S5cmd Hits v1.0 and Intro to Advanced Usage

This blog originally appeared on Medium.com and has been republished with permission from ths author.

Object storage usage continues to broaden and move into applications and analytics that demand high performance. These use-cases also necessitate utilities to drive performance, both for working with object storage and benchmarking performance limits.

S5cmd, a versatile and high-performance utility for working with object storage, has reached a v1.0 milestone. This blog post is a follow up to my previous post on s5cmd and dives into more advanced usage and benchmarking with s5cmd. For basic usage, please refer to the github README or my previous post.

S5cmd continues to be a high performance command-line tool for interacting with an object store and automating tasks, so it pairs well with an all-flash object store like Pure Storage® FlashBlade®.

Version 1.0 of s5cmd introduced several breaking changes, new features, and the removal of some features. A particularly useful new feature in s5cmd v1.0 is the "cat" subcommand which outputs the content of an object to stdout, allowing you to more easily incorporate data on S3 into standard linux operations.

```
s5cmd cat s3://joshuarobinson/one.txt | grep something
```

Also, the previous flags to control concurrency "-uw" and "-dw" have been replaced by a single "–concurrency" flag, which controls the degree of concurrency for individual GET/PUT operations.

The rest of this article will cover the following topics:

- Building the s5cmd binary from source
- Expert mode: S5cmd's "run" option
- Ansible playbook for benchmarking with s5cmd

Building S5cmd From Source

In addition to installing directly, you may want to build and install a specific branch

or even modify the source for a custom build. The following is the multi-stage Dockerfile that I use to build and install the version 1.0 branch of s5cmd. With small modifications, this could also be used to build a branch with custom code. Note the extra steps here are the installation of “staticcheck” and “unparam” for compile-time checks. Because of the multi-stage build, the final image contains only the final binary and golang runtime.

```
FROM golang:1.13-alpine AS s5cmd-builder

ARG S5CMD_VERSION

RUN apk add build-base curl git make

RUN curl -L https://github.com/dominikh/go-tools/releases/download/2020.1.3/staticcheck_linux_amd64.tar.gz | tar xvz -C /opt/ \

 && cp /opt/staticcheck/staticcheck /go/bin/

RUN go get mvdan.cc/unparam

RUN git clone https://github.com/peak/s5cmd.git --branch=v$S5CMD_VERSION

RUN cd s5cmd && make

FROM golang:1.13-alpine

COPY --from=s5cmd-builder /go/s5cmd/s5cmd /go/bin/
```

The Dockerfile construction greatly impacts the resulting container image size. Using golang:1.13-buster results in images of 800MB, but with alpine it is 380MB. Even with the alpine image, if I do not use multi-stage builds, the resulting image is 900MB.

To build the s5cmd image at version 1.0.0 with above Dockerfile in the current directory:

```
docker build --build-arg S5CMD_VERSION=1.0.0 -t mys5cmd .
```

But if you do not need to build your own version of the tool, you can use my image on docker hub at joshuarobinson/s5cmd.

Expert Mode: “s5cmd run”

The most powerful feature of s5cmd is the “run” command (formerly “-f” flag) to pass a *list of commands* to s5cmd, which are then executed in parallel. Now you can use scripting and other tools to create a simple text file of commands, which can be a mix of all operations, and then s5cmd aggressively executes those concurrently. The “-numworkers” command-line argument controls how many operations to execute *in parallel* and defaults to 256.

As an example, if the run command file contains the following four lines, the result will be that s5cmd issues four commands in parallel, two PUTs and two DELETEs.

```
cp localfile1 s3://bucketname/path/dest1

cp localfile2 s3://bucketname/path/dest2

rm s3://bucketname/otherpath/thing1

rm s3://bucketname/otherpath/thing2
```

Even better, you can pipe those commands through stdin if no filename is specified after “run”, meaning you can construct the command list on the fly with shell commands and linux pipes.

To illustrate, the following operation is a sequence of two s5cmd invocations, the first to list objects and the second that listens for commands via stdin. In between are commands that convert the results of the LIST operation into commands that read objects to /dev/null. The ‘awk’ command extracts the last field in the object listing, which is the key of the object.

```
BUCKET=elastic-snapshots; s5cmd ls s3://$BUCKET/* | grep -v
DIR | awk ‘{print $NF}’ | xargs -I {} echo “cp s3://$BUCKET/{}
/dev/null” | s5cmd run
```

The result of the above command is a mini-benchmark: read all objects in a bucket as fast as possible. To better understand how this command works, run without the final “s5cmd run” to see the list of commands created.

I will use the “run” command for other purposes through this post, utilizing the flexibility of s5cmd’s “expert” mode.

Backing up a local filesystem with S5cmd

One common usage of s5cmd is to copy filesystem data to an object store, either for backup or for access with S3-enabled tools. This can be done with s5cmd as follows:

```
s5cmd cp -n -s -u /mnt/joshua/src/ s3://joshuarobinson/backup/
```

The “-n”, “-s”, “-u” options instruct s5cmd to only update the target if the source file has changed size or has a newer modification time than the target object.

Be aware though, that the logic currently fails if filenames contain characters that create invalid regular expressions.

More complicated cases can be solved with “s5cmd run.” For example, migrate old data to an object store by combining the “run” option with a linux ‘find’ command to only move files older than 30 days :

```
find /mnt/joshua/nachos/ -type f -mtime +30 | xargs -I{} echo
“mv {} s3://joshuarobinson/backup/{}” | s5cmd run
```

If the origin filesystem is an NFS mount with a large file count, consider using the Rapidfile toolkit to speedup the find operation with ‘pfind.’

Multi-Host Benchmarking: Ansible Playbook

A benchmark that approximates the real workloads placed on object storage requires multiple clients working in parallel. For coordinating s5cmd operations across multiple hosts, I use an Ansible playbook to test PUT/s followed by GET/s.

The full playbook can be found here on github.

The base requirements for running this playbook are: 1) ansible installed and configured on one machine, 2) host group created, 3) docker installed on all hosts, and 4) Ansible docker module installed. There are good example playbooks for how to install the necessary Docker software.

This playbook is designed for the numbers of “forks” to be as large as the host group so that each task is executed in parallel on all hosts. Ansible will also collect timing information for each task in the playbook with the following option in your ansible.cfg:

```
callback_whitelist = profile_tasks
```

Playbook Component 1: Data Generation

The type of data can impact performance, depending on if the storage system does inline compression, like the FlashBlade, or not. This means source data from /dev/zero results in unrealistic results and we instead want *representative* data in our tests. I use lzdatagen, a tool to create synthetic data that matches a target compressibility. I have created a public docker image with the lzdatagen binary.

To quickly test this utility locally, run with docker and my public image as follows:

```
> docker run -it --rm joshuarobinson/lzdatagen lzdgen --ratio 2.5 --size 1g - > testfile
```

As part of the Ansible playbook, the data generation task uses lzdgen with Ansible variables to allow modifying the compression ratio and object size:

```
lzdgen -f --ratio {{ compressibility_ratio }} --size {{ object_size }} /working/somebytes
```

This benchmark assumes the storage is NOT a deduplicating system and uses the same underlying data for each object PUT to the object store. The result requires only a small amount of local storage. An Ansible loop could be added to the data generation task to create a larger, unique dataset.

I use a Docker volume mounted at '/working' to store the data across stages.

Playbook Component 2: Writes

With the test data generated and stored on the temporary volume, the next task orchestrates concurrent writes of the data to S3.

The sequence of commands generates "object_count" copy operations and passes to "s5cmd run" via stdin. The same source file is uploaded to different destination keys; the source data fits into the filesystem cache and so prevents the local filesystem from becoming a bottleneck to s5cmd.

```
seq {{ object_count }}

 | xargs -I {} echo 'cp /working/somebytes s3://{{ s3bucket }}/{{ s3prefix }}/{{ inventory_hostname }}/{}'

 | s5cmd --endpoint-url https://{{ s3endpoint }} run"
```

To better understand how these piped commands work, it can be helpful to rerun

the command to show the operation list by removing the final s5cmd invocation.

The s5cmd invocations leverage the endpoint-url argument in order to target a non-AWS object store, like FlashBlade.

Playbook Component 3: Reads

To test the GET/s, the read task first enumerates all objects on the test bucket, shuffles their order, and then issues parallel reads of each object. The destination is /dev/null to avoid the local filesystem becoming a bottleneck. The grep and awk commands convert the output of “s5cmd ls” to form suitable as a “run” command list and the ‘shuf’ ensures objects are read back in random order.

```
s5cmd --endpoint-url https://{{ s3endpoint }} ls s3://{{ s3bucket }}/{{ s3prefix }}/*

 | grep -v DIR | awk ‘{print $NF}’ | shuf

 | xargs -I {} echo ‘cp s3://{{ s3bucket }}/{{ s3prefix }}/{} /dev/null’

 | s5cmd -numworkers 64 --endpoint-url https://{{ s3endpoint }} run”
```

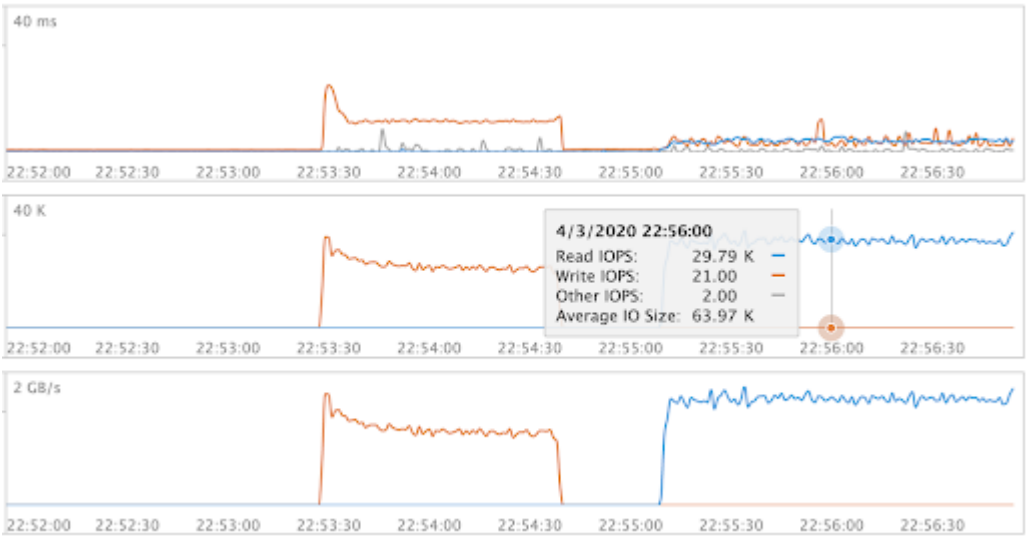
Playbook Component 4: Cleanup

The cleanup task simply removes all objects that were written by that host.

```
s5cmd --endpoint-url
href="https://joshua-robinson.medium.com/%7B%7B"
target="_blank" rel="noopener"> https://{{ s3endpoint }} rm
s3://{{ s3bucket }}/{{ s3prefix }}/{{ inventory_hostname }}/*
```

Running the Playbook on FlashBlade

Here is an example of running the Ansible playbook with FlashBlade object store that results in 30k GET/s at 64KB object size at consistently low latencies.



Summary

S5cmd is a high-performance tool for interacting with a fast object store, but it also has advanced features that make it flexible and powerful for common workflows and benchmarking.

Not all S3 tooling is performant, and this post explores deeper advanced s5cmd features: the “run” option for processing arbitrary lists of input commands and an ansible playbook for benchmarking.