


How to create Private DynamoDB tables accessible only within a VPC



Yan Cui · Follow

Published in theburningmonk.com · 4 min read · Jan 10, 2024

61



DynamoDB is a fully managed NoSQL database service known for its low latency and high scalability.

Like most other AWS services, it's protected by AWS IAM. So, despite being a publically accessible service, your data is secure. Conversely, zero-trust networking tells us we should always authenticate the caller and shouldn't trust someone just because they're in a trusted network perimeter.

All this is to say that you don't need network security to keep your DynamoDB data safe. However, adding network security on top of IAM authentication and authorization is not a bad thing. Sometimes it's even necessary to meet regulatory requirements or to keep your CSO happy!

This has come up numerous times in my consulting work. In this post, let me show you how to make a DynamoDB table accessible only from within a VPC.

Can't you just use a VPC endpoint?

You can create a VPC endpoint for DynamoDB to create a tunnel between your VPC and DynamoDB. This allows traffic to go from your VPC to DynamoDB without needing a public IP address.

This is often perceived to be more secure "because it doesn't go out to the public internet". But as far as I know, traffic from a VPC to DynamoDB (through a NAT Gateway) would never traverse over public internet infrastructure anyway. It's just that you need a public IP address so DynamoDB knows where to send the response.

More importantly, a VPC endpoint doesn't stop other people from accessing the DynamoDB table from outside your VPC.

No resource policy

Usually, this can be done using resource policies. Unfortunately, DynamoDB doesn't support resource policies [1].

I wish DynamoDB supported resource policies.

aws:SourceVpce

Instead, you can use the `aws:SourceVpc` or `aws:SourceVpce` conditions [2] in your IAM policies. You can use these conditions to ensure the IAM user/role can only access DynamoDB from a VPC, or through a VPC endpoint.

aws:SourceVpc
Works with [string operators](#).
Use this key to check whether the request comes from the VPC that you specify in the policy. In a policy, you can use this key to allow access to only a specific VPC. For more information, see [Restricting Access to a Specific VPC in the Amazon Simple Storage Service User Guide](#).

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.
- **Value type** – Single-valued

aws:SourceVpce
Works with [string operators](#).
Use this key to compare the VPC endpoint identifier of the request with the endpoint ID that you specify in the policy. In a policy, you can use this key to restrict access to a specific VPC endpoint. For more information, see [Restricting Access to a Specific VPC Endpoint in the Amazon Simple Storage Service User Guide](#).

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.
- **Value type** – Single-valued

But how we can ensure that everyone follows this requirement?

As a member of the platform team, I don't want to be a gatekeeper. I want the feature teams to create their own IAM roles for their Lambda functions or containers. But how can I make sure they don't violate our compliance requirements?

There are a few ways to do this. The easiest and most reliable is to use Service Control Policies (SCPs) with AWS Organizations.

Service Control Policies (SCPs)

SCPs let you apply broad strokes to deny certain actions in the member accounts (but *NOT* the master account) of an AWS organization. For example, you can use SCPs to:

- Stop any actions in regions other than us-east-1 or eu-west-1, where your application resides.
- Stop anyone from creating EC2 instances.

Both are common SCPs people use to protect against cryptojacking.

In our case, we can use a SCP like this to reject any requests to DynamoDB, unless they originate from within the specified VPC.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyDynamoDBOutsideVPC",
      "Effect": "Deny",
      "Action": "dynamodb:*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-xxxxxxxx"
        }
      }
    }
  ]
}
```

Every AWS account would have its own VPC(s), so you will likely need an account-specific SCP.

This SCP can also create significant friction to development.

Every action against DynamoDB tables, including the initial deployment, has to be performed from within the designated VPC. And you won't even be able to use the DynamoDB console!

A good compromise is to leave the SCP off the development accounts. That way, developers are not hamstrung by this strict SCP and you can still enforce compliance in production.

Supporting break-glass procedures

What if you have an emergency and someone needs to access the production tables through the AWS console?

Many organizations would lock down their production environment such that no one has write access. But they'd have a break-glass procedure that allows someone to temporarily assume a special role in the event of an emergency.

You can cater for these emergencies by carving out an exception in the SCP. For example, by adding the following condition:

```
"StringNotLike": {
  "aws:PrincipalArn": "arn:aws:iam::*role/your-emergency-role"
}
```

This condition allows the specified role to access DynamoDB from outside the VPC. This includes the ability to use the DynamoDB console.

How would you implement this break-glass procedure? That's another post for another day, perhaps :-)

In the meantime, if you want to learn more about building serverless applications on AWS, why not check out my [upcoming workshop](#) [3] and take your AWS game to the next level?



Links


[1] [How DynamoDB works with AWS IAM](#)

[2] [IAM Condition element](#)

[3] [My production-ready serverless workshop](#)

Originally published at <https://theburningmonk.com> on January 10, 2024.

[AWS](#) [Dynamodb](#) [Security](#) [Vpc](#) [Cloud](#)



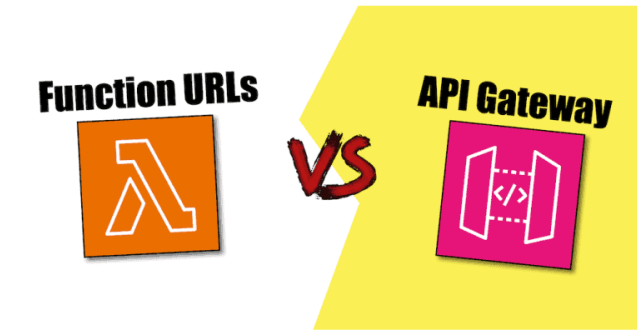
Written by Yan Cui

5.8K Followers · Editor for theburningmonk.com

AWS Serverless Hero. Follow me to learn practical tips and best practices for AWS and Serverless.

Follow

More from Yan Cui and theburningmonk.com



When to use API Gateway vs. Lambda Function URLs

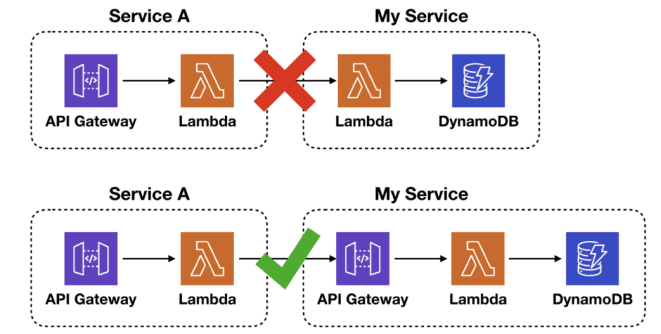
"Lambdalith" is a monolithic approach to building serverless applications where a...

5 min read · Mar 3, 2024

17

Q

...



Are Lambda-to-Lambda calls really so bad?


If you utter the words "I call a Lambda function from another Lambda function" you...

8 min read · Jul 12, 2020

90

Q 2

...



How to secure CI/CD roles without burning production to the ground

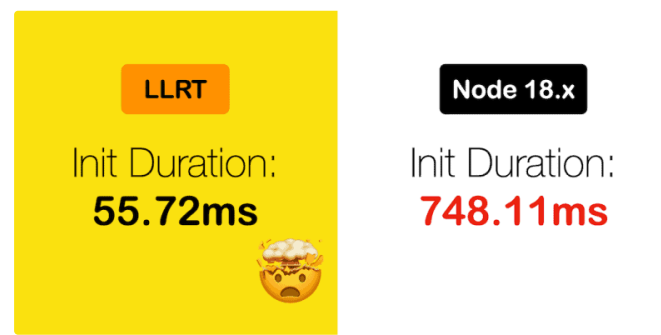
By now, most of us have moved away from using IAM users for CI/CD pipelines. Instead...

6 min read · Feb 16, 2024

5

Q

...



First impressions of the fastest JavaScript runtime for Lambda

I thought Lambda needed a specialised runtime. One that works well with its...

5 min read · Feb 27, 2024

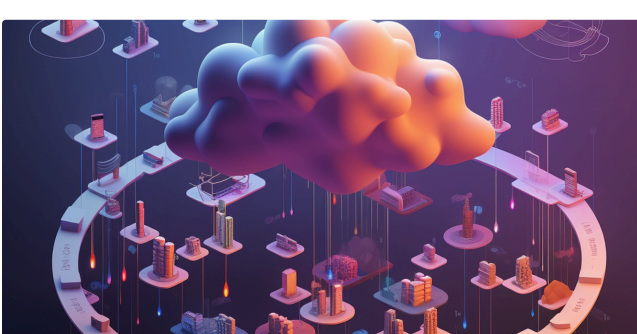
61

Q

...

[See all from Yan Cui](#) [See all from theburningmonk.com](#)

Recommended from Medium



Advanced Serverless Techniques III: Simplifying Lambda Functions...

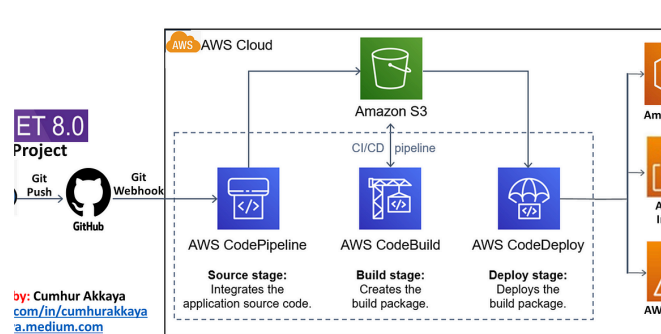
Ever found yourself repeatedly typing the same lines of code across various functions...

7 min read · Feb 9, 2024

46

Q

...



.NET Series-I: Creating CI/CD Pipeline For a .NET Application B...

First, we will install the dependencies necessary for our .NET 8.0 project to run on...


18 min read · Mar 2, 2024

16

Q 1

...

Lists



Natural Language Processing

1284 stories · 776 saves