Monitoring Denodo with Amazon CloudWatch

Revision 20200803

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

1 INTRODUCTION

Amazon CloudWatch is a monitoring and management service that collects monitoring and operational data in the form of logs, metrics, and events, providing you with a unified view of AWS resources, applications and services that run on AWS, and on-premises servers. Therefore, you can use Amazon CloudWatch Logs to monitor and troubleshoot your applications using your custom log files like Denodo Log files.

In this document, you will learn how to configure Cloudwatch to consume Denodo Server and Denodo Monitor Logs. You will also learn how to use Cloudwatch to monitor Denodo metrics and to generate alerts from them.

More information about Denodo Server and Denodo Monitor Logs can be found in the Denodo Knowledge Base.

2 INGESTING DENODO LOGS IN CLOUDWATCH

Amazon offers a unified CloudWatch agent that has the ability to collect and monitor individual log files from Amazon EC2 instances and on-premises servers, running either Linux or Windows Server. Once the CloudWatch agent has moved the files into the Amazon CloudWatch Logs, you can access the raw log event data.

In order to ingest the log files created by Denodo you must take into account that Denodo generates Denodo Server Logs that are stored at `<DENODO_HOME>/logs/vdp` and, in addition, the Denodo Monitor tool logs information under `<DENODO_HOME>/tools/monitor/denodo-monitor/logs` or under `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work`, in case you have launched the Denodo Monitor tool using the Solution Manager. In this second scenario, Denodo Monitor can collect the execution logs from a single Virtual DataPort server, or from all the servers of a cluster or environment and each log file name starts with the server name to which the data it contains belongs. Keep in mind that when the Denodo Monitor is launched by the Solution Manager, log files are created in different paths depending on the level at which you started the monitoring process:

- Log files of environments: `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work/<EnvironmentName>/logs`
- Log files of clusters: `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work/<EnvironmentName>/<ClusterName>/logs`
- Log files of servers: `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work/<EnvironmentName>/<ClusterName>/<ServerName>/logs`

Note that these paths are different in Denodo 7.0. In this version, a `<timestamp>` folder containing the `<logs>` folder, among other data, is created every time the Solution Manager starts the Denodo Monitor:

- Log files of environments: `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work/<EnvironmentName>/<timestamp>/logs`
- Log files of clusters: `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work/<EnvironmentName>/<ClusterName>/<timestamp>/logs`
- Log files of servers: `<SOLUTION_MANAGER_HOME>/resources/solution-manager-monitor/work/<EnvironmentName>/<ClusterName>/<ServerName>/<timestamp>/logs`

See the Monitoring section of the Solution Manager Administration Guide for more information.

2.1 CREATING THE CLOUDWATCH AGENT CONFIGURATION FILE

After installing the unified CloudWatch agent and before running it on any server, you have to create a CloudWatch configuration file (JSON format) in order to specify the logs that the agent is to collect.

There is a schema definition that you can use to create your own configuration file manually. It is located at `installation-directory/doc/amazon-cloudwatch-agent-schema.json` on Linux servers, and at `installation-directory/amazon-cloudwatch-agent-schema.json` on servers running Windows Server.

In order to collect logs from your Denodo Server or Denodo Monitor with the CloudWatch agent, only the logs section is required. In this section, you have to include the `files` field with the `collect_list` field that contains an array of entries, each of which specifies one log file to collect.

An entry must include the path of the log file to upload to CloudWatch Logs (`file_path` field) and it is the only mandatory field. Standard Unix glob matching rules are accepted, with the addition of `**` as a super asterisk. The standard asterisk can also be used as a wildcard but be aware that only the latest file is pushed to CloudWatch Logs based on file modification time. Therefore, you should add an entry per log file to the configuration file if you want to ingest several files that were already created before starting the ingestion process. However, note that the Denodo Server and Denodo Monitor logs have a file rotation mechanism that is supported by the CloudWatch agent, so once you have added a log file in the configuration, when a backup file is generated, the agent will upload the new file transparently.

You should establish a `log_group_name` (e.g. `denodo-monitor-vdp-queries_{instance_id}`) and a `log_stream_name` (e.g. `vdp-queries_{instance_id}`) that identifies the origin of the events to prevent confusion. If the specified log group or log stream does not already exist, it is created automatically.

The `timestamp_format` field specifies the timestamp format. It is optional but if you omit this field, the current time is used to associate with an event so you should add the `timestamp_format` field to keep the time-related information for the Denodo logs events.

You can see a configuration file example bellow:

```
{
 "logs": {
  "logs_collected": {
   "files": {
    "collect_list": [
     {
      "file_path": "C:\\Denodo\\tools\\monitor\\denodo-monitor\\logs\\vdp-queries.log",
      "log_group_name": "denodo-monitor-vdp-queries_{instance_id}",
      "log_stream_name": "vdp-queries_{instance_id}",
      "timestamp_format": "%Y-%m-%dT%H:%M:%S"
     },
     {
                    "file_path":  "C:\\Denodo\\tools\\monitor\\denodo-monitor\\logs\\vdp-
resources.log",
      "log_group_name": "denodo-monitor-vdp-resources_{instance_id}",
      "log_stream_name": "vdp-resources_{instance_id}",
      "timestamp_format": "%Y-%m-%dT%H:%M:%S"
     }
    ]
   }
  }
```

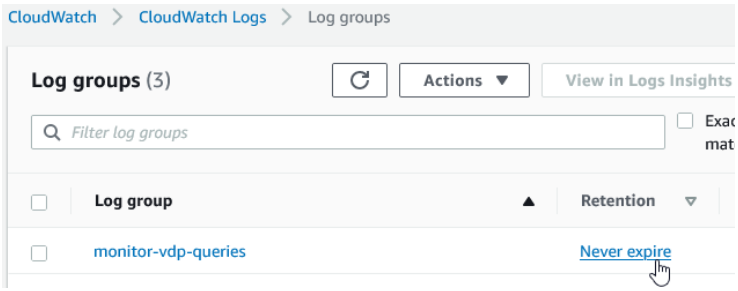
```
  }
}
```

Take into account, that any time you change the agent configuration file, you must then restart the agent to have the changes take effect.

You can find further information on creating or editing the CloudWatch configuration file manually in the Amazon CloudWatch documentation.

2.2 LOG RETENTION

By default, logs are kept indefinitely and never expire. You can adjust the retention policy for each log group, keeping the indefinite retention, or choosing a retention period between 10 years and one day.



2.3 CLOUDWATCH LOGS LIMITS

CloudWatch Logs has some limits that must be taken into account since they can cause unexpected behaviour in the process of loading log files:

- The maximum size of a log event is 256 KB and it cannot be changed. If you have log events that exceed this size, for example a log entry that includes a `CREATE JAR` statement, it will be truncated and loaded as several entries.
- If the timestamp of a log event is more than 14 days in the past or more than 2 hours in the future, the log entry is discarded.

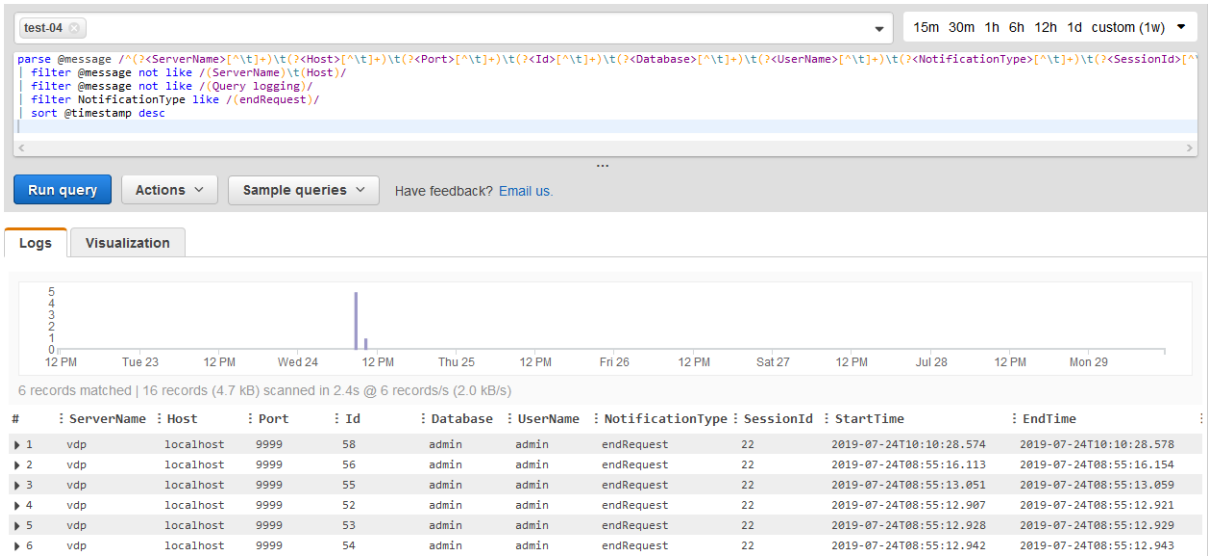
You can see more detailed information about the limits in the CloudWatch documentation.

Note that the agent generates a log file (`amazon-cloudwatch-agent.log`) while it runs. You can check this log file to find information about the loading process and see if any of these limitations have affected it.

3 INSPECTING AND SEARCHING LOG FILES

After the CloudWatch agent begins publishing log data to Amazon CloudWatch, you can begin searching and filtering the log data by creating one or more metric filters. Nevertheless, Amazon CloudWatch, at the time of this writing, does not allow you to set a delimiter value or use regular expressions in the definition of a metric filter. This limitation makes it impossible to set filters in Denodo Logs because Denodo Monitor Logs are tab delimited and Denodo Server Logs are space delimited. In both cases, a space does not serve as a criterion to delimit a field.

Fortunately, Amazon CloudWatch also offers the possibility to use the CloudWatch Logs Insights service that enables you to interactively search and analyze your log data in CloudWatch Logs. You should open the CloudWatch console and choose **`Insights`**. Then you have to select one or more log groups where the service will execute the search. It includes a purpose-built query language with a few commands that you can use in the query editor. The example below shows how to extract the fields of the vdp-queries.log from the Denodo Monitor logs. The filter command allows you to remove headers and use the ephemeral fields from the parse command to get only the entries of `endRequest` type.



The query used in the example above is:

```
parse @message /^(?<ServerName>[^\t]+)\t(?<Host>[^\t]+)\t(?<Port>[^\t]+)\t(?
<Id>[^\t]+)\t(?<Database>[^\t]+)\t(?<UserName>[^\t]+)\t(?<NotificationType>[^\t]
+)\t(?<SessionId>[^\t]+)\t(?<StartTime>[^\t]+)\t(?<EndTime>[^\t]+)\t(?
<Duration>[^\t]+)\t(?<WaitingTime>[^\t]+)\t(?<NumRows>[^\t]+)\t(?<State>[^\t]
+)\t(?<Completed>[^\t]+)\t(?<Cache>[^\t]+)\t(?<Query>[^\t]+)\t(?
<RequestType>[^\t]+)\t(?<Elements>[^\t]+)\t(?<UserAgent>[^\t]+)\t(?
<AccessInterface>[^\t]+)\t(?<ClientIP>[^\t]+)\t(?<TransactionID>[^\t]+)\t(?
<WebServiceName>[^\t]+)/
| filter @message not like /(ServerName)\t(Host)/
| filter @message not like /(Query logging)/
| filter NotificationType like /(endRequest)/
| sort @timestamp desc
```

CloudWatch Insights can be useful to analyze Denodo Logs but you cannot create metrics using this service. How to do this is explained in the following section.

4 MONITORING DENODO METRICS

Because of the limitations explained in the previous section, to monitor Denodo with Cloudwatch we need to develop custom metrics. In this section we will show step by step how to develop a Cloudwatch custom metric for monitoring the number of requests initiated in Denodo every minute.

The AWS SDK for Java provides a Java API for Amazon Web Services, including CloudWatch. The single, downloadable package includes the AWS Java library, code samples, and documentation that you can use to create your custom metrics from Denodo Logs.

The idea is to create an application to process the Denodo Logs and to extract the necessary data to prepare each of the desired metrics. You can see an example in the code below where the log `vdp-queries.log` from Denodo Monitor is processed in order to create a metric with the number of queries initiated per minute. You can find more information about the `vdp-queries.log` file in the **`VDP Queries Monitor`** section of the Log column details document.

```
public class PutMetricDataQueriesInitiatedPerMinute {

  // Start of Denodo Monitor start row (vdp-queries.log)
  public static final String VDP_QUERIES_STARTING = "Logging started
at:";
  // Start of headers row (vdp-queries.log)
  public static final String VDP_QUERIES_HEADERS = "ServerName";

  public static void main(String[] args) {

    final String USAGE =
                  "To run this example, supply a file path, a namespace
for the metric data and a metric name:\n" +
                      "Ex: PutMetricDataQueriesInitiatedPerMinute
<file_path> <namespace> <metric_name>\n";

    if (args.length != 3) {
      System.out.println(USAGE);
      System.exit(1);
    }

    String filePath = args[0];
    String namespace = args[1];
    String metricName = args[2];

    List<MetricDatum> metricDataList = new ArrayList<>();

    try (BufferedReader buffferedReader = new BufferedReader(new
FileReader(filePath)) ) {

      String currentLine = null;
      Instant previousMinute = null;
      double queriesCounter = 0;
```

```
      while (true) {
        currentLine = buffferedReader.readLine();
        if (currentLine == null) {
          break;
        }

        // Discard lines without log information
        if( !currentLine.isEmpty() &&
            !currentLine.startsWith(VDP_QUERIES_STARTING) &&
            !currentLine.startsWith(VDP_QUERIES_HEADERS)) {

          // Get the fields (tab-delimited file)
          String[] fields = currentLine.split("\t");

          String notificationType = fields[6]; // NotificationType field

          if (!"-".equals(notificationType) &&
              "startRequest".equals(notificationType)) {

            String timestampString = fields[8]; // StartTime field

            DateTimeFormatter formatter =
                DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss.SSS")
                .withZone(ZoneId.systemDefault());
            Instant instant =
Instant.from(formatter.parse(timestampString));

            // Truncate the instant to minutes in order to get the minute
of the current query
            Instant currentMinute =
instant.truncatedTo(ChronoUnit.MINUTES);

            if (previousMinute == null) {
              previousMinute = currentMinute;
            }

            if (currentMinute.compareTo(previousMinute) == 0) {
              // The entry corresponds to a new initiated query in the
minute of time we are currently dealing with
              queriesCounter++;
            } else {
              // The entry corresponds to a new initiated query in a
different minute than the one already processed.

              // We have to add the processed data in the previously
handled minute
              addMetricDataToCollection(metricDataList, metricName,
previousMinute, Double.valueOf(queriesCounter), namespace);

              // Get the new minute and restart the counter
              previousMinute = currentMinute;
              queriesCounter = 1;
            }
          }
        }
      }

      if (queriesCounter > 0 && previousMinute != null) {
```

```
        // Add processed data
        addMetricDataToCollection(metricDataList, metricName,
previousMinute, Double.valueOf(queriesCounter), namespace);
      }

    } catch (Exception e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }

    putMetricData(metricDataList, namespace);

  }

  private static void addMetricDataToCollection(final List<MetricDatum>
metricDataList, final String metricName,
            final Instant instant, final Double queriesCounter, final
String namespace) {
    // As specified in the API, the collection MetricData must not have a
size greater than 20
    if (metricDataList.size() == 20) {
      putMetricData(metricDataList, namespace);

      metricDataList.clear();
    }

    MetricDatum datum = MetricDatum.builder()
        .metricName(metricName)
        .timestamp(instant)
        .unit(StandardUnit.COUNT)
        .storageResolution(Integer.valueOf(60))// We send information
                                               // per minute.
                                               // A high-resolution
                                               // metric is not necessary
        .value(Double.valueOf(queriesCounter.doubleValue())).build();

    metricDataList.add(datum);
  }

  private static void putMetricData (final List<MetricDatum> metricData,
final String namespace) {
    try (CloudWatchClient cw = CloudWatchClient.builder().build()) {

      if (!metricData.isEmpty()) {
        PutMetricDataRequest request = PutMetricDataRequest.builder()
                        .namespace(namespace)
                        .metricData(metricData).build();

        cw.putMetricData(request);
      }
    } catch (Exception e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
  }
}
```

It is important to bear in mind that data points with timestamps from 24 hours ago or longer can take hours to become available. Besides, metrics cannot be deleted, but they automatically expire after 15 months if no new data is published to them. Data points older than 15 months expire on a rolling basis; as new data points come in, data older than 15 months is dropped.

4.1 RETENTION PERIOD OF METRICS

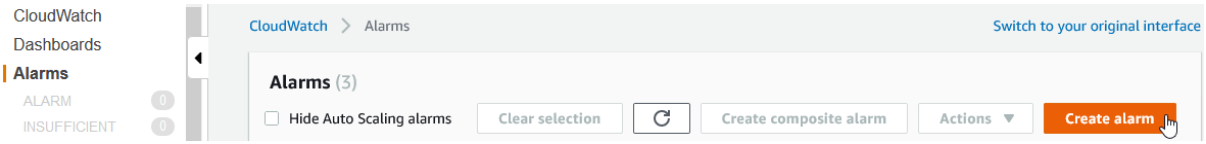
Amazon CloudWatch has a retention period policy for metrics that must be taken into account. You can look it up in the `AWS resource and custom metrics monitoring` section of the Amazon CloudWatch FAQs, under the question `What is the retention period of all metrics?`.

5 ADDING AN ALARM

Once you have your custom metric created from the data extracted from Denodo Logs you can create an alarm based on it in order to perform one or more actions based on the value of the metric relative to a threshold over a number of time periods.

You can open the alarm user interface using one of these options:

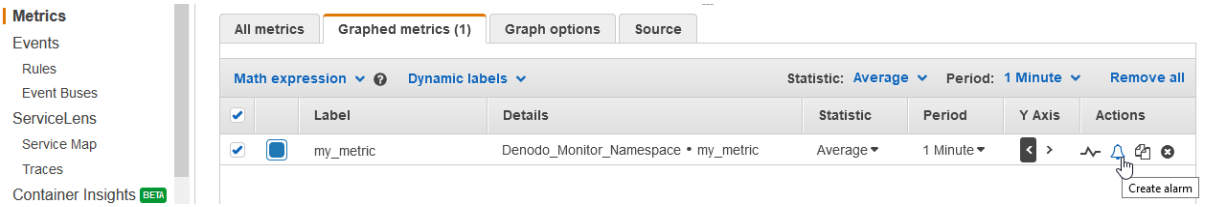
1. Clicking on **`Alarms > Create Alarm`**, in the navigation panel.



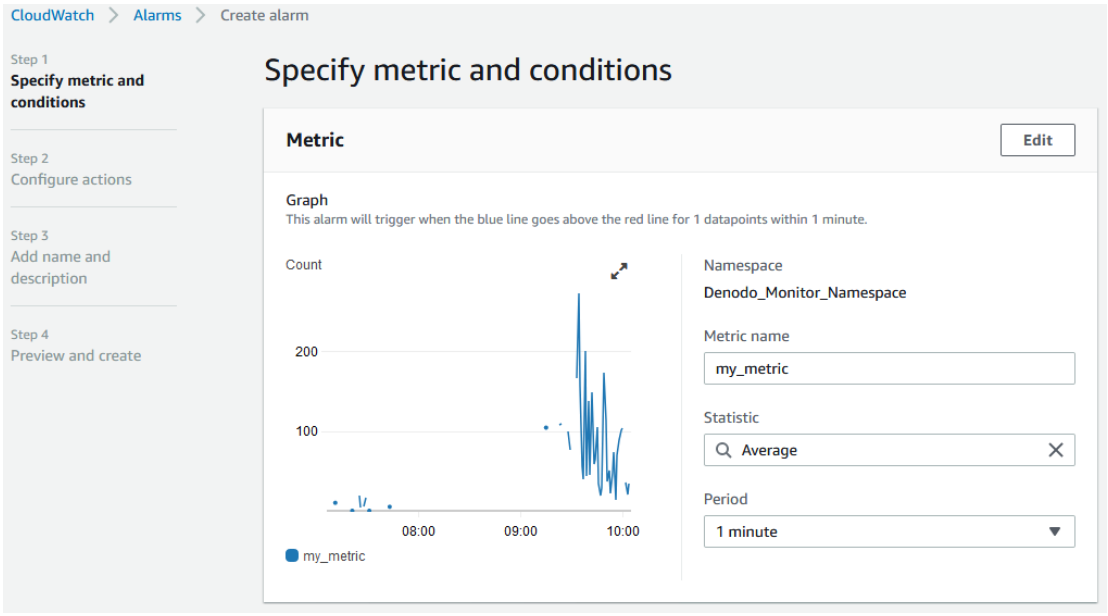
   After selecting a metric, the alarm user interface will be shown.

2. After selecting the metric, choose the **`Graphed metrics`** tab if you want to change some values as the **`Period`** and then click on the **`Select metric`** button.

   You can also create an alarm from the **`Metrics`** panel. When you select your metric and choose the **`Graphed metrics`** tab, among the available **`Actions`** you can find **`Create alarm`**.

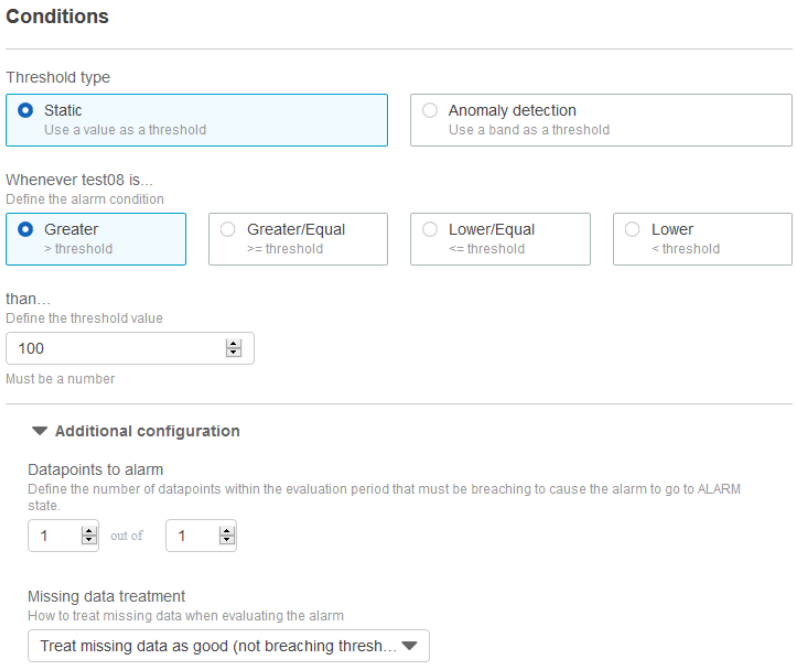


In the first step of the alarm user interface you can change de statistic and the period values:



Under **`Statistic`** you should choose one of the statistics or percentiles and under **`Period`** you have to choose the length of time to evaluate the metric or expression to create each individual data point for an alarm. When evaluating the alarm, each period is aggregated into one data point. The data of the example, with information about the number of queries initiated per minute, require a period of 1 minute. You can find more information regarding these concepts in the CloudWatch documentation.

Then you have to specify the conditions for the alarm:



The threshold type can be static when you want an alarm based on a value. The anomaly detection mines past metric data and creates a model of expected values and then you specify a number of standard derivations to determine the “normal” range of values.

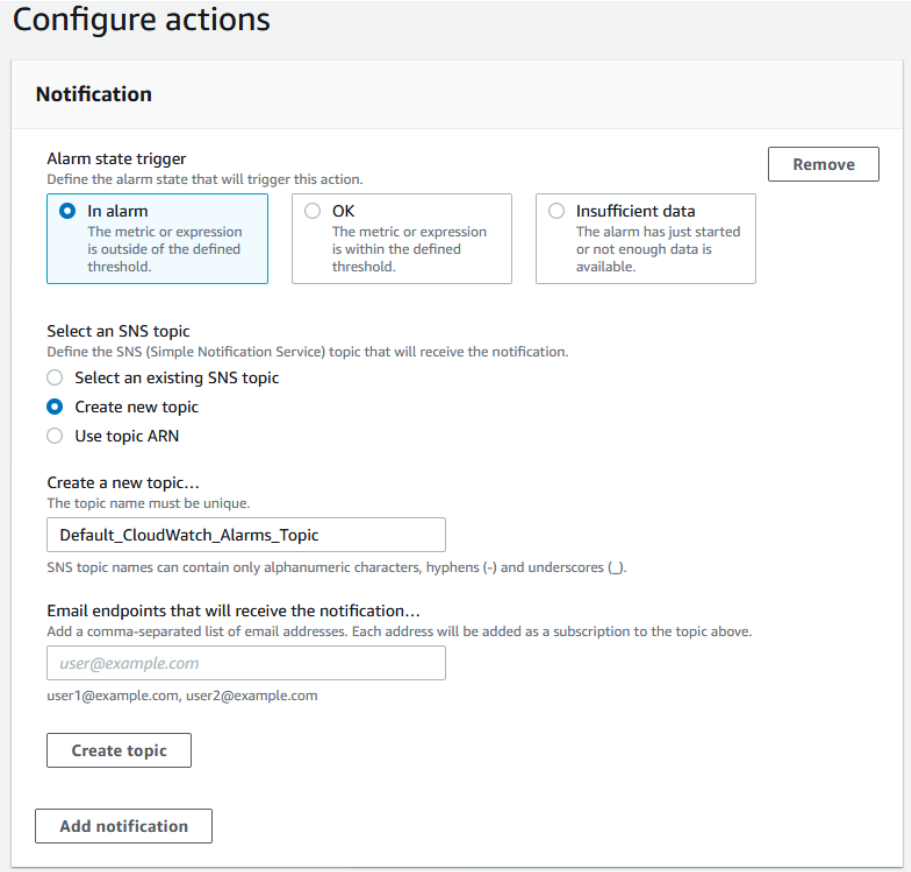
Continuing with our example, we are going to create an alarm based on a static threshold. In our scenario, we want to specify a condition to send a notification when metric values are greater than 100.

In the additional configuration we must establish how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. For this, we create an **`M out of N`** alarm where `N` (evaluation period) represents the number of the most recent periods, or data points, to evaluate when determining alarm state and `M` (data points to alarm) is the number of data points within the evaluation period that must be breaching to cause the alarm to go to the `ALARM` state. The breaching data points don't have to be consecutive. In our example, we will set `1 out of 1` to go to the ALARM state every time the threshold is exceeded. You can see further information in the Evaluating an Alarm section of the CloudWatch documentation.

The **`Missing data treatment`** chooses how to have the alarm behave when some data points are missing. With the metric created in the example, we can consider that missing data points are within the threshold. For more information, see Configuring How CloudWatch Alarms Treat Missing Data in CloudWatch documentation.

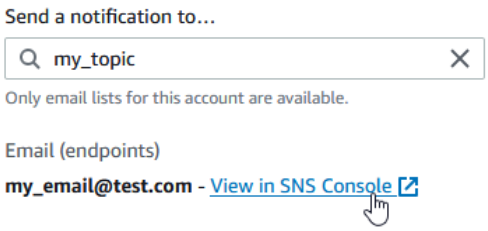
The next step allows you to configure the actions.

In the first section you have the notification settings:

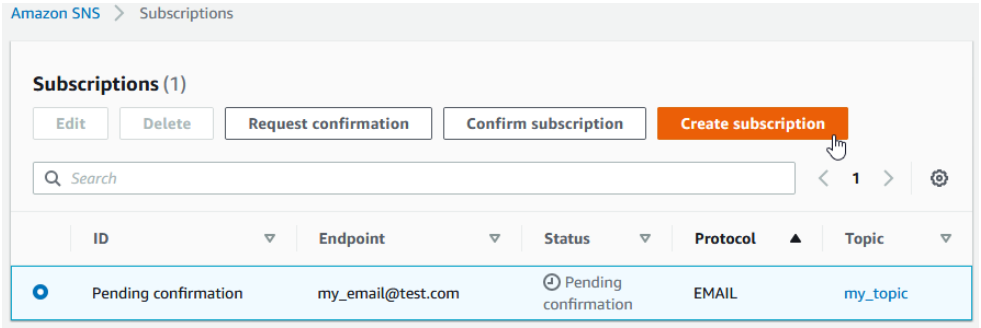


You should define the alarm state that will trigger the notification action. In our example we want to be notified when the metric is outside of the defined threshold, so we will select `in Alarm` option. An SNS (Simple Notification Service) topic will be

created to send the email notification. After creating a new SNS topic you will need to subscribe to it and you can perform this action clicking on `View in SNS Console`:



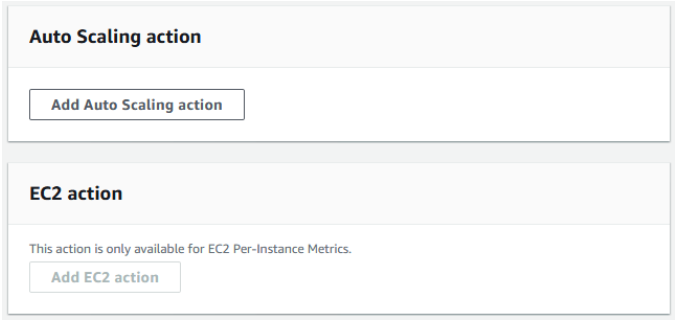
The SNS console has a button to create the subscription:



From your email application, you have to open the message from AWS Notifications and confirm your subscription.

Note that you can send multiple notifications for the same alarm state or for different alarm states, clicking on the `Add notification` button.

The following sections of the second step allows you to perform Auto Scaling or EC2 actions:

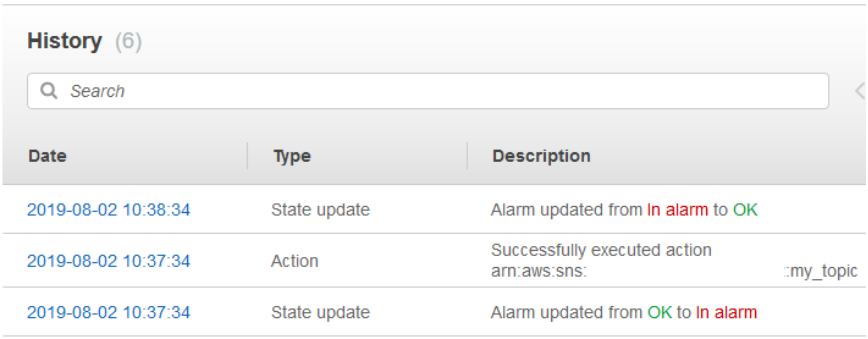


Now, you should continue with the third step where you must enter a name and a description for the alarm.

Finally, in the fourth step, you can confirm that the information and conditions are correct and then create the alarm.

You can also create an alarm that watches the result of a math expression based on your metrics, instead of watching a single metric. The alarm creation process in this scenario is the same as described for metric based alarms.

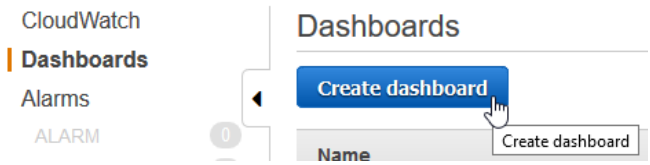
On the Alarms panel you can view the details of an alarm clicking on its name. The alarm history, available for 14 days, is also accessible in this view.


History (6)
Search
Date
Type
Description
2019-08-02 10:38:34
State update
Alarm updated from In alarm to OK
2019-08-02 10:37:34
Action
Successfully executed action arn:aws:sns:
:my_topic
2019-08-02 10:37:34
State update
Alarm updated from OK to In alarm

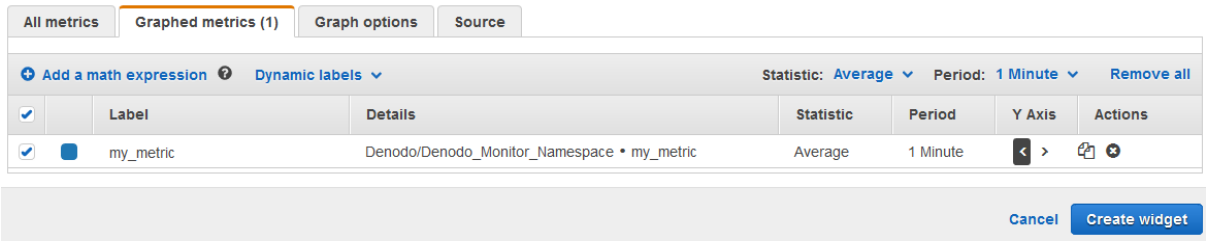
6 CREATING A DASHBOARD

Amazon CloudWatch offers you the possibility to create customized views of the metrics and alarms using dashboards.

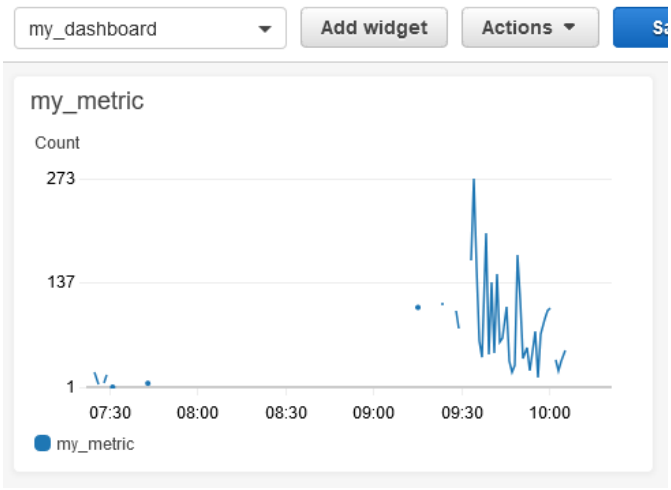
In the navigation panel you have to choose **Dashboards** and then **Create dashboard**.



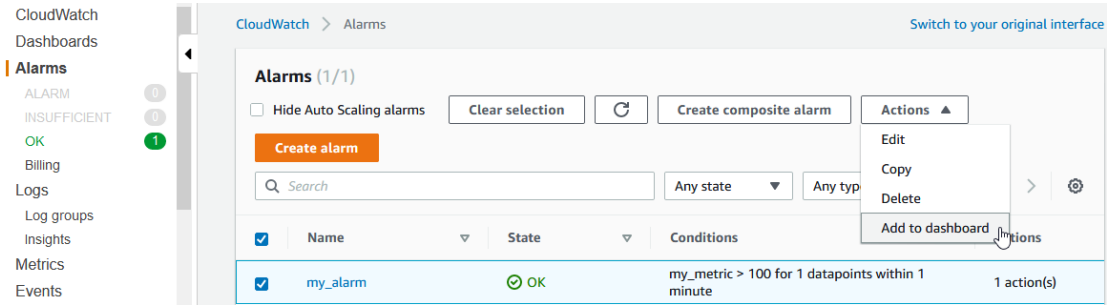
You must enter a name for the dashboard and use a dialog box to add a widget type. The custom metric of the example, representing the number of queries initiated per minute, can be added using a Line widget. Check, during the process, the values in the Graphed metrics tab to add the widget with the desired values. In our scenario, the period must be 1 minute.



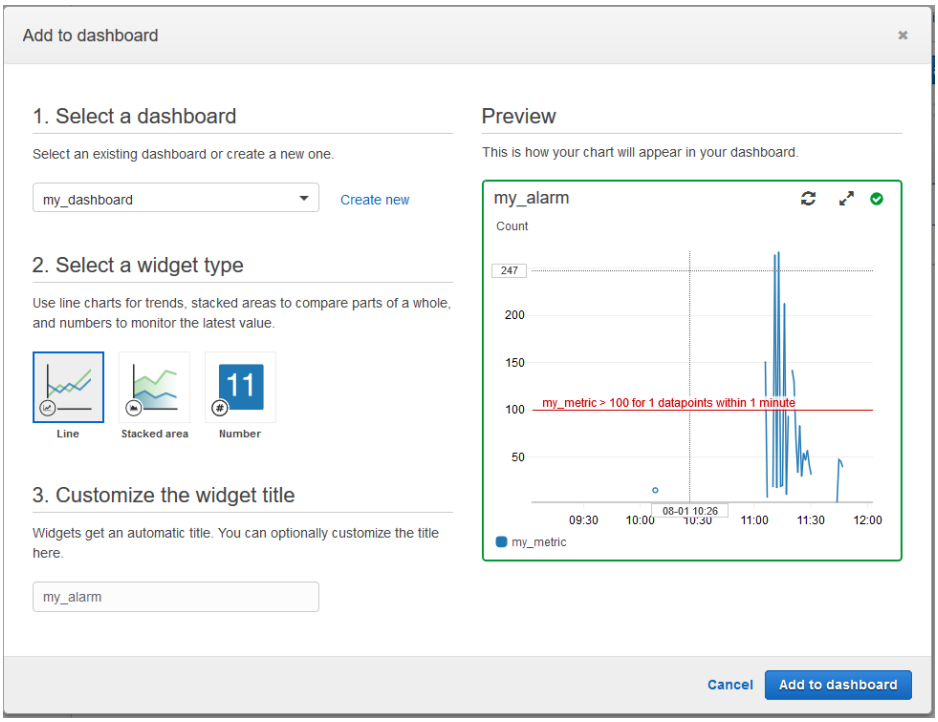
After this operation, a graph with the metric values is displayed in the dashboard.



You can also add alarms to the dashboard choosing the **Add to Dashboard** option, inside the **Actions** dropdown, in the **Alarms** section.



A new window will show you some information and a preview of this new widget.



Notice that when an alarm is on the dashboard, it turns red when it's in the ALARM state.