AWS re:Invent

WIN401

Architecting ASP.NET Core Microservices Applications on AWS

Kirk Davis
Sr. Specialized Solutions Architect
AWS

Nicki Klein
Sr. Developer Advocate
AWS

AWS re:Invent

Agenda

Why .NET Core?

Architecture goal: microservices w/o managing infrastructure

Architecture overview

Authentication & authorization

Distributed tracing and logging

Session state in Amazon DynamoDB—why and how

Organizing the code (solution, projects, folders, git repos)

The CI/CD pipeline for this project

Related sessions

Thursday, November 29

WIN304 Building Well Architected .NET Apps (breakout)

11:30 am – 12:30 pm | MGM, Level 3, Premier Ballroom 316, T1

Tuesday, November 27

WIN323 Deploying serverless .NET Applications (chalk talk)

11:30 am – 12:30 pm | Venetian, Level 2, Veronese 2406, T1

Wednesday, November 28

WIN306 Simplifying Microsoft Architectures with AWS Services (breakout)

4:45 pm – 5:45 pm | MGM, Level 3, Premier Ballroom 319

Why .NET Core?

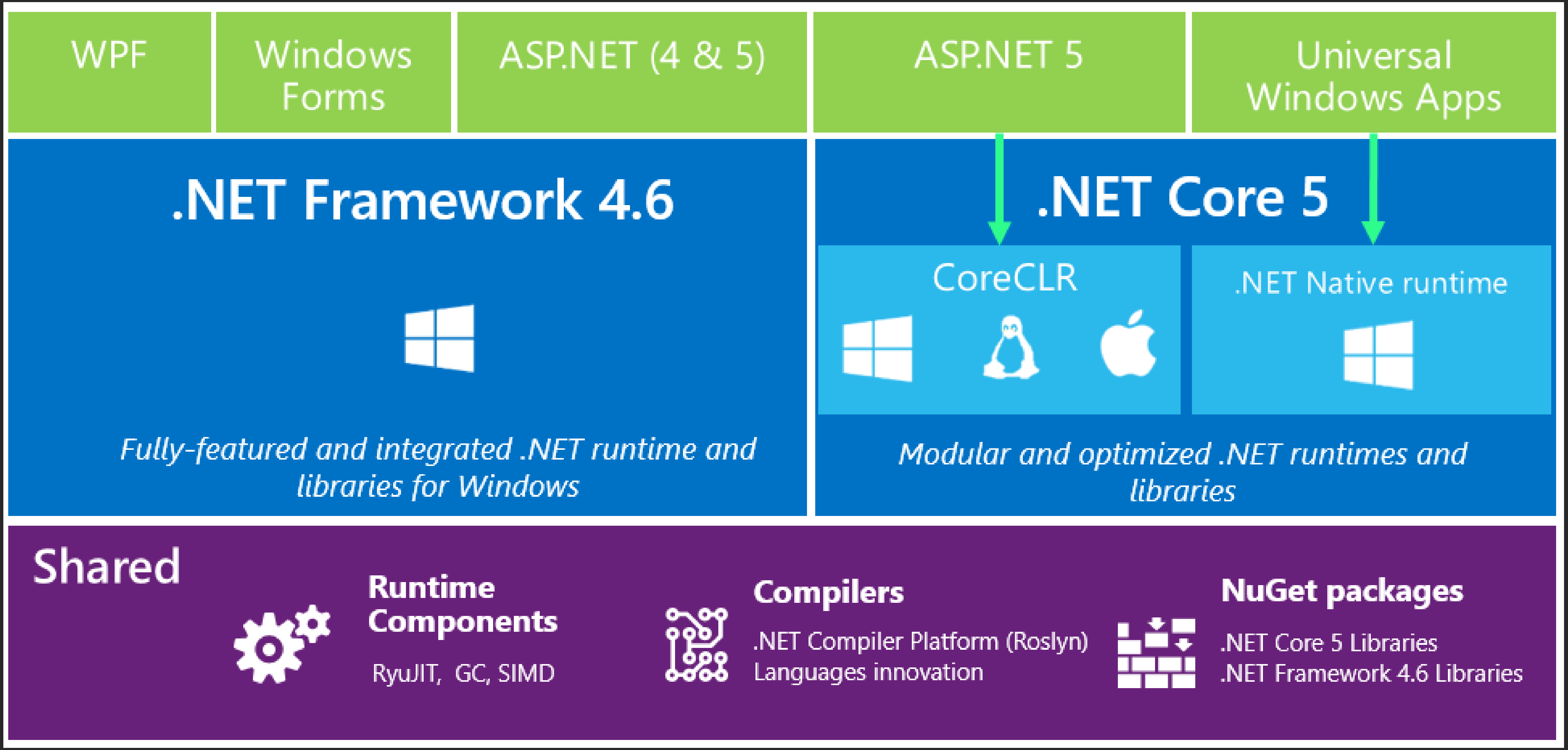
.NET Core is:

- The future of .NET
- X-Platform: Windows, macOS, Linux
- Modular, light weight, and open source
- Runs in AWS Lambda, AWS Fargate, Amazon ECS, Amazon EKS, Amazon EC2 (full compute spectrum)

Customers have in-house skills, tools, and experience with .NET

Performance: .NET Core performs faster than Node.js and Java in numerous benchmarks, and 2.1 brought even faster performance

.NET history


WPF
Windows Forms
ASP.NET (4 & 5)
ASP.NET 5
Universal Windows Apps
.NET Framework 4.6
Fully-featured and integrated .NET runtime and libraries for Windows
.NET Core 5
CoreCLR
.NET Native runtime
Modular and optimized .NET runtimes and libraries
Shared
Runtime Components
RyuJIT, GC, SIMD
Compilers
.NET Compiler Platform (Roslyn)
Languages innovation
NuGet packages
.NET Core 5 Libraries
.NET Framework 4.6 Libraries

"I've always loved C# … It's a truly modern language."

Werner Vogels,
CTO, AWS

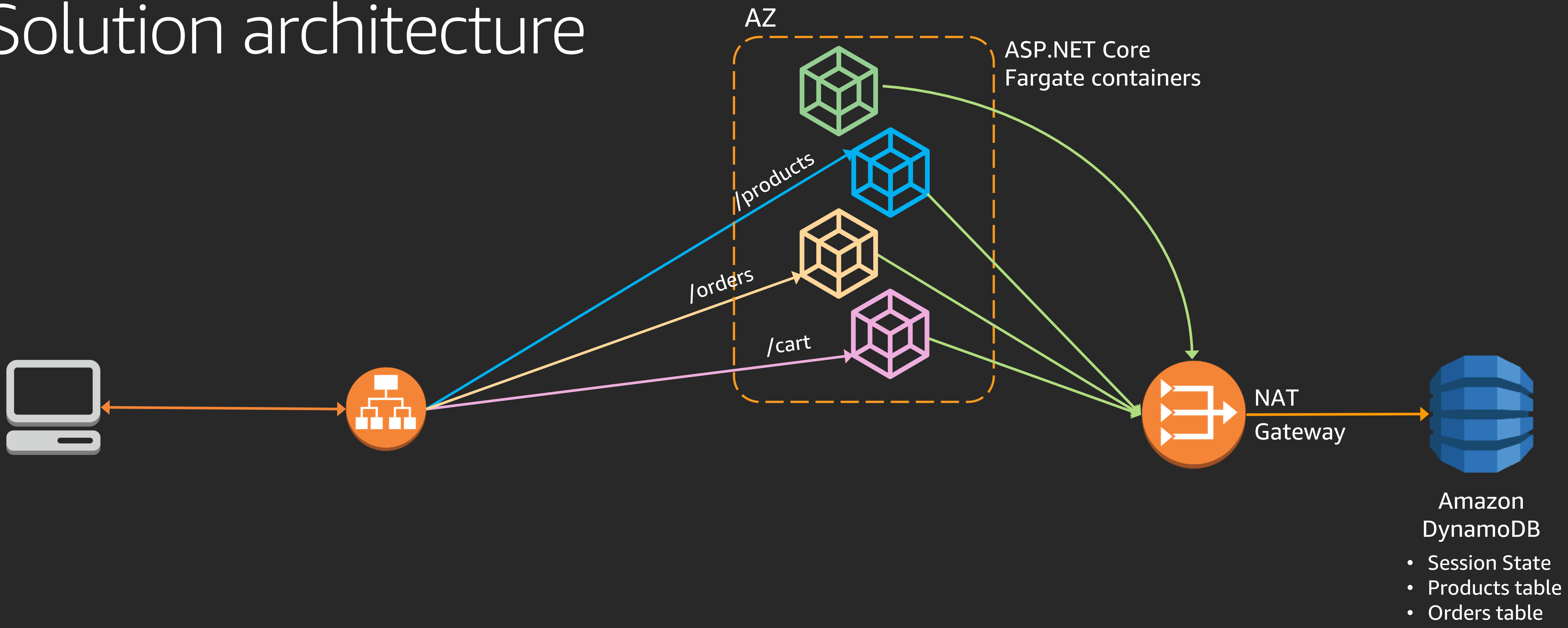
Architecture considerations & constraints

- Highly available and easily scalable application
- Microservices architecture
- Services available from both internet and backend (other services)
  - Some services private only, not accessible via the internet
  - Health checks on our microservices
- Support for both anonymous and authenticated users
- No infrastructure to manage
- Logging and request tracing
- Automated CI/CD pipeline

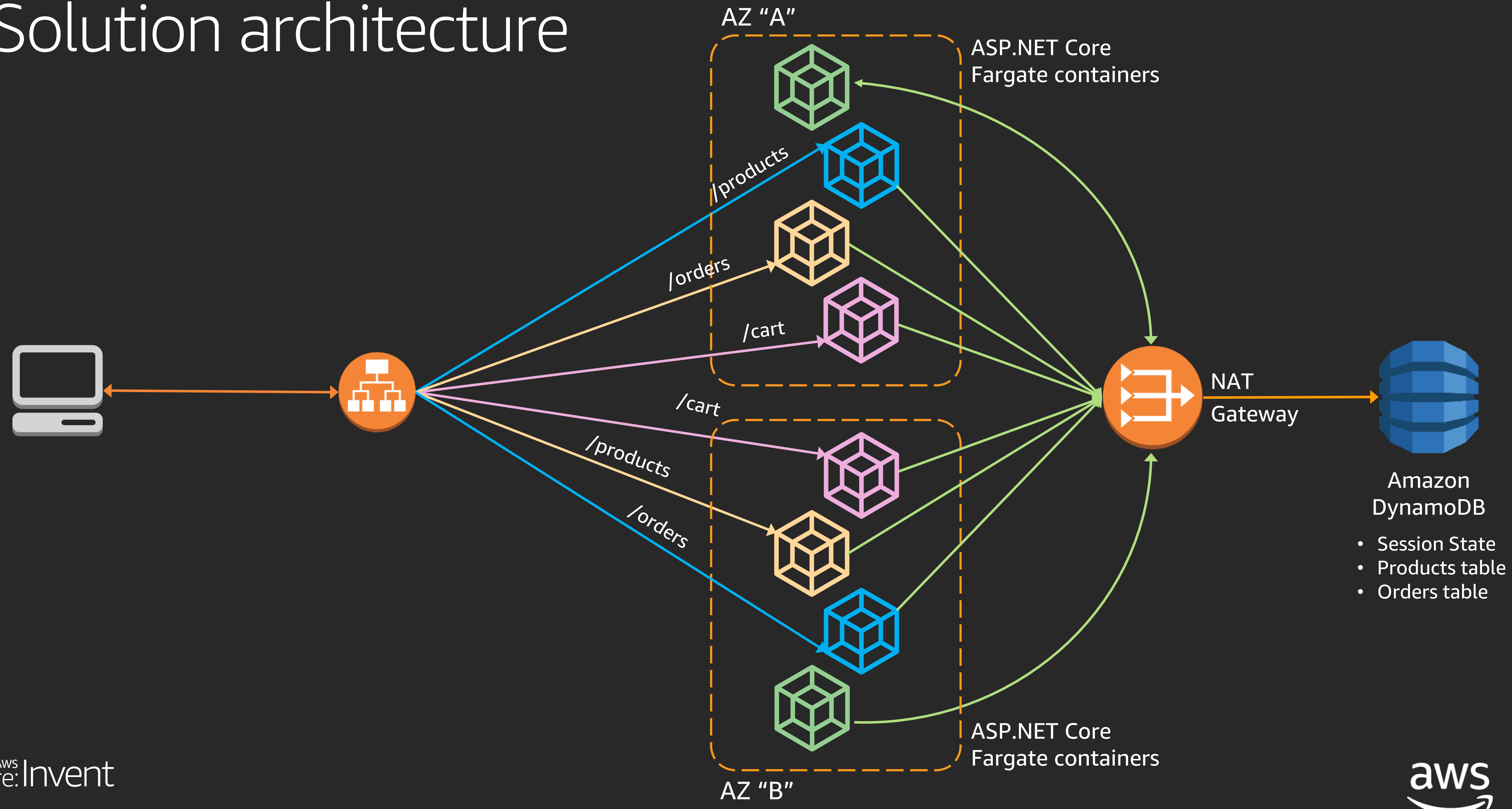
Multiple approaches/choices

- No-infrastructure compute
  - AWS Lambda (serverless)
  - AWS Fargate (containers on Amazon ECS)
- No-infrastructure storage
  - Amazon S3
  - Amazon DynamoDB
- Identity management (authentication)
  - Amazon Cognito
  - Third-party IdP
  - Build your own…

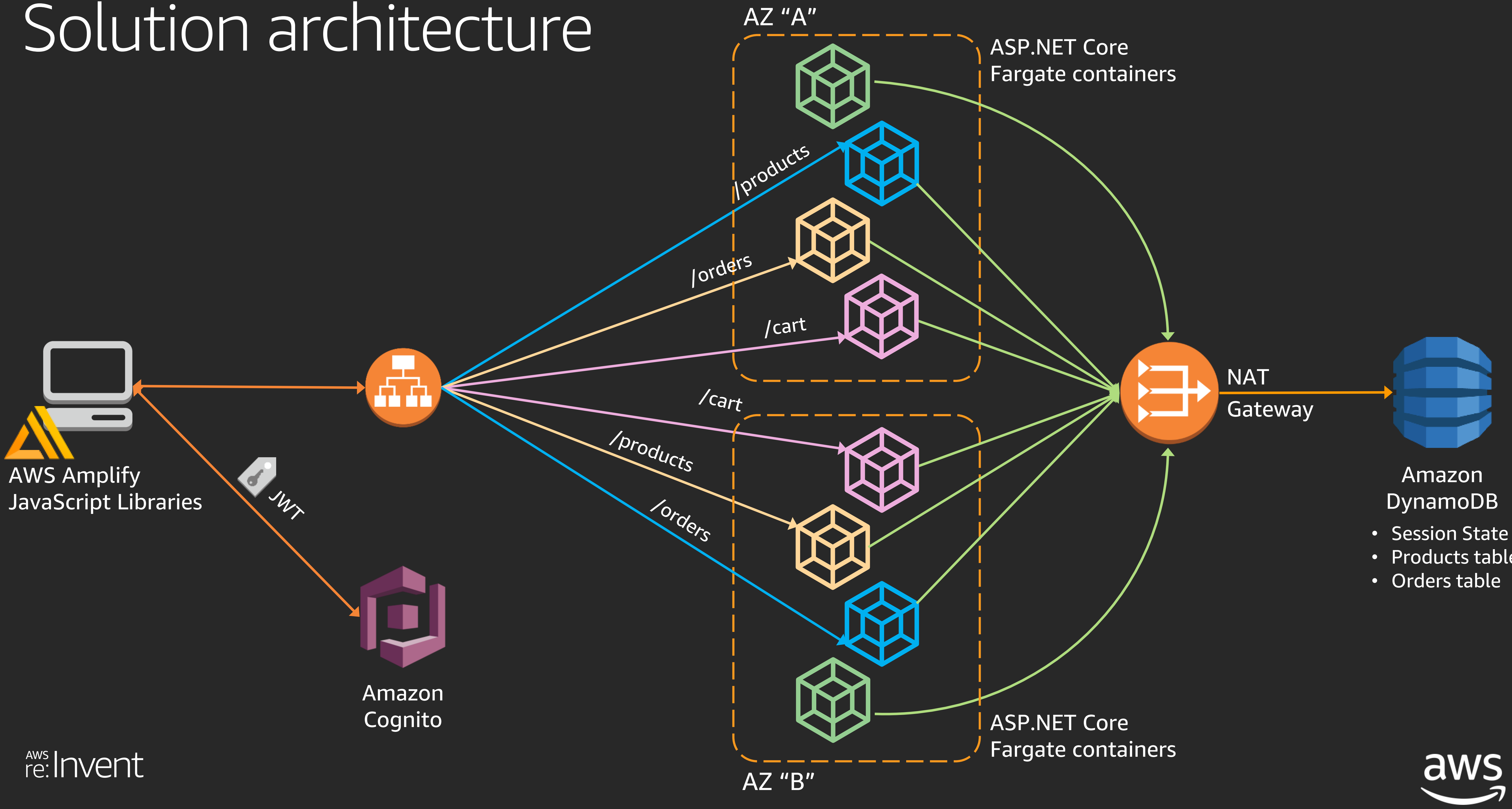
Solution architecture



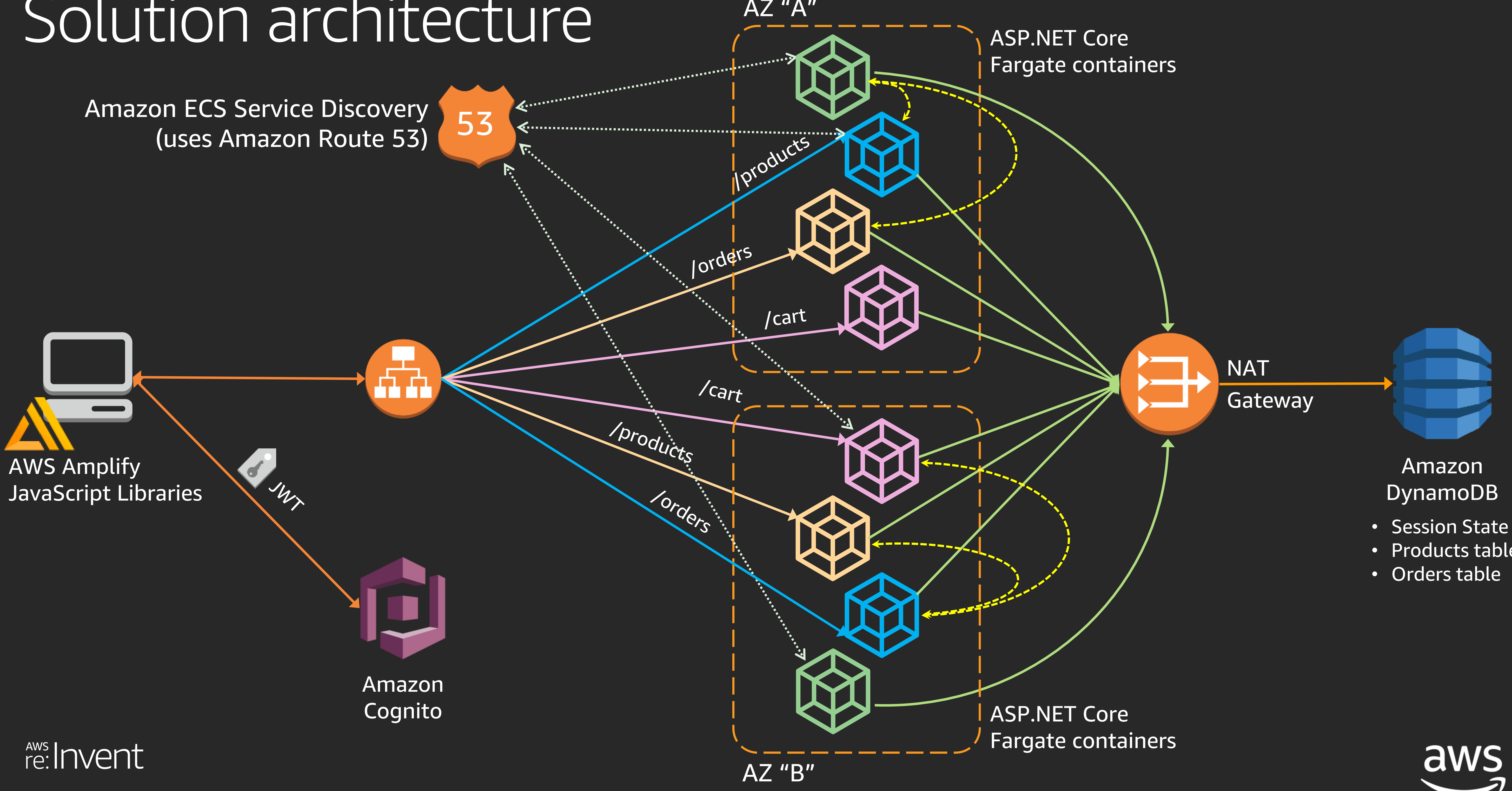
Solution architecture



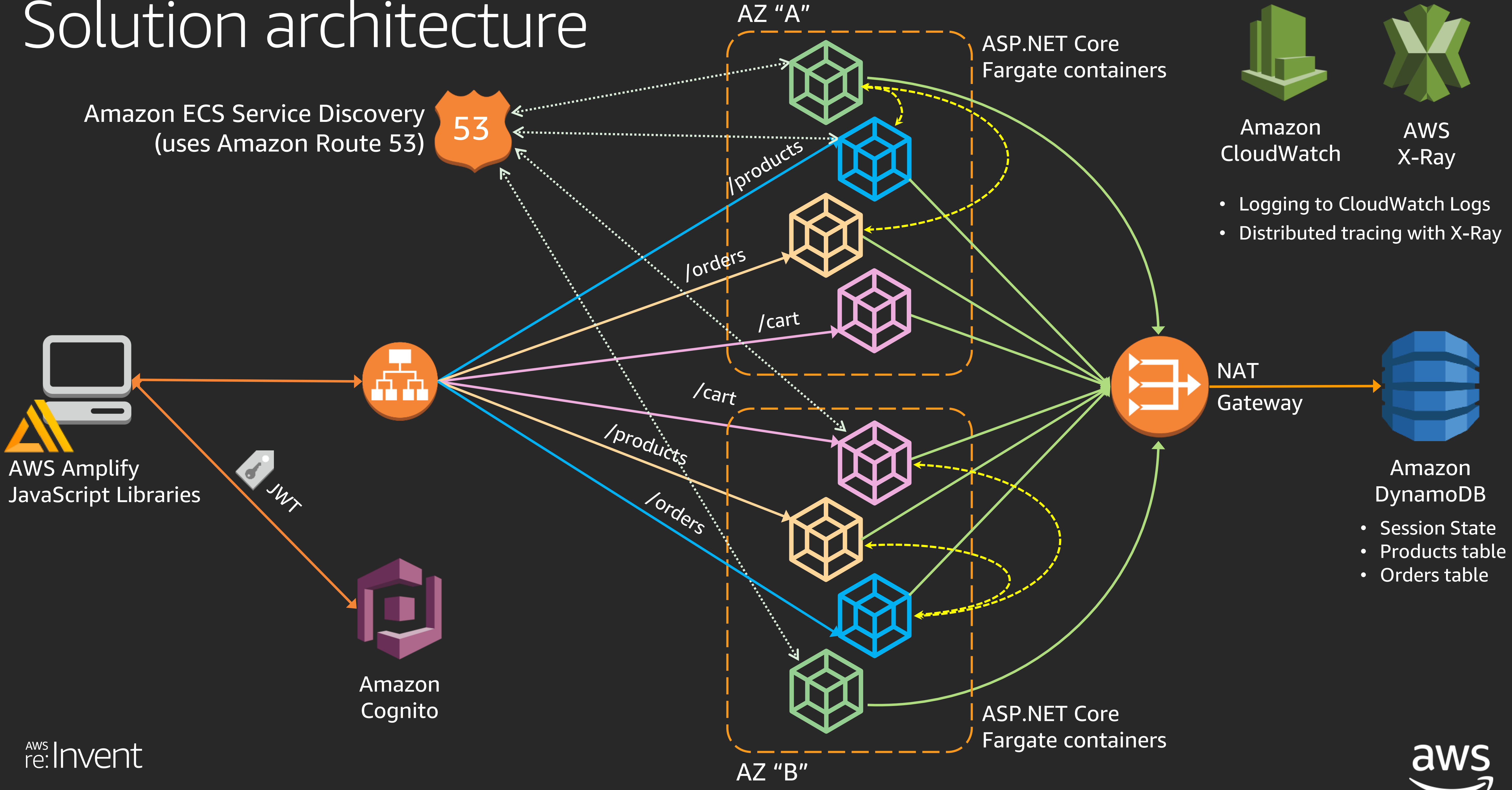
Solution architecture



Solution architecture



Solution architecture



- Logging to CloudWatch Logs
- Distributed tracing with X-Ray

- Session State
- Products table
- Orders table

AWS re:Invent

aws

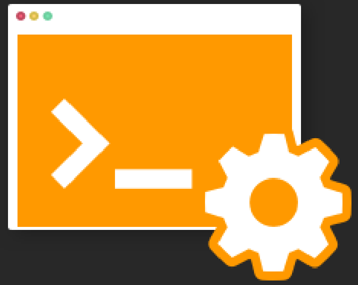
Web front-end

- Our front-end is static:
  - Built with Angular 6 (HTML, CSS, JavaScript)
  - Uses AWS Amplify to simplify Amazon Cognito auth & signup flows
  - Hosted in Amazon S3 bucket with static website hosting
  - No infrastructure, and extremely cost efficient

AWS Amplify: open-source foundation for web-apps!

https://aws-amplify.github.io/

Easy-to-use library



Powerful toolchain

Beautiful UI components

Service routing & service discovery

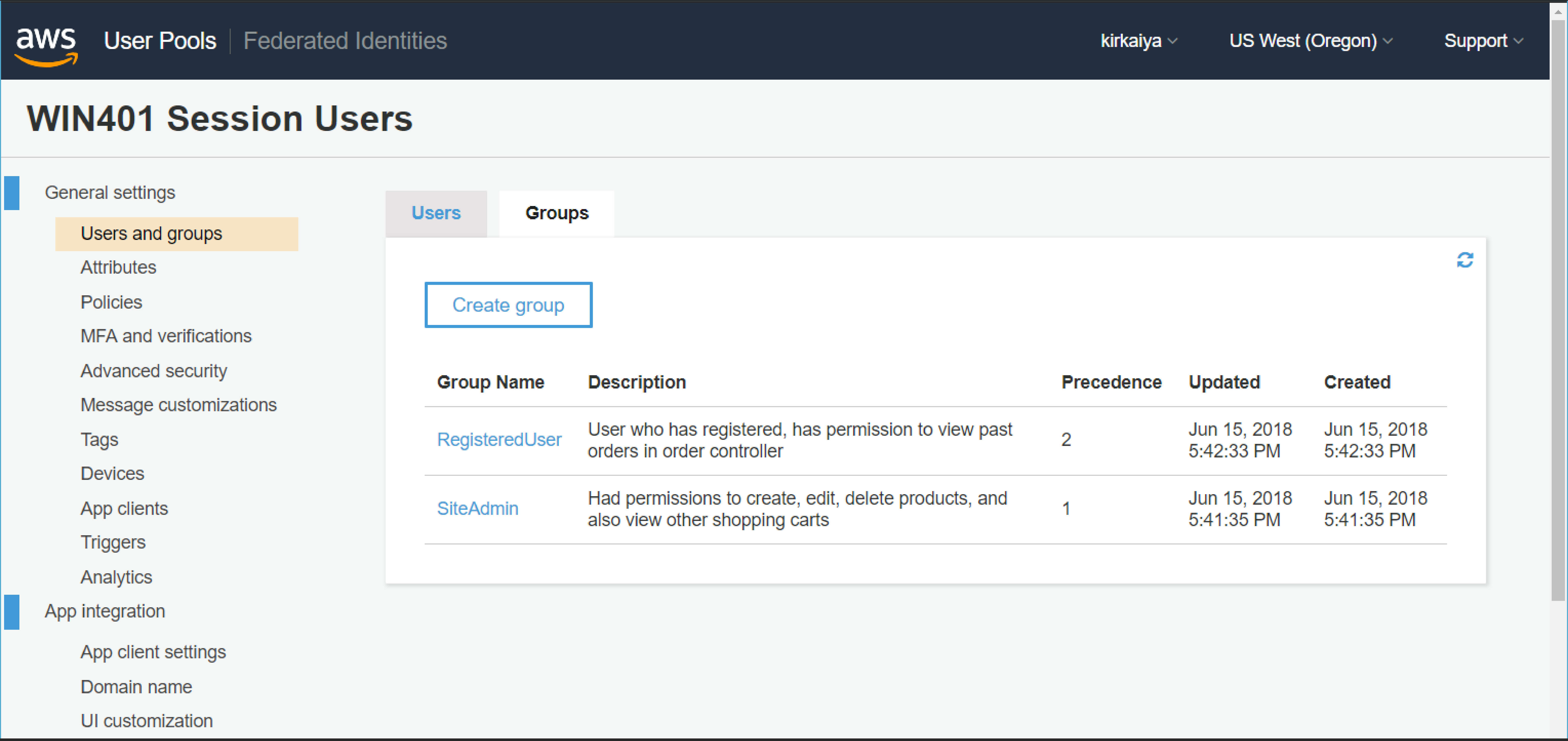
<u>External (JavaScript/browser) callers</u>

- ALB Target Groups, with path-based routing
- Integrated health checks
- Multi-AZ
- Paths:
  - /orders
  - /products
  - /cart
  - / *[root]*

<u>Internal (MVC & microservice) callers</u>

- ECS Service Discovery, built on Route 53 (using A records*)
- Integrated health checks
- Multi-AZ
- Service (DNS) names:
  - orders.techsummit
  - products.techsummit
  - cart.techsummit
  - *Other private services*

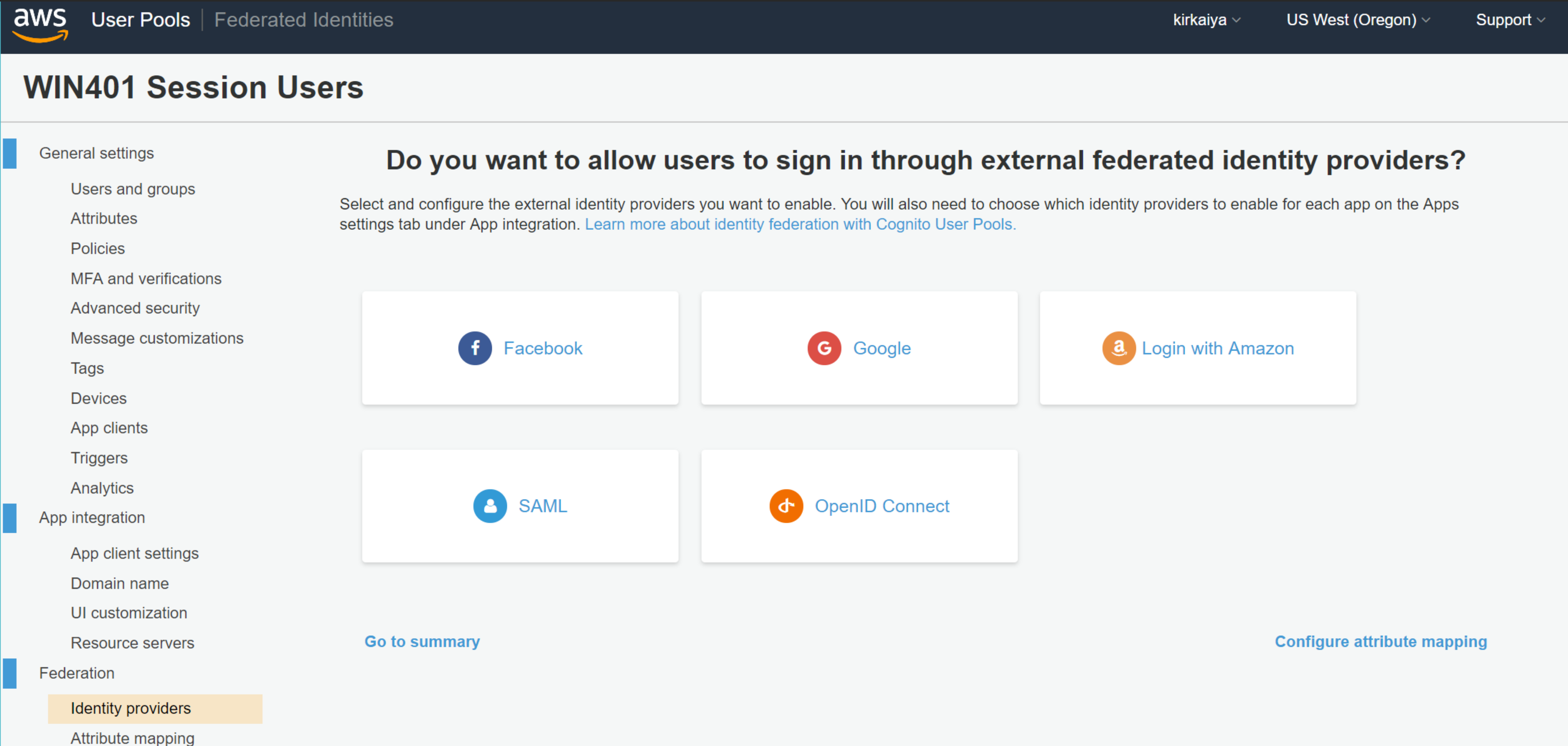
Identity management: Amazon Cognito user pools



WIN401 Session Users

Group Name	Description	Precedence	Updated	Created
RegisteredUser	User who has registered, has permission to view past orders in order controller	2	Jun 15, 2018 5:42:33 PM	Jun 15, 2018 5:42:33 PM
SiteAdmin	Had permissions to create, edit, delete products, and also view other shopping carts	1	Jun 15, 2018 5:41:35 PM	Jun 15, 2018 5:41:35 PM

Amazon Cognito user pools—external identity providers



WIN401 Session Users

Do you want to allow users to sign in through external federated identity providers?

Select and configure the external identity providers you want to enable. You will also need to choose which identity providers to enable for each app on the Apps settings tab under App integration. Learn more about identity federation with Cognito User Pools.

Amazon Cognito + custom authorizer for controllers

Options for using Amazon Cognito groups for authorization with ASP.NET

- Use API Gateway w/integrated Amazon Cognito Authorization
  - Pros: can vary authorization by request type (GET/PUT/POST/etc.)
  - Cons: need additional service, complicates service-to-service authorization
- Use ALB Integrated Amazon Cognito Authorization feature
  - It’s an integrated feature of ALB
  - Cons: authorization is per-path only (e.g., per microservice)
- Custom authorization-handler in C#
  - Pros: granular authorization per controller method and request type, easy proxying of user JWT from service to service, use [`Authorize`] attribute
  - Cons: need to write some code (not really a con!)

Custom authorization handler—the code

```
class CognitoGroupAuthorizationHandler :
        AuthorizationHandler<CognitoGroupAuthorizationRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                   CognitoGroupAuthorizationRequirement requirement)
    {
        if (context.User.HasClaim(c => c.Type == "cognito:groups" &&
                                       c.Value == requirement.CognitoGroup))
            {
                context.Succeed(requirement);
            }
            else
            {
                context.Fail();
            }
            return Task.CompletedTask;
    }
}
```

Custom authorization handler—register in startup

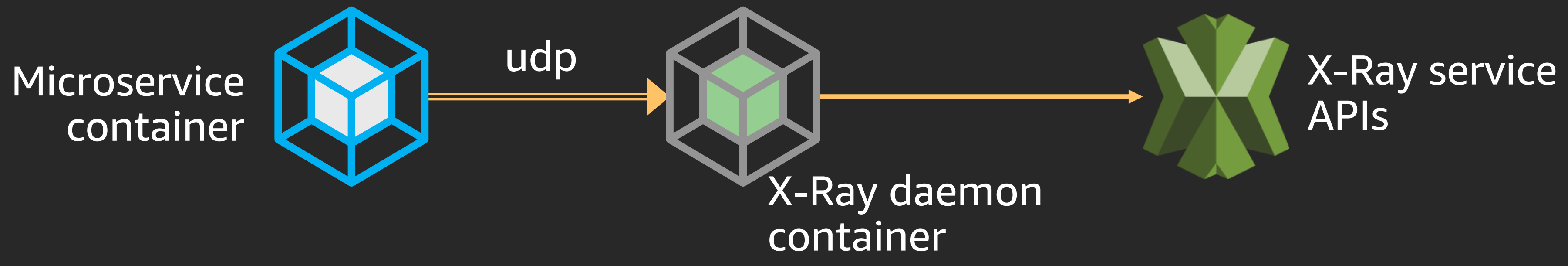
```
// Add Cognito group authorization requirements for SiteAdmin and RegisteredUser User Groups
services.AddAuthorization(
  options =>
  {
    options.AddPolicy("IsSiteAdmin", policy =>
      policy.Requirements.Add(new CognitoGroupAuthorizationRequirement("SiteAdmin"))
    );
    options.AddPolicy("IsRegisteredUser", policy =>
      policy.Requirements.Add(new CognitoGroupAuthorizationRequirement("RegisteredUser")));
  }
);
```

Custom authorization handler—usage example

```
// POST: api/Products
[HttpPost]
[Authorize(Policy = "InSiteAdminGroup")]
public async Task<string> Post([FromBody]Product product)
{
    // method code here...
    var context = new DynamoDBContext(_ddbClient);
    await context.SaveAsync(product);

    return product.ProductId;
}
```

Distributed tracing: AWS X-Ray



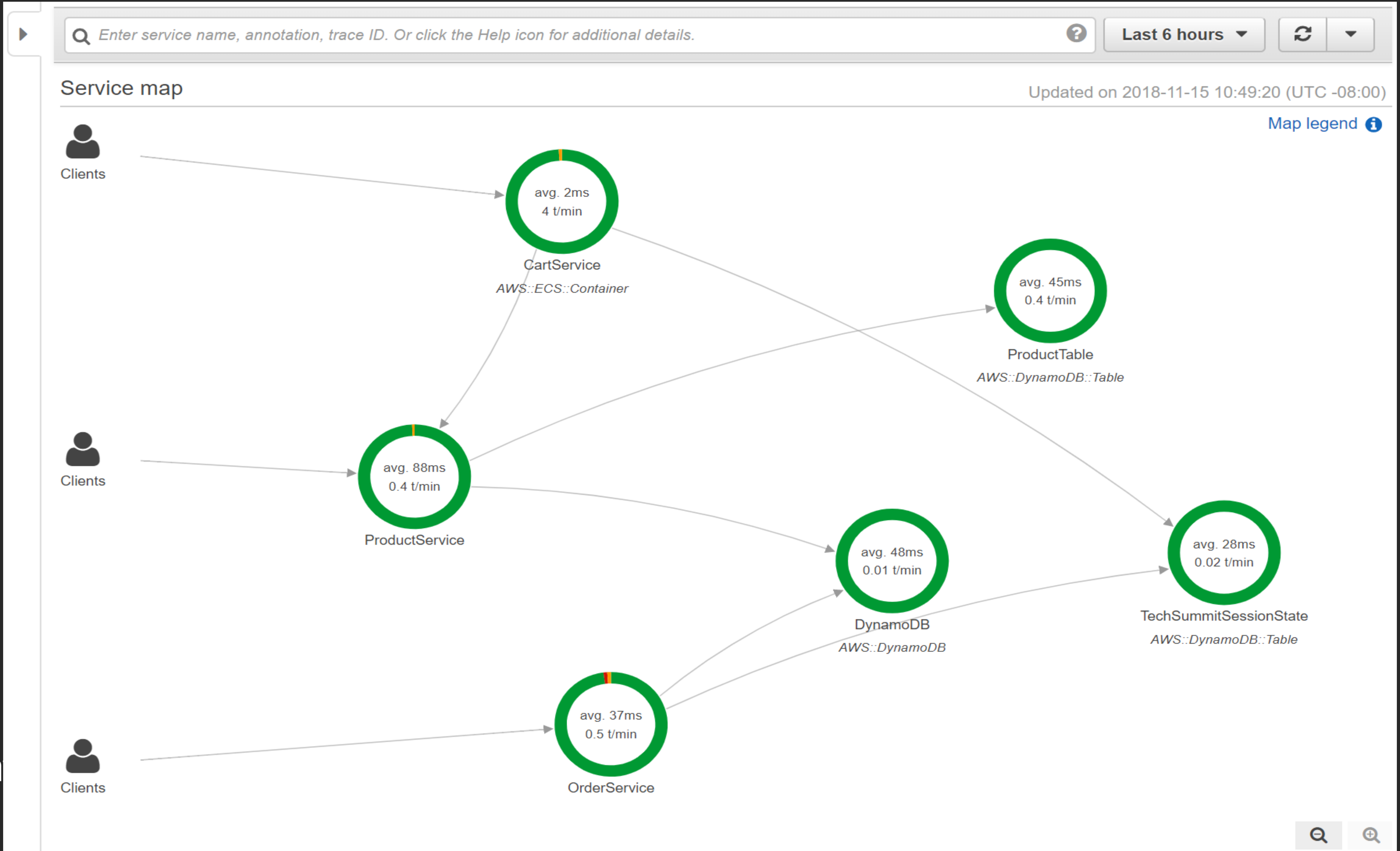
Benefits of using X-Ray

- Identify performance bottlenecks and exceptions
- Pinpoint issues experienced to specific service(s)
- Identify impact of issues on users
- Visualize the call-graph for requests

Using X-Ray for ASP.NET Core apps:

- NuGet package AWSXrayRecorder (meta-package)
- Wire up in startup
- Install X-Ray daemon (for AWS Fargate, run daemon in side-car container)

X-Ray service-map visualization



Consolidated logging: Amazon CloudWatch Logs

- CI/CD build logs
  - AWS CodeBuild automatically logs to CloudWatch Logs
  - Includes output from `docker build`, `dotnet build` (all commands in Dockerfile)
- Container logs
  - ECS logs container-level messages, error codes, and more
- Application (service-level) logs
  - Can log full ASP.NET Core lifecycle (all requests, controller method entry, and more)
  - Your application logging code
  - NuGet package AWS.Logger.AspNetCore

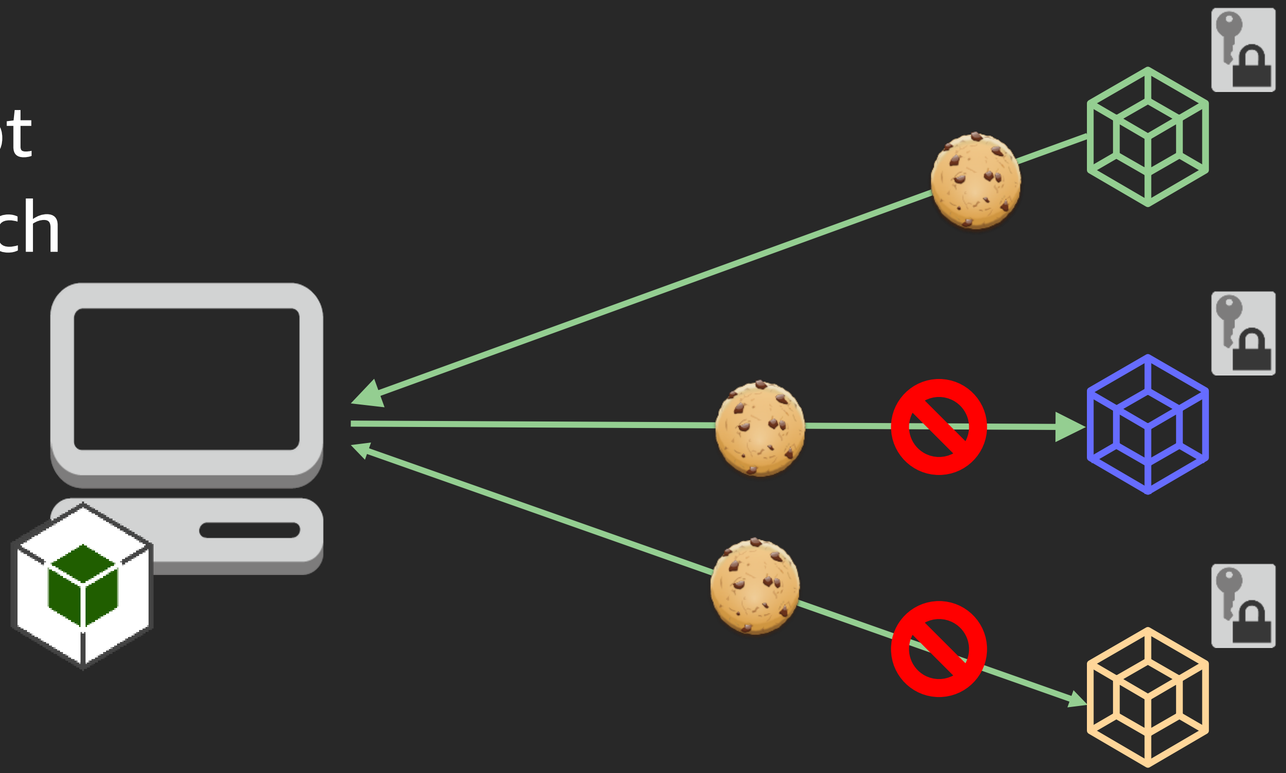
Session in ASP.NET Core

- You may not need to track sessions in a micro-services application
- Can be useful for tracking non-authenticated users, or if you're using cookie-based authentication with OIDC, local accounts, or others
- ASP.NET relies on encrypted session cookies to track users (the default .AspNetCore.Session cookie)
- Using session cookies with a distributed application takes some extra setup in ASP.NET Core

Session state in DynamoDB—sharing session

ASP.NET Core encrypts session cookies.

One microservice can't decrypt another service's cookies, which kills session state.

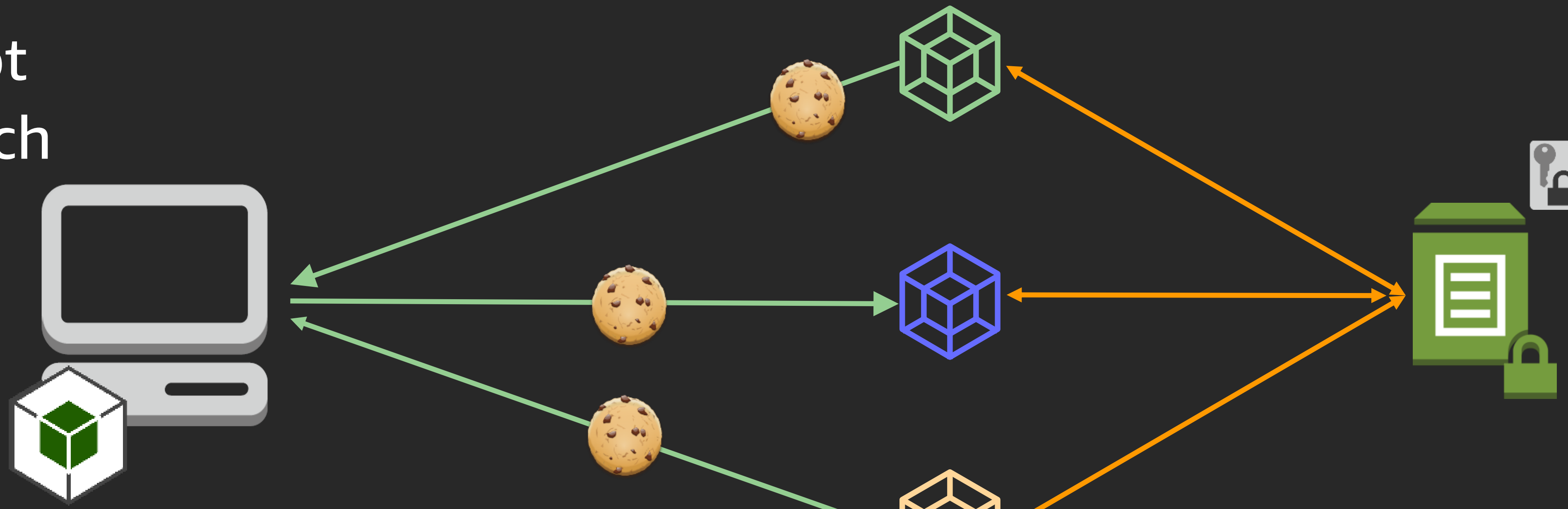


Session state in DynamoDB—sharing session

ASP.NET Core encrypts session cookies

One microservice can't decrypt another service's cookies, which kills session state

We can add a simple implementation of `IXmlRepository` to store the keys in Amazon DynamoDB (or another service)



IXmlRepository implementations

Choose a storage mechanism that supports encryption.

- AWS Secrets Manager
- AWS Parameter Store*
- Amazon DynamoDB
- Amazon S3

There are some implementations others have done on GitHub. The code is so simple, you can easily write your own also.

Cookie encryption keys in Parameter Store

```
public class ParameterStoreXmlRepository : IXmlRepository
```

```
public IReadOnlyCollection<XElement> GetAllElements()
{
    var request = new GetParametersByPathRequest { Path = "/CookieEncryptionKey" };
    var response = _client.GetParametersByPathAsync(request).Result;
    var result = new List<XElement>(response.Parameters.Count);
    response.Parameters.ForEach(x => result.Add(XElement.Parse(x.Value)));
    return result;
}
```

```
public void StoreElement(XElement element, string friendlyName)
{
    var request = new PutParameterRequest {
        Name = "/CookieEncryptionKey/" + friendlyName,
        Value = element.ToString(),
        Type = ParameterType.String
    };
    _client.PutParameterAsync(request);
}
```

Cookie encryption keys in DynamoDB

```
public class DdbXmlRepository : IXmlRepository
```

```
public IReadOnlyCollection<XElement> GetAllElements()
{
    var context = new DynamoDBContext(_dynamoDb);
    var search = context.ScanAsync<XmlKey>(new List<ScanCondition>());
    var results = search.GetRemainingAsync().Result;

    return results.Select(x => XElement.Parse(x.Xml)).ToList();
}
```

```
public void StoreElement(XElement element, string friendlyName)
{
    var key = new XmlKey
    {
        Xml = element.ToString(SaveOptions.DisableFormatting),
        FriendlyName = friendlyName
    };

    var context = new DynamoDBContext(_dynamoDb);
    context.SaveAsync(key).Wait();
}
```

Cookie encryption keys in DynamoDB—XmlKey

```
[DynamoDBTable("AspXmlKeys")]
public class XmlKey
{
    [DynamoDBHashKey]
    public string KeyId { get; set; } = Guid.NewGuid().ToString();
    public string Xml { get; set; }
    public string FriendlyName { get; set; }
}
```

Storing data in session

- You may not need to store data in session (modern browsers have local-storage options)
- Default session state store in ASP.NET Core is in-process
- Microsoft provides `IDistributedCache` implementations for SQL Server & Redis
- DynamoDB is fast, scalable, durable, and highly-available so …
- We created an IDistributedCache implementation backed by DynamoDB
- Use the DynamoDB TTL feature to expire items (rather than from code)

Session state in DynamoDB—IDistributedCache

```
public class DynamoDbCache : IDistributedCache { …
    private static Table _table;
    private readonly AmazonDynamoDBClient _client;
…
    public async Task<byte[]> GetAsync(string key, CancellationToken token =
                                       default(CancellationToken
    {
        var value = await _table.GetItemAsync(key);
        if (value == null || value["Session"] == null)
        {
            return null
        }
        return value["Session"].AsByteArray();
 }
```

AWS re:Invent

aws

Demo

Demo of the website + API endpoints

Organizing the code

Q: How do we let devs work on these microservices in a single view (single VS instance) yet deploy each separately?

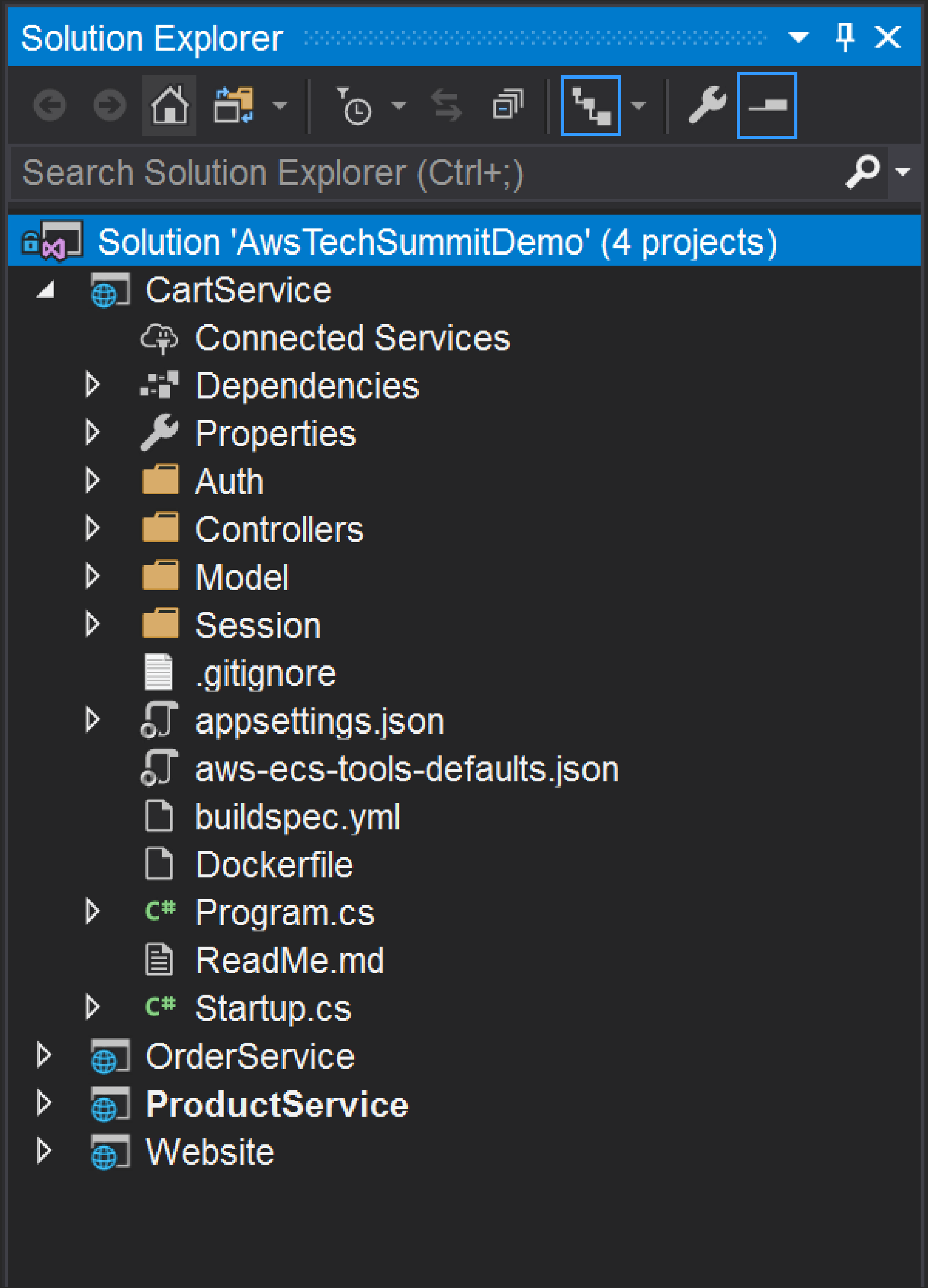
A: Multiple ways to solve this.

We chose one Solution file, stored in one repo (along with any solution-wide files), and each microservice in a project stored in its own folder that is its own git repo. Can test service-to-service communication locally with Visual Studio.

Total number of git repos = # of micro-services + 1

Alternatively, you could put everything in a single repo and do conditional build and container publishing in AWS CodeBuild.

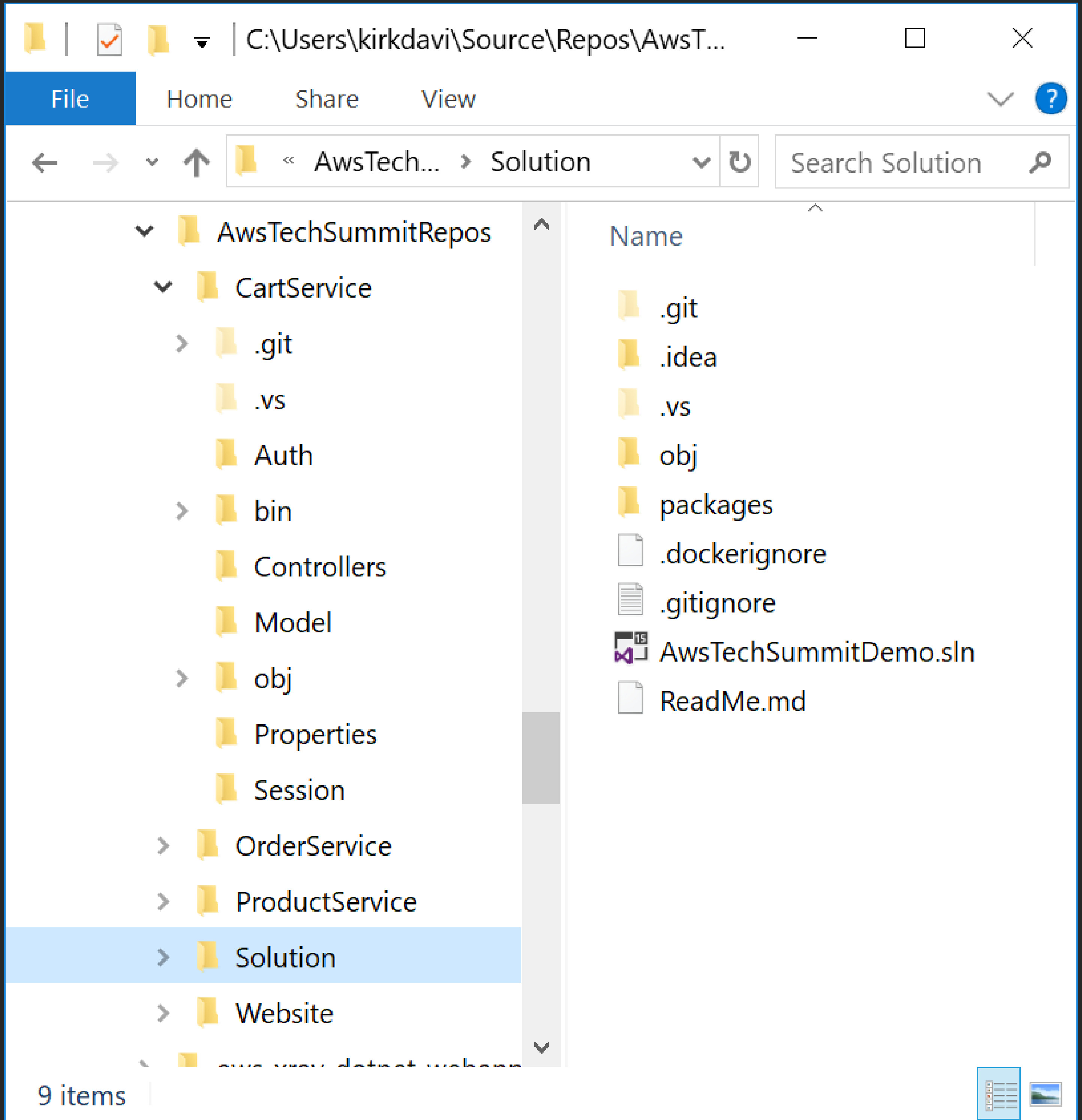
Organizing the code



Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'AwsTechSummitDemo' (4 projects)
- CartService
  - Connected Services
  - Dependencies
  - Properties
  - Auth
  - Controllers
  - Model
  - Session
  - .gitignore
  - appsettings.json
  - aws-ecs-tools-defaults.json
  - buildspec.yml
  - Dockerfile
  - Program.cs
  - ReadMe.md
  - Startup.cs
- OrderService
- **ProductService**
- Website



C:\Users\kirkdavi\Source\Repos\AwsT...

File Home Share View

« AwsTech... › Solution

Search Solution

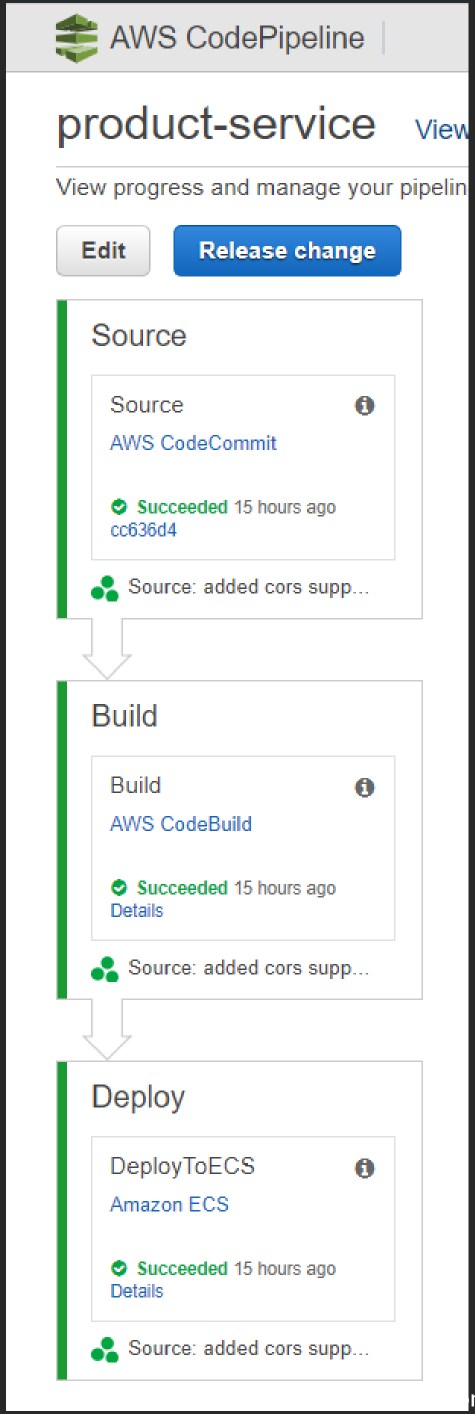
- AwsTechSummitRepos
  - CartService
    - .git
    - .vs
    - Auth
    - bin
    - Controllers
    - Model
    - obj
    - Properties
    - Session
  - OrderService
  - ProductService
  - Solution
  - Website

Name
- .git
- .idea
- .vs
- obj
- packages
- .dockerignore
- .gitignore
- AwsTechSummitDemo.sln
- ReadMe.md

9 items

The CI/CD pipeline

- 4 x AWS CodePipeline pipelines
- Each pipeline uses one AWS CodeCommit repo as source
- AWS CodeBuild (we didn't create any tests for this demo project)
- Deploy with CodePipeline's ECS Deployment Provider


AWS CodePipeline

product-service View

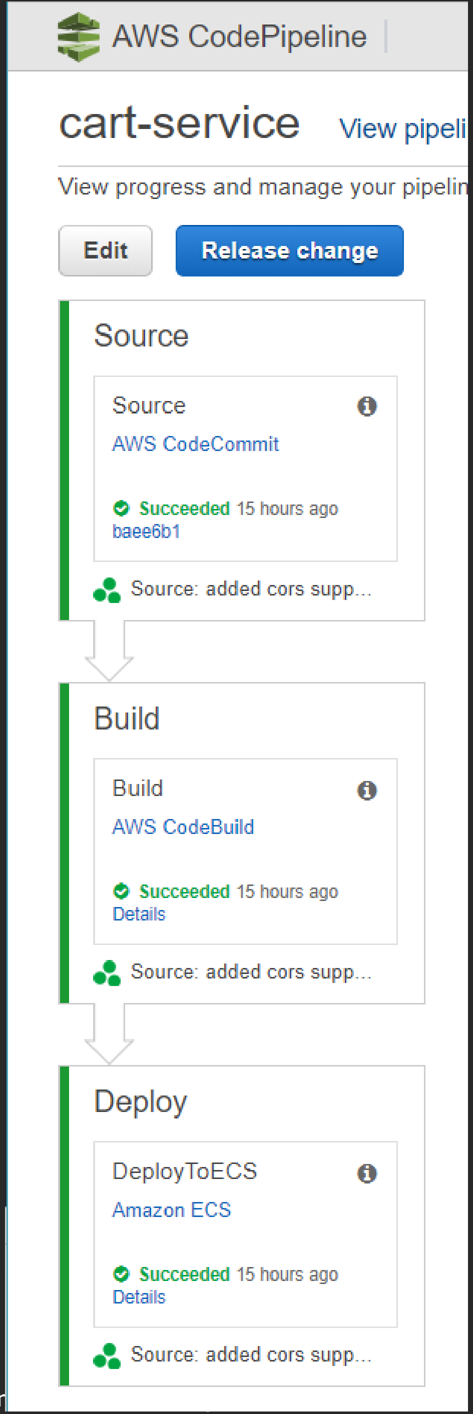
View progress and manage your pipelin

Edit | Release change

Source
Source
AWS CodeCommit
Succeeded 15 hours ago
cc636d4
Source: added cors supp...

Build
Build
AWS CodeBuild
Succeeded 15 hours ago
Details
Source: added cors supp...

Deploy
DeployToECS
Amazon ECS
Succeeded 15 hours ago
Details
Source: added cors supp...


AWS CodePipeline

cart-service View pipeli

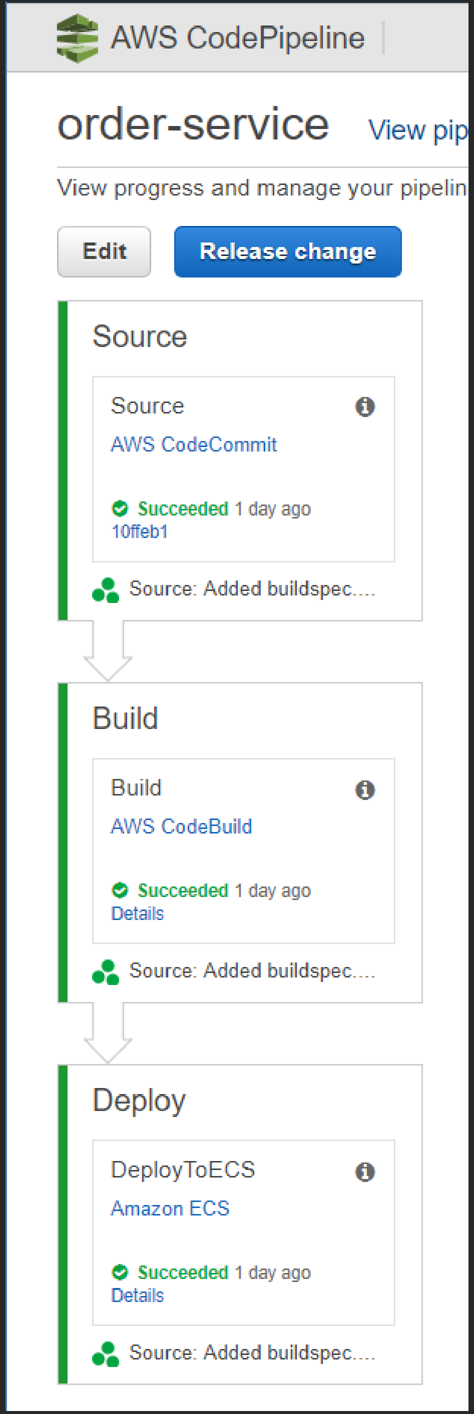
View progress and manage your pipelin

Edit | Release change

Source
Source
AWS CodeCommit
Succeeded 15 hours ago
baee6b1
Source: added cors supp...

Build
Build
AWS CodeBuild
Succeeded 15 hours ago
Details
Source: added cors supp...

Deploy
DeployToECS
Amazon ECS
Succeeded 15 hours ago
Details
Source: added cors supp...


AWS CodePipeline

order-service View pip

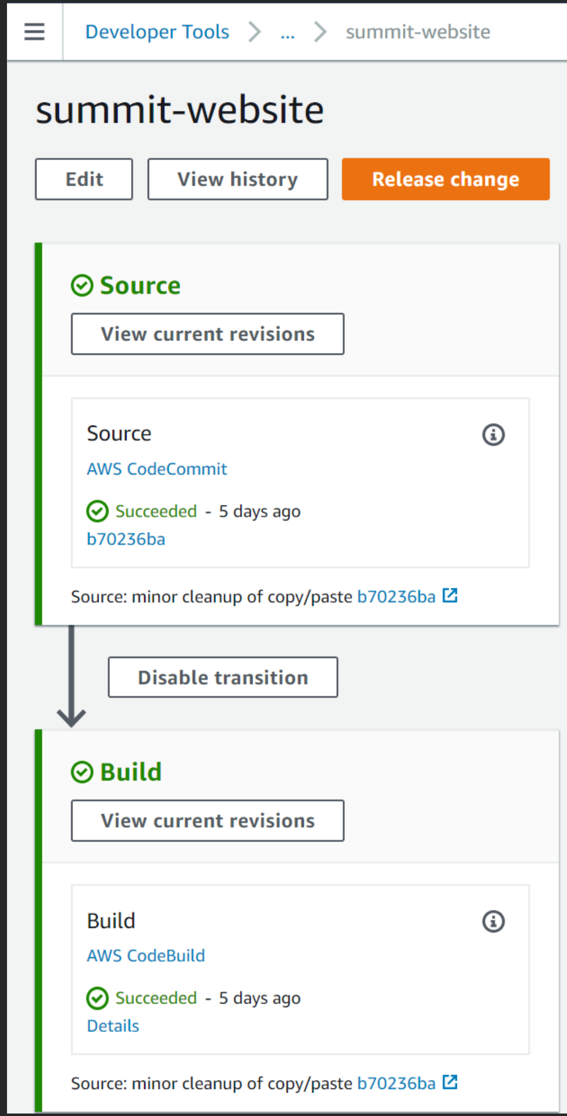
View progress and manage your pipelin

Edit | Release change

Source
Source
AWS CodeCommit
Succeeded 1 day ago
10ffeb1
Source: Added buildspec....

Build
Build
AWS CodeBuild
Succeeded 1 day ago
Details
Source: Added buildspec....

Deploy
DeployToECS
Amazon ECS
Succeeded 1 day ago
Details
Source: Added buildspec....


Developer Tools > ... > summit-website

summit-website

Edit | View history | Release change

Source
View current revisions
Source
AWS CodeCommit
Succeeded - 5 days ago
b70236ba
Source: minor cleanup of copy/paste b70236ba

Disable transition

Build
View current revisions
Build
AWS CodeBuild
Succeeded - 5 days ago
Details
Source: minor cleanup of copy/paste b70236ba

Demo of CI/CD pipeline

AWS re:Invent

aws

Thank you!

Kirk Davis kirkdavi@amazon.com

Nicki Klein nickik@amazon.com, @nicki_23 (twitter)

AWS re:Invent

aws

Please complete the session survey in the mobile app.