Title: One Way To: Backup a PSQL database to AWS S3 using Linux cron or Kubernetes CronJob

Description: Perform scheduled rotating encrypted backups of a PostgreSQL (PSQL) database to AWS S3, in a virtual machine using Linux cron, or Kubernetes cluster using CronJob.

## ## Intro

The argument for spending time setting up database backups is similar to that of paying for insurance. It seems like a waste at the outset, and it is easy to put off, however, when things go wrong, you will wish you had it.

This article describes a way to set up scheduled rotating encrypted backups of a PostgreSQL database to AWS S3, in two typical deployment environments:

- a virtual machine scheduled using Linux cron

- a Kubernetes cluster scheduled using CronJob

You can find a GitHub repository containing the script and template environment variables and Kubernetes config here: [cpcwood/psql-backup-s3](https://github.com/cpcwood/psql-backup-s3)

## ## Backup Strategy

A database backup and restore strategy should meet a few criteria:

- \*\*Separate remote storage\*\* - One of the main reasons for database backups is the mitigation of media failure, natural disaster, and user error. It tends to be challenging and costly to recover data in these circumstances, so backups must be kept off-site in a backup destination with reliable storage. A common cloud backup destination is Amazon's [AWS S3](https://aws.amazon.com/s3/) service.

- \*\*Automation\*\* - Backups should be made automatically and on regular intervals. Most systems have a built-in job scheduler such as Linux's ```cron``` and Kubernetes' CronJob.

- \*\*Encryption\*\* - Database backups are usually dumps of SQL, used to rebuild the database structure and data. Since these dumps are typically plain text, to prevent sensitive data from being leaked, database backups should be encrypted while being stored in the backup destination. [GnuPG](https://gnupg.org/) (GPG) is a commonly used implementation of the PGP cryptographic software suite, which can encrypt database backups.

- \*\*Rotation\*\* - Off-site storage costs. Therefore, optimise resource requirements by keeping the number of stored backups to a cost-effective level. One way of doing this is backup rotation. While there are plenty of different [backup rotation schemes](https://gomindsight.com/insights/blog/data-backup-schemes/), 'first in first out' scheme, where old backups are removed as new ones are added, is effective and easy to script for s3 storage.

## ## Setup AWS

### ### Create an AWS S3 Bucket

Create a private Amazon AWS S3 bucket to store your database backups: [AWS 'create bucket' guide](https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html).

### ### Create IAM User

Create an IAM user in your AWS account with access to the S3 bucket created above: [AWS 'create user' guide](https://docs.aws.amazon.com/IAM/latest/UserGuide/id\_users\_create.html)

The script requires, list, put, and delete access on the s3 bucket. So, the S3 policy JSON attached to the IAM user might look like:

```json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:PutObject",

"s3:ListBucket",

"s3:DeleteObject"

],

"Resource": [

"arn:aws:s3:::<bucket-name>/\*",

"arn:aws:s3:::<bucket-name>"

]

}

]

}

```

Make sure to download or keep hold of the new user security credentials so you can add them to the backup script environment later.

## ## Create PGP Keys

On a separate (ideally air-gapped) machine, install [GPG](<https://gnupg.org/>) so encryption keys can be generated:

```sh

apt install gnupg

```

Then create a pair of public and private encryption keys. Using public-key cryptography to encrypt the backup on the server will help prevent the database backup from being compromised if the environment variables are leaked.

- Generate a keypair using your email for ID: ```gpg --gen-key```

- Export the public key: ```gpg --armor --export <your-email>```

- Export the secret key and move to [secure storage](<https://lwn.net/Articles/734767/>).

## ## Create Backup Script

### ### Script Requirements

The backup program will need to do the following:

- Check required config is present

- Create a temporary backup directory

- Create database backup

- Compress backup

- Encrypt backup

- Upload backup to off-site storage (AWS S3)

- Remove expired backups from off-site storage

- Cleanup

### ### Write the Script

Since the program needs to interact with several other CLIs, such as ```psql``` and ```aws-cli```, a shell script would be suitable. Make a shell script to meet the above requirements, for example:

```sh

vim ~/psql-backup-s3.sh

```

```sh

#! /bin/sh

# PSQL Database Backup to AWS S3

echo "Starting PSQL Database Backup..."

# Ensure all required environment variables are present

if [ -z "$GPG\_KEY" ] || \

[ -z "$GPG\_KEY\_ID" ] || \

[ -z "$POSTGRES\_PASSWORD" ] || \

[ -z "$POSTGRES\_USER" ] || \

[ -z "$POSTGRES\_HOST" ] || \

[ -z "$POSTGRES\_DB" ] || \

[ -z "$AWS\_ACCESS\_KEY\_ID" ] || \

[ -z "$AWS\_SECRET\_ACCESS\_KEY" ] || \

[ -z "$AWS\_DEFAULT\_REGION" ] || \

[ -z "$S3\_BUCKET" ]; then

>&2 echo 'Required variable unset, database backup failed'

exit 1

fi

# Make sure local bin in path

export PATH=/usr/local/bin:$PATH

# Import gpg public key from env

echo "$GPG\_KEY" | gpg --batch --import

# Create backup params

backup\_dir=$(mktemp -d)

backup\_name=$POSTGRES\_DB'--'$(date +%d'-'%m'-'%Y'--'%H'-'%M'-'%S).sql.bz2.gpg

backup\_path="$backup\_dir/$backup\_name"

# Create, compress, and encrypt the backup

PGPASSWORD=$POSTGRES\_PASSWORD pg\_dump -d "$POSTGRES\_DB" -U "$POSTGRES\_USER" -h "$POSTGRES\_HOST" | bzip2 | gpg --batch --recipient "$GPG\_KEY\_ID" --trust-model always --encrypt --output "$backup\_path"

# Check backup created

if [ ! -e "$backup\_path" ]; then

echo 'Backup file not found'

exit 1

fi

# Push backup to S3

aws s3 cp "$backup\_path" "s3://$S3\_BUCKET"

status=$?

# Remove tmp backup path

rm -rf "$backup\_dir"

# Indicate if backup was successful

if [ $status -eq 0 ]; then

echo "PSQL database backup: '$backup\_name' completed to '$S3\_BUCKET'"

# Remove expired backups from S3

if [ "$ROTATION\_PERIOD" != "" ]; then

aws s3 ls "$S3\_BUCKET" --recursive | while read -r line; do

stringdate=$(echo "$line" | awk '{print $1" "$2}')

filedate=$(date -d"$stringdate" +%s)

olderthan=$(date -d"-$ROTATION\_PERIOD days" +%s)

if [ "$filedate" -lt "$olderthan" ]; then

filetoremove=$(echo "$line" | awk '{$1=$2=$3=""; print $0}' | sed 's/^[ \t]\*//')

if [ "$filetoremove" != "" ]; then

aws s3 rm "s3://$S3\_BUCKET/$filetoremove"

fi

fi

done

fi

else

echo "PSQL database backup: '$backup\_name' failed"

exit 1

fi

```

## ## Deploy

Depending on your deployment setup, there will be different ways to deploy the script to run the backup job regularly.

Here are the setup methods for two typical deployment types:

- Traditional VM - Linux cron

- Kubernetes - CronJob

## ## Traditional VM

### ### Copy Script to Machine

Copy the script to the virtual machine (VM) into a suitable directory and make sure it is executable for the user who will be running it:

```sh

chmod 744 ./psql-backup-s3/psql-backup-s3.sh

```

### ### Install Dependencies

Install the script dependencies on the VM:

- \*\*GPG\*\* - Install [GPG](https://gnupg.org/) to encrypt backup files: ```apt install gnupg```

- \*\*AWS-CLI\*\* - Install AWS CLI tool to transfer backup to AWS S3: see [AWS guide](https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html)

- \*\*date\*\* - Ensure date is GNU core utilities date, not included in alpine linux (busybox) by default: ```apt install coreutils```

### ### Linux cron

cron is a time-based job scheduler built into Linux which runs processes on the system at scheduled times.

### #### Config

The backup script gets its configuration from environment variables. The variables required are:

```sh

export ROTATION\_PERIOD= # days to keep backups (exclude to stop backups from deleting)

export POSTGRES\_PASSWORD= # PSQL PASSWORD

export POSTGRES\_USER= # PSQL USERNAME

export POSTGRES\_HOST= # postgres hostname

export POSTGRES\_DB= # name of database to backup

export AWS\_ACCESS\_KEY\_ID= # AWS IAM USER ID

export AWS\_SECRET\_ACCESS\_KEY= # AWS IAM USER KEY

export AWS\_DEFAULT\_REGION= # aws s3 bucket region

export S3\_BUCKET= # aws s3 bucket name

export GPG\_KEY\_ID= # email id used in gpg key generation

export GPG\_KEY= # exported public gpg key

```

cron jobs do not inherit the same environment as a job run from the command line. Instead, their default environment is from ```/etc/environment```, read more about why in [this IBM article](https://www.ibm.com/support/pages/cron-environment-and-cron-job-failures). Therefore, load the environment variables required for the backup script in the job definition.

One way to do this <i>source</i> a shell script using the ["dot" command](https://tldp.org/LDP/abs/html/special-chars.html#DOTREF) in the crontab.

First, create a script exporting the above environment variables on the VM.

```sh

vim ~/psql-backup-s3/psql-backup-s3.env

```

Since it contains credentials, ensure the only the crontab user can only access it:

```sh

chmod 700 ~/psql-backup-s3/psql-backup-s3.env

```

It can then be sourced before the backup job in the crontab, as shown below.

### #### Create the cron Job

Add a new cron job using crontab. The job should periodically load the environment variables and then run the backup script. For example, to run the backup daily at 3.30 am:

```sh

crontab -e

```

```sh

30 3 \* \* \* . $HOME/psql-backup-s3/psql-backup-s3.env && $HOME/psql-backup-s3/psql-backup-s3.sh 2>&1 | logger -t psql-backup-s3

```

For more info on setting up a job using crontab, checkout [ubuntu's guide here](https://help.ubuntu.com/community/CronHowto). [crontab guru](https://crontab.guru/) can be helpful for defining schedules.

Note: When setting up it can be useful to set the job schedule to a short interval such as ```\*/2 \* \* \* \*``` so you can check for any misconfiguration or errors.

## ## Kubernetes CronJob

[Kubernetes CronJob](https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/) is a built-in feature which allows jobs to be run in containers periodically.

### ### Config

Create a [Kubernetes Secret](https://kubernetes.io/docs/concepts/configuration/secret/) object to store the sensitive credentials for the backup script. For example:

```yaml

---

apiVersion: v1

kind: Secret

metadata:

name: psql-backup-s3

data:

POSTGRES\_USER: # PSQL USERNAME

POSTGRES\_PASSWORD: # PSQL PASSWORD

AWS\_ACCESS\_KEY\_ID: # AWS IAM USER ID

AWS\_SECRET\_ACCESS\_KEY: # AWS IAM USER KEY

---

```

Create a [Kubernetes ConfigMap](https://kubernetes.io/docs/concepts/configuration/configmap/) object to store the non-sensitive configuration details for the script. For example:

```yaml

---

apiVersion: v1

kind: ConfigMap

metadata:

name: psql-backup-s3

data:

ROTATION\_PERIOD: # days to keep backups (exclude to stop backups from deleting)

POSTGRES\_HOST: # postgres k8s service hostname

POSTGRES\_DB: # name of database to backup

AWS\_DEFAULT\_REGION: # aws s3 bucket region

S3\_BUCKET: # aws s3 bucket name

GPG\_KEY\_ID: # email id used in gpg key generation

GPG\_KEY: # exported public gpg key

---

```

Make sure to [apply](https://kubernetes.io/docs/tasks/manage-kubernetes-objects/) the newly created secret and config objects to your cluster in the correct namespace.

### ### Create the CronJob

Create a [Kubernetes CronJob](https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/) object to run the backup job on a schedule. The container image used in the job must have the required dependencies and backup script included. An example Dockerfile can be found in the GitHub repository [cpcwood/psql-backup-s3](https://github.com/cpcwood/psql-backup-s3) and respective image can be pulled from dockerhub under the repository [```cpcwood/psql-backup-s3```](<https://hub.docker.com/r/cpcwood/psql-backup-s3>). For example, to run the backup daily at 3.30 am:

```yaml

---

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: psql-backup-s3

spec:

schedule: "30 3 \* \* \*"

jobTemplate:

spec:

template:

spec:

containers:

- name: psql-backup-s3-container

image: cpcwood/psql-backup-s3:latest

envFrom:

- configMapRef:

name: psql-backup-s3

env:

- name: POSTGRES\_USER

valueFrom:

secretKeyRef:

name: psql-backup-s3

key: POSTGRES\_USER

- name: POSTGRES\_PASSWORD

valueFrom:

secretKeyRef:

name: psql-backup-s3

key: POSTGRES\_PASSWORD

- name: AWS\_ACCESS\_KEY\_ID

valueFrom:

secretKeyRef:

name: psql-backup-s3

key: AWS\_ACCESS\_KEY\_ID

- name: AWS\_SECRET\_ACCESS\_KEY

valueFrom:

secretKeyRef:

name: psql-backup-s3

key: AWS\_SECRET\_ACCESS\_KEY

restartPolicy: Never

---

```

Note: When setting up it can be useful to set the job schedule to a short interval such as ```\*/3 \* \* \* \*``` so you can check for any misconfiguration or errors.

[Apply](https://kubernetes.io/docs/tasks/manage-kubernetes-objects/) the CronJob object to your cluster.

The job should now run the backup script periodically as scheduled in the object definition.

## ## Restore

To restore a backup:

- Download the encrypted database dump from aws S3

- Copy to the machine containing the private gpg key

- Decrypt downloaded file using gpg: ```gpg --output <decrypted file name>.sql.bz2 --decrypt <downloaded file name>.sql.bz2.gpg```

- Move to server hosting PostgreSQL database

- Unzip decrypted file using bzip: ```bzip2 -d <decrypted file name>.sql.bz2```

- Restore the database dump using the ```psql``` command, for details see the documentation on [backup dumps](https://www.postgresql.org/docs/current/backup-dump.html) for your version of PostgreSQL.

## ## Conclusion

The backup script will now be scheduled to run on regular intervals as defined by your crontab or CronJob.

There are plenty of additional features which could be added to the script in the future such as success notifications using [AWS SNS](https://aws.amazon.com/sns/), or more complex backup rotation scheme such as [Grandfather-father-son](https://en.wikipedia.org/wiki/Backup\_rotation\_scheme).

### ### Footnote

Drop me a message through the contact section of this site or via LinkedIn if you have any comments, suggested changes, or see any bugs.