

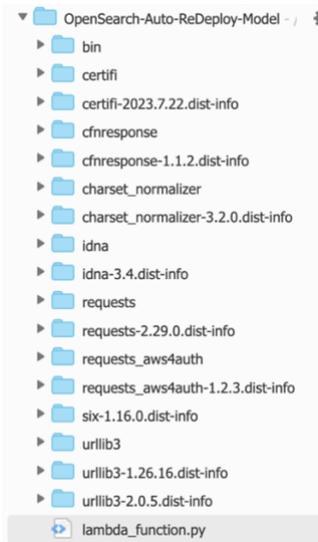
## README.pdf

### 1. Purpose:

This is a lambda helpful function to help the customers to conduct auto-deploy model when the models are undeployed in a node, for example, when adding a new node and the model is not deployed to the new node yet. This helpful lambda function can be added with a trigger to run auto deployment in a schedule.

### 2. About the zip file:

In the zip file, please note that the lambda\_function.py is the main file to run in the lambda job, the other folders are imported packages. Those are dependencies for the lambda\_function.py to run successfully.



### 3. Set-up Steps by Steps:

3.1 Create [IAM role](#) to give lambda access to OpenSearch

3.1.1 Use the following Custom trust policy to create an AWS IAM Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

### Trusted entity type

**AWS service**  
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

**AWS account**  
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

**Web identity**  
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

**SAML 2.0 federation**  
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

**Custom trust policy**  
Create a custom trust policy to enable others to perform actions in this account.

### Custom trust policy

Create a custom trust policy to enable others to perform actions in this account.

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Principal": {
7         "Service": "Lambda.amazonaws.com"
8       },
9       "Action": "sts:AssumeRole"
10    }
11  ]
12 }
```

**Edit statement**

Select a statement  
Select an existing statement in the policy or add a new statement.

[+ Add new statement](#)

[+ Add new statement](#)

## 3.1.2 Add AmazonOpenSearchServiceFullAccess

### Add permissions Info

#### Permissions policies (1/923) Info

Choose one or more policies to attach to your new role.

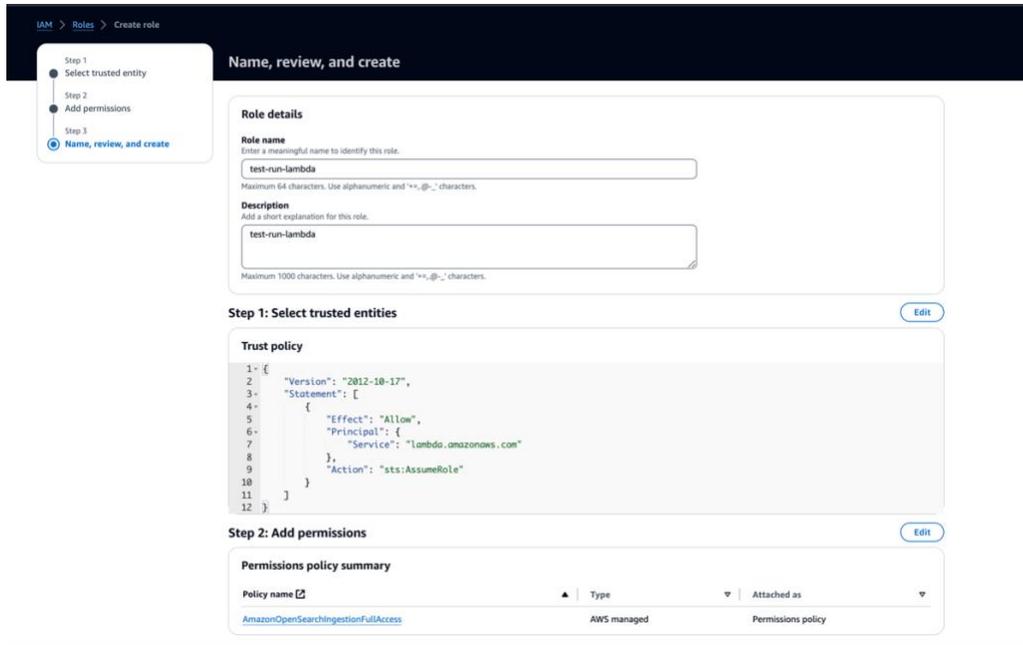
Filter by Type

opensear  All types 6 matches

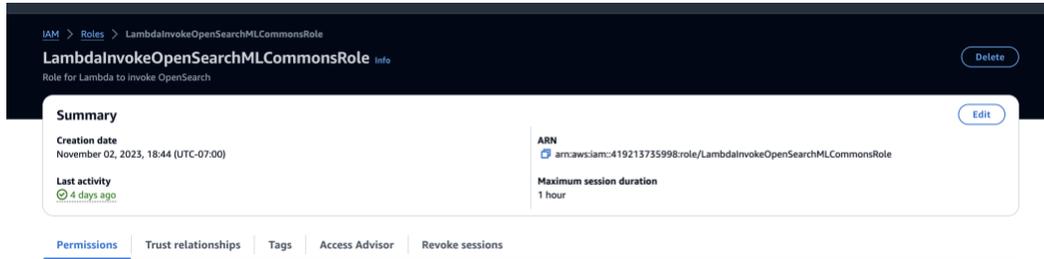
<input type="checkbox"/>	Policy name	Type	Description
<input checked="" type="checkbox"/>	<a href="#">AmazonOpenSearchIngestionFullAccess</a>	AWS managed	Allows Amazon OpenSearch Ingestion ...
<input type="checkbox"/>	<a href="#">AmazonOpenSearchIngestionReadOnlyA...</a>	AWS managed	Provides read only access to the Amaz...
<input type="checkbox"/>	<a href="#">AmazonOpenSearchServiceCognitoAccess</a>	AWS managed	Provides access to the Amazon Cognit...
<input type="checkbox"/>	<a href="#">AmazonOpenSearchServiceFullAccess</a>	AWS managed	Provides full access to the Amazon Op...
<input type="checkbox"/>	<a href="#">AmazonOpenSearchServiceReadOnlyAccess</a>	AWS managed	Provides read-only access to the Amaz...
<input type="checkbox"/>	<a href="#">AWSQuicksightOpenSearchPolicy</a>	AWS managed	Provides access to Amazon OpenSearc...

► Set permissions boundary - optional

Cancel [Previous](#) [Next](#)

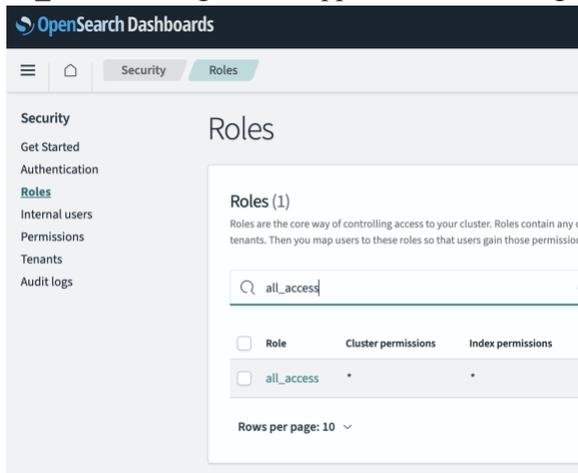


3.1.3 After creating the new IAM role, please save the role ARN for later config.

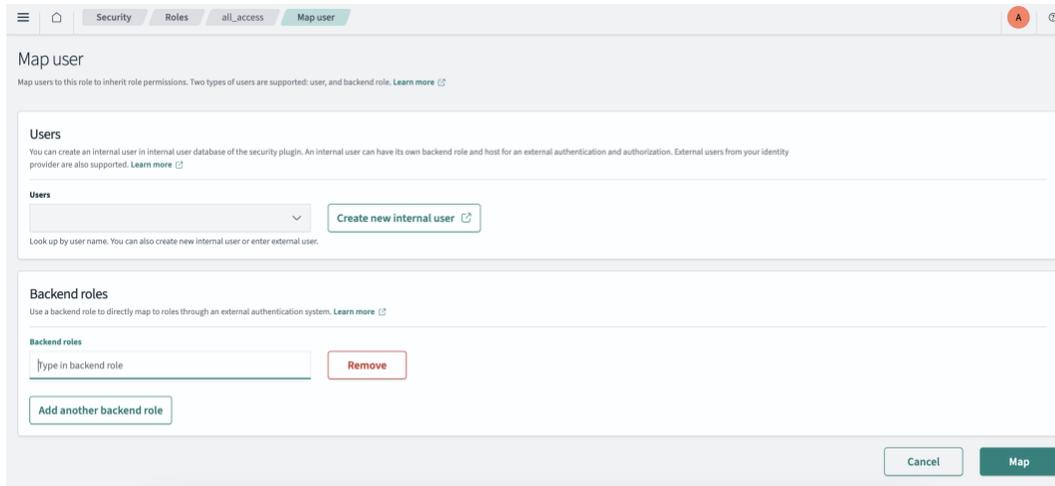


3.1.4 Map role to backend role with all\_access

Navigate to the OpenSearch Dashboard -> Security -> Roles, find all\_access role, click on all\_access. Navigate to Mapped users -> Managed Mappings

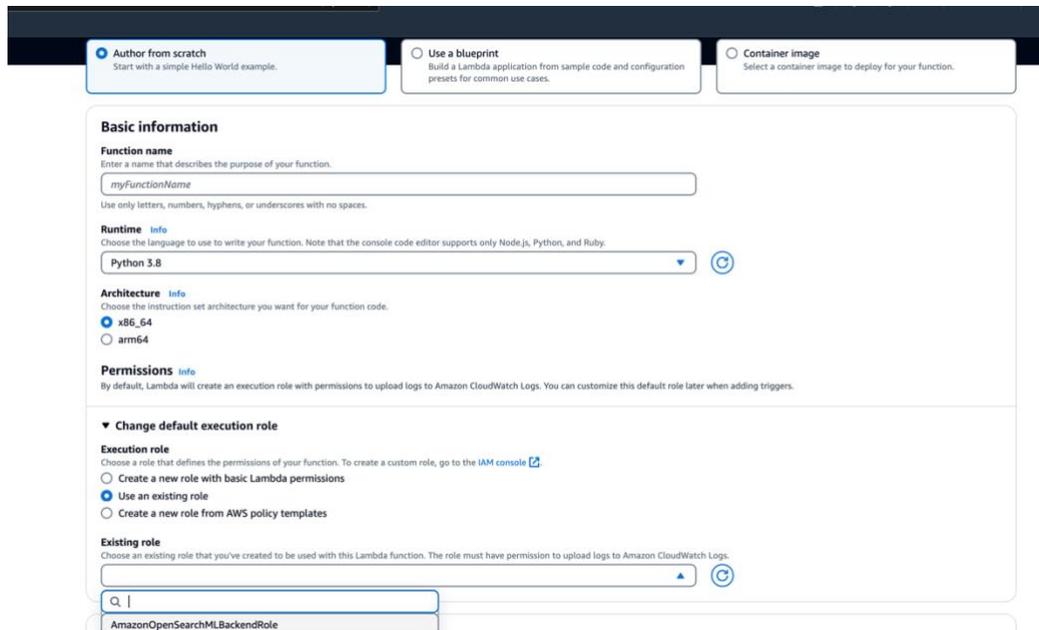


Map the admin role with the new IAM role created in 3.1.2 step.



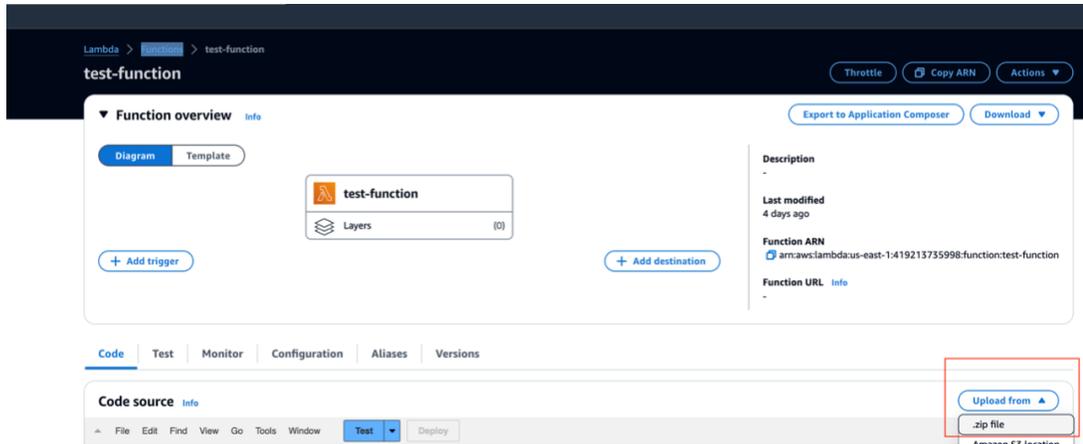
### 3.2 Create a new AWS [lambda](#) function:

In 'create function' config, choose RunTime as Python 3.8 and choose use existing role, click on the role name that you created previous in 3.1.2, leave the rest of the default setting, then click "Create Function"



### 3.3 Upload the zipfile

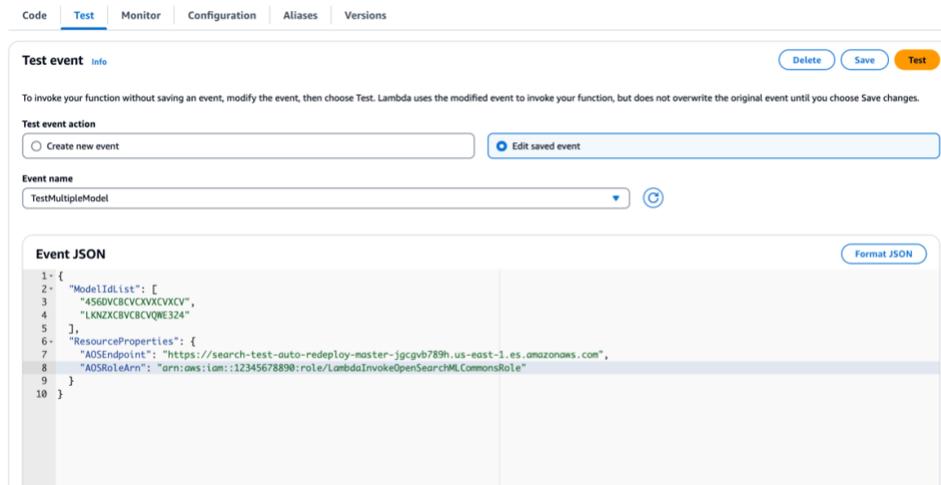
On the new function page, click 'Upload from' in the Code Tap, choose the provided zip file.



### 3.4 Testing

Now you can see that the `lambda_function.py` in the Code source window. Click on Test Tab. Please put the `model_id` into `ModelIdList` that you would like to conduct auto model deployment, input the AOS endpoint which you can find out from AOS domain config and input the lambda role arn that you created in previous 3.1.2. Click “Test” to run auto model deployment.

```
{
  "ModelIdList": [
    "<model_id1>",
    "<model_id2>"
  ],
  "ResourceProperties": {
    "AOSEndpoint": "<AOSEndPoint>",
    "AOSRoleArn": "<RoleARN>"
  }
}
```

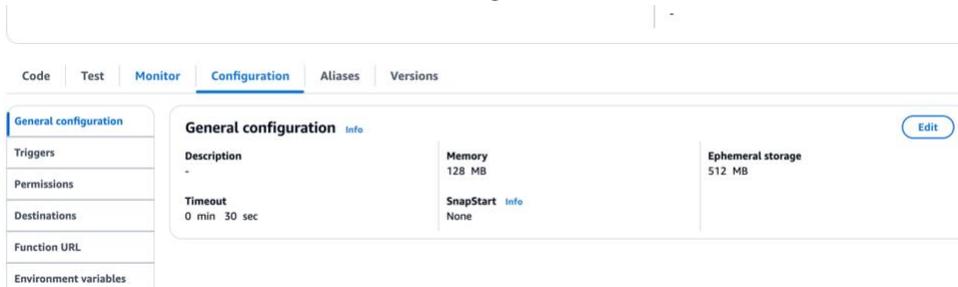


### 3.4.1 Test Success

Please make sure the test success before adding trigger. The sample success outcome is similar to this.



3.4.2 If lambda timeout, set Timeout to longer timeframe, maximum can be 15 minutes.

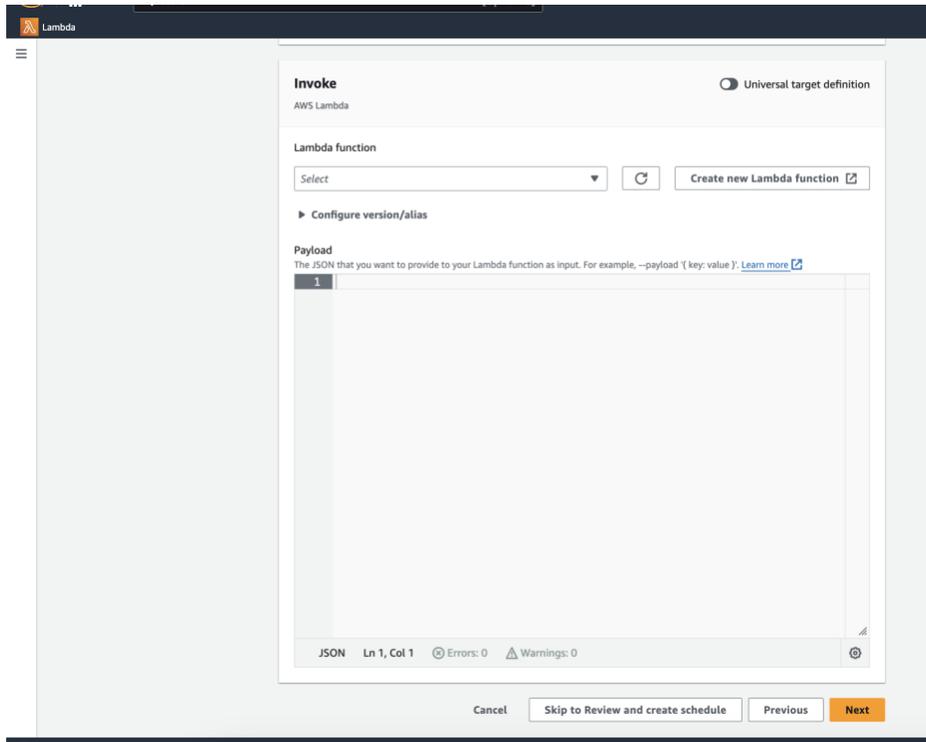


### 3.5 Add EventBridge rule to schedule the lambda job

Open the Amazon EventBridge console. In the navigation pane, choose Rules. Choose Create rule. For Event bus, choose default event bus. For Rule type, choose Schedule. Choose Next. For Schedule pattern, choose a schedule that runs at a regular rate, such as every 10 minutes (0/10 \* ? \* MON-FRI \*). Please refer to the [Cron expressions reference](#) to config different schedules. Choose Next. For Target types, choose AWS service. For Select a target, choose Lambda function from the drop-down list, choose the same lambda job that you created in step 3.2.

In payload, input the same event that you tested in 3.4.1, this payload will pass to lambda job in every scheduled trigger. Review the details of the rule and choose Create rule.

```
{
  "ModelIdList": [
    "<model_id1>"
  ],
  "ResourceProperties": {
    "AOSEndpoint": "<AOSEndPoint>",
    "AOSRoleArn": "<RoleARN>"
  }
}
```



Now, the auto deployment lambda job is detecting undeployed models from your provided model list and conduct auto-deployment in a schedule.