PURESTORAGE®

JFrog

TECHNICAL WHITE PAPER

Modernizing JFrog Artifactory on S3-compatible FlashBlade

Simplify and accelerate Artifactory repositories.

Contents

Introduction

Software is constantly changing the business landscape and consumer experiences with better quality products and services. Software development and delivery cycles generate and consume data that requires processing and storage during various workflow stages. But storing, managing, and protecting data in silos leads to higher costs and manageability overhead. Universal binary package managers, like JFrog Artifactory, integrate with various workflows in the software development lifecycle (SDLC) on Pure Storage® FlashBlade™ for scalable performance, cost efficiency, and manageability.

Pure Storage technology can complement and enhance a JFrog Artifactory high-availability implementation using NFS-based filesystems for PostgreSQL database and S3-compliant buckets for FlashBlade repositories. Additionally, most modern application-development environments use Kubernetes to optimize the infrastructure resources and provide flexibility in software development and delivery workflows. You can also implement Artifactory in a Kubernetes cluster using Pure Service Orchestrator™ to automate mounting of the PostgreSQL database and Artifactory home (/var/opt/jfrog/artifactory) directory over NFS on-demand before copying the data over to a S3 bucket configured on FlashBlade data platform.

Continuous Everything

In recent times, enterprises and small-to-medium-size businesses are rapidly transitioning from agile methodologies to DevOps processes that involve continuous integration (CI), continuous delivery (CD), and continuous testing (CT) to drive innovation and faster time to market. DevOps is about:

- **Speed of delivering but also maintaining the software quality.** The iterative process of making small changes to the code, testing, and identifying bugs at an early stage of the development cycle during the CI/CT process generates hundreds and thousands of binary versions and packages that are continuously deployed and released into production.
- **Adopting CI/CD processes but also requires automation.** The automation not only applies to various workflow pipelines, but also to consume as code using REST APIs. It requires infrastructure automation to process, store, access, and protect data. Most CI/CT environments require a scalable infrastructure over local servers that can run parallel software builds. Software builds and their dependencies require a data platform to store and manage binary packages.

Source Code and Binaries

Source code and binary repositories are two distinct parts of the CI pipeline. The application source code could be written in any language like C/C++, Python, etc. You have to track and manage every change to source code. Tools like Git and its various flavors, Perforce, SVN are some of the commonly used tools that are used for version source control.

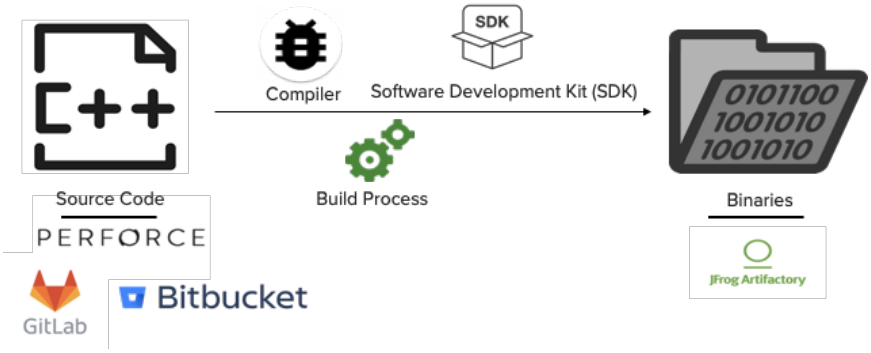


Figure 1. Build process from source code to binaries.

You need to compile the source code using various artifacts like compilers, interpreters, and SDKs during the build process. The outcome of the build process results in many versions of the binaries that you need to package, store, and manage. JFrog Artifactory is an artifact and binary package manager. It provides effective versioning for the different artifacts and binary packages that are used and generated during the CI process.

JFrog Artifactory and Pure Storage

JFrog Artifactory's universal binary package manager not only stores and manages the binaries generated during the CI process, but also automates the promotion of the binaries for continuous deployment during the CD pipeline. JFrog also has additional products designed to enable and enhance software development and delivery.

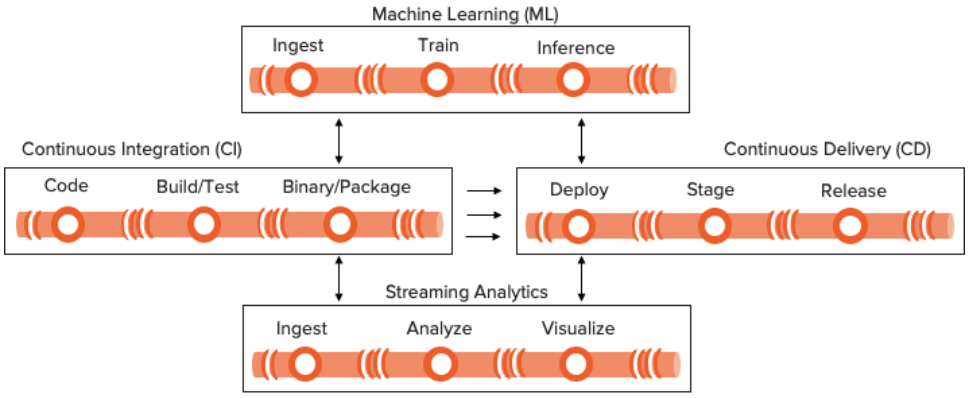


Figure 2. Assembly line for different pipelines.

Software development is no longer limited to develop and deliver pipelines. Other extended workflows like analytics, artificial intelligence and machine learning (AI/ML) have blended into the CI/CD pipelines as a part of a bigger assembly line driving modern businesses. In Figure 2, a binary package manager like JFrog Artifactory plays a pivotal role to connect with the rest of the pipelines until the binary is finally released into production. For performance reasons, it is recommended that you don't store binaries in source code repositories like Git.

All the different stages of the larger assembly line generate data that you need to store, process, manage, and protect in a cost-efficient manner. FlashBlade has multiple advantages, including that it:

- Provides cost efficiency with a smaller footprint for active and dormant datasets using data-reduction features like compression.
- Supports NFS-based filesystems and S3-compliant objectstore on the same data platform.
- Provides data access and continuity for various application workflows to scale on demand with respect to capacity and performance based on the workload requirements.

Architecture and Data Challenges

> ***Note:*** *For this document, we've validated Artifactory Enterprise because Artifactory-HA supports S3-compatible buckets on data platforms like FlashBlade. Artifactory Pro and Artifactory Pro X do not support S3-compatible object stores for on-premises storage. This document uses the terms repository, filestore, and filesystem interchangeably.*

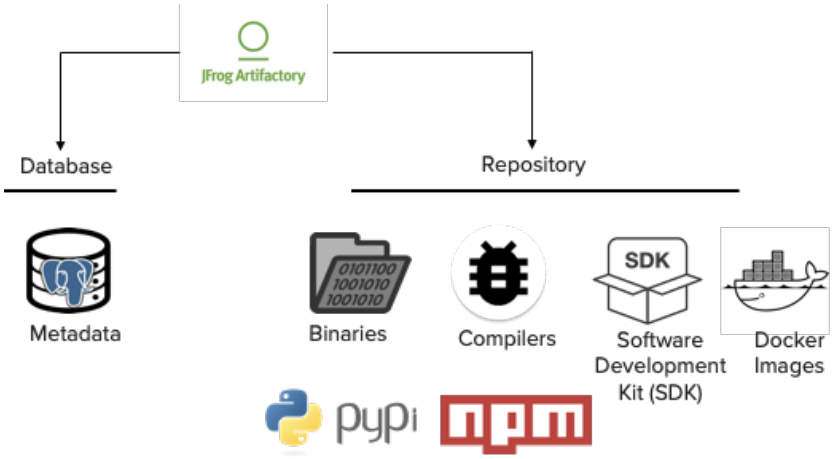


Figure 3. JFrog Artifactory components.

Artifactory ideally consists of two main components:

- A database that stores all the metadata information and compliance rules. The path to reach the respective artifacts and binary packages is part of the metadata information stored in the database. The database also stores version history of the artifacts required for the software build process, build data and time of the new binaries, metrics for code coverage, documentation, Shard information, etc.
- The repository, or filesystem, stores artifacts like interpreters, compilers, SDKs, binaries, and other packages using checksums. When you upload a file to Artifactory, a unique checksum is first calculated for every file. The checksum generated is stored in the respective folder with the first two characters as the name of the folder. It then creates a database entry and maps to the file's checksum to the path to which it was uploaded in a repository. JFrog Artifactory supports generic and custom repositories for Docker, PyPi, NPM among many others as shown in Figure 3.

In many production environments JFrog Artifactory is installed on local servers as illustrated in Figure 4. You can set up the JFrog Artifactory data in a separate silo from the rest of the workflow, which would require moving data between the different workflows in the DevOps pipeline.

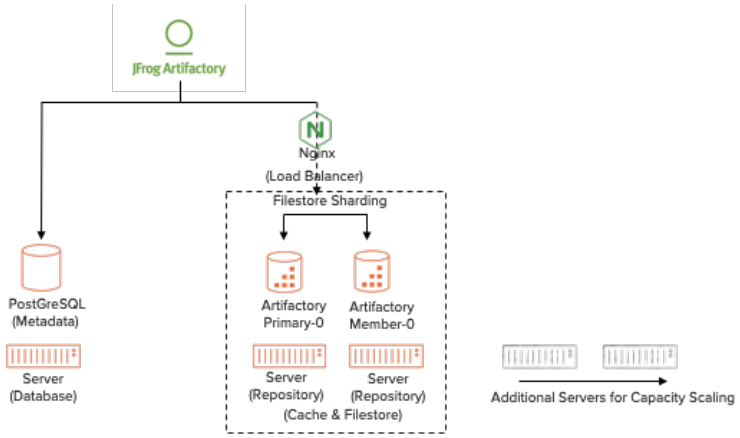


Figure 4. JFrog Artifactory HA environment.

Artifactory architecture supports Filestore Sharding. The files/checksums uploaded are first staged in the data/eventual(_pre) folder before they're written to the persistent storage as shown in the directory structure in Figure 5. The eventual works as a write cache for the checksums before finally being written to two local storage locations: shared-fs-1 from both the members in the high-availability configuration.

The first node "artifactory-ha-primary-0" in the high-availability cluster first writes the copy of the checksum to the local storage in the shard-fs-1 folder. A subsequent internal process is triggered to the second node "artifactory-ha-member-0" in the HA cluster to write the same checksum into shard-fs-1 in its own local storage. The internal process eventually syncs both members of the high-availability cluster to which the artifact is written successfully in both the shard-fs-1 locations.

```
root@init47d-v37:~# docker exec -it b84eb300cc98 /bin/bash
artifactory@artifactory-ha-artifactory-ha-primary-0:/$ cd /var/opt/jfrog/artifactory/
artifactory@artifactory-ha-artifactory-ha-primary-0:/var/opt/jfrog/artifactory$ ls -al
total 4
drwxrwsrwx    1 root     artifact         0 Jan 31 22:10 .
drwxr-xr-x    3 root     root          4096 Oct  8 14:12 ..
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:10 access
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:10 backup
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:10 data
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:10 etc
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:11 logs
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:10 metadata
drwxrwsr-x    1 artifact artifact         0 Jan 31 22:10 replicator
```

Figure 5. JFrog Artifactory home location directories

Most of the folders are located under the Artifactory home directory /var/opt/jfrog/artifactory including the read/write cache locations. The data/cache(_pre) or the cache-fs or the read cache works as a read cache to reduce download latency for the artifacts and the binaries from the backend persistent storage. If the artifacts are not accessed for a long time, they are completely evicted from the cache. The default size of the cache-fs is 5GB.

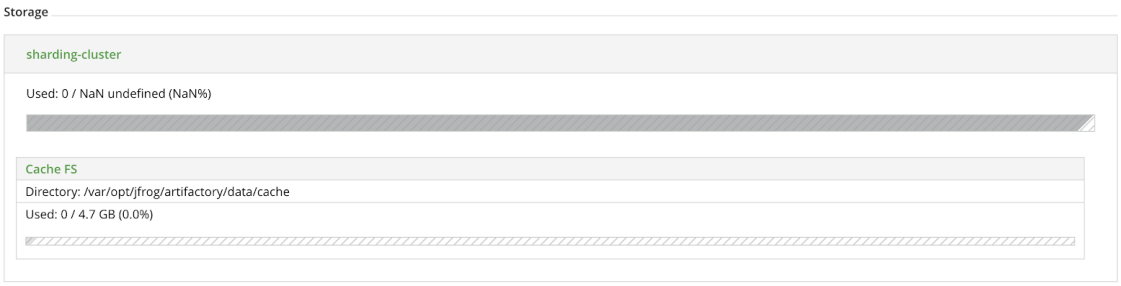


Figure 6. Default read cache size in JFrog Artifactory.

JFrog Artifactory uses Tomcat as the web server and nginx as the load balancer as an entry point to the Artifactory HA nodes to distribute the user requests to them. Designing Artifactory on local servers leads to challenges in the data layer, including:

- Redundant copies of checksums on local storage leads to a higher volume of data footprint in the production environment.
- Adding more file systems to the Artifactory nodes for capacity scaling is easy in the shared filesystem in Artifactory but requires additional servers that lead to management overhead.
- Scaling or increasing the read cache size (default 5GB) for low latency artifacts and binary downloads will require adding servers for more cache space.
- Data has gravity. Data mobility and managing different islands of data platforms are an overhead.
- The Nginx node may sometimes exhibit slow performance when a high number of requests hit the Artifactory nodes at the same time to upload and download large numbers of artifacts and binaries.
- Backup targets can get inflated as Artifactory stores multiple copies of the artifacts in all the repositories that referenced it.

JFrog Artifactory on FlashBlade: Validated Architecture

FlashBlade as a data platform supports both NFS and S3-compliant object store data formats. JFrog Artifactory configured in a Kubernetes cluster using PSO provides data from the application layer to the end users. We configured JFrog Artifactory components in a hybrid mode: the database and the Artifactory cache was configured over NFS, and the persistent storage for the filestore or repositories used S3-compliant buckets on FlashBlade.

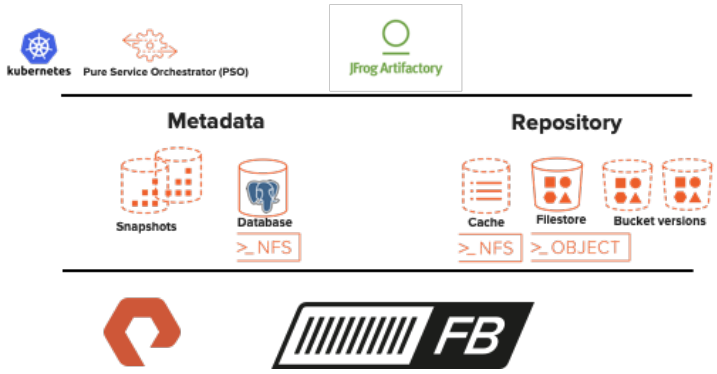


Figure 7. JFrog Artifactory architecture on Pure Storage FlashBlade.

Implementing JFrog Artifactory on FlashBlade has some unique advantages over local storage. The advantages and benefits setting up Artifactory on a shared data platform are listed below:

- PSO provides two different storage classes, “pure-file” and “pure-block,” that can provision storage on-demand from Kubernetes cluster.
- Storage Class “pure-file” is used to set up Artifactory Enterprise components like the database and the filestore on FlashBlade.
- FlashBlade has been the recommended data platform for other integrated workflows in the development, delivery, and other extended pipelines. JFrog Artifactory connects to different data pipelines that are part of the larger assembly line. Data is reused, managed, and protected on a standard data platform. There is no overhead in moving data between workloads.
- Adding more capacity for Artifactory data growth leads to higher server sprawl as FlashBlade provides the ability to scale capacity on demand and eliminating any management overhead.
- NFS provides more resilience from failures and high availability (HA) to the end users to access the Artifactory database. The artifacts and the binaries in the repository cannot be reached without the Artifactory DB access. You can scale the number of pods in the Kubernetes cluster seamlessly with the Artifactory database setup over NFS. In a non-Kubernetes environment, you can configure more than one bare-metal or virtual machine (VM) node to point to the PostGreSQL DB over NFS.
- You can create the NFS shares for Artifactory database and read/write caches on demand from Kubernetes cluster using PSO during the Artifactory setup. No manual NFS mount is required to setup Artifactory in a Kubernetes cluster using PSO.
- The filestore or the repository consists of the read/write cache and S3 is used as the back end persistent to finally store the artifacts and the binaries.
  - The read/write cache is configured on FlashBlade over NFS, which allows us to scale the cache size for better upload and download performance speed instead of adding more local servers.
  - Configuring the artifactory-ha-primary-0 and artifactory-ha-member-0 on FlashBlade over NFS does not require filesystem sharding and redundant datasets. Both members of the HA cluster can point to the same NFS share for the cache and the S3-complaint bucket for the repositories.
  - The S3 compliant bucket on FlashBlade is configured to provide persistent storage to generic and custom binary packages.
- Array-level features like snapshots and bucket versioning for database and the repositories, respectively, improves the cost efficiency for Artifactory data backup and recovery.

JFrog Artifactory Validation: Test Results and Observations

As referred in Figure 7, Artifactory Enterprise 6.13.1 was configured in a Kubernetes cluster and provisioned on FlashBlade using PSO. The PostGreSQL DB for Artifactory is configured using pure-file storage class from the Kubernetes cluster. The data/cache directory under the Artifactory home location /var/opt/jfrog/artifactory is configured over NFS for scalability and reduced data footprint.

```
10.61.169.30:/pure-csi-pvc-8e258006-1ed4-4344-8115-1fb91a979eff on /bitnami/postgresql type nfs
(rw,relatime,vers=3,rsize=524288,wsize=524288,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,mo
untaddr=10.61.169.30,mountvers=3,mountport=2049,mountproto=udp,local_lock=none,addr=10.61.169.30)

10.61.169.30:/pure-csi-pvc-2b89e962-b8ad-43f9-bc7f-ff138677c760 on /var/opt/jfrog/artifactory type nfs
(rw,relatime,vers=3,rsize=524288,wsize=524288,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,mo
untaddr=10.61.169.30,mountvers=3,mountport=2049,mountproto=udp,local_lock=none,addr=10.61.169.30)
```

Figure 8. NFS mounts for JFrog Artifactory PostGreSQL DB on FlashBlade.

FlashBlade provides a standard data platform for various Artifactory components like PostgreSQL DB and read/write cache over NFS and persistent backend storage for artifacts and binary packages over S3. Artifactory Enterprise is capable of handling NFS and S3 requests to FlashBlade.

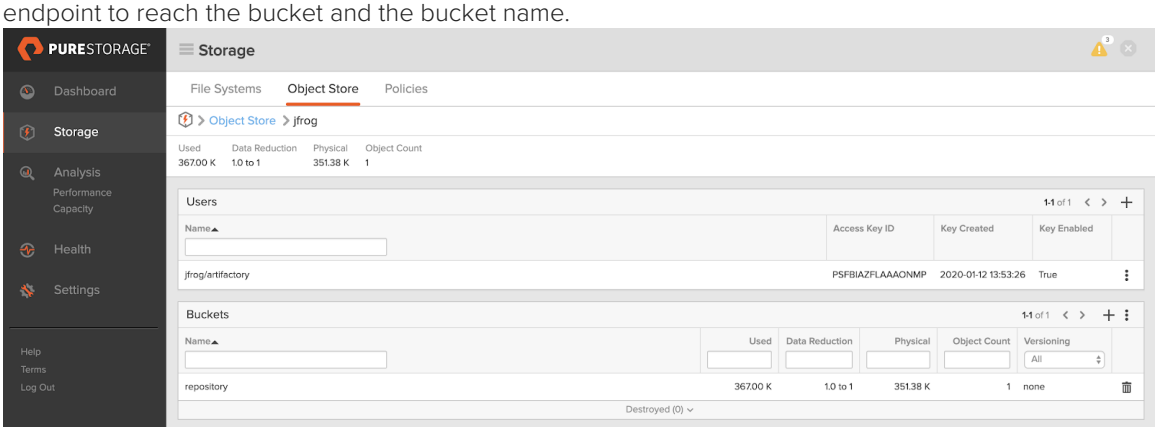
The binarystore.xml file in the /var/opt/jfrog/artifactory/etc/ location needs to be changed to provide the S3 complaint bucket information that was created on FlashBlade.

```
artifactory@artifactory-ha-artifactory-ha-primary-0:/var/opt/jfrog/artifactory/etc$ cat
binarystore.xml
<config version="2">
<chain template="cluster-s3"/>
<provider id="s3" type="s3">
<endpoint>http://10.61.169.31</endpoint>
<identity>PSFBIAZFLAAAONMP</identity>
<credential>2EC30000b/42d9+5966EC7Eac369f20EHGKOANIIPPDFFP</credential>
<bucketName>repository</bucketName>
<httpsOnly>false</httpsOnly>
<property name="s3service.disable-dns-buckets" value="true">
</property>
</provider>
<provider id="cache-fs-eventual-s3" type="cache-fs">
<maxCacheSize>10000000000</maxCacheSize>
</provider>
</config>
```

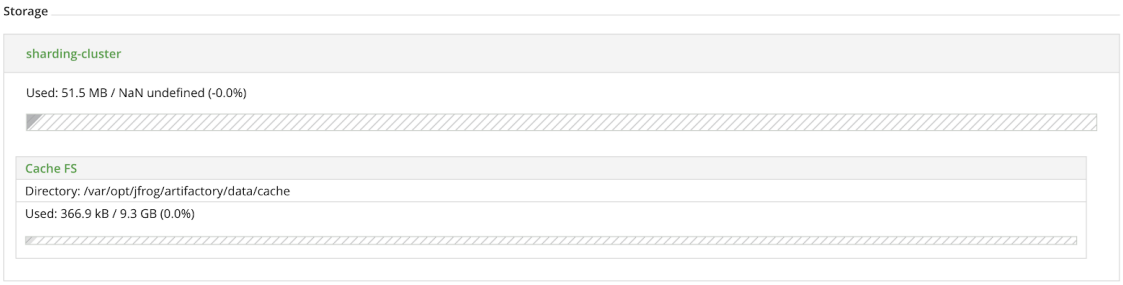
Figure 9. The binarystore.xml file in JFrog Artifactory updated with the S3-compliant bucket on FlashBlade.

In Figure 9, the /var/opt/jfrog/artifactory/etc/binarystore.xml file is used to perform two different functions:

1. Configure the S3-compliant bucket on FlashBlade with the correct bucket credentials (access key, secret key), endpoint to reach the bucket and the bucket name.



2. The read cache size can be increased to 10GB from its default 5GB size for better download speed. The read cache size is configured in bytes the binarystore.xml files shown in the table above.



For the purpose of this testing, we used the default read cache size of 5GB.

The docker containers for artifactory-ha-primary-0 and artifactory-ha-member-0 are restarted for Artifactory to point to the S3-compliant bucket. The repository on FlashBlade is shown in the screenshot for storing the binary packages. Restarting the containers also enforces the new read cache size for Artifactory on FlashBlade.

We tested to measure the download speed from Artifactory configured on AWS S3 and compare the results on S3 compliant bucket from FlashBlade in three different scenarios:

- All the Artifactory components configured on AWS using AWS S3 as the persistent storage.
- Configure Artifactory cache on local servers and repository on S3 complaint bucket on FlashBlade.
- Configure Artifactory cache over NFS and repository on S3 compliant bucket on FlashBlade.

A shell script was used to upload the files with sizes 1k, 10k, 100k, 1M, 10M, 100M, 1G, 10G to a generic-local repo that was created in Artifactory for the above listed scenarios. All the cache was removed before testing the download speed.

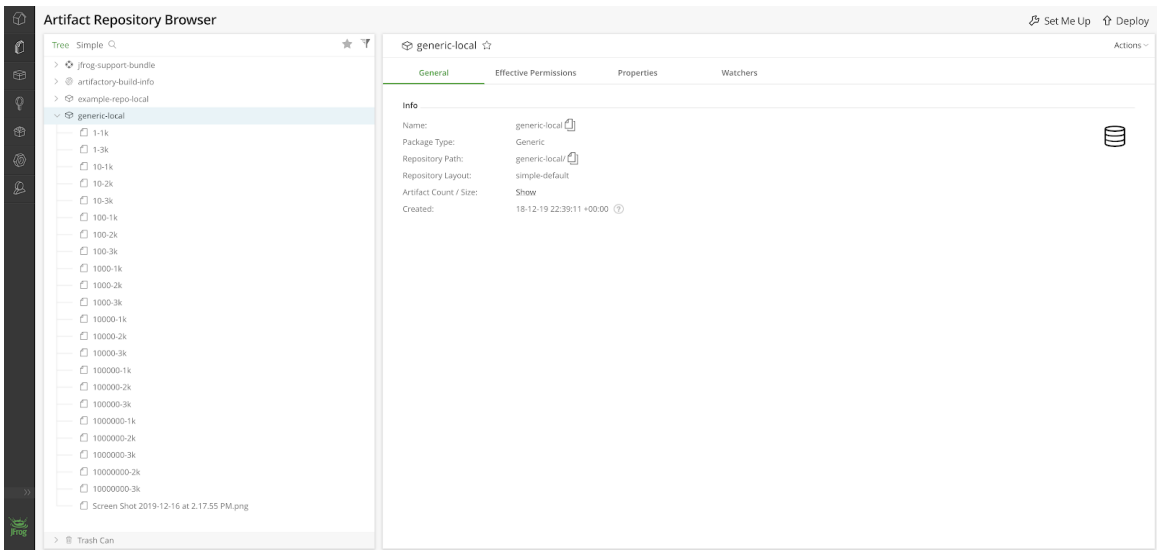


Figure 10. List of all the checksums in Artifactory under the generic-local.

Once the files were pushed or deployed to Artifactory, the Cloudberry browser connected to the S3 compliant repository bucket on FlashBlade listed all the checksums stored by artifactory.

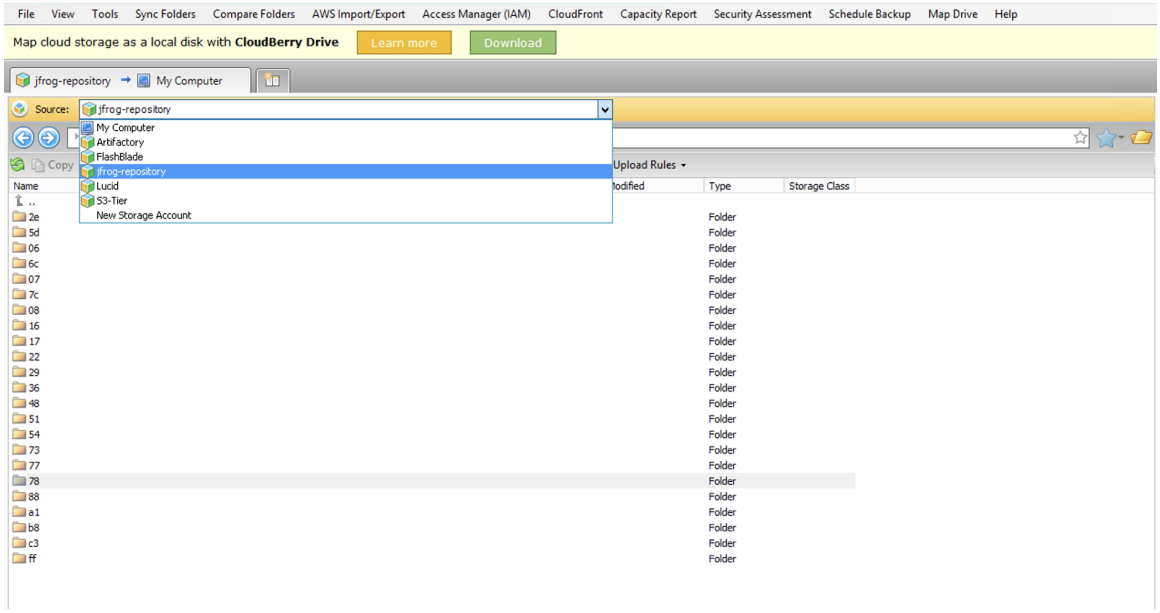


Figure 11. Cloudberry S3 / MSP360 browser lists all the checksums in the FlashBlade S3 compliant bucket.

The files were downloaded 8 times in 3 batches; a total of 24 downloads. The download speed for each of the files were measured for the three scenarios listed above. The average download speed of these files is summarized in Figure 12.

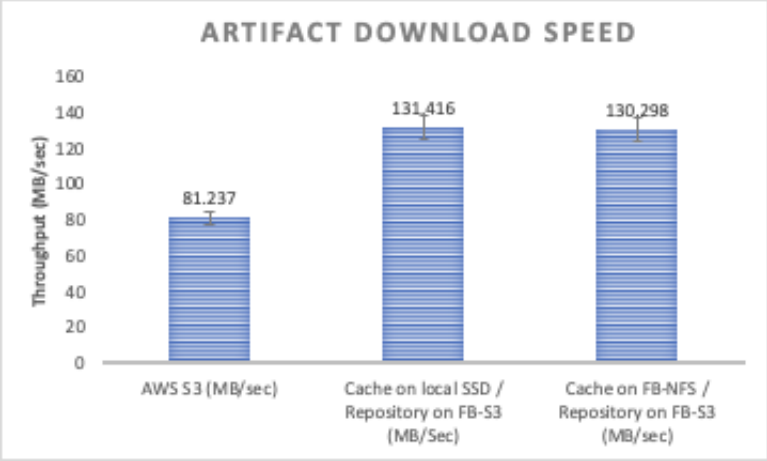


Figure 12. JFrog Artifactory download speed comparison.

Observations

The download speed from S3-compliant bucket repository on FlashBlade performed 62% faster than AWS S3. The download speed for having the Artifactory cache on the local server and the FlashBlade was almost the same, but having the cache on FlashBlade has advantages:

- An observed data reduction of 2:1 on FlashBlade allows optimization of the cache space enabling more data for faster download compared to S3 backend storage. Artifactory does not support any form of data compression for the cache. The data compression depends on the type of binary stored in Artifactory.
- Scaling the cache size on the FlashBlade is simple and easy with better data management capability compared to adding more servers for capacity scaling.

Recommendations

Apart from all the advantages that we have validated for running Artifactory Enterprise on FlashBlade so far, we also recommended optimizing the Nginx pod or Virtual Machine (VM) for better user request handling at scale.

- Depending on the development team and environment, configure the Nginx on a bare metal machine instead of VM. The nginx load balancer for Artifactory requires more resources for scalable performance. The nginx pod should be labeled and have a dedicated bare metal node in the Kubernetes cluster for better caching of user requests.
- Configure the nginx node or pod in Kubernetes cluster in the same network as the Artifactory nodes.
- Increase the nginx cache size for environments with a very high number of parallel user requests for downloading large binary contents. In a Kubernetes cluster, the nginx.conf file is modified to higher value depending on the Kubernetes available resource size to increase the proxy_buffer_size, proxy_buffers, and proxy_busy_buffers_size parameters using the following command:

```
kubectl edit configmap -n artifactory-ha artifactory-ha-nginx-conf -o yaml
```

In case nginx for Artifactory is configured on bare metal nodes, we recommend you use the local storage to increase the cache size for better upload and download of the artifacts and binaries. Configure nginx.conf file as shown below.

```
proxy_cache_path /mnt/nginxdisk1/cache levels=1:2 keys_zone=disk1:20m
max_size=750g inactive=180m  use_temp_path=off;
proxy_cache_path /mnt/nginxdisk2/cache levels=1:2 keys_zone=disk2:20m
max_size=750g inactive=180m  use_temp_path=off;
```

We recommended that you increase the size of the read cache using the binarystore.xml file for optimum performance depending on the size and the rate of upload and download of the artifacts and the binaries. The following example increases cache-fs to 10GB. You can increase the read cache to more than 10GB on FlashBlade for better cache capacity size scalability.

> ***Note:*** *Every time you resize the read cache with a new value in binarystore.xml, the Artifactory pods need to scale down to '0' and then scale back the pods to the default original value ('1' for artifactory-ha-primary-0 container and '2' for artifactory-ha-member-0 pods in the Kubernetes cluster). You need to restart Artifactory if the setup is on bare metal or VM nodes.*

Conclusion

Modern software development is getting bigger with a high volume of large binaries and various artifact versions that are shared across many data pipelines. FlashBlade provides the ideal data platform for integrated workloads that use Artifactory as the binary package manager during CI/CD and other extended workflows. JFrog Artifactory-validated architecture on Kubernetes, PSO and FlashBlade enhances the end user experience through:

- Scalable performance with improved cost efficiency
- Data access and continuity with Zero Storage Touch with REST APIs and PSO
- Simple data protection at the array level for better manageability

Pure Storage, Inc.
650 Castro Street, #400
Mountain View, CA 94041