Cloud Services And Infrastructure

by,

Group C

Introduction

- We have designed a retail website where people can login, browse the product, add them to cart, checkout and pay for it.

- The website now sells kids and womens clothes and accessories.

Frontend – Container Expose Port 80

- Angular Project
- Ngnix - Serves static files from /dist folder
- JWT based authentication

Backend – Container Expose port 5000

- Flask app
- psql database
- Alembic migration (sqlalchemy)

Traefik – Load Balancer/ Reverse Proxy

- Traffic from / - forward to frontend service container (port 80)
- Traffic from /api – backend service container (port 5000)

Architecture

- Client types – https://booboofashions.com in browser
- Traefik forwards it to frontend service at (port 80)
- Frontend container (which has nginx) receives the request and nginx serves the static files that was stored in its /dist folder *
- In this case nginx serves the frontend page of the application which has “signup”, “signin” options.

(* dist folder has angular project build – which docker builds and github workflows -later explained in slides)

Architecture Continues….

- User clicks signup button in the frontend page
- Traefik forwards it to frontend service at (port 80)
- Frontend container (which has nginx) receives the request and serves the static files that was stored in its /dist folder
- User enters signup information in the application page and angular calls booboofashions.com/api/signup post api .
- Traefik recognises that /api and forwards it to backend service
- Flask api endpoint (/api/signup) uses the parameters in the request to do signup function and possibly updates the data in database and send the response back to frontend. (an invalid information received or dashboard for users if the signup post was successful)

Local Deployment

- *Download the repository:*

  git clone - git@github.com:dhivyarrk/cloudservice.git

- *Deploy*:
  - Run locally using Docker compose
    - Install docker compose
    - cd cloud service
    - docker compose build –no-cache
    - docker compose up
  - tear the application down
    - docker compose down

Deployment – Digital ocean

- SSH to the manager droplet:

  ssh -i ~/.ssh/id_ed25519 dockeruser@159.89.10.174

  Traffic deployment - https://github.com/dhivyarrk/cloudservice/blob/main/traefik-ssl.yml

  - docker stack deploy -c traefik-stack.yml traefik-stack

  Portainer deployment - *https://github.com/dhivyarrk/cloudservice/blob/main/portainer.yml*

  - docker stack deploy -c portainer-stack.yml portainer-stack

Deployment Continues...

Application-stack deployment

- Pulls the frontend and backend images from github container registry and deploys
- Docker swarm is used – uses stack name as build in prefix so all service will get appended and this has to be noted.

  Eg my-stack_*
- Docker swam cannot use portainer registry credentials . So if deploying without portainer add docker credentials directly to nodes, create a secret so docker swarm can pull images .
  - echo "github PAT" | docker login ghcr.io -u dhivyarrk –password-stdin
  - docker secret create regcred ~/.docker/config.json
  - docker stack deploy -c my-stack.yml my-stack ***--with-registry-auth***

SSL Certificate

- Namecheap → bought a domain named booboofashions.com
- Added digital ocean name servers to that domain name names servers in namecheap so that digital ocean can manage dns records
- In digital ocean add following domains to point to manager droplets:

  Networking → domain → add A record
  - traefik.booboofashions.com
  - portainet.booboofashions.com
  - booboofashions.com
  - backend.booboofashions.com
  - frontend.booboofashions.com

Continues...

- let traefik generate ssl certificate from lets encrypt acme.
- test with staging acme as rate limits are there for generating certificates .
- get digital ocean api token and feed as secret to traefik so that it can write the txt record when acme presents the dns challenge.

Continues...

- Note for dns challenge traefik cannot read the token from DO_AUTH_TOKEN so give the environment variable as DO_TOKEN_FILE and feed the file where the token is.
- (tried with nip.io domains but quickly hit rate limits for the week so switched to buying real domain)
- how to generate digital ocean api token:
  - api -> generate new token -> scopes (read and write)

CI/CD Workflows

- Frontend and backend images are build using github workflows and pushed to github container registry. they are stored as private images.

  https://github.com/dhivyarrk/cloudservice/tree/main/.github/workflows
- Frontend builds the image from frontend/Dockerfile
- Backend builds the image from backend/Dockerfile
- Whenever a commit is pushed to github with commit message [build: frontend] frontend image is build and pushed to github container registry .
- Whenever a commit is pushed with commit message [build: backend] backend image is build and pushed to github container registry .

  https://github.com/dhivyarrk/cloudservice/actions

Major Challenges

- **For generating acme certificate:**

  Note for dns challenge traefik cannot read the token from DO_AUTH_TOKEN so give the environment variable as DO_AUTH_TOKEN_FILE and feed the file where the token .
  (This was not documented and harder to find so I had to read through the code and errors to figure out a way)

- DNS propagation took longer (hours)

- Docker swarm always uses stack name to prefix all the services. There is no way to give empty prefix. So the overall code need to adapt in anticipation.

- Domain name: nip.io or free domains had rate limits for generating lets encrypt certificate-which often was hit and needed to wait. Eventually moved to buying a custom domain from namecheap – booboofashions.com

Challenges Continues...

- Figuring out the structure of application. For example if traefik should forward to backend and frontend or let nginx forward using proxy_pass to backend – Eventually decided to let traefik handle the routing as it can be usefull in scaling.

- Docker swarm did not use the GHCR credentials from services (provided as secret in yaml file). It needed it to be available in the host in such a way that docker swarm can use it. Simply docker login and then pull in the node worked but when using docker swarm the deployment did not pull the image. Docker swarm did not have the credentials to pull the image. So needed to docker login in the nodes (scripted to support multiple nodes login at the same time) and then create secret with it and deploy the application stack with –with-registry-auth option.

Application Demo of Boo Boo Fashions