National College of Ireland

Optimizing Task Orchestration for Latency Reduction in 5G-Enabled Cloud Environments

MSc Research Project
Cloud Computing

Siddhartha Reddy Thigala
Student ID: 24135003

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Siddhartha Reddy Thigala
Student ID:	24135003
Programme:	Cloud Computing
Year:	2025-2026
Module:	MSc Research Project
Supervisor:	Ahmed Makki
Submission Due Date:	11/12/2025
Project Title:	Optimizing Task Orchestration for Latency Reduction in 5G-Enabled Cloud Environments
Word Count:	2403
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Siddhartha Reddy Thigala
Date: 11/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Siddhartha Reddy Thigala
24135003

1 System Requirements

This part of the document explains what basic hardware and software are needed to work with the project "Optimizing Task Orchestration for Latency Reduction in 5G-Enabled Cloud Environments", whether it is running on a personal machine or deployed using Amazon Web Services (AWS).

1.1 Hardware Requirements

- **Central Processing Unit**(CPU) - The CPU must be a dual-core with a minimum clock speed of 2.0 GHz or a faster (e.g., quad) CPU for faster experiment times.
- **Memory**: At least 8 GB RAM (16 GB recommended when running multiple trials or other applications in parallel).
- **Storage**: It's best to have at least 10 GB of free space available, so that there is enough room for Python packages, generated logs, and output files.
- **Network**: Stable broadband connection for cloning the GitHub repository and interacting with AWS services.

1.2 Software Requirements

- **Operating System**: Ubuntu 20.04 LTS or later, macOS Monterey or later, or Windows 10+.
- **Python**: Version 3.10 or later, with `pip` package manager available.
- **Git**: For cloning the project repository from GitHub.
- **VS Code (optional)**: Open Visual Studio Code and take advantage of the "Remote SSH" extension, which lets you work on the cloud machine that's running on EC_2.
- **AWS Command Line Interface:**: A tool that uses for interacting with AWS services, set up using an IAM identity that has the required access privileges.

1.3 Python Dependencies

All Python dependencies are defined in the project's `requirements.txt` file. Key libraries include:

- **NumPy** and **Pandas** for numerical processing and CSV handling.
- **Matplotlib** and **Seaborn** for generating latency and reward plots.
- **Boto3** for interacting with Amazon S3 and CloudWatch.
- **PyTest** for running the test suite.

These packages are installed automatically during the installation steps in Section 2.

1.4 AWS Account and IAM Requirements

To run experiments on AWS, an active AWS account is required. The project assumes an IAM role named `LatencyOrchestrator` with the following managed policies (or equivalent custom policies) attached:

- **AmazonS3FullAccess**: full access to S3 buckets used for experiment artefacts.
- **AmazonEC2FullAccess**: ability to launch and manage EC2 instances.
- **CloudWatchLogsFullAccess**: permission to create log groups, streams and publish logs.

This role is later attached to the EC2 instance that executes the remote training job.

2 Installation Guide

This part outlines how to prepare the environment on an AWS virtual machine so that the project can run remotely. All configuration steps are carried out directly on the chosen EC2 server. In general, the process follows a simple end-to-end sequence:

1. Set up a new IAM role named `LatencyOrchestrator` and attach the required policies.
2. Create Linux virtual machine on AWS EC2 and save the key pair while creating.
3. Connect to the EC2 instance via SSH in VS Code.
4. Prepare the machine by refreshing its package list, then add the required development tools such as Python, pip, and Git.
5. Retrieve the source code from the GitHub repository and navigate into the directory that contains the project files.
6. Install the required libraries from `requirements.txt`.
7. Run the `main_remote.py` file to execute experiments.
8. Verify that results are saved in S3 and logs are created in CloudWatch.

2.1 Step 1: Creating IAM Role and Adding Policies

To allow EC2 instance to access S3 and CloudWatch, create IAM role with necessary permissions:

1. Go to AWS dashboard and go to the Identity and Access Management *IAM* area, then choose the option for managing Roles. *Roles*.
2. Click *Create role*, indicate that the permissions will be used by an *AWS service*, and assign them to *EC2* for this project
3. Name the role`LatencyOrchestrator`.
4. Attach the following three managed policies:
   - `AmazonS3FullAccess` – provides full access to S3 buckets for storing experiment artefacts.
   - `AmazonEC2FullAccess` – allows full access to EC2 instances and related resources.
   - `CloudWatchLogsFullAccess` – grants permission to create log groups, streams and publish logs to CloudWatch.
5. Complete the wizard and save the role.

A figure showing the IAM role creation screen can be included:

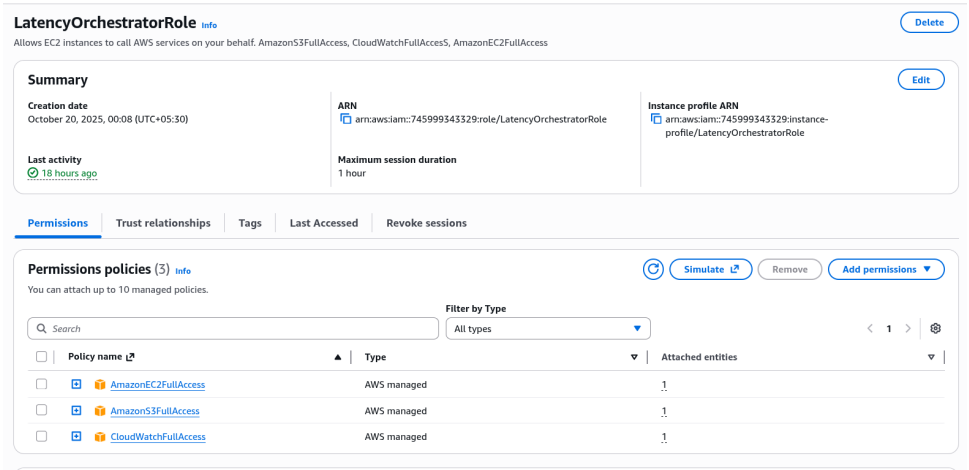

Figure 1: Creating the `LatencyOrchestrator` IAM role with required policies.

2.2 Step 2: Creating the EC2 Instance and Downloading the Key Pair

Launch an EC2 Linux instance that will run the training and evaluation:

1. In the AWS Console, navigate to the *EC2* service and click *Launch instance*.
2. Choose an Ubuntu Server 20.04 LTS AMI (or later).

3. Select an instance type (e.g., `t3.medium` or `t3.large`).

4. Under *Key pair (login)*, either select an existing key pair or create a new one. **Important:** If creating a new key pair, download the `.pem` file immediately and store it securely. This file is required for SSH access and cannot be downloaded again.

5. Under *Network settings*, configure a security group that allows SSH (`TCP` port 22) from your IP address.

6. Under *Advanced details*, in the *IAM instance profile* dropdown, select the `LatencyOrchestrator` role created in Step 1.

7. Review the configuration and click *Launch instance*.

8. Note the public IP address or public DNS name of the launched instance. This will be needed for SSH connection.

A diagram of the EC2 launch process can be included:

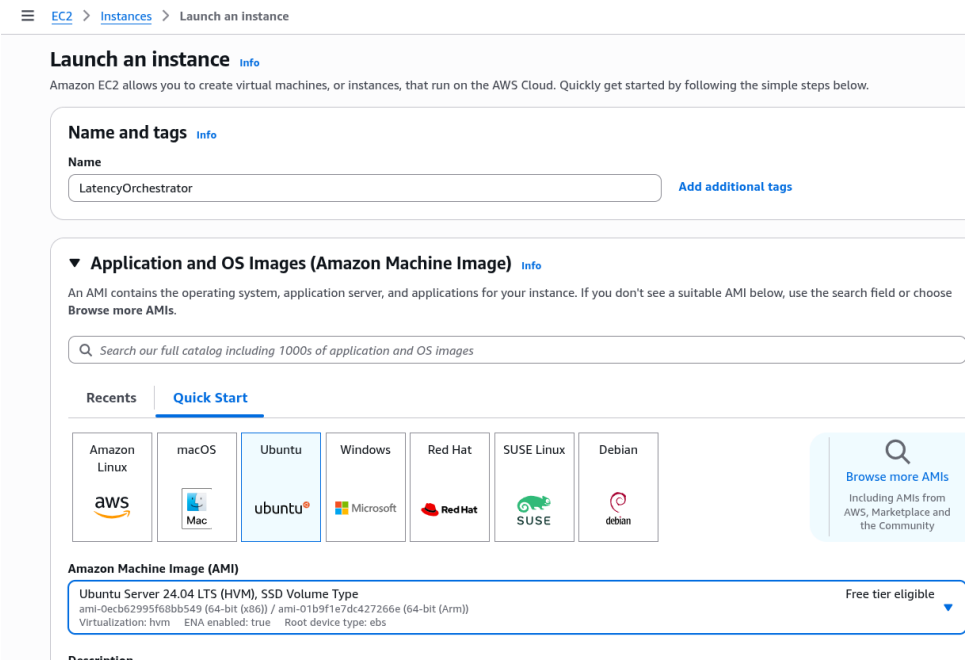

Figure 2: Launching the Ubuntu EC2 instance with IAM role and key pair.

2.3 Step 3: Connecting to EC2 via SSH in VS Code

Use Visual Studio Code with the Remote SSH extension to connect to the EC2 instance:

1. Install the "Remote – SSH" extension in VS Code if not already installed.

2. Open your local SSH configuration file (typically `~/.ssh/config`) and add a new host entry:

```
Host latency-orchestrator-ec2
    HostName <EC2_PUBLIC_IP>
    User ubuntu
    IdentityFile ~/.ssh/your-key-pair.pem
```

Replace `<EC2_PUBLIC_IP>` with the actual public IP address of your EC2 instance, and `your-key-pair.pem` with the path to your downloaded key pair file.

3. Set the correct permissions on the key file (required on Linux/macOS):

```
chmod 400 ~/.ssh/your-key-pair.pem
```

4. In VS Code, open the Command Palette (Ctrl+Shift+P or Cmd+Shift+P) and select "Remote-SSH: Connect to Host".
5. Choose the host entry you just created (e.g., `latency-orchestrator-ec2`).
6. A new VS Code window opens, connected to the remote EC2 instance.
7. Open a terminal in VS Code (Terminal → New Terminal) to proceed with the installation steps.

A screenshot of the VS Code Remote SSH connection can be referenced:

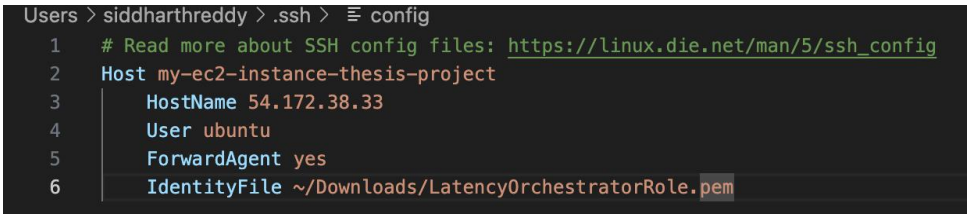

Figure 3: Connecting to the EC2 instance using VS Code Remote SSH.

2.4 Step 4: Updating the System and Installing Python, pip, and Git

Once connected to the EC2 instance via VS Code (or a terminal), update the system packages and install the required tools:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y python3 python3-venv python3-pip git
```

This command:

- Updates the package list and upgrades existing packages.
- Installs Python 3, the Python virtual environment module, pip (Python package manager), and Git.

Verify the installations:

```
python3 --version
pip3 --version
git --version
```

2.5 Step 5: Cloning the GitHub Repository and Changing Directory

Clone the project repository from GitHub and navigate to the project directory:

```
git clone https://github.com/cdarthreddy/LatencyReductionInCloud.git
cd LatencyReductionInCloud
```

This creates a local copy of the project source code on the EC2 instance.

2.6 Step 6: Installing Required Libraries

Set up a Python virtual environment and install all project dependencies:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

The virtual environment isolates project dependencies from system Python packages. After activation, the shell prompt should be prefixed with `(venv)`. The `requirements.txt` file installs all necessary libraries including NumPy, Pandas, Matplotlib, Seaborn, Boto3, and PyTest.

A figure illustrating the EC2 terminal after installation:

2.7 Optional: Local Setup

For local development and testing, you can also set up the project on your local machine:

Local Installation Steps

1. Clone the repository:

   ```
   git clone https://github.com/cdarthreddy/LatencyReductionInCloud.git
   cd LatencyReductionInCloud
   ```

2. Create and activate a virtual environment:

   ```
   python3 -m venv venv
   source venv/bin/activate  # On Windows: .\venv\Scripts\Activate.ps1
   ```

3. Install dependencies:

   ```
   pip install -r requirements.txt
   ```

4. Run a local sanity check:

   ```
   python main.py
   ```

If desired, you may illustrate the local workflow with a figure:

At this point, the EC2 instance is fully configured and ready to run experiments. Proceed to Section 3 for instructions on executing the main scripts and verifying results.

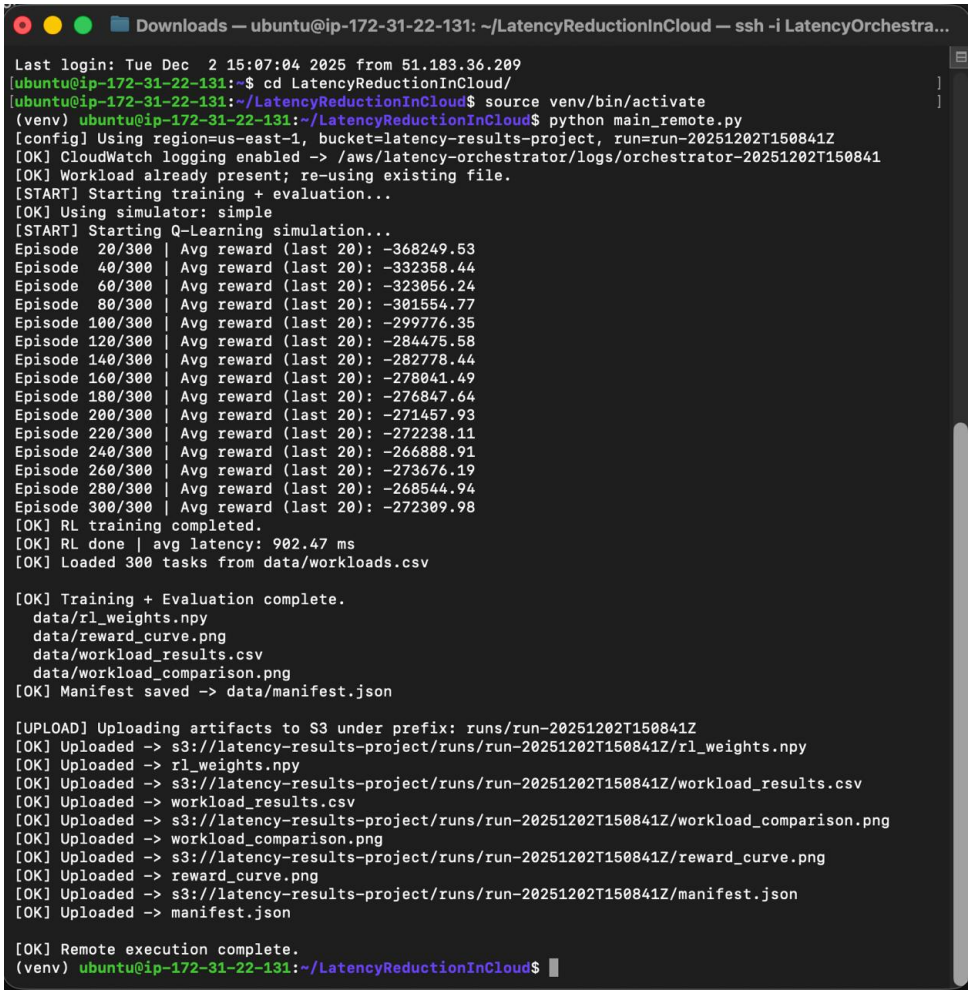

Figure 4: EC2 terminal after cloning the repository and installing dependencies.

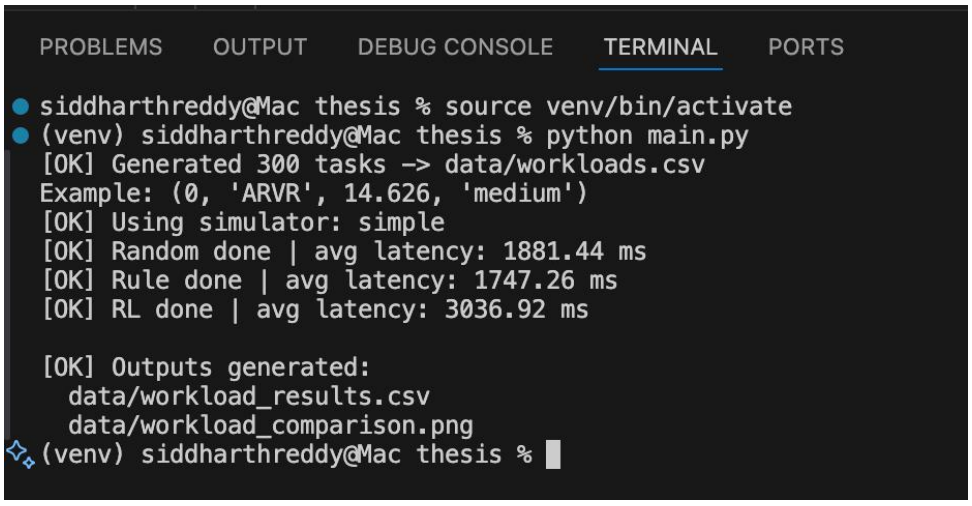

Figure 5: Local installation and test workflow (optional).

3 Usage Guide

This section explains how to run the main experiment script on the EC2 instance and verify that results have been stored in Amazon S3 and logged to CloudWatch.

3.1 Running the Main Remote Script

After completing all installation steps in Section 2, you are ready to run the remote orchestration script on the EC2 instance:

1. Ensure you are connected to the EC2 instance via VS Code Remote SSH (or terminal).

2. Navigate to the project directory and activate the virtual environment:

   ```
   cd LatencyReductionInCloud
   source venv/bin/activate
   ```

3. Run the remote orchestration script:

   ```
   python main_remote.py
   ```

The `main_remote.py` script performs the following operations:

- Checks if a workload file exists in `data/workloads.csv`; if not, generates one automatically.
- Trains the reinforcement-learning orchestrator using Q-learning with the configured number of episodes (default: 300).
- Evaluates the learned policy on the workload and records per-task latency results.
- Generates output artefacts including:
  - `rl_weights.npy` – learned Q-table weights
  - `reward_curve.png` – training progress visualization
  - `workload_results.csv` – per-task latency results
  - `workload_comparison.png` – latency comparison plot
  - `manifest.json` – run metadata including configuration and average latency
- Uploads all artefacts to the configured S3 bucket under a unique run prefix (e.g., `runs/run-20251201T120000Z/`).
- Publishes logs and custom metrics to CloudWatch for monitoring and troubleshooting.

The script execution may take several minutes depending on the number of episodes and the workload size. Progress messages will be displayed in the terminal, and logs will be written to CloudWatch in real-time.

3.2 Verifying Results in Amazon S3

After `main_remote.py` completes execution, verify that the results have been successfully saved to the S3 bucket:

1. Open the AWS Console and navigate to the *S3* service.
2. Locate the bucket configured in `config.py` (typically via the `S3_BUCKET` constant or `S3_BUCKET` environment variable).
3. Navigate to the `runs/` folder within the bucket.
4. Find the folder corresponding to your run ID (e.g., `run-20251201T120000Z/`), which is an ISO-style timestamp generated at the start of the run.
5. Confirm that the following artefacts are present:
   - `rl_weights.npy` – learned Q-table weights
   - `reward_curve.png` – training reward curve visualization
   - `workload_results.csv` – per-task latency results
   - `workload_comparison.png` – latency comparison plot
   - `manifest.json` – run metadata and configuration

If all files are present, the experiment has completed successfully and results have been uploaded to S3. You can download these files for local analysis if needed.

An illustrative screenshot of the S3 bucket contents can be included:

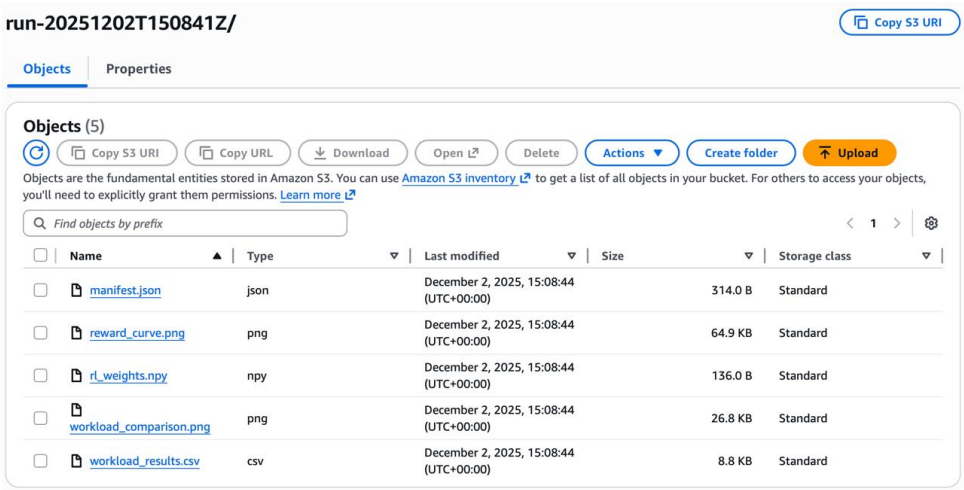

Figure 6: Artifacts for the Experiment Loaded into Amazon S3 Bucket.

3.3 Verification of CloudWatch Logs

To confirm that logs have been created in CloudWatch and monitor the executions performed on those logs follow this process:

1. Open AWS Console and select the *CloudWatch* service.

2. Select *Logs* from the left navigation menu, then choose *Log groups*.
3. Locate the log group configured for the project (typically `/aws/latency-orchestrator/logs`).
4. Click on the log group to view log streams.
5. Open the most recent log stream to review the execution logs.
6. Verify that the following information is present:
    - Training start and completion messages
    - Episode progress and reward information
    - S3 upload status for each artefact
    - Run ID and S3 prefix information
    - Any errors or warnings that occurred during execution

The CloudWatch logs provide detailed information about the experiment execution, making it easy to troubleshoot issues and monitor progress. Logs are written in real-time, so you can monitor the execution as it happens.

You may include figures showing example CloudWatch log groups and streams:

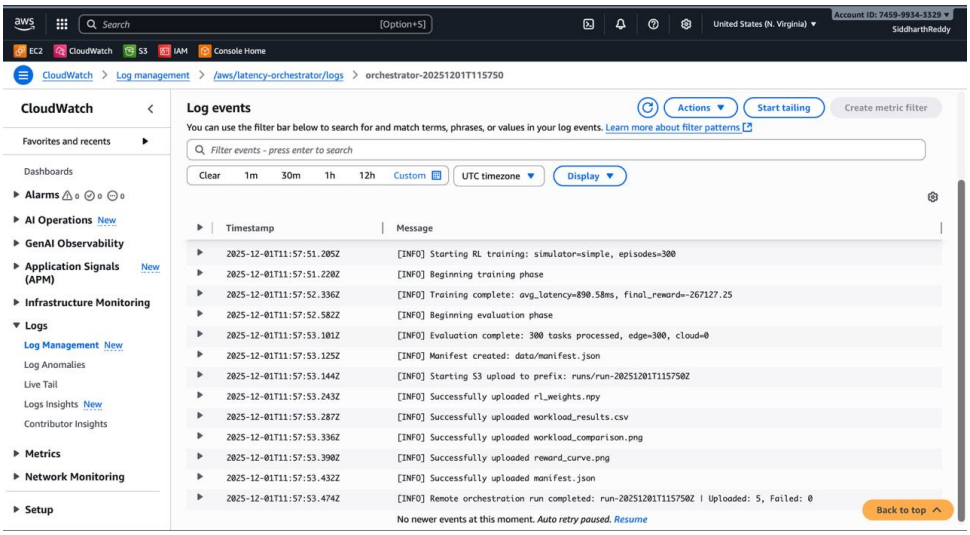

Figure 7: CloudWatch log group and streams for the orchestrator.

These metrics provide quantitative insights into system performance and can be used for monitoring and alerting.

3.4 Troubleshooting

If results are not appearing in S3 or logs are not being created in CloudWatch:

- Verify that the IAM role `LatencyOrchestrator` is correctly attached to the EC2 instance and has the required policies.
- Check that the S3 bucket name in `config.py` is correct and accessible.

- Ensure that the CloudWatch log group exists or can be created by the IAM role.
- Review the terminal output for any error messages during script execution.
- Check the CloudWatch logs for detailed error information.