restic-compose-backup

Release 0.6.0

May 27, 2020

Contents:

Simple backup with restic for docker-compose setups.

CHAPTER 1

Introduction

1.1 Install

restic-compose-backup is available at docker docker hub.

```
docker pull restic-compose-backup
```

Optionally it can be built from source using the github repository.

```
git clone https://github.com/ZettaIO/restic-compose-backup.git
cd restic-compose-backup
# Build and tag the image locally
docker build src/ --tag restic-compose-backup
```

1.2 Bug reports and issues

Please report bugs an issues on github

1.3 Development setup

Getting started with local development is fairly simple. The github repository contains a simple `docker-compose.yaml`

```
docker-compose up -d
# Enter the container in sh
docker-compose run --rm backup sh
```

The dev compose setup maps in the source from the host and the spawned backup container will inherit all the volumes from the backup service ensuring code changes propagates during development.

Set up a local venv and install the package in development mode:

```
python -m venv .venv
. .venv/bin/activate
pip install -e ./src
```

CHAPTER 2

Configuration

2.1 Environment Variables

2.1.1 RESTIC_REPOSITORY

Sets the restic repository path.

This is a standard environment variable the `restic` command will read making it simple for us to enter the container and use the restic command directly.

More about this value and supported backends: https://restic.readthedocs.io/en/stable/030_preparing_a_new_repo.html

2.1.2 RESTIC_PASSWORD

Sets the password is used to encrypt/decrypt data. Losing this password will make recovery impossible.

This is a standard environment variable the `restic` command will read making it simple for us to enter the container running the command directly.

2.1.3 RESTIC_KEEP_DAILY

Default value: `7`

How many daily snapshots (grouped by path) back in time we want to keep. This is passed to restic in the `forget --keep-daily` option.

2.1.4 RESTIC_KEEP_WEEKLY

Default value: `4`

How many weeks back we should keep at least one snapshot (grouped by path). This is passed to restic in the `forget --keep-weekly` option.

2.1.5 RESTIC_KEEP_MONTHLY

Default value: `12`

How many months back we should keep at least on snapshot (grouped by path). This is passed to restic in the `forget --keep-monthly` option.

The schedule parameters only accepts numeric values and is validated when the container starts. Providing values cron does not understand will stall all backup.

2.1.6 RESTIC_KEEP_YEARLY

Default value: `3`

How many years back we should keep at least one snapshot (grouped by path). This is passed to restic in the `forget --keep-yearly` option.

2.1.7 CRON_SCHEDULE

Default value: `0 2 * * *` (daily at 02:00)

The cron schedule parameters. The crontab is generated when the container starts from the `CRON_SCHEDULE` and `CRON_COMMAND` env variables.

```
┌───────────── minute (0 - 59)
│ ┌───────────── hour (0 - 23)
│ │ ┌───────────── day of the month (1 - 31)
│ │ │ ┌───────────── month (1 - 12)
│ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday)
│ │ │ │ │
│ │ │ │ │
│ │ │ │ │
* * * * * command to execute
```

2.1.8 CRON_COMMAND

Default value: `source /env.sh && rcb backup > /proc/1/fd/1`

The command executed in the crontab. A single line is generated when the container starts from the `CRON_SCHEDULE` and `CRON_COMMAND` environment variables.

The default command sources a dump of all env vars, runs the backup command and directs output to pid 1 so it appears in docker logs.

By default the crontab will look like this:

```
0 2 * * * source /env.sh && rcb backup > /proc/1/fd/1
```

2.1.9 LOG_LEVEL

Default value: `info`

Log level for the `rcb` command. Valid values are `debug`, `info`, `warning`, `error`.

2.1.10 EMAIL_HOST

The email host to use.

Alerts can be tested using the `rcb alerts` command. This will send a test message to all configured alert backends.

2.1.11 EMAIL_PORT

The port to connect to

Alerts can be tested using the `rcb alerts` command. This will send a test message to all configured alert backends.

2.1.12 EMAIL_HOST_USER

The user of the sender account

Alerts can be tested using the `rcb alerts` command. This will send a test message to all configured alert backends.

2.1.13 EMAIL_HOST_PASSWORD

The password for the sender account

Alerts can be tested using the `rcb alerts` command. This will send a test message to all configured alert backends.

2.1.14 EMAIL_SEND_TO

The email address to send alerts

Alerts can be tested using the `rcb alerts` command. This will send a test message to all configured alert backends.

2.1.15 DISCORD_WEBHOOK

The discord webhook url. And administrator can quickly set this up by going to server settings in the discord client and create a webhook that will post embedded messages to a specific channel.

The url usually looks like this: `https://discordapp.com/api/webhooks/...`\`

2.1.16 DOCKER_HOST

Default value: `unix://tmp/docker.sock`

The socket or host of the docker service.

2.1.17 DOCKER_TLS_VERIFY

If defined verify the host against a CA certificate. Path to certs is defined in `DOCKER_CERT_PATH` and can be copied or mapped into this backup container.

2.1.18 DOCKER_CERT_PATH

A path to a directory containing TLS certificates to use when connecting to the Docker host. Combined with `DOCKER_TLS_VERIFY` this can be used to talk to docker through TLS in cases were we cannot map in the docker socket.

2.1.19 INCLUDE_PROJECT_NAME

Define this environment variable if your backup destination paths needs project name as a prefix. This is useful when running multiple projects.

2.1.20 EXCLUDE_BIND_MOUNTS

Docker has to volumes types. Binds and volumes. Volumes are docker volumes (`docker`volume list`). Binds are paths mapped into the container from the host for example in the `volumes` section of a service.

If defined all host binds will be ignored globally. This is useful when you only care about actual docker volumes. Often host binds are only used for mapping in configuration. This saves the user from manually excluding these bind volumes.

2.1.21 SWARM_MODE

If defined containers in swarm stacks are also evaluated.

2.2 Compose Labels

What is backed up is controlled by simple labels in the compose yaml file. At any point we can verify this configuration by running the `rcb status` command.

Here we can see what volumes and databases are detected for backup.

2.2.1 Volumes

To enable volume backup for a service we simply add the *restic-compose-backup.volumes: true* label. The value must be `true`.

Example:

```
myservice:
  image: some_image
  labels:
    restic-compose-backup.volumes: true
  volumes:
    - uploaded_media:/srv/media
```

(continues on next page)

(continued from previous page)

```
    - uploaded_files:/srv/files
    - /srv/data:/srv/data

volumes:
  media:
  files:
```

This will back up the three volumes mounted to this service. Their path in restic will be:

- /volumes/myservice/srv/media
- /volumes/myservice/srv/files
- /volumes/myservice/srv/data

A simple *include* and *exclude* filter for what volumes should be backed up is also available. Note that this includes or excludes entire volumes and are not include/exclude patterns for files in the volumes.

Note: The `exclude` and `include` filtering is applied on the source path, not the destination.

Include example including two volumes only:

```
myservice:
  image: some_image
  labels:
    restic-compose-backup.volumes: true
    restic-compose-backup.volumes.include: "uploaded_media,uploaded_files"
  volumes:
    - uploaded_media:/srv/media
    - uploaded_files:/srv/files
    - /srv/data:/srv/data

volumes:
  media:
  files:
```

Exclude example achieving the same result as the example above.

```
example:
  image: some_image
  labels:
    restic-compose-backup.volumes: true
    restic-compose-backup.volumes.exclude: "data"
  volumes:
    # Excluded by filter
    - media:/srv/media
    # Backed up
    - files:/srv/files
    - /srv/data:/srv/data

volumes:
  media:
  files:
```

The `exclude` and `include` tag can be used together in more complex situations.

2.2.2 mariadb

To enable backup of mariadb simply add the `restic-compose-backup.mariadb: true` label.

Credentials are fetched from the following environment variables in the mariadb service. This is the standard when using the official mariadb image.

```
MYSQL_USER
MYSQL_PASSWORD
```

Backups are done by dumping all databases directly into restic through stdin using `mysqldump`. It will appear in restic as a separate snapshot with path `/databases/<service_name>/all_databases.sql`.

Example:

```
mariadb:
  image: mariadb:10
  labels:
    restic-compose-backup.mariadb: true
  env_file:
    mariadb-credentials.env
  volumes:
    - mariadb:/var/lib/mysql

volumes:
  mariadb:
```

2.2.3 mysql

To enable backup of mysql simply add the `restic-compose-backup.mysql: true` label.

Credentials are fetched from the following environment variables in the mysql service. This is the standard when using the official mysql image.

```
MYSQL_USER
MYSQL_PASSWORD
```

Backups are done by dumping all databases directly into restic through stdin using `mysqldump`. It will appear in restic as a separate snapshot with path `/databases/<service_name>/all_databases.sql`.

Example:

```
mysql:
  image: mysql:5
  labels:
    restic-compose-backup.mysql: true
  env_file:
    mysql-credentials.env
  volumes:
    - mysql:/var/lib/mysql
```

volumes: mysql:

2.2.4 postgres

To enable backup of mysql simply add the `restic-compose-backup.postgres: true` label.

Credentials are fetched from the following environment variables in the postgres service. This is the standard when using the official postgres image.

```
POSTGRES_USER
POSTGRES_PASSWORD
POSTGRES_DB
```

Backups are done by dumping the `POSTGRES_DB` directly into restic through stdin using `pg_dump`. It will appear in restic as a separate snapshot with path `/databases/<service_name>/<POSTGRES_DB>.sql`.

Warning: Currently only the `POSTGRES_DB` database is dumped.

Example:

```
postgres:
  image: postgres:11
  labels:
    # Enables backup of this database
    restic-compose-backup.postgres: true
  env_file:
    postgres-credentials.env
  volumes:
    - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

CHAPTER 3

The *rcb* command

The `rcb` command is is basically what this entire project is. It provides useful commands interacting with the compose setup and restic.

The command can be executed inside the container or through `run`.

```
# Get the current status using run
$ docker-compose run --rm backup rcb status

# by entering the container
$ docker-compose exec backup sh
/restic-compose-backup # rcb status
```

Log level can be overridden by using the `--log-level` flag. This can help you better understand what is going on for example by using `--log-level debug`.

3.1 version

Displays the version.

Example output:

```
/restic-compose-backup # rcb version
0.4.0
```

3.2 status

Shows the general status of our setup. The command is doing the following operations

- Displays the name of the compose setup
- Displays the repository path

- Tells us if a backup is currently running
- Removes stale backup process containers if the exist
- Checks is the repository is initialized
- Initializes the repository if this is not already done
- Displays what volumes and databases are flagged for backup

Example output:

```
INFO: Status for compose project 'myproject'
INFO: Repository: '<restic repository>'
INFO: Backup currently running?: False
INFO: --------------- Detected Config ---------------
INFO: service: mysql
INFO:  - mysql (is_ready=True)
INFO: service: mariadb
INFO:  - mariadb (is_ready=True)
INFO: service: postgres
INFO:  - postgres (is_ready=True)
INFO: service: web
INFO:  - volume: media
INFO:  - volume: /srv/files
```

3.3 alert

Sends a test message to all configured alert backends and is there for you to verify that alerts are in fact working and configured correctly.

The format of this message:

```
subject: myproject: Test Alert
body: Test message
```

3.4 snapshots

Displays the latest snapshots in restic. This can also be done with `restic snapshots`.

Example output:

```
/restic-compose-backup # rcb snapshots
repository f325264e opened successfully, password is correct
ID        Time                 Host          Tags        Paths
---------------------------------------------------------------------------------------
↪-------
19928e1c  2019-12-09 02:07:44  b3038db04ec1              /volumes
7a642f37  2019-12-09 02:07:45  b3038db04ec1              /databases/mysql/all_
↪databases.sql
883dada4  2019-12-09 02:07:46  b3038db04ec1              /databases/mariadb/all_
↪databases.sql
76ef2457  2019-12-09 02:07:47  b3038db04ec1              /databases/postgres/test.sql
---------------------------------------------------------------------------------------
↪-------
4 snapshots
```

3.5 backup

Starts a backup process by spawning a new docker container. The network stack, mounted volumes, env vars etc. from the backup service are copied to this container.

We attach to this container and stream the logs and delete the container with the backup process is completed. If the container for any reason should not be deleted, it will be in next backup run as these containers are tagged with a unique label and detected.

If anything goes wrong the exist status of the container is non-zero and the logs from this backup run will be sent to the user through the configure alerts.

This command is by default called by cron every day at 02:00 unless configured otherwise. We can also run this manually is needed.

Running this command will do the following:

- Checks if a backup process is already running. If so, we alert the user and abort
- Gathers all the volumes configured for backup and starts the backup process with these volumes mounted into `/volumes`
- Checks the status of the process and reports to the user if anything failed

The backup process does the following:

- `status` is first called to ensure everything is ok
- Backs up `/volumes` if any volumes were mounted
- Backs up each configured database
- Runs `cleanup` purging snapshots based on the configured policy
- Checks the health of the repository

Example:

```
$ docker-compose exec backup sh
/restic-compose-backup # rcb backup
INFO: Starting backup container
INFO: Backup process container: loving_jepsen
INFO: 2019-12-09 04:50:22,817 - INFO: Status for compose project 'restic-compose-
↪backup'
INFO: 2019-12-09 04:50:22,817 - INFO: Repository: '/restic_data'
INFO: 2019-12-09 04:50:22,817 - INFO: Backup currently running?: True
INFO: 2019-12-09 04:50:23,701 - INFO: ------------------------- Detected Config ------
↪-------------------
INFO: 2019-12-09 04:50:23,701 - INFO: service: mysql
INFO: 2019-12-09 04:50:23,718 - INFO:  - mysql (is_ready=True)
INFO: 2019-12-09 04:50:23,718 - INFO: service: mariadb
INFO: 2019-12-09 04:50:23,726 - INFO:  - mariadb (is_ready=True)
INFO: 2019-12-09 04:50:23,727 - INFO: service: postgres
INFO: 2019-12-09 04:50:23,734 - INFO:  - postgres (is_ready=True)
INFO: 2019-12-09 04:50:23,735 - INFO: service: web
INFO: 2019-12-09 04:50:23,736 - INFO:  - volume: /some/volume
INFO: 2019-12-09 04:50:23,736 - INFO: ------------------------------------------------
↪-------------------
INFO: 2019-12-09 04:50:23,736 - INFO: Backing up volumes
INFO: 2019-12-09 04:50:24,661 - INFO: Backing up databases
INFO: 2019-12-09 04:50:24,661 - INFO: Backing up mysql in service mysql
INFO: 2019-12-09 04:50:25,643 - INFO: Backing up mariadb in service mariadb
```

(continues on next page)

(continued from previous page)

```
INFO: 2019-12-09 04:50:26,580 - INFO: Backing up postgres in service postgres
INFO: 2019-12-09 04:50:27,555 - INFO: Forget outdated snapshots
INFO: 2019-12-09 04:50:28,457 - INFO: Prune stale data freeing storage space
INFO: 2019-12-09 04:50:31,547 - INFO: Checking the repository for errors
INFO: 2019-12-09 04:50:32,869 - INFO: Backup completed
INFO: Backup container exit code: 0
```

3.6 crontab

Generates and verifies the crontab. This is done automatically when the container starts. It can be user to verify the configuration.

Example output:

```
/restic-compose-backup # rcb crontab
10 2 * * * source /env.sh && rcb backup > /proc/1/fd/1
```

3.7 cleanup

Purges all snapshots based on the configured policy. (`RESTIC_KEEP_*` env variables). It runs `restic forget` and `restic purge`.

Example output:

```
/restic-compose-backup # rcb cleanup
2019-12-09 05:09:52,892 - INFO: Forget outdated snapshots
2019-12-09 05:09:53,776 - INFO: Prune stale data freeing storage space
```

3.8 start-backup-process

This can only be executed by the backup process container. Attempting to run this command in the backup service will simply tell you it's not possible.

The backup process is doing the following:

- `status` is first called to ensure everything is ok
- Backs up `/volumes` if any volumes were mounted
- Backs up each configured database
- Runs `cleanup` purging snapshots based on the configured policy
- Checks the health of the repository

CHAPTER 4

Advanced

Currently work in progress. These are only notes :D

4.1 Temp Notes

- Quick setup guide from start to end
- we group snapshots by path when forgetting
- explain rcb commands
- examples of using restic directly
- Explain what happens during backup process
- Explain the backup process container
- cache directory
- Not displaying passwords in logs

4.2 Inner workings

- Each service in the compose setup is configured with a label to enable backup of volumes or databases
- When backup starts a new instance of the container is created mapping in all the needed volumes. It will copy networks etc to ensure databases can be reached
- Volumes are mounted to */volumes/<service_name>/<path>* in the backup process container. */volumes* is pushed into restic
- Databases are backed up from stdin / dumps into restic using path */databases/<service_name>/dump.sql*
- Cron triggers backup at 2AM every day

CHAPTER 5

Indices and tables

- genindex
- modindex
- search