

RIS Server Documentation

Table of Contents

1. [Overview](#)
2. [Architecture](#)
3. [Installation & Setup](#)
4. [Configuration](#)
5. [API Reference](#)
6. [Client Management](#)
7. [Task Orchestration](#)
8. [Multi-Tenant Architecture](#)
9. [Security Framework](#)
10. [Database Management](#)
11. [Monitoring & Observability](#)
12. [Deployment Strategies](#)
13. [Performance & Scaling](#)
14. [Troubleshooting](#)
15. [Administration Guide](#)

Overview

The RIS (Remote Integrated Services) Server is the central command and control hub for the distributed RIS ecosystem. It provides unified management, orchestration, and monitoring capabilities for multiple remote RIS clients across different environments. The server implements a sophisticated hub-and-spoke architecture with enterprise-grade security, scalability, and reliability features.

Key Features

- **Centralized Management:** Unified control plane for all remote RIS clients
- **Real-time Communication:** WebSocket-based persistent connections with clients
- **Multi-Tenant Support:** Complete tenant isolation with secure data separation
- **Task Orchestration:** Intelligent task routing and execution across clients
- **High Availability:** Clustered deployment with automatic failover
- **Security:** End-to-end encryption, authentication, and comprehensive audit logging
- **Scalability:** Horizontal scaling supporting 1000+ concurrent client connections
- **Monitoring:** Real-time monitoring, metrics collection, and alerting

System Requirements

Minimum Requirements

- **Operating System:** Linux (Ubuntu 20.04+, CentOS 8+, RHEL 8+)
- **CPU:** 2 vCores
- **Memory:** 4 GB RAM
- **Storage:** 20 GB SSD

- **Network:** High-speed internet with static IP

Recommended Requirements

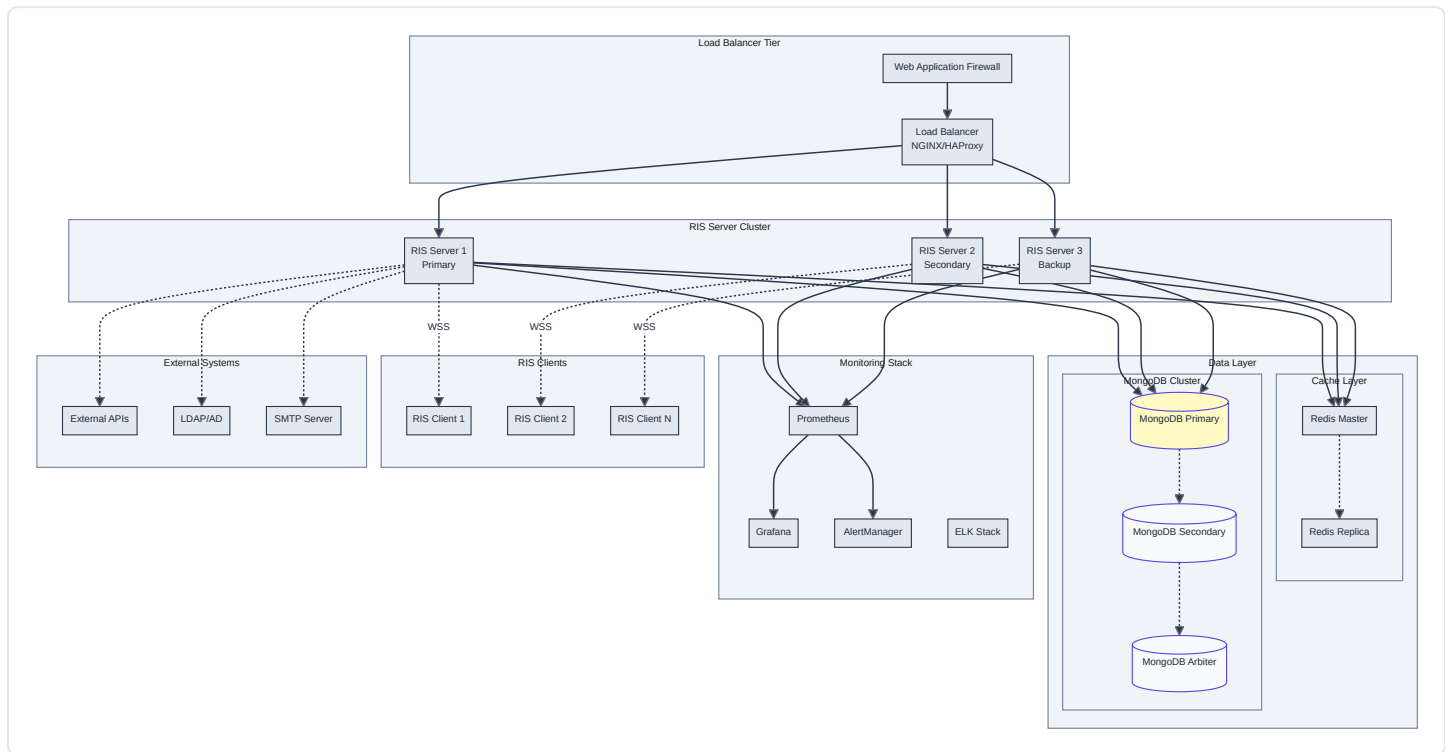
- **Operating System:** Linux (Ubuntu 22.04+, CentOS 9+, RHEL 9+)
- **CPU:** 8 vCores
- **Memory:** 16 GB RAM
- **Storage:** 100 GB SSD NVMe
- **Network:** Dedicated network with load balancer

Dependencies

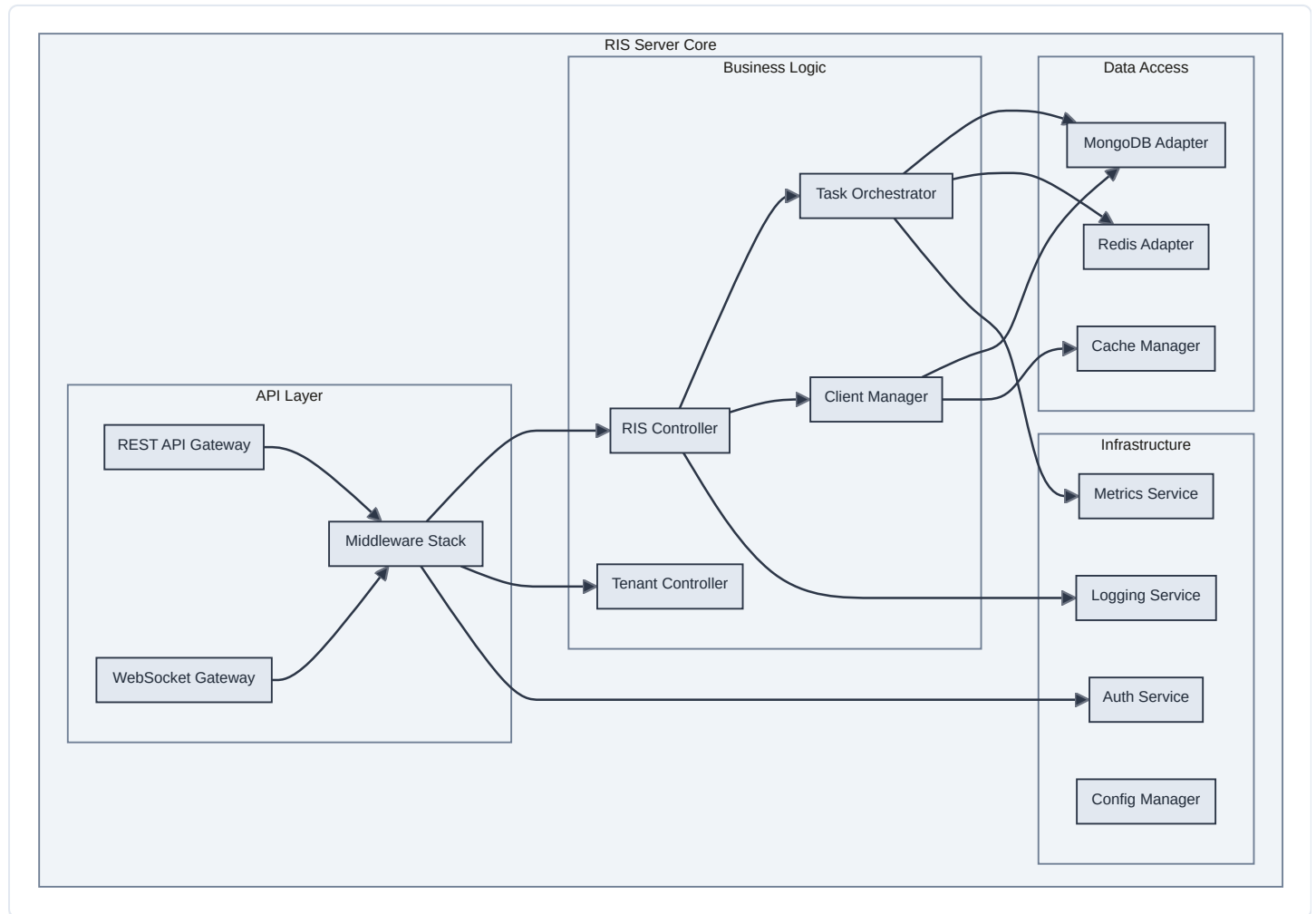
- **MongoDB:** Version 6.0+ (Primary database)
- **Redis:** Version 7.0+ (Caching and session management)
- **Docker:** Version 20.10+ (Containerization)
- **Go Runtime:** Version 1.19+ (for development)
- **TLS Certificates:** Valid certificates for HTTPS/WSS

Architecture

High-Level Architecture



Component Architecture



Detailed Component Design

1. API Gateway Layer

- **REST API Gateway:** HTTP/HTTPS endpoint management
- **WebSocket Gateway:** Real-time client communication
- **Middleware Stack:** Authentication, authorization, logging, rate limiting
- **Request Router:** Intelligent request routing and load balancing

2. Business Logic Layer

- **RIS Controller:** Client lifecycle and status management
- **Tenant Controller:** Multi-tenant data and operation isolation
- **Task Orchestrator:** Task routing, execution, and monitoring
- **Client Manager:** Connection management and health monitoring

3. Data Management Layer

- **MongoDB Integration:** Primary data persistence
- **Redis Integration:** Caching and session management
- **Cache Manager:** Intelligent caching strategies
- **Data Access Objects:** Abstracted data access patterns

Installation & Setup

Docker Deployment (Recommended)

Docker Compose Configuration

```
version: '3.8'

services:
  ris-server:
    image: securaa/ris-server:latest
    container_name: ris-server
    restart: unless-stopped
    ports:
      - "8057:8057"
      - "9090:9090" # Metrics
    environment:
      - RIS_ENV=production
      - MONGODB_URI=mongodb://mongo1:27017,mongo2:27017,mongo3:27017/ris?replicaSet=rs0
      - REDIS_URI=redis://redis-master:6379
      - TLS_CERT_FILE=/app/certs/server.crt
      - TLS_KEY_FILE=/app/certs/server.key
    volumes:
      - ./config:/app/config
      - ./certs:/app/certs
      - ./logs:/app/logs
    networks:
      - ris-network
    depends_on:
      - mongodb
      - redis
    healthcheck:
      test: ["CMD", "curl", "-f", "https://localhost:8057/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 60s

  mongodb:
    image: mongo:6.0
    container_name: ris-mongo
    restart: unless-stopped
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=secure-password
      - MONGO_INITDB_DATABASE=ris
    volumes:
      - mongodb_data:/data/db
      - ./mongo-init:/docker-entrypoint-initdb.d
    networks:
      - ris-network
    command: mongod --replSet rs0 --bind_ip_all

  redis:
    image: redis:7.0-alpine
    container_name: ris-redis
    restart: unless-stopped
    ports:
      - "6379:6379"
    environment:
      - REDIS_PASSWORD=secure-redis-password
    volumes:
      - redis_data:/data
      - ./redis.conf:/usr/local/etc/redis/redis.conf
    networks:
      - ris-network
    command: redis-server /usr/local/etc/redis/redis.conf

  nginx:
    image: nginx:alpine
    container_name: ris-nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./certs:/etc/nginx/certs
```

```

networks:
  - ris-network
depends_on:
  - ris-server

networks:
  ris-network:
    driver: bridge

volumes:
  mongodb_data:
  redis_data:

```

Production Environment Setup

```

# Create directory structure
mkdir -p ris-server/{config,certs,logs,data}
cd ris-server

# Download configuration templates
curl -O https://raw.githubusercontent.com/securaa/ris-server/main/deployments/docker/docker-compose.yml
curl -O https://raw.githubusercontent.com/securaa/ris-server/main/deployments/docker/nginx.conf

# Generate TLS certificates
openssl req -x509 -newkey rsa:4096 -keyout certs/server.key -out certs/server.crt -days 365 -nodes

# Initialize MongoDB replica set
docker-compose exec mongodb mongo --eval "rs.initiate()"

# Start services
docker-compose up -d

# Verify installation
curl -k https://localhost:8057/health

```

Kubernetes Deployment

Namespace and ConfigMap

```

# namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: ris-system
  labels:
    name: ris-system

---
# configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ris-server-config
  namespace: ris-system
data:
  config.json: |
    {
      "server": {
        "host": "0.0.0.0",
        "port": 8057,
        "tls_enabled": true,
        "cert_file": "/app/certs/tls.crt",
        "key_file": "/app/certs/tls.key"
      },
      "database": {
        "mongodb_uri": "mongodb://ris-mongodb:27017/ris",
        "redis_uri": "redis://ris-redis:6379"
      },
      "logging": {
        "level": "info",
        "format": "json"
      }
    }
}

```

Deployment and Service

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ris-server
  namespace: ris-system
  labels:
    app: ris-server
spec:
  replicas: 3
  selector:
    matchLabels:
      app: ris-server
  template:
    metadata:
      labels:
        app: ris-server
    spec:
      containers:
        - name: ris-server
          image: securaa/ris-server:latest
          ports:
            - containerPort: 8057
              name: https
            - containerPort: 9090
              name: metrics
          env:
            - name: CONFIG_PATH
              value: "/app/config/config.json"
          volumeMounts:
            - name: config
              mountPath: /app/config
            - name: certs
              mountPath: /app/certs
      livenessProbe:
        httpGet:
          path: /health
          port: 8057
          scheme: HTTPS
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /ready
          port: 8057
          scheme: HTTPS
        initialDelaySeconds: 5
        periodSeconds: 5
      resources:
        requests:
          memory: "512Mi"
          cpu: "250m"
        limits:
          memory: "2Gi"
          cpu: "1000m"
      volumes:
        - name: config
          configMap:
            name: ris-server-config
        - name: certs
          secret:
            secretName: ris-server-tls
---
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: ris-server-service
  namespace: ris-system
  labels:
    app: ris-server
spec:
  selector:
    app: ris-server
  ports:
    - name: https
      port: 8057
      targetPort: 8057
    - name: metrics
```

```

    port: 9090
    targetPort: 9090
    type: ClusterIP
---
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ris-server-ingress
  namespace: ris-system
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  tls:
  - hosts:
    - ris-server.example.com
      secretName: ris-server-tls
  rules:
  - host: ris-server.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ris-server-service
            port:
              number: 8057

```

Binary Installation

System Setup

```

# Create system user
sudo useradd -r -s /bin/false ris-server
sudo mkdir -p /opt/ris-server/{bin,config,logs,data}
sudo mkdir -p /etc/ris-server
sudo chown -R ris-server:ris-server /opt/ris-server /etc/ris-server

# Download and install binary
wget https://github.com/securaa/ris-server/releases/latest/download/ris-server-linux-amd64.tar.gz
tar -xzf ris-server-linux-amd64.tar.gz
sudo mv ris-server /opt/ris-server/bin/
sudo chmod +x /opt/ris-server/bin/ris-server

```

Systemd Service

```
sudo tee /etc/systemd/system/ris-server.service << EOF
[Unit]
Description=RIS Server
After=network.target mongodb.service redis.service
Wants=network.target
Requires=mongodb.service redis.service

[Service]
Type=simple
User=ris-server
Group=ris-server
ExecStart=/opt/ris-server/bin/ris-server
Restart=always
RestartSec=10
Environment=CONFIG_PATH=/etc/ris-server/config.json
WorkingDirectory=/opt/ris-server
StandardOutput=journal
StandardError=journal
SyslogIdentifier=ris-server

# Security settings
NoNewPrivileges=yes
ProtectSystem=strict
ProtectHome=yes
ReadWritePaths=/opt/ris-server/logs /opt/ris-server/data

[Install]
WantedBy=multi-user.target
EOF

# Enable and start service
sudo systemctl daemon-reload
sudo systemctl enable ris-server
sudo systemctl start ris-server
sudo systemctl status ris-server
```

Configuration

Master Configuration File

```
{
  "server": {
    "host": "0.0.0.0",
    "port": 8057,
    "tls": {
      "enabled": true,
      "cert_file": "/app/certs/server.crt",
      "key_file": "/app/certs/server.key",
      "ca_file": "/app/certs/ca.crt",
      "min_version": "1.2",
      "max_version": "1.3",
      "cipher_suites": [
        "TLS_AES_256_GCM_SHA384",
        "TLS_CHACHA20_POLY1305_SHA256",
        "TLS_AES_128_GCM_SHA256"
      ],
      "client_auth": "require"
    },
    "timeouts": {
      "read": "30s",
      "write": "30s",
      "idle": "120s"
    },
    "cors": {
      "enabled": true,
      "allowed_origins": ["https://admin.example.com"],
      "allowed_methods": ["GET", "POST", "PUT", "DELETE"],
      "allowed_headers": ["Authorization", "Content-Type"]
    }
  },
  "database": {
    "mongodb": {
      "uri": "mongodb://username:password@localhost:27017/ris?replicaSet=rs0",
      "database": "ris",
      "max_pool_size": 100,
      "min_pool_size": 10,
      "max_idle_time": "10m",
      "connection_timeout": "10s",
      "socket_timeout": "30s",
      "server_selection_timeout": "5s"
    },
    "redis": {
      "uri": "redis://localhost:6379",
      "password": "secure-password",
      "database": 0,
      "max_retries": 3,
      "pool_size": 50,
      "min_idle_conns": 10,
      "idle_timeout": "5m",
      "dial_timeout": "5s",
      "read_timeout": "3s",
      "write_timeout": "3s"
    }
  },
  "authentication": {
    "jwt": {
      "secret": "your-jwt-secret-key",
      "expiry": "24h",
      "refresh_expiry": "168h",
      "issuer": "ris-server",
      "algorithm": "HS256"
    },
    "session": {
      "duration": "8h",
      "renewal_threshold": "1h",
      "max_concurrent": 5
    }
  },
  "ldap": {
    "enabled": false,
    "url": "ldap://ldap.example.com:389",
    "base_dn": "ou=users,dc=example,dc=com",
    "bind_dn": "cn=admin,dc=example,dc=com",
    "bind_password": "admin-password",
    "user_filter": "(uid=%s)",
    "group_filter": "(member=%s)"
  }
}
```

```

},
"authorization": {
  "rbac": {
    "enabled": true,
    "default_role": "readonly",
    "admin_users": ["admin@example.com"],
    "super_admin_users": ["superadmin@example.com"]
  }
},
"multi_tenant": {
  "enabled": true,
  "default_tenant": "default",
  "tenant_isolation": "database",
  "auto_provision": false
},
"task_orchestration": {
  "max_concurrent_tasks": 1000,
  "task_timeout": "300s",
  "retry_policy": {
    "max_attempts": 3,
    "backoff_strategy": "exponential",
    "initial_delay": "1s",
    "max_delay": "60s"
  },
  "queue": {
    "size": 10000,
    "priority_levels": 5,
    "dead_letter_enabled": true
  }
},
"client_management": {
  "max_clients": 1000,
  "heartbeat_interval": "30s",
  "connection_timeout": "60s",
  "reconnection_policy": {
    "max_attempts": 5,
    "backoff_strategy": "exponential",
    "initial_delay": "5s",
    "max_delay": "300s"
  }
},
"security": {
  "rate_limiting": {
    "enabled": true,
    "requests_per_minute": 100,
    "burst_size": 20,
    "ip_whitelist": ["10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16"]
  },
  "audit": {
    "enabled": true,
    "events": ["authentication", "authorization", "task_execution", "client_management"],
    "retention_days": 90,
    "format": "json"
  },
  "encryption": {
    "data_at_rest": {
      "enabled": true,
      "algorithm": "AES-256-GCM",
      "key_rotation_days": 90
    },
    "data_in_transit": {
      "enabled": true,
      "min_tls_version": "1.2"
    }
  }
},
"monitoring": {
  "metrics": {
    "enabled": true,
    "port": 9090,
    "path": "/metrics",
    "interval": "15s"
  },
  "health_check": {
    "enabled": true,
    "endpoint": "/health",
    "detailed": true
  },
  "alerting": {
    "enabled": true,
    "webhook_url": "https://alerts.example.com/webhook",
    "thresholds": {
      "cpu_usage": 80,
      "memory_usage": 85,

```

```

    "error_rate": 5,
    "response_time": 5000
  }
},
"logging": {
  "level": "info",
  "format": "json",
  "output": "stdout",
  "file": {
    "enabled": true,
    "path": "/app/logs/ris-server.log",
    "max_size": "100MB",
    "max_backups": 10,
    "max_age": 30,
    "compress": true
  },
  "audit_log": {
    "enabled": true,
    "path": "/app/logs/audit.log",
    "max_size": "100MB",
    "max_backups": 30,
    "max_age": 365
  }
},
"performance": {
  "cache": {
    "ttl": "1h",
    "max_size": "500MB",
    "eviction_policy": "lru"
  },
  "connection_pooling": {
    "max_connections": 200,
    "idle_timeout": "300s",
    "keep_alive": true
  },
  "graceful_shutdown": {
    "timeout": "30s",
    "drain_timeout": "60s"
  }
},
"features": {
  "websocket_compression": true,
  "http2_enabled": true,
  "debug_mode": false,
  "development_mode": false,
  "auto_backup": true,
  "telemetry": false
}
}

```

Environment-Specific Configurations

Development Configuration

```
{
  "server": {
    "host": "localhost",
    "port": 8057,
    "tls": {
      "enabled": false
    }
  },
  "database": {
    "mongodb": {
      "uri": "mongodb://localhost:27017/ris_dev"
    },
    "redis": {
      "uri": "redis://localhost:6379",
      "database": 1
    }
  },
  "logging": {
    "level": "debug",
    "output": "stdout"
  },
  "features": {
    "debug_mode": true,
    "development_mode": true
  }
}
```

Production Configuration

```
{
  "server": {
    "host": "0.0.0.0",
    "port": 8057,
    "tls": {
      "enabled": true,
      "cert_file": "/app/certs/server.crt",
      "key_file": "/app/certs/server.key",
      "client_auth": "require"
    }
  },
  "database": {
    "mongodb": {
      "uri": "mongodb://username:password@mongo1:27017,mongo2:27017,mongo3:27017/ris?replicaSet=rs0&ssl=true"
    },
    "redis": {
      "uri": "redis://redis-cluster:6379",
      "password": "secure-redis-password",
      "pool_size": 100
    }
  },
  "logging": {
    "level": "info",
    "format": "json",
    "output": "file"
  },
  "security": {
    "rate_limiting": {
      "enabled": true,
      "requests_per_minute": 60
    },
    "audit": {
      "enabled": true,
      "retention_days": 365
    }
  },
  "features": {
    "debug_mode": false,
    "development_mode": false,
    "telemetry": true
  }
}
```

Environment Variables

VARIABLE	DESCRIPTION	DEFAULT
RIS_ENV	Environment (dev/staging/prod)	development
CONFIG_PATH	Configuration file path	/app/config/config.json
MONGODB_URI	MongoDB connection string	mongodb://localhost:27017/ris
REDIS_URI	Redis connection string	redis://localhost:6379
TLS_CERT_FILE	TLS certificate file path	/app/certs/server.crt
TLS_KEY_FILE	TLS private key file path	/app/certs/server.key
JWT_SECRET	JWT signing secret	Generated
LOG_LEVEL	Logging level	info
METRICS_PORT	Metrics server port	9090

API Reference

Authentication

JWT Token Authentication

```
POST /auth/login
Content-Type: application/json

{
  "username": "admin@example.com",
  "password": "secure-password",
  "tenant_code": "default"
}
```

Response:

```
{
  "success": true,
  "data": {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "expires_in": 86400,
    "token_type": "Bearer",
    "user": {
      "id": "user-123",
      "email": "admin@example.com",
      "role": "admin",
      "tenant_code": "default"
    }
  }
}
```

Token Refresh

```
POST /auth/refresh
Content-Type: application/json
Authorization: Bearer <refresh_token>

{
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

RIS Client Management

Get All RIS Clients

```
GET /platform/v1/ris
Authorization: Bearer <access_token>
```

Response:

```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "client-001",
      "description": "Production client",
      "host": "192.168.1.100",
      "status": "active",
      "connection_status": "connected",
      "version": "1.0.0",
      "unique_client_id": "uuid-string",
      "tenant_code": "default",
      "created_date": 1696752000000,
      "updated_date": 1696752000000,
      "last_seen": "2025-10-07T10:30:00Z",
      "capabilities": ["task_execution", "service_management"]
    }
  ],
  "total": 1,
  "page": 1,
  "per_page": 10
}
```

Add New RIS Client

```
POST /platform/v1/ris
Content-Type: application/json
Authorization: Bearer <access_token>
```

```
{
  "name": "client-002",
  "description": "Development client",
  "host": "192.168.1.101",
  "tenant_code": "dev"
}
```

Update RIS Client

```
PUT /platform/v1/ris
Content-Type: application/json
Authorization: Bearer <access_token>
```

```
{
  "id": 1,
  "name": "client-001-updated",
  "description": "Updated production client",
  "status": "active"
}
```

Delete RIS Client

```
DELETE /platform/v1/ris
Content-Type: application/json
Authorization: Bearer <access_token>
```

```
{
  "id": 1
}
```

Get Client Configuration

```
GET /platform/v1/risclient/{client-id}/{hostname}
Authorization: Bearer <access_token>
```

Response:

```
{
  "success": true,
  "data": {
    "client_config": {
      "server_host": "https://ris-server.example.com:8057",
      "websocket_path": "/pingpong",
      "heartbeat_interval": 30,
      "reconnect_policy": {
        "max_attempts": 5,
        "backoff_strategy": "exponential"
      }
    },
    "certificates": {
      "client_cert": "-----BEGIN CERTIFICATE-----\n...",
      "client_key": "-----BEGIN PRIVATE KEY-----\n...",
      "ca_cert": "-----BEGIN CERTIFICATE-----\n..."
    },
    "tenant_info": {
      "tenant_code": "default",
      "tenant_name": "Default Tenant",
      "database_config": {
        "host": "tenant-db.example.com",
        "database": "tenant_default"
      }
    }
  }
}
```

Task Management

Execute Task on Client

```
POST /platform/v1/task/execute
Content-Type: application/json
Authorization: Bearer <access_token>

{
  "client_id": "uuid-string",
  "task": {
    "type": "http_request",
    "method": "GET",
    "url": "https://api.example.com/status",
    "headers": {
      "Authorization": "Bearer token"
    },
    "timeout": 30
  },
  "priority": "normal",
  "async": false
}
```

Response:

```
{
  "success": true,
  "data": {
    "task_id": "task-123",
    "status": "completed",
    "result": {
      "status_code": 200,
      "headers": {},
      "body": "OK",
      "execution_time": 1250
    },
    "started_at": "2025-10-07T10:30:00Z",
    "completed_at": "2025-10-07T10:30:01Z"
  }
}
```

Get Task Status

```
GET /platform/v1/task/{task-id}
Authorization: Bearer <access_token>
```

List Tasks

```
GET /platform/v1/tasks?status=running&client_id=uuid-string&limit=50
Authorization: Bearer <access_token>
```

Service Management

Deploy Service to Client

```
POST /platform/v1/service/deploy
Content-Type: application/json
Authorization: Bearer <access_token>
```

```
{
  "client_id": "uuid-string",
  "service": {
    "name": "web-app",
    "image": "nginx:latest",
    "replicas": 3,
    "ports": [
      {
        "container_port": 80,
        "host_port": 8080
      }
    ],
    "environment": {
      "ENV": "production"
    },
    "resources": {
      "limits": {
        "memory": "512m",
        "cpu": "0.5"
      }
    }
  }
}
```

Scale Service

```
POST /platform/v1/service/scale
Content-Type: application/json
Authorization: Bearer <access_token>
```

```
{
  "client_id": "uuid-string",
  "service_name": "web-app",
  "replicas": 5
}
```

Get Service Status

```
GET /platform/v1/service/{client-id}/{service-name}
Authorization: Bearer <access_token>
```

Tenant Management

Get Tenant Information

```
GET /internal/v1/remotetenantclient/{tenant-uuid}
Authorization: Bearer <access_token>
```

Get Tenant Configuration

```
GET /platform/v1/remotetenantconfigdata/{tenant-id}/{hostname}
Authorization: Bearer <access_token>
```

Monitoring and Health

Server Health Check

```
GET /health
```

Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "uptime": "2h30m15s",
  "timestamp": "2025-10-07T10:30:00Z",
  "components": {
    "database": {
      "status": "connected",
      "response_time": 12,
      "connections": {
        "active": 15,
        "idle": 5,
        "max": 100
      }
    },
    "cache": {
      "status": "connected",
      "response_time": 2,
      "memory_usage": "45.2%",
      "hit_ratio": "89.5%"
    },
    "clients": {
      "total": 25,
      "connected": 23,
      "disconnected": 2,
      "last_heartbeat": "2025-10-07T10:29:45Z"
    },
    "tasks": {
      "running": 12,
      "queued": 3,
      "completed_today": 1547,
      "failed_today": 8
    },
    "system": {
      "cpu_usage": 25.5,
      "memory_usage": 65.2,
      "disk_usage": 42.1,
      "network_io": {
        "bytes_in": 1048576,
        "bytes_out": 2097152
      }
    }
  }
}
```

Readiness Check

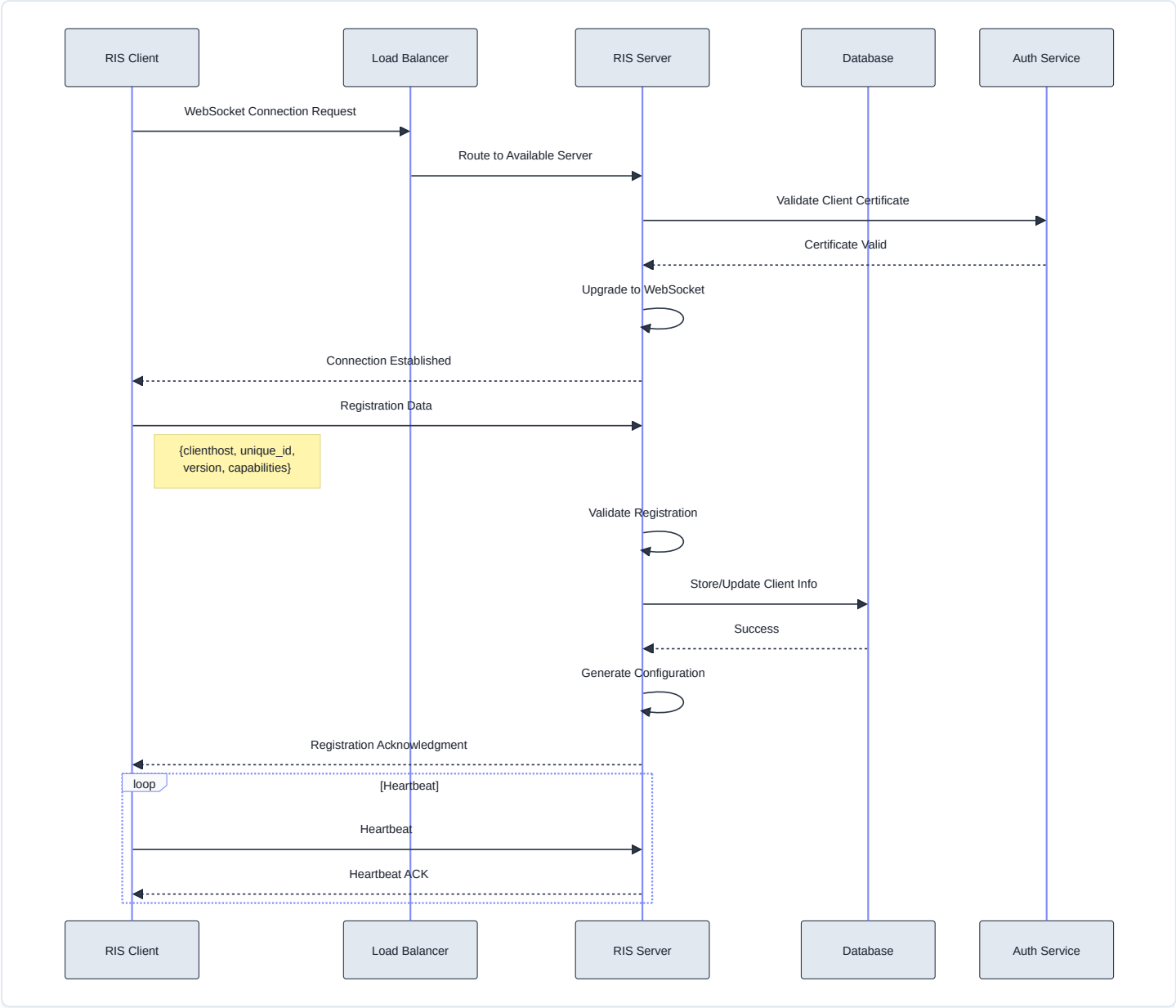
GET /ready

Metrics Endpoint

GET /metrics

Client Management

Client Registration Process



Client Lifecycle Management

Registration

```
type ClientRegistration struct {
    ClientHost      string `json:"clienthost"`
    UniqueClientID  string `json:"unique_client_id"`
    Version         string `json:"version"`
    Capabilities    []string `json:"capabilities"`
    TenantCode      string `json:"tenantcode"`
    HostType        string `json:"host_type"`
    IsFirstPing     string `json:"isfirstping"`
    IsNetworkChanged string `json:"isnetworkchanged"`
    RISNatIP        string `json:"ris_nat_ip"`
}
```

Status Monitoring

```
type ClientStatus struct {
    ID                int    `json:"id"`
    UniqueClientID    string `json:"unique_client_id"`
    Status            string `json:"status"`
    ConnectionStatus  string `json:"connection_status"`
    LastSeen          time.Time `json:"last_seen"`
    IPAddress         string `json:"ip_address"`
    Version           string `json:"version"`
    Capabilities      []string `json:"capabilities"`
    ResourceUsage     struct {
        CPU    float64 `json:"cpu"`
        Memory float64 `json:"memory"`
        Disk   float64 `json:"disk"`
    } `json:"resource_usage"`
}
```

Connection Management

```
type ConnectionManager struct {
    clients          map[string]*ClientConnection
    disconnectedCount map[string]int
    mutex            sync.RWMutex
    heartbeatInterval time.Duration
    timeoutDuration  time.Duration
}

func (cm *ConnectionManager) AddClient(clientID string, conn *websocket.Conn) error
func (cm *ConnectionManager) RemoveClient(clientID string) error
func (cm *ConnectionManager) GetClient(clientID string) (*ClientConnection, error)
func (cm *ConnectionManager) BroadcastToClients(message []byte) error
func (cm *ConnectionManager) GetConnectedClients() []string
```

Client Authentication and Authorization

Certificate-Based Authentication

```
type ClientAuth struct {
    CertificateAuth bool `json:"certificate_auth"`
    ClientCert      string `json:"client_cert"`
    ClientKey       string `json:"client_key"`
    CACert          string `json:"ca_cert"`
}

func ValidateClientCertificate(cert *x509.Certificate, caCert *x509.Certificate) error {
    // Certificate validation logic
    roots := x509.NewCertPool()
    roots.AddCert(caCert)

    opts := x509.VerifyOptions{Roots: roots}
    _, err := cert.Verify(opts)
    return err
}
```

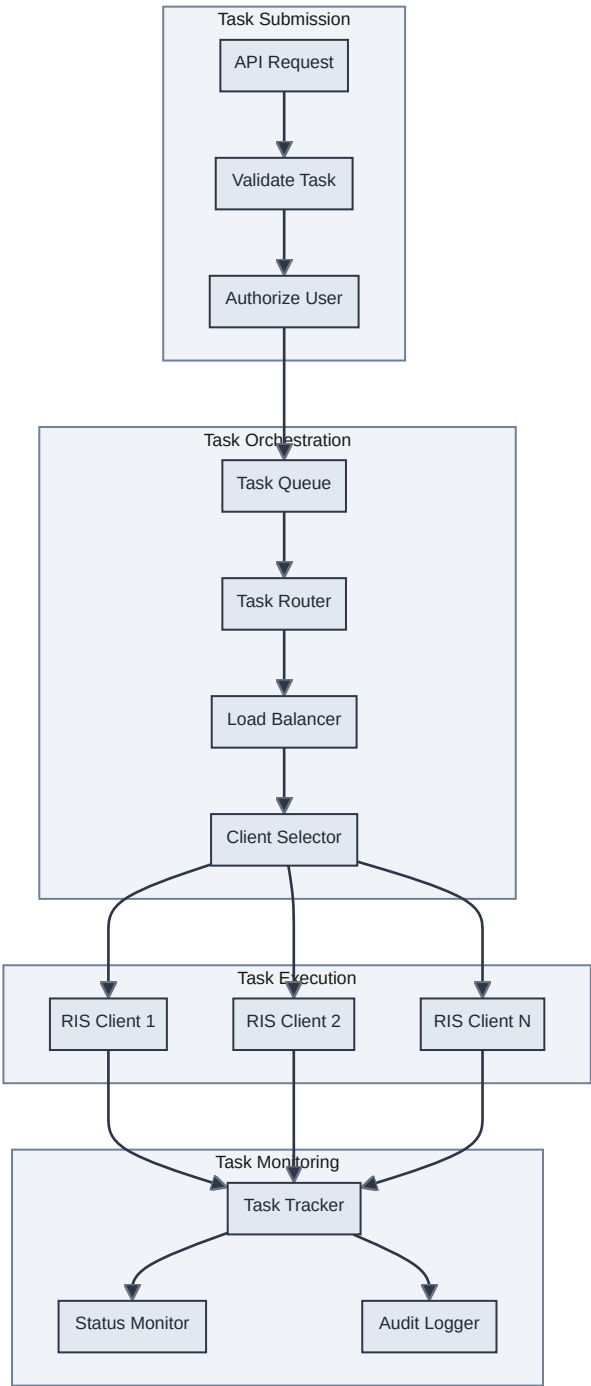
Token-Based Authentication

```
type ClientTokenAuth struct {
    TokenType    string    `json:"token_type"`
    AccessToken  string    `json:"access_token"`
    RefreshToken string    `json:"refresh_token"`
    ExpiresAt    time.Time `json:"expires_at"`
}

func ValidateClientToken(token string) (*ClientClaims, error) {
    // JWT token validation
    claims := &ClientClaims{}
    _, err := jwt.ParseWithClaims(token, claims, func(token *jwt.Token) (interface{}, error) {
        return jwtSecret, nil
    })
    return claims, err
}
```

Task Orchestration

Task Routing Engine



Task Types and Schemas

HTTP Request Task

```

type HTTPTask struct {
    TaskID      string          `json:"task_id"`
    Type        string          `json:"type"`
    Method      string          `json:"method"`
    URL         string          `json:"url"`
    Headers     map[string]string `json:"headers"`
    Body        string          `json:"body"`
    Timeout     int             `json:"timeout"`
    Retries     int             `json:"retries"`
    Auth        AuthConfig      `json:"auth"`
    Validation  ValidationConfig `json:"validation"`
}

type AuthConfig struct {
    Type        string `json:"type"`
    Username    string `json:"username"`
    Password    string `json:"password"`
    BearerToken string `json:"bearer_token"`
    APIKey      string `json:"api_key"`
    CustomHeader string `json:"custom_header"`
}

```

Docker Service Task

```

type DockerTask struct {
    TaskID      string          `json:"task_id"`
    Type        string          `json:"type"`
    Action      string          `json:"action"`
    ServiceName string          `json:"service_name"`
    Image       string          `json:"image"`
    Replicas    int             `json:"replicas"`
    Ports       []PortMapping   `json:"ports"`
    Environment map[string]string `json:"environment"`
    Volumes     []VolumeMapping `json:"volumes"`
    Networks    []string        `json:"networks"`
    Resources   ResourceConstraints `json:"resources"`
    HealthCheck HealthCheckConfig `json:"health_check"`
}

type PortMapping struct {
    ContainerPort int `json:"container_port"`
    HostPort      int `json:"host_port"`
    Protocol      string `json:"protocol"`
}

```

System Command Task

```

type CommandTask struct {
    TaskID      string          `json:"task_id"`
    Type        string          `json:"type"`
    Command     string          `json:"command"`
    Args        []string        `json:"args"`
    WorkingDir  string          `json:"working_dir"`
    Environment map[string]string `json:"environment"`
    Timeout     int             `json:"timeout"`
    RunAsUser   string          `json:"run_as_user"`
    Privileged  bool            `json:"privileged"`
}

```

Task Execution Flow

Task Lifecycle

```

type TaskStatus string

const (
    TaskStatusPending   TaskStatus = "pending"
    TaskStatusQueued     TaskStatus = "queued"
    TaskStatusRunning    TaskStatus = "running"
    TaskStatusCompleted TaskStatus = "completed"
    TaskStatusFailed     TaskStatus = "failed"
    TaskStatusCancelled TaskStatus = "cancelled"
    TaskStatusTimeout    TaskStatus = "timeout"
)

type Task struct {
    ID            string          `json:"id" bson:"_id"`
    Type          string          `json:"type" bson:"type"`
    Status        TaskStatus      `json:"status" bson:"status"`
    Priority       int             `json:"priority" bson:"priority"`
    ClientID      string          `json:"client_id" bson:"client_id"`
    TenantCode    string          `json:"tenant_code" bson:"tenant_code"`
    UserID        string          `json:"user_id" bson:"user_id"`
    Payload       map[string]interface{} `json:"payload" bson:"payload"`
    Result        *TaskResult     `json:"result,omitempty" bson:"result,omitempty"`
    CreatedAt     time.Time       `json:"created_at" bson:"created_at"`
    StartedAt     *time.Time      `json:"started_at,omitempty" bson:"started_at,omitempty"`
    CompletedAt   *time.Time      `json:"completed_at,omitempty" bson:"completed_at,omitempty"`
    Timeout       time.Duration   `json:"timeout" bson:"timeout"`
    Retries       int             `json:"retries" bson:"retries"`
    MaxRetries    int             `json:"max_retries" bson:"max_retries"`
    Metadata      map[string]interface{} `json:"metadata" bson:"metadata"`
}

```

Task Orchestrator Implementation

```

type TaskOrchestrator struct {
    queue         TaskQueue
    router         TaskRouter
    clientManager ClientManager
    taskTracker   TaskTracker
    config         OrchestratorConfig
    metrics        MetricsCollector
    logger         Logger
}

func (to *TaskOrchestrator) SubmitTask(task *Task) error {
    // Validate task
    if err := to.validateTask(task); err != nil {
        return fmt.Errorf("task validation failed: %w", err)
    }

    // Add to queue
    if err := to.queue.Enqueue(task); err != nil {
        return fmt.Errorf("failed to queue task: %w", err)
    }

    // Start processing if not async
    if !task.Async {
        return to.processTask(task)
    }

    return nil
}

func (to *TaskOrchestrator) processTask(task *Task) error {
    // Select appropriate client
    client, err := to.selectClient(task)
    if err != nil {
        return fmt.Errorf("client selection failed: %w", err)
    }

    // Route task to client
    return to.router.RouteTask(task, client)
}

```

Priority and Queue Management

Priority Queue Implementation

```

type PriorityQueue struct {
    queues    map[int]*Queue // Priority level -> Queue
    mutex     sync.RWMutex
    maxSize   int
    metrics   *QueueMetrics
}

func (pq *PriorityQueue) Enqueue(task *Task) error {
    pq.mutex.Lock()
    defer pq.mutex.Unlock()

    priority := task.Priority
    if priority < 1 || priority > 5 {
        priority = 3 // Default priority
    }

    queue, exists := pq.queues[priority]
    if !exists {
        queue = NewQueue(pq.maxSize / 5)
        pq.queues[priority] = queue
    }

    return queue.Enqueue(task)
}

func (pq *PriorityQueue) Dequeue() (*Task, error) {
    pq.mutex.Lock()
    defer pq.mutex.Unlock()

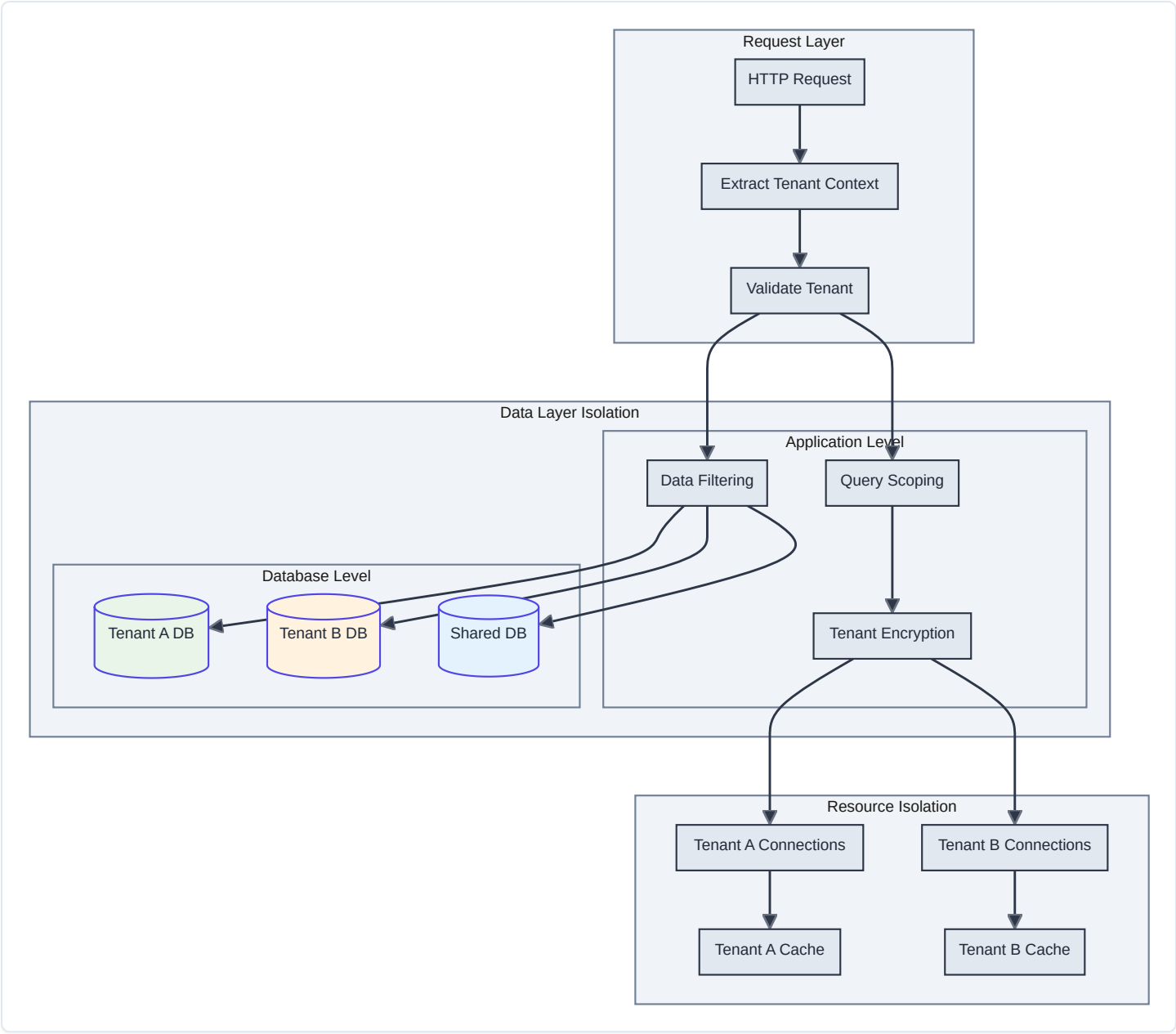
    // Process highest priority first
    for priority := 5; priority >= 1; priority-- {
        if queue, exists := pq.queues[priority]; exists {
            if task, err := queue.Dequeue(); err == nil {
                return task, nil
            }
        }
    }

    return nil, ErrQueueEmpty
}

```

Multi-Tenant Architecture

Tenant Isolation Strategies



Tenant Management

Tenant Model

```

type Tenant struct {
    ID            string          `json:"id" bson:"_id"`
    Code          string          `json:"code" bson:"code"`
    Name          string          `json:"name" bson:"name"`
    Description    string          `json:"description" bson:"description"`
    Status        string          `json:"status" bson:"status"`
    DatabaseInfo  DatabaseInfo    `json:"database_info" bson:"database_info"`
    Settings      TenantSettings  `json:"settings" bson:"settings"`
    Limits        ResourceLimits  `json:"limits" bson:"limits"`
    CreatedAt     time.Time       `json:"created_at" bson:"created_at"`
    UpdatedAt     time.Time       `json:"updated_at" bson:"updated_at"`
    Metadata      map[string]interface{} `json:"metadata" bson:"metadata"`
}

type DatabaseInfo struct {
    Type      string `json:"type" bson:"type"`
    Host      string `json:"host" bson:"host"`
    Port      int    `json:"port" bson:"port"`
    Database  string `json:"database" bson:"database"`
    Username  string `json:"username" bson:"username"`
    Password  string `json:"password" bson:"password"`
    SSL       bool   `json:"ssl" bson:"ssl"`
    MaxPoolSize int    `json:"max_pool_size" bson:"max_pool_size"`
}

type TenantSettings struct {
    TimeZone      string          `json:"timezone" bson:"timezone"`
    Language      string          `json:"language" bson:"language"`
    Features      []string        `json:"features" bson:"features"`
    SecurityPolicy SecurityPolicy  `json:"security_policy" bson:"security_policy"`
    NotificationConfig NotificationConfig `json:"notification_config" bson:"notification_config"`
}

type ResourceLimits struct {
    MaxClients    int `json:"max_clients" bson:"max_clients"`
    MaxTasks      int `json:"max_tasks" bson:"max_tasks"`
    MaxStorageGB  int `json:"max_storage_gb" bson:"max_storage_gb"`
    MaxBandwidthMBs int `json:"max_bandwidth_mbs" bson:"max_bandwidth_mbs"`
}

```

Tenant Context Management

```

type TenantContext struct {
    TenantID      string          `json:"tenant_id"`
    TenantCode    string          `json:"tenant_code"`
    UserID        string          `json:"user_id"`
    Permissions    []string        `json:"permissions"`
    Settings      TenantSettings  `json:"settings"`
    Limits        ResourceLimits  `json:"limits"`
    DBSession     database.Session `json:"- "`
    CacheClient    cache.Client     `json:"- "`
    Metadata      map[string]interface{} `json:"metadata"`
}

type TenantManager struct {
    tenantRepo    TenantRepository
    dbManager     DatabaseManager
    cacheManager  CacheManager
    encryptionSvc EncryptionService
    mutex         sync.RWMutex
    tenantCache   map[string]*TenantContext
}

func (tm *TenantManager) GetTenantContext(tenantCode string) (*TenantContext, error) {
    // Check cache first
    if ctx, exists := tm.tenantCache[tenantCode]; exists {
        return ctx, nil
    }

    // Load from database
    tenant, err := tm.tenantRepo.GetByCode(tenantCode)
    if err != nil {
        return nil, fmt.Errorf("tenant not found: %w", err)
    }

    // Create database session
    dbSession, err := tm.dbManager.CreateTenantSession(tenant.DatabaseInfo)
    if err != nil {
        return nil, fmt.Errorf("failed to create DB session: %w", err)
    }

    // Create cache client
    cacheClient, err := tm.cacheManager.CreateTenantClient(tenantCode)
    if err != nil {
        return nil, fmt.Errorf("failed to create cache client: %w", err)
    }

    ctx := &TenantContext{
        TenantID:    tenant.ID,
        TenantCode:  tenant.Code,
        Settings:    tenant.Settings,
        Limits:      tenant.Limits,
        DBSession:   dbSession,
        CacheClient: cacheClient,
    }

    // Cache context
    tm.tenantCache[tenantCode] = ctx

    return ctx, nil
}

```

Database Isolation

Tenant-Specific Database Connections

```

type DatabaseManager struct {
    defaultDB      database.Database
    tenantDBs      map[string]database.Database
    poolManager    ConnectionPoolManager
    encryptionSvc  EncryptionService
    mutex          sync.RWMutex
}

func (dm *DatabaseManager) CreateTenantSession(dbInfo DatabaseInfo) (database.Session, error) {
    tenantKey := fmt.Sprintf("%s:%s", dbInfo.Host, dbInfo.Database)

    // Check if tenant DB connection exists
    db, exists := dm.tenantDBs[tenantKey]
    if !exists {
        // Create new connection
        var err error
        db, err = dm.createTenantConnection(dbInfo)
        if err != nil {
            return nil, fmt.Errorf("failed to create tenant connection: %w", err)
        }

        dm.mutex.Lock()
        dm.tenantDBs[tenantKey] = db
        dm.mutex.Unlock()
    }

    // Create session with tenant context
    session := db.NewSession()
    session.SetTenantContext(dbInfo.Database)

    return session, nil
}

func (dm *DatabaseManager) createTenantConnection(dbInfo DatabaseInfo) (database.Database, error) {
    // Decrypt database credentials
    password, err := dm.encryptionSvc.Decrypt(dbInfo.Password)
    if err != nil {
        return nil, fmt.Errorf("failed to decrypt password: %w", err)
    }

    // Build connection URI
    uri := fmt.Sprintf("mongodb://%s:%s@%s:%d/%s",
        dbInfo.Username, password, dbInfo.Host, dbInfo.Port, dbInfo.Database)

    // Configure connection options
    opts := &database.Options{
        MaxPoolSize:    dbInfo.MaxPoolSize,
        ConnectTimeout: 30 * time.Second,
        SocketTimeout:  60 * time.Second,
        SSL:            dbInfo.SSL,
        ReplicaSet:     "", // Configure if using replica set
    }

    return database.Connect(uri, opts)
}

```

Data Encryption per Tenant

```

type TenantEncryption struct {
    keyManager      KeyManager
    encryptionSvc   EncryptionService
    tenantKeys      map[string][]byte
    mutex           sync.RWMutex
}

func (te *TenantEncryption) EncryptTenantData(tenantCode string, data []byte) ([]byte, error) {
    key, err := te.getTenantKey(tenantCode)
    if err != nil {
        return nil, fmt.Errorf("failed to get tenant key: %w", err)
    }

    return te.encryptionSvc.Encrypt(data, key)
}

func (te *TenantEncryption) DecryptTenantData(tenantCode string, encryptedData []byte) ([]byte, error) {
    key, err := te.getTenantKey(tenantCode)
    if err != nil {
        return nil, fmt.Errorf("failed to get tenant key: %w", err)
    }

    return te.encryptionSvc.Decrypt(encryptedData, key)
}

func (te *TenantEncryption) getTenantKey(tenantCode string) ([]byte, error) {
    te.mutex.RLock()
    if key, exists := te.tenantKeys[tenantCode]; exists {
        te.mutex.RUnlock()
        return key, nil
    }
    te.mutex.RUnlock()

    // Generate or retrieve tenant-specific key
    key, err := te.keyManager.GetTenantKey(tenantCode)
    if err != nil {
        return nil, err
    }

    te.mutex.Lock()
    te.tenantKeys[tenantCode] = key
    te.mutex.Unlock()

    return key, nil
}

```

Document Version: 1.0

Last Updated: October 7, 2025

Comprehensive RIS Server Documentation