Infrastructure as Code: Create CloudFormation Stack

CloudFormation will be used in this lesson to recreate the same instrastructure we set up in the previous lessons. We are going to automate the process of instrastructure creation, instead of setting everything up manually through the AWS console.

We'll cover the following

Objective

- Recreate our infrastructure using CloudFormation.

Steps

- Configure the AWS CLI.
- Create a CloudFormation Stack.

In this section, we'll recreate the same infrastructure we set up in the previous section, but this time, we'll use CloudFormation to automate the process, instead of setting everything up manually through the AWS console.

Configuring the AWS CLI

We're going to use the AWS CLI to access AWS resources from the command line rather than the AWS console. If you don't already have it installed, follow the official directions for your system. Then, configure a profile named `awsbootstrap` using a newly generated Access Key ID and Secret Access Key, as described in the AWS documentation.

```
aws configure --profile awsbootstrap
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: json
```

terminal

We can test our AWS CLI configuration by listing the EC2 instances in our account.

```
aws ec2 describe-instances --profile awsbootstrap
```

terminal

Now that we have the AWS CLI working, we could write a bunch of scripts that call the various AWS API commands to automate the process for setting up our infrastructure. But that would be very brittle and complicated. Luckily, there's a better way.

Infrastructure as code

Infrastructure as code is the idea of using the same processes and tools to update your infrastructure as you do for your application code. We will now start defining our infrastructure into files that can be linted, schema-checked, version controlled, and deployed without manual processes. Within AWS, the tool for this is CloudFormation.

We'll use the AWS CLI to submit infrastructure updates to CloudFormation. Although we could interact with CloudFormation directly from the AWS CLI, it is easier to write a script containing the necessary parameters. We'll call the script `deploy-infra.sh` and use it to deploy changes to our CloudFormation stack. A stack is what CloudFormation calls the collection of resources that are managed together as a unit.

```
#!/bin/bash

STACK_NAME=awsbootstrap
REGION=us-east-1
CLI_PROFILE=awsbootstrap

EC2_INSTANCE_TYPE=t2.micro

# Deploy the CloudFormation template
```

```
echo -e "\n\n========== Deploying main.yml =========="
aws cloudformation deploy \
  --region $REGION \

  --profile $CLI_PROFILE \
  --stack-name $STACK_NAME \
  --template-file main.yml \
  --no-fail-on-empty-changeset \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides \
    EC2InstanceType=$EC2_INSTANCE_TYPE
```

deploy-infra.sh

Line #3: The stack name is the name that CloudFormation will use to refer to the group of resources it will manage.

Line #4: The region to deploy to.

Line #5: We use the `awsbootstrap` profile that we created in the previous section.

Line #7: An instance type in the free tier.

Line #15: The `main.yml` file is the CloudFormation template that we will use to define our infrastructure.

Line #18: These correspond to the input parameters in the template that we'll write next.

Before we move on, let's make our helper script executable.

```
chmod +x deploy-infra.sh
```

terminal

Now it's time to start creating our CloudFormation template. It uses the following three top-level sections.

Parameters

These are the input parameters for the template. They give us the flexibility to change some settings without having to modify the template code.

Resources

This is the bulk of the template. Here is where we define and configure the resources that CloudFormation will manage for us.

Outputs

These are like return values for the template. We use them to make it easy to find some of the resources that CloudFormation will create for us.

We're going to name our template file `main.yml`. There will be other template files later, but they will all be referenced from here. This file will become quite large, so let's start by sketching out its high-level structure.

```
AWSTemplateFormatVersion: 2010-09-09

Parameters:

Resources:

Outputs:
```

main.yml

Next, let's fill in the input parameters to accept the instance type. The parameter names we'll use in the template will need to match the parameters we used in the `deploy-infra.sh` script. For now, we have `EC2InstanceType`. We will add other parameters throughout the book. `EC2AMI` is a bit special. We use the AWS SSM provided value that specifies the most up-to-date AMI.

```
Parameters:
  EC2InstanceType:
    Type: String
  EC2AMI:
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2'
```

main.yml

On line #5: This is a special parameter type that allows our template to get the latest AMI without having to specify the exact version.

The first resource that we're going to define is our security group. This functions like a firewall for the EC2 instance that we'll create. We need to add a rule to allow TCP traffic to port 8080 (to reach our application) and to port 22 (for SSH access).

```
Resources:

  SecurityGroup
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: !Sub 'Internal Security group for ${AWS::StackName}'
```

```
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 8080
        ToPort: 8080
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
```

main.yml

Line #6: `!Sub` is a CloudFormation function that performs string interpolation. Here, we interpolate the stack name into the security group description.

Line #6: `AWS::StackName` is a CloudFormation pseudo parameter. There are many other useful ones.

Line #16: Tags are great. There are many ways to use them. At the very least, it makes sense to tag most resources with the stack name if you are going to have multiple stacks in the same AWS account.

Line #18: `!Ref` is a CloudFormation function for referring to other resources in your stack.

The next resource we'll create is an IAM role, which our EC2 instance will use to define its permissions. At this point, our application doesn't need much, as it isn't using any AWS services yet. For now, we will grant our instance role full access to AWS CloudWatch, but there are many other managed polices, which you can choose based on what permissions your application needs.

```
Resources:
  SecurityGroup: ...

  InstanceRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          Effect: Allow
          Principal:
            Service:
              - "ec2.amazonaws.com"
          Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/CloudWatchFullAccess
      Tags:
```

```
      - Key: Name
        Value: !Ref AWS::StackName
```

main.yml

Next, we'll create an instance profile to tie our IAM role to the EC2 instance that we'll create.

```
Resources:
  SecurityGroup: ...
  InstanceRole: ...

  InstanceProfile:
    Type: "AWS::IAM::InstanceProfile"
    Properties:
      Roles:
        - Ref: InstanceRole
```

main.yml

Now it's time to create our final resource, the EC2 instance itself.

```
Resources:
  SecurityGroup: ...
  InstanceRole: ...
  InstanceProfile: ...

  Instance:
    Type: AWS::EC2::Instance
    CreationPolicy:
      ResourceSignal:
        Timeout: PT15M
        Count: 1
    Metadata:
      AWS::CloudFormation::Init:
        config:
          packages:
            yum:
              wget: []
              unzip: []
    Properties:
      ImageId: !Ref EC2AMI
      InstanceType: !Ref EC2InstanceType
      IamInstanceProfile: !Ref InstanceProfile
      Monitoring: true
      SecurityGroupIds:
        - !GetAtt SecurityGroup.GroupId
      UserData:
        # ...
      Tags:
        - Key: Name
          Value: !Ref AWS::StackName
```

main.yml

Line #8: This tells CloudFormation to wait for a signal before marking the new

Line #8: This tells CloudFormation to wait for a signal before marking the new instance as created (we'll see how in the install script).

Line #15: Here we define some prerequisites that CloudFormation will install on our instance (the `wget` and `unzip` utilities). We'll need them to install our application.

Line #20: The AMI ID that we take as a template parameter.

Line #21: The EC2 instance type that we take as a template parameter.

Line #25: `!GetAtt` is a CloudFormation function that can reference attributes from other resources.

Line #27: See the next code listing for how to fill in this part.

Next, let's fill in the `UserData` section for the EC2 instance. This allows us to run commands on our instance when it launches.

```
UserData:
  Fn::Base64: !Sub |
    #!/bin/bash -xe

    # send script output to /tmp so we can debug boot failures
    exec > /tmp/userdata.log 2>&1

    # Update all packages
    yum -y update

    # Get latest cfn scripts; https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-
    yum install -y aws-cfn-bootstrap

    # Have CloudFormation install any files and packages from the metadata
    /opt/aws/bin/cfn-init -v --stack ${AWS::StackName} --region ${AWS::Region} --resource Instance

    cat > /tmp/install_script.sh << EOF
      # START
      echo "Setting up NodeJS Environment"
      curl https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash

      # Dot source the files to ensure that variables are available within the current shell
      . /home/ec2-user/.nvm/nvm.sh
      . /home/ec2-user/.bashrc

      # Install NVM, NPM, Node.JS
      nvm alias default v12.7.0
      nvm install v12.7.0
      nvm use v12.7.0

      # Download latest code, unzip it into /home/ec2-user/app
      wget https://github.com/<username>/aws-bootstrap/archive/master.zip
      unzip master.zip
      mv aws-bootstrap-master app
```

```
    # Create log directory
    mkdir -p /home/ec2-user/app/logs


    # Run server
    cd app
    npm install
    npm start
  EOF

  chown ec2-user:ec2-user /tmp/install_script.sh && chmod a+x /tmp/install_script.sh
  sleep 1; su - ec2-user -c "/tmp/install_script.sh"

  # Signal to CloudFormation that the instance is ready
  /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} --region ${AWS::Region} --resource Ins
```

main.yml

Line #6: The output of the `UserData` script will be written to `/tmp/userdata.log`. Look there if you need to debug any launch issues.

Line #15: This is where the `wget` and `unzip` utilities will get installed.

Line #17: This script replaces the manual setup we ran in the previous section.

Line #32: Replace `<username>` with your GitHub username.

Line #46: Runs the install script as the ec2-user.

Line #49: Signals to CloudFormation that the instance setup is complete.

In the next lesson, we will deploy our CloudFormation stack.