Production: Add Stack Name to our Application

We will add the stack name to our Hello World application in this lesson so that our application will know what environment it is running in.

We'll cover the following

- Objective
- Steps
- Adding the stack name to Hello World

Objective#

- Create separate environments for staging.

Steps#

- Adding the stack name to our Hello World application.

In the real world, we will generally want to have at least one environment to test our application and infrastructure before rolling changes out to production. A common model is to call the testing environment ‘staging’ and production ‘prod’. We’ll set these up next.

In this and upcoming two lessons, we will decommission our existing infrastructure and replace it with two CloudFormation nested stacks infrastructure and replace it with two CloudFormation nested stacks representing our staging and prod environments. We will also update our

CodePipeline so that it will promote updates to prod only after they've successfully passed through staging.

Adding the stack name to Hello World#

Let's start by making a small change to the `start-service.sh` script so that our application will know what environment it is running in.

```
#!/bin/bash -xe
source /home/ec2-user/.bash_profile
cd /home/ec2-user/app/release

# Query the EC2 metadata service for this instance's region
REGION="`wget -qO- http://instance-data/latest/meta-data/placement/availability-
# 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/instan
# Query EC2 describeTags method and pull our the CFN Logical ID for this instanc
export STACK_NAME=`aws --region $REGION ec2 describe-tags \
  --filters "Name=resource-id,Values=${INSTANCE_ID}" \
            "Name=key,Values=aws:cloudformation:stack-name" \
  | jq -r ".Tags[0].Value"`

npm run start
```

start-service.sh

Then, we also need to update the `start` script in `package.json` to pass the stack name environment variable to the application.

```
node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws --log ../logs
```

package.json

Now, let’s change the response in `server.js` to include the stack name.

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

server.js

Let’s push these changes to GitHub.

```
git add start-service.sh package.json server.js
git commit -m "Add stack name to server output"
git push
```

terminal

Finally, let’s wait for the changes to go through the pipeline, and we should see our stack name when we hit our application’s endpoint.

```
for run in {1..20}; do curl -s http://awsbo-LoadB-13F2DS4LKSCVO-10652175.us-east
9 Hello World from ip-10-0-50-202.ec2.internal in awsbootstrap
11 Hello World from ip-10-0-68-58.ec2.internal in awsbootstrap
```

terminal

Now in the next lesson, we will create our nested stack for staging.

← Back

Scaling: Remove Instances

Next →

Production: Create Staging Stack

☑ Mark as Completed

Report an Issue