Designing Security for Cloud Apps & Infrastructure

Cassandra Young aka muteki

@muteki_rtw

Me!

- R&I, Cloud Security Engineering & Architecture @ Security Risk Advisors
- Computer Science Grad Student
  (aka glutton for punishment)
- Blue Team Village Organizer
- Lives for international travel, scuba diving, woodworking, jigsaw puzzles and baking

Cassandra Young

Pronouns: she/her

Twitter: @muteki_rtw

LinkedIn: linkedin.com/in/cassandray

GitHub: github.com/muteki-apps

Key Areas

Take Note!

THINK COMPONENTS

LEARN CODE

BE FLEXIBLE

Fundamentals | The Shared Responsibility Model

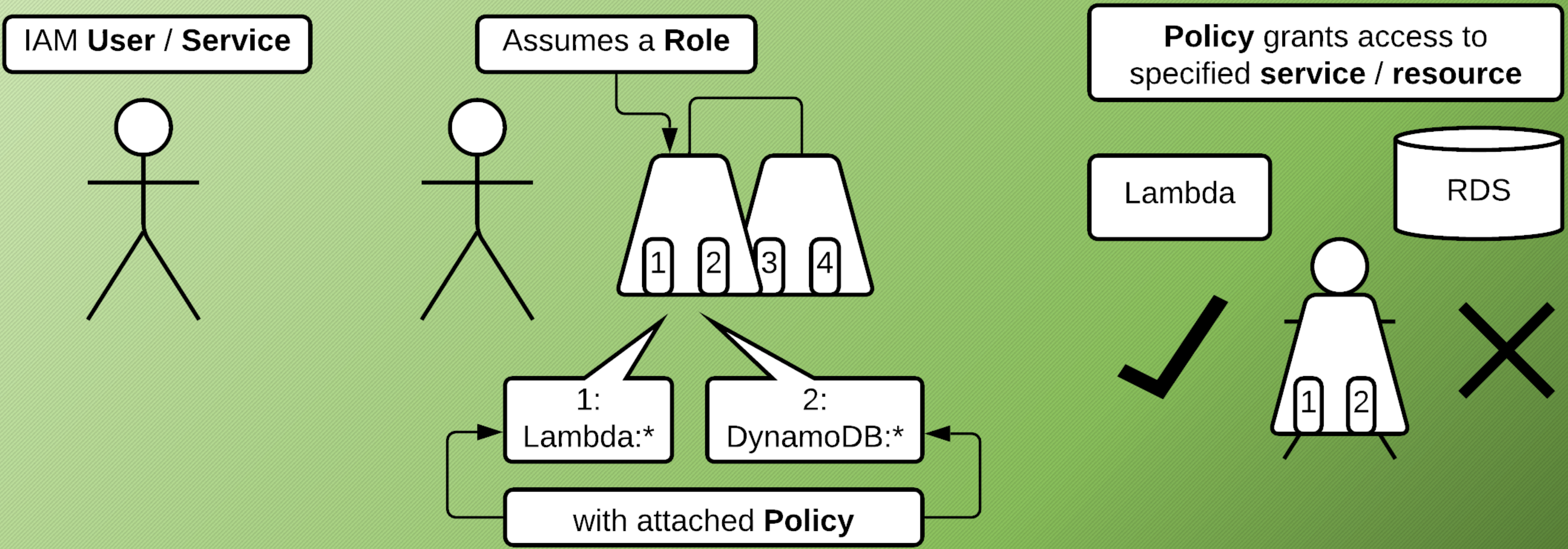
Responsibility	SaaS	PaaS	IaaS	On-prem	
Information and data	🟦	🟦	🟦	🟦	**RESPONSIBILITY ALWAYS RETAINED BY CUSTOMER**
Devices (Mobile and PCs)	🟦	🟦	🟦	🟦	
Accounts and identities	🟦	🟦	🟦	🟦	
Identity and directory infrastructure	◥	◥	🟦	🟦	**RESPONSIBILITY VARIES BY SERVICE TYPE**
Applications	⬜	◥	🟦	🟦	
Network controls	⬜	◥	🟦	🟦	
Operating system	⬜	⬜	🟦	🟦	
Physical hosts	⬜	⬜	⬜	🟦	**RESPONSIBILITY TRANSFERS TO CLOUD PROVIDER**
Physical network	⬜	⬜	⬜	🟦	
Physical datacenter	⬜	⬜	⬜	🟦	

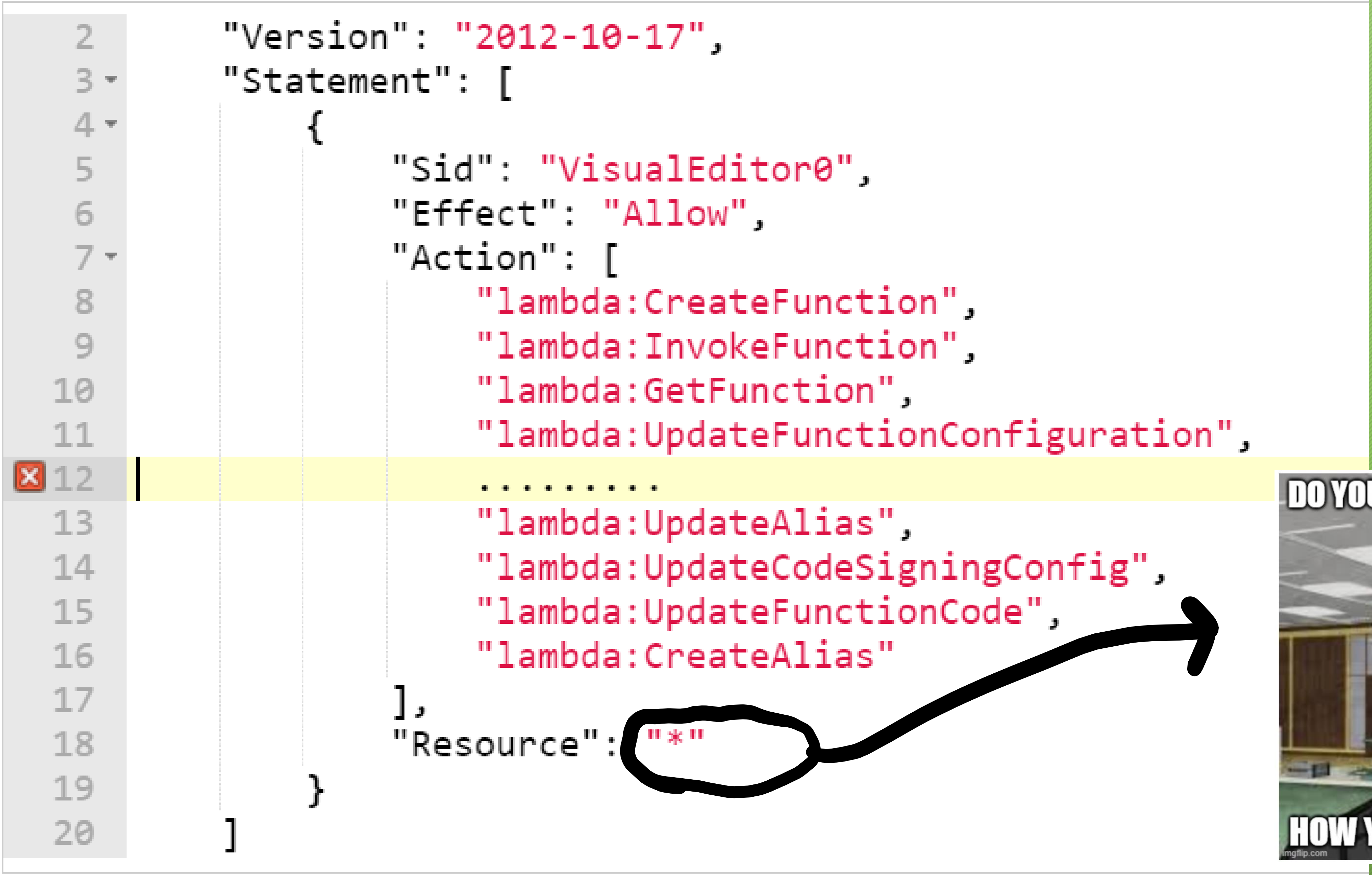
⬜ Microsoft 🟦 Customer

Key Areas to Understand for Cloud Security

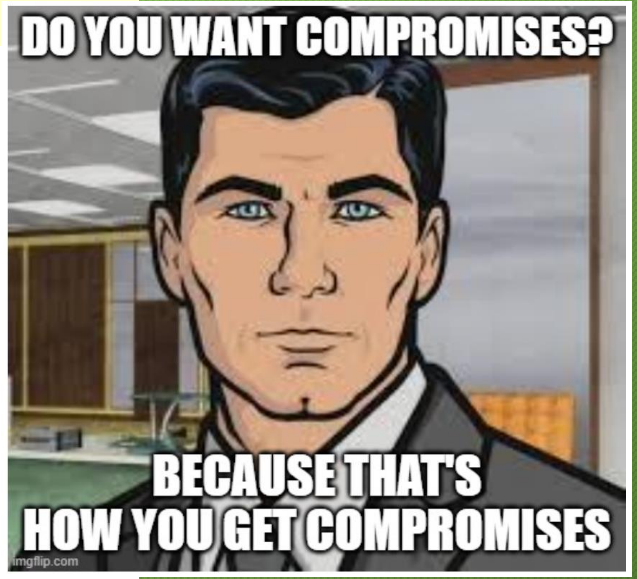
1. Permissions Perimeter
2. Resource Segmentation
    1. Via Networks & Security Groups
3. Cloud Logging & Monitoring

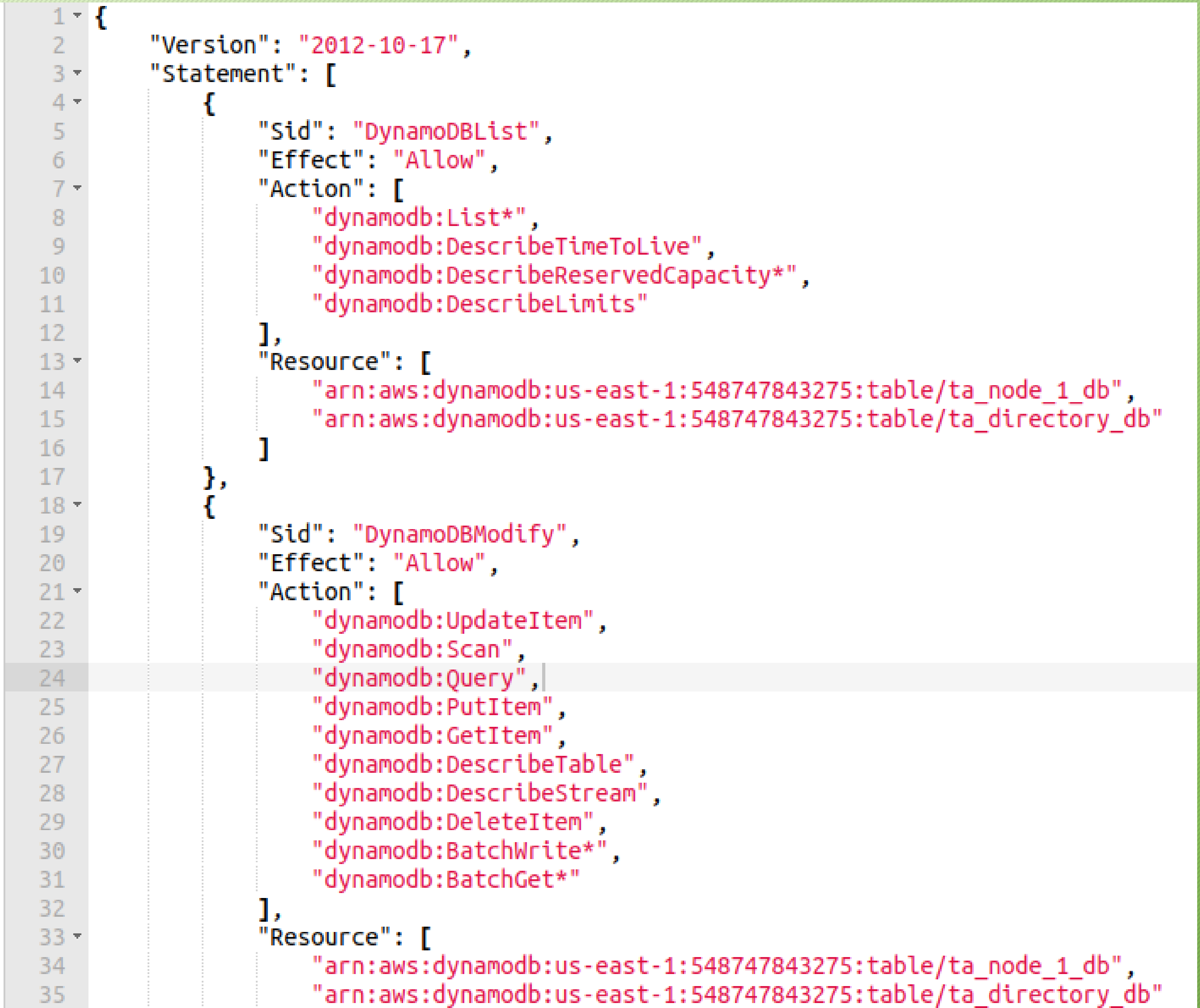
Key Areas | Identity & Access Management


IAM **User** / **Service**
Assumes a **Role**
1
2
3
4
1: Lambda:*
2: DynamoDB:*
with attached **Policy**
Policy grants access to specified **service** / **resource**
Lambda
RDS
1
2

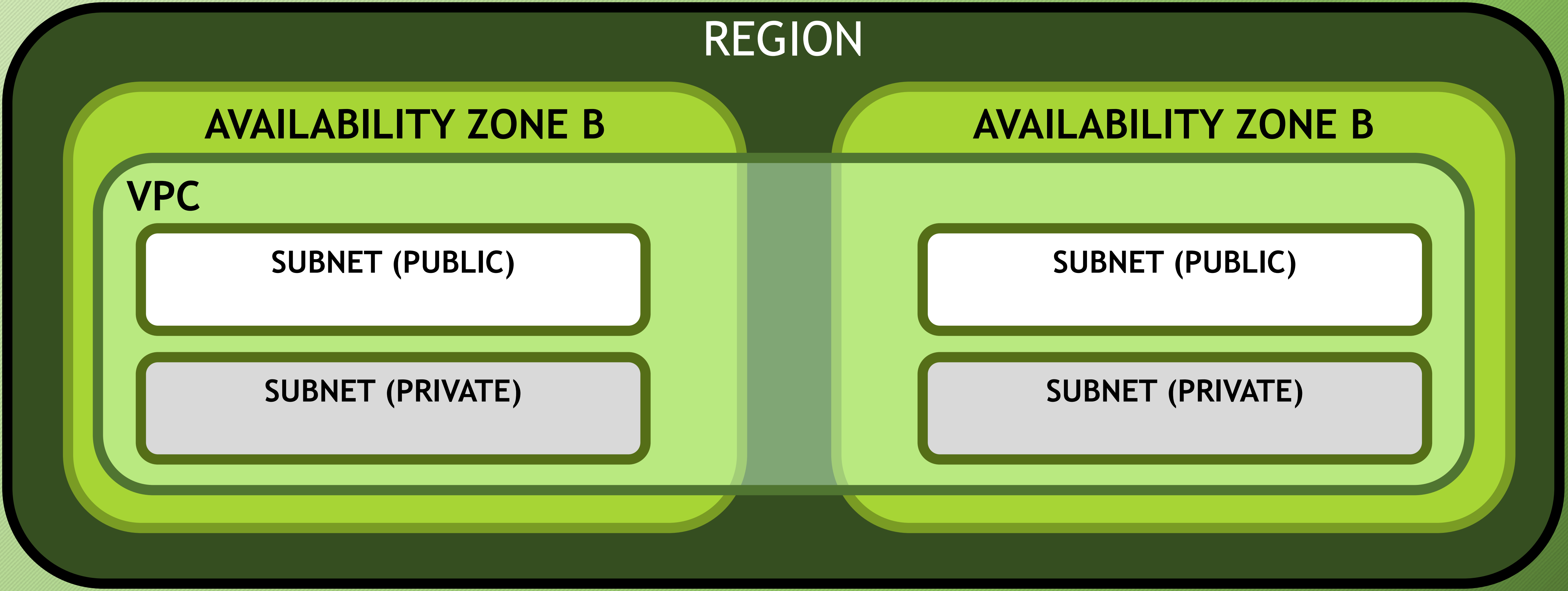


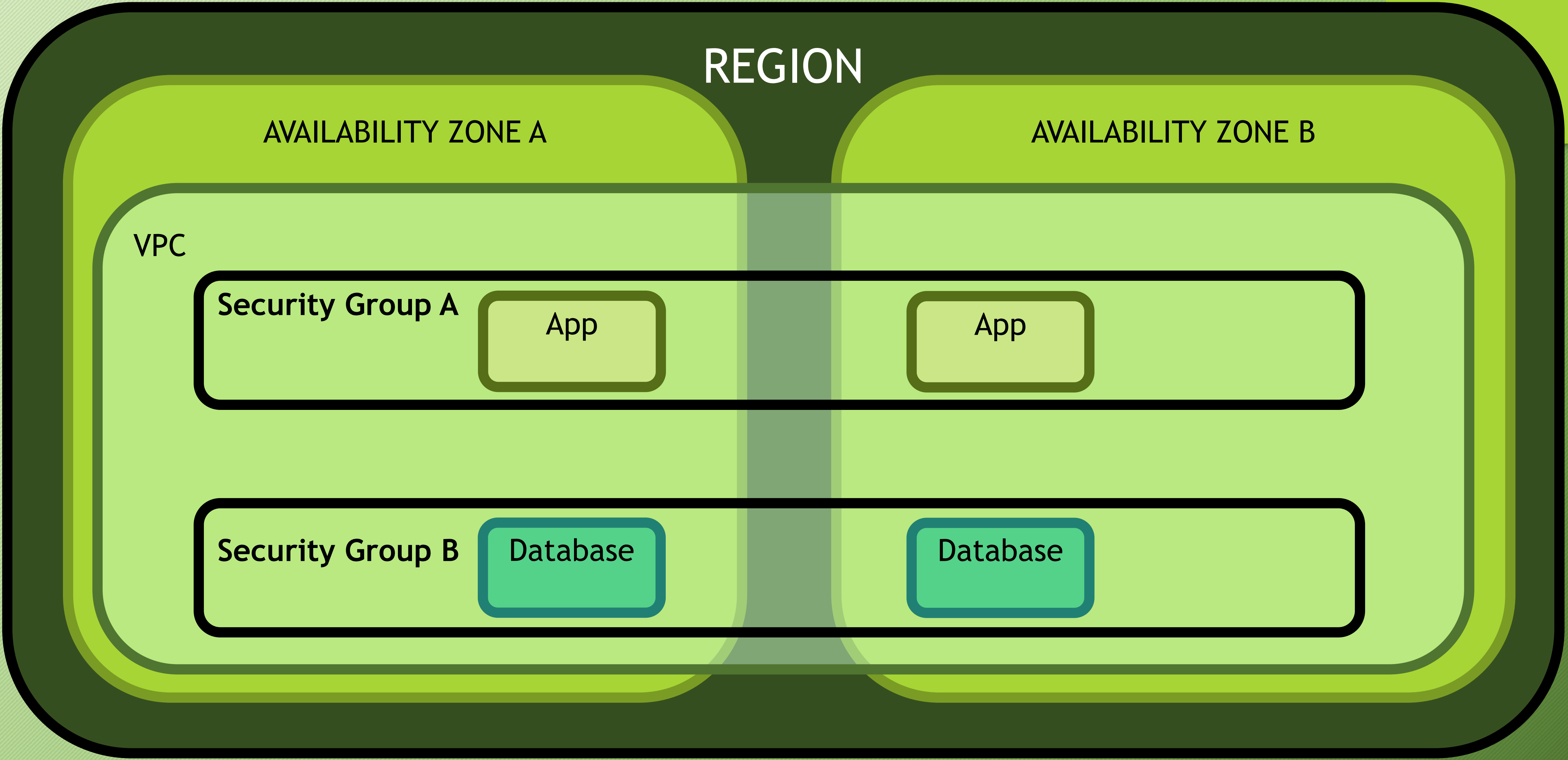
```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:UpdateFunctionConfiguration",
                ..........
                "lambda:UpdateAlias",
                "lambda:UpdateCodeSigningConfig",
                "lambda:UpdateFunctionCode",
                "lambda:CreateAlias"
            ],
            "Resource": "*"
        }
    ]
```




```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBList",
            "Effect": "Allow",
            "Action": [
                "dynamodb:List*",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:DescribeReservedCapacity*",
                "dynamodb:DescribeLimits"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:548747843275:table/ta_node_1_db",
                "arn:aws:dynamodb:us-east-1:548747843275:table/ta_directory_db"
            ]
        },
        {
            "Sid": "DynamoDBModify",
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeStream",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWrite*",
                "dynamodb:BatchGet*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:548747843275:table/ta_node_1_db",
                "arn:aws:dynamodb:us-east-1:548747843275:table/ta_directory_db"
```

Key Areas | Resource Segmentation




REGION
AVAILABILITY ZONE A
AVAILABILITY ZONE B
VPC
Security Group A
App
App
Security Group B
Database
Database

Key Areas | Logging | AWS CloudTrail, VPC Flow Logs

- Track API calls with **CloudTrail**
  - Defaults ‘on’ for many services
  - Boto3 library support
  - Integrate with CloudWatch Logs, or Splunk, SIEMs

- Log network interface traffic with **VPC Flow Logs**:
  - Log by VPC, subnet, or network interface
  - Use with other AWS services

- Monitor with **CloudWatch**:
  - Collect and analyze with dashboards
  - Build alerts, trigger other AWS services

https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-aws-service-specific-topics.html

Example App

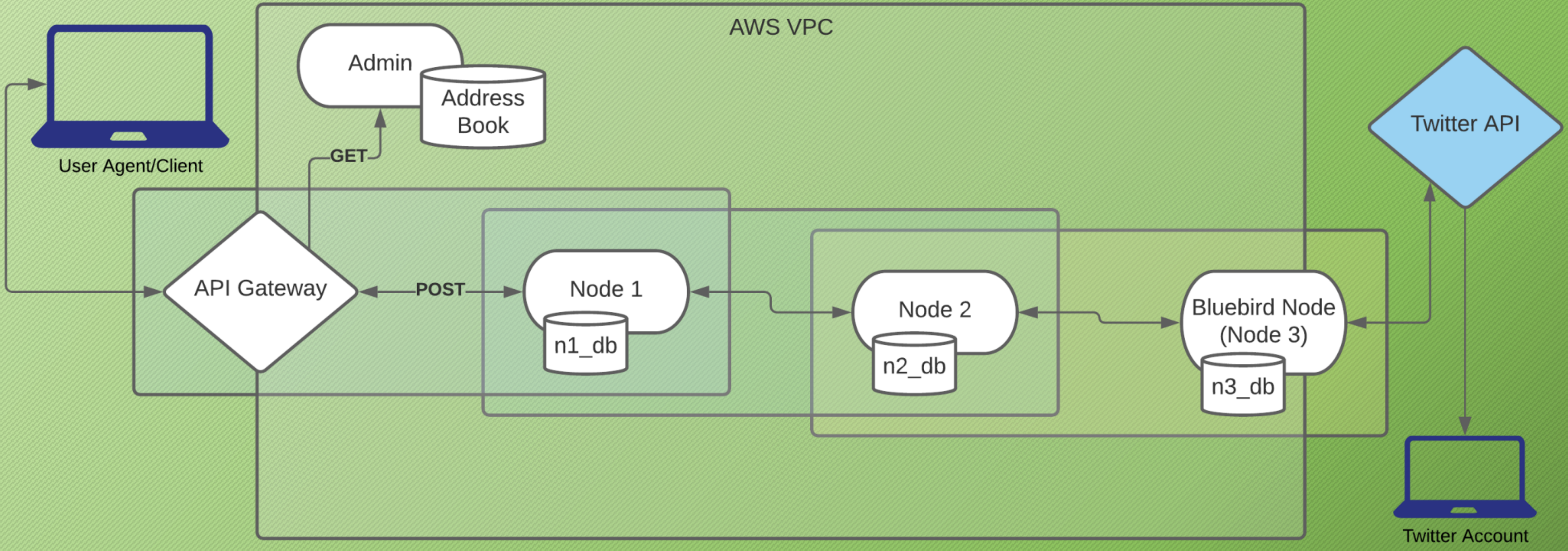
Example App: Tweeters Anonymous

1. User submits a Tweet via a Python script
2. Script pulls necessary public keys from API and **encrypts the message in layers**
3. Message is submitted to API Gateway, and is **passed through Lambda “nodes”**
4. Each **node decrypts & reads** message, which includes ‘action’: forward or publish
5. **Final node publishes** Tweet to the “Tweeters Anonymous” account via Twitter API

Tech Stack

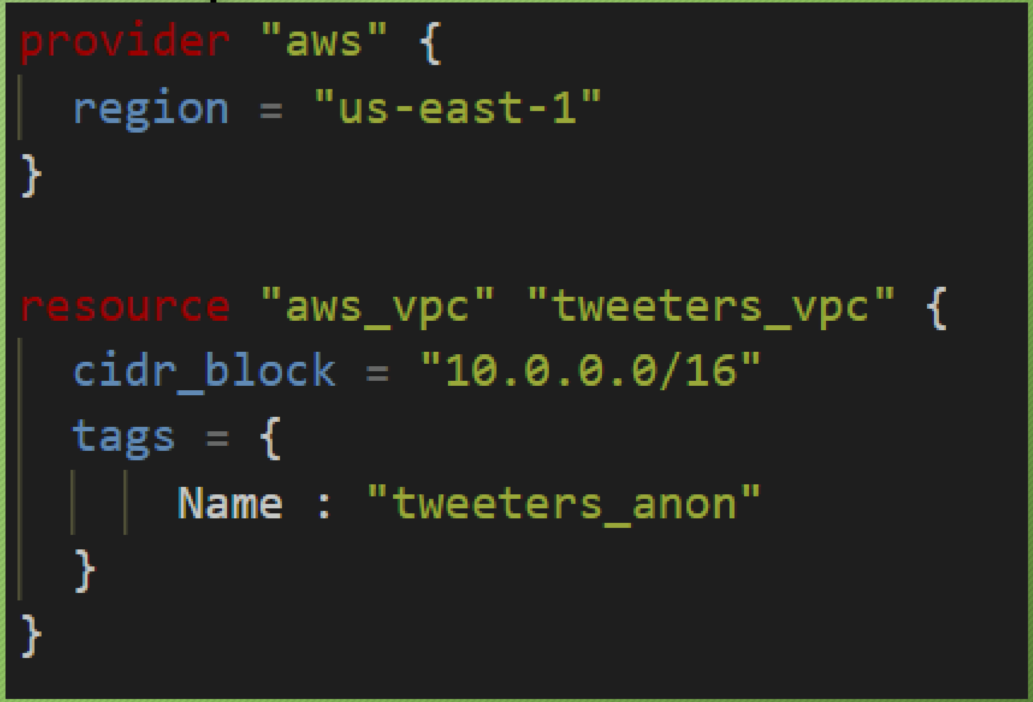
- Python (Onion Encryption, UI & “server” side code)
- AWS:
  - API Gateway – interface for UI
  - Lambda - nodes and other functions
  - DynamoDB – config & batch storage for nodes, address book
- Twitter API
- Terraform – for infrastructure creation and code deployment

Tweeters Anonymous: Architecture



Infrastructure & Automation: Terraform

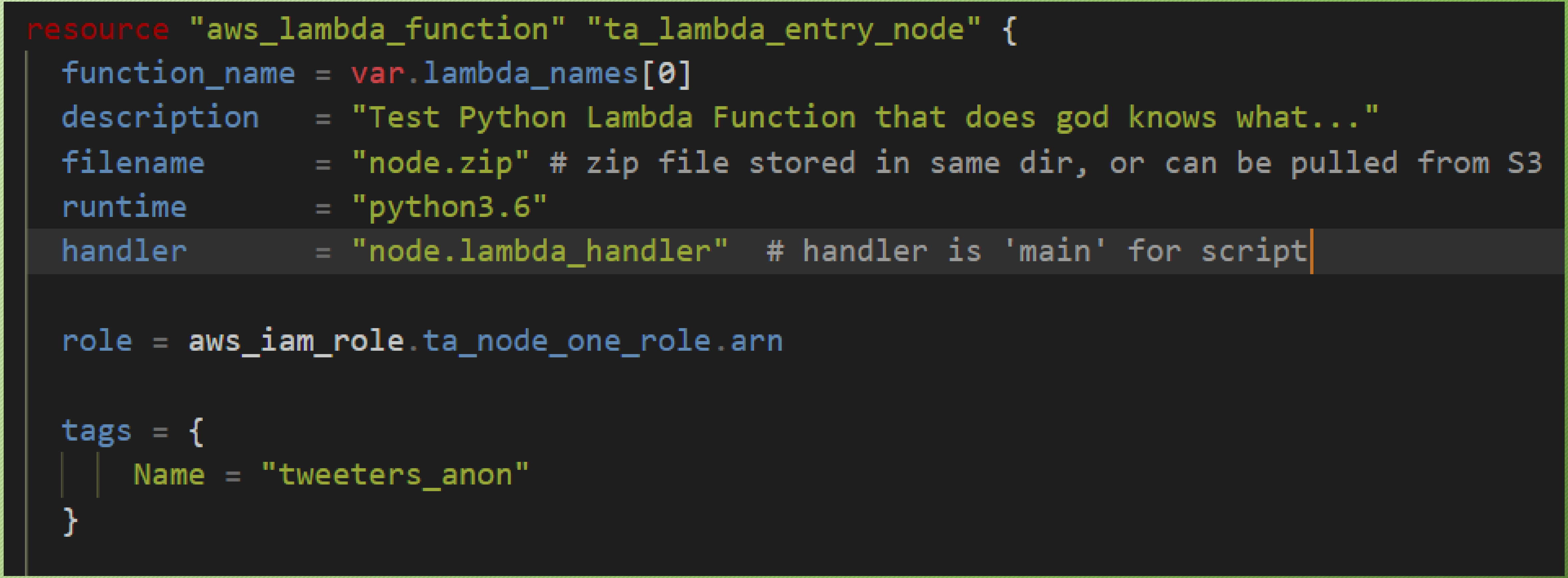
- Terraform “allows infrastructure to be expressed as code”
- Consistency is security, so why not code the blueprint?
- Use cases:
  - Destroy and recreate node instantly in case of compromise
  - Recreate entire infrastructure regularly
  - Can create configuration that allows for scalability and custom configs
- End goal: deploy code that anyone can stand up themselves



```
provider "aws" {
  region = "us-east-1"
}

resource "aws_vpc" "tweeters_vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name : "tweeters_anon"
  }
}
```

Example: Defining a Lambda

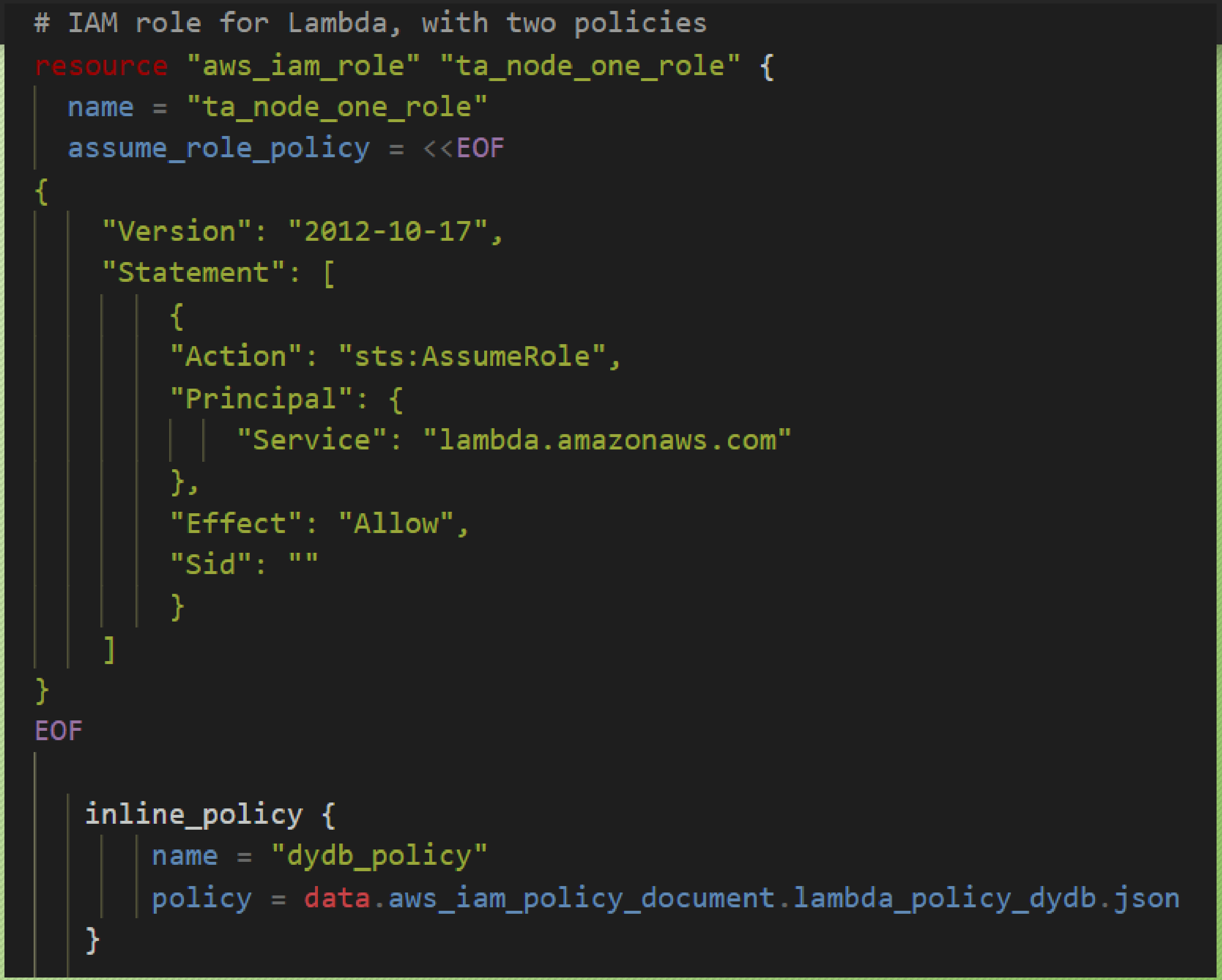


```
resource "aws_lambda_function" "ta_lambda_entry_node" {
  function_name = var.lambda_names[0]
  description   = "Test Python Lambda Function that does god knows what..."
  filename      = "node.zip" # zip file stored in same dir, or can be pulled from S3
  runtime       = "python3.6"
  handler       = "node.lambda_handler"  # handler is 'main' for script

  role = aws_iam_role.ta_node_one_role.arn

  tags = {
      Name = "tweeters_anon"
  }
```

Example: Role for the Lambda



```
# IAM role for Lambda, with two policies
resource "aws_iam_role" "ta_node_one_role" {
  name = "ta_node_one_role"
  assume_role_policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Action": "sts:AssumeRole",
        "Principal": {
            "Service": "lambda.amazonaws.com"
        },
        "Effect": "Allow",
        "Sid": ""
        }
    ]
}
EOF

   inline_policy {
     name = "dydb_policy"
     policy = data.aws_iam_policy_document.lambda_policy_dydb.json
   }
```

- Roles can be changed and applied as the program is built
- Can write in plain text, import from file, or define through Terraform’s config language
- Key idea: consistency and control

so long, and thanks for all the fish!

Cassandra Young

@muteki_rtw