MCP Vacuum Project Overview

MCP Vacuum Project Overview

Project Vision

MCP Vacuum is a Google Python ADK agent that automatically discovers MCP servers on LAN, authenticates dynamically using OAuth 2.1, and converts configurations to Kagent-compatible format.

Technical Goals

Network Discovery

Multi-protocol discovery supporting mDNS, SSDP, and ARP scanning with concurrent execution patterns

Authentication

OAuth 2.1 + PKCE with dynamic client registration following RFC 7636 security standards

Schema Conversion

Bidirectional MCP ↔ Kagent format mapping with semantic preservation and validation

ADK Integration

Production-ready agent with Vertex AI deployment and hierarchical agent composition

Scalability

Concurrent discovery with connection pooling and resource management

Architecture Components

1. Discovery Engine

Multi-protocol network service detection with support for:

- **mDNS/DNS-SD**: Service discovery with `_mcp._tcp.local` type
- **SSDP/UPnP**: Windows compatibility for enterprise networks
- **ARP Scanning**: Layer 2 enumeration for comprehensive coverage
- **Custom Registry**: Centralized service registration with health checks

2. MCP Client

Full JSON-RPC 2.0 implementation with authentication:

- Type-safe protocol implementation using Pydantic V2
- Multiple transport support (STDIO, HTTP, WebSocket, SSE)
- Connection pooling with aiohttp.TCPConnector
- Comprehensive error handling with exponential backoff

3. Schema Mapper

Intelligent conversion with validation:

- Bidirectional MCP ↔ Kagent transformation

- JSON Schema Draft 7 to Kubernetes CRD mapping
- Field name sanitization for K8s compliance
- Semantic preservation with 80%+ field overlap validation

4. ADK Agents

Hierarchical agent system for orchestration:

- **OrchestrationAgent**: Parent coordinator managing child agents
- **DiscoveryAgent**: Network scanning and service detection
- **AuthenticationAgent**: OAuth 2.1 credential management
- **ConversionAgent**: Schema transformation and validation

5. Production Runtime

Vertex AI integration with monitoring:

- Google Cloud deployment with auto-scaling
- Prometheus metrics and structured logging
- Circuit breaker patterns for resilience
- Comprehensive observability stack

Implementation Epics

🏗️ Foundation & Infrastructure (P0)

- Project setup with UV package management
- Core data models with Pydantic V2
- Development environment with DevContainer
- CI/CD pipeline with GitHub Actions

🌐 MCP Protocol Integration (P0)

- JSON-RPC 2.0 client implementation
- Tool invocation framework with validation
- MCP v1.0+ specification compliance
- Connection lifecycle management

🔐 Authentication & Security (P0)

- OAuth 2.1 + PKCE implementation
- Secure credential storage with encryption
- Dynamic client registration (RFC 7591)
- Token management with auto-refresh

🤖 Google ADK Integration (P0)

- BaseAgent framework implementation
- Vertex AI runtime configuration
- Event-driven agent communication
- Production deployment patterns

🕸️ Network Discovery Engine (P1) 🔗

- mDNS discovery with python-zeroconf
- SSDP discovery for Windows compatibility
- Concurrent scanning with resource limits
- Discovery caching with TTL management

🔄 Schema Conversion Engine (P1) 🔗

- MCP to Kagent format conversion
- Kagent to MCP reverse conversion
- Validation pipeline with detailed reporting
- Tool categorization and risk assessment

🧪 Testing & Quality Assurance (P1) 🔗

- pytest framework with asyncio support
- Integration tests with mock MCP servers
- Property-based testing with Hypothesis
- Performance benchmarks with regression detection

🚀 Production Deployment (P2) 🔗

- Kubernetes deployment manifests
- Monitoring and alerting setup
- Docker containerization
- Infrastructure as Code with Terraform

Success Criteria 🔗

Functional Requirements 🔗

- ✅ Automatic discovery of MCP servers on local network
- ✅ Secure authentication with modern OAuth flows
- ✅ Accurate schema conversion maintaining semantic integrity
- ✅ Production deployment on Google Cloud with monitoring
- ✅ Comprehensive test coverage and documentation

Performance Targets 🔗

- **Network Scan**: 100+ hosts in under 30 seconds
- **mDNS Discovery**: 50+ services in under 10 seconds
- **Memory Usage**: Under 50MB for typical home network
- **CPU Usage**: Under 20% during active discovery
- **Schema Conversion**: Under 100ms for typical configurations

Quality Standards 🔗

- **Code Coverage**: 95%+ for core logic, 85%+ overall
- **Type Safety**: Full mypy strict mode compliance
- **Performance**: Benchmarks within acceptable ranges
- **Security**: OAuth 2.1 compliance with PKCE mandatory

Timeline Estimate

MVP (P0): 3-4 weeks

Core discovery, authentication, and basic conversion functionality

Feature Complete: 6-8 weeks

Full multi-protocol discovery, comprehensive testing, advanced features

Production Ready: 10-12 weeks

Kubernetes deployment, monitoring, documentation, security audit

Technical Standards

Python Development

- Python 3.12+ with modern asyncio patterns
- UV for dependency management and virtual environments
- Pydantic V2 for data validation and serialization
- pytest with asyncio support for comprehensive testing
- Clean architecture with proper separation of concerns

Code Quality

- PEP 8 compliance with ruff formatting
- Type safety with mypy strict mode
- Comprehensive docstrings following PEP 257
- Security scanning with bandit
- Performance benchmarking with automated regression detection

DevOps Integration

- GitHub Actions CI/CD pipeline
- Docker multi-stage builds with minimal attack surface
- Kubernetes deployment with Helm charts
- Infrastructure as Code with Terraform
- Monitoring with Prometheus and structured logging

This project represents a comprehensive solution for MCP server integration with modern Python development practices, production-ready deployment patterns, and enterprise-grade security standards.