Feature Article: Microservices in robotics for environmental research

Developing a re-configurable architecture for the remote operation of marine autonomous systems

Alvaro Lorenzo Lopez, *National Oceanography Centre (UK) and Universidad de Las Palmas de Gran Canaria (Spain)*

Ashley Morris, *National Oceanography Centre (UK)*

Owain Jones, *National Oceanography Centre (UK)*

Alexander B. Phillips, *National Oceanography Centre (UK)*

Francisco Mario Hernandez Tejera, *SIANI, Universidad de Las Palmas de Gran Canaria (Spain)*

Adrian Penate-Sanchez, *SIANI, Universidad de Las Palmas de Gran Canaria (Spain)*

Abstract—In this experience report, we explain how we take advantage of microservices' inherent modular nature to accomplish a highly adaptable software architecture that can deal with the trials and tribulations often occurring in marine research environments. We will show the National Oceanography Centre's journey to develop a web system to remotely operate marine autonomous vehicles from anywhere in the world with an internet connection and how, due to new unforeseen requirements, we took the microservice pattern into a new direction to allow for standalone offline operations of Autonomous Underwater Vehicles (AUV) from research ships in some of the most challenging environments in the world.

INTRODUCTION

Operating ocean robots for scientific or commercial purposes is becoming more and more common; academia, defence, and industry spend time and money developing underwater and surface autonomous vehicles to cover a variety of scenarios, including environmental monitoring, water quality assessment, climate change studies, fauna identification or oil rig decommissioning.

At the National Oceanography Centre (NOC) we host the UK National Marine Facilities (NMF). NMF oversees and operates the National Marine Equipment Pool (NMEP), providing operational services and developing state-of-the-art technology enabling the UK marine science community to produce world-class research. We are part of the Maritime Autonomous Robotics Systems (MARS), and we operate the most extensive fleet of marine autonomous systems (MAS) for research in Europe, with more than 35 commercial ocean gliders and eight in-house developed Autonomous Underwater Vehicles (AUVs), including the famous Boaty McBoatface. To do remote operations, we require wireless technologies, including local Wi-Fi, remote satellite, and acoustic communications, to send execution plans to the vehicles. Our pilots utilize each MAS technology using the software provided by the manufacturer, which, in most cases, is not interoperable with other technologies. This lack of interoperability is not a problem piloting each MAS platform individually, but deploying fleets of vehicles from different vendors is becoming more common. Our operations span varying timeframes, from weeks to potentially years. Managing diverse MAS fleets presents the challenge of transitioning between different piloting applications, risking mistakes due to shifting visual paradigms. The constant change in contexts burdens cognitive load and demands more human resources. Rather than having a unified operator pool, we're forced to assign dedicated pilots to minimize errors, which is unsustainable.

The lack of standardized machine-to-machine interfaces complicates automation via machine learning

XXXX-XXX © 2023 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2023.3317065

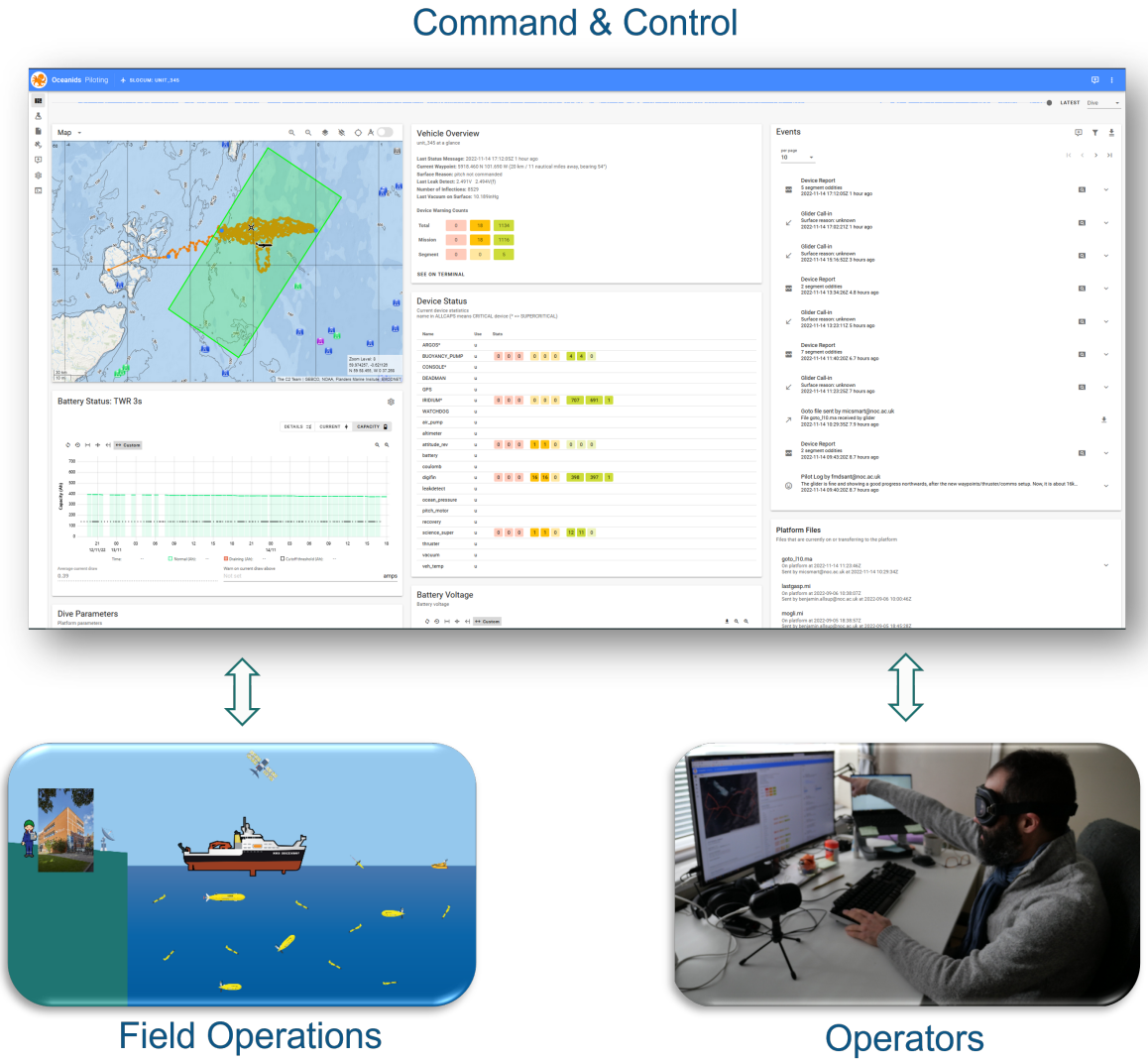

FIGURE 1. The Command and Control (C2) concept: Vehicles are operating remotely at sea, they call back to their control base stations and the telemetry data sent back gets aggregated by our C2 system, allowing pilots to interact with the vehicles through a web application from anywhere in the world.

or probabilistic path planning. Developers struggle to integrate diverse robotic platforms.

In 2016, the National Environmental Council (NERC) launched Oceanids, funding the Command and Control (C2) initiative. Its aim was a unified MAS operations software, reducing context switching for pilots (see Figure 1).

This report outlines C2's development, our experiences, and technical decisions. We offer insights into creating a Microservice system for marine research and software development within an ocean research institution, hoping to benefit IEEE Software readers.

The C2 Team

To build the C2, our team was created. We are part of a large marine research institution, where we develop software that supports ocean sciences. The C2 falls under the category of scientific-software with users not fully understanding the domain with no precise upfront requirements [1] [2].

When starting the C2 project, two software engineers were on the team. At that time in the UK (2016), there was a big push in science communities to embrace the Research Software Engineer (RSE) [3]

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2023.3317065

profile, trying to recognize software practitioners developing scientific-software. Our management believed we could integrate other RSEs from other departments and organizations to build the C2.

team has expanded from two engineers to seven, including frontend developers, backend engineers, and RSEs. This diverse team has contributed to shaping the architecture and functionalities of C2 with the frontend engineers helping us to organize backend development around end-user features, or the RSEs with expertise on robotics influencing the API design to abstract the MAS piloting and enable the future connection of generic piloting algorithms. The team's growth has been organic, with new members bringing valuable skills to meet evolving demands.

THE SYSTEM

We embarked on the project with the primary objective of unifying piloting systems for different MAS platforms. To achieve this, we recognized the critical need for uninterrupted access to the system. MAS platforms are typically operated from ships nearby, but we wanted to enable remote piloting, allowing operators to command MAS from their base or home, leveraging satellite communication. Ensuring round-the-clock availability became paramount as operators needed to periodically check vehicle behaviour, collect accurate data, and intervene in real-time to address changes in tracked ocean features or vehicle malfunctions, all while preserving platform safety. We strategically decided to build a web system instead of a traditional desktop-based application to meet this requirement.

Additionally, we identified the future potential for developing different applications tailored to specific user communities. These included a piloting app for operators, a science app for real-time data analysis, a risk and reliability app for engineering fault analysis, and visualization portals for the general public. Despite their unique user interfaces and functionalities, these applications would share a significant amount of underlying information. Hence, we decided to develop individual backends encapsulated with restful APIs. This approach allowed us to reuse and share components across multiple applications, ensuring efficient development and maintenance.

Due to limited resources, our small team developed the system using small, easily integrated and maintainable components. We planned to build prototypes to gather feedback from MAS operators and iterate quickly on requirements. Considering the flexibility required, we adopted the Microservice pattern [4], which enables the development of independent, loosely coupled services. This approach allows developers to break down problems into manageable subsystems, test them independently, and add new functionality without disrupting the rest of the system. Using microservices not only offered us flexibility, but we thought we would increase reliability by having smaller subsystems to maintain. We also liked the idea of portability (we will touch on containerization), as we thought we could deploy on different clouds. However, as we will see later, we exploded portability slightly differently.

Although we had an existing application built in Symphony, a precursor to the C2 system, we decided against a monolithic approach. Integrating the work of other groups would be more complex in a monolithic system, requiring deep knowledge of the entire system. We briefly considered SOA but found it too complex for our small team. Ultimately, the Microservice pattern aligned with our requirements, allowing us to develop RESTful web APIs as small, manageable systems and enabling external contributions while maintaining oversight of service integration.

DEVELOPING THE SYSTEM

We wanted a system that would allow us to quickly integrate work from others, which was an important consideration when we researched architectural patterns. Initially, our architecture wasn't very complex, and we focused on integrating the microservices to create frontend apps easily. We decided to do it using the API gateway pattern [5], which puts an intermediate entity (the gateway) in the middle of clients and services. The gateway pattern allows the implementation of the Authentication, load balancing, and service discovery within the gateway itself. We will talk more about the gateway in the next section.

We started without a defined development stack. In our first year, while designing the architecture, we investigated different technologies to develop the system.

We made the conscious decision to begin developing our microservices in Python, a well-established programming language in RSE communities, providing tools for web development and with increasing traction within data science and RSEs, with the use of popular libraries such as numpy, pandas, scipy, matplotlib, TensorFlow and PyTorch. In recent years, the RSE community had tended to prefer open-source, free and general-use tools, such as Python and its libraries, over licensed or domain-specific languages (Matlab, R), making it an attractive choice when we considered that we would likely want to integrate code written as

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2023.3317065

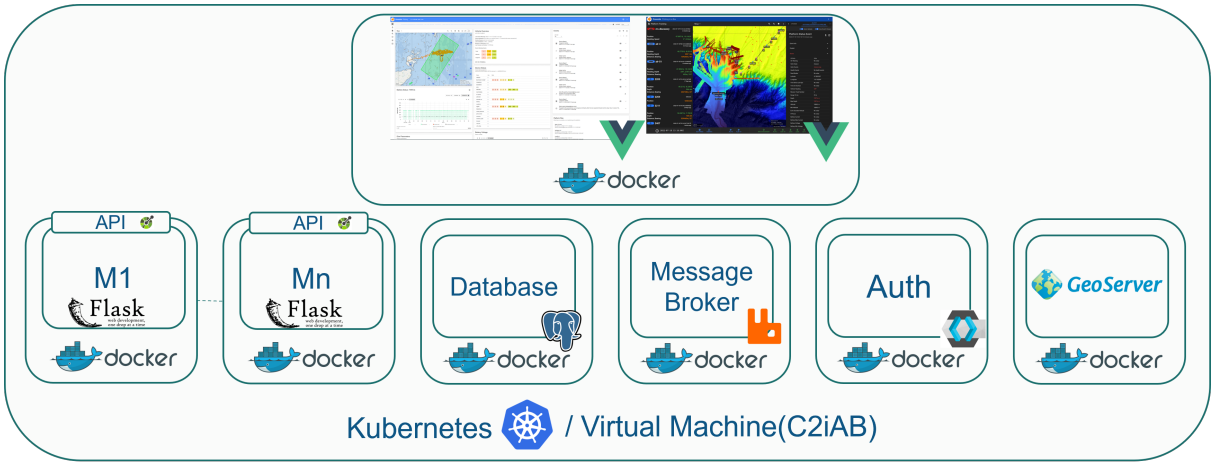

FIGURE 2. This is our stack at the beginning of 2023. We used the same microservices for our central internet-based solution (C2) and C2iAB, the system we use for operations without internet connectivity (Ships, remote areas, etc.). The only difference is that C2iAB run the entire system on virtual machines without the overhead of Kubernetes, and some of the services connect to hardware directly to talk to the robots in the field.

outputs of research projects. We considered the use of Python would make it easier for RSEs to get introduced to develop our systems and, at the same time, help external project collaborators to contribute code.

We chose Flask as the framework to develop microservices; while it is true that Python is not as fast as other languages (Golang, Scala ...), as Flask is a lightweight Python framework it helped us to keep microservices lean. We slowly incorporated the libraries we needed, like the sqlalchemy ORM or flask-restx to automatically generate Swagger/OpenAPI schemas. We now have an easy-to-use framework to speed up new functionality development and consistent maintainable code across our services. This framework provides us with everything we need when building microservices, from access control to database access; we called this framework Microserver. Microserver offers more than just Python scaffolding for business logic. It includes the tools to containerize and deploy the services within our Kubernetes cluster. We build most of our microservices with our Microserver, but it is not mandatory; we can use any language or framework, but Microserver makes our work easier.

We chose Postgres as our database due to its versatility. While we considered other options like MongoDB, InfluxDB, and ElasticSearch for specific use cases, maintaining multiple technologies proved challenging for our small team. Postgres, with plugins like PostGIS and Timescale, met our needs for storing geographical and time-series data efficiently. It offers flexibility while being well-documented and mature. If necessary, we can easily integrate new database technologies alongside Postgres using microservices.

We brought the architecture together using RabbitMQ(AMQP) real-time messaging, allowing the microservices to publish events to be picked up and processed by other microservices.

For frontend development, we use Vue.js. We experimented with Angular and native JavaScript first. Still, either option was more complex or bare-bones, with Vue giving us a mix of modern patterns and simplicity that we judged optimal then.

Our stack (see Figure 2) build using well-tested open-source technologies to create new functionality efficiently and can be adapted for different applications. In figures 3 and 4, we show slightly different applications of the same technologies.

KUBERNETES AND DEVOPS

Much of our journey has revolved around learning how to orchestrate our containerized microservices. Our original plan was to deploy our system in the cloud. Still, the complexity of estimating cloud bills upfront, the fact that research infrastructure projects have a fixed length, and the uncertainty of no follow-up money made us deploy and manage in-house.

Mastering the usage of Docker as our container solution took little time, but the orchestration with Kubernetes was a whole different story. It took us three iterations until we developed a stable solution.

The first iteration was a prototype Kubernetes in-

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2023.3317065

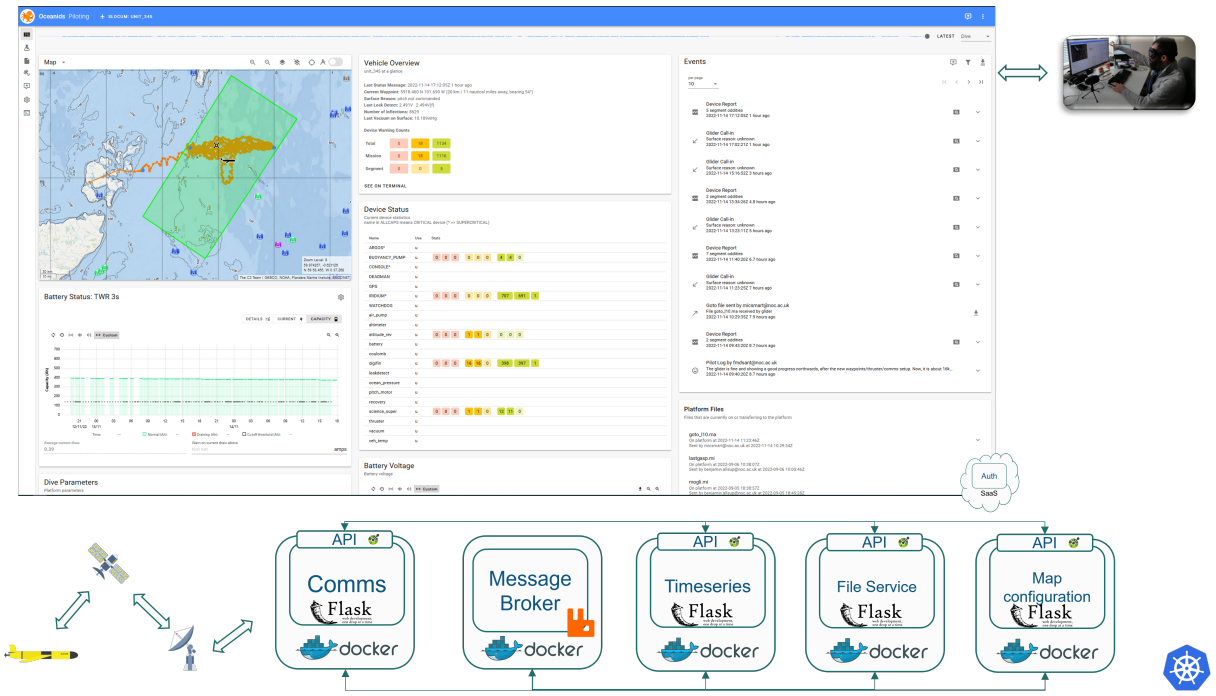

FIGURE 3. This is our piloting App, operating an ocean glider in the north sea. The robot calls using satellite communications to a base station. Our C2 system monitors the base station with a microservice managing incoming and outcoming communications. When an incoming communication arrives, the communication microservices tell the rest of the microservices using RabbitMQ, and each interested microservice accesses the information from the communication microservice using APIs. All the microservices use the same communication flow. Our frontend calls the different microservices APIs to show pilots the operational state and then sends operator instructions to the vehicles. In this figure, we are still showing our old authentication provider. The whole system is running on a Kubernetes cluster.

stallation to learn the technology and compare it with Docker Swarm, the built-in orchestration that used to come with Docker. Docker Swarm was simple to use, but the rise of Kubernetes as a preferred option in the industry came with plenty of online documentation, making it easier for us to understand how to configure it. In this first cluster, we deployed our prototype piloting app. However, we soon realized that this cluster would not be sustainable in the long run. It ran on a single VM node, limiting our growth, and was poorly configured due to its prototyping nature. Additionally, we decided that using the WSO2 gateway and the authentication system, both maintained by a different department, was unsuitable for us as we had no control over configuring and customizing them. Therefore, we decided to build a new cluster.

For the second iteration, our goal was to create a production-ready solution. We designed the cluster with three VM nodes and deployed the Ambassador gateway (now called Emissary) in its open-source version, replacing the WSO2 gateway. Ambassador was straightforward to configure and was deployed as part of the Kubernetes cluster, allowing us to easily add new API endpoints to the gateway using annotations in the Kubernetes services. To automate this process, we added a default template service file to Microserver, which simplified the creation of services by allowing programmers to configure specific environmental variables. The services were automatically deployed and loaded in the cluster, and the API endpoints were added automatically to the gateway. While Ambassador lacked some features compared to WSO2, it provided us with what we needed: the ability to configure the gateway as code, deploy it within the Kubernetes cluster, and easily add APIs. We re-implemented the authentication by integrating an OAuth workflow with Auth0, a Software as a Service (SaaS). This freed us from implementing and maintaining an authentication system in-house.

Although the second iteration solved several crucial problems, it also brought new challenges. We underestimated the knowledge required to configure

and maintain a Kubernetes installation, as the multiple layers involved in providing functionality to multiple systems (microservices) running on containers and communicating with each other through internal Kubernetes networks proved complex, especially when troubleshooting issues. Our cluster was unstable, with pods randomly going down and not recovering. We attempted to implement observability tools using ELK (Elasticsearch, Logstash, and Kibana) or Prometheus. Still, these systems' in-house configuration and maintenance added more work for the team, exacerbating the issues.

Despite the challenges, the second iteration hosted our first semi-operational piloting application, which was used in the field for trials with positive results, allowing us to operate the different vehicles. However, frustrations arose, particularly during a Loch Ness Trial, which experienced half-hour downtime in the morning almost daily. We recognized the need for improvement.

We had plans for the third iteration in June 2020, but an unforeseen opportunity presented itself due to the COVID-19 pandemic. With the world coming to a halt, our operational program was put on hold, allowing us to reevaluate our working practices and address the stability issues in our Kubernetes cluster.

We faced several challenges in our Kubernetes environment, including difficult-to-track configuration changes, manual deployment of microservices leading to a slow and cumbersome change process, a testing environment that differed significantly from production, infrequent deployment of microservices causing delays in user improvements and necessitating large release deployments, inadequate observability of the cluster with ineffective ELK and Prometheus tools, difficulties in updating Kubernetes resulting in being stuck on outdated versions with associated problems, and random pod failures without clear causes.

We identified the lack of processes as the main problem; releasing code into production was a manual and convoluted process, challenging to manage. To address these issues, we decided to implement best practices used in the software industry, particularly by DevOps practitioners [6] [7], focusing on automation and stability.

We implemented several key actions to enhance our Kubernetes environment.

First, we designed two identical Kubernetes clusters dedicated to testing and production. To facilitate cluster deployment, administration, updates, and monitoring, we chose Rancher, a Kubernetes distribution that offered a user-friendly GUI.

Next, we adopted a code management workflow based on Git Flow. This approach involved two branches: "main" for recording releases and "develop" for integration. Developers create feature branches for new functionality, eventually merging into the development branch and releasing to the main branch. For improved version alignment, we release and version all microservices simultaneously. This aids in change tracking and stable version rollbacks. All code changes pass through peer reviews and merge requests before merging into the development branch. All code changes underwent peer reviews and merge requests before merging into the development branch.

We embraced an infrastructure-as-code approach to configure our Kubernetes clusters. Configuration files were stored in Git repositories, following the Git Flow model. This facilitated version control and enabled us to roll back to stable states when necessary easily.

Furthermore, we automated merging code into development and releasing it using GitLab CI and Argo CD. Although the human intervention was still required to trigger deployments to production, the overall process became significantly more automated. Rolling back to the previous version followed the same streamlined process if needed.

Throughout 2020, we dedicated most of our efforts to implementing these changes and adapting to new work practices. Although Continuous Delivery has not been fully achieved, we have embraced many similar practices. We deploy to production at least once a month, experiencing minimal downtime and increased cluster reliability. While isolated service issues may occur occasionally, cluster-wide problems are rare, and Rancher assists in faster error detection and resolution.

RE-CONFIGURING THE SYSTEM

Our primary focus from the beginning of the project was developing a centralized system to perform over-the-horizon piloting. While traditionally, AUVs are operated from ships, we deliberately ignored that use case as operational teams were exploring adopting an off-the-shelf solution (commercial or open source). Still, none of the plans progressed, and the need to operate newly in-house developed AUVs became a priority as the Autosub Long Range (ALR) Boaty McBoatFace was going to Antarctica to be part of one of the most ambitious and high-profile environmental studies in recent years; a joint operation between the US NSF and UK NERC, TARSAN. In 2021, we needed to develop a tool to operate Boaty and the rest of our AUVs from ships.

In this section, we demonstrate adapting microser-

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2023.3317065

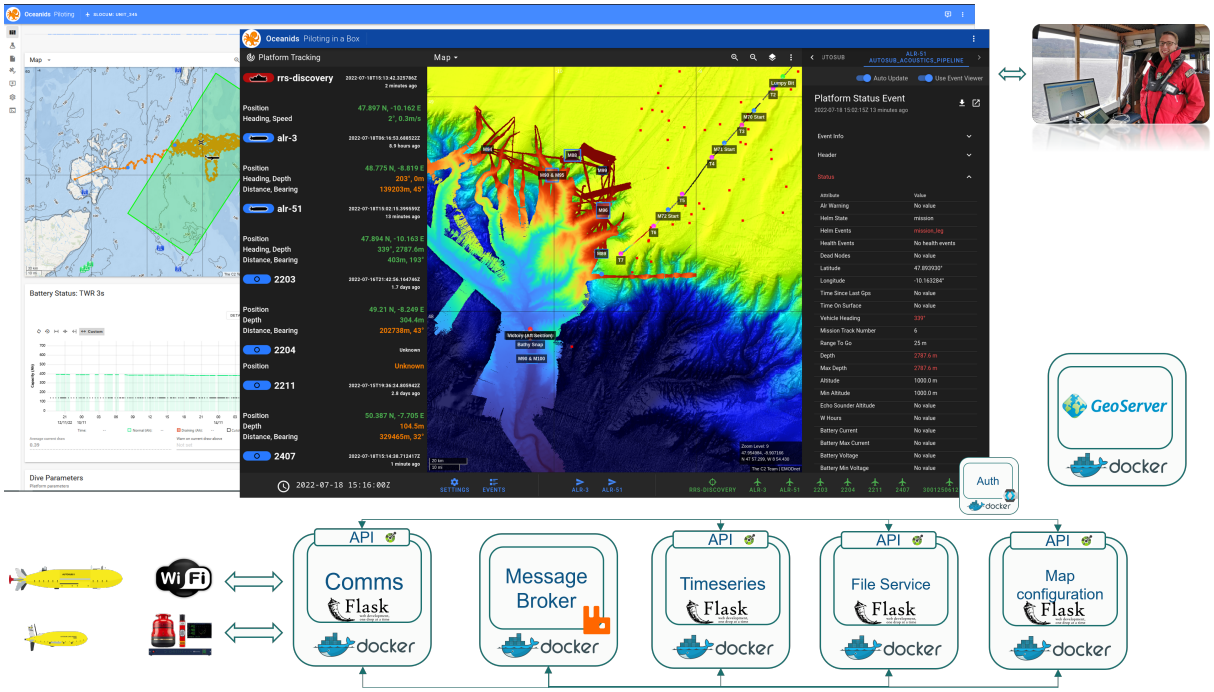

FIGURE 4. An example of how we have reconfigured our microservice system to accommodate the case of piloting from a ship; We have adapted our microservices to connect directly to the hardware in the ship (in this case, the RSS Discovery) using our communication microservice, we wrote integrations for Wi-Fi, to talk to the AUVs when on deck or in the water near the ship, and then USBL (ultra-short baseline, an underwater acoustics system), to track the AUVs underwater and send short commands to them (like emerge or start a mission). The example shown is during the cruise DY152. The integration of GeoServer allows us to show layers locally without needing an internet connection; in the map on the piloting in a box app, we are showing high-density bathymetric layers provided by the ship instruments.

vices for offline use, which is vital for research ships with limited or no internet. Our solution is dubbed "C2 in a box" (C2iAB).

With some modifications, our original system functionality could serve as a minimum viable product (MVP). Leveraging the modularity of microservices, we swiftly integrated third-party systems to enable the C2iAB functionality. While utilizing Auth0, it became evident that the system needed to operate independently of an internet connection. We opted to replace the SaaS OAuth solution with an offline system to address this. Keycloak, an open-source identity and access management service with a strong reputation and a large user community, was our chosen solution. Its OAuth flows closely resembled our existing implementation, ensuring a relatively straightforward replacement process. This change was implemented within our primary C2 solution, promoting architectural unity and reducing reliance on external providers.

Quickly including new externally developed services showed the adaptability of the microservice architecture; C2 uses online maps (Leaflet) to show operators the vehicle positions and other environmental information, but "disconnecting" the system from the internet also removed our access to maps. Deploying a GeoServer as a local GIS solution allows us to cache and show maps and map layers locally. Using GeoServer proved successful as we integrated products from the ship systems like real-time bathymetry, enriching our operators' visualization and situational awareness.

We decided to strip Kubernetes from C2iAB as it did not work when trying to connect to local ship networks with their own DHCP and, in general, as we did not need features like the high availability. We deployed the C2iAB directly on virtual machines as a single node. We managed the local infrastructure by combining docker-compose with ansible scripts. One of the challenges was how to "replicate" a lot of the features dependent on Kubernetes, like the API gateway; to do it, we run a series of scripts that for through all our microservice repositories and extract things from

Field Operations						
Operation name	**Dates**	**C2 Variant**	**Area of operation**	**Vehicles**	**Days at sea**	**Operators**
Altereco	Summer 2019	C2	North Sea	Two Gliders	74	5+
A68 Iceberg Tracking	February 2021	C2	Southern Ocean	Two Gliders	143	10+
TARSAN	January 2022	C2iAB	Antarctica	ALR	6	2
Long distance proving trial	May 2022	C2	UK Shelf	ALR	36	7+
DY152	June 2022	C2iAB	UK Shelf	ALR, Autosub-5 and 2 gliders	15	5+
In-site at sea	September 2022	C2	North Sea	ALR	42	5

TABLE 1. Examples of relevant operations since 2019 to showcase the kind of field operations our system handles. More operations have already been completed using the system, but we showcase the most relevant ones here.

them, for example, the annotations defining the API endpoints, assembling them on the docker-compose script to create an NGINX reverse proxy acting as a local gateway.

The development of C2iAB helped us refine the whole C2 system, and all the changes we made for C2iAB have made their way back to the original C2 except for some particular configurations, like Kubernetes. Figure 2 shows the current unified stack (2023) with all the changes described. Microservices for C2 and C2iAB remain under the same codebases, and it is a matter of choosing in which configuration we want to use them on deployment time.

In January 2022, we sent a C2iAB to its first science campaign, envisioned initially to operate the ALR (Boaty McBoatface) under the Thwaites glacier.

LESSONS LEARNED AND FUTURE DIRECTIONS

After seven years of development, we have produced stable solutions that get used to operating MAS platforms on the field; further details are in Table. 1, and we have picked up some lessons along the way.

Developing software in science-driven organizations is different, with no systematic approach to software development or use of software development best practices. In our case, the requirements were not precise. Microservices have allowed us to manage some uncertainty as we built our system on small systems through iterative phases. We think this is an excellent way to develop scientific-software. This approach can help to change course if new requirements arrive, as we have shown with the C2iAB; we took a classical microservice web application and turned it into an offline system with direct access to hardware devices.

A hard lesson for us has been the adoption of software best practices (like using open-source technologies or the adoption of DevOps); this was not common in our organization, and we dare to say it wasn't common in most science-driven organizations at the time, but we are a clear example that it can be done. It pays off, making software development more robust.

We praise the use of microservices, but we also must warn people to be careful when using them on public-funded scientific organizations; if teams can deploy the microservices on commercially managed Kubernetes instances, operations become easier, but if your team is small (as ours) the only way to manage a complex microservice architecture is implementing very good DevOps practices and at least for us the use of Kubernetes distributions (Rancher) [8] helped a lot.

One of our regrets has been not open-sourcing all our code from the beginning; we believe other people working in similar environments would benefit from our technology (things like Microserver), but when we started, we had not enough time to do it. Our organization and we are committed to open source the system, but it will take some time as we do it while still delivering new functionality.

Our principal founder, UK Research and Innovation, has set very ambitious plans and projects to transition the UK research infrastructure to net zero carbon emissions. One of the projects is the Net Zero Ocean Capability (NZOC), which aims to reshape the paradigm of using traditional research vessels as the centre of oceanographic activities, dramatically increasing the use of ocean autonomy. A key component will be automating the observing system operations, leveraging the introduction of Digital Twins of the Ocean (DiTTO) to command MAS platforms based on science criteria, such as acquiring observations to improve an ocean

model. While there is still not much-published work on DiTTOs, they are grouped under Digital Twins of the Earth [9] to create a digital representation of extensive environmental systems to assist in the research. There have been a few experiments following the approach we got in mind, both done from the R.V. Falkor; a deployment of MAS in the Pacific [10], combining multiple MAS platforms doing sampling guided by a model, and one in the Baltic Sea [11], with a DiTTO on the ship been fed information from platforms around and again leading the observational campaign. We aspire to make C2 the layer connecting DiTTOs with the observing system, allowing them to reconfigure it automatically.

ACKNOWLEDGMENTS

The development of the C2 infrastructure was funded by the UK's Industrial Strategy Challenge Fund (ISCF)—Natural Environment Research Council (NERC) Oceanids Capital investment in Marine Autonomous Systems.

The authors want to thank and acknowledge the work and contributions of Dr Catherine Ann Harris (NOC, UK), Dr Justin Buck (NOC, UK), Jack Farley (DOC, NZ), Jim Bacon (CEH, UK), Izzat Kamarudzman (NOC, UK), Trishna Saeharaseelan (NOC, UK), James T Kirk (NOC, UK) and Dan Jones (NOC, UK). They have contributed to the system design or work in the code itself. Special thanks to Dr Harris and Dr Buck, who have played pivotal roles in making this work possible, being part of the team, listening and advising throughout the journey.

Finally, we would like to thank Dr Kristian Thaller, the Oceanids program manager, for always being patient with this team of crazy software engineers, and to Dr Maaten Furlong (current director of NMF) and Leigh Storey (previous director of NMF) for giving the opportunity of developing the C2 and trust us in all our decisions.

REFERENCES

[1] Judith Segal and Chris Morris. "Developing scientific software". In: *IEEE software* 25.4 (2008), pp. 18–20.

[2] Jeffrey C Carver et al. "Software development environments for scientific and engineering software: A series of case studies". In: *29th International Conference on Software Engineering (ICSE'07)*. IEEE. 2007, pp. 550–559.

[3] Stephen Crouch et al. "The Software Sustainability Institute: Changing Research Software Attitudes and Practices." In: *Comput. Sci. Eng.* 15.6 (2013), pp. 74–80.

[4] *Microservices a definition of this new architectural term*. URL: https://martinfowler.com/articles/microservices.html.

[5] *Microservice Architecture*. URL: https://microservices.io/.

[6] Len Bass. "The software architect and DevOps". In: *IEEE Software* 35.1 (2017), pp. 8–10.

[7] Florian Beetz and Simon Harrer. "GitOps: The Evolution of DevOps?" In: *IEEE Software* 39.4 (2021), pp. 70–75.

[8] Marek Moravcik et al. "Kubernetes-evolution of virtualization". In: *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE. 2022, pp. 454–459.

[9] Jörn Hoffmann et al. "Destination Earth – A digital twin in support of climate services". In: *Climate Services* 30 (2023), p. 100394. ISSN: 2405-8807. DOI: https://doi.org/10.1016/j.cliser.2023.100394.

[10] Jose Pinto et al. "Coordinated Robotic Exploration of Dynamic Open Ocean Phenomena". In: *Field Robotics* 2 (Mar. 2022), pp. 843–871. DOI: 10.55417/fr.2022028.

[11] Alexander Barbie et al. "Developing an underwater network of ocean observation systems with digital twin prototypes—a field report from the baltic sea". In: *IEEE Internet Computing* 26.3 (2021), pp. 33–42.

Alvaro Lorenzo Lopez is a senior software engineer at the National Oceanography Centre, United Kingdom, leading the NOC-MARS Development C2 team and the research presented in this paper. His current research interests include standardizing command and control systems and their interoperability. He holds a degree in Computer science from the University of Las Palmas de Gran Canaria (Spain) and is currently a PhD candidate at that same University. Contact him at alvaro.lorenzo@noc.ac.uk

Ashley Morris is a Software Engineer at the National Oceanography Centre, United Kingdom, part of the NOC-MARS Development C2 team. Ashley has led the technical development of C2iAB. His current research interests include Human-Machine Interfaces, AUV operations, frontend web technologies, Software Engineering principles and the Software Development Lifecycle. He holds a BSc in Computing from the Bournemouth University (UK). Contact him at ashley.morris@noc.ac.uk

Owain Jones is a software engineer at the National Oceanography Centre, United Kingdom, part of the NOC-MARS Development C2 team. Owain leads the C2 infrastructure and DevOps. His current research interests include remote sensing, earth observation, geoinformation systems (GIS), distributed computing and AI/ML pipelines. He holds an MSc in Computer Science from Abersytwyth University (UK). Contact him at owain.jones@noc.ac.uk

Alexander Philips is the head of development at the MARS department in the National Oceanography Centre, United Kingdom. He oversees a big portfolio of autonomy development projects, including C2. His current research interests include AI and software engineering. He holds a PhD in hydrodynamics of underwater vehicles from Southampton University (UK). Contact him at abp@noc.ac.uk

Francisco Mario Hernández Tejera is a professor at the University of Las Palmas de Gran Canaria in the area of Computation and Artificial Intelligence Sciences. His research interests are Real-Time Artificial Vision, Image Processing, Shape Recognizing and Automatic Learning. He is an Industrial Engineer with a PhD. in Informatics from the University of Las Palmas de Gran Canaria (Spain). Contact him at mhernandez@iusiani.ulpgc.es.

Adrian Penate-Sanchez is a lecturer at the University of Las Palmas de Gran Canaria, Spain. His current research interests include Machine Learning and general Artificial Intelligence. He holds a PhD in computer science and artificial intelligence from the Universitat Politecnica de Catalunya (Spain). Contact him at adrian.penate@ulpgc.es