Diseño y desarrollo de un microservicio para la gestión de información de monitorización y predicciones de tráfico en red

Autor: Enrique Fernández Sánchez
Tutor: Pablo Pavón Mariño

Universidad Politécnica de Cartagena (UPCT)

4 de enero de 2023

Índice

Introducción

- *Abstract*: Aplicación que permite almacenar muestras de monitorización de tráfico en red, y a su vez, generar predicciones futuras del tráfico de red, en función de la información almacenada.

Objetivos del proyecto

- Diseñar una **aplicación** siguiendo la metodología de **microservicios**.
- Investigar herramientas de **predicción de series temporales**.
- Investigar **opciones de almacenamiento** para **muestras temporales**.
- Utilizar **herramientas de documentación** que permitan conocer la estructura de la aplicación.

Microservicios

etsit escuela técnica superior de ingeniería de telecomunicación

Definición de microservicio

Sistemas que cumplen las siguientes premisas:

- Sistemas pequeños, independientes y poco "acoplados".
- Código fuente separado entre los diferentes servicios, no necesariamente mismo lenguaje.
- Los servicios se comunican entre sí utilizando APIs.
- Cada sistema es independiente, y responsable de su persistencia de datos.

Importancia de utilizar microservicios

- asdadd

API (Application Programing Interface)

- Una API permite a dos **componentes comunicarse** entre sí **mediante** una serie de **reglas**.
- Supone un "**contrato**" en el que **se establecen las solicitudes** y respuestas **esperadas en la comunicación**.

Tipos de API

Dependiendo de la implementación, distinguimos entre cuatro tipos:

- **SOAP**. Protocolo tradicional, usa mensajes XML (HTTP solo transporte). Ambos interlocutores deben conocer la estructura de los objetos. Poco flexible
- **RPC**. Basado en llamadas a procedimientos remotos.
- **WebSocket**. Solicitud moderna, usa objetos JSON y un canal bidireccional de comunicación.
- **REST**. Solución más popular y flexible. Usa los métodos HTTP. **Elegimos este tipo**.

Bases de datos

- En función del tipo de dato a almacenar, se distinguen dos bases de datos dentro de la aplicación:

Tipo relacional (`PostgreSQL`)

- Se almacenan datos que puedan tener una relación entre ellos. Por ejemplo: información de una red a monitorizar.
- Se organizan los datos en una serie de "relaciones", y estas se almacenan en una o más "tablas".
- Cada tabla dispone de una serie de columnas. La información se agrega en forma de filas a la tabla.

Tipo serie temporal (`InfluxDB`)

- Necesario para almacenar datos en forma de serie temporal de manera eficiente.
- En comparación con el tipo relacional, la fila de datos esta identificada por un valor temporal.
- Algunas ventajas:
  - Útil para almacenar datos de telemetría.
  - Datos comprimidos automáticamente.

Lenguaje & frameworks

Python

Lenguaje de programación orientado a objetos, interpretado y de alto nivel. Muy popular en los siguientes ambitos:

- Aplicaciones web.
- Data Science
- Inteligencia Artificial

FastAPI

Framework moderno y rápido para construir APIs. Características:

- Rápido: rendimiento equivalente a otros lenguajes (NodeJS o Go).
- Intuitivo: soporta autocompletado.
- Robusto: herramienta Swagger/ReDoc automática.

Lenguaje & frameworks

Prophet

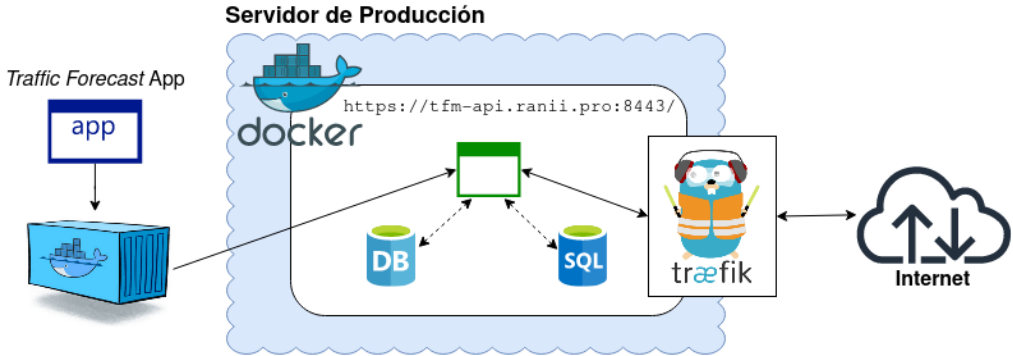
Framework del lenguaje de programación Python, desarrollado por Meta (Facebook). Agrupa una serie de procedimientos que permiten realizar predicciones en un dataset temporal. Características:

- Permite encontrar y tener en cuenta efectos no lineales (tendencias diarias, semanales, mensuales...).
- Permite predecir datos en días vacacionales.
- Basado en inferencia estadística, más eficiente que si utilizáramos técnicas de *Machine Learning*.

Despliegue en producción

Herramientas utilizadas para el despliegue del sistema en un entorno de producción:

- Docker
- docker-compose
- Traefik



Descripción API REST (I)

Para la aplicación se implementan los diferentes agentes utilizando dos metodologías:

1. **CRUD**
   - Aquellas que siguen la definición `Create`, `Read`, `Update` & `Delete`.
   - *Ejemplo*: redes o interfaces.
2. **No CRUD**
   - Aquellas que no tienen por qué seguir las reglas CRUD.
   - *Ejemplo*: ejecutar predicción de red.

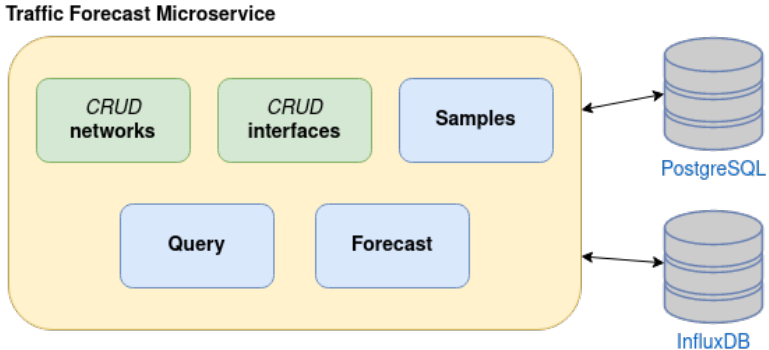
Agentes

Se identifican los diferentes "agentes" presentes en el sistema:

- **Redes** (`networks`), corresponde con una red que contiene interfaces a monitorizar.
- **Interfaces** (`interfaces`), corresponde con las interfaces de red que queremos monitorizar.
- **Muestras de monitorización** (`samples`). Valor de tráfico asociado a una interfaz en un intervalo determinado.

Descripción API REST (II)

Esquema de la aplicación implementada



Implementación (I)

content...

Implementación (II)

content...

OpenAPI. Swagger & ReDoc

- FastAPI permite generar de manera automática una serie de rutas que sirven como documentación del proyecto.
- En estas herramientas se muestran al usuario la información de:
  - Rutas de la aplicación
  - Parámetros de entrada y de salida para una petición.
  - Modelos de datos para cada una de las rutas.
  - Posibles códigos HTTP de respuesta para las peticiones.
- Permite al usuario realizar peticiones al servidor, y ver su resultado, desde la misma herramienta.

Acceso a las herramientas de documentación

- Swagger: `https://tfm-api.ranii.pro:8443/docs`
- ReDoc: `https://tfm-api.ranii.pro:8443/redoc`

Modelos de datos. SQL

- Definimos los datos y su **representación dentro** de nuestra **aplicación**.
- **En función** del **tipo de dato**, almacenaremos en **base de datos tipo relacional** (`SQL`) o **base de datos tipo serie temporal** (`InfluxDB`).

Tablas utilizadas en la base de datos SQL

Networks
`id_network`: Int, Public Key, Unique
`name`: String
`description`: String
`ip_red`: String
`influx_net`: String

Interfaces
`id_interface`: Int, Public Key, Unique
`name`: String
`description`: String
`influx_if_rx`: String
`influx_if_tx`: String
`network`: Int, Foreign Key

Modelos de datos. `InfluxDB`

- En `InfluxDB` almacenamos las muestras de tráfico en red.

Configuración InfluxDB para el sistema

InfluxDB	Descripción
`measurement`	Red a monitorizar, valor almacenado en `Networks::influx_net`
`fields`	Nombre del valor a monitorizar (*link_count*)
`tags`	Solo disponemos un `tag`, llamado `interface`, contiene la información del identificador de una interfaz
`points`	Corresponde con el valor numérico del campo `field`.

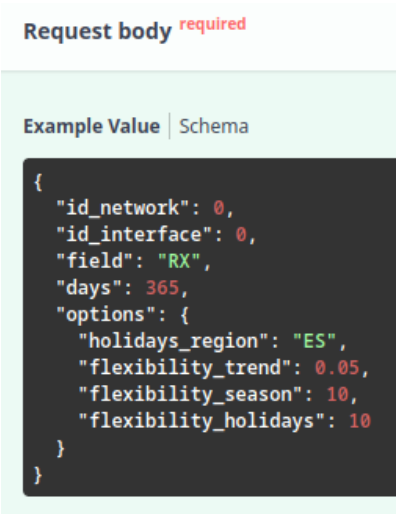
Predicción de tráfico de red (I)

- El sistema es capaz de ejecutar predicciones de tráfico de red, en función de las muestras de monitorización previamente almacenadas en el sistema.

Pasos para realizar una predicción de tráfico

1. Creamos una red a monitorizar. `POST` a `/networks`.
2. Creamos una interfaz a monitorizar. `POST` a `/networks/<net>/interfaces`
3. Importamos datos de monitorización. Dos maneras:
   - `POST` a ruta: `/samples/<net>/import_topology`
   - `POST` a ruta: `/samples/<net>/import_interface/<if>`
4. Ejecutamos predicción de tráfico en red. Configuramos la predicción y lanzamos la petición `POST` a `/forecast`.

Predicción de tráfico de red (II)



Parámetros para la ejecución de la predicción de tráfico

- `field`. Campo para elegir si RX o TX.
- `days`. Número de días a predecir.
- `options`. Opciones que modifican la flexibilidad de la predicción:
  - `holidays_region`
  - `flexibility_trend`
  - `flexibility_season`
  - `flexibility_holidays`

Resumen rutas HTTP (I)

CRUD: networks (/networks)

- **Información** de **todas** las redes:
  GET - `/networks`
- **Crear** una red:
  POST - `/networks`
- **Información** de **una** red:
  GET - `/networks/<net_id>`
- **Eliminar** una red:
  DELETE - `/networks/<net_id>`
- **Actualizar** una red:
  PATCH - `/networks/<net_id>`

CRUD: interfaces (/networks/<id1>/interfaces)

- **Información** de **todas** las interfaces:
  GET - `/networks/id/interfaces`
- **Crear** una interfaz:
  POST - `/networks/id/interfaces`
- **Información** de **una** interfaz:
  GET - `/../interfaces/<id2>`
- **Eliminar** una interfaz:
  DELETE - `/../interfaces/<id2>`
- **Actualizar** una interfaz:
  PATCH - `/../interfaces/<id2>`

Resumen rutas HTTP (II)

Samples

- **Importar** datos de topología con **más de una interfaz**:
  POST - `/samples/id/import_topology`
- **Importar** datos de **una interfaz**:
  POST - `/samples/id/import_interface/id`

Query Samples

- **Consultar datos** de monitorización **almacenados**:
  GET - `/query/`

Forecast

- **Ejecutar predicción** de tráfico en red:
  POST - `/forecast/`

Validación del sistema (I)

Para demostrar el funcionamiento del sistema, se realiza una batería de pruebas de las funcionalidades implementadas. Dichas pruebas son:

1. Crear una red a monitorizar.
2. Crear una interfaz dentro de una red a monitorizar.
3. Cargar muestras en interfaz de red.
4. Cargar una topología de red sobre una red a monitorizar.
5. Consultar datos de monitorización.
6. Ejecutar una predicción de un año.

Estas pruebas corresponden con las **pruebas unitarias** del sistema.

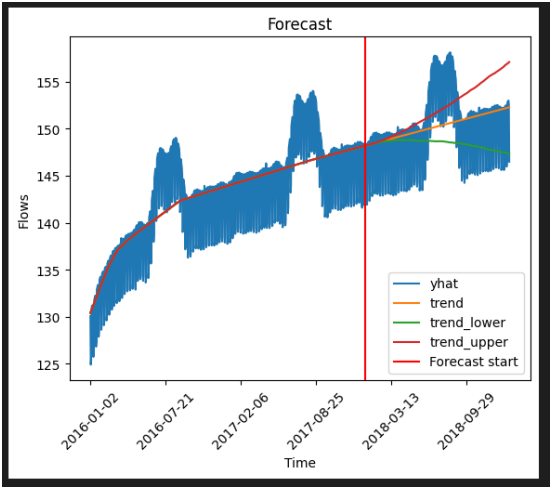
Validación del sistema (II)

Demo!

Realizamos una validación del sistema.

`https://tfm-api.ranii.pro:8443/docs`

- *Objetivo*: Completar una predicción de un año.
- *Requisito*: Asumir sistema sin datos almacenados.
- *Resultado*: Una predicción similar a la de la figura.



Conclusiones

- Alcanzados todos los objetivos propuestos.
- Se ha desarrollado una **microservicio completo**, permitiendo ser implementado por otras aplicaciones.
- El sistema es **capaz** de **almacenar** muestras de monitorización, además de poder **generar predicciones** de tráfico en red en la escala temporal que el usuario solicite.

Propuestas futuras

- Permitir la **importación** de **datos de monitorización** de **herramientas especificas** de planificación de red.
- Añadir **funcionalidad de SSE** (Server Side Event) para **tareas** que requieran un **largo tiempo de ejecución**.
- Extender la funcionalidad de las **predicciones**, permitiendo hacer selección **más selectiva**, o añadir **filtrados extra**.

Bibliografía

- La contenida en la memoria del proyecto: páginas 53 - 54

Muchas gracias por su atención

¿Preguntas?

Enlace a la aplicación:
`https://tfm-api.ranii.pro:8443/`