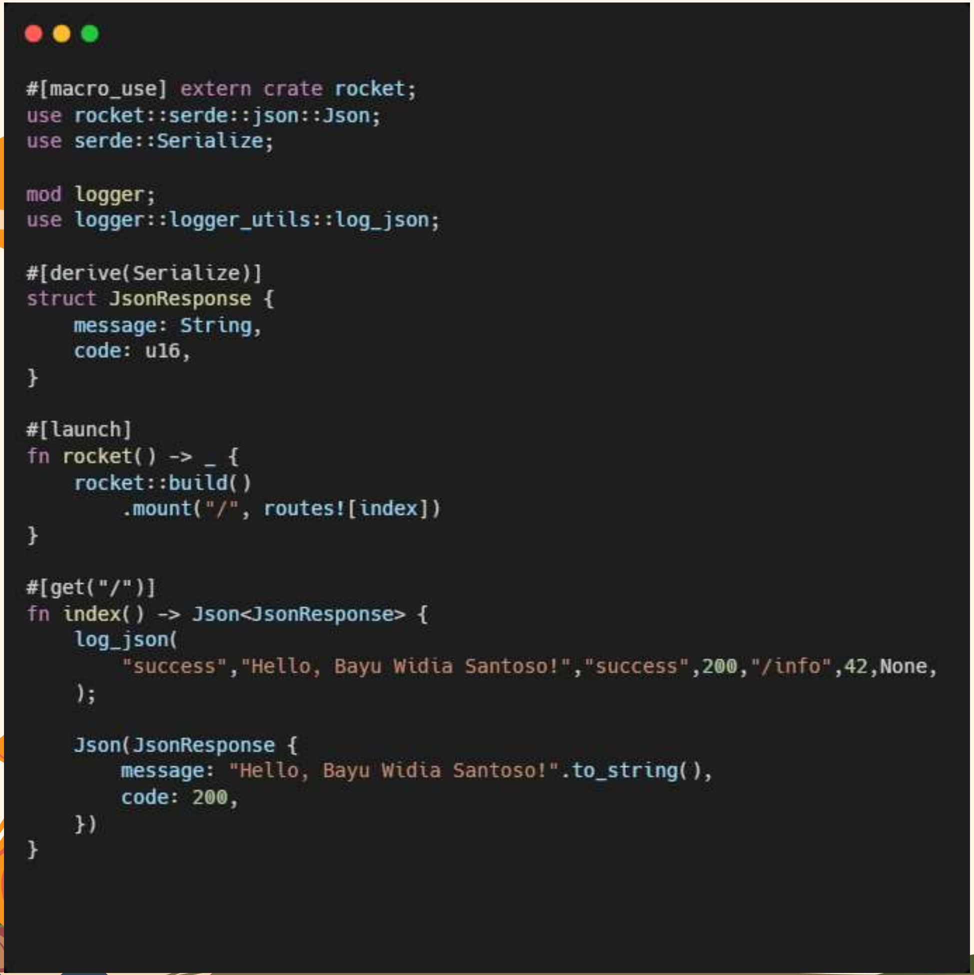
Exploring the ELK Stack for Monitoring APIs with Rust

Elastic Search

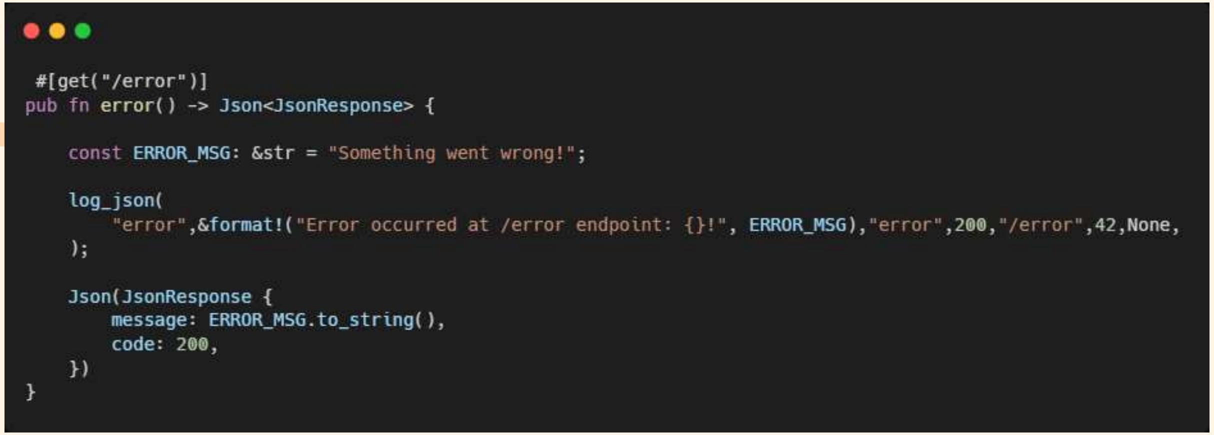
docker

Create a File in Rocket

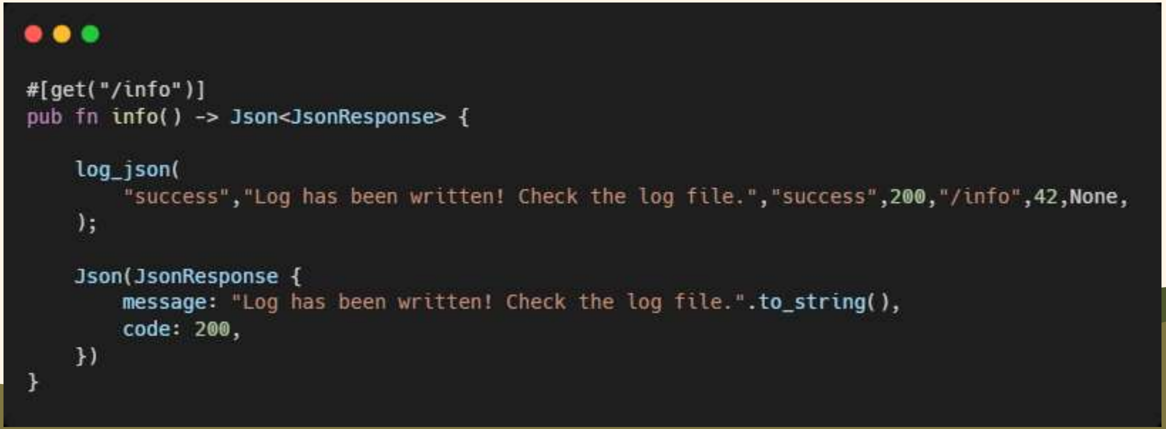


```rust
#[macro_use] extern crate rocket;
use rocket::serde::json::Json;
use serde::Serialize;

mod logger;
use logger::logger_utils::log_json;

#[derive(Serialize)]
struct JsonResponse {
    message: String,
    code: u16,
}

#[launch]
fn rocket() -> _ {
    rocket::build()
        .mount("/", routes![index])
}

#[get("/")]
fn index() -> Json<JsonResponse> {
    log_json(
        "success","Hello, Bayu Widia Santoso!","success",200,"/info",42,None,
    );

    Json(JsonResponse {
        message: "Hello, Bayu Widia Santoso!".to_string(),
        code: 200,
    })
}
```

Then, add these two methods to call it, distinguishing between "info" and "error".

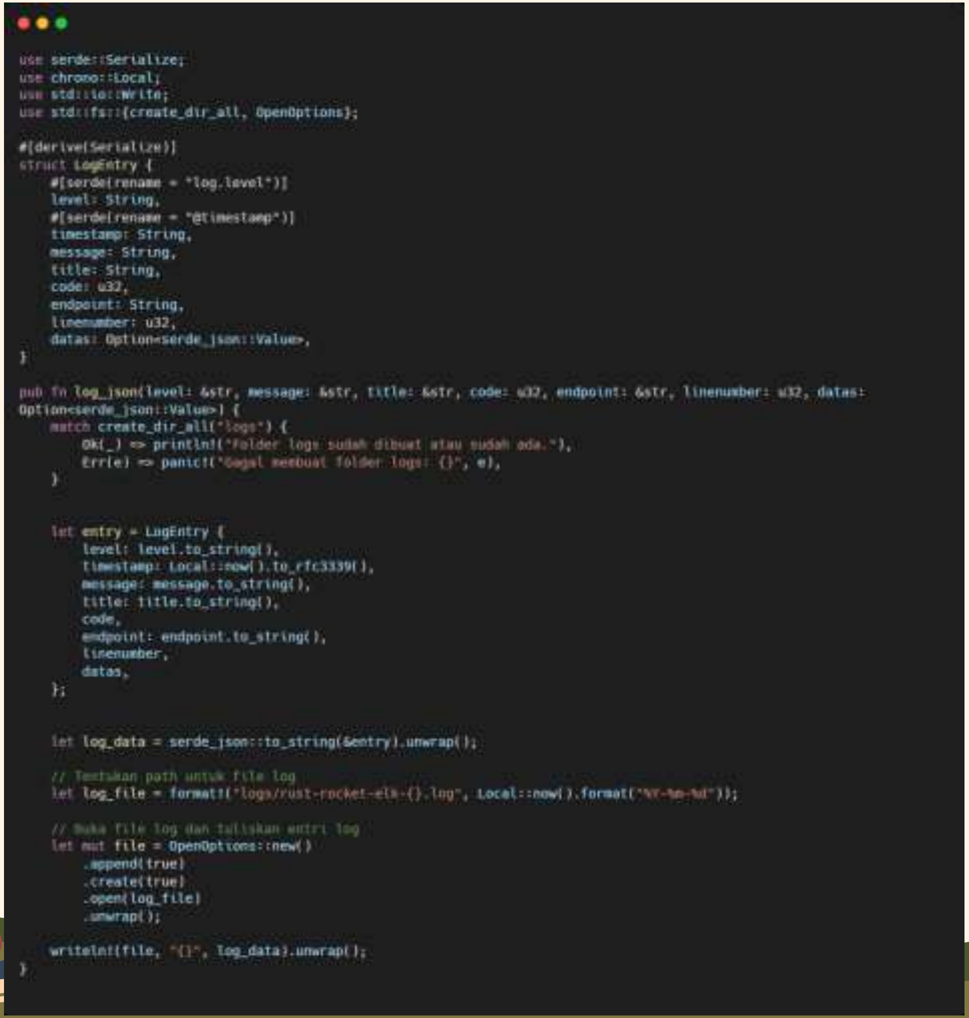


```rust
 #[get("/error")]
pub fn error() -> Json<JsonResponse> {

    const ERROR_MSG: &str = "Something went wrong!";

    log_json(
        "error",&format!("Error occurred at /error endpoint: {}!", ERROR_MSG),"error",200,"/error",42,None,
    );

    Json(JsonResponse {
        message: ERROR_MSG.to_string(),
        code: 200,
    })
}
```



```rust
#[get("/info")]
pub fn info() -> Json<JsonResponse> {

    log_json(
        "success","Log has been written! Check the log file.","success",200,"/info",42,None,
    );

    Json(JsonResponse {
        message: "Log has been written! Check the log file.".to_string(),
        code: 200,
    })
}
```

Create a File name logger_utils.rs



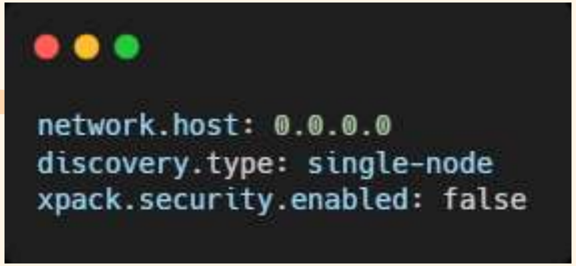
```rust
use serde::Serialize;
use chrono::Local;
use std::io::Write;
use std::fs::{create_dir_all, OpenOptions};

#[derive(Serialize)]
struct LogEntry {
    #[serde(rename = "log.level")]
    level: String,
    #[serde(rename = "@timestamp")]
    timestamp: String,
    message: String,
    title: String,
    code: u32,
    endpoint: String,
    linenumber: u32,
    datas: Option<serde_json::Value>,
}

pub fn log_json(level: &str, message: &str, title: &str, code: u32, endpoint: &str, linenumber: u32, datas:
Option<serde_json::Value>) {
    match create_dir_all("logs") {
        Ok(_) => println!("Folder logs sudah dibuat atau sudah ada."),
        Err(e) => panic!("Gagal membuat folder logs: {}", e),
    }


    let entry = LogEntry {
        level: level.to_string(),
        timestamp: Local::now().to_rfc3339(),
        message: message.to_string(),
        title: title.to_string(),
        code,
        endpoint: endpoint.to_string(),
        linenumber,
        datas,
    };


    let log_data = serde_json::to_string(&entry).unwrap();

    // Tentukan path untuk file log
    let log_file = format!("logs/rust-rocket-elk-{}.log", Local::now().format("%Y-%m-%d"));

    // Buka file log dan tuliskan entri log
    let mut file = OpenOptions::new()
        .append(true)
        .create(true)
        .open(log_file)
        .unwrap();

    writeln!(file, "{}", log_data).unwrap();
}
```

Once that's done, we will prepare some ELK stack configurations.

We will create several configuration files, including:

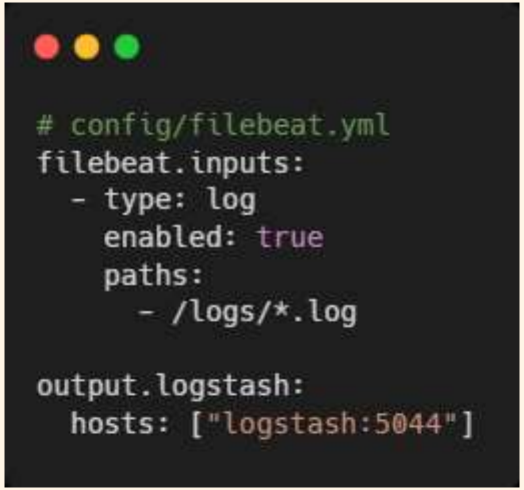
Elasticsearch
Filebeat
Kibana
Logstash

Elasticsearch



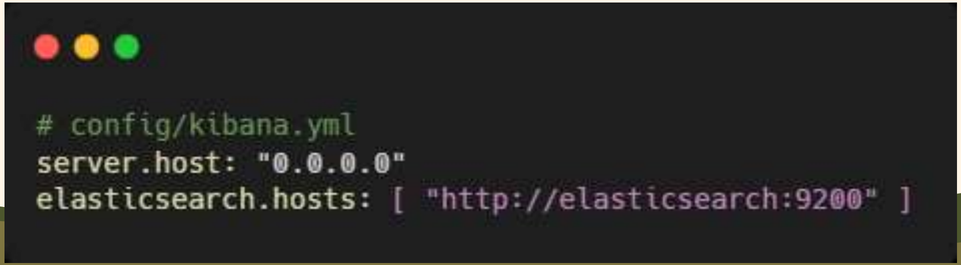
```
network.host: 0.0.0.0
discovery.type: single-node
xpack.security.enabled: false
```

Filebeat



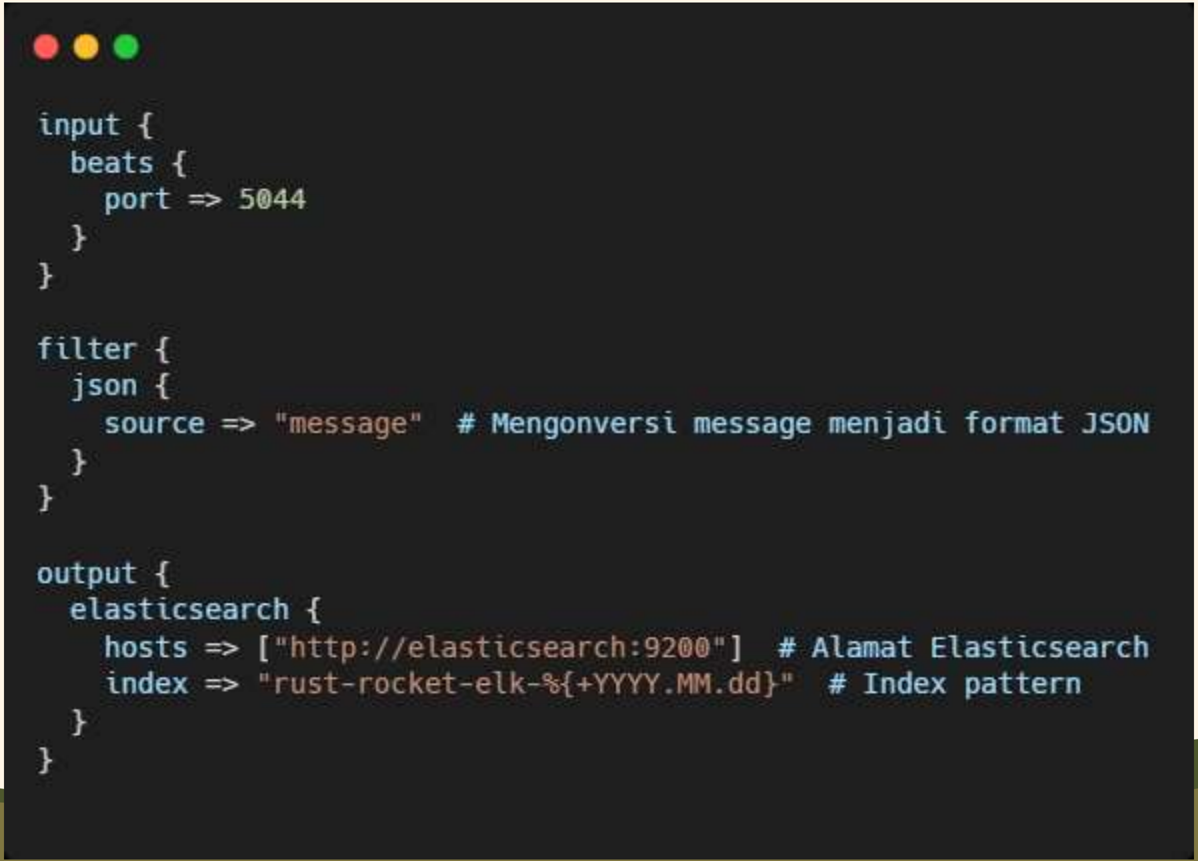
```
# config/filebeat.yml
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /logs/*.log

output.logstash:
  hosts: ["logstash:5044"]
```

kibana



```
# config/kibana.yml
server.host: "0.0.0.0"
elasticsearch.hosts: [ "http://elasticsearch:9200" ]
```

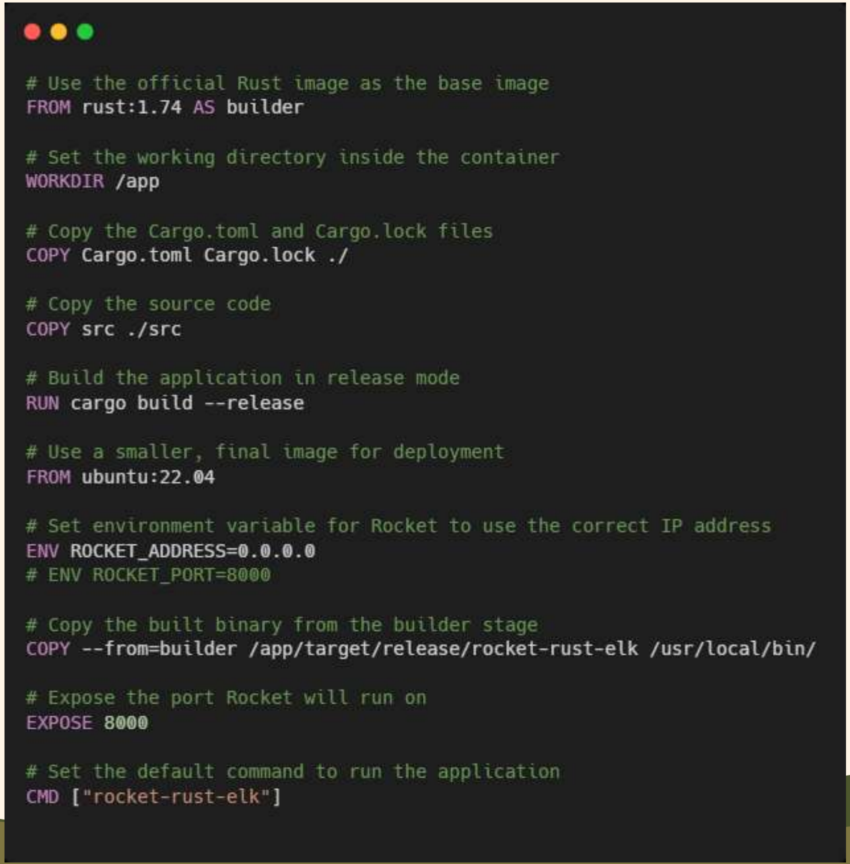
Logstash



```
input {
  beats {
    port => 5044
  }
}

filter {
  json {
    source => "message"  # Mengonversi message menjadi format JSON
  }
}

output {
  elasticsearch {
    hosts => ["http://elasticsearch:9200"]  # Alamat Elasticsearch
    index => "rust-rocket-elk-%{+YYYY.MM.dd}"  # Index pattern
  }
}
```

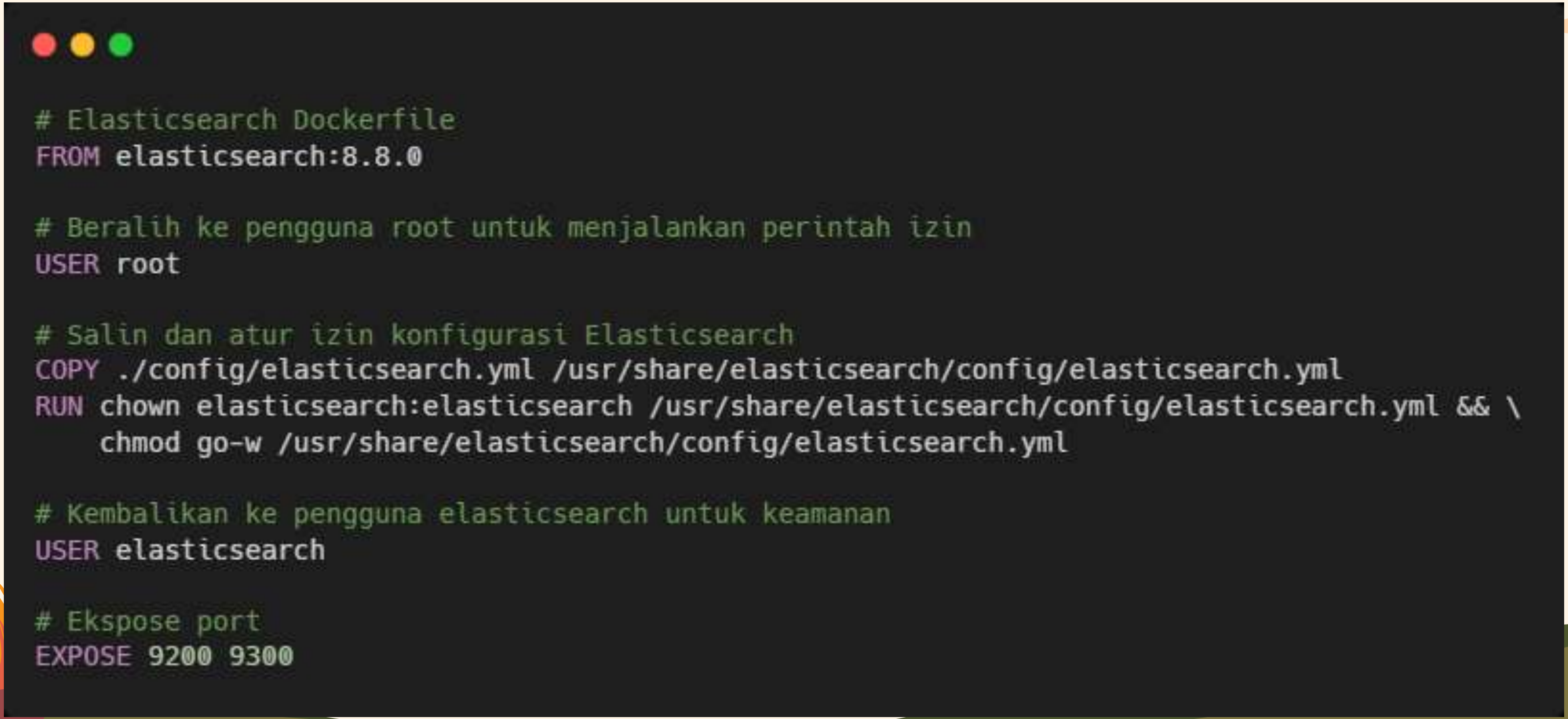
Once it's done, we will prepare the ELK stack to run using Docker.

Create a Dockerfile for the application



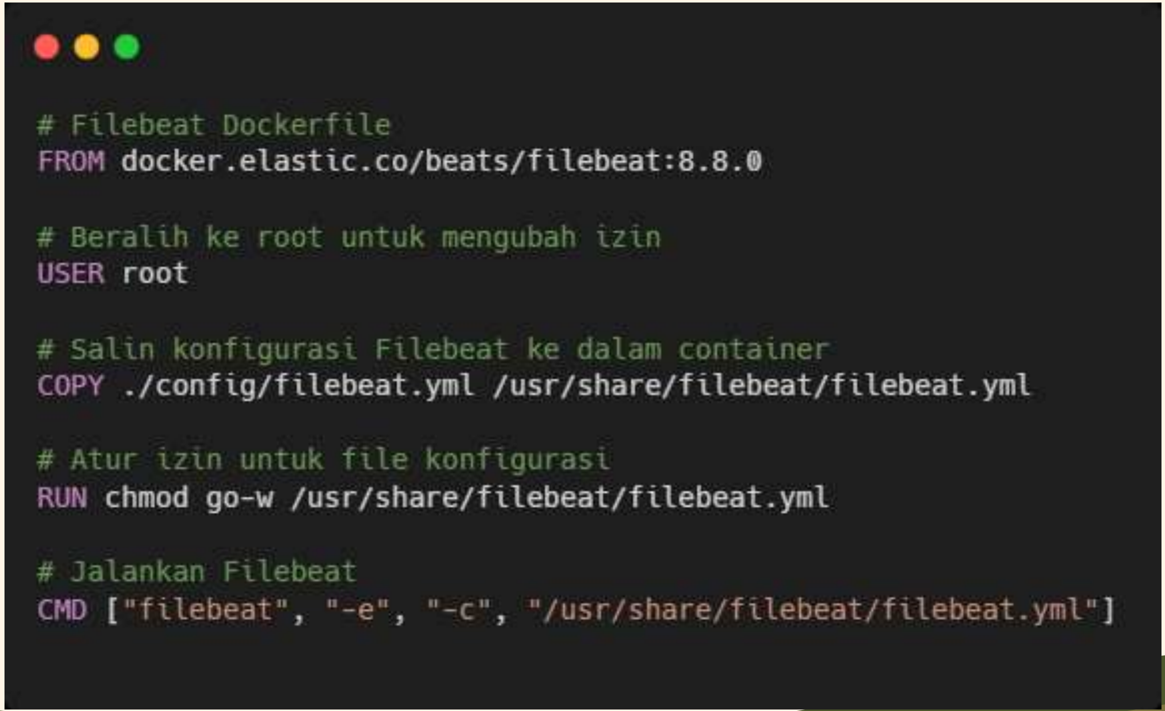
```dockerfile
# Use the official Rust image as the base image
FROM rust:1.74 AS builder

# Set the working directory inside the container
WORKDIR /app

# Copy the Cargo.toml and Cargo.lock files
COPY Cargo.toml Cargo.lock ./

# Copy the source code
COPY src ./src

# Build the application in release mode
RUN cargo build --release

# Use a smaller, final image for deployment
FROM ubuntu:22.04

# Set environment variable for Rocket to use the correct IP address
ENV ROCKET_ADDRESS=0.0.0.0
# ENV ROCKET_PORT=8000

# Copy the built binary from the builder stage
COPY --from=builder /app/target/release/rocket-rust-elk /usr/local/bin/

# Expose the port Rocket will run on
EXPOSE 8000

# Set the default command to run the application
CMD ["rocket-rust-elk"]
```

Prepare the dockerfile.elasticsearch file, which will later be called in the docker-compose file.



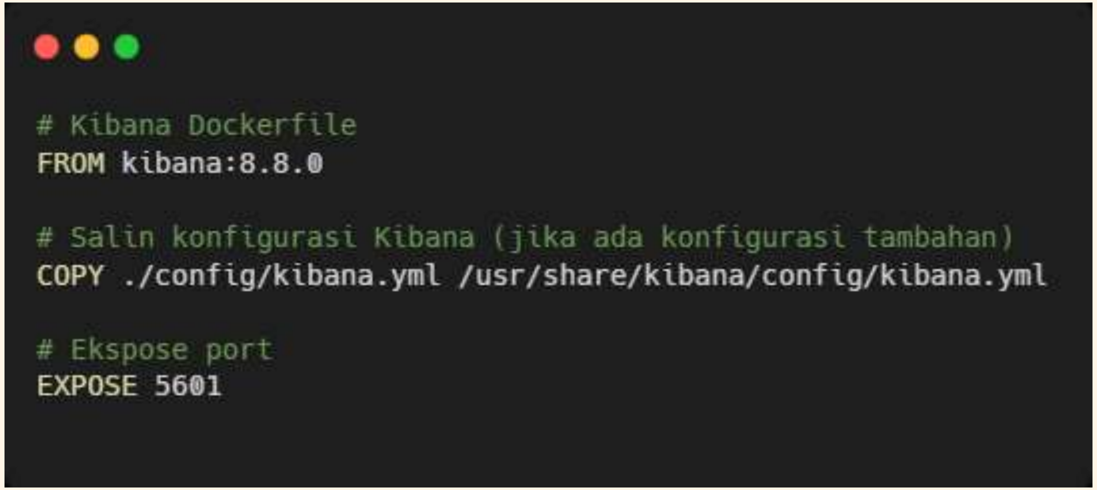
```
# Elasticsearch Dockerfile
FROM elasticsearch:8.8.0

# Beralih ke pengguna root untuk menjalankan perintah izin
USER root

# Salin dan atur izin konfigurasi Elasticsearch
COPY ./config/elasticsearch.yml /usr/share/elasticsearch/config/elasticsearch.yml
RUN chown elasticsearch:elasticsearch /usr/share/elasticsearch/config/elasticsearch.yml && \
    chmod go-w /usr/share/elasticsearch/config/elasticsearch.yml

# Kembalikan ke pengguna elasticsearch untuk keamanan
USER elasticsearch

# Ekspose port
EXPOSE 9200 9300
```

Prepare the dockerfile.filebeat file, which will later be called in the docker-compose file.



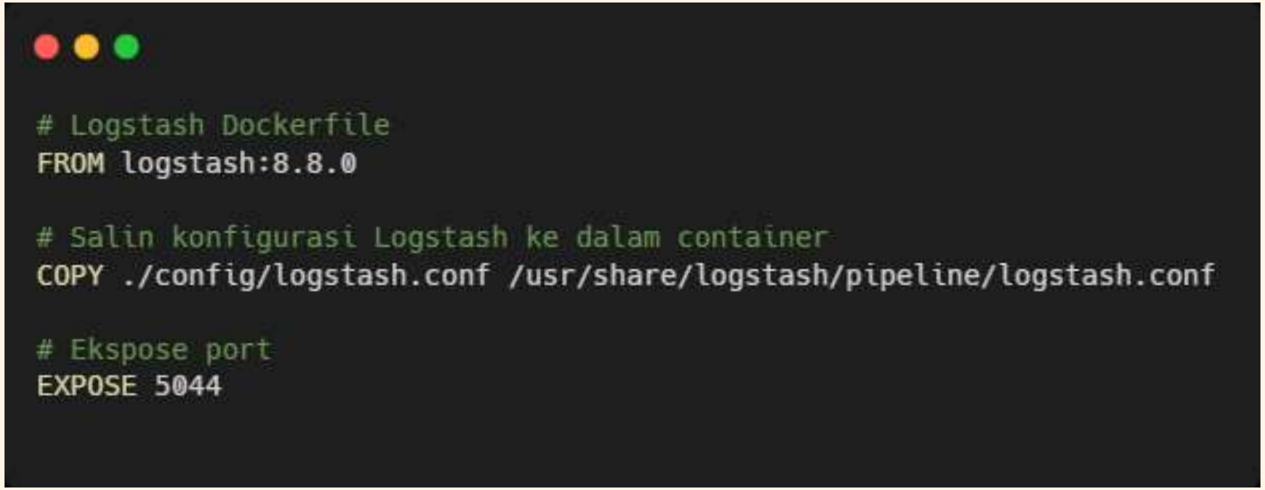
```
# Filebeat Dockerfile
FROM docker.elastic.co/beats/filebeat:8.8.0

# Beralih ke root untuk mengubah izin
USER root

# Salin konfigurasi Filebeat ke dalam container
COPY ./config/filebeat.yml /usr/share/filebeat/filebeat.yml

# Atur izin untuk file konfigurasi
RUN chmod go-w /usr/share/filebeat/filebeat.yml

# Jalankan Filebeat
CMD ["filebeat", "-e", "-c", "/usr/share/filebeat/filebeat.yml"]
```

Prepare the dockerfile.kibana file, which will later be called in the docker-compose file.



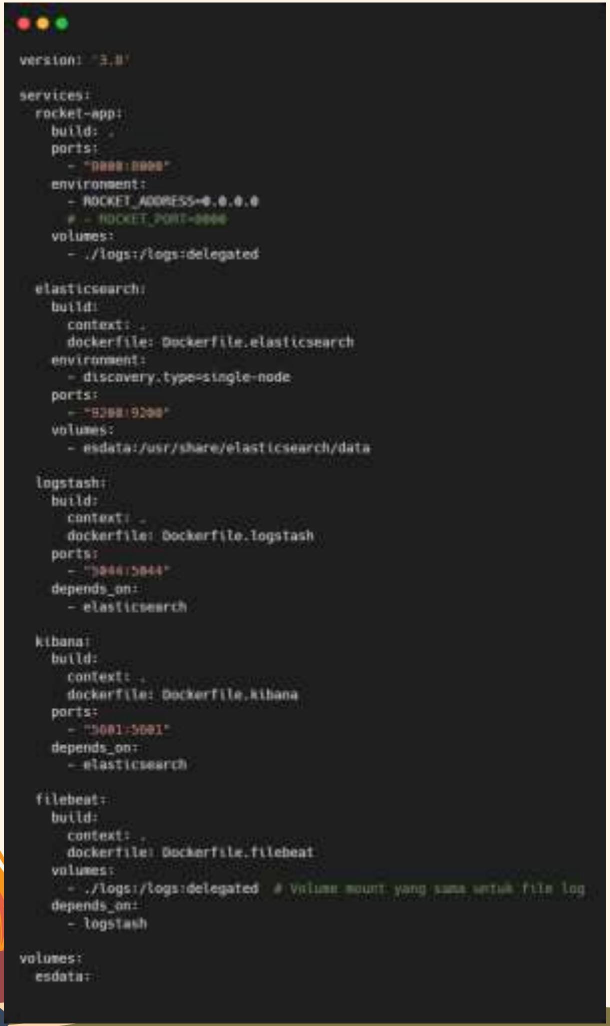
```
# Kibana Dockerfile
FROM kibana:8.8.0

# Salin konfigurasi Kibana (jika ada konfigurasi tambahan)
COPY ./config/kibana.yml /usr/share/kibana/config/kibana.yml

# Ekspose port
EXPOSE 5601
```

Prepare the dockerfile.logstach file, which will later be called in the docker-compose file.



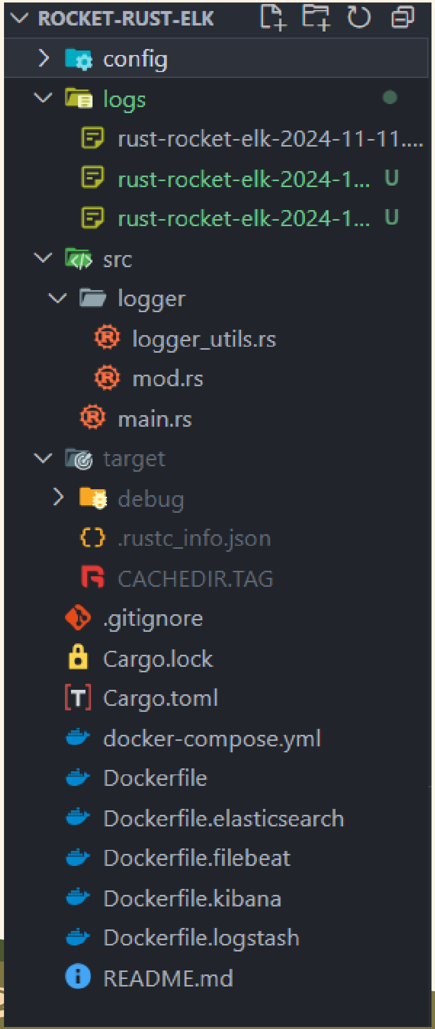
```
# Logstash Dockerfile
FROM logstash:8.8.0

# Salin konfigurasi Logstash ke dalam container
COPY ./config/logstash.conf /usr/share/logstash/pipeline/logstash.conf

# Ekspose port
EXPOSE 5044
```

Set up a Docker Compose configuration.



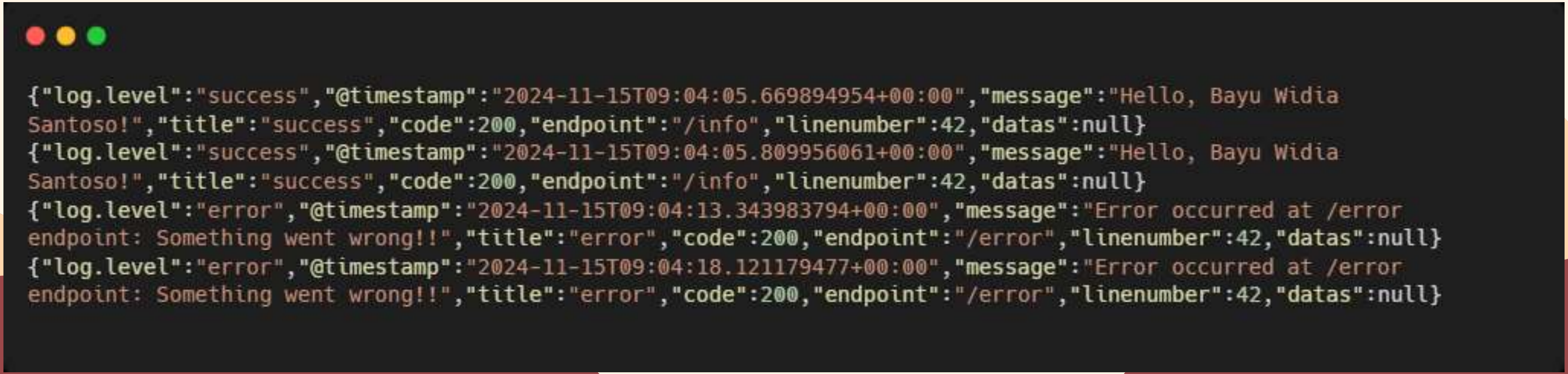
```yaml
version: '3.8'

services:
  rocket-app:
    build: .
    ports:
      - "8000:8000"
    environment:
      - ROCKET_ADDRESS=0.0.0.0
      # - ROCKET_PORT=8000
    volumes:
      - ./logs:/logs:delegated

  elasticsearch:
    build:
      context: .
      dockerfile: Dockerfile.elasticsearch
    environment:
      - discovery.type=single-node
    ports:
      - "9200:9200"
    volumes:
      - esdata:/usr/share/elasticsearch/data

  logstash:
    build:
      context: .
      dockerfile: Dockerfile.logstash
    ports:
      - "5044:5044"
    depends_on:
      - elasticsearch

  kibana:
    build:
      context: .
      dockerfile: Dockerfile.kibana
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch

  filebeat:
    build:
      context: .
      dockerfile: Dockerfile.filebeat
    volumes:
      - ./logs:/logs:delegated  # Volume mount yang sama untuk file log
    depends_on:
      - logstash

volumes:
  esdata:
```

The folder structure that has been implemented is as follows.



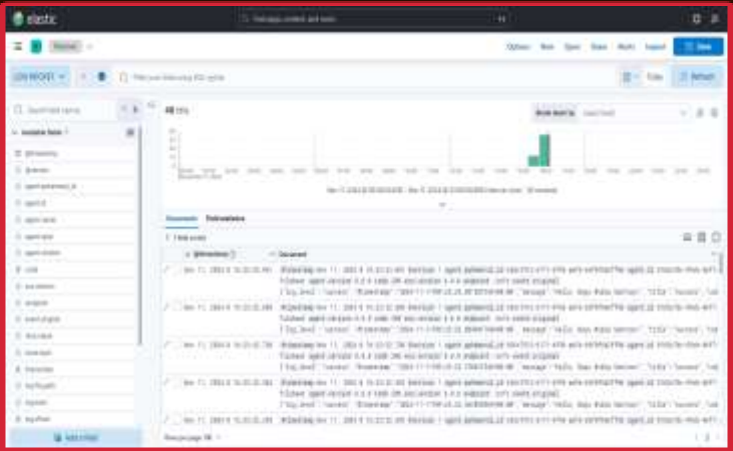
```
ROCKET-RUST-ELK
  > config
  v logs
      rust-rocket-elk-2024-11-11....
      rust-rocket-elk-2024-1...  U
      rust-rocket-elk-2024-1...  U
  v src
    v logger
        logger_utils.rs
        mod.rs
      main.rs
  v target
    > debug
      .rustc_info.json
      CACHEDIR.TAG
    .gitignore
    Cargo.lock
    Cargo.toml
    docker-compose.yml
    Dockerfile
    Dockerfile.elasticsearch
    Dockerfile.filebeat
    Dockerfile.kibana
    Dockerfile.logstash
    README.md
```

Once completed, it will generate a log like this, which can be processed using the ELK stack.



```
{"log.level":"success","@timestamp":"2024-11-15T09:04:05.669894954+00:00","message":"Hello, Bayu Widia
Santoso!","title":"success","code":200,"endpoint":"/info","linenumber":42,"datas":null}
{"log.level":"success","@timestamp":"2024-11-15T09:04:05.809956061+00:00","message":"Hello, Bayu Widia
Santoso!","title":"success","code":200,"endpoint":"/info","linenumber":42,"datas":null}
{"log.level":"error","@timestamp":"2024-11-15T09:04:13.343983794+00:00","message":"Error occurred at /error
endpoint: Something went wrong!!","title":"error","code":200,"endpoint":"/error","linenumber":42,"datas":null}
{"log.level":"error","@timestamp":"2024-11-15T09:04:18.121179477+00:00","message":"Error occurred at /error
endpoint: Something went wrong!!","title":"error","code":200,"endpoint":"/error","linenumber":42,"datas":null}
```

Here are the results that have been read by Kibana logs.



SEE YOU …