



EDB

High Availability Routing for Postgres -HARP

Release 2.1.0

EDB

Jun 05, 2022

CONTENTS

1	Release Notes	3
1.1	Version 2.1.0	3
1.2	Version 2.0.3	4
1.3	Version 2.0.2	4
1.4	Version 2.0.1	4
1.5	Version 2.0.0	5
2	HARP functionality overview	7
2.1	The importance of quorum	7
2.2	Reducing write targets	7
2.3	Basic architecture	8
2.4	How it works	10
2.4.1	Determining a leader	10
2.4.2	Connection routing	10
2.4.3	Colocation	11
2.5	Recommended architecture and use	11
2.5.1	Location configuration	11
2.5.2	BDR 3.7 compatibility	13
3	Installation	15
3.1	Software versions	15
3.2	TPAExec	15
3.3	Package installation	16
3.3.1	etcd packages	16
3.4	Consensus layer	16
3.4.1	etcd	16
3.4.2	BDR	17
4	Configuring HARP for cluster management	19
4.1	Standard configuration	19
4.1.1	Cluster name	19
4.1.2	DCS settings	20
5	Cluster bootstrapping	21
5.1	Cluster-wide bootstrapping	21
5.2	Location bootstrapping	22
5.3	Node bootstrapping	22
5.4	Proxy bootstrapping	23
6	HARP Manager	25
6.1	How it works	25

6.2	Configuration	26
6.3	Usage	26
6.4	Disabling and reenabling HARP Manager control of Postgres	27
7	HARP Proxy	29
7.1	Builtin proxy: how it works	29
7.1.1	Configuration	29
7.2	PgBouncer: how it works	29
7.2.1	PgBouncer configuration file	30
7.2.2	Disabling and reenabling HARP Proxy node management	30
7.2.3	Passthrough user authentication	30
7.2.4	Configuration	31
8	harpctl command-line tool	33
8.1	Synopsis	33
8.2	Configuration	34
8.3	Execution	34
8.4	harpctl apply	34
8.5	harpctl fence	35
8.6	harpctl get	35
8.6.1	harpctl get cluster	36
8.6.2	harpctl get leader	36
8.6.3	harpctl get location	36
8.6.4	harpctl get locations	36
8.6.5	harpctl get node	37
8.6.6	harpctl get nodes	37
8.6.7	harpctl get proxy	37
8.6.8	harpctl get proxies	37
8.7	harpctl manage	38
8.8	harpctl promote	38
8.9	harpctl set	39
8.9.1	harpctl set cluster	39
8.9.2	harpctl set node	39
8.9.3	harpctl set proxy	39
8.10	harpctl unfence	39
8.11	harpctl unmanage	40
9	Consensus layer considerations	41
9.1	BDR driver compatibility	41
9.2	Maintaining quorum	41
9.2.1	Multi-consensus variant	42
9.2.2	TPAexec and consensus	42
10	Security and roles	45
10.1	Postgres permissions	45
10.2	BDR permissions	45
10.3	BDR consensus permissions	46

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 in 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.

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 later.

RELEASE NOTES

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 in 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.

The release notes in this section provide information on what was new in each release.

Version	Release Date
<i>Version 2.1.0</i>	2022 May 17
<i>Version 2.0.3</i>	2022 Mar 31
<i>Version 2.0.2</i>	2022 Feb 24
<i>Version 2.0.1</i>	2021 Jan 31
<i>Version 2.0.0</i>	2021 Dec 01

1.1 Version 2.1.0

This is a minor release of HARP 2 that includes new features as well as fixes for issues identified in previous versions.

Type	Description
Feature	The BDR DCS now uses a push notification from the consensus rather than through polling nodes. This change reduces the time for new leader selection and the load that HARP does on the BDR DCS since it doesn't need to poll in short intervals anymore.
Feature	TPA now restarts each HARP Proxy one by one and wait until they come back to reduce any downtime incurred by the application during software upgrades.
Feature	The support for embedding PGBouncer directly into HARP Proxy is now deprecated and will be removed in the next major release of HARP. It's now possible to configure TPA to put PGBouncer on the same node as HARP Proxy and point to that HARP Proxy.
Bug Fix	<i>harpctl promote</i> <node_name> would occasionally promote a different node than the one specified. This has been fixed. (RT75406)
Bug Fix	Fencing would sometimes fail when using BDR as the Distributed Consensus Service. This has been corrected.
Bug Fix	<i>harpctl apply</i> no longer turns off routing for leader after the cluster has been established. (RT80790)
Bug Fix	Harp-manager no longer exits if it cannot start a failed database. Harp-manager will keep retrying with randomly increasing periods. (RT78516)

continues on next page

Table 2 – continued from previous page

Type	Description
Bug Fix	The internal pgbouncer proxy implementation had a memory leak. This has been remediated.

1.2 Version 2.0.3

This is a patch release of HARP 2 that includes fixes for issues identified in previous versions.

Type	Description
Enhancement	HARP Proxy supports read-only user dedicated TLS Certificate (RT78516)
Bug Fix	HARP Proxy continues to try and connect to DCS instead of exiting after 50 seconds. (RT75406)

1.3 Version 2.0.2

This is a patch release of HARP 2 that includes fixes for issues identified in previous versions.

Type	Description
Enhancement	BDR consensus now generally available. HARP offers multiple options for Distributed Consensus Service (DCS) source: etcd and BDR. The BDR consensus option can be used in deployments where etcd isn't present. Use of the BDR consensus option is no longer considered beta and is now supported for use in production environments.
Enhancement	Transport layer proxy now generally available. HARP offers multiple proxy options for routing connections between the client application and database: application layer (L7) and transport layer (L4). The network layer 4 or transport layer proxy simply forwards network packets, and layer 7 terminates network traffic. The transport layer proxy, previously called simple proxy, is no longer considered beta and is now supported for use in production environments.

1.4 Version 2.0.1

This is a patch release of HARP 2 that includes fixes for issues identified in previous versions.

Type	Description
Enhancement	Support for selecting a leader per location rather than relying on DCS like etcd to have separate setup in different locations. This still requires a majority of nodes to survive loss of a location, so an odd number of both locations and database nodes is recommended.
Enhancement	The BDR DCS now uses a push notification from the consensus rather than through polling nodes. This change reduces the time for new leader selection and the load that HARP does on the BDR DCS since it doesn't need to poll in short intervals anymore.
Enhancement	TPA now restarts each HARP Proxy one by one and wait until they come back to reduce any downtime incurred by the application during software upgrades.

continues on next page

Table 5 – continued from previous page

Type	Description
Enhancement	The support for embedding PGBouncer directly into HARP Proxy is now deprecated and will be removed in the next major release of HARP. It's now possible to configure TPA to put PGBouncer on the same node as HARP Proxy and point to that HARP Proxy.
Bug Fix	<i>harpctl promote <node_name></i> would occasionally promote a different node than the one specified. This has been fixed. [Support Ticket #75406]
Bug Fix	Fencing would sometimes fail when using BDR as the Distributed Consensus Service. This has been corrected.
Bug Fix	<i>harpctl apply</i> no longer turns off routing for leader after the cluster has been established. [Support Ticket #80790]
Bug Fix	Harp-manager no longer exits if it cannot start a failed database. Harp-manager will keep retrying with randomly increasing periods. [Support Ticket #78516]
Bug Fix	The internal pgbouncer proxy implementation had a memory leak. This has been remediated.

1.5 Version 2.0.0

This is new major release of HARP that constitutes of complete rewrite of the product.

Type	Description
Engine	Complete rewrite of system in golang to optimize all operations
Engine	Cluster state can now be bootstrapped or revised via YAML
Feature	Configuration now in YAML, configuration file changed from <i>harp.ini</i> to <i>config.yml</i>
Feature	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.
Feature	Utilizes DCS key subscription to respond directly to state changes. With relevant cluster state changes, the cluster responds immediately, resulting in improved failover and switchover times.
Feature	Compatibility with etcd SSL settings. It is now possible to communicate with etcd through SSL encryption.
Feature	Zero transaction lag on switchover. Transactions are not routed to the new lead node until all replicated transactions are replayed, thereby reducing the potential for conflicts.
Feature	Experimental BDR Consensus layer. Using BDR Consensus as the Distributed Consensus Service (DCS) reduces the amount of change needed for implementations.
Feature	Experimental built-in proxy. Proxy implementation for increased session control.

HARP FUNCTIONALITY OVERVIEW

HARP is a new approach to high availability for BDR clusters. It leverages a consensus-driven quorum to determine the correct connection endpoint in a semi-exclusive manner to prevent unintended multi-node writes from an application.

2.1 The importance of quorum

The central purpose of HARP is to enforce full quorum on any Postgres cluster it manages. Quorum is a term applied to a voting body that mandates a certain minimum of attendees are available to make a decision. More simply: majority rules.

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, doesn't strictly demand this restriction; a simple majority is enough. 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 the node that is "in charge." For a BDR cluster consisting of multiple nodes, this determines the node that is the primary write target. HARP designates this node as the lead master.

2.2 Reducing write targets

The consequence of ignoring the concept of quorum, or not applying it well enough, can lead to a "split brain" scenario where the "correct" write target is ambiguous or unknowable. In a standard Postgres cluster, it's 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 help 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 "shadow." These shadow nodes are still writable, but writing to them is discouraged unless absolutely necessary.

By leveraging quorum, it's possible for all nodes to agree on the exact Postgres node to 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 can't become the cluster leader.

This restriction prevents split-brain situations where writes unintentionally reach two Postgres nodes. Unlike technologies such as VPNs, proxies, load balancers, or DNS, you can't circumvent a quorum-derived consensus by misconfiguration 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.

2.3 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:

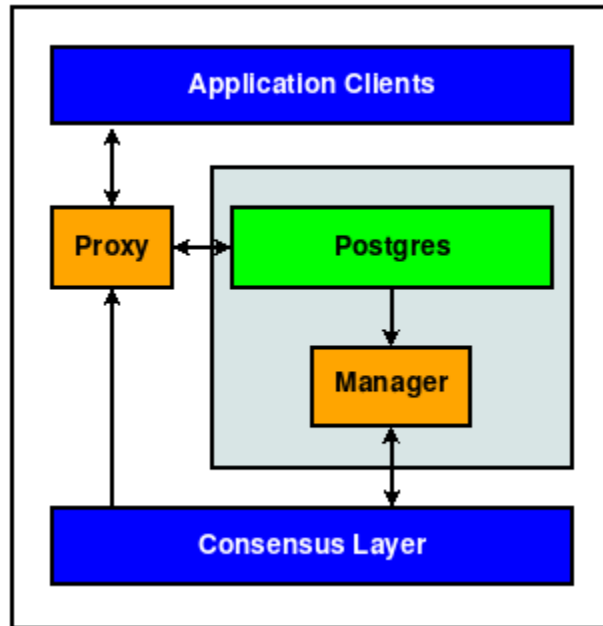


Fig. 1: HARP Unit

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 can exist independently.

While using BDR as the consensus layer, each server node resembles this variant instead:

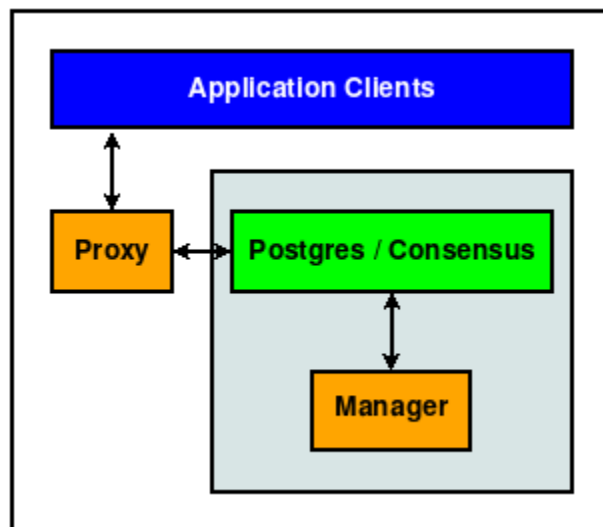


Fig. 2: HARP Unit w/BDR Consensus

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.

This is a typical design using two BDR nodes in a single data center organized in a lead master/shadow master configuration:

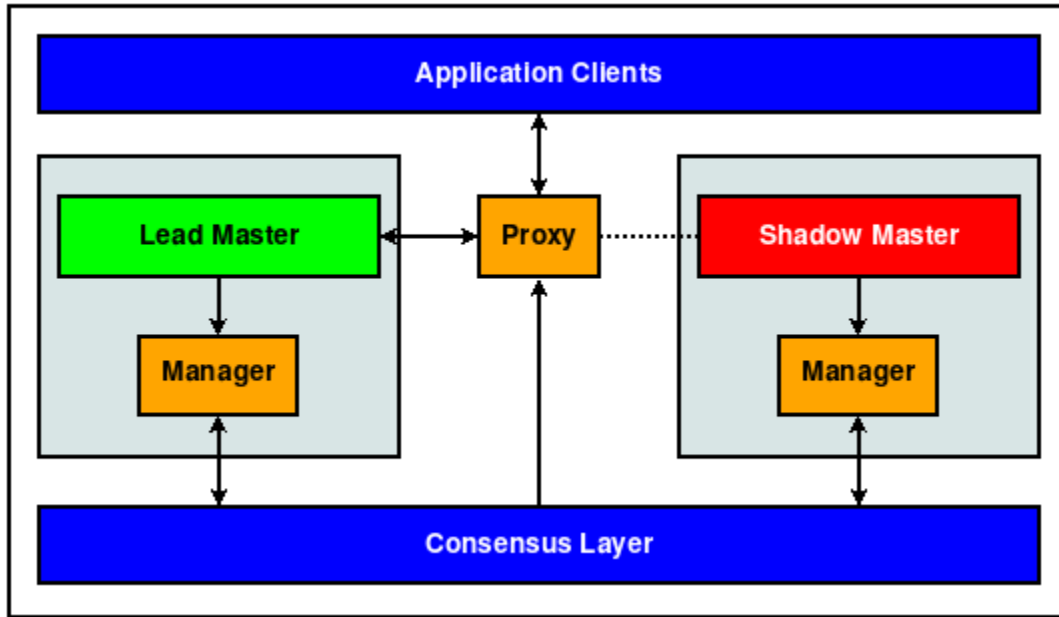


Fig. 3: HARP Cluster

When using BDR as the HARP consensus layer, at least three fully qualified BDR nodes must be present to ensure a quorum majority. (Not shown in the diagram are connections between BDR nodes.)

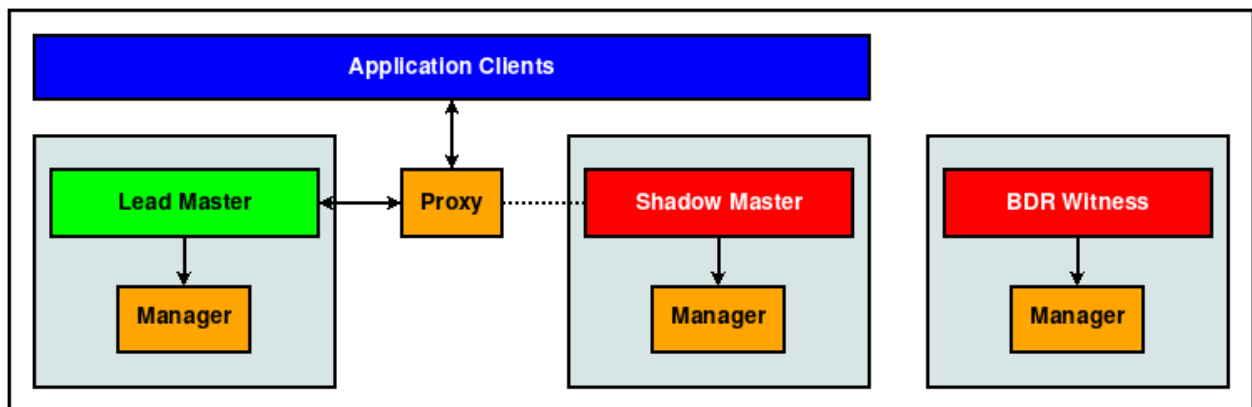


Fig. 4: HARP Cluster w/BDR Consensus

2.4 How it works

When managing a BDR cluster, HARP maintains at most one leader node per defined location. This is referred to as the lead master. Other BDR nodes that are eligible to take this position are shadow master state until they take the leader role.

Applications can contact the current leader only through the proxy service. Since the consensus layer requires quorum agreement before conveying leader state, proxy services direct traffic to that node.

At a high level, this mechanism prevents simultaneous application interaction with multiple nodes.

2.4.1 Determining a leader

As an example, consider the role of lead master in a locally subdivided BDR Always-On group as can exist in 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 isn't 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 can result in unexpected leadership transitions. - If the lease is already claimed by us, renew the lease TTL. - Otherwise do nothing.

A lot more occurs, but this simplified version explains what's happening. The leader lease can be held by only 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 subscribes to notifications. 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 doesn't hold elections or manage quorum, which is delegated to the consensus layer. A quorum of the consensus layer must acknowledge the act of obtaining the lease, so if the request succeeds, that node leads the cluster in that location.

2.4.2 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 HARP Proxy 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 accumulate and are in a paused state until a lead master appears. - Existing client connections are allowed to complete current transactions and are then reverted to a similar pending state as new connections.
3. Client connections are forwarded to the lead master.

The interplay shown in this case doesn't require any interaction with either HARP Manager or Postgres. The consensus layer is the source of all truth from the proxy's perspective.

2.4.3 Colocation

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

1. The manager and Postgres units must exist concomitantly in the same node.
2. The contents of the consensus layer dictate the prescriptive role of all operational work units.

This arrangement delegates cluster quorum responsibilities to the consensus layer, while HARP leverages it for critical role assignments and key/value storage. Neither storage nor retrieval succeeds if the consensus layer is inoperable or unreachable, thus preventing rogue Postgres nodes from accepting connections.

As a result, the consensus layer generally exists outside of HARP or HARP-managed nodes for maximum safety. Our reference diagrams show this separation, although it isn't required.

Note: To operate and manage cluster state, BDR contains its own implementation of the Raft Consensus model. You can configure HARP to leverage this same layer to reduce reliance on external dependencies and to preserve server resources. However, certain drawbacks to this approach are discussed in [Consensus layer](#).

2.5 Recommended architecture and use

HARP was primarily designed to represent a BDR Always-On architecture that resides in two or more data centers and consists of at least five BDR nodes. This configuration doesn't count any logical standby nodes.

The following diagram shows the current and standard representation:

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

This configuration results in any HARP Proxy resource in DC A connecting to BDR Node 1 and the HARP Proxy resource in DC B connecting to BDR Node 3.

Note: While this diagram shows only a single HARP Proxy per DC, this is an example only and should not be considered a single point of failure. Any number of HARP Proxy nodes can exist, and they all direct application traffic to the same node.

2.5.1 Location configuration

For multiple BDR nodes to be eligible to take the lead master lock in a location, you must define a location in the `config.yml` configuration file.

To reproduce the BDR Always-On reference architecture shown in the diagram, include these lines in the `config.yml` configuration for BDR Nodes 1 and 2:

```
location: dca
```

For BDR Nodes 3 and 4, add:

```
location: dcb
```

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

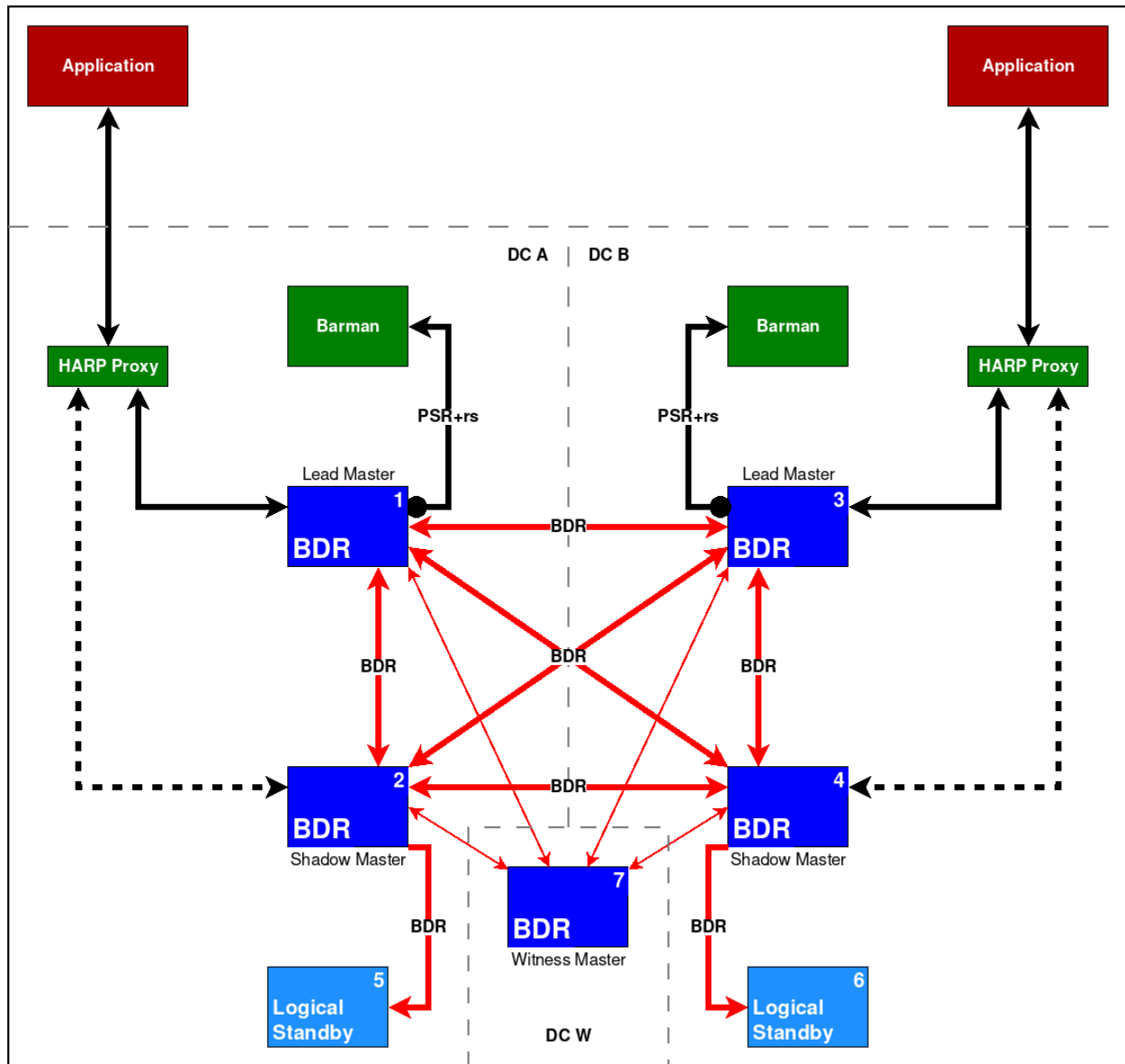


Fig. 5: BDR Always-On Reference Architecture

2.5.2 BDR 3.7 compatibility

BDR 3.7 and later offers more direct location definition by assigning a location to the BDR node. This is done by calling the following SQL API function while connected to the BDR node. So for BDR Nodes 1 and 2, you might do this:

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

And for BDR Nodes 3 and 4:

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


INSTALLATION

A standard installation of HARP includes two system services:

- HARP Manager (`harp-manager`) on the node being managed
- HARP Proxy (`harp-proxy`) elsewhere

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

3.1 Software versions

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

Software	Min version
etcd	3.4
PgBouncer	1.14

3.2 TPAExec

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

Note: TPAExec is currently available only through an EULA specifically dedicated to BDR cluster deployments. If you can't access the TPAExec URL, contact your sales or account representative.

Configure TPAexec to recognize that cluster routing is managed through HARP by ensuring the TPA `config.yml` file contains these attributes:

```
cluster_vars:
  failover_manager: harp
```

Note: Versions of TPAexec earlier than 21.1 require a slightly different approach:

```
cluster_vars:
  enable_harp: true
```

After this, install HARP by invoking the `tpaexec` commands for making cluster modifications:

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

No other modifications are necessary apart from cluster-specific considerations.

3.3 Package installation

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

3.3.1 etcd packages

Currently `etcd` packages for many popular Linux distributions aren't available by their standard public repositories. EDB has therefore packaged `etcd` for RHEL and CentOS versions 7 and 8, Debian, and variants such as Ubuntu LTS. You need access to our HARP package repository to use these libraries.

3.4 Consensus layer

HARP requires a distributed consensus layer to operate. Currently this must be either `bdr` or `etcd`. If using fewer than three BDR nodes, you might need to rely on `etcd`. Otherwise any BDR service outage reduces the consensus layer to a single node and thus prevents node consensus and disables Postgres routing.

3.4.1 etcd

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

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

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

When using TPAExec, all configured `etcd` endpoints are entered here automatically.

3.4.2 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.

To ensure quorum is possible in the cluster, always use more than two nodes so that 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:

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

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 that was simplified for readability. This file is located in the `/etc/harp` directory by default and is named `config.yml`

You can explicitly provide the configuration file location to all HARP executables by using the `-f/--config` argument.

4.1 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 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.

This means a standard preamble is always included for HARP operations, such as the following:

```
cluster:
  name: mycluster

dcs:
  ...
```

Other sections are optional or specific to the named HARP component.

4.1.1 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 and for labeling data in the DCS for the specific cluster.

HARP Manager writes information about the cluster here for consumption by HARP Proxy and `harpctl`. HARP Proxy services direct traffic to nodes in this cluster. The `harpctl` management tool interacts with this cluster.

4.1.2 DCS settings

Configuring the consensus layer is key to HARP functionality. Without the DCS, HARP has nowhere to store cluster metadata, can't hold leadership elections, and so on. Therefore this portion of the configuration is required, and certain elements are optional.

Specify all elements under a section named `dc`s with these multiple supplementary entries:

- `driver` : Required type of consensus layer to use. Currently can 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. List every node of the DCS here if possible. This ensures HARP continues to function as long as a majority of the DCS can still operate and be reached by the network.

Format when using `etcd` as the consensus layer is as follows: `edb_notranlate_1` Format when using the experimental `bdr` consensus layer is as follows: `edb_notranlate_2` Currently, `bdr` consensus layer requires the first endpoint to point to the local postgres instance.

- `request_timeout` : Time in milliseconds to consider a request as failed. If HARP makes a request to the DCS and receives no response in this time, it considers the operation as failed. This can cause the issue to be logged as an error or retried, depending on the nature of the request. Default: 250.

The following DCS SSL settings apply only when `edb_notranlate_3` yml `dc`s: `driver: etcd endpoints: - host1:2379 - host2:2379 - host3:2379` `edb_notranlate_4` yml `cluster: name: mycluster`

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

```
manager: name: node1 log_level: INFO edb_notranlate_5 yml cluster: name: mycluster
```

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

```
proxy: name: proxy1 location: dc1 pgbouncer_bin_dir: /usr/sbin edb_notranlate_6 bash harpctl set node node1  
lease_refresh_interval=100 edb_notranlate_7 bash harpctl set proxy global max_client_conn=1000 ``
```

CLUSTER BOOTSTRAPPING

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 runtime configuration either all at once or on a per-resource basis.

This process is governed through the `harpctl apply` command. For more information, see [harpctl command-line tool](#).

Set up the DCS and make sure it is functional before bootstrapping.

Important: You can combine any or all of these example into a single YAML document and apply it all at once.

5.1 Cluster-wide bootstrapping

Some settings are applied cluster-wide and you can specify them during bootstrapping. Currently this applies only to the `event_sync_interval` runtime directive, but others might be added later.

The format is as follows:

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

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

```
harpctl apply cluster.yml
```

If the cluster name isn’t already defined in the DCS, this also initializes that value.

Important: The cluster name parameter specified here always overrides 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.

5.2 Location bootstrapping

Every HARP node is associated with at most one location. This location can be a single data center, a grouped region consisting of multiple underlying servers, an Amazon availability zone, and so on. This is a logical structure that allows HARP to group nodes together such that only one represents 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 then apply it with the following:

```
harpctl apply locations.yml
```

When performing any manipulation of the cluster, include the name as a preamble so the changes are directed to the right place.

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

```
> harpctl get locations

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

Both locations are recognized by HARP and available for node and proxy assignment.

5.3 Node bootstrapping

HARP nodes exist in a named cluster and must have a designated name. Beyond this, all other settings are retained in the DCS, as they are dynamic and can affect how HARP interacts with them. To this end, bootstrap each node using one or more of the runtime directives discussed in [Location configuration](#).

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

- `name`
- `location`
- `dsn`
- `pg_data_dir`

Everything else is optional and can depend on the cluster. Because you can bootstrap multiple nodes at once, the format generally fits this structure:

```
cluster:
  name: mycluster

nodes:
```

(continues on next page)

(continued from previous page)

```
- 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 then apply it with the following:

```
harpctl apply node1.yml
```

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

```
> harpctl get nodes
```

Cluster	Name	Location	Ready	Fenced	Allow	Routing	Routing	Status	Role	Type	Lock
Duration											
-----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
mycluster	bdra1	dc1	true	false	true		ok		primary	bdr	30

5.4 Proxy bootstrapping

Unlike locations or nodes, proxies can also supply a configuration template that is applied to all proxies in 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 needed 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 isn't necessary.

Additionally, when bootstrapping the proxy template, define at least one database 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_conn: 1000
  database_name: bdrdb
  instances:
    - name: proxy1
    - name: proxy2
      default_pool_size: 50
```

This configures HARP for two proxies: `proxy1` and `proxy2` . Only `proxy2` has a custom `default_pool_size` , while using the global settings otherwise.

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

```
harpctl apply proxy.yml
```

Once the proxy template is bootstrapped, it shows 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 that interacts with the local PostgreSQL/BDR node and stores information about its state in the consensus layer. Manager determines the node that 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 must have an associated HARP Manager. Other nodes can exist, but they can't to participate as lead or shadow master roles or any other functionality that requires a HARP Manager.

Important: HARP Manager expects the be used to start and stop the database. Stopping HARP Manager will stop the database. Starting HARP Manager will start the database if it isn't already started. If another method is used to stop the database then HARP Manager will try and restart it.

6.1 How it works

Upon starting, HARP Manager uses `pg_ctl` to start Postgres if it isn't already running. After this, it periodically checks 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 that must be periodically renewed. If it expires, HARP Proxy removes 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 is `ACTIVE` except in limited cases.
- BDR Node ID for other metadata gathering.
- Other tracking values.

Important: When naming BDR nodes in HARP, the BDR node name must match the node name represented in the `node.name` configuration attribute. This occurs 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 direct switchover and failover behavior.

After updating the node metadata, HARP Manager either refreshes the lead master lease if it's already held by the local node or seeks to obtain the lease if it's expired. Since the current state of all nodes is known to all other nodes, the node that was the previous lead master is given automatic priority ranking if present. If not, all other nodes list themselves by LSN lag, node priority, and other criteria, and the most qualified node seizes the lead master lease.

This procedure happens for every defined location where nodes are present. Thus for locations DC1 and DC2, there is 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.

6.2 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
```

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

See [Location configuration](#) for further details.

6.3 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
```

There are no arguments to launch `harp-manager` as a forked daemon. This software is designed to be launched through `systemd` or in 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 Disabling and reenabling HARP Manager control of Postgres

You can temporarily pause HARP Manager control of Postgres. This results in a state where the daemon continues running but doesn't perform any operations that can affect existing behavior of the cluster. Reenabling management causes it to resume operation.

An example of temporarily disabling node management is:

```
harpctl unmanage node node1
```

See *harpctl command-line tool* for more details.

Node management by HARP Manager is enabled by default.

HARP PROXY

HARP Proxy is a daemon that 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. During a planned switchover or unplanned failover, it redirects to the new lead master node as dictated by the DCS.

You can select between `pgbouncer` or `builtin` for HARP Proxy. If you don't specify a proxy type, the default is `builtin`. When using `pgbouncer`, HARP Proxy is an interface layer between the DCS and PgBouncer. As such, PgBouncer is a prerequisite and must also be installed for HARP Proxy to fully manage its activity.

The builtin proxy doesn't require any additional software. When using `builtin`, HARP Proxy functions as a level 4 pass-through proxy.

7.1 Builtin proxy: how it works

Upon starting, HARP Proxy listens for incoming connections on the listening address and listening port specified in the bootstrap file per proxy instance. All application client traffic then passes through builtin proxy into the current lead master node for the location where this proxy is operating.

If the lead master lease isn't set, HARP Proxy disconnects all connection traffic until a new lead master is established. This also applies to circumstances when `harpcctl promote` is used to invoke a planned transition to a new lead master. The disconnect is immediate.

7.1.1 Configuration

Choose the built-in proxy by setting the proxy type to `builtin`. The only other option that applies to the built-in proxy is `max_client_conn`, which specifies the maximum allowed client connections. If `max_client_conn` is higher than what the system can handle, it is lowered to a setting that's within the capability of the system that the proxy is on.

7.2 PgBouncer: how it works

Note: If you need more configurability of `pgbouncer` than what HARP Proxy provides, the recommended setup is to use builtin proxy and have `pgbouncer` point to it.

Upon starting, HARP Proxy launches PgBouncer if it's not already running and leaves client connections paused. After, it contacts 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 then passes through PgBouncer into the current lead master node for the location where this proxy is operating.

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

If the lead master lease isn't set, HARP Proxy pauses 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're held in stasis.

7.2.1 PgBouncer configuration file

When HARP Proxy uses PgBouncer for connection management and redirection, a `pgbouncer.ini` file must exist. HARP Manager builds this file based on various runtime directives as defined in

Proxy directives .

This file is located in the same folder as the `config.yml` used by HARP Proxy. Any PgBouncer process launched by HARP Proxy uses this configuration file, and you can use it for debugging or information purposes. Modifications to this automatically generated `pgbouncer.ini` file are lost any time HARP Proxy restarts, so use `harpctl set proxy` to alter these settings instead. Calling `harpctl set proxy` doesn't update the `pgbouncer.ini` file until the proxy restarts.

7.2.2 Disabling and reenabling HARP Proxy node management

You can temporarily pause HARP Proxy control of PgBouncer. This results in a state where the daemon continues running but doesn't perform any operations that can affect existing behavior of the cluster. Reenabling management causes it to resume operation.

An example of temporarily disabling management of a specific proxy is:

```
harpctl unmanage proxy proxy1
```

See [harpctl command-line tool](#) for more details.

Proxy node management is enabled by default.

7.2.3 Passthrough user authentication

With pgbouncer, we strongly recommend configuring HARP Proxy to use the `auth_user` and `auth_query` runtime directives. If these aren't set, the PgBouncer `userlist.txt` file must include username and password hash combinations for every user PgBouncer needs to authenticate on behalf of Postgres.

Do *not* use the `pgbouncer` user, as this is used by HARP Proxy as an admin-level user to operate the underlying PgBouncer service.

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

If TPAexec isn't 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
```

(continues on next page)

(continued from previous page)

```

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>;

```

Substitute <auth_user> for the auth_user field supplied to HARP Proxy.

Then in the Bootstrap file, the following completes the configuration:

```

cluster:
  name: mycluster

proxies:
  monitor_interval: 5
  default_pool_size: 20
  max_client_conn: 1000
  auth_user: pgb_auth
  type: pgbouncer
  auth_query: "SELECT * FROM pg_catalog.pg_bouncer_get_auth($1)"
  database_name: bdrdb
  instances:
    - name: proxy1
    - name: proxy2

```

You can also 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 needs a *.pgpass* file so that *auth_user* can authenticate against Postgres.

7.2.4 Configuration

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

```

cluster:
  name: mycluster

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

```

(continues on next page)

(continued from previous page)

```
proxy:
  name: proxy1
  `` Each proxy connects to the DCS to retrieve the hosts and ports to listen on for
  ↳ connections.

### Usage

This is the basic usage for HARP Proxy:

``bash
Usage of ./harp-proxy:
  -f string
      Optional path to config file (shorthand)
  --config string
      Optional path to config file
```

There are no arguments to launch `harp-proxy` as a forked daemon. This software is designed to be launched through `systemd` or in 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.

HARPCTL COMMAND-LINE TOOL

`harpctl` is a command-line tool for directly manipulating the consensus layer contents to fit desired cluster geometry. You can use it to, for example, examine node status, “promote” a node to lead master, disable/enable cluster management, bootstrap cluster settings, and so on.

8.1 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 allowed subcommands and flags.

8.2 Configuration

harpctl must interact with the consensus layer to operate. This means a certain minimum amount of settings must 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 [Location configuration](#) for details.

8.3 Execution

Execute harpctl like this:

```
harpctl command [flags]
```

Each command has its own series of subcommands and flags. Further help for these are available by executing this command:

```
harpctl <command> --help
```

8.4 harpctl apply

You must use an `apply` command to “bootstrap” a HARP cluster using a file that defines various attributes of the intended cluster.

Execute an `apply` command 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 is integrated with the rest of the cluster.

Note: You can also use this command to bootstrap the entire cluster at once since all defined sections are applied at the same time. However, we don't encourage this use for anything but testing as it increases the difficulty of validating each portion of the cluster during initial definition.

8.5 harpctl fence

Marks the local or specified node as fenced. A node with this status is essentially completely excluded from the cluster. HARP Proxy doesn't send it traffic, its representative HARP Manager doesn't claim the lead master lease, and further steps are also taken. If running, HARP Manager stops Postgres on the node as well.

Execute a fence command like this:

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

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

8.6 harpctl get

Fetches information stored in the consensus layer for various elements of the cluster. This includes nodes, locations, the cluster, 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.
- **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.

8.6.1 harpctl get cluster

Fetches information stored in the consensus layer for the current cluster:

```
> harpctl get cluster
```

Name	Enabled
mycluster	true

8.6.2 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 attempts 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	

8.6.3 harpctl get location

Fetches location information for the specified location. If no location is passed, harpctl attempts 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>

8.6.4 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>

8.6.5 harpctl get node

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

Example:

```
> harpctl get node mynode
Cluster  Name    Location Ready Fenced Allow Routing Routing Status Role    Type Lock
↪Duration
-----
↪-----
mycluster mynode dc1      true  false true           ok           primary bdr  30
```

8.6.6 harpctl get nodes

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

Example:

```
> harpctl get nodes
Cluster  Name    Location Ready Fenced Allow Routing Routing Status Role    Type Lock
↪Duration
-----
↪-----
myclusters bdra1 dc1      true  false true           ok           primary bdr  30
myclusters bdra2 dc1      true  false false          N/A          primary bdr  30
myclusters bdra3 dc1      true  false false          N/A          primary bdr  30
```

8.6.7 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
-----
mycluster proxy1 session md5      1000      20
```

8.6.8 harpctl get proxies

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

Example:

```
> harpctl get proxies
Cluster  Name    Pool Mode Auth Type Max Client Conn Default Pool Size
-----
```

(continues on next page)

(continued from previous page)

mycluster	global	session	md5	500	25
mycluster	proxy1	session	md5	1000	20
mycluster	proxy2	session	md5	1500	30

8.7 harpctl manage

If a cluster isn't in a managed state, instructs all HARP Manager services to resume monitoring Postgres and updating the consensus layer. Do this after maintenance is complete following HARP software updates or other significant changes that might affect the whole cluster.

Execute a `manage` command like this:

```
harpctl manage cluster
```

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

8.8 harpctl promote

Promotes the next available node that meets leadership requirements to lead master in the current Location. Since this is a requested event, it invokes a smooth handover where:

1. The existing lead master releases 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_lag`.
2. The promoted node is the only valid candidate to take the lead master lease and does so as soon as it is released by the current holder. All other nodes ignore the unset lead master lease. - If CAMO is enabled, the promoted node temporarily disables client traffic until the CAMO queue is fully applied, even though it holds the lead master lease.
3. HARP Proxy, if using pgbouncer, will **PAUSE** connections to allow ongoing transactions to complete. Once the lead master lease is claimed by the promoted node, it reconfigures PgBouncer for the new connection target and resumes database traffic. If HARP Proxy is using the builtin proxy, it terminates existing connections and creates new connections to the lead master as new connections are requested from the client.

Execute a `promote` command like this:

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

Provide the `--force` option to forcibly set a node to lead master, even if it doesn't meet the criteria for becoming lead master. This circumvents any verification of CAMO status or replication lag and causes an immediate transition to the promoted node. This is the only way to specify an exact node for promotion.

The node must be online and operational for this to succeed. Use this option with care.

8.9 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 rather than using `apply`. You can use this 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.

8.9.1 harpctl set cluster

Sets cluster-related attributes only.

Example:

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

8.9.2 harpctl set node

Sets node-related attributes for the named node. Any options mentioned in Node directives are valid here.

Example:

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

8.9.3 harpctl set proxy

Sets proxy-related attributes for the named proxy. Any options mentioned in the Proxy directives are valid here. Properties set this way require a restart of the proxy before the new value takes effect.

Example:

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

Use `global` for cluster-wide proxy defaults:

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

8.10 harpctl unfence

Removes the `fenced` attribute from the local or specified node. This removes all previously applied cluster exclusions from the node so that it can again receive traffic or hold the lead master lease. Postgres is also started if it isn't running.

Execute an `unfence` command like this:

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

The node-name is optional. If you omit it, `harpctl` uses the name of the locally configured node.

8.11 harpctl unmanage

Instructs all HARP Manager services in the cluster to remain running but no longer actively monitoring Postgres, or modify the contents of the consensus layer. This means that any ordinary failover event such as a node outage doesn't result in a leadership migration. This is intended for system or HARP maintenance prior to making changes to HARP software or other significant changes to the cluster.

Execute an `unmanage` command like this:

```
harpctl unmanage cluster
```

Note: Currently you can enable or disable cluster management at only 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 information is specific to HARP's interaction with the supported DCS implementations.

9.1 BDR driver compatibility

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

For the purpose of maintaining a voting quorum, BDR Logical Standby nodes don't participate in consensus communications in a BDR cluster. Don't count these in the total node list to fulfill DCS quorum requirements.

9.2 Maintaining quorum

Clusters of any architecture require at least $n/2 + 1$ nodes to maintain consensus via a voting quorum. Thus a three-node cluster can tolerate the outage of a single node, a five-node cluster can tolerate a two-node outage, and so on. If consensus is ever lost, HARP becomes inoperable because the DCS prevents it from deterministically identifying the node that is the lead master in 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 prevents quorum in any location that can't maintain a voting majority, and thus HARP stops working.

Thus an odd-number of nodes (with a minimum of three) is crucial when building the consensus layer. 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 doesn't disrupt consensus thanks to the independently located node.

9.2.1 Multi-consensus variant

HARP assumes one lead master per configured location. Normally each location is specified in HARP using the location configuration setting. By creating a separate DCS cluster per location, you can emulate this behavior independently of HARP.

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

HARP nodes in DC-A use something like this:

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

While DC-B uses 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's no DCS communication between different data centers in this design, and thus a network partition between them doesn't affect 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 isn't 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` can't interact with nodes outside of the current location. It's impossible to obtain node information, get or set the lead master, or perform any other operation that targets the other location. Essentially this organization renders the `--location` parameter to `harpctl` unusable.

9.2.2 TPAexec and consensus

These considerations are integrated into TPAexec as well. When deploying a cluster using etcd, it constructs a separate DCS cluster per location to facilitate high availability in favor of strict consistency.

Thus this configuration example groups any DCS nodes assigned to the `first` location together, and the `second` location is a separate cluster:

```
cluster_vars:
  failover_manager: harp
  harp_consensus_protocol: etcd

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

To override this behavior, configure the `harp_location` implicitly to force a particular grouping.

Thus this example returns 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 might also be necessary to favor specific node groupings when using cloud providers such as Amazon that favor availability zones in regions over traditional data centers.

SECURITY AND ROLES

Beyond basic package installation and configuration, HARP requires Postgres permissions to operate. These allow it to gather information about Postgres or BDR as needed to maintain node status in the consensus layer.

10.1 Postgres permissions

Create the role specified in the `node.dsn` parameter in one of the following ways:

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

This syntax ensures the role can log into the database to gather diagnostic information.

Similarly, an entry must exist in `pg_hba.conf` for this role. You can do this in 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*.

10.2 BDR permissions

BDR nodes have metadata and views that are visible only 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 it is well-suited to HARP.

10.3 BDR consensus permissions

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

```
GRANT bdr_superuser TO foobar;
```

Currently access to the BDR consensus model requires superuser equivalent permission.

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