Configuration manual:

Programming Language:

Node.js v18.x, JavaScript

Frameworks and Libraries:

- **Express.js**: Used to develop the REST APIs.
- **Cluster Module & Worker Threads**: Utilized to develop the custom load-balancing library (`r-loadBalancer.js`).

Containerisation:

- **Docker**: Used to containerize the application by creating containerized images.
- Kubernetes: Employed for deploying the containerized application and managing services.

Cloud Platforms:

AWS:

- Deployed the non-containerized application on AWS EC2.
- Deployed the containerized application using AWS EKS (Elastic Kubernetes Service)

GCP:

- Deployed the non-containerized application on GCP App Engine.
- Deployed the containerized application using GCP Kubernetes Engine.

Performance Monitoring:

- AWS CloudWatch: Used to monitor the performance of the application on AWS and collect key performance metrics and system health metrics.
- GCP Cloud Monitoring: Used to monitor the performance of the application on GCP and collect system and performance metrics.

Code configuration:

1. Node.js app:
   (a) Requirements:
   - Node.js V18.x or grater version
   - VS code ide
   - Git for local
   - Postman for API testing
   - Artillery for load testing

Github URL: https://github.com/Rohith-A/thesis-experiment.git

Steps to run the app run the following commands

(1) git clone https://github.com/Rohith-A/thesis-experiment.git - clone the repository
(2) npm install - install dependencies
(3) npm start - run the application
(4) http://localhost:3000/health open this URL in browser
(5) Go to app.js to find the API endpoints
(6) Readme file has all the instructions related to endpoints

2. Cloud deployment:

(a) Create EC2 instance in AWS with t3.xlarge instance and perform git clone and npm commands
(b) Create a ECR registry and EKS cluster with 8 core CPU and 32GB RAM and use deployment.yml file in kube folder in the application to deploy app to Kubernetes cluster
(c) Before deploying app to Kubernetes cluster build a docker image a DOCKERFILE is available in the application install docker locally and run docker build command to build docker image
(d) Create an application in GCP App engine with 4core CPU and 8GB RAM and use app.yaml file in the application to deploy the app to GCP app engine
(e) Use gcp-deployment file in kube folder by logging in to the gcp console and run the kubectl commands to deploy the image to Kubernetes engine which is a 8core CPU 32GB enivironment

3. r-loadBalancer.js:
This is the custom load balancing library build to enhance the performance of the node.js application.

URL: https://www.npmjs.com/package/r-load-balancer.js
Steps to install
- npm i r-load-balancer.js

You can install this library in any node.js application and sample use is available in readme file of this library.

Cloud Deployment:

- Install Docker (for containerized deployment).
- Install AWS CLI and configure with credentials.
- Install GCP CLI and configure with credentials.
- Have a GitHub repository or a local directory containing the Node.js application.

Section 1: Deployment on AWS

(A) Non-Containerized Deployment:

(1) Provision an EC2 Instance:

- Log in to the AWS Management Console.
- Navigate to EC2 → Launch an Instance.
- Choose an AMI:
  - Amazon Linux 2 or Ubuntu Server.
  - Select an instance type (e.g., t2.medium for moderate workloads).
  - Configure security groups to allow HTTP (port 80), HTTPS (port 443), and SSH (port 22).

(2) ssh -i your-key.pem ec2-user@<EC2-Public-IP>

(3) Install Node.js and Dependencies:

```
sudo yum update -y
curl -fsSL https://rpm.nodesource.com/setup_18.x |
sudo bash -
sudo yum install -y nodejs git
cd <app-directory> npm install
```

(4) Run the application: node app.js

(B) Containerized Deployment:

(1) Create a Dockerfile

```
FROM node:18
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "app.js"]
```

(2) Build and Run the Docker Image:

```
docker build -t my-node-app .
docker run -d -p 3000:3000 my-node-app
```

(3) Push the Image to Amazon Elastic Container Registry

```
aws ecr get-login-password --region <your-region> | docker login --username AWS --
password-stdin <your-account-id>.dkr.ecr.<your-region>.amazonaws.com
```

```
docker tag my-node-app:latest <your-ecr-repo-url>:latest
docker push <your-ecr-repo-url>:latest
```

Section 2: Deployment on GCP

(A) Non-Containerized Deployment

(1) Provision a Compute Engine Instance:

- Log in to the GCP Console.
- Navigate to **Compute Engine → VM instances**.
- Click "Create Instance" and configure:
- **Machine Type**: e2-medium or higher.
- **Firewall**: Allow HTTP and HTTPS traffic.
- From the GCP Console, click **SSH** on the instance.

(2) Install Node.js and Dependencies:

```
sudo apt update
sudo apt install -y curl git
curl -fsSL https://deb.nodesource.com/
setup_18.x | sudo -E bash -
sudo apt install -y nodejs
```

(3) Clone and Run the Application:

```
git clone https://github.com/Rohith-A/thesis-
experiment.git

cd <app-directory>
npm install
node app.js
```

(D) Containerized Deployment:

Creating and building docker image is same as AWS

(1) Push the Image to Google Container Registry (GCR):

```
gcloud auth configure-docker

docker tag my-node-app:latest gcr.io/<your-project-id>/my-node-app:latest
docker push gcr.io/<your-project-id>/my-node-app:latest
```

(2) Deploy with Google Kubernetes Engine (GKE):

```
gcloud container clusters create my-cluster --num-nodes=3

kubectl create deployment my-node-app —image=gcr.io/<your-project-id>/my-
node-app:latest

kubectl expose deployment my-node-app --type=LoadBalancer --port=80 --
target-port=3000
```

This manual provides all necessary steps to deploy the Node.js application in containerized and non-containerized forms on AWS and GCP. Adapt configurations as per your application's specific needs.