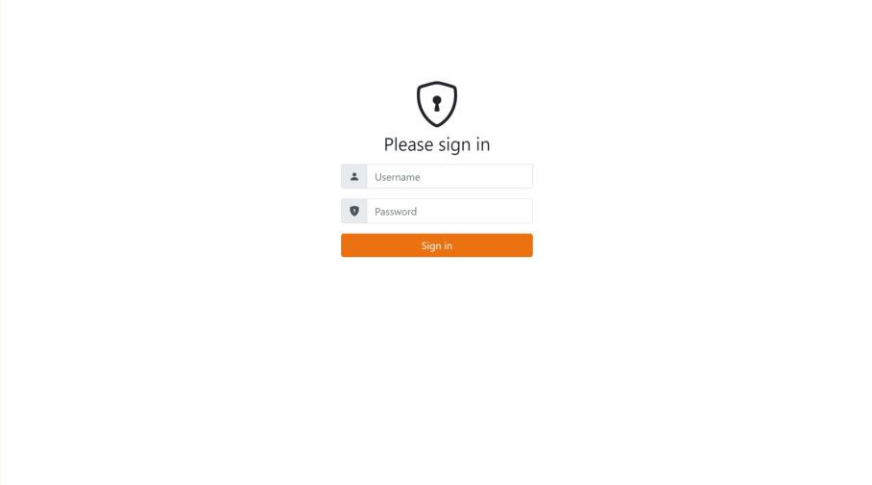
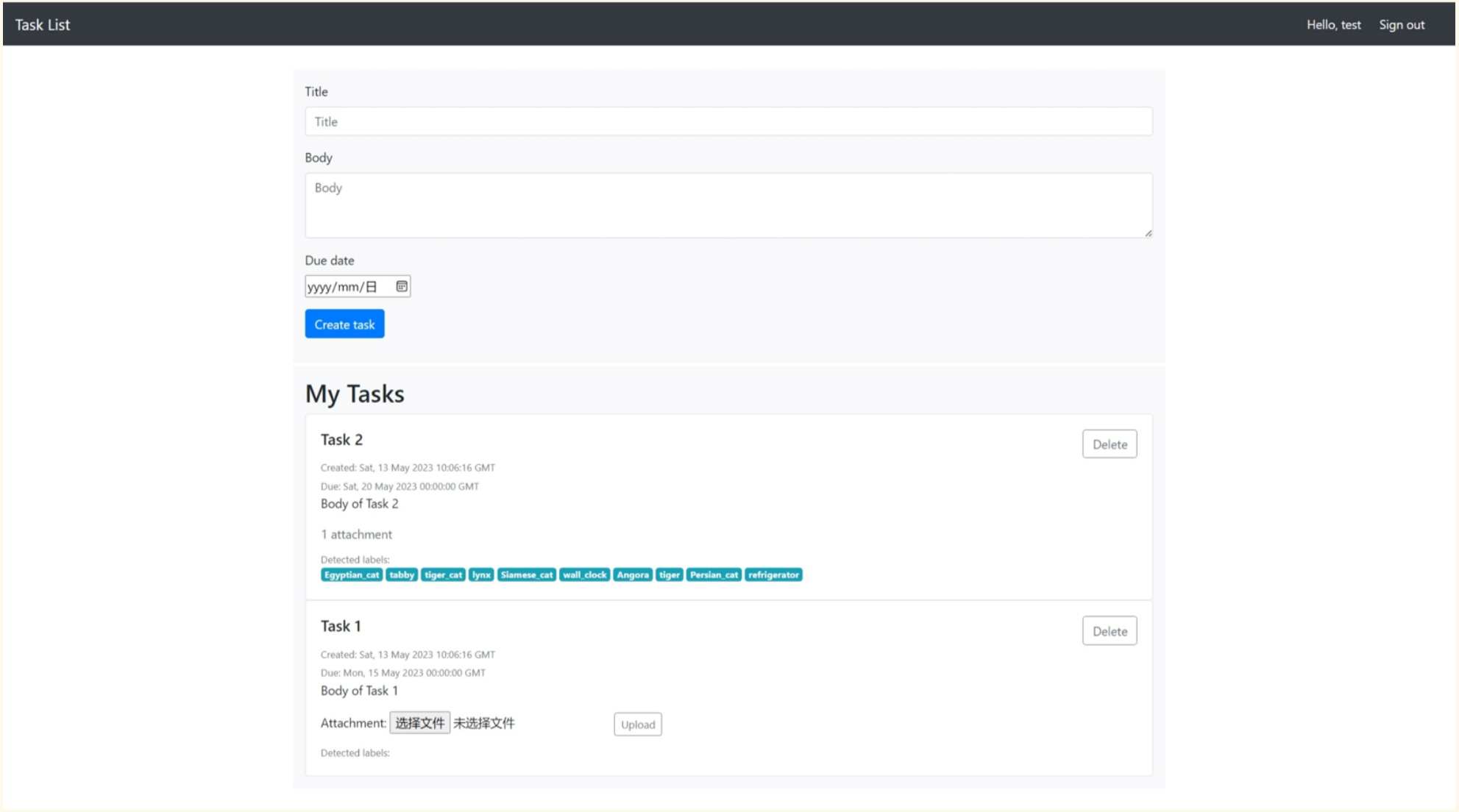
Serverless Web Application: To-Do List

01. Introduction

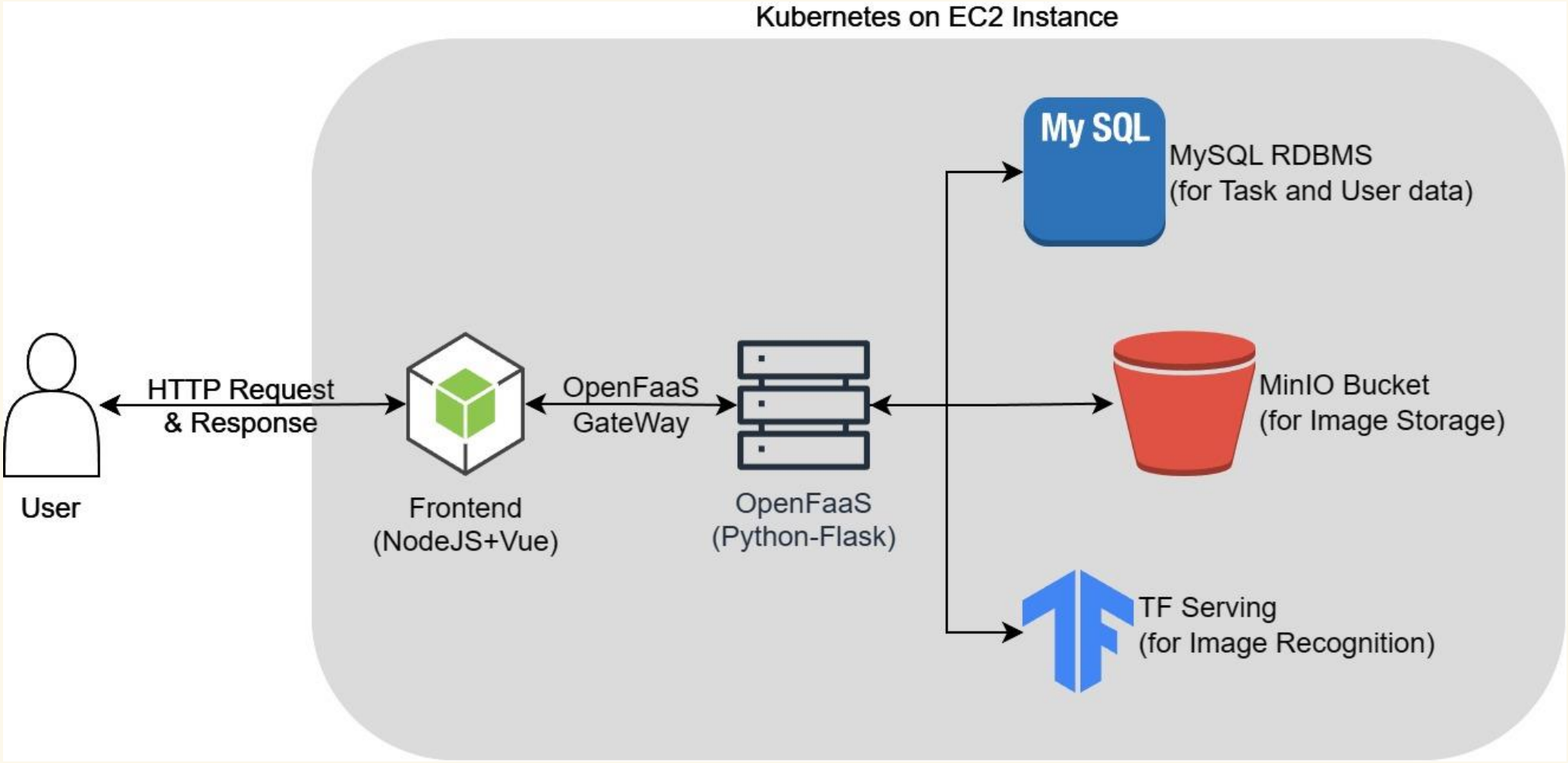
- ➔ Reimplemented an AWS serverless web application, "To-do List".
- ➔ Support features including user authentication, task management, image uploading and recognition.
- ➔ Replace AWS Lambda with OpenFaaS, DynamoDB with MySQL, S3 with MinIO and Recognition with TensorFlow Serving.
- ➔ Deployable onto a Kubernetes cluster as a set of Docker images.



01. Introduction



02. Software Design



03. Serverless Backend

```
import jwt
from flask import Flask, request
from flask_cors import CORS
from minio.deleteobjects import DeleteObject

from common import *
from img_service import get_labels

APP = Flask(__name__)
CORS(APP)
```

03. Serverless Backend

```yaml
# Information about OpenFaas and Gateway.
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:8080
# Name of the function, which we created earlier.
functions:
  flask-service:
# Specify the template, which we used while creating the function.
    lang: dockerfile
# Specify the folder (not the file) where our function code is to be found.
    handler: ./flask-service
# Specify the Docker image name of the function, which will be built with its appropriate prefix.
    image: zhanghx0905/backend-service
# Environment
    environment:
      MINIO_HOST: minio-service.default.svc.cluster.local
      MYSQL_HOST: mysql-service.default.svc.cluster.local
      TF_HOST: tensorflow-serving-service.default.svc.cluster.local
```

03. Serverless Backend

```
# Specify the of-watchdog:0.9.11 as a base image.
# The of-watchdog implements an HTTP server listening on port 8080, and acts as a reverse proxy for running functions and microservices.
# It can be used to forward the request to our application.
FROM ghcr.io/openfaas/of-watchdog:0.9.11 as watchdog

# specifies the image of our flask application
FROM python:3.10-slim-buster

# Install watchdog from the base image.
COPY --from=watchdog /fwatchdog /usr/bin/fwatchdog
RUN chmod +x /usr/bin/fwatchdog

# Uncomment if you want to use native modules
#RUN apt-get -qy update && apt-get -qy install gcc make

# Add non root user
RUN addgroup --system app && adduser app --system --ingroup app
RUN chown app /home/app

USER app
ENV PATH=$PATH:/home/app/.local/bin

WORKDIR /home/app/

COPY . .
```

03. Serverless Backend

Reimplementing the Backend with OpenFaas and Flask

- ➔ Re-implement all backend functionalities using the Flask framework
- ➔ Packaged the backend service into a Docker container and deployed it through OpenFaas via configuration
- ➔ Utilize Arkade, an open-source CLI to deploy OpenFaas.

```
# install openfaas
arkade install openfaas
```

```
faas-cli build -f faas-service.yml
faas-cli deploy -f faas-service.yml
```

03. Serverless Backend: API

- **login**
  - extracts the username and password
  - generates an authentication token
  - updates the token and commits the changes
- **create_task**
  - inserts a new task into a table named "Task"
  - commits the changes of database
  - returns a message if successful
- **delete_task**
  - retrieves the upload value
  - removes corresponding objects from a bucket in a cloud storage service
  - returns a message if successful
- **get_tasks**
  - retrieves all tasks from the database
  - returns them as a list of dictionaries
- **predict_images**
  - reads, processes and upload the image data
  - calls *get_labels()* to generate labels for the uploaded image
  - updates the labels of tasks

04. Other Components

Frontend

- Developed with Vue and JavaScript, largely inherited from original AWS project.
- Necessary adjustments in response to the new backend.
  - Backend API base URL
  - Upload picture to backend instead of S3

MySQL

To replace AWS DynamoDB

Table 1: Task Table

Column	Data Type	Constraints
id	INT	PRIMARY KEY AUTO_INCREMENT
title	VARCHAR(255)	NOT NULL
body	TEXT	NOT NULL
dueDate	DATE	NOT NULL
createdAt	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
upload	TINYINT(1)	DEFAULT 0
labels	VARCHAR(255)	DEFAULT ""

Table 2: User Table

Column	Data Type	Constraints
username	VARCHAR(255)	PRIMARY KEY
password	VARCHAR(255)	NOT NULL
token	VARCHAR(255)	

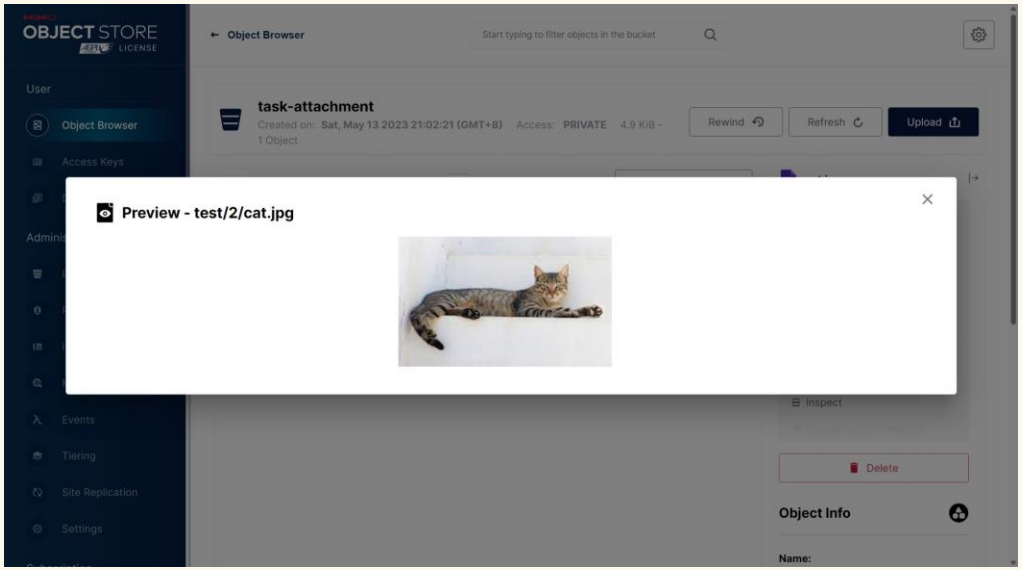
04. Other Components

TensorFlow Serving

- Image Recognition with EfficientNet V2 model,
  predicting labels on 1,000 classes within ImageNet dataset.
- Communication with backend via TF Serving RESTful API.

MinIO

- Open-source object storage system,
  full compatibility with S3.
- Provide a data management UI.



05.Deployment

Frontend and Gateway

- **Web Frontend**
- Container Construction:
  - node:14 [builder, for dependencies]
  - node:14-alpine [production]
- Connect to OpenFaaS:
  - Encapsulate OpenFaaS gateway IP
- Service Config:
  - Expose to Internet
    - Kubernetes Ingress (X)
    - ClusterIP + port-forward

- **OpenFaaS Gateway**
- Installation:
  - Arkade install
    - Checking dependencies
    - Pulling core service images
    - Creating resources
- Container Construction:
  - Images: fwatchdog + python-10.3
- Service Config:
  - Connect with backend
    - faas-cli with service names
  - Expose to Internet
    - Forwarded to port 31112

05.Deployment

Backend Services

MySQL

- VolumeMounts
  - sql initialization file mounted to /Docker-entrypoint-initdb.d

MinIO

- PersistentVolumeClaim (PVC)
  - "Read-WriteOnce", mounted to container
- Server Access
  - Port 9000 for http access: ClusterIP

TF Serving

- EfficientNet Mounting
- Server Access
  - Port 8501 for http access: ClusterIP

OpenFaaS Connection

- Host Service Name
  - connect to the services within the same Kubernetes cluster using service names

```
# Environment
    environment:
      MINIO_HOST: minio-service.default.svc.cluster.local
      MYSQL_HOST: mysql-service.default.svc.cluster.local
      TF_HOST: tensorflow-serving-service.default.svc.cluster.local
```

Thanks

Serverless Web Application: To-Do List

Group 6