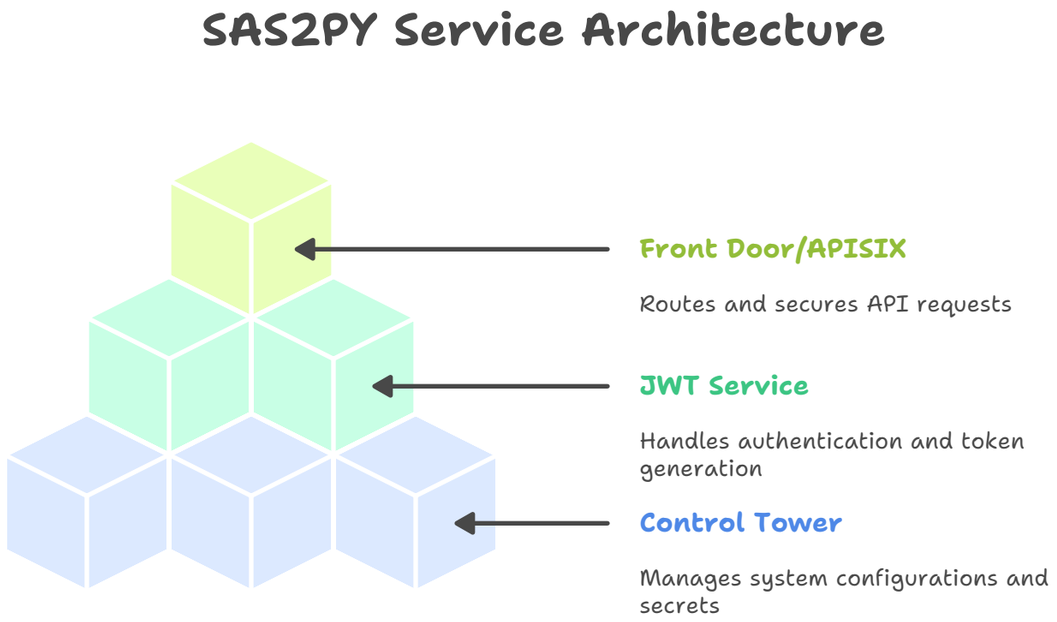
SAS2PY Architecture Summary

Executive Overview

The SAS2PY project implements a secure, enterprise-grade AI service platform for converting SAS code to Python. It leverages three core services (Control Tower, JWT Service, and Front Door/APISIX Gateway) with comprehensive security, multi-environment support, and observability.



System Components

1. Control Tower (Port 8000)

Purpose: Centralized configuration and policy management

Key Responsibilities:

- Store and serve project manifests (JSON configurations)
- Multi-vault secret management (prod-vault, dev-vault)
- Environment variable resolution and substitution
- OPA policy evaluation for access control
- Module dependency validation
- Cross-reference analysis between modules

Vault Integration:

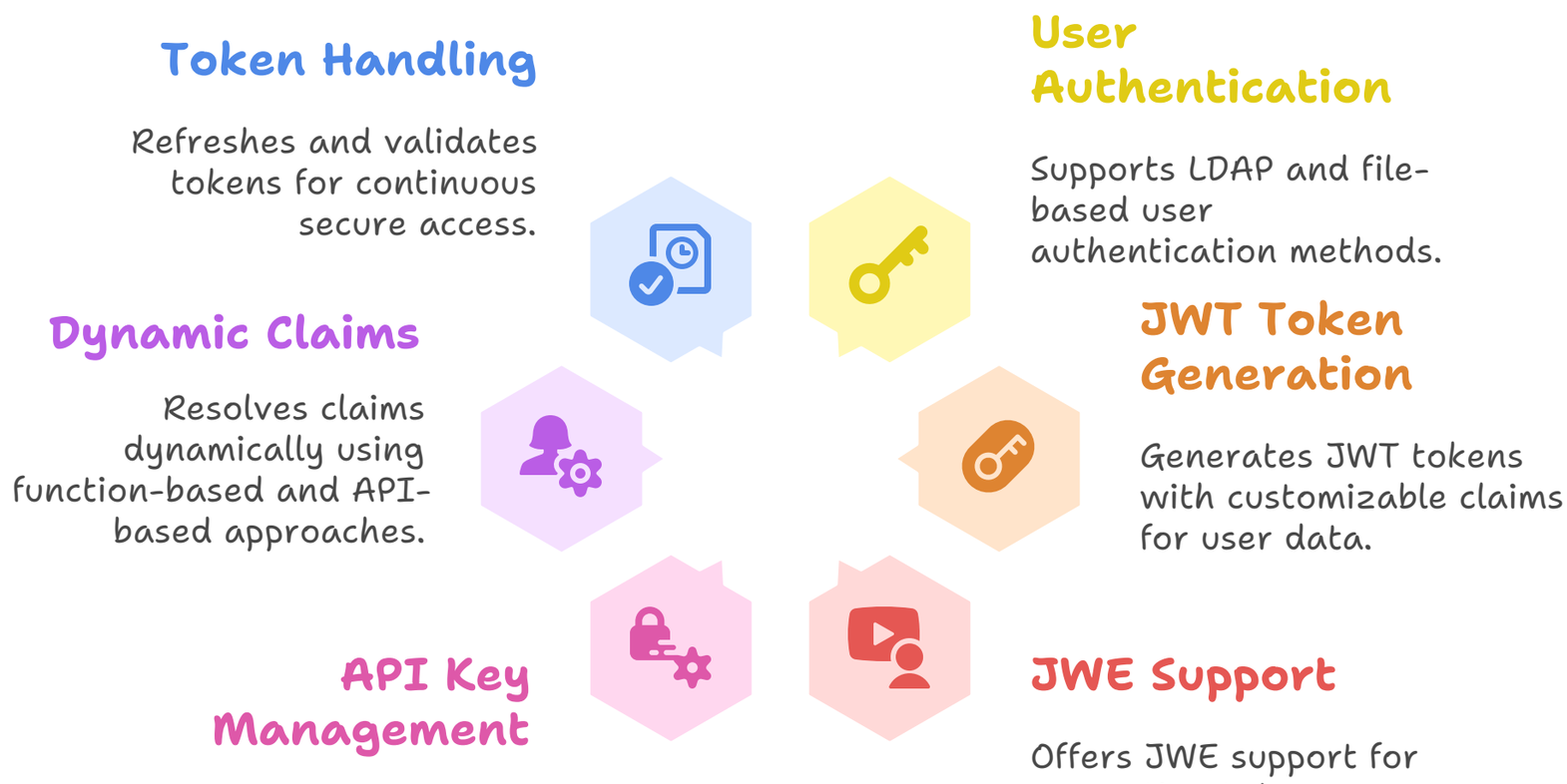
- **Multiple Vault Instances**: Supports both production and development vaults
- **Authentication Methods**: AppRole (prod), Token (dev)
- **Secret Reference Formats**:
    - vault:instance_name:secret/path#key - Vault secrets
    - config:section.key - Config file secrets
    - env:VARIABLE_NAME - Environment variables
    - encrypted:base64_string - Encrypted values

2. JWT Service (Port 5000)

Purpose: Authentication and token management

Key Responsibilities:

- User authentication (LDAP/File-based)
- JWT token generation with custom claims
- JWE (JSON Web Encryption) support for token encryption
- API key configuration management (file-based and inline)
- Dynamic claims resolution (function-based and API-based)
- Token refresh and validation



Authentication Methods:

- **LDAP Integration**: Active Directory authentication
- **File-based**: YAML user configuration with SHA-256 hashed passwords
- **Multi-factor**: Supports group and role-based claims

Claims Architecture:

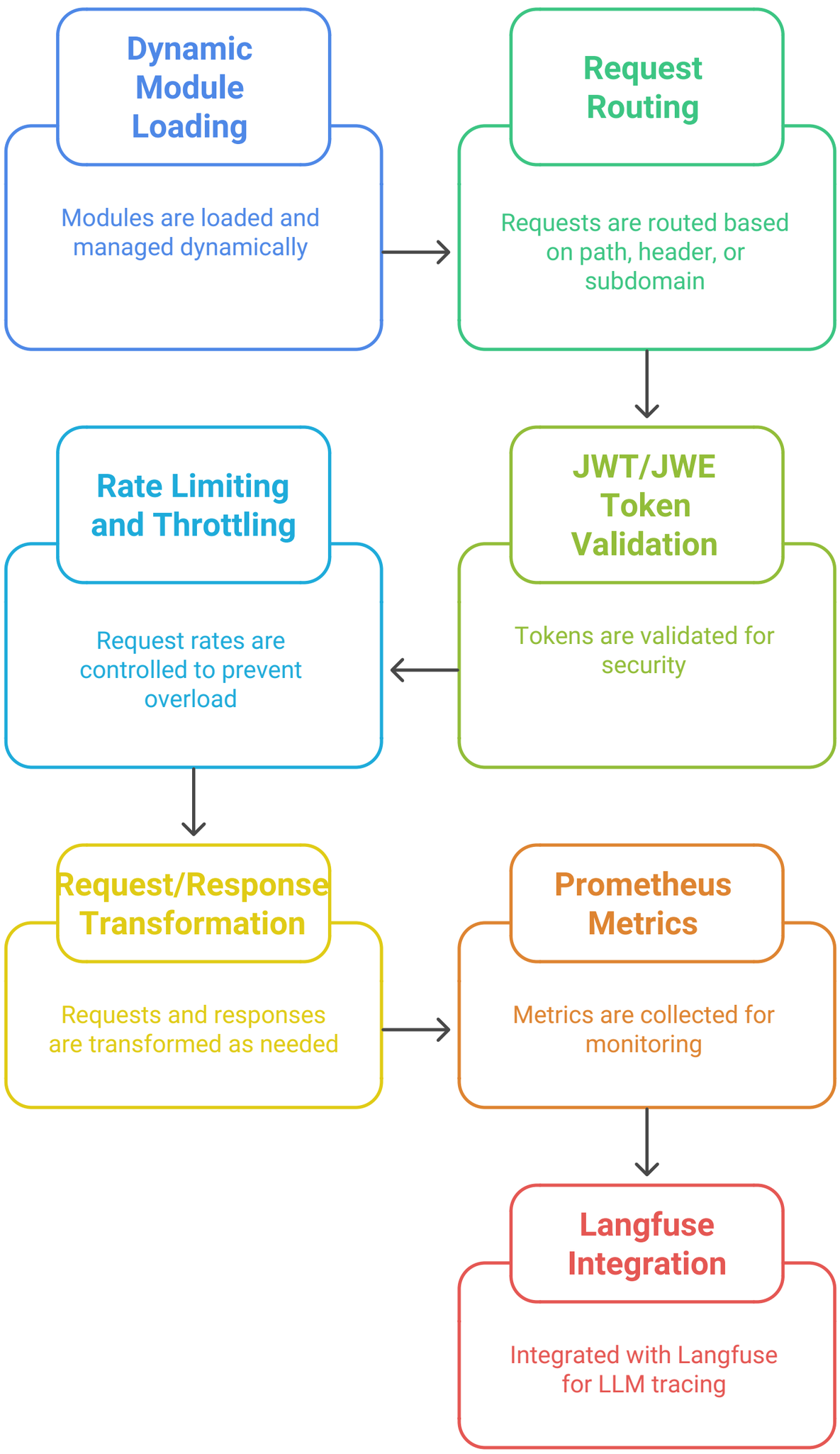
- **Static Claims**: Embedded directly in JWT (tier, models, rate_limit, project, key)
- **Dynamic Claims**: Resolved at runtime via functions or API calls
- **JWE Encryption**: Optional symmetric encryption (A256GCM) for sensitive tokens

3. Front Door / APISIX Gateway (Port 9080)

Purpose: API Gateway and request routing

Key Responsibilities:

- Dynamic module loading and lifecycle management
- Request routing based on path/header/subdomain
- JWT/JWE token validation
- Rate limiting and throttling
- Request/response transformation
- Prometheus metrics and observability
- Integration with Langfuse for LLM tracing



APISIX Plugins Used:

- jwt-auth / jwe-decrypt - Authentication
- ai-prompt-template - Dynamic prompt injection
- proxy-rewrite - Header and URI transformation
- response-rewrite - Response header manipulation
- request-id - Correlation tracking
- prometheus - Metrics collection
- serverless-pre-function - Custom Lua logic

SAS2PY Manifest Analysis

Module Breakdown

1. Vault Manager Module

```
Type: vault
Purpose: Multi-instance secret management
Instances:
  - prod-vault (AppRole auth, SSL verified)
  - dev-vault (Token auth, local)
Features:
  - KV v2 secret engine
  - Secret caching (TTL: 300s)
  - Encryption support
```

2. Authentication Modules

Simple Auth (JWT):

- Consumer key: sas2py-consumer-key
- Static claims: rate_limit=100, project=sas2py
- Environment-specific expiration (1 hour)
- Shared secret with APISIX gateway

JWE Auth (Encrypted JWT):

- Consumer key: sas2py-jwe-consumer-key
- A256GCM encryption (32-byte key)
- Embeds Groq API key in encrypted payload
- Strict validation mode

3. Inference Endpoints

Convert Endpoint (/sas2py/convert):

- Model: llama-3.1-70b-versatile
- Purpose: SAS to Python code conversion
- Temperature: 0.3 (deterministic)
- Max tokens: 2000

Test Endpoint (/sas2py/test):

- Model: llama-3.1-8b-instant
- Purpose: Python unit test generation
- Temperature: 0.5 (creative)
- Max tokens: 1500

Convert-JWE Endpoint (/sas2py/convert-jwe):

- Model: llama-3.1-70b-versatile
- JWE-encrypted authentication
- API key extracted from encrypted payload

OpenAI-Compatible Endpoint (/sas2py/v1/chat/completions):

- No predefined prompt template
- Full OpenAI API compatibility
- Direct passthrough to Groq

4. Monitoring Module

```
Type: monitoring (Langfuse)
Features:
  - LLM observability and tracing
  - 100% sampling rate
  - Project-specific tracking
  - Environment metadata tagging
```

5. API Gateway Routes (APISIX)

Route Configuration Pattern:

```
Consumer → Service → Route → Upstream → Backend (Groq)
```

Common Plugins:

- JWT/JWE authentication

- Request ID generation (UUID)
- Prometheus metrics
- AI prompt template injection
- Proxy rewrite (URI + headers)
- Response header cleanup

Multi-Environment Support

Environment Hierarchy

```
common → development → staging → production
```

Environment-Specific Configuration

Development:
- Local services (localhost URLs)
- Plain text secrets (for testing)
- Langfuse cloud integration
- Groq API base URL

Staging:
- Staging API endpoints
- Environment variable secrets (${STAGING_JWT_SECRET})
- Same Langfuse instance
- SSL verification enabled

Production:
- Production API endpoints
- Vault-backed secrets (${PROD_JWT_SECRET})
- Enhanced security policies
- Full audit logging

Secret Resolution Priority

1. **Vault secrets** (highest priority)
2. **Environment variables** (${VAR_NAME})
3. **Config file** (config:path.to.key)
4. **Encrypted values** (encrypted:base64)
5. **Literal values** (literal:plain)

Security Architecture

Authentication Flow

```
1. User → JWT Service (username/password + api_key)
2. JWT Service → LDAP/File Auth
3. JWT Service → Control Tower (fetch manifest config)
4. JWT Service → Vault (resolve secrets)
5. JWT Service → Generate JWT/JWE token
6. User → Front Door/APISIX (with token)
7. APISIX → Validate JWT/Decrypt JWE
8. APISIX → Extract claims and route
9. APISIX → Backend service (Groq)
```

Token Types

Standard JWT:
- Signed with HS256
- Base64-encoded payload (readable)
- Claims visible to intermediaries
- Validated by APISIX jwt-auth plugin

JWE (Encrypted JWT):
- Encrypted with A256GCM
- Payload completely encrypted
- Requires decryption key
- Validated by custom jwe-decrypt plugin

- API keys embedded in encrypted payload

LDAP Integration

```
Configuration:
  - LDAP Server URL
  - Base DN for user search
  - Admin credentials for binding
  - Group membership extraction
  - Role mapping to claims
```

Static vs Dynamic Claims

Static Claims (embedded in token):

- key: Consumer key for APISIX
- tier: Service tier (premium, standard)
- models: Allowed model list
- rate_limit: Requests per hour
- project: Project identifier
- environment: Deployment environment
- exp_hours: Token expiration

Dynamic Claims (resolved at runtime):

- Quota remaining (function call)
- Usage statistics (API call)
- Real-time permissions (external service)
- Cost tracking (billing API)

Request Flow Diagrams

Standard JWT Flow (Convert Endpoint)

```
Client
   │
   ├──? POST /token (JWT Service)
   │   ├── username/password
   │   └── api_key: "sas2py-consumer-key"
   │
   ├──? JWT Token (signed, readable claims)
   │
   ├──? POST /sas2py/convert (APISIX Gateway)
   │   ├── Authorization: Bearer <JWT>
   │   └── Body: {"user_input": "SAS code..."}
   │
   ├──? APISIX Processing
   │   ├── jwt-auth plugin (validate signature)
   │   ├── request-id plugin (generate UUID)
   │   ├── ai-prompt-template plugin (inject system prompt)
   │   ├── prometheus plugin (record metrics)
   │   └── proxy-rewrite plugin (transform to Groq format)
   │
   ├──? Groq API
   │   └── POST /openai/v1/chat/completions
   │
   ├──? LLM Response (Python code)
```

JWE Encrypted Flow (Convert-JWE Endpoint)

```
   ├──? response-rewrite plugin (clean headers)
   │
   └──? Client receives converted code
```

```
Client
   │
   ├──? POST /token (JWT Service)
   │   ├── username/password
   │   └── api_key: "sas2py-jwe-consumer-key"
   │
   ├──? JWE Token (encrypted, unreadable)
   │   └── Contains: JWT + Groq API Key (encrypted)
   │
   ├──? POST /sas2py/convert-jwe (APISIX Gateway)
   │   ├── Authorization: Bearer <JWE>
   │   └── Body: {"user_input": "SAS code..."}
   │
   ├──? APISIX Processing
   │   ├── jwe-decrypt plugin (decrypt JWE → JWT)
   │   ├── Extract claims from decrypted JWT
   │   ├── Extract groq_api_key from payload
   │   ├── Forward decrypted payload in X-JWE-Payload header
   │   ├── serverless-pre-function (extract API key)
   │   ├── ai-prompt-template plugin (inject prompt)
   │   └── proxy-rewrite plugin (use extracted API key)
   │     └── Authorization: Bearer $http_x_groq_api_key
   │
   ├──? Groq API (with decrypted API key)
```

Control Tower Manifest Resolution

```
Client/Service
   │
   ├──? GET /manifests/sas2py (Control Tower)
   │
   ├──? Control Tower Processing
   │   ├── Load manifest JSON from disk
   │   ├── Identify current environment
   │   ├── Apply environment overrides
   │   └── Resolve variables:
   │     ├── ${environment} → "development"
   │     ├── ${environments.development.urls.jwt_service_url}
   │     │  → "http://localhost:5000"
   │     ├── ${environments.common.secrets.groq_api_key}
   │     │  → "config:api_keys.groq"
   │     └── config:api_keys.groq
   │       └──? Vault/Config lookup → actual API key
```

Vault Secret Resolution

```
   └──? Resolved manifest with all secrets and URLs
```

```
Application
   │
   ├──? Request secret: "vault:prod-vault:myapp/database#password"
   │
   ├──? Secret Manager (Control Tower)
   │   ├── Parse reference format
   │   ├── Identify vault instance: "prod-vault"
   │   ├── Extract path: "myapp/database"
   │   └── Extract key: "password"
   │
   ├──? Multi-Vault Manager
   │   ├── Select vault instance config
   │   ├── Authenticate (AppRole/Token)
   │   └── Read secret from KV v2
   │
   ├──? Vault API
   │   └── GET /v1/secret/data/myapp/database
   │
   ├──? Secret data: {"password": "actual_password"}
   │
   ├──? Cache secret (TTL: 300s)
   │
   └──? Return: "actual_password"
```

Integration Points

1. Control Tower ↔ JWT Service

Direction: JWT Service → Control Tower
Purpose: Fetch JWT configuration from manifest
Flow:

```
JWT Service receives token request
  → GET /manifests/sas2py?resolve_env=true
  → Extract jwt_config module
  → Use claims configuration
  → Generate token with manifest-defined claims
```

2. Control Tower ↔ Front Door

Direction: Front Door → Control Tower
Purpose: Fetch routing and module configuration
Flow:

```
Front Door startup
  → GET /manifests/sas2py
  → Load all api_gateway modules
  → Configure APISIX routes dynamically
  → Set up consumers and plugins
  → Cache manifest (TTL: 300s)
```

3. JWT Service ↔ LDAP

Direction: JWT Service → LDAP Server
Purpose: User authentication and group membership
Flow:

```
User login request
  → Bind to LDAP with admin credentials
  → Search for user DN
  → Authenticate user credentials
  → Query group membership
  → Extract roles and attributes
  → Map to JWT claims
```

4. Control Tower ↔ Vault

Direction: Control Tower → Multiple Vault Instances
Purpose: Resolve secrets from multiple sources
Flow:

```
Manifest resolution request
  → Identify all secret references
  → For each vault reference:
      → Select vault instance
      → Authenticate (AppRole/Token)
      → Read secret
      → Cache result
  → Substitute resolved values
  → Return complete manifest
```

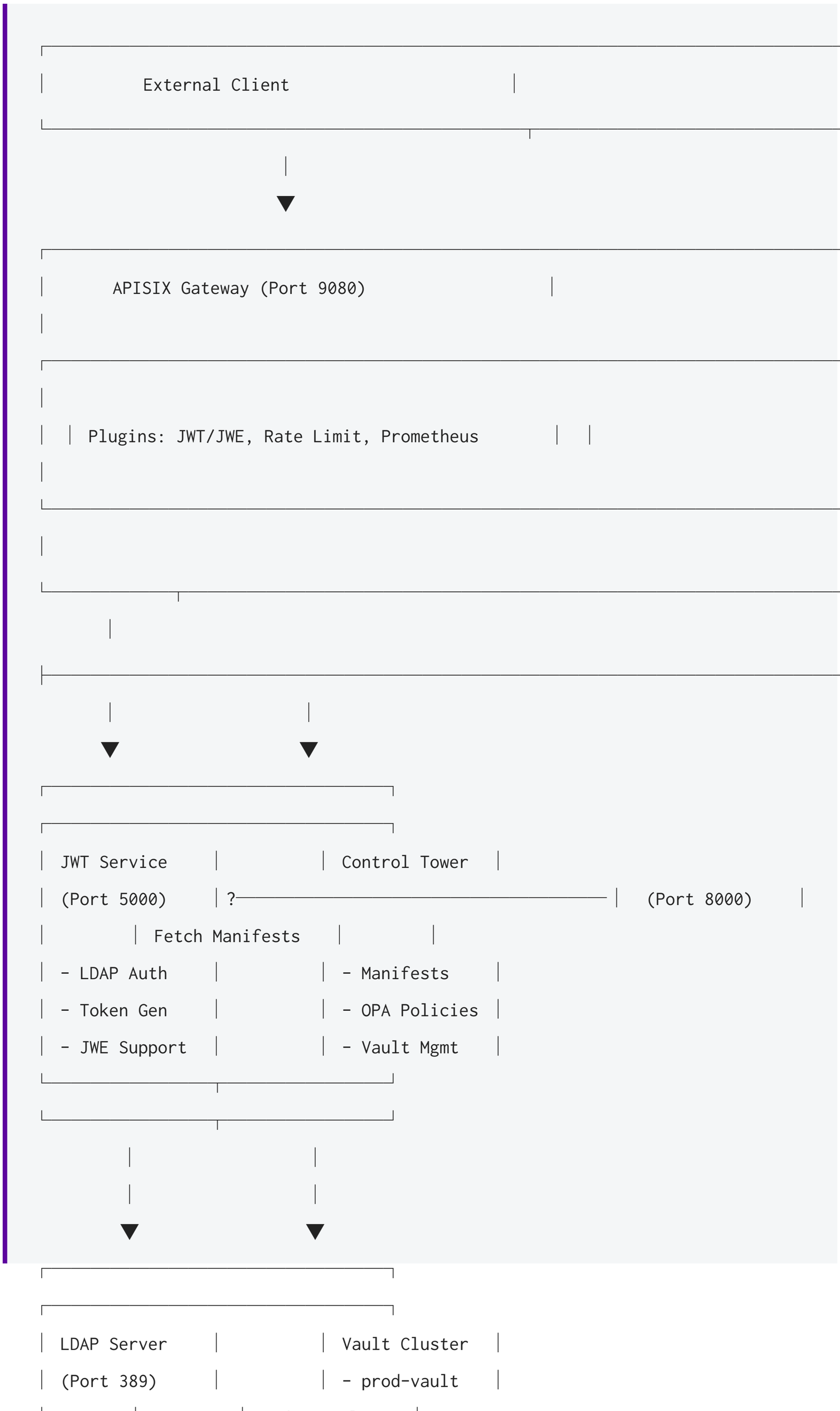
5. APISIX ↔ Langfuse

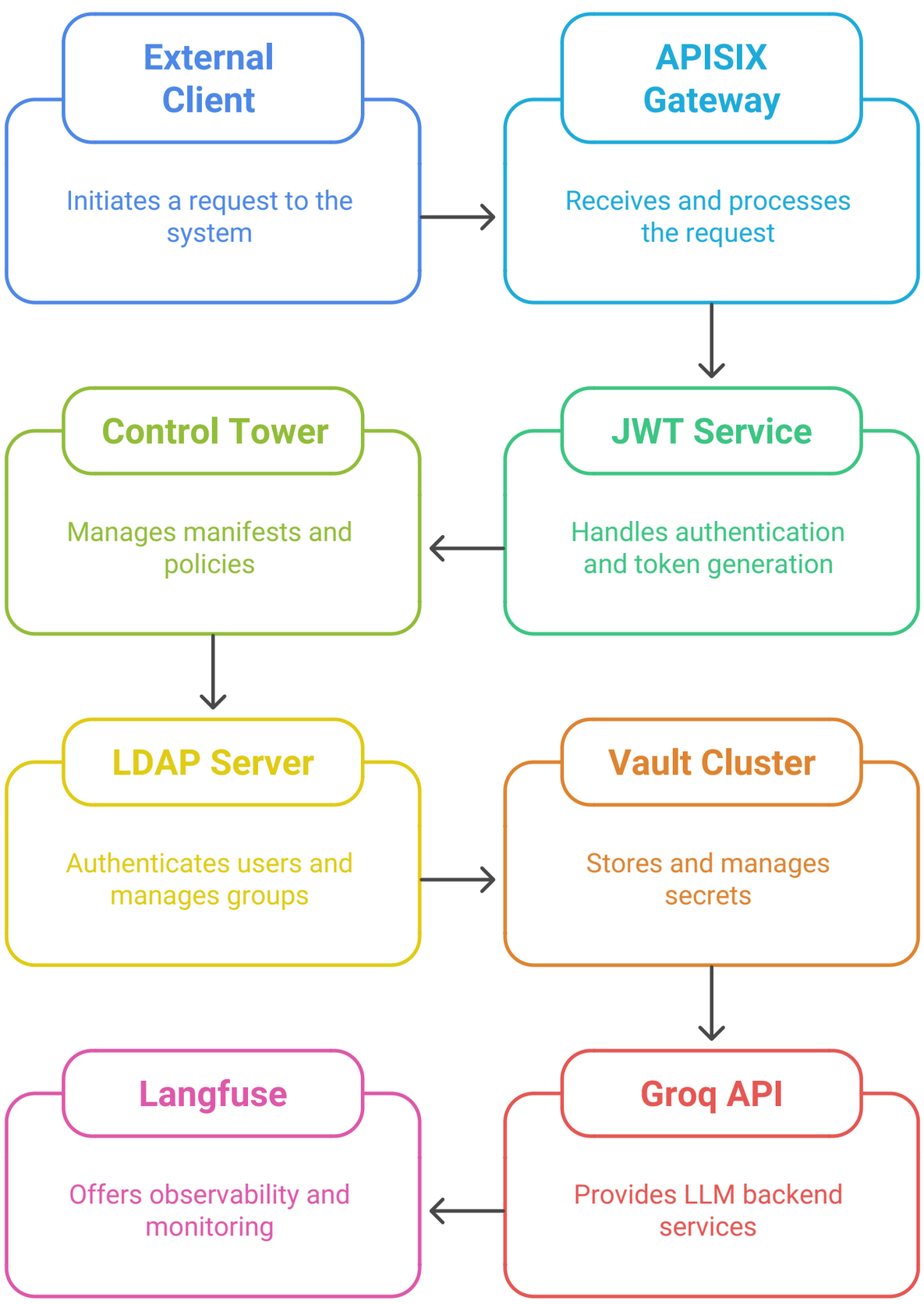
Direction: APISIX → Langfuse Cloud
Purpose: LLM observability and tracing
Flow:

```
LLM request through APISIX
  → Capture request metadata
  → Forward to LLM backend
  → Capture response and latency
  → Send trace to Langfuse
      → Public key authentication
      → Project: sas2py
      → Environment tag
      → Request/response payload
```

Deployment Architecture

Service Topology


External Client
APISIX Gateway (Port 9080)
Plugins: JWT/JWE, Rate Limit, Prometheus
JWT Service
(Port 5000)
Control Tower
(Port 8000)
Fetch Manifests
- LDAP Auth
- Token Gen
- JWE Support
- Manifests
- OPA Policies
- Vault Mgmt
LDAP Server
(Port 389)
Vault Cluster
- prod-vault
- dev-vault



Environment-Specific Deployment

Development:

- All services on localhost
- Single-node deployment
- File-based secrets
- Dev vault with token auth
- Minimal security policies

Staging:

- Distributed services
- Staging domain endpoints
- Environment variable secrets
- Prod vault with AppRole
- Enhanced logging

Production:

- High-availability setup
- Load-balanced APISIX
- Vault-backed secrets
- Full audit logging
- Rate limiting enforced
- SSL/TLS everywhere

Key Features Summary

1. Multi-Vault Secret Management

- Support for multiple Vault instances
- Different auth methods per instance (AppRole, Token)

- Unified secret reference format
- Fallback to config files and env vars
- Secret caching for performance

2. Flexible Authentication

- LDAP/Active Directory integration
- File-based authentication
- Group and role-based claims
- API key management (file and inline)
- JWE encryption for sensitive tokens

3. Dynamic Claims System

- Static claims (embedded in token)
- Dynamic claims (runtime resolution)
- Function-based claim providers
- API-based claim providers
- Context-aware placeholders

4. Multi-Environment Support

- Environment-specific configurations
- Variable substitution patterns
- Environment overrides per module
- Seamless promotion (dev → staging → prod)

5. Comprehensive Observability

- Langfuse LLM tracing
- Prometheus metrics
- Request correlation (X-Request-Id)
- Structured logging
- Performance monitoring

6. API Gateway Features

- JWT/JWE authentication
- Rate limiting per consumer
- AI prompt template injection
- Request/response transformation
- Load balancing
- Health checks

7. Security Best Practices

- Token encryption (JWE)
- Secret rotation support
- Least privilege access
- Audit logging
- SSL/TLS enforcement
- Input validation

Configuration Examples

JWT Module Configuration

```
{
  "module_type": "jwt_config",
  "name": "simple-auth",
  "config": {
    "service_url": "${environments.${environment}.urls.jwt_service_url}",
    "claims": {
      "static": {
        "key": "sas2py-consumer-key",
        "rate_limit": 100,
        "project": "sas2py",
        "environment": "${environment}",
        "exp_hours": 1
      }
    }
  }
}
```

APISIX Route Configuration

```
{
  "module_type": "api_gateway",
  "name": "apisix-convert-route",
  "config": {
    "consumer": {
      "username": "consumer",
      "plugins": {
        "jwt-auth": {
          "key": "sas2py-consumer-key",
          "secret": "${environments.${environment}.secrets.jwt_secret_key}"
        }
      }
    },
    "routes": [{
      "uri": "/sas2py/convert",
      "methods": ["POST"],
      "plugins": {
        "jwt-auth": {},
        "ai-prompt-template": {...},
        "proxy-rewrite": {...}
      }
    }]
  }
}
```

Vault Module Configuration

```
{
  "module_type": "vault",
  "name": "vault-manager",
  "config": {
    "vault_instances": [
      {
        "instance_name": "prod-vault",
        "vault_url": "https://vault-prod.example.com:8200",
        "auth_method": "approle",
        "role_id": "env:PROD_VAULT_ROLE_ID",
        "secret_id": "env:PROD_VAULT_SECRET_ID"
      }
    ]
  }
}
```

Performance Considerations

Caching Strategy

- **Manifest Cache**: 300s TTL (Control Tower)
- **Secret Cache**: 300s TTL (Vault)
- **Token Cache**: Based on JWT expiration
- **Module Cache**: In-memory, LRU eviction

Scalability

- **Horizontal Scaling**: APISIX and Front Door
- **Vertical Scaling**: Control Tower (manifest storage)
- **Database**: PostgreSQL for audit logs
- **Cache**: Redis for distributed caching

Latency Optimization

- Secret caching reduces Vault calls
- Manifest caching reduces Control Tower calls
- Connection pooling for LDAP
- HTTP/2 for APISIX upstream
- Streaming responses for LLM

Monitoring and Alerting

Metrics Collected

- Request count and latency (per route)
- Token generation rate
- Authentication failures
- Vault secret access
- LLM token usage
- Error rates by type

Langfuse Observability

- Full request/response traces
- Token usage tracking
- Model performance metrics
- Cost attribution
- User behavior analytics

Health Checks

- /health endpoint on all services
- Vault connectivity checks
- LDAP connectivity checks
- APISIX upstream health
- Database connection pool

Security Compliance

Data Protection

- Secrets never logged
- PII encryption in transit
- Token encryption (JWE)
- Audit trail for all access

Access Control

- OPA policy evaluation
- Role-based access (RBAC)
- Group-based permissions
- Rate limiting per user

Compliance Features

- HIPAA-ready (JWE encryption)
- SOC2 audit logging
- GDPR data handling
- PCI DSS secret management

Troubleshooting Guide

Common Issues

JWT Validation Failures:

- Check secret key match between JWT Service and APISIX
- Verify token expiration
- Ensure consumer key exists in APISIX

JWE Decryption Failures:

- Verify encryption key length (32 bytes for A256GCM)
- Check algorithm match
- Ensure key is base64-encoded

Vault Connection Issues:

- Verify vault URL and port
- Check authentication credentials
- Ensure network connectivity
- Validate SSL certificates

LDAP Authentication Failures:

- Verify LDAP server URL
- Check admin credentials
- Validate user DN format
- Test group membership queries

Environment Variable Resolution:

- Check placeholder syntax: ${environments.${environment}.key}
- Verify environment name matches
- Ensure secrets exist in config

Future Enhancements

Planned Features

- Multi-region deployment
- Advanced rate limiting (token bucket)
- Circuit breaker patterns
- A/B testing support
- Canary deployments
- GraphQL support
- WebSocket support
- gRPC protocol support

Security Enhancements

- mTLS for service-to-service
- Hardware security module (HSM) integration
- Biometric authentication
- Zero-trust architecture
- Automated secret rotation

Observability Enhancements

- Distributed tracing (OpenTelemetry)
- Real-time dashboards
- Anomaly detection

- Cost optimization insights
- Predictive scaling

Diagram Legend

Components:

- **CT**: Control Tower
- **JWT**: JWT Service
- **FD**: Front Door
- **APISIX**: API Gateway
- **Vault**: HashiCorp Vault
- **LDAP**: LDAP/Active Directory
- **LF**: Langfuse

Data Flows:

- →: Synchronous request
- ?: Asynchronous event
- ↔: Bidirectional communication
- ⊗: Authentication required
- ⊕: Optional component

This architecture provides a robust, secure, and scalable platform for AI service delivery with comprehensive secret management, multi-environment support, and enterprise-grade observability.