SeaweedFS + GCS S3-Compatible Storage Setup Guide

```
Version: 1.0
Date: 2026-09-21
Author: Srikanth Vavilla
```

1. Overview

This guide sets up **SeaweedFS** as an S3-compatible storage gateway backed by **Google Cloud Storage (GCS)**, as a drop-in replacement for direct AWS S3 usage. Existing S3 SDK code, presigned URL generation, and CLI tooling continue to work unmodified — only the endpoint and credentials change.

```
Client / App (aws-sdk, boto3, presigned URLs)
        │  S3 API (SigV4)
        ▼
  SeaweedFS S3 Gateway (weed s3)
        │
        ▼
  SeaweedFS Filer (metadata store: etcd/Postgres/Redis)
        │  remote.mount
        ▼
  Google Cloud Storage bucket (native GCS API, via service account)
```

Why this shape: the Filer's local volume tier acts as a read/write cache in front of GCS, so hot objects (public assets) are served fast without hitting GCS on every request, while GCS remains the durable source of truth.

2. Prerequisites

Item	Notes
GCP project + billing enabled	Where the GCS bucket and service account live
`gcloud` CLI	Authenticated (`gcloud auth login`)
SeaweedFS binary	Latest release from GitHub, or Docker image `chrislusf/seaweedfs`
A host/VM/cluster for SeaweedFS	Master, Volume, Filer, S3 gateway processes
AWS CLI or boto3	For verifying the S3-compatible endpoint
DNS + TLS termination	e.g. nginx, ALB, or a CDN in front of the gateway
(Production) External metadata store	etcd, PostgreSQL, or Redis for the Filer

3. Step 1 — GCS bucket and service account

```
        # Create the bucket
        gcloud storage buckets create gs://my-assets-bucket --location=us-central1

        # Create a dedicated service account for SeaweedFS
        gcloud iam service-accounts create seaweedfs-gcs

        # Grant it object-level access to the bucket only (least privilege)
        gcloud storage buckets add-iam-policy-binding gs://my-assets-bucket \
          --member="serviceAccount:seaweedfs-gcs@PROJECT_ID.iam.gserviceaccount.com" \
          --role="roles/storage.objectAdmin"

        # Generate a key file SeaweedFS will use to authenticate to GCS
```

```
gcloud iam service-accounts keys create gcs-key.json \
  --iam-account=seaweedfs-gcs@PROJECT_ID.iam.gserviceaccount.com
```

Store `gcs-key.json` somewhere the SeaweedFS Filer process can read (e.g. a mounted secret, not committed to source control).

4. Step 2 — Deploy SeaweedFS core components

For a quick single-node start:

```
weed master -mdir=/data/master &
weed volume -dir=/data/vol -max=100 -mserver=localhost:9333 &
weed filer -master=localhost:9333 &
```

For anything beyond a proof of concept, point the Filer's metadata store at etcd, PostgreSQL, or Redis via `filer.toml` — the default embedded (leveldb) store does not support multiple Filer instances or reliable HA. Example `filer.toml` snippet for Postgres:

```
[postgres2]
enabled = true
hostname = "filer-db-host"
port = 5432
database = "seaweedfs"
username = "seaweedfs"
password = "..."
```

5. Step 3 — Mount GCS as the remote backend

Run `weed shell` and configure + mount the remote:

```
> remote.configure -name=gcs1 -type=google_cloud_storage \
    -google_cloud_storage.google_application_credentials=/path/gcs-key.json

> remote.mount -dir=/buckets/my-assets-bucket -remote=gcs1/my-assets-bucket
```

This exposes the GCS bucket as `/buckets/my-assets-bucket` in the Filer's namespace, using on-demand fetch/cache semantics by default.

To keep local writes synced to GCS continuously in the background (recommended for a bucket receiving live writes), run the sync daemon:

```
weed filer.remote.sync -dir=/buckets/my-assets-bucket
```

6. Step 4 — Start the S3-compatible API gateway

Define identities and credentials in `s3.json` — this replaces AWS IAM users with static access/secret key pairs mapped to permissions:

```
{
  "identities": [
    {
      "name": "app-user",
      "credentials": [
        {"accessKey": "AKIA...", "secretKey": "..."}
      ],
      "actions": ["Read", "Write", "List", "Tagging", "Admin"]
    }
  ]
}
```

Start the gateway:

```
weed s3 -port=8333 -config=/path/s3.json
```

The bucket mounted at `/buckets/my-assets-bucket` is now reachable as an S3 bucket named `my-assets-bucket` at `http://your-host:8333`.

7. Step 5 — Verify and generate presigned URLs

Sanity-check with the AWS CLI first:

```
aws --endpoint-url http://your-host:8333 s3 ls
aws --endpoint-url http://your-host:8333 s3 cp ./logo.png s3://my-assets-bucket/logo.png
```

SeaweedFS's S3 gateway implements SigV4, so existing presigned-URL code works unmodified — just repoint the endpoint and force path-style addressing:

```
import boto3

s3 = boto3.client(
    "s3",
    endpoint_url="https://s3.yourdomain.com",
    aws_access_key_id="AKIA...",
    aws_secret_access_key="...",
    region_name="us-east-1",
    config=boto3.session.Config(signature_version="s3v4", s3={"addressing_style": "path"}),
)

url = s3.generate_presigned_url(
    "get_object",
    Params={"Bucket": "my-assets-bucket", "Key": "reports/q3.pdf"},
    ExpiresIn=3600,
)
```

8. Step 6 — Serving public assets

The S3 gateway itself does not do TLS/ACM the way AWS does, so front it with one of:

- **CDN as origin** — Cloud CDN, Cloudflare, or CloudFront pointed at the gateway; matches the pattern typically already used with public S3 buckets.
- **Reverse proxy for TLS** — nginx or an ALB terminating TLS in front of `weed s3`.

Either way, DNS for `s3.yourdomain.com` (or your CDN's edge) should point at this front end, not directly at the gateway port.

9. Step 7 — Migrate existing data from AWS S3

Use `rclone` for a straightforward copy, either directly into GCS or through the new SeaweedFS S3 endpoint:

```
rclone copy s3remote:old-bucket gcsremote:my-assets-bucket --transfers=32 --checkers=16
```

Validate object counts and spot-check a sample of keys (including any with special characters) after the copy completes, before cutting over application traffic.

10. Production checklist

] Filer metadata store is etcd/PostgreSQL/Redis with replication — not the default embedded store

- [ ] Master and Volume servers run with redundancy (multiple replicas, not single-node)
- [ ] `gcs-key.json` stored as a secret (not in source control), with a rotation plan
- [ ] S3 identity access/secret keys rotated on a schedule — SeaweedFS has no AWS STS equivalent, so there are no temporary credentials out of the box
- [ ] TLS terminated in front of the gateway (nginx/ALB) or via a CDN
- [ ] Monitoring/alerting on Filer, Volume, and S3 gateway processes
- [ ] GCS egress cost modeled if clients will hit the gateway directly instead of through a CDN cache
- [ ] Exact CLI flags and remote-sync behavior re-verified against the SeaweedFS version actually deployed — this feature set has changed across releases