24 OCTOBER 2025

Dev environment for the Academy Platform team

ALEKSANDRA GLESAAEN ØDEGÅRD

Who am I ?

- Worked for Omegapoint Norge for almost one year
- I have a background in theoretical physics, backend development, and optimziation
- Fairly new to Azure, so I might have gotten some things wrong
- Love learning new things, try to learn a new language every Advent of Code
- 💙 Archlinux & 💚 Neovim
- Which naturally means that I want things to work exactly how I prefer them to

What this talk is

- A demonstration of how I set up local development for the Academy Platform
- A talk on the importance of facilitating the development process
- A discussion on using real services vs emulation

What this talk isn't

- A discussion of the Academy Platform architecture
- A talk on microservices
- A talk on integration testing microservices

What is Academy Platform?

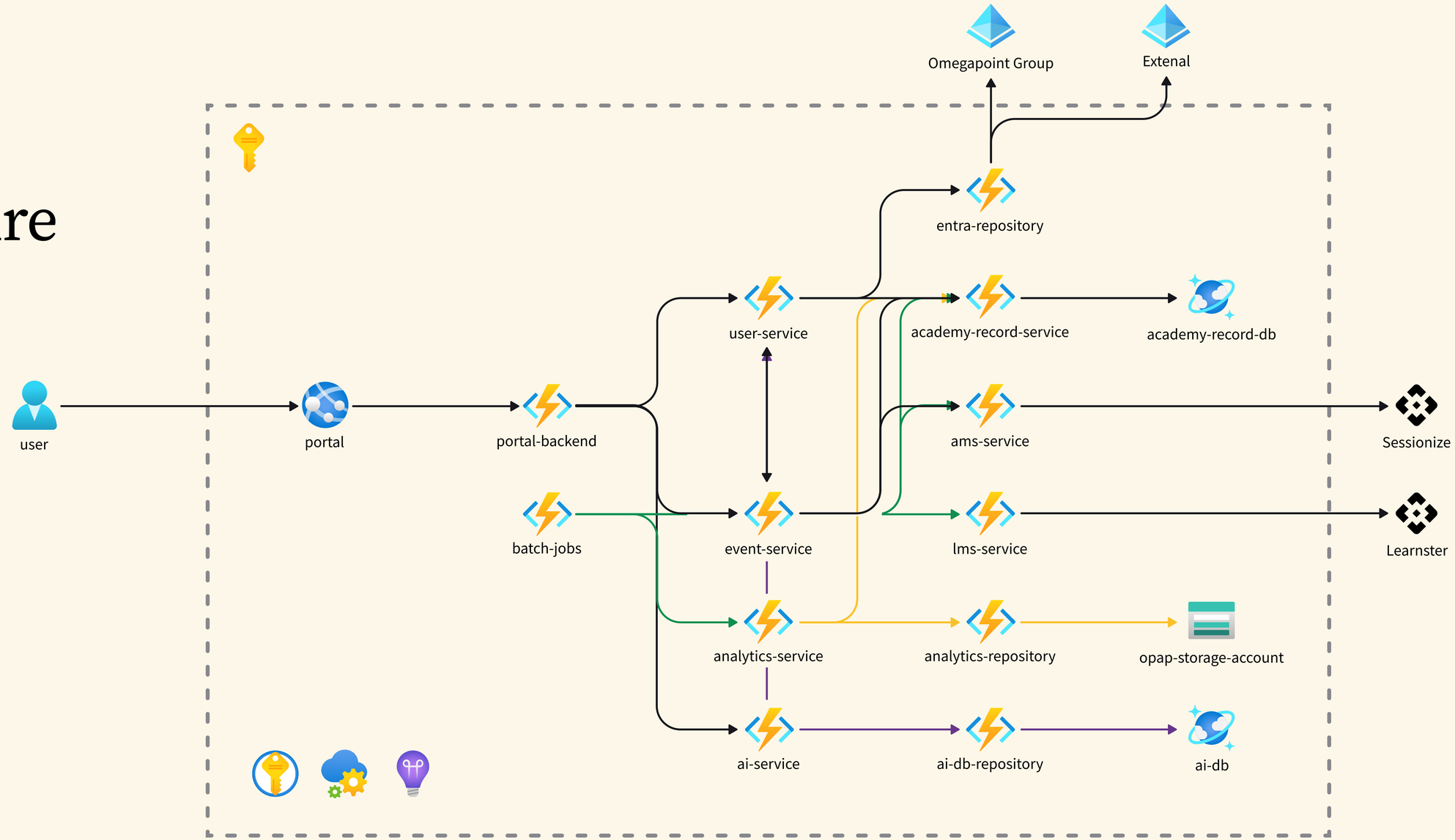
- Omegapoint’s event and learning platform
- Used to organize everything from OpKoKo to competence days to this event
- Integrates with Sessionize for talks, and Learnster for courses
- A learning opportunity for developers with no current engagement

Architecture

- Portal
- Batch jobs
- Analytics
- AI services

12 function apps

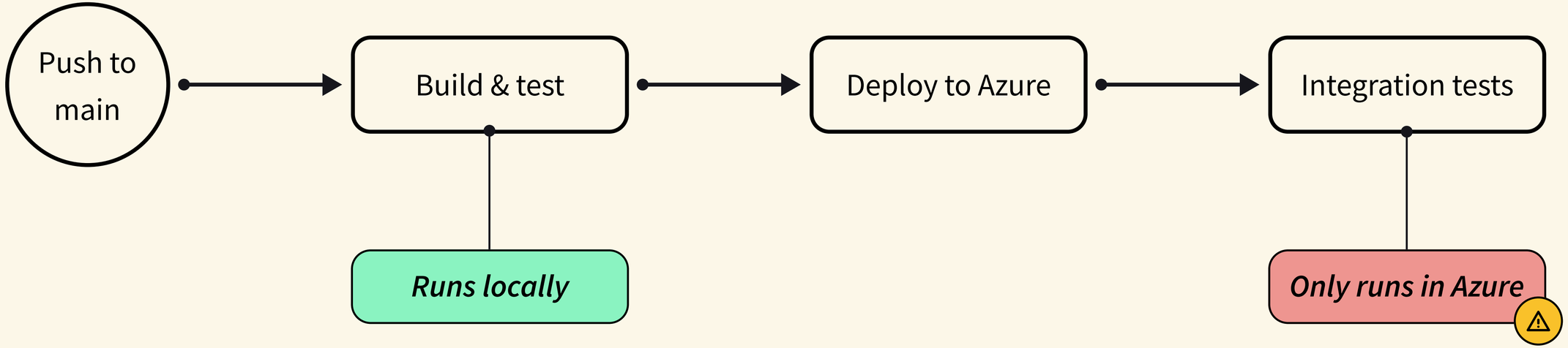
30+ dependencies



The Challenge

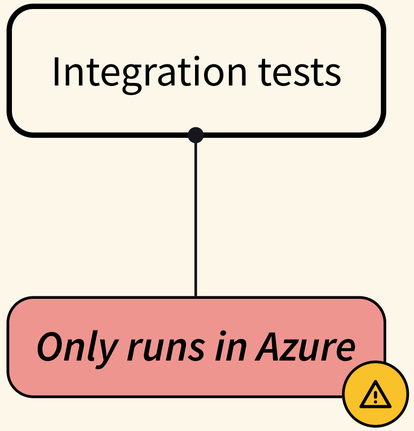
DevOps

Azure pipeline



DevOps

- A 15 minute feedback loop from changes
- Very little information available on why the test failed
  - Usually poor error messages: “Expected true, got false”
  - No logs from dependent services
- No pull-request pipeline flow for testing
- Tests and setup depends on static data inaccessible to developers



Developers want breakpoints!

My Solution

Local dev environment

- Wanted a fully local environment, with no external dependencies
- As few code changes as possible, external not internal mocks
- No Azure dependencies, we share resources, so it gets messy quickly

BEFORE	AFTER
Microsoft Entra	Mock Graph API
Azure Key-Vault	Minimal API mock
Azure App Configuration	Minimal API mock
Azure Table Storage	Azurite
Cosmos Database	Cosmos emulator
Sessionize	WireMock API
Learnster	WireMock API
Application insights	OpenTelemetry

Existing simulators

Cosmos emulator

- Developed by Microsoft
- Runs in a Docker container
- New version in development, still unstable
- Windows users can fall back to the old emulator

Azurite

- Developed by Microsoft
- Runs in a Docker container
- Simulates the Azure Storage service
- Has blob, queue, and table storage

New mocks

Mock Graph API

- An ASP.NET application
- Uses the ASP.NET OData library
- Custom implementation of the $search query
- Only mock the endpoints we use

Minimal API Mocks

- App Configuration API mocks
  - Mocks the key-value API for feature flags
- Key-vault API mocks
  - Mocks the secrets API

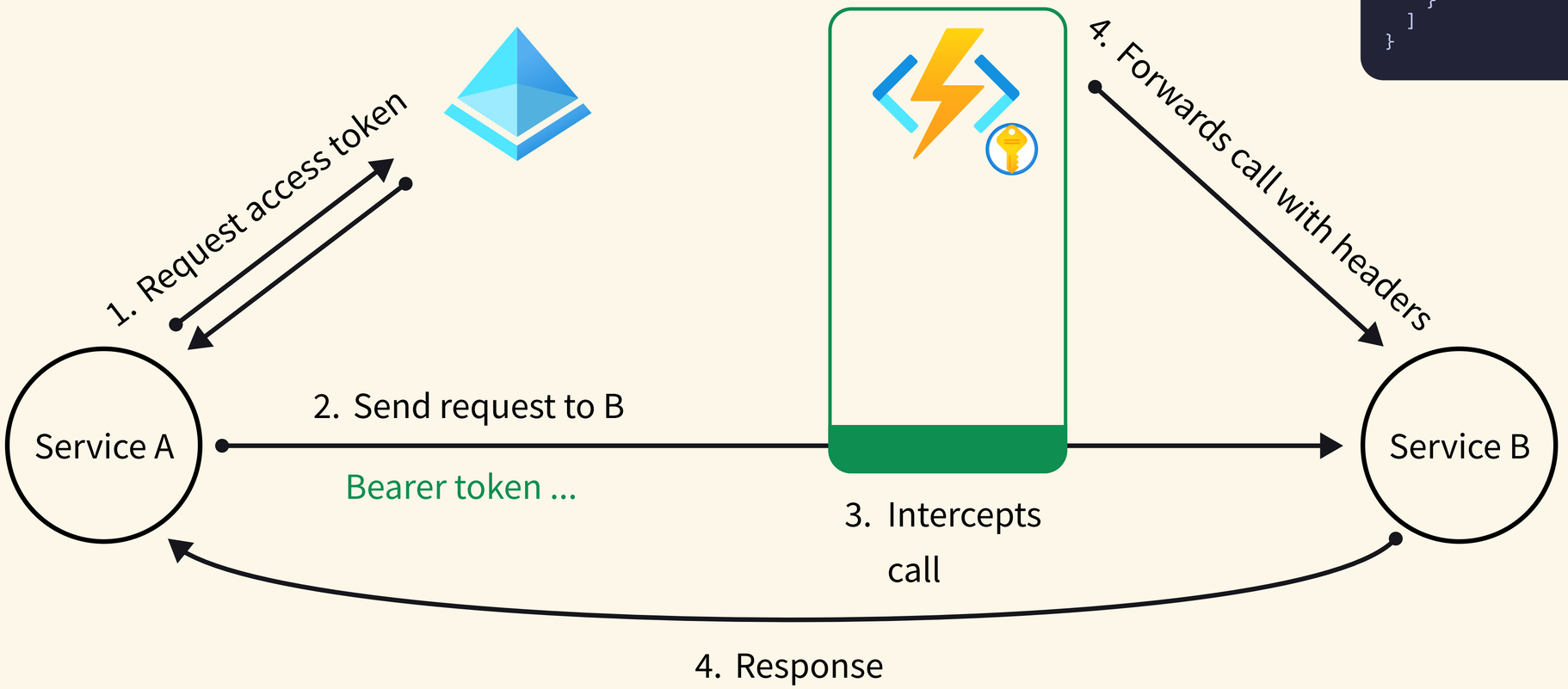
Sessionize and Learnster

Both Sessionize and Learnster are implemented as WireMock APIs. These are simple static responses at known addresses with no state.

```
{
  "mappings": [
    {
      "request": {
        "method": "GET",
        "url": "/bgd99iil/view/All"
      },
      "response": {
        "status": 200,
        "headers": {
          "Content-Type": "application/json"
        },
        "bodyFileName": "view-all-sessions-response.json"
      }
    },
    {
      "request": {
        "method": "GET",
        "url": "/empty/view/All"
      },
      "response": {
        "status": 200,
        "headers": {
          "Content-Type": "application/json"
        },
        "jsonBody": {
          "sessions": [],
          "categories": [],
          "questions": [],
          "rooms": []
        }
      }
    }
  ]
}
```

Authentication

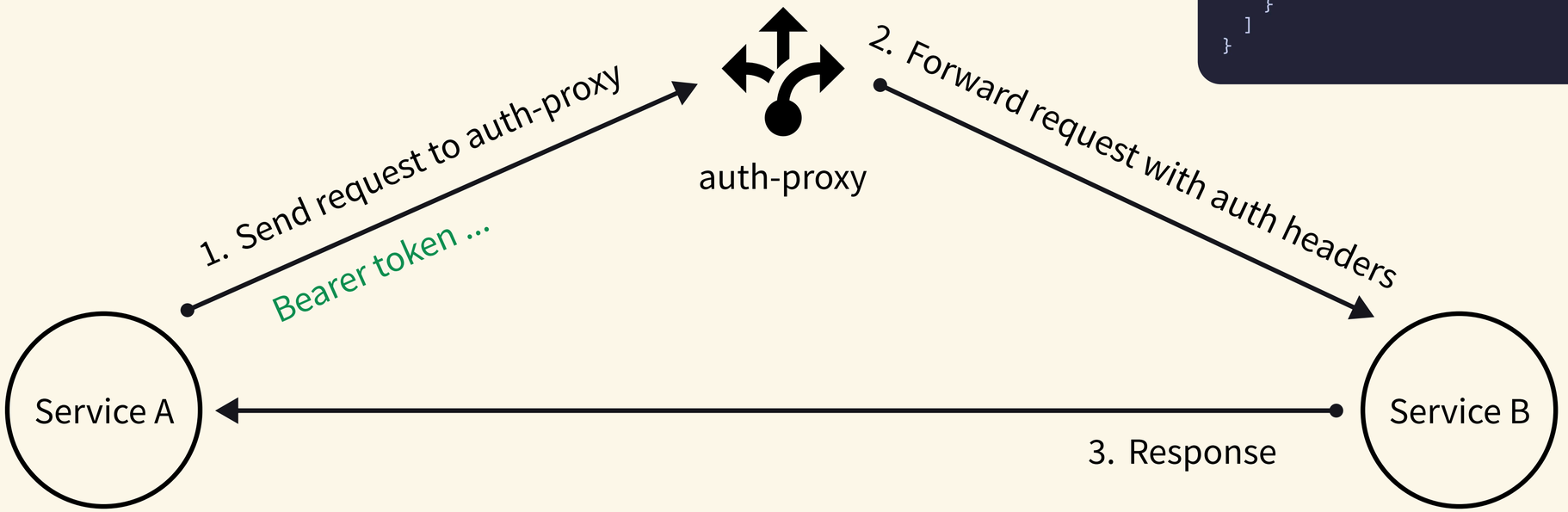
Azure auth flow




```
{
  "claims": [
    {
      "typ": "name",
      "val": "UserService.Read"
    },
    {
      "typ": "name",
      "val": "EventService.Write"
    }
  ]
}
```

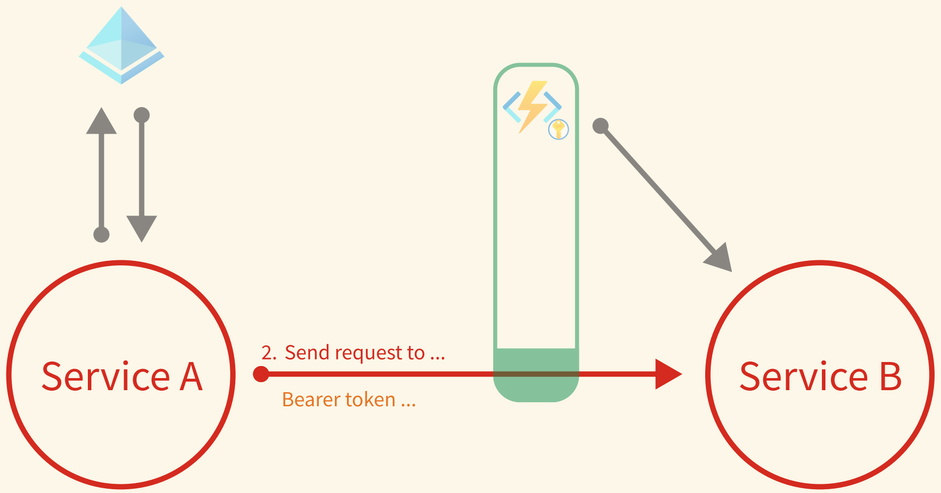
Authentication

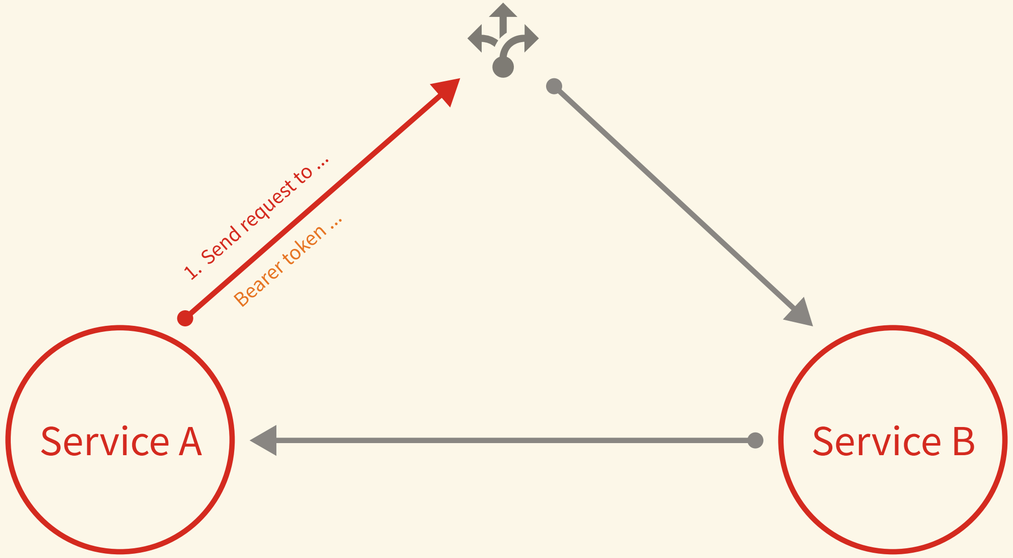
Local auth flow




```
{
  "claims": [
    {
      "typ": "name",
      "val": "UserService.Read"
    },
    {
      "typ": "name",
      "val": "EventService.Write"
    }
  ]
}
```

Authentication





THE SOLUTION

AZURE AUTHENTICATION

```
protected override async Task<HttpResponseMessage> SendAsync(
    HttpRequestMessage request,
    CancellationToken cancellationToken
)
{
    var scope = EndpointDataCollection.GetScopeForUrl(
        request.RequestUri.OriginalString
        );
    var tokenRequestContext =
        new TokenRequestContext([
            Environment.GetEnvironmentVariable("FUNCTIONS_APP_REG_ID")
            + "/.default"
        ]);
    var accessToken = await new DefaultAzureCredential()
      .GetTokenAsync(tokenRequestContext, cancellationToken);

    var credential = new ChainedTokenCredential(
        new ManagedIdentityCredential(),
        new AzureCliCredential());

    request.Headers.Authorization = new AuthenticationHeaderValue(
        "Bearer",
        accessToken.Token);
    return await base.SendAsync(request, cancellationToken);
}
```

LOCAL AUTHENTICATION

```
protected override async Task<HttpResponseMessage> SendAsync(
    HttpRequestMessage request,
    CancellationToken cancellationToken
)
{
    request.Headers.Authorization = new AuthenticationHeaderValue(
        "Bearer",
        "event-service-token");
    return await base.SendAsync(request, cancellationToken);
}
```

Containerization and orchestration

In total, we now have 12 function apps, 9 emulators, a DB seeder script, and the auth proxy. We need some way to help us start all of this whenever we want to debug something or run integration tests.

- Docker containers for each of the function apps and mock APIs
- Docker compose for orchestration
- Build containers in Azure pipelines and upload to the Azure Container registry
- Organized using profiles so one can choose which services to start

THE SOLUTION

Containerization and orchestration

One can replace dependent containers with locally running ones using docker-compose.overrides.yml

```yaml
services:
  auth-proxy:
    extra_hosts:
    - host.docker.internal:host-gateway
    environment:
      DestinationEndpoints__AcademyRecordService: http://host.docker.internal:${ARS_PORT}/api/
  academy-record-service:
    image: curlimages/curl:latest
    command: sh -c "while true; do sleep 3600; done"
    healthcheck:
      test:
      - CMD
      - curl
      - -f
      - http://host.docker.internal:${ARS_PORT}/health
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 10s
    extra_hosts:
    - host.docker.internal:host-gateway
    ports: !reset []
    expose: !reset []
    volumes: !reset []
    environment: !reset []
```

Demo

Real services vs emulation

Using real services

- Accurate
- Complete
- Increased cost
- Latency
- Concurrency and persistence

Using emulators

- Consistent
- Fast
- Only test business logic
- Partial

What do we actually need to emulate?

- Databases and external datastores are easier to have running locally
  - Things you mutate should in general be your own
- Should always mock external APIs unless they have dedicated test environments you can interact with
- Can have dev versions of key-vaults and app-configurations
- Entra is hard...

Questions?