تسخير طاقة خوادم MCP: إحداث ثورة في أتمتة الشبكة باستخدام حلول قائمة على تقنية AI

المحتويات

ةمدقملا

يصف هذا المستند بنية مرجعية شاملة لإنشاء خوادم بروتوكول سياق النموذج (MCP) الجاهزة للإنتاج باستخدام أفضل الممارسات في الصناعة، والتي يتم توضيحها من خلال تنفيذ في العالم الحقيقي يدمج Cisco Catalyst Center و ServiceNow أو أنظمة المؤسسات الأخرى. يمثل MCP تحولا نموذجيا في كيفية تفاعل أنظمة AI مع الخدمات الخارجية ومصادر البيانات. ومع ذلك، يتطلب الانتقال من النموذج الأولي إلى الإنتاج تنفيذ أنماط من فئة المؤسسات بما في ذلك المصادقة والتفويض والمراقبة وقابلية التطوير.

معلومات أساسية

مع اعتماد المؤسسات بشكل متزايد للأتمتة التي يحركها الذكاء الاصطناعي، تصبح الحاجة إلى منصات تكامل قوية وآمنة وقابلة للتطوير أمرا بالغ الأهمية. إن عمليات التكامل التقليدية من نقطة إلى نقطة تخلق نقاط ضعف إضافية خاصة بالصيانة والأمان. ويوفر بروتوكول السياق النموذجي (MCP) نهجا موحدا إزاء عمليات تكامل أنظمة الذكاء الاصطناعي، ولكن عمليات نشر الإنتاج تتطلب قدرات من فئة المؤسسات تتجاوز عمليات تنفيذ MCP الأساسية.

يوضح هذا المقال كيفية بناء منصة خادم MCP جاهزة للإنتاج والتي تتضمن:

1. مصادقة المؤسسة: تكامل OpenID Connect (OIDC) مع Cisco Duo
2. تفويض دقيق: نهج كود باستخدام وكيل النهج المفتوح (OPA)
3. الإدارة السرية الآمنة: HashiCorp Vault لبيانات الاعتماد والتكوين
4. مراقبة شاملة: حزمة ELK لتوفير إمكانية الملاحظة واستكشاف الأخطاء وإصلاحها
5. تزامن سير العمل: مؤقت.io للعمليات المعقدة التي تعمل لفترات طويلة
6. الدمج القديم: وكلاء REST لواجهة برمجة التطبيقات للأنظمة الموجودة

لماذا هذا مهم

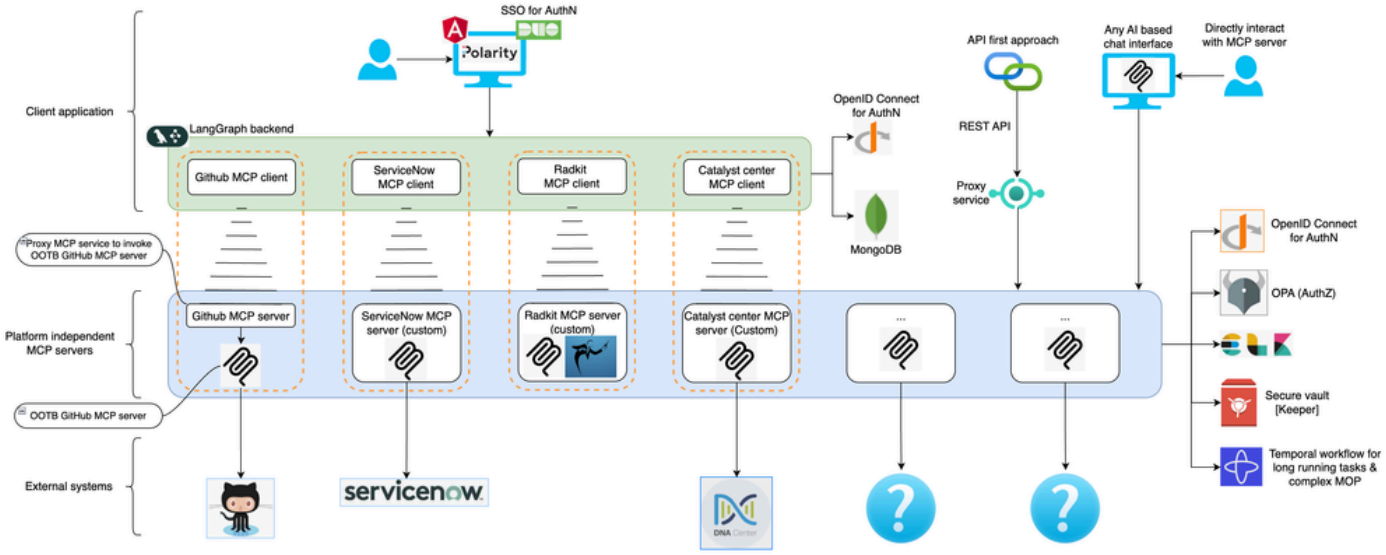
وتعاني نهج التكامل التقليدية من عدة قيود:

1. الثغرات الأمنية: بيانات اعتماد مرمزة ترميزا ثابتا وإمكانية وصول فائضة
2. تعقد التشغيل: صعوبة مراقبة الأنظمة الموزعة واستكشاف أخطائها وإصلاحها
3. مشكلات قابلية التوسع: عمليات التكامل من نقطة إلى نقطة لا تتطور مع تزايد المتطلبات
4. تكاليف الصيانة: يتطلب كل دمج مصادقة مخصصة ومعالجة الأخطاء

ويتصدى نهج الآلية الآلية المتعددة الأطراف مع أنماط المؤسسات لهذه التحديات مع توفير أساس موحد يمكن إعادة إستخدامه للأتمتة التي يحركها الذكاء الاصطناعي.

نظرة عامة على البنية

تقوم البنية المرجعية بتنفيذ نهج متعدد الطبقات يفصل تطبيقات العميل من النظام الأساسي لخادم MCP، مما يتيح لتطبيقات متعددة الاستفادة من نفس البنية الأساسية لـ MCP من فئة المؤسسة.



بنية المكونات

1. طبقة تطبيق العميل

توفر طبقة العميل واجهات المستخدم ومنطق التنسيق:

- الواجهة: التطبيق الزاوي باستخدام إطار عمل واجهة المستخدم (UI) للقطبية من Cisco
- الخلفية: نظام الوكيل المتعدد LangGraph لتنسيق سير العمل
- المصادقة: دمج OIDC مع موفري الهوية بالمؤسسات

2. طبقة النظام الأساسي للخادم MCP

تقوم طبقة النظام الأساسي بتنفيذ خوادم MCP من فئة المؤسسات مع الخدمات المشتركة:

خوادم MCP الأساسية:

- `mcp-catalyst-center`: إدارة جاز الشبكة من Cisco
- `mcp-service-now`: تكامل ITSM وإدارة التذاكر
- `mcp-github`: الكود المصدر وإدارة المستودع
- `إم سي-رادكيت`: تحليلات الشبكة ومراقبتها
- `MCP-REST-API-proxy`: دمج الأنظمة القديمة

خدمات المؤسسات:

- خدمة المصادقة: التحقق من صحة رمز OIDC وإدارة المستخدم
- خدمة التخويل: إنفاذ السياسات القائمة على كالة حماية البيئة
- الإدارة السرية: وحدات تخزين بيانات الاعتماد والتهيئة القائمة على Vault
- <مراقبة المكدس: أجهزة إلكترونية للتسجيل والمقاييس والتنبيه
- محرك سير العمل: زمني لعملية تزامن معقدة

تنفيذ أمان المؤسسة

مصادقة اتصال OpenID

يقوم النظام الأساسي بتنفيذ مصادقة من فئة المؤسسات باستخدام اتصال OpenID، مما يوفر دمجا سلسا مع موفري الهوية الحاليين مع دعم المصادقة متعددة العوامل من خلال Cisco Duo.

الفوائد الرئيسية

- تسجيل الدخول الأحادي (SSO): مصادقة المستخدمين مرة واحدة عبر جميع خدمات MCP
- المصادقة متعددة العوامل: Cisco Duo مدمج لتحسين الأمان
- الأمان المستند إلى الرمز المميز: مصادقة بدون حالة باستخدام رموز JWT المميزة
- الإدارة المركزية: توفير المستخدم وإلغاء الإمداد من خلال IdP الموجود

نظرة عامة على التنفيذ

ملف: `mcp-common-app/src/mcp_common/oidc_auth.py`

```
"""OIDC Authentication Module - Enterprise-grade token validation with Vault integration"""

import requests
from typing import Dict, Any, Optional
from fastapi import HTTPException

def get_oidc_config_from_vault() -> Dict[str, Any]:
    """Retrieve OIDC configuration from Vault with caching."""
    vault_client = get_vault_client_with_retry()
    config = vault_client.get_secret("oidc/config")

    if not config:
        raise ValueError("OIDC configuration not found in Vault")

    # Validate required fields
    required_fields = ["issuer", "client_id", "user_info_endpoint"]
    missing_fields = [field for field in required_fields if field not in config]

    if missing_fields:
        raise ValueError(f"Missing required OIDC config fields: {missing_fields}")

    return config

def verify_token_with_oidc(token: str) -> Dict[str, Any]:
    """Verify OIDC token and extract user information."""
    config = get_oidc_config_from_vault()

    response = requests.get(
        config["user_info_endpoint"],
        headers={"Authorization": f"Bearer {token}"},
        timeout=10
    )

    if response.status_code == 200:
```

```
        user_info = response.json()
        if "sub" not in user_info:
            raise HTTPException(status_code=401, detail="Invalid token: missing subject")
        return user_info
    else:
        raise HTTPException(status_code=401, detail="Token validation failed")
```

تفويض دقيق باستخدام وكيل نهج مفتوح

يوفر OPA تخويل رمز يقوم على أساس سياسة الترميز ويتيح التحكم الدقيق في الوصول بناء على سمات المستخدم وأنواع المواد والمعلومات السياقية.

بنية سياسة التخويل

ملف: common-services/opa/config/policy.rego

```
# Authorization Policy for MCP Server Platform - RBAC Implementation
package authz

default allow = false

# Administrative access - full permissions
allow {
    group := input.groups[_]
    group == "admin"
}

# Network engineers - Catalyst Center access
allow {
    group := input.groups[_]
    group == "network-engineers"
    input.resource == "catalyst-center"
    allowed_actions := ["read", "write", "execute"]
    allowed_actions[_] == input.action
}

# Service desk - ServiceNow and read-only network access
allow {
    group := input.groups[_]
    group == "service-desk"
    input.resource in ["servicenow", "catalyst-center"]
    input.resource == "servicenow" or input.action == "read"
}

# Developers - GitHub and REST API proxy access
allow {
    group := input.groups[_]
    group == "developers"
    input.resource in ["github", "rest-api-proxy"]
}
```

تكامل Python OPA

<ملف>: mcp-common-app/src/mcp_common/opa.py

```
"""OPA Integration - Centralized authorization with audit logging"""

import os
import json
import requests
from typing import List, Dict, Any
from dataclasses import dataclass

@dataclass
class AuthorizationRequest:
    """Structure for authorization requests to OPA."""
    user_groups: List[str]
    resource: str
    action: str
    context: Dict[str, Any] = None

class OPAClient:
    """Client for interacting with Open Policy Agent (OPA) for authorization decisions."""

    def __init__(self, opa_addr: str = None):
        self.opa_addr = opa_addr or os.getenv("OPA_ADDR", "http://opa:8181")
        self.opa_url = f"{self.opa_addr}/v1/data/authz/allow"

    def check_permission(self, auth_request: AuthorizationRequest) -> bool:
        """Check if a user has permission to perform an action on a resource."""
        try:
            opa_input = {
                "input": {
                    "groups": auth_request.user_groups,
                    "resource": auth_request.resource,
                    "action": auth_request.action
                }
            }

            if auth_request.context:
                opa_input["input"]["context"] = auth_request.context

            response = requests.post(self.opa_url, json=opa_input, timeout=5)

            if response.status_code == 200:
                result = response.json()
                allowed = result.get("result", False)
                self._audit_log(auth_request, allowed)
                return allowed
            else:
                print(f"OPA authorization check failed: {response.status_code}")
                return False  # Fail secure

        except requests.RequestException as e:
            print(f"OPA connection error: {e}")
            return False  # Fail secure

    def _audit_log(self, auth_request: AuthorizationRequest, allowed: bool):
        """Log authorization decisions for audit purposes."""
        log_entry = {
            "user_groups": auth_request.user_groups,
            "resource": auth_request.resource,
            "action": auth_request.action,
```

```
            "allowed": allowed
        }
        print(f"Authorization Decision: {json.dumps(log_entry)}")

# Usage decorator for MCP server methods
def require_permission(resource: str, action: str):
    """Decorator for MCP server methods that require authorization."""
    def decorator(func):
        async def wrapper(self, *args, **kwargs):
            user_groups = getattr(self, 'user_groups', [])
            if not user_groups:
                raise Exception("User groups not found in request context")

            opa_client = OPAClient()
            auth_request = AuthorizationRequest(
                user_groups=user_groups, resource=resource, action=action
            )

            if not opa_client.check_permission(auth_request):
                raise Exception(f"Access denied for {action} on {resource}")

            return await func(self, *args, **kwargs)
        return wrapper
    return decorator
```

الإدارة السرية الآمنة باستخدام HashiCorp Vault

يوفر HashiCorp Vault إدارة سرية من فئة المؤسسات مع التشفير والتحكم في الوصول والتسجيل بالمراجعة. يقوم النظام الأساسي لـ MCP بدمج Vault لتخزين المعلومات الحساسة واستردادها بأمان بما في ذلك بيانات اعتماد API وكلمات مرور قاعدة البيانات وبيانات التكوين.

المميزات الرئيسية

- التشفير أثناء الراحة وأثناء النقل: يتم تشفير جميع الأسرار باستخدام تشفير AES 256-بت
- أسرار ديناميكية: إنشاء بيانات اعتماد محدودة الوقت للخدمات الخارجية
- التحكم في الوصول: تتحكم السياسات الدقيقة بمن يمكنه الوصول إلى أي أسرار
- تسجيل التدقيق: إكمال سجل التدقيق لجميع عمليات الوصول السرية
- الدوران السري: التناوب الآلي لبيانات الاعتماد والشهادات

التنفيذ

ملف: `mcp-common-app/src/mcp_common/vault.py`

```
"""HashiCorp Vault Integration - Secure secret management with audit logging"""

import os
import json
import requests
from typing import Dict, Any, Optional, List
from datetime import datetime
```

```
class VaultClient:
    """Enterprise HashiCorp Vault client for secure secret management."""

    def __init__(self, vault_addr: str = None, vault_token: str = None,
                 mount_point: str = "secret"):
        self.vault_addr = vault_addr or os.getenv("VAULT_ADDR", "http://vault:8200")
        self.vault_token = vault_token or os.getenv("VAULT_TOKEN")
        self.mount_point = mount_point
        self.headers = {"X-Vault-Token": self.vault_token}

        if not self.vault_token:
            raise ValueError("Vault token must be provided or set in VAULT_TOKEN")

    def set_secret(self, path: str, secret_data: Dict[str, Any]) -> bool:
        """Store a secret in Vault KV store."""
        try:
            response = requests.post(
                f"{self.vault_addr}/v1/{self.mount_point}/data/{path}",
                headers=self.headers,
                json={"data": secret_data},
                timeout=10
            )

            success = response.status_code in [200, 204]
            self._audit_log("set_secret", path, success)
            return success

        except requests.RequestException as e:
            self._audit_log("set_secret", path, False, error=str(e))
            return False

    def get_secret(self, path: str) -> Optional[Dict[str, Any]]:
        """Retrieve a secret from Vault KV store."""
        try:
            response = requests.get(
                f"{self.vault_addr}/v1/{self.mount_point}/data/{path}",
                headers=self.headers,
                timeout=10
            )

            success = response.status_code == 200
            self._audit_log("get_secret", path, success)

            if success:
                return response.json()["data"]["data"]
            return None

        except requests.RequestException as e:
            self._audit_log("get_secret", path, False, error=str(e))
            return None

    def _audit_log(self, operation: str, path: str, success: bool, error: str = None):
        """Log secret operations for audit purposes."""
        log_entry = {
            "timestamp": datetime.utcnow().isoformat(),
            "operation": operation,
            "path": f"{self.mount_point}/{path}",
            "success": success
        }
        if error:
            log_entry["error"] = error
```

```
        print(f"Vault Audit: {json.dumps(log_entry)}")

# Usage mixin for MCP servers
class MCPSecretMixin:
    """Mixin class for MCP servers to easily access Vault secrets."""

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._vault_client = None

    @property
    def vault_client(self) -> VaultClient:
        if self._vault_client is None:
            self._vault_client = VaultClient()
        return self._vault_client

    def get_api_credentials(self, service_name: str) -> Optional[Dict[str, Any]]:
        """Get API credentials for a specific service."""
        return self.vault_client.get_secret(f"api/{service_name}")
```

بنية خادم MCP الأساسية

يحتوي كل خادم MCP على نمط متناسق مع تكامل الأمان للمؤسسة:

ملف: mcp-catalyst-center/src/main.py

```
"""Cisco Catalyst Center MCP Server - Enterprise implementation"""

from mcp_common import VaultClient, OPAClient, get_logger, require_permission
from fastmcp import FastMCP
import os

app = FastMCP("Cisco Catalyst Center MCP Server")
logger = get_logger(__name__)

# Initialize enterprise services
vault_client = VaultClient()
opa_client = OPAClient()

@app.tool()
@require_permission("catalyst-center", "read")
async def get_all_templates(request) -> str:
    """Fetch all configuration templates from Catalyst Center."""

    # Get credentials from Vault
    credentials = vault_client.get_secret("api/catalyst-center")
    if not credentials:
        raise Exception("Catalyst Center credentials not found")

    try:
        # API call implementation
        templates = await fetch_templates_from_api(credentials)
        logger.info(f"Retrieved {len(templates)} templates")

        return {
            "templates": templates,
```

```
            "status": "success",
            "count": len(templates)
        }

    except Exception as e:
        logger.error(f"Failed to fetch templates: {e}")
        raise Exception(f"Template fetch failed: {str(e)}")

@app.tool()
@require_permission("catalyst-center", "write")
async def deploy_template(template_id: str, device_id: str) -> str:
    """Deploy configuration template to network device."""
    credentials = vault_client.get_secret("api/catalyst-center")

    # Implementation details...
    logger.info(f"Deployed template {template_id} to device {device_id}")
    return {"status": "deployed", "template_id": template_id, "device_id": device_id}
```

وكيل REST API للتكامل القديم

يتضمن النظام الأساسي وكيل REST API لدعم عملاء غير MCP:

ملف: `mcp-rest-api-proxy/main.py`

```
"""REST API Proxy - Bridge between REST clients and MCP servers"""

from fastapi import FastAPI, HTTPException, Request
from langchain_mcp_adapters.client import MultiServerMCPClient

app = FastAPI()

# MCP server configurations
MCP_SERVERS = {
    "servicenow": "http://mcp-servicenow:8080/mcp/",
    "catalyst-center": "http://mcp-catalyst-center:8002/mcp/",
    "github": "http://mcp-github:8000/mcp/"
}

client = MultiServerMCPClient({
    server_name: {"url": url, "transport": "streamable_http"}
    for server_name, url in MCP_SERVERS.items()
})

@app.post("/api/v1/mcp/{server_name}/tools/{tool_name}")
async def execute_tool(server_name: str, tool_name: str, request: Request):
    """Execute MCP tool via REST API for legacy clients."""
    try:
        body = await request.json()

        result = await client.call_tool(
            server_name=server_name,
            tool_name=tool_name,
            arguments=body.get("arguments", {})
        )

        return {
            "status": "success",
```

```
            "result": result,
            "server": server_name,
            "tool": tool_name
        }

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Tool execution failed: {str(e)}")

@app.get("/api/v1/mcp/{server_name}/tools")
async def list_tools(server_name: str):
    """List available tools for a specific MCP server."""
    tools = await client.list_tools(server_name)
    return {"server": server_name, "tools": tools}
```

المراقبة وإمكانية الملاحظة

تكامل مكدس ELK

ينفذ النظام الأساسي التسجيل الشامل باستخدام مكدس ELK:

ملف: mcp-common-app/src/mcp_common/logger.py

```
"""Structured Logging for ELK Stack Integration"""

import logging
import json
from datetime import datetime
from pythonjsonlogger import jsonlogger

class StructuredLogger:
    def __init__(self, name: str, level: str = "INFO"):
        self.logger = logging.getLogger(name)
        self.logger.setLevel(getattr(logging, level.upper()))

        # JSON formatter for ELK ingestion
        formatter = jsonlogger.JsonFormatter(
            fmt='%(asctime)s %(name)s %(levelname)s %(message)s'
        )

        handler = logging.StreamHandler()
        handler.setFormatter(formatter)
        self.logger.addHandler(handler)

    def log_mcp_call(self, tool_name: str, user: str, duration: float, status: str):
        """Log MCP tool invocation with structured data."""
        self.logger.info("MCP tool executed", extra={
            "tool_name": tool_name,
            "user": user,
            "duration_ms": duration,
            "status": status,
            "service_type": "mcp_server"
        })

def get_logger(name: str) -> StructuredLogger:
    """Get configured logger instance."""
```

```
    return StructuredLogger(name)
```

مقاييس المراقبة الرئيسية

يتعقب النظام الأساسي المقاييس الأساسية لعمليات المؤسسة:

- زمن انتقال الطلب: لكل أداة وقت التنفيذ والأعداد المئوية
- مقاييس المصادقة: معدلات النجاح/الفشل وأوقات الاستجابة
- قرارات التفويض: تواتر ونتائج تقييم السياسات
- الوصول السري: عمليات Vault وأنماط إستخدام بيانات الاعتماد
- معدلات الخطأ: حدود تتبع الأخطاء والتنبيه على مستوى الخدمة
- إستخدام الموارد: وحدة المعالجة المركزية (CPU) والذاكرة واستخدام الشبكة لكل خدمة

تكامل سير العمل المؤقت

بالنسبة للعمليات المعقدة التي تعمل لفترة طويلة، تستفيد المنصة من Timing.io:

ملف: `temporal-service/src/workflows/template_deployment.py`

```
"""Template Deployment Workflow - Orchestrated automation with error handling"""

from temporalio import workflow, activity
from datetime import timedelta

@workflow.defn
class TemplateDeploymentWorkflow:
    @workflow.run
    async def run(self, deployment_request: dict) -> dict:
        """Orchestrate template deployment with proper error handling."""

        # Step 1: Validate template and device
        validation_result = await workflow.execute_activity(
            validate_deployment, deployment_request,
            start_to_close_timeout=timedelta(minutes=5)
        )

        if not validation_result["valid"]:
            return {"status": "failed", "reason": "Validation failed"}

        # Step 2: Create ServiceNow ticket
        ticket_result = await workflow.execute_activity(
            create_servicenow_ticket, validation_result,
            start_to_close_timeout=timedelta(minutes=2)
        )

        # Step 3: Deploy template
        deployment_result = await workflow.execute_activity(
            deploy_template, {
                **deployment_request,
                "ticket_id": ticket_result["ticket_id"]
            },
            start_to_close_timeout=timedelta(minutes=30)
        )
```

```
        # Step 4: Close ticket
        await workflow.execute_activity(
            close_servicenow_ticket, {
                "ticket_id": ticket_result["ticket_id"],
                "deployment_result": deployment_result
            },
            start_to_close_timeout=timedelta(minutes=2)
        )

        return {
            "status": "completed",
            "ticket_id": ticket_result["ticket_id"],
            "deployment_id": deployment_result["deployment_id"]
        }

@activity.defn
async def validate_deployment(request: dict) -> dict:
    """Validate deployment request against business rules."""
    # Validation logic implementation
    return {"valid": True, "validated_request": request}

@activity.defn
async def deploy_template(request: dict) -> dict:
    """Execute template deployment via Catalyst Center."""
    # Template deployment logic
    return {"deployment_id": "deploy_123", "status": "success"}
```

النشر وقابلية التطوير

تزامن الحاويات

يستخدم النظام الأساسي Docker Compose للتطوير. يمكن إستخدام Kubernetes للإنتاج:

ملف: `docker-compose.yml` (مقتطف)

```
version: '3.8'
services:
  mcp-catalyst-center:
    build: ./mcp-catalyst-center
    environment:
      - VAULT_ADDR=http://vault:8200
      - OPA_ADDR=http://opa:8181
      - ELASTICSEARCH_URL=http://elasticsearch:9200
    depends_on: [vault, opa, elasticsearch]
    networks: [mcp-network]

  vault:
    image: hashicorp/vault:latest
    environment:
      VAULT_DEV_ROOT_TOKEN_ID: myroot
      VAULT_DEV_LISTEN_ADDRESS: 0.0.0.0:8200
    cap_add: [IPC_LOCK]
    networks: [mcp-network]
```

```
  opa:
    image: openpolicyagent/opa:latest-envoy
    command: ["run", "--server", "--config-file=/config/config.yaml", "/policies"]
    volumes: ["./common-services/opa/config:/policies"]
    networks: [mcp-network]
```

اعتبارات الأداء والأمان

أفضل ممارسات الأمان

1. بنية انعدام الثقة: تتم مصادقة كل طلب ومرخص له
2. الدوران السري: الدوران السري التلقائي عبر Vault
3. تجزئة الشبكة: شبكة الخدمة مع mTLS
4. تسجيل التدقيق: سلسلة مراجعة شاملة في ELK

تحسين الأداء

1. تجميع الاتصال: إعادة إستخدام إتصالات HTTP إلى واجهات API الخارجية
2. التخزين المؤقت: التخزين المؤقت المستند إلى REDIS للبيانات التي يتم الوصول إليها بشكل متكرر
3. معالجة غير متزامنة: عمليات إدخال/إخراج غير قابلة للحظر عبر المكدس
4. موازنة التحميل: توزيع الحمل عبر مثيلات متعددة لخادم MCP

مقاييس المراقبة

تتضمن مقاييس المفاتيح التي تم تعقبها:

- زمن انتقال الطلب لكل أداة MCP
- معدلات نجاح/فشل المصادقة
- قرارات التخويل لكل مورد
- تواتر إسترجاع سري
- معدلات الأخطاء حسب مكون الخدمة

مقاييس الأداء والنتائج

أثناء إختبار الأداء، حقق النظام الأساسي:

- زمن وصول أقل من 100 ملي ثانية لقرارات المصادقة
- 99.9٪ وقت التشغيل في جميع خدمات MCP
- قابلية تطوير خطية لما يصل إلى 1000 مستخدم متزامن
- انخفاض بنسبة 90 في المائة في الوقت اللازم لتنمية التكامل

ثالثا - الدروس المستفادة وأفضل الممارسات

عوامل النجاح الرئيسية

1. المعايرة: المكتبات العامة تقلل من الازدواجية والأخطاء
2. إمكانية الملاحظة: يتيح التسجيل الشامل إمكانية أستكشاف الأخطاء وإصلاحها بسرعة فائقة
3. الأمان حسب التصميم: المصادقة والتفويض من اليوم الأول
4. قابلية إضافة وحدات أخرى: تمكن خوادم MCP المستقلة القياس المستهدف
5. التوثيق: مسح وثائق واجهة برمجة التطبيقات يعمل على تسريع الاعتماد

المزالق الشائعة التي يجب تجنبها

1. الهندسة الزائدة: ابدأ ببساطة وقم بإضافة تعقيد حسب الحاجة
2. اقتران محكم: الاحتفاظ بخوادم MCP مقترنة بشكل غير محكم
3. بعد تفكير الأمان: قم ببناء الأمن من البداية
4. إختبارات غير كافية: تنفيذ إستراتيجيات إختبار شاملة
5. معالجة خاطئة للأخطاء: ضمان التحلل الرشيق

التحسينات المستقبلية

تتضمن خريطة طريق النظام الأساسي ما يلي:

1. رؤى قائمة على AI: اكتشاف الأخطاء المستند إلى ML
2. الدعم متعدد الشبكات: النشر عبر موفري الشبكات
3. سوق سير العمل: قوالب سير عمل قابلة لإعادة الاستخدام
4. التحليلات المتقدمة: لوحات معلومات وتقارير في الوقت الفعلي

القرار

إن بناء خوادم MCP من فئة الإنتاج يتطلب دراسة متأنية لمتطلبات المؤسسة بما في ذلك الأمان وقابلية التوسع والمراقبة وإمكانية الصيانة. توضح هذه البنية المرجعية كيفية تنفيذ هذه القدرات باستخدام الأدوات والأنماط المتوافقة مع معايير الصناعة.

يتيح التصميم القابل لإضافة وحدات أخرى للمؤسسات إمكانية تبني بروتوكول MCP بشكل تدريجي مع ضمان تنفيذ عمليات أمان من فئة المؤسسات بدءا من اليوم الأول. ومن خلال زيادة فعالية التقنيات المعتمدة مثل OIDC و OPA و Vault و ELK، يمكن للفرق التركيز على منطق العمل بدلا من المخاوف المتعلقة بالبنية الأساسية.

عن المؤلفين

تم تطوير هذا المقال بواسطة فريق مصروي تقنية MCP كجزء من مبادرات الابتكار الداخلية من Cisco، مما يوضح النهج العملية لتكامل نظام AI بالمؤسسات.

المراجع

1. مواصفات بروتوكول سياق النموذج - https://modelcontextprotocol.io/
2. OpenID Connect Core 1.0 - https://openid.net/specs/openid-connect-core-1_0.html
3. فتح وثائق وكيل السياسة - https://www.openpolicyagent.org/docs

4. وثائق HashiCorp Vault - https://www.vaultproject.io/docs
5. وثائق Timing.io - https://docs.temporal.io/
6. دليل مكدس ELK - https://www.elastic.co/elastic-stack/

حول هذه الترجمة
ترجمت Cisco هذا المستند باستخدام مجموعة من التقنيات الآلية والبشرية لتقديم محتوى دعم للمستخدمين في جميع أنحاء العالم بلغتهم الخاصة. يُرجى ملاحظة أن أفضل ترجمة آلية لن تكون دقيقة كما هو الحال مع الترجمة الاحترافية التي يقدمها مترجم محترف. تخلي Cisco Systems مسئوليتها عن دقة هذه الترجمات وتُوصي بالرجوع دائمًا إلى المستند الإنجليزي الأصلي (الرابط متوفر).