# Architecture

Packet Flow:

Godot Clients ↔ Cloudflare Tunnel ↔ Azure VM

Azure VM:

Nginx ↔ Gunicorn ↔ Flask App ↔ SQLite DB

* **Godot Clients** send HTTPS requests to findio.me (heroicstudio.xyz domain is too new, doesnt work from la trobe network).
* **Cloudflare Tunnel** handles SSL termination and routes the packets to the Azure VM (cannot access Azure IP directly, untrusted by la trobe network).
* **Nginx** on the Azure VM handles the request and reverse proxies the packets to Gunicorn.
* **Gunicorn** runs multiple Flask worker processes.
* **Flask App** implements the API, performs validation, and interacts with the SQLite database.
* **SQLite** stores buildings, which store floors, which store rooms. Temperature readings reference room ids.

# 🚀 Redline Server Deployment Guide

Welcome to the setup guide for the **Redline Server**. This document will walk you through setting up:

1. 🧱 Nginx Installation (Web Server)
2. 🌐 Cloudflare Domain Setup (Required Before Tunnel)
3. 🌩️ Cloudflare Tunnel (Users connect to trusted, unblocked IP over HTTPS)
4. 🔐 Logging into GitHub and cloning repo via SSH Key (Required - Private Repo)
5. 🔑 .env Secret Key File
6. 🧪 Python Virtual Environment and Server Setup
7. 🦄 Gunicorn Systemd Service (Backend Daemon)
8. ↩ Nginx Reverse Proxy Config (route requests to Gunicorn)

## 1. 🧱 Nginx Installation (Web Server)

**Purpose:** Serves your frontend and acts as a reverse proxy for your backend API.

sudo apt install nginx -y  
sudo systemctl start nginx  
sudo systemctl enable nginx

**Optional:** Add a basic placeholder page. At this point you can navigate to your server's IP and see the page, or the Nginx default setup page if you don't have a custom index.html.

echo '<h1>Insert Digital Twin Here</h1>' | sudo tee /var/www/html/index.html

## 2. 🌐 Cloudflare Domain Setup (Required Before Tunnel)

**Purpose:** Configure Cloudflare to manage your domain so that the tunnel can route traffic through it.

### Add domain to Cloudflare:

1. Go to [<https://dash.cloudflare.com>](https://dash.cloudflare.com) and log in.
2. Click **Add a Site**.
3. Enter your domain and click **Add site**.
4. Choose the **Free** plan.

### Set Cloudflare Nameservers:

After adding the site, Cloudflare will provide two nameservers, for example:

* meera.ns.cloudflare.com
* watson.ns.cloudflare.com

Go to your domain registrar (where you purchased the domain), and update your domain's nameservers to the ones provided by Cloudflare.

### Wait for propagation:

This can take from a few minutes to a few hours. You can verify that the change is active by running:

dig NS ENTER-YOUR-DOMAIN-HERE

You should see the Cloudflare nameservers in the output.

### Continue only after this is complete.

Once your domain is using Cloudflare nameservers, proceed to the Cloudflare Tunnel section.

Note: La Trobe, and potentially other networks, do not allow just registered domains to be connected to.

## 3. 🌩️ Cloudflare Tunnel (Users connect to trusted, unblocked IP over HTTPS)

**Purpose:** Expose your local server securely to the internet using Cloudflare Tunnel, so that any requests to the domain go to a trusted cloudflare IP, not a potentially untrusted IP of the server, which is blocked on La Trobe network. Note: Cloudflare Tunnel handles TLS and makes your server run on HTTPS, no need to use Let's Encrypt or any other steps.

### Install cloudflared:

wget https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-linux-amd64.deb  
sudo dpkg -i cloudflared-linux-amd64.deb

### Authenticate and create tunnel:

cloudflared tunnel login  
cloudflared tunnel create heroicstudio3

Note: Copy the generated tunnel ID and credentials file path.

### Configure DNS routing:

cloudflared tunnel route dns heroicstudio3 ENTER-YOUR-DOMAIN-HERE

### Prepare your Cloudflare config:

sudo mkdir -p /etc/cloudflared  
sudo cp cloudflare-config/config.yml /etc/cloudflared/config.yml  
sudo nano /etc/cloudflared/config.yml

In the editor, replace YOUR-TUNNEL-ID.json with the actual filename of the credentials JSON created during:

cloudflared tunnel create heroicstudio3

Also change the username mark to your linux username.

Final structure:

tunnel: heroicstudio3  
credentials-file: /home/mark/.cloudflared/YOUR-TUNNEL-ID.json  
  
ingress:  
 - hostname: ENTER-YOUR-DOMAIN-HERE  
 service: http://localhost:80  
 - service: http\_status:404

Then run:

sudo cloudflared service install  
sudo systemctl start cloudflared  
sudo systemctl enable cloudflared

### Test the tunnel:

curl https://ENTER-YOUR-DOMAIN-HERE

Expected output:

<h1>Insert Digital Twin Here</h1>

## 4. 🔐 Logging into GitHub and cloning via SSH Key (Required - Private Repo)

**Purpose:** Authenticate securely with GitHub for cloning and pushing code.

### Generate and copy your key:

ssh-keygen -t ed25519 -C "azure"  
cat ~/.ssh/id\_ed25519.pub

Paste the output into:

GitHub → **Settings** → **SSH and GPG Keys** → **New SSH Key**

Then clone the repo using SSH:

git clone git@github.com:markipol/dih-digital-twin.git

## 5. 🔑 .env Secret Key File

**Purpose:** Store the API key used by the server.

Create a file at:

dih-digital-twin/server/.env

Contents:

SECRET\_KEY = <your-secret-key>

Do not commit this file to GitHub, it is already in the .gitignore to ignore this file anywhere.

## 6. 🧪 Python Virtual Environment and Server Setup

**Purpose:** Run the Flask API backend in a clean, isolated Python environment.

### Install and create venv:

sudo apt install python3.12-venv  
cd ~  
mkdir redline-venv  
python3 -m venv redline-venv  
source redline-venv/bin/activate

Prompt should now look like:

(redline-venv) mark@heroic-studio-3:~$

### Install Python dependencies:

pip install flask flask-cors requests python-dotenv gunicorn

Or use:

pip install -r requirements.txt

## 7. 🦄 Gunicorn Systemd Service (Backend Daemon)

**Purpose:** Run your Flask app continuously in the background using Gunicorn and systemd.

### Create the service file:

cd dih-digital-twin/server  
sudo nano heroicstudio3.service

Example service file using username of mark:

[Unit]  
Description=Gunicorn for Heroic Studio 3 Flask App  
After=network.target  
  
[Service]  
User=mark  
Group=mark  
WorkingDirectory=/home/mark/dih-digital-twin/server  
Environment="PATH=/home/mark/redline-venv/bin"  
ExecStart=/home/mark/redline-venv/bin/gunicorn -w 3 -b 127.0.0.1:8000 wsgi:app  
  
Restart=always  
  
[Install]  
WantedBy=multi-user.target

### Install and start the service:

sudo cp heroicstudio3.service /etc/systemd/system/heroicstudio3.service  
sudo systemctl daemon-reexec  
sudo systemctl daemon-reload  
sudo systemctl enable heroicstudio3  
sudo systemctl start heroicstudio3

## 8. ↩ Nginx Reverse Proxy Config (route requests to Gunicorn)

**Purpose:** Route frontend traffic to your backend app.

Overwrite the default config with the provided config:

sudo cp nginx-config /etc/nginx/sites-available/default  
sudo systemctl reload nginx

Example config:

server {  
 listen 80 default\_server;  
 listen [::]:80 default\_server;  
  
 location / {  
 proxy\_pass http://127.0.0.1:8000;  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 }  
}

The server is now live at your domain! Replace ENTER-YOUR-DOMAIN-HERE with your actual domain in the relevant commands.

# Testing

Install pytest and pytz inside the venv

(redline-venv) mark@heroic-studio-3:~$ pip install pytest pytz

Run the tests (the -v switch shows ticks and test names).

(redline-venv) mark@heroic-studio-3:~/dih-digital-twin$ pytest -v server/test\_api\_report.py

# Database Schema

CREATE TABLE buildings (  
 name TEXT NOT NULL PRIMARY KEY,  
 display\_name TEXT  
);  
  
CREATE TABLE floors (  
 id INTEGER PRIMARY KEY,  
 building\_name TEXT NOT NULL,  
 display\_name TEXT NOT NULL,  
 FOREIGN KEY (building\_name) REFERENCES buildings(name)  
);  
  
CREATE TABLE rooms (  
 id TEXT PRIMARY KEY,  
 floor\_id INTEGER NOT NULL,  
 display\_name TEXT NOT NULL,  
 FOREIGN KEY (floor\_id) REFERENCES floors(id)  
);  
  
CREATE TABLE temperature\_readings (  
 room\_id TEXT NOT NULL,  
 timestamp REAL NOT NULL,  
 temperature REAL NOT NULL,  
 PRIMARY KEY (room\_id, timestamp),  
 FOREIGN KEY (room\_id) REFERENCES rooms(id)  
);

# API Endpoints

# Test API endpoints

## GET /api/whoami

Debug endpoint to return the server’s public IP (via icanhazip.com).

* **Success** (200):
* { "ip": "203.0.113.45" }
* **Error** (500):
* { "error": "<error message>" }

## GET /api/check\_write

* **Headers**:
  + X-API-Key: <WRITE\_KEY> (required)
* **Success** (200):
* { "write\_access": true }
* **Errors** (500):
  + 401 Unauthorized – missing/incorrect API write key
  + 500 Internal Server Error – other error, further error information given in json response

## GET /api/check\_read

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **Success** (200):
* { "read\_access": true }
* **Errors** (500):
  + 401 Unauthorized – missing/incorrect API read key
  + 500 Internal Server Error – other error, further error information given in json response

# Write endpoint

Requires the correct write key, put in the X-API-Key header as "X-API-Key:

## POST /api/report

Report a new temperature reading.

* **Headers**:
  + X-API-Key: <WRITE\_KEY> (required)
  + X-Dry-Run: true|false (optional; if true, validates only, does not store)
* **Body**: JSON payload
* {  
   "room\_id": 123,  
   "timestamp": "1749003703", // Unix timestamp  
   "temperature": 21.5  
  }
* **Success** (200):
* { "status": "ok" }
* **Dry Run** (200):
* {  
   "status": "validated",  
   "dry\_run\_result": "Dry run enabled via X-Dry-Run header. No data stored. Request is validly formatted"  
  }
* **Errors**:
  + 401 Unauthorized – missing/incorrect API key
  + 400 Bad Request – missing fields, invalid timestamp format, timestamp out of range
  + 404 Not Found – room ID does not exist
  + 500 Internal Server Error – – other error, further error information given in json response

# Read endpoints

## GET /api/floor/<floor\_id>/latest

Get the latest temperature for each room on a given floor.

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **URL Parameters**:
  + floor\_id (integer) – floor identifier
* **Success** (200):
* [  
   { "room\_id": 1, "display\_name": "Conf Room", "timestamp": 1620000000.0, "temperature": 22.1 },  
   { "room\_id": 2, "display\_name": "Office A", "timestamp": 1620000300.0, "temperature": 20.8 }  
  ]
* **Errors**:
  + 404 Not Found – floor does not exist
  + 500 Internal Server Error

## GET /api/room/<room\_id>/last24h

Get all readings for a specific room in the last 24 hours.

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **URL Parameters**:
  + room\_id (room id string)
* **Success** (200):
* [  
   { "timestamp": 1620000000.0, "temperature": 22.1 },  
   { "timestamp": 1620070000.0, "temperature": 21.7 }  
  ]
* **Errors**:
  + 404 Not Found – room does not exist
  + 500 Internal Server Error

## GET /api/room/<room\_id>/day/<date>/<tz>

Get all readings for a specific room in the specified date (12am-12am).

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **URL Parameters**:
  + room\_id - room id string
  + date - date in ISO YYYY-MM-DD format
  + tz - timezone in the [tz format](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones), for example "Australia/Melbourne"
* **Success** (200):
* [  
   { "timestamp": 1620000000.0, "temperature": 22.1 },  
   { "timestamp": 1620070000.0, "temperature": 21.7 }  
  ]
* **Errors**:
  + 404 Not Found – room does not exist
  + 400 Bad Request - date format wrong or in the future
  + 500 Internal Server Error – other error, further error information given in json response

### GET /api/floor/<floor\_id>/last24h/human\_readable/<tz>

e.g. /api/floor/1/latest/human\_readable/Australia/Melbourne

Get the latest readings for all rooms on a floor, with human readable time decorations.

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **URL Parameters**:
  + floor\_id - Floor int
  + tz Timezone in the [tz format](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones), for example "Australia/Melbourne"
* **Success** (200):
* {  
   "readings": [  
   {  
   "display\_name": "101 - Central Space",  
   "local\_time": "2025-06-04T12:56:04.121600+10:00",  
   "room\_id": "101",  
   "temperature": 21.2,  
   "timestamp": 1749005764.1216,  
   "utc\_offset\_hours": 10.0  
   },  
   {  
   "display\_name": "101A - HOC Cafe",  
   "local\_time": "2025-06-05T18:30:03.613552+10:00",  
   "room\_id": "101A",  
   "temperature": 16.12,  
   "timestamp": 1749112203.613552,  
   "utc\_offset\_hours": 10.0  
   },  
   ...  
   ],  
   "timezone": "Australia/Melbourne",  
   "utc\_offset\_hours": 10.0  
  }
* **Errors**:
  + 404 Not Found – Floor does not exist
  + 500 Internal Server Error – other error, further error information given in json response

## GET /api/room/<room\_id>/day/<date>/human\_readable/<tz>

e.g. /api/room/101A/day/2025-06-04/human\_readable/Australia/Melbourne

Get all readings for a specific room on the specified day (12am-12am), with human readable time decorations.

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **URL Parameters**:
  + room\_id - Room id string
  + date - date in ISO YYYY-MM-DD format
  + tz - Timezone in the [tz format](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones), for example "Australia/Melbourne"
* **Success** (200):
* {  
   "readings": [  
   {  
   "local\_time": "2025-06-04T14:00:03.573216+10:00",  
   "temperature": 20.93,  
   "timestamp": 1749009603.573216,  
   "utc\_offset\_hours": 10.0  
   },  
   {  
   "local\_time": "2025-06-04T13:30:04.046113+10:00",  
   "temperature": 20.81,  
   "timestamp": 1749007804.046113,  
   "utc\_offset\_hours": 10.0  
   }  
   ],  
   "timezone": "Australia/Melbourne",  
   "utc\_offset\_hours": 10.0  
   }
* **Errors**:
  + 404 Not Found - Room does not exist
  + 400 Bad Request - date format wrong or in the future
  + 500 Internal Server Error

## GET /api/room/<room\_id>/latest\_day/<tz>

e.g. /api/room/101/latest\_day/Australia/Melbourne

Get the latest day's readings for a specific room (12am-12am), also returns latest\_day\_iso (ISO date string for the latest day with data).

* **Headers**:
  + X-API-Key: <READ\_KEY> (required)
* **URL Parameters**:
  + room\_id - Room id string
  + tz - Timezone in the [tz format](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones), for example "Australia/Melbourne"
* **Success** (200):
* {  
   "readings": [  
   {  
   "temperature": 22.06,  
   "timestamp": 1749391203.477826  
   },  
   {  
   "temperature": 22.06,  
   "timestamp": 1749394802.997411  
   },  
   {  
   "temperature": 22.06,  
   "timestamp": 1749398403.407318  
   },  
   {  
   "temperature": 22.06,  
   "timestamp": 1749402003.937262  
   }  
   ],  
   "latest\_day\_iso": "2025-06-09"  
  }
* **Errors**:
  + 404 Not Found - Room does not exist
  + 400 Bad Request - timezone does not exist, date format wrong or in the future
  + 500 Internal Server Error

# Contributing

We welcome contributions to improve the Flask API and its deployment setup. Please follow the workflow and guidelines below to ensure smooth collaboration.

## Workflow

1. Fork the repository on GitHub and clone it locally.
2. Create a new feature branch for your changes.
3. Make your changes, ensuring the app runs without errors and all endpoints behave as expected.
4. Run tests to verify functionality (manual or automated if available).
5. Open a pull request with a clear description of your changes.

## Development Guidelines

* Follow standard Python and Flask best practices.
* Keep code modular and reusable (use Blueprints or helper modules if needed).
* Add comments and docstrings to explain new logic or complex sections.
* If modifying the database schema, also update schema.sql.
* If adding new endpoints, include them in the **API Endpoints** section of the README.