Backup Admin Guide

In order to maintain a backup server, we created a VM via the Duke CoLab, which runs an Ubuntu 14.4 server.

Step 1: Set-up and configure 2 separate consoles for main and backup servers

Use the terminal to open two different shells: one for navigating the main development server and the other for navigating the main backup server (via the VM).

In the **backup server shell** run:

```
ssh bitnami@colab-sbx-239.oit.duke.edu
```

When prompted for a password, enter:

```
do4ievetAl
```

Navigate to the inventory_project/mysite/ using the command:

```
cd inventory_project/mysite/
```

Now, install and launch a virtual environment using the commands:

```
pip install virtualenv
virtualenv myvenv
source myvenv/bin/activate
```

In **both shells**, run the following command in order to determine the host server's IP address:

```
ip addr show eth0
```

Step 2: Initialize PG configuration files for Main server

In the **main development server shell**, update the **pg_hba.conf** file, by running the following command:

```
sudo nano pg_hba.conf
```

Add the following line to the **pg_hba.conf** file:

```
$ host    all     all     barman-backup-server-ip/24       trust
```

***NOTE:** barman-backup-server-ip is the INET IP address for the backup server:

Step 3: Install Barman on Backup server

In the **backup server shell**, install the Barman package, by running the following command:

```
sudo apt-get install barman
```

Step 4: Configure SSH Connectivity Between Main Development Server and Backup Server

First, we need to set the SSH connectivity from the main server to the backup server. Thus, in the **main development server shell**, switch to the user postgres, and then generate a SSH key-pair, by running:

```
sudo -su postgres
ssh-keygen -t rsa
```

Accept the default location by clicking ENTER and then set the key-pair's passphrase to be empty by clicking ENTER two more times. This will now save the generated keys in a .ssh file within the postgres user's home directory.

Extract the generated keys from the .ssh files, using the following commands:

```
cat ~/.ssh/bin/id_rsa.pub
```

In the **Backup server shell**, switch the user to be 'barman', by running:

```
sudo -su barman
```

Now, copy the key contents generated from the Main development server to a new .ssh directory stored on the Backup server, by running:

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
```

```
echo "public_key_string" >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

***NOTE:** "public_key_string" is the extracted key from the Main development server's .ssh files (result of the previous 'cat' command)!

To test the connection between the two servers, **switch** back to the **Main development server shell** and run the following command:

```
ssh barman@barman-backup-server-ip
```

If successful, type 'exit' and disconnect from the backup server!

Step 5: Configure SSH Connectivity Between Backup Server and Main Development Server

REPEAT the procedure from Step 4, this time establishing an SSH connection from the backup server to the main development server!

Be sure that when checking the connection between the two servers, **switch** back to the **Backup server shell** and run the following command:

```
ssh postgres@main-db-server-ip
```

***NOTE:** main-db-server-ip is the INET IP address for the main development server:

If successful, type 'exit' and disconnect from the main development server!

Step 6: Configuring Barman for Backups

In the Backup server shell, switch users to 'bitnami', by typing the following:

```
su - bitnami
ENTER PASSWORD
```

Now, open the Barman configuration file, by running:

```
sudo nano /etc/barman.conf
```

Add the following settings to the barman configuration file:

```
[barman]
barman_home = /var/lib/barman

. . .

barman_user = barman
log_file = /var/log/barman/barman.log
compression = gzip
reuse_backup = link

. . .

immediate_checkpoint = true

. . .

basebackup_retry_times = 3
basebackup_retry_sleep = 30
```

```
last_backup_maximum_age = 1 DAYS

. . .

[main-db-server]
description = "Main DB Server"
ssh_command = ssh postgres@main-db-server-ip
conninfo = host=main-db-server-ip user=postgres
retention_policy_mode = auto
retention_policy = RECOVERY WINDOW OF 12 MONTHS
wal_retention_policy = main
```
***NOTE:** main-db-server-ip is the INET IP address for the main development server:

Now, save and exit the file!

Step 7: Configuring Main Development Server for Backups

In the **Backup server shell**, switch to the user 'barman' and find the incoming backup directory, by running:
```
sudo -su barman
barman show-server main-db-server | grep incoming_wals_directory
```

Take note of the **incoming_wals_directory**! Be sure to save this filepath somewhere, so that it can be referenced later.

In the **Main development server shell**, switch to the user 'postgres' and open the PostgreSQL configuration file by running:
```
sudo -su postgres
nano $PGDATA/postgresql.conf
```

Add the following settings to the PostgreSQL configuration file:

```
wal_level = archive                 # minimal, archive, hot_standby, or logical

. . .

archive_mode = on              # allows archiving to be done

. . .

archive_command = 'rsync -a %p barman@barman-backup-server-ip:incoming_wals_directory/%f'              # 
command to use to archive a logfile segment
```

***NOTE:** Here, barman-backup-server-ip is the INET IP address of the backup server and the 'incoming_wals_directory' is the saved filepath of the incoming backup directory!

On the **Main development server shell**, switch back to the sudo user and restart the DB, by running:
```
sudo service postgresql restart
```

Step 8: Testing Barman

In the **Backup server shell**, switch to the user 'barman' and test the connection to the main development server, by running the following commands:
```
sudo -su barman
barman check main-db-server
```

If the connection is successfully established and all configurations are set properly, the output should look like this:

```
Server main-db-server:
    PostgreSQL: OK
    archive_mode: OK
    wal_level: OK
    archive_command: OK
    continuous archiving: OK
    directories: OK
    retention policy settings: OK
    backup maximum age: FAILED (interval provided: 1 day, latest backup age: No available backups)
    compression settings: OK
    minimum redundancy requirements: OK (have 0 backups, expected at least 0)
    ssh: OK (PostgreSQL server)
    not in recovery: OK
```

***NOTE:** Don't worry about FAILING the backup_maximum_age! If any other parameters FAIL, investigate the issue and fix it before moving on!

Step 9: Creating a Backup

In the **Backup server shell**, run the following command to backup the main development server:

```
barman backup main-db-server
```

To find the list of all the current backups, run the following command:

```
barman list-backup main-db-server
```

An **example output** of this, would look something like:

```
main-db-server 20151111T051954 - Wed Nov 11 05:19:46 2015 - Size: 26.9 MiB - WAL Size: 0 B
```

***NOTE:** The string colored in RED represents the 'backup_id'!

Step 10: Restoring From a Backup

In the **Main development server shell,** switch to the sudo user and stop the PG service by running the following:

```
sudo service postgresql stop
```

Switch to the **Backup server shell** and locate the details of the latest backup, by running:

```
barman show-backup main-db-server latest
```

The output should look something like this:

```
Backup 20160114T173552:
 Server Name            : main-db-server
 Status                 : DONE
 PostgreSQL Version     : 90405
 PGDATA directory       : /var/lib/pgsql/9.4/data

 Base backup information:

. . .

   Begin time           : 2016-01-14 17:35:53.164222-05:00
   End time             : 2016-01-14 17:35:55.054673-05:00
```

Next, run the following command, to restore the backup from the Backup server to the Main development server:

```
barman recover --target-time "Begin time"  --remote-ssh-command "ssh postgres@main-db-server-ip"
main-db-server   backup-id   /var/lib/postgresql/9.3/main
```

***NOTE:** 'Main-db-server-ip' is the INET IP address of the main development server. Replace 'Begin Time' and 'backup_id' with the values from the previous output. In the case that you are interested in restoring from the last backup, you can replace the 'backup_id' with the string 'latest'!

On the **Main development server shell,** switch the to the sudo user and start the PG service, by running:

```
sudo service postgresql start
```

Switch the user to be 'postgres' and run the following commands to check if restoration was successful:

```
sudo su - postgres
psql 'db_name'
```

***NOTE:** In this case, the 'db_name' is the name given to the DB when set up on the main development server!

Run the following commands to find the DB table contents:

```
db_name=# \dt
```

Finally, compare the newly restored DB tables to the latest tables within the Backup server's DB. If these are the same, then you have successfully restored the latest backup data to the main development server's database!

Step 11: Scheduling Backups

In order to schedule automatic backups, we need to create a crontab file, which periodically runs shell commands necessary for backing up our system. In order to have an automated backup system that backs up the database 7 times a week, 4 times a month, and 12 times a year, we had to create a complicated cron file. Please see the Appendix, in order to view the full crontab file!

Appendt ix:

Crontab File:

```
# Edit this file to introduce tasks to be run by cron.
0 12 * * * /usr/bin/barman backup main-db-server
1 12 * * * cp -R $(/usr/bin/barman list-files main-db-server latest | grep 'backup.info' | sed -e "s/backup.info$//")
~/daily/
2 12 * * * (/usr/bin/barman list-files main-db-server latest | grep 'backup.info' | sed -e "s/\/backup.info$//" -e
"s/^.*base\///") >> ~/daily/schedule.txt
3 12 * * * rm -R "$(head -1 ~/daily/schedule.txt)"
4 12 * * * echo "$(tail -n +2 ~/daily/schedule.txt)" >> ~/daily/schedule.txt
15 12 * * 0 cp -R $(/usr/bin/barman list-files main-db-server latest | grep 'backup.info' | sed -e "s/backup.info$//")
~/weekly/
16 12 * * 0 (/usr/bin/barman list-files main-db-server latest | grep 'backup.info' | sed -e "s/\/backup.info$//" -e
"s/^.*base\///") >> ~/weekly/schedule.txt
17 12 * * 0 rm -R "$(head -1 ~/weekly/schedule.txt)"
18 12 * * 0 echo "$(tail -n +2 ~/weekly/schedule.txt)" >> ~/weekly/schedule.txt
30 12 1 * * cp -R $(/usr/bin/barman list-files main-db-server latest | grep 'backup.info' | sed -e "s/backup.info$//")
~/monthly/
31 12 1 * * (/usr/bin/barman list-files main-db-server latest | grep 'backup.info' | sed -e "s/\/backup.info$//" -e
"s/^.*base\///") >> ~/monthly/schedule.txt
32 12 1 * * rm -R "$(head -1 ~/monthly/schedule.txt)"
33 12 1 * * echo "$(tail -n +2 ~/monthly/schedule.txt)" >> ~/monthly/schedule.txt
5 12 * * * /usr/bin/barman delete main-db-server oldest
* * * * * /usr/bin/barman cron
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
~
```