Contents

RADIANT Multi-Protocol AI Interface Architecture v3.0

Executive Summary

This document defines the production architecture for RADIANT's unified, bidirectional protocol layer supporting MCP, A2A, OpenAI, Anthropic, and Google interfaces at **1M+ concurrent connection scale**.

v3 Changes from v2:

Component	v2 (Rejected)	v3 (Final)
Connection Tier	Envoy + Lua filters	Custom Go Gateway
Message Bus	Redis Pub/Sub	NATS JetStream
Authorization	Tenant-scoped Cedar	Resource-level ABAC Cedar
Connection Draining	30s GOAWAY window	Resume Token + Session Rehydration

Critical Insight: Envoy is a *proxy*, not a message bus client. Bridging 1M WebSockets to NATS subjects requires the Gateway to be a first-class NATS client — impossible with Envoy without complex Wasm filters.

Part 1: Protocol Adapter Layer

Key elements retained from v2: - JSON Schema as canonical internal format - Dual-level versioning (protocol + provider) - `SecurityContext` injection into all adapter methods - Bidirectional support (RADIANT as server AND client)

Part 2: Go Connectivity Gateway

2.1 Architecture Overview

The Go Gateway is a lightweight, purpose-built service that:

1. Terminates TLS (mTLS for A2A, standard TLS for users)
2. Holds persistent WebSocket/SSE connections
3. Acts as a first-class NATS JetStream client
4. Bridges sockets NATS subjects bidirectionally

```
                                    INTERNET
```

```
                          NETWORK LOAD BALANCER (NLB)
```

- Layer 4 TCP passthrough
- Cross-zone load balancing
- Health checks on :8080/health
- TLS termination DISABLED (handled by Gateway)

```
                         GO CONNECTIVITY GATEWAY FLEET


   Gateway 1       Gateway 2       Gateway 3       Gateway N
   (c6g.xl)        (c6g.xl)        (c6g.xl)        (c6g.xl)
   ~80K conn       ~80K conn       ~80K conn       ~80K conn


Memory per connection: ~10KB (socket + NATS sub + session state)
Target: 50K-100K connections per c6g.xlarge (8GB RAM)

Responsibilities:
  TLS termination (mTLS cert validation, OIDC JWT extraction)
  WebSocket/SSE upgrade and frame parsing
  Protocol detection (MCP JSON-RPC, A2A, raw HTTP)
  NATS client: publish to inbox, subscribe to outbox
```

```
  Session state: connection_id   session_id mapping
  Resume token generation and validation

DOES NOT: Execute business logic, call databases, make LLM calls
```

```
                        NATS JETSTREAM CLUSTER

Streams:
  INBOX: Incoming messages from connections
    Subjects: in.{protocol}.{tenant_id}.{agent_id}
    Retention: WorkQueue (consumed once)
    Replicas: 3

  OUTBOX: Responses back to connections
    Subjects: out.{session_id}
    Retention: Limits (1 hour TTL, 1000 msgs max)
    Replicas: 3

Why NATS over Redis:
  At-Least-Once delivery (Redis Pub/Sub is At-Most-Once)
  Subject-based addressing (out.{session_id} for return path)
  Persistence survives Lambda cold starts and Gateway rebalancing
  Consumer groups for Lambda worker scaling
```

```
                        ADAPTER LOGIC TIER (Lambda)

NATS Consumer Groups:
  mcp-workers: Subscribe to in.mcp.>
  a2a-workers: Subscribe to in.a2a.>
  fc-workers: Subscribe to in.fc.>

Flow:
1. Receive message from NATS (includes SecurityContext)
2. Validate authorization via Cedar
3. Execute tool / call provider
4. Publish response to out.{session_id}
```

2.2 Key Interfaces

SessionContext - Contains all state for a connected client: - `SessionID` - Unique session identifier (survives reconnects) - `ConnectionID` - Current connection identifier (changes on

reconnect) - `TenantID` - Tenant isolation - `PrincipalID/Type` - Agent, User, or Service - `AuthType` - mTLS, OIDC, or APIKey - `Protocol` - MCP, A2A, OpenAI, Anthropic, Google - `InboxSubject/OutboxSubject` - NATS subjects for this session - `ResumeToken/Expiry` - For session rehydration

GatewayStats - Metrics exposed at `/health`: - `ActiveConnections` / `TotalConnections` - `MessagesIn` / `MessagesOut` - `ResumedConnections` / `AuthFailures`

Part 3: NATS JetStream Configuration

3.1 Stream Definitions

```yaml
# INBOX Stream - messages from clients to Lambda workers
name: INBOX
subjects: ["in.>"]  # in.{protocol}.{tenant_id}.{agent_id}
retention: workqueue  # Consumed exactly once
maxAge: 1h
replicas: 3

# OUTBOX Stream - responses back to specific sessions
name: OUTBOX
subjects: ["out.>"]  # out.{session_id}
retention: limits
maxAge: 1h  # Messages expire after 1 hour
replicas: 3

# HISTORY Stream - for session resume replay
name: HISTORY
subjects: ["history.>"]  # history.{session_id}
retention: limits
maxMsgsPerSubject: 10000
maxAge: 1h
replicas: 3
```

3.2 Consumer Definitions

Consumer	Stream	Filter	Ack Wait	Purpose
mcp-workers	INBOX	in.mcp.>	30s	MCP protocol handling
a2a-workers	INBOX	in.a2a.>	60s	A2A tasks (longer)
fc-workers	INBOX	in.fc.>	120s	LLM function calls

Part 4: Resource-Level Cedar Authorization

4.1 Cedar Schema

```
namespace Radiant {
  entity Agent in [Tenant] {
    tier: String, scopes: Set<String>, labels: Set<String>
  };
  entity User in [Tenant] {
    role: String, scopes: Set<String>, department: String
  };
  entity Tool {
    name: String, destructive: Bool, sensitive: Bool,
    requiredScopes: Set<String>, labels: Set<String>,
    namespace: String, owner: String
  };

  action "tool:call" appliesTo {
    principal: [Agent, User, Service],
    resource: [Tool],
    context: { tenantId: String, sourceProtocol: String }
  };
}
```

4.2 Key Policies

Policy	Purpose
`deny-cross-tenant`	FORBID all cross-tenant access
`user-tool-call-non-sensitive`	Allow non-destructive tools with scopes
`agent-tool-call-namespace`	Agents can call tools in their namespace
`admin-tool-call-all`	Admins bypass restrictions
`deny-sensitive-without-label`	Protect sensitive resources

Part 5: Resume Token Strategy

5.1 Session Lifecycle

```
Connect → Active → Drain → Resume → Active (same session)
```

```
NATS Subject: out.{session_id}
```

- Created when session starts
- Persists across connection changes
- Lambda publishes here regardless of Gateway node
- New Gateway subscribes on resume

KEY INSIGHT: Session durability over connection durability. The TCP socket can die and reconnect; the session (and NATS subject) remains stable.

5.2 Resume Token Contents

```
type ResumeTokenData struct {
  SessionID       string    // Survives reconnect
  TenantID        string
  PrincipalID     string
  Protocol        string
  InboxSubject    string    // in.{protocol}.{tenant}.{agent}
  OutboxSubject   string    // out.{session_id}
  IssuedAt        time.Time
  ExpiresAt       time.Time
  GatewayNode     string    // Hint for sticky routing
}
```

Token is HMAC-SHA256 signed and base64 encoded.

Part 6: Production Architecture Summary

6.1 Key Design Decisions

Decision	Choice	Rationale
Gateway	Custom Go	Envoy can't bridge sockets to NATS
Message Bus	NATS JetStream	At-least-once, subject addressing, persistence
Authorization	Cedar (ABAC)	Resource-level policies for agentic isolation
Connection Draining	Resume Tokens	Session durability > connection durability
Outbound Calls	HTTP/2 Pool	5000 concurrent streams per provider

6.2 Capacity Planning

Component	Instance	Connections	Count for 1M
Go Gateway	c6g.xlarge (8GB)	80,000	13 instances
NATS	r6g.xlarge	N/A	3 nodes (cluster)
Lambda (MCP)	1024MB	N/A	1000 concurrent
Lambda (A2A)	2048MB	N/A	500 concurrent
Lambda (FC)	2048MB	N/A	200 concurrent

Estimated Monthly Cost (1M connections): ~$8,000-15,000/month

6.3 File Structure

```
/radiant
  /apps/gateway/              # Go Gateway service
     /cmd/gateway/main.go
     /internal/
        /config/             # Configuration
        /server/             # WebSocket server, ingress, egress
        /session/            # Session context
        /auth/               # mTLS, OIDC
        /protocol/           # Protocol detection
        /resume/             # Resume tokens
        /nats/               # NATS client
        /health/             # Health endpoints
     /pkg/messages/          # Message structs

  /services/egress-proxy/     # HTTP/2 connection pool (Fargate)
     /src/
        index.ts             # Fastify server
        pool.ts              # HTTP/2 pool management
        providers.ts         # AI provider configs

  /infrastructure/docker/gateway/
     docker-compose.yml      # Local dev stack
     mock-worker/            # Mock Lambda for testing

  /packages/infrastructure/lib/stacks/
      gateway-stack.ts        # CDK deployment
```

Part 7: Implementation Status

Phase	Status	Deliverables
Go Gateway	Complete	`apps/gateway/` - 16 files (incl. TLS)
Egress Proxy	Complete	`services/egress-proxy/` - 5 files
NATS Setup	Complete	Docker Compose with JetStream
CDK Stack	Complete	`gateway-stack.ts`
Resume Tokens	Complete	HMAC-signed tokens
TLS/mTLS	Complete	`internal/server/tls.go`
Cedar Schema	Complete	`infrastructure/cedar/schema.cedarschema`
Cedar Policies	Complete	`infrastructure/cedar/policies/` (2 files)
Cedar Service	Complete	`lambda/shared/services/cedar/`
MCP Lambda Worker	Complete	`lambda/gateway/mcp-worker.ts`
Load Testing	Complete	`infrastructure/load-tests/` (k6 scripts)
Integration Tests	Complete	`__tests__/gateway/mcp-worker.test.ts`

Document Version: 3.1
Status: Implementation Complete
Last Updated: January 2026
Implementation: RADIANT v5.28.0