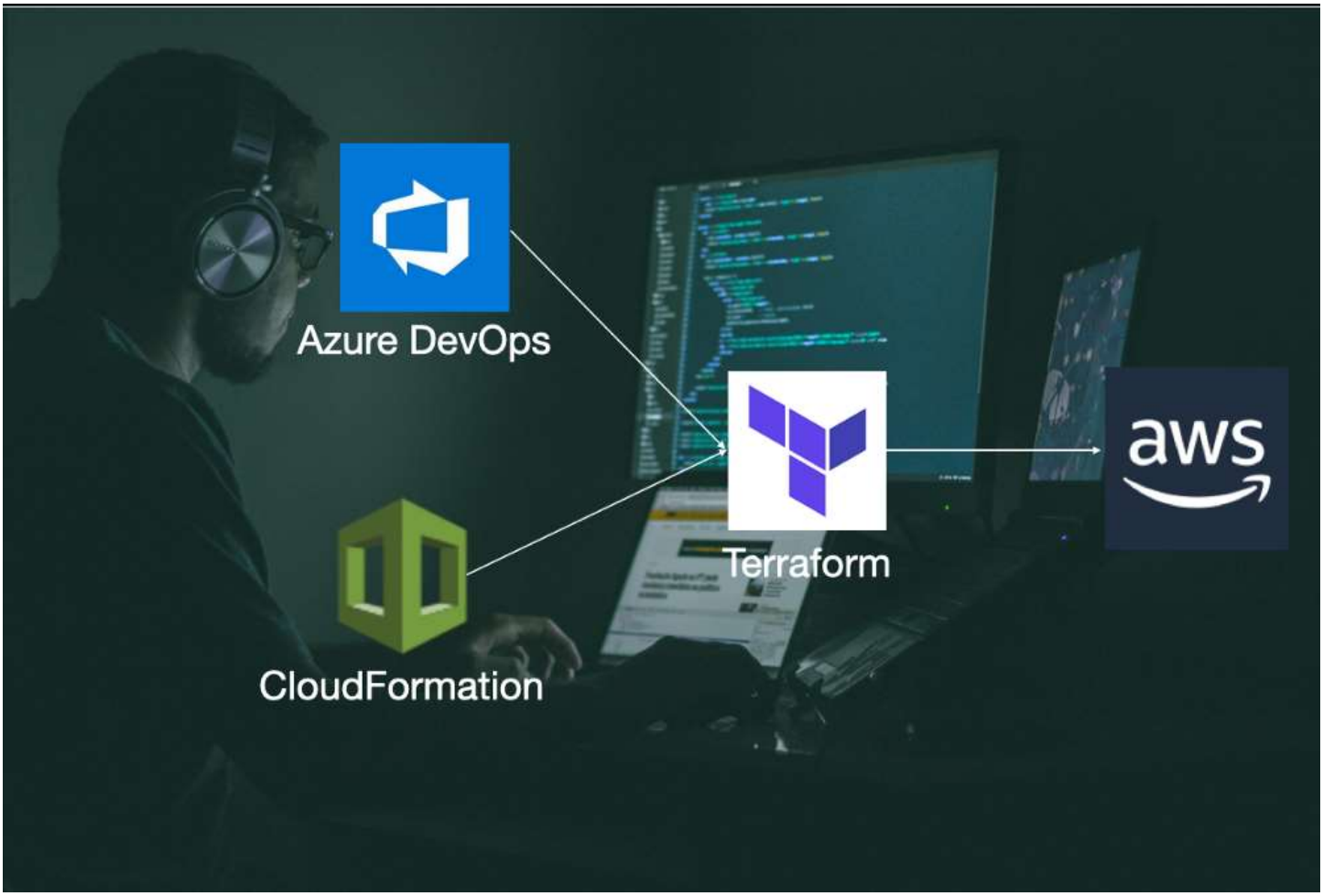
Infrastructure-As-Code for AWS using Azure DevOps, CloudFormation Template, and Terraform

An approach to set up IaC for AWS

Jagdeep Soni
Dec 26, 2020 · 4 min read



Infrastructure as code is now mainstream for cloud infrastructure provisioning and managing. There is no second argument about the need of setting up IaC as a part of modern application development. There are various frameworks and tools available

to setup IaC. Terraform is one of the most popular infrastructures as a code software tool. In this article, we will see how to use Terraform for AWS cloud.

You may be wondering why I am using all these tools together. So, here is my explanation.

> ***1)*** *Azure DevOps is used as a CI tool to create pipelines. We can also use AWS CI tools or gitlab runners or any other tool.*
>
> *2) CloudFormation is used to create first VM in AWS that could be registered as an AZ DevOps agent. We can also create this VM using Web Portal or AWS CLI.*
>
> *3) And finally Terraform, which is my tool of choice for IaC. You can continue to use CloudFormation templates for other infrastructure set up as well.*

So, what is in it for you?

The end goal is to be able to execute the Terraform code to create AWS resources. Usually, we have to deal with the complexity of managing access keys to initiate terraform but we will first create EC2 instance with an Admin role to execute terraform code.

The whole activity can be split into the following steps:

- CloudFormation template to create a VM
- Register this VM as Azure DevOps agent
- Create an Azure DevOps pipeline to trigger the CloudFormation template
- Terraform sample to create AWS resource
- Azure DevOps pipeline sample to trigger terraform

Let's get started with some code

We need a CloudFormation template to create an AWS stack. The below gist can be used to create a VM and other required resources.

```
1  AWSTemplateFormatVersion: "2010-09-09"
2
3  Description: |
4
5    VM for Azure DevOps build agent to execute terraform code
```

```
6
7   Resources:
8       AzTFVpc:
9         Type: AWS::EC2::VPC
10        Properties:
11          CidrBlock: 10.0.0.0/16
12          InstanceTenancy: default
13          EnableDnsSupport: 'true'
14          EnableDnsHostnames: 'true'
15          Tags:
16            - Key: environment
17              Value: dev
18            - Key: Name
19              Value: tf-vpc
20
21      PublicSubnet:
22        Type: AWS::EC2::Subnet
23        Properties:
24          CidrBlock: 10.0.0.0/24
25          AvailabilityZone: eu-north-1a
26          MapPublicIpOnLaunch: 'True'
27          VpcId: !Ref 'AzTFVpc'
28          Tags:
29            - Key: environment
30              Value: dev
31            - Key: Name
32              Value: tf-public-subnet
33
34      PrivateSubnet:
35        Type: AWS::EC2::Subnet
36        Properties:
37          CidrBlock: 10.0.2.0/24
38          AvailabilityZone: eu-north-1a
39          VpcId: !Ref 'AzTFVpc'
40          Tags:
41            - Key: environment
42              Value: dev
43            - Key: Name
44              Value: tf-private-subnet
45
46
47      TfIgw:
48        Type: AWS::EC2::InternetGateway
49        Properties:
50          Tags:
51            - Key: environment
52              Value: dev
53            - Key: Name
```

```
            Key: Name
            Value: tf-igw

    NetworkACL:
      Type: AWS::EC2::NetworkAcl
      Properties:
        VpcId: !Ref 'AzTFVpc'
        Tags:
          - Key: environment
            Value: dev
          - Key: Name
            Value: tf-nacl

    RoutePublic:
      Type: AWS::EC2::RouteTable
      Properties:
        VpcId: !Ref 'AzTFVpc'
        Tags:
          - Key: environment
            Value: dev
          - Key: Name
            Value: tf-public-route

    RoutePrivate:
      Type: AWS::EC2::RouteTable
      Properties:
        VpcId: !Ref 'AzTFVpc'
        Tags:
          - Key: environment
            Value: dev
          - Key: Name
            Value: tf-private-route

    Instance:
      Type: AWS::EC2::Instance
      Properties:
        DisableApiTermination: 'false'
        InstanceInitiatedShutdownBehavior: stop
        ImageId: ami-0a3a4169ad7cb0d77
        InstanceType: t3.micro
        IamInstanceProfile: !Ref 'Ec2InstanceProfile'
        Monitoring: 'true'
        Tags:
          - Key: environment
            Value: dev
          - Key: Name
            Value: tf-instance
        NetworkInterfaces:
```

```
      - AssociatePublicIpAddress: 'true'
        DeleteOnTermination: 'true'
        Description: Primary network interface
        DeviceIndex: 0
        SubnetId: !Ref 'PublicSubnet'
        GroupSet: [!Ref 'TfSgApp']

  Ec2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: /
      Roles: [ !Ref Ec2InstanceRole ]
  Ec2InstanceRole:
    Type: AWS::IAM::Role
    Properties:
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforSSM
        - arn:aws:iam::aws:policy/AdministratorAccess
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: [ ec2.amazonaws.com ]
            Action:
              - sts:AssumeRole
      Path: /

  TfSgApp:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: App server security group
      VpcId: !Ref 'AzTFVpc'
      SecurityGroupIngress:
       - IpProtocol: tcp
         CidrIp: 0.0.0.0/0
         FromPort: 80
         ToPort: 80
      Tags:
        - Key: environment
          Value: dev
        - Key: Name
          Value: AppServerSecurityGroup

  NACLEntry1:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: 0.0.0.0/0
      Egress: 'true'
```

```
      Protocol: '-1'
      RuleAction: allow
      RuleNumber: '100'
      NetworkAclId: !Ref 'NetworkACL'

  NACLEntry2:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: 0.0.0.0/0
      Protocol: '-1'
      RuleAction: allow
      RuleNumber: '100'
      NetworkAclId: !Ref 'NetworkACL'

  subnetacl1:
    Type: AWS::EC2::SubnetNetworkAclAssociation
    Properties:
      NetworkAclId: !Ref 'NetworkACL'
      SubnetId: !Ref 'PublicSubnet'

  subnetacl3:
    Type: AWS::EC2::SubnetNetworkAclAssociation
    Properties:
      NetworkAclId: !Ref 'NetworkACL'
      SubnetId: !Ref 'PrivateSubnet'


  IGWAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref 'AzTFVpc'
      InternetGatewayId: !Ref 'TfIgw'

  subnetRoutePublicA:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref 'RoutePublic'
      SubnetId: !Ref 'PublicSubnet'


  subnetRoutePrivateA:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref 'RoutePrivate'
      SubnetId: !Ref 'PrivateSubnet'

  publicroute:
```

```
196         Type: AWS::EC2::Route
197         Properties:
198           DestinationCidrBlock: 0.0.0.0/0
199           RouteTableId: !Ref 'RoutePublic'
```

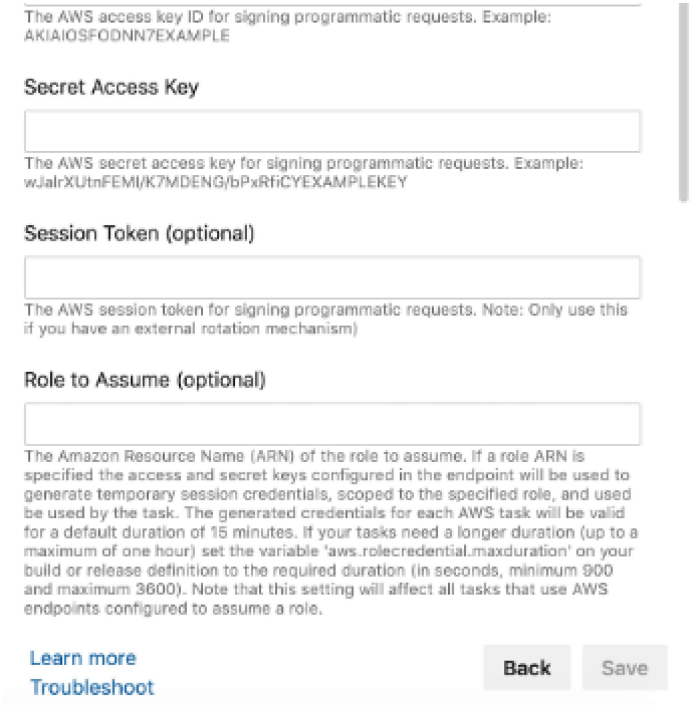
> *`arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforSSM` is required for instance connection and `arn:aws:iam::aws:policy/AdministratorAccess` to execute terraform code from this agent*

Now that we have the VM in place, we need to configure this VM as a DevOps agent. We can follow the steps described here to configure this VM as an agent. Some of these steps can be configured in `UserData` field in the above template. A small snippet is shown below.

```
UserData:
 Fn::Base64: !Sub |
  #!/bin/bash
  export PAT_TOKEN=${PatToken}
  export POOL_NAME=${PoolName}
  export AGENT_NAME=${AgentName}

## Download the Linux agent

/home/ubuntu/buildagent/config.sh --unattended --auth pat --token
$PAT_TOKEN --url https://dev.azure.com/ --pool $POOL_NAME --agent
$AGENT_NAME

sudo /home/ubuntu/buildagent/svc.sh install

sudo /home/ubuntu/buildagent/svc.sh start
```

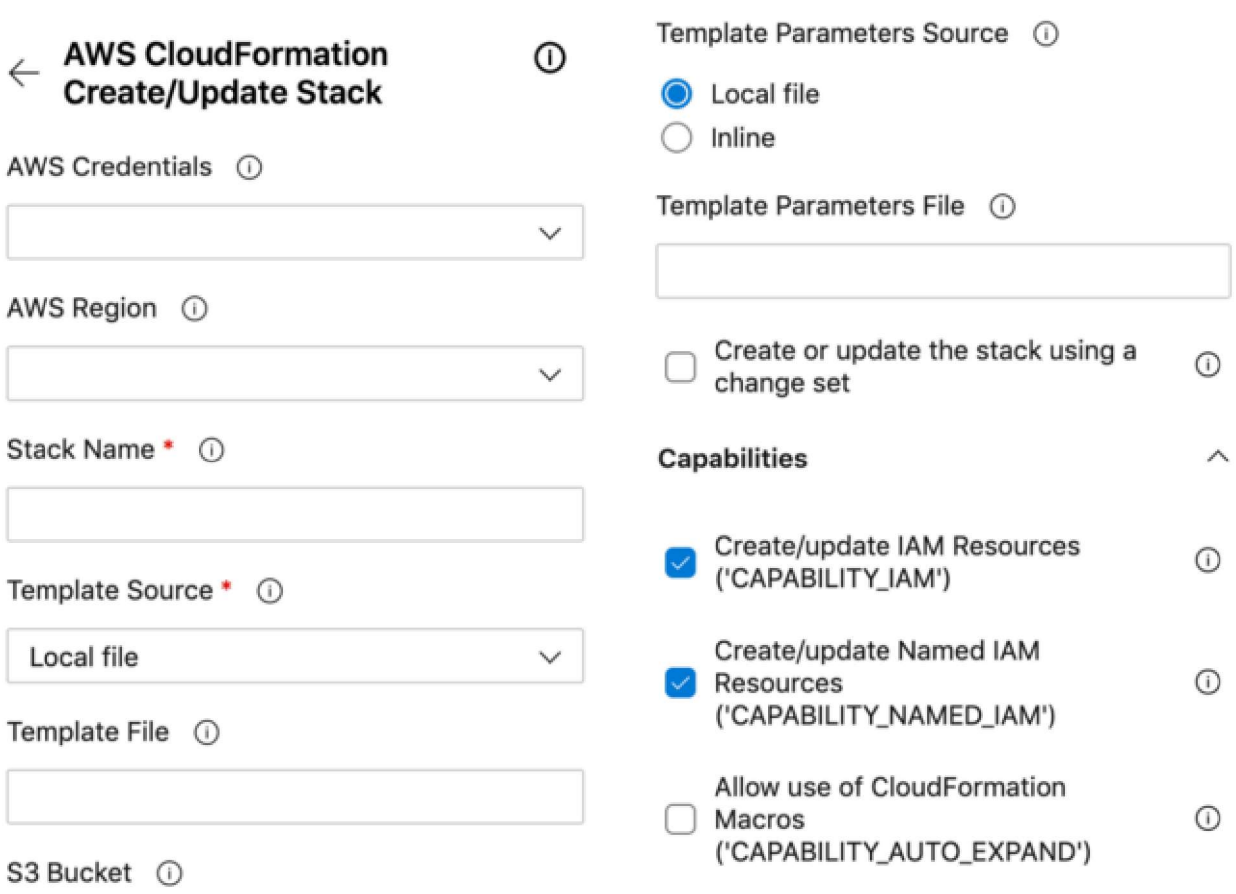
Time to put the pipeline in place

Let's create an Azure DevOps pipeline to trigger this CloudFormation template. We first need to create AWS Service Connection in Azure DevOps.



Service Connection in Azure DevOps

We can then use AWS tasks to configure this pipeline. In this case, we need the AWS CloudFormation Create/Update Stack task. We need to fill in some of the following inputs to use this task.



Finally, the pipeline yaml would look like the below snippet.

```yaml
trigger:
- development

variables:
  - group: tf-aws-agent

pool:
  vmImage: ubuntu-latest

steps:
- script: |
    sed "s/{{pat_token}}/${DEVOPS_PAT}/g; s/{{pool_name}}/${POOL_NAME}/g; s/{{agent_name}}/${AW
    cat ./cft-parameters.json
  displayName: 'Add pat token'
- task: CloudFormationCreateOrUpdateStack@1
  inputs:
    awsCredentials: 'AWS_Conn'
    regionName: 'eu-north-1'
    stackName: 'az-tf-agent'
    templateSource: 'file'
    templateFile: './utils/terraform_init/aws/cfn/azdevops-agent-vm-cft.yaml'
    templateParametersFile: './cft-parameters.json'
```

azure-pipelines.yaml hosted with ♡ by GitHub view raw

Time to create a Terraform script

There are a few good scripts in this repository. For this article, we will take a simple example to create Elastic Container Registry(ECR) using terraform.

```hcl
terraform {
  backend "s3" {
    bucket = "tfrb"
    key = "tfrb.state"
    region = "eu-west-1"
  }
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "eu-west-1"
}
```

```
18
19  resource "aws_ecr_repository" "exmple" {
20    name                 = "example"
21    image_tag_mutability = "MUTABLE"
22
23    image_scanning_configuration {
24      scan_on_push = true
25    }
26  }
```

main.tf hosted with ♡ by GitHub view raw

Now the final step is to create an Azure DevOps Pipeline to trigger terraform code. Below is the code snippet for Azure DevOps Pipeline.

> *We need to make sure that we should use same Agent Pool in which we have registered the instance created in step 1*

```
1   trigger:
2     branches:
3       include:
4         - development
5
6   pool:
7     name: AWS_AGENT_POOL # Name of the agent pool that has AWS agent created in step 1
8
9   resources:
10    containers:
11      - container: terraform-runtime
12        image: foo.dkr.ecr.eu-west-1.amazonaws.com/terraform-runtime # Use the docker image with
13        endpoint: ECR_Conn # Azure DevOps service connection to the ECR
14
15  stages:
16    - stage: Terraform
17      displayName: Terraform Create
18      jobs:
19        - job: Trigger Terraform
20          displayName: Trigger Terraform
21          container: terraform-runtime
22          steps:
23            - bash: terraform init
24              displayName: "Initialize Terraform"
25            - bash: terraform plan
26              displayName: "Planning Terraform"
27            - bash: terraform apply
28              displayName: "Apply terraform"
```

tf-pipeline.yaml hosted with ♡ by GitHub view raw

Conclusion 🍾

I hope this article will help you to kick start IaC in your application. Please feel free to share your questions, feedback, and experience in the comments section.