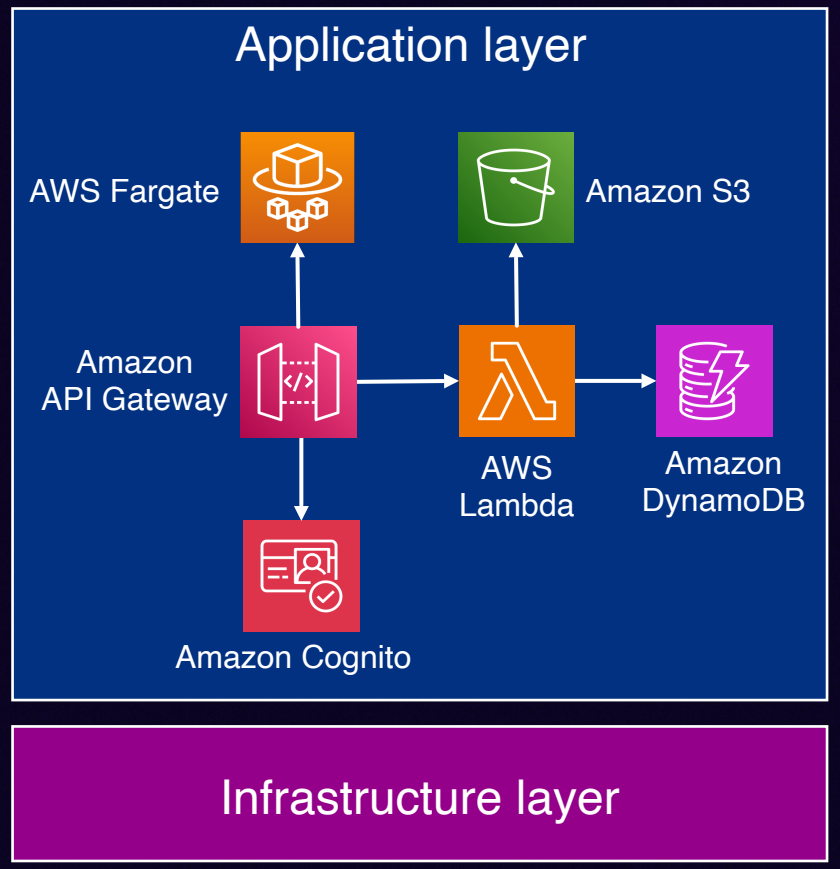
Accelerate serverless deployments using HashiCorp Terraform on AWS

Anton Aleksandrov
Pr. Solutions Architect, Serverless
AWS

Debasis Rath
Sr. Solutions Architect, Serverless
AWS

Gautam Baghel
Partner Solutions Engineer
HashiCorp

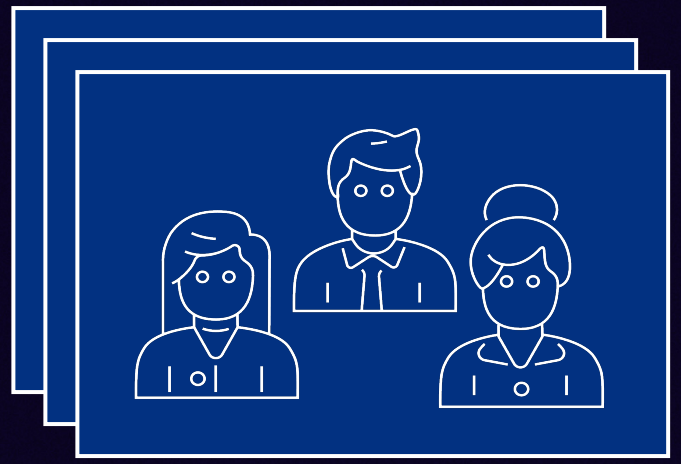
What's so special about Serverless applications



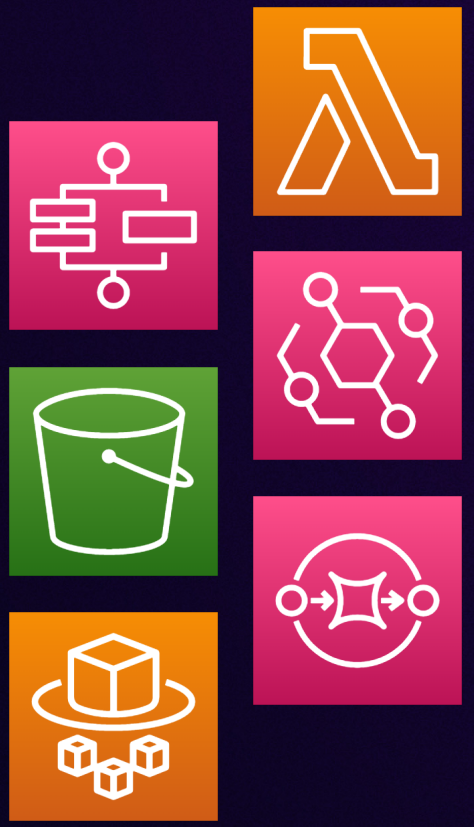
- **"Infrastructure" is redefined.**
- A function, an event-source mapping, an event routing rule are **app resources** owned by the app team
- Updating app resources with IaC tools is commonly a part of **application developer's responsibilities**.

The ownership boundaries

This is an **application resource**! We need the flexibility to control it!



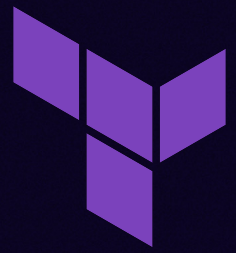
Application development teams



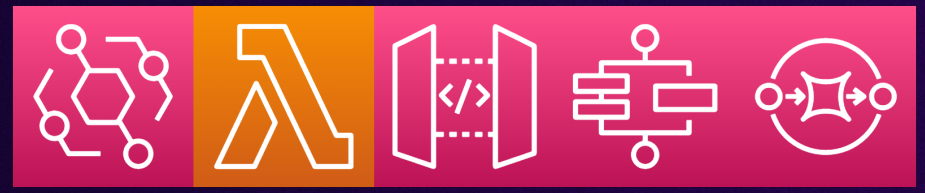
This is an **infrastructure resource**! We set the rules!

Infrastructure/ops team

Building Serverless Applications with Terraform



HashiCorp
Terraform



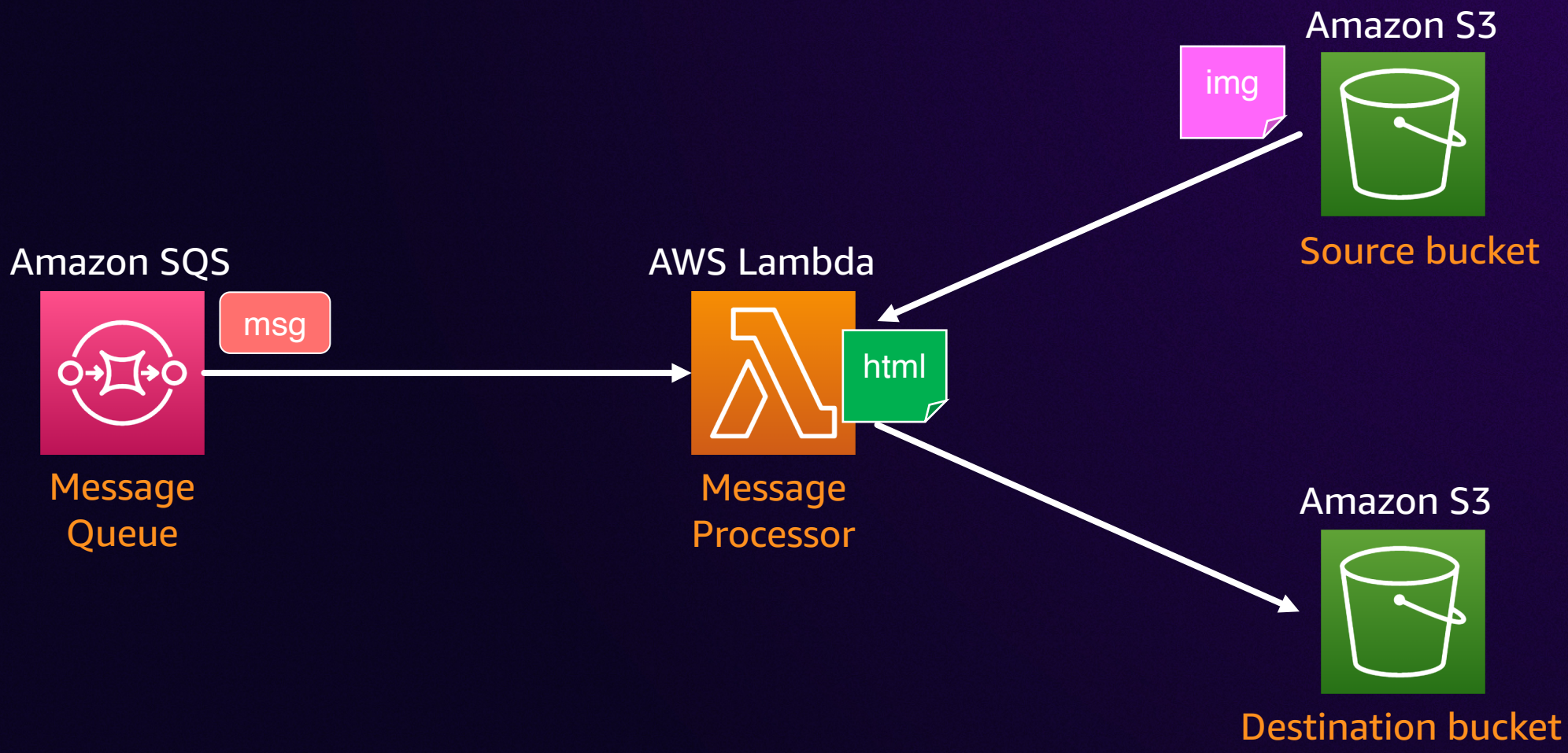
AWS Serverless

- A widely used IaC framework
- 450M+ downloads, 4,350+ customers
- A vast array of integrations, with support for over **3,000** providers
- Terraform AWS Provider is downloaded over **3 billion** times

- Run code without thinking about servers or clusters
- Over **one million** customers are using AWS Serverless services every month
- Processes **10s of trillions** requests every month

Let’s start building

What are we building



Creating S3 buckets

```
resource "aws_s3_bucket" "src_bucket" {
  bucket = "SourceBucket"
  tags = {
    environment = "Production"
  }
}

resource "aws_s3_bucket_ownership_controls" "src_bucket_ownership_controls" {
  bucket = aws_s3_bucket.src_bucket.id
  rule {
    object_ownership = "BucketOwnerPreferred"
  }
}

resource "aws_s3_bucket_acl" "src_bucket_acl" {
  depends_on = [aws_s3_bucket_ownership_controls.src_bucket_ownership_controls]
  bucket     = aws_s3_bucket.src_bucket.id
  acl        = "private"
}
```

Bucket name

Ownership controls

Access controls

Creating SQS queues

```
resource "aws_sqs_queue" "this" {
    name = "greetings_queue"
    sqs_managed_sse_enabled = true
}
```

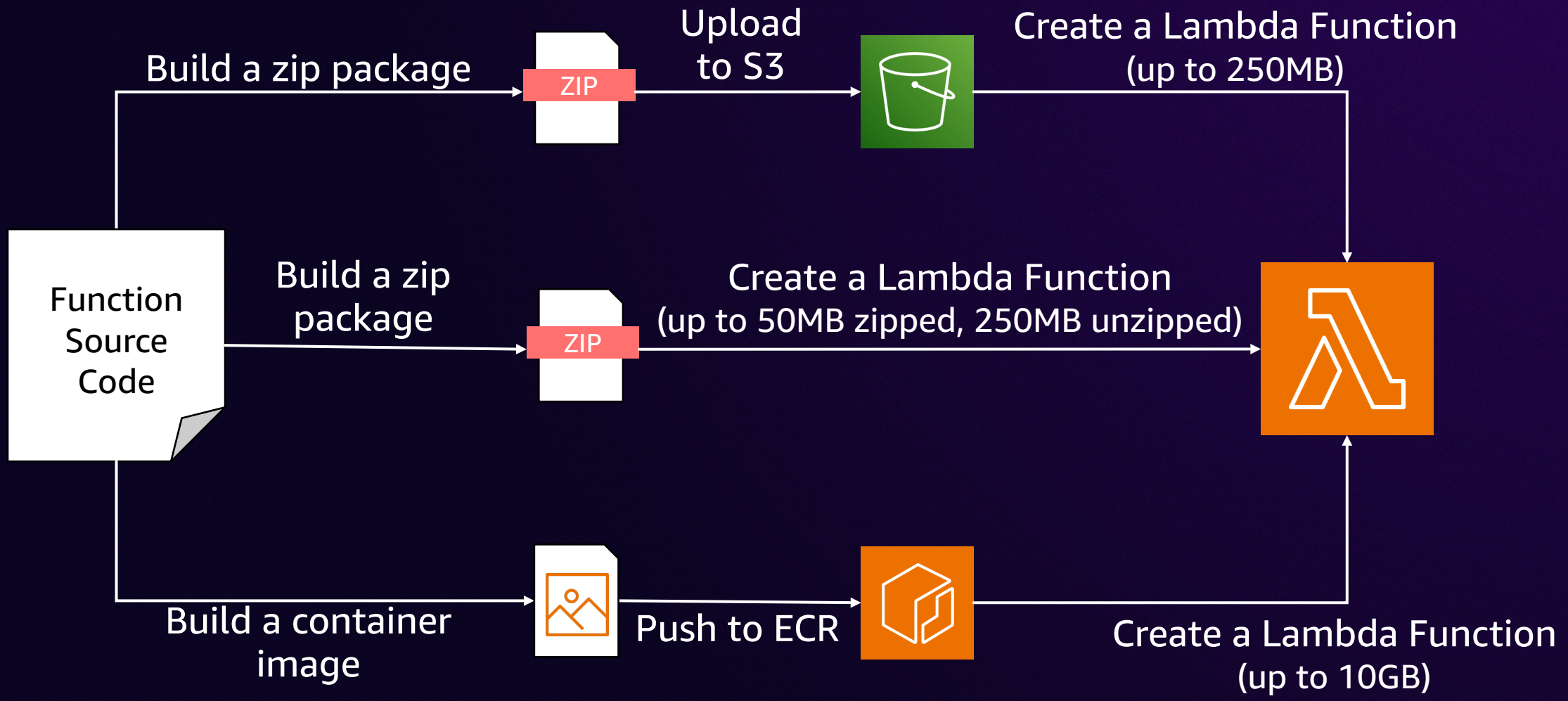
Queue name

Creating Lambda Function Execution Role

```
resource "aws_iam_policy" "this" {
    policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Action   = ["s3:GetObject"]
                Effect   = "Allow"
                Resource = aws_s3_bucket.src_bucket.arn
            },
            {
                Action   = ["s3:PutObject"]
                Effect   = "Allow"
                Resource = aws_s3_bucket.dst_bucket.arn
            },
        ]
    })
}
```

The principle of least privilege

Creating Lambda Functions



Creating Lambda Functions

```hcl
data "archive_file" "this" {
  type        = "zip"
  source_dir  = "../src/lambda/greetings"
  output_path = "greetings_lambda.zip"
}

resource "aws_lambda_function" "this" {
    function_name = "greetings_lambda"
    role          = aws_iam_role.this.arn

    handler       = "index.handler"
    runtime       = "nodejs20.x"
    memory_size   = 256

    filename         = data.archive_file.this.output_path
    source_code_hash = data.archive_file.this.output_base64sha256
```

Package source directory

Function configuration

Function code

Creating Lambda Functions

```hcl
resource "aws_lambda_function" "this" {
    function_name = "greetings_lambda"
    role          = aws_iam_role.this.arn

    handler       = "index.handler"
    runtime       = "nodejs20.x"
    memory_size   = 256

    filename         = data.archive_file.this.output_path
    source_code_hash = data.archive_file.this.output_base64sha256

    # Alternatively, if getting the zip file from S3
    # s3_bucket = ......
    # s3_key    = ......

    #Alternatively, if using function image from ECR
    # image_uri = .....
}
```

Function configuration

Function code

Creating Lambda Functions

```
resource "aws_lambda_function" "this" {
    function_name = "greetings_lambda"
    role          = aws_iam_role.this.arn

    handler       = "index.handler"
    runtime       = "nodejs20.x"
    memory_size   = 256

    filename         = data.archive_file.this.output_path
    source_code_hash = data.archive_file.this.output_base64sha256

    # Alternatively, if getting the zip file from S3
    # s3_bucket = ......
    # s3_key    = ......

    #Alternatively, if using function image from ECR
    # image_uri = .....
}
```

Function configuration

Function code

AWS SAM CLI Support for Terraform

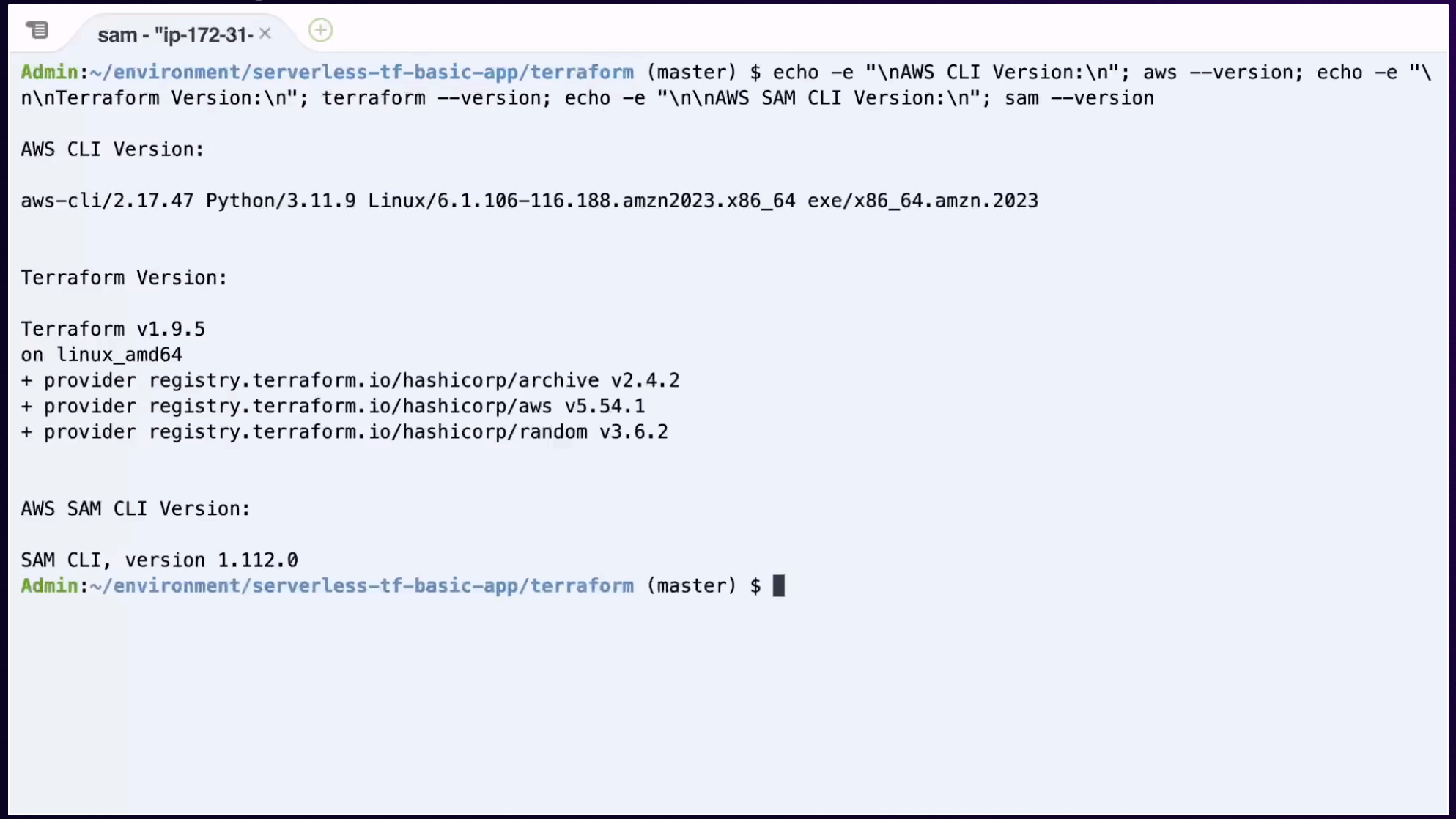
Seamless development and testing experience

Local development, debugging, and testing

Cloud environment templating and deployment

https://aws.amazon.com/blogs/compute/aws-sam-support-for-hashicorp-terraform-now-generally-available/

Local testing with AWS SAM



Scaling

Growth challenges…

Use approved runtimes only

Upload Zip directly or via S3?

Project structure

Different runtimes have different packaging tools

Applying best-practices and policies at scale

Uniformly apply tags

Principle of least privileged access

Do not reinvent the wheel

Modularization

DON'T

```
38016
38017   resource "aws_iam_role_policy_attachment"
38018     policy_arn = aws_iam_policy.greeting_lam
38019     role       = aws_iam_role.greeting_lambd
38020   }
38021
38022   resource "aws_lambda_event_source_mapping"
38023     event_source_arn = var.greeting_queue_ar
38024     function_name    = aws_lambda_function.g
38025     batch_size       = 1
38026
38027     depends_on = [aws_iam_role_policy_attach
38028   }
38029
38030   # EOF
```

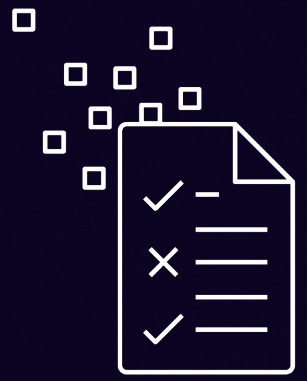
Put all the IaC templates in **one file**

DO

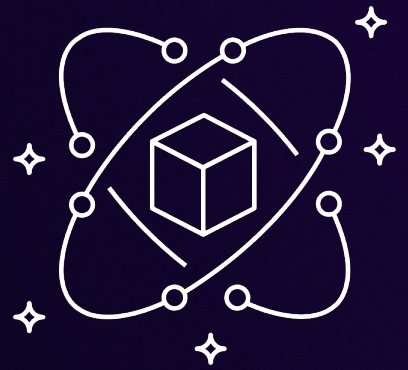
- Storage
- Web Backend
- Observability
- Async processing

Organize your IaC as a collection of **reusable modules**

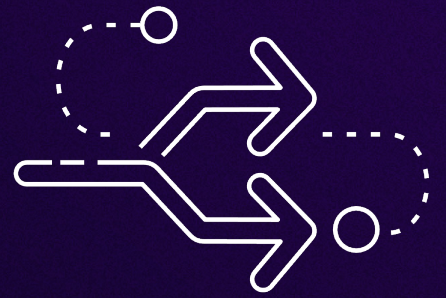
Modularization



Best practices



Reusability



Composability

A Terraform Module

Input (variables.tf)

```
variable "function_name" {
    type = string
}

variable "runtime" {
    type = string
    default = "nodejs20.x"
}
```

Cloud resources configuration (main.tf)

```
resource "aws_lambda_function" "lambda_function" {
  function_name    = var.function_name
  runtime          = var.runtime
```

A Terraform Module

Input (variables.tf)

```
variable "function_name" {
    type = string
}

variable "runtime" {
    type = string
    default = "nodejs20.x"
}

variable "log_retention_days" {
    type = number
    default = 14
}
```

Cloud resources configuration (main.tf)

```
resource "aws_lambda_function" "lambda_function" {
  function_name    = var.function_name
  runtime          = var.runtime
  logging_config {
    log_group = aws_cloudwatch_log_group.this.arn
  }
}

resource "aws_cloudwatch_log_group" "this" {
  name              = "/aws/lambda/${var.function_name}"
  retention_in_days = var.log_retention_days
}

resource "aws_iam_policy" "this" {
    ....
}
```

A Terraform Module

Input (variables.tf)

```
variable "function_name" {
}
```

Cloud resources configuration (main.tf)

```
resource "aws_lambda_function" "products" {
  function_name = local.function_name
  logging_config {
    application_log_level = "INFO"
    system_log_level      = "INFO"
    log_format            = "JSON"
    log_group             = aws_cloudwatch_log_group.lambda_log_group.name
  }
  tracing_config {
    mode = "Active"
  }
}
```

A Terraform Module

```
resource "aws_lambda_function" "products" {
  function_name = local.function_name

  handler     = "index.handler"
  runtime     = "nodejs20.x"
  memory_size = 512
  role        = aws_iam_role.lambda_role.arn
  layers      = [local.powertools_layer_arn]

  environment {
    variables = {
      POWERTOOLS_SERVICE_NAME      = "${var.resource_name_prefix}-products",
      POWERTOOLS_METRICS_NAMESPACE = "${var.resource_name_prefix}-products",
      # Below properties can help with debugging
      POWERTOOLS_LOGGER_LOG_EVENT = false,  # Logs incoming event
      POWERTOOLS_LOG_LEVEL        = "INFO", # Changes log level
      POWERTOOLS_DEV              = false   # Increases JSON indentation
    }
  }
```

A Terraform Module

Input (variables.tf)

```
variable "function_name" {
    type = string
}

variable "runtime" {
    type = string
    default = "nodejs20.x"
}

variable "log_retention_days" {
    type = number
    default = 14
}
```

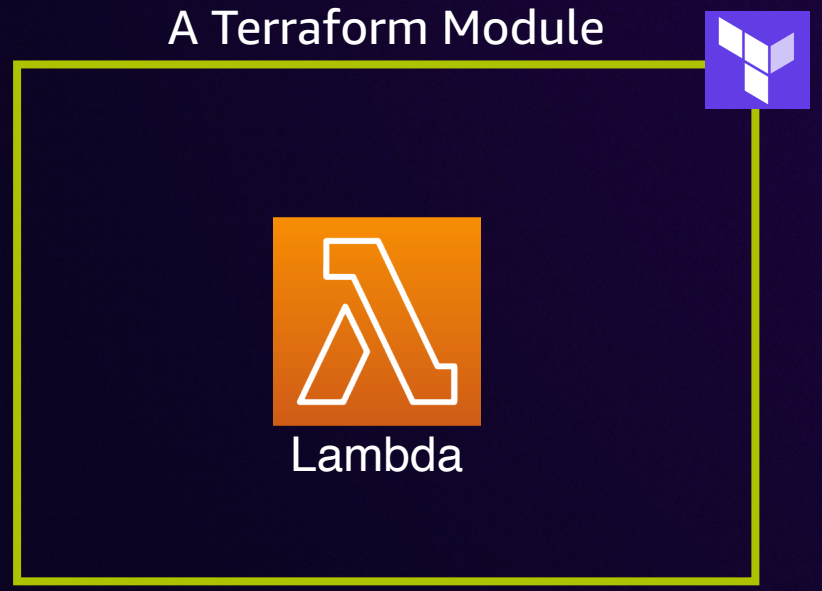
Cloud resources configuration (main.tf)

```
resource "aws_lambda_function" "lambda_function" {
  function_name    = var.function_name
  runtime          = var.runtime
  logging_config {
    log_group = aws_cloudwatch_log_group.this.arn
  }
}

resource "aws_cloudwatch_log_group" "this" {
  name              = "/aws/lambda/${var.function_name}"
  retention_in_days = var.log_retention_days
}

resource "aws_iam_policy" "this" {
    ....
}
```

A Terraform Module



Best Practices

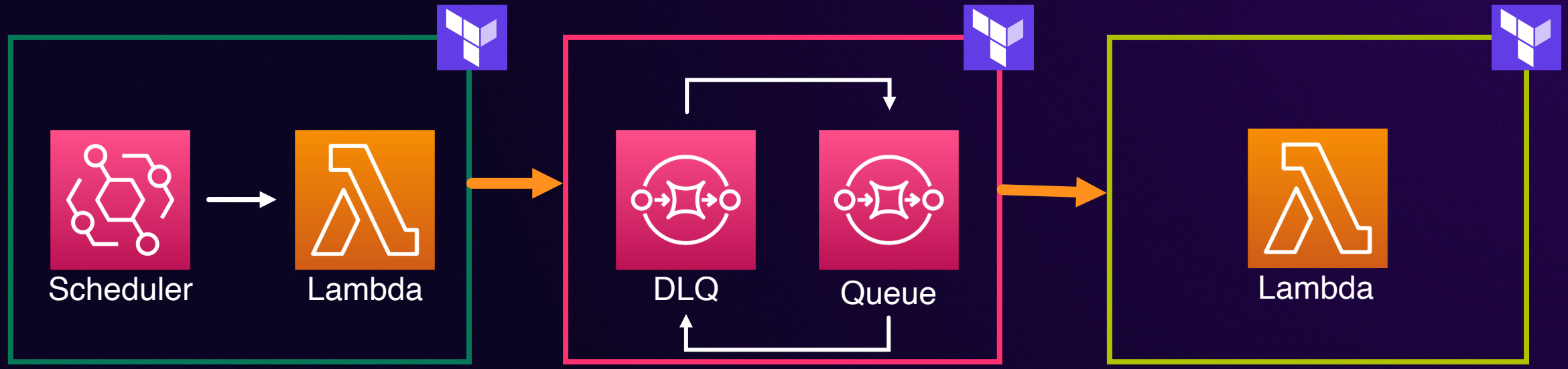
Variables

function_name
memory_size
source_path
runtime
handler
attach_vpc
log_retention_days
enable_furl
etc…

Outputs

function_arn
log_group_arn
function_url

Reusability and composability

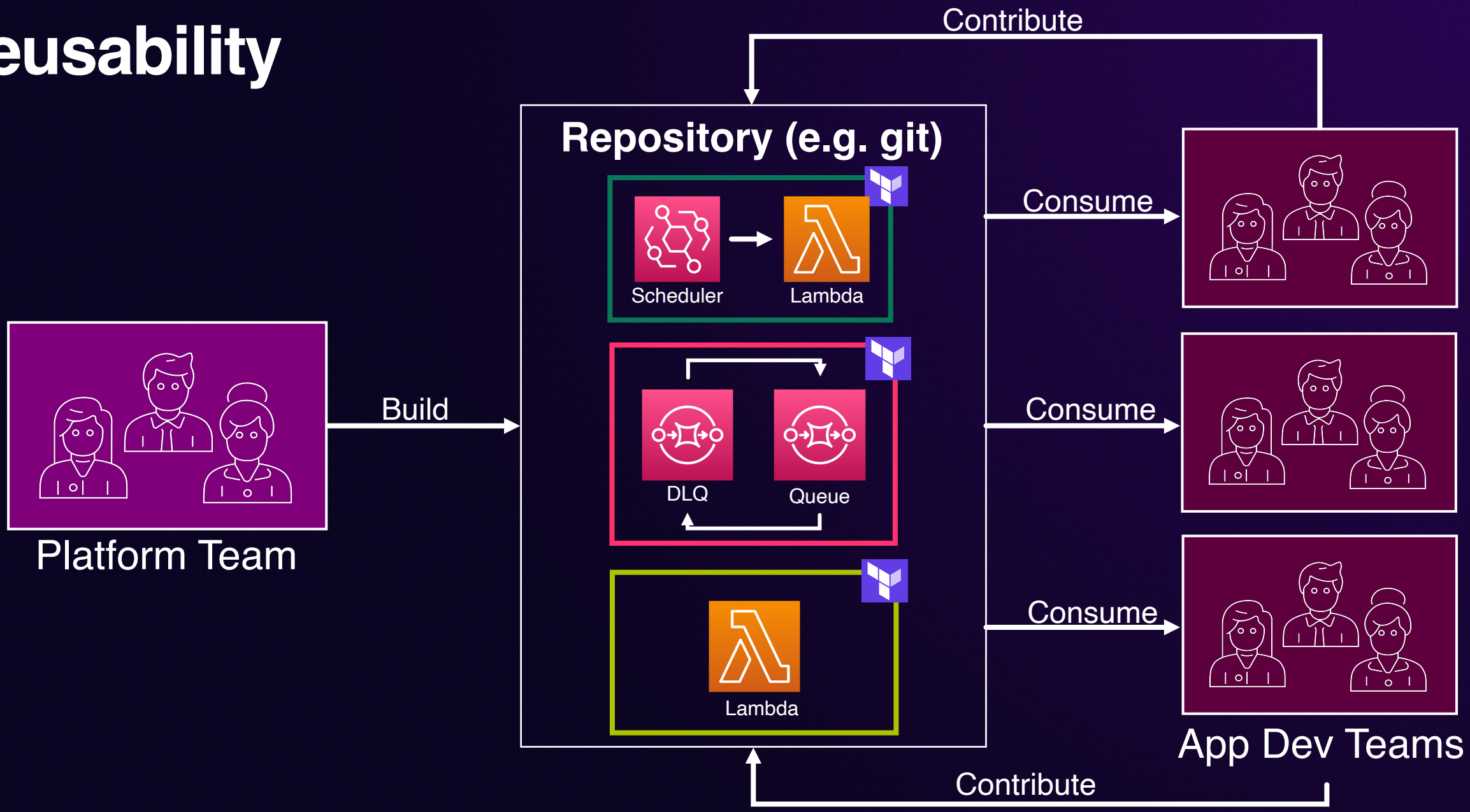


Lambda function with periodical scheduler

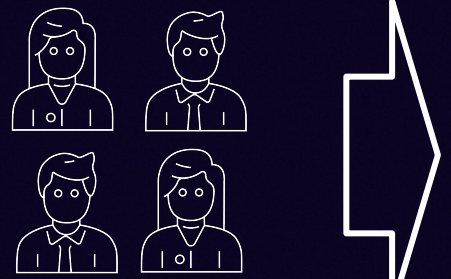
SQS Queue with redrive

Baseline Lambda function

Reusability



Evolve your IaC with Terraform Modules



Engineering teams

Terraform resource

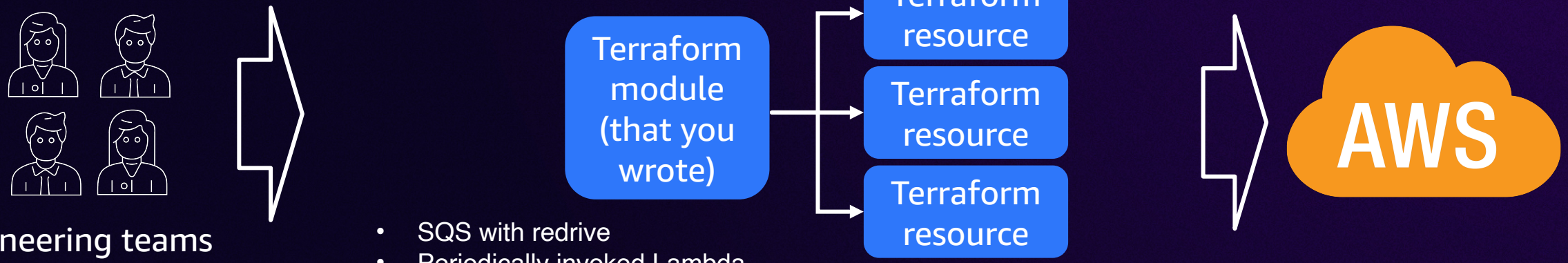
Terraform resource

Terraform resource

AWS

Maturity level: Beginner

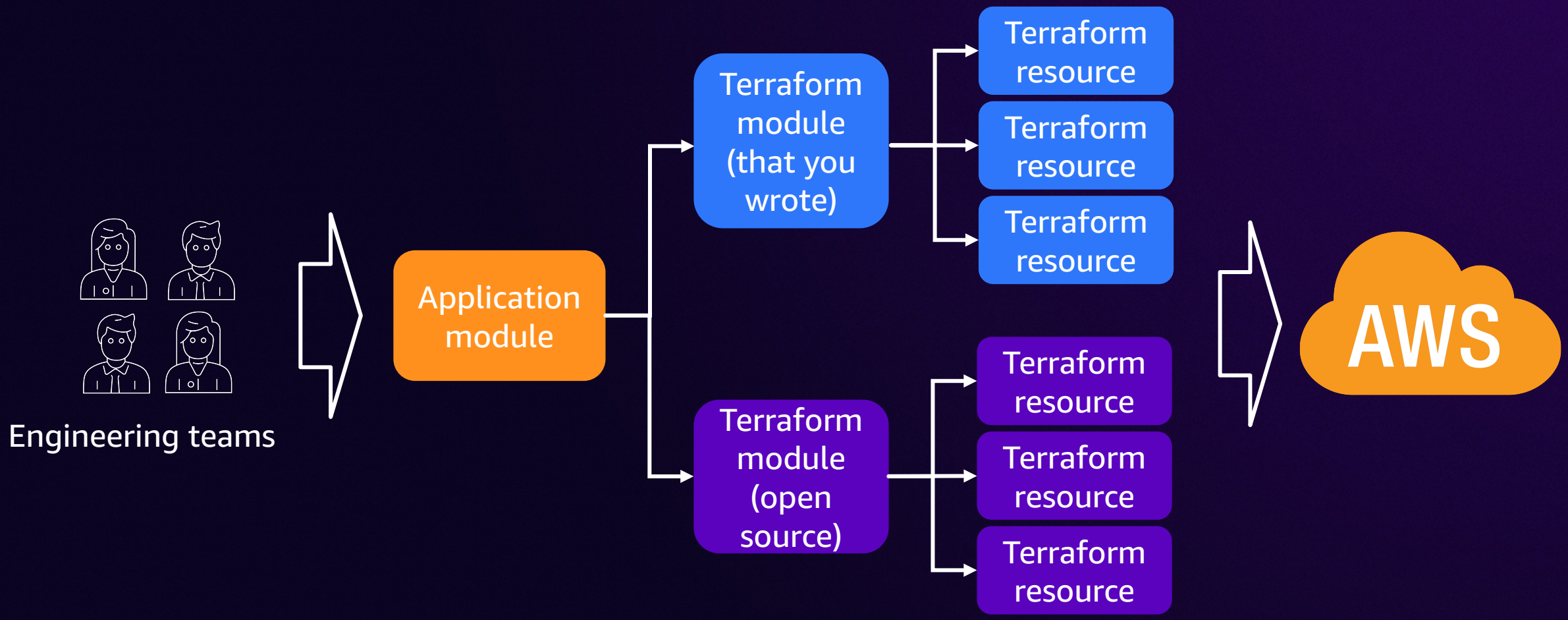
Evolve your IaC with Terraform Modules



- SQS with redrive
- Periodically invoked Lambda
- VPC-attached Lambda
- SQS+SNS with E2E encryption
- Many more…

Maturity level: Advanced

Evolve your IaC with Terraform Modules



Maturity level: Power User

Serverless.tf – community project

An opinionated, 100% open-source community framework for developing, building, deploying, and securing serverless applications on AWS using Terraform.

Over **120 million** downloads

https://serverless.tf

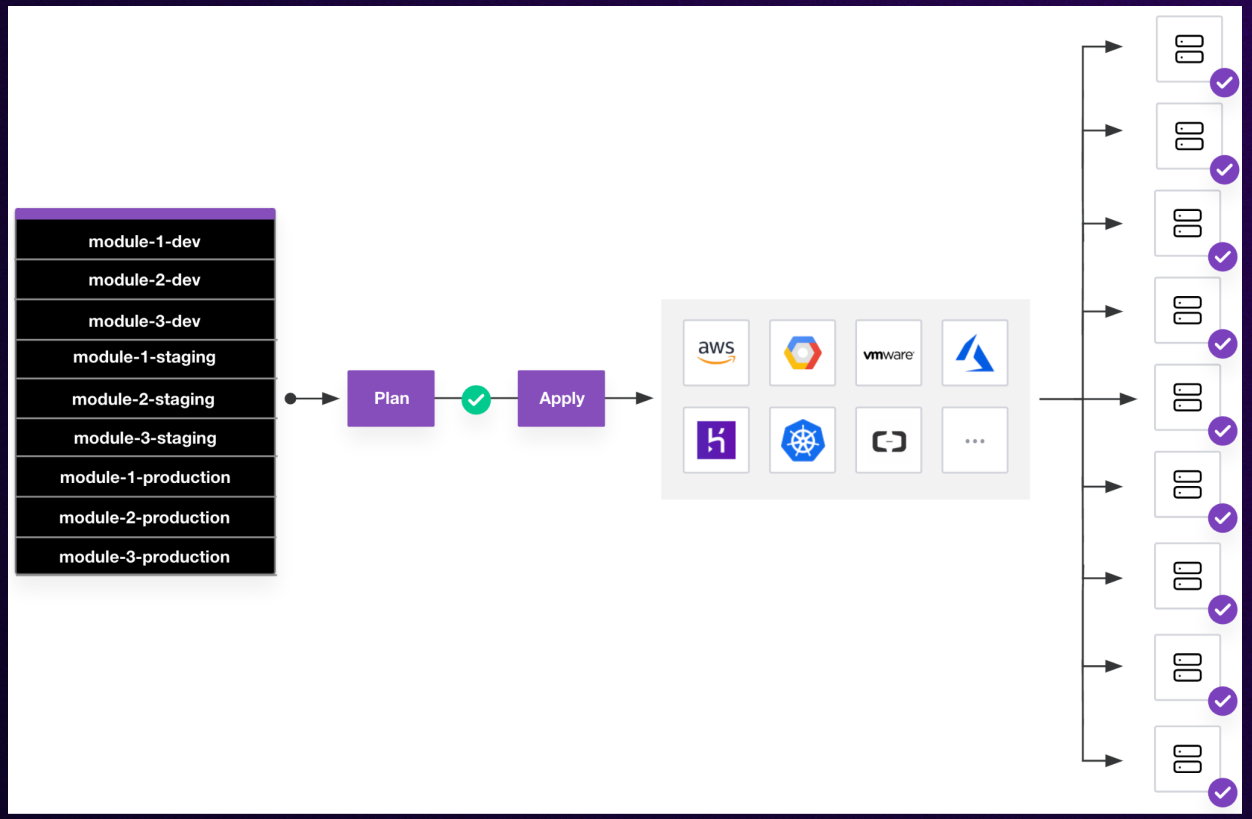
Consistent Deployments

The challenge – managing workspace complexity

HashiCorp recommends a **modular and composable** approach to structuring workspaces to optimize Terraform configurations for **speed, security, and maintainability**.

As organizations scale their Terraform usage with more complex workloads they begin to encounter challenges:

- Managing dependencies across **multiple workspaces**
- Repeating and managing **multiple instances of the same infrastructure**



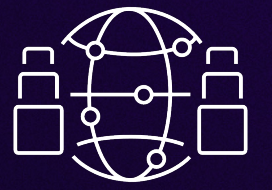
Common challenges

Deploy applications components reliably and consistently

Multiple teams

Multiple environments

Multiple accounts

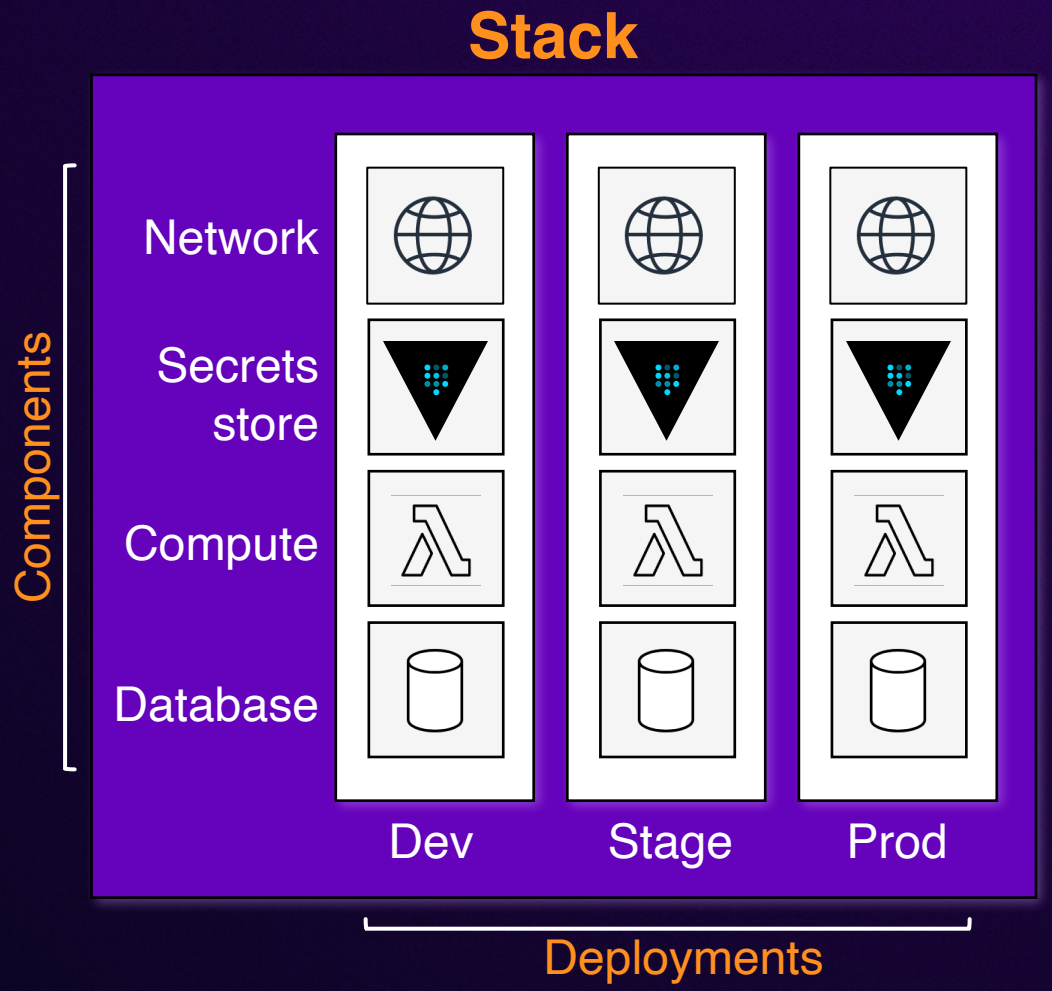


Multiple regions

Introducing Terraform Stacks

A new approach to provisioning and management of cloud application environments

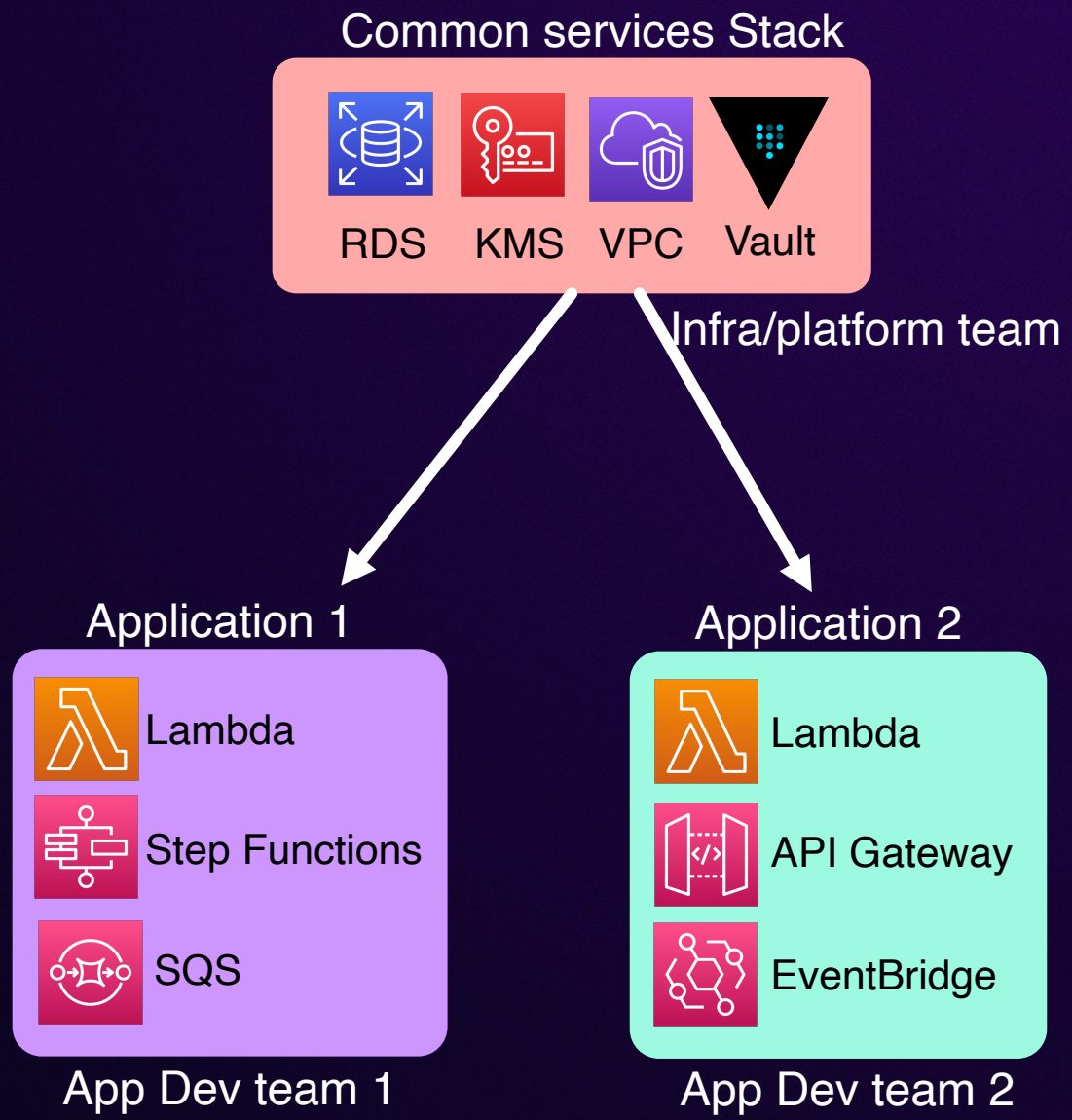
- Optimize the **coordination, provisioning, and management** of dependent Terraform configurations
- **Components**: standard Terraform modules with **dependencies defined in code**
- **Deployments**: used to repeat the components in the stack
- **Customize multiple deployments** with input variables, without duplicating code



Connecting Stacks via outputs

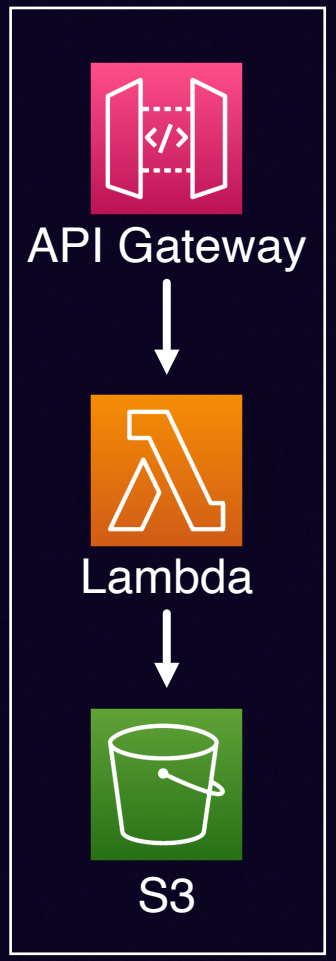
Separate stacks for based on persona

- **Outputs from one stack** deployment can be used as **inputs to another stack** deployment.
- **When an input in a downstream stack changes** as a result of an output from an upstream stack, **HCP Terraform automatically triggers plans** for any affected deployments.



Sample Serverless Application Stack

Stack Components



Deployment definitions

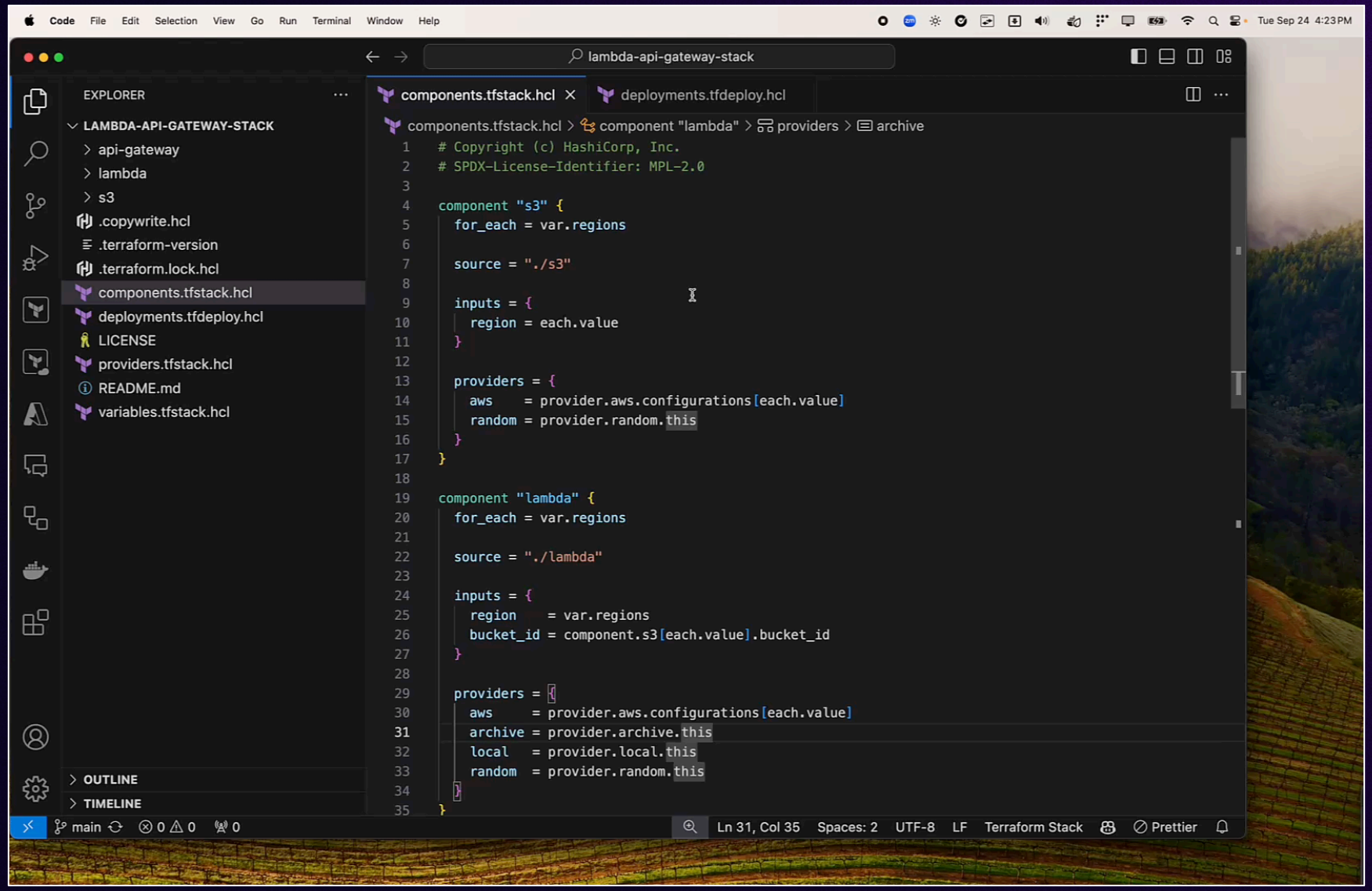
```
identity_token "aws" {
 audience = ["<Set to your AWS IAM assume-role audience>"]
}

deployment "development" {
 variables = {
   regions             = [”us-east-1"]
   role_arn            = "<Your dev AWS account IAM role ARN>"
   identity_token_file = identity_token.aws.jwt_filename
 }
}

deployment "production" {
 variables = {
   regions             = [”us-east-1", ”us-west-1"]
   role_arn            =  "<Your prod AWS account IAM role ARN>"
   identity_token_file = identity_token.aws.jwt_filename
 }
}
```

Deployments in AWS Cloud

Serverless Deployments with Terraform Stacks




```hcl
# Copyright (c) HashiCorp, Inc.
# SPDX-License-Identifier: MPL-2.0

component "s3" {
  for_each = var.regions

  source = "./s3"

  inputs = {
    region = each.value
  }

  providers = {
    aws    = provider.aws.configurations[each.value]
    random = provider.random.this
  }
}

component "lambda" {
  for_each = var.regions

  source = "./lambda"

  inputs = {
    region    = var.regions
    bucket_id = component.s3[each.value].bucket_id
  }

  providers = {
    aws     = provider.aws.configurations[each.value]
    archive = provider.archive.this
    local   = provider.local.this
    random  = provider.random.this
  }
}
```

Conclusion

1

Terraform + SAM enable you to efficiently build and test serverless applications with tools your organization already familiar with.

2

Modularizing your Terraform configurations and using existing OSS modules allows to build reusable components and patterns, so critical in the serverless world

3

Terraform Stacks and Workspaces help to streamline deployment of your serverless applications and resolve cross-team dependencies in a reliable way.

Next steps

Building Serverless Applications with Terraform Guide

Building Serverless Applications with Terraform Workshop

AWS Terraform Provider docs

Using AWS SAM with Terraform

Governance for Serverless Applications Guide

Sample serverless patterns with Terraform

Thank you!

Anton Aleksandrov
Pr. Solutions Architect, Serverless
AWS

Debasis Rath
Sr. Solutions Architect, Serverless
AWS

Gautam Baghel
Partner Solutions Engineer
HashiCorp