# 面向 AI 辅助开发者服务的企业身份：架构、实现与案例研究

Manideep Reddy Chinthareddy

高级软件工程师，美国森特维尔
chmanideepreddy@gmail.com

2025年11月

arXiv:2601.02698v1 [cs.SE] 6 Jan 2026

**摘要**

AI 辅助开发者服务正日益嵌入现代集成开发环境，但企业必须确保这些工具在现有的身份、访问控制和治理要求范围内运行。模型上下文协议（MCP）使 AI 助手能够检索结构化的内部上下文，但其规范仅提供了最低限度的授权模型，并且缺乏关于如何集成企业单点登录的指导。本文介绍了一种实用架构，将 OAuth 2.0 和 OpenID Connect（OIDC）集成到启用 MCP 的开发环境中。文章说明了集成开发环境扩展如何获取并呈现令牌、MCP 服务器如何通过身份提供商验证令牌，以及如何利用作用域和声明实施最小权限访问。一个使用 Visual Studio Code、基于 Python 的 MCP 服务器和符合 OIDC 标准的身份提供商构建的原型实现证明了该方案的可行性。一项案例研究评估了身份验证延迟、令牌验证开销、运营方面的考虑因素以及 AI 特有风险。该方法为采用 AI 辅助开发工具的组织提供了一种可部署的模式，同时维护身份保证和可审计性。

**关键词：**模型上下文协议（MCP）、身份与访问管理（IAM）、OAuth 2.0、OpenID Connect（OIDC）、单点登录（SSO）、AI 辅助开发、企业安全性、开发工具。

## 1 引言

GitHub Copilot 和集成开发环境原生 AI 助手等 AI 辅助开发工具，已迅速从实验性功能发展为主流生产力工具。与此同时，模型上下文协议（MCP）已成为一种标准机制，可将结构化的领域专属上下文集成到 AI 工作流中，使开发工具能够通过 MCP 服务器查询内部文档、源代码仓库、构建系统及其他企业资源。

尽管潜力巨大，企业在采用方面仍持谨慎态度。组织必须确保 AI 辅助交互遵循现有的身份、访问控制和治理策略。2025 年 MCP 规范通过持有者令牌引入了基本授权模型，但并未规定如何颁发令牌、如何验证令牌，或如何将令牌映射到企业角色、属性和策略。此外，该规范也未提供具体方案，说明如何将基于 IDE 的 AI 助手——这类助手作为运行于开发者设备上的公共客户端——与企业单点登录及最小权限授权集成。

本文通过提出并展示一种将企业身份与支持 MCP 的 AI 开发者服务相集成的架构，弥补了这一实现空白。重点关注 MCP 特有的机制（例如授权字段和知名元数据端点）、在 IDE 环境中管理令牌所面临的挑战，以及如何针对 AI 辅助开发者工作流而非通用 Web 应用程序调整 OAuth 2.0/OIDC 模式。

本文已提交至 IEEE，供其考虑是否发表。版权可能在不另行通知的情况下转让，届时此版本可能不再可访问。

从实践者的角度来看，我们的目标是提供具体指导，帮助团队将其应用于任何符合标准的身份提供商。虽然本文以 Keycloak 作为实现示例，但该架构同样适用于其他企业身份与访问管理平台，例如 Azure Active Directory、Okta 或 PingFederate。

### 1.1 研究动机

开发人员越来越期待在集成开发环境中获得人工智能辅助体验，但企业必须保护专有代码、基础设施细节和内部文档。若缺乏明确定义的身份集成模式，MCP 服务器就可能被授予过多权限，或配置不一致，从而带来安全性、合规性和审计挑战。此外，AI 助手可以充当强大的中介，将 MCP 工具与大语言模型串联起来；如果助手绑定了权限过大的身份，就可能在无意间泄露或过度暴露敏感数据。

### 1.2 主要贡献

本文的主要贡献如下：

- 从企业身份视角分析 MCP 授权模型及其局限性，重点关注作为公共 OAuth 2.0 客户端的基于 IDE 的 AI 助手。
- 提出一种参考架构，利用 MCP 专用发现机制和授权机制，将 MCP 服务器与基于 OAuth 2.0/OIDC 的企业单点登录集成。
- 一份实施蓝图，采用符合 OIDC 标准的身份提供商（以 Keycloak 为例）、基于 Python 的 MCP 服务器和 Visual Studio Code 人工智能扩展，并说明如何将这一模式迁移到其他身份提供商。
- 一项案例研究，展示了切实可行的集成模式、延迟和令牌验证特征、人工智能特定安全风险以及运维考量。

与以往关于 Web 和移动应用程序 OAuth/OIDC 集成的研究不同，本文首次提出一种全面的身份集成模式，专为支持 MCP 的人工智能开发环境量身打造。现有身份与访问管理文献并未探讨作为公共客户端、负责协调多个企业资源的基于 IDE 的 AI 助手，在与 MCP 服务器交互时应如何获取、呈现或限定访问令牌的范围。本文提出的架构、实施蓝图和案例研究填补了这一空白，将 MCP 授权语义与企业单点登录统一起来，并介绍了用于令牌验证、基于 PKCE 的身份验证、范围到工具的映射以及安全的 IDE 令牌处理的具体工具模式。据我们所知，此前尚无研究提出可部署的端到端身份模型，以适合生产企业环境的方式连接 MCP 服务器、符合 OIDC 标准的身份提供商和 AI 辅助的 IDE 扩展。

本文其余部分安排如下：第 2 节介绍 MCP 和企业身份的背景；第 3 节描述所提出的架构；第 4 节介绍实现细节；第 5 节讨论案例研究与评估；第 6 节总结经验教训；第 7 节对全文作总结。

## 2 背景与上下文

### 2.1 模型上下文协议（MCP）

模型上下文协议为人工智能客户端与称为 MCP 服务器的工具和数据提供方交互，定义了一种标准化方式。近期的规范引入了授权字段，允许客户端在请求中附带持有者令牌，使服务器能够依据企业策略执行授权检查。MCP 还定义了发现机制，包括受 OAuth 保护的资源的知名端点，人工智能客户端可借此了解如何获取并出示访问令牌，以便与 MCP 服务器交互。

这些功能使 MCP 成为企业身份与访问管理与 AI 辅助开发之间的天然集成点，但该规范有意将令牌签发、验证策略和策略映射留给实施者决定。因此，企业必须制定自己的方案，将 MCP 服务器连接到现有的 OAuth 2.0/OIDC 基础设施。

### 2.2 企业身份与访问管理

企业环境通常依赖实现 OAuth 2.0 和 OpenID Connect（OIDC）的身份与访问管理（IAM）平台。这些平台充当身份提供商（IdP），负责签发令牌并执行身份验证策略，例如多因素身份验证（MFA）和条件访问。常见的身份提供商包括 Keycloak、Azure Active Directory、Okta、PingFederate 以及其他符合 OIDC 标准的平台。

虽然不同产品的配置细节各异，但底层构件相似：公共客户端和机密客户端、重定向 URI、作用域、声明、JSON Web 令牌（JWT）以及通过 JSON Web 密钥集（JWKS）进行的密钥管理。MCP 服务器可视为 OAuth 2.0 资源服务器，而 IDE 扩展和 AI 助手则充当公共客户端，必须使用适用于代码交换的证明密钥（PKCE）及其他原生应用保护措施。

### 2.3 集成开发环境中的 AI 辅助开发人员服务

现代集成开发环境承载着 AI 辅助扩展，这些扩展会与模型提供商和企业系统通信。要在这一流程中集成身份，需要谨慎处理公共客户端环境中的令牌，并确保 AI 助手的行为不超出经过身份验证的开发人员的权限范围。

与基于浏览器的应用程序不同，基于 IDE 的客户端通常运行在多种类型的开发人员工作站上，可能没有传统的后端组件，并且必须通过本地操作系统密钥链或加密存储来管理令牌。当这些客户端还集成了 MCP 时，它们实际上会成为编排器，能够在单一身份下串联多个 MCP 工具，从而放大作用域或角色配置错误所带来的影响。因此，为 AI 辅助开发者服务设计身份集成模式尤为重要。

## 3 企业身份集成架构

本节将介绍一种参考架构，该架构在基于 IDE 的 AI 助手场景中，将企业身份集成纳入 MCP 的授权模型。

### 3.1 高级组件

该架构由三个主要组件构成：

1. **IDE AI 扩展**：Visual Studio Code（VS Code）或 IntelliJ 中的插件，实现 MCP 客户端并充当公共 OAuth 2.0 客户端。2. **MCP 服务器**：资源服务器，用于验证令牌，并对 MCP 工具和资源应用授权策略。3. **身份提供商（IdP）**：符合 OAuth 2.0/OIDC 标准的单点登录（SSO）提供商（例如 Keycloak、Azure Active Directory 或 Okta），负责颁发令牌并实施身份验证策略。

### 3.2 身份验证与授权流程

交互过程如下：

1. IDE 扩展尝试在没有有效凭据的情况下调用 MCP 工具。

2. MCP 服务器响应 401 未授权，并返回 WWW-Authenticate 标头，指明需要持有者令牌。

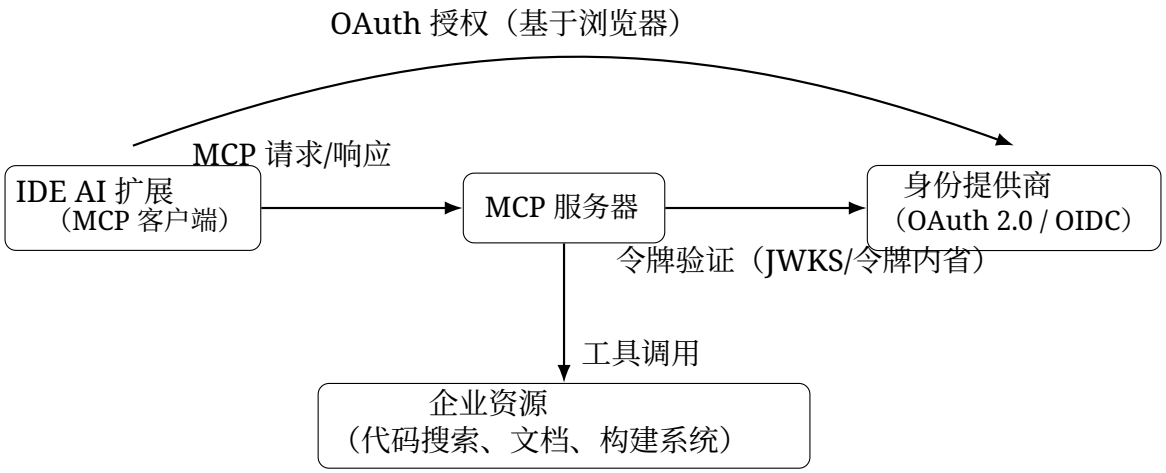



图 1：企业身份与 MCP 服务器、集成开发环境 AI 扩展及 OAuth/OIDC 身份提供商集成的高级架构。

表 1：角色与权限范围映射到 MCP 工具的示例。

| 角色 / 声明 | 作用域 | 允许使用的工具 |
|---|---|---|
| 开发人员 | mcp.docs.read, mcp.code.search | 文档搜索、代码搜索，构建系统查询 |
| 承包商 | mcp.docs.read | 仅限文档搜索 |
| 操作员 | mcp.ops.read | 部署和状态查询 |

3. 集成开发环境使用 PKCE，针对企业身份提供商触发基于浏览器的 OAuth 2.0/OIDC 流程。

4. 身份验证成功后，集成开发环境会获取访问令牌，并可选择获取刷新令牌。

5. 后续 MCP 请求会将令牌包含在 MCP 规范定义的授权字段中。

6. MCP 服务器会验证令牌，评估作用域和声明，并批准或拒绝所请求的操作。

图 3 展示了 VS Code 与企业身份提供商交互时，在 OAuth/OIDC 身份验证和客户端注册过程中的典型用户体验。用户在浏览器中成功完成身份验证后，VS Code 会请求用户同意，以便针对身份提供商对 MCP 服务器进行身份验证；如有必要，还会引导用户手动完成客户端注册。

### 3.3 授权模型

企业可以实施基于角色的访问控制（RBAC）、基于属性的访问控制（ABAC），或基于作用域的策略，将令牌声明映射到获准使用的 MCP 工具和资源。例如，开发者角色可以调用文档和代码搜索工具，而承包商角色只能使用文档工具。

表 1 展示了从角色和作用域到获准使用的 MCP 工具的简单映射。

### 3.4 威胁模型和局限性

所提出的架构假设：

- IDE 扩展是诚实但好奇的公共 OAuth 2.0 客户端，不包含客户端密钥。
- 在可用的情况下，访问令牌会使用操作系统密钥链功能进行安全存储。

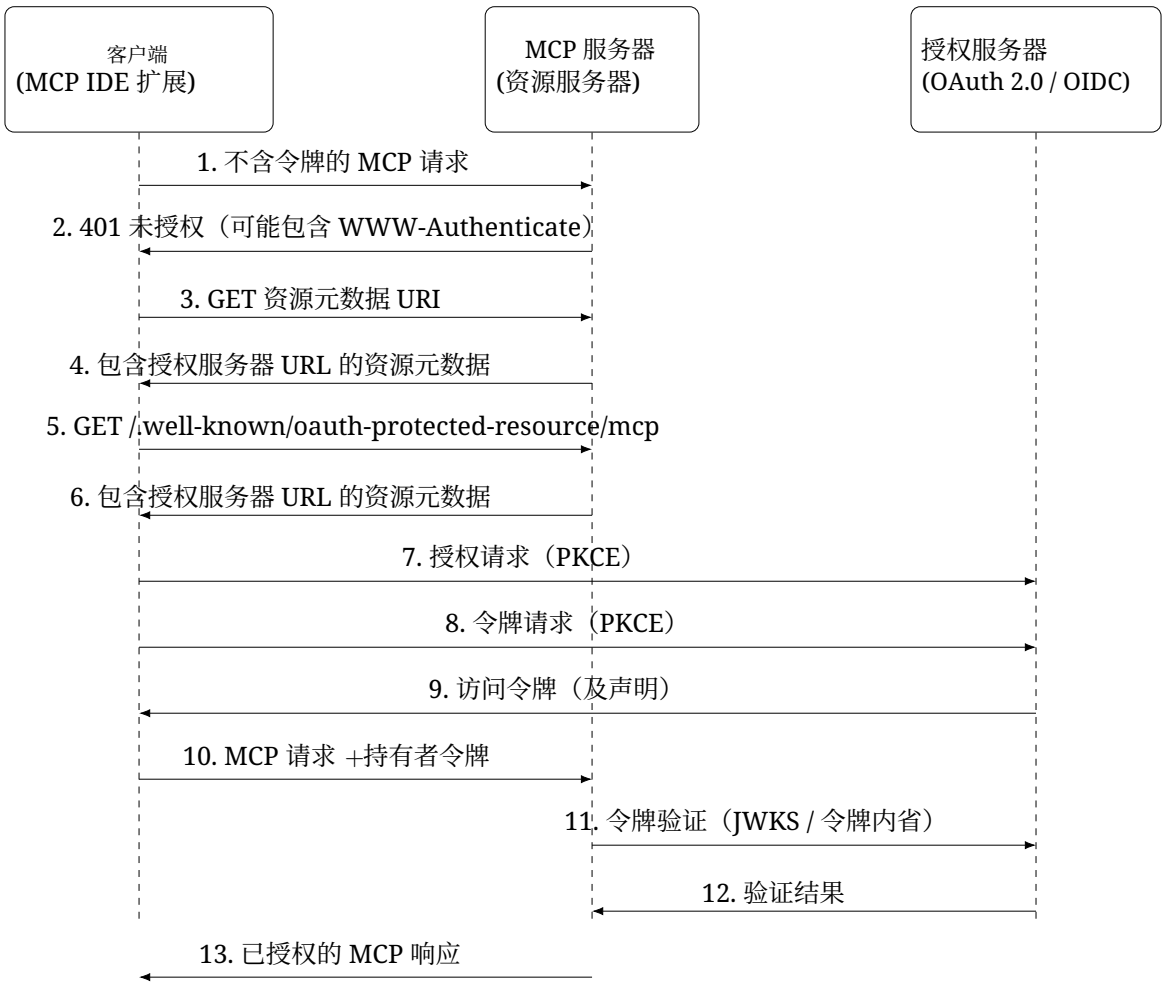



图 2：端到端 MCP 授权序列，包括元数据发现、OAuth 令牌签发、令牌验证和受保护资源访问 [1]。

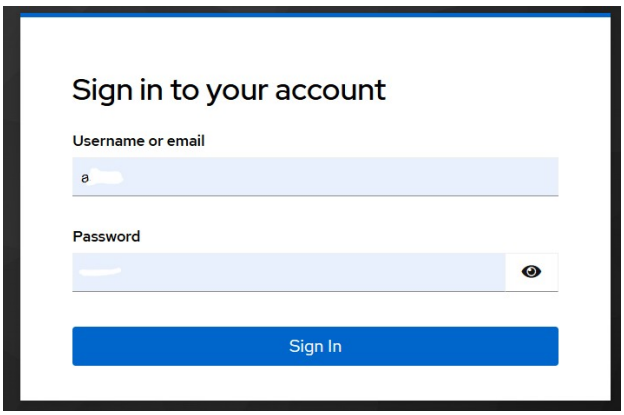
- 集成开发环境、MCP 服务器与身份提供商之间的网络通道均使用 TLS 进行保护。

如果集成开发环境受到入侵，攻击者可能会获取短期访问令牌，并在相关作用域和角色的权限范围内调用 MCP 工具。为限制影响，企业应优先采用较短的令牌有效期，限制刷新令牌签发，并强制执行细粒度作用域。如果 MCP 服务器受到入侵，其对下游企业资源的访问会受到自身服务凭据和令牌验证策略的约束，但运行时审计日志对于事件响应仍然至关重要。

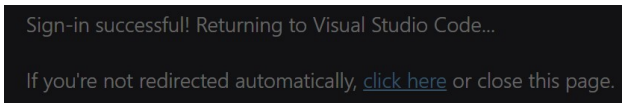
除了传统的 OAuth 威胁场景外，AI 辅助开发工具还带来了其他需要考虑的因素。由于 AI 助手可以根据自然语言提示自主串联 MCP 工具调用，任何过度授权的作用域配置都可能导致对代码、文档或运营数据的广泛且出乎意料的访问。为降低此类风险，应按照明确的数据边界和角色划分对 MCP 工具进行分组并配置作用域；敏感工具（例如，可访问生产环境配置或客户数据的工具）应要求使用独立的作用域，而这些作用域很少授予日常开发者角色。

这些限制和风险是扩展至 AI 助手的基于 OAuth 的架构的典型特征，必须在企业风险评估中予以考虑。

近期关于人工智能辅助开发环境的研究指出，当工具自主串联特权操作或处理敏感的内部内容 [9]，[10]，[11]时，会产生运营和治理风险。Fang 等人证明，在获得工具访问权限后，大语言模型智能体能够自主利用现实世界漏洞 [13]，，这凸显了严格访问边界的必要性。此外，虽然 AgentBound [12] 等运行时方法侧重于执行策略，但我们的架构强调上游身份保证，以从一开始就防止未经授权的工具访问。



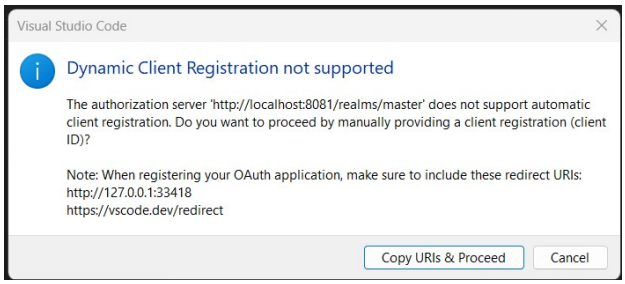

(a) 身份提供商登录页面



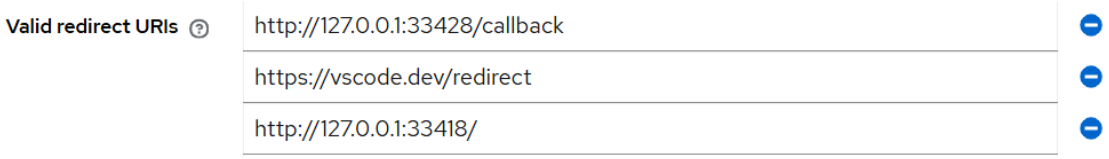

(b) 登录成功确认




(c) VS Code 授权提示




(d) 手动客户端注册提示




(e) 在身份提供商中配置重定向 URI

图 3：VS Code 使用企业身份提供商对 MCP 服务器进行身份验证时，开发人员和管理员所经历的典型 OAuth/OIDC 流程。

# 4 实现

本节介绍一种实用实现方案，采用符合 OIDC 标准的身份提供商（以 Keycloak 为例）、基于 FastAPI 的 MCP 服务器，以及 VS Code 人工智能扩展。虽然具体截图和配置名称以 Keycloak 为例，但稍作调整声明和客户端注册流程后，同一模式也适用于其他身份提供商。

## 4.1 身份提供商配置

身份提供商配置了一个专用于开发人员工具的领域或应用程序、一个代表 IDE 扩展的公共客户端，以及对应开发人员、操作员和承包商角色的角色或组。重定向 URL 包括 VS Code 使用的本地 HTTP 端点，以及备用 HTTPS 重定向。对于 Keycloak 以外的身份提供商，通常需要进行等效配置，包括注册公共客户端、启用 PKCE，并添加 IDE 的重定向 URI。

## 4.2 MCP 服务器实现

MCP 服务器使用中间件从授权字段中提取令牌，通过身份提供商的 JWKS 端点对其进行验证，并将声明映射到适用于每个 MCP 工具的内部授权规则。列表 1 展示了一个简单的配置：在身份验证墙后运行 MCP 服务器。此配置可确保 MCP 服务器向 MCP 客户端返回特定标头，告知客户端其尚未通过身份验证，并说明如何进行身份验证。

```
应用 = FastMCP(
    "示例 MCP 服务器",
    端口=8000,
    令牌_验证器=RoleAwareTokenVerifier()      ，
    身份验证=AuthSettings(
        颁发者_URL="http://localhost:8081/realms/master",
        资源_服务器_URL="http://localhost:8000/mcp",
        必需_作用域=["openid",          "profile"]
    )
)
```

清单 1：基于 OIDC 授权的 FastMCP 应用初始化。

```
fastapi = app.streamable_http_app() fastapi_lifespan = FastAPI(lifespan=
lifespan) @fastapi_lifespan.get("/.well-known/oauth-protected-resource")
async def oauth_protected_resource(): return JSONResponse({ "resource": "
http://localhost:8000/mcp", "scopes_supported": ["openid", "profile"],"
authorization_servers": ["http://localhost:8081/realms/master"],"bearer_
methods_supported": ["header","body"],})
```

清单 2：MCP 服务器公布的受 OAuth 保护的资源描述符。

```
from fastapi import Depends, Security from fastapi.security import
HTTPBearer, HTTPAuthorizationCredentials 安全性 = HTTPBearer()
@app.tool() def mcp_get(access_token: 访问令牌 =Depends( require_作用域("
mcp.docs.read"))): ...
```

清单 3：配置基于范围的工具级访问控制。

清单 1 中的 RoleAwareTokenVerifier 是一个简单直接的令牌验证器实现，它使用 IdP 的密钥验证 JWT，并将声明映射到内部角色。其逻辑遵循 OAuth 2.0 资源服务器的标准模式，因篇幅所限，此处省略。

清单 2 中的代码定义了一个受 OAuth 保护的资源描述符。MCP 客户端（此处为 VS Code）会检索此文档，以发现能够为 MCP 资源颁发令牌的授权服务器。端点 URL 必须与 MCP 规范完全一致，包括大小写。

如果需要配置工具级范围，清单 3 中的配置展示了如何为特定 MCP 工具强制执行范围要求。

清单 4 展示了身份验证和工具调用成功期间服务器日志的摘录。

### 4.3 VSCode / 集成开发环境插件行为

IDE 扩展在收到 MCP 服务器返回的未授权响应时，会启动 OAuth 流程、安全地存储令牌，并在后续 MCP 请求中附加这些令牌。身份验证完成后，扩展便可成功枚举并调用 MCP 工具。实际使用中，首次登录并授权后，这一行为对开发人员而言基本是透明的。

```
INFO "POST /mcp HTTP/1.1" 401 未授权
信息 "GET /.well-known/oauth-protected-resource HTTP/1.1" 200 OK
正在验证令牌...
已验证用户身份：a**********
信息“POST /mcp HTTP/1.1” 202 已接受
信息 "POST /mcp HTTP/1.1" 200 正常
```

清单 4：MCP 身份验证日志示例。

# 5 案例研究：保护基于 MCP 的 AI 开发人员服务 es

本案例研究在一个具有代表性的企业环境中评估了所提出的身份集成架构。目标是验证：（1）安全态势；（2）实际性能；以及（3）启用基于 MCP 的 AI 辅助开发工具后，对开发者工作流程的整体影响。

## 5.1 场景与环境

评估环境由以下组件构成：

- **身份提供商（IdP）：**一台最初部署在本地笔记本电脑上用于演示、之后部署到 AWS ECS 集群上的 OAuth 2.0/OIDC 服务器，负责强制执行多因素认证和条件访问策略。本实现使用了 Keycloak，但也可以采用其他符合 OIDC 标准的身份提供商复现相同模式。

- **MCP 服务器：**一个基于 FastAPI 的资源服务器，最初在本地运行于反向代理之后，随后容器化并部署到 AWS ECS。

- **IDE 客户端：** AVSCode 扩展（本例中为 GitHub Copilot 扩展）实现了 MCP 客户端，并作为使用 PKCE 的公共 OAuth 2.0 客户端。

- **企业资源：** 通过 MCP 工具提供的内部文档索引、架构标准元数据、代码目录搜索工具和构建系统查询。

通过在令牌验证和 MCP 工具调用前后记录服务器端时间戳，并在正常网络条件下进行多次运行后取平均值，收集延迟测量数据。

## 5.2 身份验证行为与开发人员影响

本案例研究的一个重点是确定 OAuth/OIDC 身份验证是否给开发人员带来了明显的不便或延迟。

在实践中，身份验证延迟并不是一个值得担忧的问题。由于启用 MCP 的工具几乎完全由内部开发人员访问，因此整体身份验证负载较低，企业 IAM 基础设施能够轻松应对。开发人员通常会在会话开始时进行一次身份验证，之后所有后续 MCP 交互都会重复使用缓存凭据。

即使最初的 OAuth 流程因浏览器重定向或多因素认证提示而略有延迟，这种额外耗时也只会在每个会话中出现一次，不会对可用性造成持续影响。OAuth/OIDC 带来的身份保证、集中式治理和可追溯性，远远超过了这点时间成本。

## 5.3 令牌验证与运行时性能

完成身份验证后，MCP 服务器会使用身份提供商的 JWKS 端点验证令牌。案例研究期间收集的性能测量结果显示：

- **JWKS 缓存命中延迟：**每次验证约 5 毫秒（几乎即时 ）.
- **JWKS 缓存未命中延迟：**25–35 毫秒（仅适用于初始密钥获取）。

- **MCP 工具调用延迟：**端到端 90–120 毫秒，主要耗时在资源处理和 AI 辅助分析，而非身份验证。

这些结果证实，令牌验证带来的开销可以忽略不计，不会对开发者工作流响应能力产生负面影响。

### 5.4 安全性与运营观察

在评估过程中，我们得出了几项重要的安全性和运营方面的认识：

- **PKCE 至关重要**，可用于保护 IDE 环境中的公共客户端。
- **基于范围的授权**（例如，mcp.docs.read、mcp.code.search）提供了精细的控制，符合企业最小权限策略。
- **刷新令牌的使用** 受到严格限制；短期访问令牌降低了凭据盗窃的风险。
- **审计日志记录**工具调用以及经过身份验证的用户身份，能够显著改善事件调查和合规报告。
- **客户端注册**阻力是最常见的入门问题，尤其是在不支持动态注册的情况下。
- **重定向 URL 白名单**是常见的配置缺口，因为 VSCode 使用本地回调 URL，必须将其明确添加到身份提供商。
- **AI 特定风险管理**要求将特别敏感的 MCP 工具（例如暴露生产环境操作或机密数据集的工具）映射到专用作用域和角色，并将其限制在少数用户范围内。

总体而言，这些发现表明，在推广过程中，平台工程、身份与访问管理以及开发者体验团队之间进行协调至关重要。

### 5.5 总结

案例研究证实，为基于 MCP 的人工智能开发者工具集成企业身份既切实可行，在运维方面的负担也很轻。OAuth/OIDC 身份验证带来的额外开销微乎其微，同时显著提升了身份保证、访问治理和可审计性。这些结果表明，在生产级企业环境中安全部署支持 MCP 的开发者工作流是可行的。

## 6 条经验教训

关键经验包括：明确设计作用域、实施稳健的令牌验证、在 IDE 环境中安全存储凭据，以及平台、安全和开发者体验团队之间的协作。从实践者的角度来看，计划进行类似部署的组织应当：

- 为所有基于 IDE 的 OAuth 2.0 客户端启用 PKCE。
- 设计细粒度作用域（例如 mcp.docs.read、mcp.code.search），并将其映射到 MCP 工具。如果只能优先考虑一项设计决策，那么从一开始就做好作用域设计，是防止 AI 助手获得过多权限的最有效方法。
- 优先使用短期访问令牌，并限制刷新令牌的使用。

- 在客户端注册期间，将本地集成开发环境的重定向 URL 加入身份提供商的白名单，并向开发人员清晰说明。虽然本文以 Keycloak 为例进行说明，但这种集成模式完全适用于任何符合 OIDC 标准的企业级提供商。Azure Active Directory、Okta 和 PingFederate 等平台遵循相同的 OAuth 2.0 和 OIDC 基本语义——公共客户端注册、强制执行 PKCE、基于 JWKS 的令牌验证，以及范围到资源的映射——因此无需修改协议逻辑，即可直接移植这一架构。
- 将 MCP 工具调用日志集成到现有的安全监控和审计流水线中，包括将调用归属到具体用户和作用域。

在集成开发环境和服务器日志中提供清晰可见的反馈，有助于操作员快速诊断配置问题，并缩短解决入门问题所需的时间。

## 7 结论

支持 MCP 的人工智能开发者工具为直接从集成开发环境访问企业上下文提供了强大的新能力。MCP 授权模型奠定了必要的基础，但企业必须将其与自身的身份基础设施集成，才能满足现实环境中的安全性和治理要求。

本文介绍了一种架构，将基于 OAuth 2.0/OIDC 的单点登录、MCP 服务器和 IDE 扩展整合为适用于生产环境的统一安全解决方案。通过将基于 IDE 的 AI 助手视为公共 OAuth 2.0 客户端，利用 MCP 的授权和发现机制，并对 MCP 工具应用基于范围和角色的策略，组织便能安全地向 AI 工作流开放内部开发人员资源。

实现模式和案例研究结果表明，部署所需组件只需适度的运维投入，且运行时开销可以忽略不计。实践者可以采用本文所述的模式——不受任何特定身份提供商厂商的限制——加速采用支持 MCP 的人工智能开发者服务，同时在整个软件交付环境中维持身份保证、最小权限和可审计性。

## 参考文献

[1]模型上下文协议（MCP）——基本授权规范。 [在线]。可访问：https://modelcontextprotocol.io/specification/draft/basic/authorization。访问日期：2025年11月19日。[2] D. Hardt，《OAuth 2.0 授权框架》，RFC 6749，互联网工程任务组，2012年10月。[3]OpenID 基金会，《OpenID Connect Core 1.0》，2014年。[在线]。可访问：https://openid.net/specs/openid-connect-core-1.html[4] Nat Sakimura、John Bradley 和 Naveen Agarwal，《代码交换证明密钥（PKCE）》，RFC 7636，互联网工程任务组，2015年9月。[5] W. Denniss 和 J. Bradley，《适用于原生应用的 OAuth 2.0》，RFC 8252，互联网工程任务组，2017年10月。[6]红帽，《Keycloak 文档》，2024年。[在线]。可访问：https://www.keycloak.org/documentation[7] V. C. Hu 等，《基于属性的访问控制（ABAC）指南》，NIST 特别出版物 800-162，2014年。[8] M. Jones、J. Bradley 和 N. Sakimura，《JSON Web 令牌（JWT）》，RFC 7519，2015年5月。[9] S. Amershi 等，《人机交互指南》，ACM CHI 计算机系统中的人为因素会议论文集，2019年。

[10] V. S. Narajala 和 I. Habler，《模型上下文协议（MCP）的企业级安全：框架与缓解策略》，arXiv 预印本 arXiv:2504.08623，2025年。[11] H.Errico、J. Ngiam 和 S. Sojan，《保障模型上下文协议（MCP）的安全：风险、控制措施与治理》，arXiv 预印本 arXiv:2511.20920，2025年。[12] Christoph B¨uhler、Matteo Biagiola、Luca Di Grazia 和 Guido Salvaneschi，《AI 代理执行的安全保障》，arXiv 预印本 arXiv:2510.21236，2025年。[13]Richard Fang、Rohan Bindu、Akul Gupta 和 Daniel Kang，《大语言模型智能体可以自主利用零日漏洞》，arXiv 预印本 arXiv:2404.08144，2024年。