**Session - 9**

**📂 Project: Expense Project**

**Create 3 EC2 Instances**

| **Purpose** | **Name** | **Component** |
| --- | --- | --- |
| **🟡 Backend** | **backend** | **Node.js server** |
| **🔵 Frontend** | **frontend** | **Nginx server** |
| **🟢 Database** | **database** | **MySQL or MongoDB** |

**✅ Steps to Create Each Instance:**

1. **Login to AWS Console**  
   Go to **EC2 → Instances → Launch Instance**
2. **Choose Amazon Machine Image (AMI)**
   * Use: **Amazon Linux 2023** or **Ubuntu 22.04**
3. **Choose Instance Type**
   * e.g., t2.micro (for free tier or small usage)
4. **Name and Tag**
   * Example:
     + Name: backend
     + Name: frontend
     + Name: database

**What is MySQL?**

* **MySQL** is a **Relational Database Management System (RDBMS)**.
* It is used to **store, manage, and retrieve data** using **SQL (Structured Query Language)**.
* It’s free and open-source, widely used in web apps (like WordPress, backend APIs, etc.).

**🏠 Analogy:**

* **House** = Linux server
* **Room** = One **DB server** (like MySQL)
* **Rack** = A **Schema**
* **Shelf in rack** = A **Table**
* **Files on shelf** = **Data (rows)**

1. **Install and configure MySQL Server 8.0.x because the developer has chosen MySQL as the database for your project.**
2. **Install MySQL Server 8.0.x**

dnf install mysql-server -y

**Start MySQL Service**

systemctl enable mysqld

systemctl start mysqld

To verify:

netstat -lntp

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:3306 0.0.0.0:\* LISTEN 1234/mysqld

this means:

* MySQL (mysqld) is listening on port 3306
* It's open on all network interfaces (0.0.0.0)

1. **Step: Set Root Password in MySQL**

mysql\_secure\_installation --set-root-pass ExpenseApp@1

**✅ Why this is done:**

1. **Secures the database**
   * By default, MySQL may have **no root password** after installation — this is a **security risk**.
2. **Sets a strong password**
   * ExpenseApp@1 (or your own) is now required to log in as the MySQL root user.
3. **Prepares MySQL for safe use**
   * Prevents unauthorized access to all your data.
4. **Enables login for future commands**
   * Without setting this, you won’t be able to log in securely or run SQL operations using root.
5. 🧑‍💻 **Creating a DNS Record for MySQL in Route 53**

You are setting up a **DNS record** to give your MySQL server a **friendly, easily memorable domain name** instead of using its **raw IP address**.

Your Node.js backend or other services need to connect to MySQL. Instead of hardcoding the private IP address in your configuration files or code, you use the DNS name (mysql.roboshop.internal) to refer to the MySQL server.

For example, instead of using something like 10.0.1.35, you will use a name like mysql.roboshop.internal.

**Record Name: mysql (or any custom name you want, e.g., dbserver)**

**Record Type: A (IPv4 address)**

**Value: Enter the private IP address of your MySQL server. Example: 10.0.1.35.**

**TTL: Keep it as the default (usually 300 seconds).**

**To check whether a domain is created or exists in Linux**

dig example.com

If the domain exists and has a DNS record, it will show ANSWER SECTION with an IP.

nslookup example.com

1. **Verification**

We can check data by using client package called mysql.

Usually command to connect mysql server is

**mysql -h <host-address> -u root -p<password>**

**🔹 -h <host-address>**

Specifies the **host address** of the MySQL server.

* Example: -h 127.0.0.1 or -h mydb.server.com
* If omitted, it connects to **localhost** by default.

**🔹 -u root**

Specifies the **username** to log in as.

* In this case, you're logging in as the **root user** (superuser in MySQL).

**🔹 -p<password>**

Specifies the **password** for the user.

* The password is provided **immediately after -p**, **without a space**.
* For example: -ExpenseApp@1
* If you just type -p (without the password), MySQL will **prompt you** to enter it securely.

⚠️ **Security Tip:** Avoid putting the password in the command line directly (as -pMyPass123) because it can be visible in process lists (ps aux). Use just -p instead, and enter the password interactively.

**✅ Command:**

mysql -u root -p

**🔹 Explanation:**

* -u root → Connect as MySQL's root user.
* -p → Prompts you **securely** to enter the password without showing it on the screen.
* -h is **not needed** if you're connecting to the default local server (localhost).

**🔐 Example Session:**

$ mysql -u root -p

Enter password: \*\*\*\*\*\*\*\*

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 10

Server version: 8.0.x MySQL Community Server

| **#** | **Command** | **Description** |
| --- | --- | --- |
| 1 | SHOW DATABASES; | Lists all available databases. |
| 2 | USE database\_name; | Selects a database to work with. |
| 3 | SHOW TABLES; | Lists all tables in the selected database |
| 4 | SELECT \* FROM students; | Retrieves all records from the students table. |

**1. Installing Node.js**

**The backend is written in Node.js, which is a runtime used to execute JavaScript code outside of the browser.**

**🔍 Step-by-Step**

1. **Check with developer which version is needed. Developer confirmed: NodeJS > 20.**
2. **Default version (16) is not sufficient. So we:**

**dnf module list nodejs**

**This shows all available versions.**

1. **Disable default version:**

**dnf module disable nodejs -y**

1. **Enable version 20:**

**dnf module enable nodejs:20 -y**

1. **Install Node.js 20:**

**dnf install nodejs -y**

**✅ Now Node.js 20 is ready to run your backend.**

**2. Creating a Dedicated Application User**

**System users are often created for services for security reasons.**

**useradd --system --home /app --shell /sbin/nologin --comment "expense user" expense**

* **--system: Marks this as a system user.**
* **--home /app: Home directory where the application will reside.**
* **--shell /sbin/nologin: Prevents shell access (not for human login).**
* **--comment: Descriptive label.**

**The user expense will own and run the backend service.**

**3. Application Directory Setup**

**Standard organizational practice is to keep apps in /app.**

**mkdir /app**

**4. Download and Extract Application Code**

**curl -o /tmp/backend.zip https://expense-builds.s3.us-east-1.amazonaws.com/expense-backend-v2.zip**

**cd /app**

**unzip /tmp/backend.zip**

* **This downloads the code.**
* **Extracts it into the /app directory.**

**5. Installing Application Dependencies**

**Applications in Node.js use a package.json file to define dependencies.**

**cd /app**

**npm install**

* **Installs all packages and libraries needed by the application.**

**6. Create a systemd Service**

**To manage your application like a standard Linux service:**

**vim /etc/systemd/system/backend.service**

**Paste the following:**

**[Unit]**

**Description = Backend Service**

**[Service]**

**User=expense**

**Environment=DB\_HOST="<MYSQL-SERVER-IPADDRESS>"**

**ExecStart=/bin/node /app/index.js**

**SyslogIdentifier=backend**

**[Install]**

**WantedBy=multi-user.target**

**📌 Replace <MYSQL-SERVER-IPADDRESS> with the actual MySQL database IP address. We created a record in route 53 for db(mysql) paste the URL here.**

**7. Starting and Enabling the Service**

**Reload systemd to detect new service:**

**systemctl daemon-reload**

**Start and enable at boot:**

**systemctl start backend**

**systemctl enable backend**

**8. Installing MySQL Client and Loading Schema**

To allow the app to communicate with MySQL and load the initial schema:

**dnf install mysql -y**

**Load schema into the DB:**

**mysql -h <MYSQL-SERVER-IPADDRESS> -uroot -pExpenseApp@1 < /app/schema/backend.sql**

This command is used to connect to a remote MySQL server and execute a SQL script file (backend.sql) which contains schema definitions (like table creation statements).

**🔍 Explanation of Each Part:**

| **Part** | **Meaning** |
| --- | --- |
| mysql | The MySQL client command-line tool. It lets you connect to a MySQL server and run SQL commands. |
| -h <MYSQL-SERVER-IPADDRESS> | -h stands for host. You provide the IP address or hostname of the MySQL server you want to connect to. Replace <MYSQL-SERVER-IPADDRESS> with the actual IP address (e.g., 192.168.1.100). |
| -u root | -u specifies the username. Here, it's root, which is the MySQL administrative user. |
| -pExpenseApp@1 | -p is for password. There is no space between -p and the password. In this case, the password is ExpenseApp@1. |
| < /app/schema/backend.sql | This part tells the shell to redirect the contents of the SQL file (backend.sql) as input into the MySQL client. That means MySQL will read and execute all the SQL statements in that file. |

**9. Restart Backend to Reflect Schema**

**systemctl restart backend**

Now your backend is:

* Installed and running under systemd.
* Connected to the database.
* Running on the correct Node.js version.
* Using a secure, dedicated user.

**Your Project Port Mapping**

| **Component** | **Port** | **Purpose** |
| --- | --- | --- |
| Nginx | 80 | Acts as a reverse proxy for the frontend/backend |
| Node.js | 8080 | Hosts the backend API (Express app) |
| MySQL | 3306 | Hosts the database (data storage layer) |

**Test Remote Access to Port 3306**

**From another machine (like your backend server), run:**

**telnet <MYSQL-IP or URL> 3306**

**Frontend**

The frontend is the service in Expense to serve the web content over Nginx. This will have the web frame for the web application.

This is a static content and to serve static content we need a web server. This server

Developer has chosen Nginx as a web server and thus we will install Nginx Web Server.

* 1. **Install Nginx**
  2. **dnf install nginx -y**
  3. **Enable nginx**
  4. **systemctl enable nginx**
  5. **Start nginx**
  6. **systemctl start nginx**

Try to access the service once over the browser and ensure you get some default content

Remove the default content that web server is serving.

* 1. **rm -rf /usr/share/nginx/html/\***

Download the frontend content

* 1. **curl -o /tmp/frontend.zip https://expense-builds.s3.us-east-1.amazonaws.com/expense-frontend-v2.zip**

Extract the frontend content.

* 1. **cd /usr/share/nginx/html**
  2. **unzip /tmp/frontend.zip**

Try to access the nginx service once more over the browser and ensure you get expense content.

Create Nginx Reverse Proxy Configuration.

* 1. **vim /etc/nginx/default.d/expense.conf**

Add the following content

**proxy\_http\_version 1.1;**

**location /api/ { proxy\_pass http://localhost:8080/; }**

**location /health {**

**stub\_status on;**

**access\_log off;**

**}**

Ensure you replace the localhost with the actual ip address of backend component server. Word localhost is just used to avoid the failures on the Nginx Server.

Restart Nginx Service to load the changes of the configuration.

* 1. **systemctl restart nginx**
  2. **Creating DNS Record for backend to connect with frontend.**

| Field | Value |
| --- | --- |
| Record Name | backend |
| Type | A |
| Value | <Backend-IP> (e.g., 192.168.10.20) |
| TTL | 1 sec (very short, optional) |

**🧩 Why You Create a DNS "A" Record for Backend**

When your frontend (served by Nginx) needs to communicate with the backend (Node.js app), Nginx uses the proxy\_pass directive to forward those requests.

If your backend is running on a different server/IP, it's best to:

* Create a DNS A record like backend.example.com (or just backend)
* Map it to the backend server’s IP
* Use that record name in proxy\_pass instead of a raw IP
  1. **Edit /etc/nginx/default.d/expense.conf:**

location /api/ {

proxy\_pass http://backend:8080/;

}

* 1. **sudo systemctl restart nginx**
  2. **nvlookup <url>**

**Reverse Proxy – Definition & Explanation**

A reverse proxy is a server that receives client requests and forwards them to one or more backend servers, then returns the server’s response to the client as if it originated from the proxy itself**.**

**🧠 In Simple Terms:**

Think of a reverse proxy like a receptionist at a company. Visitors (clients) don’t go directly to employees (backend servers); instead, they go to the receptionist (reverse proxy), who forwards the message to the right person.

Or

Imagine you visit https://example.com. Here's what happens with a reverse proxy setup:

* Your browser sends a request to https://example.com.
* A **reverse proxy** (like **Nginx**) receives the request.
* The reverse proxy forwards the request to a backend server, say http://localhost:3000.
* The backend server sends back the response (like an HTML page).
* The reverse proxy sends this response to your browser.

🎯 **Benefits of Reverse Proxy:**

* Load balancing: Distributes requests to multiple servers.
* Security: Hides internal server structure from clients.
* SSL termination: Handles HTTPS, reducing backend server load.
* Caching: Stores responses to reduce backend load.
* Compression: Optimizes bandwidth by compressing responses.

**🔄 Reverse Proxy vs. Forward Proxy**

| Proxy Type | Who It Serves | Common Use |
| --- | --- | --- |
| Forward Proxy | Client | Hides client from the server (e.g., VPN, corporate proxy) |
| Reverse Proxy | Server | Hides backend servers from client (e.g., Nginx, Apache) |

**Forward Proxy – Definition & Explanation**

A forward proxy is a server that acts on behalf of the client, forwarding their requests to external servers (usually on the internet), and then returning the responses from those servers back to the client**.**

**🧠 In Simple Terms:**

You're in a country where certain websites are blocked. You set up a forward proxy server in another country. Your browser connects to that proxy, which then connects to the blocked website on your behalf**.**

**Or**

Imagine you're using a company computer and try to open https://www.facebook.com, but Facebook is blocked at work.

* Your browser tries to connect to Facebook.
* The request goes through the company’s forward proxy server.
* The proxy server is configured to block social media, so it denies the request.
* You get a "Blocked by company policy" message.

**🎯 Benefits of Forward Proxy:**

* **Bypass geo-restrictions** or censorship.
* **Monitor and control** employee/student internet use.
* **Cache** content for faster access.
* **Hide client IP addresses** for privacy.

**🌐 What is an /api Proxy?**

An /api proxy is a configuration (usually in web development or server setup) where all HTTP requests that start with the path /api are forwarded to another server — often a backend server or microservice — using a reverse proxy.

**✅ Definition:**

An /api proxy is a routing rule that tells a web server (like Nginx, Apache, or a development server like Webpack, Vite, or Create React App) to forward all requests starting with /api to a different server (usually your backend).

**🔧 Why Use an /api Proxy?**

* To separate frontend and backend logic.
* To avoid CORS issues during development.
* To simplify frontend code (e.g., use /api/users instead of http://localhost:3000/api/users).
* To hide backend URLs from users.

📦 Example 1: Nginx /api Reverse Proxy

ocation /api/ {

proxy\_pass http://localhost:3000/;

}

**This means:**

Any request to http://frontend-server/api/\* is forwarded to http://localhost:3000/\* (your backend server). So /api/users → http://localhost:3000/users.

**🔧 Troubleshooting When Frontend Can't Connect to Backend**

**If your frontend server cannot connect to the backend, follow these steps to troubleshoot:**

**✅ Step-by-Step Troubleshooting**

1. **Check Connection from Frontend to Backend:**

On the frontend server, run:

telnet backend.daws84s.site 8080

* + If it connects, the network and service are fine.
  + If it says “Connection refused”, this means the backend service is either not running or blocked.

1. **Check Backend Service Status:**

On the backend server, check if the backend service is running:

systemctl status backend

* + If it shows “failed” or “inactive”, the service has crashed or didn’t start.

1. **Investigate Why Backend Service Failed:**

Go to the log directory:

cd /var/log

Use the less command to check the messages log file:

less messages

**🧾 What is the less command?**

less is a Linux command used to view large text files page-by-page or line-by-line. It’s more efficient than cat or more because you can scroll up and down, search inside the file, and it doesn’t load the whole file at once.

Example usage:

* + Use arrow keys or PgUp/PgDn to scroll.
  + Type /error to search for the word “error”.
  + Press q to quit.

1. **Restart the Backend Service:**

After reviewing the logs and fixing any issues, restart the backend service:

systemctl restart backend

Then check the status again:

systemctl status backend

**✅ Summary:**

| **Step** | **Command** |
| --- | --- |
| Test connection from frontend | telnet backend.daws84s.site 8080 |
| Check backend service status | systemctl status backend |
| Check logs for errors | cd /var/log → less messages |
| Restart backend | systemctl restart backend |

**📂 Checking Nginx Logs on Frontend Server**

**✅ Log File Locations:**

On most Linux systems (e.g., Ubuntu, CentOS), Nginx logs are stored in:

/var/log/nginx/

Inside this directory, the two main log files are:

* access.log — records every request to the web server.
* error.log — logs errors such as failed requests or misconfigurations.

**🔍 View Real-Time Logs with tail -f**

To monitor live activity (e.g., incoming user requests), use:

cd /var/log/nginx

tail -f access.log

This will show real-time entries as users access your web application.

**🧾 What is tail -f?**

tail -f is a Linux command that displays the end of a file and keeps updating it as new lines are added — perfect for monitoring logs live.

Example usage:

* tail -f access.log → See live HTTP requests.
* tail -f error.log → Watch for real-time errors or failures.

**🔄 Combine With grep (Optional)**

If you're looking for specific entries (e.g., 500 errors):

tail -f access.log | grep "500"

**✅ 1xx – Informational**

These indicate that the request was received and is still being processed.

| Code | Meaning | Description |
| --- | --- | --- |
| 100 | Continue | Initial part received, continue |
| 101 | Switching Protocols | Server is switching protocols (e.g., HTTP to WebSocket) |

**🟢 2xx – Success**

These mean the request was successfully received, understood, and accepted.

| Code | Meaning | Description |
| --- | --- | --- |
| 200 | OK | Standard response for success |
| 201 | Created | Resource successfully created |
| 202 | Accepted | Request accepted but not yet processed |
| 204 | No Content | Success but no content to return |

**🟡 3xx – Redirection**

These codes mean the client must take further action to complete the request.

| Code | Meaning | Description |
| --- | --- | --- |
| 301 | Moved Permanently | Resource has moved permanently |
| 302 | Found | Temporary redirect |
| 304 | Not Modified | Use cached version |

**🔴 4xx – Client Errors**

These indicate the request had bad syntax or cannot be fulfilled by the client.

| Code | Meaning | Description |
| --- | --- | --- |
| 400 | Bad Request | Malformed request |
| 401 | Unauthorized | Authentication required |
| 403 | Forbidden | Authenticated, but access denied |
| 404 | Not Found | Resource not found |
| 405 | Method Not Allowed | HTTP method not supported |
| 408 | Request Timeout | Client took too long |

**🔥 5xx – Server Errors**

**Th**ese indicate the server failed to fulfill a valid request.

| Code | Meaning | Description |
| --- | --- | --- |
| 500 | Internal Server Error | General server error |
| 501 | Not Implemented | Feature not supported by server |
| 502 | Bad Gateway | Invalid response from upstream server |
| 503 | Service Unavailable | Server temporarily unavailable |
| 504 | Gateway Timeout | Upstream server didn’t respond in time |