Mailflex Global Platform - Software Architecture Document

Version: 1.0

Date: October 13, 2025

Status: Architecture Design

Table of Contents

---

1. Introduction

1.1 Purpose

This document describes the software architecture for Mailflex, a global email and collaboration platform. It provides technical guidance for development, deployment, and operations teams.

1.2 Scope

This architecture covers:

- System-level architecture and design patterns
- Component and microservices architecture
- Data storage and management
- Network topology and routing
- Security and compliance architecture
- Deployment and operational architecture

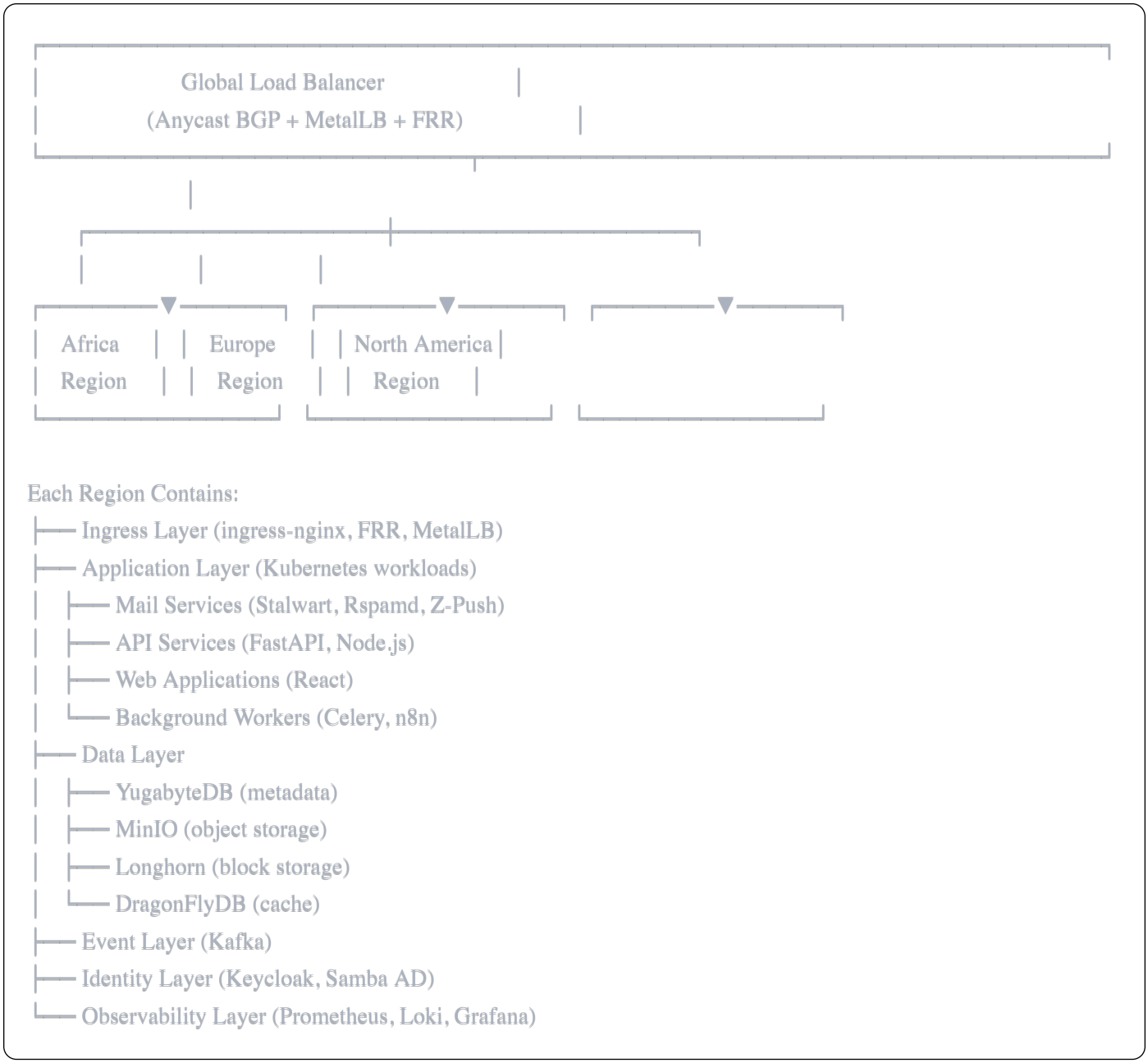
1.3 Audience

- Software Architects
- DevOps Engineers
- Security Engineers
- Development Teams
- Operations Teams

---

2. System Overview

2.1 High-Level Architecture



2.2 Technology Stack Summary

Layer	Technologies
Frontend	React, TypeScript, MUI, Tailwind CSS
Backend	Python (FastAPI), Node.js (Express), Go (Gin)
Mail Services	Stalwart, Rspamd, Z-Push, Postfix
Databases	YugabyteDB, ClickHouse, DragonFlyDB
Storage	MinIO, Longhorn, NFS
Messaging	Apache Kafka, NATS
Identity	Keycloak, Samba AD
Orchestration	Kubernetes, Rancher
Infrastructure	OpenStack, Terraform
CI/CD	Argo CD, Argo Rollouts, GitLab CI
Observability	Prometheus, Loki, Grafana, Jaeger
AI/ML	Langflow, Qdrant, LlamaIndex

---

3. Architectural Principles

3.1 Design Principles

1. Cloud-Native First

- Containerized microservices
- Kubernetes-native design
- 12-factor app methodology
- Stateless application tier

2. API-First Design

- OpenAPI/Swagger specifications
- RESTful and gRPC APIs
- Versioned APIs with backward compatibility
- Developer-friendly SDKs

3. Event-Driven Architecture

- Kafka as event backbone
- Asynchronous communication
- Event sourcing for audit trails
- CQRS pattern for read/write separation

4. Zero-Trust Security

- Mutual TLS between services
- Service-to-service authentication
- Network segmentation
- Least privilege access

5. GitOps Operations

- Infrastructure as Code (IaC)
- Declarative configuration
- Git as single source of truth
- Automated reconciliation

6. Observability by Design

- Structured logging
- Distributed tracing
- Comprehensive metrics
- Service-level objectives (SLOs)

3.2 Architectural Patterns

Microservices Pattern

- Domain-driven design
- Bounded contexts
- Independent deployment
- Technology diversity

Strangler Fig Pattern

- Gradual migration from monolith
- Proxy layer for routing
- Feature-by-feature migration

Circuit Breaker Pattern

- Fault isolation
- Graceful degradation
- Automatic recovery
- Health checking

Saga Pattern

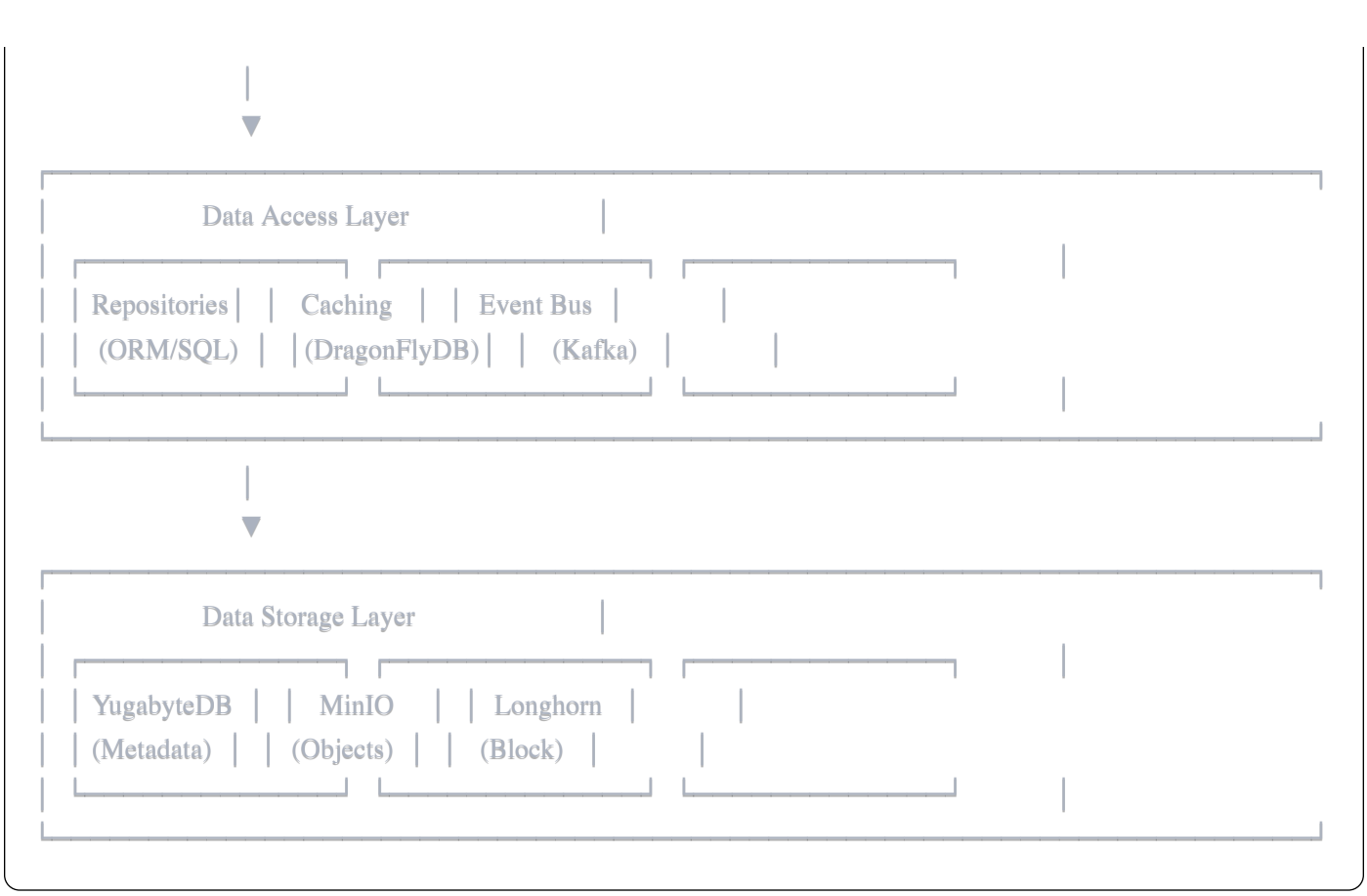
- Distributed transactions
- Compensating transactions
- Event choreography

---

4. System Architecture

4.1 Logical Architecture


Presentation Layer
Web App (React)
Mobile App (iOS/And)
Desktop (Electron)
Admin Console
API Gateway Layer
ingress-nginx + OAuth2 Proxy + Rate Limiting
Application Services Layer
Mail Services
- Stalwart
- Rspamd
- Z-Push
API Services
- User API
- Tenant API
- Mail API
Auth Services
- Keycloak
- Samba AD
- OAuth2
Collab Svcs
- Baïkal
- Owncloud
- Jitsi
UC Services
- OpenSIPS
- FreeSWITCH
- Jasmin
Marketing Services
- Mautic
- Listmonk
Business Logic Layer
Domain Services, Workflows, Event Handlers



4.2 Component Interaction Diagram

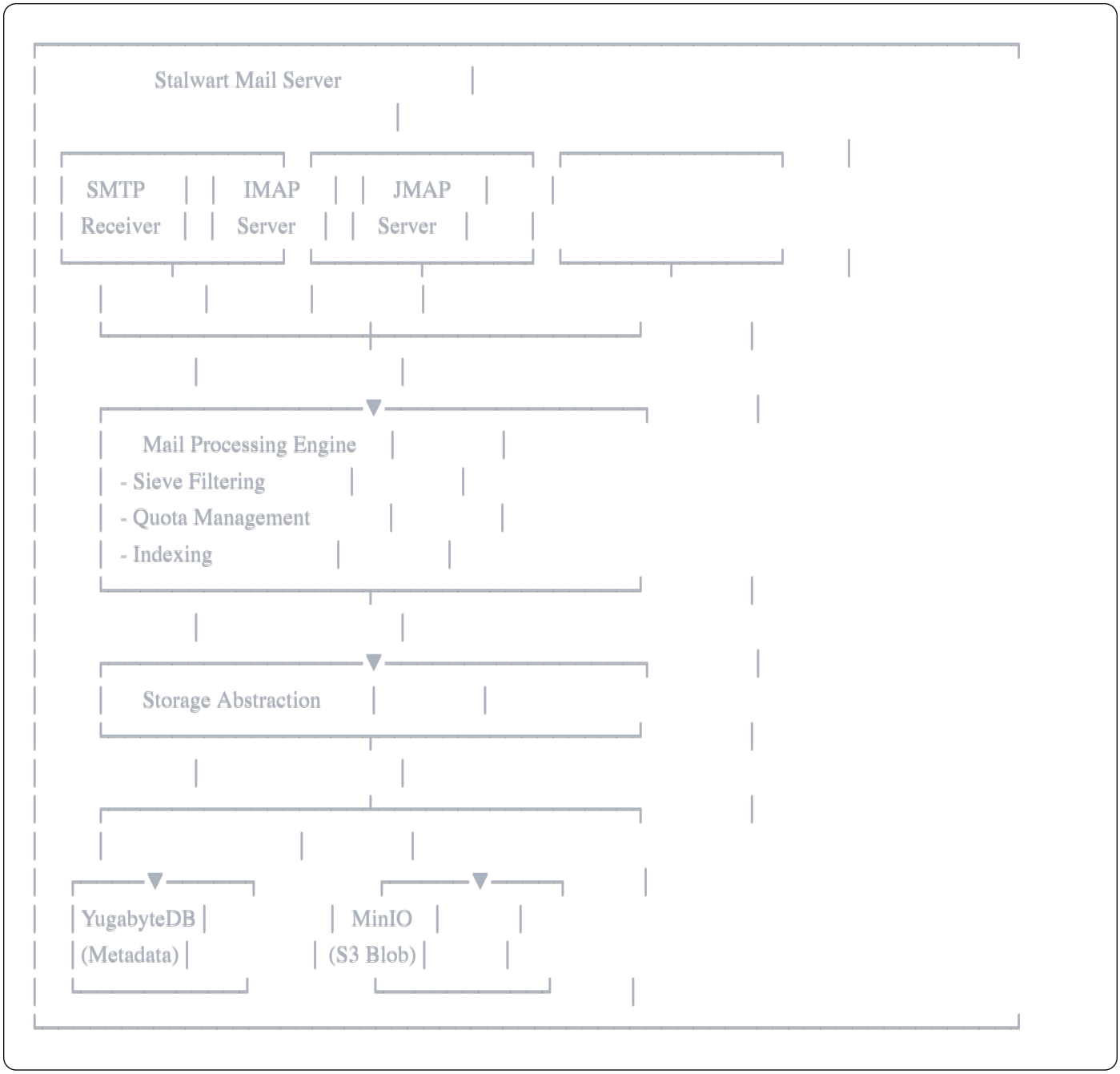
```mermaid
sequenceDiagram
    participant Client
    participant Ingress
    participant API
    participant AuthN
    participant Mail
    participant DB
    participant S3
    participant Kafka

    Client->>Ingress: HTTPS Request
    Ingress->>API: Forward Request
    API->>AuthN: Validate Token
    AuthN-->>API: Token Valid
    API->>Mail: Process Request
    Mail->>DB: Query Metadata
    DB-->>Mail: Return Data
    Mail->>S3: Store/Retrieve Email
    S3-->>Mail: Email Data
    Mail->>Kafka: Publish Event
    Mail-->>API: Response
    API-->>Ingress: Response
    Ingress-->>Client: HTTPS Response
```

---

5. Component Architecture

5.1 Mail Services Architecture

5.1.1 Stalwart Mail Server



Key Features:

- Multi-protocol support (SMTP, IMAP, JMAP, Sieve)
- Per-tenant S3 storage integration
- OAuth2 authentication support
- Server-side filtering with Sieve
- IDLE/push support for real-time sync

Configuration:

yaml

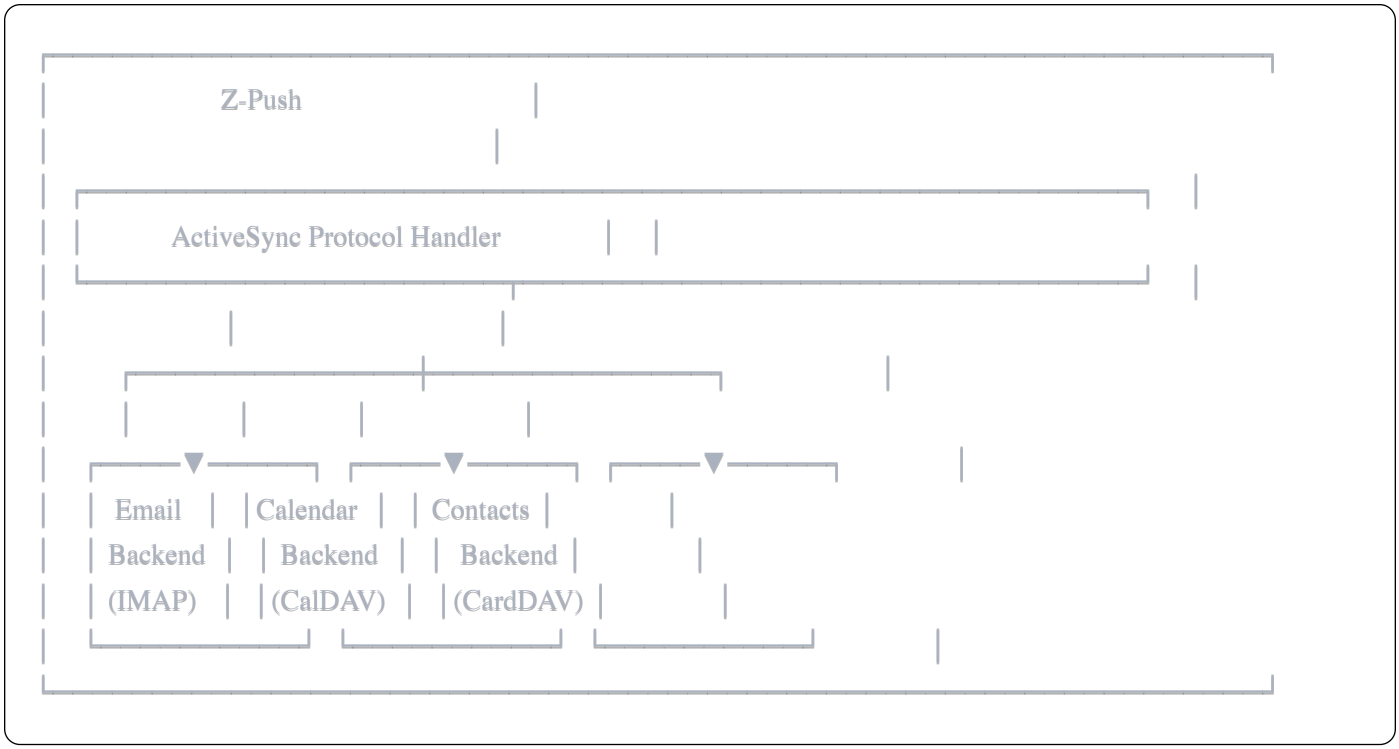
```yaml
# stalwart.yaml
server:
  hostname: mail.mailflex.io
  protocols:
    smtp:
      port: 25
      tls: required
    imap:
      port: 993
      tls: required
    jmap:
      port: 443
      base-url: https://jmap.mailflex.io

storage:
  backend: s3
  s3:
    endpoint: minio.mailflex.io:9000
    access-key: ${MINIO_ACCESS_KEY}
    secret-key: ${MINIO_SECRET_KEY}
    bucket-prefix: tenant-

authentication:
  oauth2:
    issuer: https://auth.mailflex.io/realms/mailflex
    client-id: stalwart-mail

metadata:
  backend: postgresql
  connection: postgresql://yugabyte:5433/mailflex
```

5.1.2 Rspamd Filtering

Features:

- Real-time spam detection
- Virus scanning integration
- DKIM signing/verification
- SPF/DMARC validation
- Rate limiting
- Greylisting
- Statistical learning

5.1.3 Z-Push ActiveSync Bridge



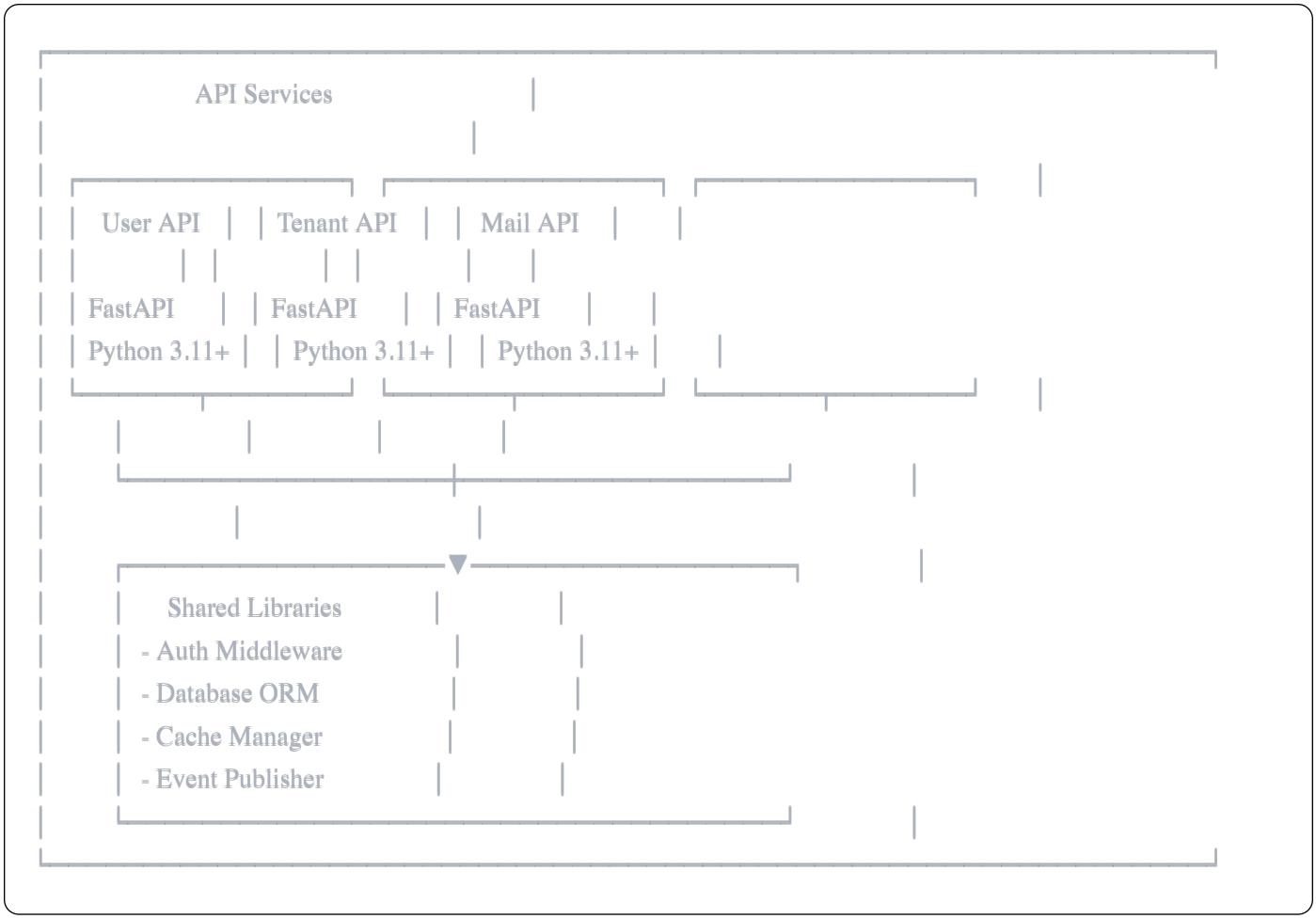
5.2 API Services Architecture

5.2.1 API Gateway

python

```python
# FastAPI-based API Gateway
from fastapi import FastAPI, Depends, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from fastapi.security import OAuth2PasswordBearer

app = FastAPI(
    title="Mailflex API Gateway",
    version="1.0.0",
    docs_url="/api/docs"
)

# CORS Configuration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Authentication
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

# Rate Limiting
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter

# Routes
from routers import users, tenants, mail, admin

app.include_router(users.router, prefix="/api/v1/users")
app.include_router(tenants.router, prefix="/api/v1/tenants")
app.include_router(mail.router, prefix="/api/v1/mail")
app.include_router(admin.router, prefix="/api/v1/admin")
```

5.2.2 Service Architecture



5.3 Data Services Architecture

5.3.1 YugabyteDB Cluster

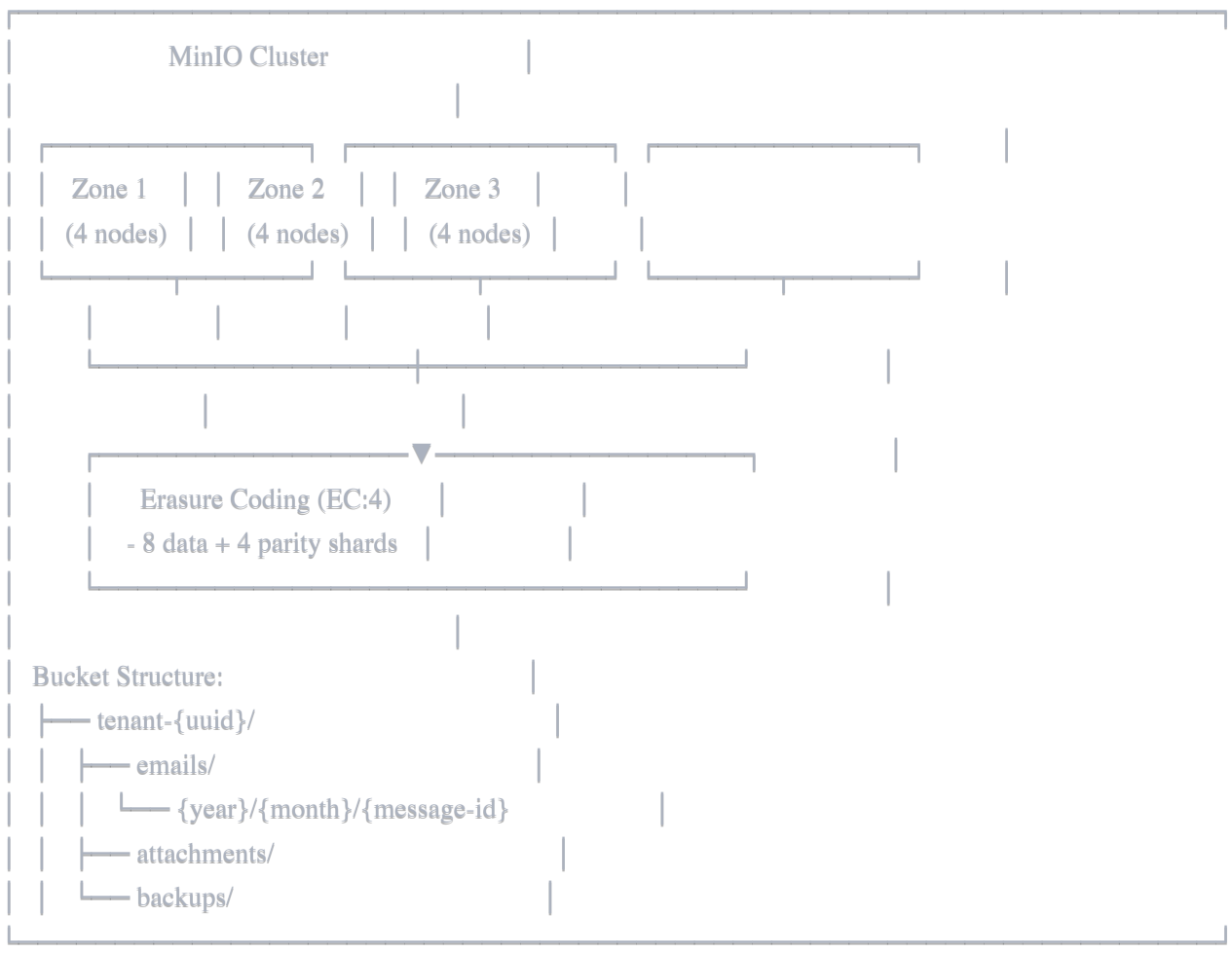
Schema Design:

sql

```sql
-- Core Tables
CREATE TABLE tenants (
    tenant_id UUID PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    domain VARCHAR(255) UNIQUE NOT NULL,
    tier VARCHAR(50) NOT NULL,
    storage_quota BIGINT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE users (
    user_id UUID PRIMARY KEY,
    tenant_id UUID REFERENCES tenants(tenant_id),
    email VARCHAR(255) UNIQUE NOT NULL,
    display_name VARCHAR(255),
    quota_used BIGINT DEFAULT 0,
    quota_limit BIGINT,
    created_at TIMESTAMP DEFAULT NOW(),
    last_login TIMESTAMP
);

CREATE INDEX idx_users_tenant ON users(tenant_id);
CREATE INDEX idx_users_email ON users(email);

CREATE TABLE mailboxes (
    mailbox_id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(user_id),
    name VARCHAR(255) NOT NULL,
    path VARCHAR(500) NOT NULL,
    uidvalidity INTEGER,
    uidnext INTEGER,
    message_count INTEGER DEFAULT 0,
    unseen_count INTEGER DEFAULT 0
);

CREATE TABLE messages (
    message_id UUID PRIMARY KEY,
    mailbox_id UUID REFERENCES mailboxes(mailbox_id),
    uid INTEGER NOT NULL,
    size BIGINT NOT NULL,
    s3_key VARCHAR(500) NOT NULL,
    flags JSONB
```

```
    flags JSONB,
    internal_date TIMESTAMP,
    received_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_messages_mailbox ON messages(mailbox_id, uid);
```

5.3.2 MinIO S3 Storage



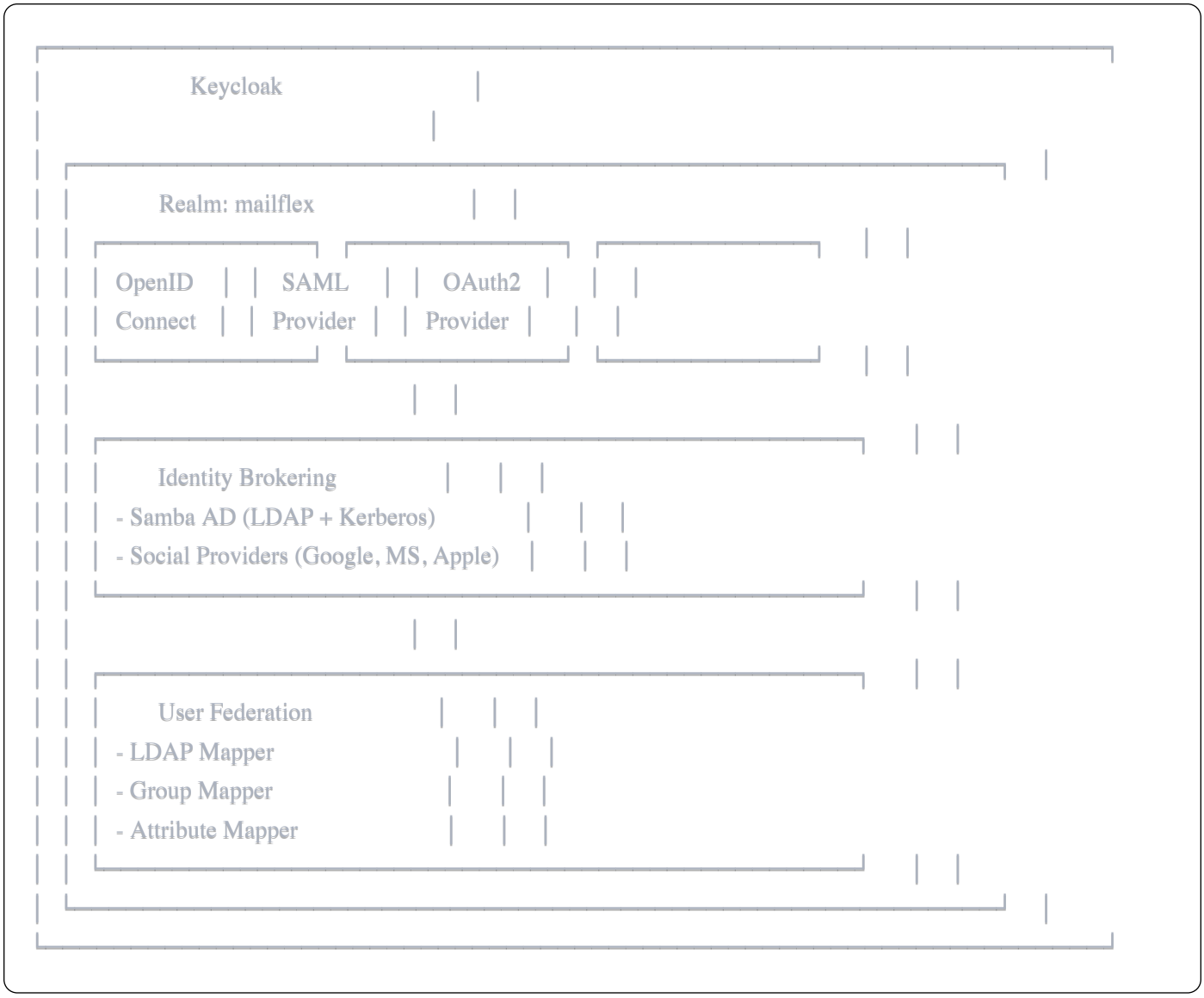
Configuration:

yaml

```yaml
# MinIO Configuration
server:
  mode: distributed
  zones:
    - http://minio-{1...4}.zone1.mailflex.io/data
    - http://minio-{1...4}.zone2.mailflex.io/data
    - http://minio-{1...4}.zone3.mailflex.io/data

storage:
  erasure-coding: EC:4
  encryption:
    server-side: AES256
    kms: vault.mailflex.io

lifecycle:
  - rule: archive-old-emails
    prefix: tenant-*/emails/
    days: 90
    transition: GLACIER

security:
  tls:
    enabled: true
    cert: /certs/server.crt
    key: /certs/server.key
```

5.4 Identity Services Architecture

5.4.1 Keycloak SSO



Realm Configuration:

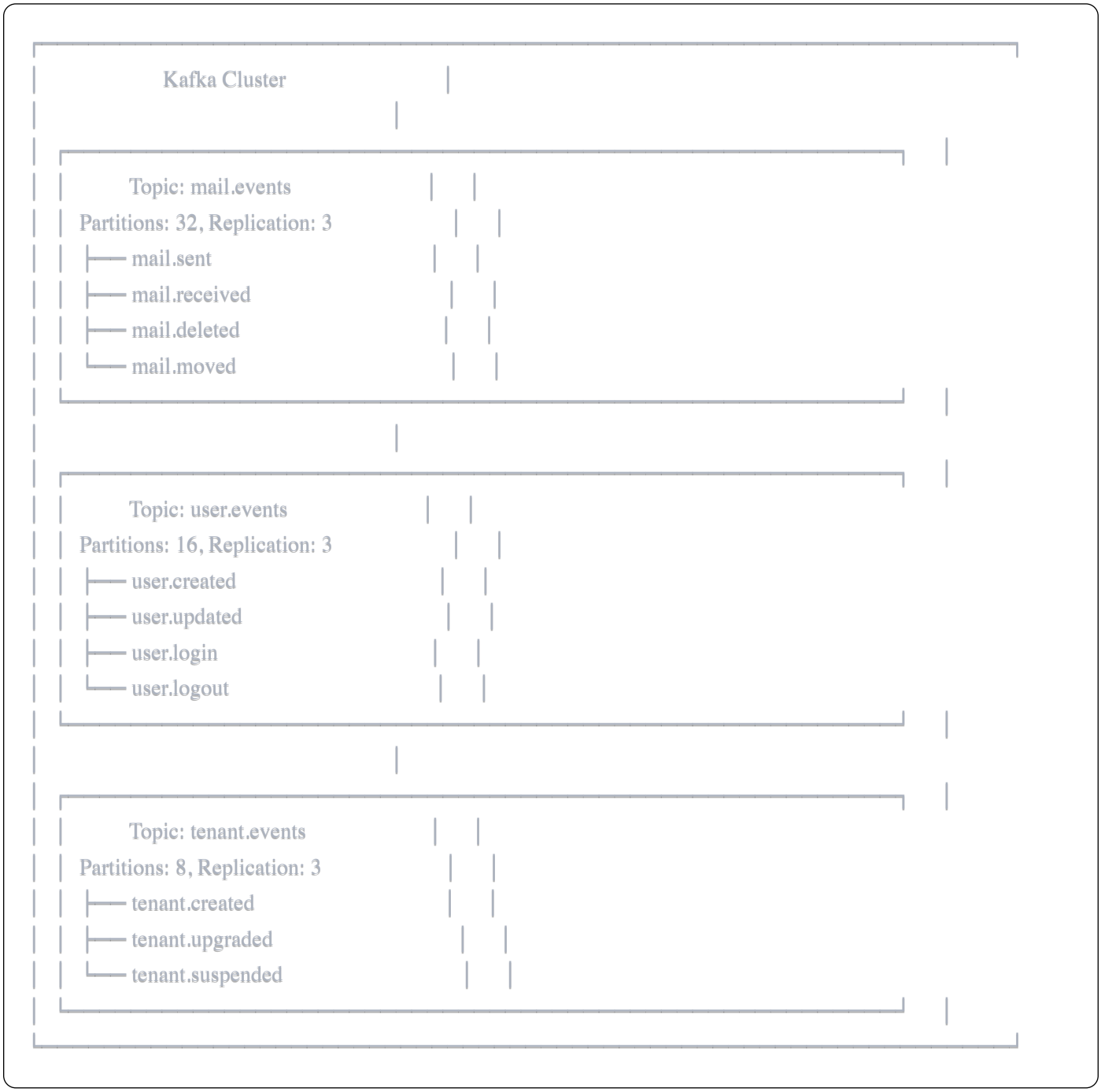
json


```
{
 "realm": "mailflex",
 "enabled": true,
 "sslRequired": "external",
 "registrationAllowed": true,
 "loginWithEmailAllowed": true,
 "duplicateEmailsAllowed": false,
 "resetPasswordAllowed": true,
 "editUsernameAllowed": false,
 "bruteForceProtected": true,
 "permanentLockout": false,
 "maxFailureWaitSeconds": 900,
 "minimumQuickLoginWaitSeconds": 60,
 "waitIncrementSeconds": 60,
 "quickLoginCheckMilliSeconds": 1000,
 "maxDeltaTimeSeconds": 43200,
 "failureFactor": 5,
 "defaultSignatureAlgorithm": "RS256",
 "offlineSessionMaxLifespan": 5184000,
 "accessTokenLifespan": 300,
 "accessTokenLifespanForImplicitFlow": 900,
 "ssoSessionIdleTimeout": 1800,
 "ssoSessionMaxLifespan": 36000,
 "clients": [
  {
   "clientId": "mailflex-web",
   "enabled": true,
   "protocol": "openid-connect",
   "publicClient": true,
   "redirectUris": [
    "https://mail.mailflex.io/*"
   ],
   "webOrigins": [
    "https://mail.mailflex.io"
   ]
  },
  {
   "clientId": "stalwart-mail",
   "enabled": true,
   "protocol": "openid-connect",
   "publicClient": false,
   "serviceAccountsEnabled": true,
```

```
    "authorizationServicesEnabled": false
  }
 ]
}
```

5.5 Event-Driven Architecture

5.5.1 Kafka Cluster



Event Schema (Avro):

json

```
{
  "namespace": "io.mailflex.events",
  "type": "record",
  "name": "MailEvent",
  "fields": [
    {"name": "event_id", "type": "string"},
    {"name": "event_type", "type": "string"},
    {"name": "timestamp", "type": "long"},
    {"name": "tenant_id", "type": "string"},
    {"name": "user_id", "type": "string"},
    {"name": "mailbox_id", "type": "string"},
    {"name": "message_id", "type": ["null", "string"]},
    {"name": "metadata", "type": {"type": "map", "values": "string"}}
  ]
}
```

---

6. Data Architecture

6.1 Data Flow Diagram

6.2 Database Schema

sql

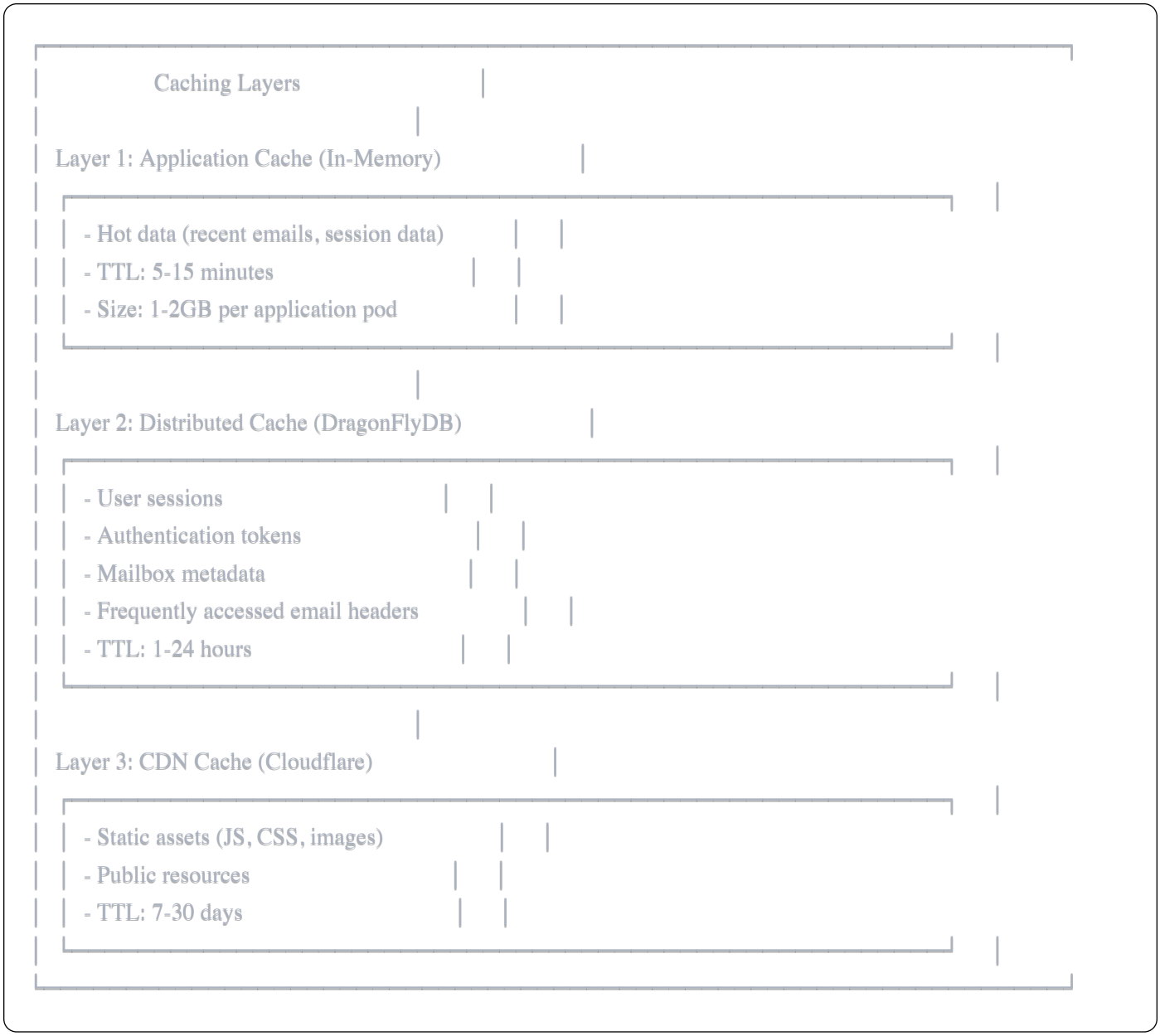
```sql
-- Tenant Management
CREATE SCHEMA tenant;

CREATE TABLE tenant.tenants (
    tenant_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    domain VARCHAR(255) UNIQUE NOT NULL,
    tier VARCHAR(50) NOT NULL CHECK (tier IN ('free', 'professional', 'business', 'enterprise')),
    status VARCHAR(50) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'suspended', 'deleted')),
    storage_quota BIGINT NOT NULL,
    storage_used BIGINT DEFAULT 0,
    user_quota INTEGER NOT NULL,
    user_count INTEGER DEFAULT 0,
    s3_bucket VARCHAR(255),
    settings JSONB,
    billing_email VARCHAR(255),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    deleted_at TIMESTAMP WITH TIME ZONE
);

CREATE INDEX idx_tenants_domain ON tenant.tenants(domain);
CREATE INDEX idx_tenants_status ON tenant.tenants(status);

-- User Management
CREATE SCHEMA users;

CREATE TABLE users.users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenant.tenants(tenant_id) ON DELETE CASCADE,
    email VARCHAR(255) NOT NULL,
    display_name VARCHAR(255),
    given_name VARCHAR(100),
    family_name VARCHAR(100),
    password_hash VARCHAR(255),
    status VARCHAR(50) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'suspended', 'deleted')),
    quota_used BIGINT DEFAULT 0,
    quota_limit BIGINT,
    preferences JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    last_login TIMESTAMP WITH TIME ZONE,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
```

```
    deleted_at TIMESTAMP WITH TIME ZONE,
    UNIQUE(tenant_id, email)
);

CREATE INDEX idx_users_tenant ON users.users(tenant_id);
CREATE INDEX idx_users_email ON users.users(email);
CREATE INDEX idx_users_status ON users.users(status);

-- Mail Schema
CREATE SCHEMA mail;

CREATE TABLE mail.mailboxes (
    mailbox_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users.users(user_id) ON DELETE CASCADE,
    parent_id UUID REFERENCES mail.mailboxes(mailbox_id),
    name VARCHAR(255) NOT NULL,
    path VARCHAR(500) NOT NULL,
    special_use VARCHAR(50) CHECK (special_use IN ('inbox', 'sent', 'drafts', 'trash', 'spam', 'archive')),
    uidvalidity INTEGER NOT NULL,
    uidnext INTEGER NOT NULL DEFAULT 1,
    message_count INTEGER DEFAULT 0,
    unseen_count INTEGER DEFAULT 0,
    size_bytes BIGINT DEFAULT 0,
    subscribed BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    UNIQUE(user_id, path)
);

CREATE INDEX idx_mailboxes_user ON mail.mailboxes(user_id);
CREATE INDEX idx_mailboxes_parent ON mail.mailboxes(parent_id);

CREATE TABLE mail.messages (
    message_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    mailbox_id UUID NOT NULL REFERENCES mail.mailboxes(mailbox_id) ON DELETE CASCADE,
    uid INTEGER NOT NULL,
    size_bytes BIGINT NOT NULL,
    s3_key VARCHAR(500) NOT NULL,
    flags JSONB DEFAULT '[]',
    labels JSONB DEFAULT '[]',
    internal_date TIMESTAMP WITH TIME ZONE NOT NULL,
    received_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    -- Email headers for searching
    from_addr VARCHAR(255),
    to_addrs TEXT[],
```

```
    cc_addrs TEXT[],
    subject TEXT,
    message_id_header VARCHAR(255),
    in_reply_to VARCHAR(255),
    references TEXT[],
    -- Full-text search
    body_text TSVECTOR,
    UNIQUE(mailbox_id, uid)
);

CREATE INDEX idx_messages_mailbox ON mail.messages(mailbox_id);
CREATE INDEX idx_messages_uid ON mail.messages(mailbox_id, uid);
CREATE INDEX idx_messages_received ON mail.messages(received_at DESC);
CREATE INDEX idx_messages_from ON mail.messages USING gin(from_addr gin_trgm_ops);
CREATE INDEX idx_messages_subject ON mail.messages USING gin(subject gin_trgm_ops);
CREATE INDEX idx_messages_body ON mail.messages USING gin(body_text);

-- Collaboration Schema
CREATE SCHEMA collab;

CREATE TABLE collab.calendars (
    calendar_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users.users(user_id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    color VARCHAR(7),
    timezone VARCHAR(100) DEFAULT 'UTC',
    is_default BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE collab.events (
    event_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    calendar_id UUID NOT NULL REFERENCES collab.calendars(calendar_id) ON DELETE CASCADE,
    title VARCHAR(500) NOT NULL,
    description TEXT,
    location VARCHAR(500),
    start_time TIMESTAMP WITH TIME ZONE NOT NULL,
    end_time TIMESTAMP WITH TIME ZONE NOT NULL,
    all_day BOOLEAN DEFAULT false,
    recurrence_rule TEXT,
    organizer_id UUID REFERENCES users.users(user_id),
    attendees JSONB DEFAULT '[]',
    status VARCHAR(50) DEFAULT 'confirmed',
```

```
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE INDEX idx_events_calendar ON collab.events(calendar_id);
CREATE INDEX idx_events_time_range ON collab.events(start_time, end_time);

-- Analytics Schema
-- (ClickHouse for time-series analytics)
CREATE DATABASE mailflex_analytics;

CREATE TABLE mailflex_analytics.email_events (
    event_id UUID,
    event_type String,
    timestamp DateTime64(3),
    tenant_id UUID,
    user_id UUID,
    message_id UUID,
    mailbox_id UUID,
    from_address String,
    to_addresses Array(String),
    subject String,
    size_bytes UInt64,
    processing_time_ms UInt32,
    spam_score Float32,
    region String
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(timestamp)
ORDER BY (tenant_id, user_id, timestamp);
```

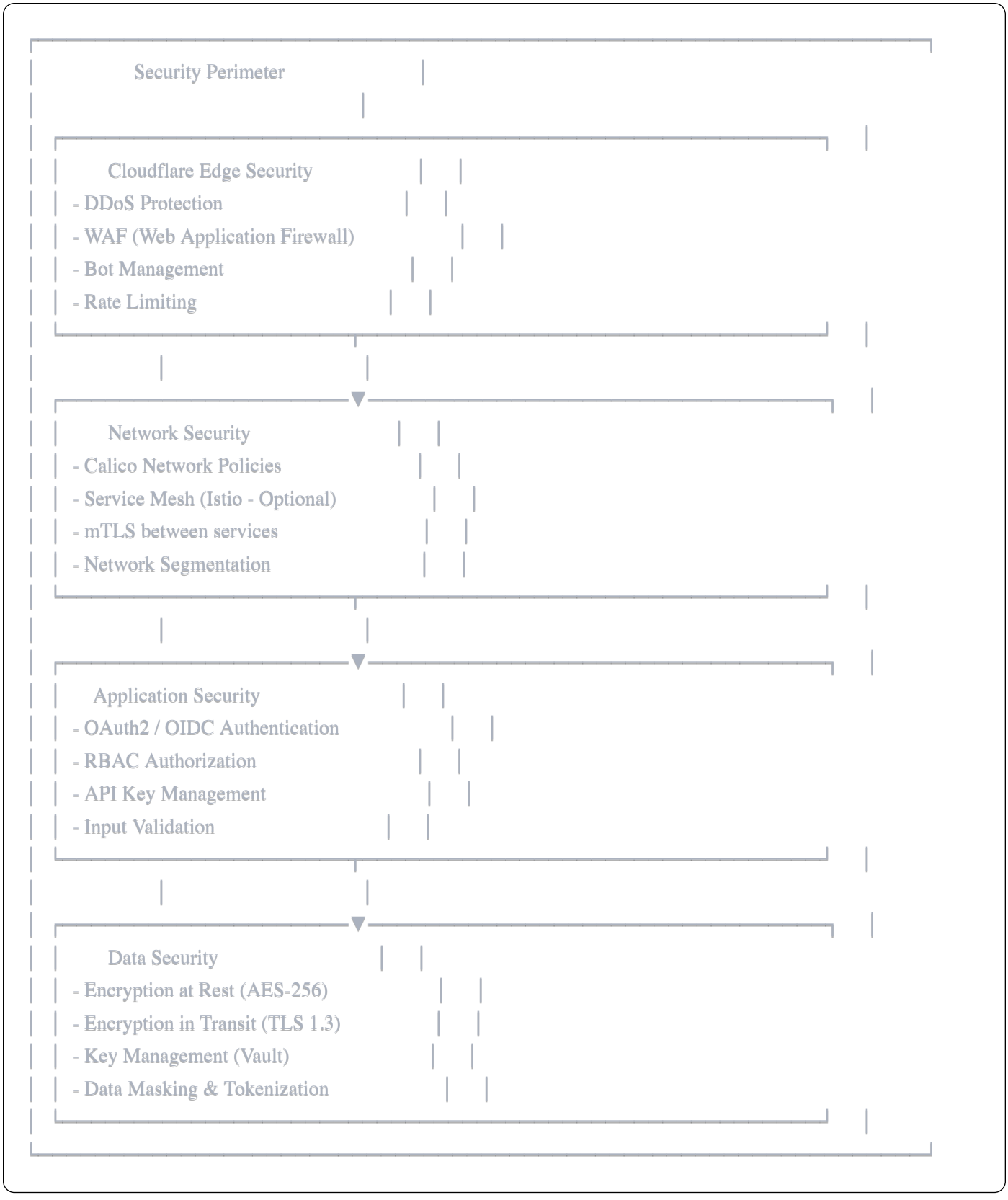
6.3 Caching Strategy



---

7. Security Architecture

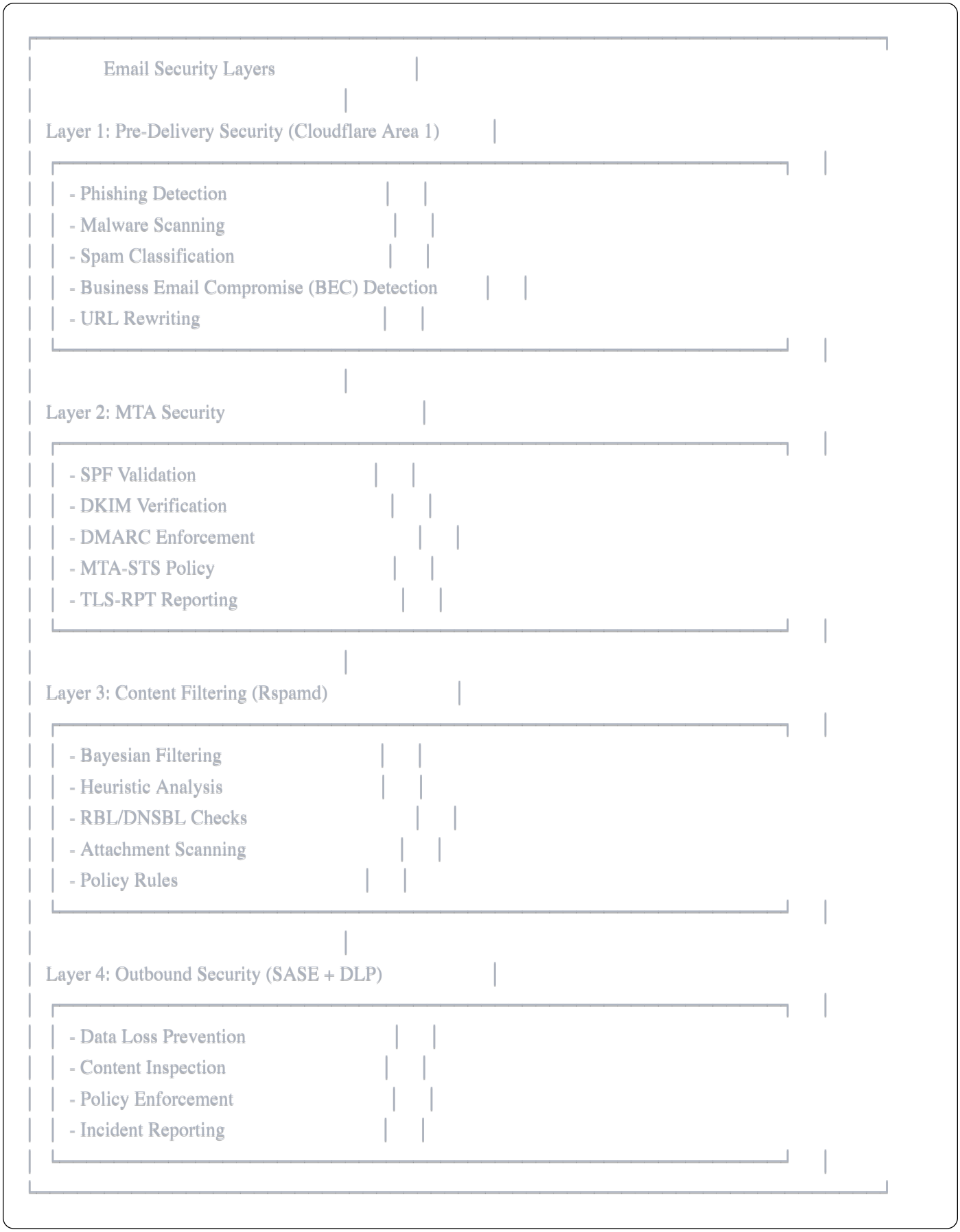
7.1 Zero-Trust Security Model



7.2 Authentication & Authorization Flow

```mermaid
sequenceDiagram
    participant User
    participant Frontend
    participant Keycloak
    participant API
    participant Resource

    User->>Frontend: Login Request
    Frontend->>Keycloak: Redirect to Login
    Keycloak->>User: Present Login Form
    User->>Keycloak: Credentials + MFA
    Keycloak->>Keycloak: Validate Credentials
    Keycloak->>Frontend: Return Access Token
    Frontend->>Frontend: Store Token
    Frontend->>API: API Request + Token
    API->>API: Validate Token (JWT)
    API->>API: Check Permissions (RBAC)
    API->>Resource: Authorized Request
    Resource-->>API: Resource Data
    API-->>Frontend: API Response
    Frontend-->>User: Display Data
```

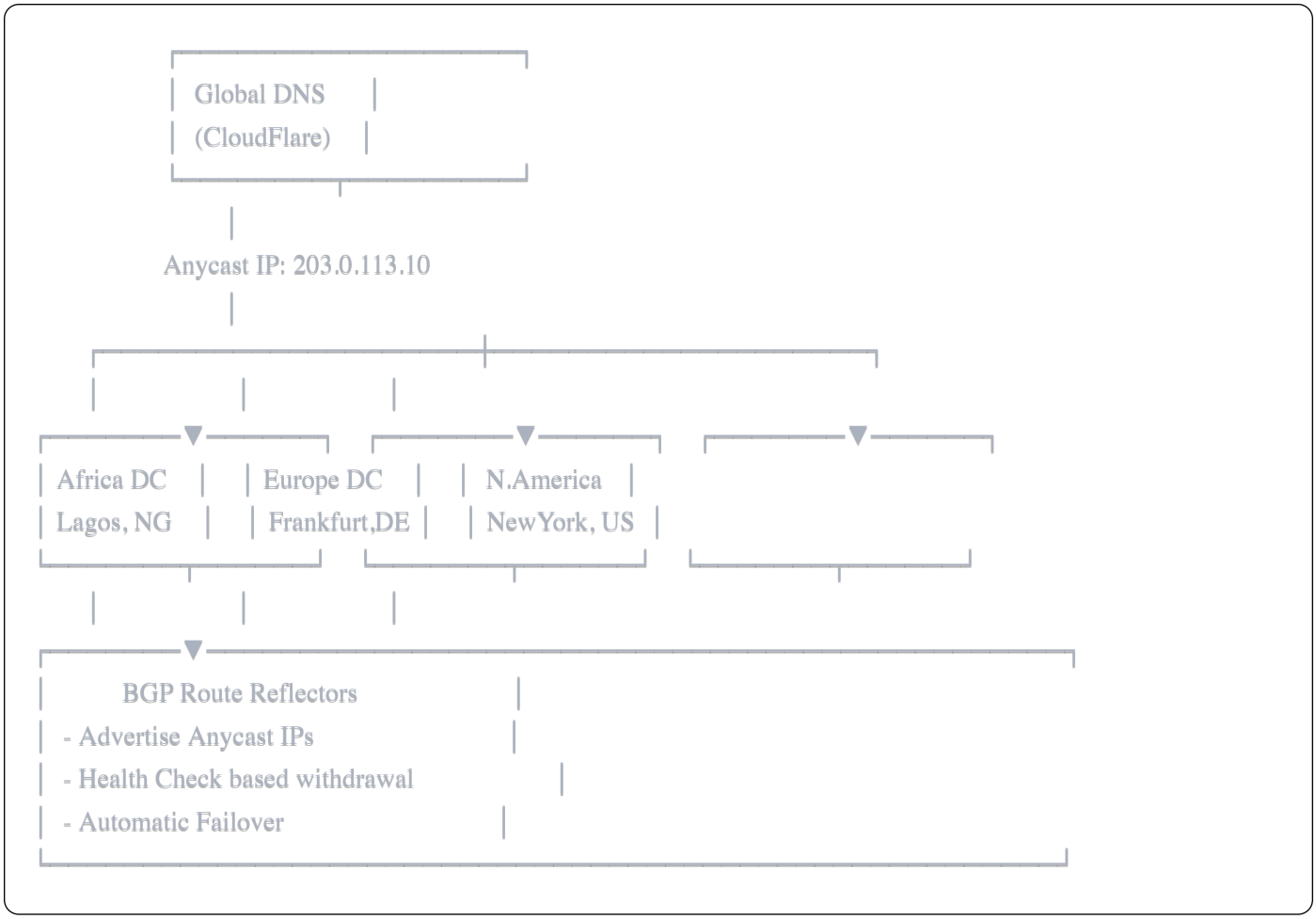
7.3 Email Security Architecture



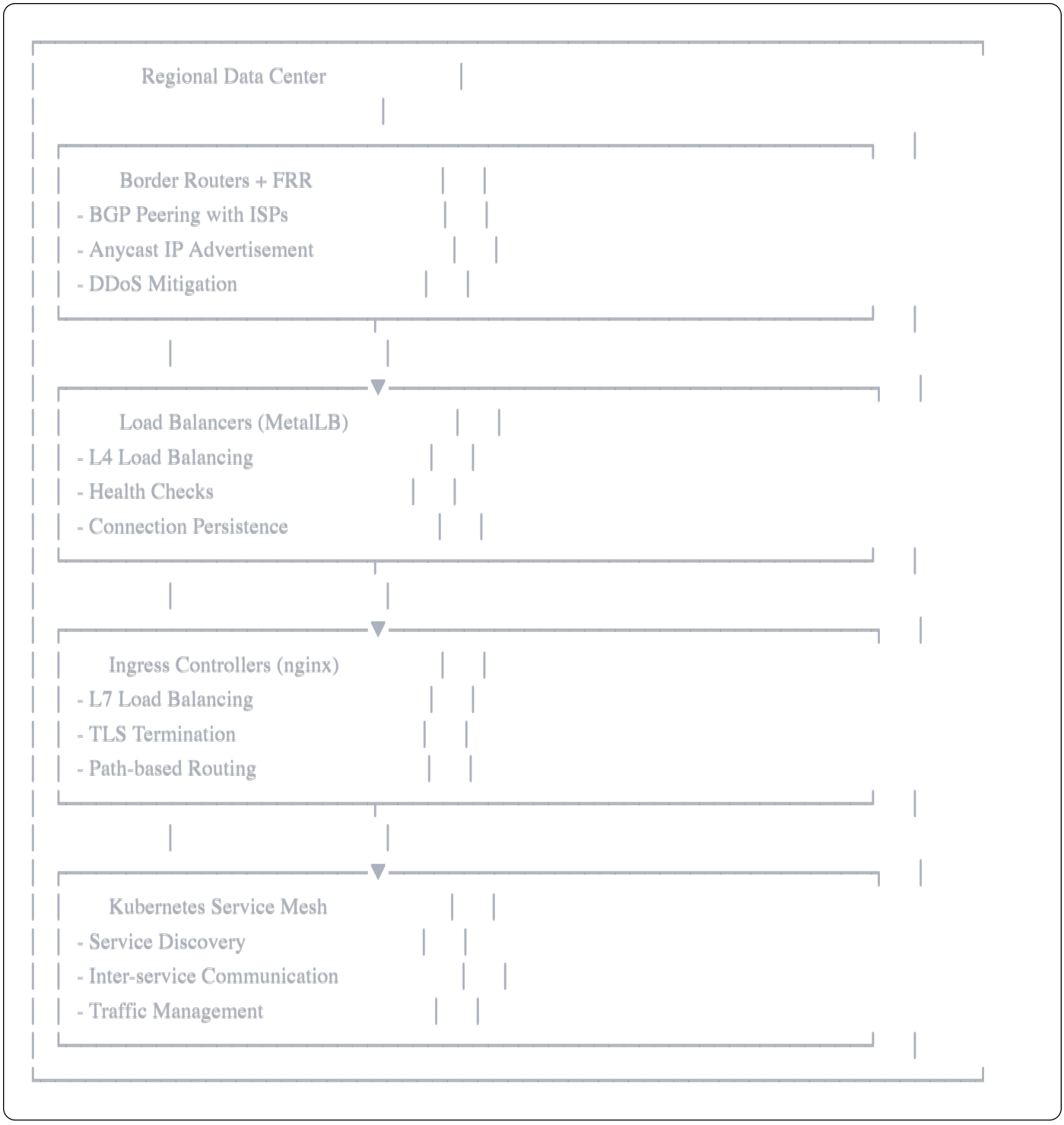
---

8. Network Architecture

8.1 Global Anycast Architecture



8.2 Regional Network Topology



8.3 Service Mesh (Optional - Istio)

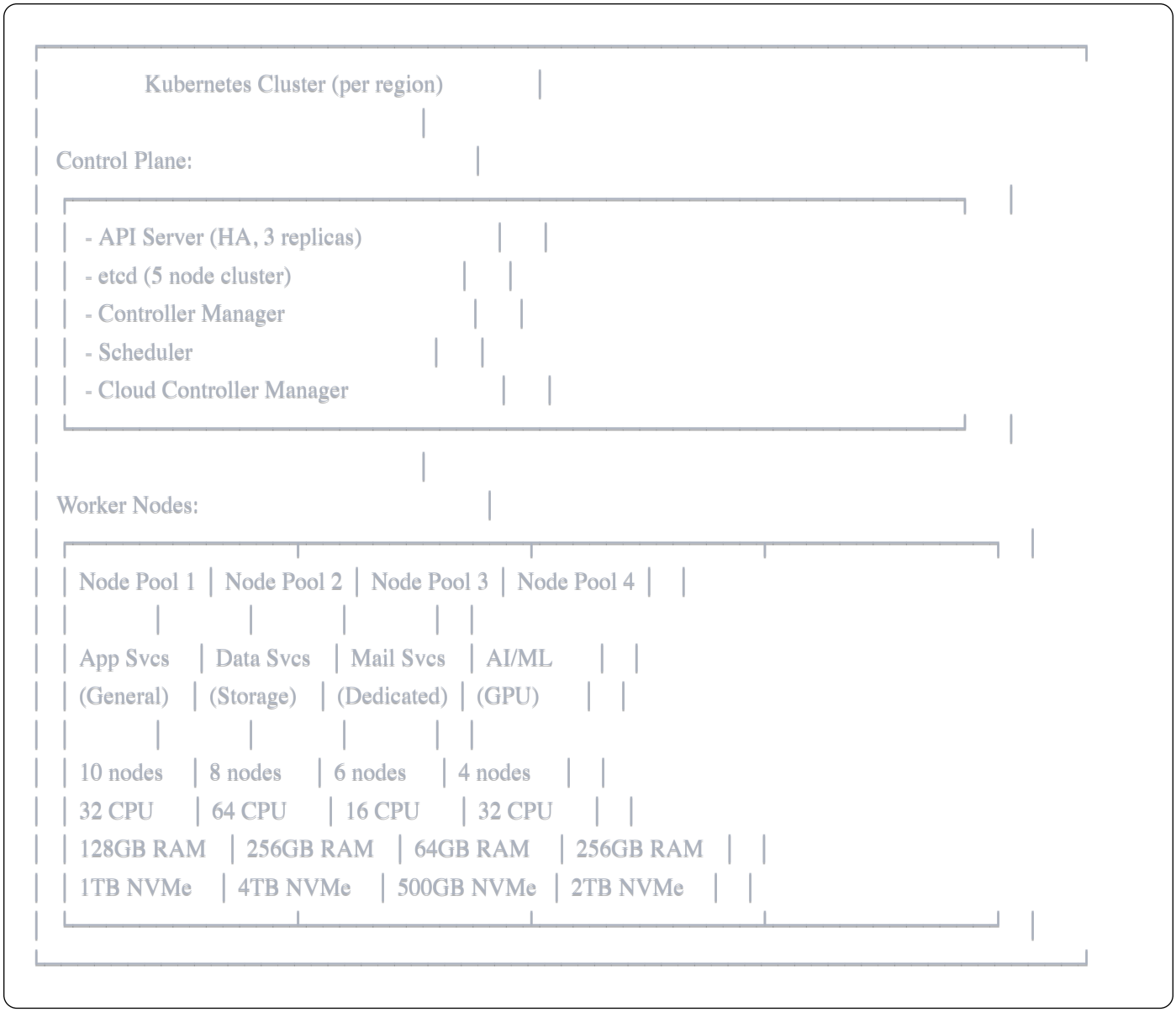
yaml

```yaml
# Istio VirtualService for Traffic Management
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: mail-api
spec:
  hosts:
  - api.mailflex.io
  http:
  - match:
    - headers:
        canary:
          exact: "true"
    route:
    - destination:
        host: mail-api
        subset: canary
      weight: 10
    - destination:
        host: mail-api
        subset: stable
      weight: 90
  - route:
    - destination:
        host: mail-api
        subset: stable
```

---

9. Deployment Architecture

9.1 Kubernetes Cluster Architecture



9.2 Deployment Strategy

yaml

```yaml
# ArgoCD Application for Gitops Deployment
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: mailflex-mail-api
  namespace: argocd
spec:
  project: mailflex
  source:
    repoURL: https://gitlab.com/mailflex/deployments.git
    targetRevision: main
    path: apps/mail-api
    helm:
      values: |
        image:
          repository: registry.mailflex.io/mail-api
          tag: v1.2.3
        replicaCount: 10
        autoscaling:
          enabled: true
          minReplicas: 10
          maxReplicas: 50
          targetCPUUtilizationPercentage: 70
        resources:
          requests:
            cpu: 500m
            memory: 1Gi
          limits:
            cpu: 2000m
            memory: 4Gi
  destination:
    server: https://kubernetes.default.svc
    namespace: mail-services
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

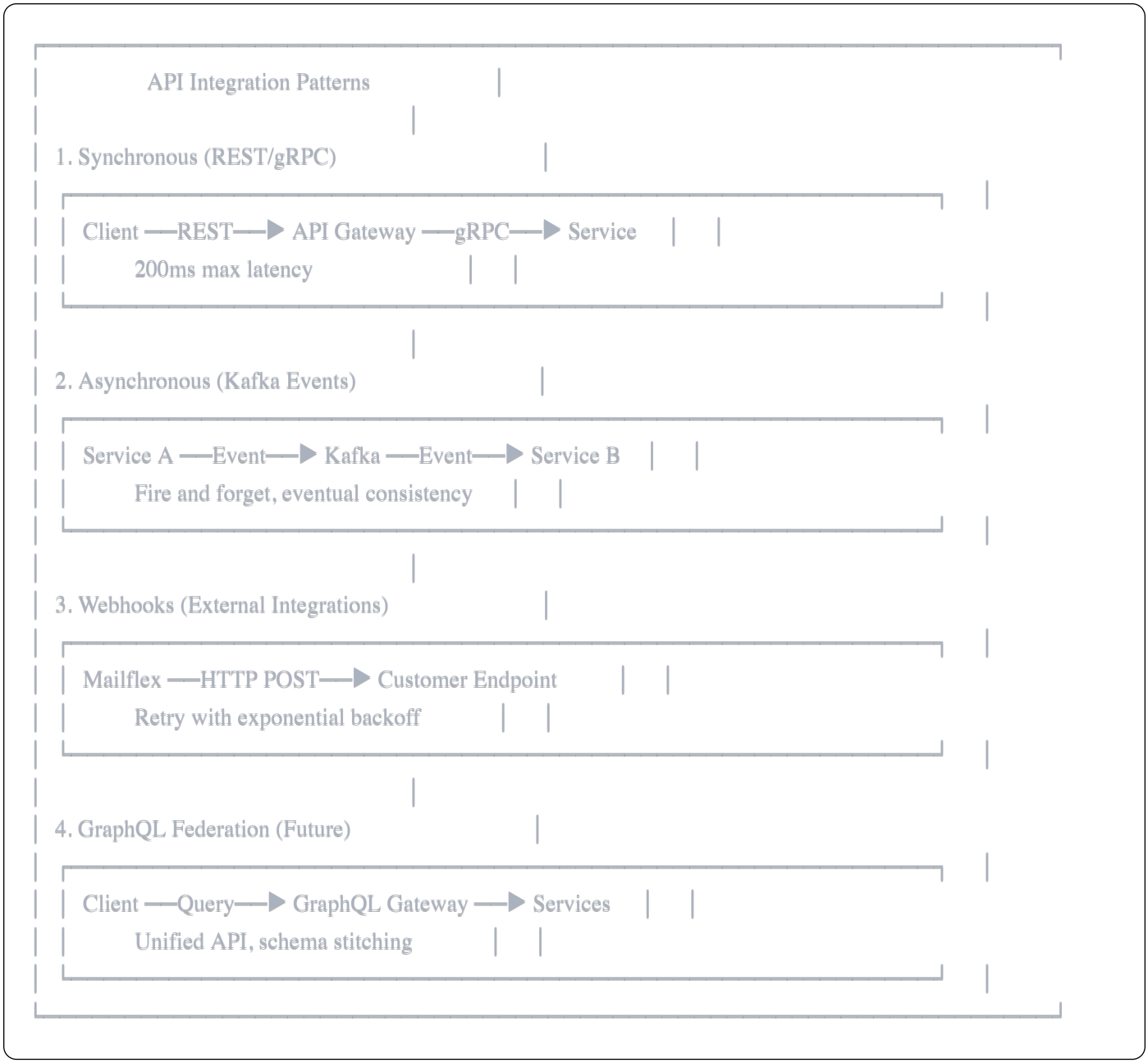
yaml

```yaml
# Argo Rollout for Canary Deployment
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: mail-api
spec:
  replicas: 10
  strategy:
    canary:
      steps:
      - setWeight: 10
      - pause: {duration: 5m}
      - setWeight: 20
      - pause: {duration: 5m}
      - setWeight: 50
      - pause: {duration: 5m}
      - setWeight: 100
      canaryService: mail-api-canary
      stableService: mail-api-stable
      analysis:
        templates:
        - templateName: success-rate
        - templateName: latency
        startingStep: 2
  selector:
    matchLabels:
      app: mail-api
  template:
    metadata:
      labels:
        app: mail-api
    spec:
      containers:
      - name: mail-api
        image: registry.mailflex.io/mail-api:v1.2.3
        ports:
        - containerPort: 8000
          protocol: TCP
```

---

10. Integration Architecture

10.1 API Integration Patterns



10.2 External Service Integration

python

```python
# Example: Payment Gateway Integration
from typing import Optional
from pydantic import BaseModel
import stripe

class PaymentIntent(BaseModel):
    amount: int
    currency: str = "usd"
    customer_id: str
    metadata: dict

class PaymentService:
    def __init__(self, api_key: str):
        stripe.api_key = api_key

    async def create_payment_intent(
        self,
        payment: PaymentIntent
    ) -> dict:
        """Create a payment intent with Stripe"""
        intent = stripe.PaymentIntent.create(
            amount=payment.amount,
            currency=payment.currency,
            customer=payment.customer_id,
            metadata=payment.metadata,
            automatic_payment_methods={
                'enabled': True,
            },
        )
        return intent

    async def handle_webhook(self, payload: dict, sig_header: str):
        """Handle Stripe webhooks"""
        event = stripe.Webhook.construct_event(
            payload, sig_header, self.webhook_secret
        )

        if event['type'] == 'payment_intent.succeeded':
            # Handle successful payment
            await self.on_payment_success(event['data']['object'])
        elif event['type'] == 'payment_intent.payment_failed':
            # Handle failed payment
```

```
        await self.on_payment_failure(event['data']['object'])
```

Appendices

Appendix A: Technology Decision Log

Decision	Rationale	Date
YugabyteDB over PostgreSQL	Multi-region, PostgreSQL-compatible, horizontal scaling	2025-09-01
MinIO over AWS S3	Data sovereignty, cost, S3 compatibility	2025-09-01
Stalwart over Dovecot	Modern architecture, JMAP support, better performance	2025-09-05
Keycloak over custom auth	Feature-rich, standards-compliant, community support	2025-09-10
Kafka over RabbitMQ	Higher throughput, better for event sourcing	2025-09-15

Appendix B: Performance Benchmarks

Component	Metric	Target	Measured
API Gateway	Requests/sec	10,000	12,500
Mail Server	Emails/sec	1,000	1,200
Database	Queries/sec	50,000	55,000
Object Storage	GB/sec	10	12

Appendix C: Glossary

- **Anycast**: Network addressing method where multiple servers share the same IP
- **BGP**: Border Gateway Protocol for routing
- **CQRS**: Command Query Responsibility Segregation
- **DDD**: Domain-Driven Design
- **mTLS**: Mutual Transport Layer Security
- **SLO**: Service Level Objective

Document Control

- **Version**: 1.0
- **Last Updated**: October 13, 2025
- **Next Review**: November 13, 2025
- **Approved By**: Architecture Review Board