DEPARTMENT OF
COMPUTER SCIENCE

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DIOGO JOÃO DE PAIVA GOMES
Bachelor in Computer Science

DECOMPOSING MONOLITIC COMPUTATIONS FOR EXECUTION ON THE EDGE FOR FUN AND PROFIT

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2022

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

DECOMPOSING MONOLITIC COMPUTATIONS FOR EXECUTION ON THE EDGE FOR FUN AND PROFIT

DIOGO JOÃO DE PAIVA GOMES

Bachelor in Computer Science

Adviser: João Leitão
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2022

Decomposing monolitic computations for execution on the Edge for fun and profit

Abstract

Cloud computing is emerging in the IT industry and is being established as the standard way of providing high quality applications and services, being able to scale when necessary and maintaining low latency. This high availability allows new users and devices to connect which may possibly cause some saturation in the network.

The edge computing paradigm has recently emerged to address raising concerns on the ability of cloud computing infrastructures (and data links connecting end user devices to such infrastructures) to receive, process, and reply to operations generated by edge devices (e.g., sensors or other more sophisticated devices) in an expedited way. One way to mitigate this challenge, put forward by the edge computing paradigm, is to decompose the computations executed by applications in smaller components that can then be pushed into the devices (either on the cloud or edge) that are in close vicinity to the location of data generation and consumption. This allows to remove pressure from network links (by avoiding to move data) and can, in particular cases, speedup the production of results to be consumed in the edge. However, achieving this intuitive vision is cumbered by several challenges, in particular:

1. How to decompose the logic of a (large) computation into smaller units;
2. Understanding the requirements for the correct processing of such smaller units;
3. Define mechanisms to compose the results of the smaller computations;
4. Effectively pushing the computations to the locations where they will be conducted and ensuring their execution despite failures.

In this thesis we will start to pave the way towards such a vision by studying how to decompose computations, classifying relevant properties of a subset of classes of these computations, and exploring the design space of the runtime support to execute smaller units in an integrated way across the cloud and edge. We will focus on settings with multiple cloud datacenters potentially with a few edge locations.

Keywords: Cloud, Edge, IoT, Computation

Resumo

A computação em Cloud (na Nuvem) está a crescer na indústria de Tecnologias de Informação e está a ser estabelecida como o padrão para fornecer aplicações e serviços de qualidade, sendo capaz de serem escalados quando necessário, mantendo sempre uma rápida resposta. Esta grande disponibilidade permite que novos utilizadores e dispositivos se conectem de uma forma exponencial, causando saturação na rede.

O paradigma da computação em Edge (na fronteira) começou a emergir para solucionar o problema da disponibilidade da infraestrutura de uma arquitetura em Cloud para receber e processar informação e operações recebidas por dispositivos de Edge (sensores ou outros dispositivos mais sofisticados) de uma forma rápida e eficiente. Uma forma de mitigar este problema, com ajuda da computação em Edge, passa por dividir as computações que são executadas pelas aplicações em blocos de menor dimensão sendo depois processados perto da localização onde a data é criada ou consumida. Este fator permite aliviar e remover pressão das conexões entre vários dispositivos na rede, evitando transferir dados e por sua vez aumentar a velocidade de saída de resultados dos dados que são consumidos.

Apesar de tudo, alcançar esta visão traz alguns desafios:

1. Como decompor a lógica de grandes computações em unidades mais pequenas;
2. Entender os requisitos para um correto processamento de cada unidade.
3. Definir mecanismos que possam compor os resultados das sucessivas unidades.
4. Passar efetivamente as computações para uma localização mais ideal onde serão tratadas e assegurar a sua execução apesar da possibilidade de falhas.

Nesta dissertação iremos começar a traçar o caminho para esta visão, estudando como se deve decompor computações, classificando as suas propriedades e explorar as possibilidades para a execução de unidades mais pequenas de uma forma integrada. Iremos focar em cenários com vários datacenters em cloud e vários dispositivos de Edge.

Palavras-chave: Cloud, Edge, IoT, Computação

Contents

List of Figures

1 Introduction

1.1 Context

Nowadays, everyone is used to carrying a smart device in their pocket throughout the day. These devices are used for checking the news, communicating with friends, playing games, or even, as a work essential tool. Our smartphones are an example of a device that is constantly posting and requesting information from a remote location (not on their local storage) and this was made possible by the appearance of Cloud Computing [43], an almost infinite scalable infrastructure that allows users and companies to use resources and services in the palm of their hands, without having the need for big devices with very computational power.

Although Cloud computing seems to be the way to go for our daily usage, it will not be a viable solution to build fast and efficient systems in the near future. The increase of Internet of Things (IoT) [49] applications and mobile devices is leading to a fast increase in the amount of data created and transmitted, which is consequently increasing the time required for the cloud data centers to process new requests. Another problem that is faced by the cloud computing paradigm is that we may have some situations where the amount of data produced at a certain time is bigger than the capacity of the transmission itself, as the network capacity is not evolving at the same rhythm as the need of more edge centric [45] applications.

Edge computing [45] is a new concept that is emerging as a way to mitigate the limitations described above, as it is considered a way to perform computations outside of the cloud data centers and closer to the devices or end-users. This is a very broad concept, as the edge can be considered as being composed of any device with the ability to communicate between the source of the data and the cloud and having the ability to perform data computations (analyzing, filtering, flagging, etc). These devices might include sensors, actuators and other endpoints, as well as IoT (Internet of Things) servers and routers.

By moving some computations from the cloud to the edge, especially when dealing with large amounts of data, it is possible to lower the network traffic and decrease the

latency for the end-user. It can also bring some security advantages as the edge can be managed by the organization itself (and not the cloud providers), and certain security policies can be applied.

1.2 Motivation

Current systems and applications that use the Cloud for storage and computation will always have its benefits but there are also ways to improve overall latency and user experience by taking advantage of edge resources.

As mentioned above, clients that are far away from the Cloud datacenter location will experience a larger latency than if they were closer to the datacenter. Cloud computing tries to minimize these issues by replicating everything in multiple locations around the world, hoping to have them closer to each user or client.

With the increasing amount of data being produced and transmitted, mostly by the emergence of IoT (Internet of Things) devices, the bandwidth of the connections between the data points and the Cloud will eventually become saturated [30]. This issue can also affect user applications and games where the user needs to wait for a response to their request to the Cloud datacenter.

With the use of Edge computing, applications can avoid unnecessary communications with the Cloud, by communicating with an intermediate node [47], executing most of the computations there. As this is mainly still a concept, there is a need to improve this kind of architectures as their current design also presents some flaws and challenges that must be addressed. We want to help developers and organizations during the development and implementation of new edge applications that are constantly receiving and processing data from multiple sources by simplifying their process.

1.3 Objective

In this thesis we will analyze how cloud computing can affect the latency of Internet of Things (IoT) devices and the data they produce. We will approach Edge computing as a possible solution for this problem, trying to improve latency and costs.

We will also study different ways to approach computation processes and how we can split monolithic tasks into smaller ones, checking if it is more efficient than the traditional way. There is also a need to improve the way that edge applications use their resources. With this in mind, we have the objective to develop a framework with the following goals:

- Help developers and organizations to create or improve edge applications by taking advantage of all the available resources with less human intervention.
- Provide mechanisms for declaring and introducing new devices at any time.

- Optimize all the resources available, taking into account their capabilities and the latency between nodes.
- Allow developers to compute data with their custom code within a set of possible functions that may include summing, subtracting, minimizing, etc.
- Minimize the amount of data that is sent across the network by filtering it directly at the source.

1.4 Expected Contributions

With this thesis we expect to provide the following contributions:

- Define different computation classes with specific properties and how different outputs can be combined.
- The design and implementation of a framework for improving edge applications by allowing to take advantage of all available resources with less human intervention.
- A comparison between current similar systems and our self managed system.

1.5 Document Structure

The rest of this document is organized as follows:

- **Chapter 2:** Presents Cloud and Edge Computing in more detail with the current solutions available and why there is a need to evolve even more into the concept of Edge computing.
- **Chapter 3:** Describes our proposed solution with the details and the approach for achieving it and evaluating accordingly, as well as a schedule with the planning for the next months.

2

Related Work

In this chapter we will address cloud computing and its limitations as well as the emerging concept of edge computing, explaining its properties and how it can help to mitigate some of the increasing limitations of using remote data centers far from the end users and devices.

We will discuss some technologies and services that are currently available for public use, including the major Cloud providers like Amazon, Microsoft and Google. We will also study and discover different aggregation methods and data processing techniques in distributed computing.

2.1 Cloud Computing

The notion of network based computing started in the decade of 1960, with the use of mainframe computers which were used by large organizations as a way of allowing their users to access data and other resources. As the years progressed, the internet evolved and the need for wireless devices increased. In 2006 big companies like Google and Amazon introduced what we now call the modern cloud computing, a way of delivering computing services like servers, storage, databases, networking, software, analytics and intelligence over the internet. According to The National Institute of Standards and Technology (NIST) [19] cloud computing *"is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*.

Cloud computing also allows companies to pay only for the resources they consume, unlike for example the mainframe computers that usually had expensive licenses and had to be maintained manually. With this in mind, from a business perspective, cloud computing is showing as the way to go as it allows to save a lot of costs and time by eliminating the need to plan ahead for provisioning and eliminating the use of unnecessary resources.

2.1.1 Concepts

Cloud computing was built in a way that can simulate our own devices and allows for applications to be deployed by using virtualization [36]. When a new service is deployed in the Cloud, it is provided with a virtual ecosystem of hardware and software. This allows users from different devices to access the same hardware in the Cloud. Virtualization comes with different features [36]:

1. **Flexibility:** Companies and organizations have the ability to share resources without sharing critical information and data across the system
2. **Security:** The risk of multiple attacks in case of a vulnerability is reduced by the ability of isolating hardware and software by virtualization
3. **Data protection:** Virtualization allows companies to move whole applications and services to several locations in a very simple way because it consolidates servers and resources into files.
4. **Cost effective:** Reduces hardware requirements and the need for maintenance
5. **Access control:** Virtualized hardware infrastructure is secured by custom rules

Having a virtualized system means that the Cloud providers can easily and automatically scale the virtual machines that are provided for the applications. An application owner can configure the provisioning in a way that it is scaled automatically in case of an increase of service demands and at the same time not being worried about maintenance expenses.

2.1.1.1 Types of Cloud Environments

Although the tendency is to move everything into the Cloud, there are some aspects that need to be considered for different types of usages and needs. Some companies, like in the banking industry, need to keep some information and services running on their own premises. This type of necessity has created 4 main types of cloud computing services in the Cloud environment, each one with its own benefits and disadvantages that need to be considered [34]:

1. **Public Cloud:** Provisioning of services and infrastructure for open use by the general public. This type of cloud provides their users benefits such as no up-front capital costs and the ability to scale on demand, making it a pay-as-you-go service. However, all the infrastructure is under full control by the Cloud provider and customers must accept their terms and conditions.
2. **Private Cloud:** This type of cloud environment is usually secured by a firewall and used by a single organization. It can be a full on-premises service managed by the

organization itself or by a cloud provider. Private cloud solutions offer more control and security making it easier to restrict access to valuable resources. However, the organization is responsible for the management and maintenance of the software and infrastructure, making it less cost effective.

3. **Hybrid Cloud:** A combination of both public and private cloud, that tries to eliminate the limitations of each approach. This design is scalable but still possible to secure important information.

4. **Community Cloud:** Not as common as the other solutions, being a collaborative platform used by different organizations and companies, allowing them to easily share and collaborate at a lower cost. However, this approach may not be the right choice for every organization as it raises some concerns in terms of security, compliance, and performance.

We will later discuss about edge computing and how it is connected to the Cloud. edge computing is a very broad concept that can be integrated in any of these 4 types of Cloud services. Refer to Figure 2.1 for more information.

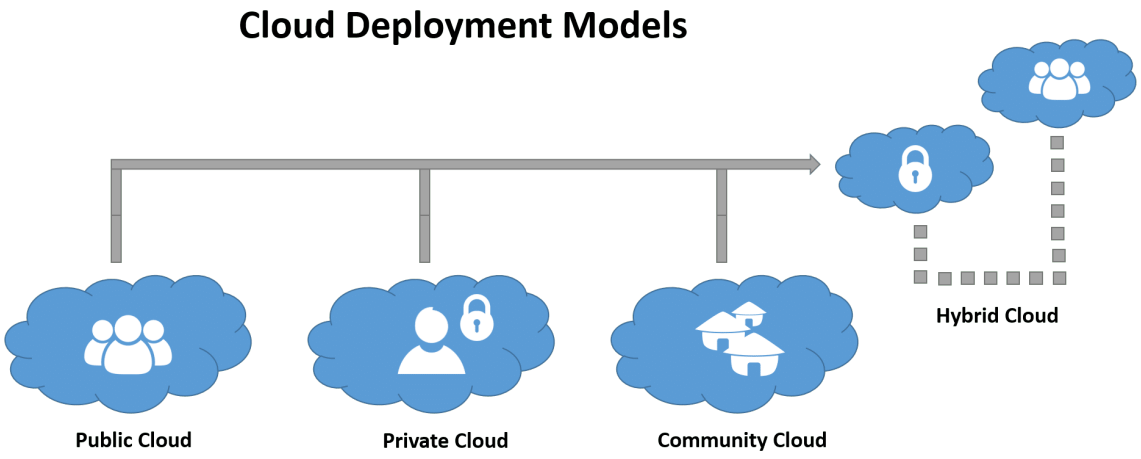

Figure 2.1: Cloud Deployment Models

2.1.1.2 Service Model

Organizations that chose to use cloud services are served with hardware and resources as a service on demand. As customers have the ability to scale when necessary, cloud providers can take part in managing some of their resources. This situation divides the service model into 3 separate types [22]:

1. **Infrastructure as a Service (IaaS):** provides access to a complete infrastructure with servers, storage capacity and networking resources that customers can use as

they would do with their on premises hardware. Customers use the hardware with an internet connection but it is still maintained by the Cloud provider.

2. **Platform as a Service (PaaS):** provides platform level resources for developing, administering and running applications. The provider manages and maintains the hardware and software included in the platform. This is an easier and more cost effective solution then IaaS, if the intention is to run applications without having to administer the OS.

3. **Software as a Service (SaaS):** provides on-demand applications, like Email, social media, and cloud file storage solutions (such as Dropbox or Google Drive) that run on cloud environments and is fully managed by the Cloud provider.

The type of service model chosen by each organization or individual is very important as it can help in the development of their service or application. Later on this document we will also introduce another service called Function as a Service (FaaS) that can be seen as a solution for splitting computations into smaller units, being then processed by independent functions. Refer to Figure 2.4 for more information.

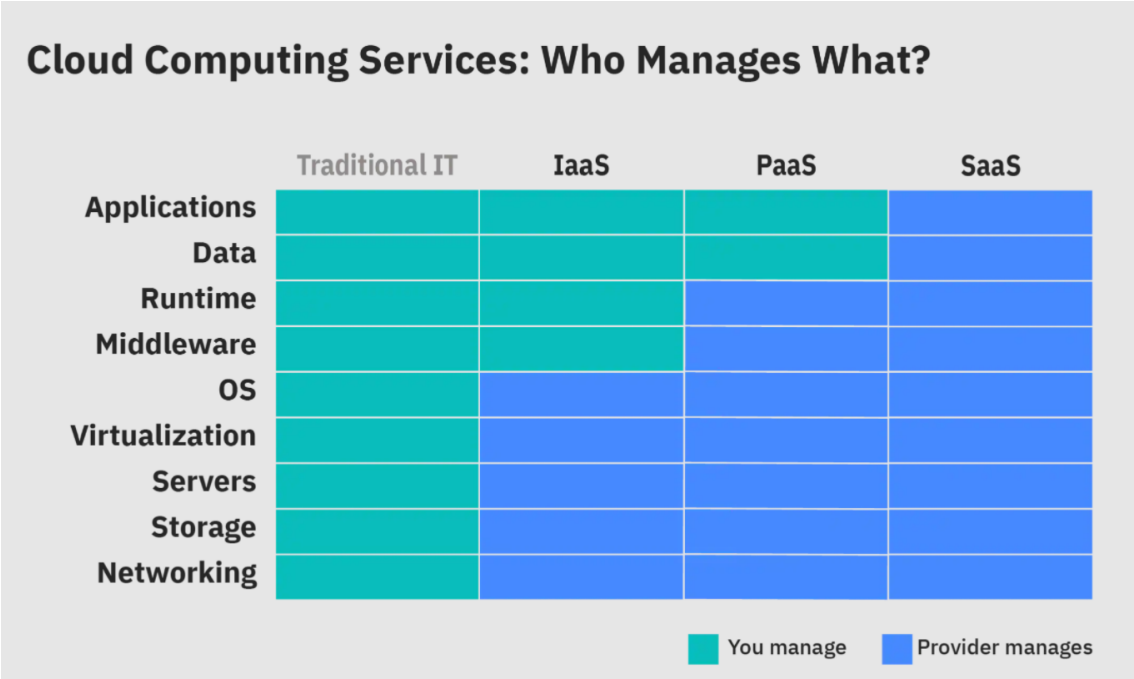

Figure 2.2: Cloud Computing Services: Who manages what?

2.1.2 Examples

There are a vast amount of cloud service providers but by far the most known and used in the industry are Microsoft Azure (Azure Cloud Services), Amazon AWS (Elastic Compute Cloud) and Google Cloud (Google App Engine) [20]. Although they all provide storage,

servers, virtualization and middleware, it is provided in different ways [36]. In this topic we will refer to these computing services and compare their differences:

1. **Google App Engine:** is a Paas (Platform as a Service) [20] that is used all over the world to host and manage web applications. With this type of service, Google deals with the scaling of the resources automatically so that developers focus only on the development itself. This service works together with Google Compute Engine that runs workloads on virtual machines running at physical data centers.

2. **Microsoft Azure Platform:** Microsoft offers Azure Cloud Services as an opponent to other web applications hosting services, with the same characteristics (provisioning, load balancing, auto scaling) [43].

3. **Amazon EC2:** is the oldest of the 3 and has bare metal instances that allows applications to access memory and the processor of the hosting server directly.

Later on in this document we will also refer to other products offered by these cloud providers that are also very helpful in our study to improve latency and user experience.

2.1.3 Discussion

Now that most applications and services work in the Cloud, they are accessible by almost any device that has an internet connection. This also means that data is being stored on remote servers (far from the end-user) and that computations are also being addressed remotely. These cloud servers are usually more powerful than our day to day devices but we are dependent on the latency created by the wireless connections.

What we are trying to understand is that even with lower computational capacity, the devices closer to the source can still provide a better response in dealing with some computation. The problem is that this situation isn't very common nowadays because there isn't much information available or frameworks and products that could help.

With cloud computing, new challenges are emerging with the growth of IoT (Internet Of Things) applications, increasing the amount of data produced and manipulated. Although the Cloud platforms and services are easily scalable, the amount of time it takes to process and communicate the data will increase. Networking connections are increasing exponentially: 2.3, but the bandwidth speed is not following up as it should, becoming a bottleneck.

The cloud will always be part of almost every application and system but it will begin to have a different role as the years and technologies evolve. The Cloud will always be essential for data storage or backups and for any type of computation that doesn't require a real time response.

In the next topic we will discuss Edge Computing, a concept that is trying to help mitigate these issues.

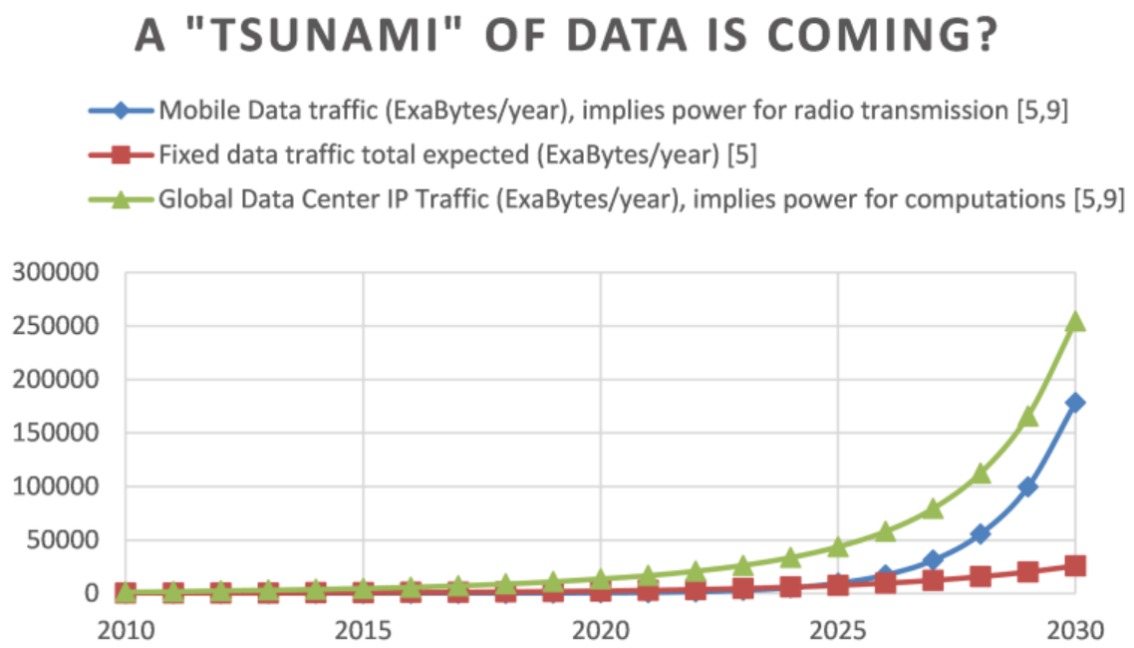

Figure 2.3: A "tsunami"of data is coming

2.2 Edge Computing

In this topic we will address the source of Edge Computing [45], how it is related with the Cloud and how this concept can directly affect cloud applications and their end-users. We will also address some of the flaws and issues associated with this type of architecture and how our framework can improve the integration of edge devices into existing or new applications.

2.2.1 Concept

As the years went by, more applications were transferred and created on Cloud environments. Although this has brought a lot of benefits, there is no way of escaping the fact that the servers hosting the applications are (relatively) far from their end-users. In most cases, latency is a very important factor to take into consideration and with the amount of IoT devices that are transferring and receiving data from the Cloud, it is inevitable that there is an increase in latency. Edge computing is trying to mitigate this by complementing the Cloud architecture with computational nodes closer to the data (users or IoT devices) [48], allowing for some (or all) computation to be made outside the Cloud.

This concept can provide faster, more stable and at a lower cost services. Users should experience a faster and more consistent experience while companies and providers will also benefit from low latency and the possibility of real time monitoring. With this, it is possible to reduce network costs by avoiding possible bottlenecks on the bandwidth, limiting service failures and a better control on sensitive data (it is possible to manage

specific data on private networks using edge nodes, without reaching the Cloud), allowing companies to enforce specific security policies.

With a faster and more stable service, it becomes possible to conduct big data analytics in real time, allowing to make changes and decisions as the data is being processed. Since the data is being handled close to the origin, it isn't compromised by transference latency. Having the possibility to collect and analyze data in real time, contributes to the appearance of AI/ML services and applications, like text translation (with an image) or autonomous driving. We are going to go more in depth with some examples on the next topic. Applications and services can take advantage of edge computing when taking data from many data points and using them to create new information and make predictions and decisions.

Big data applications (with a lot of data processing) are split into several different stages that are performed possibly in different locations (physically). The first stage is the data ingestion, in which the data is gathered by the several producers and transported. The Edge concept [45] takes part at the ingestion stage. After this, data is analyzed (used for machine learning for example), usually at the public cloud, or if there are any security constraints, at the private cloud (or mainframe). After this stage, the machine learning models are sent back to the Edge for runtime analysis. This is only possible because of the proximity between the Edge and the data producers.

Edge computing is a very important step to introduce a hybrid cloud vision that offers a good overall operation and in some cases a better user experience. Currently there aren't many services that help with introducing an Edge architecture into our day to day applications. Developers and organizations are "on their own" when it comes to this topic, meaning that they have to create and use their own devices or techniques to make this possible.

2.2.2 Edge relevant use cases

As we stated before, many edge use cases exist because of the need to process data locally in real time, when transmitting information and data to a cloud datacenter causes unacceptable levels of latency and possibly higher costs. Every day, more and more companies and services are starting to use edge computing on their daily basis:

- We can take as an example a modern factory that produces cars, in their plant there are hundreds of Internet of Things (IoT) sensors that produce data that needs to be analyzed to prevent any malfunction or for quality control. These sensors generate a big stream of data that we can easily calculate: If each sensor produces 500 bytes per second, having a total of 1000 sensors it produces 43.2 GB of data per day, 302.4 GB per week, 1.34 TB per month and 15.76 TB per year. It's faster and less expensive to process that amount of data closer to the equipment, rather than transmitting it to a remote datacenter first. But it's still desirable for the equipment to be linked through a centralized data platform. That way, for example, equipment

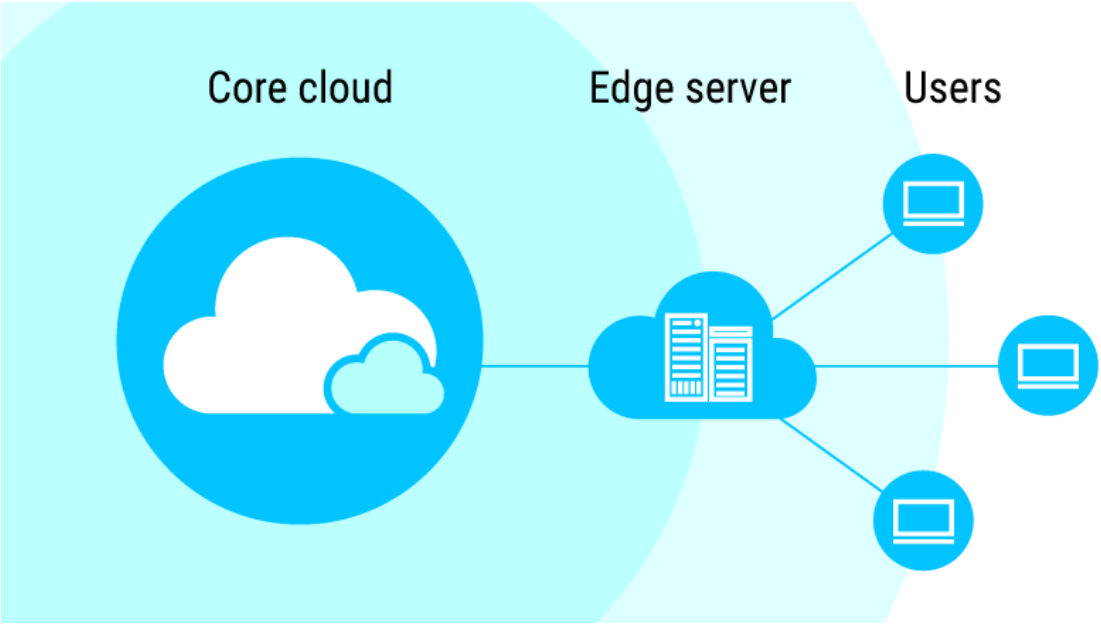

Figure 2.4: Edge and Cloud relation

can receive standardized software updates and share filtered data that can help improve operations in other factory locations. It is also important to create a remote backup of the data produced (either being computed or still raw).

- Another example is the need for real-time data analysis on autonomous vehicles. Being a real health hazard for pedestrians and other cars, this type of vehicles are suited with dozens of sensors that are analyzing the surroundings of the car in real time, producing different types of data that need to be analyzed to make decisions as the car is moving. In this situation, all of the data processing is done inside the car by a local infrastructure with its own operating system but this system is also connected to the Cloud or the Edge to receive software updates and, for example, real-time traffic information.

- With the appearance of 5G communication, the telecommunication providers are running their networks on virtual machines closer to the users. An edge computing architecture allows these providers to keep their software running in several remote locations with their custom security policies. This software running closer to the end-users reduces latency and the overall experience.

- A CDN (Content delivery network) can also be an example of edge computing [49] as a server that is storing content closer to the user. This can be applied in general websites by allowing their users to get the website content in a faster way. The CDN provider will have servers in many locations and they can work as a connection point between different networks at the Edge [40]. This will make it more efficient and faster for data to travel from the initial source to the final destination.

2.2.3 Fog Computing

The meaning of the word Fog, refers to [26] *a mass of cloud consisting of small drops of water near the surface of the earth*. As a distributed computing concept, similar to real life, Fog computing takes place between user or IoT (Internet of things) devices in an infrastructure that connects end devices with cloud data centers. Just like edge computing, Fog is a concept that tries to bring computation closer to the users or devices that are producing data, but in this case, by using the network connection between the devices and the Cloud as a computational force.

Fog nodes [48] are essential for this type of computing and they can be any device with computing power, storage capability and can be placed anywhere with a network connection (5G, Wi-Fi, wired...). These nodes act as a gateway that can receive different kinds of data and address it accordingly. For example, if the service is dealing with real-time data that needs to be analyzed immediately, it should be connected to the nearest node available and on the other hand, if the data isn't time sensitive or doesn't need to be analyzed in real time, it can be sent directly to the Cloud for further analysis or just for storing.

Fog data stream architecture has multiple layers that are used in combination with each other [30]. The **Application layer** is responsible for defining the logic of streaming jobs, the **Processing layer** carries the processing jobs and hosts stream processing engines that will be discussed later like Apache Spark and Apache Flink. The **Data layer** holds data in databases, caches or warehouses and they work together with the processing layer. The **Resource management layer** holds the virtualized system focusing on the utilization of the network, CPU, GPU, memory, storage, etc. and the **Physical layer** that is this situation holds the physical network infrastructure. This infrastructure is likely to be managed by the organization that owns the devices and the software.

As we seen, and to resume, Fog computing has some benefits:

1. **Better response time (latency):** As the initial analysis and processing of data occurs near the source, the latency is reduced and the end-user should experience a more responsive service.

2. **Bandwidth preservation:** The Fog computing concept reduces the amount of data that is sent to the Cloud, thereby reducing bandwidth consumption and related costs. This also improves the latency of the service as the computing doesn't depend on the transference stage.

3. **Multiple networks:** While edge computing generally processes data at a device level, Fog computing computes resources at LAN level (5G, Wi-Fi, wired...)

But also some drawbacks:

1. **Costs:** On one side, Fog computing uses extra infrastructure that is not required in a cloud computing solution and this may add up additional costs. On the other hand, it can also save money by reducing latency and the bandwidth usage.

2. **Physical location:** Similarly to edge computing, it usually depends on a physical location on premises, as opposed to cloud computing.

3. **Uncertainty:** This concept has been idealized many years ago but there are still different approaches on Fog computing by different organizations and providers.

Fog computing can be very helpful if used with our framework to optimize the Edge nodes because its resources can be very dynamic.

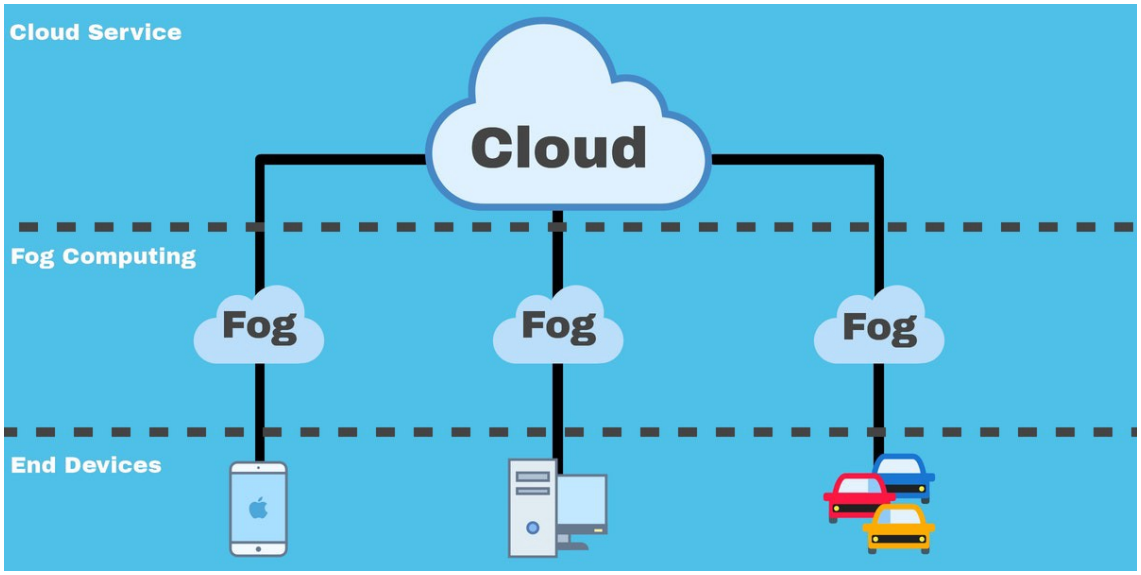

Figure 2.5: Fog Computing

2.2.4 Discussion

In the previous topic we discussed cloud computing and its architecture, uncovering some limitations that could be improved. In this topic we introduced the edge computing concept that can be seen and used around multiple scenarios. Edge computing was born from the need of a faster and better computing experience, meaning that it can be used to connect users or devices and the Cloud. We discovered that in some scenarios where a lot of data is being produced, in environments like warehouses or even web applications where the communication time has a big importance on the response latency, edge computing can help reduce this bottleneck by reducing the length that the data has to travel to reach its destination.

Edge nodes can have a lot of different shapes and sizes or can even be the device that produces the data for further processing. Knowing how the Edge works allows us to

find ways to improve even further the latency of data processing and computation. We decided to focus on edge nodes with little computational power that can't host and run the main processing softwares and with this situation, find a way to split the data into smaller units and then aggregate the final results to reach a final output.

Although these technologies and architectures can help and improve modern applications and solutions, they also have some issues and flaws. Fog computing for example requires specialized hardware that can be costly and doesn't take advantage of the devices that are really part of the Edge. Currently there is no software that helps in decomposing monolithic computations into smaller units to remove pressure from network links. Edge computing can make a big difference on applications if we are able to split computation throughout multiple locations on the Edge, meaning that the data transfer times should become irrelevant.

To better understand computation techniques we will then study and discuss some of the most used software in distributed computing platforms that are offered by the main cloud providers. We also need to understand aggregation protocols and how they can be applied to our context, where we intend to increase the amount of applications and services that use edge computing.

2.3 Kubernetes at the Edge

With the increasing number of edge devices that are able to process data there is a need to improve and unify the way that the data is transmitted and processed.

Kubernetes [50] is an open-source platform that manages workloads and services in the form of containers. It is portable and extensible, meaning that it can be automated. Kubernetes containers [23] are similar to Virtual Machines but they are able to share the same Operating System among applications. Containers, unlike Virtual Machines, are considered lightweight and this is why they can be very helpful in the concept of edge computing. Each container has their own memory, processes and filesystem but they are very portable and can be used across multiple OS's.

Being lightweight, containers have become very popular and they bring a lot of benefits to edge applications if installed properly:

1. Easy to develop and integrate new updates providing a reliable image build and deployment.
2. Allows to get information and metrics about the work state of each container and application.
3. Provides consistency across all stages of development, testing and production. Kubernetes runs on multiple machines in a similar way.
4. Runs on most operating systems independently of the device being on the Cloud or on premises.

5. Containers raise the level of abstraction and split applications into smaller pieces that are deployed and managed dynamically.

Kubernetes is a system that's responsible for managing these containers that run the applications and make sure that there is no downtime. If a container is down in a Edge node, Kubernetes is responsible for starting another one. It provides developers a framework to run the distributed systems and a way to make containers communicate with each other:

1. Containers are exposed using the DNS name or their own IP address. It also has a load balancing mechanism to distribute network traffic across the network. This is very important because Edge nodes need to communicate with each other and with a master node to perform computations.

2. Kubernetes allows developers to choose their intended storage system. On edge nodes it is possible to use a local system for faster processing. In other alternatives it is also possible to connect to a remote cloud storage system.

3. If there is a necessity to change the way that the data is processed, it is possible to change the state of the container to a new one.

4. Developers can inform Kubernetes how much memory and CPU it's required for each container and each node will be fitted with enough containers for an optimal performance.

5. Kubernetes performs a self check on containers to guarantee that they are performing as they should. If they are not, Kubernetes has the ability to delete and create new containers.

6. Kubernetes has the ability to store important information and secrets in a way that they aren't exposed to the outside.

Kubernetes is a very good platform to use at the Edge. It is possible to install containers on devices closer to the data source [35], even if they have low capacity in terms of memory and CPU. Kubernetes is a way to extend the benefits of the Cloud to the Edge with a very flexible architecture.

There are already some open source projects that make use of Kubernetes in a more improved way to manage workloads at the Edge:

1. Microsoft released an open source project called Akri with the intention of connecting small edge devices like cameras, sensors, microcontrollers and making them part of the Kubernetes cluster. Akri collects the output from the devices, extending the Kubernetes framework. This is one of the possible approaches for devices that are unable to run Kubernetes containers on their own [41]. Akri has multiple

controllers that can be configured in order to search for certain devices inside the Edge. The containers will look for the devices and if they have permission they will connect and start to receive the produced data which will then be analyzed and processed.

2. Krustlet is another open source product released by Microsoft that takes advantage of WebAssembly modules [37] which are binaries and not Operating Systems environments. These modules are very small and can be placed in edge devices or networks. Developers are able to compile code from most of the known programming languages. This code will then be executed by the Edge devices.

3. Cloud providers like Amazon, Google and Microsoft are also developing their own solutions for building edge applications. We will detail some of their products in the next topics.

2.4 Aggregation

Common computations that usually occur on the Cloud can also be performed at the Edge, depending on the Edge resources availability and capability. This leads to a need to build aggregation protocols [3] for the Edge devices, specifically for cases where these nodes can't sustain big computational challenges. Aggregation also helps in decreasing the amount of data that is transferred at once, which usually floods the network links between nodes. By splitting data into smaller units at the same time that we reduce the distance between source and destination (using edge instead of cloud), the overall efficiency will improve significantly.

In mathematics, aggregation functions are a type of computation that involves a range of values that are calculated independently and then results in a final output taking into account all of the different values [3]. Some examples of aggregation functions are the Average, Count, Maximum, Minimum, Range, Sum, etc. but each of these functions have different characteristics that need to be taken into consideration:

- Functions that have commutative and associative properties, like Maximum or Minimum are simpler to aggregate as they don't need to use other functions to reach a final result. On the other hand, the Average is an example of a decomposable aggregation function because to reach the final output we need more information from each unit like the number of values being calculated for each unit and apply another function. For the Average function, the output from each node should be a pair of (x, y) where x is the average for that set and y is the count of values.

- Another property that's important to take into account is if the aggregation function is sensitive to duplicate values. Mathematical operations like Sum are directly affected by the presence of duplicate values but on the other hand, Maximizing a set of values can ignore duplicates of a value that was already processed.

2.4.1 Aggregation on the Edge

As we mentioned before, computations at the Edge tend to provide a lower latency than in the Cloud. If we introduce aggregation into this concept it will improve the performance even more [4]. As we learned, it is better to deal and process data closer to the source (or on the source itself). For most IoT (Internet of Things) applications, what really matters is the final output, for example, analyze in real-time the risk of something collapsing or going wrong in a factory, and it is not necessary to store data. If we have sensors working in different machines inside a factory that are constantly producing logs that need to be analyzed in real-time, we can have a small device with little computational resources attached to the sensors that are summing the logs. Then, from time to time this data is transmitted to a central edge node (possibly inside the factory) that aggregates the results from each sensor device and creates a final output to be analyzed immediately by the factory workers. For compliance purposes, this data can then be stored in the Cloud as a backlog for future references or to do some machine learning. As this information doesn't affect the factory it can be calculated in the Cloud as the latency is irrelevant.

Usually each edge node holds a calculated value in a static way, meaning that it won't change for a certain amount of time or until it is transferred to the main node for analysis. If it is required an even faster analysis or processing of the data we need to start thinking about continuous aggregation protocols. This means that every time that data is produced it changes the value that was calculated in each node. The main aggregator [5] needs to be able to follow these changes to produce a real-time output.

In our work we will have to take into account the type of aggregation function that is going to be used for the data that is provided. There are some kinds of functions that require more processing capabilities than others but everything will be part of our framework. We will discuss later a way of removing unnecessary data that would only delay the processing of data.

On the next topic we will discuss some aggregation protocols that follow batch processing (data is sent from time to time in batches) and stream processing models (data is continuously being analyzed).

2.5 Distributed Computing Platforms

In this topic we will address some examples of Distributed Computing platforms that operate and manipulate data produced by users or IoT (Internet of things) devices. Cloud and edge computing are part of the Distributed Systems concept, in which multiple machines in different locations operate together to form an unified system.

Artificial intelligence and machine learning are shifting from batch processing to real time and event driven applications that require the data to be analyzed in real time at the Edge nodes near the data sources.

We will discuss and detail some products like Hadoop Map-Reduce, Apache Flink,

Apache Spark, etc. that are currently used to process large amounts of data with multiple machines and devices executing simultaneously. However, none of these existing solutions take into consideration the type of action being executed as we will conclude finally in the discussion.

2.5.1 Hadoop Map-Reduce

Map-reduce [21] was introduced because of the need to process large amounts of data, doing it in parallel by dividing the work into smaller and different independent tasks. This model was inspired by functional programming languages such as Lisp, Python, Erlang, Haskell, Clojure, etc. and targets specially data intensive computations. Data can have any format, being specified by the user or programmer, but the output is always a set of <Key, Value> pais. The algorithm is specified with the use of a Map and a Reduce function, as well with a sorting stage:

1. **Map:** The data provided as input is split into smaller sections. Depending on the size of the data and the memory available on each mapper server, the Hadoop [21] framework defines the number of mappers. After being split, each block is assigned to its mapper for further processing. These mappers are called worker nodes and are managed by the master node that checks each worker for a correct execution.

2. **Sorting:** After the Mapping function is over, the keys produced are then sorted (the values can be in any order).

3. **Reduce:** After all the worker nodes complete the mapping function, the Reduce can start. It processes and produces pairs of <Key, Value>, keeping the data type as in the beginning. All values that share the same key are assigned to a single reducer that will then process it.

We can consider that the execution of a Map-reduce function is called a job. Each job has a map, sorting and reduce phase. As an example (that will be discussed later), we can consider a job that counts the number of times each word appears in a set of documents. The map function will execute for each document the main computation (counting the occurence of each word). The input data is splitted automatically to run in parallel across the Hadoop cluster and it is received from the Hadoop Distributed File System (HDFS). HDFS enables streaming access to file system data in Apache Hadoop, its fault tolerant and stores data across multiple machines. The reduce stage will receive the result from the map tasks and form a set of parallel reduce tasks, aggregating the output of the map stage for all the documents, providing the final output. The final result is usually stored in HDFS. Before the reduce stage all of the output values from the map stage there is a shuffle process that creates a single <Key, Value> pair, where the value becomes a list of all elements sharing the same key.

As an example, we can consider that we have data splitted into 2 separate documents (doc1 and doc2). These two documents are considered the input data and are divided as chunks called splits, during the job configuration [2.6].

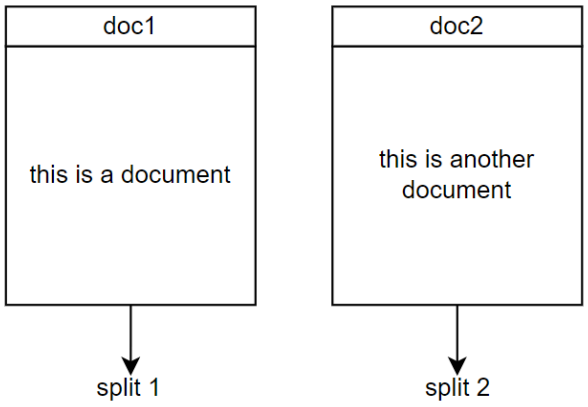

Figure 2.6: Document to split

These splits are included in the job information that is used by the job tracker. The tracker manages the tasks in each node on the cluster, creating a task for each split, transforming it into <Key, Value> pairs. In this case, each occurrence of a word represents a pair where key = word and value = 1 and it is possible to have repeated keys. Between the map and the reduce occurs the sorting and shuffling functions, where the pairs are ordered and split. The reduce stage will then receive the ordered values and produce an aggregated word count, accumulating the initial splits. The final output won't have any repeated keys [2.7].

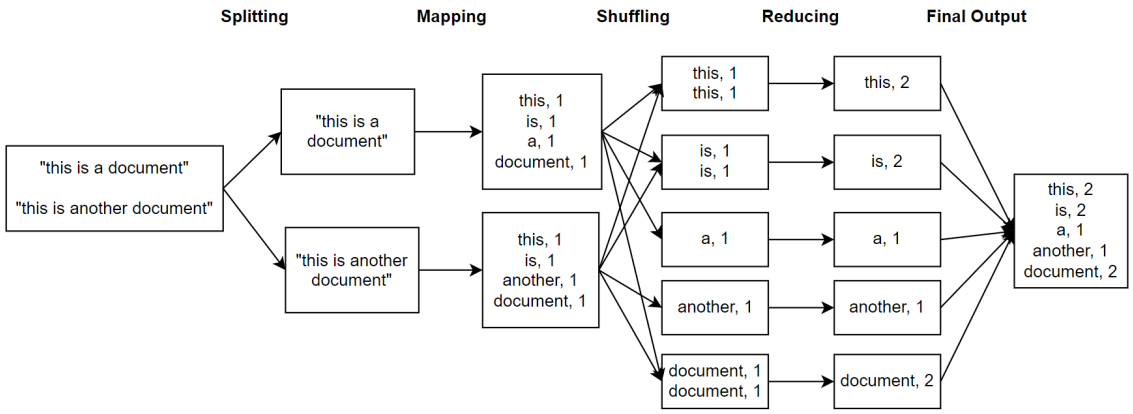

Figure 2.7: Map-Reduce phases

Hadoop Map-Reduce [21] may be very helpful in processing big data but it does it in batches (batch processing). This means that the data is analyzed all at once and it is finite. Only after the process of reducing is completed can the output be analyzed.

On the next topic we will provide an example of a stream processing data analysis product.

2.5.2 Apache Flink

Apache Flink is a data streaming processing framework [6] means that it is able to analyze data in real-time, as opposed to Hadoop MapReduce that receives data in batches. Data streams are a flow of information or data that is being transmitted from the source to the destination in a continuous way, ordered by the time it was produced. These streams can be heterogeneous, meaning that they are produced by different devices (for example sensors).

For our case study it is important to understand how we can split data streams into multiple smaller units. If the situation is that each device produces a data stream there is a possibility to have an edge node for each data producer to analyze individually the data. After this step, the output will be aggregated by a master node.

Data streams can be divided into two types:

1. **Bounded streams:** Data with a defined start and an end, meaning that it has a fixed size and the computations come to an end.

2. **Unbounded streams:** Data with a defined start but without any defined end. It is unknown the size of the data and when the computations will be over.

Although there are two different types of data streams, flink as a unified framework for their processing.

A very common use case for stream processing is the use of risk control systems. While the data is being processed in real-time (if possible on the Edge), a set of rules can be applied to check if there is any issue with the data and set an alarm for example.

With the implementation of consistent checkpoints, Flink has 24/7 availability. Checkpoints are responsible for handling faults, by consistently recording the state of operators during computations and generating snapshots for persistent storage. These states are data information that is generated in the middle of computations and it's very important to guarantee a recovery after a crash or failure. As stream computing can be considered as incremental (computation keeps increasing over time), it requires the system to query the data continuously. When some job fails within the Flink normal computation, it allows the system administrator to restore the job from a previous state checkpoint, making sure that no data is lost during the process.

2.5.3 Apache Spark

Similarly to Apache Flink, Spark [32] offers a scalable and fault tolerant data processing framework that can analyze and receive data in batches or streams. It provides the same framework for both data types and this is one of the reasons why it is used worldwide.

Opposite to Flink, Apache Spark simulates data streams by using micro batches [32] meaning that it can only provide near real time processing. Although the data is being analyzed continuously, it still suffers from the division into micro batches and aggregation. However, this increased latency can be acceptable for some applications. The throughput (amount of data as input) is directly proportional to the latency in this situation. This means that developers must find a way to optimize their Spark configuration as it is not improved on its own.

It is also possible to query data using SQL. Spark framework has SQL support to make it easier to retrieve data in a homogeneous way, independently of the data source, that may include Hive, Avro, Parquet, ORC, JSON or JDBC. There is also an API for querying data.

2.5.4 Functions as a Service (FaaS)

In this chapter we will discuss Functions as a Service computing and how it can be related to edge computing.

2.5.4.1 Concept

Functions as a Service (FaaS) is a computing service based on the Cloud environment [46] that allows to execute code when necessary (by triggers or timers for example) but without the need for the complex infrastructure that is usually required for deploying and managing micro services applications.

As we mentioned above, for deploying and hosting a web application on the Cloud, it is usually required to provision and manage infrastructure like servers and storage or even the operating system. With the FaaS concept, all of the infrastructure is provided and scaled when necessary by the Cloud provider in which you choose to deploy your FaaS service. This makes it easier to develop functions independently and in a more simple way [18]. This concept is very useful if an organization is migrating from the mainframe to the Cloud for example, allowing it to build independent functions with some benefits:

1. **Pay when you use:** FaaS computing works as an action. When the function is called, instances are deployed and the action starts. Users and organizations will only pay for the time that the action is occurring. This makes it very cost effective.

2. **Benefits of cloud infrastructure:** As this service is offered by cloud providers, it has high availability as it can be deployed on multiple availability zones in multiple regions across the world.

3. **Automatic scaling:** As it happens with cloud computing services, FaaS can be scaled automatically when necessary. When demand increases, instances will be scaled for a better performance.

4. **Focus only on code:** As we mentioned above, with FaaS there is no necessity to manage infrastructure and this allows users and organizations to focus only on the development of the code, reducing the time to produce new services.

As it enables functions to be isolated and scaled, FaaS is being used for high-volume computations in parallel. It can be used for data processing, aggregation and other forms of computing. FaaS can be used as a backend for Internet of Things (IoT) devices, approaching the concept of edge computing.

2.5.4.2 Amazon Lambda @Edge

Amazon Lambda is an example of a FaaS (Function as a Service) [46] from Amazon. Lambda functions run code pieces on a high-availability and scalable infrastructure, totally administered by Amazon. Similarly to other FaaS options, this service is only paid for the time it's being used (in 100ms increments). Lambda functions are also monitored automatically for performance logging, reporting metrics through Amazon CloudWatch.

The use of FaaS in close proximity to the users and devices brings a lot of benefits in terms of performance:

1. If there are some values that are used more frequently, they can be stored and accessed on the Edge by improving the cache hit ratio.

2. We have the possibility to create models with machine learning that will predict the requests that are made to the Edge at any specific time of the day.

3. By increasing the amount of requests that receive a response from cached data, the overall performance will increase significantly.

4. Amazon has infrastructure spreaded around the whole world and with the help of Amazon CloudFront content delivery network, users will be served by the closest location available.

We can take as an example of a smart light (connected to a smartphone) that can be turned on and off from anywhere in the world with an internet connection. When the user presses the button, it triggers an event to a function in the Cloud that switches the light. This is a great example of using FaaS as a single function that should be close to the user, it is simple to connect and it's very easy to scale to thousands of users and devices if necessary.

With Amazon Lambda @Edge it's possible to move some computation to the Edge, providing a better user experience and lower latency for client and data processing applications. Removing these kinds of events from centralized servers will allow for a more structured and stable architecture.

AWS Greengrass Amazon developed another product that can work in conjunction with Lambda functions called Greengrass [42]. It is an open source IoT (Internet of Things) edge service that helps in developing and using IoT applications and devices. AWS Greengrass can be deployed on devices to process the data locally that is being generated. With this service, developers can use edge devices to easily analyze and react to data with custom rules, making it also possible to communicate with other devices to make better decisions:

- Being an AWS service, it provides support for transferring data (processed or not) to the Amazon cloud storage services. The Greengrass software runs on multiple platforms including Windows and Linux distributions.
- It is possible to import modules that are available for public use and that are called components. Greengrass can run on Docker containers, native OS (operating systems) or custom runtimes.
- Components [10] are basically building blocks that help developers in creating more complex workloads. These components can help with machine learning, local data processing, data management, communication, etc.
- InfluxDB is an example of a component that provides a database for Greengrass edge devices and can analyze and process data in real time, monitoring operations at the Edge.
- LoRaWan protocol adapter is another component that helps in real time analysis by ingesting data from wireless IoT devices with a specific communication protocol, avoiding unnecessary communication with the Cloud.
- AWS Greengrass supports Lambda functions [9], importing them as components. This can be very useful if there are some functions deployed on Lambda that were previously used with another architecture to be transferred to an edge based architecture.
- Greengrass requires only 1GHz of computing power (Arm or x86 processors) and 96 MB of RAM and supports most programming languages like Python, Java or C.

Greengrass can be a good solution for running computations on the Edge nodes that are being managed by our framework. Functions can be used in combination with Greengrass running inside containers to perform data processing.

2.5.4.3 Apache OpenWhisk

Apache OpenWhisk [27] is another example of a Function as a Service, allowing developers to run application computing and logic from HTTP triggers or directly from any backend or web application. OpenWhisk is open source so it is not dependent on any

service like Amazon Lambda and it can be triggered using an API, allowing companies and organizations to have serverless computation capabilities.

It was also developed an attempt to improve OpenWhisk for edge usage - Lean OpenWhisk [24]. This software is intended for edge nodes with smaller and more limited nodes that can't sustain OpenWhisk. It removes some components like Kafka that are very heavy weighted by using a simple queue. As we mentioned before, in this thesis we want to take advantage of smaller edge devices with lower capability. If we divide computations into smaller units we don't need a single node to process the whole data. OpenWhisk being open sourced [8], allows us to study in more detail how we can achieve this final goal.

Similar to Greengrass, OpenWhisk is also available to run inside containers. This is very helpful because it can be triggered by a web application [11]. The master node can send information to the Edge devices that are hosting OpenWhisk, which will then perform the selected computations.

Microsoft Azure and Amazon are also starting to bring their FaaS platforms closer to the users and into the Edge with Azure Functions and AWS Greengrass, that connect to the Cloud ecosystem in a seamless way [25]. The only problem is that in order to use these services, the Edge nodes are controlled by the respective cloud provider. With the use of OpenWhisk, developers aren't affected by a possible vendor lock-in as they only depend on themselves.

2.5.5 Discussion: Lack of Computation Classifications

As we mentioned above, there are a vast amount of solutions and systems that can analyze and process large amounts of data while using multiple machines to perform computations simultaneously across the system. However, we also noticed that none of the current solutions take into account the different nature of computations when assigning different computational units to machines. This is also explained by the fact that these solutions assume a (mostly) homogeneous execution environment where all machines have similar computational resources. This means that, independently of the type of data or desired output, the way that the data processing is executed is the same.

Our solution, since focusing on the highly heterogeneous edge environment, aims to address this limitation by identifying different computational units of an application across a set of well defined classes with different properties and characteristics. If we take as an example a summing computation, it is very simple to decompose it equally into multiple units that will have a similar workload and execution, with a single output, this decomposition is sensitive to duplicate values. On the other hand, if the desired computation was to find the maximum or minimum value of the data, we will also have a single output but there is no sensitivity to duplicate values as they represent the same and would only increase the execution time and costs related with data transfers. This type of execution where the number of output values is lower than the initial input values,

should probably be executed near the data sources. Another option is to have multiple functions executing at the same time which should be taken into consideration when our framework selects the ideal node or nodes for execution.

In another scenario, there are certain types of computations in which the output of the execution is larger than the initial input. For example, in a machine learning algorithm, the model is created with a large amount of data and if this model is used to perform a prediction, the output can be larger than the input. In a weather prediction, the input can be represented as a full year of real data and the output can be two or three years of forecasts. This type of scenario where the output is larger than the input is likely to be more efficient if performed at the Cloud as to avoid increasing the data to be shipped from the Edge to the Cloud. This should also be represented in our framework as a class of computation with a different annotation for improving performance.

Different nodes have different resources and it is reflected in their throughput. Current products usually wait for all workers to finish their execution to perform final aggregation. This means that in a heterogeneous environment where the nodes have different resource power, if the amount of data being sent for processing is the same, some workers will finish their execution faster than others. We will discuss in the next chapter how we plan to fix this problem and provide a better experience.

2.6 Summary

In this chapter we addressed cloud and edge computing. Although the Edge seems to be a very good complement for current cloud applications and solutions, it still has a lot of flaws that can be fixed and improved. We decided to study ways to improve the use of edge computing on devices like sensors, cameras etc. where the computational resources are low and ways to minimize the amount of data that is unnecessary to be transferred from node to node as well as choosing the correct nodes for each type of processing function:

- We studied protocols and solutions that are designed for small devices using Function as a Service to compute the data, including some open source projects. This could be a good solution for splitting the computations into smaller blocks. If each edge device performs computations directly to the data that they are producing, it reduces the amount of data being transmitted from one side to the other. After the data is processed on each device, a final output can be transmitted from time to time to a master node for further processing or investigation.
- Another way could be to add another device with more power attached to the data producer and then perform computations on the secondary edge device. This could be a way of fixing the problem but it is costly to add new hardware to the infrastructure and the data would have to move from one node to another. If a lot of data is being produced and the Edge node doesn't have enough capabilities it can

be a solution. Because of this we studied how the data is transmitted when it is regularly being produced: in batches or in streams. Batches have a certain amount of time between transmissions and the data is grouped and sent together. In most cases there is no problem with this situation. On the other hand, stream processing allows for faster and smoother processing because the data is always being sent and analyzed.

- In cases where the network has some edge devices it can be considered as a way of fog computing. In this situation, our framework will help in optimizing the whole service by minimizing the latency taking into consideration all of the resources available on the network itself.

- We studied how computations can be split into different classes with specific characteristics and properties that require certain resources. These different classes will be represented in our framework as annotations and will take a big part in choosing the correct node for execution. In some specific cases it can be optimal to perform the computations in the Cloud instead of the Edge.

FaaS or serverless computing was one of the focuses on this document because resources can be used as functions instead of being constantly running. Currently there aren't many lightweight FaaS solutions for edge architectures and environments. To improve even further our solution, we can use FaaS as a simple operation where, for example, duplicate values are removed before being sent for processing. This allows to reduce the amount of data being transferred between nodes and therefore the bandwidth usage and overall latency and associated costs. We also studied containers as an alternative for virtual machines because these containers can be lightweight, scalable and portable. Kubernetes is a very good tool for managing these containers, including ways to improve communication between nodes. These containers can be hosted by the Edge devices and the master node, where the framework will be running and managing everything.

Since our intention is to divide the computations throughout the Edge nodes, we also studied aggregation as a way to reach a final output. Some aggregation protocols are already built for distributed systems and can be helpful to reach our objective.

3

Planning

In this chapter we will start to present our idea and concept of a solution to improve Edge computing. In Section 3.1 we will provide a solution that will be developed in the course of the year, detailing how we can take advantage of existing devices and network links to improve the data processing latency and overall efficiency and experience. From the developers of edge applications to the end-users, everyone will benefit from our solution.

Next, in Section 3.2 we will detail the evaluation plan for our framework in comparison to currently available similar solutions, taking into account the overall latency.

At last, we will present a schedule plan for our work ahead in the development of this thesis Section 3.3).

3.1 Solution: SMEdge

We are going to develop a framework called SMEdge - Self Managed Edge, to help developers build optimized edge solutions and applications. With the increase of IoT devices and the existence of multiple edge devices with different characteristics and computational power there is a need to optimize the way these devices are effectively used to execute applications. The framework provides all the necessary tools to develop edge applications, giving the programmers the possibility to add edge and IoT devices. It will take into consideration the latency and the predicted volume of the data transfers between edge nodes, the computational power and the type of operation that will be performed. We will achieve this by leveraging a conceptual framework that classifies the computations in different classes with well defined properties and characteristics such as comparing the size of the input with the output, among others.

SMEdge will make decisions at runtime in a way that can benefit several metrics:

- Latency associated with data transfers
- Operational cost associated with the number of machines or edge nodes running at each time.

- The maximum processing capability or load on each node, which also affects the latency directly. There shouldn't be any overloaded nodes.

Each node will have one or multiple containers installed depending on the availability of resources and will be managed by Kubernetes. The architecture of the nodes with containers will be based on the architecture originally proposed on [30] and presented in Figure 3.1. These nodes will be connected with a master node with the same type of architecture that will control and evaluate the whole system.

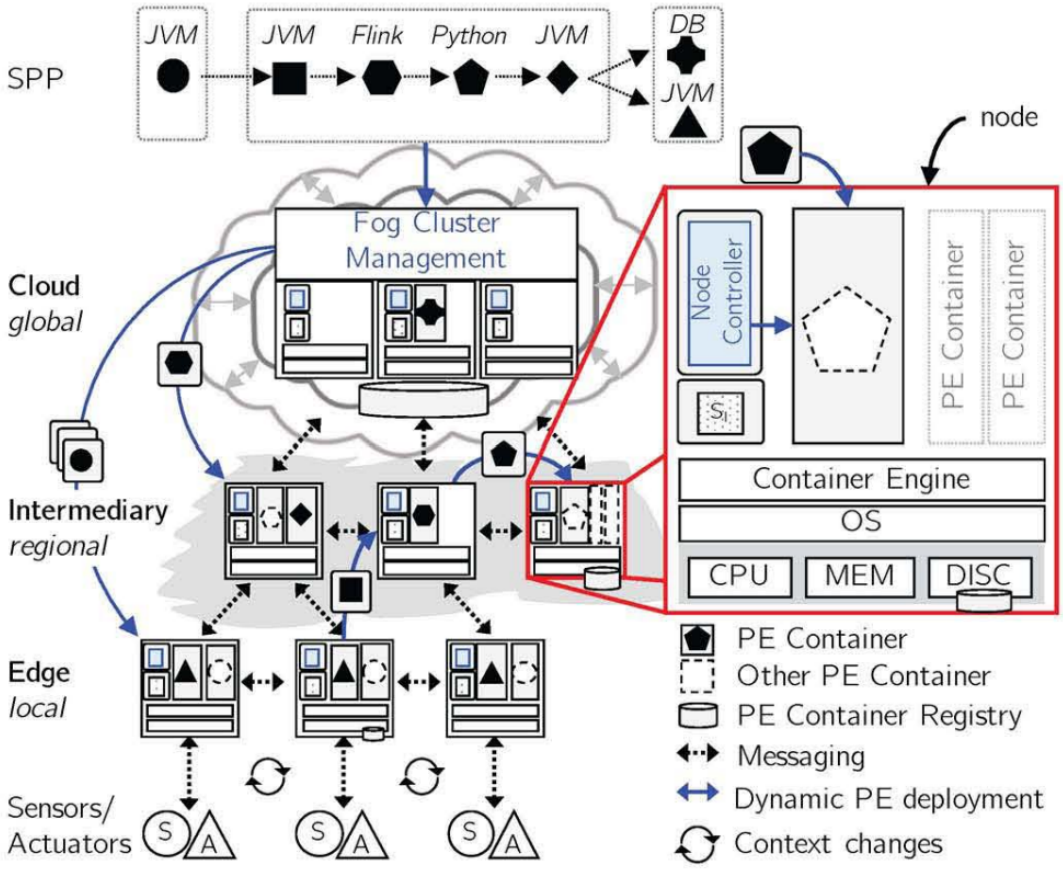

Figure 3.1: Fog computing architecture - Extracted from [30]

Developers will be able to develop small scripts to run the data processing and they will inform the master node, via framework, all the available nodes and their capacity. Each node will be pinged to check for availability and communication latency. The nodes will communicate with a REST API. This means that the developers are only responsible for selecting the type of computations they want to use among a selection of possible pre-defined classes that are represented in the framework as annotations. Depending on the type of computations and their characteristics, it can be more useful to perform specific functions in certain nodes or even on the cloud. The system will evaluate, for

example, if the type of computation produces a larger output than the initial input or vice-versa and then chose the most efficient path for execution.

Computation should be executed in the most appropriate and efficient node taking advantage of the whole infrastructure. Another step that we want to introduce is independent from the framework and will only be used in certain situations: Functions on edge devices. With a FaaS architecture directly on the data source, we want to eliminate some data that won't be necessary for future computations. This functionality is optional for the developers and can be activated in specific computations with characteristics that don't depend on duplicate values (like maximizing or minimizing values). By deleting duplicate values while they are produced, the amount of data being transferred can be reduced, making a direct impact on the costs related with network bandwidth and execution time.

As we also studied, in a heterogeneous environment, where nodes have different resources, the throughput will be different among the worker nodes. We want to fix this problem by splitting the data proportionally, in such a way, so that the master doesn't have to wait for some workers to finish their execution.

All of these decisions will be taken automatically by monitoring every component within the system. A master node needs to gather all the information necessary to make the correct decisions like the amount of processes running, the amount of data being received and the time of response on each worker. Each container should provide this information and then it can be stored and analyzed in a database or on an existent monitoring system. It is also possible to prepare certain nodes for specific types of executions, spreading different computations throughout different nodes.

3.2 Evaluation

To evaluate the framework, we will build a prototype and run tests to compare it to an approximately similar solution where there are multiple sensors or IoT devices constantly producing data. We will evaluate the following scenarios:

1. IoT devices producing data directly to the Cloud for further analysis and processing;
2. IoT devices producing data to an Edge architecture without any self managing algorithms;
3. IoT devices producing data to an Edge architecture with our framework available for optimization of resources.

As we mentioned before, we plan to evaluate the following metrics: latency, precision, server load and cost of operation. In our scenario, the cost will be directly related to the amount of resources and containers being used at each time. The precision will be evaluated by comparing the results with the exact and expected result and the latency will be evaluated by the overall time it takes to perform a certain computation task.

Everything will be compared among the three scenarios in order to showcase the benefits of the proposed solution.

3.3 Scheduling

In order to achieve our goals, we will organize our work according to the the following four main tasks and respective sub-tasks:

1. Framework development
    a) Prototype a new system
    b) Development of evaluation tools
    c) Evaluate and adjust
2. FaaS development
    a) Prototype a new system
    b) Function Integration
    c) Development of evaluation tools
    d) Evaluate and adjust
3. Decomposition and classification of computations
    a) Research and classification
    b) Creation of different annotations
    c) Evaluate and adjust
4. Evaluation
    a) Evaluate the Framework
    b) Compare to existing solutions
    c) Final adjustments
5. Document Elaboration
    a) Write the document
    b) Final adjustments

The time plan for these tasks are presented in the figure below 3.2. It is important to note that as we begin implementing the framework it will go through validations and preliminary evaluations as it evolves.

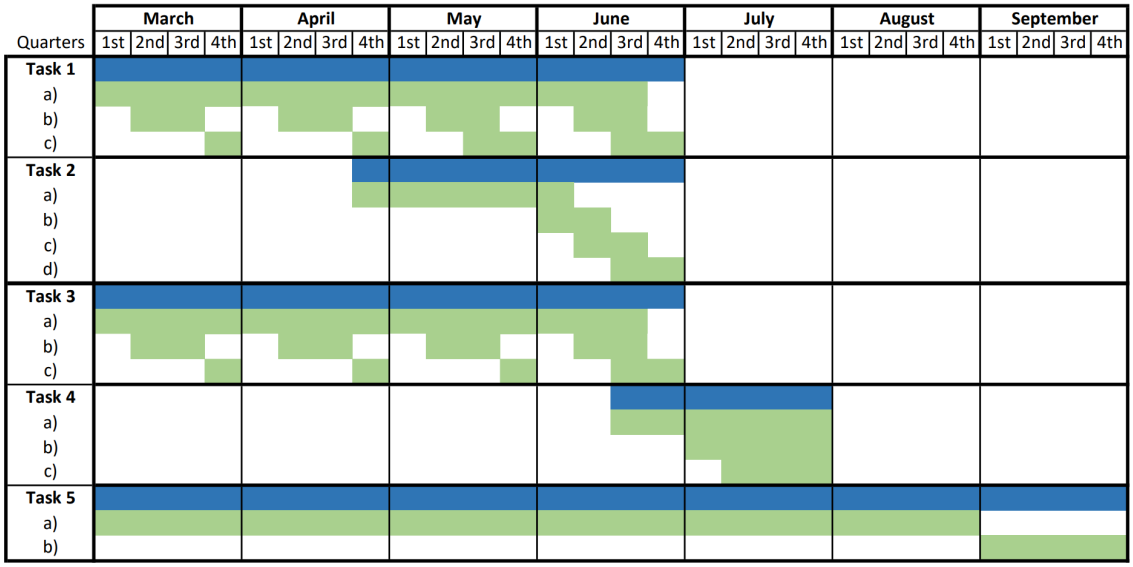

Figure 3.2: Work plan

Bibliography

[1] *A Comparison Between Edge Containers and Other Edge Computing Solutions*. https://www.medianova.com/en-blog/a-comparison-between-edge-containers-and-other-edge-computing-solutions/. Accessed: 2022-01-27.

[2] P. A. C. A. Rosa and J. Leitão. *Revisiting Broadcast Algorithms for Wireless Edge Networks*. 2019.

[3] *Aggregate function*. https://en.wikipedia.org/wiki/Aggregate_function. Accessed: 2022-02-01 (cit. on p. 16).

[4] *Aggregate Functions*. https://docs.oracle.com/database/121/SQLRF/. Accessed: 2022-02-01 (cit. on p. 17).

[5] *Aggregate Functions (Transact-SQL)*. https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql. Accessed: 2022-02-01 (cit. on p. 17).

[6] *Apache Flink Fundamentals: Basic Concepts*. https://www.alibabacloud.com/blog/apache-flink-fundamentals-basic-concepts_595727. Accessed: 2022-01-09 (cit. on p. 20).

[7] *Apache OpenWhisk*. https://www.techtarget.com/searchapparchitecture/definition/Apache-OpenWhisk. Accessed: 2022-02-03.

[8] *Apache OpenWhisk Provider Documentation*. https://www.serverless.com/framework/docs/providers/openwhisk. Accessed: 2022-02-02 (cit. on p. 24).

[9] *AWS IoT Greengrass Features*. https://aws.amazon.com/greengrass/features/. Accessed: 2022-01-13 (cit. on p. 23).

[10] *AWS IoT Greengrass ML Inference Solution Accelerator*. https://aws.amazon.com/iot/solutions/mli-accelerator/. Accessed: 2022-01-12 (cit. on p. 23).

[11] *Build cloud applications using an open source, serverless platform*. https://developer.ibm.com/components/apache-openwhisk/. Accessed: 2022-02-03 (cit. on p. 24).

[12] *Cloud computing*. https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing. Accessed: 2021-12-19.

[13] *Containers at the edge: Why the hold-up?* https://www.techradar.com/news/containers-at-the-edge-why-the-hold-up. Accessed: 2022-01-29.

[14] P. A. Costa and J. Leitão. *Practical Continuous Aggregation in Wireless Edge Environments*. 2018. URL: https://asc.di.fct.unl.pt/~jleitao/pdf/MiRAge.pdf.

[15] M. C. G. D. Mealha N. Preguiça and J. Leitão. *Data Replication on the Cloud/Edge*. 2019.

[16] *Edge Containers as a Service*. https://www.section.io/modules/edge-containers-as-a-service/. Accessed: 2022-01-27.

[17] *Edge containers tutorial*. https://cloud.google.com/vision/automl/docs/containers-gcs-tutorial. Accessed: 2022-01-28.

[18] *FaaS on Edge devices*. https://community.arm.com/arm-research/b/articles/posts/faas-runtimes-on-edge-devices. Accessed: 2022-01-19 (cit. on p. 21).

[19] *Final Version of NIST Cloud Computing Definition Published*. https://www.nist.gov/. Accessed: 2021-12-07 (cit. on p. 4).

[20] *Google Cloud Computing*. https://cloud.google.com/. Accessed: 2021-12-29 (cit. on pp. 7, 8).

[21] *Hadoop MapReduce Concepts*. https://www.javacodegeeks.com/2014/04/hadoop-mapreduce-concepts.html. Accessed: 2022-01-08 (cit. on pp. 18, 19).

[22] *IaaS vs. PaaS vs. SaaS*. https://www.ibm.com/cloud/learn/iaas-paas-saas. Accessed: 2021-12-10 (cit. on p. 6).

[23] *KubeEdge*. https://kubeedge.io/en/. Accessed: 2022-01-29 (cit. on p. 14).

[24] *Lean OpenWhisk: Open Source FaaS for Edge Computing*. https://medium.com/openwhisk/lean-openwhisk-open-source-faas-for-edge-computing-fb823c6bbb9b. Accessed: 2022-01-13 (cit. on p. 24).

[25] *Learning Apache OpenWhisk*. https://www.oreilly.com/library/view/learning-apache-openwhisk/9781492046158/ch01.html. Accessed: 2022-02-03 (cit. on p. 24).

[26] *Meaning of fog in English*. https://dictionary.cambridge.org/dictionary/english/fog. Accessed: 2021-12-20 (cit. on p. 12).

[27] *OpenWhisk - Github*. https://github.com/apache/openwhisk. Accessed: 2022-02-02 (cit. on p. 23).

[28] *Optimization problems*. https://tutorial.math.lamar.edu/classes/calci/optimization.aspx. Accessed: 2022-02-10.

[29] A. R. P. A. Costa and J. Leitão. *Enabling Wireless Ad Hoc Edge Systems with Yggdrasil*. 2020.

[30] D. R. Patrick Wiener Philipp Zehnder. *Towards Context-Aware and Dynamic Management of Stream Processing Pipelines for Fog Computing*. 2019 (cit. on pp. 2, 12, 28).

[31] *Public vs Private vs Hybrid: Cloud Differences Explained*. https://www.bmc.com/blogs/public-private-hybrid-cloud/. Accessed: 2022-01-02.

[32] *Spark Overview*. https://spark.apache.org/docs/latest/. Accessed: 2022-01-09 (cit. on pp. 20, 21).

[33] *Stateful Computations over Data Streams*. https://flink.apache.org/. Accessed: 2022-01-23.

[34] *The Different Types of Cloud Computing and How They Differ*. https://www.vxchnge.com/blog/different-types-of-cloud-computing. Accessed: 2021-12-08 (cit. on p. 5).

[35] *The Use Case for Kubernetes at the Edge*. https://thenewstack.io/the-use-case-for-kubernetes-at-the-edge/. Accessed: 2022-01-22 (cit. on p. 15).

[36] *Virtualization Gains Popularity as a Viable Solution for Enhancing Cloud Security*. https://cloudlytics.com/virtualization-gains-popularity-as-a-viable-solution-for-enhancing-cloud-security/. Accessed: 2021-12-08 (cit. on pp. 5, 8).

[37] *WebAssembly meets Kubernetes with Krustlet*. https://cloudblogs.microsoft.com/opensource/2020/04/07/announcing-krustlet-kubernetes-rust-kubelet-webassembly-wasm/. Accessed: 2022-01-21 (cit. on p. 16).

[38] *What are edge containers?* https://www.stackpath.com/edge-academy/edge-containers. Accessed: 2022-01-27.

[39] *What are public, private, and hybrid clouds?* https://azure.microsoft.com/en-us/overview/what-are-private-public-hybrid-clouds/. Accessed: 2021-12-15.

[40] *What are the differences between Cloud, Fog and Edge Computing?* https://www.mjvinnovation.com/blog/what-are-the-differences-between-cloud-fog-and-edge-computing/. Accessed: 2022-01-12 (cit. on p. 11).

[41] *What Does Kubernetes Have to Do with Edge Computing?* https://www.rtinsights.com/what-does-kubernetes-have-to-do-with-edge-computing/. Accessed: 2022-01-29 (cit. on p. 15).

[42] *What is AWS Greengrass?* https://www.amazonaws.cn/en/greengrass/. Accessed: 2022-01-09 (cit. on p. 23).

[43] *What is cloud computing?* https://aws.amazon.com/what-is-cloud-computing/. Accessed: 2022-01-02 (cit. on pp. 1, 8).

[44] *What is cloud computing?* https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/. Accessed: 2021-12-19.

[45] *What is Edge computing?* https://www.redhat.com/en/topics/edge-computing/what-is-edge-computing. Accessed: 2021-12-09 (cit. on pp. 1, 9, 10).

[46] *What is FaaS (Function-as-a-Service)?* https://www.ibm.com/cloud/learn/faas. Accessed: 2022-01-14 (cit. on pp. 21, 22).

[47] *What is fog computing?* https://internetofthingsagenda.techtarget.com/definition/fog-computing-fogging. Accessed: 2022-01-11 (cit. on p. 2).

[48] *What is fog computing?* https://www.stackpath.com/edge-academy/what-is-fog-computing. Accessed: 2022-01-11 (cit. on pp. 9, 12).

[49] *What is fog computing? Connecting the cloud to things.* https://www.networkworld.com/article/3243111/what-is-fog-computing-connecting-the-cloud-to-things.html. Accessed: 2022-01-13 (cit. on pp. 1, 11).

[50] *What is Kubernetes?* https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. Accessed: 2022-01-29 (cit. on p. 14).