# CVC World of Services

Ruben Vera Garcia

June 28, 2021

**Resumen–** En los últimos años, la tendencia global del mercado y las empresas tecnológicas es el de ofrecer Software como servicio alojado en la nube. Esto permite a una organización poder ejecutar y poner en marcha aplicaciones disponibles en línea y bajo demanda para sus clientes con un coste inicial reducido. Por lo tanto, este proyecto pretende conseguir desarrollar una plataforma para almacenar estos servicios y una metodología para estandarizar la subida del código de cada servicio. Por último, se definen posibles opciones para implementar el acceso del cliente a estos servicios en línea.

**Palabras clave–** Software como servicio, Nube, Estandarizar, Bajo Demanda, Dockerfile, Dockerizar, S3, ECR, ECS, EC2.

**Abstract–** In recent years, the global trend of the market and technology companies is to offer Software as a Service hosted in the Cloud. This enables an organization to be able to run and deploy applications available online and on-demand for its customers with a reduced initial cost. Therefore, this project aims to develop a platform to store these services and a methodology to standardize the uploading of the code for each service. In last instance, possible options are defined to implement customer access to these services online.

**Keywords–** Software as a Service, Cloud, On Demand, Dockerfile, Dockerize, S3, ECR, ECS, EC2.

## 1 Introduction - Project framework

NOWADAYS, almost every company has the necessity of contracting specific services to third companies. In order to delegate work satisfactorily, customers need access to these services at any time and on-demand. Therefore, the approach selected was to create a Software as a Service, from now on SaaS, platform.

Nevertheless, the code developed for each service needs to be encapsulated to ensure standardized access. That is why CVC World of Services is focused on creating a digital platform with every service saved and ready to deploy for customers interested. Consequently, having this platform will grant the Computer Vision Center the opportunity of uploading all its services in a repository. Moreover, CVC customers will have access to any service available on demand.

• Contact email: ruben.vera@e-campus.uab.cat
• Mention made: Computing
• Tutored work by: Daniel Franco Puntes, Department of Computer Architecture and Operating Systems
• Course 2020/21

To clarify the concept of service in CVC WoS, an example would be a convolutional neural network that extracts what is inside from any image. As a result of uploading an image or a set of images, the customer will be able to download a document with the text extracted from the image.[1]

## 2 Scope of the project - Objectives

With the context of the work provided, this section is focused on the necessity of further clarifying the objectives of the project. The objectives are as follows:

- Create a functional local and Cloud service to upload it ultimately to the platform.
- Define a functional platform executed in the Cloud, without having dependencies with machines inside the CVC, that ensures proper maintenance of a repository that can store a wide variety of services.
- Display the services to specific customers of the platform. Moreover, establish an approach, so customers

can upload their queries plus download the responses from these queries.

- Create a scalable platform, in other words, grant the functionality to administrator users to upload new services in a straightforward way. Administrators will have the power to create, read, update and delete services.

- Create a great marketing tool, with an interface user-friendly with customers.

In order to accomplish these objectives, it is necessary to go deeper into the subject and define a basic architecture of the service, shown in figure **1**, and establish the different entities who can interact or access a service:

- Firstly, there are the CVC researchers. They develop the services to which customers will have access. When the service is totally prepared, the researcher requests the upload of the service to the administrator user.

- The administrator user can create, read, update and delete, CRUD, the services in the platform.[2] This is the role on which this project is based. The done work is focused on receiving services with no uniform code, encapsulate the service code and upload these services for standardized access of the customers in the platform. Furthermore, the administrator has to ensure correct maintenance of the services and grant access to customers to specific services.

- The last entity is the customers. This group requests the creation of specific services to the researchers or demands access to already created services to the administrator user.

## 3 State of the art

The main focus of the project is to create a SaaS platform. In order to build a proper SaaS, the software has to be accessible via the Internet, without the need of downloading it onto your device because the software runs on a Cloud server.[3] Reviewing different alternatives to host the project platform SaaS, Amazon Web Services, AWS, was the one selected due to the fact of having the better variety on free-tier services plus it has the AWS educate system with free credits to employ.

For the last years, SaaS has been growing due to the fact it offers flexibility to customers with services on-demand, customers pay for what they want to utilize, and always accessible from the Internet. In addition, the Covid-19 pandemic has also boosted the increase on these types of services and companies have been transforming their business model to a pure SaaS. Figure **2** reinforces this premise because from 2019 to 2020, it can be seen a bigger growth than in the previous year.

After reviewing several SaaS companies such as Netflix, Dropbox or Prime Video, there was the necessity of creating a simple but yet describing logo of the project. Afterward creating initial versions of the logo, the final logo was designed and created, the logo is shown in figure **3**.

## 4 Methodology and Planning

As far as the methodology is concerned, the team of the project decided to follow the Agile methodology, Scrum. That is why, during the project, the team has been utilizing the application Microsoft Teams as the main communication channel.

Microsoft Teams offers to create a group with different users. The users of the team can schedule periodic meetings, sprint meetings, have conversations with other members of the group, create a repository section to upload files and have a development tab to organize the tasks with the methodology Scrum.[4] Inside this tab, the team can organize the tasks in different sections:

- Product Backlog: In this section, there are the tasks that will be done in the future, but not in the next sprint. An example would be writing the final report and the project is in an early stage of development.

- Sprint Backlog: After every sprint meeting, the team inserts the tasks to do during the following sprint.

- Doing: When moving the tasks from sprint backlog to doing, the team has started the development of the task.

- Done: This section includes every task completely finished.

In addition to Microsoft Teams, the team has also been communicating via email when there exist questions or doubts.

As far as the planning refers, the detailed planning of the project has been followed correctly with minor fixes during the development. Figure **4** shows a Gantt diagram with an explanation of every sprint, which each one of them has the duration of one week, plus mandatory deliveries of the TFG.

Taking into account that each sprint has distinct tasks and work done, the project can be split up into four sections, so it is easier to explain what has been developed:

- Section 1: From sprint 1 to writing the initial report, the project is focused on executing preliminary investigations plus starting to define what should be done.

- Section 2: From the initial report to the first follow-up report, it finished defining what to perform in the project plus the first version of a functional service was carried out locally and in the Cloud.

- Section 3: From the first to the second follow-up report, the work done was redefining the problem to be more realistic, plus creating a simple service to dockerize it and uploaded it to the project SaaS platform defined as a repository.

- Section 4: Finally, from the second follow-up report to the final report, the service was improved, the user interface to access the service was defined, and the SaaS platform was completely defined as a repository of services.

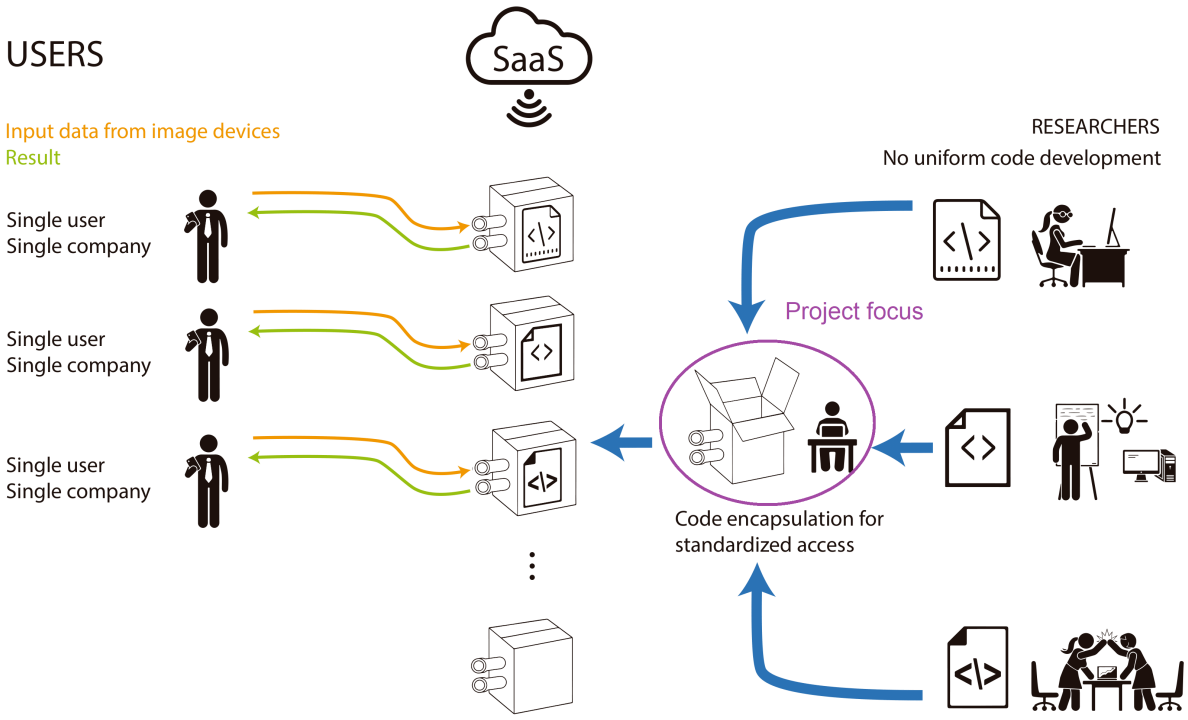


Fig. 1: Basic architecture of the service.

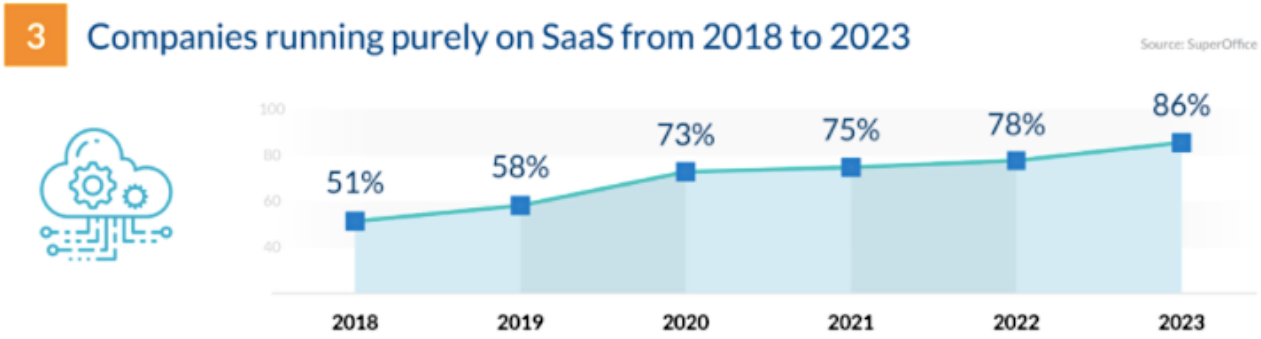


Fig. 2: Companies running purely on Saas from 2018 to 2023. Source: SuperOffice.[3]


Fig. 3: Logo of CVC World of Services, CVC WoS.

## 5 Design

### 5.1 Designing the local service

Before uploading a service to the SaaS platform in the AWS Cloud, there is the necessity of designing the concept of the service which will be utilized onwards. Moreover, the service needs to be a realistic example of an application that a researcher could send to an administrator user in order to upload it to the Cloud platform.

Therefore, the final version of the service chosen is a convolutional neural network, a ResNet50 network[5], which accepts images with the extensions *.jpg, .png, .jpeg and .gif* and returns a text file with various predictions of what could be inside the image. These predictions are made up of two parts, the prediction label and the percentage of how probable the label is for that image. In figure **5** is shown the concept of the network providing different predictions to an image.

| | Nombre | Comienzo | Fin |
|---|---|---|---|
| 1 | **Image Analysis with Deep Learning Techniques in the Cloud** | **jue 11/02/21** | **vie 09/07/21** |
| 2 | Sprint 1: Preliminar investigation | jue 11/02/21 | jue 18/02/21 |
| 3 | Sprint 2: Initial definition of the project | jue 18/02/21 | jue 25/02/21 |
| 4 | Sprint 3: Initial definition of the platform | jue 25/02/21 | jue 04/03/21 |
| 5 | Sprint 4: 2nd version definition of the platform | jue 04/03/21 | jue 11/03/21 |
| 6 | Write initial report | lun 01/03/21 | dom 14/03/21 |
| 7 | Sprint 5: Initial definition of a service | jue 11/03/21 | jue 18/03/21 |
| 8 | Sprint 6: 2nd version definition of a service | jue 18/03/21 | jue 25/03/21 |
| 9 | Sprint 7: Local implementation of a service | jue 25/03/21 | jue 01/04/21 |
| 10 | Sprint 8: Cloud implementation of a service | jue 01/04/21 | jue 08/04/21 |
| 11 | Sprint 9: Local implementation of a dockerized service | jue 08/04/21 | jue 15/04/21 |
| 12 | Sprint 10: Local and Cloud implementation of a dockerized service | jue 15/04/21 | jue 22/04/21 |
| 13 | Write follow-up report I | lun 19/04/21 | dom 25/04/21 |
| 14 | Sprint 11: Functional local and Cloud implementation of a dockerized service | jue 22/04/21 | jue 29/04/21 |
| 15 | Sprint 12: Initial version of the platform and service structure in the Cloud | jue 29/04/21 | jue 06/05/21 |
| 16 | Sprint 13: 2nd version of the platform and service structure in the Cloud | jue 06/05/21 | jue 13/05/21 |
| 17 | Sprint 14: Initial connection of the service with the platform | jue 13/05/21 | jue 20/05/21 |
| 18 | Sprint 15: 2nd version of the connection of the service with the platform | jue 20/05/21 | jue 27/05/21 |
| 19 | Write follow-up report II | lun 17/05/21 | dom 30/05/21 |
| 20 | Sprint 16: Improve service interface, solve connection issues and define local volumes | jue 27/05/21 | jue 03/06/21 |
| 21 | Sprint 17: Improve service in the Cloud plus creation of final report | jue 03/06/21 | jue 10/06/21 |
| 22 | Sprint 18: Testing the platform and services | jue 10/06/21 | jue 17/06/21 |
| 23 | Write final report | lun 07/06/21 | dom 20/06/21 |
| 24 | Create oral presentation | lun 14/06/21 | dom 27/06/21 |
| 25 | Create and organize TFG dossier | lun 31/05/21 | lun 28/06/21 |
| 26 | Create poster (optional) | lun 28/06/21 | dom 04/07/21 |
| 27 | Presentation and public debate of TFG | lun 05/07/21 | vie 09/07/21 |

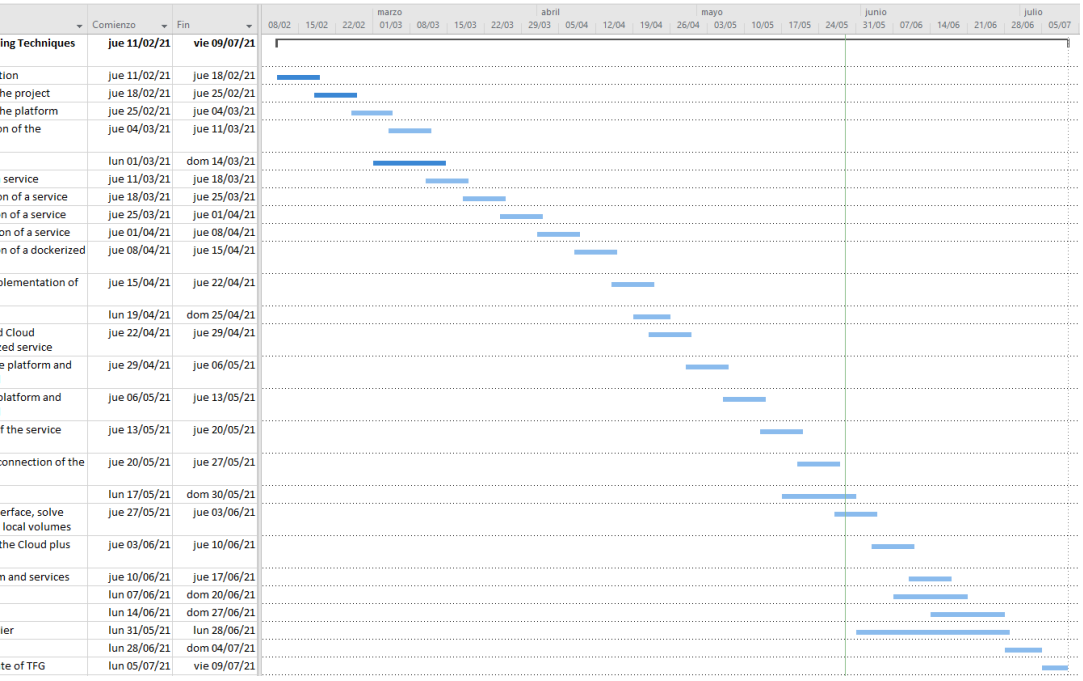


Fig. 4: Gantt diagram with the planning described.

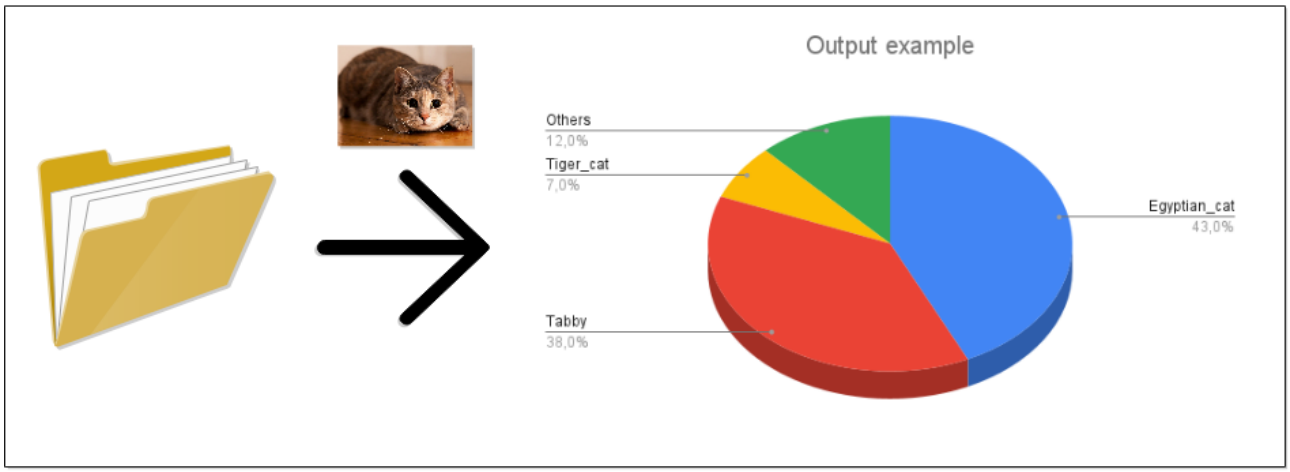


Fig. 5: Example of a prediction of the network.

Furthermore, the service is keeping an eye on a specific local folder and every 5 seconds creates a log of whether the folder has been changed or not. An example of a possible logs output is shown in figure **6**. In this example, there is a total of three logs:

- In the first four lines, the service detected two images added in the folder, *gato.jpg* and *Mario.png*.
- In the following two lines, the service created and detected the *request_resolution.txt* file in which there is the previous response. Because there are no further added images, the application does not change the *request_resolution.txt* file.
- Nothing changed!

Even though in the last example there were two different types of logs, files added and nothing changed, the code is fully prepared for four possible events:

- Files removed: In this case, the code eliminates the removed files, updating what is left in the folder.
- Files added: For this option, the service checks if the added files are images. The files which pass this test are sent to the network in order to predict what is inside plus finally uploading the response to the text file. As can be seen in the first four lines of figure **6**, the response is in JSON format
- Files removed and added: As the customer can remove and add files during the five seconds cooldown, this option is also possible.
- Nothing changed: During the five seconds, nothing changed inside the folder.

## 5.2 Designing the Cloud service

Having the local service designed, the next step was to adapt the application in order to be Cloud based. With this

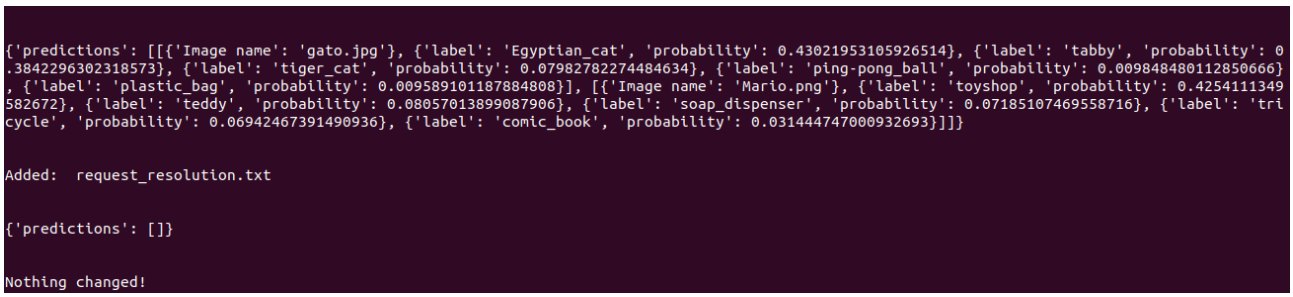


Fig. 6: Example of a logs output of the service.

change, the shared folder system is no longer available, and it needs to be changed with a Cloud solution. After much deliberation and tests, the chosen option was to adapt the code, so the application is keeping an eye on the changes inside an S3 bucket, which is an online object storage service of AWS,[6] instead of a local folder.

Therefore, the customer will have total access to the bucket to upload the images wanted and download the *request_resolution.txt* file with the response from the service as shown in figure **7**.

Consequently, the critical changes designed to the local service to evolve it into a Cloud service were the following ones:

- There is no necessity of using Docker volumes, explained later on, because the connection is no longer between folders, but between the container folder and the S3 bucket.
- Modify the code application with the Boto3 library[7] to download the images to a folder from the container and upload the response *request_resolution.txt* file with the predictions to the S3 bucket shared with the customer.

### 5.3 Life cycle of the service

To conclude the design section, in figure **8** there is a resume of the life cycle that a service has to undergo.

Firstly, in version 1, the administrator user receives a service from a researcher. The service is dockerized and tested in a local environment. However, what means dockerizing an application?

It means encapsulating the service in a Docker container. A Docker container is a process isolated from the rest of the processes of a machine.

This technology is perfect for building a SaaS platform. As shown in figure **9**, the containers can run applications in parallel plus are secure, isolated from each other, and lightweight with their own operating system independent of the host machine.

Secondly, in version 2, when the service is fully operational locally, the administrator user alters the service, so the storage and processes are made inside the Cloud and the customer has access to it.

Thirdly, in version 3, the administrator user evolves the encapsulation code in order to push it to a Cloud repository, described in the implementation section.

Lastly, the service is stored in the Cloud repository and ready to be executed.

## 6 Implementation

### 6.1 Code encapsulation and standardized access for local service

Having the fully operational local service from the researcher, the first step is to dockerize it. With the Docker technology selected, in order to dockerize the service, it is needed to create the Docker image plus run the image to create the container. A Docker image is a just-read template that has the complete set of instructions for creating the container, this template is called Dockerfile.[8] In figure **10**, there is the Dockerfile that consists of the following lines of code:

- Line 1: The Docker image is created from the specific image Python 3.8-slim-buster. It was selected to reduce the storage needed to build the image and solve issues with old library versions.
- Line 3: It creates the directory */images-saas-1* inside the Docker container.
- Line 5: It specifies the working directory.
- Line 7: Copies all the files from where the Dockerfile is executed to the container.
- Line 9: These commands build all the libraries necessary to execute the service.
- Line 11: It executes the Python file *app.py* with Python 3 instead of Python 2.

After creating the Dockerfile, it is necessary to build the Docker image as shown in figure **11**[9] and, in last instance, finish with running the container as shown in figure **12**.[10]

An important aspect to take into account is the fact that, as said, containers are completely isolated, so the last step to have the local service operable was to connect the folder in which the customer uploads images in the host to the folder inside the isolated memory of the container in which the service is keeping an eye of the changes made with Docker volumes.[11]

### 6.2 SaaS platform and the Cloud service

Having performed the proposed changes in the design section to the code of the service and Dockerfile. In order to be able to perform different tests to accomplish the Cloud service, AWS has the option of creating an AWS educate account. This type of accounts has reduced permissions and

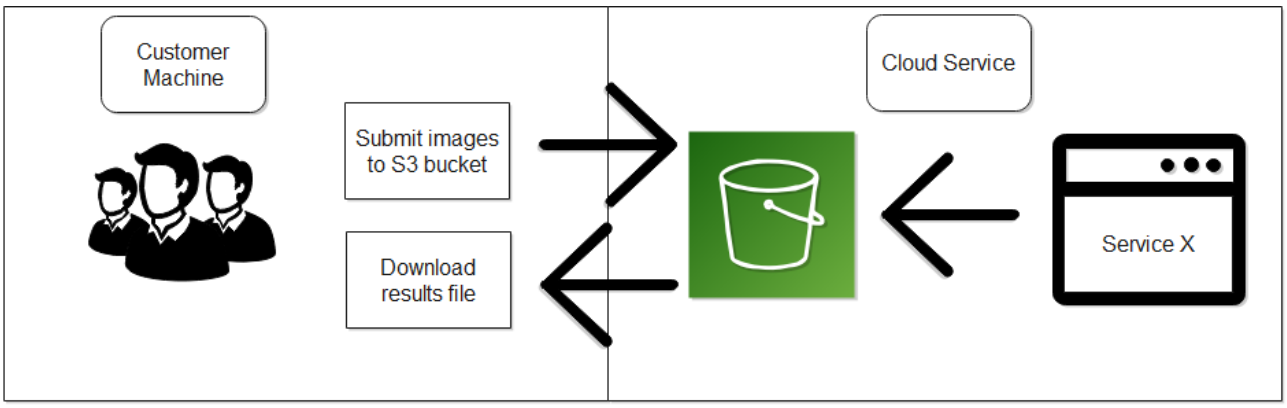


Fig. 7: Complete architecture of the Cloud service.


Fig. 8: Life cycle of a service.

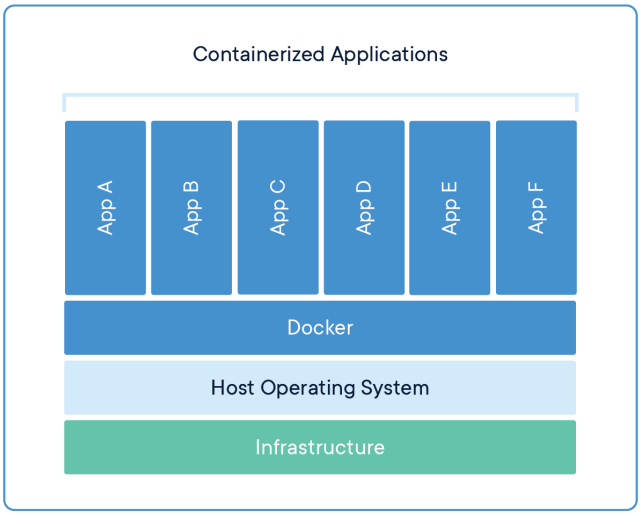


Fig. 9: Scheme of executing Docker containers.

```
1 FROM python:3.8-slim-buster
2
3 RUN mkdir -p /images-saas-1
4
5 WORKDIR /
6
7 COPY . .
8
9 RUN pip3 install boto3 tensorflow keras numpy Pillow
10
11 CMD ["python3", "./app.py"]
```

Fig. 10: Dockerfile to create the local service.

```
rubi@rubi-VirtualBox:~/Documentos/share$ sudo docker build --tag share .
```

Fig. 11: Creating the Docker image of the local service.

```
rubi@rubi-VirtualBox:~/Documentos/share$ sudo docker run -v /media/sf_Shared_file:
/media/sf_Shared_file share
```

Fig. 12: Creating the Docker container of the local service.

does not have full access to all the AWS services. However, there is a wide variety of free tier services which can be used. However, there are other services that Vocareum accounts can not access and one of them is creating IAM users. Thus, the administrator user has to utilize his own user credentials, which changes every 2 or 3 hours. This fact made every solution executed only operable for the time that the credentials are not changed.

With that in mind, the next step is to push the service to the SaaS platform, a repository of services. Investigating through the services of AWS, the selected service to create the project SaaS platform was AWS ECR, Elastic Container Registry,[12] which is a fully managed container repository

that makes it easier to store, manage, share, and deploy container images. Due to the fact of using an AWS educate account to upload a Docker image to AWS ECR, the third step of login to ECR[13] can only be performed if you are uploading the Docker image from an EC2 instance of that account. Hence, with a *scp* command, a zip file with all the necessary files is transferred to the EC2 instance, and lastly, from the localhost we connect to the EC2 instance with a *ssh* command. From this point, there is only the necessity of installing Docker in the instance[14] and execute all the steps to push an image to AWS ECR. In figure **13**, the image can be seen pushed at AWS Elastic Container Registry.

Deploying the services as Docker images in AWS ECR enables us to deploy them to AWS ECS[15], Elastic Container Service, running any quantity of containers necessary in AWS EC2 instances[16], without thinking in the orchestration of the containers. This scheme is described in figure **14** with a higher level of detail.[17]

Having done these steps correctly, if we access to AWS Elastic Container Service and wait from 3 to 5 minutes, the image pushed will be running in a number of containers specified earlier. As can be seen in figure **15**, for this test there will be only one EC2 instance at the same time, because there is no need for more and in order to stay in the free-tier services.

The last important aspect to describe is the client machine, the user interface, how a customer can access the S3 bucket to upload images and download the responses from the server. There are two different options designed and executed:

- Option 1: Utilize an Open-Source software called Filestash which allows us to create an editor link connected to a specific S3 bucket as shown in figure **16**, there exists also an option to limit the time of the link being operable.[18]
  - Advantages: More intuitive to employ for the customer.
  - Disadvantages: It is slower than the second option because you have to enter to the website from an Internet browser. Furthermore, the customer has to upload and download from the website.
- Option 2: Using a Python application that connects a customer folder with the S3 bucket, as shown in figure **17**. The customer only needs to change its folder to modify the S3 bucket and receive the *.txt* file response.
  - Advantages: Changes made directly from a local folder. Consequently, it is the fastest option.
  - Disadvantages: It needs to execute a Python file, so the customer needs basic knowledge in Python. The last disadvantage is that the AWS credentials are exposed to be discovered inside the Python file, so in a production level it should not be an option.

## 7 Results and conclusions

This project started from creating a platform with different services online to standardize the process of dockerizing a service and push it to AWS, so the customers can have access on-demand and with no risk of the service being down or lose the uploaded data.

Given the project summary, the four sections described in methodology and planning have been achieved correctly. Moreover, accomplishing these sections ended with a Minimum Viable Product, MVP, of a SaaS platform with a functional service pushed in the repository.

However, before uploading the service to the Cloud, making different versions of local services have granted me deeper knowledge in working with Ubuntu and Docker. During the degree, these technologies, especially Docker, were studied infrequently and this TFG has helped to understand and be more capable of utilizing them. I believe that Docker is a technology that combines perfectly with Cloud Services and SaaS platforms.

Secondly, this TFG is also focused on the emerging technology platform of Amazon Web Services. During the degree, Amazon Web Services was not even mentioned once in the subjects. Having in mind that this platform is increasing each year and becoming more and more relevant, I believe that working in transforming local services into Cloud services plus push them to the Amazon Cloud is an essential part of the project. Nevertheless, facing a technology for the first time, it usually needs an adaptation period to learn and improve the basics, and in the case of Amazon Web Services with its wide variety of services available, it is quite challenging to choose what services to employ and keep an eye on the budget that decreases with every move made.

Thirdly, an important aspect that is usually overlooked by students is that a project is not only the development section and writing code. Defining what your project is and why it is useful for society are as important or even more than developing the project. For computer scientists, the first time they perform this type of work, it becomes a challenge because they are not used to doing it. Therefore, defining these aspects of this TFG was challenging, but it granted an important view.

Finally, this project has been transforming and evolving from sprint to sprint in order to adapt to the limited time and to express the problematics and objectives of this TFG. That is why I believe that defining this project from the start and stick with that definition is almost impossible and not recommendable because of the project nature which in each sprint meeting is constantly mutating and fluctuating.

## 8 Acknowledgements

I would like to thank my project tutors, Daniel Franco and Antonio Espinosa, for their weekly support during the project and the facilities given to learn emerging technologies. I would also like to thank Daniel Franco for the feedback made with the documentation delivered.

Lastly, I would like to thank Coen Antens for lending me the opportunity of connecting my TFG with a CVC problematic and for providing direct contact support with AWS workers.

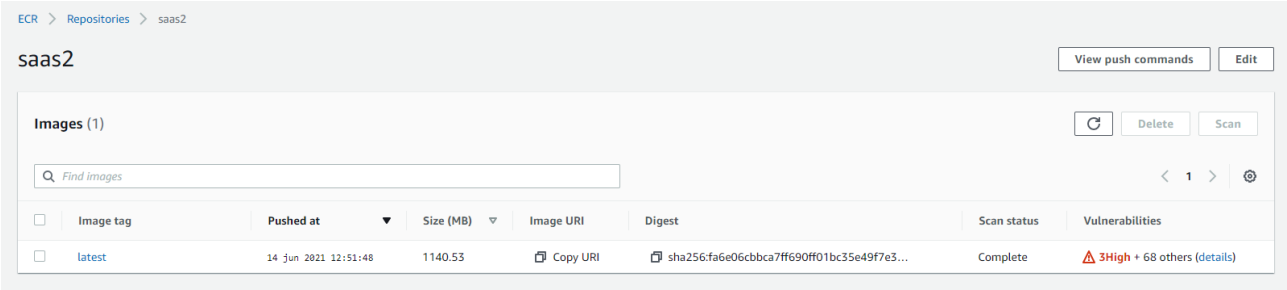


Fig. 13: Image pushed to AWS ECR.

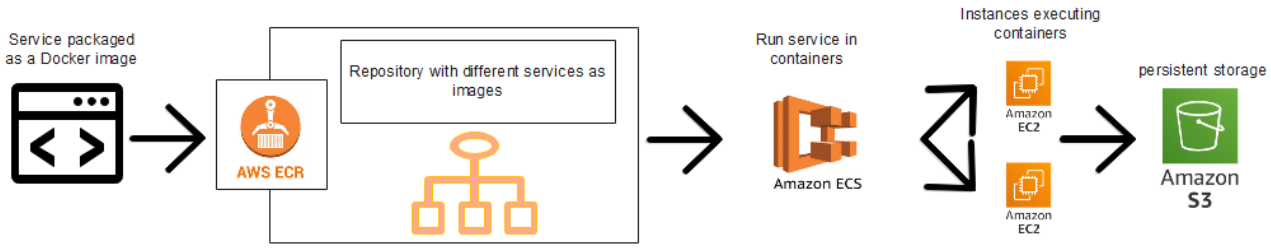


Fig. 14: Repository of services architecture.

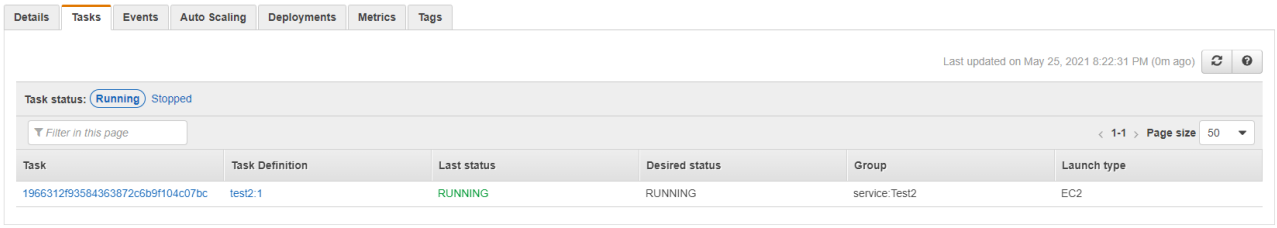


Fig. 15: Accessing to AWS ECS.

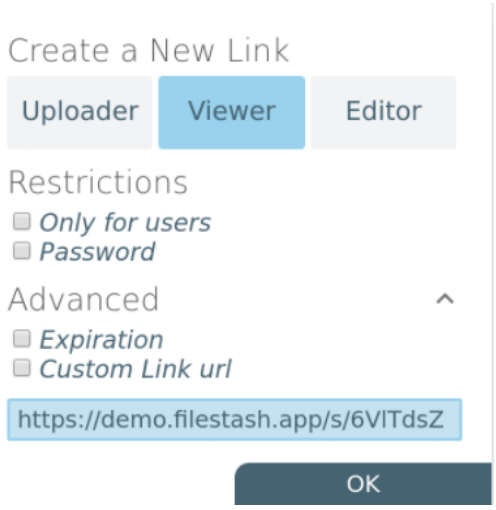


Fig. 16: Open-Source Software Filestash.

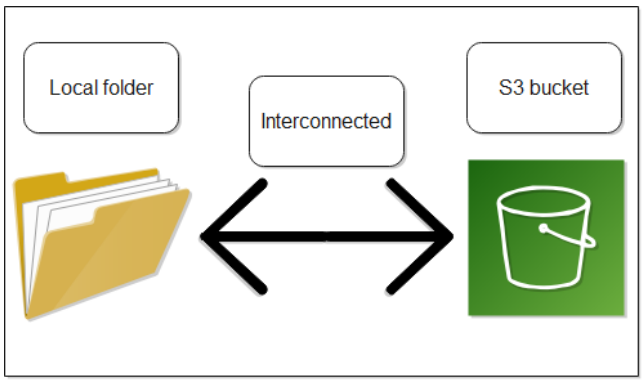


Fig. 17: Customer machine architecture.

## REFERENCES


[1] Amazon Textract, services for OCR (2021). Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://aws.amazon.com/textract/?nc1=h_ls

[2] What is CRUD?. Codeacademy. [Accessed: June 28, 2021]. Available on the Internet: https://www.codecademy.com/articles/what-is-crud

[3] Is Netflix a SaaS? 25 Examples of SaaS Companies that Are Rocking It. Single Grain. [Accessed: June 28, 2021]. Available on the Internet: https://www.singlegrain.com/saas/examples-of-saas-companies/

[4] What is Scrum?. Scrum.org. [Accessed: June 28, 2021]. Available on the Internet: https://www.scrum.org/resources/what-is-scrum

[5] TensorFlow Core v2.5.0 ResNet50. TensorFlow. [Accessed: June 28, 2021]. Available on the Internet: https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50

[6] Amazon S3. Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://aws.amazon.com/s3/?nc1=h_ls

[7] Boto 3 Docs 1.9.42 documentation. Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://boto3.amazonaws.com/v1/documentation/api/1.9.42/reference/services/s3.html

[8] Dockerize your Python Application. Runnable. [Accessed: June 28, 2021]. Available on the Internet: https://runnable.com/docker/python/dockerize-your-python-application

[9] Building your Python Image. Docker Inc. [Accessed: June 28, 2021]. Available on the Internet: https://docs.docker.com/language/python/build-images/

[10] Run your image as a container. Docker Inc. [Accessed: June 28, 2021]. Available on the Internet: https://docs.docker.com/language/python/run-containers/

[11] Docker Volumes explained in 6 minutes. TechWorld with Nana, YouTube. [Accessed: June 28, 2021]. Available on the Internet: https://youtu.be/p2PH_YPCsis

[12] Amazon ECR. Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://aws.amazon.com/ecr/?nc1=h_ls

[13] Commands guide to upload an image to ECR. GitHub. [Accessed: June 28, 2021]. Available on the Internet: https://gist.github.com/awssimplified/da49577fa48128e1da992dd6ec21085c

[14] Docker basics for Amazon ECS. Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

[15] Amazon ECS. Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://aws.amazon.com/ecs/?nc1=h_ls

[16] Amazon EC2. Amazon Web Services. [Accessed: June 28, 2021]. Available on the Internet: https://aws.amazon.com/ec2/?nc1

[17] How to deploy an application to AWS using Docker, ECS, and ECR. Medium. [Accessed: June 28, 2021]. Available on the Internet: https://tinyurl.com/3x3za9b6

[18] AWS S3 Explorer. Filestash. [Accessed: June 28, 2021]. Available on the Internet: https://www.filestash.app/aws-s3-explorer.html