perplexity

Example: MCP Server in Python with Authentication Settings

The official MCP Python SDK provides a flexible way to build MCP servers, but by default, authentication is not enabled out-of-the-box in the simplest examples[1] [2] [3]. However, the SDK and related documentation suggest that you can configure authentication (such as API keys or OAuth) by adjusting the server's initialization parameters or by integrating with external authentication middleware[4].

Below is a sample Python MCP server using the `FastMCP` class, with a placeholder for authentication settings. You can adapt this for your needs by plugging in your preferred authentication mechanism.

```
from mcp.server.fastmcp import FastMCP

# Example: Adding authentication settings via custom middleware or config
def authenticate(request):
    # Replace this logic with your actual authentication (API key, OAuth, etc.)
    api_key = request.headers.get("x-api-key")
    if api_key != "your-secret-key":
        raise PermissionError("Unauthorized")

# Initialize the MCP server with custom auth (pseudo-code for illustration)
mcp = FastMCP(
    name="SecureMCPServer",
    stateless_http=True,
    # If the SDK exposes an authsettings or middleware parameter, use it here
    # Example: authsettings={"api_key": "your-secret-key"}  # (Pseudo-code)
)

# Example tool with authentication check
@mcp.tool(description="A secure echo tool")
def secure_echo(message: str, request=None) -> str:
    authenticate(request)
    return f"Echo: {message}"

if __name__ == "__main__":
    mcp.run()
```

Notes:

- The actual method to inject authentication (such as `authsettings` or middleware) may depend on the version of the SDK and your transport (HTTP, stdio, etc.). The SDK currently does not document a direct `authsettings` parameter, so you may need to implement authentication as a decorator or middleware around your tool/resource functions[1] [3] [4].
- For HTTP servers, you can check headers or tokens in the request object. For stdio or other transports, you might handle authentication at the session or message level.

- For production use, consider integrating with OAuth or API key management as shown in cloud deployment samples[4].

References for Further Customization

- Official MCP Python SDK examples[1]
- DigitalOcean MCP server tutorial[2]
- Azure Functions MCP server with built-in auth[4]

If you need a specific authentication method (e.g., OAuth2, JWT, API Key), provide more details and I can tailor the code accordingly.

⁂

Can you give a client program in python that calls this server tool?

Absolutely! Here's a simple Python client that calls the `secure_echo` tool on your MCP server, passing the required API key for authentication. This example assumes your MCP server is running locally on the default HTTP port (`8000`) and you're using HTTP transport.

Python MCP Client Example

```
import requests

# MCP server URL and endpoint for the tool
MCP_SERVER_URL = "http://localhost:8000/tools/secure_echo"
API_KEY = "your-secret-key"  # Must match the server's expected key

def call_secure_echo(message):
    headers = {
        "x-api-key": API_KEY,  # Custom header for authentication
        "Content-Type": "application/json"
    }
    payload = {
        "message": message
    }
    response = requests.post(MCP_SERVER_URL, json=payload, headers=headers)
    if response.status_code == 200:
        print("Server response:", response.json())
    else:
        print(f"Error {response.status_code}: {response.text}")

if __name__ == "__main__":
    call_secure_echo("Hello from the client!")
```

How it works:

- The client sends a POST request to the `/tools/secure_echo` endpoint.
- The request includes the `x-api-key` header for authentication.
- The server validates the key and, if correct, echoes the message.

Troubleshooting

- Ensure the server is running and accessible at `localhost:8000`.
- The endpoint path (`/tools/secure_echo`) may differ based on your server setup. Adjust if necessary.
- The API key in the client must match the one expected by the server.

Let me know if you need an example for a different transport (e.g., stdio, websocket) or want to use a different authentication method!

Python MCP OAuth Example

Below is a concise example of how to implement OAuth 2.1 in a Python MCP client and server context, following the latest MCP specification. This example demonstrates the key steps: client registration, authorization code flow (with PKCE), and token exchange. It assumes you are using an HTTP-based MCP server that supports OAuth endpoints as per the spec[5] [6].

1. MCP Server (OAuth endpoints)

If you're using an MCP server framework, you will need to expose OAuth endpoints (`/authorize`, `/token`) and integrate with an OAuth provider (such as Auth0, Google, or GitHub). Many MCP server implementations or frameworks provide this, or you can use middleware such as `authlib` in Python.

Example server snippet (using FastAPI and Authlib):

```
from fastapi import FastAPI, Request
from authlib.integrations.starlette_client import OAuth

app = FastAPI()
oauth = OAuth()

# Register your OAuth provider (e.g., Auth0)
oauth.register(
    name='auth0',
    client_id='YOUR_CLIENT_ID',
    client_secret='YOUR_CLIENT_SECRET',
    server_metadata_url='https://YOUR_DOMAIN/.well-known/openid-configuration',
    client_kwargs={'scope': 'openid profile email'},
)
```

```
@app.get('/authorize')
async def authorize(request: Request):
    redirect_uri = request.url_for('auth')
    return await oauth.auth0.authorize_redirect(request, redirect_uri)

@app.get('/auth')
async def auth(request: Request):
    token = await oauth.auth0.authorize_access_token(request)
    user = await oauth.auth0.parse_id_token(request, token)
    # Issue your own MCP token here, bind to user session
    return {"access_token": token['access_token'], "user": user}

@app.post('/token')
async def token(request: Request):
    # Implement token exchange logic here
    ...
```

This is a minimal example. In production, you should handle session management, error handling, and secure token storage as per the MCP spec[5] [7] [6].

2. MCP Client (OAuth 2.1 Flow with PKCE)

The client must:

- Discover the server's OAuth endpoints (metadata discovery)
- Register itself (Dynamic Client Registration, if supported)
- Initiate the OAuth Authorization Code flow with PKCE
- Exchange the code for an access token
- Use the token in subsequent MCP requests

Example client using `requests` and `requests_oauthlib`:

```
from requests_oauthlib import OAuth2Session

# Discover endpoints (metadata discovery)
MCP_SERVER = "https://your-mcp-server.com"
metadata = requests.get(f"{MCP_SERVER}/.well-known/oauth-authorization-server").json()
authorization_base_url = metadata['authorization_endpoint']
token_url = metadata['token_endpoint']

client_id = 'YOUR_CLIENT_ID'
redirect_uri = 'http://localhost:8000/auth/callback'
scope = ['openid', 'profile', 'email']

# PKCE support
oauth = OAuth2Session(client_id, redirect_uri=redirect_uri, scope=scope, code_challenge_m

# Step 1: Redirect user to MCP server's /authorize endpoint
authorization_url, state = oauth.authorization_url(authorization_base_url)
print("Go to the following URL and authorize:", authorization_url)
```

```
# Step 2: After user authorizes, MCP server redirects to your redirect_uri with a code
# You need to capture the code from the redirect (in a web app, this is a route handler)
redirect_response = input("Paste the full redirect URL here: ")

# Step 3: Exchange code for token
token = oauth.fetch_token(token_url, authorization_response=redirect_response, client_sec

print("Access token:", token['access_token'])

# Step 4: Use the access token in your MCP requests
headers = {"Authorization": f"Bearer {token['access_token']}"}
response = requests.post(f"{MCP_SERVER}/tools/secure_echo", json={"message": "Hello, OAut
print(response.json())
```

Key Points

- **PKCE** is required for security in OAuth 2.1 flows[6].
- **Metadata Discovery** allows the client to find OAuth endpoints automatically[6].
- **Dynamic Client Registration** can be used for seamless onboarding if the server supports it[6].
- **Third-party providers** (Auth0, Google, GitHub) can be integrated via the server, which issues its own tokens to the client[7] [6].

> **The MCP spec encourages using OAuth 2.1 with PKCE, metadata discovery, and dynamic client registration for secure, user-friendly authorization[5] [6].**

If you need a **full working demo** or want to see how to wire this up with a specific MCP server framework, let me know your stack and I can provide a more tailored example.

⁂

Can you give a more complete server code? The code you have given is partial

Absolutely! Here's a **complete example** of an MCP-like server in Python that uses **OAuth 2.0** for authentication, using **FastAPI** and **Authlib**. This example demonstrates:

- How to protect your MCP tool endpoint with OAuth2.
- How to set up OAuth2 with Authlib and FastAPI.
- How to verify access tokens on incoming requests.

This example uses **Auth0** as the OAuth provider, but you can adapt it to any provider that supports OpenID Connect.

Complete Python MCP Server with OAuth2 (FastAPI + Authlib)

1. Install dependencies

```
pip install fastapi uvicorn authlib python-dotenv
```

2. `.env` file (for secrets)

```
AUTH0_DOMAIN=your-auth0-domain.auth0.com
AUTH0_CLIENT_ID=your-client-id
AUTH0_CLIENT_SECRET=your-client-secret
API_AUDIENCE=your-api-audience
```

3. `server.py`

```
import os
from fastapi import FastAPI, Depends, HTTPException, status, Request
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from dotenv import load_dotenv
from authlib.jose import jwt, JoseError
import requests

load_dotenv()

AUTH0_DOMAIN = os.getenv("AUTH0_DOMAIN")
API_AUDIENCE = os.getenv("API_AUDIENCE")

app = FastAPI()
bearer_scheme = HTTPBearer()

# Get JWKS for Auth0 domain
JWKS_URL = f"https://{AUTH0_DOMAIN}/.well-known/jwks.json"
jwks = requests.get(JWKS_URL).json()

def get_public_key(token):
    header = jwt.get_unverified_header(token)
    for key in jwks["keys"]:
        if key["kid"] == header["kid"]:
            return key
    raise Exception("Public key not found.")

def verify_jwt_token(credentials: HTTPAuthorizationCredentials = Depends(bearer_scheme)):
    token = credentials.credentials
    try:
        key = get_public_key(token)
        claims = jwt.decode(token, key)
        claims.validate()
        # Optionally check audience
        if API_AUDIENCE not in claims["aud"]:
            raise HTTPException(status_code=403, detail="Invalid audience")
        return claims
```

```
    except (JoseError, Exception) as e:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid or expired token",
        )

@app.get("/")
def root():
    return {"message": "MCP OAuth2 Server is running."}

@app.post("/tools/secure_echo")
def secure_echo(request: Request, claims=Depends(verify_jwt_token)):
    data = await request.json()
    message = data.get("message")
    user = claims.get("sub", "unknown user")
    return {"echo": message, "user": user}

# For development: run with `uvicorn server:app --reload`
```

4. How it works

- The `/tools/secure_echo` endpoint is protected by OAuth2 Bearer authentication.
- The server fetches the JWKS (public keys) from your Auth0 domain and uses them to verify incoming JWTs.
- If the token is valid and the audience matches, the request proceeds.
- The server echoes the message and returns the user's subject (`sub`) from the JWT.

5. Testing

Get a token from your OAuth provider (Auth0, Google, etc.) and call the endpoint:

```
curl -X POST http://localhost:8000/tools/secure_echo \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"message": "Hello, OAuth!"}'
```

Notes

- You can use any OAuth2 provider; just adjust the JWKS URL and audience.
- This is a minimal, production-grade pattern for OAuth2-protected APIs in Python.
- For a real MCP server, you would add more endpoints and MCP-specific logic, but this pattern is the foundation for secure, standards-based authentication.

If you want a **full MCP protocol implementation** or have a specific provider or workflow in mind, let me know!

What are the key components missing from the partial server code

The partial server code provided earlier for an MCP server with OAuth authentication was illustrative but **lacked several key components necessary for a fully functional, production-ready OAuth-protected server**. Here are the main missing components:

Key Missing Components in the Partial Server Code

- **Complete OAuth2 Authorization Flow**
  The code only included endpoint stubs for `/authorize` and `/token`, but did not implement the full logic for handling OAuth2 Authorization Code flow, including redirecting users, handling callbacks, and securely issuing or verifying tokens.
- **User Session and State Management**
  There was no mechanism for maintaining user sessions, storing state parameters, or handling CSRF protection during the OAuth flow, which are essential for secure authentication.
- **Token Validation Middleware**
  The code did not include middleware or dependency logic to extract, validate, and decode the OAuth access token from incoming requests before allowing access to protected endpoints.
- **JWKS (JSON Web Key Set) Fetching and Caching**
  For validating JWTs, the server should fetch the provider's public keys (JWKS) and cache them. The partial code did not demonstrate this, which is critical for verifying token signatures.
- **

⁂

Can you modify the above server program to accommodate an azure app registration as the client identifier

To modify your MCP server program to use **Azure App Registration** as the client identifier (i.e., authenticate using Microsoft Entra ID/Azure AD), you should:

- Register your API in Azure (get the **Application (client) ID** and **Directory (tenant) ID**)[8] [9].
- Use a library like `fastapi-azure-auth` for easy and secure Azure AD authentication in FastAPI[10] [9].
- Protect your endpoints with the Azure authentication dependency, referencing your Azure App Registration's client ID.

Below is a **complete FastAPI server example** using Azure App Registration for OAuth2 authentication. This code uses the `fastapi-azure-auth` package for simplicity and follows best

practices.

1. Install Required Packages

```
pip install fastapi uvicorn fastapi-azure-auth
```

2. Set up Azure App Registration

- Go to **Azure Portal** → **Azure Active Directory** → **App registrations**.
- Register a new application (single-tenant or multi-tenant as needed).
- Note the **Application (client) ID** and **Directory (tenant) ID**[8] [9].
- Under **Expose an API**, add a scope (e.g., `user_impersonation`).
- Under **Authentication**, add your redirect URIs as needed.

3. Create `server.py`

```
import os
from fastapi import FastAPI, Depends
from fastapi_azure_auth import SingleTenantAzureAuthorizationCodeBearer

# Set these as environment variables or hardcode for testing
CLIENT_ID = os.getenv("AZURE_CLIENT_ID", "your-azure-client-id")
TENANT_ID = os.getenv("AZURE_TENANT_ID", "your-azure-tenant-id")

azure_scheme = SingleTenantAzureAuthorizationCodeBearer(
    app_client_id=CLIENT_ID,
    tenant_id=TENANT_ID,
    scopes={f'api://{CLIENT_ID}/user_impersonation': 'user_impersonation'},
)

app = FastAPI(
    swagger_ui_oauth2_redirect_url='/oauth2-redirect',
    swagger_ui_init_oauth={
        'usePkceWithAuthorizationCodeGrant': True,
        'clientId': CLIENT_ID,
    },
)

@app.get("/")
def root():
    return {"message": "MCP OAuth2 Server with Azure App Registration is running."}

@app.post("/tools/secure_echo")
async def secure_echo(data: dict, user=Depends(azure_scheme)):
    message = data.get("message", "")
    # user contains claims from the Azure token
    return {"echo": message, "user": user.get("preferred_username", "unknown")}
```

```
# To run: uvicorn server:app --reload
```

4. How it Works

- The `SingleTenantAzureAuthorizationCodeBearer` dependency validates incoming tokens against your Azure App Registration[10] [9].
- The `/tools/secure_echo` endpoint is protected: only requests with a valid Azure AD access token (for your app) are allowed.
- The client must obtain a token from Azure AD using your app's client ID and include it as a Bearer token in requests.

5. References & Further Reading

- fastapi-azure-auth documentation for multi-tenant, B2C, and advanced scenarios[10] [9].
- Microsoft: Register a client application in Microsoft Entra ID[8].

Summary Table

Setting	Value from Azure Portal
CLIENT_ID	Application (client) ID
TENANT_ID	Directory (tenant) ID
Scope	api://{CLIENT_ID}/user_impersonation

This setup ensures your MCP server authenticates requests using Azure App Registration as the client identifier, leveraging Microsoft Entra ID for secure OAuth2 flows[10] [8] [9].

⁂

Can you give a client program for this and tell how to use?

Certainly! Below is a **Python client program** that authenticates using your Azure App Registration (Microsoft Entra ID/Azure AD) and calls your protected FastAPI MCP server endpoint. This example uses the [MSAL (Microsoft Authentication Library) for Python][11] [12] [13], which is the recommended way to obtain access tokens in Python for Azure-protected APIs.

Client Program Example

```
import msal
import requests

# === Azure App Registration and API Details ===
CLIENT_ID = "your-azure-client-id"           # Application (client) ID from Azure portal
TENANT_ID = "your-azure-tenant-id"           # Directory (tenant) ID from Azure portal
AUTHORITY = f"https://login.microsoftonline.com/{TENANT_ID}"
SCOPE = [f"api://{CLIENT_ID}/user_impersonation"]  # Scope exposed by your API
API_ENDPOINT = "http://localhost:8000/tools/secure_echo"  # Your FastAPI MCP server endpo

# === Acquire Token Interactively (opens browser) ===
app = msal.PublicClientApplication(CLIENT_ID, authority=AUTHORITY)
result = None

accounts = app.get_accounts()
if accounts:
    # Try to acquire token silently
    result = app.acquire_token_silent(SCOPE, account=accounts[^7_0])

if not result:
    # Interactive login (opens browser window)
    result = app.acquire_token_interactive(scopes=SCOPE)

if "access_token" in result:
    token = result["access_token"]
    print("Access token acquired.")

    # === Call the protected API endpoint ===
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"message": "Hello from Azure AD authenticated client!"}
    response = requests.post(API_ENDPOINT, json=payload, headers=headers)

    print("Status code:", response.status_code)
    print("Response:", response.json())
else:
    print("Failed to acquire token.")
    print(result.get("error"))
    print(result.get("error_description"))
    print(result.get("correlation_id"))
```

How to Use

1. **Install required packages**:

   ```
   pip install msal requests
   ```

2. **Configure your Azure App Registration**:
   - Make sure your app registration exposes the scope `user_impersonation` (or as configured in your API).
   - Add a "Redirect URI" for "Mobile & desktop applications" (e.g., `http://localhost`).

3. **Set your variables**:
    - Replace `your-azure-client-id` and `your-azure-tenant-id` with your actual Azure values.
    - Ensure the `SCOPE` matches the scope exposed by your API.
4. **Run your MCP server** (as described in the previous answer).
5. **Run this client script**:
    - The first time, it will open a browser window for you to sign in with your Microsoft account.
    - After signing in, it will acquire an access token and call your protected API endpoint.
    - The server will return the echoed message and user info.

References

- [Microsoft Authentication Library (MSAL) for Python][11] [12] [13]
- [Azure Identity client library for Python][14]
- [Azure AD Authentication in Python Web Apps][15] [16]

This approach uses the recommended interactive flow for user authentication. For service-to-service (daemon) scenarios, consider using the client credentials flow with a client secret or certificate.

Let me know if you need a non-interactive (service-to-service) example or have any trouble with your Azure configuration!

___ [15] [14] [11] [16] [12] [13]

⁂

1. https://github.com/modelcontextprotocol/python-sdk
2. https://www.digitalocean.com/community/tutorials/mcp-server-python
3. https://github.com/ruslanmv/Simple-MCP-Server-with-Python
4. https://learn.microsoft.com/en-us/samples/azure-samples/remote-mcp-functions-python/remote-mcp-functions-python/
5. https://modelcontextprotocol.io/specification/2025-03-26/basic/authorization
6. https://auth0.com/blog/an-introduction-to-mcp-and-authorization/
7. https://blog.cloudflare.com/nl-nl/remote-model-context-protocol-servers-mcp/
8. https://learn.microsoft.com/en-us/azure/healthcare-apis/register-application
9. https://intility.github.io/fastapi-azure-auth/usage-and-faq/calling_your_apis_from_python/
10. https://github.com/intility/fastapi-azure-auth
11. https://github.com/AzureAD/microsoft-authentication-library-for-python
12. https://learn.microsoft.com/ja-jp/entra/msal/python/
13. https://github.com/AzureAD/microsoft-authentication-library-for-python/blob/dev/sample/interactive_sample.py

14. https://learn.microsoft.com/en-us/python/api/overview/azure/identity-readme?view=azure-python
15. https://learn.microsoft.com/en-us/azure/active-directory-b2c/configure-authentication-sample-python-web-app
16. https://docs.azure.cn/en-us/active-directory-b2c/enable-authentication-python-web-app