LAB MANUAL
of

INTERNET OF THINGS AND CLOUD COMPUTING

III-II- B.Tech

R22 Regulation

Department of Computer Science and Engineering

CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH, Accreditedby NBA, and NAAC)
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),
Ranga Reddy (Dist.) - 501510, Telangana State

CVR COLLEGE OF ENGINEERING

(UGC Autonomous Institution, Accredited by NAAC 'A' grade and NBA)
Affiliated to JNTU Hyderabad
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),
Ranga Reddy (Dist.), Hyderabad – 501510, Telangana State.

Department of Computer Science and Engineering

VISION

To be a state of the art institution of engineering in pursuit of excellence, in the service of society.

MISSION

To excel in providing quality education at under graduate and graduate levels.

To encourage research and innovation.

To provide infrastructure and facilities to meet the latest technological needs.

To establish Centres of Excellence through active interaction with industry.

To nurture students towards holistic development with human values and ethics.

22CS381 INTERNET OF THINGS & CLOUD COMPUTING LAB

Instruction	:	3Periods /week	CIE	:	40
Tutorial	:	-	SEE	:	60
Credits	:	1.5	EndExamDuration	:	3 Hours

CourseObjectives:

1. To offer students hands onexperience in inter facing sensors Raspberry Pi with Node-Red.
2. To implement a complete ecosystem in integrating cloud offerings for IoT & Cloudcomputing.
3. To develop python programs for Cloud Computing applications.

List of Experiments: IOT

Task 1:
Installing Node-RED on a Raspberry Pi through Remote Login.

Task 2:
Create a simple Node-RED flow that takes input from an inject node and displays output in a debug node. Add a function node to modify the payload in the flow.

Task 3:
Integrate an MQTT node into a Node-RED flow and subscribe to a topic.

Task 4:
Implement an HTTP request node to interact with an external API. (Twitter, WhatsApp)

Task 5:
Build a basic IoT dashboard using the Node-RED dashboard nodes.

Task 6:
Include widgets for displaying sensor data and control buttons.

Task 7:
Configure security settings for Node-RED, including user authentication.

Task 8:
Implement SSL/TLS for secure communication in a Node-RED instance.

List of Experiments: Cloud Computing

Task 1:
Create and Manage Cloud Resources
a) Tour of Google Cloud
b) Creating a Virtual Machine
c) Getting Started with Cloud Shell and g cloud
Task 2:
Kubernetes Engine:
a) Set up Network and HTTP Load Balancers
b) Create and Manage Cloud Resources: Challenge Lab
Task 3:
Perform Foundational Infrastructure Tasks
a) Cloud Storage: Qwik Start - Cloud Console
b) IAM in AWS and setup
c) Cloud Functions
d) Cloud networking

Task 4:
Set Up and Configure a Cloud Environment
a) Cloud IAM: Qwik Start
b) Introduction to SQL for Big Query and Cloud SQL
c) Database Tasks in Cloud
d) Cloud Monitoring: Qwik Start
e) Managing Deployments Using Kubernetes Engine
f) Set Up and Configure a Cloud Environment in Google Cloud: Challenge Lab
Task 5:
1. Introduction to Amazon EC2
2. Introduction to Amazon Simple Storage Service (S3)

Task 6:
1. Introduction to Amazon Relational Database Service (RDS) - SQL Server)
2. AWS Identity and Access Management (IAM) Task

Task 7:
1. Management of Amazon Elastic Container Service
2. Hosting a static website and Deploying a web application on AWS

Task 8:
a. Continuously Querying top 10 songs in the song-list or chart with Kubernetes
b. Key Parameter Indicators visualization for Airline services using Kubernetes

TASK 9:

a. Managing resources using Terraform

b. Creating and running containers.

Course outcomes: At the end of the course, the student will be able to

CO1: Implement framing, error detection techniques of data link layer.
CO2: Design and implement different routing algorithms in network layer.
CO3: Design and Implement security algorithms in application layer.
CO4: Understand the fundamentals of creating and managing cloud
CO5: Identify the significance of Kubernetes engine and process of tasks execution

References:

1. Andrew S Tanenbaum, *Computer Networks*, 5th Edition, Pearson Education, 2011.
2. Behrouz A. Forouzan, *Data Communications and Networking*, 5th Edition, TMH, 2009.
3. William Stallings, *Network Security Essentials: Applications and Standards*, 4th Edition Pearson Education, 2011
4. B.A. Forouzan and D. Mukhopadhyay, *Cryptography and Network Security*, TMH, 2ndEdition, TMH. 2010.

List of Experiments: IOT

Task 1:- Installing Node-RED on a Raspberry Pi through Remote Login.

Installing Node-RED on a Raspberry Pi through a remote login (like SSH) is straightforward.

Here's a step-by-step guide:

1. Access Your Raspberry Pi via SSH

- If you're using a Windows PC, you can use PuTTY to SSH into your Raspberry Pi. On macOS or Linux, you can use the terminal.
- Connect to your Raspberry Pi using the following command (replace pi and raspberrypi.local with your username and the IP address or hostname of your Raspberry Pi):

```
ssh pi@raspberrypi.local
```

Enter the password when prompted.

2. Update and Upgrade Your Raspberry Pi

- Once logged in, update the package list and upgrade installed packages:

```
bash

sudo apt-get update
sudo apt-get upgrade –y
```

3. Install Node-RED

- Node-RED can be installed using the official script provided by the Node-RED team. Run the following command:

```
bash

bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

This script will:

- Install or upgrade Node.js to the latest recommended version.
- Install or upgrade Node-RED.
- Install the necessary build tools.

4. Enable Node-RED to Start on Boot (Optional)

- To have Node-RED start automatically when the Raspberry Pi boots up, you can use the following command:
- bash

  sudo systemctl enable nodered.service

5. Start Node-RED

- You can manually start Node-RED with:

  bash
  node-red-start

- You can stop it with:

  bash
  node-red-stop

- To check the status of Node-RED:

  bash
  node-red-log

6. Access the Node-RED Interface

- Open a web browser on your computer and enter the following URL:

  arduino
  http://<raspberrypi.local>:1880

- Replace <raspberrypi.local> with the IP address or hostname of your Raspberry Pi.

7. (Optional) Secure Node-RED

- You might want to secure your Node-RED instance by setting up authentication or using HTTPS. This can be configured in the settings.js file located at ~/.node-red/settings.js.

Task 2:- Create a simple Node-RED flow that takes input from an inject node and displays output in a debug node. Add a function node to modify the payload in the flow.

To create a simple Node-RED flow that takes input from an inject node, modifies the payload using a function node, and then displays the output in a debug node, follow these steps:

1. Open Node-RED

Start Node-RED by running the command node-red in your terminal, and then open the Node-RED editor in your web browser (usually at http://localhost:1880).

2. Create the Flow

- **Inject Node:**
  - Drag an "Inject" node from the palette on the left into the workspace.
  - Double-click the Inject node to configure it.
  - Set the payload type to string and enter a value like "Hello".
  - Click "Done" to save the configuration.
- **Function Node:**
  - Drag a "Function" node into the workspace.
  - Connect the Inject node's output to the Function node's input.
  - Double-click the Function node to configure it.
  - Enter a JavaScript function to modify the payload, for example:

    Javascript

    msg.payload = msg.payload + ", Node-RED!"; return msg;

  - Click "Done" to save the function.
- **Debug Node:**
  - Drag a "Debug" node into the workspace.
  - Connect the Function node's output to the Debug node's input.
  - The Debug node will display the modified payload in the debug window on the right side of the editor.

3. Deploy and Test the Flow

- Click the "Deploy" button at the top right of the Node-RED editor.
- Click the button on the left side of the Inject node to trigger the flow.
- The modified message should appear in the debug panel on the right, showing "Hello, Node-RED!".

Visual Representation of the Flow

```
css
[Inject Node] --> [Function Node] --> [Debug Node]
```

This simple flow demonstrates how to take an input, process it, and output the result in Node-RED.

Task 3:- Integrate an MQTT node into a Node-RED flow and subscribe to a topic.

To integrate an MQTT node into a Node-RED flow and subscribe to a topic, follow these steps:

1. Set Up MQTT Broker

If you don't already have an MQTT broker, you can use a public broker like broker.hivemq.com or set up a local broker using tools like Mosquitto.

2. Install MQTT Nodes (if needed)

If the MQTT nodes aren't already installed in your Node-RED instance, you can install them via the "Manage Palette" option:

- Click the menu (three horizontal lines) in the top right corner.
- Select "Manage Palette."
- Go to the "Install" tab and search for "node-red-node-mqtt."
- Click "Install."

3. Create the Flow

Step 1: Add an MQTT In Node

- Drag an "MQTT in" node from the palette into your workspace.
- Double-click the node to configure it:
    - **Server:** Click the pencil icon to add a new MQTT broker configuration.
        - Enter the broker URL (e.g., mqtt://broker.hivemq.com).
        - Set the port to 1883 (default for non-SSL connections).
        - If needed, provide credentials (for public brokers, this is often not required).
        - Click "Add."
    - **Topic:** Enter the topic you want to subscribe to (e.g., test/topic).
    - **QoS:** Set the Quality of Service level (0, 1, or 2) as per your requirement.
    - Click "Done" to save the configuration.

Step 2: Add a Debug Node

- Drag a "Debug" node into the workspace.
- Connect the output of the MQTT in node to the input of the Debug node.
- The Debug node will display messages received from the MQTT topic in the debug panel.

Optional Step 3: Add a Function Node

- If you want to process the received message, you can add a "Function" node between the MQTT in and Debug nodes.
- Configure the Function node with some JavaScript code to modify the payload, if necessary.

4. Deploy and Test the Flow

- Click the "Deploy" button at the top right of the Node-RED editor.
- The flow will start listening to the specified MQTT topic.
- When a message is published to the topic (e.g., via an MQTT client like MQTT.fx or another Node-RED flow), it will be displayed in the debug panel.

Visual Representation of the Flow

mathematica

```
[MQTT In Node] --> [Function Node (Optional)] --> [Debug Node]
```

Example Use Case

- **MQTT Topic:** sensor/temperature
- **Function Node Script:**

Javascript

```
msg.payload = "Received temperature: " + msg.payload + "°C";
return msg;
```

Task 4:- Implement an HTTP request node to interact with an external API. (Twitter, WhatsApp)

To implement an HTTP request node that interacts with an external API like Twitter or WhatsApp, you typically need to create a service or use a tool that allows you to send HTTP requests and handle responses.

Below are two examples demonstrating how to implement an HTTP request node for both Twitter and WhatsApp APIs using Node.js with the axios library.

Prerequisites

1. **Node.js**: Make sure you have Node.js installed on your machine. You can download it from nodejs.org.
2. **API Keys**: Obtain the necessary API keys and tokens for Twitter or WhatsApp. You can get these by registering your application on the respective platforms.

1. Interacting with Twitter API

Step 1: Set Up the Node.js Project

1. Create a new directory for your project:

   bash
   mkdir twitter-api-node
   cd twitter-api-node

2. Initialize the Node.js project:

   bash
   npm init -y

3. Install the required dependencies:

   bash
   npm install axios dotenv

4. Create a .env file in your project root to store your Twitter API credentials:

   plaintext
   TWITTER_BEARER_TOKEN=your_twitter_bearer_token_here

Step 2: Write the Node.js Script

1. Create an index.js file in the root of your project:

```
javascript
const axios = require('axios');
require('dotenv').config();

const twitterAPI = axios.create({
   baseURL: 'https://api.twitter.com/2',
   headers: {
      'Authorization': `Bearer ${process.env.TWITTER_BEARER_TOKEN}`
   }
});

// Function to fetch a user's recent tweets
async function getUserTweets(username) {
   try {
      const response = await twitterAPI.get(`/tweets?username=${username}`);
      return response.data;
   } catch (error) {
      console.error('Error fetching tweets:', error.response ? error.response.data :
error.message);
```

```
    }
}

// Example usage
(async () => {
    const tweets = await getUserTweets('TwitterDev');
    console.log(tweets);
})();
```

2. Run the script:

   bash
   node index.js

This script makes a GET request to the Twitter API to fetch recent tweets from a specific user. Replace 'TwitterDev' with the Twitter handle you want to query.

2. Interacting with WhatsApp API (Using Twilio)

Twilio provides a WhatsApp API that allows you to send and receive messages using their service.

Step 1: Set Up the Node.js Project

1. If you haven't done so already, create a new directory for your project:

   bash
   mkdir whatsapp-api-node
   cd whatsapp-api-node

2. Initialize the Node.js project:

   bash
   npm init -y

3. Install the required dependencies:

   bash
   npm install axios twilio dotenv

4. Create a .env file in your project root to store your Twilio credentials:

   plaintext
   TWILIO_ACCOUNT_SID=your_twilio_account_sid_here
   TWILIO_AUTH_TOKEN=your_twilio_auth_token_here
   TWILIO_WHATSAPP_FROM=whatsapp:+14155238886 # This is Twilio's WhatsApp sandbox number

Step 2: Write the Node.js Script

1. Create an index.js file in the root of your project:

   javascript
   ```
   const axios = require('axios');
   const twilio = require('twilio');
   require('dotenv').config();

   const client = twilio(process.env.TWILIO_ACCOUNT_SID,
   process.env.TWILIO_AUTH_TOKEN);

   // Function to send a WhatsApp message
   async function sendWhatsAppMessage(to, message) {
     try {
       const response = await client.messages.create({
         from: process.env.TWILIO_WHATSAPP_FROM,
         to: `whatsapp:${to}`,
         body: message
       });
       console.log('Message sent:', response.sid);
     } catch (error) {
       console.error('Error sending message:', error.message);
     }
   }

   // Example usage
   (async () => {
     await sendWhatsAppMessage('+1234567890', 'Hello from Node.js and Twilio!');
   })();
   ```

2. Run the script:

   bash
   ```
   node index.js
   ```

This script uses the Twilio API to send a WhatsApp message. Replace '+1234567890' with the recipient's WhatsApp number.

Summary

- **Twitter API**: The script uses the Twitter API v2 to fetch recent tweets from a user.
- **WhatsApp API**: The script uses Twilio's API to send a WhatsApp message.

Both examples demonstrate how to set up a simple HTTP request node to interact with external APIs using Node.js and axios. You can expand these scripts to include more complex interactions, such as handling different HTTP methods (GET, POST, PUT, DELETE), managing API responses, and integrating with other parts of an application.

Task 5:- Build a basic IoT dashboard using the Node-RED dashboard nodes.

Building a basic IoT dashboard using Node-RED dashboard nodes involves several steps. The Node-RED dashboard allows you to create a web-based user interface for monitoring and controlling IoT devices. Here's how to create a simple IoT dashboard that displays sensor data and includes controls for an IoT device.

Prerequisites

- **Node-RED Installed**: Ensure Node-RED is installed and running.
- **Node-RED Dashboard Installed**: If the Node-RED dashboard nodes are not installed, add them via the "Manage Palette" option.
    - Click the menu (three horizontal lines) in the top right corner.
    - Select "Manage Palette."
    - Go to the "Install" tab, search for "node-red-dashboard," and click "Install."

Step 1: Install and Set Up Dashboard Nodes

1. **Inject Node (Simulating Sensor Data)**:
    - Drag an "Inject" node into the workspace.
    - Configure it to inject a timestamp or numeric payload (e.g., random sensor data) at regular intervals.
    - Set the interval to inject every 1 second, for example.
2. **Function Node (Simulate Sensor Data Processing)**:
    - Drag a "Function" node into the workspace.
    - Connect the Inject node to the Function node.
    - Configure the Function node with JavaScript to simulate sensor data:

```
javascript
msg.payload = {
  temperature: Math.random() * 100, // Simulated temperature data
  humidity: Math.random() * 100     // Simulated humidity data
};
return msg;
```

3. **Dashboard Gauge Node**:
    - Drag a "Gauge" node from the dashboard category into the workspace.
    - Connect the Function node to the Gauge node.
    - Double-click the Gauge node to configure it:
        - **Group**: Create a new group (e.g., "Sensor Data") under a new tab (e.g., "IoT Dashboard").
        - **Label**: Set the label (e.g., "Temperature").
        - **Value Format**: Set the format (e.g., {{msg.payload.temperature}} °C).
        - **Range**: Set the min and max values according to the expected sensor data range (e.g., 0 to 100).
    - Click "Done."

4. **Dashboard Chart Node**:
    - Drag a "Chart" node into the workspace.
    - Connect the Function node to the Chart node.
    - Double-click the Chart node to configure it:
        - **Group**: Select the same group as the Gauge node.
        - **Label**: Set the label (e.g., "Temperature Over Time").
        - **Data Format**: Set to {{msg.payload.temperature}}.
        - **X-axis**: Set to "Time".
        - **Y-axis**: Set to the temperature range (e.g., 0 to 100).
    - Click "Done."
5. **Dashboard Switch Node (Controlling IoT Device)**:
    - Drag a "Switch" node from the dashboard category into the workspace.
    - Configure it to control a mock IoT device, such as turning an LED on or off:
        - **Group**: Create a new group (e.g., "Device Control") under the same tab.
        - **Label**: Set the label (e.g., "LED").
        - **On Payload**: Set to true.
        - **Off Payload**: Set to false.
        - **Topic**: Set to device/led (or any relevant topic for the IoT device).
    - Click "Done."
6. **Debug Node**:
    - Drag a "Debug" node into the workspace.
    - Connect the output of the Switch node to the Debug node to see the control commands in the debug panel.

Step 2: Deploy and Access the Dashboard

- **Deploy**: Click the "Deploy" button at the top right of the Node-RED editor to deploy your flow.
- **Access the Dashboard**: The Node-RED dashboard is typically accessible at http://localhost:1880/ui. Open this URL in your browser to view the dashboard.

Visual Representation of the Flow

css

[Inject Node] --> [Function Node (Sensor Data)] --> [Gauge Node] & [Chart Node]
[Dashboard Switch Node] --> [Debug Node]

Step 3: Customize and Extend

- **Customize**: Modify the dashboard's appearance by changing the layout, colors, and themes from the dashboard tab in Node-RED.
- **Extend**: Add more controls, graphs, and other widgets to the dashboard as needed. You can also connect to real IoT devices by replacing the Inject and Function nodes with actual sensor and actuator nodes (e.g., MQTT nodes for real-time data).

Example Use Case

- **Temperature and Humidity Monitoring**: The dashboard shows real-time temperature and humidity data using gauges and charts.

- **Device Control**: A switch controls an IoT device (e.g., an LED), allowing the user to turn it on or off from the dashboard.

This basic IoT dashboard demonstrates how to visualize sensor data and control devices using Node-RED. It can be expanded with additional features such as alerts, sliders, buttons, and more complex data processing.

Task 6:- Include widgets for displaying sensor data and control buttons.

For Task 6, which involves including widgets for displaying sensor data and control buttons, you can achieve this by using a web application framework or a dashboard tool. Below are the steps to create a simple dashboard with widgets for displaying sensor data and control buttons.

1. Choose a Framework or Dashboard Tool

Depending on your requirements, you can choose from several options to create the dashboard:

- **Web Frameworks:** If you prefer a custom solution, you can use web frameworks like **Flask** or **Django** (Python), **Express.js** (Node.js), or **React** (JavaScript) to create the dashboard.
- **Dashboard Tools:** If you need a ready-made solution, you can use dashboard tools like **Dash** (Python), **Streamlit** (Python), or **Grafana** for a more extensive and customizable dashboard.

2. Setting Up the Environment

- Install the necessary libraries or tools. For example, if using Flask with Python:

  bash
  pip install flask

- If using a dashboard tool like Dash:

  bash
  pip install dash

3. Create the Web Application

Option A: Using Flask (Python)

1. **Create a Basic Flask Application:**

   Create a file named app.py:

   python
   from flask import Flask, render_template
   app = Flask(__name__)

```
@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

2. **Create an HTML Template:**

   In a folder named templates, create an index.html file:

```
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sensor Dashboard</title>
</head>
<body>
    <h1>Sensor Data Dashboard</h1>

    <!-- Display sensor data -->
    <div>
        <h2>Temperature: <span id="temp_value">--</span> °C</h2>
        <h2>Humidity: <span id="humidity_value">--</span> %</h2>
    </div>

    <!-- Control buttons -->
    <div>
        <button onclick="controlDevice('ON')">Turn ON</button>
        <button onclick="controlDevice('OFF')">Turn OFF</button>
    </div>

    <script>
        // Function to control device
        function controlDevice(state) {
            fetch('/control?state=' + state)
            .then(response => response.json())
            .then(data => {
                console.log(data);
            });
        }

        // Function to fetch sensor data
        function fetchSensorData() {
            fetch('/sensor_data')
            .then(response => response.json())
```

```
            .then(data => {
                document.getElementById('temp_value').innerText = data.temperature;
                document.getElementById('humidity_value').innerText = data.humidity;
            });
        }

        // Fetch data every 5 seconds
        setInterval(fetchSensorData, 5000);
    </script>
</body>
</html>
```

3. **Add Routes for Sensor Data and Control Actions:**

   Modify app.py:

   python

```
from flask import Flask, render_template, request, jsonify

app = Flask(__name__)

# Simulated sensor data (in a real application, fetch from sensors)
sensor_data = {
    "temperature": 25.0,
    "humidity": 50.0
}

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/sensor_data')
def get_sensor_data():
    return jsonify(sensor_data)

@app.route('/control')
def control_device():
    state = request.args.get('state')
    # Implement control logic here
    print(f'Device turned {state}')
    return jsonify({"status": "success", "state": state})

if __name__ == '__main__':
    app.run(debug=True)
```

4. **Run the Application:**

   bash

python app.py

Visit http://localhost:5000 in your web browser to view the dashboard.

Option B: Using Dash (Python)

1. **Create a Dash Application:**

   Create a file named app.py:

```
python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import random

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("Sensor Data Dashboard"),
    html.Div([
        html.H2("Temperature: "),
        html.Span(id='temp_value', children="--"),
        html.Span(" °C")
    ]),
    html.Div([
        html.H2("Humidity: "),
        html.Span(id='humidity_value', children="--"),
        html.Span(" %")
    ]),
    html.Button('Turn ON', id='btn-on', n_clicks=0),
    html.Button('Turn OFF', id='btn-off', n_clicks=0),
    html.Div(id='control-output')
])

# Callback to update sensor data
@app.callback(
    [Output('temp_value', 'children'),
     Output('humidity_value', 'children')],
    [Input('btn-on', 'n_clicks'),
     Input('btn-off', 'n_clicks')]
)
def update_sensor_data(n_on, n_off):
    temperature = random.uniform(20, 30)  # Replace with real sensor data
    humidity = random.uniform(40, 60)     # Replace with real sensor data
    return f"{temperature:.2f}", f"{humidity:.2f}"

# Callback to control device
```

```
@app.callback(
    Output('control-output', 'children'),
    [Input('btn-on', 'n_clicks'),
     Input('btn-off', 'n_clicks')]
)
def control_device(n_on, n_off):
    if n_on > n_off:
        return "Device turned ON"
    elif n_off > n_on:
        return "Device turned OFF"
    return "Waiting for command..."

if __name__ == '__main__':
    app.run_server(debug=True)
```

2. **Run the Application:**

   bash
   python app.py

   Visit http://localhost:8050 in your web browser to view the dashboard.

4. Enhance the Dashboard

- **Add More Widgets:** Add additional widgets for more sensors or controls (e.g., sliders for setting values, real-time graphs).
- **Styling:** Use CSS or a UI library like Bootstrap to improve the appearance of your dashboard.
- **Real-time Updates:** Implement WebSockets for real-time data updates if needed.
- **Deploy the Application:** Deploy the application to a cloud provider (e.g., AWS, Heroku) for broader accessibility.

By following these steps, you can create a functional dashboard with widgets to display sensor data and control buttons.

Task 7:- Configure security settings for Node-RED, including user authentication.

To configure security settings and enable user authentication for Node-RED, follow these steps:

1. Install Node-RED (if not already installed)

If Node-RED is not yet installed, you can install it using npm:

bash
sudo npm install -g --unsafe-perm node-red

2. Generate a Secure Password Hash

Node-RED uses bcrypt for password hashing. To generate a bcrypt hash for your password, you can use Node-RED's built-in script:

bash
node-red admin hash-pw

You'll be prompted to enter your desired password, and the script will output a hashed version of it. Copy this hash for later use.

3. Edit the settings.js File

Locate the settings.js file, which is typically found in the .node-red directory in your home folder:

bash
~/.node-red/settings.js

Open this file in your preferred text editor.

4. Enable User Authentication

To enable user authentication, find the adminAuth section in the settings.js file and uncomment it. It should look something like this:

javascript
```
adminAuth: {
   type: "credentials",
   users: [{
      username: "admin",
      password: "<your-hashed-password>",
      permissions: "*"
   }]
},
```

- **username**: Replace "admin" with your desired username.
- **password**: Replace "<your-hashed-password>" with the hashed password you generated earlier.
- **permissions**: The "*" permission gives the user full access to the Node-RED editor and runtime.

5. (Optional) Enable HTTPS

For better security, you should enable HTTPS. To do this, you need a TLS/SSL certificate and key.

1. Obtain a certificate and key, either through a certificate authority (CA) or by generating a self-signed certificate.
2. Save the certificate and key in a secure location.
3. Update the https section in the settings.js file:

javascript
```
https: {
   key: require("fs").readFileSync('/path/to/your/privatekey.pem'),
   cert: require("fs").readFileSync('/path/to/your/certificate.pem')
},
```

Replace '/path/to/your/privatekey.pem' and '/path/to/your/certificate.pem' with the actual paths to your key and certificate files.

6. Restricting Access to the Dashboard (Optional)

If you're using Node-RED Dashboard and want to restrict access:

1. Add a httpNodeAuth section to the settings.js file:

javascript
```
httpNodeAuth: {user:"admin", pass:"<your-hashed-password>"}
```

2. You can also add httpStaticAuth to secure static content served by Node-RED:

javascript
```
httpStaticAuth: {user:"admin", pass:"<your-hashed-password>"}
```
7. Start Node-RED

After making these changes, start or restart Node-RED:

bash
```
node-red
```
8. Test Your Configuration

Visit your Node-RED instance in a web browser. You should be prompted to log in with the credentials you configured.

Summary

You've now configured Node-RED with user authentication and optional HTTPS, which significantly improves the security of your instance. For even tighter security, consider additional measures like restricting access to your server by IP or using a reverse proxy with additional authentication layers.

Task 8:- Implement SSL/TLS for secure communication in a Node-RED instance.

Implementing SSL/TLS for secure communication in a Node-RED instance involves generating or obtaining SSL certificates and configuring Node-RED to use them. Here's a step-by-step guide to help you set it up:

1. Generate SSL/TLS Certificates

- **Option 1: Use Let's Encrypt (Recommended for Production)**
  - You can use Let's Encrypt to generate free SSL certificates.
  - You can use tools like Certbot to automatically generate and renew certificates.
  - The generated certificates will typically be located in /etc/letsencrypt/live/yourdomain/.
- **Option 2: Self-Signed Certificates (Recommended for Testing)**
  - For local or development purposes, you can generate a self-signed certificate:

    bash
    openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes

  - This command generates a key.pem and cert.pem file in the current directory.

2. Locate or Install Node-RED

- Ensure you have Node-RED installed and set up on your server.
- If not installed, you can install Node-RED using npm:

  bash
  npm install -g node-red

3. Configure Node-RED to Use SSL/TLS

- Open the Node-RED settings file. This file is usually found in ~/.node-red/settings.js.
  - You can open it using a text editor:

    bash
    Copy code
    nano ~/.node-red/settings.js

- Locate the section for https configuration in the settings.js file. It might look like this:

  javascript
  // Securing Node-RED with HTTPS
  // Uncomment and configure as follows:
  https: {

```
    key: require("fs").readFileSync('/path/to/your/key.pem'),
      cert: require("fs").readFileSync('/path/to/your/cert.pem'),
      // passphrase: "optional passphrase if your key has one",
      // Uncomment the next line to require client certificates
      // requestCert: true,
      // Uncomment the next line to verify client certificates against a CA
      // ca: require("fs").readFileSync('/path/to/your/ca.pem')
    },
```

- Replace /path/to/your/key.pem and /path/to/your/cert.pem with the actual paths to your certificate files.

3. Change the Port (Optional)
4. By default, Node-RED runs on port 1880. With SSL/TLS enabled, you might want to change it to 443 (standard HTTPS port) or any other secure port.

- Modify the uiPort setting in the settings.js file:javascript

  uiPort: process.env.PORT || 443,

5. Restart Node-RED

- After saving the changes, restart Node-RED for the changes to take effect:

  bash
  node-red-stop
  node-red-start

- If you're running Node-RED as a system service:

  bash
  sudo systemctl restart nodered

6. Test the Setup

- Open a browser and navigate to https://yourdomain.com (or https://your-ip:port if using an IP address).
- You should see the Node-RED interface with a secure HTTPS connection.

7. (Optional) Redirect HTTP to HTTPS

- To ensure all traffic is encrypted, you might want to redirect HTTP traffic to HTTPS.
- This can be done by setting up a reverse proxy using Nginx or Apache in front of Node-RED.

Example Nginx Configuration:

nginx

```
server {
    listen 80;
    server_name yourdomain.com;
    return 301 https://$host$request_uri;
}
```

This will redirect all HTTP traffic to HTTPS.

List of Experiments: Cloud Computing

Task 1:

Create and Manage Cloud Resources

a) Tour of Google Cloud
b) Creating a Virtual Machine
c) Getting Started with Cloud Shell and g cloud

Here's a step-by-step guide to help you with the tasks of creating and managing cloud resources on Google Cloud Platform (GCP):

Create and Manage Cloud Resources

a) Tour of Google Cloud

1. **Sign Up for Google Cloud**
   - Go to the Google Cloud Console.
   - If you don't have an account, sign up for one. Google offers a free tier with credits for new users.
   - Once signed in, you'll be in the Google Cloud Console, which is the web-based management interface for Google Cloud.
2. **Explore the Google Cloud Console**
   - **Navigation Menu**: Located on the top left (three horizontal lines), this menu gives you access to all GCP services such as Compute Engine, Cloud Storage, BigQuery, etc.
   - **Dashboard**: The main dashboard provides an overview of your resources, billing information, and project information.
   - **Projects**: GCP organizes resources under projects. You can create multiple projects to manage different environments or applications.
   - **IAM & Admin**: Manage user permissions and access control here.
   - **Billing**: Monitor your usage and costs associated with your projects.
   - **APIs & Services**: Manage API usage, enable or disable APIs, and access API credentials.
3. **Explore Documentation and Support**
   - Access the documentation through the "Documentation" link in the console. This is a valuable resource for learning more about specific services.
   - The "Help" option offers various support options, including community forums and direct support (depending on your support plan).

b) Creating a Virtual Machine

1. **Navigate to Compute Engine**
   - In the Google Cloud Console, click on the **Navigation Menu** (top left).
   - Select **Compute Engine** > **VM instances**.
2. **Create a New VM Instance**
   - Click on the **Create Instance** button.
   - **Name** your instance something identifiable (e.g., my-vm-instance).

- **Region and Zone**: Select a region close to your user base or requirements. The zone is a specific data center within a region.
- **Machine Configuration**:
  - Choose a **machine family** (e.g., General-purpose).
  - Select a **machine type** (e.g., e2-medium with 2 vCPUs and 4 GB RAM).
- **Boot Disk**:
  - The default is a Debian Linux image, but you can choose other operating systems.
  - Set the disk size (default is 10 GB).
- **Firewall**: You can allow HTTP and HTTPS traffic if you plan to run a web server.
- **Identity and API access**: Choose default service account or a specific service account for the VM.
- Click **Create** to launch your virtual machine.

3. **Accessing Your VM**
   - Once the VM is created, it will appear in the VM instances list.
   - Click on the **SSH** button next to your VM to open a terminal session directly in your browser.
4. **Stop/Delete VM (When Not in Use)**
   - To avoid charges, remember to stop or delete your VM when it's not needed. You can do this from the VM instances page by clicking on the three-dot menu next to your instance.

c) Getting Started with Cloud Shell and gcloud

1. **Open Cloud Shell**
   - In the Google Cloud Console, look for the **Cloud Shell** icon in the upper right corner (a terminal icon).
   - Click the icon to open a Cloud Shell session. This gives you access to a Debian-based shell with gcloud and other tools pre-installed.
   - Cloud Shell is free to use, with a small amount of persistent storage.
2. **Initialize the gcloud CLI**
   - Cloud Shell will automatically authenticate with your Google account and set up the gcloud CLI.
   - You can check your gcloud configuration by running:

     ```
     bash
     gcloud config list
     ```

   - If needed, set the project you're working on:

     ```
     bash
     gcloud config set project [PROJECT_ID]
     ```

3. **Basic gcloud Commands**
   - **List Available Zones**:

     ```
     bash
     gcloud compute zones list
     ```

- **Create a VM Instance**:

  bash
  ```
  gcloud compute instances create my-vm-instance \
    --zone=us-central1-a \
    --machine-type=e2-medium \
    --subnet=default \
    --tags=http-server,https-server \
    --image-family=debian-10 \
    --image-project=debian-cloud \
    --boot-disk-size=10GB
  ```

- **SSH into a VM**:

  bash
  ```
  gcloud compute ssh my-vm-instance --zone=us-central1-a
  ```

- **Stop a VM**:

  bash
  ```
  gcloud compute instances stop my-vm-instance --zone=us-central1-a
  ```

- **Delete a VM**:

  bash
  ```
  gcloud compute instances delete my-vm-instance --zone=us-central1-a
  ```

4. **Explore Further**
   - You can explore more gcloud commands by typing gcloud in Cloud Shell, which will list all available commands and their usage.
   - Google Cloud Shell also allows you to deploy applications, manage Kubernetes clusters, and more.

Summary

By following these steps, you should have a good understanding of the Google Cloud Console, how to create and manage a virtual machine, and how to use Cloud Shell and the gcloud CLI to interact with your Google Cloud resources.

Task 2:

Kubernetes Engine:

a) Set Up Network and HTTP Load Balancers

b) Create and Manage Cloud Resources: Challenge Lab

Kubernetes Engine

a) Set Up Network and HTTP Load Balancers

In Google Kubernetes Engine (GKE), load balancers are essential for distributing traffic across your Kubernetes cluster. GKE supports two main types of load balancers: Network Load Balancers and HTTP/HTTPS Load Balancers.

1. Setting Up a Network Load Balancer

A Network Load Balancer is used to distribute TCP/UDP traffic. Here's how to set it up in GKE:

1. **Create a GKE Cluster**
   - Ensure that your GKE cluster is up and running. If you haven't created one yet, you can do so using the gcloud command:

     bash
     gcloud container clusters create my-cluster --zone us-central1-a

   - Connect to the cluster:

     bash
     gcloud container clusters get-credentials my-cluster --zone us-central1-a

2. **Deploy an Application**
   - Create a simple deployment, for example, a nginx deployment:

     yaml
     apiVersion: apps/v1
     kind: Deployment
     metadata:
       name: nginx-deployment
     spec:
       replicas: 3
       selector:
         matchLabels:
           app: nginx
       template:
         metadata:
           labels:
             app: nginx

```
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

- Apply the deployment:

```
bash
kubectl apply -f nginx-deployment.yaml
```

3. **Expose the Deployment with a Network Load Balancer**
   - Create a service of type LoadBalancer:

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

   - Apply the service:

```
bash
kubectl apply -f nginx-service.yaml
```

   - After a few moments, GKE will provision a Network Load Balancer. You can retrieve the external IP address with:

```
bash
kubectl get services
```

   - The EXTERNAL-IP field in the output will show the IP address of the Network Load Balancer.
4. **Test the Load Balancer**
   - Visit the external IP address in your web browser. You should see the default NGINX welcome page.

2. Setting Up an HTTP/HTTPS Load Balancer

An HTTP/HTTPS Load Balancer is used to route HTTP/HTTPS traffic to your application. This type of load balancer can be set up using an Ingress resource in GKE.

1. **Create an Ingress Controller**
   - GKE comes with a built-in Ingress controller. You can define an Ingress resource in your Kubernetes manifests to create an HTTP/HTTPS Load Balancer.
2. **Deploy an Application**
   - Ensure you have a deployment similar to the one created above.
3. **Create a Service**
   - The service must be of type NodePort if you're using Ingress:

     yaml
     apiVersion: v1
     kind: Service
     metadata:
       name: nginx-service
     spec:
       selector:
         app: nginx
       ports:
         - protocol: TCP
           port: 80
           targetPort: 80
           nodePort: 30001
       type: NodePort

   - Apply the service:

     bash
     kubectl apply -f nginx-service.yaml

4. **Create an Ingress Resource**
   - Define an Ingress resource:

     yaml
     apiVersion: networking.k8s.io/v1
     kind: Ingress
     metadata:
       name: nginx-ingress
       annotations:
         ingress.kubernetes.io/rewrite-target: /
     spec:
       rules:
       - host: my-app.example.com
         http:
           paths:
           - path: /
             pathType: Prefix

```
    backend:
      service:
        name: nginx-service
        port:
          number: 80
```

- Apply the Ingress resource:

  bash
  kubectl apply -f nginx-ingress.yaml

- GKE will create an HTTP/HTTPS Load Balancer and associate it with the service.

5. **Configure DNS**
   - Point your domain (my-app.example.com) to the external IP address of the HTTP/HTTPS Load Balancer.
   - You can find the external IP of the Ingress using:

     bash
     kubectl get ingress

6. **Test the Load Balancer**
   - Once the DNS propagation is complete, visiting my-app.example.com in a browser should direct traffic to your Kubernetes service.

b) Create and Manage Cloud Resources: Challenge Lab

The "Challenge Lab" typically refers to a lab environment provided by Google Cloud Skills Boost (formerly Qwiklabs) where you're required to complete a set of tasks that test your ability to manage cloud resources effectively.

Steps to Complete a Challenge Lab:

1. **Start the Lab**
   - Log into the Google Cloud Skills Boost platform.
   - Start the Challenge Lab for managing cloud resources.
2. **Understand the Requirements**
   - Read the task descriptions carefully. You'll be given specific tasks, such as setting up VMs, managing Kubernetes resources, or configuring networking components.
3. **Complete the Tasks**
   - Follow the instructions step-by-step to complete the tasks. Examples might include:
     - Creating VM instances.
     - Configuring Kubernetes Engine.
     - Setting up load balancers.
     - Managing IAM roles and permissions.
     - Configuring networking, such as VPCs and subnets.
4. **Verify Your Work**

- Most labs have a verification step where you can check if your tasks were completed correctly.
- Make sure to use the provided commands or interfaces to confirm that your configurations are correct.

5. **Submit the Lab**
   - Once all tasks are completed, submit the lab. Your score will be based on how accurately you completed the tasks within the allotted time.
6. **Review Feedback**
   - After submission, review any feedback or errors to learn from the experience and understand any mistakes.

Summary

You've now learned how to set up Network and HTTP/HTTPS Load Balancers in GKE, and how to approach a Challenge Lab that tests your skills in creating and managing Google Cloud resources. These skills are fundamental for deploying and managing applications in a cloud-native environment.

Task 3:
Perform Foundational Infrastructure Tasks
a). Cloud Storage: Qwik Start - Cloud Console
b) IAM in AWS and setup
c)Cloud Functions
d)Cloud networking
Perform Foundational Infrastructure Tasks

a) Cloud Storage: Qwik Start - Cloud Console

Google Cloud Storage is a scalable and secure object storage service for storing and retrieving any amount of data at any time. Here's how to quickly get started with Cloud Storage using the Google Cloud Console:

1. **Open Google Cloud Console**
   - Navigate to the Google Cloud Console.
2. **Create a Cloud Storage Bucket**
   - In the Navigation Menu, go to **Storage** > **Browser**.
   - Click on **Create Bucket**.
   - Enter a unique name for your bucket (bucket names must be globally unique).
   - Choose a location for your bucket. You can select **Region**, **Multi-region**, or **Dual-region** depending on your redundancy and latency needs.
   - Choose a storage class: **Standard**, **Nearline**, **Coldline**, or **Archive**.
   - Set access controls: You can choose between **Uniform** or **Fine-grained**.
   - Click **Create**.
3. **Upload Files to the Bucket**
   - Click on the newly created bucket in the **Storage** > **Browser** section.
   - Click on the **Upload Files** button.
   - Select the file(s) from your local machine to upload.
4. **Manage Bucket Permissions**
   - Click on the **Permissions** tab.
   - You can add IAM roles to control who has access to your bucket and what operations they can perform (e.g., Viewer, Editor, Admin).
5. **Download Files from the Bucket**
   - Select a file from the bucket.
   - Click on the **Download** button to save it to your local machine.
6. **Delete the Bucket**
   - To avoid charges, delete the bucket after use. Go back to **Storage** > **Browser**, select the bucket, click **Delete**, and confirm.

b) IAM in AWS and Setup

AWS Identity and Access Management (IAM) allows you to manage users, groups, roles, and their permissions in AWS.

1. **Access AWS IAM**
   - Sign in to the AWS Management Console.
   - Go to the **IAM** service from the AWS services menu.

2. **Create a New User**
    - Click on **Users** in the left navigation pane.
    - Click **Add user**.
    - Enter a username (e.g., test-user).
    - Choose the **Access type**: **Programmatic access** (for CLI/SDK/API) or **AWS Management Console access**.
    - Set a custom password if console access is selected.
    - Click **Next: Permissions**.
3. **Assign Permissions**
    - **Add user to group**: Create a new group or select an existing one. Assign policies (like AdministratorAccess, AmazonS3FullAccess, etc.).
    - Alternatively, you can **Attach policies directly** to the user.
    - Click **Next: Tags**.
4. **Add Tags** (Optional)
    - Tags help you organize and track IAM resources. Add any key-value pairs if needed.
    - Click **Next: Review**.
5. **Review and Create User**
    - Review the user details and click **Create user**.
    - Download the **.csv** file containing the user's security credentials (Access Key ID and Secret Access Key) if programmatic access is selected.
6. **Create and Attach a Role**
    - Roles allow services or applications to assume permissions to perform actions. To create a role:
    - Go to **Roles** in the IAM console.
    - Click **Create role**.
    - Select the **AWS service** that will assume this role (e.g., EC2).
    - Attach the necessary policy (e.g., AmazonS3FullAccess).
    - Name the role and create it.
    - Attach this role to the appropriate AWS service (like an EC2 instance).
7. **Review IAM Policies and Groups**
    - Regularly review IAM policies and groups to ensure that users only have the permissions they need (principle of least privilege).

c) Cloud Functions

Cloud Functions are lightweight, single-purpose functions that respond to events in Google Cloud without the need to manage a server.

1. **Access Cloud Functions in Google Cloud Console**
    - In the Navigation Menu, go to **Cloud Functions**.
2. **Create a New Cloud Function**
    - Click on **Create Function**.
    - **Name** your function (e.g., helloWorld).
    - Select **Region** where the function will run.
    - **Trigger**: Select how the function will be triggered. You can choose between HTTP, Pub/Sub, Cloud Storage, etc.

- **Authentication**: For HTTP, you can set the function to require authentication or allow unauthenticated invocations.

3. **Write the Function Code**
   - Use the inline editor or upload code. Here's an example of a simple helloWorld function in Node.js:

     ```
     javascript
     exports.helloWorld = (req, res) => {
       res.send('Hello, World!');
     };
     ```

   - You can also specify runtime (e.g., Node.js, Python, Go).
4. **Deploy the Function**
   - Set the **Memory** and **Timeout** according to your function's needs.
   - Click **Deploy** to create the function.
5. **Test the Function**
   - Once deployed, click on the function to open its details page.
   - If it's an HTTP function, you'll see a **Trigger URL**. Visit this URL in your browser or use curl to see the function's output.
   - For other triggers, test by invoking the corresponding event (e.g., uploading a file to a bucket if using Cloud Storage).
6. **View Logs**
   - Go to the **Logs** tab to view execution logs for your function, which is useful for debugging and monitoring.
7. **Delete the Function**
   - To avoid charges, delete the function when it's no longer needed by selecting it and clicking **Delete**.

d) Cloud Networking

Cloud Networking involves setting up and managing network resources in a cloud environment. Here's an overview of foundational tasks in Google Cloud:

1. **VPC (Virtual Private Cloud) Setup**
   - In the Google Cloud Console, navigate to **VPC network** > **VPC networks**.
   - Click **Create VPC network**.
   - Provide a **Name** for the VPC.
   - Choose **Automatic** or **Custom** subnet creation.
     - **Automatic**: Google creates subnets in each region.
     - **Custom**: You manually define subnets.
   - **Firewall rules**: Set default firewall rules, such as allowing internal traffic, SSH, RDP, or ICMP.
   - Click **Create**.
2. **Creating Subnets**
   - If you chose custom subnets, click **Add subnet**.
   - Define the **Name**, **Region**, and **IP range** (CIDR block).
   - Click **Done**, then **Create**.
3. **Managing Firewall Rules**

- Go to **VPC network** > **Firewall**.
- Click **Create firewall rule**.
- Define the **Name**, **Network**, **Priority**, and **Direction** of traffic.
- Set **Action on match** to **Allow** or **Deny**.
- Define the **Targets** (all instances or specific tags) and **Source IP ranges**.
- Specify the **Protocols and ports** to allow or deny.
- Click **Create**.

4. **Setting Up VPN or Cloud Interconnect**
   - If you need to connect your on-premises network with Google Cloud:
   - Go to **Hybrid Connectivity** > **VPN** or **Cloud Interconnect**.
   - Follow the setup instructions to establish a secure connection between your networks.
5. **Load Balancing**
   - To distribute traffic across multiple instances:
   - Go to **Network services** > **Load balancing**.
   - Click **Create load balancer** and choose **HTTP(S)**, **TCP/UDP**, or **SSL Proxy** based on your needs.
   - Configure the backend services, health checks, and frontend IPs.
   - Click **Create** to deploy the load balancer.
6. **Peering and Shared VPC**
   - For advanced networking configurations, set up VPC peering to connect VPCs or Shared VPC for cross-project networking.
7. **Monitoring and Logs**
   - Use **Cloud Monitoring** and **Cloud Logging** to monitor network traffic and logs. Set up alerts for unusual traffic patterns or security issues.

Summary

These steps cover the foundational tasks you might need when working with cloud infrastructure, including storage management, IAM configuration in AWS, deploying serverless functions, and setting up cloud networking. These are essential skills for managing and operating resources in a cloud environment effectively.

Task 4:
Set Up and Configure a Cloud Environment
a) Cloud IAM: Qwik Start
b) Introduction to SQL for Big Query and Cloud SQL
c) Database Tasks in Cloud
d) Cloud Monitoring: Qwik Start
e) Managing Deployments Using Kubernetes Engine
f) Set Up and Configure a Cloud Environment in Google Cloud: Challenge Lab
Set Up and Configure a Cloud Environment

a) Cloud IAM: Qwik Start

Google Cloud Identity and Access Management (IAM) allows you to manage access to resources by defining who (identity) has what access (role) to which resource.

1. **Access the IAM Console**
    - Go to the Google Cloud Console.
    - Navigate to **IAM & Admin** > **IAM**.
2. **Add a New User**
    - Click **Add** at the top of the IAM page.
    - Enter the **Email address** of the user you want to add.
    - Select the role(s) to grant to the user. Roles define the level of access:
        - **Viewer**: Can view resources but cannot make changes.
        - **Editor**: Can view and modify resources.
        - **Owner**: Has full control over resources.
        - You can also use predefined roles like **BigQuery Admin** or **Compute Engine Admin** depending on the needs.
    - Optionally, you can assign **Custom Roles** if you have created any.
    - Click **Save** to add the user with the specified role.
3. **Manage IAM Policies**
    - IAM policies control who has what access. To view or edit the policy for a resource, go to that resource's page in the Google Cloud Console and look for the **Permissions** tab.
    - You can add, remove, or change roles for users at this level.
4. **Review and Audit Access**
    - Regularly review IAM roles and permissions to ensure that users have the necessary but minimal level of access (principle of least privilege).
    - Use the **IAM Recommendations** in the IAM dashboard to get suggestions for removing overly broad permissions.

b) Introduction to SQL for BigQuery and Cloud SQL

BigQuery is Google's fully-managed, serverless data warehouse that allows for SQL-based analysis of large datasets. **Cloud SQL** is a fully-managed relational database service for MySQL, PostgreSQL, and SQL Server.

BigQuery

1. **Access BigQuery Console**
   - In the Google Cloud Console, navigate to **BigQuery**.
2. **Create a Dataset**
   - In the BigQuery Console, click on your project name in the left panel.
   - Click on **Create dataset**.
   - Provide a **Dataset ID** and choose a data location (e.g., US or EU).
   - Set default table expiration (optional) and encryption settings.
   - Click **Create dataset**.
3. **Create a Table and Query Data**
   - Click on the dataset you created, then **Create table**.
   - Choose the source of the data (e.g., upload a file, select from Google Drive, or use a public dataset).
   - Specify the table schema if needed (you can auto-detect schema for CSV files).
   - Once the table is created, you can query it using standard SQL.
   - Example query:

     ```
     sql
     SELECT *
     FROM `project_id.dataset_id.table_id`
     WHERE some_column = 'value';
     ```

4. **Running Queries**
   - Type your SQL query into the query editor and click **Run**.
   - View the results in the results pane.

Cloud SQL

1. **Create a Cloud SQL Instance**
   - In the Google Cloud Console, navigate to **SQL**.
   - Click **Create instance**.
   - Choose your database engine: **MySQL**, **PostgreSQL**, or **SQL Server**.
   - Provide an instance ID, set a root password, and configure other settings like region, machine type, and storage.
   - Click **Create**.
2. **Connect to the Cloud SQL Instance**
   - Once the instance is created, you can connect using:
     - **Cloud SQL Auth Proxy**: A secure way to connect your application.
     - **Public IP**: Whitelist your IP address and use standard MySQL or PostgreSQL clients.
     - **Private IP**: Connect securely within a VPC.
3. **Import Data**
   - In the SQL instance, go to **Databases** > **Import**.
   - Select a SQL dump file from Google Cloud Storage and import it into your database.
4. **Run SQL Queries**
   - Use the built-in **Query Editor** in the Cloud SQL console to run SQL queries.
   - Example query:

```
sql
SELECT * FROM my_table WHERE column_name = 'value';
```

c) Database Tasks in Cloud

Managing databases in the cloud involves tasks like setting up, backing up, restoring, and scaling databases.

1. **Backup and Restore in Cloud SQL**
   - **Automated Backups**: Set up automated backups by going to the instance settings, selecting **Backups**, and enabling automated backups.
   - **Manual Backups**: To create a manual backup, go to the **Backups** tab and click **Create backup**.
   - **Restore from Backup**: To restore, go to the **Backups** tab, select a backup, and click **Restore**.
2. **Scaling Cloud SQL**
   - **Vertical Scaling**: Increase machine type or storage by editing the instance. This may require downtime.
   - **Horizontal Scaling**: For MySQL, you can set up a read replica to distribute read traffic.
3. **Monitoring and Alerts**
   - Use **Cloud Monitoring** to set up alerts for performance metrics such as CPU usage, memory usage, and disk space.
   - You can also set up custom monitoring for query performance or specific database metrics.

d) Cloud Monitoring: Qwik Start

Cloud Monitoring provides visibility into the performance, uptime, and overall health of your cloud infrastructure.

1. **Access Cloud Monitoring**
   - In the Google Cloud Console, navigate to **Monitoring**.
2. **Create a Workspace**
   - If you don't have a workspace, Cloud Monitoring will prompt you to create one. A workspace is a container for monitoring and logging information.
   - Link your Google Cloud project to the workspace.
3. **Set Up Monitoring Dashboards**
   - Go to **Dashboards** in the Monitoring menu.
   - Click **Create dashboard**.
   - Add charts to monitor resources like VM instances, Cloud SQL, or Kubernetes Engine by selecting **Add Chart**.
   - Customize the charts to display metrics like CPU utilization, memory usage, disk I/O, etc.
4. **Set Up Alerts**
   - Go to **Alerting** in the Monitoring menu.
   - Click **Create Policy**.

- Choose a **Condition** to monitor (e.g., CPU usage exceeds 80% for more than 5 minutes).
- Define **Notifications** by adding email, SMS, or webhook notifications.
- Set a **Name** for the policy and save it.

5. **View Logs**
   - Use **Cloud Logging** (formerly Stackdriver Logging) to view logs generated by your cloud resources.
   - Navigate to **Logging** in the Monitoring menu, select a resource, and view the logs in real-time.

e) Managing Deployments Using Kubernetes Engine

Google Kubernetes Engine (GKE) is a managed Kubernetes service for running containerized applications.

1. **Create a Kubernetes Cluster**
   - In the Google Cloud Console, navigate to **Kubernetes Engine** > **Clusters**.
   - Click **Create Cluster**.
   - Choose a **Standard** or **Autopilot** cluster depending on your needs.
   - Configure the cluster settings such as the number of nodes, machine type, and networking options.
   - Click **Create** to provision the cluster.
2. **Deploy an Application**
   - Once the cluster is ready, connect to it using the **kubectl** CLI.
   - Deploy an application using a YAML file or directly through the command line:

     bash
     kubectl create deployment nginx --image=nginx

   - Expose the deployment to the internet:

     bash
     kubectl expose deployment nginx --type=LoadBalancer --port 80

3. **Managing Deployments**
   - **Scaling**: Scale your deployment to handle more traffic:

     bash
     kubectl scale deployment nginx --replicas=3

   - **Updating**: To update the application version, modify the image in the deployment:

     bash
     kubectl set image deployment/nginx nginx=nginx:1.19.0

4. **Monitoring and Logging**
   - Use **Cloud Monitoring** and **Cloud Logging** to monitor the health of your Kubernetes clusters and view logs from your pods.

- Set up alerts for high CPU or memory usage, or when pods crash.

f) Set Up and Configure a Cloud Environment in Google Cloud: Challenge Lab

The **Challenge Lab** for setting up and configuring a cloud environment will test your ability to perform various tasks using Google Cloud services.

1. **Start the Challenge Lab**
   - Go to the Google Cloud Skills Boost platform and start the Challenge Lab.
2. **Understand the Requirements**
   - The lab will have specific requirements, such as setting up IAM roles, configuring Cloud SQL, deploying applications on GKE, and more. Read each task carefully.
3. **Complete the Tasks**
   - Set up the environment as instructed. This may include:
     - Creating and managing IAM roles and permissions.
     - Setting up Cloud SQL instances and running queries.
     - Deploying applications using Kubernetes Engine.
     - Configuring Cloud Monitoring dashboards and alerts.
     - Managing cloud storage and networking configurations.
4. **Verification and Submission**
   - Most Challenge Labs include verification steps to check if you've completed tasks correctly. Use the provided commands

Task 5:

1. Introduction to Amazon EC2
2. Introduction to Amazon Simple Storage Service (S3)

AWS Essentials

1. Introduction to Amazon EC2

Amazon EC2 (Elastic Compute Cloud) is a web service that provides resizable compute capacity in the cloud, making web-scale cloud computing easier for developers.

Steps to Get Started with Amazon EC2:

1. **Access the AWS Management Console**
   - Sign in to the AWS Management Console.
   - Navigate to **EC2** under the "Compute" section.
2. **Launch an EC2 Instance**
   - Click on **Launch Instance**.
   - **Choose an Amazon Machine Image (AMI)**:
     - Select an AMI from the list. The AMI includes the operating system and applications pre-installed.
     - Popular choices include Amazon Linux 2, Ubuntu, and Windows Server.
   - **Choose an Instance Type**:
     - Select an instance type based on your needs. Common types include t2.micro (which is free-tier eligible), m5.large, etc.
     - Click **Next: Configure Instance Details**.
   - **Configure Instance Details**:
     - Set the number of instances, network settings, IAM roles, and other advanced options.
     - Click **Next: Add Storage**.
   - **Add Storage**:
     - Define the size and type of storage (EBS volumes) to attach to your instance. The default is usually sufficient for basic setups.
     - Click **Next: Add Tags**.
   - **Add Tags**:
     - Optionally, add tags in key-value pairs to help manage and identify your instance.
     - Click **Next: Configure Security Group**.
   - **Configure Security Group**:
     - A security group acts as a virtual firewall. Set rules to control traffic to your instance.
     - For basic setups, add an SSH rule to allow access from your IP.
     - Click **Review and Launch**.
   - **Review and Launch**:
     - Review your instance settings and click **Launch**.
     - You'll be prompted to create or select an SSH key pair for secure access to the instance.
     - Choose **Create a new key pair** or use an existing one, then click **Launch Instances**.

3. **Connect to Your EC2 Instance**
    - Once the instance is running, click **View Instances** to see your instance details.
    - To connect via SSH:
        - Select your instance, click **Connect**.
        - Follow the instructions provided, which typically involve using a command like:

          bash
          ssh -i /path/to/key-pair.pem ec2-user@your-ec2-public-dns

    - For Windows instances, you'll connect using RDP (Remote Desktop Protocol).
4. **Manage and Monitor the Instance**
    - Use the **Instance State** actions to start, stop, reboot, or terminate the instance.
    - Monitor your instance's performance via **CloudWatch** metrics (CPU usage, network traffic, etc.).
5. **Terminate the Instance**
    - To avoid charges, terminate the instance when you no longer need it.
    - Select the instance, click **Instance State** > **Terminate**.

2. Introduction to Amazon Simple Storage Service (S3)

Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance.

Steps to Get Started with Amazon S3:

1. **Access the AWS Management Console**
    - Sign in to the AWS Management Console.
    - Navigate to **S3** under the "Storage" section.
2. **Create an S3 Bucket**
    - Click on **Create bucket**.
    - **Bucket Name**: Enter a globally unique name for your bucket.
    - **Region**: Choose an AWS Region where the bucket will reside.
    - **Bucket Settings**: Configure settings like versioning, server access logging, and encryption. These can be adjusted later if necessary.
    - **Block Public Access**: By default, S3 buckets block public access. This setting can be changed based on your needs.
    - Click **Create Bucket**.
3. **Upload Objects to S3**
    - Click on your newly created bucket.
    - Click **Upload**.
    - Drag and drop files or use the file picker to select files from your local machine.
    - Set object properties like storage class and encryption if needed.
    - Click **Upload** to store the files in S3.
4. **Manage Bucket Permissions**
    - Go to the **Permissions** tab in your bucket.
    - You can manage access using bucket policies, access control lists (ACLs), and IAM roles.

- To make files public, you can change the bucket policy or set specific permissions on individual objects.

5. **Enable Versioning**
   - Versioning helps keep multiple versions of an object in one bucket. To enable it:
   - Go to your bucket settings, and under **Properties**, find **Versioning**.
   - Click **Edit** and enable versioning.
6. **Set Up Lifecycle Policies**
   - Lifecycle policies allow you to automatically transition objects between storage classes or delete them after a certain period.
   - Go to the **Management** tab, and click **Create lifecycle rule**.
   - Define the rule to transition objects to cheaper storage classes like **Glacier** or delete them after a set time.
7. **Access Objects in S3**
   - You can access objects via the AWS S3 Console or programmatically using the AWS SDKs or CLI.
   - To generate a pre-signed URL for temporary access:

     bash
     aws s3 presign s3://bucket-name/object-key --expires-in 3600

8. **Delete S3 Bucket**
   - To delete a bucket, first delete all objects within it.
   - Go back to the **Bucket List**, select the bucket, and click **Delete**.
   - Confirm the deletion by typing the bucket name.

These steps should help you get started with EC2 and S3, two of the most foundational services in AWS. They are widely used and provide the building blocks for most applications and services deployed on AWS.

Task 6:

1. Introduction to Amazon Relational Database Service (RDS) - SQL Server)
2. AWS Identity and Access Management (IAM) Task

1. Introduction to Amazon Relational Database Service (RDS) - SQL Server

Amazon RDS (Relational Database Service) makes it easy to set up, operate, and scale a relational database in the cloud. With RDS, you can run several database engines, including SQL Server, MySQL, PostgreSQL, Oracle, and MariaDB.

Steps to Get Started with Amazon RDS - SQL Server:

1. **Access the AWS Management Console**
   - Sign in to the AWS Management Console.
   - Navigate to **RDS** under the "Database" section.
2. **Create a SQL Server RDS Instance**
   - Click on **Create database**.
   - **Database Creation Method**:
     - Choose either **Standard Create** for detailed setup options or **Easy Create** for basic configurations.
   - **Engine Options**:
     - Select **Microsoft SQL Server** from the list of database engines.
     - Choose the specific SQL Server edition (e.g., Express, Web, Standard, or Enterprise).
   - **Version**:
     - Select the version of SQL Server you want to use (e.g., SQL Server 2019).
   - **Instance Specifications**:
     - Choose the **DB instance class** based on your performance needs (e.g., db.t3.micro for a free-tier eligible instance or db.m5.large for higher performance).
   - **Storage**:
     - Set the allocated storage size. You can also enable **storage autoscaling** to allow the database to grow automatically as needed.
   - **Database Settings**:
     - Set the **DB Instance Identifier** (name of your database instance).
     - Define the **Master Username** and **Master Password**.
   - **VPC and Network Settings**:
     - Select the VPC, subnet, and security group settings that will govern network access to your instance.
   - **Backup and Maintenance**:
     - Configure automated backups, backup retention period, and the preferred backup window.
     - Set up **maintenance window** if you want to specify a particular time for updates and patches.
   - Click **Create database** to start the creation process.
3. **Connect to the SQL Server RDS Instance**
   - Once the RDS instance is available (this may take a few minutes), you can connect to it using SQL Server Management Studio (SSMS) or another SQL client.
   - **Endpoint**:

        - Find the **endpoint** and **port** in the RDS console under the **Connectivity & security** tab.
        - Use these details in your SQL client to connect.
    - **Connect via SSMS**:
        - Open SSMS, enter the RDS endpoint as the **Server name**.
        - Choose **SQL Server Authentication** and enter the master username and password.
4. **Manage and Monitor the RDS Instance**
    - Use the **RDS Dashboard** to monitor the instance's performance, including CPU usage, memory, and storage.
    - Configure **CloudWatch Alarms** to alert you if the database goes beyond certain thresholds (e.g., high CPU usage or low available storage).
    - Perform administrative tasks like **creating snapshots** (manual backups), restoring backups, or scaling the instance size.
5. **Backup and Restore**
    - **Automated Backups**: Ensure automated backups are enabled. You can configure the backup retention period.
    - **Manual Snapshots**: Create a manual snapshot to capture the current state of your database. This snapshot can be used later for restoring or replicating the database.
    - **Restore a Database**: To restore, go to the **Snapshots** tab, select the snapshot, and choose **Restore Snapshot**.
6. **Security and Access Control**
    - Use **AWS IAM** to control who can manage the RDS instances.
    - Ensure **encryption** is enabled for data at rest by using AWS-managed keys.
    - Use **SSL/TLS** to secure data in transit between your application and the RDS instance.
7. **Terminate the RDS Instance**
    - When you no longer need the RDS instance, terminate it to avoid charges.
    - Go to the **RDS Dashboard**, select the instance, and click **Delete**. You can choose to retain or delete the final snapshot during the deletion process.

2. AWS Identity and Access Management (IAM) Task

AWS Identity and Access Management (IAM) enables you to securely control access to AWS services and resources for your users.

Steps to Set Up and Use AWS IAM:

1. **Access the AWS Management Console**
    - Sign in to the AWS Management Console.
    - Navigate to **IAM** under the "Security, Identity, & Compliance" section.
2. **Create IAM Users**
    - Click on **Users** in the IAM dashboard.
    - Click **Add user**.
    - **User Details**:
        - Enter a username.
        - Choose **AWS Management Console access** if the user needs to log into the console.

        - Optionally, enable **Programmatic access** if the user needs access via the AWS CLI, SDKs, or APIs.
    - **Set Permissions**:
        - Choose to **Attach existing policies directly** to assign predefined AWS policies (e.g., AdministratorAccess, PowerUserAccess, S3FullAccess).
        - Alternatively, use **Add user to group** to assign users to an IAM group with predefined permissions.
        - You can also use **Attach custom policies** if you need fine-grained access control.
    - **Review and Create**:
        - Review the settings and click **Create user**.
    - If console access is granted, download the **CSV** file containing the login credentials or send the credentials directly to the user.
3. **Create IAM Roles**
    - IAM roles allow you to delegate permissions to AWS services or users from other AWS accounts.
    - In the IAM dashboard, click on **Roles** > **Create role**.
    - **Select Trusted Entity**:
        - Choose the AWS service (e.g., EC2, Lambda) or another AWS account that will assume this role.
    - **Attach Permissions**:
        - Attach policies that define the role's permissions (e.g., AmazonS3FullAccess, AmazonRDSFullAccess).
    - **Role Name**:
        - Give the role a name and optional description, then create the role.
4. **Create IAM Groups**
    - Use IAM groups to manage permissions for multiple users easily.
    - In the IAM dashboard, click **Groups** > **Create New Group**.
    - **Group Name**: Enter a name for the group.
    - **Attach Policies**: Attach one or more policies to the group.
    - **Add Users to the Group**: Assign users to the group to grant them the associated permissions.
5. **Set Up Multi-Factor Authentication (MFA)**
    - MFA adds an extra layer of security to your AWS accounts.
    - In the IAM dashboard, select **Users** and click on a user.
    - Under the **Security credentials** tab, click **Assign MFA device**.
    - Choose the type of MFA device (virtual, U2F security key, or hardware).
    - Follow the on-screen instructions to configure the device.
6. **Manage IAM Policies**
    - **Policies** define permissions. They can be attached to users, groups, or roles.
    - In the IAM dashboard, click **Policies** > **Create policy**.
    - **Policy Editor**: Use the **Visual editor** or **JSON editor** to define the policy.
        - Example JSON policy allowing read-only access to S3:

```
json
{
 "Version": "2012-10-17",
 "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": "s3:Get*",
      "Resource": "*"
    }
  ]
}
```

   - **Policy Name**: Name the policy and attach it to users, groups, or roles as needed.
7. **Monitor and Audit IAM Activities**
   - Use **AWS CloudTrail** to monitor and log IAM activities such as user logins, permission changes, and role assumptions.
   - Regularly review IAM policies and permissions to ensure compliance with the principle of least privilege.
   - Implement **AWS Config** rules to check IAM configurations against best practices.
8. **Set Up IAM Access Analyzer**
   - IAM Access Analyzer helps you identify resources that are shared with an external entity.
   - In the IAM console, go to **Access Analyzer** and create an analyzer.
   - Review findings to understand which resources (e.g., S3 buckets, IAM roles) are accessible outside your account.

These steps will guide you through setting up and managing an RDS instance for SQL Server as well as configuring IAM to secure and manage access to your AWS resources.

Task 7:

1. Management of Amazon Elastic Container Service
2. Hosting a static website and Deploying a web application on AWS

To accomplish Task 7, which involves the management of Amazon Elastic Container Service (ECS) and hosting/deploying a web application on AWS, follow these steps:

1. Management of Amazon Elastic Container Service (ECS)

Amazon ECS Overview: Amazon ECS is a fully managed container orchestration service that makes it easy to deploy, manage, and scale containerized applications using Docker. It supports both Fargate and EC2 launch types.

Steps to Manage ECS:

1. **Create an ECS Cluster:**
    - Navigate to the ECS console.
    - Click on "Create Cluster".
    - Choose between the "EC2 Linux + Networking" or "Fargate" launch type, depending on your needs.
    - Configure the cluster with a name, number of instances (if using EC2), instance types, and networking settings.
    - Create the cluster.
2. **Create a Task Definition:**
    - In the ECS console, go to "Task Definitions".
    - Click "Create new Task Definition".
    - Choose between the "Fargate" or "EC2" launch type.
    - Define the task by configuring the container details such as image (from Docker Hub or ECR), memory, CPU, environment variables, port mappings, and log configurations.
    - Save the task definition.
3. **Deploy a Service:**
    - Go to the ECS cluster you created.
    - Click "Create" under Services.
    - Select the task definition created earlier.
    - Define the service name, number of tasks, and deployment type (e.g., rolling update).
    - Configure load balancer settings if needed (optional).
    - Deploy the service.
4. **Monitor and Scale the Service:**
    - Monitor the service and task using CloudWatch and ECS console metrics.
    - Scale up or down by adjusting the desired number of tasks in the ECS service.
5. **Update ECS Services:**
    - To update a running service (e.g., new image version), create a new revision of the task definition and update the service with the new task definition.

2. Hosting a Static Website and Deploying a Web Application on AWS

A. Hosting a Static Website on Amazon S3:

1. **Create an S3 Bucket:**
    - Go to the S3 console.
    - Create a new bucket with a globally unique name.
    - Set the region according to your preference.
2. **Upload Website Files:**

- Upload your static website files (HTML, CSS, JS) to the bucket.

3. **Configure the Bucket for Website Hosting:**
   - Select the bucket and navigate to "Properties".
   - Enable "Static website hosting".
   - Provide the index document (e.g., index.html) and error document (optional).
4. **Make the Bucket Public:**
   - Go to "Permissions" and update the bucket policy to allow public read access to the objects.
   - Example bucket policy:

     ```
     json
     {
      "Version": "2012-10-17",
      "Statement": [
       {
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::your-bucket-name/*"
       }
      ]
     }
     ```

5. **Access the Website:**
   - Use the website endpoint provided in the S3 bucket properties to access your static site.

B. Deploying a Web Application on AWS (Using EC2 or Elastic Beanstalk):

Option 1: Deploying with EC2:

1. **Launch an EC2 Instance:**
   - Navigate to the EC2 console and launch a new instance.
   - Choose an Amazon Machine Image (AMI), such as Amazon Linux 2.
   - Select the instance type and configure instance details (VPC, subnets, etc.).
   - Configure security groups to allow HTTP/HTTPS traffic.
   - Launch the instance.
2. **Install Web Server:**
   - SSH into the EC2 instance.
   - Install necessary software (e.g., Apache, Nginx).

     ```
     bash
     sudo yum update -y
     sudo yum install httpd -y
     sudo systemctl start httpd
     sudo systemctl enable httpd
     ```

   - Place your web application files in the web server's root directory.

3. **Access the Application:**
    - Use the public IP or DNS of the EC2 instance to access the deployed application.

Option 2: Deploying with Elastic Beanstalk:

1. **Create an Elastic Beanstalk Application:**
    - Go to the Elastic Beanstalk console.
    - Click on "Create New Application".
    - Provide a name and create an environment (e.g., Web server environment).
2. **Upload and Deploy the Application:**
    - Choose a platform (e.g., Node.js, Python, Java).
    - Upload your application code (ZIP file) and deploy.
    - Elastic Beanstalk will handle provisioning and scaling automatically.
3. **Manage the Application:**
    - Monitor the application health and logs using the Elastic Beanstalk dashboard.
    - Make updates to your application by deploying new versions.

These steps should cover the management of ECS and the hosting/deployment of web applications on AWS effectively.

Task 8:
a. Continuously Querying top 10 songs in the song-list or chart with Kubernetes
b. Key Parameter Indicators visualization for Airline services using Kubernetes

For Task 8, which involves continuously querying the top 10 songs in a song-list or chart and visualizing key parameter indicators for airline services using Kubernetes, here's how you can approach each part:

a. Continuously Querying Top 10 Songs in the Song-List or Chart with Kubernetes

Overview: Kubernetes can be used to run a service that continuously queries the top 10 songs from a music chart API and stores or processes that data. This can involve setting up a cron job or a continuous polling mechanism within a Kubernetes cluster.

Steps to Implement:

1. **Set Up a Kubernetes Cluster:**
    - You can create a Kubernetes cluster using a cloud provider like GKE (Google Kubernetes Engine), EKS (Amazon Elastic Kubernetes Service), or AKS (Azure Kubernetes Service). Alternatively, use a local setup with Minikube.
2. **Develop the Querying Service:**
    - Write a small application or script (e.g., using Python) that queries the top 10 songs from a music chart API (such as Billboard or a similar service).
    - The script should handle the API request, parse the response, and potentially store the results in a database like MySQL, PostgreSQL, or a file storage service like S3.
3. **Containerize the Application:**

- Create a Dockerfile for your application to containerize it.

  Dockerfile
  ```
  FROM python:3.9-slim
  WORKDIR /app
  COPY . /app
  RUN pip install requests
  CMD ["python", "query_songs.py"]
  ```

- Build and push the Docker image to a container registry like Docker Hub or ECR.

4. **Create a Kubernetes Deployment:**
   - Define a Kubernetes Deployment YAML file for your application.
   - Example:

     yaml
     ```
     apiVersion: apps/v1
     kind: Deployment
     metadata:
       name: song-query-deployment
     spec:
       replicas: 1
       selector:
         matchLabels:
           app: song-query
       template:
         metadata:
           labels:
             app: song-query
         spec:
           containers:
           - name: song-query
             image: your-docker-image:latest
             ports:
             - containerPort: 80
     ```

5. **Set Up a CronJob (Optional):**
   - If you want the querying to happen periodically, use a Kubernetes CronJob.
   - Example:

     yaml
     ```
     apiVersion: batch/v1
     kind: CronJob
     metadata:
       name: song-query-cron
     spec:
       schedule: "*/5 * * * *" # Every 5 minutes
       jobTemplate:
         spec:
     ```

```
    template:
      spec:
        containers:
        - name: song-query
          image: your-docker-image:latest
        restartPolicy: OnFailure
```

6. **Deploy to Kubernetes:**
   - Apply the Deployment or CronJob to your Kubernetes cluster using kubectl apply -f <your-yaml-file>.yaml.
   - The application will now continuously query the top 10 songs at the interval specified.

b. Key Parameter Indicators Visualization for Airline Services using Kubernetes

Overview: For visualizing key parameter indicators (KPIs) for airline services using Kubernetes, you'll likely need to deploy a system that collects, processes, and visualizes these indicators in real-time. This can involve using tools like Prometheus for metrics collection, Grafana for visualization, and Kubernetes for orchestrating the services.

Steps to Implement:

1. **Set Up a Kubernetes Cluster:**
   - As in part a, set up a Kubernetes cluster using your preferred method.
2. **Deploy Prometheus for Metrics Collection:**
   - Prometheus can be used to collect and aggregate metrics from various sources. You can either deploy Prometheus directly on Kubernetes or use the Prometheus Operator for easier management.
   - Example Prometheus deployment:

```
yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'airline-service-metrics'
        static_configs:
        - targets: ['airline-service:8080']
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-deployment
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
      - name: prometheus
        image: prom/prometheus
        volumeMounts:
        - name: config-volume
          mountPath: /etc/prometheus/
```

3. **Deploy Grafana for Visualization:**
   - Grafana can be used to create dashboards that visualize the KPIs.
   - Deploy Grafana in your Kubernetes cluster.
   - Example Grafana deployment:

```
yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
      - name: grafana
        image: grafana/grafana
        ports:
        - containerPort: 3000
```

4. **Collect Airline KPI Data:**
   - Develop an airline service that exposes metrics such as on-time performance, delays, customer satisfaction, etc., in a format that Prometheus can scrape.
   - Implement the service in a containerized application and deploy it on Kubernetes.
5. **Create Grafana Dashboards:**

- Access the Grafana UI (usually via port 3000 on the service's endpoint).
- Add Prometheus as a data source.
- Create dashboards to visualize the KPIs, using graphs, tables, and other visualization widgets.

6. **Deploy Everything to Kubernetes:**
   - Deploy the Prometheus, Grafana, and airline services to your Kubernetes cluster using kubectl apply -f <your-yaml-files>.yaml.
   - Access the Grafana dashboard to view real-time visualizations of the KPIs for the airline services.

By following these steps, you can set up a Kubernetes-based system that continuously queries song charts and visualizes airline service KPIs.

TASK 9:

a. Managing resources using Terraform

b. creating and running containers.

To accomplish Task 9, which involves managing resources using Terraform and creating and running containers, here's a detailed guide for each part:

a. Managing Resources Using Terraform

Overview: Terraform is an open-source Infrastructure as Code (IaC) tool that allows you to define and provision infrastructure using a high-level configuration language. Terraform can manage resources across various cloud providers like AWS, Azure, GCP, and others.

Steps to Manage Resources Using Terraform:

1. **Install Terraform:**
   - Download and install Terraform from the official website.
   - Verify the installation:

     bash
     terraform --version

2. **Create a Terraform Configuration File:**
   - Define your infrastructure in a .tf file. For example, to create an EC2 instance in AWS:

     hcl
     provider "aws" {
       region = "us-west-2"
     }

     resource "aws_instance" "example" {
       ami   = "ami-0c55b159cbfafe1f0"
       instance_type = "t2.micro"

       tags = {
         Name = "ExampleInstance"
       }
     }

   - This example sets up an AWS provider and defines a single EC2 instance.
3. **Initialize Terraform:**
   - Run the following command to initialize Terraform. This command downloads the required provider plugins:

     bash
     terraform init

4. **Preview the Changes:**
   - Before applying the changes, you can preview what Terraform will do:

     bash
     terraform plan

5. **Apply the Configuration:**

- To create the resources defined in your configuration file, run:

  bash
  terraform apply

- Terraform will prompt you to confirm the action. Type yes to proceed.

6. **Managing Resources:**
   - Terraform can be used to manage the entire lifecycle of your infrastructure, including updates and deletions.
   - To update resources, modify the .tf file and run terraform apply again.
   - To delete resources, use:

     bash
     terraform destroy

7. **Version Control and Collaboration:**
   - Store your Terraform configuration files in a version control system like Git.
   - Use Terraform Cloud or a similar service for remote state management and collaboration.

b. Creating and Running Containers

Overview: Creating and running containers typically involves using Docker, the most popular containerization platform. Docker allows you to package an application and its dependencies into a container that can run consistently across different environments.

Steps to Create and Run Containers:

1. **Install Docker:**
   - Install Docker by following the instructions on the Docker website.
   - Verify the installation:

     bash
     docker --version

2. **Create a Dockerfile:**
   - A Dockerfile is a text document that contains all the commands to assemble a container image.
   - Example Dockerfile for a simple Node.js application:

     Dockerfile
     # Use an official Node.js runtime as a parent image
     FROM node:14

     # Set the working directory in the container
     WORKDIR /usr/src/app

     # Copy the local files to the container

```
COPY package*.json ./

# Install any needed dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Make port 8080 available to the world outside this container
EXPOSE 8080

# Run the app when the container launches
CMD ["node", "app.js"]
```

3. **Build the Docker Image:**
   - Navigate to the directory containing the Dockerfile and run:

     ```
     bash
     docker build -t my-node-app .
     ```

   - This command builds the Docker image and tags it as my-node-app.
4. **Run the Docker Container:**
   - Use the following command to run the container:

     ```
     bash
     docker run -p 8080:8080 my-node-app
     ```

   - The -p 8080:8080 flag maps port 8080 on your local machine to port 8080 in the container.
5. **Manage Containers:**
   - To list running containers:

     ```
     bash
     docker ps
     ```

   - To stop a container:

     ```
     bash
     docker stop <container_id>
     ```

   - To remove a container:

     ```
     bash
     docker rm <container_id>
     ```

   - To remove an image:

     ```
     bash
     ```

```
docker rmi <image_id>
```

6. **Using Docker Compose (Optional):**
   - For complex applications with multiple services, use Docker Compose to manage multi-container Docker applications.
   - Example docker-compose.yml file:

     ```
     yaml
     version: '3'
     services:
       web:
         image: my-node-app
         ports:
           - "8080:8080"
       redis:
         image: "redis:alpine"
     ```

   - Start the services with:

     ```
     bash
     docker-compose up
     ```

By following these steps, you can effectively manage cloud resources using Terraform and create, manage, and run containerized applications using Docker.