AWS re:Invent

WIN315-R

Converting a monolithic .NET app into a modern application

Artur Rodrigues
Solutions Architect – Public Sector
Amazon Web Services

Amit Jha
Solutions Architect – MSFT Technologies
Amazon Web Services

AWS re:Invent

aws

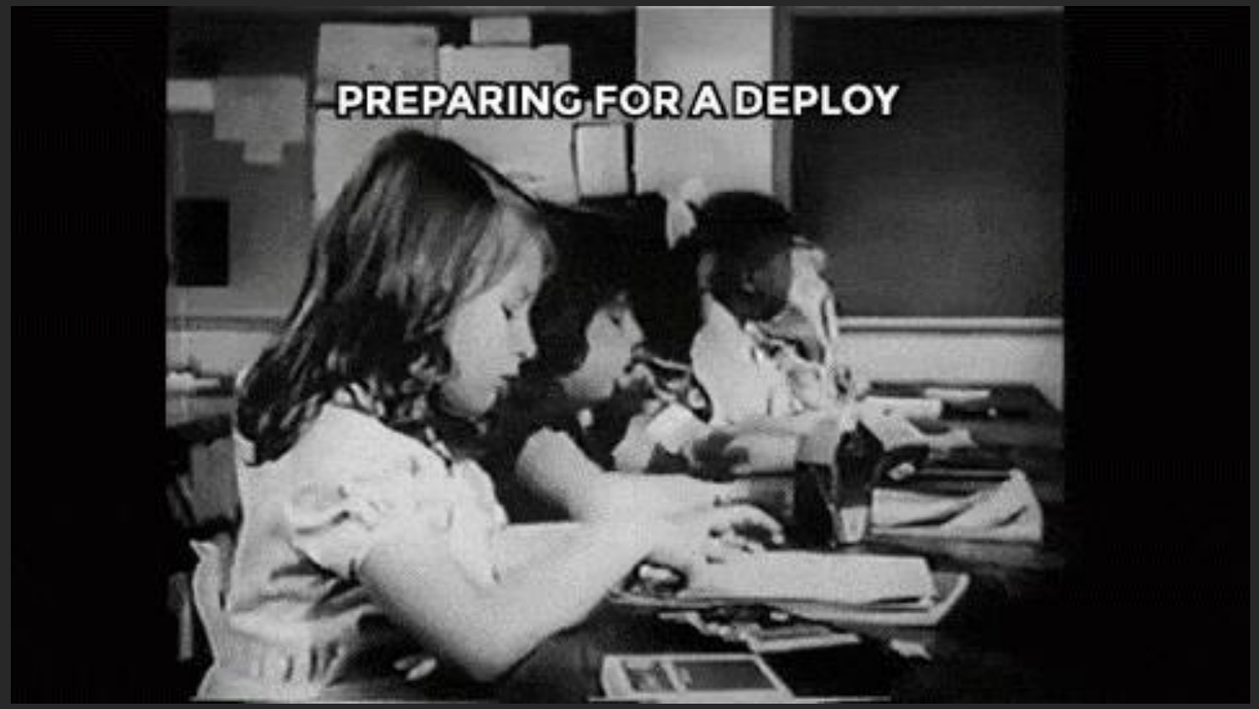
Agenda (interactive discussion throughout)

- Monolithic -> Microservices primer
- .NET Framework -> .NET Core
- Demo App -> Journey of an MVC app on-premises to a full serverless app
- DevOps -> Primer tools, processes
- Various paths for legacy .NET-based applications

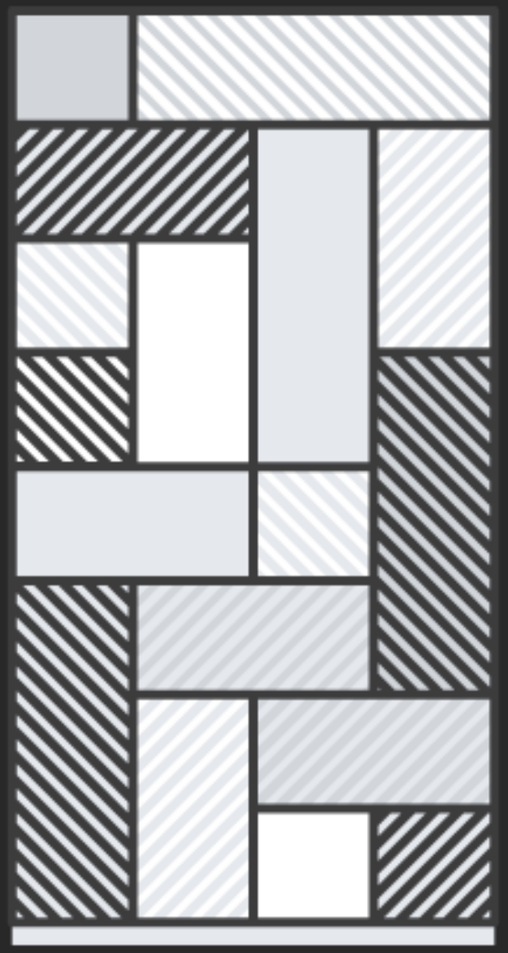
Metathesiophobia

The fear of change, it is a persistent, unrealistic, intense anxiety about and fear of new or different situations

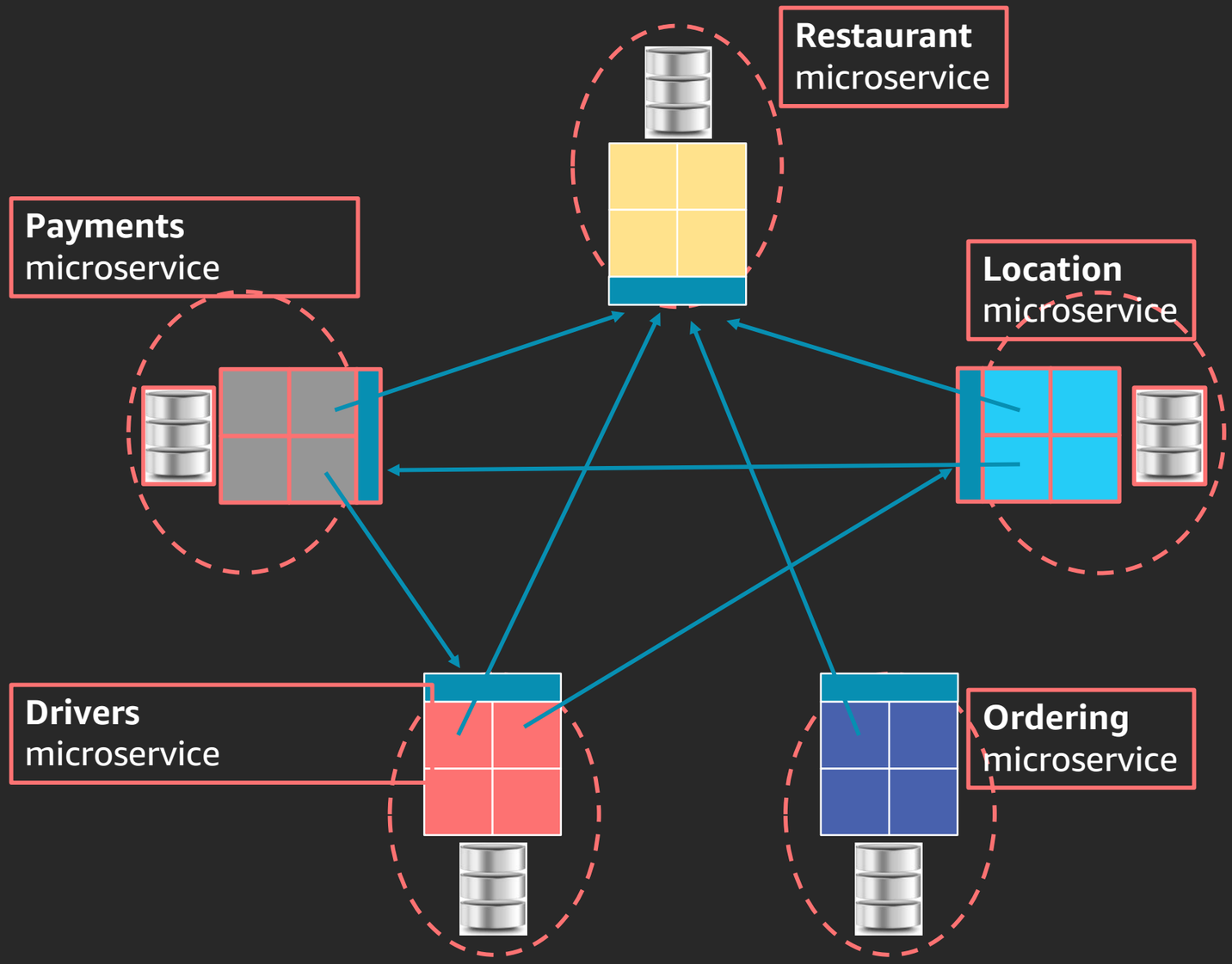
Change is the root cause of most outages



“Preparing for a deploy,” by Olli https://t.co/GwM6QMLQLn, DevOps Reactions (@devopsreactions)



Ecosystem of microservices



.NET Core is the future of .NET

Summary

New applications should be built on .NET Core. .NET Core is where future investments in .NET will happen. Existing applications are safe to remain on .NET Framework which will be supported. Existing applications that want to take advantage of the new features in .NET should consider moving to .NET Core. As we plan into the future, we will be bringing in even more capabilities to the platform. You can read about our plans here.

Scott Hunter

Director Program Management, .NET

Follow Scott

https://devblogs.microsoft.com/dotnet/net-core-is-the-future-of-net/

.NET Core is the future of .NET

AWS Open Source Blog

AWS Joins the .NET Foundation

by Norm Johanson | on 23 SEP 2019 | in .NET, Open Source | Permalink | Comments | Share

Fred Wurden, General Manager of AWS Windows and Enterprise
and Norm Johanson, Senior Software Dev Engineer, AWS SDKs and Tools

We're excited to announce today that AWS is joining the .NET Foundation as a corporate sponsor. AWS has a long-standing commitment to .NET, with a decade of experience running Microsoft Windows and .NET on AWS. Joining the .NET Foundation is a natural step for us to further invest and participate in this community.

https://aws.amazon.com/blogs/opensource/aws-joins-the-net-foundation/

ParticipationHours app: How I moved my MVC on-premises to a full serverless app

- “The Frankensteimation” (Frankenstein automation)
  - Amazon EC2 + Amazon EC2 Auto Scaling + VS with Web Deploy + Amazon S3 + AWS Systems Manager
- AWS Elastic Beanstalk + VS integration
- From .NET Standard 4.5 to .NET Core 2.0
  - Using AspNetCore.DataProtection.Aws.S3
  - Converting services/controllers/views into Razor + Ajax calls
  - AWS Lambda + Amazon API Gateway + custom authorizer
- Moving authentication (entity framework identity) to Amazon Cognito
  - Better integration with API Gateway
- Converting MS SQL server tables to Amazon DynamoDB (next release)

Demo: ParticipationHours app

Software moves faster today

DevOps – Teaming, architecture, CI/CD practices

Two-pizza team | **Delivery pipeline** | **Service**

Secure development environments in AWS

CI/CD tools


AWS CodeStar


AWS CodeBuild


AWS CodeCommit


AWS CodeDeploy


AWS CodePipeline

Infrastructure as code


AWS CloudFormation

AWS Cloud Development Kit

IDE


AWS Cloud9

Monitoring and tracing


AWS X-Ray


Amazon CloudWatch

Web apps


AWS Elastic Beanstalk

IDE and DevOps toolkits


Visual Studio Code


IntelliJ


PyCharm


Visual Studio


Eclipse


VSTS

CLI and scripting tools


AWS Command Line Interface


Tools for PowerShell

Languages

Amazon Corretto

Mobile


AWS Amplify

SDKs


JavaScript


Python


PHP


.NET


Ruby


Java


Go


Node.js


C++

DevSecOps architecture pillars

- Microservices architecture – Integrated via API
- Infrastructure as code – Integrated via API
- Security policy as code – Integrated via API
- Automated configuration management and control
- Continuous integration and continuous delivery
- Continuous lightweight/“rightweight” governance
- Intelligent automated logging and monitoring
- Automated event management and incident response

Possible phases of modernization

- Phase 1: Migrate existing workloads to the cloud
- Phase 2: Convert as many standard .NET applications to .NET Core using the .NET Portability Analyzer
- Phase 3: Containerize .NET applications and deploy to Windows or Linux containers; use ECS and/or EKS or AWS Fargate (.NET Core only currently)
- Phase 4: Offload pieces of your .NET applications to become serverless microservices (Strangler pattern)
- Phase 5: Take advantage of ML/AI technologies AWS has to offer, such as Amazon SageMaker, Amazon Transcribe, Amazon Translate, Amazon Polly, Amazon Lex, etc.

Not all phases need to be done in this exact order. Rearranging of order can and does occur. This list is not all encompassing either.

Strangler pattern for legacy app modernization

https://martinfowler.com/bliki/StranglerFigApplication.html

Resources

- Deploying an ASP.NET Core Application to AWS Fargate
- Deploy an ASP.NET Core Application to Windows Containers Using Amazon ECS
- Managing ASP.NET Session State with Amazon DynamoDB
- Identity Middleware for ASP.NET Core Web Applications Running Behind an Application Load Balancer (ALB) Configured With the OpenID Connect Authentication Feature

- Many more code samples/step-by-step guides/tools/webinars at AWS Developer Center for .NET : https://aws.amazon.com/developer/language/net/

Thank you!

Artur Rodrigues
artrodri@amazon.com

Amit Jha
amitjh@amazon.com

AWS re:Invent

aws

Please complete the session survey in the mobile app.