Opulent Horizons MCP Gateway

Complete Technical Overview — Generated 2026-02-11 | Repository: `MCP-Gateway-Claude-Desktop` | Version 1.0.0

Contents

1. Purpose
2. Architecture
3. Tech Stack
4. All MCP Tools & Resources (8 tools)
5. Connectors & Integrations
6. Workflows
7. Database Schema
8. Observability
9. Deployment
10. Environment Variables
11. Testing
12. Project Structure
13. Version History

1. Purpose

A production-ready **Model Context Protocol (MCP)** gateway for **real estate lead management**. Two standalone MCP servers let Claude Desktop, Claude Code, or any MCP-compatible AI agent ingest leads from multiple channels and synchronise them bidirectionally with Zoho CRM — all through structured tool calls over **stdio** or **HTTP**.

Migrated from a monolithic FastAPI/JSON-RPC app to the official Anthropic MCP Python SDK (`v1.26.0+`), eliminating ~80 lines of hand-rolled protocol code and gaining native multi-transport support.

2. Architecture

```
Claude Desktop / Claude Code / Remote Agent
         │ stdio              │ stdio
┌──────────────────┐  ┌────────────────────┐
│  Lead Ingest       │  │  Zoho CRM Sync       │
│  :8001  (5 tools)  │  │  :8002  (3 tools)    │
└──────────────────┘  └────────────────────┘
   │      │      │          │          │
   ↓      │      ↓          ↓          ↓
PostgreSQL   │   Webhook     Zoho CRM      OAuth2
(asyncpg)    │  (n8n/custom) REST API    Token Refresh
lead_context │               /Leads
workflow_evt │
             ↓
      OHID Resolution
      (email/phone dedup)
```

Shared layer (`shared/`): Pydantic domain models, repository abstraction (Postgres + in-memory), structured logging middleware with correlation IDs, Zoho OAuth2 token manager.

3. Tech Stack

Layer	Technology
Language	Python 3.11+ (1,978 LOC across 13 source files)
Protocol	Anthropic MCP SDK `>=1.26.0` / FastMCP
Validation	Pydantic 2.0+
Database	PostgreSQL 16 via `asyncpg >=0.30` (direct pool, no ORM)
HTTP Client	`httpx >=0.27` (async)
Retry	`tenacity >=9.0`
Container	Docker (`python:3.12-slim`) / Docker Compose
Cloud	Azure Container Apps (Bicep IaC, 0–3 replica autoscale)
CI/CD	GitHub Actions (build → ACR push → Bicep deploy → pytest)
Testing	pytest + pytest-asyncio (unit, stdio integration, live e2e)
Linting	ruff (100-char line, py311 target)
Transports	stdio, Streamable HTTP, SSE

4. All MCP Tools & Resources

4.1 Lead Ingest Server — `servers/lead_ingest.py` 5 tools 1 resource

#	Tool	Parameters	Behavior
1	`ingest_lead`	`source_system` , `source_lead_id` , `channel` , `first_name` , `last_name` , `marketing_consent` , optional: `email` , `phone` , `budget_range` , `location` , `property_type` , `free_text` , `consent_source` , `raw_payload`	Builds `LeadIngestRequest` (Pydantic-validated). Calls `resolve_ohid()` — looks up existing OHID by email/phone or generates new UUID. Inserts into `lead_context` + creates `LeadIngested` workflow event. Publishes event to webhook. Returns `{ohid, ingest_id, source_system, status: "ingested"}` .
2	`process_cloudtalk_event`	`event_type` , `call_id` , `direction` , `from_number` , `to_number` , optional: `recording_url` , `raw`	Maps `call.started` / `call.ringing` → `CallReceived` , anything else → `CallCompleted` . Inserts workflow event (ohid=None). Publishes to webhook. Returns `{event_id, event_type, accepted: true}` .
3	`process_notion_event`	`payload` (dict)	If payload has `"challenge"` key → returns challenge (Notion webhook verification handshake). Otherwise inserts `NotionEvent` workflow event, publishes to webhook. Returns `{event_id, accepted: true}` .
4	`lookup_ohid`	optional: `email` , `phone` (at least one required)	Queries `lead_context` table via `find_ohid_by_contact()` . Returns `{ohid, found: true}` or `{found: false}` .

5	`verify_webhook_signature`	`body_hex` , `signature` , `source` (default: "cloudtalk")	**CloudTalk**: HMAC-SHA256 with `CLOUDTALK_WEBHOOK_SECRET` , compares hexdigest. **Notion**: HMAC-SHA256 with `NOTION_WEBHOOK_SECRET` , strips `sha256=` prefix. Returns `{valid: bool}` .

Resource: `status://pipeline` — Returns server config: sources, channels, DB type (postgres/in-memory), webhook availability.

4.2 Zoho CRM Sync Server — `servers/zoho_crm_sync.py` 3 tools 1 resource

#	Tool	Parameters	Behavior
1	`sync_lead`	`zoho_lead_id` , `sync_direction` (inbound|outbound|bidirectional), optional: `source` , `property_db_lead_id` (required for outbound/bidirectional)	Validates via `ZohoCRMSyncRequest` (field validator enforces `property_db_lead_id` for outbound). **Inbound**: `GET /Leads/{id}` . **Outbound**: `POST /Leads/upsert` with Email dedup. Status: both OK → "success", mixed → "partial", both fail → "failed". Returns `ZohoCRMSyncResponse` with per-direction flags + `execution_time_ms` .
2	`get_zoho_lead`	`lead_id`	`GET /Leads/{id}` with OAuth token. Returns `{found, lead}` or `{found: false, error}` .
3	`upsert_zoho_lead`	`last_name` , optional: `email` , `phone` , `first_name` , `company` , `lead_source` , `source_attribution`	Builds lead dict, `POST /Leads/upsert` with Email dedup. If `source_attribution` provided, sets `Description` . Returns `{success, zoho_lead_id, action}` .

Resource: `status://zoho-sync` — Returns auth mode (oauth2_refresh / static_token / not_configured), API base URL, supported sync directions.

5. Connectors & Integrations

5.1 Inbound Lead Sources (via `ingest_lead`)

Source System	Channel	Example
`META`	`META_LEAD_AD`	Facebook / Instagram lead ad form
`WEB`	`WEB_FORM`	Website contact form
`CLOUDTALK`	`INBOUND_CALL` / `OUTBOUND_CALL`	VoIP telephony
`ZOHO_CRM`	`CRM`	CRM-originated lead
`ZOHO_SOCIAL`	`SOCIAL`	Social media engagement

Pattern-validated at the Pydantic model level — any value outside these enums is rejected before reaching the database.

5.2 CloudTalk Telephony Connector

- Accepts `CloudtalkWebhookPayload` events (`call.started` , `call.ringing` , `call.completed` , etc.)
- Signature verification via HMAC-SHA256 (`CLOUDTALK_WEBHOOK_SECRET`)
- Events stored as workflow events; not yet linked to OHID (linkage happens downstream)

5.3 Notion Database Connector

- Handles Notion webhook challenge-response handshake

- Accepts page/database events, stores as `NotionEvent` workflow events
- Signature verification via HMAC-SHA256 (`NOTION_WEBHOOK_SECRET` , `sha256=` prefix format)

5.4 Zoho CRM Connector

Direction	HTTP Call	Behavior
Inbound (Zoho → local)	`GET {ZOHO_API_BASE}/Leads/{id}`	Fetch lead data
Outbound (local → Zoho)	`POST {ZOHO_API_BASE}/Leads/upsert`	Create/update with Email dedup
Bidirectional	Both calls in sequence	Returns per-direction success flags

Auth: `ZohoTokenManager` — OAuth2 with automatic refresh 5 minutes before expiry, `asyncio.Lock` to prevent concurrent refresh races. Supports global / AU / EU / IN data centres via `ZOHO_TOKEN_URL` . Falls back to static `ZOHO_ACCESS_TOKEN` if OAuth credentials not provided.

5.5 Outbound Webhook Publisher

After every tool call that creates a lead or event, a non-blocking `POST` fires to `WORKFLOW_WEBHOOK_URL` or `N8N_WEBHOOK_URL` (whichever is set). Payload: `{event_type, ...event_data}` . Errors are silently swallowed — the tool call succeeds regardless.

6. Workflows

6.1 Full Lead Lifecycle

```
External source (META ad / web form / phone call)
  → Agent calls ingest_lead(source="META", channel="META_LEAD_AD", ...)
    → Pydantic validates source_system + channel patterns
    → resolve_ohid() : SELECT from lead_context by email/phone
      → Found? Reuse existing OHID
      → Not found? Generate new UUID
    → INSERT into lead_context (JSONB payload + consent)
    → INSERT "LeadIngested" into workflow_event
    → POST to webhook (n8n triggers email sequence / Slack alert)
  → Return {ohid, ingest_id, status: "ingested"}
```

6.2 CloudTalk Call Processing

```
CloudTalk webhook fires
  → Agent calls process_cloudtalk_event(event_type="call.completed", ...)
    → Map: call.started/call.ringing → "CallReceived", else → "CallCompleted"
    → INSERT into workflow_event (ohid=None)
    → POST to webhook
  → Return {event_id, event_type: "CallCompleted", accepted: true}
```

6.3 Notion Webhook Handling

```
Handshake:
  → process_notion_event(payload={challenge: "abc"})
  → Return {challenge: "abc"}

Event:
  → process_notion_event(payload={id: "...", type: "page.created"})
    → INSERT "NotionEvent" into workflow_event
```

  → POST to webhook
→ Return `{event_id, accepted: true}`

6.4 Zoho Bidirectional Sync

Agent calls `sync_lead(zoho_lead_id="123", direction="bidirectional", property_db_lead_id="456")`
→ Pydantic validates (property_db_lead_id required for outbound/bidirectional)
→ INBOUND: `GET /Leads/123` → inbound_ok = true/false
→ OUTBOUND: `POST /Leads/upsert` {Last_Name, Lead_Source, External_ID__c} → outbound_ok = true/false
→ Status: both ok → "success", one ok → "partial", neither → "failed"
→ Return `ZohoCRMSyncResponse {status, inbound_success, outbound_success, execution_time_ms}`

6.5 Direct Zoho Operations

`upsert_zoho_lead(last_name="Smith", email="j@co.com", lead_source="Website")`
→ POST /Leads/upsert with Email dedup
→ Return `{success: true, zoho_lead_id: "874...", action: "insert"}`

`get_zoho_lead(lead_id="874...")`
→ GET /Leads/874...
→ Return `{found: true, lead: {...Full_Name, Email, Phone, ...}}`

7. Database Schema

`lead_context` — every ingested lead

Column	Type	Notes
`id`	UUID PK	ingest_id
`ohid`	UUID NOT NULL	Opulent Horizons ID (deduped by email/phone)
`source_system`	VARCHAR(50)	META, WEB, CLOUDTALK, etc.
`source_lead_id`	VARCHAR(255)	External system's ID
`channel`	VARCHAR(50)	WEB_FORM, META_LEAD_AD, etc.
`payload`	JSONB	Full `LeadIngestRequest`
`consent`	JSONB	`Consent` object
`created_at`	TIMESTAMPTZ	Default `now()`

Indexes: `ohid`, `payload->'person'->>'email'` (partial), `payload->'person'->>'phone'` (partial)

`workflow_event` — event log for lead lifecycle

Column	Type	Notes
`id`	UUID PK	event_id
`ohid`	UUID (nullable)	NULL for leadless events (CloudTalk, Notion)
`event_type`	VARCHAR(100)	LeadIngested, CallCompleted, NotionEvent, etc.
`payload`	JSONB	Event-specific data
`occurred_at`	TIMESTAMPTZ	Default `now()`

`source_system`	VARCHAR(50)	Origin system

Indexes: `ohid` (partial, WHERE NOT NULL), `event_type`

8. Observability

Feature	Implementation	Detail
Correlation IDs	`shared/middleware.py`	Per-tool-call UUID propagated via `ContextVar` across async boundaries
Structured audit logging	`wrap_tool_with_logging()`	Wraps every tool with `tool.start` / `tool.end` / `tool.error` events including `duration_ms` , `correlation_id` , error type — JSON to stderr
Health resources	MCP resources	`status://pipeline` and `status://zoho-sync` expose runtime config and auth state
Docker health checks	`docker-compose.yml`	HTTP probes on `/mcp` endpoint for both servers

9. Deployment

Method	Command	What Runs
stdio (Claude Desktop)	`python -m servers.lead_ingest`	Single server, in-memory DB
HTTP (production)	`python -m servers.lead_ingest --transport streamable-http --port 8001`	HTTP on port 8001
Docker	`docker build -t opulent-mcp . && docker run -p 8001:8001 --env-file .env opulent-mcp`	Configurable via `MCP_SERVER` env var
Docker Compose	`docker compose up`	Postgres:5432 + Lead Ingest:8001 + Zoho Sync:8002
Azure	`az deployment group create --template-file infra/main.bicep`	Container Apps (0–3 replicas, HTTP autoscale at 10 concurrent reqs)
CI/CD	Push to `main`	GitHub Actions: build → ACR push → Bicep deploy → pytest

Azure Infrastructure (Bicep)

- **Azure Container Registry** (Basic SKU, admin enabled)
- **Container Apps Environment** (`opulent-mcp-{env}-env`)
- **Lead Ingest Container App** — 0.25 CPU, 0.5 GiB, external ingress on :8001
- **Zoho Sync Container App** — 0.25 CPU, 0.5 GiB, external ingress on :8002, Zoho secrets injected
- Both apps scale 0–3 replicas on HTTP concurrency (threshold: 10)

10. Environment Variables (Complete)

Variable	Server	Purpose
`PGHOST` , `PGPORT` , `PGUSER` , `PGPASSWORD` , `PGDATABASE`	Lead Ingest	PostgreSQL connection (absent = in-memory fallback)

`ZOHO_API_BASE`	Zoho Sync	CRM API endpoint (regional, e.g. zohoapis.com.au)
`ZOHO_TOKEN_URL`	Zoho Sync	OAuth2 token endpoint (regional)
`ZOHO_CLIENT_ID`	Zoho Sync	OAuth2 client ID
`ZOHO_CLIENT_SECRET`	Zoho Sync	OAuth2 client secret
`ZOHO_REFRESH_TOKEN`	Zoho Sync	OAuth2 refresh token (scope: ZohoCRM.modules.ALL)
`ZOHO_ACCESS_TOKEN`	Zoho Sync	Legacy static token fallback (~1hr expiry)
`PROPERTY_DB_HOST/PORT/USER/PASSWORD/NAME`	Zoho Sync	Property database DSN
`WORKFLOW_WEBHOOK_URL`	Lead Ingest	Downstream event webhook (n8n / Temporal)
`N8N_WEBHOOK_URL`	Lead Ingest	Alternative webhook URL for n8n
`CLOUDTALK_WEBHOOK_SECRET`	Lead Ingest	HMAC-SHA256 verification for CloudTalk
`NOTION_WEBHOOK_SECRET`	Lead Ingest	HMAC-SHA256 verification for Notion
`MCP_SERVER`	Docker	Which server module to start
`MCP_PORT`	Docker	Port override (default 8001/8002)

11. Testing

Layer	File	Count	What It Covers
Unit	`test_lead_ingest.py`	10	Server name, tool count, tool names, HMAC verification (5 scenarios for CloudTalk & Notion)
Unit	`test_zoho_sync.py`	3	Server name, tool count, tool names
Integration (stdio)	`test_lead_ingest.py`	5	`ingest_lead`, `process_cloudtalk_event` (2 event types), `process_notion_event`, `lookup_ohid` — InMemoryRepository
Integration (stdio)	`test_zoho_sync.py`	3	`get_zoho_lead`, `upsert_zoho_lead`, `sync_lead` — empty token guards
E2E (live API)	`e2e_zoho_live.py`	3	Real Zoho CRM: get lead, upsert lead, inbound sync (requires credentials)

Run: `pytest tests/ -v --ignore=tests/e2e_zoho_live.py`

12. Project Structure

```
MCP-Gateway-Claude-Desktop/
├─ servers/
│  ├─ lead_ingest.py          # Lead ingestion MCP server (5 tools, 1 resource)
│  └─ zoho_crm_sync.py        # Zoho CRM sync MCP server (3 tools, 1 resource)
├─ shared/
│  ├─ models.py               # Pydantic v2 domain models (7 models)
│  ├─ repository.py           # Repository abstraction (Postgres + in-memory)
│  ├─ middleware.py           # Correlation IDs + structured audit logging
│   ├─ zoho_auth.py            # OAuth2 token manager with auto-refresh
│   └─ schema.sql              # PostgreSQL DDL (2 tables, 5 indexes)
├─ tests/
│   ├─ test_lead_ingest.py     # Unit + integration tests (15 tests)
│   ├─ test_zoho_sync.py       # Unit + integration tests (6 tests)
│   └─ e2e_zoho_live.py        # Live Zoho CRM e2e tests (3 tests)
├─ infra/
│   └─ main.bicep              # Azure Container Apps IaC (238 lines)
├─ .github/workflows/
│   └─ deploy.yml              # CI/CD: build + deploy + test
├─ docs/
│   ├─ CHANGELOG.md            # Version history (v0.2.0 → v1.0.0)
│   └─ DEPLOYMENT_LOG.md       # Operational deployment & bug fix records
├─ pyproject.toml                   # Dependencies & build config
├─ Dockerfile                       # python:3.12-slim production image
├─ docker-compose.yml               # Full stack: Postgres + 2 servers
├─ claude_desktop_config.json       # Drop-in config for Claude Desktop (Windows)
└─ .env.example                     # Environment variable template
```

13. Version History

Version	Date	Summary
v1.0.0	2026-02-10	Full migration from FastAPI/JSON-RPC to MCP SDK. Two servers, 8 tools, asyncpg, OAuth2 auto-refresh, Docker, Azure Bicep, GitHub Actions CI/CD. Removed FastAPI, Uvicorn, SQLAlchemy, hand-rolled JSON-RPC dispatcher.
v0.2.0	2025-12-01	Legacy FastAPI version (archived, superseded).

Known Bugs Fixed (v1.0.0)

#	Severity	Issue	Fix
1	CRITICAL	`from __future__ import annotations` breaks MCP SDK tool schema introspection (PEP 563 stringifies types)	Removed from both server files + added warning comments
2	MEDIUM	`pyproject.toml` pinned `mcp>=1.9.0` but validated against v1.26.0	Updated to `mcp>=1.26.0`
3	MEDIUM	Wrong import path `mcp.server.mcpserver.MCPServer`	Changed to `mcp.server.fastmcp.FastMCP`
4	LOW	Invalid `version=` constructor parameter	Removed from FastMCP constructor
5	LOW	`run()` receiving host/port as args	Moved to `mcp.settings`