Introduction

This document provides a comprehensive guide for setting up Nginx as a load balancer for gRPC services, specifically tailored for the App1 and App2 setup. App1 serves as the gRPC server, implementing EmployeeService for managing employee data in a MySQL database and FileTransferService for handling file operations. App2 acts as the client, sending requests to App1 via a consolebased interface. Nginx distributes traffic across multiple App1 instances, ensuring scalability and reliability. This guide is based on the provided setup instructions and aligns with best practices for gRPC and load balancing configurations, as detailed in resources like the official Nginx documentation and tutorials on gRPC load balancing.

Overview

App1 and App2 communicate using gRPC, a high-performance remote procedure call framework that leverages HTTP/2. Nginx acts as a load balancer to distribute client requests across multiple App1 instances, improving performance and fault tolerance.

App1

App1 is the server, hosting two gRPC services:

• EmployeeService: Manages employee records (create, read, query) in a MySQL database, supporting operations like adding employees or listing by role.

• FileTransferService: Handles file uploads, downloads, and metadata operations, including conflict resolution for uploads.

App1 contains all logic to process requests, such as querying the database or storing files, as defined in the gRPC service implementations.

App2

App2 is the client, sending requests to App1's services via a console-based menu. It does not implement any service logic; it constructs and sends gRPC requests to App1 and displays responses.

Connection

App2 connects to App1 through Nginx, which listens on port 50050 and forwards requests to App1 instances running on ports like 50051 and 50052. The services are defined in proto files within the Common_File folder, and App2 uses these to generate requests.

Workflow

1. Set up the MySQL database with sample data and configure the file storage directory.
2. Configure and run multiple instances of App1 to start the gRPC servers.
3. Configure Nginx to load balance traffic across App1 instances.
4. Run App2 to connect to Nginx and send requests for employee or file operations.
5. App1 processes requests and sends responses, handling file conflicts when necessary.

Prerequisites

Before proceeding, ensure the following are installed and configured:

- Java 21 JDK: Download from Oracle or OpenJDK, set JAVA_HOME, and verify with java -version.
- MySQL Server: Installed and running on localhost:3306 with admin access.
- Eclipse IDE: Download from Eclipse for project management.
- Project Files: App1, App2, and Common_File folders containing gRPC proto files, JARs, and MySQL Connector.
- File Storage Directory: Use the storage folder in App1 for file operations.
- Nginx: Installed for load balancing gRPC traffic.

App1 Setup and Running Instructions

Introduction

This document provides instructions on how to set up and run App1, a Java-based application that connects to a MySQL database and starts a server.

Prerequisites

To run App1, you need the following:

- **Java JDK installed**: Ensure you have Java JDK version 8 or higher installed on your system.
- **MySQL server running**: A MySQL server must be installed and running, with a database containing the required employees table.
- **App1 JAR file**: The application JAR file, named App1loadbalanertest_nginx.jar, must be available.

Running the Application

To run App1, execute the following command in your terminal or command prompt:

java -jar App2loadbalanertest_nginx.jar

When you run this command, the application will prompt you for several inputs. Follow the instructions below for each prompt:

1. **Initial IP address to check:**
   Enter the IP address where you want the server to run, e.g., localhost or 192.168.1.100. If you're running on your local machine, use localhost.
2. **Initial port number to check (1-65535):**
   Enter the port number you want the server to use, e.g., 8081. Ensure this port is not already in use by another application. The application will verify if the port is available.
3. **MySQL host:**
   Enter the hostname or IP address of your MySQL server, e.g., localhost if it's on the same machine.
4. **MySQL port:**
   Enter the port number of your MySQL server. The default is 3306.
5. **Username:**
   Enter your MySQL database username.
6. **Password:**
   Enter your MySQL database password.

7. **Connection timeout in milliseconds:**
   Enter the timeout value for the database connection, e.g., 3000 (3 seconds).
8. **Choose an option for database selection:**
   You will be presented with the following options:
   - Enter database name manually
   - Select from available databases
   - Exit

     Choose option 2 to select from available databases.
9. **Select database:**
   The application will list available databases. Enter the number corresponding to the database you want to use, e.g., 11 for employee_db.
10. **Enter directory path:**
    Enter the path to the directory where you want to store data or configuration files, e.g., C:\Users\suthar-pt7931\eclipse-workspace\Appservice1\Storage.

After entering all the required information, the server will start on the specified IP and port.

Additional Notes

- Ensure that the MySQL server is running and accessible with the provided credentials.
- Make sure the specified port is not in use by another application.
- The application requires a database with a table named employees having the following columns: id, name, role, salary, district, branch.

NGINX Load Balancer Setup for gRPC Services

1. Important Notes

- **Before Starting NGINX:**
  Move your terminal or command line to the directory where the `nginx.exe` and its configuration files are located.

- **NGINX Configuration File:**
  Inside the `conf` directory, you will find the file:
  `conf/nginx/nginx.conf`
  The file has a `.conf` extension and needs to be edited according to your server requirements.

---

2. NGINX Configuration Sample

You must define the gRPC backend servers inside the `http` block using the `upstream` directive.

```
events {
}
http {
    client_max_body_size 5000m;
    upstream grpcservers {
        ip_hash;
        server 10.92.56.70:50051;
        server 10.92.56.70:50052;
        server 10.92.56.70:50053;
        # Add or remove servers as needed
    }
    server {
        listen 10.92.56.70:50050 http2;

        location / {
            grpc_pass grpc://grpcservers;
        }
    }
}
```

Note: Modify the IPs and ports as per your application setup.

---

3. Commands for Managing NGINX and Application

Start NGINX Load Balancer

bash
CopyEdit

```
start nginx
```

Check if NGINX is Running

bash
CopyEdit

```
tasklist /fi "imagename eq nginx.exe"
```

Example Output:

pgsql
CopyEdit

```
Image  Name                                PID  Session  Name
Session#    Mem Usage
=========================      ========      ================
===========  ============
nginx.exe                                  4952  Console
3      9,924 K
nginx.exe                                 24276  Console
3     10,172 K
```

Check Which Process is Using Port 50050

bash
CopyEdit

```
netstat -ano | findstr :50050
```

Stop NGINX Load Balancer

```
nginx.exe -s stop
```

---

4. Running the Client Application

To run the client (App2) after starting the NGINX load balancer:

```
java -Xms256m -Xmx12g -jar App2loadbalanertest_nginx.jar
```

- **Enter your IP address (host) where NGINX is running:**
  `localhost`
- **Enter your port address where NGINX is listening:**
  `50051`

App2 will then send gRPC requests through the NGINX load balancer, which forwards them to App1 instances.

---

5. Startup Order

1. Start **App1** (your gRPC server instances).
2. Then start **NGINX** using the `start nginx` command.
3. Finally, start **App2** (client) using the above Java command.

App1 and App2 Communication via gRPC with NGINX Load Balancer

---

System Overview

This system consists of two primary applications:

- **App1** – A **gRPC server**, hosting the core business logic and services.
- **App2** – A **gRPC client**, interacting with the user and forwarding requests to App1 through an **NGINX load balancer**.

Network & Deployment Configuration

App1: Server Setup

- **Runs on IP**: `10.92.56.70`
- **Ports**: `50051`, `50052`, `50053` (multiple gRPC servers for load balancing)
- **Implements**:
    - `EmployeeService`: Handles operations on employee records in a MySQL database.
    - `FileTransferService`: Manages file upload/download using SHA-256 hash conflict resolution.
- Each instance of App1 binds to a unique port and runs independently on the same host IP.

NGINX: Load Balancer

- **Listens on**: `10.92.56.70:50050` using HTTP/2 protocol (required for gRPC)
- **Load Balancing Strategy**: `ip_hash` ensures requests from the same client go to the same backend
- **Forwards traffic** to:
    - `10.92.56.70:50051`
    - `10.92.56.70:50052`
    - `10.92.56.70:50053`

✅ NGINX handles the distribution of gRPC requests across App1 instances.

App2: Client Setup

- **Connects to**: `10.92.56.70:50050` (NGINX)
- **Sends gRPC requests** using stubs generated from `.proto` files.
- App2 doesn't contain any business logic; it only sends requests and displays responses.

NGINX Configuration Sample (`nginx.conf`)

nginx
CopyEdit

```
events {}

http {
```

```
    client_max_body_size 5000m;

    upstream grpcservers {
        ip_hash;
        server 10.92.56.70:50051;
        server 10.92.56.70:50052;
        server 10.92.56.70:50053;
    }

    server {
        listen 10.92.56.70:50050 http2;

        location / {
            grpc_pass grpc://grpcservers;
        }
    }
}
```

How App1 and App2 Work Together (Through Load Balancer)

Connection Workflow

App1 is started on multiple ports:

bash
CopyEdit

```
java -jar App1.jar --server.port=50051
java -jar App1.jar --server.port=50052
java -jar App1.jar --server.port=50053
```

Start NGINX Load Balancer:

```
start nginx
```

App2 connects to NGINX at:

```
java -Xms256m -Xmx12g -jar App2loadbalanertest_nginx.jar
```

1. **App2 sends requests**, which are routed by NGINX to any available App1 instance.

---

gRPC Request Lifecycle

1. Before a Request

- App1 must be running and registered in NGINX.
- App2 creates a gRPC **channel to the NGINX server (10.92.56.70:50050)**.
- A stub is initialized to send requests (blocking or async).
- User provides input via App2 (e.g., employee ID, file path).
- App2 builds the request message using `.proto` definitions.

2. On Next

- App2 sends the request through NGINX.
- NGINX routes it to one of the App1 instances.
- App1 processes the request (e.g., queries database, handles file logic).

- App1 sends back a response to App2 via NGINX.

3. On Error

- If App1 encounters a problem (e.g., invalid ID), it returns a gRPC error.
- App2 catches the error and notifies the user.

4. On Complete

- App1 sends the final response and closes the stream.
- App2 displays the results and returns to the main menu.

App2 Operations Overview

Option 1: Employee Operations

- Managed by `EmployeeService`
- Connects to MySQL database via App1
- Supported Operations:
    1. Create Employee
    2. Get Employee by ID
    3. Get Employees by Role
    4. Get Employees by Salary

5. Get Employees by District
6. Get Employees by Branch
7. Get All Employees
8. Get by Role and Branch
9. Get with Salary >
10. Get with Salary Between
11. Get by Role and Min Salary
12. Get by District Ordered by Salary
13. Get Names and Roles by Branch

Option 2: File Transfer Operations

- Managed by `FileTransferService`
- Operates in App1's storage directory: `App1/storage`

List Files

- Lists all available files.
- Blocking stub used for fast, simple response.

Upload File

- File is split into 64KB chunks and streamed using an async stub.

- SHA-256 used to detect conflicts:
    - **Identical File**:
        - Option 1: Keep original
        - Option 2: Rename and save
    - **Different File**:
        - Option 1: Overwrite
        - Option 2: Rename and save
        - Option 3: Discard
- User is prompted via App2 on how to resolve conflicts.

Download File

- Streams a file from App1 to App2 in 64KB chunks.
- Progress updates are printed every 10MB.

Important Commands & Tools

Start NGINX

```
start nginx
```

Check if NGINX is Running

```
tasklist /fi "imagename eq nginx.exe"
```

Check Port Usage

```
netstat -ano | findstr :50050
```

Stop NGINX

```
nginx.exe -s stop
```

Run App2

```
java -Xms256m -Xmx12g -jar App2loadbalanertest_nginx.jar
```

Why This Architecture Matters

- Enables horizontal scalability of App1 using NGINX.
- Provides a centralized entry point for App2.
- Offers fault tolerance: if one App1 instance fails, NGINX reroutes traffic.
- Keeps client logic simple while server handles all business complexity.

Future Plan

1. We are going to develop two components: a Server and an Agent.
   These two parts will work together as a distributed system for performing tasks and communication.

2. The Server will be developed in Java.
   It will be responsible for managing tasks, sending instructions, and

coordinating with multiple agents.

3. The Agent will be developed in C++.
   It will receive instructions from the server and perform system-level operations accordingly.

Communication between the Server and Agent will happen using Protocol Buffers (protobuf).