RemoteIQ — Backend Master Specification

Author: RemoteIQ RMM • Scope: Backend/API/Agents/Infra • Version: 1.0

# 1. Objectives & Non‑Goals

• Deliver a production‑grade backend for an open‑source RMM (RemoteIQ) with multi‑tenancy, RBAC, agents, automation, patching, reporting, and auditability.

• Ensure high reliability, security (least privilege, strong authN/Z, mTLS for agents), and clear contracts via OpenAPI.

Non‑Goals: UI details (covered by frontend spec), vendor‑specific cloud lock‑ins, and on‑prem AD management.

# 2. Architecture Overview

• Language/Framework: Option A) FastAPI (Python 3.11+). Option B) NestJS (TypeScript). Choose one per deployment; contracts remain identical.

• Services: API Gateway, Job/Worker, WebSocket Streamer, Artifact Storage (S3/MinIO), Message Broker (NATS/RabbitMQ), DB (Postgres), Cache (Redis).

• Packaging: Docker (rootless), docker‑compose profiles (dev/demo/prod); optional Kubernetes manifests later.

• Contracts: OpenAPI 3.1 with strict schemas; generated SDK for frontend (TypeScript).

# 3. Multi‑Tenancy & RBAC

• Tenant scoping on every query. Entities contain tenant\_id. Cross‑tenant reads prohibited.

• Role model: Owner, Admin, Technician, Auditor (least privilege). Custom RBAC via policy rules (resource, action, scope).

• Audit log on every privileged action: who, when, IP, resource, before/after snapshots where safe.

# 4. Data Model (Logical)

Core entities:

• Tenant (Organization) → Locations → Departments

• User (belongs to Tenant) with Roles/Policies

• Device (Endpoint): OS, facts, metrics, status, tags

• Agent: version, keys, last\_seen, connectivity

• Script: language (ps1/sh/py), version, provenance

• Job (Run): target set, parameters, schedule, status, logs, artifacts

• Patch: posture, approvals, windows, actions

• Software: inventory per device; actions (uninstall/request)

• Report: template, schedule, recipients, artifacts

• AuditEvent

Indexes: device.last\_seen, device.org/location/department, job.created\_at, job.status; GIN on tags.

# 5. API Surface (OpenAPI‑Driven)

Authentication: JWT (short TTL) with refresh; all endpoints require Authorization unless explicitly public.

Common patterns: pagination (cursor, page\_size), filtering (RHS params), sorting (sort=field:asc|desc), ETags for cache, idempotency keys on mutation.

Error model: { code:string, message:string, details?:object, trace\_id:string }.

Routes (representative; exact schemas in OpenAPI):

• GET /health

• POST /auth/login

• POST /auth/refresh

• GET /me

• GET /tenants

• POST /tenants

• GET /tenants/{id}

• GET /devices

• GET /devices/{deviceId}

• POST /devices:search

• GET /agents/{deviceId}

• POST /automation/runs

• GET /automation/runs

• GET /automation/runs/{runId}

• POST /scripts

• GET /scripts

• POST /patch/actions

• GET /patch/posture

• GET /software

• DELETE /software/{id}:uninstall

• GET /reports

• POST /reports/schedule

• GET /audit

• GET /ws (Upgrades to WebSocket; see Events)

# 6. Realtime Events & Streams

Channel: /ws with JWT auth. Multiplex topics (JSON messages with type field):

• device.status.changed

• job.run.updated (per‑device progress, stdout/stderr frames)

• alert.raised / alert.resolved

• agent.connected / agent.disconnected

Backpressure: bounded buffers, drop oldest non‑critical frames; reconnect with exponential backoff.

# 7. Jobs, Queues, and Workers

• Message broker: NATS or RabbitMQ. Queue per job type with tenant isolation keys. Dead‑letter queues for failed items.

• Job lifecycle: queued → dispatched → running → succeeded/failed/cancelled. Retries with exponential backoff; max\_attempts per policy.

• Concurrency control: per‑tenant and global limits; throttle/jitter; maintenance windows honored.

# 8. Agents (Windows/macOS/Linux)

• Secure channel: outbound‑only HTTPS with mTLS (device cert). Device enrolls using short‑lived bootstrap token; rotates keys on schedule.

• Capabilities: heartbeat, facts collection, metrics sampling, remote exec, file ops, service/process, event logs, patch actions.

• Packaging: Windows MSI + service; Linux .deb/.rpm + systemd; macOS pkg + launchd. Installers branded per tenant; offline bootstrap supported.

• Update strategy: staged rollout; agent self‑update with signature verification.

# 9. Artifact Storage

• Object storage: S3 or MinIO. Buckets by tenant. Server‑side encryption. Presigned URLs for download. Size quotas per tenant.

• Artifacts include: job logs, script outputs, report PDFs, exports, evidence bundles.

# 10. Security & Compliance

• CSP, secure cookies, no secrets in client, rate limiting, IP allowlists optional per tenant.

• Input validation (Zod/DTOs), output escaping, command execution sandboxes, path traversal protections on file ops.

• Secrets management: parameters are referenced keys; injection occurs server‑side only; never returned to client.

# 11. Performance Targets & SLOs

• API p95 < 250ms for common reads; list pagination costs O(1) per page via cursor.

• Device filter resolve for 50k devices < 1.5s p95 (with proper indexes/caching).

• Enqueue 10k‑device run < 5s; stream progress under 1s lag at p95.

• Uptime SLO: 99.9% for API; data loss SLO: zero for durable artifacts.

# 12. Observability

• Structured logs (trace\_id, tenant\_id, user\_id).

• Metrics: request rate/latency, queue depth, job success rate, WS clients, DB health.

• Traces: API→worker→artifact pipeline spans; sampling with tail‑based retention for errors.

# 13. Environments & CI/CD

• Envs: dev (local compose), staging (seeded), prod.

• CI: lint, unit, integration (DB/broker), contract tests, container build, SBOM, vulnerability scan.

• CD: canary deploy; migration gating; health‑check rollbacks.

# 14. Testing Strategy

• Unit (services, validators), integration (DB, broker), e2e (critical flows), contract (OpenAPI), load tests (k6).

• Test data: factories/fixtures per tenant; time‑travel for schedule logic.

# 15. Migrations & API Versioning

• DB migrations (Alembic/Prisma/TypeORM). Backward‑compatible rollouts with online schema changes where possible.

• API versioning: /v1 path; additive changes preferred; deprecate with headers and changelog entries.

# 16. OpenAPI & Client SDK

• Authoritative OpenAPI 3.1 spec in repo. CI validates schemas; generates TS client `/sdk` consumed by frontend.

• Lint rules: no `any` types, explicit enums, error payloads standardized, pagination/filters consistent.

# 17. Standard Error Codes

• AUTH\_INVALID\_CREDENTIALS

• AUTH\_TOKEN\_EXPIRED

• TENANT\_FORBIDDEN

• VALIDATION\_FAILED

• RESOURCE\_NOT\_FOUND

• CONFLICT

• RATE\_LIMITED

• INTERNAL\_ERROR

# 18. Rate Limits & Quotas

• Global and per‑tenant limits; sliding window with headers (X‑RateLimit‑Remaining, X‑RateLimit‑Reset).

• Quotas on artifact storage and concurrent jobs; configurable per plan/policy.

# 19. Deployment Profiles

• Demo (single‑node compose), Small (API+worker+WS, single DB/broker), Medium (HA DB/broker, multiple workers), Large (sharded jobs, autoscale).

# 20. Appendix — Abbreviated Schemas

Device: { id, tenant\_id, hostname, os, status, last\_seen, tags[], facts{}, metrics{} }

Job: { id, tenant\_id, type, targets[], params{}, schedule?, status, created\_at, updated\_at }

RunEvent: { run\_id, device\_id, ts, phase, message, stream: stdout|stderr }