White Paper

YugabyteDB Backup And Restore

A Bird's Eye View of Data Protection Strategies and Capabilities

yugabyteDB

Table of Contents

Introduction

App developers, IT leaders and everyone in between are involved in transforming companies into data-driven digital businesses that can thrive in the face of growing uncertainty. Data plays a critical role in these businesses and the new services they offer. Protecting that data and eliminating any disruptions that might impact customer experiences is critical.

An intelligent, distributed data layer is becoming the first line of defense against challenges like user errors, cloud outages, and even natural disasters. While implementing a modern data layer like YugabyteDB is a key first step, it's still important to have a solid backup and restore strategy in place for those rare instances they are required.

Why Organizations Need A Backup And DR Strategy

In our data-centric world, data is an essential part of running any business, from the largest enterprises to a local bicycle shop. The loss of that data can result in sleepless nights, frustrated customers, and the loss of revenue. For these reasons, most businesses rely on some form of data backup and recovery to avoid any data loss, or at least to minimize the negative impact in the event of some theft, damage or unexpected disaster.

Organizations implement backup and recovery strategies to protect their data against a wide range of issues that can put primary data at risk. Some of the main reasons include:

- User error or human-caused events like a malicious attack
- Major failures including hardware or software failures, data corruption, cloud outage or other natural disaster impacts
- Severe disasters, which are rare, but involve cluster failures or simultaneous failures across systems
- Compliance and governance often require certain strategies in place to protect the end consumer or user

For traditional, monolithic databases, a complete backup and DR strategy was essential to protect against any of the above scenarios. Whenever they occurred, complex processes were required and teams had to manually address numerous issues, including reconciling data changes between various systems.

For modern, distributed SQL databases like YugabyteDB some of these common scenarios are addressed natively by the distributed architecture—helping organizations avoid the manual, expensive recovery process. The rest of this paper explores the advantages of distributed SQL databases for high resiliency, what role backup and recovery strategies still play, and what are the various backup options available in YugabyteDB.

Seamless Protection With Distributed SQL Databases

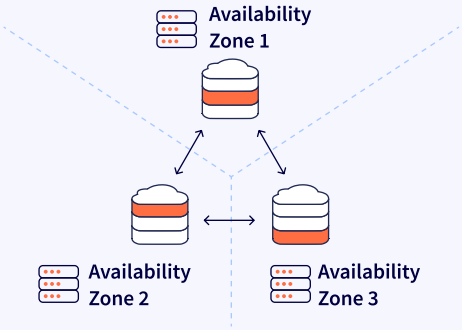
Unlike traditional single-instance databases, YugabyteDB is a distributed SQL database whose core architecture was built for fault tolerance and high availability. By maintaining three or more copies of the data across multiple nodes, data regions or clouds, YugabyteDB makes sure no data losses occur in the event a single node or even an entire data region becomes unavailable.

Thanks to the distributed nature of YugabyteDB, organizations no longer need to rely on classic backup and recovery processes for many common failures. As with cloud native applications and microservices, YugabyteDB automatically replicates data synchronously within a cluster so that hardware failures, network issues, cloud outages, and more, are seamlessly handled.

Beyond just replicating data, YugabyteDB includes native automatic failover and repair, meaning that the system will self-heal in the event of any of these common failures. Along with self-healing, the system will automatically and intelligently re-balance data across the available nodes. When new nodes are added or recovered, then the system rebalances again seamlessly.

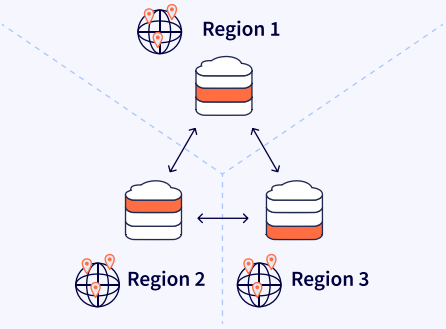
Resilient And Strong Consistent Across Failure Domains

1. Single Region, Multi-Zone



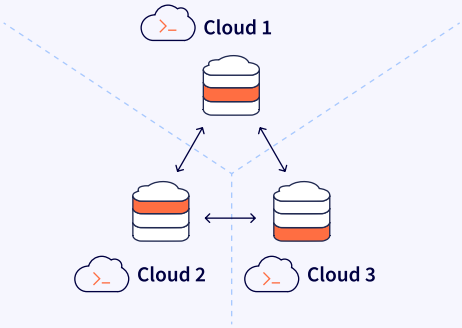
Consistent across zones no WAN latency but no Region-level failover/repair

2. Single Cloud, Multi-Region



Consistent across regions with auto Region-level failover/repair

3. Multi-Cloud, Multi-Region



Consistent across cloudswith auto Cloud-level failover/repar

As a result of this modern and distributed architecture, backups with YugabyteDB play a smaller role than in the past; however, there are still situations where a backup and recovery process is needed and they should always be part of any plan. A backup and recovery strategy is still needed for these three key scenarios:

1. **Fix User Error:** Recover from a user or software error, such as an accidental table removal or other permanent operations that were not intended.
2. **Recover from Severe Disaster:** In the event of a severe disaster scenario, like a full cluster failure or a simultaneous outage of multiple data regions (probability of such scenarios is extremely low), then a proven backup and restore process is needed. Despite the low odds of facing these severe scenarios, it is still recommended to maintain a way to recover from them.
3. **Address Regulation Requirements:** Maintain a remote copy of the data as required by data protection regulations.

In general, the day-to-day protection of the distributed database in the event of a down node or region outage are seamlessly handled by the distributed nature of YugabyteDB. Multiple copies are stored based on the user defined levels of protection required, and the database automatically rebalances data as needed to ensure optimal performance and protection with the available resources.

YugabyteDB Backup Options

As stated above, the best practice is to always have a data backup and restore process defined and tested to cover those additional scenarios that, although less common than classic outages and hardware failures, can still happen. YugabyteDB comes with three features that allow users to set up a data backup strategy:

- SQL/CQL export
- Distributed snapshots
- Point-in-time-recovery

Method 1: SQL/CQL Export

Leveraging core capabilities derived from Postgres and Cassandra APIs, YugabyteDB allows exporting data to a SQL or CQL script. Use of this process is recommended only if you intend to restore the exported data on a database other than YugabyteDB or if having data in SQL/CQL format is a requirement for other external reasons (e.g., regulations).

For YSQL data, export is performed via **ysql_dump** and **ysql_dumpall** scripts. The former exports a single database, while the latter exports all databases along with global objects like users, roles, and permissions.

```
./postgres/bin/ysql_dump -d <bd-
        name> > <file>
```

For YCQL data, export is performed via **DESCRIBE** and **COPY TO** CQL commands.

```
./bin/ycqlsh -e "DESC KEYSPACE
  <keyspace_name>" > <file>
```

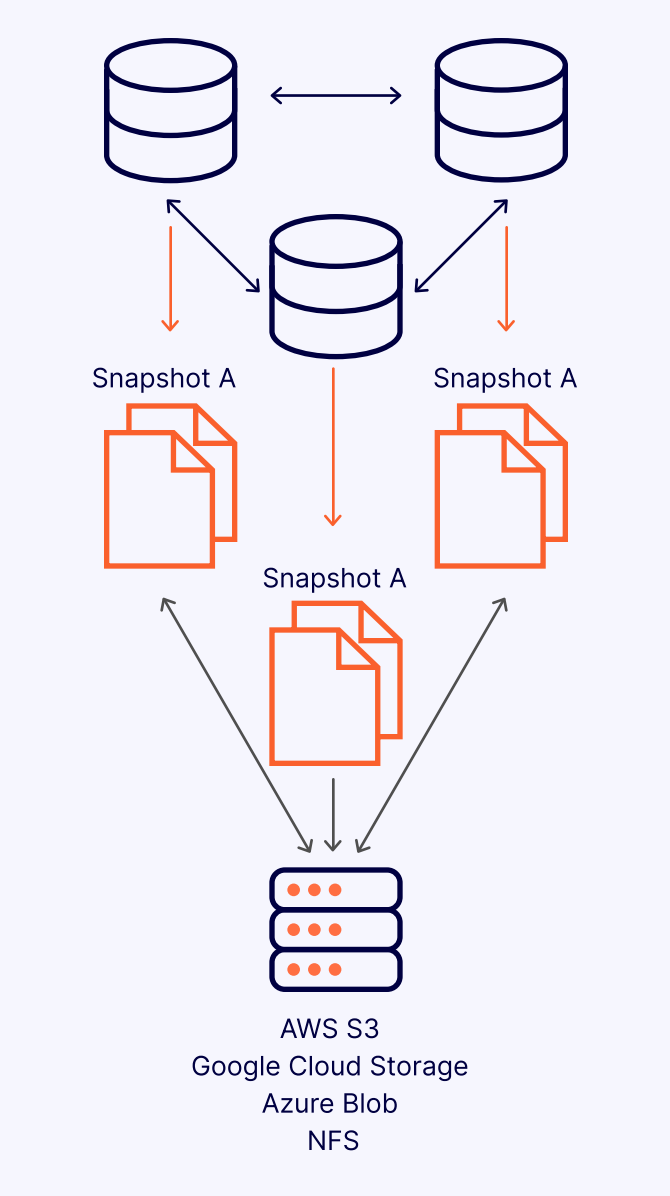
For most users, the distributed snapshots capability described in the next section is the recommended way to create backups as they are more efficient and robust.

Method 2: Distributed Snapshots

The most efficient and robust way to backup the data stored in YugabyteDB is to create a distributed snapshot. A distributed snapshot is a consistent cut of the data taken across all the nodes in the cluster that can be created either on demand or as a part of a pre-defined schedule. Distributed snapshots provide a higher level of efficiency versus a simple SQL/CQL Export because a snapshot is created independently on every node, which makes the process highly scalable.

A newly created snapshot is kept in-cluster and stored on the same storage volumes where the data itself resides. If required, you can export and copy any snapshot to an external storage.

Functionality is provided by the **yb-admin** script. A snapshot is created for a single database (YSQL) or a single keyspace (YCQL). With YCQL, you can also create a snapshot for a specific table.



In addition, YugabyteDB Anywhere provides an API and UI for backup automation. YugabyteDB Anywhere helps organizations to quickly and easily deploy a self-managed DBaaS in any environment (public, private and hybrid cloud) with built-in automation to deploy, scale, and monitor YugabyteDB clusters at scale. Supported storages for the backup automation are AWS S3, Google Cloud Storage, Azure Blob, and generic NFS.

In regards to incremental backups, YugabyteDB always creates an efficient full snapshot of all the data, eliminating the need and extra work required to provide incremental in-cluster snapshots. The underlying mechanism is efficient enough both in terms of speed and space consumption to not benefit in any meaningful way from an incremental approach to in-cluster snapshots. The Yugabyte team is continuing to investigate ways to further enhance distributed snapshots, such as providing incremental off-cluster backups in the future.

How Distributed Snapshots Work

Distributed snapshots in YugabyteDB are performed in-cluster, meaning that they are created locally by the Yugabyte Tablet Server (YB-TServer) service. The snapshots are stored in the same storage volumes where the actual data is also stored.

When a user requests a snapshot, the request is sent to every YB-TServer. Each YB-TServer then goes through all local tablets that correspond to the database being backed up and creates hard links to the relevant data files. The system also marks the current hybrid timestamp and assigns it to the snapshot.

The distributed snapshot creation completes quickly and efficiently as it requires minimal coordination, is executed locally on every node (and therefore is scalable), and does not imply any file copying.

In-Cluster Vs. Off-Cluster

The speed and scalability of distributed snapshots comes with a price of inflated requirements for the data storage. Since all snapshots are stored in-cluster, they utilize the primary cluster storage. To overcome the higher storage requirements, any snapshot can be moved to a cheaper external storage (such as S3). Moving the snapshots can be done either manually or programmatically by utilizing the API and UI in YugabyteDB Anywhere.

Method 3: Point-In-Time Recovery (PITR)

The third and final backup and restore strategy for YugabyteDB is point-in-time recovery (PITR). PITR in YugabyteDB enables recovery from a user or software error, while minimizing recovery point objective (RPO), recovery time objective (RTO), and overall impact on the cluster. It works by restoring to the latest known working state of a database, as opposed to a time of snapshot creation.

PITR is particularly useful in the following two scenarios:

- DDL errors, such as an accidental table removal
- DML errors, such as execution of an incorrect update statement against one of the tables

Given these common scenarios, you typically know when the data was corrupted and would want to restore to the closest possible uncorrupted state. With PITR, you can achieve that by providing a restore timestamp. You can specify the time with the precision of up to 1 microsecond, far more precision than is possible with the regular snapshots that are typically taken hourly or daily.

While PITR is primarily based on distributed snapshots, there are two additional requirements that need to be satisfied in order to restore to a specific point in time:

1. Snapshots need to be created as a part of a schedule.
2. Snapshots need to be kept in-cluster. You can still copy them to an external storage, but removing them from the in-cluster storage will prevent you from using PITR.

The scope of PITR is always a single database (YSQL) or a single keyspace (YCQL). Currently, PITR is not supported for a single table for either YSQL or YCQL. Functionality is provided by the **yb-admin** script.

How Point-In-Time Recovery Works

Let's explore more of the details of how PITR in YugabyteDB works. PITR is actually based on a combination of two key features: the flashback capability and periodic distributed snapshots.

Flashback is a feature that allows you to rewind the data back in time. At any moment, YugabyteDB stores not only the latest state of the data, but also the recent history of changes. With flashback, you can rollback to any point in time in the history retention period. The history is also preserved when a snapshot is taken, which means that by creating snapshots periodically, you effectively increase the flashback retention.

For example, if your overall retention target for PITR is 3 days, you can use the following configuration:

- History retention interval is 24 hours
- Snapshots are taken daily
- Each snapshot is kept for 3 days

By default, the history retention period is controlled by the **history retention interval flag**. The flag is applied cluster-wide to every YSQL database and YCQL keyspace. However, when PITR is enabled for a database or a keyspace, YugabyteDB adjusts the history retention for that database/ keyspace based on the interval between the snapshots. You are not required to manually set the cluster-wide flag in order to use PITR.

There are no technical limitations on the retention target. However, it's important to keep in mind that by increasing the number of snapshots stored, you also increase the amount of space required for the database. The actual overhead depends on the workload, so we recommend estimating it by running tests based on your applications.

Looking back at our example above, the configuration ensures that at any moment there is a continuous change history maintained for the last 3 days. When you trigger a restore, YugabyteDB will pick the closest snapshot to the timestamp you provide, and then use flashback within that snapshot.

Let's say the snapshots are taken daily at 11:00 PM, current time is 5:00 PM on April 14th, and you want to restore to 3:00 PM on April 12th. In this case, YugabyteDB:

1. Locates the snapshot taken on April 12th (which is the closest snapshot taken after the restore time), and restores that snapshot.
2. Flashes back 8 hours to restore to the state at 3:00 PM (as opposed to 11:00 PM, which is when the snapshot was taken). a

Conclusion

While this paper focused on the options to backup and restore YugabyteDB, it's important to first recognize that the fundamental benefits of a modern, distributed database architecture are that single node, region or even cloud failures are seamlessly handled without the need for expensive and complex recovery options typical to monolithic database solutions.

However, some scenarios still exist that require backup and restore. This means it's highly recommended that you are familiar with these strategies and have a defined process in place.

The most efficient way to set up a backup strategy for YugabyteDB is to create a distributed snapshot schedule based on your retention requirements. A snapshot created as part of the schedule can be used to restore to the moment of its creation, or – as long as it is kept in-cluster – to a point in time that represents the latest known working state of the database using PITR.

If you intend to restore the data on a database other than YugabyteDB, or need to store data in a human-readable format (SQL/CQL/CSV), use the SQL/CQL export functionality.

Get Started Today

For more information on YugabyteDB backup and restore options, reach out directly to the **Yugabyte team**, follow us on **LinkedIn** or join our **Community Slack**.

You can also download YugabyteDB for free **here**.

yugabyteDB

Get in Touch

www.yugabyte.com | contact@yugabyte.com