Contents:

1 Overview 3
   1.1 System architecture ..................................................... 4
   1.2 Links ................................................................ 4
   1.3 Milestones & Backlogs ................................................. 5

2 Deployment 7
   2.1 Step-by-step deployment guide ......................................... 7
   2.2 Configuring a Drone for CloudStation .................................. 9
   2.3 AWS RDS (Aurora engine) - Experimental ................................. 10
   2.4 Authors .................................................................. 10
   2.5 Troubleshooting ............................................................... 11

3 How to Use 13

4 How to Contribute 15
   4.1 Technologies ................................................................. 15
   4.2 General Notes ............................................................... 16
   4.3 Feature Improvements ...................................................... 16

5 Contributors 17
   5.1 Authors .................................................................. 17
   5.2 Advisor .................................................................. 17
   5.3 Contributing Developers ................................................... 17
   5.4 Contact Information ......................................................... 17

6 License 19
The goal of this project is to develop a cloud-based ground control station for Ardupilot drones. The web app is designed to be easily scalable so users can control multiple drones at the same time. The project is still under development and we would love to have more developers joining us!

Website: CloudStation.
CHAPTER 1

Overview

LICENSE GitHub
1.1 System architecture

1.2 Links

[J78] Lyuyang Hu ; Omkar Pathak ; Zeyu He ; Hunkyu Lee ; Mina Bedwany ; Jace Mica ; Peter J. Burke “Cloud-Station”: A Cloud-based Ground Control Station for Drones ” IEEE Journal on Miniaturization for Air and Space Systems (2020)

CloudStation - UCI MCS/MSWE Capstone Presentation ‘20
1.3 Milestones & Backlogs

[x] System architecture and prototype (8/31/19)
[x] Rewrite mavlink streaming code (9/5/19)
[x] Add map to html (10/15/19)
[x] Deploy on AWS (10/15/19) We are online!!!
[x] Build hardware stack with Omnibus F4 and Raspberry Pi (10/30/19)
[x] Two way communication between vehicle and the server (11/5/19)
[x] User authentication (11/16/19)
[x] Mark drone location on map (11/24/19)
[x] Migrate to AWS RDS (2/25/20)
[x] Communicate with multiple drones at the same time (1/23/20)
[ ] Distinguish different users. Only send vehicle updates to authorized users
[x] Develop an improved UI for telemetry data
[x] Use a more robust background tasks solution (we don’t use django_background_tasks any more!)
[x] Get automated SITL drone swarms running with CloudStation
[x] Add customizable telemetry options
[] Link customizable telemetry options to specific users
The instructions describe our setup process on an EC2 instance running Ubuntu 18.04 LTS. The steps should be similar if you use a server with Linux Distributions.

CloudStation source code

2.1 Step-by-step deployment guide

2.1.1 Prerequisites

Before you start, you will need an AWS account (free trial version is OK). The following deployment steps have been developed for a fresh EC2 instance. Although deployment on a local machine is possible, it may require additional steps and configuration changes not covered in the guide.

You will also need a free MapBox account. Make sure you can find your Mapbox public token, located on the home page after logging in.

2.1.2 Deployment Steps

For a step by step video guide, see here.

1. Launch an EC2 instance on AWS with Ubuntu 18.04 LTS
   • t2.micro (free-tier eligible) is good enough to test the deployment.
   • Step 6: Configure Security Group (AWS EC2 Console)
     – SSH (TCP) Port:22 Source:My IP
     – HTTP(TCP) Port:80 Source:Anywhere
     – Custom UDP Rule Port:14550 Source:Anywhere
MAVLink (vehicle messages) is routed to 14550 via UDP in the current configuration. Any available port can be used instead of 14550. If you want to connect multiple vehicles, enter a range of ports, such as 14550-14560.

- Create or use existing key pairs. This is used for SSH.

2. Associate an Elastic IP to the EC2 (Please take note of the IP/DNS address, you will need it in step 3 and 6)
   - "An Elastic IP address is a static IPv4 address". It is a public address we use so the IP address of the deployed CloudStation will stay the same.
   

   - This is only a temporary solution but it serves our testing purpose well. This step is optional but if elastic IP is not used, the server IP address will change from time to time.

3. Connect to Linux instance using an SSH client

4. Set up EC2 (run only once)

   ```bash
   cd ~
git clone https://github.com/CloudStationTeam/cloud_station_deployment.git
bash ~/cloud_station_deployment/setup_server.sh
   ```

   This deploys an instance of the CloudStation web server based on the latest commit. Note that the latest commit may be developmental. For a more stable version, you can run setup_server.sh with a flag to indicate a certain release tag, e.g. bash ~/cloud_station_deployment/setup_server.sh --tag=v2.0.

   - You can see the list of releases here. The tag for the release is indicated by the text following the tag icon to the left of the release icon, e.g. v2.0 or v1.0.

   - setup_server.sh does the following
     1. Update Ubuntu
     2. Install NGINX and docker
     3. Clone cloud_station_web source code
     4. Set up a Python virtual environment (install dependencies)

5. Modify cloud_station_web/webgms/settings.py

   - Add EC2 IP address/DNS to ALLOWED_HOST
     
     - DNS example: “ec2-xx-xx-xxx-xxx.us-west-1.compute.amazonaws.com” (it should be a string, please do not forget the quotation marks)

   - Set DEBUG to False

   - Set MAPBOX_PUBLIC_KEY to your Mapbox public token

6. Modify cloud_station_deployment/nginx.conf

   - add EC2 IP/DNS address to Line 68: server_name ec2-xx-xx-xxx-xxx.us-west-1.compute.amazonaws.com

7. Configure NGINX, Daphne and Django (run only once)

   ```bash
   bash ~/cloud_station_deployment/configure_web_server.sh
   ```

   The script does the following:

   1. Write database migrations
   2. Collect staticfiles to ~cloud_station_web/static
   3. Configure NGINX with nginx.conf
4. Configure systemctl to automatically run Daphne as a service (daphne.service)
5. Download redis and start running redis in a docker container
8. In your web browser, go to your EC2 instance’s DNS address (ec2-xx-xx-xxx-xxx.us-west-1.compute.amazonaws.com) and you should see the CloudStation website.

2.1.3 Redeployment

To reload the server (after a code update)

- Run `bash ~/cloud_station_deployment/reload_server.sh`
- The script does the following:
  1. Pull latest version of the src code
  2. Write database migrations
  3. Collect staticfiles
  4. Reload NGINX and Daphne
  5. Run django_background_tasks

2.1.4 Testing Using SITL

To test CloudStation with a simulated drone instead of a real drone, you can install and run SITL. If you have a Windows computer running Cygwin, you can also use our SITL deployment script to automate running multiple SITL instances.

2.1.5 Restarting Ubuntu

If you need to restart Ubuntu, run the following: `bash ~/cloud_station_deployment/configure_web_server.sh` This procedure is needed to restart redis.

2.2 Configuring a Drone for CloudStation

In order for the drone to connect to CloudStation, the drone needs to be set up to send Mavlink packets over UDP to the CloudStatoin IP address. The UDP port is also the “ID” of the drone in the CloudStation UI. As far as Cloudstation is concerned, it does not matter how the drone is configured to send/receive Mavlink traffic over UDP.

There are many ways to configure the drone to do this. The easiest is to have a on-board “companion computer”. For example, the companion computer, e.g., can connect to the flight controller over UART. The companion computer can run Mavproxy, and the Mavproxy configuration file can be set to point send Mavlink packets over UDP to the IP address of the CloudStation. You will need to make sure firewalls are appropriately configure. A detailed example of how to do this over 4G network is also at 4guav and in ref:

[J70] Peter J. Burke “A Safe, Open Source, 4G Connected Self-Flying Plane With 1 Hour Flight Time and All Up Weight (AUW) <300 g: Towards a New Class of Internet Enabled UAVs” IEEE Access, 7(1), 67833 – 67855 (2019).
2.3 AWS RDS (Aurora engine) - Experimental

Note that the project uses SQLite due to its low cost and ease of use with Django. However, AWS RDS can be configured for scalability and robustness.

1. Launch an RDS instance on AWS with Aurora with MySQL compatibility
   - db.r5.large is good enough to test the deployment
     - Configure Security Group (AWS EC2 Console)
       * MySQL/Aurora(TCP) Port:3306 Source:My RDS endpoint

2. Add the RDS_DB_NAME, RDS_USERNAME, RDS_PASSWORD, RDS_HOSTNAME and RDS_PORT to the environment variables
   - You can do this by editing the ~/.env file and adding the variables in the following format.
   
   ```
   variable_name=value
   ```

   - Add this to the ~/.bash_profile file
   
   ```
   set -a
   . ~/.env
   set +a
   ```

   - Make sure to run this command after editing the ~/.bash_profile file.
   
   ```
   source ~/.bash_profile
   ```

3. Edit the cloud_station_web/webgms/settings.py file and change the DATABASES field to the following

   ```python
   import os
   DATABASES = {
   'default': {
   'ENGINE': 'django.db.backends.mysql',
   'NAME': os.environ['RDS_DB_NAME'],
   'USER': os.environ['RDS_USERNAME'],
   'PASSWORD': os.environ['RDS_PASSWORD'],
   'HOST': os.environ['RDS_HOSTNAME'],
   'PORT': os.environ['RDS_PORT'],
   }
   }
   ```

4. Edit the cloud_station_deployment/backgroundtasks.service and add this line to the [Service] section.

   ```
   EnvironmentFile=/home/ubuntu/.env
   ```

5. Run `bash ~/cloud_station_deployment/configure_web_server.sh`

6. Run `bash ~/cloud_station_deployment/reload_server.sh`

2.4 Authors

- Lyuyang Hu
- Omkar Pathak
2.5 Troubleshooting

1. How do I know my hardware setup is correct (the vehicle is sending mavlink messages to the server)?
   
   ```
   sudo tcpdump -n udp port 14550 -X will print the messages received at port 14550 (UDP).
   ```

2. How do I know NGINX and Daphne is running? How do I know if there are errors?
   
   ```
   service nginx status
   service daphne status
   ```

3. How do I know the status of django_background_tasks?
   
   ```
   service backgroundtasks status
   ```

4. The telemetry textbox shows that the websocket connection between server and browser has been disconnected.
   What do I do?
   
   ```
   • This usually means Redis fails. Following the Django Channels recommendation, we use Redis as the backing store for the channel layer. We use Docker to run Redis.
   
   • To check status of Docker: service docker status
   
   • To show all Docker containers on the machine: sudo docker ps -a
   
   • To restart Docker and Redis:
   
   ```
   sudo systemctl start docker
   sudo systemctl enable docker
   sudo docker run -p 6379:6379 -d redis:2.8
   ```

5. If you are using a Python version past 3.6, Python does not like “pkg-resources” anymore. Go to the requirements.txt file and comment out the following line: “"pkg-resources==0.0.0"”
For a video guide on how to use CloudStation, go here.

1. In your web browser, go to your AWS instance’s DNS address. You will see a map that takes up the whole screen. In the menu at the top, click “Log in/Sign up” and select “Sign up.”

2. Sign up for an account, using a password that you are not using for any other accounts.

3. After logging in with your new account, you will see a box that says “Connect to Vehicle via ID.”
   - Make sure your drone or SITL instance is connected to one of your CloudStation instance’s open UDP ports.
   - Enter the number of the UDP port your drone is connected to (e.g., 14550). This will be that vehicle’s drone ID.
4. Controlling the drone:

- Please note that only rovers are currently supported by CloudStation. You will not be able to control any flying vehicle type. CloudStation has also only been tested on drones running ArduPilot firmware.
- You can arm/disarm the drone or change its flight mode by clicking on the drone’s icon on the map.
- To set a fly-to point, make sure the tab of the drone you want to control is selected on the right-hand menu. Left-click anywhere on the map to drop a pin, then right-click the pin to bring up the pop-up menu. From here, you can either “clear pin” or have the drone “fly-to” the pin.

5. Adjusting displayed telemetry data:

- On any connected drone, click the “Edit Data” button under the default telemetry table. This will bring up a form with several columns of checkboxes.
- Select the data fields you would like to monitor and click “Submit.” Data for the fields you selected will now be displayed in the “Other Data” section of the telemetry monitor. This change will apply for all drones.
How to Contribute

This contributor guide aims to list all the technologies used for this project to help student developers get started quickly. If you just want to quickly set up a development environment, please follow the instructions in WebApp-Getting Started.

4.1 Technologies

1. Hardware: The focus of the project isn’t on the hardware. However, CloudStation is built for the hardware so it is important to have a good understanding of how it works and where to find answers if you run into issues.
   1. Ardupilot (autopilot software)
      • Mission Planner
      – You will find it helpful to debug with this GCS that runs on local computer
   2. MAVLink: messaging protocol used for drone-to-drone and drone-to-GCS communication
      • documentation for all message types can be found here

2. Web development
   • Front end: html, javascript, bootstrap, MapBox
   • Back end: Django
      – SQLite is used in the code for simplicity. Django provides interfaces for various flavors of SQL databases. We also have instructions in Deployment -> AWS RDS to help you get started with AWS RDS

3. Deployment
   Please refer to the bash scripts in CloudStation Deployment. Some technologies we used are:
   • NGINX, Daphne
4.2 General Notes

1. The front end currently uses vanilla JS. It would be best to refactor it in a way that is more sustainable for future development. Using some kind of framework (Angular, React, Vue) should help.

2. Back End
   - The APIs we implemented don’t follow the popular RESTful style. We decided to make them structured in a way that’s very similar to MAVLink commands/messages. It makes initial development a lot easier but I imagine it will make it difficult to add some more advanced features. I think having RESTful APIs actually does make sense for this project. An example of the resources would be a JSON document of all the telemetry information of the vehicle. The backend should send the correct command according to the current speed in that document, etc. This design should make it easier to implement the front end and makes it more structured.

3. Security
   - This is probably the biggest problem of the current implementation. I don’t have any experience in this field so I can’t provide much information.
   - Currently, CloudStation cannot be used safely for multiple users. User accounts do not actually link drones to their respective accounts, and there are no safeguards in place to prevent one user from entering the drone ID of a different user. Editing the additional telemetry data displayed in “Other Data” will also affect settings for all users.

4.3 Feature Improvements

1. Enhancing multi-user support and security (see General Notes - Security)
   - Saving drone IDs for each user
   - Some sort of safeguard to prevent a user from controlling a drone that is not theirs
   - Saving telemetry data displayed in “Other Data” individually for each user

2. There is currently no support for takeoff, landing, etc., so vehicle types other than rovers cannot be controlled through CloudStation.

3. The fly-to feature currently sets all fly-to points to 0 altitude, which means it can only be used for rovers. If support for flying vehicles is added, the fly-to feature will need altitude controls.

4. Waypoint missions are currently partially implemented in the backend but have no front-end interface and have not been tested.

5. Controlling multiple drones at a time feels very unintuitive and clunky. Improvements could include:
   - Automatically selecting the tab of a drone when it is clicked on
   - Automatically bringing up the popup for a drone when its tab is selected
   - Automatically selecting the tab of a drone when a pin belonging to that drone is clicked
   - Color-coding or otherwise differentiating pins belonging to different drones
   - etc.

6. Adding an HUD (with an artificial horizon, etc.) for the currently selected drone

7. Adding video streaming support for the currently selected drone

8. Using different vehicle icons based on the type of vehicle detected
5.1 Authors

- Mina Bedwany - full stack
- Lyuyang Hu - project management, software architecture design, full stack, hardware
- Zeyu He - front end
- Hunkyu Lee - hardware
- Jace Mica - back end
- Omkar Pathak - infrastructure

5.2 Advisor

- Professor Peter Burke

5.3 Contributing Developers

- Chris Zhang
- Hyun Jay Yang
- Lindsey Garcia

5.4 Contact Information

Lyuyang Hu: lyuyangh at uci dot edu
Chapter 6

License

GNU General Public License v3.0