Gradual Deployments

In this lesson, you will get familiar with routing configurations and AWS CodeDeploy as well as how AWS gradually deploys code.

WE'LL COVER THE FOLLOWING

Routing configurations and AWS CodeDeploy

An alias always points to some numerical version, but it can also point to more than one version at the same time. This becomes incredibly useful for safe deployments. Lambda supports automatic load balancing between versions assigned to the same alias, using a feature called *routing configuration*.

Another AWS deployment product, called AWS CodeDeploy, can modify the routing configuration over time to gradually switch aliases between several versions of code and infrastructure. CodeDeploy can, for example, use the new version of an application only for 10% of the traffic and wait for a short period while monitoring for unexpected problems. If everything looks OK, CodeDeploy can expose the new version to everyone and shut down the old version. On the other hand, if the new version seems to be problematic, CodeDeploy will move all the users back to the old infrastructure and destroy the experimental version. Reassigning aliases to published configuration versions is very quick (much faster than a redeployment) making it easy and quick to recover from deployment errors.

SAM automates most of the heavy
Application developers just need to

SAM automates most of the heavy lifting when setting up gradual deployments. It will automatically create CodeDeploy resources for Lambda functions, set up security permissions, and configure aliases and routing configurations. CloudFormation manages all resources in a template as a group, so if a single function causes gradual deployment to roll back, all the other resources will be automatically restored to the previous compatible state. Because Lambda pricing is based on actual utilisation, not reserved capacity, having a gradual deployment does not cost more than just using a single version. The experimental infrastructure and the old one will split the requests, so the combined utilisation is the same. The only downside to adding gradual deployments is that an application update takes longer, in order to run the experiment and decide whether the new version is good enough to keep.

Application developers just need to define what exactly it means for a deployment to be problematic. For example, you could run an experimental version for a few minutes and check for Lambda errors or time-outs. This is an easy way to prevent integration problems that were not detected during testing. Alternatively, you could run the experiment over a longer period of time and measure user behaviour, such as funnel conversion or purchases. For example, if the unit and integration tests were passed, but something unexpected is causing users to buy less with the new version, would it not be smart to automatically protect revenue by rolling back the update and alert someone to investigate it? You can set up those kinds of checks with CodeDeploy.

Modifying routing configurations manually

SAM only manages routing configuration for deployments. If you want to play with this feature outside deployments, for example to run A/B tests in production, configure the alias routing configuration using the AWS command line tools. Use the `aws lambda update-alias` command. You can do this even for functions created using SAM.

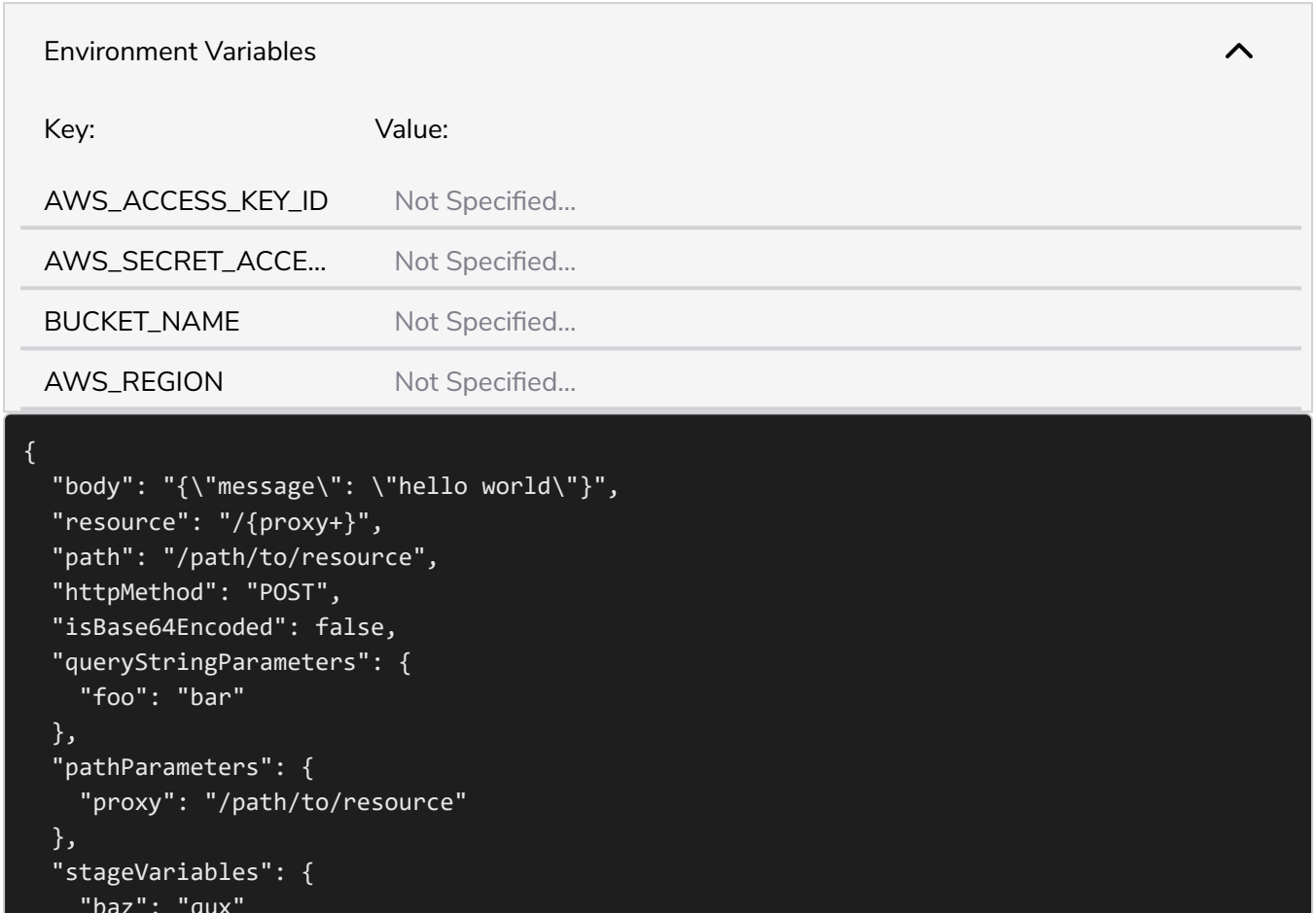
To set up gradual deployment, add the `DeploymentPreference` section to the function properties. This should be on the same indentation level as the other function properties. To avoid formatting issues, add the section immediately below the `AutoPublishAlias`. (Routing configurations only work with aliases, not numerical versions, so the alias configuration needs to stay in the template.)

```
DeploymentPreference:
  Type: Linear10PercentEvery1Minute
```

app/template.yaml

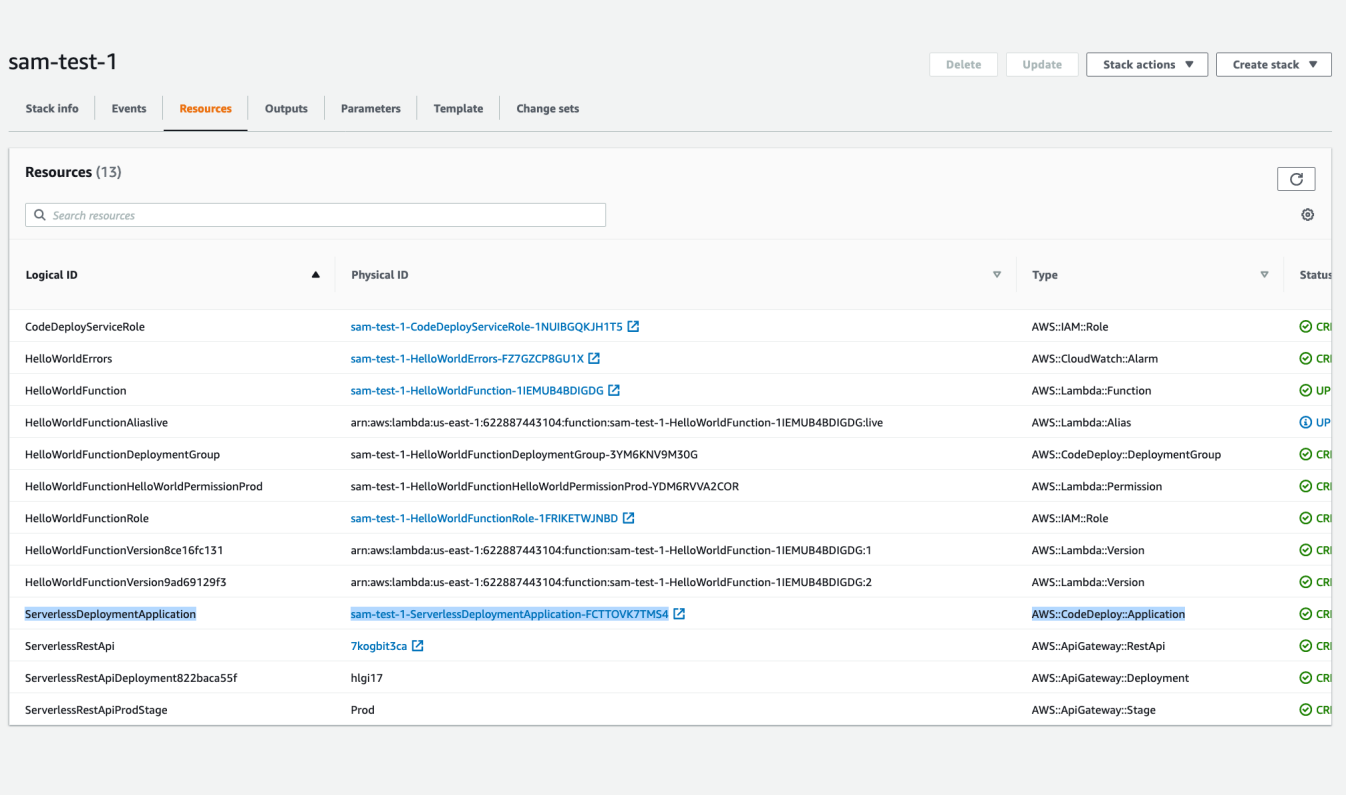
Change the welcome message in `hello-world/app.js` so you can see the difference between versions (for example, make it `hello world 3`). Now press the **Run** button which will build, package, and deploy the stack again.

Note that this deployment will take significantly longer than usual (about 10 minutes), because of the gradual traffic shifting. During deployment, try accessing the web page of your application. You should see the old message nine out of 10 times, but the new message should also show occasionally. The deployment should last roughly 10 minutes. As the traffic shifts, the new message should show up more frequently.



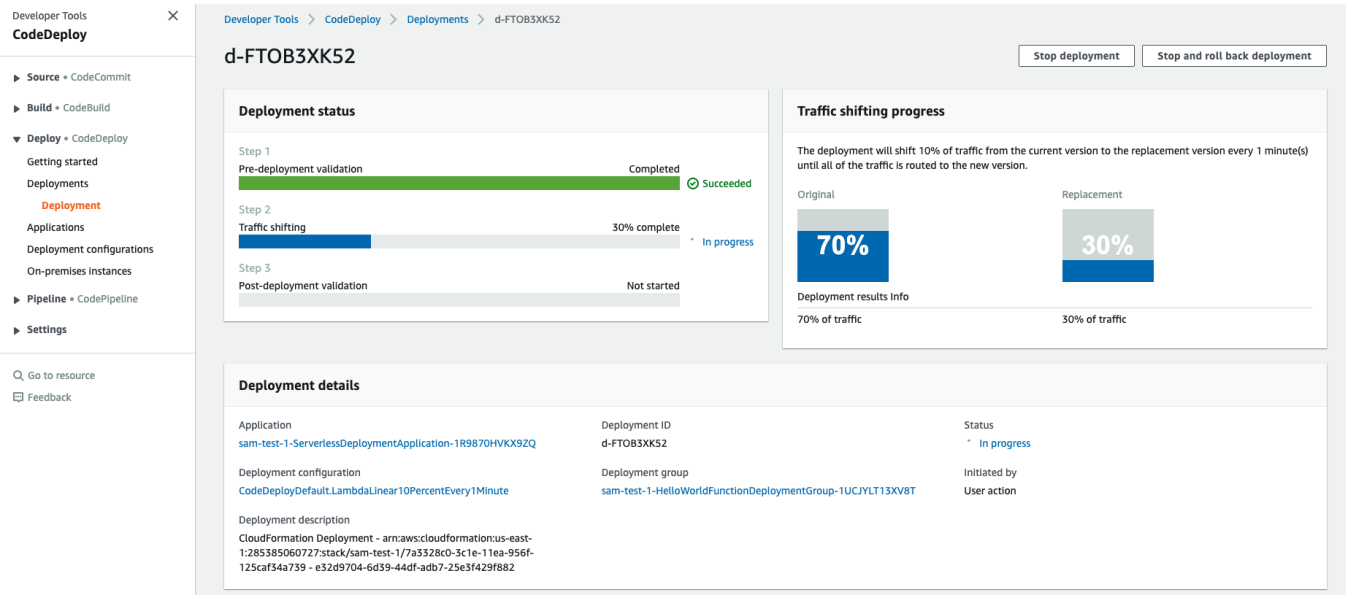
```
  },
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, sdch",
    "Accept-Language": "en-US,en;q=0.8",
    "Cache-Control": "max-age=0",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Desktop-Viewer": "true",
    "CloudFront-Is-Mobile-Viewer": "false",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Viewer-Country": "US",
    "Host": "1234567890.execute-api.us-east-1.amazonaws.com",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Custom User Agent String",
    "Via": "1.1 08f323deadbeefa7af34d5feb414ce27.cloudfront.net (CloudFront)",
    "X-Amz-Cf-Id": "cDehVQoZnx43VYQb9j2-nvCh-9z396Uhbp027Y2JvkCPNLmGJHqlaA==",
    "X-Forwarded-For": "127.0.0.1, 127.0.0.2",
    "X-Forwarded-Port": "443",
    "X-Forwarded-Proto": "https"
  },
  "requestContext": {
    "accountId": "123456789012",
    "resourceId": "123456",
    "stage": "prod",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "requestTime": "09/Apr/2015:12:34:56 +0000",
    "requestTimeEpoch": 1428582896000,
    "identity": {
      "cognitoIdentityPoolId": null,
      "accountId": null,
      "cognitoIdentityId": null,
      "caller": null,
      "accessKey": null,
      "sourceIp": "127.0.0.1",
      "cognitoAuthenticationType": null,
      "cognitoAuthenticationProvider": null,
      "userArn": null,
      "userAgent": "Custom User Agent String",
      "user": null
    },
    "path": "/prod/path/to/resource",
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "apiId": "1234567890",
    "protocol": "HTTP/1.1"
  }
}
```


SAM command line tools do not provide any indication of gradual deployment progress, but you can see traffic shifting in action in the AWS Web Console. Find the application stack in the AWS CloudFormation Web Console during deployment, and open the *Resources* tab. Within the resources, you'll see that SAM created a CodeDeploy application (resource type `AWS::CodeDeploy::Application`).



CloudFormation console shows a link to a CodeDeploy task in progress.

There should be a hyperlink in the same row as that resource. Click the link, and AWS Console will open the deployment application details. Switch to the `Deployments` tab, and you should see a single deployment in progress. Click the link in the *Deployment Id* column, and you'll see the status of your gradual deployment.



CodeDeploy has started sending a small percentage of requests to the new version.

CodeDeploy with existing aliases

> If you set up aliases and CodeDeploy during the same deployment, CodeDeploy won't do much initially, because it relies on updating an existing alias and shifting traffic from the old version associated with it. This is why you first created an alias and then added a deployment preference in this chapter.

Linear and canary deployments

The `Type` property inside `DeploymentPreference` controls how the traffic moves from the old version to the new one. SAM can automate two types of deployment preferences: linear and canary.

Linear deployments work by incrementally moving traffic during a period of time. In this example, you set it to `Linear10PercentEvery1Minute`. This, unsurprisingly, turns on the tap slightly more every minute, so that the new version starts with only 10% of the requests and each minute gets 10% more. That is why the new message will show up more frequently as the deployment progresses.

Canary deployments work in two steps. At the start of the deployment, CodeDeploy sets up routing so that the new version gets a specific percentage of requests. If there are no problems until the end of the deployment, the alias is completely assigned to the new version. For example, `Canary10Percent15Minutes` would send 10% of the traffic to the new version at the start, wait 15 minutes, and then move the remaining 90% to the new version.

See the AWS Documentation page on Gradual Code Deployments for an up-to-date list of all the deployment preference types.

In the next lesson, you will look at how to add alerts for your deployments.