Contents

pgBackRest Reliable PostgreSQL Backup & Restore

Introduction

pgBackRest aims to be a simple, reliable backup and restore solution that can seamlessly scale up to the largest databases and workloads by utilizing algorithms that are optimized for database-specific requirements.

pgBackRest v2.06 is the current stable release. Release notes are on the Releases page.

Documentation for v1 can be found here. No further releases are planned for v1 because v2 is backward-compatible with v1 options and repositories.

Features

Parallel Backup & Restore

Compression is usually the bottleneck during backup operations but, even with now ubiquitous multi-core servers, most database backup solutions are still single-process. pgBackRest solves the compression bottleneck with parallel processing.

Utilizing multiple cores for compression makes it possible to achieve 1TB/hr raw throughput even on a 1Gb/s link. More cores and a larger pipe lead to even higher throughput.

Local or Remote Operation

A custom protocol allows pgBackRest to backup, restore, and archive locally or remotely via SSH with minimal configuration. An interface to query PostgreSQL is also provided via the protocol layer so that remote access to PostgreSQL is never required, which enhances security.

Full, Incremental, & Differential Backups

Full, differential, and incremental backups are supported. pgBackRest is not susceptible to the time resolution issues of rsync, making differential and incremental backups completely safe.

Backup Rotation & Archive Expiration

Retention polices can be set for full and differential backups to create coverage for any timeframe. WAL archive can be maintained for all backups or strictly for the most recent backups. In the latter case WAL required to make older backups consistent will be maintained in the archive.

Backup Integrity

Checksums are calculated for every file in the backup and rechecked during a restore. After a backup finishes copying files, it waits until every WAL segment required to make the backup consistent reaches the repository.

Backups in the repository are stored in the same format as a standard PostgreSQL cluster (including tablespaces). If compression is disabled and hard links are enabled it is possible to snapshot a backup in the repository and bring up a PostgreSQL cluster directly on the snapshot. This is advantageous for terabyte-scale databases that are time consuming to restore in the traditional way.

All operations utilize file and directory level fsync to ensure durability.

Page Checksums

PostgreSQL has supported page-level checksums since 9.3. If page checksums are enabled pgBackRest will validate the checksums for every file that is copied during a backup. All page checksums are validated during a full backup and checksums in files that have changed are validated during differential and incremental backups.

Validation failures do not stop the backup process, but warnings with details of exactly which pages have failed validation are output to the console and file log.

This feature allows page-level corruption to be detected early, before backups that contain valid copies of the data have expired.

Backup Resume

An aborted backup can be resumed from the point where it was stopped. Files that were already copied are compared with the checksums in the manifest to ensure integrity. Since this operation can take place entirely on the backup server, it reduces load on the database server and saves time since checksum calculation is faster than compressing and retransmitting data.

Streaming Compression & Checksums

Compression and checksum calculations are performed in stream while files are being copied to the repository, whether the repository is located locally or remotely.

If the repository is on a backup server, compression is performed on the database server and files are transmitted in a compressed format and simply stored on the backup server. When compression is disabled a lower level of compression is utilized to make efficient use of available bandwidth while keeping CPU cost to a minimum.

Delta Restore

The manifest contains checksums for every file in the backup so that during a restore it is possible to use these checksums to speed processing enormously. On a delta restore any files not present in the backup are first removed and then checksums are taken for the remaining files. Files that match the backup are left in place and the rest of the files are restored as usual. Parallel processing can lead to a dramatic reduction in restore times.

Parallel, Asynchronous WAL Push & Get

Dedicated commands are included for pushing WAL to the archive and getting WAL from the archive. Both commands support parallelism to accelerate processing and run asynchronously to provide the fastest possible response time to PostgreSQL.

WAL push automatically detects WAL segments that are pushed multiple times and de-duplicates when the segment is identical, otherwise an error is raised. Asynchronous WAL push allows transfer to be offloaded to another process which compresses WAL segments in parallel for maximum throughput. This can be a critical feature for databases with extremely high write volume.

Asynchronous WAL get maintains a local queue of WAL segments that are decompressed and ready for replay. This reduces the time needed to provide WAL to PostgreSQL which maximizes replay speed. Higher-latency connections and storage (such as S3) benefit the most.

The push and get commands both ensure that the database and repository match by comparing PostgreSQL versions and system identifiers. This virtually eliminates the possibility of misconfiguring the WAL archive location.

Tablespace & Link Support

Tablespaces are fully supported and on restore tablespaces can be remapped to any location. It is also possible to remap all tablespaces to one location with a single command which is useful for development restores.

File and directory links are supported for any file or directory in the PostgreSQL cluster. When restoring it is possible to restore all links to their original locations, remap some or all links, or restore some or all links as normal files or directories within the cluster directory.

Amazon S3 Support

pgBackRest repositories can be stored on Amazon S3 to allow for virtually unlimited capacity and retention.

Encryption

pgBackRest can encrypt the repository to secure backups wherever they are stored.

Compatibility with PostgreSQL >= 8.3

pgBackRest includes support for versions down to 8.3, since older versions of PostgreSQL are still regularly utilized.

Getting Started

pgBackRest strives to be easy to configure and operate:

- User guide for Debian & Ubuntu / PostgreSQL 9.4.
- Command reference for command-line operations.
- Configuration reference for creating pgBackRest configurations.

Contributions

Contributions to pgBackRest are always welcome!

Code fixes or new features can be submitted via pull requests. Ideas for new features and improvements to existing functionality or documentation can be submitted as issues. You may want to check the Feature Backlog to see if your suggestion has already been submitted.

Bug reports should be submitted as issues. Please provide as much information as possible to aid in determining the cause of the problem.

You will always receive credit in the release notes for your contributions.

Support

pgBackRest is completely free and open source under the MIT license. You may use it for personal or commercial purposes without any restrictions whatsoever. Bug reports are taken very seriously and will be addressed as quickly as possible.

Creating a robust disaster recovery policy with proper replication and backup strategies can be a very complex and daunting task. You may find that you need help during the architecture phase and ongoing support to ensure that your enterprise continues running smoothly.

Crunchy Data provides packaged versions of pgBackRest for major operating systems and expert full life-cycle commercial support for pgBackRest and all things PostgreSQL. Crunchy Data is committed to providing open source solutions with no vendor lock-in, ensuring that cross-compatibility with the community version of pgBackRest is always strictly maintained.

Please visit Crunchy Data for more information.

Recognition

Primary recognition goes to Stephen Frost for all his valuable advice and criticism during the development of pgBackRest.

Crunchy Data has contributed significant time and resources to pgBackRest and continues to actively support development. Resonate also contributed to the development of pgBackRest and allowed early (but well tested) versions to be installed as their primary PostgreSQL backup solution.

Armchair graphic by Sandor Szabo.