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. 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 email or text pages based on results of a test. Other requirements we had include two-factor authentication for accessing the Jenkins GUI with administrator privileges and account management through LDAP. In this paper we describe our experience in deploying Jenkins as a user-level system-wide regression testing and monitoring framework for Blue Waters.**

***Keywords*-Performance; Benchmarking; Computer architecture**

***Abstract*—LONG-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.

I. Introduction

Monitoring large and complex high-performance computing systems has many challenges. Layers and multiple versions of software stack, system-level configurations, (parallel) file systems, diversity in users need and usage, and variable network traffics all interact to increase the system's complexity. Although there exist monitoring systems for each of these components, they often do not capture the behavior of the system in aggregate, which in fact is what users and their applications experience. Therefore an application-level system monitoring is needed to give us a more complete picture of the system behavior.

Over time, either due to configuration changes or hardware aging, performance of such complex system may regress. To identify and correct such regression, one must have some method to do regression testing of the system performance. This requires systematic record of the the system performance over time.

These two requirements can be concisely summarized as an "application-level regression testing". Our goal was then to come up with an automated solution for running these user-level regression tests to evaluate system usability and performance. For this purpose we have chosen to use Jenkins[1].

[1] http://jenkins.io

Jenkins is an open source automation server. Although it is typically used as continuous integration tool in software development process, its wide-ranging features makes it suitable for any task that can be automated. These tasks may include building applications, executing them, deploying software, and running various custom-written tests and scripts. We chose Jenkins as the tool for our regression testing framework because its active development, wide community support, and the readily available "plugins" to perform various specific tasks.

In this paper we describe in details our experience in using Jenkins to perform application-level regression testing on high-performance computing systems such as the Blue Water supercomputer at the National Center for Supercomputing Application, Darter and Beacon at the National Institute for Computational Sciences (NICS). These systems are Cray XK, XC, and CS series. The outline of this paper is as follows. [NEED TO WRITE THIS]

II. Jenkins Configuration

A. Introduction to Jenkins

Jenkins [2] is a self-contained, open source automation server which can be used to automate all sorts of tasks such as building, testing, and deploying software. Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with the Java Runtime Environment installed. To install from rpm as root for centos:

```
wget -O /etc/yum.repos.d/jenkins.repo \
  http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo
rpm --import \
  https://jenkins-ci.org/redhat-stable/jenkins-ci.org.key
yum install jenkins
```

B. Master-Node Configuration

We added a standard user account to our cluster that will execute the Jenkins tests. Unique ssh keys were generated on the Jenkins Server for accessing the user account on production and test environments. The keys on the server are located in Jenkins home directory (/var/lib/jenkins/.ssh). Using the GUI management console, the keyfile location and passphrase was entered under Manage Jenkins, Configure System, SSH remote hosts section. A unique SSH site is configured for our TDS, production and development login nodes.

C. Accessing Login Nodes

The ssh public key generated on Jenkins Server was deployed in the login node user account. The login node sshd_config was modified to allow Public Key Authentication from the Jenkins Server IP.

III. Authentication and Authorization

A. Security Considerations

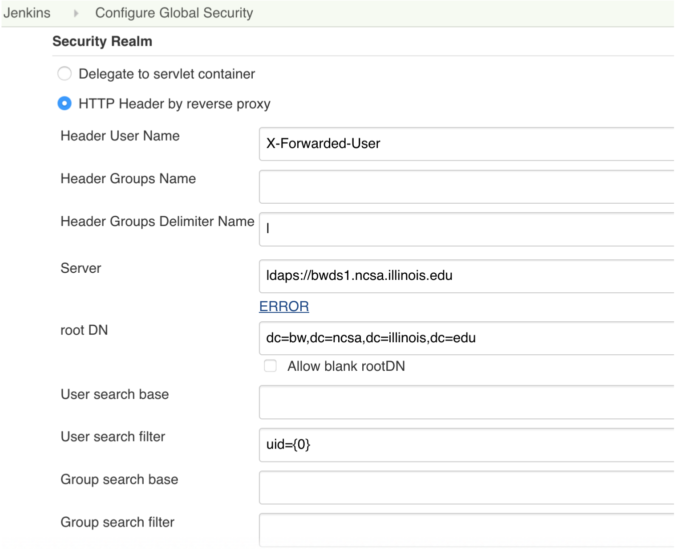
Enabling Jenkins access to Blue Waters posed some unique security hurdles that we had to overcome. Blue Waters is an OTP only restricted access system. Jenkins has the ability to execute user-level commands on our login nodes. Jenkins has some unfortunate default settings. The access control is set to "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 signed in, you have full admin access to the Jenkins system instance. This required us initially to test the deployment with our test and development (TDS) CRAY. This resulted in the following requirement: restricted OTP access to the physical system hosting Jenkins. In addition, OTP only access to the Jenkins GUI command console 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

The 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 password-less access (commands to the login nodes can be input via Jenkins GUI) using ssh keys scoped to the VM. We restrict access to the physical VM (bwjenkins) and the Jenkins GUI 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/conf.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.

```
 (Jenkins sever configuration 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/conf.d/ssl.conf:
 # -- 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"

# -- The following sets OTP for Jenkins SSL connection
 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>

/usr/local/bin/pwauth+session: https://github.com/reuben
BasicAuthOTPSession/blob/master/BasicAuthOTPSession.py#L
```

C. LDAP with Jenkins



Jenkins ▸ Configure Global Security

Security Realm

○ Delegate to servlet container

● HTTP Header by reverse proxy

Header User Name: X-Forwarded-User

Header Groups Name:

Header Groups Delimiter Name: |

Server: ldaps://bwds1.ncsa.illinois.edu

ERROR

root DN: dc=bw,dc=ncsa,dc=illinois,dc=edu

□ Allow blank rootDN

User search base:

User search filter: uid={0}

Group search base:

Group search filter:

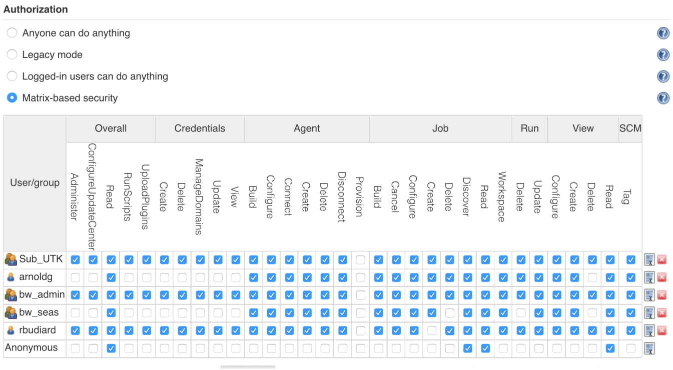
Figure 1: Use LDAP with Jenkins

To enable LDAP, the LDAP Plugin was installed. To configure, from the console view, select Manage Jenkins, then 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

We 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



Authorization

○ Anyone can do anything

○ Legacy mode

○ Logged-in users can do anything

● Matrix-based security

User/group	Overall					Credentials					Agent							Job								Run		View				SCM
	Administer	ConfigureUpdateCenter	Read	RunScripts	UploadPlugins	Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Read	Workspace	Delete	Update	Configure	Create	Delete	Read	Tag
Sub_UTK	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
arnoldg	□	□	☑	□	□	□	□	□	□	□	☑	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	□	☑	☑
bw_admin	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
bw_seas	□	□	☑	□	□	□	□	□	□	□	☑	☑	☑	☑	☑	☑	□	☑	☑	☑	☑	□	☑	☑	☑	□	☑	☑	☑	□	☑	☑
rbudiard	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	□	☑	☑	☑	□	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Anonymous	□	□	☑	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	☑	☑	□	□	□	□	□	□	☑	□

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 each other's jobs.

IV. Anatomy of a Test

New software projects are required to be tested on our TDS and approved before deployment on Blue Waters and this is one of our key best practices. Once a test is deployed it is assigned to a specific Jenkins tab via the ”Edit View” link associated with that tab. Tests still in development reside in the ”All” tab view until they have been peer reviewed by at least one other test developer.

A. test description

We require a description for each test to describe: what is tested, resources used by the test, intent of the test, and scheduling frequency.

B. limits log files

The Jenkins test VM filled to capacity after the first few weeks of intensive use and an influx of several new tests that output long source code build detail. We retroactively modified all tests to limit the number of old build logs retained (max 50 for most tests).

C. ssh to a login host

There are several target ssh hosts which may be selected from our instance: the test rack JYC, the main system, and the new software deployment login node within the main system. None of our tests run locally within the Jenkins VM.

D. run shell commands

A test is composed of shell command and where possible we try to be as transparent with our test construction as possible. We prefer commands in the Jenkins configuration over running a script in the filesystem on the remote ssh target. Scripts that are remote run with verbose mode or are displayed via ”cat -n” to save output to the Jenkins console log and make for simpler debugging when a test fails.

1) test completes on login node: Some tests run only on the ssh login node and return more or less immediately. Examples include tests for batch system functionality, ssh functionality, and filesystem functionality. These tests are very similar in style to unit tests which target a limited set of functionality.

2) test is submitted to batch system: More comprehensive tests typically build and run an application or benchmark through the batch system. These tests exercise multiple aspects of the system: (filesystem, compilers and modules, high-speed network, external connectivity to the internet...). They block to completion so that we do not overburden the batch system with tests. Unlike tests that run entirely on a login node, completion time can be highly variable depending on the queue depth and available backfill windows for the batch system.

E. post test actions

1) scp files back to Jenkins host: Most comprehensive tests produce output that we use when debugging a failed test. They may also produce plot data. These files are copied back to the Jenkins host so that they can be viewed and/or incorporated into the plot feature of Jenkins.

2) plots: Successful tests save plot files and contribute a new data point to a plot. The Jenkins server accumulates the data as rows in a spreadsheet (under /var/lib/jenkins/jobs/) and plots them via the ”Plots” link for a test.

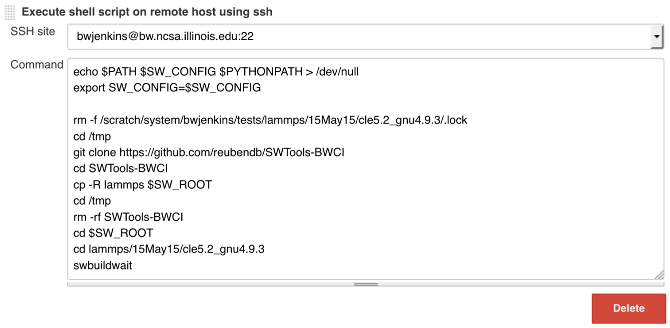
F. email notification

When the state of a test changes, email notification is sent to the test owner. As a best practice we also use the email notification as a method of tracking authorship and each test has an email owner even if notifications are disabled for the test.

V. SWTools Integration

A. SWTools with Jenkins

B. SWTools with Lammps



Execute shell script on remote host using ssh

SSH site: bwjenkins@bw.ncsa.illinois.edu:22

Command:

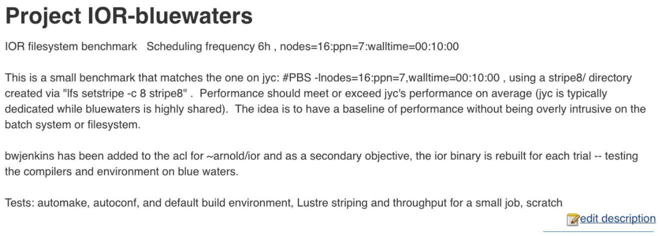
```
echo $PATH $SW_CONFIG $PYTHONPATH > /dev/null
export SW_CONFIG=$SW_CONFIG

rm -f /scratch/system/bwjenkins/tests/lammps/15May15/cle5.2_gnu4.9.3/.lock
cd /tmp
git clone https://github.com/reubendb/SWTools-BWCI
cd SWTools-BWCI
cp -R lammps $SW_ROOT
cd /tmp
rm -rf SWTools-BWCI
cd $SW_ROOT
cd lammps/15May15/cle5.2_gnu4.9.3
swbuildwait
```

Figure 3: SWTools with Lammps

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.



Project IOR-bluewaters

IOR filesystem benchmark   Scheduling frequency 6h , nodes=16:ppn=7:walltime=00:10:00

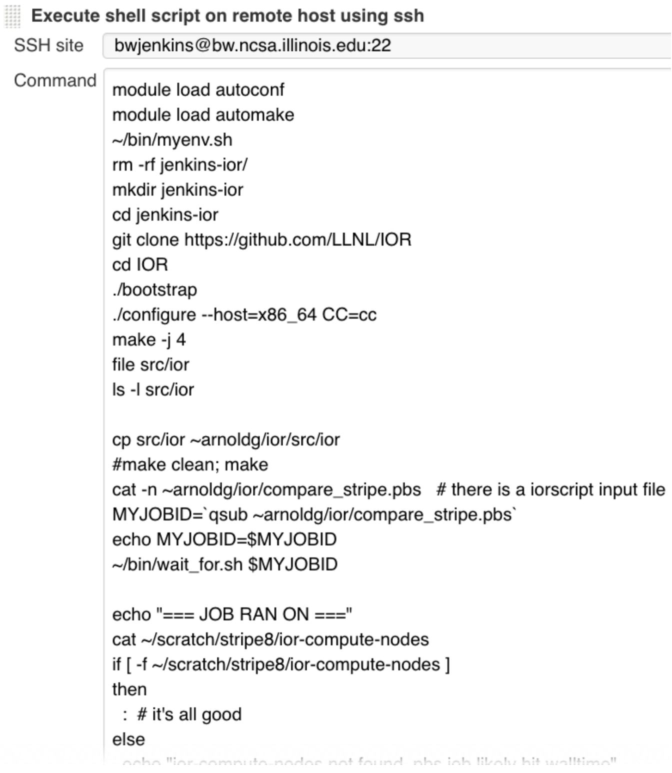
This is a small benchmark that matches the one on jyc: #PBS -lnodes=16:ppn=7,walltime=00:10:00 , using a stripe8/ directory created via "lfs setstripe -c 8 stripe8" . Performance should meet or exceed jyc's performance on average (jyc is typically dedicated while bluewaters is highly shared). The idea is to have a baseline of performance without being overly intrusive on the batch system or filesystem.

bwjenkins has been added to the acl for ~arnold/ior and as a secondary objective, the ior binary is rebuilt for each trial -- testing the compilers and environment on blue waters.

Tests: automake, autoconf, and default build environment, Lustre striping and throughput for a small job, scratch

Figure 4: 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.



bwjenkins@bw.ncsa.illinois.edu:22

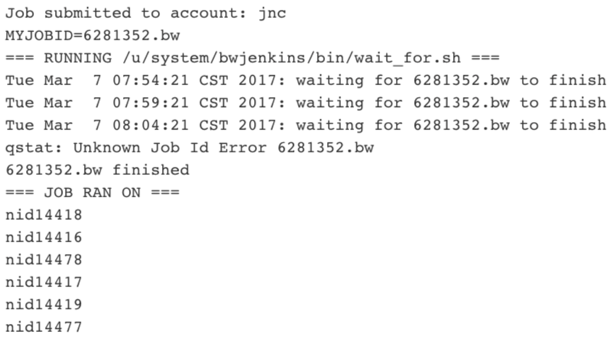
```
module load autoconf
module load automake
~/bin/myenv.sh
rm -rf jenkins-ior/
mkdir jenkins-ior
cd jenkins-ior
git clone https://github.com/LLNL/IOR
cd IOR
./bootstrap
./configure --host=x86_64 CC=cc
make -j 4
file src/ior
ls -l src/ior

cp src/ior ~arnoldg/ior/src/ior
#make clean; make
cat -n ~arnoldg/ior/compare_stripe.pbs   # there is a iorscript input file
MYJOBID=`qsub ~arnoldg/ior/compare_stripe.pbs`
echo MYJOBID=$MYJOBID
~/bin/wait_for.sh $MYJOBID

echo "=== JOB RAN ON ==="
cat ~/scratch/stripe8/ior-compute-nodes
if [ -f ~/scratch/stripe8/ior-compute-nodes ]
then
  :  # it's all good
else
  echo "ior-compute-nodes not found, pbs job likely hit walltime"
```

Figure 5: 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
```



```
Job submitted to account: jnc
MYJOBID=6281352.bw
=== RUNNING /u/system/bwjenkins/bin/wait_for.sh ===
Tue Mar  7 07:54:21 CST 2017: waiting for 6281352.bw to finish
Tue Mar  7 07:59:21 CST 2017: waiting for 6281352.bw to finish
Tue Mar  7 08:04:21 CST 2017: waiting for 6281352.bw to finish
qstat: Unknown Job Id Error 6281352.bw
6281352.bw finished
=== JOB RAN ON ===
nid14418
nid14416
nid14478
nid14417
nid14419
nid14477
```

Figure 6: 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
```

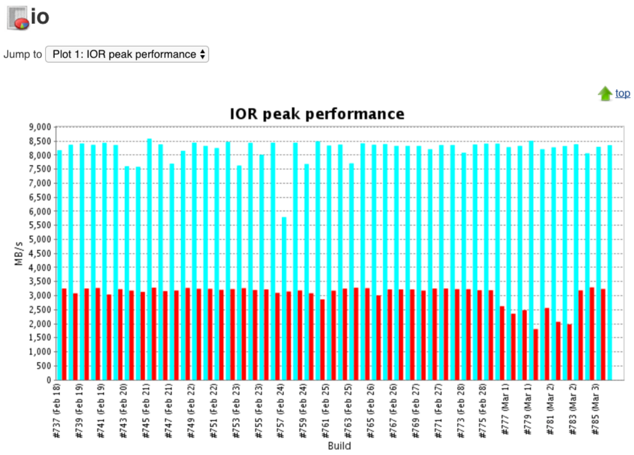


Figure 7: IOR plot view

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.

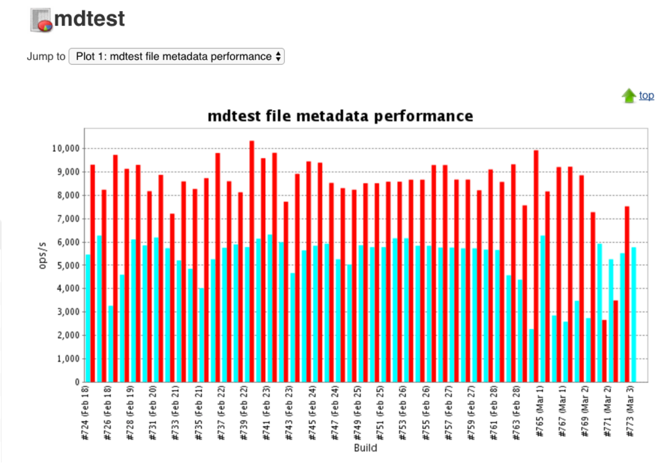
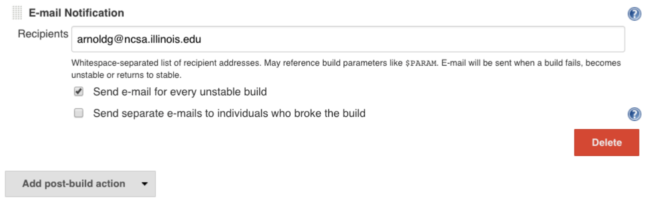


Figure 8: mdtest plot view

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 once again when successful).



Recipients: arnoldg@ncsa.illinois.edu

Whitespace-separated list of recipient addresses. May reference build parameters like $PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

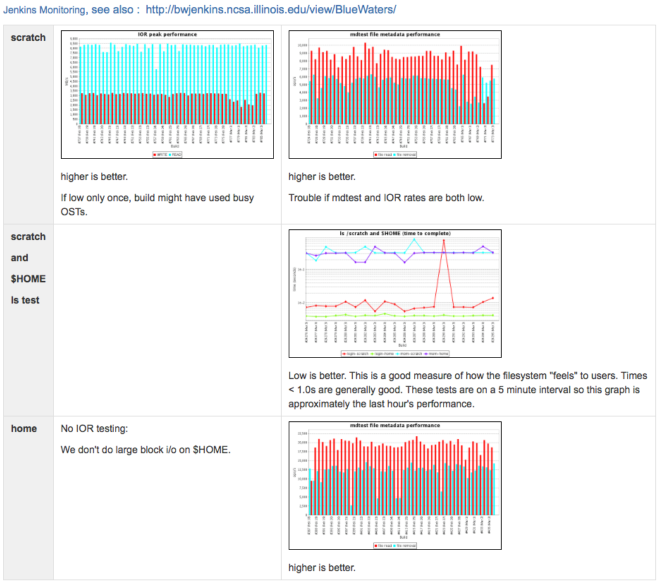
☑ Send e-mail for every unstable build

☐ Send separate e-mails to individuals who broke the build

Figure 9: 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.


IOR peak performance
mdtest file metadata performance
ls /scratch and $HOME (time to complete)
mdtest file metadata performance

Jenkins Monitoring, see also : http://bwjenkins.ncsa.illinois.edu/view/BlueWaters/

scratch

higher is better.

If low only once, build might have used busy OSTs.

higher is better.

Trouble if mdtest and IOR rates are both low.

scratch and $HOME ls test

Low is better. This is a good measure of how the filesystem "feels" to users. Times < 1.0s are generally good. These tests are on a 5 minute interval so this graph is approximately the last hour's performance.

home

No IOR testing:

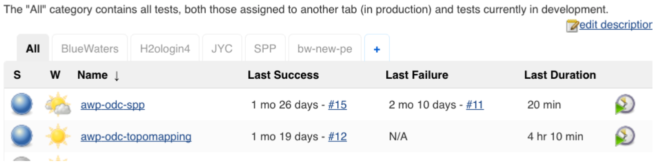
We don't do large block i/o on $HOME.

higher is better.

Figure 10: 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).



The "All" category contains all tests, both those assigned to another tab (in production) and tests currently in development.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		awp-odc-spp	1 mo 26 days - #15	2 mo 10 days - #11	20 min
		awp-odc-topomapping	1 mo 19 days - #12	N/A	4 hr 10 min

Figure 11: tabs for project areas

VII. Best Practices

- Tests are developed on the JYC development rack and not assigned into a Jenkins tab until reviewed.
- Email notification is enabled for most tests and the field is always populated with a test owner.
- All tests contain a description that describes what is tested, resources required, scheduling frequency, and overall summary to be readable by the management team.
- Log file count is limited in the test configuration to keep from filling filesystem space with verbose logs.
- Tests are coded to be more verbose than necessary so that failures can be isolated quickly and easily.

VIII. Conclusion

A. reproducibility and regression testing

Jenkins provides the framework we need to maintain tests in a consistent fashion and track the results over time (logs and plots). Tests typically are not changed so that we get apples-apples comparisons. When a new test is needed for a particular activity, we attempt to derive it from an existing test to maintain consistency and simplify test creation. Reusable test components (watchdog scripts, plotting metrics) are employed to make construction of completely new tests straightforward.

B. rapid reaction

The notification feature keeps test owners engaged in the testing activity of Jenkins and they are often our first responders to system issues. This results in a shorter reaction time when the system is not performing to expectations–staff are notified automatically when the user experience is impacted.

C. caveats

1) transient errors: Transient errors with Jenkins sometimes occur–particularly with the Jenkins ssh subsystem. It's not unusual to get ssh failure rates of 1/100 which are completely transient (immediately running the test again manually clears the issue on the dashboard).

2) test scale and frequency: There is some subjectivity in the frequency and scale (resource utilization) of tests. Testing too often, too large, or some combination of the two may perturb the system such that other tests are impacted or create bottlenecks in the system for the real user community. This is the primary reason we require peer review for new tests.

Acknowledgment

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

References

[1] http://inca.sdsc.edu/ Inca Monitoring: periodic, automated, user-level cyberinfrastructure testing [2] https://jenkins.io/ Jenkins: open source automation server written in Java