AWS re:Invent

SVS341-R

An in-depth tour of AWS SAM

Alex Wood
Senior Software Development Engineer
Amazon Web Services

Agenda

Becoming a “sam init” power user

Working with AWS Identity and Access Management (IAM) policies

Effective testing of AWS Serverless Application Model (AWS SAM) applications

Deployment best practices

AWS SAM

- Framework for building serverless applications
- Shorthand syntax to express functions, APIs, databases, and event source mappings
- Model with YAML, deploy using AWS CloudFormation
- Open source!
- https://github.com/awslabs/serverless-application-model

AWS SAM CLI

- Create, build, test, and deploy AWS SAM applications
- Step-through debugging and IDE support
- Open source!
- https://github.com/awslabs/aws-sam-cli

Becoming a “sam init” power user

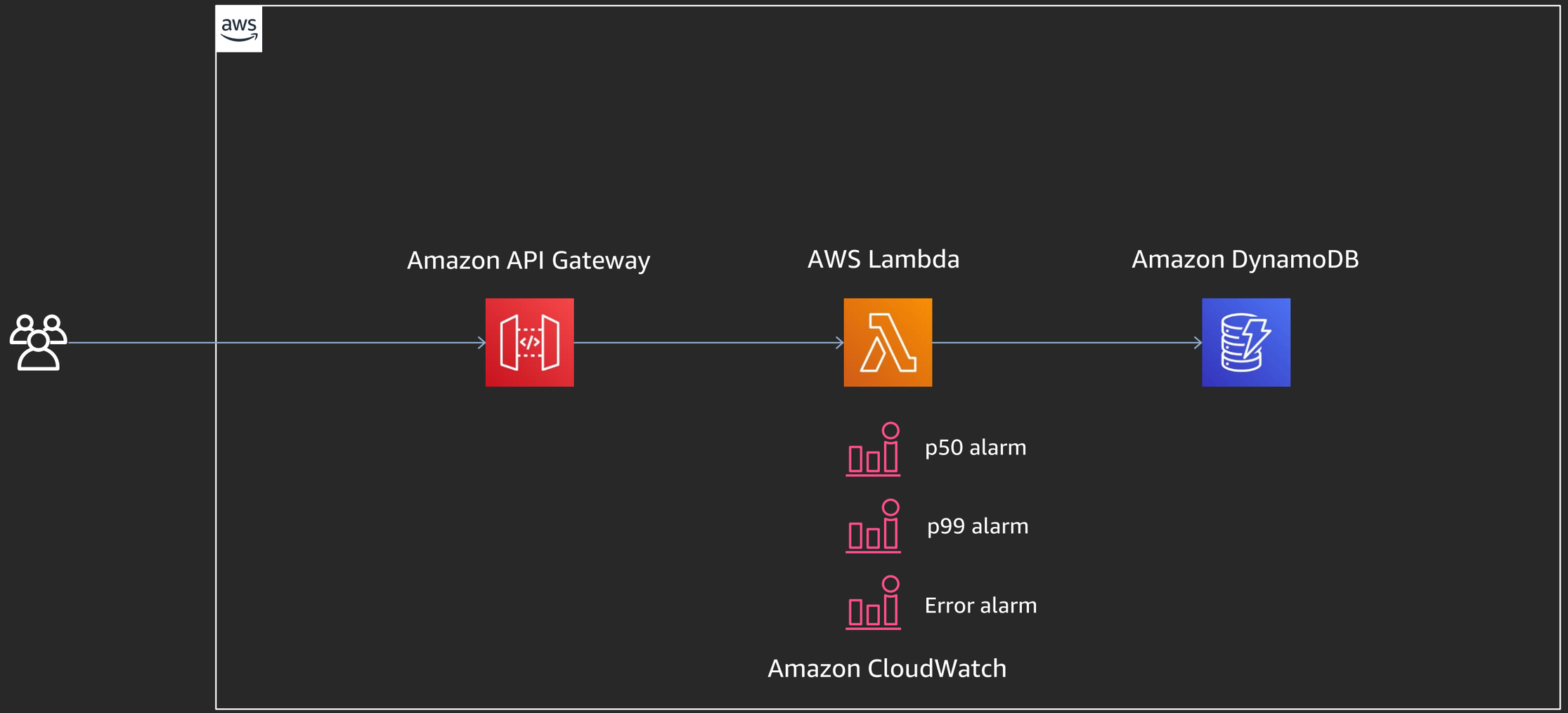
The challenge

- sam init templates bundled with the AWS SAM CLI are examples
- Over time, you develop organizational serverless best practices
- How do you bake in serverless best practices from the start?

AWS SAM CLI supports custom init sources

```
sam init --location https://github.com/awood45/aws-sam-sinatra-template
```

App template structure



Example: Built-in alarms

```
SinatraErrorAlarm:
     Type: AWS::CloudWatch::Alarm
     Properties:
       ComparisonOperator:
GreaterThanOrEqualToThreshold
       EvaluationPeriods: 3
       Threshold: 1
       Dimensions:
         - Name: FunctionName
           Value: !Ref SinatraFunction
       Statistic: Sum
       TreatMissingData: missing
       Namespace: AWS/Lambda
       Period: 60
       MetricName: Errors
```

Example: Hide marshalling work

```
APP = Rack::Builder.parse_file('config.ru').first

def handler(event:,context:)
  $baseHost ||= event.fetch(
    'headers’,
    {}
  ).fetch('Host', nil)
  body =
    if event['isBase64Encoded']
      Base64.decode64(event['body'])
    else
      event['body']
    end
# etc...
```

```
require 'sinatra'
require_relative 'app_table'

get "/" do
  # your app logic here
end
```

AWS SAM CLI Commands

```
sam init --location https://github.com/awood45/aws-sam-sinatra-template

sam build

sam deploy --guided
```

Demo

Working with IAM policies

The challenge

- Following the principle of least privilege is important
- Most AWS SAM templates have permissions to create IAM roles
  - We want to limit what those created roles can do, systemically
- Good intentions are not enough

2 ways to write IAM policies in AWS SAM

```
ServerlessFunctionCannedPolicy:
  Type: AWS::Serverless::Function
  Properties:
    Handler: lambda.handler
    Runtime: ruby2.5
    Policies:
      - DynamoDBCrudPolicy:
          TableName: !Ref AppTable
```

```
ServerlessFunctionCustomPolicy:
  Type: AWS::Serverless::Function
  Properties:
    Handler: lambda.handler
    Runtime: ruby2.5
    Policies:
      - Version: ”2012-10-17”
        Statement:
          - Effect: Allow
            Action:
              - dynamodb:GetItem
              - dynamodb:PutItem
              - dynamodb:UpdateItem
        Resource:
          - !Sub
arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::
AccountId}:table/${AppTable}
```

IAM permissions boundaries

- Apply to IAM entities (users or roles)
- Use a managed policy to set maximum permissions that can be granted to an IAM entity
- That entity can only perform actions allowed by *both* its identity-based policies and the permissions boundary
- The AWS::Serverless::Function resource allows you to pass a permission boundary policy

Permissions boundary in action

```
TestPermissionsBoundary:
   Type: AWS::IAM::ManagedPolicy
   Properties:
     PolicyDocument:
       Version: 2012-10-17
       Statement:
         - Effect: Allow
           Action:
             - dynamodb:GetItem
             - cloudwatch:PutMetricData
             - logs:CreateLogGroup
             - logs:CreateLogStream
             - logs:PutLogEvents
           Resource: "*"
```

```
HelloWorldFunction:
   Type: AWS::Serverless::Function
   Properties:
     CodeUri: hello_world/
     Handler: app.lambda_handler
     Runtime: ruby2.5
     Policies:
       - DynamoDBCrudPolicy:
           TableName: !Ref AppTable
     PermissionsBoundary: !Ref
TestPermissionsBoundary
     Events:
       HelloWorld:
         Type: Api
         Properties:
           Path: /hello
           Method: get
```

Resulting permissions

IAM policy

dynamodb:GetItem

dynamodb:DeleteItem

dynamodb:PutItem

dynamodb:Scan

dynamodb:Query

logs:CreateLogGroup

logs:CreateLogStream

logs:PutLogEvents

etc.

Actual permissions

dynamodb:GetItem

logs:CreateLogGroup

logs:CreateLogStream

logs:PutLogEvents

IAM permissions boundary

dynamodb:GetItem

cloudwatch:PutMetricData

logs:CreateLogGroup

logs:CreateLogStream

logs:PutLogEvents

Permissions boundary from AWS CodePipeline

- Pattern: Create a permissions boundary policy in your CI/CD toolchain
- Configure your CloudFormation role to only be allowed to create AWS IAM roles that include this permissions boundary.
- Pass the permissions boundary managed policy ARN to your application template.

Demo

Permissions boundary in action

```
TestPermissionsBoundary:
   Type: AWS::IAM::ManagedPolicy
   Properties:
     PolicyDocument:
       Version: 2012-10-17
       Statement:
         - Effect: Allow
           Action:
             - dynamodb:GetItem
             - logs:CreateLogGroup
             - logs:CreateLogStream
             - logs:PutLogEvents
           Resource: "*"
```

```
def lambda_handler(event:, context:)
  # this uses GetItem
  item = AppTable.find(hkey: "foo")
  return {
    statusCode: 200,
    body: item.body
  }
end
```

Permissions boundary in action

```
TestPermissionsBoundary:
   Type: AWS::IAM::ManagedPolicy
   Properties:
     PolicyDocument:
       Version: 2012-10-17
       Statement:
         - Effect: Allow
           Action:
             - dynamodb:GetItem
             - logs:CreateLogGroup
             - logs:CreateLogStream
             - logs:PutLogEvents
           Resource: "*"
```

```
def lambda_handler(event:, context:)
  # this uses GetItem
  item = AppTable.find(hkey: "foo")
  return {
    statusCode: 200,
    body: item.body
  }
end
```

Permissions boundary in action

```
TestPermissionsBoundary:
   Type: AWS::IAM::ManagedPolicy
   Properties:
     PolicyDocument:
       Version: 2012-10-17
       Statement:
         - Effect: Allow
           Action:
             - dynamodb:GetItem
             - logs:CreateLogGroup
             - logs:CreateLogStream
             - logs:PutLogEvents
           Resource: "*"
```

```
def lambda_handler(event:, context:)
  # this uses Scan
  item = AppTable.scan.first
  return {
    statusCode: 200,
    body: item.body
  }
end
```

Permissions boundary in action

```
TestPermissionsBoundary:
   Type: AWS::IAM::ManagedPolicy
   Properties:
     PolicyDocument:
       Version: 2012-10-17
       Statement:
         - Effect: Allow
           Action:
             - dynamodb:GetItem
             - logs:CreateLogGroup
             - logs:CreateLogStream
             - logs:PutLogEvents
           Resource: "*"
```

```
def lambda_handler(event:, context:)
  # this uses Scan
  item = AppTable.scan.first
  return {
    statusCode: 200,
    body: item.body
  }
end
```

Effective testing of AWS SAM applications

AWS re:Invent

aws

The challenge

- Local testing is useful for development but has inherent limitations
- Understanding how to fit remote testing into CI/CD

AWS SAM Local

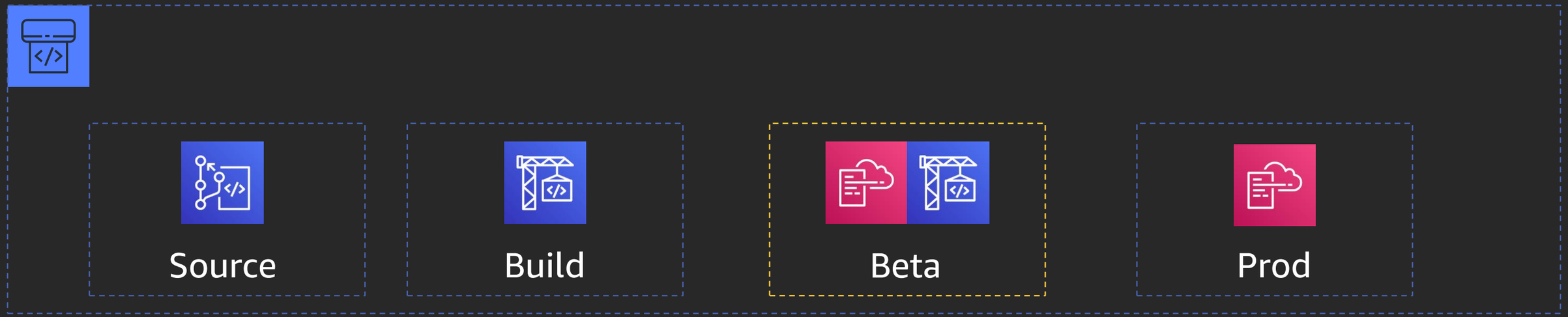
- Uses Docker to simulate execution on AWS Lambda
- Several supported modes:
  - sam local start-api (Create an endpoint that simulates your API Gateway endpoint)
  - sam local start-lambda (Create an endpoint that simulates the Lambda API)
  - sam local invoke (Single invocation)
- Useful for quick development cycles/iterations
- With IDEs, can do step-through debugging

Demo

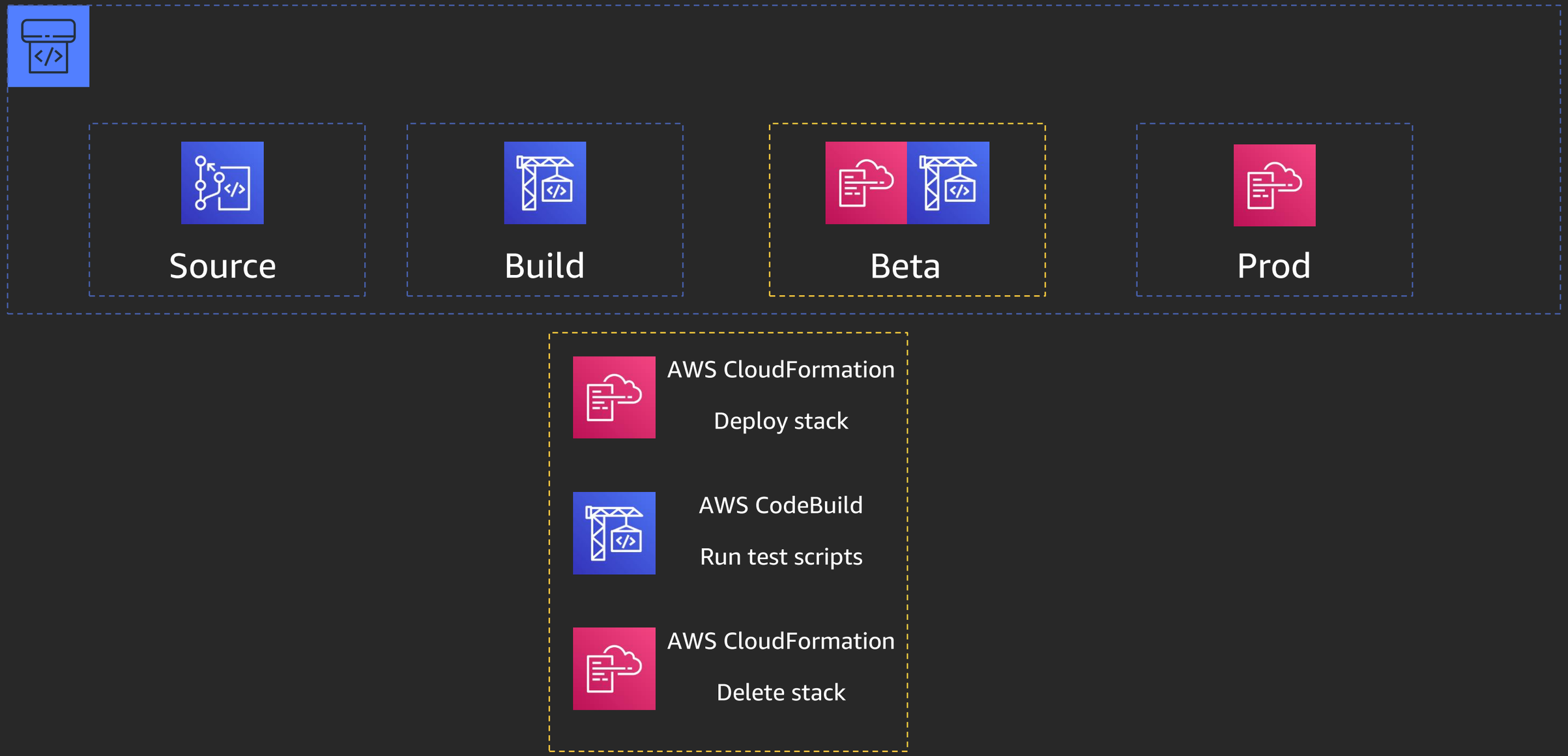
Live integration testing

- Serverless + AWS CloudFormation = Easy to test on prod-like environment
- Example AWS CodePipeline stage:
  - AWS CloudFormation action: Create change set (beta stack)
  - AWS CloudFormation action: Execute change set (beta stack)
  - AWS CodeBuild: Run integration test scripts against beta environment
  - AWS CloudFormation action: Delete stack (beta stack)

Live integration testing



Live integration testing



Demo

Testing summarized

- AWS SAM Local testing is helpful for experimental testing on a developer machine
- AWS CloudFormation + AWS CodePipeline = 😎
- All of this provides a feedback loop as you develop, and as you ship changes to your application

Deployment best practices

The challenge

- How do we deploy safely?
- How do we detect problems and rollback with minimal impact?

AWS SAM deployment options

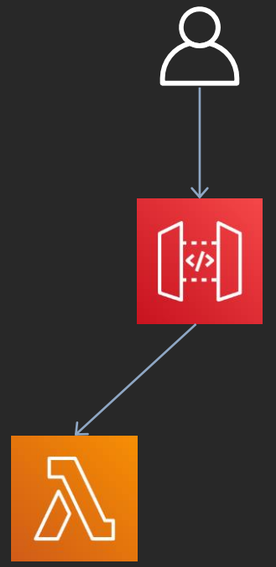
- Instant traffic shifting with Lambda aliases
- Pre-traffic and post-traffic hooks
- Traffic shifting using AWS CodeDeploy

Pre-/Post-traffic hooks

```
FunctionName: 'CodeDeployHook_preTrafficHook’
DeploymentPreference:
  Enabled: false
Policies:
  - Version: "2012-10-17”
    Statement:
      - Effect: "Allow”
        Action:
          - "codedeploy:PutLifecycleEventHookExecutionStatus”
        Resource: "*”
  - Version: "2012-10-17”
    Statement:
      - Effect: "Allow”
        Action:
          - "lambda:InvokeFunction”
        Resource: !GetAtt MyLambdaFunction.Arn
```

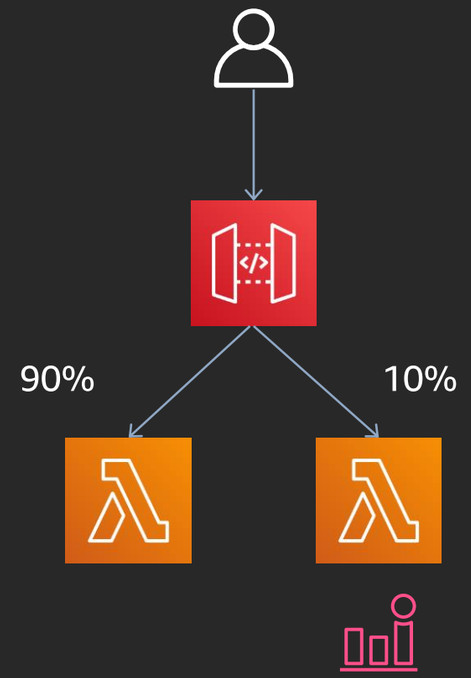
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



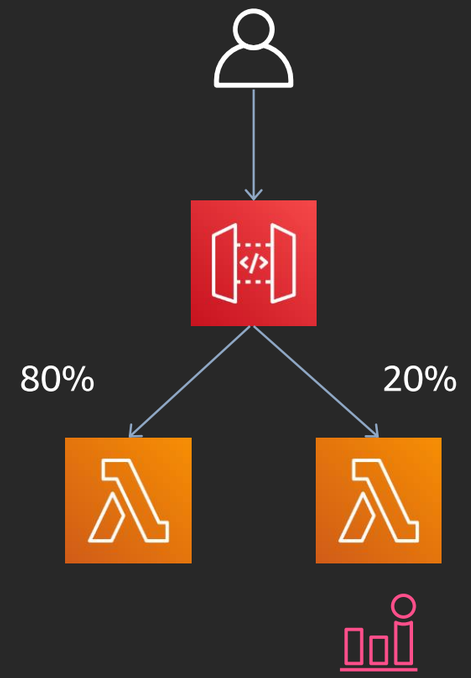
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



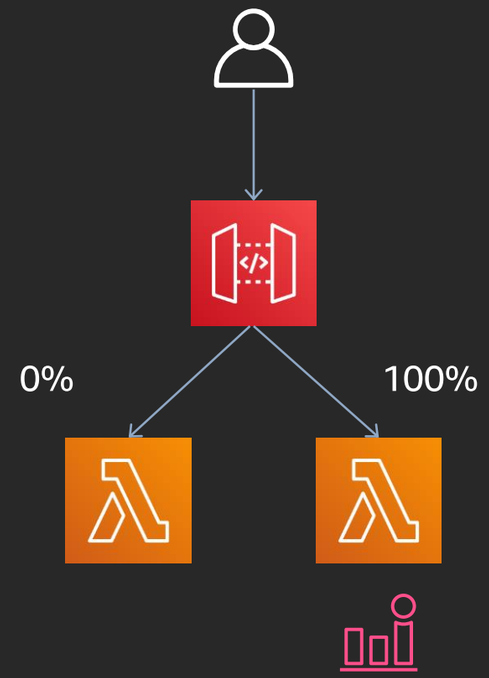
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



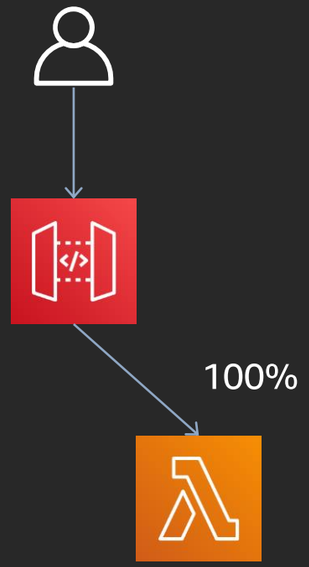
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



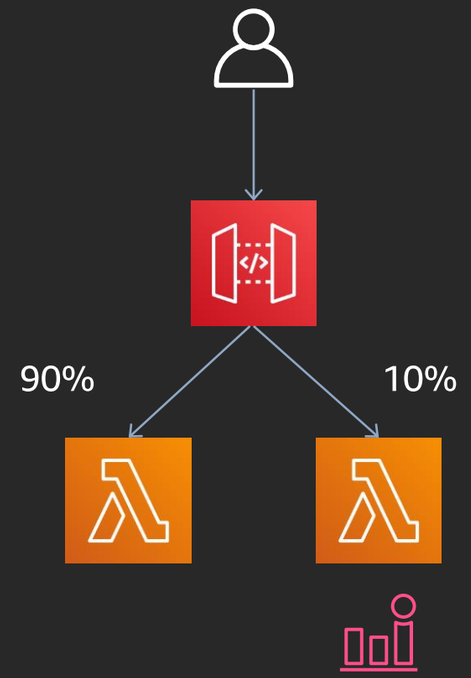
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



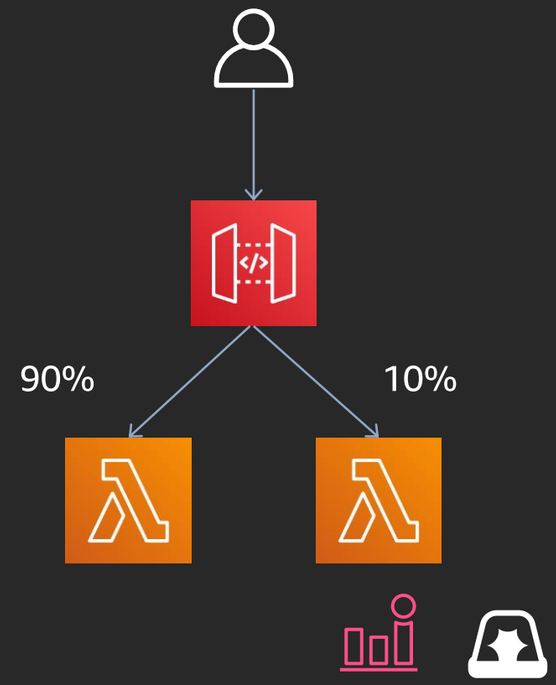
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



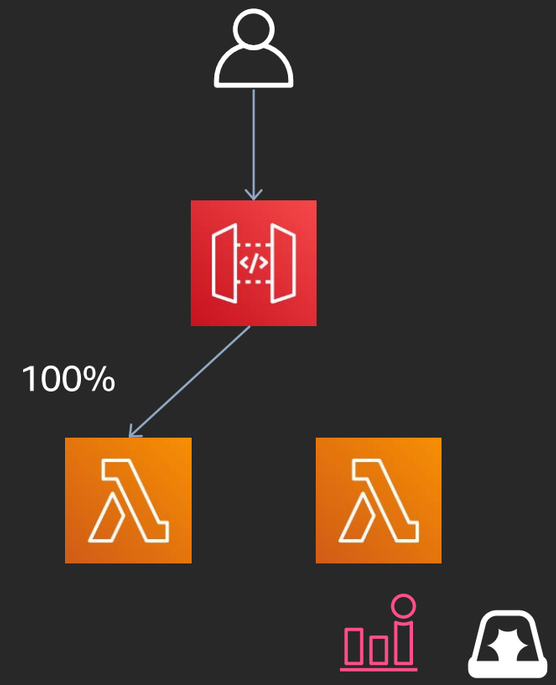
Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



Traffic shifting

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs10.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        - !Ref AliasErrorMetricAlarm
        - !Ref LatestVersionErrorAlarm
```



Demo

Conclusion

Summary

- Use AWS SAM CLI's init function to ingrain your best practices
- IAM provides a number of ways to secure your applications
- Local and remote testing
- AWS SAM provides safe deployment helpers

Related breakouts

SVS308: Moving to event-driven architectures

SVS336: CI/CD for serverless applications

SVS401: Optimizing your serverless applications

SVS402: Building APIs from front to back

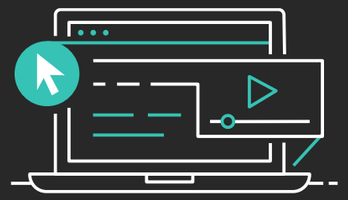
SVS405: A serverless journey: AWS Lambda under the hood

SVS406: Asynchronous-processing best practices with AWS Lambda

SVS407: Architecting and operating resilient serverless systems at scale

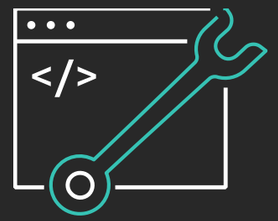
Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development



Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures



Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

aws training and certification

Thank you!

Alex Wood

Twitter: @alexwwood

Please complete the session survey in the mobile app.