Adding Deployment Alerts

Here, you'll set up alerts for some metrics using CloudWatch for your application!

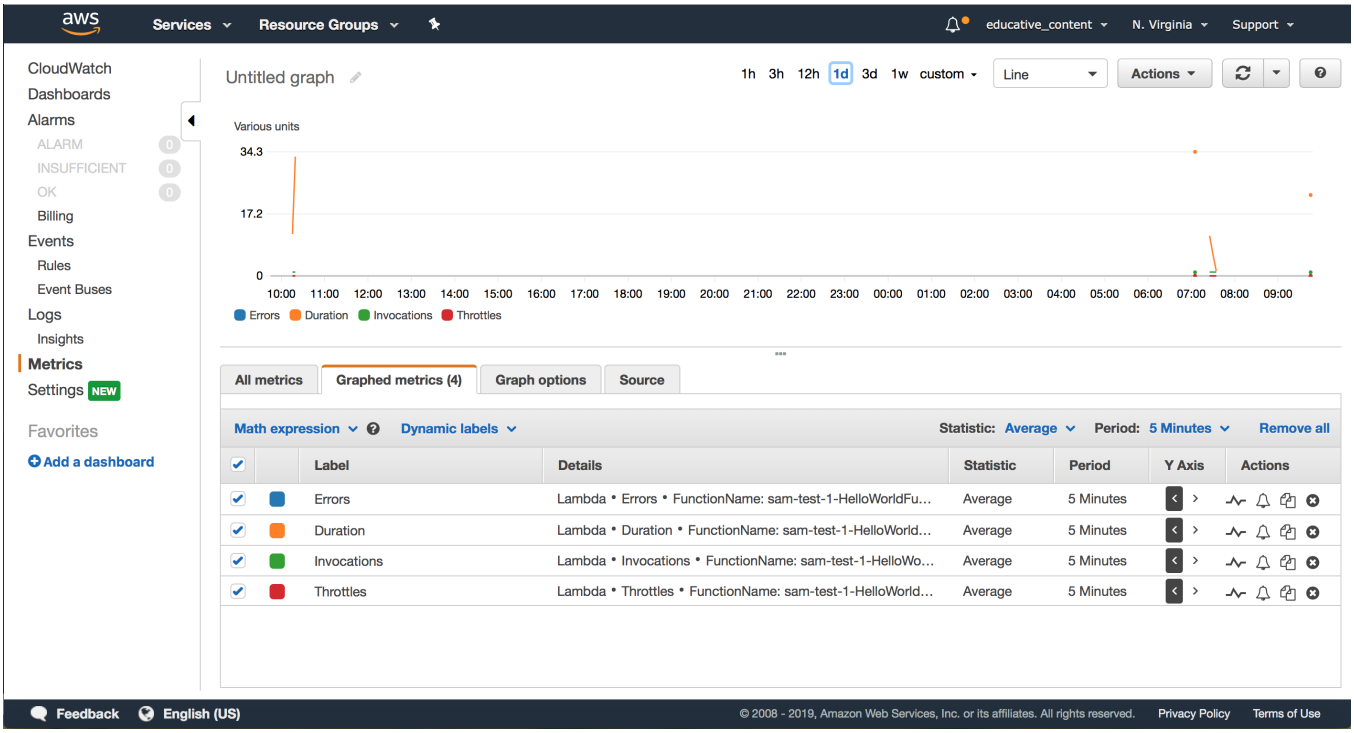
WE'LL COVER THE FOLLOWING

CloudWatch

So far, you haven’t set up any tests o compare the old and new versions. You could manually try things out during the deployment and then roll back in case of problems from the CodeDeploy Web Console, but that’s not really efficient or sustainable. It would be better if CodeDeploy did this automatically.

Luckily, CloudWatch can look out for errors automatically. You used CloudWatch to access logs in the previous chapter, but you can also use it to inspect and monitor operational statistics about your application. CloudWatch automatically tracks lots of interesting information about AWS Lambda and API Gateway, such as latency, number of invocations, and number of errors. Here is how you can see this information:

1. Go to the AWS Web Console, https://aws.amazon.com, and sign in if necessary.
2. Select *CloudWatch* in the list of services.
3. Select *Metrics* in the left-hand menu.
4. In the list of metric namespaces, select *Lambda* then *By Function name*, and select check boxes next to errors and invocations for your function.



CloudWatch automatically tracks useful operational metrics for Lambda functions and API Gateway resources.

Setting alarms

Monitoring changes over time might be interesting and even amusing, but you should not rely on human interaction to spot errors. CloudWatch can automatically send notifications if metrics exceed a certain threshold, using a feature called *alarms*. You could set up an alarm to send someone an email in case user requests start taking too long to execute, or even invoke a Lambda function when something bad happens.

To create an alarm manually, switch to the *Graphed metrics* tab, then click the bell icon in the *Actions* column next to a metric row in the bottom table on the page. You can, for example, use that to send yourself an email if a Lambda function starts throwing errors.

Of course, you can also set up alarms directly in a CloudFormation template. SAM does not have any specific shortcuts for that, so you'll just use the standard AWS::CloudWatch::Alarm resource. You will add the block from the following example to your application template, in the `Resources` section (it should be at the same indentation level as the `HelloWorldFunction`).

Note that you will use the source code for this course (`ch5` folder) so this has already been done for you.

```
HelloWorldErrors:
  Type: AWS::CloudWatch::Alarm
  Properties:
    MetricName: Errors
    Statistic: Sum
    ComparisonOperator: GreaterThanThreshold
    Threshold: 5
    Period: 60
    EvaluationPeriods: 1
    TreatMissingData: notBreaching
    Namespace: "AWS/Lambda"
    Dimensions:
      - Name: FunctionName
        Value: !Ref HelloWorldFunction
```

Line 29 to Line 42 of app/template.yaml

The previous resource block will configure an automatic alarm if `HelloWorldFunction` (lines 11-14) reports more than five errors (lines 4-7) over 60 seconds (line 8). Note how the alarm refers to another resource in the same template, in line 14. The exclamation mark (`!`) signals a function call in CloudFormation, and `Ref` is a function that turns a logical resource ID into its actual physical name. CloudFormation will replace `!Ref HelloWorldFunction` with the actual function name when it creates the resource.

To tell CodeDeploy to roll back the update in case of problems, you need to add a reference to this alarm in the `Alarms` list of the function `DeploymentPreference`. YAML uses dashes (`-`) to prefix list elements, so you need to add a dash before the reference. You can use `!Ref` again to convert a logical identifier into an actual resource name.

```
DeploymentPreference:
  Type: Linear10PercentEvery1Minute
  Alarms:
    - !Ref HelloWorldErrors
```

Line 18 to Line 21 of app/template.yaml

The changes to `template.yaml` have already been done for you. Simply press the **Run** button to execute the commands.

Environment Variables

Key:	Value:
AWS_ACCESS_KEY_ID	Not Specified...
AWS_SECRET_ACCE...	Not Specified...
BUCKET_NAME	Not Specified...
AWS_REGION	Not Specified...


```
{
  "body": "{\"message\": \"hello world\"}",
  "resource": "/{proxy+}",
  "path": "/path/to/resource",
  "httpMethod": "POST",
  "isBase64Encoded": false,
  "queryStringParameters": {
    "foo": "bar"
  },
  "pathParameters": {
    "proxy": "/path/to/resource"
  },
  "stageVariables": {
    "baz": "qux"
  },
  "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"
  }
}
```

With that change, the deployment configuration will automatically test for Lambda errors and prevent a bad release if early users start getting unexpected integration problems. The nice aspect of the current configuration is that it is completely generic, so you can apply it to all your Lambda functions. The downside is that it just checks for technical errors, not for business problems.

The `Alarms` property is a list, so you can set up more than one alarm to monitor your deployments. For example, you can add a custom metric to CloudWatch to record sales, then check for unexpected changes in sales patterns. CloudWatch alarms can perform some basic statistics on custom metrics to spot trend changes (see the Create a CloudWatch Alarm based on a Metric Math Expression page in the CloudWatch documentation for more information). You can, of course, execute more complex calculations using Lambda functions. For example, instead of sending individual order information to CloudWatch, a Lambda function could add up all sales in one-hour intervals and just submit the difference from the previous period as a CloudWatch metric.

CodeDeploy can also run custom code before or after the deployment, to set up or clean up after tests. Unsurprisingly, this kind of custom integration uses Lambda functions. Use the `Hooks` property to define a `PreTraffic` function which will be executed before a gradual deployment, and a `PostTraffic` function that will be executed after the deployment completes.

Here is an example of a full setup for a `DeploymentPreference`:

```
DeploymentPreference:
 Type: Canary10Percent10Minutes
 Alarms:
   - !Ref CheckForLambdaErrors
   - !Ref CheckForDropInSales
   - !Ref CheckForDropInConversion
```

```
Hooks:

  PreTraffic: !Ref ClearStatisticsLambda
  PostTraffic: !Ref NotifyAdminsLambda
```

This example is illustrative (the code is not in the source code package for the course), but shows how you could set up the alarms and related Lambda functions to manipulate custom data. This will be your homework.

That is from this chapter. Next, you have some interesting experiments to try in the next lesson.