



EDB

High Availability Routing for Postgres (HARP)

Release 2.0.0

EnterpriseDB

Dec 19, 2021

CONTENTS

1	HARP Functionality Overview	3
1.1	The Importance of Quorum	3
1.2	Reducing Write Targets	3
1.3	Basic Architecture	4
1.4	How it Works	5
1.4.1	Determining a Leader	6
1.4.2	Connection Routing	6
1.4.3	Colocation	7
1.5	Recommended Architecture and Use	7
1.5.1	Location Configuration	7
1.5.2	BDR 3.7 Compatibility	9
2	Installation	11
2.1	Software Versions	11
2.2	TPAExec	11
2.3	Package Installation	12
2.3.1	etcd Packages	12
2.4	Consensus layer	12
2.4.1	etcd	12
2.4.2	BDR	12
3	Configuring HARP for Cluster Management	15
3.1	Standard Configuration	15
3.1.1	Cluster Name	15
3.1.2	DCS Settings	16
3.1.3	HARP Manager Specific	17
3.1.4	HARP Proxy Specific	17
3.2	Run-Time Directives	19
3.2.1	Cluster Wide	19
3.2.2	Node Directives	19
3.2.3	Proxy Directives	22
4	Cluster Bootstrapping	27
4.1	Cluster-Wide Bootstrapping	27
4.2	Location Bootstrapping	28
4.3	Node Bootstrapping	28
4.4	Proxy Bootstrapping	29
5	HARP Manager	31
5.1	How it Works	31

5.2	Configuration	32
5.3	Usage	32
5.4	Disabling and Re-enabling HARP Manager Control of Postgres	33
6	HARP Proxy	35
6.1	How it Works	35
6.2	Configuration	35
6.3	Usage	36
6.4	PgBouncer Configuration File	36
6.5	Disabling and Re-enabling HARP Proxy Node Management	36
6.6	Passthrough User Authentication	37
7	harpctl Command-line Tool	39
7.1	Synopsis	39
7.2	Configuration	40
7.3	Execution	40
7.4	harpctl apply	40
7.5	harpctl fence	41
7.6	harpctl get	41
7.6.1	harpctl get cluster	42
7.6.2	harpctl get leader	42
7.6.3	harpctl get location	42
7.6.4	harpctl get locations	42
7.6.5	harpctl get logs	43
7.6.6	harpctl get node	43
7.6.7	harpctl get nodes	43
7.6.8	harpctl get proxy	43
7.6.9	harpctl get proxies	44
7.7	harpctl manage	44
7.8	harpctl promote	44
7.9	harpctl set	45
7.9.1	harpctl set cluster	45
7.9.2	harpctl set location	45
7.9.3	harpctl set node	45
7.9.4	harpctl set proxy	46
7.10	harpctl unfence	46
7.11	harpctl unmanage	46
8	Consensus Layer Considerations	47
8.1	BDR Driver Compatibility	47
8.2	Maintaining Quorum	47
8.2.1	Multi-Consensus Variant	48
8.2.2	TPAexec and Consensus	48
9	Security and Roles	51
9.1	Postgres Permissions	51
9.2	BDR Permissions	51
9.3	BDR Consensus Permissions	52
10	Release Notes	53
10.1	Release 2.0.0 (2021-12-01)	53
10.1.1	Engine	53
10.1.2	Configuration	53
10.1.3	Enhancements	53
10.2	Release 1.0.1 (2021-06-23)	54

10.2.1	Documentation	54
10.2.2	Bug fixes	54
10.3	Release 1.0 (2021-06-15)	54
10.3.1	Enhancements	54
10.3.2	Bug fixes	54
10.4	Release 0.2 (2021-02-23)	54
10.5	Release 0.1 (2020-08-13)	55

High Availability Routing for Postgres (HARP) is a cluster management tool for [Bi-directional Replication \(BDR\)](#) clusters. The core design of the tool is to route all application traffic within a single data center or region to only one node at a time. This node, designated the Lead Master, acts as the principle write target to reduce the potential for data conflicts.

HARP leverages a distributed consensus model to determine availability of the BDR nodes in the cluster. On failure or unavailability of the Lead Master, HARP elects a new Lead Master and redirects application traffic accordingly.

Together with the core capabilities of BDR, this mechanism of routing application traffic to the Lead Master node enables fast failover and switchover without risk of data loss.

HARP requires BDR versions 3.6 and above.

HARP FUNCTIONALITY OVERVIEW

HARP is a new approach to High Availability for BDR clusters. It leverages consensus-driven Quorum to determine the correct connection end-point in a semi-exclusive manner to prevent unintended multi-node writes from an application.

1.1 The Importance of Quorum

The central purpose of HARP is to enforce full Quorum on any Postgres cluster it manages. Quorum is merely a term generally applied to a voting body that mandates a certain minimum of attendees are available to make a decision. Or perhaps even more simply: Majority Rules.

In order for any vote to end in a result other than a tie, an odd number of nodes must constitute the full cluster membership. Quorum however does not strictly demand this restriction; a simple majority will suffice. This means that in a cluster of N nodes, Quorum requires a minimum of $N/2+1$ nodes to hold a meaningful vote.

All of this ensures the cluster is always in agreement regarding which node should be “in charge”. For a BDR cluster consisting of multiple nodes, this determines which node is the primary write target. HARP designates this node as the Lead Master.

1.2 Reducing Write Targets

The consequence of ignoring the concept of Quorum, or applying it insufficiently, may lead to a Split Brain scenario where the “correct” write target is ambiguous or unknowable. In a standard Postgres cluster, it is important that only a single node is ever writable and sending replication traffic to the remaining nodes.

Even in Multi-Master capable approaches such as BDR, it can be beneficial to reduce the amount of necessary conflict management to derive identical data across the cluster. In clusters that consist of multiple BDR nodes per physical location or region, this usually means a single BDR node acts as a “Leader” and remaining nodes are “Shadows”. These Shadow nodes are still writable, but doing so is discouraged unless absolutely necessary.

By leveraging Quorum, it’s possible for all nodes to agree exactly which Postgres node should represent the entire cluster, or a local BDR region. Any nodes that lose contact with the remainder of the Quorum, or are overruled by it, by definition cannot become the cluster Leader.

This prevents Split Brain situations where writes unintentionally reach two Postgres nodes. Unlike technologies such as VPNs, Proxies, load balancers, or DNS, a Quorum-derived consensus cannot be circumvented by mis-configuration or network partitions. So long as it’s possible to contact the Consensus layer to determine the state of the Quorum maintained by HARP, only one target is ever valid.

1.3 Basic Architecture

The design of HARP comes in essentially two parts consisting of a Manager and a Proxy. The following diagram describes how these interact with a single Postgres instance:

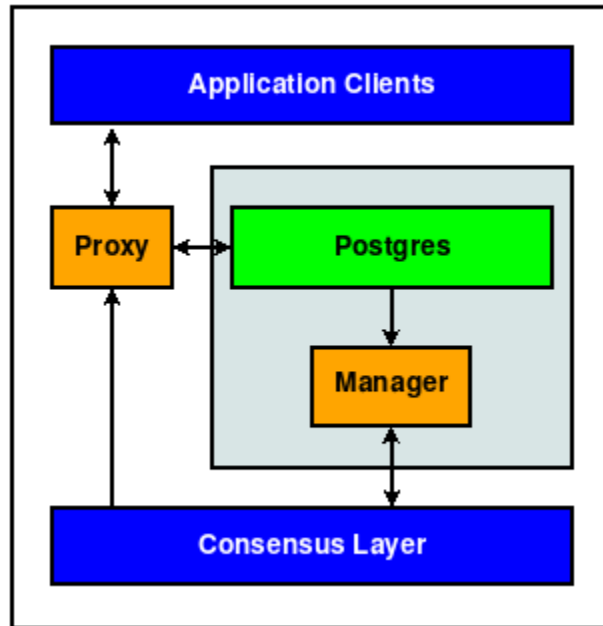


Fig. 1: HARP Unit

The Consensus Layer is an external entity where Harp Manager maintains information it learns about its assigned Postgres node, and HARP Proxy translates this information to a valid Postgres node target. Because Proxy obtains the node target from the Consensus Layer, several such instances may exist independently.

While using BDR itself as the Consensus Layer, each server node resembles this variant instead.

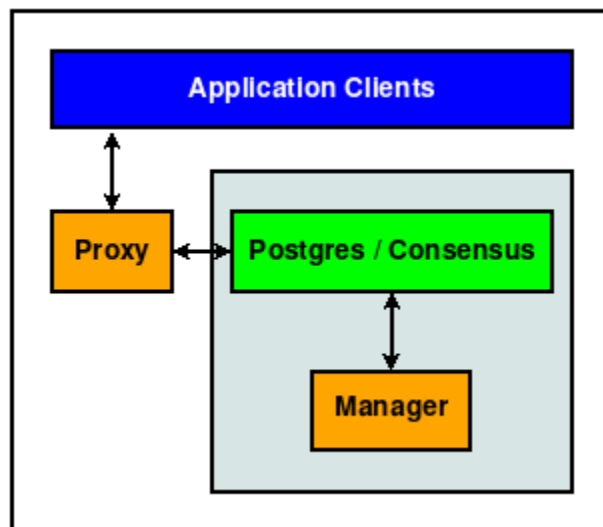


Fig. 2: HARP Unit w/BDR Consensus

In either case, each unit consists of the following elements:

- A Postgres or EDB instance
- A Consensus Layer resource, meant to track various attributes of the Postgres instance
- A HARP Manager process to convey the state of the Postgres node to the Consensus Layer
- A HARP Proxy service that directs traffic to the proper Lead Master node, as derived from the Consensus Layer

Not every application stack has access to additional node resources specifically for the Proxy component, so it can be combined with the application server to simplify the stack itself.

This is a typical design using two BDR nodes in a single Data Center organized in a Lead Master / Shadow Master configuration:

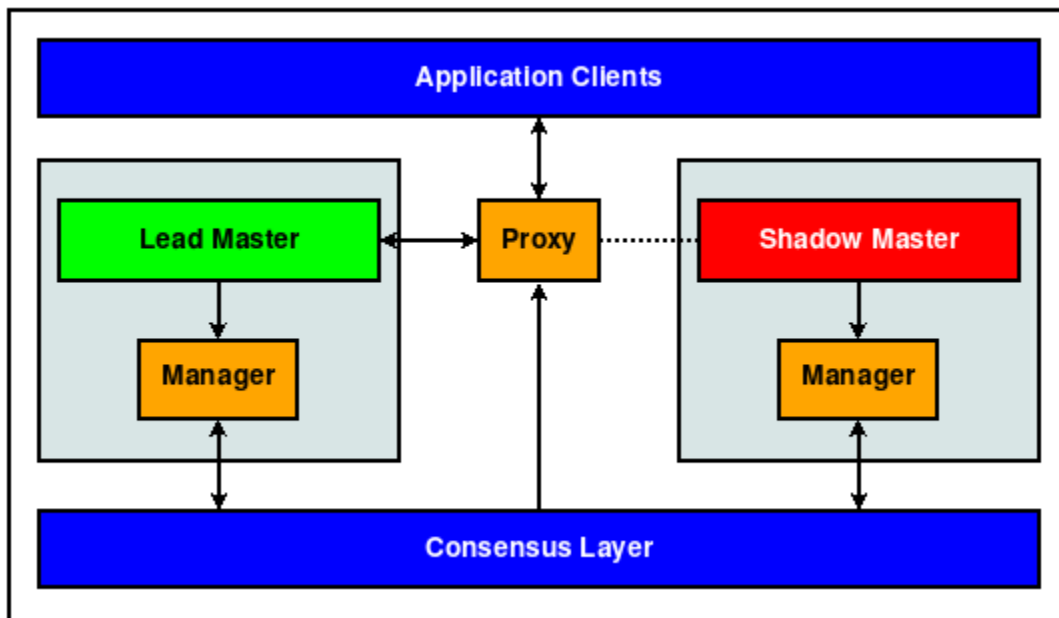


Fig. 3: HARP Cluster

Note that when using BDR itself as the HARP Consensus Layer, at least three fully qualified BDR nodes must be present to ensure a quorum majority.

(Not shown in the above diagram are connections between BDR nodes.)

1.4 How it Works

When managing a BDR cluster, HARP maintains at most one “Leader” node per defined Location. Canonically this is referred to as the Lead Master. Other BDR nodes which are eligible to take this position are Shadow Master state until such a time they take the Leader role.

Applications may contact the current Leader only through the Proxy service. Since the Consensus Layer requires Quorum agreement before conveying Leader state, any and all Proxy services will direct traffic to that node.

At a high level, this is ultimately what prevents application interaction with multiple nodes simultaneously.

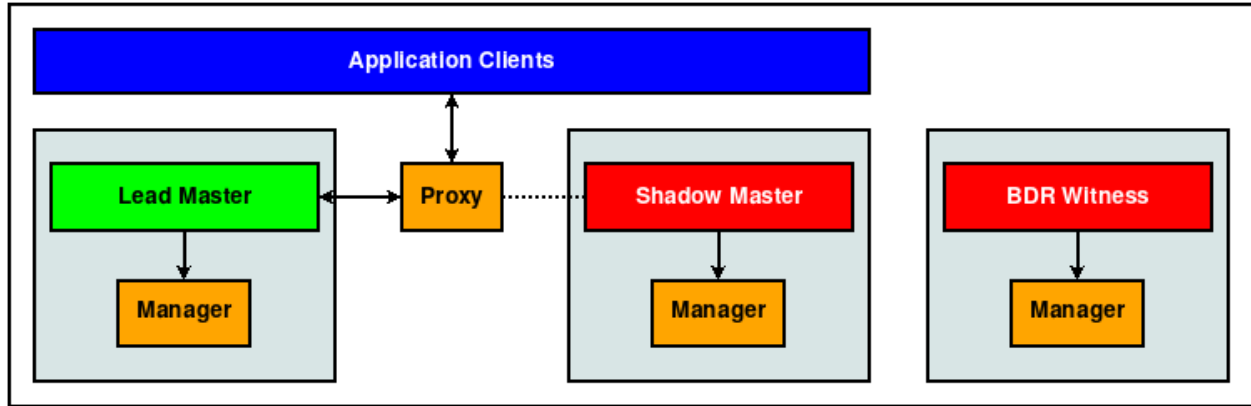


Fig. 4: HARP Cluster w/BDR Consensus

1.4.1 Determining a Leader

As an example, consider the role of Lead Master within a locally subdivided BDR Always-On group as may exist within a single data center. When any Postgres or Manager resource is started, and after a configurable refresh interval, the following must occur:

1. The Manager checks the status of its assigned Postgres resource. - If Postgres is not running, try again after configurable timeout. - If Postgres is running, continue.
2. The Manager checks the status of the Leader lease in the Consensus Layer. - If the lease is unclaimed, acquire it and assign the identity of the Postgres instance assigned to this Manager. This lease duration is configurable, but setting it too low may result in unexpected leadership transitions. - If the lease is already claimed by us, renew the lease TTL. - Otherwise do nothing.

Obviously a lot more happens here, but this simplified version should explain what's happening. The Leader lease can only be held by one node, and if it's held elsewhere, HARP Manager gives up and tries again later.

!!! Note * Depending on the chosen Consensus Layer, rather than repeatedly looping to check the status of the Leader lease, HARP will subscribe to notifications instead. In this case, it can respond immediately any time the state of the lease changes, rather than polling. Currently this functionality is restricted to the etcd Consensus Layer.

This means HARP itself does not hold elections or manage Quorum; this is delegated to the Consensus Layer. The act of obtaining the lease must be acknowledged by a Quorum of the Consensus Layer, so if the request succeeds, that node leads the cluster in that Location.

1.4.2 Connection Routing

Once the role of the Lead Master is established, connections are handled with a similar deterministic result as reflected by HARP Proxy. Consider a case where HAProxy needs to determine the connection target for a particular backend resource:

1. HARP Proxy interrogates the Consensus layer for the current Lead Master in its configured location.
2. If this is unset or in transition; - New client connections to Postgres are barred, but clients will accumulate and be in a paused state until a Lead Master appears. - Existing client connections are allowed to complete current transaction, and are then reverted to a similar pending state as new connections.
3. Client connections are forwarded to the Lead Master.

Note that the interplay demonstrated in this case does not require any interaction with either HARP Manager or Postgres. The Consensus Layer itself is the source of all truth from the Proxy's perspective.

1.4.3 Colocation

The arrangement of the work units is such that their organization is required to follow these principles:

1. The Manager and Postgres units must exist concomitantly within the same node.
2. The contents of the Consensus Layer dictate the prescriptive role of all operational work units.

This delegates cluster Quorum responsibilities to the Consensus Layer itself, while HARP leverages it for critical role assignments and key/value storage. Neither storage or retrieval will succeed if the Consensus Layer is inoperable or unreachable, thus preventing rogue Postgres nodes from accepting connections.

As a result, the Consensus Layer should generally exist outside of HARP or HARP managed nodes for maximum safety. Our reference diagrams reflect this in order to encourage such separation, though it is not required.

!!! Note * In order to operate and manage cluster state, BDR contains its own implementation of the Raft Consensus model. HARP may be configured to leverage this same layer to reduce reliance on external dependencies and to preserve server resources. However, there are certain drawbacks to this approach that are discussed in further depth in the section on the [Consensus Layer](#).

1.5 Recommended Architecture and Use

HARP was primarily designed to represent a BDR Always-On architecture which resides within two (or more) Data Centers and consists of at least five BDR nodes. This does not count any Logical Standby nodes.

The current and standard representation of this can be seen in the following diagram:

In this diagram, HARP Manager would exist on BDR Nodes 1-4. The initial state of the cluster would be that BDR Node 1 is the Lead master of DC A, and BDR Node 3 is the Lead Master of DC B.

This would result in any HARP Proxy resource in DC A connecting to BDR Node 1, and likewise the HARP Proxy resource in DC B connecting to BDR Node 3.

!!! Note * While this diagram only shows a single HARP Proxy per DC, this is merely illustrative and should not be considered a Single Point of Failure. Any number of HARP Proxy nodes may exist, and they will all direct application traffic to the same node.

1.5.1 Location Configuration

In order for multiple BDR nodes to be eligible to take the Lead Master lock in a location, a Location must be defined within the `config.yml` configuration file.

To reproduce the diagram above, we would have these lines in the `config.yml` configuration for BDR Nodes 1 and 2:

```
location: dca
```

And for BDR Nodes 3 and 4:

```
location: dcb
```

This applies to any HARP Proxy nodes which are designated in those respective data centers as well.

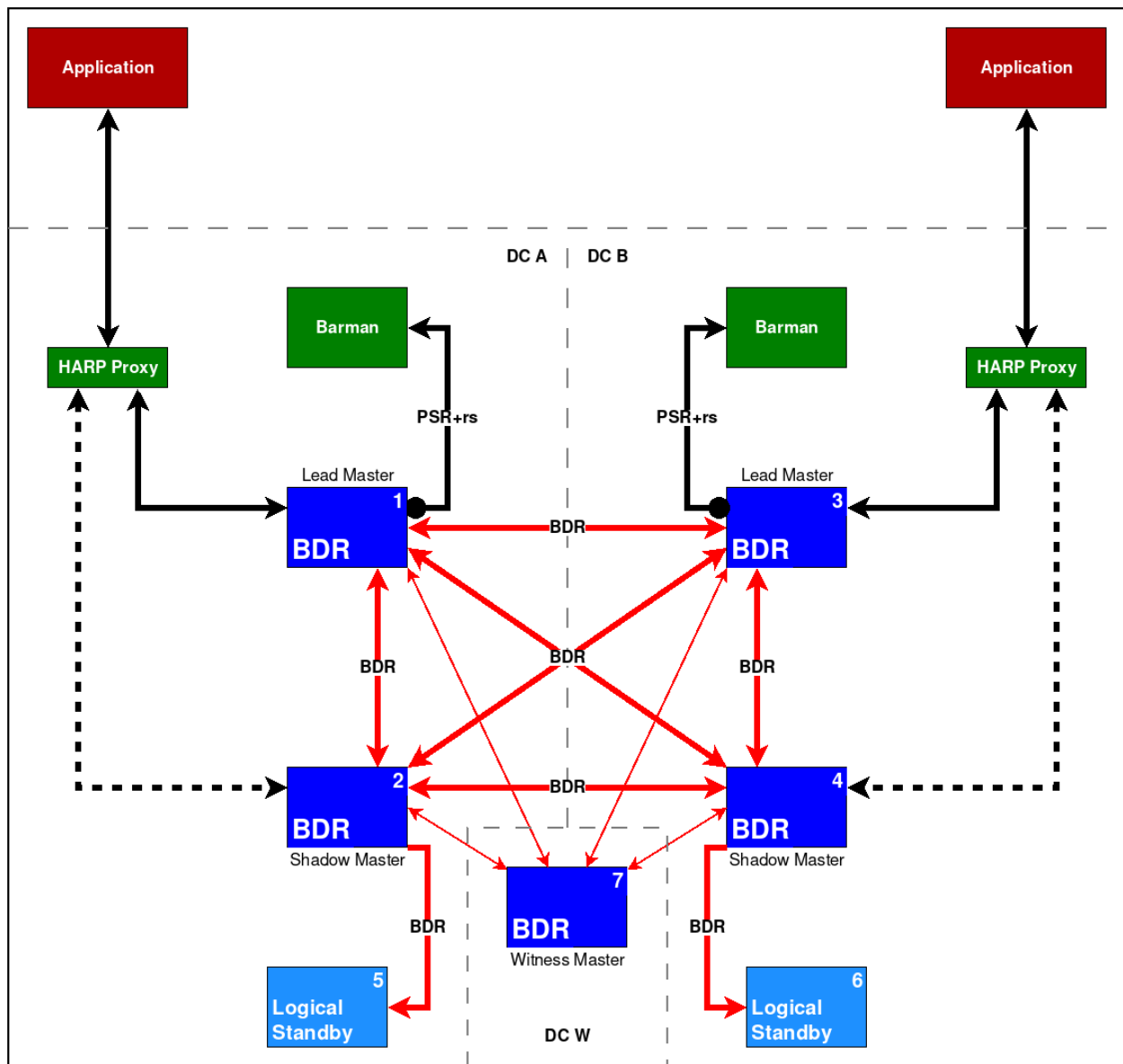


Fig. 5: BDR Always-On Reference Architecture

1.5.2 BDR 3.7 Compatibility

BDR 3.7 and above offers more direct Location definition by assigning a Location to the BDR node itself. This is done by calling the following SQLAPI function while connected to the BDR node. So for BDR Nodes 1 and 2, we might do this:

```
SELECT bdr.set_node_location('dca');
```

And for BDR Nodes 3 and 4:

```
SELECT bdr.set_node_location('dcb');
```

Afterwards, future versions of HARP Manager would derive the location field directly from BDR itself. This HARP functionality is not available yet, so we recommend using this and the setting in `config.yml` until HARP reports compatibility with this BDR API method.

INSTALLATION

A standard installation of HARP includes two system services:

- HARP Manager (`harp_manager`) on the node being managed
- HARP Proxy (`harp_router`) elsewhere

There are generally two ways to install and configure these services to manage Postgres for proper Quorum-based connection routing.

2.1 Software Versions

HARP does have dependencies on external software. These must fit a minimum version as listed here.

Software	Min Ver
etcd	3.4
PgBouncer	1.14

2.2 TPAExec

The easiest way to install and configure HARP is to use EDB's TPAexec utility for cluster deployment and management. For details on this software, see the [TPAexec product page](#).

!!! Note * TPAExec is currently only available through an EULA specifically dedicated to BDR cluster deployments. If you are unable to access the above URL, please contact your sales or account representative for more information.

TPAexec itself must be configured to recognize that cluster routing should be managed through HARP by ensuring the `TPA config.yml` file contains these attributes:

```
cluster_vars:
  * failover_manager: harp
```

!!! Note * Versions of TPAexec prior to 21.1 require a slightly different approach:

```
* cluster_vars:
*   enable_harp: true
```

After this, HARP will be installed by invoking the regular `tpaexec` commands for making cluster modifications:

```
tpaexec provision ${CLUSTER_DIR}
tpaexec deploy ${CLUSTER_DIR}
```

No other modifications should be necessary, barring cluster-specific considerations.

2.3 Package Installation

Currently CentOS/RHEL packages are provided via the EDB packaging infrastructure. For details, see the [HARP product page](#).

2.3.1 etcd Packages

Currently etcd packages for many popular Linux distributions are not available via their standard public repositories. EDB has therefore packaged etcd for RHEL and CentOS versions 7 and 8, Debian, and variants such as Ubuntu LTS. Again, access to our HARP package repository is necessary to use these libraries.

2.4 Consensus layer

HARP requires a distributed consensus layer to operate. Currently this must be either bdr or etcd. If using fewer than 3 BDR nodes, it may become necessary to rely on etcd. Otherwise any BDR service outage will reduce the consensus layer to a single node and thus prevent node consensus and disable Postgres routing.

2.4.1 etcd

If using etcd as the consensus layer, etcd must be installed either directly on the Postgres nodes, or in some separate location they can access.

To set etcd as the consensus layer, include this in the HARP `config.yml` configuration file:

```
dcs:
  * driver: etcd
  * endpoints:
  *   - host1:2379
  *   - host2:2379
  *   - host3:2379
```

When using TPAExec, all configured etcd endpoints will be entered here automatically.

2.4.2 BDR

The bdr native consensus layer is available from BDR 3.6.21 and 3.7.3. This Consensus Layer model requires no supplementary software when managing routing for a BDR cluster.

As previously mentioned, to ensure Quorum is possible in the cluster, always use more than two nodes so BDR's consensus layer remains responsive during node maintenance or outages.

To set BDR as the consensus layer, include this in the `config.yml` configuration file:

```
dcg:
* driver: bdr
* endpoints:
*   - host=host1 dbname=bdrdb user=harp_user
*   - host=host2 dbname=bdrdb user=harp_user
*   - host=host3 dbname=bdrdb user=harp_user
```

As can be seen here, the endpoints for a BDR consensus layer follow the standard Postgres DSN connection format.

CONFIGURING HARP FOR CLUSTER MANAGEMENT

The HARP configuration file follows a standard YAML style formatting which has been simplified for readability. This file can be found in the `/etc/harp` directory by default, and is named `config.yml`

The configuration file location can be explicitly provided to all HARP executables with the `-f/--config` argument.

3.1 Standard Configuration

HARP essentially operates as three components:

- HARP Manager
- HARP Proxy
- `harpctl`

Each of these use the same standard `config.yml` configuration format, which should always include the following sections:

- `cluster.name` - The name of the cluster to target for all operations.
- `dcs` - DCS driver and connection configuration for all endpoints.

Essentially this means a standard preamble will always be included for HARP operations, and will resemble this:

```
cluster:
  * name: mycluster

dcs:
  * ...
```

Other sections should be considered optional or specific to the named HARP component.

3.1.1 Cluster Name

The `name` entry under the `cluster` heading is required for *all* interaction with HARP. Each HARP cluster has a name for both disambiguation purposes and for labeling data within the DCS for the specific cluster.

HARP Managers will write information about the cluster here for consumption by HARP Proxy and `harpctl`. HARP Proxy services will direct traffic to nodes within this cluster. The `harpctl` management tool will interact with this cluster.

3.1.2 DCS Settings

Configuring the Consensus Layer is key to HARP functionality. Without the DCS, HARP has nowhere to store cluster metadata, cannot hold leadership elections, and so on. Therefore this portion of the configuration is required, though certain elements are optional.

All elements should be specified under a section named `dc`s with multiple supplementary entries that will be described here.

- **`driver`**: Required type of consensus layer to use.
 - Currently may be `etcd` or `bdr`. Support for `bdr` as a consensus layer is
 - experimental. Using `bdr` as the consensus layer reduces the
 - additional software for consensus storage, but expects a minimum of three
 - full BDR member nodes to maintain quorum during database maintenance.
- **`endpoints`**: Required list of connection strings to contact the DCS.
 - Every node of the DCS should be listed here if possible. This ensures HARP
 - will continue to function so long as a majority of the DCS is still
 - operational and reachable via the network.
- **`request_timeout`**: Time in milliseconds to consider a request as failed.
 - If HARP makes a request to the DCS and receives no response within this time
 - period, it should consider the operation as failed. This may cause the issue
 - to be logged as an error or retried, depending on the nature of the request.
 - Default: 250.
- **`ssl`**: Either `on` or `off` to enable SSL communication with the DCS.
 - Default: `off`
- **`ssl_ca_file`**: Client SSL Certificate Authority (CA) file.
- **`ssl_cert_file`**: Client SSL certificate file.
- **`ssl_key_file`**: Client SSL key file.

Example

Here is an example of how HARP should be configured to contact an `etcd` DCS consisting of three nodes:

```
dc:
* driver: etcd
* endpoints:
*   - host1:2379
*   - host2:2379
*   - host3:2379
```

3.1.3 HARP Manager Specific

Besides the generic service options required for all HARP components, Manager needs at least one more setting:

- **`**log_level**`**: One of ``DEBUG``, ``INFO``, ``WARNING``, ``ERROR``, or ``CRITICAL``
 - which may alter the amount of log output from HARP services.
- **`**name**`**: Required name of the Postgres node represented by this Manager.
 - Since Manager can only represent a specific node, that node is named here and
 - also serves to name this Manager. If this is a BDR node, it should match the
 - value used at node creation when executing the
 - `bdr.create_node(node_name, ...)` function and as reported by the
 - `bdr.local_node_summary.node_name` view column. Alphanumeric characters
 - and underscores only.

Thus a complete configuration example for HARP Manager could resemble this:

```
cluster:
  * name: mycluster

dcs:
  * driver: etcd
  * endpoints:
  *   - host1:2379
  *   - host2:2379
  *   - host3:2379

manager:
  * name: node1
  * log_level: INFO
```

Note that this is essentially the DCS contact information, any associated service customizations, the name of the cluster itself, and the name of the node. All other settings are associated with the node itself and is stored within the DCS.

Please read the section on [Node Bootstrapping](#) for more about specific node settings and initializing nodes to be managed by HARP Manager.

3.1.4 HARP Proxy Specific

Some configuration options are specific to HARP Proxy. These affect how the daemon itself operates, and thus are currently located in the `config.yml` file itself.

Proxy-based settings may be specified under a `proxy` heading, and include:

- **`**location**`**: Required name of Location HARP Proxy should represent.
 - HARP Proxy nodes are directly tied to the Location where they are running, as
 - they always direct traffic to the current Lead Master node. This must be
 - specified for any defined proxy.

- **log_level**: One of `DEBUG`, `INFO`, `WARNING`, `ERROR`, or `CRITICAL`

- which may alter the amount of log output from HARP services.
- * Default: `INFO`

- **name**: Name of this specific Proxy.

- Each Proxy node is named to ensure any associated statistics or operating
- state are available in status checks and other interactive events.

- **type**: Specifies whether pgbouncer or the experimental simple passthrough proxy will be used. All proxies must use the same proxy type. It is recommended to only experiment with the simple proxy in combination with the experimental BDR DCS.

- May be pgbouncer or simpleproxy.
-
- * Default: `pgbouncer`

- **pgbouncer_bin_dir**: Directory where PgBouncer binaries are located.

- As HARP utilizes PgBouncer binaries, it needs to know where these are
- located. This can be platform or distribution dependent, so has no
- default. The assumption is that the appropriate binaries are in the
- environment's `PATH` variable otherwise.

Example

HARP Proxy requires the cluster name, DCS connection settings, location, and name of the proxy in operation. An example lies below:

```
cluster:
  * name: mycluster

dcs:
  * driver: etcd
  * endpoints:
  *   - host1:2379
  *   - host2:2379
  *   - host3:2379

proxy:
  * name: proxy1
  * location: dc1
  * pgbouncer_bin_dir: /usr/sbin
```

All other attributes are obtained from the DCS upon Proxy startup.

3.2 Run-Time Directives

While it is possible to configure HARP Manager, HARP Proxy, or `harpctl` with a minimum of YAML in the `config.yml` file, some customizations are held within the DCS itself. These values must either be initialized via bootstrap or set specifically with `harpctl set` directives.

This section will outline these, and how they may be specified.

3.2.1 Cluster Wide

Settings here should be set under a `cluster` YAML heading during bootstrap, or modified with a `harpctl set cluster` command.

- `***event_sync_interval***`: Time in milliseconds to wait for synchronization.
 - When events occur within HARP, they do so asynchronously across the cluster.
 - HARP Managers start operating immediately when they detect metadata changes,
 - and HARP Proxies may pause traffic and start reconfiguring endpoints. This is
 - a safety interval that is meant to roughly approximate the maximum amount of
 - event time skew that may exist between all HARP components.
 -
 - For example, suppose Node A goes offline and HARP Manager on Node B commonly
 - receives this event 5 milliseconds before Node C. A setting of at least 5ms
 - would then be necessary to ensure all HARP Manager services have received the
 - event before they begin to process it.
 - This also applies to HARP Proxy.

3.2.2 Node Directives

Most node-oriented settings can be changed and subsequently applied while HARP Manager is active. These items are retained in the DCS after initial bootstrap, and thus may be modified without altering a configuration file.

Settings here should be set under a `node` YAML heading during bootstrap, or modified with a `harpctl set node` command.

- `***camo_enforcement***`: Whether CAMO queue state should be strictly enforced.
 - When set to `strict`, HARP will never allow switchover or failover to a BDR
 - CAMO partner node unless it is fully caught up with the entire CAMO queue at
 - the time of the migration. When set to `lag_only`, only standard lag
 - thresholds such as `maximum_camo_lag` are applied.
- `***dcs_reconnect_interval***`: The interval, measured in ms, between attempts that a disconnected node tries to reconnect to the DCS.
 - * Default 1000.

- ****`dsn`****: Required full connection string to the managed Postgres node.

- This parameter applies equally to all HARP services and enables
- micro-architectures which run only one service per container.
- !!! Note

- HARP sets the ``sslmode`` argument to ``require`` by default and will prevent connections to servers that do not require SSL. To disable this behavior, explicitly set this parameter to a more permissive value such as ``disable``, ``allow``, or ``prefer``.

- ****`db\_data\_dir`****: Required Postgres data directory.

- This is required by HARP Manager to start, stop, or reload the Postgres
- service. It is also the default location for configuration files, which may
- be used at a later time for controlling promotion of streaming replicas.

- ****`db\_conf\_dir`****: Location of Postgres configuration files.

- Some platforms prefer storing Postgres configuration files away from the
- Postgres data directory itself. In these cases, this should be set to that
- expected location.
- * Default `db_data_dir` - ``db_log_file```: Location of Postgres log file. * Default `/tmp/pg_ctl.out`

- ****`fence\_node\_on\_dcs\_failure`****: In the event HARP is unable to reach the DCS, several readiness keys and the leadership lease itself will expire. This will implicitly prevent a node from routing consideration. However, such a node is not officially fenced, and the manager will not stop monitoring the database if ``stop_database_when_fenced`` is set to false.

- * Default: False - ``leader_lease_duration```: Amount of time in seconds the Lead Master lease will persist if not refreshed. This allows any HARP Manager a certain grace period to refresh the lock, before expiration allows another node to obtain the Lead Master lock instead.
- * Default: 30

- ****`lease\_refresh\_interval`****: Amount of time in milliseconds between

- refreshes of the Lead Master lease. This essentially controls the time
- between each series of checks HARP Manager performs against its assigned
- Postgres node, and when the status of the node is updated in the Consensus
- layer.
- * Default: 5000 - ``max_dcs_failures```: The amount of DCS request failures before marking a node as fenced according to `fence_node_on_dcs_failure``. This prevents transient communication disruptions from shutting down database nodes.
- * Default: 10 - ``maximum_lag```: Highest allowable variance (in bytes) between last recorded LSN of previous Lead Master and this node before being allowed to take the Lead Master lock. This prevents nodes experiencing terminal amounts of lag from taking the Lead Master lock. Set to -1 to disable this check.

- * Default: 1048576 (1MB)

• **`**`maximum_camo_lag`**`: Highest allowable variance (in bytes) between last**

- received LSN and applied LSN between this node and its CAMO partner(s).
- This should only apply to clusters where CAMO is both available and enabled.
- Thus this only applies to BDR EE clusters where `pg2q.enable_camo` is set.
- Clusters with particularly stringent CAMO apply queue restrictions should set
- this very low, or even to 0 to avoid any unapplied CAMO transactions. Set to
- -1 to disable this check.
- * Default: 1048576 (1MB)

• **`**`ready_status_duration`**`: Amount of time in seconds the node's readiness**

- status will persist if not refreshed. This is a failsafe that will remove a
- node from being contacted by HARP Proxy if the HARP Manager in charge if it
- stops operating.
- * Default: 30

• **`**`db_bin_dir`**`: Directory where Postgres binaries are located.**

- As HARP utilizes Postgres binaries, such as `pg_ctl`, it needs to know where
- these are located. This can be platform or distribution dependent, so has no
- default. The assumption is that the appropriate binaries are in the
- environment's `PATH` variable otherwise.

• **`**`priority`**`: Any numeric value greater than or equal to 0.**

- In the case two nodes have an equal amount of lag and other qualified
- criteria to take the Lead Master lease, this acts as an additional ranking
- value to prioritize one node over another. Any node where this option is set
- to 0 will be unable to take the Lead Master role, even when attempting to
- explicitly set the Lead Master using `harpctl`.
- * Default: 100

• **`**`safety_interval`**`: Time in milliseconds required before allowing routing**

- to a newly promoted Lead Master. This is intended to allow automated checks
- against HARP Router to fail across the cluster before transitioning new
- connections to the promoted node. This helps enforce fully synchronized
- routing targets. 0 to disable.
- * Default: 100

• **`**`stop_database_when_fenced`**`: Rather than simply removing a node from all possible routing, stop the database on a node when it is fenced. This is an extra safeguard to prevent data from other sources than HARP Proxy from reaching the database, or in case proxies are unable to disconnect clients for some other**

(continued from previous page)

- * Default: False - ``consensus\_timeout``: Amount of milliseconds before aborting a read or write to the consensus layer. In the event the consensus layer loses quorum or becomes unreachable, we want near-instant errors rather than infinite timeouts. This prevents blocking behavior in such cases. Note: When using bdr as the consensus layer, the highest recognized timeout is 1000ms.
- * Default: 250

All of these run-time directives can be modified via `harpctl`. Consider if we wished to decrease the `lease_refresh_interval` to 100ms on node1:

```
harpctl set node node1 lease_refresh_interval=100
```

3.2.3 Proxy Directives

Certain settings to the Proxy can be changed while the service is active. These items are retained in the DCS after initial bootstrap, and thus may be modified without altering a configuration file. Many of these settings are direct mappings to their PgBouncer equivalent, and we will note these where relevant.

Settings here should be set under a `proxies` YAML heading during bootstrap, or modified with a `harpctl set proxy` command.

- ```auth_file```: The full path to a PgBouncer-style `userlist.txt` file.

- HARP Proxy will use this file to store a pgbouncer user which will have
- access to PgBouncer's Admin database. This file may be used for other users
- as well. Proxy will modify this file to add and modify the password for the
- pgbouncer user.
- * Default `/etc/harp/userlist.txt`

- ```auth_type```: What type of Postgres authentication to use for password

- matching. This is actually a PgBouncer setting and is not fully compatible
- with the Postgres `pg_hba.conf` capabilities. We recommend using `md5`, `pam`
- `cert`, or `scram-sha-256`.
- * Default `md5`

- ```auth_query```: Query to verify a user's password with Postgres.

- Direct access to `pg_shadow` requires admin rights. It's preferable to use a
- non-superuser that calls a `SECURITY DEFINER` function instead. If using
- TPAexec to create a cluster, a function named `pgbouncer_get_auth` will be
- installed on all databases within the `pg_catalog` namespace to fulfill this
- purpose.

- ```auth_user```: If `auth_user` is set, then any user not specified in

- `auth_file` will be queried through the `auth_query` query from `pg_shadow`

- in the database, using `auth_user`. The password of `auth_user` will be
- taken from `auth_file`.

• `**`client_tls_ca_file`**`: Root certificate file to validate client

- certificates. Requires `client_tls_sslmode` to be set.

• `**`client_tls_cert_file`**`: Certificate for private key. Clients can

- validate it. Requires `client_tls_sslmode` to be set.

• `**`client_tls_key_file`**`: Private key for PgBouncer to accept client

- connections. Requires `client_tls_sslmode` to be set.

• `**`client_tls_protocols`**`: Which TLS protocol versions are allowed for

- client connections.
- Allowed values: `tlsv1.0`, `tlsv1.1`, `tlsv1.2`, `tlsv1.3`.
- Shortcuts: `all` (`tlsv1.0`, `tlsv1.1`, `tlsv1.2`, `tlsv1.3`),
- `secure` (`tlsv1.2`, `tlsv1.3`), `legacy` (`all`).
- * Default `secure`

• `**`client_tls_sslmode`**`: Whether to enable client SSL functionality.

- May be one of `disable` `allow` `prefer` `require` `verify-ca` `verify-full`.
- * Default `disable`

• `**`database_name`**`: Required name that represents which database clients

- will use when connecting to HARP Proxy. This is a stable endpoint that will
- not change and points to the current node, database name, port, etc.,
- necessary to connect to the Lead Master. The global value `*` may be used
- here so all connections get directed to this target regardless of database
- name.

• `**`default_pool_size`**`: The maximum amount of active connections to allow

- per database / user combination. This is for connection pooling purposes,
- but will do nothing in session pooling mode. This is a PgBouncer setting.
- * Default `25`

• `**`ignore_startup_parameters`**`: By default, PgBouncer allows only

- parameters it can keep track of in startup packets: `client_encoding`,
- `datestyle`, `timezone`, and `standard_conforming_strings`. All other
- parameters will raise an error. To allow other parameters, they can be
- specified here so that PgBouncer knows that they are handled by the admin

- and it can ignore them. It is often necessary to set this to
 - `extra_float_digits` for Java applications to function properly.
 - * Default `extra_float_digits`
- **`listen_address`**: IP address(es) where Proxy should listen for
 - connections. This is a PgBouncer setting.
 - * Default `0.0.0.0`
- **`listen_port`**: System Port where Proxy should listen for connections.
 - This is a PgBouncer setting.
 - * Default `6432`
- **`max_client_conn`**: The total maximum amount of active client
 - connections that are allowed on the Proxy. This can be many orders of
 - magnitude greater than `default_pool_size`, as these are all connections that
 - have yet to be assigned a session, or have released a session for use by
 - another client connection. This is a PgBouncer setting.
 - * Default `100`
- **`monitor_interval`**: Time in seconds between Proxy checks of PgBouncer.
 - Since HARP Proxy manages PgBouncer as the actual connection management
 - layer, it needs to periodically check various status and stats to verify
 - it's still operational. Some of this information may also be logged or
 - registered to the DCS.
 - * Default `5`
- **`server_tls_protocols`**: Which TLS protocol versions are allowed for
 - server connections.
 - Allowed values: `tlsv1.0`, `tlsv1.1`, `tlsv1.2`, `tlsv1.3`.
 - Shortcuts: `all` (`tlsv1.0`, `tlsv1.1`, `tlsv1.2`, `tlsv1.3`),
 - `secure` (`tlsv1.2`, `tlsv1.3`), `legacy` (`all`).
 - * Default `secure`
- **`server_tls_sslmode`**: Whether to enable server SSL functionality.
 - May be one of `disable` `allow` `prefer` `require` `verify-ca` `verify-full`.
 - * Default `disable`
- **`session_transfer_mode`**: Method by which to transfer sessions.
 - May be one of `fast` `wait` `reconnect`.

- * Default wait
- *****`server\_transfer\_timeout`****: The number of seconds Harp proxy will wait before giving up on a PAUSE and issuing a KILL command.
-
- * Default 30

When using `harpctl` to change any of these settings for all proxies, use the `global` keyword in place of the proxy name. Example:

```
harpctl set proxy global max_client_conn 1000
```


CLUSTER BOOTSTRAPPING

In order to use HARP, a minimum amount of metadata must exist in the DCS. The process of “bootstrapping” a cluster essentially means initializing node, location, and other run-time configuration either all at once, or on a per-resource basis.

This entire process is governed through the `harpctl apply` command. For more information on this, please check the [harpctl docs](#).

This section assumes the DCS is set up and functional.

!!! Important * While examples in this document imply bootstrapping must be done section by section, this is not the case. It is possible to combine any or all of these into a single YAML document and apply it all at once. We simply split up these sections for illustrative purposes and to simplify.

4.1 Cluster-Wide Bootstrapping

Some settings are applied cluster-wide, and can be specified during bootstrapping. Currently this only applies to the `event_sync_interval` run-time directive, but others may be added later.

The format for this is as follows:

```
cluster:
  * name: mycluster
  * event_sync_interval: 100
```

Assuming that file was named `cluster.yml`, you would then apply it with the following:

```
harpctl apply cluster.yml
```

If the cluster name is not already defined within the DCS, this will also initialize that value.

!!! Important * The Cluster name parameter specified here will always override the cluster name supplied in `config.yml`. The assumption is that the bootstrap file supplies all necessary elements to bootstrap a cluster or some portion of its larger configuration. A `config.yml` file is primarily meant to control the execution of HARP Manager, HARP Proxy, or `harpctl` specifically.

4.2 Location Bootstrapping

Every HARP node is associated with at most one Location. This Location may be a single Data Center, a grouped Region consisting of multiple underlying servers, an Amazon Availability Zone, and so on. This is merely a logical structure that allows HARP to group nodes together such that only one will represent the nodes in that Location as the Lead Master.

Thus it is necessary to initialize one or more locations. The format for this is as follows:

```
cluster:
  * name: mycluster

locations:
  * - location: dc1
  * - location: dc2
```

Assuming that file was named `locations.yml`, you would then apply it with the following:

```
harpctl apply locations.yml
```

Note that when performing any manipulation of the cluster, the name should be included as a preamble so the changes get directed to the right place.

Once Locations are bootstrapped, they should show up with a quick examination:

```
> harpctl get locations
```

Cluster	Location	Leader	Previous Leader	Target Leader	Lease Renewals
mycluster	dc1				<nil>
mycluster	dc2				<nil>

As can be seen here, both locations are recognized by HARP and available for node and Proxy assignment.

4.3 Node Bootstrapping

HARP Nodes exist within a named cluster, and must have a designated name. Beyond this, all other settings are retained within the DCS itself, as they are dynamic and may affect how HARP interacts with them. To this end, each node should be bootstrapped using one or more of the run-time directives discussed in the [Configuration](#) documentation.

While bootstrapping a node, there are a few required fields:

- name
- location
- dsn
- pg_data_dir

Everything else is optional and may depend on the cluster itself. Because it is possible to bootstrap multiple nodes at once, the format generally fits this structure:

```
cluster:
  * name: mycluster
```

(continues on next page)

(continued from previous page)

```
nodes:
  * - name: node1
  *   location: dc1
  *   dsn: host=node1 dbname=bdrdb user=postgres
  *   pg_data_dir: /db/pgdata
  *   leader_lease_duration: 10
  *   priority: 500
```

Assuming that file was named `node1.yml`, you would then apply it with the following:

```
harpctl apply node1.yml
```

Once Nodes are bootstrapped, they should show up with a quick examination:

```
> harpctl get nodes

Cluster  Name  Ready Role Type Location Fenced Lease Duration
-----  -
mycluster node1 true          dc1      false  30
```

4.4 Proxy Bootstrapping

Unlike Locations or Nodes, Proxies can also supply a configuration template that is applied to all proxies within a Location. These are stored in the DCS under the `global` designation. Each proxy also requires a name to exist as an instance, but no further customization is necessary unless some setting needs a specific override.

This is because there are likely to be multiple proxies that have the same default configuration settings for the cluster, and repeating these values for every single proxy shouldn't be necessary.

Additionally, when bootstrapping the Proxy template, at least one database should be defined for connection assignments. With these notes in mind, the format for this is as follows:

```
cluster:
  * name: mycluster

proxies:
  * monitor_interval: 5
  * default_pool_size: 20
  * max_client_connections: 1000
  * database_name: bdrdb
  * instances:
  *   - name: proxy1
  *   - name: proxy2
  *   default_pool_size: 50
```

This would configure HARP for two proxies: `proxy1` and `proxy2`, which only `proxy2` would have a custom `default_pool_size`, while using the global settings otherwise.

Assuming that file was named `proxy.yml`, you would then apply it with the following:

```
harpctl apply proxy.yml
```

Once the Proxy template is bootstrapped, it should show up with a quick examination:

```
> harpctl get proxies
```

Cluster	Name	Pool Mode	Auth Type	Max Client Conn	Default Pool Size
-----	----	-----	-----	-----	-----
mycluster	global	session	md5	1000	20
mycluster	proxy1	session	md5	1000	20
mycluster	proxy2	session	md5	1000	50

HARP MANAGER

HARP Manager is a daemon which interacts with the local PostgreSQL / BDR node and stores information about its state in the Consensus Layer. Manager determines which node currently holds Leader status for a respective location, and enforces configuration (lag, CAMO lag, etc.) constraints to prevent ineligible nodes from Leader consideration.

Every Postgres node in the cluster should have an associated HARP Manager. Other nodes may exist, but they will not be able to participate as Lead or Shadow Master roles, or other functionality HARP supports in the future.

5.1 How it Works

Upon starting, HARP Manager will use `pg_ctl` to start Postgres if it is not already running. After this, it will periodically check the server as defined by the `node.lease_refresh_interval` setting. HARP Manager collects various bits of data about Postgres including:

- The node's current LSN
- If Postgres is running and accepting connections. This particular data point
- is considered a lease which must be periodically renewed. If it expires, HARP
- Proxy will remove the node from any existing routing.
- The current apply LSN position for all upstream BDR peer nodes.
- If CAMO is enabled:
 - – Name of the CAMO partner
 - – Peer CAMO state (`is_ready`)
 - – CAMO queue received and applied LSN positions
- Node type, such as whether the node is BDR or regular Postgres.
- The node's current role, such as a read/write, physical streaming replica,
- logical standby, and so on.
- BDR node state, which should be `ACTIVE` except in limited cases.
- BDR Node ID for other metadata gathering.
- Other tracking values.

!!! Important * When naming BDR nodes within HARP, the BDR node name should match the node name represented in the `node.name` configuration attribute. This should have already been done in the bootstrap process.

The data collected here is fully available to other HARP Manager processes, and is used to evaluate lag, partner readiness, and other criteria that will directly switch over and failover behavior.

After updating the node metadata, HARP Manager will either refresh the Lead Master lease if it is already held by the local node, or seek to obtain the lease if it has expired. Since the current state of all nodes is known to all other nodes, the node which was the previous Lead Master is given automatic priority ranking if present. If not, all other nodes will list themselves by LSN lag, node priority, and other criteria, and the most qualified node will seize the Lead Master lease.

This procedure happens for every defined Location where nodes are present. Thus for Locations DC1 and DC2, there would be a Lead Master node in each, with a separate lease and election process for both.

HARP Manager repeats these Postgres status checks, lease renewals, and elections repeatedly to ensure the Cluster always has a Lead Master target for connections from HARP Proxy.

5.2 Configuration

HARP Manager expects the `dc`s, `cluster`, and `manager` configuration stanzas. The following is a functional example:

```
cluster:
  * name: mycluster

dcs:
  * driver: etcd
  * endpoints:
  *   - host1:2379
  *   - host2:2379
  *   - host3:2379

manager:
  * name: node1
  * postgres_bin_dir: /usr/lib/postgresql/13/bin
```

Changes to the configuration file (default: `/etc/harp/config.yml`) can be applied by issuing `SIGHUP` to the running instance, or by calling a service-level reload.

See [Configuration](#) for further details.

5.3 Usage

This is the basic usage for HARP Manager:

```
Usage of ./harp_manager:
  * -f string
  *   Optional path to config file (shorthand)
  * --config string
  *   Optional path to config file
```

Note that there are no arguments to launch `harp_manager` as a forked daemon. This software is designed to be launched through `systemd` or within a container as a top-level process. This also means output is directed to `STDOUT` and `STDERR` for capture and access through `journald` or an attached container terminal.

5.4 Disabling and Re-enabling HARP Manager Control of Postgres

It is possible to temporarily pause HARP Manager control of Postgres. This results in a state where the daemon continues running but does not perform any operations that could affect existing behavior of the cluster. Re-enabling management causes it to resume operation.

An example of temporarily disabling node management would be:

```
harpctl unmanage node node1
```

See the [harpctl](#) documentation for more details.

Node management by HARP Manager is enabled by default.

HARP PROXY

HARP Proxy is a daemon which acts as an abstraction layer between the client application and Postgres. It interfaces with the Consensus Layer to obtain the identity of the current Lead Master node and directs traffic to that location. In the event of a planned switchover or unplanned failover, it will automatically redirect to the new Lead Master node as dictated by the DCS.

This HARP component is currently an interface layer between the DCS and PgBouncer. As such, PgBouncer is a prerequisite and should be installed in addition, in order for HARP Proxy to fully manage its activity.

6.1 How it Works

Upon starting, HARP Proxy will launch PgBouncer if it is not already running, and leave client connections in a paused state. Afterwards, it will contact the DCS to determine the identity of the Lead Master, configure PgBouncer to use this as the target for database connections, and resume connection activity. All application client traffic will then pass through PgBouncer into the current Lead Master node for the Location where this proxy is operating.

While PgBouncer is running, HARP Proxy will check its status based on the `monitor_interval` configuration setting within the DCS, and store it in the DCS for monitoring purposes. This will allow interrogation with `harpctl` to retrieve status of all configured proxies, or any one proxy in particular.

In the event the Lead Master lease is not set, HARP Proxy will pause all connection traffic until a new Lead Master is established. This also applies to circumstances when `harpctl promote` is used to invoke a planned transition to a new Lead Master. It uses a PgBouncer `PAUSE` command for this, so existing sessions are allowed to complete any pending transactions before they are held in stasis.

6.2 Configuration

HARP Proxy expects the `dc`, `cluster`, and `proxy` configuration stanzas. The following is a functional example:

```
cluster:
  * name: mycluster

dc:
  * driver: etcd
  * endpoints:
  *   - host1:2379
  *   - host2:2379
  *   - host3:2379
```

(continues on next page)

(continued from previous page)

```
proxy:
  * name: proxy1
```

6.3 Usage

This is the basic usage for HARP Proxy:

```
Usage of ./harp_proxy:
  * -f string
    Optional path to config file (shorthand)
  * --config string
    Optional path to config file
```

Note that there are no arguments to launch `harp_proxy` as a forked daemon. This software is designed to be launched through `systemd` or within a container as a top-level process. This also means output is directed to `STDOUT` and `STDERR` for capture and access through `journald` or an attached container terminal.

6.4 PgBouncer Configuration File

Since HARP Proxy currently utilizes PgBouncer for connection management and redirection, a `pgbouncer.ini` file must exist. HARP Manager builds this file based on various run-time directives as defined in the [Proxy Directives](#) documentation.

This file will be located in the same folder as the `config.yml` used by HARP Proxy. Any PgBouncer process launched by HARP Proxy will use this configuration file, and it may be used for debugging or information purposes. Modifications to this automatically generated `pgbouncer.ini` file will be lost any time HARP Proxy is restarted, so use `harpctl set proxy` to alter these settings instead.

6.5 Disabling and Re-enabling HARP Proxy Node Management

It is possible to temporarily pause HARP Proxy control of PgBouncer. This results in a state where the daemon continues running but does not perform any operations that could affect existing behavior of the cluster. Re-enabling management causes it to resume operation.

An example of temporarily disabling management of a specific proxy would be:

```
harpctl unmanage proxy proxy1
```

See the [harpctl](#) documentation for more details.

Proxy node management is enabled by default.

6.6 Passthrough User Authentication

We strongly recommend configuring HARP Proxy to use the `auth_user` and `auth_query` run-time directives. If these are not set, the PgBouncer `userlist.txt` file must include username and password hash combinations for every user PgBouncer needs to authenticate on Postgres' behalf.

this should *not* be the `pgbouncer` user itself, as this is utilized by HARP Proxy as an admin-level user in order to operate the underlying PgBouncer service.

In clusters administered by TPAexec, a function will be created and installed in the `pg_catalog` schema in the `template1` database during provisioning. This means any subsequently created databases will also include the function, and it will be available to PgBouncer regardless of which database the user is attempting to contact.

If TPAexec is not used, we still recommend this function definition:

```
CREATE OR REPLACE FUNCTION pg_catalog.pgbouncer_get_auth(p_username TEXT)
RETURNS TABLE(username TEXT, password TEXT) AS $$
BEGIN
    * RETURN QUERY
    * SELECT username::TEXT, passwd::TEXT FROM pg_catalog.pg_shadow
    * WHERE username = p_username;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER

REVOKE ALL ON FUNCTION pg_catalog.pgbouncer_get_auth(p_username TEXT)
    * FROM PUBLIC

GRANT EXECUTE ON FUNCTION pg_catalog.pgbouncer_get_auth(p_username TEXT)
    * TO <auth_user>;
```

Don't forget to substitute `<auth_user>` for the `auth_user` field supplied to HARP Proxy.

Then in the Bootstrap file, the following will complete the configuration:

```
cluster:
    * name: mycluster

proxies:
    * monitor_interval: 5
    * default_pool_size: 20
    * max_client_connections: 1000
    * auth_user: pgb_auth
    * auth_query: "SELECT * FROM pg_catalog.pgbouncer_get_auth($1)"
    * database_name: bdrdb
    * instances:
    *   - name: proxy1
    *   - name: proxy2
```

It is also possible to define these fields with `harpctl set proxy:`

```
harpctl set proxy global auth_user=pgb_auth
```

!!! Note * This means the `postgres` or `enterprisedb` OS user that launches HARP Proxy will need a `.pgpass` file so that `auth_user` can authenticate against Postgres.

HARPCTL COMMAND-LINE TOOL

`harpctl` is a command-line tool for directly manipulating the Consensus Layer contents to fit desired cluster geometry. It can be used to e.g. examine `nodestatus`, “promote” a node to Lead Master, disable/enable cluster management, bootstrap cluster settings, and so on.

7.1 Synopsis

```
$ harpctl --help
```

Usage:

```
* harpctl [command]
```

Available Commands:

```
* apply      Command to set up a cluster
* completion generate the autocompletion script for the specified shell
* fence      Fence specified node
* get        Command to get resources
* help      Help about any command
* manage     Manage Cluster
* promote    Promotes the sepcified node to be primary,
* set       Command to set resources.
* unfence    unfence specified node
* unmanage   Unmanage Cluster
* version    Command to get version information
```

Flags:

```
* -c, --cluster string      name of cluster.
* -f, --config-file string  config file (default is /etc/harp/config.yml)
* --dcs stringArray         Address of dcs endpoints. ie: 127.0.0.1:2379
* -h, --help                help for harpctl
* -o, --output string       'json, yaml'.
* -t, --toggle              Help message for toggle
```

Use "`harpctl [command] --help`" **for** more information about a command.

In addition to this basic synopsis, each of the available commands has its own series of allowable sub-commands and flags.

7.2 Configuration

It's important to be aware that `harpctl` must interact with the Consensus Layer to operate. This means a certain minimum amount of settings should be defined in `config.yml` for successful execution. This includes:

- `dc.driver`
- `dc.endpoints`
- `cluster.name`

As an example using `etcd`:

```
cluster:
  * name: mycluster
dc:
  * driver: etcd
  * endpoints:
  *   - host1:2379
  *   - host2:2379
  *   - host3:2379
```

See [Configuration](#) for details.

7.3 Execution

Execute `harpctl` like this:

```
harpctl command [flags]
```

Each command has its own series of sub-commands and flags. Further help for these are available by executing the command this way:

```
harpctl command --help
```

7.4 `harpctl apply`

It is necessary to use an `apply` command to “bootstrap” a HARP cluster using a file which defines various attributes of the intended cluster.

An `apply` command should be executed like this:

```
harpctl apply <filename>
```

This essentially creates all of the initial cluster metadata, default or custom management settings, and so on. This is done here because the DCS is used as the ultimate source of truth for managing the cluster, and this makes it possible to change these settings dynamically.

This can either be done once to bootstrap the entire cluster, once per type of object, or even on a per-node basis for the sake of simplicity.

This is an example of a bootstrap file for a single node:

```
cluster:
  * name: mycluster
nodes:
  * - name: firstnode
    dsn: host=host1 dbname=bdrdb user=harp_user
    location: dc1
    priority: 100
    db_data_dir: /db/pgdata
```

As seen here, it is good practice to always include a cluster name preamble to ensure all changes target the correct HARP cluster, in case several are operating in the same environment.

Once apply completes without error, the node will be integrated with the rest of the cluster.

!!! Note * This command can also be used to bootstrap the entire cluster at once since all defined sections are applied at the same time. However, we do not encourage this use for anything but testing as it increases the difficulty of validating each portion of the cluster during initial definition.

7.5 harpctl fence

Marks the local or specified node as **fenced**. A node with this status is essentially completely excluded from the cluster. HARP Proxy will not send it traffic, its representative HARP Manager will not claim the Lead Master lease, and further steps are also taken. If running, HARP Manager will stop Postgres on the node as well.

A **fence** command should be executed like this:

```
harpctl fence (<node-name>)
```

The node-name itself is optional; if omitted, **harpctl** will use the name of the locally configured node.

7.6 harpctl get

Fetches information stored in the Consensus Layer for various elements of the cluster. This includes nodes, locations, the cluster itself, and so on. The full list includes:

- **cluster** - Returns the Cluster state
- **leader** - Returns the current or specified location leader
- **location** - Returns current or specified location information
- **locations** - Returns list of all locations
- **logs** - Returns important events logged in the DCS
- **node** - Returns the specified Postgres node
- **nodes** - Returns list of all Postgres nodes
- **proxy** - Returns current or specified proxy information
- **proxies** - Returns list of all Proxy nodes

7.6.1 harpctl get cluster

Fetches information stored in the Consensus Layer for the current cluster:

```
> harpctl get cluster
```

Name	Leader	Previous Leader	Enabled	Lock Duration
mycluster			true	30

7.6.2 harpctl get leader

Fetches node information for the current Lead Master stored in the DCS for the specified location. If no location is passed, harpctl will attempt to derive it based on the location of the current Node where it was executed.

Example:

```
> harpctl get leader dc1
```

Cluster	Name	Ready	Role	Type	Location	Fenced	Lock Duration
mycluster	mynode	true	primary	bdr	dc1	false	30

7.6.3 harpctl get location

Fetches location information for the specified location. If no location is passed, harpctl will attempt to derive it based on the location of the current Node where it was executed.

Example:

```
> harpctl get location dc1
```

Cluster	Location	Leader	Previous Leader	Target Leader	Lease Renewals
mycluster	dc1	mynode	mynode		<nil>

7.6.4 harpctl get locations

Fetches information for all locations currently present in the DCS.

Example:

```
> harpctl get locations
```

Cluster	Location	Leader	Previous Leader	Target Leader	Lease Renewals
mycluster	dc1	mynode	mynode		<nil>
mycluster	dc2	thatnode	thatnode		<nil>

7.6.5 harpctl get logs

Fetches all significant events logged directly to the DCS.

Example:

```
> harpctl get logs
```

Cluster	Time		Id	Source	Message
-----	----		--	-----	-----
mycluster	2021-09-29 21:02:17.934711738 +0000 UTC	500	Manager	mynode:	manager started
mycluster	2021-09-29 21:05:19.869309853 +0000 UTC	500	Manager	mynode:	manager started
mycluster	2021-09-29 21:05:24.842886355 +0000 UTC	508	Manager	mynode:	attempting to ↵
					↵start database
mycluster	2021-09-29 21:05:26.185850583 +0000 UTC	503	Manager	mynode:	setting as ↵
					↵location leader with routing status ok

7.6.6 harpctl get node

Fetches node information stored in the DCS for the specified node.

Example:

```
> harpctl get node mynode
```

Cluster	Name	Ready	Role	Type	Location	Fenced	Lock	Duration
-----	----	-----	-----	-----	-----	-----	-----	-----
mycluster	mynode	true	primary	bdr	dc1	false	30	

7.6.7 harpctl get nodes

Fetches node information stored in the DCS for the all nodes in the cluster.

Example:

```
> harpctl get nodes
```

Cluster	Name	Ready	Role	Type	Location	Fenced	Lock	Duration
-----	----	-----	-----	-----	-----	-----	-----	-----
mycluster	mynode	true	primary	bdr	dc1	false	30	
mycluster	thatnode	true	primary	bdr	dc2	false	30	

7.6.8 harpctl get proxy

Fetches proxy information stored in the DCS for specified proxy. Specify global to see proxy defaults for this cluster.

Example:

```
> harpctl get proxy proxy1
```

Cluster	Name	Pool	Mode	Auth	Type	Max Client Conn	Default Pool Size
---------	------	------	------	------	------	-----------------	-------------------

(continues on next page)

(continued from previous page)

```
-----
mycluster proxy1 session md5 1000 20
```

7.6.9 harpctl get proxies

Fetches proxy information stored in the DCS for all proxies in the cluster. Additionally will list the global pseudo-proxy for default proxy settings.

Example:

```
> harpctl get proxies

Cluster  Name    Pool Mode Auth Type Max Client Conn Default Pool Size
-----
mycluster global session md5      500      25
mycluster proxy1 session md5      1000     20
mycluster proxy2 session md5      1500     30
```

7.7 harpctl manage

In the event a cluster is not in a managed state, instructs all HARP Managerservices to resume monitoring Postgres and updating the Consensus Layer. This should be done after maintenance is complete following HARP software updates or other significant changes that could affect the whole cluster.

A `manage` command should be executed like this:

```
harpctl manage cluster
```

!!! Note * Currently it is only possible to enable or disable cluster management at the `cluster` level. Later versions will also make it possible to do this for individual nodes or proxies.

7.8 harpctl promote

Promotes the local or specified node to Lead Master in the current Location. Since this is a requested event, it invokes a smooth handover where:

1. The existing Lead Master will release the Lead Master lease, provided:
 - If CAMO is enabled, the promoted node must be up to date and CAMO ready, and the CAMO queue must have less than `node.maximum_camo_lag` bytes remaining to be applied.
 - Replication lag between the old Lead Master and the promoted node is less than `node.maximum_lag2`.
 The promoted node will be the only valid candidate to take the Lead Master lease, and will do so as soon as it is released by the current holder. All other nodes will ignore the unset Lead Master lease.
 - If CAMO is enabled, the promoted node will temporarily disable client traffic until the CAMO queue is fully applied, even though it holds the Lead Master lease.
3. HARP Proxy will PAUSE connections to allow ongoing transactions to complete. Once the Lead Master lease is claimed by the promoted node, it will reconfigure PgBouncer for the new connection target and resume database traffic.

A `promote` command should be executed like this:

```
harpctl promote (<node-name>)
```

The `--force` option can be provided to forcibly set a node to Lead Master, even if it does not meet the criteria for becoming lead master. This will circumvent any verification of CAMO status or replication lag and cause an immediate transition to the promoted node.

Note that the node must be online and operational for this to succeed. This option should be used with care.

7.9 harpctl set

Sets a specific attribute in the cluster to the supplied value. This is used to tweak configuration settings for a specific node, proxy, location, or the cluster itself rather than using `apply`. This can be used for the following object types:

- `cluster` - Sets cluster-related attributes.
- `location` - Sets specific location attributes.
- `node` - Sets specific node attributes.
- `proxy` - Sets specific proxy attributes.

7.9.1 harpctl set cluster

Sets cluster-related attributes only. There's only one of these at the moment, but future versions of HARP may add more.

Example:

```
harpctl set cluster event_sync_interval=200
```

7.9.2 harpctl set location

Sets location-related attributes only. There are none of these at the moment, and calling this command will result in an error regarding unrecognized options.

!!! Note * This is a placeholder for future capabilities.

7.9.3 harpctl set node

Sets node-related attributes for the named node. Any options mentioned in the “Node Directives” section of the [Configuration](#) documentation are valid here.

Example:

```
harpctl set node mynode priority=500
```

7.9.4 harpctl set proxy

Sets proxy-related attributes for the named proxy. Any options mentioned in the “Proxy Directives” section of the [Configuration](#) documentation are valid here.

Example:

```
harpctl set proxy proxy1 max_client_conn=750
```

Use global for cluster-wide proxy defaults:

```
harpctl set proxy global default_pool_size=10
```

7.10 harpctl unfence

Removes the **fenced** attribute from the local or specified node. This will remove all previously applied cluster exclusions from the node so that it can again receive traffic or hold the Lead Master lease. Postgres will also be started if it is not running.

An unfence command should be executed like this:

```
harpctl unfence (<node-name>)
```

The node-name itself is optional; if omitted, harpctl will use the name of the locally configured node.

7.11 harpctl unmanage

Instructs all HARP Manager services in the cluster to remain running but no longer actively monitor Postgres, or modify the contents of the Consensus Layer. This will mean that any ordinary failover event such as a node outage will not result in a leadership migration. This is intended for system or HARP maintenance, and should be done prior to making changes to HARP software or other significant modifications to the cluster.

An unmanage command should be executed like this:

```
harpctl unmanage cluster
```

!!! Note * Currently it is only possible to enable or disable cluster management at the `cluster` level. Later versions will also make it possible to do this for individual nodes or proxies.

CONSENSUS LAYER CONSIDERATIONS

HARP is designed so that it can work with different implementations of Consensus Layer, also known as Distributed Control Systems (DCS).

Currently the following DCS implementations are supported:

- etcd - BDR

This section provides information specific to HARP's interaction with the supported DCS implementations.

8.1 BDR Driver Compatibility

The `bdr` native consensus layer is available from BDR versions 3.6.21 and 3.7.3 respectively.

Note that for the purpose of maintaining a voting Quorum, BDR Logical Standby nodes do not participate in consensus communications within a BDR cluster. Do not count these in the total node list to fulfill DCS Quorum requirements.

8.2 Maintaining Quorum

Clusters of any architecture require at least $n/2 + 1$ nodes to maintain Consensus via a voting Quorum. Thus a 3-node cluster may tolerate the outage of a single node, a 5-node cluster can tolerate a 2-node outage, and so on. If consensus is ever lost, HARP will become inoperable because the DCS prevents it from deterministically identifying which node is the Lead Master within a particular Location.

As a result, whichever DCS is chosen, more than half of the nodes must always be available *cluster-wide*. This can become a non-trivial element when distributing DCS nodes among two or more Data Centers. A Network Partition will prevent Quorum in any Location that cannot maintain a voting majority, and thus HARP will cease operations.

Thus an odd-number of nodes (with a minimum of 3) is crucial when building the Consensus Layer itself. An ideal case distributes nodes across a minimum of three independent locations to prevent a single Network Partition from disrupting Consensus.

One example configuration is to designate two DCS nodes in two Data Centers coinciding with the primary BDR nodes, and a fifth DCS node (such as a BDR Witness) elsewhere. Using such a design, a network partition between the two BDR Data Centers would not disrupt Consensus thanks to the independently located node.

8.2.1 Multi-Consensus Variant

HARP itself assumes one Lead Master per configured Location. Normally each Location is specified within HARP using the `location` configuration setting. By creating a separate DCS cluster per Location, it becomes possible to emulate this behavior independently of HARP.

To accomplish this, HARP should be configured in `config.yml` to use a different DCS connection target per desired Location.

HARP nodes in DC-A would use something like this:

```
location: dca
dcs:
  * driver: etcd
  * endpoints:
    * - dcs-a1:2379
    * - dcs-a2:2379
    * - dcs-a3:2379
```

While DC-B would use different hostnames corresponding to nodes in its canonical Location:

```
location: dcb
dcs:
  * driver: etcd
  * endpoints:
    * - dcs-a1:2379
    * - dcs-a2:2379
    * - dcs-a3:2379
```

There is no DCS communication between different Data Centers in this design, and thus a Network Partition between them will not impact HARP operation. A consequence of this is that HARP is completely unaware of nodes in the other Location, and each Location operates essentially as a separate HARP cluster.

This is not possible when using BDR as the DCS, as BDR maintains a Consensus Layer across all participant nodes.

A possible drawback to this approach is that `harpctl` is unable to interact with nodes outside of the current Location. It will be impossible to obtain node information, get or set the Lead Master, or any other operation that targets the other Location. Essentially this organization renders the `--location` parameter to `harpctl` unusable.

8.2.2 TPAexec and Consensus

The above considerations are integrated into TPAexec as well. When deploying a cluster using `etcd`, it will automatically construct a separate DCS cluster per Location to facilitate High Availability in favor of strict Consistency.

Thus this configuration example:

```
cluster_vars:
  * failover_manager: harp
  * harp_consensus_protocol: etcd

locations:
  * - Name: first
  * - Name: second
```

Would group any DCS nodes assigned to the `first` location together, and the `second` location would be a separate cluster. To override this behavior, configure the `harp_location` implicitly to force a particular grouping.

Thus this example would return all etcd nodes into a single cohesive DCS layer:

```
cluster_vars:
  * failover_manager: harp
  * harp_consensus_protocol: etcd

locations:
  * - Name: first
  * - Name: second
  * - Name: all_dcs

instance_defaults:
  * vars:
  *   harp_location: all_dcs
```

The harp_location override may also be necessary to favor specific node groupings when using cloud providers such as Amazon which favor AvailabilityZones within Regions in favor of traditional Data Centers.

SECURITY AND ROLES

Beyond basic package installation and configuration, HARP itself does require Postgres permissions to operate. These allow it to gather information about Postgres or BDR as necessary to maintain node status within the ConsensusLayer.

9.1 Postgres Permissions

The role specified in the `node.dsn` parameter must have been created one of the following ways:

- `CREATE USER ...`
- `CREATE ROLE ... WITH LOGIN`

This ensures the role is capable of logging into the database in order to gather diagnostic information.

Similarly, an entry must exist in `pg_hba.conf` for this role. This can be done many ways. As an example, consider a VPN subnet where all database hosts are located somewhere in `10.10.*`. In such a case, the easiest approach is to add a specific line:

#	TYPE	DATABASE	USER	ADDRESS	METHOD
hostssl		all	harp_user	10.10.1.1/16	scram-sha-256

!!! Note * In this case we've used the more modern `scram-sha-256` authentication rather than `md5` which is now deprecated. We've also elected to require SSL authentication by specifying `hostssl`.

9.2 BDR Permissions

BDR nodes have metadata and views that are only visible when certain roles are granted to the HARP-enabled user. In this case, the HARP user requires the following:

```
GRANT bdr_monitor TO harp_user;
```

The `bdr_monitor` BDR role is meant for status monitoring tools to maintain ongoing information on cluster operation, thus is perfectly suited to HARP.

9.3 BDR Consensus Permissions

When the `dc.driver` configuration parameter is set to `bdr`, HARP will utilize BDR itself as the Consensus Layer. As such, it requires access to API methods that are currently only available to the `bdr_superuser` role. This means the HARP-enabled user requires the following:

```
GRANT bdr_superuser TO foobar;
```

This may change in future versions of BDR, but currently access to the BDRconsensus model does require superuser equivalent permission.

!!! Important * BDR Superusers *are not* Postgres superusers. The `bdr_superuser` role is merely granted elevated privileges within BDR itself, such as access to restricted functions, tables, views, and other objects.

RELEASE NOTES

10.1 Release 2.0.0 (2021-12-01)

10.1.1 Engine

- Complete rewrite of system in go lang to optimize all operations
- Cluster state can now be bootstrapped or revised via YAML

10.1.2 Configuration

- Rewritten in YAML
- Configuration file changed from `harp.ini` to `config.yml`

10.1.3 Enhancements

- HARP Proxy deprecates need for HAProxy in supported architecture.
- The use of HARP Router to translate DCS contents into appropriate online or offline states for HTTP-based URI requests meant a load balancer or HAProxy was necessary to determine the Lead Master. HARP Proxy now does this automatically without periodic iterative status checks.
- Utilizes DCS key subscription to respond directly to state changes.
- With relevant cluster state changes, the cluster will respond immediately, thereby resulting in improved failover and switchover times.
- Compatibility with etcd SSL settings.
- It is now possible to communicate with etcd through SSL encryption. * Zero transaction lag on switchover.
- The new lead node will not have transactions routed to it until all replicated transactions are replayed, thereby reducing the potential for conflicts. * Experimental BDR Consensus layer
- Using BDR Consensus as the Distributed Consensus Service (DCS) reduces amount of change needed for implementations. * Experimental Proxy
- Proxy implementation for increased session control.

10.2 Release 1.0.1 (2021-06-23)

10.2.1 Documentation

- Standardize resolution of the HARP acronym

10.2.2 Bug fixes

- Fix CAMO lag check to accommodate cases where `maximum_camo_lag` is set to 0

10.3 Release 1.0 (2021-06-15)

10.3.1 Enhancements

- `--dry-run` option added to `harpctl leader set`
- minimum configuration values will be enforced
- `lock_interval` parameter can be specified as fractions of a second
- logging and output improvements
- replication lag query updated to handle parallel apply

10.3.2 Bug fixes

- `harpctl` returns an error code if `leader set` fails
- prevent corner-case failure when node peer progress not returned
- handle potentially empty node record
- catch unhandled exception when deciding the lead node candidate

10.4 Release 0.2 (2021-02-23)

This is a maintenance release with following changes:

- documentation available via the EnterpriseDB customer portal
- report non-availability of nodes other than the lead master
- when using BDR as a DCS layer, fix potential failure situations when a
- BDR node is not running
- fixes RPM packaging issue preventing a new start on fresh installations

10.5 Release 0.1 (2020-08-13)

This is an initial beta release providing HARP support for BDR, including:

- Usage of native BDR (3.6.21 and later) as a consensus layer
- Usage of etcd as a consensus layer

Note that currently HARP does not support operation on a physical streaming replica when BDR is used as a consensus layer.