# Application-Level Regression Testing Framework using Jenkins

Timothy Bouvet, Reuben Budiardja, Galen Arnold
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign, 61801
National Institute for Computational Sciences
The University of Tennessee, Knoxville, TN 37996
{tbouvet@illinois.edu,reubendb@utk.edu,gwarnold@illinois.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 multitude 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 password-less 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 tests of the programming environment, queueing system, license servers and they 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 allows 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**

## I. Motivation

## II. Jenkins Configuration

### A. Introduction to Jenkins

### B. Master-Node Configuration

### C. Accessing Login Nodes

## III. Authentication and Authorization

### A. Security Considerations

Enabling Jenkins access to Blue Waters posed some unique security hurdles that we had to overcome. Blue Waters is an OTP only restricted access system. Jenkins has the ability to execute user-level commands on our login nodes. Jenkins has some unfortunate default settings. The access control is set to "Jenkins' own user database," and the "Allow users to sign up" option is enabled, so if you don't have an account, you can create one by providing your name, e-mail and a password. Additionally, there's an authorization option enabled which says "Logged-in users can do anything." So, once you complete the sign up form, you have full admin access to the system. This required us initially to test the deployment with our test and development (TDS) CRAY. This resulted in the following requirements. Restricted OTP access to the physical system that will host Jenkins. In addition, OTP only access to the Jenkins GUI command console that must be contained within our IP space. The Jenkins GUI will also restrict the command console access to our Blue Waters LDAP server.

### B. Configuration Choices

Jenkins instance was deployed on a vSphere managed VM running Centos 6.8. The VM allows restricted host-based ssh access from two secure OTP only administrative systems. The Jenkins GUI presented additional challenges. The Jenkins GUI is configured to log in as a standard user account with pass-wordless access (commands to the login nodes can be input via Jenkins GUI) using ssh keys scoped to the VM. We restricting access to the physical VM and Jenkins GUI (bwjenkins) utilizing iptables, SSL and OTP. A reverse proxy was used to run Jenkins behind apache/http

with SSL for access to the administrative console. All the configuration is in /etc/httpd/config.d/ssl.conf. Anonymous access to the Jenkins GUI was allowed for limited read only view of test status and historical plots. The Anonymous access view is not restricted.
(config file snippets below)

```
/etc/ssh/sshd_config:
Match Host bwbh1.ncsa.illinois.edu,
bwbh2.ncsa.illinois.edu
HostbasedAuthentication yes

/etc/sysconfig/iptables:
-A INPUT -s 141.142.0.0/16 -m state --state NEW -m tcp
-p tcp --dport 22 -j ACCEPT
-A INPUT -s 172.16.0.0/15 -m state --state NEW -m tcp
-p tcp --dport 22 -j ACCEPT *delete maybe*
-A INPUT  -m state --state NEW -m tcp -p tcp --dport 80
-j ACCEPT
-A INPUT -s 141.142.0.0/16 -m state --state NEW -m tcp
-p tcp --dport 443 -j ACCEPT

/etc/httpd/config.d/ssl.conf:
Listen 443
<VirtualHost _default_:443>
SSLSessionCache         shmcb:/var/cache/mod_ssl/scache
SSLSessionCacheTimeout  300
ServerName bwjenkins.ncsa.illinois.edu:443
 # -- The following sets ReverseProxy for Jenkins
  ProxyRequests       Off
  ProxyPreserveHost   On
  AllowEncodedSlashes On
  <Proxy *>
    Order deny,allow
    Allow from all
  </Proxy>
  ProxyPass        / http://localhost:8080/ nocanon
  ProxyPassReverse / http://localhost:8080/
  ProxyPassReverse / http://bwjenkins.ncsa.illinois.edu/
  RequestHeader set X-Forwarded-Proto "https"
  RequestHeader set X-Forwarded-Port "443"

  DefineExternalAuth pwauth+session pipe /usr/local/bin/
  pwauth+session
  <LocationMatch / >
    AuthType Basic
    AuthBasicProvider external
    AuthName "NCSA_RSA_OTP_login"
    AuthExternal pwauth+session
    require valid-user

    RequestHeader unset "X-Forwarded-User"
    RequestHeader set "X-Forwarded-User" (REMOTE_USER)

  </LocationMatch>
```

### C. LDAP with Jenkins

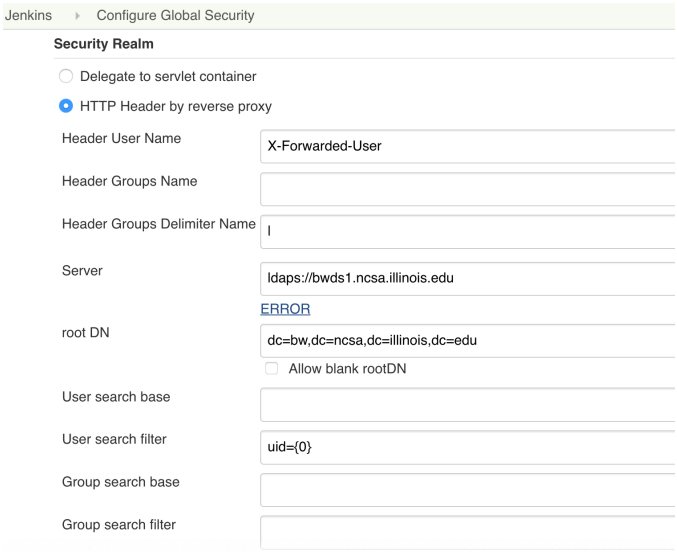


Figure 1: Use LDAP with Jenkins

From the console view, select Manage Jenkins, select Configure Global Security. In the Security Realm section select the Advanced... tab. Input your LDAP server name and root DN. Click the Test Button.

My first stab at this failed as an anonymous connection to our LDAP because we are using a local CA. WARNING: Failed to search LDAP for username, sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target

Had to add our certificate to the keystore: /usr/java/default/jre/bin/keytool -list -keystore cacerts -imle /etc/openldap/cacerts/cacert.pem Enter keystore password: Trust this certificate? [no]: yes
Certificate was added to keystore

### D. Configure Global Security

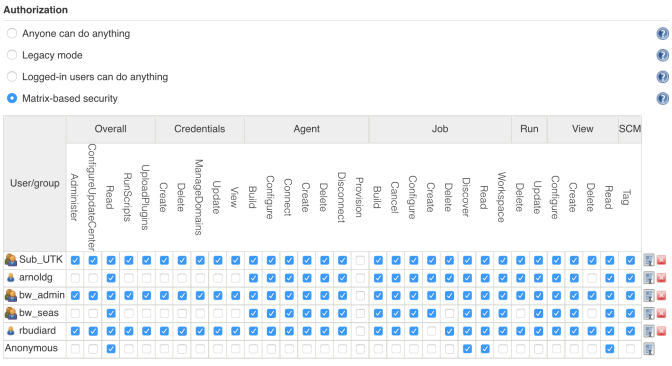


Figure 2: Configure Global Security

We are allowing access by username and LDAP groups. The permissions can be scoped for individuals and groups. An example is removing Job delete permissions to prevent one user or group from deleting others jobs.

## IV. Anatomy of A Test

## V. SWTools Integration

## VI. Use Cases

### A. IOR Lustre scratch test

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

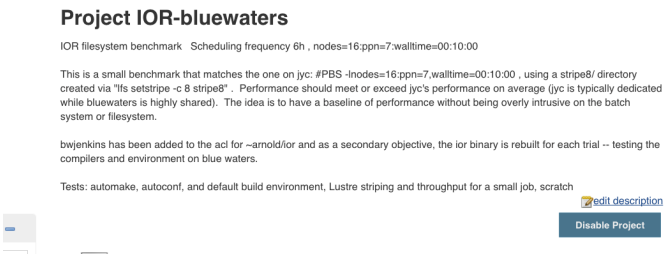


Figure 3: IOR description in jenkins

Where possible, tests build from source and exercise multiple user-facing system components such as: git for external connectivity and modules to test defaults in the environment. Where scripts on the remote 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.

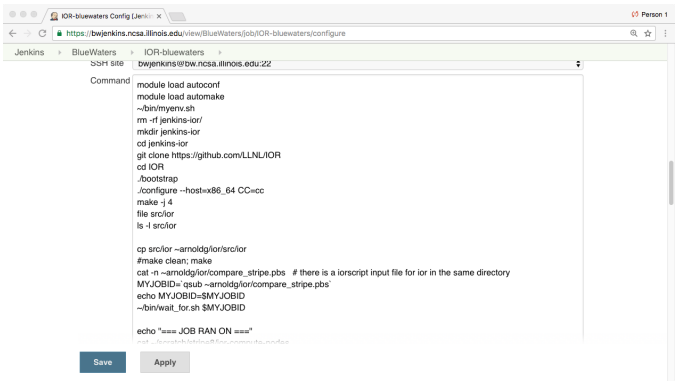


Figure 4: IOR configuration sample

The IOR test is set to run as scheduled by Jenkins with a best-effort through our batch system. The test synchronously blocks such that the next test will not start until the previous one has completed. We employ a watchdog script to monitor the batch queue and keep the test active until the system marks it as finished. This has the side effect of making the minimum test time an integer multiple of the sleep time in the watchdog script. This may be adjusted to fit individual site needs.

Listing 1: 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
```

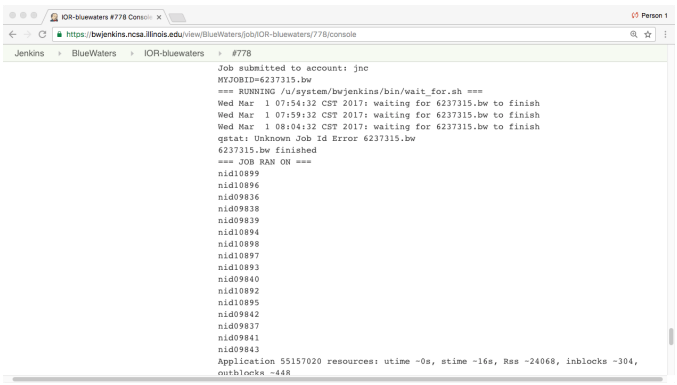


Figure 5: IOR watchdog console output

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

Listing 2: sample YVALUE output file

```
bwjenkins$ cat myiorREADnumber.dat
YVALUE=8352.71
```

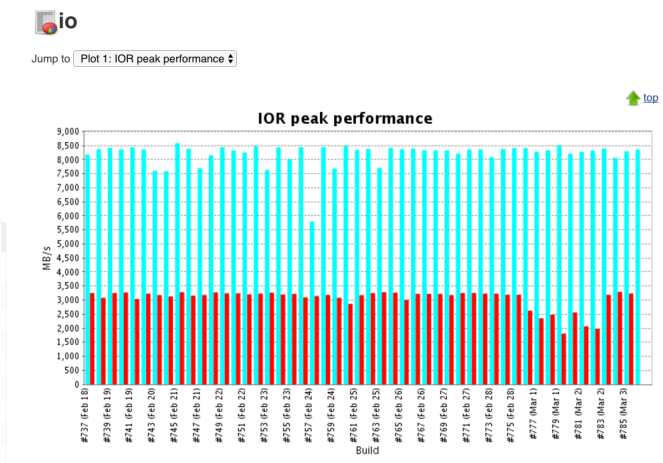


Figure 6: IOR plot view

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

### 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.

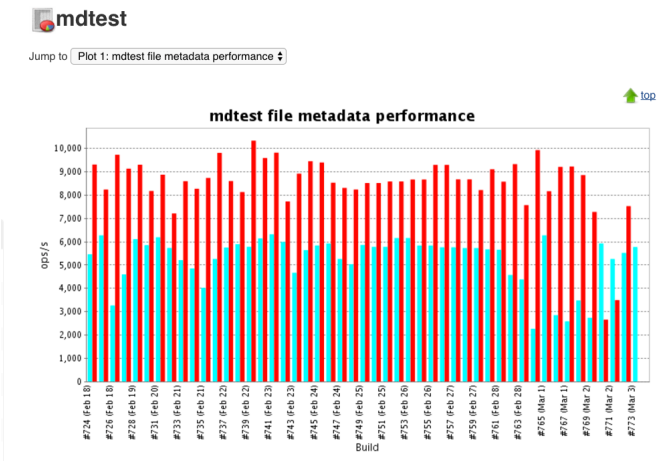


Figure 7: mdtest plot view

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

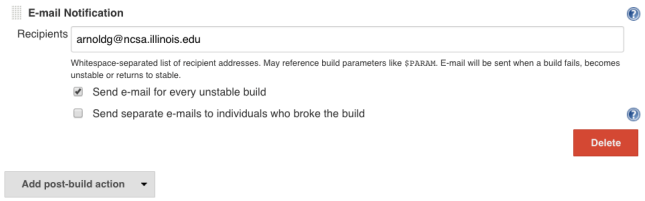


Figure 8: mdtest email configuration

### C. *filesystem dashboard*

Our Jenkins instance is a source of information for a filesystem dashboard display we maintain in our wiki. The ”http://” urls for the plot images are shown on the dashboard and provide a recent-time display of filesystem metrics from a user point of view. We add a Jenkins test to track response time of /bin/ls on the login nodes–a reasonable metric of metadata server health and also the most common problem reported by users when we are having filesystem issues.

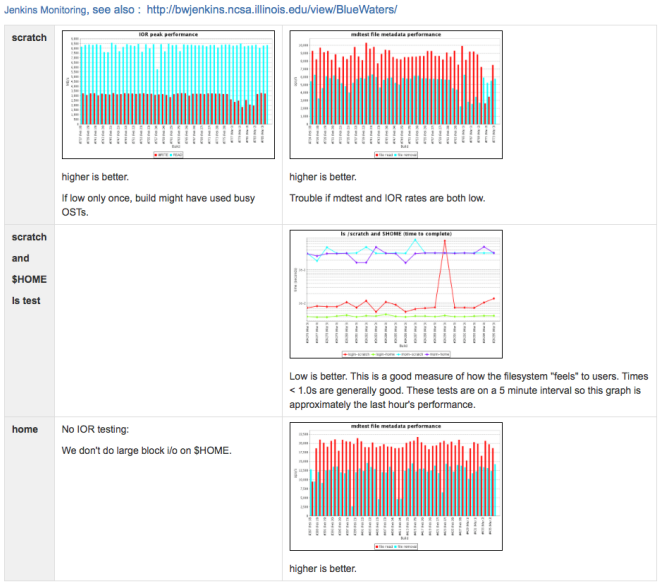


Figure 9: filesystem dashboard displaying multiple Jenkins plots

### D. *special projects*

The initial display tabs in our Jenkins instance are 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 development mule. The sustained petascale performance benchmarks are in their own tab because they run at scale and are only run on-demand by staff (not scheduled by Jenkins ).

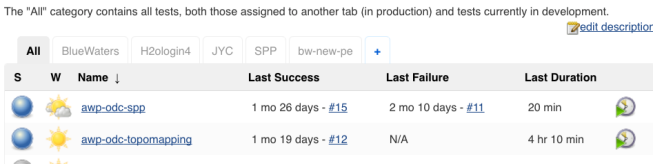


Figure 10: tabs for project areas

## VII. Conclusion


### Acknowledgment

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