ELK-Stack exercise

Juan Sebastian Medina Cordero

Universidad Icesi

Ingeniería Telemática

Plataformas II

[**Introduction: 3**](#_yb9oojmz2we9)

[**AI-Prompt 3**](#_klqgiupqfyub)

[**Preliminary Configuration 4**](#_fatzcq43ezad)

[Secret Creation: 5](#_ict9mv1kp900)

[Microservices Deployment: 5](#_r083vtj76mu6)

[**ELK Manifests Creation 7**](#_gfnqljkt6g85)

[Elasticsearch 7](#_hna7o3els13e)

[Deployment: 8](#_d5xxuuy3lqs0)

[Service: 8](#_xrdfi6twpon5)

[PVC: 9](#_kafgz15ne7z6)

[Logstash: 9](#_s09wf5tryphe)

[Deployment: 10](#_sxfc5p7ws1zq)

[Service: 11](#_p4458mj5e1qo)

[Configmap: 11](#_s2jtkcrd9fiv)

[Kibana 12](#_uazcp6y1zw8v)

[Deployment: 13](#_xen8uw1lsub3)

[Service: 13](#_5z8jo2t9lvsw)

[**Scripts 13**](#_7cb5sainl2ty)

[**Usage 15**](#_piz5w2lshrw0)

# 

# Introduction:

The following project consists of implementing an ELK-Stack on an existing project of our liking, this by taking advantage of artificial intelligence tools. In this case we chose the microservice-app-example project and for our artificial intelligence tool we chose chatGPT.

# AI-Prompt

Estoy trabajando en un proyecto de microservicios desplegado en Minikube. Ya cuento con todas las imágenes de los microservicios, así como los manifiestos deployment.yml, service.yml y configmap.yml para cada componente. La estructura de mis archivos es la siguiente:

manifests/

├── auth-api/

│ ├── deployment.yml

│ └── service.yml

├── config/

│ ├── configmap.yml

│ └── prometheus-configmap.yml

├── frontend/

│ ├── deployment.yml

│ └── service.yml

├── grafana/

│ ├── deployment.yml

│ └── service.yml

├── log-message-processor/

│ └── deployment.yml

├── prometheus/

│ ├── deployment.yml

│ └── service.yml

├── redis/

│ ├── deployment.yml

│ └── service.yml

├── todos-api/

│ ├── deployment.yml

│ └── service.yml

├── users-api/

│ ├── deployment.yml

│ └── service.yml

└── zipkin/

├── deployment.yml

└── service.yml

Ahora quiero extender esta solución incorporando el stack ELK (Elasticsearch, Logstash, Kibana) para capturar y visualizar los logs generados por los microservicios. Para esto:

Requisitos:

Separación de namespaces:

microservices: para auth-api, todos-api, users-api, frontend, redis, etc.

observability: para prometheus, grafana, zipkin, y todo el stack ELK (elasticsearch, logstash, kibana).

Objetivo:

Desplegar los componentes en sus respectivos namespaces.

Configurar los servicios y variables de entorno necesarios para que los microservicios generen logs que sean capturados por Logstash.

Visualizar los logs de forma local en Kibana sobre Minikube.

Consulta:

¿Me puedes dar un paso a paso detallado para hacer esto en Minikube, considerando la separación de namespaces, el orden correcto de despliegue, y la configuración necesaria (por ejemplo, volúmenes, servicios, configuración de Filebeat o Logstash, etc.) para integrar correctamente el stack ELK con mis microservicios?

También necesito que indiques:

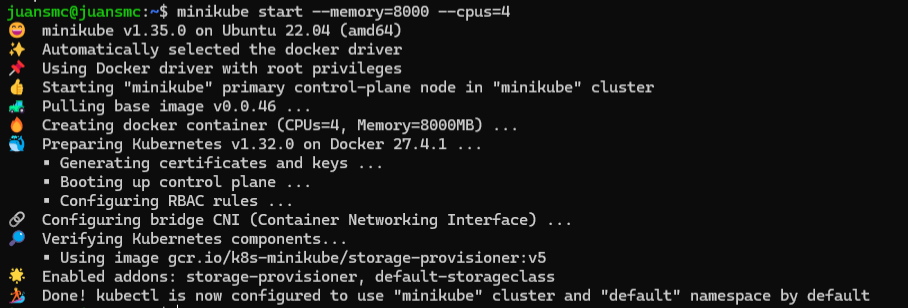
Cómo exponer Kibana para visualizar los logs desde el navegador.

Cómo enviar logs de mis microservicios hacia Logstash.

Qué configuración debo tener en los deployment.yml o configmaps para que el envío de logs funcione correctamente.

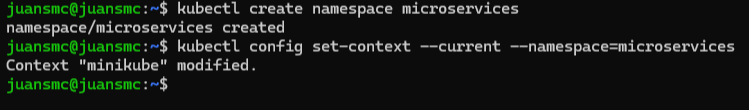
# Preliminary Configuration

First, it is necessary to start Minikube, and given that this stack appears to have high resource consumption, especially when combined with the microservices application. We will allocate approximately 8 gigabytes of RAM and 4 CPU cores to the cluster.



Next, it is important to define the namespaces that will be used. One namespace will be dedicated to deploying the various microservices, and another will be used for deploying the logging services (ELK stack).

For the purpose of using bash scripts, the current context will be set to the microservices namespace.





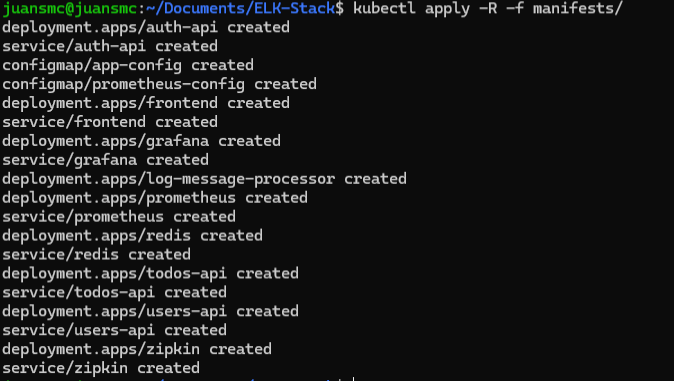
## Secret Creation:

We create a Kubernetes Secret to avoid exposing the JWT token directly within the manifests. This adds a layer of security to our deployment by keeping sensitive data out of version control.

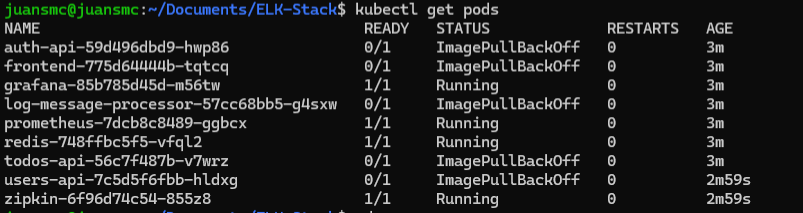


## Microservices Deployment:

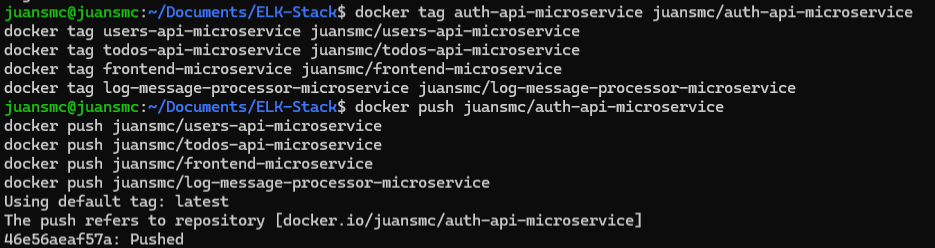
Since the various manifests were already created for the cloud pipelines workshop, we reused them for deploying the application. However, we modified the image source to use local Docker images.



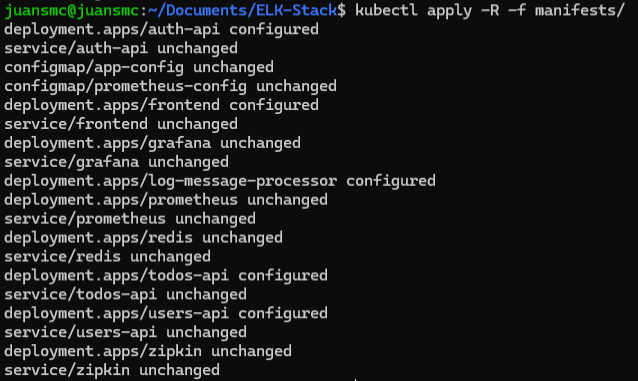
Upon inspecting the pods, we found that the images could not be pulled because Minikube cannot access Docker images directly.



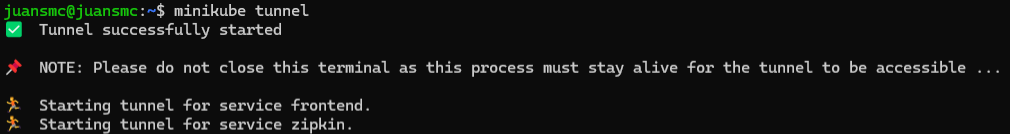
To solve this issue, we had to push the Docker images to Docker Hub.



Once the images were published to Docker Hub, we executed the kubectl apply command on the existing manifests and made the necessary updates to the microservice deployment configurations.



A tunnel was opened to access the services using a LoadBalancer type. In this case, we focused on exposing the frontend service in order to interact with the application through the browser.



# ELK Manifests Creation

We proceeded to create the manifests for the components of the ELK Stack, considering the possibility of automating the resource creation process.

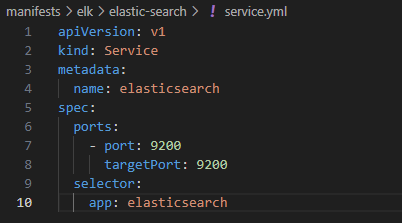
## Elasticsearch

Elasticsearch is a NoSQL database designed for efficient search operations. It is responsible for storing, indexing, and allowing search over the logs.

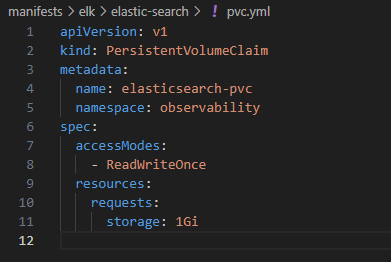
* It receives and stores logs sent by Logstash or Filebeat.
* It indexes the logs to enable fast and complex search queries.
* Since it stores the different logs persistently, we define a PersistentVolumeClaim (PVC).

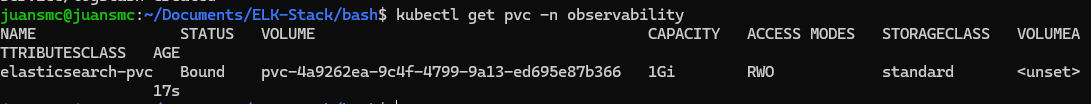
### Deployment:

### Service:



### PVC:





## Logstash:

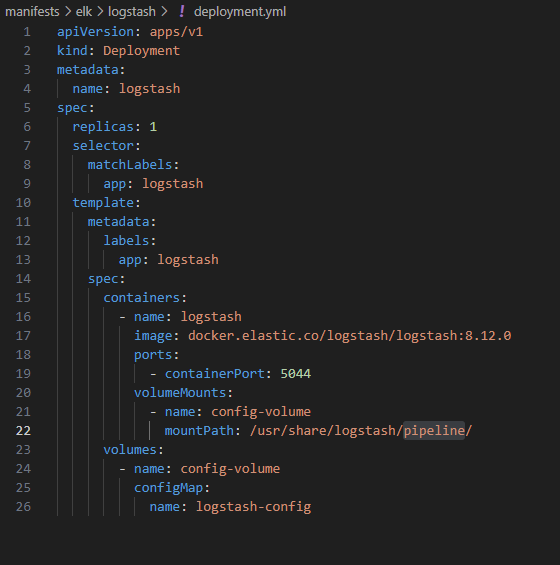
Logstash is a data processing pipeline that ingests, transforms, and forwards data to Elasticsearch (or other destinations).

* It can receive logs from multiple sources (e.g., Redis, files, Kafka).
* It is capable of transforming and filtering data using rules such as regex or JSON field extraction.

For this setup, we configured Logstash to receive preformatted logs generated by the todos-api, which are sent to Redis. Logstash pulls the logs from Redis and forwards them to Elasticsearch.

### 

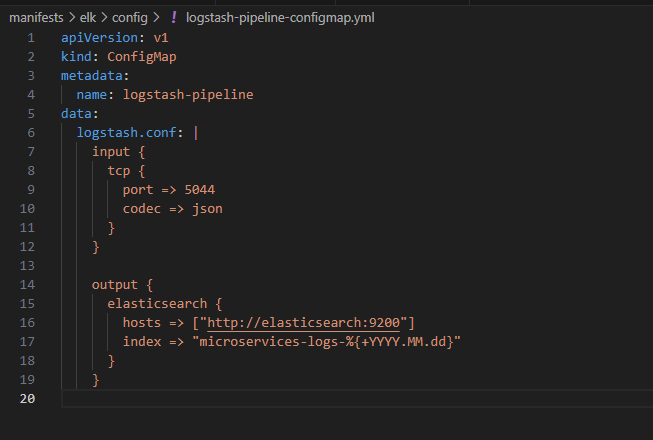
### Deployment:



### Service:

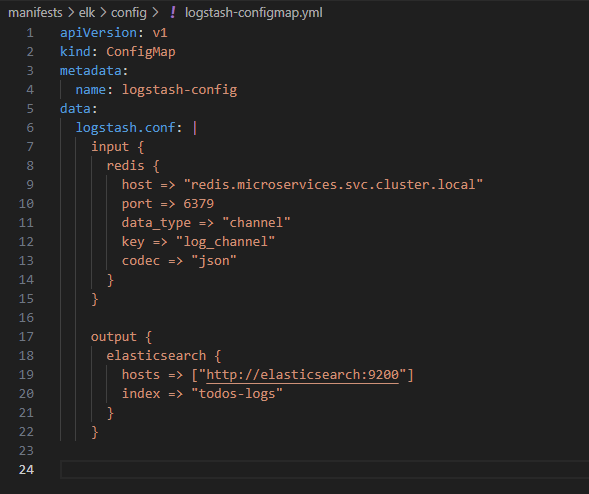
### Configmap:

First version:

This version was meant for receiving different logs created directly from the different microservices but it would make the practice far more complex.

Definitive version:

So this version was used taking advantage of the already created and formatted logs sent to redis-cache.

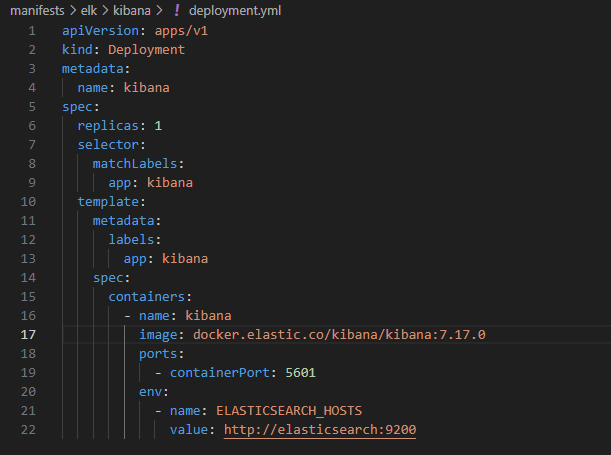


## Kibana

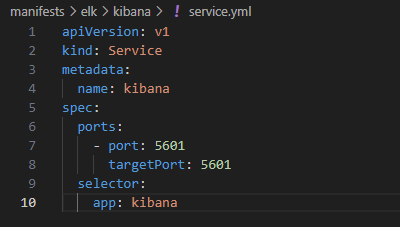
Kibana is a web interface used for visualizing and exploring the data stored in Elasticsearch.

* It allows users to create dashboards with charts, tables, and metrics.
* It supports search and filtering of logs by fields and date ranges.
* It helps with error debugging and system analysis.

### Deployment:



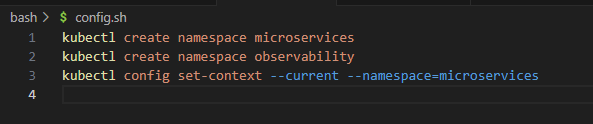
### Service:



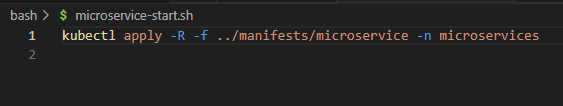
# Scripts

Some scripts were created to make it easier to test and return to different states of the development.  
The first one is for creating the namespaces and setting the current context so the following scripts work.

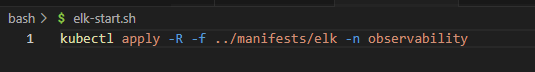
As said before there are two namespaces, one for the microservices deployment and other for the logging of the ELK-stack.



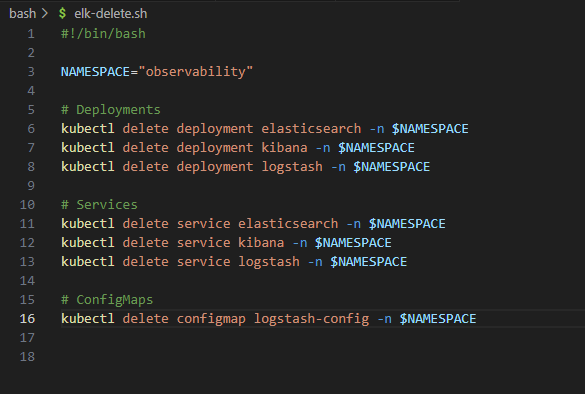
The following is for running all of the manifests of the microservices namespace which were created previously.



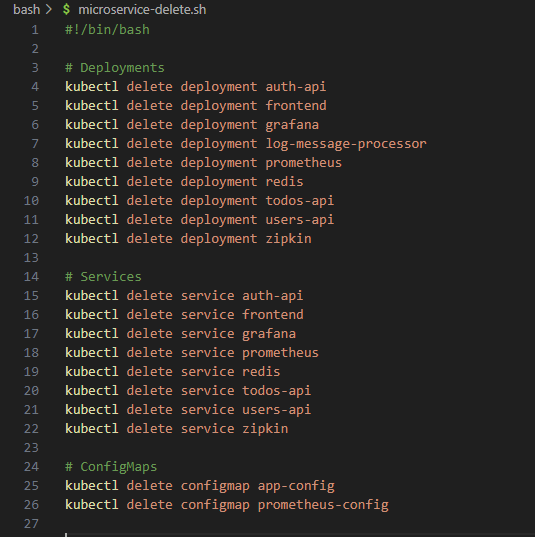
And aside we have another one for running the different elements of the ELK-Stack.



We also have one script for removing all of the created resources for the ELK-stack including deployments, services and configmaps so if a mistake was made in the manifest creation part, it would be easier to make changes.

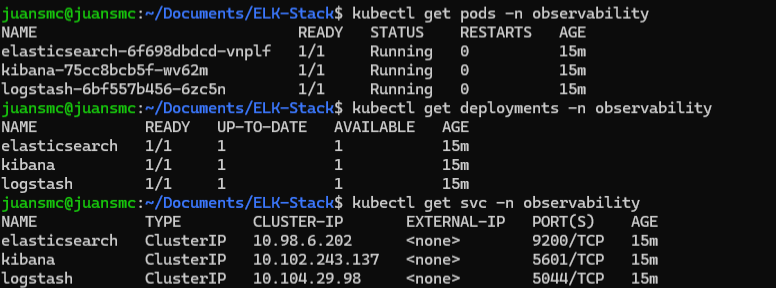


As well as we have one for deleting all of the resources referring to microservices also including deployments, services and configmaps.

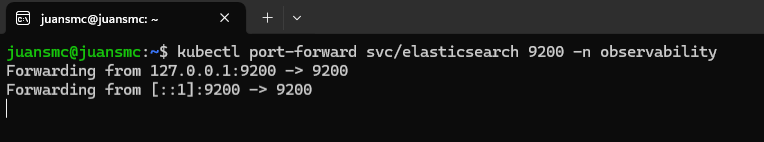


# Usage

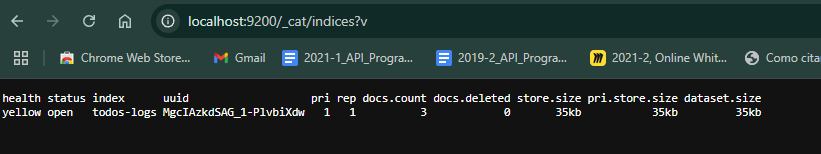
After running the elk-start.sh we can see that everything is mounted correctly.



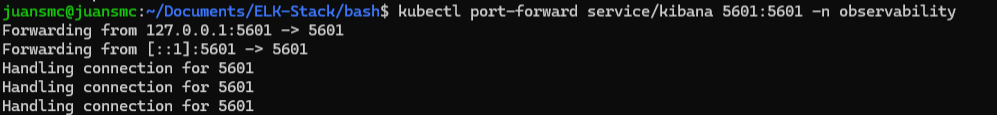
The next step is checking if logstash is passing correctly the different logs retrieved from redis. So we will check the elasticsearch service by port-forwarding and checking its endpoint.



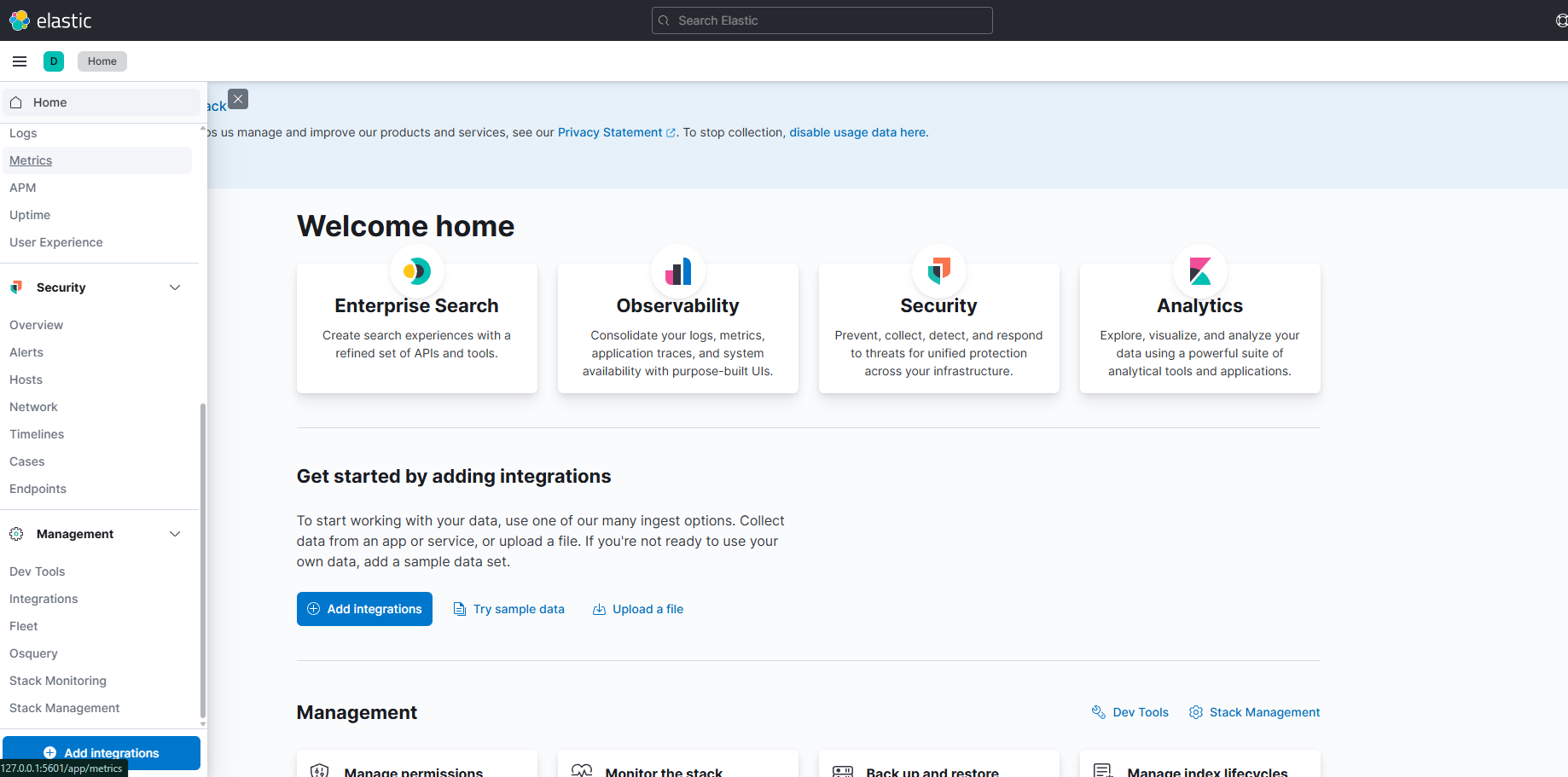
As we can see elasticsearch is retrieving the logs correctly and the index is called todos-logs.



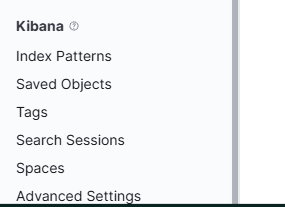
Now it's time to configure Kibana for visualizing this information in a good interface.



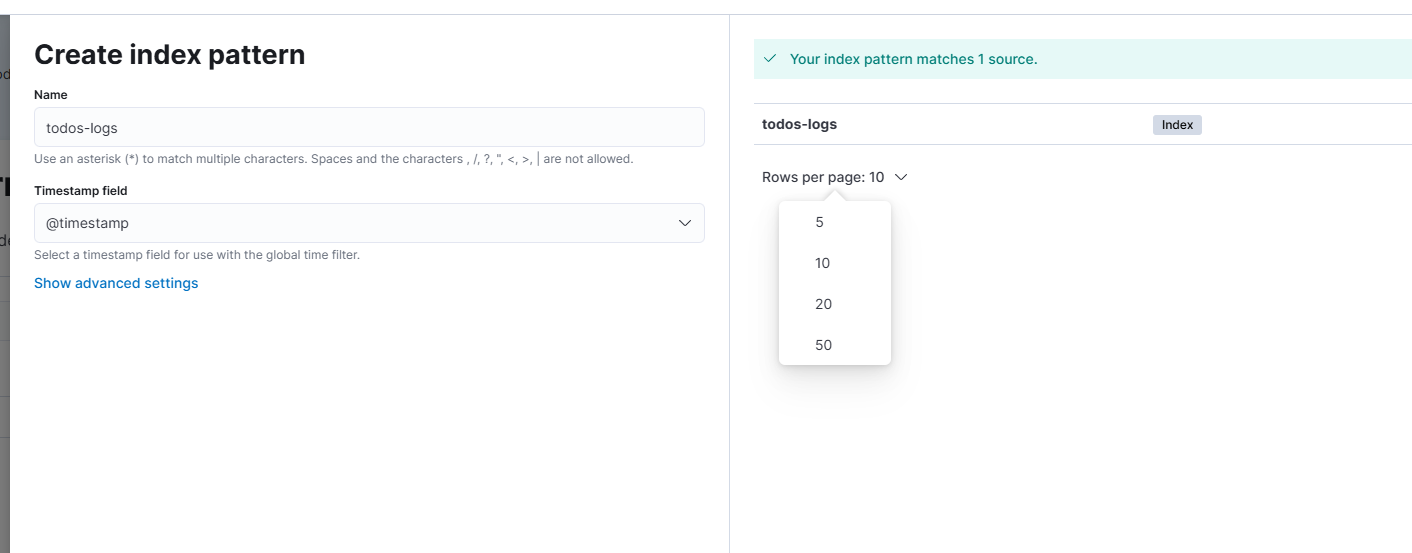
When opening kibana we have to get into stack management in the left menu.



And then inside the next page scroll in the left menu to kibana and index patterns.



Here we have to create a new index pattern that has the name of the index we saw in the elastic search request.



And finally we can see all of the retrieved logs as documents and we can generate our desired visualizations

