Great! Based on your provided doc (MasterClass Project.docx), here's how you can **provision a similar web app stack on GCP using only JavaScript (Node.js) instead of PHP**, keeping all other steps conceptually aligned:

**✅ MasterClass GCP Setup – Node.js Version**

**Step 1: Provision a VM on Google Cloud**

* Use GCP Console or gcloud CLI:

gcloud compute instances create nodejs-vm \

--zone=us-central1-c \

--machine-type=e2-medium \

--image-family=debian-12 \

--image-project=debian-cloud \

--tags=http-server

**Step 2: Install Node.js, npm, and MySQL Client**

SSH into your instance:

sudo apt update

sudo apt install curl mysql-client -y

curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -

sudo apt install -y nodejs

Verify:

node -v

npm -v

mysql --version

**Step 3: Test DB Connectivity**

Assuming you have a Cloud SQL instance:

mysql --ssl-mode=DISABLED --host=<YOUR\_SQL\_IP> --user=root -p

Then run:

CREATE DATABASE mydb;

USE mydb;

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100) UNIQUE

);

INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com');

**Step 4: Node.js App Setup**

**📁 Directory structure:**

masterclass-app/

│

├── app.js

├── db.js

└── views/

├── index.html

└── add.html

**db.js**

const mysql = require('mysql2');

const conn = mysql.createConnection({

host: 'YOUR\_SQL\_IP',

user: 'root',

password: 'Welcome1',

database: 'mydb'

});

conn.connect(err => {

if (err) throw err;

console.log("Connected to MySQL!");

});

module.exports = conn;

**app.js**

const express = require('express');

const app = express();

const conn = require('./db');

app.use(express.urlencoded({ extended: true }));

app.get('/', (req, res) => {

conn.query('SELECT \* FROM users', (err, results) => {

if (err) return res.send('Error fetching users');

let html = '<h2>Users</h2><ul>';

results.forEach(user => {

html += `<li>${user.name} (${user.email})</li>`;

});

html += '</ul><a href="/add">Add User</a>';

res.send(html);

});

});

app.get('/add', (req, res) => {

res.send(`

<form method="POST" action="/add">

<input name="name" required />

<input name="email" type="email" required />

<button>Add</button>

</form>

`);

});

app.post('/add', (req, res) => {

const { name, email } = req.body;

conn.query('INSERT INTO users (name, email) VALUES (?, ?)', [name, email], (err) => {

if (err) return res.send('Insert error: ' + err.message);

res.redirect('/');

});

});

app.listen(3000, () => {

console.log('App running on port 3000');

});

**Install dependencies:**

npm init -y

npm install express mysql2

**Step 5: Allow HTTP traffic & Run App**

sudo ufw allow 3000

node app.js

Access it at:  
http://<EXTERNAL\_IP>:3000

Would you like me to generate a Terraform script for provisioning this setup on GCP too?

Migration

e's a more detailed breakdown of the process:

1. Prepare Your Application for Containerization:

* **Identify Dependencies:**

Determine all the libraries, frameworks, and tools your web server application relies on.

* **Create a Dockerfile:**

Write a Dockerfile that specifies how to build your container image, including the base image, application code, and any necessary dependencies.

* **Build the Container Image:**

Use Docker to build your container image from the Dockerfile.

* **Tag and Push the Image:**

Tag the image with a unique name and push it to a container registry like Google Container Registry (GCR).

2. Deploy to Cloud Run:

* **Create a Cloud Run Service:** In the Google Cloud Console, navigate to Cloud Run and create a new service.
* **Configure the Service:**
  + **Service Name:** Choose a descriptive name for your service.
  + **Container Image:** Specify the location of your container image in GCR.
  + **Memory and CPU:** Set the memory and CPU resources for your service.
  + **Networking:** Configure networking settings, including VPC network and ingress settings.
* **Deploy the Service:** Deploy your Cloud Run service.

3. Configure Traffic Handling:

* **Traffic Splitting (Optional):** If you want a gradual rollout, use traffic splitting to route a portion of traffic to the new Cloud Run service while the remaining traffic goes to the VM.
* **DNS Configuration:** Update your DNS records to point to the Cloud Run service's URL.
* **Monitor and Optimize:** Monitor the performance of your Cloud Run service and make adjustments as needed.

4. Migrate Data (If Necessary):

* **Identify Data:** Determine which data is needed by your web server application.
* **Choose a Migration Strategy:** Choose a strategy for migrating data, such as:
  + **Data Replication:** Replicate data from the VM to a Cloud SQL instance or other data store used by Cloud Run.
  + **Data Migration:** Migrate data directly from the VM to the new Cloud Run service.
* **Execute the Migration:** Execute the chosen migration strategy.

5. Finalize the Migration:

* **Remove the VM:**

Once you've verified that the Cloud Run service is working correctly and all data has been migrated, you can remove the VM.