

**“Improving the
Architecture & Development
Process for AIRPACT”**

Mentor: Joe Vaughan

Instructor: Aaron Crandall

Course: Computer Science 423

Team: Kyler Little, Garrett Rudisill, Slater
Weinstock, Jeff Kremer

Date: 12/01/2019

Table of Contents

Background	2
Project Description	2
Features	3
Literature Review	4
Stakeholders Requirements and Needs	5
Mapping of Requirements to Technical Specifications	6
Detailed Implementation Framework	6
Concepts, Algorithms, or Other Formal Solutions to Build the Product	9
Solution Selection Process	9
Software Testing Implementation	10
Software Testing Results	11
Summary of the State of This Project	13
References	15

I. Background

The purpose of the AIRPACT (Air Indicator Report for Public Access and Community Tracking) project is to predict the air quality for the region including Idaho, Oregon, and Washington using data gathered from a variety of resources. These resources include, but are not limited to, fire emissions, mobile emissions, point source emissions, meteorological data, and other ancillary data. Air pollution is a primary cause in a number of health conditions, and can exacerbate others. The model predicts more than a hundred species of chemicals, including ozone (O_3), nitric oxide species (NO_x), carbon monoxide (CO), and particulates (PM_{2.5}). AIRPACT uses many input data sources in order to do so. These input sources contain information on emissions, meteorological data, initial conditions (the prior day's results), and boundary conditions (data from bordering geographical regions). The model then forecasts those pollutants for 48 hours on a 4 km² grid to indicate the characteristic grid spacing. The current AIRPACT model is AIRPACT-5 and consists of 285 columns East/West, 258 rows North/South, and 37 layers vertically.

II. Project Description

As the title suggests, the overall goal of this project is to improve the architecture and development process for the AIRPACT-5 system. The ultimate goal in doing so is to allow for containerization of AIRPACT-5. As a “case study”, we seek to containerize a component of the AIRPACT-5 system known as the BlueSky Framework and to establish a better development process while doing so. In this way, we can document our learnings and how to proceed with future containerization of other AIRPACT-5 components.

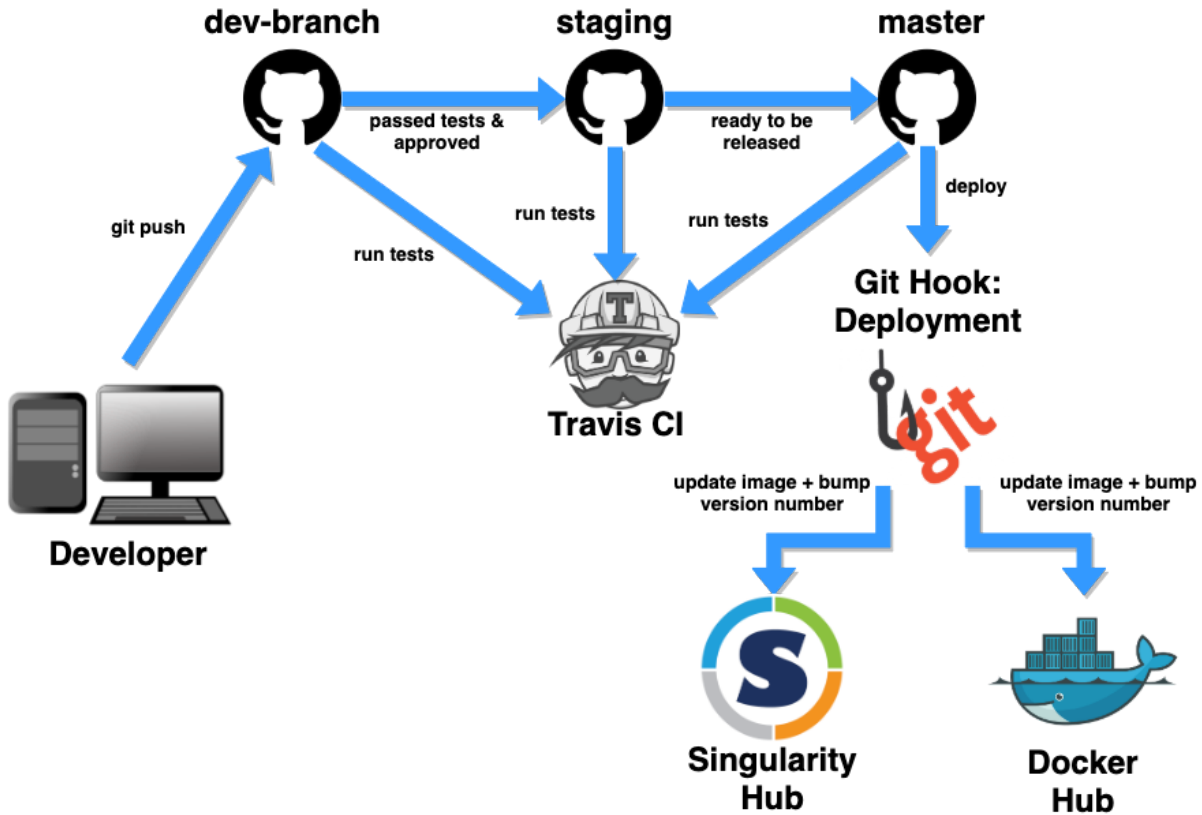
The benefit of containerizing this specific component of AIRPACT-5 is that it will free up a physical machine running a dedicated Ubuntu Server. This is because the BlueSky Framework requires an Ubuntu operating system to operate, while Aeolus is a CentOS-based server cluster. Thus, in containerizing the BlueSky Framework, we would free up the server space for use by other systems.

The reason containerization is so critically important is that it allows for portability of software moving forward. Containerization provides the ability to encapsulate components of the system, as well as the ability to encapsulate the entirety of the system into a virtual machine that can be run on any system that supports Singularity or Docker. This portability and modularity allows for effortless parallel runs, the ability to swap out modules, and create varied output datasets. While the results of any given execution of AIRPACT are dependent upon the input data, the results are also dependent on configurable parameters given to the software system. Scientists in charge of AIRPACT currently use a fixed set of parameters that they deem are best. With concurrent execution on different hardware, scientists could utilize a variety of parameters to produce different models, a technique known as ensemble modeling. Ensemble modeling is known to produce higher quality predictions, entailing better air quality tracking for the entire Pacific Northwest Region. Thus, in a seemingly round-a-bout fashion, our project's ultimate goal is to produce better air quality predictions for the Pacific Northwest.

III. Features

- Functional
 - Replacement of existing BlueSky Framework Server with [officially maintained BlueSky Framework Docker Image](#) on the Aeolus HPCC
 - Creation of a “Standardized Procedure” for containerization of a generic component of the AIRPACT-5 system, including:
 - a) Documentation
 - b) Shared storage
 - (1) Options: Docker Volume, Docker Bind Mount, tmpfs mount
 - c) Usage of version control system (e.g. Git) and repository hosting service (e.g. GitHub, GitLab, etc.) for the given component
 - d) Ability to specify computing resources used by container
 - e) Verification process of the container’s output (i.e. that it aligns with the actual component’s output)
 - (1) Automated testing so far as possible
 - (2) Documentation for testing that cannot be automated
 - f) Continuous integration & continuous deployment
- Limitations
 - Usage of version control system (VCS)
 - Could be problematic due to size constraint on remote repositories
 - Ability to containerize component of AIRPACT-5 system:
 - Could be problematic due to the monolithic nature of AIRPACT-5 System, countless hard-coded paths, dependency of component on Aeolus file system, and lack of existing version control
- “Standardized Procedure” architecture overview
 - The idea is that each component of AIRPACT-5 will have its own repository with CI/CD set up. The architecture for one of these repositories can be viewed in Figure 1 below.
 - a) It utilizes a Continuous Integration (CI) system known as Travis CI. It also utilizes a Continuous Deployment (CD) system, which is built on top the CI system. Once the CI system deems that a branch is ready to be merged and all developers approve it, Git Hooks will deploy these successfully tested artifacts to DockerHub and SingularityHub.

Figure 1: Visual of Architecture



IV. Literature Review

This project will involve an assortment of software that materializes to a finalized and scalable product. AIRPACT-5 is the core software – a government sponsored weather and environment modeling software used to provide forecasts of air quality over the Pacific Northwest region. This software is run on a HPC. These runs create highly accurate models based on various parameters and equation types. The benefits of containerization would be the failsafe capability of having duplicate runs on AIRPACT-5 on systems and services other than Aeolus, the possibility of getting better forecasts from ensemble averaging, and the averaging of multiple runs with slight variations in input data and/or control parameters. More specific info can freely be obtained online¹. For all intents and purposes, the AIRPACT-5 software components will be treated as a collection of black box systems, without any need for interference or modification on our part.

The success of this project hinges on successful use containerization services to create a scalable instance of the AIRPACT software. Our primary intent is to use Singularity² to run an image of BlueSky Framework on the Aeolus cluster. Singularity images can be generated from Docker images³, which contain a script with instructions on how to build a standardized virtual machine, including the operating system, software package (in our case BlueSky Framework), and any other scripts necessary to get the system booted and functional. With this in mind, the

next technology to be considered is the final deployment location, given the HPC needs for the AIRPACT software.

We will also be implementing version control into the software development cycle for AIRPACT-5. To serve this purpose we will use Git Version Control⁴ and GitHub Repositories⁵. These systems serve as a means to track changes made to the files of AIRPACT-5, as well as allowing other services such as reverting new changes, and instituting a testing pipeline. Along with GitHub, we will be using SingularityHub⁶ and DockerHub⁷. These services provide the support for Docker and Singularity containers in much the same way GitHub support software development. Git, GitHub, SingularityHub, and DockerHub are vital tools for maintaining the software base of AIRPACT-5, and will slow the growth of new technical debt within the system, if used properly.

The testing pipeline we have implemented for the BlueSky Framework within its repository relies on Travis CI (Continuous integration)⁸. Travis CI is a free, open source, testing software. The purpose of this software is to ensure that changes made to systems during development do not break those systems. This allows for automated error checking any time changes are made to the software base.

V. Stakeholders Requirements and Needs

Stakeholder	Main Interests	Needs	Priority & Project Impact
Joe Vaughan	<ul style="list-style-type: none"> AIRPACT to be more robust/failsafe To explore if ensemble runs of AIRPACT are feasible To explore whether those ensemble runs yield better forecasts 	<ul style="list-style-type: none"> Containerization of BlueSky framework Containerization of each component of AIRPACT-5 Increased long-term system health Benefits of different containerization methods/software 	1/High
Aaron Crandall	<ul style="list-style-type: none"> Student evaluations Managerial role 	<ul style="list-style-type: none"> Deliverable Program before thanksgiving Testing Framework set up 	2/High
Andrew Bates	<ul style="list-style-type: none"> Aeolus use Permissions Understanding of storage and singularity versions 	<ul style="list-style-type: none"> Tickets for permissions modifications Tickets for singularity issues 	2/High

To expand on the needs of our stakeholders, our first step will be to upload the official BlueSky Framework container to Aeolus for integration. Following this, we will document the process, as well as the systems set up or existing to support these containers. Along with documenting the process of containerization and how to support it, we will institute a sustainable version control system to ensure the long-term health of AIRPACT-5 as it is continued to be containerized. This will ensure the tracking of changes, and who is making those changes, as well as allowing reverts of the system if needed. In the future, changing to the new BlueSky system, instead of BlueSky Framework, should be done. This progress, along with other modifications of components, will be tracked in GitHub.

VI. Mapping of Requirements to Technical Specifications

Our requirements are relatively straightforward given the type of project. Our project inherits all requirements of AIRPACT-5 and any software dependencies within the project including, but not limited to: BlueSky, CMAQ, WRF, and SMOKE. Docker is merely a virtualization container, which will run the required OS, as well as a respective instance of the AIRPACT-5 software. A currently scheduled job, on the Aeolus HPCC, will use up to 2 nodes with 60 CPU cores/threads each, and requires a minimum of 100 GB of hard disk space for data files involved in the calculations. The only new requirement/restriction that comes into play with this project is the performance overhead of virtualization. This can vary greatly depending on the servers the Docker images are spun up on. Each instance requires the full resources of a natively run instance of AIRPACT-5.

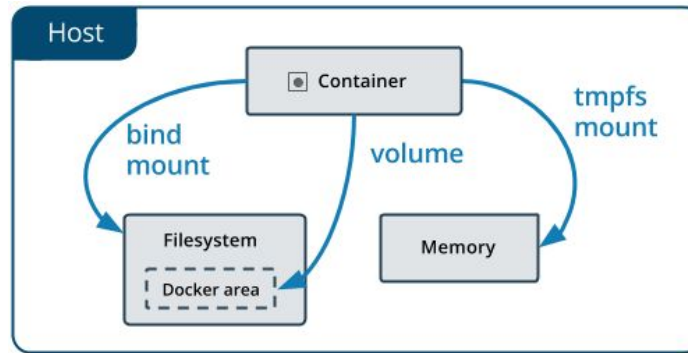
Another option we have is to create Docker containers for each of AIRPACT-5's sub-jobs. This could lead to reduced overhead on the individual nodes, while still keeping full functionality of AIRPACT-5. The sub-job we would most likely do this to would be the BlueSky system. This is because BlueSky currently runs on a dedicated ubuntu server that could then be freed up if we created a working Docker image of the system. This would allow the BlueSky server to be run in an as-needed style, on a wide variety of hardware.

VII. Detailed Implementation Framework

1. Work with Andrew Bates and Dr. Vaughan in order to set up a shared working directory for the four of us on the Aeolus HPCC.
2. Each person on the team creates a basic Docker Image that runs a "Hello World" program. Each person will choose a different programming language and run the Docker Container. This will familiarize the team with Docker technology.
3. Create a configurable and parameterized Docker Image for sample program, as this is how Docker Images should be configured for the jobs of AIRPACT.
4. Create a Docker Image for parallelized code. Since some of the jobs of AIRPACT (i.e. MEGAN, CMAQ) are parallelized, this knowledge is valuable.

5. Containerize the BlueSky job, one of many batch jobs that AIRPACT invokes, using Docker, as our alpha prototype. The batch job currently runs on an Ubuntu-based server (IP 10.1.1.28) within the Aeolus Cluster. The server/machine that runs this job is exclusively dedicated to running the BlueSky job, so we can free up this machine for other purposes once the BlueSky job is containerized and ported elsewhere. This is additionally important because it will allow us to create a framework for using Docker, containerization practices, and potentially cloud computing. This framework will be used to containerize future batch jobs of AIRPACT.
 - a. Identify and include dependencies (software packages) utilized by the BlueSky job within our Dockerfile to be used by the Docker Image.
 - b. Finish containerization of BlueSky job using Docker on our own local machines, using Git and GitHub as our VCS and Remote Repository Manager, respectively. Modify the code to redirect all output to a local destination within the container, to avoid affecting AIRPACT.
6. Since there exists a maintained Docker Image for BlueSky, replace the existing BlueSky Server with this officially maintained BlueSky Docker Image on the Aeolus HPCC.
 - a. Port the maintained Docker Image to Aeolus HPCC by [creating a Singularity Image from the Docker Image](#).
 - b. As much as possible, verify that the maintained BlueSky Singularity Image produces output that is consistent with the existing BlueSky Server's output.
 - c. Replace the scripting call to the BlueSky Server in AIRPACT-5 to utilize the Singularity Container rather than the dedicated Ubuntu server. We can then free up this machine for other purposes.
7. Create a standardized procedure for containerization of a generic component of the AIRPACT-5 system as our minimum viable product. At a very minimum, the following technical concepts should be addressed in this procedure.
 - a. Shared Storage
 - Investigate the proper way to handle shared storage between containers. Due to some of the components of AIRPACT-5 depending on each others' outputs, the containers must have access to some shared storage. Options for Docker include Docker Volumes, Docker Bind Mounts, and tmpfs Mounts, as seen in the figure below.

Figure 2: Docker Storage Options



For sake of ease, it seems that bind mounts are the easiest solution for the purposes of this project, albeit not a good long term solution.

b. Version Control

- Establish the usage of a Version Control System (e.g. git) and repository hosting service (e.g. GitHub, Gitlab, etc.). A GitHub Organization for the LAR Group will be used to house repositories for the individual components of AIRPACT-5. Besides the usual benefits of using a VCS (easy testing, developer maintenance, etc.), VCSs have the added benefit of the ability to use CI/CD (see bullet point 'd').

c. Testing

- Create an automatic verification process for the container's output, so that we can be reasonably confident that a container's output aligns with the non-containerized component's output.
 - Create an automated testing framework insofar as possible.
 - Create documentation for testing that cannot be automated.
- Investigate the ability to specify computing resources used by a container.
- Investigate the ability to deploy a container to a cloud provider (e.g. AWS, Azure, etc.)

d. Continuous Integration (CI) & Continuous Deployment (CD)

- Create a development process such that code can be continuously integrated into staging and master development branches when successfully tested. Additionally, implement a process such that successfully tested containers (which individually run a component of AIRPACT) can be continuously deployed to their respective remote registries (e.g. DockerHub).

e. Documentation

- Write extensive documentation for how this process can be replicated on any generic component of AIRPACT.

*As an important note, the components we will actually be writing are the Dockerfiles, tests, and scripts to automate deployment & testing. The Docker Images/Containers are "Commercial Off-The-Shelf" (COTS) components produced by Docker; similarly, Singularity Software is a COTS product used to run Docker Containers on an HPCC.

VIII. Concepts, Algorithms, or Other Formal Solutions to Build the Product

This project revolves around the concept of virtualization and the very related topic of containerization. Virtualization utilizes special CPU instruction sets and software to emulate or mimic certain configurable software/hardware combinations regardless of what platform they run on. A commonplace example of virtualization would be running Parallels on a Mac to access Windows and run applications that are written for Windows only. One of the primary advantages of virtualization is that one can continue to run legacy or unsupported software, even as hardware/software on the underlying system changes. This is extremely advantageous in the HPCC world, as groups upgrade servers and underlying operating systems and potentially lose native support for older necessary programs.

Containerization simply builds upon the advantageous nature of virtualization. Containerization is the process of creating single purpose virtualized machines, which can be deployed on any number of HPCCs and server providers. An example of this would be a web server. One can containerize a simple web server, which requires only 2 CPU cores, 4GB of RAM, and 20GB of storage space, and run multiple instances of this server to provide redundancy on the physical hardware running the virtualization software, such as Docker. A typical server within the last 3-5 years may have 16 CPU cores, hundreds of GB of RAM, and terabytes of storage.

Given the strongly cemented nature of the pre-existing codebase, containerization and virtualization is the primary method of tackling the problem of scaling older software. The only other method that could adequately tackle the problem at hand would require significant overhaul of the existing hardware, which is neither realistic nor feasible in our given time constraints and knowledge of the underlying code base.

Our project also will heavily be focused on a modular style. Specifically, we want to develop a modular style of containerizing a component of the AIRPACT-5 system such that the process can easily be replicated in the future. The AIRPACT-5 codebase contains not only many languages but also varying system types. The BlueSky Framework component of AIRPACT-5 runs on a different operating system from the host machine, which requires it to be its own container. This style of treating various jobs, such as BlueSky, as their own containers will not only make containerization of the project easier by reducing what each container individually encapsulates, but also by allowing the AIRPACT-5 system to be upgraded in an easily maintainable way by switching out containers in various parts of the process.

IX. Solution Selection Process

We will be using Docker to accomplish our goal of containerizing the BlueSky Framework. Docker is the best containerization tool available on the market and is available for free. Additionally, Docker containers can be deployed to AWS and other cloud services, which contributes to achieving our scalability goal. Lastly, Docker images can easily be converted into Singularity images, which is a requirement for this project. Singularity is a containerization tool specifically designed for HPCCs. As a result, it is the only containerization tool available on the Aeolus HPCC, and so it's necessary for us to use in order to deliver our work at the end of the

semester. Thus, using Docker allows us to have the most flexibility in terms of deployment. For our Version Control System, we will use Github. GitHub is also one of the best VCS tools on the market and integrates well with CI/CD systems. We will use Travis CI as our CI/CD system since it so easily integrates with GitHub. Additionally, it is a free-to-use service if the repository is open source. Since no code written by WSU in AIRPACT-5 is proprietary, we think this is a fair sacrifice.

Due to concerns raised by the system administrator maintaining Aeolus, where AIRPACT-5 currently runs, we may run into some software integrity issues with the system. Various environment variables are present within the system in undocumented forms which may be blocking to progress or may be documented as limitations of the project as a whole. These variables may be difficult to reproduce entirely without limiting the goals of modularity and extensibility of the project. These variables likely affect data output, but their full effects are unknown to both the development team as well as the system administrators. As we continue to run into difficulties with the original system, the goal of the project is shifting slowly to creating and using a process which will allow us to progress as far as possible within our development and containerization time window, as well as enable future development teams to progress with fewer difficulties from the previous architecture.

X. Software Testing Implementation

Our software testing automation is implemented using Travis CI (continuous integration) along with scripts written in Python3. The testing script is dependent on the use of NumPy for data parsing. In this section I will explain both the architecture behind our testing, as well as some of the more intricate details of the testing process.

Our software test follows a relatively simple process going through the following steps:

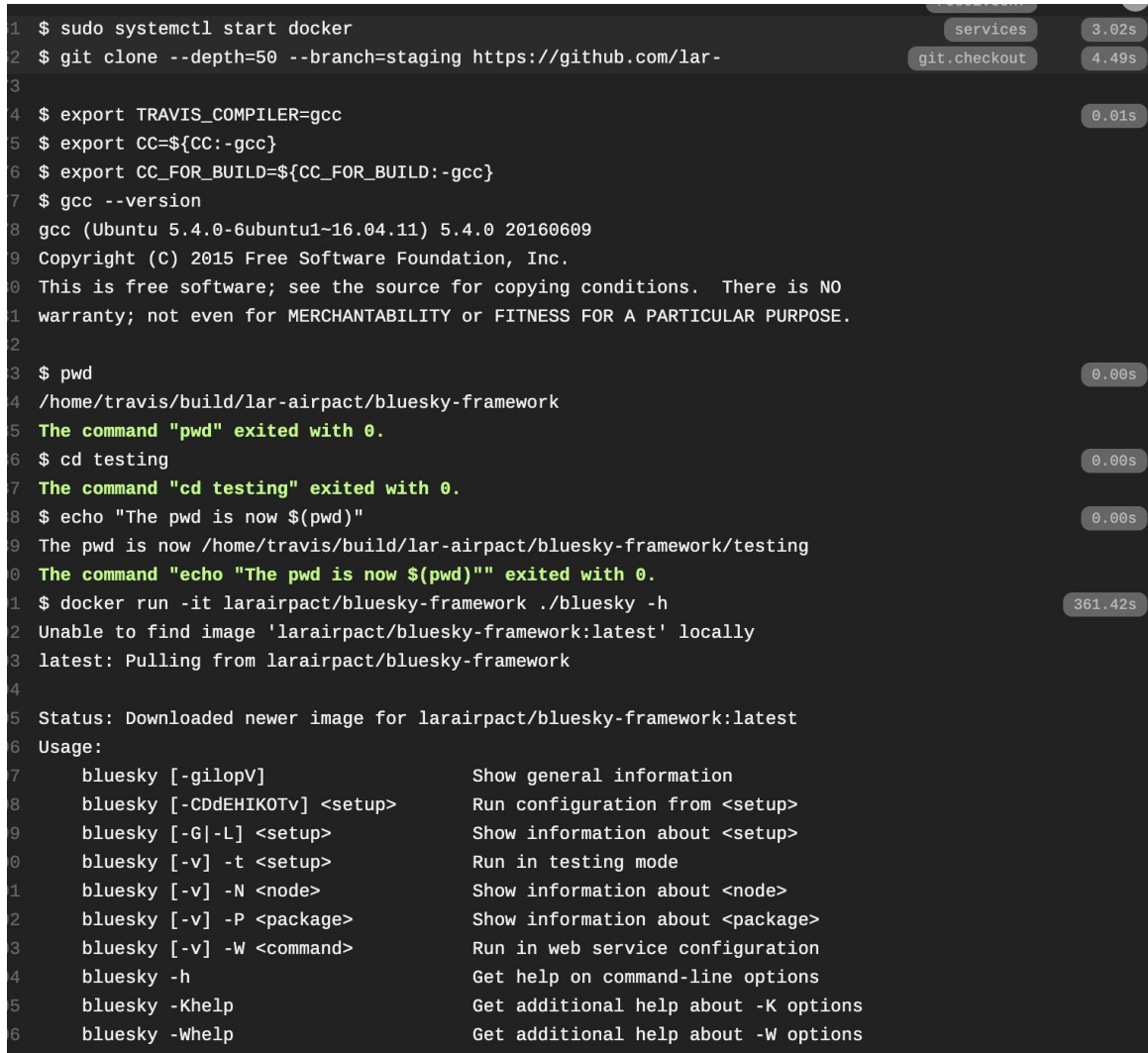
1. Ensure the the BlueSky Framework image builds/runs by pulling the image and running the help command
2. Test that the image under test can complete the run of BlueSky Framework to output data
3. Using the python script, parse the known data and the new data into sets
 - File format is csv
 - Each line of the csv has multiple fields such as an ID number, date, time, etc
 - Each line is treated as a unit of output, which is inserted into a python set
 - Python sets have no duplicates
4. Assert that the sets generated from the given output and generated output are 100% equal.

If all the following conditions are met, the test run is considered a success, and the code under test can be merged into desired branches with approval to forward the development process.

XI. Software Testing Results

Results from testing have been very promising. The tests have successfully verified that the BlueSky Framework docker image produces identical output to the current BlueSky Framework system in place on Aeolus. Below are figures which demonstrate various steps in the testing process of a successful run.

Figure 3: Verifying BlueSky Framework build runs



```
1 $ sudo systemctl start docker
2 $ git clone --depth=50 --branch=staging https://github.com/lar-
3
4 $ export TRAVIS_COMPILER=gcc
5 $ export CC=${CC:-gcc}
6 $ export CC_FOR_BUILD=${CC_FOR_BUILD:-gcc}
7 $ gcc --version
8 gcc (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609
9 Copyright (C) 2015 Free Software Foundation, Inc.
10 This is free software; see the source for copying conditions. There is NO
11 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
12
13 $ pwd
14 /home/travis/build/lar-airpact/bluesky-framework
15 The command "pwd" exited with 0.
16 $ cd testing
17 The command "cd testing" exited with 0.
18 $ echo "The pwd is now $(pwd)"
19 The pwd is now /home/travis/build/lar-airpact/bluesky-framework/testing
20 The command "echo 'The pwd is now $(pwd)'" exited with 0.
21 $ docker run -it larairpact/bluesky-framework ./bluesky -h
22 Unable to find image 'larairpact/bluesky-framework:latest' locally
23 latest: Pulling from larairpact/bluesky-framework
24
25 Status: Downloaded newer image for larairpact/bluesky-framework:latest
26 Usage:
27     bluesky [-gilopV]                Show general information
28     bluesky [-CDdEHIKOTv] <setup>    Run configuration from <setup>
29     bluesky [-G|-L] <setup>          Show information about <setup>
30     bluesky [-v] -t <setup>          Run in testing mode
31     bluesky [-v] -N <node>           Show information about <node>
32     bluesky [-v] -P <package>        Show information about <package>
33     bluesky [-v] -W <command>        Run in web service configuration
34     bluesky -h                       Get help on command-line options
35     bluesky -Khhelp                  Get additional help about -K options
36     bluesky -Whhelp                  Get additional help about -W options
```

Figure 4: Verifying the data output of BlueSky Framework Docker Image

```

0 WildfireGrowth: Persistence from package BlueSky v3.5.1
1 WildfireGrowth: Persistence model created 599 new fire records
2 PrescribedFireGrowth: Persistence from package BlueSky v3.5.1
3 PrescribedFireGrowth: Persistence model created 3600 new fire records
4 OtherFireGrowth: NoGrowth from package BlueSky v3.5.1
5 ConsolidateGrowth: ConsolidateFires from package BlueSky v3.5.1
6 FCCS: FCCS Fuel Loading Module v2
7 FCCS: Successfully added fuel loadings for 5037 of 6018 input fires
8 ConsumptionSplit: FireTypeSplitter from package BlueSky v3.5.1
9 WildfireConsumption: NoConsumption from package BlueSky v3.5.1
0 PrescribedFireConsumption: NoConsumption from package BlueSky v3.5.1
1 OtherFireConsumption: NoConsumption from package BlueSky v3.5.1
2 ConsolidateConsumption: ConsolidateFires from package BlueSky v3.5.1
3 NoTimeProfile: NoTimeProfile from package BlueSky v3.5.1
4 NoEmissions: NoEmissions from package BlueSky v3.5.1
5 NoPlumeRise: NoPlumeRise from package BlueSky v3.5.1
6 StandardFiles: Writing fire locations to standard format file
7 StandardFiles: Successfully wrote 6018 fire locations
8 BlueSky: Completed in 2 minutes 40.00 seconds
9 The command "docker run -it --mount type=bind,source="$(pwd)",target=/bluesky/dist/bluesky/output/
  larairpact/bluesky-framework ./bluesky -d 2019090100Z -K no-archive defaultLAR_SFonly" exited with 0.
0 $ mv 2019090100.1 newinput 0.01s
1 The command "mv 2019090100.1 newinput" exited with 0.
2 $ echo "testing data output" 0.00s
3 testing data output
4 The command "echo "testing data output"" exited with 0.
5 $ python3 test.py 1.14s
6 ....
7 -----
8 Ran 4 tests in 0.249s
9
0 OK
1 The command "python3 test.py" exited with 0.
2
3
4 Done. Your build exited with 0.

```

Finally, since the end product will be running on the Aeolus HPCC, which only uses Singularity as its containerization technology, we needed to test that running the code from within a Singularity Image also worked. The main unit which runs all of the code for this: “BSF_EFO_AP5_SFonly.csh” script. Its output and behavior was manually verified to meet expectations. Within the script itself, the LAR Group calls “ls -al \$TARGET_DIR” as a final command to verify that the data was correctly placed. The “ls” command lists the files in the specified directory, while the “la” options ensure all files are listed in a verbose manner. Lastly, TARGET_DIR is where the LAR Group desired that files were moved to.

In the following figures, we can see the output from running the “BSF_EFO_AP5_SFonly.csh” in our Singularity Image on both a local development machine as well as the Aeolus HPCC. The expected output is a log file, fire_locations CSV file, fire_locations KML file (for the webpage), a

ptday ORL file (for further computation done by AIRPACT-5), a ptinv ORL file (for further computation done by AIRPACT-5), and lastly, a simple summary text file. As seen below, the actual outputs meet our expectations.

Figure 5: Output from BlueSky Framework Singularity Image on Local Dev Machine

```
kylerlittle@Kylers-MacBook-Air:~/Desktop/Projects/bluesky-framework$ singularity exec --bind /Users/kylerlittle/Desktop/Projects/bluesky-framework/bsf_output_temp/output:/bluesky/dist/bluesky/output --bind /Users/kylerlittle/Desktop/Projects/bluesky-framework/bsf_output_temp/working:/bluesky/dist/bluesky/working --bind /Users/kylerlittle/Desktop/Projects/bluesky-framework/bsf_output_temp/log:/bluesky/dist/bluesky/log --bind /Users/kylerlittle/Desktop/Projects/bluesky-framework/bsf_output_temp/conversion/output:/bluesky/dist/bluesky/conversion/output bluesky-framework_latest.sif /bluesky/dist/bluesky/BSF_EF0_AP5_SFonly.csh 20190901
total 3540
drwxr-xr-x 1 kylerlittle dialout 256 Oct 11 15:01 .
drwxr-xr-x 1 kylerlittle dialout 128 Oct 7 03:56 ..
-rw-r--r-- 1 kylerlittle dialout 29 Oct 11 15:01 bluesky_job.log
-rw-r--r-- 1 kylerlittle dialout 2436215 Oct 11 15:01 fire_locations.csv
-rw-r--r-- 1 kylerlittle dialout 227896 Oct 11 15:01 fire_locations_20190901.kml
-rw-r--r-- 1 kylerlittle dialout 828627 Oct 11 15:01 ptday-2019090100.orl
-rw-r--r-- 1 kylerlittle dialout 127805 Oct 11 15:01 ptinv-2019090100.orl
-rw-r--r-- 1 kylerlittle dialout 847 Oct 11 15:01 summary.txt
END OF SCRIPT /bluesky/dist/bluesky/BSF_EF0_AP5_SFonly.csh
[ 166.431518] reboot: Power down
```

Figure 6: Output from BlueSky Framework Singularity Image on Aeolus HPC

```
[[klitte@aeolus demo]$ singularity exec --bind /home/klitte/demo/bsf_output_temp/output:/bluesky/dist/bluesky/output --bind /home/klitte/demo/bsf_output_temp/working:/bluesky/dist/bluesky/working --bind /home/klitte/demo/bsf_output_temp/log:/bluesky/dist/bluesky/log --bind /home/klitte/demo/bsf_output_temp/conversion/output:/bluesky/dist/bluesky/conversion/output bluesky-framework_latest.sif /bluesky/dist/bluesky/BSF_EF0_AP5_SFonly.csh 20190901
total 3580
drwxr-xr-x 1 klitte lar 222 Oct 11 09:18 .
drwxr-xr-x 1 klitte lar 16 Oct 7 17:15 ..
-rw-r--r-- 1 klitte lar 25108 Oct 11 09:18 bluesky_job.log
-rw-r--r-- 1 klitte lar 2436215 Oct 11 09:18 fire_locations.csv
-rw-r--r-- 1 klitte lar 227896 Oct 11 09:18 fire_locations_20190901.kml
-rw-r--r-- 1 klitte lar 828627 Oct 11 09:18 ptday-2019090100.orl
-rw-r--r-- 1 klitte lar 127805 Oct 11 09:18 ptinv-2019090100.orl
-rw-r--r-- 1 klitte lar 847 Oct 11 09:18 summary.txt
END OF SCRIPT /bluesky/dist/bluesky/BSF_EF0_AP5_SFonly.csh
[[klitte@aeolus demo]$
```

XII. Summary of the State of This Project

The goal of this project was to use containerization technology in order to improve the architecture and portability of AIRPACT-5, allowing the use of systems such as the Kamiak HPC. In order to achieve this, work began on a subsystem of AIRPACT-5, named BlueSky Framework (BSF). BSF is no longer supported by the developers at PNWAirfire. Support for the software has been shifted to a new service, named “BlueSky”, which supports piping and redirection. Due to the current structure of AIRPACT-5, migration to the new BlueSky was not performed. Instead the current BlueSky Framework, version 3.5.1, was used to best mimic the current system in use.

Due to the nature of this project, and the lack of a version control system, a GitHub repository for lar-airpact was created, and forks of the BlueSky, BlueSky-Framework, and CMAQ development branches were created alongside a separate airpact5-senior-design repository. Furthermore, Continuous Integration & Development pipelines have been created using TravisCI. When a commit or pull request is made, the system checks and validates that the changes to the edited files do not break the system. Upon passing these tests, a pull request can be approved and then merged into the staging branch. The master branch serves to hold full releases, and should only be merged into when a new version is to be released. Along with Github, DockerHub and SingularityHub repositories were set up, allowing the latest

containerized images to be pulled when AIRPACT-5 runs BSF. This allows for greater maintainability and porting of code changes.

AIRPACT-5 uses many ambiguous environment variables and contains circular dependencies. For instance, the BlueSky Framework writes its output to a directory that exists on the caller's system (Aeolus). In general, a child system shouldn't know about its caller's system, as this creates a circular dependency. This significant coupling is present in many of AIRPACT-5's components because components write their data to where the subsequent system expects. Any further development should aim to decouple these interdependencies.

In order to facilitate further development, it is our recommendation that forks are made of all of the AIRPACT-5 subsystems that are available on GitHub into the lar-airpact organization. Further, we recommend that a team of software engineers investigate this system thoroughly, and develop an overall hierarchy, as well as a dependency tree of the script calls and system organization. Following this, determining which systems have new developments, or have been deprecated and understanding the version differences will be required. Finally, a thorough cleanup of the AIRPACT-5 directory on Aeolus will be needed.

In conclusion, our team has successfully containerized BSF v3.5.1 using Singularity and Docker, and have verified the running of the system on the Aeolus HPCC. A CI/CD pipeline has been set up, such that the output of any changes must be verified to match expected output, prior to merging to the staging branch. Scripts have been modified within Aeolus to pull the latest BSF image from SingularityHub and run the singularity container. This successfully replaces the need for a dedicated Ubuntu system and allows for easier portability and platform independence in the future.

References

1. <http://lar.wsu.edu/airpact/>
2. <https://singularity.lbl.gov/user-guide>
3. <https://docs.docker.com/>
4. <https://www.git-scm.com/doc>
5. <https://guides.github.com/>
6. <https://singularityhub.github.io/singularityhub-docs/>
7. <https://docs.docker.com/docker-hub/>
8. <https://docs.travis-ci.com/>