Cloudera Data Engineering 1.25.0

Using Cloudera Data Engineering resources

Date published: 2020-07-30
Date modified: 2025-11-20

https://docs.cloudera.com/

Legal Notice

Contents

Overview of using Cloudera Data Engineering resources

Key resource management strategies in Cloudera Data Engineering include leveraging Python virtual environments for managing job dependencies and utilizing custom Spark runtime Docker images to integrate specialized packages and libraries into Spark jobs.

Using Python virtual environments

Cloudera Data Engineering supports Python virtual environments to manage job dependencies by using the python-env resource type.

For more information, see Using Python virtual environments.

Using custom Spark runtime Docker images via API/CLI

Custom Spark runtime Docker images are used when custom packages and libraries need to be installed and used when executing Spark jobs. These custom packages and libraries can be proprietary software packages like RPMs that need to be compiled to generate the required binaries. Docker images allow you to pre-bake these dependencies into a self-contained Docker file that can be used across multiple Spark jobs.

For more information, see Using custom Spark runtime Docker images via API/CLI.

Using Python virtual environments with Cloudera Data Engineering

Cloudera Data Engineering supports Python virtual environments to manage job dependencies by using the python-env *resource* type.

A *resource* in Cloudera Data Engineering is a named collection of files or other resources referenced by a job. The python-env resource type allows you to specify a requirements.txt file that defines a virtual environment that you can then associate with a Cloudera Data Engineering job. You can specify any Python package in the requirements.txt file, including those with C dependencies.

Creating a Python virtual environment resource

After you have created the requirements.txt file, you can create the Python virtual environment resource.

Note:

- For python-env resources, you can only upload a requirements.txt file. Python environment resources do not support arbitrary file upload. If the local file is named something other than requirements.txt, you must add the flag --resource-path requirements.txt to the command, which renames the file to requirements.txt in the resource.

  You can also specify a PyPi mirror for a Python virtual environment resource using the --pypi-mirror flag. This requires network access to the mirror from the Cloudera environment.
- If a Python package specified in the requirements.txt file is not found or does not support the underlying OS architecture amd64/arm64, then the Python virtual environment build fails.

For CDE CLI

Before you begin

- Download and configure the CDE CLI.

- Create a requirements.txt file specifying the Python package and version dependencies required by your Cloudera Data Engineering job.
- Ensure that the following hostnames are reacheable from within the cluster, to install the Python package successfully if no PyPi mirror is configured:
    - pypi.python.org
    - pypi.org
    - pythonhosted.org
    - files.pythonhosted.org

Steps

1. Run the cde resource create command as follows to create a Python virtual environment resource.

    Note: You can also specify a PyPi mirror for a Python virtual environment resource using the --pypi-mirror flag. This requires network access to the mirror from the CDP environment.

    ```
    cde resource create --name cde-python-env-resource --type python-env --p
    ython-version python3
    ```

2. Upload the requirements.txt file to the resource.

    Note: For python-env resources, you can only upload a requirements.txt file. Python environment resources do not support arbitrary file upload. If the local file is named something other than requirem ents.txt, you must add the flag --resource-path requirements.txt to the command, which renames the file to requirements.txt in the resource.

    ```
    cde resource upload --name cde-python-env-resource --local-path ${HOME}/
    requirements.txt
    ```

Result

When you first create a Python virtual environment resource, Cloudera Data Engineering builds the environment according to the requirements.txt file. During this build time, you cannot run a job associated with the virtual environment. You can check the status of the environment by running cde resource list-events --name *<RESOURCE_NAME>*. For example:

```
cde resource list-events --name cde-python-env-resource
```

The environment is ready when you see a message similar to the following:

```
    {
      "id": 4,
      "message": "Job pp-84kgdgf6-resource-builder-cde-python-env-resourc
e-1634911572 succeeded, marking resource with ready status",
      "created": "2021-10-22T14:09:13Z"
    }
```

For Web UI

Before you begin

Important: The user interface for Cloudera Data Engineering 1.17 and above has been updated. The left-hand menu was updated to provide easy access to commonly used pages. The steps below will vary slightly, for example, the Overview page has been replaced with the Home page. The new home page still displays Virtual Clusters, but now includes quick-access links located at the top for the following categories: Jobs, Resources, and Download & Docs.

- Create a requirements.txt file specifying the Python package and version dependencies required by your Cloudera Data Engineering job.

- Ensure that the following hostnames are reacheable from within the cluster, to install the Python package successfully if no PyPi mirror is configured:
  - pypi.python.org
  - pypi.org
  - pythonhosted.org
  - files.pythonhosted.org

Steps

1. In the Cloudera management console, click the Data Engineering tile and click Overview.
2. In the CDE Services column, select the service containing the virtual cluster where you want to create the Python virtual environment.
3. In the Virtual Clusters column on the right, click the View Jobs icon for the virtual cluster where you want to create the Python virtual environment.
4. Click Resources in the left menu.
5. Click Create Resource at the top right.
6. Specify a resource name, and then select Python Environment from the Type drop-down menu.
7. Choose the Python version for the environment and optionally specify the PyPi Mirror URL. The PyPi mirror must be accessible from the Cloudera environment.
8. Click Create.
9. Click Upload File and select the requirements.txt file from your local machine. You can also drag-and-drop the file to the outlined area on the page.

Result

The UI displays Building the resource... while the Python virtual environment is building. After the environment is built, the page displays the Python packages and versions included in the environment.

Associating a Python virtual environment with a Cloudera Data Engineering job

You can associate the Python virtual environment with a Cloudera Data Engineering job at the time of creation, or you can update an existing job.

For CDE CLI

Before you begin

- Download and configure the CDE CLI.
- Create a Python virtual environment Cloudera Data Engineering resource.
- Create a Cloudera Data Engineering job.

Steps

1. Using the CDE CLI, run the cde job update command to associate a Python virtual environment with the job.

```
cde job update --name pyspark-example --python-env-resource-name cde-pyt
hon-env-resource
```

Note: You can specify a Python virtual environment resource at job creation time as well, using the flag --python-env-resource-name. For example:

```
cde job create --type spark --application-file pyspark-example.p
y --python-env-resource-name cde-python-env-resource --name pysp
ark-example
```

For Web UI

Before you begin

Important: The user interface for Cloudera Data Engineering 1.17 and above has been updated. The left-hand menu was updated to provide easy access to commonly used pages. The steps below will vary slightly, for example, the Overview page has been replaced with the Home page. You can also manage a job by clicking Jobs on the left-hand menu, then selecting your desired Virtual Cluster from a drop-down at the top of the Jobs page. The new home page still displays Virtual Clusters, but now includes quick-access links located at the top for the following categories: Jobs, Resources, and Download & Docs.

- Create a Python virtual environment Cloudera Data Engineering resource.
- Create a Cloudera Data Engineering job.

Steps

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Job Runs on the left navigation menu. The Job Runs page displays.
3. Using the dropdown menu, select the virtual cluster containing the application you want to manage.
4. Click the name of the job you want to modify.
5. Go to the Configuration tab.
6. Click Edit.
7. In the Python Environment section, click Select Python Environment.
8. Select the Python virtual environment resource you want to use, and then click Select Resource.
9. At the bottom of the page, click Update and Run to run the job immediately, or click the drop-down arrow on the button and select Update to update the job without running it.

Associating a Python virtual environment with a Cloudera Data Engineering Session

You can associate the Python virtual environment with a Cloudera Data Engineering Session at the time of creation.

For CDE CLI

Before you begin

- Download and configure the CDE CLI.
- Create a Python virtual environment Cloudera Data Engineering resource.

Steps

1. Using the CDE CLI, run the cde session create command to associate a Python virtual environment with the session.

```
cde session create --name python-env-example  --python-env-resource-name
 cde-python-env-resource
```

For Web UI

Before you begin

- Create a Python virtual environment Cloudera Data Engineering resource.
- Create a Cloudera Data Engineering Session.

Steps

1. In the Cloudera console, click the Data Engineering tile. The Cloudera Data Engineering Home page displays.
2. Click Sessions on the left navigation menu.
3. Using the dropdown menu, select the virtual cluster containing the application you want to manage.

4. Click Create.

   Note: The Python environment can only be specified when you create a Session.

5. In the Python Environment section, click Select Python Environment.
6. Select the Python virtual environment resource that you want to use, and then click Select Resource.
7. Set additional configurations as needed.
8. Click Create.

Updating Python virtual environment resources

Currently, Python virtual environments cannot be updated. Instead, create a new Python virtual environment resource and update the job to reference the new resource.

For CDE CLI

Before you begin

- Download and configure the CDE CLI.
- Create a new requirements.txt file specifying the Python package and version dependencies required by your Cloudera Data Engineering job.

Steps

1. Create a new python-env resource.

```
cde resource create --name new-cde-python-env-resource --type python-env
 --python-version python3
```

2. Upload the new or updated requirements.txt file to the new resource.

```
cde resource upload --name new-cde-python-env-resource --local-path ${HO
ME}/requirements.txt
```

3. Update the Cloudera Data Engineering job to specify the new resource.

```
cde job update --name pyspark-example --python-env-resource-name new-cde
-python-env-resource
```

For Web UI

Before you begin

- Create a new requirements.txt file specifying the Python package and version dependencies required by your Cloudera Data Engineering job.

Steps

1. Create a new Python virtual environment using the new requirements.txt file, following the instructions in Creating a Python virtual environment resource .
2. Update the Cloudera Data Engineering job to reference the new Python virtual environment, following the instructions in Associating a Python virtual environment with a Cloudera Data Engineering job .

Related Information

Creating a Python virtual environment resource

Associating a Python virtual environment with a Cloudera Data Engineering job

Using Custom Spark Runtime Docker Images via API/CLI

This is a detailed user guide that demonstrates how to run Spark jobs using custom Spark runtime Docker images via API/CLI.

About this task

Custom Spark runtime Docker images are used when custom packages and libraries need to be installed and used when executing Spark jobs. These custom packages and libraries can be proprietary software packages like RPMs that need to be compiled to generate the required binaries. Docker images allow you to pre-bake these dependencies into a self-contained Docker file that can be used across multiple Spark jobs.

Important: If you are using Cloudera Data Engineering 1.18 and above, you cannot use Custom Spark Runtime Docker Images with Cloudera Data Engineering 1.17 and below.

Note: You need to recompile the custom runtime images for each Cloudera Data Engineering version.

Before you begin

Entitlement for Customer Docker images

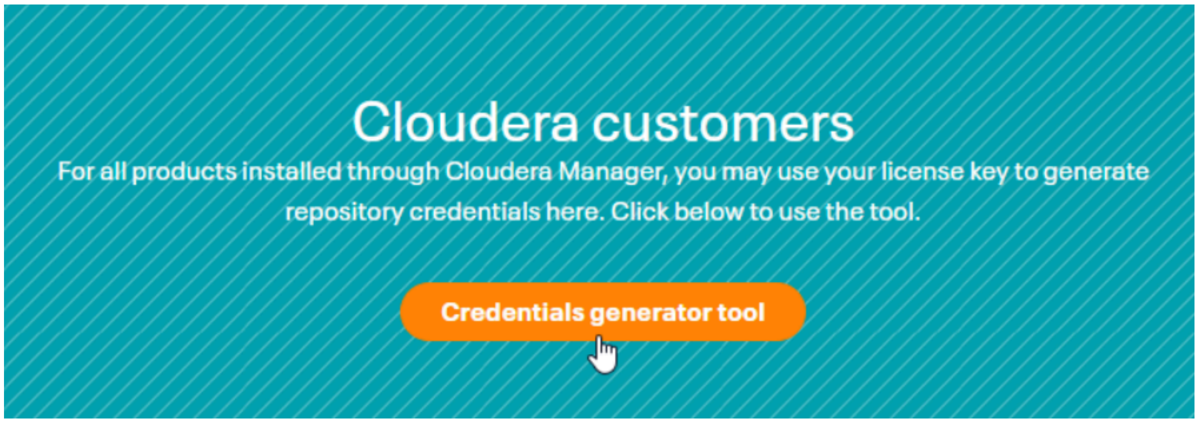
To use custom Spark runtime Docker images, your Cloudera tenant must have the DE_CUSTOM_RUNTIME entitlement enabled. If not yet in place, you can request it via your Cloudera Account Team (primarily your Solution Engineer). They will fulfill the request internally and confirm to you when the entitlement has been applied. Please allow 24 hours for fulfillment.

Note: The DE_CUSTOM_RUNTIME entitlement must be enabled before a Virtual Cluster is created to be able to create custom runtime resources within that cluster.

Docker repository credentials

To pull the base Docker image, you must have credentials and authenticate your Docker client to docker.repository.cloudera.com. To get credentials:

1. Raise an “Admin” type case with Cloudera Support and request a License Key.
2. Once the License Key is received, navigate to https://www.cloudera.com/downloads.html and log in with your MyCloudera credentials.
3. Use the Credentials Generator tool - this is about half-way down the page (you will see an orange Sign In button if not already logged in).


Cloudera customers
For all products installed through Cloudera Manager, you may use your license key to generate repository credentials here. Click below to use the tool.
Credentials generator tool

As directed on the page, copy and paste the entire contents of your license file into the text box and click the Get Credentials button to generate your username and password.

Procedure

1. Create a custom Docker image.

   Build the custom-spark-dex-runtime image based on the dex-spark-runtime image of the Cloudera Data Engineering version.

   The image should be based on the dex-spark-runtime of the current dex version.

   The relevant dex-spark-runtime images are as follows.

   - Spark 3 Cloudera security hardened images

     <registry-host>/cloudera/dex/dex-spark-runtime-<spark version>-<cdh version>:<CDE version>

     Example: DockerFile for DEX 1.24.0-b711, Spark 3.3.2 and Cloudera Runtime version 7.1.9.1015

     ```
     FROM
     docker.repository.cloudera.com/cloudera/dex/dex-spark-runtime-3.3.2-7.1
     .9
     .1015:1.24.0-b711
     USER root
     RUN apk add --no-cache git
     RUN pip3 install virtualenv-api
     USER ${DEX_UID}
     ```

   - Spark 3 Redhat (insecure and deprecated) images

     <registry-host>/cloudera/dex/dex-spark-runtime-<spark version>-<cdh version>-compat:<CDE ver sion>

     Example: DockerFile for DEX 1.24.0-b711, Spark 3.3.2 and Cloudera Runtime version 7.1.9.1015

     ```
     FROM docker.repository.cloudera.com/cloudera/dex/dex-spark-runtime-3.3.2
     -7.1.9.1015-compat:1.24.0-b711
     USER root
     RUN yum install -y git && yum clean all && rm -rf /var/cache/yum
     RUN pip2 install virtualenv-api
     RUN pip3 install virtualenv-api
     USER ${DEX_UID}
     ```

   - Spark 2 Redhat (insecure and deprecated) images

     <registry-host>/cloudera/dex/dex-spark-runtime-<spark version>-<cdh version>:<CDE version>

     Example: DockerFile for DEX 1.24.0-b711, Spark 2.4.8 and Cloudera Runtime version 7.1.9.1015

     ```
     FROM
     docker.repository.cloudera.com/cloudera/dex/dex-spark-runtime-2.4.8-7.1
     .9
     .1015:1.24.0-b711
     USER root
     RUN yum install -y git && yum clean all && rm -rf /var/cache/yum
     RUN pip2 install virtualenv-api
     RUN pip3 install virtualenv-api
     USER ${DEX_UID}
     ```

2. Build the Docker image by tagging it with the custom registry and push it to the custom registry.

   Example:

   ```
   mac@local:$ docker build --network=host -t
   docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.8
   -7.2.14.0:1.15.0-b117-custom . -f Dockerfile
   mac@local:$ docker push
   docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.8
   ```

```
-7.2.14.0:1.15.0-b117-custom
```

Here, the custom registry is docker.my-company.registry.com and the registry namespace is custom-dex.

3. Create a custom runtime image resource.

Register custom-spark-dex-runtime docker image as a resource of type Custom-runtime-image.

a) Create a resource for the registries which do not require any authentication. If using a public Docker registry or if the Docker registry is in the same environment, for example, the same AWS account or Azure subscription where the Cloudera Data Engineering service is running, then you do not need to create credentials.

CLI

```
mac@local:$ cde resource create --name custom-image-resource
--image
docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.8
-7.2.14.0:1.15.0-b117-custom --image-engine spark2 --type
custom-runtime-image
```

Note: To obtain $CDE_TOKEN to execute the REST API examples, follow the Getting a Cloudera Data Engineering API access token document.

REST API

```
curl -X POST -k 'https://<dex-vc-host>/dex/api/v1/resources \
  -H "Authorization: Bearer ${CDE_TOKEN}" \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  --data '{
  "customRuntimeImage": {
    "engine": "spark2",
    "image":
"docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.
8-7.2.14.0:1.15.0-b117
-custom"
  },
  "name": "custom-image-resource",
  "type": "custom-runtime-image"
}'
```

Once done, skip to step 4 to submit the job.

b) Create a resource which requires the credentials to access the registry. Use the following command or the API request to create the credentials. These credentials are stored as a secret.

CLI

```
mac@local:$ ./cde credential create --name docker-creds --type
docker-basic --docker-server docker-sandbox.infra.cloudera.com
--docker-username my-username
```

REST API

```
curl -X POST -k 'https://<dex-vc-host>/dex/api/v1/credentials' \
  -H "Authorization: Bearer ${CDE_TOKEN}" \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  --data '{
  "dockerBasic": {
    "password": "password123",
    "server": "docker-sandbox.infra.cloudera.com",
    "username": "my-username"
  },
  "name": "docker-creds",
  "type": "docker-basic"
}'
```

c) Register the custom-spark-dex-runtime Docker image as a resource of type custom-runtime-image by specifying the name of the credential created above.

CLI

```
mac@local:$ ./cde resource create --name custom-image-resource
--image
docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.8
-7.2.14.0:1.15.0-b117
-custom --image-engine spark2 --type custom-runtime-image
--image-credential docker-creds
```

REST API

```
curl -X POST -k 'https://<dex-vc-host>/dex/api/v1/resources \
  -H "Authorization: Bearer ${CDE_TOKEN}" \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  --data ‘{
  "customRuntimeImage": {
    "credential": "docker-creds",
    "engine": "spark2",
    "image":
"docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.
8-7.2.14.0:1.15.0-b117
-custom"
  },
  "name": "custom-image-resource",
  "type": "custom-runtime-image"
}'
```

4. Submit a job by setting the custom-spark-dex-runtime image as a resource using the CDE CLI.

SPARK command

```
mac@local:$ ./cde --user cdpuser1 spark submit
/Users/my-username/spark-examples_2.11-2.4.4.jar
--class org.apache.spark.examples.SparkPi 1000
--runtime-image-resource-name=custom-image-resource
```

JOB command

```
mac@local:$ ./cde --user cdpuser1 resource create --name
spark-jar
mac@local:$ ./cde --user cdpuser1 resource upload --name
spark-jar --local-path spark-examples_2.11-2.4.4.jar
mac@local:$ ./cde --user cdpuser1 job create --name
spark-pi-job-cli --type spark --mount-1-resource spark-jar
--application-file spark-examples_2.11-2.4.4.jar --class
org.apache.spark.examples.SparkPi --user cdpuser1 --arg 22
--runtime-image-resource-name custom-image-resource
```

5. The Spark driver/executor pods should use this image and you can confirm it by opening a shell into those pods and verifying if the external installed libraries or files exist.

Accessing ECR repository from different AWS environments

About this task

Use Case:

There are separate AWS environments for different business units and there is an Amazon Elastic Container Registry (ECR) repository which contains the common Docker images that needs to be accessed from these environments (having different AWS accounts).

By default, the ECR is accessible to the AWS services running in the same AWS environment. The same ECR can be accessed from different AWS environments by updating the ECR repository permissions.

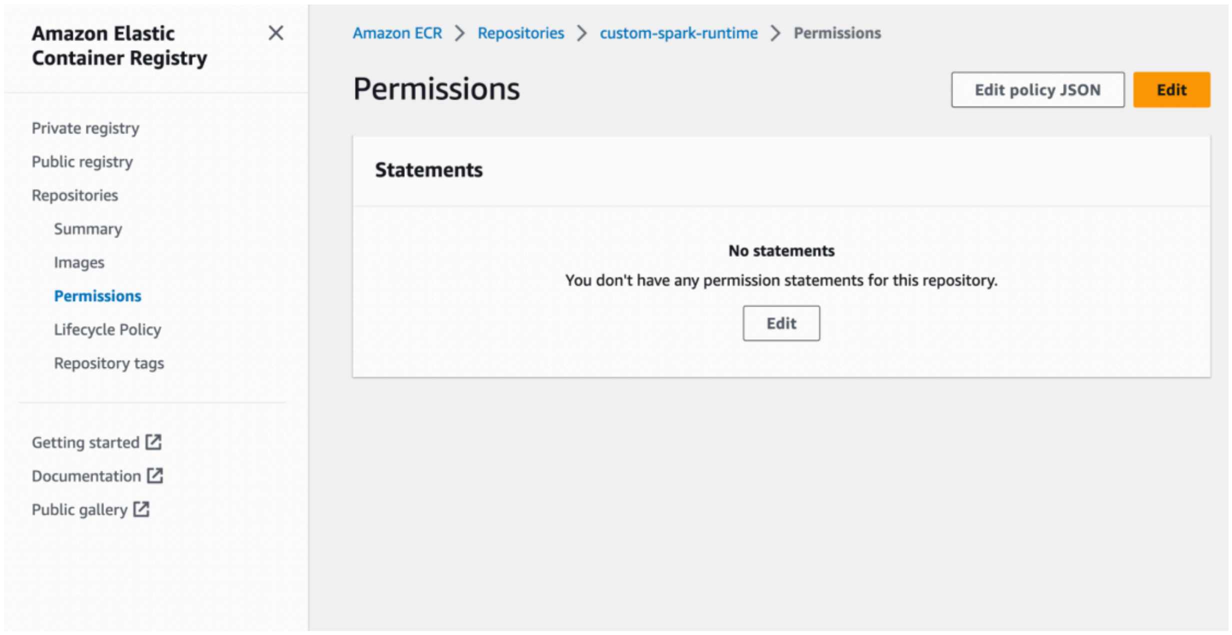
For Example:

- ECR repository is in env1 (AWS-ACCOUNT-1)
- Sales Team services are running in env2 (AWS-ACCOUNT-2)
- Finance Team services are running in env3 (AWS-ACCOUNT-3)

Follow these steps to update the ECR permissions to allow the access for both Sales and Finance teams' services.

Procedure

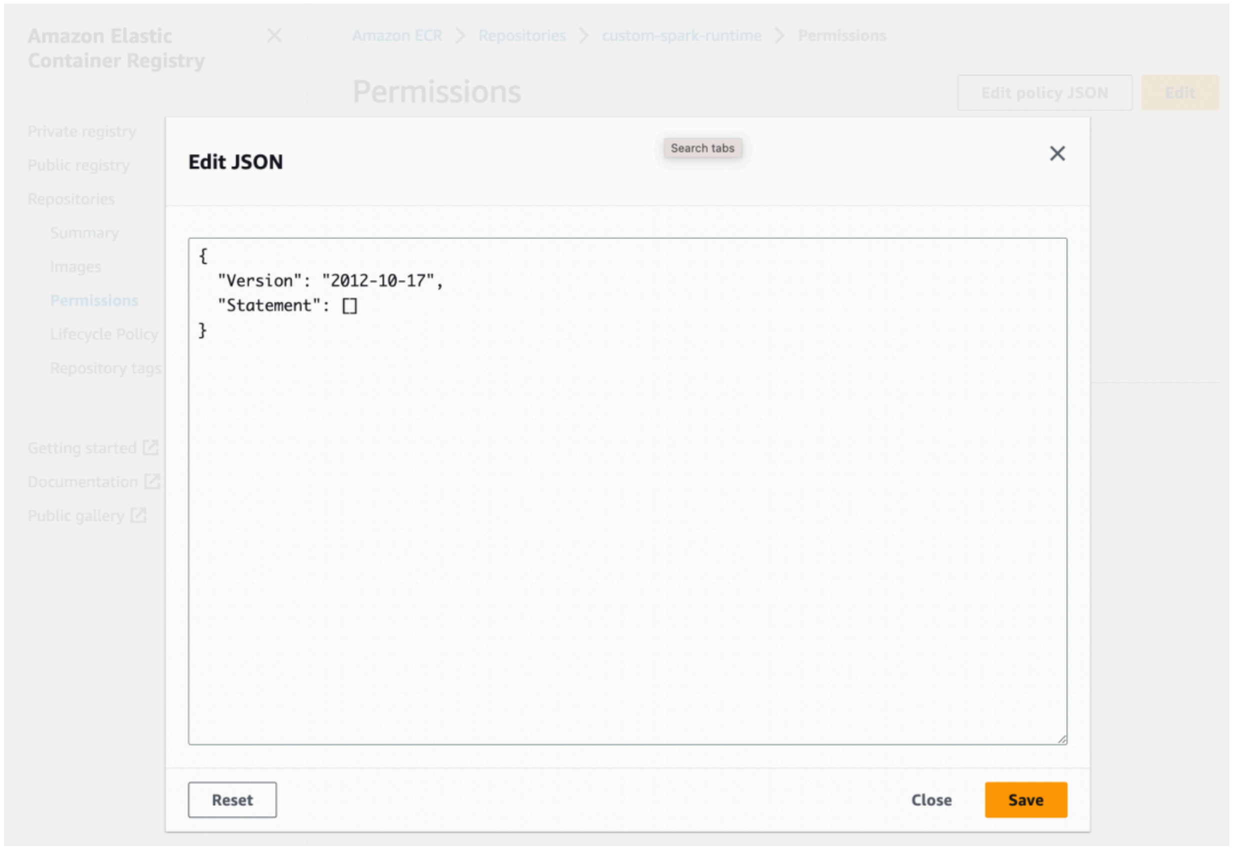
1. Login in to the AWS console and open the desired ECR repository.
2. Click on the Permissions tab.



No statements

You don't have any permission statements for this repository.

3. Click on Edit policy JSON. This shows you the current JSON policy document for the repository.



```
{
  "Version": "2012-10-17",
  "Statement": []
}
```

4. Modify the current JSON policy document by adding a new statement. If the repository has no permissions set yet, then you can copy and paste the JSON policy document below. Make sure to update the required AWS account IDs in this JSON document before saving it.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "crossAccountAllowRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<AWS-ACCOUNT-2>:root",
          "arn:aws:iam::<AWS-ACCOUNT-3>:root"
        ]
      },
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer",
        "ecr:ListImages"
      ]
    }
  ]
}
```

5. Click Save.
6. Verify if the policy has been set successfully by clicking on the Permissions tab again. The new permissions and the AWS accounts IDs should be there.

Results

Note: Ensure that the AWS account IDs are correct, otherwise you may see errors while saving the policy.

Once the permissions are set successfully on ECR for different AWS accounts, then the services running in these AWS environments can access the ECR without any changes in the services or environment. There is no need to provide any credentials to access the ECR from these AWS accounts.

Related Information

Troubleshooting

Troubleshooting

Error: Custom image resource with missing or wrong credentials

Creating a custom image resource with missing or wrong credentials results in the following error, which is listed in the logs or in Kubernetes pod events.

```
Failed to pull image
"docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.
8-7.2.14.0:1.15.0-b117-custom":
rpc error: code = Unknown desc = Error reading manifest
1.15.0-b117-custom in
docker.my-company.registry.com/custom-dex/dex-spark-runtime-2.4.8
-7.2.14.0:errors: denied: requested access to the resource is
denied unauthorized: authentication required
```