Module 19
CloudWatch Metrics and Alarms

Overview

So far we have used CloudWatch to log text information from a cloud service such as a Lambda function. In other words, we used CloudWatch like "a log file in the cloud". CloudWatch has many more capabilities that are always used in professional cloud applications. In this module we will look at two of these: **Metrics** and **Alarms**.

Almost all cloud software services have dashboards displayed on large TV screens. You can see them when you walk in many companies. These dashboards show metrics critical to the health of some service. For example, if your service allows searching for some data, one important metric you might want to calculate and capture is the average time it takes to process a query. Maybe you are OK with 250 milliseconds average query response time but not 10 seconds – 10 seconds might indicate some problem. Or let's assume your service averages 10,000 calls per hour. This metric is also worth capturing because if the service dashboard shows only 50 calls per hours, then there might be some underlying issue happening - maybe thousands of calls are failing to authenticate indicating a problem in the authentication component. So dashboards are the equivalents of a control room of a chemical plant (Figure 1): We want to see what is happening to learn about problems before they become severe enough to shut down the plant.

Figure 1: A service dashboard is the equivalent of a chemical plant control and monitoring room.

Metrics

Metrics are variables that you want to measure for your applications and resources. For example, here are some metrics:

- Error count: how many times my Lambda function failed?
- Duration: what is the average duration of execution? This is informative because long unexpected execution durations might indicate a problem in our code or in some other dependency our code is using.
- Invocations: how many times was the function called? This is very informative as well as it indicates if customers are using our product or not. Because time is usually reported with the measures, we can also get an idea of when are users using our application: more in the morning, spread out throughout the day, or late in the evening?

The above 3 metric examples are generic and AWS provides them to you out of the box (plus a few other ones as well). To see examples of metrics:

- Go to the Lambda service in the AWS console.
- Make sure **Dashboard** is selected.
- You should see metrics about functions (Figure 2).

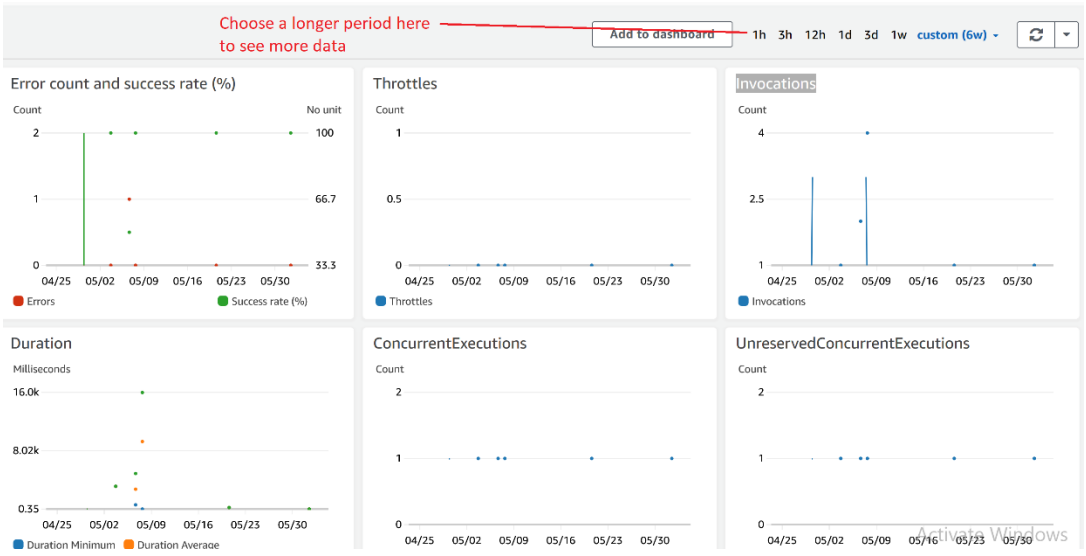


Figure 2: Example built-in metrics.

The above (like **Invocations** and **Error count and success rate**) are built-in metrics. That is, metrics that AWS is capturing for you out of the box. In most cases, however, you also need to collect metrics that are specific to your application. For example, you might be interested in collecting the number of successful login attempts and unsuccessful login attempts. Or maybe some other condition that might signal stress that needs to be looked at immediately. CloudWatch allows you to create custom metrics. Let's go through an example of how we can send custom metrics to CloudWatch:

1. Login to the AWS Console and go to the Lambda service.
2. Create a new function.
3. Choose the Author from scratch option.
4. For function name, use: **metric-data-function**
5. For Runtime, choose the latest version of Python (Python 3.8)
6. For execution role, choose the **Use an existing role** option, then select **LabRole**.
7. Click the **Create function** button.
8. In the code editor of the function, enter the code below:

```
import json
import random
import boto3
import time
from datetime import datetime

def get_metric_data(glucode_level):
    # Give your metric a name: let's call it BloodGlucose

    metric_data = [
        {
            'MetricName' : 'BloodGlucose',
            'Dimensions' : [
                {
                    'Name' : 'GLUCOSE_TEST',
                    'Value' : 'Blood Glucose Level'
                },
            ],
            'Timestamp' : datetime.now(),
            'Unit' : 'Count',
            'Value' : glucode_level
        }
    ]

    return metric_data
```

```
def lambda_handler(event, context):
    cloudwatch = boto3.client("cloudwatch")

    num_blood_tests = 5

    for _ in range(num_blood_tests):
        # Get a random glucose level value between 50 and 500
        glucose_level = random.randint(50, 500)

        # Create the metric data
        metric_data = get_metric_data(glucose_level)

        # Publish the metric data to cloudwatch
        response = cloudwatch.put_metric_data(Namespace='MyMedicalApp/Diabetes', MetricData=metric_data)

        print("response:", response)
        time.sleep(0.1)

    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Finished {} blood tests'.format(num_blood_tests))
    }
```

9. Click the **Deploy** button to save your changes.

10. In the **Test** button dropdown, click the **Configure test events**.

11. For Event name, choose: Test

12. Leave everything as is and click the Save button.

13. While the **Test** menu is selected, click the **Test** button (fix code errors, if any). If there are no errors and things work OK, you should see the response as shown below:

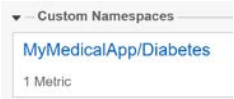
```
Test Event Name
Test

Response
{
  "statusCode": 200,
  "body": "\"Finished 5 blood tests\""
}
```

14. Click the **Test** button 5 or 6 more times.

    If you look at the code, the statement `cloudwatch.put_metric_data(…)` is sending our custom metric data to CloudWatch.

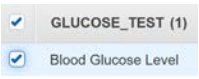
15. Go to the CloudWatch service page.

16. In the navigation bar on the left, click the **Metrics** -> **All metrics** link. You should see the following:

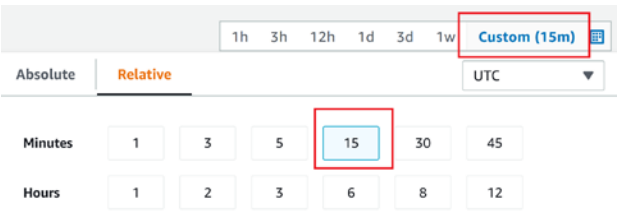


17. Click the above link, then check the CheckBox next to BloodGlucose metric.

18. Click the Blood Glucose Level

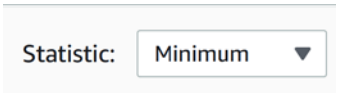


19. Above the graph area, click on Custom and choose 15 minutes.



20. You should see a blue point. Hover over it to read some data.

21. Change the value of the Statistics dropdown to something else (try Minimum, Maximum, etc).



22. You can go back to the Lambda function click the Test more and observe if something changes.

What we have done is added a custom metric that has some meaning to our application (blood glucose level). And this metric looks similar to the built-in AWS metrics like Invocations, Duration, and Errors count. Not only similar in the UI and how it is graphed, but you can use similar CloudWatch commands to query it (the same commands one uses to query the built-in metrics). We won't look at these commands, but you can search them for yourself in the Amazon CloudWatch User Guide.

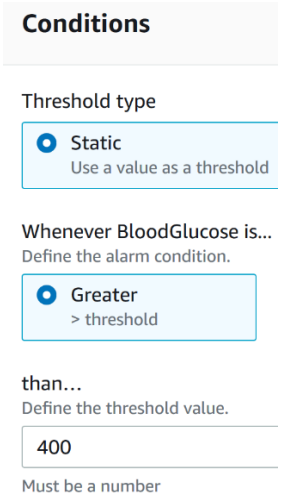
What can we do with this new glucose level metric? The simplest case is to login to the console and look at the graph. Although people do that, it is not the most interesting case because it does not allow us to immediately react to a dangerous situation that needs immediate attention from the software engineers developing and maintaining our application. A more interesting use case is to tie this metric to an **alarm**.

Alarms

"CloudWatch alarms send notifications or automatically change the resources you are monitoring based on rules that you define. For example, you can monitor the CPU usage and disk reads and writes of your Amazon EC2 instances. Then, use this data to determine whether you should launch additional instances to handle increased load. You can also use this data to stop under-used instances to save money. In addition to monitoring the built-in metrics that come with AWS, you can monitor your own custom metrics. With CloudWatch, you gain system-wide visibility into resource utilization, application performance, and operational health." (From the CloudWatch CLI manual).

As described above we can monitor our custom metrics (blood glucose level in our example) and have alarms notify us when some condition we define take place. Let's do that:

1. Go to the CloudWatch service.
2. From the navigation bar on the left, click the **Alarms** -> **All alarms** link.
3. Click the **Create alarm** button.
4. Click **Select metric**.
5. Choose **MyMedicalApp/Diabetes** then **GLUCOSE_TEST**.
6. Check the checkbox to the left of **Blood Glucose Level**, then click the **Select metric** button.
7. Leave the **Metric** section fields as-is.
8. In the **Conditions** section, use these settings:



9. Click the **Next** button.

10. On the **Notification** screen:

    a. Leave the **In alarm** option selected.
    b. Choose the **Create new topic** option.
    c. Give it a name: GlucoseLevelTooHighTopic
    d. Enter your email address.
    e. Click the **Create topic** button.

    NOTE: you can do other things like send a mobile text message to the nurse observing the patient, alert the doctor on call, enter a record in the patients' database, etc. (whichever way you want to react to this alarm).

    Click the **Next** button.

11. In the **Name** use: BloodGlucoseAtCriticalLevel
12. In the Description field, enter a description of your choice

13. Click the **Next** button.

14. Click the **Create alarm** button at the bottom of the Preview screen. You should see the alarm you just created in the list:

Check your email now. You should have received an email from "AWS Notifications". Open it and confirm the subscription by clicking the **Confirm subscription** link in the email message.

Wait until the **Actions** property is not in the **Pending** state (use the Refresh button to refresh the page)

Now let's try to trigger the alarm:

15. Go to the **Lambda** service and open the **metric-data-function** Lambda.
16. Let's force the blood glucose to be above 400. Make the following changes to the code:

    Change

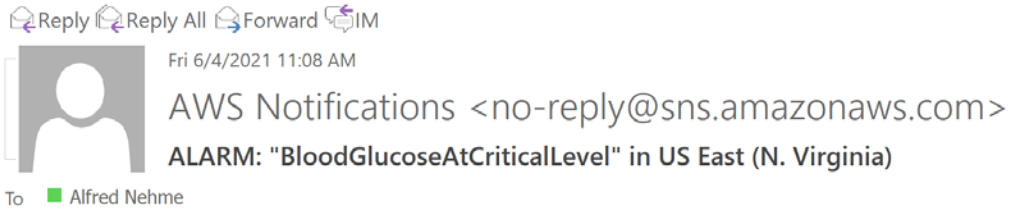
    ```
    glucose_level = random.randint(50, 500)
    ```

    to

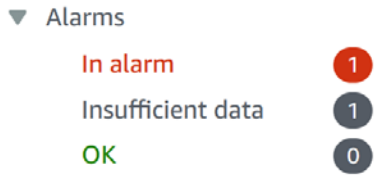
    ```
    glucose_level = random.randint(395, 500)
    ```

    By making the random number start at 395, and generating 5 of them, we will sure get lucky and have at least one above 400.

17. Save the changes by clicking the **Deploy** button.
18. Click the **Test** button to test the code.
19. Wait for about a minute or two. You should receive an alarm email from AWS.

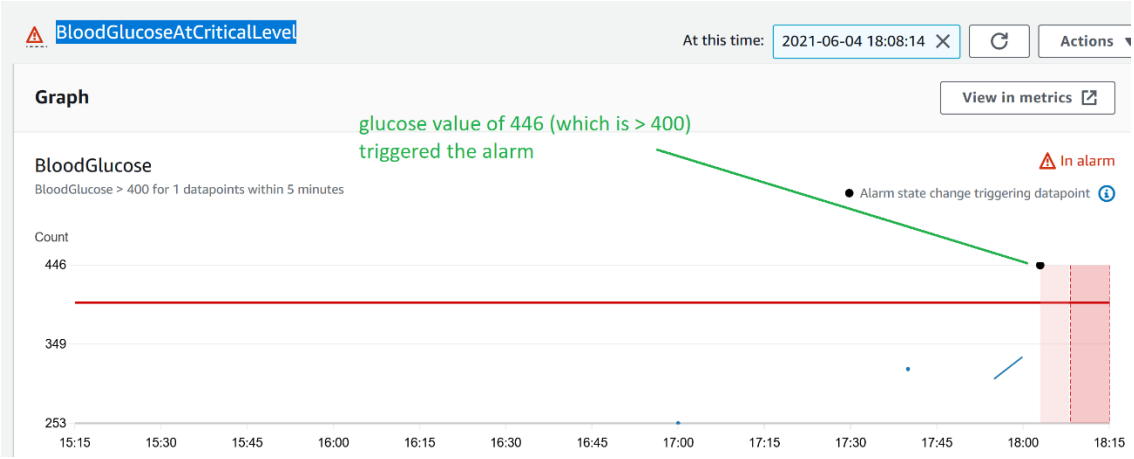


20. Go to the CloudWatch service, you should see that you have an alarm.



NOTE: It takes a while to see it and get the email. Wait for 5 minutes…

Click **Alarms**, then click the **BloodGlucoseAtCriticalLevel** alarm. In my case, for example, a value of 446 (greater than the 400 value we defined as the threshold) triggered the alarm:



Summary

We looked at 2 new functionalities of CloudWatch: **metrics** and **alarms**, and how they can be used together for monitoring cloud applications operational health. You can do other notifications (in addition to emails). We chose email here because it is the simplest.

In your cloud applications, you should emit metrics and alarm on conditions that are critical to your service.

At Amazon, metrics under certain conditions raise alarms that automatically log a ticket in Amazon's bug tracking system, and depending on the severity, automatically send a notification to an app (that engineers have on their cell phone) of the on-call engineer that happens to be on-duty for that time. The on-call engineer is immediately and automatically notified about such situations.

You can use metrics and alarms to add similar health-monitoring intelligence to your custom applications.

What to Submit

Nothing to submit for this module