

Software Engineering for the cloud Project

SimpleAuthJWT



EL BAZ Jonathan

CHOUKROUNE Simon

Summary

1. Architecture

- 1.1) Introduction.....
- 1.2) Design overview.....

2. Demonstration

- 2.1) Service mesh deployment.....
- 2.2) Client side.....

3. Additional contents

1. Architecture

This particular project serves as a comprehensive demonstration that showcases the process of developing a fundamental authentication application. The application employs the use of JSON Web Tokens (JWT) and one-time passwords (OTP) to implement heightened security measures. These security features are integrated seamlessly within a microservices architecture, with an additional layer provided by a service mesh.

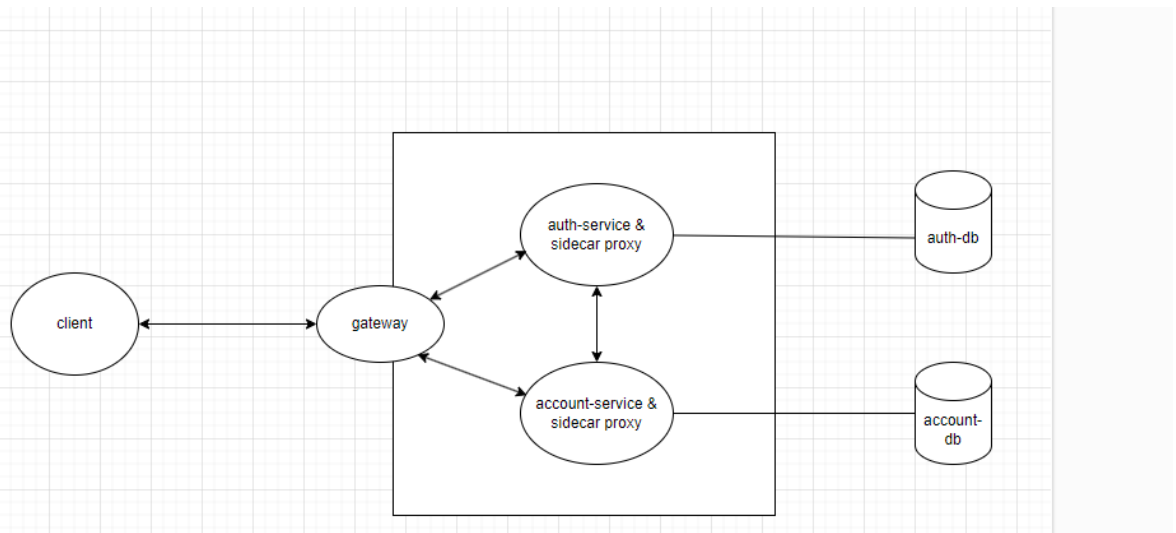
1.1) Introduction

The central focus of this initiative is to elucidate the intricacies involved in crafting a robust authentication system that not only utilizes modern security standards such as JWT for secure token-based authentication but also integrates the added layer of one-time passwords for an extra layer of security. The microservices architecture employed in the development of this application underscores the modular and scalable nature of the system, allowing for efficient management and maintenance.

Furthermore, the inclusion of a service mesh in the architecture contributes to the overall reliability and resilience of the authentication system. The service mesh acts as a dedicated infrastructure layer that facilitates communication and connectivity between microservices, ensuring seamless and secure data exchange. This holistic approach to security and architecture design aims to provide a comprehensive understanding of the implementation of cutting-edge technologies for authentication within a microservices environment

1.2) Design overview

The proposed architectural framework that we aim to implement is a carefully designed structure comprising two distinct services: 'auth' and 'account.' The 'account' service is specifically crafted to oversee the management of users' personal data, while the 'auth' service is dedicated to handling user authentication processes. The architecture adheres to the 'One database per service' pattern, ensuring that each service possesses its own dedicated database. This pattern is strategically employed to maintain the confidentiality of persistent data associated with each microservice, restricting access solely to its respective API.



In order to enhance control and orchestration between these services, we leverage a service mesh, a pivotal element that facilitates seamless communication, observability, and control within the microservices architecture. To achieve this, a 'sidecar proxy' is incorporated into each service, serving as an intermediary to govern the flow of data within the confines of each service. This meticulous integration of a service mesh not only fosters efficient communication but also provides a heightened level of visibility into the interactions between microservices, bolstering overall control and maintainability of the system.

In summary, the envisioned architecture goes beyond a mere deployment plan; it represents a thoughtfully structured system where 'auth' and 'account' services operate autonomously with dedicated databases, and a service mesh, featuring 'sidecar proxies,' acts as a comprehensive mechanism for regulating communication, observability, and control within the microservices environment.

2. Architecture

2.1) Service mesh Deployment

When cloning the repository, navigate to the 'k8s' folder. Inside, you should find a Helm chart with subdependencies: 'auth,' 'account,' and 'gateway.' In the root 'k8s' folder, deploy the infrastructure by entering the following command: **helm install 'name-of-deployment' .**

```

nasty@nasty:~/Documents/Projects/k8s$ helm install deploy .
NAME: deploy
LAST DEPLOYED: Mon Jan 22 21:11:12 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=simple-auth-deploy,app.kubernetes.io/instance=deploy" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
nasty@nasty:~/Documents/Projects/k8s$ kubectl get pods

```

After executing the deployment command, wait for a while to observe the running pods using the following command: **kubectl get pods.**

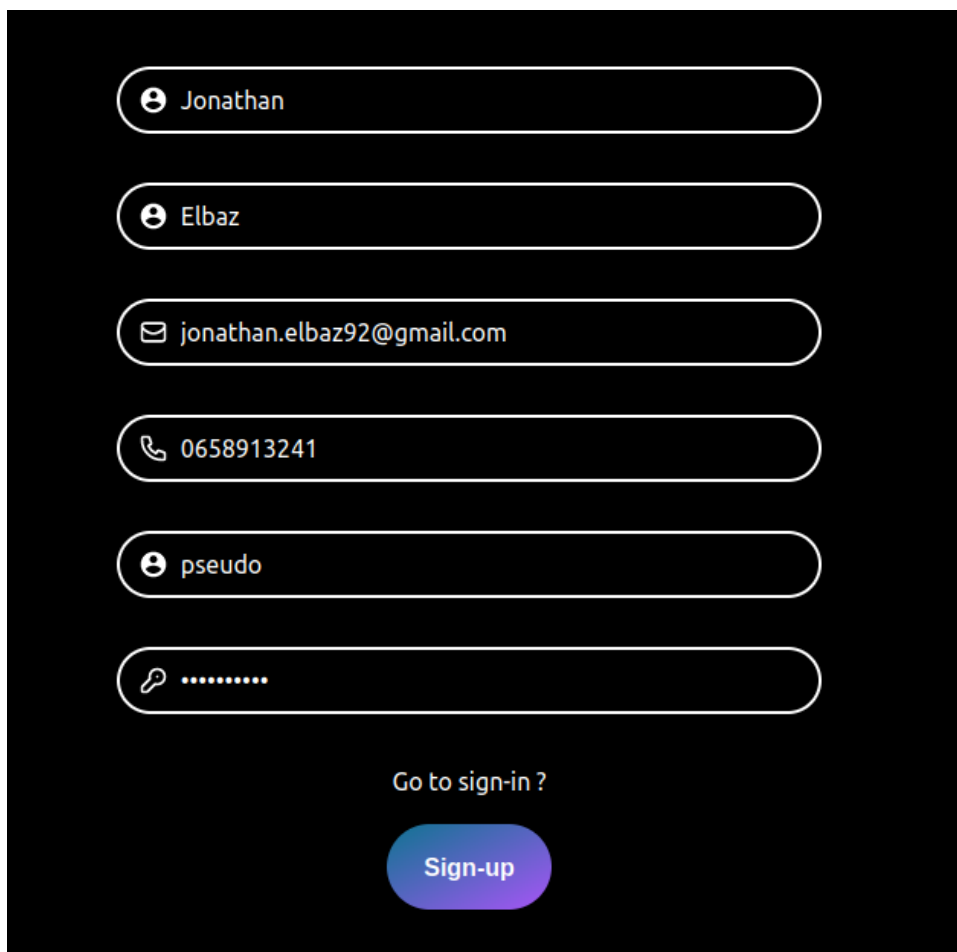
```
nasty@nasty ~/Documents/SimpleAuthJWT/k8s$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
account-db-deployment-7b45bc54bc-p65mq 2/2     Running   0           26s
account-deployment-6664847c6b-ghv1k    2/2     Running   0           26s
auth-db-deployment-5988ccf996-jmpmn    2/2     Running   0           26s
auth-deployment-98f6ddbfc-58rx9        2/2     Running   0           26s
dnsutils                               1/1     Running   0           19 <4h20m ago> 44d
nasty@nasty ~/Documents/SimpleAuthJWT/k8s$
```

At this point, the infrastructure is deployed, and the pods containing services are running. To forward the port of the ingress and access it locally, you will find a bash script named 'ingress-forward.sh' in the repository. Execute this script:

```
nasty@nasty ~/Documents/SimpleAuthJWT/k8s$ sudo -E ./ingress-forward.sh
[sudo] password for nasty:
Open Ingress at http://localhost:80/
Forwarding from 127.0.0.1:80 -> 8080
Forwarding from [::]:80 -> 8080
```

3.2) Client side

With the infrastructure deployed, running, and bound to the localhost HTTP port, proceed to test it. Navigate to the 'client' folder, enter the command 'yarn,' followed by 'yarn build,' and finally 'yarn dev' to run the website. You should land on the sign-up page; simply register your account and observe the outcome.



Jonathan

Elbaz

jonathan.elbaz92@gmail.com

0658913241

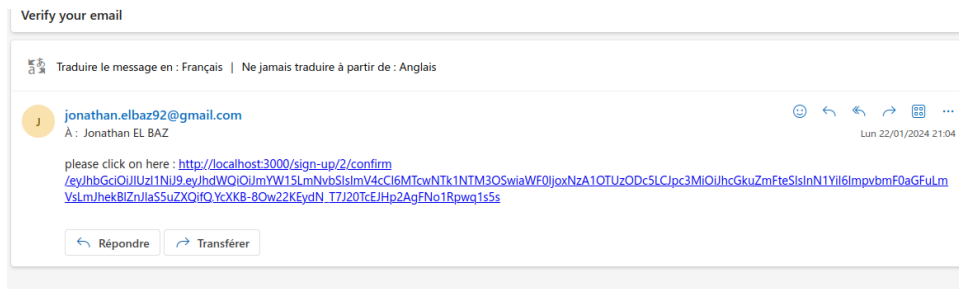
pseudo

.....

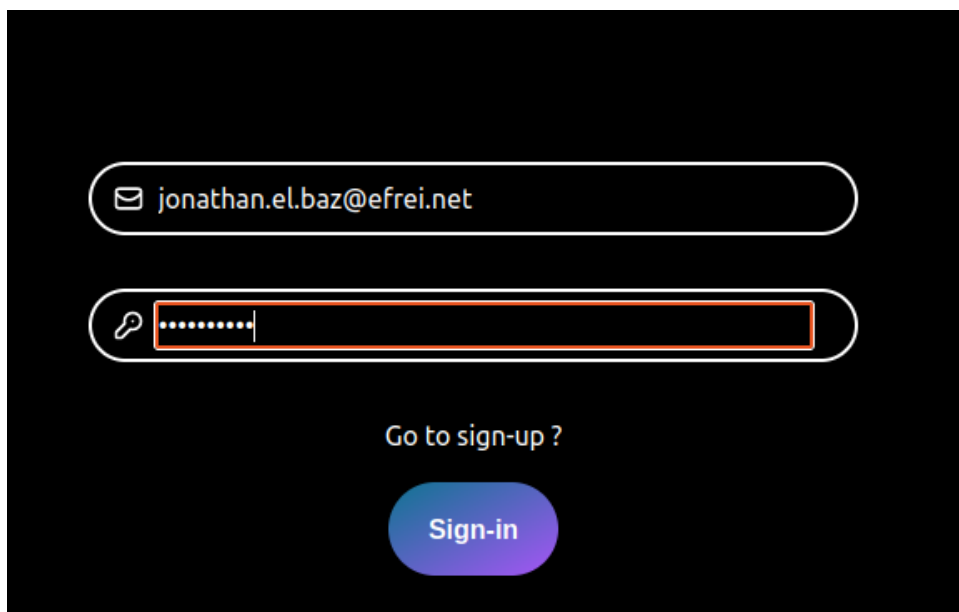
Go to sign-in ?

Sign-up

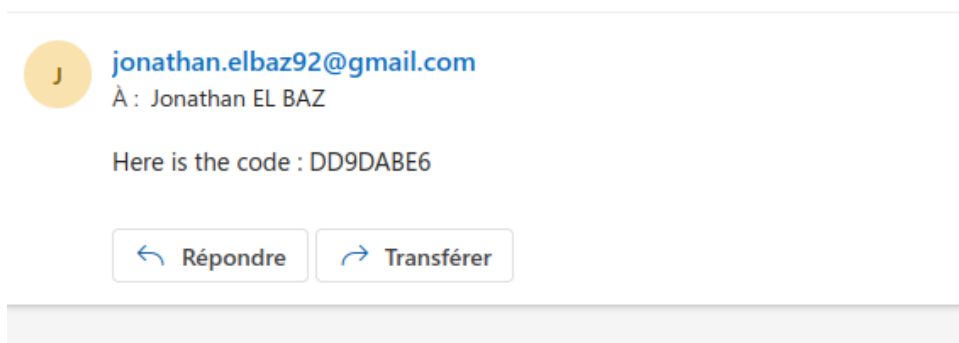
Subsequently, similar to other websites, an email will be sent containing a link. Visit this link to complete the registration and verification process.



At this stage, you can sign in by navigating to the '/sign-in' page and following the provided instructions



You will receive a One Time password, just type it on the screen and you will be redirected to /connected url to illustrate the success of the app.



3. Additional content

Activité	Type	Date de début	Date de fin	Score	Réussie
Infrastructure as Code avec Terraform	Atelier	22 nov. 2023	22 nov. 2023	0.0/100.0	
Infrastructure as Code avec Terraform	Atelier	22 nov. 2023	22 nov. 2023	0.0/100.0	
Infrastructure as Code avec Terraform	Atelier	22 nov. 2023	22 nov. 2023	0.0/100.0	
Infrastructure as Code avec Terraform	Atelier	22 nov. 2023	22 nov. 2023	0.0/100.0	
Introduction to Docker	Atelier	19 nov. 2023	19 nov. 2023	100.0/100.0	✓
Configuring Networks via gcloud	Atelier	19 nov. 2023	19 nov. 2023	100.0/100.0	✓

Links:

<https://microservices.io/patterns/data/database-per-service.html>