Automatic Deployments: Install CodeDeploy Agent on EC2

We will be installing CodeDeploy agent on our EC2 instance in this lesson.

We'll cover the following

Objective

- Automatically update our application when a change gets pushed to GitHub.

Steps

- Install the CodeDeploy agent on our EC2 instance.

The deployment pipeline

Now it's time to create the deployment pipeline in CloudFormation. Let's start by setting up a few environment variables in our `deploy-infra.sh` script with information about our GitHub credentials.

```
# Generate a personal access token with repo and admin:repo_hook
#    permissions from https://github.com/settings/tokens
GH_ACCESS_TOKEN=$(cat ~/.github/aws-bootstrap-access-token)
GH_OWNER=$(cat ~/.github/aws-bootstrap-owner)
GH_REPO=$(cat ~/.github/aws-bootstrap-repo)
GH_BRANCH=master
```

deploy-infra.sh

And then we can pass these variables to our `main.yml` script as parameters.

```
# 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 \
    GitHubOwner=$GH_OWNER \
    GitHubRepo=$GH_REPO \
    GitHubBranch=$GH_BRANCH \
    GitHubPersonalAccessToken=$GH_ACCESS_TOKEN \
    CodePipelineBucket=$CODEPIPELINE_BUCKET
```

deploy-infra.sh

To do that, we also need to update the `Parameters` section in `main.yml` to receive the GitHub information.

```
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'
  CodePipelineBucket:
    Type: String
    Description: 'The S3 bucket for CodePipeline artifacts.'
  GitHubOwner:
    Type: String
    Description: 'The username of the source GitHub repo.'
  GitHubRepo:
    Type: String
    Description: 'The source GitHub repo name (without the username).'
  GitHubBranch:
    Type: String
    Default: master
    Description: 'The source GitHub branch.'
  GitHubPersonalAccessToken:
    Type: String
    NoEcho: true
    Description: 'A GitHub personal access token with "repo" and "admin:repo_hook" permissions.'
```

main.yml

Next, we need to add a new managed policy to allow our EC2 instance to access CodeDeploy.

```
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
      - arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforAWSCodeDeploy
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
```

main.yml

Line #14: Allows our EC2 instance to access CodeDeploy.

We also need to create a new IAM role to allow the CodeBuild, CodeDeploy, and CodePipeline services to access our AWS resources.

```
DeploymentRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        Effect: Allow
        Principal:
          Service:
            - codepipeline.amazonaws.com
            - codedeploy.amazonaws.com
            - codebuild.amazonaws.com
        Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/PowerUserAccess
```

main.yml

Then we can define our CodeBuild project.

```
BuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: !Ref AWS::StackName
    ServiceRole: !GetAtt DeploymentRole.Arn
    Artifacts:
      Type: CODEPIPELINE
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:2.0
    Source:
      Type: CODEPIPELINE
```

main.yml

Next, we can define our CodeDeploy application. This lets CodeDeploy know that our deployment target is EC2.

```
DeploymentApplication:
  Type: AWS::CodeDeploy::Application
  Properties:
    ApplicationName: !Ref AWS::StackName
    ComputePlatform: Server
```

main.yml

Line #5: In this case, `Server` means EC2.

To complete the CodeDeploy setup, we also need to define a deployment group. For now, we're going to have one deployment group called *Staging*. This will be our pre-production environment. We will add another deployment group for production when we get to the Production section.

```
StagingDeploymentGroup:
  Type: AWS::CodeDeploy::DeploymentGroup
  DependsOn: Instance
  Properties:
    DeploymentGroupName: staging
    ApplicationName: !Ref DeploymentApplication
    DeploymentConfigName: CodeDeployDefault.AllAtOnce
    ServiceRoleArn: !GetAtt DeploymentRole.Arn
    Ec2TagFilters:
      - Key: aws:cloudformation:stack-name
        Type: KEY_AND_VALUE
        Value: !Ref AWS::StackName
```

main.yml

Line #7: For pre-production, we can choose to deploy as fast as possible. We'll do this differently in production.

Line #9: These filters define how CodeDeploy will find the EC2 instances to deploy to.

And finally, we just need to define our pipeline. Let's do that in the next lesson.