Explorative study about FogFlow IoT framework

Matteo Secco (12345678)

(matteo.secco@mail.polimi.it)

-Mohammad Rahbari Solout

(10688365)

(mohammad.rahbari@mail.polimi.it)

Professor : Elisabetta Di Nitto

Teacher assistant : Mehrnoosh Askarpour

POLITECNICO
MILANO 1863

ABSTRACT

We conducted an explorative study on the FogFlow framework, in order to find out the benefits it brings in the perspective of software development and architecture. To conduct this study, we simulated a scenario of a smart factory where humans and robots work together in the same physical space. We found that FogFlow has several advantages to offer in the field of IoT development: the paradigm of fog computing brings the computation closer to where the world phenomena happens, reducing network latency. The software architecture allows to bypass some limitations that the IoT devices may have on a software level using stateful behaviour, but mitigating the disadvantages of statefulness (wrt RESTfulness) thanks to the locality of the computation and a containerized development. FogFlow emerged to be flexible, allowing changes in the infrastructure at runtime.

INTRODUCTION

Collecting massive amounts of data in everyday life poses huge challenges for the user to keep control of his/her data in terms of managing access, sharing and protection. The industrial process requires most of the tasks to be performed locally because of delay and security requirements and structured data to be communicated over the Internet to web services and the cloud. To achieve this task, middleware support is required between the industrial environment and the cloud/web services. In this context, fog is a potential middleware that can be very useful for different industrial scenarios. Fog computing can provide local processing support with acceptable latency to actuators and robots in a manufacturing industry. Additionally, as industrial big data are often unstructured, it can be trimmed and refined by the fog locally, before sending it to the cloud. One of the new platform which applies this kind of computations is FogFlow, which is a distributed data processing framework for IoT platforms to support serverless fog computing with regards to device mobility and system reliability. In the following paragraphs we took a research-based and performance-oriented look at this platform based on scenario-deriven. First of all you will see a brief description about FogFlow, afterward our scenario about a Smart Factory for having a real interaction assessment and applying it on the mentioned platform. here are some of its specifications:

FOGFLOW

FogFlow is an IoT edge computing framework to automatically orchestrate dynamic data processing flows over cloud and edges driven by context, which is the unique feature of this platform. Therefore FogFlow is capable to orchestrate data processing based on three types of contexts:

- System context: available resources which are changing over time.
- Data context: the structure and registered metadata of available data, including both raw sensor data and intermediate data.
- Usage context: high level intents defined by all different types of users (developers, service consumers, data providers) to specify what they want to achieve.

This three contexts empower FogFlow to orchestrate IoT services in a more intelligent and automatic manner. Context orchestration is the key factor to distinguish this platform from the others such as EdgeX, Azure IoT Edge, Amazon Greengrass. This aspect is enabled by the design of introducing a new layer, namely IoT Discovery, which provides a update summary of available entity data on all brokers. As compared to event or topic based orchestration, our context-based orchestration in FogFlow is more flexible and more lightweight. This is because the orchestration decisions in FogFlow can be made based on aggregated context, without reading through all involved data streams. Here you can see briefly some key features of FogFlow:

A. Context-driven orchestration mechanism
B. Serverless edge computing
C. Based on Docker
D. Dynamic data orchestrator

SCENARIO

PoliBot is a (fictional) smart factory where robots and humans work and move around in the same space. An existing IoT infrastructure exists, to enforce safety policies around the factory. In particular, two safety rules have to be verified:

- A robot must not be in Idle for more than a given threshold. If this happens, it is likely that the robot is experiencing some kind of malfunctioning, and is to be repaired.
- At any moment, robots and humans should be not closer that a fixed distance. This is to ensure that no human may accidentally get harmed by a robot.

The existing infrastructure collects position and status (Idle, Moving, Working) of the robots, and uses smart bracelets to collect the position of the humans.

One day, PoliBot decided to open other factories in different locations. Each factory comes with the same IoT infrastructure, but there is the desire from the direction to have aggregated data about the violation of the rules above available in the central office.

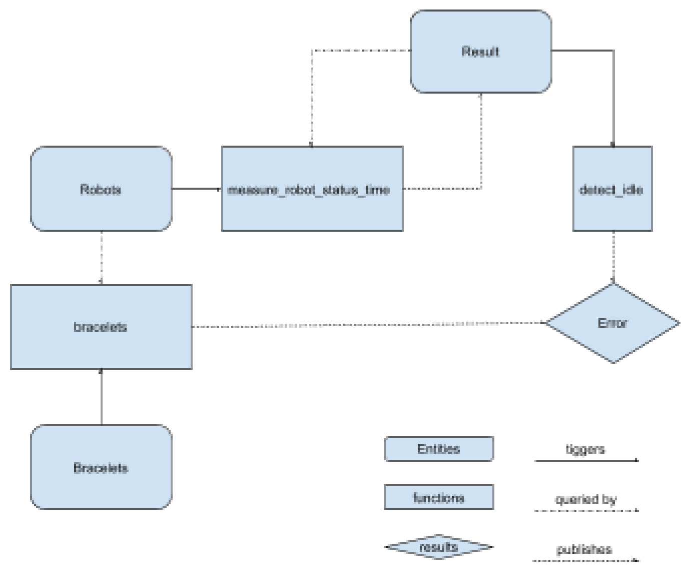
Our goal was to simulate the application of the FogFlow framework to this scenario, in order to study advantages and disadvantages of the framework itself.

IMPLEMENTATION

At the beginning of our explorative research, we wanted to develop the safety rules described in the Scenario section. We implemented 3 fog-functions, written in java using the Spring Boot framework, and available on Docker Hub:

- measure_robot_status_time
- detect_idle
- bracelets

These functions interact each other and with the environment according to the following diagram:



ata and execution flow

Functions description

Follows a summary of the behaviour of each function:

measure_robot_status_time

This function is triggered by any entity update of type Robot. Let's call robot the entity updated. On start, it queries FogFlow for an entity of type Result through the available ones having as id *Result.robot-id*. Three cases are possible:

- Such an entity doesn't exist: this means that the received update is the first update for robot. → A new Result is published, having
  result-id=robot-id,
  result-time=actual time measured by the server,
  result-last_interval=0,
  result-status=robot-status.
- Such a result entity exists, and result-status ≠ robot-status → result is updated, setting
  result-time=actual time measured by the server,
  result-last_interval=0,
  result-status=robot-status.
- Such a result entity exists, and result-status = robot-status → result is updated, setting
  result-time=actual time measured by the server,
  result-status=robot-status,
  result-last_interval=result-last_interval + actual time - result-time.

detect_idle

This function is triggered by an entity update of type Result (result). If result-last_interval is greater than a fixed threshold, the function publishes an Error entity indicating the Id of the robot that was idle for too long (the Id is extracted from the result Id), how long it has been in idle, and at which time the error was detected.

bracelets

This function is triggered by an entity update of type Bracelet (let bracelet be the entity that triggered the update). It queries FogFlow for all the available Robot entities. Then, for each of them, calculates the distance between the robot and bracelet. If the resulting distance is greater than a fixed threshold, an error is published indicating the Ids of both the robot and bracelet, and the positions of both

Deploying on FogFlow

When the code was ready, we registered the functions into the fogflow system. This operation consists of three steps:

1. Register an operator for each function
2. Link a docker image to each operator (more than one image can be linked to each operator if we want to provide different code for different operating systems or hardware architectures, but this wasn't our case)
3. Register a fog function for each operator

At the beginning we used to perform such operations by the FogFlow dashboard, but with time we preferred to automate the process using simple bash scripts performing curl requests.

Preparing the simulation

We built a simulation of a factory written in Node.js. We created it editing the powerpanel simulation available on FogFlow public repository. We started by developing the Robot entity, with movement and working capabilities. The entities work in discrete time, so at each update, the robot may perform a fixed

percentage of the current job, move by a fixed distance, change status, or do nothing (in case it is in idle). In any case, a robot will never change status while finishing a job, and when it finishes the job it will perform an attempt to change status.
The percentage completed at each update, the distance travelled, and the probability of changing status, are coded into a json file, named robot_profile.json.

After the robot has been completed we prepared the Bracelet entity, which is in fact a robot always in the "moving" status.

Running the simulation

In order to run the simulation, we first had to setup FogFlow on our machines as explained in the official documentation.

Then, we had to update the IP address of the discoveryURL property contained in the simulation_profile.json file according to the address of the machine where FogFlow was running.

Finally, we needed to register the operators and fog functions as explained in the section Deploying on FogFlow. In case we used the scripts, we needed to provide the same IP address used before.

After all these steps were done, we was able to run the simulation and see the results on the FogFlow dashboard: the resulting errors will get published as entities and appear in the System Status/Device section.

CONCLUSIONS

FogFlow emerged as an interesting research prototype. The containerized structure adopted for the fog functions brings all the advantages of containerization: portability, agility, speed, fault isolation, efficiency, ease of management.

FogFlow is language-agnostic, and any component of a development team could code in any programming language he want: all the data are shared by the NGSI protocol.

Compared to other serverless technologies, in particular with AWS Lambda, FogFlow exhibit some peculiarities: it is distributed by design, allowing the developers to bring the computation closer to where the world phenomena happens (so to get improvements in network latency), and to decentralize the information (bringing both performance and security improvements).

The framework is stateful, peculiarity that allows to bypass eventual limitations of the IoT devices by software (like we did to track for how long a robot remained in a state). This is in our opinion a good practice for the purpose of the framework: it reduces the costs of the IoT devices needed, and the tradeoff (losing the advantages of a RESTful architecture) is outbalanced by the locality of computation and containerized paradigm.

Finally, the infrastructure can be changed at runtime (adding cloud and edge nodes). The change is not perfect yet (for example, when adding a node the existing devices are not moved to that node, even if it is more convenient), but works perfectly fine in the perspective of this scenario (assuming the company wouldn't open two factories one after the other, and so devices would be "sticky"). Anyway this may be an issue in the perspective of more complex scenarios, for example a smart city.

FUTURE WORK

There are many features of FogFlow that we didn't explore but may be interesting for the scenario we analyzed. In particular, we would like to explore more about the integration with Grafana, that would allow a user-friendly data representation for the Business people, the mechanism of service topologies, which allows for on-demand interrogations about the system, and experimenting with the entity aggregation mechanisms offered by FogFlow.

RESOURCES

Fog functions

- measure_robot_status_time
- detect_idle
- bracelets

Github repository

- smart_factory

Bash scripts

- register operator
- register fog function

REFERENCES

- B. Cheng, J. Fürst, G. Solmaz, T. Sanada, "Fog Function: Serverless Fog Computing for Data Intensive IoT Services," in the proceedings of 2019 IEEE Conference on Service Computing (IEEE SCC'19) (Won the best paper award), Milan, 2019, pp.28-35
- M. Fadel Argerich, B. Cheng, J. Fuerst, "Reinforcement Learning based Orchestration for Elastic Services", in the proceedings of the 5th IEEE World Forum on Internet of Things, WF-IoT 2019, Limerick, Ireland, April 15-18, 2019, pp. 352-357
- B. Cheng, E. Kovacs, A. Kitazawa, K. Terasawa, T. Hada, M. Takeuchi, "FogFlow: Orchestrating IoT Services over Cloud and Edges", NEC Technical Journal, 2018/11
- B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa and A. Kitazawa, "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities", in IEEE Internet of Things Journal, 2017
- FogFlow official documentation