Load Balancing: Add a second EC2 Instance

We'll cover the following

Objective

- Run our application on more than one EC2 instance.

Steps

- Add a second EC2 instance.

Currently, our application is running on a single EC2 instance. To allow our application to scale beyond the capacity of a single instance, we need to introduce a load balancer that can direct traffic to multiple instances.

Adding a second instance

We could naively add a second instance by simply duplicating the configuration we have for our existing instance. But that would create a lot of configuration duplication. Instead, we're going to pull the bulk of the EC2 configuration into an EC2 launch template, and then we'll simply reference the launch template from both instances.

We can almost copy our EC2 instance configuration into a new launch template resource as is, but there are slight differences between the two specifications. In addition, we'll also need to change the `cfn-init` and `cfn-signal` calls at the end of the `UserData` script to dynamically determine the instance ID at runtime.

```
InstanceLaunchTemplate:
  Type: AWS::EC2::LaunchTemplate
  Metadata:
    AWS::CloudFormation::Init:
```

```
      config:
        packages:
          yum:
            ruby: []
            jq: []
        files:
          /home/ec2-user/install:
            source: !Sub "https://aws-codedeploy-${AWS::Region}.s3.amazonaws.com/latest/install"
            mode: "000755" # executable
        commands:
          00-install-cd-agent:
            command: "./install auto"
            cwd: "/home/ec2-user/"
  Properties:
    LaunchTemplateName: !Sub 'LaunchTemplate_${AWS::StackName}'
    LaunchTemplateData:
      ImageId: !Ref EC2AMI
      InstanceType: !Ref EC2InstanceType
      IamInstanceProfile:
        Arn: !GetAtt InstanceProfile.Arn
      Monitoring:
        Enabled: true
      SecurityGroupIds:
        - !GetAtt SecurityGroup.GroupId
      UserData:
        # ...
```

main.yml

Line #30: See the next code listing for how to fill in this part.

Now let's update the `UserData` script.

```
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-p
    yum install -y aws-cfn-bootstrap

    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
```

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

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

  # Query the EC2 metadata service for this instance's instance-id
  export INSTANCE_ID="`wget -q -O - http://169.254.169.254/latest/meta-data/instance-id`"

  # Query EC2 describeTags method and pull our the CFN Logical ID for this instance
  export LOGICAL_ID=`aws --region ${AWS::Region} ec2 describe-tags \
    --filters "Name=resource-id,Values=${!INSTANCE_ID}" \
              "Name=key,Values=aws:cloudformation:logical-id" \
    | jq -r ".Tags[0].Value"`

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

main.yml

Line #39: We're using the Instance Metadata service to get the instance id.

Line #41: Here, we're getting the tags associated with this instance. The `aws:cloudformation:logical-id` tag is automatically attached by CloudFormation. Its value is what we pass to `cfn-signal` to signal a successful launch.

Line #41: Note the usage of `${!INSTANCE_ID}`. Since this is inside a CloudFormation `!Sub`, if we used `${INSTANCE_ID}`, CloudFormation would have tried to do the substitution itself. Adding the `!` tells CloudFormation to rewrite it for `bash` to interpret.

Now, we can change our instance resource to reference the new launch template.

```
Instance:
  Type: AWS::EC2::Instance
  CreationPolicy:
    ResourceSignal:
      Timeout: PT5M
      Count: 1
  Properties:
    LaunchTemplate:
      LaunchTemplateId: !Ref InstanceLaunchTemplate
      Version: !GetAtt InstanceLaunchTemplate.LatestVersionNumber
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
```

main.yml

Line #10: Each time we update our launch template, it will get a new version number. We always want to use the latest.

Adding a second instance is now as easy as creating a new instance resource that references the same launch template.

```
Instance2:
  Type: AWS::EC2::Instance
  CreationPolicy:
    ResourceSignal:
      Timeout: PT5M
      Count: 1
  Properties:
    LaunchTemplate:
      LaunchTemplateId: !Ref InstanceLaunchTemplate
      Version: !GetAtt InstanceLaunchTemplate.LatestVersionNumber
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
```

main.yml

We also need to add an inline IAM policy to allow the `UserData` script to access the EC2 `DescribeTags` API. Let's modify the `InstanceRole` resource to add it.

```
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
    Policies:
      - PolicyName: ec2DescribeTags
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            - Effect: Allow
              Action: 'ec2:DescribeTags'
              Resource: '*'
    Tags:
      - Key: Name
        Value: !Ref AWS::StackName
```

main.yml

Line #16: This policy allows our instance to query EC2 for tags.

Now, we can change the output of our CloudFormation template to return a URL for both instances.

```
Outputs:
  InstanceEndpoint1:
    Description: The DNS name for the created instance
    Value: !Sub "http://${Instance.PublicDnsName}:8080"
    Export:
      Name: InstanceEndpoint1

  InstanceEndpoint2:
    Description: The DNS name for the created instance
    Value: !Sub "http://${Instance2.PublicDnsName}:8080"
    Export:
      Name: InstanceEndpoint2
```

main.yml

Finally, let's change our `deploy-infra.sh` script to give us these URLs.

```
# If the deploy succeeded, show the DNS name of the created instance
if [ $? -eq 0 ]; then
  aws cloudformation list-exports \
    --profile awsbootstrap \
    --query "Exports[?starts_with(Name,'InstanceEndpoint')].Value"
fi
```

deploy-infra.sh

If we run the `deploy-infra.sh` script now, we should see a pair of URLs when the deployment finishes.

```
./deploy-infra.sh


========== Deploying setup.yml ==========

Waiting for changeset to be created..

No changes to deploy. Stack awsbootstrap-setup is up to date


========== Deploying main.yml ==========

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - awsbootstrap
[
    "http://ec2-52-91-223-254.compute-1.amazonaws.com:8080",
    "http://ec2-3-93-145-152.compute-1.amazonaws.com:8080"
]
```

terminal

Our old instance should have been terminated, and two new instances should have started in its place.

At this point, let's also checkpoint our progress into GitHub.

```
git add main.yml deploy-infra.sh
git commit -m "Add a second instance"
git push
```

terminal

Next, let's modify our application a little bit to allow us to see how our requests get routed. We can do this by simply including the hostname in the response message.

```
const { hostname } = require('os');
const http = require('http');
const message = `Hello World from ${hostname()}\n`;
const port = 8080;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname()}:${port}/`);
});
```

server.js

Line #3: Changes the message to include the hostname.

Let's push this change to GitHub and wait for the deployment to finish.

```
git add server.js
git commit -m "Add hostname to hello world message"
git push
```

terminal

We can follow the deployment progress from the CodePipeline console. Once the deployment is complete, we can verify our change by making a request to both URLs.

```
curl http://ec2-52-91-223-254.compute-1.amazonaws.com:8080
Hello World from ip-10-0-113-245.ec2.internal
curl http://ec2-3-93-145-152.compute-1.amazonaws.com:8080
```

```
Hello World from ip-10-0-61-251.ec2.internal
```

terminal

> 🔍 The hostname that the server is printing is not the same as the *Public DNS* assigned to the instance. It is actually a private domain name that EC2 assigns to each instance. You can find this private domain in the EC2 console under the *Private DNS* field.

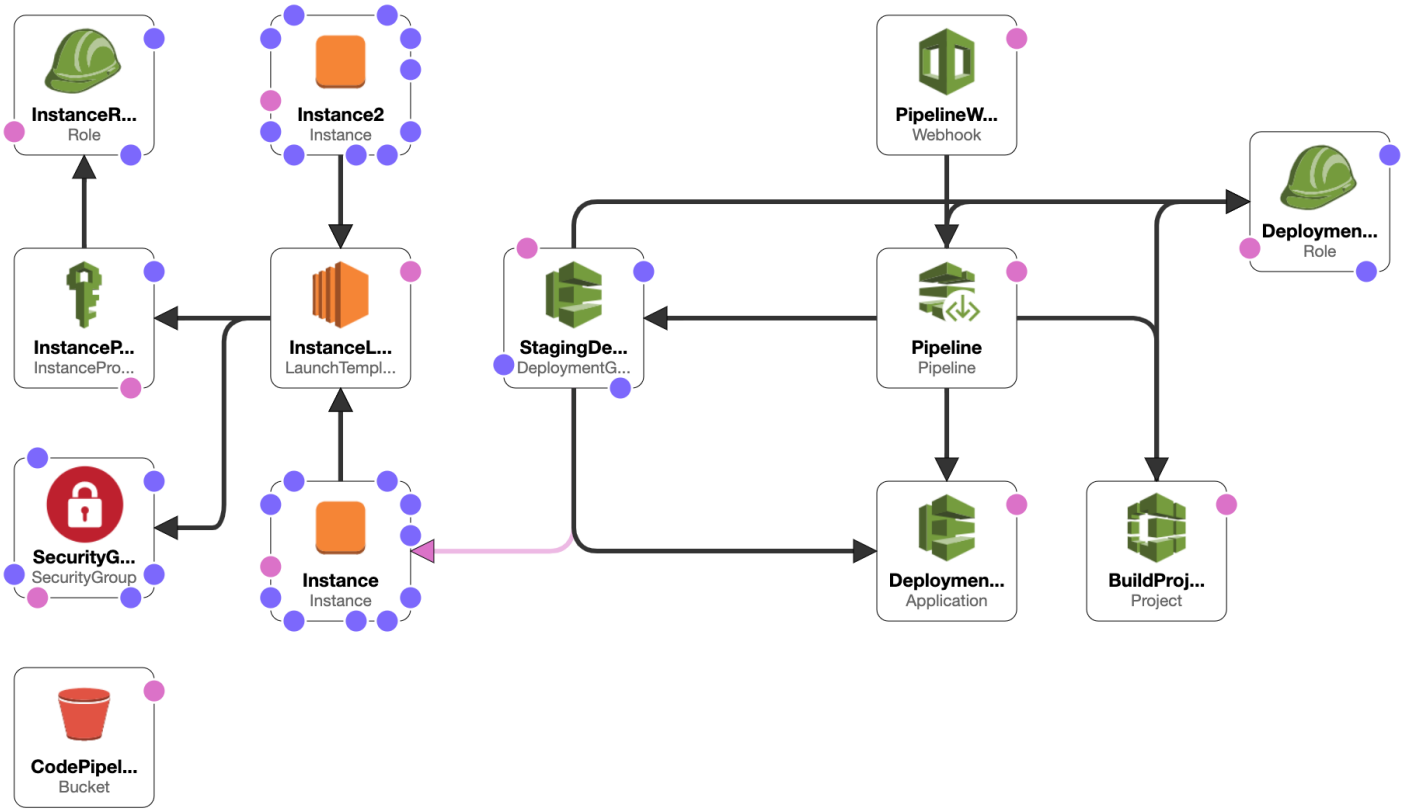
> **Note:** All the code has been already added and we are pushing it on our repository as well.

This code requires the following API keys to execute:

username	Not Specified...
AWS_ACCESS_KE...	Not Specified...
AWS_SECRET_AC...	Not Specified...
AWS_REGION	us-east-1
Github_Token	Not Specified...

```
const { hostname } = require('os');
const http = require('http');
const message = `Hello World from ${hostname()}\n`;
const port = 8080;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
  server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname()}:${port}/`);
});
```

In order to get a pictorial view of our developed cloudformation stack so far, below is the design view which shows the resources we created and their relationships.



Add a second EC2 instance

Let’s go ahead and add a load balancer to have a single endpoint for our application in the next part of this lesson.