

Group 9

Group Members:

Name
Sherry Ahuja
Gagan Deep Singh
Gurjot Singh
Mansimar Singh Bhatia
Jasleen Bedi

Western Engineering

ECE 9016 Cloud Computing

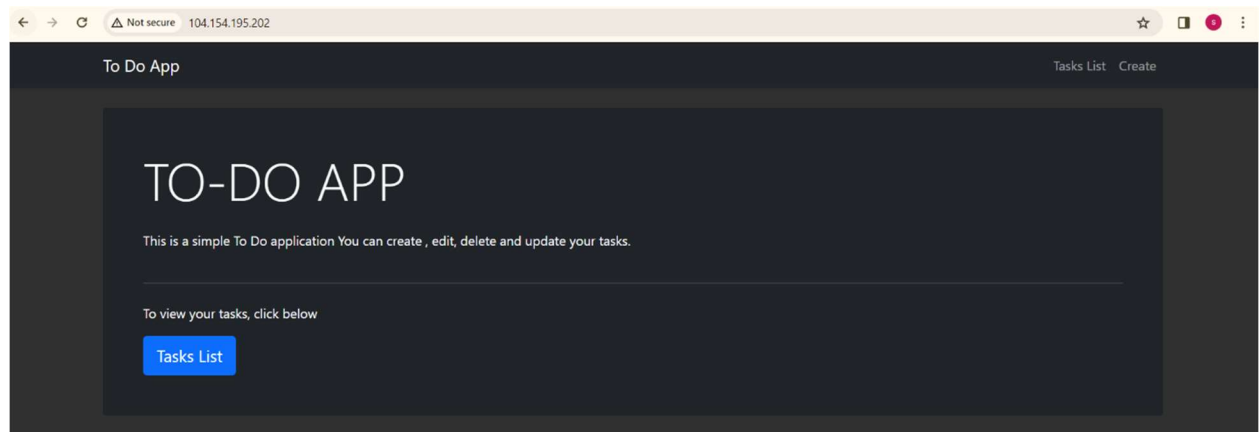
Project Report

1. Introduction

In the realm of cloud computing, practical experience with state-of-the-art technologies and methodologies is crucial for nurturing the next generation of engineers. The ECE 9016 Cloud Computing Project is crafted to offer a deep dive into deploying and managing web applications in the cloud, with a strong emphasis on Kubernetes, an open-source system for automating deployment, scaling, and management of containerized applications. This project bridges the gap between theoretical knowledge and real-world application deployment of a simple, yet insightful, **to-do list application** on Google Cloud Platform (GCP) using Kubernetes, thereby illuminating the intricacies of cloud infrastructure, high availability, and load balancing.

The project's backbone is a **CI/CD pipeline**, which closely mirrors the collaborative and automated workflows prevalent in the software industry today. Our application—a straightforward **To-Do App**—serves as an exemplary case study for understanding the deployment complexities and scalability afforded by cloud technologies. This app allows users to manage their tasks with basic functionalities such as create, read, update, and delete operations, making it an ideal candidate for demonstrating Kubernetes' capabilities in orchestrating containers in diverse environments.

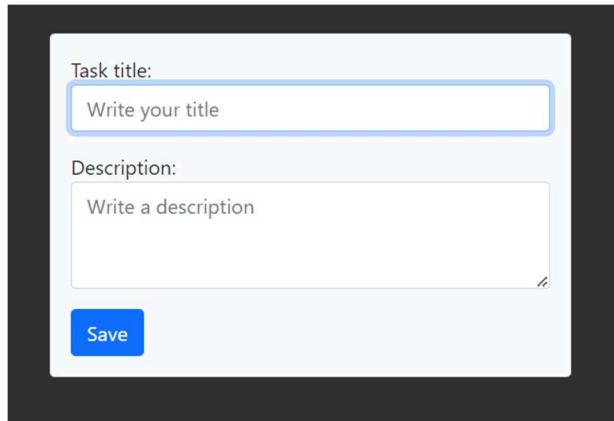
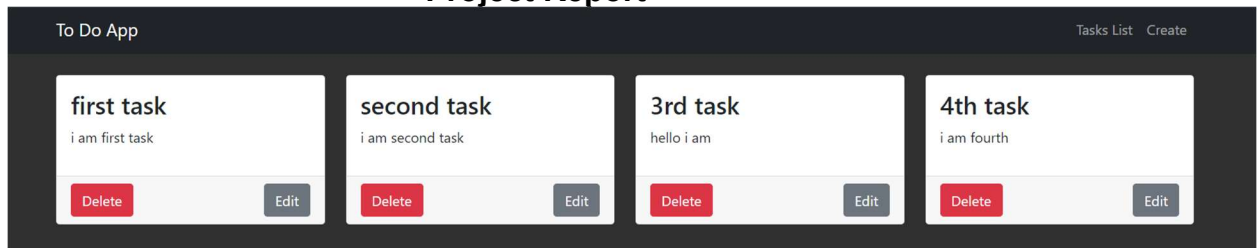
Kubernetes plays a pivotal role in this project, facilitating the orchestration of our to-do application's containers across development and production environments. This orchestration is supported by an array of tools and practices essential to modern software development, including GitHub Repositories for collaborative coding, Docker for containerization, Artifact Registry for managing container images, and Cloud Build, alongside Triggers, for automating the build and deploy processes. These elements collectively enhance the development pipeline, enabling more efficient collaboration, and the swift, reliable release of software updates.



Western Engineering

ECE 9016 Cloud Computing

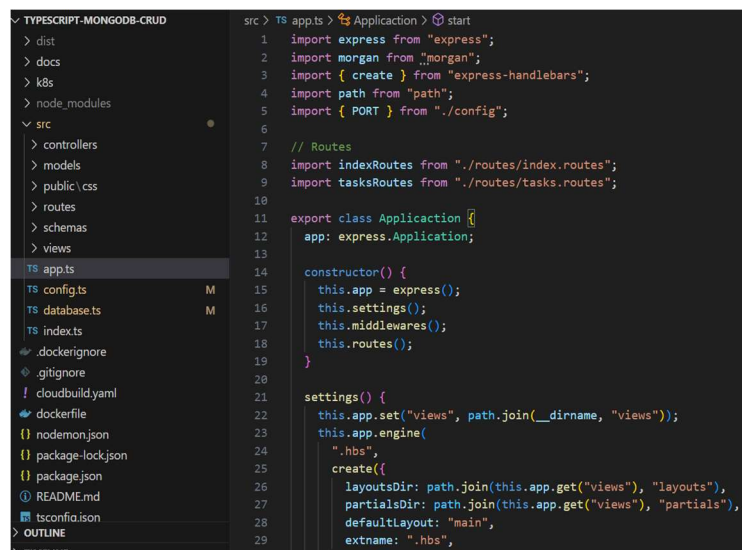
Project Report



Live website url - <http://104.154.195.202/>

2. System architecture and design.

1. Source code



Western Engineering

ECE 9016 Cloud Computing

Project Report

```
src > TS database.ts > ...
1  import { connect, connection } from "mongoose";
2  import { MONGODB_URI } from "../config";
3
4  export async function connectToMongodb() {
5    try {
6      await connect("mongodb://sherry:sherry@mongo-service-prod/tododb");
7    } catch (error) {
8      console.log("Error:", error);
9    }
10 }
11
12 connection.on("connected", () => {
13   console.log("Mongodb connected to:", connection.db.databaseName);
14 });
15
16 connection.on("error", (error) => {
17   console.error("error", error);
18 });
19
20 connection.on("disconnected", () => {
21   console.log("Mongodb disconnected");
22 });
23
```

In the app.ts file from the src folder, we define the Express application setup. It involves importing required modules, configuring the view engine with express-handlebars, and initializing middleware and routes for the application.

The database.ts file, also within the src directory, sets up the MongoDB connection using Mongoose. It exports an asynchronous function to connect to the database and handles connection events such as connected, error, and disconnected, logging the status to the console. This modular approach allows for a clean separation between the web server and the database logic.

2. Git Repository

sahuja49 update readme		a902143 · last week	7 Commits
docs	initial commit	last week	
k8s	k8s deployment and service and cloudbuild added	last week	
src	initial commit	last week	
.dockerignore	Create .dockerignore	last week	
.gitignore	initial commit	last week	
README.md	update readme	last week	
cloudbuild.yaml	update cloudbuild	last week	
dockerfile	initial commit	last week	
nodemon.json	initial commit	last week	
package-lock.json	initial commit	last week	
package.json	initial commit	last week	
tsconfig.json	initial commit	last week	

Western Engineering

ECE 9016 Cloud Computing

Project Report

This GitHub repository serves as the codebase for our project, encompassing all the necessary files and configurations for both development and production deployments. It is structured into several directories and files, each with a specific role in the application lifecycle:

- **docs:** Holds documentation related to the project.
- **k8s:** Contains Kubernetes manifests for orchestrating container deployment, services, and workload management within the cluster.
- **src:** The source code directory for the application, containing all the TypeScript files.
- **.dockerignore:** Lists files and directories to be ignored during Docker builds, optimizing build context.
- **.gitignore:** Specifies intentionally untracked files to be ignored by Git, such as dependencies or build outputs.
- **README.md:** Provides a descriptive overview of the project, setup instructions, and other pertinent information.
- **cloudbuild.yaml:** The configuration file for Google Cloud Build, automating the build, test, and deploy processes.
- **Dockerfile:** Contains the instructions to assemble the application's Docker image.
- **nodemon.json:** Configuration file for Nodemon, a utility that monitors for any changes in the source and automatically restarts the server.
- **package-lock.json:** Auto-generated file ensuring consistent installations and dependency resolution.
- **package.json:** Defines the project's dependencies, scripts, and versioning information.
- **tsconfig.json:** The TypeScript compiler configuration file, specifying how the TypeScript code is compiled to JavaScript.

The organization of this repository, with separate concerns neatly categorized, reflects our commitment to best practices in version control and CI/CD processes. It enables us to manage our application development flow efficiently, from writing code to deploying it into production environments.

3. Docker File

```
# Use latest Node.js version as the base image
FROM node:latest AS builder

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Build TypeScript code
RUN npm run build

# Start a new stage with a minimal Node.js image
FROM node:slim

# Set the working directory in the container
WORKDIR /app

# Copy only necessary files from the previous stage
COPY --from=builder /app/package*.json ./
COPY --from=builder /app/dist ./dist/

# Install only production dependencies
RUN npm install --only=production

# Expose the port your app runs on
EXPOSE 3000

# Command to run your application
CMD ["npm", "run", "start"]
```

Western Engineering

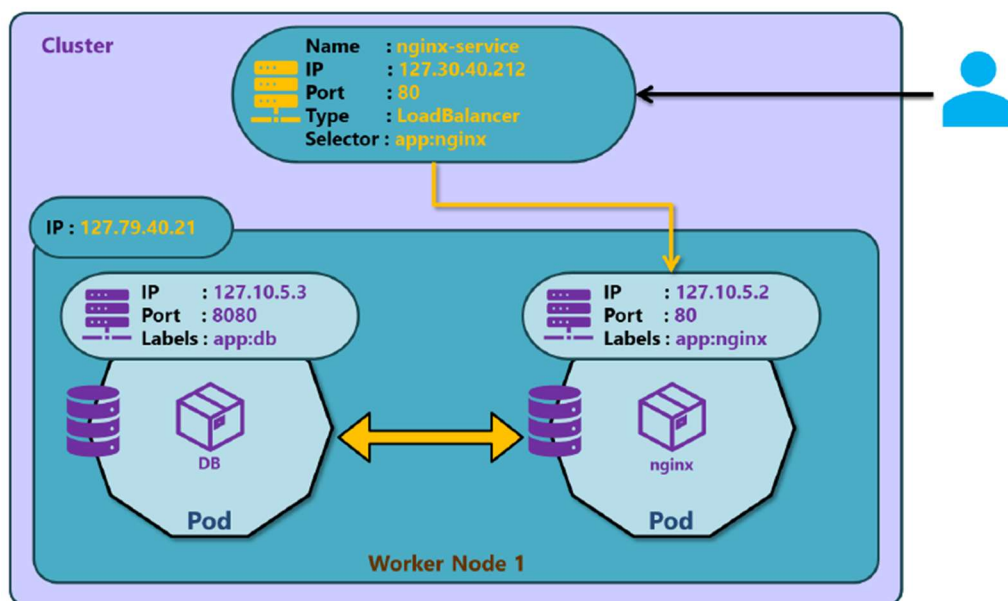
ECE 9016 Cloud Computing

Project Report

- **Multi-stage Build:**
 - Divides the build process into two stages: **builder** for building the TypeScript code and **node:slim** for the final runtime environment.
- **Builder Stage:**
 - Uses **node:latest** as the base image.
 - Sets the working directory to **/app**.
 - Copies **package.json** and **package-lock.json**, installs dependencies, and copies application code.
 - Builds TypeScript code using **npm run build**.
- **Node Slim Stage:**
 - Starts with **node:slim** as the base image.
 - Sets the working directory to **/app**.
 - Copies only necessary files from the builder stage (**package.json**, **dist/** directory).
 - Installs production dependencies only.
 - Exposes port **3000** for external access.
 - Defines the command to run the application: **npm run start**.

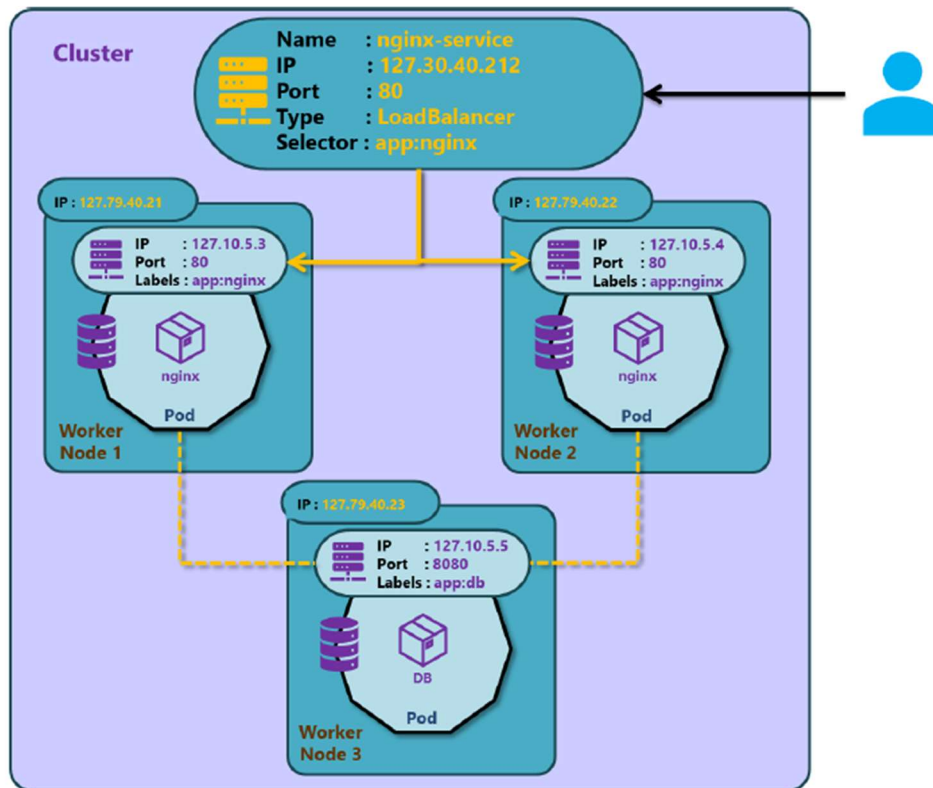
4. Development environment

It contains single node comprising two pods: The first Pod contains a web server that contains a to-do application that reads from mongo db database. The second Pod contains the database where the information is stored.



5. Production environment

1. **First Worker Node** have a single Pod containing a web server of your choice with your application deployed.
2. **Second Worker Node** similar to the first node, where it contains a redundant web server with the same application deployed.
3. **Third Worker Node:** contains a database of your choice where the deployed application in the first and second node can connect to the database to retrieve information.
4. **Load Balancer:** add a load balancer between the first and second node/pods to distribute the traffic and allow external access to the application.
5. **Service Configurations and Labels:** Essential for smooth operation and management within the Kubernetes environment.




Western Engineering

ECE 9016 Cloud Computing

Project Report


6. CI/CD Pipeline and Triggers


Name ↑	Description	Repository	Event	Build configuration	Status	
todo	typescript dev trigger	 typescript-app	Push to branch	cloudbuild.yaml	Disabled	RUN ⋮
typescript-app-prod	typescript prod trigger	 typescript-app	Push to branch	cloudbuild-prod.yaml	Enabled	RUN ⋮


In our Continuous Integration and Continuous Deployment (CI/CD) pipeline, we have implemented two distinct triggers to automate the development and production workflows for our to-do application. These triggers are integral to our pipeline, facilitating the automation of build and deployment processes upon code commits to our repository.




The delineation between the development and production triggers within our CI/CD pipeline is crucial. It allows for a bifurcated approach where development can proceed without impacting the production environment, while still maintaining a streamlined path to production once code is ready for release. The utilization of separate configuration files (cloudbuild.yaml for development and cloudbuild-prod.yaml for production) ensures that the appropriate steps and environments are used during the build process, safeguarding the integrity of our deployment workflow.


7. Artifact Repository

 Artifact Registry

 Repositories

 Settings


← Images for typescriptto-repository  DELETE  EDIT REPOSITORY  SETUP INSTRUCTIONS

us-central1-docker.pkg.dev > smart-processor-412022 > typescriptto-repository 



Repository Details

Format	Docker
Type	Standard

[SHOW MORE](#)

 Filter

 Enter property name or value

<input type="checkbox"/>	Name ↑	Connection	Created	Updated
<input type="checkbox"/>	 typescript-app	—	8 days ago	6 days ago
<input type="checkbox"/>	 typescript-app-prod	—	6 days ago	55 minutes ago

We have used Google Cloud Artifact Registry for the **typescript-to-repository**, highlighting the management of Docker images for our to-do application. It showcases two key images: **typescript-app for development**, and **typescript-app-prod for production**. The registry, categorized under the Docker format and standard type within the **us-central1 region**(same as

Western Engineering

ECE 9016 Cloud Computing

Project Report

cluster), facilitates robust version control and seamless updates to our application, signifying an active and well-maintained CI/CD workflow. This setup underscores the efficiency of our development practices, ensuring that the latest code changes are consistently reflected in our containerized environments.

8. Load Balancing

[← Load balancer details](#) [EDIT](#) [DELETE](#) [→ VIEW IN NETWORK TOPOLOGY](#)

balancers list page

a741186f65a2645b59bebf9db8b7d792

Target-pool network load balancer

Frontend

Protocol ↑	IP version	IP:Port	Network Tier ?
TCP	IPv4	104.154.195.202:80	Premium

Backend

Name	Region	Health check
a741186f65a2645b59bebf9db8b7d792	us-central1	k8s-9bfe12daaf644b64-node

[ADVANCED CONFIGURATIONS](#)

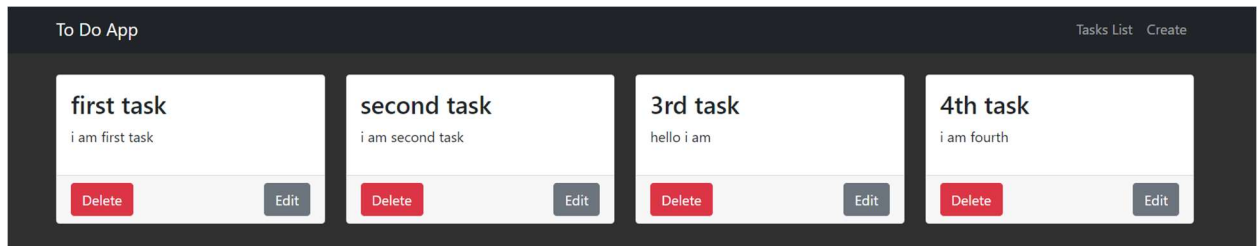
Instance ↑	Zone	104.154.195.202
gk3-todo-prod-cluster-pool-2-572b36a9-zt7q	us-central1-a	✓
gk3-todo-prod-cluster-pool-2-9fee8168-9cx4	us-central1-f	✓
gk3-todo-prod-cluster-pool-2-9fee8168-s9b4	us-central1-f	✓

The image presents the details of a production environment load balancer named **a741186f65a2645b59bebf9db8b7d792** within the **todo-prod-cluster**. The load balancer is configured to use TCP over IPv4 with an assigned IP and port combination of **104.154.195.202:80**, signifying its role as a gateway for incoming traffic at the network's edge. Positioned in the premium network tier, this setup suggests an optimized environment for performance and reliability.

In the backend section, the same alphanumeric identifier is used for the load balancer, indicating it is designated for a specific set of services, with health checks implemented, named **k8s-9bfe12daaf644b64-node**. This health check is pivotal for ensuring that the traffic is only directed to healthy instances. The listed instances under 'Advanced Configurations' show three different nodes within the **us-central1** region, across zones **us-central1-a** and **us-central1-f**, all of which have passed the health check, as shown by the check marks. This robust setup underscores the resilience and high-availability aspects of the **todo-prod-cluster**, ensuring that the production environment remains reliable and efficient for handling user requests.

3. Deployment Process (Dev and Prod)

- a) **Tested the source code in docker desktop Locally:** To Do application underwent comprehensive testing on Docker Desktop locally to confirm its functionality and ensure compatibility within a containerized setting.



- b) **GitHub Repository Preparation and GCP Integration:** A GitHub repository was established to host the application code, supporting both development and production for robust version control. Also, Cloud source repository was connected to github also to make it faster. This repository was seamlessly integrated with Google Cloud Platform services to facilitate collaborative development.

sahuja49 update readme ✖			a902143 · last week	🕒 7 Commits
docs	initial commit			last week
k8s	k8s deployment and service and cloudbuild added			last week
src	initial commit			last week
.dockerignore	Create .dockerignore			last week
.gitignore	initial commit			last week
README.md	update readme			last week
cloudbuild.yaml	update cloudbuild			last week
dockerfile	initial commit			last week
nodemon.json	initial commit			last week
package-lock.json	initial commit			last week
package.json	initial commit			last week
tsconfig.json	initial commit			last week

Western Engineering

ECE 9016 Cloud Computing

Project Report

Cloud Source Repositories

Search for code or file:

typescript-app > master

Files

Outline

Repository root

docs

k8s

src

.dockerignore

.gitignore

Dockerfile

README.md

cloudbuild-prod.yaml

cloudbuild.yaml

nodemon.json

package-lock.json

package.json

tsconfig.json

Repository Root

tynescrypt Nodejs MongoDB CRUD on

History

ID	Author	Commit date	Description
411fe5b	sahuja49	2024-04-05 18:42	modify to mysql
8392b9d	sahuja49	2024-04-05 08:48	db on separate node
a640f70	sahuja49	2024-04-03 09:00	final final
e29ebde	sahuja49	2024-04-03 08:47	final v2
26ef2eb	sahuja49	2024-04-03 08:43	final
3381044	sahuja49	2024-04-03 08:29	update cred
6f2295d	sahuja49	2024-04-02 17:01	final v0

- c) **Utilization of Artifact Registry:** Container images of the application were stored and managed in the Artifact Registry, optimizing storage and distribution within the CI/CD pipeline and bolstering deployment reliability.

Artifact Registry	Images for typescriptto-repository	DELETE	EDIT REPOSITORY	SETUP INSTRUCTIONS
Repositories	us-central1-docker.pkg.dev > smart-processor-412022 > typescriptto-repository			
Settings	Repository Details			
	Format	Docker		
	Type	Standard		
	SHOW MORE			
	Filter	Enter property name or value		
	Name	Connection	Created	Updated
	typescript-app	—	8 days ago	6 days ago
	typescript-app-prod	—	6 days ago	55 minutes ago

- d) **Automation with Cloud Build Triggers:** Deployment processes were automated with Cloud Build Triggers, which were configured to initiate builds and deployments in response to repository changes, thus enhancing the CI/CD pipeline's efficiency.

Name	Description	Repository	Event	Build configuration	Status	
todo	typescript dev trigger	typescript-app	Push to branch	cloudbuild.yaml	Disabled	RUN
typescript-app-prod	typescript prod trigger	typescript-app	Push to branch	cloudbuild-prod.yaml	Enabled	RUN

- e) **Development Environment Deployment:** Using Kubernetes manifests tailored for the development environment, the application's dev version was deployed, detailing the required state, container configurations, and services necessary for the development phase.

a) Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: typescript-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: typescript-app
  template:
    metadata:
      labels:
        app: typescript-app
    spec:
      containers:
        - name: typescript-app
          image: us-central1-docker.pkg.dev/to-do-418823/typescriptto-
repository/typescript-app:tag1
          ports:
            - containerPort: 3000
          imagePullPolicy: Always
```

b) Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: typescript-app
spec:
  selector:
    app: typescript-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

c) Mongodb-deployment.yaml

Western Engineering

ECE 9016 Cloud Computing

Project Report

```
    apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - typescript-app
              topologyKey: kubernetes.io/hostname
      containers:
        - name: mongodb
          image: mongo
          ports:
            - containerPort: 27017
          imagePullPolicy: Always
          # Additional settings for MongoDB configuration
          # Uncomment and modify the following lines as needed:

          # Volume mounts for persistent storage
          volumeMounts:
            - name: mongodb-data
              mountPath: /var/lib/mongodb

          # Environment variables for authentication
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              value: admin
            - name: MONGO_INITDB_ROOT_PASSWORD
```

```
value: admin

# PersistentVolumeClaim to persist data across pod restarts
volumes:
- name: mongodb-data
  persistentVolumeClaim:
    claimName: mongodb-data-pvc

# PersistentVolumeClaim for MongoDB data
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-data-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi # Adjust storage size as needed
```

d) Mongodb-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: typescript-app
spec:
  selector:
    app: typescript-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 3000
  type: LoadBalancer
```

e) CloudBuild.yaml

```
steps:
  # Step 1: Build the Docker image using the Dockerfile in the repository
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'us-central1-docker.pkg.dev/to-do-418823/typescriptto-repository/typescript-app:tag1', '.']

  # Step 2: Push the Docker image to Google Container Registry
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'us-central1-docker.pkg.dev/to-do-418823/typescriptto-repository/typescript-app:tag1']

  # Step 3: Set up kubectl to use the cluster's credentials
  - name: 'gcr.io/cloud-builders/gcloud'
    args: ['container', 'clusters', 'get-credentials', 'todo-cluster', '--zone', 'us-central1-a', '--project', 'to-do-418823']

  # Step 4: Apply the Kubernetes deployment manifest
  - name: 'gcr.io/cloud-builders/kubectl'
    args: ['apply', '-f', 'k8s/deployment.yaml']
    env:
      - 'CLOUDSDK_COMPUTE_REGION=us-central1-a'
      - 'CLOUDSDK_CONTAINER_CLUSTER=todo-cluster'
      - 'CLOUDSDK_CORE_PROJECT=to-do-418823'

  # Step 5: Apply the Kubernetes service manifest
  - name: 'gcr.io/cloud-builders/kubectl'
    args: ['apply', '-f', 'k8s/service.yaml']
    env:
      - 'CLOUDSDK_COMPUTE_REGION=us-central1-a'
      - 'CLOUDSDK_CONTAINER_CLUSTER=todo-cluster'
      - 'CLOUDSDK_CORE_PROJECT=to-do-418823'

  # Step 6: Apply the MongoDB Kubernetes deployment manifest
  - name: 'gcr.io/cloud-builders/kubectl'
    args: ['apply', '-f', 'k8s/mongodb-deployment.yaml']
    env:
      - 'CLOUDSDK_COMPUTE_REGION=us-central1-a'
      - 'CLOUDSDK_CONTAINER_CLUSTER=todo-cluster'
```

Western Engineering

ECE 9016 Cloud Computing

Project Report

```
- 'CLOUDSDK_CORE_PROJECT=to-do-418823'

# Step 7: Apply the MongoDB Kubernetes service manifest
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['apply', '-f', 'k8s/mongodb-service.yaml']
  env:
    - 'CLOUDSDK_COMPUTE_REGION=us-central1-a'
    - 'CLOUDSDK_CONTAINER_CLUSTER=todo-cluster'
    - 'CLOUDSDK_CORE_PROJECT=to-do-418823'

images:
  - 'us-central1-docker.pkg.dev/to-do-418823/typescriptto-
repository/typescript-app:tag1'
```

This code defines a series of steps to deploy a containerized application and its associated MongoDB database to a Kubernetes cluster using Google Cloud Build. Here's a brief explanation of each step:

1. **Build Docker Image:** This step uses the Cloud Build Docker builder to build the Docker image for the application using the Dockerfile present in the repository. The resulting image is tagged with a version (v1) and stored in the Google Container Registry (GCR).
2. **Push Docker Image:** After building the Docker image, this step pushes the image to the Google Container Registry (gcr.io) in the specified project and repository (us-central1-docker.pkg.dev/tonal-limiter-419416/contacts-dev-repo).
3. **Set up Kubernetes Credentials:** This step sets up kubectl to use the credentials of the specified Kubernetes cluster (contacts-project-cluster) located in the us-central1-a zone and associated with the project tonal-limiter-419416.
4. **Apply Deployment Manifest:** This step applies the Kubernetes deployment manifest (deployment.yaml) for the application to the cluster. It specifies the desired state for the application, such as the number of replicas and the container image to use.
5. **Apply Service Manifest:** Similarly, this step applies the Kubernetes service manifest (service.yaml) to expose the application within the cluster. The service enables communication with the application pods.
6. **Apply MongoDB Deployment Manifest:** This step applies the Kubernetes deployment manifest (mongodb-deployment.yaml) for the MongoDB database to the cluster. It defines how the MongoDB instance should be deployed, including replicas and resource requirements.
7. **Apply MongoDB Service Manifest:** Finally, this step applies the Kubernetes service manifest (mongodb-service.yaml) to expose the MongoDB service within the cluster. It allows other components to interact with the MongoDB instance.

Western Engineering

ECE 9016 Cloud Computing

Project Report

The env variables in each step are used to set the context for the Cloud SDK (gcloud) commands, specifying the region, cluster name, and project ID. Once all steps are completed successfully, the application and MongoDB database will be deployed and accessible within the Kubernetes cluster.

- f) **Production Environment Deployment:** The production version of the application was also deployed with Kubernetes manifests, fine-tuned to meet the demands of the production environment. This included scaling up replicas, refining selectors and labels for clear environment distinction, and updating container image references to ensure a robust and scalable user-facing application.

a) **deployment-prod.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: typescript-app-prod
spec:
  replicas: 2
  selector:
    matchLabels:
      app: typescript-app-prod
  template:
    metadata:
      labels:
        app: typescript-app-prod
    spec:
      containers:
        - name: typescript-app-prod
          image: us-central1-docker.pkg.dev/smart-processor-412022/typescriptto-repository/typescript-app-prod:v1
          ports:
            - containerPort: 3000
          imagePullPolicy: Always
```

b) **service-prod.yaml**

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: typescript-app-service-prod
spec:
  selector:
    app: typescript-app-prod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

c) **Mongodb-service-prod.yaml**

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-prod
spec:
  selector:
    app: mongodb-prod
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

d) **Mongodb-deployment-prod.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb-prod
  template:
    metadata:
```

```
labels:
  app: mongodb-prod
spec:
  containers:
  - name: mongodb-prod
    image: mongo
    ports:
    - containerPort: 27017
    # Additional settings for MongoDB configuration
    # Uncomment and modify the following lines as needed:

    # Volume mounts for persistent storage
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db

    # Environment variables for authentication
    env:
    - name: MONGO_INITDB_ROOT_USERNAME
      value: admin
    - name: MONGO_INITDB_ROOT_PASSWORD
      value: admin

    # PersistentVolumeClaim to persist data across pod restarts
    volumes:
    - name: mongodb-data
      persistentVolumeClaim:
        claimName: mongodb-data-pvc
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - typescript-app-prod
          topologyKey: "kubernetes.io/hostname"

    # PersistentVolumeClaim for MongoDB data
    ---
  apiVersion: v1
  kind: PersistentVolumeClaim
```

```
metadata:
  name: mongodb-data-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

d) CloudBuild-prod.yaml

```
steps:
  # Step 1: Build the Docker image using the Dockerfile in the
  repository
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'us-central1-docker.pkg.dev/smart-processor-412022/typescriptto-repository/typescript-app-prod:v1',
    '.']

  # Step 2: Push the Docker image to Google Container Registry
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'us-central1-docker.pkg.dev/smart-processor-412022/typescriptto-repository/typescript-app-prod:v1']

  # Step 3: Set up kubectl to use the cluster's credentials
  - name: 'gcr.io/cloud-builders/gcloud'
    args: ['container', 'clusters', 'get-credentials', 'todo-prod-cluster', '--region', 'us-central1', '--project', 'smart-processor-412022']

  # Step 4: Apply the Kubernetes deployment manifest
  - name: 'gcr.io/cloud-builders/kubectl'
    args: ['apply', '-f', 'k8s/deployment-prod.yaml']
    env:
      - 'CLOUDSDK_COMPUTE_REGION=us-central1'
      - 'CLOUDSDK_CONTAINER_CLUSTER=todo-prod-cluster'
      - 'CLOUDSDK_CORE_PROJECT=smart-processor-412022'

  # Step 5: Apply the Kubernetes service manifest
```

```
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['apply', '-f', 'k8s/service-prod.yaml']
  env:
    - 'CLOUDSDK_COMPUTE_REGION=us-central1'
    - 'CLOUDSDK_CONTAINER_CLUSTER=todo-prod-cluster'
    - 'CLOUDSDK_CORE_PROJECT=smart-processor-412022'

# Step 6: Apply the MongoDB Kubernetes deployment manifest
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['apply', '-f', 'k8s/mongodb-deployment-prod.yaml']
  env:
    - 'CLOUDSDK_COMPUTE_REGION=us-central1'
    - 'CLOUDSDK_CONTAINER_CLUSTER=todo-prod-cluster'
    - 'CLOUDSDK_CORE_PROJECT=smart-processor-412022'

# Step 7: Apply the MongoDB Kubernetes service manifest
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['apply', '-f', 'k8s/mongodb-service-prod.yaml']
  env:
    - 'CLOUDSDK_COMPUTE_REGION=us-central1'
    - 'CLOUDSDK_CONTAINER_CLUSTER=todo-prod-cluster'
    - 'CLOUDSDK_CORE_PROJECT=smart-processor-412022'

images:
  - 'us-central1-docker.pkg.dev/smart-processor-412022/typescriptto-repository/typescript-app-prod:v1'
```

g) Check Kubernetes Deployments, services, pods

```
sahuja49@cloudshell:~/typescript-app/src (smart-processor-412022)$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
mongodb              1/1     1             1           5d20h
typescript-app-prod  2/2     2             2           5d20h
sahuja49@cloudshell:~/typescript-app/src (smart-processor-412022)$ kubectl get services
NAME                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes          ClusterIP           34.118.224.1    <none>            443/TCP           5d21h
mongodb-prod        ClusterIP           34.118.227.86   <none>            27017/TCP         5d20h
typescript-app-service-prod  LoadBalancer      34.118.233.190  104.154.195.202  80:31548/TCP     5d20h
sahuja49@cloudshell:~/typescript-app/src (smart-processor-412022)$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
mongodb-cd57996f6-v4wgw  1/1     Running   0           2d4h
typescript-app-prod-75d79bd5d8-9zsz4  1/1     Running   0           42h
typescript-app-prod-75d79bd5d8-pb72p  1/1     Running   0           42h
sahuja49@cloudshell:~/typescript-app/src (smart-processor-412022)$
```

- The **kubectl get deployments** output lists two deployments: **mongodb** with 1/1 pods ready, and **typescript-app-prod** with 2/2 pods ready, signifying all instances are up and running without issues.

Western Engineering

ECE 9016 Cloud Computing

Project Report

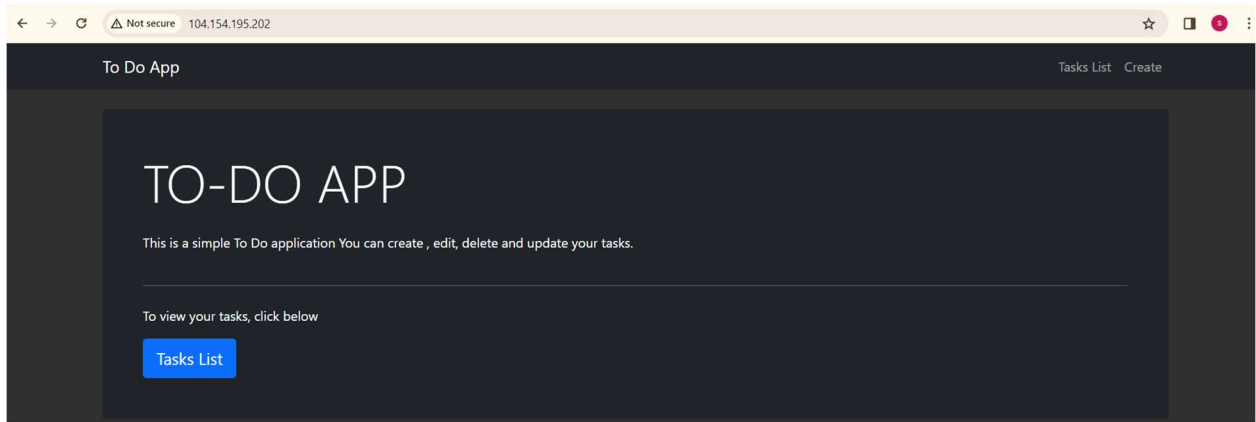
- The **kubectl get services** output reveals two services: a **ClusterIP** service for **mongodb-prod** and a **LoadBalancer** service for **typescript-app-service-prod**. The **LoadBalancer** type exposes **typescript-app-service-prod** externally at IP **104.154.195.202** on port **80**, routing to port **31548** on the cluster.
- The **kubectl get pods** command shows individual pod details for the MongoDB deployment and the typescript app production deployment. Both applications' pods are running, with no restarts, indicating stability in the current configuration.

4. High Availability Implementation and Verification

In our production environment, we have deployed a to do application with two pods and a MongoDB database with one pod. To ensure high availability, we have implemented a setup that includes load balancing and active-active node configurations.

Load Balancing:

We have set up a load balancer to distribute incoming traffic across the two pods of our To Do application. The load balancer ensures that requests are evenly distributed, optimizing resource utilization, and providing fault tolerance. Our load balancer has an external IP address (104.154.195.202) and forwards traffic to port 80 of the pods (todo-service-prod).



NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT (S)	AGE
kubernetes	ClusterIP	34.118.224.1	<none>	443/TCP	5d21h
mongodb-prod	ClusterIP	34.118.227.86	<none>	27017/TCP	5d20h
typescript-app-service-prod	LoadBalancer	34.118.233.190	104.154.195.202	80:31548/TCP	5d20h

Western Engineering

ECE 9016 Cloud Computing

Project Report

← Load balancer details EDIT DELETE → VIEW IN NETWORK TOPOLOGY

balancers list page

a741186f65a2645b59bebf9db8b7d792

Target-pool network load balancer

Frontend

Protocol ↑	IP version	IP:Port	Network Tier ?
TCP	IPv4	104.154.195.202:80	Premium

Backend

Name	Region	Health check
a741186f65a2645b59bebf9db8b7d792	us-central1	k8s-9bfe12daaf644b64-node

ADVANCED CONFIGURATIONS

Instance ↑	Zone	104.154.195.202
gk3-todo-prod-cluster-pool-2-572b36a9-zt7q	us-central1-a	✓
gk3-todo-prod-cluster-pool-2-9fee8168-9cx4	us-central1-f	✓
gk3-todo-prod-cluster-pool-2-9fee8168-s9b4	us-central1-f	✓

Active-Active Node Configuration:

To achieve high availability, we have configured our To do application pods in an active-active setup. Both pods are actively serving requests, allowing us to handle increased loads and providing redundancy in case one of the pods becomes unavailable. This configuration ensures continuous availability of our application even during maintenance or failures.

```
sahuja49@cloudshell:~/typescript-app/src (smart-processor-412022) $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mongodb-cd57996f6-v4wgw            1/1     Running   0           2d4h
typescript-app-prod-75d79bd5d8-9zzs4 1/1     Running   0           42h
typescript-app-prod-75d79bd5d8-pb72p 1/1     Running   0           42h
```

Failover Mechanism:

In the event of a pod failure, Kubernetes automatically restarts the failed pod and redirects traffic to the remaining healthy pods. This failover mechanism ensures minimal downtime and maintains seamless operation of our application. Additionally, our load balancer monitors the health of the pods and adjusts its routing accordingly, further enhancing the reliability of our setup.

```
sahuja49@cloudshell:~ (smart-processor-412022)$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
mongodb             1/1     1             1           6d5h
typescript-app-prod 1/1     1             1           6d5h
```

Our high availability setup ensures that our production environment is resilient to failures and capable of meeting the demands of our users. By implementing load balancing and active-active node configurations, we provide uninterrupted service and maintain the reliability of our application. Continuous monitoring, documentation, and scalability strategies further enhance the stability and performance of our infrastructure.

5. Replication for 3rd party

Follow below steps to replicate the project:

1. Clone our GitHub repository from the link below.

<https://github.com/sahuja49/bookstore-cloud.git>

```
git clone https://github.com/sahuja49/bookstore-cloud.git
```

2. Create a GKE cluster, use the Google Cloud Console or the following gcloud command

```
gcloud container clusters create [CLUSTER_NAME] --zone [ZONE]
```

3. Create two triggers for Dev and Prod environments.

- Navigate to Google Cloud Build in the Google Cloud Console.
- Create a trigger for the Dev environment:
- Click on "Triggers" in the left menu.
- Click on "Create Trigger".
- Configure trigger settings as per your requirements (e.g., choose the repository, branch, build configuration file, etc.).
- Repeat the above steps to create a trigger for the Prod environment

4. Create Artifacts Registry.

- Navigate to Artifact Registry in the Google Cloud Console.
- Click on "Create Repository".
- Configure repository settings as per your requirements (e.g., choose the repository location, permissions, etc.).

Western Engineering

ECE 9016 Cloud Computing

Project Report

5. Make a commit to any branch to trigger a Cloud build. Make changes to any file in your local repository. Stage and commit the changes

```
git add .  
git commit -m "Trigger Cloud Build"
```

Push the changes to the remote repository

```
git push origin [BRANCH_NAME]
```

This will trigger a Cloud Build based on the configured triggers.