severalnines

PostgreSQL Management and Automation with ClusterControl

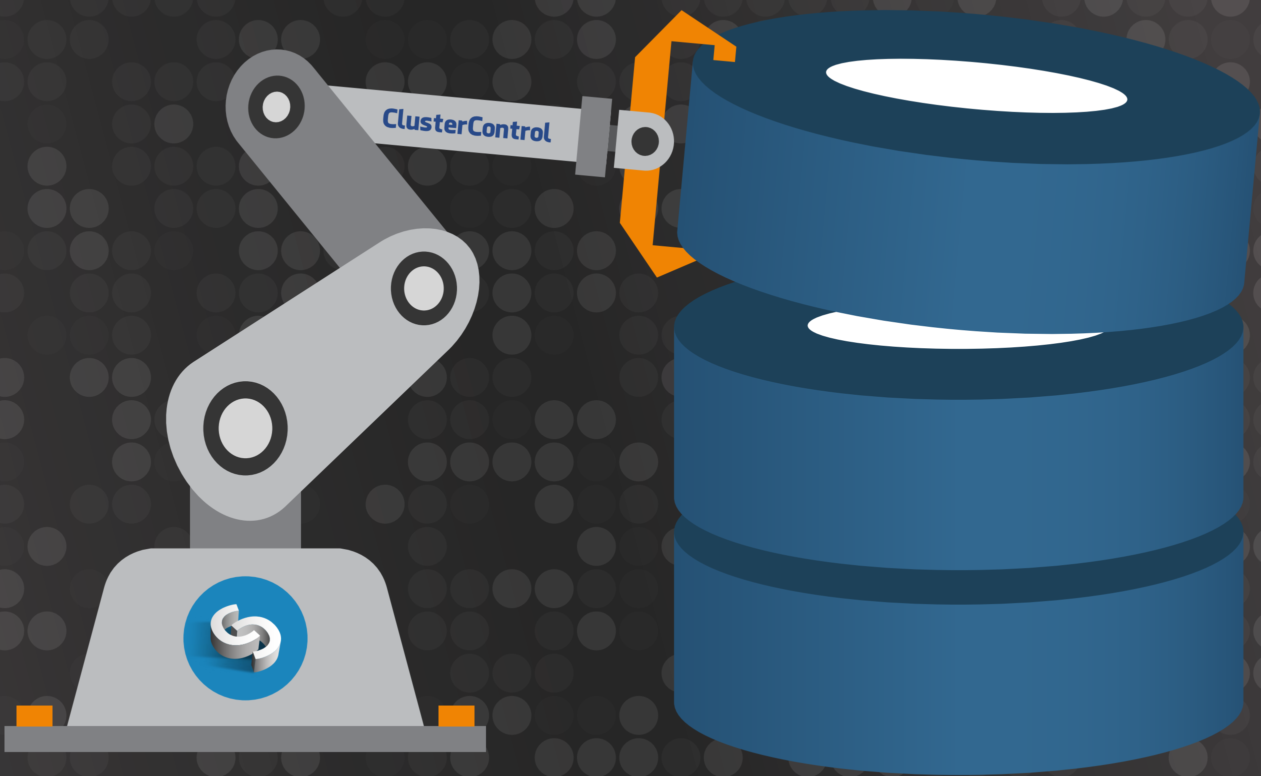

Table of Contents

Introduction

PostgreSQL is an object-relational database management system (ORDBMS) developed at the University of California at Berkeley Computer Science Department.

It supports a large part of the SQL standard and offers many modern features:

- complex queries
- foreign keys
- triggers
- updateable views
- transactional integrity
- multiversion concurrency control

Also, PostgreSQL can be extended by the user in many ways, for example by adding new:

- data types
- functions
- operators
- aggregate functions
- index methods
- procedural languages

And because of the liberal license, PostgreSQL can be used, modified, and distributed free of charge by anyone for any purpose, be it private, commercial, or academic.

These features have consolidated the engine in the top 4 of the most used databases.

Rank			DBMS
Apr 2018	Mar 2018	Apr 2017	
1.	1.	1.	Oracle
2.	2.	2.	MySQL
3.	3.	3.	Microsoft SQL Server
4.	4.	4.	PostgreSQL

Figure 1: PostgreSQL Rank

PostgreSQL offers natively some of the most industry demanded feature , such as master-slave replication, backup and recovery, transactionality, partitioning.

Anyway, there are still some other demanded features that need to be accomplished by external tools, such as sharding, master-master setups, monitoring, load balancing and statistics reporting.

Backup and recovery

PostgreSQL supports logical and physical backup.

The logical backup, flat text files with the SQL statements necessary to recreate the base (or part of it) in another environment, can be used for recovery in a different operating system or architecture, as well as different engine versions. The files are small, as they do not keep index data, and can be easily compressed.

The physical (binary) backup is basically a copy of the datafiles. The files are not compatible between operating systems, architecture or engine version, but they can be faster to recover.

They also keep a flag of the last executed transaction, so the backup knows which transactions need to be applied during a recovery.

For logical backups we have the pg_dump and pg_dumpall utilities, that allows us to backup one, several or all the databases in our installation. pg_dumpall will also include the users and tablespaces.

For pg_dump we have several flags to enable the extraction of the schema only, only the data, particular tables or objects, as well as enable different format and compression options.

A simple example of dumping and loading a database can be, to dump a database called mydb into a SQL-script file:

```
$ pg_dump mydb > db.sql
```

To reload such a script into a (freshly created) database named newdb:

```
$ psql -d newdb -f db.sql
```

One more advanced example can be something like, to dump all schemas whose names start with east or west and end in gsm, excluding any schemas whose names contain the word test:

```
$ pg_dump -n ‘east*gsm’ -n ‘west*gsm’ -N ‘*test*’ mydb >
db.sql
```

For physical backups of a running PostgreSQL database cluster, we can use pg_basebackup. These are taken without affecting other client connections to the database, and can be used both for point-in-time recovery and as the starting point for a log shipping or streaming replication standby servers.

To create a base backup of the server at mydbserver and store it in the local directory /usr/local/pgsql/data:

```
$ pg_basebackup -h mydbserver -D /usr/local/pgsql/data
```

For more details on the PostgreSQL backup facilities please check:

- Blog: Become a PostgreSQL DBA - Logical & Physical PostgreSQL Backups
- Blog: Top Backup Tools for PostgreSQL

Point in Time Recovery implies the ability to recover a database up to a given point in time. It can be useful to recover from a logical error like delete data or drop a table, or it can be used for example with auditing purposes, to check the state of a database at a given point.

For being able to perform a PITR, we need a backup previous to the point until we want to recover, and all the logs from the time the backup ended until that given point. The process will apply all the modifications made until that specified point, and then rollback the uncommitted ones.

For more details on PITR and how to perform it please check: Blog: Become a PostgreSQL DBA: Point-in-Time Database Restoration.

HA setups

Database servers can work together to allow a second server to take over quickly if the primary server fails (high availability), or to allow several computers to serve the same data (load balancing).

For HA configuration we can have several architectures, but the basic ones would be master-slave and master-master architectures.

3.1. Master-Slave architectures

These architectures enable us to maintain an master database with one or more standby servers ready to take over operations if the primary server fails. These standby databases will remain synchronized (or almost synchronized) with the master.

The replication between the master and the slaves can be made via SQL statements (logical standbys) or via the internal data structure modifications (physical standbys).

PostgreSQL uses a stream of write-ahead log (WAL) records to keep the standby databases synchronized. If the main server fails, the standby contains almost all of the data of the main server, and can be quickly made the new master database server. This can be synchronous or asynchronous and can only be done for the entire database server.

Setting up streaming replication is a task that requires some steps to be followed thoroughly. For those steps and some more background on this subject, please see: Become a PostgreSQL DBA - How to Setup Streaming Replication for High Availability.

From version 10, postgresql includes the option to setup logical replication.

Logical replication allows a database server to send a stream of data modifications to another server. PostgreSQL logical replication constructs a stream of logical data modifications from the WAL. Logical replication allows the data changes from individual tables to be replicated. It doesn't require a particular server to be designated as a master or a replica but allows data to flow in multiple directions.

You can find more information regarding logical replication: Blog: An Overview of Logical Replication in PostgreSQL.

To effectively ensure high availability, it is not enough to have a master-slave architecture. We also need to enable some automatic form of failover, so if something fails we can have the smallest possible delay in resuming normal functionality.

PostgreSQL does not include an automatic failover mechanism to identify failures on the master database and notify the salve to take ownership, so that will require a little bit of work on the DBA's side. You should work on a script that includes the pg_ctl promote command, that will promote the slave as a new master. There are also some third party tools for this automation. Many such tools exist and are well integrated with the operating system facilities required for successful failover, such as IP address migration.

After a failover happens, you need to modify your application accordingly to work with the new master. You will also have only one server working, so re-creation of the master-slave architecture needs to be done, so we get back to the same normal situation that we had before the issue.

3.2. Master-Master architectures

This architecture provides a way of minimizing the impact of an error in one of the nodes, as the other node can take care of all the traffic, maybe slightly affecting the performance, but never losing functionality. It is also used to accomplish (and maybe this is even a more interesting point) horizontal scalability (scale-out), opposite to the concept of vertical scalability where we add more resources to a server (scale-up).

For implementing this architecture, you will need to use external tools, as this feature is not (yet) natively supported by PostgreSQL.

You must be very careful when choosing a solution for implementing master-master, as there are many different products. A lot of them are still "green" , with few serious users or success cases. Some other projects have, on the other hand, been abandoned, as there are no active maintainers.

For more information on the available tools please refer to: Blog: Top PG Clustering HA Solutions for PostgreSQL.

3.2.1. Load Balancing and connection pooling

There are several load balancer tools that can be used to manage the traffic from your application to get the most of your database architecture. In the same way, there are some others that can help you manage the way the application connects to the database, by pooling these connections and reusing them between different requests.

There are some products that are used for both purposes, like the well known pgpool, and some others that will focus in only one of these features, like pgbouncer (connection pooling) and HAProxy (used for load balancing).

Monitoring

When working with database systems, you should be able to monitor them. That will enable you to identify trends, plan for upgrades or improvements or react effectively to any problems or errors that may arise.

This activity actually involves several steps, like gathering metrics, analyzing, computing statistics and generating summaries and graphs regarding the performance or the capacity of a system, as well as generating alerts in case of unexpected problems or failures which require immediate attention or action.

There are several things to monitor, like database statistics that live in the metadata tables or operating system metrics (you can check some of the most importants metrics here). There are also a number of notification and alerting tools that can react on events. This makes the monitoring and alerting setup a complex and time consuming task, as you will have to play with a lot of information and external tools. You have to manage this carefully, and find that balance where you get the necessary information to keep your system under control, but avoid getting overloaded by alarms and notifications.

Synopsis

We tried to briefly describe some of the challenges that you may face when managing PostgreSQL. Setting an HA environment, ensuring a disaster recovery strategy, managing and optimizing the load of your database and effectively monitoring your system are not out the box tasks. For a well managed system, you need to investigate and experiment with the procedures as they require a certain level of knowledge to be implemented in a stable manner.

We will now look into ClusterControl, that manages a good deal of these tasks and can help accomplish them from a unified interface.

Automation with ClusterControl

ClusterControl provides automation for most of the PostgreSQL tasks described above, in a centralized and user-friendly way. With this system you will be able to easily configure things that, manually, will take time and effort. We will now review some of its main features.

6.1. Deployment

ClusterControl itself can be installed on a dedicated VM or host using an installer script. It is an agentless management and monitoring system, and requires SSH access to the database hosts.

Once we enter the ClusterControl interface, the first thing to do is deploy a new cluster or import an existing one.

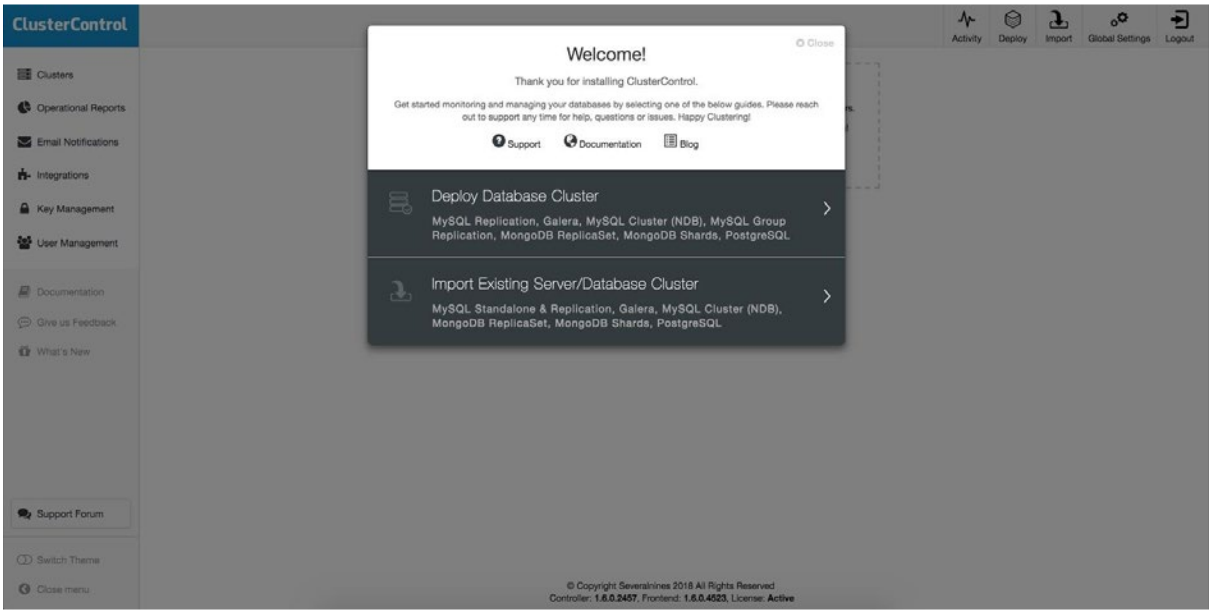

Figure 2: ClusterControl PostgreSQL Deploy 1

To perform a deployment, simply select the option “Deploy Database Cluster” and follow the instructions that appear.

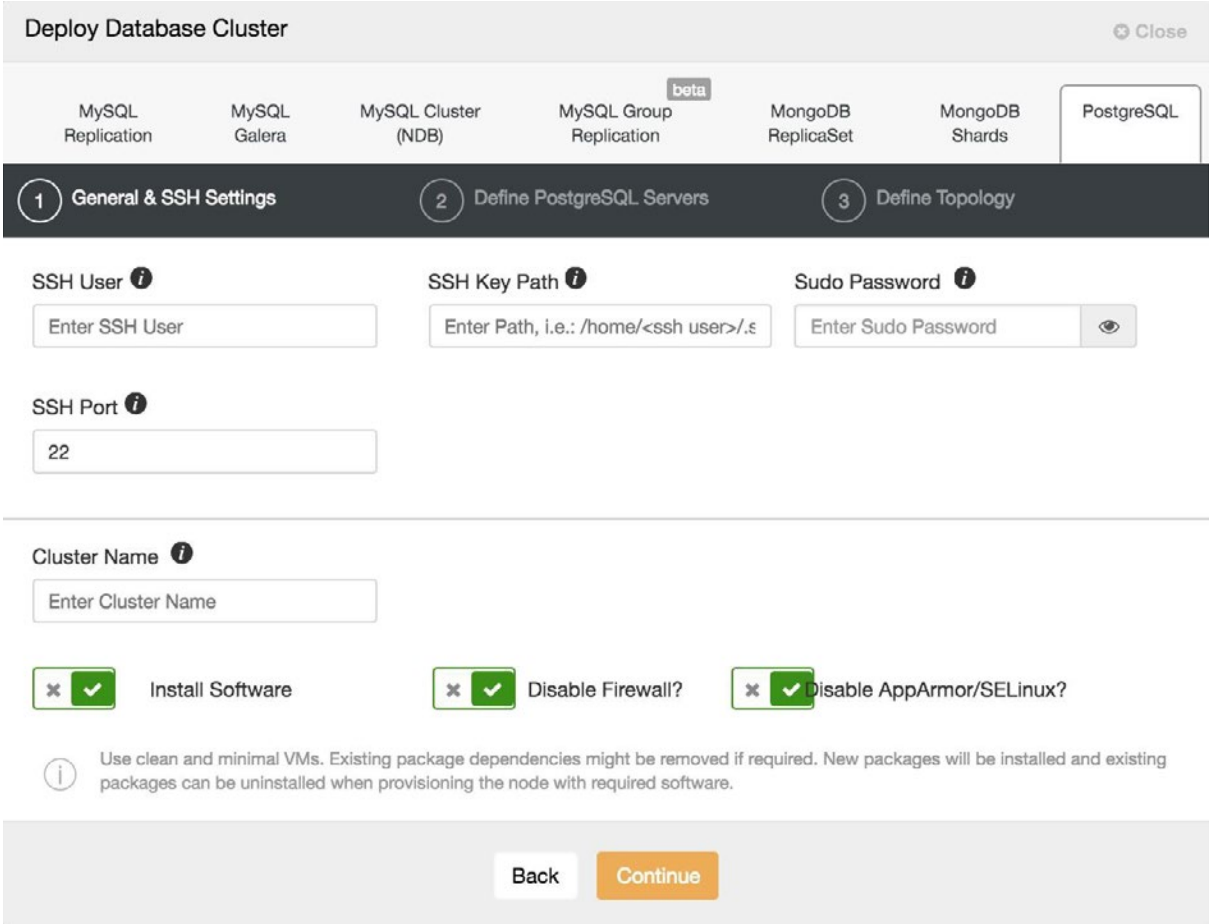

Figure 3: ClusterControl PostgreSQL Deploy 2

When selecting PostgreSQL, we must specify User, Key or Password and port to connect by SSH to our servers. We also need the name for our new cluster and if we want ClusterControl to install the corresponding software and configurations for us.

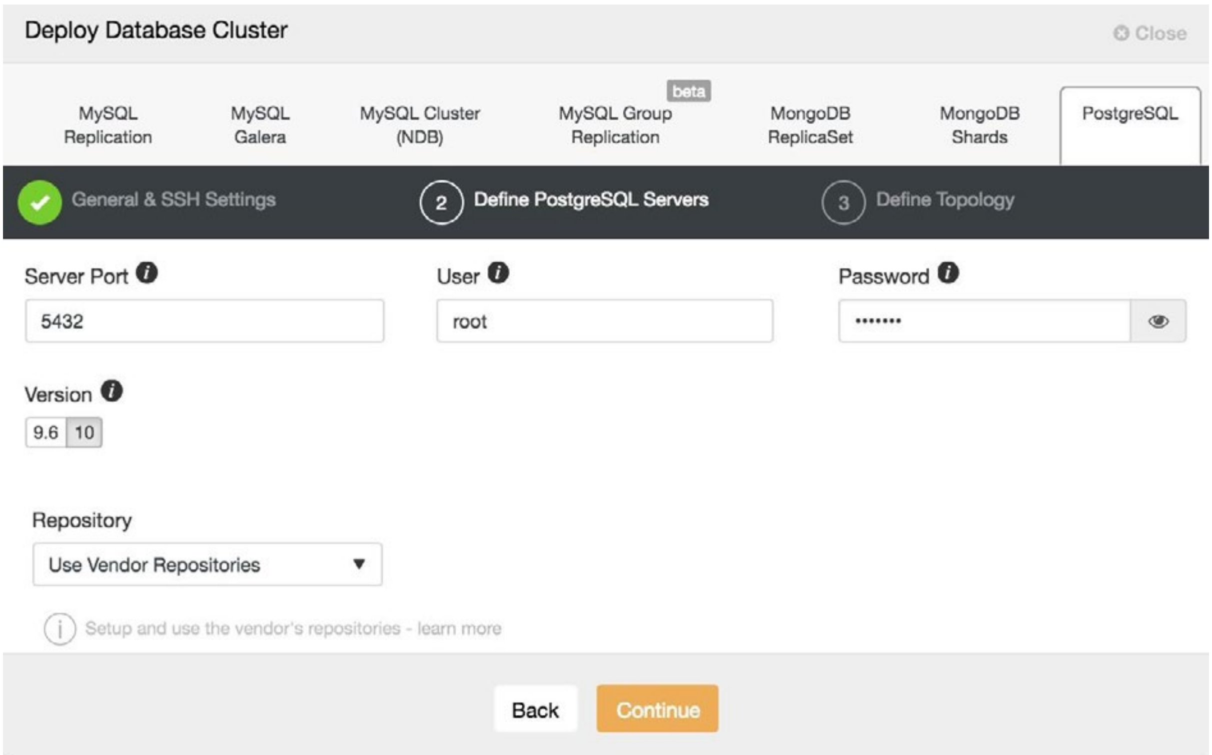

Figure 4: ClusterControl PostgreSQL Deploy 3

After setting up the SSH access information, we must enter the data to access our database.

We can also specify which repository to use.

In the next step, we need to add our servers to the cluster that we are going to create.

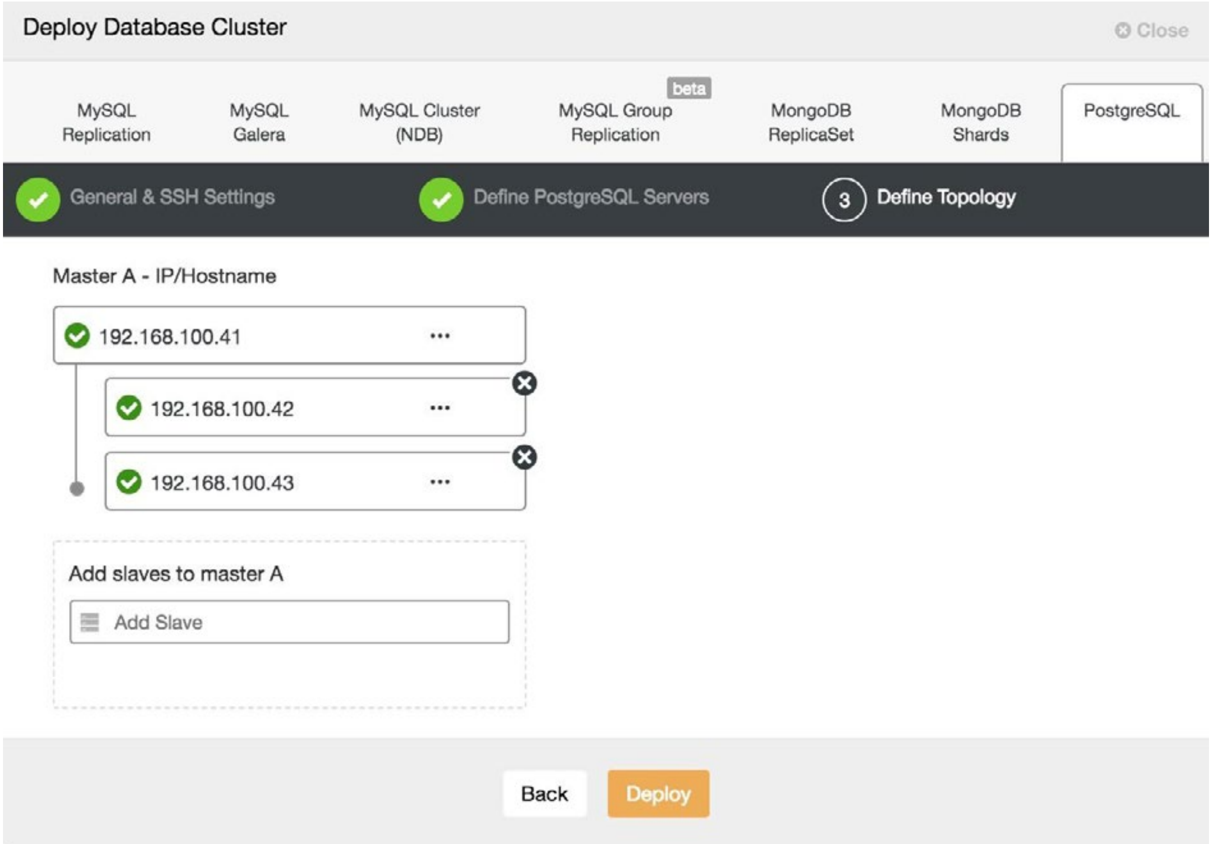

Figure 5: ClusterControl PostgreSQL Deploy 4

When adding our servers, we can enter IP or hostname. For the latter, we must have a DNS server or have added our PostgreSQL servers to the local resolution file (/etc/hosts) of our ClusterControl, so it can resolve the corresponding name that you want to add.

For our example we will create a cluster in PostgreSQL with 3 servers, one master and two slaves.

We can monitor the status of the creation of our new cluster from the ClusterControl activity monitor.

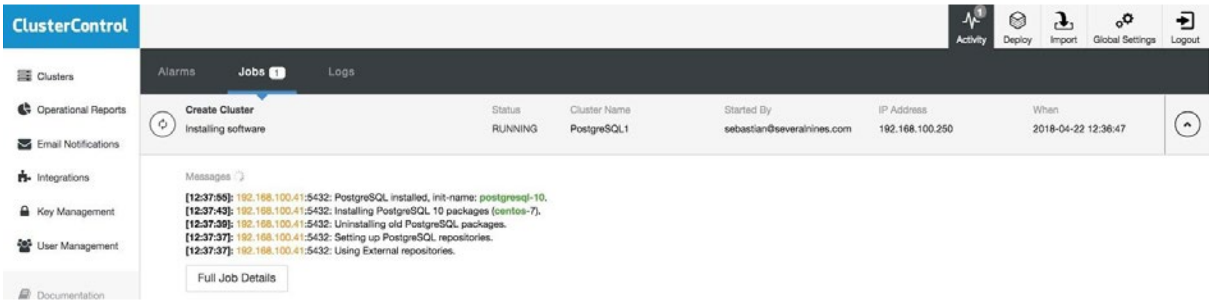

Figure 6: ClusterControl PostgreSQL Deploy 5

Once the task is finished, we can see our cluster in the main ClusterControl screen.

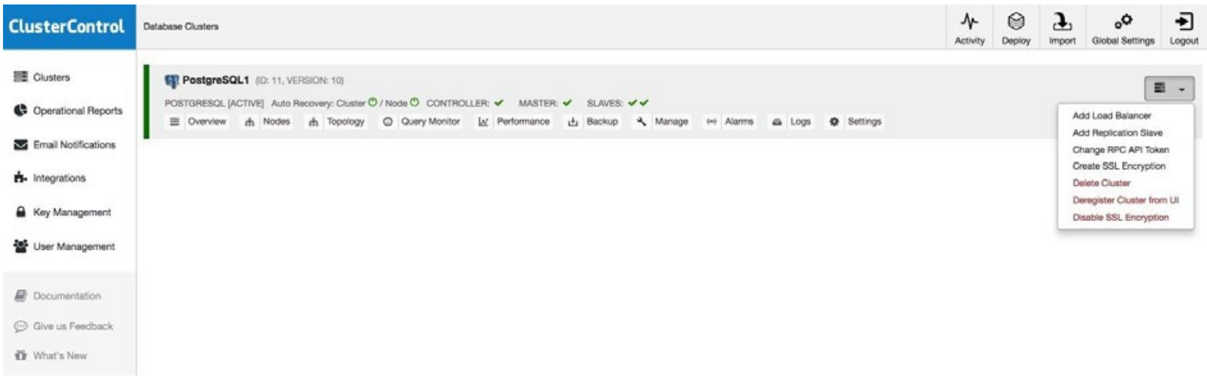

Figure 7: ClusterControl Cluster View

As we can see in the image, once we have our cluster created, we can perform several tasks on it, like adding a load balancer (HAProxy) or a new replica.

6.2. Import

We also have the option to manage an existing cluster by importing it into ClusterControl.

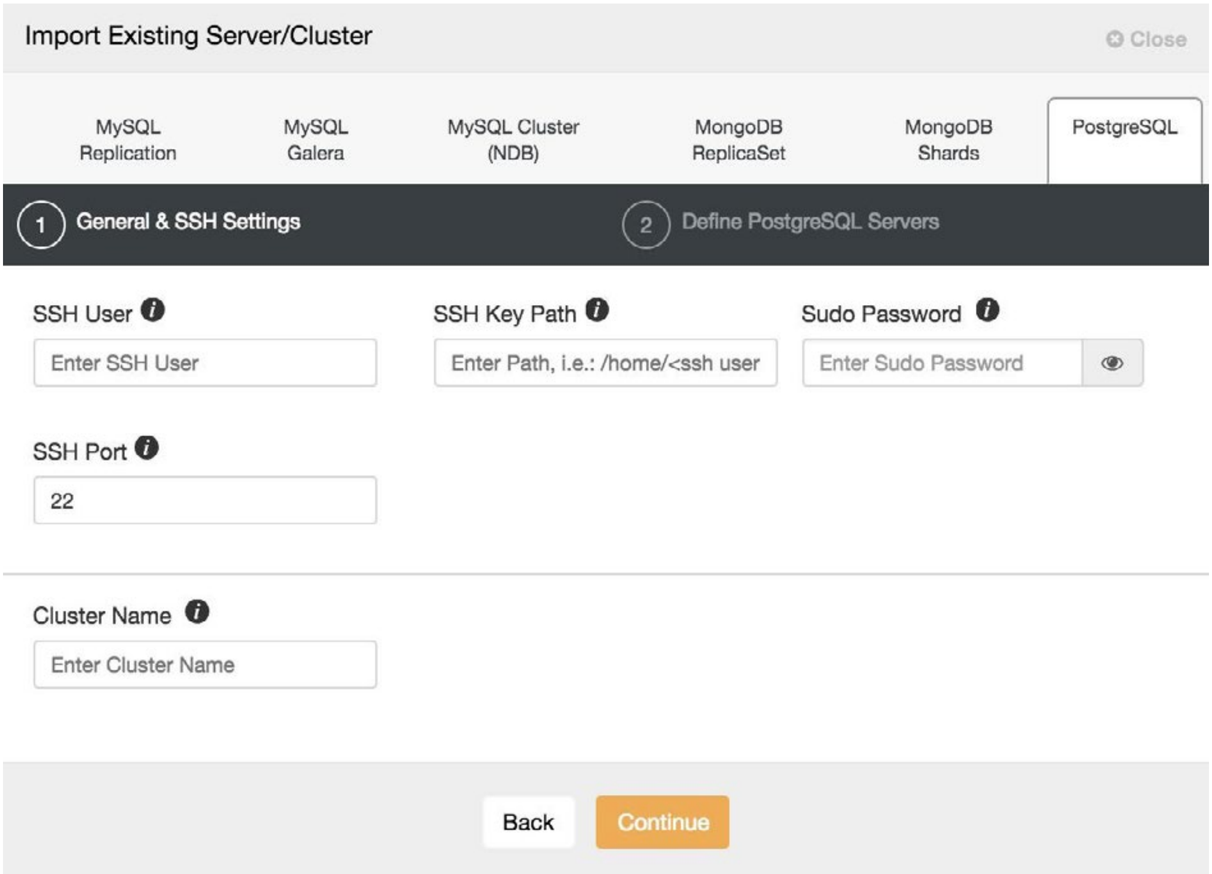

Figure 8: ClusterControl PostgreSQL Import 1

First, we must enter the SSH access credentials to our servers.

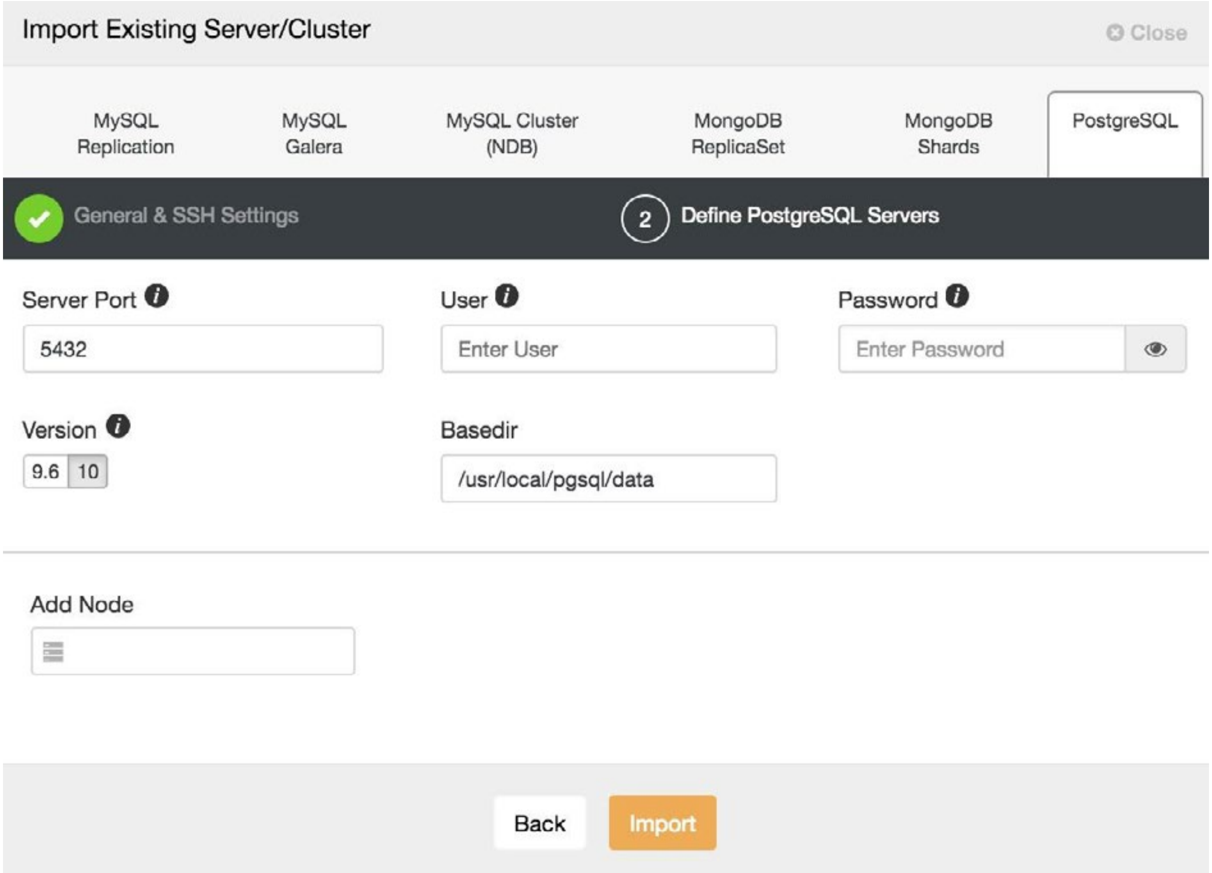

Figure 9: ClusterControl PostgreSQL Import 2

Then we enter the access credentials to our database, the basedir and the version. We add the nodes by IP or hostname, in the same way as when we deploy, and press on Import. Once the task is finished, we are ready to manage our cluster from ClusterControl.

6.3. Scalability

As we saw earlier in *figure 7*, we can add slaves to our topology very easily.

If we go to cluster actions and select "Add Replication Slave", we can either create a new replica from scratch, or add an existing PostgreSQL database as a replica.

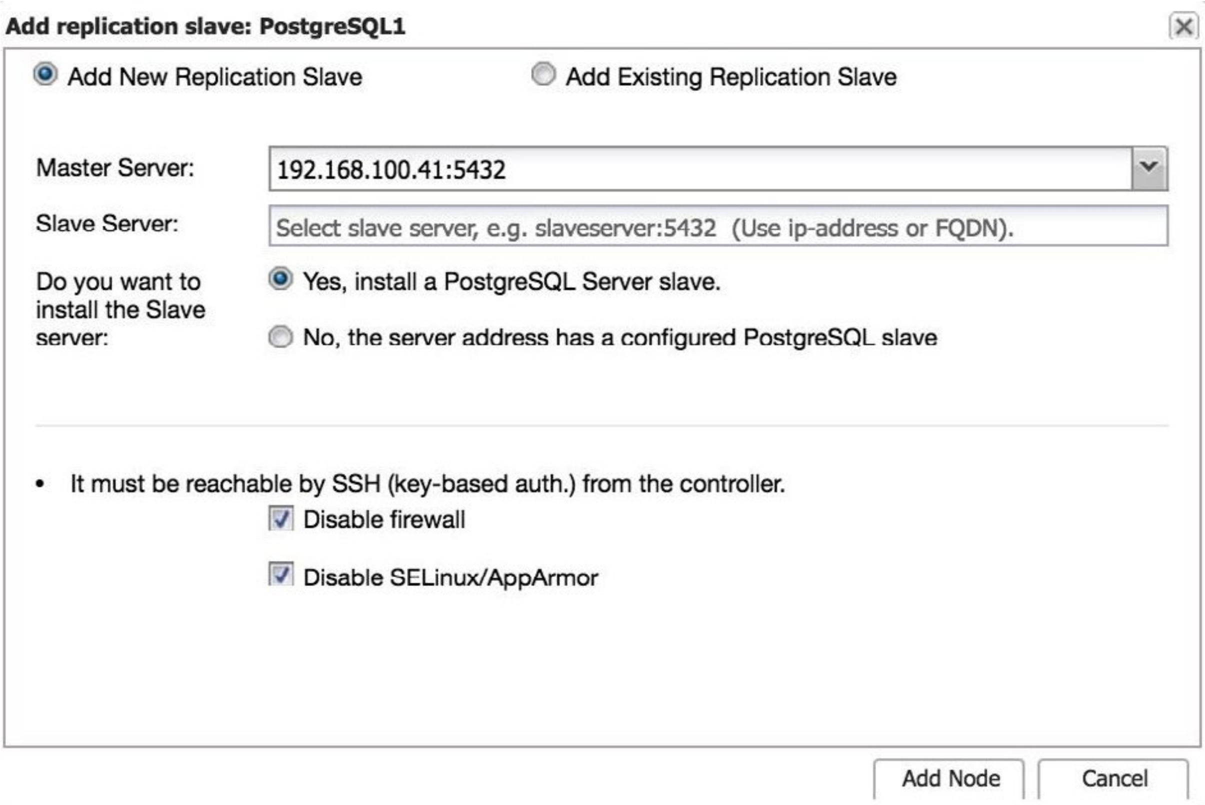

Figure 10: ClusterControl PostgreSQL Import 3

As you can see in the image, we can have our new replica running in a few minutes.

In this way we can add as many replicas as we want, and spread read traffic between them using a load balancer, which we can also implement with ClusterControl.

6.4. Failover

ClusterControl manages failover on our replication setup. It detects master failures and promotes a slave with the most current data as new master. It also fails over the rest of the slaves to replicate from the new master. As for client connections, it leverages 2 main tools for the task: HAProxy and Keepalived.

HAProxy is a load balancer that distributes traffic from one origin to one or more destinations and can define specific rules and/or protocols for this task. If any of the destinations stops responding, it is marked as offline, and the traffic is sent to the rest of the available destinations. This prevents traffic from being sent to an inaccessible destination and prevents the loss of this information by directing it to a valid destination.

Keepalived allows you to configure a virtual IP within an active/passive group of servers. This virtual IP is assigned to an active “Main” server. If this server fails, the IP is automatically migrated to the “Secondary” server that was found to be passive, allowing it to continue working with the same IP in a transparent way for our systems.

Suppose we have the following topology:

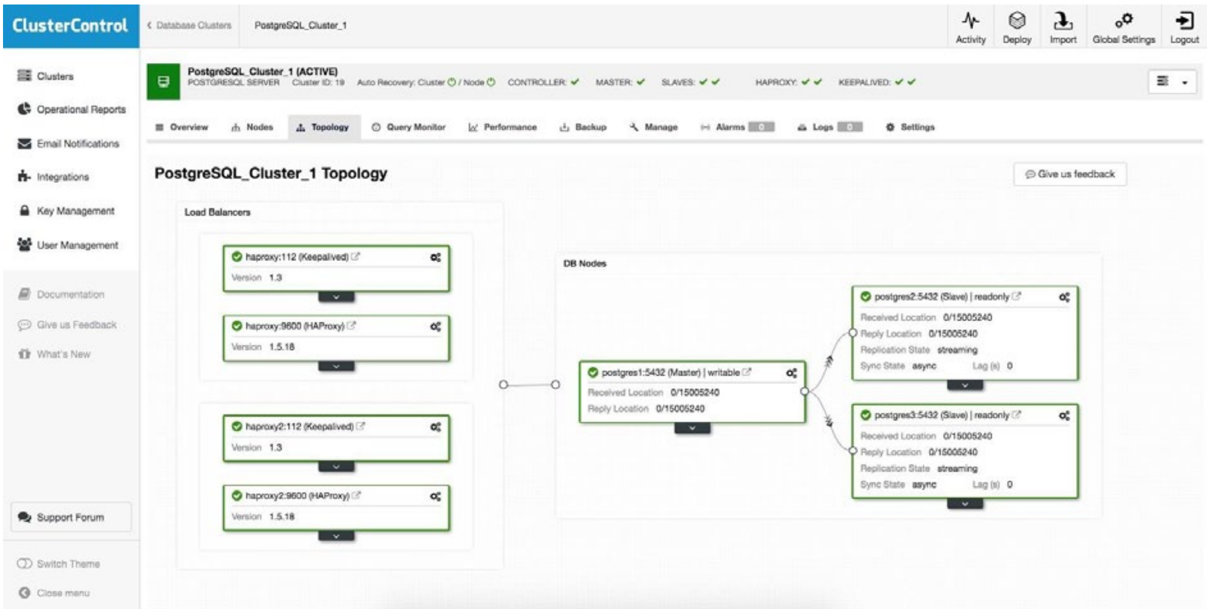

Figure 11: ClusterControl PostgreSQL Failover Topology 1

We have 2 load balancers (HAProxy) configured with Keepalived, in front of 3 PostgreSQL database nodes (1 master and 2 slaves).

As a first case, let's see what happens if our master database fails.

Having the "Autorecovery" option ON, our ClusterControl will perform an automatic failover as well as notify us of the problem. In this way, our systems can recover in seconds, and without our intervention.

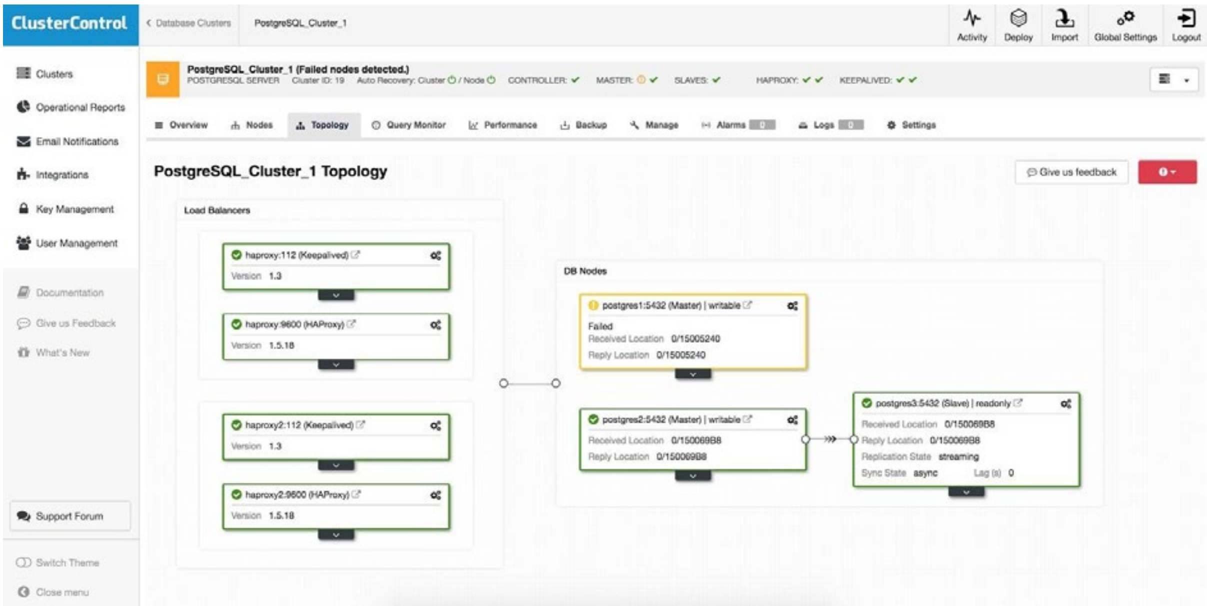

Figure 12: ClusterControl PostgreSQL Failover Topology 2

To perform the failover, two things are taken into account, one is the blacklist or whitelist (if it is configured). If any slave is in blacklist, it will not be taken into account to be promoted to master. The nodes that are in the whitelist are candidates for master promotion.

The second thing to keep in mind is the most advanced slave, for this ClusterControl checks the pg_current_xlog_location (PostgreSQL 9+) or pg_current_wal_lsn (PostgreSQL 10+) depending on the version of our database.

Once it has determined what is going to be our new master, ClusterControl will promote it and at this point our load balancer comes into play.

HAProxy is configured with two different ports, one read-write and one read-only.

In our read-write port, we have our master server as online and the rest of our nodes as offline, and in the read-only port we have both the master and the slaves online. In this way we can balance the reading traffic between our nodes but we make sure, that at the time of writing, the read-write port will be used, writing in the master that is the server that is online.

When HAProxy detects that one of our nodes, either master or slave, is not accessible, it automatically marks it as offline and does not take it into account for sending traffic to it. This check is done by healthcheck scripts that are configured by ClusterControl at time of deployment. These check whether the instances are up, whether they are undergoing recovery, or are read-only.

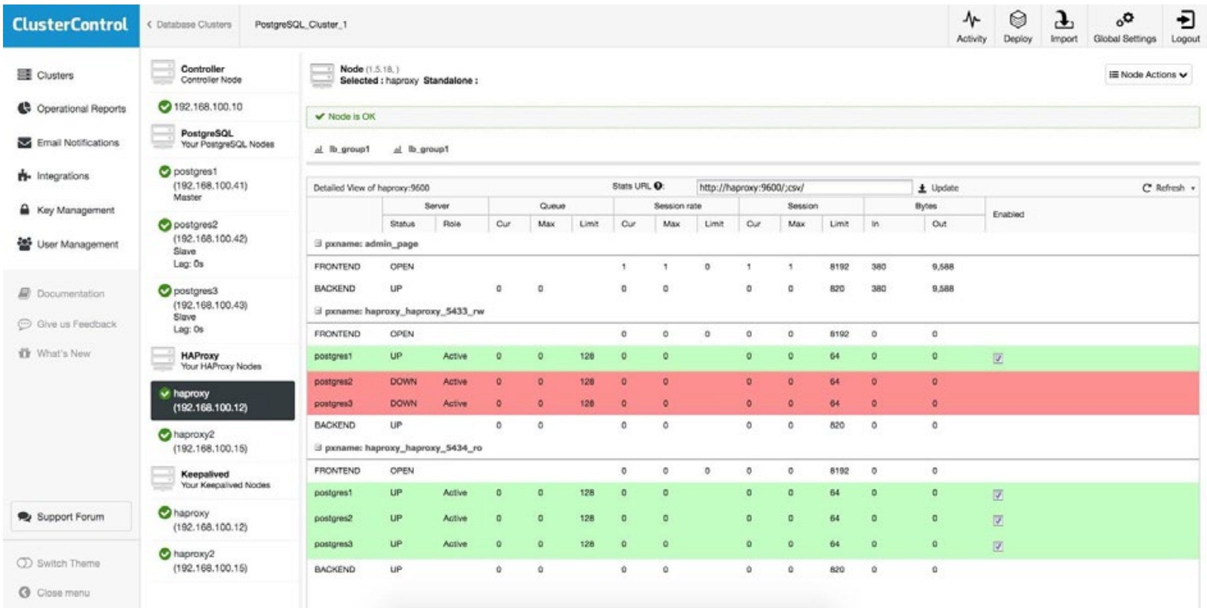

Figure 13: ClusterControl PostgreSQL Failover HAProxy 1

When ClusterControl promotes a slave to master, our HAProxy marks the old master as offline (for both ports) and puts the promoted node online (in the read-write port). In this way, our systems continue to operate normally.

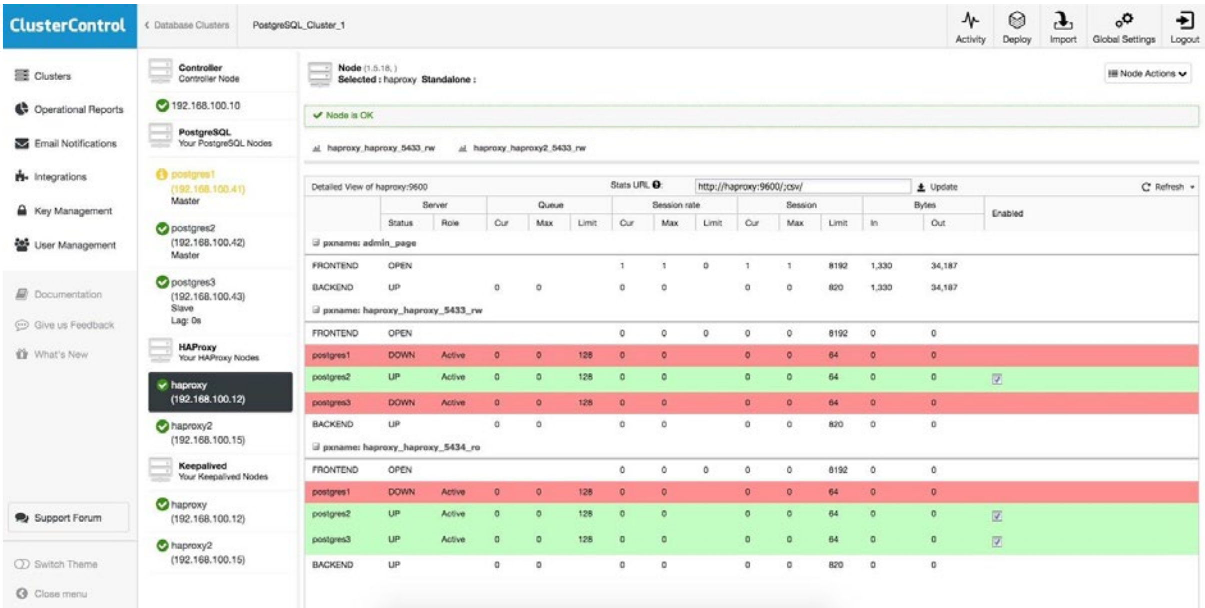

Figure 14: ClusterControl PostgreSQL Failover HAProxy 2

Note that if we manage to recover our old failed master, it will NOT be re-introduced automatically to the cluster, neither as a master nor as a slave. We need to do it manually. One reason for this is that, if our replica was delayed at the time of the failure, if we add the old master to the cluster either as a master or as a slave, it would mean loss of information or inconsistency of data across nodes. We might also want to analyze the issue in detail, but when adding it to our cluster, we would possibly lose diagnostic information.

Another important detail is that, if failover fails, no further attempts are made, manual intervention is required to analyze the problem and perform the corresponding actions. This is to avoid the situation where ClusterControl, as the high availability manager, tries to promote the next slave and the next one. There might be a problem, and we do not want to make things worse by attempting multiple failovers.

To add our old master to the cluster, we must go to the actions of the node and select “Rebuild Replication Slave”. Once added to the cluster, we can promote it to master by selecting the option “Promote Slave”.

As a second case, let’s see what happens if our active load balancer fails.

If our active HAProxy, which is assigned a Virtual IP address to which our systems connect, fails, Keepalived migrates this IP to our passive HAProxy automatically. This means that our systems are then able to continue to function normally.

Let’s see an example.

Here is HAProxy server (active keepalived node):

```
$ ip addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 192.168.100.12/24 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.100.13/32 scope global eth0
       valid_lft forever preferred_lft forever
```

As you can see, we have 2 differents IP assigned. The second one is our Virtual IP Address.

We can check this with the following command:

```
$ grep “virtual_ipaddress” -A1 /etc/keepalived/keepalived.conf
   virtual_ipaddress {
       192.168.100.13                       # the virtual IP
```

And here is HAProxy2 server (passive keepalived node):

```
$ ip addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 192.168.100.15/24 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
```

We have only one IP, because this server is the passive keepalived node.

If you have any issue with HAProxy active node:

```
$ shutdown -h now
Connection to haproxy closed by remote host.
```

And you check the IP address of HAProxy2 again:

```
$ ip addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 192.168.100.15/24 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.100.13/32 scope global eth0
       valid_lft forever preferred_lft forever
```

Now, our Virtual IP is in HAProxy2.

Once our active HAProxy is recovered, the virtual IP is reassigned to the original active server and our current active server returns to its passive role.

6.5. Load balancing

Returning to *figure 7*, if we select “Add Load Balancer” or, from the cluster view, we go to Manage -> Load Balancer, we can add load balancers to our database topology.

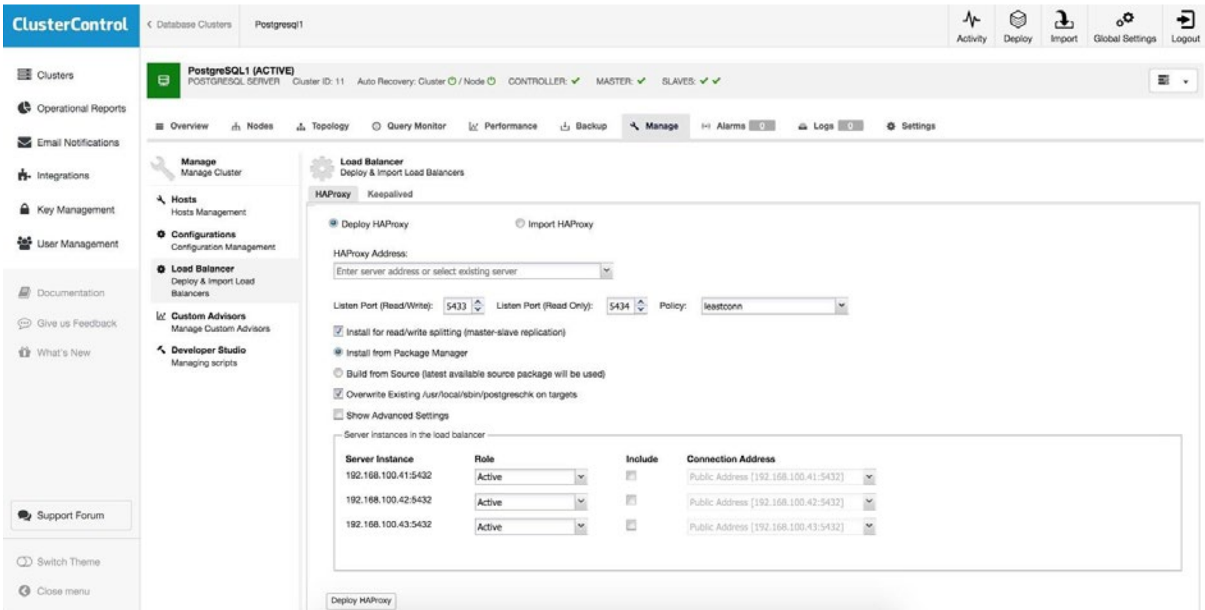

Figure 15: ClusterControl PostgreSQL Load Balancer

We could see that, the configuration to create our new load balancer is very simple. We only need to add IP/Name, port, policy and the nodes we are going to use.

Also, as we saw earlier in the failover, we can add two load balancers and add keepalived to them, which allows us to have an automatic failover of our load balancer in case of failure.

Keepalived uses a virtual IP, and migrates it from one load balancer to another in case of failure, so our setup can continue to function normally.

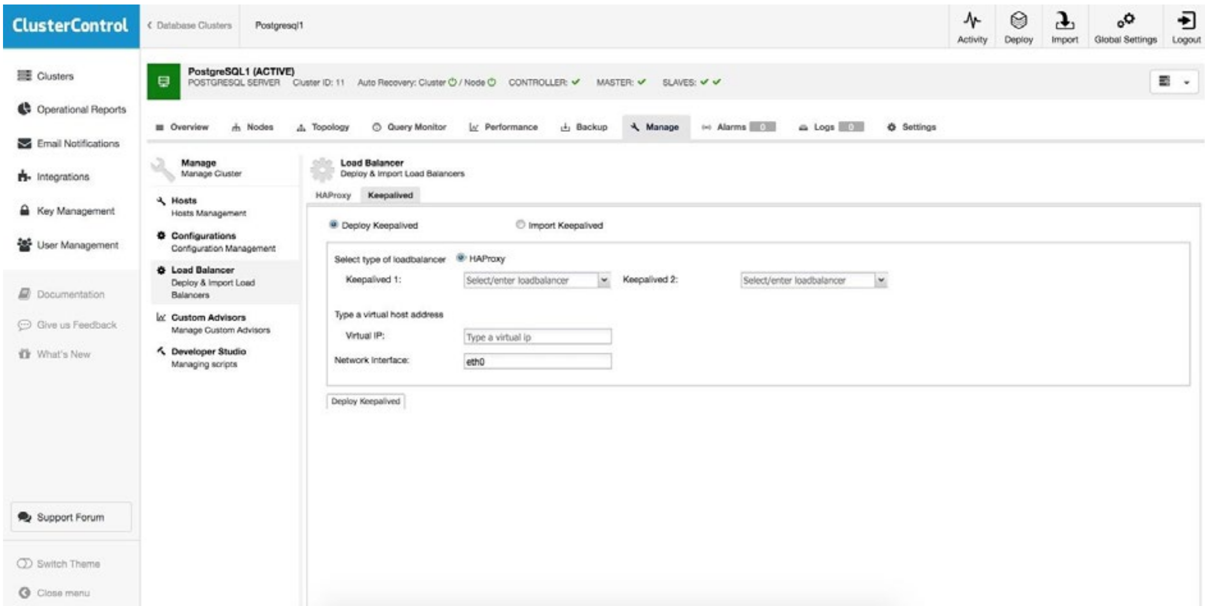

Figure 16: ClusterControl PostgreSQL Keepalived

In the example we discussed, our topology would be as follows (using the topology view in ClusterControl).

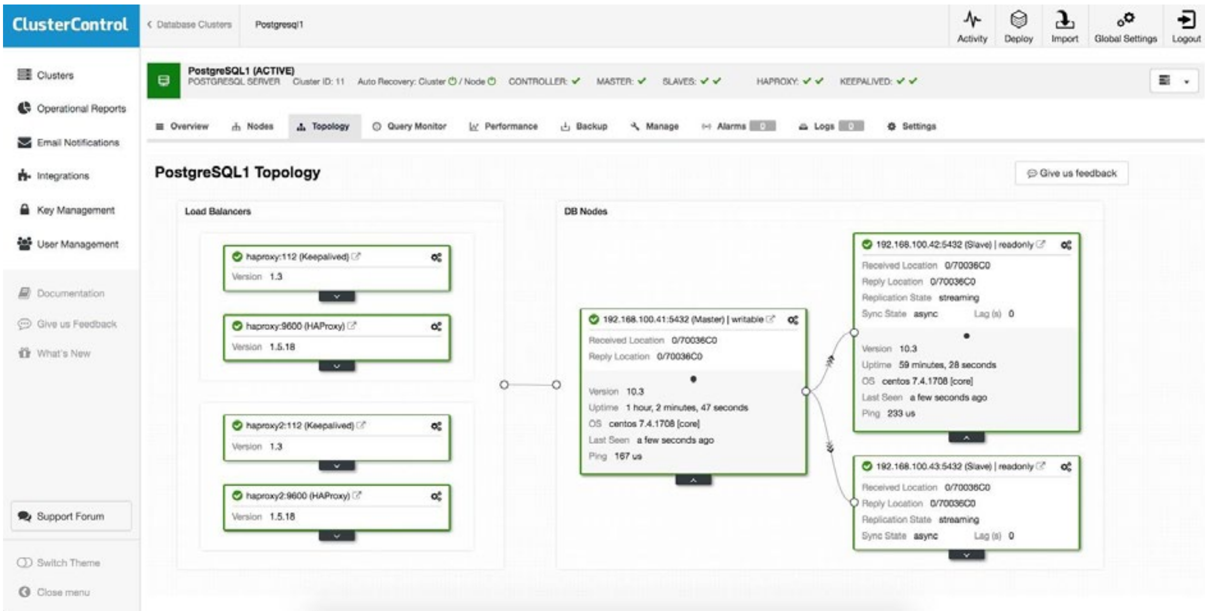

Figure 17: ClusterControl PostgreSQL Topology

6.6. Monitoring

ClusterControl allows us to monitor our servers in real time. We will have graphs with basic data such as CPU, Network, Disk, RAM, IOPS, as well as database metrics collected from the PostgreSQL instances. Database queries can be viewed from the Query Monitor.

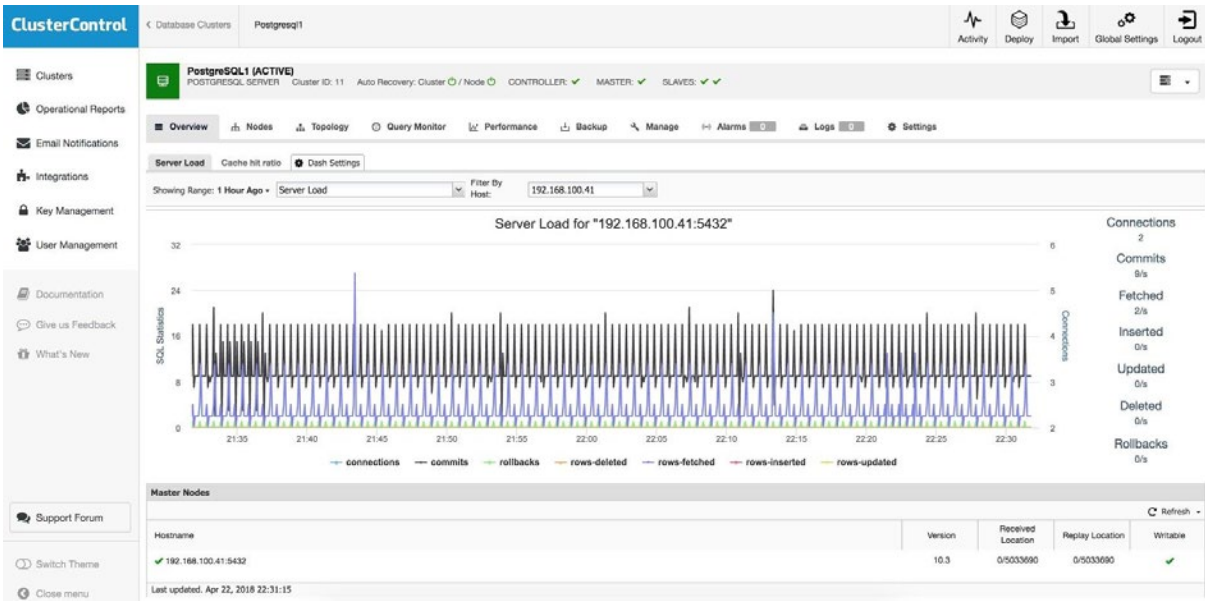

Figure 18: ClusterControl PostgreSQL Cluster Overview

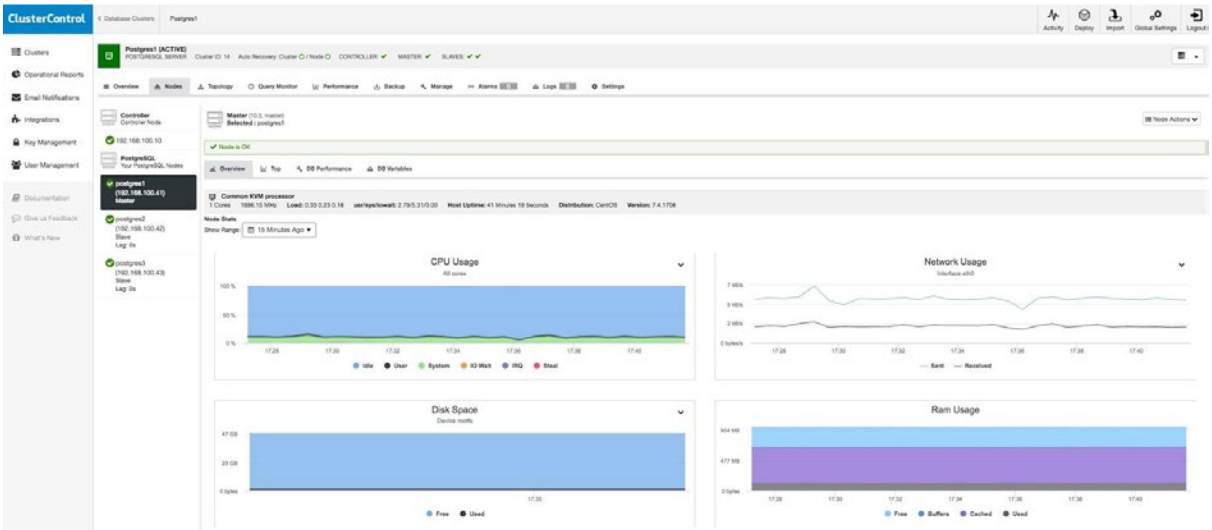

Figure 19: ClusterControl PostgreSQL Node Overview

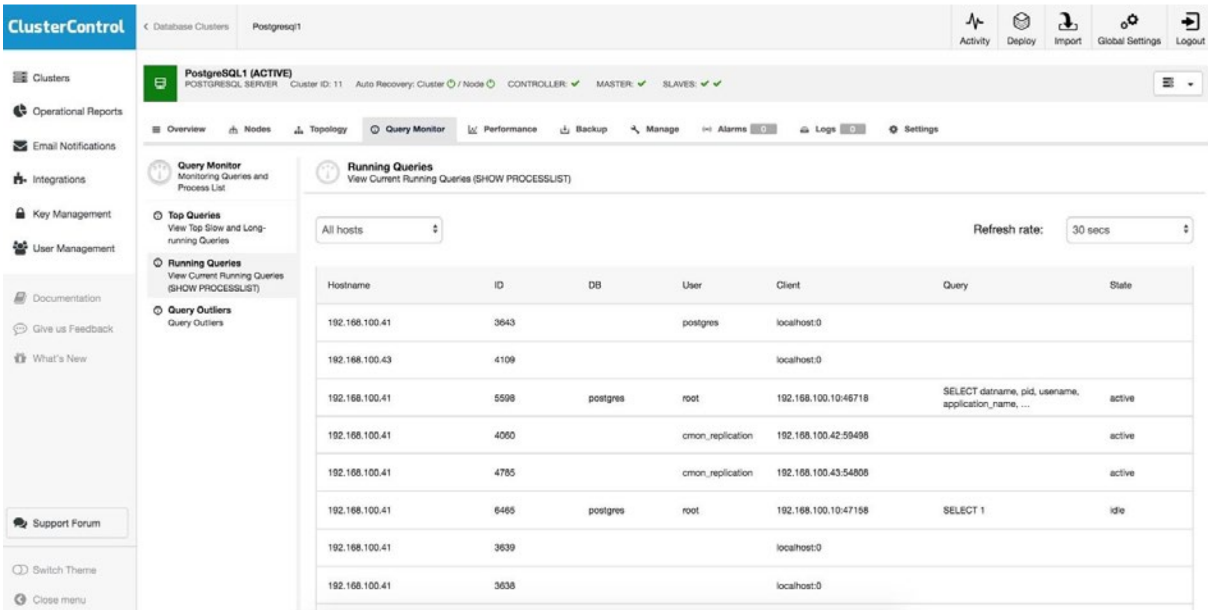

Figure 20: ClusterControl PostgreSQL Query Monitor

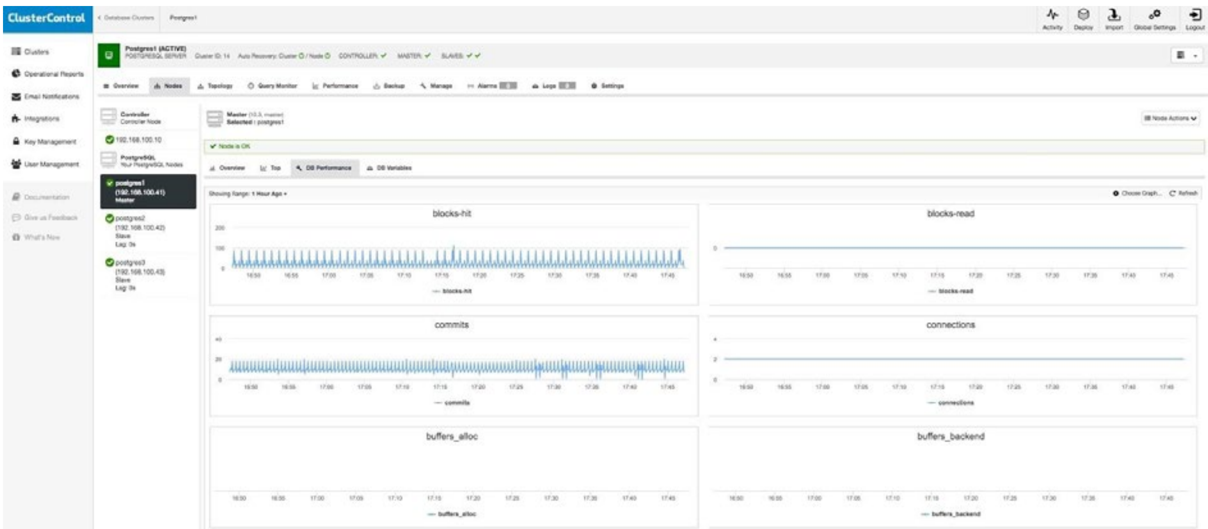

Figure 21: ClusterControl PostgreSQL DB Performance

In this way, we can have our cluster fully monitored, without adding additional tools or utilities.

6.7. Alerts

In the same way that we enable monitoring from ClusterControl, we can also setup alerts, which inform us of events in our cluster.

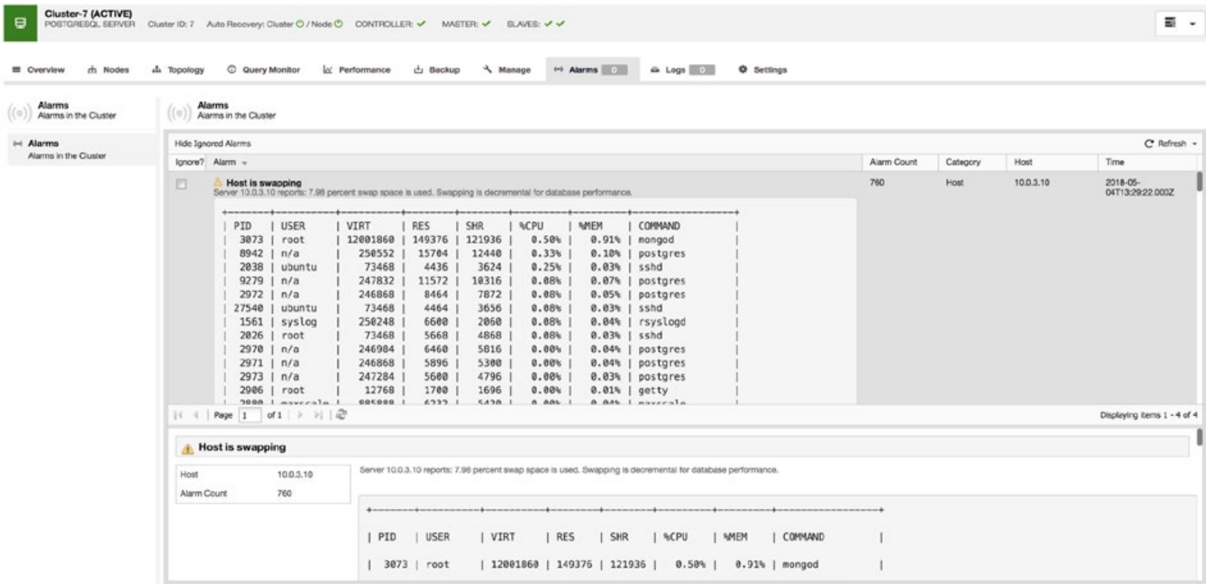

Figure 22: ClusterControl PostgreSQL Alarms

These alerts are configurable, and can be personalized as needed.

We can see the predefined advisors in Cluster -> Performance -> Advisors.

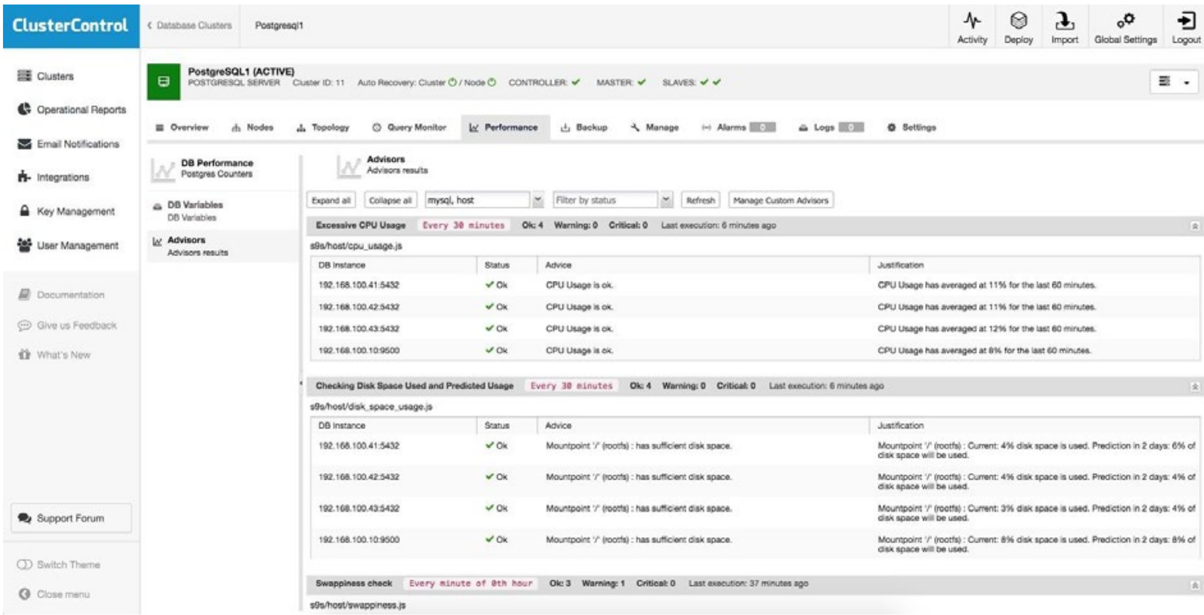

Figure 23: ClusterControl PostgreSQL Advisors

As we mentioned, we can easily configure our own advisors, as seen in figure 24.

We can check our custom advisors in Cluster -> Manage -> Custom Advisors.

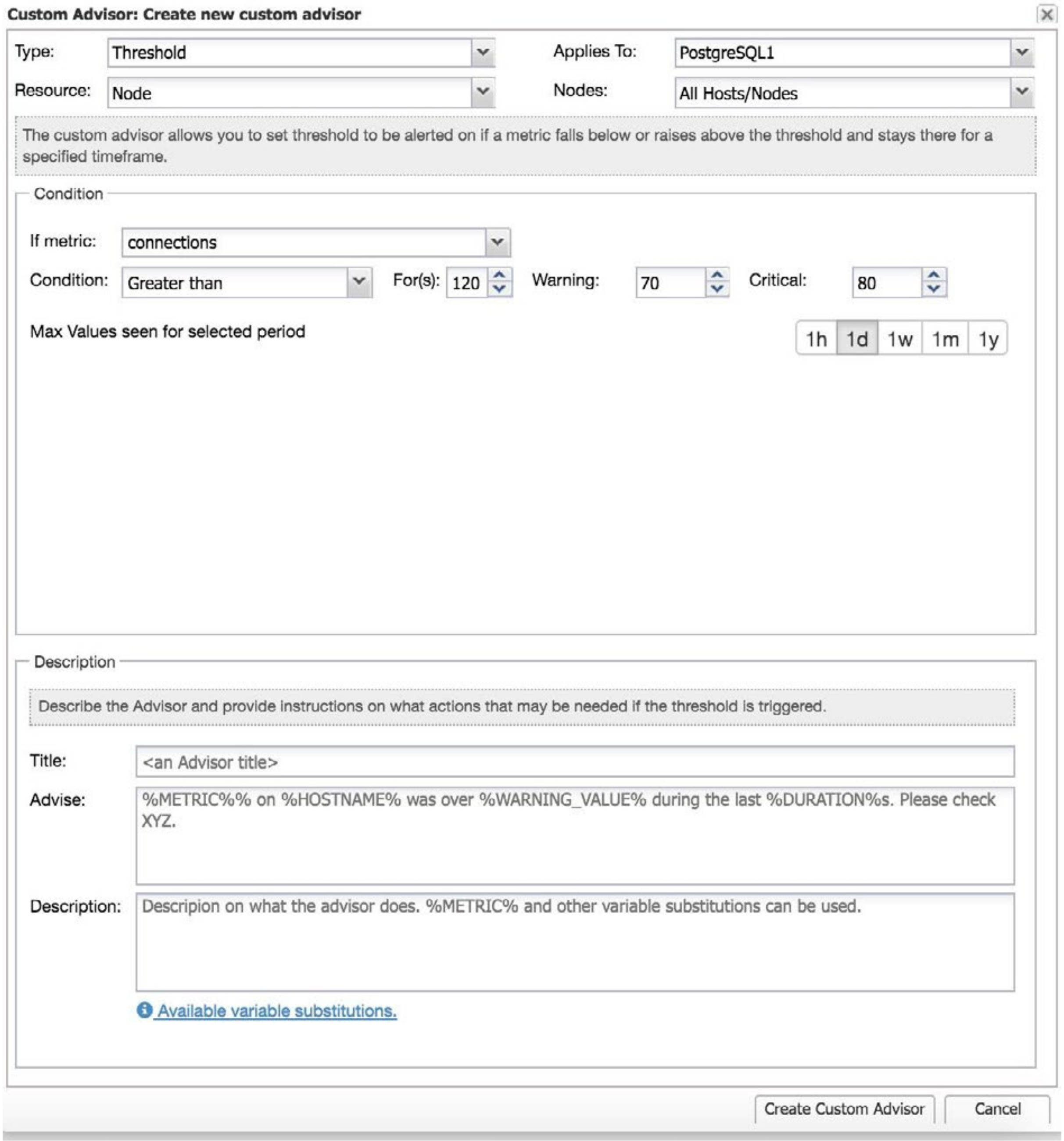

Figure 24: ClusterControl PostgreSQL Custom Advisors

6.8. Reports

As IT infrastructure plays a key role in the business, it is important to understand how it is evolving with time. From a database standpoint, one would keep track of any change or behavior to the database systems - are we having new peaks in traffic, how are the databases growing, will we be running out of disk space, and so on. With current and historical data, we can generate a history of the state of our systems for analysis.

ClusterControl has the ability to generate reports automatically, and store or send them by mail.

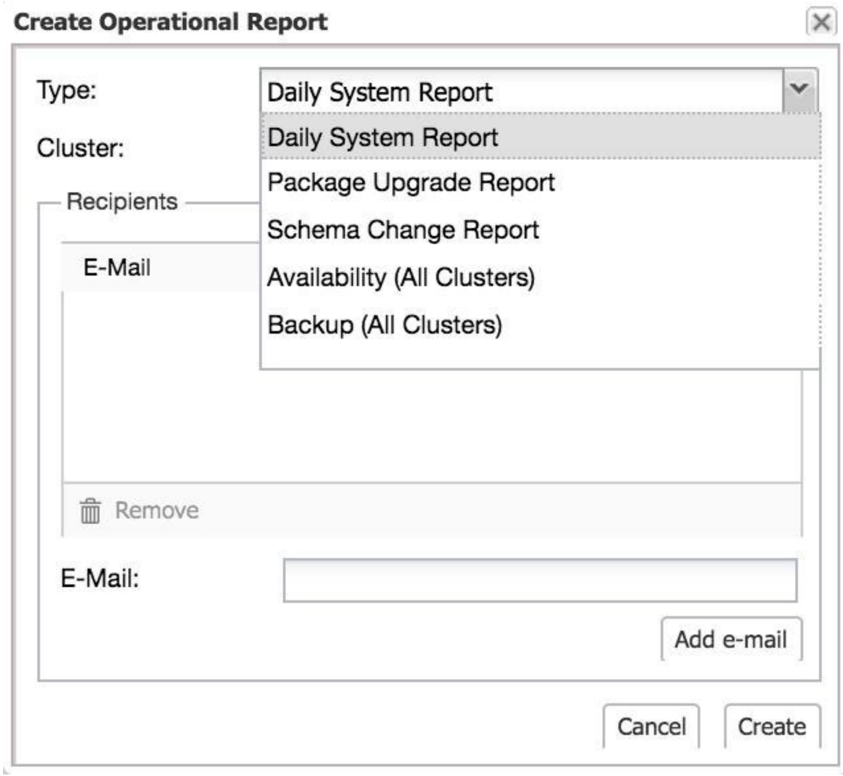

Figure 25: ClusterControl PostgreSQL Create Report

As you can see in figure 25, ClusterControl has several types of reports to generate, each one with specific information.

If we look at figure 26, we can see an example of a report generated with ClusterControl, that contains information about one of our nodes.

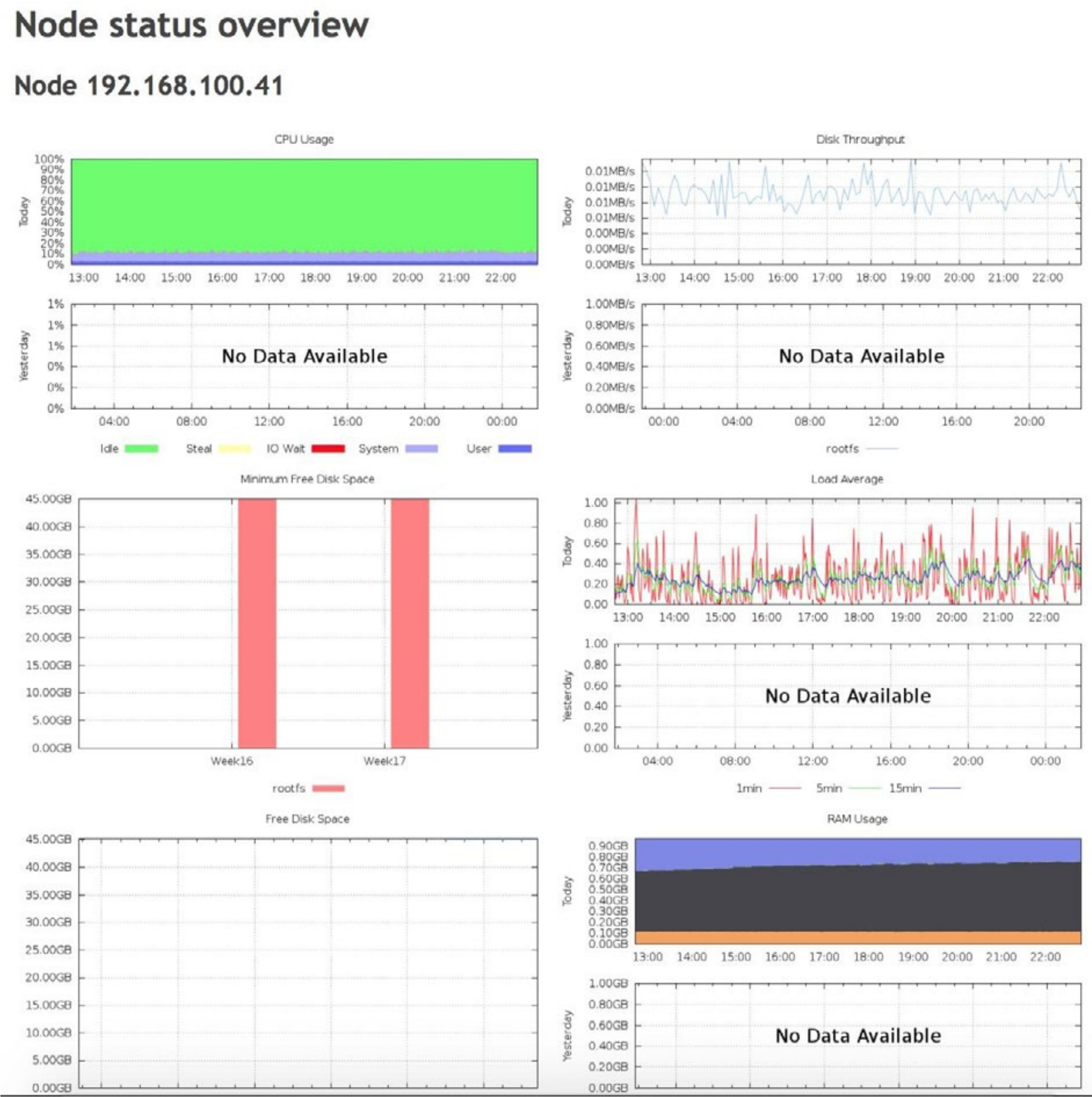

Figure 26: ClusterControl PostgreSQL Node Report

6.9. Backups

We have already discussed the importance of having backups, either for disaster recovery or to consult historical information that is not required to have online.

ClusterControl provides the functionality either to generate an immediate backup or to schedule one, and automate the task in a simple and fast way.

We can choose between two backup methods, pgdump (logical) and pg_basebackup (binary). We can also specify where to store the backups (on the database server, on the ClusterControl server or in the cloud) and the compression level.

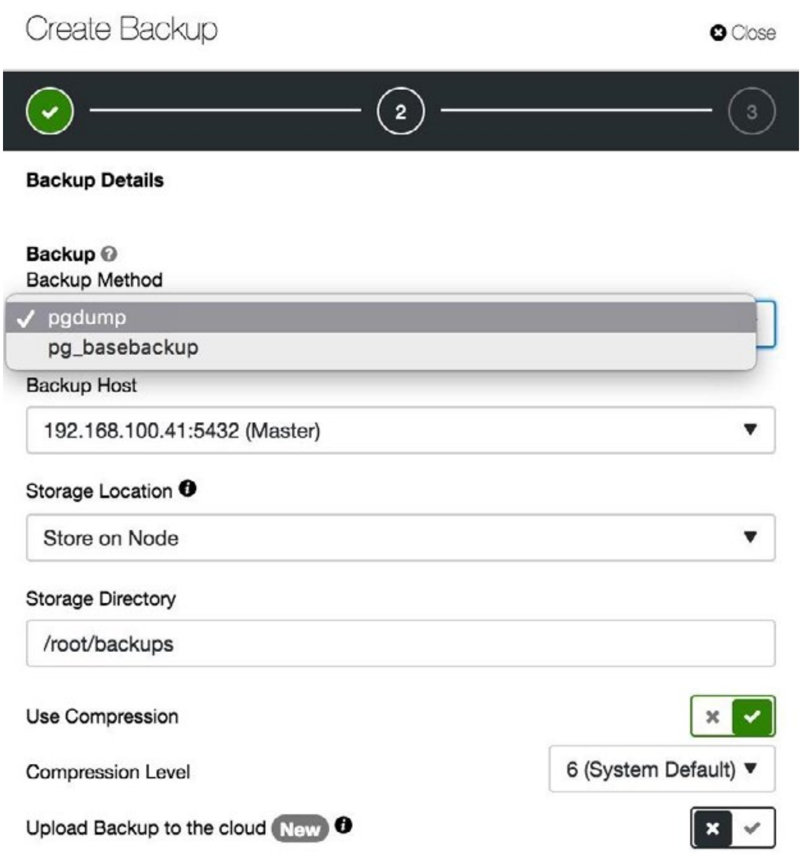

Figure 27: ClusterControl PostgreSQL Create Backup 1

In the next step we can encrypt our backup and specify the retention period.

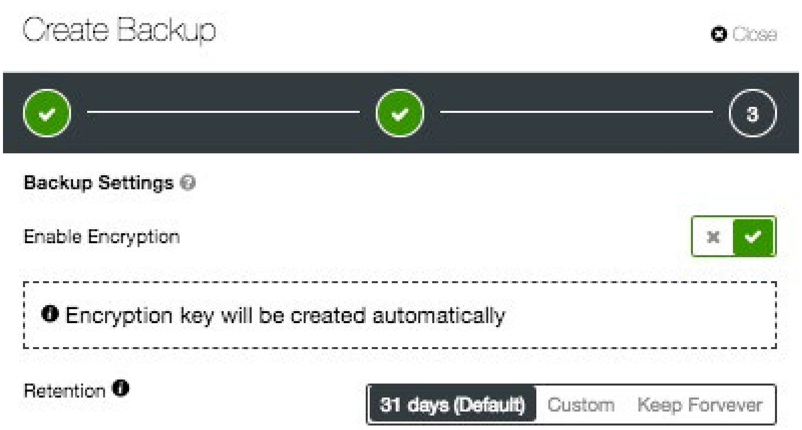

Figure 28: ClusterControl PostgreSQL Create Backup 2

If we selected the option "Upload Backup to the cloud" in the previous step, in the next, we can choose between 3 of the main cloud providers - Amazon Web Services, Google Cloud or Microsoft Azure.

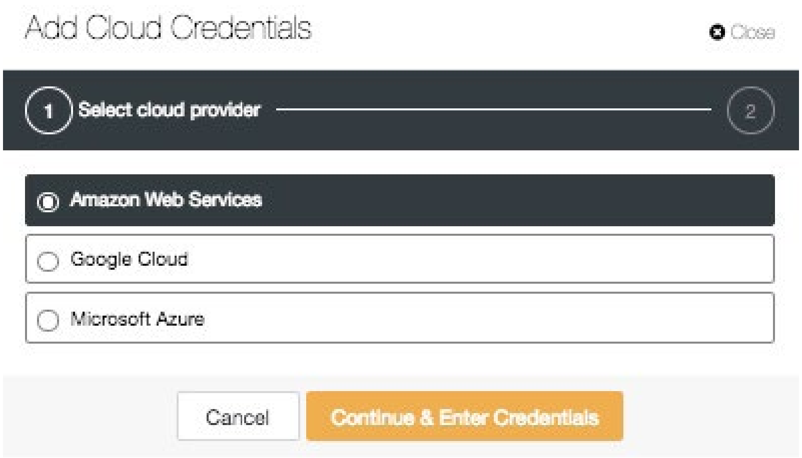

Figure 29: ClusterControl PostgreSQL Create Backup 3

When selecting any of them, we will be asked for the information that corresponds to upload the backup to our cloud.

Figure 30: ClusterControl PostgreSQL Create Backup 4

When scheduling a backup, in addition to select the options mentioned above, we also need to specify when these backups will be made and how often.

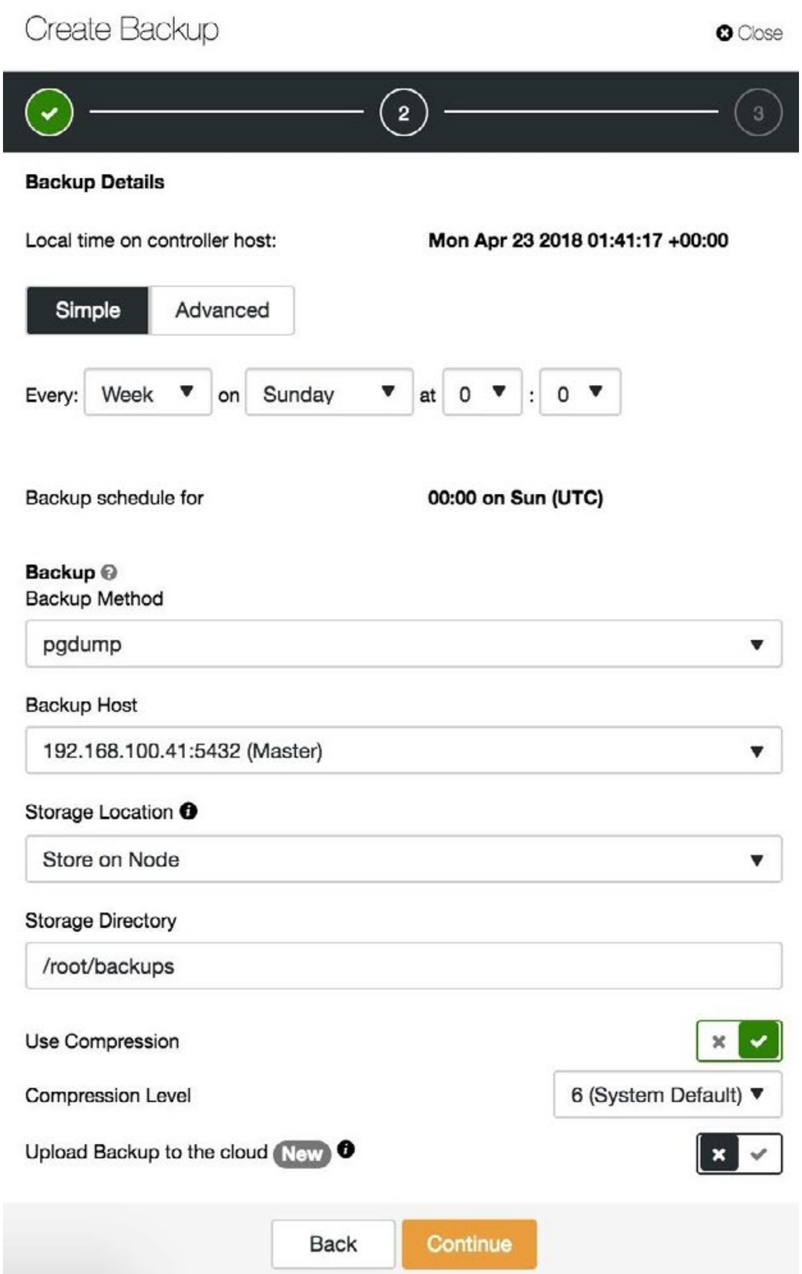

Figure 31: ClusterControl PostgreSQL Scheduled Backup

6.10. Topology view

A very interesting feature implemented by ClusterControl is the topology view.

To use it we must go to Cluster -> Topology. It allows us to visualize our database topology and load balancers, and check the status of our servers and replicas. We can even perform actions on our nodes from there, like recreating a replica, promoting a node to master or restart a node. To reconstruct a replica, we can also simply drag one of our nodes over the master and, within few seconds, we can have our reconstructed replica.

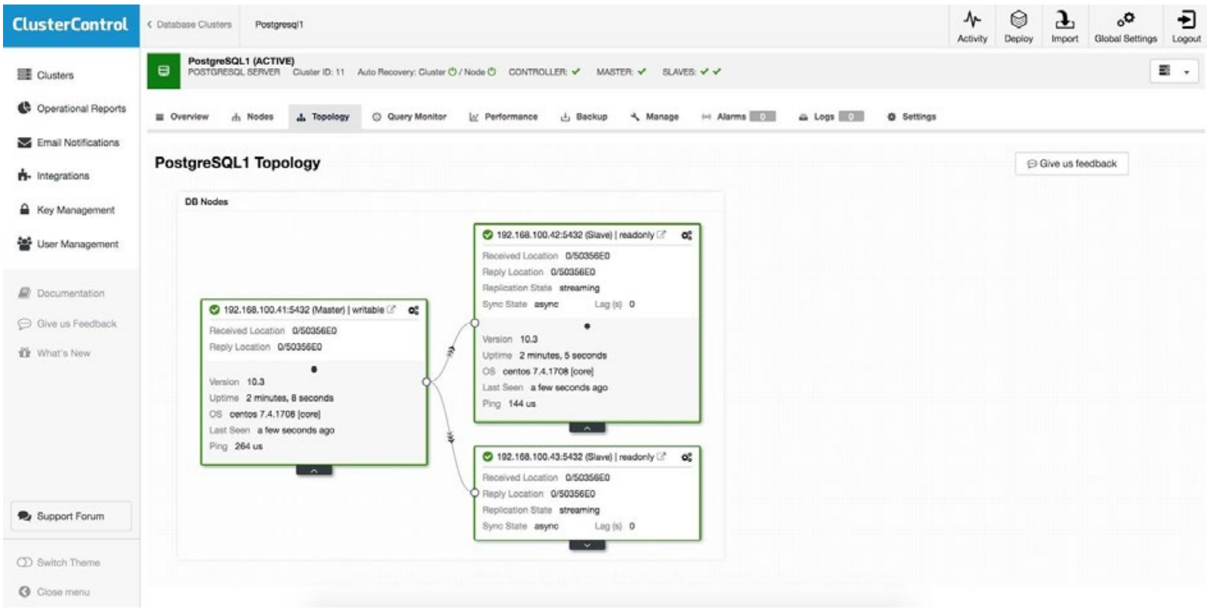

Figure 32: ClusterControl PostgreSQL Topology View

6.11. Integrations

ClusterControl allows us to integrate the tool with different services such as PagerDuty or Slack.

In this way we can receive our events in the tools we use daily, in order to centralize our tasks. In the same way we can manage our ClusterControl from external services, such as Slack.

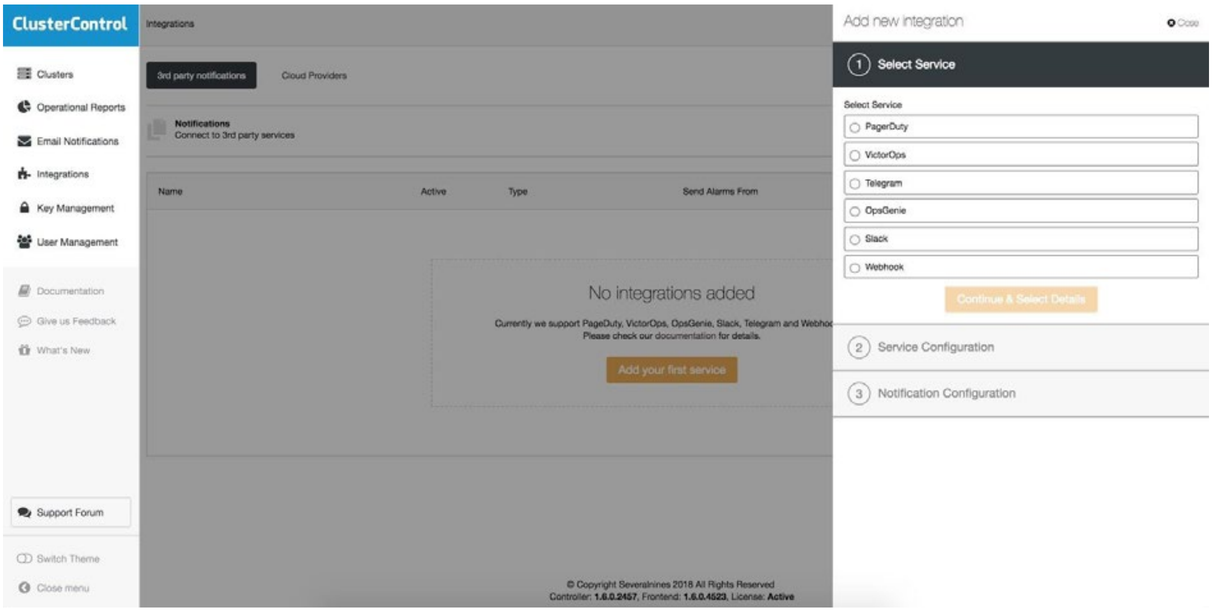

Figure 33: ClusterControl PostgreSQL Integrations

ChatOps via CCBot

CCBot is a chatbot that use the ClusterControl APIs to execute request on your clusters. You will be able to run administration tasks, for example, create backups, read logs, deploy clusters, as well as keep your team up to date on the status of your clusters, jobs and backups. It supports most of the major chat services like Slack, Flowdock and Hipchat.

CCBot is integrated with s9s command line, so you have several commands to use with this tool.

To install CCBot, once we have installed ClusterControl, we must execute the following script:

```
$ /var/www/html/clustercontrol/app/tools/install-ccbot.sh
```

We select which adapter we want to use, in our example we will select Slack.

```
-- Supported Hubot Adapters --
1. slack
2. hipchat
3. flowdock

Select the hubot adapter to install [1-3]: 1
```

It will then ask us for some information, such as an email, a description, the name we will give to our bot, the port, the API token and the channel to which we want to add it.

```
? Owner (User <user@example.com>)
? Description (A simple helpful robot for your Company)
Enter your bot’s name (ccbot):
Enter hubot’s http events listening port (8081):
Enter your slack API token:
Enter your slack message room (general):
```

To obtain the API token, we must go to our Slack -> Apps (On the left side of our Slack window), we look for Hubot and select Install.

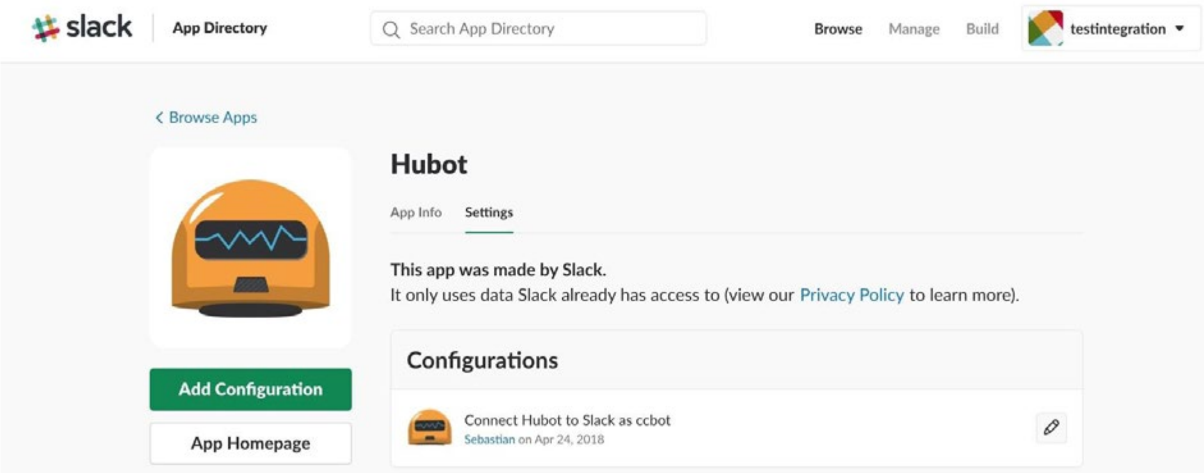

Figure 34: ClusterControl PostgreSQL Hubot

We enter the Username, which must match our bot name.

In the next window, we can see the API token to use.

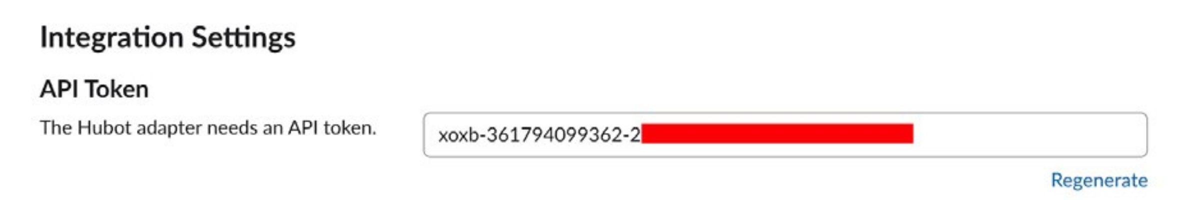

Figure 35: ClusterControl PostgreSQL API Token

```
Enter your slack API token: xoxb-111111111111-
XXXXXXXXXXXXXXXXXXXXXXXX
```

```
CCBot installation completed!
```

Finally, to be able to use all the s9s command line functions with CCBot, we must create a user from ClusterControl:

```
$ s9s user --create --cmon-user=cmon --group=admins  --con-
troller="https://localhost:9501" --generate-key cmon
```

We can now use our CCBot from Slack.

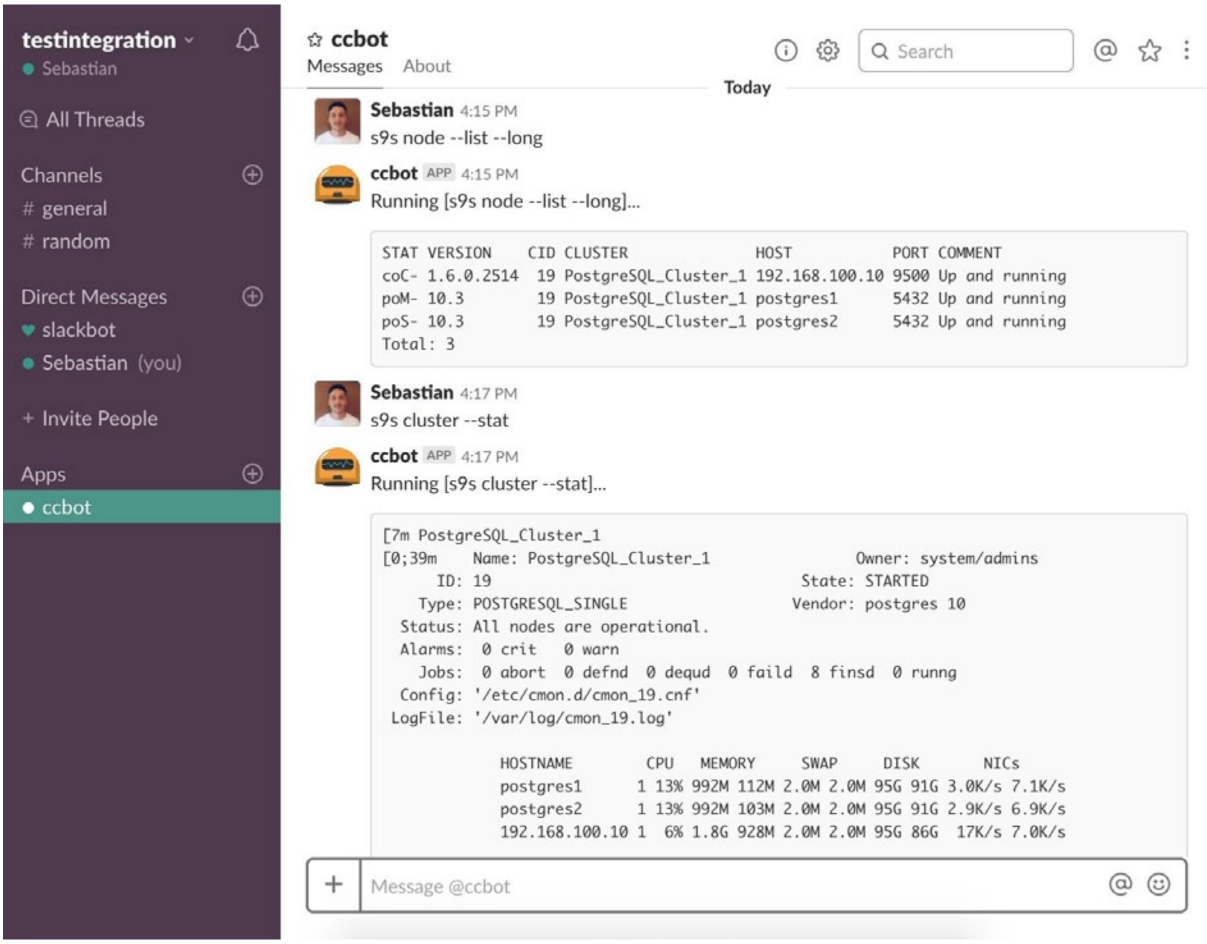

Figure 36: ClusterControl PostgreSQL CCBot

Some examples of commands:

```
$ s9s --help
```

```
$ s9s cluster --list --long
```

```
$ s9s node --list --long
```

```
$ s9s job --list
```

```
$ s9s backup --create --backup-method=<backup meth-
od> --cluster-id=<cluster id> --nodes=<list of node:port>
--backup-directory=<backup directory>
```

If we try to use CCBot from a Slack channel, we must add “@ccbot_name” at the beginning of our command:

```
@ccbot s9s backup --create --backup-method=xtrabackupfull
--cluster-id=1 --nodes=10.0.0.5:3306 --backup-directory=/
storage/backups
```

The CCBot makes it easier for the users, mainly the ones that are not very used to work with the command line, to handle ClusterControl, as it is fully integrated with the tools they handle on a daily basis.

> **Note**
>
> If we have the following error when wanting to run the CCBot installer in our ClusterControl:
>
> ```
> $ s9s job --list
> ```
>
> We must update the version of nodejs package.

7.1. Command Line

ClusterControl includes a tool called s9s, which allows us to perform administration tasks, monitoring, implementation, and several tasks that we have already seen, from the command line. In this way, we can easily integrate ClusterControl with the automation tools that we currently have, such as Puppet or Chef, without the need of using the UI.

Next we will see some examples of tasks that we can perform with this tool.

7.1.1. Help

```
$ s9s --help

Usage:
  s9s COMMAND [OPTION...]

Where COMMAND is:
  account - to manage accounts on clusters.
   backup - to view, create and restore database backups.
  cluster - to list and manipulate clusters.
      job - to view jobs.
    maint - to view and manipulate maintenance periods.
 metatype - to print metatype information.
     node - to handle nodes.
  process - to view processes running on nodes.
   script - to manage and execute scripts.
   server - to manage hardware resources.
     user - to manage users.

Generic options:
  --help                        Show help message and exit.
  -v, --verbose                 Print more messages than nor-
mally.
```

```
  -V, --version                Print version information and exit.
  -c, --controller=URL         The URL where the controller is found.
  -P, --controller-port INT    The port of the controller.
  --rpc-tls                    Use TLS encryption to controller.
  -u, --cmon-user=USERNAME     The username on the Cmon system.
  -p, --password=PASSWORD      The password for the Cmon user.
  --private-key-file=FILE     The name of the file for authentication.

Formatting:
  --batch                      No colors, no human readable, pure data.
  --color=always|auto|never    Sets if colors should be used in the output.
  --config-file=PATH         Set the configuration file.
  --date-format=FORMAT         The format of the dates printed.
  -l, --long                   Print the detailed list.
  --no-header                  Do not print headers.
  --only-ascii                 Do not use UTF8 characters.
  --print-json                 Print the sent/received JSon messages.

Job related options:
  --log                        Wait and monitor job messages.
  --recurrence=CRONTABSTRING Timing information for recurring jobs.
  --schedule=DATE&TIME         Run the job at the specified time.
  --timeout=SECONDS            Timeout value for the entire job.
  --wait                       Wait until the job ends.
```

7.1.2. PostgreSQL deploy cluster

The following command deploys a setup of 3 PostgreSQL nodes in version 10. The name of the new cluster will be Postgres_S9S.

```
$ s9s cluster --create --cluster-type=postgresql --vendor=’default’ --nodes=”10.0.0.11;10.0.0.12;10.0.0.13” --provider-version=10 --db-admin-passwd=’pa$$word’ --os-user=root --cluster-name=’Postgres_S9S’ --wait

Creating PostgreSQL Cluster
Job 3564 RUNNING    [█         ]  15% Installing helper packages
```

7.1.3. PostgreSQL create backup

The following command creates a backup with the pgdump method, from node 10.0.0.11, from our Postgres_S9S cluster and saves it in / root / backups.

```
$ s9s backup --create --backup-method=pgdump --clus-
ter-name='Postgres_S9S' --nodes=10.0.0.11 --backup-directo-
ry=/root/backups/ --wait

Create pgdump Backup
Job 3568 RUNNING3    [          █ ] ---% Job is running
```

7.1.4. PostgreSQL cluster status

This command shows us the status of the Postgres_S9S cluster.

```
Postgres_S9S
    Name: Postgres_S9S                    Owner: cmon/admins
      ID: 15                              State: STARTED
    Type: POSTGRESQL_SINGLE              Vendor: postgres 10
  Status: All nodes are operational.
  Alarms:  0 crit   0 warn
    Jobs:  0 abort  0 defnd  0 dequd  0 faild  3 finsd  0 runng
  Config: '/etc/cmon.d/cmon_15.cnf'
 LogFile: '/var/log/cmon_15.log'

                                                  HOSTNAME       CPU   MEMORY    SWAP      DISK      NICs
                                                  192.168.100.41 1  8% 992M 109M 2.0M 2.0M 95G 91G 2.4K/s 5.6K/s
                                                  192.168.100.42 1  8% 992M 110M 2.0M 2.0M 95G 91G 2.3K/s 5.4K/s
                                                  192.168.100.43 1  8% 992M 110M 2.0M 2.0M 95G 91G 2.4K/s 5.6K/s
                                                  192.168.100.10 1 17% 1.8G 770M 2.0M 2.0M 95G 86G  20K/s  35K/s
```

Figure 37: ClusterControl PostgreSQL s9s Cluster Status

We can also see a list of all our created clusters:

```
$ s9s cluster --list --long
```

7.1.5. Jobs status

With the following commands, we can list our current jobs, as well as view the status and the log for each one.

```
$ s9s job --list
ID    CID    STATE        OWNER    GROUP    CREATED    RDY
TITLE
3563    0    FAILED        cmon    admins    19:40:30
7%    Creating PostgreSQL Cluster
3564    0    RUNNING    cmon        admins     19:50:10
15%     Creating PostgreSQL Cluster
```

```
$ s9s job --log --job-id=3564
10.0.0.11:5432: Installing new node.
10.0.0.11:5432: Setting SELinux in permissive mode.
10.0.0.11:5432: Disabling firewall.
10.0.0.11:5432: Tuning OS parameters.
10.0.0.11:5432: Setting vm.swappiness = 1.
10.0.0.11:5432: Installing helper packages.
10.0.0.11: Upgrading nss.
10.0.0.11: Upgrading ca-certificates.
10.0.0.11: Installing net-tools.
```

```
$ s9s job --wait --job-id=3564
Creating PostgreSQL Cluster
Job 3564 RUNNING    [█         ]  15% Installing helper packages
```

You can check the list of available commands in the documentation.

Conclusion

In this Whitepaper, we went though some of the challenges that may arise when administering a PostgreSQL database. We reviewed some of the most important tasks that an administrator need to handle, and we included some detailed links on how to effectively handle each of them.

From this it is apparent that, without a centralized tool, we will need to use a number of differents tools, which can be time consuming and hard to manage. After reviewing these challenges we introduced ClusterControl as a single platform to automate these tasks. The aim of this was for you to be able to compare how much time and effort can be saved, as well as how the risks can be mitigated by the usage of a unified management platform.

About ClusterControl

ClusterControl is the all-inclusive open source database management system for users with mixed environments that removes the need for multiple management tools. ClusterControl provides advanced deployment, management, monitoring, and scaling functionality to get your MySQL, MongoDB, and PostgreSQL databases up-and- running using proven methodologies that you can depend on to work. At the core of ClusterControl is it's automation functionality that let's you automate many of the database tasks you have to perform regularly like deploying new databases, adding and scaling new nodes, running backups and upgrades, and more. Severalnines provides automation and management software for database clusters. We help companies deploy their databases in any environment, and manage all operational aspects to achieve high-scale availability.

About Severalnines

Severalnines provides automation and management software for database clusters. We help companies deploy their databases in any environment, and manage all operational aspects to achieve high-scale availability.

Severalnines' products are used by developers and administrators of all skills levels to provide the full 'deploy, manage, monitor, scale' database cycle, thus freeing them from the complexity and learning curves that are typically associated with highly available database clusters. Severalnines is often called the "anti-startup" as it is entirely self-funded by its founders. The company has enabled over 12,000 deployments to date via its popular product ClusterControl. Currently counting BT, Orange, Cisco, CNRS, Technicolor, AVG, Ping Identity and Paytrail as customers. Severalnines is a private company headquartered in Stockholm, Sweden with o ces in Singapore, Japan and the United States. To see who is using Severalnines today visit:

https://www.severalnines.com/company

Deploy Manage Monitor Scale

Related Resources

ClusterControl for PostgresSQL

ClusterControl provides management and monitoring for PostgreSQL deployments including replication and configuration management.

Learn more

Become a PostgreSQL DBA Blog Series

Read our popular blog series on how to become a PostgreSQL DBA: we cover everything from deployment and monitoring via management through to scaling your PostgreSQL database setups.

Read blogs

Become a ClusterControl DBA Blog Series

Learn everything you need to know about ClusterControl, the all-inclusive database management system for open source databases, and how to best administer open source databases.

Read blogs

This white paper discusses some of the challenges that may arise when administering a PostgreSQL database as well as some of the most important tasks an administrator needs to handle; and how to do so effectively ... with ClusterControl. See how much time and effort can be saved, as well as risks mitigated, by the usage of such a unified management platform.

Deploy

Manage

Monitor

Scale