Security in DevOps: understanding the most efficient way to integrate security in the agile software development process

Master of Science in Technology Thesis
University of Turku
Department of Future Technologies
Security of Networked Systems
October 2020
Mirko Schicchi

Supervisors:
Karo Vallittu (Awake.AI)
Bruno Crispo (University of Trento)

Examiners:
Petri Sainio (University of Turku)
Seppo Virtanen (University of Turku)

UNIVERSITY OF TURKU
Department of Future Technologies

MIRKO SCHICCHI: Security in DevOps: understanding the most efficient way to integrate security in the agile software development process

Master of Science in Technology Thesis, 85 p.
Security of Networked Systems
October 2020

Modern development methodologies follow a fast and dynamic pace, which gives great attention to customers' satisfaction in the delivery of new releases. On the other hand, the work pursued to secure a system, if not adapted to the new development trend, can risk to slow down the delivery of new software and the adaptability typical for an Agile environment.

Therefore, it is paramount to think about a new way to integrate security into the development framework, in order to secure the software in the best way without slowing down the pace of the developers. Moreover, the implementation of automatic and repeatable security controls inside the development pipeline can help to catch the presence of vulnerabilities as early as possible, thus reducing costs, comparing to solving the issues at later stages.

The thesis presents a series of recommendations on how to best deploy a so called *DevSecOps* approach and applies the theory to the use case of Awake.AI, a Finnish startup company focusing its business on the maritime industry. It is not always easy and feasible to practically apply all the suggestions presented in the literature to a real case scenario, but rather the recommendations need to be adapted and forged in a way that best suits the situation and the current target.

It is undeniable that the presence of a strong and efficient secure development framework can give substantial advantage to the success of a company. In fact, not only it makes sure that the delivery of good quality code to the customers is not slowed down, but it also dramatically reduces the risk of incurring in expensive security incidents. Lastly, it is valuable to also mention that, being able to show a clean and efficient approach to security, the framework improves the reputation and trustfulness of the company under the eyes of the customers.

Keywords: AWS, development, DevOps, DevSecOps, security, cloud computing, agile, vulnerabilities

Contents

List of Figures

1 Introduction

The developing processes have been rapidly changing in the last few years, with a shift towards speed of delivery and automation. DevOps[1] and cloud infrastructures have revolutionized the way new software is released to the customers. At the same time, the number of cybersecurity threats has increased as well, pushing organizations to put more resources to secure their assets. However, integrating security in a fast pacing model as the DevOps one can be very challenging and efficient strategies need to be put in place in order to get the best results.

Knowing that, the goal of this thesis is to understand the best approaches to embrace security in a DevOps and agile environment, describing all the necessary steps to build an efficient Secure Software Development Model. This framework would aim at discovering security flaws as early as possible in the development pipeline, so to reduce the costs that need to be faced when mitigations have to be implemented at later stages.

The results and suggestions present in the thesis have been taken from the relevant literature, and compared and criticized by describing the approach used in *Awake.AI*, a software company based in Finland, whose business is related to the maritime industry.

The thesis is organized as follows.

Chapter 2 talks about cloud computing and the solutions used to automate the creation of environments and services on top of it.

Chapter 3 describes the concepts of virtualization, Docker containers, Kubernetes and

[1]https://resources.collab.net/devops-101/what-is-devops

serverless paradigm, giving recommendations regarding the security best practices to apply on them.

Chapter 4 analyses the differences between traditional and modern development methodologies, taking into consideration the way security work is handled within them.

In Chapter 5 we introduce the concept of DevSecOps[2] as a new solution to embrace the security work inside the DevOps development process.

Chapter 6 describes how to collect useful information which can be crucial to recover from incidents and perform post-mortem forensic analysis.

Chapter 7 presents the case study of Awake.AI.

Finally Chapter 8 presents the conclusions, the results of our research and a proposal for future studies.

[2] `https://www.redhat.com/en/topics/devops/what-is-devsecops`

2 Cloud Environments

This chapter will give an introduction on cloud computing technology and the framework used to automate the deployment of a new infrastructure.

2.1 What is the Cloud

Following the National Institute of Standards and Technology (NIST[1]) definition in [1], *"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"*. In this way users can get access to the resources and services hosted by these servers from any kind of device, without having to worry about data loss since, most of the time, the information is replicated all around the globe.
This is very beneficial both for single users and for companies. In fact, this technology allows the former to get access to their own data from any device, and the latter to remove the burdens of running their own physical infrastructure, thus reducing the costs and the complexity of the system being built.

Cloud computing is very widespread nowadays and, according to a study published by Canalys [2], the total cloud computing market was worth more than 107 billion dollars during 2019, with a 37% increase with respect to the previous year. This huge spike in

[1]`https://www.nist.gov/`

investments on cloud computing has been carried on by the large number of tech companies which constantly compete with each other for customer market share. Within the industry these companies are called *Cloud Providers* and in the following section we will analyze the most relevant ones.

2.2 Cloud Providers

Cloud Providers differ from each others in terms of pricing, technologies implemented and business model followed. As stated in [3], there are three different business models which a cloud provider can approach, each of them having its own advantages and disadvantages. They are *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) and *Software-as-a-Service* (SaaS). In the following lines there are the three service models' definitions provided by NIST [1].

- **IaaS** enables customers to have complete control over the infrastructure that is physically deployed by the provider, leaving to them the freedom to choose the operating system to use, the network architecture and other infrastructural options.
- **PaaS** allows customers to deploy applications and services on top of the infrastructure managed by the provider. Therefore it's not responsibility of the customer to manage the operating system, the network topology and the storage.
- **SaaS** gives to customers the capability to use a specific application managed by the provider.

Cloud providers are also different from each others for what it concerns the number of parties that share the computing resources. In fact, we can go from a private cloud, which is used by a single company, to a public cloud, in which resources are shared and contended with many other actors. However, in both cases, the cloud providers guarantee a complete and efficient logical separation of resources owned by different parties.

There are three big cloud providers which are worth mentioning: *Amazon Web Services*[2] (AWS), *Microsoft Azure*[3] and *Google Cloud Platform*[4]. As reported in [2], AWS owns the largest share of the market, followed by Azure and Google, which have also seen a huge increment in the last years. The number of organizations that are signing multi-annual contracts with cloud providers is rapidly increasing, with a substantial amount of them which prefers to adopt two different cloud solutions in order to take the best features from each one. Given the fact that AWS is the biggest cloud provider and that it is the one used in Awake.AI, it is beneficial to describe its main features more in details in the next section.

2.3 Amazon Web Services (AWS)

Amazon Web Services (AWS) is a platform that offers reliable, efficient, flexible, scalable and cost-effective cloud computing solutions [4]. It was launched in 2006 by Amazon and since then it has seen huge increments in sales year by year, reaching an annual revenue of more than 35 billion dollars in 2019, as shown in Fig. 2.1.

The AWS infrastructure is spread all over the world and built around physically located *AWS Regions*, which may contain multiple *AWS Availability Zones* [6]. The latter can consist of one or more data-centers, all deployed in different geographical locations, but extremely connected to each others to ensure high availability, redundancy and fault tolerance. The communication between them is heavily encrypted and the ultra-low latency makes possible to achieve almost real-time replication. Each AWS Region is completely separated and presents different prices with respect to the others. A map of the current (year 2020) and new-coming AWS Regions and Availability Zones is shown in Fig. 2.2.

[2]`https://aws.amazon.com/`

[3]`https://azure.microsoft.com/en-us/`

[4]`https://cloud.google.com/`

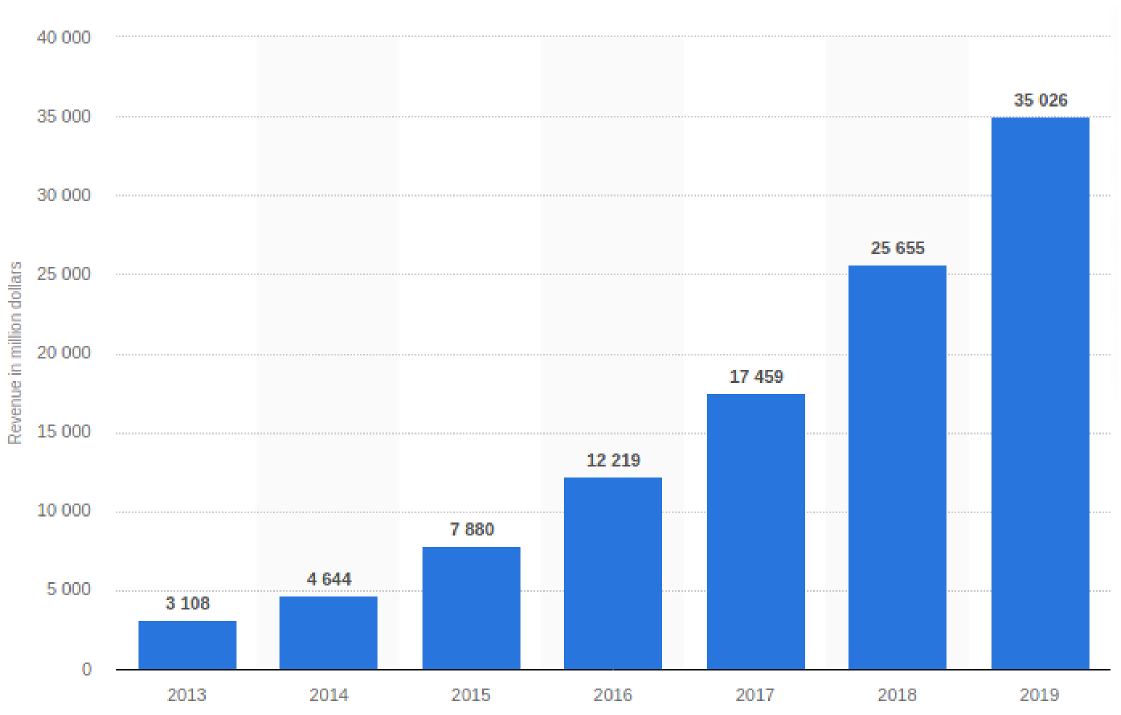

Figure 2.1: AWS annual revenue from 2013 to 2019 [5]

AWS offers an approach of shared responsibility for what concerns the security in the cloud (see Fig. 2.3). In fact, the cloud provider is responsible for the *"security of the cloud"*, that is security of the infrastructure and of the machines, while the customer is responsible for the *"security in the cloud"*, which means the security of the applications and services deployed. This shared security responsibility model offers freedom in the choices that customers can take, and at the same time, works as a warranty for the quality of the safety measures provided by the provider. In this regard, in addition to data protection, AWS provides means to easily meet compliance requirements, useful for the customer to be able to receive valuable certifications, which can increase the value of the product being built. Moreover, since the security of the bottom layers is AWS's task, it is easier for the cloud customer to scale quickly and, at the same time, save money.

In order to configure its cloud infrastructure a user can interact either with a management console or by typing command in a command line interface (i.e. CLI). The console is more user-friendly because it presents an user interface, nevertheless the command line

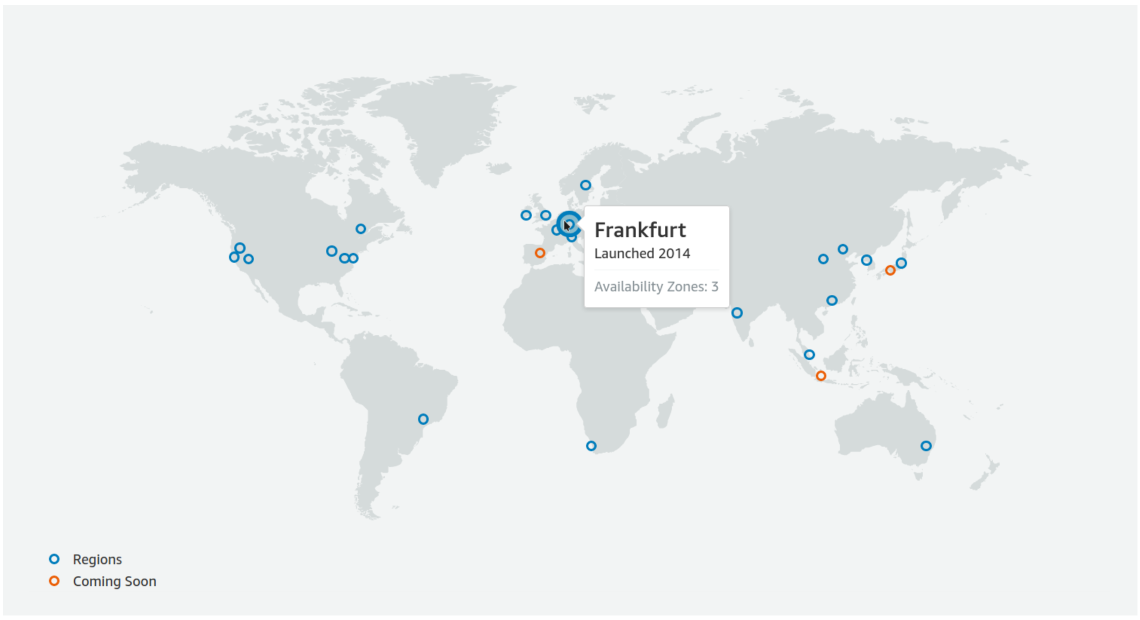

Figure 2.2: AWS Regions and Availability Zones [7]

can provide more speed and efficiency. The CLI and the Console allow to access the huge number of services which are present in the AWS portfolio. For the purpose of the thesis it is not valuable to describe in details all the services provided by AWS (also because there exist hundreds of them), but some of them are worth noticing since they are heavily used by Awake.AI, our case study. These are *EC2 machines*, *ECS*, *EKS clusters*, *AWS Lambda functions* and *RDS databases*, and will be described in the following paragraphs.

Amazon Elastic Compute Cloud (EC2) It is a service[5] that provides secure, reliable and scalable computing capabilities in the cloud. EC2 instances are sort of virtual machines (see Sec. 3.1) which can be completely customized in terms of resources and memory, with great importance given to scalability and booting speed. They support different operating systems and give the user the possibility to configure all the layers above the physical one. The instances can be requested on-demand by paying only for the time of actual utilization, or they can be reserved in order to have them available independently of the utilization level. EC2 machines are supported by the *Amazon EC2 Auto Scaling* ser-

[5]`https://aws.amazon.com/ec2/`

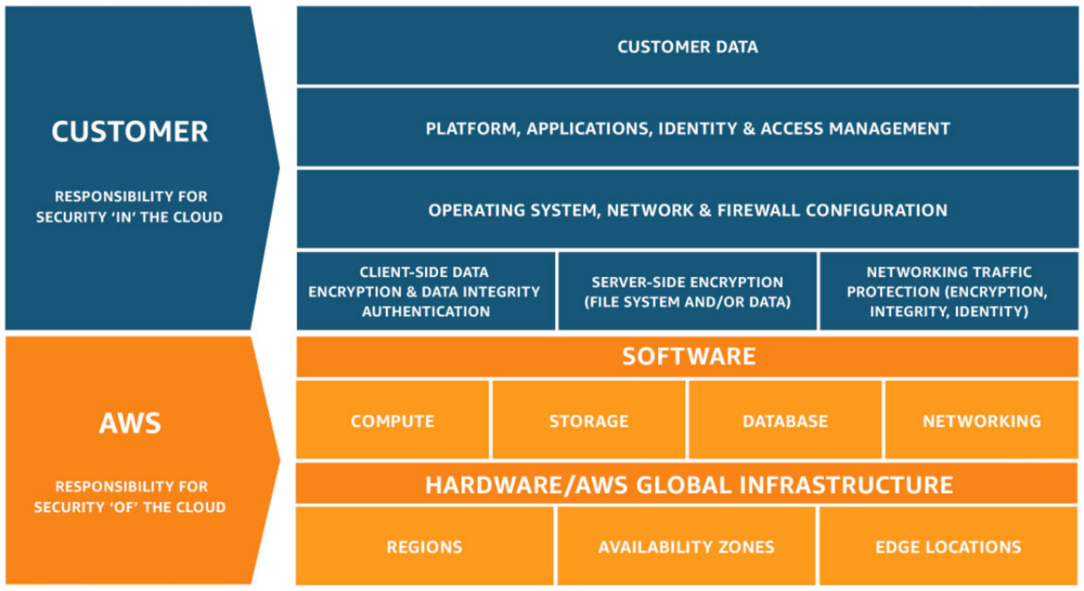
CUSTOMER
RESPONSIBILITY FOR SECURITY 'IN' THE CLOUD
CUSTOMER DATA
PLATFORM, APPLICATIONS, IDENTITY & ACCESS MANAGEMENT
OPERATING SYSTEM, NETWORK & FIREWALL CONFIGURATION
CLIENT-SIDE DATA ENCRYPTION & DATA INTEGRITY AUTHENTICATION
SERVER-SIDE ENCRYPTION (FILE SYSTEM AND/OR DATA)
NETWORKING TRAFFIC PROTECTION (ENCRYPTION, INTEGRITY, IDENTITY)
AWS
RESPONSIBILITY FOR SECURITY 'OF' THE CLOUD
SOFTWARE
COMPUTE
STORAGE
DATABASE
NETWORKING
HARDWARE/AWS GLOBAL INFRASTRUCTURE
REGIONS
AVAILABILITY ZONES
EDGE LOCATIONS

Figure 2.3: AWS shared security responsibility principle [8]

vice[6], which permits to automatically scale the deployment of EC2 instances depending on the traffic encountered.

Amazon Elastic Container service Mostly known for its acronym, *ECS*[7] is a container orchestration service which offers support for *Docker* containers (see Sec. 3.3) and provides all the capabilities for scaling and deploying new applications. It is similar to *Kubernetes* (see Sec. 3.4), with the only difference of being completely developed by AWS.

Amazon Elastic Container Service for Kubernetes Also called *EKS*[8], it offers a platform for easily deploying Kubernetes clusters, distributing them in multiple availability zones in order not to have a single point-of-failure.

[6]`https://aws.amazon.com/ec2/autoscaling/`

[7]`https://aws.amazon.com/ecs/`

[8]`https://aws.amazon.com/eks/`

AWS Lambda AWS Lambda[9] is the *Function-as-a-Service* (see Sec. 3.5.2) solution built by Amazon. It allows to write code that is executed and deployed without having to worry about managing a machine. Lambda works in a stateless manner, thus each execution is independent from the other executions.

Amazon Relational Database Service Also known with the acronym *RDS*[10], it contains interfaces for creating and managing MySQL and PostgreSQL databases, ensuring high speed and reliability in the operations.

2.4 Infrastructure as a Code

In the past, during their daily job, system administrators had to manually configure hundreds of machines and hardware components, taking care of the network infrastructure and the multiple updates needed. This job was often very tedious and repetitive, and the risk of misconfiguring something and thus leaving space to criminals to compromise the entire system was very high. Moreover, this operation was very costly since it was time consuming and required the effort of many engineers. In case of an incident, recovering the normal state of the infrastructure would have been very complex, with a very high probability of missing something.

With the introduction of cloud computing, the need to automate and make the infrastructure creation repeatable has lead to the development of a new paradigm called **Infrastructure as a Code** (IaaC). IaaC, as stated in [9], is a method that allows to deploy and manage service components and machines by executing some code written in specific configuration files. In other words, it provides a sort of *"programmable infrastructure"*, in which everything can be scaled, destroyed and deployed by executing scripts.

[9] `https://aws.amazon.com/lambda/`

[10] `https://aws.amazon.com/rds/`

HashiCorp Terraform[11] is one of the most famous scripting languages capable of managing a cloud infrastructure. Although it is not completely cloud agnostic, it is supported by all the major cloud providers and presents a good safety procedure that requires the user, before applying a modification, to carefully review the changes that are going to be performed. Thanks to the possibility to create modules (small scripts that perform atomic operations), it is possible to reuse already written code and share pieces of the infrastructure with other components, thus reducing the cost of development and complexity of the code base. Terraform maintains a state of the current infrastructure and applies only required changes, therefore it can be defined as *"idempotent"*. In fact, unlikely other scripting languages used to build cloud infrastructures (e.g. Ansible[12]), before applying a modification it fetches the current state of the system and calculates the differences between the current and the planned solutions. In this way, for example, executing a script to deploy two Linux virtual machines does not mean that new machines are deployed at every execution of the code, but rather Terraform will make sure that exactly two machines of that kind are running on the infrastructure. This feature is very useful because there is no need for the operator to have a look to the current state of the cloud before applying some changes, thus reducing the time needed and the risk of making mistakes.
In the next chapter we will introduce the concepts of virtualization, containers and serverless approaches, which are heavily used in the cloud.

[11] `https://www.terraform.io/`

[12] `https://www.ansible.com/`

3 Virtualization, Containers and Microservices

In this chapter we will describe the concepts of virtualization (Sec. 3.1) and containers technology (Sec. 3.2 and 3.3), describing the differences between them. Later we will introduce Kubernetes (Sec. 3.4) as a container orchestration service useful to manage clusters of containers. In the end of the chapter we will talk about serverless technology (Sec. 3.5) highlighting its benefits and drawbacks. All the topics presented will be accompanied by some related security considerations.

3.1 Virtualization

3.1.1 Introduction

Cloud computing is heavily dependent on **virtualization** technology. As stated in [10], virtualization consists of creating a virtual image of a server, operating system or network component and abstracting it from the hardware layer, so that multiple instances can be run simultaneously on the same machine. In this way it is possible to reduce the waste of computing resources and hardware energy consumption, thus cutting down the costs for maintaining and deploying services. By abstracting the hardware level, it is possible to run multiple instances of different services, each one on an isolated environment. In an ideal implementation, each virtual machine is sandboxed from the others and therefore it should not be able to access the data storage of another instance, despite being hosted on the same physical hardware.

3.1.2 Brief history

Virtualization technology has been studied starting already from the 1960s, when companies like IBM[1] started to research new methodologies for debugging their mainframe computers without having to deploy additional hardware. However, this technology had to wait until the early 2000s, when Linux and other Unix-like operating systems began to use it on a larger scale. The first step in the way towards virtualization widespread was the development of tools (e.g. hypervisors) which give multiple users the possibility to access a single computer at the same time.

3.1.3 How virtualization works

The hypervisor partitions the physical resources available on one computer in many virtual environments whose size can be adjusted accordingly, depending on the requirements of the virtualized service. Virtual machines are independent from the host system and so they can be moved and deployed into different computers very easily.
These guest machines, instead of interacting directly with the real hardware, they interact with an emulation provided by the hypervisor.

Virtualization can be done at different layers and for different purposes and, nowadays, there exist at least six different kind of this technology:

- **Network virtualization** consists in splitting the available network bandwidth into channels, in order to be able to independently assign a channel to a specific service.
- **Storage virtualization** allows to abstract the complexity of a distributed storage system to make it appear as a single unit, centrally managed [11].

[1] `https://www.ibm.com/fi-en`

- **Server virtualization** aims at hiding the real resources of a server to its users in order to reduce complexity to the end customer.
- **Data virtualization** abstracts data details to mask location, format and management.
- **Desktop virtualization** allows to securely access a remote desktop.
- **Application virtualization** is used to abstract the application layer from the OS, adding a layer of isolation. The architecture is shown in Fig. 3.1: the hypervisor works as a mediator between the virtual machines and the underlying host operating system, exchanging messages through the use of Application Programming Interfaces (APIs[2]).

3.2 Shifting to container technology

A **container** can be seen as a process running on a machine, packaging the image of an application and having all the required libraries, binaries and instructions needed to deliver an atomic service [12]. Containers abstract the application layer and run on top of the operating system of the host machine. They are completely isolated from each others and from the host machine, thus ensuring a high level of security and reliability.

Another key factor is their startup speed which can be in the order of hundreds of milliseconds: this characteristic makes them very desirable in a fast pacing development environment. This is due to the fact that a container contains only the essentials packages and software needed to perform a specific action, eliminating all the services that usually run on an operating system only consuming computational power. In this way, the attack surface is also reduced to the minimum.

[2]`https://red.ht/3cJdhq1`

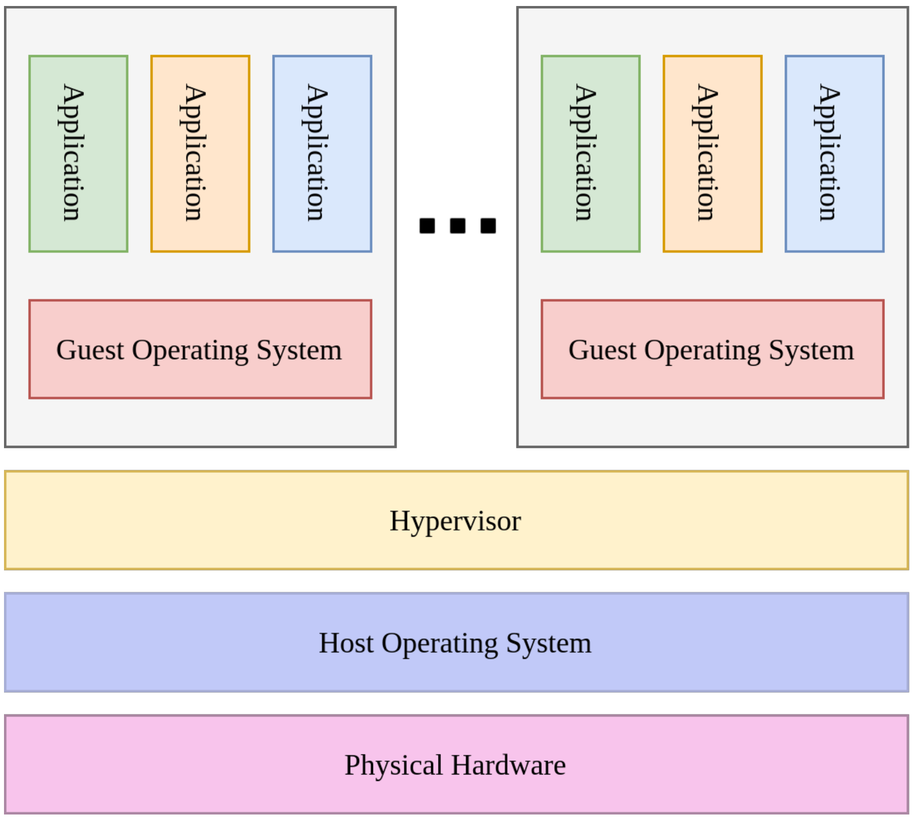

Figure 3.1: Architecture of a virtual machine environment

Shifting from using virtual machines to container technology can be very beneficial for the development process and the overall infrastructure of a company, but comes with some challenges that need to be addressed. As described in [13], there are three approaches that can be followed to perform the transition towards the adoption of containers technology. The first of them is called *lift and shift* and is the fastest one since it consists in moving the entire application that is running inside a VM into a container. There is only the need to fine-tune some configurations, but the overall work effort is minimal. However, this solution does not allow to fully benefit from the container technology, and so it is not recommended for serious projects.

The second approach consists in *refactoring* the application code before moving it into a container, in order to make it more lightweight and clean. Although this solution is quite easy to implement and gives some benefits, it still does not have all the good characteris-

tics of a real native container application.
The third, and more suggested solution is to re-architect the entire application, in order to adapt it to be containerized. This means splitting the big project into small micro-services which have to interact with each others and will be hosted on separate containers. In this way, everything is more easily understandable, with the addition of being more maintainable and scalable.

Although there are many containers technology solutions in the market, almost all of them are derived from a common *Linux LXC*[3] userspace interface, which offers some important kernel functionalities such us *namespaces*, essentials to implement containers isolation. Nowadays, the most famous containers solution is Docker. We will talk more in details about it in the next section.

3.3 Docker

3.3.1 Introduction

Docker is an open source Platform-as-a-Service product created by the so called (at that time) *dotCloud* company and launched in 2011 [14]. The project was then open sourced in March 2013, fact that led to the rewriting of its core in *Go language*[4].

Docker provides a fast and efficient environment in which code can run in a reliable and portable way, so that it is easy to move containers between machines running different operating systems without having to worry about fine tuning the dockerized application to meet specific criteria.
A Docker container is built starting from an image, which can be composed by multiple layers of applications built on top of each others. In this way it is possible to personalize an image, starting from an already existing base and adding the required functionalities.

[3]`https://linuxcontainers.org/lxc/introduction/`

[4]`https://golang.org/`

The instructions followed by the Docker engine to create an image are described in a special file called *Dockerfile*. A Dockerfile[5] is a text document containing a succession of commands that need to be executed by the Docker engine in order to create the application image and modify its file-system. Once the image is built, it's very straightforward to run it in any machine having Docker installed.

Docker images can be placed in *registries*, which act as repositories through which it is possible to share containerized software. These registries can be private (e.g. a Docker registry owned by a company), or public, like *Dockerhub*[6].

3.3.2 Docker vs. Virtual Machines

Docker containers and Virtual Machines differ greatly in the way the operating system supports them, the security measures applied, portability and guaranteed performance [15].

Firstly VMs have their guest operating system on top of the host OS, thus making them heavier than containers which, instead, use the functionalities of the host machine's kernel and share the resources with other Docker containers. On the other hand, this fact guarantees more security to VMs with respect to containers because if an attacker is able to tamper one container, he may manage to compromise both other containers running in the same host and the host machine itself.

Containers don't need the presence of a hypervisor, since they interact with the Host OS through the *Docker engine*, which accepts commands in the form of API requests (Fig. 3.2).

Talking about portability, containers have an advantage since they are very light-weight and are not dependent on the underlying operating system. The same applies for what it regards performance metrics, since containers can startup very quickly and consume only

[5]`https://docs.docker.com/engine/reference/builder/`

[6]`https://hub.docker.com/`

the minimum required amount of resources.

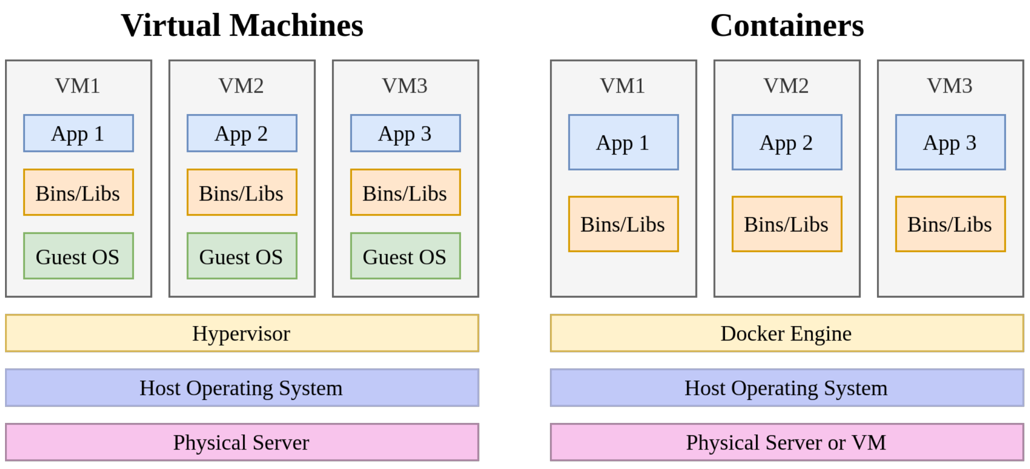

Figure 3.2: Docker vs. VM architecture

3.3.3 Security considerations

For all the benefits that bring in fastening the software development process, Docker containers, in the last years have met great interest among the community of developers. However, a lot of attention is needed to verify the security of this overwhelming technology.

Firstly, there have been some debates about who should be responsible for ensuring security in containers. The majority of the people think that developers should be responsible for securing their containers, while a minor part agrees in shifting the burden to operations and security teams. This is what Snyk, an information security company, report [16], adding that only 19% of the developers actually care to test the security of their Docker images at developing time. This means that a huge number of projects reaches the completion and release phase without having faced a single security scan, to then being discovered full of flaws in a later stage, when the fixes to be made may be extremely expensive. Moreover, even if it can sound controversial, the most downloaded and used Docker images from the DockerHub, are also the ones which contain the biggest number of vulnerabilities. In particular, the scan performed by Snyk in March 2019 found more than 500 vulnerable libraries in the NodeJS container image [16]. This information should alert most of the developers since official images are usually used as bases to build applications, and so by using vulnerable bases, the overall project inevitably inherits all the flaws.

Therefore, it is clear that security should be carefully taken into consideration when building Docker images, and in the following lines we will explain the best practices to follow. One of the most important steps is to start from a very essential and small base image which contains only the libraries that are required to run the application being deployed. In this way it is possible to reduce the attack surface to the lowest possible range and, at the same time, drastically lower the amount of resources consumed to run a container. As an example, if a *Linux Ubuntu*[7] image is needed, it is advisable to check if the *Alpine*[8] version contains the requested libraries, because it is preferred with respect to the others for the fact that is composed by only the essentials building blocks. Also removing any kind of bash or user prompt inside the container can help to deter many possible source of attacks. This last thing is the objective of *distroless*[9] containers, namely images reduced to have only the essentials packages to run the application. This approach is very smart since, most of the time, there is no need to run an entire operating system to host just a simple application. In this way, as said before, it is possible to reduce the number of vulnerabilities and improve the overall performance of the software.

Another option to take into consideration is building the image in a *multi-stage*[10] fashion. Docker, in fact, allows to build an image in more stages by copying some portions of an image into another one, so to retrieve only the very required blocks and avoid to stack unnecessary functionalities on top of each others.

[7] `https://ubuntu.com/`

[8] `https://alpinelinux.org/`

[9] `https://github.com/GoogleContainerTools/distroless`

[10] `https://docs.docker.com/develop/develop-images/multistage-build/`

Sometimes it's possible to remove some vulnerabilities present in official images, by just manually rebuilding a personal image in order to avoid to install unnecessary and security risky libraries.

The purpose of containers is to run in a very lightweight and efficient way a single application. Therefore, it is paramount not to containerize multiple applications together because a hypothetical flaw affecting one of them can have disastrous repercussions on all the others.

As a last advice, it is highly suggested to scan the images at each stage of the development pipeline, so to try to catch vulnerabilities as early as possible, and avoid the incoming of huge expenses in later stages.

3.4 Kubernetes

3.4.1 Introduction

Kubernetes[11] is a container orchestration system developed by Google and open sourced in 2014. Its main job is to manage a cluster of containers and eliminate any manual job needed to deploy and scale containerized applications.

Kubernetes provides a *Domain Name System (DNS)*[12] service to reach containers in the clusters, ensuring a load balancing strategy by scaling the resources and number of containers deployed automatically, depending on the specific traffic requirements.
It is also able to detect malfunctioning containers by performing periodic health checks, and restart them without any user intervention.
Moreover, it provides a way to store secrets (e.g. API keys, private keys, etc.) needed by containers for their normal business logic.

[11]`https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`

[12]`https://www.cloudflare.com/learning/dns/what-is-dns/`

3.4.2 Glossary

The following definitions and details about Kubernetes elements are taken from an article published by New Relic [17].
In order to explain the elements involved in Kubernetes, we start from the definition of *cluster* which is a group of either physical or virtual machines, each of them having the role of *master* or *node*.
Clusters can be virtually subdivided into *namespaces*[13], useful to ensure security in a multi-team project. Going more into the details, a namespace is a virtual cluster inside the physical cluster, whose job is to provide complete separation between the cluster resources, so that it is possible to limit the list of users which can access a resource.

In the following paragraphs we give the definition of *Master* and *Node*.

Master Each cluster needs to have at least one *master*, which is the entry point for the developer to interact with the internal nodes and to manage and schedule the deployment of containers. The master stores the configuration about the nodes in a key/value format and this information is used by the slaves to set their configuration.
In order to issue commands to the other nodes the master uses API calls, supported by many daemons (i.e. programs that run on background) that, by running, make sure that each container follows the prescribed instructions. Extremely important, in order to provide high availability and distribute the workload, is the *kube-scheduler*.

Node In Kubernetes glossary, a *node* is a physical or virtual machine which contains groups of containers called *pods*. Containers inside a pod have the same network address (they just use different ports) and can share data inside storage disks called *volumes*. Each node has a special service named *kubelet* whose job is to manage the status of the node, collecting health information useful for the master to take scheduling decisions.

[13]https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

Everything in Kubernetes is scripted by writing files using the *yaml*[14] language, which is similar to the *JSON*[15] format, but more human-readable. In this way, it is possible define all the details concerning a Pod, the Docker images to use to create the containers, the information about the volumes to mount and many other things, and deploy them by launching a simple command. This scripting approach allows to restart a cluster in a very short time, and make modifications to the infrastructure in a very easy and understandable way.

3.4.3 Security considerations

Kubernetes does not offer any security measures on the applications that are running in its pods, because it doesn't inspect containers in search of security flaws. Moreover, it does not have any built-in security measures to harden the network traffic between the containers running in the pods nor any way to filter the traffic that reach the nodes.
The only security feature offered by Kubernetes is the possibility to enforce a role-based access control to set the permissions that a specific container or user has over the other resources running in the same environment.

However, this access control framework is not typically enforced by default and needs accurate evaluation and compliance checks to meet the *Principle of Least Privilege* (see Sec. 5.7). Moreover, in order to achieve isolation in systems handled by a fair amount of users, it is useful to split the cluster into multiple *namespaces* so that components and nodes which are not suppose to interact with each others are kept safely separated.

In order to reduce the attack surface, it can be beneficial to make the file-system of the Pods read-only whenever possible. In this way, a potential attacker does not have the chance to inject malicious scripts or tamper some files, resulting in the disruption of the entire cluster.

[14]`https://yaml.org/`

[15]`https://www.json.org/json-en.html`

3.5 Serverless approach

3.5.1 Introduction

Serverless Computing is growing as an emerging trend in development of cloud applications. As stated in [18], serverless is a cloud computing approach which provides a model based on stateless functions. It takes the concept of Platform-as-a-Service (PaaS) to a point where developers just have to write arbitrary code, which can be as small as simple functions: for this reason the term *Function-as-a-Service* (see Sec. 3.5.2) is often used.

In order to properly give a place to the serverless model, it's useful to refresh the concepts of Infrastructure-as-a-Service (IaaS), Platoform-as-a-Service (PaaS) and Software-as-a-Service (SaaS), which have already been explained in Sec. 2.2.
In the IaaS model the developer can manage both the cloud infrastructure, organizing the machines needed and the network configurations, and the application code. Completely different are PaaS and SaaS on which the developer has no control on the infrastructure, but has only the duty to deploy the application code. Any aspect related to the infrastructure security or hardware management is left to the cloud provider.
The serverless approach can be put in the middle between the PaaS and the SaaS. This is because it uses a shared infrastructure as it is done in the SaaS paradigm, but the application code can be completely customized as it happens on the PaaS model. Serverless services can be shut down completely when no traffic is encountered, thus enormously reducing the costs. This claim is supported by some calculations that researchers in [19] have done, demonstrating that the serverless approach can reduce the operational costs to over 99.8% with respect to the use of standard virtual machines using the major cloud providers. Therefore, it is easy to understand that serverless paradigm is very useful to handle services that need to be up only for a short burst of time, eliminating the need to keep a server continuously up and running in a idle state.

Moreover, the fact that serverless solutions scale accordingly to the traffic, gives a lot of benefits in terms of cost savings when dealing with a service that has periodic congestion spikes. In fact, in a traditional manner, it would be needed to deploy a machine capable of managing the highest traffic load encountered, even if those spikes are only rarely encountered. Instead, with the serverless approach the resources are automatically scaled and reduced in real time, following the traffic trends.

3.5.2 Function as a Service (FaaS)

In 2014 Amazon introduced an innovative solution, called *AWS Lambda*[16], in which the serverless approach was taken to the extreme. There are no servers perpetually running in the background, but an event mechanism which triggers the allocations of a specific amount of resources needed to run a piece of code. The service can have a time span as short as milliseconds and then die until it's recreated when triggered another time. This allows developers to focus only on building quality code, without having to worry about deploying and maintaining a web server which, by using the traditional approach, may struggle when the number of users using the application grows. Moreover, the customer is charged by the cloud provider only when the code is actually executed and this can present huge cost reductions.

Serverless functions are often triggered as a response to certain events being fired, and can be used as a tool to compose and aggregate API calls in order to reduce the number of requests sent to the back-end services. In order to achieve API aggregation, an *API Gateway*[17] is needed. This component provides a public interface to the client and automatically triggers the API calls needed to fulfill the request.

FaaS is not dependent on any specific library or framework. Any kind of supported programming language can be used and it is cloud provider's responsibility to deploy the underlying environment in which the application can be executed.

[16]`https://aws.amazon.com/lambda/`

[17]`https://aws.amazon.com/api-gateway/`

However, FaaS is not always the best option. In fact, although it is useful to manage micro services, bigger services still benefit more from a Platform-as-a-Service approach. This is because with PaaS the development team can control in a more efficient way a project of a decent size, and have the possibility to use the *read-only cache* functionality, which can be very beneficial for applications that need to run very often. In fact, FaaS does not provide cache functionalities since, by definition, each execution of the code must be independent from the others.
A way to enforce the benefits of FaaS in a big project is to subdivide it in small chunks which can run autonomously: doing so, each small service can be served by a serverless function.

3.5.3 Drawbacks

Till this point, we have discussed all the benefits of running serverless applications. However, this technology presents some drawbacks that may be relevant in certain situations. This section will try to give an overview of them.

First of all, a serverless approach forces to rely completely on the vendor for what concerns downtime, upgrades and general management. From one side, this is a good thing since developers can concentrate on writing their code but, on the other side, the lack of control can cause troubles when the details of the implementation need to be fine-tuned.

Secondly, serverless forces the sharing of physical resources with other parties, since cloud providers sell the computational power of a single hardware to multiple companies in order to generate as much revenues as possible. Although this is a very remote possibility, it can happen that the shared machine is affected by some flaws which would create some holes in the applications running on top of it, allowing a customer to interact with the services owned by another customer.

Serverless is also not suited, by default, for storing local state, since each code execution should not be correlated at all with the previous ones. This concept follows strictly the 6th rule of the *Twelve-factor app* [20], that demands processes to be stateless and to use external storage systems for persisted data. Nevertheless, sometimes the urge to store locally the data at execution time can be mandatory, thus making serverless approach not a suitable solution.

As a last point, serverless is not the recommended option if the task implemented needs a lot of time to be executed. In this case it is way more convenient and cheap to use a traditional server approach which can take all the time needed to perform a specific action.

3.5.4 Security considerations

One big security concern about adopting serverless architecture is the fact that functions owned by different parties may share the computing resources with other companies. Although the cloud provider may claim that its service is completely secure and offers extreme isolation between the services hosted on the same bare-metal machine, it is still possible that a hidden vulnerability may be exploited to jump on the memory sectors allocated for a third party.

Traditionally, the main entry-points for an injection attack were the input fields, file upload components and all the others parts where the user was able to insert arbitrary data. Shifting to a serverless approach the number of options available for data insertion grows since information can come from emails, code modifications, cloud storage events, as described in the *OWASP Top 10 for Serverless applications* [21]. The same applies for XSS[18] attacks which can be originated from a big variety of sources.

Moreover, it can be very complex, if not even impossible, to scan a serverless function using automatic security scanners, because many applications don't use HTTP protocol to consume the input, but rather rely on different communication protocols, and there are

[18]`https://owasp.org/www-community/attacks/xss/`

not so many scanners capable of performing this job.

Given the fact that the application owner does not have to deploy and manage the infrastructure, the traditional methods for inspecting and blocking malicious traffic using *Intrusion Detection Systems*[19] or *Firewalls*[20] cannot be applied. As said before, it's Cloud Provider's responsibility to take care of the security of the layers upon which the application is running.

For what concerns *Denial of Service*[21] attacks, we can say that although it is possible for an attacker to trigger the execution of a serverless function by sending a huge number of requests, the system should be able to scale accordingly to handle the traffic. The only problem is that, even if the service would keep working, the customer's bill would be overwhelmed by the expenses.

Authorization and privileges must be granted granularly to each serverless application, by enforcing the *Principle of Least Privilege* (see Sec. 5.7). Taking as an example the AWS infrastructure and its serverless solution called AWS Lambda, authorizations to access certain resources and perform specific actions are granted through the help of the *AWS Identity and Access Management (IAM)*[22]. It allows to set policies and permissions to set access to the AWS resources. Despite being a very powerful tool, the configuration of policies and permissions depend on the customer, and so they need to be properly performed and checked.

[19]https://www.geeksforgeeks.org/intrusion-detection-system-ids/

[20]https://www.forcepoint.com/cyber-edu/firewall

[21]https://www.us-cert.gov/ncas/tips/ST04-015

[22]https://aws.amazon.com/iam/

4 Traditional vs Modern Development

When developing a software it is very important to define a strategy and a process to follow to build the artifact in the best and efficient way. An efficient *Software Development Life Cycle* is paramount to gather all the requirements from the customers and build an actual product out of them. It helps development teams to be focused and motivated on the tasks to perform and gives strict process guidelines useful to develop quality software. Nowadays there are two main Software Development Life Cycles, the traditional and most rigorous one and the Agile methodology. Sec. 4.1 will describe the traditional methods used for software development, while Sec. 4.2 will concentrate on describing Agile methodologies.

4.1 Traditional models

There are many different traditional development models, but all of them follow the following six common stages [22]:

1. **Requirement gathering and analysis**. During this phase product owners and software engineers talk with customers and try to gather as much information as possible about the real needs that must be fulfilled with the product that they are going to build. It is important to take input from experts in the field of study and from financial professionals who can evaluate the economical feasibility of the product.

2. **Definition of requirements**. Once all the requirements have been gathered and

analyzed, they should be precisely documented and approved by the client.

3. **Product architecture design**. The next step is to design the architecture of the system and draw as many diagrams as possible about any aspect of it (e.g. data flow, important components). *Unified Modeling Language (UML)*[1] diagrams with user cases are also very important as they model in a very understandable way the various actions that need to be performed. In this way it is possible to practice a risk evaluation and estimate time and budget needed.

4. **Product implementation or development**. This is the most technical and practical stage, in which the actual code is generated by following the guidelines produced during the previous stages. Developers need to organize and prioritize their work in order to release the software in time.

5. **Product testing**. During this phase the code is reviewed and refactored, and intensive tests are performed in order to find bugs and vulnerabilities.

6. **Market operation and maintenance**. The product can be sold to customers and a maintenance team must be activated in order to solve possible problems.

In the following sections we will describe in more details two models, highlighting advantages and drawbacks for each of them. In particular, we will analyze the *Waterfall* model (Subsec. 4.1.1) and the *Incremental* model (Subsec. 4.1.2).

4.1.1 Waterfall model

The most famous traditional development process is the **Waterfall** model [23]. It was designed in 1970 by W. Royce and it strictly follows a sequential process in which each stage must be entirely completed before jumping to the next one. The five stages that compose this model are shown in Fig. 4.1 and there is no need to explain them further

[1]`https://www.uml.org/what-is-uml.htm`

since they are basically the same described in Sec. 4.1. This model allows new developers to easily join the team even in the middle of the development phase, thanks to the rich documentation created in the previous phases. The problems with this approach are that, being so rigorous, it's very difficult to deal with changing plans and unexpected situations that may arise. Therefore, this model is appropriate when dealing with small and well documented projects, with low risk of having ambiguous requirements.

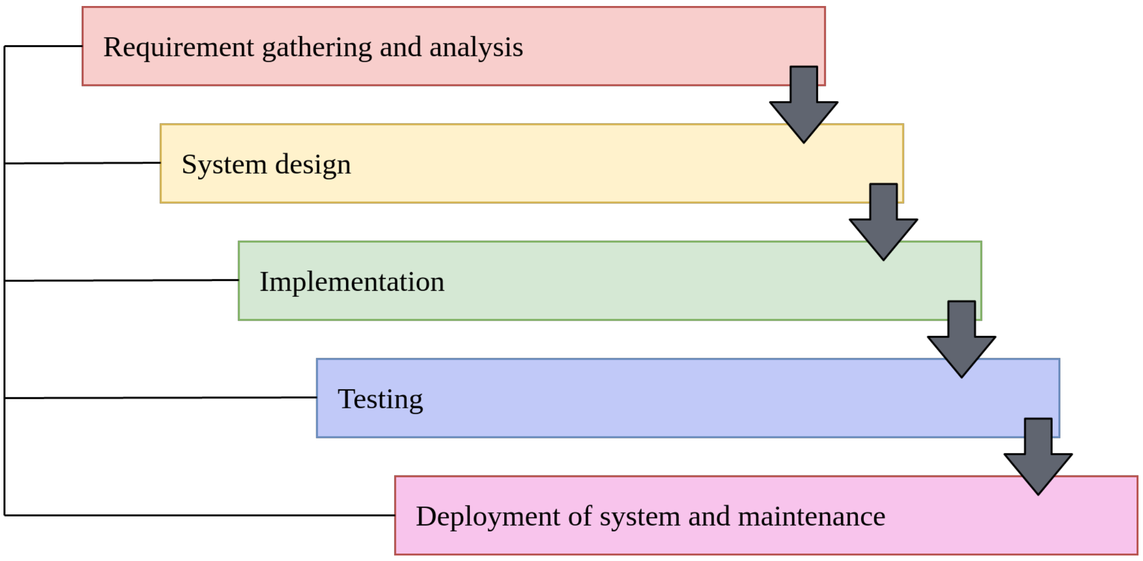
Requirement gathering and analysis
System design
Implementation
Testing
Deployment of system and maintenance

Figure 4.1: Waterfall model diagram

4.1.2 Incremental model

The **Incremental** model splits the requirements into multiple groups and applies to each group the same approach presented in the Waterfall model [24]. Therefore, the product is built incrementally, by adding new features and functionalities at each iteration. The advantages are that users' feedback can be gathered more frequently and that, being more flexible, the costs resulting from a misunderstood requirement are lower with respect to the Waterfall model. However, the flexibility is paid with higher costs. On the other side, it is paramount to have a clear initial idea of all the requirements so to be able to spread them accordingly around the various iterations.

4.1.3 Security in traditional models

A big problem that affects traditional software development methodologies is the lack of a strict and efficient methods to apply security during the development lifecycle. In fact, traditionally, the security actions were taken only during the testing phase and were often underestimated and not properly applied. This led either to not discovering possible security flaws, or, in case some vulnerability had been detected, to the impossibility of properly fixing them due to the huge costs that would need to be encountered. Moreover, developers may have had to analyze some code that they had written a long time before, thus wasting time just trying to understand it.
During the years, it has been tried to incorporate security in the traditional software development lifecycle and, while some of them have resulted in good performance improvements, others have just scraped the surface of the real problem. It is paramount that every phase of the development process is accompanied by a security mechanism which can allow, firstly to develop more secure code and secondly to catch security vulnerabilities as soon as possible. One important factor is developers' awareness with respect to cybersecurity risks. It is very beneficial to organize training sessions to give programmers a base on how to write secure code and how to reduce the risk of creating vulnerable software. These lessons should be given to each new coming employee and repeated at least once in a year in order to refresh the main concepts and update the overall knowledge.

The security work should start already at the requirements gathering stage, during which security engineers can put on the table a general idea of the main security practices that need to be fulfilled.

However, the strict structure of traditional models can fail to adapt promptly when new vulnerabilities are discovered, resulting in two potential bad scenarios. The first possibility is the development process being heavily slowed down by the security work, and the second one is the vulnerability found not being properly handled in order not to block the work-flow. Both the scenarios are very negative and unavoidable, thus a more adaptive approach to development, called Agile and explained in Sec. 4.2, can potentially improve the integration of security work and software development.

4.2 Agile model

The **Agile** methodology was popularized in 2001 when a group of seventeen software developers published the *Manifesto for Agile Software Development*[2], in which they summarized the essence of the approaches that had been already presented in the 90s. They claimed the importance of frequent interaction between team members, the delivery of quality software over spending time in writing long documentation, the need for frequent customers' feedback and the adaptability to frequent and fast changes. Therefore, the most important aspect that the agile methodology wants to ease is the interaction between customers and development team, to quickly adapt and change depending on the requested features. The adaptability can also be applied in the case a security vulnerability is being discovered in the software, and so when changes are needed in order to make sure that the software is secure.

In the following sections, we will take a look to the most important Agile development frameworks: Scrum (Sec. 4.2.1) and Kanban (Sec. 4.2.2).

4.2.1 Scrum

Scrum was presented for the first time in 1995 by Jeff Sutherland and Ken Schwaber and polished in 2002 [25]. This framework is generally described with three attributes: lightweight, easy to understand and difficult to master. It can be effectively applied in many different situations and manages to handle correctly even big and complex projects. The main and necessary elements composing the Scrum framework are the *Scrum team*, the *events*, the *artifacts* and the *rules*. Scrum is an empirical process based on past experi-

[2]`http://agilemanifesto.org/`

ences and facts and is based on three main pillars:

- **Transparency**. All the participants need to have a clear idea of the tools and languages used and understand the process completely.
- **Inspection**. It is important to frequently review the work in progress in order to be able to find errors and unfulfilled requirements.
- **Adaptation**. If the inspection highlights a wrong behaviour, it is paramount to smoothly adapt to a change of plans in order to solve the issue as early as possible.

Inspection and adaptation are supported by four important events, which are the *Sprint planning*, the *Daily Scrum*, the *Sprint review* and the *Sprint retrospective*. We will talk more in details about these events in the next lines.

The Scrum team is composed by three different figures:

- **Product owner**. He is responsible for creating items for the backlog starting from the user's requirements, order and group them accordingly to prioritize the work for a specific sprint. The items must be transparent and easy to understand by the developers.
- **Developers**. They do the actual coding work by following the directives of the product owners. They can be subdivided into teams, which need to have a decent size for proper and advantageous interaction, and have to frequently share their progress and ask for help from other colleagues in order to maintain a clear idea of what is being built.
- **Scrum master**. He is the leader of the development team and the person entitled in heading the daily meeting and the various situations that may arise. He can also be seen as a mediator between the developers and the product owner, easing the interaction between them and translating the topics in a language understandable by both parties.

There are four different kind of events which compose the framework. In order to understand them we have to introduce the concept of *Sprint*. A Sprint is a period of time (spanning between 2 to 4 weeks) in which a specific set of tasks need to be completed. A Sprint should be carefully designed and changes should be avoided when it's already running. The design is incorporated in a phase called *Sprint planning*, which is a maximum 8 hours sessions in which the entire team decides the work that should be done in that particular Sprint. The input is the product backlog from which some items are taken, depending on the priority and on the work load needed. The developers can estimate the time needed for implementing a task and can start breaking big tasks into smaller ones which are understandable and feasible.

The second event that we need to take into consideration is the *Daily Scrum*. It is a daily meeting of around 15 minutes, during which each member of the team tells about the work he has done in the last 24 hours and can ask for help or suggestions in case there is something blocking his progression. The meeting is held by the Scrum Master who has to take notes and ensure that everything is done smoothly and without wasting too much time. The team takes into consideration the Sprint backlog and moves items in it depending on the state of each task.

At the end of the Sprint period there is a session called *Sprint review* during which the team analyses the results of the last weeks and evaluate the overall performance with respect to the results achieved. It is also very important to perform a demo session, to show to the others the value of the increment of the last Sprint. It is possible that some tasks have not been completed and so they will be moved to the next Sprint.

Between the end of one Sprint and the start of the other there should be a short meeting session called *Sprint retrospective*. During this time the team identifies good and bad things of the past Sprint and proposes improvements for the next one.

4.2.2 Kanban

Kanban [26] is a framework that didn't originally arise from the software development field, but rather was experimented for the first time in the 1940s in the Toyota car manufacturing process. Toyota's idea was based on what supermarkets usually do to manage their stocks: trying to align the items stored in the inventory with their consumption level. In other words, the idea was to store in the productions site only the right amount of resources needed to complete a small task. Once the items used in production were finished, a sheet of paper, called *kanban*, was sent to the team managing the inventory stock to request for new supplies.

This concept was easily adapted to the field of software development by matching the amount of work in progress (i.e. WIP) to the team's capacity. The developer team only needs a board to which to attach cards, and this can be either physical or virtual. The cards contain the tasks that need to be done and are moved on the board (which is divided, at least, into three sections), depending on the state of the work. In a basic configuration a task can be in the state *"To Do"*, *"In Progress"* or *"Done"*. Each team can also add as many stages as needed to the board: an example can be a *"Review"* stage, in which the task's completeness is evaluated before being moved to the "Done" state.

The cards on the board give to the team full visibility on what is being done and by whom, allowing to keep track of the work done and, by taking into consideration the previous work, forecast the time needed to conclude future tasks. Extremely important, in this regard, is to have some tools that help to graphically visualize the work progress and find possible problems in the workflow. For example, a diagram as the one shown in Fig. 4.2, illustrates very clearly the state of the tasks over time.

The product owner generates the cards and prioritizes them in a stack structure, so that developers can take the ones on the top and start working on them. The process is quite similar to the one followed by Scrum, with the difference that there are not fixed-time Sprints and everything evolves continuously. For this reason, we talk about *continuous* *delivery*, which consists in frequently releasing new features to the customers as soon as they are ready.

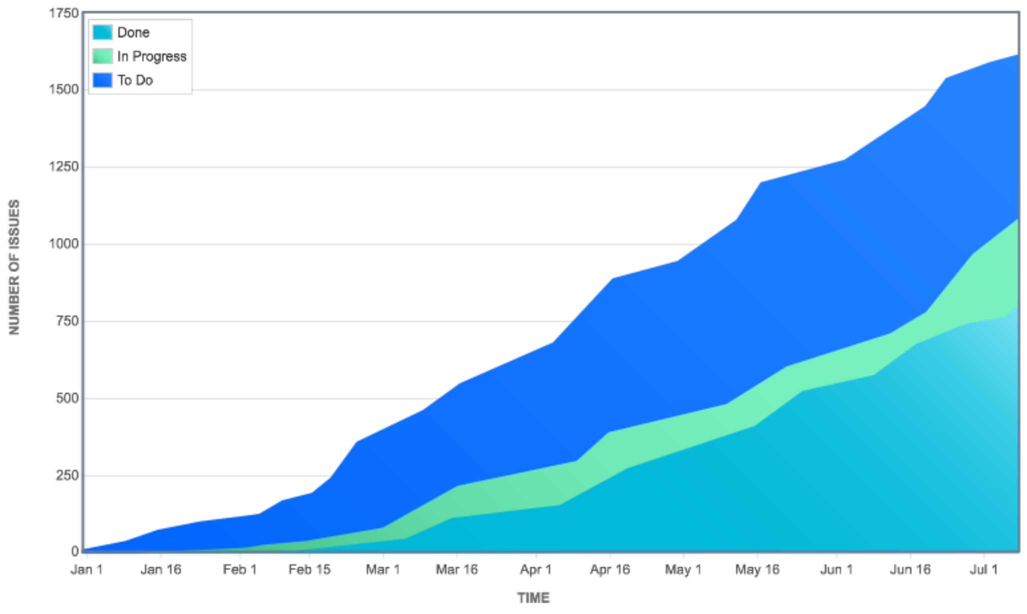

Figure 4.2: Kanban progress diagram example

4.3 DevOps approach

4.3.1 Introduction

It is not very easy to say what **DevOps** is, but we particularly agree on the definition given by Gartner [27], which sees DevOps as a shift of culture in the development process to aim at the rapid delivery of stable, high-quality software from concept to customer. Every stage of the development process must be automated, including security and stability tests, in order to get quality software and cut down the costs. Moreover, DevOps emphasizes the collaboration between the developer teams (Dev) and the operational teams (Ops). The word was coined in 2009 by Patrick Debois and, from that time, the DevOps culture has grown enormously in the IT sector. DevOps takes its inspiration from the work of old system administrators, who in the past were responsible for managing and building the infrastructure of a system, and the agile development process which, as said in Sec. 4.2, promotes collaboration and rapidity of development.

4.3.2 Why DevOps?

The creation of the DevOps paradigm was pushed by the fact that, in old organizations, developer and operation teams were concerned towards two opposite objectives. The former were requested by customers to release new features as fast as possible, things that goes in contrast with the request for operational people to maintain stability in the systems. The DevOps approach could solve this dilemma by associating development to deployment with the goal to rapidly deliver high quality software, through automated and repeatable processes.

4.3.3 The pillars of DevOps

In the previous sections we have seen the main definition of DevOps and the motivations that lead to its creation. Although there are many variations of the approach, everything is based on the following strong pillars.

- **Collaboration**. Everyone involved from software development to delivery should collaborate to improve the overall quality of the final product.
- **Automation**. There is the need to use and develop tools that can allow to automate the larger number of steps possible in the process.
- **Continuous integration**. This refers to the frequent merge of new feature into the shared mainline. By continuously integrating, it is possible to catch conflicts in a shorter time, thus reducing the risk of having troubles later on.
- **Continuous testing**. Quality and security tests need to be performed continuously, every time new software is deployed. The tests need to be automatic to aim at

finding problems early on in the project, without slowing down the developer's work. They help to ensure the quality of the product and find known security risks inside the code. These tests may include UI testing, load testing, integration testing and API reliability testing.

- **Continuous delivery**. This relates to the ability of a company to frequently release new features on the production pipeline. The automated tools and the continuous integration allow to always have well tested code, which can be easily deployed to production. The speed and the frequency of the delivery varies from company to company, and are affected by customers' requirements and internal business logic. Very similar, but different, is the concept of **continuous deployment**: with it the new changes are automatically deployed into production without the need of a manual approval, that is instead required by a continuous delivery state.

- **Continuous monitoring**. Together with testing, it is important to monitor the performance of the systems continuously starting from the development environment till reaching the production release. Services and machines should be monitored to detect possible stability problems and scale them accordingly to the traffic encountered.

- **Infrastructure-as-a-Service**. IaaS, described in Sec. 2.2, is essential in DevOps environment to remove the burden of managing and operating physical machines. DevOps engineers relies on Cloud Providers to deploy their infrastructure and scale easily with reduced costs. IaaS is programmable by nature and allows operators to write code to automatically deploy new elements to the infrastructure, without having to do it manually.

4.3.4 Difference with traditional ops

DevOps, as shown in Fig. 4.3, can speed up software release up to four times compared to traditional ops method [28]. This is due to the fact that manually deploying an infrastructure consumes most of the time in the software development process, and this time can be cut down by using tools which automate the deployment process and make it repeatable. Moreover, automated tests and reviews remove the risk to find bugs and vulnerabilities when the software is already mature and thus more complex to be fixed.

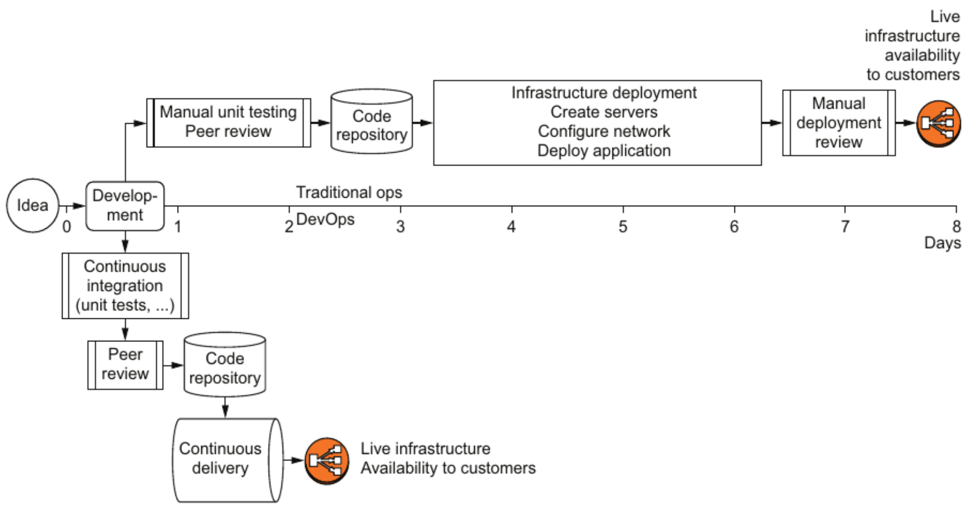

Figure 4.3: DevOps vs. Traditional Ops [28]

5 DevSecOps Model

5.1 Introduction

DevOps and agile methodologies aim at a fast delivery of the software, and most of the time, in order to meet the deadlines and customers' requests, developers and operators don't take the necessary attention to security. The reason is not that these people are not aware about security measures, but that they miss a stable and efficient plan to apply the correct controls. Therefore, it is very important to find out the best way to integrate security in the software development lifecycle. The general idea can be used as a ground base for many companies, but everyone should fine-tune it properly to meet its way of working.

In their daily job, every person involved in the DevOps pipeline focuses completely on the customer. In fact, product owners try to engage and keep the customer's attention high, developers are busy on developing as many functionalities as possible and operators are concerned about uptime and response times of the infrastructure.
On the other hand, security professionals focus on compliance with security standards, the number of security incidents and the discovery of vulnerabilities in the system, so driving towards a direction that hardly meets the goals of the other teams.
For this reason, it's hard to direct both parties towards a common goal, without following a new secure software development lifecycle.

The integration of security inside the DevOps development process takes the name of

DevSecOps. The term does not have an official definition but was naturally born when security started to be involved inside the *DevOps* process. It can be described as a way to automate the implementation of security measures at each step of the software development lifecycle, thus enabling the continuous integration and delivery of new software with security at the forefront [29]. This mechanism allows to reduce the need of human intervention and gives to security professionals the time to focus on less trivial issues, like finding business logic flaws in the applications and doing governance work.

5.2 Challenges

In a DevOps environment every step of the pipeline is automated, in order to be more efficient and fast in shipping new software. Security practices, if not adapted to this fast trend, struggle in following the process and risk to be left incomplete and badly implemented. Speed and adaptability is one of the challenges that a modern secure software development lifecycle must address. The solution would be the integration of what is called *continuous security*. It is composed of three areas, each of them focusing on a particular step of the development pipeline:

- **Test-driven security (TDS)** is the first step in the security pipelines and concerns the implementation of test security controls, which have to be as much automated as possible in order to follow the pace of the DevOps development [30].
- **Monitoring and responding to attacks** regards the detection of attacks and the strategy to apply in order to respond to a security incident [31].
- **Assessing risks and maturing security** moves the attention to security management and governance issues, because security is not only a practical subject but is also about documentation and policies.

These three areas should be tackled accordingly and adapted to the specific needs of a company. We will describe them more in detail in the next sections.

5.3 Shifting security to the left

The approach of shifting security to the left has been presented in many occasions as a good solution to integrate security in the development process [32]. In the past, security was seen as a factor slowing down the development of the software and therefore was often pushed to the "right" in the development stages, namely when the product has already been built. However, since vulnerabilities could have been discovered only in the later stages, it would have been very expensive to fix them.

Thanks to DevOps, it is now possible to fasten and automate the deployment of new releases, and so also security tests and scanning can be performed earlier on. They can be integrated inside the development pipeline and fired every time new code is committed to the repository. They should also be run periodically in order to catch new risks that can arise while the vulnerability database is being updated.

In this way the integration of security does not slow down the development process because the tests are performed automatically and can be fixed in less time, with respect to a security analysis performed just before the release to production. Moreover, given the fact that the tests are performed periodically, after the first scans, which will obviously detect the largest number of vulnerabilities, the process will be always faster and faster because the bigger amount of flaws should have been already fixed.

5.4 The DevSecOps pipeline

Each process in the DevOps development cycle is organized in one or multiple pipelines, which allow to set the rules and the steps to follow to compile, build and deploy the code to the production platforms [33]. In order to work properly and provide automation, the pipeline needs to be supported by a number of tools useful to perform specific tasks. It is also very important to make sure that these tools are able to inter-operate with each others, so that the output of one matches exactly the input patterns required by the next one in the pipeline.

The basic functionalities needed to properly build and secure a pipeline are:

- **Planning phase**. In this stage it is important to gather requirements from customers and start planning the features that need to be implemented by the developers. Therefore, the user stories which describe in general terms the functionality requested, are split in multiple actionable issues and added to the development board. From the security side, it is important to perform *Threat modeling* sessions (see Sec. 5.5 for further details) in order to have an understanding of the security requirements.

- **Development phase**. During this phase developers write the actual code on their own machines. It is good practice to follow directives to write clean and secure code already from the early stages. This can be achieved by using the *pair programming*[1] technique which consists in two people sitting in front of the same workstation, so that while one writes the code, the other focuses on checking that everything is correct. This technique is very effective, but sometimes, for time and resource reasons it cannot be applied.

  Integrating automatic tools like static *code analyzers* and *linters* is instead always suggested, because they can extend the quality of the code without impacting the productivity of the developers. Linters are tools which analyse the code to find programming errors, stylistic imperfections and bad constructs [34]. They are used to standardize the style of the code, so that it appears like to have been written by a single person, and can improve performance of the software by detecting bad code patterns. Although they may not directly enhance security, having well written code is already a good step in avoiding the presence of trivial flaws. They work by

[1]`https://bit.ly/3cL0cxk`

performing static analysis on the code and looking for patterns that can contains bad quality code or security flaws.

- **Code versioning phase**. During this phase the code is pushed to the code repositories. There are many open-source and proprietary platforms that can be used for code versioning and sharing. This is the place where the source code is placed and therefore it is extremely important to make sure that the environment is secured from outsiders' intrusion. In fact, if some criminals manage to get access to the source code, the company would risk a complete disruption, because most of the critical industrial secrets would be disclosed. Among the main source code repository platform we can count *GitLab*[2], *GitHub*[3], *Bitbucket*[4], *SourceForge*[5] and so on. GitLab, for example, offers a mechanism to fine-tune the permissions that users have over a specific project. Thanks to this it is possible to distribute the privileges between the developers, depending on their real needs (see Sec. 5.7 for additional information about the Principle of Least Privilege), in order to reduce the risk of a malicious employee tampering and dumping some sensitive portions of the source code.

- **Build phase**. Objective of this phase is to take the source code stored in repositories and build a binary or a container from it. Among the *Continuous Integration Platforms* there are multiple options (e.g. *GitLab*, *Jenkins*[6], *CircleCI*[7]) which differ for the different capabilities they can offer to the development environment. They help to define the actions to be taken when new code is pushed on the repositories, and so the commands to execute to automatically build and test the software.

[2]`https://gitlab.com/`

[3]`https://github.com/`

[4]`https://bitbucket.org/product`

[5]`https://sourceforge.net/`

[6]`https://www.jenkins.io/`

[7]`https://circleci.com/`

- **Test phase**. During this phase the software is tested to discover flaws and security vulnerabilities. The tests can either demonstrate the overall quality of the software checking the correct integration of different components, and the security of the system through the use of automatic security scans (see Sec. 5.6). The security scans can be static, if they analyse the source code without actually compiling and running the software, or dynamic, if they run on a compiled binary. Usually dynamic code analyzers are more sophisticated and can detect more advanced injections, by stressing the software with different kind of payloads.
  Moreover, since modern software is built on top of a large number of third party libraries, it is important to integrate some tools that perform dependency scanning to discover the presence of known vulnerabilities on them. These tools usually work by looking at a database containing known vulnerabilities (e.g. *CVE*[8]) and checking if the libraries used in the project are vulnerable. The vulnerabilities discovered are then classified depending on their severity, so that it is possible to filter only the ones that mostly impact the security of the product.
  The same tools described can be also implemented in the *monitoring phase* and periodically run to make sure that the software does not present any newly discovered vulnerability. During the build phase, depending on the configurations applied, the pipeline can be blocked if the vulnerabilities discovered have a big impact on the security.

- **Deployment phase**. Nowadays, most of the applications built with a DevOps approach are deployed inside containers, as described in Chapter 3. Therefore, the containers built during the CI phase need to be stored in a repository, in order to be available for download whenever needed. There exists public repositories (e.g.

[8]`https://cve.mitre.org/`

DockerHub[9]) or private ones (e.g. hosted on *JFrog Artifactory*[10]). It is extremely important to secure the container repositories because, if there are some flaws, an attacker may be able to tamper an image or push his own version of it, which then will be released and deployed. In this case, it is essential to secure the secret keys used to access the repositories and implement proper access control mechanisms on them.

- **Operating phase**. During this phase the containers and images present in the repositories are pulled and deployed on the actual machines provided by the cloud infrastructure. Most of the time the containers are organized in clusters and controlled by a container orchestrator (e.g. Kubernetes, see Sec. 3.4), which takes care of resetting failed containers and scale up depending on the traffic encountered.

- **Monitoring phase**. Upon reaching this phase, monitoring systems run indefinitely to provide insights about the health of the system and the resource consumption. Monitoring would not be possible without having an efficient and stable logging system (see Sec. 6.1 for further details) able to collect, analyse and show the information gathered. During this phase is also possible to detect some ongoing attacks, for example by examining abnormal behaviours in the software.

5.5 Threat modeling techniques

The following information about threat modeling are mainly taken from the book *Threat modeling: Designing for security* written by Adam Shostack [35], with the support of other external sources. Threat modeling is all about making more secure software. It allows to discover what can go wrong in a system and aims at proposing mitigation and controls for the risks identified. Threat modeling should not be considered a skill owned

[9]`https://hub.docker.com/`

[10]`https://jfrog.com/artifactory/`

only by security professionals, but rather every developer should apply it to his daily work. However, in order to be correctly performed, it needs some experience and knowledge about past attack scenarios and the structure of the system being built. An ideal threat modeling session should involve security professionals, but also product owners and engineers that know the requirements and architecture of the system.
Before starting, it's very important to have a clear idea about what is being built, thus the first step requires to draw a good picture or diagram of the system involved. This diagram should show all the main components, the data flow between them, the network boundaries and the actors involved in it. It is also strongly suggested to show the *trust boundaries* [36] to point out "who controls what", because most of the threats usually happen between two boundaries. The diagram can be either drawn on a whiteboard if all the people involved in the process have the possibility to personally meet, or using some drawing software.

After that a diagram of the architecture under study has been created, the next phase aims at researching the circumstances that can damage the system. In this regard, as previously mentioned, there are multiple threat modeling techniques, but for sure the two most worth mentioning are *STRIDE* by Microsoft (see Sec. 5.5.1) and *P.A.S.T.A.*. In Sec. 5.5.2 we will also describe a more light-weight approach called *Rapid risk-assessment*.

However, independently from the threat modeling technique used, in order to successfully identify threats, it is important to start by analyzing events and activities that are external to the system but constitute an access point to it. Moreover, it is advisable not to skip the attention from any threat found, even if this cannot be classified inside the category the discussion is about. On the other hand, it is necessary to stop the analysis at a certain level and not focus on threats which can potentially be exploited, but with a very low probability. In this way it is possible to focus the attention on items that could have a higher impact. Understanding when it's time to consider some potential flaws less meaningful is not always an easy task. In fact, the decision needs deep thinking and years of experience in the threat modeling field in order to be taken in the most appropriate way, because a bad evaluation can have critical impacts in the case of an incident.

Once a decent number of important threats has been identified, the next step, for each one of them, is to decide whether to mitigate, eliminate, transfer or accept the risk [37]. In the following lines the four options are explained.

- **Mitigating** the threat means applying some security controls which help to reduce the impact and/or the likelihood of an incident.
- In order to **eliminate** a threat the only possibility is to remove the feature that is affected from it. The whole team needs to decide if that specific feature is needed for the business logic.
- **Transferring** threats is about leaving to someone else the burden to handle them. For example, it is possible to pass the risks to the customers.
- It can happen that, after many researches and evaluations, the mitigation for a specific threat (whose feature cannot be deleted) are so expensive and hard to be applied that the company decides to leave everything as it is and **accept** the risk. This decision must be taken after accurate calculations of the impact of an incident over the cost to implement the control.s

Threat modeling is not performed in a single meeting, rather it might be necessary to organize multiple meetings to fine-tune the decisions taken and evaluate the progress and the results of what has been done. Moreover, the advent of new technologies and incident examples from other companies, can determine a change in the strategy over the controls applied. In addition, it cannot be said that threat modeling is complete, since there can always be some hidden issues that can damage the system; it is only correct to say that the threat modeling process has found and mitigated enough threats to be able to lower the risks to an acceptable level.

5.5.1 STRIDE

STRIDE is a a threat modeling technique created by Microsoft in 1999 [35]. The word is an acronym for *Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege*, as shown in Fig. 5.1. It allows to find the threats which can undermine six of the major security features that a software must have, in the following line ordered with respect to the attacks: authentication, integrity, non-repudiation, confidentiality, availability and authorization.

After having drawn diagrams and schema of the system to be analyzed, the objective of the security team is to consider, following the order, the six kind of attacks which compose the acronym, and find out possible attack scenarios related to those. During the brainstorming phase, it is possible to start working singularly and then join the other people to share the ideas, or start already working together on a common whiteboard.

In the following paragraphs we will talk more in detail about the six threats areas to be analyzed.

Spoofing threats Spoofing [38] is about stealing the identity of someone else and use it to authenticate as a legitimate user to the system. Spoofing attacks can affect different targets. It is possible to spoof a machine by, for example, spoofing ARP requests, IP addresses or DNS packets in order to redirect the network traffic to somewhere else.
People can be spoofed by performing *phishing attacks*, used to trick the victim into disclosing his own credentials.

Tampering threats Tampering [39] is about modifying data and tricking the system into accepting it as legitimate. If, for example, the access control lists on the file-system are not properly configured, an attacker may be able to modify some files with malicious code which the operating system may then execute. A malicious actor may also get access to a database and modify the values on it, or tamper the data transiting over the network.

Repudiation threats Repudiation [40] is about claiming to not have performed a specific action. In a well implemented system every action performed by either machines and humans should be properly logged in order to maintain a history of "who did what". Many times attackers, after having performed a malicious action on the system, try to breach the log service in order to delete the tracks of their intrusion. In fact, the *"if it is not on the logs, it never happened"* philosophy is very common. For this reason, even if repudiation attacks are not usually a threat themselves, they are always performed as a results of a more serious attack.

Information disclosure threats Information disclosure [41] is about giving to unauthorized parties the possibility to access sensitive data. This can happen through a process leaking memory addresses and secrets such as passwords or private keys. Databases or files that do not have a proper permission system can also leak information which can be beneficial for the attacker and damage the victim of the attack. Data can be stolen also when it traverses the network, especially in non-encrypted environments.

Denial-of-Service threats Denial-of-Service [42] (i.e. DoS) happens when the resources of a system are consumed and depleted. DoS attacks can be persistent, if their effects are still present even when the attacker stops executing his actions, or non-persistent in the opposite case. Moreover, they are divided into amplified and unamplified: in the first case, even a small effort from the attacker side can lead to huge damages to the system.

Elevation of Privilege threats Elevation of privilege [43] regards the possibility for an unauthorized user to perform actions he is not allowed to do, normally handled only by users having higher permissions. This can be performed by corrupting the memory of a process to bypass privilege checks or by exploiting buggy authorization configuration files and systems.

After having pointed out all the threats that fall inside the areas described before, the participants to the threat modeling session can start proposing mitigation and actions to be taken to properly handle the findings.

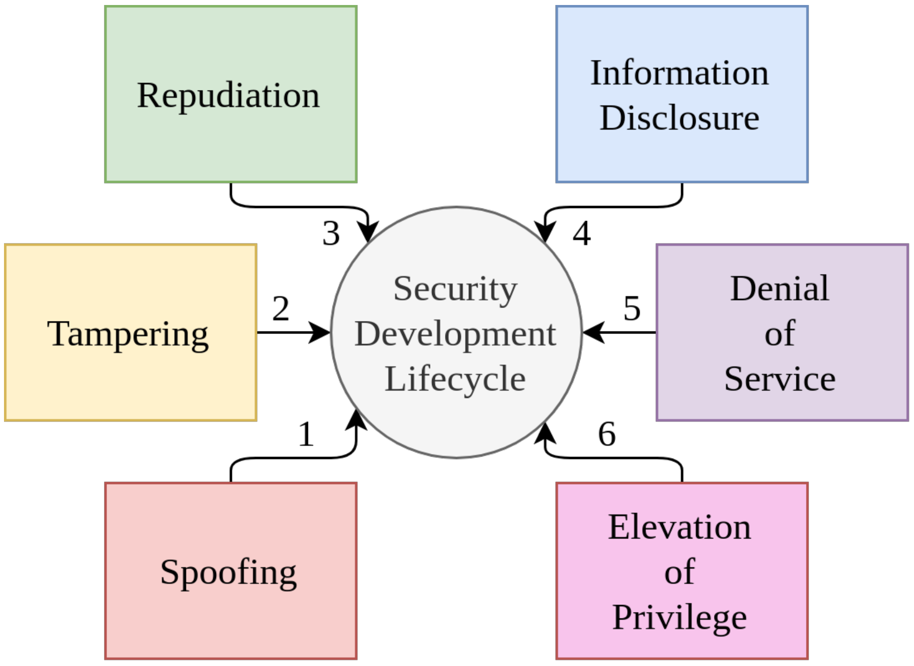

Figure 5.1: Stride threat modelling framework

5.5.2 Rapid risk-assessment

Classic risk-assessment methodologies provide huge improvements to the security of an organization, giving deep insights about the controls to be applied and the best practices to mitigate the risk of cyber incidents.

However, when implemented in a fast development environment as DevOps is, they can be seen as slow and unpractical to be applied. In fact, many times the development workflow is so fast that the security team cannot properly follow the pace with his work. Therefore, it is very important to define a risk-assessment and threat modeling strategy that is light-weight and fast to be performed for specific services. For this purpose, the

Mozilla Foundation[11] has developed a risk-assessment framework that takes less than one hour per project to be performed. It is called **Rapid risk-assessment** (RRA) framework [44] and, although it cannot be considered a complete threat modeling technique, it gives good insights about the impact of a service to the reputation, finances, productivity of the project or business.

The first phase requires gathering information about the target system, talking with the people involved in the development and the experts in the field. In this way, the security team can reach a decent level of understanding to be able to go further in the analysis of the potential threats.

During the second phase, the focus is on creating a *"data dictionary"* section in which all the information that the system handles is listed. In this way, it is possible to have an idea about the assets that need to be protected, evaluating their business level importance. In this regard, it is important to classify the data by following a hierarchical structure, similar to the one used in the military field to classify documents.

In the RRA the measurements of the risks follow the triad: *confidentiality*, *integrity* and *availability*. Then, each areas is decomposed into three impact areas being repudiation, productivity and finances of the organization, resulting in the creation of nine fields. The goal of the framework is to apply the rapid threat modeling approach to each one of the nine fields. The output of this phase is the creation of a risk table in which each of the three attributes of the security triad are evaluated against the three impact types listed before, calculating the impact, the likelihood and the resulting risk level.

During the last phase, the security team should make recommendations on how to fix or mitigate possible problems, given the output of the risk tables. It can also happen that the developers involved in the meeting figure out on their own the best way to face the threats discovered, thus making the threat modeling session successful.

[11] `https://foundation.mozilla.org/en/`

5.5.3 What to do with the results of threat modeling

In order to be effective and practical, the results of a threat modeling sessions need to be properly handled. One good approach is to create a ticket for each threat identified and put it inside an issue tracker (e.g. GitLab). The ticket needs to be placed in the correct board (so that the right people can work on it) and labeled accordingly to show that it is not a normal issue, but it comes from a threat modeling session. In this way developers and security professionals can have something concrete to work on, plan the time to spend on the task and define the requisites to fulfill in order to consider the issue solved.

Having a board with all the security measures that need to be enforced in order to secure the feature that is going to be implemented is crucial to make security part of the development lifecycle. Without a threat modeling session, it may happen that security flaws that could have been discovered in time during the design phase, come to the table when it's already too late and most of the features have already been built.

5.6 Automatic security scans

In order to discover flaws and vulnerabilities the software needs to be tested and scanned. There exist a plethora of automatic vulnerability scanners which are able to detect malicious patterns in the code, the use of vulnerable libraries (by comparing them with a database of known vulnerabilities) and incorrect configurations in general.

Security scanners can be distinguished into two big categories: *Static Application Security Testing (SAST)* and *Dynamic Application Security Testing (DAST)*, whose jobs are different but complementary [45].
SAST is a white box method of testing because it needs to read the source code in order to discover vulnerabilities, while DAST is a black box method and analyses the application at run-time. SAST can be run as soon as the source code is ready, while DAST needs the compiled application to operate, thus the static method allows to discover the vulnerabil-

ities earlier in the development lifecycle, providing cost reductions to fix them. However, DAST is able to detect problems that arise only when the application is run, thus giving a more accurate security analysis for the software. In general, both the two methodologies need to be implemented in the development lifecycle in order to have a complete security scan.

Nevertheless, automatic security scanners do not eliminate the need to perform manual penetration testing from time to time. In fact, although the automatic tools have become very efficient and precise, they are not able to discover more complex flaws that need the intervention of a human in order to be exploited. For this reason, whenever the software has reached a good maturity level, it is advisable to perform the so called *Red Team*[12] exercises, which consist in attacking the application using the same approaches that real malicious hackers would apply.

5.7 Principle of least privilege

NIST defines the *Principle of least privilege* as the practice of providing only the minimum access and time required to perform an action over a resource [46]. The application of this rule can avoid incidents caused by mistakes or malicious actions, while also reducing the auditing surface during a forensic investigation. In fact, with regard to the latter point, by knowing that only a small portion of the users or systems can have access to a specific set of resources, it is possible to shrink the area of investigation about the threat.

In order to enforce the principle it is useful to implement a *Role-base Access Control* (RBAC) mechanism. This solution allows to map users to specific roles, each one having different permissions on a set of resources. The granularity offered by the RBAC system is very beneficial because it allows to distribute properly the authorization among users over the resources, in addition to the easy management of the permission system.

[12]`https://www.sisainfosec.com/services/red-team-exercise/`

In a DevSecOps environment, that is full of resources which need to be accessed by multiple users, it is paramount to build a similar system. In an organization context, it can be useful to organize developers in a hierarchical level and give to each group only the permissions that are strictly needed to perform their work. For example, normal developers should have full access and right to deploy new releases only to the development environment, leaving the permission to access more business critical environments only to certain people in the company. The roles and permissions don't regard only humans but also machines. In fact, in a context where everything is automated, each system should be configured accordingly to receive only the permissions needed to perform the work for which it is designed, so that, in case of an attack, the set of actions that the malicious actor could exploit would be limited.

Moreover it is advisable to split administrative powers among multiple parties, each one capable of performing a small set of actions on some resources. Doing so, we remove the need to have a single user with super-admin permissions, which can be dangerous either in the case of a human error and in the case of a cyber-attack.

5.8 How to communicate with other teams

Communication is an essential pillar of any software development company, since the work is never done by only one person but involves multiple professionals having different skill-set from each others. If communication between developers can be challenging, it can be even harder for security professionals to find out the best way to interact with other people inside the company. This is because, as said before, security work has some objectives that may slow down the development process, thus meeting the dislike of developers, who instead are focused on releasing good software in the least time possible.

The *Global Developer Report* [47] talking about DevSecOps published by GitLab in 2019 gives good insights about how companies in the world are securing the DevOps

approach, and how the communication between developers and security professionals is changing. The report claims that around 23% of the developers that have taken part to the survey think that security is badly integrated in their organizations, while 25% of them are happy with the security work performed. The dissatisfaction can be caused by the fact that a company may still completely miss the presence of security professionals, or that, even if these figures are present, those are not able to guide properly the developers in writing more secure code and catching vulnerabilities at the earliest stages.
Troubles are not only caused by security professionals which are not able to properly instruct the developers and standardize the security measures to apply, because, as the Gitlab Developer Report [47] emphasizes, nearly 49% of the security professionals surveyed struggle in convincing the coders about fixing vulnerabilities as a priority. As said before, this can be given from the fact that security work can slow down the velocity of the software development process, even if, a well implemented DevSecOps process, should not impact too much the developer's output. In fact, the report [47] claims that by embracing correctly a DevSecOps approach, security teams are three times more likely to discover bugs before the code is merged into the *Master branch* and 90% more likely to be able to test almost all the code-base in the first stages. These two factors are extremely important for reducing both the costs and the time needed to fix vulnerabilities when they have been already deployed to production. However, in order to be convincing, a security expert needs to have good presentation and communications skills that can allow him to convince developers about the importance of implementing a certain mitigation. It is also a critical fact that, most of the times, security professionals need to interact with people which are not familiar with the technology field. In this case, it is very important for a security engineer to be able to explain complex concepts in an easy way, taking care of explaining the economical repercussions of a security control over the business of the company.

In some cases, it can happen that an incident or a decision that has lead to something bad damages the trust that developers pose on the security team, pushing them not to anymore follow the suggestions provided. This can have a very bad impact on the future work, since personal relationship and accountability is essential for the correct progress of an organization.

Nevertheless, an open debate regards the decision about who is responsible for security. Someone think that developers, being the ones who actually write the code, are the main responsible for the security of the system, while others believe that the hardest burden should be placed on the shoulders of security professionals, for the fact that it is their job. We retain that there is no correct answer and that both parties should work and communicate with each others in order to deliver secure and good quality software, in the least time possible.

5.8.1 Security training and tools

In order to make sure that developers work with a security mindset it is beneficial to provide them with some guidelines that are both complete and easy to follow. Firstly, it is very beneficial to periodically organize security training sessions [48] in which security professionals present to the rest of the organization some topics related to the security fields. These short seminaries should highlight a particular cyber attack, describing the risks derived and proposing a good coding style useful to mitigate the risk of suffering from this kind of threat. Despite not being very complex and detailed, these presentations offer an important boost in the security awareness among developers. And being security aware means writing more secure code already from the beginning, avoiding the risk of taking trivial flaws to the later stages of the development.

Since human mind works well in an organized and standardized environment, it is also crucial for security professionals to create a sort of checklist that developers can examine to make sure to be following the security best practices. One example is the

SWAT checklist[13] which lists all the items to take into considerations when developing web applications. The content of these documents can be then carefully checked against the current version of the software and, in case of bad implementation, new issues should be added to the development board.

[13]`https://software-security.sans.org/resources/swat`

6 Incidents identification and response

Collecting logs is extremely important for forensic analysis in case of a security incident and they represent a key factor for the implementation of a strong DevSecOps framework. In fact, they can give a clue of what happened, list the systems impacted and the amount of data being compromised. Moreover, although they cannot be considered security controls, as they don't provide any additional security to the system, their presence is extremely important and can help to consistently reduce the impact of a potential attack.
Giving their importance in facilitating and enhancing the security workload, it is valuable, in the scope of this thesis, to describe more in details how the entire logging system works. In the following sections we will analyse the layers that compose a standard logging environment and we will give some guidelines about the steps to take to perform forensic analysis and recover from an incident. The information are mostly taken from the book *Securing Devops: Security in the Cloud*, written by Julien Vehent [28], but other sources are referenced as well.

6.1 Logging layers

Logs are originated from many different sources and their level of verbosity depends on the particular implementation or the configurations decided by the developers and operators.
The logging pipeline is usually divided into multiple layers, in which each layer's output is given as input to the next layer. In this way, starting from the raw logs (i.e. not polished

information, verbose and not clear) it is possible to manipulate and process the data in order to give as a result a human-readable dashboard from which analysing the events. In the following sections we will describe the main layers that compose the logging pipeline.

6.1.1 Collection layer

The collection layer is responsible, as the name suggests, for collecting the logs originated by applications, network components and other kind of systems. There are four main broad categories of logs which will be analyzed in the following paragraphs:

- **Systems logs** are usually generated by web servers and services running on top of the operating system and are usually full of information [49]. They are very useful to inspect the sessions opened on the system, access history on web servers, actions taken by software that resides very close to the OS, and the traffic that traversed a firewall. In Linux machines there are two categories of logs: the *syslog* and the *system calls audit logs*. The syslog is the oldest and more used way to collect logs and, apart from writing to files, it offers the capability of sending logs through the network. It describes the component that has originated the log and the severity level of it, so that it is easy to filter all the logs depending on specific search criteria. However, this approach doesn't provide a common granularity level among the systems being logged, leaving discrepancies in the visibility and prioritization of the information. In order to solve this problem, and also give to the developer the possibility to decide how verbose the logs should be, *syscall auditing* was introduced. Additionally, this solution offers a higher visibility because the events logged are directly generated by the Kernel, paying the cost of consuming a huge amount of memory.

- **Application logs** are generated by software and in the case of in-house developed product, developers can implemented as many logs level as requested. Usually

the logs are streamed through the application standard output and collected into a logging pipeline. In order to be understandable, it is important to store the logs in a structured format (e.g. JSON, XML), standardize the timestamp format to make it easy to reconstruct the temporal line of events, identify the origin of the events by defining the unique information about the logs source, and customize each application to write arbitrary and personalized data.
Since it's developers' task to decide how much and what to log, it is important to provide them with a guide on how to best implement a logging solution. One very valuable guide that we have encountered is the *OWASP Logging Cheat Sheet*[1] which is quite easy to follow and gives guidelines on the events that an application should record. It's not worth for the objective of the thesis to go into the details of what is explained in the cheat sheet, but the general takeaway is that it's important to log failed and succeeded authentications, input insertions, calls to sensitive APIs, data changes, suspicious behaviours, extreme memory consumption and modification to the configurations.

- **Infrastructure logging** catches events generated from the underlying infrastructure, being it company managed or outsourced from a Cloud Provider. Regarding cloud environments, among the AWS services there is one called *CloudTrail*[2] whose job is to keep track of every operation performed either on the System Console and through the CLI over the cloud infrastructure. The information is very detailed and is usually stored inside *AWS Simple Storage Service (S3) buckets*[3], in which they are maintained for a certain amount of time, before being destroyed to release memory space. The only problem is that the streaming of information is not real time and

[1]https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

[2]https://aws.amazon.com/cloudtrail/

[3]https://aws.amazon.com/s3/

so a potential attacker may have a small window of time to perform his malicious actions without being tracked.
Network logs fall also in this category, and it is paramount to record them in order to analyze the internet traffic in case of an incident. A very useful tool in this regard is *NetFlow*[4], developed by Cisco and easily integrated inside the AWS infrastructure. It collects data from routers and network devices and routes the finding to a centralized system.
Some of the logs present in this layer can be redundant with respect to the ones present at the system layer, and they pay their additional security level with a major complexity for the analysis.

- **Logs from third-party software** really vary in details and verbosity depending on the service. Sometimes it is possible for the customer of a service to decide the logging level to be used, even if it is not always the case. As an example we can take *GiLab Logs*[5] which monitor the access to resources, the person who performed a specific action and many kind of other metrics.

6.1.2 Streaming layer

After the logs have been collected by the collection layer, they are sent to a *message broker* which belongs to the streaming layer. A message broker, according to the definition given by IBM [50], is a software that enables communication between two applications or services, one of them being the publisher and the other being the consumer. The publisher pushes data through the message broker, which contains a queue to handle more messages. Then the message broker decides how to route the messages to the correct consumers.
The consumers belong to the *analysis layer*, and their job is to analyze the messages

[4]`https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html`

[5]`https://docs.gitlab.com/ee/administration/logs.html`

and behave accordingly. There are multiple message brokers in the market, but one that is worth noticing is *Apache Kafka*[6], being one of the most used as it is also capable of maintaining the logs in its own memory for some amount of time.

Brokers can implement two modes to share the messages to the consumers: the *round-robin mode* and the *fan-out mode*. The first works by sending a message to a single consumer within a group. Fan-out mode is instead used to distribute the same topic to multiple consumers. The choice of the distribution strategy to adopt depends on the type of messages and on the system details.

6.1.3 Analysis layer

The consumers receiving messages from the broker belongs to the analysis layer [51]. Here, as the name suggests, the logs are evaluated and analyzed to extrapolate useful information or simply store them in the databases.

The simplest type of consumer, receives the raw logs and stores them in a database where they are maintained for a certain amount of time. Storing the logs is very important because it allows to perform forensic analysis for incidents that may had happened long time before the relative discovery. Database and file-systems used to save logs belongs to the *storage layer*.
There exist also consumers capable of monitoring the logs and sending alerts to operators in case of strange events or abnormal behaviours have been detected. The alerts can be about excessive memory usage or full disk storage, but can also regard security threats. In this case, it is paramount to implement an efficient and accurate model to categorize logs and assign to them a risk level, because if the evaluation is not done correctly, there is the risk of either missing some important events or firing too many false alarms. Moreover, it is usually the case that the operators are provided with some sort of dashboard to visualize the collected logs, in a layer called *accessing layer*. This layer is very important to provide

[6]`https://kafka.apache.org/`

visibility and must be properly secured from intrusions because it can be victim of attacks and tampering from hackers trying to hide their malicious footsteps.

Talking about implementation, we can say that consumers should be small pieces of software, completely dedicated on a single task. Since their job is requested only whenever important logs are passed by the message broker, it is smart to give them a serverless architecture, deploying them, for example, by using *AWS Lambda* (see Sec. 3.5.2).

6.2 Forensic analysis

In case of an incident the best way to reconstruct what happened is to read the logs [52]. This can be a very long and tedious work but, if the logging system is clear and concise, it is possible to discover the origin of the flaw and the assets that have been damaged by the attack. The investigation process needs to be taken carefully, reducing to the minimum the interaction with the system so to avoid to spoil the information about the past events.

The logs that usually provide more insights about a cyber incident are related to access or authorization requests, the IP address of the machine originating the attack, the modifications of the file system and the injection of attacker payload. Once the forensic analysis phase is concluded, the security team will have a picture of the assets that have been compromised, the dimension of the attack and its impact and the economical loss derived from it. With all this information gathered, the next step consists in preparing a plan to recover the correct functioning of the impacted assets, taking care of resolving the vulnerabilities that have been exploited in the attack.

6.3 Recovery

The recovery phase is crucial to try to restore the normal functioning of the system exploited. Since broken and malfunctioning systems cause money loss to an organization, it is paramount that this phase is as smooth and fast as possible. However, this cannot be done so easily if the company does not have a proper recovery plan designed to face such cases, and this fact, depending on the scale of the incident, can lead to fatal damages for the business.

If we take as an example the disruption of a service running in the cloud, the use of the Infrastructure-as-a-Code (described in Sec. 2.4) paradigm can reduce the reconstruction of the broken platform to a matter of running a script which will re-deploy automatically the service [53]. Therefore, it is very important to script as many of the functionalities as possible and reduce the manual work so that, in case of an incident, everything can be recreated in a very easy way.

7 Case study: Awake.AI

In the previous chapters we have described the theory that literature dictates regarding the development frameworks, the best practices for implementing specific technologies and the approaches to follow to integrate security in the process. However, the theory does not always match the practical counterpart, and for this reason, it is valuable to apply the knowledge described before on a real use case scenario: the approach used in *Awake.AI*[1].

Awake.AI is a software company whose objective is to became the leader in digital maritime industry by 2025. Its focus is on the logistic of maritime transportation and the exchange of information between the multiple parties involved in the shipping industry. The goal of the company is to apply data analytics and machine learning models to reduce the environmental impacts and costs that affect the maritime industry up to 40%. It was founded in January 2019 and, at the time of writing, it has almost 30 employees.
The team comprehends front-end and back-end developers, machine learning and analytics professionals, devops experts, experts in the maritime field and cybersecurity professionals.

In the next sections we will describe the strategies used to develop the product, taking particular attention to the way the security work is integrated into the development lifecycle and how the approach followed can help reducing the costs derived from applying security measures.

[1] `https://www.awake.ai/`

7.1 Development process

Company's technical experts are spread between multiple *features teams*. The job of each team is to take care of a certain set of functionalities and services and they comprehend all the required expertise needed to perform their job independently. In fact, inside a team there are front-end and back-end developers, operations professionals, testers and security professionals. In the ideal case a team would be able to implement the features assigned without having to ask for help to people belonging to other teams. However, this fact does not restrict the possibility for members of different teams to interact with each others to synchronize and coordinate the work on common features.

The company follows an agile development methodology which takes inspiration both from Scrum (see Sec. 4.2.1) and from Kanban (see Sec. 4.2.2). Likewise the original Scrum method, the time-span is divided into Sprints whose length can be either two or three weeks long, depending on the particular deadlines or festivities present during the period.
Four Sprints are then grouped together into what is called *Mission*, which is the time needed to release major features to the customers. Before the beginning of a Mission, the teams spend an entire week planning and designing the features to implement in the next period, spreading the tasks between the four Sprints, depending on their priorities and requirements. During this session, product owners have to explain to developers the new features that need to be implemented: this is done through the use of *user stories*. User stories are statements that define which capabilities a specific user would like the software to offer so as to satisfy a certain need. In other words, the stories give a general and very high level overview of which features need to be implemented, and are taken as a starting point to describe the actual objectives that need to be accomplished. By using their expertise and experience, developers are asked to translate the stories into actionable tasks, and also to try to estimate the time needed to implement each specific functionality.

The development framework makes use of multiple boards, hosted on GitLab, to which the issues are attached and labeled depending on their state of progress, like it is suggested by the Kanban model. In general, when an issue is opened, it is placed in a general lane together with all the other issues, then takes the *To be done* state whenever is planned to be taken into consideration during the current Sprint, is moved to the *Doing* state whenever someone is working on it, then reaches the *Review* state when the task has been completed and needs to be reviewed by another person and, eventually, reaches the *Closed* state whenever it has been completed. It can also happen that the work on an issue cannot continue anymore, either because it has been discovered not being technically feasible or because the implementation of other functionalities is required before the task can be completed. In this case the issue is labeled as *Blocked* and remains pending until some other factors comes to unlock it.

As claimed by the Scrum literature, every day the teams perform a daily meeting to talk about the work done in the previous day, describe the planned work for that day and ask for help in case of necessity. The discussion is lead by the Scrum master, who usually presents the team board on his screen and gives word to the colleagues following the order of the issues present on the *Doing* lane. In the company the team daily meetings usually run very smoothly and are kept under a time limit of fifteen minutes. Although there are not always security related tasks to be discussed during the meeting, it is good approach for the team security expert to participate so to have an understanding of what the developers are building and provide security support in case of necessity.

At the end of every Sprint the teams need to prepare a major release candidate containing all the new features implemented in the last three weeks. Continuously delivering new versions of the software keeps alive the interaction with the customers, who can give timely feedback and comments. In order to make sure that the functionalities released to the customers meet specific quality criteria, the company makes use of four different environments: *development*, *stage*, *qa* and *production*. All these environments contain almost the same cloud infrastructure and services and are used to test and develop the software.

Whenever a developer commits some changes to GitLab, a pipeline starts building the image of the service involved and deploys it into the development environment. Therefore, this environment contains all the newest changes, but also a big amount of bugs, since at this point the code has not been yet properly tested. Upon reaching a certain maturity, the changes are deployed into the stage environment in which the software receives a deeper level of inspection and testing. Next step is the deployment into the qa environment, that is basically a copy of the production environment and is used to have a clear idea of what will be the final product released to the customers. After passing all the quality and acceptance checks, the new functionality is released to the production environment, thus becoming available to the customers.

Apart from quality and testing purposes, the different environments are useful to make sure that, before being released to the customers, the security of the software has been heavily checked. In fact, by running security scanners and code analyzers already inside the development environment, it is possible to catch vulnerabilities in the earliest stages and fix them more easily. This fact does not mean that the production environment is free from vulnerabilities, but allows to discover in time most of the flaws, before they can even be exposed in the public environment.

7.2 Overview of the technology used

The company relies completely on a cloud-based infrastructure provided by AWS (see Sec. 2.3), which gives the opportunity to scale quickly and reduce the operational costs. Among all the services, the ones more used are EC2 instances, Lambda functions, Relational Databases and data analytics services, which are quickly described in Sec. 2.3. In particular, the Infrastructure-as-a-Code paradigm (see Sec. 2.4) is heavily used and, as a result, most of the infrastructure is scripted by using HashiCorp Terraform. IaaC not only helps to scale in a fast and reliable way, but gives a good opportunity to security professionals to implement policies and controls, and safely patch vulnerabilities in the cloud. Moreover, as also stated in Sec. 6.3, it plays an important role in the reconstruction of the infrastructure in case of a major disruption. Therefore, it is an important factor contributing to the reduction of costs to be faced in a secure software development process.

Talking about the architecture of the system, we can say that everything is split into micro-services built inside independent Docker containers (see Sec. 3.3). The networking and communication between the micro-services is orchestrated by Kubernetes (see Sec. 3.4), which provides multiple replicas of the same service in order to ensure the high availability of the whole system. Following the best practices regarding container security, the containers, whenever it is possible, present a *read-only file-system*, so to prevent malicious actors from injecting arbitrary payload into the services. Whenever it is not possible to set the entire file-system in read-only mode, because the application needs to write some information in specific folders, the approach taken is to mount the paths that require write permissions on separated volumes, so that the rest of the file-system can have only read permissions. This solution makes sure that the *Principle of least privileges* (see Sec. 5.7) is applied.
During the *building phase* (see Sec. 5.4), the containers are scanned to discover the presence of known vulnerabilities affecting the dependencies used, through the help of automatic security scanners (e.g. *Npm audit*[2] for JavaScript projects). In the case some severe vulnerabilities have been discovered, the building phase fails and developers have to update the vulnerable dependencies before being able to deploy the new version of the software. This preliminary security approach can already save a lot of time and effort that would be needed in case the vulnerabilities would have been discovered at later stages. Moreover, in order to reduce the attack surface to the minimum, the container images are built starting from very light-weight base images which contains only a reduced amount

[2]`https://docs.npmjs.com/cli/audit`

of libraries. An example, in case of the Linux image, is the use of the *Alpine*[3] base image which provides only a small set of necessary libraries. Starting from that, it is possible to install only the libraries that are really needed to run the application.
Another good practice followed, is the one that suggests to build the images in multiple stages, by taking some libraries from a base image and copying them into a second image. In this way, instead of stacking an entire image on top of another, it is possible to copy only the required parts, thus saving resources and limiting the attack surface.

All the images that are used as bases to build the company's images are stored in a private repository hosted on *JFrog Artifactory*[4], and are constantly updated in order to have the latest security vulnerabilities solved. It is crucial to periodically rebuild the images in order to get all the new updates and avoid the risk to face again vulnerabilities that have already been fixed. In this regard, it is useful to avoid using caching mechanisms when re-building a container to make sure that everything is pulled updated from the internet. Storing always updated images inside the private repositories removes the need to update them manually when they are used inside a *Dockerfile*, thus allowing to solve the security issues at the root level and eliminating the costs to be faced in order to perform the same fixes for many different services.

As said before, the company relies on Kubernetes as a container orchestrator and runs a single cluster for each of the four main environments. There are strong *Role Based Access Control*[5] policies put in place inside the cluster to give the right amount of permissions to the users and, as in the case of Docker containers, the file-system is set to read-only mode whenever is possible. The applications running inside the pods log on their *standard out*, which will then redirect it towards a *syslog*[6] to display the information

[3]`https://alpinelinux.org/`

[4]`https://jfrog.com/artifactory/`

[5]`https://searchsecurity.techtarget.com/definition/role-based-access-control-RBAC`

[6]`https://en.wikipedia.org/wiki/Syslog`

on *AWS CloudWatch*[7]. This solution provides visibility and transparency on the applications running in the cluster, highlighting abnormal behaviours or excessive resource consumption.

Many events and API calls are not handled by a containerized application, but rather deployed in a serverless fashion by using the *AWS Lambda* service (see Sec. 3.5). Lambda functions are used for pieces of code which have to run upon the firing of an event and die after a short time span. By using them, the company does not have to operate a specific infrastructure and can just concern about the quality of the code. The only disadvantage is that these so-called *serverless applications* are very hard to scan for security purposes by using the standard security scanners. That being said, it is paramount to manually perform security testing on them by trying, for example, to inject payloads from non-trivial sources, because, as said in Sec. 3.5.4, serverless solutions present more entry-point options for user input injection with respect to traditional applications. This can be a time-consuming task, but it is very important to guarantee that there are no trivial source of attacks.

7.3 DevSecOps

Security professionals are deeply integrated into the development framework. In fact, they are members of the feature teams and actively participate in the daily meetings. With their work they provide insights to developers about the best practices to follow to securely implement a particular feature, they help hardening the network communications between the components and they highlight security vulnerabilities that need to be fixed. The interaction between security experts and developers is very smooth, thanks also to the fact that all the software engineers are well prone towards security. The communication happens either via face-to-face meetings or through the use of online chatting and video-

[7]`https://aws.amazon.com/cloudwatch/`

calling, which ensures that everyone is updated about the progress of the development.

Threat modeling sessions are starting to be rolled out in this period, and unfortunately most of the services will be threat modeled a posteriori, because they have been built before a secure development framework was ever instantiated. Therefore, there is the risk that some major architectural flaws will be discovered when the system is already in place, thus causing big expenses to perform the required fixes. However, when a strong threat modeling framework, which can be based on STRIDE (see Sec. 5.5.1), will be instantiated, all the future services will be analyzed during the design phase, so to point out already at the beginning the security requirements that need to be fulfilled.

Automatic security tests are put in place and they run each time new code is deployed to the environments. In this way, if major issues are detected, the entire build in the pipeline fails and developers can work to try to fix the problem.
Moreover, during the night, multiple dependency security scanners (one for each programming language used) are run to detect the presence of known vulnerabilities in the libraries imported. The scanners work by checking if the libraries used are mentioned in a database of cyber-threats which highlights the dependencies that have been discovered being vulnerable.

Some days before the end of each Sprint, the security team reviews the most impacting findings from the dependency scanners so to discuss which action needs to be taken to solve them. From the review it can come out that vulnerabilities affecting some libraries are considered not harmful because, in the particular scenario in which they are used, not all the requirements needed to exploit them are fulfilled. The action of suppressing some vulnerability findings speeds up incredibly the development workflow because it reduces the time spent on fixing useless stuff.
On the other hand, if the vulnerability is considered to potentially have a big impact on the services, the person responsible for the service affected is called to update the library as soon as possible. And this, although it may seem to be a trivial task, requires

some reasoning because performing an update to a package can break completely the correct functioning of the application. This is one of the points in which security fixes can present expenses because they may require re-thinking the architecture of the system. Nevertheless, by following this approach, the flaws are discovered very early and their fix can be cheaper than what it would be by discovering them in later stages.

For what concerns the code written in-house, it is very important for developers to be able to write secure code. For this reason, security awareness in the company is taken into serious consideration. As a matter of fact, the security team periodically organizes security training sessions for developers in order to provide some insights about different topics related to the cybersecurity world. Although being quite simple, the training lessons give developers a grasp of security topics that can be taken as a starting point for further detailed learning. This factor is extremely important because it gives the opportunity to write code having a security mindset, thus reducing the risk of creating vulnerable code which may be expensive to fix in later stages. There exist also tools that guide developers into writing more secure and quality code: *linters* (see Sec. 5.4). Linters embedded inside *Integrated development environment*[8] (IDEs) analyse in real-time the code being written to detect the presence of bad patterns that can lead to known vulnerabilities. Despite not being able to discover the presence of more complex flaws, they allow to skim the most trivial ones.

It has been planned, for the next future, to start rolling out a *Red teaming exercise*[9], which consists in hiring a team of professionals penetration testers to try to break the services using the same approach exploited by malicious hackers. This exercise has not been done yet because it is effective when the services reach a certain level of maturity and complexity, which is not the case at the moment. However, this will surely be done before the first official and monetized release to the customers.

[8]https://en.wikipedia.org/wiki/Integrated_development_environment

[9]https://www.sisainfosec.com/services/red-team-exercise/

In addition, the security team continuously works to update and write new documentation containing checklists, guidelines and learning material for the developers.

7.3.1 Incident response and recovery

Every company will, sooner or later, face a cybersecurity incident so it is a good practice to be ready for when this will happen. As we said in Chapter 6, the first requirement to fulfill in order to detect an attack is to have an efficient logging system. It is paramount for the logging system to be complete but, at the same time, not too verbose because otherwise there is the risk of missing important events.

However logs don't have any value if they are not properly analyzed and processed to extrapolate interesting information. Moreover, there is the need to provide visibility regarding the security of the cloud. In this respect, the company uses *AWS GuardDuty*[10] and *SecurityHub*[11], which monitor the traffic reaching the instances running in the cloud to detect malicious actions or misconfigurations.

For what concerns the recovery phase, in case of a major disruption of the cloud infrastructure, the company relies on the fact that every resource deployed in AWS has been scripted by using the Infrastructure-as-a-Code paradigm. Therefore, it is just a matter of launching some scripts and everything would be automatically recreated.
More investigations need to be done regarding the approach to follow in case of a data leakage, describing the solutions that need to be applied to ensure that *GDPR*[12] and other governance rules are taken into consideration.

The security team in Awake.AI has a template file for writing a report describing the cybersecurity incidents that may hit the company. The document clarifies the timeline of the events that led to the compromise, describes the damaged assets, the actors involved

[10]`https://aws.amazon.com/guardduty/`

[11]`https://aws.amazon.com/guardduty/`

[12]`https://gdpr-info.eu/`

and presents the actions that have been taken to recover from the incident. Moreover, suggestions on how to improve the system in order to mitigate future flaws are included. The incident report is a quite sensitive document and so it needs to be shared only with certain parties, who can be the Head of Engineers, the CEO and other leading figures. In addition, it is important to create an environment of trust and transparency inside the company which promotes the report of security incidents, without accusing or victimizing anyone. In this way it is possible to drastically reduce the risk that someone, fearing the consequences, decides not to report a mistake that has lead to a compromise.

8 Conclusions

A development framework that stands on the fast and continuous delivery of software needs that a precise and adaptable security framework is applied, in order to be able to deliver high quality and secure applications. The method applied in Awake.AI is, for sure, not perfect, but the results achieved in the last months regarding the improvements to the security framework have been substantial. The various mechanisms described in the thesis have, in fact, increased the speed and efficiency of the development framework, and reduced the friction between developers and security professionals. This is demonstrated by the fact that the throughput of the development has not been impacted from the integration of the described security framework, but rather the number of vulnerabilities and flaws detected at earlier stages has been substantial. The latter has been achieved through the integration of automatic security scanners which have contributed into *"shifting security to the left"*, thus reducing the number of vulnerabilities which end up in the production environment. Being a startup company, the amount of code written in the last months in Awake has been huge, and this fast pace would not have been possible without a proper secure development framework put in place. In fact, especially in startups, the available time is very restricted and thus the pressure which developers have to stand can easily make them forget about writing secure software, which is discovered being vulnerable only in later stages.

The great results achieved comes from the adoption of multiple, even sometimes minimal, best practices, which together allow to create a flexible and efficient security framework. Among them we can highlight the security training sessions provided to the developers that have allowed to build a security mindset which can help in reducing the presence of security flaws already during the coding phase and facilitate the discussion about security topics. The automation provided by the DevOps paradigm has allowed to integrate security scanners in the pipelines, giving the possibility to discover vulnerabilities as early as possible. The active participation of security engineers to the daily meetings has helped in reducing to the minimum the friction between the pace of development and the delivery of secure code. The adoption of the *Principle of Least Privileges* (see Sec. 5.7) has restricted the access to critical resources present in the cloud or running in the Kubernetes clusters, thus allowing to reduce the risk of malicious actors tampering or accessing sensitive data. The integration of a strong Infrastructure as a Code culture has speed up the deployment of new systems in the cloud and, in the case of a major disruption, will play a key role in the fast reconstruction of the infrastructure.

Talking about the things that should be improved in the company, we can list the fact that architectural documentation is a bit lacking or obsolete. This often leads to the need of reverse engineering a functionality developed by other people in order to understand it and work on it. However, the lack of a proper documentation is a common trend in many companies and especially in startups, either because systems change so often that a document would became obsolete quite soon, and either because developers prefer to spend their time implementing new features rather than writing instructions. In this regard, it may be interesting to study further how to best integrate documentation writing to the development of new software.

The thesis didn't analyse a mechanism for prioritizing the work between what is considered the development of new features and what is instead the work done for fixing security vulnerabilities. Therefore, it may be interesting as a future work, to find out a proper framework capable of providing an easy way to decide whether to work on developing a new feature for the customers or spend the time to solve a vulnerability. It is certain that the decision would depend on the severity of the vulnerability encountered and on the deadlines that need to be respected to release a new functionality.

In conclusion, it is valuable to mention that the presence of a good based security framework in Awake has been used as a key factor for attracting the attention and trust from potential customers. Therefore, security must not be considered just a cost for a company, but can bring value and indirect revenues by itself.

References

[1] P. M. Mell and T. Grance, "The nist definition of cloud computing", 2011. DOI: 10.6028/nist.sp.800-145.

[2] *Global cloud infrastructure market q4 2019 and full year 2019*. [Online]. Available: https://canalys.com/newsroom/canalys-worldwide-cloud-infrastructure-Q4-2019-and-full-year-2019 (Accessed: 13.7.2020).

[3] Q. Devery, *The three cloud computing service models*. [Online]. Available: https://www.paranet.com/2012/06/18/bid-128267-the-three-types-of-cloud-computing-service-models/ (Accessed: 13.7.2020).

[4] S. Mathew and J. Varia, "Overview of amazon web services", *Amazon Whitepapers*, 2020.

[5] J. Clement and Feb, *Amazon web services revenue 2019*, Feb. 2020. [Online]. Available: https://www.statista.com/statistics/233725/development-of-amazon-web-services-revenue/.

[6] *Regions and availability zones*, 2018. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/?p=ngi&loc=2 (Accessed: 13.7.2020).

[7] *Debt information for teens: Tips for a successful financial life, including facts about the economy and personal finances, money management, interest rates, loans, credit*

cards, predatory lending practices, and resolving debt-related problems, 2018. [Online]. Available: `https://aws.amazon.com/about-aws/global-infrastructure/` (Accessed: 22.9.2020).

[8] M. Roth, *Compliance*, 2018. [Online]. Available: `https://aws.amazon.com/compliance/shared-responsibility-model/` (Accessed: 22.9.2020).

[9] D. Merron, *What is infrastructure as code? iac explained*, Dec. 2018. [Online]. Available: `https://www.bmc.com/blogs/infrastructure-as-code/` (Accessed: 13.7.2020).

[10] "Virtualization in cloud computing", *Journal of Information Technology & Software Engineering*, vol. 04, no. 02, 2014. DOI: `10.4172/2165-7866.1000136`.

[11] *Storage virtualization*. [Online]. Available: `https://www.enterprisestorageforum.com/storage-hardware/storage-virtualization.html` (Accessed: 13.7.2020).

[12] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review", *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, Jan. 2019. DOI: `10.1109/tcc.2017.2702586`.

[13] *Updating and modernizing: Moving from virtual machines to containers*, Aug. 2019. [Online]. Available: `https://caylent.com/moving-from-virtual-machines-to-containers` (Accessed: 13.7.2020).

[14] *Docker inc.* [Online]. Available: `https://web.archive.org/web/20140702231323/https://www.dotcloud.com/about.html` (Accessed: 13.7.2020).

[15] *Docker vs virtual machine - understanding the differences*, Sep. 2019. [Online]. Available: `https://geekflare.com/docker-vs-virtual-machine/` (Accessed: 13.7.2020).

[16] *Shifting docker security left*, Jul. 2019. [Online]. Available: `https://snyk.io/blog/shifting-docker-security-left/` (Accessed: 13.7.2020).

[17] A. Gerrard, *What is kubernetes? an introduction to the container orchestration platform*, Aug. 2019. [Online]. Available: `https://blog.newrelic.com/engineering/what-is-kubernetes/` (Accessed: 13.7.2020).

[18] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and et al., "Serverless computing: Current trends and open problems", *Research Advances in Cloud Computing*, pp. 1–20, 2017. DOI: `$10.1007/978-981-10-5026-8_1$`.

[19] G. Adzic and R. Chatley, "Serverless computing: Economic and architectural impact", *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, 2017. DOI: `10.1145/3106237.3117767`.

[20] A. Wiggins, *The twelve-factor app*. [Online]. Available: `https://12factor.net/` (Accessed: 13.7.2020).

[21] *Owasp top 10 (2017): Interpretation for serverless*. [Online]. Available: `https://owasp.org/www-pdf-archive/OWASP-Top-10-Serverless-Interpretation-en.pdf` (Accessed: 13.7.2020).

[22] M. Stoica, M. Mircea, and B. Ghilic-Micu, "Software development: Agile vs. traditional", *Informatica Economica*, vol. 17, no. 4/2013, pp. 64–76, 2013. DOI: `10.12948/issn14531305/17.4.2013.06`.

[23] *Waterfall model: What is it and when should you use it?*, Nov. 2017. [Online]. Available: `https://airbrake.io/blog/sdlc/waterfall-model` (Accessed: 29.9.2020).

[24] M. Says, V. S. says, U. Says, D. D. A. Says, J. C. Says, M. A. says, S. R. Says, S. says, S. R. B. Says, C. M. Says, and et al., *Home*. [Online]. Available: `http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/` (Accessed: 29.9.2020).

[25] W. Lynch, *The brief of history of scrum*, Jan. 2019. [Online]. Available: `https://medium.com/@warren2lynch/the-brief-of-history-of-scrum-15efb73b4701` (Accessed: 22.9.2020).

[26] Atlassian, *Kanban - a brief introduction*. [Online]. Available: `https://www.atlassian.com/agile/kanban` (Accessed: 13.7.2020).

[27] G. Inc, *Devops*. [Online]. Available: `https://www.gartner.com/en/information-technology/glossary/devops` (Accessed: 13.7.2020).

[28] J. Vehent, *Securing DevOps: security in the cloud*. Manning Publications Co., 2018.

[29] L. Mukherjee, *Devsecops: A definition, explanation exploration of devops security*, Apr. 2020. [Online]. Available: `https://sectigostore.com/blog/devsecops-a-definition-explanation-exploration-of-devops-security/` (Accessed: 23.9.2020).

[30] Atlassian, *Devsecops: Injecting security into cd pipelines*. [Online]. Available: `https://www.atlassian.com/continuous-delivery/principles/devsecops` (Accessed: 25.9.2020).

[31] *What is continuous security monitoring?*, Sep. 2018. [Online]. Available: `https://digitalguardian.com/blog/what-continuous-security-monitoring` (Accessed: 23.9.2020).

[32] C. P. Software, *What is shift left security?*, Aug. 2020. [Online]. Available: `https://www.checkpoint.com/cyber-hub/cloud-security/what-is-shift-left-security/` (Accessed: 23.9.2020).

[33] D. Merron, *What is a pipeline in software engineering? intro to deployment, ci, & cd pipelines*, Nov. 2018. [Online]. Available: `https://www.bmc.com/blogs/deployment-pipeline/` (Accessed: 13.7.2020).

[34] G. GuimarãesCo-founder, *What is a linter and why your team should use it?*, Apr. 2020. [Online]. Available: `https://sourcelevel.io/blog/what-is-a-linter-and-why-your-team-should-use-it`.

[35] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[36] Templateninja. [Online]. Available: `https://websec.io/2013/08/27/Core-Concepts-Trust-Boundaries.html` (Accessed: 29.9.2020).

[37] M. admin, *4 risk response strategies for project management success* |, Feb. 2020. [Online]. Available: `https://mrmcentral.com/4-risk-response-strategies-for-project-management-success/` (Accessed: 24.9.2020).

[38] *What is spoofing?*, Feb. 2020. [Online]. Available: `https://www.forcepoint.com/it/cyber-edu/spoofing` (Accessed: 29.9.2020).

[39] *Data tampering in the world today*. [Online]. Available: `https://www.ukessays.com/essays/information-technology/data-tampering-in-the-world-today.php` (Accessed: 29.9.2020).

[40] *Repudiation attack*. [Online]. Available: `https://owasp.org/www-community/attacks/Repudiation_Attack` (Accessed: 29.9.2020).

[41] N. S. Team, *Information disclosure issues and attacks in web applications*, Jun. 2019. [Online]. Available: `https://www.netsparker.com/blog/web-security/information-disclosure-issues-attacks/` (Accessed: 29.9.2020).

[42] *Article: What is... denial-of-service (dos): F-secure*. [Online]. Available: https://www.f-secure.com/v-descs/articles/denial-of-service.shtml (Accessed: 29.9.2020).

[43] J. Melnick, *What is privilege escalation?*, Mar. 2020. [Online]. Available: https://blog.netwrix.com/2018/09/05/what-is-privilege-escalation/ (Accessed: 29.9.2020).

[44] *Rapid risk assessment*. [Online]. Available: https://infosec.mozilla.org/guidelines/risk/rapid_risk_assessment (Accessed: 13.7.2020).

[45] *Sast vs. dast: What's the difference?: Synopsys*, Aug. 2019. [Online]. Available: https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference/ (Accessed: 13.7.2020).

[46] G. Stoneburner, C. Hayden, and A. Feringa, "Engineering principles for information technology security (a baseline for achieving security), revision a", 2004. DOI: 10.6028/nist.sp.800-27ra.

[47] *2019 global developer report: Devsecops finds security roadblocks divide teams*. [Online]. Available: https://about.gitlab.com/blog/2019/07/15/global-developer-report/ (Accessed: 13.7.2020).

[48] *Security awareness training - what it is and why it's critical*, Sep. 2020. [Online]. Available: https://www.mediapro.com/security-awareness-training/ (Accessed: 29.9.2020).

[49] *What is the system log (syslog)? - definition from techopedia*. [Online]. Available: https://www.techopedia.com/definition/1858/system-log-syslog (Accessed: 29.9.2020).

[50] *What are message brokers?* [Online]. Available: https://www.ibm.com/cloud/learn/message-brokers (Accessed: 13.7.2020).

[51] J. Kernel, *What is log analysis, why you need it, tools, practices and examples*, Jul. 2020. [Online]. Available: `https://www.xplg.com/what-is-log-analysis-and-why-do-you-need-it/` (Accessed: 29.9.2020).

[52] *Strengthening cybersecurity with log forensic analysis*. [Online]. Available: `https://www.graylog.org/post/strengthening-cybersecurity-with-log-forensic-analysis` (Accessed: 24.9.2020).

[53] B. Adair, *Using infrastructure as code as a poor man's dr*, Dec. 2018. [Online]. Available: `https://www.awsadvent.com/2018/12/09/using-infrastructure-as-code-as-a-poor-mans-dr/` (Accessed: 24.9.2020).