Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Елаборат

Континуирана интеграција и испорака на веб апликација

Студент: Кире Бошковски

Индекс: 225040

Факултет: ФИНКИ – Скопје

Ментор: проф. Панче Рибарски

Датум: септември 2025

1. Вовед

Целта на овој проект е демонстрација на концептите на континуирана интеграција и континуирана испорака (CI/CD) со повеќе сервисна архитектура. Проектот е изработен исклучиво за потребите на курсот и служи како практична вежба за развој и автоматизација на деплојмент процеси.

2. Архитектура на системот

Системот е **full-stack web application** составен од повеќе слоеви:

- **Angular Frontend** – SPA (Single Page Application) за интеракција со корисникот.
- **ExpressJS Backend – Authentication Service**
    - Регистрација на корисници.
    - Генерирање на JWT токени при најава на корисници.
    - Освежување на токени (refresh tokens).
- **ExpressJS Backend – Bussiness logic service**
    - Бизнис логика за управување со ентитети „expenses“.
- **MongoDB база на податоци** со два ентитети:
    - **users**
    - **expenses**

3. CI/CD Pipeline

За континуирана испорака користам Github Actions кадешто на секој push на код на master гранка, се билдаат апликациите за фронтенд, бекенд и секјурити и се постирани на DockerHub, додека базата ја користи официјалната слика за MongoDB. Во прилог се 3 прилагодени Докер слики за овој проект:

```
backend > Dockerfile > ...
1   FROM node:20-alpine (last pushed 1 week ago)
2   RUN apk update && apk upgrade
3   WORKDIR /app
4   COPY package*.json ./
5   COPY tsconfig.json ./
6   RUN npm install
7   COPY src/ ./src/
8   RUN npm run build && npm prune --production
9   EXPOSE 5000
10  CMD ["node", "dist/server.js"]
```

1. Иста за backend u authentication

```
frontend > Dockerfile > ...
1   FROM node:20-alpine AS build (last pushed 1 week ago)
2   WORKDIR /app
3   COPY package*.json ./
4   RUN npm install
5   COPY . .
6   RUN npm run build --prod
7
8   FROM nginx:stable-alpine (last pushed 3 weeks ago)
9   COPY --from=build /app/dist/frontend/browser /usr/share/nginx/html
10  COPY nginx.conf /etc/nginx/conf.d/default.conf
11  EXPOSE 80
12  CMD ["nginx", "-g", "daemon off;"]
```

2. frontend

Workflow-то се извршува автоматски:

- при **push** на master гранка,
- при **pull request** кон master,
- или рачно преку **workflow_dispatch** (копче „Run workflow“ во GitHub UI).

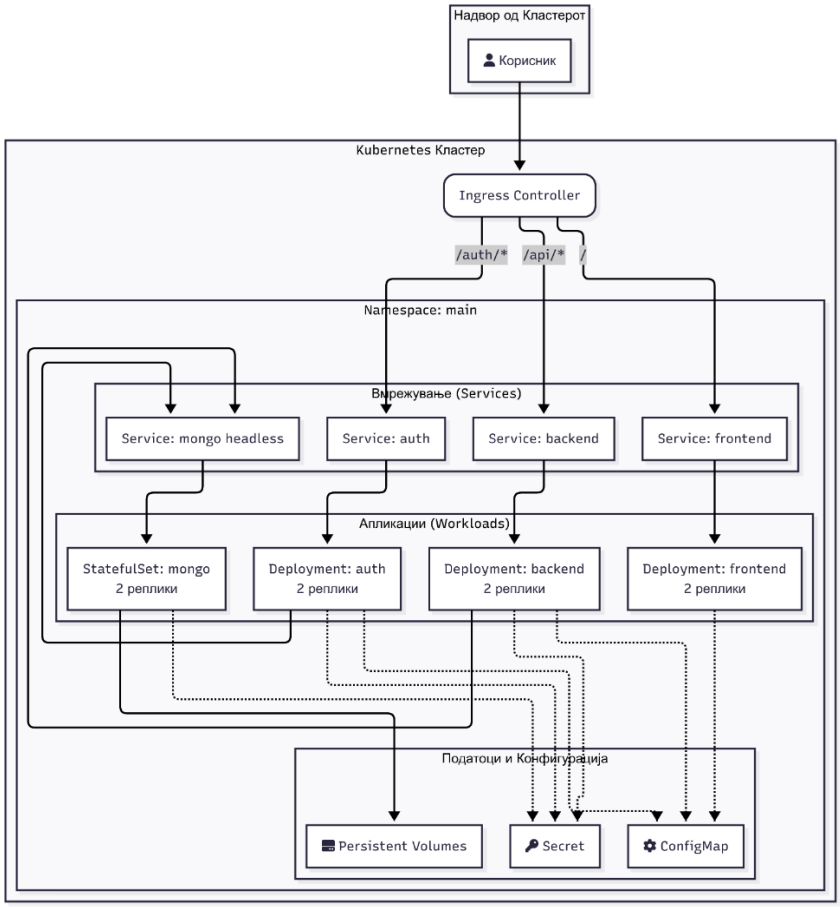
Дефиниран е job со име „Build and Push Docker Images“ кој се извршува на GitHub-овата виртуелна машина со Linux (ubuntu-latest). Користам стратегија matrix која овозможува паралелно извршување на истите чекори за повеќе вредности. Тука има три вредности – **authentication**, **backend**, **frontend** – односно трите сервиси. GitHub ќе покрене посебен паралелен процес за секој сервис. Workflow-то работи на начин што за секоја вредноста во матриксот го зема контекстот на тој директориум од репото ја билда сликата со контекстот и ја праќа сликата на DockerHub. Со овој workflow секој пат кога ќе се направи промена на master гранката, автоматски и паралелно се градат Docker слики за **authentication**, **backend** и **frontend**, а потоа се објавуваат на DockerHub со две ознаки: latest и ознака со хашот на commit-от.

Ова обезбедува секогаш да имаш свежи Docker слики спремни за деплојмент во Kubernetes.

При користење на апликацијата без Кубернетес со docker-compose.yml file би можело да се стартне апликацијата и да работи во продукциска околина со сликите на DockerHub поставени од претходно.

4. Kubernetes

За оркестрација и автоматизација на сите микросервиси се користи **Kubernetes кластер**. Кластерот овозможува скалабилност, автоматско распределување на оптоварувањето, лесно ажурирање и висока достапност (High availability).



Кратко објаснување за манифестите:

- Прво се креира свој **namespace**: main во кој ќе живеат сите подови на апликацијата.
- **Secret** – ресурс за чување чувствителни информации како лозинки, токени и клучеви. Во овој проект сите тајни не се баш тајни поради тоа што тие се

енкодирани (формат кој е потребен од страна на кубернетес за да работи) наместо енкриптирани. за да се енкирптира најчесто се користи посебен совтвер кој треба да се конфигурира, но за овој проект го скокнав овој чекот.

- **ConfigMap** – Kubernetes ресурс за чување на неконфиденцијални конфигурациски податоци. Податоците од делот data се користат во останатите манифести како на пример за деплојмент.
- **StatefulSet** за MongoDB со реплика сет - објект за апликации каде што секоја реплика треба да има стабилен идентификатор (секој од подовите има исто исто име пример: mongo-0, mongo-1 ...) и свој перзистентен диск. При скалирање подовите се креираат и намалуваат редоследно. Погодно за правилно и лесно скалирање на базата со податоци.

  * за правилно да се скалира монго беше потребно истиот да се исконфигурира, да му кажете на првиот под да биде primary, а на вториот да биде secondary во реплика сетот. Тоа се прави со следните команди:

```
kubectl exec -n main mongo-0 -- mongo
> rs.initiate({
_id: "rs0",
members: [
{ _id: 0, host: "mongo-0.mongo.main.svc.cluster.local:27017" },
{ _id: 1, host: "mongo-1.mongo.main.svc.cluster.local:27017" }
]
})
```

- **Services** – објекти кои обезбедуваат стабилна внатрешна адреса и DNS име за пристап до подови. Подовите во Kubernetes се динамични – нивните IP адреси се менуваат при секое ресоздавање. **Service** е слој на стабилност: дава фиксно DNS име (и фиксна IP) за група подови со ист label. Со selector Service знае кон кои подови да го рутира сообраќајот. Стандардно е ClusterIP – достапен е само внатре во кластерот. За потребата за mongo подовите користиме Headless Service. Не добива сопствена IP адреса. Наместо тоа, создава DNS записи за секој под од StatefulSet (на пр. mongo-0.mongo.main.svc.cluster.local и mongo-1.mongo.main.svc.cluster.local). Ова е потребно за да реплика сетот може да знае точно кој е кој член.
- **Deployment** е Kubernetes ресурс кој ја опишува желбата за бројот на реплики (pods), верзијата на сликата која треба да се користи и конфигурациските параметри за апликацијата. Тој овозможува автоматско креирање и одржување на подови, ролинг ажурирања при промена на сликите (new image tags), автоматско ресоздавање на подови ако некој падне (self-healing), во проектот се

користат три деплојменти, еден за секој сервис: **auth**, **backend** и **frontend**. Кога ќе се пуштат нови Docker слики од GitHub Actions workflow, Kubernetes може да ги ажурира деплојментите со командата:

```
kubectl set image deployment/backend backend=kireboshkovski/backend:latest -n main
```

Ова иницира **rolling update**: новите подови се стартуваат постепено, а старите се гаснат кога новите стануваат Ready. Нема downtime бидејќи секогаш постојат активни реплики додека се ажурираат другите.

- **Ingress** во Kubernetes е ресурс кој управува со надворешниот пристап (HTTP/HTTPS) до сервисите во кластерот. Тоа е како „врата“ кон твоите подови. Со Ingress можеш да дефинираш:
  - hostnames (на пример expense-tracker.local)
  - URL патеки (routes)
  - SSL и CORS правила
  - load balancing и rewrite правила
  - Тука се користи IngressClassName: traefik, што значи дека Traefik ingress контролер ќе го обработува сообраќајот.

Како функционира во пракса:

1. Корисникот го отвора браузерот на http://expense-tracker.local.
2. Ingress го погледнува URL-то и го праќа барањето до соодветниот сервис:
   - /auth/login → auth service
   - /api/expenses → backend service
   - /dashboard → frontend service
3. Ако frontend сака да комуницира со backend или auth API, тоа може да биде преку патеките /api или /auth.

Со оваа конфигурација има **единствена влезна точка** за сите три сервиси, без потреба од директен пристап кон секој Pod или Service. Наместо да се користи сервис кој ќе биде NodePort / LoadBalancer се користи Ingress поради тоа што:

- Секој сервис има посебна IP/Port, за 3 сервиси – корисникот треба да знае 3 различни URL/Ports. Ќе се отежни рутирање кон патеки, ќе бидат потребни посебни DNS записи или порти.

- Може да креираме 3 LoadBalancer сервиси со кои ќе работи системот но, зошто кога за Ingress потребно е само 1 LoadBalancer за сите внатрешни сервиси.
- Доколку сакаме Https врска многу лесно можеме да го исконфигурираме Ingress.