# MCP Server Implementation Guide for ServiceNow Integration

## Part 1: Introduction and Overview

---

### Document Series

This implementation guide is organized into five parts:

1. **Part 1: Introduction and Overview** (This Document)
2. Part 2: Server Foundation - Core Infrastructure Setup
3. Part 3: MCP Protocol Implementation and Tools
4. Part 4: OAuth 2.1 Implementation
5. Part 5: Appendices and Recommendations

---

### Document Purpose

This guide provides prescriptive implementation guidance for building Model Context Protocol (MCP) servers that integrate with ServiceNow environments. It follows MCP protocol standards (versions 2025-06-18 through 2025-11-25) and addresses ServiceNow-specific integration requirements.

**Target Audience:** MCP practitioners with mixed experience levels - from those new to MCP server implementation to experienced developers seeking ServiceNow integration patterns.

**Scope:** This document covers MCP protocol compliance requirements and ServiceNow integration "must haves" using cloud-agnostic implementation patterns.

**Code Approach:** Implementation guidance uses three formats:

- **Pseudocode**: Language-agnostic logic for developers using any programming language
- **JavaScript**: Rapid prototyping and Node.js implementations
- **TypeScript**: Production-grade, type-safe implementations for cloud deployments

#### Prerequisites and Assumptions:

This guide assumes practitioners have:

- Basic understanding of MCP protocol concepts and architecture
- Familiarity with OAuth 2.0/2.1 authentication fundamentals
- Experience with Node.js or TypeScript development

- Understanding of HTTP/REST API principles
- Basic knowledge of ServiceNow AI Platform capabilities
- Familiarity with command-line tools and environment variable configuration

---

## Documentation Artifacts

This implementation guide includes the following resources:

**Documentation Series (5 Parts):**

1. Part 1: Introduction and Overview (this document)
2. Part 2: Server Foundation - Core Infrastructure Setup
3. Part 3: MCP Protocol Implementation and Tools
4. Part 4: OAuth 2.1 Implementation
5. Part 5: Appendices and Recommendations

**Reference Implementations:**

- `server-compliant.js` - Local VM deployment (JavaScript, file-based storage, Redis)
- `gcp-compliant.ts` - Google Cloud deployment (TypeScript, Firestore storage)

**Template Files:**

- `server-example.js` - White-label template for customization
- `.env.example` - Environment configuration template

**Visual Aids:**

- `oauth-flow-diagram.mermaid` - OAuth 2.1 sequence diagram (Mermaid format)
- `oauth-flow-diagram.svg` - OAuth 2.1 sequence diagram (SVG format)

---

## Reference Documentation

Before proceeding, practitioners should familiarize themselves with:

**MCP Protocol Specifications:**

- Official MCP Documentation: https://modelcontextprotocol.io
- MCP GitHub Repository: https://github.com/modelcontextprotocol/sdk

- Protocol Version Coverage: 2025-06-18 through 2025-11-25

**Protocol Version Guidance:**

- **ServiceNow Currently Supports:** MCP protocol version 2025-06-18
- **This Guide Covers:** Versions 2025-06-18 through 2025-11-25 for future compatibility
- **Recommended Implementation:** Use version 2025-06-18 for ServiceNow integration
- **Version Compatibility:** Differences between versions are minimal; implementations following 2025-06-18 remain compatible with newer versions

**ServiceNow Integration:**

- ServiceNow MCP Client Documentation: https://www.servicenow.com/docs/r/intelligent-experiences/mcp-client.html
- ServiceNow AI Platform: MCP client support available in Yokohama Patch 9+ and Zurich Patch 2+
- Protocol Support: ServiceNow currently supports MCP protocol version 2025-06-18

---

## ServiceNow Integration Requirements

### Overview

ServiceNow's MCP client implementation requires specific patterns and capabilities from MCP servers. This section outlines the mandatory elements and highly recommended patterns for successful integration.

### Protocol Version

**Requirement:** ServiceNow supports MCP protocol version **2025-06-18**

**Implementation Note:** While this guide covers protocol versions through 2025-11-25, ServiceNow's current support is at 2025-06-18. Ensure your server advertises this version during the `initialize` handshake for compatibility.

### Transport Method

**Requirement:** HTTP/HTTPS transport with SSE (Server-Sent Events) support

**ServiceNow Does NOT Support:**

- STDIO transport
- WebSocket transport (at this time)

**Authentication Method**

**Recommended:** OAuth 2.1 with PKCE (Proof Key for Code Exchange)

**Must-Have Elements:**

- Authorization Code Grant flow
- PKCE with S256 challenge method (SHA-256)
- JWT Tokens Recommended
- Refresh token support
- Token revocation capability

**Highly Recommended:**

- Dynamic Client Registration (DCR) per RFC 7591
- OAuth 2.0 Authorization Server Metadata (RFC 8414)

**Endpoint Architecture**

**Requirement:** Single JSON-RPC 2.0 endpoint for all MCP protocol operations

**Pattern:**

```
POST /mcp
```

All MCP methods (initialize, tools/list, tools/call, etc.) are sent to this single endpoint with method differentiation via the JSON-RPC `method` field.

**Authentication:**

- `initialize` method: Public (no authentication required)
- All other methods: Require valid OAuth access token

**Required MCP Features**

**Must Implement:**

1. **Tools**: ServiceNow's primary use case - AI agents executing tools
    - `tools/list`: Return available tools with JSON Schema definitions
    - `tools/call`: Execute tool with provided arguments

**Optional (ServiceNow Does Not Currently Support):**

- **Resources**: Structured data sources
- **Prompts**: Templated workflows
- **Sampling**: Server-initiated LLM calls

#### CORS Configuration

**Recommended:** Proper CORS headers to allow ServiceNow instance access

**Recommended Headers:**

- `Access-Control-Allow-Origin`: Your ServiceNow instance URL
- `Access-Control-Allow-Methods`: GET, POST, OPTIONS
- `Access-Control-Allow-Headers`: Content-Type, Authorization
- `Access-Control-Allow-Credentials`: true

**Note:** Some deployment platforms (e.g., Google Cloud Run, AWS API Gateway) may handle CORS automatically or through platform configuration. Verify your platform's CORS handling before implementing custom middleware.

---

## What is NOT Covered in This Guide

#### MCP Features Not Implemented

This guide focuses on ServiceNow integration requirements. The following MCP protocol features are NOT covered:

**Resources** (Structured Data Sources)

- ServiceNow does not currently support MCP Resources
- Resources provide read-only access to structured data
- Not required for ServiceNow AI agent functionality

**Prompts** (Templated Workflows)

- ServiceNow does not currently support MCP Prompts
- Prompts provide pre-defined message templates and workflows
- Not required for ServiceNow AI agent functionality

**Sampling** (Server-Initiated LLM Calls)

- ServiceNow does not currently support MCP Sampling

- Sampling allows servers to request LLM completions from clients
- Not required for ServiceNow AI agent functionality

**Transport Methods Not Covered**

**STDIO Transport**

- ServiceNow requires HTTP/HTTPS transport
- STDIO transport used by local MCP clients (Claude Desktop, Cline)
- Not applicable to ServiceNow cloud integration

**WebSocket Transport**

- ServiceNow does not currently support WebSocket transport
- May be supported in future ServiceNow releases

**Advanced OAuth Patterns Not Covered**

**Client Credentials Grant**

- Service-to-service authentication without user context
- Not used by ServiceNow MCP client

**Device Authorization Grant**

- OAuth for input-constrained devices
- Not applicable to ServiceNow integration

**Implicit Grant**

- Deprecated in OAuth 2.1
- Not used by ServiceNow

**Deployment Topics Not Covered**

**Multi-Tenant Architecture**

- Serving multiple organizations from single server
- Mentioned as storage consideration, not implemented in examples

**Horizontal Scaling**

- Load balancing across multiple server instances

- Requires shared storage (mentioned in examples, not detailed implementation)

**Container Orchestration**

- Kubernetes, Docker Swarm deployment patterns
- Platform-specific, beyond scope of this guide

**CI/CD Pipelines**

- Automated testing and deployment
- Mentioned in Appendix A, not detailed implementation

**Non-ServiceNow MCP Clients**

This guide focuses exclusively on ServiceNow integration. Patterns for other MCP clients are not covered:

- Claude Desktop
- Cline
- Cursor
- Windsurf
- Custom MCP clients

**Note:** The implementation patterns in this guide are generally compatible with other MCP clients, but client-specific features and requirements are not addressed.

---

## Building Blocks - Core Dependencies and Recommendations

**Core Dependencies (Must Have)**

**HTTP Server Framework**

- **Purpose**: Handle HTTP requests/responses, routing, middleware
- **JavaScript Options**: Express.js, Fastify, Koa
- **TypeScript Options**: Express.js with @types/express, NestJS, Fastify
- **Why Required**: MCP protocol requires HTTP/HTTPS transport for ServiceNow integration

**JSON-RPC 2.0 Handler**

- **Purpose**: Parse and validate JSON-RPC 2.0 message format
- **Options**: Custom implementation, json-rpc-2.0 library
- **Why Required**: MCP protocol uses JSON-RPC 2.0 message structure

**Body Parser**

- **Purpose**: Parse JSON and URL-encoded request bodies
- **JavaScript/TypeScript Options**: body-parser, built-in Express middleware
- **Why Required**: OAuth token endpoint requires `application/x-www-form-urlencoded` support

**CORS Middleware**

- **Purpose**: Configure Cross-Origin Resource Sharing headers
- **JavaScript/TypeScript Options**: cors package, custom middleware
- **Why Recommended**: Enables ServiceNow cross-origin requests; some platforms handle CORS automatically, but explicit configuration ensures portability

**JWT Library**

- **Purpose**: Create, sign, and verify JSON Web Tokens
- **JavaScript/TypeScript Options**: jsonwebtoken, jose
- **Why Required**: Stateless token authentication for OAuth 2.1 implementation

**Cryptographic Functions**

- **Purpose**: Generate secure tokens, validate PKCE challenges
- **JavaScript/TypeScript Options**: Node.js crypto (built-in), bcrypt
- **Why Required**: OAuth 2.1 PKCE requires SHA-256 hashing, secure random token generation

**UUID Generator**

- **Purpose**: Generate unique identifiers for clients, tokens, authorization codes
- **JavaScript/TypeScript Options**: uuid package, crypto.randomUUID() (Node.js 14.17+)
- **Why Required**: Client IDs, JWT IDs (jti), user IDs need unique identifiers

---

**Authentication & Token Management (Recommended)**

**Token Blacklist Storage**

- **Purpose**: Persist revoked token identifiers across server restarts
- **Options**: Redis, PostgreSQL, MongoDB, file-based storage
- **Recommendation**: Redis for performance and automatic expiration

- **Why Recommended**: In-memory storage loses revoked tokens on restart; persistent storage prevents token replay attacks

**Rate Limiting**

- **Purpose**: Prevent abuse, authentication flooding, token farming
- **Options**: express-rate-limit, rate-limiter-flexible, custom middleware
- **Recommendation**: express-rate-limit for simplicity, rate-limiter-flexible for Redis-backed distributed rate limiting
- **Why Recommended**: Production security best practice; prevents DOS attacks on OAuth endpoints

**Environment Configuration**

- **Purpose**: Manage secrets, configuration variables outside code
- **Options**: dotenv, environment variables, cloud secret managers (GCP Secret Manager, AWS Secrets Manager)
- **Recommendation**: dotenv for development, cloud secret managers for production
- **Why Recommended**: Never hardcode secrets; enables environment-specific configuration

**Client Persistence Storage**

- **Purpose**: Store registered OAuth client credentials (DCR)
- **Options**: File-based JSON, PostgreSQL, MongoDB, Redis
- **Recommendation**: File-based for single-server, database for multi-server deployments
- **Why Recommended**: Client registrations must survive server restarts

---

**Production Enhancements (Optional but Valuable)**

**HTTP Client Library**

- **Purpose**: Make outbound HTTP requests (e.g., to LLM APIs, external services)
- **Options**: node-fetch, axios, got
- **Use Case**: If your tools need to call external APIs (like Ollama for LLM generation)

**Logging Framework**

- **Purpose**: Structured logging, log levels, log rotation
- **Options**: winston, pino, bunyan
- **Use Case**: Production debugging, audit trails, compliance

**Process Manager**

- **Purpose**: Auto-restart, log management, monitoring
- **Options**: PM2, systemd, Docker/Kubernetes health checks
- **Use Case**: Production deployments requiring high availability

**Validation Library**

- **Purpose**: Validate JSON Schema for tool input parameters
- **Options**: ajv, joi, zod (TypeScript)
- **Use Case**: Ensure tool arguments match declared JSON Schema before execution

---

## Next Steps

Proceed to **Part 2: Server Foundation - Core Infrastructure Setup** to begin building your MCP server implementation.

---

## Document Status

- **Part:** 1 of 5
- **Version:** 1.0
- **Last Updated:** January 29, 2026
- **Status:** Complete