

Table des matières

1	Introduction	2
2	Project	2
3	Architecture	2
4	Mush Service	3
5	Run App	3

1 Introduction

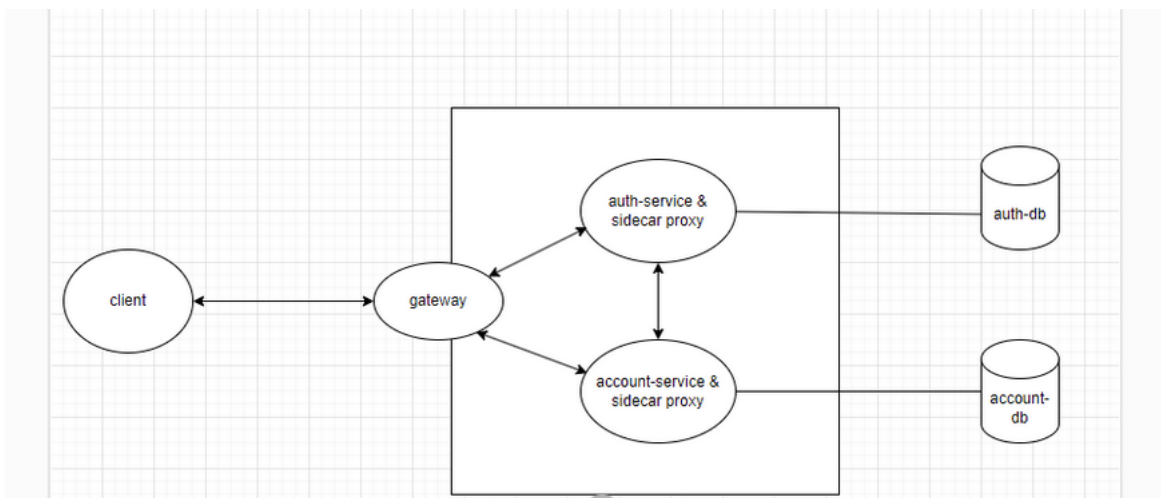
In the rapidly evolving landscape of technology, the fusion of various cutting-edge tools and methodologies has become crucial for innovative web service development. This project, titled 'Synergizing Modern Technologies : A Journey through Web Services, Containerization, and Cloud Integration,' is designed to explore the integration of diverse technologies such as Java, Node.js, Docker, and Kubernetes, along with various cloud deployment strategies. Our journey begins with the development of a mini-application, incorporating both front-end and back-end elements, and progresses through stages of containerization using Docker, orchestration via Kubernetes, and options for deployment in cloud environments. Emphasizing practical application, the project will demonstrate the synergy between these technologies, providing a comprehensive understanding of how they coalesce to create efficient, scalable, and robust web services. This exploration is not just about learning individual technologies but understanding how they interact in real-world scenarios, preparing us for the challenges of modern web development.

2 Project

This project serves as an illustrative example of creating a basic authentication application using JWT tokens and one-time passwords (OTP) for enhanced security measures using microservices architecture with the service mesh.

3 Architecture

Here is the designed architecture that we want to deploy. It composed of two services : auth and account. The account service is designed to manage the personal data of users. The auth service is designed to manage the authentication of users.



Each services has his own database following the “One database per service” pattern to Keep each microservice’s persistent data private to that service and accessible only via its API.

4 Mush Service

To add more control between services, we use the service mesh to facilitate communication, observability, and control between services in a microservices architecture. We inject a proxy in all services, a 'sidecar proxy' to control the flow of data in each services.

5 Run App

When clone the repository, on the k8s folder, you should see helm chart with subdependencies : auth, account and gateway.

On the root k8s folder, to deploy the infrastructure, type helm install 'name-of-deployment' :

```
nasty@nasty ~$ 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 ~$ kubectl get pods
```

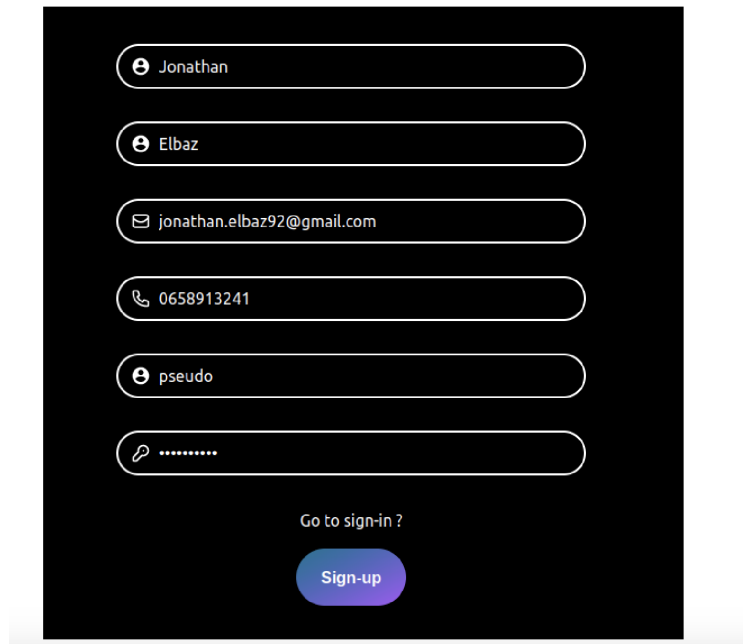
After that, wait some times to see the pods running with this command : kubectl get pods.

```
nasty@nasty ~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
account-db-deployment-7b45bc54bc-p65mq 2/2     Running   0           26s
account-deployment-6664847c6b-ghvulk 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   19 (4h20m ago) 44d
```

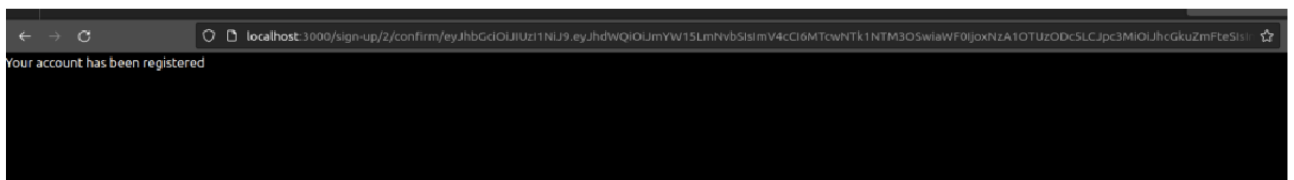
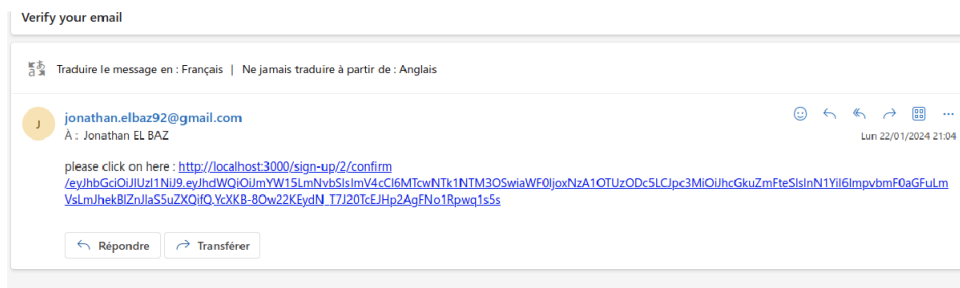
For now, the infrastructure is deployed and the pods containing services is running. To forwarding port of ingress and so, use it at localhost, you should see a bash script named ingress-forward.sh. This script contains the requirements to forward the port of the ingress. Just execute this script :

```
nasty@nasty ~$ 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
```

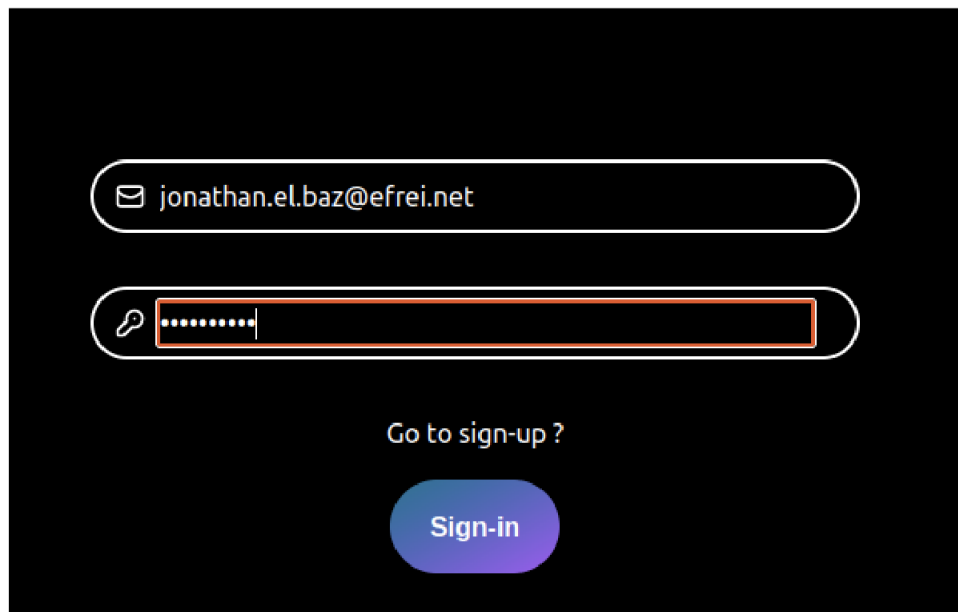
For now, the infrastructure is deployed, running and binded to the localhost http port. To test it, I recommend you to run and build the client website. Just go to the client folder, type yarn, yarn build and finally yarn dev to run the website. You should arrive at sign-up page, just register your account and see what happens.



After that, like another website, an email is sent with a link, go to this link to be registered and verified :



For now, you can sign-in, go to the /sign-in page and follow the 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.

