

The background features a dark navy blue field with abstract, overlapping organic shapes in shades of magenta and deep red. Two thin, light blue lines intersect diagonally across the upper right portion of the image.

AWS re:Invent

DECEMBER 2 – 6, 2024 | LAS VEGAS, NV

SVS320

Accelerate serverless deployments using Terraform with proven patterns

Anton Aleksandrov

Principal Solutions Architect, Serverless AWS

Anton Babenko

Core maintainer of Terraform AWS modules
Creator of serverless.tf



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

best practices
proven patterns
test applications locally

modular, reusable architectures
manage ownership
efficiently deploying
predictably, and repeatedly

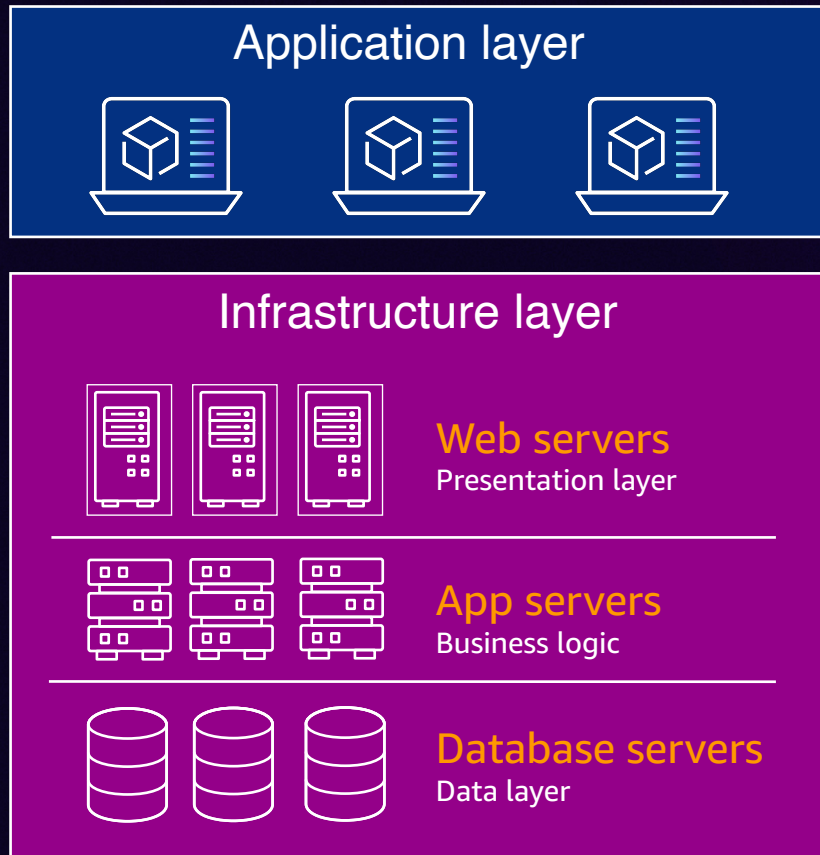
Leave equipped with practical skills
open source frameworks

Build and test

Scale and deploy

Action items

Traditional applications

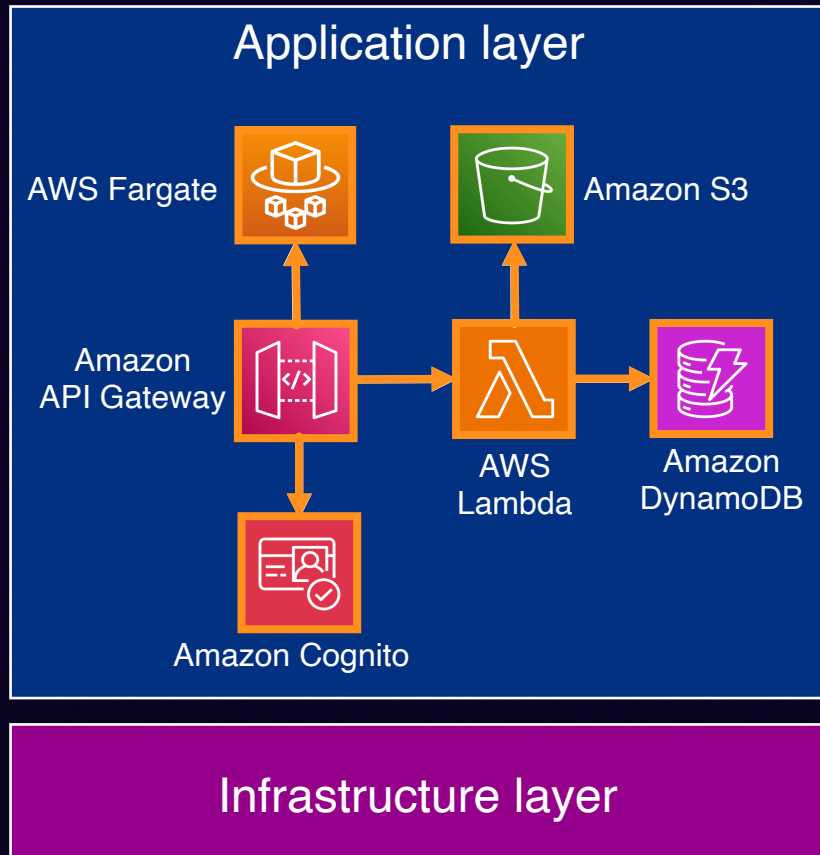


Rigid separation between infrastructure and application teams, tools, processes

Frequently **coupled, manual workflows** for release and quality control

Long cycles – need a new database?
Open a ticket, we'll get back to you

Serverless applications



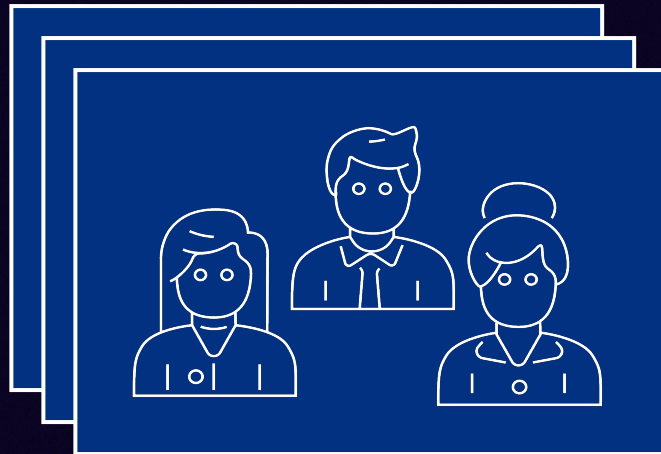
“Infrastructure” is redefined. A function, an event-source mapping, and an event routing rule are **app resources** owned by the app dev teams

IaC tools are no longer exclusive to Infra teams, but **commonly used by app dev teams**

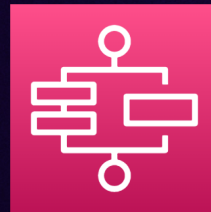
Infrastructure-as-code and **integration-as-code**

The ownership boundaries

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



Application development teams

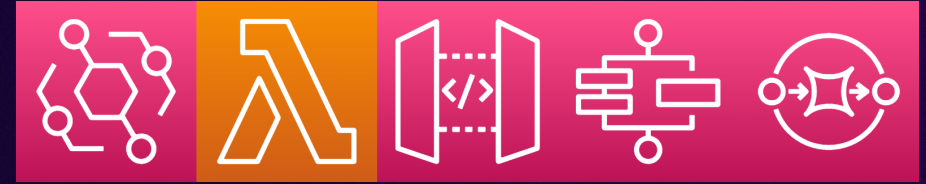
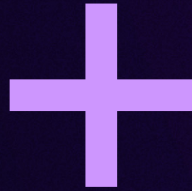


This is an **infrastructure resource**! We own and manage it!



Infrastructure/ops team

Building serverless applications with Terraform



AWS Serverless

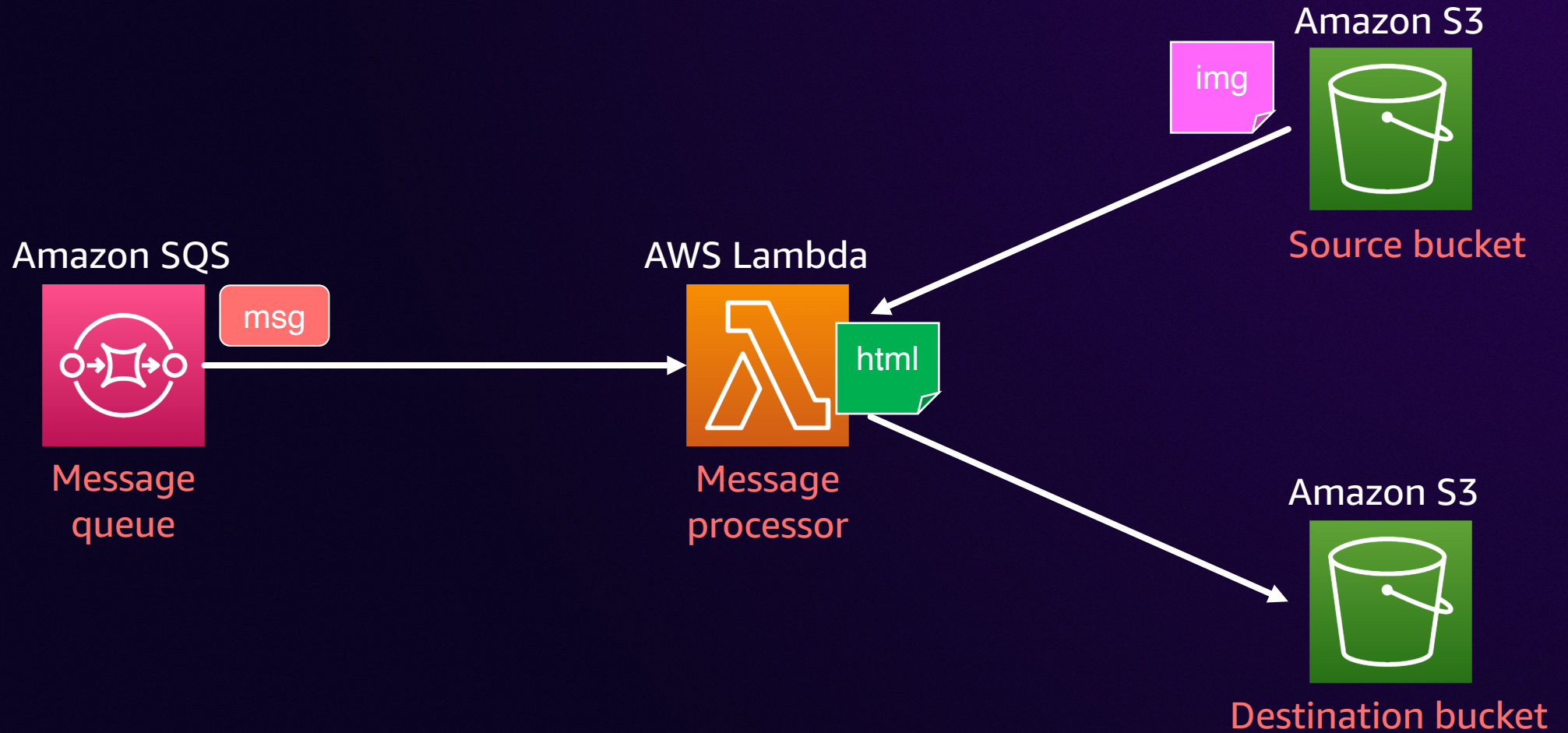
- Widely used IaC framework
 - 450M+ downloads, 4,350+ customers
 - Vast array of integrations, with support for over **3,000** providers
 - Terraform AWS Provider downloaded over **3 billion** times
- Run code without thinking about servers or clusters
 - Over **one million** customers using AWS Serverless services every month
 - Processes **10s of trillions** requests every month

Let's start building?



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What are we building



Creating SQS queues

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

Queue name

Security best practice

Creating S3 buckets

```
resource "aws_s3_bucket" "src_bucket" {  
  bucket = "SourceBucket"  
}
```

Bucket name

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

Ownership controls

```
resource "aws_s3_bucket_acl" "src_bucket" {  
  bucket = aws_s3_bucket.src_bucket.id  
  acl    = "private"  
}
```

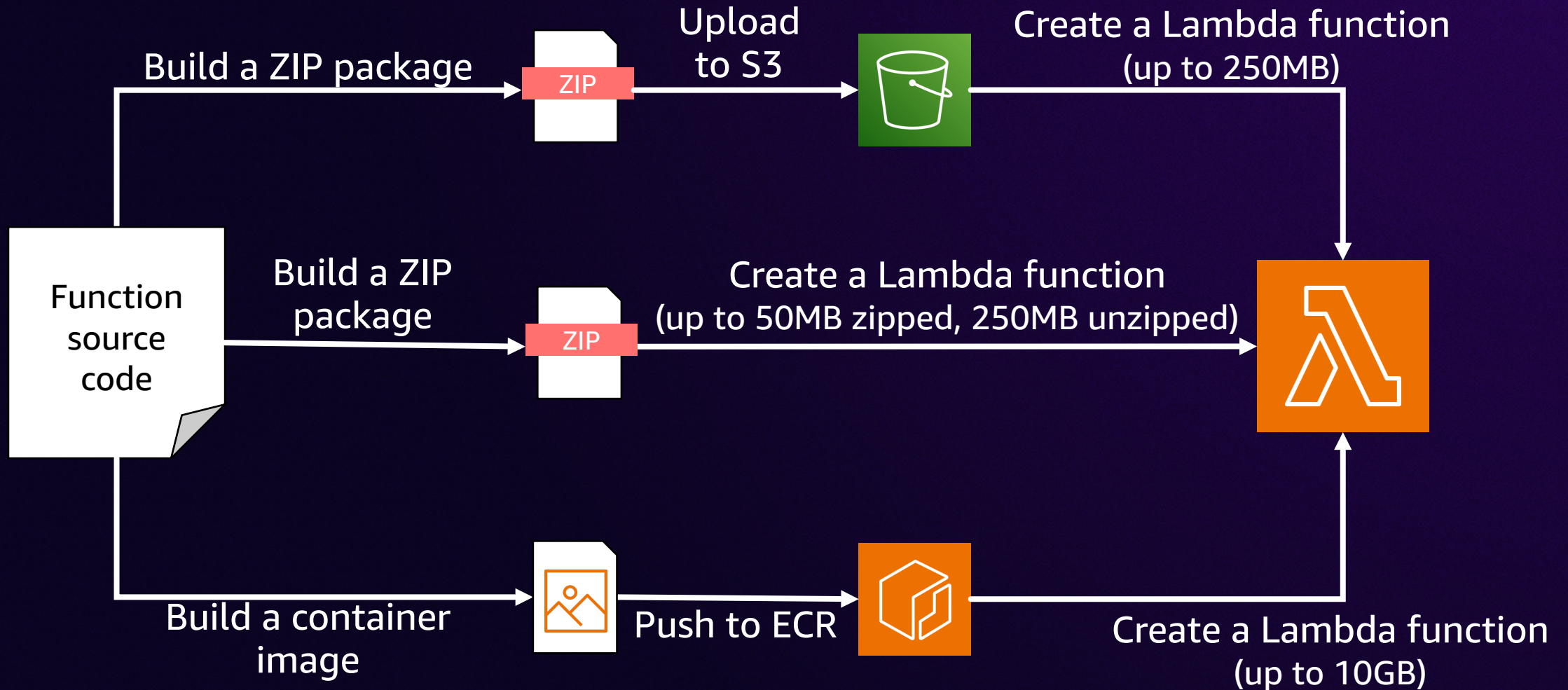
Access controls

Creating Lambda function execution role

```
resource "aws_iam_policy" "lambda_s3_access" {
  policy = jsonencode({
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["s3:GetObject"],
        Resource: "${aws_s3_bucket.src_bucket.arn}/*"
      },
      {
        Effect: "Allow",
        Action: ["s3:PutObject"],
        Resource: "${aws_s3_bucket.dst_bucket.arn}/*"
      },
    ],
  })
}
```

Principle of least privilege

Creating Lambda functions



Creating Lambda functions

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

Package source directory

```
resource "aws_lambda_function" "this" {  
  function_name = "greetings_lambda"  
  role          = aws_iam_role.this.arn  
  
  handler      = "index.handler"  
  runtime      = "nodejs20.x"  
  memory_size  = 256
```

Function configuration

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

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
```

Function configuration

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

Function code

```
  # Alternatively, if getting the zip file from S3  
  # s3_bucket = .....  
  # s3_key    = .....  
  
  #Alternatively, if using function image from ECR  
  # image_uri = .....
```

```
}
```


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
```

Function configuration

```
  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 = .....
```

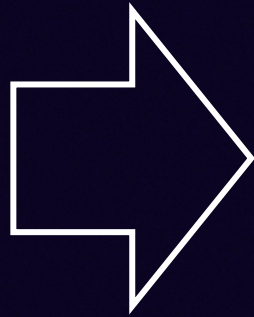
Function code

```
}
```

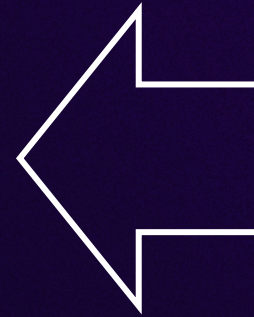

Testing serverless applications with SAM



Local development,
debugging, and testing



**Seamless
development
and testing
experience**



Cloud environment
templating and deployment

Local testing with AWS SAM

```
sam - "ip-172-31- x (+)
```

Admin:~/environment/serverless-tf-basic-app/terraform (master) \$ echo -e "\nAWS CLI Version:\n"; aws --version; echo -e "\n\nTerraform Version:\n"; terraform --version; echo -e "\n\nAWS SAM CLI Version:\n"; sam --version

AWS CLI Version:

aws-cli/2.17.47 Python/3.11.9 Linux/6.1.106-116.188.amzn2023.x86_64 exe/x86_64.amzn.2023

Terraform Version:

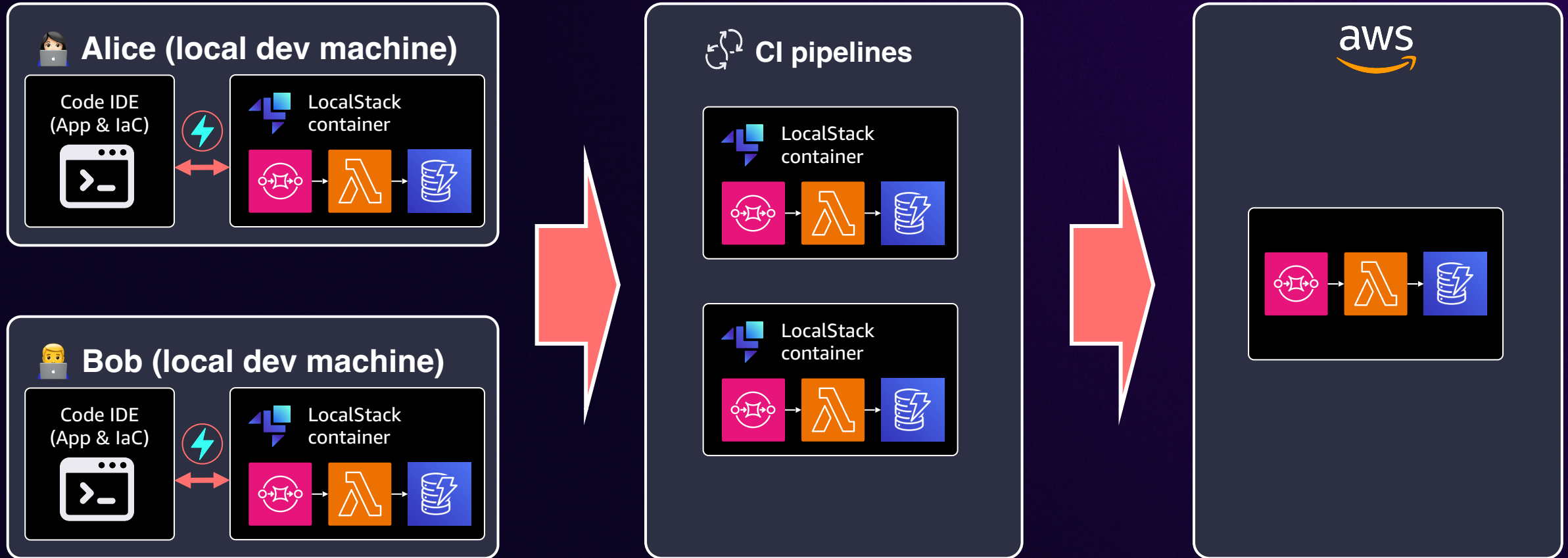
Terraform v1.9.5
on linux_amd64
+ provider registry.terraform.io/hashicorp/archive v2.4.2
+ provider registry.terraform.io/hashicorp/aws v5.54.1
+ provider registry.terraform.io/hashicorp/random v3.6.2

AWS SAM CLI Version:

SAM CLI, version 1.112.0

Admin:~/environment/serverless-tf-basic-app/terraform (master) \$ █

LocalStack – Test your serverless apps locally



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

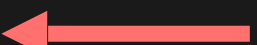
Principle of least
privileged access

Do not reinvent the wheel

Modularization

DON'T

```
38016
38017 resource "aws_iam_role_policy_attachment"
38018     policy_arn = aws_iam_policy.greeting_lambda_policy.arn
38019     role       = aws_iam_role.greeting_lambda_role.name
38020 }
38021
38022 resource "aws_lambda_event_source_mapping"
38023     event_source_arn = var.greeting_queue_arn
38024     function_name     = aws_lambda_function.greeting_lambda_function.name
38025     batch_size        = 1
38026
38027     depends_on = [aws_iam_role_policy_attachment.greeting_lambda_role_policy_attachment]
38028 }
38029
38030 # EOF
```



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

DO

Storage

Web Backend

Observability

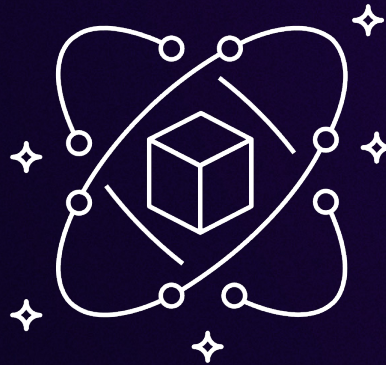
Async
processing

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

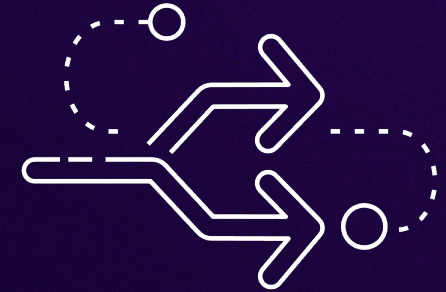
Modularization



Best practices



Reusability



Composability

A Terraform module

Input (variables.tf)

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

Output (outputs.tf)

```
output "function_arn" {  
    value = aws_lambda_function.lambda_function.arn  
}
```

Cloud resources configuration (main.tf)


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


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" "lambda_function" {  
  function_name = var.function_name
```

```
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

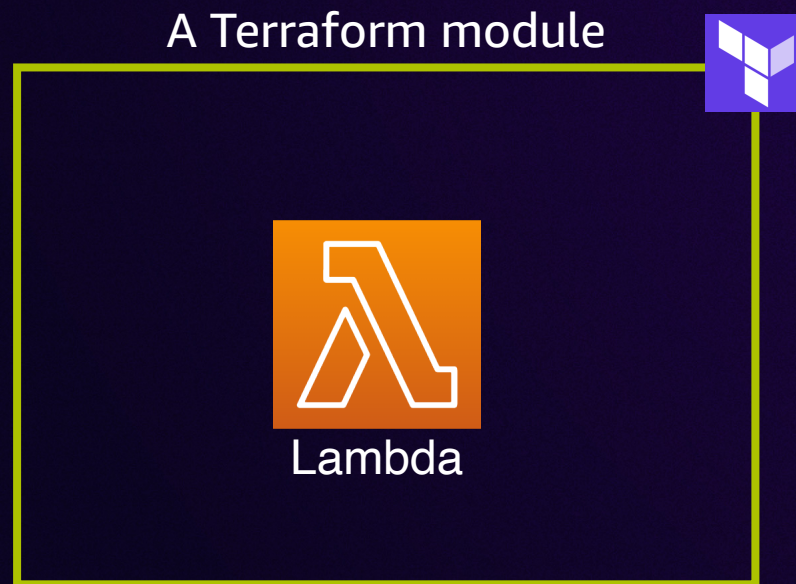
Input (variables.tf)

```
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...



A Terraform module



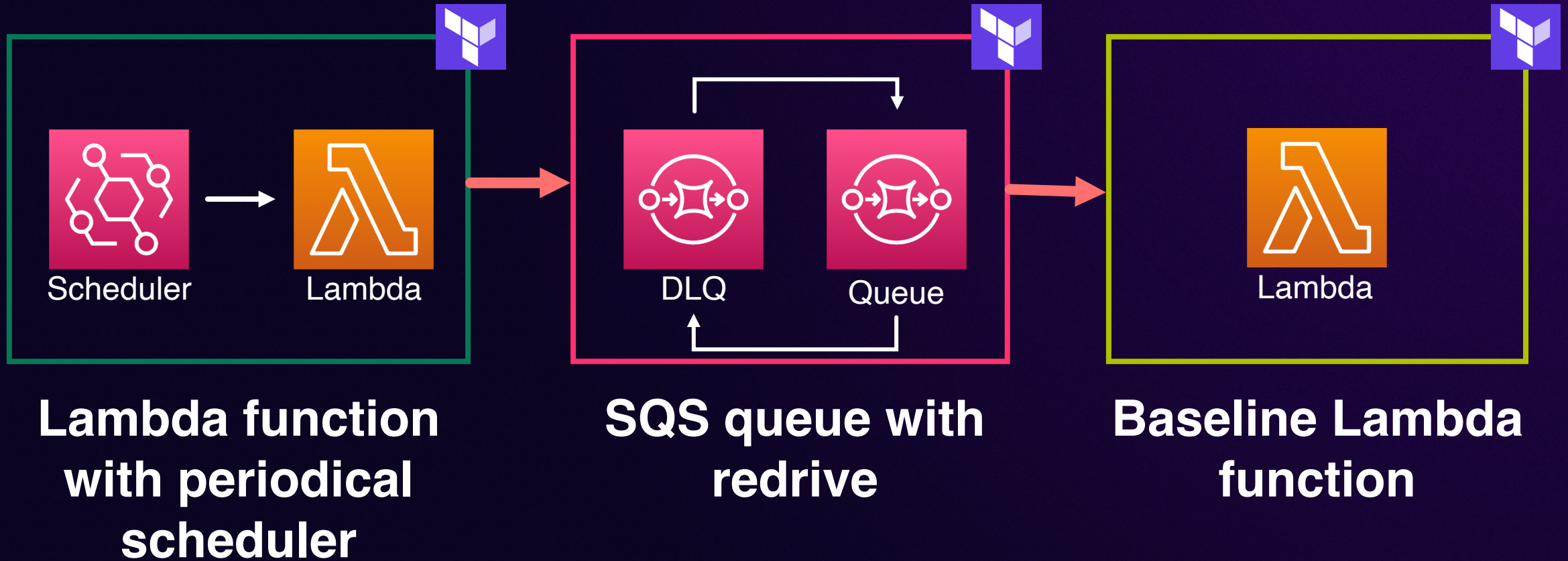
Lambda



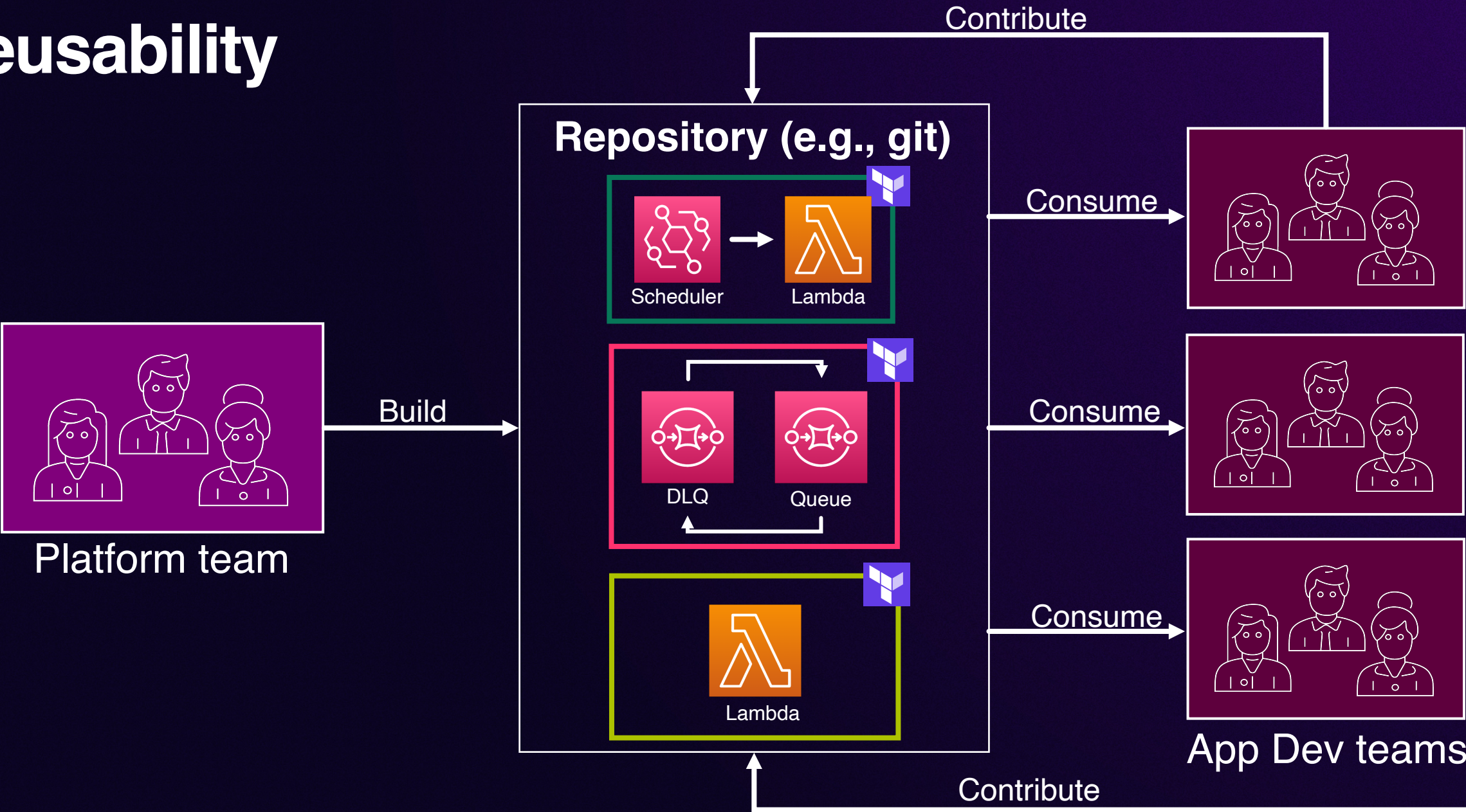
Outputs

function_arn
log_group_arn
function_url

Reusability and composability



Reusability



The journey to scale



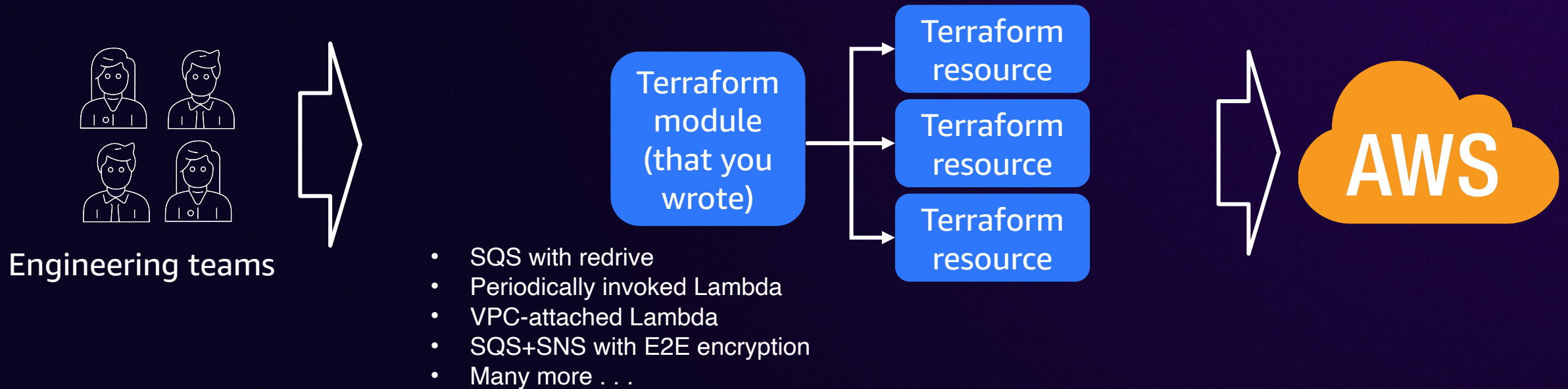
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Evolve your IaC with Terraform modules



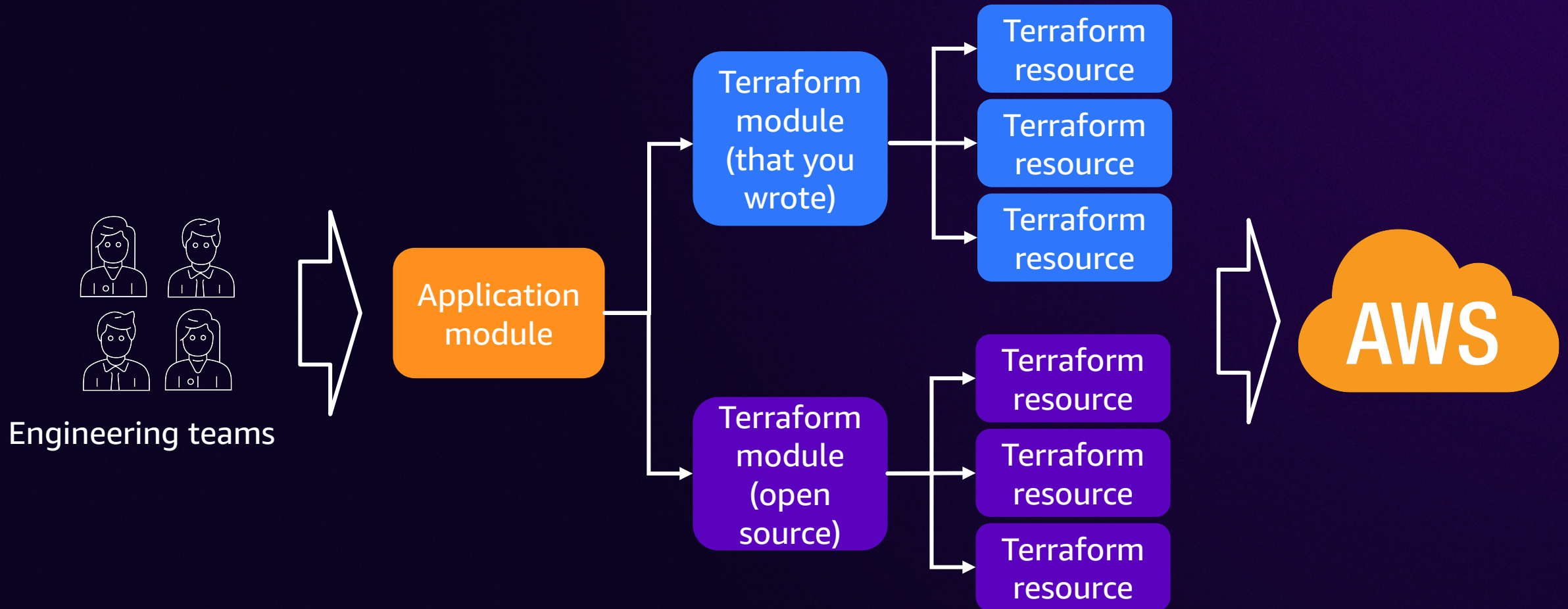
Maternity level: **Beginner**

Evolve your IaC with Terraform modules



Maturity level: Advanced

Evolve your IaC with Terraform modules



Maternity 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>

Supported AWS serverless services



- AWS Lambda
- AWS Lambda@Edge
- AWS AppSync
- Amazon EventBridge
- AWS Step Functions
- Amazon CloudFront
- Amazon API Gateway
- Amazon DynamoDB
- Amazon CloudWatch
- Amazon Aurora Serverless
- Amazon RDS Proxy
- Amazon S3
- Amazon SNS
- Amazon SQS
- AWS CodeDeploy
- AWS AppConfig
- AWS SSM Parameter Store
- AWS Secrets Manager

<https://serverless.tf>

Terraform-aws-lambda - packaging

Different Lambda runtimes have different packaging mechanisms

Build



pip install
poetry install



npm install



gradle install
maven install



cargo
lambda build

Deploy

UpdateFunctionCode

Terraform-aws-lambda – Packaging

Nodejs

```
module "nodejs_lambda_package" {  
  source = "terraform-aws-modules/lambda/aws"  
  version = "~> 7.0"  
  
  create_function = false  
  
  runtime      = "nodejs20.x"  
  source_path = "${path.module}/../src"  
  
  build_in_docker = true  
}
```

Python

```
module "python_lambda_package" {  
  source = "terraform-aws-modules/lambda/aws"  
  version = "~> 7.0"  
  
  create_function = false  
  
  runtime      = "python3.12"  
  source_path = "${path.module}/../src"  
  
  build_in_docker = true  
}
```

- Npm (package.json), PIP (requirements.txt), Poetry
- Dockerfile, Docker image, Docker with SSH agent

Terraform-aws-lambda – Packaging

Python with poetry

```
module "python_poetry_lambda_package" {  
  source  = "terraform-aws-modules/lambda/aws"  
  version = "~> 7.0"  
  
  create_function = false  
  
  runtime = "python3.12"  
  source_path = [  
    {  
      path          = "${path.module}/../python-app-poetry"  
      poetry_install = true  
    }  
  ]  
  
  build_in_docker = true  
  docker_image    = "build-python-poetry"  
  docker_file     = "${path.module}/../python-app-poetry/Dockerfile"  
}
```


Terraform-aws-lambda – Packaging

Rust

```
module "rust_lambda_package" {  
  source = "terraform-aws-modules/lambda/aws"  
  version = "~> 7.0"  
  
  create_function = false  
  
  handler      = "bootstrap"  
  runtime      = "provided.al2023"  
  architectures = ["arm64"]  
  
  trigger_on_package_timestamp = false  
  
  source_path = [  
    {  
      path = "${path.module}/../fixtures/runtimes/rust"  
      commands = [  
        "cargo lambda build --release --arm64",  
        "cd target/lambda/rust-app1",  
        ":zip",  
      ]  
      patterns = [  
        "!.*",  
        "bootstrap",  
      ]  
    }  
  ]  
}
```



www.cargo-lambda.info

- Any runtime, any architecture
- Custom build commands
- Watches for file changes

Local testing and hot reloading with LocalStack

```
module "lambda_function" {  
  source = "terraform-aws-modules/lambda/aws"  
  version = "~> 7.0"  
  
  function_name = "localstack-lambda"  
  handler       = "handler.lambda_handler"  
  runtime       = "python3.12"  
  architectures = ["arm64"]  
  
  create_package = false  
  
  s3_existing_package = {  
    bucket = "hot-reload"  
    key    = abspath("${path.module}/src")  
  }  
}
```

- 1 Install LocalStack wrappers for Terraform and AWS CLI

```
$ pip install terraform-local awscli-local  
$ tflocal init  
$ tflocal apply
```

- 2 Do not create a package, use LocalStack hot reloading
- 3 Invoke the function locally

```
$ awslocal lambda invoke \  
|   --function-name localstack-lambda
```


Cross-service integrations (without Serverless.tf)

- `aws_api_gateway_rest_api`
- `aws_api_gateway_resource`
- `aws_api_gateway_method`
- `aws_api_gateway_integration`
- `aws_api_gateway_deployment`
- `aws_iam_role`
- `aws_iam_policy_document`
- `aws_iam_policy`
- `aws_iam_role_policy_attachment`

- `archive_file`
- `aws_lambda_function`
- `aws_iam_role`
- `aws_iam_policy_document`
- `aws_iam_policy`
- `aws_iam_role_policy_attachment`



Cross-service integrations (with Serverless.tf)

```
module "api_gateway" {  
  source = "terraform-aws-modules/apigateway-v2/aws"  
  version = "~> 5.0"  
  
  name = "prod-api"  
  routes = {  
    "POST /customer" = {  
      integration = {  
        uri = module.lambda_customer.lambda_function_arn  
      }  
    }  
  }  
}
```








```
module "lambda_customer" {  
  source = "terraform-aws-modules/lambda/aws"  
  version = "~> 7.0"  
  
  function_name = "customer"  
  handler       = "customer.register"  
  runtime       = "nodejs20.x"  
  
  allowed_triggers = {  
    APIGateway = {  
      service      = "apigateway"  
      source_arn = "${module.api_gateway.api_execution_arn}/*/POST/customer"  
    },  
  }  
}
```



Additional integration patterns

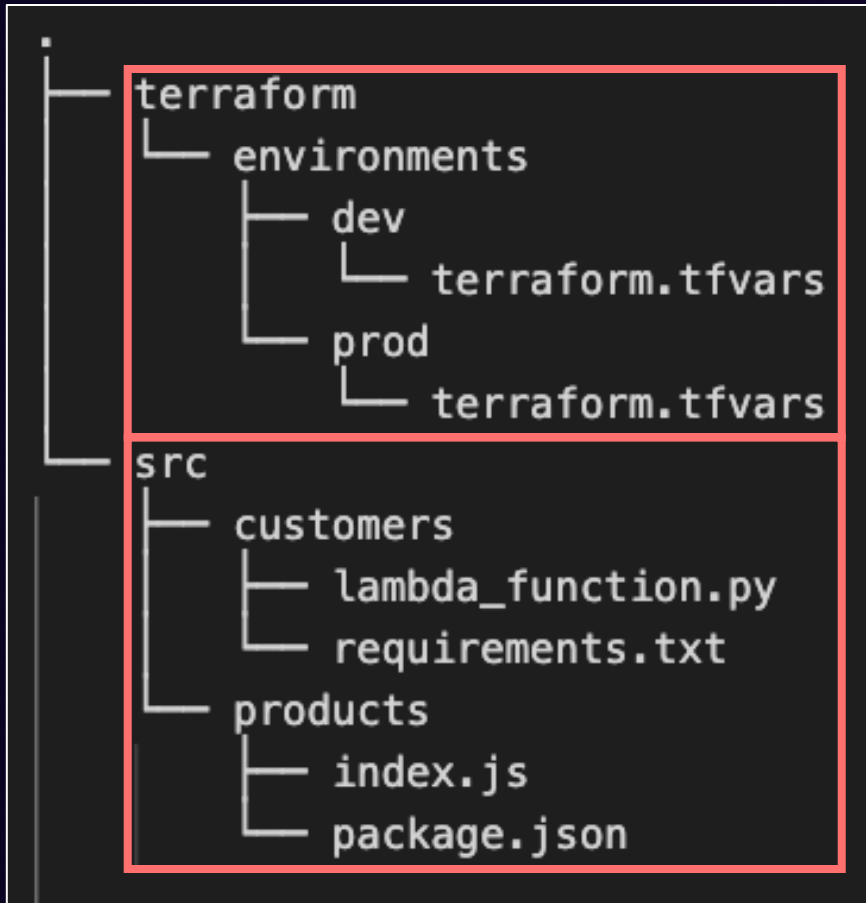


Serverless Land

 API Gateway	→	 AWS Lambda
<p>Amazon API Gateway (HTTP) to AWS Lambda</p> <p>This pattern creates an Amazon API Gateway (HTTP) and an AWS Lambda function.</p> <p>Terraform Python View pattern</p>		
 Amazon DynamoDB	→	 AWS Lambda
<p>Amazon DynamoDB Streams to AWS Lambda</p> <p>This pattern creates an Amazon DynamoDB table with streaming enabled, AWS Lambda function and setup event sourcing from DynamoDB to Lambda function.</p> <p>Terraform Python View pattern</p>		
 Amazon EventBridge	→ Schedule	 AWS Lambda
<p>Scheduled EventBridge rule for Lambda</p> <p>Create a scheduled EventBridge rule that invokes a Lambda function</p> <p>Terraform Python View pattern</p>		
 Lambda with Lambda Layers		
<p>Lambda Function and Lambda Layers</p> <p>Create a Lambda Function that has a Lambda Layer added to it</p> <p>Terraform Python View pattern</p>		

- API Gateway to Lambda
- DynamoDB streams to Lambda
- EventBridge scheduled rule for Lambda
- Lambda function with layers
- Execute Step Functions from Lambda
- S3 bucket notifications to Lambda
- S3 with S3 Object Lambda
- SQS to Lambda
- And more . . .

A project structure that works

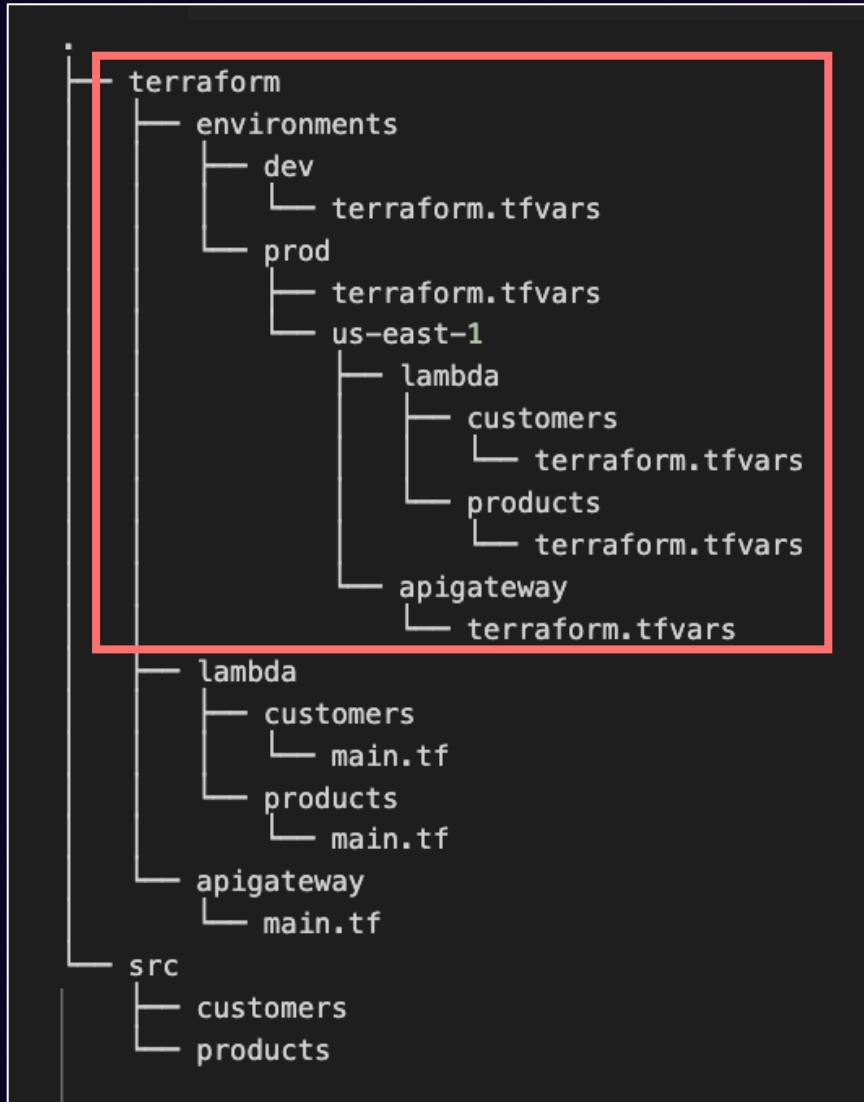


Per-project monorepo – IaC and application code are parts of the same project

Keep your project **modular**, IaC and application code go into separate directories

Each environment and each function goes into **its own separate directory**

A project structure that works



Use **dedicated variables file** for each planned environment

Add **fine-grained variables** if required, e.g. per region or per specific resource

Applying environment-specific variables

DON'T

```
terraform apply \  
  -var="runtime=nodejs20.x" \  
  -var="memory_size=512" \  
  -var="foo=bar" \  
  -var="baz=qux"
```

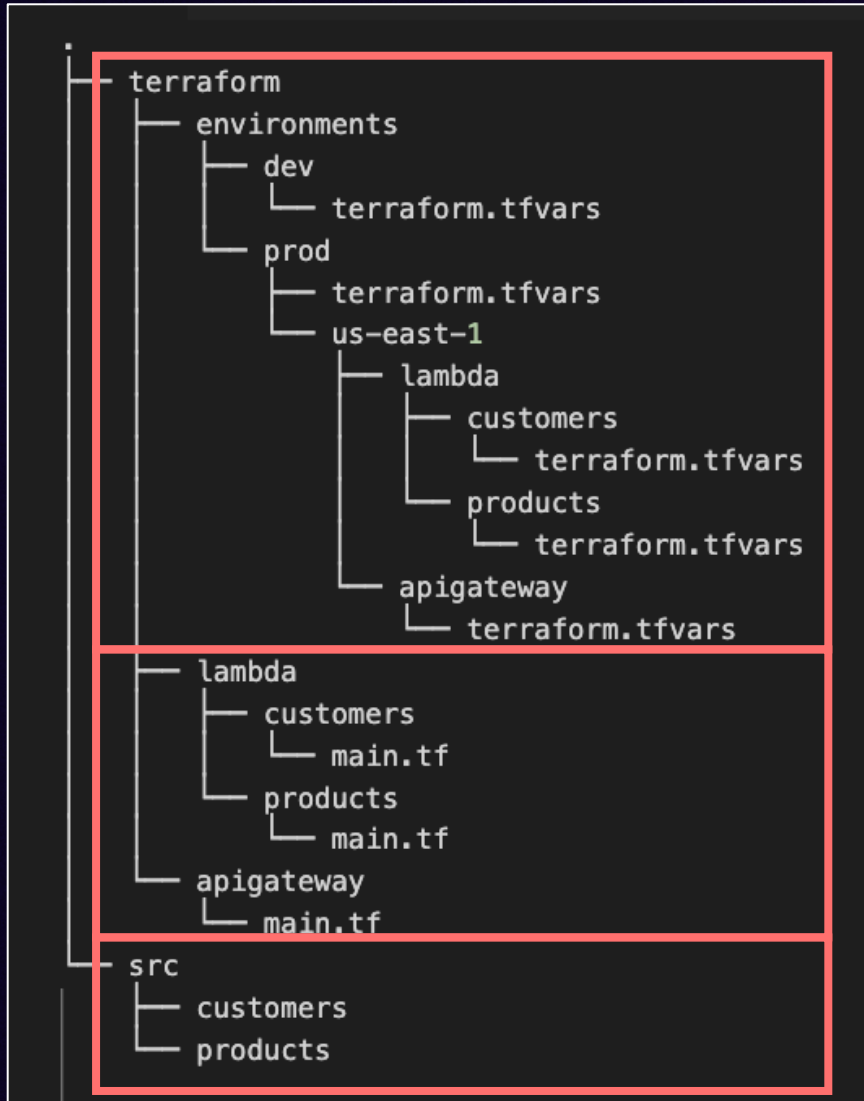
Manually supply each variable value

DO

```
terraform apply \  
  --var-file="envs/prod/terraform.tfvars" \  
  --var-file="envs/prod/us-east-1/terraform.tfvars"
```

Store environment specific variables under **dedicated directories**

A project structure that works

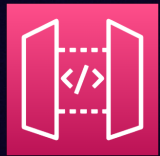


Match directory structure for environment values (.tfvars), resources (.tf), and source code

Prioritize using **versioned modules** instead of sourced locally

Use data sources (e.g. Terraform Remote State) to pass values between configurations

Managing dependencies between IaC configurations



API Gateway



Lambda



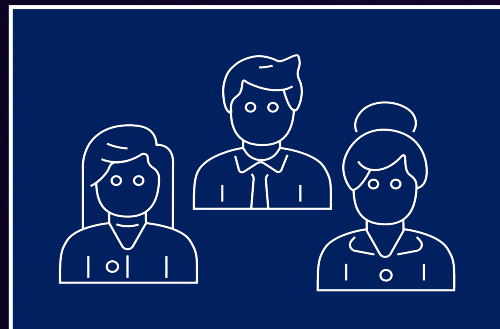
VPC



RDS



The Edge team



The App team



The Infra team

Managing dependencies between IaC configurations

DON'T

Hey dev team,

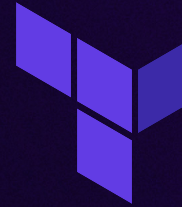
Send me your Lambda ARNs so I can attach them to the API Gateway

--

Bob | Infrastructure Team

Depend on manual
processes like emails or
Slack

DO



Use designated **dependency**
automation and management tools

Passing values between Terraform configurations

AWS Systems Manager Parameter Store



Infrastructure team IaC

```
resource "aws_ssm_parameter" "db_endpoint" {  
  name = "db_endpoint"  
  value = aws_db_instance.my_db.endpoint  
}  
  
resource "aws_db_instance" "my_db" {...}
```

AppDev team IaC

```
data "aws_ssm_parameter" "db_endpoint" {  
  name = "db_endpoint"  
}  
  
resource "aws_lambda_function" "my_function" {  
  environment {  
    variables = {  
      DB_ENDPOINT = data.aws_ssm_parameter  
                    .db_endpoint.value  
    }  
  }  
}
```


Terragrunt

/lambda/customers/terragrunt.hcl

```
terraform {
  source = "tfr:///terraform-aws-modules/lambda/aws//.?version=7.13.0"
}

inputs = {
  function_name = "customers"
  handler       = "index.lambda_handler"
  runtime       = "python3.12"

  source_path = jsonencode("${get_parent_terragrunt_dir()}/src/customers")
}
```

/apigateway/terragrunt.hcl

```
terraform {
  source = "tfr:///terraform-aws-modules/apigateway-v2/aws//.?version=5.2.0"
}

dependency "lambda_customers" {
  config_path = "../lambda/customers"
}

inputs = {
  name              = "registry-api-gateway"
  protocol_type     = "HTTP"
  create_domain_name = false

  routes = {
    "POST /customers" = {
      integration = {
        uri = dependency.lambda_customers.outputs.lambda_function_arn
      }
    }
  }
}
```


Terramate

/lambda/customers/stack.tm.hcl

```
stack {
  name = "customers"
  id   = "lambda_customers"
}

output "lambda_function_arn" {
  backend = "default"
  value   = module.lambda.lambda_function_arn
}

generate_hcl "_main.tf" {
  content {
    module "lambda" {
      source = "terraform-aws-modules/lambda/aws"
      version = "7.13.0"

      function_name = "customers"
      handler       = "index.lambda_handler"
      runtime       = "python3.12"

      source_path = "${terramate.stack.path.to_root}/terramate/
src/customers"
    }
  }
}
```

/apigateway/stack.tm.hcl

```
stack {
  name = "api-gateway"
  id   = "api-gateway"

  after = ["../lambda/customers"]
}

input "lambda_function_arn" {
  backend      = "default"
  from_stack_id = "lambda_customers"
  value        = outputs.lambda_function_arn.value
}
```

/apigateway/main.tf

```
module "api_gateway" {
  source = "terraform-aws-modules/apigateway-v2/aws"
  version = "5.2.0"

  name           = "registry-api-gateway"
  protocol_type  = "HTTP"
  create_domain_name = false

  routes = {
    "POST /customers" = {
      integration = {
        uri = var.lambda_function_arn
      }
    }
  }
}
```


Deploying at scale



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Common challenges

Deploy applications components reliably and consistently



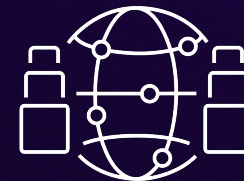
Multiple teams



Multiple environments

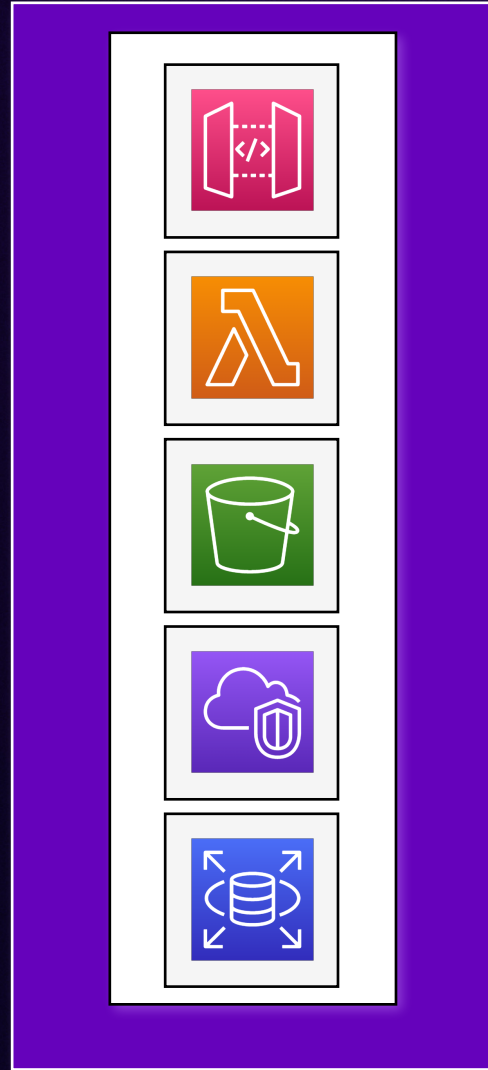


Multiple accounts



Multiple regions

Terraform stacks



Terraform stacks

Components

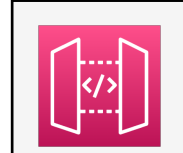
API
management

Compute

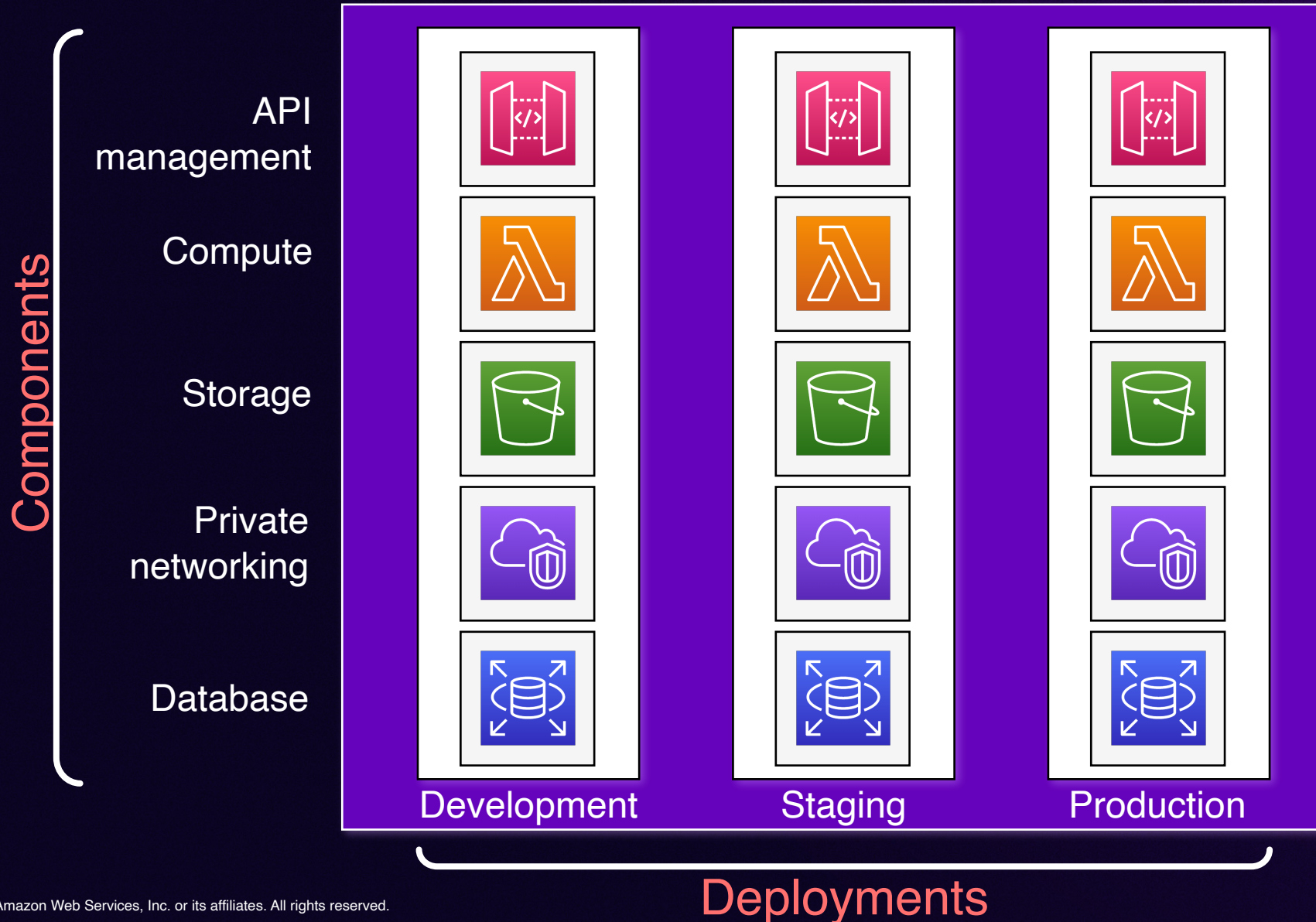
Storage

Private
networking

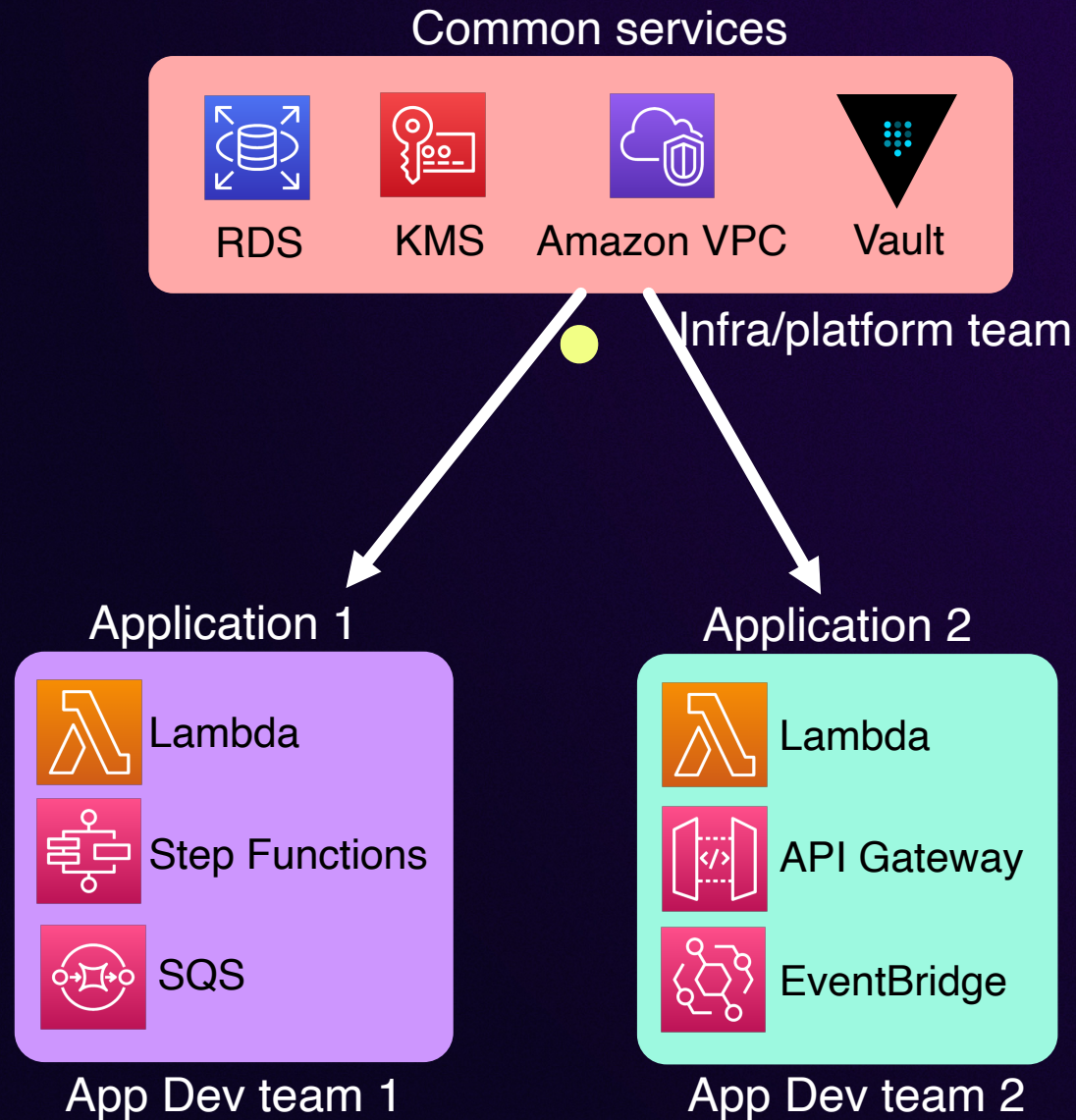
Database



Terraform stacks



Connecting stacks via outputs



Sample serverless application stack

Stack components



Deployment definitions

```
identity_token "aws" {
  audience = ["aws.workload.identity"]
}

deployment "development" {
  inputs = {
    regions      = ["us-east-1"]
    role_arn     = "<Your dev AWS account IAM role ARN>"
    identity_token = identity_token.aws.jwt
    default_tags = {
      environment = "development"
    }
  }
}

deployment "production" {
  inputs = {
    regions      = ["us-east-1", "us-west-1"]
    role_arn     = "<Your prod AWS account IAM role ARN>"
    identity_token = identity_token.aws.jwt
    default_tags = {
      environment = "production"
    }
  }
}
```

Deployments in AWS cloud



Serverless deployments with Terraform stacks

serverless-stack-01

Public

Pin

Unwatch 1

Fork 0

Star 0

main 1 Branch 0 Tags

Go to file

Code

aal80 Update main.tf 18cf408 · 4 minutes ago 71 Commits		
apigateway	Update main.tf	35 minutes ago
lambda	Update main.tf	4 minutes ago
s3	Update main.tf	2 hours ago
src	Update index.js	34 minutes ago
.terraform-version	Create .terraform-version	2 days ago
.terraform.lock.hcl	Create .terraform.lock.hcl	2 days ago
README.md	Initial commit	2 months ago
components.tfstack.hcl	Update components.tfstack.hcl	22 minutes ago
deployments.tfdeploy.hcl	Update deployments.tfdeploy.hcl	23 minutes ago
providers.tfstack.hcl	Create providers.tfstack.hcl	2 days ago
variables.tfstack.hcl	Create variables.tfstack.hcl	2 days ago

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages



Conclusion

1

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

2

Modularizing your Terraform configurations allows you to build reusable components and patterns, which are critical in the serverless world

3

Use OSS community projects and samples provided by AWS to accelerate your Serverless+Terraform journey

Next steps



- These slides
- Building Serverless Applications with Terraform – **a guide and a workshop**
- AWS Terraform Provider **best practices**
- **Terraform resources** by Anton Babenko
- **Governance** for Serverless Applications Guide
- Sample **serverless patterns** with Terraform
- And more

<https://aal80.github.io/reinvent2024-svs320/>

Check out these other sessions

SVS337

Building Serverless Applications Using Terraform – Workshop
Monday (today), 3:00 PM – Mandalay Bay Lagoon F

SVS401

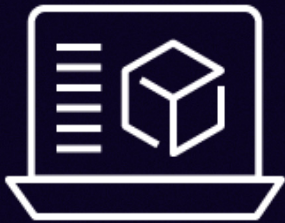
Best Practices for Serverless Developers
Monday (today), 3:00 PM – Venetian Lido 3002

SVS324

Implementing Security Best Practices for Serverless Applications
Wednesday (Dec. 4), 10:30 AM – MGM Grand 122

Continue your AWS serverless learning

Learn at your
own pace



Expand your serverless
skills with our learning plans
on **AWS Skill Builder**

Increase your
knowledge



Use our **Ramp-Up Guide**
to build your serverless
knowledge

Earn AWS
Serverless badge



Demonstrate your
knowledge by achieving
digital badges




<https://s12d.com/serverless-learning>

Thank you!

Anton Aleksandrov

 antonal80

Anton Babenko

 antonbabenko



Please complete the session survey in the mobile app