Configuring Auto Scaling of Denodo in AWS

Revision 20191029

Denodo North America & APAC: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA
Denodo Iberia & Latino América: Montalbán 5, 28014 Madrid, Spain
Denodo EMEA: 21st Floor, Portland House, Bressenden Place, London SW1E 5RS. UK
Denodo DACH: Karlstraße 10, 80333 München. Germany
www.denodo.com

CONTENTS

1 INTRODUCTION

In cloud environments (like AWS), a typical use case is to use auto scaling capabilities in order to allow client applications to increase / decrease servers capacity according to a different set of rules (time intervals, CPU, memory usage…). This ability to automatically adjust the capacity of a Denodo cluster in AWS require taking into account several aspects:

1. When a new instance is launched, the servers that are automatically launched in startup scripts, need to be registered in the Solution Manager in order to get a working license. In this document we will show step by step how to automatically register/deregister nodes in the Solution Manager when using auto scaling in AWS.

2. Denodo servers logs are stored on the local filesystem. For that reason, when you have a Denodo cluster in AWS in which you want to switch off instances when you no longer need them or even control the switch on and switch off from a scheduled action or from an auto scaling action based on a Denodo custom metric; you should save the logging information in Amazon S3 to keep the log data accessible even if the instance is terminated or deleted. In this way you avoid losing log information.

   You can find detailed information about the configuration process in the How to store Denodo logs in Amazon S3 document.

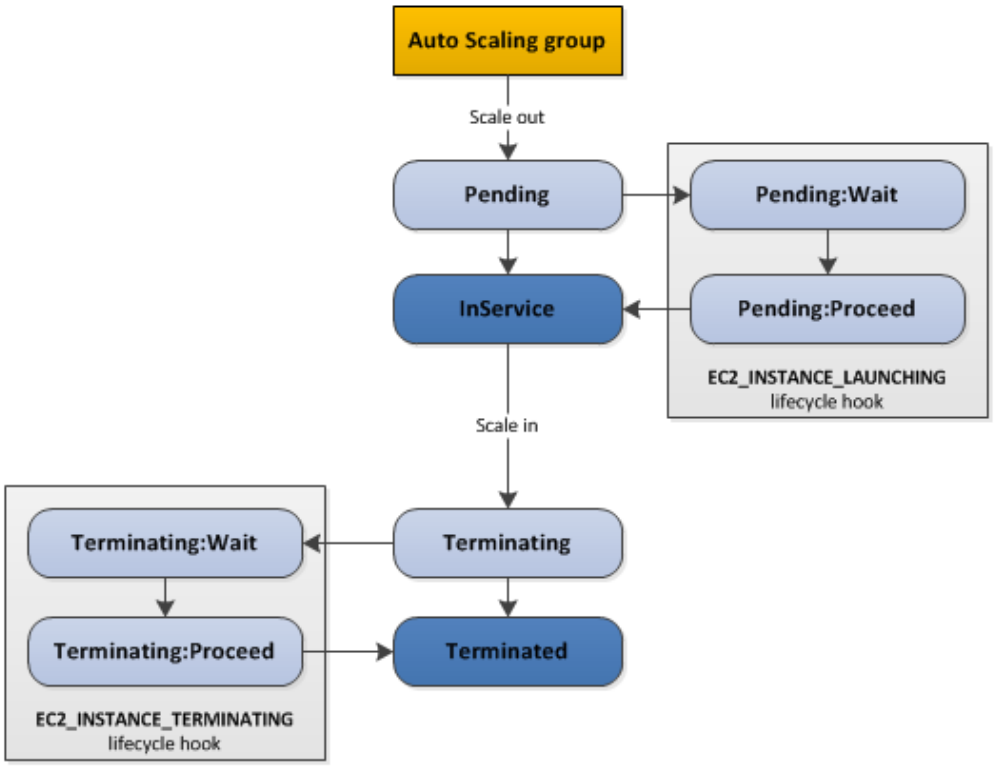
   Notice that Denodo Monitor Logs are not affected by this problem because Denodo Monitor generates the log files in the <SOLUTION_MANAGER_HOME> folder.

In addition, as explained in the Monitoring Denodo with Amazon CloudWatch document, it is possible to expose metrics of Denodo to AWS Cloudwatch so they can be used when configuring auto scaling actions and policies. Therefore, this document also provides examples of how to configure an auto scaling group of Denodo instances in two scenarios: using a scheduled action and using a Denodo custom metric.

2 DYNAMICALLY REGISTERING/DEREGISTERING NODES IN THE SOLUTION MANAGER

2.1 OVERVIEW

The following image illustrates the auto scaling lifecycle in AWS when instances are dynamically launched or stopped:



It is possible to add a lifecycle hook when an instance is launched or terminated, in order to execute an operation / execute some code using a lambda function.

2.1.1 Scale-out process needs

When a new instance is launched it is necessary to register the new server in the Solution Manager catalog, so the Virtual DataPort server can get a working license and start correctly.

Ideally, the servers could register themselves at startup time to avoid the manual registration. This could be done:

a. In a startup script
b. Or in a lambda function executed during EC2_INSTANCE_LAUNCHING lifecycle hook.

In both cases, the script / code executed will need to:

- Log in against Solution Manager server (*/login* endpoint).
- Invoke */servers* endpoint with a post operation with the server data. The server name will be the instance id of the instance.

2.1.2 Scale-in process needs

When an auto scaling group terminates an instance, the instance is killed (it is not a normal shutdown), so the server might not be stopped normally and the license usage not released. It is necessary to free the license usage for the server and delete the server from Solution Manager catalog. These operations can be executed in a terminate instance lifecycle hook.

In order to perform the desired operations, a lambda function will execute the following operations:

- Log in against the Solution Manager server (*/login* endpoint).
- Invoke /servers/deleteCloudServer endpoint with the instance id. This operation will try to find a server with a name corresponding to the given instance id. If the server exists the server is deleted. Also, if the server has an active license usage, this license usage is released.

2.2 CONFIGURE THE SCALE-IN PROCESS

In this section we describe how to configure the scale-in process. The sequence of steps we will follow is:

- Download and configure the script that will be executed when an instance is terminated.
- Create / Import a lambda function that will release the license for the terminated instance and will delete the instance from the Solution Manager catalog.
- Create a CloudWatch event that will redirect termination events for the auto scaling group to the lambda created in previous step.
- Add an EC2_INSTANCE_TERMINATING lifecycle hook to the auto scaling group to trigger the event.

2.2.1 Terminate instances script and configuration

A deployment package is a ZIP archive that contains function code and dependencies. The deployment package of the lambda function contains:

- *Terminate_instance.py* script: deletes a server in the Solution Manager catalog and, if the server was up and running, it releases the corresponding license usage entry.
- *TerminateInstanceConfiguration.properties* file: configuration file.
- Another libraries / dependencies to run the script.

You can download the deployment package of this lambda function here: Terminate Instance Package

Unzip the deployment package in a folder.

Edit the *TerminateInstanceConfiguration.properties* file to configure the necessary data to access the Solution Manager. For instance:

```
com.denodo.sm.host=localhost
com.denodo.sm.port=10090

com.denodo.sm.user=username
com.denodo.sm.password=clearOrEncriptedPassword

com.denodo.sm.sslEnabled=false
```

Where:

- `com.denodo.sm.host`: IP of the Solution Manager server.
- `com.denodo.sm.port`: port of the Solution Manager server (not license manager server port).
- `com.denodo.sm.user`: User to authenticate against Solution Manager server.
- `com.denodo.sm.password`: Password to authenticate against Solution Manager server. The password value can be encrypted using "encrypt_password" script available in /bin folder of a Denodo Platform installation.
- `com.denodo.sm.sslEnabled`: set this property to "true" if you have SSL/TLS enabled in the Solution Manager server.

When the configuration is ready, zip all the files inside the deployment package folder.

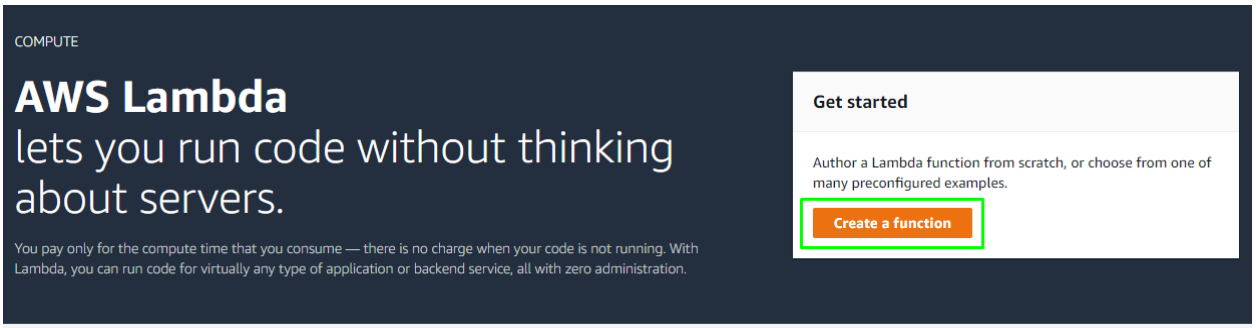
Now we need to create the lambda function in AWS and import the deployment package.

2.2.2 Create Lambda function

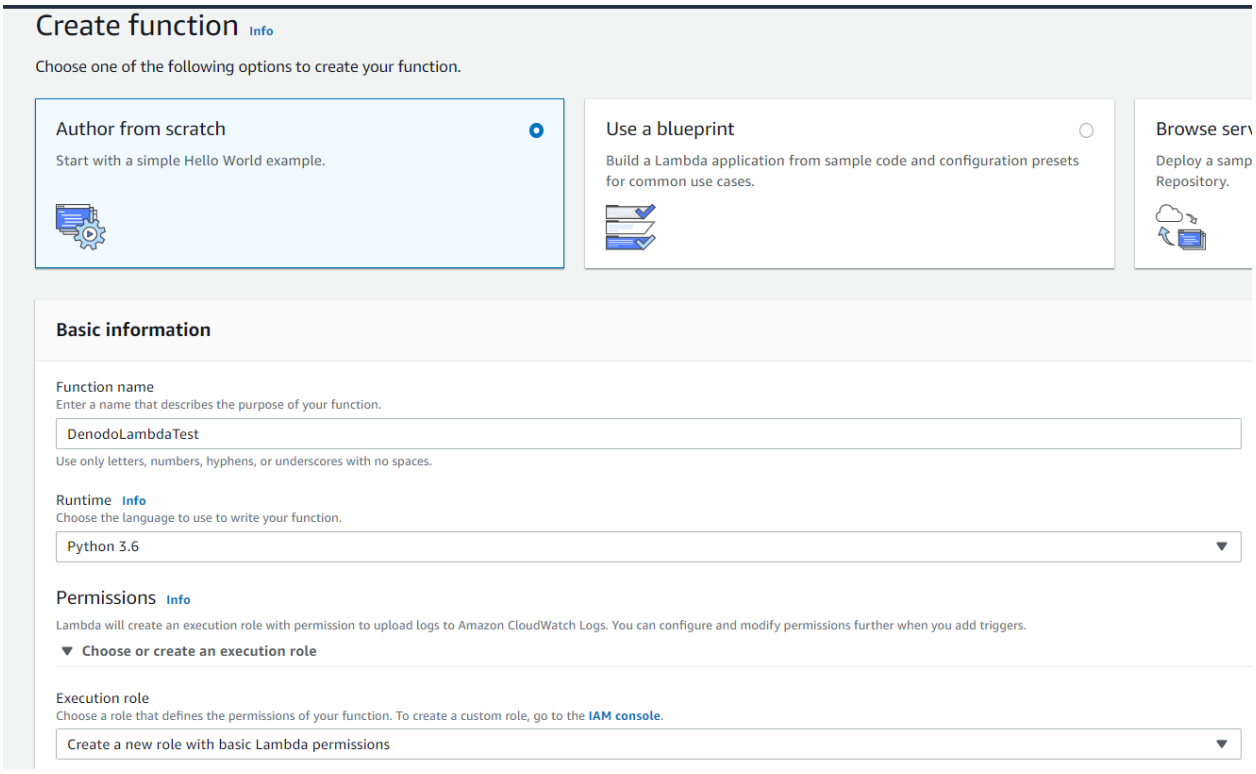
The documentation regarding lambda functions is available here.

In order to work with lambda functions there are several approaches. To test simple scripts, the easiest way is to use the lambda console :

1. Open the lambda console and click on option "Create a function" (this example link uses eu-central-1 region, make sure to select your region).



2. Select the *Author from scratch* option to create a lambda function and select the Python 3.6 runtime.

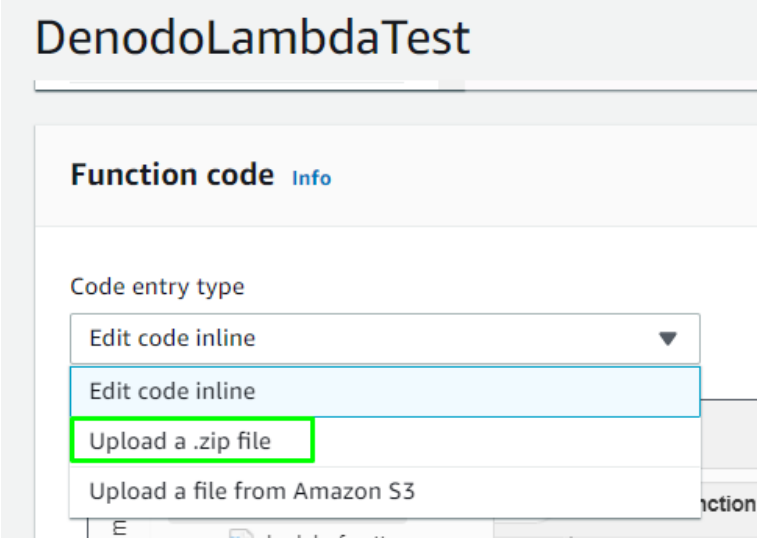


3. In the “Permissions” section, leave the default option. AWS will create and assign a role with the basic permissions to execute the lambda function.

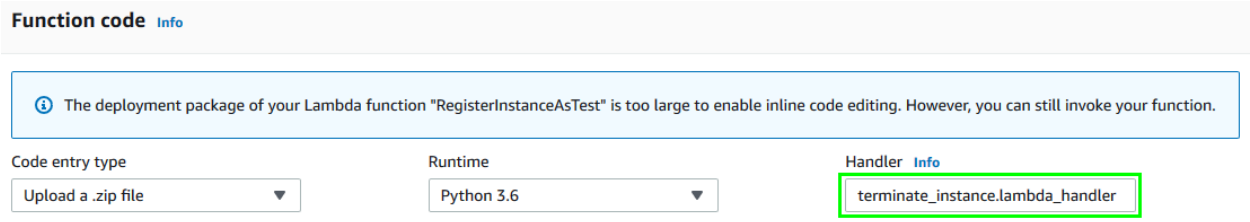
2.2.2.1 Import Lambda function

Once the basic lambda function is created, we will update it with zip file that we created in the previous step.

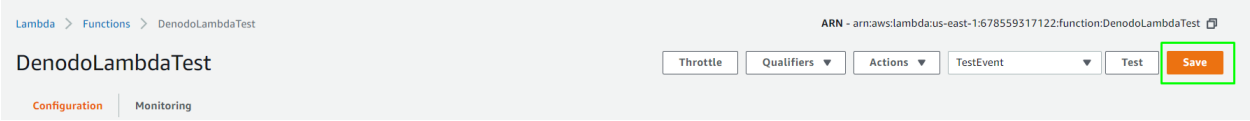
In order to import the deployment package of the lambda function, select the option “Upload a .zip file” and then select the zip file in your machine.



Update Handler to `terminate_instance.lambda_handler`



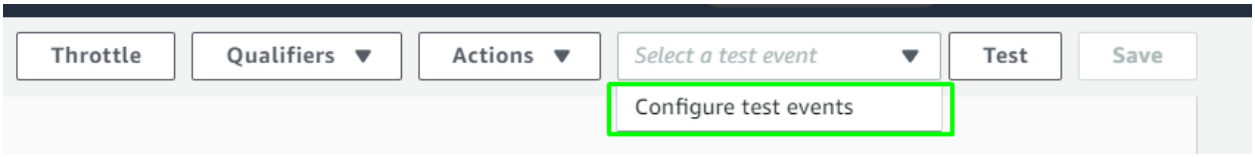
Press the “Save” button in order to upload the function in the zip file.



2.2.2.2 Test the lambda function

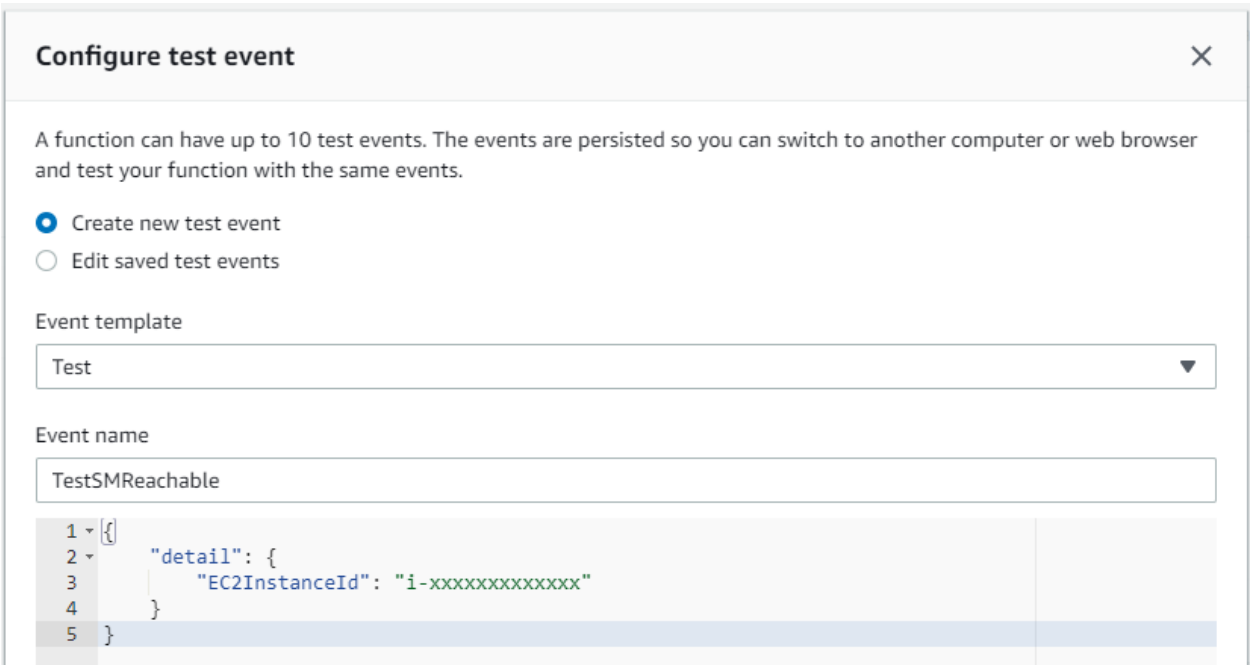
The script defines a “lambda_handler” function to handle termination events for instances.

1. It is possible to configure test events to execute the lambda with a specific event input.

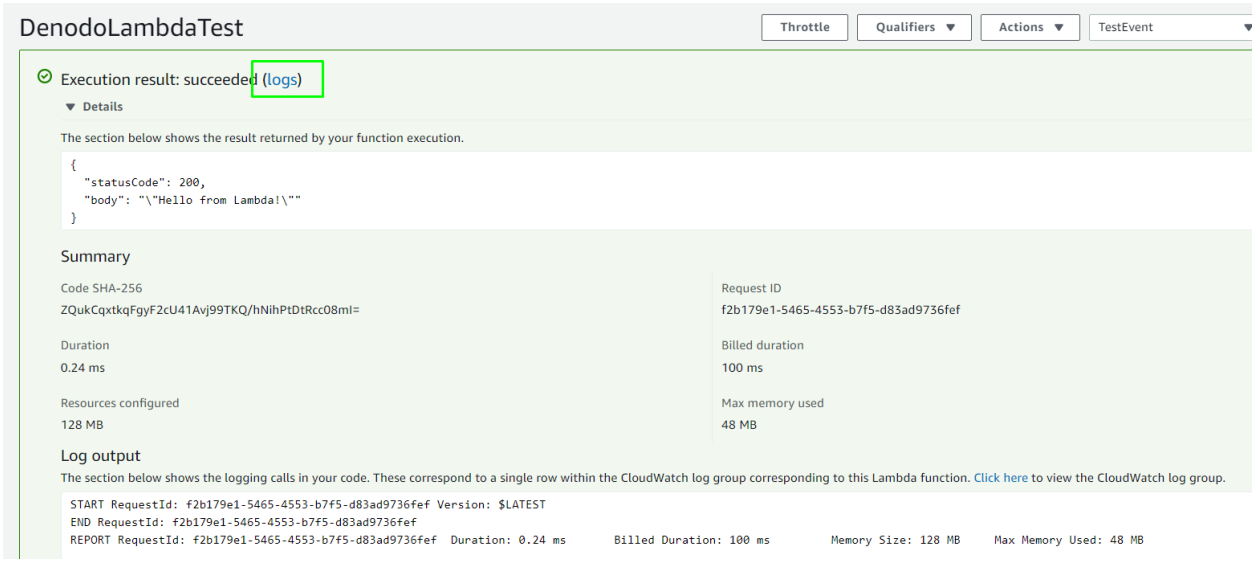


You can test that the lambda function works correctly and has access to the Solution Manager server. Paste the following json in the event content:

```
{
      "detail": {
            "EC2InstanceId": "i-xxxxxxxxxxxxx"
      }
}
```



2. To execute the test, select the event to simulate and press the “Test” button. You can check the execution result of the function in the execution result output and the logs stored in CloudWatch by clicking the “logs” option. In the case of the example json, the execution should fail if the instance “i-xxxxxxxxxxxxx” is not registered in the Solution Manager server.



2.2.3 Create CloudWatchEvent

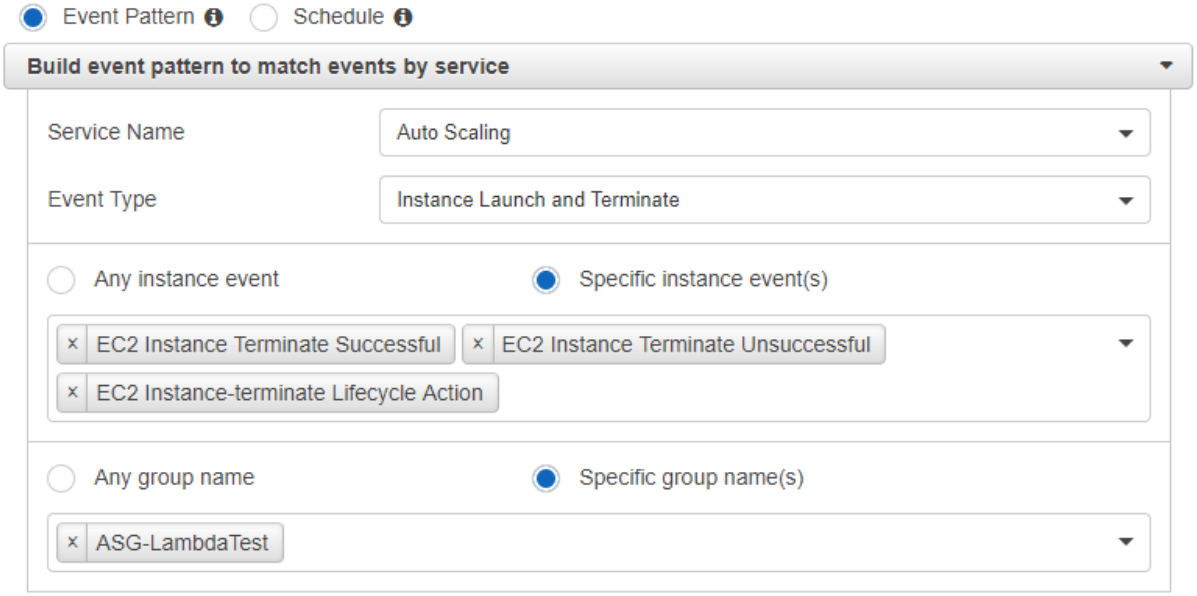
We will use CloudWatch events to invoke the lambda function every time the auto-scaling group terminates an instance.

You can read more about the lifecycle hooks and notifications possibilities here.

In order to create and configure a CloudWatch event to invoke the desired lambda function during instance termination, you can follow the next steps:

1. Open the CloudWatch console.
2. Create a new rule in “Events” - “Rules”
    a. In “Event Source”, select “Event Pattern”.
    b. Select the option “”Events by service”

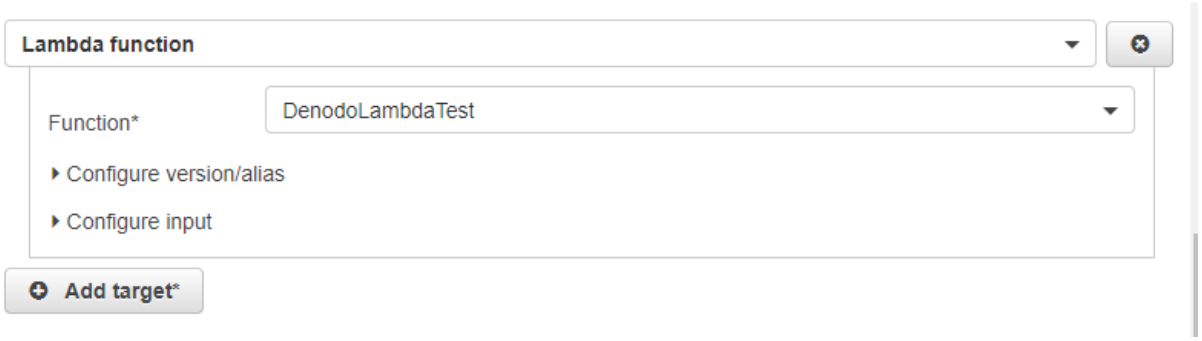
    c. Select the “Auto Scaling” service and the “Instance Launch and Terminate” option in event type. Select the 3 terminate instance events and the desired auto scaling group as shown below.



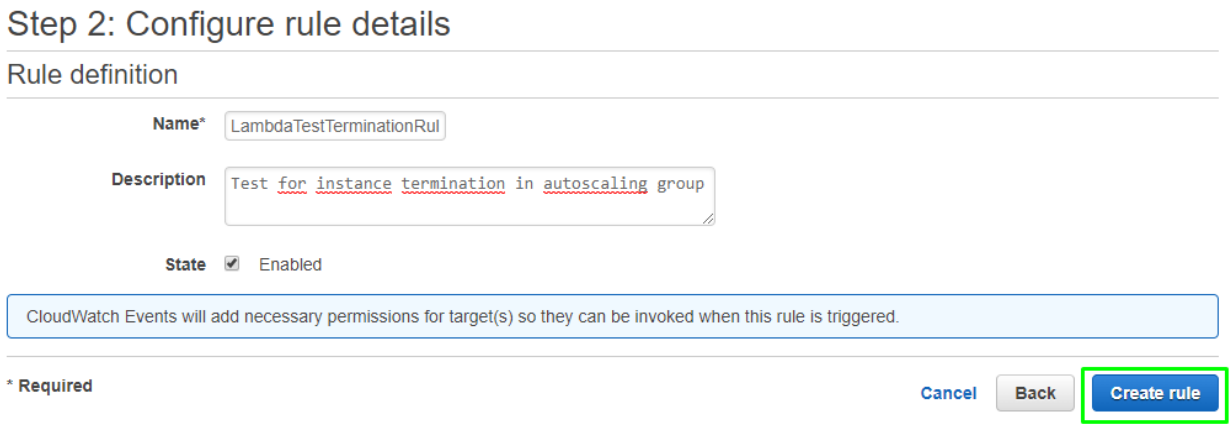
Alternatively, we could also create the event editing the "Event Pattern" textarea with the following JSON (changing the *AutoScalingGroupName* attribute to the name of the corresponding auto scaling group):

```
{
  "source": [
    "aws.autoscaling"
  ],
  "detail-type": [
    "EC2 Instance Terminate Successful",
    "EC2 Instance Terminate Unsuccessful",
    "EC2 Instance-terminate Lifecycle Action"
  ],
  "detail": {
    "AutoScalingGroupName": [
      "ASG-LambdaTest"
    ]
  }
}
```

3. Add the Lambda function as a target of the event rule. Click "Add target" option and select the lambda function.



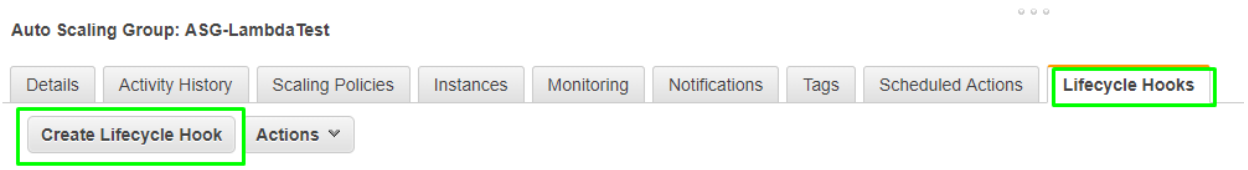
4. Click “Configure details” option at the bottom and fill in the name and description of the rule. Leave the state option check “Enabled” and click on the “Create rule” option.



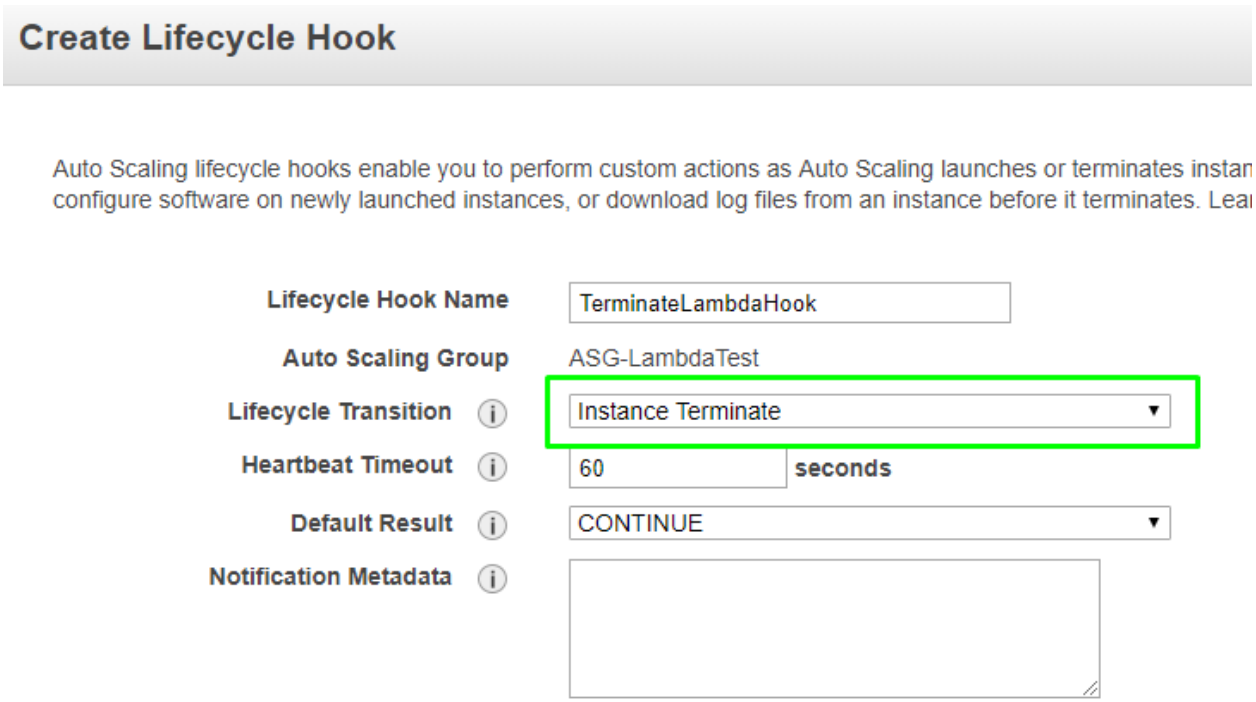
2.2.4 Add Lifecycle hook to auto scaling group

Once we have the lambda function and the CloudWatch event created, we need to add the lifecycle hook to the auto scaling group for the terminating instance phase.

In order to create the Lifecycle Hook, go to the “Lifecycle Hooks” tab inside the auto scaling group and click “Create Lifecycle Hook”.



Fill the information with the configuration you want:



The Heartbeat timeout in a termination hook is the time that the instance remains in the Terminating:Wait state of the cycle. We recommend to change it to a lower value, about 60 seconds (default is 3600 seconds) than the default one to proceed with the termination process.

2.3 CONFIGURING THE SCALE-OUT PROCESS

In this section we describe how to configure the scale-out process. The sequence of steps we will follow is similar to the previous section:

- Download and configure the script that will be executed when a new instance is launched.
- Create / Import a lambda function that will register the new instance in the Solution Manager catalog.
- Create a CloudWatch event that will redirect launch events of the auto scaling group to the lambda created in previous step.
- Add an EC2_INSTANCE_LAUNCHING lifecycle hook to the auto scaling group to trigger the event.

2.3.1 Start instances script and configuration

In the same way as the terminate instance, there is a deployment package for the register lambda function. This package contains:

- register_autoscaling_server.py script: Obtains the private IP of the launched instance with the describe instances operation and registers the server in the Solution Manager.
- ServerData.json file: configuration file. In this configuration file you can define the Solution Manager connection properties and other values to register the server in the Solution Manager.
- Other libraries / dependencies to run the script.

You can download the deployment package of this lambda function here:
Register Server Package

Unzip the deployment package in a folder.

Edit the "ServerData.json" file to configure the necessary data to access the Solution Manager and server default values:

The *"register_autoscaling_server.py"* script registers a server in a cluster in the Solution Manager catalog. The script receives a configuration file as an argument. This is an example configuration file:

```
{
 "com.denodo.sm.user" : "admin",
 "com.denodo.sm.password" : "clearOrEncryptedPassword",
 "com.denodo.sm.host" : "localhost",
 "com.denodo.sm.port" : 10090,
 "clusterId" : 2,
 "defaultDatabase" : "admin",
 "username" : "admin",
 "password" : "encryptedPassword",
 "port" : 9999,
 "useKerberos" : false,
 "usePassThrough" : false,
 "solutionManagerUsesSSL" : false
}
```

Where:

- `com.denodo.sm.user`: User to authenticate against Solution Manager server
- `com.denodo.sm.password`: Password to authenticate against Solution Manager server. The password value can be encrypted using "encrypt_password" script available in /bin folder of a Denodo Platform installation. You can also put the clear password, but it is not recommended.
- `com.denodo.sm.host`: IP of the Solution Manager server.
- `com.denodo.sm.port`: port where Solution Manager server is running.
- `clusterId`: identifier of the cluster where you want to register the servers. You can get the cluster identifier exporting the Solution Manager catalog and finding the id of the desired cluster or using the REST API listing the clusters for the desired environment.
- `defaultDatabase`: server default database.
- `username`: user used to connect to the server.
- `password`: the password used to connect to the server. Provide the password encrypted using "encrypt_password" script available in /bin folder of a Denodo Platform installation.
- `useKerberos`: flag to specify if kerberos is used.
- `usePassThrough`: create the revisions using the credentials of the user that is logged in the Solution Manager.
- `solutionManagerUsesSSL`: flag to specify if Solution Manager server is configured with SSL, so the script invokes an https or http endpoint accordingly.

The script registers the server in the solution manager with:

- Host / ip: The private IP of the AWS instance.
- Name: the instance id of the AWS virtual machine: **The name will be used to identify the servers, so it is mandatory to not update the name of these servers.**

The example assumes a scenario where the instances run in a private subnet (without public ips and unreachable from the internet). The Solution Manager server can be located in:

- The same VPC. In this case you will need to configure the “com.denodo.sm.host” property with the private IP address of the instance where it is running.
- On premises with a VPN to access the VPC in AWS. In this case, you can also configure the “com.denodo.sm.host” with the private IP of the Solution Manager server.

When the configuration is ready, save the changes and zip all the files inside the deployment package folder.

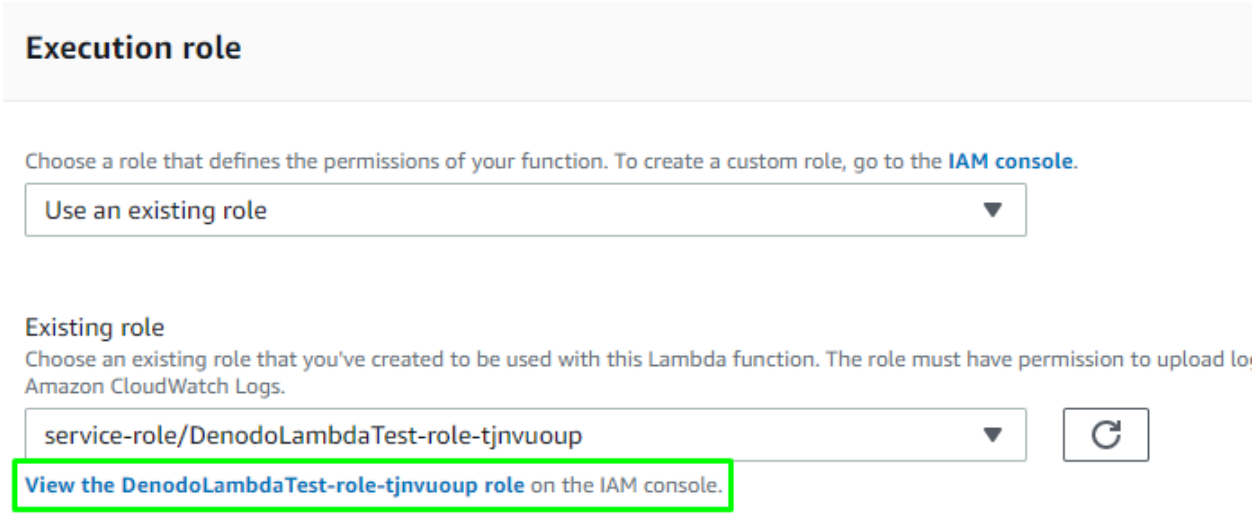
Now we need to create the lambda function in AWS and import the deployment package.

2.3.2 Create Lambda function

Perform the same steps as described in the terminate instances section.

Once the lambda function is created correctly, it is necessary to edit the role automatically created for the lambda function, in order to give the lambda function permissions to execute the DescribeInstances API operation invoked during the script execution.

Open the IAM console (you can access directly clicking on the role in the lambda function).

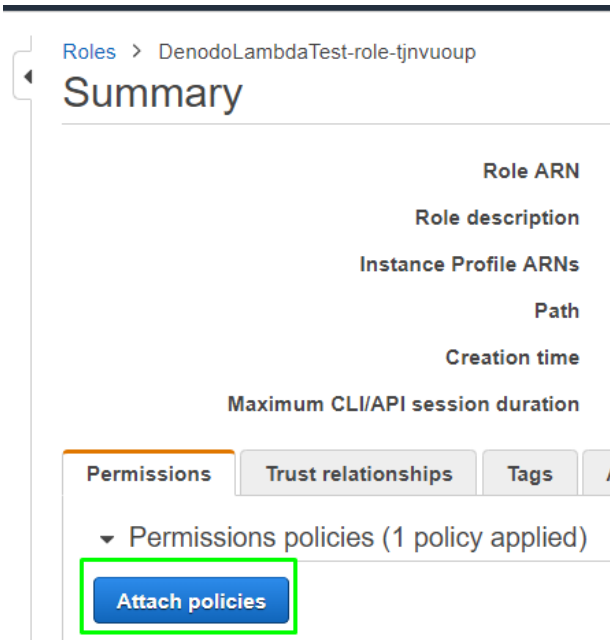


There are two options to add the new permissions:

1. You can create a new policy with the following permissions:

```
{
      "Version": "2012-10-17",
      "Statement": [{
            "Effect": "Allow",
            "Action": [
                  "ec2:DescribeInstances"
            ],
            "Resource": "*"
      }]
}
```

Then, attach the created policy to the role using the “Attach policies” option.



2. Or you can edit the policy using the “Edit policy” option
Add the following statement in the JSON tab to allow DescribeInstances operation

```
{
      "Effect": "Allow",
      "Action": [
            "ec2:DescribeInstances"
      ],
      "Resource": "*"
}
```

Visual editor | JSON

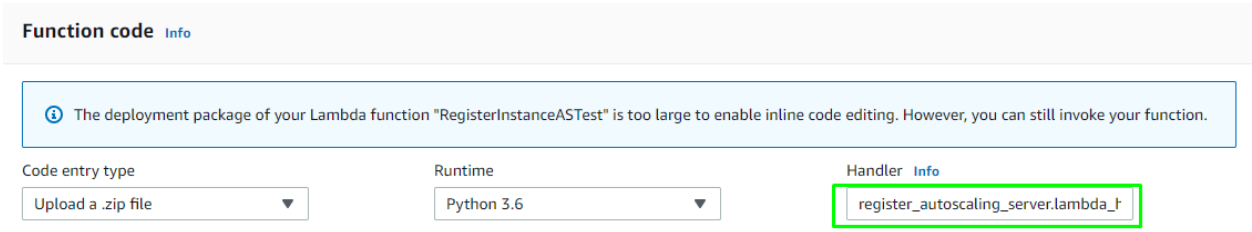
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-1:[redacted]:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:[redacted]:log-group:/aws/lambda/DenodoLambdaTest:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

Click review policy option and save the changes.

2.3.2.1 Import Lambda function

Import the lambda function as described before for terminate instances lambda function.

In this case, update 'Handler' to `register_autoscaling_server.lambda_handler` and save.



2.3.2.2 Test the lambda function

You can test the lambda function defining a test event and executing the lambda function like in the terminate instances scenario.

2.3.3 Create CloudWatchEvent

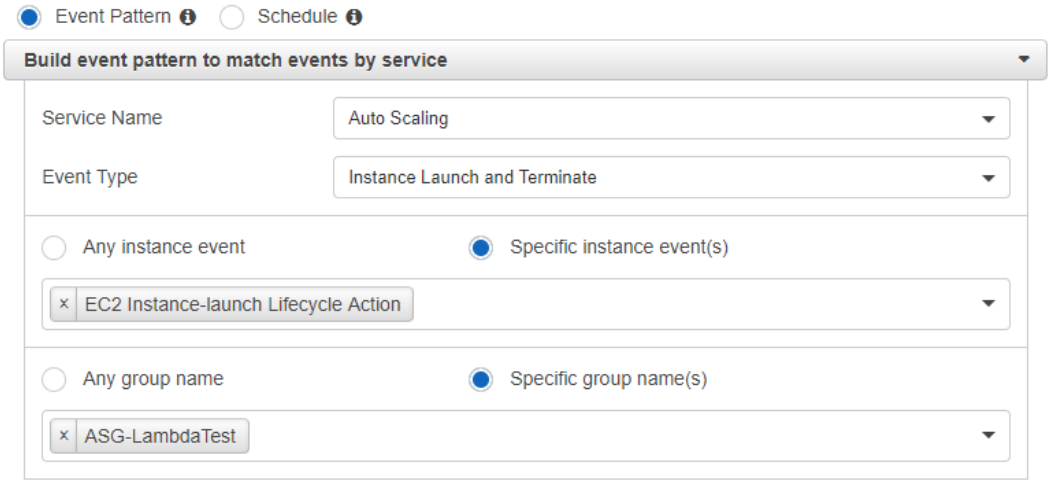
We will use CloudWatch events to invoke the lambda function every time the auto scaling group starts a new instance.

You can read more about the lifecycle hooks and notifications possibilities here.

In order to create and configure a CloudWatch event to invoke the desired lambda function during instance launch, you can follow the next steps:

1. Open CloudWatch console.
2. Create a new rule in “Events” - “Rules”
    a. In “Event Source”, select “Event Pattern”.
    b. Select the option “”Events by service”

    c. Select the “Auto Scaling” service and “Instance Launch and Terminate” option in event type. Select the “EC2 Instance-launch Lifecycle Action” event and the specific auto scaling group.

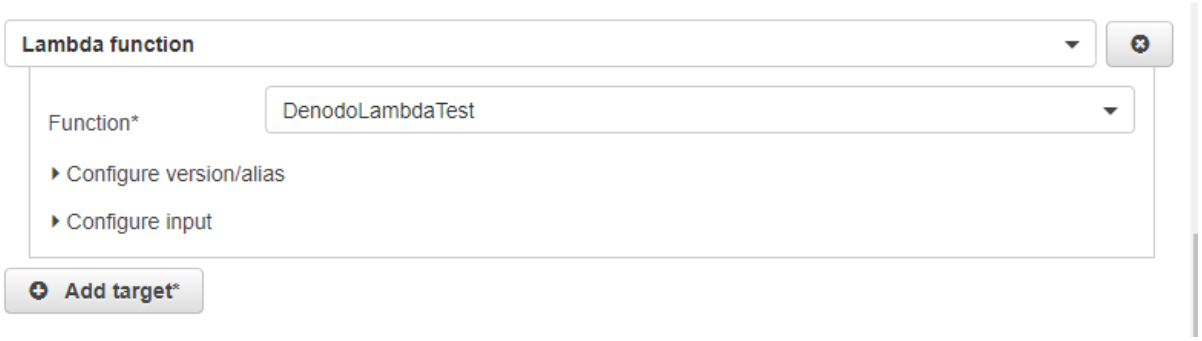


In this case, we could also create the event editing the “Event Pattern” textarea with the following JSON (changing the *AutoScalingGroupName* attribute to the name of the corresponding auto scaling group):

```
{
  "source": [
    "aws.autoscaling"
  ],
  "detail-type": [
    "EC2 Instance-launch Lifecycle Action"
  ],
```

```
  "detail": {
    "AutoScalingGroupName": [
      "ASG-LambdaTest"
    ]
  }
}
```

3. Add the Lambda function as a target of the event rule. Click “Add target” option and select the lambda function.

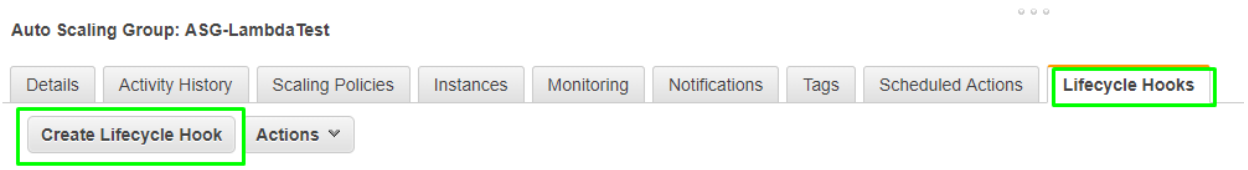


4. Click the “Configure details” option at the bottom and fill in the name and description of the rule. Leave the state option check “Enabled” and click on the “Create rule” option.

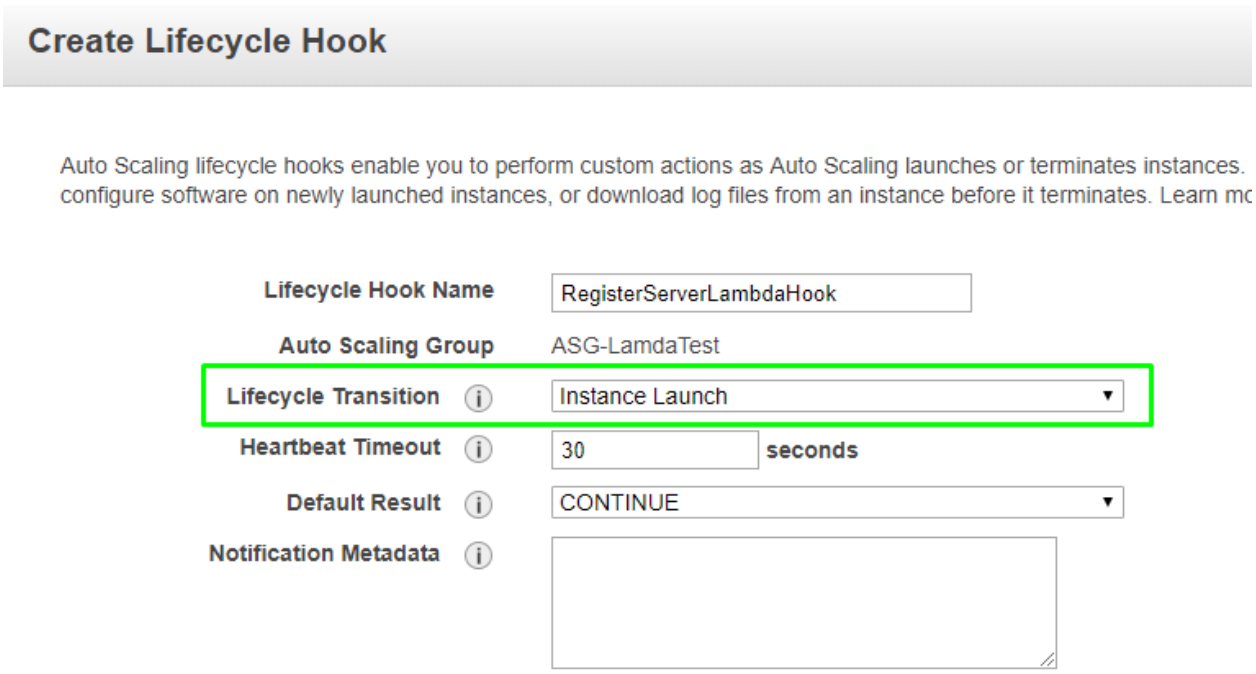
2.3.4 Add Lifecycle hook to auto scaling group

Once we have the lambda function and the CloudWatch event created, we need to add the lifecycle hook to the auto scaling group for the launching instance phase.

In order to create the Lifecycle Hook, go to “Lifecycle Hooks” tab inside the auto scaling group and click “Create Lifecycle Hook”.



Fill the information with the configuration you want:



The Heartbeat timeout in a termination hook is the time that the instance remains in the Pending:Wait state of the cycle. Change it to a lower value, about 30 seconds (default is 3600 seconds).

3 AUTO SCALING EXAMPLES

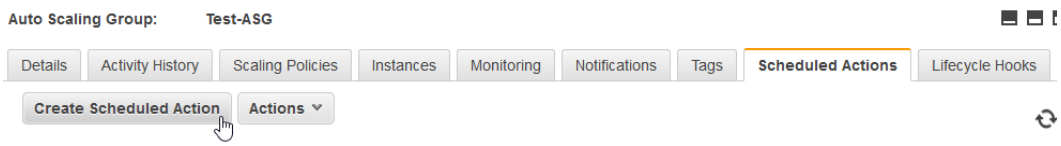
3.1 SCHEDULING AUTO SCALING

Some scenarios allow you to set your own scaling schedule due to predictable load changes. For example, you can detach and terminate an instance at night, when the cluster workload is lower and launch a new instance again in the morning.

First of all you may take into account the Auto Scaling group termination policy. It determines which instances to terminate when a scale-in event occurs and it is important because these instances enter the `Terminating` state and cannot be put back into service.

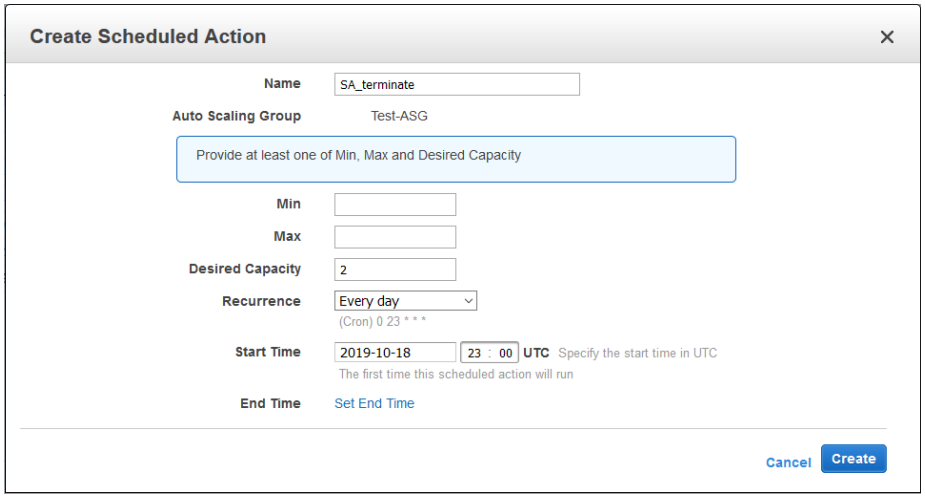
Let's imagine that you have an Auto Scaling group in which the number of instances you want to run is 3 and the minimum and maximum number of instances the Auto Scaling group should have at any time is 2 and 3, respectively. After reviewing the considerations mentioned above, you may create two scheduled actions in order to scale your Auto Scaling group on a recurring schedule:

1. Open the Amazon EC2 console.
2. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Select your Auto Scaling group, `Test-ASG` in our example.
4. On the **Scheduled Actions** tab, choose **Create Scheduled Action**.



5. On the Create Scheduled Action page you have to specify the size of the group establishing a new desired value. Continuing with the example, the desired value will be decreased by one unit at night and increased by one unit in the morning. Therefore, you have to create two scheduled actions.

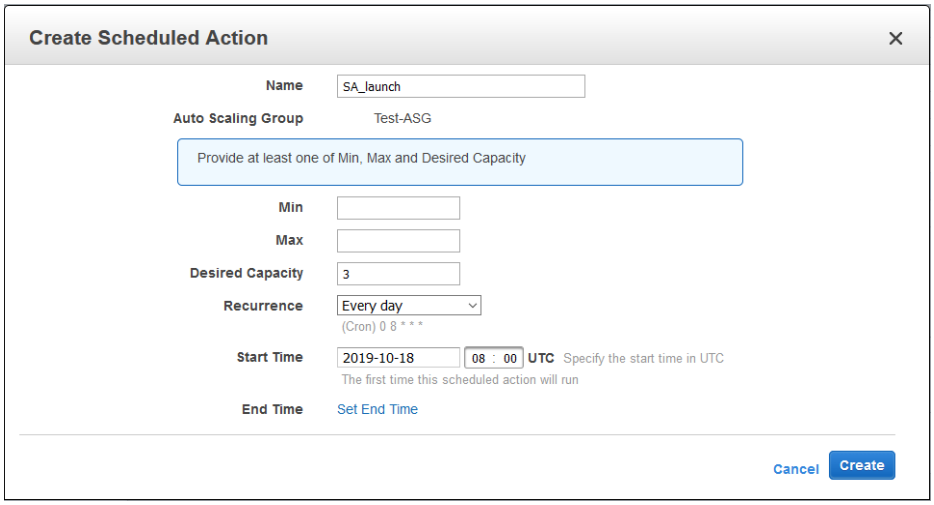
    - Scheduled action to terminate an instance at night.



The `Desired Capacity` field should be set to 2 with the aim of terminating one of the instances, because 2 instances will be sufficient to manage the workload during the night. In order to perform this action every night, the `Recurrence` must be set to `Every day` and then the cron expression will be created for you. The `Start Time`, 23:00 UTC, specifies the earliest time the action is performed.

With this action, every night at 23:00 UTC, a new instance will be terminated in the `Test-ASG` auto scaling group.

- Scheduled action to launch an instance in the morning



When the workload is expected to return to normal levels, we must again raise the `Desired Capacity` to 3. In order to perform this action every morning, the `Recurrence` must be set to `Every day` and then the cron expression will be created for you. The `Start Time`, 08:00 UTC, specifies the earliest time the action is performed.

With this action, every morning at 08:00 UTC, a new instance will be launched in the `Test-ASG` auto scaling group.

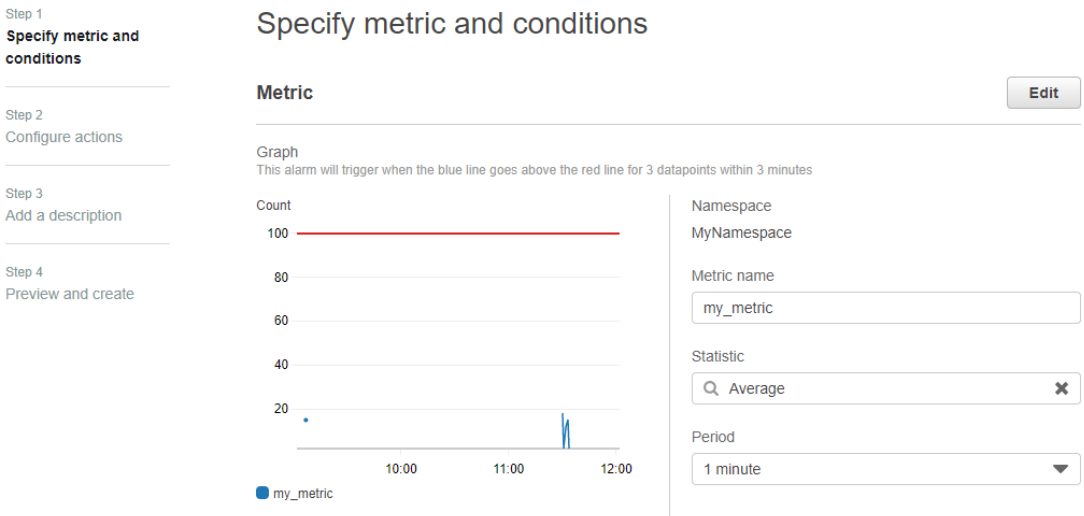
Note that the desired capacity must be less than or equal to the maximum size of the group. If your new value for `Desired` is greater than `Max`, you must update `Max`.

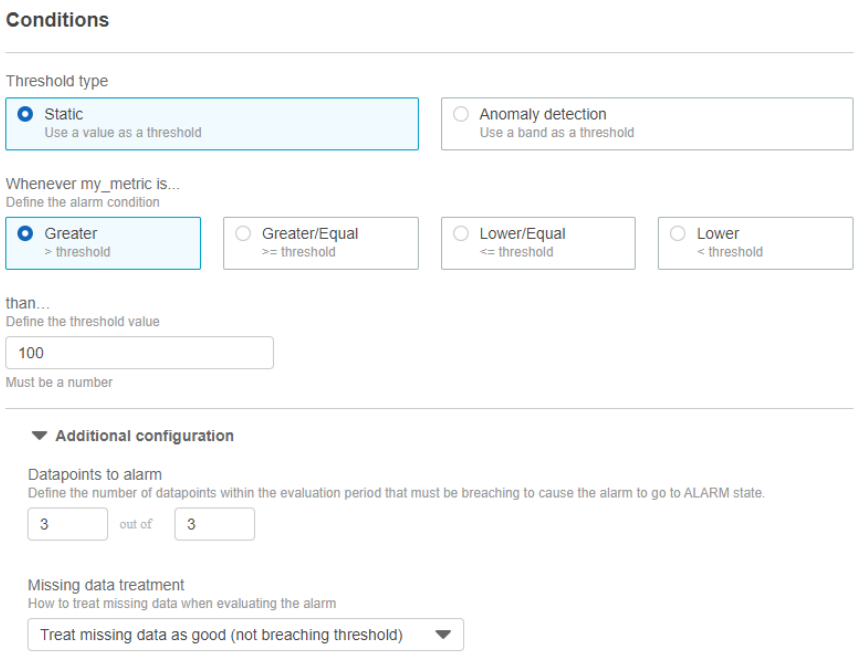
3.2 AUTO SCALING BASED ON A DENODO CUSTOM METRIC

CloudWatch allows you to create alarms based on metrics and define actions to be taken when the alarm changes the state. One of these available actions is the Amazon EC2 Auto Scaling action.

Auto Scaling groups should be created specifying the number of instances you want to run. In our example the desired number of instances is 2 and the minimum and maximum number of instances the Auto Scaling group should have at any time is 2 and 3, respectively. In this scenario, we need to configure an Auto Scaling action in response to a high workload that will increase the number of instances and also an Auto Scaling action in response to a low workload that will decrease the number of instances. But note that regardless of the actions, the Auto Scaling group will maintain the minimum and maximum number of instances without further configuration. That is why the execution of an action is going to fail when it tries to scale out but the number of running instances in the Auto Scaling group is 3 (the maximum) or when it tries to scale in and the number of instances is 2 (the minimum).

For example, you can develop a Denodo custom metric in CloudWatch for monitoring the number of requests initiated in Denodo per server every minute and add an alarm that goes to `ALARM` state when the number of queries in the last 3 minutes is greater than 100. Take into account that it is not available for anomaly detection alarms therefore you have to specify a static threshold in order to add this kind of action.





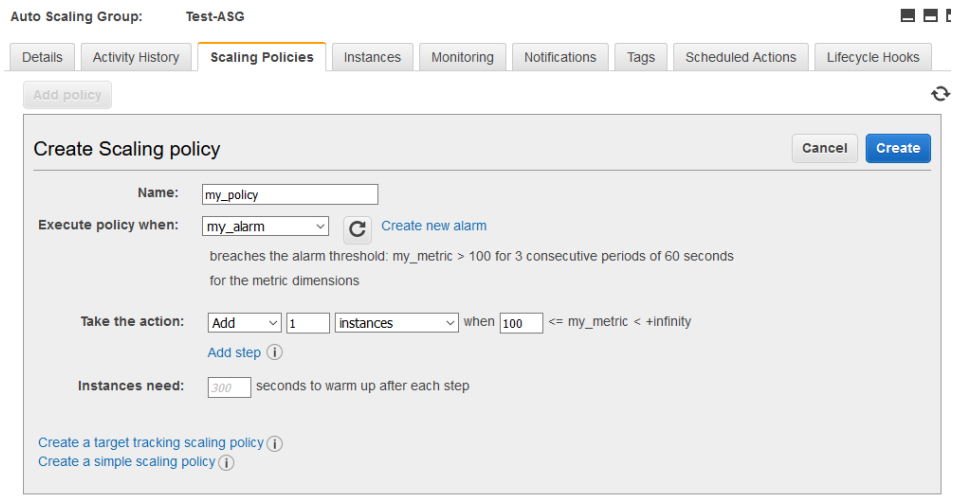
See the 'Monitoring Denodo Metrics' and 'Adding an alarm' sections of the Monitoring Denodo with Amazon CloudWatch document for more detailed information about creating Denodo custom metrics and adding alarms based on metrics.

When you are creating or editing an alarm, the second step allows you to configure the actions. Since you are going to use our alarm as an indicator of high workload, we want to launch a new instance in our Auto Scaling group, `Test-ASG`, whenever the alarm enters in the `ALARM` state. Similarly, you have to define as an indicator of low workload an alarm in which the `ALARM` state will enter when the number of requests initiated in Denodo per server every minute in the last 3 minutes is lower than 40.

On the other hand, a scaling policy associated with the Auto Scaling group is necessary to define an Auto Scaling action. Therefore, you must create the alarm, called `my_alarm`, without any action.

To create a scaling policy you should:

1. Open your EC2 console and choose **Auto Scaling Groups**, on the navigation pane.
2. Select your group, `Test-ASG` in our example, and go to **Scaling Policies** tab.
3. Click on the "**Add Policy"** button and select **Create a scaling policy with steps**.
4. Give a name to the policy and select the alarm created to warn of a high workload in `Executing policy when` field. In the `Take the action` section choose `Add 1 instances` to increase the capacity of the Auto Scaling group. You should leave the by default configuration for the upper bound for this step adjustment, which is the alarm threshold, and the lower bound is null (negative infinity).

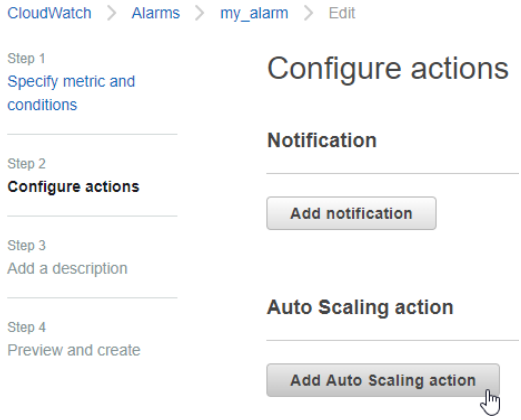


To create a scaling policy in response to the need of decreasing the number of instances, select the alarm created as an indicator of low workload in the `Execution policy when` field and in the `Take the action section` choose `Remove 1 instances`.

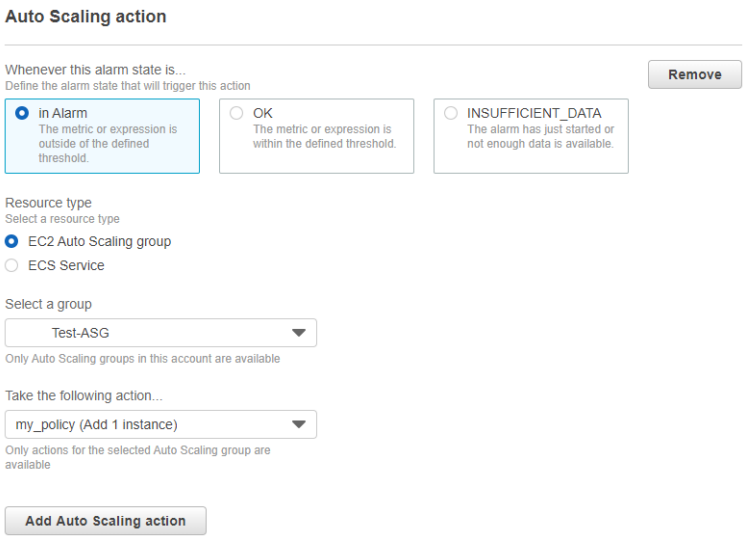
5. Click on the “Create” button.

Now you can add the Auto Scaling action to `my_alarm` alarm in CloudWatch editing it.

1. Click on the “Add Auto Scaling action” button in the `Configure actions` step.



2. Select:
   - `in Alarm` as the state that will trigger the action.
   - EC2 Auto Scaling group as resource type and select your group, `Test-ASG`.
   - Choose the policy, `my_policy`, in the `Take the following action` section.



3. Click on the “Update alarm” button.

Note that after an alarm invokes an Amazon EC2 Auto Scaling action due to a change in state, the alarm continues to invoke the action for every period that the alarm remains in the new state. Nevertheless, in the history of the alarm the action appears only once, between the two state updates: