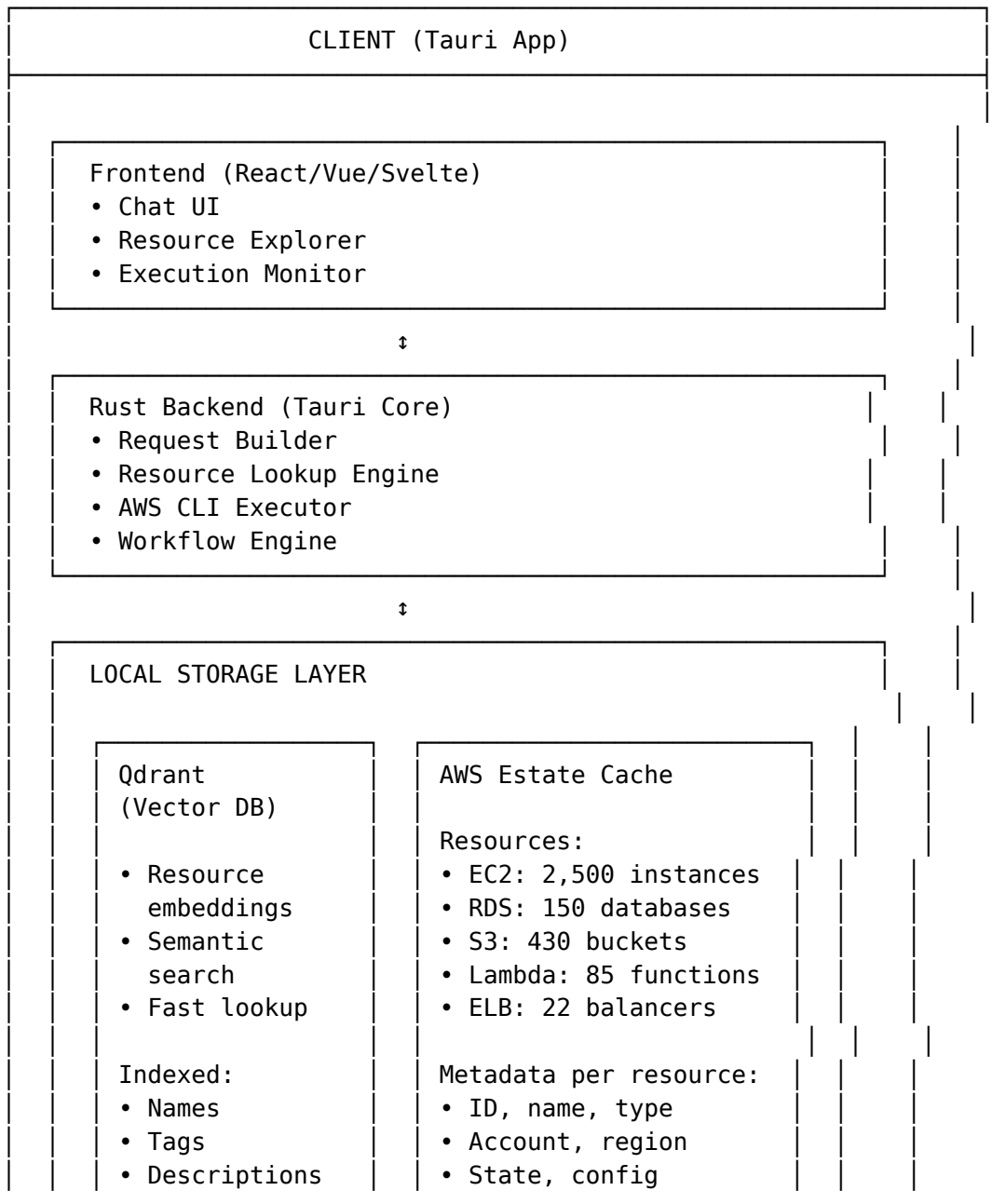
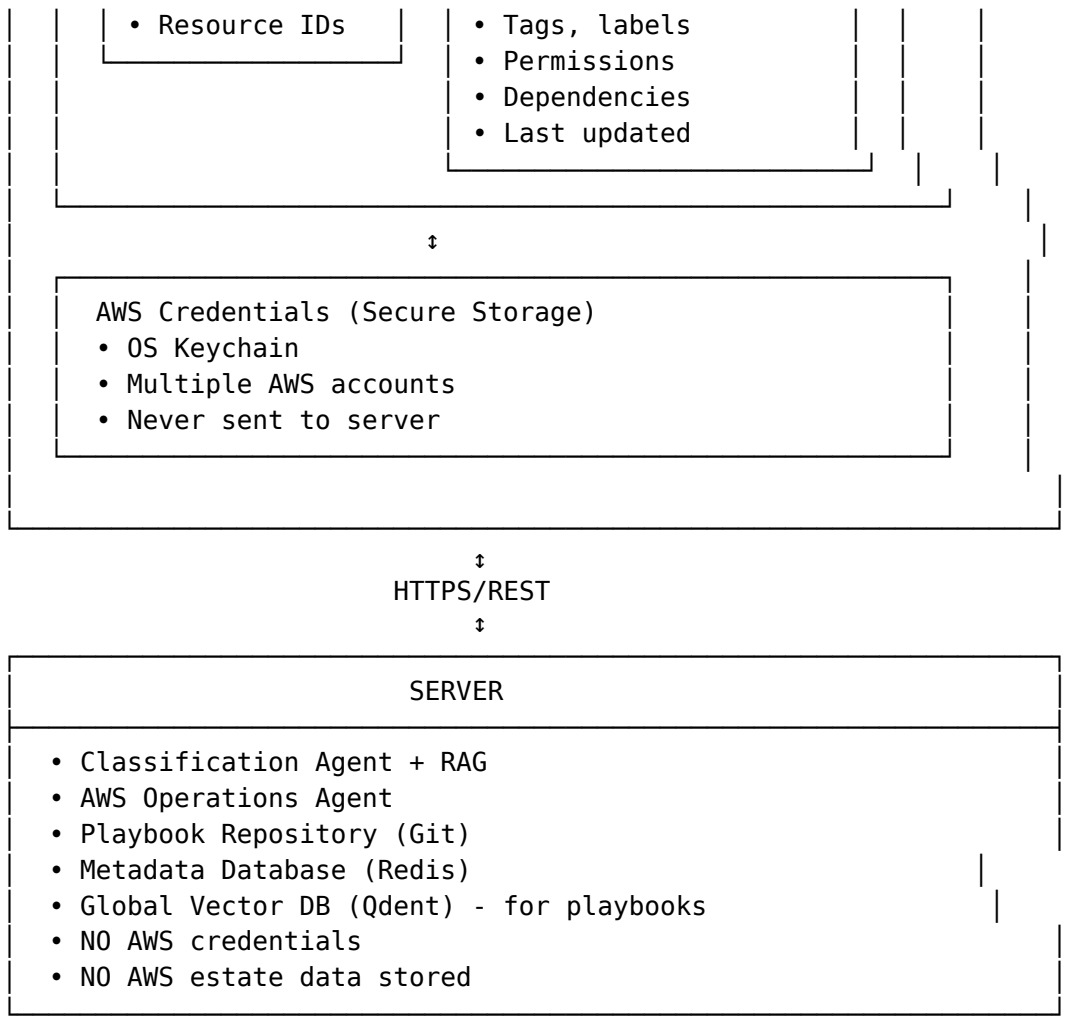
AWS CloudOps AI Agent - Architecture with Tauri + Local AWS Estate

Updated Architecture Overview





Key Architecture Changes

1. Tauri Benefits

Aspect	Electron	Tauri	Impact
Size	~100MB	~3-5MB	95% smaller
Performance	Node.js	Rust	Faster execution
Memory	~100-200MB	~30-50MB	70% less RAM

Aspect	Electron	Tauri	Impact
Security	JS isolation	Rust + OS sandbox	More secure
Local DB	Better Node support	Better Rust support	Qdrant native

2. Local Qdrant Vector Database

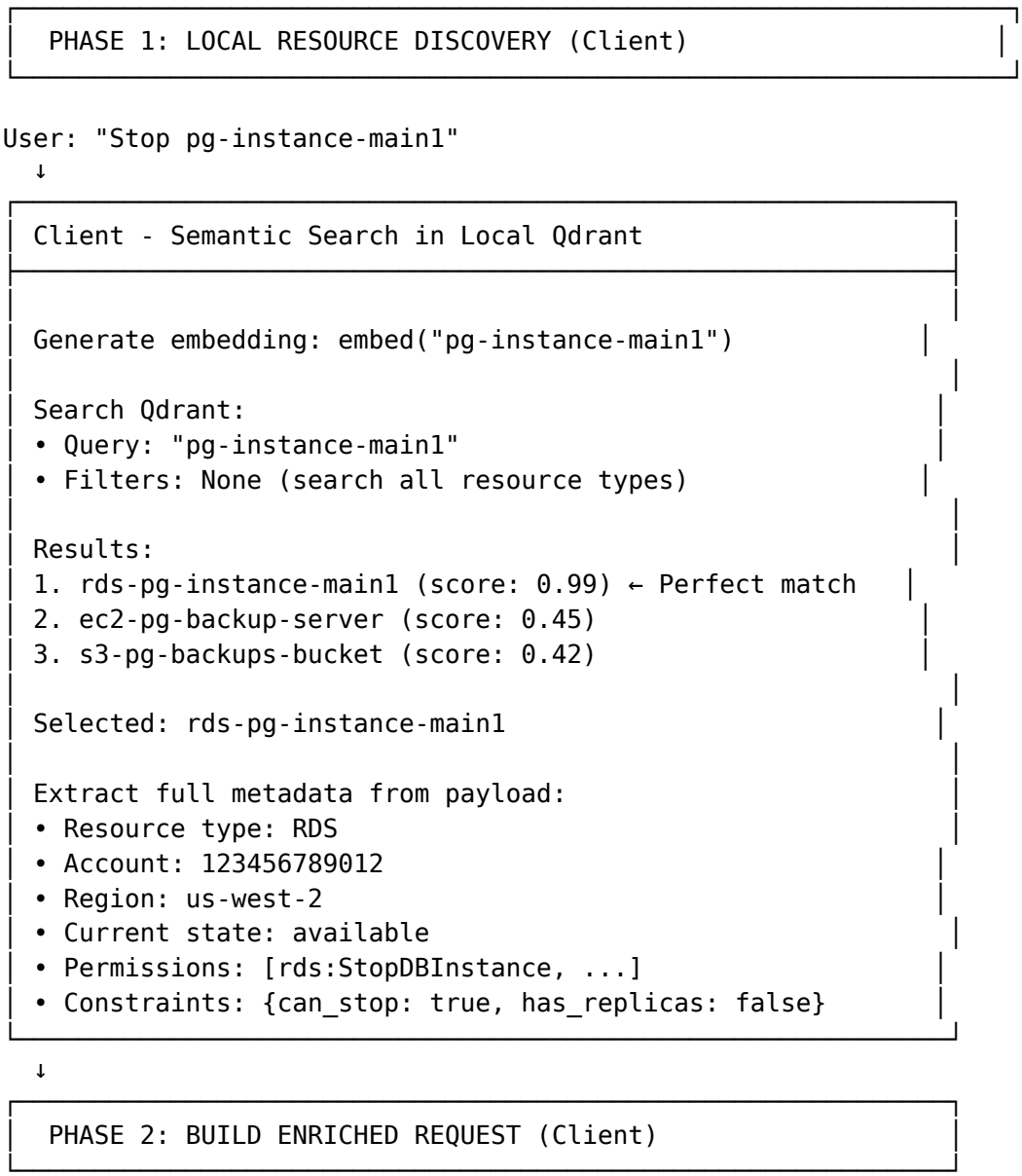
Purpose: Fast semantic search over AWS resources

Why Qdrant locally? - Semantic search: “production database” → finds pg-instance-main1 - Tag matching: “env=prod AND app=main” → finds related resources - Name variations: “pg-main1” → finds “pg-instance-main1” - Offline capability: Works without internet - Privacy: AWS estate never leaves client

Architecture:

```
Local Qdrant Collections:

Collection: "aws_resources"
├─ Vectors: Embeddings of resource metadata
├─ Payload: Full resource details
└─ Index: Fast similarity search

Example Vector:
{
  "id": "rds-pg-instance-main1",
  "vector": [0.123, 0.456, ...],  // Embedding
  "payload": {
    "resource_type": "rds_instance",
    "db_instance_identifier": "pg-instance-main1",
    "account": "123456789012",
    "region": "us-west-2",
    "engine": "postgres",
    "state": "available",
    "tags": {"env": "production", "app": "main"},
    "permissions": ["rds:StopDBInstance", "rds:CreateDBSnapshot"],
    "constraints": {...}
  }
}
```

Enhanced Request Flow Architecture



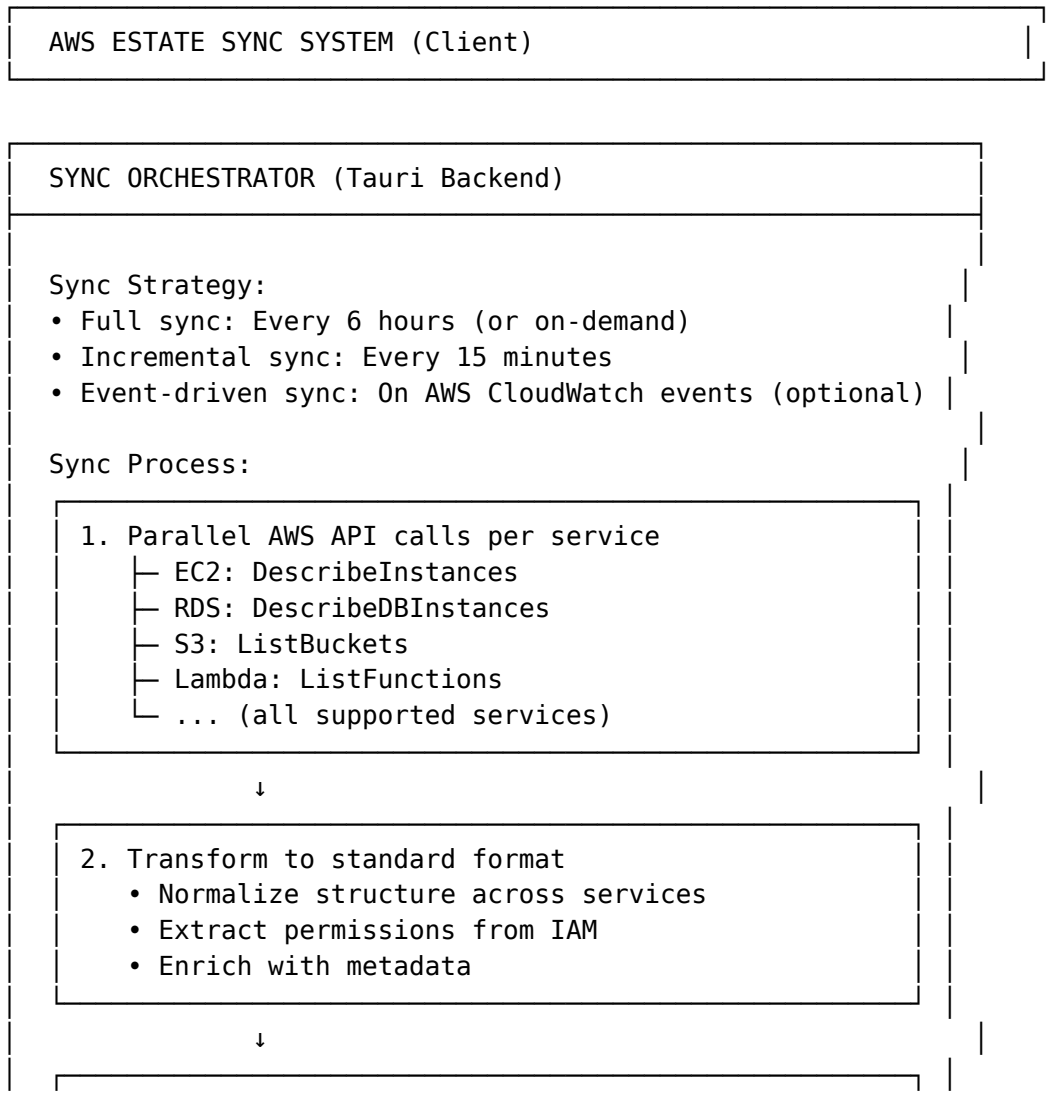
```
  "prompt": "Stop pg-instance-main1",

  "identified_resources": [
    {
      "resource_type": "rds_instance",
      "db_instance_identifier": "pg-instance-main1",
      "account_id": "123456789012",
      "region": "us-west-2",
      "current_state": "available",
      "engine": "postgres",
      "engine_version": "14.7",
      "multi_az": true,
      "tags": {"environment": "production"},
      "available_permissions": ["rds:StopDBInstance", ...],
      "constraints": {
        "can_stop": true,
        "has_read_replicas": false,
        "automated_backups_enabled": true
      }
    }
  ],

  "user_context": {
    "user_id": "user-123",
    "default_region": "us-west-2",
    "aws_accounts": ["123456789012"]
  }
}
  ↓
Send to Server
  ↓
┌──────────────────────────────────────────────────────────────┐
│  PHASE 3: SERVER PROCESSING                                  │
└──────────────────────────────────────────────────────────────┘

Server receives COMPLETE context (no guessing needed)
  ↓
┌──────────────────────────────────────────────────────────────┐
│ Classification Agent                                         │
├──────────────────────────────────────────────────────────────┤
│ • Intent: "Stop RDS instance"                                │
│ • Confidence: 0.99 (resource already identified)             │
│ • System: AWS RDS                                            │
│ • Operation: Stop                                            │
```

```
│ • Route to: AWS Operations Agent                                  │
└──────────────────────────────────────────────────────────────────┘
  ↓
┌──────────────────────────────────────────────────────────────────┐
│ RAG Search (Server's Playbook Vector DB)                          │
├──────────────────────────────────────────────────────────────────┤
│ Query: "stop rds instance"                                        │
│ Filters: {system: "aws", service: "rds"}                          │
│                                                                   │
│ Results:                                                          │
│ 1. aws_rds_stop_instance (score: 0.95)                           │
│ 2. aws_rds_stop_with_snapshot (score: 0.89)                      │
│ 3. aws_rds_reboot_instance (score: 0.45)                         │
└──────────────────────────────────────────────────────────────────┘
  ↓
┌──────────────────────────────────────────────────────────────────┐
│ AWS Operations Agent                                              │
├──────────────────────────────────────────────────────────────────┤
│ Load playbook: aws_rds_stop_instance.yaml                         │
│                                                                   │
│ Parameter Resolution:                                             │
│ • db_instance_identifier: "pg-instance-main1"                     │
│   (from identified_resources)                                     │
│ • region: "us-west-2"                                             │
│   (from identified_resources)                                     │
│ • account: "123456789012"                                         │
│   (from identified_resources)                                     │
│                                                                   │
│ ALL PARAMETERS PRE-FILLED - NO AMBIGUITY                          │
│                                                                   │
│ Risk Assessment:                                                  │
│ • Production database (from tags)                                 │
│ • Currently available (can be stopped)                            │
│ • No read replicas (safe to stop)                                 │
│ • Multi-AZ enabled (consider impact)                              │
│ • Risk: Medium                                                    │
│ • Approval: Required                                              │
└──────────────────────────────────────────────────────────────────┘
  ↓
┌──────────────────────────────────────────────────────────────────┐
│  PHASE 4: SERVER RESPONSE                                         │
```

```
└──────────────────────────────────────────────────────────────────────┘

Server → Client:
{
  "explain_plan": "I will stop the RDS PostgreSQL instance
                   'pg-instance-main1' in us-west-2. This is
                   a production Multi-AZ database. The instance
                   will be unavailable but can be restarted
                   at any time.",

  "script": {
    "steps": [
      {
        "step_id": "1",
        "command": "aws rds stop-db-instance",
        "args": [
          "--db-instance-identifier", "pg-instance-main1",
          "--region", "us-west-2"
        ]
        // ALL PARAMETERS FILLED - READY TO EXECUTE
      }
    ]
  },

  "risk_assessment": {
    "level": "medium",
    "reasons": [
      "Production database",
      "Multi-AZ enabled",
      "No read replicas to affect"
    ],
    "impact": "Database will be unavailable until restarted"
  },

  "approval_required": true
}
  ↓
┌──────────────────────────────────────────────────────────────────────┐
│  PHASE 5: CLIENT EXECUTION                                           │
└──────────────────────────────────────────────────────────────────────┘

Client receives ready-to-execute script
  ↓
Display to user + Request approval
  ↓
```

```
User approves
  ↓
Execute: aws rds stop-db-instance --db-instance-identifier
         pg-instance-main1 --region us-west-2
  ↓
Show result to user
```

Data Synchronization Architecture



```
│   │ 3. Generate embeddings                                    │ │
│   │    • Name + tags + description → embedding                │ │
│   │    • Local embedding model (optional)                     │ │
│   │    • Or server embedding API                              │ │
│   └──────────────────────────────────────────────────────────┘ │
│                          ↓                                      │
│   ┌──────────────────────────────────────────────────────────┐ │
│   │ 4. Update Qdrant                                          │ │
│   │    • Upsert vectors                                       │ │
│   │    • Update payloads                                      │ │
│   │    • Remove deleted resources                             │ │
│   └──────────────────────────────────────────────────────────┘ │
│                                                                 │
│  Sync Status Tracking:                                          │
│  • Last sync timestamp per service                              │
│  • Sync errors and retries                                      │
│  • Resource count changes                                       │
└─────────────────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────────────────────┐
│  MULTI-ACCOUNT SUPPORT                                          │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  User has multiple AWS accounts:                                │
│  • Production Account (123456789012)                           │
│  • Staging Account (987654321098)                               │
│  • Development Account (456789012345)                           │
│                                                                 │
│  Qdrant Structure:                                              │
│  {                                                              │
│    "id": "rds-prod-pg-instance-main1",                          │
│    "payload": {                                                 │
│      "account_id": "123456789012",                              │
│      "account_name": "production",                              │
│      ...                                                        │
│    }                                                            │
│  }                                                              │
│                                                                 │
│  Search with account filter:                                    │
│  • Default: Search all accounts                                 │
│  • Explicit: "Stop pg-main1 in production account"              │
```

```
│  • Filter by context                                                    │
└─────────────────────────────────────────────────────────────────────────┘
```

Benefits of This Architecture

1. Precision

Without Local Estate	With Local Estate
Server guesses parameters	All parameters known upfront
"Which account?" "Which region?"	Account + region pre-filled
Ambiguous resource names	Exact resource identification
Multiple back-and-forth	Single request-response

2. Performance

Aspect	Impact
Resource lookup	Instant (local Qdrant)
No guessing	Faster server processing
Fewer LLM calls	Lower cost
Offline search	Works without internet

3. Security

Aspect	Benefit
AWS estate stays local	Never sent to server (privacy)
Credentials local	Never leave client
Fine-grained permissions	Known per-resource
Audit trail	Local execution history

4. User Experience

Feature	UX Benefit
Fuzzy search	"pg-main" finds "pg-instance-main1"
Smart suggestions	Autocomplete resource names
Visual explorer	Browse AWS estate locally
Fast response	No waiting for server lookup

Architecture Comparison

Old (Stateless Client)

```
User: "Stop pg-instance-main1"
  ↓
Client → Server: "Stop pg-instance-main1"
  ↓
Server: "Which account? Which region? Is this RDS or EC2?"
  ↓
Client ← Server: Clarifying questions
  ↓
User provides more info
  ↓
Client → Server: Updated request
  ↓
Server generates script
```

Issues: Multiple round-trips, ambiguity, slow

New (Stateful Client with Local Estate)

```
User: "Stop pg-instance-main1"
  ↓
Client searches Qdrant: Found RDS in us-west-2, account 123456
  ↓
Client → Server: Complete context (resource + metadata)
  ↓
Server generates precise script (no ambiguity)
  ↓
Client ← Server: Ready-to-execute script
  ↓
Client executes immediately
```

Benefits: Single round-trip, precise, fast

Key Architecture Decisions

Decision	Rationale
Tauri over Electron	Smaller, faster, more secure, better for Qdrant
Qdrant local vector DB	Semantic search, fuzzy matching, offline capability

Decision	Rationale
AWS estate on client	Privacy, speed, precision, offline mode
Enriched context to server	Server gets exact parameters, no guessing
Server stays stateless	Scalable, no estate data storage
Periodic sync	Fresh data without constant API calls

Component Breakdown

Client Components

1. **Sync Engine** - Pulls AWS estate periodically
2. **Qdrant Manager** - Indexes and searches resources
3. **Resource Lookup** - Semantic search and exact matching
4. **Context Builder** - Packages complete resource context
5. **Script Executor** - Runs AWS CLI commands
6. **Workflow Engine** - Handles multi-step execution

Server Components

1. **Classification Agent** - Routes requests (now easier with context)
2. **AWS Operations Agent** - Generates scripts (now with exact params)
3. **Playbook Repository** - YAML playbooks in Git
4. **Metadata DB** - Playbook metadata for RAG
5. **Vector DB** - Playbook embeddings (separate from client's resource DB)

Summary

This architecture gives you:

 Fast - Local semantic search in Qdrant
 Precise - Server receives exact resource details
 Private - AWS estate never leaves client
 Offline-capable - Search works without internet
 Smart - Fuzzy matching, tag-based search
 Scalable - Server remains stateless
 Secure - Credentials and estate stay local

The key insight: **Client knows everything about AWS resources, server knows everything about operations**. Together they generate perfect execution scripts.