

Application-Level Regression Testing Framework using Jenkins

Author1, Timothy Bouvet, Galen Arnold
National Institute for Computational Sciences
The University of Tennessee, Knoxville, TN 37996
{email1,tbouvet@illinois.edu,gwarnold@illinois.edu}@domain.edu

Abstract—This paper will explore the challenges of regression testing and monitoring of large scale systems such as NCSAs Blue Waters. Our goal was to come up with an automated solution for running user-level regression tests to evaluate system usability and performance. Another requirement was to find an automated solution for running a suite of test jobs that reveal the system health after upgrades and outages before returning the system to service. We evaluated test frameworks such as Inca [1] and Jenkins [2]. Jenkins was chosen for its versatility, large user base, and multitudes of plugins including plotting test results over time. We utilize these plots to track trends and alert us to system-level issues before they are reported by our partners (users). Not only does Jenkins have the ability to store historical data but it can also send customized notifications (e.g. send email or text pages) based on the result of a test. Some of the requirements we had include two-factor authentication to access Jenkins GUI with privileges to execute tests and account management through LDAP. In this paper we describe our implementation of these requirements to ensure a secure and usable deployment of a Jenkins instance.

Our Jenkins instance was deployed on a vSphere managed VM running Centos 6.8. Security was of the highest concern since Jenkins has the ability to execute commands on our login nodes. Jenkins is set to log in as a standard user account with passwordless access (commands to the login nodes can be input via Jenkins GUI) using ssh keys scoped to the VM. The VM (bwjenkins) was closely vetted by our security team on our test and development system (TDS) before it was allowed access to Blue Waters. Iptables was used to lock bwjenkins down to a small internal ip space to ensure the highest security. An anonymous user account was enabled with read-only access to view current and historical test results. During a test, Jenkins downloads software, builds the software, submits a job, awaits completion, gathers and plots the results or returns error information during a failure. These are full end to end functionality testing of the programming environment, queueing system, license servers and provide historical metrics to quickly detect regressions.

In this paper we describe in detail our challenges and experiences in deploying Jenkins as a user-level system-wide regression testing and monitoring framework for Blue Waters. We will also show some application-based system-level tests we have implemented in Jenkins and share results of those tests. The deployment of Jenkins allow us to monitor and detect issues as early as possible from the perspective of a user on Blue Waters, eventually providing a more efficient service for better user experience.

Keywords—Performance; Benchmarking; Computer architecture

Listing 1: Apache Configuration Example

```
# First, we configure the "default" to be a very
# restrictive set of features.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

I. MOTIVATION

II. JENKINS CONFIGURATION

- A. Introduction to Jenkins
- B. Master-Node Configuration
- C. Accessing Login Nodes

III. AUTHENTICATION AND AUTHORIZATION

- A. Security Considerations
- B. Configuration Choices

IV. ANATOMY OF A TEST

V. SWTOOLS INTEGRATION

VI. USE CASES

- A. IOR Lustre scratch test

We created Jenkins tests to run the IOR MPI parallel filesystem test to run at a modest scale and validate filesystem functionality and performance for the scratch filesystem on a periodic basis. The test is scaled to provide representative performance numbers for a typical small application while not creating issues with other jobs or the filesystem at large. One of our best practices is to provide a management level summary description for casual jenkins users who want to view the test but do not need to understand the jenkins configuration and execution details.

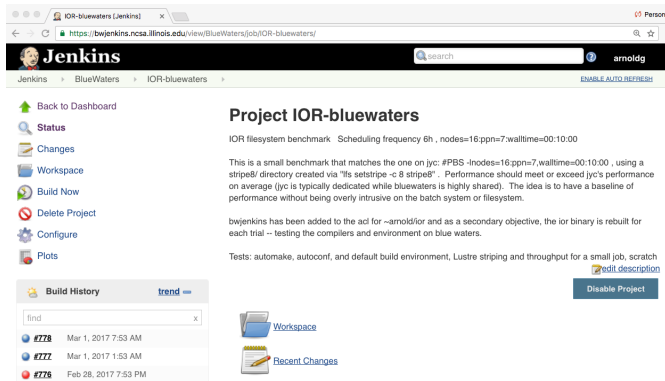


Figure 1: IOR description in jenkins

Where possible, tests build from source and exercise multiple user-facing system components such as: git for external connectivity and loading modules to test the defaults in the environment. Where scripts on the SSH site are used, they are echoed and/or displayed with "cat -n" to make console output verbose and easy to debug in case of errors.

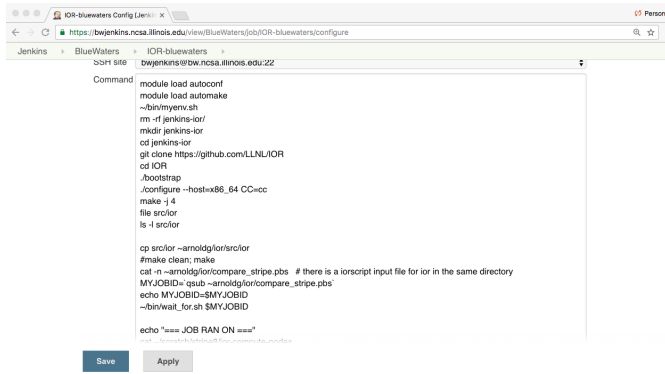


Figure 2: IOR configuration sample

The test is set to run as scheduled by Jenkins with a best-effort through our batch system. The test blocks such that the next test will not start until the previous one has completed. We employ a watchdog script to block the exit from the test until the batch system marks it as finished.

Listing 2: pbs/torque watchdog script

```
#!/bin/bash
echo "===_RUNNING_$0_==="
if [ $# -lt 1 ]
then
    echo "$0: missing argument for _jobid_"
    exit 1
fi
while true
do
    MYQSTAT=`qstat $1`
    if test "$MYQSTAT" = ""
    then
        echo $1 finished
        exit
    else
        DATE=`date`
        echo "$DATE: waiting for $1 to finish"
    fi
    sleep 5m
done
```

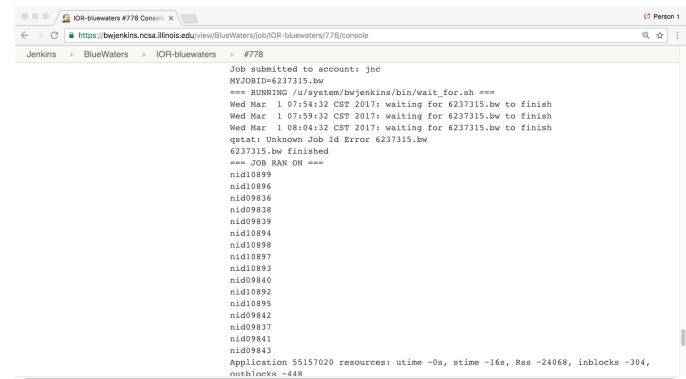


Figure 3: IOR watchdog console output

For tests that produce an interesting performance metric like IOR, we save that output to a file and use the Jenkins plotting feature to produce plots.

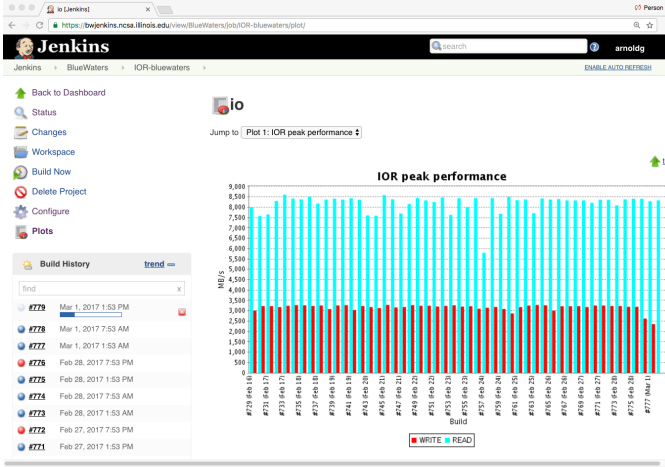


Figure 4: IOR plot view

A plot like this helps us with performance baselines and we can quickly react to changes in the software environment or filesystem configuration.

B. mdtest Lustre tests

The mdtest metadata test is run on the home and scratch filesystems to validate metadata server performance. The IOR and mdtest user-facing tests complement the fine-grained detail we are able to glean from the backed system-side counters in our OVIS database where we record details about individual server metrics and network traffic on the Cray Gemini fabric.

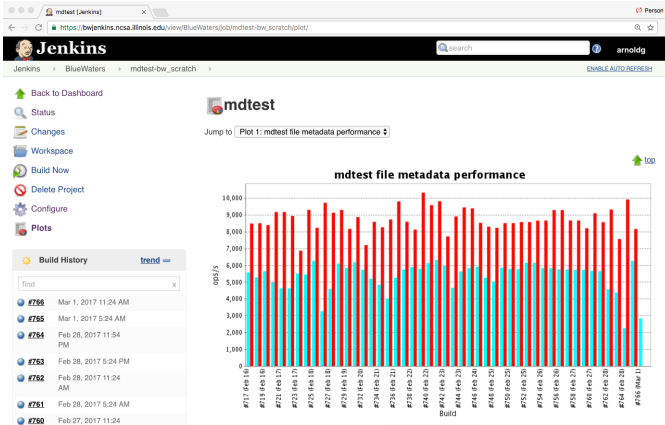


Figure 5: mdtest plot view

An additional best practice is to enable E-mail Notifications to the test author and/or other interested parties. E-mail for unstable builds will cause an E-mail every time the test state changes (from successful to failing and again when back to successful).

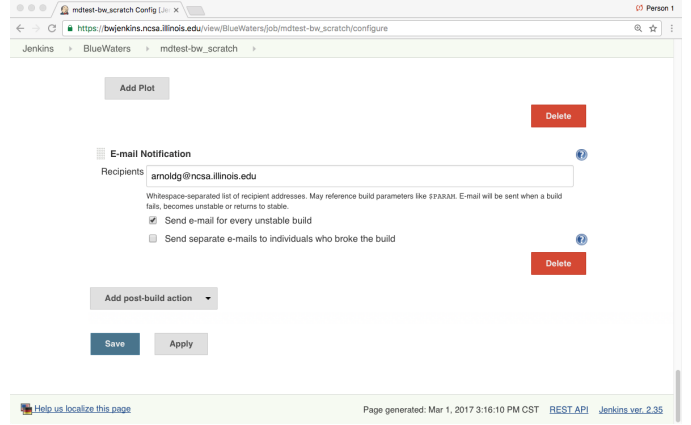


Figure 6: mdtest email configuration

C. special projects

The tabs display of our Jenkins instance is used to group tests and we add tabs for special projects. For example we clone existing tests for evaluating a new software stack on our test login node. Our test rack has a separate set of tests from our production system and is used as our Jenkins test development area. The sustained petascale performance benchmarks are in their own tab since they can run at scale and are only run on-demand (not scheduled by Jenkins).

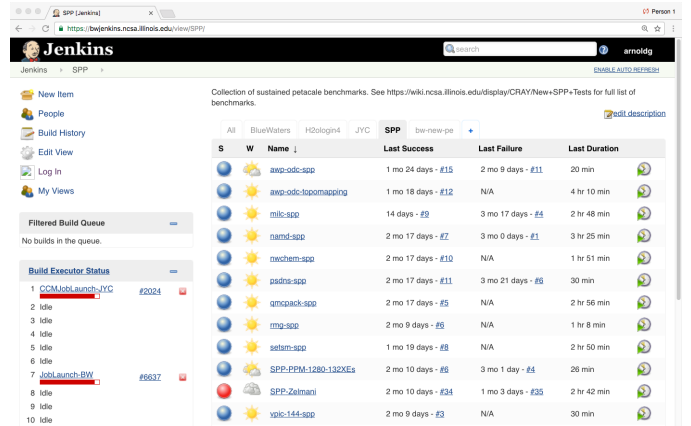


Figure 7: tabs for project areas

VII. CONCLUSION

ACKNOWLEDGMENT

This research used resources at the National Institute for Computational Sciences, funded by the National Science Foundation (NSF).