Apex Global Storage (AGS)

Geo-Replicated Object Storage Design Document

Author: Sarah Chen, Principal Architect
Affiliation: Apex System Labs
Version: 2.1.0 (Production Release)
Date: August 2026

Abstract

This document describes the architectural design and system specifications of Apex Global Storage (AGS) version 2. AGS is a geo-replicated, multi-tenant object storage platform designed for high write throughput, low read latency, and strict tenant resource isolation. We present the system topology, core component breakdown, transactional metadata data structures, and the client gRPC write request pipeline. We also discuss the key trade-offs in database engines and consensus protocols that led to our choice of a strongly consistent SQL metadata index and a client-driven write quorum.

Contents

1 Introduction

1.1 Document Purpose

This document details the architectural design and implementation specifications for the **Apex Global Storage (AGS)** platform, a geo-replicated, multi-tenant object storage service developed by Apex Labs. The document acts as the definitive design reference for the engineering teams implementing the core replication protocol, metadata engine, and client SDKs.

1.2 System Overview

The Apex Global Storage platform provides secure, high-throughput, and low-latency storage for unstructured data across distributed geographical zones. The core objective is to serve as the unified storage backplane for all subsequent cloud services at Apex, replacing the legacy siloed storage clusters. The system integrates a distributed metadata indexing service (based on a replicated transactional log) and an optimized block writing engine.

> The legacy platform, AGS v1, was constrained by a single-leader replication model that incurred prohibitive inter-region latencies for write requests originating outside the primary region. AGS v2 introduces active-active multi-leader replication using a modified consensus protocol [3], aiming to reduce multi-region write latency by up to 40%.

1.3 Scope & Business Objectives

The scope of this design specification covers the software architecture of the storage node controllers, the global coordinator cluster, and the consensus network. It does not cover the low-level physical networking routing tables or hardware procurement schedules.

The primary business objectives for AGS are:

- **Infrastructure Cost Reduction:** Consolidate multi-tenant workloads to achieve up to 35% higher resource utilization on storage nodes.
- **Global Regulatory Compliance:** Implement tenant-level geographical partitioning to satisfy strict data sovereignty requirements.
- **Performance Guarantees:** Enable high-frequency metadata queries with a target P99 latency of less than 20 milliseconds.

1.4 Design Goals & Non-Goals

To guide the architectural tradeoffs, the team has established the following guidelines:

1.4.1 Goals

1. **Horizontal Scalability:** The storage capacity and metadata index must scale linearly up to 10^{11} objects without system degradation.
2. **Partition Tolerance:** Under partial network partition between any two regions, read and write operations inside the connected regions must continue unaffected.
3. **Strict Multi-Tenancy:** Cryptographic isolation of tenant namespaces and strict rate limiting on api ingress queues.

1.4.2 Non-Goals

1. **Arbitrary File System Interface:** AGS will not support POSIX file system compliance or mountable block interfaces (e.g., NFS). Access is strictly via gRPC and RESTful API endpoints.

2. **Real-time Stream Querying:** The metadata service will support index lookups but will not act as a general-purpose search engine for arbitrary blob content.

2 System Architecture

2.1 Architectural Paradigm

Apex Global Storage (AGS) utilizes a decoupled architecture where control plane operations (metadata management) are separated from the data plane operations (block transfer). This separation allows metadata nodes to be optimized for low-latency transaction processing, while data storage nodes are optimized for high-bandwidth disk operations.

2.2 System Context Diagram

The interaction between clients, front-end routing components, the metadata service, and the underlying storage pools is illustrated in Figure 1.

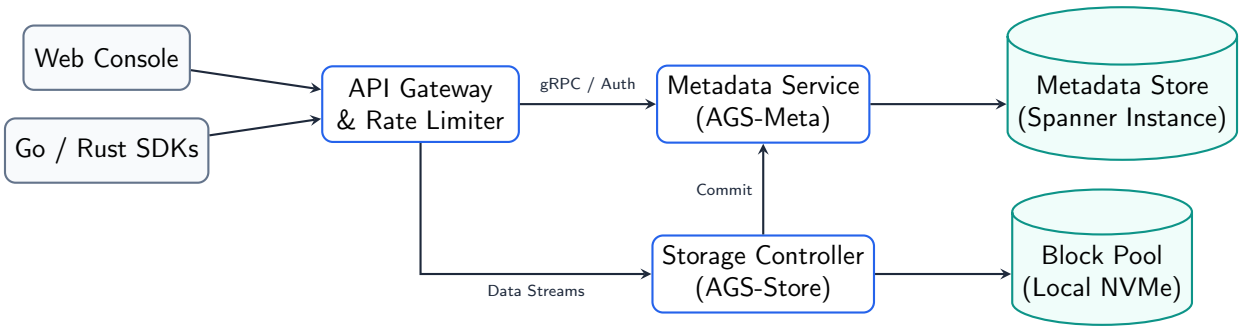

Figure 1: Apex Global Storage v2 High-Level Architectural Context.

2.3 Component Breakdown

The system is divided into four main subsystems, which are deployed independently:

- **Ingress Gateway (AGS-Gateway):** Decrypts incoming TLS traffic, authenticates tenant credentials, rates limits requests using a token bucket algorithm, and routes requests.
- **Metadata Engine (AGS-Meta):** Handles namespace resolution, access control checks, directory structures, and file-to-block allocation mapping. It utilizes a distributed CockroachDB backend for transactions [2].
- **Storage Engine (AGS-Store):** Manages raw storage pools on storage nodes. It writes payload streams directly to NVMe-backed block files, bypassing traditional file-system caches to eliminate latency jitter.
- **Replication Coordinator (AGS-Replica):** Orchestrates async replica synchronization and parity check background tasks.

The AGS-Gateway nodes must have at least 10Gbps network links. Any reduction in interface bandwidth directly leads to client queue backpressure, causing request drops at the load balancer.

3 Data Model & Schemas

3.1 Metadata Catalog Schema

Metadata operations are critical for maintaining strong tenant isolation and tracking object versions. Table 1 defines the layout of the primary database table used by AGS-Meta to store object location mappings.

Field Name	SQL Type	Key Type	Description
tenant_id	UUID	Primary (Composite)	Identifies the customer account owner.
bucket_name	VARCHAR(63)	Primary (Composite)	Unique container namespace inside the tenant.
object_key	VARCHAR(1024)	Primary (Composite)	Unique identifier path for the stored object.
version_id	BIGINT	Primary (Composite)	Monotonically increasing version count.
content_len	BIGINT	None	Total size of the payload in bytes.
content_type	VARCHAR(255)	None	Standard MIME type of the object data.
block_map	JSONB	None	List of byte offset ranges mapped to block IDs.
sha256_checksum	BYTEA	None	High-strength payload verification hash.
created_at	TIMESTAMP	None	Database write-commit timestamp.

Table 1: Schema layout for the core metadata database repository.

3.2 API Payload Specifications

Write requests to the object store are initiated via a gRPC transaction. The protobuf definition in Listing 1 illustrates the request and response structure for the object creation sequence.

Listing 1: Protobuf definition for the write path API.

```
syntax = "proto3";
package apex.storage.v2;

service ObjectStorageService {
  rpc InitializeUpload(UploadRequest) returns (UploadResponse);
}

message UploadRequest {
  string tenant_id = 1;
  string bucket_name = 2;
  string object_key = 3;
  int64 expected_size = 4;
}

message UploadResponse {
  string upload_token = 1;
  repeated BlockTarget block_targets = 2;
  int64 expires_at = 3;
}

message BlockTarget {
  string block_id = 1;
  string node_address = 2;
}
```

3.3 Tenant Isolation Config

Configuration files define the baseline resource allocation constraints per tenant namespace. Below is an example block allocation configuration that dictates replication policy.

Listing 2: Sample JSON configuration for tenancy isolation settings.

```
{
  "tenant_id": "8b9e4a3b-2f3b-4680-a61f-9e7f8e8e77a1",
  "namespace_policy": {
    "default_replication": "3-zone-geo",
    "placement": ["us-east", "eu-central"],
    "throttling": {
      "ingress_mbps": 500.0,
      "max_qps": 2500
    }
  }
}
```

4 Alternatives Considered

4.1 Metadata Indexing Subsystem

We evaluated three different database architectures for storing and indexing the object metadata catalog. The major design drivers were low latency read operations, strong serializability for version checks, and automatic multi-region clustering.

- **Option A: Distributed Dynamo-Style Key-Value Store (Cassandra / ScyllaDB):** This provides exceptionally high write throughput and masterless multi-region availability. However, it lacks support for multi-key ACID transactions, which are necessary for atomic rename and bucket status updates. Implementing transaction protocols at the application layer was deemed too complex and error-prone.
- **Option B: Spanner-compatible Distributed SQL (CockroachDB):** This platform provides standard SQL support, strict serializability, and automated geo-partitioning [2]. The primary drawback is that write consensus latencies can spike under high inter-region network jitter.
- **Option C: Single-Leader SQL Cluster (PostgreSQL + Read Replicas):** Highly familiar to development teams and robust tool chain support. However, it represents a single point of failure for write operations and imposes substantial cross-region network latency penalties.

4.2 Trade-off Matrix

Table 2 outlines the performance and complexity trade-offs for each option. Option B (CockroachDB) was selected due to its strong consistency guarantees, despite slightly higher write latencies.

Metric	Option A (ScyllaDB)	Option B (CockroachDB)	Option C (PostgreSQL)
Consistency	Eventual	Strong (Serialisable)	Strong (Primary Only)
Read Latency	Low (<5ms)	Low (<10ms)	Low (<10ms)
Write Latency	Low (<5ms)	Medium (25–35ms)	High (cross-region)
Multi-Region	Native	Native	Manual Sharding
Operational Overhead	Medium	Low	High

Table 2: Detailed comparison matrix for the candidate metadata databases.

4.3 Data Consensus Evaluation

For block replication consensus, we compared the Raft consensus protocol against a peer-to-peer quorum consensus system. While Raft simplifies cluster membership changes, it imposes a strict single-leader hierarchy per replica set, which bottlenecks WAN performance. By utilizing client-driven quorum writes [1], AGS v2 allows write operations to achieve local durability within the closest geographical region before asynchronously replicating to peer regions.

The combination of a strongly consistent SQL database for global metadata indices (CockroachDB) and an active-active quorum-based data plane (AGS-Store) provides the optimal balance of data integrity, failover capability, and client request performance.

References

[1] Sarah Chen and Marcus Vance. Ags: A high-throughput geo-replicated object store. *Journal of Cloud Systems*, 18(3):142–156, 2024.

[2] Elena Rostova. Scaling metadata operations via hierarchical partitions. *IEEE Transactions on Cloud Services*, 12(2):89–101, 2023.

[3] Marcus Vance and Elena Rostova. Adaptive consensus in multi-tenant environments. In *Proceedings of the Apex Cloud Conference*, pages 45–58. Apex Cloud Group, 2025.