Production: Create Production Stack

We'll cover the following

Objective

- Create separate environments for production.

Steps

- Create a separate prod stack for production.

Adding the prod stage

Extracting the staging stack out of the main template was most of the work, so adding prod is going to be quite straightforward.

First, let's add a deployment group so that our production hosts can be configured for deployment.

```
ProdDeploymentGroup:
  Type: AWS::CodeDeploy::DeploymentGroup
  Properties:
    DeploymentGroupName: prod
    AutoScalingGroups:
      - !GetAtt Prod.Outputs.ScalingGroup
    ApplicationName: !Ref DeploymentApplication
    DeploymentConfigName: CodeDeployDefault.OneAtATime
    ServiceRoleArn: !GetAtt DeploymentRole.Arn
```

main.yml

Line #8: We could have chosen to put the deployment groups in `stage.yml`, but it is often useful to have different deployment configurations for different stages. For example, here we set the prod deployment to go on one host at a time, whereas the staging deployment was set to go all at once.

Next, we'll add a new stage in our `Pipeline` resource called `Prod` at the end of the list.

```
- Name: Prod
  Actions:
    - Name: Prod
      InputArtifacts:
        - Name: Build
      ActionTypeId:
        Category: Deploy
        Owner: AWS
        Version: 1
        Provider: CodeDeploy
      Configuration:
        ApplicationName: !Ref DeploymentApplication
        DeploymentGroupName: !Ref ProdDeploymentGroup
      RunOrder: 1
```

main.yml

Then we'll add the new nested stack to `main.yml`.

```
Prod:
  Type: AWS::CloudFormation::Stack
  DependsOn: Staging
  Properties:
    TemplateURL: stage.yml
    TimeoutInMinutes: 30
    Parameters:
      EC2InstanceType: !Ref EC2InstanceType
      EC2AMI: !Ref EC2AMI
```

main.yml

Line #3: Updates to the prod stack will not be enacted until the staging stack successfully applies stack updates.

Finally, we need to add an output in `main.yml` to return the prod endpoint.

```
ProdLBEndpoint:
  Description: The DNS name for the prod LB
  Value: !GetAtt Prod.Outputs.LBEndpoint
  Export:
    Name: ProdLBEndpoint
```

And now let's deploy.

```
./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://awsbo-LoadB-1DDGVDL6XEI9S-615344013.us-east-1.elb.amazonaws.com:80",
    "http://awsbo-LoadB-1SN04P0UGU5RV-1429520787.us-east-1.elb.amazonaws.com:80"
]
```

terminal

The staging endpoint should still be working when the deployment finishes, but our new production endpoint won't be yet. This is because the deployment groups are configured so that the ASG will launch all new instances with the most recent revision deployed. But there will be no recent revision until the first proper deployment occurs. To fix this, we need to go to our pipeline and click *Release Changes* in order to get a deployment to prod. Once deployment finishes, then the prod hosts should start responding as well.

```
for run in {1..20}; do curl -s http://awsbo-LoadB-1DDGVDL6XEI9S-615344013.us-east-1.elb.amazonaws.
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

Let's wrap this up by pushing these changes to our GitHub repository.

```
git add main.yml
git commit -m "Add prod nested stack"
git push
```

terminal

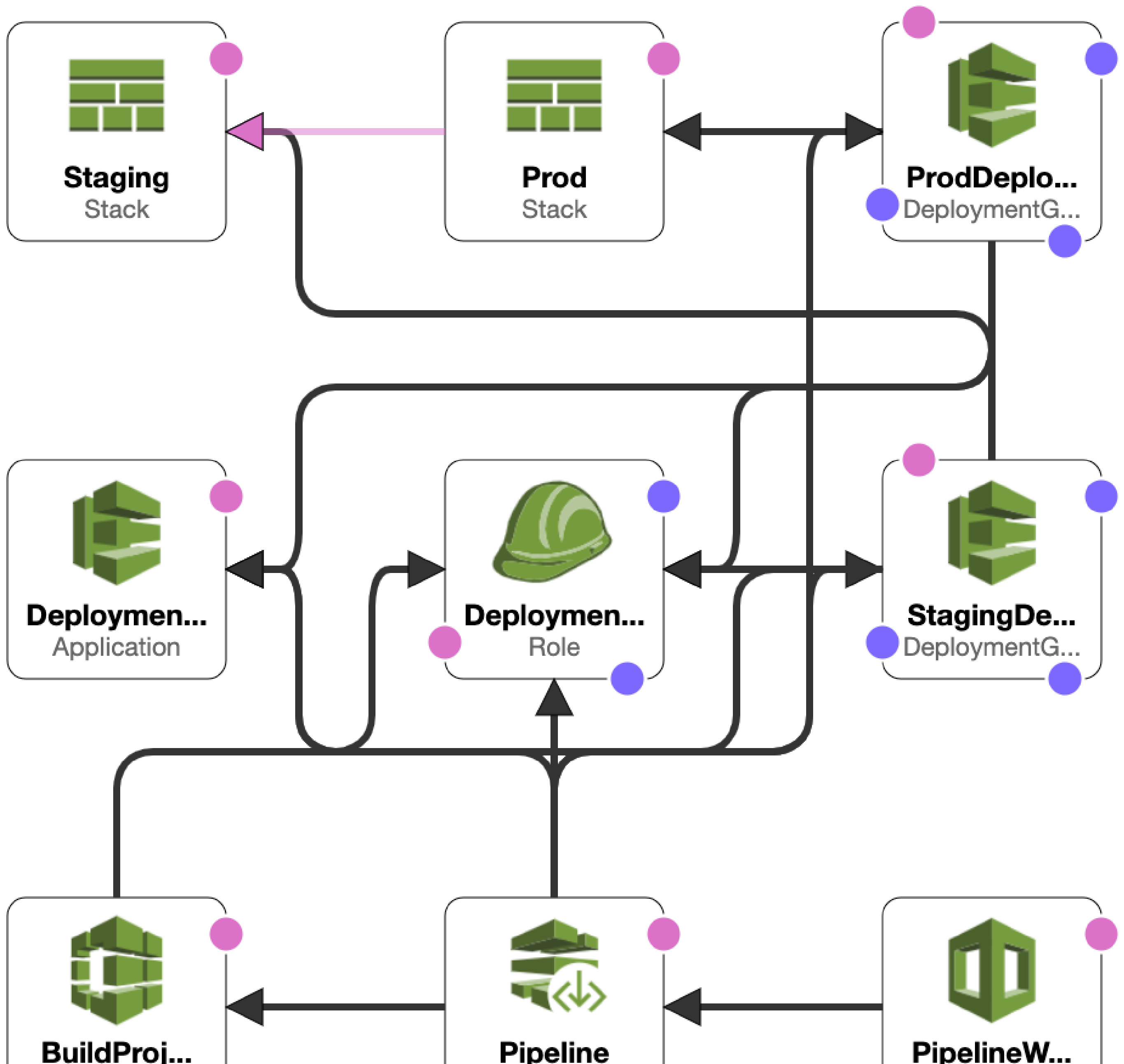
This code requires the following API keys to execute:

username	Not Specified...
AWS_ACCESS_KE...	Not Specified...
AWS_SECRET_AC...	Not Specified...

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

```
{
  "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.



Staging & Production stack

In the next lesson, we will access our application from a custom domain.