NERC Environmental Data Service

Next-generation sensor data acquisition onboard NERC deep water research vessels

A deliverable report for the AMPLIFY-EDS project

Authors:

Alex J. Tate[1] https://orcid.org/0000-0002-7880-3271
Michael C. Crosier[1]
Claudette J. Lopez[1] https://orcid.org/0000-0002-4840-0383
Benjamin Dawson[1] https://orcid.org/0009-0000-1500-7772
Daniel P. Phillips[2] https://orcid.org/0000-0002-0280-4788
Martin Bridger[2]
Juan Ward[2] https://orcid.org/0000-0002-5672-6842

Affiliation:

1 – British Antarctic Survey
2 – National Oceanography Centre

Date: 30 November 2025

Executive Summary

The aim of Amplify activity 1.1b was to inform the design of a NERC Environmental Data Service (EDS) sensor commons (activity 1.1a) through a series of experiments and reflections. This report describes the outcomes of one of these experiments, as envisaged in the original proposal:

> *"In the first experiment, British Antarctic Survey (BAS) and the National Oceanography Centre (NOC) will co-develop a prototype of a reimagined vessel data acquisition system based on cloud-native principles, including the use of Kubernetes to enable a heterogeneous set of software to work together robustly. In addition, this prototype will investigate the use of a stream processing platform such as Apache Kafka which will allow: different software components to communicate without knowing about each other; in-flight processing/transformation of data streams for near-real time QA/QC; delivery guarantees with persistent, fault tolerant storage of messages in transit and no data loss in the event of hardware failure; utilisation of a rich ecosystem of existing plugins (e.g. database connection, replication, alerting)."*

Development work between May 2024 and October 2025 has delivered the core elements within the proposal text above. A prototype system, based around an Apache Kafka message broker containerised within Kubernetes, was deployed on the RRS *Sir David Attenborough* (SDA) in October 2024 and ran in parallel with the primary logging system throughout the 2024/25 Antarctic season. During this time, the prototype received a significant subset of the existing real-time sensor messages, writing them to disk and database, as well as mirroring these messages to a replica shore-side system in Cambridge. Following this successful trial, the prototype was fully incorporated within the BAS IT infrastructure onboard and took over as the primary logging system in early October, ahead of the 2025/26 Antarctic season.

The chosen architecture consists of open-source components and has greatly reduced the amount of in-house code that was present within the previous system. While connecting and configuring these components was not always plain sailing, and learning curves were often steep, most of the benefits outlined in the proposal have been realised.

This report includes: the background and history of data acquisition onboard NERC vessels, the previous acquisition system (RVDAS) and its limitations, the architecture of the new system (DAMP) and lessons learned in its development, and a list of next steps after the Amplify project concludes.

Contents

Background

There has been a long history of digital data acquisition systems onboard NERC research vessels, primarily focussing on permanently fitted underway sensors that continuously output data (e.g. wind velocity from anemometers). At their heart, these systems listen or poll for sensor data, accurately timestamp these data on receipt, and then write them securely to disk for onward reuse. Acquisition systems can have myriad additional functions (metadata, events, monitoring, reformatting, QA/QC processing, visualisation etc.) but they are all built around this core 'Receive-Timestamp-Store' functionality.

The 'ABC' system was developed by the Southampton Oceanography Centre's Research Vessel Services (RVS) group in the 1980's and was used onboard NOC and BAS vessels until the late 2000's. It had three acquisition levels (A, B and C) corresponding to distinct levels of data processing, from raw through to structured calibrated data. From the mid 2000's, IFREMER's TECHSAS[1] replaced the RVS system as the primary acquisition system on NOC vessels while BAS moved to NOAA's SCS[2] software. Both systems have the advantage of being developed and maintained by external groups and were/are provided free of charge. However, their development is focussed on national research fleet priorities (TECHSAS -> France, SCS -> United States) and they are closed-source software making modifications difficult for third parties. The use of different acquisition systems across the NERC vessels was also a disadvantage for researchers who regularly sail on BAS and NOC vessels as they had to interact with the differing aspects of each system.

In 2017, staff within the National Marine Facilities (NMF) group at NOC undertook a gap analysis to identify areas of improvement in their underway data acquisition systems. This resulted in an in-house development of a prototype Research Vessel Data Acquisition System (RVDAS) that was used in a secondary capacity onboard the RRS *Discovery* and RRS *James Cook*. At around the same time, BAS were designing IT and data systems for the new UK polar research vessel, RRS *Sir David Attenborough* (SDA). A requirement gathering exercise (aided by a community workshop in Nov. 2018) was followed by an evaluation of current marine data acquisition software from around the world. The evaluation resulted in the prototype RVDAS software being used on the SDA from the outset and the RVDAS project became a BAS/NOC co-development.

Initially (2019-2021) BAS acted as an intelligent customer providing requirements and use cases while NOC led on the software development. However, SDA science commissioning from 2021 onwards and the recruitment of an internally funded temporary software developer in late 2022 meant that BAS have taken a more active development role in the last few years.

---

[1] https://www.flotteoceanographique.fr/en/Facilities/Shipboard-software/Gestion-de-missions-et-des-donnees/TECHSAS

[2] https://scsshore.noaa.gov/

Research Vessel Data Acquisition System (RVDAS)

Overview

Figure 1.1 provides a graphical overview of the current version of RVDAS which is operational on all three NERC deep-water research vessels, noting that exact implementations vary slightly between BAS and NOC.

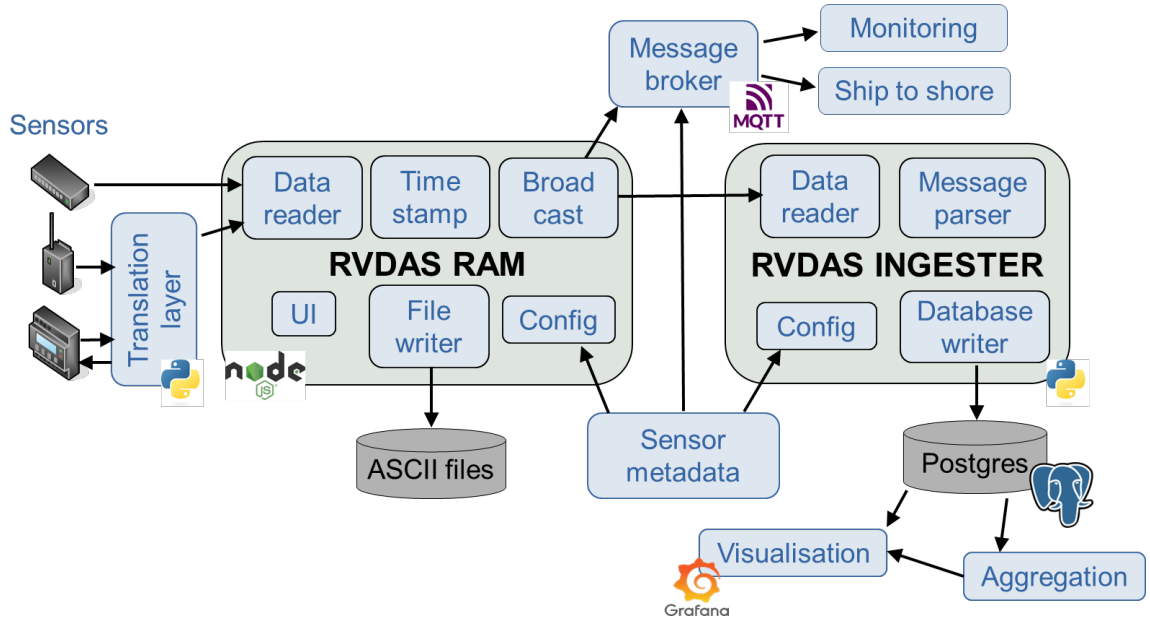

Figure 1.1 – A schematic of the modules and functional elements of the current (October 2025) RVDAS production version with arrows showing the flow of sensor information.

Component Architecture

The RVDAS Acquisition Module (RAM) was developed with a narrowly defined scope of listening for NMEA[3] sentences over the User Datagram Protocol (UDP). RAM timestamps the incoming messages, rebroadcasts them over the local network (also UDP) and writes them to disk, usually as per-sensor daily ascii files. RAM is written in Node.js and has a web-based interface for monitoring and configuration, the latter augmented with sensor metadata held within JSON files.

A small number of network-ready sensors provide NMEA sentences directly to RAM, but the majority must route through a mixture of hardware and software to be translated into a form that RAM can receive. This is referred to as the 'Translation layer' in Figure 1.1 and includes serial-to-ethernet hardware units and Python code to interact with sensor protocols not addressed by RAM (TCP listening, Modbus polling, MQTT subscriptions, SNMP listening etc.) and messages that don't to conform to the NMEA-standard syntax. This translation layer is more extensive on the SDA, reflecting the fact that the vessel is newer and has a greater number of sensors providing data in a wider variety of ways.

[3] https://gpsd.gitlab.io/gpsd/NMEA.html

The Ingester module, written in Python, listens for the messages broadcast by RAM and parses them based on sensor metadata configuration values. These parsed messages are then written to PostgreSQL database tables for onward visualisation and querying. TimescaleDB[4], a dedicated time-series PostgreSQL extension, performs real-time aggregations allowing for efficient onward querying and display.

The RAM module broadcasts to an onboard MQTT message broker (currently only on the Discovery and James Cook) used for real-time monitoring and for transfer to a shore-side message broker. This ship-to-shore pipeline has been utilised elsewhere within the Amplify project (Task 1.2) to send real-time temperature sensor data and metadata to a shore-side FROST server providing an onward OGC SensorThings API connection (i.e. see https://linkedsystems.uk/sensorthings/v1.1/Sensors)

Before outlining the limitations in the current system and how this experiment aims to improve matters, it is worth noting that RVDAS has worked remarkably well over the last few years. Functionally, it has addressed all the initial must-have requirements and it has proved robust in operational use. For example, RVDAS on the SDA receives ~500 messages per second from ~70 different sensors and the system has achieved 99.8% uptime over the last three years of continuous running.

Limitations

The RVDAS system has the following limitations:

- **Bespoke code** - Almost all the RVDAS components have been created from scratch using in-house software code (Python or NodeJS). This code has grown organically to solve specific issues and is becoming time-consuming to maintain.
- **Lack of flexibility** - The design of the existing system makes it difficult to add new features (e.g. the ship-to-shore message broker) and when these functions are bolted on, they increase the maintenance burden further. New sensors cannot be added, and existing sensors cannot be modified without restarting the RAM or Ingester modules, thus causing temporary data loss across all sensors. This is because processes are not sufficiently parallelised or independent from each other.
- **Limited resilience** – While RVDAS has proved remarkably reliable in operation, this is more through luck than design. For example, a problem that causes the RAM module to crash would stop data ingestion for all sensors with no message buffering occurring and no automatic procedures in place to back off and restart.
- **Duplicated functions –** There are several generic functions that are repeated (using differing code) across the various modules. For example, there is generic UDP listening functionality in the translation layer, as well as both the RAM and

[4] https://www.timescale.com/

Ingester modules. This duplication adds to the technical debt and means messages are spending more time transiting the network than is necessary.

Data Acquisition and Metadata Platform (DAMP)

Overview

To address limitations with RVDAS, design work for a revised architecture has been ongoing since at least early 2023. Initial work looking at component orchestration by Juan Ward at NOC, was taken forward by Mike Crosier in late 2023 and formed the basis of the proposed experiment within the original Amplify proposal in May 2024.

As described in the summary, a prototype revision of RVDAS, termed the Data Acquisition and Metadata Platform (DAMP) was installed on the SDA in October 2024.

Figure 2.1 provides a graphical overview of DAMP as it exists on the SDA at the time of writing.

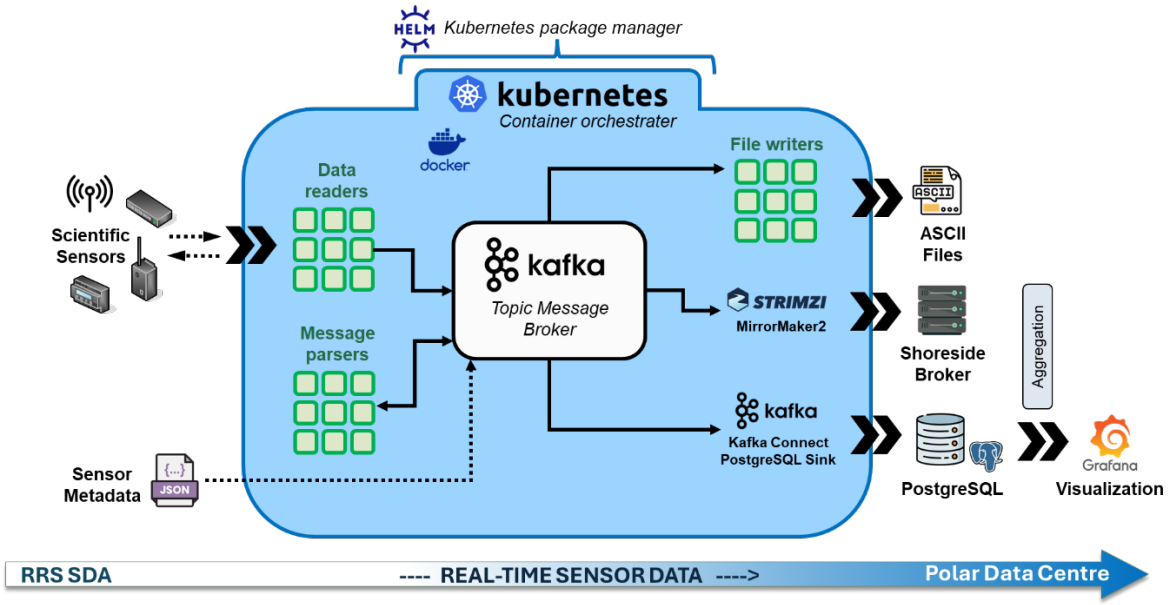

Figure 2.1 – A schematic of the components of the current (October 2025) DAMP production version with arrows showing the flow of sensor information.

DAMP is implemented as a Helm chart for Kubernetes (specifically K3s[5]), with inter-component communication built around an Apache Kafka[6] message broker.

Component Architecture

DAMP consists of containerised components, each of which performs a single function. They operate independently of each other, communicating via Kafka topics. A particular

[5] https://k3s.io/
[6] https://kafka.apache.org/

DAMP installation may have only a subset of components enabled, depending on the required functionality.

Readers

Readers are responsible for acquiring messages from a sensor and inserting them into a raw topic in Kafka, without modifying the message in any way. There is one reader per sensor and one raw topic per sensor. These readers may be of different types, depending on how the sensor communicates: for example, one type of reader receives UDP messages; another type communicates with Modbus devices. Kafka topic metadata includes the timestamp of each message received.

Raw file writers

Raw file writers consume messages from a Kafka raw topic and write the message content plus received timestamp to a file on the local file system. There is one raw file writer per sensor (per raw topic). DAMP is configured to create one ascii file per sensor per day.

Parsers

Parsers consume messages from a Kafka raw topic and parse them into individual fields, according to schema information contained within the sensor metadata topic. Parsed values are inserted into Kafka parsed topics. Any message which cannot be parsed (e.g. because it is corrupt) is inserted into a single parsing failures topic, along with error messages, for troubleshooting purposes. There is one parser per sensor (per raw topic). A parser may write to more than one parsed topic; for example, in the case where a single sensor emits multiple sentences per message. Additionally, multiple parsers (for different sensors) may write to the same parsed topic, if the sensors emit messages of the same type / share sensor metadata; for example, multiple instances of the same instrument placed in different locations.

Parsers monitor the sensor metadata topic and reload themselves whenever updated metadata for their sensor is received.

The actual parsing logic is implemented via a system of plug-ins, each of which handles a specific message format (e.g. NMEA, CSV). The choice of which plug-in to use is a property of the sensor metadata.

Currently, the parsers are also responsible for creating (but not writing to) database tables. This is an interim measure and will be moved to a more appropriate component in the future.

Database writer (JDBC sink)

The database writer consumes messages from Kafka parsed topics and inserts them into a database. There is a one-to-one correspondence between parsed topics and database tables. The database writer runs under Kafka Connect and handles all sensors concurrently.

Sensor metadata loader

The sensor metadata loader allows populating the Kafka sensor metadata topic based on config sent via Helm. It runs as a Helm post-install hook. This is a transitional measure and will eventually be replaced by a more direct means of pushing sensor metadata into DAMP.

Ship-to-shore message replication

Kafka MirrorMaker2 is used to replicate the raw and sensor metadata topics from the onboard Kafka cluster back to shore. A shore-side DAMP installation then parses the messages and writes them to file and database, producing an exact replica of the vessel installation in near real time – testing shows that the delay is less than a second for most sensors, assuming the satellite link is stable.

Visualisation

Real-time data visualisation is still undertaken by Grafana using PostgreSQL plus TimescaleDB as the primary data source.

Transition from RVDAS to DAMP

While DAMP appears architecturally very different from RVDAS, it is actually an evolutionary step along a longer-term development path. For example, much of the bespoke Python code that made up the sensor-to-RAM translation layer (see Figure 1.1) still exists within DAMP. However, the translation modules are now containerised within per-sensor readers and the previous translator management functions have been replaced by in-built Kubernetes functionality. A future enhancement could replace the remaining bespoke translator code with open-source equivalents (see next steps section).

Care was taken to ensure the transition would be mostly invisible to end users, and it was important to have a prototype DAMP installation running (in parallel with RVDAS) for a significant period before it became the primary logging system, to test for reliability.

Addressing RVDAS limitations

DAMP has addressed many of the limitation described in the RVDAS section above. However, none of these limitations have been completely removed, and work will continue to make a more resilient, flexible, scalable system in the future.

Bespoke code

DAMP contains many more lines of code than RVDAS but the majority of this is boilerplate configuration associated with the main architectural components (Kubernetes/Helm, Kafka, Docker etc.). This appears more complex and impenetrable than the bespoke RVDAS components and code that it replaces, but the big difference is that the boilerplate is well documented online and any issues that arise are almost certainly discussed and solved by countless other developers in online fora.

Lack of flexibility

The use of containers and their orchestration through Kubernetes has allowed new sensors to be added to DAMP with no effect on existing data flows. The same is true for modifying the setup of existing sensors. The use of a Kafka message broker at the core of DAMP is already advantageous for the exiting functions (parsing, file writing, database writing, ship-to-shore mirroring) but it provides the foundational platform for a host of other components to be added in the future (stream processing, additional writers, monitoring etc.).

Limited resilience

Much of the resilience of DAMP is built-in to the chosen architectural components by default. For example, Kubernetes provides self-healing functionality whereby it can redeploy components (e.g. readers, parsers, writers etc.) if they should fail, to bring the system back to a given desired state. The DAMP Kafka component is running as a three-node cluster within Kubernetes, providing the high availability and fault tolerance needed for a system that needs to operate all the time. Containerised components like the sensor readers are independent from one another meaning that an unrecoverable failure in one reader will not impact other data flows.

Duplicated functions

DAMP has eliminated the functional duplication that existed within the RVDAS RAM and Ingester modules, mostly replacing it with inter-component communication via Kafka. In other areas, 'good' duplication has been introduced within DAMP, using containerisation. This has allowed DAMP components to operate independently from each other, improving resilience.

Other DAMP enhancements

During the development of the DAMP prototype, and transition to a production version, we made several additional enhancements that have streamlined the management and deployment of DAMP services including:

GitLab CI/CD improvements

RVDAS code is managed within Git repositories (using GitLab) and the same is true for DAMP. Prior to the Amplify project we had discussed the use of CI/CD (Continuous Integration/Delivery/Deployment) pipelines, and these ideas were taken forward during the development of DAMP. Deployment of DAMP is now almost entirely undertaken via Gitlab CI/CD pipelines.

Reduction in server-level configuration

At least onboard the SDA, RVDAS requires server-level automation code to deploy and run the various components (via Puppet[7]). Deployment automation is now undertaken

[7] https://github.com/puppetlabs/puppet

within DAMP or through DAMP CI/CD pipelines and the Puppet code is now much simpler and focusses on server-level aspects like firewalls settings.

Highly configurable DAMP installations

DAMP has a hierarchy of configuration files that allow deployment of the software in different settings. In order, this allows deployment in different organisational environments (e.g., BAS vs NOC), on different platforms (e.g., SDA vs Discovery), and to different servers (e.g., production vs testing) with each level inheriting, and optionally overriding, the one above.

User engagement

This section describes user engagement activities that prioritised the underway data logging system (versus other data management activities onboard) and informed the eventual design of DAMP. Some of these activities were funded through the Amplify project while others occurred as part of other projects or simply as business as usual.

Vessel data systems questionnaire and workshop (BAS, 2018) – While this occurred some time ago, the results of both the questionnaire and workshop have helped to inform our development strategies ever since. One firm conclusion from this engagement was that the underway logging system was the highest priority vessel data system component and should be allocated significant time and resource to achieve a set of user requirements. These requirements were also articulated in the engagement exercise and formed the basis of the early RVDAS development work.

Marine cruise participants (onboard NERC research vessels, ongoing): Marine cruise legs typically last between four and eight weeks and often host >40 researchers as well as technical support staff and crew. This large number of ever-changing, highly focussed end users is an ideal population to get real-time suggestions and feedback on data systems. This has ensured that the development of DAMP has continuously focussed on practical outcomes for end users rather than being an academic exercise in stitching together software components.

Vessel data user workshop (NOC, Sept 2024) – A one day workshop for end users was held in Southampton as part of the Amplify project. The day included a show-and-tell on DAMP developments up to that point and then focussed on discussion topics including ship-to-shore data transfer, real-time data processing levels, and sensor metadata. User requirements that came out of the break-out groups were used to inform onward development where appropriate.

Amplify workshop presentation (Cranfield, October 2025) – A poster describing the DAMP exemplar was presented by Claudette Lopez at the Amplify workshop that formed part of the annual NERC Digital Gathering event.

NERC EDS webinars (online, throughout project) – Perhaps not end user engagement, but a series of EDS webinars throughout 2024 and 2025 have highlighted sensor data

developments across the other NERC centres. While thematic scopes are often very different it is reassuring (but probably not surprising) that many of the architectural concepts within DAMP are present in other projects.

Lessons learned

Open-source software patchworking

Mike Crosier (DAMP's architect) was recruited to BAS as a software developer, but he did not describe what he did during the Amplify period as software development. Instead, most of his activities involved stitching together open-source software components with just enough custom functionality to make the result a vessel data acquisition system versus some other type of application. We are not entirely sure what the job title should have been, but it is a useful finding when putting together future job descriptions for similar work.

BAS IT engagement was key

The initial DAMP prototype was developed and deployed on virtual machines that were managed by the Polar Data Centre. However, it became clear early on that the long-term viability of DAMP was dependent on whether the BAS IT team would adopt and support some of DAMP's architectural components. After the successful deployment of the DAMP prototype onboard the SDA, a commitment was made to integrate DAMP within the central BAS IT infrastructure, and this was honoured during the Amplify extension period.

Kubernetes networking is hard work

Kubernetes brings a lot of benefits, but some elements have a steep learning curve with its internal networking being particularly difficult to get right. For example, the production deployment of DAMP onboard the SDA has a (hopefully) temporary script to relay incoming UDP messages originating from the VLAN's broadcast address. The script sits outside of Kubernetes and relays the messages to the localhost, without which the DAMP readers pods would fail to pick them up. Despite lots of effort, we have not found the magic Kubernetes/Docker network configuration that will allow us to get rid of this redundant relaying step.

Next Steps

Following on from the Amplify-funded DAMP exemplar presented in this report, the development tasks below are the natural follow-on activities.

Kubernetes cluster

To reduce complexity, DAMP is deployed within a single node Kubernetes installation. However, throughout the Amplify period, staff at NOC have been investigating how to operationalise a multi-node Kubernetes installation. This would improve resilience even further and allow for periodic maintenance of Kubernetes itself and the host virtual machines, without any downtime.

Data translation layer

Bespoke Python code is still used to listen/poll sensors (within each DAMP reader) and while this has some advantages (there is a Python module to suit every occasion), there is a lot of variability in the code and solutions found are often narrow in scope and don't scale well. Engagement with other NERC EDS partners through the Amplify project has shown that Apache NiFi[8] is good open-source candidate to replace most, if not all, of the existing data translators with a huge library of additional data pipeline functionality to satisfy as-yet unencountered sensor workflows. In-house testing of NiFi (as an individual software component) has confirmed its potential, the outstanding question is how it can be best integrated within DAMP.

Real-time stream processing

DAMP's architecture now makes it much easier to add real-time data processing capability. A processor could subscribe to raw Kafka topics, perform a range of processing functions, and then publish the results back to a processed Kafka topic for onward use. Examples include data quality flagging, merged/aggregated products like true wind velocity, and anomaly detection. This type of functionality is provided by a range of open-source software such as Flink[9], Spark[10], and Kafka Streams[11] (all Apache projects). Apache NiFi also has advanced stream processing capability.

Open-sourcing DAMP

The primary aim of this work was to develop a real-time data logging exemplar that might inform the design decisions of similar systems across the EDS. However, it is possible that there might be broader interest in the software patchworking within DAMP, and this could be advertised by making the source code publicly available via GitHub (or similar). Work would be required to ensure the current codebase was cleared of any security issues and reviewed for areas that are too specific or not documented well enough.

Acknowledgements

The authors would like to thank Petra ten Hoopen, Alice Frémand and Jeremy Robst from the British Antarctic Survey for feedback throughout the project. Also, thanks to the wider team of RVDAS/DAMP co-developers at the National Oceanography Centre (Zoltan Nemeth, Emmy McGarry, Basem Drawil, Joshua Pedder). This work has benefited from ongoing feedback from staff at the British Oceanographic Data Centre as well as hundreds of science cruise participants onboard the RRS *Sir David Attenborough*, RRS *James Cook*, and RRS *Discovery*.

[8] https://nifi.apache.org/
[9] https://flink.apache.org/
[10] https://spark.apache.org/
[11] https://kafka.apache.org/documentation/streams/