## 

Manual de instalación

Laboratorio III

*Profesores:*

*Martin Salaberri*

*Mariana Falco*

*Ignacio Berdiñas*

*Matias Cicilia*

*Alumnos:*

*Maximiliano Adaro*

*Katia Cammisa*

*Pedro Gardeliano*

*Joaquín Iannuzzi*

# Índice

[**Índice**](#_5wq9fikwb5qx) **2**

[**Descripción del sistema**](#_jf7d65jn4gx6) **3**

[Base de datos](#_tt6pdx6gfgoh) 3

[Machine Learning](#_r6cv2q8buwuv) 3

[Backend](#_aocvdmqv4zmj) 3

[Frontend web](#_ivh8x8gm9stx) 3

[**Diagrama de arquitectura**](#_5b7v2t7zhloz) **3**

[**Recursos**](#_ss11qvuqjowj) **3**

[**Instalación del software**](#_c9plf23t4jye) **4**

[Elastic Beanstalk](#_s1cxb61c1k30) 4

[RDS](#_2uj3kld9a69t) 4

[S3 Bucket](#_f6qvy9r3fkgq) 5

[Cloudfront](#_wk36mzph4dw) 5

[Route 53](#_ephur3aqnrfj) 5

# 

# Descripción del sistema

Está compuesto por:

## Base de datos

Compuesta por PostgreSQL, usando la extensión Postgis para guardar datos geoespaciales.

## Machine Learning

Usa Python como lenguaje de programación, usando las librerías Pandas, Scikit-learn y Numpy. El algoritmo utilizado es Xgboost y finalmente tiene Flask para hacer las consultas al servidor.

## Backend

Usa Kotlin como lenguaje de programación, utilizando Spring cómo framework.

## Frontend web

Usa TypeScript como lenguaje de programación y React cómo framework.

# Diagrama de arquitectura

# 

# 

# Recursos

**Machine learning:**

El código fuente se encuentra en: <https://github.com/joacoiannuzzi/ubicar-ml>.

Es necesario contar con Python 3.9 y Pipenv

**Backend:**

El código fuente se encuentra en: <https://github.com/MaximilianoAdaro/ubicar-api>.

Es necesario contar con Kotlin 1.4, Gradle 7 y Java 11.

**Frontend web:**

El código fuente se encuentra en: <https://github.com/MaximilianoAdaro/ubicar-client>.

Es necesario contar con Node 15.

# Instalación del software

De acuerdo con la arquitectura a utilizar haciendo uso de AWS como cloud provider, tienen una documentación muy potable donde vamos a estar extrayendo la mayoría de los artículos para la creación y configuración de los servicios a utilizar.

La página con la documentación es la siguiente: [AWS Documentation](https://docs.aws.amazon.com/index.html)

## Elastic Beanstalk

Primero y principal, elastic beanstalk fue elegido como nuestra base de la arquitectura por su configuración rápida con automatización y la personalización que te permite. Este servicio nos permite configurar las instancias EC2 (donde vamos a correr nuestro backend tanto la API principal como el machine learning), load balancer para equilibrar las cargas a los servicios y AutoScaling group, para que el sistema escale automáticamente en caso de que sea necesario.

Creamos una aplicación en elastic beanstalk siguiendo los pasos del siguiente enlace:

[Managing and configuring Elastic Beanstalk applications](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/applications.html)

Luego de tener una aplicación lo que creamos son diferentes ambientes dentro de esta siguiendo el siguiente artículo:

[Managing environments - AWS Elastic Beanstalk](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.html)

Para la configuración del EB tenemos lo siguiente:

[Configuring Elastic Beanstalk environments](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-containers.html)

## RDS

Necesitamos configurar nuestra base de datos PostgreSQL con la extensión PostGIS. Por ende, vamos a hacer uso del servicio de RDS para levantar nuestra base de datos relacional que nos provee AWS.

Llevar a cabo el siguiente tutorial para poder así configurar la base de datos dentro de nuestro Elastic Beanstalk:

[Using Elastic Beanstalk with Amazon RDS](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.RDS.html)

Una vez configurada, se debe agregar PostGIS como extensión de la misma, siguiendo el siguiente artículo:

[Working with the PostGIS extension - Amazon Relational Database Service](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.PostgreSQL.CommonDBATasks.PostGIS.html)

Por último, agregar las credenciales correspondientes a los archivos de configuración YAML.

## S3 Bucket

Usamos Amazon Simple Storage Service (Amazon S3), que es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento, para contener la aplicación web hecha con React.

Primero creamos un bucket y después deployamos la aplicación React, siguiendo este artículo:

[How to deploy your React App with AWS S3 | by Andrew Bestbier | DailyJS | Medium](https://medium.com/dailyjs/a-guide-to-deploying-your-react-app-with-aws-s3-including-https-a-custom-domain-a-cdn-and-58245251f081)

## Cloudfront

Crear una distribución de Cloudfront que apunte al bucket S3:

[Creating a distribution - Amazon CloudFront](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-creating-console.html)

Después, apuntar al load balancer:

[Use a CloudFront web distribution to serve content from multiple origins](https://aws.amazon.com/premiumsupport/knowledge-center/cloudfront-distribution-serve-content/)

## Route 53

Por último, tenemos al Route 53, un servicio web de sistema de nombres de dominio (DNS). Lo usaremos para tres funciones principales: registro de dominio, direccionamiento de DNS y comprobación de estado.

En primer lugar entonces vamos a registrar un nuevo dominio:

[Registering a new domain - Amazon Route 53](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/domain-register.html)

En segundo lugar, configuramos el dominio para que este redireccione al Cloudfront:

[Configuración de Amazon Route 53 como servicio DNS](https://docs.aws.amazon.com/es_es/Route53/latest/DeveloperGuide/dns-configuring.html)

Por último, hacemos la comprobación del estado de los recursos, para verificar que está accesible, disponible y operativo:

[How Amazon Route 53 checks the health of your resources](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/welcome-health-checks.html)