Agentes Autónomos de Prevención

El Ojo de Sauron

Taller de Programación I

[75.12/95.04/TA045]

1er Cuatrimestre 2024

Profesores

Pablo Deymonnaz
Mauro Di Pietro
Rafael Berenguel

Integrantes

ALVAREZ Mateo - 108666
ANDRESEN Joaquin - 102707
GISMONDI Máximo - 110119
NORIEGA Cristhian David - 109164

Problema

Se busca diseñar e implementar un sistema de vigilancia para el gobierno de la Ciudad de Buenos Aires. Este deberá contar con una red de mensajería que permita conectar a una flota de drones, un sistema de cámaras y una central de monitoreo para realizar tareas de prevención de incidentes.

Entidades

Monitor

El monitor cuenta con una interfaz gráfica que permite a los operadores:

- Visualizar un mapa con las distintas entidades e incidentes.
- Visualizar una lista de todos los drones con su posición, estado y batería.
- Visualizar una lista de cámaras proporcionadas por el sistema de cámaras con su posición y estado.
- Visualizar una lista de incidentes, con su posición, estado y detalles del problema.
- Registrar los usuarios y contraseñas para que solo aquellos drones que sean autenticados por el servidor puedan interactuar con el sistema.
- Crear nuevos incidentes indicando los detalles del problema.
- Cerrar incidentes resueltos por los drones para que retornen a su posición original.

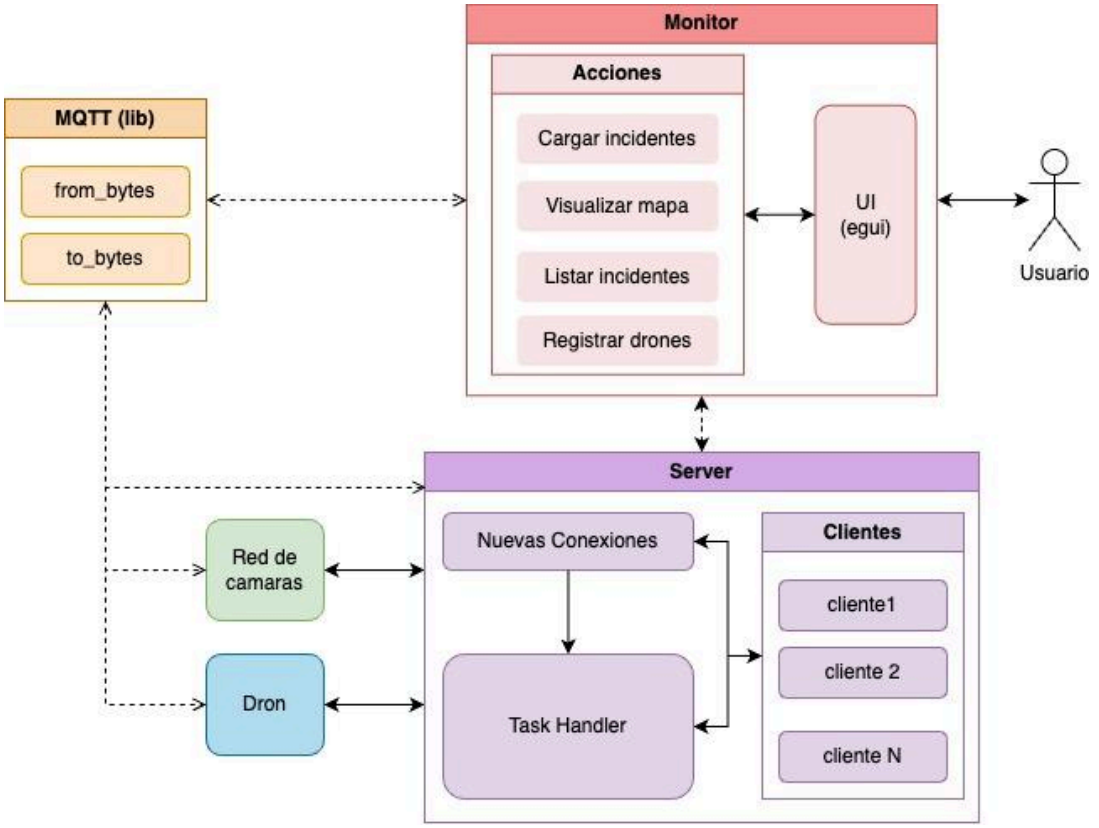
Drones

Los drones forman un cuerpo de vigilancia en su rango determinado. Cada uno debe ser autenticado con usuario y contraseña para formar parte de la red. Una vez registrado, este puede formar parte de la red y estará a cargo de su zona de operación, que cuenta con un punto de ancla en el cual se encontrará a la espera de nuevos incidentes y un radio efectivo en el que el dron puede viajar para resolver todo tipo de problemas. Al tratarse de un dispositivo inalámbrico, este cuenta con una batería limitada la cual debe ser recargada en una estación de carga común para varios drones.

Sistema de cámaras

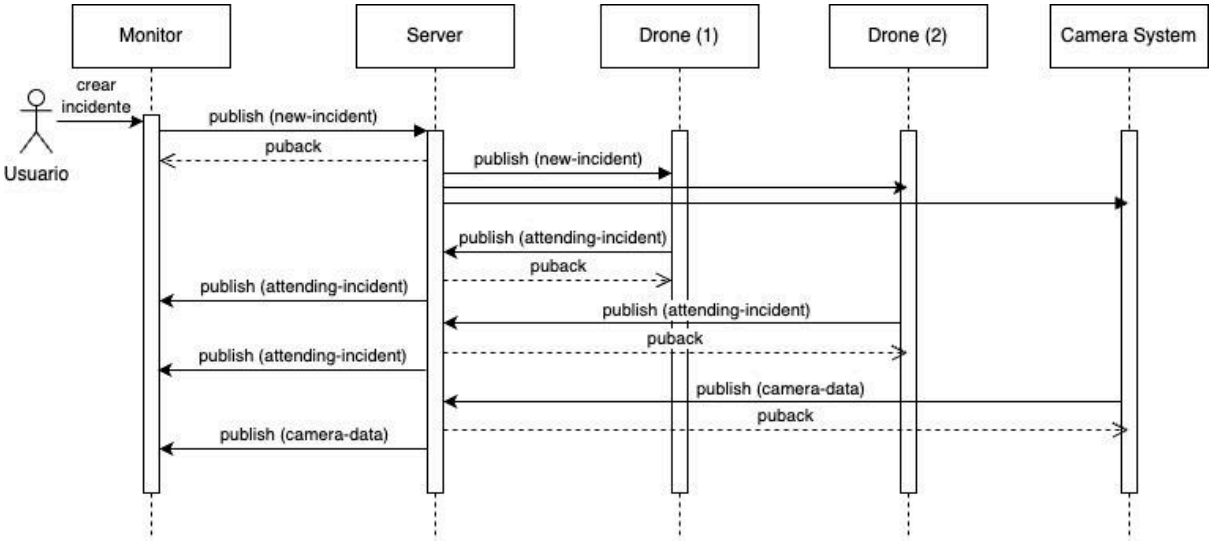
El sistema de cámaras administra una red de cámaras esparcida en puntos estratégicos de la Ciudad de Buenos Aires. El sistema alerta a las cámaras cercanas en el momento que un nuevo incidente se registra en el sistema. Cuando esto ocurre y el incidente se encuentra en el rango de una cámara, pasa a un estado activo. Una vez que el incidente se encuentra resuelto, las cámaras vuelven a un modo de ahorro de energía donde su consumo será reducido.

Arquitectura



Diagramas de Secuencia

Nuevo incidente

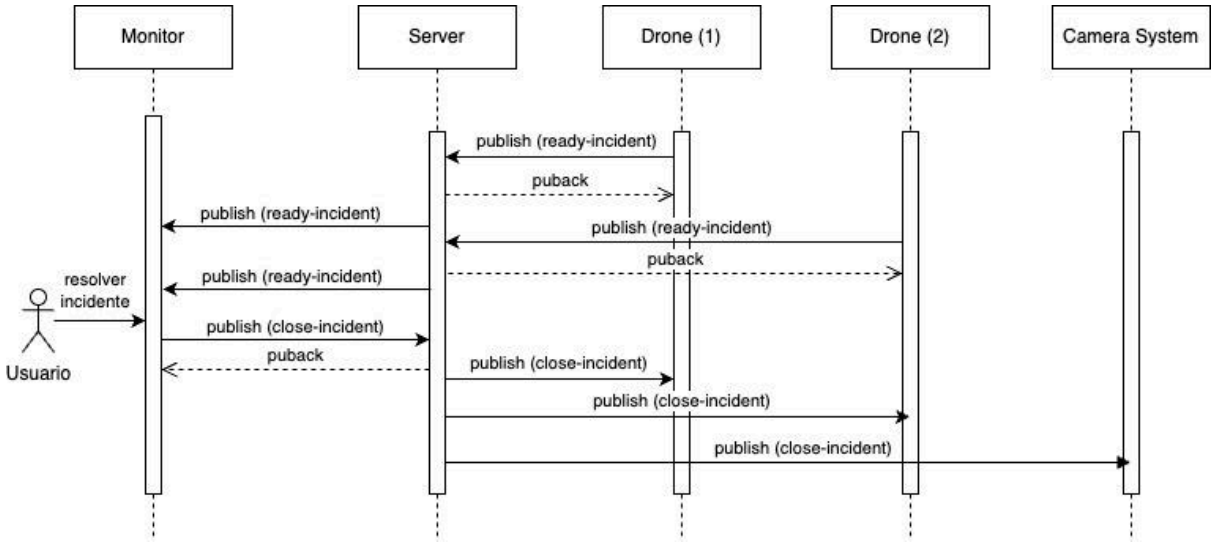


En este diagrama se puede observar el intercambio de mensajes entre los componentes del sistema cuando un usuario crea un incidente utilizando el monitor. En este caso quienes reciben el mensaje son dos drones y el sistema de cámaras.

Suponiendo que ambos drones están cerca del incidente, comienzan a moverse y le avisan al monitor. Mientras que el sistema de cámaras enciende las cámaras que se encuentran cerca y le envía una actualización al monitor.

Todos los mensajes siempre pasan por el servidor, el cual se encarga de facilitar la comunicación entre partes.

Resolución de un incidente



El segundo diagrama de secuencia muestra el comportamiento del sistema cuando se resuelve un incidente.

Primero, dos drones le avisan al monitor que ya terminaron de atender el incidente. Cuando el monitor recibe dos mensajes de este tipo, le permite al usuario cerrar el incidente.

Finalmente, cuando el usuario cierra el incidente, el monitor publica un mensaje para avisarles a todas las entidades que el incidente está cerrado para que estos se liberen.

Red de mensajería

Para permitir la comunicación dentro de nuestro sistema, implementamos una red de mensajería basada en el protocolo MQTT v3.1.1, que cuenta con todas las prestaciones requeridas por el gobierno.
Este protocolo es de tipo ligero basado en el modelo publisher-subscriber donde tenemos varios clientes que publican y se suscriben a los distintos tópicos y un servidor central que funciona como broker e intermediario entre los clientes.

Autenticación

El protocolo cuenta con un formato estándar para enviar usuarios y contraseñas a la hora de conectarse, que se debe acoplar con un sistema personalizado para validar los datos. En nuestro caso, guardamos los usuarios y contraseñas válidos en el mismo broker que realiza las conexiones y tenemos previamente registrada una cuenta que permite hacer los registros de nuevos usuarios.

Encriptación

Para proteger los datos, utilizaremos un sistema de encriptación simétrica que permite a los paquetes viajar a través de la red de manera segura. Aunque estos paquetes podrían ser interceptados, no podrán ser interpretados, gracias a la encriptación. Cada paquete se encripta con una contraseña conocida únicamente por los endpoints gubernamentales, y esta contraseña se puede cambiar con la frecuencia que el gobierno considere necesaria para mantener la seguridad.

Utilizamos una contraseña alfanumérica de 32 caracteres, lo que ofrece una protección robusta contra ataques de fuerza bruta. Si alguien intentara descifrar mediante fuerza bruta, y suponiendo una capacidad de 1 millón de intentos por minuto, le llevaría aproximadamente 10^40 años descubrir la contraseña. Para poner esto en perspectiva, este tiempo es mucho mayor que la edad del universo. Por lo tanto, actualizar la contraseña cada 24 horas es más que suficiente para garantizar la seguridad del sistema.

Calidad de servicio

El protocolo cuenta con el mecanismo para tener una calidad de servicio de al menos uno, que asegura que siempre y cuando el cliente se encuentre conectado, se intentará hacer la publicación de un mensaje hasta lograrlo avisando de vuelta al cliente con un mensaje de tipo acknowledge.

Configuracion

Se cuenta con un archivo de configuración que permite configurar parámetros básicos del servidor como el puerto que expone, el usuario y contraseña del administrador de usuarios, la ruta de los archivos de logging, backup y demás.

Logging

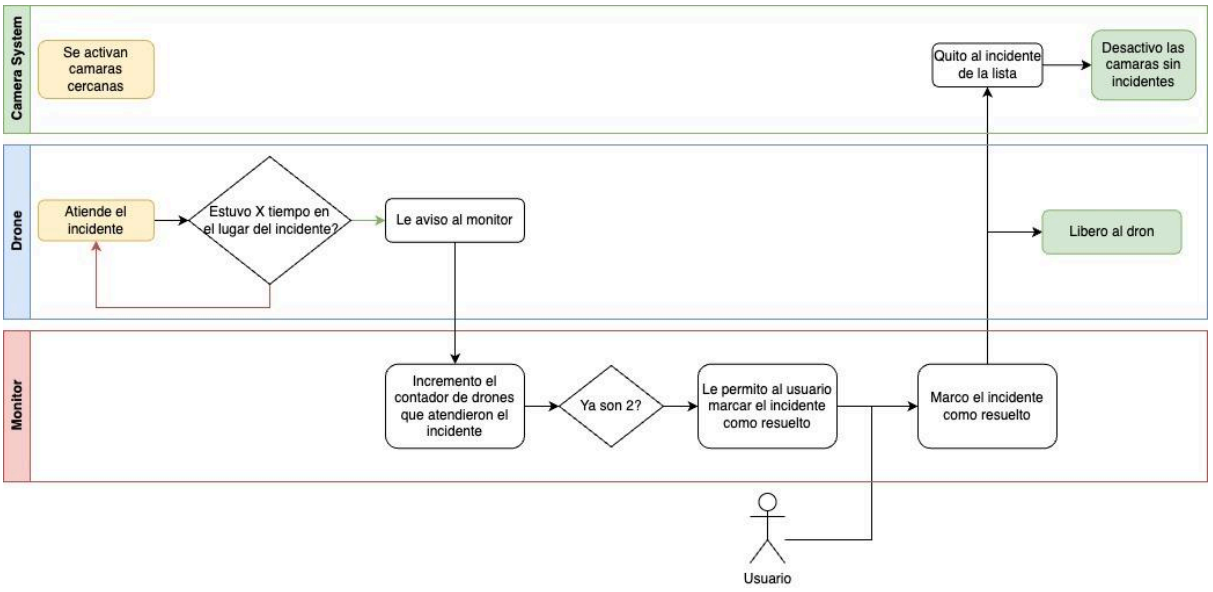
Se implementó un sistema que guarda la información relevante de todos los paquetes transmitidos por la red junto con su estampa de tiempo de una forma que pueda ser legible a la hora de buscar por mensajes antiguos.

Decisiones de diseño

Para llevar a cabo el proyecto se tomaron una serie de decisiones de diseño clave que permiten que todo el sistema funcione y que van más allá de los requerimientos iniciales. A continuación se detallan cada una de ellas, junto con su motivo.

Cierre de incidentes

Para que un incidente pueda ser cerrado, primero cada uno de los drones, debe considerar que está listo para cerrarse, una vez esto ocurra el operador del monitor tendrá la potestad de cerrarlo enviando a los drones de vuelta a su posición inicial. Esto le agrega la validación humana al cierre de un incidente minimizando el riesgo de falsos positivos y aprovechando la expertise de los operadores del gobierno.



Camaras lindantes

Para tener un registro completo de lo que ocurrió en una zona al momento en la que se haya reportado un incidente, no solo se contará con la cámara más cercana abocada a ese incidente, sino que todas aquellas que se encuentren en el radio próximo al mismo.

Especificaciones técnicas

Elección del lenguaje

Empleamos la versión 1.79.0 de RUST para la totalidad del desarrollo del sistema. RUST es un lenguaje robusto y seguro, muy preparado para resolver problemas de seguridad y concurrencia de una forma altamente performante. En nuestro caso nos permite asegurarnos de atrapar todos los casos, evitando la mayor cantidad de errores inesperados de modo sea lo suficientemente robusto para tratarse de un sistema con estándares de seguridad gubernamentales. Además al trabajar con muchas cosas al mismo tiempo, necesitaremos que la concurrencia sea uno de los pilares de nuestro sistema, por lo que una implementación en RUST cumple con todos los requerimientos en cuanto a lenguaje se refiere. Además se trata de un lenguaje en pleno crecimiento por lo que cada vez más desarrolladores son capaces de mantener sistemas como estos, lo que abarataría mucho los costos a largo plazo.

Elección de la versión del protocolo

Decidimos implementar la versión 3.1.1 de MQTT ya que no solo es una versión más que completa para lo que nuestro sistema necesita, sino que es una versión que se encuentra en el alcance de nuestro proyecto. A su vez, fueron usadas la mayoría de las características que el protocolo ofrece. Entre ellas:

- Wildcards en los filtros de los tópicos
- QoS 0 y 1
- Inicio de sesión
- Mensajes de reservados del sistema
- Identificadores de paquetes

Reconocimiento de Imágenes

Selección del servicio de IA

Para incorporar la tecnología de reconocimiento de imágenes al sistema de cámaras decidimos utilizar el servicio de **AWS Rekognition** junto con **AWS S3** para realizar la carga de imágenes.

Esta elección se debe a que algunos integrantes ya estaban familiarizados con AWS lo cual nos ahorraba mucho tiempo de aprendizaje. Además, AWS junto con Azure era el servicio más flexible ya que nos permite hasta 5000 consultas gratis por mes mientras que GCP solo nos permite 1000.

Análisis de costos

Para un sistema de 50 cámaras, donde cada cámara envía 10 imágenes por minuto, calculamos los siguientes costos para un año de funcionamiento:

- 500 imágenes por minuto
- 262.980.000 imágenes en un año

AWS	US$ 65.745	Amazon Rekognition – Pricing - AWS
Azure	US$ 105.192	Pricing - Computer Vision API | Microsoft Azure
GCP	US$ 262.980	Pricing | Cloud Vision API

Funcionamiento del reconocimiento de imágenes

AWS Rekognition nos permite analizar imágenes que tengamos guardadas en un bucket en S3. En nuestro caso utilizamos las funciones **PutObject** y **DetectLabels** subir archivos a S3 y pedirle a AWS que detecte ciertos labels y categorías en la imágen indicada.

Cada vez que se copie una nueva imagen al directorio local, esta misma será subida a un bucket de S3 y luego será analizada por AWS Rekognition. Si alguno de los labels especificados tiene un porcentaje de confianza mayor al porcentaje indicado en el archivo de configuración del sistema de cámaras entonces se generará un incidente donde está ubicada la cámara.

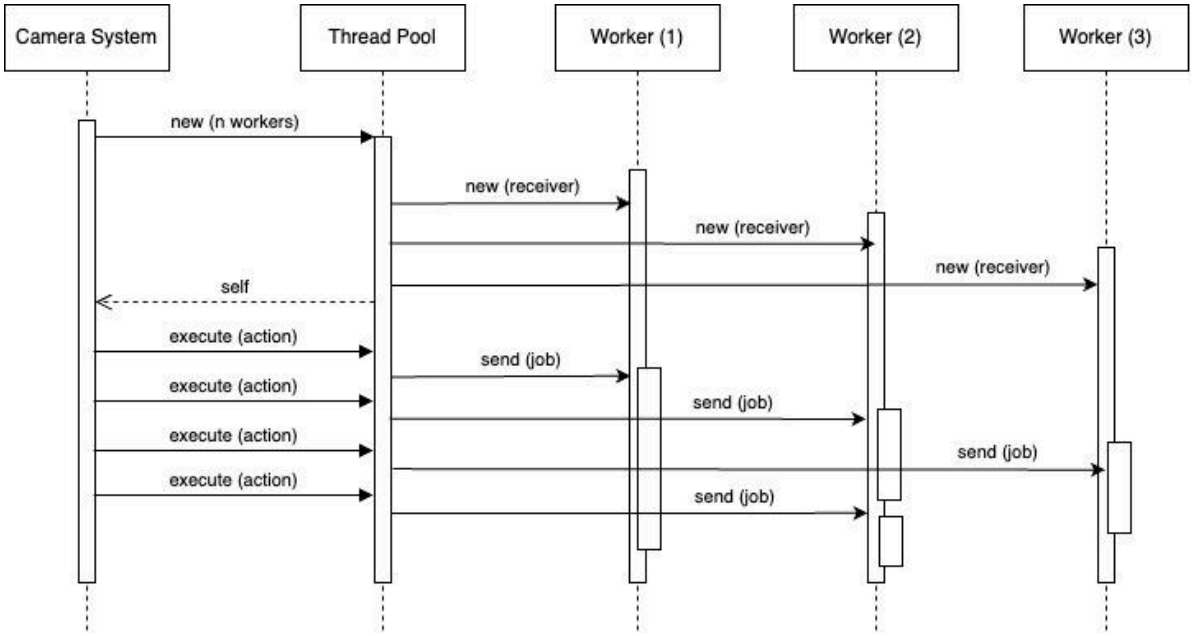
Estrategia de multithreading

Para resolver la concurrencia de llamadas a los servicios de AWS desarrollamos un módulo llamado **ThreadPool** siguiendo la documentación de Rust. Este módulo requiere especificar

la cantidad de threads que se dedicarán a resolver tareas, y crea internamente esa misma cantidad de **workers**.

El ThreadPool recibe las tareas denominadas **Jobs**, que se escriben en un canal escuchado por los distintos workers. El worker disponible que bloquee el channel tomará la tarea y la ejecutará de forma asíncrona.

Con este módulo, podemos enviarle peticiones de forma totalmente síncrona al ThreadPool, permitiendo que el trabajo se ejecute en segundo plano. En este caso, se encarga de cargar la imagen, leer las etiquetas y, si se determina que hay un incidente, notificarlo al sistema.



Dependencias

Para desarrollar el sistema en los plazos pactados con el gobierno, empleamos una serie de crates externas de las cuales debemos estar al tanto y actualizar periódicamente para tener los últimos parches de seguridad que eviten distintos tipos de vulnerabilidad.

CHRONO

Para la gestión de tiempos y fechas, utilizamos la librería “chrono” en su versión 0.4.

EGUI

Este crate, nos ayudó en toda la parte gráfica de nuestro sistema brindando una interfaz flexible y sencilla para todos los operadores del gobierno.

WALKERS

Esta librería forma parte de Egui y proporciona el soporte para poder colocar un mapa de la Ciudad de Buenos Aires en la interfaz del monitor y agregar entidades como incidentes, drones y cámaras de forma dinámica a partir de coordenadas.

AES-GCM

Para llevar a cabo la encriptación simétrica, utilizamos la librería aes-gcm en su versión 0.10.0. La librería elegida implementa el cifrado “Advanced Encryption Standard” en modo GCM. Este modo proporciona encriptación segura, junto con autenticación de datos, asegurando que cualquier modificación no autorizada de los datos en tránsito sea detectable. El crate elegido está diseñado para ser rápido y compatible con diferentes arquitecturas, asegurando que no haya un impacto significativo en el rendimiento.

AWS SDK

Incluye aws-config, aws-sdk-rekognition, aws-sdk-s3, aws-types y tokio. Estos crates nos permiten interactuar con los servicios de AWS que requerimos para el reconocimiento de imágenes.

Enlaces útiles

Privados

- Código fuente del sistema
- Documentación del proyecto

Públicos

- Especificación de MQTT 3.1.1
- Documentación de Chrono en Rust
- Código fuente de EGUI
- AWS SDK para Rust