Serverless Workflows

For the Enterprise

Forrest Brazeal, Chris Munns

11/14/18

Forrest Brazeal

Sr Cloud Architect at Trek10
AWS Serverless Hero
Writer, Cartoonist at A Cloud Guru

Chris Munns

Principal Developer Advocate
AWS – Serverless
munns@amazon.com - @chrismunns

Enterprises + Serverless: Common Pain Points

- How do multiple developers/teams collaborate on serverless applications?
- How to enforce safety/security/auditability during production changes?
- How should I logically manage the resources that make up my serverless app?
- How to ”build serverless apps serverlessly”?

aws

What We'll Learn

- ✓ Managing serverless apps with SAM
- ✓ How to lay out a multi-account deployment pipeline using AWS developer tools
- ✓ When to test locally vs. in the cloud
- ✓ Dynamic feature branch pipelines for testing in the cloud
- ✓ Secrets management & cross-account security
- ✓ Rolling out gradual code changes with safe Lambda deployments

aws

Deployment Framework: SAM

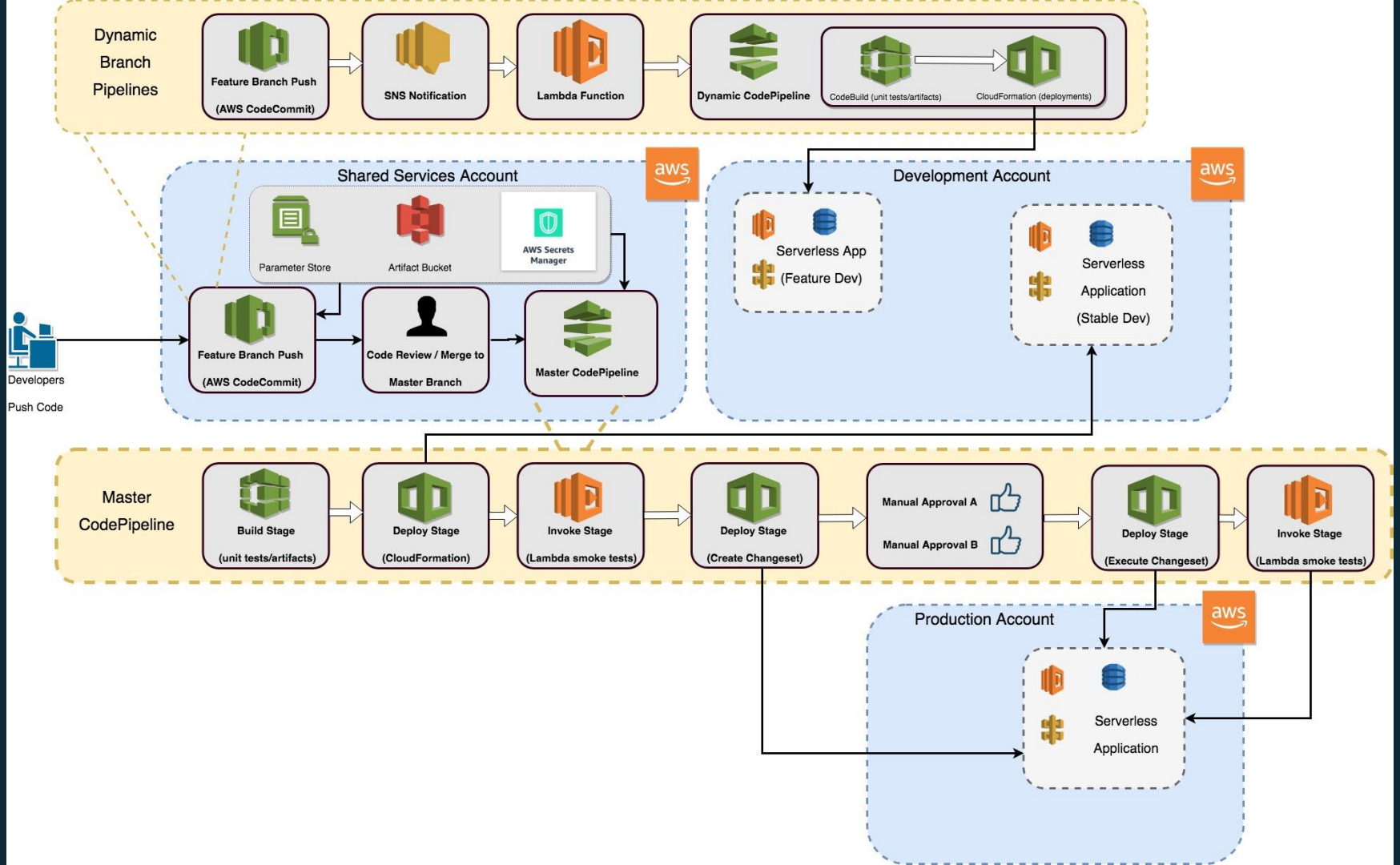
AWS Serverless Application Model

Extension of CloudFormation that makes it easier to define and deploy common “serverless” resources (Lambda, API Gateway, DynamoDB)

SAM CLI (previously SAM Local) supports some local testing

“Applications” view in Lambda lets you see all components

Lots of magic around code rollouts (stay tuned)


Dynamic Branch Pipelines
Feature Branch Push (AWS CodeCommit)
SNS Notification
Lambda Function
Dynamic CodePipeline
CodeBuild (unit tests/artifacts)
CloudFormation (deployments)
Shared Services Account
aws
Parameter Store
Artifact Bucket
AWS Secrets Manager
Development Account
Serverless App (Feature Dev)
Serverless Application (Stable Dev)
Developers Push Code
Feature Branch Push (AWS CodeCommit)
Code Review / Merge to Master Branch
Master CodePipeline
Master CodePipeline
Build Stage (unit tests/artifacts)
Deploy Stage (CloudFormation)
Invoke Stage (Lambda smoke tests)
Deploy Stage (Create Changeset)
Manual Approval A
Manual Approval B
Deploy Stage (Execute Changeset)
Invoke Stage (Lambda smoke tests)
Production Account
Serverless Application

AWS Developer Tools Used In This Webinar

AWS CodeCommit

AWS-hosted Git repository

AWS CodeBuild

Run build jobs in ephemeral Docker containers

AWS CodeDeploy

Manage code rollouts (traffic shifting, blue/green, etc)

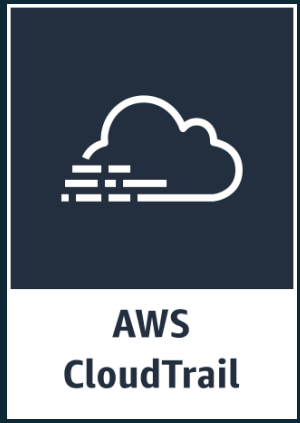
AWS CodePipeline

Orchestrate build, test, and deployment phases of your app lifecycle

Why AWS Developer Tools?

Manage your pipeline infrastructure the same way you manage your applications:

- Integrated security model via AWS IAM, AWS CloudTrail, AWS Config, etc
- The boilerplate of working with the underlying services is abstracted away (ex: CloudFormation waits)
- Fully “serverless” model – no operational overhead for undifferentiated work



Best Practices For Testing

- ✓ Test function code locally (unit tests + mocks)
- ✓ Test interactions between services in the cloud
- ✓ Check out SAM CLI (previously SAM Local) - Docker container to emulate your functions
- ✓ Rule of thumb: the more cloud services you use, the less value you'll get from local testing

aws

Dynamic Feature Pipelines

Use case: multiple developers testing different features

Problem: each person needs their own test stack

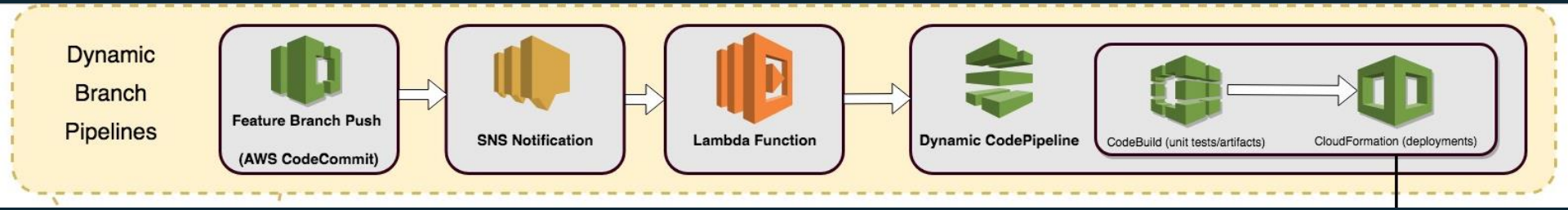
Solution:

1. Dynamically create stacks when feature branch is pushed to repository
2. Tear down when branch is deleted

Cost effective for serverless!

aws

Dynamic Feature Pipelines



Best Practices: Secrets Management

No plaintext secrets in Lambda environment variables!!!

Options for secure secret storage:

- SSM Parameter Store + KMS
- AWS Secrets Manager

Prefer not to put KMS-encrypted secrets in env vars

- Better to have centralized location

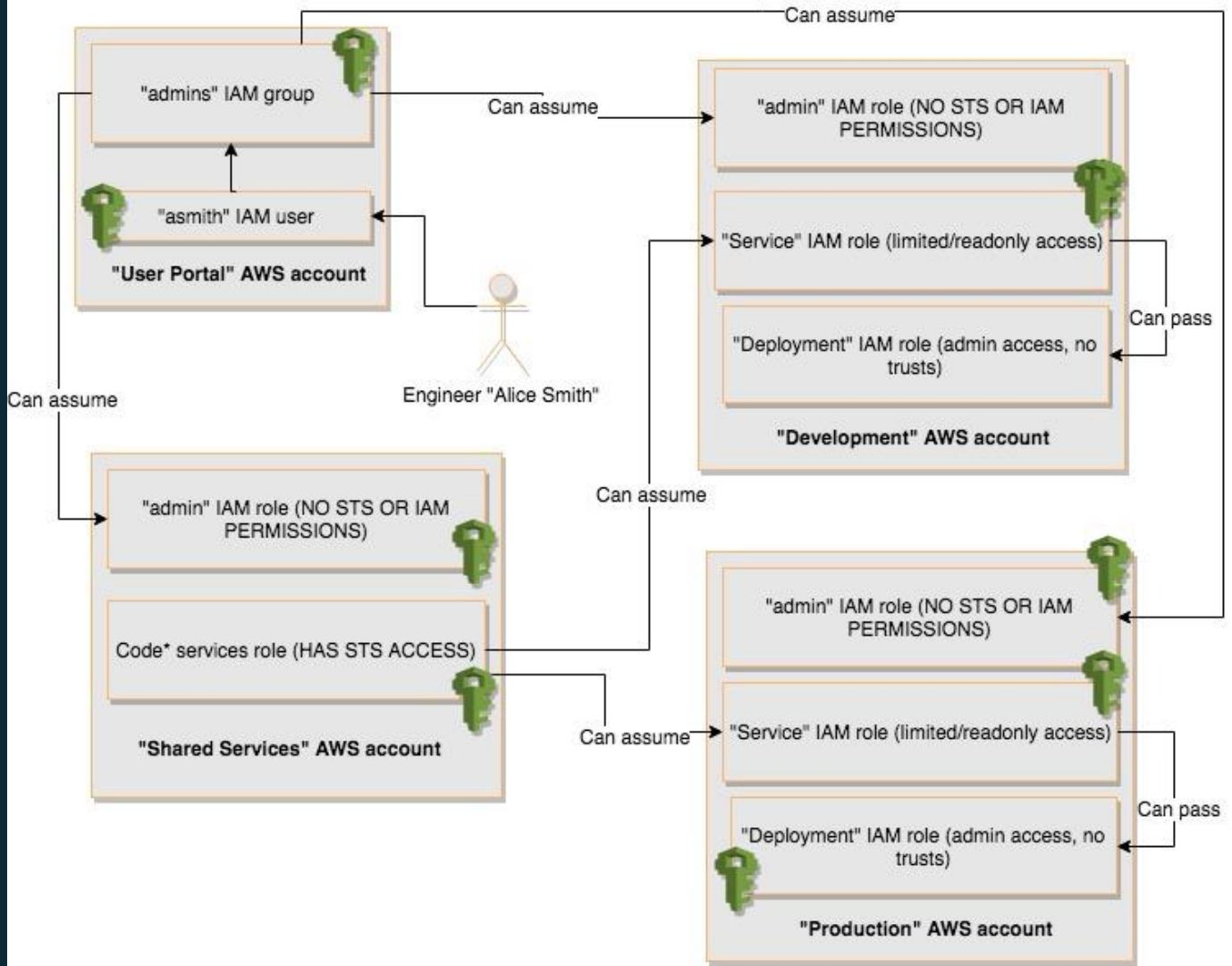
Need some type of caching solution

Parameter Store

Best Practices: Multi-Account Security

- ✓ Production deployments should be done by pipelines, not people
- ✓ Separate the ability to deploy from the ability to approve
- ✓ Manage changes to IAM entities via pipeline as well
- ✓ Pass deployment roles to CloudFormation
- ✓ KMS-encrypt artifacts (CodePipeline makes this easy)

aws


Can assume
"admins" IAM group
Can assume
"admin" IAM role (NO STS OR IAM PERMISSIONS)
"asmith" IAM user
"Service" IAM role (limited/readonly access)
"User Portal" AWS account
Can pass
"Deployment" IAM role (admin access, no trusts)
Engineer "Alice Smith"
"Development" AWS account
Can assume
Can assume
"admin" IAM role (NO STS OR IAM PERMISSIONS)
"admin" IAM role (NO STS OR IAM PERMISSIONS)
Code* services role (HAS STS ACCESS)
"Service" IAM role (limited/readonly access)
Can assume
"Shared Services" AWS account
Can pass
"Deployment" IAM role (admin access, no trusts)
"Production" AWS account

Multiple Approvers (The Nuclear Option)

CodePipeline lets you lock down IAM permissions to specific named stages

Can force multiple IAM entities from separate groups to approve a change

Caveat: requires careful user management

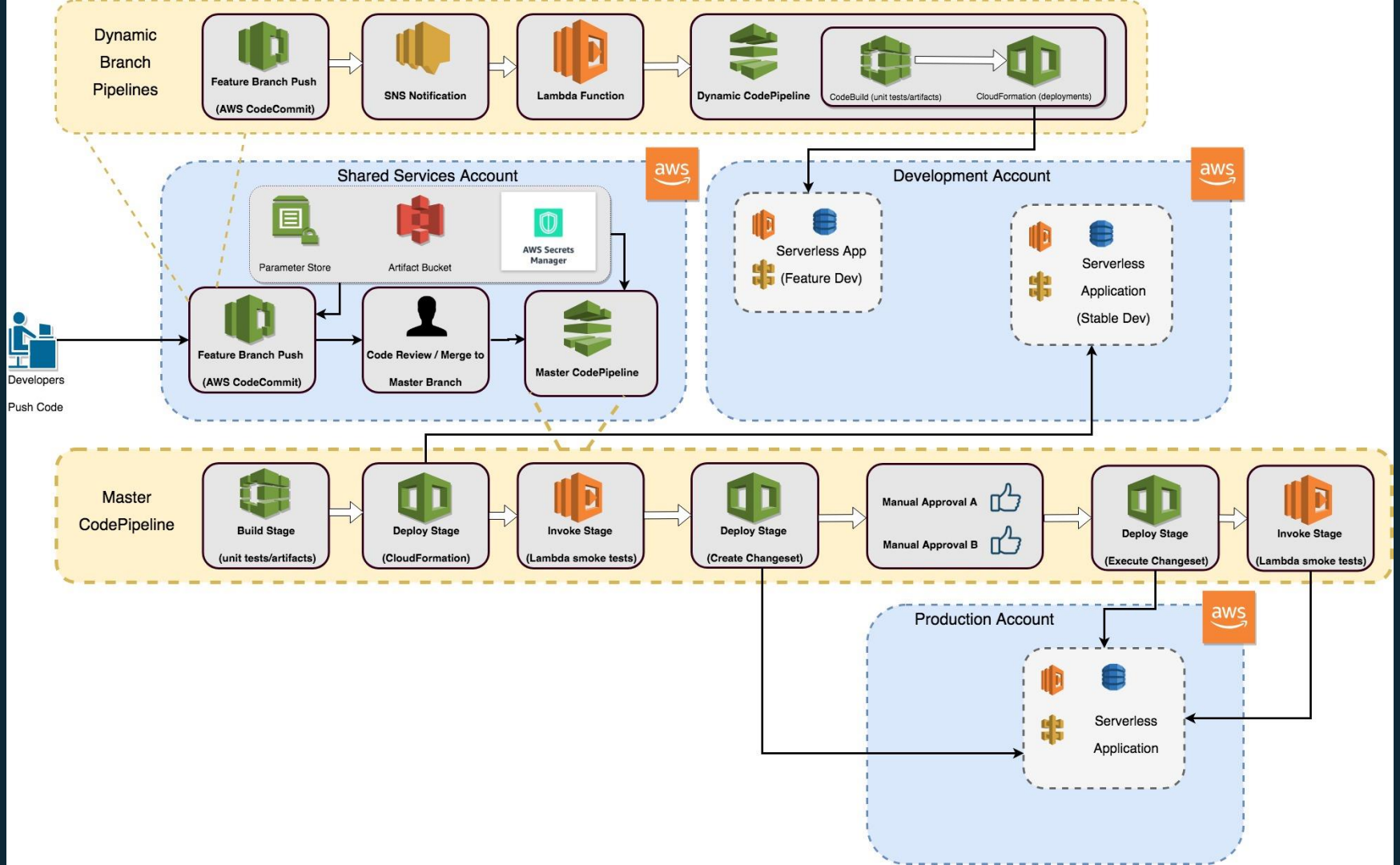
Blog post: https://www.trek10.com/blog/enforcing-two-person-rule-aws-codepipeline/

aws

Best Practices: Code Rollouts

- ✓ Use separate CloudFormation stacks (and separate AWS accounts if possible) per stage (dev, QA, prod)
- ✓ Use gradual deployments via SAM/CodeDeploy for safe Lambda rollouts if you can (with automated hooks to validate functionality)
- ✓ May need separate stacks with DNS failover if you have resources that don’t support blue/green updates (ex: AppSync)
- ✓ Use different rollout policies per environment

aws


Dynamic Branch Pipelines
Feature Branch Push (AWS CodeCommit)
SNS Notification
Lambda Function
Dynamic CodePipeline
CodeBuild (unit tests/artifacts)
CloudFormation (deployments)
Shared Services Account
aws
Parameter Store
Artifact Bucket
AWS Secrets Manager
Development Account
Serverless App (Feature Dev)
Serverless Application (Stable Dev)
Developers Push Code
Feature Branch Push (AWS CodeCommit)
Code Review / Merge to Master Branch
Master CodePipeline
Master CodePipeline
Build Stage (unit tests/artifacts)
Deploy Stage (CloudFormation)
Invoke Stage (Lambda smoke tests)
Deploy Stage (Create Changeset)
Manual Approval A
Manual Approval B
Deploy Stage (Execute Changeset)
Invoke Stage (Lambda smoke tests)
Production Account
Serverless Application

Questions?

Twitter - @forrestbrazeal

Email – fbrazeal@trek10.com

AWS Developer Center – developer.aws