Custom Domains: Map our Domain to Load Balancers

We'll cover the following

Objective

- Access our application from a custom domain.

Steps

- Map our domain to the load balancers.

Creating a DNS record per stage

Let's start by adding two new input parameters in our `stage.yml` to receive the stage domain and subdomain.

```
Domain:
  Type: String
SubDomain:
  Type: String
```

stage.yml

Then, let's add a resource to create a Route 53 A record that points `<subdomain>.<domain>` to the load balancer.

```
DNS:
  Type: AWS::Route53::RecordSet
  Properties:
    HostedZoneName: !Sub '${Domain}.'
    Name: !Sub '${SubDomain}.${Domain}.'
    Type: A
    AliasTarget:
      HostedZoneId: !GetAtt LoadBalancer.CanonicalHostedZoneID
      DNSName: !GetAtt LoadBalancer.DNSName
```

stage.yml

Next, let's change the stage output to return the URL with our custom domain rather than the load balancer's default endpoint.

```
LBEndpoint:
  Description: The DNS name for the stage
  Value: !Sub "http://${DNS}"
```

stage.yml

Then we also need to add an input parameter in `main.yml` to receive our custom domain name.

```
Domain:
  Type: String
```

main.yml

And finally, we need to pass our custom domain to the nested stacks.

```
Staging:
  Type: AWS::CloudFormation::Stack
  Properties:
    TemplateURL: stage.yml
    TimeoutInMinutes: 30
    Parameters:
      EC2InstanceType: !Ref EC2InstanceType
      EC2AMI: !Ref EC2AMI
      Domain: !Ref Domain
      SubDomain: staging

Prod:
  Type: AWS::CloudFormation::Stack
  Properties:
    TemplateURL: stage.yml
    TimeoutInMinutes: 30
    Parameters:
      EC2InstanceType: !Ref EC2InstanceType
      EC2AMI: !Ref EC2AMI
      Domain: !Ref Domain
      SubDomain: prod
```

main.yml

Line #9 and #20: Passes the domain name to the nested stack.

Line #10 and #21: Passes a stack-specific subdomain to the nested stack.

At this point, let's add our domain name as an environment variable at the top of

`deploy-infra.sh`.

```
DOMAIN=the-good-parts.com
```

deploy-infra.sh

Line #1: Replace with your domain name.

And now we can pass our domain to the CloudFormation template.

```
# Deploy the CloudFormation template
echo -e "\n\n=========== Deploying main.yml ==========="
aws cloudformation deploy \
  --region $REGION \
  --profile $CLI_PROFILE \
  --stack-name $STACK_NAME \
  --template-file ./cfn_output/main.yml \
  --no-fail-on-empty-changeset \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides \
    EC2InstanceType=$EC2_INSTANCE_TYPE \
    Domain=$DOMAIN \
    GitHubOwner=$GH_OWNER \
    GitHubRepo=$GH_REPO \
    GitHubBranch=$GH_BRANCH \
    GitHubPersonalAccessToken=$GH_ACCESS_TOKEN \
    CodePipelineBucket=$CODEPIPELINE_BUCKET
```

deploy-infra.sh

Line #12: Passes the domain name to `main.yml`.

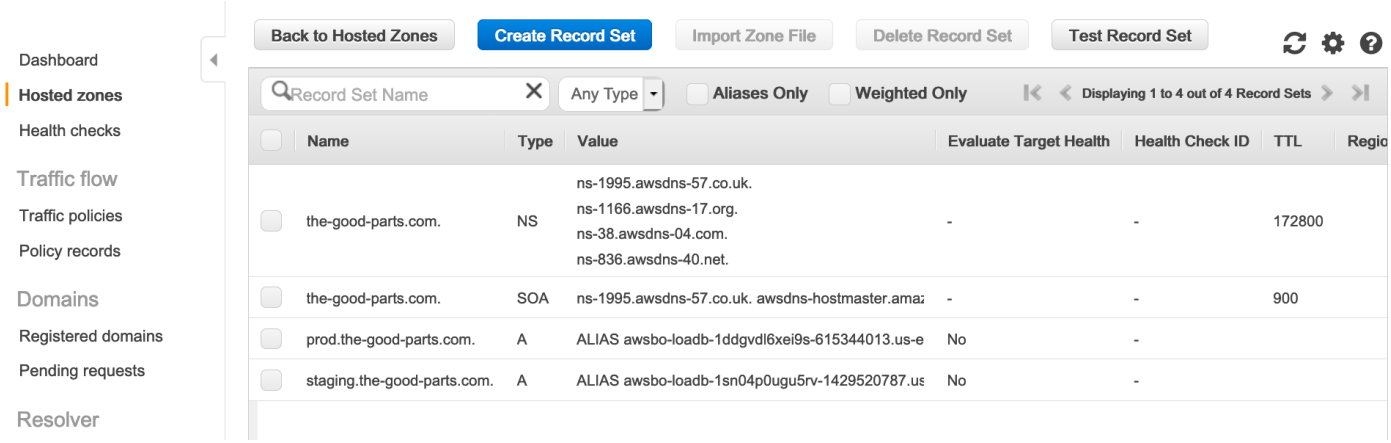
Let's deploy to see our custom domain in action.

```
./deploy-infra.sh


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

Waiting for changeset to be created..

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


=========== Packaging main.yml ===========


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

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - awsbootstrap
[
    "http://prod.the-good-parts.com",
```

```
    "http://staging.the-good-parts.com"
]
```

terminal

And now have a much more human-friendly endpoint for our two stages. We should also be able to see the A records in our Route 53 hosted zone.



New A Records

The DNS propagation can take a few minutes. After a while, we should be able to reach our application through our custom domain.

```
for run in {1..20}; do curl -s staging.the-good-parts.com; done | sort | uniq -c
9 Hello World from ip-10-0-102-103.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
11 Hello World from ip-10-0-61-182.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
```

terminal

```
for run in {1..20}; do curl -s prod.the-good-parts.com; done | sort | uniq -c
10 Hello World from ip-10-0-46-233.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
10 Hello World from ip-10-0-77-238.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
```

terminal

> 🔍 If the `curl` commands work, but your browser times out trying to connect, it may be trying to upgrade to HTTPS in order to provide better security. You can try from another browser or wait until we enable HTTPS in the next section.

Now we can commit all our changes to checkpoint our progress.

```
git add deploy-infra.sh main.yml stage.yml
```

```
git commit -m "Add a custom domain"
git push
```

terminal

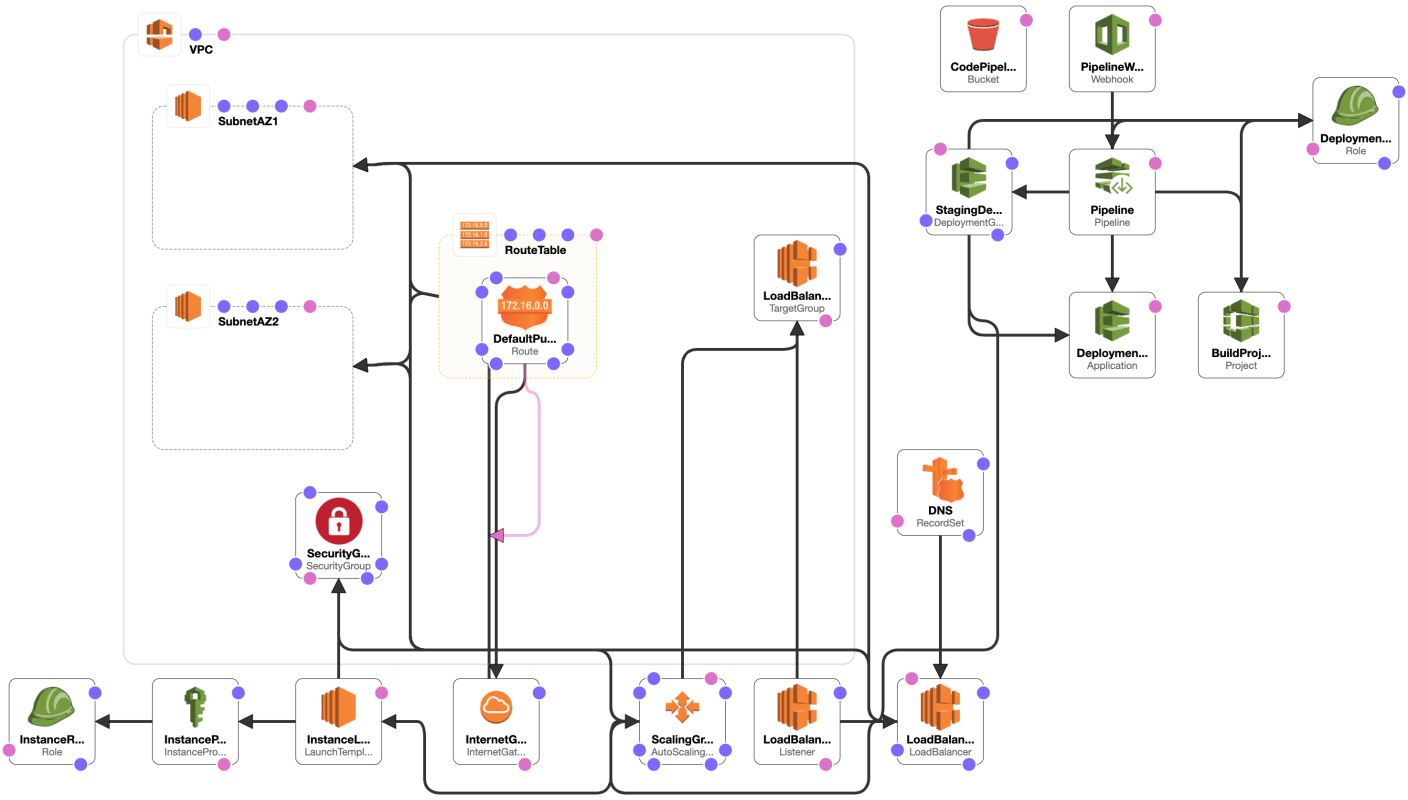
> **Note:** All the code has been already added and we are pushing it on our repository as well.

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...

```
{
  "name": "aws-bootstrap",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws --log ../logs/app
    "stop": "node ./node_modules/pm2/bin/pm2 stop hello_aws",
    "build": "echo 'Building...'"
  },
  "dependencies": {
    "pm2": "^4.2.0"
  }
}
```

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.



Custom Domain

In the next lesson, we will migrate our endpoint from HTTP to HTTPS.