CloudWatch Alert Monitoring Python Script

Introduction

CW Monitoring script is a Python script designed to perform the following functions:

- Generate reports for alarm validation in Amazon Web Services (AWS) resources like EC2, RDS, ALB, Lambda etc.
- Create missing AWS CloudWatch alarms for EC2, RDS, ALB and Lambda based on validation.
- If validation fails then it will create alarms based on provided alarm functionalities in "**input.yaml**" files.

Provide flexibility to include custom resources for alarm validation and creation.

Prerequisites:-

Before using the script, ensure you have the following in place:

1) Python3
2) Pip3
3) **Python Dependencies**:

   Also, remember to execute the **requirements.txt** file to install the necessary packages for the python script to run.

   Command to execute requirement.txt file:

   ```
   pip3 install -r requirement.txt
   ```

4) Attach the provided IAM policy to the instance where you intend to run the script. These permissions are the minimum requirements for running the script.

IAM-policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:DescribeInstances",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:ListTagsForResource"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "lambda:ListFunctions",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTags"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:PutMetricAlarm"
      ],
      "Resource": "*"
    }
  ]
}
```

Input File Structure :

1. Input File

The script uses a YAML configuration file (**input.yaml**) to specify the following information:

Services to monitor and their associated alarm configurations.

- **region_name**:

  Specify the region where you intend to execute the script and monitor CloudWatch alarms.

  ```
  region_name: <region>
  ```

- **Prefix**:

  Define the prefix to be used when creating alarm names. This prefix ensures that alarm names are easily identifiable.

  ```
  prefix: <prefix>
  ```

- **Services:**

  In the 'Services' section of the 'input.yaml' file, you can define the AWS services for which you want to use the script. E.g. EC2, RDS, LAMBDA, ALB.

  The script uses the tags specified in the **'Tag'** section of the 'input.yaml' file to search for AWS services resources. The tags act as filters to identify which resources to monitor.

Let's take an EC2 service to understand the section.

E.g

```
Services:
  EC2:
    enabled: true
    Tag:
      Project:
        - aws
      Environment:
        - Prod
        - Dev
```

Enabled: Set to true to include EC2 instances in the script's processing. Set to false to exclude EC2 instances.

If you set the value of enable **true** then it will check for that service. And if you set it as **false** then that service will be **excluded**.

Tag: This section for each service contains the tagging criteria that must be met for the service's resources to be included in the monitoring process. The script will search for resources that match these tags and create a list and that list of resources will be passed to the respective function to validate and create alarms.

❖ **Project**: The "Project" tag value must match the values present on your resources. **Project is the keys and below you can mention the value.** You can add multiple values for the key.

This overview outlines the services you can configure, their enabling/disabling options, and how the specified tags determine the resources to be monitored for each service.

- **sns_action** :

This line specifies the Amazon SNS (Simple Notification Service) topic(s) to which alarms will send notifications. The provided value is the ARN of the SNS topic where notifications will be published. Alarms configured in the script will send notifications to this topic when triggered.

E.g.

```
sns_action: ['arn1', ‘arn2’, ….]
```

- **service** :

  This section specifies the AWS services for which you are defining alarm configurations, for example, 'EC2', ‘RDS’, ‘LAMDA’, ‘ALB’.

  Let's take an example for EC2 to understand.

  Under the 'EC2' service section, you define alarm configurations for specific metrics related to EC2 instances. Like CPU, Memory, Disk, System etc.

  Here, 'CPU' is the name of the metric you are Monitoring an alarm for. You can create similar sections for other metrics like 'Memory' or any other custom metric.

```
service:
  EC2:
    CPU:
    - MetricName:
      DatapointsToAlarm:
      EvaluationPeriods:
      Period:
      Threshold:
      Statistic:
      ComparisonOperator:
      TreatMissingData:
      AlarmDescription:
```

- Under CPU section you need to provide **‘MetricName’, 'DatapointsToAlarm', 'EvaluationPeriods', 'Period', 'Threshold', 'Statistic', 'ComparisonOperator', 'TreatMissingData', 'AlarmDescription'.**

- Please ensure that the **'MetricName'** for each alarm configuration matches the AWS CloudWatch metric name exactly as specified in the AWS CloudWatch documentation. E.g 'CPUUtilization' 'mem_used_percent', 'disk_used_percent', 'StatusCheckFailed'.
- Please customize the values of 'DatapointsToAlarm,' 'EvaluationPeriods,' 'Period,' 'Threshold,' 'Statistic,' 'ComparisonOperator,' 'TreatMissingData,' and 'AlarmDescription' to match your specific requirements for validating and creating alarms.
- **The validation process involves comparing the input values for 'DatapointsToAlarm,' 'EvaluationPeriods,' 'Period,' 'Threshold,' 'Statistic,' 'ComparisonOperator,' and 'TreatMissingData' with the corresponding values configured for each alarm. If these values match, the validation is considered successful ('pass'). If there are differences then the validation result is marked as 'fail.**
- In the generated report i.e. Excel sheet, when a validation fails, the report will provide detailed reasons explaining why the validation failed. These reasons will help identify the specific aspects of the alarm configuration that require attention or adjustment.

Now Second Part of input file:

Resources:

- In this section, you can verify whether alarms have been configured for specific AWS resources.
- E.g if you have a Jenkins instance and you need to check that the alarm is configured or not and need to check validation, then you can use this section. It will also work as above if validation fails then it will create alarms.
- The structure will be like this

```
Resources:
  <services>:
    <service_ID>:
      <Metric>:
            - ResourceName:
              MetricName:
              DatapointsToAlarm:
              EvaluationPeriods:
              Period:
              Threshold:
              Statistic: "Average"
              ComparisonOperator:
              TreatMissingData:
              AlarmDescription:
```

- **Services**:

  'EC2', 'RDS', 'LAMDA', 'ALB' etc

- **service_ID** :

  i-00ce7534221229f12, Database1, app/test-alb/044201faecb18ff9 like this

- **Metric** :

  CPU, Memory, Disk, Error … etc

How To Run Script

Script has divided into classes for each services but we need to execute just **main.py**

To run the script, follow these steps:

You need to run the main.py file.

```
python3 main.py
```

Upon running the script, you will be prompted with a menu:

```
Select an option:
1. Generate report for alarm validation
2. Create missing alarms
3. Exit
```

Choose option 1:
To generate a report for alarm validation. This option will compare the configured alarms with the existing alarms for provided resources and display the results as well as create excel sheets.
The report will be saved in an Excel file named **CW-Monitoring_<current_date>.xlsx**.

Choose option 2: To create missing alarms based on the configuration file as well as validation Status. The script will create CloudWatch alarms for provided resources where alarm validation fails.
Choose option 3: to exit the script.

FYI
1) If you don't want to monitor/check any of the services then you can comment/remove that service from the input file.
2) It will also work for the 2nd part i.e. custom resources in the input file.

Here is an example structure of the **input.yaml** file:

```
region_name: ap-south-1
prefix: custom

Services:
  EC2:
    enabled: true
    Tag:
      Project:
        - aws
      Environment:
        - Prod
        - Dev

  RDS:
    enabled: true
    Tag:
      Project:
        - aws
      Environment:
        - Prod
        - Dev

  ALB:
    enabled: true
    Tag:
      Project:
        - aws
      Environment:
        - NProd
        - Dev

  LAMBDA:
    enabled: false


sns_action: ['arn:aws:sns:ap-south-1:342646517446:demo',arn:aws:sns:ap-south-1:342646517446:test]

service:
  EC2:
    CPU:
      - MetricName: "CPUUtilization"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 75.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average CPU utilization is too high."
    Memory:
      - MetricName: "mem_used_percent"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 75.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average memory utilization is too high"
    Disk:
      - MetricName: "disk_used_percent"
        DatapointsToAlarm: 1
        EvaluationPeriods: 1
        Period: 300
        Threshold: 75.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average Disk utilization is too high."
    System:
      - MetricName: "StatusCheckFailed"
        DatapointsToAlarm: 1
        EvaluationPeriods: 1
        Period: 300
        Threshold: 0.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "System Health check failed more than 2 times."

  RDS:
    CPU:
      - MetricName: "CPUUtilization"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 70.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average CPU utilization is too high."
    Memory:
      - MetricName: "FreeableMemory"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 500
        Statistic: "Average"
        ComparisonOperator: "LessThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average CPU utilization is too high."
    Connections:
      - MetricName: "DatabaseConnections"
        DatapointsToAlarm: 1
        EvaluationPeriods: 1
        Period: 300
        Threshold: 5.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average CPU utilization is too high."
   Lambda:
    Errors:
      - MetricName: "Errors"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 4
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average Errors is too high."
   ALB:
    RequestCount:
      - MetricName: "RequestCount"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 60
        Threshold: 15
        Statistic: "Sum"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average CPU utilization is too high"

  Resources:
   EC2:
    i-00ce7534221229f12:
      CPU:
      - ResourceName: test
        MetricName: "CPUUtilization"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 75.0
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average CPU utilization is too high"
  Lambda:
    demo:
      Errors:
      - ResourceName: demo
        MetricName: "Errors"
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 300
        Threshold: 4
        Statistic: "Average"
        ComparisonOperator: "GreaterThanThreshold"
        TreatMissingData: "missing"
        AlarmDescription: "Average Errors is too high."
  ALB:
    app/test-alb/a3ca4e84aef6f020:
      RequestCount:
      - ResourceName: test-alb
        MetricName: RequestCount
        DatapointsToAlarm: 2
        EvaluationPeriods: 2
        Period: 60
        Threshold: 15
        Statistic: Sum
        ComparisonOperator: GreaterThanThreshold
        TreatMissingData: missing
        AlarmDescription: Average CPU utilization is too high
```