Truong, Linh

Using IoTCloudSamples as a software framework for simulations of edge computing scenarios

Published in:
Internet of Things

DOI:
10.1016/j.iot.2021.100383

Published: 01/06/2021

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY

Please cite the original version:
Truong, L. (2021). Using IoTCloudSamples as a software framework for simulations of edge computing scenarios. *Internet of Things*, *14*, Article 100383. https://doi.org/10.1016/j.iot.2021.100383

Internet of Things 14 (2021) 100383

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Using IoTCloudSamples as a software framework for simulations of edge computing scenarios

Hong-Linh Truong

Department of Computer Science, Aalto University, Finland

ARTICLE INFO

Article history:
Received 22 October 2020
Revised 19 February 2021
Accepted 22 February 2021
Available online 27 February 2021

Keywords:
Scenario-based development
Edge computing
Software framework
IoT
Edge-simulation-as-code

ABSTRACT

Realizing the potential of edge computing and networks connecting the edge and the cloud, researchers from academia and industries have increasingly developed techniques and tools for edge infrastructures and applications. This paper focuses on supporting complex edge application interactions, which span different layers and subsystems in edge-cloud environments. This paper addresses (i) diverse types of software components for emulating realistic functionality and configurations, like data transformation and service API interoperability, and (ii) techniques for connecting emulated scenarios to real edge software development and operations, and to IoT and cloud counterparts. We present IoTCloudSamples as a software framework with (i) modeling and implementation of diverse types of IoT, edge and cloud elements for complex edge scenarios, (ii) methods for constructing and steering emulations to study the interoperability across layers among different edge platforms and protocols, and (iii) extensive emulated scenarios and experiments integrated with real-world IoT and cloud services. IoTCloudSamples supports the approach of *edge-simulation-as-code* to allow the reuse and runtime steering of realistic emulation operations. We will present examples from our real-world projects concentrating on edge analytics applications.

1. Introduction

Edge computing [1,2] has increasingly attracted many researchers from academia and industries to develop novel solutions for many application domains, like connected cars, indoor shopping analytics, city transport analytics and smart factories [3–5]. An adequate edge computing system for testing and simulating realistic edge scenarios must connect IoT devices to edge infrastructures to cloud data centers through different types of networks and middleware, using a variety of software frameworks. In the literature, researchers have widely used simulation techniques to study interactions and characteristics of complex designs in the edge. Unfortunately, edge simulations tools [6–9], many based on discrete event simulations, are far from the real need from the developer. First, these simulation techniques mainly focus on network and compute resource usage and on theoretical performance and energy consumption analysis. However, a typical edge application requires many elements, including data, software components, diverse protocol supports, to name just a few. The developer must also simulate application-level data flows, service interactions and analysis algorithms, at the same time with studying network and

E-mail address: linh.truong@aalto.fi

https://doi.org/10.1016/j.iot.2021.100383

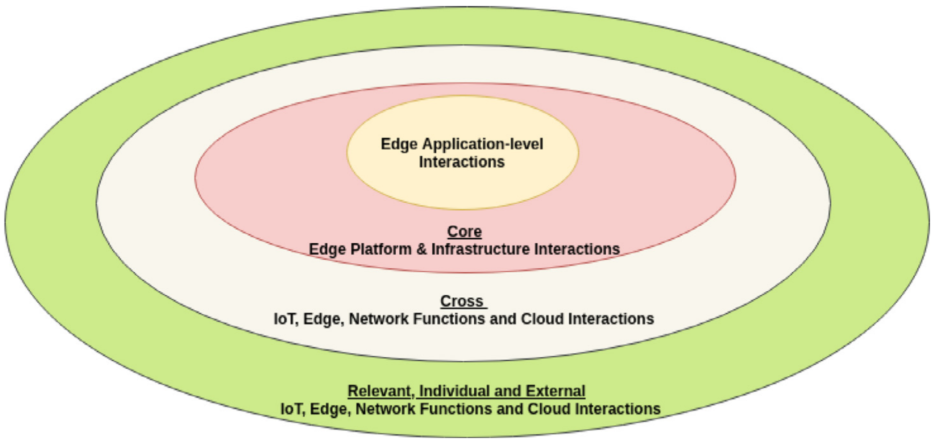

Edge Application-level Interactions
Core
Edge Platform & Infrastructure Interactions
Cross
IoT, Edge, Network Functions and Cloud Interactions
Relevant, Individual and External
IoT, Edge, Network Functions and Cloud Interactions

Fig. 1. Focused simulation activities for edge application interactions.

infrastructure behaviors. Therefore, discrete event simulation approaches alone are not adequate. Second, the scale of edge applications is large and changing. While existing simulation approaches, e.g., discrete event simulations, can bring theoretical simulation results of very scalable networks and infrastructures, they lack support for realistic settings, such as the dynamics of real containers and application-level service models in real edge software designs.

Given the state-of-the-art (see our detailed discussion in Section 5), we focus on supporting edge application developers to simulate their applications and complex interactions within the applications in connection with selected underlying edge infrastructures.[1] These simulations and tests have to be positioned in concert with IoT, network functions and cloud counterparts. As shown in Fig. 1, we focus on simulation activities for the application-level interaction first (the innermost block in Fig. 1), before expanding our work to other activities, such as core edge platform and infrastructure interactions (e.g., resource management for edge applications), cross IoT, edge, network functions and cloud interactions at the platform and infrastructural level, and then to relevant, external IoT, edge, network functions and cloud services. For such interactions, application data, software service units, application protocols, stakeholder relationships, etc., are the key aspects to be simulated.

Our goal is to provide a framework for edge application-level interactions within ensembles of IoT, Edge, Network Functions and Clouds [10]. In complex scenarios, the developers usually need to address key engineering aspects of their edge applications, such as, integration among services, interoperability, application-level performance, data transformation, and runtime deployment. Such simulation features are not available in existing discrete event simulation tools. We aim to enable new simulation designs and tests to be real-world as much as possible through the use of real artefacts and services for simulation. To this end, our approach is to enable *symbiotic simulation engineering* for edge systems and applications through *edge-simulation-as-code*. This paper presents `IoTCloudSamples` as a software framework, which includes different types of IoT, edge, network functions and cloud elements, samples of data and testing scenarios for different purposes. We make the following contributions:

- Diverse types of elements: we provide various service units and providers, covering IoT, edge, network functions and cloud functionality for simulations; they are implemented as containerized microservices.
- Symbiotic simulation techniques: our simulation is based on real-world artefacts and can be connected to real, running edge and cloud systems.
- *Edge-simulation-as-code* approach: we present various techniques and steps to build simulations through coding, following DevOps processes.
- Real world scenarios: we include many realistic simulation scenarios as case studies for broad edge interaction studies.

In this paper, we do not present some common aspects of using simulation techniques, such as, simulating system scaling and failures. The diversity of business goals and the pervasiveness of application protocols, services and interactions in different scenarios in edge computing steer our simulation features towards edge software development methods, interactions integration pipelines, data transformation, interoperabilities and service reusability. Thus, `IoTCloudSamples` utilizes service and virtualization concepts to enable key requirements from the developers, leveraging real infrastructures to test different types of interactions in combination with runtime steering of IoT, edge and cloud services. We will explain our

[1] We use the term "simulation" here because the developed applications include certain parts emulating realistic requirements. However, components in our work are either real software or mockup/simplified ones running in realistic environments. The scenarios and applications designed and built with the help of our approach can be real-world, being deployed and operational in real edge and cloud infrastructures. Nevertheless, such designs and applications are mostly not completed in the view of business functionality and are mainly for feasibility study and research. Therefore, the term "simulation" in this paper should be interpreted in this setting.

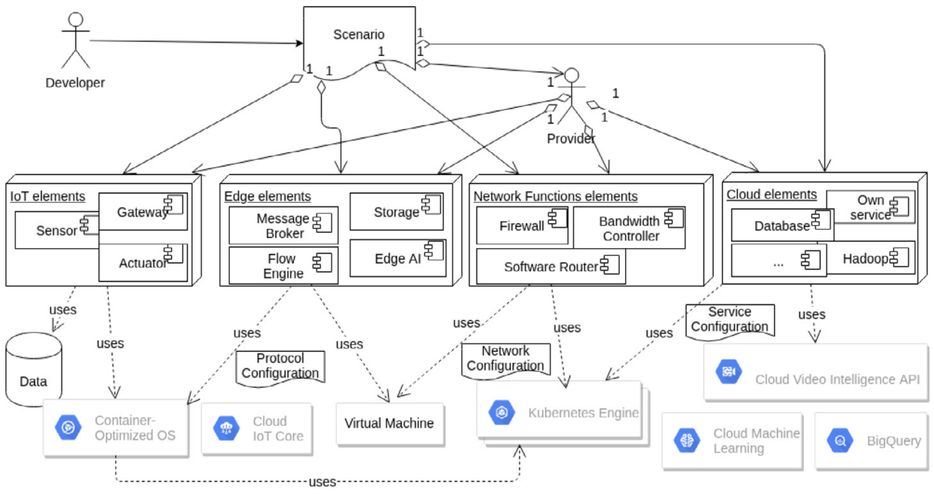


Fig. 2. (Simplified) developer needs and elements required by simulation scenarios.

key requirements in Section 2 and key designs in Section 3. We have implemented `IoTCloudSamples` and released it as an open source under GitHub. We will present various examples on how to use `IoTCloudSamples` for simulating edge applications in Section 4 and will discuss our future plan in Section 6.

2. Artefacts for emulating edge computing interactions

2.1. Key requirements

Within an edge application, the developer has to deal with a set of diverse types of elements covering infrastructures, platforms, protocols and data. These elements come from different subsystems and providers for IoT, edge, network functions and cloud resources. Examples of interactions in a quite straightforward edge analytics application are (i) taking IoT data as messages from an MQTT message broker, then (ii) transforming the data into another format (e.g., CSV to JSON), before (iii) passing the transformed data to another messaging system (e.g., Apache Kafka) for near-realtime processing and allowing only certain services to receive the result from the near-realtime processing. Thus, a simulation of an edge scenario requires many types of artefacts, including software, data and processes, which are common, domain- and application-specific. Moreover, the simulation must incorporate the roles of stakeholders inherent in edge computing, such as a service provider with high cost and quality versus a service provider with free usage and no quality agreement.

Fig. 2 describes a high-level view of a simulation scenario from the developer's view. Although the main objective of the simulation scenario might target to edge computing aspects by leveraging *Edge elements* (e.g., edge broker, flow processing engine and edge AI), the simulation scenario will have (i) *IoT elements* for acting as data sources for and objects being controlled by the Edge elements, (ii) *Network Functions elements* for acting as network features among the Edge, the IoT and the Cloud, and (iii) *Cloud elements* for acting cloud services used by the edge. Such elements are also associated with data sources used for simulations, and with application-level protocol, network and service configurations. Conceptually, these elements and their artefacts – software, data, and configurations including processes – can be offered by the developer and the provider, whereas the provider might be communities, public organizations or real-world pay-per-use providers.

To leverage these artefacts, in a realistic manner, besides simulation artefacts, we must provide additional techniques to integrate simulations with real-world services. The key requirements for such simulations are:

1) RQ1 – Provide diverse types of artefacts close to real design and deployment: adequate software elements covering different aspects must be provided. We have the following types of elements:

- IoT elements: include, e.g., sensors, actuators and samples of data. They are used for simulating data sources and IoT systems being controlled at the edge.
- Edge elements: include, e.g., edge-based brokers, message processing and edge-based AI [11]. They are used to simulate processing, storage and message capabilities at the edge.
- Network Functions elements: include, e.g., firewall, routers and network controllers. They are mainly used for simulating network functions.
- Cloud elements: include, e.g., databases, big data processing and infrastructural computing resources. They are used to simulate cloud backend services for the edge.

Furthermore, many of these elements must be composed and interacted through known application protocols, such as LoRaWAN, MQTT, AMQP and REST, and can be controlled at runtime to allow the realistic design of today's edge applications.

2) RQ2 – Simulate service models: due to the widely offering service models, e.g., pay-per-use models in IoT, cloud and edge computing [12,13], the simulation should be based on the composition and steering of the above-mentioned elements as a service. Generally, existing artefacts belong to not only the developer creating simulations but also other providers (and some are freely available). Furthermore, even though an artefact belongs to the developer, the developer might use it for simulating a service provisioned by a third-party (hosting) provider. For example, a customized message broker based on MQTT can be configured by the developer, but be deployed into an edge machine, offered by a provider, to simulate a pay-per-use IoT Data Hub. For this reason, we need to distinguish available artefacts from available services for simulations. The latter are either acquired or provisioned for the simulations but they are from other providers. In practice, we have diverse types of providers for IoT, Edge, Network Functions or Cloud elements.

3) RQ3 – Simulate application-level scenarios considering dependencies among layers and subsystems: edge interactions across layers exist in application-level scenarios. Different studies have shown that edge interactions are complex [14,15], thus simulating a single layer, e.g., compute resource provisioning and management, is not enough. Many interactions can be simulated within a subsystem, for example, examining how edge infrastructures would scale when processing IoT data using edge flow engines. In this case, a simulation needs all elements mentioned in *RQ1* but the simulation study would be focused only on Edge elements. However, other studies, such as, controlling data aggregation and ingestion flows to/from the edge from/to the IoT/Cloud, require us to steer different subsystems of IoT, Edge, Network Functions and Cloud elements.

4) RQ4 – Reuse infrastructures and able to control different layers as well as to combine simulated elements with real systems: Instead of doing simulation for a single layer or making changes in every layers of a single simulation, we should allow multiple simulations running atop another simulation. One example is to develop a simulation for the infrastructural layer *across* IoT, Edge, Network Functions, and Cloud by using appropriate platform services. Then, atop such an infrastructural simulation, additional simulations about application-/domain-specific interactions can be added, for example, how data transformation would behave or how reliable application services would fluctuate when changing the infrastructural resources.

2.2. Approach

Our approach supports simulating edge scenarios by using real deployable and runnable artefacts with a minimum implemented and emulated business logic, followed symbiotic engineering principles. Given the complexity of interactions, it is hard to provide a simple script or a single language based on that everything can be generated for the simulation. Therefore, our method is *edge-simulation-as-code* and is characterized by:

- Provide ready-to-use elements for simulations so that the developer just needs to specify configurations of elements and deploy them for simulations.
- Enable modification and addition of elements so that the developer can tailor existing elements and add new ones for simulation easily.
- Provide provisioning and steering utilities for controlling elements, like in real development environments, so that the way the developer create and control simulations is like that for a real software systems in edge computing.

This leads to not only providing a software framework but also incorporating our best practices and experiences of the typical DevOps activities and tools that the developer uses in design and execution of edge applications.

3. `IoTCloudSamples` framework and techniques

We present three core aspects for supporting simulation of edge interactions. First, we present a software framework with diverse types of elements for building simulations. Second, we present our techniques, based on *edge-simulation-as-code* principles, for development and operation activities during the DevOps of simulation scenarios. Third, we present methods for including extended software ecosystems into simulations based on `IoTCloudSamples`.

3.1. Software framework design

Fig. 3 presents key concepts in `IoTCloudSamples`. Key concepts in our simulation are `Unit`, `Provider`, `Configuration`, `Artefact`, `Scenario` and `Stakeholder`.

3.1.1. Units

A `Unit` is a basic software component, which is categorized into IoT, Edge, Network Functions and Cloud. A `Unit` is associated with an `Artefact` that can be a standalone executable program, a dockerized container or a service running in a set of (virtual) machines/containers. `Units` are diverse: some units are complex, e.g., a MongoDB database, which might require a deployment with several machines/containers. Other units, like a firewall, might interface to low-level APIs, which are not necessarily accessible to the developer. The key aspect of `IoTCloudSamples` is that `Units` are services and are executed in virtualized/containerized environments. Thus, they can be instantiated on-demand and controlled at runtime for real edge and cloud scenarios.

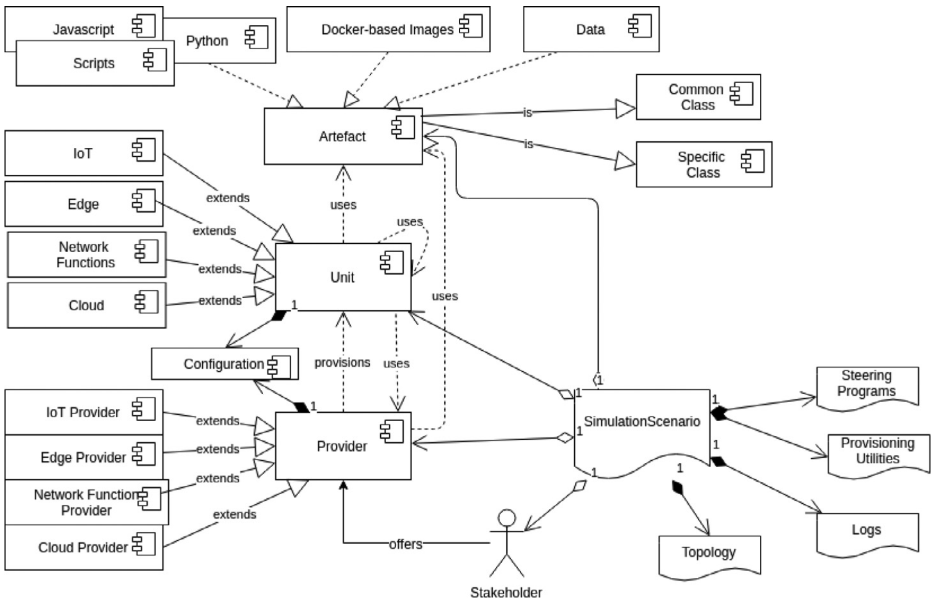


Fig. 3. (Simplified) conceptual view of key elements in `IoTCloudSamples`.

Table 1
Examples of units for protocol bridges. In many scenarios, we need to integrate different protocols. This subclass of units provides common bridges for different APIs and protocols for exchanging data.

Units	Description
`PubSubToCoAP`	a bridge between generic MQTT and AMQP brokers with CoAP servers to transform messages between MQTT/AMQP protocols and the CoAP protocol
`RESTtoAMQP`	a bridge between REST HTTP calls and AMQP brokers
`http2datastorage`	a bridge HTTP server to Google Storage, allowing simulating HTTP to Google Storage native protocol
`mqttamqpbridge`	a bridge between MQTT brokers and AMQP brokers, allowing simulating message relay and protocol interoperability
`mqttkafkabridge`	a bridge between an MQTT broker and a Kafka messaging system

Table 2
Examples of domain and application-specific units.

Units	Description
`port-control-service`	a service simulating a seaport control, which accepts requests from vessels and other stakeholders
`portAlarmService`	a service simulating alarm management within a seaport
`portVessel`	a service simulating a vessel, which is approaching a seaport
`portalarmsensor`	a service simulating alarm sensors in a seaport
`testRig`	a service simulating a remote rig system whose objects can be controlled to move in a space

We have common, domain-independent units as well as domain- and application-specific units for simulations. Examples of common units are sensors, actuators, message brokers, flow engine and storage. Domain-specific units are dependent on scenarios and applications domains; examples of domain-specific and application-specific units could be protocol bridges shown in Table 1 and application-specific units shown in Table 2.

3.1.2. Providers

`Units` provide basic functionality but they do not reflect the business model that the stakeholders want to simulate in scenarios. In order to support the service and pay-per-use model in simulations, we develop `Providers`, which provision and manage units as resources for simulations. `Providers` are associated with `Artefact` and will provide `Units` as resource instances. However, `Providers` can be controlled to enable different business and service models, allowing `Units` instances for different purposes. Thus, we can simulate many service providers of the same type of resources. Given a type of `Units`, we can use different providers of the same type: e.g., an MQTT broker can be from a real cloud IoT Hub provider or from an emulated edge MQTT provider in an edge machine by the developer (on-premise). This is useful for testing scenarios in which different providers of the same type of resources offer different quality of services, security settings and locations. Currently, we have the following types of Providers: (i) from existing real cloud and edge

Table 3
Examples of some providers implemented.

Units	Description
`IoTCameraDataProvider`	a service provider interfacing to several real cameras to offer camera data as a resource for simulation
`alarm-client-provider`	a provider for creating alarms for different users
`bigQueryProvider`	a wrapped provider for emulating big data resources atop Google BigQuery
`ingestion-provider`	a service provider supporting simulating services for data ingestion
`sensor-provider`	a service provider supporting simulating sensors (creating, controlling and removing sensors)
`dockerizedserviceprovider`	a generic service provider for running dockerized services, which are common or application-specific
`kubernetesFirewallProvider`	a service provider for creating firewall as a network function for a Kubernetes environment
`mosquitt-mqtt-provider`	a service provider that can be used to create MQTT brokers for different users
`test-rig-provider`	a dockerized rig service provider for simulation of control of movable objects
`vessel-provider`	a provider for simulating vessel creation for companies

providers in the market, (ii) by simply creating instances from common units available in open sources and public repositories, and (ii) by developing common and domain-/application-specific providers based on assumption business models. Similar to `Units`, most `Providers` can be deployed as container-based services (e.g., Docker or Kubernetes). Table 3 shows some examples of providers which we have developed.

3.1.3. Data

The data must be close to realistic cases. In our framework, we consider to enable the developer to use widely available public datasets and private real datasets. Furthermore, there are many open IoT testbeds and public platforms that we can use to obtain data. When the data is not available, our method is to emulate data created from real datasets. In this case, we use existing scripts and libraries, such as Faker[2] and `mock-data-generator`[3] to create emulated data based on real data. For example, Listing 1 shows an example of creating simulated data based on real format of emission data from a seaport (based on a real sample from Valencia).

```
var mocker = require('mocker-data-generator').default
var emissionCabins = {
    emissionCabinId: {
      faker: 'random.number({"min": 1, "max": 100})'
    },
    name: {
        randexp: /Cabina VR-(001|002|003|004)/
    },
    portId: {
        faker: 'random.number({"min": 1, "max": 20})'
    },
    description: {
        randexp: /Ubicada en Caseta Ecoport/
    },
    latitude: {
      faker: 'random.number({"min":26, "max":27})'
    },
    longitude: {
      faker: 'random.number({"min": 18, "max": 19})'
    },
};

mocker()
    .schema('emissionCabins', emissionCabins, 6)
    .build(function(error, data) {
        if (error) {
            throw error
        }
        var valenciaport = {
          "valenciaPortData": data
        }
        console.log(JSON.stringify(valenciaport));
    })
```

Listing 1. Creating emulated data for a scenario.

Currently, `IoTCloudSamples` provides some sample of data collected from real systems. Examples are from monitoring real Base Transceiver Stations[4], network monitoring data, or public cameras collected and shared only on-demand.

[2] https://github.com/marak/Faker.js/
[3] https://www.npmjs.com/package/mocker-data-generator
[4] A small set of data is published at: https://github.com/rdsea/IoTCloudSamples/tree/master/data/bts whereas many months of data are kept confidential.

Table 4
Examples of scenarios.

Scenarios	Simulation goal	Related artefacts, data and protocols
Base Transceiver Station (BTS) Analytics	Understanding edge infrastructural resource scalability; data transformation; access control; integration with cloud	IoT sensors; edge broker; edge data ingestion; firewall; cloud storage; Hadoop; MQTT/AMQP and REST
Test Rig	Understanding uncertainties in edge/IoT control	REST-based edge control service; actuators; sensors
Accessing Video Data in seaport	data interoperability; protocol interoperability; runtime dynamic resource provisioning; integration with cloud services	camera data; edge container-based execution platform; Google Storage
Access data in a seaport	data transformation and integration; middleware interoperability; protocol interoperability; data interoperability with different stakeholders	sensor data; MQTT/AMQP brokers; flow engine/Node-RED; edge container-based computing platform

3.1.4. Scenarios

From `Units`, `Providers` and data, the developer can create different simulation scenarios. Each scenario describes a system to be designed that might be used for different purposes. In our framework, `Scenario` is abstract and is linked to various artefacts. One type of artefacts is the system topology of service units and providers that can be specified in common structures like TOSCA[5] or resource ensembles [16]. When services in a topology are deployed and executed, the elements for the simulation are started, and the simulation runs and can be controlled. Artefacts in scenarios, scripts for running scenarios, and logs of scenarios are currently managed by the developer. Depending on simulations, bootstrapping and simulation control programs can be simple or complex, e.g., written in Python or JavaScript programs to invoke APIs of service units and providers in the simulation. Following *edge-simulation-as-code*, we consider the freedom of the developer to write such programs. We provide some scenarios extracted from real projects in Table 4.

3.1.5. Prototype

`IoTCloudSamples` – available in GitHub[6] – is implemented with Python and JavaScript/NodeJS with various state-of-the-art technologies, e.g., Kubernetes, Docker, MongoDB, BigQuery and Node-RED. Simulations are carried out in virtual environments of containers and virtual machines. Thus, we leverage both public and private cloud and edge infrastructures for running simulations. The deployment of simulation scenarios can be accomplished manually for selected elements in the simulation or automatically by existing tools available for service deployment in virtual environments.

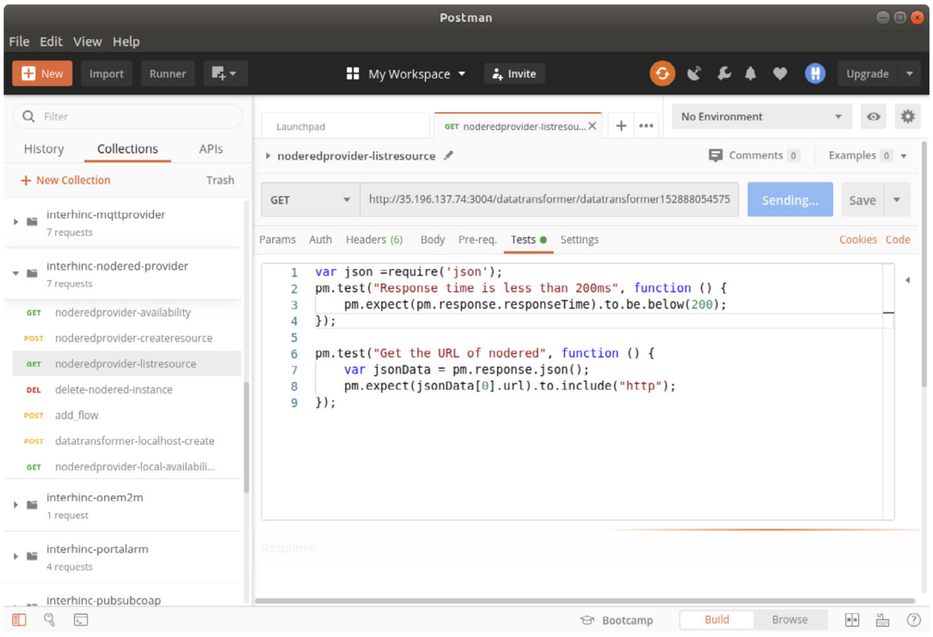
3.2. Simulation-as-code and its DevOps

In *edge-simulation-as-code*, we consider three main phases, each includes many typical activities in DevOps, but focusing on *edge-simulation-as-code*:

Phase 1 (Devs)– Design and development of scenarios:

- *Search simulation artefacts and services for* `Units` *and* `Providers`: this includes artefacts in existing local repositories, potential artefacts from public repositories, such as DockerHub, and available artefacts and services from public edge and cloud systems.
- *Identify missing* `Units`: most common units, but domain-specific or application-specific ones, are usually available. Like any design, these missing units have to be identified by the developer.
- *Identify* `Providers` *and their business models*: the developer has to decide which types of providers and their business models for the simulation scenario. The simplest case is to have only one provider and one business model defined by the developer. However, usually simulations will mix different provider profiles, such as a simulated provider for Edge elements and Google Cloud Platform as another provider for Cloud elements.
- *If needed, build and modify* `Units` *and* `Providers`: common situations are that the domain-specific and application-specific units are missing and, while units are available, no suitable provider exists. In this case, additional effort needs to be spent for building and/or tailoring units and providers. Since the goal is for simulation, the effort for developing core functionality of such units and providers will be light, while the interface/APIs and the service implementation must be performed properly to allow the integration with other services.
- *Build/configure suitable configurations for* `Units` *and* `Providers`: this is typically dependent on the goal of the simulation.
- *Identify external utilities for scenarios*: during the simulation, the developer might need to manipulate some common features, e.g., network bandwidth manipulation. In this case, finding suitable utilities, which are used in real systems, is more appropriate.

[5] https://wiki.oasis-open.org/tosca/TOSCA-implementations
[6] https://github.com/rdsea/IoTCloudSamples


Postman
File Edit View Help
New Import Runner
My Workspace Invite Upgrade
Filter
History Collections APIs
New Collection Trash
interhinc-mqttprovider
7 requests
interhinc-nodered-provider
7 requests
GET noderedprovider-availability
POST noderedprovider-createresource
GET noderedprovider-listresource
DEL delete-nodered-instance
POST add_flow
POST datatransformer-localhost-create
GET noderedprovider-local-availabili...
interhinc-onem2m
1 request
interhinc-portalarm
4 requests
interhinc-pubsubcoap
Launchpad
GET noderedprovider-listresou...
No Environment
noderedprovider-listresource
Comments 0
Examples 0
Sending...
Save
Params Auth Headers (6) Body Pre-req. Tests Settings
Cookies Code
Response
Bootcamp Build Browse

GET http://35.196.137.74:3004/datatransformer/datatransformer152888054575

```
var json =require('json');
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Get the URL of nodered", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[0].url).to.include("http");
});
```

Fig. 4. Example of using Postman to check a flow engine unit created by a flow engine provider deployed in an edge machine (the flow engine unit is based on Node-RED).

The developer can combine different ways to perform simulations, such as provisioning using manual tools, writing scripts to deploy simulations or services, and running simulations:

Phase 2 (Ops)– Provisioning and execution of scenarios – since the whole framework is based on virtualized services, key activities in Phase 2 (Ops) are quite similar to typical operation activities in cloud engineering:

- *Create topologies of resources as service to be deployed*: the topology includes `Units` and `Providers`. Typical models for service topology structures can be used. Such topologies can be created manually or generated by existing tools.
- *Deploy and provision* `Units` *and* `Providers`: many deployment tools can be used but the developer can also use command-lines and scripts provided together with `Units` and `Providers`, such as from Docker, Kubernetes, and Google Cloud Platform.
- *Run simulation* : this is similar to running any service system in virtual environments. As elements in a simulation are service-based, we can use tools or code to invoke service APIs to run the simulation. For example, Fig. 4 shows how to use Postman to check a service in a simulation.

During the simulation, the developer can steer the simulation. Common steering operations are, e.g., to change infrastructural elements, inject data processing flows, and to add new elements into the simulation. This will be done in the following phase:

Phase 3 (Steering) – Simulation steering for scenarios: since it is code and many APIs are available during the test we can steer the services. Such actions can be done through code. For example, the following excerpt shows to control creation and deletion of service units during the simulation:

```
with open(args.test_output, ’w’,newline=’’,encoding=’utf-8’) as csvfile:
 fieldnames = [’’nrclient’’,’’serviceping’’,’’creationtime’’,’’listtime’’,’’deletiontime’’]
 writer = csv.writer(csvfile)
 writer.writerow(fieldnames)
 for max_workers in range(1, num_client, 2):
 print(’’Test with ’’, max_workers)
 results=[]
 with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as executor:
 for i in range(max_workers):
 future = executor.submit(single_test,max_workers,num_test)
 results.append(future.result())
 print(results)
 #print(np.average(results, axis=0))
writer.writerow(np.insert(np.average(results, axis=0),0,max_workers))
```

Extra tools can also be used, for example for searching artefact or automatically deployment. In our work, for certain simulations, we also use rsiHub[7] for discovery. Since our approach relies on software artefacts that can be instantiated as services and external realistic services, many activities in **Phase 2** and **Phase 3** can be interwoven by utilizing suitable tools. For example, using dynamic techniques [16], one can perform runtime changes of the topology and resources for the simulation.

3.3. Integration with external, real software ecosystems

The key related software in our framework are (i) external providers and (ii) utilities for connecting external providers with simulations. Such utilities can be developed by the developer as the developer knows which real services should be included. In this case, the developer can follow convention and best practices and examples in `IoTCloudSamples` to build connectors and adaptors. Then, within simulation steering code, invocations of the newly deployed services can be written and executed. This allows the elasticity of external services in existing simulations. Another way is to leverage some existing utilities for discovery and deployment of simulation elements, such as rsiHub [16].

4. Illustrative examples and experiments

4.1. Edge analytics in seaport examples

We used several scenarios at a seaport to illustrate simulation examples. Our seaport analytics scenarios are extracted from the H2020 Inter-IoT project[8]. Other available examples are for edge analytics for telco infrastructures and for a remote rig lab.

4.1.1. Simulation software artefacts

Artefacts are available in our prototype in GitHub. Some common `Units` and `Providers` in `IoTCloudSamples` used for examples are:

IoT cameras: in our scenarios, in the seaport, there are many cameras, which provide real-time data (and historical data). Each camera has metadata about the location, video data format, etc., and is an IoT service unit, whose data can be pushed or pulled. There is a camera provider for the seaport that manages cameras and allows consumers to search and request camera data resources. Since, in practice, the developer is not allowed to access cameras in the real seaport, we can simulate seaport cameras by using public cameras. In our simulation, our `IoTCameraDataProvider` implementation[9] interfaces to real cameras.

IoT sensors: in the seaport there are many sensors, which provide information about emission cabins, weather stations, weather measurements, emission measurement and sound measurements. Real data was obtained for simulations but no direct access to sensors was given. For simulations, we do not need to develop a new domain-specific sensor code for each type of sensors. Instead, we use a common sensor code to handle different types of sensor sample data. The common sensor code[10] simulates real sensors and data providers by taking existing datasets from the Valencia seaport and replaying them.

Network Functions: we assume the following service model for a seaport: an edge computing platform provider offers various services within the seaport. In the seaport a Network Function provider was used to offer firewall functions that can control the traffic in/out the seaport. We emulate this by assuming that the seaport infrastructure is running as a Google cloud virtual infrastructure. Our firewall provider leverages Google firewall features[11]. Another way to simulate firewall as network function is to build a service atop UFW[12].

Edge computing brokers: in the seaport there is a provider, which can offer message brokers on demand. The broker we use in our example is MQTT. If a consumer needs a message broker, the provider will provide an instance of the broker for the consumer[13].

Edge computing workflow engines: In the seaport there is a provider which can offer a data processing workflow engine that a consumer can use. The data processing workflow we use is Node-RED and a data transformation provider has been developed for simulating data processing tasks[14].

Cloud services: there are many cloud services available outside the seaport. We use, e.g., Google BigQuery, Google Cloud virtual machines and Google Storage. Furthermore, the computing brokers and computing workflows (like in the edge situation within the seaport) can also be provided as cloud services.

[7] https://github.com/rdsea/HINC
[8] https://inter-iot.eu/
[9] https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/IoTCameraDataProvider
[10] https://github.com/rdsea/IoTCloudSamples/tree/master/IoTProviders/sensor-provider
[11] https://github.com/rdsea/IoTCloudSamples/tree/master/NetworkfunctionsUnits/SimpleFirewallController
[12] https://help.ubuntu.com/community/UFW
[13] https://github.com/rdsea/IoTCloudSamples/tree/master/IoTProviders/mosquitt-mqtt-provider
[14] https://github.com/rdsea/IoTCloudSamples/tree/master/InterOpProviders/nodered-datatransformer-provider

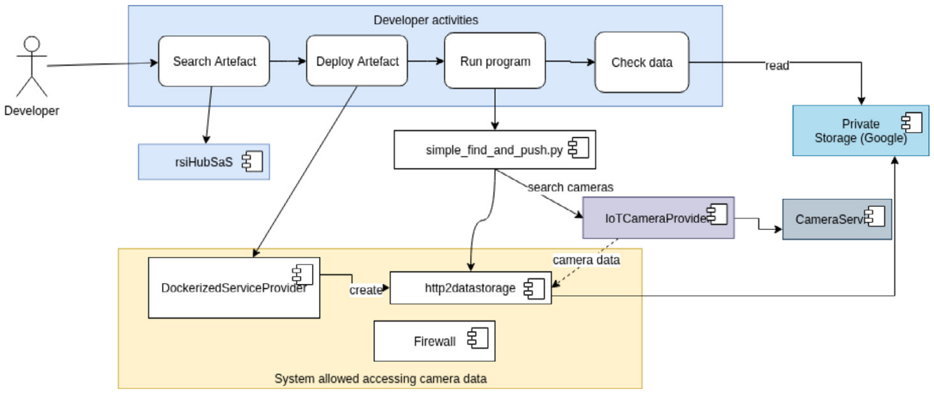


Fig. 5. Activities and interactions in provisioning and accessing services.

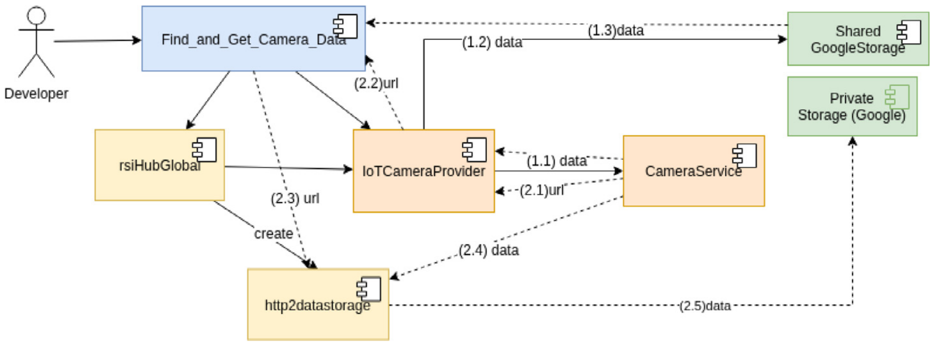


Fig. 6. Interactions in obtaining data units.

Domain- and application-specific services: we use some of the services given in examples in Section 3.1 for vessels, vessel providers, truck providers, cranes providers, and port management services.

4.1.2. Accessing video data in seaport

Simulation scenario: a consumer within the seaport requests some cameras and obtains the video camera data by pulling the data itself (using HTTP). Another consumer outside the seaport does the same. Assume that an emergency happens, the consumer outside the seaport asks the camera provider to reconfigure resources and push the data to an outside cloud service, e.g., Google Storage. As a consequence, the network function in the edge must allow “pushing data” to Google Storage.

Study goals: we can use this scenario to simulate interactions with edge services with real, external services. We can also study performance of data transfers between the edge and the cloud at the application level.

Simulation setup: based on the designed scenario, the developer has several activities for setting up the simulation. Fig. 5 shows activities of developers, possible software artefacts, their instances of units and providers and extra utilities. Key activities are Search Artefact, Deploy Artefact, Run Program and Check Data are corresponding to activities in Section 3.2. The main simulation control program is `simple_find_and_push.py` which is used to control services. Except that `Google Storage` and real `Camera Services` are available from third parties (public and pay-per-use), other artefacts are provided by `IoTCloudSamples` and we also use `rsiHub` utilities for artefact discovery.

Examples of simulation actions: Fig. 6 presents further detail of how actions are done within the simulated scenarios. For the simulation, the developer can develop code to perform different tasks, such as, to find cameras, retrieve data and push data to cloud services[15]. For example, the following log shows (simplified) results of a request to take the latest camera video based on an input geohash and push the result to Google Storage:

```
$python3 src/simple_find_and_push.py --provider_url http://xxx.xxx.xxx.xxx:3000/camera
{''name'':''http://xxx.xxx.tv/chn/DNG57/v204117.ts'',''timestamp'':''1537981980''}
We will call an external program to store
https://storage.googleapis.com/userexchangedata/1537956804387_v204117.ts
```

[15] Examples of python code are available at: https://github.com/rdsea/HINC/tree/master/scenarios/camerainseaport/src

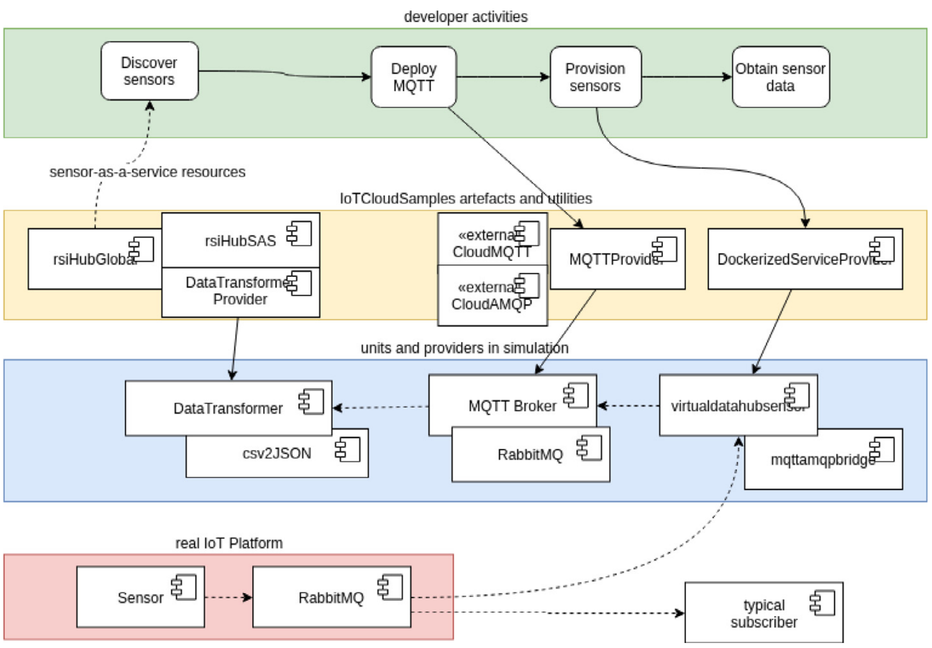


Fig. 7. Combining transformation with virtual sensor services.

In detail, a protocol broker `http2datastorage` was deployed as a docker container and the steering program, after searched and obtained suitable URIs of data from `CameraService`, requested the `http2datastorage` to retrieve the data through HTTP API of `CameraService` and then push the data to Google Storage. To allow pushing data, `Firewall` can be controlled to change the network configuration.

4.1.3. Data transformation and virtual data-as-a-service

Simulation scenario: a consumer in the seaport wants to access sensor data in the seaport with the condition of non-sharing data exchange middleware. A simulation will be created to allow different ways to request and receive selected data in the seaport.

Study goals: we can use this scenario to study middleware interoperability, communication protocol interoperability, and data interoperability. Furthermore, performance issues associated with these interactions can be studied.

Simulation setup: for non-sharing middleware in receiving data (e.g., due to security and data regulation), a private instance of virtual sensor-as-a-service must be provided. We can use the generic `virtualdatahubsensor` for Virtual IoT DataSensor-as-a-service and use `DockerizedServiceProvider` as a `Provider` for Virtual Sensor Provider. The idea is that when a tenant wants to access a specific type of data, we can create an instance of `virtualdatahubsensor` with the right configuration. The simulation's simple steering program creates such instances. Fig. 7 shows activities for the provisioning and operation.

Examples of simulation actions: a consumer wants to process data from the MQTT broker using a separate workflow engine within the seaport. The simulation is reconfigured with a new Node-RED instance and the consumer adds a workflow into Node-RED. Second, another consumer wants to access the sensor data from the broker but finds that the data is in CSV, thus the consumer wants to deploy a resource to transform CSV data to JSON. Two possible solutions: (i) a new component is deployed that takes data from MQTT and transforms the CSV to the JSON[16] or (ii) a new Node-RED instance is created and a workflow for data transformation is pushed into the instance. The two cases achieve the same goal but have very different techniques. Another situation is that a third consumer needs multiple sensor data from different brokers, MQTT and RabbitMQ, but the third consumer wants to receive data only through MQTT. Based on the consumer needs, a `virtualdatahubsensor` will be deployed to use `mqttamqpbridge` to translate data from RabbitMQ broker to MQTT broker.

4.1.4. Data exchange and control in emergency situation

Simulation scenario: in this scenario, we assume there are alarms occurring in a seaport. The alarms are propagated through an MQTT broker. Usually, there are some analytics applications subscribing the alarms queues in the broker. One

[16] https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson

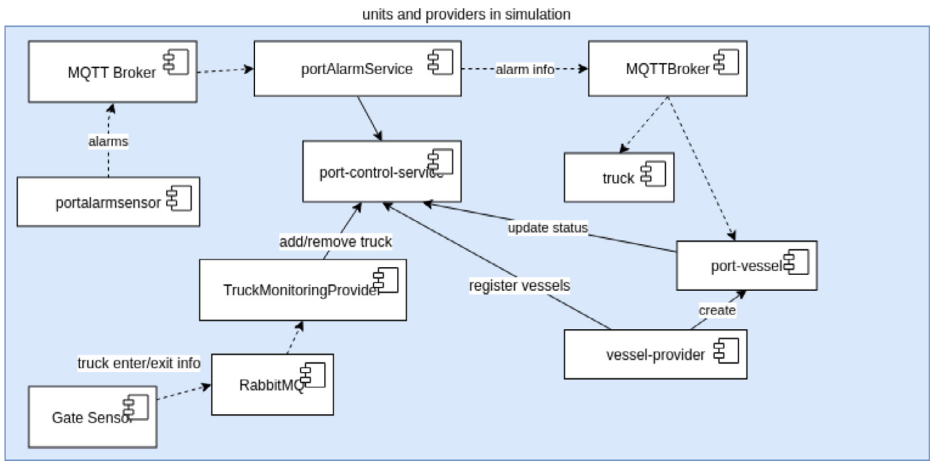


Fig. 8. Units and providers for simulating emergency in a seaport.

of such alarms analytics applications finds alarms related to terminals in the seaport. It queries a PortControlService (PCS) to obtain the list of vessels approaching the seaport. Based on the information about the vessels and the service providers of the vessels, the alarms analytics application creates new brokers as resources, or connects to existing communication means of the vessel providers, to share the information about the situations. The application can also send requests to ask vessels to stop or change the arrival plan. Similarly, another analytics application can inform other relevant objects around the terminals (e.g., by using geohash to query cranes and trucks) and request them to stop or change the plan. Another analytics program can request camera providers (for cameras close to the terminal, using geohash to filter the cameras) to provide videos to separate channels that can be accessed by polices and other relevant third parties.

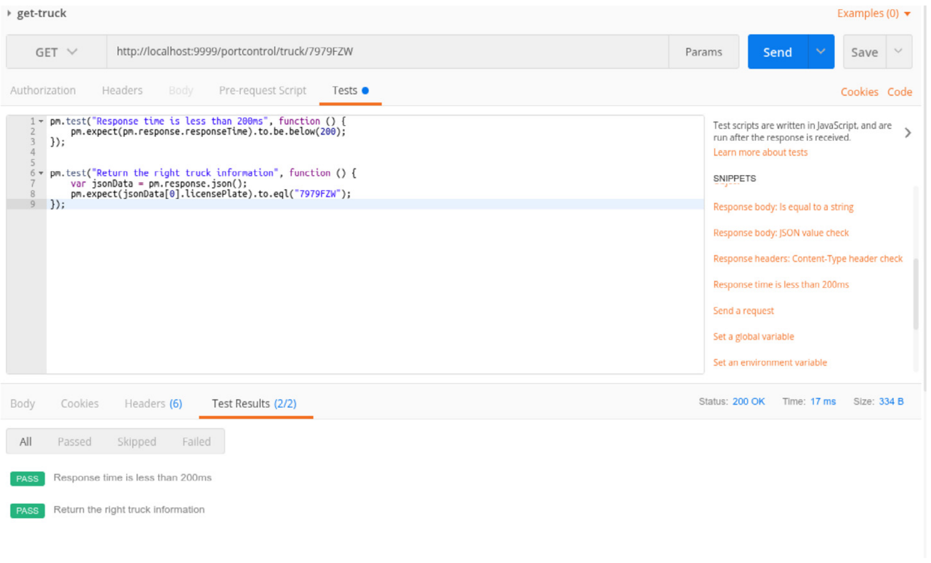
Study goals: Since vessel/truck/crane providers accept different forms of data and application protocols, we can study interoperability issues in data transformation and protocol integration. Furthermore, we can study online response time and error handling in interactions between services.

Simulation setup: in addition to common units and providers, various related application-specific artefacts are exemplified in Section 3.1. We use a common MQTT broker for alarms and `portalarmsensor` to generate samples of alarms. `AlarmService` simulates an alarm analytics application. A REST `port-control-service` simulates the PCS. The PCS has APIs for querying vessels and for updating vessels positions. A set of vessel emulators – `portVessel` emulates the movement of vessels. A vessel emulator subscribes information from its providers via a queue. Vessel providers, based on `vessel-provider`, accept a different format of data (JSON/CSV with different structures) and use different protocols (MQTT, AMQP and REST). Cranes and trucks are simulated in a similar manner with their corresponding providers. Vessels/trucks/cranes/cameras have their GPS positions so that geohash can be used to query them. For example, `TruckMonitoringProvider` monitors the enters and exists of trucks by obtaining events from RabbitMQ. To simulate the diversity of providers, we have different providers for each category of vessels, trucks and cranes. Except RabbitMQ, all elements are container-basedices.

Examples of simulation actions: Fig. 8 shows artefacts, units, providers and activities. To start with the simulation, `port-control-service`, `portAlarmService`, `vessel-provider`, `TruckMonitoringProvider`, `MQTT Broker` and `RabbitMQ` can be deployed. Then, `Gate Sensor` is started with real data obtained from the seaport to simulate flows of trucks entering and existing the seaport. At the same time, we control `vessel-provider` to create `vessel` to simulate vessels approaching port terminals. Fig. 9 shows an example of checking truck information and Fig. 10 for vessels. As long as `vessels` and `truck` are registered within `port-control-service`, we start to simulate alarms by running `portalarmsensor`. Based on alarms analyzed, `portAlarmService` will create requests to `vessels` and `trucks` to ask them to stop or move. We can activate many trucks and vessels and to test if `port-control-service` and `portAlarmService` can be elastic by changing alarms frequencies and volumes.

4.2. Creating scenarios by code

As mentioned, elements of edge applications under simulation and their underlying systems are exposed as (dockerized) services. Therefore, many scenarios can be created by writing code to deploy and control these elements at the provisioning and operation times. Consider in a simulation scenario, a stakeholder offers an edge flow engine provider based on Node-RED and Kubernetes for data transforming. Using a script and command-lines the developer can easily deploy the flow engine provider to Kubernetes. During the simulation, assume that the developer plans to create a new unit instance of a flow engine for handling new IoT data from a user, the following excerpt can be used to create a new instance when needed


get-truck

GET http://localhost:9999/portcontrol/truck/7979FZW

```
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});


pm.test("Return the right truck information", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[0].licensePlate).to.eql("7979FZW");
});
```

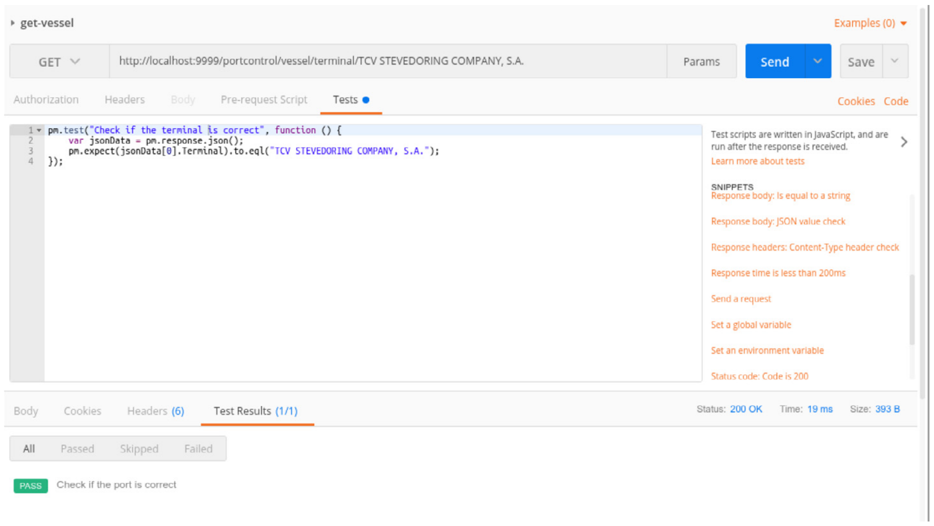
Test Results (2/2) Status: 200 OK Time: 17 ms Size: 334 B

PASS Response time is less than 200ms

PASS Return the right truck information

Examples (0)
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Cookies
Code
Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests
SNIPPETS
Response body: Is equal to a string
Response body: JSON value check
Response headers: Content-Type header check
Response time is less than 200ms
Send a request
Set a global variable
Set an environment variable
Body
Cookies
Headers (6)
All
Passed
Skipped
Failed

Fig. 9. Examples of obtaining truck information during simulation.


get-vessel

GET http://localhost:9999/portcontrol/vessel/terminal/TCV STEVEDORING COMPANY, S.A.

```
pm.test("Check if the terminal is correct", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[0].Terminal).to.eql("TCV STEVEDORING COMPANY, S.A.");
});
```

Test Results (1/1) Status: 200 OK Time: 19 ms Size: 393 B

PASS Check if the port is correct

Examples (0)
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Cookies
Code
Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests
SNIPPETS
Response body: Is equal to a string
Response body: JSON value check
Response headers: Content-Type header check
Response time is less than 200ms
Send a request
Set a global variable
Set an environment variable
Status code: Code is 200
Body
Cookies
Headers (6)
All
Passed
Skipped
Failed

Fig. 10. Examples of checking terminals for vessels.

for interactions (e.g., before sending more IoT data to test the performance of the flow engine) and then to deploy new flows into the instance:

```
#data transformer is a flow service provider based on Node-RED and Kubernetes
parser.add_argument('--datatransformer_provider', help='URL of the Data Transformer Provider')
#API for creating an new instance of a flow engine
base_url=args.datatransformer_provider
create_resource=base_url+''/''
# create a flow engine instance for a private use
create_response=requests.post(create_resource)
#get and return the URL of the instance
json_response = json.loads(create_response.text)
datatransformerId=json_response['datatransformerId']
get_data =list_resource+''/''+datatransformerId
```

Overall, using a combination of scripts, command-lines and Python/JavaScript code the developer can perform both provisioning and configuration of services for simulations.

However, the developer can divide the provisioning, configuration and control of simulations into different parts, each can be reused. Let us consider an example for simulating the analysis of monitoring data for Base Transceiver Stations (BTS). We have obtained real IoT data about power management, electricity, temperature, etc. However, to test designs, we need to simulate various subsystems: create and control sensors (IoT), edge message brokers and edge-to-cloud data ingestion, cloud services and firewall as a network function. The whole process might follow the principles of DevOps by developing and testing individual subsystems before connecting them to create a complete scenario. Listing 2 shows the main deployment gluing various parts to create the scenario[17].

```
const scenarioCreator = require(''./creator_modules/scenario_creator'');
const createSensor = require('./creator_modules/adapters_and_providers/create_bts-
    sensor-provider');
const createMQTT = require('./creator_modules/adapters_and_providers/create_mqtt-
    provider');
const createBigQuery = require('./creator_modules/adapters_and_providers/
    create_bigquery-provider');
const createIngestion = require('./creator_modules/adapters_and_providers/
    create_ingestion-provider');
const createAMQP = require('./creator_modules/adapters_and_providers/create_amqp-
    provider');
const createFirewall = require('./creator_modules/adapters_and_providers/
    create_kube-firewall-provider');
scenarioCreator.addAdapterProviderCreator(createSensor);
scenarioCreator.addAdapterProviderCreator(createMQTT);
scenarioCreator.addAdapterProviderCreator(createBigQuery);
scenarioCreator.addAdapterProviderCreator(createIngestion);
scenarioCreator.addAdapterProviderCreator(createAMQP);
scenarioCreator.addAdapterProviderCreator(createFirewall);
scenarioCreator.createScenario();
```

Listing 2. Example of creating scenario.

Basically, this way helps to reuse various snippets for simulation scenarios. We can have a stable and reusable version for provisioning and controlling certain parts of a scenario, e.g., a subsystem of sensors and IoT protocols, a subsystem of message brokers, infrastructural resources and network functions, or a subsystem of cloud services, and then reuse these parts in different scenarios.

4.3. Steering simulations

The developer would need support for steering simulations when studying complex interactions. Simulation steering operations can be applied to, e.g., infrastructural resources, service processing capabilities, and network controls. They are very different kinds of steering activities supported at runtime in `IoTCloudSamples`. The key enabling technique for this runtime support is to use APIs of service units and providers (the APIs are mostly implemented through standard protocols, like REST, MQTT and AMQP), and containerized capabilities.

The developer decides the levels of steering: individual service units and providers, a subsystem of units, or the whole simulation. We achieve this by using scripts, command-lines and external tools. In most cases, we use steering programs written in Python and JavaScript. The following script presents an example of a python-based test that perform a test which is a part of three steps for a simple resource slice: (1) create a Node-RED resource, (2) upload a flow to the Node-RED resource, (3) remove the flow, and (4) remove the resource. The example script just shows the upload and removal of flows into a Node-RED. A flow here can be used for IoT data processing or transformation that can be simulated at runtime:

[17] Available at https://github.com/rdsea/HINC/tree/master/examples

```
parser = argparse.ArgumentParser()
parser.add_argument('--nodered_provider', help='URL of the Data Transformer Provider')
parser.add_argument('--workflow_file',help='workflow_file')
args = parser.parse_args()
post_workflow_data =json.load(open(args.workflow_file))
headers = {'content-type': 'application/json'}
def test_upload_workflow(nodered_url):
 workflow_post_response=requests.post(nodered_url+''/flows'',data=json.dumps(
     post_workflow_data),headers=headers)
 print(workflow_post_response.text)
 if (workflow_post_response.text):
 json_workflow_response = json.loads(workflow_post_response.text)
 workflow_id=json_workflow_response['id']
 delete_workflow_response=requests.delete(nodered_url+''/flow/''+workflow_id)
 print(delete_workflow_response)
test_upload_workflow(args.nodered_provider)
```

From the related existing tools, the developer can also use test tools like Postman to carry out steering operations. Furthermore, our external utilities in rsiHub can also allow the developer to update part of the simulation topology, as shown in Fig. 11. When dealing with typical network bandwidth and failure manipulation, common tools like `tc`[18] and `Pumba`[19], can be used to control network aspects. Such a tool can be used separately or invoked in the steering code as discussed before.

4.4. Performance measurement and interpretation

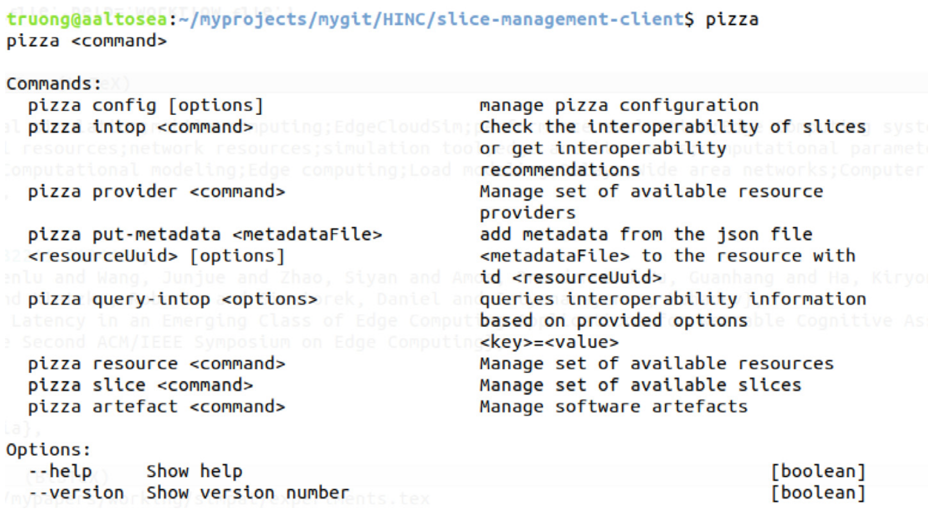
As `Units` and `Providers` are software services, well-known monitoring and instrumentation tools and techniques, such as Prometheus[20] and Fluentd[21], can be used to capture performance information (besides using testing tools like Postman, as discussed before). For example, from the steering example workflow shown in Section 4.3, we can see that it is relatively straightforward to modify the code and capture performance data by instrumenting the Python code. Currently, we leave performance measurement to the developer's work. Note that performance and scaling are strongly dependent on constraints set by scenarios, third-party `Units` and `Providers` and underlying infrastructures. Measuring and interpreting performance for a specific scenario must be considered together with the design goal of the scenario. Therefore, we do not present performance studies in this section, due to the complexity of the real business design of presented scenarios. Since our method for building edge scenarios relies real-world software components reused widely, we recommend the developer to focus on interpreting measured performance in suitable contexts.

Any deployment configuration has a very high impact on user acceptance criteria. Thus, we need to consider the user real deployment for analyzing performance. In our experiences, when `Providers` use Kubernetes to provision resources as containerized microservices, scaling service requests in provisioning resources is not an issue, mainly depending on the underlying Kubernetes deployment. However, the locations of `Providers` and clients strongly impact the performance. For instance, Figs. 12 and 13 show an example of a performance variability in two different production and test environments for the same interactions in creating a resource using `GenericLightweightIoTProvider` (which creates generic resources running under external processes). For each environment, with the change of a small number of concurrent clients, the response times were quite stable. However, the network latency and bandwidth between clients and providers in two environments are different, causing the end-to-end performance responses to be very different.

4.5. Further discussion

We have focused on illustrating examples of how to build scenarios. However, we have not presented performance or quality assessment associated with simulation scenarios or with effort spent on creating simulations. As part of simulation scenario is based on real environment setting and our focus is on supporting interactions at the application and service levels, performance evaluation for a specific scenario is left to the developer who builds the scenario. One important aspect is whether our framework speeds up the study of scenarios (mainly simulation and/or testing designs). In our work, we have used the framework for both research projects and teaching students to build IoT, Edge and Cloud scenarios since many

[18] http://manpages.ubuntu.com/manpages/bionic/man8/tc.8.html
[19] https://github.com/alexei-led/pumba
[20] https://prometheus.io/
[21] https://www.fluentd.org/



```
truong@aaltosea:~/myprojects/mygit/HINC/slice-management-client$ pizza
pizza <command>

Commands:
  pizza config [options]                      manage pizza configuration
  pizza intop <command>                       Check the interoperability of slices
                                              or get interoperability
                                              recommendations
  pizza provider <command>                    Manage set of available resource
                                              providers
  pizza put-metadata <metadataFile>           add metadata from the json file
  <resourceUuid> [options]                    <metadataFile> to the resource with
                                              id <resourceUuid>
  pizza query-intop <options>                 queries interoperability information
                                              based on provided options
                                              <key>=<value>
  pizza resource <command>                    Manage set of available resources
  pizza slice <command>                       Manage set of available slices
  pizza artefact <command>                    Manage software artefacts

Options:
  --help     Show help                                                  [boolean]
  --version  Show version number                                        [boolean]
```

Fig. 11. Snapshot of Pizza utility that can be used to change topologies by adding and removing resources as well as deploying new resources for simulation.

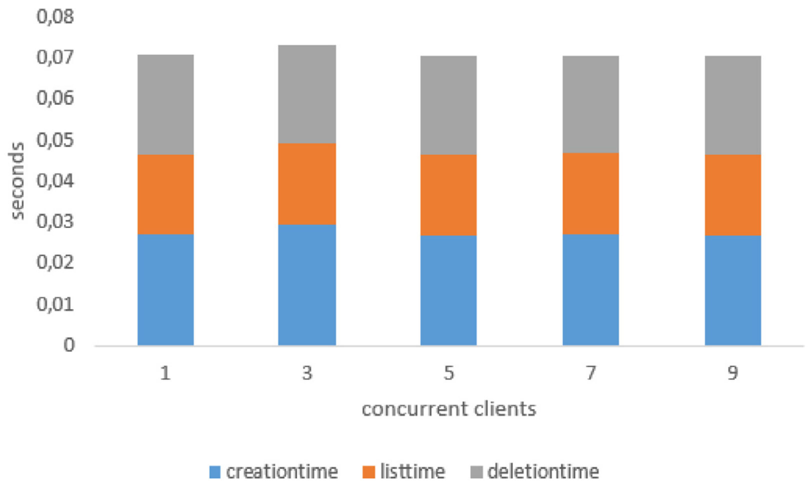


Fig. 12. Simple measurements of creation, listing and deletion of resources provided by `GenericLightweightIoTProvider`. The provider is deployed in a local edge server. The server is with 4 CPUs, 2 cores per CPU, Intel(R) Core(TM) i7-5500U CPU 2.40 GHz. The requests were sent from a test client running in a laptop in the same network with the provider.

years. However, we have not had an empirical study to measure the real software development productivity, e.g., in terms of development time and cost saving, when using our framework. This is actually challenging as there is no comparative framework for us to measure the productivity at such a high-level application-level design and simulation. We believe that such an empirical study for existing edge computing simulation tools is also missing to date. However, the empirical study is a separate scientific paper itself.

Compared with some simulation tools, our framework might be considered “incompleteness” – it is not a framework that the end user, in our case the researcher and practitioner, can just pickup simulation models, configure parameters and run the simulation. This is because our target is to support the simulation with real code, which is close to the application design and service interaction models. The disadvantage is that our framework cannot be used to study low-level layers like network performance, theoretical scaling, and energy consumption, like in many simulation tools (see the related work in Section 5). However, with our method, the developer has simulations very close to the realistic scenarios. Furthermore, it can be combined with other real software systems to create complex simulation scenarios. In our framework the technologies used are also the common ones in edge software development and the steps of performing simulations have also be built in best practices and experiences from our real work. Therefore, the developer also benefits from the combination between simulation/testing techniques and software development in the same study.

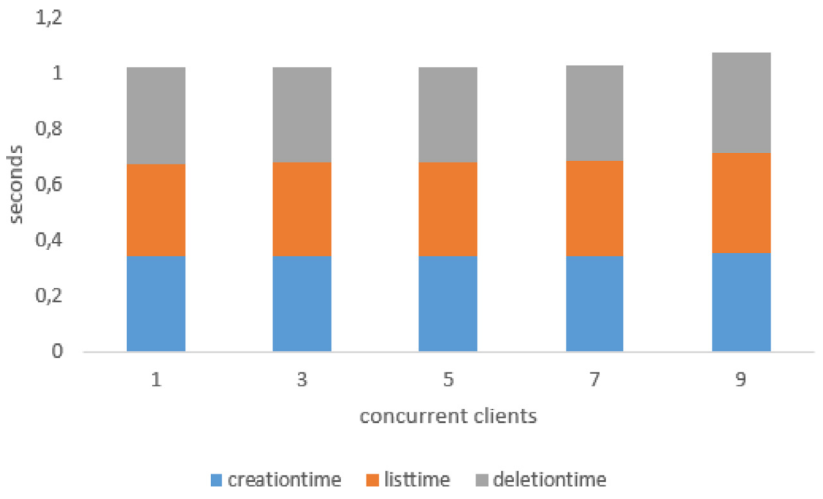


Fig. 13. Simple measurement of creation, listing and deletion of resources for `GenericLightweightIoTProvider`. The provider is running in a small Google virtual machine with machine type as n1-standard-1 (1 vCPU, 3.75 GB memory) in us-central1-a. The requests were sent from a client running in Austria.

5. Related work

The integration of IoT, edge and cloud has attracted many research and industrial works. Many papers have discussed the challenges and requirements for such integration [1,14,17,18]. Our work is focused on the software development challenges, especially for end-to-end software systems in edge-cloud continuum. However, we do not focus on programming languages like the works in [19,20].

Recently we have seen many tools support simulation and emulation of fog and edge computing. Even though comprehensive empirical studies [21] bring valuable insights, new edge designs cannot make use of empirical studies. Therefore, for new edge designs, especially w.r.t. task scheduling, compute resources and network structures, simulations and estimation methods to validate experiments have been intensively used by researchers [3,22]. To date there is no lack of papers describing/surveying existing and potential IoT Cloud and edge computing scenarios (and the number is continuously increasing). Our framework could support design simulation and evaluation with real software artefacts for such scenarios. However, it is important to understand the aspects to be supported in our framework. Our goal is not to work at a single aspect like deployment or network analytics. Our goal is to support end-to-end IoT-Edge-Cloud applications. Therefore, although "simulation" is emphasized in this paper – our work targets the developers, who work on the development of (new) edge applications and software systems.

Most edge/fog simulation frameworks are discrete event simulators, many are built from cloud and network simulators. A majority of such edge/fog computing simulation frameworks are dedicated for studying infrastructures, especially resource management and this direction is quite similar to simulations for cloud computing. MyiFogSim [6] just simulates virtual machine migration. EdgeCloudSim [8] concentrates on performance simulation for edge task execution in edge architectures. iFogSim [7] simulates many stuffs. EmuFog [23] allows one to define a topology of infrastructures and perform simulations of network/fog topology using discrete event simulations. FogNetSim++[24] focuses on networks in edge/fog and again it is a purely discrete event based simulation. Combined different simulations to create a hybrid simulation for IoT/edge scenarios is discussed in [25]. However, the simulation methods are not based on real code and systems like ours. From the implementation viewpoint, it is based on pure simulation code and tools, e.g., using OMNeT++ and MATLAB. It also does not focus on services and interoperability aspects. The work in [26] focuses only IoT devices simulations in edge computing.

Most of the existing work mentioned above simulate resources at the infrastructure. It is important to note that our approach and methods are different from these works because we do not rely on (parallelized) discrete-event based simulation and other types of simulators, like agent-based simulator. As we mentioned before, the use of "simulation" in our paper is a generic way as the applications to be designed are not 100% real – they include real code, mockup and emulated realistic environment. We support symbiotic simulations where models are built and run with real code, although the business logic is simplified. Our work is different because we focus on real software artefacts running in real testbeds emulating edge systems for cross application interactions. In principle, it is possible to estimate costs in our work through measurement of service usage. In our work, using monitoring integrated with services we could obtain the information to determine costs. However, in this paper, we have not discussed it as the main point of our work.

Only a few papers have been focused on scenario simulations. YAFS [27] simulates IoT scenarios but it is also a discrete event simulation. MockFog [28] uses clouds to emulate fog infrastructures including networks and machines, thus its approach is inline with ours through the use of real software. However, it focuses on infrastructures rather application interactions and application-specific scenarios. Phileas [29] aims to support simulations of Fog services by simulating value-based models mainly in operations for producing, passing and consuming IoT data. In the work of [25] various scenarios are given but its simulations are not based on real code and are not focused service integration, interoperability and application-level interactions like ours. `IoTCloudSamples` can enable similar scenarios but we provide more generic realistic software components for different layers.

Overall, most related works focus on common metrics, like performance and scalability through theoretical simulations. In this particular paper, we do not present scalability and performance studies for application cases illustrated. The main reason is that we focus on the pervasiveness of resources and services in real edge and cloud ecosystems in terms of software development, service mode, integration and interoperability in the end-to-end setting of edge systems. In fact, using our *edge-simulation-as-code*, although possible with containers and Kubernetes[22], it will be hard and expensive to study the scalability of a very large number of elements, e.g., at the scale of 10 thousands, in an edge system. For such cases, existing simulation tools should be used. Related works also do not enable the study of the stakeholder relationships and integration with real-world services and using real-world infrastructures to run emulation. Furthermore, our support is based on scenarios, rather than specific issues, like resource management, energy consumption or performance. Integrated scenario simulations require various types of artefacts, like IoT, edge, cloud units and data.

6. Conclusions and future work

Application scenarios in edge computing are complex, as they need to interact with diverse types of IoT, edge, networks and cloud services. In order to understand and simulate such scenarios, we need various types of artefacts, including software, data and processes. In this paper, we have presented `IoTCloudSamples` as a framework for enabling *edge-simulation-as-code* for edge scenarios with different stakeholders with different artefacts. Our framework allows the developer to define complex scenarios to test various high-level aspects, like data transformation and protocol interoperability across layers and subsystems in connection with dynamic changes of underlying infrastructural resources and business models. In our framework, we enable possible connections to real systems to make simulations more realistic, supporting symbiotic engineering principles for edge computing. `IoTCloudSamples` is an open source and its artefacts and scenarios are continuously updated, enabling symbiotic simulations of edge applications. Our key contributions in this paper show that realistic simulation scenarios are complex and require various artfacts and strong integration with real software systems. Furthermore, we have presented key steps in building simulations for complex edge software systems through typical software development activities.

More integration with network functions, especially with NFV and 5G network services, will be investigated. We are integrating mobile elements into our framework. Our tool can support an end-to-end activities for simulation, although certain activities require the developer to write code and simulated units. Currently, simulation scenarios are not specified via high-level languages and the developer has to manage various types information in a simulation. Our goal is to explore scenario-based languages and scenario management to support the developer. Many new aspects, such as transparency and accountability, will be our focus in future development.

Acknowledgment

In many years of its development, `IoTCloudSamples` has been benefited from contributions of many students, especially Lingfan Gao, Michael Hammerer and Duc-Hung Le, who have made a substantial contribution to the development of tools, service units, providers, scenarios and test code in `IoTCloudSamples` as well as `rsiHub` (https://github.com/rdsea/HINC). Exemplified code used in this paper have been partially written by them as well. The development of `IoTCloudSamples` has been supported through research activities in H2020 Projects U-Test and Inter-IoT where our project partners also provide useful scenarios and samples of data. Especially, we thank our colleagues in Inter-IoT for sharing Valencia seaport data, in BachPhu for sharing BTS data, and in Nordic Medtest AB for allowing interfacing to a real rig system. Some experiences reported in this paper have also been shared in a tutorial in the 8th International Conference on the Internet of Things IoT [30]. We thank many reviewers from different journals who give critical comments that help to sharpen the work.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: vision and challenges, SIGCOMM Comput. Commun. Rev. 45 (5) (2015) 37–42.
[2] M. Satyanarayanan, The emergence of edge computing, Computer (Long Beach Calif) 50 (1) (2017) 30–39, doi:10.1109/MC.2017.9.

[22] for example, the industry has tested with one million containers:https://www.hashicorp.com/c1m

[3] W. Zhang, Z. Zhang, H. Chao, Cooperative fog computing for dealing with big data in the internet of vehicles: architecture and hierarchical resource management, IEEE Commun. Mag. 55 (12) (2017) 60–67, doi:10.1109/MCOM.2017.1700208.
[4] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, H. Flinck, Mobile edge computing potential in making cities smarter, IEEE Commun. Mag. 55 (3) (2017) 38–43, doi:10.1109/MCOM.2017.1600249CM.
[5] F.A. Kraemer, A.E. Braten, N. Tamkittikhun, D. Palma, Fog computing in healthcarea review and discussion, IEEE Access 5 (2017) 9206–9222, doi:10.1109/ACCESS.2017.2704100.
[6] M.M. Lopes, W.A. Higashino, M.A. Capretz, L.F. Bittencourt, Myifogsim: a simulator for virtual machine migration in fog computing, in: Companion Proceedings of the10th International Conference on Utility and Cloud Computing, in: UCC '17 Companion, ACM, New York, NY, USA, 2017, pp. 47–52, doi:10.1145/3147234.3148101.
[7] H. Gupta, A.V. Dastjerdi, S.K. Ghosh, R. Buyya, Ifogsim: a toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments, CoRR abs/1606.02007 (2016).
[8] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: an environment for performance evaluation of edge computing systems, Trans. Emerg. Telecommun. Technol. 29 (11) (2018) e3493, doi:10.1002/ett.3493.
[9] D.N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R.K. Naha, S.K. Battula, S. Garg, D. Puthal, P. James, A. Zomaya, S. Dustdar, R. Ranjan, Iotsim-edge: a simulation framework for modeling the behavior of internet of things and edge computing environments, Softw.: Pract. Exp. 50 (6) (2020) 844–867, doi:10.1002/spe.2787.
[10] H.-L. Truong, N.C. Narendra, K.-J. Lin, Notes on ensembles of IOT, network functions and clouds for service-oriented computing and applications, Serv. Orient. Comput. Appl. 12 (1) (2018) 1–10, doi:10.1007/s11761-018-0228-2.
[11] S.B. Calo, M. Touna, D.C. Verma, A. Cullen, Edge computing architecture for applying ai to iot, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 3012–3016, doi:10.1109/BigData.2017.8258272.
[12] D. Georgakopoulos, P.P. Jayaraman, Internet of things: from internet scale sensing to smart services, Computing 98 (10) (2016) 1041–1058, doi:10.1007/s00607-016-0510-0.
[13] J. Qadir, N. Ahmed, F.Z. Yousaf, A. Taqweem, Network as a service: the new vista of opportunities, arXiv preprint arXiv:1606.03060(2016).
[14] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, O. Rana, The internet of things, fog and cloud continuum: integration and challenges, Internet of Things 3–4 (2018) 134–155.
[15] C. Li, Y. Xue, J. Wang, W. Zhang, T. Li, Edge-oriented computing paradigms: a survey on architecture design and system management, ACM Comput. Surv. 51 (2) (2018) 39:1–39:34.
[16] H.-L. Truong, L. Gao, M. Hammerer, Service architectures and dynamic solutions for interoperability of iot, network functions and cloud resources, in: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, in: ECSA '18, ACM, New York, NY, USA, 2018, pp. 2:1–2:4, doi:10.1145/3241403.3241407.
[17] J. Ren, D. Zhang, S. He, Y. Zhang, T. Li, A survey on end-edge-cloud orchestrated network computing paradigms: transparent computing, mobile edge computing, fog computing, and cloudlet, ACM Comput. Surv. 52 (6) (2019), doi:10.1145/3362031.
[18] C. Esposito, A. Castiglione, F. Pop, K.R. Choo, Challenges of connecting edge and cloud computing: a security and forensic perspective, IEEE Cloud Comput. 4 (2) (2017) 13–17.
[19] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, A. Kitazawa, Fogflow: easy programming of IOT services over cloud and edges for smart cities, IEEE Internet Things J. 5 (2) (2018) 696–707.
[20] N.K. Giang, M. Blackstock, R. Lea, V.C.M. Leung, Developing IOT applications in the fog: a distributed dataflow approach, in: 2015 5th International Conference on the Internet of Things (IOT), 2015, pp. 155–162.
[21] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, M. Satyanarayanan, An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance, in: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, in: SEC '17, ACM, New York, NY, USA, 2017, pp. 14:1–14:14, doi:10.1145/3132211.3134458.
[22] L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, Cost efficient resource management in fog computing supported medical cyber-physical system, IEEE Trans. Emerg. Top Comput. 5 (1) (2017) 108–119, doi:10.1109/TETC.2015.2508382.
[23] R. Mayer, L. Graser, H. Gupta, E. Saurez, U. Ramachandran, Emufog: extensible and scalable emulation of large-scale fog computing infrastructures, in: 2017 IEEE Fog World Congress (FWC), 2017, pp. 1–6, doi:10.1109/FWC.2017.8368525.
[24] T. Qayyum, A.W. Malik, M.A. Khan Khattak, O. Khalid, S.U. Khan, Fognetsim++: a toolkit for modeling and simulation of distributed fog environment, IEEE Access 6 (2018) 63570–63583, doi:10.1109/ACCESS.2018.2877696.
[25] G. D'Angelo, S. Ferretti, V. Ghini, Distributed hybrid simulation of the internet of things and smart territories, Concurr. Comput. Pract. Exp. 30 (9) (2018), doi:10.1002/cpe.4370.
[26] A. Kertesz, T. Pflanzner, T. Gyimothy, A mobile iot device simulator for iot-fog-cloud systems, J. Grid Comput. (2018), doi:10.1007/s10723-018-9468-9.
[27] I. Lera, C. Guerrero, C. Juiz, YAFS: a simulator for iot scenarios in fog computing, CoRR abs/1902.01091 (2019).
[28] J. Hasenburg, M. Grambow, E. Grunewald, S. Huk, D. Bermbach, MockFog: emulating fog computing infrastructure in the cloud, in: Proceedings of the First IEEE International Conference on Fog Computing 2019 (ICFC 2019), IEEE, 2019.
[29] F. Poltronieri, C. Stefanelli, N. Suri, M. Tortonesi, Phileas: a simulation-based approach for the evaluation of value-based fog services, in: 23rd IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD 2018, Barcelona, Spain, September 17–19, 2018, IEEE, 2018, pp. 1–6, doi:10.1109/CAMAD.2018.8514969.
[30] H.-L. Truong, Dynamic iot data, protocol, and middleware interoperability with resource slice concepts and tools: Tutorial, in: Proceedings of the 8th International Conference on the Internet of Things, in: IOT 18, Association for Computing Machinery, New York, NY, USA, 2018, doi:10.1145/3277593.3277642.