Laboratorio 7

Oscar Grande, Didier Posse.

November 2025

1 Consulta De Tecnologías.

1.1 Terraform.

Terraform es una herramienta de infraestructura con codigo (IaC) creada por HashiCorp (2014) con un lenguaje HCL (HashiCorp Configuration Lenguage) con un paradigma declarativo done el usuario define el estado deseado. con una arquitectura clave con archivos ".tf" (donde va el codigo), o de estado ".tfstate" (donde es el mapeo realidad vs codigo). Ademas con proveedores de plugins para AWS, Azure, etc, y tambien con una arquitectura para backend donde se almacenara el estado.

- Ventajas:
  - Brinda una ventaja estratégica al ser Multi-Cloud siendo un mismo código para AWS, Azure, GCP.
  - Con un versionado donde Git controla cambios de infraestructura.
  - Colaboración de múltiples desarrolladores en una misma infraestructura.
  - Habilidades transversales en DevOps, Cloud, SRE.
  - Gran demanda y en crecimiento exponencial
  - Es base de otras tecnologías como para cloud-native
  - Incremento en salarios con un +30 por ciento para ingenieros con Terraform.

Como casos de uso concretos se puede ver en:

- **Migraciones de Cloud** donde se levanta una infraestructura idéntica en una nueva región/cloud. C on pruebas A/B en diferentes configuraciones
- **Compliance y seguridad** con una infraestructura pre aprobada por seguridad y una auditoria completa de cambios.

- Startups Tech:

```
# Despliegue completo de aplicación web
resource "aws_vpc" "main" { ... }
resource "aws_subnet" "web" { ... }
resource "aws_instance" "app_server" { ... }
resource "aws_db_instance" "database" { ... }
```

Figure 1: ejemplo despliegue aplicación web.

- **Integracion con otros sistemas.** donde parte de codigo en GITHUB y este pasa a Terraform Cloud para luego pasar por AWS/AZURE y terminar siendo monitorizado. Con un ecosistema de Vault (secretos), Consul (Redes y servicio discovery.), Kubernetes (orquestacion de contenedores.), CI/CD (GitLab CI o GitHub actions.).

Lo anteriormente mencionado fortalecerá y complementara habilidades con AWS, Linux, Git, Python/bash y kubernetes, donde se aplican en roles de DevOps, Cloud engineer, SRE (Site Reliability Engineer) o ingeniero en infraestructura.

1.2 Ansible.

Ansible es una plataforma de software libre para la automatización de tareas IT, como el aprovisionamiento de infraestructura, la gestión de configuraciones, el despliegue de aplicaciones y la orquestación de workflows complejos. Con principios funadmentales de agenteless donde se conecta mediante SSH en linux utilizando credenciales existentes, tambien con Idempotencia donde una misma tarea puede ejecutarse multiples veces sobre un mismo sistema, con un lenguaje declarativo donde los usuarios describen el estado final deseado de un sistema, donde Ansible se encarga de determinar como lograr ese estado.

- **Arquitectura:**
  - **Nodo de Control**La máquina donde se instala Ansible y desde la cual se lanzan los comandos y playbooks.
  - **Nodos Gestionados.**Los servidores o dispositivos (Linux, Windows, network devices) que son automatizados por Ansible.
  - **Inventario.** Un archivo (normalmente INI o YAML) que lista todos los nodos gestionados, permitiendo agruparlos (ej: [webservers], [databases]).
  - **Modulos.**Son las unidades de código que ejecutan las acciones específicas. Ansible viene con una extensa biblioteca de módulos (para gestionar paquetes, servicios, archivos, usuarios, etc.).

  - **Playbooks**El corazón de Ansible. Son archivos escritos en YAML que definen un conjunto de tareas y configuraciones que se aplicarán a un grupo de hosts. Describen políticas deseadas o pasos de un proceso general.

  - **Tasks** Una unidad de acción dentro de un playbook. Llama a un módulo con argumentos específicos.

  - **Roles.** Una estructura predefinida de directorios para agrupar y reutilizar playbooks, variables, archivos y plantillas de manera automática. Fomentan la modularidad y el compartir código.

  - **Handlers.**Tareas especiales que solo se ejecutan si son notificadas por otra tarea. Se usan típicamente para reiniciar servicios después de un cambio de configuración.

- **Ventajas.**

  - Baja barrera de entrada gracias a su uso de Yaml y la ausencia de agentes es mas facil de aprender a implementar.

  - Robusto y flexible por la enorme colección de módulos y su integración con prácticamente cualquier tecnología lo cual lo hace muy versátil.

  - Idempotencia Garantizada al reducir los riesgos en entornos de producción.

  - Con alta comunidad y soprte activa con un fuerte respaldo de RED HAT.

Es por esto que la gestión de configuraciones garantiza que cientos de servidores tengan una configuración consistente y cumplan con las políticas de seguridad. Con un despliegue continuo que automatiza el proceso de llevar una nueva versión de una app desde el repositorio de código hasta los entornos de producción.

También con un aprovisionamiento que le permite crear y configurar maquinas virtuales, contenedores o instancias en la nube de manera automatizada, adicionando la orquestación de procesos complejos que involucren múltiples sistemas, lo cual lo hace bastante útil para la **Automatización de redes (NetDevOps)** donde puede configurar switches, routers o firewalls con un gran cumplimiento y seguridad.

1.3 Rabbit

RabbitMQ es un software de mensajería de código abierto que implementa el protocolo AMQP (Advanced Message Queuing Protocol.) el cual actua como

middleware de mensajería o message Broker, permitiendo la comunicación asíncrona y confiable entre aplicaciones distribuidas.

SU arquitectura es basada en brokers (centralizada), con un sistemas de colas de mensajes robustas y con protocolo AMQP que es estándar industrial, su alta persistencia de mensajes ofrece una alta disponibilidad y escalabilidad.

- **Arquitectura:**
  - **Producer(Productor):** Es aquella aplicación quien envía los mensajes.
  - **Consumer (Consumidor):** Aplicación que recibe mensajes.
  - **Queue (Cola):**Almacenamiento temporal de mensajes.
  - **Exchange (Intercambiador):** Recibe mensajes y los enruta a colas.
  - **Binding (Enlace):**Regla que conecta exchanges con colas.
  - **Direct:** Enrutamiento exacto (1:1).
  - **Fanout:** Broadcast a todas las colas (1:N).
  - **Topic:** Enrutamiento basado en patrones.
  - **Headers:** Enrutamiento basado en atributos.
- **Ventajas:**
  - Confiable por su persistencia y confirmación de mensajes.
  - Flexible con múltiples patrones de mensajería.
  - Escalable con clústeres y mirrors de colas.
  - Interfaz web incluida
  - Impacto en nuevas tendencias como arquitecturas de microservicios, Sistemas financieros, tecnologías IoT y procesamiento de datos en tiempo real.

  Es por ello que Rabbit se orienta mas en usos a comunicaciones entre micro servicios desacoplando servicios para mayor resiliencia y permitiendo escalado independiente. El procesamiento asíncrono con tareas en segundo paso y el manejo de los picos de carga acompañado de un balanceo de carga el cual distribuye mensajes entre múltiples consumidores. Integrándose en sistemas Legacy conectando aplicaciones modernas con sistemas heredados, con un Buffer de picos de trafico el cual previene la saturación de sistemas.

1.4 Referencias.

-

2 Análisis De Sentimientos Por Imágenes.

2.1 Codigo.py

Este es el programa principal que usa la cámara, detecta el rostro con MediaPipe y analiza gestos como sonrisa, tristeza o enojo. Procesa los puntos del rostro, calculando distancias y aplicando reglas para determinar la emoción. para mayor visualización de este y mayor explicación de este puede ir directamente al código que se encuentra en la carpeta de AnalisisSentimientos.

Figure 2: Sentimientos.py

2.2 Dockerfile

Con el archivo Dockerfile definiremos la construcción de la imagen del proyecto, instalando los paquetes, copiando el codigo, las dependencias del sistema, creando el entorno donde correrá el programa.

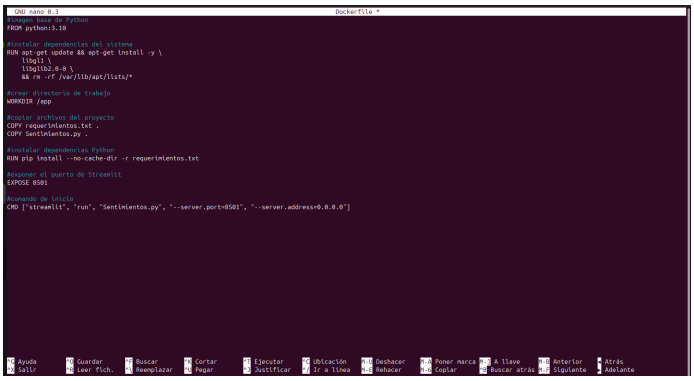

Figure 3: archivo Dockerfile.

2.3 Requerimientos.txt

Este archivo lista todas las libreria que usa el codigo para que el Dockerfile pueda instalarlas automaticamente.

Figure 4: requerimientos.txt

2.4 Docker compose.yml

Sera quien permita levantar la construcción de la imagen de una manera mur corta con "una solo comando", levantando el archivo Dockerfile y exponiendo la información en los puertos.

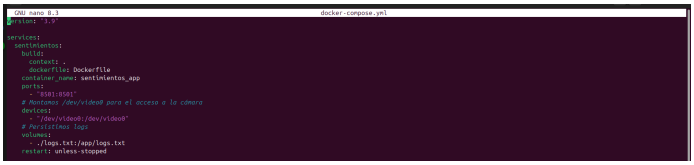

Figure 5: docker-compse.yml

2.5 Proceso.

Una vez tengamos los archivos correspondientes para el funcionamiento del proyecto lo que debemos hacer es tener en cuenta que estos 4 archivos deben estar incluidos en un mismo directorio para la construcción de la imagen, donde iremos paso por paso:

Como se puede evidenciar en la imagen se tiene una carpeta con "Proyectos" donde tenemos la carpeta de este proyecto la cual es "AnalisisSentimientos"

donde se encuentra los archivos de construccion para el proyectos y la imagen necesaria para el funcionamiento de este, la ifnormacion incluida en cada uno de los archivos la encuentra en las secciones anteriores con su contenido y explicación.

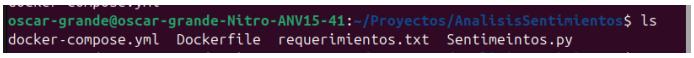

Figure 6: Directorio Proyecto.

Hecho esto debemos tener el entorno virtual activado donde lo puede hacer de la siguiente manera:

Debe realizar como primer paso el python3 -m .... Luego de esto debe activarlo con source de la siguiente manera y una vez esté, este activo lo que hará es verse en la consola como en la ultima linea el (Entorno).

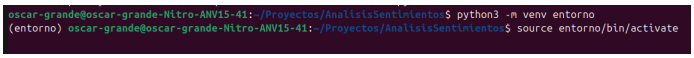

Figure 7: Activación entorno virtual.

Para la construcción de la imagen debe realizarlo de la siguiente manera, con docker compose. Para este paso debe tener en cuenta que el nombre del archivo del docker compose, requiere de una manera de reconocimiento donde su nombre de archivo debe ser exclusivamente "docker-compose.yml" ya que al escribirlo de una manera distinta este no sera reconocido y le arrojara fallos para construir la imagen. Ademas debe tener en su conocimiento que los nombres de los archivos donde tiene almacenado el funcionamiento de la lógica del código deben concluir con el llamado de estos en el archivo Dokcerfile, de tal manera que si tiene algún fallo la consola le determinara cual es el error que se esta cometiendo.

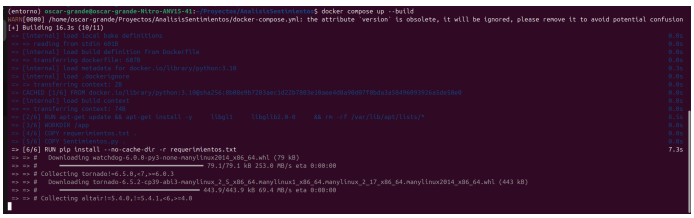

Figure 8: Imagen en construcción.

Una vez contruida podemos ver el funcionamiento de esta.

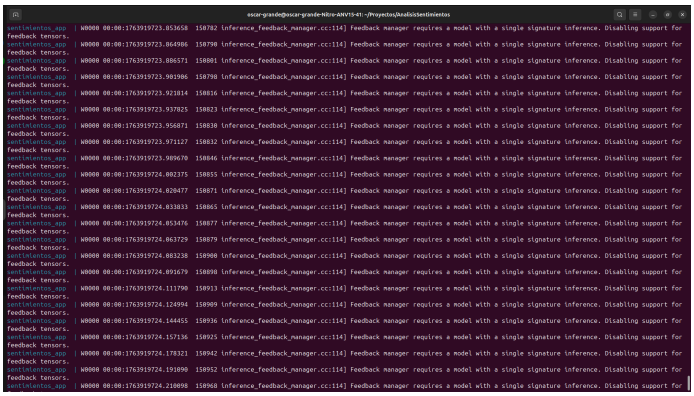

Figure 9: Funcionamiento.

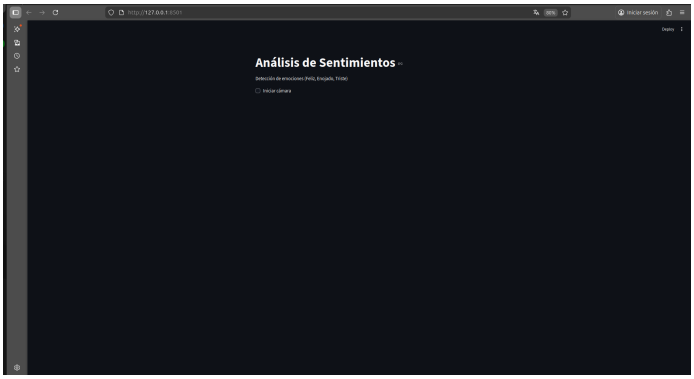

Figure 10: Despliegue del streamlit Puerto 8501.

Una vez aquí podemos dar click en abrir cámara par que empiece a funcionar el detector.

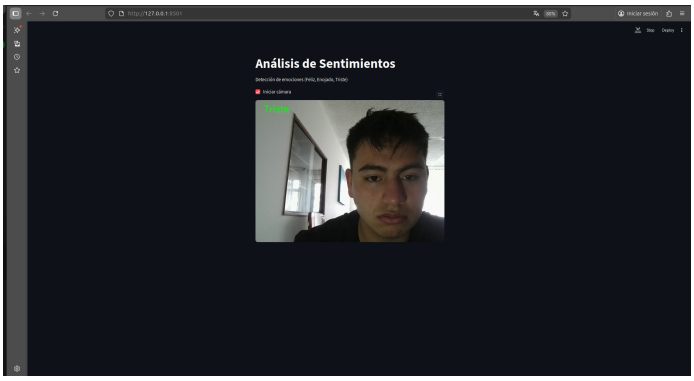

Figure 11: Triste.

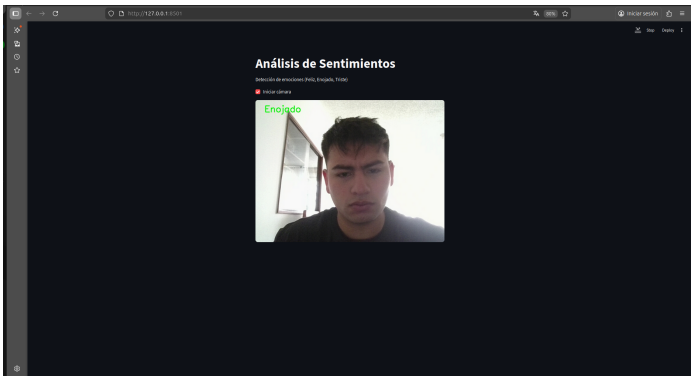

Figure 12: Enojado.

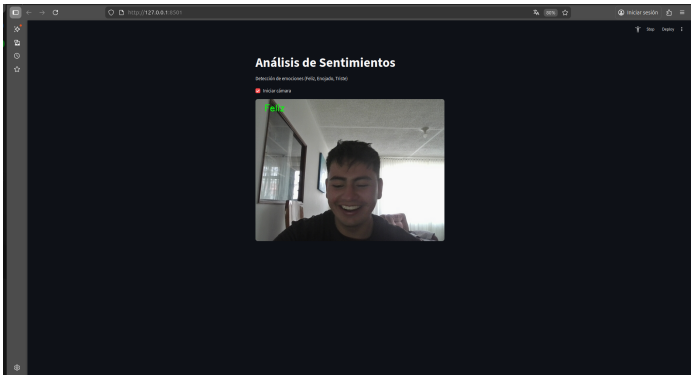

Figure 13: Feliz.

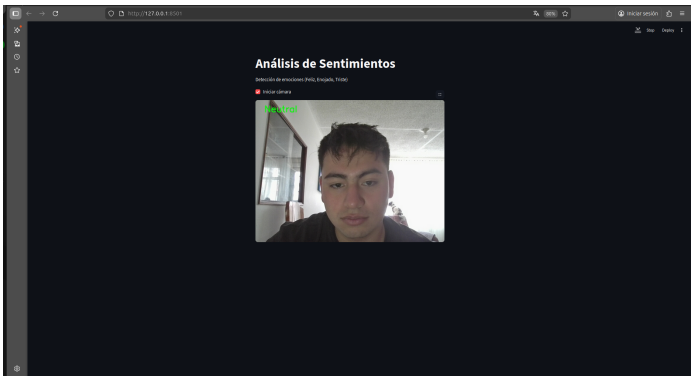

Figure 14: Neutral.

3 Desarrollo ETL De Una Base De Datos.

Como objetivo del ejercicio se busca desarrollar un sistemas completo de extracción, transformación y carga junto con un dashboard interactivo para visualizar y analizar datos de sensores proveniente de un archivo de toma de datos excel.

- **Lenguajes y herramientas:**
  - BASH: automatización de procesos ETL.
  - Python3: Procesamiento de datos y dashboard. con librerias de pandas (procesamiento de datos.), streamlit, openyxl(para la lectura del excel.), plotly (para visualizaciones.), numpy (para calculos numericos.), sqlite (base de datos, built).
  - SQlite: Tecnicas de almacenamiento de datos.
  - Streamlit: Visualizacion web interactiva.
  - Pandas: Manipulacion de datos.
  - Plotly: Graficos interactivos.

Para el desarrollo de extracción, transformación y carga de datos se debe compartir el siguiente esquema de trabajo.

```
oscar-grande@oscar-grande-Nitro-ANV15-41:~/Sensor_dashboard$ ls
BD_SENSORES.xlsx  database        logs            setup.sh
dashboard.py      diagnostic.sh   run_project.sh  venv_sensors
data              etl_process.sh  scripts
oscar-grande@oscar-grande-Nitro-ANV15-41:~/Sensor_dashboard$
```

Figure 15: Esquema de trabajo.

- **FASE 1 Extracción.** Para la extraccion se realiza la lectura multiple de las hojas de excel donde se encuentran los sensores. Por ejemplo; 'SENP1', 'SENP2', 'SENP3', 'SENP4', 'SENP5', 'SENV', etc.

- **FASE 2 Transformación** Es donde se procede a realizar la limpieza de los datos, removiendo metadatos y formulas, convirtiendo valores de voltaje a valores numéricos estandarizando los nombre de columnas y teniendo en cuenta el manejo da valores nulos, con el fin de hacer hacer un filtrado y una lectura de datos mas eficiente.

  Para luego realizar la estructuración agregando esos metadatos consolidando las múltiples hojas de datos y generando IDs únicas.

- **FASE 3 Carga.** Carga de base de datos con 3 tablas con datos crudos, la estadística por sensor, y los resúmenes por tipo.

- **Proceso resumido.**

  1. Creación y manejo del entorno virtual en python.
  2. Procesamiento de datos heterogéneos con una limpieza robusta.
  3. Automatizacion de procesos con la integracion entre bash y python con manejo de errores y creacion automatica de estructura de directorios.
  4. Interfaz de usuario con visualizaciones interactivas.

- El archivo BDSENSORES.xlsx es el archivo originario donde se encuentran los datos originales.

- El archivo runproject.sh es el script principal de la ejecucion del programa

- etlproject.sh es que el archivo con el script correspondiente para la extraccion, transformacion y carga.

- setup.sh es el archivo que contiene el script de instalacion de dependencias.

- dashborad.py es el archivo con el dashboard desplegado en streamlit.

- diagnostic.sh es el archivo con el script de diagnostico como tal.

- En la carpeta data derivan otras 2 carpetas (raw quien contendrá los datos crudos) y (processed con los datos procesados).

- en database se almacenara la base datos generada.

- y en la carpeta log se encontraran los archivos log del programa.

- **RESULTADOS.**

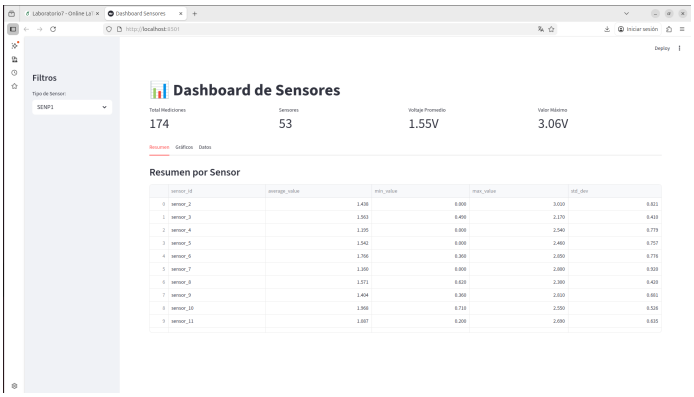

Figure 16: Dashboard.

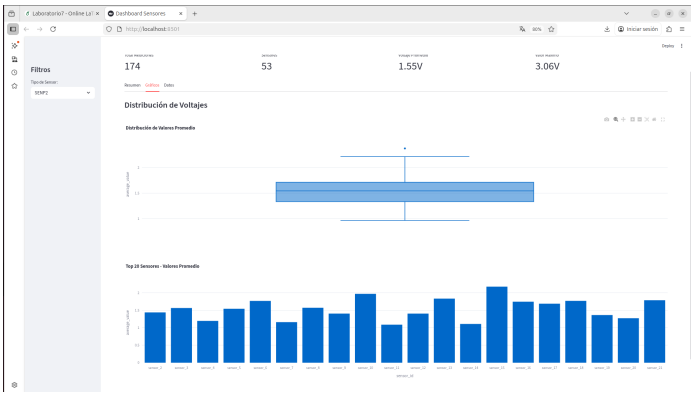

Figure 17: Visualización de gráficas del dashboard.

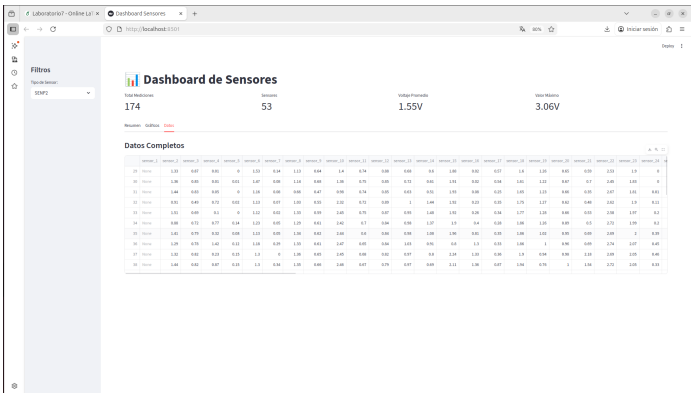

Figure 18: Datos completos del dashboard.

Para mayor visualización de los códigos implementados puede verlos en la carpeta creada en GITHUB con el nombre del proyecto que contiene los archivos que se emplearon.

4 Proyecto MinCiencias.

4.1 Propuesta.

Colombia es un país el cual se enfrenta y esta en constantes riesgos ambientales y naturales como sismos, derrumbes, inundaciones, deslizamientos de tierra y huracanes debido a su geografía, clima tropical y actividad sísmica. Por estos eventos se generan perdidas humanas, económicas, culturales y sociales, por eso esta implementación se busca adaptarla en zonas de alto riesgo y poco monitorizadas así con el fin de evitar desastres como el que fue la tragedia de armero en 1985.

Actualmente, la información sobre posibles desastres llega demasiado tarde o está fragmentada, lo que impide una respuesta oportuna de las autoridades y comunidades. Además, existe dependencia de soluciones externas que no se adaptan a las particularidades del territorio colombiano. Por ello se busca fortalecer las capacidades nacionales con IA y tecnología integrando infraestructura de alto rendimiento, modelos de predicción avanzados y sistemas de alerta automatizados, contribuyendo al territorio tecnológico del país y posicionandolo como referente regional en innovación y seguridad ciudadana.

4.1.1 Objetivos y Desarrollo.

- **Objetivo general:** Desarrollar un sistemas inteligente de alerta temprana de desastres naturales el cual integre datos geoespaciales, geográficos, ambientales y sociales, desarrollando una infraestructura tecnológica de datos robusta con modelos de inteligencia artificial para predecir eventos,

generar alertas automáticas y minimizar el impacto negativo a la población y la infraestructura del país.

- **Objetivos específicos.**
  - Implementar una infraestructura de alto rendimiento como servidores en linux. contenedores y almacenamiento en la nube, el cual procesara grande volumenes de datos en tiempo real.
  - Desarrollar modelos de prediccion de desastres basados en machine learning adaptados a las condiciones geográficas y climáticas de Colombia.
  - Integrar LLM para análisis de información textual de reportes, redes sociales y alertas de medios locales.
  - Diseño de dashboards interactivos con sistemas de notificaciones para autoridades y población.
  - Garantizar la disponibilidad, seguridad y escalabilidad del sistemas para que opere de manera optima durante eventos críticos o evitando estos.
- **Etapas de solución.**
  - **1.Recolección de datos** EL sistema tendra que recopilar informacion de distintas fuentes de datos:
    - Sensores (IoT): Realizando la respectiva ETL de humedad del suelo, niveles de ríos, inclinación de la tierra, vibraciones sismicas, etc.
    - Estaciones meteorológicas: Temperatura, precipitación, velocidad del viento. Esto a través de información compartida por la red o almacenada en la nube.
    - Imágenes Satelitales: Cambios en nomenclaturas terrenales o cuerpos de agua.
    - Redes sociales (comunidad): Análisis veredicto de información relacionada a desastres reportados por la población.
  - **2. Procesamiento y almacenamiento.**
    - Los datos ingresaran a pipelines ETL para su respectiva normalización de los datos.
    - Estos datos serán almacenados en una base de datos relacionales en la nube, con replicación e implementación de encriptado para evitar alteración de estos datos.
    - Establecer una base de servidores que asegure la estabilidad y control puede ser linux.

    * Realizar un empaquetado de Dockerización con el fin de tener los distintos módulos de ingesta, análisis, predicción y alerta.

- **Análisis con IA.**
    * Modelado de machine learning para prediccion de desastres.
    * Redes neuronales convolucionales para el procesamiento de imágenes y detección de cambios.
    * LLM para la interpretación de reportes de texto, en distintos medios de comunicación o por la población.
    * Alertas automáticas generadas según los criterios de riesgo que se apliquen y la probabilidad de un desastre.
- **Sistema de notificación.**
    * Implementación de un dashboard en tiempo real para las autoridades pertinentes en el asunto, con mapa, gráfica de riesgo y predicciones.
    * Notifcado automático a telefonía móvil, radio o sirenas inteligentes en las zonas vulnerables o zonas con riesgo natural.
    * sistema escalable capaz de adaptarse a diferentes regiones con distintos riesgos.
- **Seguridad del sistema.** Monitoreo continuo de la información encriptada anteriormente, implementación de barreras de acceso o firewall, con sistemas estables que no generen cuellos de botella en la información suministrada.

4.2 Diagrama del sistema.

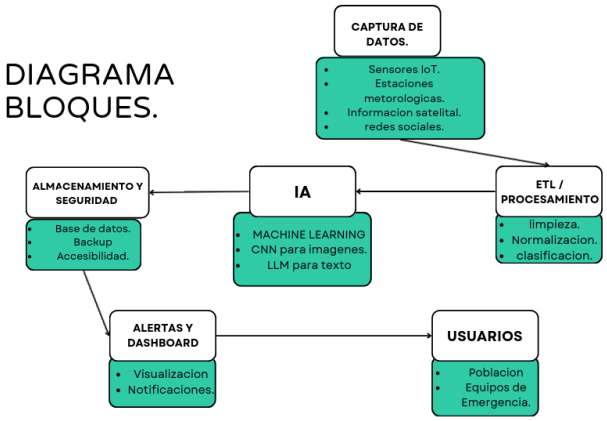

Figure 19: Diagrama de bloques.

4.3 Tecnologías Futuras.

- Digital Twins: Permite replicar "obejtos" en tiempo real, siendo así un simulador en el cual se pueda aplicar pruebas de cualquier tipo a ese "objeto", en nuestro caso terreno o ciudad.

- IA Multimodal avanzada: Es aquella que amplia capacidades generativas de texto, imágenes o datos con el fin de procesar y comprender la información de diferentes modos sensoriales. Lo cual al aplicarlo en el sistema podría brindar predicciones mas precisas.

- EDGE COMPUTING: Es una arquitectura de procesamiento de datos que acerca el almacenamiento y la computación a la fuente que genera los datos (nuestro caso sensores)en vez de enviarlos a un centro de datos centralizado, con ayuda de redes 5G se podra procesar datos cerca del lugar del evento, reduciendo latencias en alertas criticas.

- Blockchain: Ya que tenemos un sistema descentralizado, podemos aplicar el blockchain, el cual ayuda a generar una trazabilidad e integridad de los datos, la cual no se podrá manipular y solo puedes ser actualizada y validad por todo el conceso de la red.