HVR High Availability

Contents

Introduction

The ability to replicate data between data stores has a number of dependencies. Source and target databases/data stores must be available and accessible, all of the infrastructure involved in the data replication must be available and allow connectivity. The software that makes data replication possible must work as designed. At any point in time, one or more of the components may fail.

This document describes how to configure HVR data replication for High Availability (HA). Fundamentally, the software is designed to resume replication at the point where replication was stopped. With an approach to HA data replication, downtime is either avoided or limited so that data keeps flowing between source(s) and target(s).

Backup, Disaster Recovery, High Availability

There are multiple strategies to get a setup that is no longer functioning back into a functioning state.

- A backup is a copy of your application data. If the data replication setup fails due to a component failure or corruption, it can be restored on the same or new equipment, and from there data replication can be recovered.
- Disaster Recovery (DR) is a completely separate setup that can take over in case of a major event - a disaster such as flooding or an earthquake - taking out many components at the time e.g. entire data centers, the electricity grid, or network connectivity for a large region.
- High Availability (HA) is a setup with no single point of failure. HA introduces redundancy into the setup to allow for components to step in if there is a failure.

What availability/recovery strategy or combination of strategies works best for your organization depends on a number of factors, including the complexity of the environment, the available budget to ensure availability, and the extent of replication downtime your organization can afford.

Recovery Time Objective

An important HA consideration is the impact of downtime on operations and with that the business cost to replication downtime. When replication is down, data is no longer flowing between source(s) and target(s). If the replication downtime is unrelated to the availability of source(s) and target(s) then latency will increase until replication is restored. However, data will still be available on the target, getting staler as time goes on, with current data available in the source. Facing high latency is similar to replication unavailability.

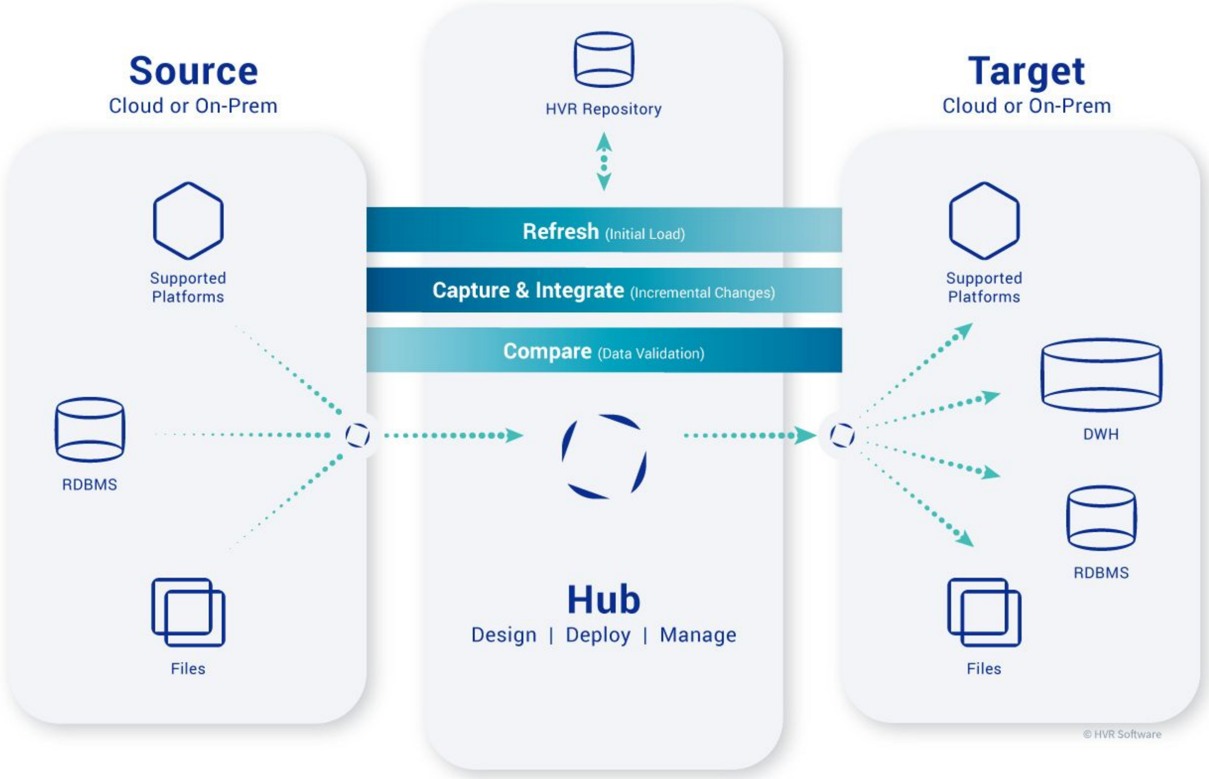
This paper discusses HA for data replication. The time period for which you can sustain the situation with data replication down determines your so-called Recovery Time Objective (RTO): how long can you afford for replication to be down. For some organizations, the RTO will be as low as minutes if not less, for example, if data replication is a core part of a broader application high availability strategy for a mission-critical system. Other organizations may have the flexibility to redirect workloads allowing them to afford a replication downtime of multiple minutes if not hours. Some organizations may run non-mission-critical workloads on their replicated data sets and some occasional data replication downtime can be coped with relatively easily.

Weigh your RTO against the cost and complexity of implementing HA as you architect your data replication configuration.

HVR Architecture

HVR's real-time replication solution features a distributed approach to log-based Change Data Capture (CD C) and continuous integration. The HVR software consists of a single installation that can act as a so-called hub controlling data replication, or as an agent performing work as instructed by the hub. The hub and agents communicate over TCP/IP, optionally using encrypted messages. However, the architecture is flexible. Any installation of the software can act as an agent or as a hub, and technically the use of agents in a setup is optional.

The image below shows the distributed architecture, with, in this case, a single source agent and a single target agent. Real-world deployments often use a separate agent per source and a separate agent per target.



The distributed architecture provides several benefits:

- Offloading/distributing some of the most resource-intensive processing to avoid bottleneck processing changes centrally.
- Optimizing network communication with data compressed by the agent before sent over the wire.
- Improved security with unified authorization and the ability to encrypt data on the wire.

Agents

Agents are typically installed close to the source database(s) if not on the database server(s), and close to the destination systems. Agents store very limited state information (configuration dependent). Based on the state information stored on the hub, data replication can always resume at the point of stoppage if replication is stopped or fails. A single hub may orchestrate data replication between hundreds of agents.

Hub

The HVR hub is an installation of HVR software that is used to configure data replication flows. Metadata is stored in a repository database that can be any one of the relational databases that are supported as a source or Teradata. The hub also runs a scheduler to start jobs and make sure jobs are restarted if they fail. During operations, the scheduler requires a connection to the repository database. The repository database can either be local to the host running the hub, or remote using a remote database connection.

The main functions of the hub are:

- Maintain the replication state.
- Restart/resume replication in case of a failure.
- Route compressed transaction files arriving from the source(s) to the correct target(s).
- Be the access point for operator maintenance and monitoring.

Environments with a dedicated hub server will see the bulk of data replication processing taking place on the agent servers, with very little data replication load on the hub.

High Availability for HVR Agents

Data replication is recoverable based on the state information stored on the hub. High availability for an agent can be achieved by having a duplicate installation for the agent available. HVR identifies an agent in the so-called Location Configuration through a host name or IP-address where the HVR remote listener must be running to establish a connection. The remote listener is a light-weight process, similar to a database listener, that forks new processes to perform work.

If the agent runs on the source or target database server then the agent should be available if the server is available, and the remote listener is running. Make sure to configure the remote listener to start upon system startup through systemd, (x)inetd or hvr_boot on Linux/Unix or a service on Windows. See Installing HVR on Unix or Linux for more details.

Consider using a floating virtual IP-address/host name to identify the agent, or use a load balancer in front of the agent for automatic failover.

Cloud providers have services available that can help implement high availability for agents.

State Information Stored by HVR Agent

An HVR agent performing CDC on the source may store state information about long-running open transactions to prevent a long recovery time when the capture process restarts. By default, every five minutes the capture process writes a so-called checkpoint to disk, storing the in-memory state of the transactions open longer than 5 minutes. Upon restart, the most recent complete checkpoint becomes the starting point of CDC, followed by re-reading (backed up) transaction logs from the point of the checkpoint forward, ensuring no data changes are lost.

Long-running transactions may occur on Oracle databases, especially when using packaged applications such as SAP or Oracle eBusiness Suite. Long-running transactions are less likely on other databases. The default location to store the checkpoint is at the agent, but setting option **CheckPointStorage** to HUB for action **Capture** will move the checkpoints to the hub so that in case of a restart, the checkpoint stored on the hub can be the starting point. Note that storing the checkpoint on the hub will take more time than storing the checkpoint locally where the agent runs.

The integration agent stores state information:

- in the target, in the so-called state tables if the target is a database.
- implicitly using a directory **hvr_state** if the target is a file system (except for S3 as a file system, when the **hvrmanifestagent** should be used to publish files).

With the agent state information stored in the actual target location, data replication can leverage the current state irrespective of whether the same or a different instance of an agent is used. here is no need to consider integrate state when considering high availability for the integration agents.

High Availability for HVR Hub

For replication to function, the hub's scheduler must be running, which requires a connection to the repository database to retrieve and maintain the job state (e.g. **RUNNING**, **SUSPENDED**, etc.). Data replication definitions stored in the same repository are not required at runtime. However, data replication can be re-instantiated from the current definitions.

Any state information about the data replication beyond just the job state is stored on the file system in a directory identified by the environment setting **HVR_CONFIG**. To resume replication where it left off prior to a failure, files in this directory must be accessible.

High availability setup for the HVR hub requires the **Scheduler** to run, which requires:

- Repository database to be accessible (note that the repository database can be remote from the hub server or local to it).
- Access to up-to-date and consistent data in the **HVR_CONFIG** directory.

A common way to implement high availability for the HVR hub is to use a cluster with shared storage (see I nstalling HVR in a Cluster for installation instructions). In a clustered setup, the cluster manager is in charge of the HVR scheduler as a cluster resource, making sure that within the cluster only one hub scheduler runs at any point in time. File system location **HVR_CONFIG** must be on the attached storage, shared between the nodes in the cluster or switched over during the failover of the cluster (e.g. in a Windows cluster). If the repository database is local to the hub like an Oracle RAC Database or a SQL Server AlwaysOn cluster, then the connection to the database can always be established to the database on the local node, or using the cluster identifier. For a remote database (network) connectivity to the database must be available, and the database must have its own high availability setup that allows remote connectivity in case of failure.

Cloud environments provide services to support an HVR high availability configuration for the **HVR_CONFIG** file system like Elastic File System (EFS) on Amazon Web Services (AWS), Azure Files on Microsoft Azure, and Google Cloud Filestore on Google Cloud Services (GCS). The cloud providers also provide database services with built-in high availability capabilities and redundancies in the connectivity to allow for failures without impacting availability.

Recovering HVR Replication

If HVR replication fails and there is no high availability setup, or for whatever reason, the high availability configuration was unable to prevent an outage (e.g. in a perfect storm of failures or a disaster scenario, when DR could have provided business continuity but high availability cannot), then you must recover replication. How can you do this?

Restore and Recover from Backup

Make sure to always have a current backup of your HVR definitions by using the **Export Catalogs** function. In the worst case, with data replication otherwise unavailable, you can restore the environment from the export of the data definitions:

- Create a new repository database (with empty tables).
- Import the export file.
- Configure **Resilient** processing on **Integrate** (temporarily) to instruct HVR to merge changes into the target.

For a target that writes an audit trail of changes (i.e. configured using **TimeKey**), you will need to identify the most recently applied transaction and use this information to re-initialize data replication to avoid data overlap, or you must have downstream data consumption cope with possible data overlap.

- Initialize the data replication channels, using **Capture Rewind** (if possible, i.e. if transaction file backups are still available and accessible) to avoid loss of data.
- If backup transaction log files (or archived logs) are no longer available, then use **HVR Refresh** to re-sync the table definitions between a source and a target. Depending on the target and data volumes, a row-by-row Refresh, also referred to as repair, may be appropriate, and/or use action **Re strict** to define a suitable filter condition to re-sync the data.

Consider the backup retention strategy for your transaction logs if your recovery strategy for data replication includes recovery from a backup to allow for replication to recover by going through the backups of the transaction log files rather than having to re-sync from the source directly.

Disaster Recovery

To recover data replication from a disaster recovery environment is not unlike the recovery from a backup, except that the recovery time is likely lower, because the environment is ready and available. The disaster recovery environment may even be able to connect to the primary repository database for up-to-date data replication definitions and current job state information. However, the disaster recovery environment would not have access to an up-to-date **HVR_CONFIG** location so the data replication state would have to be recreated.

With this, the steps to implement a disaster recovery environment are:

- Configure Resilient processing on **Integrate** (temporarily) to instruct HVR to merge changes into the target.

  For a target that writes an audit trail of changes (i.e. configured using **TimeKey**) you will need to identify the most recently applied transaction and use this information to re-initialize data replication to avoid data overlap, or you must have downstream data consumption cope with possible data overlap.

- **Initialize** the data replication channels, using **Capture Rewind** (if possible, i.e. if transaction file backups are still available and accessible) to avoid loss of data.
- If backup transaction log files (or archived logs) are no longer available, then use **HVR Refresh** to re-sync the table definitions between a source and a target.

Disaster Recovery - Using Heartbeat Table

To restore data replication from a backup or even in a disaster recovery environment is challenging because the data replication state, from **HVR_CONFIG** on the failed primary environment, is not available. With the loss of **HVR_CONFIG**, the following state information is lost:

- Capture status stored in the so-called **cap_state** file. Most importantly for recovery, this file includes the transaction log's position of the beginning of the oldest open transaction capture is tracking.
- Transaction files that have yet to be integrated into the target. Depending on the setup, it may be normal that there are hundreds of MBs waiting to be integrated into the target, and regularly copying these (consistently) to a disaster recovery location may be unrealistic.
- Table Enrollment information containing the mapping between table names and object IDs in the database. This information doesn't change frequently and it is often safe to recreate the Table Enrollment information based on current database object IDs. However, there is a potential for data loss if data object IDs did change, for example for a SQL Server source if primary indexes were rebuilt between the point in time where capture resumes and the current time.

The concept of the heartbeat table includes:

- A heartbeat table on every source database that HVR captures from. The table can be created in the application schema or in a separate schema that the HVR capture user has access to (for example, the default schema for the HVR capture user).
- The heartbeat table becomes part of every data replication channel that captures from this source and is integrated into every target.

- The heartbeat table is populated by a script, taking current capture state information from the capture state file and storing it in the database table. With the table being replicated, the data will make it into the target as part of data replication. Optionally, the heartbeat table also includes current Table Enrollment information.
- The script to populate the heartbeat table can be scheduled to run by the OS scheduler on the hub server, or may be invoked as an agent plugin as part of the capture cycle (with optionally some optimizations to limit the overhead to store state information as part of the capture cycle).
- Data replication recovery reads the state of the heartbeat table on the target, allowing it to recreate the capture state file to avoid data loss due to missing long-running transactions, and regenerate any transaction files that were not yet applied to the target will be recreated (since the heartbeat table becomes part of the regular data replication stream).

Using a heartbeat table requires privileges and database objects beyond the out-of-the-box HVR installation, as well as setup steps that go beyond regular running of HVR. Please contact your HVR representative should you need professional services help to implement these.

Conclusion

Data replication availability has a number of dependencies, including the source(s) and target(s) availability, open network communication, and software to function as designed. Individual components in such a complex setup may fail, resulting in replication downtime. However, downtime in replication doesn’t necessarily mean no access to data. Data in the target(s) would be stale and become staler as time progresses, similar to data replication introducing latency.

This paper discusses strategies to achieve High Availability (HA) for HVR replication. To what extent you invest in an HA setup depends on your Recovery Time Objective (RTO) related to the direct cost or risk of replication downtime, but also your willingness to manage a more complex setup. Alternatives to high availability include restore and recovery from a backup, using a disaster recovery (DR) environment, and automating the disaster recovery through a custom heartbeat table.