National College of Ireland

Configuration Manual

Improve Cold Start Time for Stateful Serverless Functions Using Pooled Resource Injection

Tony Doolin
Student ID: 23420855

School of Computing
National College of Ireland

Supervisor: Sai Emani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Tony Doolin
Student ID:	23420855
Programme:	MSCCLOUD_A **Year:** 2024
Module:	Research Project
Supervisor:	Sai Emani
Submission Due Date:	14 September 2025
Project Title:	Improve Cold Start Time for Stateful Serverless Functions Using Pooled Resource Injection
Word Count:	~1500 **Page Count:** 12

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: 

Date: 15 September 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

Tony Doolin
Student ID: 23420855

This document describes how to reproduce the results for my research project titled “Improve Cold Start Time for Stateful Serverless Functions Using Pooled Resource Injection”

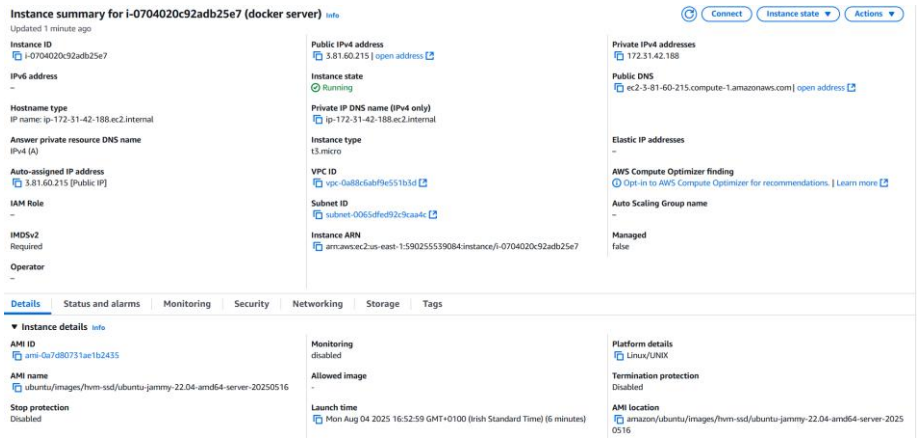
1 Environment Setup

I used the college-supplied learner lab: https://awsacademy.instructure.com/courses/123758 as my environment.

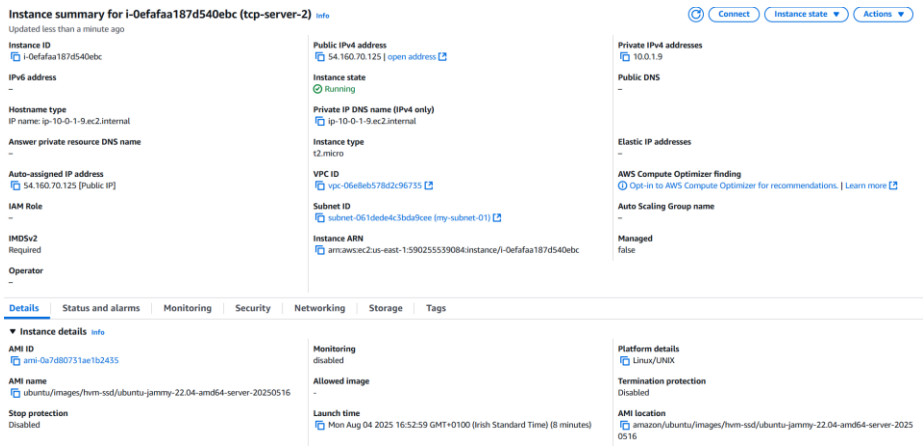
1.1 Virtual Machine Setup

The tests were run on AWS EC2 Virtual Mancine instances. I set up the following VMs on the us-east region, as that was the one available in the lab environment.

1. Docker-server



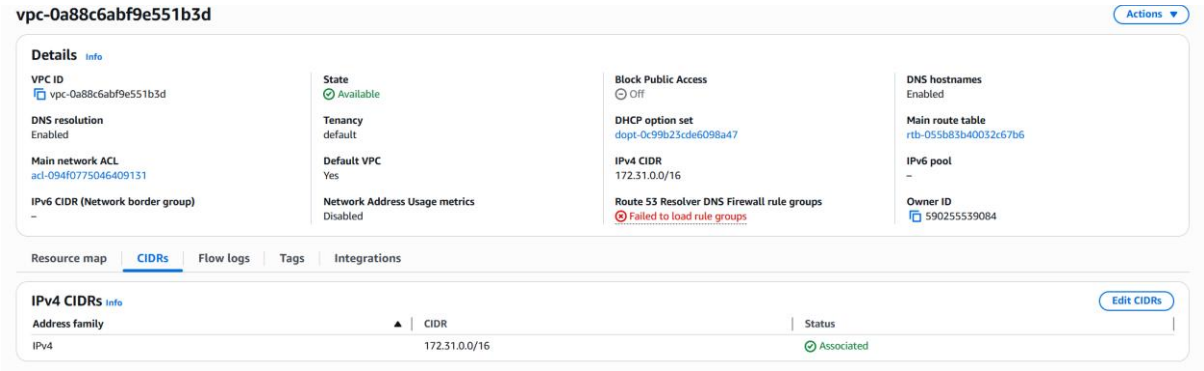
2. Tcp-server2



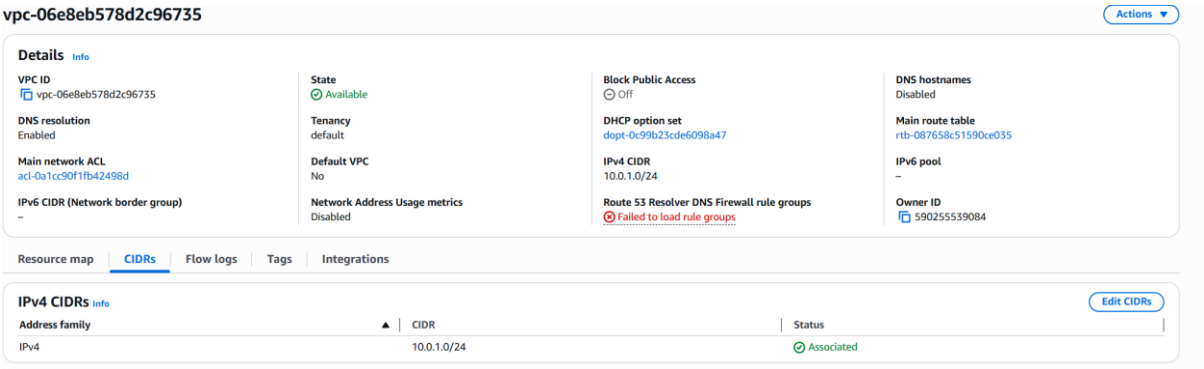
1.2 Network Setup

Each VM was set up on its own Virtual Private Cloud (VPC), with a peer connection between them

1.2.1 Docker Server VPC

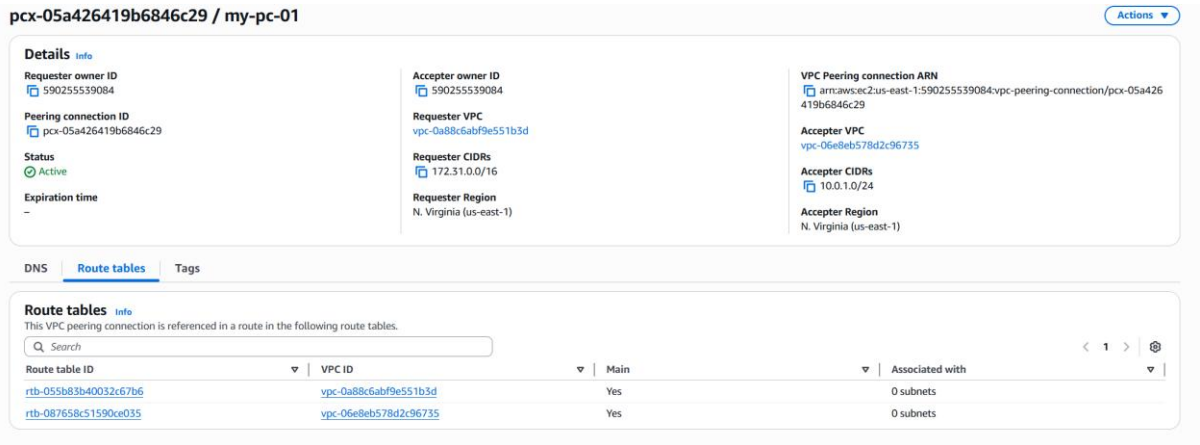


1.2.2 TCP Server VPC



1.2.3 Peering Connection

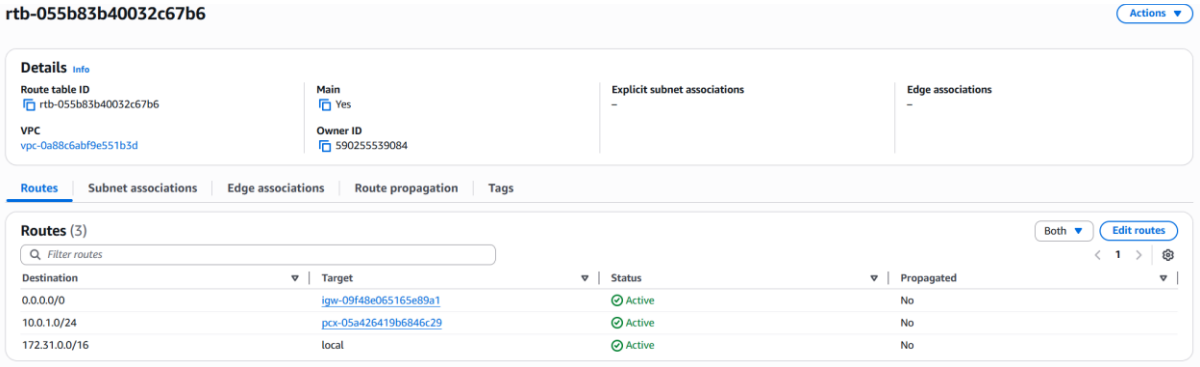
I also set up a Peering Connection between the 2 VPCS so that they can connect to each other:



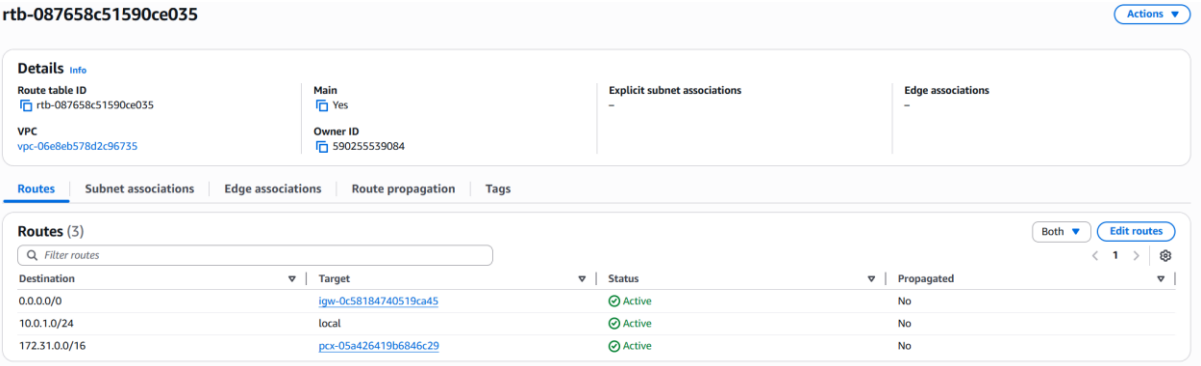
1.2.4 Routing Tables

I set up a routing table for each VPC :

1.2.4.1 Docker server Routing Table



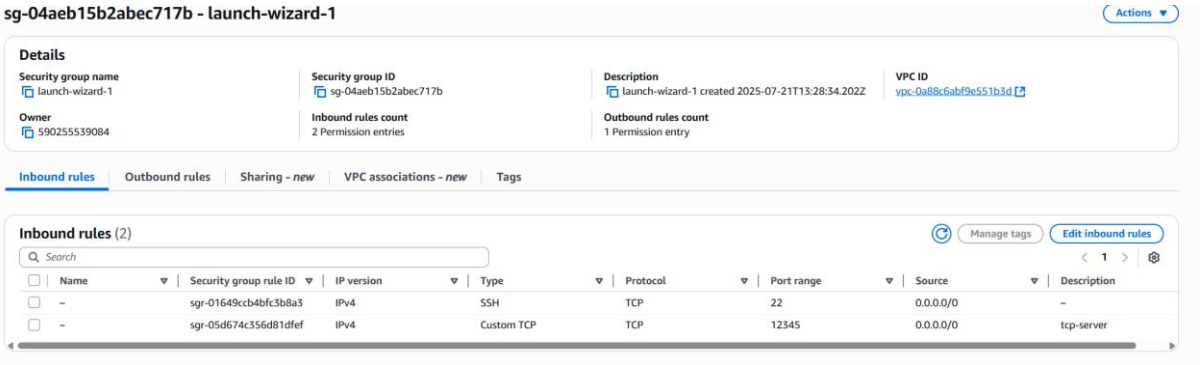
1.2.4.2 TCP Server Routing Table



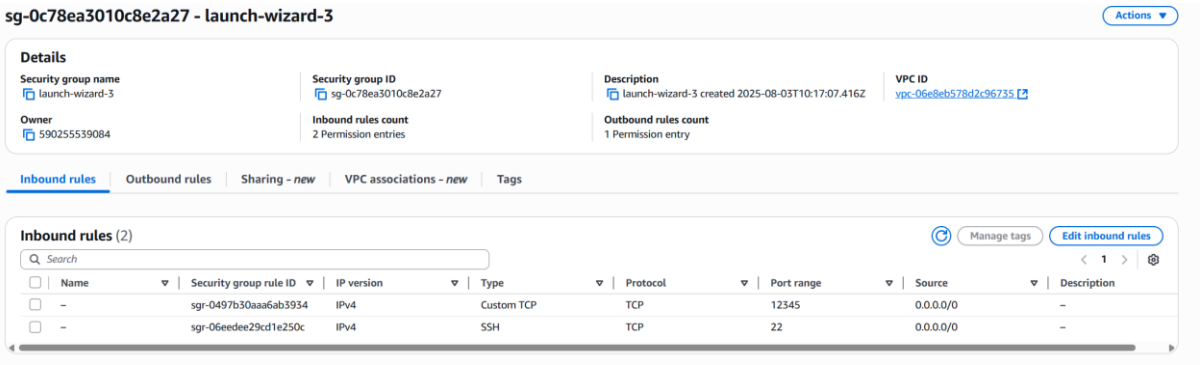
1.2.5 Security Groups

Finally, I modified the Inbound Rules section of the security group configuration of each EC2 instance so that it would accept network traffic from port 12345, which was the port I was using to connect to my server

1.2.5.1 Docker Server Security Group



1.2.5.2 TCP Server Security Group



The environment infrastructure is now setup and ready for configuration of each server

2 Operating System ulimit Configuration

In order to ensure the system has the capability to open 1000 simultaneous network connections, the max number of open file descriptors the system allows needs to be increased **on each server**

This can be done by editing the /etc/security/limits.conf

```
sudo vi /etc/security/limits.conf
```

and adding the following lines

```
*               soft    nofile          10000
*               hard    nofile          10000
```

Then restart your shell, and the new limits will be applied. This can be verified using the ulimit -n command :

```
ubuntu@ip-10-0-1-9:~$ ulimit -n
10000
```

3 Software Installation and setup

3.1 Install Docker

Docker needs to be installed on the Docker server using the following commands :

```
# Remove old versions, if any
sudo apt remove docker docker-engine docker.io containerd runc -y

# Update and install required packages
sudo apt update
sudo apt install -y ca-certificates curl gnupg

# Add Docker's GPG key
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

# Add Docker's stable repo
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update again to pull from new repo
sudo apt update

# Install Docker Engine and plugins
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin

# Activate the service
sudo systemctl start docker
sudo systemctl enable docker

# Setup docker gropup

sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker

# Finally, verify by launching a sample container
docker run hello-world
```

4 Test Execution

4.1 Deploy code

Unpack the code artifact onto both the Docker Server and the TCP Server

```
gzip -d thesis-project.zip
```

There are 2 types of test executions, a direct connection test where the Docker container connects directly with the TCP server, and an socket connection test, where the Docker container connects to a tcp-pool-manager process to receive an open TCP connection.

4.2 Direct Connection Test Execution

4.2.1 TCP server setup

On the TCP Server :

```
cd tcp-server
python3 ./tcp_server.py
```

This will start the server listening on port 12345 on all interfaces. All connections will be accepted

4.2.2 TCP Connection Delay

For some tests, it was required to implement a delay in the SYN/SYN-ACK/ACK TCP connection negotiation process. In order to do this, I wrote a script which uses the tc (Traffic Control) command to introduce a delay into the handshake by delaying the SYN-ACK response from the TCP server back to the client. In order to add the delay in to a test run, run the setup_synack_delay.sh, with the input parameter of the delay in ms. Note that this only delays connections on port 12345, not on all connections. To disable the delay, the cleanup_synack_delay.sh command can be run
Eg

```
cd tcp-server
./setup_synack_delay.sh 100 # Introduce a 100ms delay on port 12345
# Now launch the TCP server
python3 ./tcp_server.py

#
#Execute test….
#
./cleanup_synack_delay.sh # Remove the delay
```

4.2.3 Docker Server setup

On the Docker server

```
cd direct-container
./container-runner.sh 1000 # Serially launch 1000 containers and
connect to TCP server
```

Note that this will create 2 separate log files which capture the application logs and the container lifecycle logs:

```
container-logs-YYYYMMDD_HHMMSS.txt
startup-report-YYYYMMDD_HHMMSS.txt
```

4.3 Socket Connection Test Execution

4.3.1 TCP server setup

The TCP server setup is the same as for the Direct Connection test :

On the TCP Server:

```
cd tcp-server
python3 ./tcp_server.py
```

This will start the server listening on port 12345 on all interfaces. All connections will be accepted

4.3.2 Docker Server setup

To set up the socket connection test, 2 processes need to be started on the Socker server

Connection Pool Manager:

```
cd tcp-pool-manager
python3 tcp_pool_manager.py
```

This will start and make 1100 connections to the TCP server (I made the pool slightly larger than was required for the test). Note that if any configuration changes need to be made to the number of connections or the IP address or port, this can be done by editing the echo_server_config.py and restarting the process.
This process will also open an IPC socket connection in preparation for the containers connecting to it.

Socket Docker container:

To start the 1000 docker containers, there is a runner script :

```
cd socket-container
./container-runner.sh # Serially launch 1000 containers and connect to
Connection Pool Manager to get network connection
```

On completion, this will create 2 separate log files:

```
container-logs-YYYYMMDD_HHMMSS.txt
startup-report-YYYYMMDD_HHMMSS.txt
```

4.4 TLS Server setup

In order to determine the time, it takes to execute an TLS connection setup in my environment, I implemented a simple SSL server and client and executed some tests to measure the connection times. This test is not container based, and it is run as follows:

On TCP Server:

```
cd ssl-server
python3 ssl_server.py
```

On Docker server:

```
cd ssl-server
./client-runner.sh # Start 1000 ssl clients and output their
connection times
```

5 Results Gathering & Parsing

5.1 TCP Connection Time

For Connection start times

```
cat <container_logfile> | grep micro | cut -d ':' -f 2 >
connection_times.txt
```

This command will create a file containing the connection time of each execution of the container, one value per line.

5.2 Container Lifecycle Time

In order to calculate the container lifecycle time, I needed to build a script to subtract the start time from the end time. This can be run as follows and creates output with the same format as the input with 2 extra columns, the time spent executing the container function, and overall container lifecycle time.
Eg

```
results/parse_container_times.sh startup_report_20250804_115829.txt >
startup_report_direct_serial_gateway_0ms.csv
```

I then transferred the TCP connection Time and Container Startup file into Excel for further analysis. These excel spreadsheets are available in the results directory

5.3 Further Analysis

I also used a Jupyter notebook to create some box plots and calculate variance between sets of data. This notebook is in

```
analysis/analysi.ipynb
```

and can be executed using any IDE which can parse Jupyter notebooks. The notebook includes installation of the required python libraries, seaborn and scikit-learn

I also set up a python 3.10 virtual env in order to run the Jupyter scripts

References

N/A