Monitoring Exercises

CloudWatch

CloudWatch Logs

You will upload a Lambda in the `monitoring_lambda.zip` file which will help us generate logs that we can query. Once uploaded and ran, it will generate 100 logs. You can upload it on the AWS CLI with the following command:

```
aws lambda create-function --function-name [function-name] --zip-file
fileb://monitoring_lambda.zip --handler lambda_function.lambda_handler --runtime
python3.8 --role [role-arn] --profile [profile-name] --region eu-west-1 --
timeout 60
```

Amend the following in the command:

- `[function-name]` - Give the lambda a unique name that you will be able to identify soon after.
- `[role-arn]` - Your instructor will supply the role arn to pass.
- `[profile-name]` - Your AWS CLI profile name.

Now take the following steps:

1. Once uploaded, head to the AWS Lambda console. Hit the `Test` button, configure a test and run it (any test input will work).
2. Wait for the execution to finish. It may take up to a few minutes.
3. On the `Monitor` tab, select `View Logs in CloudWatch`.
4. Select the log stream, it should be called something like this, as an example: `2021/03/15/[$LATEST]374caa61c151478d8315ae5e9ae5a769`.
5. You should see 100 logs once the function has terminated successfully.

For context, this Lambda establishes a new logger and loops 100 times. Within this loop, it calculates two properties that we can see in the logs. `statusCode` is randomly chosen each time as success (`200`) or failure (`500`). Elapsed time is calculated by randomly selecting a number between 0 and 1 and then sleeps for that duration in seconds. The logged time is the total time the lambda ran. Both properties are logged to CloudWatch using a logger and logged in the form of JSON. You can see the two properties bundled up in the `msg` property.

Your job now is to use the filter input to try and filter the logs in a handful of ways.

1. You can use the terms `ERROR` or `INFO` to see only the error logs, or the info logs.
2. You can filter the logs by time right down to the second by their timestamp. For instance, if you only want to see logs that happened at the time of `2021-03-14T14:25` then you can input that as a search, wrapped in quotation marks like so: `"2021-03-14T14:25"`.
3. You can filter the logs with the two properties discussed above. Try these out and see what they do:

```
{ $.msg.statusCode = 200 }
{ $.msg.statusCode = 500 }
```

```
{ $.msg.executionTime > 0.9 }
{ $.msg.executionTime < 0.1 }
```

The `$` sign refers to a property selector, which means we are specifying what JSON object(s) we want to check. You can find out more about filtering and syntax patterns here.

CloudWatch Logs Insights

CloudWatch Logs is a useful service for quickly looking at logs, and perhaps making a quick filter on them. This is useful for debugging, but the capabilities are very limited.

CloudWatch Logs Insights enables you to interactively search and analyze your log data in Amazon CloudWatch Logs. You can perform queries to help you more efficiently and effectively respond to operational issues. If an issue occurs, you can use CloudWatch Logs Insights to identify potential causes and validate deployed fixes.

CloudWatch Logs Insights includes a purpose-built query language with a few simple but powerful commands. CloudWatch Logs Insights provides sample queries, command descriptions, query autocompletion, and log field discovery to help you get started. Sample queries are included for several types of AWS service logs.

It also automatically discovers fields in logs from AWS services such as Amazon Route 53, AWS Lambda, AWS CloudTrail, and Amazon VPC, and any application or custom log that emits log events as JSON.

1. On the CloudWatch console, navigate to `Insights` on the left.
2. Select the log groups dropdown and pick your Lambda.
3. Select `Run query`, you should see the same logs outputted. Expand one of the results and notice the slight changes from before in terms of how the log is structured.
4. Select `Fields` on the right. It will show you a list of fields that you can query on for the logs automatically. Notice that `msg.executionTime` and `msg.StatusCode` appear.

Insights provide you with a much more in-depth ability to query your logs. You can use the `Help` icon on the right to provide you with details on how to construct queries. Here are a few things you can do:

```
# show only statusCode
fields msg.statusCode

# show only statusCode with value of 200
fields msg.statusCode
| filter msg.statusCode = 200
```

Use this information to figure out how to find out the following:

1. The sum of how many `200` status codes there are, as well as `500`.
2. The average execution time.
3. How many executions had level name `ERROR` and how many `INFO`.
4. How many executions were above 0.5s and how many were below.
5. Sort the logs by execution time.

6. What others can you come up with?

You can find more information about the syntax here.

Grafana

This exercise will get you to start up an instance of Grafana. You will set up two data sources and display them in a dashboard on different panels.

Setting up Grafana

1. Ensure Docker is running on your machine.
2. To both pull down the Grafana image and start the container, run the following command:

```
docker run -d -p 3000:3000 --name=grafana -e
"GF_INSTALL_PLUGINS=grafana-simple-json-datasource" grafana/grafana
```

3. Confirm that you can see the dashboard by opening `localhost:3000` on your browser.
4. You can login with the default credentials. Both username and password are `admin`. It will ask you to choose a new password, input whatever you want here.

Data source setup

We will need some way of generating our data visualisations, so run the `grafana_data.py` file that was supplied to you as part of this exercise.

This will run a service on port `5000` and will generate some random data for us to display in Grafana. You may need to install flask to set this up correctly. Run `pip install flask` if so.

Once you have logged into the Grafana dashboard, you can tell Grafana where to look for data. We will be using a combination of the random data generated from `grafana_data.py`, as well as a test database that comes with Grafana.

1. On the left hand side, hover your mouse over the configuration icon and select `Data Sources`.
2. Select `Add data source`.
3. Select "TestData DB" from the list. Enter a name and then select `Save & Test`.
4. Check the Green OK message is created and then press the `Back` button.
5. Again, select `Add data source`.
6. Select `SimpleJson` from the list.
7. In the `URl` textbox, set it to http://host.docker.internal:5000. Enter a name and then select `Save & Test`.
8. Check the Green OK message is created and then press the `Back` button.

Setting up a dashboard

Now that we have the relevant data sources, we can hook them up to some visualisations for a dashboard we will put together.

1. Hover over the `+` icon on the left and select `Dashboard`.
2. Press the `Add New Panel` button. You will see a random looking graph and a bunch of other configuration.
3. Under the `Query` tab under the graph, select `TestData DB` from the dropdown.
4. Select `Random Walk` from the `Scenario` dropdown.
5. Select the `Apply` button in the top right corner. This will take us to our new dashboard with a generated graph.
6. Select the `Add panel` button in the top right and then the `Add new panel` button.
7. Under the `Query` tab under the graph, select `SimpleJson` from the dropdown.
8. Select `my_series` from the `select metric` dropdown.
9. Select the `Apply` button in the top right corner. This will take us back to our dashboard with another generated graph.
10. The dashboard should now have two panels.

Changing our data

1. Select the dropdown next to the refresh button in the top right corner. Set the option to "5s" or "10s" to set the auto refresh time.
2. In `grafana_data.py`, comment the current data points and un-comment the new data points. Watch what happens to the displayed data. It will have tilted the line.
3. Stop the python app and see what happens to the panel. It will automagically tell us that there is no data to display.

Changing our panels

There are plenty of ways of displaying data in Grafana. How do we change how our data is being displayed?

1. Press the arrow by the `Panel Title` to bring up the menu and select `Edit`
2. In the `Scenario` dropdown, select a few different scenarios and see the change in the panel data. Lots of cool looking visuals going on!

Final Project

For your final project you will need to setup an EC2 instance which will run Grafana on Docker (one instance per team).

Setting up EC2 and Grafana

1. Nominate one person to setup the EC2 instance. You can take the same steps you took for the AWS exercise to achieve this. There will be some configuration differences:
    - The instance should have the `grafana-role` IAM role attached
    - The instance should be provisioned in the `Redshift` VPC in the Public Subnet
2. SSH into the instance with the SSH key you downloaded. DO NOT LOSE THIS KEY!
3. Run the following commands to install and setup Docker:

```
sudo amazon-linux-extras install docker -y  # install
sudo service docker start                   # start
sudo usermod -a -G docker ec2-user          # create new user group
sudo chkconfig docker on                    # allow docker to run on
startup
```

4. Setup a Grafana container:

```
sudo docker run -d -p 80:3000 grafana/grafana
```

5. Open a new browser tab and paste your instance's public IPv4 address and hit enter. You will now see your new Grafana board that your whole team can access. Make sure your browser doesn't prepend `https` to the IP, otherwise it may fail.

Optional: Connect to Grafana with SSM

1. Navigate to your Grafana EC2 instance in the AWS Console
    1. Select the `Connect` button on the instance page in the AWS console.
    2. Select the `Session Manager` tab and select `Connect`. This will open a terminal in a new tab with you already logged in to the instance.

Setup Grafana users

To create a new user login for each team member, navigate to `Server Admin --> Users --> New user` and begin creating unique users.

Connecting Grafana to Cloudwatch

Just like in the earlier exercise, we need to connect a data source in order to generate some graphs and metrics.

1. In Grafana, navigate to `Configuration --> Data Sources`. Select `Add data source`, search for `Cloudwatch` and select.
2. Give it a name, or leave as default.
3. Leave other settings as default.
4. Set region to `eu-west-1`.
5. Select `Save & Test`. You should see a confirmation `Data source is working`.

Creating a Lambda metric

1. Create a new dashboard and add a new panel.
2. Select `Cloudwatch` as the query type, and `Cloudwatch Metrics` as the query mode.
3. Select `AWS Lambda` as the namespace, and `Invocations` as the metric name.
4. Add a new Dimension. Select `Function name` as the resource and select the dimension value as your teams ETL lambda.

5. Update the time query to be last 24 hours or 2/7 days if you need to go back that far to see data being graphed.

You should be able to now see how many times your lambda has been invoked over the time elapsed configured for the time period. You can choose different metric options to suit your needs. For example, you can select `Error` and `Duration` as the metric name, as well as different stats such as `Average`, `Sum`, `Min` and `Max`.

As team, think about what kind of monitoring metrics you can establish to display on your new dashboard.