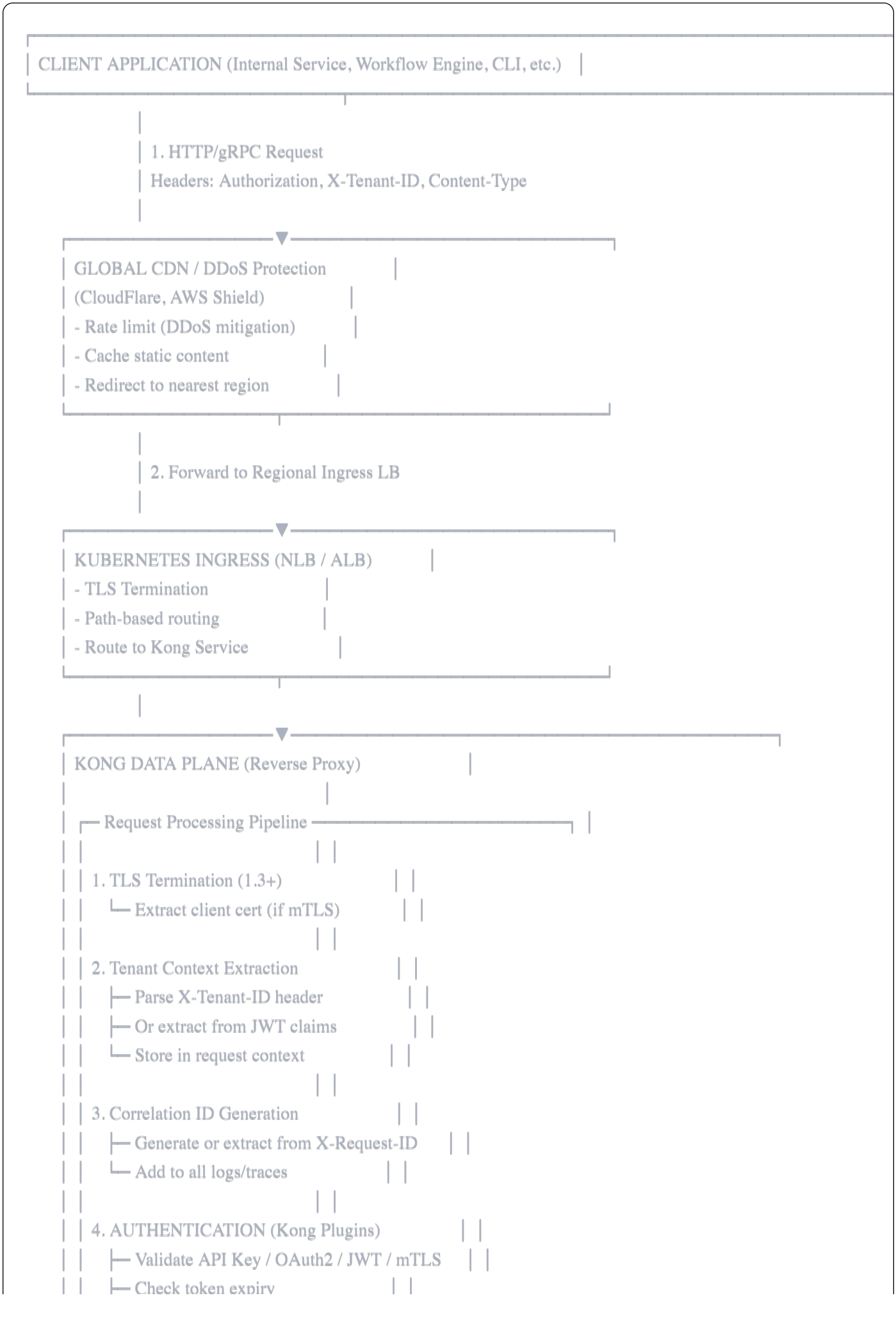
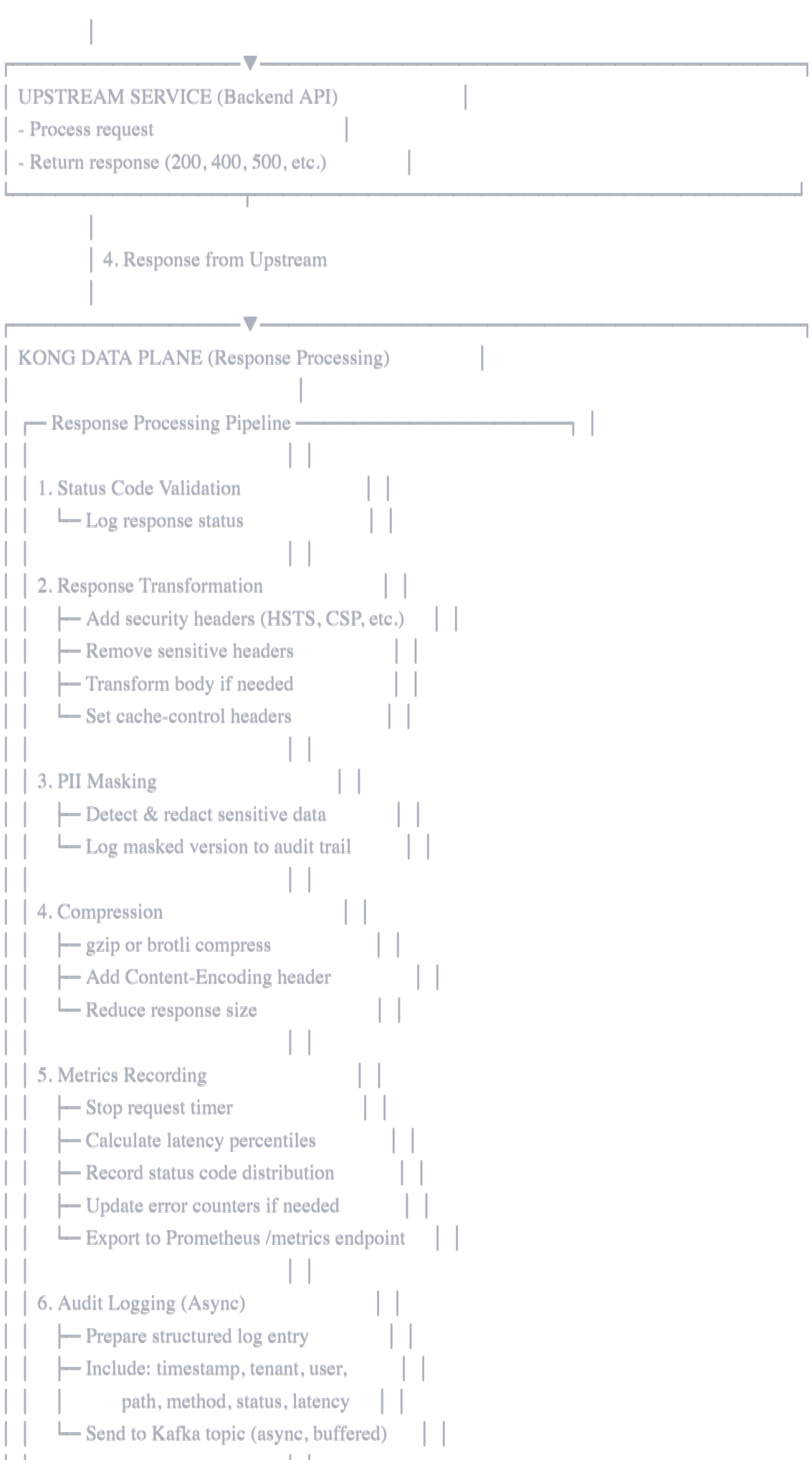
Enterprise API Management Platform - Architecture Diagrams & Component Interactions

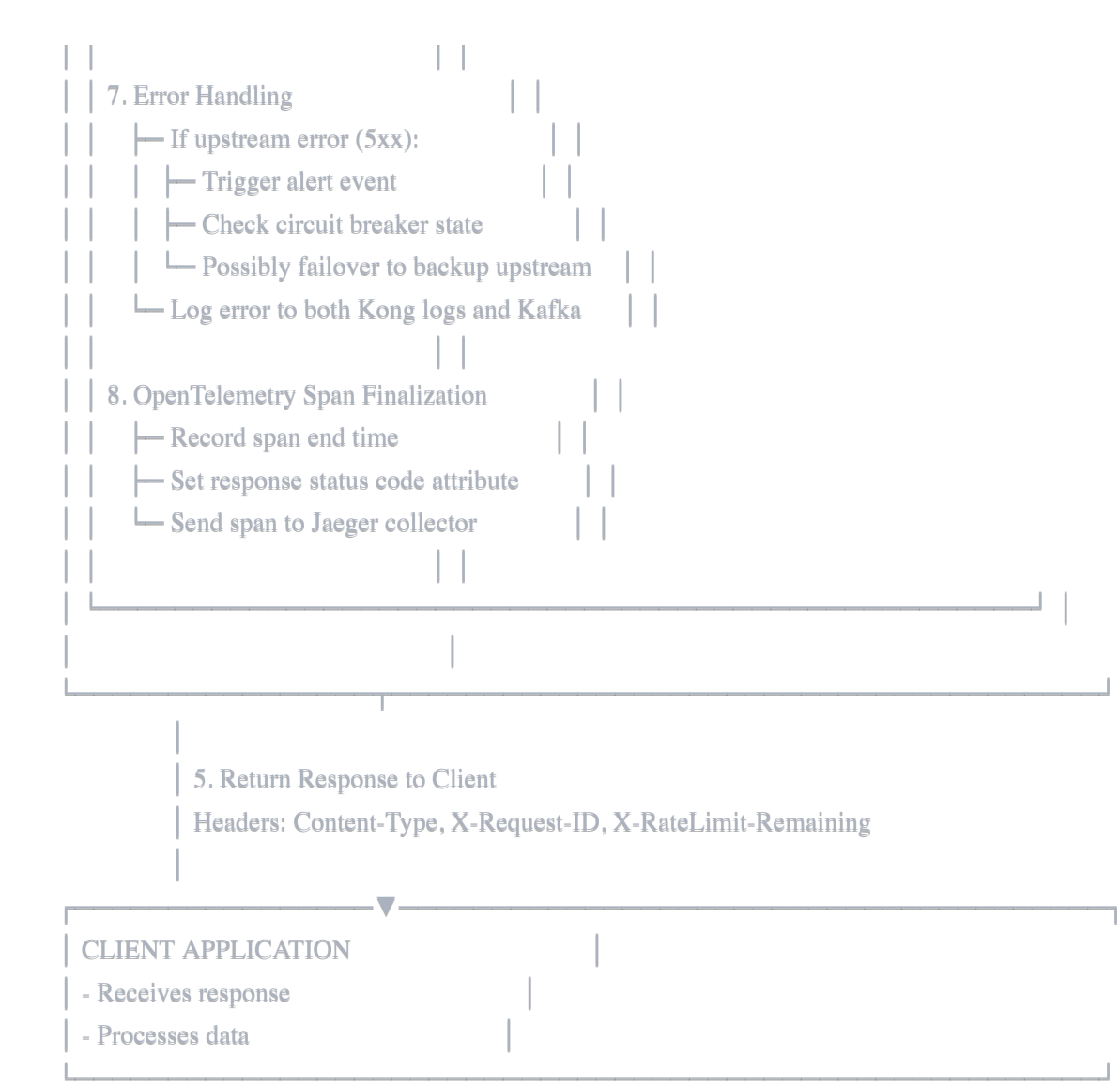
---

1. REQUEST FLOW DIAGRAM (End-to-End)

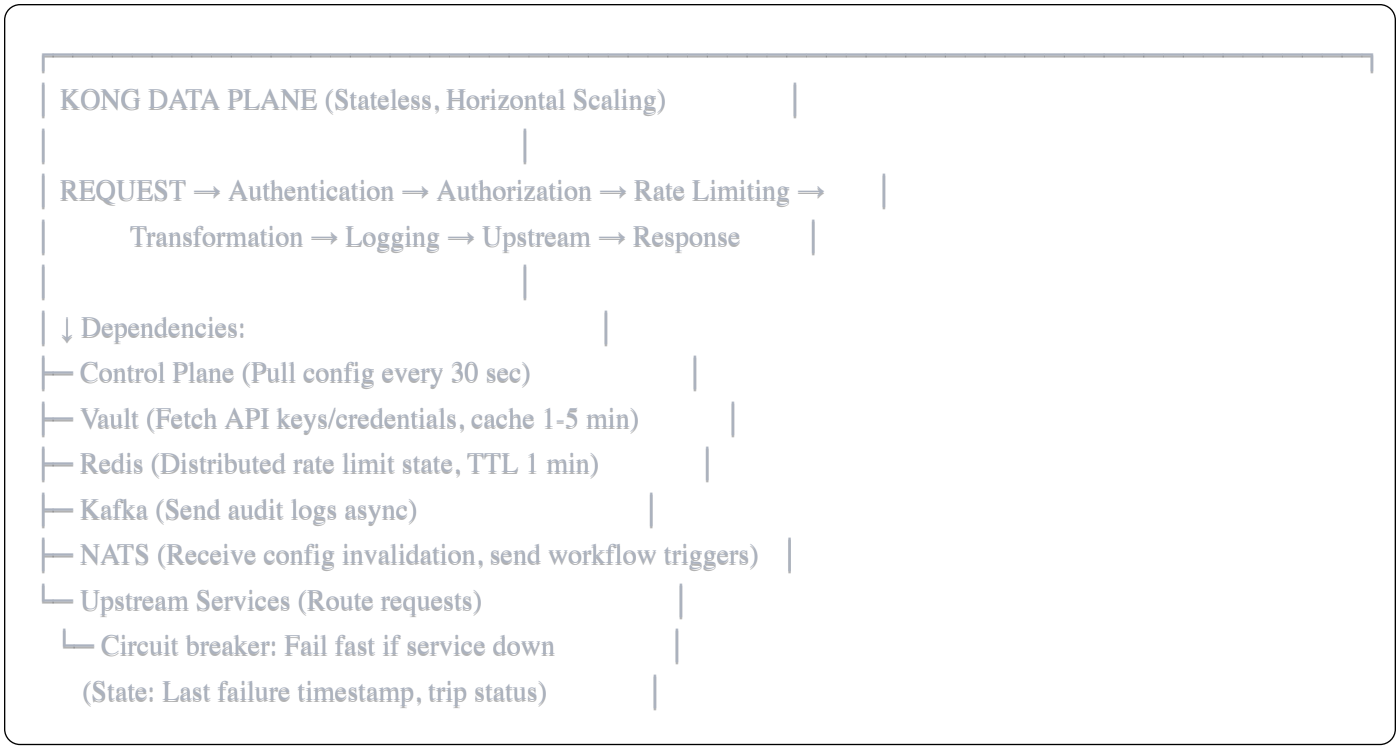

CLIENT APPLICATION (Internal Service, Workflow Engine, CLI, etc.)
1. HTTP/gRPC Request
Headers: Authorization, X-Tenant-ID, Content-Type
GLOBAL CDN / DDoS Protection
(CloudFlare, AWS Shield)
- Rate limit (DDoS mitigation)
- Cache static content
- Redirect to nearest region
2. Forward to Regional Ingress LB
KUBERNETES INGRESS (NLB / ALB)
- TLS Termination
- Path-based routing
- Route to Kong Service
KONG DATA PLANE (Reverse Proxy)
Request Processing Pipeline
1. TLS Termination (1.3+)
Extract client cert (if mTLS)
2. Tenant Context Extraction
Parse X-Tenant-ID header
Or extract from JWT claims
Store in request context
3. Correlation ID Generation
Generate or extract from X-Request-ID
Add to all logs/traces
4. AUTHENTICATION (Kong Plugins)
Validate API Key / OAuth2 / JWT / mTLS
Check token expiry


├─ Cache validation result (2 min TTL)
└─ If invalid: Return 401 + Log to Kafka
5. AUTHORIZATION (RBAC/ABAC)
├─ Check tenant access to route
├─ Check user permissions
├─ Attribute-based policy evaluation
└─ If denied: Return 403 + Log violation
6. RATE LIMITING
├─ Fetch current rate limit counter
├─ Increment counter (distributed state)
├─ Compare against configured limits
├─ If exceeded: Return 429 + Retry-After
└─ Log rate limit event to Kafka
7. REQUEST TRANSFORMATION
├─ Add headers (X-User-ID, X-Tenant-ID)
├─ Rewrite path if needed
├─ Transform body (JSON ↔ XML)
└─ Add metadata headers
8. WORKFLOW TRIGGER CHECK
├─ Check if route matches any trigger
├─ If match: Send trigger event to NATS
├─ Attach X-Workflow-Execution-ID header
└─ Continue request processing (async)
9. OBSERVABILITY HOOKS
├─ Create OpenTelemetry span
├─ Set span attributes (tenant, user, path)
└─ Start request timer
10. LOAD BALANCING TO UPSTREAM SERVICE
├─ Lookup service by route
├─ Round-robin or least-connections
├─ Use connection pool (keep-alive)
└─ Set upstream service headers
3. Forward to Upstream Service
Headers: X-User-ID, X-Tenant-ID, X-Request-ID, X-Trace-ID

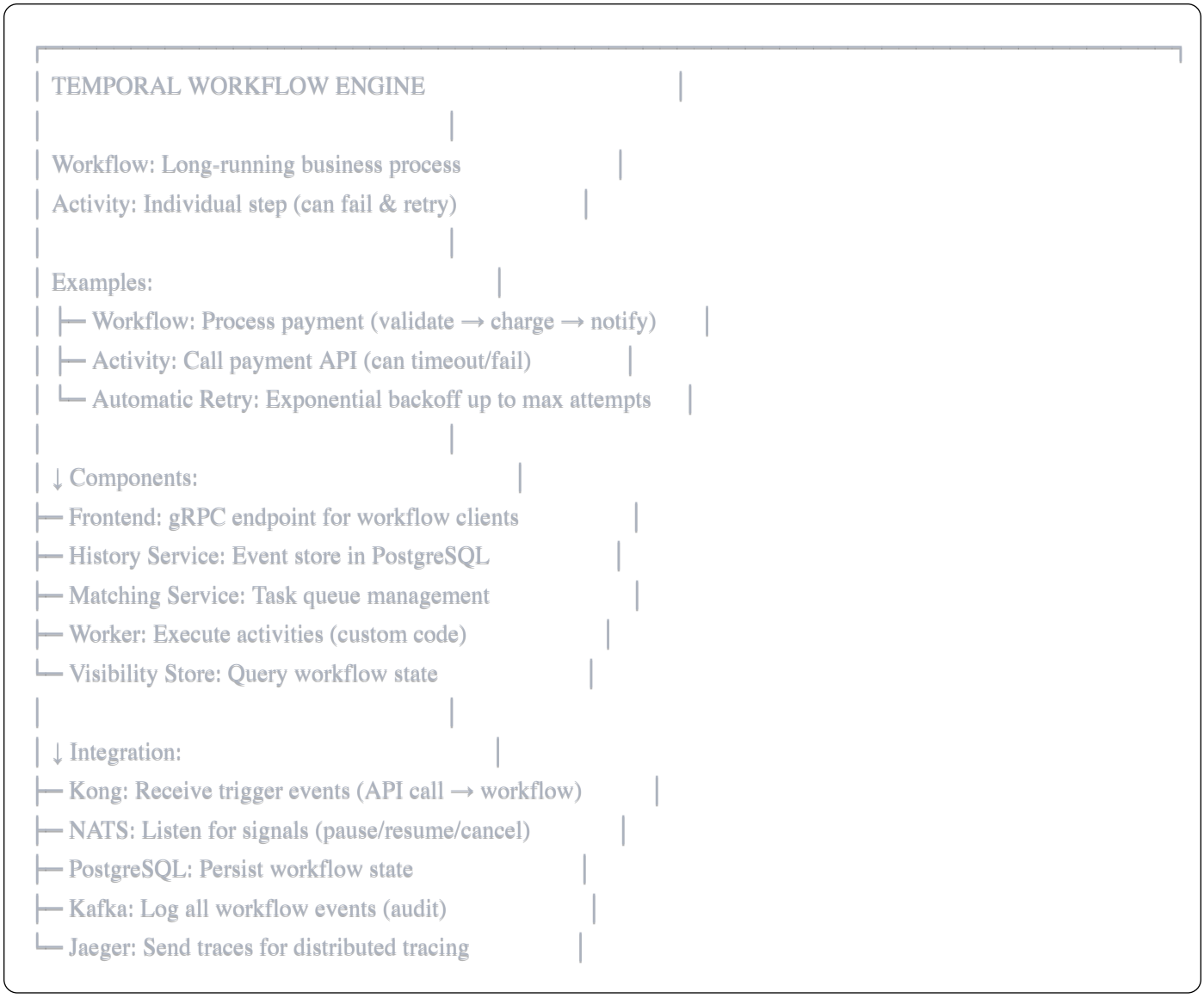

UPSTREAM SERVICE (Backend API)
- Process request
- Return response (200, 400, 500, etc.)
4. Response from Upstream
KONG DATA PLANE (Response Processing)
Response Processing Pipeline
1. Status Code Validation
Log response status
2. Response Transformation
Add security headers (HSTS, CSP, etc.)
Remove sensitive headers
Transform body if needed
Set cache-control headers
3. PII Masking
Detect & redact sensitive data
Log masked version to audit trail
4. Compression
gzip or brotli compress
Add Content-Encoding header
Reduce response size
5. Metrics Recording
Stop request timer
Calculate latency percentiles
Record status code distribution
Update error counters if needed
Export to Prometheus /metrics endpoint
6. Audit Logging (Async)
Prepare structured log entry
Include: timestamp, tenant, user,
path, method, status, latency
Send to Kafka topic (async, buffered)



2. ARCHITECTURE COMPONENT INTERACTIONS


KONG DATA PLANE (Stateless, Horizontal Scaling)
REQUEST → Authentication → Authorization → Rate Limiting →
Transformation → Logging → Upstream → Response
↓ Dependencies:
├─ Control Plane (Pull config every 30 sec)
├─ Vault (Fetch API keys/credentials, cache 1-5 min)
├─ Redis (Distributed rate limit state, TTL 1 min)
├─ Kafka (Send audit logs async)
├─ NATS (Receive config invalidation, send workflow triggers)
└─ Upstream Services (Route requests)
└─ Circuit breaker: Fail fast if service down
(State: Last failure timestamp, trip status)


KONG CONTROL PLANE
Admin API:
├─ POST /services (Create API service)
├─ POST /routes (Create API endpoint)
├─ POST /plugins (Configure plugins)
├─ GET /config (Return current state)
└─ DELETE / PATCH (Modify entities)
↓ Storage:
├─ PostgreSQL HA (Primary + 2-3 replicas)
├─ Synchronous replication (zero RPO)
└─ Automatic failover via Patroni
↓ Integration:
├─ Data Plane: Pull changes (pull-based, no push load)
├─ Vault: Store credentials, fetch as needed
├─ Kafka: Publish config change events
└─ NATS: Broadcast invalidation signals


TEMPORAL WORKFLOW ENGINE
Workflow: Long-running business process
Activity: Individual step (can fail & retry)
Examples:
├─ Workflow: Process payment (validate → charge → notify)
├─ Activity: Call payment API (can timeout/fail)
└─ Automatic Retry: Exponential backoff up to max attempts
↓ Components:
├─ Frontend: gRPC endpoint for workflow clients
├─ History Service: Event store in PostgreSQL
├─ Matching Service: Task queue management
├─ Worker: Execute activities (custom code)
└─ Visibility Store: Query workflow state
↓ Integration:
├─ Kong: Receive trigger events (API call → workflow)
├─ NATS: Listen for signals (pause/resume/cancel)
├─ PostgreSQL: Persist workflow state
├─ Kafka: Log all workflow events (audit)
└─ Jaeger: Send traces for distributed tracing


KAFKA EVENT BUS (Global, Multi-Region)
Topics (Immutable Log Structure):
├─ api.audit-trail (Compacted)
│ ├─ Producer: Kong data planes
│ ├─ Consumer: Analytics, Compliance archiver
│ ├─ Retention: 2 years (immutable audit log)
│ └─ Partition Key: tenant_id (co-locate tenant events)
├─ api.events (High-volume, workflow triggers)
│ ├─ Producer: Kong plugins, 3rd-party adapters
│ ├─ Consumer: Temporal trigger consumer, Alerting
│ ├─ Retention: 7 days
│ └─ Partition Key: tenant_id, event_type
├─ third-party.interactions (3rd-party API calls)
│ ├─ Producer: Kong outbound proxy plugin
│ ├─ Consumer: Analytics, Monitoring
│ ├─ Retention: 30 days
│ └─ Partition Key: tenant_id, service_name
└─ api.config-changes (Control plane events)
├─ Producer: Kong control plane
├─ Consumer: Data planes (pull trigger), Audit
├─ Retention: 30 days
└─ Partition Key: tenant_id
↓ Replication:
└─ Multi-region: Replication factor 2-3 across regions
└─ Failover: Broker loss tolerance = replication - 1

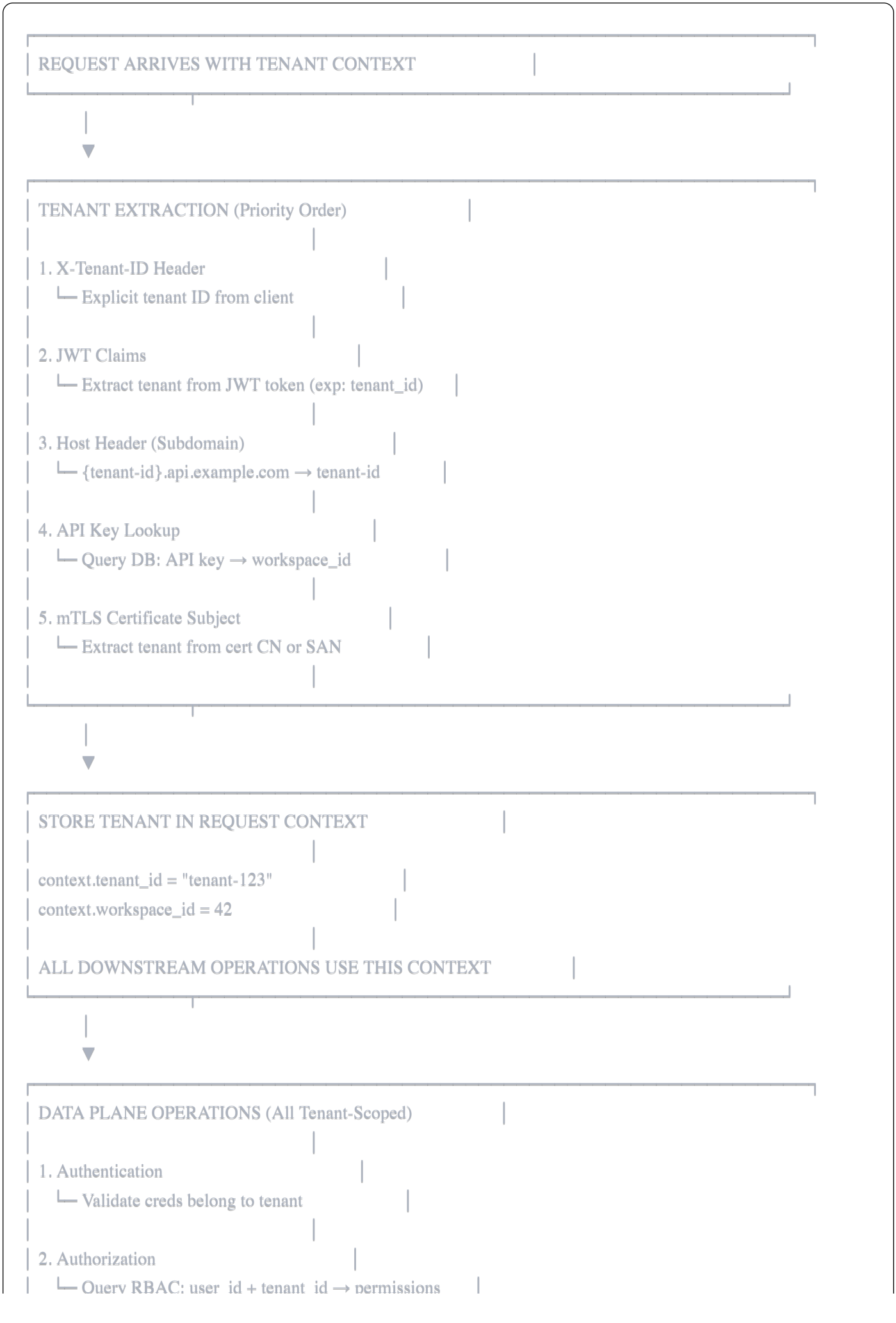

NATS JETSTREAM (Local Per-Region, Low-Latency)
Subjects (Stream Subjects):
├─ workflow.triggers.{tenant_id}
│ ├─ Producer: Kong when API matches trigger rule
│ ├─ Consumer: Temporal (subscribe to trigger events)
│ └─ Latency: Sub-millisecond
├─ workflow.signals.{tenant_id}
│ ├─ Producer: External systems, APIs
│ ├─ Consumer: Temporal (send signal to workflow)
│ └─ Use case: Approve/reject workflow manually
├─ config.invalidations
│ ├─ Producer: Kong control plane (config changed)
│ ├─ Consumer: All Kong data planes (broadcast)
│ ├─ Action: Purge local cache, fetch fresh config
│ └─ Delivery: Exactly-once guarantees
└─ health.heartbeats
├─ Producer: All services (every 30 sec)
├─ Consumer: Monitoring/alerting system
└─ Action: Detect service failures in <10 sec
↓ Multi-Tenancy:
└─ Subjects scoped per tenant → No cross-tenant leakage
├─ Subjects: workflow.triggers.tenant-123
├─ Authorization: JWT-based per-tenant
└─ Isolation: Complete (no way to access other tenants)


VAULT (Secrets Management)
Secret Types:
├─ API Keys (for 3rd-party services)
│ ├─ Path: secret/data/3rdparty/{service}/{api_key}
│ ├─ Rotation: Every 30-90 days
│ ├─ Accessed by: Kong plugins (via HTTP API)
│ └─ Cache: 5 minutes in-memory
├─ OAuth Secrets
│ ├─ Path: secret/data/oauth/{provider}/{client_secret}
│ └─ Accessed by: OAuth2 plugin
├─ mTLS Certificates
│ ├─ Path: secret/data/certs/{service}/{cert_pem}
│ ├─ Rotation: Automatic via cert-manager (1 week before)
│ └─ Cached: In Kong worker memory
└─ Database Credentials
├─ Path: secret/data/db/{environment}/{role}
└─ Rotation: Every 30 days
↓ High Availability:
├─ 5-node HA cluster (Raft consensus)
├─ Automatic failover
├─ Encrypted storage (KMS + encryption at rest)
└─ Backup: Daily snapshots to S3 (encrypted)
↓ Audit:
└─ All secret access logged with: who, what, when, result


POSTGRESQL (Operational State Store)
Tables:
├─ workspaces (Tenant definitions)
│ ├─ id, name, description, created_at
│ └─ Used for multi-tenancy isolation
├─ rbac_roles (Permission definitions)
│ ├─ admin, operator, developer, viewer, custom
│ └─ Assigned to users per workspace
├─ services (API backend definitions)
│ ├─ name, url, protocol, health_check_url
│ ├─ Partitioned by workspace_id
│ └─ Referenced by routes
├─ routes (API endpoints exposed to clients)
│ ├─ name, service_id, path, methods, hosts
│ ├─ upstream_url, transformations
│ ├─ Indexed by: workspace_id, path, method
│ └─ Returns by GET /api/routes?workspace=X
├─ plugins (Functionality extensions)
│ ├─ name (auth, ratelimit, transform, etc.)
│ ├─ service_id or route_id (applies to which entity)
│ ├─ config (JSON) - plugin-specific settings
│ └─ enabled (boolean)
├─ rate_limiting_policies (Traffic control)
│ ├─ workspace_id, route_id, user_id (optional)
│ ├─ limit (e.g., 1000 requests)
│ ├─ window (e.g., 1 hour)
│ ├─ algorithm (sliding window, token bucket)
│ └─ Cached in Redis for performance
├─ audit_logs (Immutable change log)
│ ├─ timestamp, user_id, action, entity_type, entity_id
│ ├─ before, after (JSON snapshots)
│ ├─ Partitioned by (workspace_id, date)
│ ├─ Immutable: INSERT only, no DELETE
│ └─ TTL-based purging (default 2 years)

---

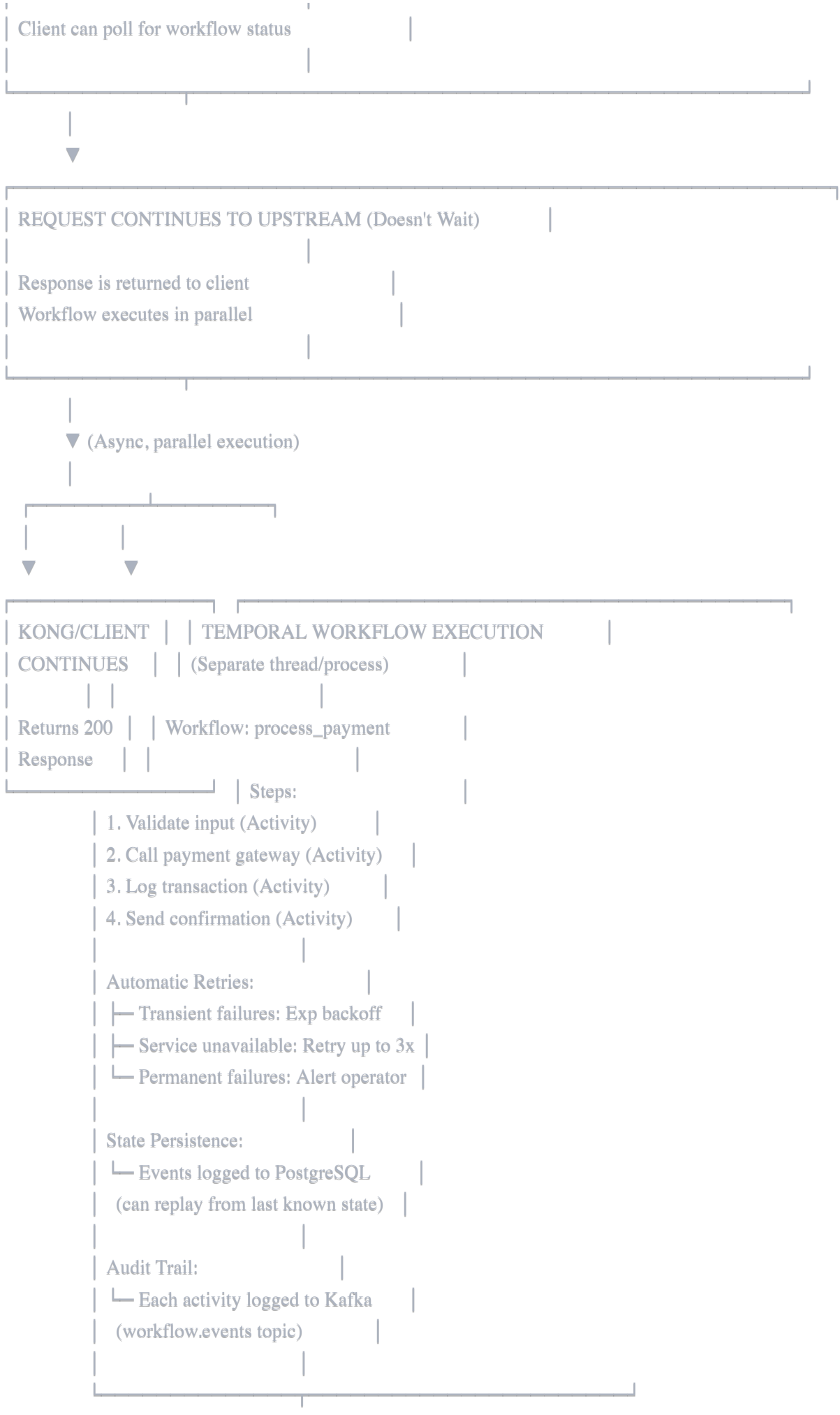
3. MULTI-TENANT ISOLATION FLOW

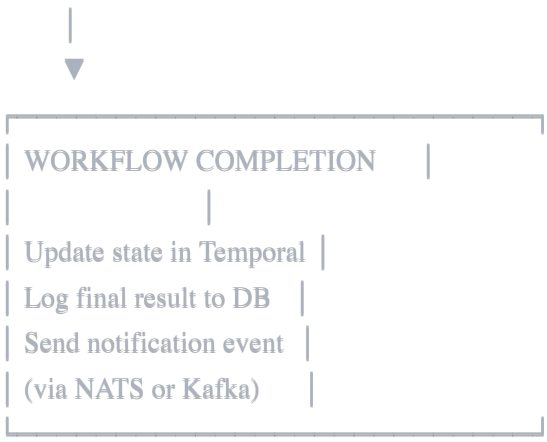

REQUEST ARRIVES WITH TENANT CONTEXT
TENANT EXTRACTION (Priority Order)
1. X-Tenant-ID Header
Explicit tenant ID from client
2. JWT Claims
Extract tenant from JWT token (exp: tenant_id)
3. Host Header (Subdomain)
{tenant-id}.api.example.com → tenant-id
4. API Key Lookup
Query DB: API key → workspace_id
5. mTLS Certificate Subject
Extract tenant from cert CN or SAN
STORE TENANT IN REQUEST CONTEXT
context.tenant_id = "tenant-123"
context.workspace_id = 42
ALL DOWNSTREAM OPERATIONS USE THIS CONTEXT
DATA PLANE OPERATIONS (All Tenant-Scoped)
1. Authentication
Validate creds belong to tenant
2. Authorization
Query RBAC: user_id + tenant_id → permissions

---

4. WORKFLOW INTEGRATION FLOW


API REQUEST ARRIVES AT KONG
WORKFLOW TRIGGER PLUGIN (In Kong Chain)
if route.id matches workflow_trigger rule:
{
workflow_name: "process_payment"
tenant_id: extract_from_request()
input: {
user_id: request.user_id,
amount: request.body.amount,
currency: request.body.currency
}
}
TRIGGER EVENT PUBLISHED TO NATS
(Async, non-blocking)
Subject: workflow.triggers.tenant-123
Message: {
workflow_name: "process_payment",
workflow_id: "uuid-generated",
tenant_id: "tenant-123",
input: {...},
api_call_id: "request-id-from-header"
}
ADD TO RESPONSE HEADERS
X-Workflow-Execution-ID: uuid-generated
X-Workflow-Status-Endpoint: /workflow/status/{uuid}

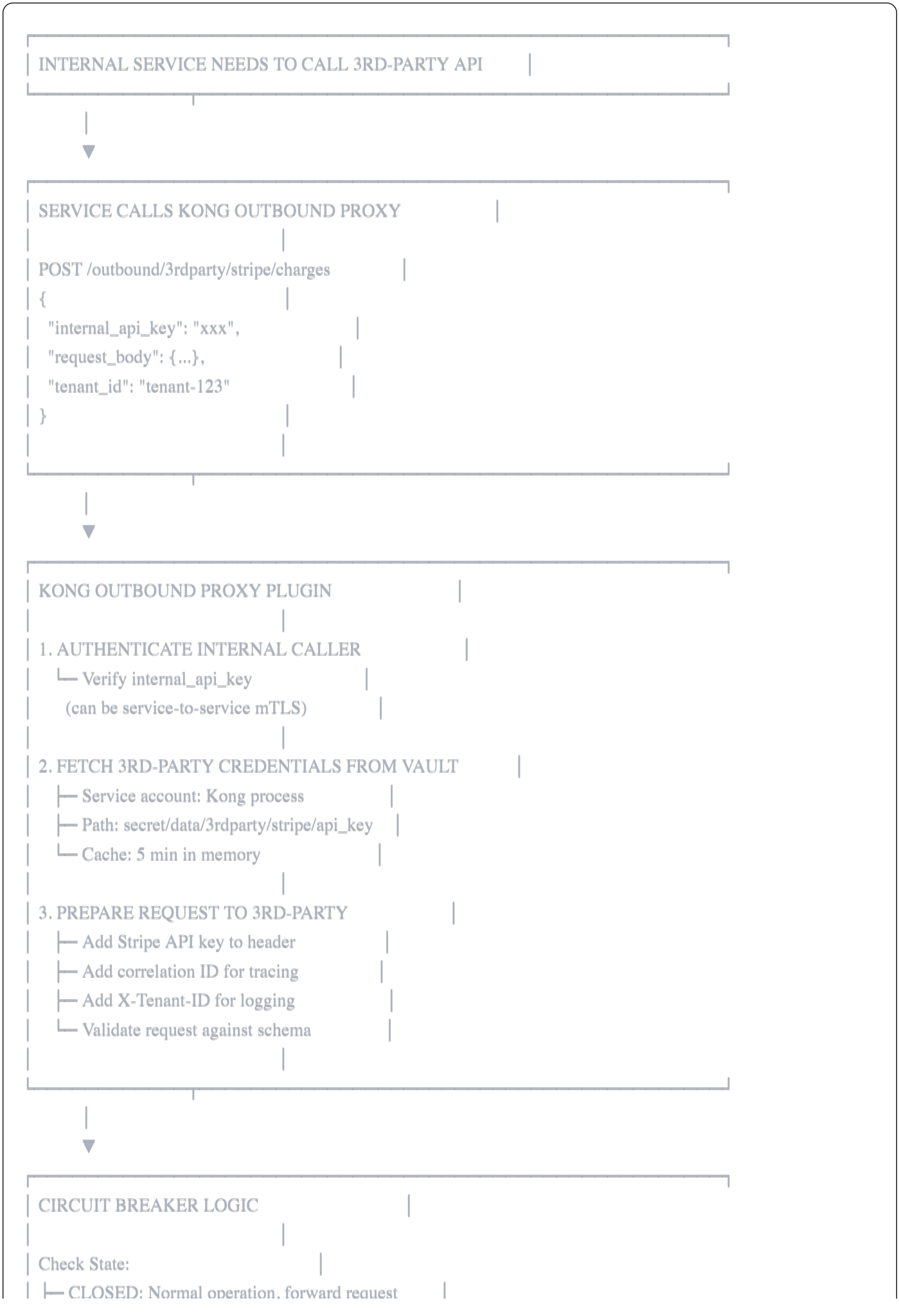

Client can poll for workflow status
REQUEST CONTINUES TO UPSTREAM (Doesn't Wait)
Response is returned to client
Workflow executes in parallel
(Async, parallel execution)
KONG/CLIENT CONTINUES
Returns 200 Response
TEMPORAL WORKFLOW EXECUTION
(Separate thread/process)
Workflow: process_payment
Steps:
1. Validate input (Activity)
2. Call payment gateway (Activity)
3. Log transaction (Activity)
4. Send confirmation (Activity)
Automatic Retries:
├─ Transient failures: Exp backoff
├─ Service unavailable: Retry up to 3x
└─ Permanent failures: Alert operator
State Persistence:
└─ Events logged to PostgreSQL
(can replay from last known state)
Audit Trail:
└─ Each activity logged to Kafka
(workflow.events topic)



CLIENT CAN QUERY WORKFLOW STATUS:
├─ GET /api/workflows/{execution_id}
├─ Returns: current_step, status, progress, last_update
└─ Webhook callback available (when workflow completes)

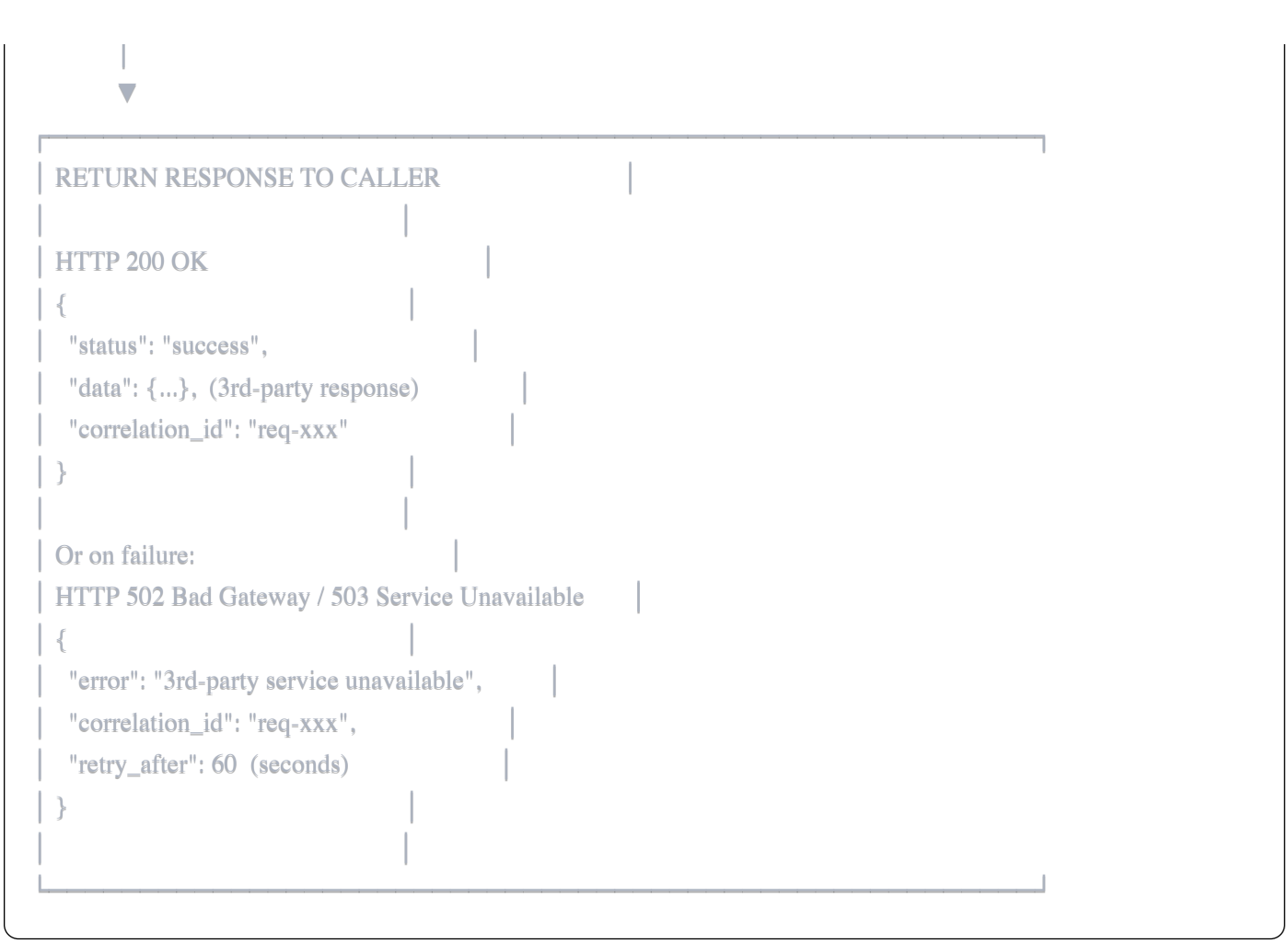
---

5. THIRD-PARTY API INTEGRATION FLOW


INTERNAL SERVICE NEEDS TO CALL 3RD-PARTY API
SERVICE CALLS KONG OUTBOUND PROXY
POST /outbound/3rdparty/stripe/charges
{
"internal_api_key": "xxx",
"request_body": {...},
"tenant_id": "tenant-123"
}
KONG OUTBOUND PROXY PLUGIN
1. AUTHENTICATE INTERNAL CALLER
└─ Verify internal_api_key
(can be service-to-service mTLS)
2. FETCH 3RD-PARTY CREDENTIALS FROM VAULT
├─ Service account: Kong process
├─ Path: secret/data/3rdparty/stripe/api_key
└─ Cache: 5 min in memory
3. PREPARE REQUEST TO 3RD-PARTY
├─ Add Stripe API key to header
├─ Add correlation ID for tracing
├─ Add X-Tenant-ID for logging
└─ Validate request against schema
CIRCUIT BREAKER LOGIC
Check State:
├─ CLOSED: Normal operation, forward request


├─ OPEN: Service down, return cached error
└─ HALF_OPEN: Testing if recovered
Thresholds:
├─ Failure rate > 50% for 30 sec → OPEN
├─ Timeout > 5 sec (3 consecutive) → OPEN
└─ Recovery test every 60 sec → HALF_OPEN
▼ (If not open)
CALL 3RD-PARTY SERVICE
HTTP POST https://api.stripe.com/v1/charges
Headers:
Authorization: Bearer {api_key_from_vault}
X-Correlation-ID: {from_request}
X-Tenant-ID: tenant-123
X-Request-Timeout: 5s
With Timeout:
└─ 5 second timeout (fail fast)
▼
RESPONSE HANDLING
2xx (Success):
├─ Update circuit breaker (success)
├─ Cache successful response (if applicable)
└─ Log to Kafka topic: third-party.interactions
4xx (Client Error):
├─ Return error to caller (bad request)
├─ Log for debugging (alert if suspicious)
└─ Circuit breaker: Don't penalize (client issue)
5xx (Service Error):
├─ Increment failure counter
├─ Check circuit breaker state
├─ If failures > threshold → OPEN circuit


├─ Retry logic (with exponential backoff)
├─ Retry 1: Wait 100ms
├─ Retry 2: Wait 200ms
└─ Retry 3: Wait 400ms
└─ If all retries fail → Return error
Timeout:
├─ Treat as service error
├─ Increment failure counter
├─ Retry with backoff
└─ Circuit breaker may open if too many
LOG TO KAFKA (Async)
Topic: third-party.interactions
Partition Key: tenant_id + service_name
Message:
{
timestamp: ISO8601,
tenant_id: "tenant-123",
service: "stripe",
method: "POST",
endpoint: "/v1/charges",
request_id: "req-xxx",
status_code: 200,
latency_ms: 234,
request_size: 512,
response_size: 2048,
error_code: null,
circuit_breaker_state: "CLOSED"
}
Consumers:
├─ Analytics: Aggregate metrics per service
├─ Monitoring: Alert on high failure rates
├─ Billing: Track API usage for chargeback
└─ Audit: Immutable record for compliance



---

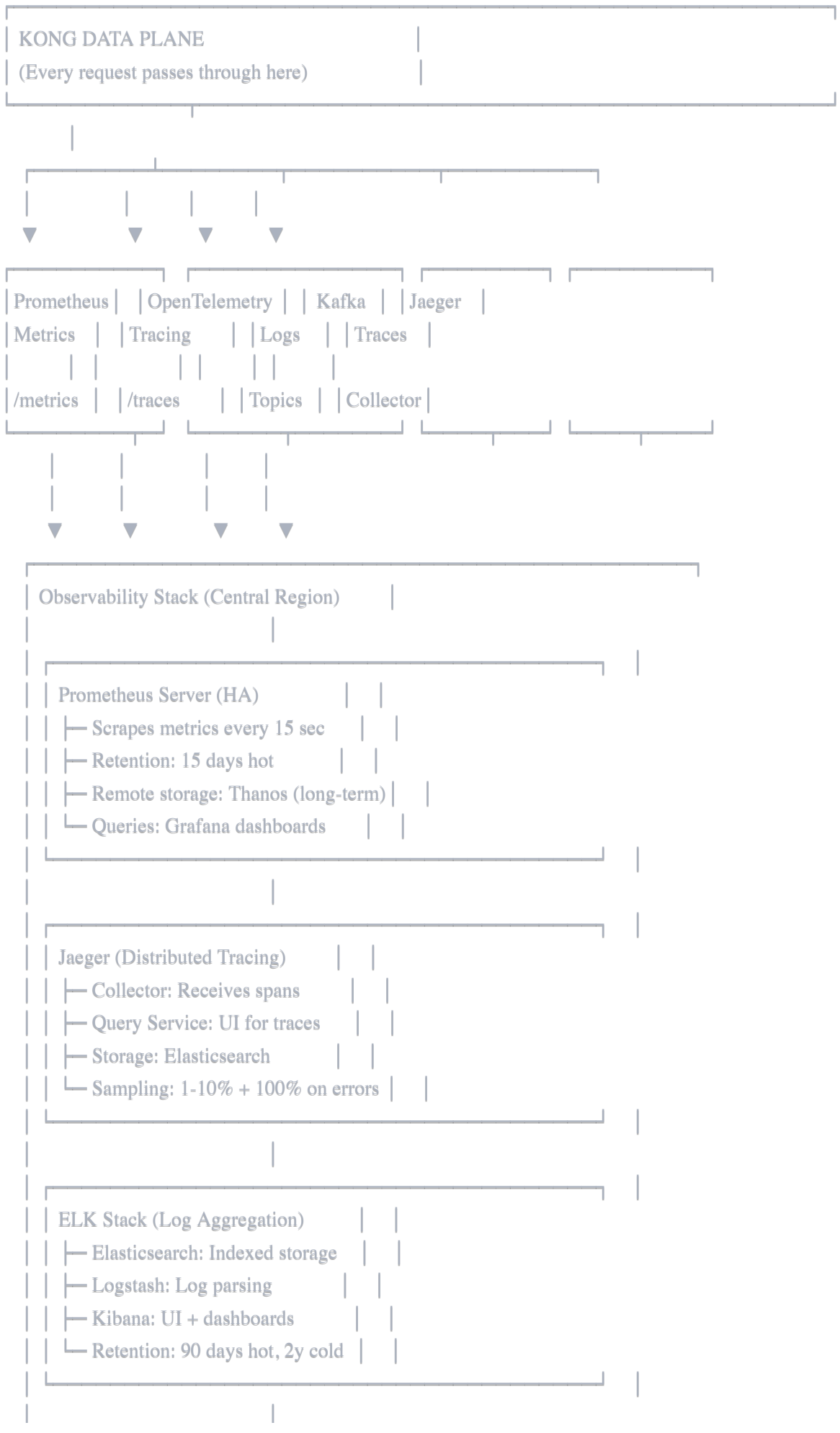
6. SCALABILITY FLOW (Auto-Scaling in Action)

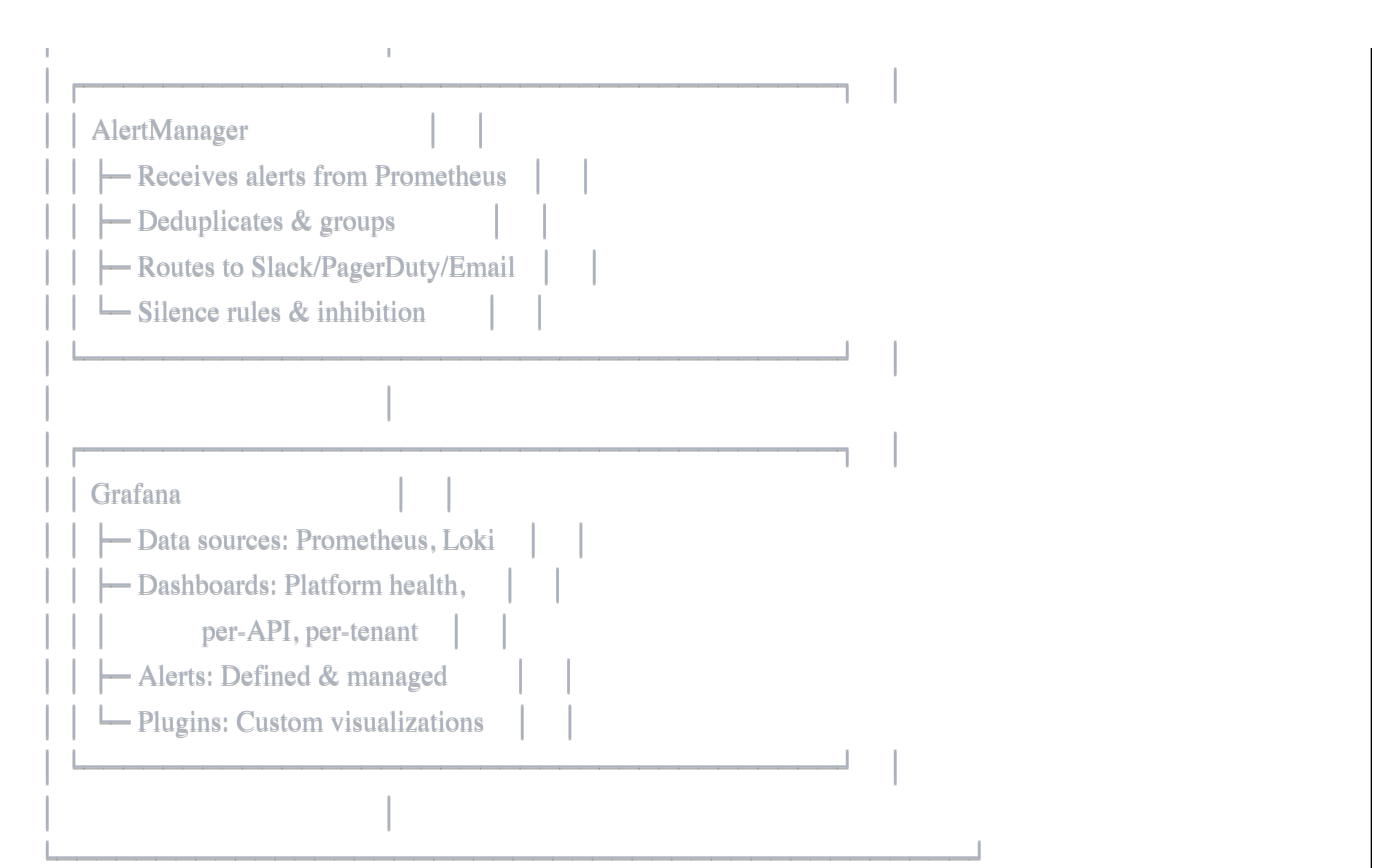
```
TIME: 10:00 AM - BASELINE LOAD
│
├─ Kong Data Plane: 3 nodes (1 per AZ)
├─ CPU usage: ~40% per node
├─ Memory: ~200MB per node
└─ Status: Healthy

TIME: 10:15 AM - TRAFFIC SPIKE DETECTED
│
├─ Requests/sec jump from 5K to 12K
├─ CPU usage rises to 75% on all nodes
├─ Memory usage increases to ~400MB
│
└─ HPA (Horizontal Pod Autoscaler) triggered:
   ├─ Threshold: CPU > 70% for 30 sec
   └─ Action: Scale from 3 → 6 replicas

TIME: 10:18 AM - KUBERNETES SCALING
│
├─ Cluster Autoscaler checks available capacity
│
├─ Scenario 1: Capacity available on existing nodes
│  └─ Deploy pods to free slots (1-2 sec)
│
└─ Scenario 2: No available capacity
   └─ Trigger cluster autoscaler:
      ├─ Request new worker node from cloud
      ├─ AWS/GCP: Provision takes 1-2 minutes
      └─ KEDA: Scale based on Kafka lag (if applicable)

TIME: 10:20 AM - STEADY STATE AT HIGH LOAD
│
├─ Kong Data Plane: 6 nodes (+ extras may be provisioning)
├─ CPU usage: ~45% per node (balanced across 6)
├─ Memory: ~250MB per node
├─ Requests distributed evenly (load balancer)
└─ Status: Healthy, fully scaled

TIME: 10:35 AM - TRAFFIC SUBSIDES
│
├─ Requests/sec drops back to 6K
├─ CPU usage: ~35% per node
├─ HPA scale-down: Enabled after 5 min of low CPU
│
```

```
└─ Action: Scale from 6 → 4 replicas
   ├─ Wait for graceful termination (30 sec drain)
   ├─ Existing requests complete
   └─ Pods drain connections

TIME: 10:40 AM - BACK TO BASELINE
│
├─ Kong Data Plane: 4 nodes (consolidating)
├─ Cluster Autoscaler: May remove excess node (5 min delay)
├─ CPU usage: ~40% per node
└─ Status: Normal

SCALING METRICS:
├─ CPU Utilization: 40-80% target range
├─ Memory Usage: 200-500MB per node
├─ Kafka Consumer Lag: <10 sec (for event processing)
├─ Request Latency (p99): <500ms
└─ Error Rate: <0.1%

AUTOSCALING CONFIGURATION:
│
├─ HPA Min Replicas: 3 (HA requirement)
├─ HPA Max Replicas: 50 (cost control)
├─ CPU Threshold: 70%
├─ Memory Threshold: 80%
├─ Scale-up delay: 30 sec
├─ Scale-down delay: 5 min
├─ KEDA enabled: YES (for Kafka lag-based scaling)
└─ Graceful termination period: 30 sec
```

---

7. OBSERVABILITY INTEGRATION MAP


KONG DATA PLANE
(Every request passes through here)
Prometheus
Metrics
/metrics
OpenTelemetry
Tracing
/traces
Kafka
Logs
Topics
Jaeger
Traces
Collector
Observability Stack (Central Region)
Prometheus Server (HA)
├─ Scrapes metrics every 15 sec
├─ Retention: 15 days hot
├─ Remote storage: Thanos (long-term)
└─ Queries: Grafana dashboards
Jaeger (Distributed Tracing)
├─ Collector: Receives spans
├─ Query Service: UI for traces
├─ Storage: Elasticsearch
└─ Sampling: 1-10% + 100% on errors
ELK Stack (Log Aggregation)
├─ Elasticsearch: Indexed storage
├─ Logstash: Log parsing
├─ Kibana: UI + dashboards
└─ Retention: 90 days hot, 2y cold



METRIC EXAMPLES (Exported by Kong):

```
├─ http_requests_total{method="POST", path="/api/users", status="200", tenant="tenant-123"}
├─ http_request_duration_seconds{quantile="0.99", path="/api/users"}
├─ kong_upstream_target_health{upstream="backend-service", target="10.0.1.5:8080", state="up"}
├─ rate_limit_check_duration_seconds (custom)
└─ auth_token_validation_duration_seconds (custom)
```

TRACE EXAMPLES (Sent to Jaeger):

```
├─ Trace: 4a2c461b1d6a3b5e
│  ├─ Span 1: Kong Request Processing (10ms)
│  │  ├─ Event: TLS Handshake (2ms)
│  │  ├─ Event: Auth Plugin (1ms)
│  │  ├─ Event: Rate Limit Check (0.5ms)
│  │  └─ Event: Route Resolution (0.1ms)
│  │
│  ├─ Span 2: Upstream Service Call (45ms)
│  │  ├─ Event: DNS Lookup (1ms) [if needed]
│  │  ├─ Event: Connect (2ms)
│  │  └─ Event: Request/Response (42ms)
│  │
│  └─ Span 3: Kong Response Processing (5ms)
│     ├─ Event: Transformation (2ms)
│     └─ Event: Logging (3ms)
│
│  Total Trace: 60ms (visible in Jaeger UI)
```

```
LOG EXAMPLES (Sent to ELK):
├─ {timestamp: "2025-10-22T14:30:15Z", tenant: "tenant-123", user: "user-456", method: "POST", path: "/api/payments", status: 200, latency_ms: 234, request_id: "req-abc"}
├─ {timestamp: "2025-10-22T14:30:20Z", tenant: "tenant-123", error: "Rate limit exceeded", limit: 1000, window: "1h"}
└─ {timestamp: "2025-10-22T14:30:25Z", tenant: "tenant-123", auth_error: "Invalid API key"}

ALERT EXAMPLES (Triggered from Prometheus):
├─ CRITICAL: Kong Data Plane Down
│  └─ Condition: No nodes responding for 1+ min
│
├─ WARNING: High Error Rate
│  └─ Condition: Error rate > 1% for 5 min
│
├─ CRITICAL: P99 Latency High
│  └─ Condition: p99 latency > 500ms for 5 min
│
├─ WARNING: Kafka Lag High
│  └─ Condition: Consumer lag > 10K messages
│
└─ INFO: Scaling Event
   └─ Triggered: 6 nodes (up from 3)
```

---

8. FAILOVER & DISASTER RECOVERY FLOW

SCENARIO 1: SINGLE DATA PLANE NODE FAILURE

```
│
├─ 10:15 AM: Node crashes (OOM, kernel panic, etc.)
│
├─ Detection (K8s)
│  └─ Liveness probe fails (3 consecutive failures)
│     └─ Latency: ~30 seconds
│
├─ Action (K8s)
│  ├─ Mark pod as "Not Ready"
│  ├─ Load balancer removes from target group
│  ├─ Pod evicted
│  └─ Replica set spawns new pod on different node
│
├─ Recovery
│  ├─ New pod starts (~10 sec)
│  ├─ Rejoins load balancer (~5 sec)
│  ├─ RTO: ~45 seconds total
│  └─ Data loss: ZERO (stateless data plane)
│
└─ Impact: Minimal (traffic distributed to other 4 nodes)
```

SCENARIO 2: ENTIRE KUBERNETES CLUSTER FAILURE (Regional)

```
│
├─ 10:15 AM: Cluster networking failure
│  └─ etcd cluster becomes unavailable
│
├─ Detection
│  ├─ Multiple services report high latency
│  ├─ Prometheus scrapers timeout
│  └─ Alerting: Critical alert fires
│
├─ Action (Manual or Automatic)
│  ├─ Automatic: DNS failover to secondary region
│  ├─ Manual: Update Route 53 to secondary region
│  └─ Latency: 30-60 sec (TTL propagation)
│
├─ Secondary Region Activation
│  ├─ Kong data planes in secondary region already running
│  ├─ PostgreSQL: Promote secondary replica to primary
│  │  ├─ Current replication lag: <100ms
│  │  ├─ Data loss: Zero to minimal
│  │  └─ Promotion time: ~30 sec
```

```
│ │
│ ├─ Temporal: Workflow state in replicated DB
│ │  └─ Workflows resume from last persisted state
│ │
│ └─ Kafka: Multi-region replication (RF=2)
│    └─ Secondary cluster already has events
│
├─ Recovery Steps
│ ├─ 1. Detect primary region unavailable (2 min)
│ ├─ 2. DNS failover to secondary (1 min)
│ ├─ 3. Promote secondary DB (1 min)
│ ├─ 4. Validate secondary region health (1 min)
│ └─ Total RTO: ~5 minutes
│
├─ Data Consistency
│ ├─ In-flight requests: May be lost (acceptable)
│ ├─ Persisted state: Recovered from replicas
│ ├─ Workflows: Replay from last committed state
│ └─ RPO: <5 minutes (sync replication to secondary)
│
└─ Post-Recovery
  ├─ Investigate root cause (human)
  ├─ Restore primary region (if applicable)
  └─ Failback to primary (planned)


SCENARIO 3: POSTGRESQL PRIMARY FAILURE
│
├─ 10:15 AM: Primary DB becomes unavailable
│
├─ Detection (Patroni)
│ ├─ Replica loses connection to primary
│ ├─ Quorum broken (can't reach primary)
│ └─ Latency: ~10 seconds
│
├─ Automatic Failover (Patroni)
│ ├─ Quorum elects best replica as new primary
│ ├─ Promotion: Secondary → Primary (with standby)
│ ├─ TTL: <30 seconds
│ └─ Write operations resume on new primary
│
├─ Impact
│ ├─ Read operations: Minimal impact (secondary already handling)
│ ├─ Write operations: Brief pause (<30 sec)
```

```
│  ├─ In-flight transactions: May roll back
│  └─ Data loss: Zero (synchronous replication)
│
└─ Post-Failover
   ├─ Verify replication to remaining secondaries
   ├─ Spin up new standby (from backup or rebuild)
   └─ Update Route 53 (already updated by Patroni)


SCENARIO 4: KAFKA BROKER FAILURE
│
├─ 10:15 AM: Kafka broker 2/3 fails
│
├─ Kafka Cluster State
│  ├─ Cluster: 3 brokers (RF=3 by default)
│  ├─ Leader election: New broker elected
│  ├─ Replication: Continues on healthy brokers
│  └─ Latency: ~5 seconds (Zookeeper/KRaft election)
│
├─ Impact
│  ├─ Audit logs: Continue writing to other brokers
│  ├─ Event stream: No data loss (RF=3)
│  ├─ Producers: May see increased latency
│  └─ Consumers: No impact
│
├─ Recovery
│  ├─ Identify failed broker
│  ├─ Replace/restart broker
│  ├─ Wait for replication catch-up (5-30 min depending on backlog)
│  └─ RTO: <5 min to regain full capacity
│
└─ Data Protection
   └─ Backups: Daily snapshots to S3 (cross-region)
      └─ Can restore entire cluster if needed


DISASTER RECOVERY PROCEDURES:
│
├─ Backup Strategy
│  ├─ PostgreSQL: Daily snapshot to S3 (cross-region)
│  ├─ Kafka: Replication RF=2-3 (no explicit backups needed)
│  ├─ Vault: Daily snapshots encrypted to S3
│  └─ Recovery: Can restore to any point in time
│
└─ RPO (Recovery Point Objective)
```

```
├─ RPO (Recovery Point Objective)
│  ├─ PostgreSQL: <5 min (sync replication)
│  ├─ Kafka: Real-time (3 replicas)
│  ├─ Configuration: <30 sec (config caching)
│  └─ Audit logs: <1 sec (buffered writes)
│
├─ RTO (Recovery Time Objective)
│  ├─ Data plane failure: <1 min (auto-recovery)
│  ├─ Regional failure: <5 min (DNS + promotion)
│  ├─ Database failure: <1 min (Patroni promotion)
│  └─ Full cluster rebuild: <30 min (from backups)
│
└─ Testing & Validation
   ├─ Quarterly: Run full DR simulation
   ├─ Monthly: Test individual component failure
   ├─ After deploy: Chaos engineering tests
   └─ Monthly: Backup restoration test
```

End of Architecture Diagrams Document

This document provides visual flows and detailed component interactions to complement the main architectural documentation.