# Microservice-based Architecture for the NRDC

Vinh D. Le, Melanie M. Neff, Royal V. Stewart
Richard Kelley, Eric Fritzinger, Sergiu M. Dascalu, Frederick C. Harris, Jr.
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, Nevada, United States of America

*Abstract*—**The NSF EPSCOR funded Solar Nexus Project is a collaborative effort between scientists, engineers, educators, and technicians to increase the amount of renewable solar energy in Nevada while eliminating its adverse effects on the surrounding environment and wildlife, and minimizing water consumption. The project seeks to research multiple areas, including water usage at power plants, the effect of power plant construction on the surrounding ecology, alternative wastewater methods to maintain solar panels, and interdisciplinary solutions to improve solar energy in Nevada. In order to organize and analyze this data to produce effective change, Nexus needs a centralized database to store collected data. To this end the Nevada Research Data Center is designed to collect, format, and store data for scientists to view and consider. This paper presents a new architecture solution for the NRDC. Based in microservices, the solution aims to ensure scalability, reliability, and maintainability of this data center. Background on NRDC is provided in the paper, together with details on the proposed solution's software specification, design, and prototype implementation. A discussion of the microservice-based architecture's benefits and an outline of planned directions of future work are also included.**

***Keywords—Microservice Architecture; Monolithic Architecture; Software Specifications; Design; Prototype Implementation; NRDC.***

## I. Introduction

In 2013, the state of Nevada started the NSF EPSCOR funded Nexus project [1]. In order to store early data, an initial database system was created and implemented. As the Nexus project evolved over the years, this older database system did not. Our team has been assigned to redesign the current Nevada Research Data Center (NRDC) [2], formerly the Nevada Climate Change Portal (NCCP) [3]. We were tasked with reconstruct the system with a microservice architecture, an improvement from the original monolithic architecture. Microservice architecture is a relatively new approach to web-based service distribution, relying on independent instances of services that communicate directly to the client rather than the entire system. A microservice architecture will allow for rapid deployment of services, scalable resources, and dependable data management for the NRDC. An increase in demand of services paired with changing requirements for incoming data required a change in structure to the NCCP. These changes will bring about new avenues of growth and flexibility as the project itself progresses into its second year of development.

Our current microservices run from five flask servers, each housing a service to access one of five PostgreSQL databases. We have implemented a GUI for easy, quick access to database information. Entries in databases can be edited or destroyed through a terminal with proper authority, and new database entries can be added, with their unique ids individually generated. These modules are to eventually be linked to databases with data from Nexus sites, updated regularly by data loggers. The five modules are as follows: Person, Project, System, Component, and Service Entry. Each module is defined by the database information it possess and provides access to.

This paper, in its remaining part, is structured as follows. Section II provides background on NRDC; Section III presents details of our proposed solution's software specification; Section IV describes in detail the new microservice-based architecture for NRDC; Section V presents the prototype created to demonstrate the microservice-based architecture; Section VI includes a discussion about microservice-based architecture's benefits as a normal distributed service architecture; and finally, Section VII contains our conclusions and several pointers to future developments.

## II. NRDC

NRDC and NCCP were both created by the Cyberinfrastructure component of the NSF Nevada Nexus Track 1 Projects [1, 4] to gather and hold data for the scientists associated with the project. The Nexus Project's goal is to increase research, awareness, and productivity of alternative energy sources and the conservation of natural resources in the state of Nevada. The University of Nevada, Reno; the University of Nevada, Las Vegas; and the Desert Research Institute all cooperate across multiple disciplines as part of the NSF EPSCOR Track 1 Nexus Project. Cyberinfrastructure is the data management, storage, processing, and distribution component that all other components rely on to accurately interpret raw data. The remaining five components are the different fields of science that relate to the Nexus Project's goals [3, 4].

The Current NCCP system architecture, including the NevCAN sensor system, were described in the prior publications [5, 6, 7, 8, 9]. NRDC, as heir of NCCP, encompasses not only data, but also research results, including new CI methods and accompanying tools, such as DEMETER [10, 11], ATMOS [12], SUNPRISM [13], WEDMIT [14], and VISTED [15].

The current NRDC was created following the development of the NCCP, in order to handle unforeseen issues with its flexibility, scope, and complexity. Unfortunately, many faults from the monolithic architecture of the NCCP had persisted in haunting the current NRDC site. There was virtually no form of network monitoring, maintenance on the system required complete suspension of services, and the scalability limited the site's very own growth. Our design addresses these oversights of the system architecture, effectively decoupling the front and back ends of the system architecture so we can add new services like these at will.

The redesigned NRDC prototype aims at tackling each of these faults and provide a robust and efficient system. Instead of continuing work with the current architecture, the purpose of the redesign is to begin anew with a microservice architecture in mind. This entails the creation of separate enclosed services existing outside the scope of each other, completely autonomous and unaware of its fellow services[16]. Each service will be independent of one another and answer to a greater service registry for communication. Services are completely self-directed, with no dependencies on each other. The impact of this architecture design will bring about a drastic change in two major ways. First, the new NRDC will now be able to persistently remain actively live, even during a time of maintenance. Shutting down one service will not affect the overall site, as the other services will be live to support the website. As long as one service remains active, so will the entire site. Secondly, the architecture supports the swapping, addition, or removal of services. This feature finally allows the site a solution to the scalability issue, allowing for immense potential in growth.

Although the NRDC prototype itself is uniquely based around Nexus Track 1 Project, the idea of Microservice architecture is not. Throughout the development of this project, distribution models such as Netflix's Video Streaming Application and Amazon.com served as main sources of inspiration. Netflix and Amazon both use a cloud-based microservice architecture through Amazon Web Services (AWS). The idea behind the architectures is tentatively the same as the redesigned NRDC website, as to both host a vast amount of services at a given time to perform their basic functionality. However, a noticeable difference between Netflix or Amazon with the NRDC site is that the NRDC will be hosting its services on physical servers, as opposed to the instanced servers on AWS.

## III. Software Specifications

In terms of the Functional Requirements, a base level requirement of the NRDC will be providing subsequent service modules in accordance to the amount of services desired. As of the current iteration, the amount of services total up to 6, including People, System, Service Entries, Components, Deployments, and Projects. Although the services have a plethora of relationships amongst each other, the services will remain independent and communicate to each other by means of the Service Discovery requirement further along in this section. Another base level requirement will be the NRDC's ability to monitor network traffic, detect and handle anomalies, and acquire statistics like website hits. This gives the site the means to provide its administrators with the ability to track in-going and out-going requests, calls, etc. It will help sustain the site and optimize resource efficiency and distribution. The last base level requirement will be granting each service the ability to Create, Read, Update, and Delete entries in its database. To clarify, as the project is RESTful in nature, the services will enable these abilities via various POST and GET calls to specialized URLs.

Aside from the base level requirements, the NRDC will contain 3 secondary level requirements. The first requirement in this level deals with the creation of a centralized Service Discovery in order to locate and affect currently active services. As mentioned above, the Service Discovery deals primarily with the relationships amongst the independent services. However, this will be explained in depth in the Software Design section to come. The second requirement will be having the actual NRDC system remain active even while a service is inactive. The last requirement pertains to the ability for a service currently in use by the system to be swapped out for another at any given moment. This requirement is a key component of the NRDC's scalable architecture. Should a service in use become due for an upgrade, a separate service with the necessary specifications can be spawned and integrated in place of the previous outdated service. Or, should a service overhaul become necessary, the old version can remain in production until the development of the update is complete.

The NRDC additionally contains a singular third level requirement in the form of utilizing Amazon Web Services (AWS) to spawn services virtually instead of physically on a given server. Although this is not implemented in our current iteration, more discussion and information can be found further along in the Future Developments section.

In terms of Non-Functional Requirements, the NRDC Redesign currently hosts 5 main requirements. The first requirement is that the new NRDC system utilize the python programming language as its primary coding language. The second requirement is for the NRDC to contain the ability to deploy on any platform, be it Windows, Linux, or otherwise. The third requirement is to utilize Python Flask as the main library and framework for each service. The fourth requirement is that each service utilize the Python Requests library in conjunction with Python Flask. Finally, the last requirement would be for the NRDC's database to be that of a PostgreSQL. More detailed information can be found below in the Software Design section.

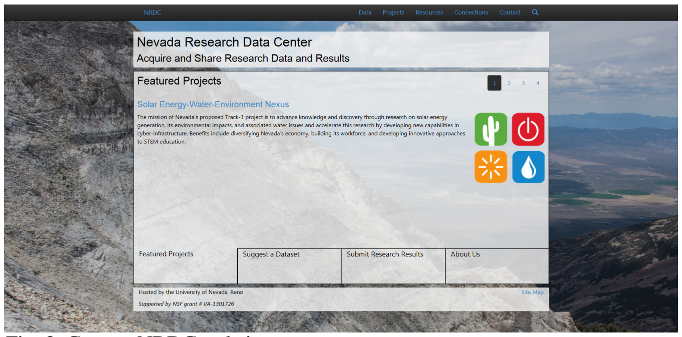
## IV. Software Design

The redesign of the NRDC consists of five major parts: modules, service discovery, website, the database and the Application Program Interface (API). The modules take the place of the typical monolithic server-side application, and use the service discovery as a type of local DNS to learn the locations of the other services. Their relation to each other can be seen in Figure 1.

The modules are the different services that receive input from the Service Discovery and the API to retrieve and present information requested from the databases. These autonomous modules do not communicate directly, and are only aware of each other through querying of the Service Discovery table. The modules' independence is an essential characteristic of the microservice architecture. This allows modules to be taken down, adjusted, or created without having to halt or alter any of the other modules. The modules are the only component of the NRDC that makes calls to the database for requested information. The current list of modules include: People, Projects, Service Entry, and Deployments.

The Service Discovery component communicates directly with the modules and acts as a lookup table to find the locations of the other modules by IP address and port number. Service Discovery holds the locations of each service as each service does not know the address of the other. If new instances of services were to be created their new address would be registered with the Service Discovery until that service was terminated. In some instances a module may request information from another module; the service discovery would be responsible for the communication between the two modules. In a way, the service discovery acts as a local DNS server, relying information about addresses and locations to services that request it. Our current Service Discovery system is Netflix's Eureka system. Eureka allows for both service discovery and monitoring of services. Eureka was originally designed to be used in Amazon Web Services (AWS) in a cloud-based distribution system, which will be covered in the upcoming section Future Development.

The website acts as the main interface, GUI, and the front end for the NRDC. Information about the services and the Nexus Project will be presented here, as well as information visualization. As mentioned earlier, as a component of Track 1, the website will emphasize not only information about the NRDC, but other associated research and helpful, educational links and materials related to Nexus. The overarching goal of the Nexus project is to further renewable energy ambitions in Nevada, and we will reflect this goal in the NRDC. This website will act in the same way as the NCCP website did previously, providing users with project information and a simple way to view database entries. This is the most common purpose of the NRDC. A prototype of the website [3] has already been created as shown in Figure 2.



Nevada Research Data Center

Acquire and Share Research Data and Results

Featured Projects

Solar Energy-Water-Environment Nexus

The mission of Nevada's proposed Track-1 project is to advance knowledge and discovery through research on solar energy generation, its environmental impacts, and associated water issues and accelerate this research by developing new capabilities in cyber-infrastructure. Benefits include diversifying Nevada's economy, building its workforce, and developing innovative approaches to STEM education.

Hosted by the University of Nevada, Reno

*Supported by NSF grant # IIA-1301726*

Fig. 2. Current NRDC website

The API is an alternative way to access functionality in the NRDC. Users have the option of using either the website or the API to access the NRDC. The API is accessed through a command line and uses a RESTFUL API design to make calls to the NRDC. The API primary function is to allow users to write scripts or programs to interface with the NRDC. These functions include: creating, updating, reading, and deleting of information depending on the user's authentication level. Currently, information sent back to the user is in the form of a JSON array, and information that needs to be sent to the NRDC must come in the form of a JSON. This way, information can be quickly translated into other forms, and is easily readable to the human eye.

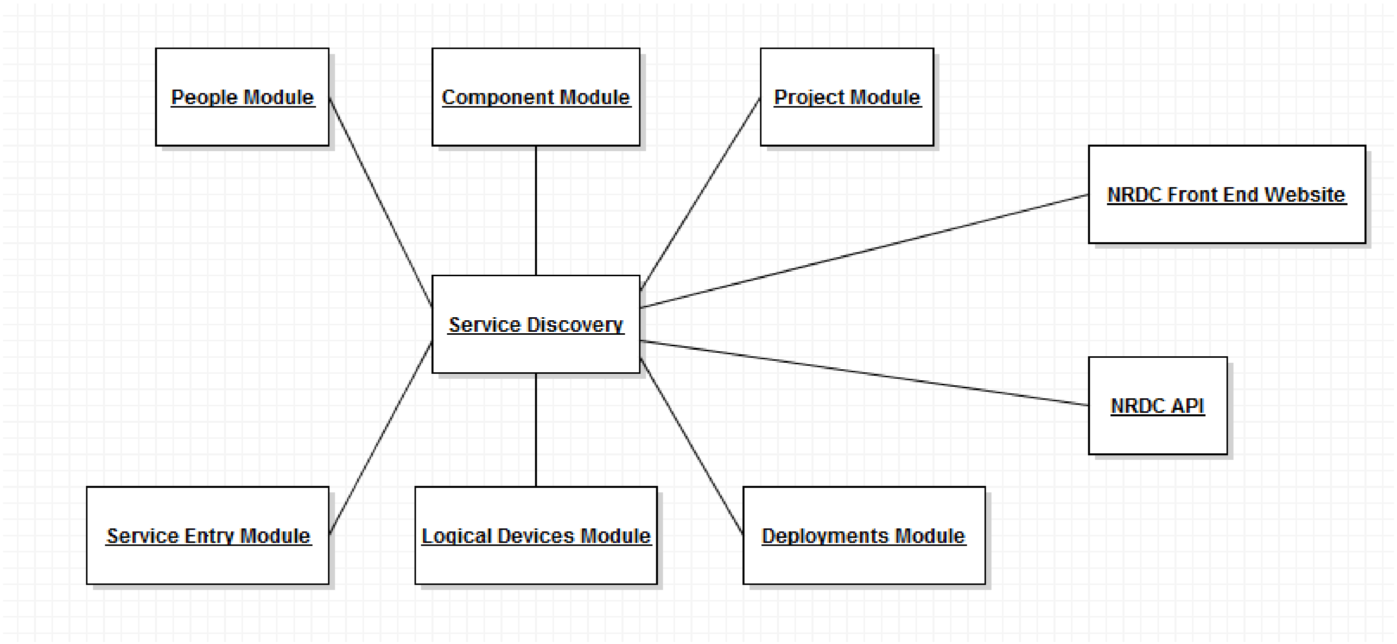


Fig. 1. NRDC software design diagram

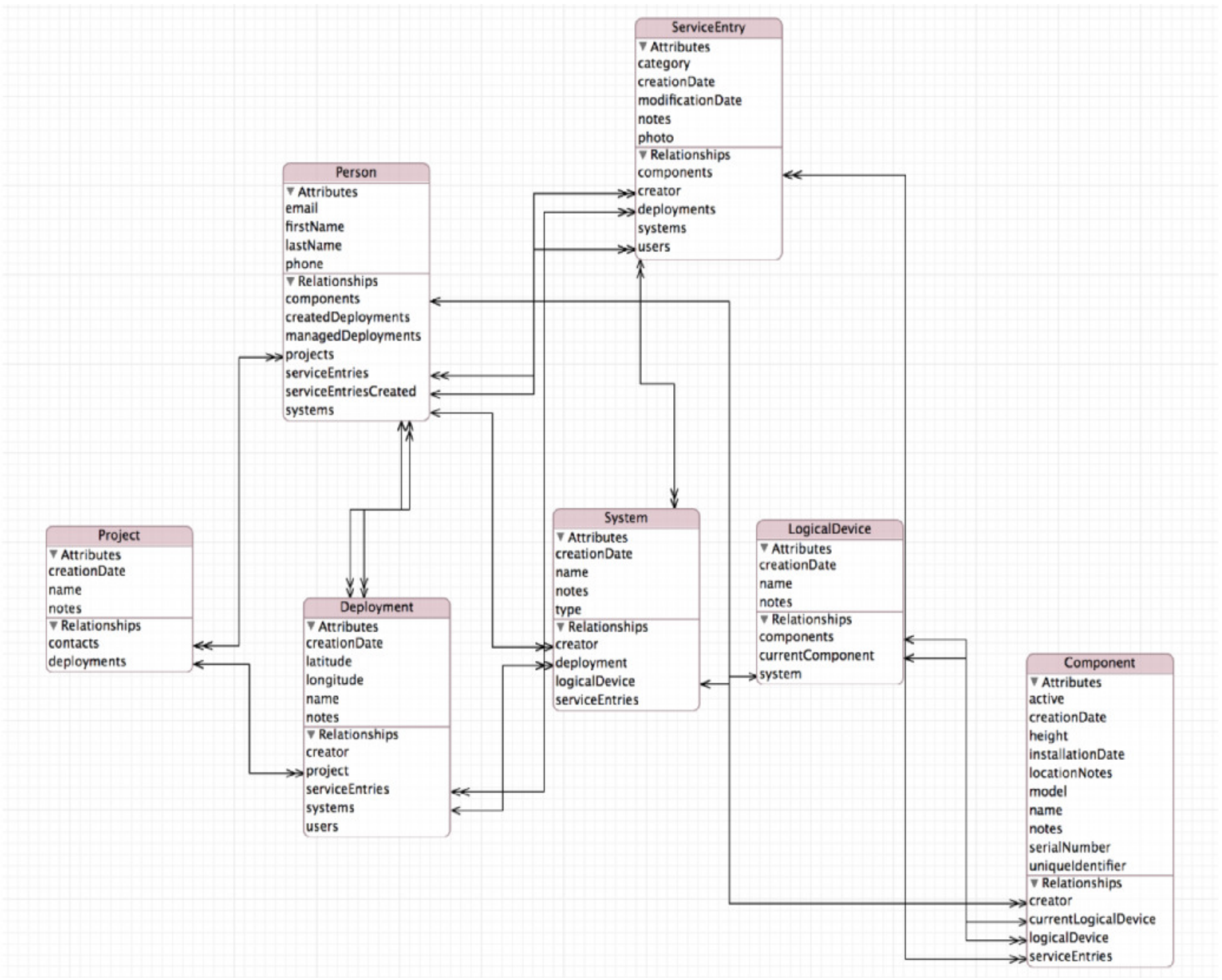


Fig. 3. NRDC relational database diagram

The database holds all information that is accessed or as a local DNS server, relying information about addresses added by the services. The databases being used are PostgreSQL databases. PostgreSQL was chosen because it is a relational database and is highly regarded for its robustness [17]. We anticipate high amounts of data in many complicated formats, and chose a database that could handle such a large task. Like all SQL databases, information is stored in relational tables. Each table's name and variables are determined by what modules are currently in use in the NRDC. There are many relations between tables, as displayed in Figure 3, so JOIN tables are created to link certain information from one table to another.

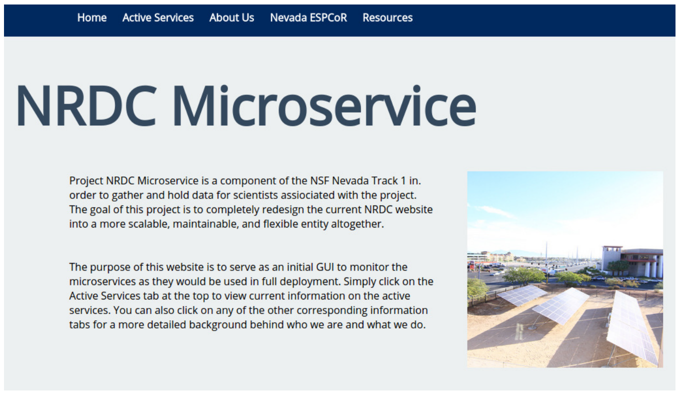
As an example, if a user wants to make a simple request to the NRDC to read information from the People Module, the user will access either the NRDC API or the NRDC Front End Website. From there, information from their chosen interface is sent to the Service Discovery to locate the appropriate service, where it is then sent to the respected module of the request. The People Module makes a SQL call with the related information from the Service Discovery to the Database. The Database sends the information back to the People Module and that information is relayed back to the user.

## V. Prototype Details

The current prototype of the NRDC is a collection of essential services running on Flask servers. These services, which are congruent with the modules described above, manage PostgreSQL databases and are written in the Python programming language.

We chose to use the Python programming language due to its incredible ease of use and vast amounts of libraries that add almost limitless functionality. Compared with other programming languages, Python is hailed as one of the easiest and most useful languages, and its popularity facilitates the creation of multiple useful software tools for Python programs. The readability of Python was also taken into consideration as it is a much simpler language to read and understand with little to no programming experience. Considering that this project will be maintained and developed by other individuals after the team's graduation, Python was a natural choice to ensure that our code was optimally understandable. Python's string concatenation is very easy to use and allows for better control when creating SQL commands from the modules. As mentioned earlier, the modules pass information through a JSON between database and end user. Python allows for easy JSON parsing of both incoming and outgoing JSON, providing us full control of data processing without additional overhead.

The Flask microframework provides a flexible, ideally minimal template for our microservices, giving us the opportunity to select our own database. Flask, a python web application framework, allows not only the independent deployment of servers, but also incredible flexibility to administer each server. These Flask servers are coded independently and can be uploaded to Github as templates for future development of modules, which can then be easily added on the fly.



NRDC Microservice

Project NRDC Microservice is a component of the NSF Nevada Track 1 in. order to gather and hold data for scientists assiociated with the project. The goal of this project is to completely redesign the current NRDC website into a more scalable, maintainable, and flexible entity altogether.

The purpose of this website is to serve as an initial GUI to monitor the microservices as they would be used in full deployment. Simply click on the Active Services tab at the top to view current information on the active services. You can also click on any of the other corresponding information tabs for a more detailed background behind who we are and what we do.

Fig. 4. NRDC demonstration GUI

It provides tools to expand the framework, such as FlaskAdmin, but we opted to design our own tool for database manipulation using CRUD. We evolved the prototype from the original Django framework because we feel Django's central "project" structure is contradictory to our microservice modularity goals. Flask's microframework for web applications corresponds nicely with our microservice design. In our Flask servers we use the psycopg2 library to perform the necessary SQL operations on our PostGRES databases, yet another useful Python library. An important factor in our design is the ability to constantly change it. We could write a new service in a different programming language, or using a different framework, if it better suited the function of that module. For our current modules, Flask and Python were just concise, modular, and elastic enough to fit our changing needs.

Once the database information is retrieved, it is presented in JSON format through our GUI. The JSON formatter tool helps organize and format data in a way that is readable to the human eye, which is helpful for our GUI, since its sole purpose is to read data. Figure 4 shows our GUI that was solely for demonstration purposes and does not reflect what the final or the current NRDC will look like. Our GUI is a simple website, designed with jquery, that displays the raw JSON data and all its attributes based on which module was called at the time. The GUI updates in real time, since it only makes calls to the respected module at the time the user make a request. As a part of a recent demonstration, we developed the GUI and linked it to our services, then proceeded to add users to the database. Without having to refresh the webpage, the user can simply select the respected module and the information within will be displayed through the GUI. As mentioned earlier, this idea will be further implemented in the NRDC as more data becomes available to us for processing and presentation.

## VI. Discussion

To examine the benefits of a microservice architecture, one must first understand a monolithic architecture. A monolithic architecture is generally composed of some centralized server application that handles requests, a client-side application that makes requests, and a database where information is fetched and stored. Everything runs from a single executable, and any interaction must spin up the entire application. This style has many limitations and shortcomings. Design choices such as coding language, application frameworks, and database selection are fixed, and need to be consistent throughout the application. Scalability is greatly limited, as any additions only add to the size of the original executable, which is already limited by hardware and web container specifications [3]. On a developer level, monolithic code is difficult to understand and modify because of its massiveness, and making changes requires extensive reworking of the code as a whole.

A microservice architecture seeks to alleviate these problems, and provide a scalable, maintainable, and flexible structure that is easy to modify and renovate[18]. A single microservice is a completely autonomous unit of execution with a single, clearly-defined purpose. Microservices are deployed individually of any central structure, and can have unique characteristics and components relevant to its provided function. They can be written in different languages or using different web frameworks, and implement differing database structures. New microservices can be created easily, and without making changes to other services already in production. Microservices can even be replaced completely by new updates, with considerably reduced maintenance downtime. Microservices allow for modularity in program design, so code is easily understandable and recognizable. They also lend a certain intuitiveness to software development, as teams of engineers can work on various respective services with clearly defined goals and tasks.

As software engineers, we are constantly being asked to change, to adapt, and to evolve as technology continues to present new opportunities. The microservice style of software application design provides a means for meeting these demands by maintaining system integrity with growth through clear modularity, and by allowing the use of multiple differing program languages, frameworks, and software tools within a single application. Most importantly, its suite of independently deployable, maintainable, and upgradable services offer an efficient alternative to the clunky, limited monolithic architecture of the past.

## VII. Conclusion and Future Developments

This paper has presented a new architecture for NRDC based on the modern solution offered by microservices. The main characteristics of the proposed new architecture have been described via software specification, design components, and prototype details. We believe this architecture brings several significant benefits to NRDC, including scalability, reliability and maintainability.

Future developments of NRDC include transferring all services and modules to a cloud based operation. This allows for more rapid deployment of services, easier scalability, and easier administration of services for the future. NRDC could implement its own cloud service or an existing service such as Amazon Web Services.

The microservice architecture lends itself naturally to a cloud-based system. Services would each run their own instance of a server, whose size could be fitted exactly to the requirements of each service, ultimately optimizing hardware usage and efficiency. Moving to the cloud would eliminate hardware limitations, and allow virtually infinite access to resources, which could be scaled up or down in parallel with the microservices.

In the future, we imagine the NRDC to be among the most relevant and important data systems for scientists in Nevada. This means the amount of data, and the complexity of that data, will certainly increase. The NRDC could eventually implement a big data solution such as Apache Hadoop and Google MapReduce, to facilitate data analytics and data management with distributed processing. While PostGRES databases are robust, the NRDC already handles a large amount of raw data, and we will eventually need to explore new strategies to handle its processing in order to accommodate new projects and new avenues of research beyond the Nexus project.

## Acknowledgment

This material is based in part upon work supported by the National Science Foundation under grant number IIA-1301726. 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] Nevada System Sponsored Programs and EPSCoR. http://epscorspo.nevada.edu/ [last accessed on February 17, 2015 ]

[2] Nevada Climate Change Portal. http://sensor.nevada.edu/ [last accessed on February 17, 2015 ]

[3] Nevada Research Data Center. http://sensor.nevada.edu/nrdc/ [last accessed on February 17, 2015 ]

[4] NSHE Solar Nexus Project. http://nvsolarnexus.org/ [last accessed on February 17, 2015 ]

[5] S. Dascalu, F.C. Harris, Jr., M. McMahon Jr., E. Fritzinger, S. Strachan, R. Kelley, "An Overview of the Nevada Climate Change Portal" Proceedings of The 7th International Congress on Environmental Modelling and Software (iEMSs 2014) , Vol 1, pp 75-82, June 15-19, 2014, San Diego

[6] M.J. McMahon, Jr., S. Dascalu, F.C. Harris, S. Strachan, and F. Biondi (2011). Architecting Climate Change Data Infrastructure for Nevada, in Salinesi, C. and Pastor, O. (eds.), *Advanced Information Systems Engineering Workshops CAISE-2011, Lecture Notes in Business Information Processing, LNBIP-83*, June 2011, Springer, pp. 354-365.

[7] M.J. McMahon, Jr., F.C. Harris, Jr., S. Dascalu, and S. Strachan (2011). S.E.N.S.O.R.- Applying Modern Software and Data Management Practices to Climate Research, *Procs. of the 2011 Workshop on Sensor Network Applications (SNA-2011)*, Nov. 2011, Honolulu, HI, pp. 147-153.

[8] R. Motwani, M. Motwani, F.C. Harris, and S. Dascalu (2010). Towards a Scalable and Interoperable Global Environmental Sensor Network Using Service-Oriented Architecture, *Proceedings of the 6th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP-2010),* December 2010, Brisbane, Australia, pp. 151-156.

[9] Mensing, S. S. Strachan, J. Arnone, L. Fenstermaker, F. Biondi, D. Devitt, B. Johnson, B. Bird and E. Fritzinger. 2013. A network for observing Great Basin climate change. EOS, 94(10):105-112.

[10] S. Dascalu, E. Fritzinger, S. Okamoto and F.C. Harris, Jr. (2011). Towards a Software Framework for Model Interoperability, *Procs. of the 9th IEEE International Conf. on Industrial Informatics (INDIN 2011),* July 2011, Lisbon, Portugal, IEEE Computer Society, pp. 705-710.

[11] E. Fritzinger, S.M. Dascalu, D.P. Ames, K. Benedict, I. Gibbs, M. McMahon, and F.C. Harris (2012). The Demeter Framework for Model and Data Interoperability. *Proceedings of the International Congress on Environmental Modeling & Software (iEMSs-2012)*, Leipzig, Germany, July 2012, pp. 1535-1543.

[12] A. Dittrich, S. Dascalu, and M. Gunes (2013). ATMOS: A Data Collection and Presentation Toolkit for the Nevada Climate Change Portal. *Proceedings of the International Conf. on Software Eng. and Applications (ICSOFT-EA 2013),* Reykjavik, Iceland, July 2013, pp. 206-213.

[13] S. Okamoto, R.V. Hoang, S.M. Dascalu, F.C. Harris, and N. Belkhatir, (2012). SUNPRISM: An Approach and Software Tools for Collaborative Climate Change Research. *Procs. of the 13th Intl. Conf. on Collab. Tech. and Systems (CTS-2012)*, May 2012, Denver, CO, pp. 583-590.

[14] J. Patel, S. Okamoto, S.M. Dascalu, and F.C. Harris (2012). Web-Enabled Toolkit for Data Interoperability Support. *Procs. of the International Conference on Software Engineering and Data Engineering (SEDE-2012)*, Los Angeles, CA, June 2012, pp. 161-166.

[15] L. Ravi, Q. Yan, S.M. Dascalu, and F.C. Harris, Jr. (2013). A Survey of Visualization Techniques and Tools for Environmental Data. *Proceedings of the 2013 Intl. Conference on Computers and Their Applications (CATA 2013),* March 2013, Honolulu, Hawaii.

[16] M. Fowler, "Microservices". ThoughtWorks. http://martinfowler.com/articles/microservices.html [last accessed on February 17, 2015 ]

[17] P. Shaughnessy, "Following a Select Statement Through Postgres Intervals".http://patshaughnessy.net/2014/10/13/following-a-select-statement-through-postgres-internals [last accessed on February 17, 2015 ]

[18] C. Richardson, "Pattern: Microservices Architecture" Microservices.io. http://microservices.io/patterns/microservices.html [last accessed on February 17, 2015 ]