UAVs-as-a-Service: Cloud-based Remote Application Management for Drones

Jerico Moeyersons, Martijn Gevaert, Karl-Erik Réculé, Bruno Volckaert and Filip De Turck
IDLab, Department of Information Technology
Ghent University - imec, Ghent, Belgium
Email: jerico.moeyersons@ugent.be

***Abstract*—In recent years, the Internet of Drones (IoD) became an important research topic for both industry and academy. An IoD environment consist of different drones, called Unmanned Aerial Vehicles (UAVs), flying in different zones whereby communication is important. Therefore, drones are becoming increasingly ambiguous, capable and more cost effective than ever before. These drones have been equipped with different sensors, making it IoT-enabled drones, capable of capturing multiple data sources and send them to the cloud for further research, but the continuous advance in drone technology has not necessarily made drone application development easier. While mature Infrastructure-as-a-Service (IaaS) platforms offer features such as hardware abstraction, resource allocation and tools to manage applications remotely, commercial drones often offer a restrictive software environment instead. Inspired by the technical success and convenience of IaaS platforms, this article sets out to bring that experience to drones, resulting in the creation of the UG-One UAVs-as-a-Service (UAVaaS) platform. Many of the technologies used in the UAVaaS platform can be found in the world of Cloud computing as well. Applications created for drones are containerized using Docker and application management can be done through a web interface. The drones host a REST API for platform management and they have a Linux onboard computer. Developers can deploy applications on the drones or forward the required data and deploy their applications on a remote server instead. This approach has delivered promising results when evaluated using several reference applications that either represent real world applications such as video streaming and movement control or instead just stress tests to check for resource availability and reliability. In the end, the UG-One platform, is shown to succeed in simplifying drone application development and management while maintaining the reliability and versatility required from any drone platform.**

***Index Terms*—Application management, Docker, Drones, Infrastructure-as-a-Service, UAVs-as-a-Service**

I. Introduction

The last decade, Internet of Things (IoT), where various objects (or things) are connected through the Internet, has made a technological advancement [1]. Drones are one of these things [2], consisting of several IoT smart devices, such as LIDAR, thermal sensors, cameras, chemical sensors and many more [3]. They have become increasingly ambiguous and drone manufacturers have been able to create cost effective devices that can fly longer, carry heavier loads and have better on-board hardware. Even drone management systems have been developed to maintain a good overview of several deployed drones. [4]. The continuous advance in drone technology certainly has improved the capabilities of drones, but it has not necessarily made drone application development easier.

When building applications that need to run in the cloud, developers create containerized applications which are remotely deployed and managed, have guarantees when it comes to resource availability and developers certainly do not need to worry about the vendor of the hardware on which their applications are deployed. This is not the case when developing an application for a drone, where a developer needs to take into account different aspects such as specific hardware constraints, power constraints and load constraints. Therefore, this paper proposes a platform, further called the UG-One platform, unlocking the potential of these drone innovations towards a larger number of less drone-savvy developers.

By creating the UG-One UAVaaS platform based on cloud technologies, developers can now work in a familiar environment were applications are containerized, platform components communicate using REST APIs, drones run Linux and application deployment happens through a web interface.

However drones are not cloud servers, they need to make real-time decisions and run applications that often need access to on-board hardware such as cameras and actuators, or even require the ability to plot a new course for the drone. Instead of trying to create an all-in-one system that provides all the required functionality, the UG-One drone design is based on separate non-proprietary components with each component optimized for specific tasks. On a hardware level, the autopilot controls the drone while a Linux-based on-board computer runs applications. On a software level, each part of the UG-One drone system is containerized and can work independently without influencing other applications or components.

While a UG-One drone can be used on its own, deploying applications through a REST API or Swagger-UI is far from user-friendly and application management quickly becomes complex when dealing with multiple drones. For this purpose, the UG-One back-end has been created. This cloud system allows users to manage the applications running on a fleet of drones and simplifies things such as deploying applications that remotely control the drones, or even individually control drones through ground control software.

The remainder of this paper is organised as follows: Section II covers how drones can fly reliably and the need for an on-board generic computing platform while Section III

978-3-903176-32-4

explains the use of Docker on this on-board computer. In Section IV, the proposed system is illustrated en explained in detail followed by some reference implementations and validation in Section V. Section VI explains how a drone fleet can be managed and conclusions are summarized in Section VII

II. Reliable drone flight and on-board computer

As with most aviation systems, drones need to be reliable and thoroughly tested, which can slow down development. Each time software or hardware that controls the drone is modified, all aspects needs to be evaluated again. To prevent this kind of slowdown, open-source autopilots such as Pixhawk [5] and ArduPilot [6] can be used. These autopilots are good at following precise instructions such as follow a specific path, fly to these coordinates and fly at this speed. These instructions are communicated through the open-source MAVLink protocol [7], which is utilized by the two most popular and advanced open-source autopilot software projects, PX4 [8] and ArduPilot. Since most developers do not have any experience with MAVLink, the open-source MAVSDK library has been created by the team behind Pixhawk [9]. With these thoroughly tested, maintained and documented autopilots, developers do not need the time or skills required to create an autonomous drone themselves, instead they can focus on their applications which are deployed on a separate Linux-based on-board computer. If the on-board computer fails, the autopilot simply pilots the drone back home. The on-board computer hardware and software can as such be optimized for application deployment, performance and efficiency instead of redundancy and reliability at all cost. This allows the usage of mainstream low-energy consumption embedded computers such as a Raspberry Pi. Since most of the drone specific challenges are handled by the autopilot, the on-board computer can now be interpreted as a remote deployment server. The remote connection between the drone and the ground has been implemented using a simple network connection over Wi-Fi during development, but this can be done over cellular as well.

While MAVSDK can be used to "Talk MAVLink", a communication pipeline between the autopilot and the applications that use MAVSDK is still needed. Between the autopilot and the onboard computer, this happens over Universal Asynchronous Receiver-Transmitter (UART). On the onboard computer itself, a program called MAVLink-Router [10] then forwards these messages over UDP and routes them to applications on the drone or anywhere else, such as a cloud back end or possibly even another drone. Since multiple autopilots utilize the MAVLink protocol, it should now also be possible to simply switch out one autopilot with another, or in other words, deploy the same application on multiple drones with different autopilots.

III. Dockerised Drone

Because the on-board computer now acts as a remote deployment server, the software running on it can be developed in a way that takes full advantage of container technologies such as Docker [11]. In fact, each platform component or application that runs on the drones is containerized. By making use of container virtualisation technology, the system becomes inherently more reliable, it has improved hardware compatibility and it is easier to deploy (e.g. new hardware requires only that docker is supported).

It is also much easier to manage the resources available to containerized applications. This is critical as without resource management, any application can utilize as much of the available resources as it wants to use, resulting in applications slowing down or even crashing.

A. Resource management

Resource management is done by a containerized component running on each UG-One drone called the UG-One API. This application, among other responsibilities, exposes a REST API which can be used to easily integrate the drones into a cloud back end. The API is designed to provide all of the Docker functionality that is needed to manage applications on a drone, but with the required limitations to guarantee resources. It also exposes endpoints that provide information on the system such as connected USB devices, available disk space and current memory usage.

While the UG-One API can guarantee a fixed amount of memory available to applications, using underlying Docker functionality, it does not guarantee a fixed percentage of CPU time. Instead, it works by allowing developers to assign a CPU priority weight. As long as the CPU is not being used 100%, any application can utilize as much of the CPU as it wants. However, as soon as applications start competing for processing power, the distribution of CPU time will be made based on the weight of each application that is making calculations at that time.

B. Unmanaged resources

While CPU and memory resources are handled by the system, the system still lacks network bandwidth prioritization and disk space limiting. Network bandwidth prioritization is not yet part of Docker and while it is possible to limit the size of a Docker container, it is not yet possible to limit the volume on the host that users can mount.

Despite these shortcomings, the UG-One API is still capable of reliably managing containers and guaranteeing two critical resources. Because of this, it is used to manage all the components on a UG-One drone, including itself.

IV. System overview and implementation details

This section covers the system overview, illustrated in Figure 1. The upper part covers the autopilot and the on-board computer on the drone, which was covered in Section II and Section III. The other part visualizes the UG-One back-end and is explained below.

A. Back-end API

To ensure that the front-end can obtain dynamic content from the database, the API servers and from the drones, there

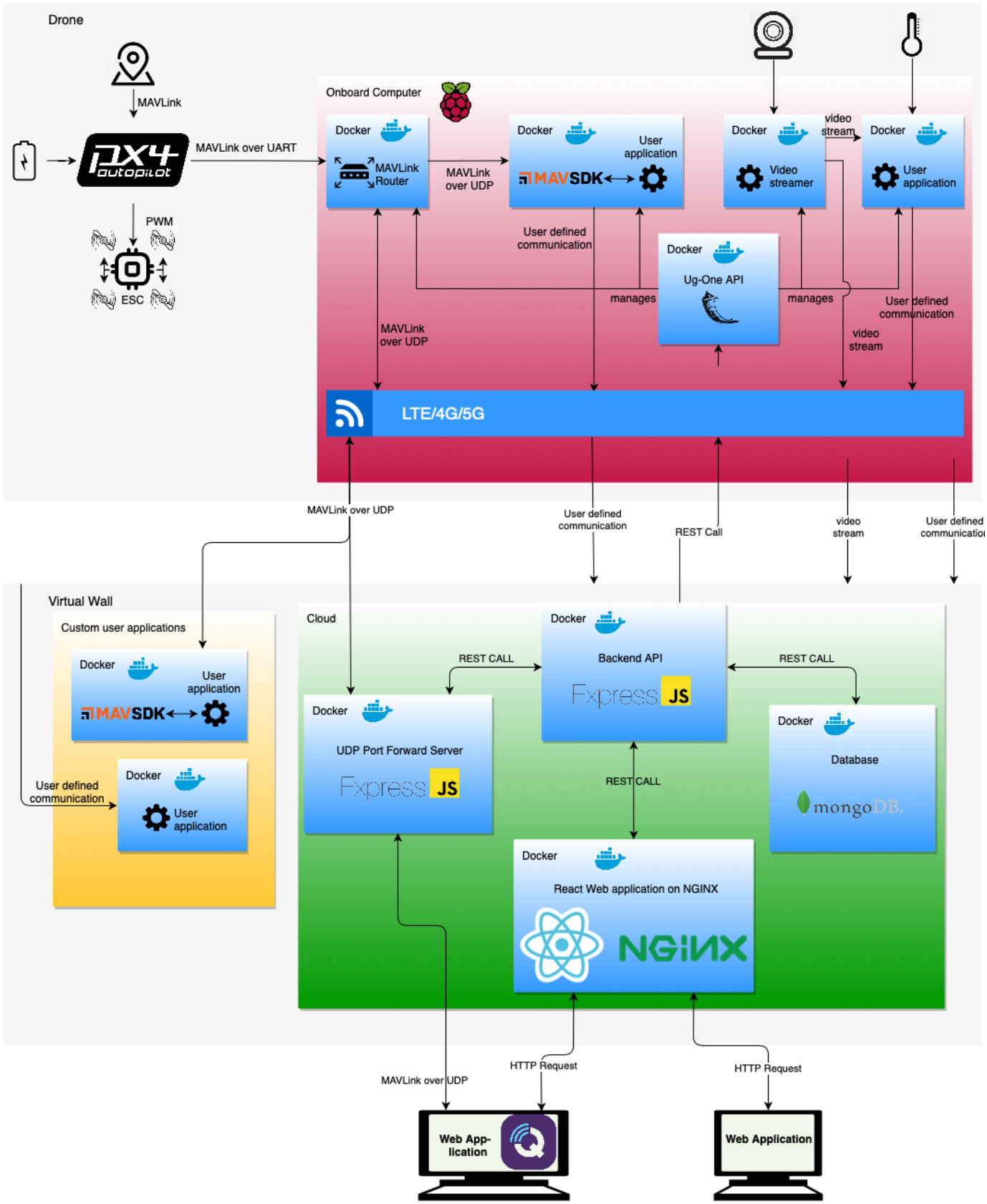

Fig. 1. UG-One system components overview

is a need for a central unit that will retrieve this information. This unit also needs to ensure that the drones can extract data from the database and make it possible that drones can retrieve their configuration. This central unit is the Back-end API which is created by a Node.js [12] server and the npm module Express [13].

To make a connection with the database, drones and API servers, several node modules are used. The first node module is Axios [14] which is a promise-based HTTP client npm module for the browser and Node.js. Through Axios, it is possible to retrieve information from the drones, manage the applications and request resources by using their UG-One API. Via the node module docker-hub-api [15], information of Docker Hub images from a public registry can be retrieved. To retrieve information from a Docker Image on GitLab, an attempt was first made to access it via a node module. These node modules were node-gitlab [16] and gitlab [17]. Due to lack of documentation and the absence of features which can retrieve Docker images, the GitLab API is chosen. With the GitLab API server [18], the different Docker images as their meta-information can be retrieved. The Mongoose node module [19] is used to access the MongoDB [20] database in the cloud to manage the data and modify the database structure.

For the development of a cloud platform, a data store is not to be missed out. For the cloud infrastructure, only one database was chosen for simplicity, but multiple types of databases can be used. Before a database can be deployed, research was done to determine if a SQL or a NoSQL data store is best suited for the cloud platform [21]. By weighing up the advantages and disadvantages, a choice was made to choose for a NoSQL database. NoSQL databases are easier to scale horizontally than SQL databases [22], because a NoSQL database ensures that if the data that needs to be stored grows, the database can easily be expanded. Also, there is no need to create a pre-defined schema and NoSQL databases are more suited to handle big data then SQL databases [21]. This is important because different applications on the drones, can store many different types of data in different structures. By comparing the different NoSQL databases and taking the CAP theorem [23] into account, the MongoDB database has been chosen. The CAP theorem states that it is difficult for a distributed datastore to simultaneously provide more than two out of the following elements: consistency, availability and partition tolerance. For the cloud infrastructure of the UG-One platform, consistency and partition tolerance are the two most important elements. Therefore, MongoDB was chosen because it offers consistency (all nodes see the same data at the same time) and partition tolerance (the system must work continuously without the loss of messages or partition failure).

B. UDP Port Forward Server

When the drone is configured and starts up, it is able to send MAVLink messages through the UDP Protocol to an application on a client's device such as a ground control station. A ground control station is a typical software application that runs on a computer on the ground. Through wireless telemetry, the computer on the ground communicates with the UAV. A ground control station can display real-time data about the performance, position, altitude, etc. of the UAVs. It can also be used to control an in-flight UAV, upload new mission commands and set parameters like altitude [24]. When the drone is ready to fly, one possibility is to send MAVLink messages directly to the client's device. However this has a couple of disadvantages. When the drone boots up, an IP address must be given to the drone to send MAVLink messages to. This IP address cannot be changed when the drone is active so when the IP address of the user changes, the connection with the drone will be lost. Also, when another user wants to take over the drone, this architecture prevents the user from controlling the drone. Another drawback is when multiple drones are sending their MAVLink messages to the same endpoint of an application or on the back-end. One can choose to try to distinguish the MAVLink messages from the different drones in that endpoint, but this can lead to complex routing systems within the endpoint. To solve these problems of the direct drone-client connection architecture, a protocol between the drone and the Back-end API has been set up and an UDP Port forward server was developed.

To make sure that not all drones send their MAVLink messages to the same port on the back-end, a protocol between the drone and the back-end is created. At drone startup time, the IP address of the back-end is given, which always has the same IP address. However, the drone needs to be given a specific port to which it can send its data. Not all drones can send their data to the same port because then the back-end would have issues in knowing which data stems from which drone. The solution used in this paper is to tie each drone to one specific port on the back-end. When a drone boots up, one of the first actions it performs is to send a request to the Back-end API to negotiate for a MavlinkPort. When the drone is registered to the platform, it receives a MavlinkPort. This Mavlinkport is the port on the UDP Port Forward Server to which the drone needs to send his MAVLink messages to. This MavlinkPort makes sure that only that specific drone is allowed to send to that specific port. This solves the problem that different drones would send their MAVLink messages to the same endpoint.

To solve the issues when a client IP address changes or another client wants to take over the drone, an UDP Port Forward Server is created. As the name implies, this server will forward UDP messages to another port on a different IP address. To configure the forwarding of the UDP messages, a REST request must be sent to the UDP Port Forward Server. This REST request will dynamically open or close the ports and configure the forwarding mechanism. Through this UDP Port Forward Server, multiple drones can send their MAVLink messages to the server and it forwards them to the right device of the client. Through the web application, the user can make sure that these MAVLink messages arrive on the application on the client's device or on an application on the back-end. The web application sends a request to the back-end API with

the configuration data. The back-end API will in turn extract additional data from the database and send it to the UDP Port Forward Server. Using this configuration data, the UDP Port Forward Server can open the MavlinkPort as well as a ClientPort on the UDP Port Forward Server. The ClientPort is a port that is used to send data from the client back to the MavlinkPort on the drones IP Address. This way, multiple drones can connect to the cloud and users can monitor all their drones with a ground control station.

When the IP address of the user changes or when another user wants to monitor or control the drone, this is no problem anymore. By reconfiguring the ports and IP addresses of the UDP Port Forward Server, a user can reconnect with their drone or another user can take control of a drone. We investigated how such an UDP Port Forward Server could be properly incepted. The first solution was to use a Router of which the Routing- and IPTables could be modified [25], [26]. This solution had one drawback, the router had to be rebooted every time the UDP Port Forward Server would set up new forwarding connections. The solution that therefore was used here is an API Server. On this server, UDP sockets (ports) can dynamically be opened and closed. Also, incoming requests with configuration details can be handled and ports can easily be set up. For simplicity, Express.js was used to set up an API Server. To forward the UDP Messages, several possibilities were evaluated. Eventually the node module dgram [27] was chosen to forward UDP messages. It has extensive documentation and can be configured easily.

C. Front-end

To ensure that users can easily manage their drones and their applications, a web application is created as illustrated in Figure 2. Through the web application, different Docker applications can be deployed on the drone. It is also possible to retrieve information, stop, restart and delete these applications. Through the web application, Docker images, which can be stored on Docker Hub or GitLab, can be linked to our solution. Those linked images can then be used for deployment of the Docker applications on the drone. To create this web-based front-end, several front-end frameworks were studied and the choice was made to use the framework React. React [28] is an UI library that uses a component-based architecture just like Angular. React is characterized by several features such as one-way-databinding [29], JSX [30] and the Virtual DOM [31]. Unlike the Angular framework, React does not have an app-level state. All the states are stored in all the different components of the web application. When a component wants to use information from another component, different components must pass their state to each other so that these components in their turn will pass their state to other components and so on. For larger projects, this can lead to errors and difficulties in managing the application. This problem can be solved by using Redux [32], [33] or Context API [34]. Redux and Context API [35] add structures and components to manage the app-level state. They also offer the possibility to work more conveniently by using this one central

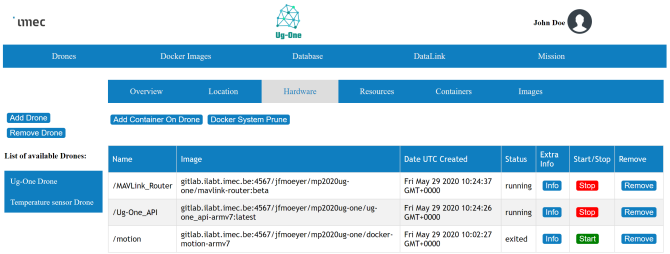

Fig. 2. The UG-One front-end showing the active containers on a specific drone

unit to store the state. When creating the web application, a combination of Redux and Context API was used [36], [37]. The Context API methods are used to recreate the structure and components of Redux. Redux itself is not easy to understand and to use. The combination of both app-level state managers makes it easier and faster for a developer to create a web application with an app-level state. But when creating a bigger more complex web application, Redux is recommended because it can manage the data better and clearer and offers tools to help manage the app-level sate.

For the deployment of the web server, NGINX was chosen [38]. This web server has a low memory load, is lightweight and offers the basic functionalities to host a web server.

V. Reference implementations and validation

In order to validate and demonstrate some of the capabilities of the UG-One platform, several component reference implementations have been created. These have the added benefit of demonstrating how certain applications can be implemented by developers that want to start developing for the UG-One platform. The source code is publicly available through https://github.com/IBCNServices/UG-One.

A. Video streaming

First of, a reference implementation [39] is created that connects to an on-board drone camera and streams video to both on-board and off-board applications. While this implementation can still be optimized significantly, it was already capable of providing a high-resolution stream to on-board and low-resolution stream to off-board applications, reducing bandwidth and battery usage.

B. Stress testing

The second application was created to simply run a stress test and use as much resources as possible. This application was used to validate the resource management capabilities of the UG-One drones. As expected, these applications could never exceed their memory limit and could only claim CPU time in relation to their weight. This means that no critical container with a significantly higher CPU share value, or weight, ever experienced any real slowdowns while a stress tests was running.

C. MAVLink application

Finally, as mentioned before, one of the reference applications was a container that reads out telemetry data provided by the autopilot over MAVLink.

VI. Managing a drone fleet

Users can deploy and manage their applications, request system resources and more through the REST API on the drone. But this way of working is not user-friendly and certainly not when multiple applications need to be managed on multiple drones. To be able to manage those applications in a fast and easy way, a back-end cloud infrastructure is needed. The proposed cloud infrastructure that was created offers a web application that enables users to easily deploy applications, keep an overview of their drone fleet, manage resources on the drones, etc. Because the web interface shows dynamic content to the users, there is also a need for a back-end API and a database. This back-end API is the central unit of the cloud infrastructure which sends and receives requests to and from the web server, database, drones, API servers, etc. Next to the web server, back-end API and database, there is one last component present on the cloud, the UDP Port Forward Server.

VII. Conclusion

In line with the established cloud service models, a new UAVaaS model and platform has been created, allowing developers with no previous drone experience to develop and deploy applications on drones. Container technologies are used to offer the convenience of application and drone management through a web interface while it also continues to be modular and very capable as well. Applications can run on any drone that has the required hardware components and often can run on the back-end as well with little to no modification.

As resources on the drone on-board computing unit are considered scarce, in future work we will investigate the use of more secure container technology alike Kata containers [40] and on using uni-kernels, stripping out functionality not required by the applications running on the drone.

References

[1] B. Bera, D. Chattaraj, and A. K. Das, "Designing secure blockchain-based access control scheme in iot-enabled internet of drones deployment," *Computer Communications*, vol. 153, pp. 229–249, 2020.

[2] G. Choudhary, V. Sharma, and I. You, "Sustainable and secure trajectories for the military internet of drones (iod) through an efficient medium access control (mac) protocol," *Computers & Electrical Engineering*, vol. 74, pp. 59–73, 2019.

[3] M. B. Ghorbel, D. Rodriguez-Duarte, H. Ghazzai, M. J. Hossain, and H. Menouar, "Energy efficient data collection for wireless sensors using drones," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*. IEEE, 2018, pp. 1–5.

[4] S.-C. Choi, N.-M. Sung, J.-H. Park, I.-Y. Ahn, and J. Kim, "Enabling drone as a service: Onem2m-based uav/drone management system," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2017, pp. 18–20.

[5] (2020) Pixhawk. [Online]. Available: https://pixhawk.org/

[6] (2020) Ardupilot. ArduPilot. [Online]. Available: https://ardupilot.org/

[7] (2020) Mavlink developer guide. Dronecode. [Online]. Available: https://mavlink.io/en/

[8] (2019) PX4 documentation v1.10.1. [Online]. Available: https://docs.px4.io/v1.10/en

[9] (2020) Mavsdk documentation. Dronecode. [Online]. Available: https://mavsdk.mavlink.io/develop/en/

[10] Mavlink router. Intel Corporation. [Online]. Available: https://github.com/intel/mavlink-router

[11] (2020) Docker. [Online]. Available: https://www.docker.com/

[12] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[13] (2020) Express. express. [Online]. Available: https://expressjs.com/

[14] (2020) axios. npm. [Online]. Available: https://www.npmjs.com/package/axios

[15] (2020) Docker-hub-api. npm. [Online]. Available: https://www.npmjs.com/package/docker-hub-api

[16] (2020) node-gitlab. npm. [Online]. Available: https://www.npmjs.com/package/node-gitlab

[17] (2020) node module gitlab. npm. [Online]. Available: https://www.npmjs.com/package/gitlab

[18] (2020) Projects api. gitlab. [Online]. Available: https://docs.gitlab.com/ee/api/projects.html

[19] (2020) npm mongoose. npm. [Online]. Available: https://www.npmjs.com/package/mongoose

[20] V. Abramova and J. Bernardino, "Nosql databases: Mongodb vs cassandra," in *Proceedings of the international C* conference on computer science and software engineering*, 2013, pp. 14–22.

[21] V. Sharma and M. Dave, "Sql and nosql databases," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 8, 2012.

[22] A. Korpal. (2019) Scaling horizontally and vertically for databases. [Online]. Available: https://bit.ly/2Bas3sJ

[23] R. Greiner. (2014) Cap theorem: Explained. [Online]. Available: https://robertgreiner.com/cap-theorem-explained/

[24] (2020) Uav ground control station. Unmanned system technology. [Online]. Available: https://www.unmannedsystemstechnology.com/category/supplier-directory/ground-control-systems/ground-control-stations-gcs/

[25] E. Ma. (2019) Port forwarding using iptables. [Online]. Available: https://www.systutorials.com/port-forwarding-using-iptables/

[26] P. S. (2019) Iptables tutorial. [Online]. Available: https://www.hostinger.com/tutorials/iptables-tutorial

[27] (2020) Udp/datagram sockets. Node.js. [Online]. Available: https://nodejs.org/api/dgram.html

[28] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, "Single page application using angularjs," *International Journal of Computer Science and Information Technologies*, vol. 6, no. 3, pp. 2876–2879, 2015.

[29] (2020) Data binding. angularjs. [Online]. Available: https://docs.angularjs.org/guide/databinding

[30] (2020) What is jsx. react enlightment. [Online]. Available: https://www.reactenlightenment.com/

[31] (2020) Understanding the virtual dom. bitsofcode. [Online]. Available: https://bitsofco.de/understanding-the-virtual-dom/

[32] (2020) Redux in react. Reactredux. [Online]. Available: https://react-redux.js.org/introduction/why-use-react-redux

[33] M. K. Caspers, "React and redux," *Rich Internet Applications w/HTML and Javascript*, p. 11, 2017.

[34] (2020) Context in react. Reactjs. [Online]. Available: https://reactjs.org/docs/context.html

[35] A. B. Thakur. (2020) Redux vs context api. [Online]. Available: https://dev.to/ayushmanbthakur/redux-vs-context-api-3182

[36] T. Media. (2020) Node.js & express api — expense tracker. [Online]. Available: https://youtu.be/KyWaXA_NvT0

[37] P. Ranasinghe. (2019) Build a redux-like store with react context and hooks. [Online]. Available: https://bit.ly/36pGxAJ

[38] (2020) Nginx web server. NGINX. [Online]. Available: https://www.nginx.com/

[39] N. Tijtgat, W. Van Ranst, T. Goedeme, B. Volckaert, and F. De Turck, "Embedded real-time object detection for a uav warning system," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 2110–2118.

[40] A. Randazzo and I. Tinnirello, "Kata containers: An emerging architecture for enabling mec services in fast and secure way," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019, pp. 209–214.