

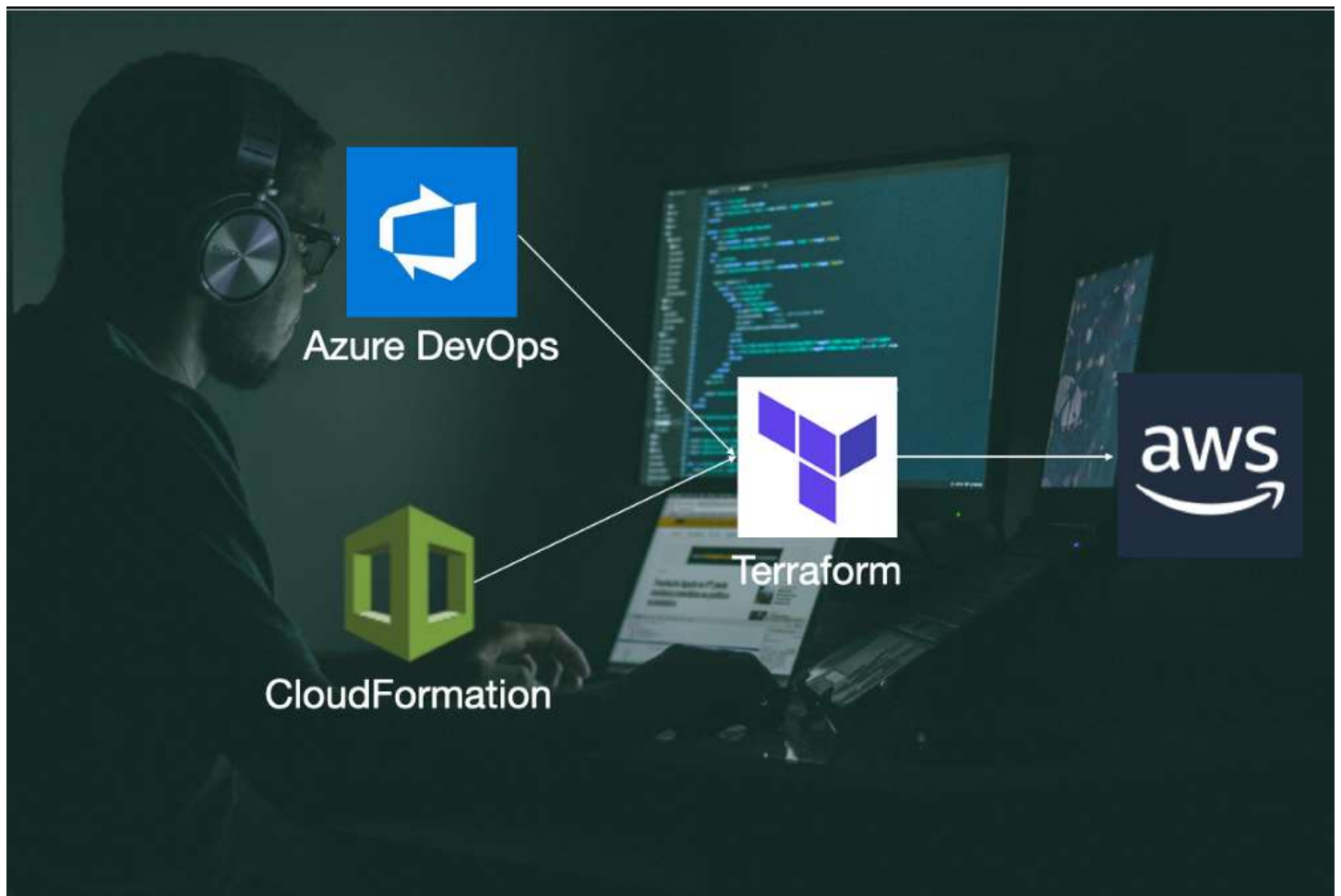
Infrastructure-As-Code for AWS using Azure DevOps, CloudFormation Template, and Terraform

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
```

```
53         Key: Name
54         Value: tf-igw
55
56     NetworkACL:
57         Type: AWS::EC2::NetworkAcl
58         Properties:
59             VpcId: !Ref 'AzTFVpc'
60             Tags:
61                 - Key: environment
62                   Value: dev
63                 - Key: Name
64                   Value: tf-nacl
65
66     RoutePublic:
67         Type: AWS::EC2::RouteTable
68         Properties:
69             VpcId: !Ref 'AzTFVpc'
70             Tags:
71                 - Key: environment
72                   Value: dev
73                 - Key: Name
74                   Value: tf-public-route
75
76     RoutePrivate:
77         Type: AWS::EC2::RouteTable
78         Properties:
79             VpcId: !Ref 'AzTFVpc'
80             Tags:
81                 - Key: environment
82                   Value: dev
83                 - Key: Name
84                   Value: tf-private-route
85
86     Instance:
87         Type: AWS::EC2::Instance
88         Properties:
89             DisableApiTermination: 'false'
90             InstanceInitiatedShutdownBehavior: stop
91             ImageId: ami-0a3a4169ad7cb0d77
92             InstanceType: t3.micro
93             IamInstanceProfile: !Ref 'Ec2InstanceProfile'
94             Monitoring: 'true'
95             Tags:
96                 - Key: environment
97                   Value: dev
98                 - Key: Name
99                   Value: tf-instance
100             NetworkInterfaces:
```

```
101     - AssociatePublicIpAddress: 'true'
102     DeleteOnTermination: 'true'
103     Description: Primary network interface
104     DeviceIndex: 0
105     SubnetId: !Ref 'PublicSubnet'
106     GroupSet: [!Ref 'TfSgApp']
107
108   Ec2InstanceProfile:
109     Type: AWS::IAM::InstanceProfile
110     Properties:
111       Path: /
112       Roles: [ !Ref Ec2InstanceRole ]
113   Ec2InstanceRole:
114     Type: AWS::IAM::Role
115     Properties:
116       ManagedPolicyArns:
117         - arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforSSM
118         - arn:aws:iam::aws:policy/AdministratorAccess
119       AssumeRolePolicyDocument:
120         Statement:
121           - Effect: Allow
122             Principal:
123               Service: [ ec2.amazonaws.com ]
124             Action:
125               - sts:AssumeRole
126       Path: /
127
128   TfSgApp:
129     Type: AWS::EC2::SecurityGroup
130     Properties:
131       GroupDescription: App server security group
132       VpcId: !Ref 'AzTFVpc'
133       SecurityGroupIngress:
134         - IpProtocol: tcp
135           CidrIp: 0.0.0.0/0
136           FromPort: 80
137           ToPort: 80
138       Tags:
139         - Key: environment
140           Value: dev
141         - Key: Name
142           Value: AppServerSecurityGroup
143
144   NACLEntry1:
145     Type: AWS::EC2::NetworkAclEntry
146     Properties:
147       CidrBlock: 0.0.0.0/0
148       Egress: 'true'
```

```
149     Protocol: '-1'
150     RuleAction: allow
151     RuleNumber: '100'
152     NetworkACLId: !Ref 'NetworkACL'
153
154   NACLEntry2:
155     Type: AWS::EC2::NetworkAclEntry
156     Properties:
157       CidrBlock: 0.0.0.0/0
158       Protocol: '-1'
159       RuleAction: allow
160       RuleNumber: '100'
161       NetworkACLId: !Ref 'NetworkACL'
162
163   subnetacl1:
164     Type: AWS::EC2::SubnetNetworkAclAssociation
165     Properties:
166       NetworkACLId: !Ref 'NetworkACL'
167       SubnetId: !Ref 'PublicSubnet'
168
169   subnetacl3:
170     Type: AWS::EC2::SubnetNetworkAclAssociation
171     Properties:
172       NetworkACLId: !Ref 'NetworkACL'
173       SubnetId: !Ref 'PrivateSubnet'
174
175
176   IGWAttachment:
177     Type: AWS::EC2::VPCGatewayAttachment
178     Properties:
179       VpcId: !Ref 'AzTFVpc'
180       InternetGatewayId: !Ref 'TfIgw'
181
182   subnetRoutePublicA:
183     Type: AWS::EC2::SubnetRouteTableAssociation
184     Properties:
185       RouteTableId: !Ref 'RoutePublic'
186       SubnetId: !Ref 'PublicSubnet'
187
188
189   subnetRoutePrivateA:
190     Type: AWS::EC2::SubnetRouteTableAssociation
191     Properties:
192       RouteTableId: !Ref 'RoutePrivate'
193       SubnetId: !Ref 'PrivateSubnet'
194
195   publicroute:
```

```

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

```

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.

New AWS service connection



Authentication

Access Key ID

The AWS access key ID for signing programmatic requests. Example: AKIAIOSFODNN7EXAMPLE

Secret Access Key

The AWS secret access key for signing programmatic requests. Example: wJalrXUtnFEMI/K7MDENG/bPxrFcYEXAMPLEKEY

Session Token (optional)

The AWS session token for signing programmatic requests. Note: Only use this if you have an external rotation mechanism)

Role to Assume (optional)

The Amazon Resource Name (ARN) of the role to assume. If a role ARN is specified the access and secret keys configured in the endpoint will be used to generate temporary session credentials, scoped to the specified role, and used by the task. The generated credentials for each AWS task will be valid for a default duration of 15 minutes. If your tasks need a longer duration (up to a maximum of one hour) set the variable 'aws.rolecredential.maxduration' on your build or release definition to the required duration (in seconds, minimum 900 and maximum 3600). Note that this setting will affect all tasks that use AWS endpoints configured to assume a role.

Learn more

Troubleshoot

Back

Save

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.

← AWS CloudFormation Create/Update Stack ⓘ

AWS Credentials ⓘ

AWS Region ⓘ

Stack Name * ⓘ

Template Source * ⓘ

Local file

Template File ⓘ

S3 Bucket ⓘ

Template Parameters Source ⓘ

☒ Local file

☐ Inline

Template Parameters File ⓘ

☐ Create or update the stack using a change set ⓘ

Capabilities ^

☒ Create/update IAM Resources ('CAPABILITY_IAM') ⓘ

☒ Create/update Named IAM Resources ('CAPABILITY_NAMED_IAM') ⓘ

☐ Allow use of CloudFormation Macros ('CAPABILITY_AUTO_EXPAND') ⓘ

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


```
1 trigger:
2   - development
3
4 variables:
5   - group: tf-aws-agent
6
7 pool:
8   vmImage: ubuntu-latest
9
10 steps:
11   - script: |
12       sed "s/{{pat_token}}/${DEVOPS_PAT}/g; s/{{pool_name}}/${POOL_NAME}/g; s/{{agent_name}}/${AGENT_NAME}/g" \
13       cat ./cft-parameters.json
14   displayName: 'Add pat token'
15   - task: CloudFormationCreateOrUpdateStack@1
16   inputs:
17     awsCredentials: 'AWS_Conn'
18     regionName: 'eu-north-1'
19     stackName: 'az-tf-agent'
20     templateSource: 'file'
21     templateFile: './utils/terraform_init/aws/cfn/azdevops-agent-vm-cft.yaml'
22     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.

```
1 terraform {
2   backend "s3" {
3     bucket = "tfrb"
4     key    = "tfrb.state"
5     region = "eu-west-1"
6   }
7   required_providers {
8     aws = {
9       source = "hashicorp/aws"
10      version = "~> 3.0"
11    }
12  }
13 }
14
15 provider "aws" {
16   region = "eu-west-1"
17 }
```

```
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.

👏 **Join FAUN today and receive similar stories each week in your inbox!** Get your weekly dose of the must-read tech stories, news, and tutorials.

Follow us on [Twitter](#) 🐦 and [Facebook](#) 👤 and [Instagram](#) 📷 and join our [Facebook](#) and [Linkedin](#) Groups 💬

If this post was helpful, please click the clap 🖐 button below a few times to show your support for the author! ↓

[Infrastructure As Code](#)[Azure Devops](#)[AWS](#)[Terraform](#)[Cloudformation](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

