**Distributed Software Architecture**

**Lab 5 Report**

# GOALS

Implement Traefik gateway as a single, secure entry point to force using of HTTPS for communication from the outside.

Authentication and authorization are handled by Keycloak as an OAuth 2.0/OIDC provider, requiring valid JWT bearer tokens.

A zero-trust internal network is created by updating to mutual TLS, forcing services to verify each other's identity.

# SOURCE CODE

*https://github.com/tvph1996/fifth\_lab*

# SYSTEM SETUP

Two new blocks added to the existing stack: Traefik & Keycloak

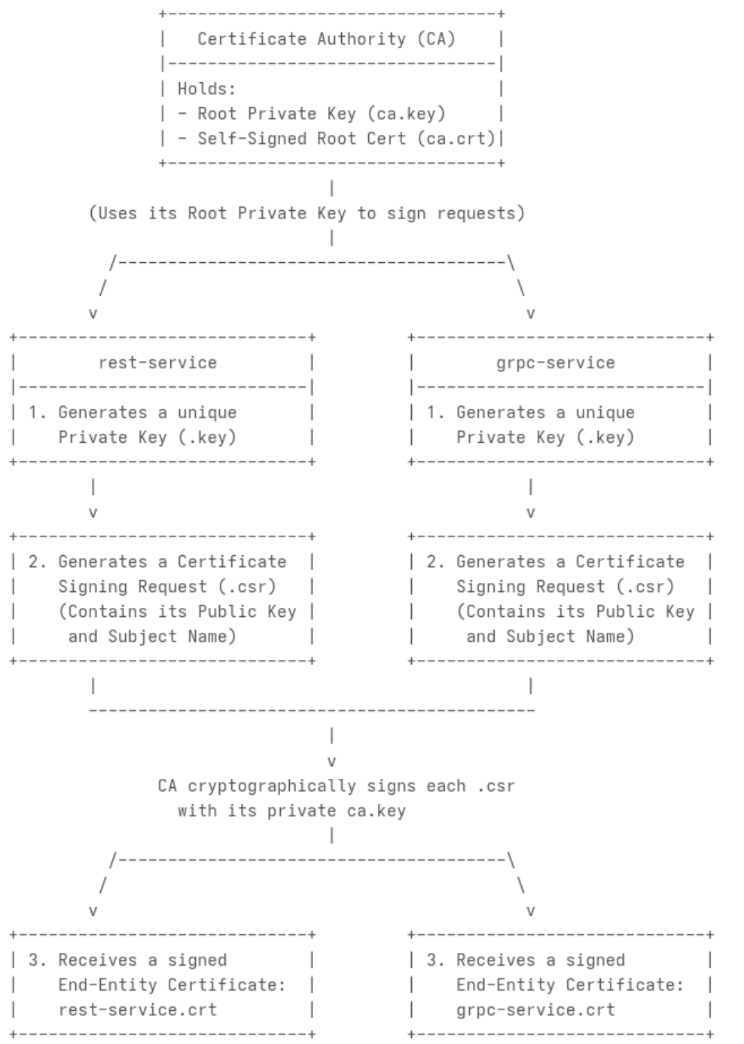
Traefik acts as the gateway, managing all incoming HTTPS traffic and routing it to the internal services while hiding the rest of the system.

Keycloak handles user login and gives temporary access tokens (JWTs) for authentication and authorization.

Used in the lab is an own self-signed CA, which acts as the root of trust for other services, including a root private key (ca.key) for signing other certificates and a public certificate (ca.crt) for verification.

Similarly, both *rest-service* and *grpc-service* also has their respective private key and public certificate.

The process is as below:



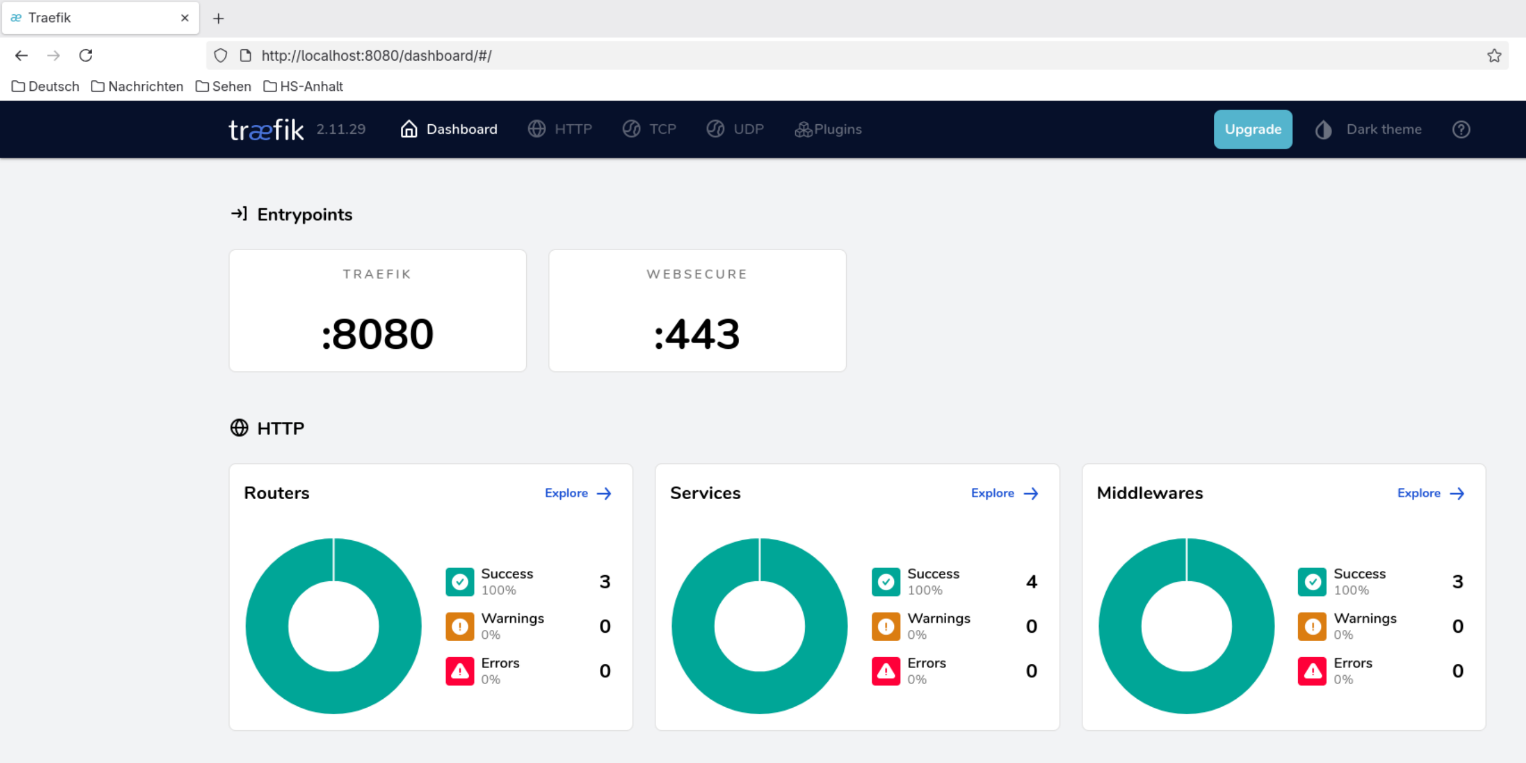
In additional, another pair of private key and public certificate is created for *localhost*, differs from using rest-service certificate as the lab guideline. This is to avoid making rest-service identity visible to the outside, in case the attacker steal *localhost*, he can’t call grpc-service -> MongoDB directly. Not only that, a separate hostname of a certificate is still better.

mTLS is setup between rest-service and grpc-service. Both must prove their identity to each other before any communication is allowed, achieving a zero-trust security posture.

# OUTPUT

### A screenshot of a computer AI-generated content may be incorrect.Browser with green padlock on https://localhost/api/healthz

### Traefik



A screenshot of a computer

AI-generated content may be incorrect.

### Keycloak login page

A screenshot of a computer

AI-generated content may be incorrect.

### Keycloak realm, user and role

A screenshot of a computer

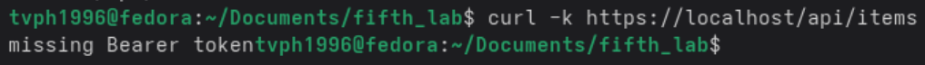
AI-generated content may be incorrect.

### Successfully use token to get an Item

A computer screen shot of a program code

AI-generated content may be incorrect.

### Fail when access without token



### Fail when token expired

A screen shot of a computer code

AI-generated content may be incorrect.

### Fail when using fake private key

A close up of a white background

AI-generated content may be incorrect.



### Unreadable output from packet snip

A black screen with white dots

AI-generated content may be incorrect.

# Protection and Zero-Trust in each layer

1. Traefik Gateway

* The first layer of defence. Using HTTPS so communication from the user's browser is kept secret. It also hides the system’s internal services, reducing the number of targets an attacker can see.
* Zero Trust: Traefik doesn't make any service public. A service is only visible from the outside if explicitly allow it with a label.

1. Keycloak with JWTs

* Once a request gets past the gateway, this layer checks the user's ID.
* It separates the job of authentication & verification from the application. If the rest-service has a bug, user passwords is still safe because they are managed by a separate system.
* Using tokens that expire quickly also adds security. If a token is stolen, it's only useful for a few minutes (default is 5 minutes).
* Zero Trust: always verify users. rest-service only accept calls with a valid token.

### Mutual TLS

* Add another protection layer with separate pair of key and certificate for each service. If an attacker get past the gateway or 1 out of the services, he can still not roam freely in the system.
* Zero Trust: Services are not allowed to talk to each other freely. Before they can communicate, both services must prove their identity to each other by showing a certificate and pass a check to verify they have a valid private key.

End of Report