Passwordless authentication on web

HOW TO SECURE WEB APPLICATIONS AND WEB SERVICES WITHOUT USING PASSWORD

Topics

- Passwordless authentication between:
  1. user & frontend
  2. frontend & backend
  3. backend service & backend service
- Demo: service to service authentication without password
- Focusing mainly on Azure technologies/options

User to Frontend - Traditional password-protected systems

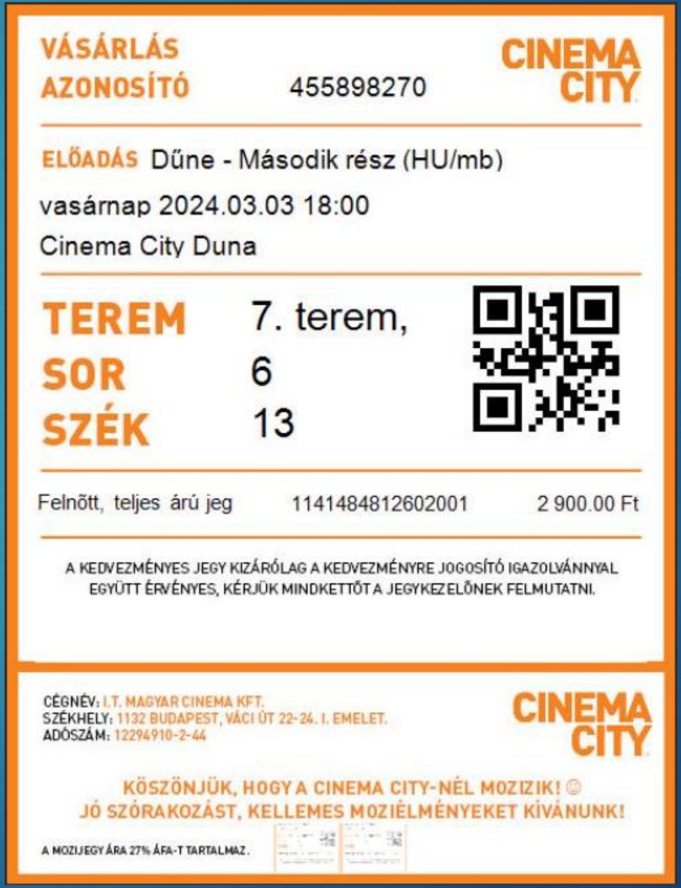
- Today the primary method of authentication
- Disadvantages:
  - Needs to be complex -> hard to remember
  - Need to change frequently
  - Reusing the same password at multiple places
  - Need for a password management system
  - Threat from hackers (phishing, brute force)

User to Frontend - Passwordless alternatives in Azure (EntraID and Entra External ID)

- **Entra External ID** (replacement of AD B2C): A CUSTOMER cloud-based identity access management service from Microsoft (sign up, sign in, and manage customer, external workforce profiles).
- **EntraID** (replacement of AD B2B): An EMPLOYEE cloud-based identity and access management service from Microsoft. It helps your employees sign in and access resources (eg. PC, Microsoft Office 365, Azure…etc).
- **Options for authentication:**
  - Biometric data (face recognition, fingerprint)
  - OTP: authenticator app, sms on phone, email
  - Hardware keys (FIDO2 standard)

Frontend to backend – access token (JWT)

- Analogy – cinema ticket:
  - Gives access to one move
  - VIP/3D/normal
  - Gives access to a limited time
  - Preserves one seat
  - Can use bufe, toalet
  - Ticket usher validates it
  - Cannot be (easily) forged



VÁSÁRLÁS AZONOSÍTÓ 455898270

CINEMA CITY

ELŐADÁS Dűne - Második rész (HU/mb)

vasárnap 2024.03.03 18:00

Cinema City Duna

TEREM 7. terem,

SOR 6

SZÉK 13

Felnőtt, teljes árú jeg 1141484812602001 2 900.00 Ft

A KEDVEZMÉNYES JEGY KIZÁRÓLAG A KEDVEZMÉNYRE JOGOSÍTÓ IGAZOLVÁNNYAL EGYÜTT ÉRVÉNYES, KÉRJÜK MINDKETTŐT A JEGYKEZELŐNEK FELMUTATNI.

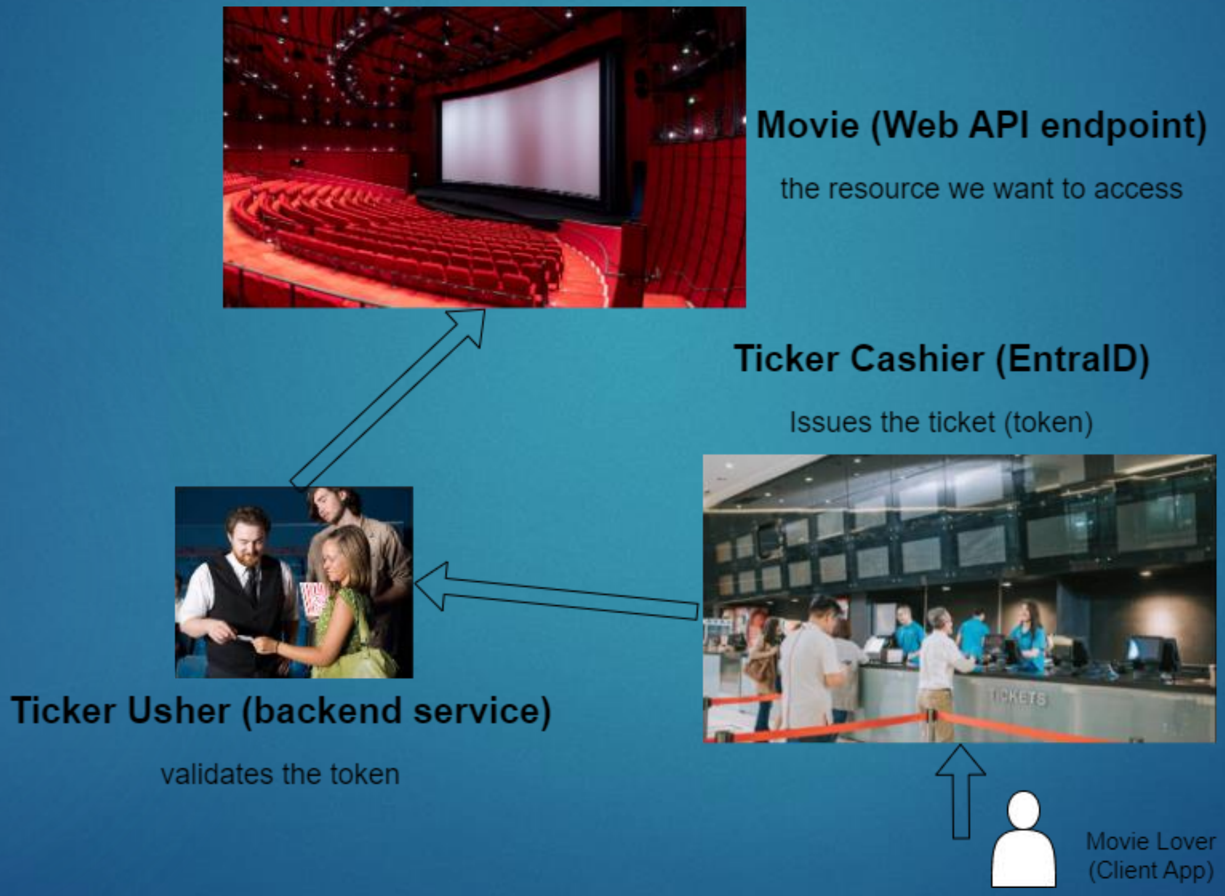
CÉGNÉV: I.T. MAGYAR CINEMA KFT.
SZÉKHELY: 1132 BUDAPEST, VÁCI ÚT 22-24. I. EMELET.
ADÓSZÁM: 12294910-2-44

CINEMA CITY

KÖSZÖNJÜK, HOGY A CINEMA CITY-NÉL MOZIZIK! ☺
JÓ SZÓRAKOZÁST, KELLEMES MOZIÉLMÉNYEKET KÍVÁNUNK!

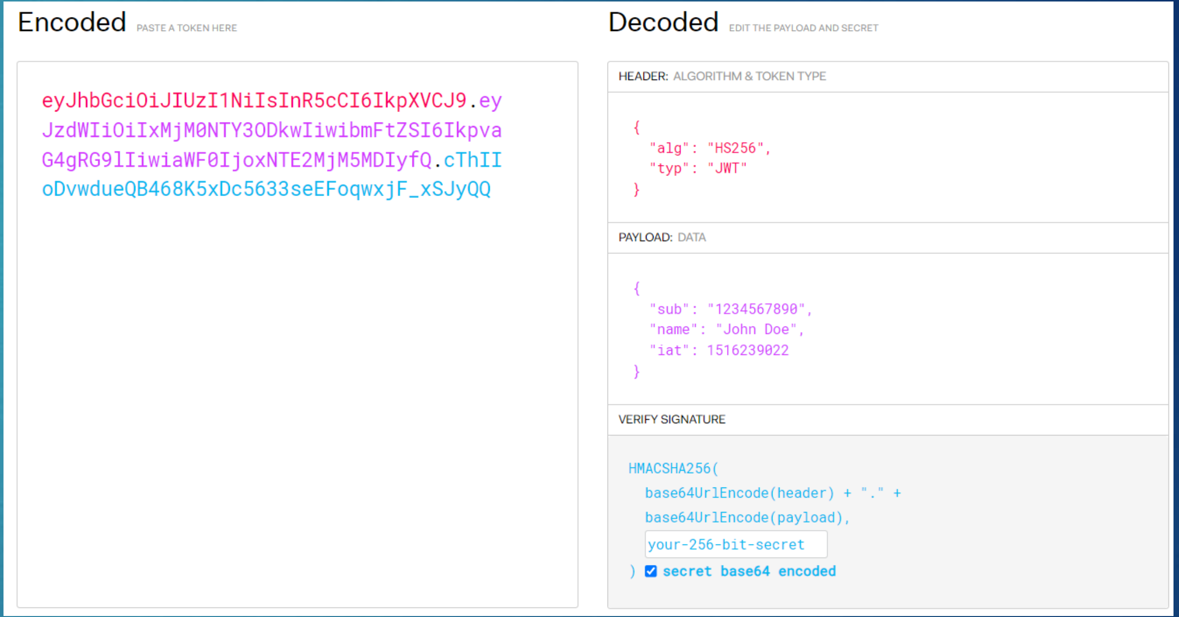
A MOZIJEGY ÁRA 27% ÁFA-T TARTALMAZ.

Frontend to backend - Cinema / Web analogy



Frontend to backend – JWT tokens

- Access token:
  - A base64 encoded string
  - Parts: head, payload, signature
  - Specifies: who you are, to what you have access to, how long the token is valid, when it was issued…etc (standardized fields)
  - Digitally signed: hash+encrypted (private key at IAM server, public key can be used to decrypt)
  - Issuing/modification only with the private key -> only IAM server has it


Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThII
oDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

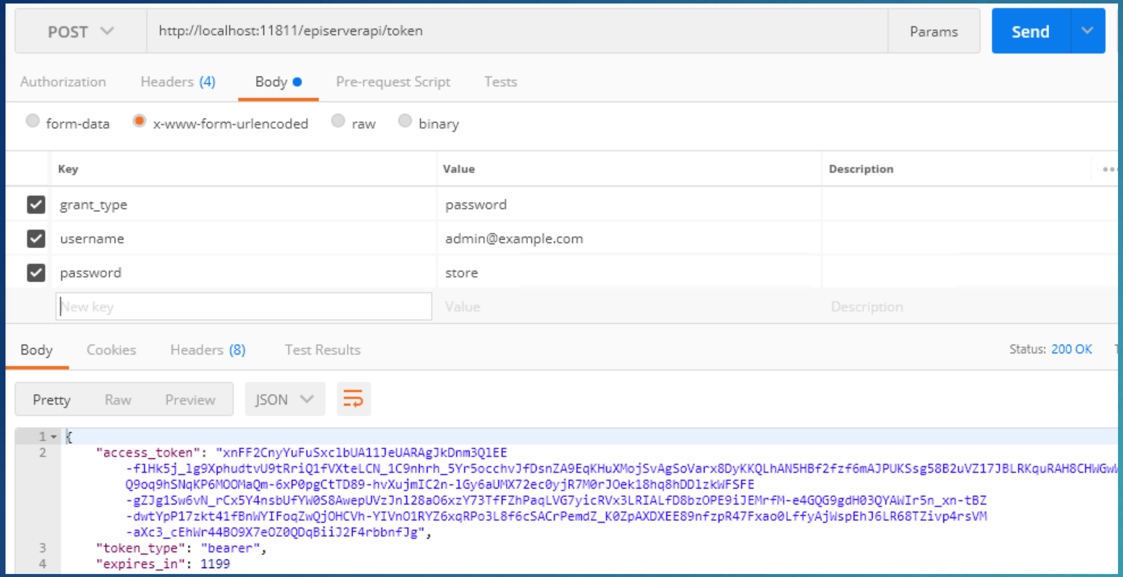
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ✓ secret base64 encoded
```

Frontend to backend – JWT tokens



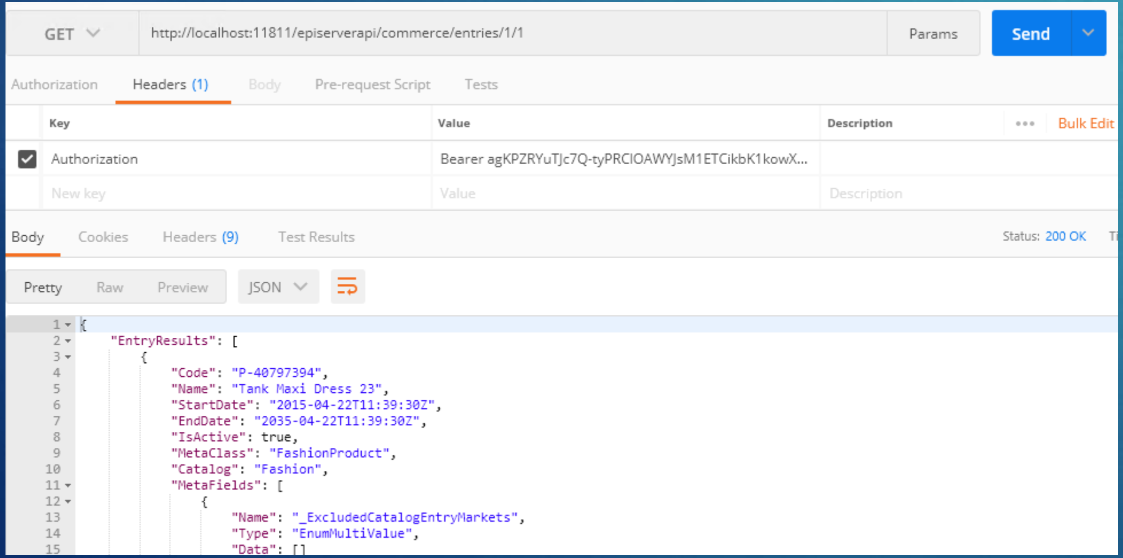
POST http://localhost:11811/episerverapi/token

Body: x-www-form-urlencoded

Key	Value	Description
✓ grant_type	password	
✓ username	admin@example.com	
✓ password	store	

Status: 200 OK

```
{
    "access_token": "xnFF2CnyYuFuSxclbUA11JeUARAgJkDnm3QlEE
        -flHk5j_lg9XphudtvU9tRriQ1fVXteLCN_1C9nhrh_5Yr5occhvJfDsnZA9EqKHuXMojSvAgSoVarx8DyKKQLhAN5HBf2fzf6mAJPUKSsg58B2uVZ17JBLRKquRAH8CHWGw
        Q9oq9hSNqKP6MOOMaQm-6xP0pgCtTD89-hvXujmIC2n-1Gy6aUMX72ec0yjR7M0rJOek18hq8hDDlzkWFSFE
        -gZJg1Sw6vN_rCx5Y4nsbUfYW0S8AwepUVzJnl28aO6xzY73TfFZhPaqLVG7yicRVx3LRIALfD8bzOPE9iJEMrfM-e4GQG9gdH03QYAWIr5n_xn-tBZ
        -dwtYpP17zkt41fBnWYIFoqZwQjOHCVh-YIVnO1RYZ6xqRPo3L8f6cSACrPemdZ_K0ZpAXDXEE89nfzpR47Fxao0LffyAjWspEhJ6LR68TZivp4rsVM
        -aXc3_cEhWr44BO9X7eOZ0QDqBiiJ2F4rbbnfJg",
    "token_type": "bearer",
    "expires_in": 1199
```



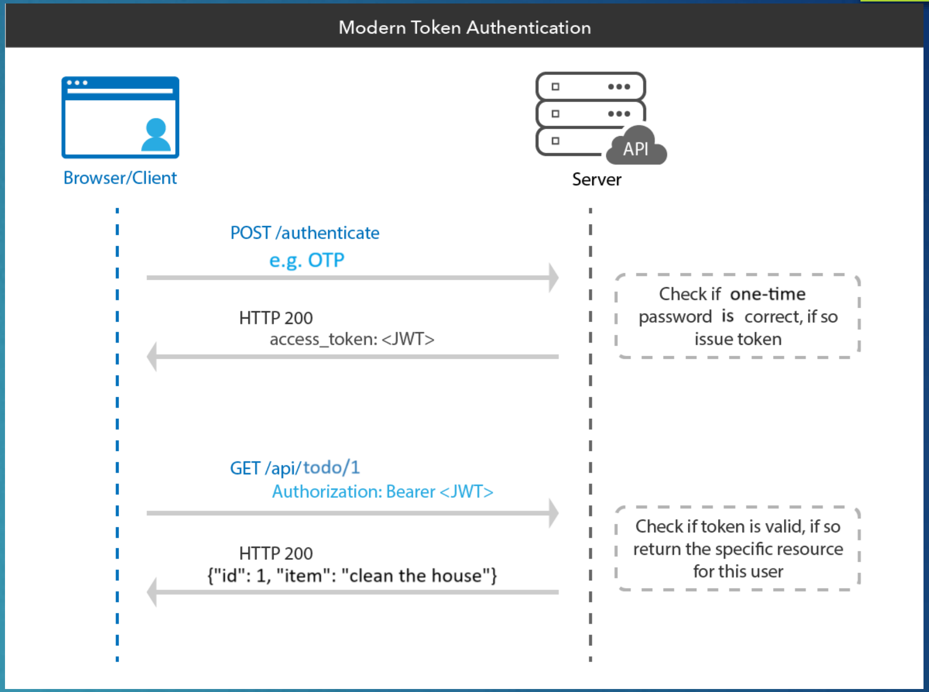
GET http://localhost:11811/episerverapi/commerce/entries/1/1

Headers (1)

Key	Value	Description
✓ Authorization	Bearer agKPZRYuTJc7Q-tyPRClOAWYJsM1ETCikbK1kowX...	

Status: 200 OK

```
{
    "EntryResults": [
        {
            "Code": "P-40797394",
            "Name": "Tank Maxi Dress 23",
            "StartDate": "2015-04-22T11:39:30Z",
            "EndDate": "2035-04-22T11:39:30Z",
            "IsActive": true,
            "MetaClass": "FashionProduct",
            "Catalog": "Fashion",
            "MetaFields": [
                {
                    "Name": "_ExcludedCatalogEntryMarkets",
                    "Type": "EnumMultiValue",
                    "Data": []
```

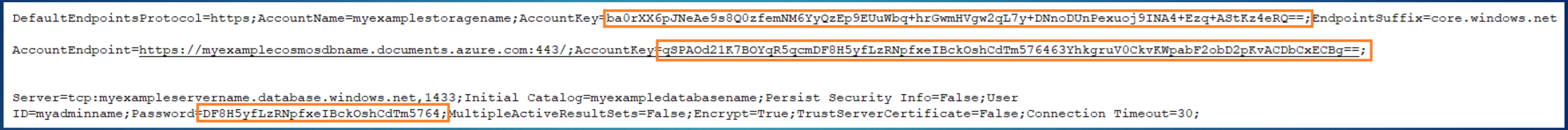


Modern Token Authentication
Browser/Client
Server
API
POST /authenticate
e.g. OTP
Check if one-time password is correct, if so issue token
HTTP 200
access_token: <JWT>
GET /api/todo/1
Authorization: Bearer <JWT>
Check if token is valid, if so return the specific resource for this user
HTTP 200
{"id": 1, "item": "clean the house"}

- When user signs into the frontend of an application using one of the passwordless methods we've discussed, EntraID generates a JWT token
- When the frontend needs to communicate with the backend (eg.: to fetch data or perform an action) it includes this token in the request. The backend then validates the token using EntraID. If it's valid, the backend processes the request, all without needing a password.
- No credential is sent in the network

Backend service to backend service – The traditional way

CosmosDb, Storage Account, SQL DB connection string:



```
DefaultEndpointsProtocol=https;AccountName=myexamplestoragename;AccountKey=ba0rXX6pJNeAe9s8Q0zfemNM6YyQzEp9EUuWbq+hrGwmHVgw2qL7y+DNnoDUnPexuoj9INA4+Ezq+AStKz4eRQ==;EndpointSuffix=core.windows.net

AccountEndpoint=https://myexamplecosmosdbname.documents.azure.com:443/;AccountKey=qSPAOd21K7BOYqR5qcmDF8H5yfLzRNpfxeIBckOshCdTm576463YhkgruV0CkvKWpabF2obD2pKvACDbCxECBg==;


Server=tcp:myexampleservername.database.windows.net,1433;Initial Catalog=myexampledatabasename;Persist Security Info=False;User
ID=myadminname;Password=DF8H5yfLzRNpfxeIBckOshCdTm5764;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```

We can store them in:

- Config files
- Environement variables
- Secret management tools (eg. Azure Key Vault)
- Docker and Kubernetes secrets

Lets follow the same pattern as on FE:

- Use only the endpoint
- Re-use access token approach
- Do not manage secrets at all!

Azure provides a unique solution!

Backend service to backend service – Managed Identity

- As Users can have profiles in EntraID, services can also have ”profiles” which are called Managed Identites
- Both kind of profiles are objects that represent either the user or service in the IAM system
- If I have a ”profile” for a service e.g.: backend API, I can give ”read permission” for a database e.g.: CosmosDb – role base access control
- When the web API wants to access the DB, it does not provide the DB key, but gets an access token from EntraID and sends it during the DB read http request -> DB verifies the token and sends back the entries

- Pros of MIs:
  - Automatic and hidden credential management (create, store, rotation)
  - No secrets in the code!
  - No secret management needed at all
  - No cost in Azure
  - More simple code -> implementation not needed (or minimal effort)
  - Lifecycle management -> if service is deleted, MI is also deleted
  - Simple authentication: service gets token and provides token easily (sometimes hidden)

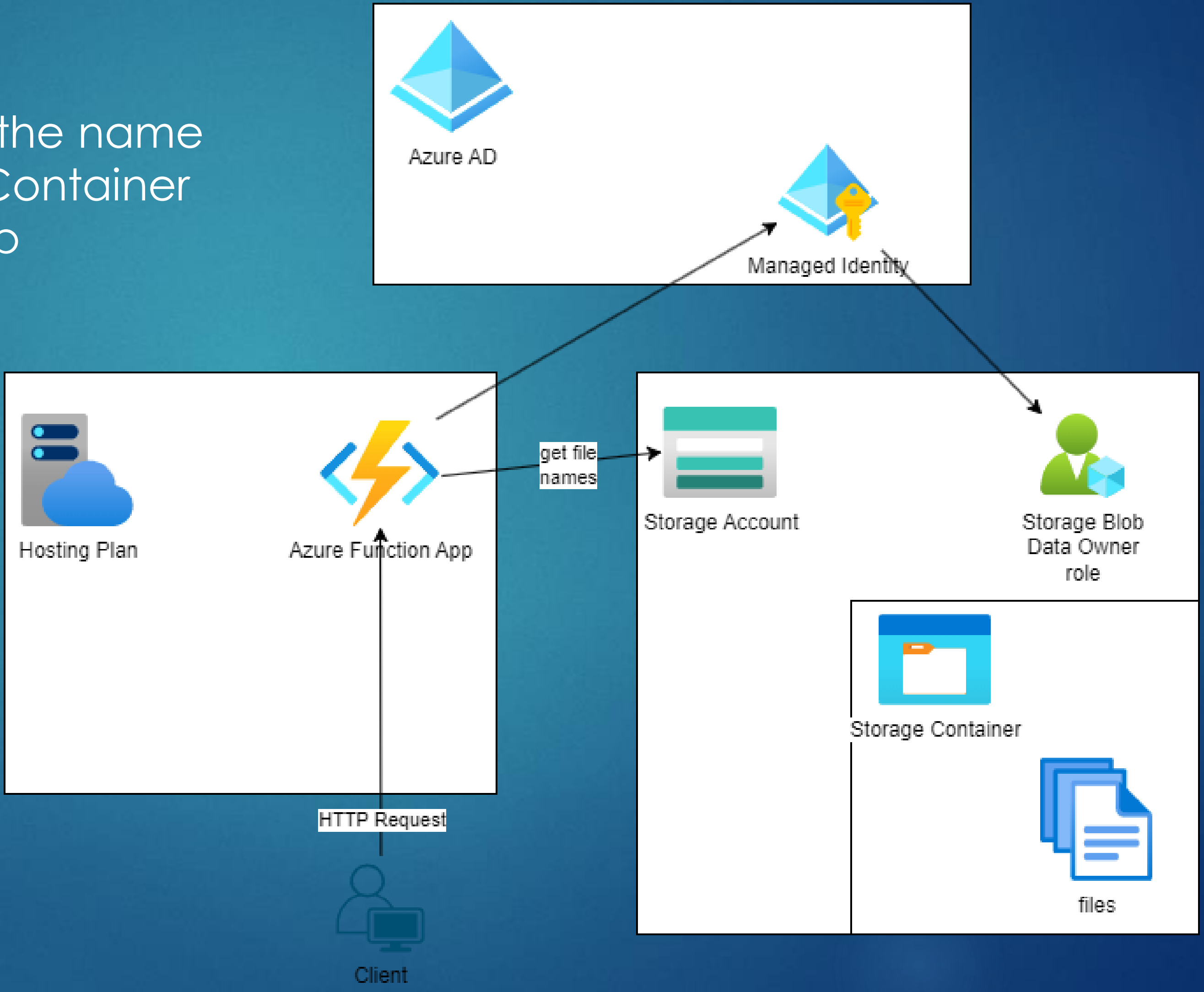
Backend service to backend service – Managed Identity + RBAC

Example:

- How to use MI to access a Storage Account from a Function App:
  - Create a MI in Azure for the Function App
  - Assign a "Storage Blob Data Contributor" role in the in the scope of the SA for the previously created MI
  - Use the endpoint of the SA without the credentials to access it
- How it works underneath:
  - The identity is used to obtain a token from EntraID.
  - This token is then presented to the second backend service.
  - The second service validates this token against EntraID to ensure it's valid.

DEMO: Service to service

- An Azure Function reads the name of blobs from a Storage Container and return the list in a http response body.



Passwordless authentication on the web

Q&A