**Deploying and Managing a Scalable Tour Planner Web Application**

**University of Canada West**

Shilpi Kumari

**Table of Contents**

[**1.** **Introduction** 2](#_Toc193221970)

[**1.2 Project Objectives:** 3](#_Toc193221971)

[**1.3 Key Features of the Deployment:** 3](#_Toc193221972)

[**2.** **Components Required for Hosting the Web Application** 4](#_Toc193221973)

[**3.** **Creating VPC** 5](#_Toc193221974)

[**4.** **EC2 Instance** 9](#_Toc193221975)

[**5.** **RDS MySQL Database** 13](#_Toc193221976)

[**6.** **Amazon S3 Bucket** 22](#_Toc193221977)

[**7.** **Testing the Web Application** 23](#_Toc193221978)

[**8.** **CloudWatch Monitoring** 27](#_Toc193221979)

[**9.** **Load Balancer** 27](#_Toc193221980)

[**10.** **Architecture of the Web Application Deployment** 29](#_Toc193221981)

[**11. Conclusion** 29](#_Toc193221982)

# **Introduction**

The digital landscape demands businesses to implement web applications which offer high availability together with scalability as well as affordable operation for growing user populations. The goal of this project involves creating and deploying a Tour Planner Web Application through AWS cloud infrastructure for reliable operation with scalable features and enhanced security but maintaining operational efficiency.

The **Tour Planner Web Application** is a dynamic web-based platform that allows users to browse, book, and manage travel experiences seamlessly. To achieve a robust and scalable deployment, this project implements a **load-balanced architecture** with **auto-scaling**, **database integration**, and **monitoring solutions** for effective management.

## **1.2 Project Objectives:**

* **Deploy a fault-tolerant web application** using AWS services.
* **Ensure high availability and scalability** by implementing **Load Balancing and Auto Scaling**.
* **Optimize costs** while maintaining security and reliability.
* **Integrate a relational database (Amazon RDS)** for storing tour and booking information.
* **Implement Cloud Monitoring** using **AWS CloudWatch** for real-time application health tracking.

## **1.3 Key Features of the Deployment:**

1. **AWS EC2**: Hosts the backend server for processing user requests.
2. **AWS RDS (MySQL)**: Stores structured data such as tours, users, and bookings.
3. **Application Load Balancer (ALB)**: Distributes traffic across multiple EC2 instances for **better performance**.
4. **Auto Scaling**: Dynamically scales instances based on traffic demand.
5. **Amazon S3**: Used for **storing static assets** (images, tour-related content).
6. **CloudWatch Monitoring**: Tracks application performance, logs, and alerts.

This deployment ensures that the **Tour Planner Web Application** can handle varying traffic loads while maintaining optimal performance and security. By leveraging AWS best practices, the system will be **cost-effective**, **easily maintainable**, and **highly available** for users.

# **Components Required for Hosting the Web Application**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **AWS Service Used** | **Justification** | **Cost Estimate** |
| **Web Server** | EC2 Instance (t2.micro) | Provides a virtual server to host the frontend and backend application | Free-tier eligible for 750 hours/month, otherwise ~$8/month |
| **Database Server** | Amazon RDS (MySQL, db.t3.micro) | Provides a managed database service with automatic backups | Free-tier for 750 hours/month, otherwise ~$15/month |
| **Storage** | Amazon S3 | Used to store tour images and static content | ~$0.023 per GB |
| **Networking** | Security Groups, VPC, Subnets | Ensures secure communication between services | Free |
| **Security Measures** | IAM Roles & Policies, Security Groups | Controls access to AWS resources | Free |
| **Load Balancer** | Application Load Balancer (ALB) | Distributes incoming traffic across multiple instances for high availability and fault tolerance | ~$18/month (based on usage) |
| **Monitoring** | Amazon CloudWatch, AWS CloudTrail | Monitors performance, logs system activities, and sets up alerts for critical issues | CloudWatch: Free-tier includes basic metrics, otherwise ~$2/month per alarm |

# **Creating VPC**

|  |  |
| --- | --- |
| Step 3.1 | VPC (Virtual Private Cloud) creation |
| Settings and Parameters | Name tag → TourPlanner-VPC  IPv4 CIDR block → 10.0.0.0/16 |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 3.2 | Subnets created |
| Settings and Parameters | We needed 4 subnets in two Availability Zones:   * 2 Public Subnets (for EC2 & Load Balancer) * 2 Private Subnets (for RDS Database)   Name → TourPlanner-Public-Subnet-1 & TourPlanner-Public-Subnet-2  VPC ID → Select TourPlanner-VPC  Availability Zone → us-east-1a & us-east-1b  IPv4 CIDR → 10.0.1.0/24 & 10.0.2.0/24  Name → TourPlanner-Private-Subnet-1 & TourPlanner-Private-Subnet-2  Availability Zone → us-east-1a & us-east-1b  IPv4 CIDR → 10.0.3.0/24 & 10.0.4.0/24 |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 3.3 | Created and Attached an Internet Gateway |
| Settings and Parameters | TourPlanner-IGW |
| Screenshot |  |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 3.4 | Created a Public Route Table |
| Settings and Parameters | TourPlanner-Public-RT |
| Screenshot |  |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 3.5 | Added Internet Access to the Route Table |
| Settings and Parameters | **Destination** → 0.0.0.0/0 (Allows Internet traffic)  **Target** → Select TourPlanner-IGW |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 3.6 | Associated Public Subnets with Public Route Table |
| Settings and Parameters | -- |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 3.7 | Added a NAT Gateway for Private Subnets |
| Settings and Parameters | -- |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nil |

# **EC2 Instance**

|  |  |
| --- | --- |
| Step 4.1 | Create and Launch EC2 Instance and security group |
| Settings and Parameters | **Name** → TourPlanner-WebServer  **AMI (Amazon Machine Image)** → **Amazon Linux 2**  **Instance Type** → t3.micro (Free Tier eligible)  **Key Pair** → TourPlanner-Key  **Network Settings**:  **VPC** → TourPlanner-VPC  **Subnet** → **TourPlanner-Public-Subnet-1**  **Auto-assign Public IP** → **Enable** |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nill |

|  |  |
| --- | --- |
| Step 4.2 | Create Security group |
| Settings and Parameters | **Name** → TourPlanner-SG  Rules:  Allow **HTTP (80)** from **anywhere (0.0.0.0/0)**.  Allow **HTTPS (443)** from **anywhere (0.0.0.0/0)**.  Allow **SSH (22)** from **your IP only** (for security).  **Storage** → Default **8GB EBS Volume**. |
| Screenshot |  |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 4.3 | Connect to EC2 instance |
| Settings and Parameters | -- |
| Screenshot |  |
| Challenges encountered | A screenshot of a computer  AI-generated content may be incorrect.  While trying to connect to the EC2 instance, we faced the above problem. Through the following debugging we were able to resolve the issue and launch the instance.  Troubleshooting  Checked EC2 Instance State – to confirm that the instance is running  Checked EC2 Status Checks - **Checked if both checks passed**:   * **Instance Reachability** ✅ * **System Reachability** ✅   Verified Route Table & Subnet Configuration - Clicked **Routes** and **confirmed**:   * **Destination:** 0.0.0.0/0 * **Target:** igw-0a4b4232d0e796ad8 (**Internet Gateway**).   Checked Security Group Rules - **Inbound Rules** and ensure we have:   * **Rule 1: SSH (22)** → **Source:** 0.0.0.0/0 (or your IP). * **Rule 2: HTTP (80)** → **Source:** 0.0.0.0/0. * **Rule 3: HTTPS (443)** → **Source:** 0.0.0.0/0.   Checked VPC Network ACL Rules - **Check Inbound and Outbound Rules**:   * + **Inbound:** Allow **SSH (22), HTTP (80), HTTPS (443) from 0.0.0.0/0**.   + **Outbound:** Allow **all traffic (0.0.0.0/0)**.   ✅ **If any rule is blocking SSH (22), edit and allow it**.  Problem Identified: Network ACL is Blocking All Traffic  **Network ACL (NACL) has a deny rule for "All traffic"**, which is likely causing the **SSH connection timeout**.    Added new rule   * **Type:** All traffic * **Protocol:** All * **Port range:** All * **Source:** 0.0.0.0/0 * **Action:** Allow   With this the issue is resolved and were able to launch the instance successfully. |

|  |  |
| --- | --- |
| Step 4.4 | Update and Install Apache (HTTPD) and Create a Sample Web Page, Test in Browser |
| Code used | sudo yum update -y # Update the package list  sudo yum install httpd -y # Install Apache  sudo systemctl start httpd  sudo systemctl enable httpd  sudo systemctl status httpd – to verify Apache is running  echo "<h1>Welcome to Tour Planner Web App!</h1>" | sudo tee /var/www/html/index.html – to create a sample webpage |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Corrected the code - echo '<h1>Welcome to Tour Planner Web App!</h1>' | sudo tee /var/www/html/index.html and issue is resolved. |

# **RDS MySQL Database**

|  |  |
| --- | --- |
| Step 5.1 | Create an RDS MySQL Database |
| Settings and Parameters | **Engine options**:   * **Engine type** → Select **MySQL**. * **Version** → Choose the latest **MySQL 8.x**.   **Deployment Type** → Single-AZ (cheaper option)  **DB instance identifier** → TourPlanner-DB  **Master username** → admin  **Master password** →  **DB instance class** → db.t3.micro (**Free Tier Eligible**).  **Storage type** → General Purpose (SSD)  **Allocated Storage** → 20 GB  **Networking Settings**:   * **VPC** → Select TourPlanner-VPC * **Subnet Group** → Select the **Private Subnets**. * **Public Access** → ❌ **Disable** (RDS should be private). * **Security Group** → Create a new security group (TourPlanner-DB-SG):   + Allow **MySQL/Aurora (Port 3306)** **ONLY from EC2 instance**.   **Additional Configuration**:   * **Enable automatic backups**. * **Monitoring** → Disable (to save cost). * **Deletion Protection** → Enable (**prevents accidental deletion**). |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 5.2 | Allow EC2 to Connect to RDS |
| Settings and Parameters | Update RDS Security Group (TourPlanner-DB-SG)  Under EC2 Dashboard.  Click **Security Groups** → Find **TourPlanner-DB-SG**.  Click **Inbound Rules** → **Edit Inbound Rules**.  **Add Rule**:   * **Type:** MySQL/Aurora * **Port:** 3306 * **Source:** **TourPlanner-WebServer-SG** (EC2 instance security group). |
| Code used | To install MySQL - sudo yum install mysql -y  To test the connection from EC2e - mysql -h <RDS\_ENDPOINT> -u admin -p |
| Screenshot |  |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 5.3 | Create Database & tables in SQL |
| Settings and Parameters | -- |
| Code used | CREATE DATABASE tourplanner;  USE tourplanner;  CREATE TABLE IF NOT EXISTS tours (  id INT AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(255) NOT NULL,  location VARCHAR(255) NOT NULL,  price DECIMAL(10,2) NOT NULL,  description TEXT,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  CREATE TABLE IF NOT EXISTS users (  id INT AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(100) NOT NULL,  email VARCHAR(100) UNIQUE NOT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  INSERT INTO tours (name, location, price, description) VALUES  ('Paris Adventure', 'Paris, France', 1200.50, 'Explore the city of lights with an unforgettable adventure.'),  ('Bali Getaway', 'Bali, Indonesia', 800.00, 'Relax in paradise with our tropical Bali tour.'),  ('Tokyo Explorer', 'Tokyo, Japan', 1500.75, 'Experience the best of Tokyo with our guided city tour.');  CREATE TABLE bookings (  id INT AUTO\_INCREMENT PRIMARY KEY,  tour\_id INT,  customer\_name VARCHAR(255),  email VARCHAR(255) NOT NULL,  phone VARCHAR(20),  booking\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (tour\_id) REFERENCES tours(id) ON DELETE CASCADE  ); |
| Screenshot | -- |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 5.4 | Creating visually appealing frontend |
|  | **Creating new directory for your app –**  cd /var/www/html  sudo mkdir tourplanner  cd tourplanner  Creating index.html - sudo nano /var/www/html/index.html |
| Code used | <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Tour Planner</title>  <link rel="stylesheet" href="styles.css">  </head>  <body>  <h1>Welcome to Tour Planner</h1>  <button id="fetchTours">View Available Tours</button>  <div id="toursContainer"></div>  <!-- Booking Form Modal -->  <div id="bookingModal" class="modal">  <div class="modal-content">  <span class="close">&times;</span>  <h2>Book Your Tour</h2>  <form id="bookingForm">  <input type="hidden" id="tour\_id">  <label for="customer\_name">Name:</label>  <input type="text" id="customer\_name" required>  <label for="email">Email:</label>  <input type="email" id="email" required>  <label for="phone">Phone:</label>  <input type="text" id="phone" required>  <button type="submit">Confirm Booking</button>  </form>  </div>  </div>  <script src="script.js"></script>  </body>  </html> |
| Screenshot |  |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 5.5 | More styling to the webpage |
|  | Creating styles.css - sudo nano /var/www/html/styles.css |
| Code used | body {  font-family: Arial, sans-serif;  text-align: center;  background: linear-gradient(to right, #00c6ff, #0072ff);  color: white;  margin: 0;  padding: 20px;  }  header {  font-size: 24px;  font-weight: bold;  margin-bottom: 20px;  }  .container {  background: white;  color: black;  padding: 20px;  border-radius: 10px;  width: 50%;  margin: auto;  }  input {  padding: 10px;  width: 80%;  border: 1px solid #0072ff;  border-radius: 5px;  margin-bottom: 20px;  }  .tour {  padding: 10px;  border-bottom: 1px solid #ccc;  } |
| Screenshot |  |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 5.6 | Creating app.js |
|  | Creating styles.css - sudo nano /var/www/html/script.js |
| Code used | document.getElementById('fetchTours').addEventListener('click', function() {  fetch('http://54.175.151.23:5000/tours')  .then(response => response.json())  .then(data => {  const toursContainer = document.getElementById('toursContainer');  toursContainer.innerHTML = ''; // Clear existing tours    data.forEach(tour => {  const tourCard = document.createElement('div');  tourCard.classList.add('tour-card');  tourCard.innerHTML = `  <img src="${tour.image\_url}" alt="${tour.name}" class="tour-image">  <h3>${tour.name}</h3>  <p><strong>Location:</strong> ${tour.location}</p>  <p><strong>Price:</strong> $${tour.price}</p>  <button class="bookBtn" data-id="${tour.id}">Book Now</button>  `;  toursContainer.appendChild(tourCard);  });  // Add event listeners to all book buttons  document.querySelectorAll('.bookBtn').forEach(button => {  button.addEventListener('click', function() {  const tourId = this.getAttribute('data-id');  document.getElementById('tour\_id').value = tourId;  document.getElementById('bookingModal').style.display = 'block';  });  });  })  .catch(error => alert("Failed to load tours. Please check the backend."));  });  // Handle Booking Form Submission  document.getElementById('bookingForm').addEventListener('submit', function(event) {  event.preventDefault();  const bookingData = {  tour\_id: document.getElementById('tour\_id').value,  customer\_name: document.getElementById('customer\_name').value,  email: document.getElementById('email').value,  phone: document.getElementById('phone').value  };  fetch('http://54.175.151.23:5000/book', {  method: 'POST',  headers: { 'Content-Type': 'application/json' },  body: JSON.stringify(bookingData)  })  .then(response => response.json())  .then(data => {  alert(data.message);  document.getElementById('bookingModal').style.display = 'none';  })  .catch(error => alert("Booking failed. Please try again."));  });  // Close Modal on Click  document.querySelector('.close').addEventListener('click', function() {  document.getElementById('bookingModal').style.display = 'none';  }); |
| Screenshot | python3 -c "import flask; print('Flask is installed!')" |
| Challenges encountered | Nil |

|  |  |
| --- | --- |
| Step 5.7 | Installation of required software’s |
| Code used | sudo yum install python3 -y  sudo yum install python3-pip -y  pip3 install flask pymysql  sudo python3 app.py  python3 -c "import flask; print('Flask is installed!')" - Verify Flask Installation |

# **Amazon S3 Bucket**

|  |  |
| --- | --- |
| Step 6.1 | Creation on Amazon S3 Bucket & Hosting images |
| Settings and Parameters | Bucket name - tourplanner-images  AWS Region - us-east-1  AWS S3 Console → Use an existing bucket.  Copied Image URL’s and used it in database. |
| Screenshot |  |

# **Testing the Web Application**

|  |  |
| --- | --- |
| Step 7.1 | Test the Web Application |
| Settings and Parameters | Run the Flask app - sudo python3 app.py  Test API - curl http://<EC2\_PUBLIC\_IP>:5000/tours  http://<EC2\_PUBLIC\_IP>/tourplanner/index.html |
|  |  |
| Screenshot |  |
| Challenges encountered | We encountered an error where the web page failed to load the backend data. We fixed the error using the following code  Fixed CORS Error in Backend   * Opened app.py - sudo nano /var/www/html/tourplanner/backend/app.py * Modified - CORS(app, resources={r"/\*": {"origins": "\*"}}) * Restarted flask app - cd /var/www/html/tourplanner/backend * nohup python3 app.py > output.log 2>&1 &   Update API Endpoint in script.js   * Opened java script - sudo nano /var/www/html/script.js * Replaced old IP with new IP * Restarted Apache - sudo systemctl restart httpd |

|  |  |
| --- | --- |
| Step 7.2 | Open the Web Application |
| Settings and Parameters | http://<54.158.32.110>/tourplanner/index.html and below is the fully functional web page, wherein you can view different tour packages and select one and click on book now and it will ask to feed information and confirm the booking. |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect.  A screenshot of a computer  AI-generated content may be incorrect.  A screenshot of a computer  AI-generated content may be incorrect.  A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | * Nill |

# **CloudWatch Monitoring**

|  |  |
| --- | --- |
| Step 8.1 | CloudWatch Monitoring |
| Settings and Parameters | CPU Monitoring (Detect High Usage)  Network Activity (Check Incoming & Outgoing Traffic)  Disk Space Monitoring (Prevent Storage Issues) |
|  |  |
| Screenshot | A screenshot of a computer  AI-generated content may be incorrect. |
| Challenges encountered | * Nil |

# **Load Balancer**

|  |  |
| --- | --- |
| Step 9.1 | Load Balancer for the Tour Planner Web Application |
| Settings and Parameters | Target Group   * Target Type – Instances * Name the target group - TourPlanner-TG * **Protocol & Port:** HTTP **(Port 80)** * VPC – TourPlanner-WebServer   Application Load Balancer (ALB)  Name: TourPlanner-ALB   * Scheme: Internet-facing. * IP Address Type: IPv4. * VPC: Select the same VPC. * Availability Zones: Select at least 2 AZs.   Configure Listeners   * Protocol: HTTP. * Port: 80. * Forward to: Select the Target Group created earlier (TourPlanner-TG).   Security Group Configuration  HTTP (Port 80) → Anywhere 0.0.0.0/0 (for public access).  HTTPS (Port 443) (if using SSL). |
| Screenshot |  |
| Challenges encountered | * Nil |

# **Architecture of the Web Application Deployment**

Figure 1: Architecture of Tour planner web application deployment .

A screenshot of a computer

Description automatically generated

# **11. Conclusion**

AWS cloud infrastructure enables the successful release of the Tour Planner Web Application which proves cloud platforms offer successful methods to construct web applications that scale while delivering reliability and cost-effectiveness. This project achieves high system availability with fault tolerance through the combination of EC2 instances alongside an Application Load Balancer (ALB) and Auto Scaling and RDS for database management and S3 for storage and CloudWatch for monitoring.

The system uses load balancing alongside auto-scaling capabilities to adapt resources automatically for peak performance while decreasing expenses which provides continuous system operation for users. Real-time system monitoring and logging is possible through the integration of AWS CloudWatch with CloudTrail.

The project demonstrates the need for cloud-based deployment strategies which improve web application scalability as well as security features and reduced costs. The upcoming implementation will focus on serverless architecture through AWS Lambda as well as database caching through Amazon ElastiCache and advanced security configurations to build higher efficiency and resilience.

Overall, the **Tour Planner Web Application** deployment is a **robust cloud-based infrastructure model**, showcasing AWS best practices to **deliver a seamless and reliable service to users worldwide**.