Improving Amazon Pipelines Efficiency for Web Applications

Roger Piera Fernandez de Simon

Resum– Las aplicaciones web que usamos en nuestro día a día funcionan gracias a toda una infraestructura que permite que las podamos usar. En este proyecto analizaremos y mejoraremos la infraestructura que usamos en algunas de las aplicaciones web que usamos en Idneo. Estas aplicaciones usan Amazon Web Services para mantenerse. Explicaremos los varios servicios de AWS usados para dar soporte a las aplicaciones web y encontraremos formas de mejorarlos para reducir costes y mejorar la calidad de esta infraestructura.

Paraules clau– ECS, EC2, Amazon AWS, ECR, CDK, Code Build, Pipeline, S3 buckets, Jmeter, Jenkins

Abstract– The web applications we use in our day to day are supported by an infrastucture to make them avaliable for our use. In this project we will analyse and improve upon the infrastucture that supports some of the web applications used in Idneo. In this project the applications are hosted using Amazon Web Services. AWS will be the main source of infrastucture used to support our applications. We will explain and compare the various services from AWS used to support our applications and find out ways to improve the currently used systems to reduce costs and improve the quality of the infrastucure.

Keywords– ECS, EC2, Amazon AWS, ECR, CDK, Code Build, Pipeline, S3 buckets, Jmeter, Jenkins

✦

1 INTRODUCTION

IDNEO [29] is a global engineering company providing the full value chain to create innovative and technological products and services to support customers to differentiate and step-up their products line-up. It is a company with more than 200 workers and multiple offices accross the world. Having offices in USA California, Barcelona and Malaga.

The Devops [30] team of Idneo [29] is tasked with maintaining and improving the infrastructure that supports all the applications and providing tools to developers to automate their tasks and improve productivity. This project comes as a task assigned to the Devops team which the author of this paper is part off.

In Idneo [29] we have some of our applications running on a pipeline that scales vertically [1] and takes a long time to compile. Furthermore, every change to an application requires that all the other applications be compiled again. This causes a lot of lost time during development and bug fixing of the applications.

Every new application added will compound this problem and it will also have the added negative of requiring a bigger machine to run all the applications which can be very expensive [2].

The motivation of this project is to resolve these issues and make the pipeline scale horizontally [1] and be more efficient to make development of new applications easier.

2 OBJECTIVES

The overarching objective of this project is to migrate some of the currently used applications in Idneo to a new pipeline to reduce build times, costs, and improve scalability.

To be able to accomplish this objective, some more especific objectives must be achived. This objectives are:

- **Data Gathering:** Data will be gathered to find build and deployment times, costs and scalability of the currently used pipeline. In order to accomplish this an

- Contact e-mail: roger.piera@e-campus.uab.cat
- Menció realitzada: Tecnologies de la Informació
- Tutored work by: Ana Oropesa Física (departament)
- Course Year: 2020/21

analysis will be done of the current system using tools provided by Amazon and other third party tools like Jmeter. This analysis will be performed during the preliminary phase of the project.

- **Technology Selection:** The next goal will be to find out the best technology to solve the bottlenecks shown by the previously mentioned analysis. The main technologies that will be studied for this project are S3 buckets [3], ECS [4] and Kubernetes [5]. In order to select the best technology pricing, difficulty to implement and efficiency will be analysed. This will happen during the preliminary phase.
- **Pipeline Adoption:** The next goal will be for Idneo to adopt the new pipeline and migrate all its applications there using the previously selected technology. This will happen during the development phase.
- **Result Analysis:** The last goal will be for the pipeline to pass the requirements of being faster and more scalable than the previous pipeline. To find out, a stress test will be done to it. This phase will happen during the pipeline analysis phase.

3 Methodology

The methodology used for this project will be represented using a Gantt [7] chart. The chronology for this project has been divided into weeks, each week consisting of 5 work days. The tasks have been divided in groups and each task has a start and end date. For each task there are a few days of leeway in case it's not finished on time.

The task groups, as can be seen in the Gantt chart in Appendix 1, are the following:

- **Preliminary Phase:** In the preliminary phase all the relevant information of the currently used pipeline will be gathered. A diagram will be drawn for ease of understanding. In this phase research will be done about potential technologies to improve the pipeline with a focus on ECS [4]. of the current system. This analysis will be performed during the preliminary phase of the project.
- **Development Phase:** In the Development phase the new pipeline will be built and the currently used applications will be migrated there.
- **Pipeline Analysis Phase:** In the Pipeline analysis phase the new pipeline will be compared with the old one.
- **Optional Improvements Phase:** In the optional improvements phase any spare time will be used to make additional adjustments to further improve on the pipeline. A potential one being using CDK [6] code.
- **Documentation and Presentation Phase:** In the documentation and presentation phase all the documentation for this project will be finalized. The presentation will be prepared and the poster created.

4 Amazon Web Services Concepts

In this section we will introduce Amazon Web Services (AWS) [35] explaining what they offer and giving an overview of some of the services we will talk about through this project.

4.1 What is AWS

Amazon Web Services [35] is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

AWS has a lot of different services, and a lot of features within those services From infrastructure technologies like compute, storage, and databases–to emerging technologies, such as machine learning and artificial intelligence, data lakes and analytics, and Internet of Things.

4.2 What are some of the services that AWS offers

Amazon Web Services has many services, in this project we will only contemplate a small part of them and end up using a smaller part.

Some of the services we will mention during this project are:

- **Code Pipeline [11]:** A pipeline, also known as a data pipeline, is a set of data processing elements connected in series, where the output of one element is the input of the next one. AWS CodePipeline is a fully managed continuous delivery service that helps us automate our release pipelines for fast and reliable application and infrastructure updates. CodePipeline automates the build, test, and deploy phases. It has integration with third party services like Github or Jenkins.
- **Code Build [13]:** AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy.
- **ECR [14]:** Amazon Elastic Container Registry (ECR) is a fully managed container registry that makes it easy to store, manage, share, and deploy our container images and artifacts anywhere. Amazon ECR eliminates the need to operate our own container repositories or worry about scaling the underlying infrastructure.
- **EC2 [2]:** Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud.
- **Load Balancers [17]:** Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses.
- **Route 53 [16]:** Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost effective way to route end users to Internet applications by translating

names like www.example.com into the numeric IP addresses like 192.0.2.1 that computers use to connect to each other.

- **Target Groups[18]:** Target Groups are used in conjunction with Load Balancers. Each target group is used to route requests to one or more registered targets. When you create each listener rule, you specify a target group and conditions. When a rule condition is met, traffic is forwarded to the corresponding target group. You can create different target groups for different types of requests.
- **Security Groups[33]:** A security group acts as a virtual firewall for our instance to control inbound and outbound traffic.
- **VPC[34]:** Amazon Virtual Private Cloud (Amazon VPC) is a service that lets us launch AWS resources in a logically isolated virtual network that we define. We have complete control over your virtual networking environment, including selection of our own IP address range, creation of subnets, and configuration of route tables and network gateways.

We will also talk about ECS [4] and S3 buckets [3], but those services will be explained later in this project.

5 Current State of the pipeline

In this section the pipelines being currently used will be explained. The current pipeline has been in use since 2019, three years. This pipeline is used because it allows an automated deployment of applications, which increases the speed of development. The pipelines use EC2 [9] machines to host the websites in this model.

5.1 Pipeline Explained

The currently used pipelines can be seen in detail in Appendix 2. Its functionality will be explained in this section using a simplified diagram.

The pipeline can be divided in two sections. The Preproduction stage and the Production Stage. These two sections mirror each other having the same architecture, but allow for the deployment of applications first in the Prepoduction stage, to test the changes made, and once the testing is done the Production deployment can be done, making the applications available to the general users.

The pipeline functions in the following way:

- Once a new version of the code is ready to be deployed a Jenkins [10] trigger will be activated and the pipeline will start. This Jenkins [10] trigger will download the code from the version control system into a zip file. This zip file is then saved into an S3 bucket [3]. This can be seen as the Step 1 on Fig. 1.
- Afterwards, aws CodePipeline [11] is activated, starting with the CodeSource [12] module that unzips the content of the zip file and sends it to the CodeBuild [13] module. This can be seen as the Step 2 and Step 3 on Fig. 1.

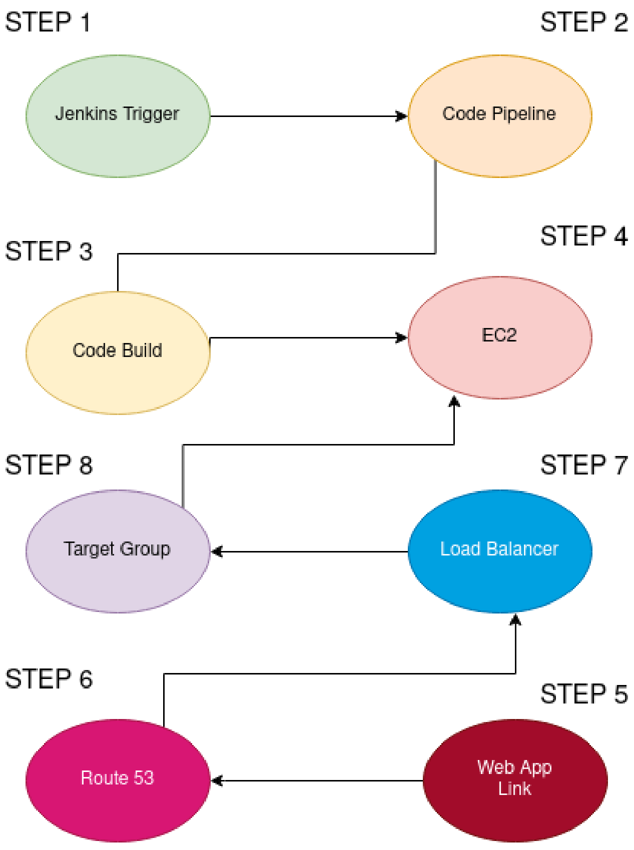


Fig. 1: Simplified EC2 pipelines

- CodeBuild [13] builds the dockerfiles of the applications and pushes the images into aws ECR [14]. CodeBuild also pushes the dockercompose [15] files into the EC2 [9] machine of the applications. This can be seen as the Step 3 on Fig. 1.
- This dockercompose [15] file is activated which downloads the ECR [14] images into the EC2 [9] instance, building the application into it. This can be seen as the Step 4 on Fig. 1.
- On the other end, once the client enters the URL of the application, which is created using aws Route 53 [16], the IP connects to a Load Balancer [17] of the specific application. This can be seen as the Step 5, Step 6 and Step 7 on Fig. 1.
- This Load Balancer [17] connects to a specific port of the EC2 [9] machine using a aws Target Group [18] assigned to it. This port will be open and will have one of the applications assigned to it. This can be seen as the Step 7, Step 8 and Step 4 on Fig. 1.

5.2 Cost and Deployment time of the Pipeline

Considering five applications being supported by this pipeline the cost of the components is the following:

- **Code Pipeline [11]:** 1€/month per pipeline = 2€/month (pre prod and prod pipelines).
- **Code Build [13]:** Negligible.
- **ECR [14]:** Negligible.

- **EC2 [2]:** 20€/month per EC2 instance = 40€/month (pre prod and prod EC2 instance).
- **Load Balancer [17]:** 20€/month per Load Balancer = 200€/month (1 Load Balancer per application, 5 applications in pre prod and 5 in prod = 10 Load Balancers).

Total Cost: 242€/month.
Time required to deploy a new version of an application: 17 minutes.

5.3 Weak Points of the Pipeline

The current pipelines presents a series of problems that this work aims to address.

First of all is the method of scaling of the current pipelines. Currently if more applications are to be added two new Load Balancers [17] have to be added as well. These Load Balancers are expensive [19], each costing 20€ / month. Furthermore, a bigger EC2 [2] machine would need to be created to support more applications, bringing up the cost of it as well.

Another issue is the time to build the applications. Each time a change is made to one of the applications all of them need to be rebuilt since they share the same EC2 [9] instance. This implies currently 17 minutes of build time every time a new change is launched.

EC2 [9] instances have a fixed costs while they are functioning, this means that if the traffic ever slowed down the cost of maintaining the instance would still be the same. This presents a problem since traffic won't always be the same and the pipeline needs to adapt to keep the costs down.

6 Potential Solutions

In this section potential alternatives to the current pipelines that aim to solve the issued presented previously will be analyzed.

6.1 Amazon Elastic Container Service

Amazon Elastic Container Service (Amazon ECS) [4] is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster.

The use of ECS presents a number of potential benefits to our pipeline:

- Containers are defined in a task definition that we use to run individual tasks or tasks within a service. In this context, a service is a configuration that enables you to run and maintain a specified number of tasks simultaneously in a cluster. We can run your tasks and services on a server-less infrastructure that is managed by AWS Fargate [20].
- We can schedule the placement of our containers across our cluster based on our resource needs, isolation policies, and availability requirements. With Amazon ECS [4], we don't have to operate our own cluster management and configuration management systems or worry about scaling our management infrastructure.
- ECS [4] allows for automatic scaling using the amazon Fargate feature associated with it. This means that, automatically, during peak traffic times new containers will be created, scaling horizontally and during low traffic times containers will be stopped to keep the costs low.
- Since all the applications are already dockerized [21], and ECS [4] works by using docker images [21], this would lower the work load if an ECS [4] pipeline is chosen.
- ECS [4] also allows new applications to be deployed and built much faster than on the EC2 [9] pipeline used until now. The reason for this is that each application is independent of one another, so if an application is changed it wont be necessary to build all the other applications again.

6.2 Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) [3] is storage for the Internet. It is designed to make web-scale computing easier for developers.

Amazon S3 [3] has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

You can use Amazon S3 [3] to host a static website [22]. On a static website, individual webpages include static content. They might also contain client-side scripts. By contrast, a dynamic website relies on server-side processing, including server-side scripts such as PHP, JSP, or ASP.NET. Amazon S3 [3] does not support server-side scripting.

6.3 Kubernetes

Kubernetes [5] (also known as k8s or ”kube”) is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

In other words, you can cluster together groups of hosts running Linux containers, and Kubernetes [5] helps you easily and efficiently manage those clusters.

Kubernetes [5] clusters can span hosts across on-premise, public, private, or hybrid clouds. For this reason, Kubernetes [5] is an ideal platform for hosting cloud-native applications that require rapid scaling.

6.4 Comparison

In this section the proposed solutions will be compared.

ECS [4] and S3 buckets [3] are both part of Amazon Web Services [35] which make the integration with the rest of the already designed pipelines easier.

Kubernetes is an independent solution not included in AWS. There is a service that makes use of Kubernetes called Amazon Elastic Kubernetes Service (EKS) [36], but its very expensive for the usage our pipelines have. Kubernetes [5] rapid scaling is faster than ECS [4] or S3 [3], but that speed is not required for our applications since they will not see high traffic levels (in the milions of requests, which would make Kubernetes [5] more relevant).

For this reasons, Kubernetes [5] will be discarded as an option going forward. In the following sections ECS [4] and S3 buckets [3] will be analysed in more detail to make the final decision.

7 Criteria to chose the pipeline

In this section we will explain what criteria we are going to use to select the best pipeline for our situation.

First of all, we will be making a model pipeline of ECS [4] and S3 buckets [3] to have an idea of what the end result would look like.

Afterwards we will make an analysis of how ECS [4] and S3 buckets [3] work, to be able to know the complexity involved in using each technology.

Finally, we will be comparing the pricing of each technology, since one of the objectives of this project is to reduce the costs.

We wont be able to compare the deployment time without building the pipeplines, but since we can deploy each application independently of each other it's assumed that both ECS [4] and S3 buckets [3] are faster than the original EC2 [9] pipeline.

We wont be able to compare scaling either, but since both ECS [4] and S3 buckets [3] can scale automatically they already accomplish the objective.

8 ECS Based Pipeline

In this section it will be explained how ECS [4] works and a pipeline will be modeled using it.

8.1 How does ECS Work

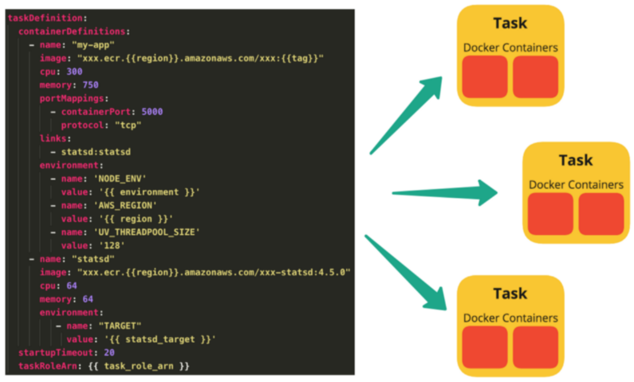
Amazon ECS [4] has multiple components:

- **Task Definitions [8]:** This is the blueprint describing which Docker containers to run and represents an application. It details the images to use, the CPU and memory to allocate, environment variables, ports to expose, and how the containers interact.

- **Task [8]:** An instance of a Task Definition [8], running the containers detailed within it. Multiple Tasks can be created by one Task Definition [8], as demand requires. An example of a Task can be seen in Fig. 2.

- **Service [8]:** Defines the minimum and maximum Tasks [8] from one Task Definition [8] run at any given time, autoscaling, and load balancing.

- **Cluster [8]:** A Cluster [8] is a group of ECS Container Instances. Amazon ECS handles the logic of scheduling, maintaining, and handling scaling requests to these instances. It also takes away the work of finding the optimal placement of each Task [8] based on CPU and memory needs. A Cluster [8] can run many Services [8]. An example of an ECS Cluster can be seen in Fig. 3.



```yaml
taskDefinition:
  containerDefinitions:
    - name: "my-app"
      image: "xxx.ecr.{{region}}.amazonaws.com/xxx:{{tag}}"
      cpu: 300
      memory: 750
      portMappings:
        - containerPort: 5000
          protocol: "tcp"
      links:
        - statsd:statsd
      environment:
        - name: 'NODE_ENV'
          value: '{{ environment }}'
        - name: 'AWS_REGION'
          value: '{{ region }}'
        - name: 'UV_THREADPOOL_SIZE'
          value: '128'
    - name: "statsd"
      image: "xxx.ecr.{{region}}.amazonaws.com/xxx-statsd:4.5.0"
      cpu: 64
      memory: 64
      environment:
         - name: "TARGET"
           value: '{{ statsd_target }}'
  startupTimeout: 20
  taskRoleArn: {{ task_role_arn }}
```

Fig. 2: ECS Tasks to the right. Code to create the task to the left.

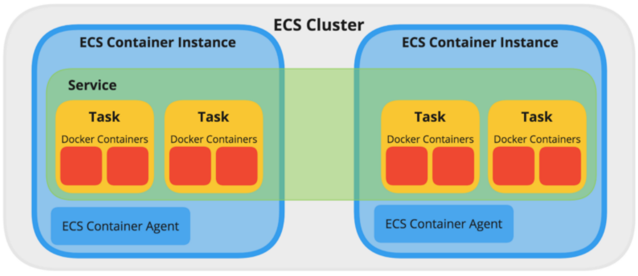


Fig. 3: ECS Cluster with two Services (ECS Container Instance).

8.2 ECS Pipeline Model

The proposed pipeline model can be seen in Appendix 3. Its functionality will be explained in this section.

This new pipeline swaps the EC2 [9] machine for an ECS cluster [8] where all the applications are placed as Services.

Each service scales automatically by using ECS Fargate [20] instances.

An Nginx [23] reverse proxy server [24] will route the traffic to service according to the IP. This server is used to centralize the traffic in order to only need to maintain one SSL certificate [25], otherwise each application would require the SSL [25] certificate in order to use Https [26]. The Nginx [23] server will be another service inside the cluster and the only one open to public requests.

8.3 Cost of ECS

ECS [4] is priced per vCPU and memory used. 0.25vCPU and 0.5GB machine costs 9.010€/month. 0.5 vCPU and 0.5GB machine costs 16.398€/month. 0.25 vCPU and 1GB machine costs 10.632€/month.

This shows that CPU is priced higher than RAM memory.

9 S3 Based Pipeline

In this section it will be explained how S3 buckets [3] work and a pipeline will be modeled using it.

9.1 How does S3 work

Amazon S3 [3] has multiple components:

- **Bucket [27]:** A bucket is a container for objects stored in Amazon S3. Every object is contained in a bucket. For example, if the object named photos/puppy.jpg is stored in the awsexamplebucket1 bucket in the US West (Oregon) Region, then it is addressable using the URL *https://awsexamplebucket1.s3.us-west-2.amazonaws.com/photos/puppy.jpg*.

  Buckets serve several purposes:

  - They organize the Amazon S3 [3] namespace at the highest level.
  - They identify the account responsible for storage and data transfer charges.
  - They play a role in access control.
  - They serve as the unit of aggregation for usage reporting.

- **Objects [27]:** Objects are the fundamental entities stored in Amazon S3. Objects [27] consist of object data and metadata. The data portion is opaque to Amazon S3 [3]. The metadata is a set of name-value pairs that describe the object. These include some default metadata, such as the date last modified, and standard HTTP metadata, such as Content-Type. You can also specify custom metadata at the time the object is stored.

9.2 S3 Pipeline Model

The proposed pipeline model can be seen in Appendix 4. Its functionality will be explained in this section.

This new pipeline swaps the EC2 [9] machine for S3 [3] buckets and an ECS [4] cluster.

The S3 [3] buckets host the frontend of the web applications since they are static. Since S3 [3] buckets do not use images to work, the files will be transfered on the CodeBuild phase instead of taking an ECR [14] image.

The ECS [4] cluster hosts the backends of each application since S3 buckets can only host static frontends. They will use ECR [14] images for each backend service.

CloudFront [28] will be used to cache the web applications across different amazon servers in the world to help reduce the traffic time.

9.3 Cost of S3 Buckets

The pricing of S3 [3] buckets is very low. Storage and requests are what incur costs. Storage is priced at 0.021€/GB. PUT, COPY, POST, LIST requests 0.005€ per 1000 requests GET, SELECT, and all other requests are priced at 0.0004€ per 1000 requests.

10 Pipeline Choice

We will proceed to compare the different Pipeline possibilities. The comparison points will be the cost of each pipeline, the deployment types, the scaling type and the complexity to implement each pipeline.

As we can see on Table 1, S3 [3] and ECS [4] can deploy each application individually, making the deployment time much faster than in the current EC2 pipeline.

TABLE 1: Pipeline Deployment Types

Pipeline	**Deployment Type**
Current EC2 Pipeline	All applications at the same time
ECS Pipeline	Each application individually
S3 Buckets Pipeline	Each application individually
Kubernetes	DISCARDED

TABLE 2: Pipeline Costs

Pipeline	**Monetary Cost**
Current EC2 Pipeline	242€/month
ECS Pipeline	Medium Price (lower than EC2)
S3 Buckets Pipeline	Low Price (lower than ECS)
Kubernetes	DISCARDED

The cost is compared in Table 2. We can only make an estimate of the cost of each pipeline, since to know exactly how much it would cost we would need to build the pipeline first. From the pricing of ECS [4] and S3 buckets [3] that we discussed previously we can estimate that the cheapest option would be S3 buckets. But ECS will still be cheaper than the current EC2 pipeline.

TABLE 3: Pipeline Scaling Type

Pipeline	**Scaling Type**
Current EC2 Pipeline	Manual Scaling
ECS Pipeline	Automatic Scaling
S3 Buckets Pipeline	Automatic Scaling
Kubernetes	DISCARDED

As we can see from Table 3, both S3 buckets [3] and ECS [4] have options to automatically scale the applications, which EC2 [9] did not have.

TABLE 4: Pipelines Complexity

Pipeline	**Complexity to implement**
Current EC2 Pipeline	Already in use pipeline
ECS Pipeline	Easy to implement
S3 Buckets Pipeline	Hard to implement
Kubernetes	DISCARDED

As we can see from Table 4, the complexity of implementing S3 buckets is higher than using ECS [4]. The reason is that S3 buckets would require the usage of ECS for the backend and S3 buckets [3] for the frontends. We would need to combine both systems which increases the complexity of the implementation. If we only use ECS the complexity is much lower.

To chose the final pipeline we can base our decision on Cost and Complexity, since the other parameters are the same for both options.

After analysing the proposed alternatives, the decision has been to use the ECS [4] based pipeline. The main motive is that it would allow to have a compact design having the front end and backend of the applications deployed using the same system, ECS, instead of having to spread it among S3 [3] buckets and ECS. Another reason is due to time constrains, since the implementation is easier using ECS it will make it easier to finish this project before the due date.

11 Final Pipeline Design

In this section it will be explained how the final design of the pipeline works, its strengths and weakness, its cost and how it scales.

11.1 Pipeline Model

The final pipeline design can be seen in detail in Appendix 25. Its functionality will be explained in this section using a simplified diagram.

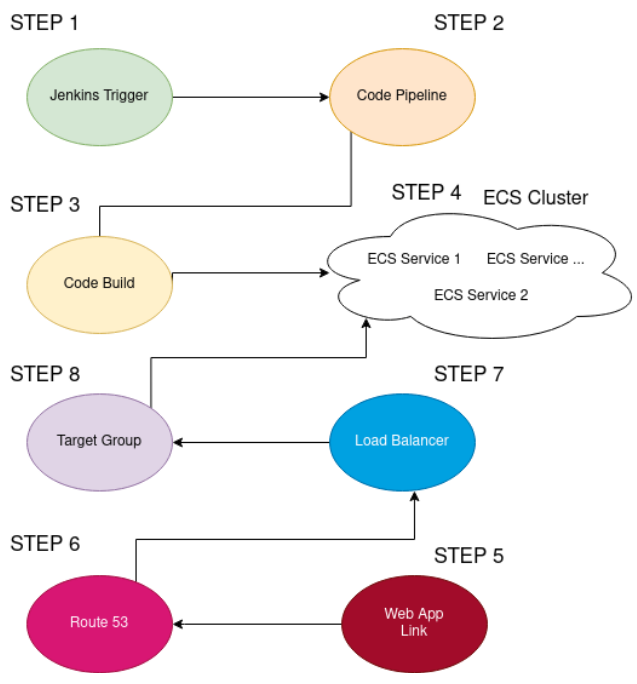


Fig. 4: Simplified final ECS pipeline design

The final pipeline will be based on the ECS [4] model shown previously on Appendix 3. The main difference is that, instead of using an Nginx reverse proxy, an Application Load Balancer [17] will be used.

The reason for that is that Amazon provides the Load Balancer, and even if it costs 20€/month its a good investment since it provides reliability that an Nginx [23] not managed my amazon wont have. The Application Load Balancer [17] will receive all the requests and it will distribute the traffic among the ECS tasks to balance the load.

We can observe this behaviour on Steps 7, 8 and 4. The Application Load Balancer will redirect the petition to the corresponding Target Group, and the Target Group will link it to the ECS Service associated.

The ECS Services are created from the images stored in the ECR repository. This happens during Step 3.

11.2 Pipeline Strengths and Weaknesses

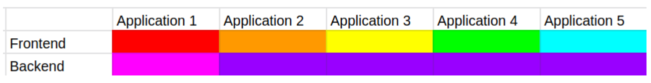
The main advantage of this pipeline is that each application is isolated now, which means that if a change to a specific application is released, only that specific application will need to be deployed again. In the previous pipeline all applications where in the same EC2 [9] machine, which meant that a change to one application required the deployment of all applications.

Another advantage of this pipeline design is that all the maintenance of the ECS [4] machines where the images are running is automated by Amazon. That means that, in case of a sudden spike in traffic ECS will create new tasks and load balance among them.

With the health checks it's also possible to know at all times if a machine is working or not. Another advantage is that deploying new versions of the image requires only a single command, which makes maintaining the pipeline very time efficient.

This pipeline has some weakness as well. The autoscaling requires times to activate, if there is a very sudden spike in traffic it will take a bit of time for new tasks to be created which may cause the application to lag. This is not a big problem because situations with that big of a traffic spike are not likely to occur with the use case its going to get in Idneo.

11.3 Pipeline Cost and Deployment Time



	Application 1	Application 2	Application 3	Application 4	Application 5
Frontend	🟥	🟧	🟨	🟩	🟦
Backend	🟣	🟪	🟪	🟪	🟪

Fig. 5: Five applications with their backend and frontend

As we can observe on Fig. 5 we will consider 5 applications. Each of these applications has a Frontend and a Backend. The Front end is unique for each of these applications, but the backends are shared by Applications 2, 3, 4 and 5. We can observe this by the color code on Fig. 5, each color representing a different component.

We can count a total of: 2 backends and 5 frontends. Or 7 different components. The cost of the components is the following:

- **Code Pipeline [11]:** 1€/month per pipeline = 10€/month (1 per application, each application being in pre prod and prod).
- **Code Build [13]:** Negligible.
- **ECR [14]:** Negligible.
- **Load Balancer [17]:** 20€/month per Load Balancer = 20€/month (1 Load Balancer).
- **ECS [4]:** The backend of application 1 and the frontends cost 9.010€/month per task = 36.04€/month. The backend of application 2 costs 18.020€/month. This is due to the backend of application2 requiring more resources than the other tasks.

Total Cost: 84.06€/month.
Time required to deploy a new version of an application: 5 minutes.

11.4 Scaling

In order to check if the auto scaling is working as intended Amazon Cloudwatch [31] will be used to get a graphic representation of the CPU usage. The test is performed using Apache Jmeter [32], a load testing tool for analyzing and measuring the performance of a variety of services. Using Jmeter, 1000 requests where generated in the span of 1 minute on the application. The results are the following:

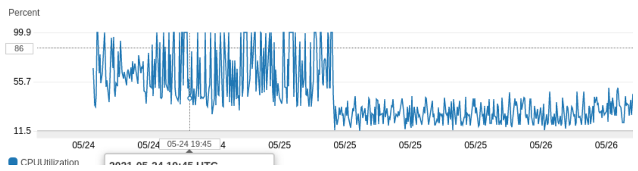


Fig. 6: CPU Usage of an Application before and after performing a Load Test.

This graphic shows how after reaching a 99% CPU usage ECS [4] scaled generating a new instance and reducing the CPU usage to 40% by distributing the load. Under normal circumstances ECS would have scaled before reaching 99% usage of a resource, but due to the fact that the requests where generated in such a low span of time ECS got overwhelmed fast. This situation is not representative of a realistic case in Idneo where such a sudden spike is very unlikely.

12 Result Analysis

This section will compare the new ECS [4] based pipeline with the previous EC2 [9] using pipeline. To do this, the Pipeline Costs, Deployment Time and Scalability Type will be used.

In the following tables we can see the cost, deployment time and scaling type of the original EC2 pipeline and the new ECS one.

TABLE 5: Deployment Times of the pipelines

Pipeline	Deployment Time
Old EC2 Pipeline	17 minutes
New ECS Pipeline	5 minutes

TABLE 6: Cost of the Pipelines

Pipeline	Monetary Cost
Old EC2 Pipeline	242€/month
New ECS Pipeline	84.06€/month

From Table 5 we can observe how, changing to the new pipeline designed in this project we can deploy applications 12 minutes faster. Each application taking only 5 minutes to deploy and being independent of the other applications. This means that, in case of the failure of one application all the other applications will keep working and the only the application with the failure will be affected.

TABLE 7: Scaling of the Pipelines

Pipeline	Scaling Type
Old EC2 Pipeline	Manual
New ECS Pipeline	Automatic

From Table 6 we can observe how, changing to the new pipeline designed in this project we saved 157.94€/month. The total cost of the pipelines being 84.06€/month. This implies a yearly saving of 1895.28€, which accomplishes the goal of reducing the costs associated with deploying our web applications.

Finally, on Table 7 we can observe how this new pipeline designed in this project is able to automatically scale. We demonstrated this on section 11.4 and it will reduce the amount of maintenance required, the pipelines will be able to react to an increase in traffic without the Devops team having to manually increase the amount of resources available to the applications.

To conclude, the total money saved is: 157.94€/month.

The total time saved is: 12 minutes.

The new pipelines scale automatically.

13 Conclusions

As seen from the result analysis, the objective of reducing the cost of the pipeline has been accomplished, saving 157.94€/month and 12 minutes per deployment.

The time saved from deploying new versions of an application will reduce the amount of time developers and the Devops team will have to invest into solving issues with applications, further reducing costs.

Furthermore, previously when a new change to an application was deployed, all the applications became unavailable for 20 minutes. With the new version of the pipeline only the changed application will have downtime, making the whole design more available, stable and requiring less maintenance.

The goal to automatically scale pipelines if required has also been accomplished, which will further reduce the maintenance required for the pipeline, since the Devops team won't have to manually scale the resources associated with the pipelines.

The decision to use ECS was a good one since, with it, the risk of failure of applications is very low due to Amazon itself maintaining the ECS instances.

During the development of this project, the main issues I faced where related to choosing ECS over S3 buckets. The decision came to ease of use in the end, since the time needed to implement an S3 bucket pipeline might have been to high.

14 Potential Improvements

This section will discuss further improvements that can be made in the future on the design of the pipeline.

- One improvement would be the use of CDK [6] to automate the creation of the pipeline itself to make deploying new applications as easy as executing an script.
- Another improvement would be to centralize the logs of all the applications somewhere to make keeping track of the state of the apps easier. This would make maintenance of the applications easier, and would also allow developers an easy way to check if their applications are working as intended.
- To further reduce costs another potential improvement would be to bring some of the applications that allow it to S3 [3] buckets since they are cheaper to maintain than ECS [4].

Acknowledgments

I would like to thank Milu, PauS and PauB from Idneo for their help during the development of this project. Without their help I would not have been able to finish it. I would also like to thank Ana, my tutor from UAB for her help and guidance. Without her corrections this project would not be what it is.

References

[1] Vertical Scaling
https://www.section.io/blog/scaling-horizontally-vs-vertically/

[2] Price of stronger EC2 machines
https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/

[3] S3 Buckets
https://docs.aws.amazon.com/AmazonS3/latest/dev-retired/UsingBucket.html

[4] ECS
https://aws.amazon.com/ecs/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&ecs-blogs.sort-by=item.additionalFields.createdDate&ecs-blogs.sort-order=desc

[5] Kubernetes
https://kubernetes.io/

[6] CDK
https://aws.amazon.com/cdk/

[7] Gantt Graph
https://www.gantt.com/

[8] ECS Task and Cluster
https://www.freecodecamp.org/news/amazon-ecs-terms-and-architecture-807d8c4960fd/

[9] EC2
https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc

[10] Jenkins
https://www.jenkins.io/

[11] AWS Codepipeline
https://aws.amazon.com/codepipeline/

[12] AWS Codesource
https://docs.aws.amazon.com/codestar/latest/APIReference/API_CodeSource.html

[13] AWS Codebuild
https://aws.amazon.com/codebuild/

[14] AWS ECR
https://aws.amazon.com/ecr/

[15] Docker Compose
https://docs.docker.com/compose/

[16] AWS Route 53
https://aws.amazon.com/route53/

[17] AWS Load Balancer
https://aws.amazon.com/elasticloadbalancing/?whats-new-cards-elb.sort-by=item.additionalFields.postDateTime&whats-new-cards-elb.sort-order=desc

[18] AWS Target Groups
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html

[19] Pricing Load Balancers
https://aws.amazon.com/elasticloadbalancing/pricing/

[20] AWS Fargate
https://aws.amazon.com/fargate/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&fargate-blogs.sort-by=item.additionalFields.createdDate&fargate-blogs.sort-order=desc

[21] Docker
https://www.docker.com/

[22] AWS S3 Static Web Hosting
https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteHosting.html

[23] Nginx
https://www.nginx.com/

[24] Reverse Proxy Server
https://www.nginx.com/resources/glossary/reverse-proxy-server/

[25] SSL Certificates
https://www.verisign.com/en_US/website-presence/online/ssl-certificates/index.xhtml

[26] HTTPS
https://www.cloudflare.com/learning/ssl/what-is-https/

[27] AWS S3 Components
https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[28] AWS Cloudfront
https://aws.amazon.com/cloudfront/

[29] Idneo
https://www.idneo.com/about-idneo/

[30] Devops
https://aws.amazon.com/devops/what-is-devops/

[31] AWS Cloudwatch
https://aws.amazon.com/cloudwatch/

[32] Jmeter
https://jmeter.apache.org/

[33] AWS Security Groups
https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

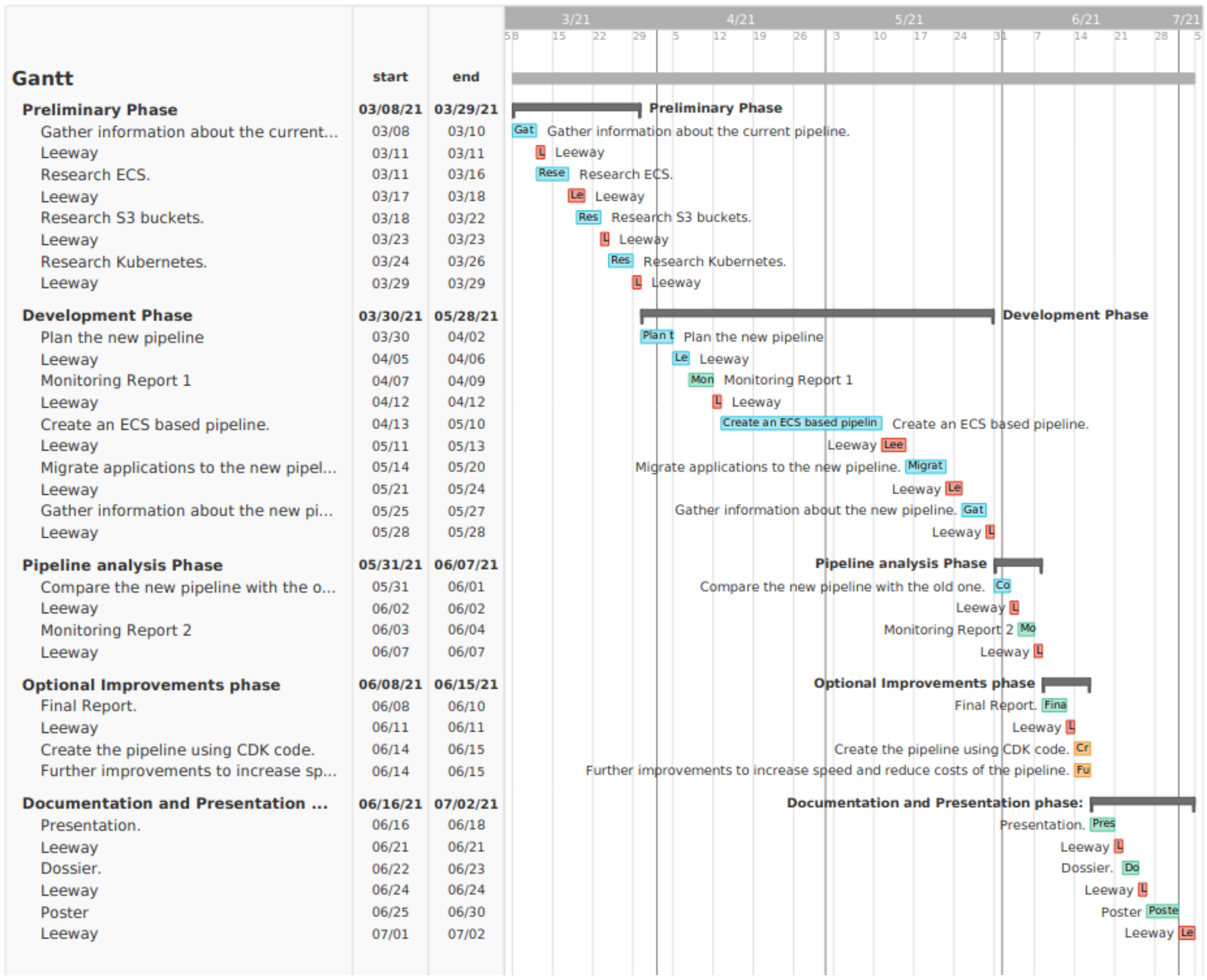
[34] AWS VPC
https://aws.amazon.com/vpc/?vpc-blogs.sort-by=item.additionalFields.createdDate&vpc-blogs.sort-order=desc

[35] Amazon Web Services
https://aws.amazon.com/

[36] AWS EKS
https://aws.amazon.com/eks/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&eks-blogs.sort-by=item.additionalFields.createdDate&eks-blogs.sort-order=desc

APÈNDIX

A.1 Gantt Chart

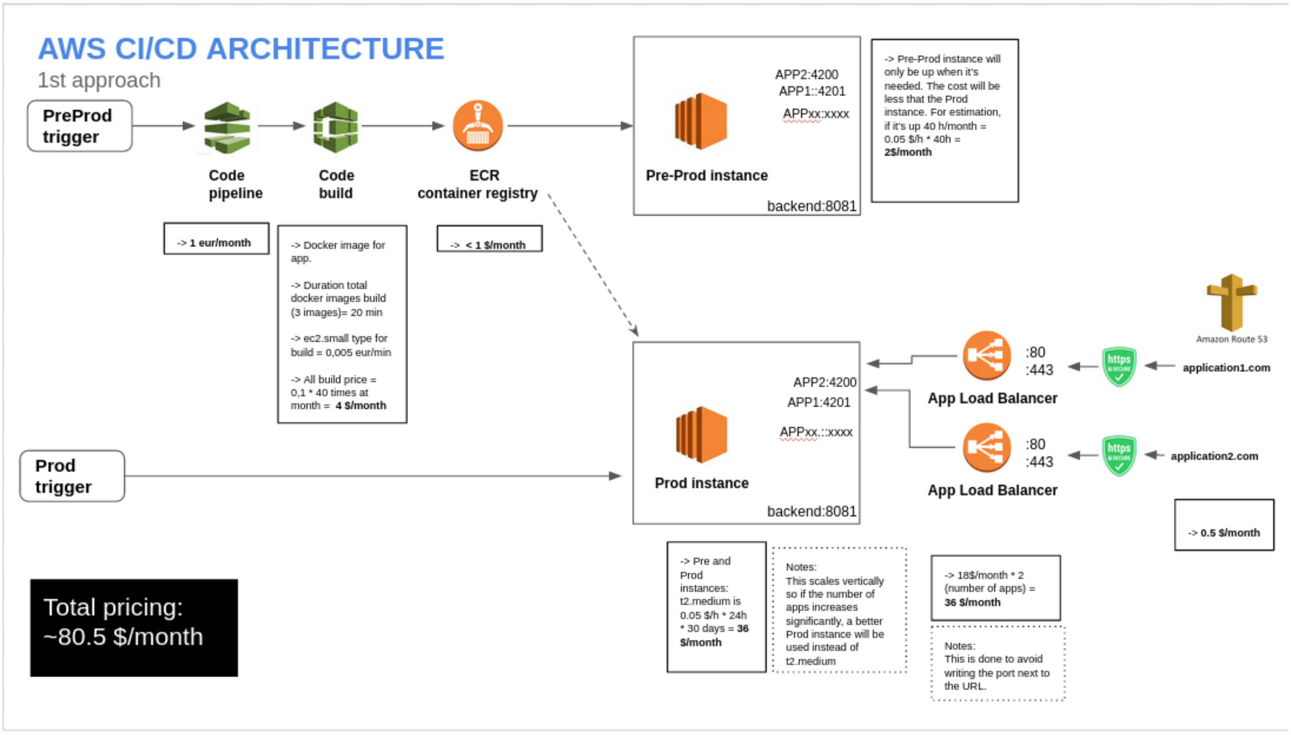


Gantt	start	end
Preliminary Phase	**03/08/21**	**03/29/21**
Gather information about the current...	03/08	03/10
Leeway	03/11	03/11
Research ECS.	03/11	03/16
Leeway	03/17	03/18
Research S3 buckets.	03/18	03/22
Leeway	03/23	03/23
Research Kubernetes.	03/24	03/26
Leeway	03/29	03/29
Development Phase	**03/30/21**	**05/28/21**
Plan the new pipeline	03/30	04/02
Leeway	04/05	04/06
Monitoring Report 1	04/07	04/09
Leeway	04/12	04/12
Create an ECS based pipeline.	04/13	05/10
Leeway	05/11	05/13
Migrate applications to the new pipel...	05/14	05/20
Leeway	05/21	05/24
Gather information about the new pi...	05/25	05/27
Leeway	05/28	05/28
Pipeline analysis Phase	**05/31/21**	**06/07/21**
Compare the new pipeline with the o...	05/31	06/01
Leeway	06/02	06/02
Monitoring Report 2	06/03	06/04
Leeway	06/07	06/07
Optional Improvements phase	**06/08/21**	**06/15/21**
Final Report.	06/08	06/10
Leeway	06/11	06/11
Create the pipeline using CDK code.	06/14	06/15
Further improvements to increase sp...	06/14	06/15
Documentation and Presentation ...	**06/16/21**	**07/02/21**
Presentation.	06/16	06/18
Leeway	06/21	06/21
Dossier.	06/22	06/23
Leeway	06/24	06/24
Poster	06/25	06/30
Leeway	07/01	07/02

3/21
4/21
5/21
6/21
7/21
Preliminary Phase
Gather information about the current pipeline.
Research ECS.
Research S3 buckets.
Research Kubernetes.
Development Phase
Plan the new pipeline
Monitoring Report 1
Create an ECS based pipeline.
Migrate applications to the new pipeline.
Gather information about the new pipeline.
Pipeline analysis Phase
Compare the new pipeline with the old one.
Monitoring Report 2
Optional Improvements phase
Final Report.
Create the pipeline using CDK code.
Further improvements to increase speed and reduce costs of the pipeline.
Documentation and Presentation phase:
Presentation.
Dossier.
Poster
Leeway

Fig. 7: Gantt Chart of the project

A.2 EC2 Based Pipelines. Currently used in Idneo.



-> 1 eur/month

-> Docker image for app.

-> Duration total docker images build (3 images)= 20 min

-> ec2.small type for build = 0,005 eur/min

-> All build price = 0,1 * 40 times at month = **4 $/month**

-> **< 1 $/month**

-> Pre-Prod instance will only be up when it's needed. The cost will be less that the Prod instance. For estimation, if it's up 40 h/month = 0.05 $/h * 40h = **2$/month**

-> Pre and Prod instances: t2.medium is 0.05 $/h * 24h * 30 days = **36 $/month**

Notes:
This scales vertically so if the number of apps increases significantly, a better Prod instance will be used instead of t2.medium

-> 18$/month * 2 (number of apps) = **36 $/month**

Notes:
This is done to avoid writing the port next to the URL.

-> **0.5 $/month**

Total pricing:
~80.5 $/month

Fig. 8: Pipeline originally used in Idneo using EC2

A.3 ECS Based Pipelines, Proposed Model.

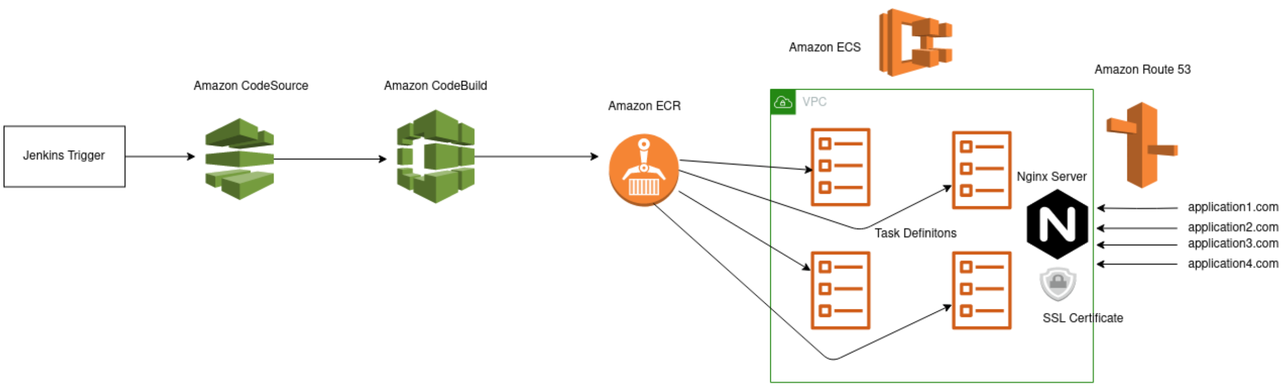


Fig. 9: Proposed Pipeline Model using ECS

A.4 S3 Based Pipelines, Proposed Model.

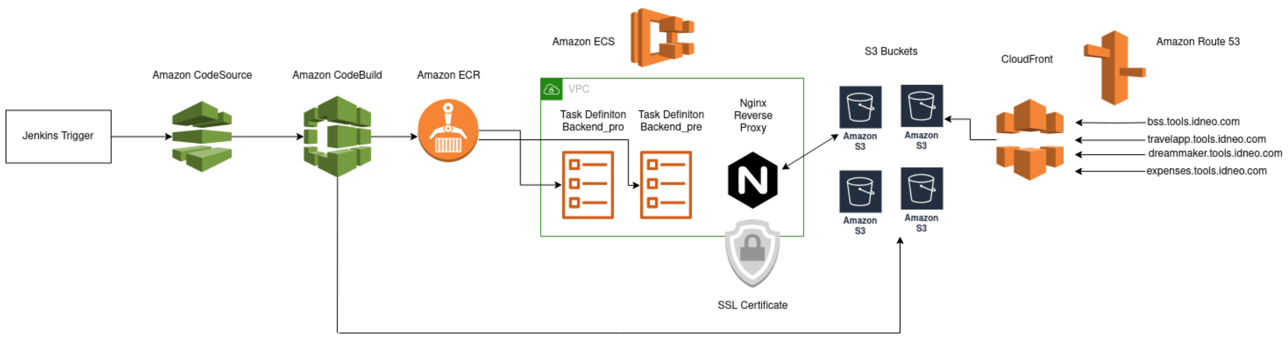


Fig. 10: Proposed Pipeline Model using S3 buckets

A.5 Final Pipeline Design, ECS based.

afds

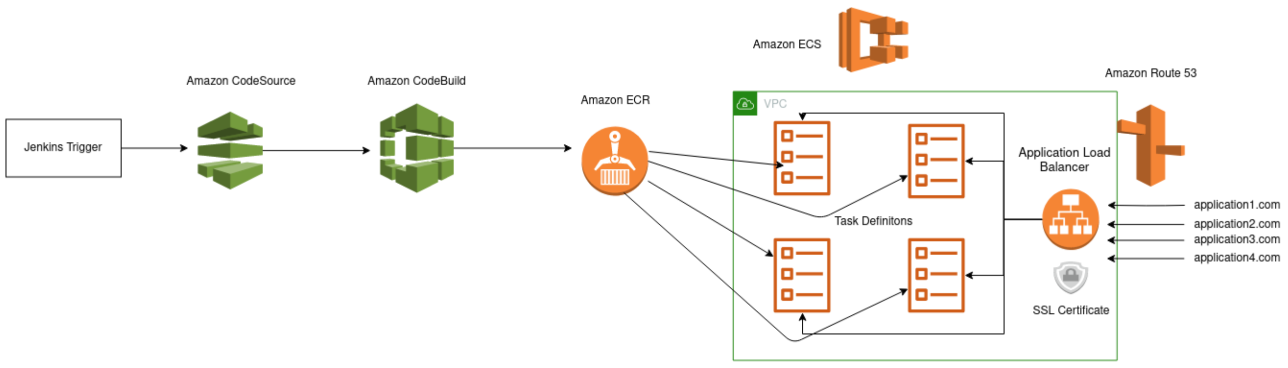


Fig. 11: Pipeline designed in this project using ECS