AGNTCon + MCPCon Japan 2026

HITACHI

Hands-on Workshop — 95 min

Building MCP Authorization with Keycloak and agentgateway

Presented by

Yoshiyuki Tabata, Hitachi, Ltd.

Michito Okai, Hitachi, Ltd.

Date

September 10, 2026

Agenda

1. **Introduction — 10 min**
   Why authorization matters, what the MCP Authorization specification defines, and the environment you will build
2. **Hands-on — 80 min**
   Set up, call a tool unprotected, protect it, get a token, inspect it, then step up to a stronger scope
3. **Wrap-up — 5 min**
   Key takeaways and Q&A

Reference material
AAIF blog: Exposing OpenAPI Operations as Authorized MCP Tools with agentgateway and Keycloak — https://aaif.io/blog/exposing-openapi-operations-as-authorized-mcp-tools-with-agentgateway-and-keycloak

HITACHI

Welcome – your instructors

Yoshiyuki Tabata

Chief OSS Consultant @ Hitachi, Ltd.

Tech Lead, CNCF TAG Security & Compliance

CNCF & AAIF Ambassador

LinkedIn: @ytabata / X: @yo_tabata / GitHub: @y-tabata

Michito Okai

Software Engineer @ Hitachi, Ltd.

Specialist in authentication and authorization

Keycloak & Conformance Tests for MCP contributor

LinkedIn: @michito-okai / GitHub: @Michito-Okai

HITACHI

What You Will Take Away Today

Today you will build MCP authorization with your own hands, and by doing so understand what the authorization part of the MCP specification actually says.

What you will do

Call an MCP tool, watch it be rejected, and make it work with an access token.

What it means

Why the specification asks for each step, and what a token actually represents.

How to implement it

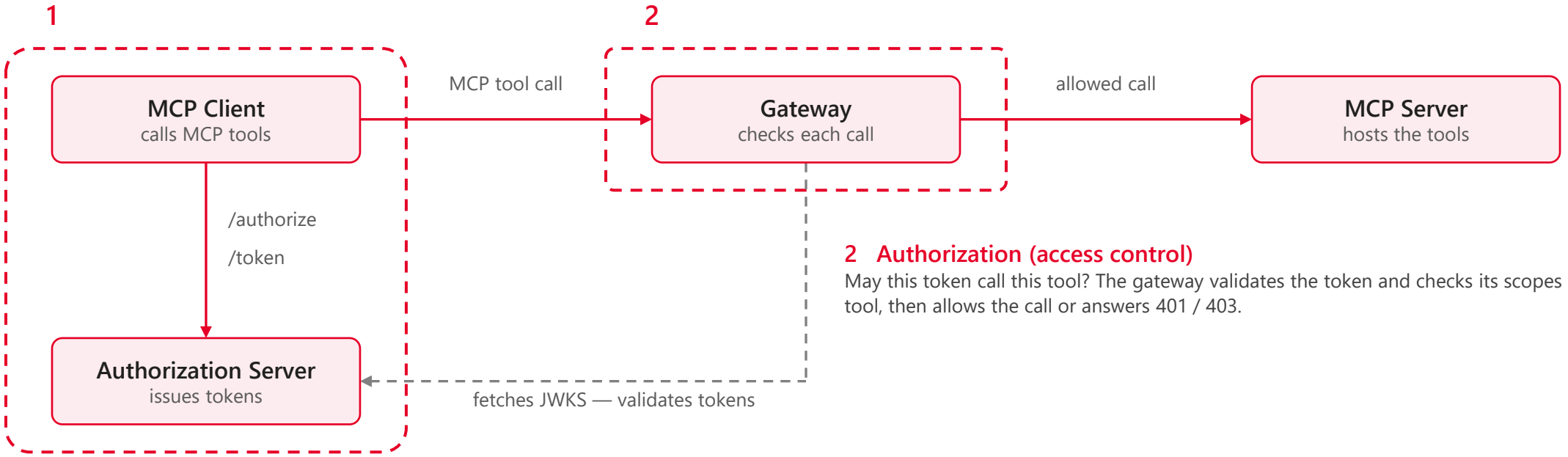
Which endpoints, errors and scopes you configure — and how to verify them.

No OAuth experience needed — every term is introduced at the moment you meet it in the hands-on.

HITACHI

Two Meanings of Authorization

In a typical MCP architecture, the word authorization means two different things, and they happen in two different places. Today's hands-on is about the first one, delegation.



2 Authorization (access control)

May this token call this tool? The gateway validates the token and checks its scopes for each tool, then allows the call or answers 401 / 403.

1 Authorization (delegation)

Does the user grant this client permission? The client discovers the authorization server, the user consents, and an access token carrying scopes is issued. This is what MCP Authorization standardizes.

HITACHI

What Happens When Nobody Delegated Anything

Authorization here means delegation: the owner of the data lets a client act on their behalf, and only that far. Take that step away and the client simply helps itself.

Resource owner
the data and operations are theirs

no grant, no consent

MCP Client
calls tools on the owner's behalf

the call runs anyway

MCP Server
runs whatever it is asked to run

Nothing was delegated — yet every tool the server exposes is reachable by whoever asks

data nobody agreed to share is read out

records are changed or removed

and the company running it answers for it

None of this is a bug — the server did exactly what it was asked. Without authorization, nothing limits what can be asked.

Illustrations on this slide were created with generative AI.

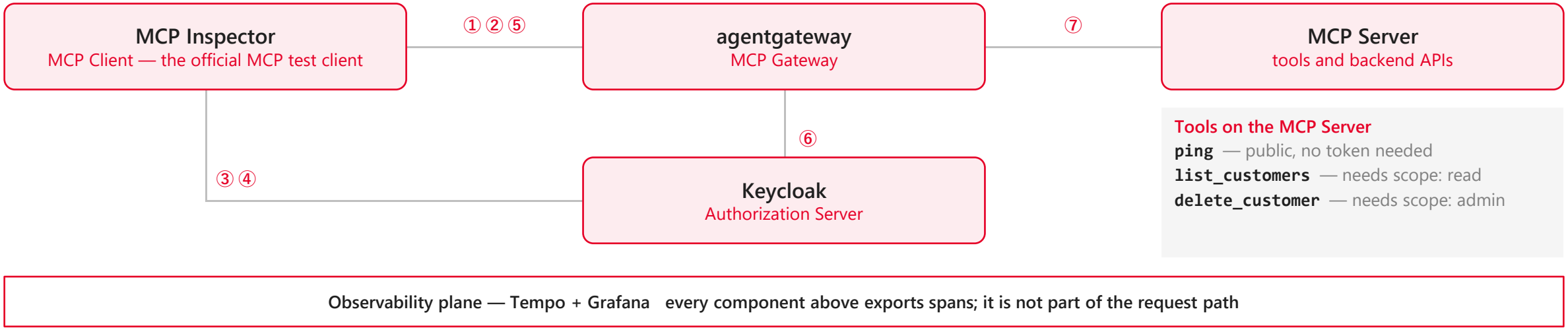
HITACHI

That Is Why MCP Defines an Authorization Section

MCP took that risk seriously and standardized authorization. Every question a client has was already answered by OAuth specifications that have earned years of trust and proven operation across the API ecosystem.

What the client must figure out	Which existing specification answers it	What it does
Where do I authorize?	Protected Resource Metadata (RFC 9728)	points at the authorization server
Which endpoints do I call?	Authorization Server Metadata (RFC 8414)	lists the endpoints
How do I get a token safely?	OAuth 2.1 + PKCE (RFC 7636)	authorization code flow: the client redirects the user to the authorization server, where the user authenticates and consents; the client then exchanges the returned one-time code for a token. PKCE ensures only the party that started the flow can make that exchange.
How do I send it, and read errors?	Bearer Token Usage (RFC 6750)	how to send the token and read 401 / 403

The Workshop Environment



① MCP Inspector calls a tool through agentgateway.

② The gateway answers 401 with WWW-Authenticate and the protected resource metadata (PRM) URL.

③ The client reads the PRM, then fetches the authorization server metadata from Keycloak.

④ Using those endpoints the client runs /authorize (login and consent) and /token (code + PKCE).

⑤ The client retries the same tool call, now carrying the access token.

⑥ agentgateway validates the token issued by Keycloak and checks the required scope.

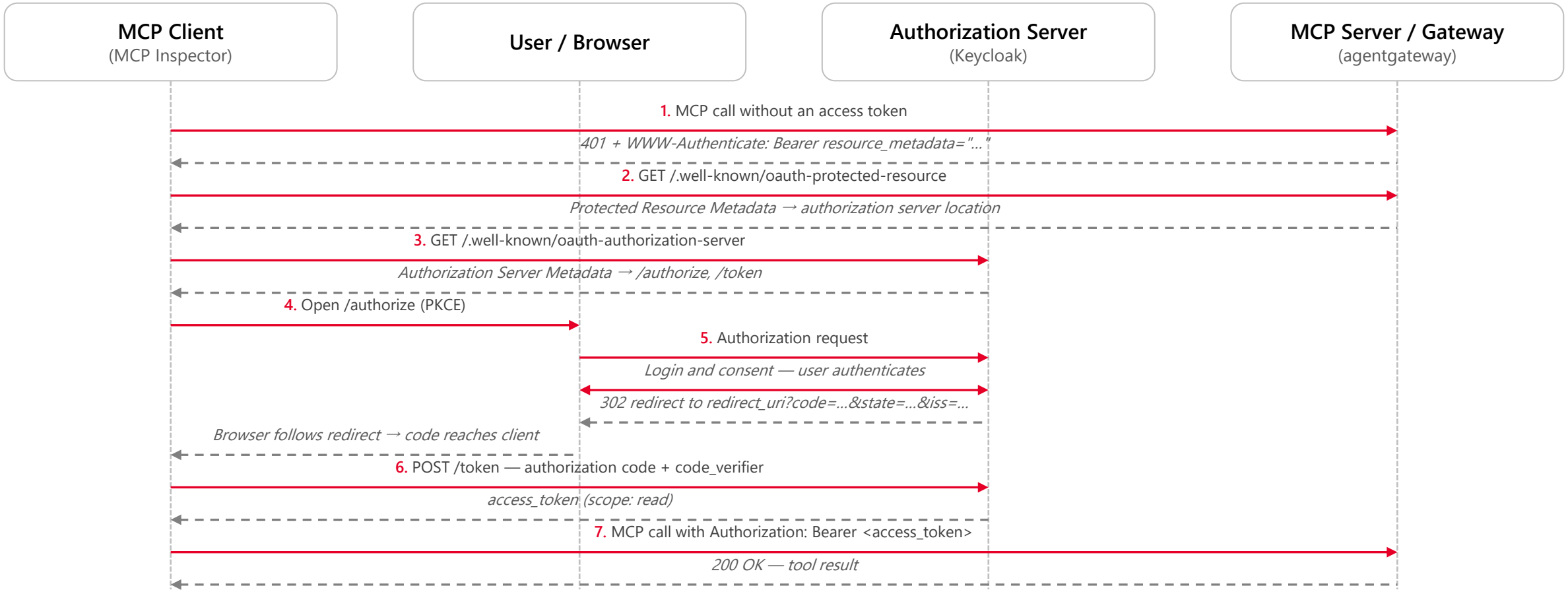
⑦ The authorized call finally reaches the MCP Server and its backend APIs.

Runtime All five components run as Docker containers — you start and stop them with docker compose.

HITACHI

The Authorization Code Flow in MCP

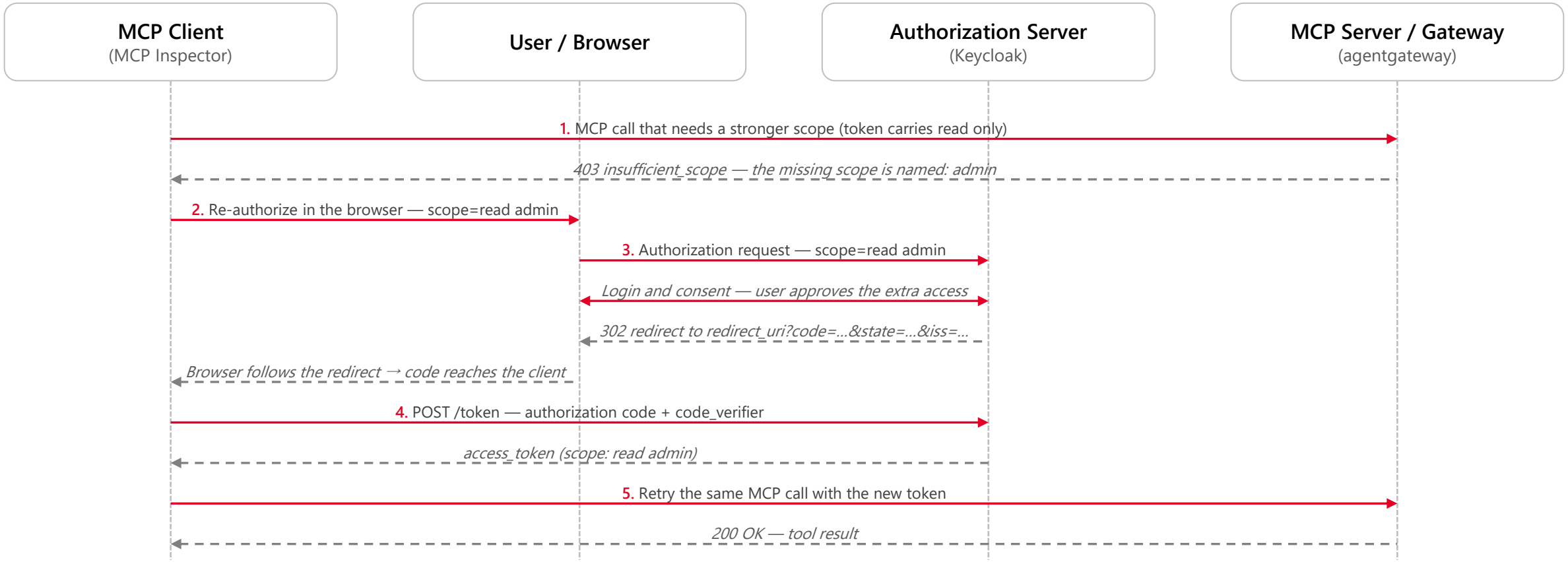
In the hands-on you verify this flow in two steps. Step one, on this slide: the whole round trip — the rejected call, discovery, user authorization, the token, and the tool call that finally succeeds.



HITACHI

Step-up Authorization

Step two, continuing from the previous slide: the client already holds a token, but a stronger call needs more. In our workshop environment, the gateway identifies the missing scope and the client repeats the authorization flow to obtain a token with expanded permissions.

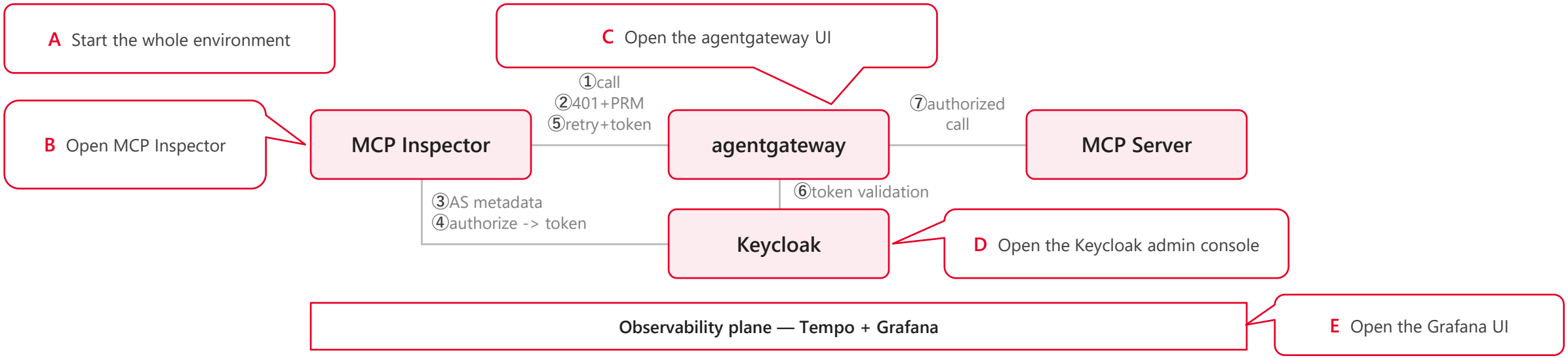


Hands-on

1. Step 0 Set up the environment: MCP Inspector, agentgateway, MCP Server, Keycloak — 10 min
2. Step 1 Call an MCP tool while every tool is still public — 10 min
3. Step 2 Protect the tool, read the 401, and get an access token — 15 min
4. Step 3 Retry the call with the token — 5 min
5. Step 4 Look inside the access token — 5 min
6. Step 5 Hit an insufficient scope error (403) — 10 min
7. Step 6 Grant the missing permission in Keycloak — 10 min
8. Step 7 Perform step-up authorization — 10 min
9. Step 8 Verify the new token and retry — 5 min

HITACHI

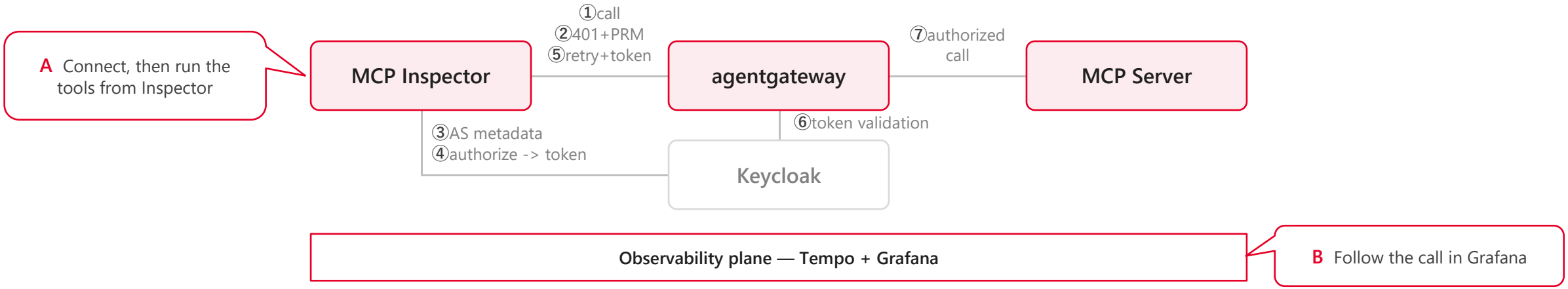
Set Up the Environment



Steps, commands and URLs

A `git clone https://github.com/Hitachi/oss-assets.git` — *get the workshop sources*
A `cd oss-assets/sessions/2026/agntcon-jp/workshop` — *move into the workshop directory*
A `./setup-certs.sh` — *issue the CA and TLS certificates for keycloak.localtest.me*
A `docker compose up -d` — *start all five containers*
B `http://localhost:6274` — *MCP Inspector, the client you drive*
C `http://localhost:8088/ui` — *agentgateway UI: listeners, routes, policies, live logs*
D `https://keycloak.localtest.me:8443/admin` — *Keycloak admin console (sign in with admin / admin): realm, client, scopes*
E `http://localhost:3001` — *Grafana (sign in with admin / admin): traces of your calls*

Call an MCP Tool Without Authorization



Steps, commands and URLs

A `Connect` — *connect MCP Inspector to agentgateway first*

A `Tools → ping → [Execute Tool] → Success` — *a public tool answers immediately*

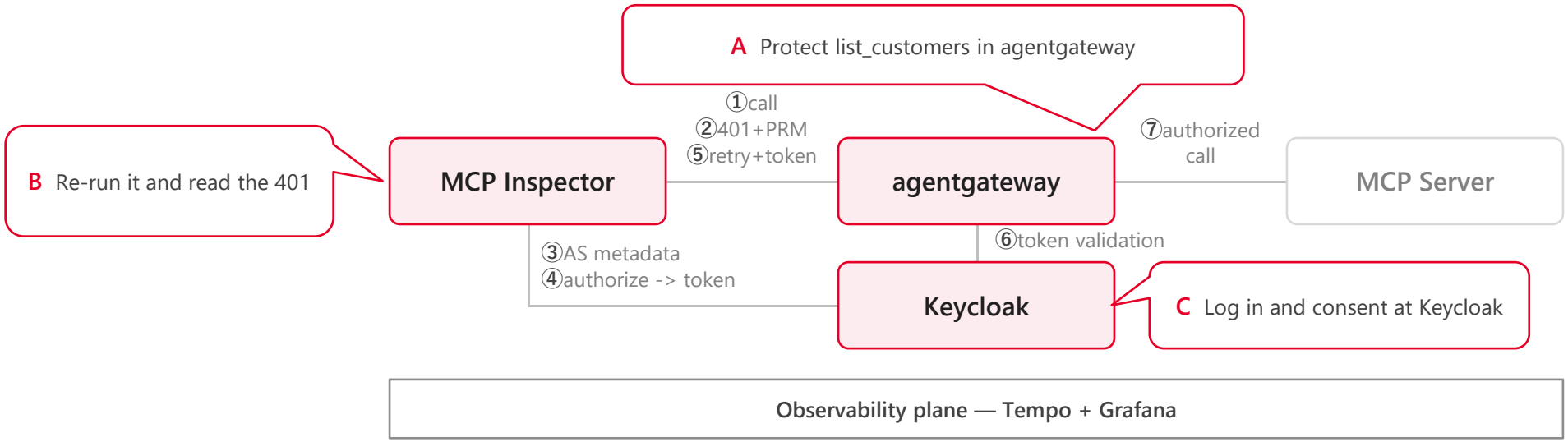
A `Tools → list_customers → [Execute Tool] → Success` — *it answers too — nothing is protected yet*

A `Tools → delete_customer → [Execute Tool] → Success` — *it runs as well — nothing is protected yet*

B `http://localhost:3001 → Explore → Service Name: mcp-inspector → [Run query] → open the trace` — *one span chain: Inspector -> agentgateway -> MCP Server, no OAuth*

HITACHI

Protect the Tool, Then Get a Token



Steps, commands and URLs

A `cp agentgateway/config/config-auth.yaml agentgateway/config/config.yaml` — *swap in the config that protects list_customers*

B `Tools → list_customers → [Execute Tool] → HTTP 401` — *no token yet, so the gateway rejects the call*

B `WWW-Authenticate: Bearer resource_metadata="..."` — *the 401 points at the protected resource metadata*

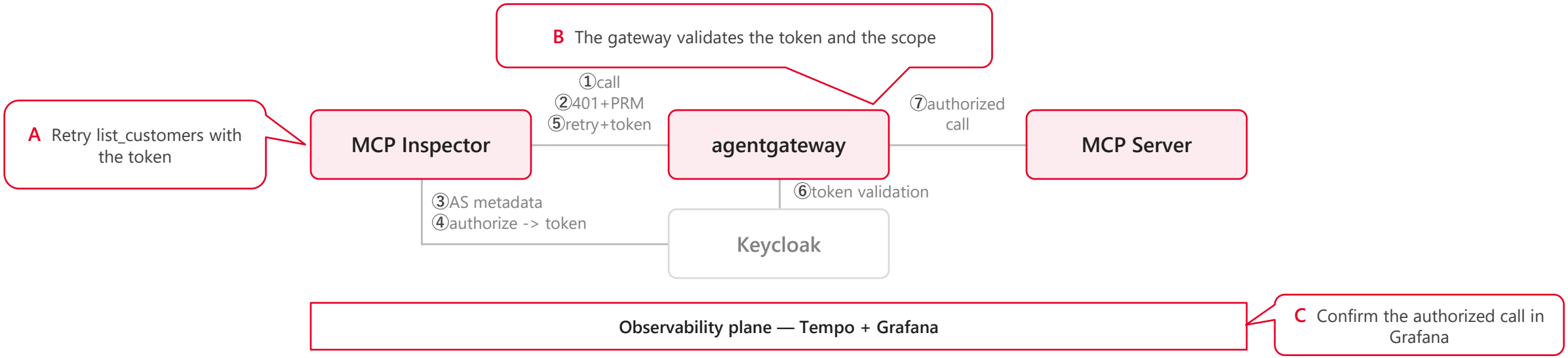
C `PRM → AS metadata → /authorize` — *Inspector discovers Keycloak and opens its login page*

C `login as workshop-user (alice / alice) → consent read` — *approve the read scope in the browser*

C `/token code + PKCE → access token` — *Inspector stores the token for the next call*

HITACHI

Retry the Call with the Token



Steps, commands and URLs

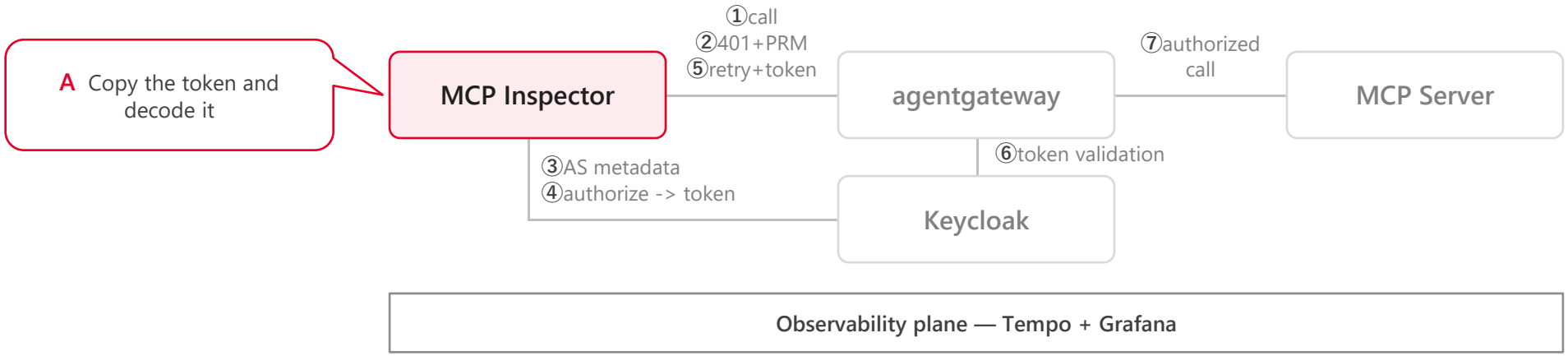
A `Tools → list_customers → [Execute Tool] → Success` — *retry the same call, now with the token*

A `Authorization: Bearer eyJ...` — *Inspector attaches the token it just obtained*

B `JWKS signature + required scope read` — *agentgateway checks the token, then lets the call through*

C `http://localhost:3001 → Explore → Service Name: mcp-inspector → [Run query] → open the trace` — *the retried call now reaches the MCP Server*

Look Inside the Access Token



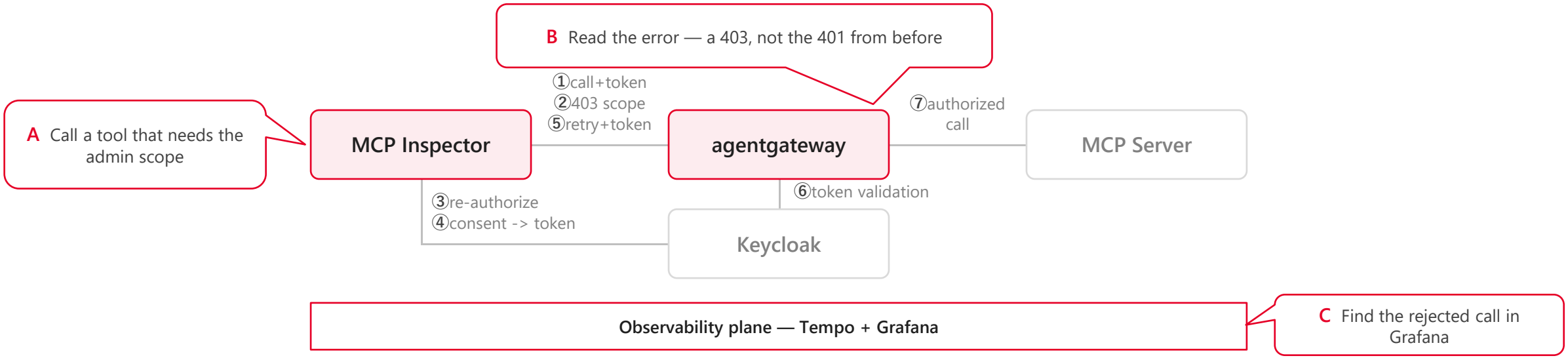
Steps, commands and URLs

A `Inspector → Connection Info → [Decode JWT]` — *decode the access token inside Inspector*

A `Payload  "scope": "read"` — *the permission the gateway checks*

HITACHI

Hit an Insufficient Scope Error



Steps, commands and URLs

A `Browser DevTools → Network` — *open it before the call, or the 403 is not captured*

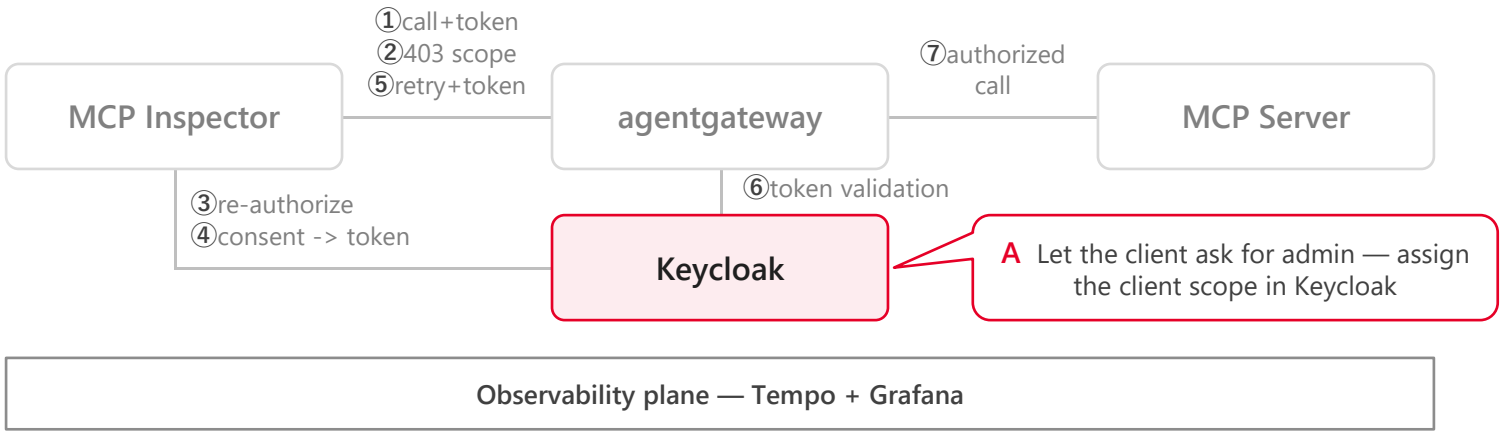
A `Tools → delete_customer → [Execute Tool] → HTTP 403 Forbidden` — *the old token only carries read*

B `{ "error": "insufficient_scope" }` — *the gateway names the missing scope*

C `http://localhost:3001 → Explore → Service Name: mcp-inspector → [Run query] → open the trace` — *the span stops at the gateway, tagged insufficient_scope*

HITACHI

Fix It: Grant the Missing Permission



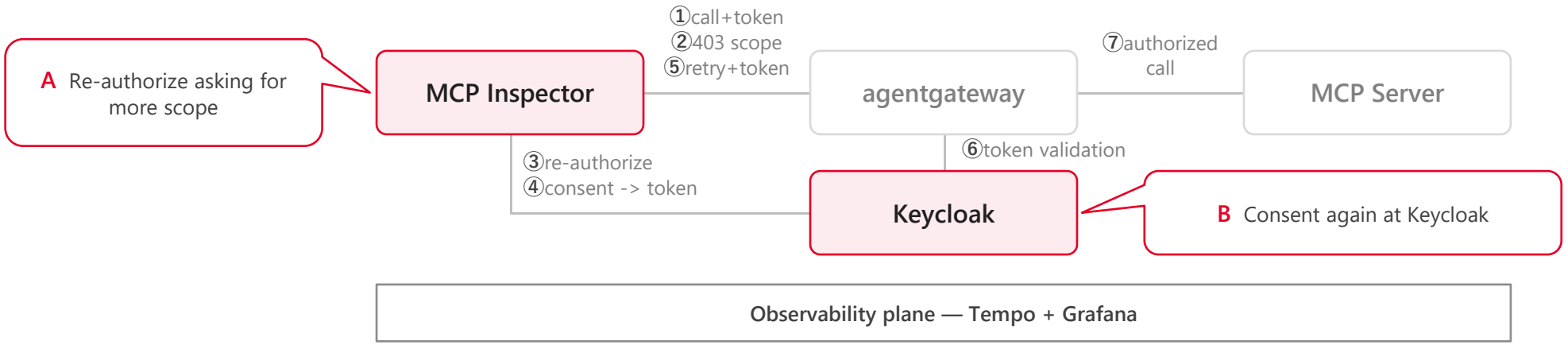
Steps, commands and URLs

A `https://keycloak.localtest.me:8443/admin` — *the 403 named the scope you are missing: admin*

A `Clients -> mcp-inspector -> Client scopes` — *the client must be allowed to ask for it*

A `Add client scope -> admin -> Optional` — *optional = requested only when needed*

Perform Step-Up Authorization



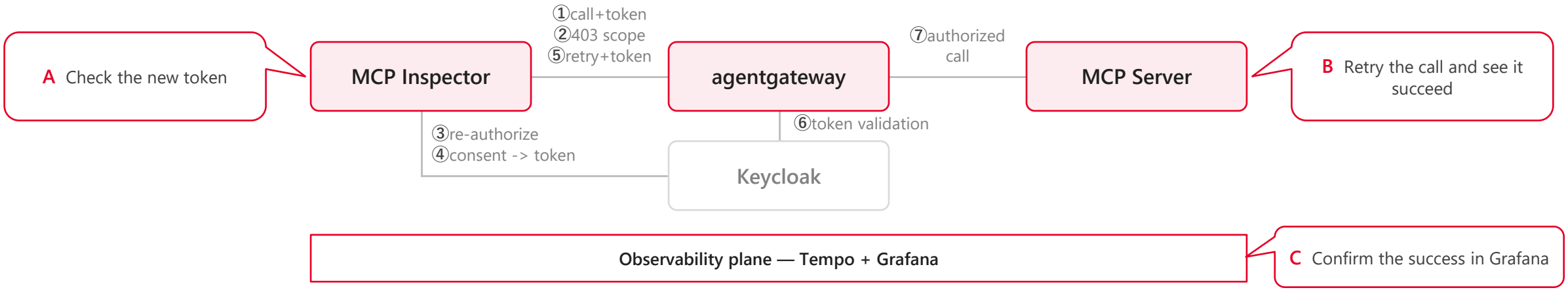
Steps, commands and URLs

A `Inspector → [Re-authenticate]` — *press the re-authenticate button first, then run the flow*

A `Scope requested  read admin` — *incremental authorization, not a new tool*

B `Consent at Keycloak -> new token` — *the user approves the wider scope*

Verify the New Token and Retry



Steps, commands and URLs

A `Inspector → Connection Info → [Decode JWT]` — *decode the new token inside Inspector*

A `Payload  "scope": "read admin"` — *the scope grew, the tool did not change*

B `Tools → delete_customer → [Execute Tool]  →  Success` — *retry the same call*

C `http://localhost:3001 → Explore → Service Name: mcp-inspector → [Run query] → open the trace` — *compare this span with the 403 one from Step 5*

Wrap-up

1. Key takeaways
2. Q&A

HITACHI

Key Takeaways

- Authorization matters: without it, every tool an MCP server exposes is reachable by anyone, in anyone's name
- The MCP Authorization specification defines how a client obtains a user-delegated token for an MCP server, on top of OAuth specifications the API ecosystem has trusted and operated for years
- MCP Authorization can be implemented with standards-based components: Keycloak as the MCP authorization server and agentgateway as the MCP gateway, combined into a setup that conforms to the specification

Learn more

AAIF blog: Exposing OpenAPI Operations as Authorized MCP Tools with agentgateway and Keycloak — https://aaif.io/blog/exposing-openapi-operations-as-authorized-mcp-tools-with-agentgateway-and-keycloak

HITACHI

Trademarks

- OpenID is a trademark or registered trademark of OpenID Foundation in the United States and other countries.
- Keycloak is a trademark or registered trademark of Red Hat, Inc. in the United States and other countries.
- GitHub is a trademark or registered trademark of GitHub, Inc. in the United States and other countries.
- X is a trademark or registered trademark of X Corp. in the United States and other countries.
- Other brand names and product names used in this material are trademarks, registered trademarks, or trade names of their respective holders.

AGNTCon + MCPCon Japan 2026

HITACHI

Thank you

Presented by

Yoshiyuki Tabata, Hitachi, Ltd.

Michito Okai, Hitachi, Ltd.

Date

September 10, 2026