

# 企業環境でのRoam Research MCP統合 - 技術的知見レポート

## プロジェクト概要

**目的:** 企業ネットワーク環境で2b3pro/roam-research-mcpをClaude Desktopと統合

**期間:** 2025年8月7日

**結果:** 部分的成功（MCP統合成功、API接続制限で一部機能制限）

## 成功した取り組み

### 1. HTTP/SSE機能の完全削除

**問題:** 企業ファイアウォールによるポート8088/8087の制限

**解決:** stdio-only版への改変

### 削除したコード

```
// src/server/roam-server.ts から削除
import { StreamableHTTPServerTransport } from
  '@modelcontextprotocol/sdk/server/streamableHttp.js';
import { SSEServerTransport } from '@modelcontextprotocol/sdk/server/sse.js';
import { createServer, IncomingMessage, ServerResponse } from 'node:http';
import { findAvailablePort } from '../utils/net.js';
import { CORS_ORIGIN } from '../config/environment.js';

// async run()メソッドのHTTP関連部分（約40行）を削除
```

### 削除したファイル

- src/utils/net.ts (ポート確認機能)

### 環境変数の整理

```
// src/config/environment.ts
// 削除: HTTP_STREAM_PORT, SSE_PORT, CORS_ORIGIN
export { API_TOKEN, GRAPH_NAME }; // 必要最小限に
```

### 2. フォーク戦略の成功

**リポジトリ:** keides2/roam-research-mcp-enterprise

**説明:** "Enterprise-focused MCP Server for Roam Research (stdio-only, firewall-safe)"

### フォークの価値

- 企業環境特化の機能
- オリジナルと明確な区別
- 独立した開発・保守


### 3. Claude Desktop統合の成功

**設定ファイル:** %APPDATA%\Claude\claude\_desktop\_config.json

```
{
  "mcpServers": {
    "roam-research-enterprise": {
      "command": "node",
      "args": [
        "C:/Users/shimatani/mcp-servers/roam-research-mcp-enterprise/build/index.js"
      ],
      "env": {
        "ROAM_API_TOKEN": "your-actual-roam-token",
        "ROAM_GRAPH_NAME": "your-actual-graph-name"
      }
    }
  }
}
```

**結果:** MCP接続成功、ツール認識完了

## 解決できなかった問題

Roam API接続制限 

**エラー:** MCP error -32603: Roam API error: fetch failed

### 原因分析

#### 1. DNS解決の失敗

```
Test-NetConnection -ComputerName roamresearch.com -Port 443
# WARNING: Name resolution of roamresearch.com failed

nslookup roamresearch.com
# DNS request timed out
```

#### 2. プロキシ経由では接続可能

```
curl -x http://gwproxy.daikin.co.jp:3128 https://roamresearch.com
#  HTML取得成功
```

### 3. Node.js Roam API SDKのプロキシ非対応

- 環境変数での設定では不十分
- SDK内部のHTTP接続がプロキシ設定を認識しない

#### 試行した解決策（すべて失敗）

##### 方法1: プロキシ環境変数設定

```
"env": {
  "HTTP_PROXY": "http://gwproxy.daikin.co.jp:3128",
  "HTTPS_PROXY": "http://gwproxy.daikin.co.jp:3128",
  "NO_PROXY": "localhost,127.0.0.1,.daikin.co.jp",
  "NODE_TLS_REJECT_UNAUTHORIZED": "0"
}
```

##### 方法2: ローカルプロキシ

```
"env": {
  "HTTP_PROXY": "http://127.0.0.1:17200",
  "HTTPS_PROXY": "http://127.0.0.1:17200"
}
```




##### 方法3: コードレベルのプロキシ対応

```
// 依存関係追加
npm install http-proxy-agent https-proxy-agent node-fetch@2

// constructor内でglobalAgent設定
const { HttpProxyAgent } = require('http-proxy-agent');
const { HttpsProxyAgent } = require('https-proxy-agent');
require('http').globalAgent = new HttpProxyAgent(httpProxy);
require('https').globalAgent = new HttpsProxyAgent(httpsProxy);
```

## 技術的知見

### 1. 企業ネットワーク制限の段階的理解

Layer 1: ファイアウォール → ポート制限  解決済み  
Layer 2: DNS制限 → 外部ドメイン解決阻止  未解決  
Layer 3: API通信制限 → 企業ポリシーレベル  未解決

### 2. MCP vs Roam API の分離

## 重要な発見: MCP接続とRoam API接続は別の問題

- **✅ MCP接続:** stdio経由で完全に機能
- **❌ Roam API接続:** 企業ネットワーク制限で失敗

### 証明:

```
// ネットワーク不要なツールは正常動作
roam_markdown_cheatsheet → ✅ 成功
roam_fetch_page_by_title → ❌ fetch failed
```

## 3. プロキシ対応の複雑さ

Node.jsでのプロキシ対応は3層の対応が必要:

1. **HTTP/HTTPS接続:** http-proxy-agent, https-proxy-agent
2. **DNS解決:** カスタムDNS resolver
3. **SDK内部処理:** ライブラリ固有の実装

## 次期解決策

### ハイブリッド戦略

#### Phase 1: リモートMCPサーバー構築（短期解決）

```
企業内PC → 企業プロキシ → AWS Lambda → Roam API
  ↑           ↑           ↑
Claude Desktop HTTPS通信 MCPサーバー
```

### 技術スタック:

- AWS Lambda / Google Cloud Functions
- Express.js + MCP Protocol
- JWT Authentication
- Roam API プロキシ機能

**実装期間:** 1-2週間

**運用コスト:** \$5-20/月

#### Phase 2: Roam Research改善要望（長期解決）

### GitHub Issue作成:

- タイトル: "Enterprise proxy support for Roam API SDK"
- 企業環境でのプロキシ対応要求
- 具体的なエラーログと解決案
- ビジネスケースの説明

# 学習成果

## 1. MCPアーキテクチャの理解

- stdio vs HTTP/SSE transport の違い
- 企業環境での制約とそれに対する適応

## 2. 企業ネットワークの複雑性

- ファイアウォール、DNS制限、プロキシ設定の段階的理解
- セキュリティポリシーとAPI統合の課題

## 3. オープンソース改善の実践

- フォーク戦略の有効性
- 企業環境特化版の価値

# 推奨事項

## 他の企業ユーザー向け

### 1. まずstdio-only版を試行

- HTTP/SSE機能が不要な環境では十分
- ポート制限問題を完全回避

### 2. プロキシ設定の段階的テスト

```
# 接続テスト手順
Test-NetConnection -ComputerName roamresearch.com -Port 443
curl -x http://your-proxy:port https://roamresearch.com
```

### 3. リモートサーバー戦略の検討

- 即座の解決が必要な場合
- クラウドコストと企業ポリシーのバランス

## Roam Research開発チーム向け

### 1. SDK内プロキシ対応の強化

- http-proxy-agentの統合
- 環境変数ベースの自動設定

### 2. 企業向けドキュメントの整備

- プロキシ設定ガイド
- 企業ネットワーク制約の対応方法

### 3. エンタープライズ機能の検討

- On-premises deployment option

- Enterprise SSO integration

## 結論

今回のプロジェクトにより、企業環境でのMCP統合における技術的障壁を明確に特定し、部分的な解決を達成しました。HTTP/SSE削除による`studio-only`版の作成は完全に成功し、企業ファイアウォール問題を解決しました。

一方で、Roam API接続の制限は企業ネットワークポリシーレベルの問題であり、技術的解決だけでは限界があることも判明しました。

この知見は、同様の企業環境でAPI統合を行う他のプロジェクトにも応用可能な貴重な経験となりました。

---

**作成日:** 2025年8月7日

**プロジェクト:** keides2/roam-research-mcp-enterprise

**次期目標:** リモートMCPサーバー構築 + Roam Research改善要望