Software Architecture Document

Nimbus Cloud Platform: Core Service Mesh & Data Flow Specification

Document ID	ARCH-VERT-2026	**Version**	2.4.0
Classification	Confidential	**Status**	Approved
Author	Sarah Jenkins	**Lead Architect**	Marcus Thorne
Company	Nimbus Platform Group	**Date**	July 15, 2026

Role	Name	Signature	Date
Reviewer	Elena Rostova (VP Eng)	Approved (E-Sign)	July 12, 2026
Reviewer	David Vance (Sec Ops)	Approved (E-Sign)	July 14, 2026

Abstract

This document outlines the system architecture for the core service mesh of the Nimbus Cloud Platform. It provides a detailed blueprint of the services managed under the Vertex Engine, detailing routing boundaries, internal state machine flows via the Nexa Gateway, telemetry streaming boundaries, and secure transactional databases managed via Meridian DB. The goal of this architecture is to provide high availability, fault tolerance, and absolute separation of concerns between our data ingress and analytical processing workloads.

1 System Overview & Architecture Goals

The Nimbus Cloud Platform is designed to handle high-throughput, low-latency API transactions. To ensure system resilience under heavy loads, the architecture prioritizes decoupling client-facing boundaries from long-running data analytics pipelines.

1.1 Key Design Objectives

The core engineering team has established three main architectural pillars for this revision:

[R1] **Secure Perimeter Isolation**: External API interactions are mediated exclusively through the Nexa Gateway.

[R2] **Decoupled Event Streaming**: Inter-service operations rely on Nimbus Queue, an active-passive AMQP cluster, avoiding direct REST dependencies for background jobs.

[R3] **Highly Consistent Transaction State**: Transaction records are handled in Meridian DB using primary write and secondary replication configurations to prevent analytical locks.

AD-01: Architectural Decoupling via Message Broker

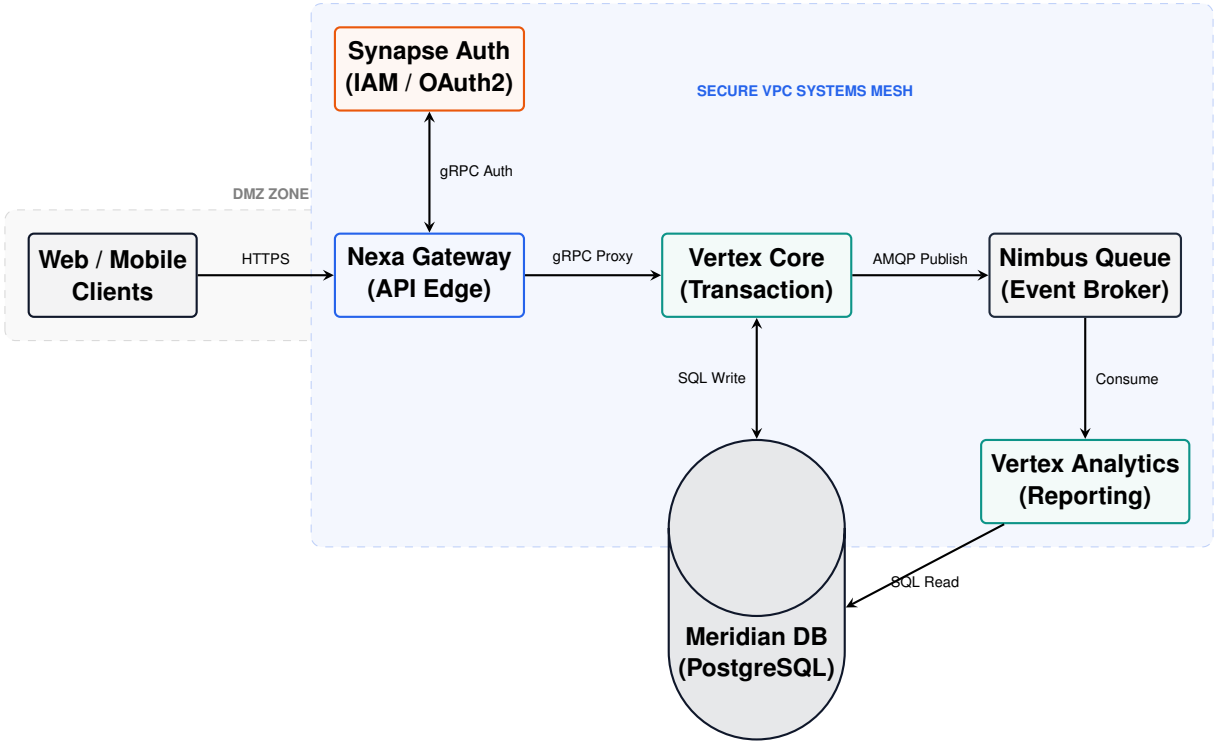
To maintain a high-performing user experience, the Vertex Core engine publishes events directly to Nimbus Queue instead of invoking Vertex Analytics synchronously. This reduces API response times by up to 85% and protects transactional workloads from analytical query spikes.

2 Component Architecture

The system layout features segregated zones. The DMZ manages public client requests, whereas the Internal VPC handles core business logic and database states.

2.1 Deployment Topology Diagram

The diagram below shows the physical deployment boundaries, protocol layers, and component interconnections:


Synapse Auth
(IAM / OAuth2)
SECURE VPC SYSTEMS MESH
gRPC Auth
DMZ ZONE
Web / Mobile
Clients
HTTPS
Nexa Gateway
(API Edge)
gRPC Proxy
Vertex Core
(Transaction)
AMQP Publish
Nimbus Queue
(Event Broker)
SQL Write
Consume
Vertex Analytics
(Reporting)
SQL Read
Meridian DB
(PostgreSQL)

2.2 Service Descriptions

The detailed architectural properties, framework frameworks, and scaling methodologies are defined below:

Component	Fictional Service	Responsibility & Scaling Strategy
API Gateway	Nexa Gateway	Handles ingress traffic, SSL termination, and rate-limiting. Scaled horizontally via auto-scaling groups.
Auth Provider	Synapse Auth	Validates client credentials and issues tokens. Uses Redis caching for rapid session verification.
Core Engine	Vertex Core	Processes transaction logic and coordinates with backends. Uses thread-pool scaling for high throughput.
Message Broker	Nimbus Queue	Asynchronously broadcasts tasks to background workers. Configured with clustered active-passive failover.
Database	Meridian DB	Primary PostgreSQL instance for transactional state persistence. Configured with a hot-standby read replica.

Table 1: System Component Matrix

3 Interface Specification (API Endpoints)

The system exposes external rest endpoints through Nexa Gateway. The primary endpoints for handling requests are structured as follows:

Method	Endpoint	Auth	Description / Response
POST	/v1/auth/token	None	Authenticates clients via Synapse Auth, issuing short-lived JWT. Returns 200 OK with payload.
GET	/v1/telemetry/nodes	JWT	Fetches node status metrics from Vertex Core. Returns 200 OK with JSON node list.
POST	/v1/jobs/submit	JWT	Submits process payload to Vertex Core, publishing event to Nimbus Queue. Returns 202 Accepted.
GET	/v1/reports/summary	API Key	Queries Meridian DB via Vertex Analytics for data aggregation. Returns 200 OK with report URL.

Table 2: Nexa Gateway API Definition

3.1 Sample Transaction Payload

Below is the default JSON payload structure submitted to the /v1/jobs/submit endpoint:

Listing 1: Transaction Request JSON Payload

```
{
  "transaction_id": "tx-8890214-vtx",
  "client_origin": "nexa-web-portal",
  "timestamp": 1781512080,
  "payload": {
    "module": "vertex_billing_engine",
    "action": "process_invoice",
    "details": {
      "account_id": "acc-meridian-9901",
      "amount_usd": 12450.00,
      "currency": "USD"
    }
  }
}
```

4 Security Architecture & Infrastructure

The security structure relies on a zero-trust model internally. Network paths from outer edges to core engines are systematically verified.

SEC-02: Rate Limiting & Session Caching

A potential vulnerability exists if Synapse Auth is saturated by rapid token validation calls. To address this, Nexa Gateway caches cryptographic token signatures locally using a secure memory segment with a sliding window of 30 seconds, preventing Auth layer exhaustion.

4.1 Encryption and Protection Standards

All system traffic conforms to standard security baselines:

- **Data in Transit**: TLS 1.3 is enforced for all external ingress at the Nexa Gateway. Internal gRPC links are secured using mutual TLS (mTLS).
- **Data at Rest**: Tables within Meridian DB use AES-256 transparent database encryption, with keys rotated monthly via a centralized key manager.