“Improving the Architecture & Development Process for AIRPACT: Testing Document”

Mentor: Joe Vaughan

Instructor: Aaron Crandall

Course: Computer Science 423

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

Date: 10/19/2019

Testing Plan

Testing Granularity

> Unit Tests

Unit	**Description**	**Expected Output / Result**	**Passed / Approved By**
BlueSky Framework Binary (bluesky)	Running the BlueSky Binary	Identical output files (fire_locations.csv, fire_events.csv) + similar summary.txt files as those produced by existing Ubuntu Server	Passed
fire_ptday_SFonly_Zhang.py	Running python script converts BlueSky fire info (fire_locations.csv) into ORL ptday SMOKE files for CMAQ inline fire plume rise	Produces ORL ptday file.	Approved by Kyler Little
fire_ptinv_SFonly_Zhang.py	Running python script converts BlueSky fire info (fire_locations.csv) into ORL ptday SMOKE files for CMAQ inline fire plume rise	Produces ORL ptinv file.	Approved by Kyler Little
write_kml.py	Running python script converts BlueSky fire info into KML for AIRPACT5 webpage	Produces fire_locations_YYYYMMDD.kml file to be displayed on AIRPACT5 webpage	Approved by Kyler Little

> Integration Tests

BSF_EFO_AP5_SFonly.csh	Run BlueSky SMARTFIRE for EmissionFilesOnly for AIRPACT-5 Date	Runs the above 4 scripts successfully, transfers output files to “TARGET_DIR = ~/AIRHOME/run_ap5_day1/emis/fire_orl/transfer”, deletes the BlueSky Framework output files	Approved by Kyler Little

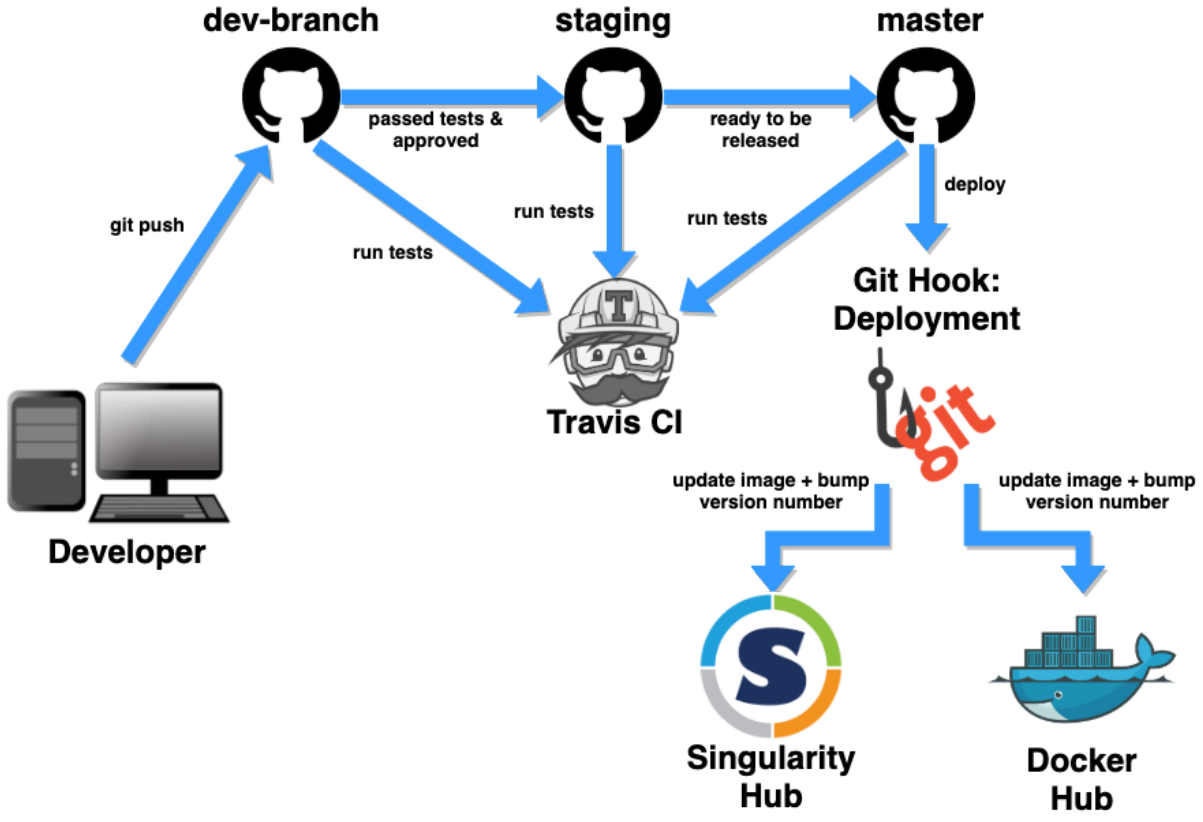
Testing Architecture

The testing architecture for this project can be viewed in Figure 1 below. 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.

The testing process itself runs our containerized Bluesky Framework System and compares the output data from the container to the output data from the Bluesky Framework System that is currently running as an Ubuntu Server on the Aeolus High Performance Computing Cluster. These tests are run automatically any time that commits are pushed to a branch or pull requests are made against another branch. This ensures that all changes do not affect the output of the system when moved into the production environment.

A visual of the entire process is displayed below:

Figure 1: Visual of Testing Architecture



Testing Implementation Details

The automated testing, performed on the “BlueSky Framework Binary” unit, is done programmatically through a Python 3 based script. This script has a dependency on numpy for parsing and populating data. Testing is primarily done through the following process.

1. Parse the saved output data (which we will verify against) into a set.
2. Parse the new output data from the image under test into a set.
3. Assert that these tests are equal, therefore saying the data is equivalent

Because the testing checks for perfect equivalency of files, we are able to ensure that our Docker & Singularity Images represent perfect replicas of the existing Bluesky Framework System running on the Ubuntu Server in the Aeolus HPCC.

Testing Results

Through automated testing, we were able to verify that the output of the containerized Bluesky Framework correctly matched the data output of the in-place Bluesky Framework system being hosted on Aeolus. This test now serves as the primary metric against which all future development on the Bluesky Framework system is done. This specific test is crucial because everything else is dependent on this key job. Specifically, once it's been determined that this output is the same, every subsequent unit's work (e.g. the three python scripts) is guaranteed to match what we expect.

In the following figures are various displays of the process of testing and the output during/finishing the test run.

Figure 2: Verifying Bluesky Framework build runs

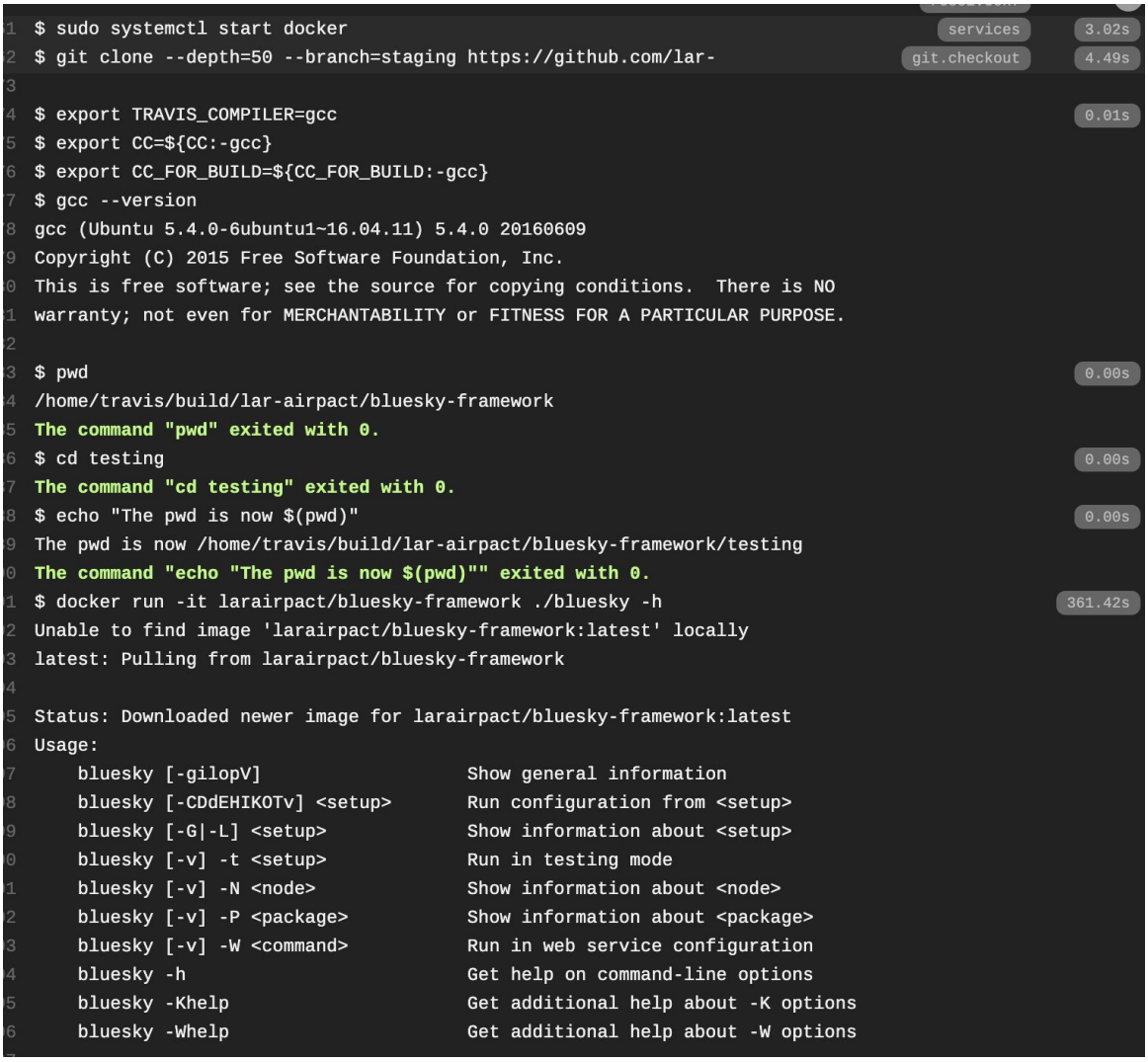


Figure 3: Verifying the data output of Bluesky Framework Docker Image

```
WildfireGrowth: Persistence from package BlueSky v3.5.1
WildfireGrowth: Persistence model created 599 new fire records
PrescribedFireGrowth: Persistence from package BlueSky v3.5.1
PrescribedFireGrowth: Persistence model created 3600 new fire records
OtherFireGrowth: NoGrowth from package BlueSky v3.5.1
ConsolidateGrowth: ConsolidateFires from package BlueSky v3.5.1
FCCS: FCCS Fuel Loading Module v2
FCCS: Successfully added fuel loadings for 5037 of 6018 input fires
ConsumptionSplit: FireTypeSplitter from package BlueSky v3.5.1
WildfireConsumption: NoConsumption from package BlueSky v3.5.1
PrescribedFireConsumption: NoConsumption from package BlueSky v3.5.1
OtherFireConsumption: NoConsumption from package BlueSky v3.5.1
ConsolidateConsumption: ConsolidateFires from package BlueSky v3.5.1
NoTimeProfile: NoTimeProfile from package BlueSky v3.5.1
NoEmissions: NoEmissions from package BlueSky v3.5.1
NoPlumeRise: NoPlumeRise from package BlueSky v3.5.1
StandardFiles: Writing fire locations to standard format file
StandardFiles: Successfully wrote 6018 fire locations
BlueSky: Completed in 2 minutes 40.00 seconds
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.
$ mv 2019090100.1 newinput                                                    0.01s
The command "mv 2019090100.1 newinput" exited with 0.
$ echo "testing data output"                                                  0.00s
testing data output
The command "echo "testing data output"" exited with 0.
$ python3 test.py                                                             1.14s
....
----------------------------------------------------------------------
Ran 4 tests in 0.249s

OK
The command "python3 test.py" exited with 0.


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

Figure 4: Output from Bluesky Framework Singularity Image on Local Dev Machine

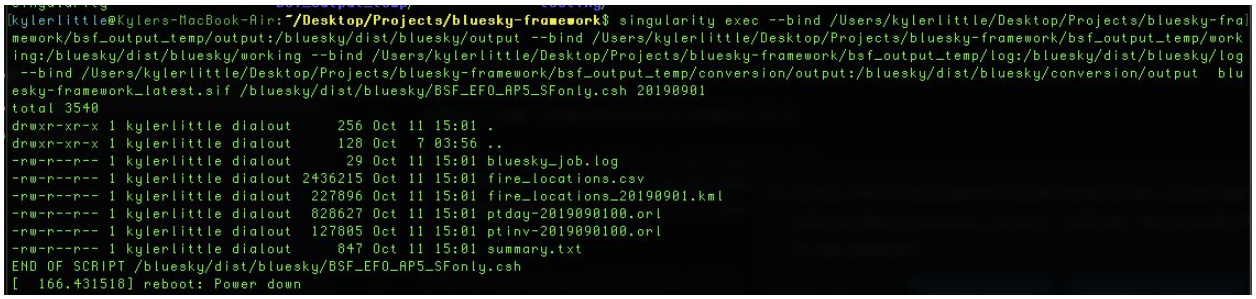


Figure 5: Output from Bluesky Framework Singularity Image on Aeolus HPCC