HITACHI
Inspire the Next

WHITE PAPER

Hitachi Content Platform as a Continuous Integration Build Artifact Storage System

A Hitachi Data Systems Case Study

By Hitachi Data Systems

March 2015

Contents

Executive Summary

There are many continuous integration (CI) tools for building a software product. While all of them excel at building source code into usable artifacts, they suffer when it comes to the storage of the resulting artifacts. The primary use of these tools is continuous integration, not storage, so it is not unexpected that the storage features of these tools are lackluster in comparison to their CI capabilities. At best, the storage component of these tools can be considered a bonus and, at worst, a complete afterthought.

The biggest concern with storing build artifacts directly on the CI tool is the lack of high availability. Once the build artifacts are produced and quality assurance (QA) users are notified of the new build, the artifacts must be readily available at all times, as QA users and the continuously running automated test cases must be able to consume the artifacts as needed. Whether the CI tool must be restarted, an upgraded, re-configured, a plugin was added, a bad build occured, or regular hardware maintenance is necessary, build artifacts must always be available. Also missing from CI tools is the capability to seamlessly grow the storage capacity without creating downtime for the end users. Another must-have feature is the ability for end users to easily query for specific build artifacts given a set of query parameters. Lastly, administrators need automated ways to ensure build artifacts of high importance cannot be accidentally deleted, while tiering off or deleting artifacts of lesser importance after a certain period of time.

All of these concerns are caused by the tight coupling of the CI responsibilities with those of the build artifact storage responsibilities. Relieving the CI tool from the burden of storing the artifacts allows the developers to innovate with the CI tool without impacting the QA team or automation tools. The only remaining question is where to store the build artifacts, if not on the CI tool.

Hitachi Content Platform (HCP) resolves the issues listed above and provides excellent storage features. By design, HCP is architected in a multi node fashion, enabling high availability through redundancy at the software and hardware levels. HCP is also extensible, allowing for simple storage additions without taking the system offline. All objects are indexed and query-able through the metadata query engine. Customizable retention and disposition policies let administrators determine which build artifacts stay on the system, specify how long they are retained, and protect important artifacts from accidents. HCP also uses technologies such as compression and duplicate elimination to cut down on overall storage consumption.

Your build artifacts are important to you, and you can trust HCP to keep those artifacts available and protected.

Introduction

This white paper highlights the problems posed from storing build artifacts inside the CI tool that built them, and how the HDS engineering team resolved these problems in a scalable manner.

Intended Audience

The intended audience is any engineering team using a continuous integration tool to build their products.

Terminology and Technologies Used

- Build pipeline or process: The process by which source code is compiled, unit tested, and packaged into a consumable artifact. This process is also referred to in the industry as “continuous integration,” or “CI” for short.
- Continuous integration: See “build pipeline.”
- Build artifacts: The build artifacts refer to the resulting consumable package produced by the build process. This artifact can be distributed to the QA team for qualification or end users for general availability.
- Build: Refers to a single, versioned set of artifacts representing the state of the product at the point in time at which the specific build occurred.
- Jenkins: An open-source continuous integration tool, which builds source code into consumable artifacts. HDS uses a local on-premises install of Jenkins CI.
- Automation Test Framework (ATF): A homegrown test harness that takes the product build artifacts, installs them into on a test system, and automatically tests and exercises all use cases of that product. It identifies all defects to be fixed prior to general availability release.

Relevant Hitachi Content Platform Concepts and Terminology

- System: Refers to the entire HCP cluster as a whole, including the individual HCP nodes (servers), any local, SAN-attached, or extended storage, and all HCP software components, such as tenants, namespaces and objects.
- Tenant: HCP supports multitenancy, allowing the HCP system administrator to divide the system into separate tenants, each with their own management, access control, storage quotas and unique settings.
- Namespace: An HCP tenant can also be subdivided by the tenant administrator into namespaces, each with their own access controls, quotas and unique settings. The namespace acts as the container in which objects are stored on the HCP system.
- Data protection level (DPL): The DPL is a namespace setting that can control how many copies of each object are being stored for redundancy purposes. For example, a DPL setting of 4 will mean each object will have four copies spread across four different nodes.
- Content classes: These describe how the custom metadata associated with HCP objects should be indexed. The content class defines which pieces of information will be indexed for future searches via MQE and labels them accordingly.
- Metadata query engine (MQE): Parses the custom metadata and indexes the “interesting” information as defined by the content classes. This data is then searchable by end users via the search user interface (UI) or via the RESTful MQE API.

Statement of Problem

After adopting Jenkins as their new CI tool, the HDS engineering team was impressed with the flexibility and power the tool afforded them when building source code into a usable product. Unfortunately, for all its strengths in the continuous integration department, Jenkins underperformed on the build artifacts storage side in the following ways:

- Frequent downtime. Even though Jenkins supports a distributed model for building software, all resulting build artifacts are stored back on the master server. Whenever any maintenance is necessary on the Jenkins master (whether it be software or hardware upgrades or restarts, and so forth) all automated and manual testing is halted due to the unavailability of the build artifacts hosted on the Jenkins master.
- Slowed innovation. Relying on Jenkins to host build artifacts establishes a single point of failure for the continuous integration system, and the testing framework. This discourages developers from trying out new approaches, new plugins, or making any changes on Jenkins out of fear that these changes might impact the entire testing pipeline instead of just the CI system, stifling innovation.
- Unwanted complexity. Jenkins' API uses its back-end "build ID" (for example, 123) to uniquely identify a specific product build. The "custom build number" plugin can be used to generate a full version number (for example, 7.1.0.123) for use in the products. This leads to a disconnect between the version numbers expected by the end users and the ones returned by the Jenkins API, resulting in added code complexity on the client-side to work around this issue.
- Concealed dependencies. By having the automation framework interact directly with the Jenkins APIs, the testing framework is tightly coupled the with the CI tool, resulting in vendor lock-in. Any future decision to change which CI tool to use will also come with major re-work of critical components of the testing framework.
- Inadequate storage features. While Jenkins excels at building software from the source, its artifacts storage capabilities are lackluster and extremely limiting. All storage management activities are left to the administrator as manual operations, with few features to help them control the explosion of data.
- Scalability issues. As the number of build artifacts grows, it is necessary to expand the storage capacity. As Jenkins administrators know, adding more storage to a Jenkins master node is not usually an online operation: It relies on the underlying operating system and filesystem type. Jenkins also offers no features in the way of compressing existing artifacts or deduplicating repeated files to save on overall storage use.

Requirements for a New Solution

As the organization grew and the number of build jobs and build artifacts exploded, the team realized that having Jenkins store all of the build artifacts was not scalable. The current structure had essentially tightly coupled two distinct and very different tasks into a single system:

- Continuous integration.
- Storage of build artifacts.

The team knew that if they could somehow separate these two ideas and come up with two distinct solutions for each of the tasks above, Jenkins would continue its primary mission statement of building software from source code and then handing off all storage responsibilities to another tool to figure out the best possible way to store these builds artifacts. This would also avoid vendor lock-in, allowing the team to easily swap out Jenkins for another CI tool should they ever decide to migrate away from Jenkins.

Desired attributes for an ideal build artifacts storage solution include:

- Highly available. The ideal storage system for build artifacts needs to be online as much as possible, since the entire testing framework is reliant on the builds artifact being available for download. The system should be resilient against hardware upgrades, software upgrades and unexpected hardware failures.
- Simple. Clients should have an easy way to "tag" each build artifact with the relevant information that describes that artifact, such as the version number. These "tags" should be searchable such that a different client or end user can easily find the corresponding build artifacts given a version number.
- Smart storage management functionality. To save on storage costs, the ideal solution should automatically compress and deduplicate the build artifacts that it is storing. Administrators should also have the capability to define policies that age out older, unwanted builds while protecting the new, more important builds from user accidents or rogue scripts.
- Scalable. Administrators should be able to add more storage capacity as needed without causing an outage for end users and clients.

The Hitachi Content Platform Solution

After compiling the requirements above, the team concluded that Hitachi Content Platform was an optimal fit for these use cases. The HCP features described below are an excellent repository for build artifacts:

- High availability. HCP is a collection of a minimum of four nodes working together in a cluster to guarantee the availability of all data stored on the system.
- Extensibility. A major characteristic of HCP is the ability to seamlessly add more storage while the system is still online. This allows administrators to extend the storage capacity and easily scale up without causing an outage for end users.
- Query-ability. HCP software comes with a query interface called the metadata query engine or MQE. This allows end users to search for specific objects that match the given criteria via either the object's POSIX metadata or any custom metadata defined by the application (or user) on ingest time.
- Density. HCP offers an astounding amount of usable disk storage in a small physical footprint for the data center it resides in. With the release of the HCP S-series nodes, even more dense configurations per node will be possible. HCP can also integrate seamlessly with existing Hitachi storage arrays on the back end.
- Multitenancy. HCP allows administrators to act as service providers and carve up the existing storage into distinct namespaces, each with configurable quotas and settings. This is especially useful as the same HCP system can also be used for storing other data sources such as backups, documents, test results, or even a generic storage share, in addition to storing the build artifacts.

Data Protection

Hitachi Content Platform introduces the concept of "data protection level," a setting that defines how many copies of an object should be kept at all times on the system. Configuring a DPL of two or higher allows the systems administrator to perform upgrades of the hardware or HCP software without any visible impact to the end users reading and writing objects to the system. The protection service, which runs regularly on the system, ensures that the DPL value is enforced on all objects. It makes extra copies when a node becomes unavailable for extended periods of time, and it removes the extra copies when necessary.

To also protect against entire site outages or other catastrophic disasters, a second HCP can optionally be configured as an off-site replica of the first. The systems administrator can configure which tenants and namespaces to replicate and what time of day the data should be replicated. This feature can also be used to distribute build artifacts to another geographically distant data center.

For protection against user mishaps or scripts gone rogue, HCP offers the ability to set retention policies on ingested objects. This capability guarantees that objects cannot be deleted under any circumstance until the predefined date has lapsed (configurable either as a specific date or time differential). Retention policies guarantee that each build artifact is kept for a minimum amount of time (for example, each build is kept for three months after they are built). Optionally, a *retention hold* may be issued against specific build artifacts to trump the defined retention date and keep the objects forever.

Smart Data Management

In conjunction with the retention features described above, the disposition feature may be enabled, indicating to HCP that any build artifacts coming out of retention can be cleaned up. In this way, it reclaims the storage space for newer, more relevant builds to take their place.

HCP also allows administrators to configure different tiers of storage pools and define policies around how to tier different objects from one pool to another. This ability can be especially useful for build artifacts that must never be deleted, but are seldom read: These build artifacts can be tiered to a spin-down array to save on power and cooling costs in the data center.

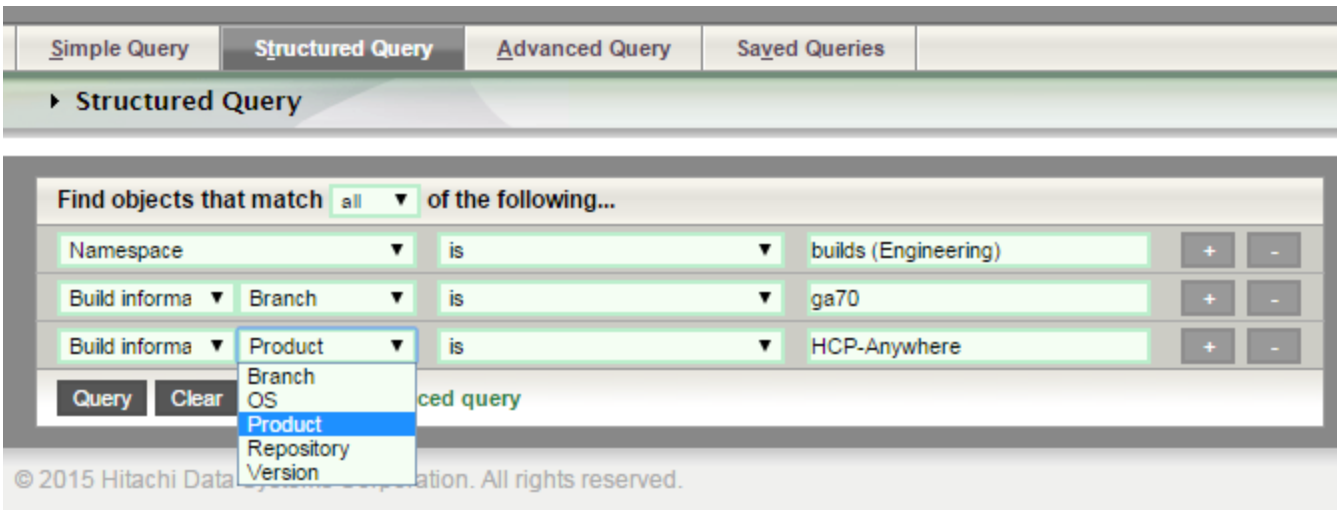
Putting It All Together

Having decided on a solution, the HDS development team set off to integrate the existing Jenkins infrastructure with HCP. A DPL-2 namespace with search and replication enabled was created for the purpose of housing these build artifacts. Once the namespace was configured, the Jenkins jobs were updated to have an additional step at the tail end of their pipelines: uploading the build artifacts to HCP. This step was accomplished using a few simple cURL commands to ingest the build artifacts as well as their corresponding custom metadata via the RESTful HTTP API. The custom metadata is an XML file containing the relevant CI information about the build that produced them. For example it can include the version number, the branch name, the product name, which build server produced it, and so forth.

Once Jenkins was updated to upload all build artifacts to HCP, and tag these artifacts with relevant metadata, HCP had to be configured to index these objects. This was done using content classes: providing HCP with a sample custom metadata XML teaches the platform the standard XML format for a build artifact's metadata and how to parse it. The parsed data then becomes indexed along with the object. This process allows for users or clients to programmatically search for the build artifacts they are looking for using any of the parameters defined in the custom metadata XML file.

The last step was to update the test clients to go looking for the build artifacts on the HCP. To accomplish this, the MQE search engine was enabled at the HCP system-level. Finally, the Jenkins API calls were replaced with RESTful MQE queries to HCP, also using cURL. In addition to providing a robust and extensible API for querying for build artifacts, MQE also provides a Web console. End users, such as the QA team, can use this console to easily search for the build artifact that they are looking for (see Figure 1).

Figure 1. MQE Web Console



Conclusion

An ideal solution for the storage of build artifacts in a rapid development, continuously integrated environment, Hitachi Content Platform enables reliable automation testing frameworks.

The HDS engineering team has been successfully using HCP as a storage platform for their build artifacts. HCP has ingested a total of 37.5TB of data across 50 million objects. It has reclaimed 12.25TB in storage savings thanks to compression and deduplication. Further, there have been 0 hours of downtime for end users in 1.5 years since Jenkins started publishing build artifacts to HCP.

Hitachi Data Systems

Corporate Headquarters
2845 Lafayette Street
Santa Clara, CA 96050-2639 USA
www.HDS.com community.HDS.com

Regional Contact Information
Americas: +1 408 970 1000 or info@hds.com
Europe, Middle East and Africa: +44 (0) 1753 618000 or info.emea@hds.com
Asia Pacific: +852 3189 7900 or hds.marketing.apac@hds.com

WP-533-A Hitachi Data Systems April 2015