FaaSET: A Jupyter notebook to streamline every *facet* of serverless development

Robert Cordingly, Wes Lloyd
rcording@uw.edu, wlloyd@uw.edu

April 9th 2022

School of Engineering and Technology
University Of Washington, Tacoma

5th Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf 2022)

Outline

➡ Introduction
- Supporting Tools
- FaaSET Workflow
  - Develop, Deploy, Test
  - Execute Experiments
  - Data Analysis
- Evaluation
- Conclusions

Serverless Computing

Serverless Function-as-a-Service platforms offer many appealing features:

- No infrastructure management
- Automatic scaling
- Fine grained usage-based billing models

But packaging, deploying, testing and running experiments across multiple FaaS platforms leads to unique challenges:

- Vendor lock-in requires specific tools, services, application design

jupyter + FaaSET

Outline

- Introduction
- ➡ Supporting Tools
- FaaSET Workflow
  - Develop, Deploy, Test
  - Execute Experiments
  - Data Analysis
- Evaluation
- Conclusions

Supporting Tools

SAAF

SAAF supports profiling Function-as-a-Service (FaaS) workload performance, resource utilization, and infrastructure enabling accurate performance and cost characterizations.

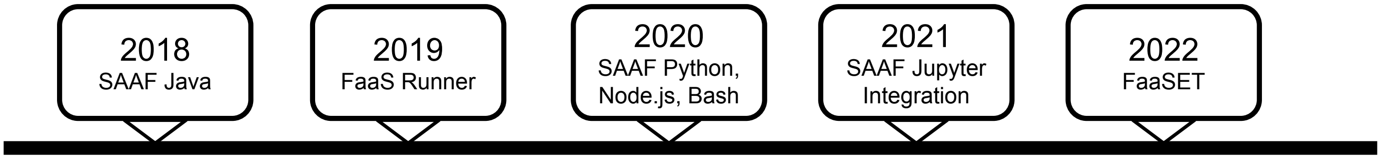
SAAF supports profiling deployments to AWS Lambda, Google Cloud Functions, IBM Cloud Functions, OpenFaaS, and Azure Functions written in Java, Python, Javascript, and BASH.

FaaS Runner

FaaS Runner is a client-side Python application used in conjunction with SAAF and the FaaSET notebook.

FaaS Runner can invoke large batches of functions synchronously, or asynchronously and orchestrate complex pipelines of functions.

Experiments are defined using functions and experiment files that explain how functions show be executed and how the results from SAAF should be processed.



2018
SAAF Java

2019
FaaS Runner

2020
SAAF Python, Node.js, Bash

2021
SAAF Jupyter Integration

2022
FaaSET

Outline

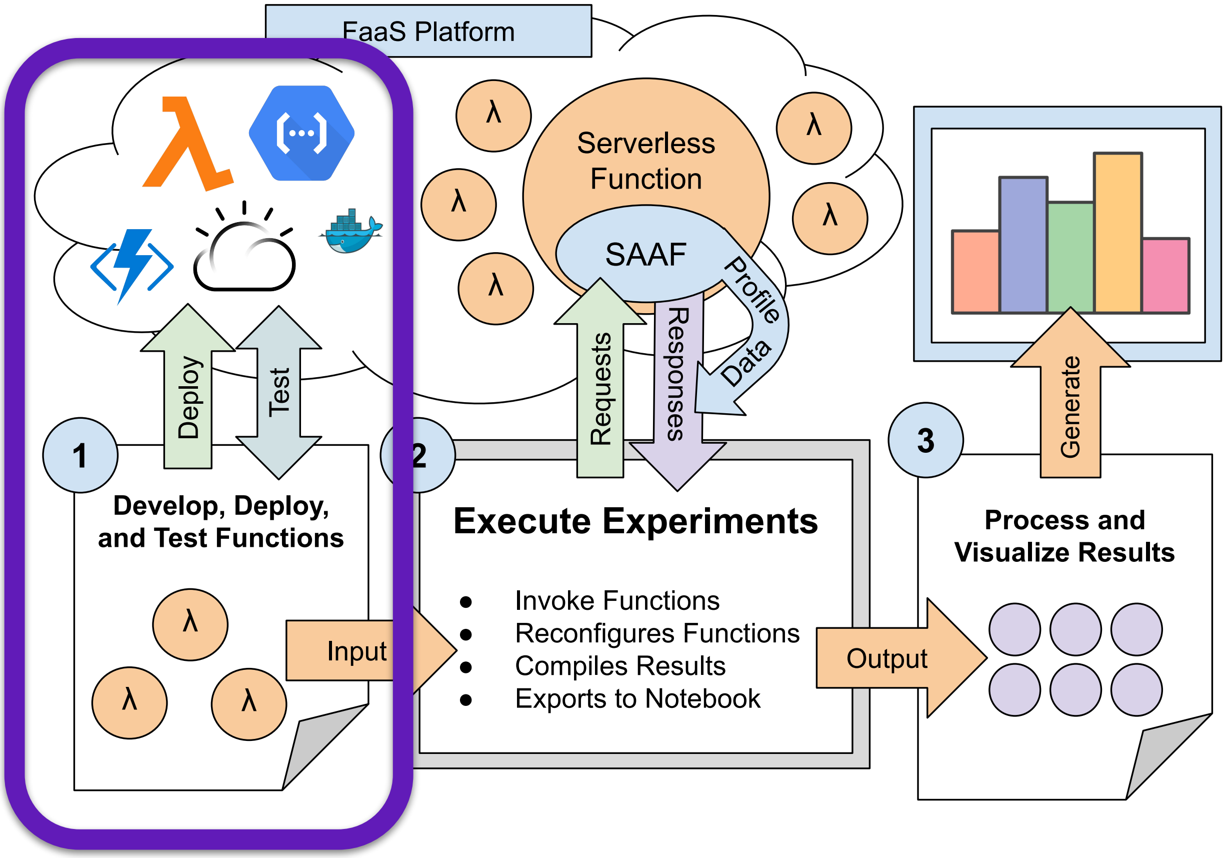
- Introduction
- Supporting Tools
- ➡ FaaSET Workflow
  - ○ Develop, Deploy, Test
  - ○ Execute Experiments
  - ○ Data Analysis
- Evaluation
- Conclusions

Function-as-a-Service Experiment Toolkit (FaaSET)

FaaSET provides aggregated tools to write, deploy, test, and run experiments on FaaS platforms all in a unified Jupyter Notebook workspace.

FaaSET supports many commercial and open source FaaS platforms:

- AWS Lambda (x86 and ARM64 with or w/o Docker Containers)
- Google Cloud Functions (Gen 1 and 2)
- IBM Cloud Functions/OpenWhisk (with or w/o Docker Containers)
- Azure Cloud Functions
- OpenFaaS



Develop, Deploy, and Test Functions

Execute Experiments

- Invoke Functions
- Reconfigures Functions
- Compiles Results
- Exports to Notebook

Process and Visualize Results

Function Development and Deployment with FaaSET

```python
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

Function Development and Deployment with FaaSET

```python
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

```python
import FaaSET
@FaaSET.cloud_function(platform="AWS", config={"memory":256})
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

Function Development and Deployment with FaaSET

```
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

```
import FaaSET
@FaaSET.cloud_function(platform="AWS", config={"memory":256})
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

Function Development and Deployment with FaaSET

```
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

```
import FaaSET
@FaaSET.cloud_function(platform="AWS", config={"memory":256})
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

```
hello_world({'name': 'Bob'}, None)
```

```
>> Deploying to AWS Lambda…
>> {"message": "Hello Bob!"}
```

Function Development and Deployment with FaaSET

```
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

```
import FaaSET
@FaaSET.cloud_function(platform="AWS", config={"memory":256})
def hello_world(request, context):
   return {"message": "Hello " + str(request["name"]) + "!"}
```

```
hello_world({'name': 'Bob'}, None)
```

```
>> Deploying to AWS Lambda…
>> {"message": "Hello Bob!"}
```

Write Once Deploy Across Multiple Platforms

```
@cloud_function(platform="AWS")
def hello_world(request, context):
   return {"message": "Hello Lambda!"}
```

```
@cloud_function(platform="IBM")
def hello_world(request, context):
   return {"message": "Hello IBM!"}
```

```
@cloud_function(platform="AWS ARM")
def hello_world(request, context):
   return {"message": "Hello Lambda!"}
```

```
@cloud_function(platform="GCF Gen2")
def hello_world(request, context):
   return {"message": "Hello Google!"}
```

```
@cloud_function(platform="GCF")
def hello_world(request, context):
   return {"message": "Hello Google!"}
```

```
@cloud_function(platform="Azure")
def hello_world(request, context):
   return {"message": "Hello Azure!"}
```

```
@cloud_function(platform="OpenFaaS")
def hello_world(request, context):
   return {"message": "Hello OpenFaaS!"}
```

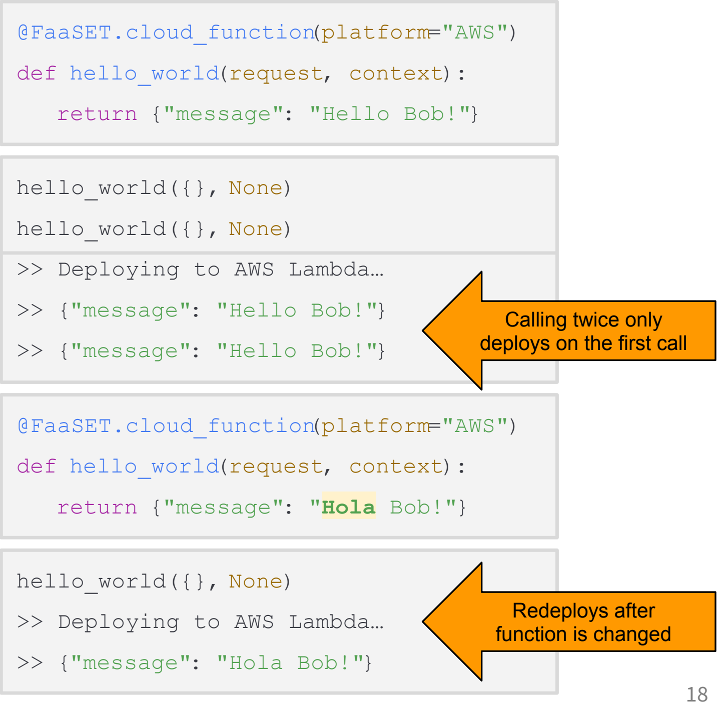
```
@cloud_function(platform="IBM Docker")
def hello_world(request, context):
   return {"message": "Hello IBM!"}
```

FaaSET Function Management Features:

- FaaSET tracks changes and only deploys when functions are modified
- Supports a simplified function development workflow using a Notebook while also allowing full control over function source code, packaging, and deployment
- Invoke existing functions already deployed in your notebooks
- Reconfigure functions on the fly without rebuilding package/containers
- Functions are automatically defined on startup allowing immediate access

FaaSET Function Management Features:

- ➡ FaaSET tracks changes and only deploys when functions are modified
- Supports a simplified function development workflow using a Notebook while also allowing full control over function source code, packaging, and deployment
- Invoke existing functions already deployed in your notebooks
- Reconfigure functions on the fly without rebuilding package/containers
- Functions are automatically defined on startup allowing immediate access



```
@FaaSET.cloud_function(platform="AWS")
def hello_world(request, context):
   return {"message": "Hello Bob!"}
```

```
hello_world({}, None)
hello_world({}, None)
```

```
>> Deploying to AWS Lambda…
>> {"message": "Hello Bob!"}
>> {"message": "Hello Bob!"}
```

Calling twice only deploys on the first call

```
@FaaSET.cloud_function(platform="AWS")
def hello_world(request, context):
   return {"message": "Hola Bob!"}
```

```
hello_world({}, None)
>> Deploying to AWS Lambda…
>> {"message": "Hola Bob!"}
```

Redeploys after function is changed

FaaSET Function Management:

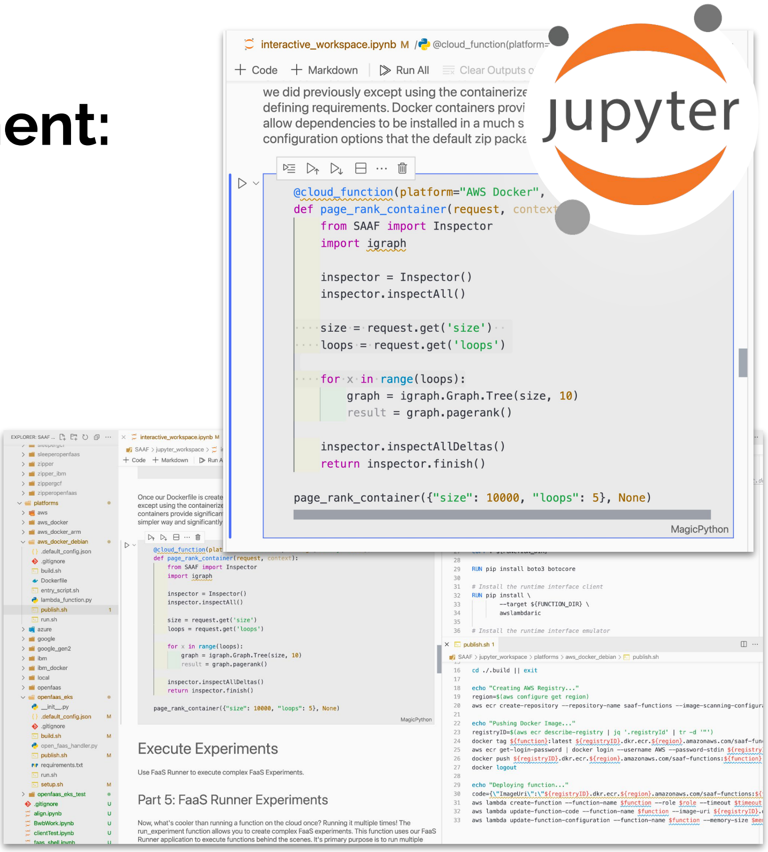
- FaaSET tracks changes and only deploys when functions are modified

➡ Supports a simplified function development workflow using a Notebook while also allowing full control over function source code, packaging, and deployment

- Invoke existing functions already deployed in your notebooks

- Reconfigure functions on the fly without rebuilding package/containers

- Functions are automatically defined on startup allowing immediate access


interactive_workspace.ipynb M / @cloud_function(platform=
+ Code + Markdown Run All Clear Outputs o
jupyter
MagicPython

we did previously except using the containerize defining requirements. Docker containers provi allow dependencies to be installed in a much s configuration options that the default zip packa

```
@cloud_function(platform="AWS Docker",
def page_rank_container(request, context
    from SAAF import Inspector
    import igraph

    inspector = Inspector()
    inspector.inspectAll()

    size = request.get('size')
    loops = request.get('loops')

    for x in range(loops):
        graph = igraph.Graph.Tree(size, 10)
        result = graph.pagerank()

    inspector.inspectAllDeltas()
    return inspector.finish()

page_rank_container({"size": 10000, "loops": 5}, None)
```

Once our Dockerfile is create except using the containerize containers provide significan simpler way and significantly

```
@cloud_function(plat
def page_rank_container(request, context):
    from SAAF import Inspector
    import igraph

    inspector = Inspector()
    inspector.inspectAll()

    size = request.get('size')
    loops = request.get('loops')

    for x in range(loops):
        graph = igraph.Graph.Tree(size, 10)
        result = graph.pagerank()

    inspector.inspectAllDeltas()
    return inspector.finish()

page_rank_container({"size": 10000, "loops": 5}, None)
```

Execute Experiments

Use FaaS Runner to execute complex FaaS Experiments.

Part 5: FaaS Runner Experiments

Now, what's cooler than running a function on the cloud once? Running it multiple times! The run_experiment function allows you to create complex FaaS experiments. This function uses our FaaS Runner application to execute functions behind the scenes. It's primary purpose is to run multiple

```
RUN pip install boto3 botocore

# Install the runtime interface client
RUN pip install \
        --target ${FUNCTION_DIR} \
        awslambdaric

# Install the runtime interface emulator
```

publish.sh 1
SAAF > jupyter_workspace > platforms > aws_docker_debian > publish.sh

```
cd ./.build || exit

echo "Creating AWS Registry..."
region=$(aws configure get region)
aws ecr create-repository --repository-name saaf-functions --image-scanning-configura

echo "Pushing Docker Image..."
registryID=$(aws ecr describe-registry | jq '.registryId' | tr -d '"')
docker tag ${function}:latest ${registryID}.dkr.ecr.${region}.amazonaws.com/saaf-fun
aws ecr get-login-password | docker login --username AWS --password-stdin ${registry
docker push ${registryID}.dkr.ecr.${region}.amazonaws.com/saaf-functions:${function}
docker logout

echo "Deploying function..."
code={\"ImageUri\":\"${registryID}.dkr.ecr.${region}.amazonaws.com/saaf-functions:${
aws lambda create-function --function-name $function --role $role --timeout $timeout
aws lambda update-function-code --function-name $function --image-uri ${registryID}.
aws lambda update-function-configuration --function-name $function --memory-size $me
```

FaaSET Function Management Features:

- FaaSET tracks changes and only deploys when functions are modified

- Supports a simplified function development workflow using a Notebook while also allowing full control over function source code, packaging, and deployment

➡

- Invoke existing functions already deployed in your notebooks

- Reconfigure functions on the fly without rebuilding package/containers

- Functions are automatically defined on startup allowing immediate access

```
@cloud_function(platform="IBM",deploy=False)
def java_function(request, context):
   pass
```

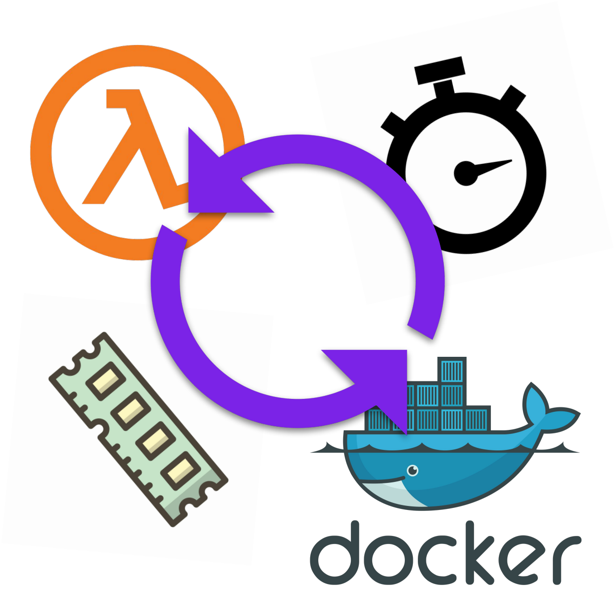
```
@cloud_function(platform="Azure",deploy=False)
def nlp_pipeline(request, context):
   pass
```

```
@cloud_function(platform="GCF",deploy=False)
def imageprocessor(request, context):
   pass
```

```
@cloud_function(platform="OpenFaaS",deploy=F…
def node_info(request, context):
   pass
```

FaaSET Function Management Features:

- FaaSET tracks changes and only deploys when functions are modified
- Supports a simplified function development workflow using a Notebook while also allowing full control over function source code, packaging, and deployment
- Invoke existing functions already deployed in your notebooks
- Reconfigure functions on the fly without rebuilding package/containers
- Functions are automatically defined on startup allowing immediate access

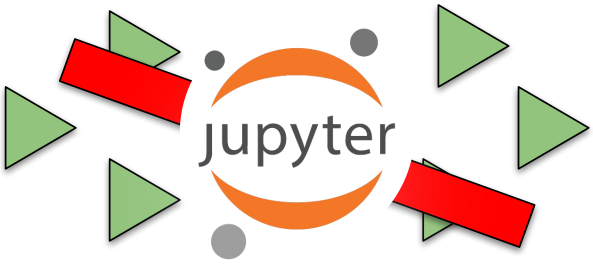


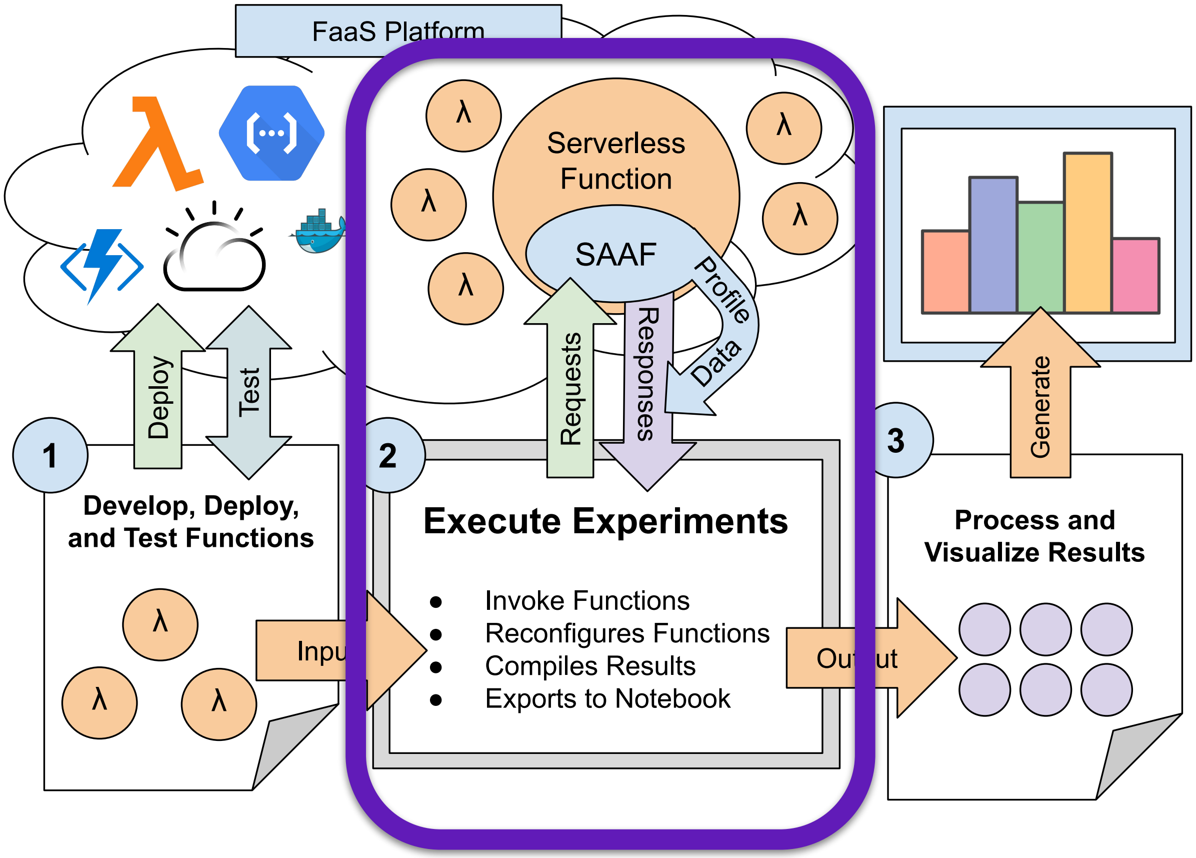
FaaSET Function Management Features:

- FaaSET tracks changes and only deploys when functions are modified
- Supports a simplified function development workflow using a Notebook while also allowing full control over function source code, packaging, and deployment
- Invoke existing functions already deployed in your notebooks
- Reconfigure functions on the fly without rebuilding package/containers
- Functions are automatically defined on startup allowing immediate access

```
import FaaSET
>> Loading platforms…
>> Platforms: AWS, GCF, IBM, Azure, AWS Docker…
>> Loading functions…
>> Functions: hello_world, nlp_pipeline, node_info…
```

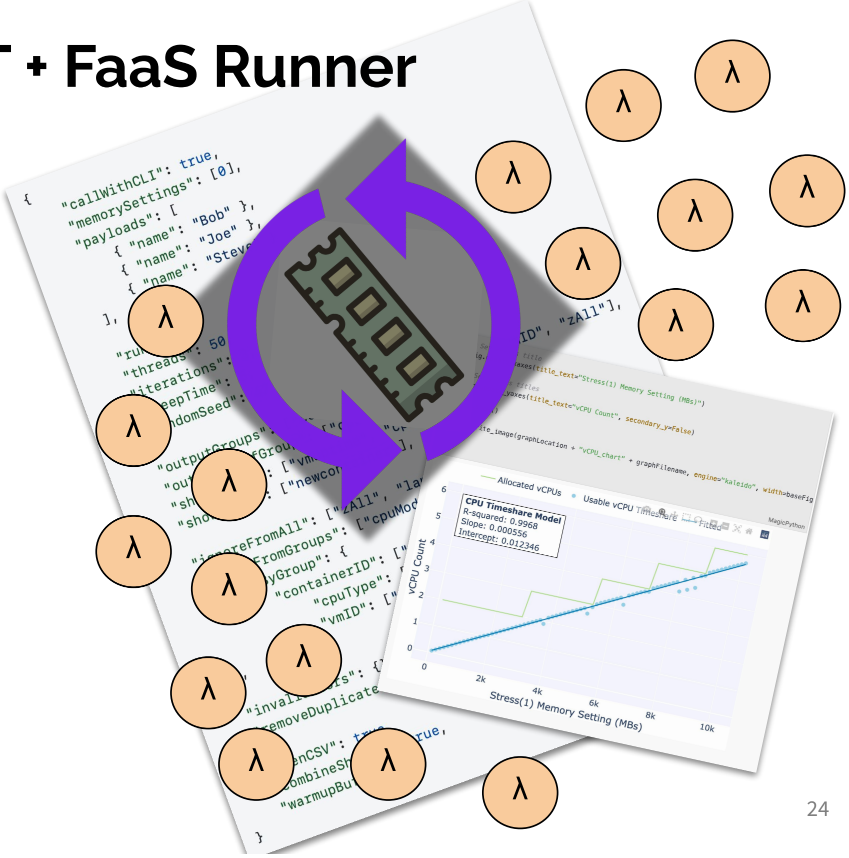
```
FaaSET.hello_world({}, None)
>> {"message": "Hola Bob!"}
```

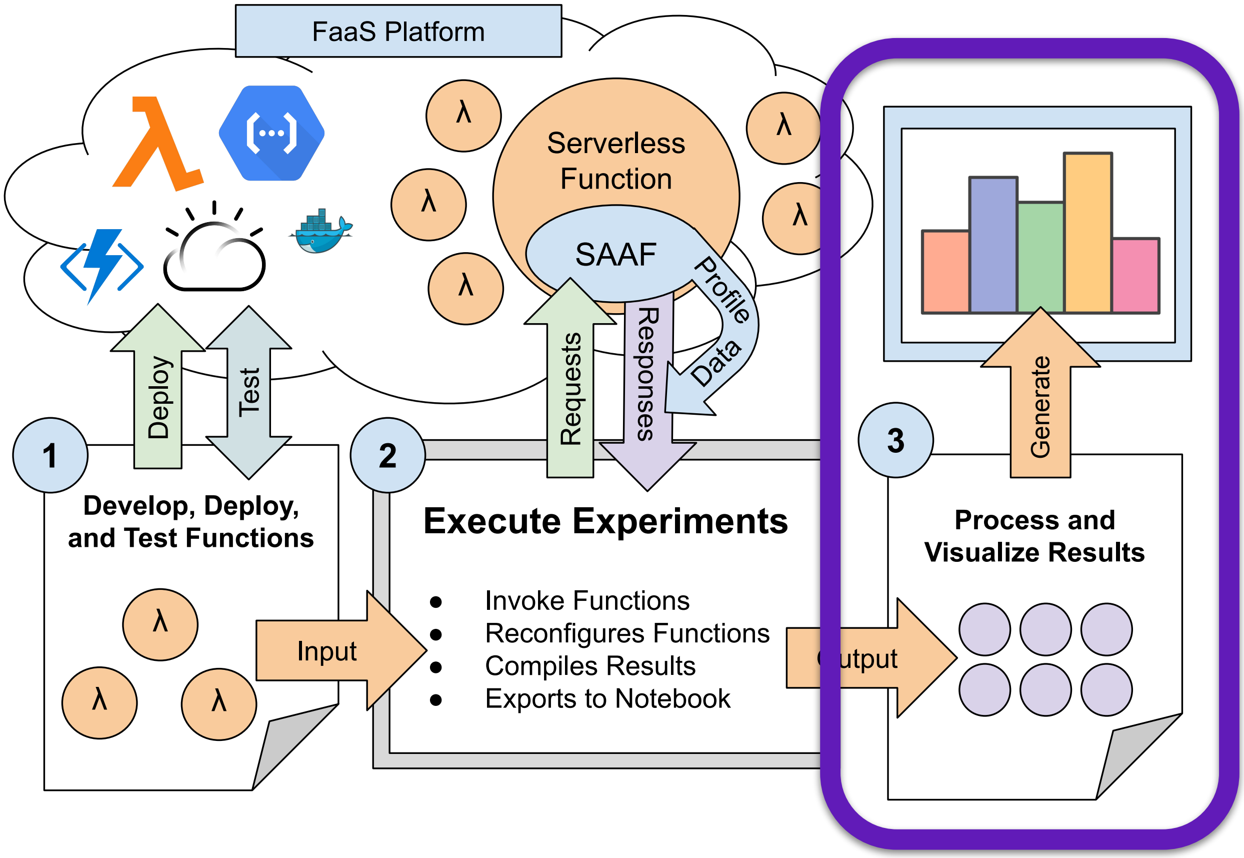




Experiments with FaaSET + FaaS Runner

- FaaSET includes the FaaS Runner tool
- Define experiment parameters such as number of runs, threads, payloads, and call order
- Utilize FaaSET's reconfiguration tool to automate complex experiments
- Results are imported into the Notebook as a Pandas dataframes





FaaS Platform

Serverless Function

SAAF

Profile Data

Requests

Responses

Deploy

Test

1

Develop, Deploy, and Test Functions

Input

2

Execute Experiments

- Invoke Functions
- Reconfigures Functions
- Compiles Results
- Exports to Notebook

Output

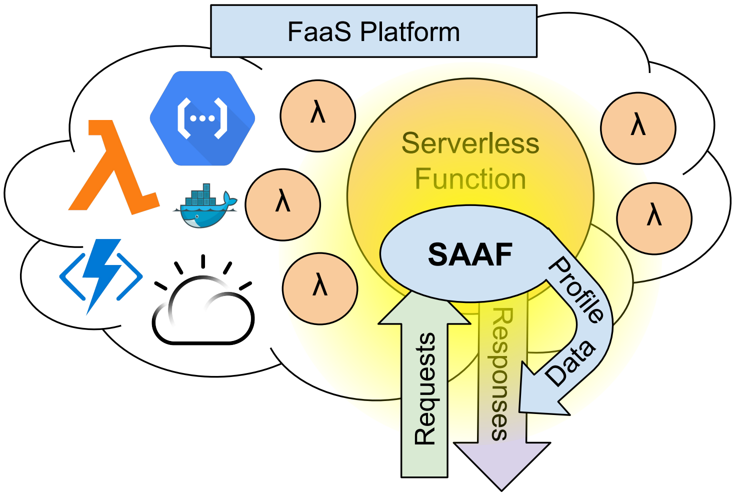
3

Process and Visualize Results

Generate

Data Analysis with FaaSET + SAAF

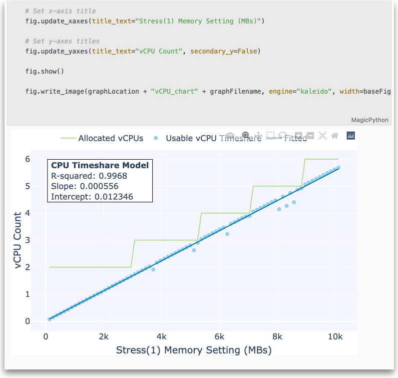
- FaaSET integrates the Serverless Application Analytics Framework
    - SAAF is placed in the deployment package of the function
    - Collects information about the host infrastructure, resource utilization metrics, and FaaS platform
- Combining FaaSET and SAAF improves accessibility of observations into FaaS platforms from a Jupyter Notebook:
    - Tenancy, warm/cold infrastructure, latency, round-trip time, and more



FaaSET

Data Analysis in FaaSET

- Since FaaSET is designed to be used inside Jupyter Notebooks, existing libraries can be used for data analysis:
  - Numpy
  - Pandas
  - Matplotlib
  - Plotly
  - Scikit-learn
  - Scipy
- FaaS Runner experiments directly output results in Pandas dataframes

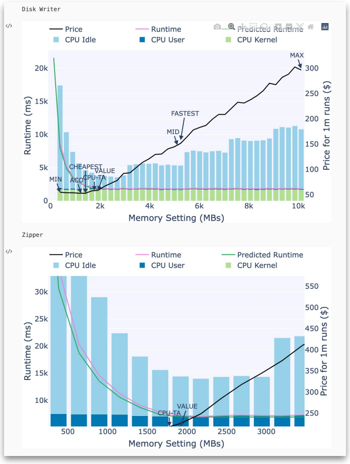


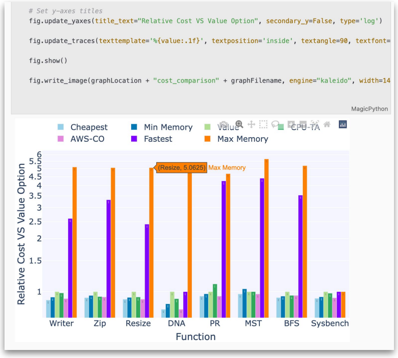
```
# Set x-axis title
fig.update_xaxes(title_text="Stress(1) Memory Setting (MBs)")

# Set y-axes titles
fig.update_yaxes(title_text="vCPU Count", secondary_y=False)

fig.show()

fig.write_image(graphLocation + "vCPU_chart" + graphFilename, engine="kaleido", width=baseFig
```



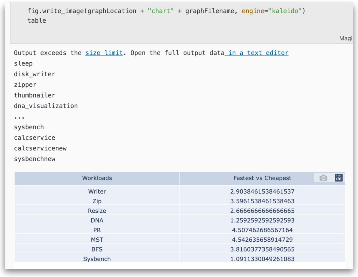


```
# Set y-axes titles
fig.update_yaxes(title_text="Relative Cost VS Value Option", secondary_y=False, type='log')

fig.update_traces(texttemplate='%{value:.1f}', textposition='inside', textangle=90, textfont=

fig.show()

fig.write_image(graphLocation + "cost_comparison" + graphFilename, engine="kaleido", width=14
```



```
fig.write_image(graphLocation + "chart" + graphFilename, engine="kaleido")
table
```

```
Output exceeds the size limit. Open the full output data in a text editor
sleep
disk_writer
zipper
thumbnailer
dna_visualization
...
sysbench
calcservice
calcservicenew
sysbenchnew
```

Workloads	Fastest vs Cheapest
Writer	2.9038461538461537
Zip	3.5961538461538463
Resize	2.6666666666666665
DNA	1.2592592592592593
PR	4.507462686567164
MST	4.542635658914729
BFS	3.8160377358490565
Sysbench	1.0911330049261083

Outline

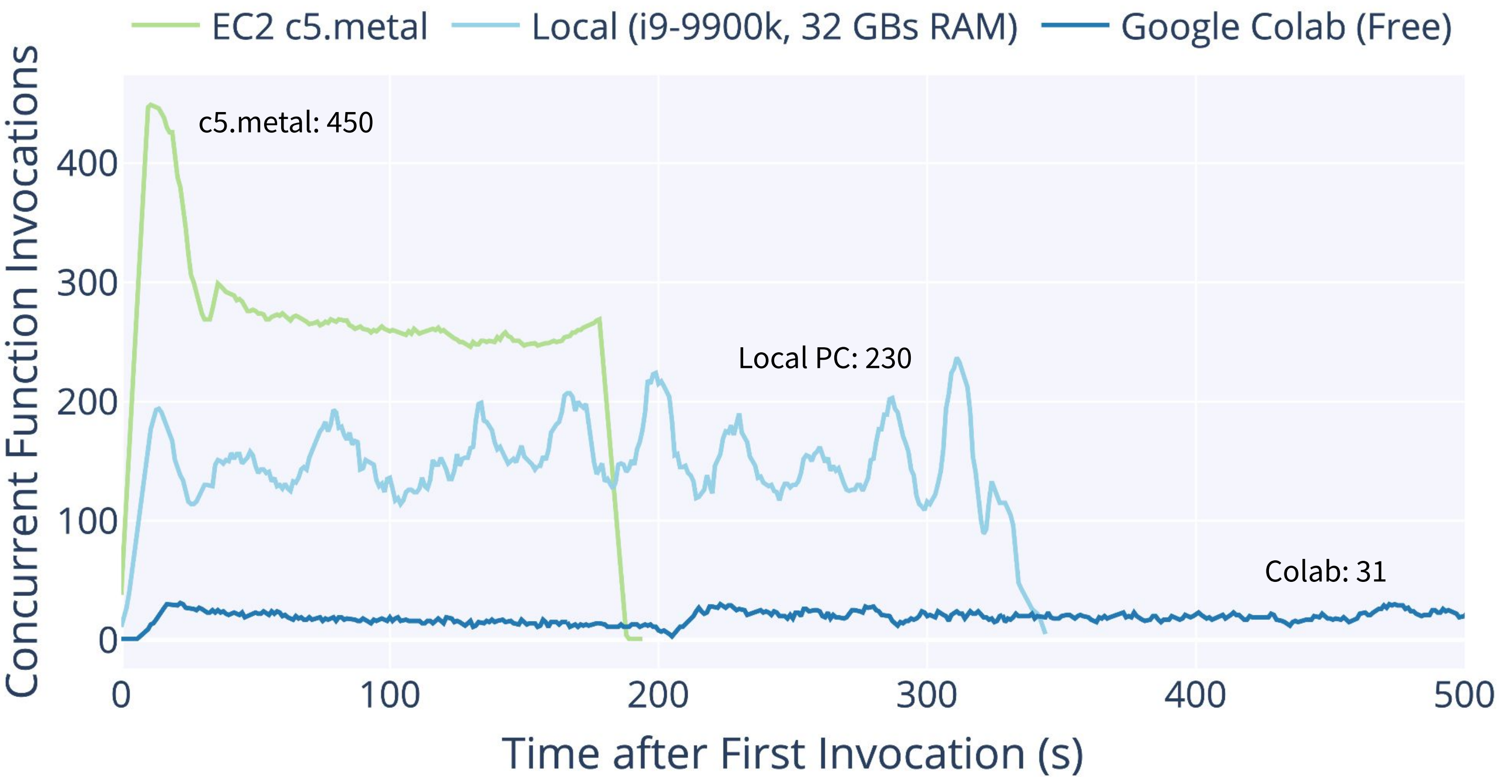
- Introduction
- Supporting Tools
- FaaSET Workflow
  - Develop, Deploy, Test
  - Execute Experiments
  - Data Analysis
- Evaluation
- Conclusions

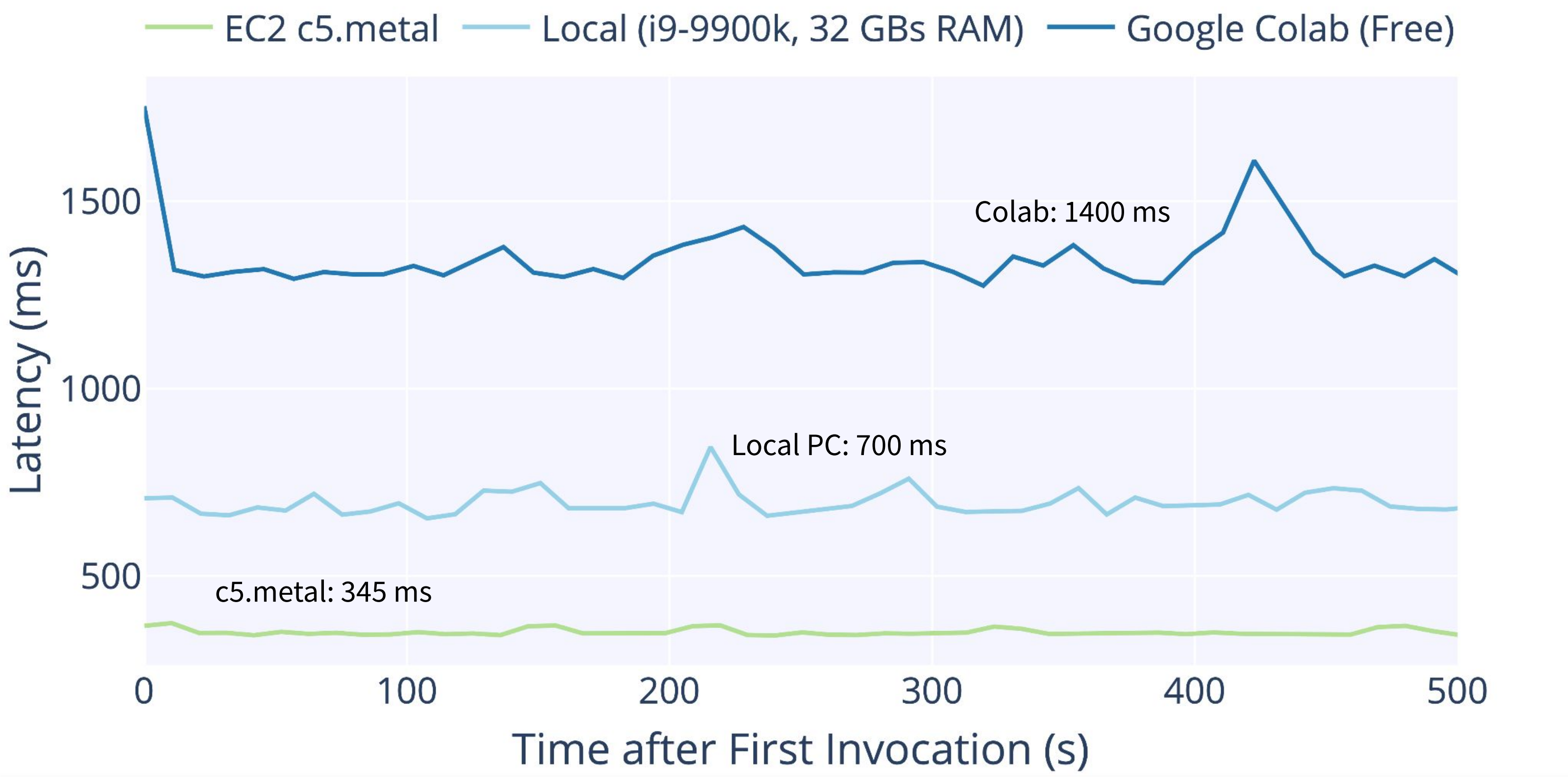
Hosting FaaS Experiments

- To run experiments on FaaS platforms a host is required to invoke the functions. This can be a local PC, powerful cloud virtual machine, or Jupyter Notebook-as-a-Service platforms (e.g. Google Colaboratory).

- We compared the performance of running 1,000 concurrent function invocations on AWS Lambda using a local PC (i9-9900k CPU, 1Gbps Network), powerful EC2 instance (c5.metal), and the free tier of Google Colab.

Host	CPU	vCPUs	Memory	Network
PC	i9-9900k @ 3.6 GHz	10	32 GBs	~1 Gbps
c5.metal	Xeon 8275L @ 3 GHz	96	192 GBs	25 Gbps
Colab	Xeon @ 2.2 Ghz	2	12 GBs	Unknown





Outline

- Introduction
- Supporting Tools
- FaaSET Workflow
  - Develop, Deploy, Test
  - Execute Experiments
  - Data Analysis
- Evaluation

➡ Conclusions

Conclusions

- FaaSET provides many features for deploying, testing, and running experiments on FaaS platforms.
  - FaaSET's goal is to provide a streamlined development environment for developers or researchers running experiments on FaaS platforms.

- While Google Colaboratory has the worse performance, it is the easiest to set up, is free, and has useful collaboration features making it great for small experiments:
  Try FaaSET by visiting: https://bit.ly/3DNVeOE

  Note: FaaSET on Google Colaboratory only supports AWS Lambda

Thank You!

Get FaaSET on GitHub: https://github.com/wlloyduw/SAAF
Try FaaSET in Google Colab: https://bit.ly/3DNVeOE

Any Questions?

Get FaaSET on GitHub: https://github.com/wlloyduw/SAAF

Try FaaSET in Google Colab: https://bit.ly/3DNVeOE

FaaSET: A Jupyter notebook to streamline every *facet* of serverless development

Robert Cordingly, Wes Lloyd

rcording@uw.edu, wlloyd@uw.edu

April 9th 2022

School of Engineering and Technology
University Of Washington, Tacoma
5th Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf 2022)

Results Summary

- The host evaluation experiment found:
    - Maximum concurrent function calls:
        - c5.metal: 450
        - i9 PC: 230
        - Google Colab: 31
    - Average total Latency:
        - c5.metal: 345 ms
        - i9 PC: 700 ms
        - Google Colab: 1400 ms