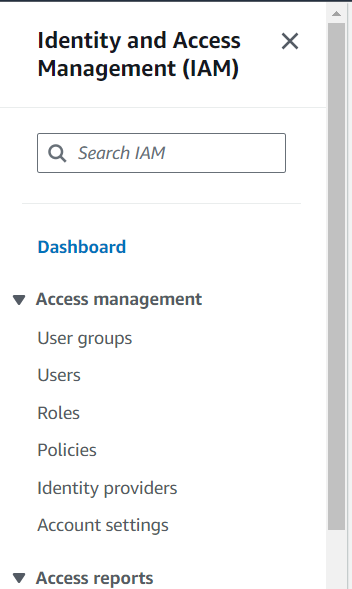
# Scheduling EC2 instance Auto Start and Stop:

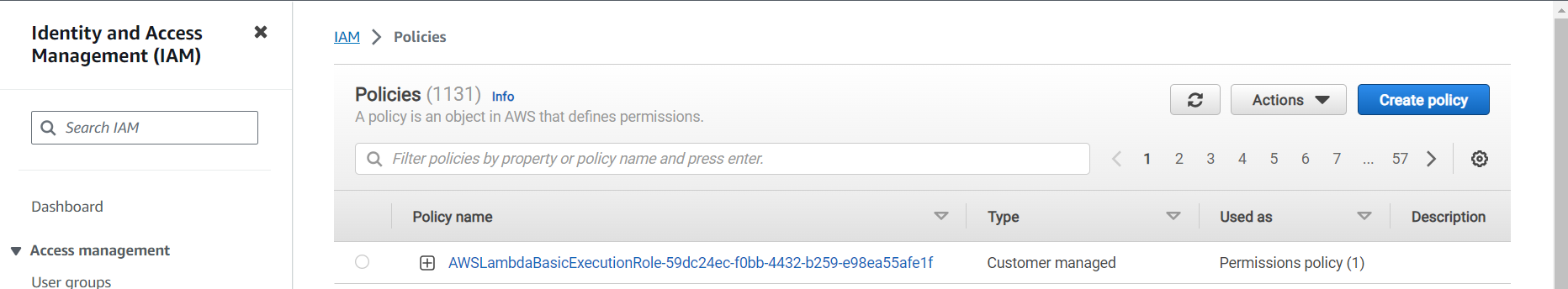
Process of scheduling the automatic start and stop of Amazon Elastic Compute Cloud (EC2) instances using AWS Lambda and Amazon CloudWatch Events.

## Create IAM Role for Lambda

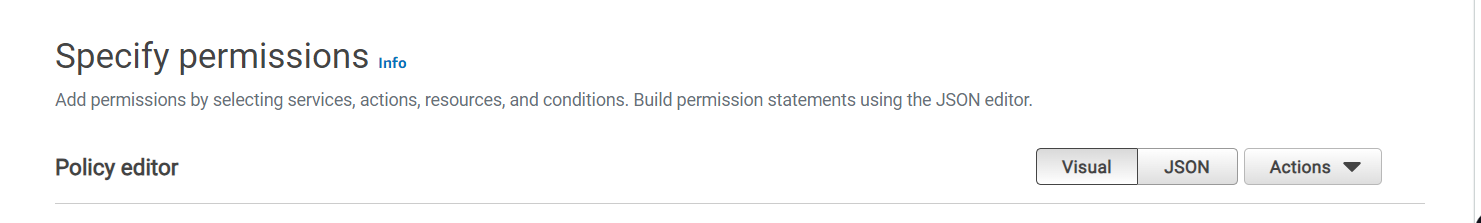
Go to the IAM Console.

### Go to Policies and create a policy.

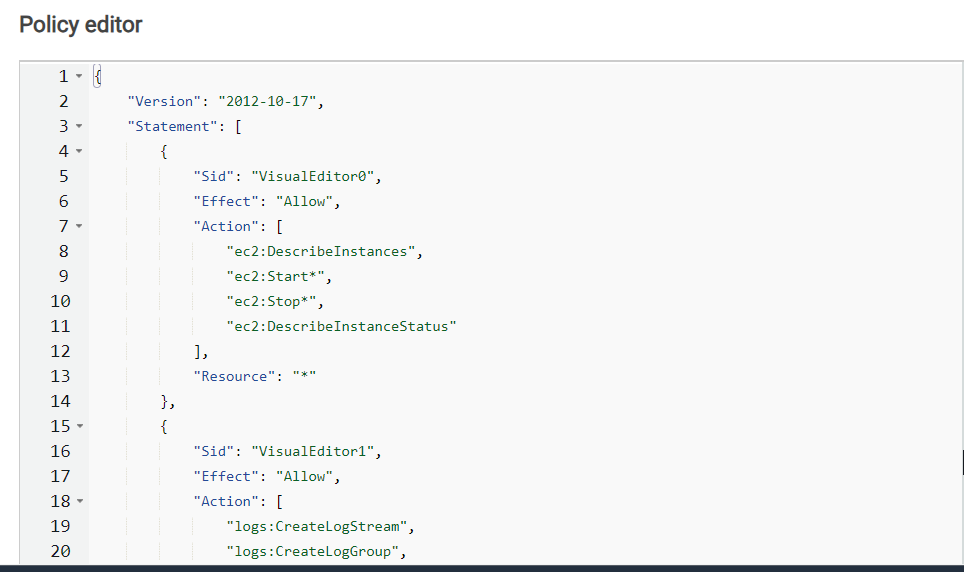




Click on JSON and paste the following JSON.



In the Policy Editor add the below script:



{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "VisualEditor0",

"Effect": "Allow",

"Action": [

"ec2:DescribeInstances",

"ec2:Start\*",

"ec2:Stop\*",

"ec2:DescribeInstanceStatus"

],

"Resource": "\*"

},

{

"Sid": "VisualEditor1",

"Effect": "Allow",

"Action": [

"logs:CreateLogStream",

"logs:CreateLogGroup",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:eu-central-1:60\*\*\*\*\*\*\*96:log-group:\*"

}

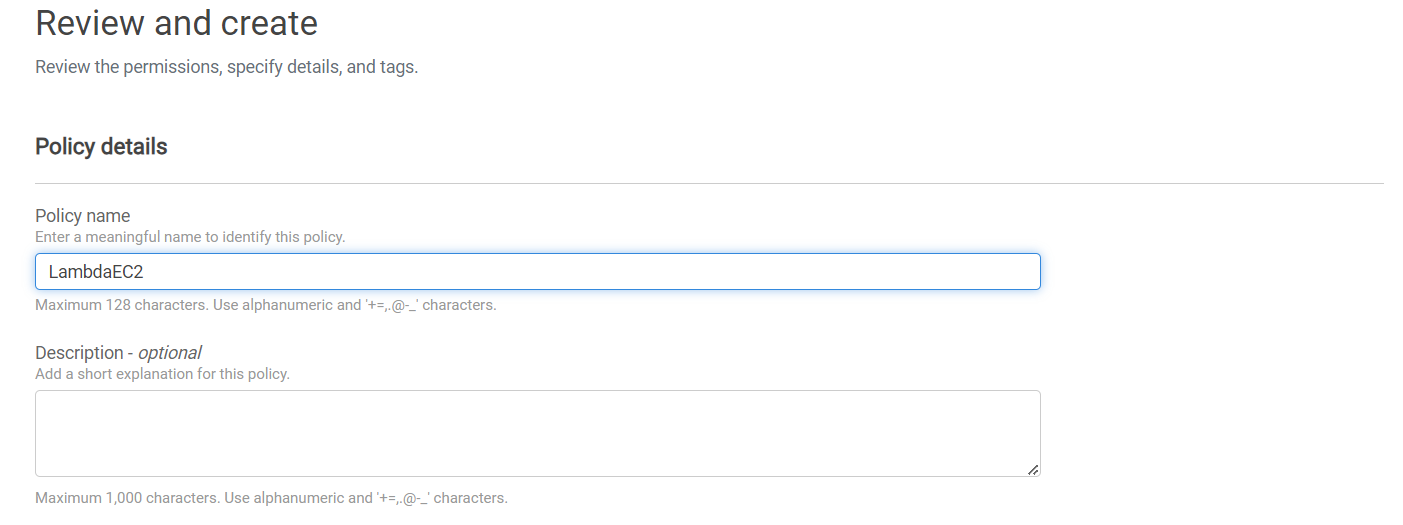
]

}

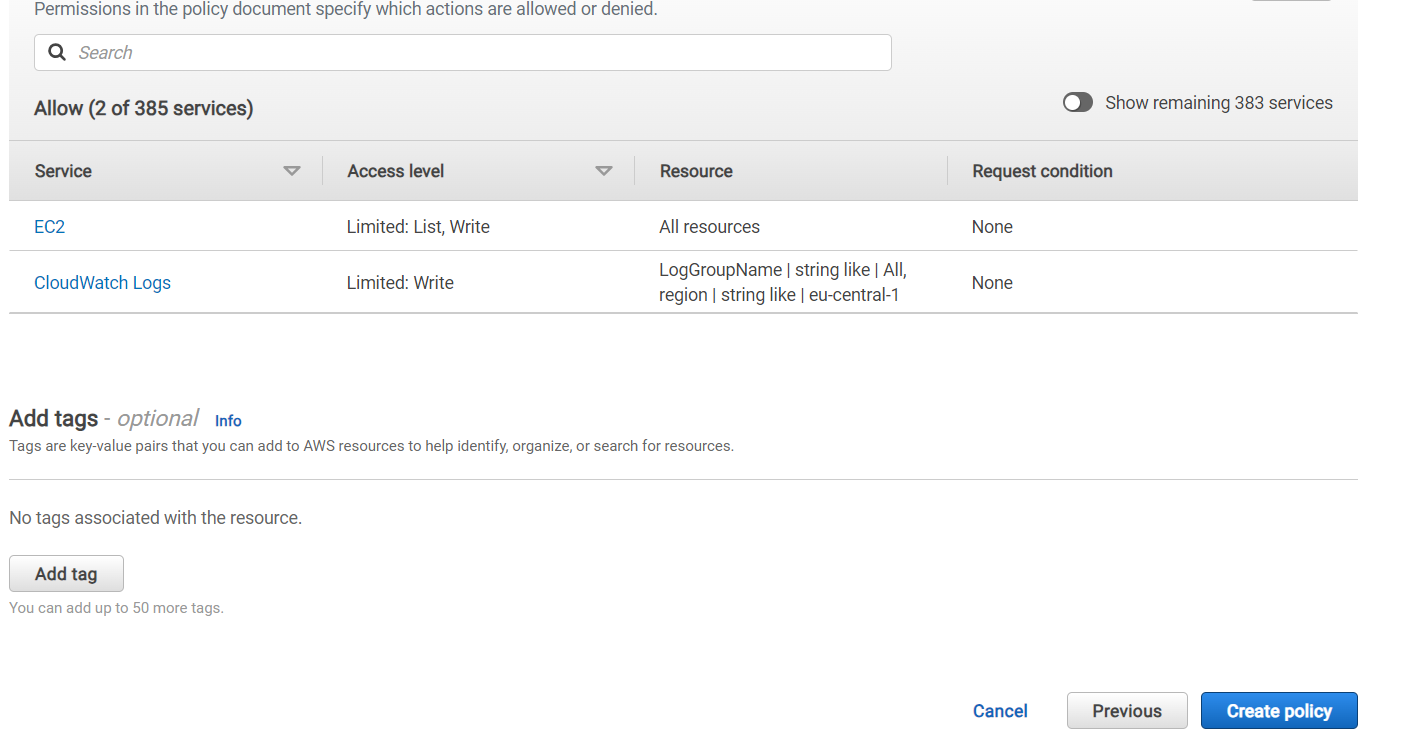
**\* "Resource": "arn:aws:logs:REGION:ACCOUNTID:log-group:\*"**

**Click on next button.**

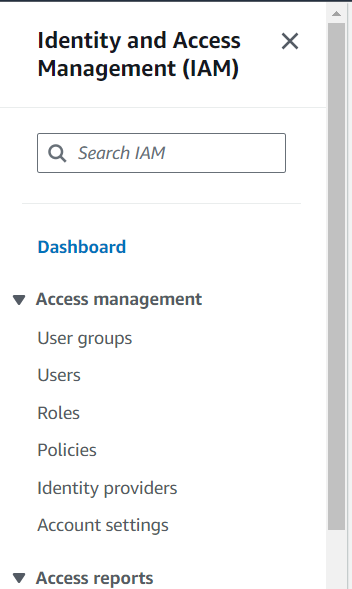
Then Enter Policy Name



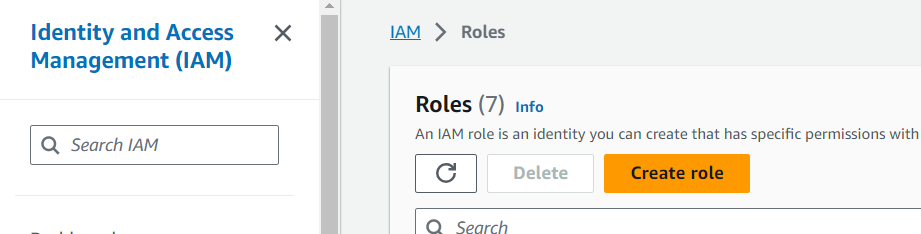
**Click on next button.**



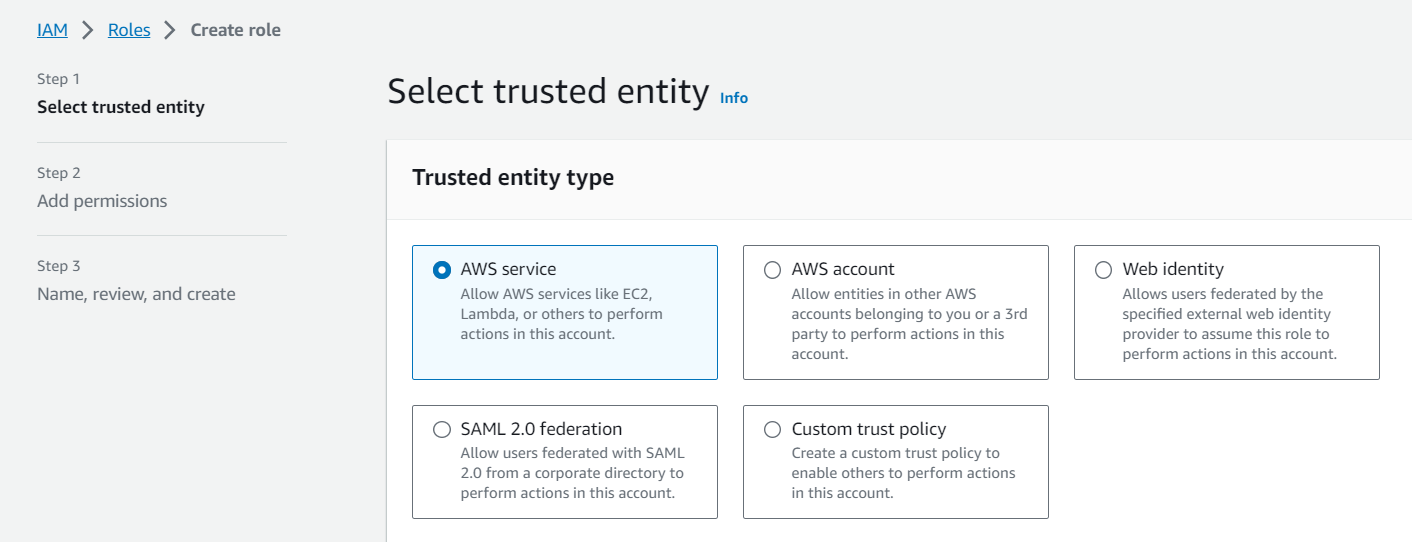
### Click on "Roles" in the left sidebar.



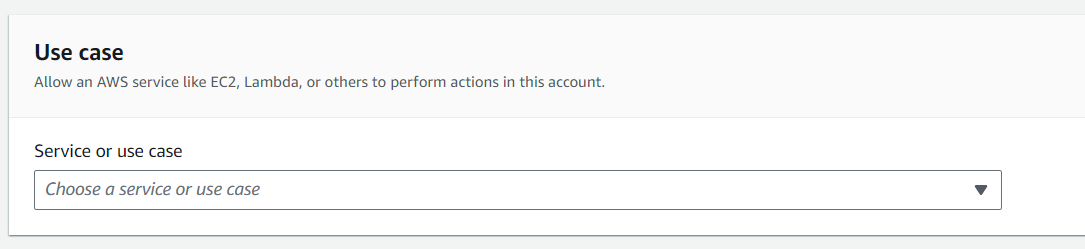
Click the "Create role" button.



Under "Select type of trusted entity," choose "AWS service."

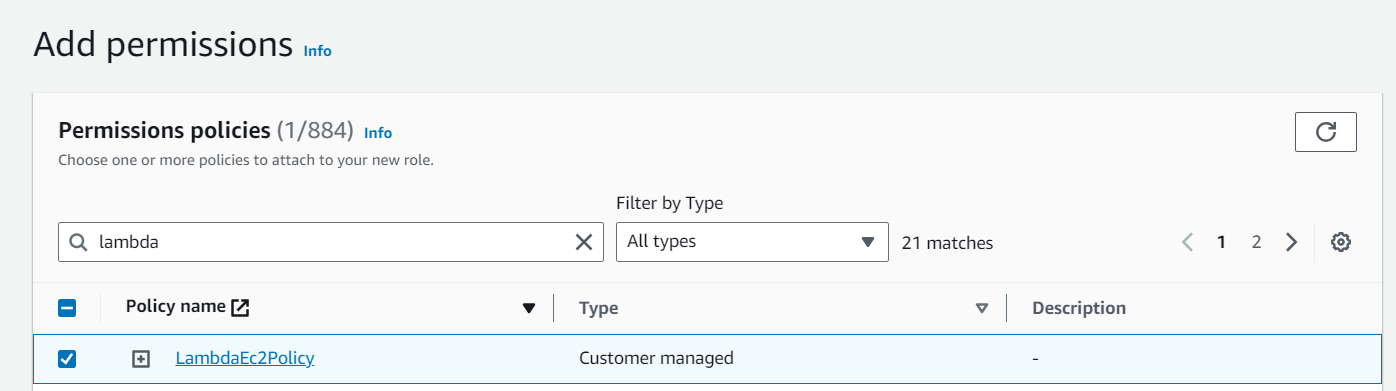


In the "Choose a use case" section, select "Lambda."

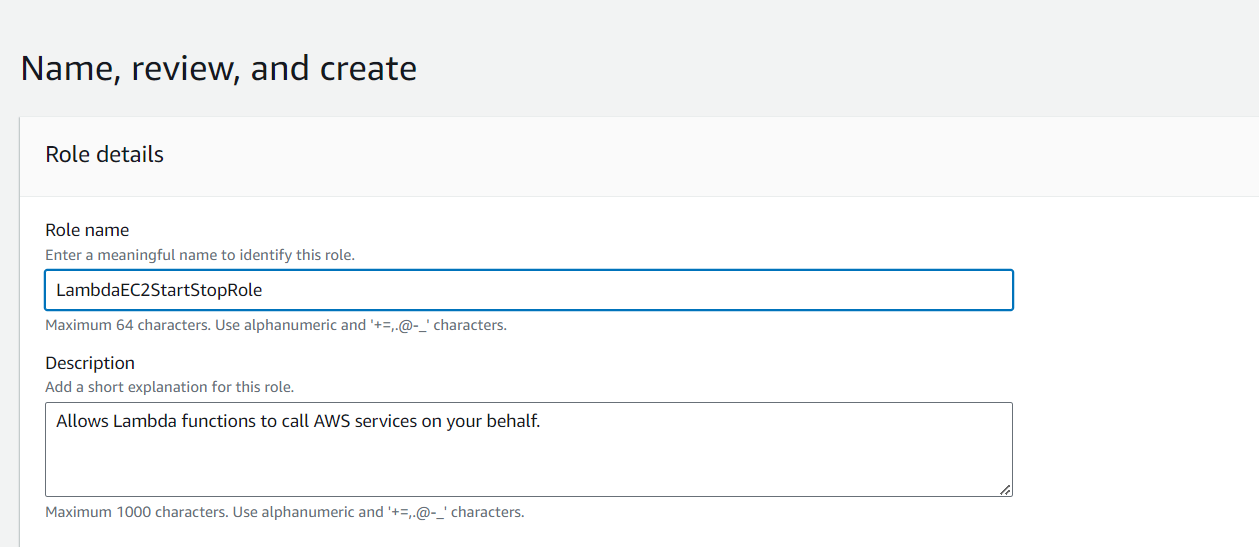


Click "Next: Permissions."

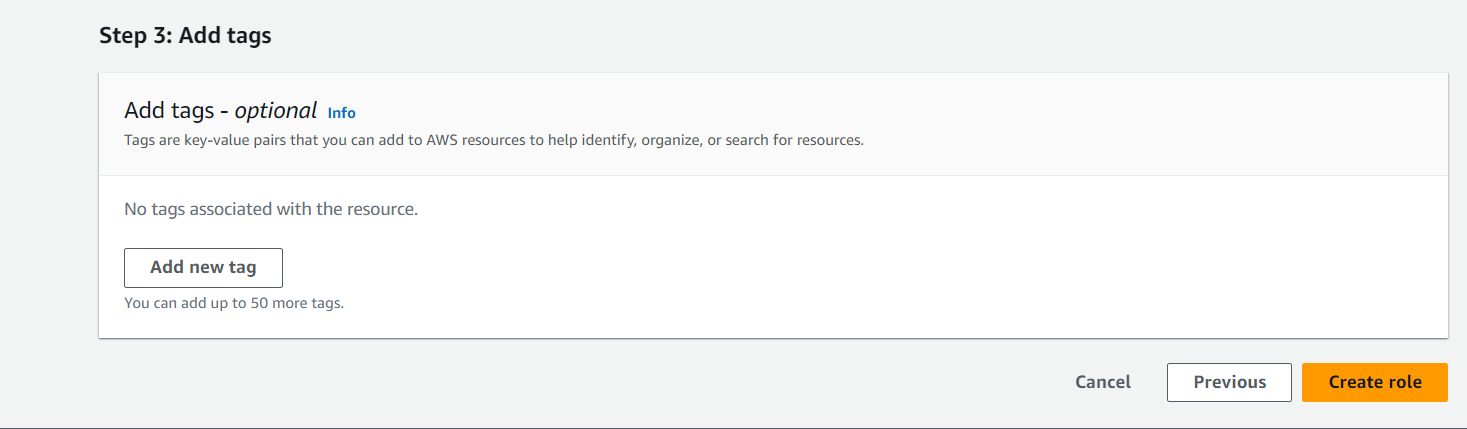
Search and select the "LambdaEc2Policy" policy. This policy provides Lambda with the necessary permissions.



Enter a meaningful name for your role, e.g., "LambdaEC2StartStopRole"

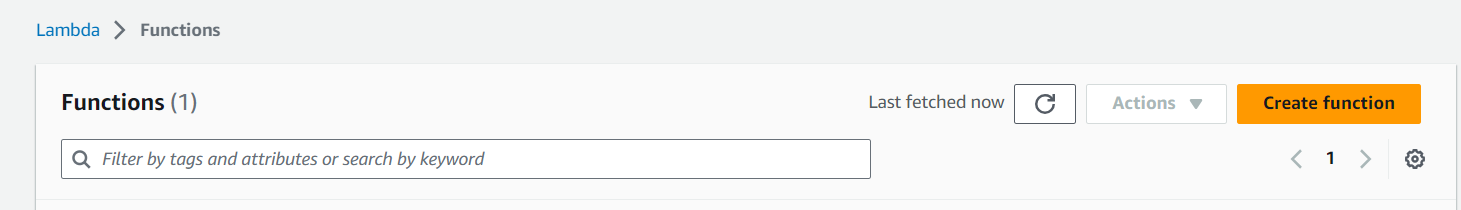


Add a description if needed, and click "Create role."

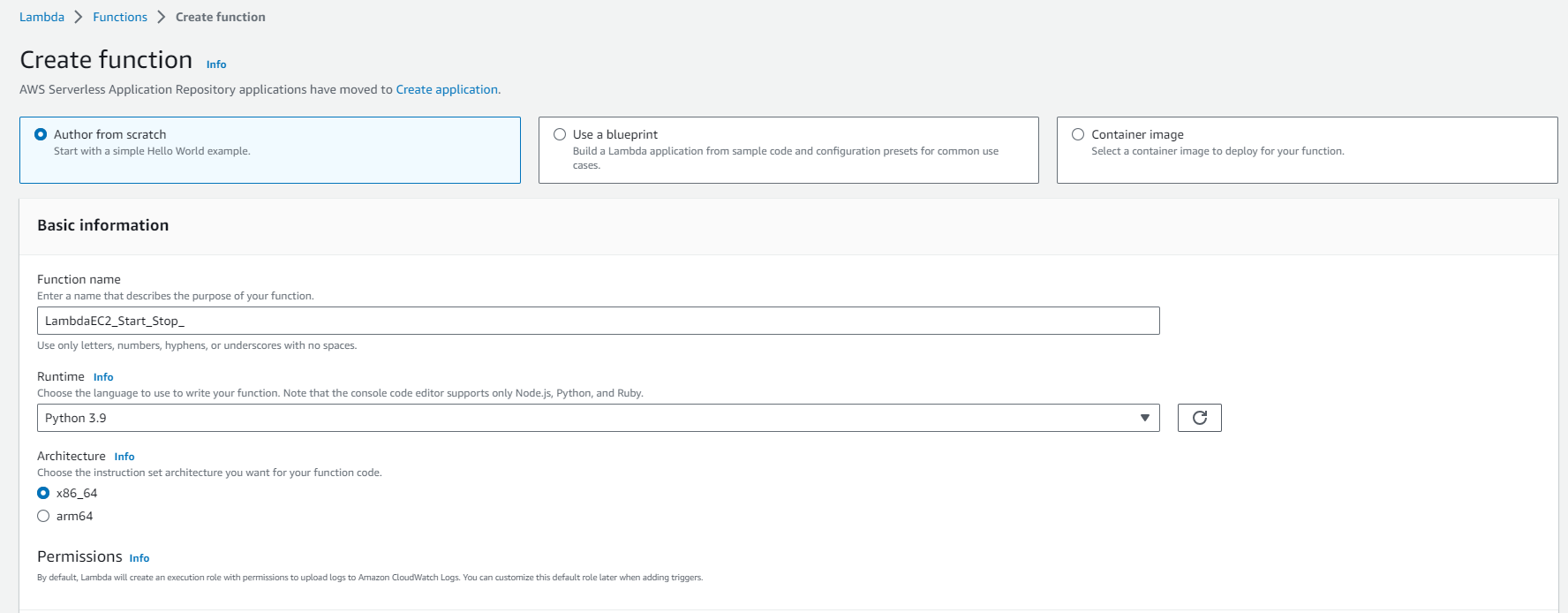


## Create Lambda Function

Go to the Lambda Console.



Click the "Create function" button.

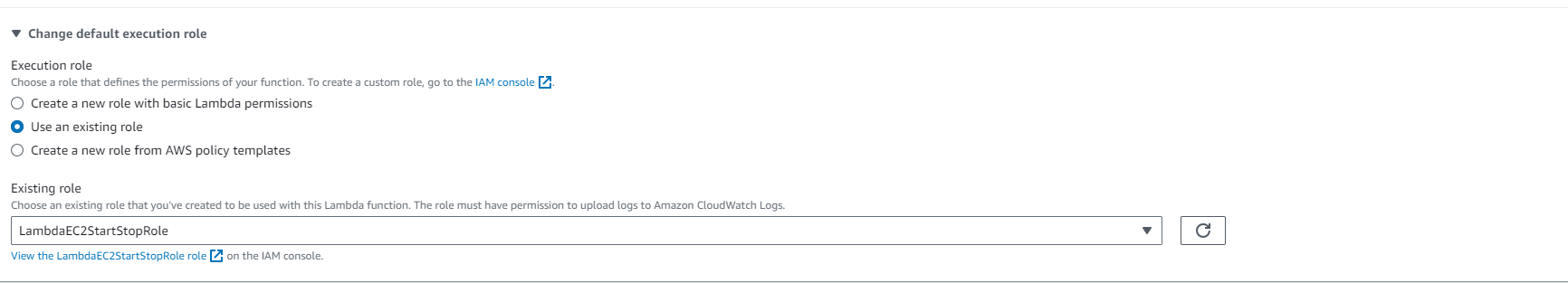


Choose "Author from scratch."

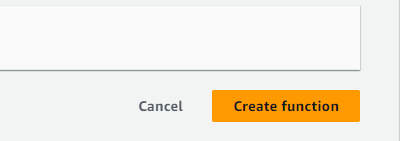
Fill in the following details:

Function name: Enter a name for your Lambda function.

Runtime: Choose a runtime (e.g., Python 3.9).

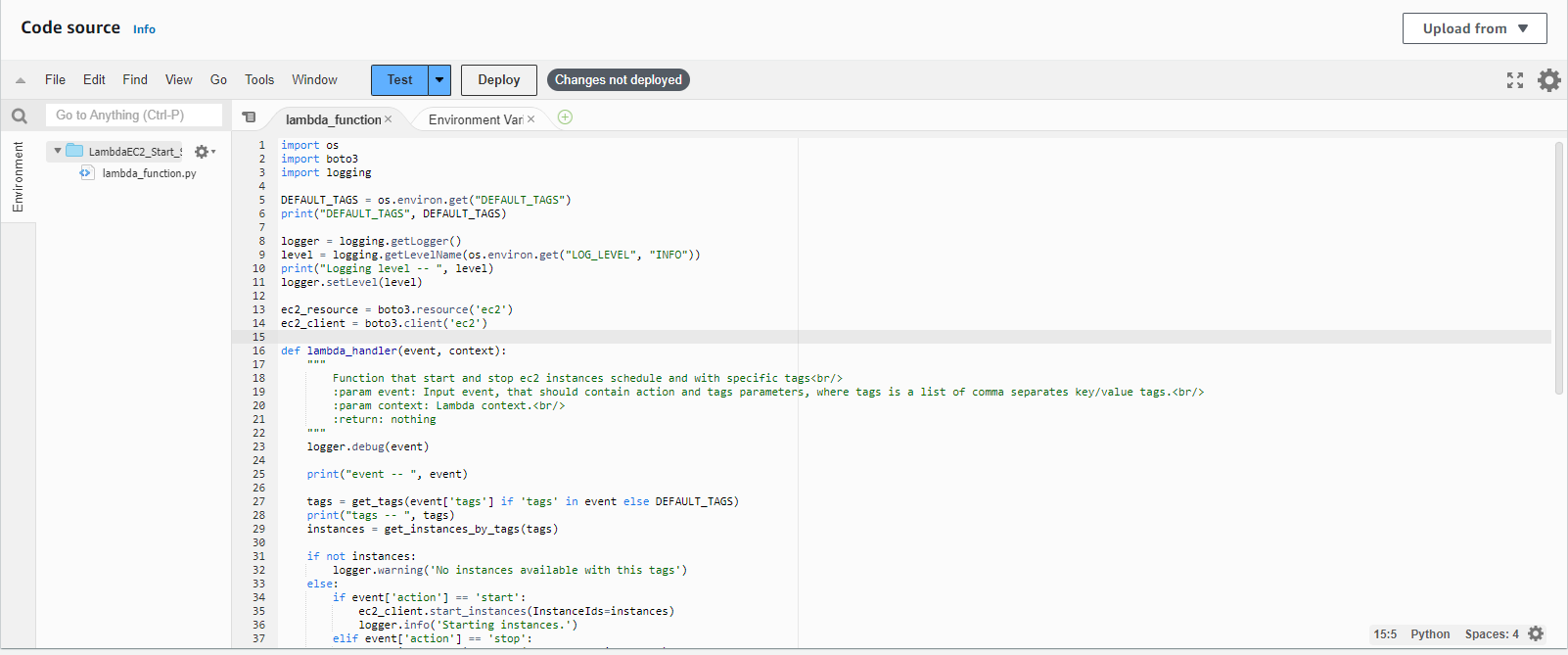


Role: Select the IAM role created in the previous step ("LambdaEC2StartStopRole").



Click "Create function."

In the function code section, paste the Lambda function code for starting and stopping EC2 instances. Add the custom script below code and click on deploy.



import os

import boto3

import logging

DEFAULT\_TAGS = os.environ.get("DEFAULT\_TAGS")

print("DEFAULT\_TAGS", DEFAULT\_TAGS)

logger = logging.getLogger()

level = logging.getLevelName(os.environ.get("LOG\_LEVEL", "INFO"))

print("Logging level -- ", level)

logger.setLevel(level)

ec2\_resource = boto3.resource('ec2')

ec2\_client = boto3.client('ec2')

def lambda\_handler(event, context):

"""

Function that start and stop ec2 instances schedule and with specific tags<br/>

:param event: Input event, that should contain action and tags parameters, where tags is a list of comma separates key/value tags.<br/>

:param context: Lambda context.<br/>

:return: nothing

"""

logger.debug(event)

print("event -- ", event)

tags = get\_tags(event['tags'] if 'tags' in event else DEFAULT\_TAGS)

print("tags -- ", tags)

instances = get\_instances\_by\_tags(tags)

if not instances:

logger.warning('No instances available with this tags')

else:

if event['action'] == 'start':

ec2\_client.start\_instances(InstanceIds=instances)

logger.info('Starting instances.')

elif event['action'] == 'stop':

ec2\_client.stop\_instances(InstanceIds=instances)

logger.info('Stopping instances.')

else:

logger.warning('No instances availables with this tags')

def get\_tags(tags):

"""

Method that split comma separated tags and return a formed tags filter<br/>

:param tags: Comma separated string with the tags values.<br/>

:return: tags structure

"""

final\_tags = []

split\_tags = tags.split(",")

for tag in split\_tags:

values = tag.split('=')

final\_tags.append({

'Name': values[0],

'Values': [values[1]]

})

return final\_tags

def get\_instances\_by\_tags(tags):

"""

Method that filter all ec2 instances and return only the instances with specific tags<br/>

:param tags: Filter structure with tag values.<br/>

:return: list of ec2 instances

"""

response = ec2\_resource.instances.filter(Filters=tags)

print("Response -- ", response)

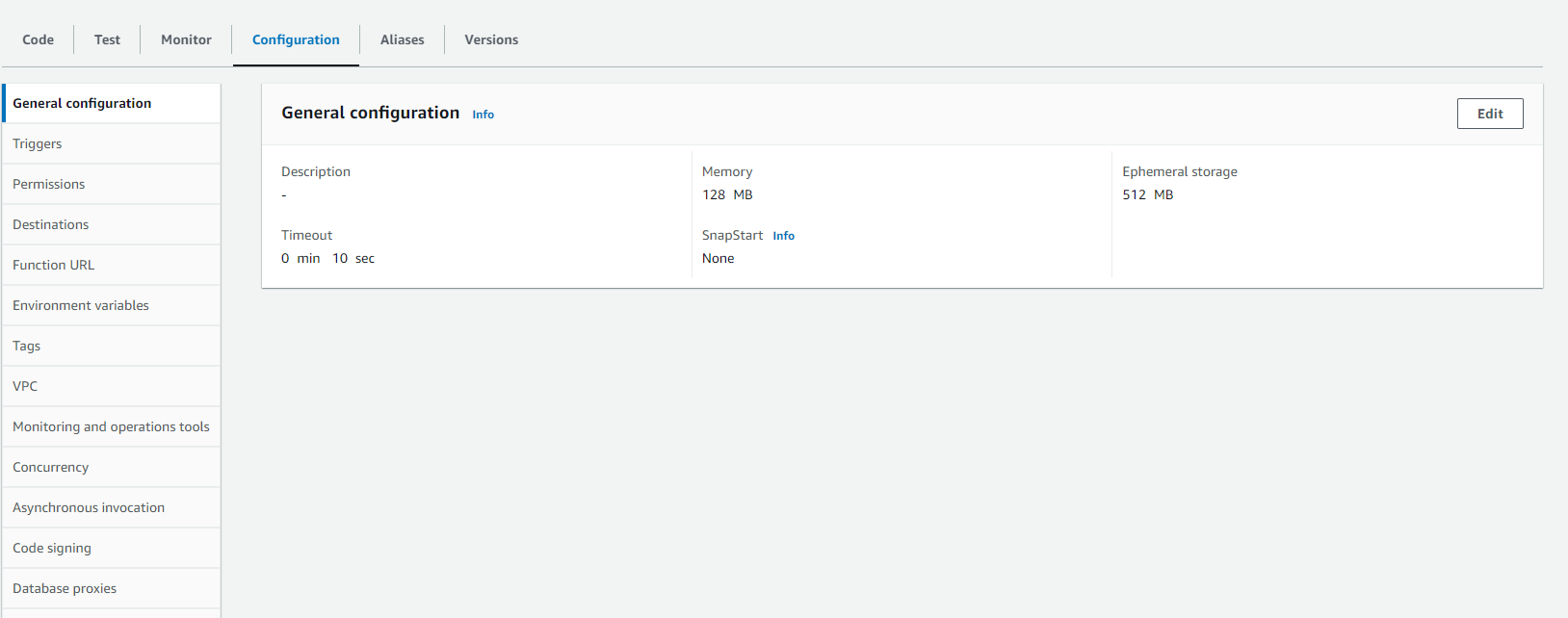
for instance in response:

print("Instance -- ", instance)

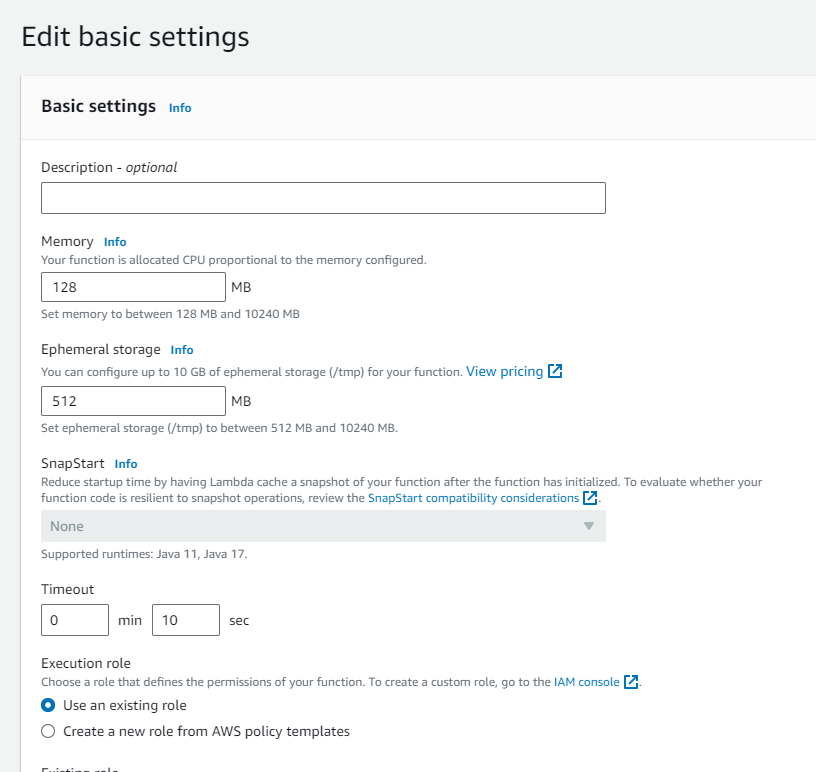
intance\_ids = [instance.id for instance in response]

print("intance\_ids -- ", intance\_ids)

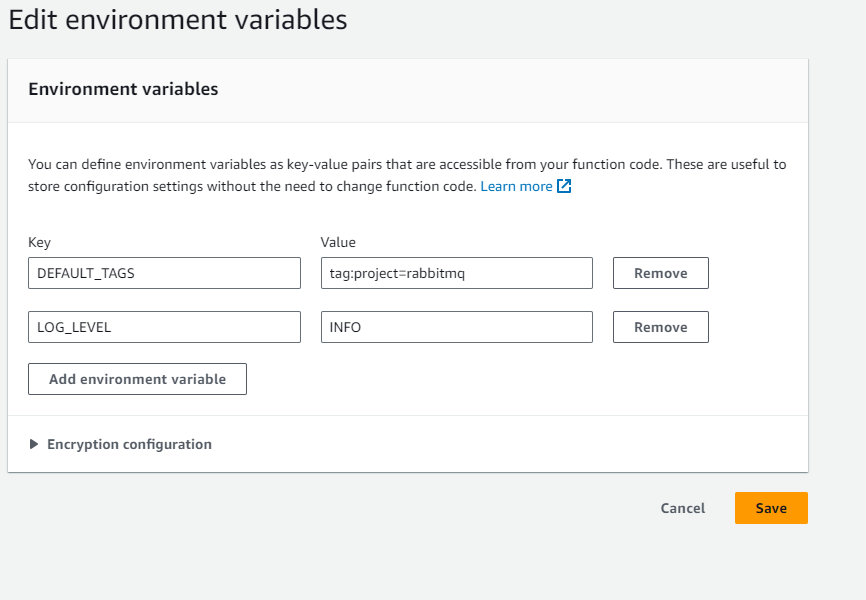
return intance\_ids



In the configuration Tab, edit the general configuration



Change timeout 3 sec to 10 sec.

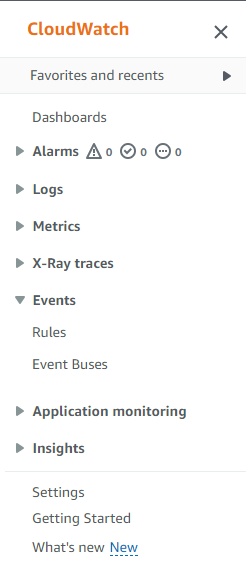


Configure environment variables and save.

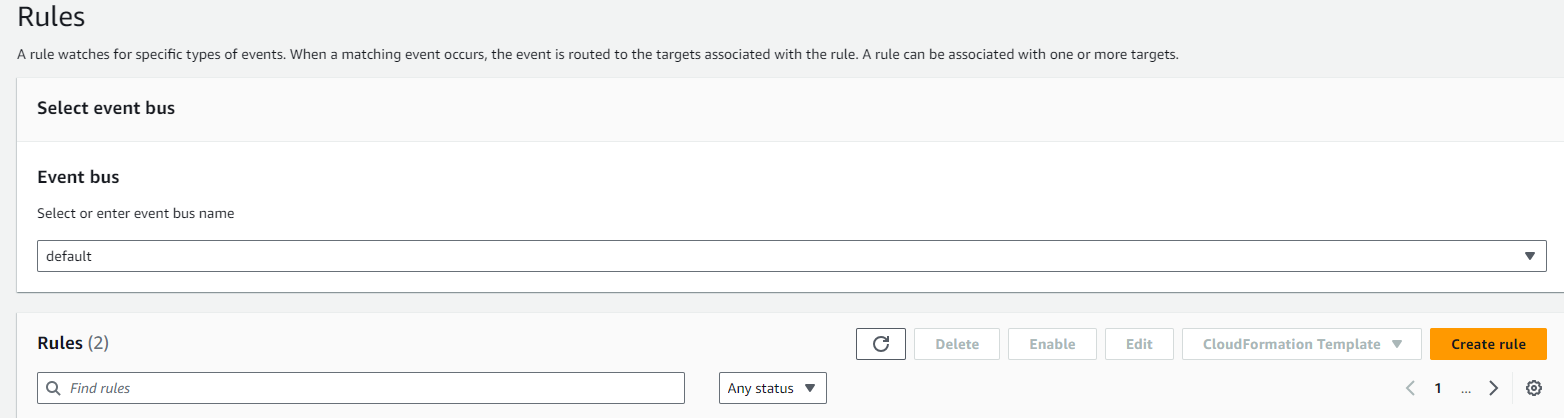
\* environment variables depend on Tags that are already added in the instances. Lambda script totally works on tags that why we added this.

## Set Up CloudWatch Events

Go to the CloudWatch Console.

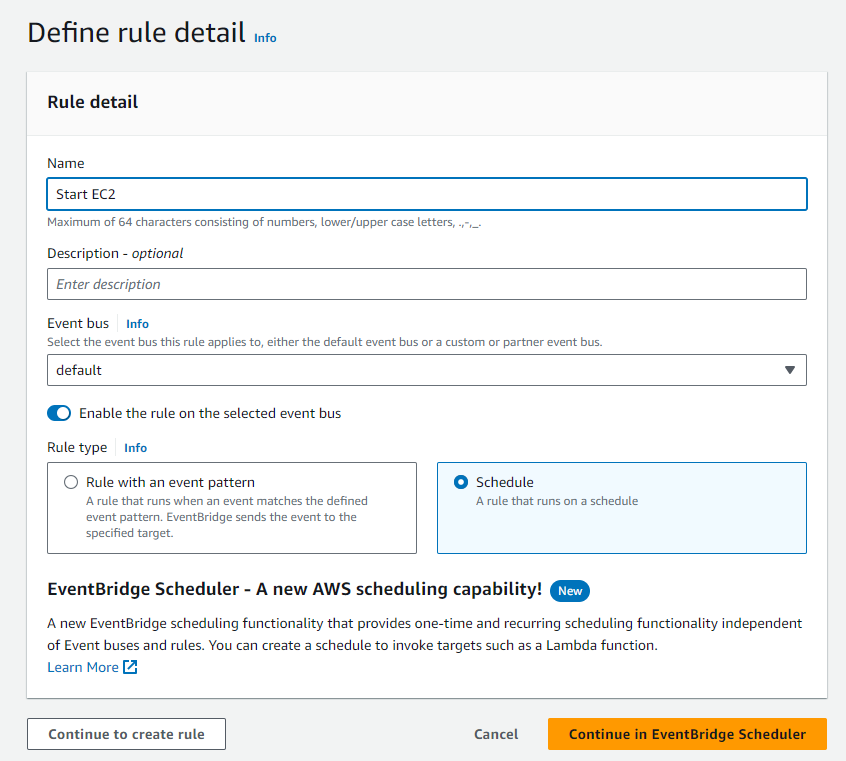


In the left sidebar, under "Events," click on "Rules."

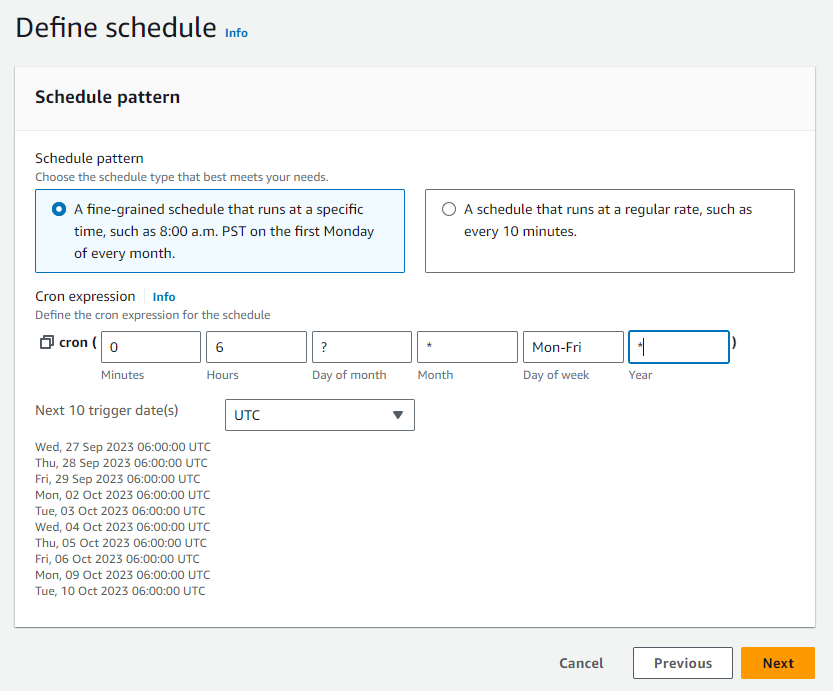


Click "Create rule."

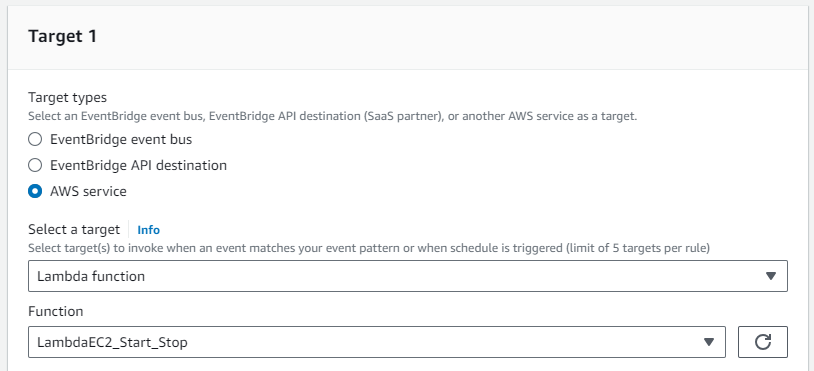
### Start EC2 Instance Rule:



Add Name Under "Rule Type" choose " Schedule." Then click to create rule



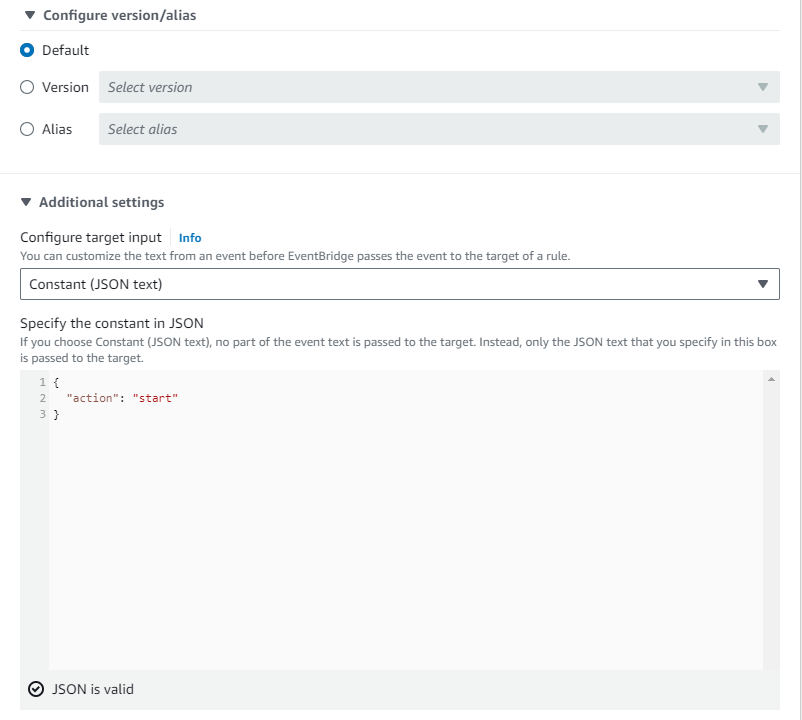
Configure the schedule for when you want to start your EC2 instance at 06 am UTC (11am Pakistan Time) Monday to Friday using a cron expression.



Under "Targets," click "Add target" and select "Lambda function."

Choose the Lambda function created earlier.

Click "Configure details."



{

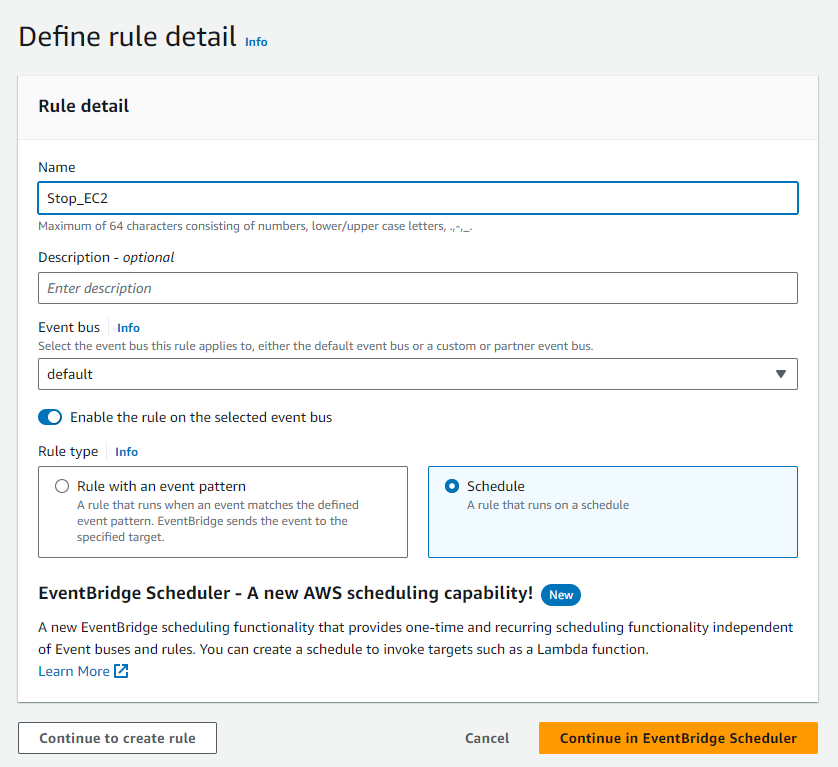
"action": "start"

}

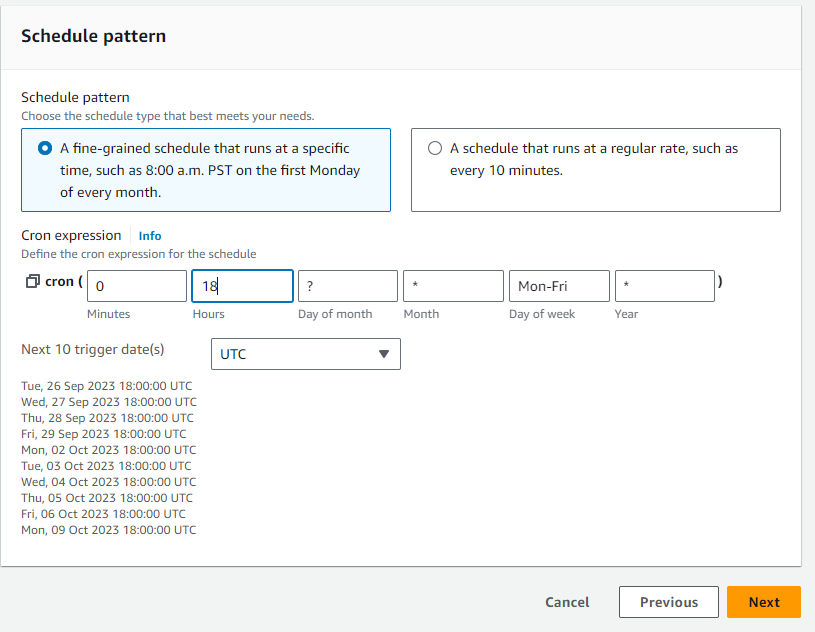
Click on **Next**, and Optionally, configure tags for the rule.

Click "**Create rule.**"

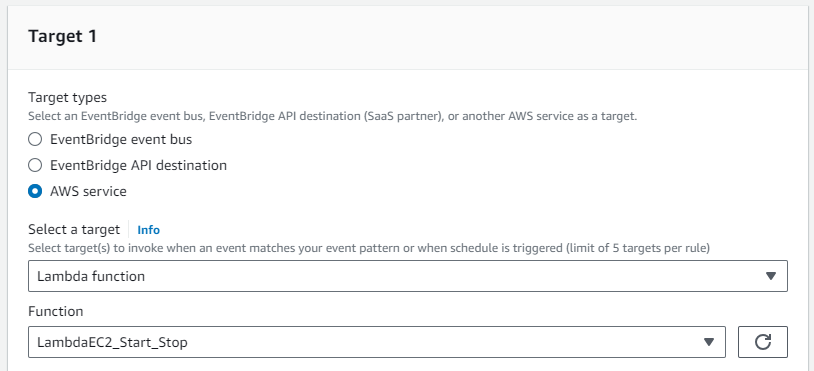
### Stop EC2 Instance Rule:



Add Name Under "Rule Type" choose " Schedule." Then click to create rule



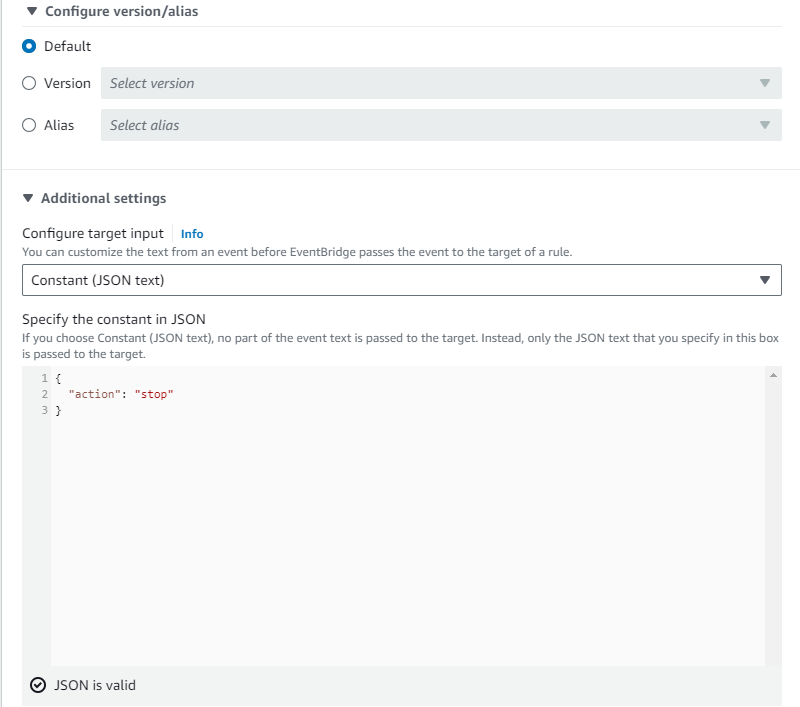
Configure the schedule for when you want to stop your EC2 instance at 18 pm UTC (11pm Pakistan Time) Monday to Friday using a cron expression.



Under "Targets," click "Add target" and select "Lambda function."

Choose the Lambda function created earlier.

Click "Configure details."



{

"action": "start"

}

Optionally, configure tags for the rule.

Click "Create rule."

## Conclusion

Automating the start and stop of EC2 instances using AWS Lambda and CloudWatch Events is an effective way to reduce costs and improve resource management in your AWS environment. By following this guide, you can schedule your instances to be active only when needed, ensuring optimal resource utilization.