MCP OAuth Demo – Architecture Diagrams

This document provides visual, high-level and detailed views of the current **MCP OAuth demo** you're running locally.

It covers:

- Components and boundaries
- End-to-end Authorization Code + PKCE flow
- Normal tool call path after auth
- Where a **Gateway/TES** would slot in (future step)

---

1) High-Level Component View

```
flowchart LR
  subgraph AgentBox["AI Agent (MCP Client)"]
    AG["agent.py
- launches browser for auth
- stores token"]
  end

  subgraph OAuthBox["Authorization Server (AS)"]
    AS["auth/src/index.js
- .well-known metadata
- /authorize (PKCE)
- /token"]
  end

  subgraph RSBox["MCP Resource Server (RS)"]
    RS["server/src/index.js
- /mcp/echo (protected)"]
  end

  AG -- 1) Discover RS --> RS
  RS -- 2) WWW-Authenticate + metadata URL --> AG
  AG -- 3) Fetch AS metadata --> AS
  AG -- 4) /authorize (browser) --> AS
  AS -- 5) redirect back (code) --> AG
  AG -- 6) /token (code+PKCE) --> AS
  AS -- 7) access_token (JWT) --> AG
  AG -- 8) Bearer call --> RS
  RS -- 9) data --> AG

  classDef svc fill:#eef,stroke:#557;
  classDef agent fill:#efe,stroke:#575;
  class AG agent
  class AS,RS svc
```

Legend

- **AG** = MCP client (your AI agent)
- **AS** = Authorization Server (auth service)
- **RS** = Resource Server (MCP server / tools)

---

2) Authorization Code + PKCE Sequence

```
sequenceDiagram
  autonumber
  participant User as User (Browser)
  participant AG as Agent (agent.py)
  participant RS as MCP Resource Server (9091)
  participant AS as Authorization Server (9092)

  Note over AG: Start without token
  AG->>RS: GET /echo (no auth)
  RS-->>AG: 401 + WWW-Authenticate\n(resource metadata URL)

  AG->>RS: GET /.well-known/oauth-protected-resource
  RS-->>AG: Resource metadata (authorization_servers[])

  AG->>AS: GET /.well-known/oauth-authorization-server
  AS-->>AG: Authorization metadata (authorize, token, PKCE=S256)

  Note over AG: Create PKCE code_verifier + code_challenge
  AG->>User: Open /authorize?
client_id&redirect_uri&scope&state&code_challenge
  User->>AS: Consent (auto-approved in demo)
  AS-->>User: 302 redirect back with ?code&state
  User->>AG: Callback with authorization code

  AG->>AS: POST /token (code + code_verifier + client_id + redirect_uri)
  AS-->>AG: {access_token, token_type, expires_in}

  AG->>RS: GET /echo Authorization: Bearer
  RS-->>AG: { ok: true, ... }
```

---

3) Normal Tool Call Path (After Auth)

```
flowchart LR
  AG[Agent\nhas access_token] -->|Bearer| RS[(MCP Server)]
  RS -->|200 JSON| AG
```

Typical RS behavior:

1. Verify JWT signature (HS256 in demo)

2. Check claims (`aud`, `scope`, `exp`)
3. Execute tool (e.g., `/echo`) and return result

---

4) Where the Gateway (TES) Will Sit (Future Step)

```
flowchart LR
  subgraph Client["AI Agent (MCP Client)"]
    AG[agent.py]
  end

  subgraph Gateway[TES / Gateway\nPolicy, Consent, Auditing]
    GW[HTTP reverse proxy + policy engine]
  end

  subgraph MCP[MCP Resource Server / Tools]
    RS[/RS: /echo /secure .../]
  end

  subgraph Auth[Authorization Server]
    AS[/AS: /authorize /token/]
  end

  AG -- Bearer + requests --> GW
  GW -- enforced, signed assert --> RS
  RS -- responses --> GW -- filtered/logged --> AG

  GW -- OIDC/OAuth --> AS
```

When added, the Gateway/TES will:

- Intercept the agent's calls
- Enforce **fine-grained policy** per request
- **Hide** raw OAuth tokens from the agent/tools (credential firewall)
- Optionally swap Bearer tokens for **internal assertion JWTs**
- Centralize audit logs

---

5) Ports & Processes (Local)

```
flowchart TB
  A[Agent\npython agent.py] -->|HTTP| B[AS\n:9092]
  A -->|HTTP| C[RS\n:9091]

  subgraph Terminal 1
    C
  end
  subgraph Terminal 2
    B
  end
  subgraph Terminal 3
    A
  end
```

- **9091** – MCP Resource Server
- **9092** – Authorization Server
- Agent runs in its own terminal, launches the browser for the `/authorize` step.

---

6) Minimal Data Model (JWT)

```
classDiagram
  class AccessToken {
    string iss
    string sub
    string aud
    number iat
    number exp
    string scope
  }

  class AgentState {
    string sessionId
    string~scopes~
    string userId
  }

  AccessToken --> AgentState : used by
```

In the demo:

- `alg: HS256`, `aud: "mcp-demo"`, `scope: "echo:read"`
- RS validates these claims before serving tools

---

7) Quick Reference

- **AS** (9092): `/.well-known/oauth-authorization-server`, `/authorize`, `/token`
- **RS** (9091): `/.well-known/oauth-protected-resource`, `/echo`, `/secure`
- **Agent**: runs Authorization Code + PKCE, stores token, calls RS with Bearer
- **Next step**: Insert Gateway/TES between Agent and RS/AS for enterprise authN/Z