restic

Release 0.8.3

May 06, 2018

Contents

CHAPTER 1

Introduction

Restic is a fast and secure backup program. In the following sections, we will present typical workflows, starting with installing, preparing a new repository, and making the first backup.

CHAPTER 2

Installation

2.1 Packages

Note that if at any point the package you're trying to use is outdated, you always have the option to use an official binary from the restic project.

These are up to date binaries, built in a reproducible and verifiable way, that you can download and run without having to do additional installation work.

Please see the *Official Binaries* section below for various downloads.

2.1.1 Mac OS X

If you are using Mac OS X, you can install restic using the homebrew package manager:

```
$ brew install restic
```

2.1.2 Arch Linux

On Arch Linux, there is a package called `restic-git` which can be installed from AUR, e.g. with `pacaur`:

```
$ pacaur -S restic-git
```

2.1.3 Nix & NixOS

If you are using Nix or NixOS there is a package available named `restic`. It can be installed uisng `nix-env`:

```
$ nix-env --install restic
```

2.1.4 Debian

On Debian, there's a package called `restic` which can be installed from the official repos, e.g. with `apt-get`:

```
$ apt-get install restic
```

Warning: Please be aware that, at the time of writing, Debian *stable* has `restic` version 0.3.3 which is very old. The *testing* and *unstable* branches have recent versions of `restic`.

2.1.5 RHEL & CentOS

restic can be installed via copr repository.

```
$ yum install yum-plugin-copr
$ yum copr enable copart/restic
$ yum install restic
```

2.1.6 Fedora

restic can be installed via copr repository.

```
$ dnf install dnf-plugin-core
$ dnf copr enable copart/restic
$ dnf install restic
```

2.1.7 Solus

restic can be installed from the official repo of Solus via the `eopkg` package manager:

```
$ eopkg install restic
```

2.1.8 OpenBSD

On OpenBSD 6.3 and greater, you can install restic using `pkg_add`:

```
# pkg_add restic
```

2.2 Official Binaries

2.2.1 Stable Releases

You can download the latest stable release versions of restic from the restic release page. These builds are considered stable and releases are made regularly in a controlled manner.

There's both pre-compiled binaries for different platforms as well as the source code available for download. Just download and run the one matching your system.

2.2.2 Unstable Builds

Another option is to use the latest builds for the master branch, available on the restic beta download site. These too are pre-compiled and ready to run, and a new version is built every time a push is made to the master branch.

2.2.3 Windows

On Windows, put the *restic.exe* into *%SystemRoot%System32* to use restic in scripts without the need for absolute paths to the binary. This requires Admin rights.

2.3 Docker Container

We're maintaining a bare docker container with just a few files and the restic binary, you can get it with *docker pull* like this:

```
$ docker pull restic/restic
```

Note:

Another docker container which offers more configuration options is available as a contribution (Thank you!). You can find it at https://github.com/Lobaro/restic-backup-docker

2.4 From Source

restic is written in the Go programming language and you need at least Go version 1.8. Building restic may also work with older versions of Go, but that's not supported. See the Getting started guide of the Go project for instructions how to install Go.

In order to build restic from source, execute the following steps:

```
$ git clone https://github.com/restic/restic
[...]

$ cd restic

$ go run build.go
```

You can easily cross-compile restic for all supported platforms, just supply the target OS and platform via the command-line options like this (for Windows and FreeBSD respectively):

```
$ go run build.go --goos windows --goarch amd64

$ go run build.go --goos freebsd --goarch 386

$ go run build.go --goos linux --goarch arm --goarm 6
```

The resulting binary is statically linked and does not require any libraries.

At the moment, the only tested compiler for restic is the official Go compiler. Building restic with gccgo may work, but is not supported.

2.5 Autocompletion

Restic can write out man pages and bash/zsh compatible autocompletion scripts:

```
$ ./restic generate --help

The "generate" command writes automatically generated files like the man pages
and the auto-completion files for bash and zsh).

Usage:
  restic generate [command] [flags]

Flags:
      --bash-completion file   write bash completion file
  -h, --help                   help for generate
      --man directory          write man pages to directory
      --zsh-completion file    write zsh completion file
```

Example for using sudo to write a bash completion script directly to the system-wide location:

```
$ sudo ./restic generate --bash-completion /etc/bash_completion.d/restic
writing bash completion file to /etc/bash_completion.d/restic
```

CHAPTER 3

Preparing a new repository

The place where your backups will be saved at is called a "repository". This chapter explains how to create ("init") such a repository. The repository can be stored locally, or on some remote server or service. We'll first cover using a local repository, the remaining sections of this chapter cover all the other options. You can skip to the next chapter once you've read the relevant section here.

3.1 Local

In order to create a repository at `/srv/restic-repo`, run the following command and enter the same password twice:

```
$ restic init --repo /srv/restic-repo
enter password for new backend:
enter password again:
created restic backend 085b3c76b9 at /srv/restic-repo
Please note that knowledge of your password is required to access the repository.
Losing your password means that your data is irrecoverably lost.
```

Warning: Remembering your password is important! If you lose it, you won't be able to access data stored in the repository.

For automated backups, restic accepts the repository location in the environment variable `RESTIC_REPOSITORY`. The password can be read from a file (via the option `--password-file` or the environment variable `RESTIC_PASSWORD_FILE`) or the environment variable `RESTIC_PASSWORD`.

3.2 SFTP

In order to backup data via SFTP, you must first set up a server with SSH and let it know your public key. Passwordless login is really important since restic fails to connect to the repository if the server prompts for credentials.

Once the server is configured, the setup of the SFTP repository can simply be achieved by changing the URL scheme in the `init` command:

```
$ restic -r sftp:user@host:/srv/restic-repo init
enter password for new backend:
enter password again:
created restic backend f1c6108821 at sftp:user@host:/srv/restic-repo
Please note that knowledge of your password is required to access the repository.
Losing your password means that your data is irrecoverably lost.
```

You can also specify a relative (read: no slash (`/`) character at the beginning) directory, in this case the dir is relative to the remote user's home directory.

Note: Please be aware that sftp servers do not expand the tilde character (`~`) normally used as an alias for a user's home directory. If you want to specify a path relative to the user's home directory, pass a relative path to the sftp backend.

The backend config string does not allow specifying a port. If you need to contact an sftp server on a different port, you can create an entry in the `ssh` file, usually located in your user's home directory at `~/.ssh/config` or in `/etc/ssh/ssh_config`:

```
Host foo
    User bar
    Port 2222
```

Then use the specified host name `foo` normally (you don't need to specify the user name in this case):

```
$ restic -r sftp:foo:/srv/restic-repo init
```

You can also add an entry with a special host name which does not exist, just for use with restic, and use the `Hostname` option to set the real host name:

```
Host restic-backup-host
    Hostname foo
    User bar
    Port 2222
```

Then use it in the backend specification:

```
$ restic -r sftp:restic-backup-host:/srv/restic-repo init
```

Last, if you'd like to use an entirely different program to create the SFTP connection, you can specify the command to be run with the option `-o sftp.command="foobar"`.

3.3 REST Server

In order to backup data to the remote server via HTTP or HTTPS protocol, you must first set up a remote REST server instance. Once the server is configured, accessing it is achieved by changing the URL scheme like this:

```
$ restic -r rest:http://host:8000/
```

Depending on your REST server setup, you can use HTTPS protocol, password protection, or multiple repositories. Or any combination of those features, as you see fit. TCP/IP port is also configurable. Here are some more examples:

```
$ restic -r rest:https://host:8000/
$ restic -r rest:https://user:pass@host:8000/
$ restic -r rest:https://user:pass@host:8000/my_backup_repo/
```

If you use TLS, restic will use the system's CA certificates to verify the server certificate. When the verification fails, restic refuses to proceed and exits with an error. If you have your own self-signed certificate, or a custom CA certificate should be used for verification, you can pass restic the certificate filename via the `--cacert` option.

REST server uses exactly the same directory structure as local backend, so you should be able to access it both locally and via HTTP, even simultaneously.

3.4 Amazon S3

Restic can backup data to any Amazon S3 bucket. However, in this case, changing the URL scheme is not enough since Amazon uses special security credentials to sign HTTP requests. By consequence, you must first setup the following environment variables with the credentials you obtained while creating the bucket.

```
$ export AWS_ACCESS_KEY_ID=<MY_ACCESS_KEY>
$ export AWS_SECRET_ACCESS_KEY=<MY_SECRET_ACCESS_KEY>
```

You can then easily initialize a repository that uses your Amazon S3 as a backend, if the bucket does not exist yet it will be created in the default location:

```
$ restic -r s3:s3.amazonaws.com/bucket_name init
enter password for new backend:
enter password again:
created restic backend eefee03bbd at s3:s3.amazonaws.com/bucket_name
Please note that knowledge of your password is required to access the repository.
Losing your password means that your data is irrecoverably lost.
```

It is not possible at the moment to have restic create a new bucket in a different location, so you need to create it using a different program. Afterwards, the S3 server (`s3.amazonaws.com`) will redirect restic to the correct endpoint.

Until version 0.8.0, restic used a default prefix of `restic`, so the files in the bucket were placed in a directory named `restic`. If you want to access a repository created with an older version of restic, specify the path after the bucket name like this:

```
$ restic -r s3:s3.amazonaws.com/bucket_name/restic [...]
```

For an S3-compatible server that is not Amazon (like Minio, see below), or is only available via HTTP, you can specify the URL to the server like this: `s3:http://server:port/bucket_name`.

3.5 Minio Server

Minio is an Open Source Object Storage, written in Go and compatible with AWS S3 API.

- Download and Install Minio Server.
- You can also refer to https://docs.minio.io for step by step guidance on installation and getting started on Minio Client and Minio Server.

You must first setup the following environment variables with the credentials of your running Minio Server.

```
$ export AWS_ACCESS_KEY_ID=<YOUR-MINIO-ACCESS-KEY-ID>
$ export AWS_SECRET_ACCESS_KEY= <YOUR-MINIO-SECRET-ACCESS-KEY>
```

Now you can easily initialize restic to use Minio server as backend with this command.

```
$ ./restic -r s3:http://localhost:9000/restic init
enter password for new backend:
enter password again:
created restic backend 6ad29560f5 at s3:http://localhost:9000/restic1
Please note that knowledge of your password is required to access
the repository. Losing your password means that your data is irrecoverably lost.
```

3.6 OpenStack Swift

Restic can backup data to an OpenStack Swift container. Because Swift supports various authentication methods, credentials are passed through environment variables. In order to help integration with existing OpenStack installations, the naming convention of those variables follows official python swift client:

```
# For keystone v1 authentication
$ export ST_AUTH=<MY_AUTH_URL>
$ export ST_USER=<MY_USER_NAME>
$ export ST_KEY=<MY_USER_PASSWORD>

# For keystone v2 authentication (some variables are optional)
$ export OS_AUTH_URL=<MY_AUTH_URL>
$ export OS_REGION_NAME=<MY_REGION_NAME>
$ export OS_USERNAME=<MY_USERNAME>
$ export OS_PASSWORD=<MY_PASSWORD>
$ export OS_TENANT_ID=<MY_TENANT_ID>
$ export OS_TENANT_NAME=<MY_TENANT_NAME>

# For keystone v3 authentication (some variables are optional)
$ export OS_AUTH_URL=<MY_AUTH_URL>
$ export OS_REGION_NAME=<MY_REGION_NAME>
$ export OS_USERNAME=<MY_USERNAME>
$ export OS_PASSWORD=<MY_PASSWORD>
$ export OS_USER_DOMAIN_NAME=<MY_DOMAIN_NAME>
$ export OS_PROJECT_NAME=<MY_PROJECT_NAME>
$ export OS_PROJECT_DOMAIN_NAME=<MY_PROJECT_DOMAIN_NAME>

# For authentication based on tokens
$ export OS_STORAGE_URL=<MY_STORAGE_URL>
$ export OS_AUTH_TOKEN=<MY_AUTH_TOKEN>
```

Restic should be compatible with OpenStack RC file in most cases.

Once environment variables are set up, a new repository can be created. The name of swift container and optional path can be specified. If the container does not exist, it will be created automatically:

```
$ restic -r swift:container_name:/path init    # path is optional
enter password for new backend:
enter password again:
created restic backend eefee03bbd at swift:container_name:/path
Please note that knowledge of your password is required to access the repository.
Losing your password means that your data is irrecoverably lost.
```

The policy of new container created by restic can be changed using environment variable:

```
$ export SWIFT_DEFAULT_CONTAINER_POLICY=<MY_CONTAINER_POLICY>
```

3.7 Backblaze B2

Restic can backup data to any Backblaze B2 bucket. You need to first setup the following environment variables with the credentials you obtained when signed into your B2 account:

```
$ export B2_ACCOUNT_ID=<MY_ACCOUNT_ID>
$ export B2_ACCOUNT_KEY=<MY_SECRET_ACCOUNT_KEY>
```

You can then easily initialize a repository stored at Backblaze B2. If the bucket does not exist yet, it will be created:

```
$ restic -r b2:bucketname:path/to/repo init
enter password for new backend:
enter password again:
created restic backend eefee03bbd at b2:bucketname:path/to/repo
Please note that knowledge of your password is required to access the repository.
Losing your password means that your data is irrecoverably lost.
```

The number of concurrent connections to the B2 service can be set with the `-o b2.connections=10`. By default, at most five parallel connections are established.

3.8 Microsoft Azure Blob Storage

You can also store backups on Microsoft Azure Blob Storage. Export the Azure account name and key as follows:

```
$ export AZURE_ACCOUNT_NAME=<ACCOUNT_NAME>
$ export AZURE_ACCOUNT_KEY=<SECRET_KEY>
```

Afterwards you can initialize a repository in a container called `foo` in the root path like this:

```
$ restic -r azure:foo:/ init
enter password for new backend:
enter password again:

created restic backend a934bac191 at azure:foo:/
[...]
```

The number of concurrent connections to the Azure Blob Storage service can be set with the `-o azure.connections=10`. By default, at most five parallel connections are established.

3.9 Google Cloud Storage

Restic supports Google Cloud Storage as a backend.

Restic connects to Google Cloud Storage via a service account.

For normal restic operation, the service account must have the `storage.objects.{create,delete,get,list}` permissions for the bucket. These are included in the "Storage Object Admin" role. `restic init` can

create the repository bucket. Doing so requires the `storage.buckets.create` permission ("Storage Admin" role). If the bucket already exists, that permission is unnecessary.

To use the Google Cloud Storage backend, first create a service account key and download the JSON credentials file. Second, find the Google Project ID that you can see in the Google Cloud Platform console at the "Storage/Settings" menu. Export the path to the JSON key file and the project ID as follows:

```
$ export GOOGLE_PROJECT_ID=123123123123
$ export GOOGLE_APPLICATION_CREDENTIALS=$HOME/.config/gs-secret-restic-key.json
```

Restic uses Google's client library to generate [default authentication material](https://developers.google.com/identity/protocols/application-default-credentials), which means if you're running in Google Container Engine or are otherwise located on an instance with default service accounts then these should work out the box.

Once authenticated, you can use the `gs:` backend type to create a new repository in the bucket `foo` at the root path:

```
$ restic -r gs:foo:/ init
enter password for new backend:
enter password again:

created restic backend bde47d6254 at gs:foo2/
[...]
```

The number of concurrent connections to the GCS service can be set with the `-o gs.connections=10`. By default, at most five parallel connections are established.

3.10 Other Services via rclone

The program rclone can be used to access many other different services and store data there. First, you need to install and configure rclone. The general backend specification format is `rclone:<remote>:<path>`, the `<remote>:<path>` component will be directly passed to rclone. When you configure a remote named `foo`, you can then call restic as follows to initiate a new repository in the path `bar` in the repo:

```
$ restic -r rclone:foo:bar init
```

Restic takes care of starting and stopping rclone.

As a more concrete example, suppose you have configured a remote named `b2prod` for Backblaze B2 with rclone, with a bucket called `yggdrasil`. You can then use rclone to list files in the bucket like this:

```
$ rclone ls b2prod:yggdrasil
```

In order to create a new repository in the root directory of the bucket, call restic like this:

```
$ restic -r rclone:b2prod:yggdrasil init
```

If you want to use the path `foo/bar/baz` in the bucket instead, pass this to restic:

```
$ restic -r rclone:b2prod:yggdrasil/foo/bar/baz init
```

Listing the files of an empty repository directly with rclone should return a listing similar to the following:

```
$ rclone ls b2prod:yggdrasil/foo/bar/baz
    155 bar/baz/config
    448 bar/baz/keys/4bf9c78049de689d73a56ed0546f83b8416795295cda12ec7fb9465af3900b44
```

Rclone can be configured with environment variables, so for instance configuring a bandwidth limit for rclone can be achieved by setting the `RCLONE_BWLIMIT` environment variable:

```
$ export RCLONE_BWLIMIT=1M
```

For debugging rclone, you can set the environment variable `RCLONE_VERBOSE=2`.

The rclone backend has two additional options:

- `-o rclone.program` specifies the path to rclone, the default value is just `rclone`
- `-o rclone.args` allows setting the arguments passed to rclone, by default this is `serve restic --stdio --b2-hard-delete --drive-use-trash=false`

The reason for the two last parameters (`--b2-hard-delete` and `--drive-use-trash=false`) can be found in the corresponding GitHub issue #1657.

In order to start rclone, restic will build a list of arguments by joining the following lists (in this order): `rclone.program`, `rclone.args` and as the last parameter the value that follows the `rclone:` prefix of the repository specification.

So, calling restic like this

```
$ restic -o rclone.program="/path/to/rclone" \
  -o rclone.args="serve restic --stdio --bwlimit 1M --b2-hard-delete --verbose" \
  -r rclone:b2:foo/bar
```

runs rclone as follows:

```
$ /path/to/rclone serve restic --stdio --bwlimit 1M --b2-hard-delete --verbose b2:foo/
↪bar
```

Manually setting `rclone.program` also allows running a remote instance of rclone e.g. via SSH on a server, for example:

```
$ restic -o rclone.program="ssh user@host rclone" -r rclone:b2:foo/bar
```

The rclone command may also be hard-coded in the SSH configuration or the user's public key, in this case it may be sufficient to just start the SSH connection (and it's irrelevant what's passed after `rclone:` in the repository specification):

```
$ restic -o rclone.program="ssh user@host" -r rclone:x
```

3.11 Password prompt on Windows

At the moment, restic only supports the default Windows console interaction. If you use emulation environments like MSYS2 or Cygwin, which use terminals like `Mintty` or `rxvt`, you may get a password error:

You can workaround this by using a special tool called `winpty` (look here and here for detail information). On MSYS2, you can install `winpty` as follows:

```
$ pacman -S winpty
$ winpty restic -r /srv/restic-repo init
```

CHAPTER 4

Backing up

Now we're ready to backup some data. The contents of a directory at a specific point in time is called a "snapshot" in restic. Run the following command and enter the repository password you chose above again:

```
$ restic -r /srv/restic-repo --verbose backup ~/work
open repository
enter password for repository:
password is correct
lock repository
load index files
start scan
start backup
scan finished in 1.837s
processed 1.720 GiB in 0:12
Files:        5307 new,     0 changed,     0 unmodified
Dirs:         1867 new,     0 changed,     0 unmodified
Added:      1.700 GiB
snapshot 40dc1520 saved
```

As you can see, restic created a backup of the directory and was pretty fast! The specific snapshot just created is identified by a sequence of hexadecimal characters, `40dc1520` in this case.

If you don't pass the `--verbose` option, restic will print less data (but you'll still get a nice live status display).

If you run the command again, restic will create another snapshot of your data, but this time it's even faster. This is de-duplication at work!

```
$ restic -r /srv/restic-repo backup --verbose ~/work
open repository
enter password for repository:
password is correct
lock repository
load index files
using parent snapshot d875ae93
start scan
start backup
```

(continues on next page)

(continued from previous page)

```
scan finished in 1.881s
processed 1.720 GiB in 0:03
Files:           0 new,     0 changed,  5307 unmodified
Dirs:            0 new,     0 changed,  1867 unmodified
Added:       0 B
snapshot 79766175 saved
```

You can even backup individual files in the same repository (not passing `--verbose` means less output):

```
$ restic -r /srv/restic-repo backup ~/work.txt
enter password for repository:
password is correct
snapshot 249d0210 saved
```

If you're interested in what restic does, pass `--verbose` twice (or `--verbose 2`) to display detailed information about each file and directory restic encounters:

```
$ echo 'more data foo bar' >> ~/work.txt

$ restic -r /srv/restic-repo backup --verbose --verbose ~/work.txt
open repository
enter password for repository:
password is correct
lock repository
load index files
using parent snapshot f3f8d56b
start scan
start backup
scan finished in 2.115s
modified  /home/user/work.txt, saved in 0.007s (22 B added)
modified  /home/user/, saved in 0.008s (0 B added, 378 B metadata)
modified  /home/, saved in 0.009s (0 B added, 375 B metadata)
processed 22 B in 0:02
Files:           0 new,     1 changed,     0 unmodified
Dirs:            0 new,     2 changed,     0 unmodified
Data Blobs:      1 new
Tree Blobs:      3 new
Added:       1.116 KiB
snapshot 8dc503fc saved
```

In fact several hosts may use the same repository to backup directories and files leading to a greater de-duplication.

Please be aware that when you backup different directories (or the directories to be saved have a variable name component like a time/date), restic always needs to read all files and only afterwards can compute which parts of the files need to be saved. When you backup the same directory again (maybe with new or changed files) restic will find the old snapshot in the repo and by default only reads those files that are new or have been modified since the last snapshot. This is decided based on the modify date of the file in the file system.

Now is a good time to run `restic check` to verify that all data is properly stored in the repository. You should run this command regularly to make sure the internal structure of the repository is free of errors.

4.1 Including and Excluding Files

You can exclude folders and files by specifying exclude patterns, currently the exclude options are:

- `--exclude` Specified one or more times to exclude one or more items

- `--exclude-caches` Specified once to exclude folders containing a special file
- `--exclude-file` Specified one or more times to exclude items listed in a given file
- `--exclude-if-present` Specified one or more times to exclude a folders content if it contains a given file (optionally having a given header)

  Let's say we have a file called `excludes.txt` with the following content:

:: # exclude go-files *.go # exclude foo/x/y/z/bar foo/x/bar foo/bar foo/*/bar

It can be used like this:

```
$ restic -r /srv/restic-repo backup ~/work --exclude="*.c" --exclude-file=excludes.txt
```

This instruct restic to exclude files matching the following criteria:

- All files matching `*.go` (second line in `excludes.txt`)
- All files and sub-directories named `bar` which reside somewhere below a directory called `foo` (fourth line in `excludes.txt`)
- All files matching `*.c` (parameter `--exclude`)

Please see `restic help backup` for more specific information about each exclude option.

Patterns use filepath.Glob internally, see filepath.Match for syntax. Patterns are tested against the full path of a file/dir to be saved, even if restic is passed a relative path to save. Environment-variables in exclude-files are expanded with os.ExpandEnv.

Patterns need to match on complete path components. For example, the pattern `foo`:

- matches `/dir1/foo/dir2/file` and `/dir/foo`
- does not match `/dir/foobar` or `barfoo`

A trailing `/` is ignored, a leading `/` anchors the pattern at the root directory. This means, `/bin` matches `/bin/bash` but does not match `/usr/bin/restic`.

Regular wildcards cannot be used to match over the directory separator `/`. For example: `b*ash` matches `/bin/bash` but does not match `/bin/ash`.

For this, the special wildcard `**` can be used to match arbitrary sub-directories: The pattern `foo/**/bar` matches:

- `/dir1/foo/dir2/bar/file`
- `/foo/bar/file`
- `/tmp/foo/bar`

By specifying the option `--one-file-system` you can instruct restic to only backup files from the file systems the initially specified files or directories reside on. For example, calling restic like this won't backup `/sys` or `/dev` on a Linux system:

```
$ restic -r /srv/restic-repo backup --one-file-system /
```

By using the `--files-from` option you can read the files you want to backup from a file. This is especially useful if a lot of files have to be backed up that are not in the same folder or are maybe pre-filtered by other software.

For example maybe you want to backup files which have a name that matches a certain pattern:

```
$ find /tmp/somefiles | grep 'PATTERN' > /tmp/files_to_backup
```

You can then use restic to backup the filtered files:

```
$ restic -r /srv/restic-repo backup --files-from /tmp/files_to_backup
```

Incidentally you can also combine `--files-from` with the normal files args:

```
$ restic -r /srv/restic-repo backup --files-from /tmp/files_to_backup /tmp/some_
↪additional_file
```

Paths in the listing file can be absolute or relative.

4.2 Comparing Snapshots

Restic has a *diff* command which shows the difference between two snapshots and displays a small statistic, just pass the command two snapshot IDs:

```
$ restic -r /srv/restic-repo diff 5845b002 2ab627a6
password is correct
comparing snapshot ea657ce5 to 2ab627a6:

 C   /restic/cmd_diff.go
+    /restic/foo
 C   /restic/restic

Files:           0 new,     0 removed,     2 changed
Dirs:            1 new,     0 removed
Others:          0 new,     0 removed
Data Blobs:     14 new,    15 removed
Tree Blobs:      2 new,     1 removed
  Added:   16.403 MiB
  Removed: 16.402 MiB
```

4.3 Backing up special items and metadata

Symlinks are archived as symlinks, `restic` does not follow them. When you restore, you get the same symlink again, with the same link target and the same timestamps.

If there is a **bind-mount** below a directory that is to be saved, restic descends into it.

Device files are saved and restored as device files. This means that e.g. `/dev/sda` is archived as a block device file and restored as such. This also means that the content of the corresponding disk is not read, at least not from the device file.

By default, restic does not save the access time (atime) for any files or other items, since it is not possible to reliably disable updating the access time by restic itself. This means that for each new backup a lot of metadata is written, and the next backup needs to write new metadata again. If you really want to save the access time for files and directories, you can pass the `--with-atime` option to the `backup` command.

4.4 Reading data from stdin

Sometimes it can be nice to directly save the output of a program, e.g. `mysqldump` so that the SQL can later be restored. Restic supports this mode of operation, just supply the option `--stdin` to the `backup` command like this:

```
$ mysqldump [...] | restic -r /srv/restic-repo backup --stdin
```

This creates a new snapshot of the output of `mysqldump`. You can then use e.g. the fuse mounting option (see below) to mount the repository and read the file.

By default, the file name `stdin` is used, a different name can be specified with `--stdin-filename`, e.g. like this:

```
$ mysqldump [...] | restic -r /srv/restic-repo backup --stdin --stdin-filename␣
↪production.sql
```

4.5 Tags for backup

Snapshots can have one or more tags, short strings which add identifying information. Just specify the tags for a snapshot one by one with `--tag`:

```
$ restic -r /srv/restic-repo backup --tag projectX --tag foo --tag bar ~/work
[...]
```

The tags can later be used to keep (or forget) snapshots with the `forget` command. The command `tag` can be used to modify tags on an existing snapshot.

CHAPTER 5

Working with repositories

5.1 Listing all snapshots

Now, you can list all the snapshots stored in the repository:

```
$ restic -r /srv/restic-repo snapshots
enter password for repository:
ID        Date                 Host    Tags   Directory
----------------------------------------------------------------------
40dc1520  2015-05-08 21:38:30  kasimir        /home/user/work
79766175  2015-05-08 21:40:19  kasimir        /home/user/work
bdbd3439  2015-05-08 21:45:17  luigi          /home/art
590c8fc8  2015-05-08 21:47:38  kazik          /srv
9f0bc19e  2015-05-08 21:46:11  luigi          /srv
```

You can filter the listing by directory path:

```
$ restic -r /srv/restic-repo snapshots --path="/srv"
enter password for repository:
ID        Date                 Host    Tags   Directory
----------------------------------------------------------------------
590c8fc8  2015-05-08 21:47:38  kazik          /srv
9f0bc19e  2015-05-08 21:46:11  luigi          /srv
```

Or filter by host:

```
$ restic -r /srv/restic-repo snapshots --host luigi
enter password for repository:
ID        Date                 Host    Tags   Directory
----------------------------------------------------------------------
bdbd3439  2015-05-08 21:45:17  luigi          /home/art
9f0bc19e  2015-05-08 21:46:11  luigi          /srv
```

Combining filters is also possible.

5.2 Checking a repo's integrity and consistency

Imagine your repository is saved on a server that has a faulty hard drive, or even worse, attackers get privileged access and modify your backup with the intention to make you restore malicious data:

```
$ sudo echo "boom" >> backup/index/
↪d795ffa99a8ab8f8e42cec1f814df4e48b8f49129360fb57613df93739faee97
```

In order to detect these things, it is a good idea to regularly use the `check` command to test whether everything is alright, your precious backup data is consistent and the integrity is unharmed:

```
$ restic -r /srv/restic-repo check
Load indexes
ciphertext verification failed
```

Trying to restore a snapshot which has been modified as shown above will yield the same error:

```
$ restic -r /srv/restic-repo restore 79766175 --target /tmp/restore-work
Load indexes
ciphertext verification failed
```

By default, `check` command does not check that repository data files are unmodified. Use `--read-data` parameter to check all repository data files:

```
$ restic -r /srv/restic-repo check --read-data
load indexes
check all packs
check snapshots, trees and blobs
read all data
```

Use `--read-data-subset=n/t` parameter to check subset of repository data files. The parameter takes two values, `n` and `t`. All repository data files are logically devided in `t` roughly equal groups and only files that belong to the group number `n` are checked. For example, the following commands check all repository data files over 5 separate invocations:

```
$ restic -r /srv/restic-repo check --read-data-subset=1/5
$ restic -r /srv/restic-repo check --read-data-subset=2/5
$ restic -r /srv/restic-repo check --read-data-subset=3/5
$ restic -r /srv/restic-repo check --read-data-subset=4/5
$ restic -r /srv/restic-repo check --read-data-subset=5/5
```

CHAPTER 6

Restoring from backup

6.1 Restoring from a snapshot

Restoring a snapshot is as easy as it sounds, just use the following command to restore the contents of the latest snapshot to `/tmp/restore-work`:

```
$ restic -r /srv/restic-repo restore 79766175 --target /tmp/restore-work
enter password for repository:
restoring <Snapshot of [/home/user/work] at 2015-05-08 21:40:19.884408621 +0200 CEST>␣
↪to /tmp/restore-work
```

Use the word `latest` to restore the last backup. You can also combine `latest` with the `--host` and `--path` filters to choose the last backup for a specific host, path or both.

```
$ restic -r /srv/restic-repo restore latest --target /tmp/restore-art --path "/home/
↪art" --host luigi
enter password for repository:
restoring <Snapshot of [/home/art] at 2015-05-08 21:45:17.884408621 +0200 CEST> to /
↪tmp/restore-art
```

Use `--exclude` and `--include` to restrict the restore to a subset of files in the snapshot. For example, to restore a single file:

```
$ restic -r /srv/restic-repo restore 79766175 --target /tmp/restore-work --include /
↪work/foo
enter password for repository:
restoring <Snapshot of [/home/user/work] at 2015-05-08 21:40:19.884408621 +0200 CEST>␣
↪to /tmp/restore-work
```

This will restore the file `foo` to `/tmp/restore-work/work/foo`.

6.2 Restore using mount

Browsing your backup as a regular file system is also very easy. First, create a mount point such as `/mnt/restic` and then use the following command to serve the repository with FUSE:

```
$ mkdir /mnt/restic
$ restic -r /srv/restic-repo mount /mnt/restic
enter password for repository:
Now serving /srv/restic-repo at /mnt/restic
Don't forget to umount after quitting!
```

Mounting repositories via FUSE is not possible on OpenBSD, Solaris/illumos and Windows.

Restic supports storage and preservation of hard links. However, since hard links exist in the scope of a filesystem by definition, restoring hard links from a fuse mount should be done by a program that preserves hard links. A program that does so is `rsync`, used with the option –hard-links.

6.3 Printing files to stdout

Sometimes it's helpful to print files to stdout so that other programs can read the data directly. This can be achieved by using the *dump* command, like this:

```
$ restic -r /srv/restic-repo dump latest production.sql | mysql
```

CHAPTER 7

Removing backup snapshots

All backup space is finite, so restic allows removing old snapshots. This can be done either manually (by specifying a snapshot ID to remove) or by using a policy that describes which snapshots to forget. For all remove operations, two commands need to be called in sequence: `forget` to remove a snapshot and `prune` to actually remove the data that was referenced by the snapshot from the repository. This can be automated with the `--prune` option of the `forget` command, which runs `prune` automatically if snapshots have been removed.

It is advisable to run `restic check` after pruning, to make sure you are alerted, should the internal data structures of the repository be damaged.

7.1 Remove a single snapshot

The command `snapshots` can be used to list all snapshots in a repository like this:

```
$ restic -r /srv/restic-repo snapshots
enter password for repository:
ID        Date                 Host      Tags  Directory
----------------------------------------------------------------------
40dc1520  2015-05-08 21:38:30  kasimir         /home/user/work
79766175  2015-05-08 21:40:19  kasimir         /home/user/work
bdbd3439  2015-05-08 21:45:17  luigi           /home/art
590c8fc8  2015-05-08 21:47:38  kazik           /srv
9f0bc19e  2015-05-08 21:46:11  luigi           /srv
```

In order to remove the snapshot of `/home/art`, use the `forget` command and specify the snapshot ID on the command line:

```
$ restic -r /srv/restic-repo forget bdbd3439
enter password for repository:
removed snapshot d3f01f63
```

Afterwards this snapshot is removed:

```
$ restic -r /srv/restic-repo snapshots
enter password for repository:
ID        Date                 Host     Tags  Directory
----------------------------------------------------------------------
40dc1520  2015-05-08 21:38:30  kasimir        /home/user/work
79766175  2015-05-08 21:40:19  kasimir        /home/user/work
590c8fc8  2015-05-08 21:47:38  kazik          /srv
9f0bc19e  2015-05-08 21:46:11  luigi          /srv
```

But the data that was referenced by files in this snapshot is still stored in the repository. To cleanup unreferenced data, the `prune` command must be run:

```
$ restic -r /srv/restic-repo prune
enter password for repository:

counting files in repo
building new index for repo
[0:00] 100.00%  22 / 22 files
repository contains 22 packs (8512 blobs) with 100.092 MiB bytes
processed 8512 blobs: 0 duplicate blobs, 0B duplicate
load all snapshots
find data that is still in use for 1 snapshots
[0:00] 100.00%  1 / 1 snapshots
found 8433 of 8512 data blobs still in use
will rewrite 3 packs
creating new index
[0:00] 86.36%  19 / 22 files
saved new index as 544a5084
done
```

Afterwards the repository is smaller.

You can automate this two-step process by using the `--prune` switch to `forget`:

```
$ restic forget --keep-last 1 --prune
snapshots for host mopped, directories /home/user/work:

keep 1 snapshots:
ID        Date                 Host        Tags        Directory
----------------------------------------------------------------------
4bba301e  2017-02-21 10:49:18  mopped                  /home/user/work

remove 1 snapshots:
ID        Date                 Host        Tags        Directory
----------------------------------------------------------------------
8c02b94b  2017-02-21 10:48:33  mopped                  /home/user/work

1 snapshots have been removed, running prune
counting files in repo
building new index for repo
[0:00] 100.00%  37 / 37 packs
repository contains 37 packs (5521 blobs) with 151.012 MiB bytes
processed 5521 blobs: 0 duplicate blobs, 0B duplicate
load all snapshots
find data that is still in use for 1 snapshots
[0:00] 100.00%  1 / 1 snapshots
found 5323 of 5521 data blobs still in use, removing 198 blobs
will delete 0 packs and rewrite 27 packs, this frees 22.106 MiB
```

(continues on next page)

(continued from previous page)

```
creating new index
[0:00] 100.00%  30 / 30 packs
saved new index as b49f3e68
done
```

7.2 Removing snapshots according to a policy

Removing snapshots manually is tedious and error-prone, therefore restic allows specifying which snapshots should be removed automatically according to a policy. You can specify how many hourly, daily, weekly, monthly and yearly snapshots to keep, any other snapshots are removed. The most important command-line parameter here is `--dry-run` which instructs restic to not remove anything but print which snapshots would be removed.

When `forget` is run with a policy, restic loads the list of all snapshots, then groups these by host name and list of directories. The grouping options can be set with `--group-by`, to only group snapshots by paths and tags use `--group-by paths,tags`. The policy is then applied to each group of snapshots separately. This is a safety feature.

The `forget` command accepts the following parameters:

- `--keep-last n` never delete the `n` last (most recent) snapshots
- `--keep-hourly n` for the last `n` hours in which a snapshot was made, keep only the last snapshot for each hour.
- `--keep-daily n` for the last `n` days which have one or more snapshots, only keep the last one for that day.
- `--keep-weekly n` for the last `n` weeks which have one or more snapshots, only keep the last one for that week.
- `--keep-monthly n` for the last `n` months which have one or more snapshots, only keep the last one for that month.
- `--keep-yearly n` for the last `n` years which have one or more snapshots, only keep the last one for that year.
- `--keep-tag` keep all snapshots which have all tags specified by this option (can be specified multiple times).

Additionally, you can restrict removing snapshots to those which have a particular hostname with the `--hostname` parameter, or tags with the `--tag` option. When multiple tags are specified, only the snapshots which have all the tags are considered. For example, the following command removes all but the latest snapshot of all snapshots that have the tag `foo`:

```
$ restic forget --tag foo --keep-last 1
```

This command removes all but the last snapshot of all snapshots that have either the `foo` or `bar` tag set:

```
$ restic forget --tag foo --tag bar --keep-last 1
```

To only keep the last snapshot of all snapshots with both the tag `foo` and `bar` set use:

```
$ restic forget --tag foo,tag bar --keep-last 1
```

All the `--keep-*` options above only count hours/days/weeks/months/years which have a snapshot, so those without a snapshot are ignored.

For safety reasons, restic refuses to act on an "empty" policy. For example, if one were to specify `--keep-last 0` to forget *all* snapshots in the repository, restic will respond that no snapshots will be removed. To delete all snapshots, use `--keep-last 1` and then finally remove the last snapshot ID manually (by passing the ID to `forget`).

All snapshots are evaluated against all matching `--keep-*` counts. A single snapshot on 2017-09-30 (Sun) will count as a daily, weekly and monthly.

Let's explain this with an example: Suppose you have only made a backup on each Sunday for 12 weeks. Then `forget --keep-daily 4` will keep the last four snapshots for the last four Sundays, but remove the rest. Only counting the days which have a backup and ignore the ones without is a safety feature: it prevents restic from removing many snapshots when no new ones are created. If it was implemented otherwise, running `forget --keep-daily 4` on a Friday would remove all snapshots!

Another example: Suppose you make daily backups for 100 years. Then `forget --keep-daily 7 --keep-weekly 5 --keep-monthly 12 --keep-yearly 75` will keep the most recent 7 daily snapshots, then 4 (remember, 7 dailies already include a week!) last-day-of-the-weeks and 11 or 12 last-day-of-the-months (11 or 12 depends if the 5 weeklies cross a month). And finally 75 last-day-of-the-year snapshots. All other snapshots are removed.

CHAPTER 8

Encryption

"The design might not be perfect, but it's good. Encryption is a first-class feature, the implementation looks sane and I guess the deduplication trade-off is worth it. So... I'm going to use restic for my personal backups." Filippo Valsorda

8.1 Manage repository keys

The `key` command allows you to set multiple access keys or passwords per repository. In fact, you can use the `list`, `add`, `remove`, and `passwd` (changes a password) sub-commands to manage these keys very precisely:

```
$ restic -r /srv/restic-repo key list
enter password for repository:
 ID          User        Host        Created
----------------------------------------------------------------------
*eb78040b    username    kasimir   2015-08-12 13:29:57

$ restic -r /srv/restic-repo key add
enter password for repository:
enter password for new key:
enter password again:
saved new key as <Key of username@kasimir, created on 2015-08-12 13:35:05.316831933␣
↪+0200 CEST>

$ restic -r /srv/restic-repo key list
enter password for repository:
 ID          User        Host        Created
----------------------------------------------------------------------
 5c657874    username    kasimir   2015-08-12 13:35:05
*eb78040b    username    kasimir   2015-08-12 13:29:57
```

CHAPTER 9

Scripting

This is a list of how certain tasks may be accomplished when you use restic via scripts.

9.1 Check if a repository is already initialized

You may find a need to check if a repository is already initialized, perhaps to prevent your script from initializing a repository multiple times. The command `snapshots` may be used for this purpose:

```
$ restic -r /srv/restic-repo snapshots
Fatal: unable to open config file: Stat: stat /srv/restic-repo/config: no such file␣
↪or directory
Is there a repository at the following location?
/srv/restic-repo
```

If a repository does not exist, restic will return a non-zero exit code and print an error message. Note that restic will also return a non-zero exit code if a different error is encountered (e.g.: incorrect password to `snapshots`) and it may print a different error message. If there are no errors, restic will return a zero exit code and print all the snapshots.

CHAPTER 10

Examples

10.1 Setting up restic with Amazon S3

10.1.1 Preface

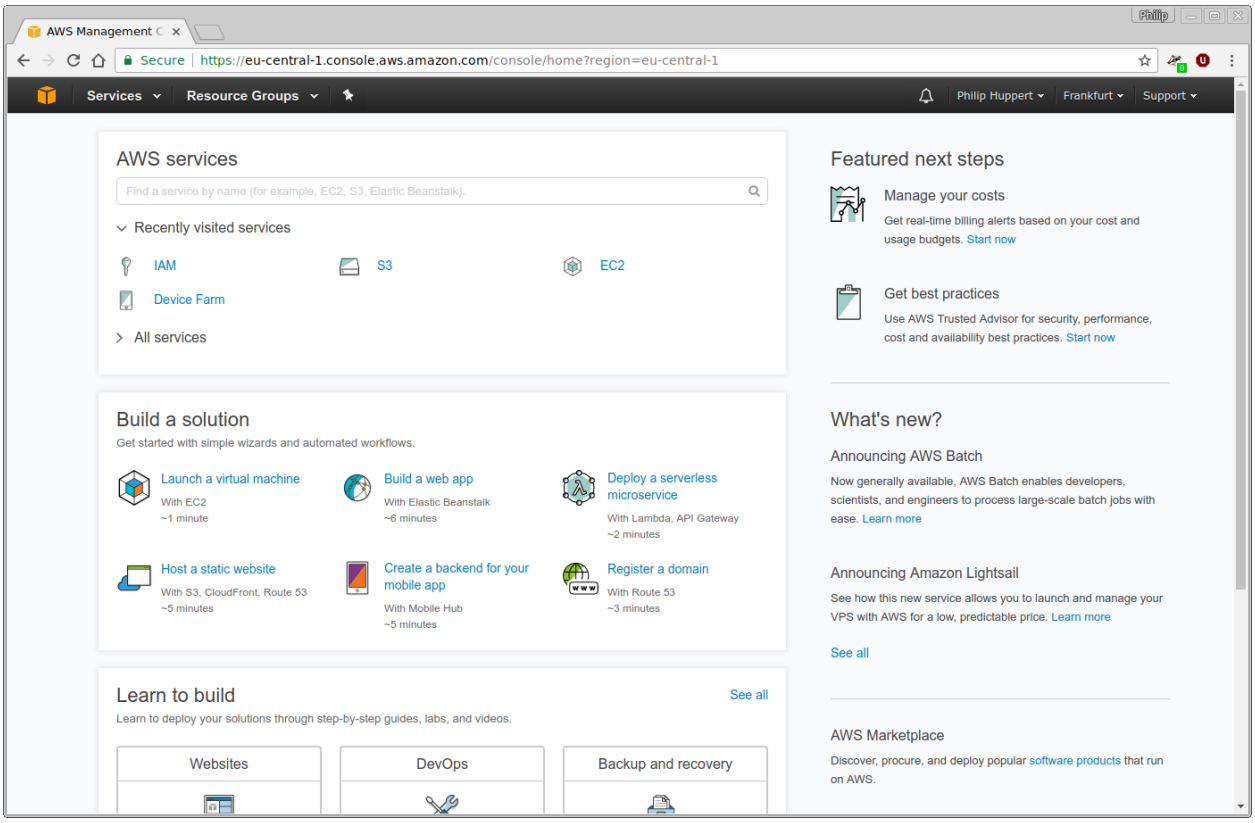
This tutorial will show you how to use restic with AWS S3. It will show you how to navigate the AWS web interface, create an S3 bucket, create a user with access to only this bucket, and finally how to connect restic to this bucket.

10.1.2 Prerequisites

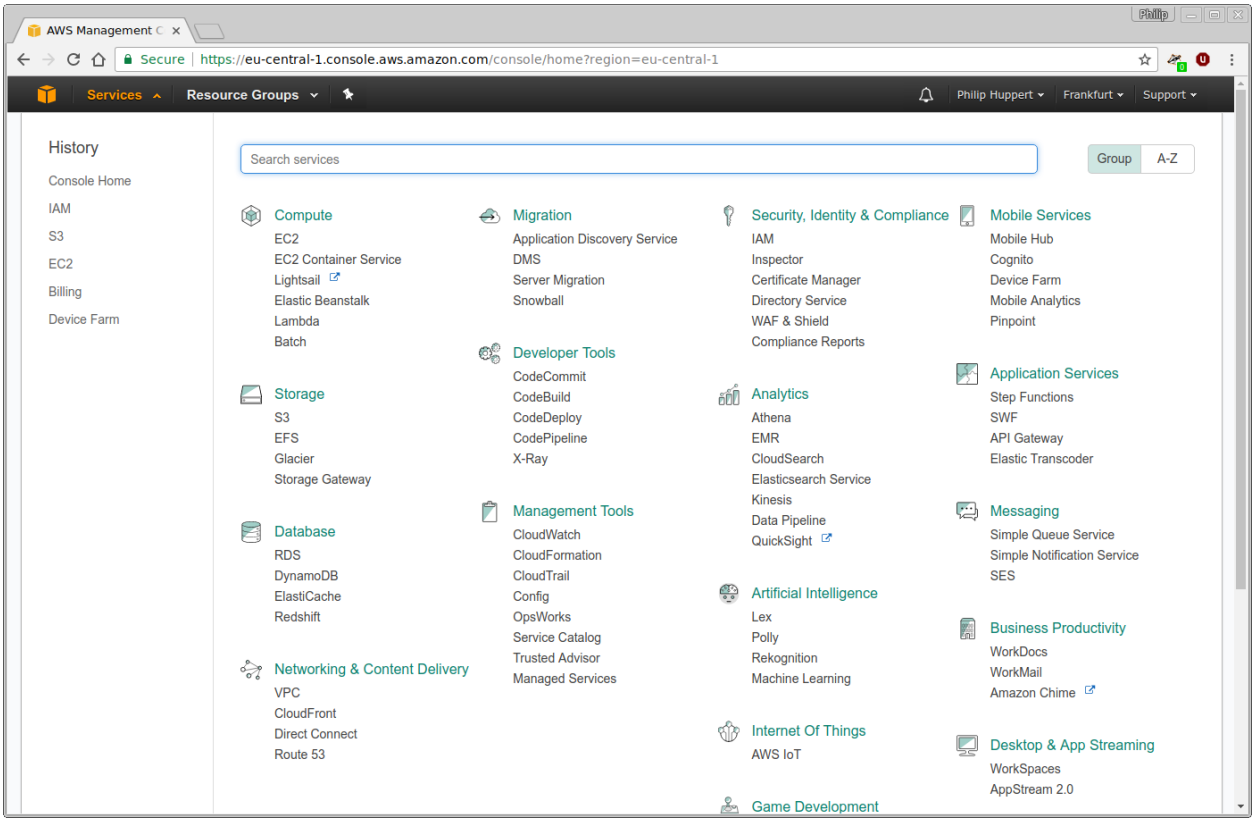
You should already have a `restic` binary available on your system that you can run. Furthermore, you should also have an account with AWS. You will likely need to provide credit card details for billing purposes, even if you use their free-tier.

10.1.3 Logging into AWS

Point your browser to https://console.aws.amazon.com and log in using your AWS account. You will be presented with the AWS homepage:



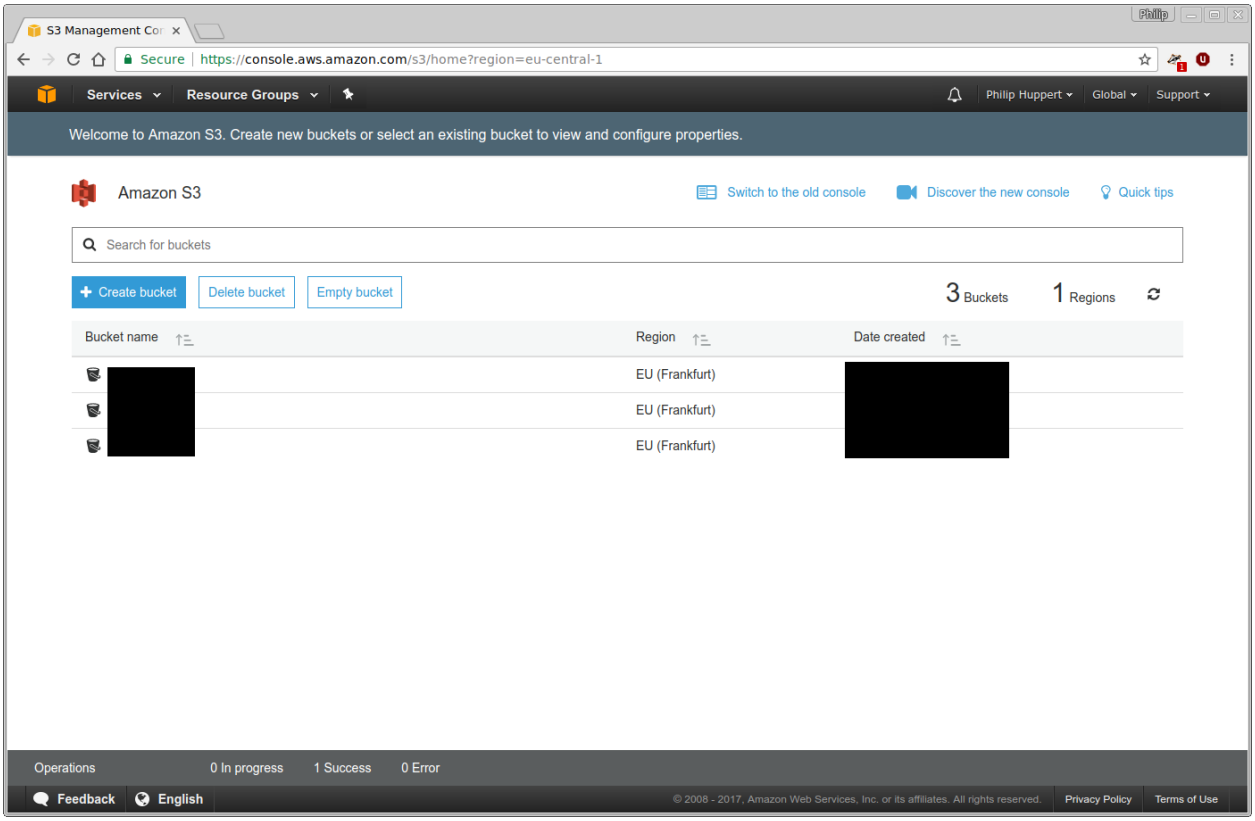
By using the "Services" button in the upper left corder, a menu of all services provided by AWS can be opened:



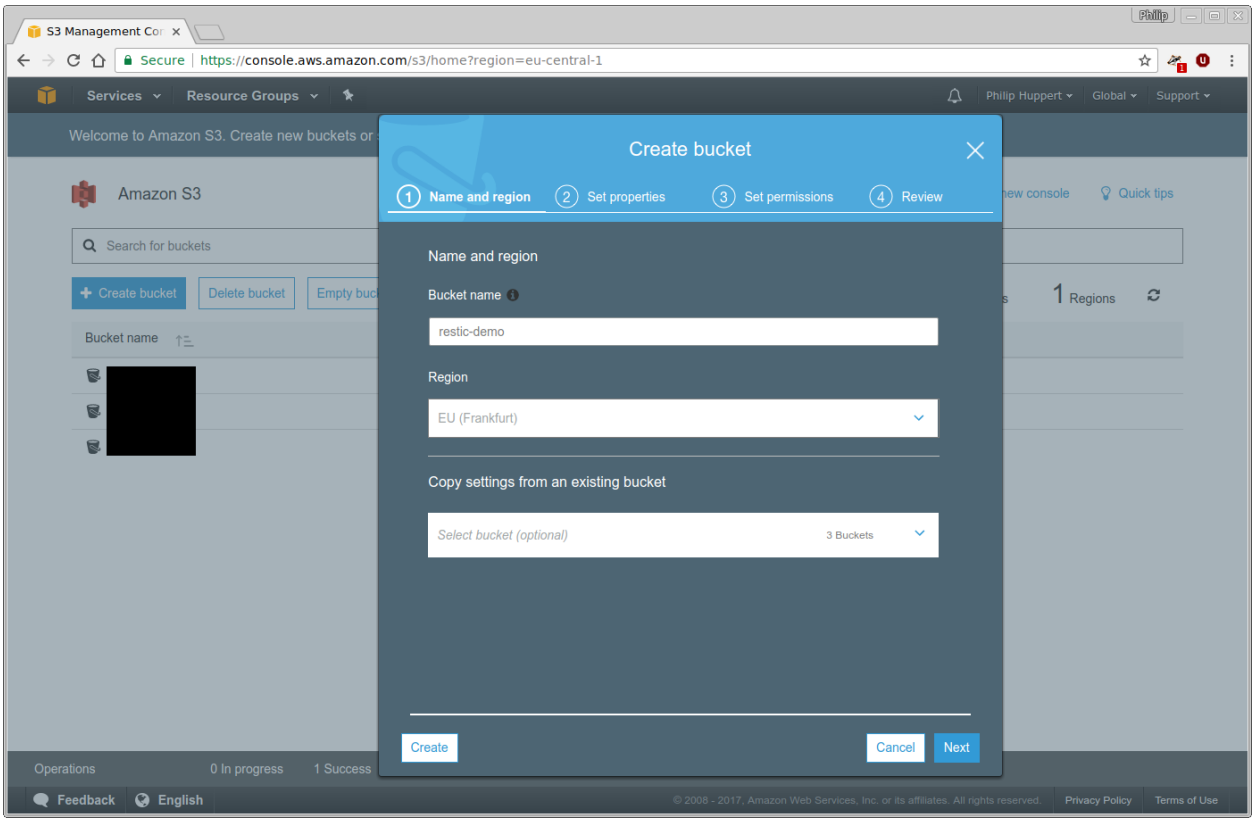
For this tutorial, the Simple Storage Service (S3), as well as Identity and Access Management (IAM) are relevant.

10.1.4 Creating the bucket

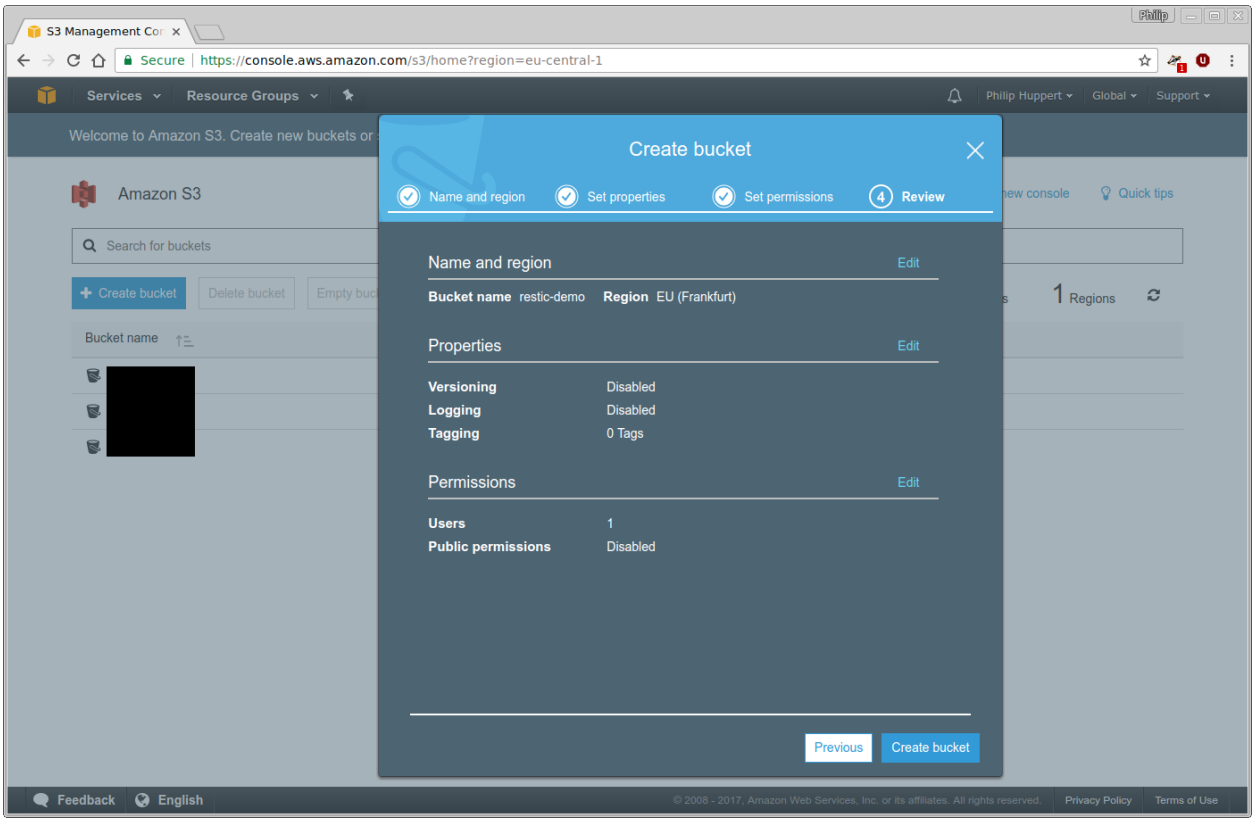
First, a bucket to store your backups in must be created. Using the "Services" menu, navigate to S3. In case you already have some S3 buckets, you will see a list of them here:



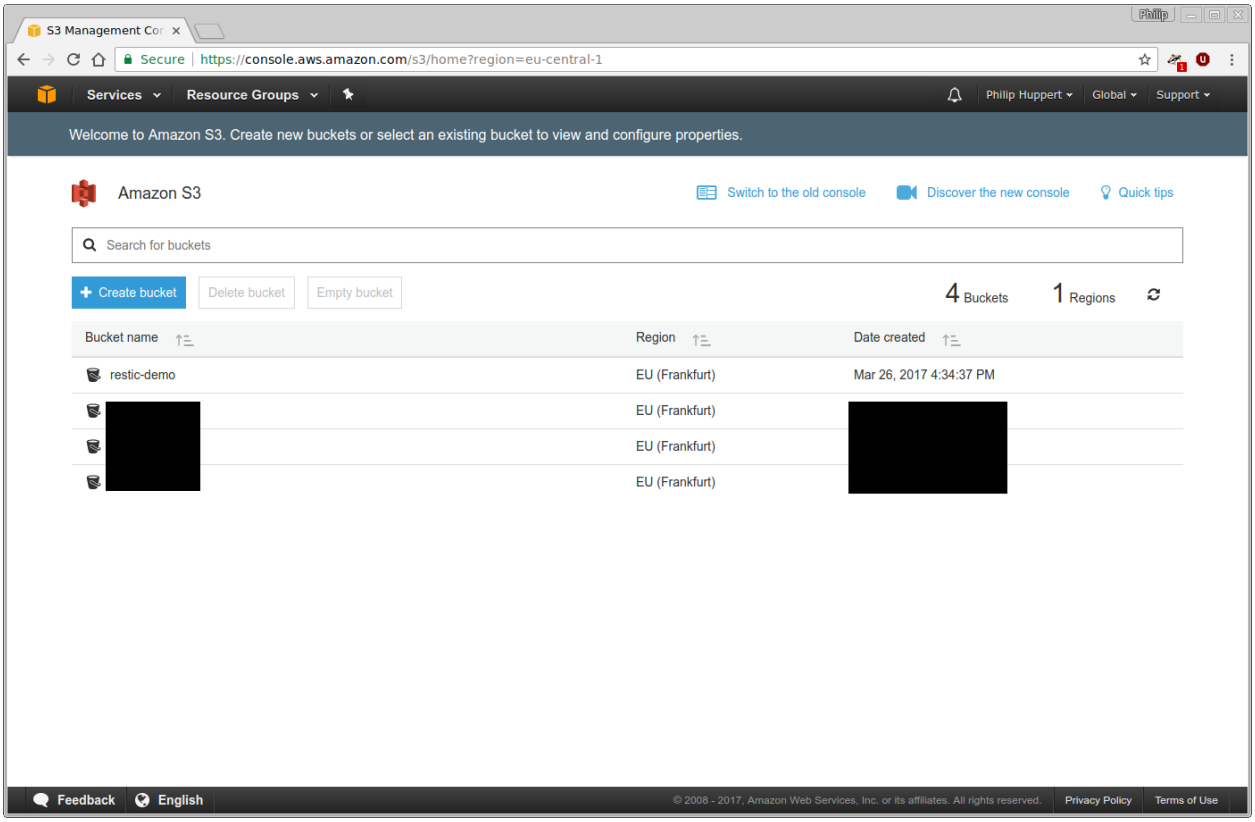
Click the "Create bucket" button and choose a name and region for your new bucket. For the purpose of this tutorial, the bucket will be named `restic-demo` and reside in Frankfurt. Because the bucket name space is shared among all AWS users, the name `restic-demo` may not be available to you. Be creative and choose a unique bucket name.



It is not necessary to configure any special properties or permissions of the bucket just yet. Therefore, just finish the wizard without making any further changes:

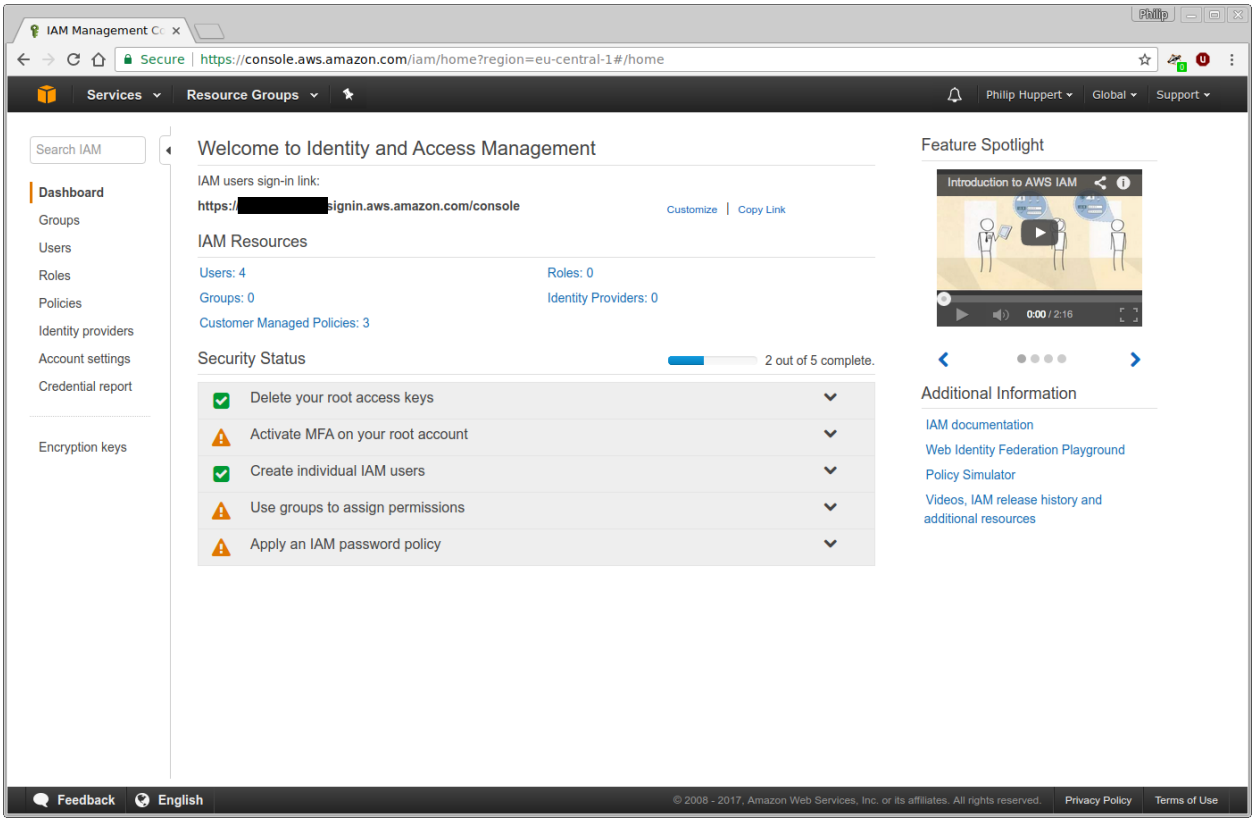


The newly created `restic-demo` bucket will now appear on the list of S3 buckets:

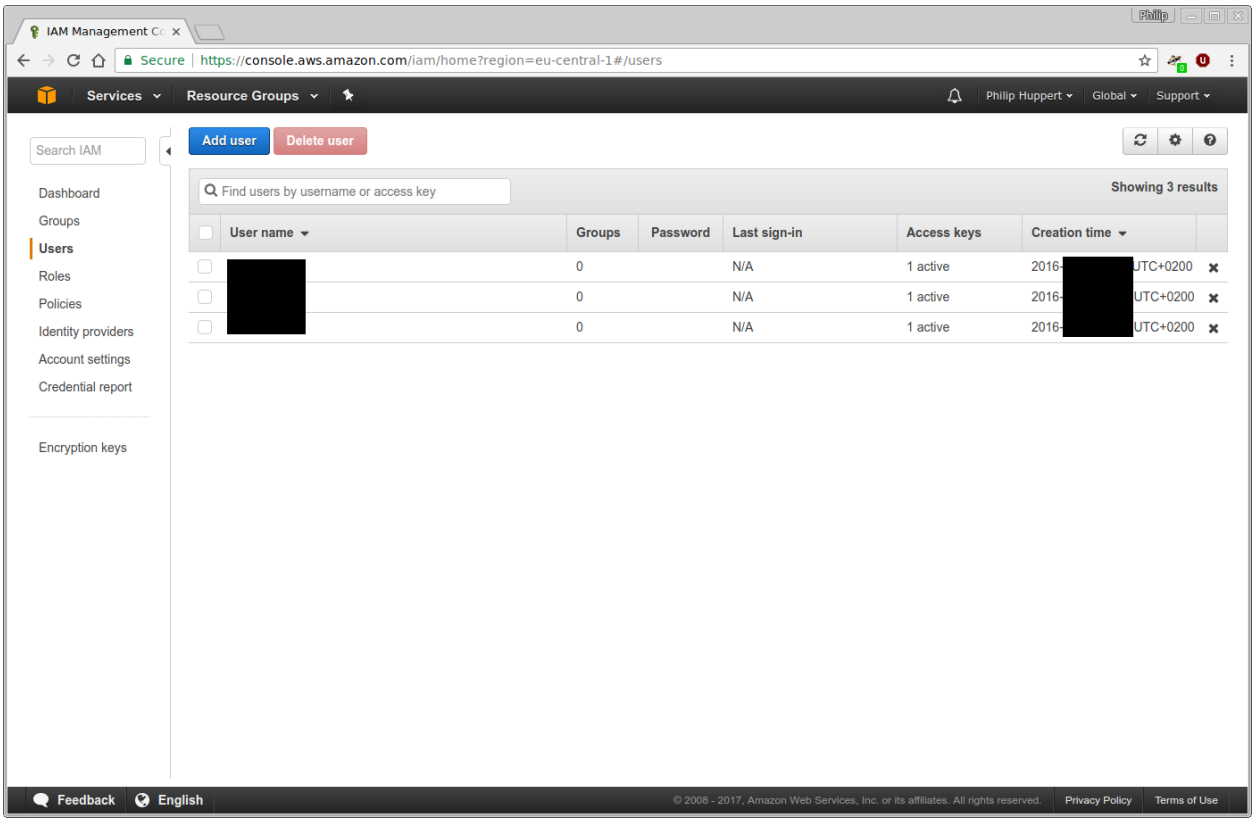


10.1.5 Creating a user

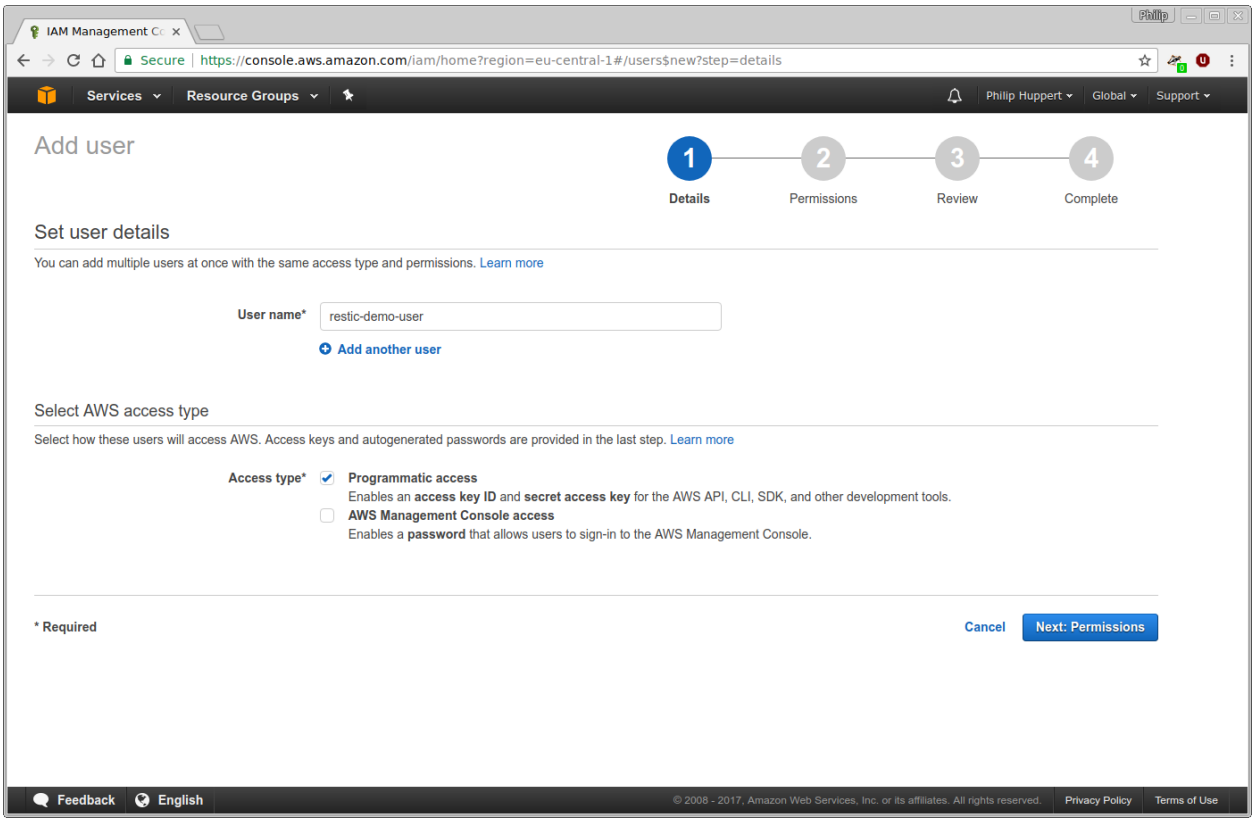
Use the "Services" menu of the AWS web interface to navigate to IAM. This will bring you to the IAM homepage. To create a new user, click on the "Users" menu entry on the left:



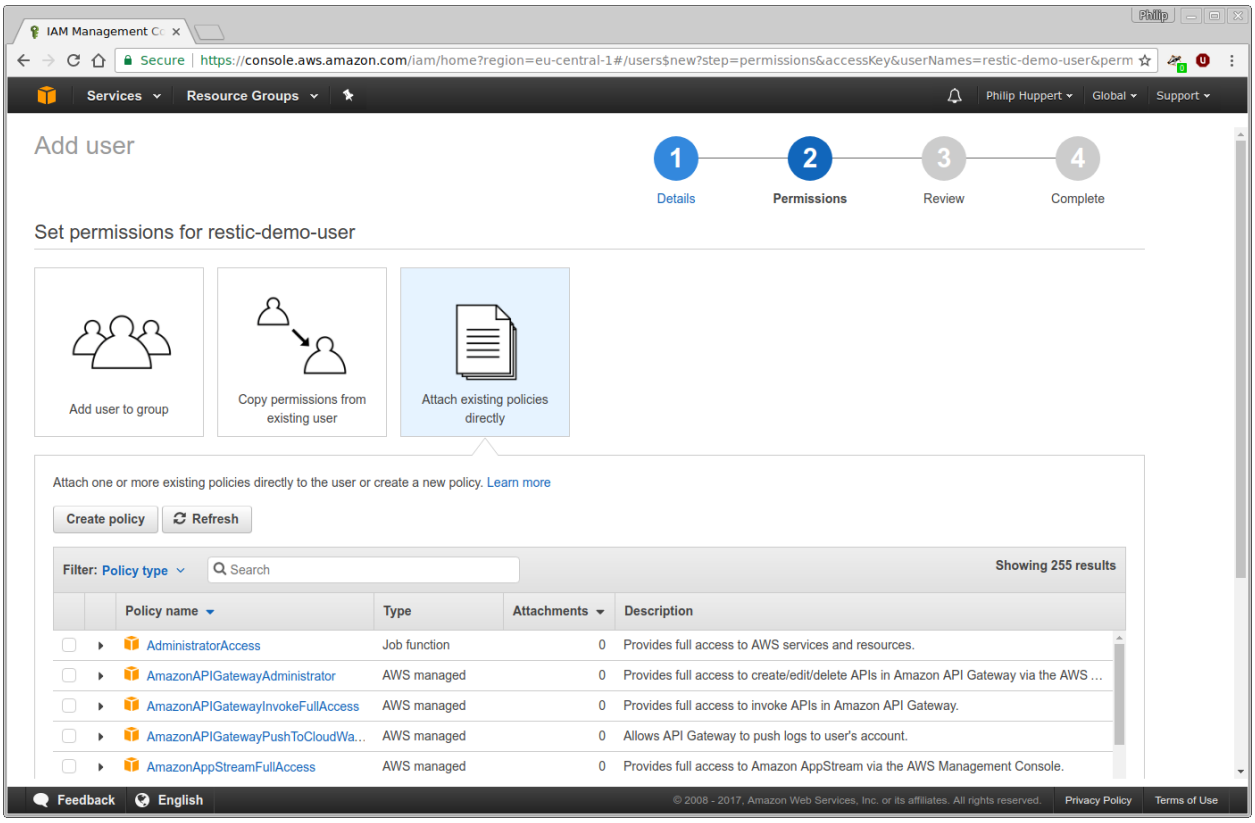
In case you already have set-up users with IAM before, you will see a list of them here. Use the "Add user" button at the top to create a new user:



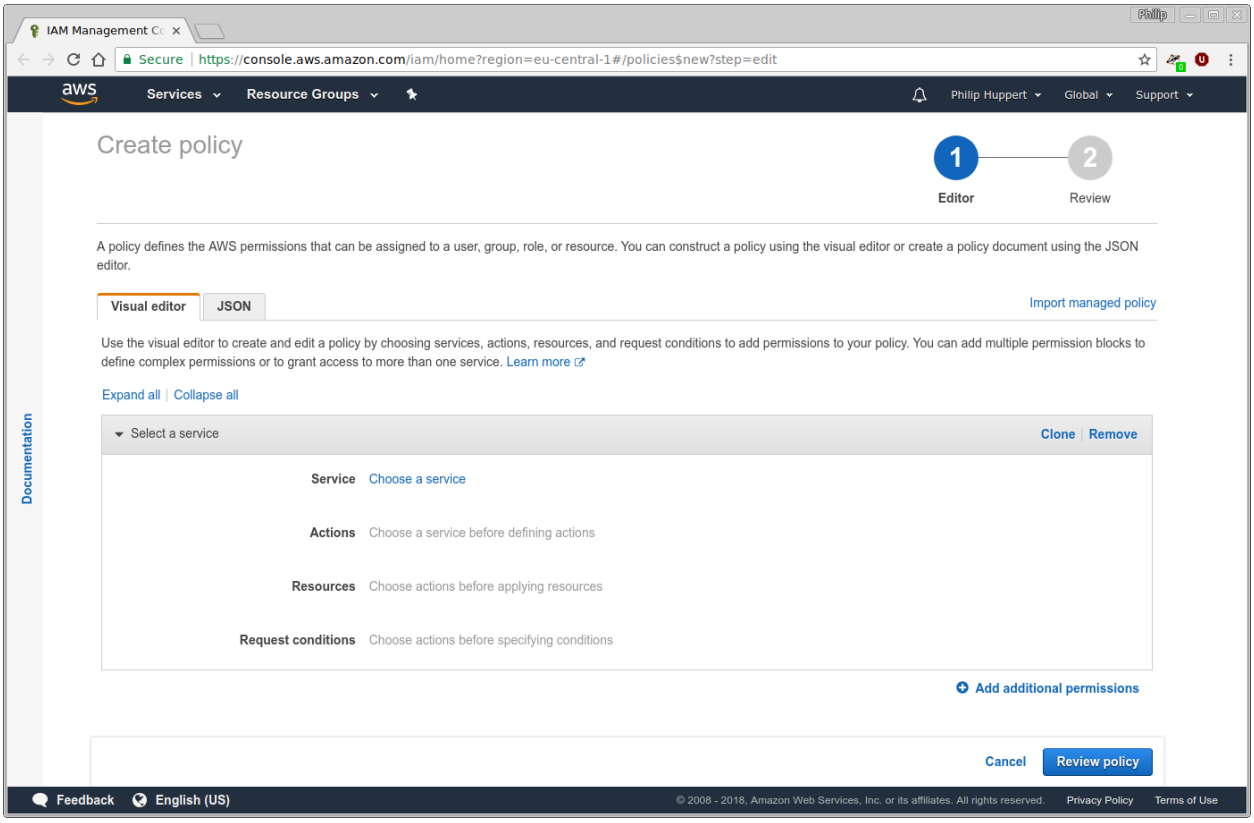
For this tutorial, the new user will be named `restic-demo-user`. Feel free to choose your own name that best fits your needs. This user will only ever access AWS through the `restic` program and not through the web interface. Therefore, "Programmatic access" is selected for "Access type":



During the next step, permissions can be assigned to the new user. To use this user with restic, it only needs access to the `restic-demo` bucket. Select "Attach existing policies directly", which will bring up a list of pre-defined policies below. Afterwards, click the "Create policy" button to create a custom policy:



A new browser window or tab will open with the policy wizard. In Amazon IAM, policies are defined as JSON documents. For this tutorial, the "Visual editor" will be used to generate a policy:



For restic to work, two permission statements must be created using the visual policy editor. The first statement is set up as follows:

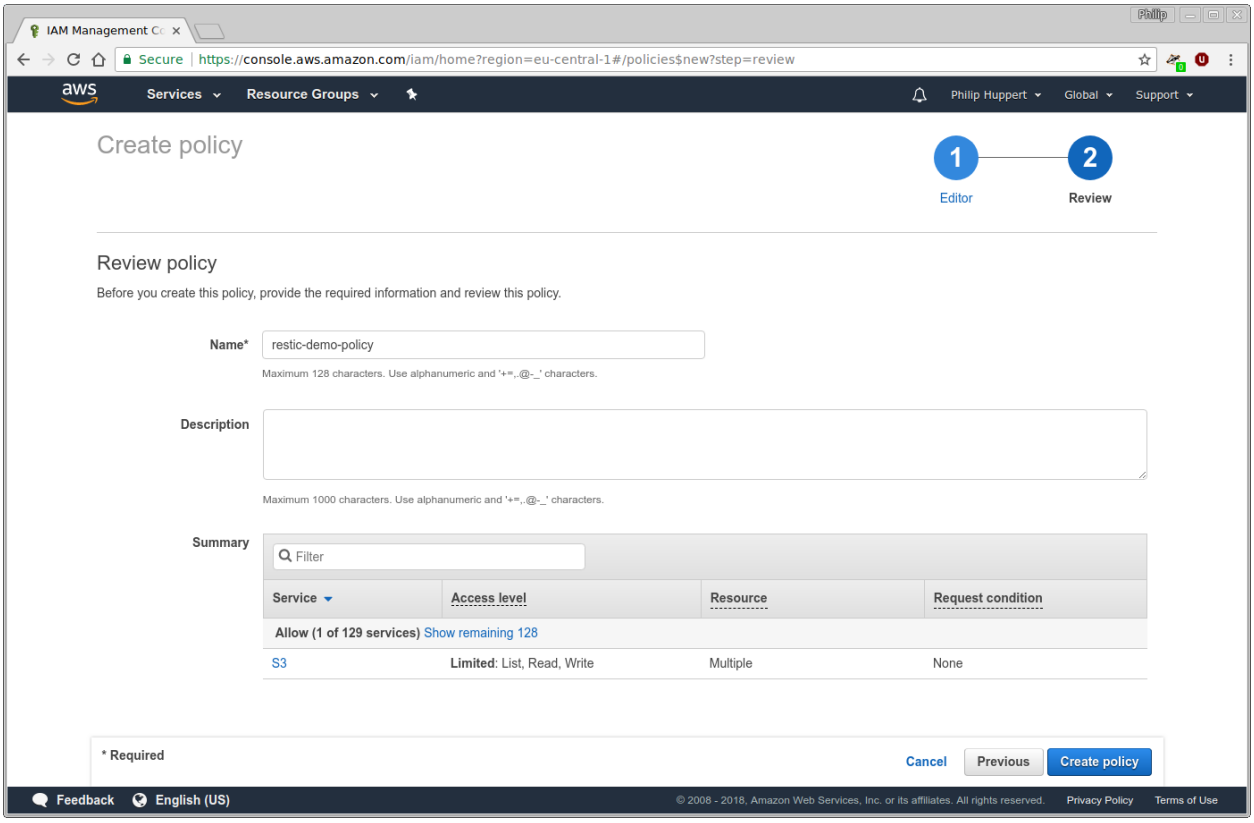
```
Service: S3
Allow Actions: DeleteObject, GetObject, PutObject
Resources: arn:aws:s3:::restic-demo/*
```

This statement allows restic to create, read and delete objects inside the S3 bucket named `restic-demo`. Adjust the bucket's name to the name of the bucket you created earlier. Next, add a second statement using the "Add additional permissions" button:

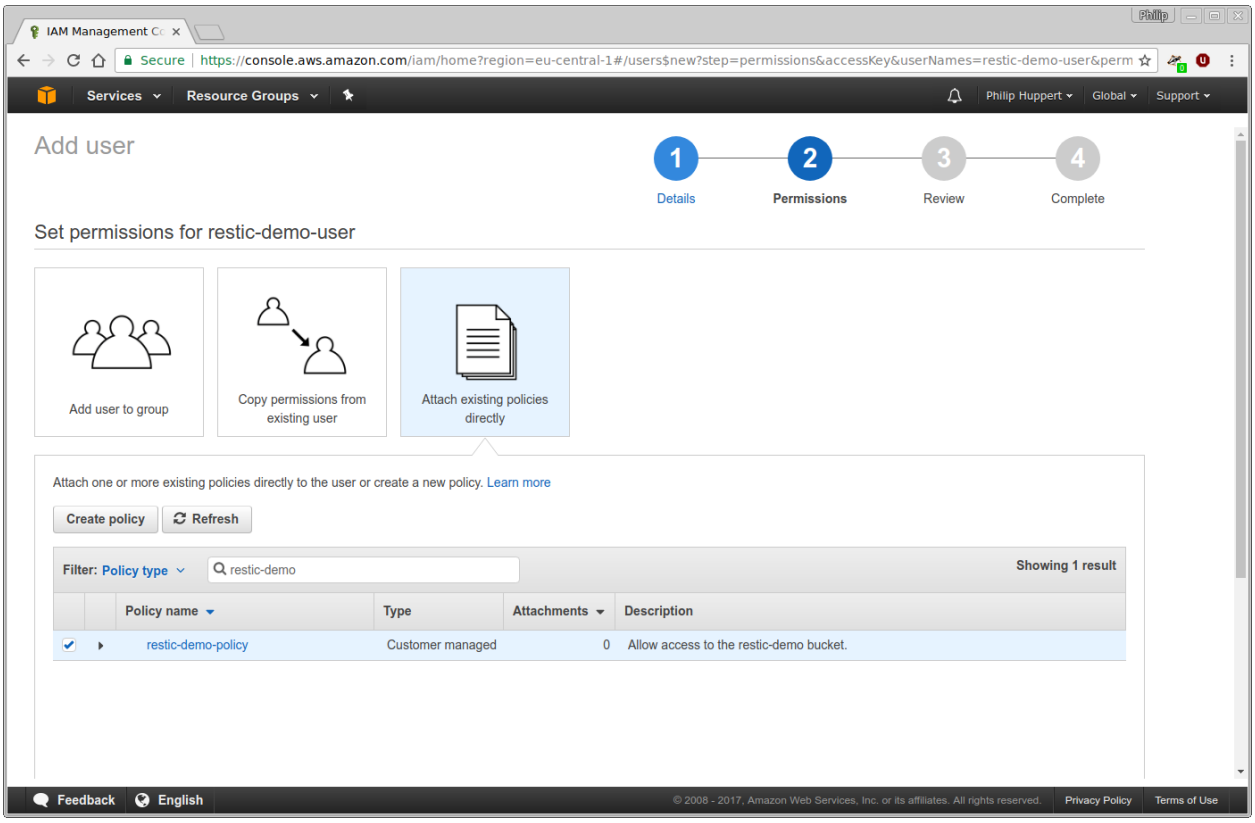
```
Service: S3
Allow Actions: ListBucket, GetBucketLocation
Resource: arn:aws:s3:::restic-demo
```

Again, substitute `restic-demo` with the actual name of your bucket. Note that, unlike before, there is no `/*` after the bucket name. This statement allows restic to list the objects stored in the `restic-demo` bucket and to query the bucket's region.

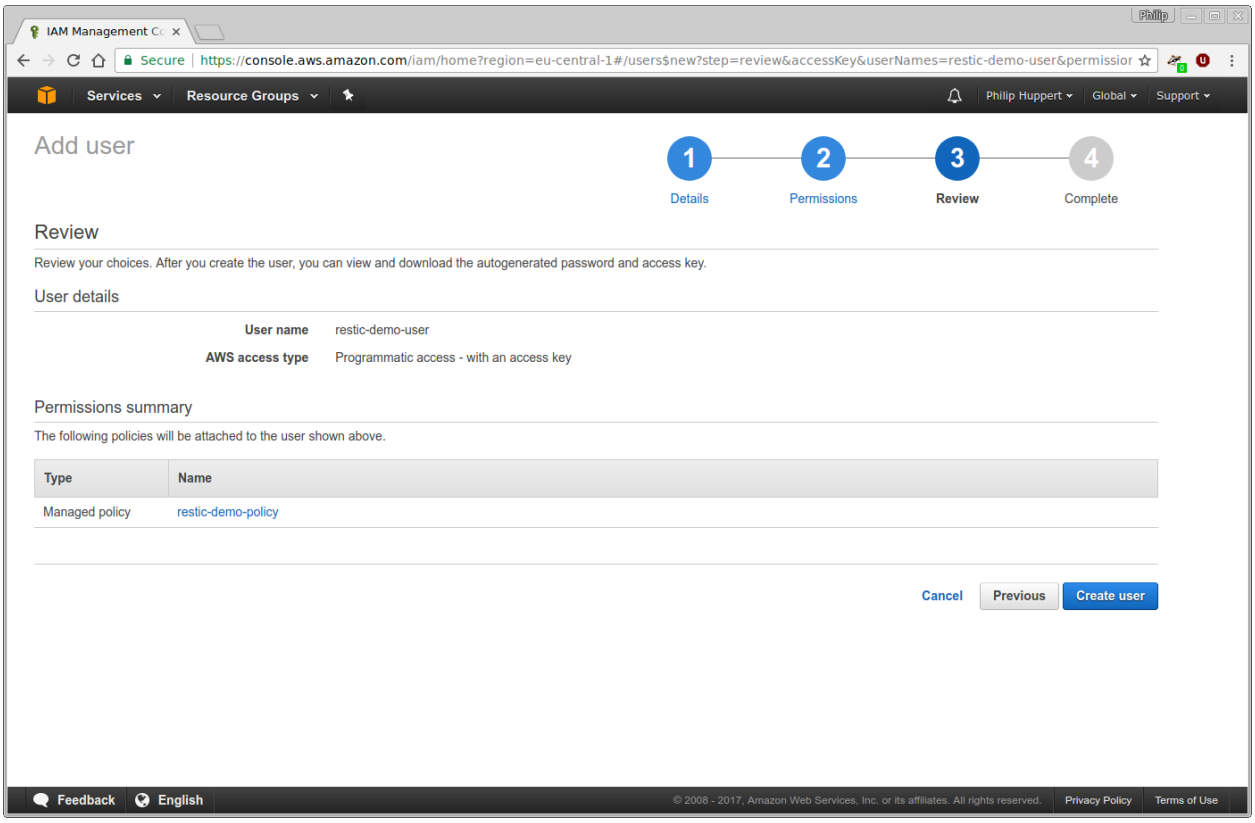
Continue to the next step by clicking the "Review policy" button and enter a name and description for this policy. For this tutorial, the policy will be named `restic-demo-policy`. Click "Create policy" to finish the process:



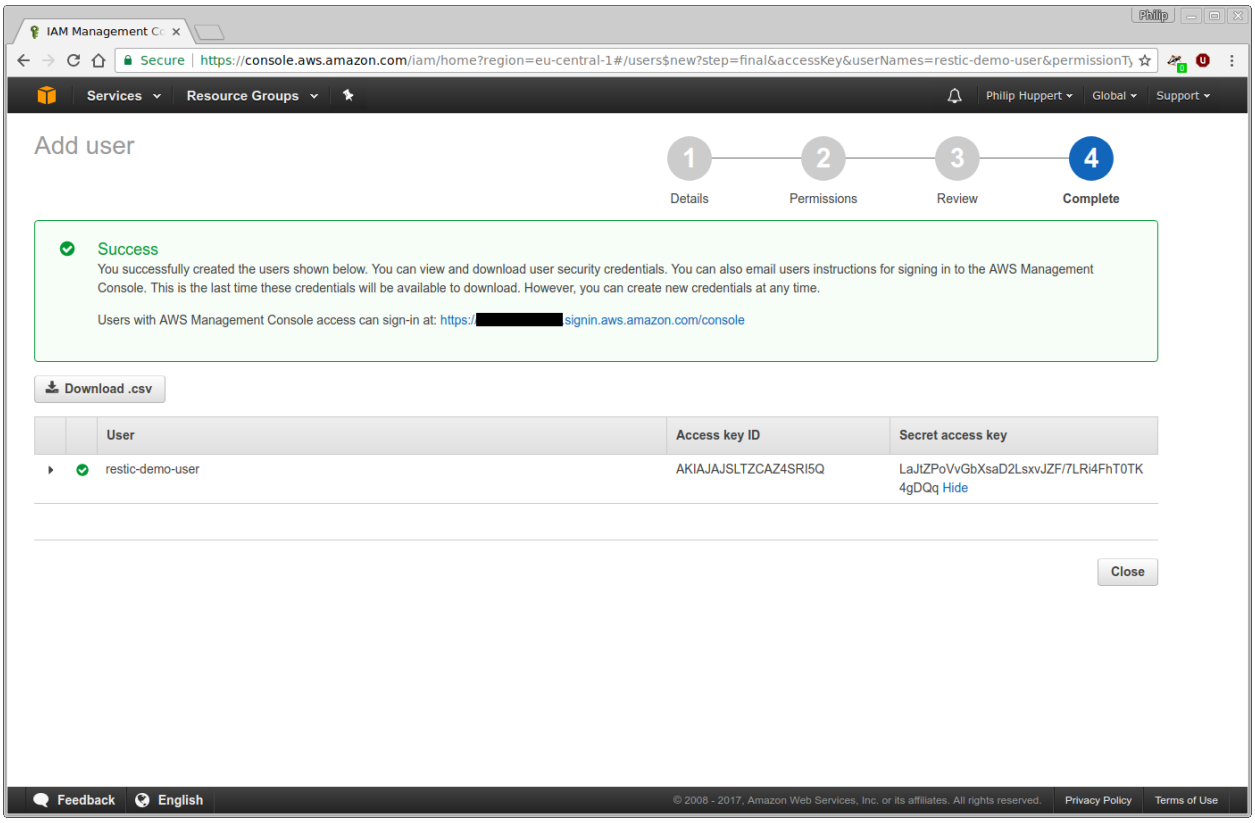
Go back to the browser window or tab where you were previously creating the new user. Click the button labeled "Refresh" above the list of policies to make sure the newly created policy is available to you. Afterwards, use the search function to search for the `restic-demo-policy`. Select this policy using the checkbox on the left. Then, continue to the next step.



The next page will present an overview of the user account that is about to be created. If everything looks good, click "Create user" to complete the process:



After the user has been created, its access credentials will be displayed. They consist of the "Access key ID" (think user name), and the "Secret access key" (think password). Copy these down to a safe place.



You have now completed the configuration in AWS. Feel free to close your web browser now.

10.1.6 Initializing the restic repository

Open a terminal and make sure you have the `restic` binary ready. First, choose a password to encrypt your backups with. In this tutorial, `apg` is used for this purpose:

```
$ apg -a 1 -m 32 -n 1 -M NCL
I9n7G7G0ZpDWA3GOcJbIuwQCGvGUBkU5
```

Note this password somewhere safe along with your AWS credentials. Next, the configuration of restic will be placed into environment variables. This will include sensitive information, such as your AWS secret and repository password. Therefore, make sure the next commands **do not** end up in your shell's history file. Adjust the contents of the environment variables to fit your bucket's name and your user's API credentials.

```
$ unset HISTFILE
$ export RESTIC_REPOSITORY="s3:https://s3.amazonaws.com/restic-demo"
$ export AWS_ACCESS_KEY_ID="AKIAJAJSLTZCAZ4SRI5Q"
$ export AWS_SECRET_ACCESS_KEY="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"
$ export RESTIC_PASSWORD="I9n7G7G0ZpDWA3GOcJbIuwQCGvGUBkU5"
```

After the environment is set up, restic may be called to initialize the repository:

```
$ ./restic init
created restic backend b5c661a86a at s3:https://s3.amazonaws.com/restic-demo

Please note that knowledge of your password is required to access
```

(continues on next page)

(continued from previous page)

```
the repository. Losing your password means that your data is
irrecoverably lost.
```

restic is now ready to be used with AWS S3. Try to create a backup:

```
$ dd if=/dev/urandom bs=1M count=10 of=test.bin
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0,0891322 s, 118 MB/s

$ ./restic backup test.bin
scan [/home/philip/restic-demo/test.bin]
scanned 0 directories, 1 files in 0:00
[0:04] 100.00%  2.500 MiB/s  10.000 MiB / 10.000 MiB  1 / 1 items ... ETA 0:00
duration: 0:04, 2.47MiB/s
snapshot 10fdbace saved

$ ./restic snapshots
ID        Date                 Host        Tags        Directory
----------------------------------------------------------------------
10fdbace  2017-03-26 16:41:50  blackbox                /home/philip/restic-demo/test.
↪bin
```

A snapshot was created and stored in the S3 bucket. This snapshot may now be restored:

```
$ mkdir restore

$ ./restic restore 10fdbace --target restore
restoring <Snapshot 10fdbace of [/home/philip/restic-demo/test.bin] at 2017-03-26␣
↪16:41:50.201418102 +0200 CEST by philip@blackbox> to restore

$ ls restore/
test.bin
```

The snapshot was successfully restored. This concludes the tutorial.

10.2 Backing up your system without running restic as root

10.2.1 Motivation

Creating a complete backup of a machine requires a privileged process that is able to read all files. On UNIX-like systems this is traditionally the `root` user. Processes running as root have superpower. They cannot only read all files but do also have the power to modify the system in any possible way.

With great power comes great responsibility. If a process running as root malfunctions, is exploited, or simply configured in a wrong way it can cause any possible damage to the system. This means you only want to run programs as root that you trust completely. And even if you trust a program, it is good and common practice to run it with the least possible privileges.

10.2.2 Capabilities on Linux

Fortunately, Linux has functionality to divide root's power into single separate *capabilities*. You can remove these from a process running as root to restrict it. And you can add capabilities to a process running as a normal user, which

is what we are going to do.

10.2.3 Full backup without root

To be able to completely backup a system, restic has to read all the files. Luckily Linux knows a capability that allows precisely this. We can assign this single capability to restic and then run it as an unprivileged user.

First we create a new user called `restic` that is going to create the backups:

```
root@a3e580b6369d:/# useradd -m restic
```

Then we download and install the restic binary into the user's home directory.

```
root@a3e580b6369d:/# mkdir ~restic/bin
root@a3e580b6369d:/# curl -L https://github.com/restic/restic/releases/download/v0.8.
↪0/restic_0.8.0_linux_amd64.bz2 | bunzip2 > ~restic/bin/restic
```

Before we assign any special capability to the restic binary we restrict its permissions so that only root and the newly created restic user can execute it. Otherwise another - possibly untrusted - user could misuse the privileged restic binary to circumvent file access controls.

```
root@a3e580b6369d:/# chown root:restic ~restic/bin/restic
root@a3e580b6369d:/# chmod 750 ~restic/bin/restic
```

Finally we can use `setcap` to add an extended attribute to the restic binary. On every execution the system will read the extended attribute, interpret it and assign capabilities accordingly.

```
root@a3e580b6369d:/# setcap cap_dac_read_search=+ep ~restic/bin/restic
```

From now on the user `restic` can run restic to backup the whole system.

```
root@a3e580b6369d:/# sudo -u restic /opt/restic/bin/restic --exclude={/dev,/media,/
↪mnt,/proc,/run,/sys,/tmp,/var/tmp} -r /tmp backup /
```

CHAPTER 11

Participating

11.1 Debugging

The program can be built with debug support like this:

```
$ go run build.go -tags debug
```

Afterwards, extensive debug messages are written to the file in environment variable `DEBUG_LOG`, e.g.:

```
$ DEBUG_LOG=/tmp/restic-debug.log restic backup ~/work
```

If you suspect that there is a bug, you can have a look at the debug log. Please be aware that the debug log might contain sensitive information such as file and directory names.

The debug log will always contain all log messages restic generates. You can also instruct restic to print some or all debug messages to stderr. These can also be limited to e.g. a list of source files or a list of patterns for function names. The patterns are globbing patterns (see the documentation for path.Glob), multiple patterns are separated by commas. Patterns are case sensitive.

Printing all log messages to the console can be achieved by setting the file filter to `*`:

```
$ DEBUG_FILES=* restic check
```

If you want restic to just print all debug log messages from the files `main.go` and `lock.go`, set the environment variable `DEBUG_FILES` like this:

```
$ DEBUG_FILES=main.go,lock.go restic check
```

The following command line instructs restic to only print debug statements originating in functions that match the pattern `*unlock*` (case sensitive):

```
$ DEBUG_FUNCS=*unlock* restic check
```

11.2 Contributing

Contributions are welcome! Please **open an issue first** (or add a comment to an existing issue) if you plan to work on any code or add a new feature. This way, duplicate work is prevented and we can discuss your ideas and design first.

More information and a description of the development environment can be found in CONTRIBUTING.md. A document describing the design of restic and the data structures stored on the back end is contained in Design.

If you'd like to start contributing to restic, but don't know exactly what do to, have a look at this great article by Dave Cheney: Suggestions for contributing to an Open Source project A few issues have been tagged with the label `help wanted`, you can start looking at those: https://github.com/restic/restic/labels/help%20wanted

11.3 Security

Important: If you discover something that you believe to be a possible critical security problem, please do *not* open a GitHub issue but send an email directly to alexander@bumpern.de. If possible, please encrypt your email using the following PGP key (0x91A6868BD3F7A907):

```
pub   4096R/91A6868BD3F7A907 2014-11-01
      Key fingerprint = CF8F 18F2 8445 7597 3F79  D4E1 91A6 868B D3F7 A907
      uid                          Alexander Neumann <alexander@bumpern.de>
      sub   4096R/D5FC2ACF4043FDF1 2014-11-01
```

11.4 Compatibility

Backward compatibility for backups is important so that our users are always able to restore saved data. Therefore restic follows Semantic Versioning to clearly define which versions are compatible. The repository and data structures contained therein are considered the "Public API" in the sense of Semantic Versioning. This goes for all released versions of restic, this may not be the case for the master branch.

We guarantee backward compatibility of all repositories within one major version; as long as we do not increment the major version, data can be read and restored. We strive to be fully backward compatible to all prior versions.

11.5 Building documentation

The restic documentation is built with Sphinx, therefore building it locally requires a recent Python version and requirements listed in `doc/requirements.txt`. This example will guide you through the process using virtualenv:

```
$ virtualenv venv # create virtual python environment
$ source venv/bin/activate # activate the virtual environment
$ cd doc
$ pip install -r requirements.txt # install dependencies
$ make html # build html documentation
$ # open _build/html/index.html with your favorite browser
```

CHAPTER 12

References

12.1 Design

12.1.1 Terminology

This section introduces terminology used in this document.

Repository: All data produced during a backup is sent to and stored in a repository in a structured form, for example in a file system hierarchy with several subdirectories. A repository implementation must be able to fulfill a number of operations, e.g. list the contents.

Blob: A Blob combines a number of data bytes with identifying information like the SHA-256 hash of the data and its length.

Pack: A Pack combines one or more Blobs, e.g. in a single file.

Snapshot: A Snapshot stands for the state of a file or directory that has been backed up at some point in time. The state here means the content and meta data like the name and modification time for the file or the directory and its contents.

Storage ID: A storage ID is the SHA-256 hash of the content stored in the repository. This ID is required in order to load the file from the repository.

12.1.2 Repository Format

All data is stored in a restic repository. A repository is able to store data of several different types, which can later be requested based on an ID. This so-called "storage ID" is the SHA-256 hash of the content of a file. All files in a repository are only written once and never modified afterwards. This allows accessing and even writing to the repository with multiple clients in parallel. Only the `prune` operation removes data from the repository.

Repositories consist of several directories and a top-level file called `config`. For all other files stored in the repository, the name for the file is the lower case hexadecimal representation of the storage ID, which is the SHA-256 hash of the file's contents. This allows for easy verification of files for accidental modifications, like disk read errors, by simply running the program `sha256sum` on the file and comparing its output to the file name. If the prefix of a filename is unique amongst all the other files in the same directory, the prefix may be used instead of the complete filename.

Apart from the files stored within the `keys` directory, all files are encrypted with AES-256 in counter mode (CTR). The integrity of the encrypted data is secured by a Poly1305-AES message authentication code (sometimes also referred to as a "signature").

In the first 16 bytes of each encrypted file the initialisation vector (IV) is stored. It is followed by the encrypted data and completed by the 16 byte MAC. The format is: `IV || CIPHERTEXT || MAC`. The complete encryption overhead is 32 bytes. For each file, a new random IV is selected.

The file `config` is encrypted this way and contains a JSON document like the following:

```
{
  "version": 1,
  "id": "5956a3f67a6230d4a92cefb29529f10196c7d92582ec305fd71ff6d331d6271b",
  "chunker_polynomial": "25b468838dcb75"
}
```

After decryption, restic first checks that the version field contains a version number that it understands, otherwise it aborts. At the moment, the version is expected to be 1. The field `id` holds a unique ID which consists of 32 random bytes, encoded in hexadecimal. This uniquely identifies the repository, regardless if it is accessed via SFTP or locally. The field `chunker_polynomial` contains a parameter that is used for splitting large files into smaller chunks (see below).

Repository Layout

The `local` and `sftp` backends are implemented using files and directories stored in a file system. The directory layout is the same for both backend types.

The basic layout of a repository stored in a `local` or `sftp` backend is shown here:

```
/tmp/restic-repo
├── config
├── data
│   ├── 21
│   │   └── 2159dd48f8a24f33c307b750592773f8b71ff8d11452132a7b2e2a6a01611be1
│   ├── 32
│   │   └── 32ea976bc30771cebad8285cd99120ac8786f9ffd42141d452458089985043a5
│   ├── 59
│   │   └── 59fe4bcde59bd6222eba87795e35a90d82cd2f138a27b6835032b7b58173a426
│   ├── 73
│   │   └── 73d04e6125cf3c28a299cc2f3cca3b78ceac396e4fcf9575e34536b26782413c
│   [...]
├── index
│   ├── c38f5fb68307c6a3e3aa945d556e325dc38f5fb68307c6a3e3aa945d556e325d
│   └── ca171b1b7394d90d330b265d90f506f9984043b342525f019788f97e745c71fd
├── keys
│   └── b02de829beeb3c01a63e6b25cbd421a98fef144f03b9a02e46eff9e2ca3f0bd7
├── locks
├── snapshots
│   └── 22a5af1bdc6e616f8a29579458c49627e01b32210d09adb288d1ecda7c5711ec
└── tmp
```

A local repository can be initialized with the `restic init` command, e.g.:

```
$ restic -r /tmp/restic-repo init
```

The local and sftp backends will auto-detect and accept all layouts described in the following sections, so that remote repositories mounted locally e.g. via fuse can be accessed. The layout auto-detection can be overridden by specifying

the option `-o local.layout=default`, valid values are `default` and `s3legacy`. The option for the sftp backend is named `sftp.layout`, for the s3 backend `s3.layout`.

S3 Legacy Layout

Unfortunately during development the AWS S3 backend uses slightly different paths (directory names use singular instead of plural for `key`, `lock`, and `snapshot` files), and the data files are stored directly below the `data` directory. The S3 Legacy repository layout looks like this:

```
/config
/data
  ├── 2159dd48f8a24f33c307b750592773f8b71ff8d11452132a7b2e2a6a01611be1
  ├── 32ea976bc30771cebad8285cd99120ac8786f9ffd42141d452458089985043a5
  ├── 59fe4bcde59bd6222eba87795e35a90d82cd2f138a27b6835032b7b58173a426
  ├── 73d04e6125cf3c28a299cc2f3cca3b78ceac396e4fcf9575e34536b26782413c
[...]
/index
  ├── c38f5fb68307c6a3e3aa945d556e325dc38f5fb68307c6a3e3aa945d556e325d
  └── ca171b1b7394d90d330b265d90f506f9984043b342525f019788f97e745c71fd
/key
  └── b02de829beeb3c01a63e6b25cbd421a98fef144f03b9a02e46eff9e2ca3f0bd7
/lock
/snapshot
  └── 22a5af1bdc6e616f8a29579458c49627e01b32210d09adb288d1ecda7c5711ec
```

The S3 backend understands and accepts both forms, new backends are always created with the default layout for compatibility reasons.

12.1.3 Pack Format

All files in the repository except Key and Pack files just contain raw data, stored as `IV || Ciphertext || MAC`. Pack files may contain one or more Blobs of data.

A Pack's structure is as follows:

```
EncryptedBlob1 || ... || EncryptedBlobN || EncryptedHeader || Header_Length
```

At the end of the Pack file is a header, which describes the content. The header is encrypted and authenticated. `Header_Length` is the length of the encrypted header encoded as a four byte integer in little-endian encoding. Placing the header at the end of a file allows writing the blobs in a continuous stream as soon as they are read during the backup phase. This reduces code complexity and avoids having to re-write a file once the pack is complete and the content and length of the header is known.

All the blobs (`EncryptedBlob1`, `EncryptedBlobN` etc.) are authenticated and encrypted independently. This enables repository reorganisation without having to touch the encrypted Blobs. In addition it also allows efficient indexing, for only the header needs to be read in order to find out which Blobs are contained in the Pack. Since the header is authenticated, authenticity of the header can be checked without having to read the complete Pack.

After decryption, a Pack's header consists of the following elements:

```
Type_Blob1 || Length(EncryptedBlob1) || Hash(Plaintext_Blob1) ||
[...]
Type_BlobN || Length(EncryptedBlobN) || Hash(Plaintext_Blobn) ||
```

This is enough to calculate the offsets for all the Blobs in the Pack. Length is the length of a Blob as a four byte integer in little-endian format. The type field is a one byte field and labels the content of a blob according to the following table:

Type	Meaning
0	data
1	tree

All other types are invalid, more types may be added in the future.

For reconstructing the index or parsing a pack without an index, first the last four bytes must be read in order to find the length of the header. Afterwards, the header can be read and parsed, which yields all plaintext hashes, types, offsets and lengths of all included blobs.

12.1.4 Indexing

Index files contain information about Data and Tree Blobs and the Packs they are contained in and store this information in the repository. When the local cached index is not accessible any more, the index files can be downloaded and used to reconstruct the index. The files are encrypted and authenticated like Data and Tree Blobs, so the outer structure is `IV || Ciphertext || MAC` again. The plaintext consists of a JSON document like the following:

```
{
  "supersedes": [
    "ed54ae36197f4745ebc4b54d10e0f623eaaaedd03013eb7ae90df881b7781452"
  ],
  "packs": [
    {
      "id": "73d04e6125cf3c28a299cc2f3cca3b78ceac396e4fcf9575e34536b26782413c",
      "blobs": [
        {
          "id": "3ec79977ef0cf5de7b08cd12b874cd0f62bbaf7f07f3497a5b1bbcc8cb39b1ce",
          "type": "data",
          "offset": 0,
          "length": 25
        },{
          "id": "9ccb846e60d90d4eb915848add7aa7ea1e4bbabfc60e573db9f7bfb2789afbae",
          "type": "tree",
          "offset": 38,
          "length": 100
        },
        {
          "id": "d3dc577b4ffd38cc4b32122cabf8655a0223ed22edfd93b353dc0c3f2b0fdf66",
          "type": "data",
          "offset": 150,
          "length": 123
        }
      ]
    }, [...]
  ]
}
```

This JSON document lists Packs and the blobs contained therein. In this example, the Pack `73d04e61` contains two data Blobs and one Tree blob, the plaintext hashes are listed afterwards.

The field `supersedes` lists the storage IDs of index files that have been replaced with the current index file. This happens when index files are repacked, for example when old snapshots are removed and Packs are recombined.

There may be an arbitrary number of index files, containing information on non-disjoint sets of Packs. The number of packs described in a single file is chosen so that the file size is kept below 8 MiB.

12.1.5 Keys, Encryption and MAC

All data stored by restic in the repository is encrypted with AES-256 in counter mode and authenticated using Poly1305-AES. For encrypting new data first 16 bytes are read from a cryptographically secure pseudorandom number generator as a random nonce. This is used both as the IV for counter mode and the nonce for Poly1305. This operation needs three keys: A 32 byte for AES-256 for encryption, a 16 byte AES key and a 16 byte key for Poly1305. For details see the original paper The Poly1305-AES message-authentication code by Dan Bernstein. The data is then encrypted with AES-256 and afterwards a message authentication code (MAC) is computed over the ciphertext, everything is then stored as IV ‖ CIPHERTEXT ‖ MAC.

The directory `keys` contains key files. These are simple JSON documents which contain all data that is needed to derive the repository's master encryption and message authentication keys from a user's password. The JSON document from the repository can be pretty-printed for example by using the Python module `json` (shortened to increase readability):

```
$ python -mjson.tool /tmp/restic-repo/keys/b02de82*
{
    "hostname": "kasimir",
    "username": "fd0"
    "kdf": "scrypt",
    "N": 65536,
    "r": 8,
    "p": 1,
    "created": "2015-01-02T18:10:13.48307196+01:00",
    "data": "tGwYeKoM0C4j4/9DFrVEmMGAldvEn/+iKC3te/QE/6ox/V4qz58FUOgMa0Bb1cIJ6asrypCx/
↪Ti/
↪pRXCPHLDkIJbNYd2ybC+fLhFIJVLCvkMS+trdywsUkglUbTbi+7+Ldsul5jpAj9vTZ25ajDc+4FKtWEcCWL5ICAOoTAxnPgT+Lk
↪",
    "salt": "uW4fEI1+IOzj7ED9mVor+yTSJFd68DGlGOeLgJELYsTU5ikhG/83/
↪+jGd4KKAaQdSrsfzrdOhAMftTSih5Ux6w==",
}
```

When the repository is opened by restic, the user is prompted for the repository password. This is then used with `scrypt`, a key derivation function (KDF), and the supplied parameters (`N`, `r`, `p` and `salt`) to derive 64 key bytes. The first 32 bytes are used as the encryption key (for AES-256) and the last 32 bytes are used as the message authentication key (for Poly1305-AES). These last 32 bytes are divided into a 16 byte AES key `k` followed by 16 bytes of secret key `r`. The key `r` is then masked for use with Poly1305 (see the paper for details).

Those keys are used to authenticate and decrypt the bytes contained in the JSON field `data` with AES-256 and Poly1305-AES as if they were any other blob (after removing the Base64 encoding). If the password is incorrect or the key file has been tampered with, the computed MAC will not match the last 16 bytes of the data, and restic exits with an error. Otherwise, the data yields a JSON document which contains the master encryption and message authentication keys for this repository (encoded in Base64). The command `restic cat masterkey` can be used as follows to decrypt and pretty-print the master key:

```
$ restic -r /tmp/restic-repo cat masterkey
{
    "mac": {
      "k": "evFWd9wWlndL9jc501268g==",
      "r": "E9eEDnSJZgqwTOkDtOp+Dw=="
    },
    "encrypt": "UQCqa0lKZ94PygPxMRqkePTZnHRYh1k1pX2k2lM2v3Q=",
}
```

All data in the repository is encrypted and authenticated with these master keys. For encryption, the AES-256 algorithm in Counter mode is used. For message authentication, Poly1305-AES is used as described above.

A repository can have several different passwords, with a key file for each. This way, the password can be changed without having to re-encrypt all data.

12.1.6 Snapshots

A snapshot represents a directory with all files and sub-directories at a given point in time. For each backup that is made, a new snapshot is created. A snapshot is a JSON document that is stored in an encrypted file below the directory `snapshots` in the repository. The filename is the storage ID. This string is unique and used within restic to uniquely identify a snapshot.

The command `restic cat snapshot` can be used as follows to decrypt and pretty-print the contents of a snapshot file:

```
$ restic -r /tmp/restic-repo cat snapshot 251c2e58
enter password for repository:
{
  "time": "2015-01-02T18:10:50.895208559+01:00",
  "tree": "2da81727b6585232894cfbb8f8bdab8d1eccd3d8f7c92bc934d62e62e618ffdf",
  "dir": "/tmp/testdata",
  "hostname": "kasimir",
  "username": "fd0",
  "uid": 1000,
  "gid": 100,
  "tags": [
    "NL"
  ]
}
```

Here it can be seen that this snapshot represents the contents of the directory `/tmp/testdata`. The most important field is `tree`. When the meta data (e.g. the tags) of a snapshot change, the snapshot needs to be re-encrypted and saved. This will change the storage ID, so in order to relate these seemingly different snapshots, a field `original` is introduced which contains the ID of the original snapshot, e.g. after adding the tag `DE` to the snapshot above it becomes:

```
$ restic -r /tmp/restic-repo cat snapshot 22a5af1b
enter password for repository:
{
  "time": "2015-01-02T18:10:50.895208559+01:00",
  "tree": "2da81727b6585232894cfbb8f8bdab8d1eccd3d8f7c92bc934d62e62e618ffdf",
  "dir": "/tmp/testdata",
  "hostname": "kasimir",
  "username": "fd0",
  "uid": 1000,
  "gid": 100,
  "tags": [
    "NL",
    "DE"
  ],
  "original": "251c2e5841355f743f9d4ffd3260bee765acee40a6229857e32b60446991b837"
}
```

Once introduced, the `original` field is not modified when the snapshot's meta data is changed again.

All content within a restic repository is referenced according to its SHA-256 hash. Before saving, each file is split into

variable sized Blobs of data. The SHA-256 hashes of all Blobs are saved in an ordered list which then represents the content of the file.

In order to relate these plaintext hashes to the actual location within a Pack file , an index is used. If the index is not available, the header of all data Blobs can be read.

12.1.7 Trees and Data

A snapshot references a tree by the SHA-256 hash of the JSON string representation of its contents. Trees and data are saved in pack files in a subdirectory of the directory `data`.

The command `restic cat blob` can be used to inspect the tree referenced above (piping the output of the command to `jq .` so that the JSON is indented):

```
$ restic -r /tmp/restic-repo cat blob␣
↪2da81727b6585232894cfbb8f8bdab8d1eccd3d8f7c92bc934d62e62e618ffdf | jq .
enter password for repository:
{
  "nodes": [
    {
      "name": "testdata",
      "type": "dir",
      "mode": 493,
      "mtime": "2014-12-22T14:47:59.912418701+01:00",
      "atime": "2014-12-06T17:49:21.748468803+01:00",
      "ctime": "2014-12-22T14:47:59.912418701+01:00",
      "uid": 1000,
      "gid": 100,
      "user": "fd0",
      "inode": 409704562,
      "content": null,
      "subtree": "b26e315b0988ddcd1cee64c351d13a100fedbc9fdbb144a67d1b765ab280b4dc"
    }
  ]
}
```

A tree contains a list of entries (in the field `nodes`) which contain meta data like a name and timestamps. When the entry references a directory, the field `subtree` contains the plain text ID of another tree object.

When the command `restic cat blob` is used, the plaintext ID is needed to print a tree. The tree referenced above can be dumped as follows:

```
$ restic -r /tmp/restic-repo cat blob␣
↪b26e315b0988ddcd1cee64c351d13a100fedbc9fdbb144a67d1b765ab280b4dc
enter password for repository:
{
  "nodes": [
    {
      "name": "testfile",
      "type": "file",
      "mode": 420,
      "mtime": "2014-12-06T17:50:23.34513538+01:00",
      "atime": "2014-12-06T17:50:23.338468713+01:00",
      "ctime": "2014-12-06T17:50:23.34513538+01:00",
      "uid": 1000,
      "gid": 100,
      "user": "fd0",
```

(continues on next page)

(continued from previous page)

```
      "inode": 416863351,
      "size": 1234,
      "links": 1,
      "content": [
        "50f77b3b4291e8411a027b9f9b9e64658181cc676ce6ba9958b95f268cb1109d"
      ]
    },
    [...]
  ]
}
```

This tree contains a file entry. This time, the `subtree` field is not present and the `content` field contains a list with one plain text SHA-256 hash.

The command `restic cat blob` can also be used to extract and decrypt data given a plaintext ID, e.g. for the data mentioned above:

```
$ restic -r /tmp/restic-repo cat blob␣
↪50f77b3b4291e8411a027b9f9b9e64658181cc676ce6ba9958b95f268cb1109d | sha256sum
enter password for repository:
50f77b3b4291e8411a027b9f9b9e64658181cc676ce6ba9958b95f268cb1109d  -
```

As can be seen from the output of the program `sha256sum`, the hash matches the plaintext hash from the map included in the tree above, so the correct data has been returned.

12.1.8 Locks

The restic repository structure is designed in a way that allows parallel access of multiple instance of restic and even parallel writes. However, there are some functions that work more efficient or even require exclusive access of the repository. In order to implement these functions, restic processes are required to create a lock on the repository before doing anything.

Locks come in two types: Exclusive and non-exclusive locks. At most one process can have an exclusive lock on the repository, and during that time there must not be any other locks (exclusive and non-exclusive). There may be multiple non-exclusive locks in parallel.

A lock is a file in the subdir `locks` whose filename is the storage ID of the contents. It is encrypted and authenticated the same way as other files in the repository and contains the following JSON structure:

```
{
  "time": "2015-06-27T12:18:51.759239612+02:00",
  "exclusive": false,
  "hostname": "kasimir",
  "username": "fd0",
  "pid": 13607,
  "uid": 1000,
  "gid": 100
}
```

The field `exclusive` defines the type of lock. When a new lock is to be created, restic checks all locks in the repository. When a lock is found, it is tested if the lock is stale, which is the case for locks with timestamps older than 30 minutes. If the lock was created on the same machine, even for younger locks it is tested whether the process is still alive by sending a signal to it. If that fails, restic assumes that the process is dead and considers the lock to be stale.

When a new lock is to be created and no other conflicting locks are detected, restic creates a new lock, waits, and checks if other locks appeared in the repository. Depending on the type of the other locks and the lock to be created,

restic either continues or fails.

12.1.9 Backups and Deduplication

For creating a backup, restic scans the source directory for all files, sub-directories and other entries. The data from each file is split into variable length Blobs cut at offsets defined by a sliding window of 64 byte. The implementation uses Rabin Fingerprints for implementing this Content Defined Chunking (CDC). An irreducible polynomial is selected at random and saved in the file `config` when a repository is initialized, so that watermark attacks are much harder.

Files smaller than 512 KiB are not split, Blobs are of 512 KiB to 8 MiB in size. The implementation aims for 1 MiB Blob size on average.

For modified files, only modified Blobs have to be saved in a subsequent backup. This even works if bytes are inserted or removed at arbitrary positions within the file.

12.1.10 Threat Model

The design goals for restic include being able to securely store backups in a location that is not completely trusted, e.g. a shared system where others can potentially access the files or (in the case of the system administrator) even modify or delete them.

General assumptions:

- The host system a backup is created on is trusted. This is the most basic requirement, and essential for creating trustworthy backups.

The restic backup program guarantees the following:

- Accessing the unencrypted content of stored files and metadata should not be possible without a password for the repository. Everything except the metadata included for informational purposes in the key files is encrypted and authenticated.
- Modifications (intentional or unintentional) can be detected automatically on several layers:
    1. For all accesses of data stored in the repository it is checked whether the cryptographic hash of the contents matches the storage ID (the file's name). This way, modifications (bad RAM, broken harddisk) can be detected easily.
    2. Before decrypting any data, the MAC on the encrypted data is checked. If there has been a modification, the MAC check will fail. This step happens even before the data is decrypted, so data that has been tampered with is not decrypted at all.

However, the restic backup program is not designed to protect against attackers deleting files at the storage location. There is nothing that can be done about this. If this needs to be guaranteed, get a secure location without any access from third parties. If you assume that attackers have write access to your files at the storage location, attackers are able to figure out (e.g. based on the timestamps of the stored files) which files belong to what snapshot. When only these files are deleted, the particular snapshot vanished and all snapshots depending on data that has been added in the snapshot cannot be restored completely. Restic is not designed to detect this attack.

12.2 Local Cache

In order to speed up certain operations, restic manages a local cache of data. This document describes the data structures for the local cache with version 1.

12.2.1 Versions

The cache directory is selected according to the XDG base dir specification. Each repository has its own cache sub-directory, consting of the repository ID which is chosen at `init`. All cache directories for different repos are independent of each other.

The cache dir for a repo contains a file named `version`, which contains a single ASCII integer line that stands for the current version of the cache. If a lower version number is found the cache is recreated with the current version. If a higher version number is found the cache is ignored and left as is.

12.2.2 Snapshots, Data and Indexes

Snapshot, Data and Index files are cached in the sub-directories `snapshots`, `data` and `index`, as read from the repository.

12.2.3 Expiry

Whenever a cache directory for a repo is used, that directory's modification timestamp is updated to the current time. By looking at the modification timestamps of the repo cache directories it is easy to decide which directories are old and haven't been used in a long time. Those are probably stale and can be removed.

12.3 REST Backend

Restic can interact with HTTP Backend that respects the following REST API.

The following values are valid for `{type}`:

- `data`
- `keys`
- `locks`
- `snapshots`
- `index`
- `config`

The API version is selected via the `Accept` HTTP header in the request. The following values are defined:

- `application/vnd.x.restic.rest.v1` or empty: Select API version 1
- `application/vnd.x.restic.rest.v2`: Select API version 2

The server will respond with the value of the highest version it supports in the `Content-Type` HTTP response header for the HTTP requests which should return JSON. Any different value for this header means API version 1.

The placeholder `{path}` in this document is a path to the repository, so that multiple different repositories can be accessed. The default path is `/`. The path must end with a slash.

12.3.1 POST {path}?create=true

This request is used to initially create a new repository. The server responds with "200 OK" if the repository structure was created successfully or already exists, otherwise an error is returned.

12.3.2 DELETE {path}

Deletes the repository on the server side. The server responds with "200 OK" if the repository was successfully removed. If this function is not implemented the server returns "501 Not Implemented", if this it is denied by the server it returns "403 Forbidden".

12.3.3 HEAD {path}/config

Returns "200 OK" if the repository has a configuration, an HTTP error otherwise.

12.3.4 GET {path}/config

Returns the content of the configuration file if the repository has a configuration, an HTTP error otherwise.

Response format: binary/octet-stream

12.3.5 POST {path}/config

Returns "200 OK" if the configuration of the request body has been saved, an HTTP error otherwise.

12.3.6 GET {path}/{type}/

API version 1

Returns a JSON array containing the names of all the blobs stored for a given type, example:

```
[
  "245bc4c430d393f74fbe7b13325e30dbde9fb0745e50caad57c446c93d20096b",
  "85b420239efa1132c41cea0065452a40ebc20c6f8e0b132a5b2f5848360973ec",
  "8e2006bb5931a520f3c7009fe278d1ebb87eb72c3ff92a50c30e90f1b8cf3e60",
  "e75c8c407ea31ba399ab4109f28dd18c4c68303d8d86cc275432820c42ce3649"
]
```

API version 2

Returns a JSON array containing an object for each file of the given type. The objects have two keys: `name` for the file name, and `size` for the size in bytes.

```
[
  {
    "name": "245bc4c430d393f74fbe7b13325e30dbde9fb0745e50caad57c446c93d20096b",
    "size": 2341058
  },
  {
    "name": "85b420239efa1132c41cea0065452a40ebc20c6f8e0b132a5b2f5848360973ec",
    "size": 2908900
  },
  {
    "name": "8e2006bb5931a520f3c7009fe278d1ebb87eb72c3ff92a50c30e90f1b8cf3e60",
    "size": 3030712
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "name": "e75c8c407ea31ba399ab4109f28dd18c4c68303d8d86cc275432820c42ce3649",
    "size": 2804
  }
]
```

12.3.7 HEAD {path}/{type}/{name}

Returns "200 OK" if the blob with the given name and type is stored in the repository, "404 not found" otherwise. If the blob exists, the HTTP header `Content-Length` is set to the file size.

12.3.8 GET {path}/{type}/{name}

Returns the content of the blob with the given name and type if it is stored in the repository, "404 not found" otherwise.

If the request specifies a partial read with a Range header field, then the status code of the response is 206 instead of 200 and the response only contains the specified range.

Response format: binary/octet-stream

12.3.9 POST {path}/{type}/{name}

Saves the content of the request body as a blob with the given name and type, an HTTP error otherwise.

Request format: binary/octet-stream

12.3.10 DELETE {path}/{type}/{name}

Returns "200 OK" if the blob with the given name and type has been deleted from the repository, an HTTP error otherwise.

CHAPTER 13

Talks

The following talks will be or have been given about restic:

- 2016-01-31: Lightning Talk at the Go Devroom at FOSDEM 2016, Brussels, Belgium
- 2016-01-29: restic - Backups mal richtig: Public lecture in German at CCC Cologne e.V. in Cologne, Germany
- 2015-08-23: A Solution to the Backup Inconvenience: Lecture at FROSCON 2015 in Bonn, Germany
- 2015-02-01: Lightning Talk at FOSDEM 2015: A short introduction (with slightly outdated command line)
- 2015-01-27: Talk about restic at CCC Aachen (in German)

CHAPTER 14

FAQ

This is the list of Frequently Asked Questions for restic.

14.1 `restic check` reports packs that aren't referenced in any index, is my repository broken?

When `restic check` reports that there are pack files in the repository that are not referenced in any index, that's (in contrast to what restic reports at the moment) not a source for concern. The output looks like this:

```
$ restic check
Create exclusive lock for repository
Load indexes
Check all packs
pack 819a9a52e4f51230afa89aefbf90df37fb70996337ae57e6f7a822959206a85e: not referenced␣
↪in any index
pack de299e69fb075354a3775b6b045d152387201f1cdc229c31d1caa34c3b340141: not referenced␣
↪in any index
Check snapshots, trees and blobs
Fatal: repository contains errors
```

The message means that there is more data stored in the repo than strictly necessary. With high probability this is duplicate data. In order to clean it up, the command `restic prune` can be used. The cause of this bug is not yet known.

14.2 How can I specify encryption passwords automatically?

When you run `restic backup`, you need to enter the passphrase on the console. This is not very convenient for automated backups, so you can also provide the password through the `--password-file` option, or one of the environment variables `RESTIC_PASSWORD` or `RESTIC_PASSWORD_FILE`. A discussion is in progress over implementing unattended backups happens in #533.

Important: Be careful how you set the environment; using the env command, a *system()* call or using inline shell scripts (e.g. *RESTIC_PASSWORD=password restic ...*) might expose the credentials in the process list directly and they will be readable to all users on a system. Using export in a shell script file should be safe, however, as the environment of a process is accessible only to that user. Please make sure that the permissions on the files where the password is eventually stored are safe (e.g. *0600* and owned by root).

14.3 How to prioritize restic's IO and CPU time

If you'd like to change the **IO priority** of restic, run it in the following way

```
$ ionice -c2 -n0 ./restic -r /media/your/backup/ backup /home
```

This runs `restic` in the so-called best *effort class* (`-c2`), with the highest possible priority (`-n0`).

Take a look at the ionice manpage to learn about the other classes.

To change the **CPU scheduling priority** to a higher-than-standard value, use would run:

```
$ nice --10 ./restic -r /media/your/backup/ backup /home
```

Again, the nice manpage has more information.

You can also **combine IO and CPU scheduling priority**:

```
$ ionice -c2 nice -n19 ./restic -r /media/gour/backup/ backup /home
```

This example puts restic in the IO class 2 (best effort) and tells the CPU scheduling algorithm to give it the least favorable niceness (19).

The above example makes sure that the system the backup runs on is not slowed down, which is particularly useful for servers.

14.4 Creating new repo on a Synology NAS via sftp fails

Sometimes creating a new restic repository on a Synology NAS via sftp fails with an error similar to the following:

```
$ restic init -r sftp:user@nas:/volume1/restic-repo init
create backend at sftp:user@nas:/volume1/restic-repo/ failed:
    mkdirAll(/volume1/restic-repo/index): unable to create directories: [...]
```

Although you can log into the NAS via SSH and see that the directory structure is there.

The reason for this behavior is that apparently Synology NAS expose a different directory structure via sftp, so the path that needs to be specified is different than the directory structure on the device and maybe even as exposed via other protocols.

Try removing the /volume1 prefix in your paths. If this does not work, use sftp and ls to explore the SFTP file system hierarchy on your NAS.

The following may work:

```
$ restic init -r sftp:user@nas:/restic-repo init
```

CHAPTER 15

Manual

15.1 Usage help

Usage help is available:

```
$ ./restic --help
restic is a backup program which allows saving multiple revisions of files and
directories in an encrypted repository stored on different backends.

Usage:
  restic [command]

Available Commands:
  backup        Create a new backup of files and/or directories
  cat           Print internal objects to stdout
  check         Check the repository for errors
  diff          Show differences between two snapshots
  dump          Print a backed-up file to stdout
  find          Find a file or directory
  forget        Remove snapshots from the repository
  generate      Generate manual pages and auto-completion files (bash, zsh)
  help          Help about any command
  init          Initialize a new repository
  key           Manage keys (passwords)
  list          List objects in the repository
  ls            List files in a snapshot
  migrate       Apply migrations
  mount         Mount the repository
  prune         Remove unneeded data from the repository
  rebuild-index Build a new index file
  restore       Extract the data from a snapshot
  snapshots     List all snapshots
  tag           Modify tags on snapshots
  unlock        Remove locks other processes created
```

(continues on next page)

(continued from previous page)

```
  version       Print version information

Flags:
      --cacert stringSlice       path to load root certificates from (default: use␣
↪system certificates)
      --cache-dir string         set the cache directory
      --cleanup-cache            auto remove old cache directories
  -h, --help                     help for restic
      --json                     set output mode to JSON for commands that support it
      --limit-download int       limits downloads to a maximum rate in KiB/s.␣
↪(default: unlimited)
      --limit-upload int         limits uploads to a maximum rate in KiB/s. (default:␣
↪unlimited)
      --no-cache                 do not use a local cache
      --no-lock                  do not lock the repo, this allows some operations on␣
↪read-only repos
  -o, --option key=value         set extended option (key=value, can be specified␣
↪multiple times)
  -p, --password-file string     read the repository password from a file (default:
↪$RESTIC_PASSWORD_FILE)
  -q, --quiet                    do not output comprehensive progress report
  -r, --repo string              repository to backup to or restore from (default:
↪$RESTIC_REPOSITORY)
      --tls-client-cert string   path to a file containing PEM encoded TLS client␣
↪certificate and private key
  -v, --verbose count[=-1]       be verbose (can be specified multiple times)

Use "restic [command] --help" for more information about a command.
```

Similar to programs such as `git`, restic has a number of sub-commands. You can see these commands in the listing above. Each sub-command may have own command-line options, and there is a help option for each command which lists them, e.g. for the `backup` command:

```
$ ./restic backup --help
The "backup" command creates a new snapshot and saves the files and directories
given as the arguments.

Usage:
  restic backup [flags] FILE/DIR [FILE/DIR] ...

Flags:
  -e, --exclude pattern                  exclude a pattern (can be specified multiple␣
↪times)
      --exclude-caches                   excludes cache directories that are marked␣
↪with a CACHEDIR.TAG file
      --exclude-file file                read exclude patterns from a file (can be␣
↪specified multiple times)
      --exclude-if-present stringArray   takes filename[:header], exclude contents of␣
↪directories containing filename (except filename itself) if header of that file is␣
↪as provided (can be specified multiple times)
      --files-from string                read the files to backup from file (can be␣
↪combined with file args)
  -f, --force                            force re-reading the target files/
↪directories (overrides the "parent" flag)
  -h, --help                             help for backup
      --hostname hostname                set the hostname for the snapshot manually.␣
↪To prevent an expensive rescan use the "parent" flag
```

(continues on next page)

(continued from previous page)

```
  -x, --one-file-system                  exclude other file systems
      --parent string                    use this parent snapshot (default: last␣
↪snapshot in the repo that has the same target files/directories)
      --stdin                            read backup from stdin
      --stdin-filename string            file name to use when reading from stdin␣
↪(default "stdin")
      --tag tag                          add a tag for the new snapshot (can be␣
↪specified multiple times)
      --time string                      time of the backup (ex. '2012-11-01 22:08:41
↪') (default: now)
      --with-atime                       store the atime for all files and directories

Global Flags:
      --cacert stringSlice       path to load root certificates from (default: use␣
↪system certificates)
      --cache-dir string         set the cache directory
      --cleanup-cache            auto remove old cache directories
      --json                     set output mode to JSON for commands that support it
      --limit-download int       limits downloads to a maximum rate in KiB/s.␣
↪(default: unlimited)
      --limit-upload int         limits uploads to a maximum rate in KiB/s. (default:␣
↪unlimited)
      --no-cache                 do not use a local cache
      --no-lock                  do not lock the repo, this allows some operations on␣
↪read-only repos
  -o, --option key=value         set extended option (key=value, can be specified␣
↪multiple times)
  -p, --password-file string     read the repository password from a file (default:
↪$RESTIC_PASSWORD_FILE)
  -q, --quiet                    do not output comprehensive progress report
  -r, --repo string              repository to backup to or restore from (default:
↪$RESTIC_REPOSITORY)
      --tls-client-cert string   path to a file containing PEM encoded TLS client␣
↪certificate and private key
  -v, --verbose n[=-1]           be verbose (specify --verbose multiple times or␣
↪level n)
```

Subcommand that support showing progress information such as `backup`, `check` and `prune` will do so unless the quiet flag `-q` or `--quiet` is set. When running from a non-interactive console progress reporting will be limited to once every 10 seconds to not fill your logs. Use `backup` with the quiet flag `-q` or `--quiet` to skip the initial scan of the source directory, this may shorten the backup time needed for large directories.

Additionally on Unix systems if `restic` receives a SIGUSR1 signal the current progress will be written to the standard output so you can check up on the status at will.

15.2 Manage tags

Managing tags on snapshots is done with the `tag` command. The existing set of tags can be replaced completely, tags can be added or removed. The result is directly visible in the `snapshots` command.

Let's say we want to tag snapshot `590c8fc8` with the tags `NL` and `CH` and remove all other tags that may be present, the following command does that:

```
$ restic -r /srv/restic-repo tag --set NL --set CH 590c8fc8
create exclusive lock for repository
modified tags on 1 snapshots
```

Note the snapshot ID has changed, so between each change we need to look up the new ID of the snapshot. But there is an even better way, the `tag` command accepts `--tag` for a filter, so we can filter snapshots based on the tag we just added.

So we can add and remove tags incrementally like this:

```
$ restic -r /srv/restic-repo tag --tag NL --remove CH
create exclusive lock for repository
modified tags on 1 snapshots

$ restic -r /srv/restic-repo tag --tag NL --add UK
create exclusive lock for repository
modified tags on 1 snapshots

$ restic -r /srv/restic-repo tag --tag NL --remove NL
create exclusive lock for repository
modified tags on 1 snapshots

$ restic -r /srv/restic-repo tag --tag NL --add SOMETHING
no snapshots were modified
```

15.3 Under the hood

15.3.1 Browse repository objects

Internally, a repository stores data of several different types described in the design documentation. You can `list` objects such as blobs, packs, index, snapshots, keys or locks with the following command:

```
$ restic -r /srv/restic-repo list snapshots
d369ccc7d126594950bf74f0a348d5d98d9e99f3215082eb69bf02dc9b3e464c
```

The `find` command searches for a given pattern in the repository.

```
$ restic -r backup find test.txt
debug log file restic.log
debug enabled
enter password for repository:
found 1 matching entries in snapshot␣
↪196bc5760c909a7681647949e80e5448e276521489558525680acf1bd428af36
  -rw-r--r--   501    20      5 2015-08-26 14:09:57 +0200 CEST path/to/test.txt
```

The `cat` command allows you to display the JSON representation of the objects or their raw content.

```
$ restic -r /srv/restic-repo cat snapshot␣
↪d369ccc7d126594950bf74f0a348d5d98d9e99f3215082eb69bf02dc9b3e464c
enter password for repository:
{
  "time": "2015-08-12T12:52:44.091448856+02:00",
  "tree": "05cec17e8d3349f402576d02576a2971fc0d9f9776ce2f441c7010849c4ff5af",
  "paths": [
```

(continues on next page)

(continued from previous page)

```
    "/home/user/work"
  ],
  "hostname": "kasimir",
  "username": "username",
  "uid": 501,
  "gid": 20
}
```

15.3.2 Metadata handling

Restic saves and restores most default attributes, including extended attributes like ACLs. Sparse files are not handled in a special way yet, and aren't restored.

The following metadata is handled by restic:

- Name
- Type
- Mode
- ModTime
- AccessTime
- ChangeTime
- UID
- GID
- User
- Group
- Inode
- Size
- Links
- LinkTarget
- Device
- Content
- Subtree
- ExtendedAttributes

15.4 Scripting

Restic supports the output of some commands in JSON format, the JSON data can then be processed by other programs (e.g. jq). The following example lists all snapshots as JSON and uses `jq` to pretty-print the result:

```
$ restic -r /srv/restic-repo snapshots --json | jq .
[
  {
```

(continues on next page)

(continued from previous page)


```
    "time": "2017-03-11T09:57:43.26630619+01:00",
    "tree": "bf25241679533df554fc0fd0ae6dbb9dcf1859a13f2bc9dd4543c354eff6c464",
    "paths": [
      "/home/work/doc"
    ],
    "hostname": "kasimir",
    "username": "fd0",
    "uid": 1000,
    "gid": 100,
    "id": "bbeed6d28159aa384d1ccc6fa0b540644b1b9599b162d2972acda86b1b80f89e"
  },
  {
    "time": "2017-03-11T09:58:57.541446938+01:00",
    "tree": "7f8c95d3420baaac28dc51609796ae0e0ecfb4862b609a9f38ffaf7ae2d758da",
    "paths": [
      "/home/user/shared"
    ],
    "hostname": "kasimir",
    "username": "fd0",
    "uid": 1000,
    "gid": 100,
    "id": "b157d91c16f0ba56801ece3a708dfc53791fe2a97e827090d6ed9a69a6ebdca0"
  }
]
```


15.5 Temporary files

During some operations (e.g. `backup` and `prune`) restic uses temporary files to store data. These files will, by default, be saved to the system's temporary directory, on Linux this is usually located in `/tmp/`. The environment variable `TMPDIR` can be used to specify a different directory, e.g. to use the directory `/var/tmp/restic-tmp` instead of the default, set the environment variable like this:

```
$ export TMPDIR=/var/tmp/restic-tmp
$ restic -r /srv/restic-repo backup ~/work
```

15.6 Caching

Restic keeps a cache with some files from the repository on the local machine. This allows faster operations, since meta data does not need to be loaded from a remote repository. The cache is automatically created, usually in an OS-specific cache folder:

- Linux/other: `~/.cache/restic` (or `$XDG_CACHE_HOME/restic`)
- macOS: `~/Library/Caches/restic`
- Windows: `%LOCALAPPDATA%/restic`

The command line parameter `--cache-dir` can each be used to override the default cache location. The parameter `--no-cache` disables the cache entirely. In this case, all data is loaded from the repo.

The cache is ephemeral: When a file cannot be read from the cache, it is loaded from the repository.

Within the cache directory, there's a sub directory for each repository the cache was used with. Restic updates the timestamps of a repo directory each time it is used, so by looking at the timestamps of the sub directories of the cache

directory it can decide which sub directories are old and probably not needed any more. You can either remove these directories manually, or run a restic command with the `--cleanup-cache` flag.