POWER8 Continuous Integration Technology Review

Leon Leighton, Thomas Olson, Derek Wong
Project 35

Abstract

This document provides a technology review of nine pieces of the IBM POWER8 Continuous Integration project. For each piece three options are compared using criteria based on the goal the piece is attempting to achieve. A final choice of best option is chosen based on the evaluation.

CONTENTS

I. Introduction

This document contains an evaluation of nine pieces of technology that we will be using in the IBM POWER8 Continuous Integration project. We define evaluation criteria based on the goal we are attempting to achieve by using the stated piece. For each piece we have selected three options, and we choose the best option based on the evaluation.

The nine pieces are:

Cluster Management The software that will be used to manage a cluster of POWER8 nodes. Authored by Leon Leighton.

Continuous Integration Software The software that will perform the building and testing of projects. Authored by Leon Leighton.

Configuration Management System that will be used to manage the configuration of the system. Authored by Leon Leighton.

Linux Distribution Support The Linux distribution that projects will be built and tested on. Authored by Thomas Olson.

Platform For Running Builds The type of platform used for building and testing. Authored by Thomas Olson.

Configuration File Formats The file format that will be used by projects for configuring builds. Authored by Thomas Olson.

Login/Authentication Method for users to login to the system. Authored by Derek Wong.

Frontend/Web Frameworks Method of creating a user interface for interacting with the system. Authored by Derek Wong.

Tracing State of Builds/Tests Functionality to show users the status of their builds and tests. Authored by Derek Wong.

II. Cluster Management

A. Options

1) OpenStack: OpenStack is a cluster management solution originally created by Rackspace and NASA, and now managed by the OpenStack Foundation. It contains a number of projects that provide services such as object and block storage, identity management, networking and compute resource management, bare metal provisioning, and DNS services. This modular approach allows users to select which parts they need to accomplish their goals without having to install components they will not be using.

2) CloudStack: CloudStack is an Apache Software Foundation cluster management project which aims to provide many of the same services as OpenStack. It follows a more monolithic approach where most components are distributed as part of a single binary.

3) Ganeti: Ganeti is a cluster management solution from Google that focuses on managing virtual machines. It can be used to create new virtual machines, manage disks, and manage failover of virtual machine instance.

B. Goals for use

The primary goal for using cluster management software is to manage the creation and destruction of temporary virtual machines and containers that will be used for building and testing projects.

C. Criteria for Evaluation

1) Community Support: Our chosen solution should have a community that provides support in the form of documentation, bug fixes, and assistance.
2) Linux Support: Linux must be supported as both host and guest.
3) Container Support: Optional, but we would like to have support for building and testing in containers.
4) Jenkins Plugin Support: Jenkins is a likely piece of our solution, therefore having a Jenkins plugin that interacts with our cluster management solution is highly desired.

D. Table

Criteria	OpenStack	CloudStack	Ganeti
Community Support	Yes	Yes	Yes
Linux Support	Yes	Yes	Yes
Container Support	Yes	Partial	No
Jenkins Plugin Support	Yes	Yes	No

E. Discussion

OpenStack, CloudStack, and Ganeti can all run on Linux and can host Linux virtual machines. Ganeti does little more than this, however, and will not be considered further. Both OpenStack and CloudStack can interact with Jenkins through a plugin. OpenStack appears to have a larger community and a more mature container solution than CloudStack.

F. Selection of best option

OpenStack is our chosen solution for cluster management. It meets all of our criteria for evaluation.

III. Continuous Integration Software

A. Options

1) Jenkins: Jenkins is an automation server that can be used to automate builds, tests, and deployments. It is extensible and has a large number of plugins that enable it to interact with other systems.

2) Buildbot: Buildbot is another system that enables the automation of builds, tests, and deployments. It can be extended through the use of Python configuration files.

3) Strider: Strider is a continuous integration server written in Node.js. Like Jenkins it uses plugins to extend its functionality.

B. Goals for use

Automate building and testing Open Source projects.

C. Criteria for Evaluation

1) Open Source: As our target audience is Open Source software developers, we should use an Open Source solution to be consistent with the community.
2) Language Support: We need to be able to build and test projects written in a variety of languages.
3) Notifications: We need some way of providing the user with feedback on the status of their builds and tests.
4) Extensible: If we find that there are missing features we will need some way of extending the functionality of the software.
5) Number of Existing Plugins: A larger number of plugins increases the probability that someone may have already provided functionality that we need that is not already in the software.

D. Table

Criteria	Jenkins	Buildbot	Strider
Open Source	Yes	Yes	Yes
Language Support	Any	Any	Any
Notifications	Yes	Yes	Yes
Extensible	Yes	Yes	Yes
Number of Plugins	High	Low	Medium

E. Discussion

All three of our options meet our basic evaluation criteria. However, there is quite a large difference in the number of plugins available, with Jenkins having a much larger number than Buildbot or Strider. There is also a large variety in the type of plugins for Jenkins, covering functionality such as source code management, reports and notifications, and UI elements.

F. Selection of best option

With its large number and variety of plugins, Jenkins is our chosen solution for continuous integration software.

IV. Configuration Management

A. Options

1) Ansible: Ansible is a configuration management and infrastructure automation solution. It is agentless, using SSH to connect to and run commands on nodes. Ansible uses YAML files for configuration.

2) Chef: Chef is a configuration management solution that uses agents running on each node to poll a central server to access configuration 'cookbooks' that are written in a Ruby.

3) Puppet: Puppet is also a configuration management solution. Like Chef, it uses agents on each node and a central 'puppet master' server. Configuration is done in a language specific to Puppet that is meant to be accessible to system administrators.

B. Goals for use

Using a configuration management system will allow us to create a solution that can be replicated by others who may wish to setup their own POWER8 Continuous Integration system. Used in conjunction with a version control system, it will also give us the ability to roll back changes that have been made, giving us flexibility in experimenting with different options as we progress through the project. At a higher level, a configuration management system will allow us to specify what we want the state of the system to be, and it will also give us an ultimate 'source of truth' for the project.

C. Criteria for Evaluation

1) Support for installing Jenkins
2) Overhead
3) Difficulty

D. Table

Criteria	Ansible	Chef	Puppet
Installs Jenkins	Yes	Yes	Yes
Overhead	Low	Moderate	Moderate

E. Discussion

As we are likely to use Jenkins for our continuous integration software we would like there to be an already existing method to install and configure it with the option we choose for configuration management. All three options under consideration meet this criteria through user created modules. We would also like to reduce the overhead involved in using a configuration management system. Both Puppet and Chef require the use of a master server and an agent installed on any node that will be managed. While this is not a high level of overhead, we do contrast that with Ansible which uses SSH which is already included on most Linux systems. It should be noted, however, that since Ansible uses SSH, we will need to create and secure the SSH keys that will be used for authentication.

F. Selection of best option

While all three options would allow us to accomplish our goals, we have selected Ansible as our best option.

V. Linux Distribution Support

A. Options

1) RedHat Enterprise Linux: RedHat Enterprise Linux is widely considered to be one of the standard Linux distributions for server environments, having been around for a number of years and being very stable. It is supported by the Red Had corporation and has a ten-year supported production cycle for each major version, followed by an extended support period[1]. Unlike most Linux distributions, it has a paid subscription requirement for support and the extended support has a higher price. However, as a paid product, Red Hat provides dedicated support and certifications which can be important if such certifications are required for a project or if the project requires high availability and dedicated support in case issues arise.

2) CentOS: CentOS is an unbranded version of RedHat Enterprise Linux, providing the same functionality and stability with only very minor functional differences. It is supported by the CentOS Project and is free for all use. As a result, it does not have the certifications nor the dedicated support of RedHat Enterprise Linux. As it tries to track the development of RedHat Enterprise Linux very closely, CentOS has approximately the same ten-year support cycle[2], but it does not have the same extended support period provided by Red Hat.

3) Ubuntu Server: Ubuntu Server is a version of the Ubuntu distribution that differs from the desktop version only by the installer and the life cycle[3]. It is supported by Canonical and has a five-year life cycle per major version. Like CentOS, it is free to use, but Canonical optionally offers a contract for dedicated support. Ubuntu also tends to update application packages sooner, trading some amount of stability for newer functionality.

B. Goals for use

The operating system selection will have a major effect on the stability and ease-of-use of the entire system. Our goal is to select an operating system that is stable, widely supported and easy to work with.

C. Criteria for Evaluation

1) Stability: It is important that the operating system used for running builds has as few issues as possible so as to minimize the need for manual intervention in case there's a failure.
2) Support: The operating system should have a strong support base to maximize ease of maintenance and help ensure longevity.
3) Usability: The greater the usability and ease of configuration, the easier it becomes to maintain and improve the reliability of the system.
4) Cost: The less the entire system costs, the better. Additionally,

D. Table

Criteria	RedHat	CentOS	Ubuntu Server
Stability	Highest	High	Moderate
Support	Highest	Moderate	Moderate
Usability	High	High	High
Cost	High	None	None

E. Discussion

As this project does not require high availability or dedicated support, the benefits of buying a subscription for Red Hat Enterprise Linux and negligible, especially being that CentOS has much of the same functionality and the same length support cycle. This project also does not require an operating system that provides newer functionalities sooner like Ubuntu does, stability being a more important requirement.

F. Selection of best option

We have opted for using CentOS as the operating system that builds will be ran on and tested against. It provides the necessary stability and a wide base of support with a ten-year life cycle, while also being easy to configure, deploy and maintain.

VI. Platform for Running Builds

A. Options

1) Virtual Machine: Virtual Machines perform hardware-level emulation and do not allow the guests any access to the underlying operating system or hardware. They're the standard for situations where multiple OS's need to be deployed. Because they emulate all the hardware, there's nearly no risk of a malicious guest system affecting the host system. However, running a virtual machine requires a full operating system to be installed on each individual instance, causing them to take up a relatively significant amount of space per virtual machine compared to to size of the applications running on them. Additionally, their performance suffers due to having to emulate assembly-level instructions instead of executing them directly.

2) Container: Containers perform operating system virtualization, allowing for multiple independent user-spaces to exist at the same time. Because they emulate parts of the operating system and not the physical hardware hardware, containers still need to allow some level of direct hardware access for applications to run properly. They also use some libraries provided by the host operating system instead of providing their own. As a result, they tend to be much smaller than a full-fledged virtual machine and run somewhat faster. A relatively new technology compared to virtual machines, they're rapidly gaining popularity in situations where hardware emulation is not required and restricting access to the host operating system is not a concern.

3) Bare metal: Bare metal servers would have no virtualization or emulation; builds and tests would be ran directly on the hardware itself. The benefits to bare metal are speed and security; no emulation layer exists between the applications and the hardware and there is no host operating system for a malicious guest to interfere with. Builds would also be unable to hoard system resources, preventing a situation where one build slows down another. The single major downside is the requirement for a full physical server per each build that would be ran, greatly restricting the ability for multiple builds to be ran in parallel

B. Goals for use

The goal of selecting the platform the builds would interact with is to allow builds a host system to be ran and tested on while providing reliability and security for the underlying system.

C. Criteria for Evaluation

1) Scalability: Ideally, it should be easy to scale the system up to allow a large number of simultaneous builds.
2) Performance: Our selection should not significantly decrease the performance of the build-test process.
3) Security: Being that we will compiling and testing untrusted and unverified code, it is important for the selected platform to minimize the ability of the build to negatively affect the host.
4) Reliability: It is preferable that one build does not use up so many resources as to significantly reduce the speed of other builds being ran.

D. Table

Criteria	Virtual Machines	Containers	Bare metal
Scalability	High	Highest	Low
Performance	Moderate	High	Highest
Security	High	Moderate	Highest
Reliability	High	High	Highest

E. Discussion

Although bare metal servers have undeniably the best performance, security and reliability, they simply do not have the scalability required for a project like this. Depending on storage space and resources, virtual machines and containers could potentially run upwards of one hundred builds simultaneously on a single server, negating the performance benefits. Furthermore, virtual machines provide nearly the same level of security and safety that a bare metal server provides, while containers can nearly match them in terms of performance.

Between virtual machines and containers, virtual machines provide better security and have years of support behind them, while containers provide better performance; virtual machines can have a performance penalty upwards of ten percent due to overhead.[4] Both virtual machines and containers allow for limiting the resources provided to a running instance, meeting the reliability requirements. They are also both exceedingly easy to configure for automated deployment, making them highly scalable.

F. Selection of best option

Although virtual machines can have a noticeable performance penalty due to overhead, this can be reduced with proper tuning. Furthermore, we believe that the better isolation and security provided by virtual machines can easily outweigh the performance penalties. Therefore, we have decided that virtual machines are the ideal platform for running the builds and tests on.

VII. Configuration File Formats

A. Options

1) YAML:

2) JSON:
3) XML:

B. Goals for use

The configuration file will be included in a Git repository and will be how the system determines builds and tests are ran and interpreted. It should use a format powerful enough to accurately express all the information that the system will require without being unnecessarily complicated.

C. Criteria for Evaluation

1) Expressiveness: The file format needs to be able to sufficiently express the users needs for building and testing their project.
2) Readability: The ideal file format will be easy for a user to create and modify to suit the needs of their project.

D. Discussion

E. Selection of best option

VIII. Login/Authentication

A. Options

1) Google Login Plugin: The Google Login Plugin uses google account for authentication.
2) Active Directory Plugin: The Active Directory Plugin is a plugin for Jenkins and it uses Active Directory to authenticate the username and password.
3) Jenkins Login: The Jenkins login is simply a Jenkins account and users are able to create it on the Jenkins website.

B. Goals for use

The goal for having an authentication is to provide our users a safe and secure place to build their project and to use our continuous integration service. It will help prevent hackers from obtaining private information that could be detrimental to our users. This is a security measure that is absolutely necessary for our project.

C. Criteria for Evaluation

1) Security: Security is what keeps users confidential information such as source code from being leaked.

D. Table

Criteria	Google Login Plugin	Active Directory Plugin	Jenkins Login
Security	Yes	Yes	Yes

E. Discussion

The google login plugin is a really straight forward method for authentication. Users can simply use their google accounts to login. This method requires an OAuth 2.0 credentials from the Google Developers Console and there are clear instructions on the internet to accomplish this task [5]. The second option is to use the active directory plugin. This plugin is used with Jenkins to authenticate the username and password through active directory [6]. The third option is to use Jenkins login. Users can simply create a Jenkins account and have access to its functionalities. They will also have the ability to give authorizations to different accounts for their projects.

F. Selection of best option

The best option is to use Jenkins login. We intend to build our continuous integration service project on a POWER8 architecture using a Jenkins backend; the most reasonable approach is to use Jenkins own login system.

IX. Frontend/Web frameworks

A. Options

1) Tomcat: Tomcat is an open source implementation of Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies.

2) Glassfish: GlassFish is an open source application server; more specifically it is for the Java EE platform (Java Enterprise Edition).

3) Jetty: Jetty provides a Web server and javax.servlet container, and it supports many componenets such as HTTP/2, WebSocket, OSGi (Open Service Gateway Initiative), JMX (Java Management Extensions), JNDI (Java Naming and Directory Interface), JAAS (Java Authentication and Authorization Service) and many other integrations.

B. Goals for use

The goal for having a web framework is so that our users have a platform to use our continuous integration service. We will provide a simple and easy to use user interface (UI) so that our users would not have a complicated time using our services.

C. Criteria for Evaluation

1) Usability: Having an easy-to-use user interface can give our users an easier time to use our service and it will help reduce confusion.
2) Security: Security is important for our web framework because we don't want hackers to gain access to our services and obtain confidential information.
3) Stability: Stability is needed because our service need to maintain at a functional state at all time so that it doesn't hinder our users from working
4) Load time: The page load time should load at a reasonable speed so that our users would not have to wait a long time using our service.

D. Table

Criteria	Tomcat	Glassfish	Jetty
Usability	High	High	Moderate
Security	Moderate	High	High
Stability	Highest	High	High
Load time	High	High	High

E. Discussion

Tomcat is very popular and it is known as a lightweight application that offers all the basic features required running a server. It is an open source project so it is cost free. It is highly flexible because it allows users to tweak their code as they see fit, and it has many built-in features [7]. The second option is Glassfish and it is also an open source project, which means that it is free of cost. It has high performance and it was considered the fastest open source application server, according to SPECjAppServer2004 benchmark results [8]. Jetty provides a Web server and javax.servlet container. They have a heavy focus on multi-connection HTTP and features such as SPYDY, which can significantly reduce page load latencies [9].

F. Selection of best option

Overall, all three of the options are great, but the best option to use is Tomcat. Glassfish has a lot of features and is the biggest out of the three options, but we are only using it for our frontend, so most features are unnecessary. It has a big memory footprint, which means that it would consume more resources. Jetty is a very light application and it consumes the least amount of resources but it is too small for our project. Tomcat is ideal because it sits in between these two options and it offers a good amount of features to use.

X. Tracing State of builds/tests

A. Options

1) Build Monitor Plugin: The Build Monitor Plugin is a plugin in Jenkins that provides a detailed view of the status of selected Jenkins jobs.

2) Lava Lamp Notifier: The Lava Lamp Notifier is a notifier that indicate job status using a Lava Lamp.

3) Radiator View Plugin: The Radiator View Plugin is a plugin in Jenkins and Hudson that provides a job view, which displays project status.

B. Goals for use

The goal to having the functionality to trace the build and test state is so that our users can see the status of their projects. This will reveal the success or failure of projects that were build by our service and it will show where the failures have occurred.

C. Criteria for Evaluation

1) Usability: Having an easy-to-use user interface (UI) allows our users to operate our service with minimal issues.
2) Feature to display pass or fail builds:The ability to display pass or fail build is the main purpose for this technology so this is a must have
3) Feature to display progress of builds: The feature to display progress when building a project is helpful for our users to see how much longer it will take until their build is complete
4) Feature to display where an error has occured for build fails: The feature to see where an error has occured in the build process is another important critieria; it will help developers ave time from manually tracking down the problem.

D. Table

Criteria	Build Monitor Plugin	Lava Lamp Notifier	Radiator View Plugin
Usability	Yes	None	Yes
Display pass or fail builds	Yes	Yes	Yes
Display progress of builds	Yes	None	None
Display where an error has occured if build fails	Yes	None	None

E. Discussion

The build monitor plugin supports many features and incorporates other plugins as well. The features in this plugin include:

- Displays the status and progress of selected jobs, the view is updated automatically every couple of seconds using AJAX. No ‘Enable Auto Refresh’ needed.
- Displays the names of people who might be responsible for ‘breaking the build’.
- Supports the Claim plugin, so that you can see who’s fixing a broken build.
- Supports View Job Filters, so that you can easily create Build Monitors for ‘slow builds’, ‘only failing’, etc.
- Supports Build Failure Analyzer, so that you know not only who, but also what broke the build.
- Supports CloudBees Folders Plugin, so that you can have project- and team-specific nested Build Monitors.
- The number of columns and size of the font used is easily customizable, making it trivial to accommodate screens of different sizes.
- UI configuration is stored in a cookie, making it possible to display different number of columns and using different font size on each of the screens at your office.
- Can work in a colour-blind-friendly mode
- [10]

The lava lamp notifier is a really simple design created to indicate if a build passed or fail. The only necessary component to use this type of method to track our build status is a USB LED light. The light will simply light up green for pass and red for fail [11]. This is not ideal for our type of project because our users would like to know more information than just a pass or fail notification. The third option is the radiator view plugin. This plugin is somewhat similar to the build monitor plugin but with less features. It will display all the project that are building and indicate pass or fails on screen [12].

F. Selection of best option

The best option to use is the build monitor plugin. This technology has the most features out of all the other options I listed, and all the features are very useful for our users.

REFERENCES

[1] Red Hat. (2016, November) Red hat enterprise linux life cycle. [Online]. Available: https://access.redhat.com/support/policy/updates/errata

[2] CentOS Project. (2016, November) Centos product specifications. [Online]. Available: https://wiki.centos.org/About/Product

[3] Canonical. (2012, November) What's the difference between desktop and server? [Online]. Available: https://help.ubuntu.com/community/ServerFaq#What.27s_the_difference_between_desktop_and_server.3F

[4] F. W., F. A., R. R., and R. J., "An updated performance comparison of virtual machines and linux containers," IBM Research Division, Tech. Rep. RC25482, July 2014. [Online]. Available: http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B

[5] recampbell Campbell. (2015, Nov) Google login plugin. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Google+Login+Plugin

[6] K. Kawaguchi. (2016, Oct) Active directory plugin. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Active+Directory+plugin

[7] A. Mangat. (2010, Sept) Top 7 features in tomcat 7: The new and the improved. [Online]. Available: http://www.developer.com/java/web/article.php/3904871/Top-7-Features-in-Tomcat-7-The-New-and-the-Improved.htm

[8] Oracle. (2010, Jun) The oracle glassfish server advantage for small businesses. [Online]. Available: http://www.oracle.com/us/products/middleware/application-server/glassfish-server-adv-sbiz-wp-080573.pdf

[9] webtide. (2016) Why choose jetty. [Online]. Available: https://webtide.com/why-choose-jetty/

[10] J. Molak. (2016, Nov) Build monitor plugin. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Build+Monitor+Plugin

[11] E. Randall. (2010, Nov) Lava lamp notifier. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Lava+Lamp+Notifier

[12] M. Howard. (2011, May) Radiator view plugin. [Online]. Available: http://wiki.hudson-ci.org/display/HUDSON/Radiator+View+Plugin