
AWS
re:Invent

SVS315

Best practices for growing a serverless application

Benjamin Smith

Senior Developer Advocate, Serverless

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Serverless applications – Day 1

Challenge 1: The “Lambda-lith”

Challenge 2: The mono-repo

Challenge 3: Enforcing security best practices

Challenge 4: Managing multiple AWS accounts



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

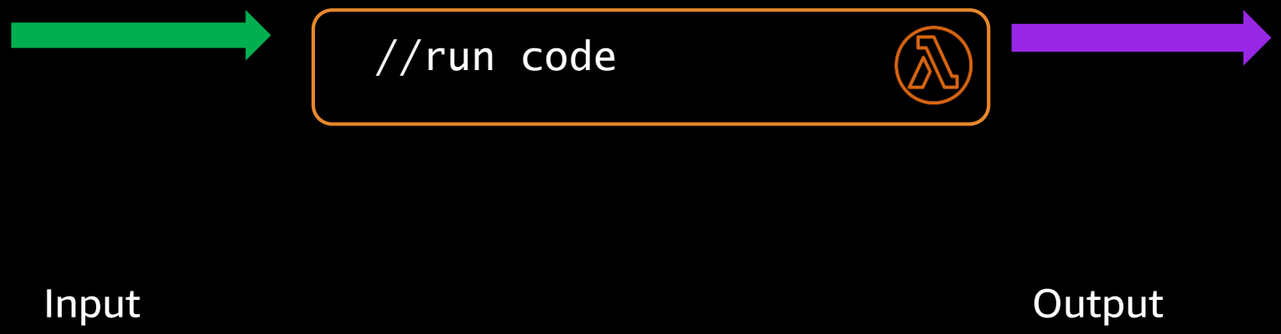
Your serverless application – Day 1



- Moves quickly from prototype to production
- Scales automatically
- Managed infrastructure

Your serverless application – Day 100

GROWTH CHALLENGE 1 – MONOLITHIC AWS LAMBDA FUNCTIONS

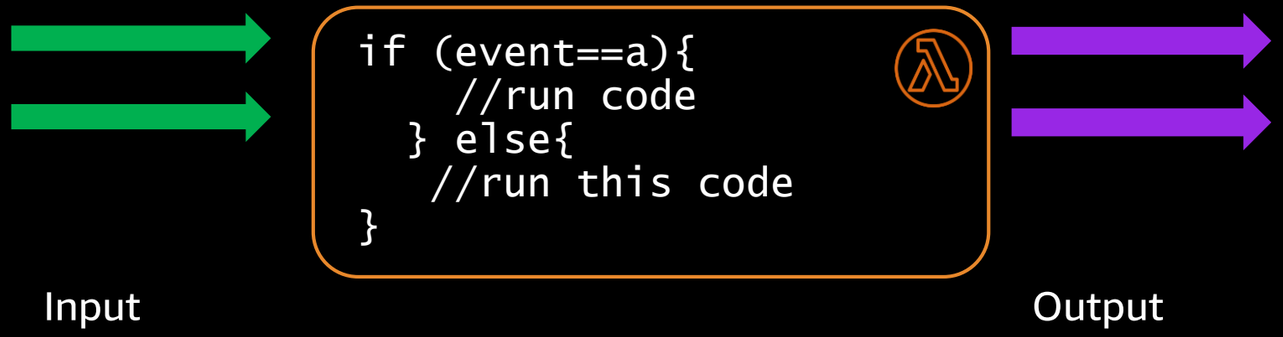


Application grows more complex over time, performing multiple tasks to handle increasingly complex business logic, leading to

- Tightly coupled code base
- Slower release cadence
- Poor discoverability
- Additional complexity

Your serverless application – Day 100

GROWTH CHALLENGE 1 – MONOLITHIC AWS LAMBDA FUNCTIONS

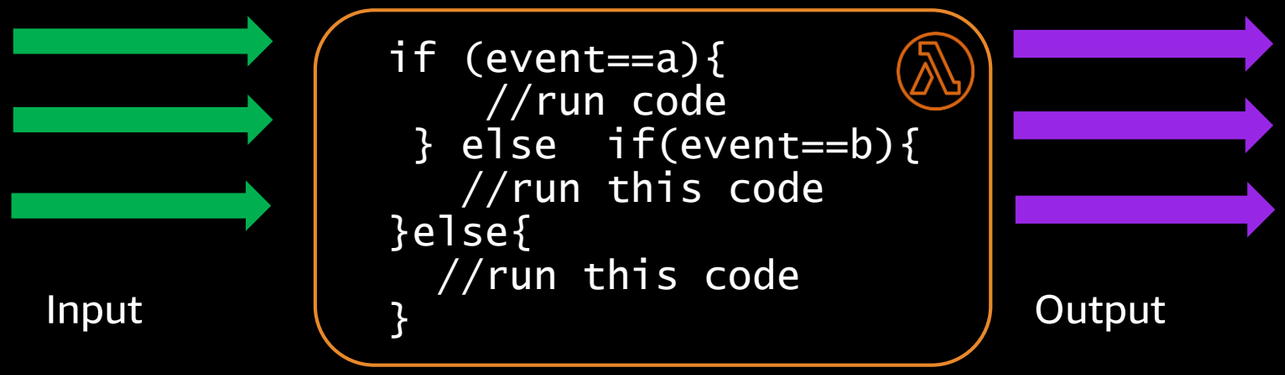


Application grows more complex over time, performing multiple tasks to handle increasingly complex business logic, leading to

- Tightly coupled code base
- Slower release cadence
- Poor discoverability
- Additional complexity

Your serverless application – Day 100

GROWTH CHALLENGE 1 – MONOLITHIC AWS LAMBDA FUNCTIONS

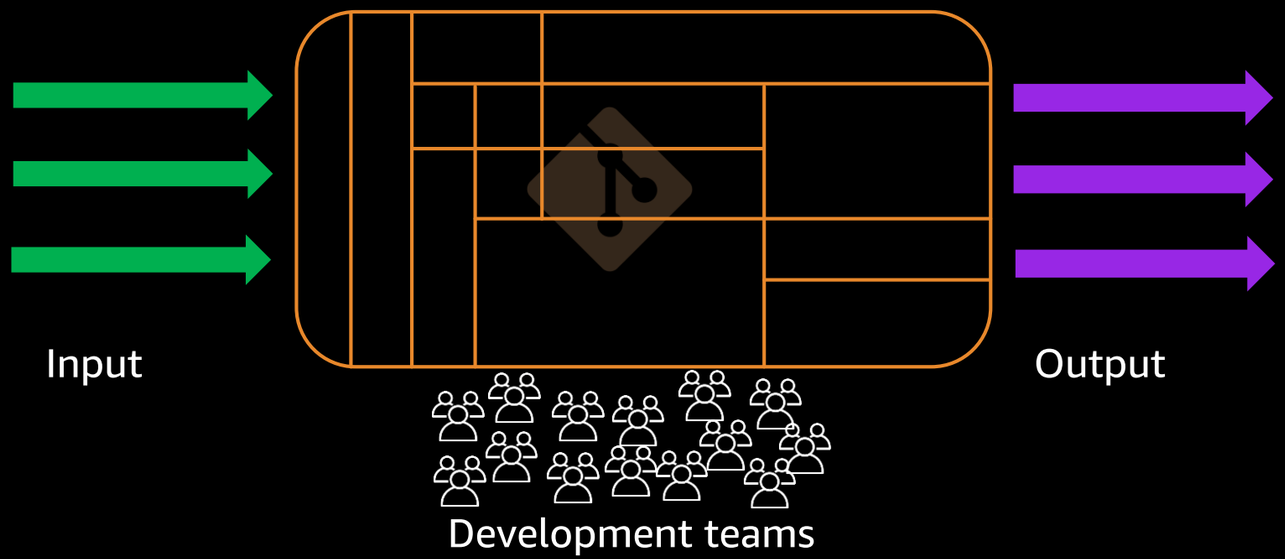


Application grows more complex over time, performing multiple tasks to handle increasingly complex business logic, leading to

- Tightly coupled code base
- Slower release cadence
- Poor discoverability
- Additional complexity

Your serverless application – Day 100

GROWTH CHALLENGE 2 – MONO-REPOS

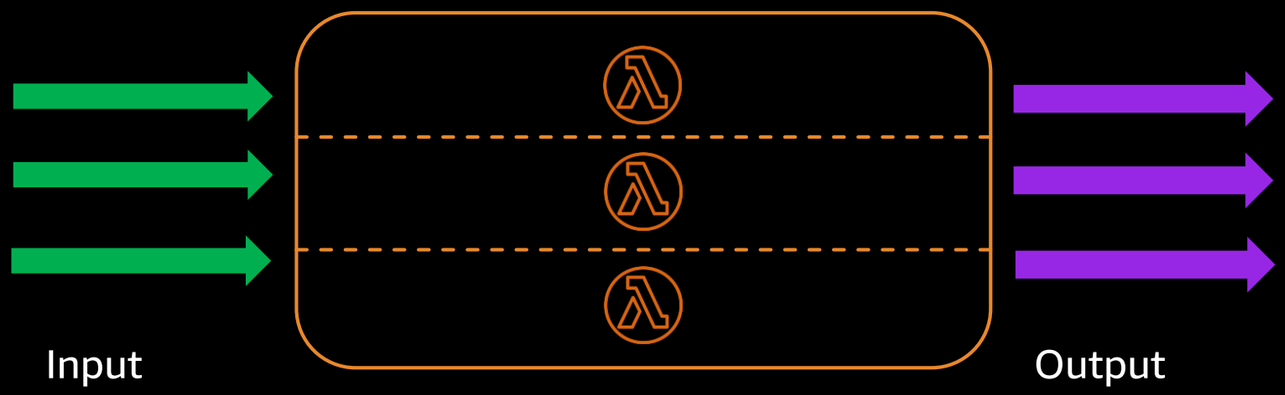


A single repository containing the entire application logic

- Difficult for new developers to onboard
- Difficult to separate responsibility of ownership
- Requires a single language
- Difficult to maintain

Breaking apart a “Lambda-lith”

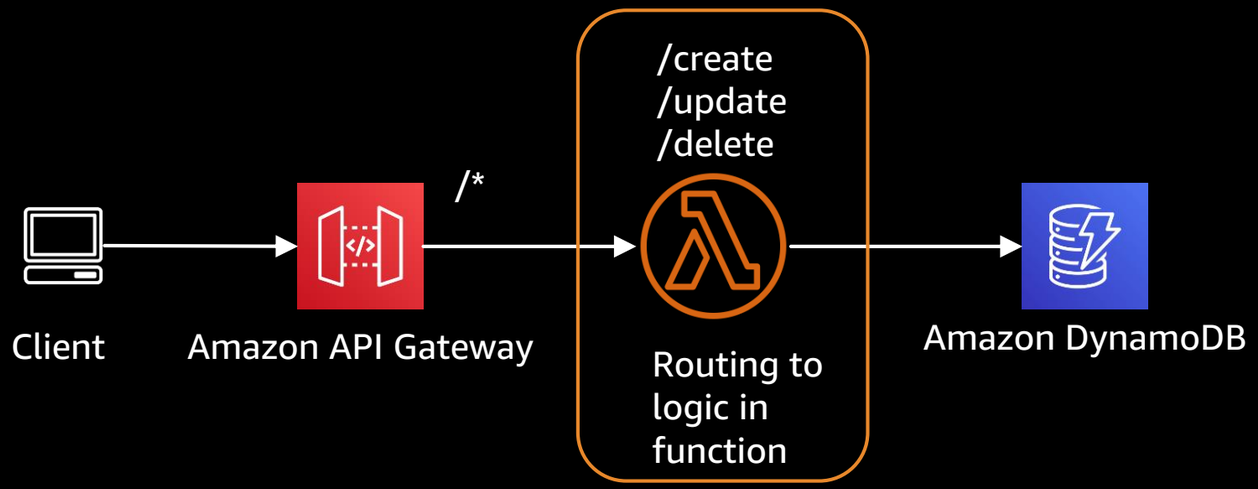
SPLIT APPLICATIONS INTO MULTIPLE FUNCTIONS OR SERVICES



- Split Lambda functions based on finite functionality and scope
- Reduce complexity where possible

Breaking apart a “Lambda-lith”

A REAL-WORLD EXAMPLE

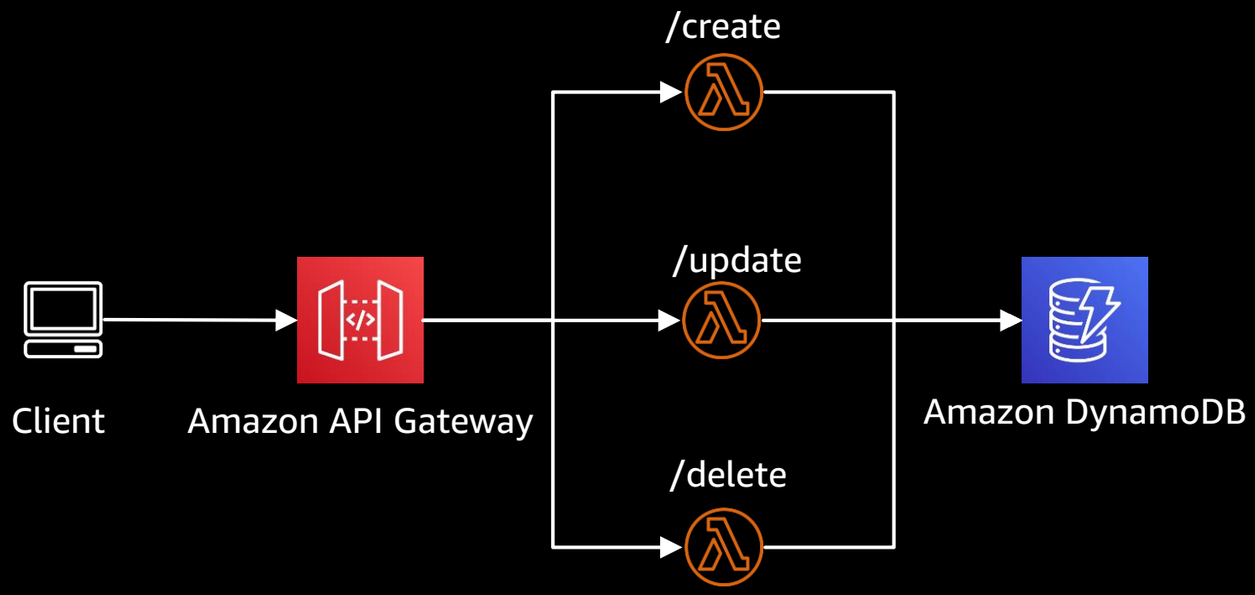


In this example, Amazon API Gateway acts as a proxy by routing all requests to a single Lambda function that runs the appropriate code based on its route configuration

- Security permissions applied to the whole
- Performance setting applied to the whole
- Duration and space limits apply to whole

Breaking apart a “Lambda-lith”

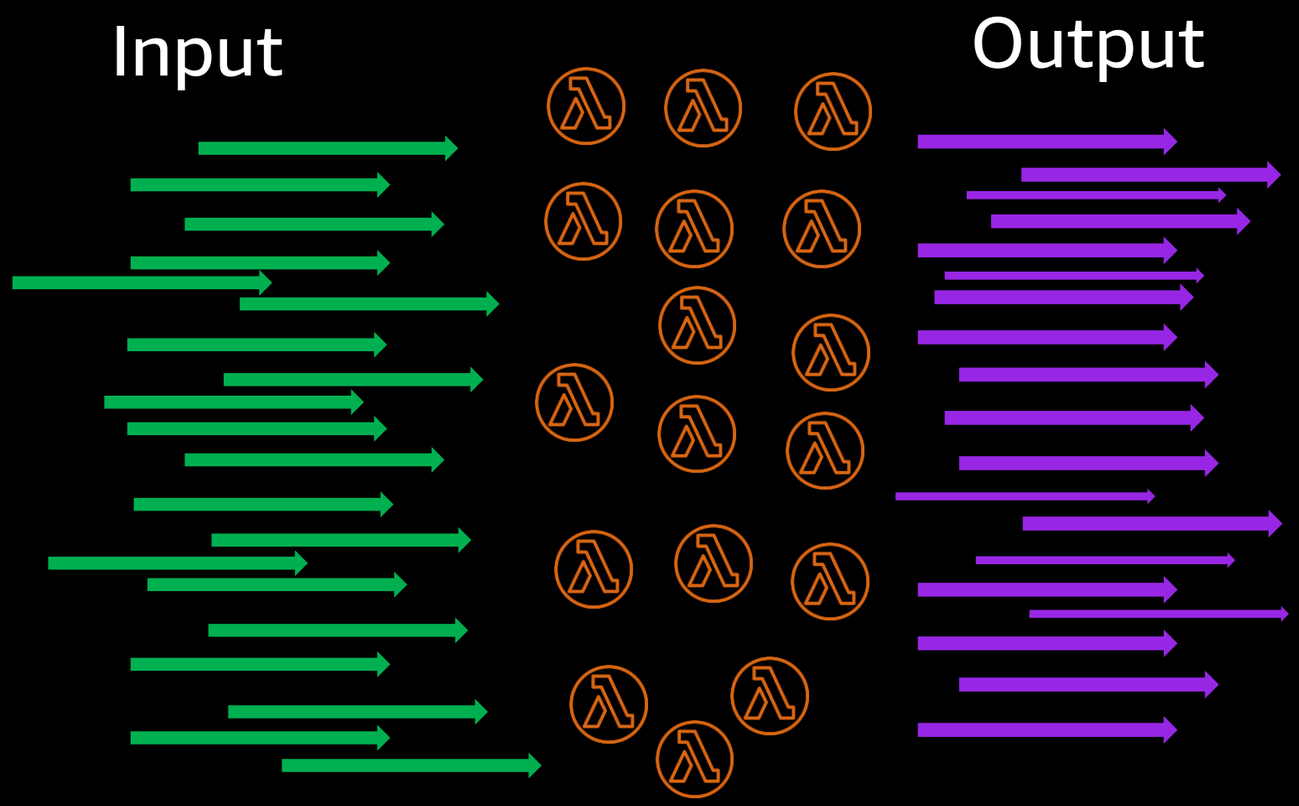
A REAL-WORLD EXAMPLE



- Each CRUD operation becomes a separate Lambda function
- API Gateway routes request to a specific Lambda function

Independent reusable functions

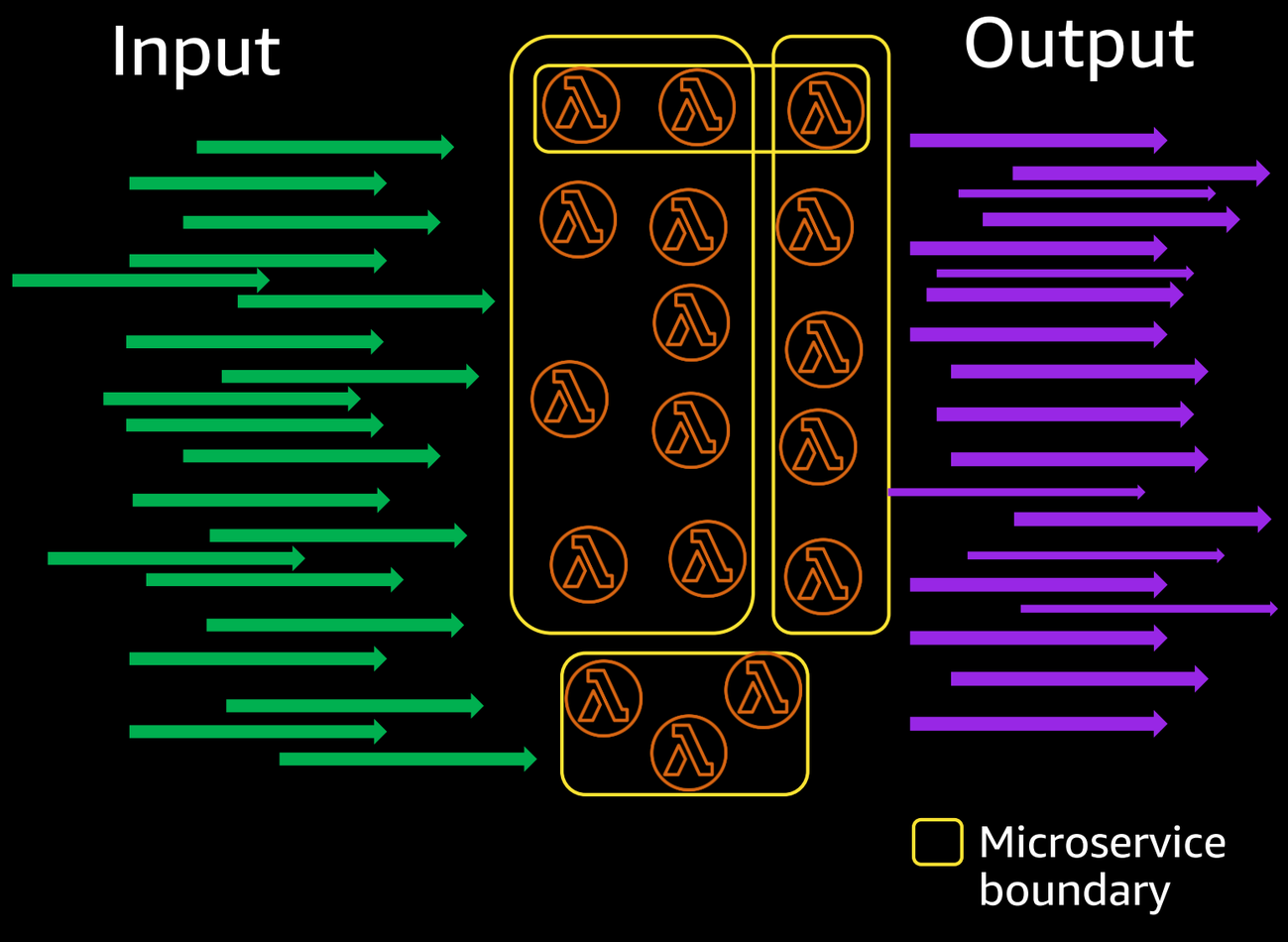
SCOPE LAMBDA FUNCTIONS TO FINITE BUSINESS LOGIC



- Each function is independent
- Reduces the impact of bugs or failures in one area of code that affect other application operations
- Allows functions to scale independently according to demand

From functions to microservices

SPLIT APPLICATIONS INTO MULTIPLE SERVICES



- Each component is a single building block that makes up an application together with other blocks
- Each microservice delivers a set of capabilities on a specific domain
- Communicate between services using events
- Change, upgrade, and replace with no impact on the remaining microservices components

Reusable Lambda functions

BUILD FASTER WITH SMALL REUSABLE MODULES

Microservice



What if two separate microservices access or own the same function?

- Share the logic, not the function
- Consider merging services / making function standalone
- Use AWS services where possible
- Assume failures will occur
- Use contract testing and versioning

Microservice interaction models

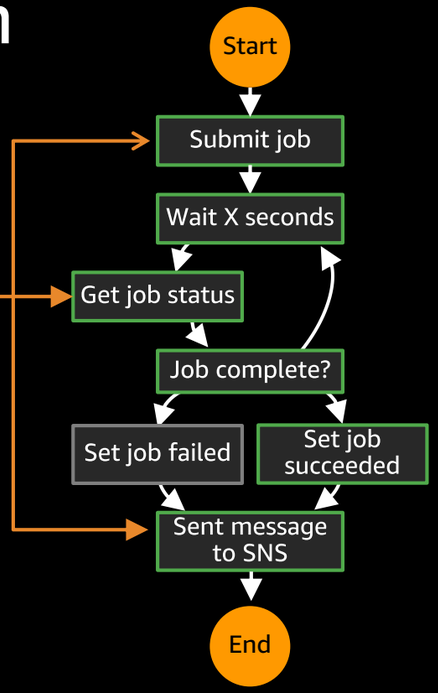
AS YOU GROW AND EXPAND, COORDINATION IS COMPLICATED

Orchestration



AWS Step Functions

AWS Lambda functions

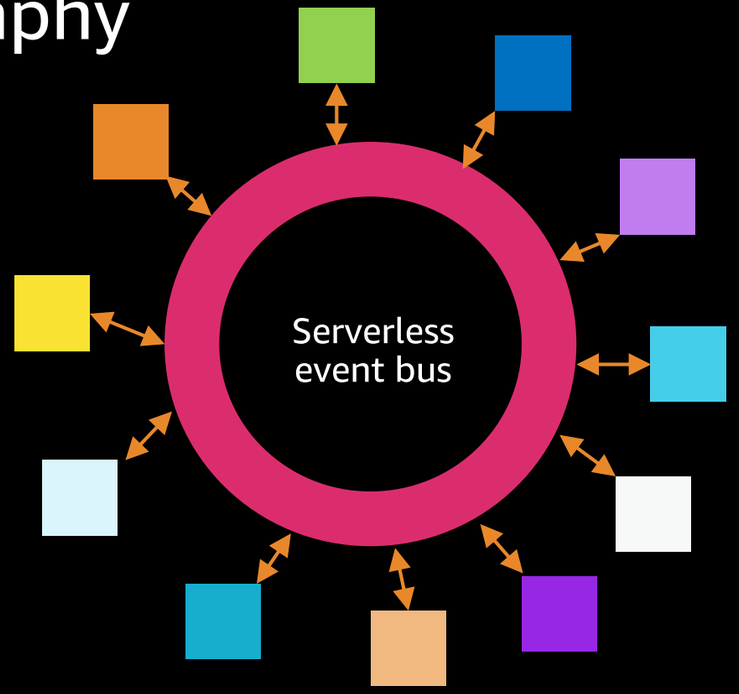


Coordinate the components of distributed applications and microservices using visual workflows

Choreography

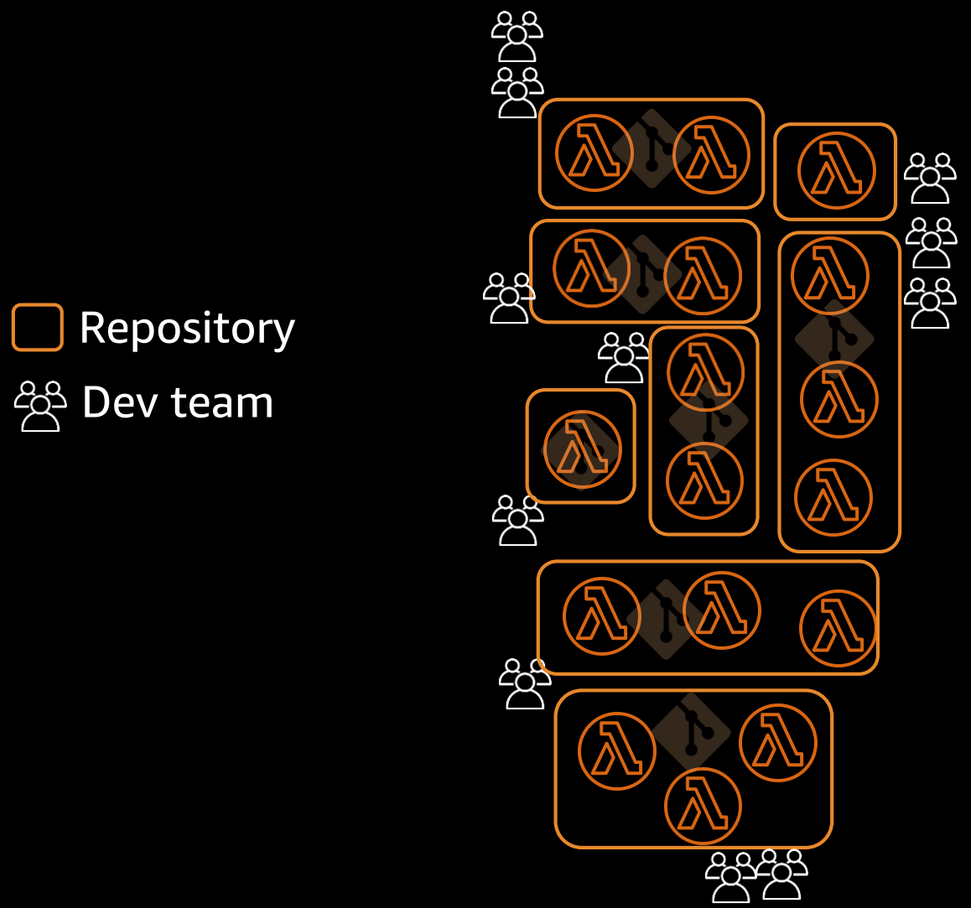


Amazon EventBridge



Produce and consume messages from a serverless event bus; services don’t need to know about each other, just about the bus

Organizing code repositories

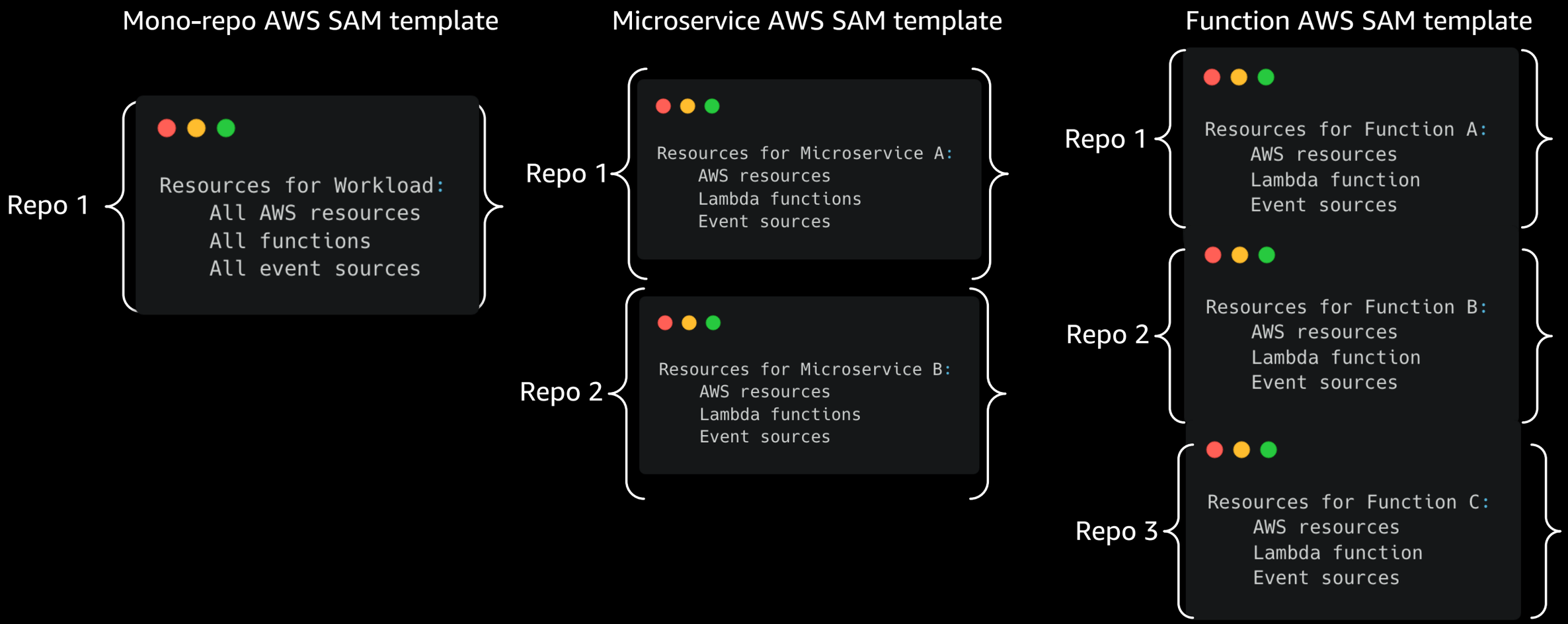


- Group common functionality together for ownership and repositories
- Create repos around groups of functions and resources that define a microservice
- Independent functions that do not share other AWS resources may be appropriate to create as a single repository

Organizing code repositories

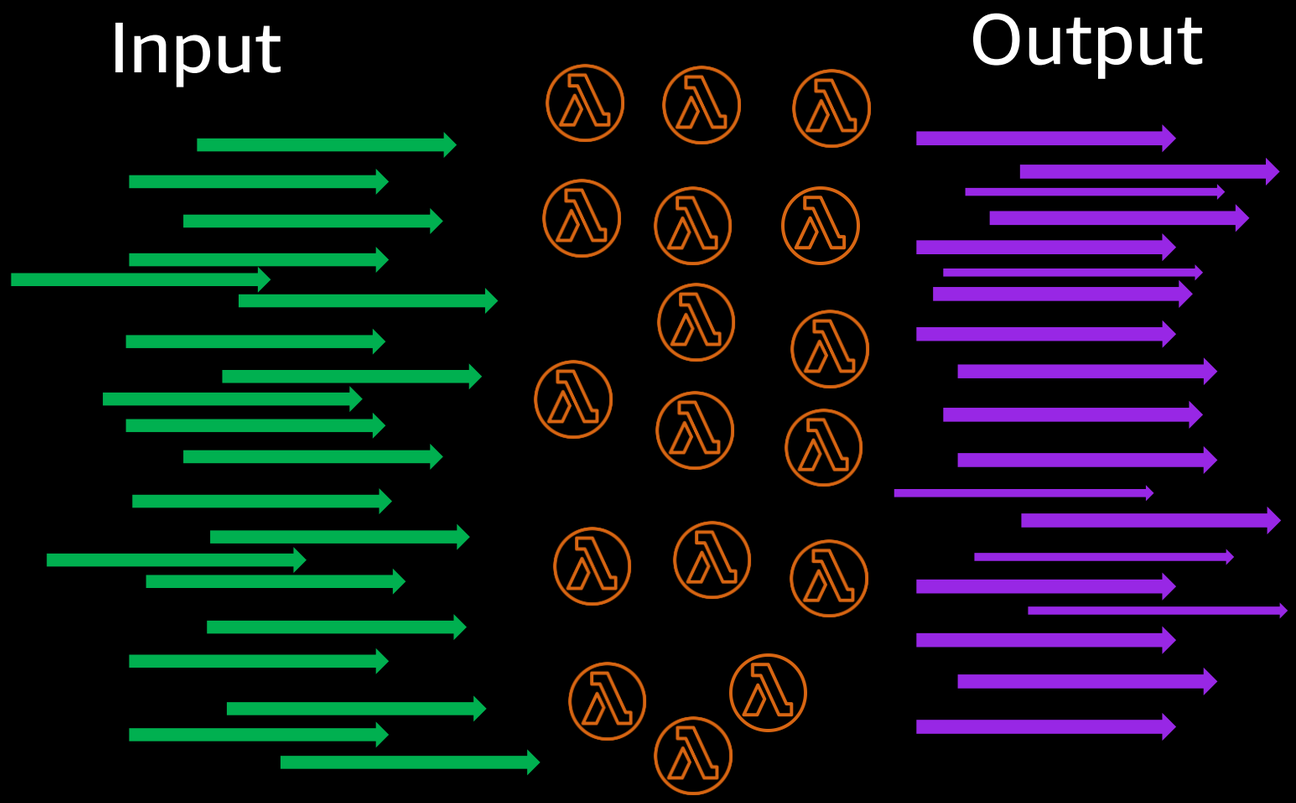
AWS SAM template contains all the resources and function definitions needed for an application

Break application into microservices with separate templates to split repos and resource groups



Your serverless application – Day 100

GROWTH CHALLENGE 3 – ENFORCING SECURITY BEST PRACTICES



With the changing structure of a growing decoupled serverless application, some new security challenges arise

- Do more resources = more permissions to manage?
- Observability into serverless applications across multiple versions and Regions
- Restricting code libraries to prevent “bad actor” attempts

Enforcing security – Castle and moat / Zero Trust

CHANGING THE WAY WE THINK ABOUT SECURITY

From

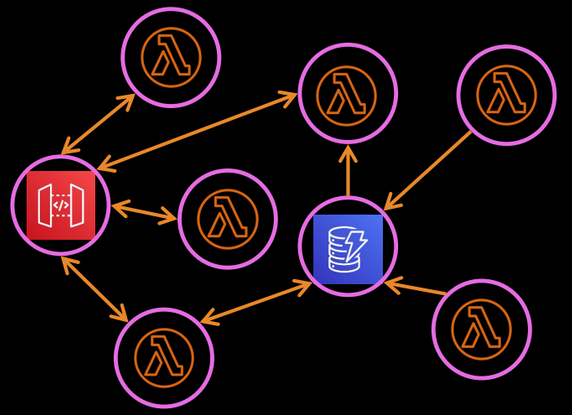
Castle and moat



Protecting network perimeters with firewalls, proxy servers, honeypots, and other intrusion prevention tools

to

Zero Trust



Trust “no one” by default

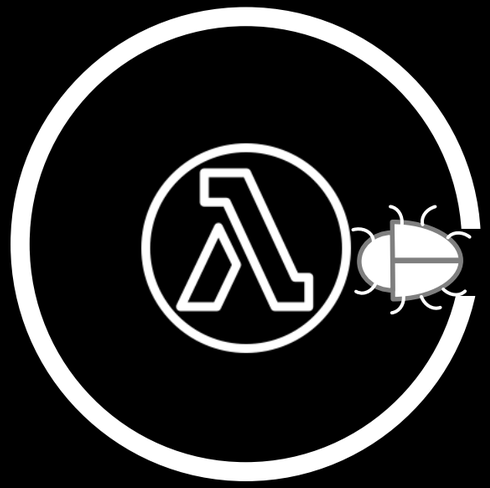
Build micro-perimeters around each resource to enforce strict verification of every person or service

Enforcing security – Castle and moat / Zero Trust

CHANGING THE WAY WE THINK ABOUT SECURITY

From

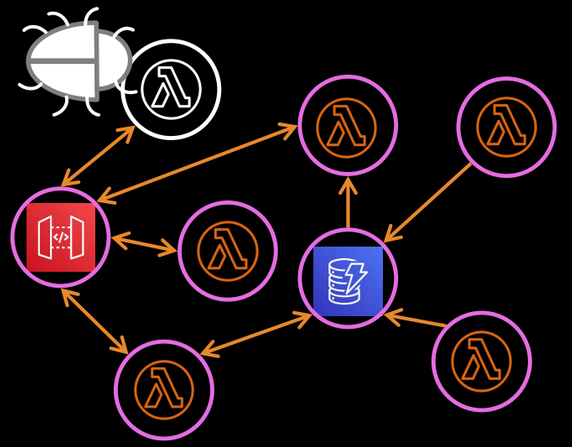
Castle and moat



If a bad actor gains access to the network, they have access to everything inside

to

Zero Trust

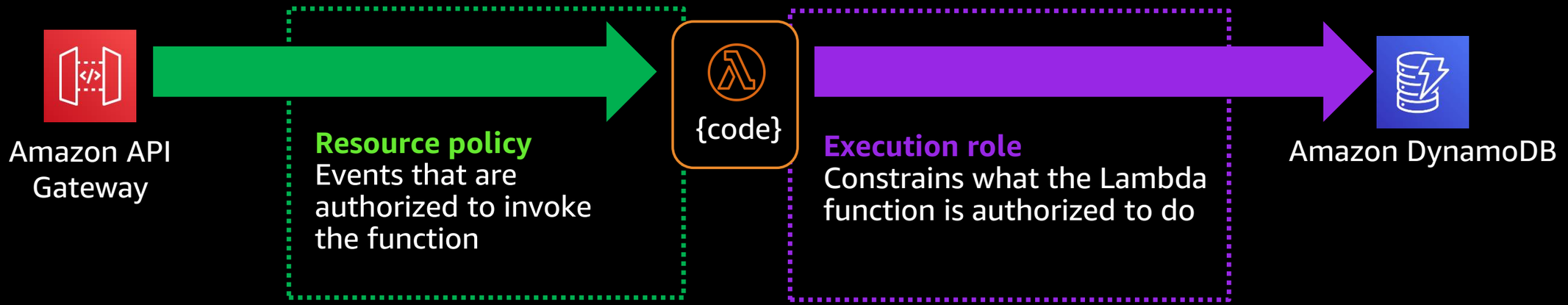


If a bad actor gains entry, they are limited to only the service within that micro-perimeter

Enforcing security – Implementing Zero Trust

DECOUPLE BUSINESS LOGIC FROM SECURITY POSTURE WITH ROLES AND PERMISSIONS

- Use AWS IAM resource policies and execution roles
- Default to deny – never use * permissions



Enforcing security – Implementing Zero Trust

DECOUPLE BUSINESS LOGIC FROM SECURITY POSTURE WITH ROLES AND PERMISSIONS

- Use AWS IAM resource policies and execution roles
- Default to deny – never use * permissions

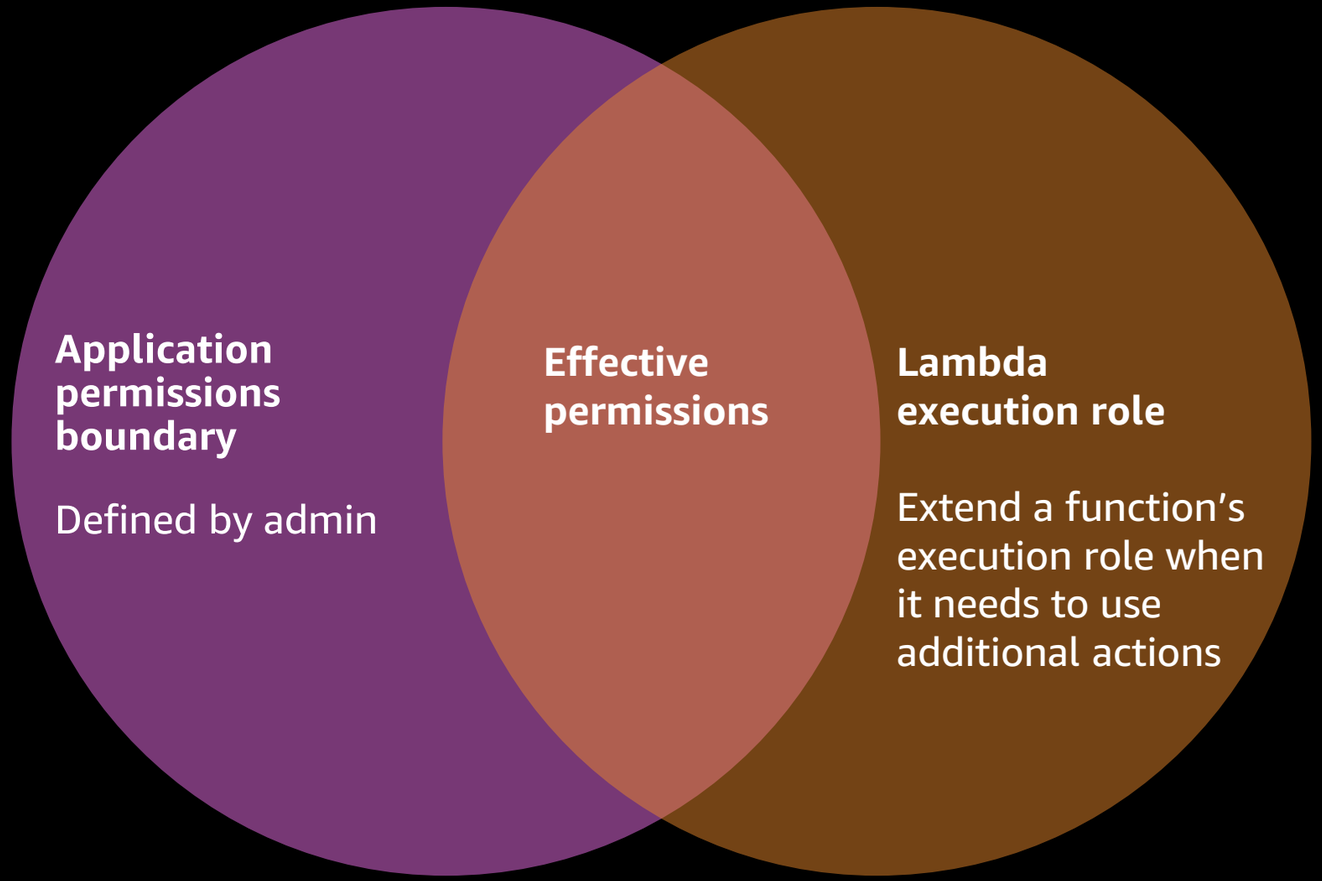






Enforcing security – Permissions boundaries

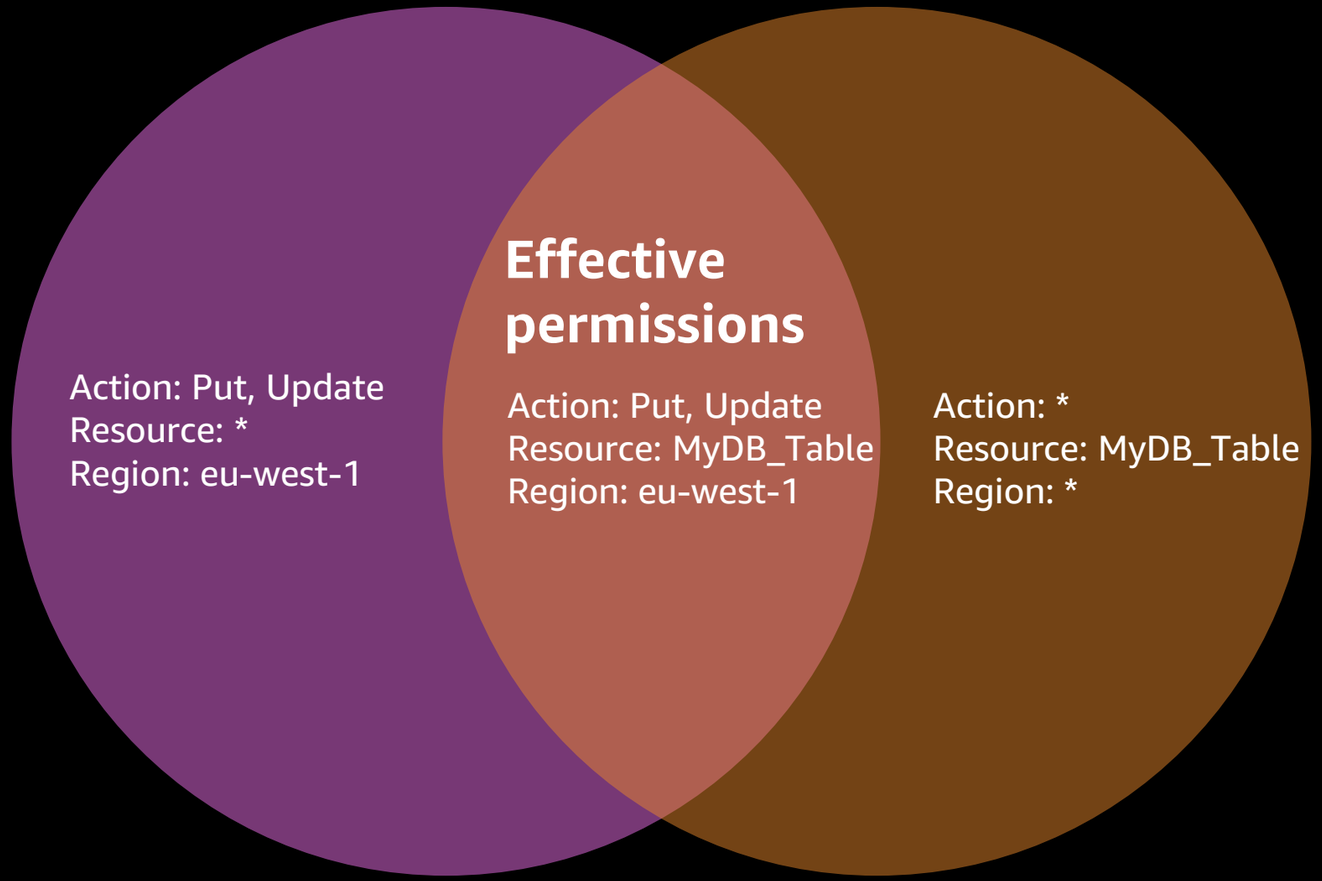
PERMISSIONS BOUNDARIES LIMIT THE SCOPE OF A FUNCTION'S EXECUTION ROLE



Execution role permissions can be limited by the application's permissions boundaries

Enforcing security – Permissions boundaries

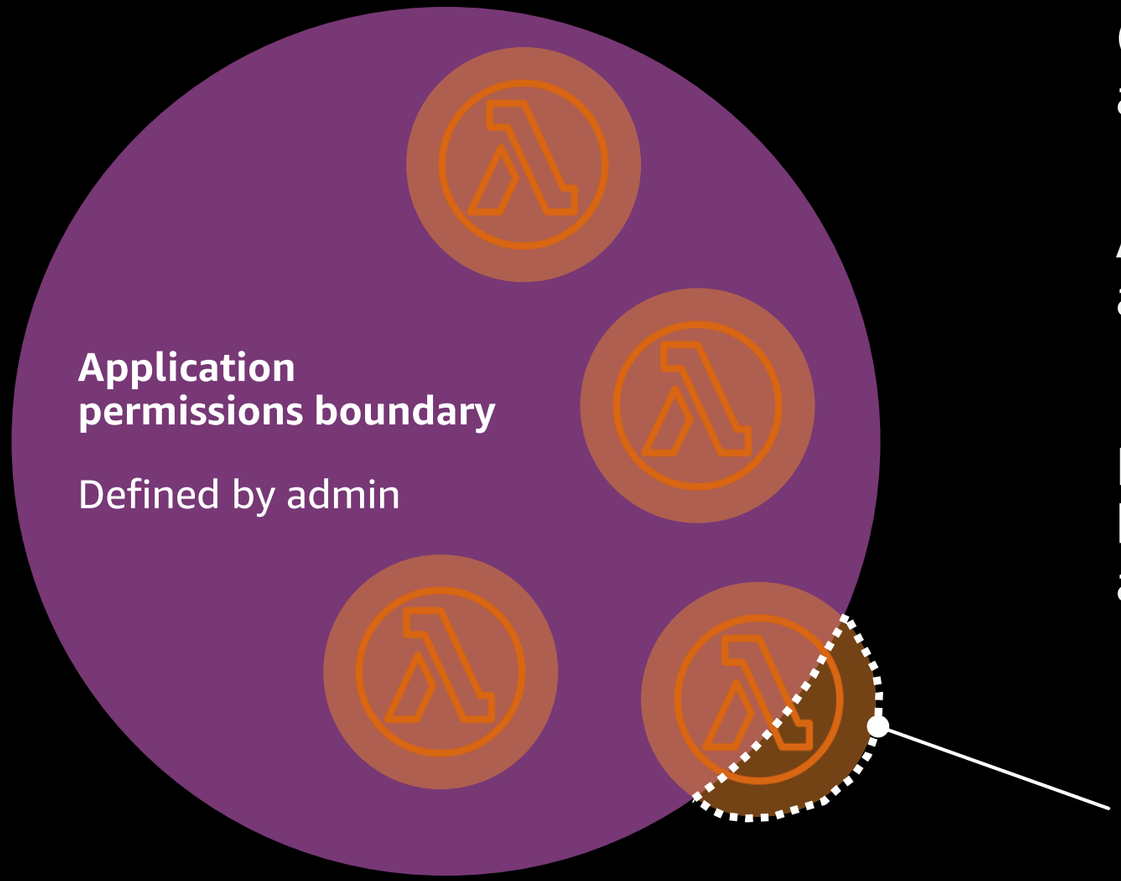
PERMISSIONS BOUNDARIES LIMIT THE SCOPE OF A FUNCTION'S EXECUTION ROLE



Execution role permissions can be limited by the application's permissions boundaries

Enforcing security – Permissions boundaries

PERMISSIONS BOUNDARIES LIMIT THE SCOPE OF A FUNCTION'S EXECUTION ROLE



Create a permissions boundary at the application level

All functions associated with the application are limited by that boundary

By default, an application's permissions boundaries enable functions to perform actions on the resources in the application

Use AWS services where possible

AWS SERVICES CAN HELP YOU APPLY SECURITY BEST PRACTICES

Monitoring



AWS X-Ray

Keys and secrets



AWS Secrets Manager



Parameter Store

Business approved-code libraries



AWS Lambda layers

Security – Monitoring

CONTINUOUSLY MONITOR YOUR ZERO TRUST ECOSYSTEM

Visibility must be decoupled from application

Monitor for insecure flows and attempts to force a function into an unsafe code path



AWS X-Ray

Tag functions so that they are easy to track and group together

Historic view helps to provide compliance reporting

Real-time view to map out workflows and implement policies

AWS Partners

APPDYNAMICS part of Cisco
New Relic.
aqua
pulumi
dashbird
serverless
DATADOG
snyk
epsagon
splunk>
Kong
STACKERY
lumigo
THUNDRA

Security – Keeping secrets

STORE SECRETS IN AWS SECRETS MANAGER, NOT ENVIRONMENT VARIABLES

- Create single source of truth
- Enables automated scheduled rotation
- Audit secrets access

Pass secret ID as environment variable

```
Parameters
  ExampleSecret:
    Type: AWS::SSM::Parameter::Value<String>
    Default: 'SECRET1234567890'

Resources:
  ExampleLambdaFunction:
    Type: AWS::Serverless::Function # Function Resource:
    Properties:
      Environment:
        Variables:
          SECRET_ID: !Ref ExampleSecret
```

Retrieve secrets using AWS SDK

```
var params = {
  SecretId: process.env.SECRET_ID, /* required */
};
secretsmanager.getSecretValue(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
});
```

Security – Code libraries

SHARE BUSINESS-APPROVED CODE WITH AWS LAMBDA LAYERS AND AWS SERVERLESS APPLICATION REPOSITORY

Use Lambda layers published by others within your AWS Organizations organization or even by trusted third-party publishers

Use of published Lambda layers can be controlled via layer permissions policies

Package custom libraries or dependencies within Lambda layers and use them within your Lambda functions

Maintain a single source for libraries and dependencies for easier patching and updates

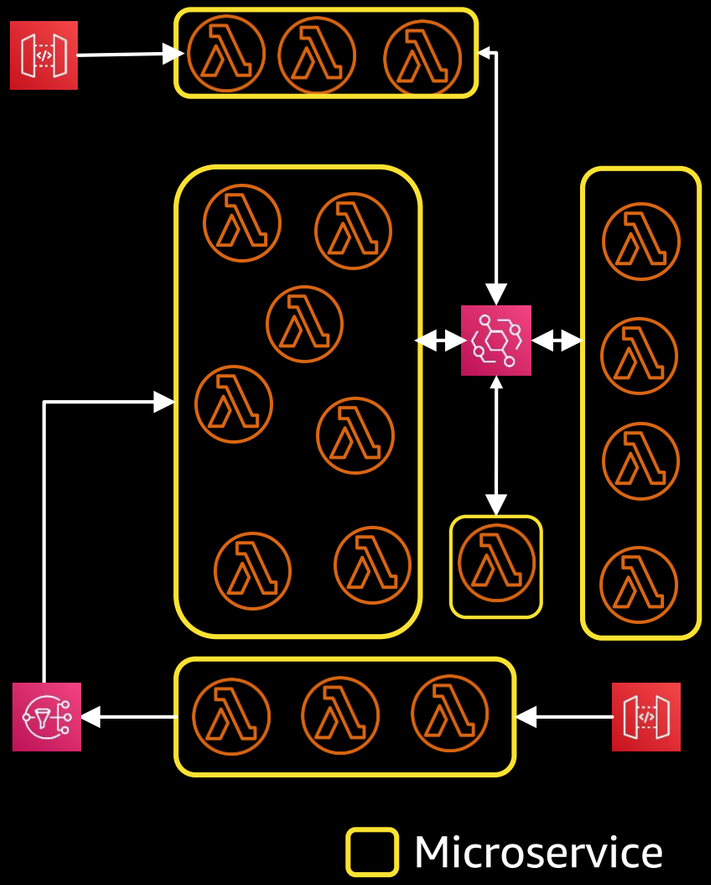
Example: Grant permission to layer version 3 for all accounts in an organization

```
aws lambda add-layer-version-permission
--layer-name my-layer \
--statement-id engineering-org \
--version-number 3 --principal '*' \
--action lambda:GetLayerVersion \
--organization-id o-t194hfs8cz \
--output text
```



Your serverless application – Day 100

GROWTH CHALLENGE 4 – MANAGING MULTIPLE AWS ACCOUNTS

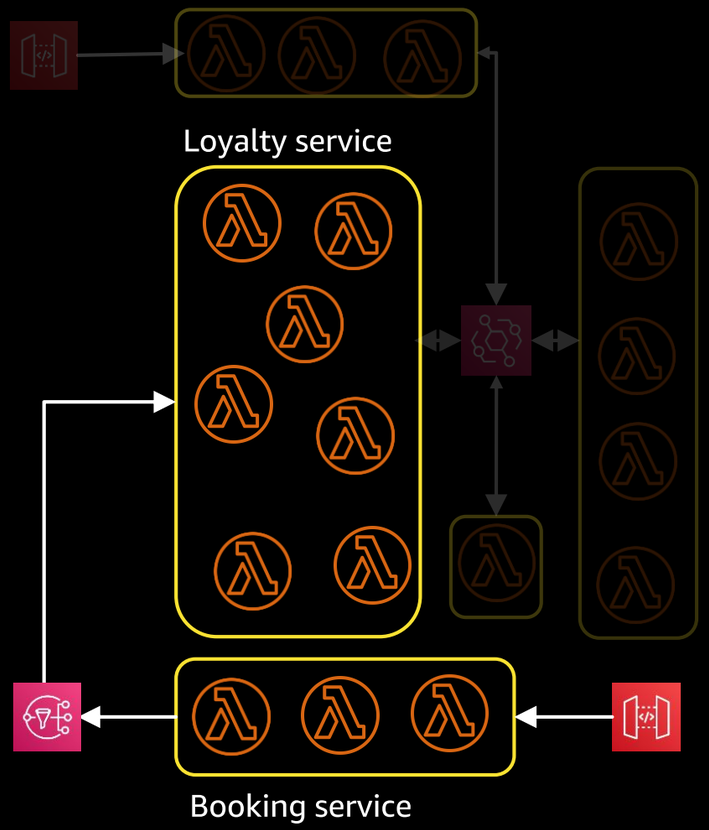


Resources separated into different AWS accounts for security, billing, and resource isolation – can sometimes result in the following challenges

- Cross-account deployment
- Cross-account lookup
- Cross-account communication

Cross-account deployment using AWS CodePipeline

EXAMPLE AIRLINE BOOKING APPLICATION – TWO SERVICES



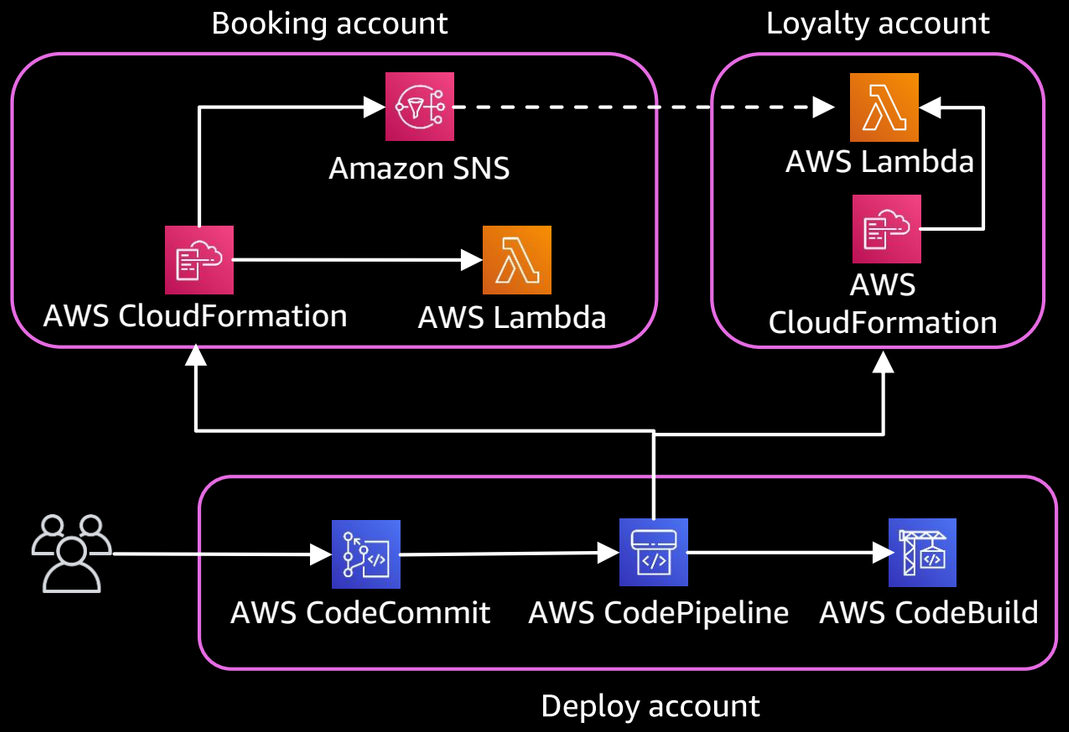
Loyalty service

Booking service

Communicating via Amazon SNS

Cross-account deployment using AWS CodePipeline

EXAMPLE AIRLINE BOOKING APPLICATION – DEPLOYED ACROSS THREE ACCOUNTS



Deploy account: A central location for CI/CD services

Booking account: For booking microservice resources

Loyalty account: For loyalty points microservice resources

Cross-account deployment using AWS CodePipeline

EXAMPLE AIRLINE BOOKING APPLICATION – DEPLOYED ACROSS THREE ACCOUNTS

```
- Name: DeployToTest
  Actions:
    - Name: CreateChangeSetTest
      ActionTypeId:
        Category: Deploy
        Owner: AWS
        Version: 1
        Provider: CloudFormation
      Configuration:
        ChangeSetName: !Join ['-',[!Ref ProjectName, 'lambda', 'CS']]
        ActionMode: CHANGE_SET_REPLACE
        StackName: !Join ['-',[!Ref ProjectName, 'lambda']]
        Capabilities: CAPABILITY_NAMED_IAM
        TemplatePath: BuildOutput::output-sam-template.yml
        TemplateConfiguration: BuildOutput::sam-config.json
        RoleArn: !Ref NonProdCloudFormationServiceRole
      InputArtifacts:
        - Name: BuildOutput
      RunOrder: 1
      RoleArn: !Ref NonProdCodePipelineActionServiceRole
```

AWS CloudFormation template in the deploy account

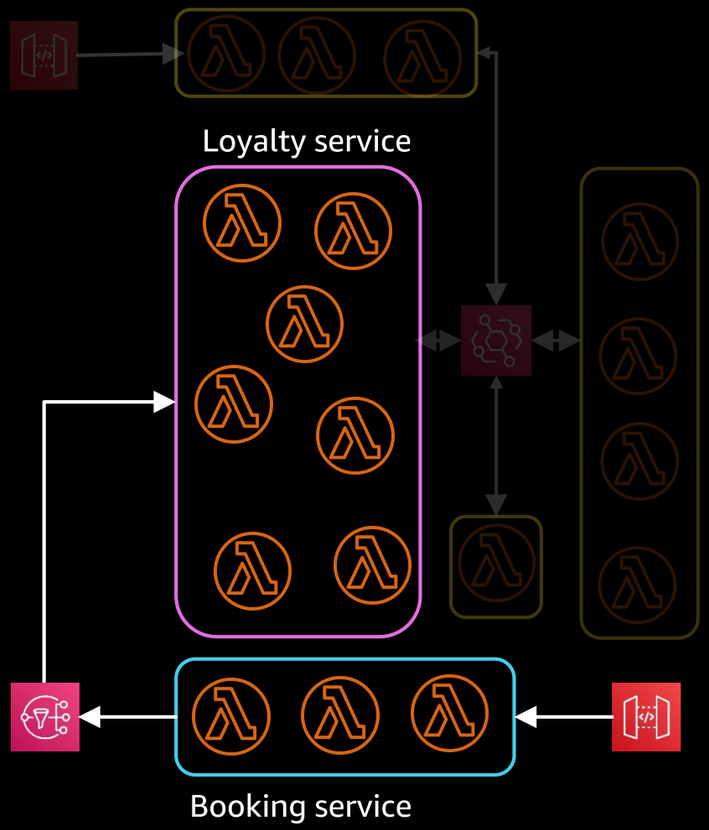
2 roles created in the booking account

Service role assumed by CloudFormation when it carries out the CHANGE_SET_REPLACE action in the booking account

Allows access to the CodePipeline artifacts in the Amazon S3 bucket in the deploy account

Cross-account resource communication

CUSTOM RESOURCES



The publisher: Publishes a flight booking event onto an SNS topic that exists in the booking account

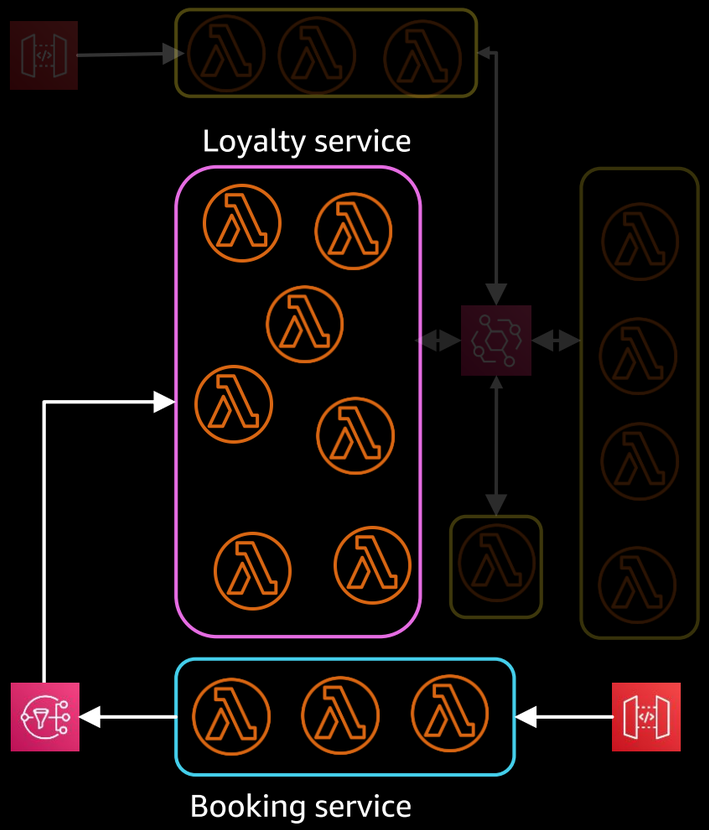
The consumer: Looks up the ARN of the SNS topic at deployment time in order to set up a subscription to it

Resource names should not be hardcoded into CloudFormation templates

Use a CloudFormation custom resource to enable cross-account stack referencing

Cross-account resource communication

USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



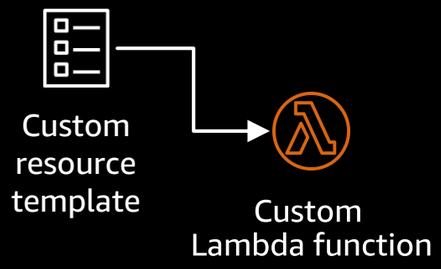
Does not create a dependency between the two stacks

CloudFormation calls the Lambda API to invoke the function whenever the custom resource is created, updated, or deleted

The custom resource is used to list the stack export variables in another account and return these values to the calling CloudFormation stack

Cross-account resource communication

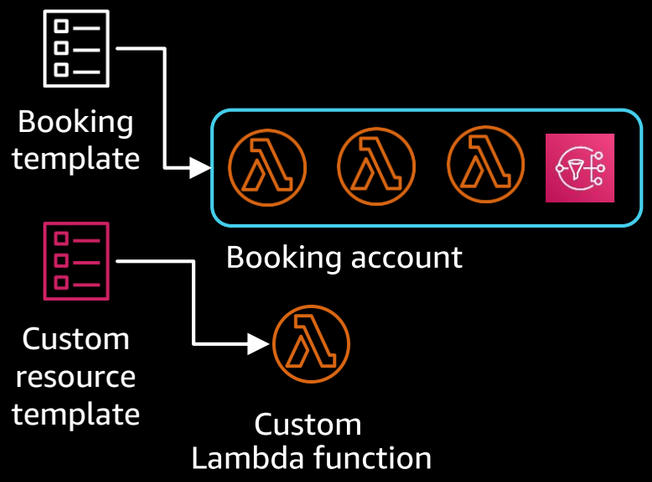
USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



Deploying the custom resource with a separate template

Cross-account resource communication

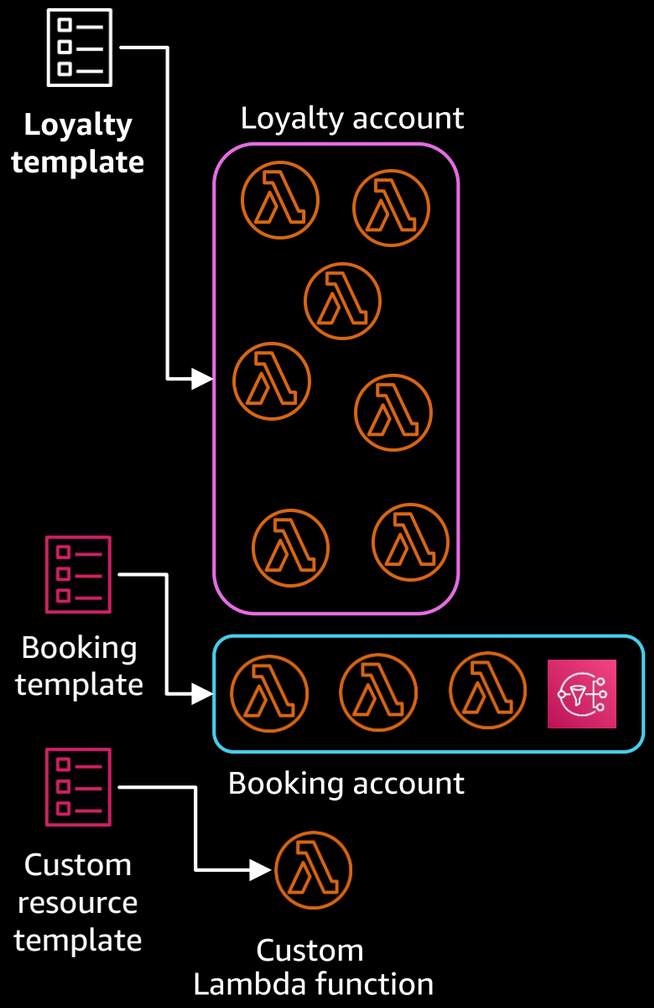
USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



The booking microservice is created by the booking SAM template and deployed onto the booking account

Cross-account resource communication

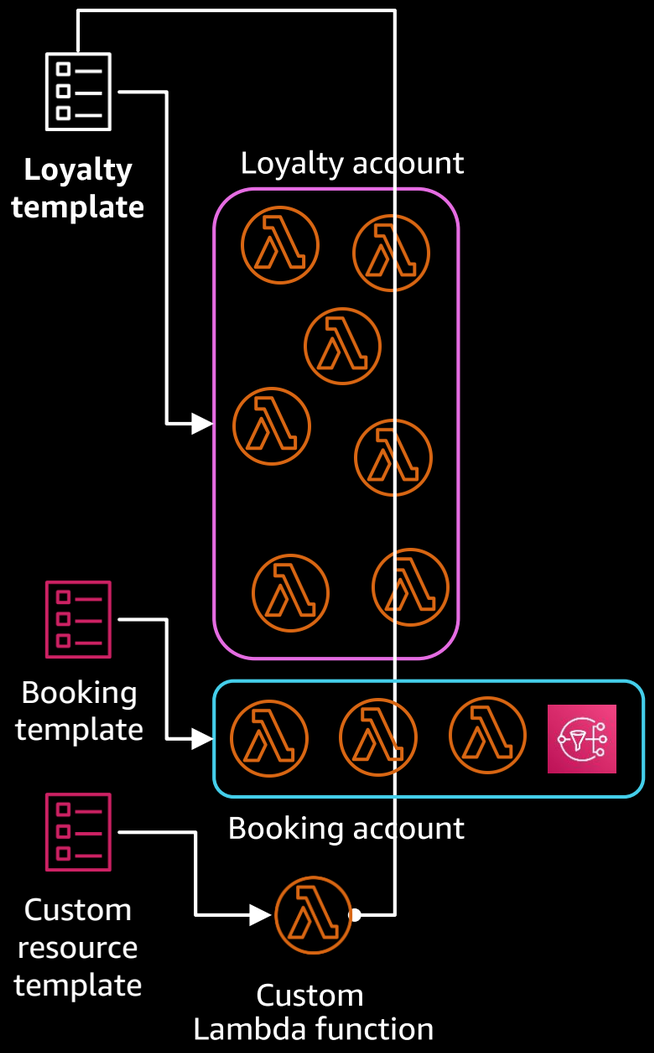
USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



The loyalty microservice is created by the loyalty SAM template and deployed onto the loyalty account

Cross-account resource communication

USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS

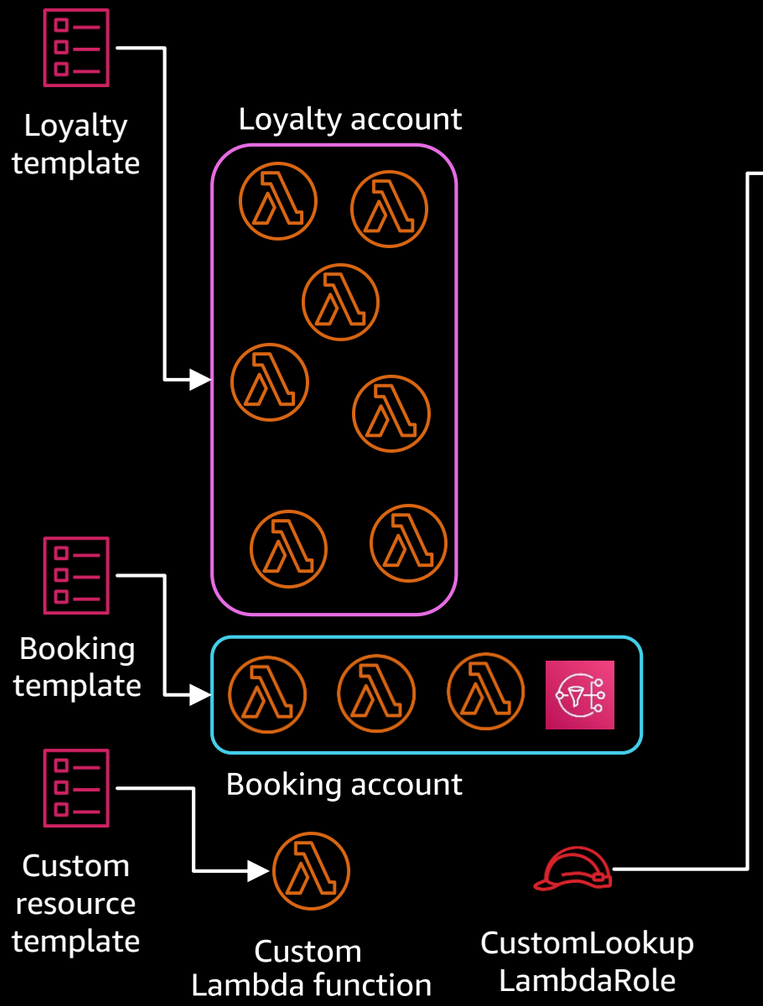


This template uses the custom Lambda resource to look up the ARN of the booking SNS topic – the custom resource is specified by the **CustomLookup** resource

```
CustomLookup:
  Type: Custom::Lookup
  Properties:
    ServiceToken: !ImportValue custom-lookup-exports
    loglevel: info
```

Cross-account resource communication

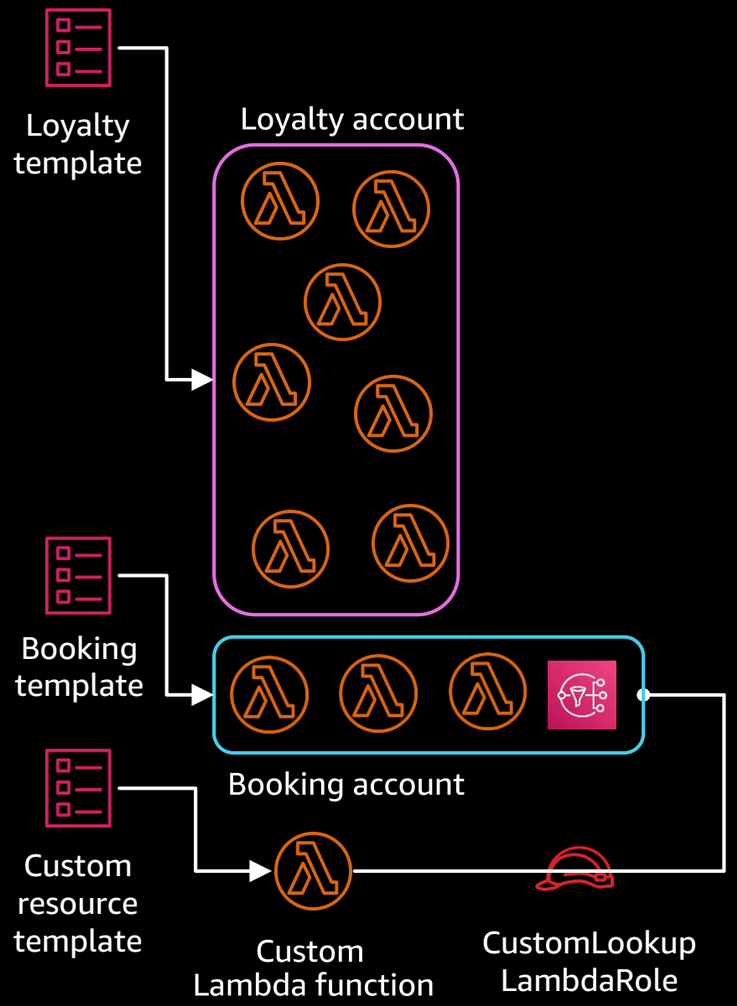
USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



The Lambda function runs with the CustomLookupLambdaRole IAM role – this allows it to assume a cross-account role in the *deploy* account

Cross-account resource communication

USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



The custom resource uses the AWS SDK to obtain all the stack exports in the booking account

```
def get_exports():
    print('getting stack exports')
    stackexports=[]

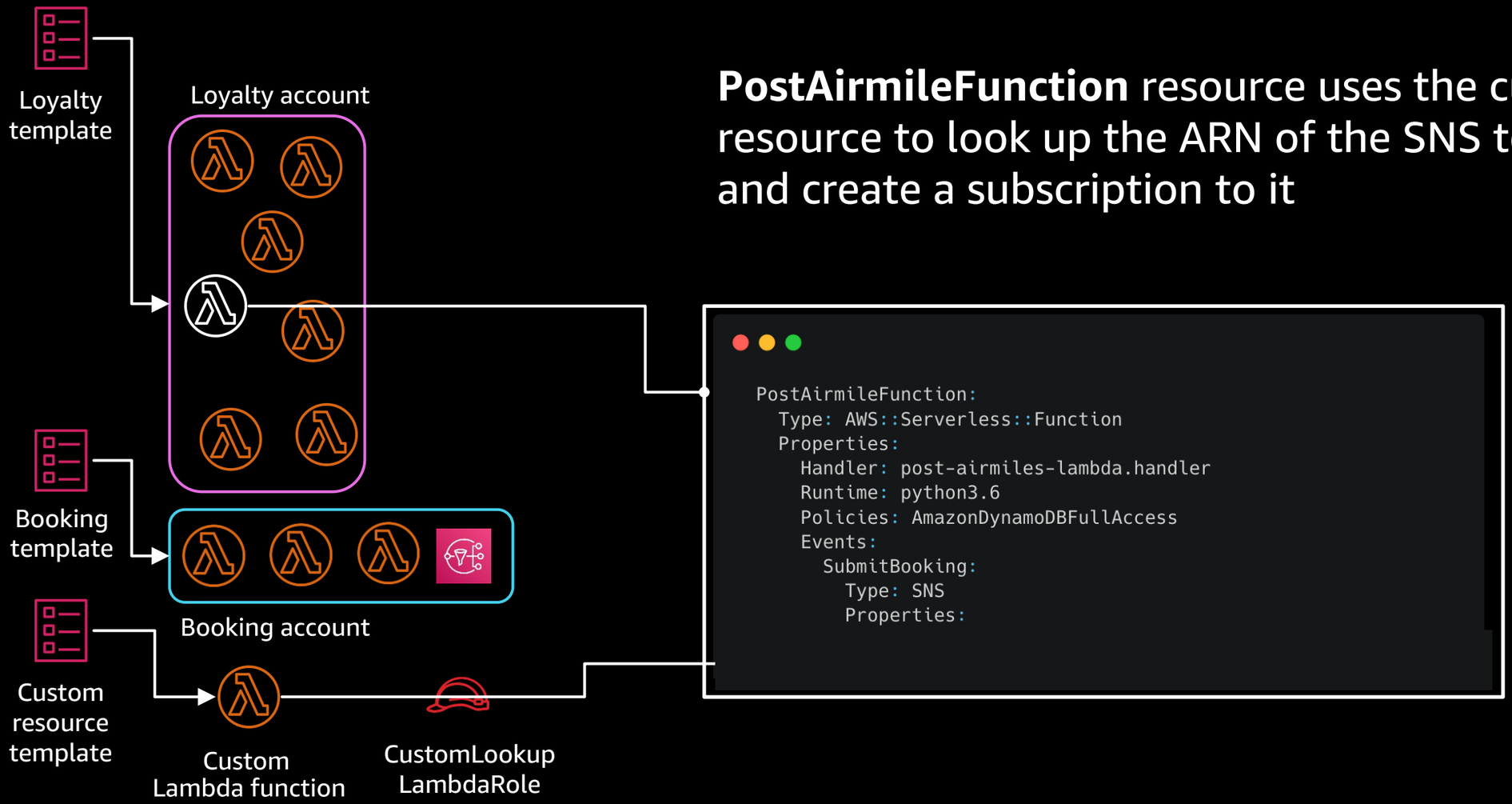
    response = cfn.list_exports()
    stackexports.extend(response['Exports'])
    while 'NextToken' in response and response['NextToken'] is not None:
        token = response['NextToken']
        response = cfn.list_exports(NextToken=token)
        stackexports.extend(response['Exports'])
    print('list of stack exports: ' + str(stackexports))

    #create dict of stack exports
    stackexportvalues = {}
    for stackexport in stackexports:
        stackexportvalues[stackexport['Name']] = stackexport['Value']
    print('list of stack exports in dict: ' + str(stackexportvalues))

    return stackexportvalues
```

Cross-account resource communication

USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS

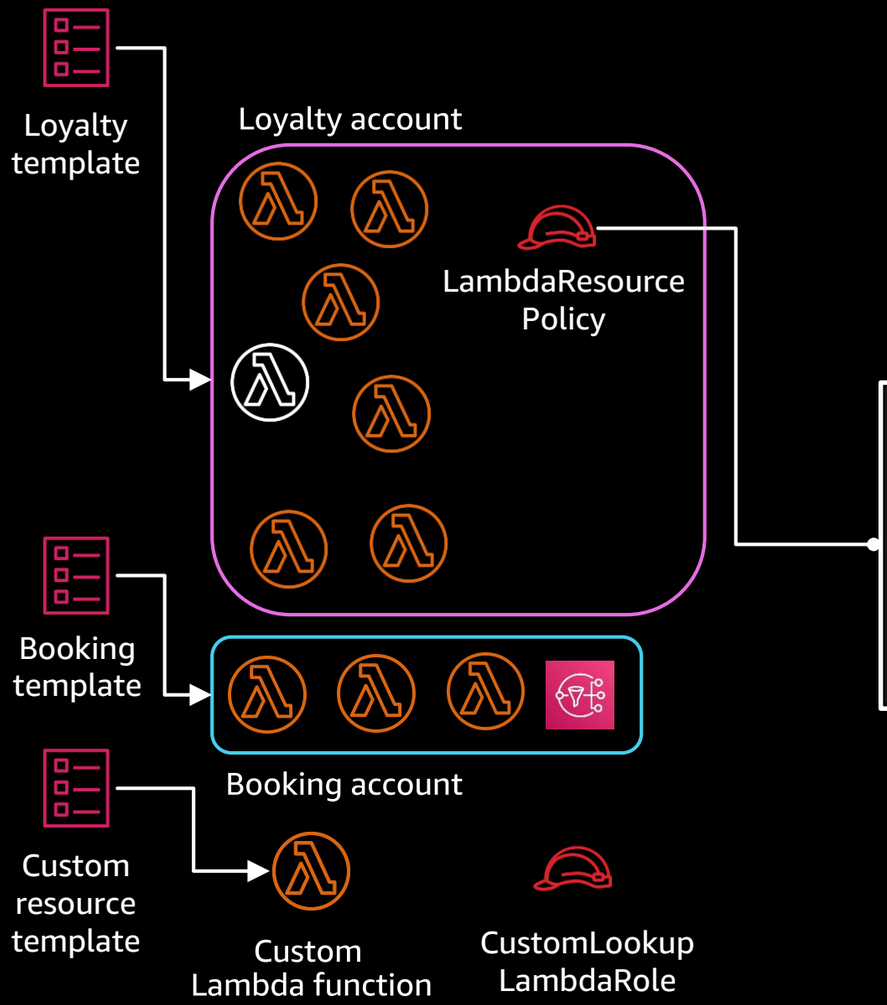


PostAirmileFunction resource uses the custom resource to look up the ARN of the SNS topic and create a subscription to it

```
PostAirmileFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: post-airmiles-lambda.handler
    Runtime: python3.6
    Policies: AmazonDynamoDBFullAccess
    Events:
      SubmitBooking:
        Type: SNS
        Properties:
```

Cross-account resource communication

USING A CUSTOM RESOURCE TO LOOK UP STACK EXPORTS



Permissions for SNS to invoke the Lambda function is granted by the ***LambdaResourcePolicy*** resource

```
LambdaResourcePolicy:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !GetAtt PostAirmileFunction.Arn
      Principal: sns.amazonaws.com
      Action: lambda:InvokeFunction
      SourceArn : !GetAtt CustomLookup.booking-lambda-BookingTopicArn
```

To recap



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In summary

WHEN BUILDING A DECOUPLED SERVERLESS APPLICATION, NEW CHALLENGES CAN ARISE

	Challenges	Can be met by
Monolithic Lambda functions	Tightly coupled codebase adds complexity, with multiple permissions	Breaking Lambda functions apart to create independent reusable building blocks
Mono-repos	Difficult to separate responsibility of ownership	Create repos around groups of functions and resources that define a microservice
Enforcing security best practices	Multiple permissions to manage	Use the Zero Trust model; decouple business logic from security posture
Managing multiple AWS accounts	Cross-account deployment and communication	Use custom resources to enable cross-account stack referencing

For more resources visit serverlessLand.com



https://s12d.com/svs315

Thank you!

Benjamin Smith

 @benjamin_l_s

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.