Software engineering for edge computing

Athanasopoulos, D. (2022). Software engineering for edge computing. In G. Karakonstantis, & C. J. Gillan (Eds.), *Computing at the EDGE. New challenges for service provision* (pp. 163-182). Springer International Publishing AG. https://doi.org/10.1007/978-3-030-74536-3_7

Published in:
Computing at the EDGE. New challenges for service provision

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
Copyright 2022 Springer Nature Switzerland AG.
This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Open Access
This research has been made openly available by Queen's academics and its Open Research team. We would love to hear how access to this research benefits you. – Share your feedback with us: http://go.qub.ac.uk/oa-feedback

Download date:19. May. 2025

Software Engineering for Edge Computing

Dionysis Athanasopoulos

School of Electronics, Electrical Engineering and Computer Science

Queen's University Belfast, Northern Ireland, UK

D.Athanasopoulos@qub.ac.uk

Contents

Introduction 3

Background 4

Software Engineering 4

Software Process 5

Edge Computing 6

Multi-Tier Edge Infrastructure & Modular Software 6

Standardizing Edge Computing 8

Related Work 8

Architectures, Infrastructures and Algorithms for the Edge 8

Software-Defined Networking for the Edge 9

Network Applications for the Edge 9

Edge Computing for Smart Cities 10

Cyber Security and Privacy for the Edge 10

Software Engineering for the Edge 10

Software-Engineering Aspects of Existing Approaches 11

Software Application & Edge Infrastructure Specification 11

Architecture & Implementation of Software Applications 11

Partitioning Software Applications 11

Offloading Software Applications 12

Deployment of Software Applications 12

QoS Requirements of Software Applications 12

Performance of Edge Infrastructure 13

Maintenance of Software Applications 13

QoS Requirements of Software Applications 14

Performance of Edge Infrastructure 14

Abstract Software Process for Edge Computing 15

Software Application & Edge Infrastructure Specification 16

Specification of Edge-Deployable Modular Software 16

Specification of Edge Infrastructure 17

Edge-Deployable Modular Software 17

Architecture of Edge-Partitioned Modular Software 18

Architecture of Edge-Offloaded Modular Software 18

Microservice Implementation/Test of Modular Software 19

Edge-Eligible Deployment of Service Components 19
Edge-Eligible Deployment Plan 19
From Deployment Plan to the Deployment to Machines 20
Edge-Enabled Maintenance of Software Applications 20
QoS-Based Maintenance of Service Components 21
Performance-Driven Maintenance of Edge Infrastructure 21
Conclusions 22
References 22

Introduction

Motivation

We apparently live in the data-driven era of interconnected mobile devices. Cisco estimates that there will be around 50 billion interconnected mobile devices by the end of 2020 [2]. As the hardware of the mobile devices is drastically getting improved, we have come to expect more from our mobile devices in terms of the types of tasks and data processing they can perform. However, these devices are not usually able to meet these demands on their own. Due to storage, computational and battery life constraints, mobile devices increasingly require the support of remote powerful (e.g., cloud) machines. While this type of architecture is advantageous for resource-constrained mobile devices, the usage of remote services is not always the best option for all of the types of software applications. In particular, as time-sensitive and location-aware applications emerge (e.g., patient monitoring, real-time manufacturing, self-driving cars, flocks of drones, or cognitive assistance), the distant cloud is not always able to satisfy the ultra-low latency requirements of these applications or to provide location-aware services [3].

To address the above challenges, Edge/Fog computing has been recently introduced by both industry and academia to quench the need for a computing paradigm close to mobile devices [4].

Edge/Fog computing bridges the gap between the cloud and mobile devices by enabling computing, storage, networking, and data management in edge nodes within the close vicinity of end-users' devices.

There are various surveys about Edge/Fog computing in the literature (see the *Related Work* section below). *What is missing though in the above surveys is the description of the software-engineering aspects of the applications that are built/deployed via the edge*. These aspects would be then useful for abstracting *the software-engineering process that is/should be followed by practitioners to build their edge-enabled software applications*.

Contribution

The contribution of the current chapter is twofold by aiming at covering the following objectives:

1. To highlight the software-engineering aspects of the current edge-computing approaches
2. To abstract a software-engineering process for edge computing and to outline research challenges in this process.

Chapter Structure

To cover the above objectives, we first specify the core concepts of the general-purpose software-engineering process, the multi-tier architecture of edge infrastructure, and how software applications are deployed to such an infrastructure (*Background* Section). Following, we describe the related surveys for edge computing and how the current chapter goes beyond those surveys (*Related Work* Section). We then describe the software-engineering aspects of edge-computing approaches (*Software-Engineering Aspects of Existing Approaches* Section). Finally, we outline the view and the role of a software-engineering process for edge computing, along with research challenges in this process (*Abstract Software Process for Edge Computing* Section).

Background

The current chapter presents the related work that comes from the following two research fields: (i) the general-purpose software-engineering process (ii) the multi-tier architecture of edge infrastructure and how software applications are deployed to such an infrastructure.

Software Engineering

What the software does is usually specified by the *functional requirements* of software. When we talk about the quality of software, we usually adopt the term *quality of service (QoS)* or *non-functional attributes* of software [11]. The latter term is not related to what the software does but it is related to the software behavior while it is executed. Examples of these attributes are the response time of software from an end-user perspective, the understandability of the program code, the cyber-security of software, etc.

Software engineering is a discipline that is concerned with all aspects of software production from the early stages of system specification to the maintenance of systems. The systematic approach used in engineering software is known as *software process*. A software process is a sequence of activities that lead to the production and facilitate the maintenance of a software system. There are the following generic activities usually met in software processes [11]:

1. *Software specification*: the functional requirements and the QoS attributes of software are specified
2. *Software architecture/design*: the artifacts (a.k.a., subsystems, packages, components, classes) of the code are defined, along with their relationships
3. *Software implementation/testing*: software is programmed to implement/test its architecture/design meeting the functional requirements and QoS attributes
4. *Software deployment*: software is made available for use (e.g., software installation to users' or cloud machines)
5. *Software maintenance*: software is modified to reflect changing requirements/attributes.

Different types of systems may need different development processes. For example, real-time software in an aircraft has to be completely specified before development begins. In e-commerce systems, the specification and the development activities are usually made together. Consequently, these generic activities may be organized in different ways and described at different levels of detail depending on the type of software being developed.

Software Process

As described above, a software process is a set of related activities that leads to the production of a software system. These activities may be followed to develop software from scratch or to integrate/extend/modify existing systems. These activities are not necessarily followed in a sequential way (e.g., some of them may be executed in parallel). Overall, software processes are categorized as either plan-driven processes or agile processes [11]. Plan-driven processes are

processes where all process activities are planned in advance and followed in a specified order. In agile processes, planning is incremental and it is easier to change the process to reflect changing requirements/attributes. The most-well known software processes are the following [11]:

1. *Waterfall model*: it executes the core activities described above (software specification, architecture, implementation, testing, deployment, maintenance) in a row and it considers them as separate phases.
2. *Incremental development*: it interleaves most of the above activities and develops the system as a series of versions (increments), with each version adding functionality to the previous version.
3. *Reuse-oriented software engineering*: it is based on reusable components and focuses on integrating these components into a system rather than developing them from scratch.

Towards abstracting the software-engineering activities that are executed by existing edge-computing approaches, *we use the waterfall software-process as a suitable process for edge computing*. The purpose of the above process is to provide a starting point for practitioners/developers who want to produce software applications deploy-able and run-able to edge infrastructures. The abstract software-process is described in the last section of the current chapter.

Edge Computing

Edge computing has been proposed to mainly provide low latency and location awareness, to support geographic distribution and device mobility, along with the realization of real-time applications. However, the adoption of edge computing has also introduced research challenges [8]. Among others, one of the challenges is how software applications should be developed to be deploy-able and run-able on the edge. The minimum requirement of edge-deployable applications is that they should have *modular architecture*, i.e., applications consist of separate components that are connected together. The components with which end-users interact are known as *front-end* components. The components that have computation and/or storing responsibilities are known as *back-end* components. The current subsection focuses on the multi-tier architecture of edge infrastructure and how modular software is deployed to such an infrastructure.

Multi-Tier Edge Infrastructure & Modular Software

The underlying infrastructure of edge computing generally has a multi-tier architecture [8]. In particular, the three-tier architecture is one of the most widely used architectures in edge computing. The first tier includes end-user devices (e.g., IoT-enabled devices, sensors, smartphones, tablets, smart vehicles). These end devices are often termed as Terminal Nodes (TNs). *TNs usually run the front-end components of modular software (e.g., GUI pages)*.

The second tier that is usually termed edge/fog layer is composed of network devices such as a router, gateway, switch, and access points. These edge nodes can collaboratively share storage and computing facilities. The first and the second tier nodes are usually connected to each other by a single network hop. *Edge nodes usually run the back-end components of modular software*. Back-ends are mainly responsible for analyzing and storing data. Back-ends are usually accessible over the Web and are offered by following the Web-service technology [10]. *Edge nodes may further run software controllers that send data from the application front-ends to the application back-ends* (e.g., by using programming service-clients).

The third tier is optional and includes traditional cloud servers and data-centers that are remotely located from the end-users' devices. This tier usually has sufficient storage and computing resources. If cloud nodes support/participate in the edge infrastructure, *cloud nodes run the back-end components of modular software*. In this case, the interplay between the edge and the cloud node is challenging. In particular, the following cases of interactions usually appear: edge-to-cloud interactions, edge-to-TN interactions, and edge-to-edge interactions. The above interaction cases emerge the following interesting research question: when do the application back-ends run to edge nodes or to cloud nodes and how is it decided? We present in a next section the current approaches that give answers to the above question.

The description of the layered architecture of edge infrastructure (e.g., physical layer, virtualization layers, security layer, transport layer) is out of the scope of this chapter.

Please underline that the concept of fog computing has great similarity to edge computing. Both of the paradigms focus on the devices near/at the edges of the network. Currently there is no universally accepted open standard on what defines the network edge [13]. In general, fog computing can be treated as a special form of edge computing [13]. In particular, an edge infrastructure that makes use of both edge devices and the cloud machines is referred to as fog computing [12]. *We hereafter use the term edge to interchangeably refer to edge computing or fog computing*. Moreover, we survey related approaches and we contribute a software process for both edge and fog computing.

Standardizing Edge Computing

The OpenFog Consortium[1] includes famous companies and academic institutions worldwide that aim at standardizing the concepts used in the field of edge computing and at creating a reference architecture for edge infrastructures. The consortium was founded by ARM, Cisco, Dell, Intel, Microsoft, and the Princeton University Edge Laboratory in 2015. It currently counts 57 members across North America, Asia, and Europe [8]. The current version of the reference architecture is based on eight core pillars that include security, scalability, openness, autonomy, reliability, availability, serviceability, agility, hierarchy, and programmability [8]. *What is currently missing is the software-engineering pillar for edge computing*.

Related Work

There are related studies that survey the research papers in the field of edge/fog computing. Those surveys present the existing approaches from various perspectives that are detailed in the following subsections.

Architectures, Infrastructures and Algorithms for the Edge

The authors of [6] present a comprehensive review of the current literature for edge computing with a focus on architectures and algorithms. According to [6], the architectures of edge infrastructures are either application agnostic or application specific. The application-agnostic architectures have focused on service provision, resource management, communication issues, and cloud-fog federation. The application-specific architectures have focused on healthcare systems, interconnected vehicles, and smart living. Moreover, the underlying algorithms have focused on the data analysis, the data storage/distribution, and the energy consumption.

In a similar vein, the authors of [12] survey and categorize the current architectures for edge infrastructures as follows. Data-flow architectures are based on the direction of movement of workloads and data from the users' devices to edge nodes or alternatively from cloud servers to edge nodes. Control architectures are based on how the resources are controlled, e.g., a single controller or central algorithm may manage the edge nodes or alternatively a distributed approach

[1] https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf

may be employed. Tenancy architectures are based on the support provided for hosting multiple entities, e.g., either a single application or multiple applications could be hosted by an edge node.

[12] further details the parts of the infrastructures for resource management in edge/fog computing. The hardware part includes devices like network gateways, WiFi Access Points, small home servers, cars, and drones. The software part (e.g., operating systems, virtualization software) runs directly on hardware resources and manages resources. The middleware part runs on an operating system and provides complementary services that are not supported by the software part.

Finally, [12] discusses four algorithms for resource management: (i) discovery algorithms for identifying edge resources, (ii) benchmarking algorithms for capturing the performance of resources for decision-making to maximize the performance of deployments, (iii) load-balancing algorithms for distributing workloads across resources, and (iv) placement algorithms for identifying resources appropriate for deploying a workload.

The authors of [13] further focus on the architecture design and the system management of peer-to-peer edge computing, mobile grid edge computing, and mobile crowd edge computing.

Software-Defined Networking for the Edge

The survey in [7] presents Software-Defined Networking (SDN) approaches for edge computing. SDN is an edge solution that hides all of the orchestration and the management tasks from application developers and service providers. In particular, the complexities resulting from deploying the cloud-like services at the edge can be solved by the SDN control mechanism. All of the data flow management, the service orchestration and other management tasks are accomplished by the SDN controller. For instance, SDN deals with service commissioning and migration when the target edge devices are occupied. In this case, SDN decides where to commission the service based on various performance factors such as server utilization and network conditions.

Network Applications for the Edge

[8] surveys the network applications that run at the edge. According to [8], computing and storage devices in data centers are interconnected by Data Center Network (DCN). DCN is related to the network topology, the network protocols, and routing/switching equipment. Virtualized DCN

includes servers, routers, switches, and links that are virtualized. Some of the requirements of traditional data centers (e.g., performance isolation, server utilization) can be met by server virtualization technologies (e.g., VMware and Xen). However, some other requirements of data centers (e.g., application deployment, management flexibility) cannot be easily addressed by the above technologies.

Edge Computing for Smart Cities

[9] focuses on the aspects of the device connectivity and configuration aspects of edge computing for smart-city applications. In particular, several types of edge devices exist for smart cities (e.g., sensors, mobile devices, smart watches). Each of these devices can become a leaf node in a software application (e.g., IoT application) to perform edge analytics. The tight limits on the power, memory, and processing resources of those devices lead to strict upper-bounds on state, code space, and processing cycles, making optimization of energy and network bandwidth usage a dominating factor.

According to [9], the least implemented features of edge computing are the multi-protocol support at application level, the context discovery/awareness, and the semantic annotation. Regarding security, existing approaches propose a cloud-based solution that implements user-behavior profiling and can be used to mitigate attacks that are focused on data theft.

Edge computing for smart cities should offer solutions for the following characteristics of smart cities: smart economy, smart people, smart governance, smart mobility, smart environment and smart living. Smart cities are required to accommodate growing populations and their needs with limited resources. Smart cities also need to make sure the limited edge resources do not run out. The best strategy to achieve this is to use resources in the most efficient and optimum ways. To do this, it is essential to understand how cities and its citizens behave and consume resources. Edge computing can bring efficiency and sustainability to the above tasks.

Cyber Security and Privacy for the Edge

The authors of [14] discuss techniques to address cyber security and privacy challenges for edge computing. Those techniques concern cyber security and privacy related to identity authentication, access control, intrusion detection, resilience to sybil attacks, trust management, transient storage, and decentralized computation.

Software Engineering for the Edge

Overall, we observe that the related surveys present the existing edge-computing approaches from the following perspectives: architectures, infrastructures, algorithms, software-defined networking, network applications, smart cities, cyber security and privacy. Thus, the current surveys do not cover the software-engineering aspects of the edge-computing approaches.

Software-Engineering Aspects of Existing Approaches

Existing edge-computing approaches engineer software applications to make them deploy-able and run-able to edge infrastructures. Those approaches may execute all or some of the software-engineering activities that were described in a previous section. We categorize below existing approaches with respect to the above activities. Please note that we indicatively present edge-computing approaches for each activity. *The current chapter is by no means a complete survey of the existing approaches*.

Software Application & Edge Infrastructure Specification

The existing approaches take as input the QoS attributes of software applications, like response time, availability, cyber-security, hardware (e.g., memory, hard-disk) requirements, software requirements (e.g., operating system, database management system), etc. For instance, the approaches allocate the components of software applications to machines only if the QoS attributes of application components (e.g., CPU) can be med by the machine characteristics (e.g., CPU cores) [1].

Other approaches consider the *non-functional attributes* (e.g., performance) *of the underlying edge infrastructure*. For instance, the approaches allocate the components of software applications to machines only if the overall performance of the machines is kept high [1].

Architecture & Implementation of Software Applications

From architecture and implementation perspectives, existing approaches can be organized into the following categories: application *partitioning* approaches or application *offloading* approaches.

Partitioning Software Applications

The approaches of this category assume that *parts* of (some or all of) the application components are executed in edge or cloud machines only if the applications benefit from the remote execution. Remote execution is usually adopted to support the resource-constrained edge machines with energy savings on parts that benefit from the remote execution (e.g., [5]). However, these approaches usually rely on programmers who should specify how and when to *partition* and *modify* the source code of the application components.

Offloading Software Applications

The approaches of this category offload the execution of (some or all of) the application components from edge machines to cloud machines and vice versa (e.g., [15]). The advantage of the offloading approaches is that they do not depend on programmers because applications do not need to get modified. To automate the offloading decision-making, [16] proposes the conversion of the application components to autonomous ones that can build predictive models of the application performance and of the underlying machines.

Deployment of Software Applications

Existing approaches generate deployment plans that map application components to edge/cloud machines. We divide existing approaches into two categories: (i) the approaches that produce deployment plans to meet the QoS requirements of software applications; (ii) the approaches that produce deployment plans to consider the performance requirements of the underlying edge infrastructure.

QoS Requirements of Software Applications

QoS requirements of software applications are (ranges/sets of) values on the QoS attributes of the applications (e.g., response time, availability, etc.). Such requirements are taken into account during the initial deployment of applications and may be dynamically validated during the execution of applications.

[1] proposes an automated mechanism for generating eligible deployment plans, i.e., the plans meet the application requirements for the latency and the bandwidth of edge-to-edge, edge-to-cloud, and cloud-to-cloud communication links. Given that the mapping of many back-ends to many machines is an NP-hard problem (subgraph isomorphism problem), [1] proposes a heuristic to get approximate solutions.

[17] proposes an automated mechanism for generating deployment plans that minimize the delay of an edge-cloud infrastructure to serve application requests. To approximate optimal deployment plans, [17] adopts a mixed integer nonlinear programming technique.

Performance of Edge Infrastructure

The performance of edge infrastructure mainly depends on two factors: (i) the amount of the computing resources of an infrastructure that are consumed by the deployed applications; (ii) the delay of infrastructure to service application requests.

The service delay of an edge infrastructure mainly depends on the workload that serves at unit time [18]. The infrastructure performance is dynamically checked in this approach at the execution time of applications. Central mechanisms (proposed by the approach) are used to coordinate and control the initial and the runtime performance of applications.

[19] proposes the generation of deployment plans that reduce the power consumption of applications in edge machines. [20] builds deployment plans that meet the application requirements about the power consumption. In particular, [20] maps application back-ends to machines via checking (traversing) paths of the topology graph of the underlying edge infrastructure. [21] produces deployment plans that consider both service delay of the edge infrastructure and the power consumption from the applications. [21] approximates optimal deployment plans balancing service time and power consumption.

Maintenance of Software Applications

Existing approaches for deploying and executing applications via edge infrastructures adopt central mechanisms for controlling the execution of the applications. However, a central control mechanism does not scale well with the ever-increasing numbers of the deployed applications and the edge devices. Moreover, most of the existing approaches do not offer seamless switching of the execution of application components between edge and remote machines. On the contrary, the switching is performed at deployment time, suspending the application execution, redeploying the application components, and resuming app running.

As we previously discussed, there are approaches that generate initial deployment plans, i.e., mappings between application components and edge/cloud machines. These plans aim at

meeting the QoS requirements of applications and/or considering the performance of the underlying edge infrastructure.

On top of the above approaches, there are approaches that (re-)generate the deployment plans at the runtime of applications. Those approaches remap/redeploy application components to other machines if the QoS requirements are violated or the infrastructure performance is degraded. These approaches deal with the runtime modeling of the QoS attributes of the applications, along with the adaptation of the deployment plans of the applications by using the above models.

QoS Requirements of Software Applications

The runtime validation of QoS requirements for time-critical applications is even more crucial than other application types because time-critical applications offer their functionality only if time-related QoS requirements are met (e.g., smart traffic-light, early disaster-warning, cyber-physical applications).

[22] proposes an approach for the QoS-driven migration of applications. It migrates application back-ends from an edge device to another when the latency of a back-end exceeds a time threshold. The target edge node is selected via checking (traversing) paths of the tree structure of the edge infrastructure. This approach does not build predictive models. On the contrary, it uses the current latency of back-ends to migrate them to other machines.

[23] presents an approach for the runtime modeling of the empirical performance of applications in function of (software or hardware) control knobs (managed by the cloud providers to support QoS) and environmental primitives (e.g., number of service requests). To build performance models, [23] uses machine-learning algorithms. The determined models are related to the response time, throughput, availability, and reliability of applications.

[24] builds models and verifies rule-based auto-scaling policies for cloud-based applications in function of discrete elapsed time. To build and verify models, it uses a discrete-time Markov chain. The determined models are related to the CPU utilization of the applications.

[16] proposes the runtime modeling of the response time of applications in function of the input data and the characteristics of the underlying machines. The constructed mathematical models are then used to decide the proper binding to the edge or to the cloud.

Performance of Edge Infrastructure

The infrastructure performance should be checked during the execution of applications. The check of the infrastructure performance for time-critical applications is more crucial than the other application types because low performance may negatively affect the time-related requirements of the applications.

[25] proposes an analytical model for measuring the service delay of an edge infrastructure in function of the elapsed discrete time. Based on this model, [25] offloads computation from an edge node to another in an online manner towards reducing the service delay perceived by end-users.

[18] proposes indexing of edge machines with respect to the trade-off between the service delay and the power consumption. The determined indexing models are in function of the elapsed discrete time. When offloading is needed, the edge device with the smallest index is selected.

[15] proposes an offloading mechanism for migrating application back-ends. This mechanism is initiated by the end-users who provide their budgets. Then, the offloading mechanism maps (one-to-one bipartite mapping) back-ends to machines based on the computation costs of the machines.

Abstract Software Process for Edge Computing

We present in this section the software-engineering activities that should be followed by practitioners for deploying and running modular software in edge infrastructure. In particular, we present a *waterfall software process* for edge computing (Figure 1).

In a nutshell, the first activity of the process includes the specification of the capabilities of the machines of the target edge infrastructure and the specification of the software/hardware requirements of an edge-enabled software application. The second activity concerns the definition of the edge-deployable modular architecture of software application as a set of service components. The third activity is related to the microservice implementation of service components. The fourth activity has to do with the generation of edge-eligible deployment plans of service components to machines, along with the deployment of those components to the machines. The fifth activity includes the maintenance activities that are needed to meet the runtime requirements/attributes of application.

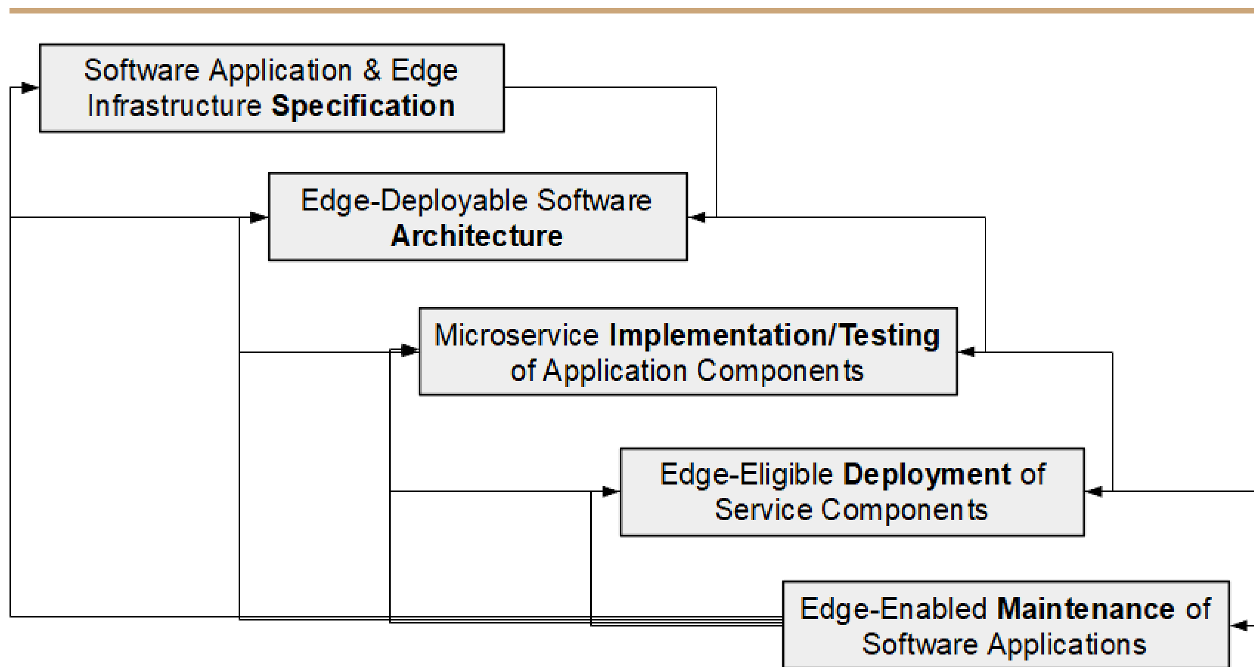

Figure 1. The waterfall software process for edge computing.

Software Application & Edge Infrastructure Specification

Specification of Edge-Deployable Modular Software

An edge-deployable software application consists of a set of software components (a.k.a., modular architecture). To be accessible over the Web, each component is usually developed as a Web service [10] and exposes its programming Web API (a.k.a., RESTful API [26]). Web APIs are lightweight alternatives to WSDL/SOAP-based Web services and use REST as the communication protocol and JSON as the content format[2].

To avoid hardware/software incompatibilities between the service components and the edge machines, service components are encapsulated into containers (e.g., Docker[3]).

Each service component usually has its own resource requirements and possibly further software requirements. In particular, each service component may have the following requirements:

- hardware-resource requirements
    - e.g., CPU, Hard disk, RAM

[2] https://www.json.org/json-en.html
[3] https://www.docker.com/

- software-capabilities requirements
    - operating system (e.g., Linux, Windows)
    - software platform (e.g., JDK, .NET).

To put it more formally, a (front-end and back-end) service component is defined as follows.

Definition 1. *Edge-deployable service component* is defined by the tuple C = (HR[], SC[], URI), where HR[] denotes the array of the hardware-resource requirements, SC[] denotes the (possibly empty) array of software requirements, and URI denotes the endpoint address of the service deployed to an edge/cloud machine.

Overall, modular software consists of a set of service components. It is challenging though the automated generation of the specification of the hardware-resource requirements of edge-deployable service components.

Specification of Edge Infrastructure

A virtual or a physical machine of an edge infrastructure is characterized by the hardware resources and the software capabilities of the machine, as defined below.

Definition 2. *Edge/cloud machine* is defined by the tuple M = (HR[], SC[]), where HR[] is the array of the hardware resources of the machine and SC[] is the array of the software capabilities of the machine.

Overall, an edge infrastructure consists of a set of machines that can be modeled as an undirected graph as follows.

Definition 3. *Edge infrastructure* is defined by the undirected graph I = (M[], L[]), where the array M[] of the graph corresponds to the edge/cloud machines and the array L[] of the graph contains the edge-to-edge, edge-to-cloud, and cloud-to-cloud communication links between the machines.

It is challenging though the automated dynamic management of edge infrastructures.

Edge-Deployable Modular Software

An edge-deployable software application can be defined as a composition of service components (a.k.a., modular architecture). The service composition can be modeled as a directed graph as follows.

Definition 4. *Edge-deployable software application* is defined by the tuple A = (C[], D[]). The array C[] of the graph corresponds to the set of the (front-end and back-end) service components of the application. The array D[] of the graph corresponds to the dependency links between the components. A component depends on another component (having a dependency link) if the former component invokes the Web API of the latter component.

As described in a previous section, the architecture of an edge-enabled software application can follow the partitioning technique or the offloading technique.

Architecture of Edge-Partitioned Modular Software

The approaches of this category assume that while the components of an application have been deployed to edge machines, parts of the components are executed in powerful (e.g., cloud) machines only if the applications benefit from the remote execution. The architecture of edge-partitioned software application is defined below.

Definition 5. *Architecture of edge-partitioned modular software* is defined by the tuple A = (C[], D[]). The array C[] of the graph corresponds to the service components of the application. Each component C consists of two partitions, C = (CE, CC). The partition CE represents the edge partition of the component. The partition CC represents the cloud partition of the component. The array D[] of the graph corresponds to the dependency links between the components. Each dependency link is instantiated by two graph edges, D = (DE, DC). The edge DE represents a dependency link of the edge partition of the component. The edge DC represents a dependency link of the cloud partition of the component.

Architecture of Edge-Offloaded Modular Software

The approaches of this category offload the execution of some (or all) of the components of applications. The architecture of edge-offloaded software applications is defined below.

Definition 6. *Architecture of edge-offloaded modular software* is defined by the tuple A = (C[], D[]). The array C[] of the graph corresponds to the service components of the application. Each component C consists of at most two instances, C = (CE, CC). The instance CE represents the edge instance of the component. The instance CC represents the cloud instance of the component. The array D[] of the graph corresponds to the dependency links between the components. Each dependency link is instantiated by two graph edges, D = (DE, DC). The edge

DE represents a dependency link of the edge instance of the component. The edge DC represents a dependency link of the cloud instance of the component.

It is challenging though the (semi-)automated specification and maintenance (of high quality) of the architecture of edge-partitioned and edge-offloaded software applications.

Microservice Implementation/Test of Modular Software

The implication behind the implementation of the modular components of software applications is that the components are light-weight enough so as the components are deployable to resource-constrained edge machines. Services whose footprint is very small are known as microservices [27]. Microservices have emerged in recent years as the main target type of service-oriented computing. However, as surveyed in [28], the development of complex software applications in the form of microservices is currently performed by software engineers in a manual way.

It is interesting to mention that the platforms that exist for implementing edge-enabled software applications assume that the application components have already been implemented as microservices and wrapped by containers. For instance, SONM platform[4] is a powerful distributed Web platform for hosting software applications that offers a pure edge infrastructure (as an alternative of a cloud or edge-cloud infrastructure). To host an application component in SONM, developers should prepare a Docker container with the component and upload the container to the Docker storage[5].

Edge-Eligible Deployment of Service Components

Edge-Eligible Deployment Plan

An edge-eligible software application is an application whose components can be deployed to the machines of an edge infrastructure. To keep track of the mapping of software components to machines, a deployment plan is generated. A deployment plan usually assumes that all software components of an application have been deployed to machines. A deployment plan further assumes that an eligible mapping has been produced. Eligible is the mapping that the

[4] https://docs.sonm.com
[5] https://docs.sonm.com/getting-started/as-a-consumer#Task_execution

requirements of the deployed software components are met by the used machines, as defined below.

Definition 7. *Edge-eligible deployment plan* is an one-to-many boolean mapping table *map[machines, components]* of (edge and cloud) machines that host application components. The value (true or false) of an element of a boolean table means that a machine hosts a software component, as formally specified below.

$$\sum_{M=1}^{N} (C.HR[i] \, * \, map[M,C]) \leq M.HR[i] \; and \; C.SC[j] \in M.SC[j]$$

where M is a machine, N is the number of the used machines, C.HR[i] is the value of the i-th hardware-resource requirement of a component *C*, M.HR[i] is the value of the i-th hardware-resource capability of a machine M, C.SC[j] is the value of the j-th software capability of a component C, and M.SC[j] is the value of the j-th software capability of a machine M.

From Deployment Plan to the Deployment to Machines

After having decided/generated a deployment plan, it is then specified in a machine-readable format. The commonly used format is YAML[6]. In particular, deployment engines used by edge and/or cloud platforms usually accept as input YAML documents.

To automate the cloud deployment and the management (e.g., scaling) of the deployed containerized services, Kubernetes platform has been widely used[7]. More recently, KubeEdge platform has been proposed[8]. The key goal for KubeEdge is to extend the Kubernetes ecosystem for the cloud and the edge.

It is challenging the scalable deployment of service components to a large number of edge machines whose resources are potentially controlled by a federation of KubeEdge platforms.

Edge-Enabled Maintenance of Software Applications

The previously mentioned deployment plans are initial mappings between application components and edge/cloud machines. These plans aim at meeting the QoS requirements of applications

---

[6] http://cloud.google.com/appengine/docs/flexible/java/yaml-configuration-files

[7] https://kubernetes.io

[8] https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro/=

and/or considering the performance of the underlying edge infrastructure. However, the deployment plans should be adapted at the application runtime if the QoS requirements are violated or the infrastructure performance is degraded.

QoS-Based Maintenance of Service Components

To enable the redeployment of application components based on their QoS values, we extend the definition of an application component as follows (Def. 8 is an extension of Def. 1).

Definition 8. *Edge-redeployable service component* is defined by the tuple C = (HR[], SC[], QoS[], URI), where QoS[] is an array of the runtime values of QoS attributes (e.g., response time, etc.).

Given the updated Definition 8 of a service component, we define below the redeployment plan of service components based on their runtime QoS attributes (Def. 9 is an extension of Def. 7).

Definition 9. *Edge-enabled QoS-based redeployment plan* contains an one-to-many boolean mapping table *map* of (edge and cloud) machines that host service components and minimize the values of the QoS attributes of the service components as follows:

$$\sum_{M=1}^{N} (C.HR[i] * map[M,C]) \leq M.HR[i] \text{ and } C.SC[s] \epsilon M.SC[s] \text{ and } C.QoS[j] \text{ is } min,$$

where *M* is a machine, *N* is the number of the used machines, C.HR[i] is the value of the i-th hardware-resource requirement of a component C, M.HR[i] is the value of the i-th hardware-resource capability of a machine M, C.SC[s] is the value of the s-th software-capability of a component C, M.SC[s] is the value of the s-th software-capability of a machine M, and C.QoS[j] is the value of the j-th QoS attribute of a component *C*.

Performance-Driven Maintenance of Edge Infrastructure

To enable the redeployment of application components based on the performance of the underlying edge infrastructure, we extend the definition of a machine as follows (Def. 10 is an extension of Def. 2).

Definition 10. An *edge/cloud performance-aware machine* is defined by the tuple M = (HR[], SC[], P[]), where P[] is the array of the runtime performance of the machine.

Given the updated Definition 10 of a machine, we define below the redeployment plan of service components based on the runtime performance of the machine (Def. 11 is an extension of Def. 2).

Definition 11. A *performance-driven redeployment plan* is an one-to-many boolean mapping table of (edge and cloud) machines that redeploy service components by minimizing the values of the performance of the machines as follows:

$$\sum_{M=1}^{N} (C.HR[i] * map[M,C]) \leq M.HR[i] \; and \; C.SC[s] \in M.SC[s] \; and \; M.P[j] \; is \; min$$

where *M* is a machine, *N* is the number of the used machines, C.HR[i] is the value of the i-th hardware-resource requirement of a component C, M.HR[i] is the value of the i-th hardware-resource capability of a machine M, C.SC[s] is the value of the s-th software-capability of a component C, M.SC[s] is the value of the s-th software-capability of a machine M, and M.P[j] is the value of the j-th performance attribute of a machine M.

It is challenging though the scalable (e.g., for a large number of machines) maintenance of the deployment plans of software applications to meet the runtime QoS requirements of applications and to achieve high runtime performance for the edge infrastructures.

Conclusions

The distant cloud is not always the best solution for time-sensitive and location-aware software applications. Edge computing has been recently introduced by both industry and academia to quench the need for a computing paradigm close to mobile devices. In this chapter, we specified the core concepts of the software-engineering process and how software applications can be deployed to the multi-tier architecture of edge infrastructure. We highlighted the software-engineering aspects of existing edge-computing approaches. Finally, we abstracted a software-engineering process suitable for edge computing and we outlined the research challenges that exist in this process.

References

1. A. Brogi, S. Forti, QoS-Aware Deployment of IoT Applications Through the Fog, IEEE Internet of Things Journal, 4 (5), pp. 1185-1192, 2017

2. A. McAfee, E. Brynjolfsson, T.H. Davenport, D. Patil, D. Barton, Big data: the management revolution, Harvard Business Review, 90 (10), pp. 60-68, 2012.
3. Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, Jason P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture, 98, pp. 289-330, 2019
4. OpenFogConsortium, Openfog reference architecture for fog computing, Available: https://www.openfogconsortium.org/ra, February 2017.
5. B. G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: elastic execution between mobile device and cloud, European Conference on Computer Systems, European conference on Computer systems, pp. 301-314, 2011
6. C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, P. A. Polakos, A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges, IEEE Communication Surveys and Tutorials, 20, 1, pp. 416--464, 2018
7. A. C. Baktir, A. Ozgovde, C. Ersoy, How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions, IEEE Communications Surveys and Tutorials, 19, 4, pp. 2359-2391, 2017
8. M. Mukherjee, L. Shu, D. Wang, Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges, in IEEE Communications Surveys and Trends, 20, 3, pp. 1826-1857, 2018
9. C. Perera, Y. Qin, J. C. Estrella, S. R. Marganiec, A. Vasilakos, Fog Computing for Sustainable Cities: A Survey, ACM Computing Surveys, 50, 3, pp. 32:1 - 32:43, 2017
10. T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 2005
11. I. Sommerville, Software engineering, 8th Edition, International computer science series, Addison-Wesley, 9780321313799, 2007
12. C. H. Hong and B. Varghese, Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms, ACM Computing Surveys, 52, 5, pp. 97:1--97:37, 2019
13. C. Li, Y. Xue, J. Wang, W. Zhang, T. Li, Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management, ACM Computing Surveys, 51, 2, pp. 39:1-39:34, 2018

14. J. Ni, K. Zhang, X. Lin, X. Shen, Securing Fog Computing for Internet of Things Applications: Challenges and Solutions, IEEE Communications Surveys & Tutorials, 20, pp. 601-628, 2018
15. D. H. Tran, N. H. Tran, C. Pham, S. M. A. Kazmi, E. N. Huh, C. S. Hong, OaaS: Offload as a Service in Fog Networks, ACM Computing, 99 (11), pp. 1081-1104, 2017
16. D. Athanasopoulos, M. McEwen, A. Rainer, Mobile Apps with Dynamic Bindings Between the Fog and the Cloud, International Conference on Service-Oriented Computing, pp. 539-554, 2019
17. R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption, IEEE Internet of Things Journal, 3 (6), pp. 1171-1181, 2016
18. X. Guo, R. Singh, T. Zhao, Z. Niu, An Index Based Task Assignment Policy for Achieving Optimal Power-Delay Tradeoff in Edge Cloud Systems, IEEE International Conference on Communications, pp. 1-7, 2016
19. A. Brogi, S. Forti, A. Ibrahim, How to Best Deploy Your Fog Applications, Probably, International Conference on Fog and Edge Computing, pp. 105-114, 2017
20. H. Gupta, A. V. Dastjerdi, S. K. Ghosh, R. Buyya, iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments, Software: Practice and Experience Journal, 47 (9), pp. 1275-1296, 2017
21. R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption, IEEE Internet of Things Journal, 3 (6), pp. 1171-1181, 2016
22. E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwalder, Incremental Deployment and Migration of Geo-distributed Situation Awareness Applications in the Fog, ACM International Conference on Distributed and Event-based Systems, pp. 258-269, 2016
23. T. Chen, R. Bahsoon, Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services, IEEE Transactions on Software Engineering, 43 (5), pp. 453-475, 2017
24. A. Evangelidis, D. Parker, R. Bahsoon, Performance modelling and verification of cloud-based auto-scaling policies, Future Generation Computer Systems, 87, pp. 629-638, 2018
25. A. Yousefpour, G. Ishigaki, J. P. Jue, IEEE International Conference on Edge Computing, Fog Computing: Towards Minimizing Delay in the Internet of Things, pp. 17-24, 2017

26. L. Richardson, S. Ruby, Restful Web Services, First Edition, O'Reilly, 9780596529260, 2007
27. S. Newman, Building Microservices, O’Reilly Media, Inc., 1st edition, 1491950358, 2015
28. D. Taibi, V. Lenarduzzi, C. Pahl, A. Janes, Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages, ACM International Conference on Agile Software Development, pp. 23:1-23:5, 2017.