MONDRAGON UNIBERTSITATEA

24TH JANUARY 2020

POPBL-1
ADVANCED ATTACK DETECTION IN A MANUFACTURER BUSINESS

MONDRAGON UNIBERTSITATEA

ANDER BOLUMBURU, IKER OCIO, HARITZ SAIZ

1. MASTER'S DEGREE IN DATA ANALYSIS, CYBERSECURITY AND CLOUD COMPUTING

ABSTRACT

This document describes the first POPBL project developed at the University of Mondragon in mid-2020 in the Data Analysis, Cybersecurity and Cloud Computing master's degree. This project aims to combine a data analytics, cybersecurity and cloud computing branches in one big project. The general objective is going to be a cloud infrastructure with some trusted microservices which simulate a manufacturing environment with an API for http requests, some honeypots for attack information gathering and a machine learning/data analytics machine for business intelligence and information visualization. This information is going to be used for attack anomaly detection using machine learning techniques, such us clustering or classification algorithms types.

RESUMEN

Este documento describe el primer proyecto POPBL de la universidad de Mondragon para el máster en Análisis de datos, Ciberseguridad y Cloud Computing. Este proyecto pretende combinar las ramas del análisis de datos, ciberseguridad y computación en la nube un gran proyecto común. El objetivo general es crear una infraestructura cloud legitima con microservicios que simulan un entorno de fabricación industrial y una API Gateway para permitir peticiones http a los microservicios, la instalación de honeypots para la detección y recogida de información de ataques y máquinas enfocadas a la visualización de datos y machine learning. Esta información será utilizada a posterior para la detección de ataques por anomalías utilizando técnicas de clustering o clasificación.

LABURPENA

Dokumentu honetan Mondragon Unibertsitateko Datu analisia, zibersegurtasuna eta hodei konputazioko masterreko POPBL proiektua deskribatzen da. Lan honen xedea datu analisia, zibersegurtasuna eta hodei konputazioko zenbait kontzepturen konbinazioan datza. Helburu nagusiari dagokionez, zenbait mikrozerbitzuek osatutako cloud azpiegitura bat sortu da. Bertan, mikrozerbitzuen bidez fabrikrazio ingurune bat simulatzea lortu da. Bertako trafikoarekin, bisualizazio tresna batzuk garatu dira eta beste aldetik, adimen artifizialen bidez ezagutza erauzi da.

RESUM

Aquest document descriu el primer projecte POPBL de la universitat de Mondragon per el màster en anàlisis de dades, cibersecuritat e informàtica en núvol. Aquest projecte pretén combinar les diferents matèries en un projecte comú. El objectiu general es crear una infraestructura cloud legítima amb microserveis d'un entorn d'una fàbrica industrial i una API Gateway per permetre peticions hhtp al microserveis, la instal·lació de honeypots per a la detecció y recollida d'atacs i maquines enfocades a la visualització de dades y Machine learning. Aquesta informació serà utilitzada a posterior per la detecció d'atacs per anomalies utilitzant tècniques de clustering o classificació.

Table of contents

1. Introduction 1
2. Problem Description 2
3. Proposed Solution 3
3.1. Task plan 4
4. Development 5
4.1. Architecture and automatization 5
4.1.1. Service automatization – Docker 6
4.1.2. Subnets 6
4.2. Microservices 12
4.2.1. Proposed Solution 13
4.2.1. New microservice 14
4.2.2. Saga 15
4.2.3. Messaging 20
4.2.4. API Gateway and Service Discovery 21
4.2.5. Logging 21
4.2.6. Security 22
4.3. Data collection/Data gathering 22
4.3.1. Used software 23
4.3.2. Honeypots 25
4.4. Cloud security/Services 28
4.4.1. Secrets manager tool. Vault 29
4.4.4. SSL/TLS Certs 31
4.4.6. Backup Policies 32
4.5. Data Analytics 32
4.5.1. Introduction to Data Analytics scenario 33
4.5.2. Theorical framework for model validation 33
4.5.3. Data processing 35
4.5.4. Theorical framework in practice 37
4.5.5. Data visualization 37
4.5.6. Interaction 39

4.6. Kibana dashboard 41
4.6.1. Attacks map 42
4.6.2. CPU Usage 43
4.6.3. RAM Memory usage 44
4.6.4. Event duration average 45
4.6.5. Most used usernames and passwords 46
5. Conclusions 48
5.1. Infrastructure as Code 48
5.2. Security 48
5.3. Microservices 48
5.4. Data analytics 49
6. Future lines 50
7. Annexes 51
8. Bibliography 52

ILLUSTRATION TABLE

Figure 1 - Amazon Data Center 2
Figure 2 - Cloudformation architecture 5
Figure 3 - NAT Host USERDATA 9
Figure 4 - Microservices basic architecture 13
Figure 5 - Microservices new architecture 14
Figure 6 - Create Order Saga with abort state 16
Figure 7 - Cancel order Saga 16
Figure 8 - Order creation 17
Figure 9 - Order finish check 18
Figure 10 - Order not cancellable example 18
Figure 11 - Order creation example 19
Figure 12 - Order cancellation request 19
Figure 13 - Manual cancel 20
Figure 14 - Messaging architecture 20
Figure 15 - Service Discovery architecture 21
Figure 16 - Kibana Dashboard Example 24
Figure 17 - Data collection architecture 25
Figure 18 - Vault with Elastic credentials 30
Figure 19 - Certificate 32
Figure 20 - Packetbeat controller dashboard 38
Figure 21 - Visual Data Exploration diagram 40
Figure 22 - Kibana Dashboard 42
Figure 23 - Attack location map 42
Figure 24 - CPU Usage gauge 43
Figure 25 - CPU Usage time series 43
Figure 26 - RAM Memory usage gauge 44
Figure 27 - RAM Memory usage time series 44
Figure 28 - RAM Usage visualization 45
Figure 29 - Event duration visualization 46
Figure 30 - Most used Username/Password visualization 46
Figure 31 - Most used Username/Password visualization (with more data) 47

1. Introduction

The aim of this document is to describe how a manufacturing factory manages their production using an microservice based application as well as securing its deployment and communication from possible unknown attacks.

In order to solve the problem, they are going to deploy a cloud infrastructure with some honeypots and microservices for gathering information about the attacks in order to protect the normal flow of the network and resources.

The project consists on:

- Honeypots to obtain information about the attackers, as well as their attempts to penetrate the systems.
- Monitoring and supervising information related to the behaviour of the systems that works in cloud deployed infrastructure. Visual representation of these attacks and justification of the taken measures to prevent them.
- The use of microservices to identify legitimate or non-legitimate traffic has been recorded.

The source code developed for this project is attached within the provided deliverable.

2. Problem Description

A manufacturing company is receiving different attacks on its web platform composed of different microservices. The origin of the attacks is unknown and normal user traffic is quite high.

It would be good to get more information about the attackers in order to defend the web infrastructure from them.

In parallel, a new microservice is going to be added in the production environment and the DevOps responsible must be able to deploy the infrastructure on the new Amazon Web Services cloud servers to take in advantage the most AWS resources that Amazon offers. (See Figure 1 - Amazon Data Center)

Figure 1 - Amazon Data Center

3. Proposed Solution

The cybersecurity expert has recommended placing some honeypot to collect more information about the attacks, discover as much as possible about them and try to establish solutions so that the maximum of legitimate users is not affected by what happened.

With this information, an expert analyst in machine learning has been temporarily hired to make the most of all the information collected in the last month and try to identify the key points used by the attackers every time they try to penetrate our network. In addition, we would like to have a sufficiently descriptive dashboard so that a security expert can visualize the state of the entire infrastructure in the future and quickly identify anomalies in the hosts as it can be seen in 4.6 section.

As the company's infrastructure has been migrated to Amazon Cloud Services, the resources it offers have been leveraged to put different layers of protection between the hosts, separating public and private areas in order to mitigate the attack surface of potential attackers.

3.1. Objectives

The objectives to be completed in this project are

- The implementation of a saga that may conflict with the existing one in order to provoke race conditions and avoid them.
- The deployment of honeypots in order to obtain information about typical attacks on a network.
- The deployment of microservices (not production) to obtain information about unclassified traffic.
- The analysis of data collected from heterogeneous sources in order to obtain intelligence about cyberattacks.
- Development of different interfaces that facilitate obtaining knowledge about the data using visual variables. Allow the user to manipulate different parameters to improve the knowledge.
- Ensure the integrity, confidentiality and availability of deployed resources.
- Automate the deployment of the cloud services and resources that take part in the infrastructure

3.2. Task plan

The following plan has been developed at the beginning of the project in order to achieve the objectives. (See Figure 2 - Task plan). (ANNEX A: Gantt Diagram)

Subject	Tareas/Fecha	16-dic	17-dic	18-dic	19-dic	20-dic	21-dic	22-dic	23-dic	24-dic	25-dic	26-dic	27-dic
VD	Elegir representaciones a mostrar en el dashboard y porque												
AA	Definir que datos y como recoger desde el minuto 1 para el analisis posterior												
AA	Definir modelo de datos para prediccion de anomalias												
SIR	Definir posibles ataques a monitorizar y que lo caracteriza												
SIR	Definir Nº de honeypots, vpcs, subredes...												
PI/SIR	Identificar mecanismos de seguridad AWS												
PI	Estructura basica de la arquitectura												
AAS	Definir microservicos y funcionalidades nuevas a desarrollar												
PI	Crear una infraestructura básica automatizada con Cloudformation												
PI/SIR	Definir ACL y Security Groups												
PI	Automatización Cloudformation												
PI/SIR	Desplegar Honeypot Cowrie												
PI/SIR	Desplegar Honeypot Dioneaea												
AAS	Desarrollo nuevo microservicio												
AAS	Desarrollo nueva saga												
PI/SIR	Conectar Honeypots con Elastic												
VD	Despliegue Elastic + Kibana												
VD	Desarrollo Plotly												
PI/SIR	Contenedorizar aplicaciones												
PI/SIR	Despliegue de Vault												
FAA	Validacion del modelo												
AA	Definicion marco teorico-practico para validar modelo no supervisado												
AA	Preproceso de datos recogidos en Elastic												
AA	Estudio de fuentes de datos												
AA	Conclusiones del analisis de datos de los resultados obtenidos												
VD	Justificacion de la solucion visual dada en Kibana												
	Redacción memoria												
	Diapositivas presentacion												

Figure 2 - Task plan

4. Development

This section will describe and detail the development of microservices and services that support the correct functioning of the infrastructure and the detection of anomalies. A moderate cost plan of the infrastructure has been done at time of developing this project. (ANNEX B: AWS Pricing)

4.1. Architecture and automatization

The cloud infrastructure has been deployed in four different subnets. Three of them are public subnets with public IP address in their hosts and the last one is a private subnet without any public IP in their hosts.

This network segmentation (See Figure 3 - CloudFormation architecture) has been designed with security in mind. This way there are some critical hosts with client's information that cannot be accessed by any attacker.

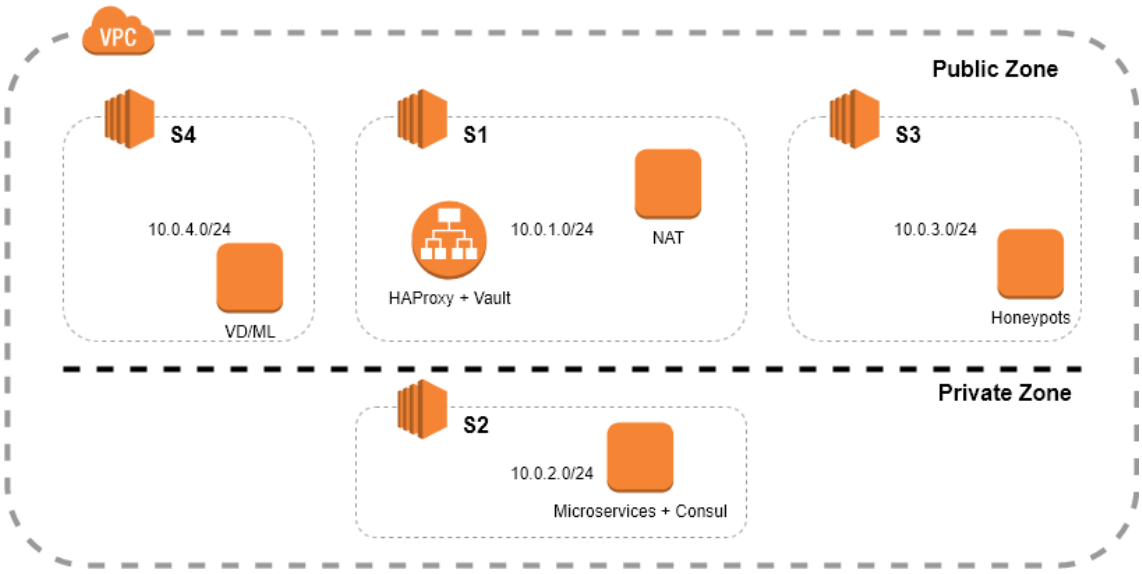


Figure 3 - CloudFormation architecture

There are 5 hosts in total with several dockers on each of them simulating a bigger infrastructure that in reality is. This has been done to improve the easy of deployment using Docker Compose. The following section will detail each subnet topology

This architecture has been automated using CloudFormation script to replicate as many times as wished. The microservices + HAProxy + Consul deployment is totally automated. On the other hand, honeypots + Elastic/Kibana need some manual configuration after the script has been executed such us changing Metricbeats/Packetbeats IP addresses to send the information to Elastic Server.

Each service running on each machine is contained so that it can be automatically deployed from commands in the USER DATA. The containerization technology used is Docker containers.

4.1.1. Service automatization – Docker

There are four different main docker-compose files.

The first one has all microservices + Consul + HAProxy.

- The second one has Vault.
- The third one has Elastic + Kibana.
- The fourth one has Cowrie and Dionaea honeypots.

There are used on different hosts depending on the service that needs to be deploy on the start-up step.

All of them are on our personal Git account, so it can be downloaded from there. (ANNEX C: Source Code)

4.1.2. Subnets

There are 4 subnets named S1, S2, S3 and S4. Each subnet has its own role in the deployment that are going to be explained following.

4.1.2.1. S1 - Bastion subnet

This subnet centralizes http traffic directed to HAPROXY. The aim of this subnet is to provide a transparent entry point to the whole microservice infrastructure as well as hiding its structure as it may provide critical information to potential attackers.

All hosts inside this area have its own public IP address. There is one host with two docker containers:

- HAProxy. The LB/API Gateway to redirect traffic to each microservice.
- Vault. The secrets manager which has all related about tokens, passwords, secrets.

All those hosts have their own Security Group for inbound and outbound traffic. The rules are as follows:

Inbound

Protocol	*Port Range*	*Source*	*Description*
TCP	80	0.0.0.0/0	Http port
TCP	8200	0.0.0.0/0	Vault

TCP	14000	0.0.0.0/0	HAProxy
TCP	22	0.0.0.0/0	SSH Port
TCP	14001	0.0.0.0/0	HAProxy Stats
TCP	443	0.0.0.0/0	Https

Outbound

Protocol	*Port Range*	*Destination*	*Description*
TCP	22	10.0.2.0/24	Microservices SSH
TCP	80	0.0.0.0/0	Yum Http
TCP	443	0.0.0.0/0	Yum https
UDP	8600	10.0.2.0/24	Consul IP request
TCP	1024-65535	0.0.0.0/0	Default outbound

The main idea behind Security Groups, is to have well defined host port security. This resource is used to open or close needed ports of each machine. In this case, HAProxy needs to connect with yum resources in the deployment step to install docker and other software, it is also needs to ask to Consul Service Discovery about microservices IP addresses, regular SSH connections with microservice's subnet and so on.

Vault has been installed on this host for didactic purposes but in a real scenario, it should be better to have it installed in another, more protected, subnet to avoid internet connection requests.

The other host is a NAT server to provide internet connectivity to the microservices on S2 subnet.

In addition, the NAT server has defined IP Tables rules to redirect the traffic to two ports of the private subnet in order to display the Consul and RabbitMQ management interface. These rules are as follows:

sudo iptables -t nat -A PREROUTING -p tcp --dport 8500 -j DNAT --to-destination 10.0.2.10:8500

sudo iptables -t nat -A PREROUTING -p tcp --dport 15671 -j DNAT --to-destination 10.0.2.10:15671

This host has its own IP route defined to provide internet connection to microservices.

Subnet ACLs are used to control the traffic flow between subnets and Internet.

Having well-defined rules in the ACLs to regulate traffic increases the level of safety but requires a very high level of maintenance. For every new host added or removed, these rules should be changed. To facilitate ACL maintenance, more generic

rules are used, which regulate the flow of traffic between subnets, leaving the task of protecting the host to the Security Groups.

Rules are evaluated starting with the lowest numbered rule. As soon as a rule is matched, it's applied regardless of any higher-numbered rule that might contradict it. It is important to have into account. Many firewalls also work like this.

ACL Inbound

Rule	*Protocol*	*Port Range*	*Source*	*Allow/Deny*
100	TCP	0-65535	0.0.0.0/0	ALLOW
200	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

ACL Outbound

Rule	*Protocol*	*Port Range*	*Destination*	*Allow/Deny*
100	TCP	0-65535	0.0.0.0/0	ALLOW
200	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

4.1.2.2. S2 - Microservices subnet

This subnet has the microservices deployed along with Consul's Service Discovery. This subnet is private, so it is not directly accessible from the Internet and uses the NAT server installed on the public S1 subnet to have internet connectivity.

In addition, the host where the microservices are located, must always be deployed once the NAT server is up and running. Otherwise, the USER DATA script (See Figure 4 - NAT Host USERDATA) will not work properly when it comes to downloading the necessary resources from the Internet. This problem is solved by the Depends On instruction in the CloudFormation script, as shown in the image below.

```
  UserData:
    Fn::Base64: !Sub |
      #!/bin/bash
      exec > >(tee /var/log/user-data.log|logger -t user-data ) 2>&1
      sysctl kernel.hostname=Microservices
      sudo yum install -y git
      cd /home/ec2-user
      git clone https://gitlab.danz.eus/haritz.saiz/pbl1
      sudo chmod 400 pbl1/POPBL1.pem
      sudo yum update -y
      sudo yum install -y docker
      sudo service docker start
      sudo usermod -a -G docker ec2-user
      sudo curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(un
      sudo chmod +x /usr/local/bin/docker-compose
      sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
      cd pbl1/devops/microservicios
      sudo docker-compose up -d store auth delivery machine order log payment rabitmq consul
      sleep 120
      sudo docker-compose up -d store auth delivery machine order log payment rabitmq consul
DependsOn: NatServer
```

Figure 4 - NAT Host USERDATA

Microservice's host has lot of inbound rules in the Security Group because there are several Dockers running inside with microservices, Consul and RabbitMQ.

Outbound ephemeral ports are opened (from 1024 to 65535) because all services need to response the requests.

Inbound

Protocol	*Port Range*	*Source*	*Description*
TCP	13002	0.0.0.0/0	Payment Micros.
TCP	13003	0.0.0.0/0	Delivery Micros.
TCP	13001	0.0.0.0/0	Auth Micros.
TCP	4369	0.0.0.0/0	RabbitMQ
UDP	8600	0.0.0.0/0	Consul Port 5
TCP	13004	0.0.0.0/0	Order Micros.
UDP	8300-8301	0.0.0.0/0	Consul Port 3
TCP	13010	0.0.0.0/0	Store Micros.
TCP	5671	0.0.0.0/0	RabbitMQ server
TCP	8300-8301	0.0.0.0/0	Consul Port 1
TCP	25672	0.0.0.0/0	RabbitMQ Port
UDP	8302	0.0.0.0/0	Consul Port 7
TCP	22	0.0.0.0/0	SSH
TCP	13005	0.0.0.0/0	Machine Micros.
TCP	15671	0.0.0.0/0	RabbitMQ Manage.
TCP	8302	0.0.0.0/0	Consul Port 6
TCP	13006	0.0.0.0/0	Log Micros.
TCP	8600	0.0.0.0/0	Consul Port 4
TCP	8500-8501	0.0.0.0/0	Consul Manage.

Outbound

Protocol	*Port Range*	*Destination*	*Description*

TCP	80	0.0.0.0/0	Http
TCP	443	0.0.0.0/0	Https
TCP	1024-65535	0.0.0.0/0	Ephemeral

The ACL rules on this subnet have nothing remarkable about them. The subnet itself is protected by the NAT server from possible attacks from the Internet.

ACL Inbound

Rule	*Protocol*	*Port Range*	*Source*	*Allow/Deny*
100	TCP	0-65535	0.0.0.0/0	ALLOW
200	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

ACL Outbound

Rule	*Protocol*	*Port Range*	*Destination*	*Allow/Deny*
100	TCP	0-65535	0.0.0.0/0	ALLOW
200	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

4.1.2.3. S3 - Honeypots subnet

In this subnet, honeypot hosts are going to be installed. A separated area from normal manufacturing microservices has been created in order to create a more controlled area for contingency purposes.

This hosts are prepared to be attacked by any attacker from internet and to recollect information about them. All interaction with this host is going to be considered an attack. These interactions are going to be send to the Elastic host in S4 subnet.

The Security Group installed in this host is a default one without any port protection. If there is no service listening on that port, it will be because the honeypot is exposing some simulating services.

Inbound

Protocol	*Port Range*	*Source*	*Description*
ALL	0-65535	0.0.0.0/0	ALL

Outbound

Protocol	*Port Range*	*Destination*	*Description*
ALL	0-65535	0.0.0.0/0	ALL

A contingency wall should be established between this sub-network and the rest of the infrastructure in order to mitigate possible horizontal movements in case an attacker can penetrate the honeypot. Therefore, ACL rules have been defined so there is no case that an attacker can move between the 10.0.3.0/24 subnet and the others.

Sended information to the S4 subnet must be allowed since that is where the information of the honeypot attacks will be sent, so traffic to that network will not be denied.

ACL Inbound

Rule	*Protocol*	*Port Range*	*Source*	*Allow/Deny*
100	TCP	0-65535	10.0.1.0/24	DENY
101	UDP	0-65535	10.0.1.0/24	DENY
200	TCP	0-65535	10.0.2.0/24	DENY
201	UDP	0-65535	10.0.2.0/24	DENY
400	TCP	0-65535	0.0.0.0/0	ALLOW
401	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

ACL Outbound

Rule	*Protocol*	*Port Range*	*Destination*	*Allow/Deny*
100	TCP	0-65535	10.0.1.0/24	DENY
101	UDP	0-65535	10.0.1.0/24	DENY
200	TCP	0-65535	10.0.2.0/24	DENY
201	UDP	0-65535	10.0.2.0/24	DENY
400	TCP	0-65535	0.0.0.0/0	ALLOW
401	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

4.1.2.4. S4 - ML-Elastic-Kibana visualization subnet

This subnet focused on data flows sent by Elastic Beats from different points of the infrastructure. The data is also pre-processed and stored here and then, displayed

in Kibana. Since traffic from the subnet where the Honeypot is located will be allowed, special care must be taken to filter the traffic from that network.

Security Groups used on this host are next ones:

Inbound

Protocol	*Port Range*	*Source*	*Description*
TCP	22	0.0.0.0/0	SSH
TCP	5044	0.0.0.0/0	Logstash API port
TCP	9600	0.0.0.0/0	Logstash Elastic
TCP	9300	0.0.0.0/0	Elastic 2 port
TCP	443	0.0.0.0/0	Kibana https
TCP	9200	0.0.0.0/0	Elastic 1 port

Outbound

Protocol	*Port Range*	*Destination*	*Description*
TCP	80	0.0.0.0/0	Http
TCP	443	0.0.0.0/0	Https
TCP	1024-65535	0.0.0.0/0	Ephemeral

The ACL rules of this subnet are basically the same as those of S1 since it is needed an Internet connection to view the Kibana management windows.

ACL Inbound

Rule	*Protocol*	*Port Range*	*Source*	*Allow/Deny*
100	TCP	0-65535	0.0.0.0/0	ALLOW
200	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

ACL Outbound

Rule	*Protocol*	*Port Range*	*Destination*	*Allow/Deny*
100	TCP	0-65535	0.0.0.0/0	ALLOW
200	UDP	0-65535	0.0.0.0/0	ALLOW
*	ALL	ALL	0.0.0.0/0	DENY

4.2. Microservices

The main objective of the microservices deployed at AWS is to carry out the task of manufacturing some pieces. In this case, the customer has asked us to add some extra functionalities to the previous system.

Therefore, we have started from scratch and added new functionalities such as the fact that the parts can now be accumulated in a warehouse or that the order placed by the customer can be cancelled. Finally, a customer may order a set of pieces of certain type: A, B or C.

In order to add these new functionalities, the proposed solution is explained in the next section. You can see a better description of these microservices. (ANNEX D & E: Microservice development)

4.2.1. Proposed Solution

This was the architecture (See Figure 5 - Microservices basic architecture) that our "basic" microservices presented before adding the new functionalities.

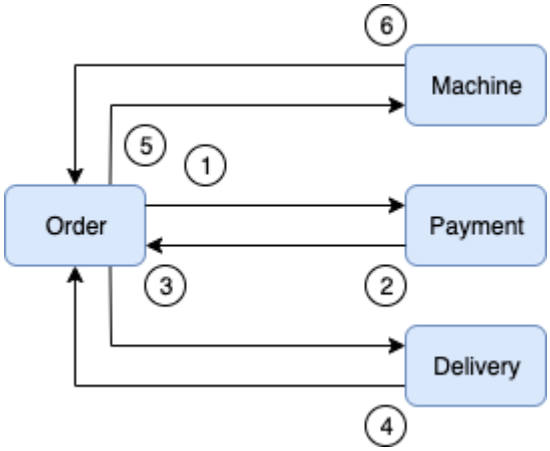


Figure 5 - Microservices basic architecture

The basic operation of these services is as follows:

First the customer initiates an order. Once this order is done, the system checks if the client has the necessary money to carry out the operation. If not, the operation is rejected, and the customer is notified of the rejection.

If the operation can go ahead, it is checked that the shipping address is correct, if it is, the machine is sent to make the parts corresponding to that order. If not, the operation is rejected and clearing operations are made, such as the customer's money being returned.

Once everything is checked, the operation is finished.

As we were told, the system should incorporate new functionalities such as the possibility that the order can be cancelled or the fact that there are different types that can be stored in a warehouse.

In order to the architecture fulfil with the new functionalities, both a new saga and a new service have been added.

4.2.2. New microservice

The main functionality of this microservice is to save different type of pieces on warehouse.

In addition to this, a system has also been implemented to ensure that there are always x units of pieces in the warehouse (By default, the minimum stock is set to 5 pieces per type). This way, the first orders are dispatched faster, as they do not have to wait for the parts to be manufactured by the machine.

Once the new microservice (See Figure 6 - Microservices new architecture) is implemented and deployed the topology is as follows:

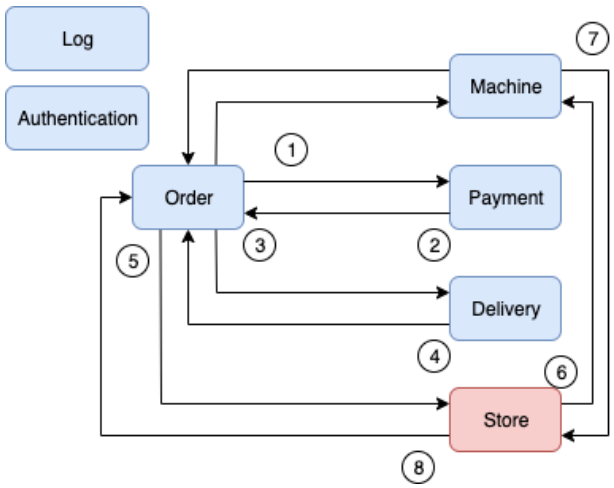


Figure 6 - Microservices new architecture

The operation with the new microservice alters the overall process of the system.

Now, after creating the order, the corresponding microservices checks that the customer has enough money to pay for the order and that the destination is allowed. If a valid destination code is provided, the warehouse checks if there are available pieces.

If so, the pieces in the warehouse are assigned to that particular order and the process is completed. If not, the number of pieces required to carry out the order is sent to the machine to be produced.

Once the order has been carried out and there are not more orders to be processed, and if the stock is low, the system will ensure that the necessary pieces are produced. So when a new order enters the warehouse, at least part of it, is available and the order will not take so long to be processed.

On the other hand, the possibility of cancelling the order comes into play, which has repercussions on the previously developed saga, which will be explained in the next section.

4.2.3. Saga

The new developed saga will orchestrate the cancellation process. First, when a new cancellation request is dispatched, the new saga is initialized to a cancellation request state. Then the saga will look at the order state. If the order is on a cancellable state, then the order is updated setting a cancel_request flag to True. This flag is the core component to resolve the problem occasioned when two sagas are being orchestrated at the same time and both are working with the same object. The flag will abort the order creation saga. This will be done because the order creation saga states, before switching to another state, will look to that flag. If said flag is set to True, then the saga aborts, if not the saga switches to another valid state. The order creation saga, if aborted, will not oversee refunding the money to the client (or any countermeasure operations). That will be managed by the cancel request saga.

The following two states diagrams represent all possible stats for each saga. The first one (See Figure 7 - Create Order Saga with abort state) corresponds to the modified saga: Create Order. The second state machine diagram (See Figure 8 - Cancel order Saga) corresponds to the newly developed saga: Cancel Order

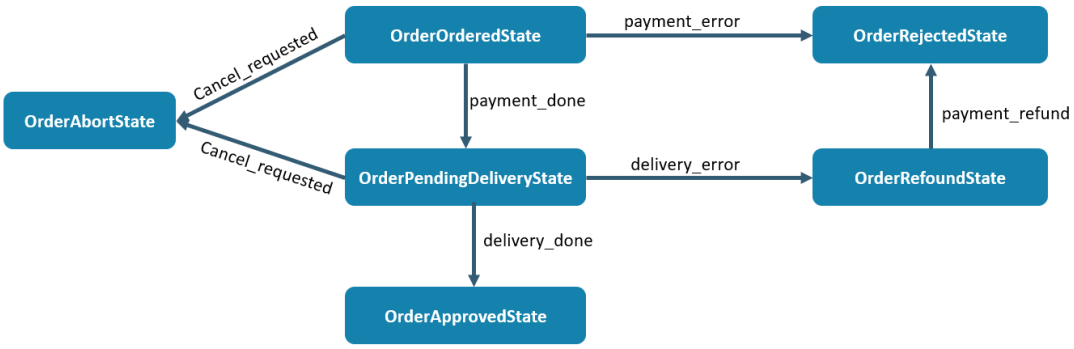


Figure 7 - Create Order Saga with abort state

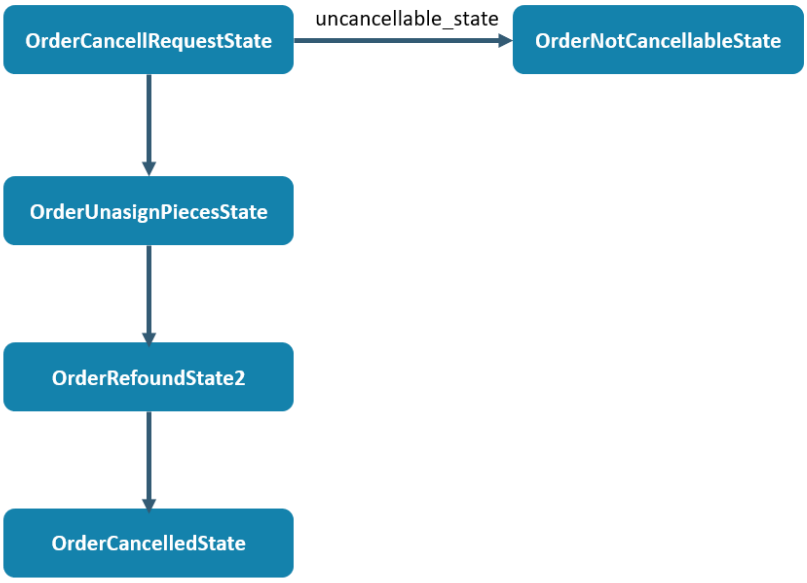


Figure 8 - Cancel order Saga

The other core aspect to handle the conflict, is the state of said order. As we told previously, in order to cancel an order, it must be in a cancellable state. Let's analyse each possible order states:

STATUS_INIT	*Order is initialized*
STATUS_PAYED	Order has been paid
STATUS_APPROVED	Order has a valid delivery code
STATUS_STORED	Order is queued in the store
STATUS_ACTIVE	Order is being manufactured. Is the first one in the store queue
STATUS_CREATED	Order has all pieces ready and assigned
STATUS_DELIVERED	Order has been delivered
STATUS_FINISHED	Order acknowledges the delivery

STATUS_CANCELED_BAD_ZIP_CODE	Order rejected because the client provided a bad ZIP code
STATUS_CANCELED_NO_MONEY	Order rejected because the client had not enough money
STATUS_CANCELED_MANUAL	Order rejected because the client manually cancelled the order

Not all these states are cancellable. Just some of them. The cancellable states are: STATUS_INIT, STATUS_APPROVED, STATUS_PAYED, STATUS_ACTIVE

In general, we could say that an order is cancellable before it leaves the store. Once it leaves the store, the order is no longer cancellable.

Let's look at some examples. First, we will create an order composed of 5 pieces of type A but we will not cancel it until we are certain that the order has finished.

Let's create the order. (See Figure 9 - Order creation)

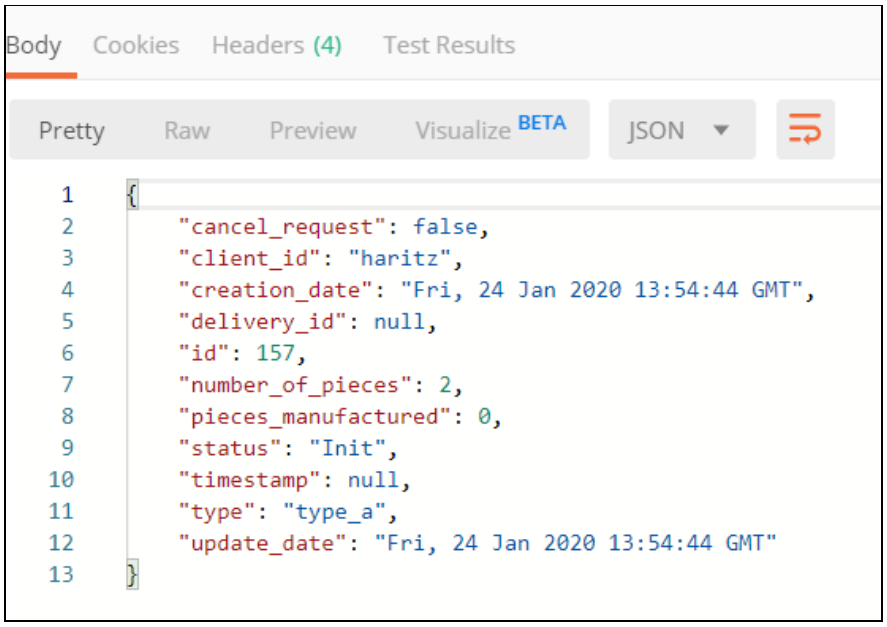


Figure 9 - Order creation

Next, check that the order has finished. (See Figure 10 - Order finish check)

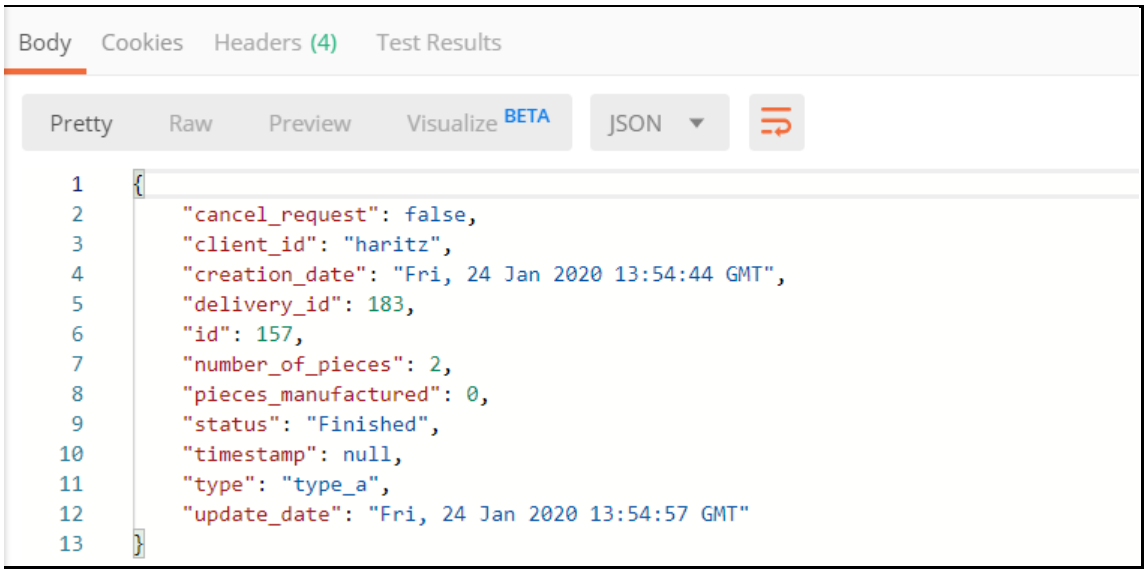


Figure 10 - Order finish check

Finally try cancelling the order. (See Figure 11 - Order not cancellable example)

Figure 11 - Order not cancellable example

Now let’s create and order and immediately after, before it is produced, let's cancel said order. (See Figure 12 - Order creation example)

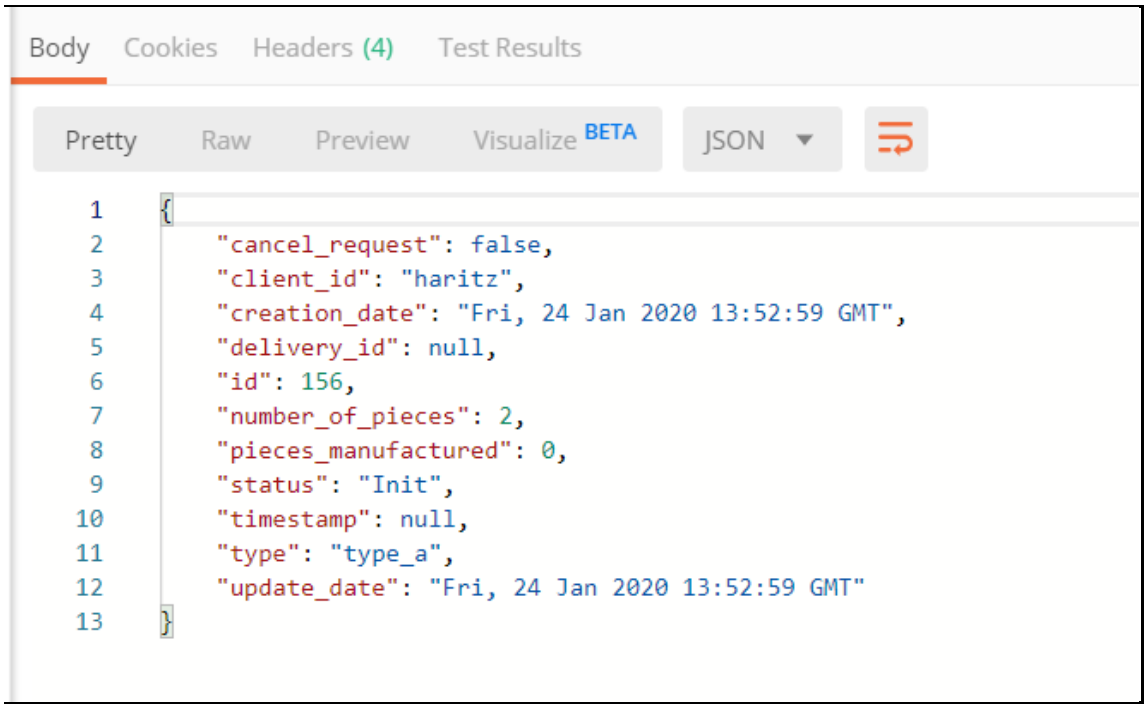


Figure 12 - Order creation example

Now we try to cancel the order. (See Figure 13 - Order cancellation request)

Figure 13 - Order cancellation request

If we check the order status, we can get confirmation that the order was cancelled manually. (See Figure 14 - Manual cancel)

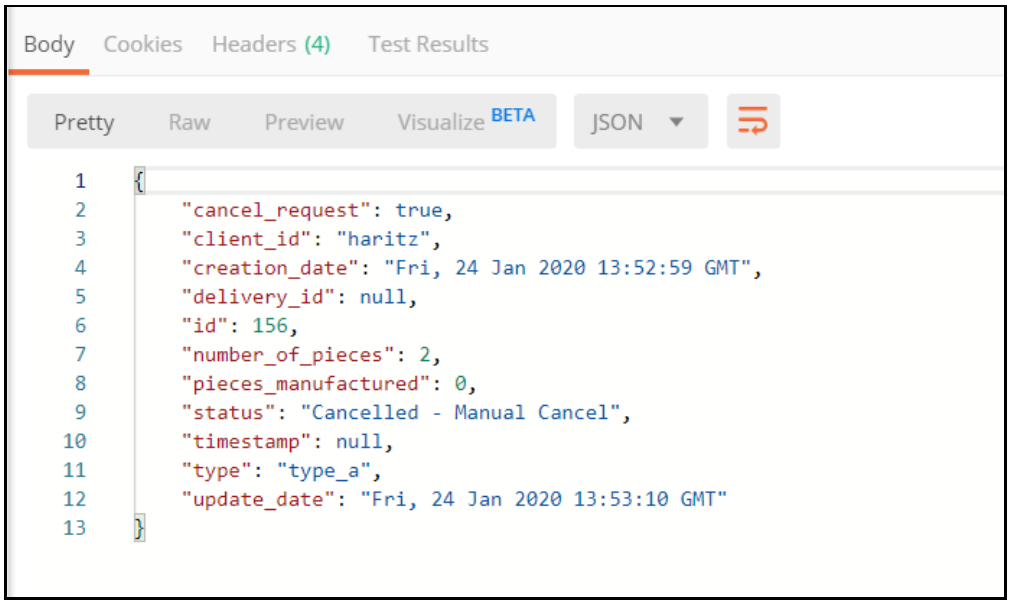


Figure 14 - Manual cancel

4.2.4. Messaging

All communications held between the microservices are done via messaging (except the first https request to obtain the auth public key). The communication structure was already well developed to make use of the loosely coupled pattern. The new microservices also follows the same pattern (See Figure 15 - Messaging architecture). Let's look to the communications that the store service uses:

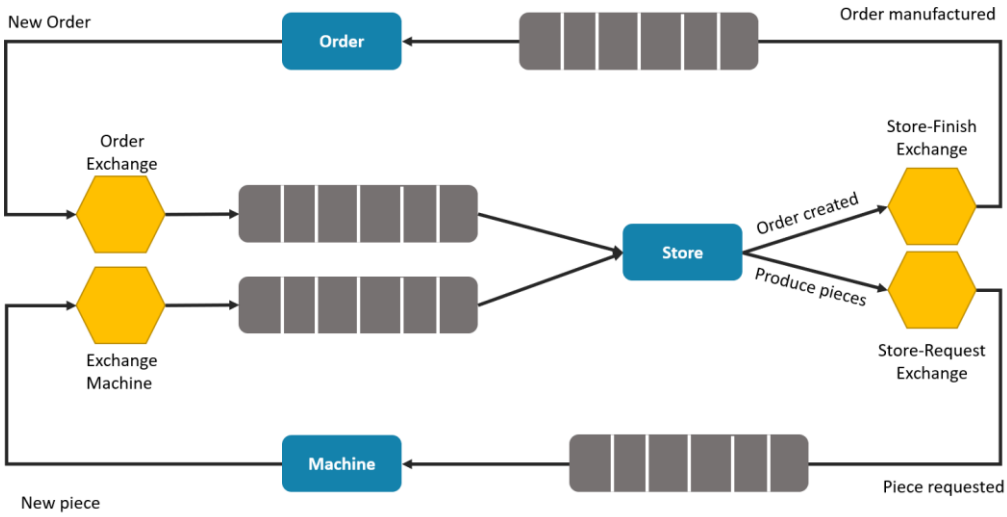


Figure 15 - Messaging architecture

It is worth mentioning that even though we haven't used a worker queue style of communication to request new pieces production, as we have used a fanout, it would

be advisable to use this type of queue if we were aiming to replicate the machine service. That way a piece could be consumed just once.

4.2.5. API Gateway and Service Discovery

As explained earlier, the factory is divided in many microservices. Each microservice provides an API endpoint to attend REST petitions. That means that each service has its own IP and port. If a user was to interact with some of those services, it would have to know in advance that it would have to change the IP and port each time it interacts with different services. From a user's experience point of view, that is not a very good design. To avoid all those issues an API gateway has been deployed. To be more precise we have used an HAPROXY gateway. That way a user may just know the IP and port to interact with all the services.

The API gateway is also in charge of performing periodic health checks to get the status of each microservice. (See Figure 16 - Service Discovery architecture)

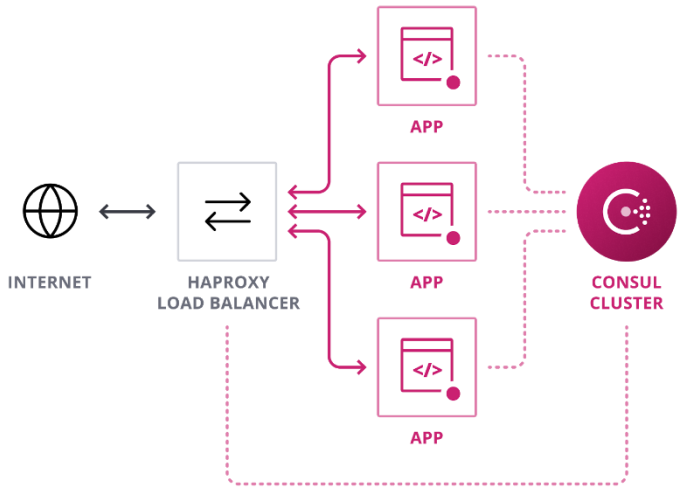


Figure 16 - Service Discovery architecture

To avoid assigning predefined IP and work in static environment, thus each service having to previously know the IP of the rest of microservices, a service discovery system has been used to alleviate this problem using the Consul tool.

4.2.6. Logging

The logging system was already developed in the previous iteration, on this iteration, we have centralized the logging system from a code point of view. We have externalized all logs classes to just one that are used by all microservices. On top of that,

the initialization of the logging object is done just once per service, that way the amount of code needed to log is reduced.

We also have added and differentiated old logs: for example, all the logs that were generated each time the health check method was called, on the previous iteration they were labelled as DEBUG. Now, these logs are labelled as HEALTH. And the same happens for the RabbitMQ logs. Now those log fall into a new label: COMUNICATIONS.

The main .env file controls the logs printed in the standard output for an improved debugging (all logs are still registered no matter the level). If the LOG_LEVEL is set to 0, then all logs are printed (Connection + Debug + Info + Health + Error), if set to 1 it will only print Debug + Info + Error and if set to 2 it will display just Info + Error logs

4.2.7. Security

The last delivered version of the factory had all the communications secured (https for client to HAPROXY, TLS for RabbitMQ coms and HTTPS for HAPROXY to microservices coms) but the ones used by consul. In this new version, those communications have been secure with SSL/TLS. In order to do that, the corresponding config file has been modified to use said secure communications protocols.

It is also worth mentioning that each microservice validates the provided JWT token to perform almost every action with them. As it was well detailed and documented in the previous document, we won't detail anything more because new functionalities regarding this topic have not been developed.

4.3. Data collection/Data gathering

Once the cloud infrastructure and the new microservices are deployed as explained before, is time to talk about the data collection. When we talk about data collection, we are talking about the process of gathering and measuring information on targeted variables in an established system, which then enables one to answer relevant questions and evaluate outcomes.

In order to carry it out, an infrastructure of two AWS EC2 instances with the following objectives have been deployed

The objective of one of the instances is to have some honeypots deployed so that the attacker has something to interact with.

On the other hand, this machine has installed other software that sends to the collector machine different information about its condition such as:

- Network packets passing through the machine.
- Metrics about the machine itself. (CPU, RAM Usage...)
- The logs of the honeypots.

The second instance has the main objective of collecting all the information that the other machine sends to it. Apart from that, it has a software that facilitates the visualization of the collected data without exporting it, which is a good feature.

After explaining the deployed infrastructure composed by the two machines, the software that houses each one will be explained.

4.3.1. Used software

The software in order to collect all the data is **Elastic Search** [1]. Elasticsearch is a distributed, open source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. In our case, it is going to store different aspects of the machine like the network traffic information, honeypot logs or metrics.

The choice of this tool over others has been based on the following aspects:

- It allows to zoom out to your data using aggregation and make sense of billions of log lines.
- It combines different type of searches: structured, unstructured, Geo, application search, security analytics, metrics, and logging.
- It is fast, and it can run the same way on your laptop with a single node or on a cluster with hundreds of servers, making very easy prototyping.
- It uses standard RESTful APIs and JSON. The community has also built and maintains clients in many languages such as Java, Python, .NET, SQL, Perl, PHP etc.
- Tools like Kibana and Logstash (Elastic Stack, useful in our context) allows to make sense of our data in very simple and immediate ways by using charts and performing granular searches.
- It provides where integrated services to cover all the key aspects of Big Data: from data capturing, to pre-processing filters, storage and visual explotation.

In order to visualize the stored the data we have chosen to use Kibana [2]. Kibana is a data visualization and management tool for Elasticsearch that provides real-time histograms, line graphs, pie charts, and maps. Kibana also includes advanced applications such as Canvas, which allows users to create custom dynamic infographics based on their data, and Elastic Maps for visualizing geospatial data. (See Figure 17 - Kibana Dashboard Example)

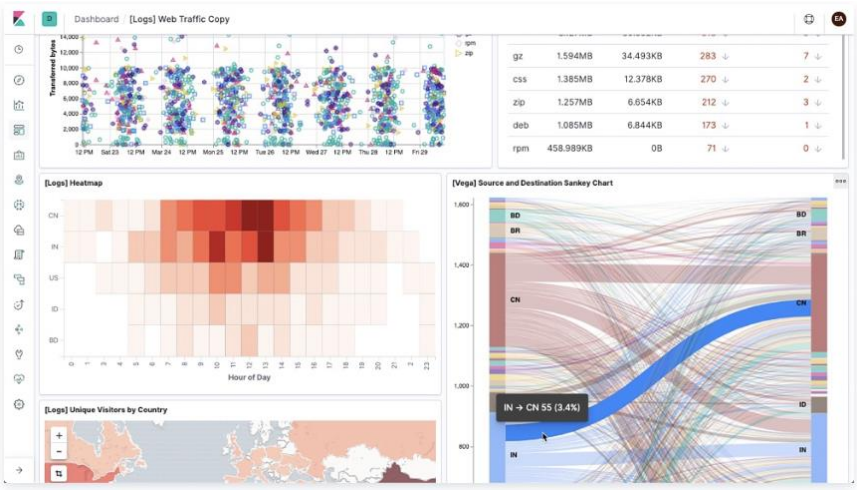


Figure 17 - Kibana Dashboard Example

The main reason for choosing Kibana over other data viewers is that it has a very simple complement to our data store, in this case ElasticSearch.

To send logs from the deployed honeypots to Elasticsearch, **Filebeat** is also included in the infrastructure. Filebeat is a lightweight shipper for forwarding and centralizing log data. Installed as an agent on our system, Filebeat monitors the log files or locations that you specify, collects log events, and forwards them to either to Elasticsearch or Logstash for indexing.

In this case, it receives the log from the honeypots and sends to Logstash for a treatment.

On the other hand, we have chosen **Logstash** [3] as our log management tool. Logstash supports several inputs, codecs, filters and outputs. The inputs are the data sources. Codecs essentially convert an input format into a Logstash accepted format, as well as transforming from the Logstash format to the desired output format. These are commonly used if the data source is not a plain line of text. Filters are actions that are used to process events and allow you to modify them or delete events after they have been processed. Finally, outputs are the destinations where the processed data must be derived, in our case, Elastic Search.

In our case, Logstash receives the log of the honeypot from Filebeat and when a source IP direction is included, it discovers the location of that IP and adds it to the index, so that it can be viewed through Kibana.

Another point to consider, is that different metric of the host is taken with **Metricbeat** [4]. Metricbeat is a lightweight shipper that can be installed on servers (in our case an AWS EC2 instance) to periodically collect metrics from the operating system or services running on the server. Metricbeat takes the metrics and statistics that it collects and ships them to the output that you specify, such as Elasticsearch or Logstash.

In addition, it can send the metrics to disk, so the pipeline doesn't skip a data point, even when interruptions such as network issues occur. Metricbeat holds onto incoming data and then ships those metrics to Elasticsearch or Logstash when things are back online, which is a very good feature.

And eventually, Metricbeat is part of the elastic stack, which means it works perfectly with Logstash, Elasticsearch and Kibana. So, it comes in handy with the pre-selected software.

As far as sending information about network packages is concerned, among other things, it has been decided to have **Packetbeat** [5]. Packetbeat is a lightweight network packet analyzer that sends data from your hosts and containers to Logstash or Elasticsearch. In our case it is interesting because it captures all the traffic that passes through the machine, which is useful when the aim is to detect anomalies.

So, once the main software is explained the mounted topology is illustrated with the different software. (See Figure 18 - Data collection architecture)

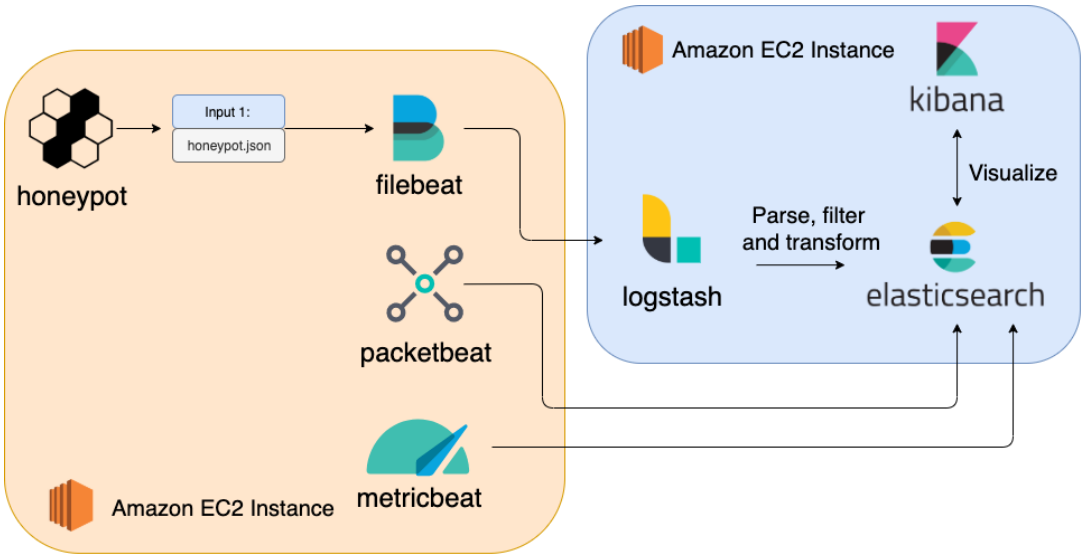


Figure 18 - Data collection architecture

After explaining the main software used on the machines, the most complex ones are going to be explained on the next section.

4.3.2. Honeypots

In computer terminology, a honeypot is a computer security mechanism set to detect, attempts at unauthorized use of information systems. Generally, a honeypot consists of data that appears to be a legitimate part of the site, but is isolated and monitored, and that seems to contain information or a resource of value to attackers.

4.3.2.1. Types

Honeypots can be classified depending on many points of view, but if we look at the deployment, honeypots can be classified as:

- **Production honeypots** are easy to use, capture only limited information, and are used primarily by corporations. Production honeypots are placed inside the production network with other production servers by an organization to improve their overall state of security. Normally, production honeypots are low-interaction honeypots, which are easier to deploy. They give less information about the attacks or attackers than research honeypots.
- **Research honeypots** are used to gather information about the motives and tactics of the black hat community targeting different networks. These honeypots do not add direct value to a specific organization; instead, they are used to research the threats that organizations face and to learn how to better protect against those threats. Research honeypots are complex to deploy and maintain, capture extensive information, and are used primarily by research, military, or government organizations.

Depending on the design, honeypots can be classified as:

- **Pure honeypots** are full-fledged production systems. The activities of the attacker are monitored by using a bug tap that has been installed on the honeypots link to the network. No other software needs to be installed. Even though a pure honeypot is useful, stealthies of the defense mechanisms can be ensured by a more controlled mechanism.
- **High-interaction honeypots** imitate the activities of the production systems that host a variety of services and, therefore, an attacker may be allowed a lot of services to waste their time. By employing virtual machines, multiple honeypots can be hosted on a single physical machine. Therefore, even if the honeypot is compromised, it can be restored more quickly. In general, high-interaction honeypots provide more security by being difficult to detect, but they are expensive to maintain. If virtual machines are not available, one physical computer must be maintained for each honeypot, which can be exorbitantly expensive.
- **Low-interaction honeypots** simulate only the services frequently requested by attackers. Since they consume relatively few resources, multiple virtual machines can easily be hosted on one physical system, the virtual systems have a short response time, and less code is required, reducing the complexity of the virtual system's security.

4.3.2.2. Selected honeypot

We have deployed two honeypots on our infrastructure: Cowrie and Dionaea.

Cowrie [6] is a medium to high interaction (depending on the configuration) SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system.

Features:

- Fake filesystem with the ability to add/remove files. A full take filesystem resembling a Debian 5 installation is included.
- Possibility of adding fake file contents so the attacker can cat files such as /etc/passwd. Only minimal file contents are included.
- Cowrie saves files downloaded with wget/curl or uploaded with SFTP and SCP for later inspection.
- Session logs are stored in an UML Compatible format for easy replay with the bin/payload utility.
- SFTP and SCP support for file upload.
- Support for SSH exec commands
- Logging of direct-tcp connection attempts
- Forward SMTP connections to SMTP Honeypot
- JSON logging for easy processing in log management solutions, what is very interesting to send it through Filebeat.
- Another point in favor is that it is docker compatible.

On the other hand, **Dionaea** [7] is a malware capturing low interaction honeypot that aims to trap malware exploiting vulnerabilities exposed by services offered over a network, and ultimately obtain a copy of the malware.

Dionaea features a modular architecture, embedding Python as its scripting language in order to emulate protocols. It can detect shellcodes using LibEmu and supports IPv6 and TLS.

The protocols that Dionaea traps malware from are the following:

- **Server Message Block (SMB):** SMB is the main protocol offered by Dionaea. SMB has a decent history of remote exploitable bugs and is a very popular target for worms.
- **Hypertext Transfer Protocol (HTTP):** Dionaea supports HTTP on port 80 as well as HTTPS. A self-signed SSL certificate is created at startup for HTTPS.
- **File Transfer Protocol (FTP):** Dionaea provides a basic FTP server on port 21. It allows creation of directories, uploading and downloading of files.

- **Trivial File Transfer Protocol (TFTP):** Dionaea provides a TFTP server on port 60 which can be used to serve files.
- **Microsoft SQL Server (MSSQL):** Dionaea implements the Tabular Data Stream protocol which is used by Microsoft SQL Server. Listening to TCP/1433 and allowing clients to login, it can decode queries run on the database.

In our case, the main function of the honeypots is to send the data collected from the honeypots to Elasticsearch (via Filebeat and Logstash) in order to be shown on the Kibana dashboard. This dashboard is described in 4.6. Kibana dashboard.

4.4. Cloud security/Services

AWS offers a multitude of services aimed at infrastructure security. The highlights of the services are aimed at preventing, detecting, responding and solving possible attacks that may occur from the Internet.

The most interesting security services offered are WAF, AWS Shield, AWS Inspector and AWS Secrets Manager.

Web Application Firewall: AWS offers a rule-based firewall that protects against SQL injection attacks, XSS attacks and the major OWASP security risks. Amazon also periodically updates the rules that protect the infrastructure from OWASP risks that arise over time.

This firewall works with packet filters based on packet content. WAF protects many AWS services such as CloudFront, Load Balancers and API Gateway.

AWS Inspector: Amazon Inspector automatically evaluates applications for exposures, vulnerabilities, and deviations from best practices. Also performs an assessment and generates a detailed list of security-related findings, sorted by severity level. These results can be reviewed directly or as part of detailed assessment reports that are available through the Amazon Inspector console or the API.

AWS Shield: AWS Shield is a distributed denial of service (DDoS) protection service that protects applications running on AWS cloud platform.

AWS Secrets Manager: AWS provides a secret management tool to store the secrets generated by the microservices or the passwords needed to secure certain parts of the infrastructure. The problem is that it cannot be used with the Vocareum student account and it has been necessary to find alternatives to this, such as using Vault.

Most of these services cannot be used with Vocareum's basic accounts, so we must limit ourselves to studying how they should be applied in an account with full AWS control.

4.4.1. Secrets manager tool. Vault

It is very common to use passwords, tokens, API keys or in short, secrets, in software applications or in complex infrastructures where multiple services interact with each other in a secure manner.

In addition, there is a risk that this sensitive information can be read by an attacker in case it penetrates in one of the systems, although there is also a risk that a software developer, who is maintaining these services, has total access to the infrastructure systems and can read that secret. A secret manager abstracts all these agents from the direct use of passwords and ensures that they are stored securely within management software. Thus, when a secret is needed, this software will be asked about a particular secret and the software will return the secret as long as the person asking for it has the necessary credentials to ask for it.

Used software is Vault, by Hashicorp.

This manager works fine with lots of different engines such us AWS, RabbitMQ, GoogleCloud, Azure Devops, Consul... or just with generic secrets using key/value dictionaries.

On this cloud infrastructure it has used to save Elastic/Kibana user/password's GUI values as a prove of concept. (See Figure 19 - Vault with Elastic credentials)

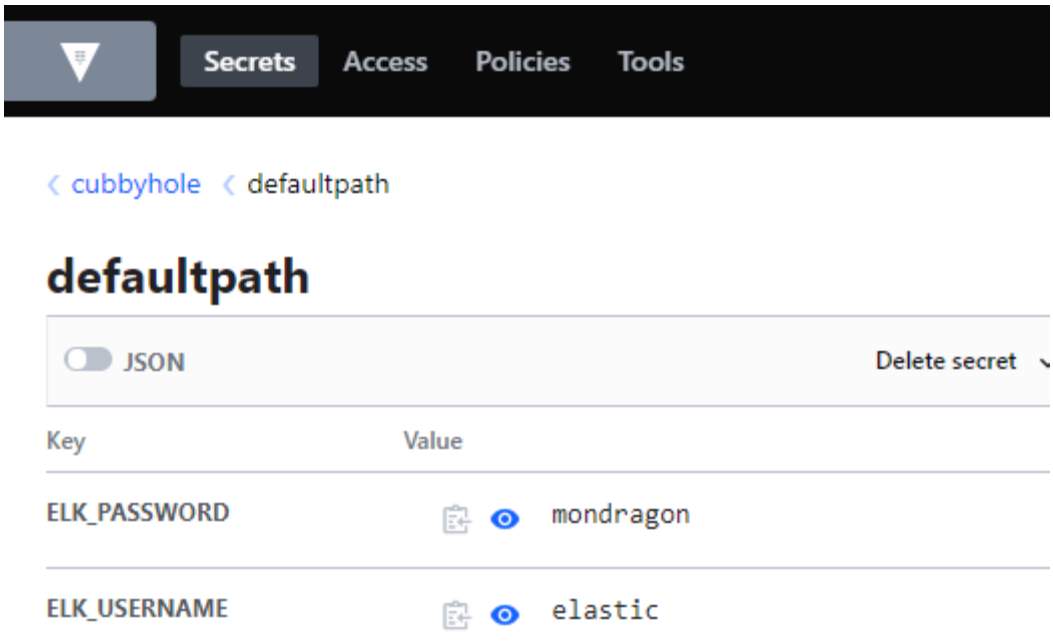


Figure 19 - Vault with Elastic credentials

Those values are token from the CloudFormation, USER DATA script as it can see in the snippet:

```
#!/bin/bash
exec > >(tee /var/log/user-data.log|logger -t user-data ) 2>&1
sysctl kernel.hostname=DataVisualization
curl -sO https://releases.hashicorp.com/vault/1.3.1/vault_1.3.1_linux_amd64.zip
unzip vault_1.3.1_linux_amd64.zip
sudo mv vault /usr/local/bin/
export VAULT_TOKEN=s.0mSjTZBlBhcEq6Zuf3jJuOpY
sudo chmod +x ./get_secret.sh
export ELK_PASSWORD=$(./get_secret.sh ELK_PASSWORD)
export ELK_USERNAME=$(./get_secret.sh ELK_USERNAME)
echo $ELK_PASSWORD
echo $ELK_USERNAME
cd elk-monitoring/logstash-elastic-kibana
docker-compose up -d
```

ELK_PASSWORD and ELK_USERNAME are exported to the environment to be used by docker-compose engine.

get_secret.sh script is a call to vault command in order to read the parameter given.

There is another way to ask passwords to vault via Python, with hvac library but it has not been used in our case.

4.4.1.1. Future improvements about Vault uses

Vault has built-in support for secret revocation. Vault can revoke not only a single secret, but also a tree of secrets. For example, Vault can revoke all secrets read by a specific user or all secrets of a specific type. Revocation assists in key rolling as well as locking down systems in the case of an intrusion. This could be great if microservices start using Vault in order to save generated JWT tokens to increase the security and the management to the microservices administrator. At this moment, generated tokens are not saved and just are checked by the public/private key.

Vault gives more flexibility in this way because tokens can be stored there for a specific user.

4.4.2. SSL/TLS Certs

All critical services deployed on this infrastructure are using SSL/TLS certificates generated by our auto signed CA that is used by all the infrastructure (microservices as well as ELK stack)

There are two main communications that need to be secured:

- **Microservice infrastructure**: Check section X to get more information about this topic
- **Elastic stack:** The rest of the host communicates with each other. Elasticsearch communicates with the honeypots through packet beat, metric beat. The honeypot also communicates with Logstash trough file beat. The microservice also uses the beat family to communicate with the rest of the stack. Kibana exposes an endpoint to a WebApp. It also communicates with Elasticsearch. All those communications explained so far, are secured with SSL/TLS.

Subject Alternative Name (SAN) is an extension to the X.509 specification that allows users to specify additional host names for a single SSL certificate. (See Figure 20 - Certificate)

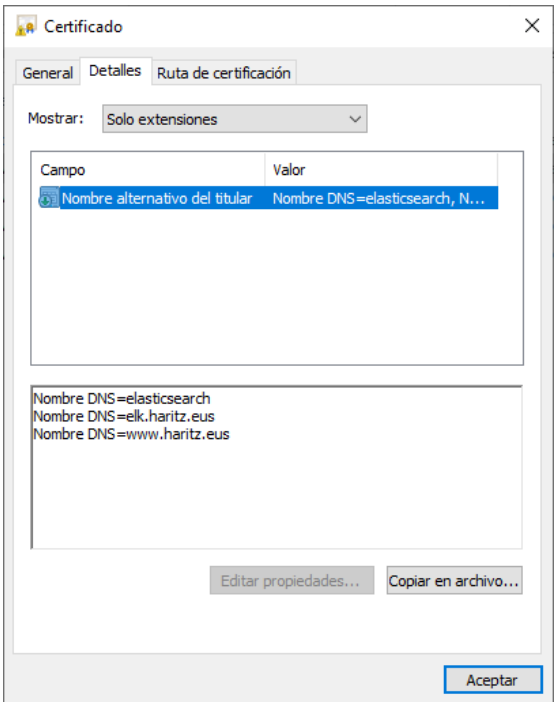


Figure 20 - Certificate

4.4.3. Backup Policies

Currently no machines are backed up, but AWS Backups has been identified as a service to automate this task in AWS. This service cannot be used with the current AWS account.

This service allows for the automation of backup scheduling through a JSON file with the backup plan, linking it through CloudFormation when generating the infrastructure.

In the future, we plan to integrate this service through a total AWS usage account with the following rules executed daily

- A total backup microservices, docker volumes, is going to be saved on S3 bucket backup.
- After 30 days of the backup, files are going to be transferred to S3 Glacier.
- After 24 months of the backup, files are going to be deleted.

4.5. Data Analytics

This section will focus on the development of data analysis and the proposed validation of the unsupervised learning model.

4.5.1. Introduction to Data Analytics scenario

One of the core pillars of this project focuses on trying to detect real anomalies on the deployed hosts and architecture. On one hand we have deployed the microservices that simulate a factory that produces orders. On the other hand, we also have deployed a set of honeypots to try to detect unlawful intrusions.

As described in previously, the data subjected to analysis is obtained with Packetbeat as well as with Metricbeat. This data does not include any information obtained by de honeypots but its monitorization. With those tools we are able to extract diverse data from two data sources.

Once the clustering algorithm is applied to the available data, it is necessary to evaluate, analyze and obtain logical and coherent conclusions. In this section we propose a validation framework to validate the obtained model.

The proposed framework is designed for this problem: Detecting anomalies in the field of cybersecurity. To be more precise, it will only work while trying to detect some anomalies that are characterized by the network traffic (or flow) and host performance.

4.5.2. Theorical framework for model validation

In this section we will describe how to validate a proposed model and how to decide whether it is valid or not.

The most basic analysis that should be done would be to observe the obtained clusters. We recommend using visuals tools to represent those clusters. As we have used **DBScan** [8] (a density bases algorithm), it should be relatively simple to understand the separation between clusters. Projecting different features in a scatter plot will display different knowledge. Plotting two features might not reveal any clear separation, but perhaps another combination of features might show a clear separation. If that is the case, the first conclusions can be extracted. A given cluster is divided because X and Y features have certain values. But that is not enough.

It is also recommended to test with different algorithms. We propose to use HDBScan. It is a variation of DBScan that focuses on transforming into a hierarchical clustering algorithm. If after applying different algorithms (and different or similar hyperparameter configurations) the same (or similar) results are obtained, then we could be one step closer to validate the coherence of the obtained results. But again, this is not enough.

If after applying other clustering algorithms different results are obtained, then it might indicate that the obtained results are not correct. In that case, it would be necessary to try different approaches. For example, using different hypermeters configuration, or different pre-processing techniques (this topic will be furtherly discussed in this section).

Another important step to validate the model, is to test it with well-known (or labelled) data. For example, if we want to detect a (D)dos attack, then we know that the CPU usage during the attack is way higher if compared with the normal average of the system. Testing the model with the simulated data will be helpful. If the model clusters the data as an outlier or as another separated cluster, then we can be sure that for that particular attack, the model subjected to validation is able to detect the attack.

Unfortunately, it is not possible to test the model with all the possible attacks. First of all, it will require to have a strong knowledge of all the possible attacks. That is, knowing the attack, understanding how they work and how they affect the host. It might happen that the recollected data is not enough to detect that particular attack. After all that, we would need to simulate the attack. That could be done by actually performing the attack and registering new data (also called simulation).

Another validation technique can be obtained by applying a cluster validation algorithm called Silhouette. Sci-kit learn provides a function that returns a score. This algorithm is calculated taking into account the mean intra-cluster distance and the mean nearest-cluster distance.

Last but not least, and perhaps the most important aspect of this topic: Try new preprocessing methods. In order to carry out a good preprocessing “pipeline”, it is strongly recommended to understand the available data. That can be done by performing an exploratory data analysis or EDA.

An EDA analysis focuses on my aspects. First, it focuses on understanding the nature of the data. That is knowing if we are dealing with numerical or qualitative data. If we take into consideration that DBSCAN only allows numerical data, a transformation will be required, also called label encoder.

Once we know the nature of the data, we will drop all the unwanted data. But, how do we know which features to use? Well, that is a tricky question. One approach would be to use PCA as it reduces the model to a set of linear components that gather as many data variation as specified. It is recommended to capture 0.95 (or greater) if using PCA. It also could be possible apply feature selection algorithms to obtain those features.

4.5.3. Data processing

In our case, we have two datasets. One focused on network traffic and another one focused on host performance. When using the first one, we have used all the available features to perform the clustering algorithm. The features used on the second dataset are just those involved to capture CPU usage or performance.

It is also strongly advised to look to the actual data. In other words, check that we have data. NaN values must be treated. There are many techniques. We could impute a value in each position using the mean or the median. But we don't recommend applying those methods as we might increase data noise. Instead, we will delete all the rows containing NaNs for that actual reason. To reduce noise.

After treating NaNs the remaining data has to be standardized or normalized. We have taken the approach of standardizing the data by subtracting the mean of each feature and dividing it by the standard deviation.

Another aspect than in another context will be considered, is the outlier's treatment. We have decided to not perform any operation or procedure to treat with them, as the core problem behind the main problem is to detect those outliers as anomalies.

With that we could conclude the preprocessing scheme and start with the actual clustering. First, we would decide the algorithm (DBScan) and then, another key and fundamental aspect, is the hypermeter configuration and optimization of said algorithm.

As a synthesis of the preprocessed carried out in our preprocessing pipeline, we have followed the following steps.

1. Read CSV
2. Select desired features
3. Remove rows containing NaNs
4. Standardize timestamps from elastic to epoch
5. Standardize numbers from elastic to integers or floats
6. Standardize data
7. Apply DBScan

Used features in Packetbeat:

```
[
'timestamp',
'destination.bytes',
'destination.ip',
'destination.packets',
'destination.port',
'event.duration',
'labels_intrusion_type',
'network.bytes',
'network.packets',
'network.transport',
'network.type',
'source.bytes',
'source.ip',
'source.packets',
'source.port'
]
```

Used features in Metricbeat:

```
[   "@timestamp",
      'system.cpu.cores',
      'system.cpu.idle.pct',
      'system.cpu.iowait.pct',
      'system.cpu.irq.pct',
      'system.cpu.nice.pct',
      'system.cpu.softirq.pct',
      'system.cpu.steal.pct',
      'system.cpu.system.pct',
      'system.cpu.total.pct',
      'system.cpu.user.pct'
]
```

In this last section we will discuss and describe why we have used the DBScan algorithm. DBScan is a clustering algorithm based on density. It finds the core clusters of high density and then expands from them to cluster all the available data. Sci-kit learn DBScan algorithm allows the tuning of some hyperparameters. The most important ones are epsilon which indicates the maximum distance between two points to be considered as they are in the same cluster. The second parameter called min_samples specifies how many data points or samples have to be to form a new cluster. If a potential cluster doesn't reach said value, then they are treated as outliers. All detected outliers are assigned to the cluster - 1.

It is said that DBScan works well with datasets that have clusters of similar density. In our case, we expect that the normal traffic and state of each host is somewhat stable and constant, so anomalies are part of that normality or they are just outliers. For that reason, we can justify the use of said algorithm.

4.5.4. Theorical framework in practice

Let's start by analyzing the results obtained with metric beat. We have applied the preprocessing pipeline described earlier to the available data. Said data contains a DDos that we have simulated with a script that requested many login attempts to the factory microservices. While we were performing that attack, we monetarized those results with Kibana and we could observer that the CPU was at its peak computational performance:

The first gauge chart acknowledges that our DDos worked correctly. In order to try to capture that attack by our model, we will use a dataset during said period.

Let's understand the second graph. The Y axis represents the CPU usage between 0 and 2. 2 means that the host is using 200% of a CPU usage. That means that it is using 2 cores. And that makes sense as the machine used was an amazon EC2 T2 medium, and that machine has 2 cores. As we can observe, there some points that are not clustered. We can understand that because they were assigned to the -1 cluster. The second cluster (cluster 0) are all those points painted in red. It can seem impossible to have at the same time a blue dot (meaning overperformance) and a red one (meaning moderate use of CPU), but if we zoom to those sections, we have the answer:

We observe that during small periods the CPU was over performing and during the rest of the time was doing a moderate usage of CPU and no overlapping occurs.

After applying the Silhouette algorithm, we obtained a = 0.47 score. It is not a good score, but it is not the only metric to take into consideration. The fact that we can discriminate those outliers because of the attack is really significant and becomes more important than a score.

To analyse the results obtained for the Packetbeat dataset, we will be using the dashboard developed with **plotly – dash** [9].

After analyzing the obtained results, we are not able to make great conclusions. The dataset and algorithm used may be poor designed or the preprocessing pipeline used was not well executed.

The only conclusion that we extracted was that differencing factor between the two clusters, was that the event duration. After analyzing the data and the results we are certain that there is a mistake somewhere in the data analytics chain for this dataset.

4.5.5. Data visualization

As shown in the previous section, data visualization is a key aspect to understand the results obtained with our model, in this section we will describe the structure of a

dashboard developed for this problem and datasets. In fact, there are two dashboards. One for each dataset, but they share the same structure. (See Figure 21 - Packetbeat controller dashboard)

Figure 21 - Packetbeat controller dashboard

The dashboard is structured in two main sections. The first section is located on the left and its main goal is to provide some controls to the user to interact with the model and data visualization. It has two inputs that affect how the model is built. For example, the user may change the epsilon field or the min samples parameters that will be used by the DBScan algorithm. In other words, the user can change the hyperparameters without having to change them from the source code. Once they release the focus of one of those parameters, the model is reevaluated with the new parameters.

This section also allows the user to change how the model is displayed. A user can change the X and Y axis as well as selecting those tooltips that will be displayed when a data point is hovered.

The second section of the dashboard consists of the parts. The first one displays the raw file size used by the model.

The second one is an interactive scatter plot that shows the cluster results. It is possible to show or hide some clusters by clicking them on the legend section. If the model had outliers (meaning that some data was clustered to the -1 group), then scatter plot paints all those data points with a black border around them. During the load time that takes to obtain the model, a loading gif is used to provide a better user experience.

It is important to make sure that the dashboard is interactable. High load times lead to worse user experience. After clustering the data, the dashboard has to paint all the data points. If the dataset is too big, the load time will increase dramatically. In order

to reduce the load time, the dashboard will only plot a limited amount of data points per cluster. By default, that limit is set to 1000 points (or less if the cluster doesn't have said amount of points) per cluster.

Finally, the last visualization tool displayed is a data table that synthesizes the clusters information. The columns displayed are related with the scatter plot as well as with the X and Y controllers. Each row shows the average value after grouping the dataset by the cluster label of said X and Y features. It is also interactive. A user can sort each column by clicking on small arrows that appears when a column data table header is hovered.

4.5.6. Interaction

We are living in a world which faces a rapidly increasing amount of data to be dealt with daily. In the last decade, the steady improvement of data storage devices and means to create and collect data along the way influenced our way of dealing with information: Most of the time, data is stored without filtering and refinement for later use.

Virtually every branch of industry or business, and any political or personal activity nowadays generate vast amounts of data. Making matters worse, the possibilities to collect and store data increase at a faster rate than our ability to use it for making decisions.

However, in most applications, raw data has no value in itself; instead we want to extract the information contained in it. The information overload problem refers to the danger of getting lost in data which may be:

- Irrelevant to the current task at hand
- Processed in an inappropriate way
- Presented in an inappropriate way.

The overarching driving vision of visual analytics [10] is to turn the information overload into an opportunity: Just as information visualization has changed our view on databases, the goal of Visual Analytics is to make our way of processing data and information transparent for an analytic discourse.

The visualization of these processes will provide the means of communicating about them, instead of being left with the results. Visual Analytics will foster the constructive evaluation, correction and rapid improvement of our processes and models and the improvement of our knowledge and our decisions.

All this can be seen perfectly in the following illustration. (See Figure 22 - Visual Data Exploration diagram)

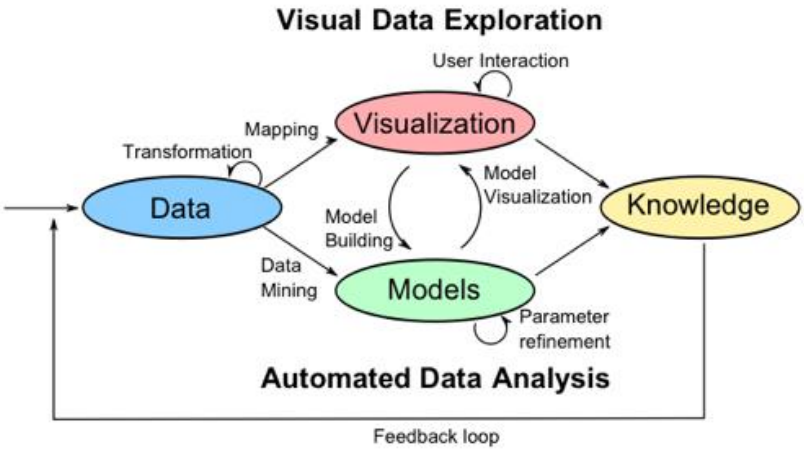


Figure 22 - Visual Data Exploration diagram

Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making based on very large and complex data sets.

The goal of visual analytics is the creation of tools and techniques to enable people to:

- Synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data.
- Detect the expected and discover the unexpected.
- Provide timely, defensible, and understandable assessments.
- Communicate assessment effectively for action.

This can be reflected in our project as follows:

We have used the developed and previously explained dashboard to extract relevant information and knowledge. When we used the dashboard for the first time, we wanted to use it to help us tune the hyperparameters used by DBScan. We adjusted those values to obtain more coherent results: If the number of clusters where too big and we could observe that some clusters that where labelled into different clusters, then we will increase the min samples value as well as the epsilon value. If the clusters were too small and we could clearly separate some of them, then we would reduce the parameters. If the results were not that great, new pre-processing techniques or methods where used to obtain different results. That tuning help us refine the final model.

Once we have decided that we had a good-looking clustering information, then we had to understand why those clusters where labelled that way. By changing the axis and adding new tooltips features we observed that some clusters, in the case of the packet beat dataset, where labelled that way because of the event duration feature as they had a clear and easy way of discrimination. In the case of the metric beat dataset,

it helped us acknowledge that the discrimination factor was that higher CPU usage led to label the dataset.

4.6. Kibana dashboard

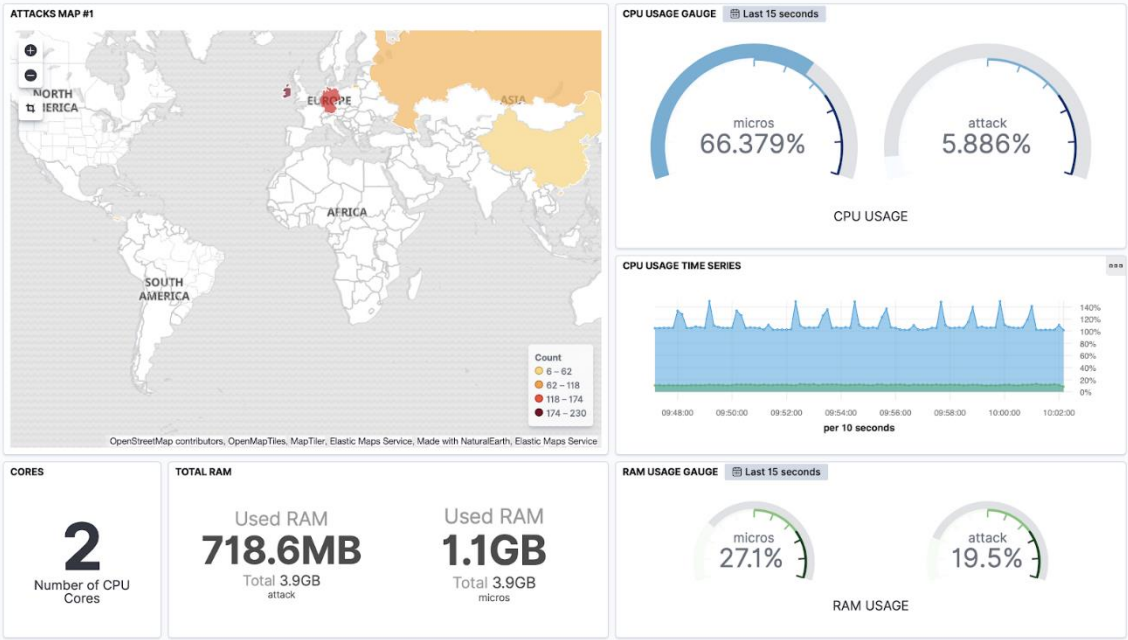
Nowadays it is convenient to have the status of deployed hosts monitored. This monitoring allows the client to have control of the situation in which the equipment is. The data that can be interesting to know are:

- CPU Usage of the hosts
- RAM Memory usage
- The location from which they are trying to access.
- The most used usernames and passwords to log into the honeypots

It is important to emphasize that besides having controlled the current situation of the system, it is convenient to show the value through time by means of a temporal series.

To solve this problem, it has been decided to implement a dashboard in Kibana since all the data to be shown are stored in Elasticsearch. And these complement each other in a simple way, so it should not be complicated to implement this interface.

Before explaining which visualization has been decided to implement to show each variable, an overview of the dashboard will be shown. (See Figure 23 - Kibana Dashboard)



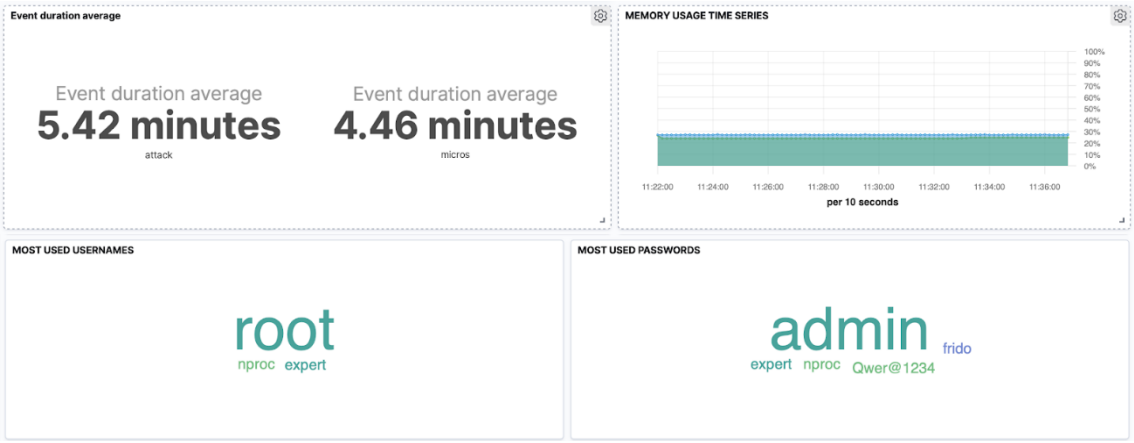


Figure 23 - Kibana Dashboard

Once we have the overview of the interface, it is time to go into detail in each display explaining which variable has been considered in each one of them.

4.6.1. Attacks map

In this case, the data to be displayed are the countries from which access is attempted/accessed. This data is collected by Logstash (remember that it receives from Filebeat, which at the same time receives from the log files of the honeypots) and stored in Elasticsearch. (See Figure 24 - Attack location map)

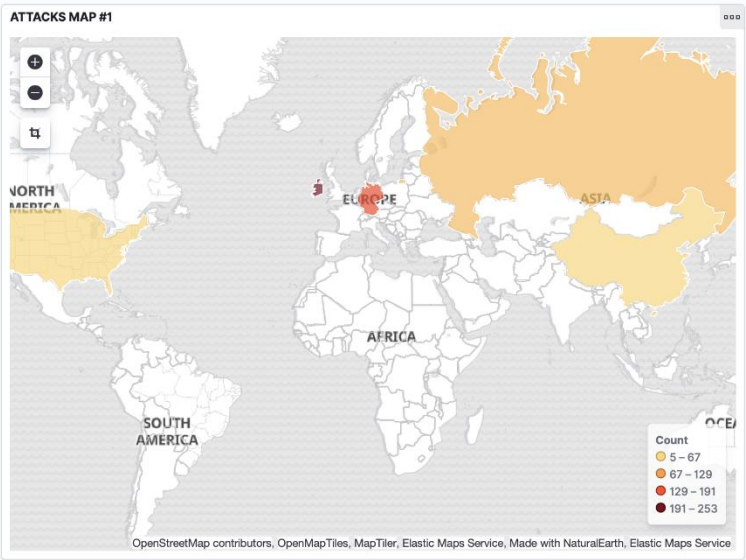


Figure 24 - Attack location map

The maps have been used since ancient times to collect geographic information and transmit it. We can understand a map as a means of visual communication that constitutes a language with an objective: the description of spatial relations. So, it seems to us to be the right one to visualize the attempts of attack in this case.

4.6.2. CPU Usage

As the CPU usage is usually shown in percentages and this usually varies through time, it is interesting to show it in a gauge type visualization, since it shows the value with respect to the totality, in this case 100%. (See Figure 25 - CPU Usage gauge)

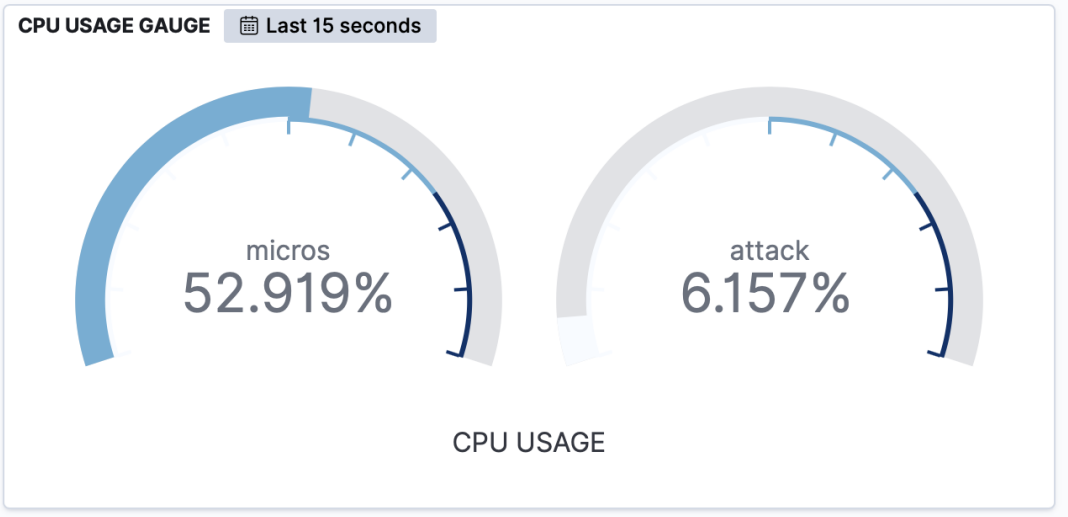


Figure 25 - CPU Usage gauge

On the other hand, in order to visualize the trend of the value taken by the percentage of CPU usage through time, it has been decided to implement a time series graph. (See Figure 26 - CPU Usage time series)

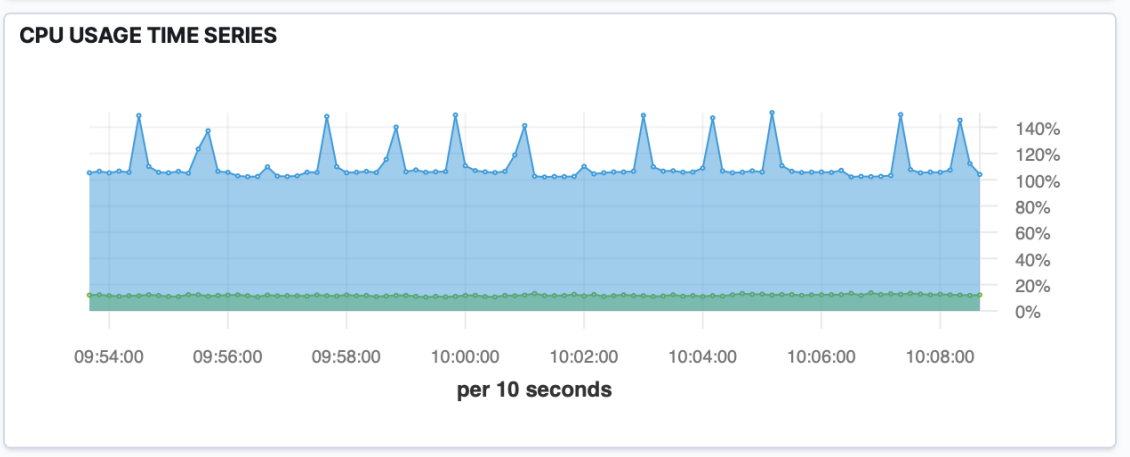


Figure 26 - CPU Usage time series

It is worth mentioning that this one exceeds 100% since the value has not been divided by two, so it should be considered alarming when the percentage of use approaches 200%.

Also, all these metrics are being collected by the Metricbeat software installed on the hosts and stored in Elasticsearch, so the implementation in Kibana has not been complicated. Apart from investigating which variables were suitable to visualize.

4.6.3. RAM Memory usage

Regarding the use of RAM memory, the implementation of its corresponding visualization has followed a similar philosophy to that of the use of the CPU. (See Figure 27 - RAM Memory usage gauge and Figure 28 - RAM Memory usage time series)

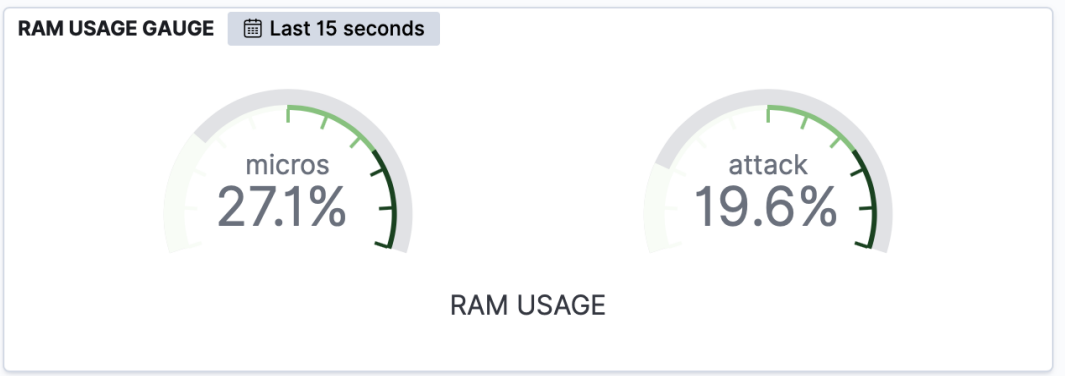


Figure 27 - RAM Memory usage gauge

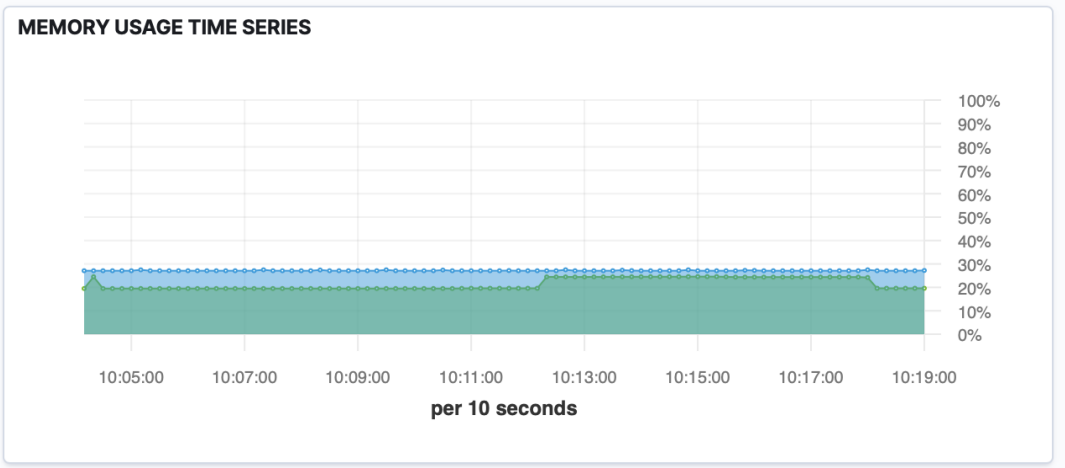


Figure 28 - RAM Memory usage time series

Although in addition to the graphs explained in the previous section, it has been decided to implement a metric that shows the amount of memory in use at that time and the total of the machine, which in this case, being an AWS EC2 T2.medium instance, is 4GB. (See Figure 29 - RAM Usage visualization)

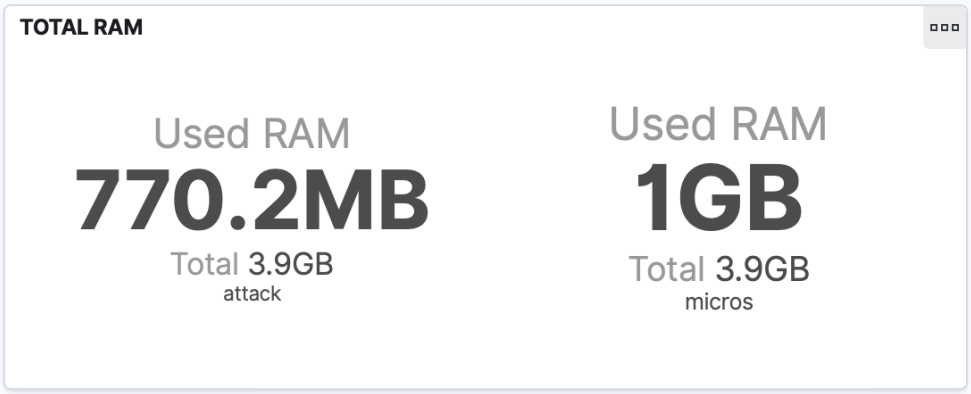


Figure 29 - RAM Usage visualization

It is worth mentioning that we have tried as much as possible to play with the visual variable of color in order to properly differentiate both the use of the hardware component that we are trying to visualize (CPU, RAM) and the machine that we are trying to visualize.

This can be seen in that the gauges that show the frequency of RAM usage are greenish and the CPU gauges are blue. On the other hand, the time series differentiates the machine with colors as well.

If these variables were not enough to differentiate both the component and the machine, some tags have been added to help us differentiate the machine from which the information is displayed. Besides, in the time series, when the hover action is performed, it shows detailed information, giving both the value and the origin.

4.6.4. Event duration average

It has been interesting to show the average duration of each transmission that happens in the system, for it we have decided to show it with a simple label since it seems to us that it describes itself wonderfully. (See Figure 30 - Event duration visualization)

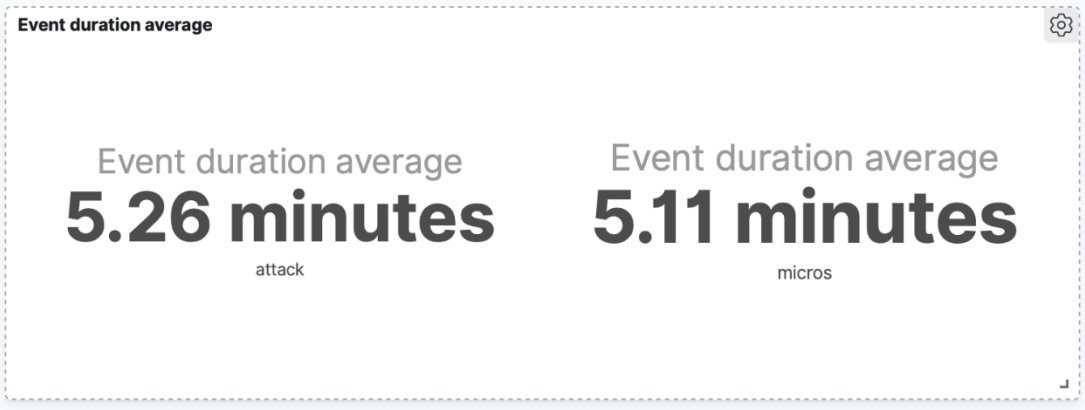


Figure 30 - Event duration visualization

This defines that the average duration of each transmission that occurs in the machine, in this case, is around 5 minutes.

4.6.5. Most used usernames and passwords

At the time of making the dashboard we found interesting to visualize to the user the usernames and passwords that the attackers used frequently in order to gain access to the machines. It could be the case that knowing these data, they could somehow ban the access attempts with those users so that they directly reject the connections with those credentials, in order not to saturate neither the networks nor the hosts.

Therefore, we have made use of a visualization provided by Kibana called Tag Cloud, where the most important values are seen in the form of a cloud.

It should be noted that this visualization plays with the visual variable of color in order to differentiate one password from another. But it is more important to highlight the visual variable of the size because through it helps us to differentiate the frequency of use of some against others.

In this case if we only contemplate the data collected in the last 15 minutes, the passwords are not too many. (See Figure 31 - Most used Username/Password visualization)

Figure 31 - Most used Username/Password visualization

But if we modify the number of data to contemplate, we can observe that this variety of data increases. (See Figure 32 - Most used Username/Password visualization (with more data))

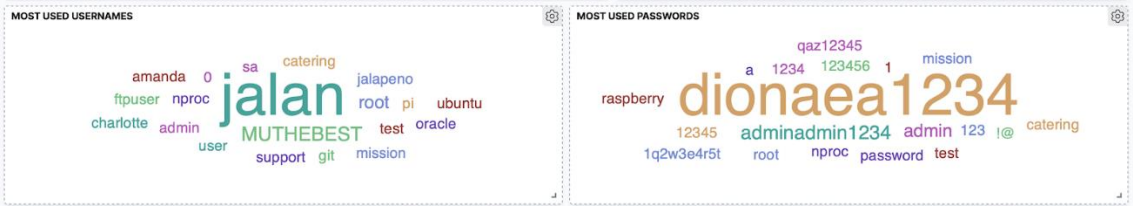


Figure 32 - Most used Username/Password visualization (with more data)

Also, with the help of visual variables of size and color, we can clearly differentiate that the most frequent user has been "jalan" and the password "dionaea1234".

5. Conclusions

The conclusions can be divided into different context. We will look at it from the point of view of automation, security, microservices, data analysis and visualization in the implemented dashboards.

Therefore, this section will be analysed by the already mentioned points.

5.1. Infrastructure as Code

The deployment of the entire infrastructure has been decided to be deployed at the AWS cloud service provider. Amazon offers different services at different levels such as IaaS, PaaS and SaaS. It also has its own tool for automating the creation of infrastructure, called CloudFormation, which is equivalent to Terraform on AWS.

This tool allows the total creation of the infrastructure through code and therefore it is possible to make a versioning of it to make small modifications to adapt to new requirements. This is also known as Infrastructure as Code.

The time and development curve to make this automation script starts being slow at the beginning, but once the writing is finished it is possible to deploy, update or remove infrastructures in a few minutes.

5.2. Security

There are different layers of security that can be implemented in an infrastructure: at the host, network or subnet level, or with the cloud provider's own tools that facilitate anomaly detection processes or firewall implementations.

In the development of the infrastructure, security mechanisms have been focused on the network and subnet layers. Due to the limitations that the AWS starter had, it has not been possible to take advantage of all the potential of the tools that the cloud provider has available and there has not been enough time to implement some tools outside the cloud provider.

Others, such as the **Vault** [11] secret manager, could be deployed with a small proof of concept that serves as an example of how to perform secret management within an infrastructure.

5.3. Microservices

When this project began, there was already a relatively solid development of microservices to be able to take advantage to continue the implementation of new microservices that add worth to the final product.

The flexibility of working with the microservices has been noticed as opposed to working with a monolithic type project as it provides a flexible way to replicate certain microservices if required.

5.4. Data analytics

After concluding the project, we have identified many improvable aspects of the data analytics.

Due to the limitation of the selected data source (Packet and Metricbeat) we did not count with good data quality (we think that the selected features provided by those sources were not enough).

After applying the cluster algorithm to the available pre-processed data, we obtained a data model that was tested using the theorical framework previously described. With help of data visualization tools, such as the own developed dashboard, we obtained the following results:

- From the point of view of the available data obtained with Packetbeat, the clustering results indicated that the discrimination factor was the event duration of the flows.
- From the point of view of the available data obtained with Metricbeat and simulating a DOS attack we were able to cluster the anomaly as an outlier, thus, detecting the attack.

6. Future lines

The short-term improvements to be made after the completion of this project will be listed in order to complete the objective rubric.

On the one hand, complete the full automation of Honeypot + Elastic services to be launched with the CloudFormation stack. [10]

On the other hand, to add the secrets found in the code of the applications, configuration files to the Vault.

Extend the functionality of the token creation authentication microservice for JWT with the features offered by the Vault.

Extend Kibana to display elements with the log microservice to know the status of microservices at all times

Make a host hardening to each host deployed in AWS according to the needs of each, create an AMI with the hardening done and use it in subsequent deployments of CloudFormation.

Use AWS services such as WAF, Shield or IDSs from the marketplace to improve the security of the infrastructure, as well as implement the backup policies defined in this document.

It would be advisable to allow a user to create an order with more than just one type of piece. Also, the messaging should be adjusted to allow the capability of replication of the machine service to attend from just one queue.

A new data capturing is recommended due to the lack of results obtained on this project. For example, the Osquery tool could be helpful.

From the point of view of the data analysis we propose using different pre-processing methods (trying different scalers, such as min-max scaler) and different clustering algorithms such as Optics as it might help to select the Epsilon value just by DBScan.

The data visualization proposed is incomplete. An extended dashboard would allow the user to obtain more information about the clustering result such as the standard deviation as it might be an indicator to understand why the cluster was labelled that way. We think that features that have a low standard deviation might be the reason of the concrete clustering.

7. Annexes

Annex A: Gantt diagram

The planning details of the project in the form of a Gantt chart explained briefly by different phases. The duration has been one month from December 2019 to January 2020.

Annex B: AWS Pricing

This document details the cost of the operating machines used in the cloud provider as a forecast of moderate cost usage based on needs.

Annex C: Source Code

Where used code to develop this project is allocated.

Annex D: Microservice development A

This first document specifies the development needed to convert the monolithic project into different microservices, defining different aspects such as messaging or data integrity.

Annex E: Microservice development B

This document describes the different implementations such as security, service discovery and health check of microservices.

8. Bibliography

[1] "Elasticsearch: El motor de búsqueda y analítica distribuido oficial | Elastic." [Online]. Available: https://www.elastic.co/es/products/elasticsearch. [Accessed: 24-Jan-2020].

[2] "Kibana: Explora, visualiza y descubre datos | Elastic." [Online]. Available: https://www.elastic.co/es/products/kibana. [Accessed: 24-Jan-2020].

[3] "Logstash: Recopila, parsea y transforma logs | Elastic." [Online]. Available: https://www.elastic.co/es/products/logstash. [Accessed: 24-Jan-2020].

[4] "Metricbeat: Lightweight Shipper for Metrics | Elastic." [Online]. Available: https://www.elastic.co/es/products/beats/metricbeat. [Accessed: 24-Jan-2020].

[5] "Packetbeat: Network Analytics Using Elasticsearch | Elastic." [Online]. Available: https://www.elastic.co/es/products/beats/packetbeat. [Accessed: 24-Jan-2020].

[6] "cowrie/cowrie: Cowrie SSH/Telnet Honeypot http://cowrie.readthedocs.io." [Online]. Available: https://github.com/cowrie/cowrie. [Accessed: 24-Jan-2020].

[7] "DinoTools/dionaea: Home of the dionaea honeypot." [Online]. Available: https://github.com/DinoTools/dionaea. [Accessed: 24-Jan-2020].

[8] "scikit-learn: machine learning in Python — scikit-learn 0.22.1 documentation." [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 24-Jan-2020].

[9] "Modern Analytic Apps for the Enterprise - Plotly." [Online]. Available: https://plot.ly/. [Accessed: 24-Jan-2020].

[10] D. Keim, G. Andrienko, J. D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual analytics: Definition, process, and challenges," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 4950 LNCS, pp. 154–175.

[11] "Vault by HashiCorp." [Online]. Available: https://www.vaultproject.io/. [Accessed: 24-Jan-2020].