`obnam-benchmark`—tool to run benchmarks

The Obnam project

2022-05-21 03:29

Contents

Chapter 1

Introduction

Obnam is a backup program. It aims to have good performance. To achieve that, we run benchmarks, and we automate that. To allow flexibility in what benchmark scenarios we have, we have a tool that reads benchmark specifications from a file and runs them. This document is the description of that tool, and its acceptance criteria.

Chapter 2

Overview

The `obnam-benchmark` tool runs a set of specified benchmarks for one version of Obnam. It works as follows:

1. Read the specification file.
2. Build the specified Obnam version.
3. Start the Obnam server with an empty chunk directory.
4. Run each specified benchmark, and take measurements.
5. Stop the Obnam server. Clean up.
6. Store measurements in a form that can be processed later.

A benchmark specification file is in YAML format. It has a list of individual benchmarks. Each benchmark has a name, and a description of how to prepare the test data for each backup generation. Example:

```
- benchmark: maildir
  backups:
  - changes:
    - create:
        files: 100000
        file_size: 0
    - rename:
        files: 1000
  - changes: []
- benchmark: video-footage
  backups:
  - changes:
    - create:
        files: 1000
        filee_size: 1G
  - changes: []
```

The example above specifies two benchmarks: "maildir", and "video-footage". The names are not interpreted by `obnam-benchmark`, they are for human consumption only. The `backups` field specifies a list of changes to the test data; the benchmark tool runs a backup for each item in the list. Each change can create, copy, delete, or rename files, compared to the previous backup. The created files will have the specified size, and the content is randomly generated, non-repetitive, binary junk. The files will be stored in a directory tree avoiding very large numbers of files per directory.

By specifying what kind of data is generated, and how it changes from backup to backup, and how many backups are made, the specification file can describe synthetic benchmarks for different use cases.

Chapter 3

Acceptance criteria

3.1 Generate random data for a backup

Requirement: The benchmark tool can generate random test data.

We verify this by having the benchmark generate some test data, using a subcommand for this purpose. If the data doesn't compress well, we assume it's sufficiently random. We can safely assume that the same code is used to generate test data for benchmarks, though of course the scenario can't verify that. Instead, it can be verified via manual code inspection.

given an installed Rust program obnam-benchmark
when I run **obnam-benchmark generate-junk 123000 junk.dat**
and I run **gzip junk.dat**
then file **junk.dat.gz** is at least **100100** bytes long

3.2 Parses a specification file

Requirement: The benchmark tool can parse a specification file.

We verify this by having the tool output a YAML specification file as JSON. Given the formats are different, this will check that the tool actually parses the file. We use an input file that contains aspects of benchmark specification that are not just YAML, with the assumption that the tool uses a YAML parser that works well, so that we only need to care about things that are specific to the benchmark tool.

Specifically, we verify that the tool parses file sizes such as "1K" correctly.

given an installed Rust program obnam-benchmark
and file **spec.yaml**
and file **expected.json**

when I run **obnam-benchmark spec spec.yaml --output spec.json**
then JSON files **spec.json** and **expected.json** match

File: **spec.yaml**

```yaml
benchmarks:
- benchmark: foo
  backups:
  - changes: []
```

File: **expected.json**

```json
{
    "benchmarks": [
        {
            "benchmark": "foo",
            "backups": [
                {
                    "changes": []
                }
            ]
        }
    ]
}
```

3.3 Smoke test to run a trivial benchmark

Requirement: The benchmark tool can benchmarks at all.

We verify this by running a trivial benchmark, which backs up an empty directory.

given an installed Rust program obnam-benchmark
and file **smoke.yaml**
when I run **obnam-benchmark run smoke.yaml --output smoke.json**
then file **smoke.json** is valid JSON

File: **smoke.yaml**

```yaml
benchmarks:
- benchmark: smoke
  backups:
  - changes: []
```

3.4 Benchmark with several backups

Requirement: The benchmark tool can benchmarks with more than one backup.

We verify this by running a benchmark with three backup generations.

given an installed Rust program obnam-benchmark
and file **three.yaml**
when I run **obnam-benchmark run three.yaml --output three.json**
then file **three.json** is valid JSON

File: **three.yaml**

```yaml
benchmarks:
- benchmark: three
  backups:
  - changes:
      - create:
          files: 1
          file_size: 1024
  - changes:
      - create:
          files: 2
          file_size: 2048
  - changes:
      - create:
          files: 4
          file_size: 4096
```