Demo: Emulating Geo-Distributed Fog Services

Moysis Symeonides*, Zacharias Georgiou*, Demetris Trihinas†, George Pallis*, Marios D. Dikaiakos*

* Department of Computer Science
University of Cyprus
{ msymeo03, zgeorg03, gpallis, mdd }@cs.ucy.ac.cy

† Department of Computer Science
University of Nicosia
trihinas.d@unic.ac.cy

I. Introduction

For more than the better parts of the last decades, we are witnessing the proliferation of IoT devices, as well as an exponential growth in the volume of data generated outside of datacenters. With the generated data at the extremes of the network and the restricted device-to-cloud bandwidth, data mitigation is becoming the major barrier of cloud-based IoT services [1]. To alleviate these challenges, Fog Computing extends the Cloud's capabilities closer to IoT devices.

However, Fog application development, testing, and performance evaluation are full of new challenges [2]. First, the adoption of Fog equipment dictates the examination of diverse alternatives with time-consuming and complicated configurations. Furthermore, resource variability found in Fog computing, caused by physical faults, bandwidth saturation, network uncertainty, energy consumption, and device mobility, harms the performance and reliability of IoT services [3]. Finally, even after all these challenges have been dealt with, compiling complex queries capable of measuring and extracting analytic insights such as QoS, energy consumption, and running monetary costs, is crucial even from the design phase, and requires the installation of external monitoring systems [4]. Consequently, practitioners utilize artificially generated workloads [5], synthetic Fog topologies [6], and Fog emulators [2] to explore application performance under various operational conditions to somehow ease the challenges mentioned above. Then, infrastructure and application performance metrics, exposed by emulation systems, can become the input of post-analysis pipelines to reveal hidden correlations between the Fog deployment and the overall performance. Though, the interconnection between emulators and analytic tools requires either exporting and transferring files from one system to another or ad-hoc integration via coding. Both prior solutions are time-consuming and error-prone due to many back-and-forth actions between emulation and analysis.

This demo presents an end-to-end system that combines Fogify [2], our interactive Fog emulator, with Jupyter, a web-based interactive tool for data analysis. Fogify helps developers by enabling the deployment of emulated Fog realms locally or on Cloud infrastructure while easing the description of Fog topologies by extending the Docker Compose to support the "fogified" model specification. With the "fogified" model, Fogify allocates resources as separated containerized processes, and establishes network connections between them. At running time, Fogify allows developers to inject ad-hoc faults, entity downtime, perform scaling actions, adjust the workload, network changes, and restrict data movement. Additionally, Jupyter along with its libraries provides a wide range of descriptive and predictive analysis methods. Last but not least, the demonstration stack is provided as a set of containerized services [7] that one can utilize with zero installation efforts.

II. Fogify Framework

We have described in detail the system design decisions of Fogify in [2]. Figure 1 depicts a high-level, abstract overview of the Fogify architecture and how it is integrated with Jupyter.

A typical workflow starts with the user editing the docker-compose file of the IoT application, and extend it to encapsulate Fogify's model. Figure 2 depicts a shortened version of demonstration's topology. The Fogify model is composed of: (i) *Fog Templates*, allowing the description of *Services*, *Nodes* and *Networks*; and (ii) the *Fog Topology*, enabling users to specify *Blueprints*. A Blueprint represents an emulated device and is a combination of a *Node*, *Service*, *Networks*, *replicas* and a *label*. Services are inherited from docker-compose while the *x-fogify* section provides all Fogify primitives. Figure 2, depicts a blueprint of a Fog node, namely *mec-node-1*, which runs the *mec-csv* service on *mec-node* device and is connected to *mec-net-1*, *mex-2-cloud-net* and *mec-2-mec-net* networks. The user should describe in that way every Fog node.

When the description is ready, the user deploys the application using the FogifySDK through a Jupyter Notebook, with the description received by the Fogify Controller. If no error is detected by the Controller, it spawns the emulated devices and creates the overlay mesh networks between them, instantiates the services, and broadcasts (any) network constraints to Fogify Agents. Specifically, the Controller translates the model specification to underlying orchestration primitives and deploys them via the Cluster Orchestrator, ensuring the instantiation of the containerized services on the emulated environment. Located on every cluster node, Fogify Agents expose an API to accept requests from the Controller, apply network QoS primitives, and monitor the emulated devices.

On a running emulated deployment, Fogify enables developers to apply *Actions* and "what-if" *Scenarios* (sequences of timestamped actions) on their IoT services, such as ad-hoc faults and topology changes. Actions and Scenarios are written by following the Fogify *Runtime Evaluation Model*. When an action or a scenario is submitted, the Fogify Controller

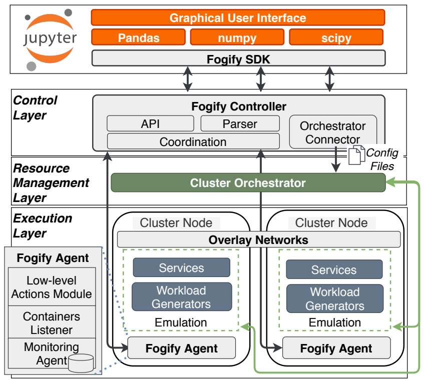
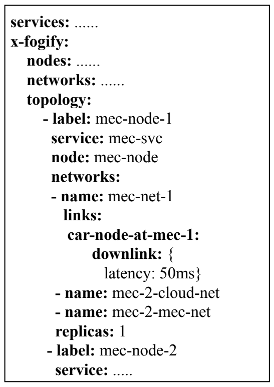
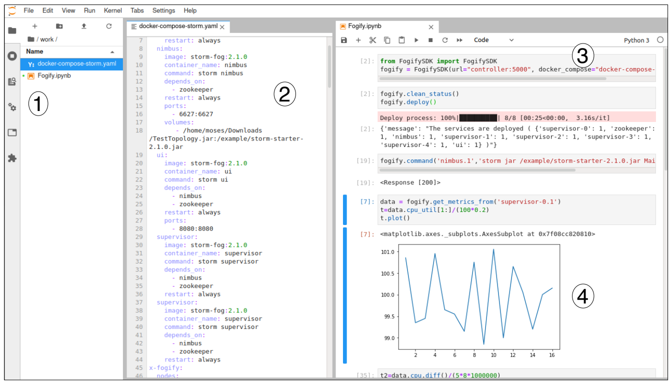


Fig. 1: Fogify Overview



```
services: ......
x-fogify:
  nodes: ......
  networks: ......
  topology:
    - label: mec-node-1
      service: mec-svc
      node: mec-node
      networks:
      - name: mec-net-1
        links:
         car-node-at-mec-1:
              downlink: {
                 latency: 50ms}
      - name: mec-2-cloud-net
      - name: mec-2-mec-net
      replicas: 1
    - label: mec-node-2
      service: .....
```

Fig. 2: Fogify Model


File Edit View Run Kernel Tabs Settings Help
/ work /
Name
docker-compose-storm.yaml
Fogify.ipynb
docker-compose-storm.yaml
Fogify.ipynb
Code
Python 3
1
2
3
4

```
    restart: always
  nimbus:
    image: storm-fog:2.1.0
    container_name: nimbus
    command: storm nimbus
    depends_on:
      - zookeeper
    restart: always
    ports:
      - 6627:6627
    volumes:
       - /home/moses/Downloads
/TestTopology.jar:/example/storm-starter-
2.1.0.jar
  ui:
    image: storm-fog:2.1.0
    container_name: ui
    command: storm ui
    depends_on:
      - nimbus
      - zookeeper
    restart: always
    ports:
      - 8080:8080
  supervisor:
    image: storm-fog:2.1.0
    container_name: supervisor
    command: storm supervisor
    depends_on:
      - nimbus
      - zookeeper
    restart: always
  supervisor:
    image: storm-fog:2.1.0
    container_name: supervisor
    command: storm supervisor
    depends_on:
      - nimbus
      - zookeeper
    restart: always
x-fogify:
  nodes:
```

```
[2]: from FogifySDK import FogifySDK
     fogify = FogifySDK(url="controller:5000", docker_compose="docker-compose-

[2]: fogify.clean_status()
     fogify.deploy()

     Deploy process: 100%|██████████| 8/8 [00:25<00:00,  3.16s/it]
[2]: {'message': "The services are deployed ( {'supervisor-0': 1, 'zookeeper':
     1, 'nimbus': 1, 'supervisor-1': 1, 'supervisor-2': 1, 'supervisor-3': 1,
     'supervisor-4': 1, 'ui': 1} )"}

[19]: fogify.command('nimbus.1','storm jar /example/storm-starter-2.1.0.jar Mai

[19]: <Response [200]>

[7]: data = fogify.get_metrics_from('supervisor-0.1')
     t=data.cpu_util[1:]/(100*0.2)
     t.plot()

[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f08cc820810>

[35]: t2=data.cpu_diff()/(5*1000000)
```

Fig. 3: Graphical User Interface

coordinates its execution with the Cluster Orchestrator and the respective Fogify Agents. Furthermore, Fogify captures performance and app-level metrics via the Fogify Agent monitoring module. All metrics are stored at the Agent's local storage and can be retrieved from the FogifySDK. In the next section, we illustrate how we exploit the FogifySDK capabilities in Jupyter to create an interactive analysis tool for emulated deployments.

III. Jupyter Interface and Integration

A computational Jupyter Notebook, is a web-based workspace that enables interactive data analysis. Specifically, users can add cells with code, which will be executed in an execution environment, namely kernel, outside of the system. When the code of a cell is processed, its results (e.g., plots), are instantly displayed. The popularity of Jupyter Notebooks contributes to a wide range of alternative web interfaces. We selected JupyterLab because it has many features found in popular IDEs, such as text highlighting and editing, while it remains focused on interactive exploratory computing. Figure 3 depicts a typical arrangement of the Fogify workspace. A users can manage Fogify's description files side-by-side with an interactive Notebook where they easily manipulate the emulation process. The left sidebar contains a file browser ①, namely a list of files created or uploaded by the user including Fog topology model and scenario files. When the user opens a file, a new tab appears with the relevant document highlighting ②, e.g. modeling files are presented with YAML highlighting. Users may use the editor to change the document's content and save it. In parallel, they interact with Fogify through a running Jupiter Notebook that is opened in a different tab ③. We pre-installed the FogifySDK library on Jupyter thus the user can (un-) deploy a Fog Topology, apply ad-hoc changes and scenarios, and, especially, retrieve runtime performance metrics. For the latter, FogifySDK stores metrics to an in-memory data structure, namely panda's dataframe, providing exploratory analysis methods that produce plots and summary statistics ④. Except of out-of-the-box plots, provided by Pandas, we extended FogifySDK with tailored functions that provide a set of plots illustrating and explaining the effects of actions and scenarios in application performance. With the wide range of analytic methods provided by FogifySDK, users extract useful insights about QoS, cost, and predictive analytics. Finally, users may integrate other libraries, like scikit-learn, to endrose their analysis with ML and AI models.

IV. Demonstration

We will demonstrate the use of Fogify in the emulation of an IoT service that is driven by real-world data [8] to showcase a scenario of a taxi-cab company that collects and analyzes location-based data from its fleet. Specifically, we will: (i) describe, via Fogify's model, a Fog application's topology, Fog devices, and networks; (ii) define a scenario where the topology will alternate dynamically; (iii) deploy the files to Fogify that will instantiate the emulation and execute the scenario; (iv) monitor every emulated node, and (v) perform examples of complex analytic tasks on the metrics mentioned before. It must be noted that both Fogify and all deployed services are open source and can run on a laptop. Thus, attendees can deploy the experiment on their laptops and perform small refinements to Fogify's configuration to see in real-time the impact on the running deployment. The target audience of this demonstration consists of three interweaving user groups:

IoT Service Developers: who wish to ease the testing of their services in geo-distributed Fog settings. With the proposed stack, developers design fog-enabled services by using description extensions to docker-compose, a familiar to them specification, and with Fogify, rapidly perform multiple tests and compare both well-functionality and performance of their services under extreme and uncertain conditions.

Academic Researchers: who wish to quickly -but thoroughly-assess novel algorithms for Fog realms. Such users want to quickly perform their analysis over various system prototypes, testing scenarios and Fog settings, to discover hidden insights without facing significant time and cost overheads of having to deploy prototypes over geo-distributed infrastructure. Ideally, they work in an environment, such as Jupyter Notebooks, with libraries, such as pandas, numpy, scipy, scikit-learn, etc.

Fog Computing Operators: who wish to assess the impact of IoT devices to their infrastructure before purchasing and deploying to production.

Acknowledgement. This work is partially supported by the EU Commission through RAINBOW 871403 (ICT-15-2019-2020) project and by the Cyprus Research and Innovation Foundation through COMPLEMENTARY/0916/0916/0171 project.

References

[1] D. Trihinas, G. Pallis, and M. Dikaiakos, “ADMin: adaptive monitoring dissemination for the internet of things,” in *IEEE INFOCOM 2017*, Atlanta, USA, May 2017.

[2] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. Dikaiakos, “Fogify: A fog computing emulation framework,” in *Proceedings of the 5th ACM/IEEE Symposium on Edge Computing*, ser. SEC ’20. New York, NY, USA: ACM, 2020.

[3] C. Powell, C. Desiniotis, and B. Dezfouli, “The fog development kit: A platform for the development and management of fog systems,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3198–3213, 2020.

[4] Z. Georgiou, M. Symeonides, D. Trihinas, G. Pallis, and M. Dikaiakos, “StreamSight: A Query-Driven Framework for Streaming Analytics in Edge Computing,” in *Proceedings of the 11th International Conference on Utility and Cloud Computing (UCC 2018)*, 2018.

[5] O. Kolosov, G. Yadgar, S. Maheshwari, and E. Soljanin, “Benchmarking in the dark: On the absence of comprehensive edge datasets,” in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, Jun. 2020. [Online]. Available: https://www.usenix.org/conference/hotedge20/presentation/kolosov

[6] T. Rausch, C. Lachner, P. A. Frangoudis, P. Raith, and S. Dustdar, “Synthesizing plausible infrastructure configurations for evaluating edge computing systems,” in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, Jun. 2020. [Online]. Available: https://www.usenix.org/conference/hotedge20/presentation/rausch

[7] “Demo repository,” https://github.com/UCY-LINC-LAB/fogify-demo.

[8] “New york yellow cab dataset,” https://on.nyc.gov/2OssELg.