Continuous integration in a Cray multiuser environment

Ben Lenard, Tommie Jackson
Argonne National Laboratory
Leadership Computing Facility
9700 S. Cass Ave.
Argonne, IL 60439
blenard@anl.gov, tjackson@alcf.anl.gov

Abstract -- Continuous integration (CI) provides the ability for developers to compile and unit test their code after commits to their repository. This is a tool for producing better code by recompiling and testing often. CI in this environment means that a build will happen either on a schedule or triggered by a commit to a software repository. In this paper, we will be looking at how Argonne National Laboratory's Leadership Computing Facility (ALCF) is implementing CI for its users with security considerations, and project isolation. We currently have implemented a Jenkins instance that has the ability to connect to external software repositories, listen for web-events, compile code, and then execute tests or submit jobs to our job scheduler Cobalt. While Jenkins provides the ability to build code on demand and execute jobs on our systems, we will also be deploying another solution that is tied to GitLab that provides seamless integration at a later time. This paper will discuss the how the facility user facing Jenkins solution was implemented at ALCF with open source plugins, what considerations where taken into account, and the success so far with projects.

Keywords--*continuous integration; supercomputers; HPC*

I. Introduction

Scientific software testing, particularly simulation software, faces many unique challenges. Oftentimes developers and scientists are unable to test their software on local systems at the scales demanded by leadership class systems, or even on smaller instances of what are often unique architectures and software stacks [11]. The ALCF has received numerous requests from facility end-users to provide a means to run automated tests of their scientific software against the unique hardware found at our facility. To support this, the ALCF has started a pilot project to provide users a Continuous Integration (CI) solution to allow users easier access to ALCF platforms for porting, extending and debugging codes that run on these unique systems.

In 2016, the ALCF acquired Theta [1], which is currently a 4392-node, 24-rack Cray XC40 system accompanied by a 10PB Lustre file system. Each Theta node is equipped with a 64-core Intel Xeon Phi 7230 Knight's Landing processors with 16GB of MCDRAM and 192GB DDR4 RAM connected to a 10 PB Lustre filesystem. The ALCF also has a 49K node, 48-rack IBM BlueGene/Q system [5], Mira, accompanied by 26PB of GPFS file systems. To complement these resources, the ALCF has a Cray visualization cluster named Cooley which is 126 nodes [6] and has access to the 26PB GPFS file systems and soon will have direct access to Theta's 10PB Lustre file system. Each Cooley node has two 2.4 GHz Intel Haswell E5-2620 v3 processors, one NVIDIA Tesla K80 (with two GPUs), 384GB RAM per node, and 24 GB GPU RAM.

The ALCF is an open science facility. The user base is primarily outside of Argonne and is distributed globally. System users do not typically host their software repositories at the ALCF, and the majority of projects running at the ALCF are hosted within their own repositories, often at an external university or laboratory repositories in various software repository formats, often git or subversion. Additionally, some of the users of ALCF systems are from industry with their own, unique considerations for their source hosting and security concerns.

CI has many different definitions but in general it means that a given user or organization has the desire to compile their code and then run a predefined set of tests, and possibly repeat these steps on various architectures. Some people view CI as the desire to compile on every commit, whereas other users have the desire to only compile and then validate their build weekly. That being said, the ALCF wanted a solution that had the ability to meet the needs of a wide range of unique user requirements as well as be able to connect to software repositories wherever they might be located.

The ALCF has deployed an open source Jenkins solution that it is currently running a pilot with facility's users. The solution provides the participating projects the ability to compile and execute code while maintaining the project's isolation. The solution utilizes open source Jenkins plugins to provide the connectivity and security requirements. The Jenkins server itself has been deployed to the ALCF's existing VMWare infrastructure.

The remainder of this paper has been organized into five sections. First, we will discuss the requirement of the ALCF for this solution. Next, we will discuss the design and implementation of the Jenkins solution. This section also covers the key Jenkins plugins used in this deployment. Then we will discuss how an ALCF project is configured within Jenkins. Lastly, we will discuss our results and future developments.

II. REQUIREMENTS

The ALCF is a user facility with various concurrent international projects that have been allocated time. As such, a set of requirements was established to ensure the success of the CI initiative:

- Security: One of the biggest issues with deploying this solution with ALCF for our users is multitenancy. We have many active projects at any given time and need to ensure isolation between these projects. Project isolation refers to separating the execution for projects as well as separating and keeping their data secure. In addition to this, we needed a way to expose the service's User Interface (UI) to the Internet since most of users remote. Furthermore, we needed the ability to integrate into the ALCF multi-factor authentication system, and log individual's interactions with the system to our central logging service.
- Multiplatform support: The ALCF has two different Cray systems, an IBM BG/Q system, and both PowerPC and x86_64 build servers, we need a solution that runs on these current platforms and would be portable to likely future platforms.
- Ease of use: We need a solution that is easy to use and well documented since we are offering it as a service to our users.
- Integration with various repositories: Since the ALCF does not host software repositories for users, we needed the solution to be able to interact with many kinds of repositories and hosts.
- Maintainable / scalable: At any given time, the ALCF can have over 356 projects with many users attached to each. We need a solution that can support a number of projects without a huge demand on resources. It would inefficient to have 356 agents sitting idle on a given system waiting for a build or execution to occur. Build events might happen every 15 minutes or could occur weekly.
- Cobalt integration: Since ALCF uses Cobalt exclusively for our scheduling services, we needed a solution that would provide the ability to interact with Cobalt. Cobalt has the ability to issue batch commands.
- Actively Maintained: We want a solution that is actively maintained so that as new technologies would emerge, the solution could hopefully embrace and support these technologies..

III. DESIGN AND IMPLEMENTATION

After review of various open source CI solutions, given the requirements set forth, and the in-house expertise, we decided to implement the open source version of Jenkins along with various open source plugins that are provided by the Jenkins community. The ALCF currently uses Jenkins for its internal development projects and support services scheduling needs. Jenkins is an open source automation server that provides CI [7] and has been around since the mid 2000's (Hudson) [8]. Since Jenkins is Java based, it will run on almost any platform assuming that there is a Java Run-Time Environment (JRE) for that platform.

ALCF has an existing VMWare infrastructure, and we determined it was the right location to host Jenkins as well as the Nginx webserver. The ALCF infrastructure is composed of numerous hypervisors as well as the features that will relocate virtual machines (VMs) on a hardware failure. The Nginx webserver effectively is a proxy from the Internet to the Jenkins application. The Argonne firewall only allows for http and https traffic to the Nginx server; we only do redirections from http to https on the http port. We also chose VMWare to house these components of the solution since we can dynamically allocate additional resources to the Virtual Machine VM. The current Jenkins VM is running Red Hat Enterprise Linux 7 with 2 vCPU's, 8G of ram, with the Jenkins home directory residing on our NFS server. The reason for hosting the Jenkins working directory on our NFS server is to provide the ability to grow the filesystem easily, the ability to take hourly, daily and weekly snapshots separate from the VM, and to provide a way to separate the Jenkins data from the VM should we need to rehost Jenkins. We have deployed a similar approach with separating the Jenkins data from the OS data for the internal ALCF Jenkins instance as well.

A brief description of how allocated projects are setup within the ALCF will be given before describing the Jenkins solution. Within the ALCF, we create and assign a Unix group to a project, and then an end user is also assigned to a project which is in turn a group. In an effort to keep the projects isolated, we deployed a service account for each project that wishes to use our CI solution. Service accounts in this case are internal accounts to ALCF with its group set to the project's group. The service account is never directly logged into by the project's users but rather it is used to execute commands under the project's behalf. The purpose of this is so that if user has data that needs to be used by the project's build process, the user just needs to ensure the group permissions are correct for that data.

For illustration purposes, for the remainder of the paper, User1 is tied to Project1 and User2 is tied to Project2, and User3 is tied to Project1 and Project2. And, Project1 has a service account called Project1_SVC and Project2 with the account Project2_SVC. Each of the service accounts are tied to the respective Unix groups for the project. The "project and group" concept is an existing practice within ALCF so we decided to leverage this for our Jenkins deployment.

The first step to creating this environment is to download and install Jenkins. There are two download options for Jenkins, one is a weekly build whereas the other is what they call Long Term Support (LTS) [9]. LTS builds are chosen every 12 weeks and tend to be more stable than the weekly builds; since this service is serving our users, we decided to deploy the LTS build.

Jenkins has a webserver built into it but instead we utilize a Ngnix proxy between the Internet and Jenkins itself. We decided to do this because, if we needed to decouple the webserver from the VM at a later date, the process would be easier since the webserver would already be independent. The use of Ngnix also allows us to expose additional webservices at a later date on ports 80 and 443 without opening additional ports. In our case, the main URL is https://cimaster.alcf.anl.gov but the Uniform Resource Identifier (URI) of /jenkins/ allows Ngnix to know to direct web traffic to the Jenkins server. As these practices become more widespread among the scientific community, this approach provides a natural and easy way to extend our support of new tools.

A new Jenkins installation is a blank canvas for one to deploy the plugins to customize Jenkins for the organization's needs. Jenkins has over one thousand plugins built by its community ranging from interacting with a git repository to controlling a web service [10]. The next section will discuss the key plugins that are used by ALCF to provide the CI solution to its users. We currently have over one hundred plugins installed; the key plugins will be discussed in detail below. At the end of the paper in Appendix A, a complete list of plugins for this configuration is provided. Everything in this paper is based on the open source plugins provided by the Jenkins community.

The key Jenkins plugins utilized are as follows:

- Folders: The key to multitenancy within Jenkins is the 'Folders' plugin. The 'Folders' plugin allows for 'Folders' within Jenkins. Like a Unix filesystem, there is a parent folder for projects, such as '/home,' and then there are directories for the projects, such as '/home/project1.' Thus, for each project that requests to use CI, a folder is created with their matching name. The matching name is not a requirement of Jenkins but rather a way for ALCF to keep a handle on organization. Given the example projects mentioned earlier, there are two folders, 'Project1' and 'Project2.' With the coordination of the LDAP and Matrix plugins, these folders are now locked-down to the project's group, namely Project1 and Project2 respectively. To take this a step further, some projects have subprojects and the 'Folders' plugin allows for subproject permissions as well. The credentials used for Jenkins to access a code repository are stored at the folder level as well. This ensures that Project1's credentials will never be seen by anyone in Project2.
- Credentials: The 'Credentials' plugin allows Jenkins to store credentials to external repositories as well as the credentials for the project's slave when a build or execution needs to occur.
- Job and Node Ownership: The 'Job and Node Ownership' allows for Jenkins to have a notion of who owns a job and node. This allows for Jenkins to decide and restrict on what slave to run a given job on. It is coupled with the 'Job Restrictions Plugin.' We assign the owner of a folder to the project's service account; this is used in the 'Job Restrictions Plugin.'
- Job Restrictions: The open source version does not have the ability to tie a slave to a folder, the commercial version of Jenkins does, so we use this plugin to restrict and direct what project runs on what slave. When the slave is created by the ALCF infrastructure staff in Jenkins, the restriction on the node is to the project's service account being the job's owner, or the person submitting the job is part of the project. This provides the job routing functionality needed in this environment.
- LDAP: The LDAP plugin allows the Jenkins server to authenticate the user against the ALCF LDAP servers which in turn contacts the ALCF SafeNet server for onetime passwords. In addition to this, the LDAP plugin provides Jenkins with the list of projects, groups, the user belongs to. These groups are how we use 'folders' and the matrix plugins to achieve isolation.
- Matrix Authorization Strategy and Matrix Project: The ALCF user management system creates a group in LDAP per project. This plugin allows for somewhat fine-grain access control on the folders, and jobs. It also allows for ALCF, or even the project's users, to restrict what individual users can do within a given folder. Similar to the Linux file system, a user can set the permissions for the group level, or even an individual. As of right now, ALCF is only doing this at the group level.
- SCM Sync Configuration: In any system, backups are key to protect yourself from failure or human mistakes. That being said, the "SCM Sync Configuration" plugin commits every Jenkins configuration change to ALCF's internal Gitlab instance. When a user creates a new job for a project, that new job is automatically committed to the Gitlab repo when the 'Save' button is pressed during job creation. When a user accidently deletes a job from Jenkins, we have a copy we can restore from git. The underlying configurations to Jenkins are XML files, and this is what gets updated in the git repository.
- SSH Slaves: In Jenkins there are three methods for invoking a slave, and for this implementation we used the SSH Slave method. The two main, three if you include Windows, methods for slave invocation are either as a static service or spawned via SSH. We chose the SSH Slave implementation since this allows Jenkins to spawn a slave on the target system when the demand for the slave arose, and shutdown the slave when the slave was idle for 1 minute, allowing the resources consumed by the slave Java process to be released.
- Workspace Cleanup: When a project builds, a workspace is created under the service account's home directory. It is helpful to provide the ability to our users to clean the build directory after their testing has completed. This is basically equivalent to running a 'make clean' as the last step after your

tests have finished. Since the service account is part of the project, the disk space used by the service account is reflected in the project's quota. That being said, the workspace cleanup plugin facilitates the removal unneeded files and reducing the project's disk usage.

IV. PROJECT ONBOARDING PROCESS

Within our Jenkins environment, each on-boarded project first has their corresponding LDAP group added to the 'Global Security Matrix' within Jenkins with the only permission of 'Overall Read.' This is required so that the Project members are able to login into Jenkins but doesn't grant them any privileges at the global level. Then a folder with the corresponding project name is created. For example, a Project1 folder is created for Project1 and a Project2 folder is also created. For each of the project folders, we enable Project-based security, and the project's group and service account, and tell Jenkins not to inherit permissions from the parent. An example of this can be found in the appendix C.

Next, we configure a Jenkins slave for project per computing resource. The ALCF infrastructure team creates this slave resource and sets the permissions such that the only thing the project has the ability to build on the slave. This is key to how we provide project isolate so that Project1 cannot execute something under Project2's slave. Without the key plugins listed above, Jenkins does not provide the security needed in a multitenant environment. When the service account is created, for example Project1's, Project1_SVC, a home directory is also created, /home/Project1_SVC, on either GPFS or Lustre. Since some projects could be running tasks on multiple compute resources, Theta, Cooley, Mira, etc, a subdirectory is created for the resource name to prevent any type of issues for Jenkins. In an effort to save resources on the various nodes, we have the slaves only spawn when the demand for them arises and then exit when they are idle. A complete list of slave settings can be found in appendix D.

One might ask why we need to setup a slave per project and resource? Since we are utilizing the open source version of Jenkins, this is a limitation of the implementation of Jenkins as well as Java.

At ALCF, one of security requirements for our users is two-factor authentication. In order to keep the environment secure, and not have idle Jenkins slaves in the environment, we have Jenkins communicate with the compute resources, the login nodes, via internal non-routable networks. Therefore, the compute resources will only accept SSH keys from specific IP's within ALCF. That being said, as part of the project's onboarding process, the ALCF infrastructure team creates the SSH keypair during the project onboarding for the slaves use.

Another reason for using the existing compute logins is so that the project members have access to the environment and understand the environment. These logins are the same logins that project member's log into when using the ALCF resources. This was also done to lower the learning curve when a project desired to use our Jenkins instance.

V. PROJECT USAGE

When a user first logs in to Jenkins, they are greeted with an empty canvas for the project: the project's folder. While Jenkins provides functionality out-of-the-box, each project is required to build their own build scripts and workflows as each project is unique. Jenkins does not create any build scripts or testing scripts for the projects, and Jenkins does not require the use of a certain framework. That being said, the scripts can usually be ether imported into Jenkins or executed by Jenkins since projects usually have their own scripts that would have been executed by a project's member.

Once the project's code has been compiled, and test, the project might have the desire to submit the job for execution on the system. That being said, Cobalt, the scheduler used by ALCF, conforms to the POSIX standard for batch schedulers with extensions to support specialized systems like flags that only matter for BlueGene platforms, or other unique architectural features or constraints. Therefore, a project would extend their scripts or Jenkins jobs to execute the necessary commands for execution. While Jenkins does have a Portable Batch System (PBS) plugin to allow it to interact with schedulers that understand PBS commands, we have not experimented with this plugin as of yet [2][3]. When an individual submits a job on with Cobalt, Cobalt looks to see if a user is associated with a project. In our example, if User1, submitted a job to Cobalt for execution, Cobalt verifies that User1 can submit a job against Project1's allocation. Therefore, Project1_SVC is also attached solely to Project1's allocation and can only submit against Project1's allocation. ALCF uses Cobalt exclusively for job submission within the environment, so research into utilizing other schedulers has not been pursued to date.

Jenkins provides a web-based graphical user interface (GUI) that provides a quick and easy to navigate view of a project' job status. When a user logs into the GUI, they are greeted with their project's folder. Under the project's folder, a list of the correspond project jobs appear along with the last successful run, last failed run, and duration of the last run. The Jenkins GUI also provides the build history as well as a report that shows the build duration and success trend over time.

While one might argue that crond could do everything that Jenkins provides, but Jenkins provides the following functionality out-of-the-box:

- Build-steps: Jenkins provides the ability to have conditional steps based on the output of the previous step.
- Build timeouts: Within Jenkins jobs, you have the ability to have Jenkins abort the job if it takes too long to complete.

- Capturing of stdout, stderr: Jenkins captures the job's output and saves it for the user corresponding with the build or run for future review. Jenkins will also purge the logs based on the policy set in place.
- The ability to throttle users concurrency: Cron will just execute a script at a given interval. In addition to executing on a given interval, Jenkins slaves provide the ability to limit the number of concurrent tasks a project is able execute on a resource. In other words, if the previous task has not finished executing, and someone else in the project tries to execute against the same resource, and there is only one degree of concurrency allowed, the second job will be forced to wait until the completion of the first task.
- Secure credentials store: Jenkins provides a method to store project's credentials to their software repositories without leaving the credentials on a shared filesystem.
- Centralization: Jenkins provides a central location for project members to interact with a CI solution across the various compute platforms, whereas crond does not provide this functionality.

VI. CONCLUSIONS AND FUTURE WORK

We have implemented a secure multitenant user facing CI system that allows our users to compile their project's code after fetching it from an external git or SVN repository. We have deployed Open Source Jenkins in a way that keeps projects isolated utilizing the existing resources within the ALCF. The CI system allows for projects to compile their code ether on a schedule or by listening for webhook events, for example on a commit. Once the code is compiled, Jenkins allows the user to submit their job to the compute resource for execution via Cobalt.

We currently have a few 'friendly' projects utilizing the Jenkins CI solution within ALCF; some of these projects execute solely against Theta and other projects execute against Theta, generic X86, and Mira. We are slowly expanding our friendly user base to both harden the current solution as well as expose it to more diverse workflows. More diverse workflows would give us a better understanding of other users' needs so we can address them. This would also allow us to experiment with some of the advanced features of Jenkins, such as Pipelines.

While we have had success with the current implementation of Jenkins, we have would like to enhance our environment. As of right now, the project onboarding process is a manual process that requires the ALCF infrastructure team's intervention. We would like to automate this process from creating the service account, to creating the resources within Jenkins. Since the underlying Jenkins system is XML based, this should not be a difficult task.

Another area of development would be Jenkins direct integration with Cobalt and other schedulers. While our users are able to execute commands against Cobalt in a batch mode, it would be nice to directly integrate into Cobalt by utilizing its API. Jenkins provides a documented plugin architecture, complete with tutorials, for plugin development.

Currently we do not have special Cobalt queue for CI job submission, so the given project submits against their awarded allocation. In an effort to promote CI, and often code testing, the ability to provide projects with a queue that they can submit jobs to without effecting their awarded allocation is something we have been looking into for future development. While the process of adding another queue is somewhat trivial, it raises additional questions about how often a project can submit to the queue, how many compute nodes should the queue allow, and other policy topics.

ACKNOWLEDGMENT

This research used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-06CH11357.

REFERENCES

[1] Harms, K., Leggett, T., Allen, B., Coghlan, S., Fahey, M., Holohan, C., . . . Rich, P. (2017). Theta: Rapid installation and acceptance of an XC40 KNL system. *Concurrency and Computation: Practice and Experience,30*(1). doi:10.1002/cpe.4336

[2] Vergara Larrea, V. G., Joubert, W., & Fuson, C. (n.d.). Use of Continuous Integration Tools for Application ... Retrieved April 15, 2018, from https://cug.org/proceedings/cug2015_proceedings/includes/files/pap147.pdf

[3] Kinoshita, P. (n.d.). Jenkins Plug-ins. Retrieved April 16, 2018, from http://biouno.org/jenkins-plugins.html

[4] (n.d.). Retrieved April 16, 2018, from https://jenkins.io/doc/developer/tutorial/

[5] Mira. (n.d.). Retrieved April 16, 2018, from https://www.alcf.anl.gov/mira

[6] Visualization Cluster. (n.d.). Retrieved April 16, 2018, from https://www.alcf.anl.gov/resources-expertise/analytics-visualization

[7] Jenkins. (n.d.). Retrieved April 16, 2018, from https://jenkins.io/

[8] Smart, J. F. (n.d.). Jenkins: The Definitive Guide. Retrieved April 16, 2018, from https://www.safaribooksonline.com/library/view/jenkins-the-definitive/9781449311155/ch01s04.html

[9] Jenkins. (n.d.). Retrieved April 16, 2018, from https://jenkins.io/download/

[10] Jenkins Plugins. (n.d.). Retrieved April 16, 2018, from https://plugins.jenkins.io/

[11] Hovy, C., & Kunkel, J. (2016). Towards Automatic and Flexible Unit Test Generation for Legacy HPC Code. *2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE)*. doi:10.1109/se-hpccse.2016.005

APENDIX A

Complete list of currently installed plugin:

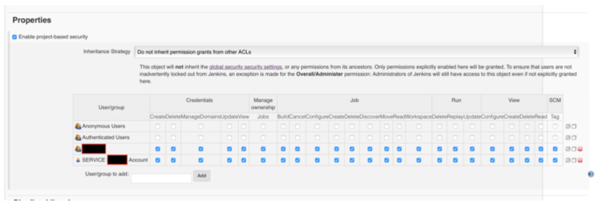
- Ant Plugin
- Apache HttpComponents Client 4.x API Plugin
- Audit Trail
- Authentication Tokens API Plugin
- Backup plugin
- Branch API Plugin
- Build Pipeline Plugin
- Build Timeout
- Build-Publisher plugin
- Cobertura Plugin
- Command Agent Launcher Plugin
- Conditional BuildStep
- Credentials Binding Plugin
- Credentials Plugin
- Dashboard View
- Display URL API
- Docker Commons Plugin
- Docker Pipeline
- Durable Task Plugin
- Email Extension Plugin
- External Monitor Job Type Plugin
- Folders Plugin
- Generic Webhook Trigger Plugin
- Git client plugin
- Git Parameter Plug-In
- Git plugin
- GIT server Plugin
- Gradle Plugin
- Green Balls
- HTML Publisher plugin
- Jackson 2 API Plugin
- Javadoc Plugin
- JavaScript GUI Lib: ACE Editor bundle plugin
- JavaScript GUI Lib: Handlebars bundle plugin
- JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) plugin
- JavaScript GUI Lib: Moment.js bundle plugin
- Job and Node ownership plugin
- Job Restrictions Plugin
- jQuery plugin
- JSch dependency plugin
- JUnit Plugin
- LDAP Plugin
- Mailer Plugin
- MapDB API Plugin
- Matrix Authorization Strategy Plugin
- Matrix Project Plugin
- Multi-configuration (matrix) project type.
- Maven Integration plugin
- Multiple SCMs plugin
- OWASP Markup Formatter Plugin
- PAM Authentication plugin
- Parameterized Trigger plugin
- Pipeline
- Pipeline Graph Analysis Plugin
- Pipeline: API
- Pipeline: Basic Steps
- Pipeline: Build Step
- Pipeline: Declarative
- Pipeline: Declarative Agent API
- Pipeline: Declarative Extension Points API
- Pipeline: Groovy
- Pipeline: Input Step
- Pipeline: Job
- Pipeline: Milestone Step
- Pipeline: Model API
- Pipeline: Multibranch
- Pipeline: Nodes and Processes
- Pipeline: REST API Plugin
- Pipeline: SCM Step
- Pipeline: Shared Groovy Libraries
- Pipeline: Stage Step
- Pipeline: Stage Tags Metadata
- Pipeline: Stage View Plugin
- Pipeline Stage View Plugin.
- Pipeline: Step API
- Pipeline: Supporting APIs
- Plain Credentials Plugin
- Resource Disposer Plugin
- Run Condition Plugin
- SCM API Plugin
- SCM Sync Configuration Plugin
- Script Security Plugin
- SSH Agent Plugin
- SSH Credentials Plugin
- SSH Slaves plugin
- Structs Plugin
- Subversion Plug-in
- Timestamper
- Token Macro
- Tracking Git Plugin
- View Job Filters
- Windows Slaves Plugin
- Workspace Cleanup Plugin

APPENDIX B

Environment details:

- VMWare 5.5
- Jenkins VM RHEL 7.x
- Oracle Java 1.8.x
- Nginx 1.x
- Jenkins version 2.107.1
- Theta logins are running SLES 12 SP2
- Generic X86 build servers are running RHEL 7.x
- Cooley is running RHEL 7.x

APPENDIX C



Properties

✓ Enable project-based security

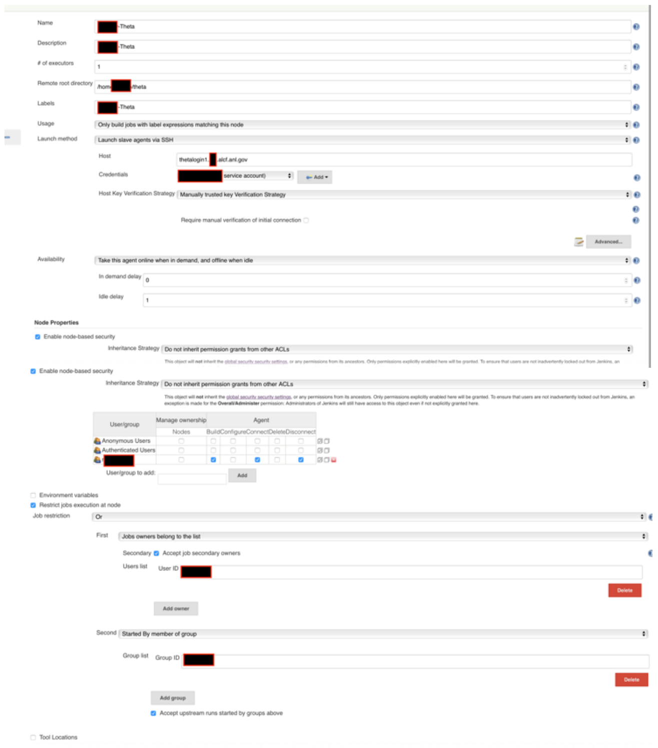
Inheritance Strategy: Do not inherit permission grants from other ACLs

This object will **not** inherit the global security security settings, or any permissions from its ancestors. Only permissions explicitly enabled here will be granted. To ensure that users are not inadvertently locked out from Jenkins, an exception is made for the **Overall/Administer** permission: Administrators of Jenkins will still have access to this object even if not explicitly granted here.

User/group	Credentials					Manage ownership	Job									Run			View				SCM
	Create	Delete	ManageDomains	Update	View	Jobs	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag
Anonymous Users	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Authenticated Users	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
[redacted]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SERVICE [redacted] Account	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

User/group to add: Add

APPENDIX D



Name	[redacted]-Theta
Description	[redacted]-Theta
# of executors	1
Remote root directory	/hom[redacted]/theta
Labels	[redacted]-Theta
Usage	Only build jobs with label expressions matching this node
Launch method	Launch slave agents via SSH
Host	thetalogin1.[redacted].alcf.anl.gov
Credentials	[redacted] service account)
Host Key Verification Strategy	Manually trusted key Verification Strategy

Require manual verification of initial connection ☐

Advanced...

Availability	Take this agent online when in demand, and offline when idle
In demand delay	0
Idle delay	1

Node Properties

✓ Enable node-based security

Inheritance Strategy: Do not inherit permission grants from other ACLs

This object will **not** inherit the global security security settings, or any permissions from its ancestors. Only permissions explicitly enabled here will be granted. To ensure that users are not inadvertently locked out from Jenkins, an

✓ Enable node-based security

Inheritance Strategy: Do not inherit permission grants from other ACLs

This object will **not** inherit the global security security settings, or any permissions from its ancestors. Only permissions explicitly enabled here will be granted. To ensure that users are not inadvertently locked out from Jenkins, an exception is made for the **Overall/Administer** permission: Administrators of Jenkins will still have access to this object even if not explicitly granted here.

User/group	Manage ownership	Agent				
	Nodes	Build	Configure	Connect	Delete	Disconnect
Anonymous Users	☐	☐	☐	☐	☐	☐
Authenticated Users	☐	☐	☐	☐	☐	☐
[redacted]	☐	✓	☐	✓	☐	✓

User/group to add: Add

☐ Environment variables

✓ Restrict jobs execution at node

Job restriction: Or

First: Jobs owners belong to the list

Secondary ✓ Accept job secondary owners

Users list — User ID: [redacted]

Delete

Add owner

Second: Started By member of group

Group list — Group ID: [redacted]

Delete

Add group

✓ Accept upstream runs started by groups above

☐ Tool Locations