PURESTORAGE®
Uncomplicate Data Storage, Forever

PERFORCE

TECHNICAL WHITE PAPER

Accelerate Perforce Database Recovery with Pure Storage

Optimize the Perforce Helix version-control system with Pure Storage® FlashArray™ and FlashBlade//S®.

Contents

Introduction 3

How to Use This Paper 4

Perforce Helix Server Checkpoints 4

Offline Checkpoints 5

Server Deployment Package 5

Scripts Used for Testing 6

Backup and Restore of Perforce Database on Pure Storage 6

- Snap-to-NFS Backup Process 6
- Prerequisites for Snap-to-NFS 7
- Backup Process Using Snap-to-NFS 8
- Observations 8

Disaster Recovery 9

- Snap-to-NFS Restore Process 9
- Observations 10

High Availability (HA) Failover 10

Conclusion 10

Appendix A 11

About the Authors 12

Introduction

The database, journal, and depot are essential components of the Perforce system. The Perforce database contains metadata about the versioned files, such as file revisions, changelists, users, and groups, whereas the depot stores the actual file contents. Backing up the Perforce database and the depot are critical in protecting an organization's digital assets and reducing interruption to business operations. Additionally, it is important to test the backup and restore process regularly to ensure that the data can be recovered in case of a disaster.

Pure Storage® has validated a hybrid architecture to optimize the Perforce Helix version-control system on FlashArray™ and FlashBlade//S® (Figure 1) to manage all digital assets generated from different EDA workflows and other industry verticals.

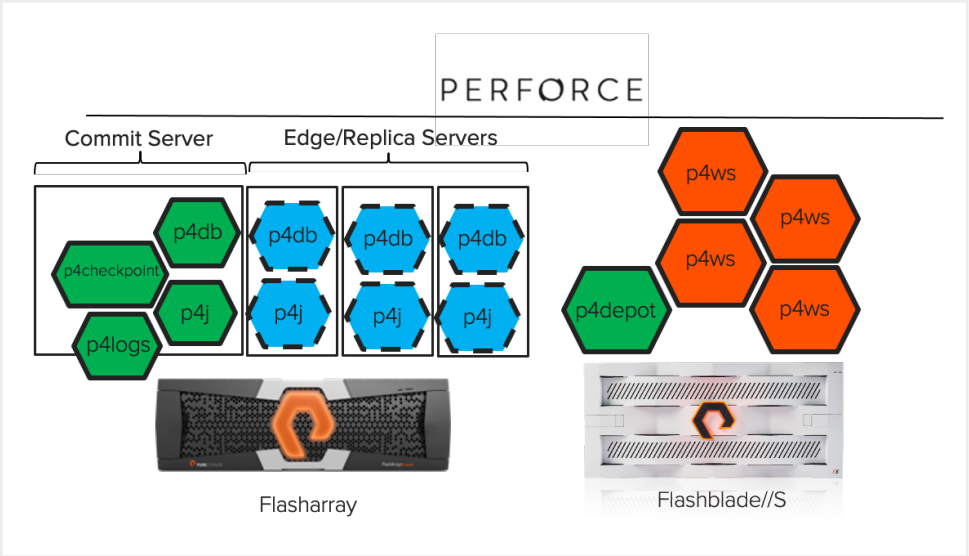

FIGURE 1 Hybrid architecture for Helix Core server on FlashArray and FlashBlade//S

The latency-sensitive Helix database is configured on FlashArray over iSCSI/FCP along with the replicas, journals, and checkpoints. Using three replicas (edge servers) with the Helix server provides parallelism and load-balances the read workload when users start to submit code changes and perform other metadata operations at scale.

Employing this validated architecture of Perforce on Pure Storage FlashArray and FlashBlade® arrays brings several benefits, including faster backup and recovery for Perforce databases.

Data protection and recovery depend on two objectives:

- **Recovery point objective (RPO)**: This is the amount of data loss. RPO determines the frequency of taking the Perforce Checkpoints on FlashArray and the snapshots for the depot filesystems on FlashBlade.
- **Recovery time objective (RTO)**: This is the time required to recover from failure. Longer recovery time leads to disruption to business operations.

Re-seeding the Perforce server after failures can be time consuming and disruptive. You can use checkpoints to recreate the Perforce database. (This doesn't apply to version files that reside inside the Depot; these should be backed up using standard mechanisms.) Compared to traditional Perforce checkpoints, taking array-level application snapshots on FlashArray and FlashBlade can significantly speed up the process of backing up and restoring the Perforce database, to minimize downtime and ensure high levels of developer productivity.

How to Use This Paper

This paper aims to demonstrate how the integration of FlashArray and FlashBlade with the Perforce system (see Figure 1) benefits faster backup and recovery for the Perforce databases. Using the FlashArray snapshots, you can accelerate the Perforce database backup process by 10x and the recovery process by 38x, thus providing excellent protection against failures and corruption.

This paper is intended for developers, project managers, DevOps engineers, and IT administrators involved in software development, digital asset management, and software control management (SCM) in the EDA/semiconductor industry. It may also be relevant to professionals interested in managing agile software development processes.

Perforce Helix Server Checkpoints

Perforce Helix Server maintains a metadata database consisting of many db.* (binary) files in the $P4ROOT directory.

To ensure effective disaster recovery, it is critical that you regularly back up the Perforce database by creating checkpoints. A checkpoint is a binary file that contains a snapshot of the current state of the Perforce database at a specific point in time. The checkpoint file can be easily backed up and restored to another server installation, which could be a replica. Additionally, it can be used for backup and restoration across different operating systems.

All updates to the metadata are logged in a journal file (in text format with markers for every atomic transaction). When creating a checkpoint, the server rotates the journal, where the active journal is renamed to a backup file with a sequential number, and a new journal is created. This rotation usually completes within a couple of seconds, and it is a straightforward operation that does not involve copying a large amount of data, even if the journal is several gigabytes in size.

To create a checkpoint, it is essential to have a consistent copy of all the db.* files. The key word is consistent. If you run the checkpoint command when the server is running, you risk an inconsistent copy.

For example, If files db.A, db.B, and db.C are updated for a single atomic transaction, it is important not to copy them when the transaction has been written to db.A but not yet to db.C.

Thus, creating a checkpoint involves the following basic steps:

1. Stopping the Helix Server
2. Copying the db.* files (which can be hundreds of GB or more)
3. Restarting the Helix Server

The obvious drawback to this method is that stopping and starting the server causes downtime for users.

There are two ways to ease the process of taking checkpoints to avoid server downtime:

- Offline checkpoints without impacting the production database
- Pure Storage array-level snapshots for Perforce Helix database

Offline Checkpoints

Offline checkpoints minimize the risk of potential outages resulting from database locking when creating a checkpoint.

The procedure for creating offline checkpoint is as follows:

1. Create an initial consistent copy (offline_db) of the production database (root containing db.*). This is done from a checkpoint or immediately following a journal rotation.
2. Rotate the live journal, which takes a few seconds.
3. Replay rotated journals from production into the offline database. They are numbered sequentially.
4. Checkpoint the offline database. This might still take hours, but it does not lock the production database during the process.
5. Optional: Remove the offline db.* files and recreate them from the newly created checkpoint. This defragments the db.* files and tests the checkpoint.

The disadvantage of an offline checkpoint is that it requires double the storage space of the "db.*" files, which are typically stored on faster (and more expensive) SSDs or similar storage devices.

Server Deployment Package

The Server Deployment Package (SDP) is a set of best practices implemented through scripts.

Characteristics of SDP include:

- Standard (but configurable) layouts
- Offline checkpoints
- Regular admin jobs (crontab) for log rotation
- Standard triggers to be implemented
- Multiple instances on the same server if required
- Easy ways to implement topologies with multiple replicas, edge servers, and backups

The SDP has evolved over a number of years. It can be implemented on different storage technologies, while still providing a resilient Helix core architecture that includes disaster recovery and high availability measures.

Combining FlashArray with the SDP to take advantage of features such as snapshots and volume can result in even faster performance. However, this paper does not cover integrating Pure Storage REST APIs with the SDP.

Scripts Used for Testing

Custom Python scripts were developed using the Pure Storage Python SDK. The Python scripts also used Perforce SDP scripts to rotate journals and to apply live journals. The custom scripts automated the backup and restore process.

The Python scripts interacted with FlashArray to perform the following operations:

- Create protection group snapshots
- Replicate protection group snapshots to the offload target
- List snapshots available locally and on the offload target
- Restore the snapshot from the offload target to FlashArray

See Appendix A for a sample execution of the scripts.

Backup and Restore of Perforce Database on Pure Storage

With Pure Storage FlashArray and FlashBlade integrated into the Perforce environment (see Figure 1), you can achieve quick and non-disruptive backup and recovery.

Snap-to-NFS Backup Process

This section elaborates the automated process of creating storage snapshot to eliminate any disruption to the users. In addition, this section outlines how to rapidly reseed the Perforce database from corruption by using Pure Storage snapshot and RESTful APIs.

The snapshots are offloaded to an NFS target (in this case, FlashBlade) for long-term retention. The Snap-to-NFS utility makes snapshots portable. The portable snapshot consists of metadata and the data required for restoring a volume. You can replicate the portable data blocks and metadata to any NFS target. Snap-to-NFS is installed and set up by Pure Storage Support.

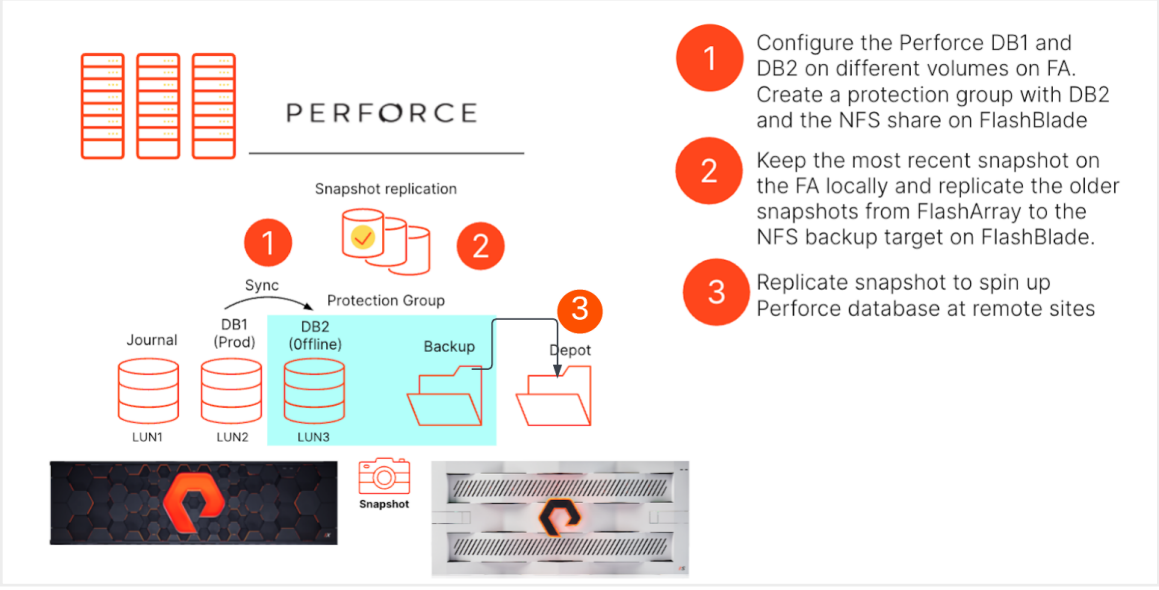

FIGURE 2 The Perforce database backup process

Figure 2 shows the recommended Perforce configuration to use the FlashArray-based snapshot for backup and restore. The Perforce instance should be configured with DB1 and DB2 on different FlashArray volumes. Create a protection group on FlashArray with DB2 volume. This protection group will also be configured with an NFS offload target to store the snapshots for longer retention. We can retain local snapshots for a shorter period and offload the snapshots for longer retention.

This testing was done with bare metal servers and iSCSI volumes from FlashArray. For virtual machines, use iSCSI volumes or vVols from FlashArray for Perforce DB1 and DB2 filesystems.

Prerequisites for Snap-to-NFS

The following prerequisites are required for setting up Snap-to-NFS:

- On the FlashArray (X70 in this case), ports eth2 and eth3 are the default replication ports. A replibond is created on the replication ports to offload the snapshots. It is recommended to have the replibond and the backup target ("nfsoffload" on the FlashBlade) in the same network.

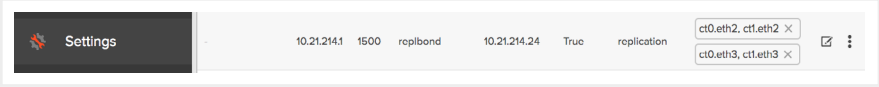

FIGURE 3 Replibond configured on FlashArray

- A protection group (PG-p4db2) is created with the source volume (a_29_p4db2_new) on FlashArray and the target filesystem (nfssnap) on FlashBlade.

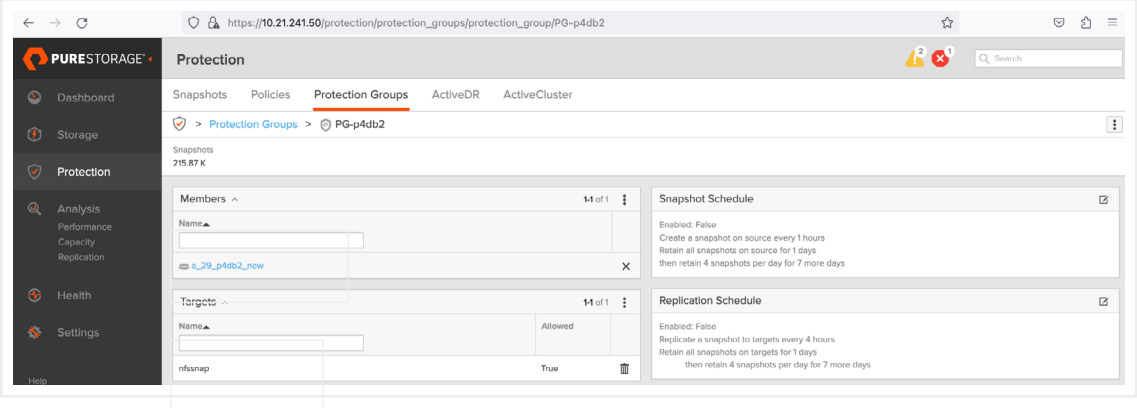

FIGURE 4 Protection group setup for the source and target volumes

Backup Process Using Snap-to-NFS

The following section outlines the operation sequence for the backup and recovery process for the checkpoints and re-seeding the Perforce database. The entire process is automated using RESTFul APIs.

Perforce database backup:

1. Use the Perforce SDP script to sync DB1 to DB2 and rotate the journal.
2. After the SDP script is complete, take a snapshot at the protection group level. Local snapshots can be retained for lesser duration.
3. A new snapshot for the DB2 volume is replicated to "nfsoffline" file system on FlashBlade using Snap-to-NFS. Snapshots on FlashBlade can be retained for longer duration.

NOTE: *With the application consistent snapshot at the array level, Perforce checkpoint might not be entirely required. But Perforce recommends that you have a checkpoint as a second-level protection.*

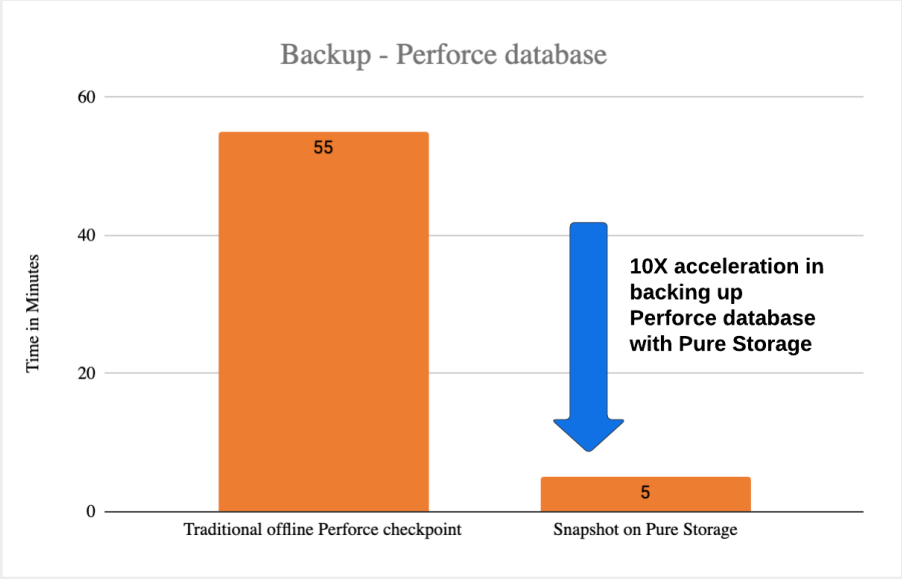

FIGURE 5 Perforce database backup time comparison

Observations

The validation was done on the production Perforce database volume, p4d. The size of the p4d volume was 500GB and the database size was 400 GB.

1. The greater the size of the journal, the longer it takes for the SDP script to rotate the journal. The backup completion time depends on the delta between DB1 and DB2.
2. An offline Perforce checkpoint of the Perforce database took 55 minutes to back up, while snapshots on FlashArray (including journal rotation) took 5 minutes (see Figure 5). When compared to the conventional Perforce offline checkpoint, there is a 10x acceleration in the backup time with the FlashArray snapshot method.
3. Replicating the DB2 volume snapshot to the backup target volume (nfsoffload) completed in under 3 minutes. Replication time depends on the size of the Perforce database.

Disaster Recovery

When performing disaster recovery for Perforce Helix, the checkpoints might be located on a different server or even in a separate data center. Ideally, you would use a standby replica of the Perforce Helix core in a non-mandatory mode. However, if the Perforce Helix core storage components are shared on FlashArray and FlashBlade, it's important to regularly snapshot the metadata and logs/journal on the FlashArray and copy them to the FlashBlade using Snap-to-NFS. This process eliminates the need for additional rsync time spent in moving the Perforce checkpoints in the event of a disaster. In such a scenario, you can easily restore the snapshot to any location, including to the original site.

Snap-to-NFS Restore Process

To recover the Perforce database using the traditional process, you need to rely on the most recent checkpoint and journal logs, which can be a time-consuming process. However, if you have a snapshot of the database using the Snap-to-NFS utility on the local FlashArray, you can quickly and easily restore the Perforce database files.

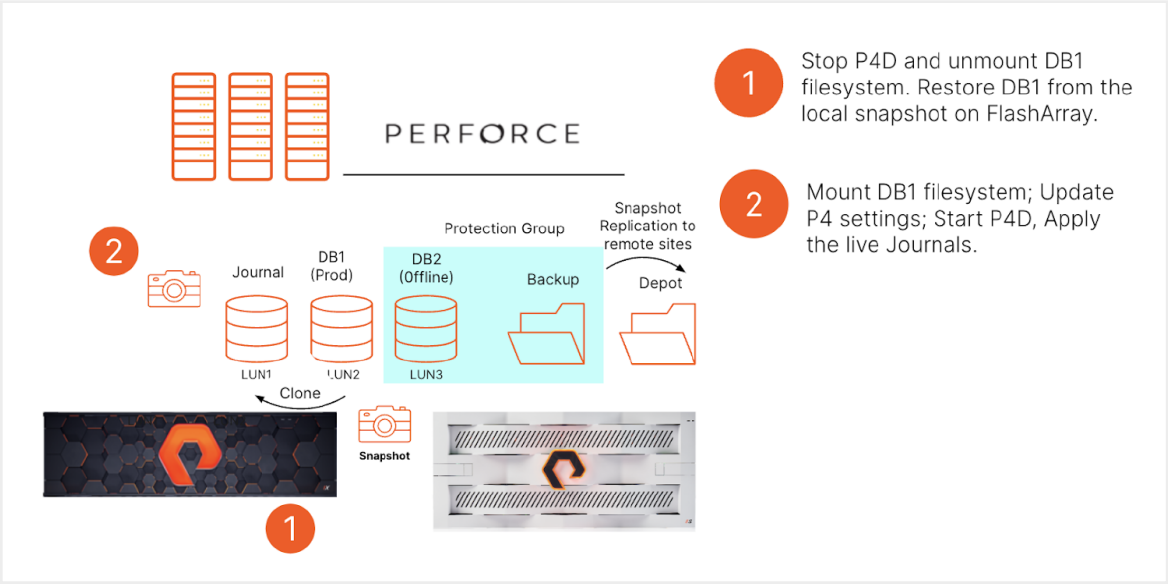

FIGURE 6 Perforce database recovery process

- If the recovery must be made from an older snapshot that is only available on nfsoffload (offload target), the snapshot is recovered from the backup target and transferred to the FlashArray using snap copy option.
- The recovered snapshot is used to overwrite one of the target volumes (a_29_p4db2_new in this example) on the FlashArray.

NOTE: *You can use the snapshot to recover the Perforce database files by overwriting the original p4db volume or any other volume that will host the new Perforce database.*

- Overwriting of the db.* files and restarting the Perforce Helix DB was completed in under 5 seconds.
- In this case, the snapshot of the DB2 volume was taken. After overwriting the DB1 volume using the snapshot, the DB1 links were updated accordingly.
- After overwriting the DB1 volume and starting the p4d service, the live journal was applied to bring the DB to the most recent state.

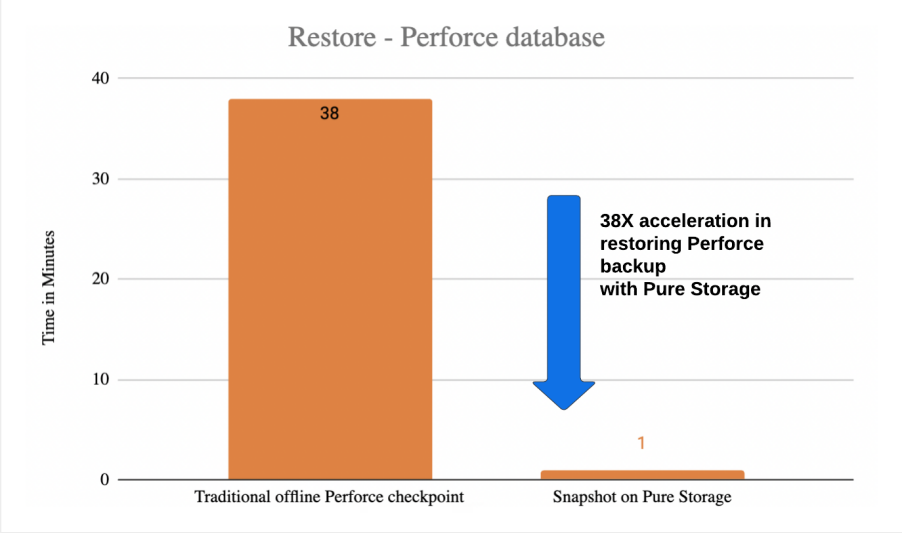

FIGURE 7 Perforce database recovery time comparison

Observations

- Re-hydrating the snapshot on a target volume allowed Helix Core to rebuild the database using files from the time the snapshot was last taken.
- Rebuilding the 4000 GB database using the Perforce checkpoint file took 38 minutes, while snapshot restore was able to reseed the Perforce database in under 1 minute (see Figure 7). When compared to the traditional Perforce checkpoint restore, there is a 38x acceleration in the Perforce restore process with the FlashArray snapshot.

High Availability (HA) Failover

For a failover situation, we assume that the same FlashArray is mounted on two different servers and contains metadata and journal/log volumes. If the main server fails or if there is a need to upgrade the operating system, we fail over to the backup.

For HA failover, deploy the Helix Server on both machines using Server Deployment Package as discussed above. It is possible to use a Perforce Mandatory Standby Replica and implement standard "p4 failover" actions.

Alternatively, since the data is already available on FlashArray, you can snapshot it regularly and replicate it quickly to the designated backup target using Snap-to-NFS. As Pure Storage snapshots contain both metadata and data, you can rapidly provision new replica or edge servers to achieve high availability and load balancing traffic.

Conclusion

Ensuring rapid data recovery to reseed the Perforce database from any disaster is critical for business continuity. In case of data corruption or loss, the FlashArray snapshot provides application-consistent data to reseed the Perforce database and the depot at a much faster rate than traditional methods.

Snap-to-NFS provides a unique approach to protect and recover snapshots from an offload NFS target, such as FlashBlade, facilitating recovery from failures. As noted in the observations, an offline Perforce checkpoint took 55 minutes to back up, while the Pure snapshot to a backup target was done in five minutes. Similarly, Snap-to-NFS was able to reseed the Perforce database in under one minute, while the conventional Perforce checkpoint recovery took 38 minutes.

Appendix A

The following script serves as an example of the scripts being executed.

```
[perforce@sn1-r6515-g04-29 ~]$ python3 fa.py createsnap
Connected to FlashArray sn1-x70-g06-23

Rotate journal
Executing source /p4/common/bin/p4_vars 5;/p4/common/bin/rotate_journal.sh 5
sleep for 5 seconds.
Take protection group snapshot
PG name: PG-p4db2

Creating protection group snapshot and replicating to offload target

Available snapshot for Volume :  a_29_p4db2_new on offload target nfssnap
sn1-x70-g06-23:PG-p4db2.16.a_29_p4db2_new
sn1-x70-g06-23:PG-p4db2.17.a_29_p4db2_new

---------------------------
Available local snapshot for Volume :  a_29_p4db2_new
PG-p4db2.16.a_29_p4db2_new
PG-p4db2.17.a_29_p4db2_new
[perforce@sn1-r6515-g04-29 ~]$

[perforce@sn1-r6515-g04-29 ~]$ python3 fa.py restorevol
Connected to FlashArray sn1-x70-g06-23

Stopping p4d. Proceed Y/N ?Y
Unmounting  p4db1 mount. Proceed Y/N ?Y
Verify  p4db1 mount
Restoring db1 volume from latest db2 snapshot on FlashArray.
Proceed Y/N ?Y
Copying PG-p4db2.17.a_29_p4db2_new to a_29_p4db1_new, with overwrite true

Overwriting the volume from latest local snapshot
Restoring from latest snapshot :  PG-p4db2.17.a_29_p4db2_new
Rescan Iscsi  and multipath
Mounting db1 filesystem
Updating p4db settings
Starting p4d
```

```
[perforce@sn1-r6515-g04-29 ~]$ python3 fa.py applyjournal
Connected to FlashArray sn1-x70-g06-23
Applying the live journal...
Executing source /p4/common/bin/p4_vars 5;p4d -r /p4/5/root -jrf /p4/5/logs/journal
Recovering from /p4/5/logs/journal...
Perforce server info:
  journal replay
--- db.config
---   pages in+out+cached 3+0+1
---   locks read/write 0/1 rows get+pos+scan put+del 0+0+0 0+0
---   total lock wait+held read/write 0ms+0ms/0ms+2ms
--- db.configh
---   pages in+out+cached 3+0+1
---   locks read/write 0/1 rows get+pos+scan put+del 0+0+0 0+0
---   total lock wait+held read/write 0ms+0ms/0ms+2ms
--- db.counters
```

About the Authors

Bikash Roy Choudhury, a director for solutions at Pure Storage, is responsible for designing and architecting solutions for DevOps workflows relevant across industry verticals including high tech, financial services, gaming, social media, and web-based organizations. He has also worked on validating solutions with Rancher/Kubernetes, GitLab, Jenkins, JFrog Artifactory, IBM Cloud Private, and Perforce using RESTful APIs and integrating them with data platforms in private, hybrid, and public clouds. In his current role, Bikash drives integrations with strategic DevOps partners, including Rancher, Mesosphere, Perforce, GitLab, and JFrog.

Unnikrishnan R is a senior solutions architect at Pure Storage, with over 17 years of experience in managing IT infrastructure, enterprise storage, cloud, and in implementing infrastructure as code (IaC) using Terraform and Ansible. He specializes in Linux and Kubernetes and has a passion for building robust DevOps solutions. In his role at Pure Storage, Unnikrishnan R is responsible for defining and executing complex solutions related to infrastructure and DevOps. He is well-versed in data management and migration, and has a deep understanding of data security and compliance.

PS2435-01-en 05/23