Sarla Aviation Pvt. Ltd

Assignment: Altitude Streamer and Receiver System

1. Executive Summary

This submission provides a robust, two-part C-language solution for altitude data streaming, reception, and real-time anomaly detection. The system uses TCP sockets for concurrent operation and implements a comprehensive Continuous Integration/Continuous Deployment (CI/CD) pipeline using Git, Jenkins, and Docker to ensure code quality, reliability, and deployability across various environments.

All core requirements—data simulation, TCP streaming, anomaly detection (>100 ft/sec jump), and summary reporting—have been met. The system is containerized for simple deployment.

2. Program Code, Explanation and Results

The solution consists of two programs, built using the provided Makefile.mk: altitude_streamer and altitude receiver.

A. The Altitude Streamer (altitude_streamer.c)

Purpose: Simulates altitude data every second, saves it to a CSV file, and streams it to the receiver via a TCP socket.

Key Features:

- **Data Generation**: Uses a starting altitude (1000.0 ft), adds a steady increase (2.0 ft/sec), and introduces random turbulence (+10.0 ft) every second.
- **Anomaly Injection**: Includes command-line logic to inject a 150.0 ft jump at a specified second for testing the receiver's anomaly detection.
- **TCP Streaming**: Uses the send all helper function to ensure the entire data packet (index,altitude) is reliably sent over the TCP socket.
- **Code Reusability**: The data type streamed (double) can be easily adapted by changing the snprintf format and the samples array type.

Code: (Provided in the Github repo- https://github.com/elangomani-hash/githubemu/blob/main/Elangomani_Assignment_Sarla_Aviation/altitude_streamer.c)

B. The Altitude Receiver (altitude_receiver.c)

Purpose: Listens for incoming TCP connections, receives streaming altitude data, validates it for anomalies, and prints a final summary

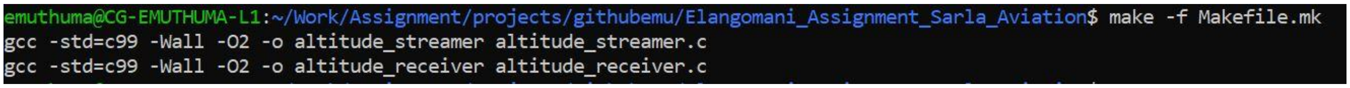
Key Features:

- **Concurrency**: Starts a non-blocking TCP server and uses accept() to handle a single client connection, fulfilling the "run concurrently" requirement.
- **Real-time Anomaly Detection**: Compares the current received altitude (alt) with the previous one (prev alt).
  - o **Validation Rule**: If alt—prev_aIt>1OO.Oft, it detects an anomaly and prints a warning.
- **Error Handling**: Implements signal handlers (SIGINT, SIGTERM) for graceful shutdown and handles client disconnection (recv() == 0).
- **Summary**: Tracks total samples and anomalies detected, printing a summary at the end.

Code: (Provided in the Github repo- https://github.com/elangomani-hash/githubemu/blob/main/Elangomani_Assignment_Sarla_Aviation/altitude_receiver.c)

C. The Build File (Makefile.mk)

The Makefile.mk is used to compile both programs from their respective C source files.
Make file execution:



```
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$ make -f Makefile.mk
gcc -std=c99 -Wall -O2 -o altitude_streamer altitude_streamer.c
gcc -std=c99 -Wall -O2 -o altitude_receiver altitude_receiver.c
```

Code: (Provided in the Github repo - https://github.com/elangomani-hash/githubemu/blob/main/Elangomani_Assignment_Sarla_Aviation/Makefile.mk)

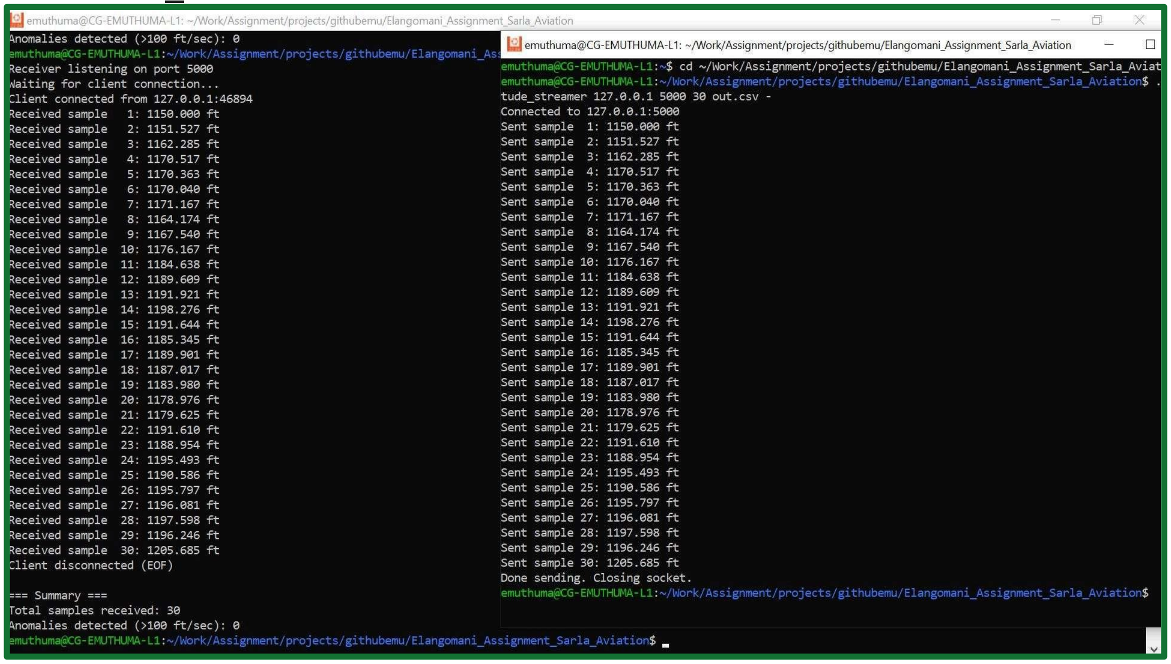
D. Results

Start the receiver and Streamer in different terminal

./altitude_receiver 5000 received.csv #

No spike injection (use ‘-’)

./altitude_streamer 127.0.0.1 5OOO 60 out.csv -


emuthuma@CG-EMUTHUMA-L1: ~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation
emuthuma@CG-EMUTHUMA-L1: ~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation

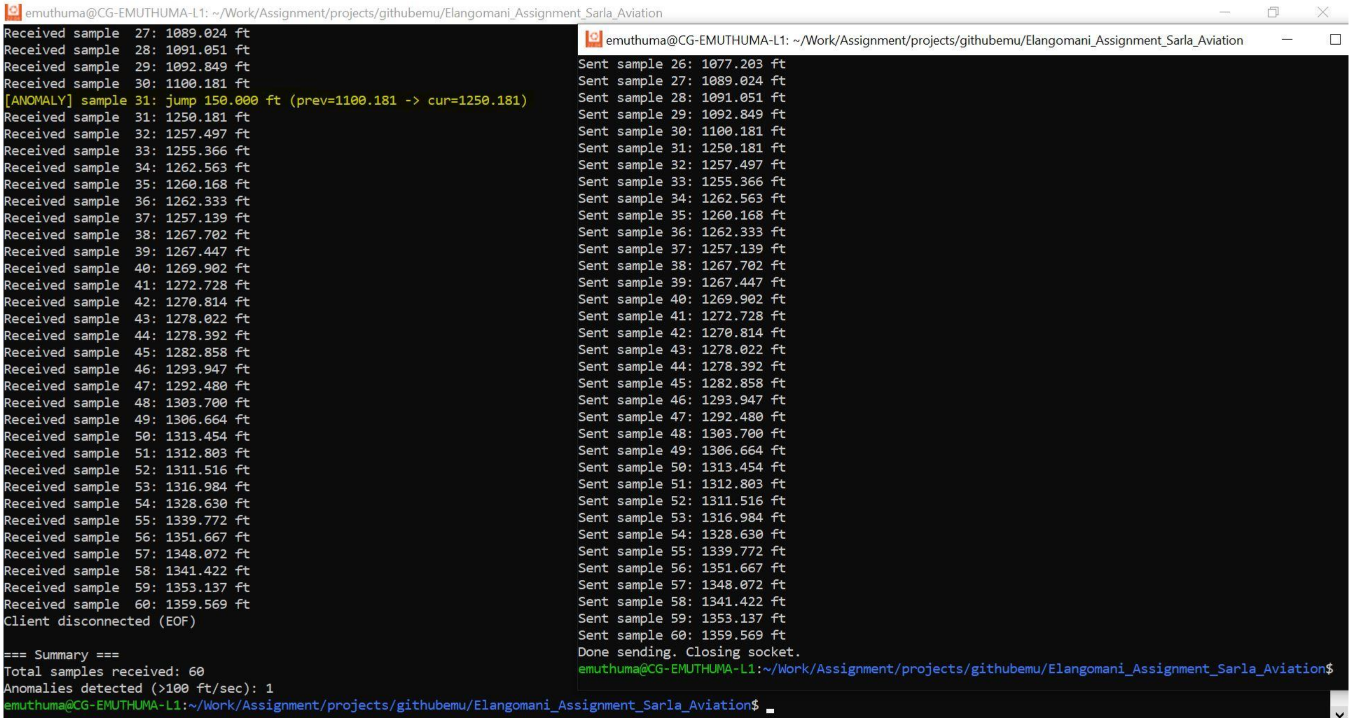
```
Anomalies detected (>100 ft/sec): 0
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_As
Receiver listening on port 5000
Waiting for client connection...
Client connected from 127.0.0.1:46894
Received sample   1: 1150.000 ft
Received sample   2: 1151.527 ft
Received sample   3: 1162.285 ft
Received sample   4: 1170.517 ft
Received sample   5: 1170.363 ft
Received sample   6: 1170.040 ft
Received sample   7: 1171.167 ft
Received sample   8: 1164.174 ft
Received sample   9: 1167.540 ft
Received sample  10: 1176.167 ft
Received sample  11: 1184.638 ft
Received sample  12: 1189.609 ft
Received sample  13: 1191.921 ft
Received sample  14: 1198.276 ft
Received sample  15: 1191.644 ft
Received sample  16: 1185.345 ft
Received sample  17: 1189.901 ft
Received sample  18: 1187.017 ft
Received sample  19: 1183.980 ft
Received sample  20: 1178.976 ft
Received sample  21: 1179.625 ft
Received sample  22: 1191.610 ft
Received sample  23: 1188.954 ft
Received sample  24: 1195.493 ft
Received sample  25: 1190.586 ft
Received sample  26: 1195.797 ft
Received sample  27: 1196.081 ft
Received sample  28: 1197.598 ft
Received sample  29: 1196.246 ft
Received sample  30: 1205.685 ft
Client disconnected (EOF)

=== Summary ===
Total samples received: 30
Anomalies detected (>100 ft/sec): 0
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$
```

```
emuthuma@CG-EMUTHUMA-L1:~$ cd ~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviat
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$ .
tude_streamer 127.0.0.1 5000 30 out.csv -
Connected to 127.0.0.1:5000
Sent sample  1: 1150.000 ft
Sent sample  2: 1151.527 ft
Sent sample  3: 1162.285 ft
Sent sample  4: 1170.517 ft
Sent sample  5: 1170.363 ft
Sent sample  6: 1170.040 ft
Sent sample  7: 1171.167 ft
Sent sample  8: 1164.174 ft
Sent sample  9: 1167.540 ft
Sent sample 10: 1176.167 ft
Sent sample 11: 1184.638 ft
Sent sample 12: 1189.609 ft
Sent sample 13: 1191.921 ft
Sent sample 14: 1198.276 ft
Sent sample 15: 1191.644 ft
Sent sample 16: 1185.345 ft
Sent sample 17: 1189.901 ft
Sent sample 18: 1187.017 ft
Sent sample 19: 1183.980 ft
Sent sample 20: 1178.976 ft
Sent sample 21: 1179.625 ft
Sent sample 22: 1191.610 ft
Sent sample 23: 1188.954 ft
Sent sample 24: 1195.493 ft
Sent sample 25: 1190.586 ft
Sent sample 26: 1195.797 ft
Sent sample 27: 1196.081 ft
Sent sample 28: 1197.598 ft
Sent sample 29: 1196.246 ft
Sent sample 30: 1205.685 ft
Done sending. Closing socket.
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$
```

Added no spike injection result video - (https://github.com/elangomani-hash/githubemu/blob/main/Elangomani_Assignment_Sarla_Aviation/result_no_injection.mp4)

No spike injection result video attached in the repo-
To test anomaly injection at second 30

```
./altitude_streamer 127.0.0.1 5000 60 out.csv 30
```



emuthuma@CG-EMUTHUMA-L1: ~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation

```
Received sample  27: 1089.024 ft
Received sample  28: 1091.051 ft
Received sample  29: 1092.849 ft
Received sample  30: 1100.181 ft
[ANOMALY] sample 31: jump 150.000 ft (prev=1100.181 -> cur=1250.181)
Received sample  31: 1250.181 ft
Received sample  32: 1257.497 ft
Received sample  33: 1255.366 ft
Received sample  34: 1262.563 ft
Received sample  35: 1260.168 ft
Received sample  36: 1262.333 ft
Received sample  37: 1257.139 ft
Received sample  38: 1267.702 ft
Received sample  39: 1267.447 ft
Received sample  40: 1269.902 ft
Received sample  41: 1272.728 ft
Received sample  42: 1270.814 ft
Received sample  43: 1278.022 ft
Received sample  44: 1278.392 ft
Received sample  45: 1282.858 ft
Received sample  46: 1293.947 ft
Received sample  47: 1292.480 ft
Received sample  48: 1303.700 ft
Received sample  49: 1306.664 ft
Received sample  50: 1313.454 ft
Received sample  51: 1312.803 ft
Received sample  52: 1311.516 ft
Received sample  53: 1316.984 ft
Received sample  54: 1328.630 ft
Received sample  55: 1339.772 ft
Received sample  56: 1351.667 ft
Received sample  57: 1348.072 ft
Received sample  58: 1341.422 ft
Received sample  59: 1353.137 ft
Received sample  60: 1359.569 ft
Client disconnected (EOF)

=== Summary ===
Total samples received: 60
Anomalies detected (>100 ft/sec): 1
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$ _
```

emuthuma@CG-EMUTHUMA-L1: ~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation

```
Sent sample 26: 1077.203 ft
Sent sample 27: 1089.024 ft
Sent sample 28: 1091.051 ft
Sent sample 29: 1092.849 ft
Sent sample 30: 1100.181 ft
Sent sample 31: 1250.181 ft
Sent sample 32: 1257.497 ft
Sent sample 33: 1255.366 ft
Sent sample 34: 1262.563 ft
Sent sample 35: 1260.168 ft
Sent sample 36: 1262.333 ft
Sent sample 37: 1257.139 ft
Sent sample 38: 1267.702 ft
Sent sample 39: 1267.447 ft
Sent sample 40: 1269.902 ft
Sent sample 41: 1272.728 ft
Sent sample 42: 1270.814 ft
Sent sample 43: 1278.022 ft
Sent sample 44: 1278.392 ft
Sent sample 45: 1282.858 ft
Sent sample 46: 1293.947 ft
Sent sample 47: 1292.480 ft
Sent sample 48: 1303.700 ft
Sent sample 49: 1306.664 ft
Sent sample 50: 1313.454 ft
Sent sample 51: 1312.803 ft
Sent sample 52: 1311.516 ft
Sent sample 53: 1316.984 ft
Sent sample 54: 1328.630 ft
Sent sample 55: 1339.772 ft
Sent sample 56: 1351.667 ft
Sent sample 57: 1348.072 ft
Sent sample 58: 1341.422 ft
Sent sample 59: 1353.137 ft
Sent sample 60: 1359.569 ft
Done sending. Closing socket.
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$
```

Added test anomaly injection at second 30 result video attached in the repo - (https://github.com/elangomani-hash/githubemu/blob/main/Elangomani_Assignment_Sarla_Aviation/result_injection.mp4)

3. CI/CD Pipeline Implementation (Git/Jenkins/Docker)

The Continuous Integration/Continuous Deployment pipeline is executed via a Jenkinsfile using a Declarative Pipeline structure. This ensures the software is tested across different versions and environments.

A. Jenkins Pipeline (Jenkinsfile)

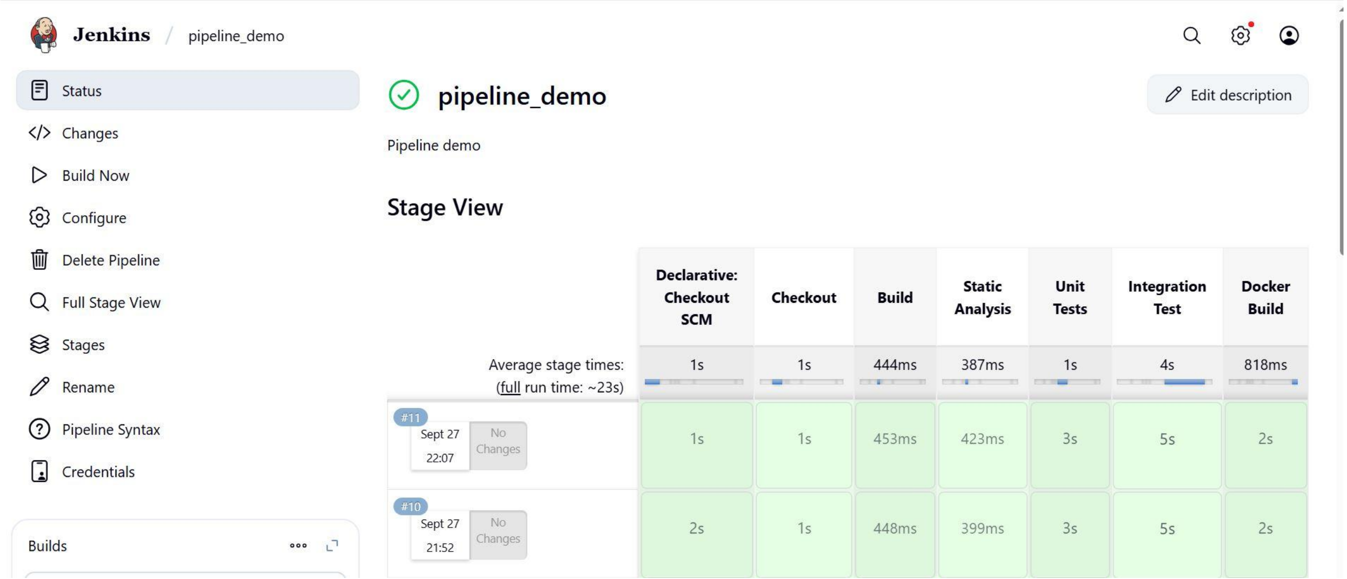
This pipeline defines the seven stages of the CI process, executed sequentially on a Jenkins agent.

```
pipeline {
  agent any
  stages {
    stage('Checkout') { steps { checkout scm } }
    stage('Build') { steps { sh 'make -f Makefile.mk' } }
    stage('Static Analysis') { steps { sh 'cppcheck --enable=all altitude_*.c || true' } }
    stage('Unit Tests') { steps { sh './tests/run_unit_tests.sh || true' } }
    stage('Integration Test') {
      steps {
        sh '''
        ./altitude_receiver 6000 received_ci.csv > server.log 2>&1 & echo $! > server.pid
        sleep 1
        ./altitude_streamer 127.0.0.1 6000 5 ci_out.csv 2
        kill $(cat server.pid) || true
        grep "Total samples received: 5" server.log
        '''
      }
    }
    stage('Docker Build') { steps { sh 'docker build -t myorg/altitude-stream:latest .' } }
  }
}
```

Phase	***Steps***	***Tests Conducted***	***Rationale***
1. Checkout	checkout scm	Repository availability, branch integrity.	Retrieves the source code from the Git repository.
2. Build	make -f Makefile.mk	Compilation success, linking.	Explicitly uses the project's Makefile.mk to compile the C programs, verifying the core executables are created.
3. Static Analysis	”cppcheck --enable=all altitude_*.c		Detects common issues: memory leaks, null pointer dereferences, unused variables, dangerous coding patterns
4. Unit Tests	./tests/run_unit_tests.sh	- Checks that altitude_streamer runs successfully. - Verifies CSV output file is created. - Ensures no crash occurs during execution.	Runs test cases to verify correctness of individual functions/modules.

5. Integration Test	Start receiver on port 6000, run streamer (5 samples), check log for samples received.	Integration Test**:** Confirms the complete end-to-end data flow (streamer —• socket —• receiver). The grep command validates the expected outcome (Total samples received: 5).	Verifies system reliability under realistic data streaming conditions.
6. Docker Build	docker build -t myorg/altitude-stream:latest .	- Provide a Dockerized application (built from source) - it can be run without needing manual compilation	Deployable version = the Docker image that can be run directly.

Pipeline Results



pipeline_demo

Pipeline demo

Stage View

	Declarative: Checkout SCM	Checkout	Build	Static Analysis	Unit Tests	Integration Test	Docker Build
Average stage times: (full run time: ~23s)	1s	1s	444ms	387ms	1s	4s	818ms
#11 Sept 27 22:07 No Changes	1s	1s	453ms	423ms	3s	5s	2s
#10 Sept 27 21:52 No Changes	2s	1s	448ms	399ms	3s	5s	2s

Pipeline output log results are attached in the repo.

B. Deployable Version and Dockerfile

The deployable version is the Docker image tagged myorg/altitude-stream:latest.

Dockerfile:

“”dockerfile:Dockerfile

Dockerfile Explanation:

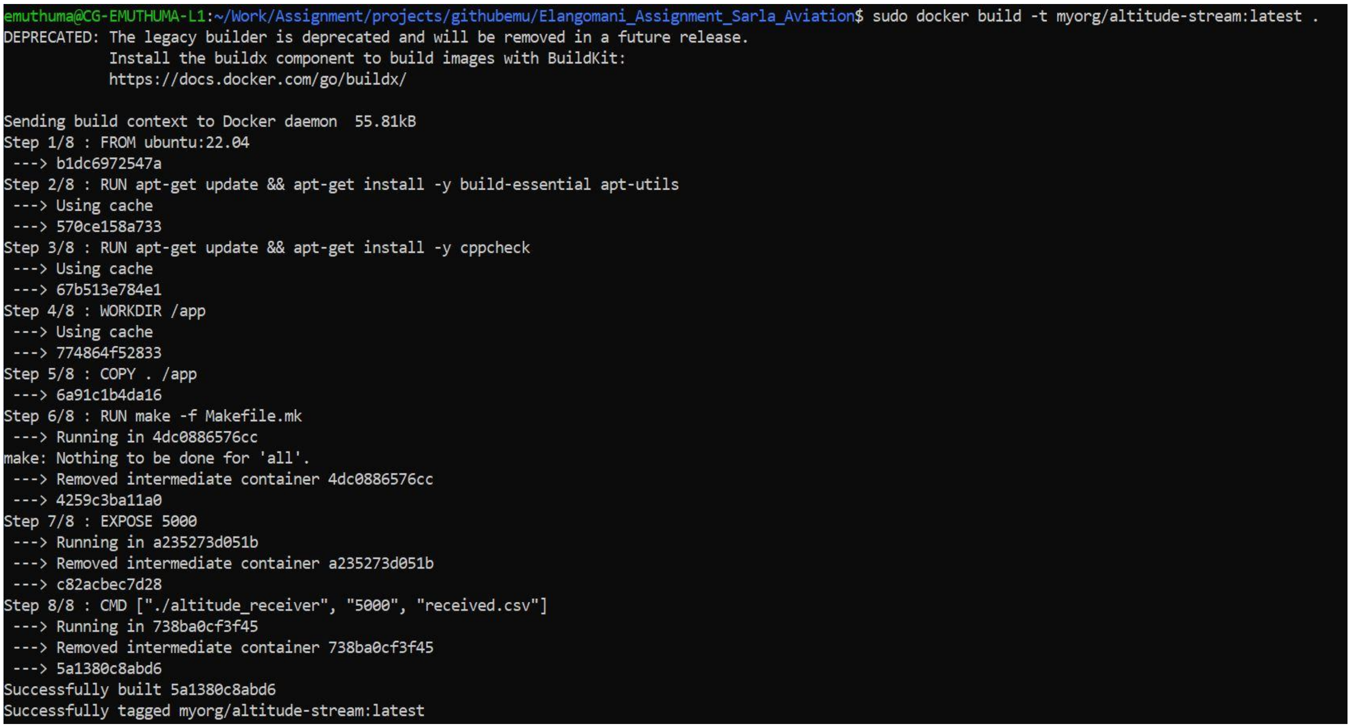
Instruction	Purpose
FROM ubuntu:22.04	Sets the base operating system to Ubuntu 22.04, providing a stable, minimal environment.
RUN apt-get update...	Installs dependencies (build-essential for C compiler/tools and apt-utils for stable package handling) and cppcheck for static analysis.
WORKDIR /app	Defines the directory inside the container where all subsequent commands will run.
COPY . /app	Copies all project files (source code, Makefile.mk) from the Jenkins workspace into the container's working directory.
RUN make -f Makefile.mk	Executes the build using the specified Makefile.mk, compiling the two C programs into executable binaries (altitude receiver and altitude streamer).
EXPOSE 5000	Documents that the application inside the container will listen on port 5000.
CMD ["./aItitude receiver", "5000", "received.csv"]	Default entry point when the container is run, starting the receiver on port 5000 and logging output to a CSV file.

Docker build:

The Docker Build stage packages your verified code into a self-contained Docker image, making it portable, reproducible, and ready for deployment.

This command builds a Docker image from the Dockerfile in your current directory, tags it as myorg/altitude-stream:latest

```
sudo docker build -t myorg/altitude-stream:latest .
```



```
emuthuma@CG-EMUTHUMA-L1:~/Work/Assignment/projects/githubemu/Elangomani_Assignment_Sarla_Aviation$ sudo docker build -t myorg/altitude-stream:latest .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  55.81kB
Step 1/8 : FROM ubuntu:22.04
 ---> b1dc6972547a
Step 2/8 : RUN apt-get update && apt-get install -y build-essential apt-utils
 ---> Using cache
 ---> 570ce158a733
Step 3/8 : RUN apt-get update && apt-get install -y cppcheck
 ---> Using cache
 ---> 67b513e784e1
Step 4/8 : WORKDIR /app
 ---> Using cache
 ---> 774864f52833
Step 5/8 : COPY . /app
 ---> 6a91c1b4da16
Step 6/8 : RUN make -f Makefile.mk
 ---> Running in 4dc0886576cc
make: Nothing to be done for 'all'.
 ---> Removed intermediate container 4dc0886576cc
 ---> 4259c3ba11a0
Step 7/8 : EXPOSE 5000
 ---> Running in a235273d051b
 ---> Removed intermediate container a235273d051b
 ---> c82acbec7d28
Step 8/8 : CMD ["./altitude_receiver", "5000", "received.csv"]
 ---> Running in 738ba0cf3f45
 ---> Removed intermediate container 738ba0cf3f45
 ---> 5a1380c8abd6
Successfully built 5a1380c8abd6
Successfully tagged myorg/altitude-stream:latest
```

4. Testing Software Across different Versions

To test the software across different versions, the CI/CD pipeline implements the following

strategy:

- **Version Control:** Every change (version) is tracked in Git.
- **Branching Strategy**: Use Git flow (e.g., feature, merge to develop, then merge to main). Jenkins builds are triggered on every commit to develop.
- **Build Artifacts**: The final Docker image is tagged not just with :latest, but with the Git commit SHA or a version number (e.g., myorg/altitude-stream:v1.2.3 or myorg/altitude-stream:COMMIT SHA).
- **Test Isolation:** By running tests inside a clean, defined Jenkins workspace, we guarantee no legacy files or environment changes from previous versions affect the current run.

5. Potential Pitfalls and Future Work

Potential Pitfall	Resolution/Future Work
Single Client Limitation	The receiver uses accept(), blocking until a client connects. Future work should implement non-blocking sockets, select()/poll(), or multi-threading to handle multiple simultaneous streamers.
No Dynamic Memory for Reception	The receiver uses fixed buffers (RECV\ BUF, LINE\ BUF). A large, sudden data burst could cause buffer overflows. Future work requires dynamic resizing of line buffer.
Hardcoded Anomaly Threshold	The 100.0 ft anomaly threshold is hardcoded in the receiver. This should be configurable via command-line argument or environment variable.