REPÜLÉSTUDOMÁNYI KÖZLEMÉNYEK
32. évfolyam (2020) 1. szám 175–190. • DOI: 10.32560/rk.2020.1.12

Vránics Dávid Ferenc[1]

Testing of a Cloud-Controlled Unmanned Aircraft System

During the last years, cloud controlled unmanned aircraft systems (UAS) have become reality. Keeping up with the trend, my research focused on design, development and testing of such a system, while keeping security concerns in mind. The present article summarises the final testing phase, including flight tests and test in lab environment. The most interesting results include successful flight tests with more than 15,000 km communication roundtrip between the ground controller station, the cloud server and the drone; evidences of the importance of georedundant installation of compute hosts to increase survivability of the service; and the demonstration of automatic horizontal scaling of the system depending on performance demands.

Keywords: *UAS, drone, cloud, testing*

Egy felhőből irányított pilóta nélküli légijármű-rendszer tesztelése

Az utóbbi évek során a felhőből irányított, pilóta nélküli légijármű-rendszerek világa valósággá vált. A jelenkor kihívásaival lépést tartva, kutatásom az ilyen rendszerek tervezésére, fejlesztésére és tesztelésére fókuszált, a biztonsági kérdéseket szem előtt tartva. Jelen publikáció összegzi a tesztelés végső fázisát, amely repülési teszteket és labor körülmények közti teszteket is magába foglalt. A legizgalmasabb eredmények között említeném a sikeres reptetést több mint 15000 km kommunikációs úttal a földi irányító állomás, a felhő szolgáltatás és a légi jármű között; a felhőt alkotó számítógépek georedundáns elhelyezése által nyújtott túlélési képesség szemléltetését; illetve a szolgáltatás horizontális skálázhatóságának szemléltetését a teljesítményigény függvényében.

Kulcsszavak: *UAS, drón, felhő, tesztelés*

1. Introduction

During the last decade, multiple implementations of cloud based UAS were demonstrated. From Internet of Drones (IoD) through Dronemap Planner and DroneDeploy to DJI FlightHub various

[1] MSc, University of Public Service, Doctoral School of Military Engineering, PhD student, vranicsd@gmail.com, ORCID: https://orcid.org/0000-0003-0637-476X

solutions have been made available to the public by publications, industrial recommendations or commercial release of services.[2] My research focuses on the security of such systems during design, development and testing phases of the product lifecycle. I developed a prototype cloud controlled UAS for the purpose of demonstrating the potential of the concept, including robustness and scalability of the flight support services.

1.1. The system under test

Analogous to the Internet of Things (IoT) systems, a cloud-based Unmanned Aircraft System (UAS) consists of the aerial subsystem and the ground-based subsystem.[3] In the ground-based subsystem, the cloud computing core infrastructure hosts the command and control services, while the end-user devices facilitate user interaction (e.g. mission planning with map integration). The aerial subsystem (that is, the Unmanned Aerial Vehicles [UAV]) interacts with the core system via a separate interface, as can be seen on Figure 1.

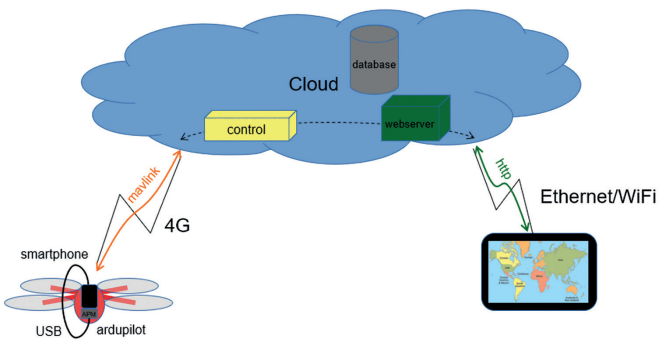

Figure 1.
Schematics of the system layout. Source: compiled by the author.

The following test cases focus on the cloud system's interface towards the UAVs. The reason behind this is that the user interface is browser-based and is communicating on a Representational State Transfer (REST) Application Programming Interface (API) with the cloud. As such, the methodology is already well-established[4] and studied in depth,[5] no novel findings surfaced during testing – except for one case, when the coordinate system of the

[2] Mirmojtaba Gharibi, Raouf Boutaba and Steven L. Waslander, 'Internet of drones.' IEEE 4 (2016), 1148–1162; Anis Koubâa, Basit Qureshi, Mohamed-Foued Sriti, Azza Allouch, Yasir Javed, Maram Alajlan, Omar Cheikhrouhou, Mohamed Khalgui and Eduardo Tovar, 'Dronemap Planner: A Service-Oriented Cloud-Based Management System for the Internet-of-Drones,' *Ad Hoc Networks* 86 (2009), 46–62; 'Drone Operations Management Solution,' DJI (Da-Jiang Innovations) FlightHub. Available: www.dji.com/hu/flighthub (29. 04. 2020.)

[3] Dávid Ferenc Vránics, Mátyás Palik and Zsolt Bottyán, 'Electronic administration of unmanned aviation with Public Key Infrastructure (PKI),' *Security & Future* 3, no 4 (2019), 152–155.

[4] Sergio Segura, José A. Parejo, Javier Troya and Antonio Ruiz-Cortés, 'Metamorphic Testing of RESTful Web APIs,' *IEEE Transactions on Software Engineering* 44, no 11 (2018), 1083–1099.

[5] Andy Neumann, Nuno Laranjeiro and Jorge Bernardino, 'An Analysis of Public REST Web Service APIs,' IEEE Transactions on Services Computing, 2018, p. 1.

Mapbox map display framework happened to be swapping the order of longitude and latitude compared to the coordinates of the MAVLink protocol (detailed later in this article).

1.2. Test infrastructure setup

According to the definition of the International Software Testing Qualification Board (ISTQB),[6] a test infrastructure is the collection of 'the organizational artifacts needed to perform testing, consisting of test environments, test tools, office environment and procedures.' In our case, two different approaches were applied: field tests were conducted to verify the functionality of the system, and lab tests to verify other non-functional requirements (such as robustness). The two approaches were covered with separate infrastructure.

1.2.1. Field tests

During field tests, a DJI F450 quadcopter platform provided the base for the aerial subsystem (see Figure 2). The UAV's ArduPilot Mega autopilot was connected to a smart phone via an Universal Serial Bus (USB) On The Go (OTG) cable, the smartphone being the host side. The smartphone ran an Android app called Micro Air Vehicle (MAV) Downlink. The app connected to the core system via 4G internet. For manual flight a traditional 2.4 GHz radio remote controller was used, this also served as a means of backup control in case of autonomous flight.

The autopilot was calibrated and set up via ArduPlot Mega (APM) Planner software, all sensors were health-checked with the built-in tools. Geofencing was also enabled and set up on the drone to make sure it does not leave the designated airspace.

Figure 2.
Closeup of the UAV used in the testing process, powered up. Source: Photographed and edited by the author.

[6] ISTQB Glossary, International Software Testing Qualification Board. Available: https://glossary.istqb.org/ (29. 04. 2020.)

The core system was run on a cloud-based virtual machine, physically located in Chicago, United States, hosted by Chicago Virtual Private Server (VPS). At this point, the cloud system hosted the UAV interface, the database and the mission planner interface in a single Virtual Machine (VM) with no scaling capabilities.

The user interface (map) was displayed on either a smartphone or a laptop in the field.

Even with the 2×7500-kilometer distance, the network latency between the drone and the ground control station (laptop or smartphone) was around 150 milliseconds (measured with Internet Control Message Protocol [ICMP] echo requests, that is, ping). This implementation of the core cloud system synchronises the two interfaces every second, so the network delay is negligible compared to the synchronisation time.

With the above setup, two test flights were conducted.

1.2.2. Lab tests

Lab tests were focusing on the characteristics of the core infrastructure, and mainly the UAV-facing interface, thus the map display interface was omitted during these tests, data was checked directly in the central database. The functionality of the map was already tested during the flight tests, apart from that, the interface is a traditional REST design and as such, there is no novelty or finding to be published, as noted before.

The core system was running on two compute machines, hosted on traditional laptops. A third laptop was running a proprietary MAVLink 1.0 drone simulator, implemented specifically for this testing setup by the author using C++ and Test and Test Control Notation version 3 (TTCN-3) language. This program simulated 100 parallel UAVs, which emitted position and heartbeat data with a 1 second interval. The basis for the data was captured directly from the drone in the field tests, then actualised by the code where necessary (drone identifiers, timestamp, checksum, and so on).

The base networking was served by a traditional home router, see Figure 3 below.

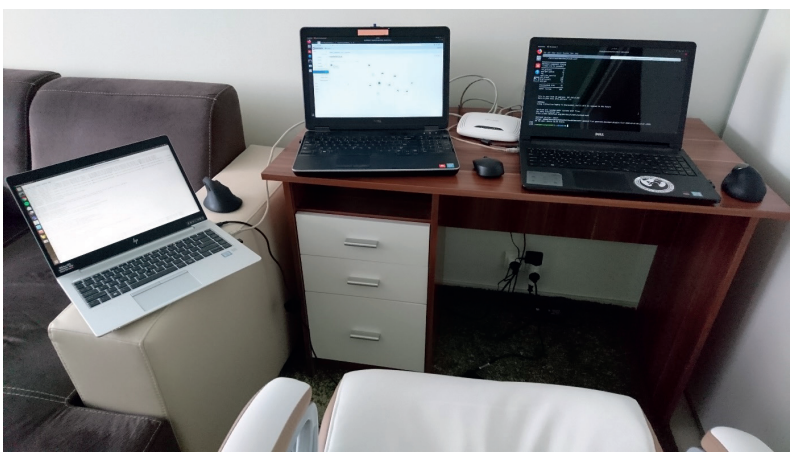

Figure 3. *The physical layout of the machines and the router: the simulator host, host 'uascontrolnode', the router, and host 'uascomputenode1' (left to right).* Photographed by the author.

The two compute machines served altogether 8 processor threads and around 23 GB memory, see Figure 4.

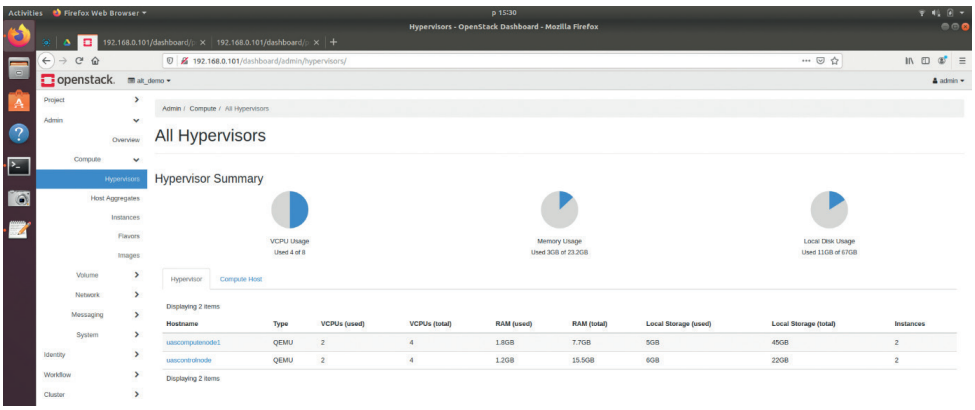

Figure 4.
Hypervisors displayed on the OpenStack horizon dashboard. Screenshot by the author.

The machines were utilised in a way through OpenStack so that automatic scaling and placement of the service could be achieved. The orchestration was conducted by the OpenStack Heat component. See Figure 5.

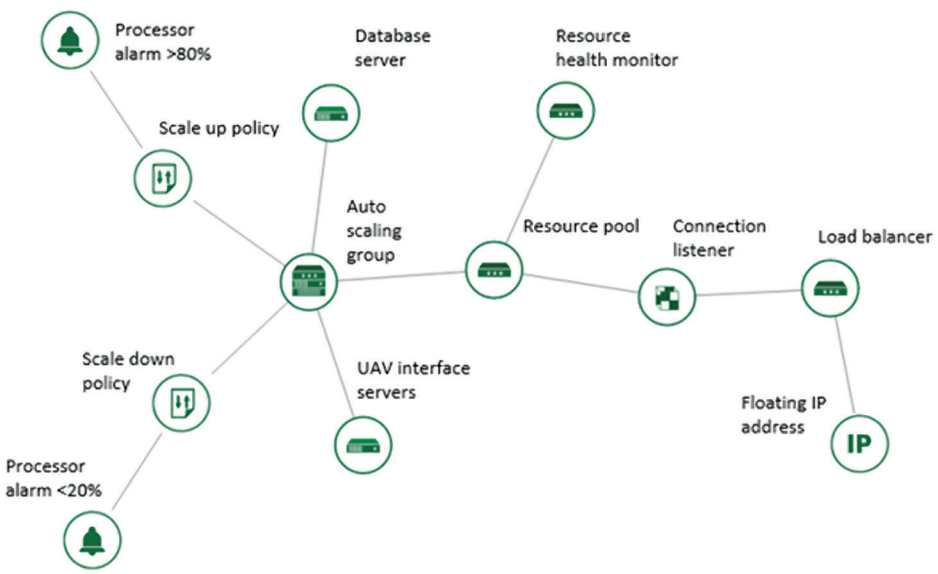

Figure 5.
Schematics of the autoscaling stack configuration. Source: compiled by the author.

1.3. Test objectives

1.3.1. Horizontal scaling

The term 'horizontal scaling' refers to creating more (or less) parallel virtual machines when needed – for example based on processor load or network throughput; opposed to growing or shrinking a single virtual machine when the resource needs dictate so. The latter is called vertical scaling.

1.3.2. Anti-affinity

In some cases, a service would require placing related virtual machines as close physically as possible (i.e. possibly on the same machine), for example to reduce network latency. This property is called affinity. In our case, the service is preferred to be spread on as many machines as possible (thus anti-affinity), to prevent a single point of hardware failure causing total service outage.

1.3.3. Robustness

A non-functional property of the system is robustness, or with other words, in our case the main goal is survivability. Anti-affinity and geo-redundant physical placement of servers are key to achieve a robust service.

1.3.4. Session persistence

For ease of connection handling, single UAVs are preferred to be routed to the same virtual machine by the load balancer every time they connect. This way, some control communication (that is, session buildup and teardown) can be spared for individual messages, the network cost of which is comparable to the actual UAV data in the case of MAVLink 1.0 protocol.

1.3.5. Flight control functionality

During flight tests, different functions of the map user interface were to be tested, including:

- UAV position display;
- Arming/disarming the rotors;
- Autonomous takeoff/landing;
- Flight path/single waypoint upload onto the UAV, deleting waypoints;
- Autonomous/manual flight.

1.4. Test strategy

The test strategy includes methodical testing (pre-planned testcases based on a checklist that evolved during the development) and dynamic testing (exploratory testing, adapting to the challenges evolving on-the-field).

2. Conducted field tests

In the preamble phase of testing, the drone was powered on by connecting the battery, while the smartphone was connected to the autopilot via USB cable and was fixed on-board on the UAV. Then the MAV Downlink app was started to relay data between the USB and the internet. Postamble phase involved reverting the previous steps.

2.1. Manual flight

The first flight was conducted on 1st June 2019. The objective was to verify overall operation of the system (that is, system test). Short manual flights were flown, while collecting position and telemetry data on the server side for later analysis, as can be seen on Figure 6.

Figure 6.
The UAV during manual flight. Photograph in the property of the author.

The flights took place in the following airspace:

A1557/19
81
470603N 0201207E
470746N 0201204E
470752N 0201748E
470611N 0201753E

470603N 0201207E
(Szolnok)
GND 2500 feet AMSL
07:00 16:00
UAV flight

The author's Drónpilóták Országos Egyesülete (DOE) membership number is 269189, while his Groupama insurance number is 930/871727715 (valid in the period of both flights).

2.1.1. Test case 1.1: Manual flight

The UAV was controlled with a 2.4GHz remote controller in STABILIZE mode. After manual arming, take off and a short hover, the drone was manually landed and disarmed. During the flight, the server side was observed, position data and flight telemetry was logged with no visible interruption. For the map display, see Figure 7.

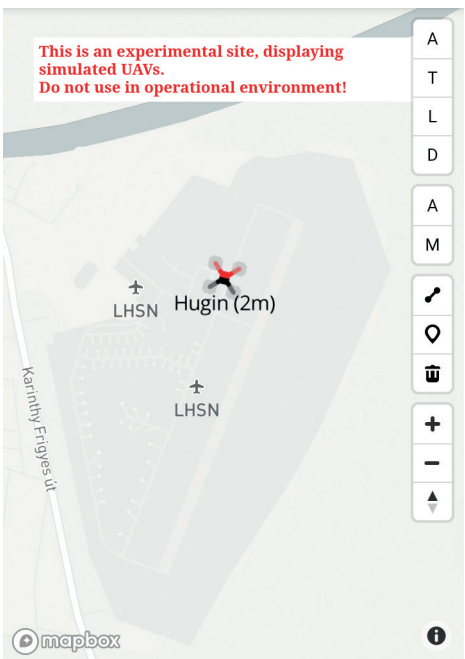

Figure 7.
The map display during manual flight. (Note: the disclaimer in red is intended for accidental visitors of the website, as access is not authenticated.) Smartphone screenshot by the author.

2.1.2. Test case 1.2: Connection loss and recovery

During a manual landing scenario, the drone tipped to side because of a wind gust and disconnected the USB cable from the smart phone. Because of this interruption in communication, no data was sent to the server side. Upon reconnecting the USB cable, the connection was restored. On the server side no outage was observed, data collection resumed as the smart phone forwarded more data.

2.2. Autonomous flight

The second flight was conducted on 17th November 2019. The main objective was to verify autonomous flight functionality and path planning. The flights took place in the following airspace:

A4036/19
84
470603N 0201207E
470746N 0201245E
470752N 0201748E
470611N 0201753E
(Szolnok)
GND 4500 feet AMSL
07:00 15:00
UAV flight

2.2.1. Test case 2.1: Remote arm/disarm

During this test case the UAV was stationary on ground. On the map interface, the first A (arm) button was clicked, the rotors activated at a healthy idle speed, then the D (disarm) button was clicked to observe the rotors stopping. Note: it was verified by checking the server logs that the rotors stopped because of the user interaction, as rotors get disarmed automatically after a few seconds of idle spin with no thrust input. The remote controller was not connected at this point.

2.2.2. Test case 2.2: Remote takeoff/land

After arming the rotors, the same way as above, the second A (auto mode) button was clicked on the user interface. Then the first A (arm) button was clicked to arm the rotors, then the T (takeoff) button was clicked. The UAV failed to take off, the rotors disarmed after a few seconds. Later investigation showed that as per documentation,[7] ArduCopter version 3.2.1 that runs on the UAV's APM 2.6 board requires manual (remote control) thrust input to start

[7] 'Copter Home.' Ardupilot. Available: https://ardupilot.org/copter/index.html (29. 04. 2020.)

mission as opposed to the newer versions, which support mission start immediately upon entering AUTO mode.[8] Note: The current implementation of the system uploads a single mission item to the UAV, which contains a takeoff type-waypoint.

2.2.3. Test case 2.3: Mission upload/delete

The mission planning and upload was verified beforehand by APM Planner and QGroundControl applications, by uploading a mission (both single waypoint and flight path missions) onto the drone via the proprietary web user interface, then reading the mission from the autopilot with the above listed, commercially available applications. This dry test had shown that the MAVLink and MapBox representations of coordinates are different in the order of latitude and longitude, causing the mission waypoints to shift from Hungary (for example 47° N, 19° E) to Saudi Arabia (19° N, 47° E). This error was fortunately revealed and fixed before actual field tests were conducted.

On the field, the mission upload and delete feature was retested by uploading a single waypoint to the drone, by placing the waypoint on the map display, by clicking the map marker button on the interface, then clicking on the map, while observing the server logs for responses from the drone. Deleting the waypoint was tested by clicking the trashcan icon on the user interface, observing the response from the autopilot in the server console and logs.

2.2.4. Test case 2.4: Mission override

A mission was uploaded following the above procedure, then a new mission was planned on the user interface. The server logs were observed to verify the response from the drone, acknowledging the new mission and deleting the old one.

2.2.5. Test case 2.5: Autonomous flight

The drone was set to manual mode (button M on the user interface) and the remote controller was switched on with throttle on zero. A single waypoint was uploaded and verified by observing the server console output. The rotors were armed by clicking the first A (arm) button, then a manual takeoff was performed by increasing the throttle. At around 5 meters altitude above ground, the UAV was kept hovering while the second A (auto mode) button was clicked on the user interface. The autopilot took over the control, and the drone proceeded towards the assigned waypoint. When the waypoint was reached, the autopilot indicated it with a MAVLink message, stopped and kept its position hovering in place, see Figure 8. Then the M (manual mode) button was clicked to take back manual control, and the drone was piloted back to the point of takeoff. During the manual landing procedure, unfortunately the UAV dropped, and the connection was lost. The most plausible cause of the crash may be that

[8] 'Archived: APM 2.5 and 2.6 Overview.' Ardupilot. Available: https://ardupilot.org/copter/docs/common-apm25-and-26-overview.html (29. 04. 2020.)

because of a sudden gust and the constant vibration, the battery cable disconnected, and this resulted in the drone crashing. Upon examination, the USB cable was also discovered to be disconnected, but the flight in that stage was in manual mode and the logged data on the server suggested that the cable disconnected as a result of the crash. The frame and all four propellers were found broken (see Figure 9), so field testing had to be concluded.

Figure 8.
The server console log and map display during autonomous flight. (Note: the orange dot on the map is the designated waypoint.) Screenshot by the author.

Figure 9.
The author with the field equipment after the tests. Photograph in the property of the author.

3. Conducted lab tests

3.1. Scaling

3.1.1. Test case 3.1: Manual scaling

Upon creation of an OpenStack Heat ScalingPolicy, an internal Uniform Resource Locator (URL) is generated. This URL can be used to trigger the policy manually to perform the configured adjustment of resources, for example to scale up or down an AutoScalingGroup. During this test, the above-mentioned manual method was tested to trigger a scale up. Figure 10 shows the successful result of said action, namely a new virtual machine was created, and it joined the service. The detailed changes in networking can also be observed in Figure 11.

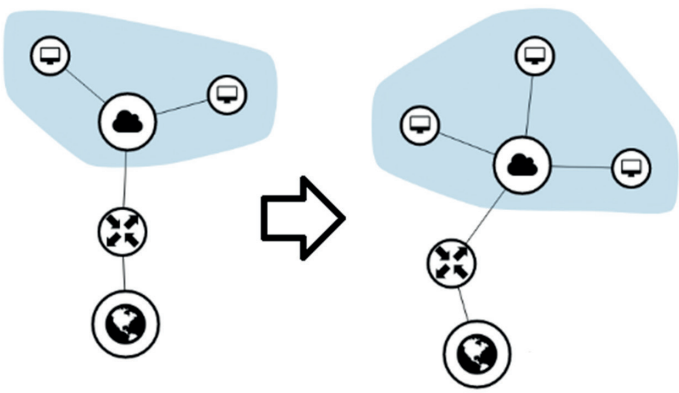

Figure 10.
A new server was created. Graphic edited from actual screenshots by the author.

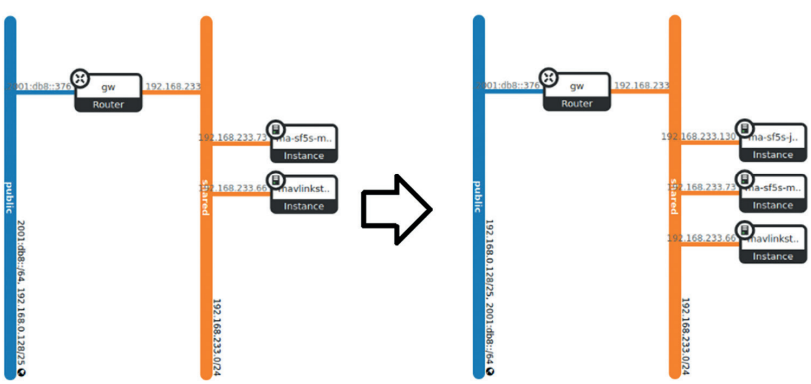

Figure 11.
The new server's addition to the network topology. Graphic edited from actual screenshots by the author.

3.1.2. Test case 3.2: Automatic scaling

Of course, it is preferred to provide scaling capability automatically, so a resource alarm was set up to both scale up and scale down policies. The resource alarm polls the value of a Central Processing Unit (CPU) metric every five minutes and triggers a policy if the desired threshold is reached. Initially, CPU load was generated by simulating 100 UAVs, each sending heartbeat and position data every second on a separate laptop in the network. As this proved to occupy only less than 10% of the CPU capacity in the single initial virtual machine, additional load was generated by executing an infinite loop on the machine manually. After a few minutes, at the next polling event, the scale up policy was successfully triggered, and an additional virtual machine got created. Stopping the load caused the system to scale down at the end of the next polling period, and the newly created virtual machine got stopped.

3.2. Anti-affinity

3.2.1. Test case 3.3: Static placement

In this case, the stack was configured to start two virtual machines immediately upon creation. The goal was to verify if the two virtual machines are started on different compute hosts. The test passed, the first machine and the database got created on the first node ('uascontrolnode'), the other machine started on the other node ('uascomputenode1').

3.2.2. Test case 3.4: Dynamic placement

This test involved starting a single virtual server, then triggering a scale up manually as in test case 3.1. After the manual scale up, the second virtual machine was placed correctly on the other physical host.

3.3. Robustness

3.3.1. Test case 3.5: Service survivability

In this case, the service is originally spread onto two compute hosts. While simulated UAVs were generating active load on the virtual machine on the second physical host, the network cable between this host and the network router was disconnected, thus simulating the loss (that is, destruction) of the machine. The health monitor component noticed this by doing periodical Transmission Control Protocol (TCP) based checks on the service, and the virtual machine's status was flagged 'Error' in the list of servers (see Figure 12). The load balancer then re-routed the incoming connections towards the other, available server. This way the service survived and suffered no noticeable interrupt.

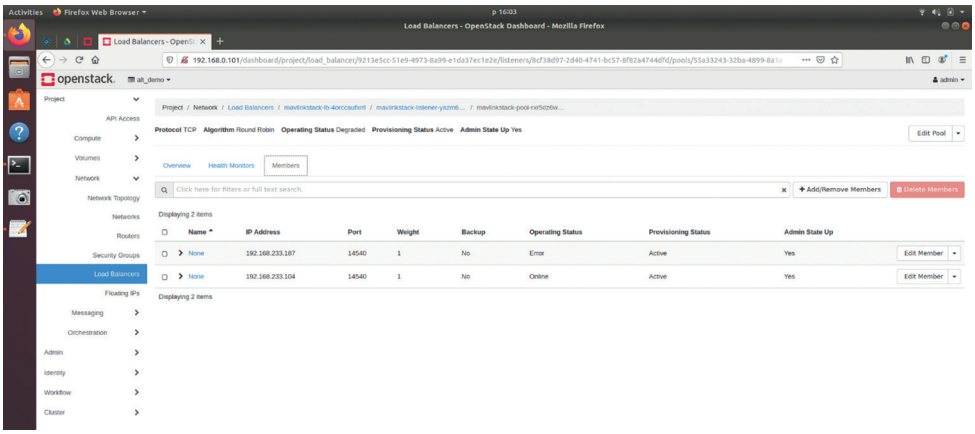

Figure 12.
The operating status of the disconnected VM turns to 'Error'. Screenshot by the author.

3.3.2. Test case 3.6: Service restoration

Test case 3.5 was extended with the case of finally reconnecting the lost compute host. After plugging back the network cable, the host networking returned, and the health monitor noticed the return of the virtual machine. The service code was designed in a way that the interrupted network connections were relieved immediately, so no connections get 'stuck' waiting for reconnection because of the rerouted clients (that were already reconnected to the other server). New clients after the reconnection were successfully routed towards the returning server.

3.4. Session persistence

3.4.1. Test case 3.7: Client reconnection

This time a client connected to the service, disconnected, then reconnected. The aim of this test was to verify resource cleanup on the server side upon client disconnection, and proper creation of new resources. This property was essential for other test cases like test case 3.6 and 3.8.

3.4.2. Test case 3.8: Persistence on current server

Initially two virtual machines were started. A client connected to the service and was routed to the first server. Upon disconnection and reconnection, it was indeed routed to the first server again, based on its source Internet Protocol (IP) address. This property would be useful if the server side waited for clients to reconnect by design. In that case the resources would not need to be completely freed when a client disconnects momentarily. In our case the only benefit was shown during the following test case.

3.4.3. Test case 3.9: Persistence on new server after service migration

Again, initially two virtual machines were started on two separate compute hosts. The connecting client was routed to the second host. The networking of the host was then interrupted (network cable again unplugged), and the client was observed to be rerouted towards the first host. The network cable was reinserted, but the client remained connected to the first host. This way the client did not need to reconnect towards the second (original) host, sparing teardown and buildup of connections which would have no added value to the client.

4. Conclusion

In this article, field and lab testing aspects of a cloud controlled UAS were reviewed. Functional, flight control capabilities of the system were verified in field environment, proving that such a concept is viable to conduct autonomous UAV missions literally from the other side of the world (in these cases the communication roundtrip was around 15,000 kilometers between Chicago and Szolnok).

Core functionality and non-functional properties of the system were verified in a multi-computer setup, successfully demonstrating scalability, robustness, session persistence and anti-affinity capabilities, while handling several parallel simulated UAV clients.

Bibliography

'Archived: APM 2.5 and 2.6 Overview.' Ardupilot. Available: https://ardupilot.org/copter/docs/common-apm25-and-26-overview.html (29. 04. 2020.)

'Copter Home.' Ardupilot. Available: https://ardupilot.org/copter/index.html (29. 04. 2020.)

'Drone Operations Management Solution.' DJI (Da-Jiang Innovations) FlightHub. Available: www.dji.com/hu/flighthub (29. 04. 2020.)

Gharibi, Mirmojtaba – Boutaba, Raouf – Waslander, Steven L.: 'Internet of drones.' IEEE 4 (2016), 1148–1162. DOI: https://doi.org/10.1109/access.2016.2537208

ISTQB Glossary. International Software Testing Qualification Board. Available: https://glossary.istqb.org/ (29. 04. 2020.)

Koubâa, Anis – Qureshi, Basit – Sriti, Mohamed-Foued – Allouch, Azza – Javed, Yasir – Alajlan, Maram – Cheikhrouhou, Omar – Khalgui, Mohamed – Tovar, Eduardo: 'Dronemap Planner: A Service-Oriented Cloud-Based Management System for the Internet-of-Drones.' *Ad Hoc Networks* 86 (2009), 46–62. DOI: https://doi.org/10.1016/j.adhoc.2018.09.013

Neumann, Andy – Laranjeiro, Nuno – Bernardino, Jorge: 'An Analysis of Public REST Web Service APIs.' IEEE Transactions on Services Computing, 2018. DOI: https://doi.org/10.1109/TSC.2018.2847344

Segura, Sergio – Parejo, José A. – Troya, Javier – Ruiz-Cortés, Antonio: 'Metamorphic Testing of RESTful Web APIs.' *IEEE Transactions on Software Engineering* 44, no 11 (2018), 1083–1099. DOI: https://doi.org/10.1109/tse.2017.2764464

Vránics, Dávid Ferenc – Palik, Mátyás – Bottyán, Zsolt: 'Electronic administration of unmanned aviation with Public Key Infrastructure (PKI).' *Security & Future* 3, no 4 (2019), 152–155.