CloudButton

WP3 - Serverless Compute Engine for Big Data

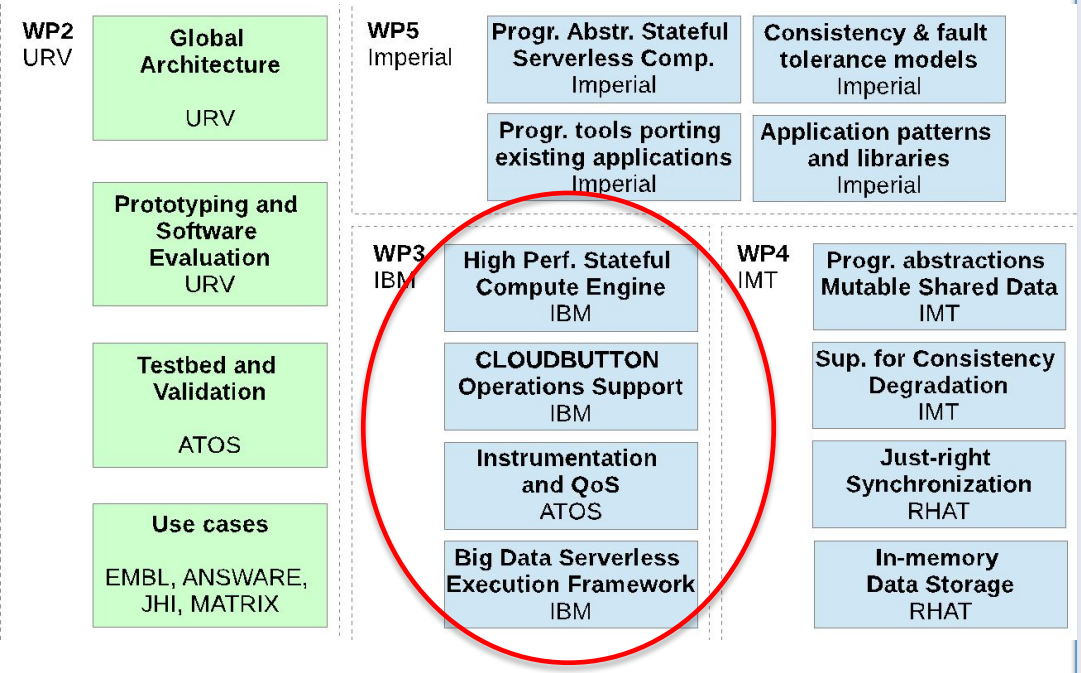
Gil Vernik

IBM Research

BIG DATA VALUE
PUBLIC-PRIVATE PARTNERSHIP

Overview of WP3

Structure - CloudButton



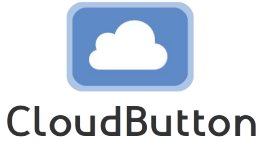
Task 3.4 - Big Data Serverless Execution Framework
Integration between applications and FaaS engines through efficient handling of fault tolerance, shuffling, caching, Big Data partition discovery, and integrations with storage high performance stateful FaaS engine

Task 3.1- High Performance Stateful FaaS Engine
Research and development of FaaS platform extensions to support stateful and highly performant execution of serverless tasks. Research and development of new techniques to optimize a tradeoff between the infrastructure utilization and performance

Task 3.2 - CloudButton Operations Support
Explore cost-effectiveness tradeoffs involved with applying serverless computing model for heterogeneous Big Data analytics. Explore relaxation of the rigid resource constraints to simplify development experience and allow for higher utilization of the infrastructure resulting in better cost-efficiency.

Task 3.3 - Instrumentation and QoS
The goal of this task is to instrument containers that host serverless functions. Instrumentation will inject into containers specific probes that will enrich available monitoring information for serverless functions.

CloudButton



- CloudButton project remains on the right “wave” and is perfectly aligned with the market direction
  - Serverless remains to be the hot topic
  - Grow beyound traditional FaaS and now used for broad range of workloads and use cases
- We developed Lithops, an open source framework that is the main part of WP3 and also served as a “glue” for other WPs developed in the project
- Lithops used in various production use cases, also beyond the CloudButton

Overall motivation and some of the goals

- Many workloads need dynamic resources during the execution phase, that are not known prior the execution started
- We should treat serverless not just as FaaS, but rather a pattern where workload deployed without manual creation of the cluster or resources
- The easy move to serverless
- Serverless for more use cases
- Cost efficiency
- Avoid vendor lock-in

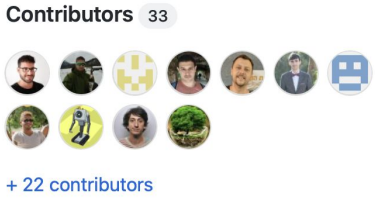
WP3 – Deep Dive

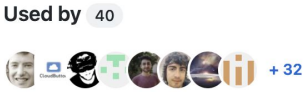
Lithops framework

- **Lithops** is a novel Python framework developed part of CloudButton Project
- Designed to scale Python applications at massive scale against major cloud providers
- Open source http://lithops.cloud
- Mature . Stable. Production ready
- Many blogs, publications, talks

Serverless for more use cases

The easy move to serverless





Lithops to scale Python code and applications

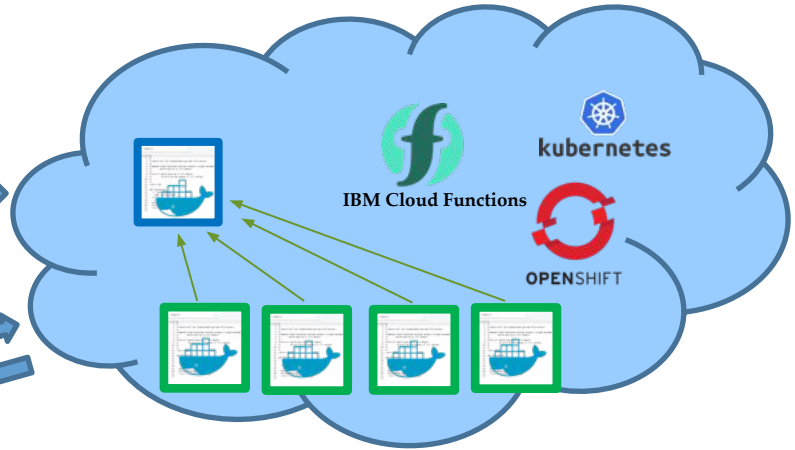
```
input data = array, COS, etc.

def my_func(x):
   //business logic
```

```
import lithops

fexec = lithops.FunctionExecutor()
fexec.map(my_func, input_data)
```

```
res =  fexec.get_result()
```

Lithops

Lithops



Lithops to scale native code or packages

```
input data = array, COS, etc.

def cmd_exec(x, bucket, ibm_cos,cmd):
    res= subprocess.call(cmd, shell = True)
```

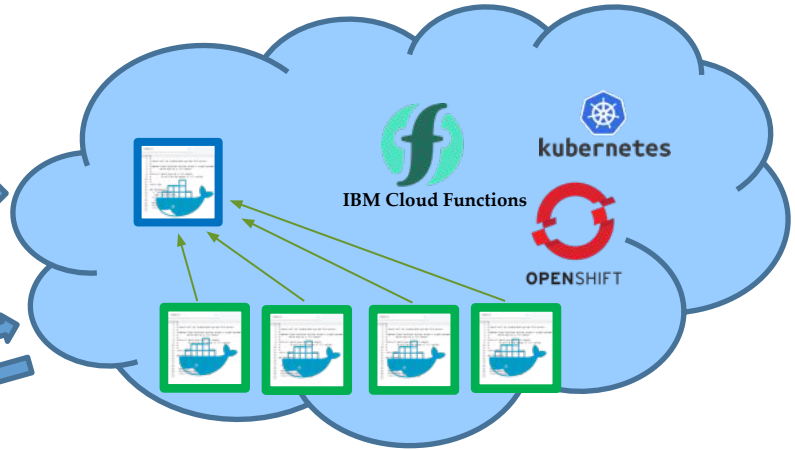
Gromacs, Protomol, Dlib, Gdal, ffmpeg, etc.

```
import lithops

fexec = lithops.FunctionExecutor()
fexec.map(cmd_exec, input_data)
```

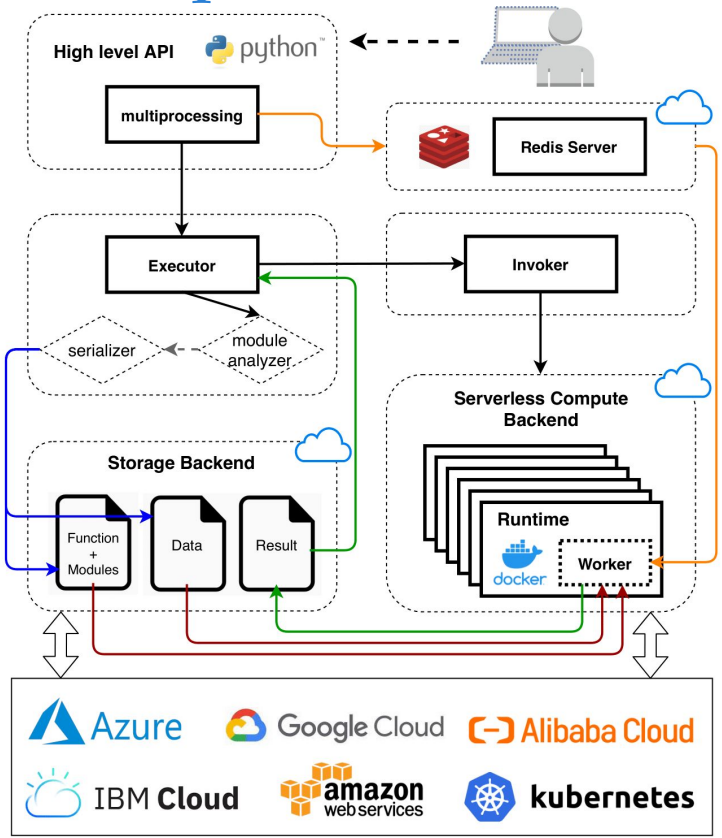
Lithops

```
res =  fexec.get_result()
```

Lithops

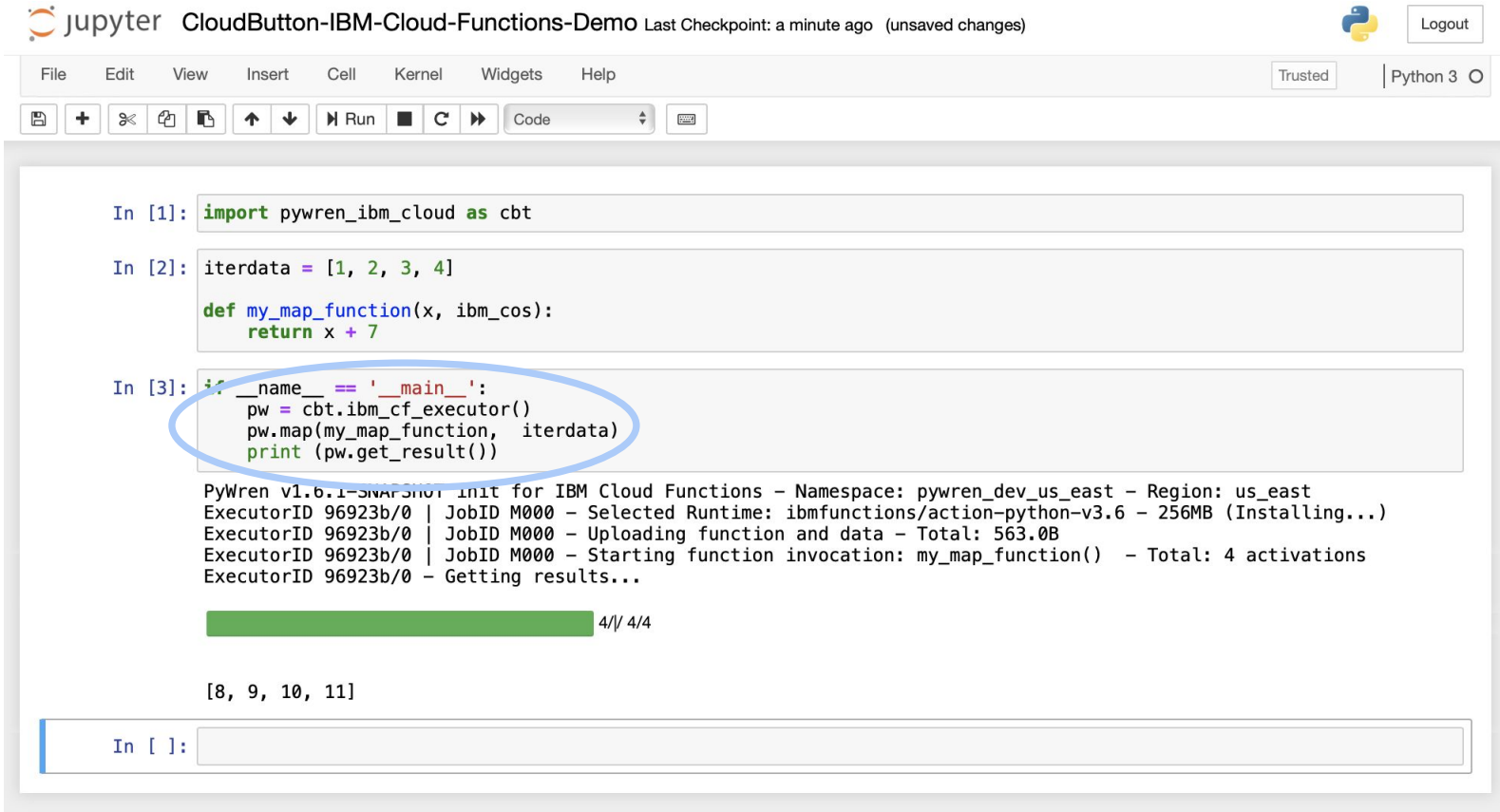


Lithops architecture



Lithops implements a similar API to the built-in python concurrent.futures library. This API is based on objects called Futures, created when Lithops spawns a function. With this Future object, it is possible to access the results and some statistics about the execution.

Python notebooks



Multi cloud with Lithops

- No vendor lock-in
- Multi cloud portability
- Can be easily extended with more backends
- Hybrid usage is supported
- An open-source framework for big data analytics and embarrassingly parallel jobs, that provides a universal API for building parallel applications in the cloud.

Futures API

```python
from lithops import FunctionExecutor

def hello(name):
    return 'Hello {}!'.format(name)

with FunctionExecutor() as fexec:
    fut = fexec.call_async(hello, 'World')
    print(fut.result())
```

Multiprocessing API

```python
from lithops.multiprocessing import Pool

def double(i):
    return i * 2

with Pool() as pool:
    result = pool.map(double, [1, 2, 3, 4, 5
    print(result)
```

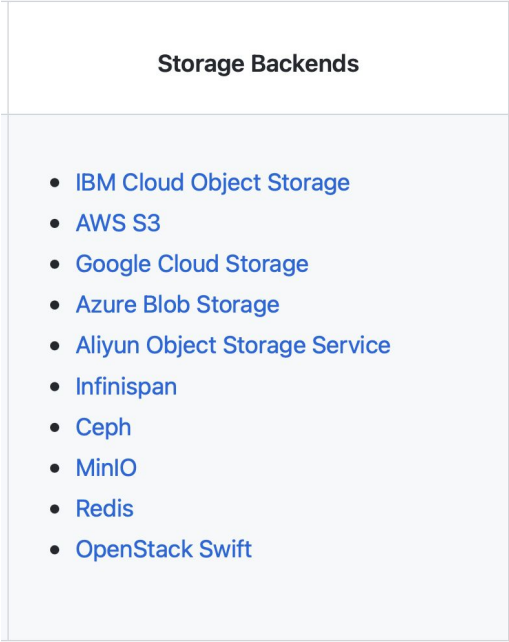
```python
from lithops import FunctionExecutor

def double(i):
    return i * 2

fexec_aws = FunctionExecutor(backend='aws_lambda')
futures_aws = fexec_aws.map(double, [1, 2, 3, 4])

fexec_ibm = FunctionExecutor(backend='ibm_cf')
futures_ibm = fexec_ibm.map(double, [5, 6, 7, 8])

print(fexec_ibm.get_result(futures_aws+futures_ibm))
```

Big Data processing with Lithops

- Lithops supports data-driven workloads for Big Data processing
- We developed advanced Big Data partitioner with chunking algorithms to support partitions of variable sizes
  - supports array,
  - URLs,
  - location in the object storage,
  - knows how to partition CSV files without breaking the line,
  - custom based partitioner, etc.

Multiple storage backends

- **Lithops can support any storage backends**
  - IBM Cloud Object Storage , Ceph, Google Storage, OpenStack Swift
  - In-memory, like Redis
  - Can be easily extended with additional storage backends via single interface
- **Lithops and Red Hat**
  - Red Hat Infinispan, Red Hat Openshift



Hybrid model

Why Hybrid?

- Hybrid approach may reduce overall costs and improve efficiency.
- Allows to apply better optimizations that will reduce number of serverless actions, thus reducing overall costs
- **Supports legacy code that can't be executed in parallel (like some of the sorts)**

Hybrid in CloudButton with Lithops

- Deploy certain flows to be executed in VMs.
- Deploy certain flows to serverless backend
- Decision happens in runtime, based on the user submit workload

EMBL use case

- **Using hybrid approach with IBM Gen2 and IBM Cloud Code Engine**

Lithops for hybrid cloud

- Public cloud is very attractive for unlimitted resources and scale
- Organizations may have sensitive data that initially must be processed on premise and can't use public cloud right away
- With Hybrid approach data processed on premise, once data is less senstive it can be moved to Cloud for the further processing
- Lithops supports various backends and can be easily used for hybrid use cases
- Can address locality by ability to run computations locally in one VM

Multi cloud hybrid experience

```
from lithops import FunctionExecutor, get_result

def double(i):
    return i*2

fexec_openshift = FunctionExecutor(backend = 'k8s')
futures_os = fexec_openshift.map(double, [1,2,3,4])

fexec_codeengine = FunctionExecutor(backend = 'code_engine')
futures_ce = fexec_openshift.map(double, [1,2,3,4])

print (get_result(futures_os+futures_ce))
```

UX remains the same for OCP, IBM Cloud, and others

Serverless without constraints

- Certain workloads (e.g., in the ML/AI or data processing space) can benefit significantly from running as much as possible on the same machine for low-latency, in-memory data sharing. To be concrete, this means up to 128 vCPUs, 1TB of memory, many TBs of local SSD disk, etc. are available per invocation and, potentially, hours of execution time if needed.
- We developed a “standalone” mode in Lithops that can provision in runtime any number of VMs, deploy the workload to those VMs and automatically dismantle them when the worklod completed (or keep them warm)

What might affect costs

- The costs of serverless usually comprises of memory, CPUs, and execution time
- Wrong memory may lead to out of memory runtime exception or consume more costs if not optimal.
- Choosing the optimal memory in runtime per analytic workload is the major key to reduce costs.
- Costs may also be affected if the invocation is stalled for any reason

Cost effective workloads

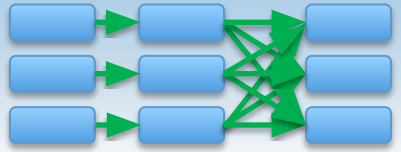
- We etimate right memory based on sampling in runtime and choose minimal memory size
- METASPACE maintain statistics from previos executions, so that it knows how much resources a job requiers based on the past executions.
- Sampling to estimate right scale to avoid timeouts
- Multi-cloud to reduce costs, improve performance and efficiency
- Lithops reusing the same authentication token across many invocations, thus greatly improving overall performance
- We developed daemons for automatic dismantle of VMs in case of user errors, etc.

Temporary Data

CloudButton engine generates status file for each invocation


```
{"exception": false, "host_submit_tstamp": 1592715321.0335932, "start_tstamp":
    1592715331.4192479, "python_version": "3.6.9", "call_id": "00000", "job_id":
    "M000", "executor_id": "0d76ac/0", "activation_id": "0b566331cc2100000",
    "type": "__end__", "function_download_time": 0.01882219, "data_download_time":
    0.10679126, "function_start_tstamp": 1592715332.0058982, "function_end_tstamp":
    1592715338.2056653, "function_exec_time": 6.19976711, "result": true,
    "output_upload_time": 0.02636218, "end_tstamp": 1592715338.2378411}
```


User map-reduce flows may need to shuffle data between stages

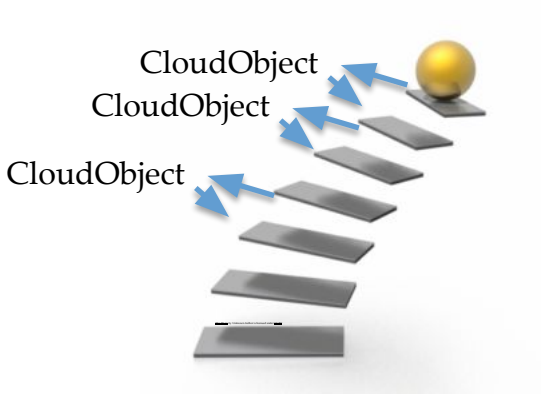


- CloudButton supports any plugable storage options for temporary data
- We extended CloudButton to use Red Hat Infinispan as it's internal storage
  - Infinispan is an in-memory, distributed, elastic NoSQL key-value datastore.
  - Infinispan may keep data in persistent storage or in memory only.

Share data and maintain state

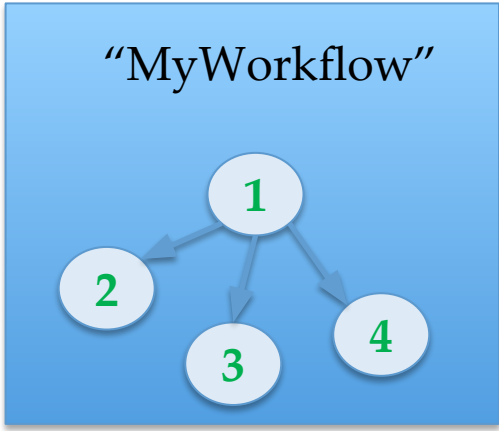
- We defined a generic user-friendly class called “CloudObject” which contains all required details to reach the storage object that it represents.

```
def my_function(data, storage):
      processed_data = …
      cloud_object = storage.put_cobject(processed_data)
      return cloud_object

And we can also load CloudObjects by:

def my_function(cloud_object, storage):
      data = storage.get_cobject(cloud_object)
```



- CloudObject can be considered as a pointer to the stored data, without the necessity of managing its storage path exactly.
- We use CloudObjects as part of an external cache system that was developed to store intermediate pipeline results and load each result when needed.

Serverless workflows



What is serverless workflow

- A serverless workflow is a DAG of computational tasks
- Executed on events such as: a new data set arrival; a clock interrupt; etc.
- With no human attendance required
- Potentially executed by different backends, such as across multiple clouds

Why we need serverless workflow

- Code reuse among data scientists and across teams
- Automation, composability
- Learning and optimization
- The engine can learn from past executions and use a pipeline structure information to optimize scheduling – important for cost-efficiency

Apache Airflow

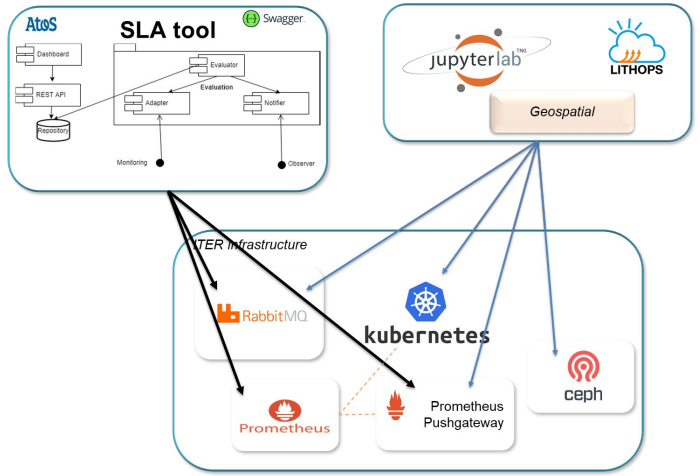
- An open source platform that provides authoring, scheduling and monitoring of workflows represented as directed acyclic graphs (DAGs). Every node of the DAG represents a task, and the edges represent dependencies between tasks, and thus, the order of execution. Apache Airflow's main objective was to be a scheduling platform for ETL workflows with a focus on the integration of different services from potentially different Cloud providers.
- The Airflow architecture, in brief, can be separated into two components. The scheduler is the component that is responsible for monitoring workflows and tasks and dispatching the corresponding tasks according to their dependencies. On the other hand, we have the workers, who are in charge of executing the logic of the assigned tasks. Airflow can scale, however, it is not designed to support all the workload on the servers themselves. Running many tasks in parallel can significantly overwhelm the load on the worker nodes and can saturate the entire system.
- On the other hand, serverless functions, and in particular the Lithops framework, provide a good opportunity to offload all this massively parallel work off the Apache Airflow cluster.
- The advantages of running a serverless workflow with Airflow are clear. On the one hand, one has the instant scalability of serverless functions available. In addition, FaaS services do not charge for idle time, so resources are used more efficiently.

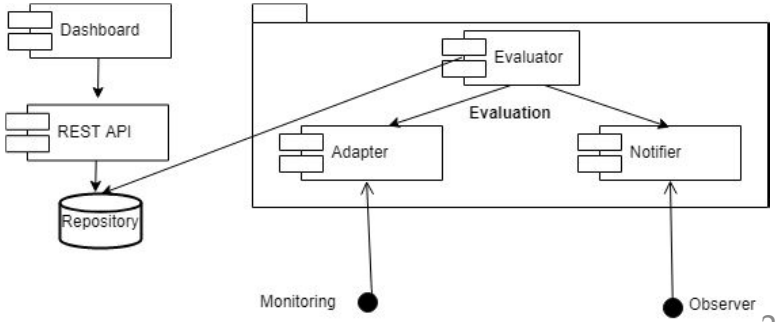
Instrumentation and QoS - goals

- Monitoring of serverless functions through specific probes and metrics
- Integration with other CloudButton components to
  - notify these components about monitoring metrics evaluations and QoS violations
  - use obtained monitoring parameters to implement elasticity and QoS mechanisms in order to provide improved stability, to prevent system oscillations, and to respond quickly to changing conditions and the time-varying requirements of analytics applications

Cloudbutton-SLA tool

- Responsible for creating, managing and evaluating SLAs based on the metrics gathered in CloudButton
- Relies on the information gathered by external monitoring tools (Lithops and Prometheus), which are used to continuously evaluate the SLAs.
- Notify other components about managed SLAs, and SLA violations related to the QoS defined for the CloudButton applications.





Instrumentation and QoS - CloudButton

- Monitoring and evaluation of Lithops and Prometheus metrics
- SLAs management
- Implementation of custom metrics for serverless functions (applications performance): “not started”, “too long
- Integration with Prometheus, RabbitMQ and Pushgateway
- Integration with Lithops via RabbitMQ
- Notifications of violations to external components
- Predicted metrics with Prometheus query functions *holt_winters()* and *predict_linear()* to anticipate the occurrence of a violation
- User Interfaces to manage and visualize the SLAs and violations: Swagger UI, Grafana

Summary and Impact

Production workloads with Lithops

- METASPACE is using Lithops in production over IBM Cloud
- Anomaly Detection (IBM Cloud) https://developer.ibm.com/apis/catalog/ai4industry--anomaly-detection-product/api/API--ai4industry--anomaly-detection-api/#connection_check
- Lithops is an official API for IBM Cloud Code Engine
- Many other customer engagement where Lithops used as a tool to deploy workloads against hybrid compute backends

BIG DATA VALUE
PUBLIC-PRIVATE PARTNERSHIP

CloudButton

UNIVERSITAT ROVIRA i VIRGILI

IBM

Imperial College London

Institut Mines-Télécom

Red Hat

EMBL

Atos

The James Hutton Institute

nsware

10 años
fundaciónMATRIX
INVESTIGACIÓN Y DESARROLLO SOSTENIBLE

THANK YOU!

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