Table of Contents

MySQL for PCF

This is documentation for the MySQL service for Pivotal Cloud Foundry (PCF).

Product Snapshot

Current MySQL for Pivotal Cloud Foundry Details

- **Version**: 1.5.0
- **Release Date**: 2015-04-08
- **Software component version**: MariaDB 10.0.16, Galera 25.3.5
- **Compatible Ops Manager Version(s)**: 1.4.x, 1.5.x
- **Compatible Elastic Runtime Version(s)**: 1.3.x, 1.4.5, 1.5.x
- **vSphere support?** Yes
- **AWS support?** Yes
- **OpenStack support?** No

Upgrading to the Latest Version

Consider the following compatibility information before upgrading MySQL for Pivotal Cloud Foundry.

Note: Before you upgrade to Ops Manager 1.4.x or 1.5.x, you must first upgrade MySQL for PCF to any version of 1.3 or 1.4. This allows MySQL for PCF upgrades after you install OpsManager 1.4.x.

For more information, refer to the full Product Version Matrix.

OpsManager Version	Supported Upgrades from Imported MySQL Installation
1.3.x	• From 1.2 to 1.3 • From 1.3.2 to 1.4.0
1.4.x and 1.5.x	• From 1.3.2 to 1.4.0 • From 1.3.2 to 1.5.0 • From 1.4.0 to 1.5.0

Release Notes

Consult the Release Notes for information about changes between versions of this product.

Limitations

- Only the InnoDB storage engine is supported; it is the default storage engine for new tables. Attempted use of other storage engines (including MyISAM) may result in data loss.
- All databases are managed by shared, multi-tenant server processes. Although data is securely isolated between tenants using unique credentials, application performance may be impacted by noisy neighbors.
- Round-trip latency between database nodes must be less than five seconds; if the latency is higher than this nodes will become partitioned. If more than half of cluster nodes are partitioned the cluster will lose quorum and become unusable until manually bootstrapped.
- MariaDB Galera Cluster - Known Limitations.

Known Issues

- All proxy instances use the same method to determine cluster health. However, certain conditions may cause the proxy instances to route to different nodes, for example after brief cluster node failures.
    - This could be an issue for tables that receive highly concurrent writes. Multiple clients writing to the same table

could obtain locks on the same row, resulting in a deadlock. One commit will succeed and all others will fail and must be retried. This can be prevented by configuring your load balancer to route connections to only one proxy instance at a time.

- Once the product is deployed with operator-configured proxy IPs, the number of proxy instances can not be reduced, nor can the configured IPs be removed from the **Proxy IPs** field. If instead the product is initially deployed without proxy IPs, IPs added to the **Proxy IPs** field will only be used when adding additional proxy instances, scaling down is unpredictably permitted, and the first proxy instance can never be assigned an operator-configured IP.

Installation

This product requires Pivotal Cloud Foundry version 1.3.4 or greater.

1. Download the product file from Pivotal Network.
2. Upload the product file to your Ops Manager installation.
3. Click **Add** next to the uploaded product description in the Available Products view to add this product to your staging area.
4. Click the newly added tile to review configurable Settings.
5. Click **Apply Changes** to deploy the service.

Settings

Service Plan

A single service plan enforces quotas of 100MB of storage per database and 40 concurrent connections per user by default. Users of Operations Manager can configure these plan quotas. Changes to quotas will apply to all existing database instances as well as new instances. In calculating storage utilization, indexes are included along with raw tabular data.

The name of the plan is **100mb-dev** by default and is automatically updated if the storage quota is modified.

Provisioning a service instance from this plan creates a MySQL database on a multi-tenant server, suitable for development workloads. Binding applications to the instance creates unique credentials for each application to access the database.

Proxy

The proxy tier is responsible for routing connections from applications to healthy MariaDB cluster nodes, even in the event of node failure.

Applications are provided with a hostname or IP address to reach a database managed by the service (for more information, see About Service Binding). By default, the MySQL service will provide bound applications with the IP of the first instance in the proxy tier. Even if additional proxy instances are deployed, client connections will not be routed through them. This means the first proxy instance is a single point of failure.

In order to eliminate the first proxy instance as a single point of failure, operators must configure a load balancer to route client connections to all proxy IPs, and configure the MySQL service to give bound applications a hostname or IP address that resolves to the load balancer.

Configuring a load balancer

In older versions of the product, applications were given the IP of the single MySQL server in bind credentials. When upgrading to v1.5.0, existing applications will continue to function but in order to take advantage of high availability features they must be rebound to receive either the IP of the first proxy instance or the IP/hostname of a load balancer.

In order to configure a load balancer with the IPs of the proxy tier before v1.5.0 is deployed and prevent applications from obtaining the IP of the first proxy instance, the product enables an operator to configure the IPs that will be assigned to proxy instances. The following instructions applies to the **Proxy** settings page for the MySQL product in Operation Manager.

- In the **Proxy IPs** field, enter a list of IP addresses that should be assigned to the proxy instances. These IPs must be in the CIDR range configured in the Director tile and not be currently allocated to another VM. Look at the **Status** pages of other tiles to see what IPs are in use.
- In the **Binding Credentials Hostname** field, enter the hostname or IP that should be given to bound applications for connecting to databases managed by the service. This hostname or IP should resolve to your load balancer and be considered long-lived. When this field is modified applications must be rebound to receive updated credentials.

Configure your load balancer to route connections for a hostname or IP to the proxy IPs. As proxy instances are not synchronized, we recommend configuring your load balancer to send all traffic to one proxy instance at a time until it fails, then failover to another proxy instance. For details, see Known Issues.

Important: To configure your load balancer with a healthcheck or monitor, use TCP against port **1936**. Unauthenticated healthchecks against port 3306 will cause the service to become unavailable, and will require manual intervention to fix.

Adding a load balancer after an initial deploy

If v1.5.0 is initially deployed without a load balancer and without proxy IPs configured, a load balancer can be setup later to remove the proxy as a single point of failure. However, there are several implications to consider:

- Applications will have to be rebound to receive the hostname or IP that resolves to the load balancer. To rebind: unbind your application from the service instance, bind it again, then restage your application. For more information see Managing Service Instances with the CLI. In order to avoid unnecessary rebinding, we recommend configuring a load balancer before deploying v1.5.0.
- Instead of configuring the proxy IPs in Operations manager, use the IPs that were dynamically assigned by looking at the **Status** page. Configuration of proxy IPs after the product is deployed with dynamically assigned IPs is not well supported; see Known Issues.

Lifecycle Errands

Two lifecycle errands are run by default: the **broker registrar** and the **smoke test**. The broker registrar errand registers the broker with the Cloud Controller and makes the service plan public. The smoke test errand runs basic tests to validate that service instances can be created and deleted, and that applications pushed to Elastic Runtime can be bound and write to MySQL service instances. Both errands can be turned on or off on the **Lifecycle Errands** page under the **Settings** tab.

Resource Config

Instance Capacity

An operator can configure how many database instances can be provisioned (instance capacity) by configuring the amount of persistent disk allocated to the MySQL server nodes. The broker will provision a requested database if there is sufficient unreserved persistent disk. This can be managed using the Persistent Disk field for the MySQL Server job in the Resource Config setting page in Operations Manager. Not all persistent disk will be available for instance capacity; about 2-3 GB is reserved for service operation. Adding nodes to the cluster increases durability, not capacity. Multiple backend clusters, to increase capacity or for isolation, are not yet supported.

In determining how much persistent disk to make available for databases, operators should also consider that MariaDB servers require sufficient CPU, RAM, and IOPS to promptly respond to client requests for all databases.

Provisioning and Binding via Cloud Foundry

As part of installation the product is automatically registered with Pivotal Cloud Foundry Elastic Runtime (see Lifecycle Errands). On successful installation, the MySQL service is available to application developers in the Services Marketplace, via the web-based Developer Console or `cf marketplace`. Developers can then provision instances of the service and bind them to their applications:

```
$ cf create-service p-mysql 100mb-dev mydb
$ cf bind-service myapp mydb
$ cf restart myapp
```

For more information on use of services, see the Services Overview.

Example Application

To help application developers get started with MySQL for PCF, we have provided an example application, which can be downloaded here. Instructions can be found in the included README.

Service Dashboard

Cloud Foundry users can access a service dashboard for each database from Developer Console via SSO. The dashboard displays current storage utilization and plan quota. On the Space page in Developer Console, users with the SpaceDeveloper role will find a **Manage** link next to the instance. Clicking this link will log users into the service dashboard via SSO.

Proxy Dashboard

The service provides a dashboard where administrators can observe health and metrics for each instance in the proxy tier. Metrics include the number of clients currently connected and the number of connections made to each of the backend database nodes.

This dashboard for each proxy instance can be found at: `https://proxy-<job index>.p-mysql.<system-domain>`. Job index starts at 0; if you have two proxy instances deployed and your system-domain is `example.com`, dashboards would be accessible at `https://proxy-0.p-mysql.example.com` and `https://proxy-1.p-mysql.example.com`.

Basic auth credentials are required to access the dashboard. These can be found in the Credentials tab of the MySQL product in Operations Manager.

For more information about SwitchBoard, read the proxy documentation

See Also

- Notes on cluster configuration
- Backing Up MySQL for PCF
  Note: For information about backing up your PCF installation, refer to the Backing Up Pivotal Cloud Foundry topic.
- Determining MySQL cluster state
- More on Cluster Scaling, Node Failure, and Quorum
- Bootstrapping an ailing MySQL cluster

Release Notes

1.5.0

- **AWS support:** The clustered database service can now be deployed on Amazon Web Services from the Operations Manager Web UI.
    - Deployment is limited to a single Availability Zone. Look for multi-AZ in future releases.
    - Single availability zone is a limitation on AWS. Operations Manager on vSphere continues to support deployment to multiple availability zones.
    - The default instance type for the cluster nodes on AWS is m3.large.
    - All jobs are deployed with SSD for ephemeral and persistent disk.
- **IaaS agnostic**
    - The same product can be deployed to both AWS and vSphere
    - Precompiled packages are no longer included
    - p-mysql 1.5.0 requires Operations Manager 1.4.0
- **New proxy tier**
    - Improved availability: We have entirely re-written the proxy to eliminate situations where clients could hang when a cluster node was unhealthy.
    - A dashboard that clearly displays node health in real time
- **Upgrade support:** This product can be automatically upgraded from version 1.3.2 or 1.4.0
- **Cluster node resources increased for vSphere**: The default resources are now 4GB RAM, 2 CPU, 10GB persistent disk
- **Faster compilation:** Default resource for the compilation jobs on vSphere are now 4GB RAM, 4 CPU, 20GB persistent disk
- **Bug fix:** Fix broker-deregistrar errand to succeed even when MySQL service is broken
- **Bug fix:** Quota enforcer could fail when broker hasn't finished initializing

Known issues:

- On AWS, this version supports deployments in the US-East region. Multi-region support is coming in a future release.
- The experimental HTTPS-only feature in Elastic Runtime 1.5 may cause issues with this version of the product. Full support for HTTPS-only traffic is coming in a future release.
- Note: BOSH Stemcell 2865.1 is required for installation on Ops Manager 1.5.x and above.

1.4.0

- **High Availability**: database server is now clustered and synchronously replicated using MariaDB Galera Cluster. A copy of each database resides on all cluster nodes, and writes to any database are replicated to all copies. All client connections are routed to a primary cluster node, and in the event of a node failure the proxy tier manages failover, routing client connections to a healthy cluster node. MySQL server, proxy, and broker jobs can all be scaled out horizontally for increased availability, eliminating single points of failure.
- **Improved logging and monitoring**: route-registration on the broker is now an independent process
- **Bug fix**: calculation of storage utilization for the purposes of quota enforcement when multiple apps are bound
- **Bug fix**: format of jdbcUrl connection string (found in VCAP_SERVICES on bind)

Notes on High Availability

- When upgrading from an older version, applications must be rebound to take advantage of high availability features. To rebind: unbind your application from the service instance, bind it again, then restage your application. For more information see Managing Service Instances with the CLI.
- Elimination of the proxy as a single point of failure requires configuration of an external load balancer to route connections to proxy instances. For details, see Proxy Settings.
- See Known Issues.

1.3.2

- **Updated stemcell addresses bash-shellshock vulnerabilities** : resolves CVEs discussed here and here.

1.3.0

- **Syslog forwarding**: Syslogs are now streamed to the same host and port configured in Elastic Runtime settings
- **Dynamic instance capacity management** : Previously operators had to manually configure the maximum number of service instances permitted by the server. This required manual calculation and a knowledge of required system headroom. Admins can now manage instance capacity simply by adjusting persistent disk allocated to mysql nodes. Remaining instance capacity is determined dynamically by subtracting a safe estimate for system headroom and reserved storage for provisioned instances.
- **Trusty stemcell**: Server and broker are now deployed on Ubuntu "Trusty" 14.04 LTS stemcells, providing improved security, performance, and a smaller resource footprint.
- **Least necessary privileges**: The MySQL service dashboard uses a new, limited permission OAuth scope to determine whether a user currently has access to a service instance. The dashboard no longer has full read access to a user's account.
- **Precompiled packages**: Most packages have been precompiled for the targeted stemcell. This will lower initial deployment times, at the cost of a larger download.

1.2.0

- Product renamed to 'MySQL for Pivotal CF'
- Plan attributes are configurable: max storage per database, max concurrent connections per user, and max databases
- Plan name is determined dynamically based on configured storage quota
- Plan features include disclaimer that the service is not for production use
- Developers can SSO to a service dashboard that displays storage utilization
- Security fixes including updates to Rails
- Service broker is registered by URL (rather than by IP). Typically has the format `https://p-mysql.<cf-domain>` .
- Lifecycle errands are used to register the broker and run tests that verify the deployment.
- Improved logging in service broker

The following components will be re-deployed:

- cf-mysql-broker
- mysql

New components:

- broker-registrar
- broker-deregistrar
- acceptance-tests

1.1.0

- Updated the format of metadata fields in the broker catalog endpoint and added additional fields. For more information, see Catalog Metadata.
- Updated Ruby to version 2.0.0p353 to fix a vulnerability in 1.9.3p448.
- Requests to delete a service instance or binding now get a 200 response with an empty JSON body instead of a 204.
- The broker now returns a clear error when there is no more capacity for additional instances during a provision request. The response has status code `507` . The user-facing error message is "Service plan capacity has been reached."

The following components will be re-deployed:

- cf-mysql-broker
- mysql

Cluster Scaling, Node Failure, and Quorum

Documented here are scenarios in which the size of a cluster may change, how the cluster behaves, and how to restore service function when impacted. Galera Cluster is used to manage the MariaDB cluster in our release.

Healthy Cluster

Galera documentation refers to nodes in a healthy cluster as being part of a primary component. These nodes will respond normally to all queries, reads, writes and database modifications.

If an individual node is unable to connect to the rest of the cluster (ex: network partition) it becomes non-primary (stops accepting writes and database modifications). In this case, the rest of the cluster should continue to function normally. A non-primary node may eventually regain connectivity and rejoin the primary component.

If more than half of the nodes in a cluster are no longer able to connect to each other, all of the remaining nodes lose quorum and become non-primary. In this case, the cluster must be manually restarted, as documented in the bootstrapping docs.

Graceful removal of a node

- Shutting down a node with monit (or decreasing cluster size by one) will cause the node to gracefully leave the cluster.
- Cluster size is reduced by one and maintains healthy state. Cluster will continue to operate, even with a single node, as long as other nodes left gracefully.

Adding new nodes

When new nodes are added to or removed from a MySQL service, a top-level property is updated with the new nodes' IP addresses. As BOSH deploys, it will update the configuration and restart all of the mysql nodes **and** the proxy nodes (to inform them of the new IP addresses as well). Restarting the nodes will cause all connections to that node to be dropped while the node restarts.

Scaling the cluster

Scaling up from 1 to N nodes

When a new MariaDb node comes online, it replicates data from the existing node in the cluster. Once replication is complete, the node will join the cluster. The proxy will continue to route all incoming connections to the primary node while it remains healthy.

If the proxy detects that this node becomes unhealthy, it will sever existing connections, and route all new connections to a different, healthy node. If there are no healthy MariaDb nodes, the proxy will reject all subsequent connections.

While transitioning from one node to a cluster, there will be an undetermined period of performance degradation while the new node sync's all data from the original node.

Note: If you are planning to scale up MariaDb nodes, it is recommended to do so in different Availability Zones to maximize cluster availability. An Availability Zone is a network-distinct section of a given Region. Further details are available in Amazon's documentation.

Scaling down from N to 1 node

When scaling from multiple nodes to a single MariaDb node, the proxy will determine that the sole remaining node is the primary node (provided it remains healthy). The proxy routes incoming connections to the remaining MariaDb node.

Rejoining the cluster (existing nodes)

Existing nodes restarted with monit should automatically join the cluster. If an existing node fails to join the cluster, it may be because its transaction record's (`seqno`) is higher than that of the nodes in the cluster with quorum (aka the primary component).

- If the node has a higher `seqno` it will be apparent in the error log `/var/vcap/sys/log/mysql/mysql.err.log`.

- If the healthy nodes of a cluster have a lower transaction record number than the failing node, it might be desirable to shut down the healthy nodes and bootstrap from the node with the more recent transaction record number. See the bootstraping docs for more details.
- Manual recovery may be possible, but is error-prone and involves dumping transactions and applying them to the running cluster (out of scope for this doc).
- Abandoning the data is also an option, if you're ok with losing the unsynced transactions. Follow the following steps to abandon the data (as root):
  - Stop the process with `monit stop mariadb_ctrl` .
  - Delete the galera state (`/var/vcap/store/mysql/grastate.dat`) and cache (`/var/vcap/store/mysql/galera.cache`) files from the persistent disk.
  - Restarting the node with `monit start mariadb_ctrl` .

Quorum

- In order for the cluster to continue accepting requests, a quorum must be reached by peer-to-peer communication. More than half of the nodes must be responsive to each other to maintain a quorum.
- If more than half of the nodes are unresponsive for a period of time the nodes will stop responding to queries, the cluster will fail, and bootstrapping will be required to re-enable functionality.

Avoid an even number of nodes

- It is generally recommended to avoid an even number of nodes. This is because a partition could cause the entire cluster to lose quorum, as neither remaining component has more than half of the total nodes.
- A 2 node cluster cannot tolerate the failure of single node failure as this would cause loss of quorum. As such, the minimum number of nodes required to tolerate single node failure is 3.

Unresponsive node(s)

- A node can become unresponsive for a number of reasons:
  - network latency
  - mysql process failure
  - firewall rule changes
  - vm failure
- Unresponsive nodes will stop responding to queries and, after timeout, leave the cluster.
- Nodes will be marked as unresponsive (innactive) either:
  - If they fail to respond to one node within 15 seconds
  - OR If they fail to respond to all other nodes within 5 seconds
- Unresponsive nodes that become responsive again will rejoin the cluster, as long as they are on the same IP which is pre-configured in the gcomm address on all the other running nodes, and a quorum was held by the remaining nodes.
- All nodes suspend writes once they notice something is wrong with the cluster (write requests hang). After a timeout period of 5 seconds, requests to non-quorum nodes will fail. Most clients return the error: `WSREP has not yet prepared this node for application use` . Some clients may instead return `unknown error` . Nodes who have reached quorum will continue fulfilling write requests.
- If deployed using a proxy, a continually inactive node will cause the proxy to fail over, selecting a different mysql node to route new queries to.

Re-bootstrapping the cluster after quorum is lost

- The start script will currently bootstrap node 0 only on initial deploy. If bootstrapping is necessary at a later date, it must be done manually. For more information on manually bootstrapping a cluster, see Bootstrapping Galera.
- If the single node is bootstrapped, it will create a new one-node cluster that other nodes can join.

Simulating node failure

- To simulate a temporary single node failure, use `kill -9` on the pid of the mysql process. This will only temporarily disable the node because the process is being monitored by monit, which will restart the process if it is not running.

- To more permenantly disable the process, execute `monit unmonitor mariadb_ctrl` before `kill -9`.
- To simulate multi-node failure without killing a node process, communication can be severed by changing the iptables config to dissallow communication:

  ```
  iptables -F && # optional - flush existing rules \
  iptables -A INPUT -p tcp --destination-port 4567 -j DROP && \
  iptables -A INPUT -p tcp --destination-port 4568 -j DROP && \
  iptables -A INPUT -p tcp --destination-port 4444 -j DROP && \
  iptables -A INPUT -p tcp --destination-port 3306 && \
  iptables -A OUTPUT -p tcp --destination-port 4567 -j DROP && \
  iptables -A OUTPUT -p tcp --destination-port 4568 -j DROP && \
  iptables -A OUTPUT -p tcp --destination-port 4444 -j DROP && \
  iptables -A OUTPUT -p tcp --destination-port 3306
  ```

  To recover from this, drop the partition by flushing all rules: `iptables -F`

Cluster Configuration

This page documents the various configuration decisions that have been made in relation to MariaDB and Galera in cf-mysql-release.

SST method

Galera supports multiple methods for State Snapshot Transfer. The `rsync` method is usually fastest. The `xtrabackup` method has the advantage of keeping the donor node writeable during SST. We have chosen to use `xtrabackup`.

InnoDB Log Files

Our cluster defaults to 1GB for log file size to support larger blob.

Max User Connections

To ensure all users get fair access to system resources, we have capped each user's number of connections to 40.

Skip External Locking

Since each Virtual Machine only has one mysqld process running, we do not need external locking.

Max Allowed Packet

We allow blobs up to 256MB. This size is unlikely to limit a user's query, but is also manageable for our InnoDB log file size. Consider using our Riak CS service for storing large files.

Innodb File Per Table

Innodb allows using either a single file to represent all data, or a separate file for each table. We chose to use a separate file for each table as this provides more flexibility and optimization. For a full list of pros and cons, see MySQL's documentation for InnoDB File-Per-Table Mode.

Innodb File Format

To take advantage of all the extra features available with the `innodb_file_per_table = ON` option, we use the `Barracuda` file format.

Temporary Tables

MySQL is configured to convert temporary in-memory tables to temporary on-disk tables when a query EITHER generates more than 16 million rows of output or uses more than 32MB of data space. Users can see if a query is using a temporary table by using the EXPLAIN command and looking for "Using temporary," in the output. If the server processes very large queries that use /tmp space simultaneously, it is possible for queries to receive no space left errors.

Proxy for MySQL for Pivotal Cloud Foundry

In cf-mysql-release, Switchboard is used to proxy TCP connections to healthy MariaDB nodes.

A proxy is used to gracefully handle failure of MariaDB nodes. Use of a proxy permits very fast, unambiguous failover to other nodes within the cluster in the event of a node failure.

When a node becomes unhealthy, the proxy re-routes all subsequent connections to a healthy node. All existing connections to the unhealthy node are closed.

Consistent Routing

At any given time, Switchboard will only route to one active node. That node will continue to be the only active node until it becomes unhealthy.

If multiple Switchboard proxies are used in parallel (ex: behind a load-balancer) there is no guarantee that the proxies will choose the same active node. This can result in deadlocks, wherein attempts to update the same row by multiple clients will result one commit succeeding and the other fails. This is a known issue, with exploration of mitigation options on the roadmap for this product. To avoid this problem, use a single proxy instance or an external failover system to direct traffic to one proxy instance at a time.

Node Health

Healthy

The proxy queries an HTTP healthcheck process, co-located on the database node, when determining where to route traffic.

If the healthcheck process returns HTTP status code of 200, the node is added to the pool of healthy nodes.

A resurrected node will not immediately receive connections. The proxy will continue to route all connections, new or existing, to the currently active node. In the case of failover, all healthy nodes will be considered as candidates for new connections.

Unhealthy

If the healthcheck returns HTTP status code 503, the node is considered unhealthy.

This happens when a node becomes non-primary, as specified by the cluster-behavior docs.

The proxy will sever all existing connections to newly unhealthy nodes. Clients are expected to handle reconnecting on connection failure. The proxy will route new connections to a healthy node, assuming such a node exists.

Unresponsive

If node health cannot be determined due to an unreachable or unresponsive healthcheck endpoint, the proxy will consider the node unhealthy. This may happen if there is a network partition or if the VM containing the healthcheck and MariaDB node died.

State Snapshot Transfer (SST)

When a new node is added to the cluster or rejoins the cluster, it synchronizes state with the primary component via a process called SST. A single node from the primary component is chosen to act as a state donor. By default Galera uses rsync to perform SST, which blocks for the duration of the transfer. However, cf-mysql-release is configured to use Xtrabackup, which allows the donor node to continue to accept reads and writes.

Proxy count

If the operator sets the total number of proxies to 0 hosts in their manifest, then applications will start routing connections directly to one healthy MariaDB node making that node a single point of failure for the cluster.

The recommended number of proxies are 2; this provides redundancy should one of the proxies fail.

Removing the proxy as a SPOF

The proxy tier is responsible for routing connections from applications to healthy MariaDB cluster nodes, even in the event of node failure.

Bound applications are provided with a hostname or IP address to reach a database managed by the service. By default, the MySQL service will provide bound applications with the IP of the first instance in the proxy tier. Even if additional proxy instances are deployed, client connections will not be routed through them. This means the first proxy instance is a single point of failure.

In order to eliminate the first proxy instance as a single point of failure, operators must configure a load balancer to route client connections to all proxy IPs, and configure the MySQL service to give bound applications a hostname or IP address that resolves to the load balancer.

Configuring load balancer

Configure the load balancer to route traffic for TCP port 3306 to the IPs of all proxy instances on TCP port 3306. Next, configure the load balancer’s healthcheck to use the proxy health port. This is TCP port 1936 by default to maintain backwards compatibility with previous releases, but this port can be configured by changing the following manifest property:

```
jobs:
- name: proxy_z1
  properties:
    proxy:
      health_port: <port>
```

Configuring cf-mysql-release to give applications the address of the load balancer

To ensure that bound applications will use the load balancer to reach bound databases, the manifest property `properties.mysql_node.host` must be updated for the cf-mysql-broker job:

```
jobs:
- name: cf-mysql-broker_z1
  properties:
    mysql_node:
      host: <load balancer address>
```

AWS Route 53

To set up a Round Robin DNS across multiple proxy IPs using AWS Route 53, follow the following instructions:

1. Log in to AWS.
2. Click Route 53.
3. Click Hosted Zones.
4. Select the hosted zone that contains the domain name to apply round robin routing to.
5. Click ‘Go to Record Sets’.
6. Select the record set containing the desired domain name.
7. In the value input, enter the IP addresses of each proxy VM, separated by a newline.

Finally, update the manifest property `properties.mysql_node.host` for the cf-mysql-broker job, as described above.

API

The proxy hosts a JSON API at `proxy-<bosh job index>.p-mysql.<system domain>/v0/`.

The API provides the following route:

Request: * Method: GET * Path: `/v0/backends` * Params: ~ * Headers: Basic Auth

Response:

```
[
  {
    "name": "mysql-0",
    "ip": "1.2.3.4",
    "healthy": true,
    "active": true,
    "currentSessionCount": 2
  },
  {
    "name": "mysql-1",
    "ip": "5.6.7.8",
    "healthy": false,
    "active": false,
    "currentSessionCount": 0
  },
  {
    "name": "mysql-2",
    "ip": "9.9.9.9",
    "healthy": true,
    "active": false,
    "currentSessionCount": 0
  }
]
```

Dashboard

The proxy also provides a Dashboard UI to view the current status of the database nodes. This is hosted at `proxy-<bosh job index>.p-mysql.<system domain>`.

Determining Cluster State

Connect to each MySQL node using a mysql client and check its status.

```
$ mysql -h NODE_IP -u root -pPASSWORD -e 'SHOW STATUS LIKE "wsrep_cluster_status";'
+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
```

If all nodes are in the `Primary` component, you have a healthy cluster. If some nodes are in a `Non-primary` component, those nodes are not able to join the cluster.

See how many nodes are in the cluster.

```
$ mysql -h NODE_IP -u root -pPASSWORD -e 'SHOW STATUS LIKE "wsrep_cluster_size";'
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
```

If the value of `wsrep_cluster_size` is equal to the expected number of nodes, then all nodes have joined the cluster. Otherwise, check network connectivity between nodes and use `monit status` to identify any issues preventing nodes from starting.

For more information, see the official Galera documentation for Checking Cluster Integrity.

Bootstrapping a Galera Cluster

Bootstrapping is the process of (re)starting a Galera cluster. Before evaluating whether manual bootstrapping is necessary, ensure the nodes are able to communicate with each other i.e. there are no network partitions. Once network partitions have been resolved, reevaluate the cluster state.

When to Bootstrap

Manual bootstrapping should only be required when the cluster has lost quorum.

Quorum is lost when less than half of the nodes can communicate with each other (for longer than the configured grace period).

If quorum has *not* been lost, then individual unhealthy nodes should automatically rejoin the quorum once repaired (error resolved, node restarted, or connectivity restored).

Note: The cluster is automatically bootstrapped the first time the cluster is deployed.

Symptoms of Lost Quorum

- All nodes appear "Unhealthy" on the proxy dashboard.
- All responsive nodes report the value of `wsrep_cluster_status` as `non-Primary`.

```
mysql> SHOW STATUS LIKE 'wsrep_cluster_status';
+----------------------+-------------+
| Variable_name        | Value       |
+----------------------+-------------+
| wsrep_cluster_status | non-Primary |
+----------------------+-------------+
```

- All responsive nodes respond with `ERROR 1047` when queried with most statement types.

```
mysql> select * from mysql.user;
ERROR 1047 (08S01) at line 1: WSREP has not yet prepared node for application use
```

See Cluster Behavior for more details about determining cluster state.

Bootstrapping

Once it has been determined that bootstrapping is required, follow the following steps to shut down the cluster and bootstrap from the nodes with the most transactions.

1. SSH to each node in the cluster and, as root, shut down the mariadb process.

```
  $ monit stop mariadb_ctrl
```

Re-bootstrapping the cluster will not be successful unless all other nodes have been shut down.

1. Choose a node to bootstrap.
   Find the node with the highest transaction sequence number (seqno):

~~~
- If a node shutdown gracefully, the seqno should be in the galera state file.

    ```sh
    $ cat /var/vcap/store/mysql/grastate.dat | grep 'seqno:'
    ```

- If the node crashed or was killed, the seqno in the galera state file should be `-1`. In this case, the seqno may be recoverable from the

    ```sh
    $ /var/vcap/packages/mariadb/bin/mysqld --wsrep-recover
    ```

    Scan the error log for the recovered sequence number (the last number after the group id (uuid) is the recovered seqno):

    ```sh
    $ grep "Recovered position" /var/vcap/sys/log/mysql/mysql.err.log | tail -1
    150225 18:09:42 mysqld_safe WSREP: Recovered position e93955c7-b797-11e4-9faa-9a6f0b73eb46:15
    ```

    Note: The galera state file will still say `seqno: -1` afterward.

- If the node never connected to the cluster before crashing, it may not even have a group id (uuid in grastate.dat). In this case there's

Use the node with the highest `seqno` value as the new bootstrap node. If all nodes have the same `seqno`, you can choose any node as the r
~~~

Important: Only perform these bootstrap commands on the node with the highest `seqno`. Otherwise the node with the highest `seqno` will be unable to join the new cluster (unless its data is abandoned). Its mariadb process will exit with an error. See cluster behavior for more details on intentionally abandoning data.

1. On the new bootstrap node, update state file and restart the mariadb process:

```
$ echo -n "NEEDS_BOOTSTRAP" > /var/vcap/store/mysql/state.txt
$ monit start mariadb_ctrl
```

You can check that the mariadb process has started successfully by running:

```
$ watch monit summary
```

It can take up to 10 minutes for monit to start the mariadb process.

1. Once the bootstrapped node is running, start the mariadb process on the remaining nodes via monit.

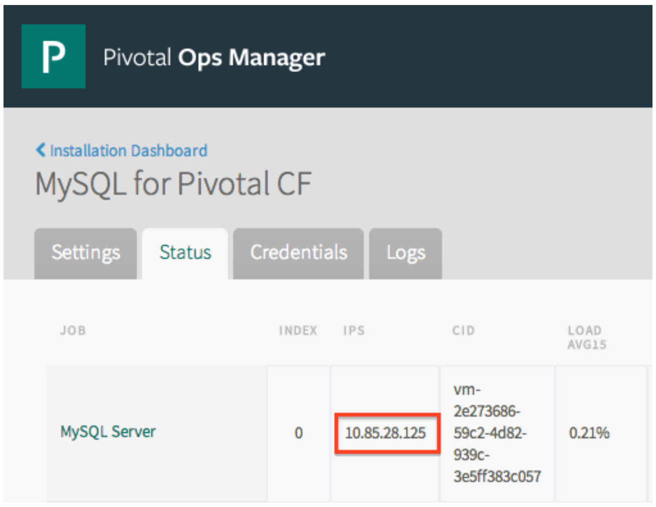
```
$ monit start mariadb_ctrl
```

1. Verify that the new nodes have successfully joined the cluster. The following command should output the total number of nodes in the cluster:

```
mysql> SHOW STATUS LIKE 'wsrep_cluster_size';
```

Back Up MySQL for Pivotal Cloud Foundry

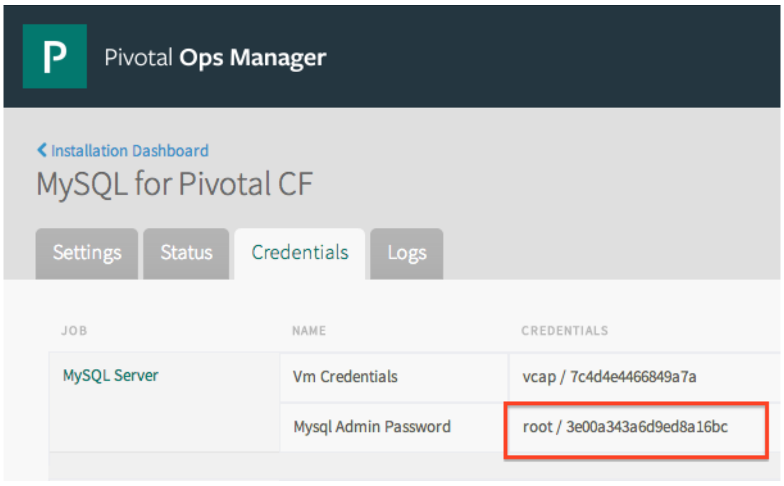
- Locate the IP address for the MySQL node in the Status tab.



MySQL for Pivotal CF

JOB	INDEX	IPS	CID	LOAD AVG15
MySQL Server	0	10.85.28.125	vm-2e273686-59c2-4d82-939c-3e5ff383c057	0.21%

- Locate the root password for the MySQL server in the Credentials tab.



MySQL for Pivotal CF

JOB	NAME	CREDENTIALS
MySQL Server	Vm Credentials	vcap / 7c4d4e4466849a7a
	Mysql Admin Password	root / 3e00a343a6d9ed8a16bc

Backup

Manual backup can be performed with mysqldump. This backup acquires a global read lock on all tables, but does not hold it for the entire duration fo the dump.

- To backup ALL databases in the MySQL deployment, use `--all-databases`:

```
$ mysqldump -u root -p -h $MYSQL_NODE_IP --all-databases > user_databases.sql
```

- To backup a single database, specify the database name:

```
$ mysqldump -u root -p -h $MHQL_NODE_IP $DB_NAME > user_databases.sql
```

Restore

Restoring from a backup is the same whether one or multiple databases were backed up. Executing the SQL dump will drop, recreate and refill the specified databases and tables.

- Restore from the data dump:

```
$ mysql -u root -p -h $MYSQL_NODE_IP < user_databases.sql
```

Examples

More examples can be found in the MariaDB documentation.