpenWhisk
- Kubeless
- TriggerMesh

Many of them use Docker containers as sandboxes and deploy them on Kubernetes

OpenWhisk

- Created under the Name Whisk in 2015 IBM research
- Launched in 2016 under the name Openwhisk
- 2016: IBM Cloud Functions from Openwhisk
- 2016: Became part of Apache Software Foundation Incubator

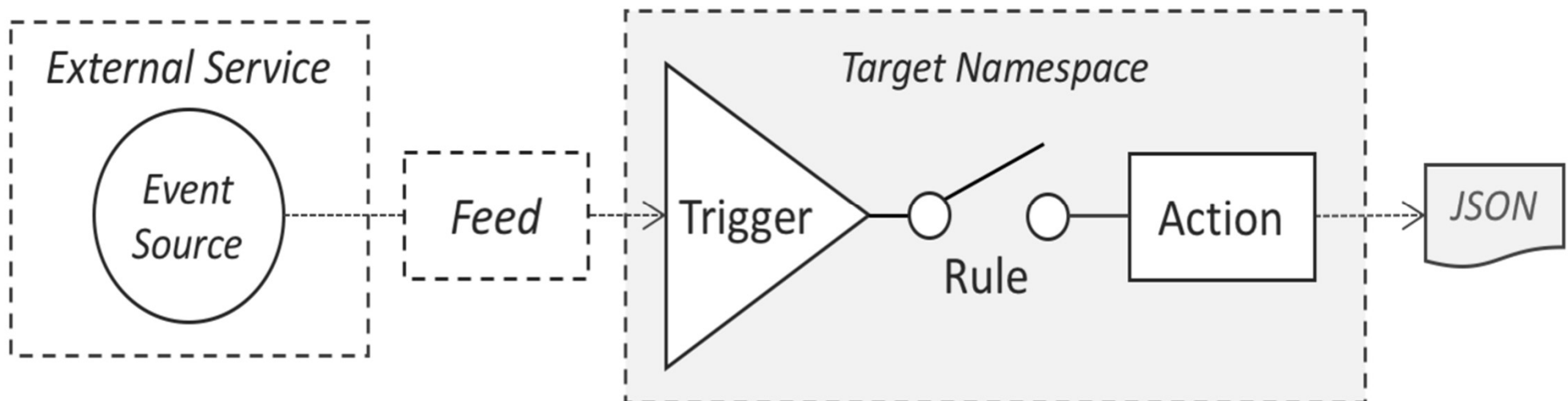


OpenWhisk

- Open source
- Serverless platform that executes functions (fx) in response to events at any scale.
- OpenWhisk manages the infrastructure, servers and scaling using Docker containers

OpenWhisk – Programming Model

The OpenWhisk platform supports a programming model in which developers write functional logic (called **Actions**), in any supported programming language, that can be dynamically scheduled and run in response to associated events (via **Triggers**) from external sources (**Feeds**) or from HTTP requests.

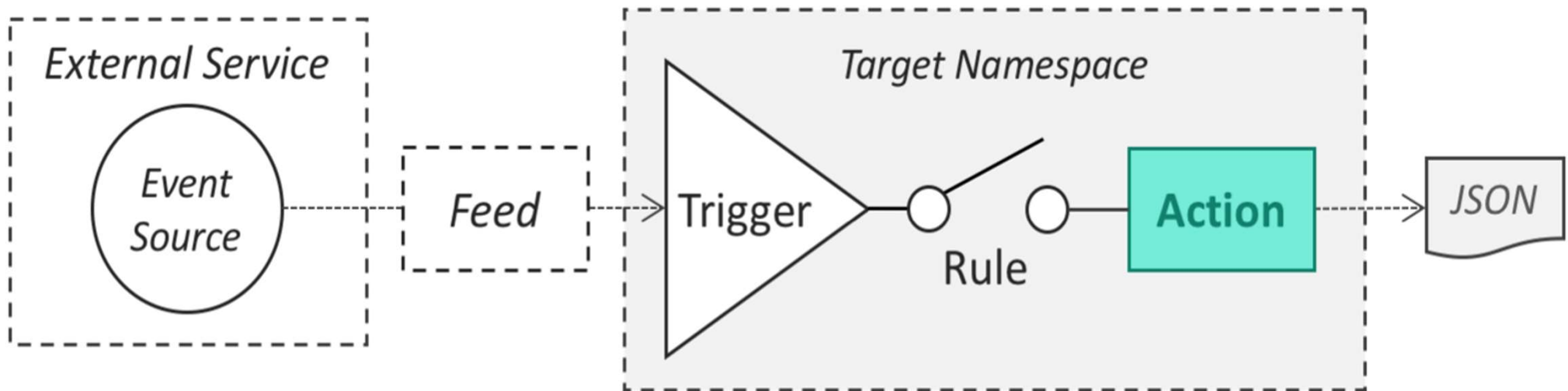


Programming Languages



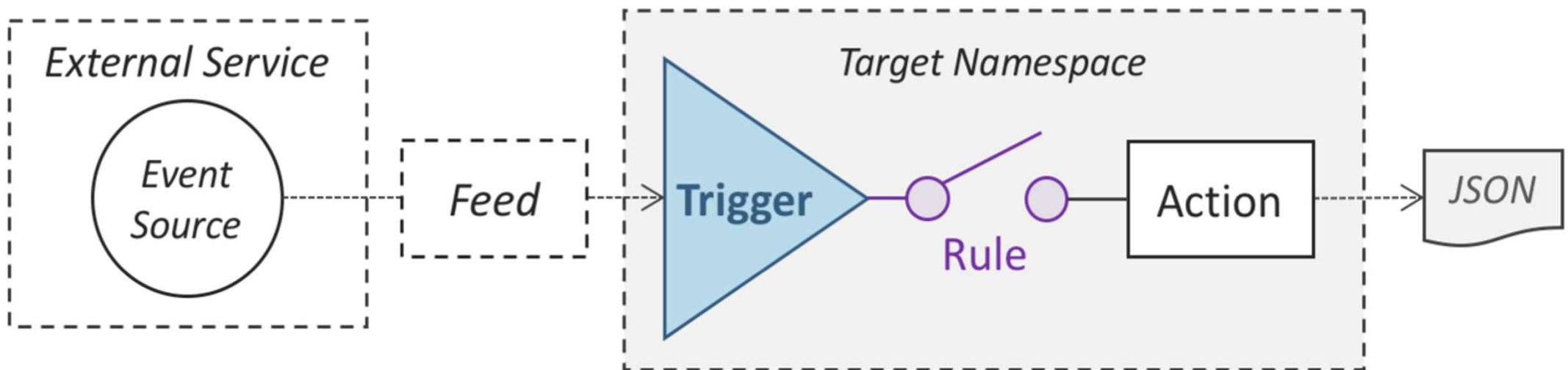
OpenWhisk - Actions

Actions are stateless functions (code snippets) that run on the OpenWhisk platform. Actions encapsulate application logic to be executed in response to events. Actions can be invoked manually by the OpenWhisk REST API, OpenWhisk CLI, simple and user-created APIs or automated via Triggers which we will discuss later.



OpenWhisk - Triggers and Rules

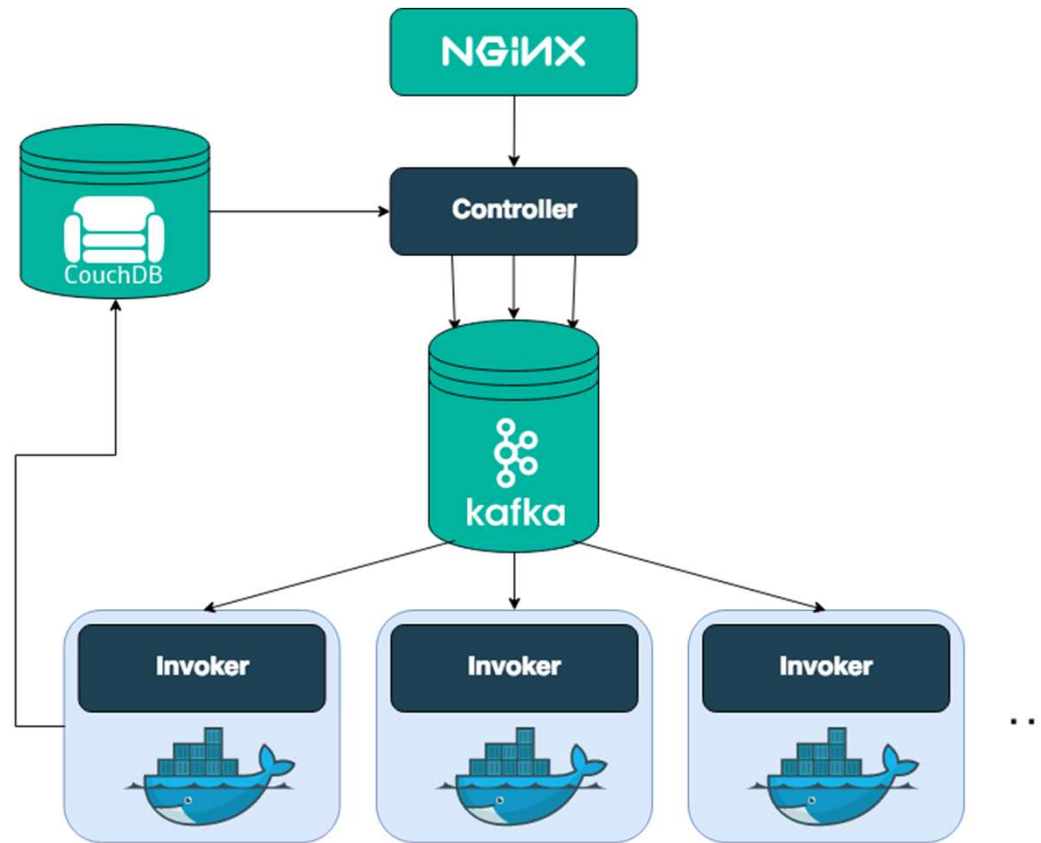
- Triggers are named channels for classes or kinds of events sent from Event Sources.
- Rules are used to associate one trigger with one action. After this kind of association is created, each time a trigger event is fired, the action is invoked.
- Event Sources? These are services that generate events that often indicate changes in data or carry data themselves (Message Queues, Changes in Databases, Changes in Document Stores, Website or Web Application interactions)



OpenWhisk - Definition

- **Trigger** : Triggers are named channels for classes or kinds of events sent from Event Sources.
- **Action** :functional logic
- **Rule** : Rules are used to associate one trigger with one action.
- **Invoke** : Invokes a deployed function. You can send event data, read logs and display other important information of the function invocation.

Openwhisk - Architecture



<https://github.com/apache/openwhisk/blob/master/docs/about.md#how-openWhisk-works>

Openwhisk - Deployment

Local or Cloud

Docker-compose

Ansible

Vagrant

Kubernetes

Use Case – Project MODIS

- What is Dnext?
 - Commodity Data Platform
 - Collect data in public and private databases such as government information, customs, lineups, or freight.
- Goal :
 - Provide data.
 - Using advantage of data science to develop
 - unique analytical features,
 - hence pushing the forecasting
 - data sharing within companies and between them



Project MODIS - Goal

The technical goal of this project is to :

- Download and extract data from satellite files
- manipulate and process these data to achieve our Business goal

The Business goal of this project is to :

- Compare historical graphs of the growth of crop for specific state
- Predict the harvest time based on the weather
- Predict the price of a crop
- Predict how many weight of crop can be harvest
- ...



Project MODIS – Use case

To compare the growth between each year for specific states, we need:

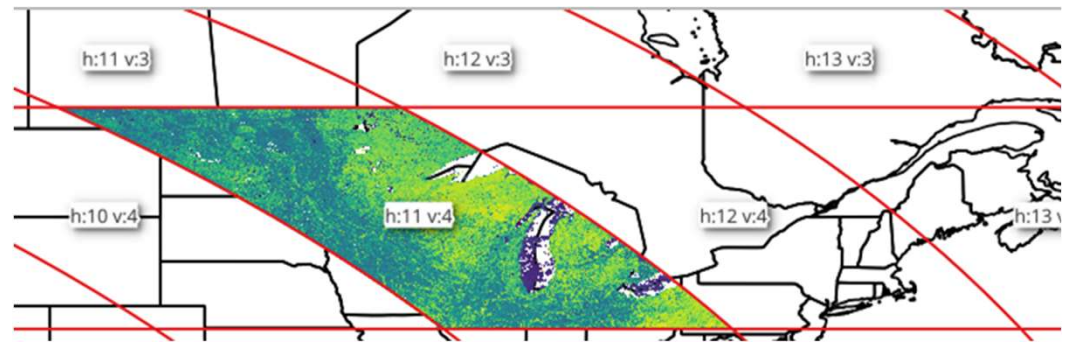
- Retrieve picture from a satellite
- Extract data corresponding to a specific crop (carrot)
- Apply filter for retrieving data for a state
- Make an average of this data
- Save in a database
- GUI for the costumers



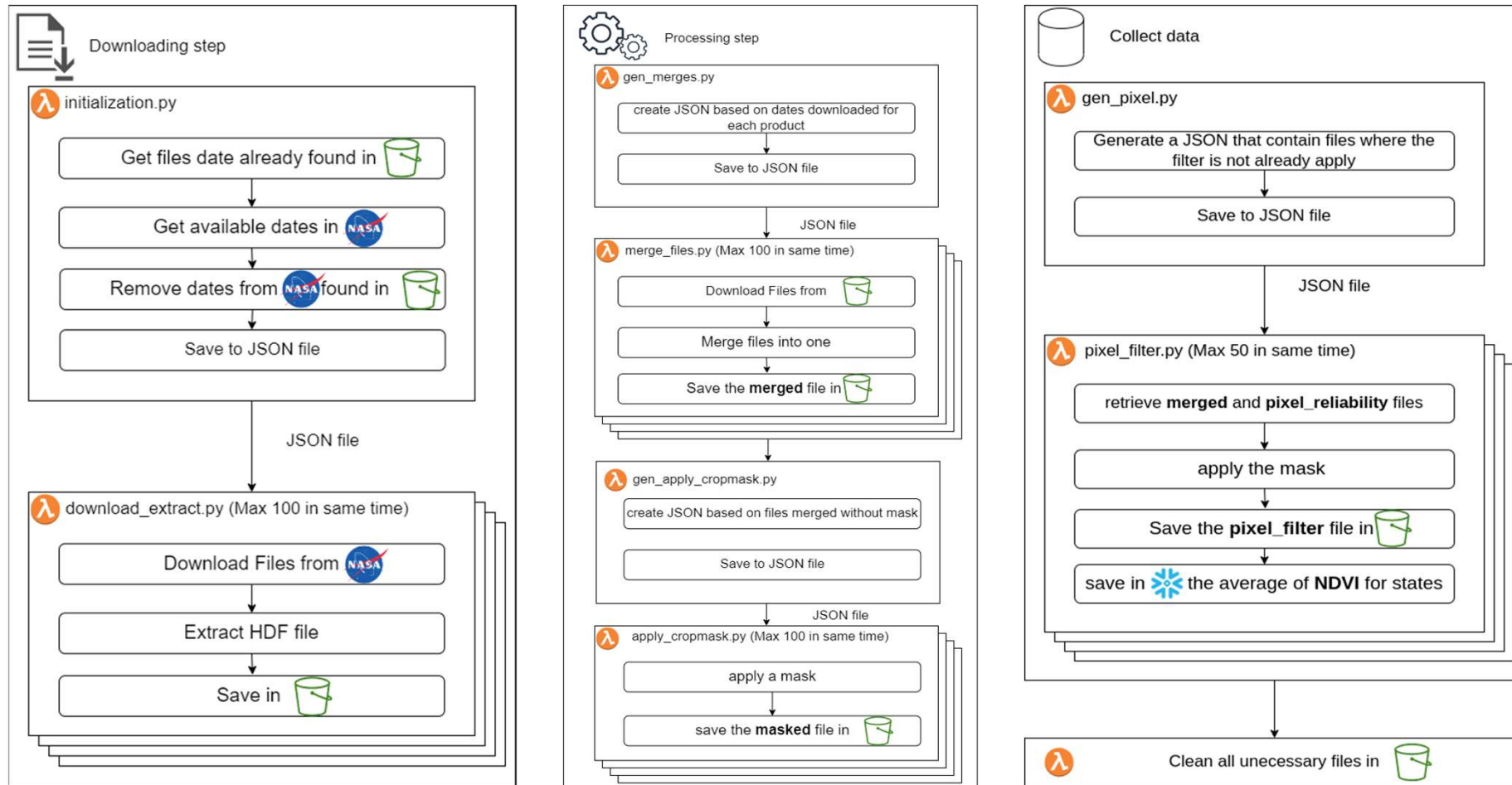
Project MODIS – Complexity

Retrieve picture from a satellites

- We choose NASA satellites MOLA and MOLT
 - Data we would retrieve is NDVI (Normalized Difference Vegetation Index, healthiness of the vegetation).
 - The size of a picture taken by a satellites is very big it's for that they decide to split it
Into multiple pictures called tiles (red line)
 - We need to download these tiles and extract the data from an HDF file (Hierarchical Data Format)
 - Merge these tiles into one file
 - Apply a mask to keep only place where
the crop is planted
 - Retrieve the boundary of a state and
make an average of the NDVI value



Project MODIS – Step function



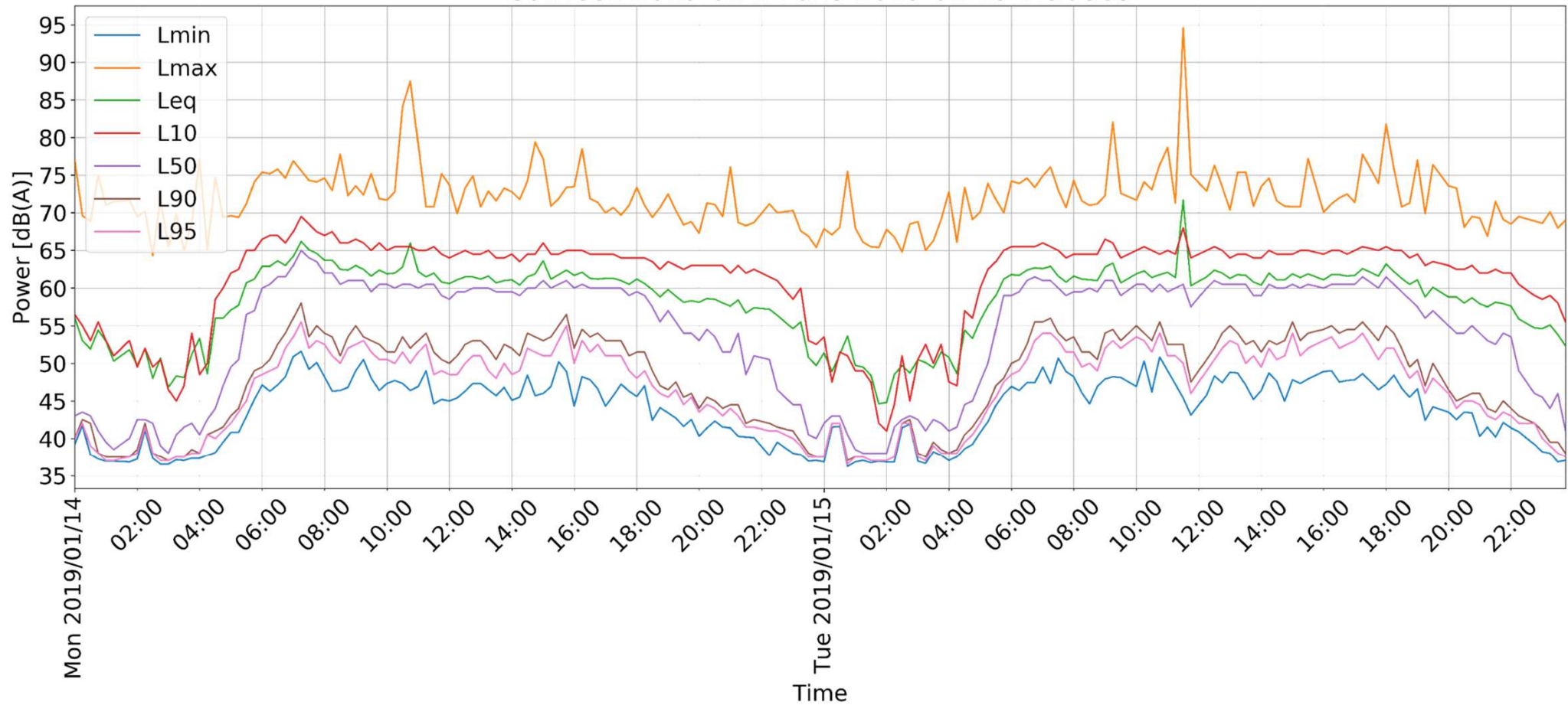
The trust Noise application

- ~ 600 sensors deployed in Carouge
- Send reports every 15 minutes: Lmin, Lmax, Leq, L10, L50, L90, L95
- Communicate through LoRaWAN network



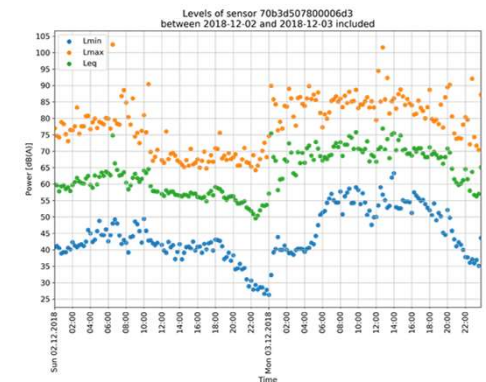
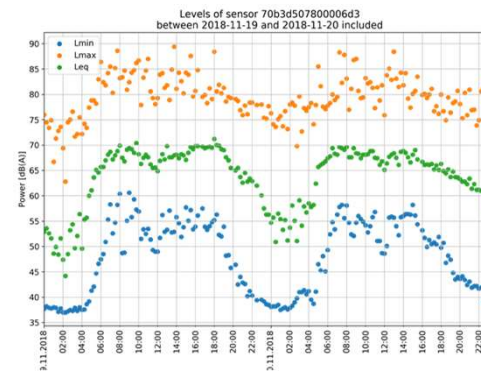
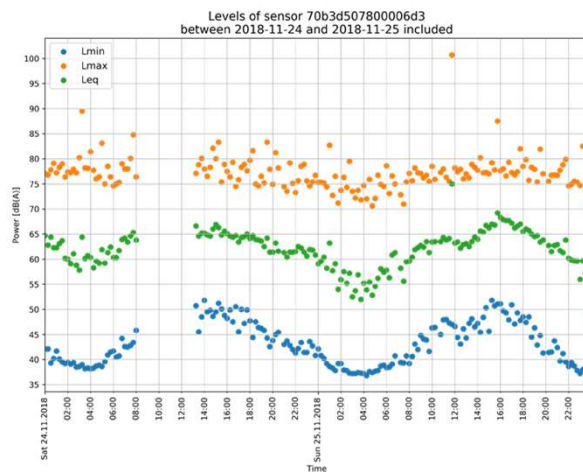
The Orbiwise noise sensors

Levels of sensor 70b3d507800006cf
between 2019-01-14 and 2019-01-15 included



The Orbiwise noise sensors

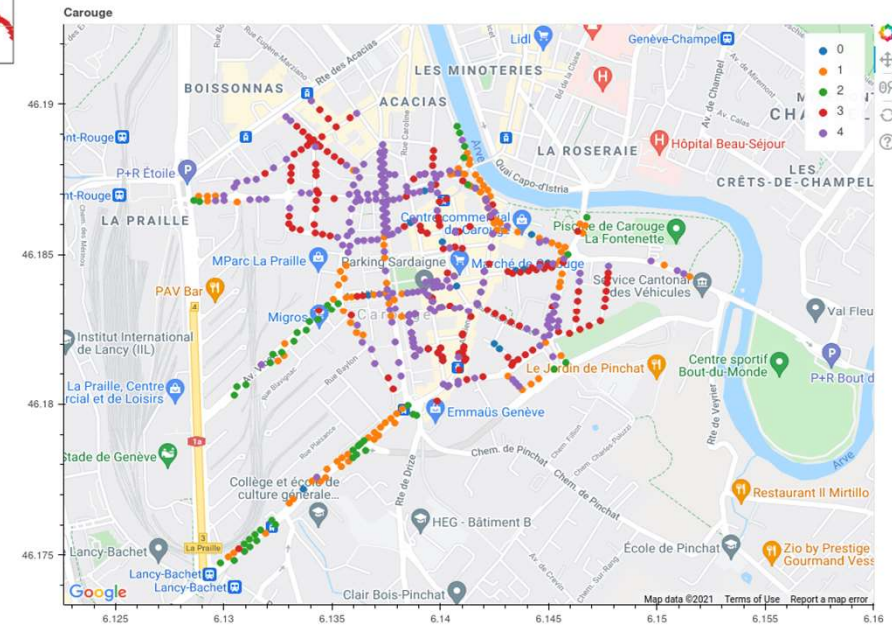
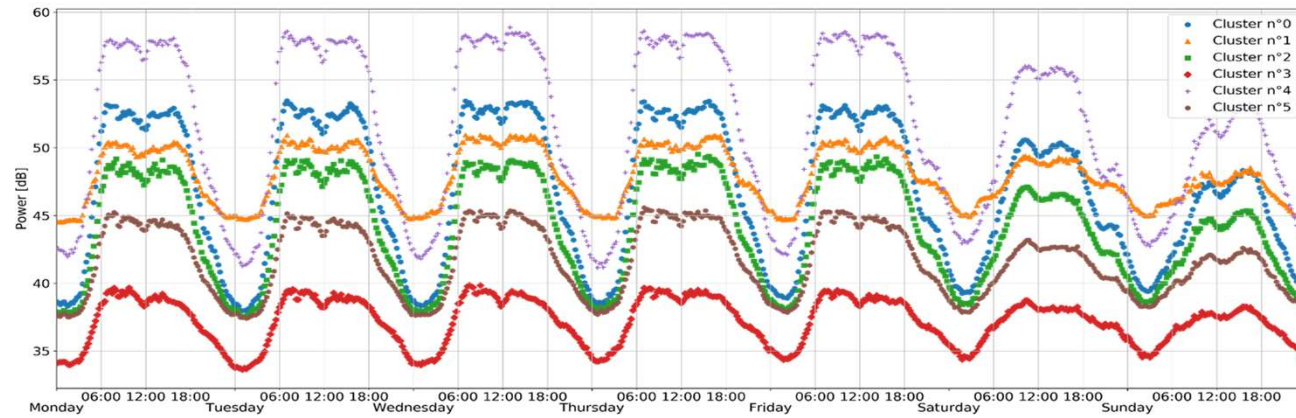
- The goal is to assess the “trustworthiness” of the retrieved data in order to deduce their reliability.
- We must deal with:
 - Misbehaving sensors
 - Network issues



Signature

- A signature represents the context of a sensor
- A context is defined by:
 - Environmental setting
 - Road configuration
 - Acoustic configurations of the streets
 - Height placement
 - Rush hours
 - Weekends vs. working days
 - Summer vs. Winter
 - etc.
- Noise sensors in a same “context” should provide data matching a similar signature (look, pattern).
- Signatures must be “extracted” from “clean” data ? We need to filter data
- Extraction of signatures relies on clustering techniques

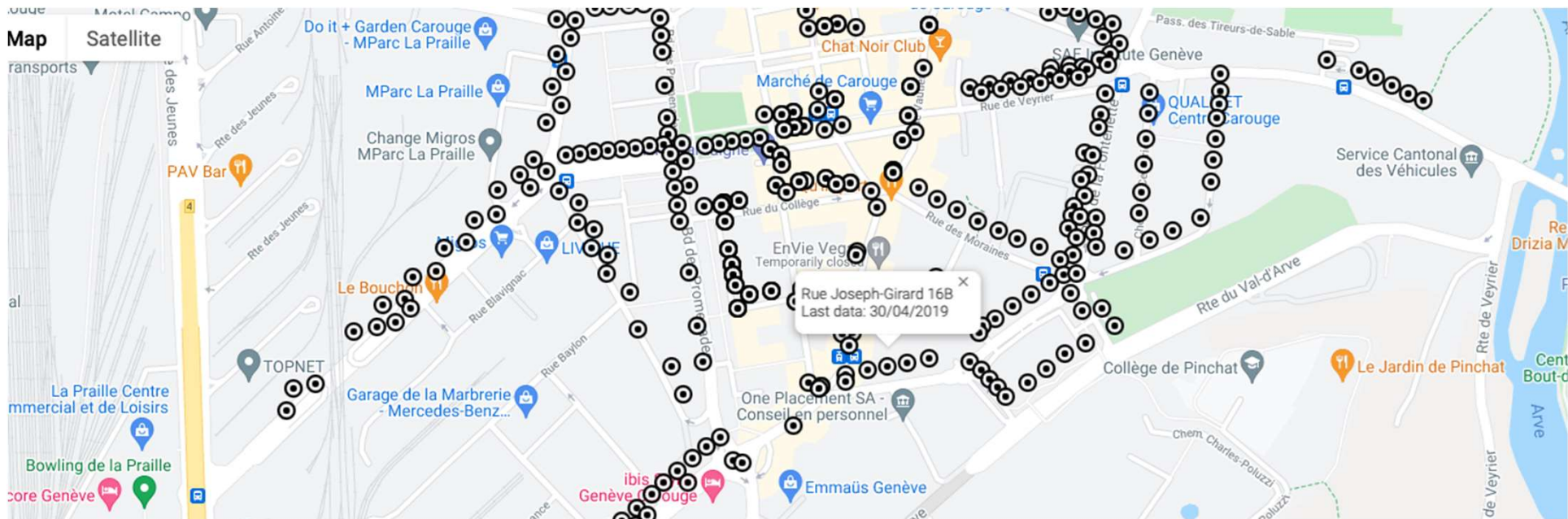
Signature



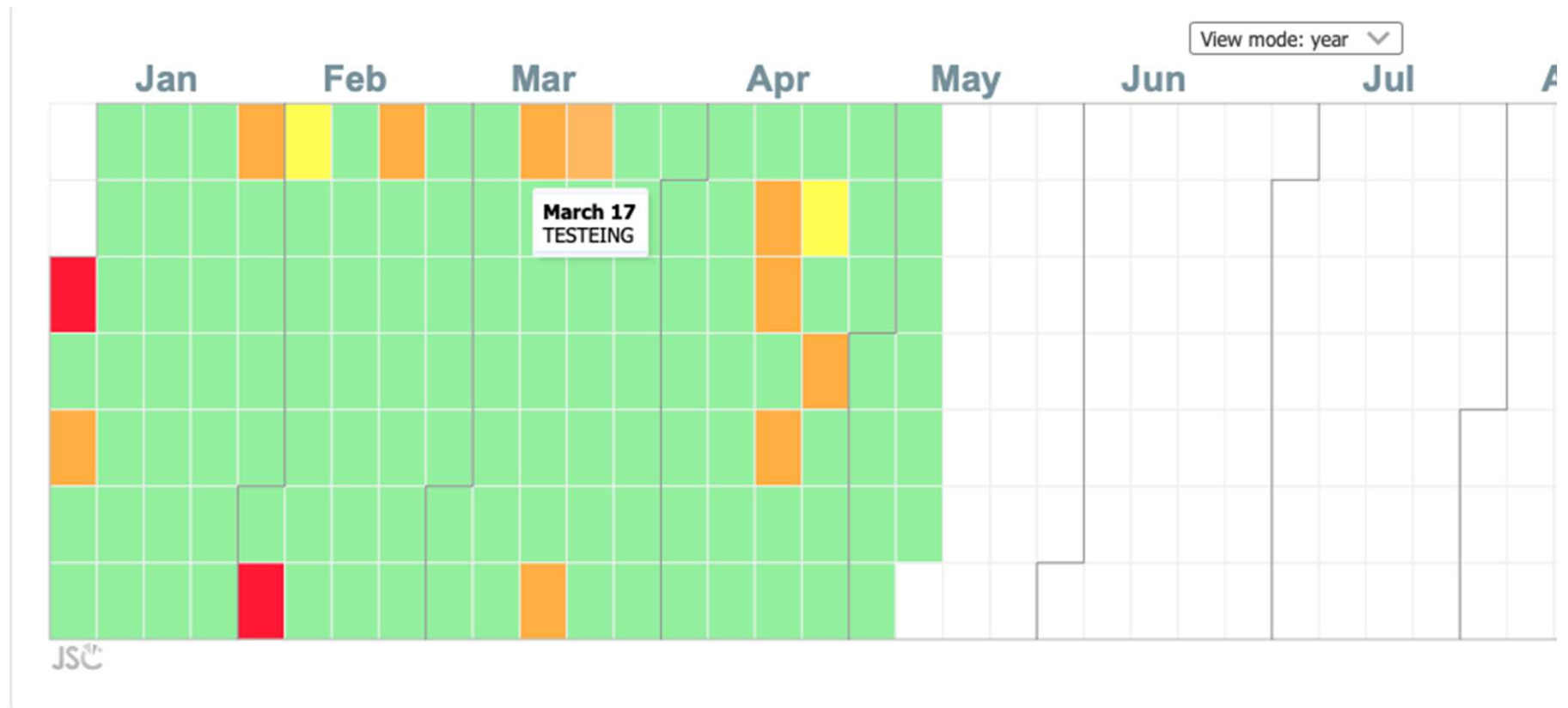
Filtering Interface

- The Filtering interface is used to support the user in the filtering phase: which data should be considered for clustering?
- The filtering interface is assumed to be a Design Support System

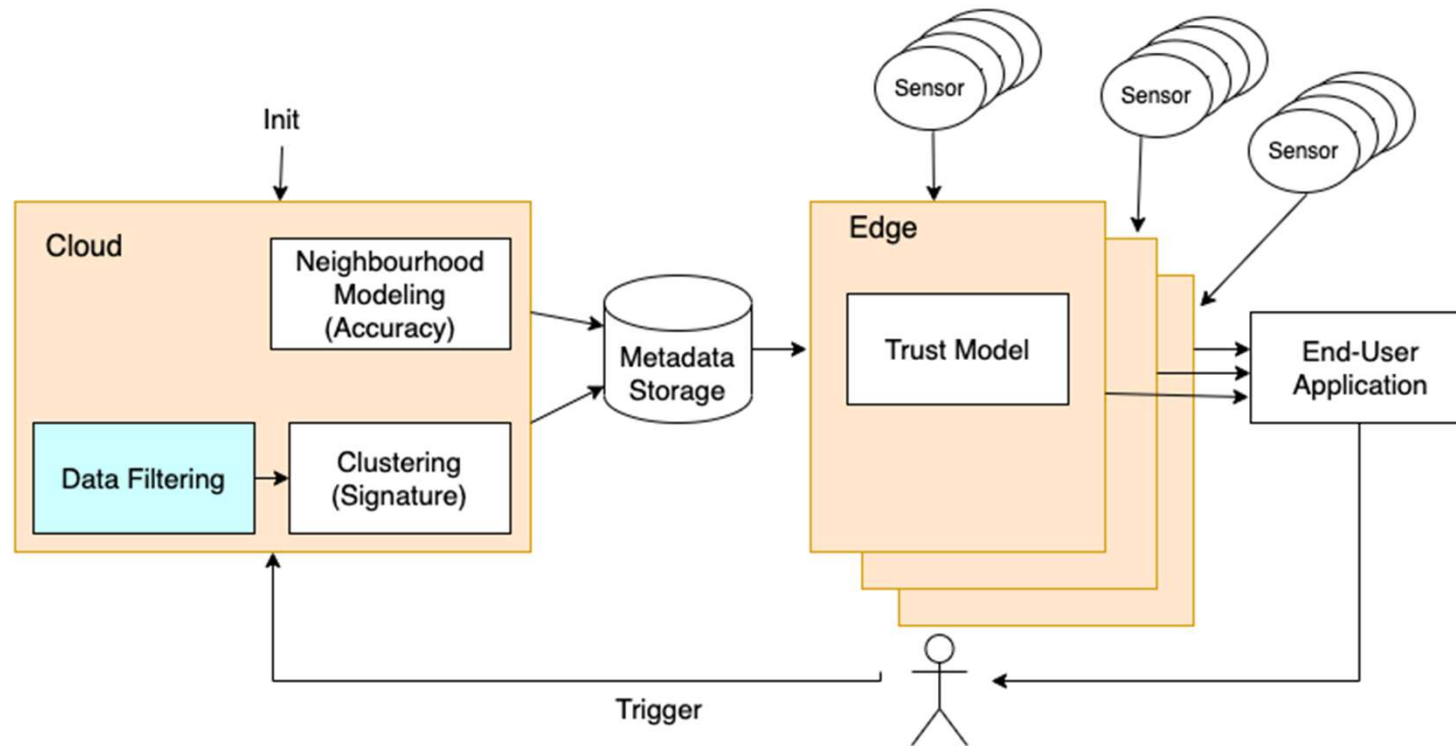
Filtering Interface



Filtering Interface



VM Deployment



Filtering Interface deployment

- Due to fairly high processing power needed, for an almost real-time delivery, there is a need to use large instances - which are costly
- Another alternative is a *Function as a Service* approach
- We used AWS Lambda

60

FaaS deployment

