Deploying a TODO app on Kubernetes

Bozorgi Behnam, Montali Simone, Murro Giuseppe

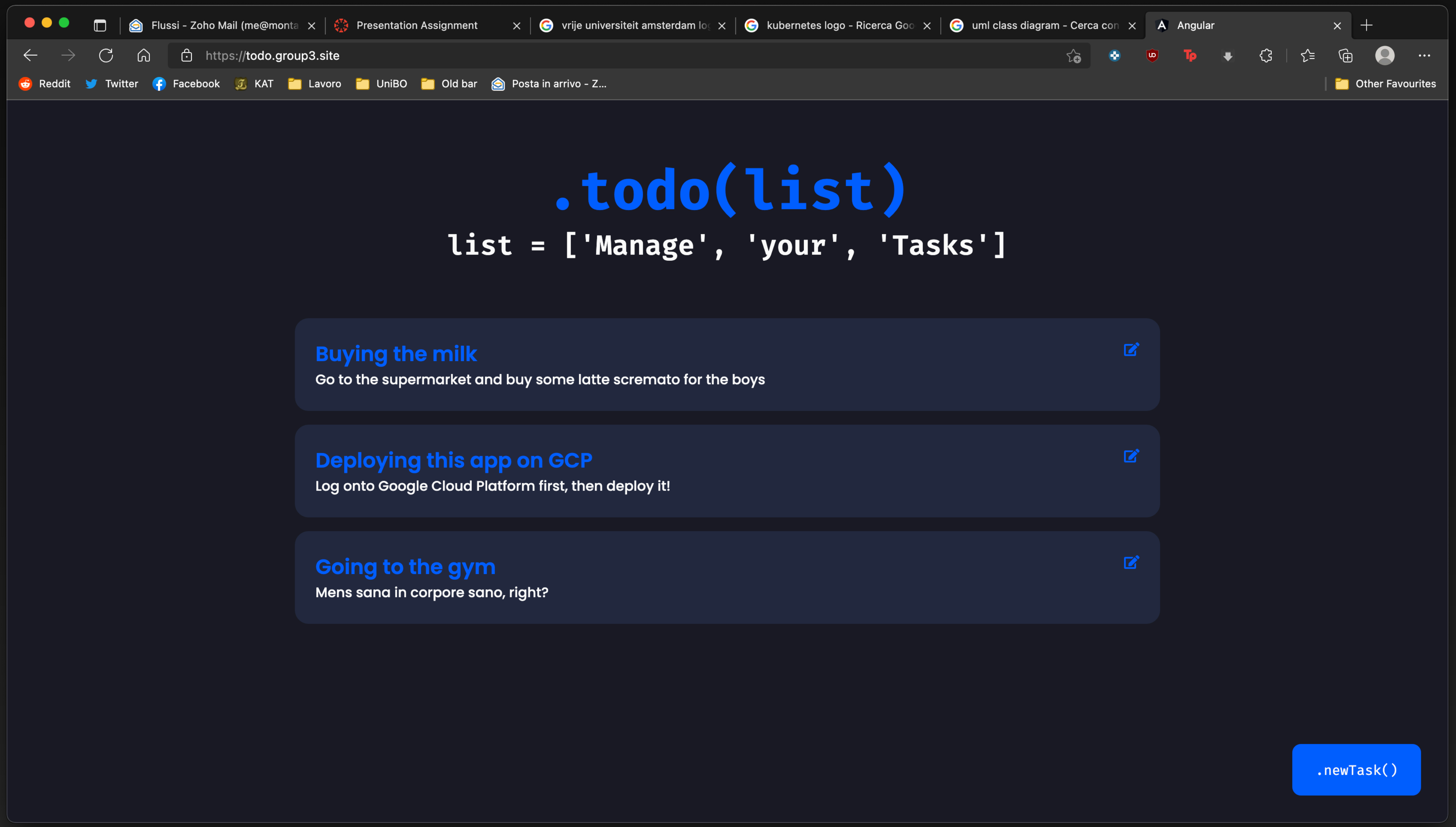
Group 3

VU UNIVERSITY AMSTERDAM

The TODO app

CRUD, made elegant.

- **TODO** app allowing simple **C**reate, **R**ead, **U**pdate, **D**elete operations
- Frontend built in **Angular**, reading data from the API
- API built in **Python** with **Flask**
- **MySQL** database, implemented on **MariaDB**



.todo(list)

list = ['Manage', 'your', 'Tasks']

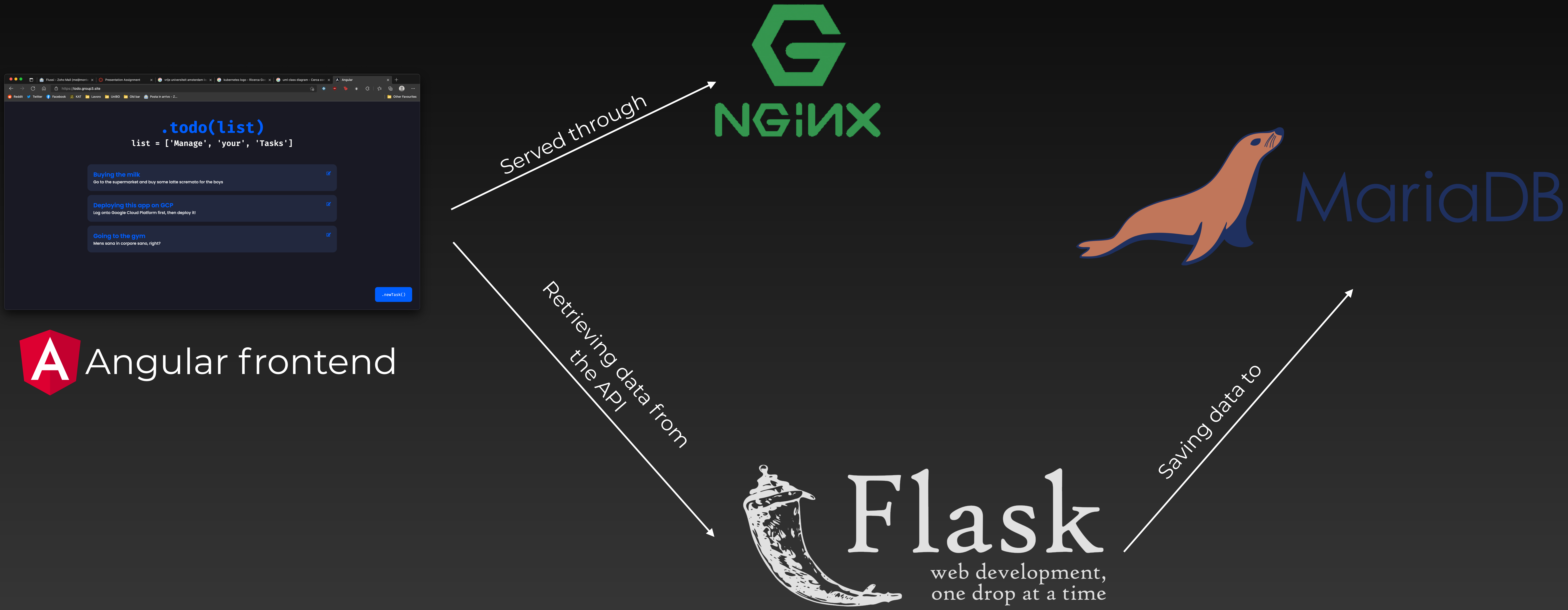
Buying the milk
Go to the supermarket and buy some latte scremato for the boys

Deploying this app on GCP
Log onto Google Cloud Platform first, then deploy it!

Going to the gym
Mens sana in corpore sano, right?

What's needed?

Building the app from scratch


Served through
NGINX
MariaDB
Retrieving data from the API
Saving data to
Flask
web development, one drop at a time
Angular frontend

https://todo.group3.site

.todo(list)

list = ['Manage', 'your', 'Tasks']

Buying the milk
Go to the supermarket and buy some latte scremato for the boys

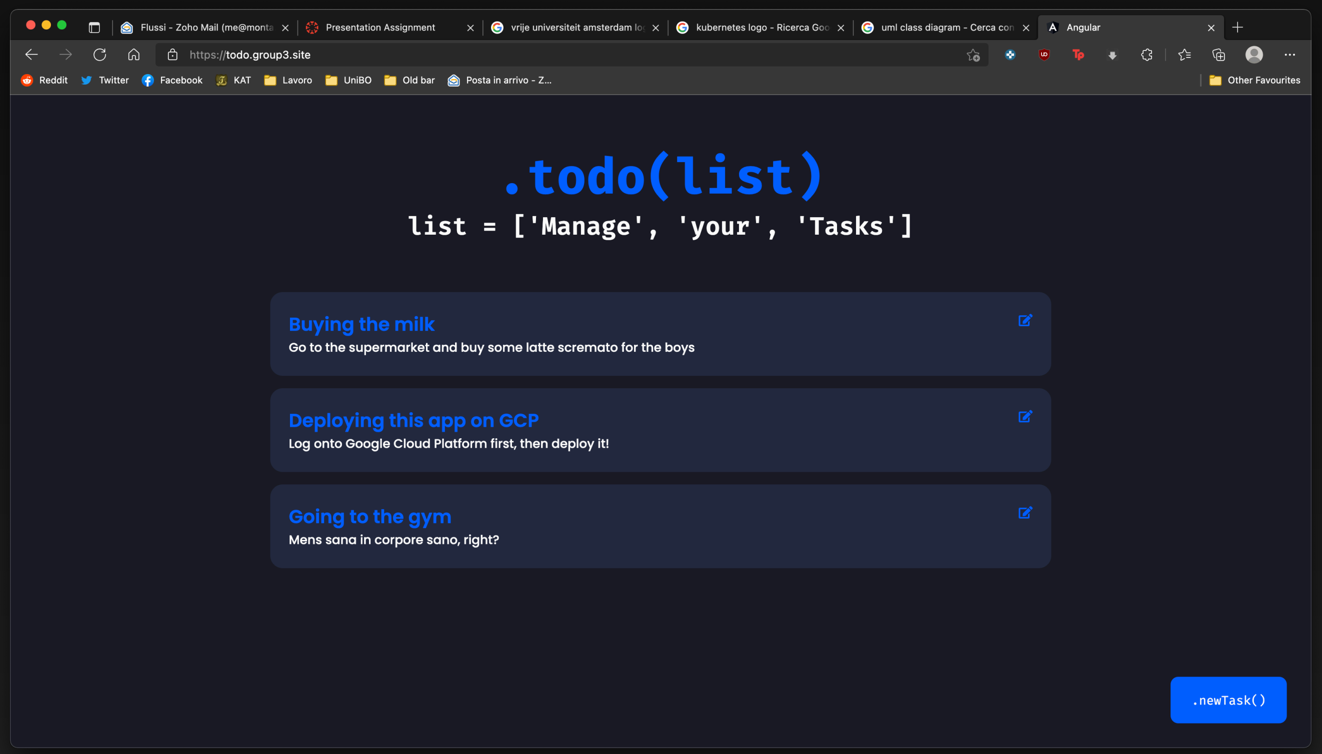
Deploying this app on GCP
Log onto Google Cloud Platform first, then deploy it!

Going to the gym
Mens sana in corpore sano, right?

.newTask()

Containerization

Abstracting from the host



https://todo.group3.site

.todo(list)

list = ['Manage', 'your', 'Tasks']

Buying the milk
Go to the supermarket and buy some latte scremato for the boys

Deploying this app on GCP
Log onto Google Cloud Platform first, then deploy it!

Going to the gym
Mens sana in corpore sano, right?

.newTask()

Angular frontend

Served through

NGINX

To containerize the frontend, we just need to:
- build the Angular project, getting HTML+CSS+JS
- Instantiate an *nginx* container
- Copy the files into the container

This is done with a simple Dockerfile:

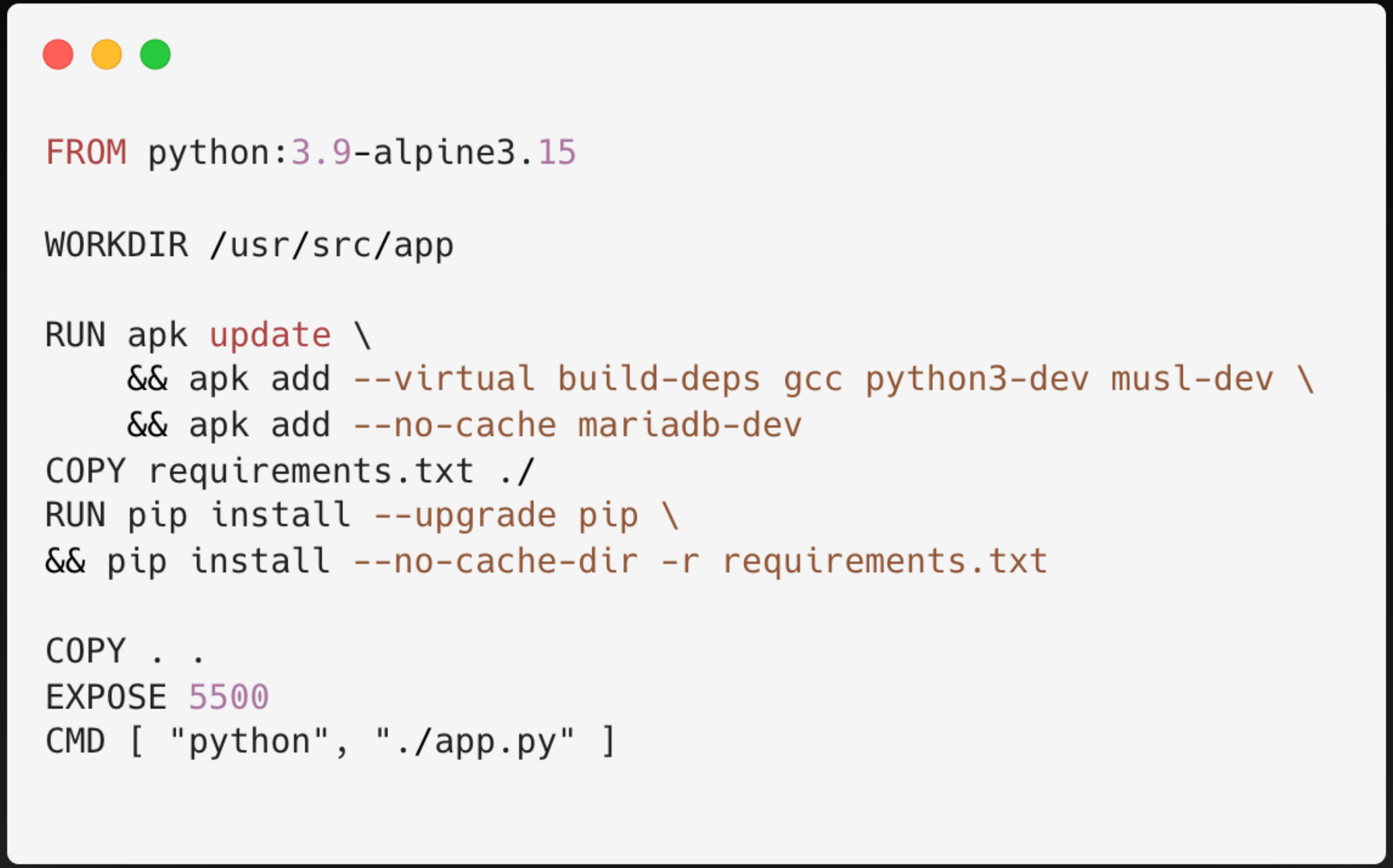
```dockerfile
FROM nginx
EXPOSE 80
COPY dist/angular /usr/share/nginx/html
```

Containerization

Abstracting from the host

To containerize the API, we can use the *python:alpine* image

- Install the requirements
- Expose the correct port
- Launch the API script



```
FROM python:3.9-alpine3.15

WORKDIR /usr/src/app

RUN apk update \
    && apk add --virtual build-deps gcc python3-dev musl-dev \
    && apk add --no-cache mariadb-dev
COPY requirements.txt ./
RUN pip install --upgrade pip \
&& pip install --no-cache-dir -r requirements.txt

COPY . .
EXPOSE 5500
CMD [ "python", "./app.py" ]
```

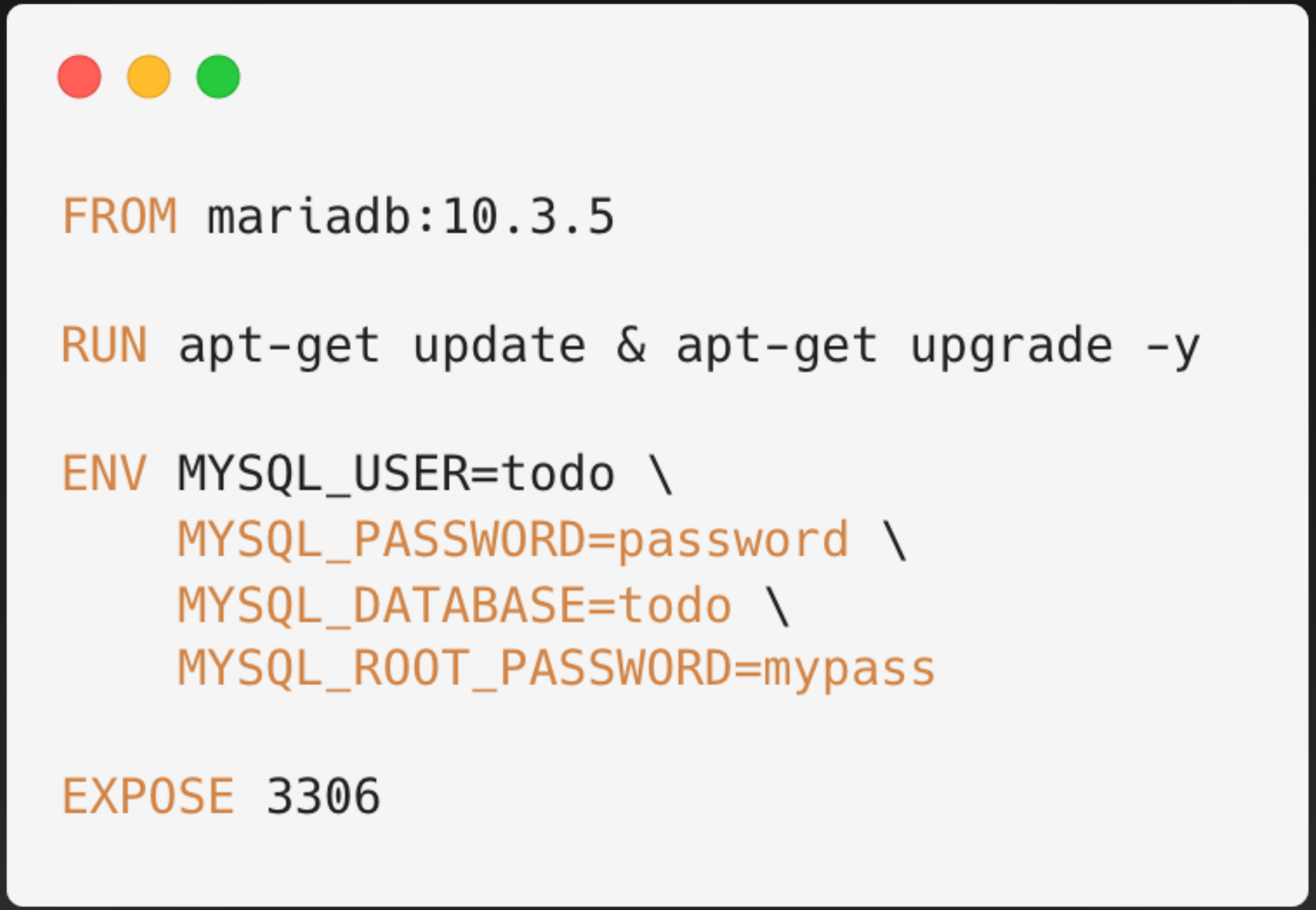
Retrieving data from the API

Flask
web development, one drop at a time

Containerization

Abstracting from the host

To containerize the DB, we can use the *mariadb:latest* image
- Setting the parameters for DB name, user, password
- Exposing the 3306 port

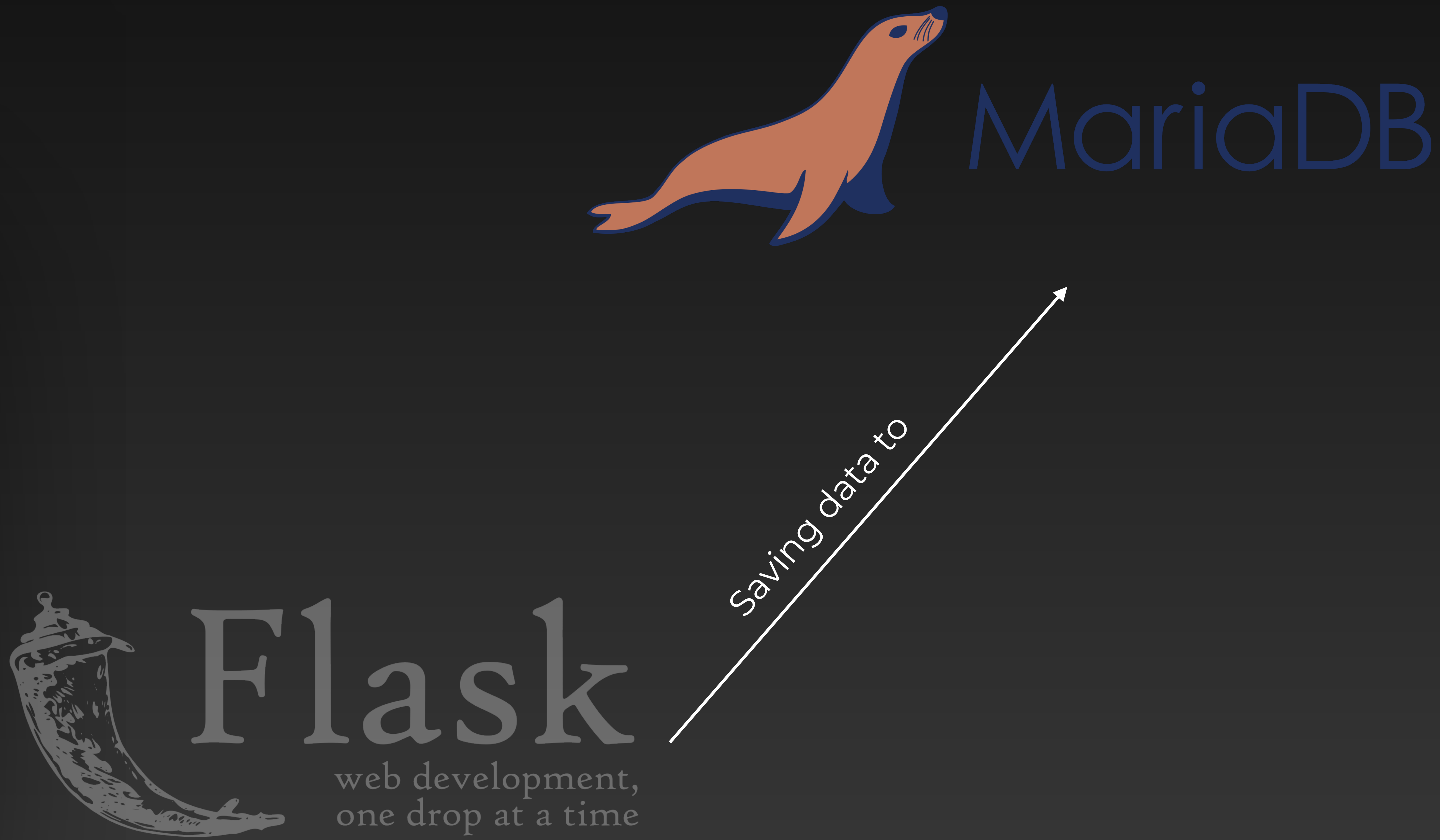


```
FROM mariadb:10.3.5

RUN apt-get update & apt-get upgrade -y

ENV MYSQL_USER=todo \
    MYSQL_PASSWORD=password \
    MYSQL_DATABASE=todo \
    MYSQL_ROOT_PASSWORD=mypass

EXPOSE 3306
```



Bringing it to Kubernetes

Containers have to be orchestrated!

- **Kubernetes** allows us to manage the orchestration of containers easily
- We will need to define different components: deployments, services, volumes, configs and secrets.
- The deployment will then have to **horizontally scale** when needed

General schema

Bringing it to Kubernetes

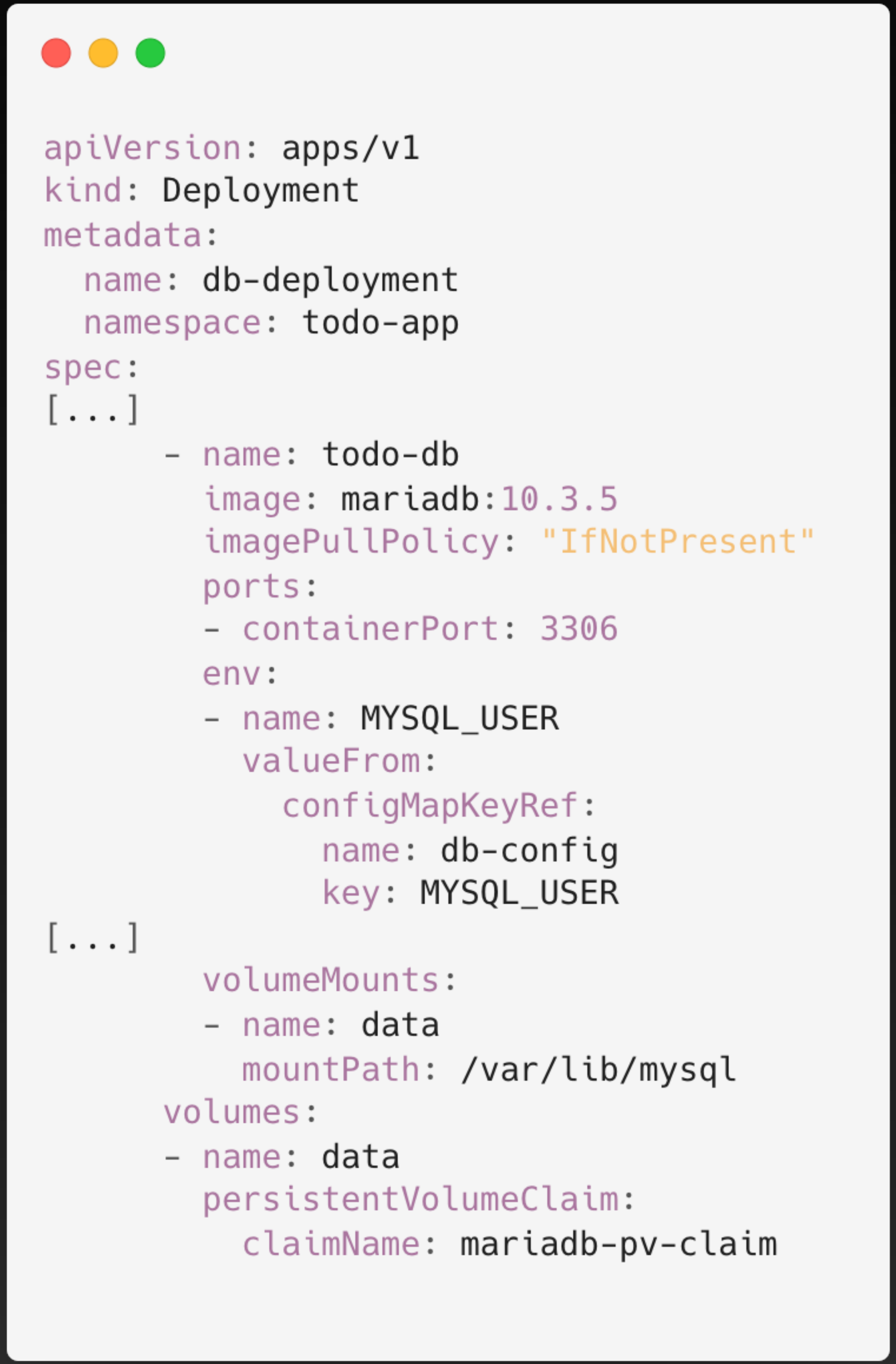
Containers have to be orchestrated!

UML schema

Deployments

Containers have to be orchestrated!

- **Deployments** are workload resources allowing us to make sure our containers are always working as they should
  - The **DB** deployment specifies which ConfigMaps to use for the env, which image to use, which volumes and secrets.
  - The **API** deployment specifies the container and the same ConfigMaps that are used for the DB, in order to connect to it
  - The **UI** deployment sets up an nginx container on port 80



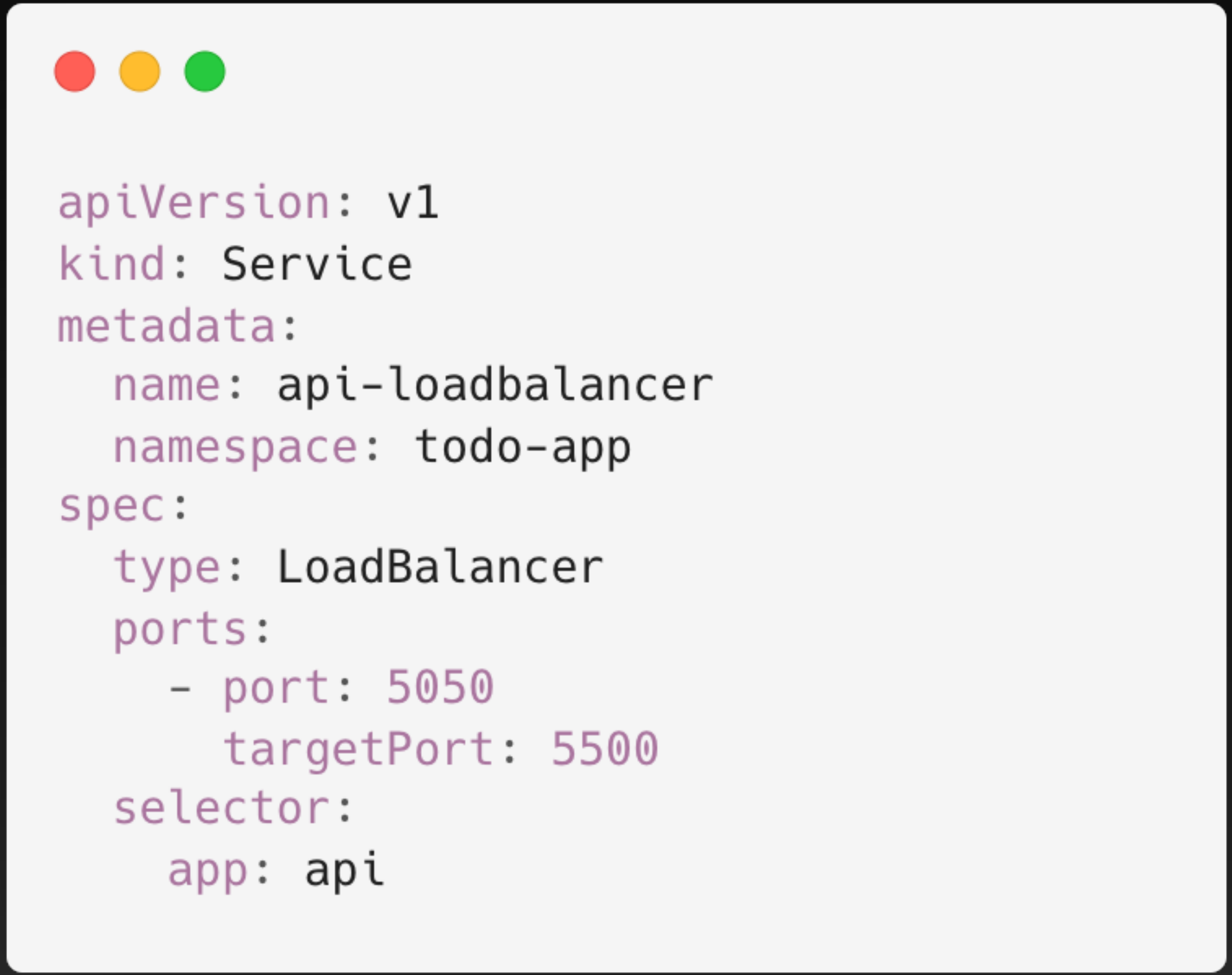
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db-deployment
  namespace: todo-app
spec:
[...]
      - name: todo-db
        image: mariadb:10.3.5
        imagePullPolicy: "IfNotPresent"
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: db-config
              key: MYSQL_USER
[...]
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: mariadb-pv-claim
```

Services

Exposing our software

- **Services** define logical sets of pods and allow us to access them
  - The **DB** service creates a *ClusterIP*, allowing us to reach the DB at a given IP:port
  - The **API** service creates a Load Balancer that equally distributes the requests across the available pods
  - The **UI** service creates a LoadBalancer as the API one

To use the LoadBalancers, we'll need MetalLB (described later)

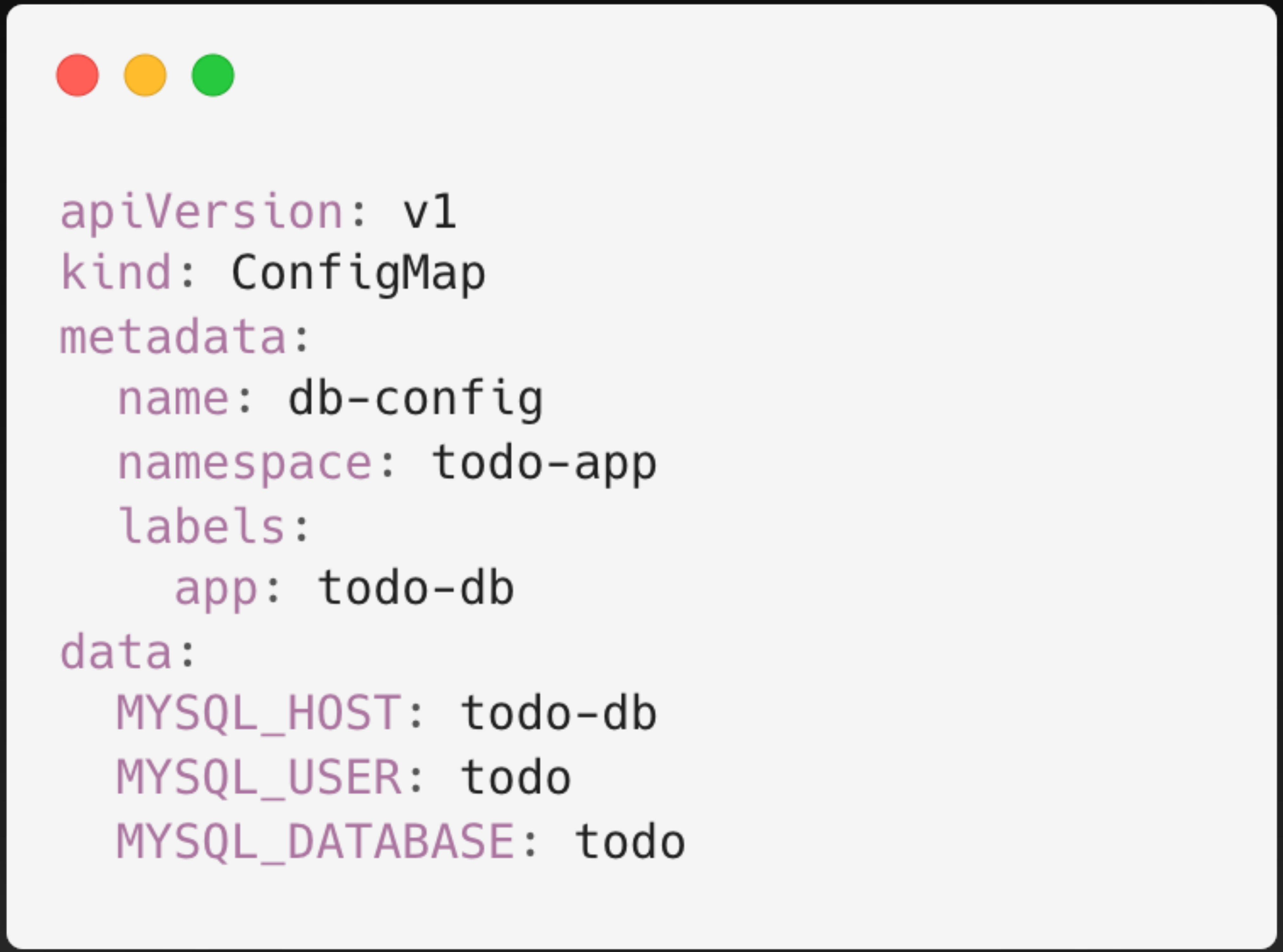


```yaml
apiVersion: v1
kind: Service
metadata:
  name: api-loadbalancer
  namespace: todo-app
spec:
  type: LoadBalancer
  ports:
    - port: 5050
      targetPort: 5500
  selector:
    app: api
```

ConfigMaps

Saving parameters

- ConfigMaps allow us to save environment variables and configuration parameters
  - The DB config contains the MySQL users, password, DB name
  - The API service is linked to the same config.

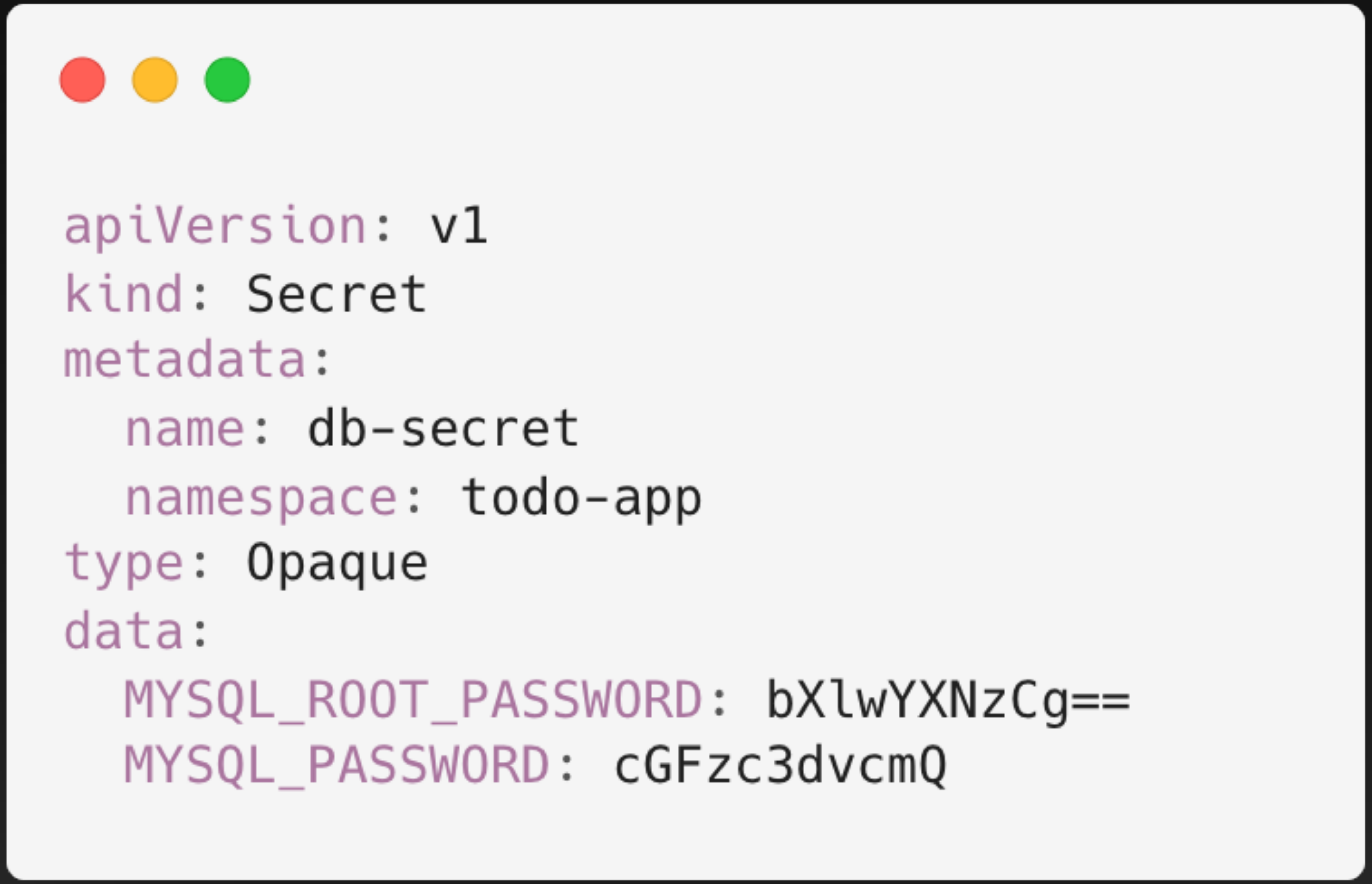


```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
  namespace: todo-app
  labels:
    app: todo-db
data:
  MYSQL_HOST: todo-db
  MYSQL_USER: todo
  MYSQL_DATABASE: todo
```

Secrets

Saving confidential data

- **Secrets** contain base64 encrypted data, as passwords and TLS keys
  - The **DB** password is saved in a secret
  - The **TLS** keys are saved in special secrets

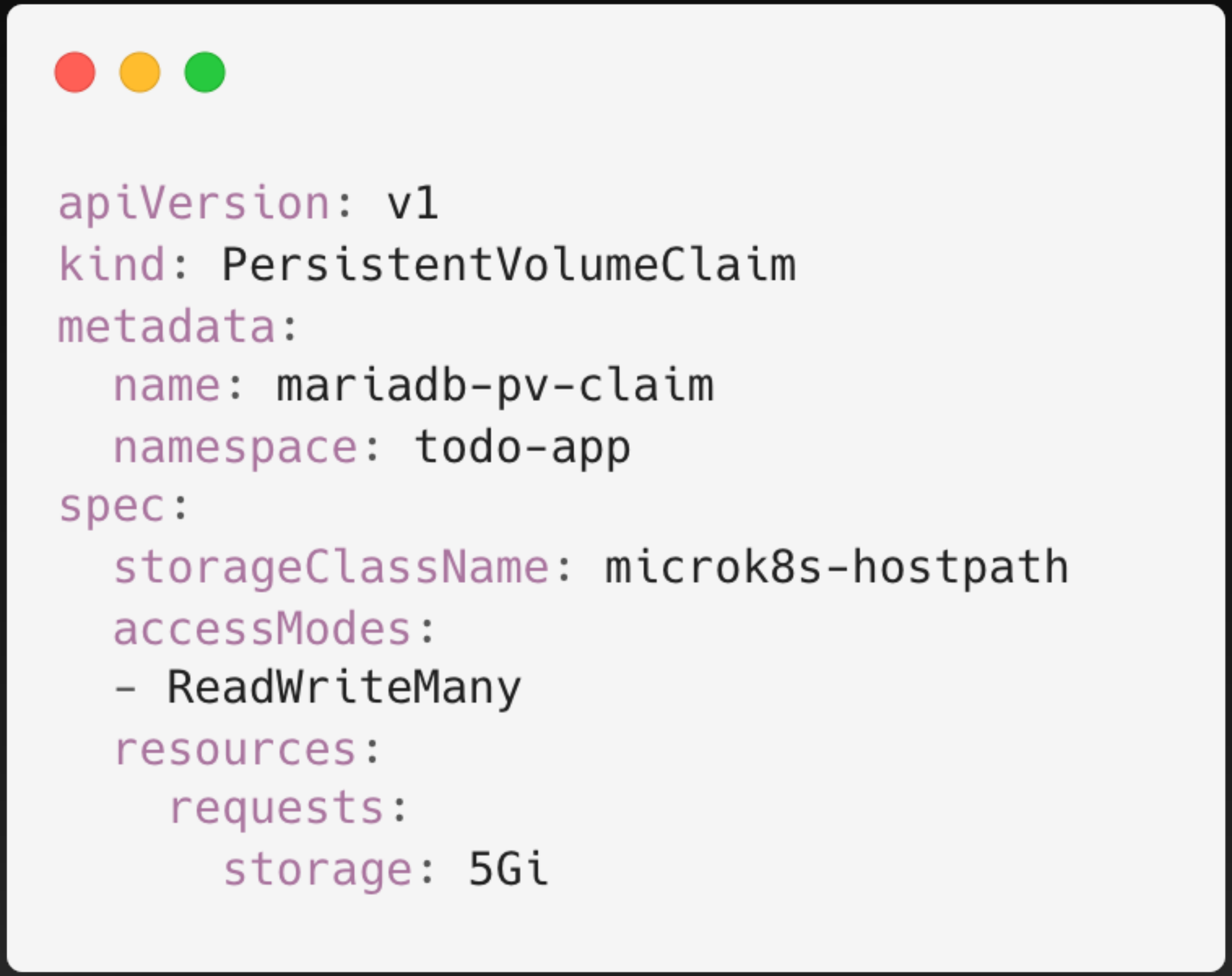


```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: todo-app
type: Opaque
data:
  MYSQL_ROOT_PASSWORD: bXlwYXNzCg==
  MYSQL_PASSWORD: cGFzc3dvcmQ
```

Volumes

Maintaining the data

- **Volumes** allow us to save data to a persistent volume, that doesn't disappear with the pod
  - The **DB** needs a persistent volume, obtained through a **PersistentVolumeClaim** that is dynamically managed by Kubernetes

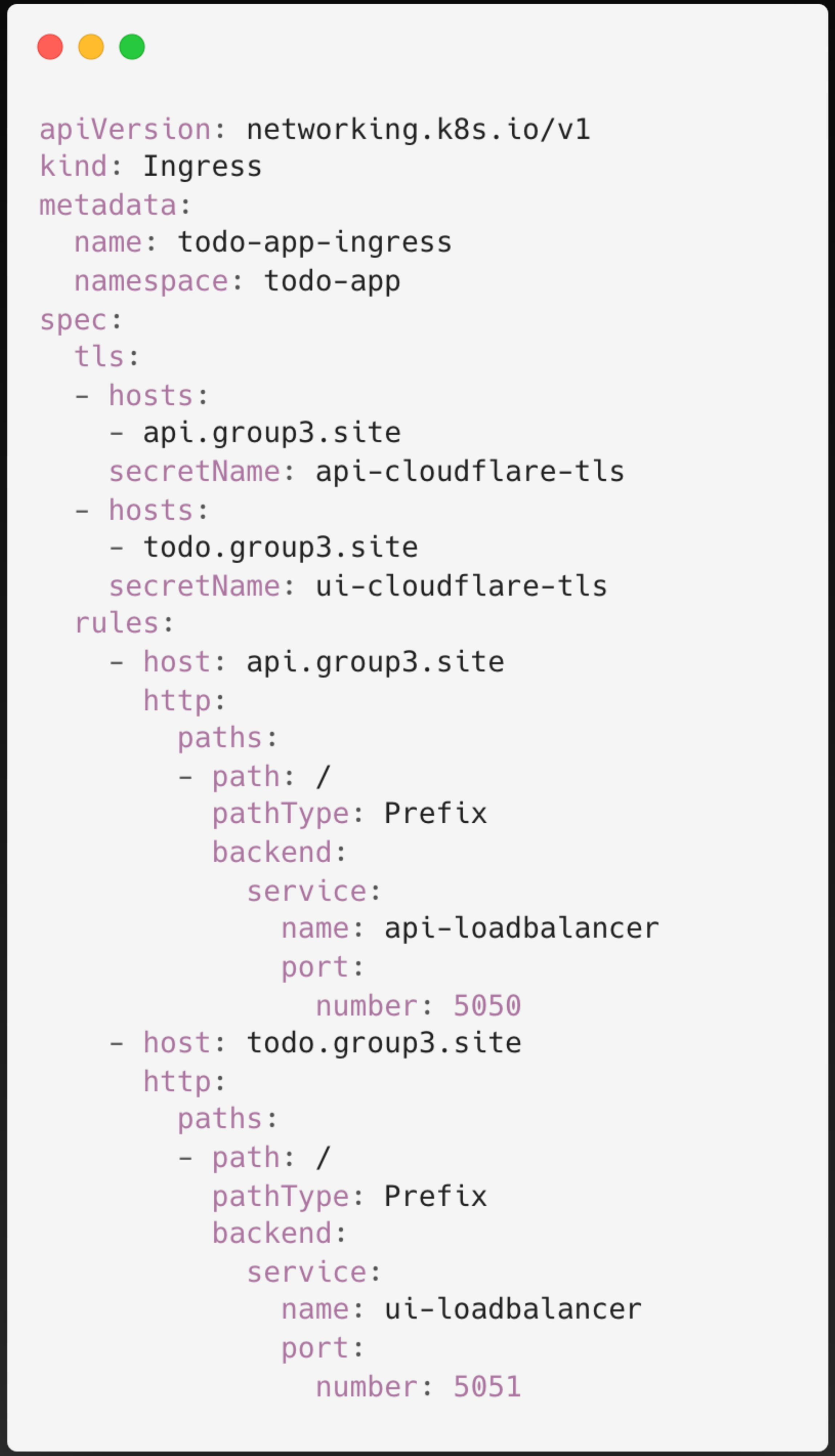


```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb-pv-claim
  namespace: todo-app
spec:
  storageClassName: microk8s-hostpath
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Ingress

Managing HTTP access

- **Ingress** is an external component providing TLS support and name-based virtual hosting
- Using Ingress, we can use **subdomains** for the APIs and the frontend, without having to use different ports



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: todo-app-ingress
  namespace: todo-app
spec:
  tls:
  - hosts:
    - api.group3.site
    secretName: api-cloudflare-tls
  - hosts:
    - todo.group3.site
    secretName: ui-cloudflare-tls
  rules:
    - host: api.group3.site
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: api-loadbalancer
              port:
                number: 5050
    - host: todo.group3.site
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: ui-loadbalancer
              port:
                number: 5051
```

Prerequisites

LoadBalancer

Distributing the requests.

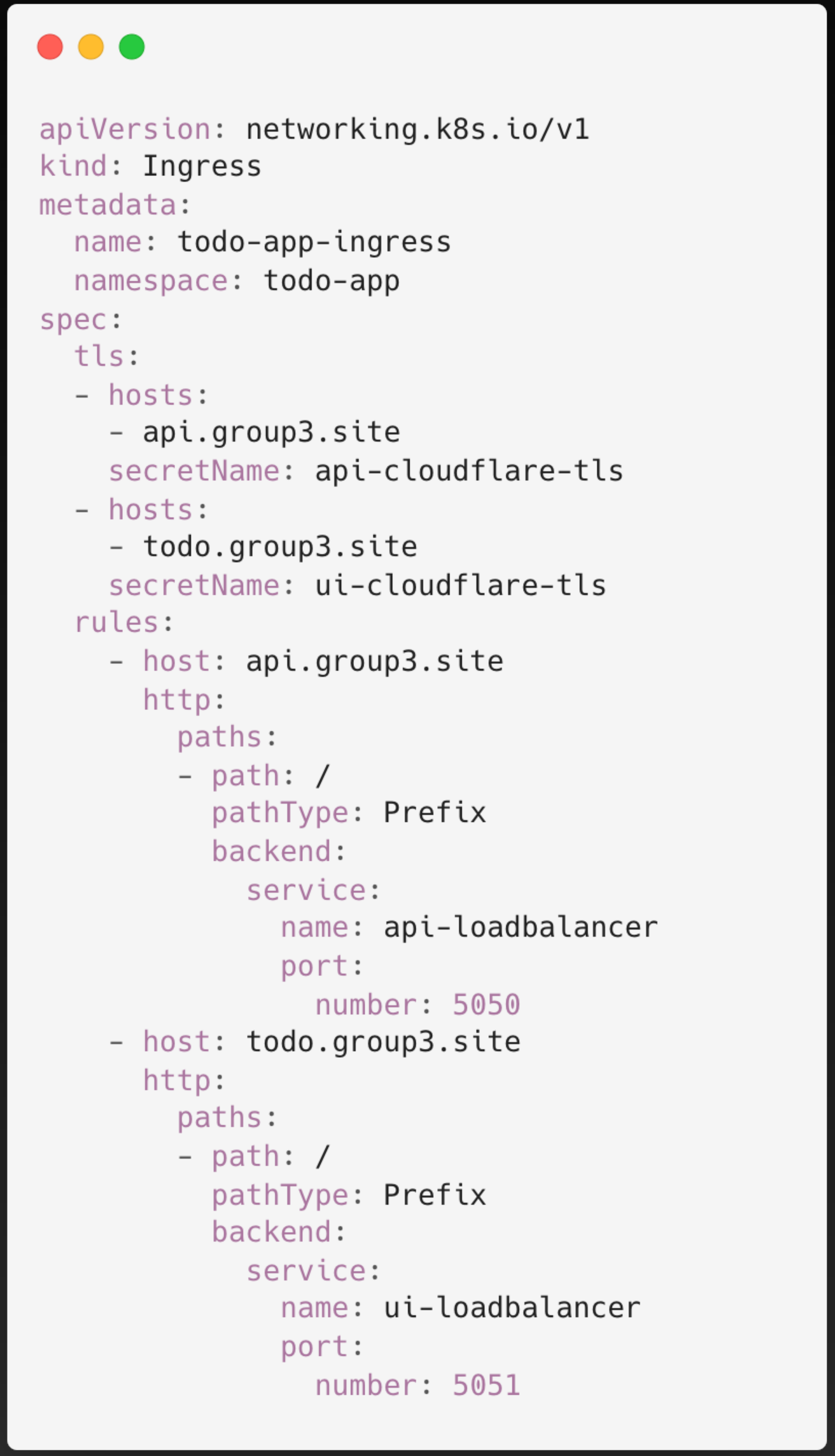
- Kubernetes doesn't offer an implementation for the LoadBalancer object
- **MetalLB** is an implementation for bare-metal Kubernetes clusters
- The LoadBalancer distributes the requests across the different pods
- It can be installed by applying the official manifest
- It requires the definition of a **range of available IPs**, which we set as:

```
192.168.1.60-192.168.1.80
```

LoadBalancer

Distributing the requests.

- Kubernetes doesn't offer an implementation for the LoadBalancer object
- MetalLB is an implementation for bare-metal Kubernetes clusters
- The LoadBalancer distributes the requests across the different pods

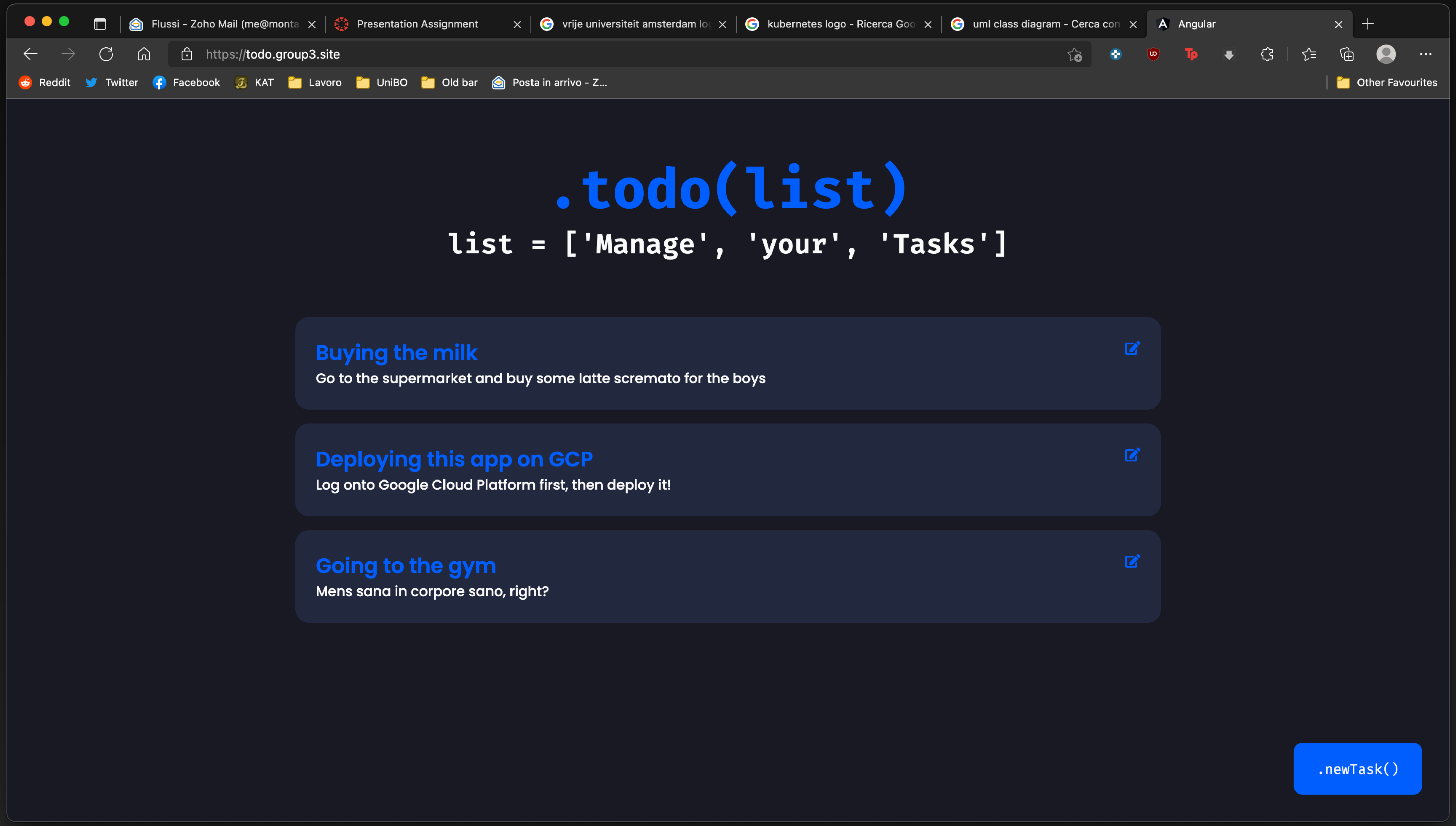


```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: todo-app-ingress
  namespace: todo-app
spec:
  tls:
  - hosts:
    - api.group3.site
    secretName: api-cloudflare-tls
  - hosts:
    - todo.group3.site
    secretName: ui-cloudflare-tls
  rules:
    - host: api.group3.site
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: api-loadbalancer
              port:
                number: 5050
    - host: todo.group3.site
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: ui-loadbalancer
              port:
                number: 5051
```

Some numbers

How the project was built, one step at a time.

- **TODO** app allowing simple **C**reate, **R**ead, **U**pdate, **D**elete operations
- Frontend built in **Angular**, reading data from the API
- API built in **Python** with **Flask**
- **MySQL** database, implemented on **MariaDB**



.todo(list)

list = ['Manage', 'your', 'Tasks']

Buying the milk
Go to the supermarket and buy some latte scremato for the boys

Deploying this app on GCP
Log onto Google Cloud Platform first, then deploy it!

Going to the gym
Mens sana in corpore sano, right?