

Deploying a TODO app on Kubernetes

Bozorgi Behnam, Montali Simone, Murro Giuseppe



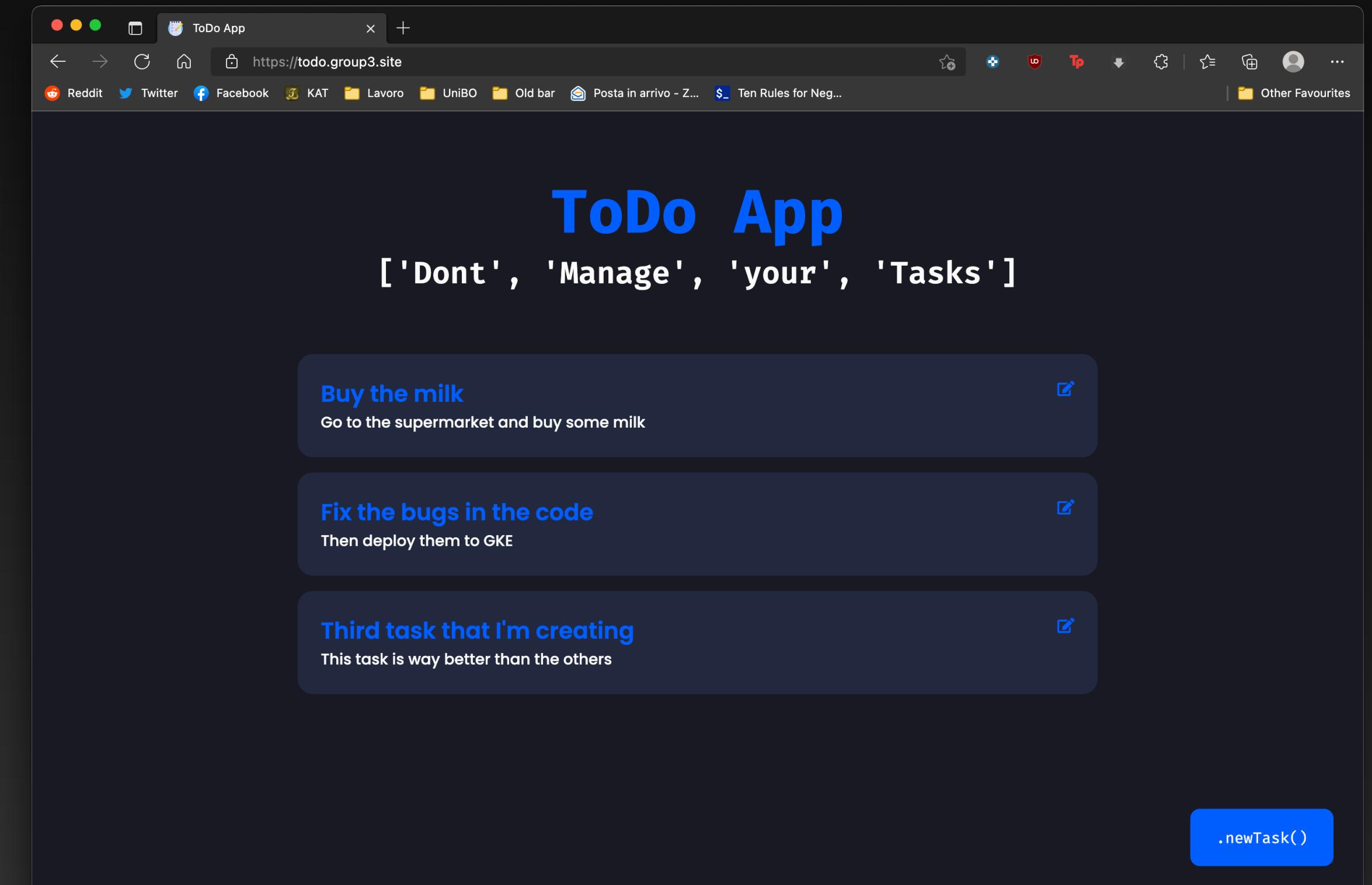
Group 3

Architecture

The TODO app

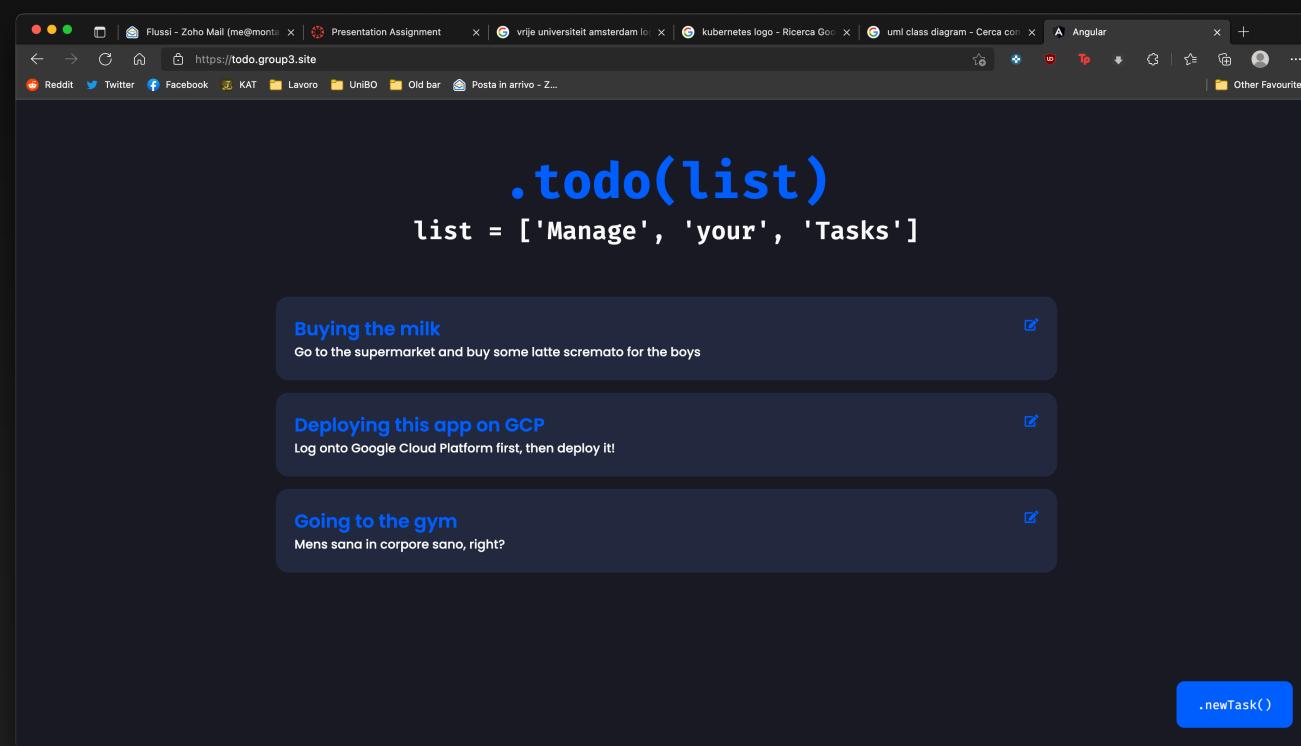
CRUD, made elegant.

- **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**



What's needed?

Building the app from scratch

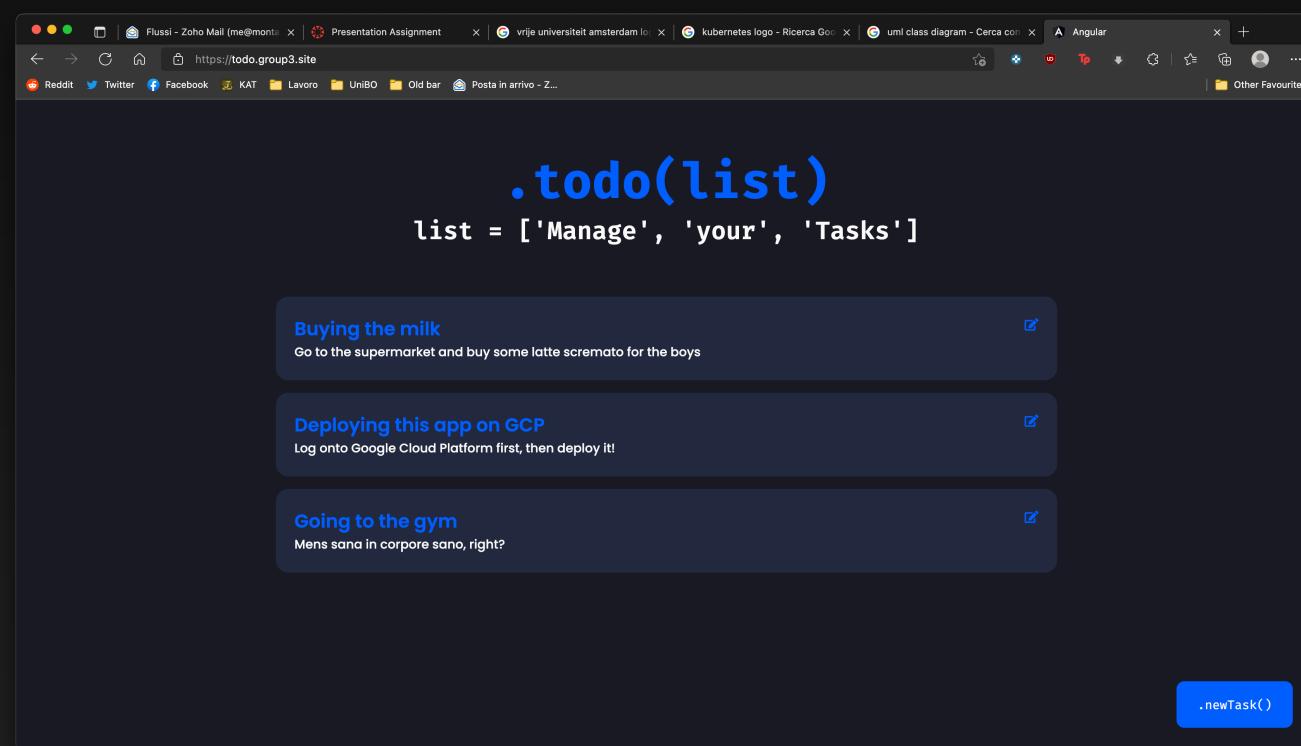


A Angular frontend



Containerization

Abstracting from the host



A Angular frontend

Served through



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:



```
FROM node:latest as build  
WORKDIR /usr/local/app  
COPY ./ /usr/local/app/  
RUN npm install  
RUN npm run build
```

```
FROM nginx:latest  
COPY --from=build  
/usr/local/app/dist/angular  
/usr/share/nginx/html  
EXPOSE 80
```

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

Retrieving data from
the API

```
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" ]
```



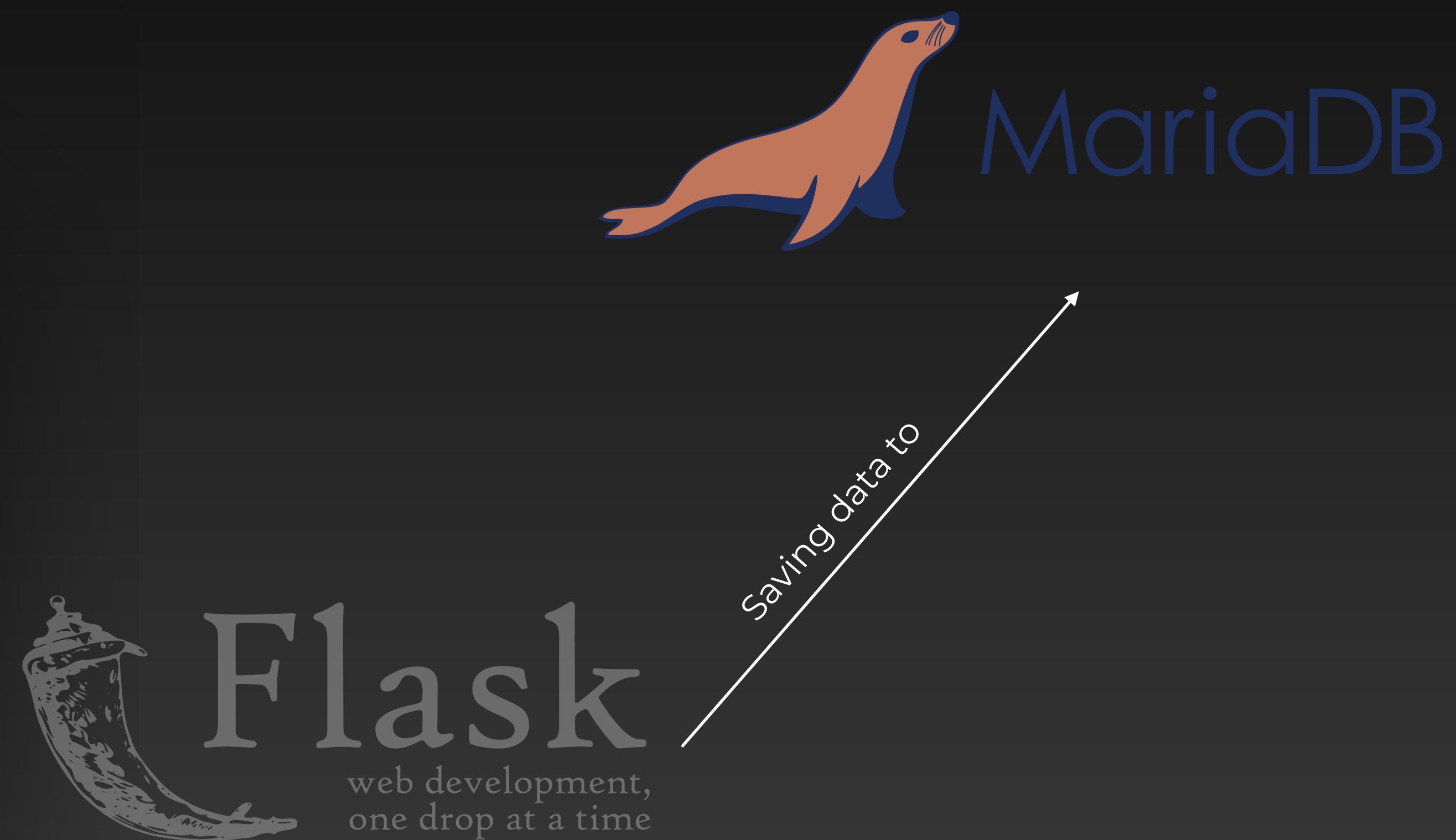
Containerization

Abstracting from the host

To containerize the DB, we can use the
mariadb:10.3.5 image

- Setting the parameters for DB name, user, password
- Exposing the 3306 port

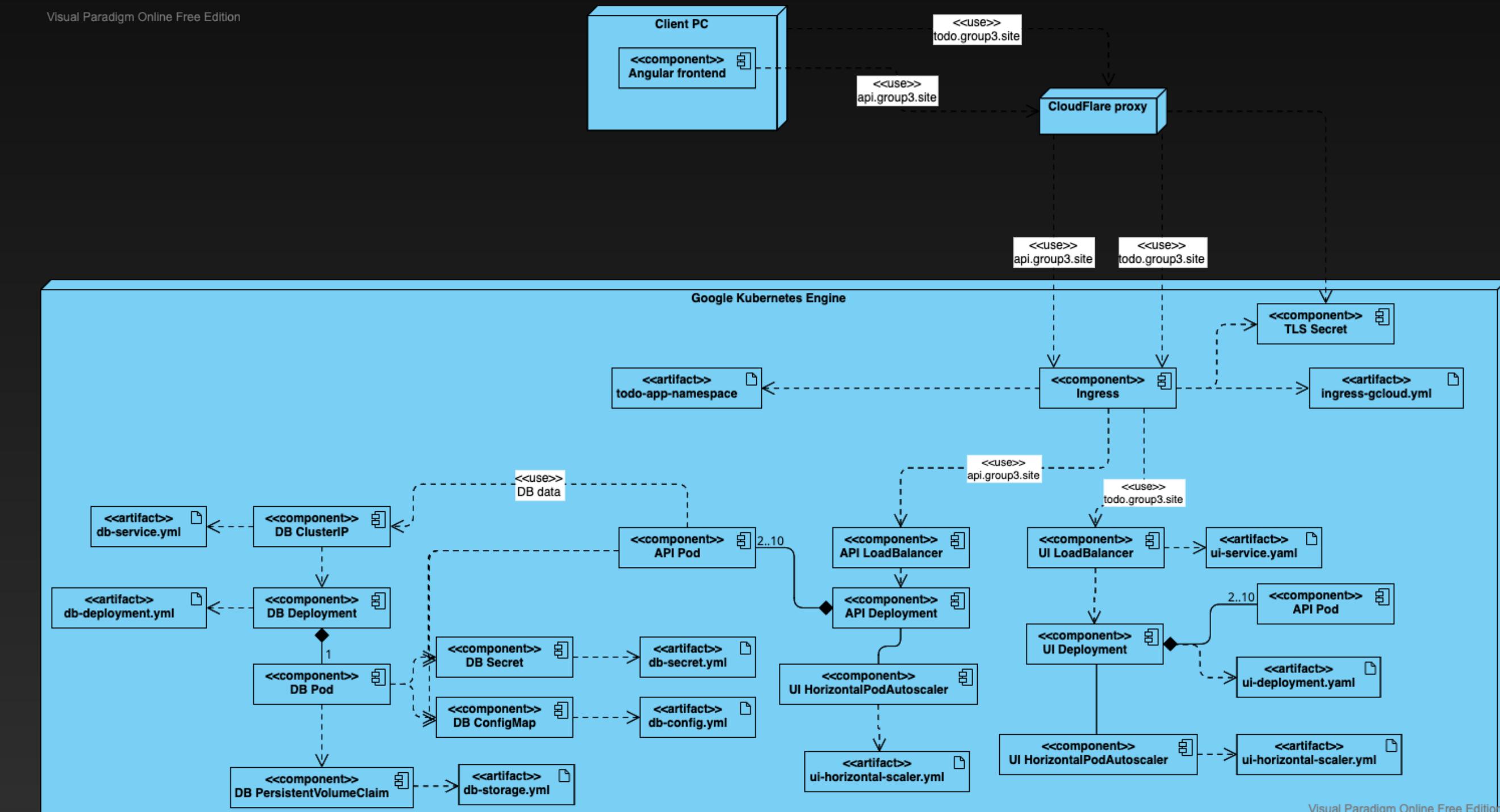
```
● ● ●  
FROM mariadb:10.3.5  
  
RUN apt-get update & apt-get upgrade -y  
  
ENV MYSQL_USER=todo \  
      MYSQL_PASSWORD=password \  
      MYSQL_DATABASE=todo \  
      MYSQL_ROOT_PASSWORD=mypass  
  
EXPOSE 3306
```



Bringing it to Kubernetes

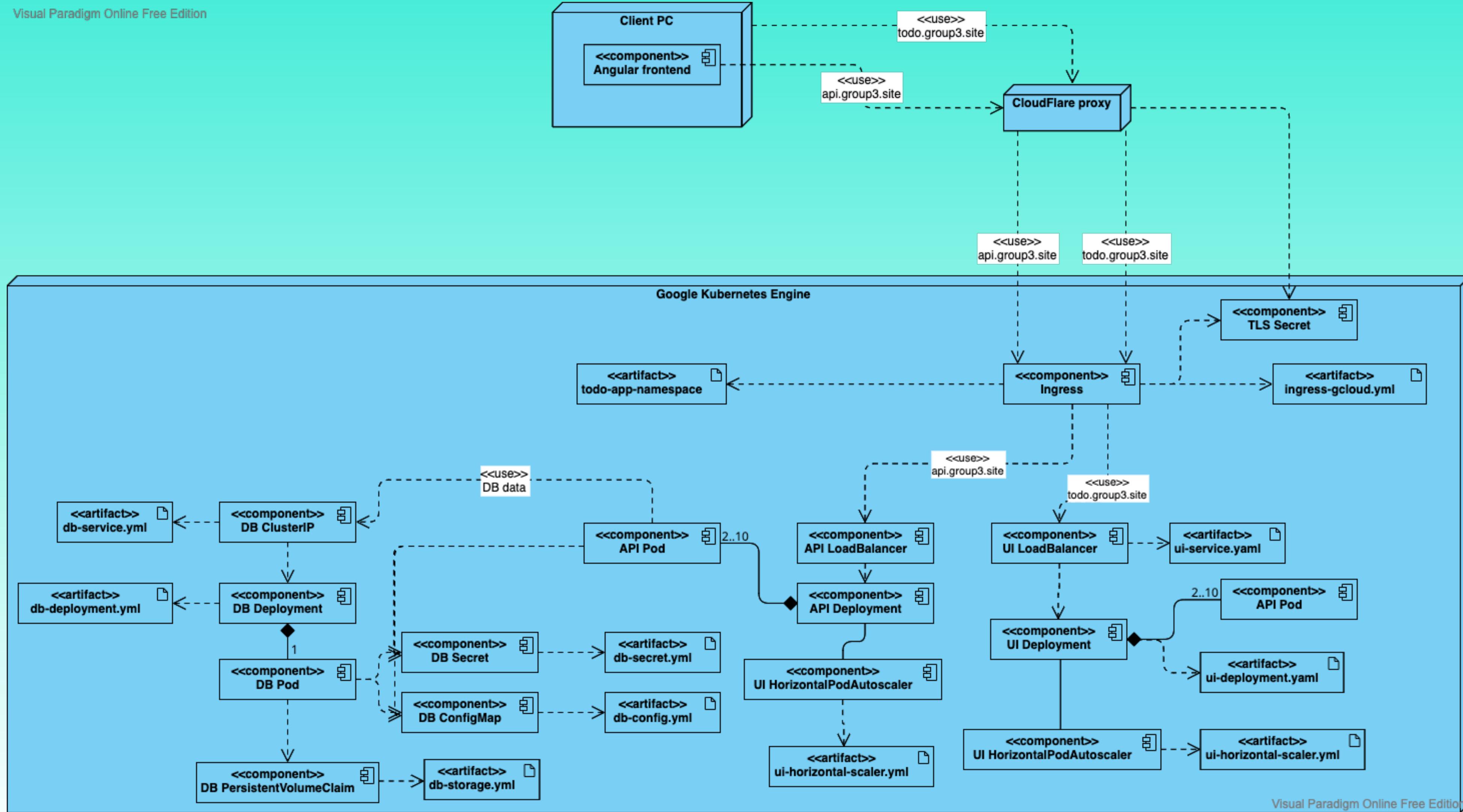
Containers have to be orchestrated!

- **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



Bringing it to Kubernetes

Containers have to be orchestrated!



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

```
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



```
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, DB name and hostname
 - The **API** service is linked to the same config.



```
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

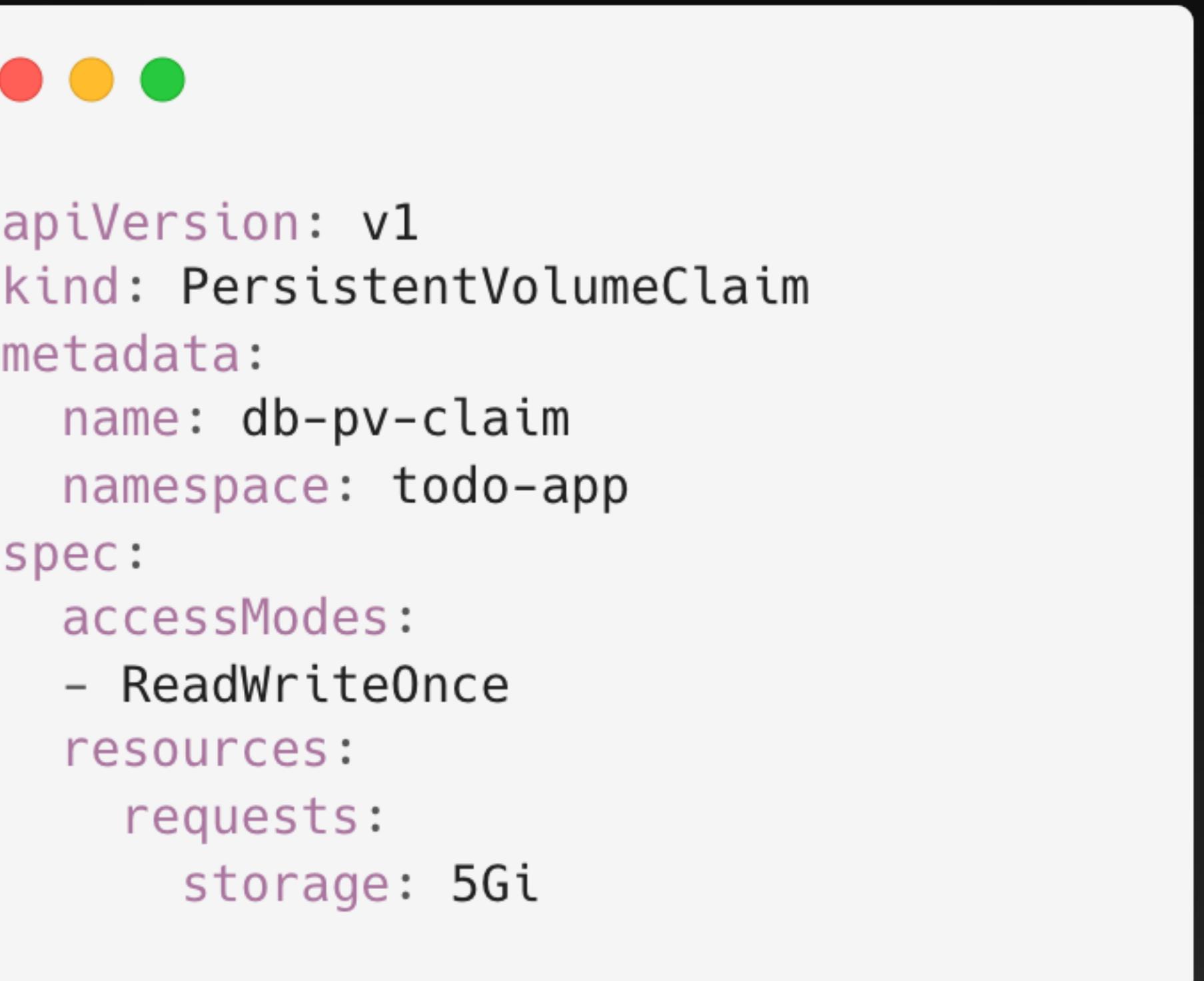
```
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: todo-app
type: Opaque
data:
  MYSQL_ROOT_PASSWORD: bXlwYXNzCg==
  MYSQL_PASSWORD: cGFzc3dvcmQ
```

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

```
~/Vrije/Lab/Containerization/todo-app ➔ main • ?  
$ kubectl get sc standard -n todo-app  
NAME      PROVISIONER      RECLAIMPOLICY      VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION      AGE  
standard (default)  kubernetes.io/gce-pd  Delete           Immediate           true                5d3h  
  
~/Vrije/Lab/Containerization/todo-app ➔ main • ?  
$ kubectl get pv -n todo-app  
NAME          CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM          STORAGECLASS  
pvc-5fffd43-88cd-4582-a3b0-b7ab48513e09  5Gi        RWO          Delete        Bound    todo-app/db-pv-claim  standard
```



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The terminal displays a YAML configuration for a PersistentVolumeClaim:

```
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: db-pv-claim  
  namespace: todo-app  
spec:  
  accessModes:  
    - ReadWriteOnce  
  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

```
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

Google Kubernetes Engine

Deploying to the cloud

Our cluster's IP

34.107.151.112

- GKE allows us to deploy the application in Google's cloud servers
- We picked the **Eemshaven, NL**, datacenter as the latency is minimal
- The cluster is deployed on **3 nodes**



| Nome ↑ | Stato | CPU richiesta | CPU allocabile | Memoria richiesta | Memoria allocabile |
|---|---------|---------------|----------------|-------------------|--------------------|
| gke-todo-cluster-default-pool-62352e64-fh1h | ✓ Ready | 851 mCPU | 940 mCPU | 588.25 MB | 2.96 GB |
| gke-todo-cluster-default-pool-62352e64-tqtz | ✓ Ready | 363 mCPU | 940 mCPU | 341.35 MB | 2.96 GB |
| gke-todo-cluster-default-pool-62352e64-wivc | ✓ Ready | 803 mCPU | 940 mCPU | 446.69 MB | 2.96 GB |

LoadBalancer

Distributing the requests.

- Kubernetes doesn't offer an implementation for the LoadBalancer object
- **MetalLB** is an implementation for bare-metal Kubernetes clusters, used for MicroK8s (used in the first stages of the project)
- Google Kubernetes Engine uses Google's **Cloud Load Balancer**, an efficient implementation
- The load balancers are **automatically instantiated** when a Service (type: LoadBalancer) is added

The screenshot shows a web browser window with multiple tabs open, including DNS, Load b., todo-cl, todo.gr, Overview, and What is Cloud. The main content is the Google Cloud Load Balancing page at https://cloud.google.com/load-balancing. The page title is "Cloud Load Balancing" and it describes "High performance, scalable load balancing on Google Cloud Platform". It features a "Go to console" button and a link to "View documentation for this product". Below this, there is a section titled "Worldwide autoscalin balancing" with a detailed description of how Cloud Load Balancing works. The main area is a table titled "LOAD BALANCERS" under the "Load balancing" tab. The table has columns for "Name", "Load balancer type", and "Actions". It lists four load balancers:

| Name | Load balancer type |
|---|-----------------------------|
| k8s2-um-ialj3161-todo-app-todo-ingress-qt0qbcjh | HTTP(S) (Classic) |
| a23e7b88452284e07bf93d9579130af6 | Network (target pool-based) |
| aabdd61a41fab4cdca11a86aa0b4bf5e | Network (target pool-based) |

Ingress and TLS

Managing HTTP requests and SSL

- Installing Ingress is instant on MicroK8s (microk8s enable ingress) but it can be done from manifests too
- **The TLS** configuration requires a TLS certificate, either self-signed or made by a certificate authority
- Self-signed certificates generate **errors on most browsers**, so they're not an option
- **CloudFlare** offers the possibility of a double encryption: from the server to CloudFlare proxy, from the proxy to the final user
- This requires using an **origin CA certificate** with Ingress

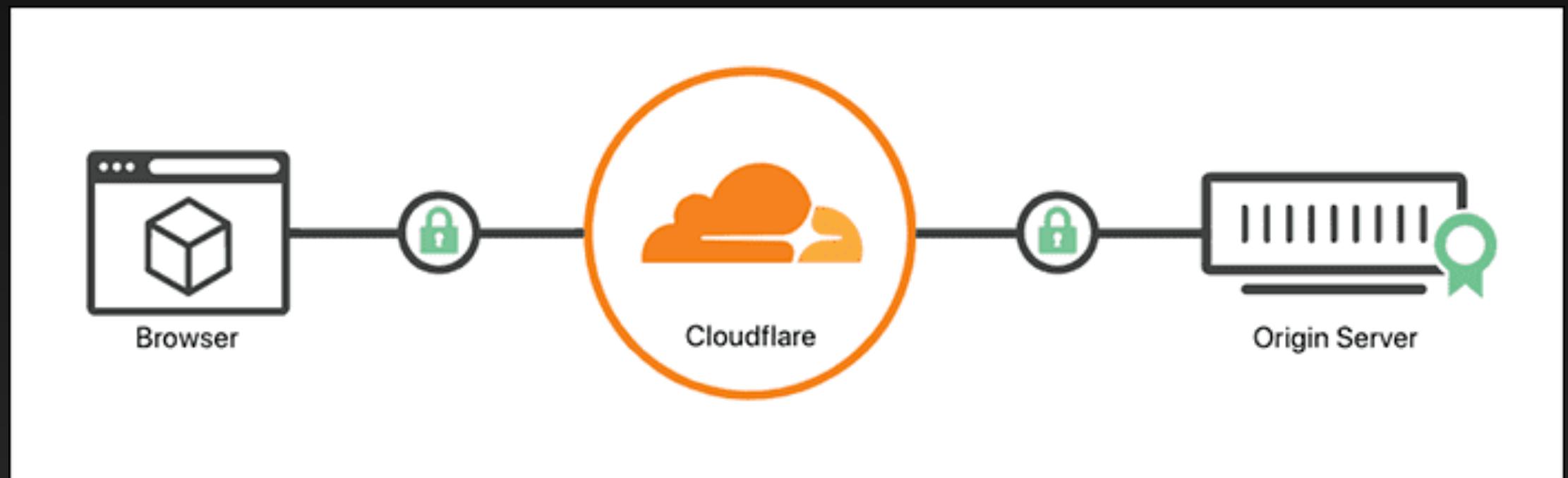
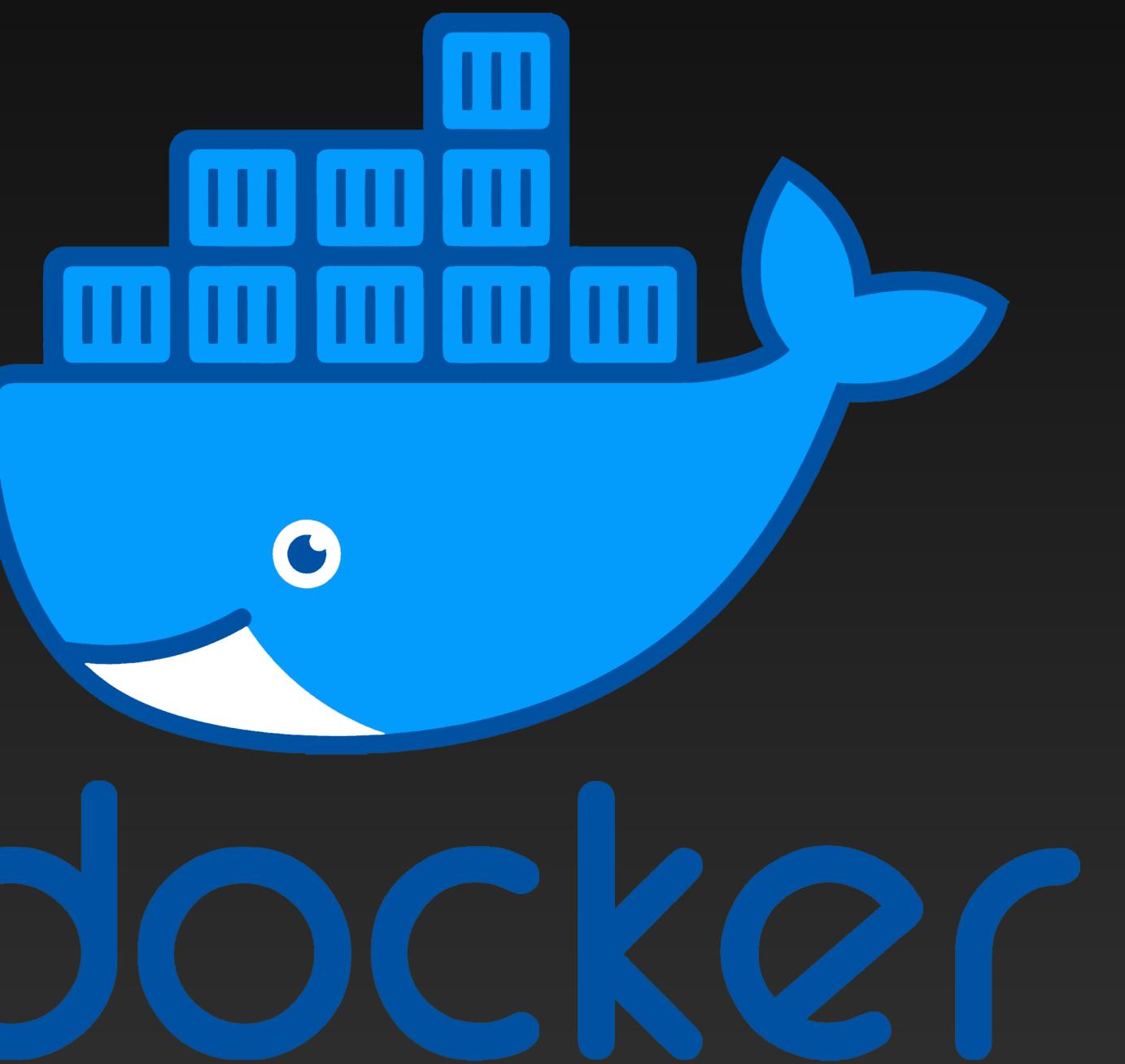


Image registry

Providing images to Kubernetes

- By default, Kubernetes pulls images from Docker Hub
- Google Kubernetes Engine provides an image registry through the **Container Registry** service
- After activating the API, it's sufficient to build and push images:

```
gcloud builds submit --tag gcr.io/todo-app-339413/todo-db:v1.0
```



Network policies

Providing images to Kubernetes

- By default, Kubernetes pods **accept traffic** from any source in the Cluster
- This is not a safe behaviour: we'd like to **isolate the pods** that don't need connections
- The DB will only be accessible by the API
- The API and the UI have to be **publicly accessible**
- In order for Network Policies to be effective, the correct option has to be enabled when creating the GKE cluster

```
● ● ●  
apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: api-db-egress-network-policy  
  namespace: todo-app  
spec:  
  podSelector:  
    matchLabels:  
      app: api  
  egress:  
    - to:  
      - podSelector:  
          matchLabels:  
            app: db  
        ports:  
          - port: 3306  
            protocol: TCP  
  policyTypes:  
    - Egress
```

Role Based Access Control

Setting up accounts and user roles

- Using **RBAC**, we can set up roles and accounts with **limited permissions** to the cluster
- We defined a **PodReader** role to allow maintainers to check the cluster pods' status
- This role has then been bound to two **Google accounts** that can manage the cluster

```
~/Vrije/Lab/Containerization/todo-app ➔ ↵ main • ?  
$ ➔ kubectl auth can-i delete pods -n todo-app --as sim.montali@gmail.com  
no  
↳  
~/Vrije/Lab/Containerization/todo-app ➔ ↵ main • ?  
$ ➔ kubectl auth can-i list pods -n todo-app --as sim.montali@gmail.com  
yes
```

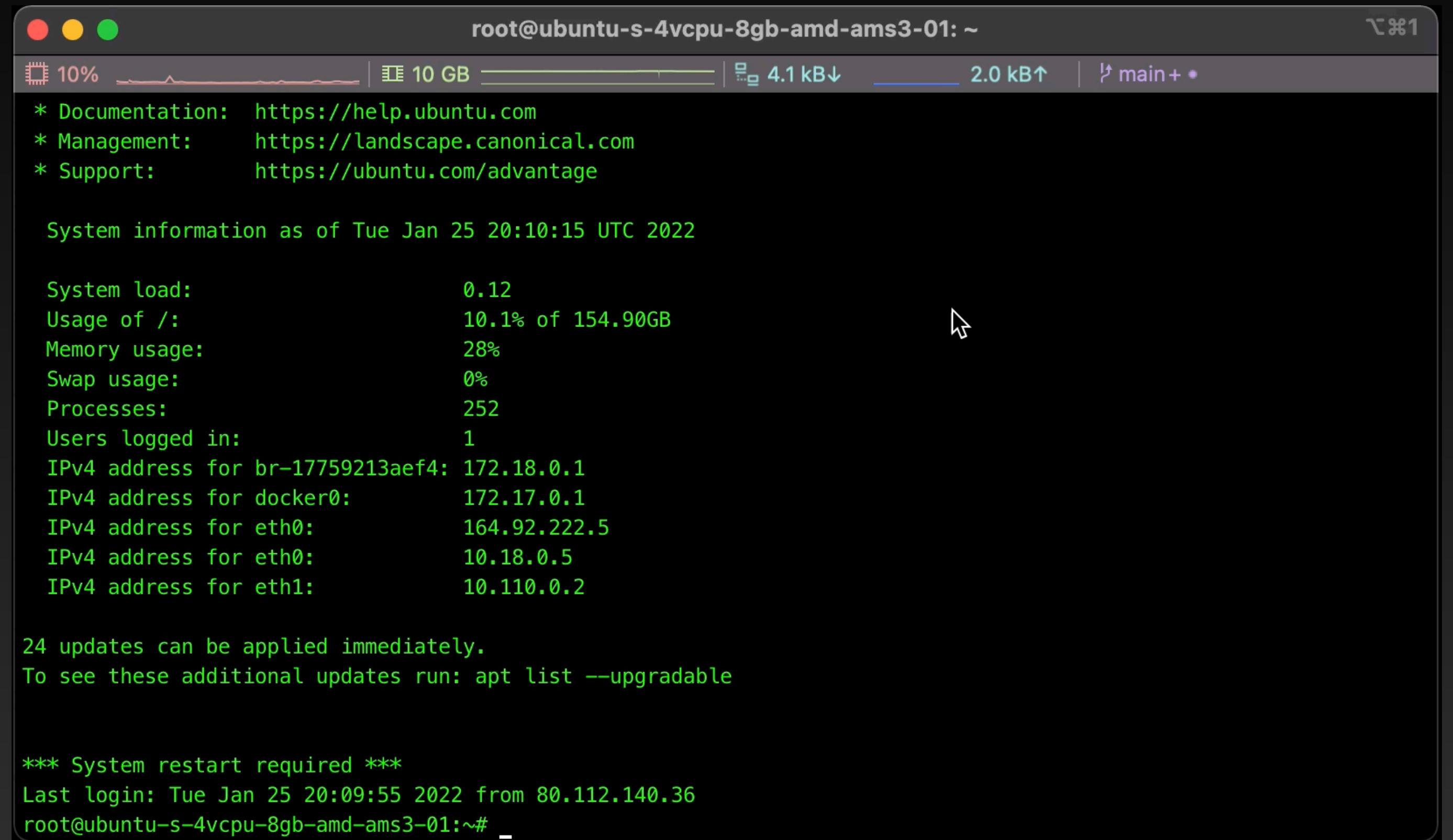


```
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
  name: admin-pods-reader  
  namespace: todo-app  
subjects:  
- kind: User  
  name: b.bozorgi92@gmail.com  
  namespace: todo-app  
- kind: User  
  name: sim.montali@gmail.com  
  namespace: todo-app  
roleRef:  
  kind: Role  
  name: pod-reader  
  apiGroup: rbac.authorization.k8s.io
```

Container build and first deployment, scaling, uninstallation

Building the images

Generating the Docker images



A screenshot of a terminal window titled "root@ubuntu-s-4vcpu-8gb-amd-ams3-01: ~". The window shows system status at the top, including battery level (10%), disk usage (10 GB), and network activity (4.1 kB↓, 2.0 kB↑). Below this, it displays documentation links for Ubuntu, management, and support. It then provides a detailed system information report as of Tuesday, January 25, 2022, 20:10:15 UTC 2022. This includes metrics like system load (0.12), memory usage (28%), swap usage (0%), and a list of processes (252). It also lists network interfaces with their IPv4 addresses (e.g., br-17759213aef4: 172.18.0.1, docker0: 172.17.0.1, eth0: 164.92.222.5, eth0: 10.18.0.5, eth1: 10.110.0.2). A message indicates 24 updates can be applied immediately, and the user is prompted to run "apt list --upgradable". A warning at the bottom states "*** System restart required ***". The terminal ends with the last login information (Tuesday, Jan 25 20:09:55 2022 from 80.112.140.36) and the root prompt.

```
root@ubuntu-s-4vcpu-8gb-amd-ams3-01: ~
[10%] 10 GB 4.1 kB↓ 2.0 kB↑ main+ *
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

System information as of Tue Jan 25 20:10:15 UTC 2022

System load:          0.12
Usage of /:           10.1% of 154.90GB
Memory usage:         28%
Swap usage:           0%
Processes:            252
Users logged in:     1
IPv4 address for br-17759213aef4: 172.18.0.1
IPv4 address for docker0: 172.17.0.1
IPv4 address for eth0: 164.92.222.5
IPv4 address for eth0: 10.18.0.5
IPv4 address for eth1: 10.110.0.2

24 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

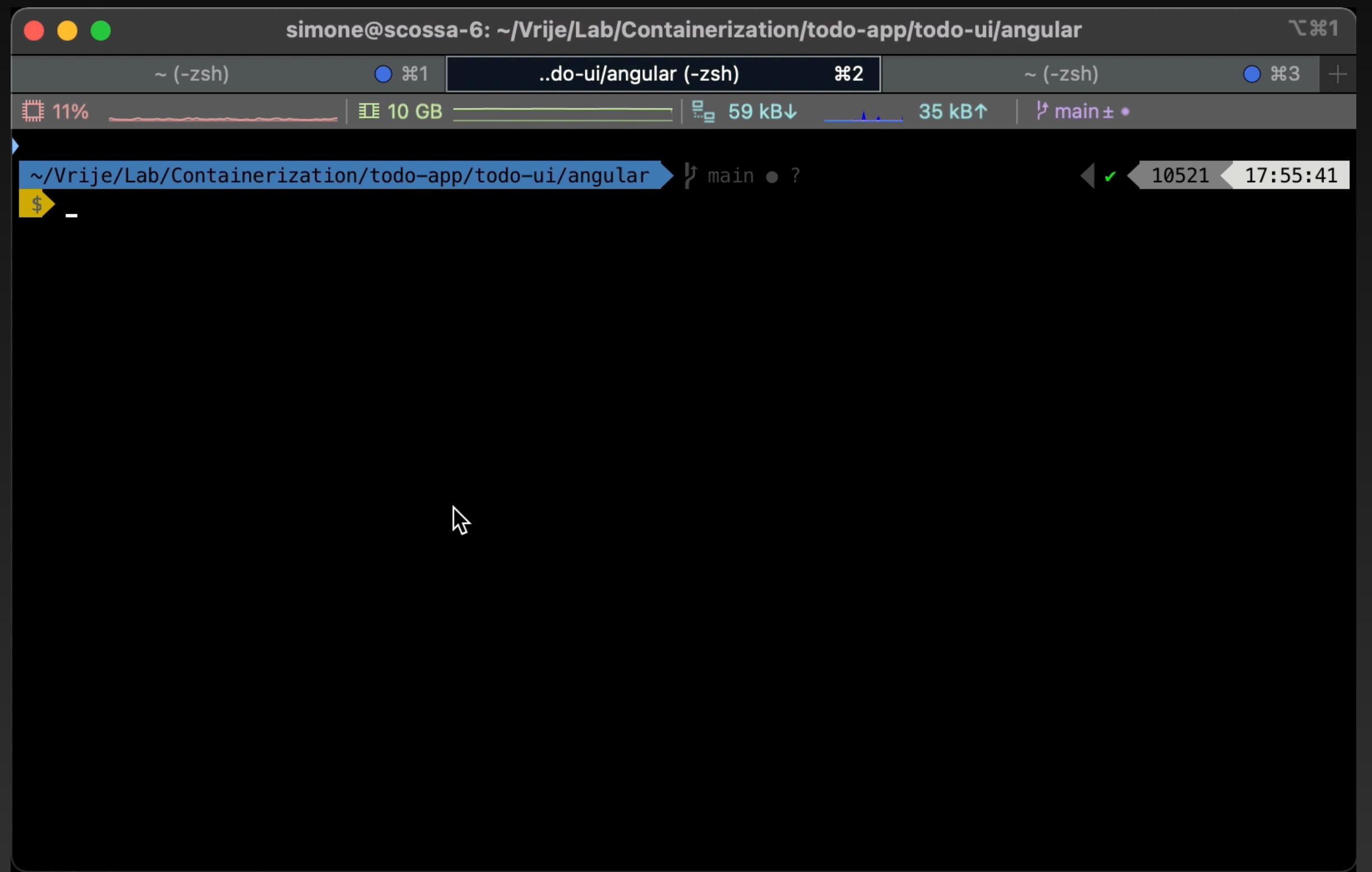
*** System restart required ***
Last login: Tue Jan 25 20:09:55 2022 from 80.112.140.36
root@ubuntu-s-4vcpu-8gb-amd-ams3-01:~#
```

First, clone the project git repository
git clone git@github.com:ThreeLittleBirdsContainerized/todo-app.git

Building the images

Generating the Docker images

After creating a Google Kubernetes Engine cluster and activating the APIs we need,



Build the Docker images and push them to GCP

```
gcloud builds submit --tag gcr.io/todo-app-339413/todo-db:v1.0
```

Installing through HELM

Starting up the cluster

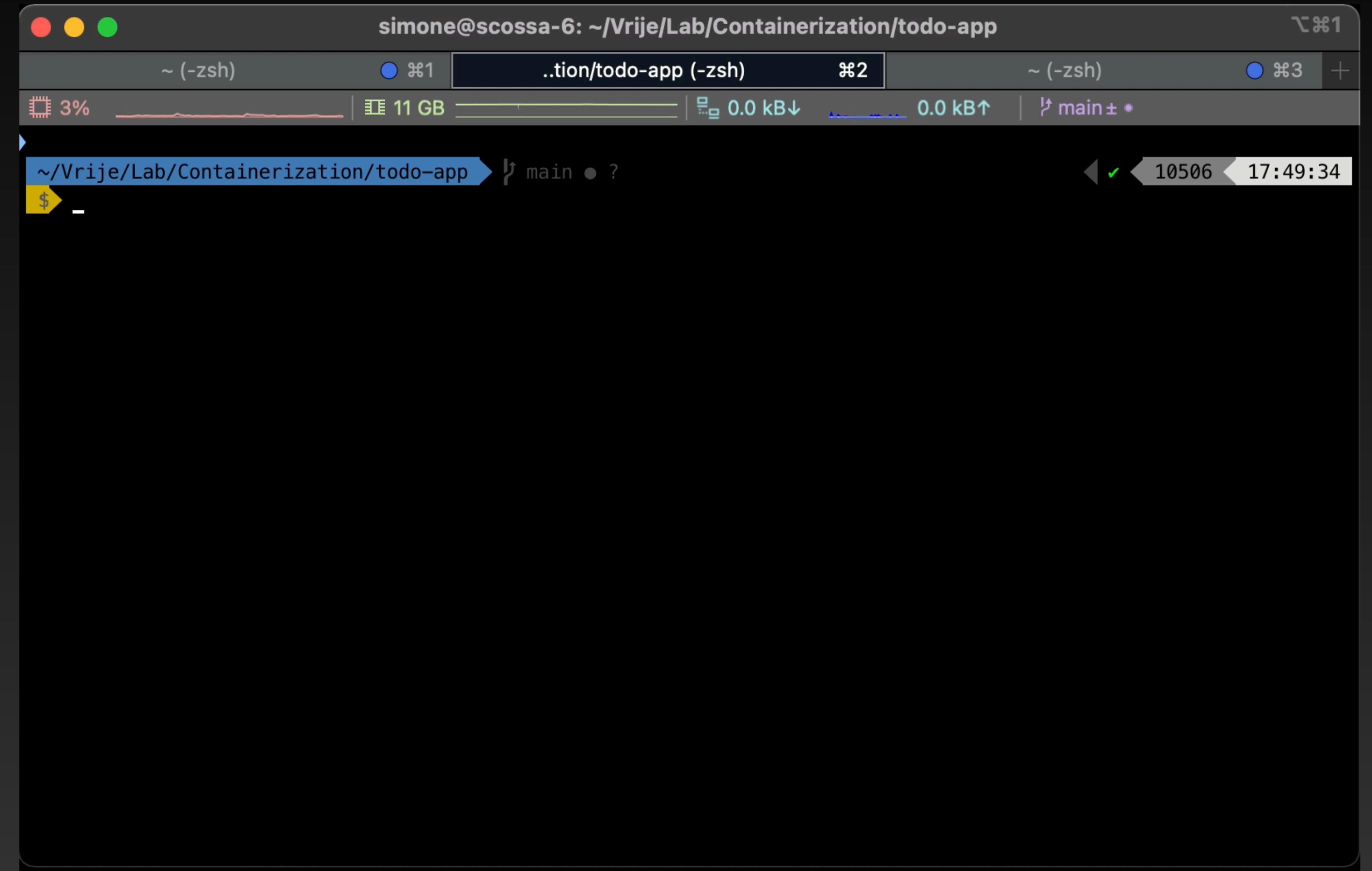
The screenshot shows a terminal window with three tabs. The active tab displays the output of a command that is removing various Homebrew packages from the user's library. The packages listed include leptonica, jpeg-xl, python@3.10, six, libomp, unbound, poppler, sdl2, nspr, libass, imagemagick, guile, p11-kit, libx11, ruby, weechat, libffi, python@3.9, gnutls, and gnupg. After the removal process, it indicates that 2 symbolic links and 14 directories were pruned from /usr/local. Below the terminal, the command \$ cd .. is shown, indicating the user is navigating back to the parent directory (~/Vrije/Lab/Containerization/todo-app). The terminal interface includes a status bar at the bottom showing file paths (~/Vrije/Lab/Containerization/todo-app/helm and ~Vrije/Lab/Containerization/todo-app), a main menu icon, and a timestamp (17:53:26).

Install everything through HELM

```
helm install helm/todo-app-chart
```

Securing the HTTP requests

TLS certificates are added from CloudFlare for a 2-points encryption



```
simone@scossa-6: ~/Vrije/Lab/Containerization/todo-app
~ (-zsh)   ☈1 ..ion/todo-app (-zsh) ☈2 ~ (-zsh)   ☈3 +
3% 11 GB 0.0 kB↓ 0.0 kB↑ ↻ main ± *
~/Vrije/Lab/Containerization/todo-app ↻ main • ?
$ -
```

Add the TLS origin certificates from CloudFlare

```
kubectl create secret tls api-cloudflare-tls --cert api.ca --key api.pk -n todo-app
kubectl create secret tls ui-cloudflare-tls --cert ui.ca --key ui.pk -n todo-app
```

The service is now up and running!
Let's check the UI

Scaling up the cluster

Horizontally scaling

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: ui-scaler
  namespace: todo-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: ui-deployment
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
    target:
      type: Utilization
      averageUtilization: 50
```

```
simone@scossa-6: ~/Vrije/Lab/Containerization/todo-app/todo-ui/angular
simone@scossa-6: ~/Vrije/Lab/Containerization/todo-app/todo-ui/angular ~ 10 GB 3.1 kB↓ 4.1 kB↑ main • 10360 17:08:19
```

Deployments scale automatically, but to do it manually:

```
kubectl scale deployment/api-deployment --replicas=10 -n todo-app
```

Uninstalling

Removing our software

To uninstall the app, it's sufficient to run an uninstall through HELM

```
helm uninstall todo
```

and delete the images:

```
gcloud container images delete gcr.io/todo-app-339413/todo-api  
gcloud container images delete gcr.io/todo-app-339413/todo-ui  
gcloud container images delete gcr.io/todo-app-339413/todo-db
```

Application upgrade and re-deployment

Deployment rollout

Upgrading the project

- Editing and re-building is easy:
 - Build the images (which builds the Angular project too)

```
gcloud builds submit --tag gcr.io/todo-app-339413/todo-api:v1.2
```

- And change the image version in the deployment

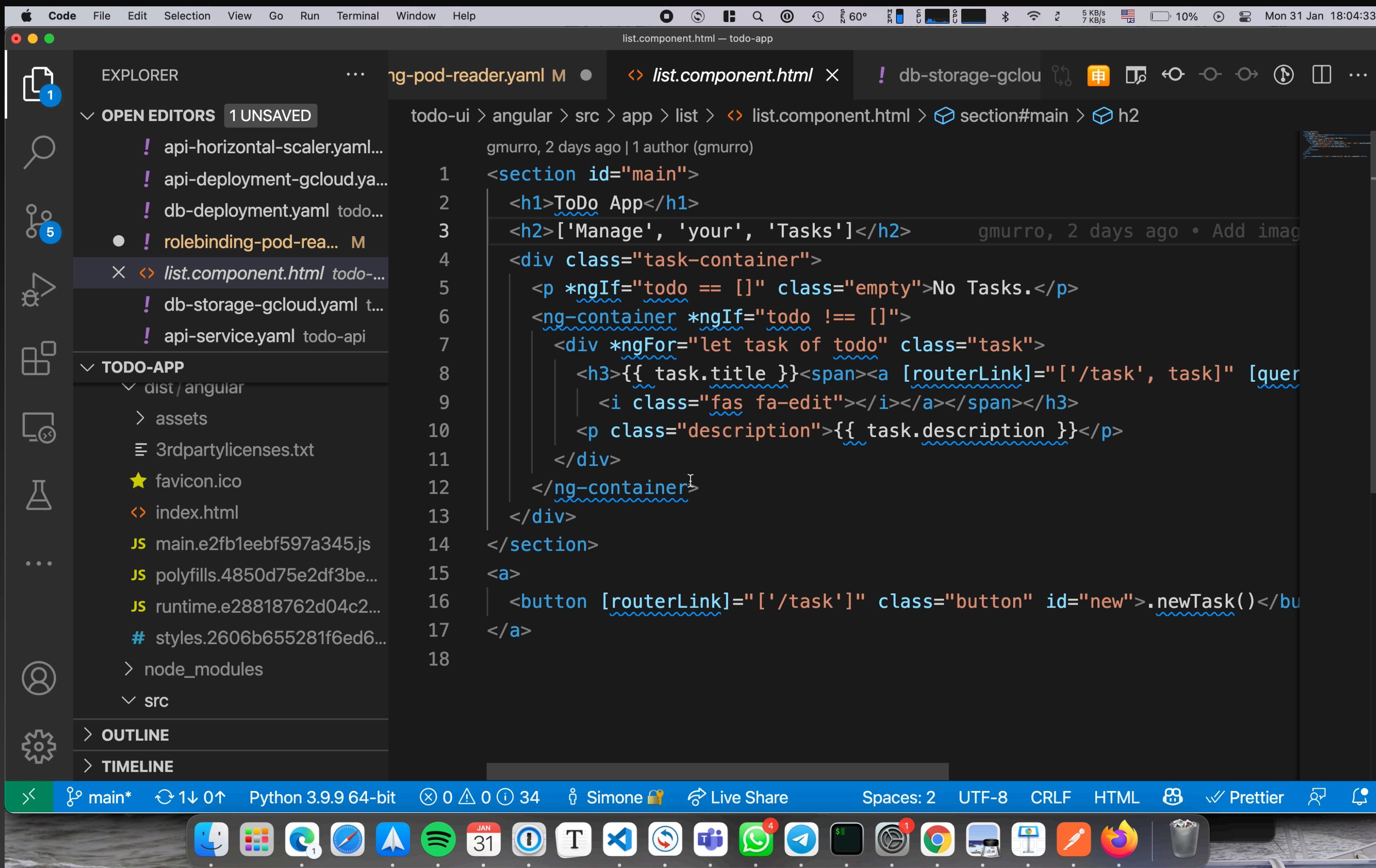
```
kubectl set image deployment/api-deployment todo-api=gcr.io/todo-app-339413/  
todo-api:v1.2 -n todo-app
```

- Or use HELM:

```
helm upgrade todo helm/todo-app-chart
```

Deployment rollout

Upgrading the project



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `ng-pod-reader.yaml`, `list.component.html`, and `db-storage-gclou`.
- Editor:** The `list.component.html` file is open, displaying Angular template code for a "ToDo App".

```
<section id="main">
  <h1>ToDo App</h1>
  <h2>['Manage', 'your', 'Tasks']</h2>
  <div class="task-container">
    <p *ngIf="todo == []" class="empty">No Tasks.</p>
    <ng-container *ngIf="todo != []">
      <div *ngFor="let task of todo" class="task">
        <h3>{{ task.title }}<span><a [routerLink]="/task", task>[quer</a><i class="fas fa-edit"></i></a></span></h3>
        <p class="description">{{ task.description }}</p>
      </div>
    </ng-container>
  </div>
</section>
<a>
  <button [routerLink]="/task" class="button" id="new">.newTask()</bu
</a>
```
- Bottom Bar:** Shows various icons for system and application management.

Canary rollout

Upgrading the project

- To perform a canary rollout, we'll serve the new version just to a % of users
 - Build the new image (which builds the Angular project too)

```
gcloud builds submit --tag gcr.io/todo-app-339413/todo-api:v1.3
```

- Create a new deployment with less Replicas and the new image

```
kubectl apply -f new-api-deployment.yml
```

- Downscale the old deployment

```
kubectl scale --replicas=9 deploy api-deployment
```

- Link both deployments to the Service (LoadBalancer)

- Re-apply the service

```
kubectl apply -f api-service.yml
```

GitHub Actions

Automating the process

- GitHub Actions allow us to automatically deploy the updates at each push to the repository
- Defining a YAML configuration that **builds** the Angular code, the **container image** and deploys to GKE
- It's required to get an **access token from GKE** and save it into the GitHub secrets

```
name: Build and Deploy UI to GKE

on:
  push:
    branches:
      - main

env:
  PROJECT_ID: ${{ secrets.GKE_PROJECT }}
  GKE_CLUSTER: todo-cluster    # Add your cluster name here.
  GKE_ZONE: europe-west4-b    # Add your cluster zone here.
  DEPLOYMENT_NAME: ui-deployment # Add your deployment name here.
  IMAGE: todo-ui

jobs:
  setup-build-publish-deploy:
    name: Setup, Build, Publish, and Deploy UI
    runs-on: ubuntu-latest
    environment: production

    steps:
      - name: Checkout
        uses: actions/checkout@v2

      # Setup gcloud CLI
      - uses: google-github-actions/setup-gcloud@94337306dda8180d967a56932ceb4ddcf01edae7
        with:
          service_account_key: ${{ secrets.GKE_SA_KEY }}
          project_id: ${{ secrets.GKE_PROJECT }}

      # Configure Docker to use the gcloud command-line tool as a credential
      # helper for authentication
      - run: |-
          gcloud --quiet auth configure-docker

      # Get the GKE credentials so we can deploy to the cluster
      - uses: google-github-actions/get-gke-credentials@fb08709ba27618c31c09e014e1d8364b02e5042e
        with:
          cluster_name: ${{ env.GKE_CLUSTER }}
          location: ${{ env.GKE_ZONE }}
          credentials: ${{ secrets.GKE_SA_KEY }}

      # Build the Angular project
      - name: Build Angular
        run: |-
          ng build

      # Build the Docker image
      - name: Build
        run: |-
          docker build \
            --tag "gcr.io/$PROJECT_ID/$IMAGE:$GITHUB_SHA" \
            --build-arg GITHUB_SHA="$GITHUB_SHA" \
            --build-arg GITHUB_REF="$GITHUB_REF" \
            .

      # Push the Docker image to Google Container Registry
      - name: Publish
        run: |-
          docker push "gcr.io/$PROJECT_ID/$IMAGE:$GITHUB_SHA"

      # Deploy the Docker image to the GKE cluster
      - name: Deploy
        run: |-
          kubectl apply -f -
          kubectl rollout status deployment/$DEPLOYMENT_NAME
          kubectl get services -o wide
```

Some numbers

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

92

Commits

27

YML artifacts

1

VPS

3100

HTTP(S) requests

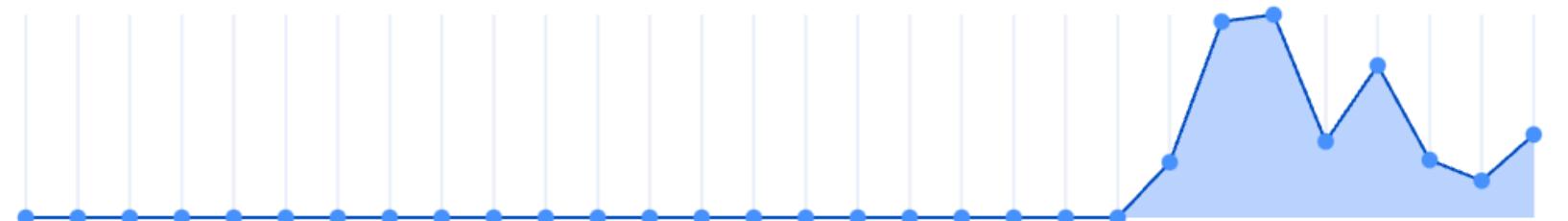
Overview

24 Hours 7 Days 30 Days

1 JANUARY — 31 JANUARY

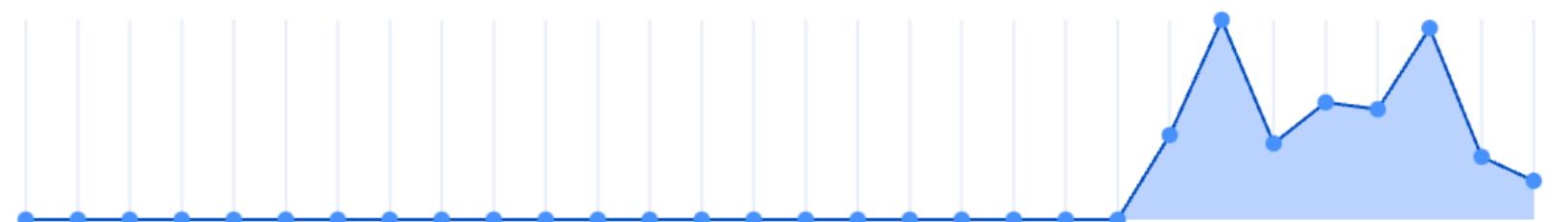
Unique Visitors

340



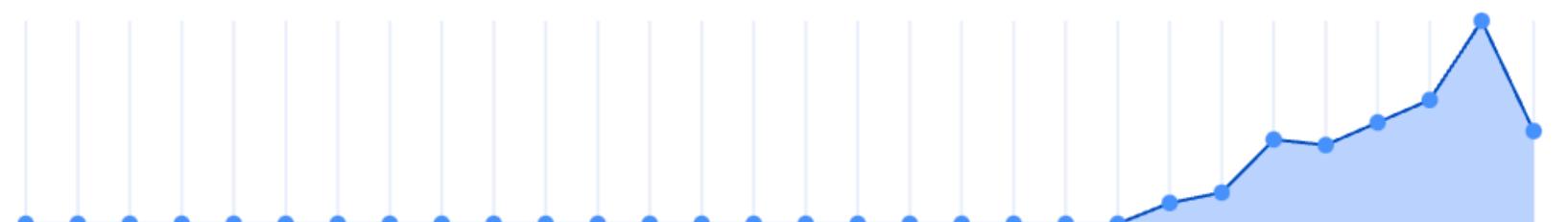
Total Requests

3.2k



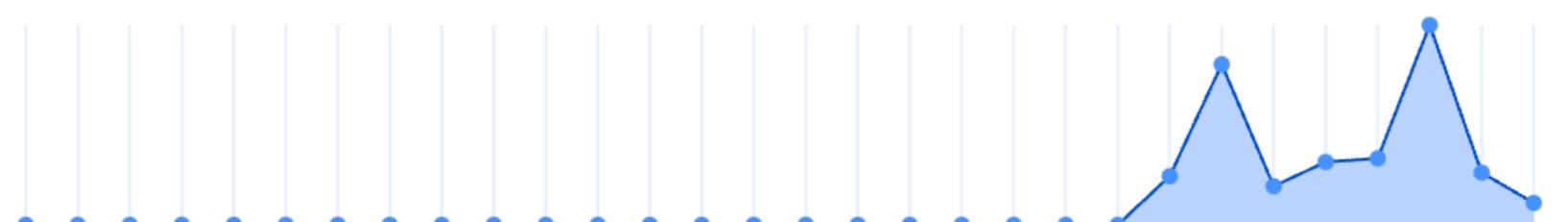
Percent Cached

29.93%



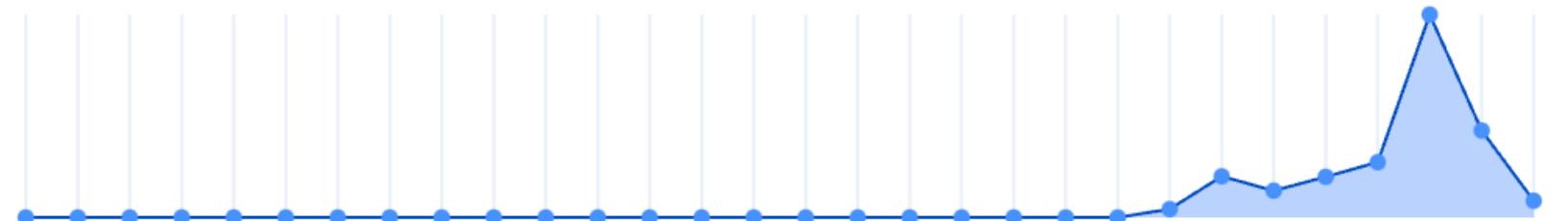
Total Data Served

22 MB



Data Cached

6 MB





Thanks!