Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 126 (2018) 1656–1665

Procedia Computer Science

www.elsevier.com/locate/procedia

22nd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Near Real-time Autonomous Quality Control for Streaming Environmental Sensor Data

Connor Scully-Allison, Vinh Le, Eric Fritzinger, Scotty Strachan, Frederick C. Harris, Jr., Sergiu M. Dascalu*

Department of Computer Science and Engineering, University of Nevada, Reno, Reno-89557, United States of America

Abstract

In this paper, we present a novel and accessible approach to time-series data validation: the Near-Real Time Autonomous Quality Control (NRAQC) system. The design, implementation, and impacts of this software are explored in detail within this paper. This software system, created in close conference with environmental scientists, leverages microservice design patterns employed for high volume web applications to develop a contemporary solution to the problem of data quality control with streaming sensor data. Through a comparative analysis between NRAQC and the GCE Toolbox, we argue that the web based deployment of QC software enhances accessibility to crucial tools required to make a robust and useful data product from raw measurements. Additionally, a key innovation of the NRAQC platform is its positive impact on modern data management practices and quality data dissemination.

Keywords: Software Engineering; Data Management; Microservices; Quality Control; Web-based Systems; Information Quality

1. Introduction

In the field of in-situ environmental sensing, invalid data is a very serious concern that affects the overall quality of a data set and pollutes the integrity of any derived data product. Invalid data occurs when hardware deployment conditions change in a way that is not anticipated by study design, intended use, or researcher assumption. A hardware failure is often the result of this change and produces gaps in data or verifiably incorrect values. Invalidity, in the context of time-series environmental data can be organized into five major categories: missing values, out of bounds, repeated values, spatially inconsistent values, and internally inconsistent values.

* Sergiu M. Dascalu. Tel.: +1-775-784-4613, Fax: +1-775-784-1877
E-mail address: dascalus@cse.unr.edu

1877-0509
10.1016/j.procs.2018.08.139

Missing values represent a period of time in which no measurements were logged. The out of bounds category details a measurement that exceeds maximum or minimum bounds of what is considered a reasonable measurement. Repeated values occur when the exact same value is logged in direct succession for an extended period of time. Spatially inconsistent data describes co-located physical data loggers recording vastly different trends in data over the same period of time. Finally, internally inconsistent data describes logical inconsistencies within a measured data stream or between two related datastreams.

To address these problems we developed the Near Real-time Autonomous Quality Control (NRAQC) (pronounced "na-rock") System, for the Nevada Research Data Center (NRDC). [12] NRAQC tests incoming data points logged autonomously at a research site like the one pictured in Fig. 1 to see if they meet the criteria of an invalid measurement. The system, with aid of user specified configurations, flags all invalid measurements with metadata that specifies the nature of the invalidity. The service provided by NRAQC is necessary for the production and distribution of a quality data product.

The remainder of this paper is structured as follows: Section 2 introduces a basic background of the project and some related works, Section 3 goes into the specifications of the software, Section 4 discusses the overall design of the software, Section 5 contains the prototype details, Section 6 includes a discussion about the differences between quality control software, and Section 7 wraps up the paper with the conclusion and future work.

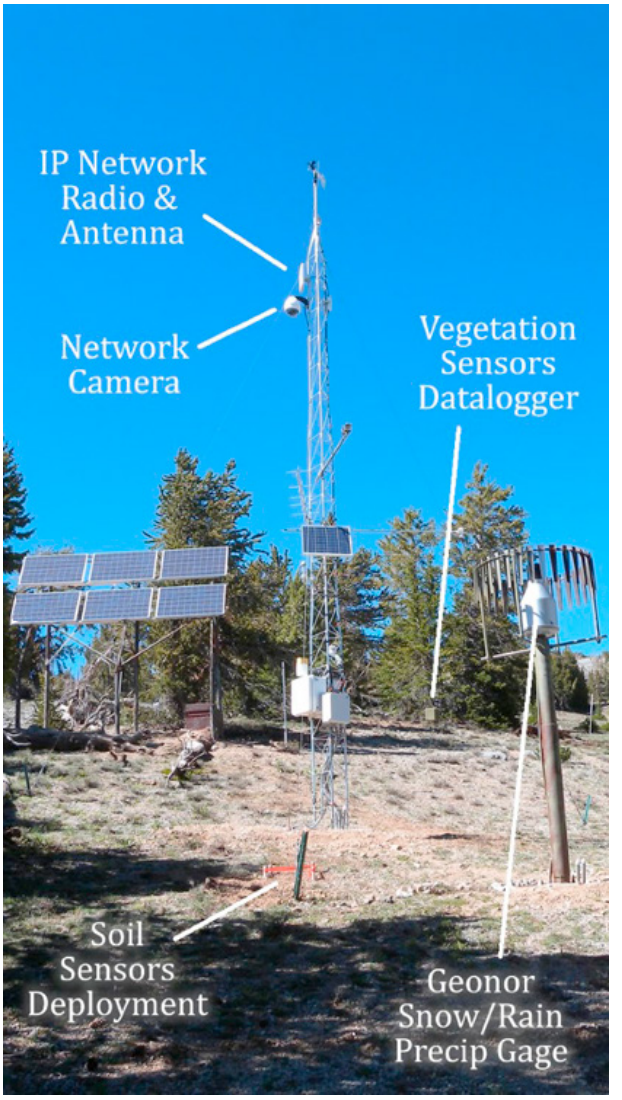

Fig. 1. An in-situ environmental monitoring station in the Snake Range. Stations like this are hard to monitor regularly and can produce errors in data when damage is incurred to one of the many delicate sensors around the site.

2. Background & Related Works

The problem of Quality Control (QC) in the realm of environmental sensor data has been explored extensively in scientific literature. In 1970, techniques to formalize Quality Control have been detailed as a necessary part of any project which produces large volumes of autonomously collected data. [5] In more recent years however, as the autonomous environmental research sites have become cheaper to install, easier to set up, and generally more ubiquitous, there has been a strong surge of research exploring the methodologies and importance of Quality Control on high volume streaming data.

Many academic and industry oriented white papers published in the last 15 years explore the importance of Quality Control as it applies to domain specific data. Meteorological data, environmental data, metadata studies: together these and many more articles paint a clear picture indicating that Quality Control is no small problem to the scientific community working with time series sensor data [6, 9, 11, 17]. Outside of these general considerations however some key literature has examined major ideas informing modern automated quality control in great detail.

At the forefront of seminal works on this issue – and the one most informative to the software produced for this paper – is the article by Campbell et. al. [2]. This paper presents a clear and critical overview of Quality Assurance(QA) and Quality Control in the context of streaming environmental sensor data. As environmental data collection becomes

increasingly automated, a need to build automated checks on these processes arises. As the paper indicates, Quality Assurance is "*process* oriented" and Quality Control is "*product* oriented", establishing a clear delineation between the two often muddled concepts. From these basic definitions, the authors expand on automated QA/QC providing a survey of state of the art in this domain: best practices, implementation suggestions, and examples of prominent extant software which successfully automates these processes.

Significant work has been done by several organizations to create robust and practical QC software. Principal among these is a software suite called the GCE Toolbox by Georgia Coastal Ecosystems LTER. This Matlab based toolbox [10] represents the gold standard of QA/QC software for environmental sensor data. Users of this software can perform automated quality control manually or with rules-based checks, visualize data and import/export data in various inter-operable and standardized formats. Unfortunately, an advanced knowledge of Matlab–and access to the costly Matlab compiler and IDE–is required to use these tools. This greatly restricts accessibility to quality QC, especially to smaller research organizations with fewer resources. NRAQC addresses this drawback with its simplicity of deployment and multiple robust interfaces for configuration.

Outside of the GCE toolbox, several other software packages attempt to address contemporary QC needs in various scientific domains. Chief among these, ArcGIS provides functionality for the automated QC of geospatial data [4]. Additionally, Campbell Scientific has support software associated with their data loggers called Loggernet. This proprietary software provides basic Quality Control functionality on streaming data [14]. NRAQC is not actively iterating on any functionality provided by ArcGIS as it only handles Time-Series data. Additionally, since NRAQC queries a database, file, or API for logged data, it is agnostic about hardware setups unlike Loggernet which interfaces directly with the physical data loggers, causing compatibility issues.

In addition to the software development efforts, many organizations are working to standardize the metadata flags which indicate specifically how a measurement was deemed invalid. Chief among these organizations are ESIP and QARTOD, with the former cataloging and supporting present efforts towards standardization and the latter putting forth a strong proposal for flagging standards in oceanographic sensor data that can easily be applied more widely [3, 8]. Standardization is crucial for data inter-operability. Understanding these standardization efforts is equally necessary to create QC software that scientific organizations will use.

The approach of utilizing a microservice architecture in a data intensive system is a recent development in distributed architectures. Even more, utilizing a microservice architecture to implement any form of quality control system is currently not a popular topic of research. It is because of these reasons that there is a severe deficit of academic papers within the last five years pertaining to both quality control and microservice architectures. With this said, this paper provides both a solution to the problem of uncoordinated quality control, and a contribution to the research regarding usages of service-oriented architectures in quality control.

Finally, the primary drive and motivation to produce this software follows from the development of a Quality Assurance application that manages metadata for the NRDC. [15] Quality Control solutions represent a natural extension of metadata management practices. Data quality flags are essentially granular metadata bundles applied to individual measurements, which denote any possible errors with the data, the means by which it was flagged and possible corrections made to the data. Together QA and QC represent two pillars supporting robust and usable data in the 21st century.

3. Software Specification

Software requirements were developed from a formal elicitation process where two stakeholders were interviewed to determine the critical functionalities of the system. The first stakeholder interviewed, Scotty Strachan, is a domain scientist specializing in in-situ environmental sensing and data management. Dr. Strachan was solicited to express the needs of the intended audience for the NRAQC. His experience with existing QC software greatly aided the development of functional requirements for this software package. The second stakeholder, Eric Fritizngер, a data management expert and software engineer working as lead developer for the NRDC, was elicited because of his expert knowledge on distributed systems and scientific software development. His input helped guide development towards a microservice-based architecture so that the NRAQC system could be better integrated with the existing microservices currently handling the collection of measurements and non-QC metadata in the NRDC. In addition, he suggested that the development of a web-based system that manages significant amounts of feedback from autonomous tests raised

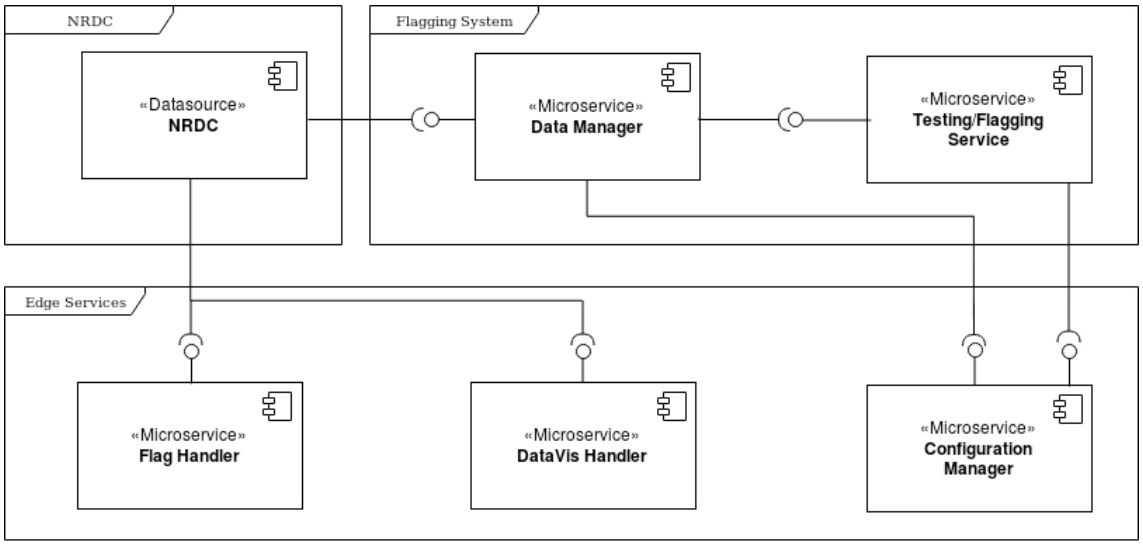

Fig. 2. Component diagram showing the high level architecture of the NRAQC system.

concerns of failure when using a monolithic system. The current compatibility with an existing system, and the risk of significant architectural failure, were the largest factors that drove the development of microservices as NRAQC's primary backend.

The requirements determined by the elicitation process were split between *functional requirements*, which describe the overall technical functionality of the system, and *non-functional requirements*, which outline constraints on the system. The functional requirements of NRAQC were broken down into two priority levels. The base level requirements indicate the critical functions that are vital to the system operability, while the second level requirements represent adjunct functionalities that are not necessary for operability.

From stakeholder elicitation it was determined that the NRAQC system depends on ten fundamental operating requirements. The first requirement is that the system utilizes a suite of microservices comprising a greater Microservice-based Architecture. The second requirement is that the system monitors a provided data source for new entries at a regular interval. The third requirement relates to the second in that the system will ingest a data point upon observation of a new measurement. The fourth requirement furthers this pipeline by having the system relay the sensed data point through a suite of automated tests. The fifth and sixth requirements outline the flagging and saving of QC related metadata based on the results of the fourth. The seventh, eighth, and ninth requirements showcase the system's ability to list, breakdown, and visualize flagged data through an intuitive user interface. The tenth, and final, base level requirement describes the system's ability to record user-specified modifications to data and streaming management.

For the second level, there are a total of five requirements that the NRAQC system provides outside the scope of the vital functionalities. For the first, the system allows data stewards and technicians affiliated with the NRDC the ability to create a "manual flag" on invalid data based off their own expertise. The second requirement dictates that the system must then save these manual flags back to the data source with metadata distinct from autonomously flagged data. The third requirement details the system's ability to prioritize the monitoring of specific datastreams. This monitoring must be configurable by a technician or data steward and relayed via email and/or the user interface. The fourth requirement makes it so that the system allows technician and data stewards the ability to configure a specific data source without needing a technical background. The fifth requirement of the second level enables the establishment of administrative hierarchy allowing designated administrators control over select systems of NRAQC while managing the content access of other users.

The NRAQC system currently operates under seven non-functional requirements. The first requirement dictates that the NRAQC system is built as a web application. The second requirement describes the backend microservices being written in Python with the Flask library as its main framework [13]. The third requirement indicates that user interface must be written with HTML, CSS, and JavaScript, with Angular4 as the primary framework [7]. The fourth

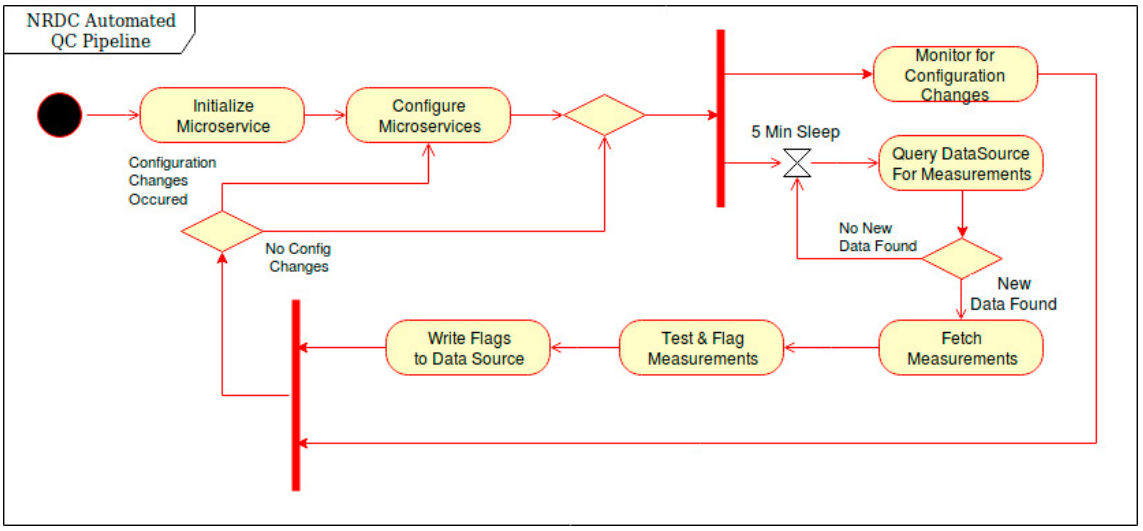

Fig. 3. Activity chart detailing the main workflow of the NRAQC system.

requirement ensures that the data model followed by the NRAQC system coincides with the model followed by the greater NRDC system. The fifth requirement describes the use of containerization provided by Docker. The sixth requirements relates to the fifth in that the system was developed inside the Linux operating system. The seventh requirement necessitates that the flagging subsystem can detect the availability of multi-core processors within its backend environment.

4. Software Design

Being a system that interacts with and permanently alters a field associated with live and valuable streaming data, the software design of NRAQC was developed to ensure a robust system and demonstrate very clearly how every component works in great detail. In this section the most important elements of this system's design and architecture are detailed. Beginning with the high level design of the overall system and continuing into explanations of the pipeline architecture, which describes the core flagging system, this section shows in detail everything required to reproduce a similar system.

4.1. High Level Design

The high level design of this system is component based, as shown in Figure 2. A component based architecture was chosen to ensure loose coupling, clear encapsulation, and strong inter-operability. In the place of traditional components – often defined as classes with abstract interfaces – this software utilized microservices as components with an HTTP API providing primary interfaces to associated services. To provide essential functionality as defined by the base level functional requirements, five components were developed: a Data Manager, a Testing/Flagging Service, a Flag Handler, a DataVis Handler, and a Configuration Manager.

The configuration service interfaces with the web client – and user – by receiving user configuration requests and sending down stored user configuration info. Additionally, the Configuration Manager implements interfaces to distribute configuration objects to the Data Manager component and the Testing/Flagging Service. It can also notify the two services of a change to the configuration and request a temporary cease of the automated flagging cycle while re-configuration occurs.

The Data Source Service provides all functionality of retrieving, formatting, disseminating and writing measurement data, flag data and related metadata. This service provides a flexible interface to the data source that can be reconfigured to accommodate an API data source or a flat file system. It requires the implementation of two other interfaces: one which allows it to transfer data between itself and the Testing/Flagging Service component and another which allows it to receive data from the Configuration Manager component.

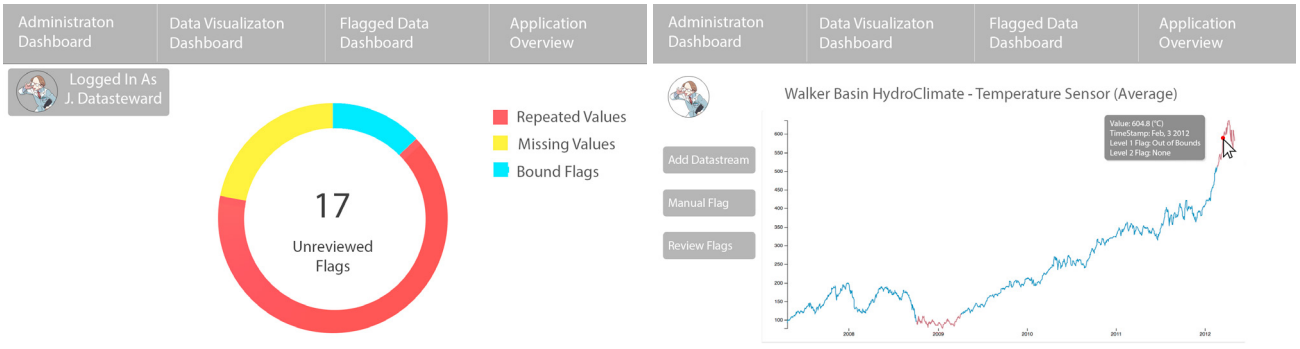

Fig. 4. A mock up showing the design of the flag dashboard (left) and the data visualization dashboard (right).

The Testing/Flagging Service provides the core processing functionality of NRAQC by testing and flagging data. It runs autonomously and requires no direct interaction from the user at any time. It provides one interface allowing the Data Manager to send measurements to it for testing and retrieve flagged measurements from it for writing to the data source. It requires the implementation of one interface from the Configuration Manager to allow the upload and modification of tests from the client.

Outside of the above three services which provide functionality related specifically conducive to the testing and flagging of invalid data, this software requires two components dedicated to conveying and formatting data from the data source to the remote UI. Unlike the other components in this system, these microservices provide more traditional CRUD functionality to the user via GET and POST requests.

The flag handler, as an edge service, provides an interface between the client and server. Its primary role is to handle JavaScript code which queries for flags. It will filter, format and bundle flags so that they may be quickly rendered upon being downloaded by the client. Formally, it implements an interface to connect to the data source.

The DataVis Handler implements an interface to connect with the data source component and retrieve desired information. This microservice handles all requests from the data visualization component of the UI and performs necessary formatting and organization to ensure that the Web page can focus on rendering data points instead of wasting power processing them. This service also provides functionality to classify flagged data points as distinct from normal data points so they will be rendered differently by the data visualization component.

4.2. Pipeline Activity

Transitioning from a static view of the component-based architecture into a process oriented view, the interaction of the three core functional components – Configuration Manager, Testing/Flagging Service and Data Manager – can be described by the activity chart shown in Figure 3. In this figure the main work flow is shown as a parallel pipeline occurring on a continuous loop every 5 minutes. After all three components are set into motion this loop should continue indefinitely with no need for human intervention until the QC software is decommissioned.

To explain this chart in greater detail, the functionality provided by the Data Manager and Testing components can be seen in the parallel flow beginning with the “5 Minute Sleep”. When new measurements are found in the database they are passed from the Data Manager to the Testing component where each measurement is run through each test and flagged when it fails. This entire testing process is contained in the“Test and Flag Measurements” activity. From there, flagged measurements are returned to the Data Manager to be written back to the data source.

This entire testing and flagging occurs in conjunction with an ambient monitoring of user configurations via the Configuration Manager. As the functionality detailed here occurs remote from the client web application where users modify configurations, there is no guarantee that a test or data source will not change mid-test. To account for this eventuality, changes will be handled in parallel with the testing process but not applied until the entirety of an active test batch is completed.

5. UI Design

The user interface for the NRDC QC application is a website containing multiple pages and components that communicate with the remote microservices detailed at length in other sections. The web application, following an initial login page, has three main top level pages which users can navigate in between: the Flagged Data Dashboard, the Data Visualization Dashboard, and the Administration Dashboard. Given that these are the primary interfaces users will interact with, the following exploration of the UI design will be broken up by those broad categories, excluding the Administration Dashboard due to space constraints.

Upon logging in, users will be greeted with the page detailed in Fig. 4. The first thing to note about this figure is the overall structure of the webpage, the navigation bar is horizontally oriented like folder tabs and there is a lot of white space. The tab like navigation bar enables a cleaner design as large visual entities like charts are more easily aligned in the center of the page. This ensures that data visualizations, like the flag related doughnut chart are given a place of importance in the center. Building upon this, the UI utilizes white space and color to draw the eye to critical information and mitigate any distraction when evaluating important data. Presently the navigation bar only has text but in actual implementation it will also contain informative icons for increased accessibility.

Via the link in the navigation bar or through the “Flagged Data Dashboard”, the users will see the page detailed in Fig 4. However, they will only see the visualized data shown here, if they came via the Flagged Data Dashboard. If they came from the navigation bar link they will need to select the “Add Datastream” button from the left toolbar and choose a datastream to be visualized before they see a line on the chart.

For additional information, a user can hover over the chart at any point and get a quick breakdown of the measurement rendered at that point. The small text box displays relevant info about this measurement like the value and the type of flag. Eventually, users will be able to click this data point to get an even more detailed breakdown of relevant flag related metadata, like information about the test that it failed or when it was flagged.

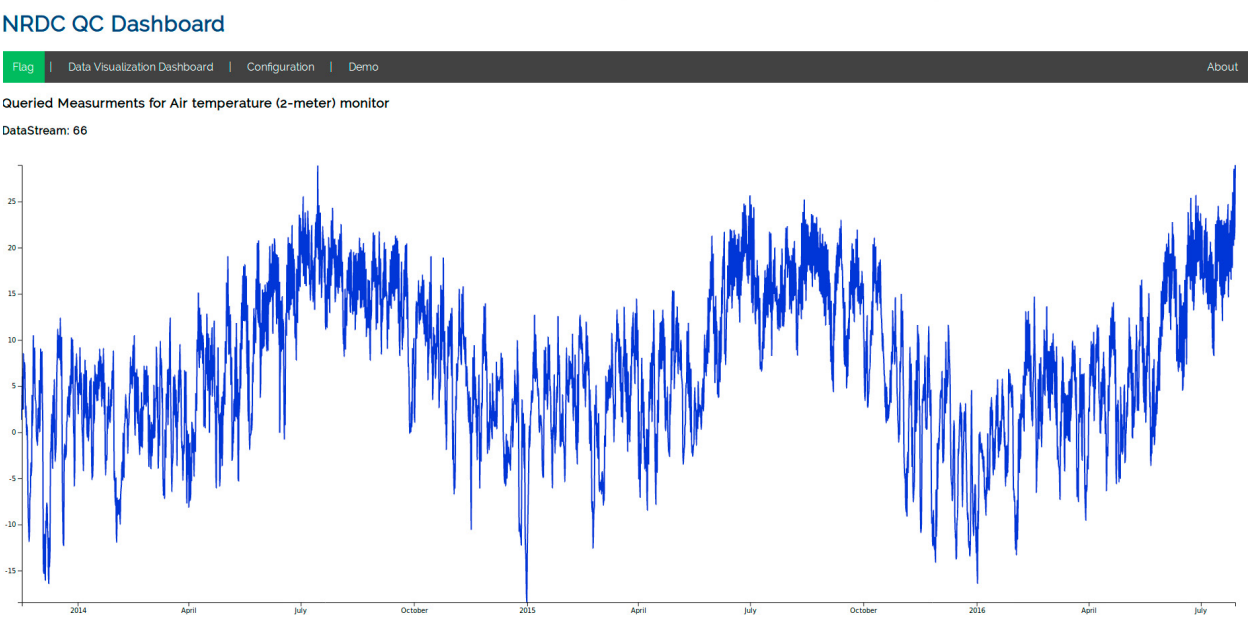

Fig. 5. The implemented prototype design of the data visualization dashboard. Although not graphically rendered, each of the data points in this graph are loaded with metadata indicating their associated flags.

6. Prototype Development

In order to prove the viability of this software a prototype was developed that implemented a subset of functionality as detailed in the Section 4. Specifically, on the server side, four microservices were implemented: the Data Manager, Testing/Flagging Service, Flag Handler, and DataVis Handler. On the client side two core webpages were implemented: the flag dashboard and the data-visualization dashboard. These specific components were chosen because they best represented the core workflow of the NRAQC system both from a user perspective and system perspective.

The aforementioned microservices were developed as a series of Flask services that called upon a core set of classes, implemented in Python, which provided the data retrieval and testing functionality. Using the methods in these classes, the Data Manager derived configuration information from a hard-coded XML file that allowed it to connect to the specified data source. In our case this data source was the NRDC database which contains thousands of datastreams that map billions of discrete measurements. For the purposes of this prototype, two data streams each containing approximately 140,000 measurements were used.

After setting up a functional writing and querying service, the Testing/Flagging service followed. Again receiving functionality from the same set of classes, this service ingested a second hard-coded configuration XML file containing testing data. Using the ingested testing data the Testing/Flagging service creates all tests dynamically as ”Test” objects and categorizes them by data stream so that they will always test the correct measurement set.

Upon successfully linking these two services together and verifying that they correctly retrieved, flagged and wrote back flagged data, development began on the client side and two edge services required to retrieve data from the data source. Development here occurred from a UI perspective and great efforts were made to mirror the initial UI designs. The results of this effort were mixed but generally promising. The doughnut chart was relatively simple to implement out-of-box, but the line chart failed to work with such a massive data set and had to be custom configured. Accordingly, to match the functionality desired from the design, this required more development than was within the scope of this prototype, so a line chart containing basic functionality was built using D3.js [1] and is visible in Fig. 5.

After selecting existing data visualization libraries to handle chart generation the Flag Handler and DataVis Handler microservices were created to serve JSON data formatted to fit the requirements of the front end. With all these components developed our prototype was set into motion and worked exactly as expected: ingesting, testing/flagging and writing measurements successfully. The few pages implemented on the front end also satisfactorily demonstrated that our intended workflow for this software was also possible.

6.1. Case Study

As a part of a case study performed to showcase the proof of concept, two datastreams logging air temperature data were chosen from dozens present within the Rockland Summit site of the Walker Basin Hydroclimate network. It must be noted that the NRAQC prototype can in fact operate on any of the hundreds of datastreams housed inside the greater NRDC system, but these two were chose to simplify the bounds of our case study. Of the two chosen datastreams, more than 140,000 measurements from each stream were cycled through the autonomous tests conducted by NRAQC.

Each measurement processed by the NRAQC system underwent three distinct tests: bounds checking tests, repeat value tests, and missing value tests. These tests are described in Section 1 and were drawn from the work of Campbell et. al. [2]. The parameters of each test were defined in an XML configuration file and adjustable without modifying any code in the classes or microservice which performed the tests. After testing the 280,000 backlogged data points, queried from and reinserted directly into the NRDC database, 178 repeat value flags were logged, 31 out-of-bounds flags were logged, and 131 missing value flags were logged. The QC flags were placed on each data value indicating the test which flagged it and the time it was flagged. For missing values, empty measurements were created and inserted into the primary data table providing well defined spaces for later data imputation.

7. Discussion

In many environmental sensing groups, the notion of Quality Control is not unfamiliar and the practice of using software to handle data sanitation is very common. However, groups tend to use third-party software such as GCE Toolbox alongside the MATLAB Computing Environment instead of creating their own. These software solutions

offer a wide array of tools to aid scientists with all sorts of management and control over their datastreams. Although these external software solutions provide significant functionality to suit the purposes of these environmental groups, they come with their own set of problems. Software such as GCE Toolbox tends to be rather rigid regarding usability. In order to use GCE Toolbox to its maximum potential, the user would be required to have a significant amount of technical knowledge. Additionally, software such as GCE Toolbox is intended to work on a very specialized computing environment, and therefore resists any form of system replication across networked workstations. Finally, most software packages like GCE Toolbox require a sizable fee in order to utilize their services. From a system-level view, this would create far too large of a barrier when it comes to accessibility, procurement, and future maintenance of the software.

The development of a system-level quality control web application addresses each of these problems. By shifting towards the development of a web application, this allows the members of the environmental group to access the services of this system from anywhere with an internet connection. This widens accessibility to this system and it eliminates the need for installation of external software packages on individual terminals.

We developed this software to be completely open sourced and free to the public. By having it open sourced, this handles the procurement problem as any environmental group may simply download the project through a public repository. Additionally, the NRAQC system focuses primarily on the non-technical usability of a quality control system. It allows users to adjust the configuration on virtually all aspects of their datastream through the dashboard without significant need for technical intervention. This intuitive design addresses the problem of future maintenance, as any scientist, technician, or data steward may operate this system without a technical background in computer science or software engineering.

As the fields of sciences, especially environmental sciences, continue to grow, the need for more complex software solutions will increase. The software that engineers build must be able to adapt for larger scales and address the needs of scientists without burdening them with impractical constraints. The NRAQC system provides the means to address the needs of growing environmental sensing projects with its reconfiguration of traditional data management practices.

This software's primary contribution and innovation comes from its principal design characteristic to be deployed as a core step in a deliberate automated data management workflow. Many common desktop QC solutions, like the GCE Toolbox and Excel, imply a post-ingestion quality control process on data. That is, quality control handled as an afterthought; a step which necessarily precedes analytics to ensure quality results. By decoupling the quality control process from the data collection/ingestion process, crucial quality control metadata is often maintained on local machines or left to languish on closed networks. Data sources, even those from prominent scientific institutions like USGS, will provide data sets without any granular metadata informing data users about what quality control processes each measurement has been verified against [16].

By injecting the Quality Control process automatically into a data-management workflow, metadata is tied fundamentally to its associated data, because they are logged nearly at the same time. The tight coupling of metadata and data forces new projects to accommodate granular metadata in their designs because it's being collected in volumes and velocities rivaling that of the data collected. This, in turn, leads to a greater accessibility of quality control metadata because it's intrinsically tied to the data being accessed and used.

This structure aids domain scientists, data users, and data managers in multiple ways. It gives data managers a clear workflow for managing and distributing quality control metadata. It makes it much easier for data users to get a transparent, quality data product with a easily traced provenance showing its pedigree and reassuring them of the validity of their calculations. Finally, domain scientists – who often fund and benefit from from large scale, autonomous data collection projects – are given a infrastructure which enables them to provide clear, understandable data to their colleagues. This, in turn, promotes better reproducibility and independent validation of published research.

8. Conclusion and Future Work

This paper presented an essential data sanitation solution for the NRDC based around the previous work done by other environmental sensing groups. We devised a method to autonomously validate each data stream by adapting existing microservice architectures and developing a suite of Python services to dynamically create tests based off of custom user configurations. This method allows technicians and data stewards with a non-technical background a greater amount of control over data managed by the NRDC system. This software provides significant benefits to the

NRDC by providing tools to verify the integrity of multiple high-volume datastreams with almost no regular intervention. Additionally, this software enables stakeholders to prepare a robust and accurate data product for dissemination to the greater environmental research community.

By creating the base NRAQC system, this opens up new avenues of development within the NRDC system and the greater environmental projects that it is affiliated with. The addition of machine learning could possibly yield results when applied to Quality Control. First, if we characterize Quality Control as a classification problem with respective flags representing possible classifications, a machine learning model could be trained to classify data as valid or invalid. Such a model could even use prior classifications to ascribe specific flags to data points with varying levels of certainty.

To better leverage the capabilities of microservices in terms of scalability and handling large amounts of data, we plan on utilizing the Consul Service Discovery system. Consul not only addresses our need for a discovery service but also provides the accessible decentralized storage needed for the system to horizontally scale. In addition, it is our plan to utilize the Jenkins Automation Server for continuous integration and delivery. With dozens of microservices in active use, it is of great significance to streamline the automation of testing, building, and deploying each service with Jenkins. Finally, as the NRDC system scales further, the microservices will be tailored towards possible cloud platforms, such as Amazon AWS.

Finally, the development of this QC software could be implemented to be even more modular and easily deployable. By leveraging the power of containers, and storing all data and metadata within localized databases inside of those containers, a robust and almost plug-and-play QC solution can be installed to support a new research project within minutes. This self-contained solution would provide significant assistance to small research groups and institutions, who often have to invest considerable time and/or money into setting up bespoke data management infrastructure.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grant number IIA1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Bostock, M., 2018. Data-driven documents. URL: https://d3js.org/. (accessed on 03 March 2018).
[2] Campbell, J.L., Rustand, L.E., Porter, J.H., Taylor, J.R., Dereszynski, E.W., Shanley, J.B., Gries, C., Henshaw, D.L., Martin, M.E., Sheldon, W.M., 2013. Quantity is nothing without quality. BioScience 63, 574–585. doi:10.1525/bio.2013.63.7.10.
[3] ESIP, 2018. Federation of earth science information partners. http://wiki.esipfed.org/index.php/Main_Page. (accessed on 23 Feb 2018).
[4] ESRI, 2017. Arcgis. http://resources.arcgis.com/en/communities/data-reviewer/01rp00000006000000.htm. (accessed on 23 Feb 2018).
[5] Essenwanger, O.M., 1970. Analytical procedures for the quality control of meteorological data. Meteorological Observations and Instrumentation , 141–147doi:10.1007/978-1-935704-35-5_19.
[6] Foken, T., Gockede, M., Mauder, M., Mahrt, L., Amiro, B., Munger, W., 2004. Post-field data quality control. Handbook of Micrometeorology Atmospheric and Oceanographic Sciences Library , 181–208doi:10.1007/1-4020-2265-4_9.
[7] Google, 2018. Angular docs. URL: https://angular.io/. (accessed on 23 Feb 2018).
[8] Gouldman, C.C., Bailey, K., Thomas, J.O., 2017. Manual for real-time oceanographic data quality control flags. IOOS .
[9] Hill, D.J., Minsker, B.S., 2010. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. Environmental Modelling & Software 25, 1014–1022. doi:10.1016/j.envsoft.2009.08.010.
[10] LTER, G.C.E., 2017. Gce data toolbox for matlab. http://gce-lter.marsci.uga.edu/public/im/tools/data_toolbox.htm.
[11] Michener, W.K., 2006. Meta-information concepts for ecological data management. Ecological Informatics 1, 3–7. doi:10.1016/j.ecoinf.2005.08.004.
[12] NRDC, 2018. Nevada Research Data Center. URL: http://sensor.nevada.edu/NRDC/. (accessed on 04 Feb 2018).
[13] Ronacher, A., 2018. Flask: web development, one drop at a time. URL: http://flask.pocoo.org/. (accessed on 03 March 2018).
[14] Scientific, C., 2017. Loggernet. https://www.campbellsci.com/loggernet. (accessed on 23 Feb 2018).
[15] Scully-Allison, C., Le, V., Strachan, S., Harris Jr., F.C., Dascalu, S., 2017. A mobile quality assurance application for the NRDC. Proceedings of the ISCA 26th International Conference on Software Engineering and Data Engineering (SEDE 2017) , 185–192.
[16] USGS, 2018. USGS. URL: https://waterdata.usgs.gov/nwis. (accessed on 10 March 2018).
[17] Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., et. al, 2016. The fair guiding principles for scientific data management and stewardship. Scientific Data doi:10.1038/sdata.2016.18.