Tips and Tricks

for the intrepid nephrologist

What is Terraform?

- Hashicorp Configuration Language, or **HCL**
  - Typical language things: variables, scalars, function library
  - But also: **Blocks**
- Providers
  - Define **data source** & **resource** blocks
- Terraform reads HCL and builds a graph of your blocks

So TF is: HCL code using the blocks from a provider to configure stuff, in the right order.

```
# Resources are config items to create.
resource "aws_kms_key" "key" {
  description = "Key for encrypting my secrets"
}

resource "aws_ssm_parameter" "secure_param" {
  name        = "/my/secret/password"
  description = "A secret password. Ooh!"
  type        = "SecureString"
  value       = "1234"

  # We're setting the key_id argument to the `arn` property from the previious block.
  # A property is like a prop on an object.
  #
  # This is also creating a relationship in the graph: can't create this block
  # until aws_kms_key.key is created, because we need one of its properties.
  key_id      = aws_kms_key.key.arn
}
```

```
# Data source block that'll look up an S3 bucket.
# This is useful for resources you _don't_ create, e.g. some other app's bucket.
# This block looks up properties with more information about the S3 bucket.
data "aws_s3_bucket" "my_bucket" {
  bucket = "my-cool-bucket-northwestern"
}

# Data source block that doesn't do any lookups in AWS.
# This is just giving you some guard-rails on an IAM policy JSON doc.
# Your IDE won't know the right formatting for the JSON, but HCL + AWS provider
# can tell it what arguments the block allows, so you'll know quicker when you mess up.
data "aws_iam_policy_document" "bad_idea_policy" {
  statement {
    effect = "Allow"

    actions   = ["s3:*"]
    resources = ["*"]
  }
}
```

Three 🔥 Tips

#1 - DRYing out the code

Adhere to the **D**on't **R**epeat **Y**ourself principle.

```
variable "runtime_secrets" {
  type        = list(string)
  derfault    = ["SECRET_ONE", "SECRET_TUESDAY", "SECRET_THREE"]
}

# Think: for(i=0; i>count(var.runtime_secrets; i++)
resource "aws_ssm_parameter" "secure_param" {
  count = length(var.runtime_secrets)
  name  = "/my-app/dev/${var.runtime_secrets[count.index]}"
}

# Wrong: aws_ssm_parameter.secure_param.arn
# Right: aws_ssm_parameter.secure_param.*.arn
```

#2 - CloudFront Breaks All the Rules

- "I'm using `us-east-2` so everything can be there" 🚫
  - CloudFront certificates & Lambda@Edge functions must exist in `us-east-1`
- "Terraform says it's deployed, we're good to go!" 🚫
  - CloudFront internally replicates changes from `us-east-1` to other edge locations, **on its own schedule**
- "I can make one log group for my Lambda@Edge function" 🚫🚫🚫
  - Lambda@Edge execution logs go to CloudWatch in the region they ran in. Which can be any region. 😩

```
# Default provider block
provider "aws" {
  region  = var.region
}

# Since it's aliased, you have to explicitly use this on a resource
provider "aws" {
  alias  = "virginia"
  region = "us-east-1"
}

resource "aws_acm_certificate" "certificate_request" {
  # . . .

  # Opt-in to the us-east-1 provider like this!
  providers = {
    aws = aws.virginia
  }
}
```

#3 - Images from Lambda/API GW

- Normally, assets (images, .xslx files) are put in a public S3 bucket and served that way
  - Anyone on the internet can download these, no problem
- But some times, that's bad.
  - Example: Wildcard photos. Letting anyone on the internet download them will upset people
  - People like the *US Dept of Education's Family Policy Compliance Office*, who enforces FERPA

Problem

Lambdas return a JSON document.

JSON is text.

So you can't put raw binary data in there; it'll break the JSON parser.

Solution: API Gateway is Clever

```
// Loaded from a /private/ S3 bucket, which this Lambda has permission to access.
const photoFromS3 = await s3.getObject(params).promise();

// Check that the user is authorized to see the picture...
if (! user.isCool()) {
  throw "ERROR! An uncool user is being uncool!";
}

return {
    headers: { "Content-Type": photoFromS3.ContentType },
    statusCode: 200,

    // API Gateway now knows to decode the body before serving!
    isBase64Encoded: true,
    body: photoFromS3.Body.toString('base64'),
}
```

That's all!

Thanks for watching.

https://github.com/nie7321/tf-tips-and-tricks