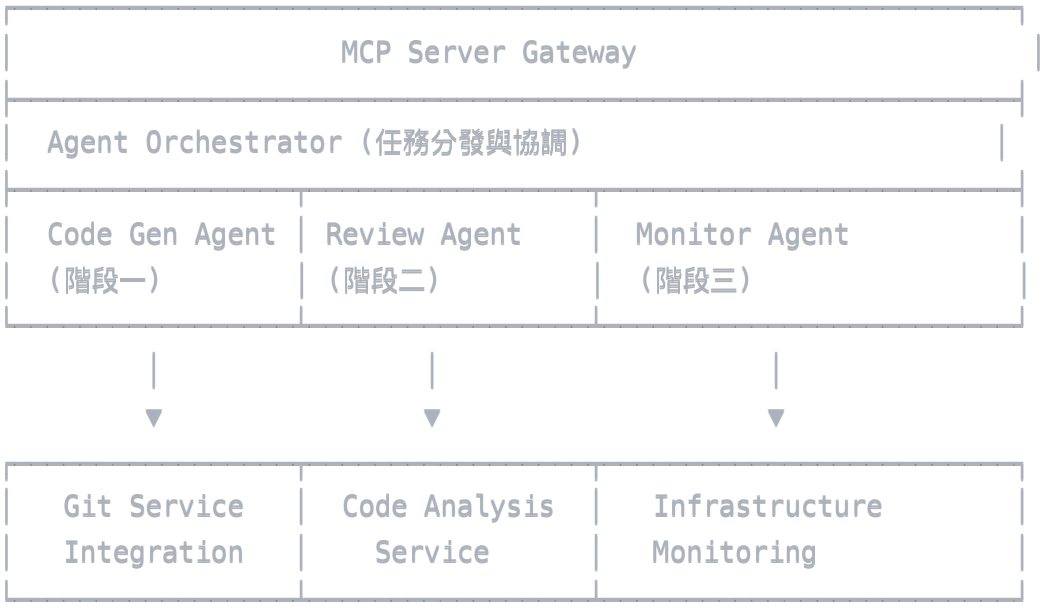
Multi-Agent MCP Server 部署策略

系統架構概覽

Core Components



階段一：需求理解 + 程式碼生成 + 自動提交

Code Generation Agent 設計

核心功能：

- 需求解析與理解（NLP + 結構化分析）
- 程式碼生成（支援多語言）
- Git 自動化操作（commit & push）

技術組件：

- **需求解析模組**：使用 LLM 解析自然語言需求
- **程式碼生成模組**：基於 template + 動態生成
- **Git 整合模組**：自動化 git 操作
- **品質檢查模組**：基本語法與格式檢查

部署考量：

```yaml
Code Gen Agent:
  - Runtime: Python/Node.js 容器
  - LLM Backend: 本地部署或 API 調用
  - Git Integration: GitLab/GitHub API
  - Storage: Redis (session) + PostgreSQL (需求歷史)
  - Security: SSH keys 管理，權限控制
```

階段二：程式碼審查 + 除錯協助

Review & Debug Agent 設計

核心功能：

- 靜態程式碼分析
- 程式碼品質評估
- Bug 識別與修復建議
- Pull Request 自動化審查

技術組件：

- **靜態分析引擎**：SonarQube/ESLint/Pylint 整合
- **AI Review 模組**：程式碼模式識別
- **Debug 助手**：錯誤診斷與解決方案推薦
- **CI/CD 整合**：自動觸發 review 流程

階段三：生產環境監控

Production Monitor Agent 設計

核心功能：

- 應用效能監控
- 錯誤日誌分析
- 自動化告警與回應
- 容量規劃建議

技術組件：

- **監控數據收集**：Prometheus/Grafana 整合
- **日誌分析**：ELK Stack 或雲端 logging
- **智能告警**：異常檢測與預測

- **自動修復**：基礎問題自動處理

Multi-Agent 協調機制

Agent Orchestrator

```python
class AgentOrchestrator:
    def route_request(self, request):
        if request.type == "code_generation":
            return self.code_gen_agent.handle(request)
        elif request.type == "code_review":
            return self.review_agent.handle(request)
        elif request.type == "monitoring":
            return self.monitor_agent.handle(request)
```

通訊協議

- **內部通訊**：gRPC 或 Message Queue (RabbitMQ/Redis)
- **外部介面**：REST API + WebSocket (即時通知)
- **資料同步**：Event-driven architecture

部署架構建議

Infrastructure Setup

```yaml
Production Environment:
  Load Balancer:
    - Nginx/HAProxy
    - SSL termination

  Application Layer:
    - Kubernetes cluster (3+ nodes)
    - Docker containers per agent
    - Auto-scaling 配置

  Data Layer:
    - PostgreSQL (主資料庫)
    - Redis (快取 + session)
    - MongoDB (日誌與監控數據)

  Monitoring:
    - Prometheus + Grafana
    - ELK Stack
    - Health checks
```

Security Considerations

- **API 認證**：JWT + API Keys
- **Git 權限**：最小權限原則，分離式 SSH keys
- **網路安全**：VPC + Security Groups
- **資料加密**：傳輸與儲存加密
- **審計日誌**：完整操作記錄

階段性實施計畫

Phase 1 (MVP - 1-2個月)

1. 建立基礎 MCP server 框架
2. 實作 Code Generation Agent
3. Git 整合與自動提交功能
4. 基礎 Web UI 介面

Phase 2 (增強版 - 2-3個月)

1. Review Agent 開發
2. CI/CD pipeline 整合
3. 程式碼品質檢查自動化

4. 團隊協作功能

Phase 3 (完整版 - 3-4個月)

1. Monitor Agent 實作
2. 生產環境整合
3. 智能告警系統
4. 效能優化與擴展

技術棧建議

Backend

- **語言**：Python (FastAPI) 或 Node.js (Express)
- **AI/ML**：Transformers, LangChain
- **資料庫**：PostgreSQL + Redis + MongoDB
- **訊息佇列**：RabbitMQ 或 Apache Kafka

Frontend (管理介面)

- **框架**：React/Vue.js
- **UI 庫**：Ant Design 或 Material-UI
- **狀態管理**：Redux/Vuex

DevOps

- **容器化**：Docker + Kubernetes
- **CI/CD**：GitLab CI 或 GitHub Actions
- **監控**：Prometheus + Grafana
- **日誌**：ELK Stack

風險評估與建議

技術風險

- **LLM 準確性**：建立 fallback 機制與人工確認流程
- **Git 操作安全**：實作 rollback 機制與權限控制
- **系統穩定性**：充分的測試與監控

營運風險

- **資源消耗**：合理的 rate limiting 與資源配額
- **成本控制**：LLM API 使用監控與預算警告

- **團隊採用**：漸進式導入與訓練計畫

成功指標

量化指標

- 程式碼生成準確率 > 85%
- 自動 commit 成功率 > 95%
- Code review 覆蓋率 > 90%
- 系統可用性 > 99.5%

質化指標

- Junior 開發者工作效率提升
- 程式碼品質改善
- 生產環境穩定性增加
- 團隊滿意度調查結果