Using AWS Lambda with Substrate

Document Level Classification

200

Introduction

AWS Lambda provides the "serverless" paradigm, an alternative deployment method compared to scheduled workloads via Kubernetes, or traditional server-based workloads in EC2.

Common use-cases for Lambda include periodic jobs, programs that primarily interface with the AWS API or AWS events, and systems that are deployed per AWS account or per region regardless of whether a Kubernetes cluster is present.

Building Lambdas

Lambda functions are typically packaged as zip files and published to S3. You can achieve this workflow using CodeFresh.

You can take a look at Substrate Vault Healthcheck to see how codefresh.yml is configured to build and release a versioned zip file via S3.

Deploying Lambdas

You can deploy your Lambda function with Terraform. This allows you to control the version that's deployed independently of your build. Substrate Vault Healthcheck is a working example of a lambda project managed with Terraform.

If you prefer to have your Lambda automatically released when new versions are built you can configure Codefresh to update the deployed Lambda version using the AWS CLI. See Using Lambda with the AWS CLI for more info. It helps if you have already deployed the first version with Terraform since there will be fewer things to configure.

Accessing Secrets from Lambda

Recommended way for secrets retrieval from Substrate Vault would be using Vault AWS Lambda Extension, extension git. This setup relies on established network access between your Lambda and Substrate Vault, for this - see Reaching Epic Internal Services.

Additionally your Lambda should have AWS IAM role attached(assumed), e.g. see Terraform Example. Last thing you will need is to get Vault auth role for your Lambda(aka VAULT_AUTH_ROLE), please reach out to us in #cloud-ops-support-ext to get it created.

You can cut a CLD Jira issue to get yourself Vault auth role, please provide us combination of your Lambda AWS IAM role arn and required policy. (Read more about Vault Policy https://confluence-epicgames.atlassian.net/wiki/spaces/CE/pages/93487474)

As a workaround approach - Lambda accepts configuration using environment variables. Environment variables are encrypted using AWS KMS. By default Lambda uses the default KMS key for an account, but you should create your own key, which you can use to control access to the secrets stored in Lambda's environment.

Networking

Normally, Lambda traffic originates from a public IP on the internet, based on whatever infrastructure AWS places your Lambda on. These public IPs **cannot** reach Epic's internal service endpoints such as Substrate Vault.

Reaching Epic Internal Services

If your lambda needs to reach Epic internal services it will need to do so via the Service Network. In order for your Lambda to reach other Epic internal services, the first step is to attach a VPC network interface. This places your Lambda's network traffic inside Epic's network instead of on the public internet. This traffic will egress from your VPC through the Service Network NAT gateway.

When reaching services running in Substrate Kubernetes clusters, you will use ingress annotations to whitelist your lambda's gateway IPs. This can be done by following the instructions in Managing inbound traffic to your application. If you are connecting to a service running in EC2, Lambda, or otherwise not using Kubernetes ingress controller, you will need to coordinate with the service owner to whitelist your Service Network NAT gateway IPs, if they are not already allowing all Service Network traffic.

Enabling Epic Services to Reach your Lambda

If you are running a lambda with a REST API or other web endpoint and want internal services to reach your lambda, you will need to follow the whitelisting process in reverse. You create an API gateway or ALB for your lambda in the Service Network subnets, and then whitelist the Service Network, Office and VPN address ranges. Reference the Service Network Architectural Diagram to see a graphical representation of the Service Network and where the load balancer or API should reside.

In Kubernetes infrastructure, ingress annotations manages security groups attached to ALBs created via the Kubernetes ingress controller. In the case of Lambda, ingress annotations do not manage your security groups so you will need to do this yourself using Terraform or a similar tool.

Terraform Example

The following example shows how to attach a VPC network interface to your Lambda function. You can see a complete, working example in the Substrate Vault Healthcheck project.

```
# Allow lambda (the AWS service) to assume an IAM role in our account
resource "aws_iam_role" "vault-healthcheck-role" {
  name = "vault-healthcheck-role"

  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sts:AssumeRole"
      ],
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
EOF
}

# This policy allows the lambda function to run inside a VPC which
# requires permissions to create, delete, and describe network interfac
resource "aws_iam_policy" "vault-healthcheck-policy" {
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action":[
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterface"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
EOF
}

# Attach the logging & network interface policy to the lambda's IAM rol
resource "aws_iam_role_policy_attachment" "vault-healthcheck" {
  role = "vault-healthcheck-role"
  policy_arn = aws_iam_policy.vault-healthcheck-policy.arn
}
```

```
# Create (and upload) the lambda function
resource "aws_lambda_function" "vault-healthcheck" {
  function_name = "vault-healthcheck"
  role = aws_iam_role.vault-healthcheck-role.arn
  ...
  vpc_config {
    security_group_ids = [aws_security_group.vault-healthcheck-outbound
    subnet_ids = [
      # Note: replace these with your VPC Service Network NAT subnets t
      "subnet-0440e459ecbf7cfd1",
      "subnet-0398712f3b19fd404",
      "subnet-0b088586324d25dc1",
    ]
  }
}
```

Page Information:
Page ID: 81068328
Space: Cloud Developer Platform
Downloaded: 2025-07-12 04:08:03