Using Custom Spark Runtime Docker Images via API/CLI (Preview)

Date published: 2022-09-06
Date modified: 2023-07-31

Legal Notice

Contents

This is a detailed user guide that demonstrates how to run Spark jobs 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.

Important: If you're using Cloudera Data Engineering (CDE) 1.18 and above, you can't use Custom Spark Runtime Docker Images with CDE 1.17 and below.

Prerequisites

Entitlement for Customer Docker images

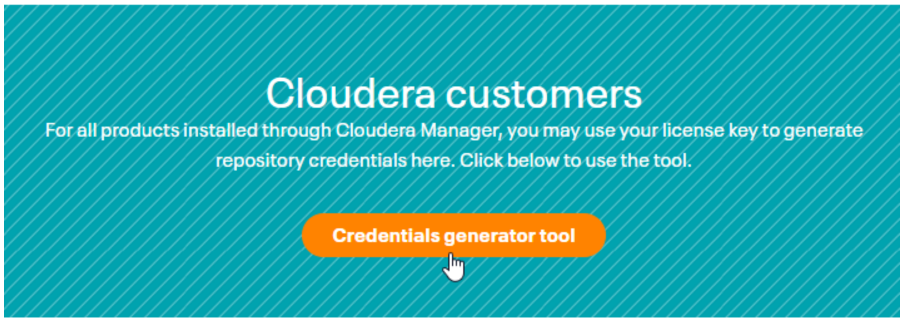
In order to use custom Spark runtime Docker images, your CDP tenant must have the **DE_CUSTOM_RUNTIME** entitlement enabled. If not yet in place, this should be requested via your Cloudera Account Team (primarily your Solution Engineer). They will fulfill the request internally and confirm to you when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

Note that the DE_CUSTOM_RUNTIME entitlement must be enabled **before** a Virtual Cluster is created in order to be able to create custom runtime resources within that cluster.

Docker repository credentials

In order 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.

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

Steps

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 (CDE) version.
   The image should be based on the dex-spark-runtime of the current dex version.
   The relevant dex-spark-runtime image is
   <registry-host>/cloudera/dex/dex-spark-runtime-<spark version>-<cdh version>:<dex version>

   Example: DockerFile for DEX `1.15.0-b117`, Spark 2.4.8 and CDP Runtime version `7.2.14.0`

```
Unset
FROM
docker.repository.cloudera.com/cloudera/dex/dex-spark-runtime-2.4
.8-7.2.14.0-7.2.14.0:1.15.0-b117
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:

```
Unset
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 CDE service is running, then you do not need to create credentials.

   CLI

```
Unset
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

```
Unset
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 command below or the API request to create the credentials. These credentials are stored as a secret.

CLI

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

REST API

```
Unset
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 "custom-spark-dex-runtime" docker image as a resource of type "custom-runtime-image" by specifying the name of the credential created above.

CLI

```
Unset
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

```
Unset
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

```
Unset
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

```
Unset
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

Use Case:

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

By default, The ECR will be 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.

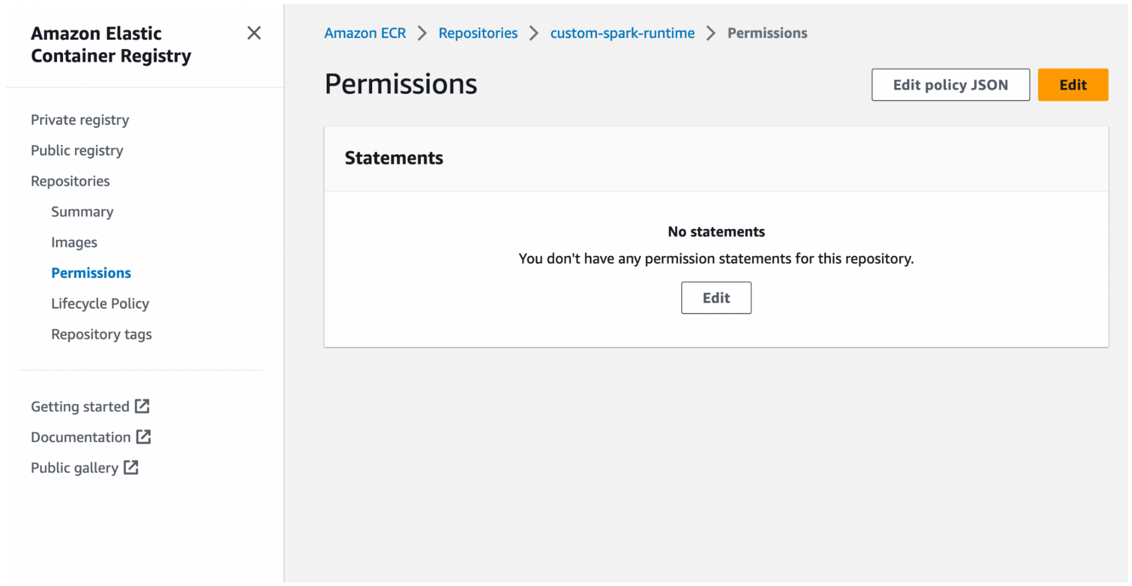
For Example :

- 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)

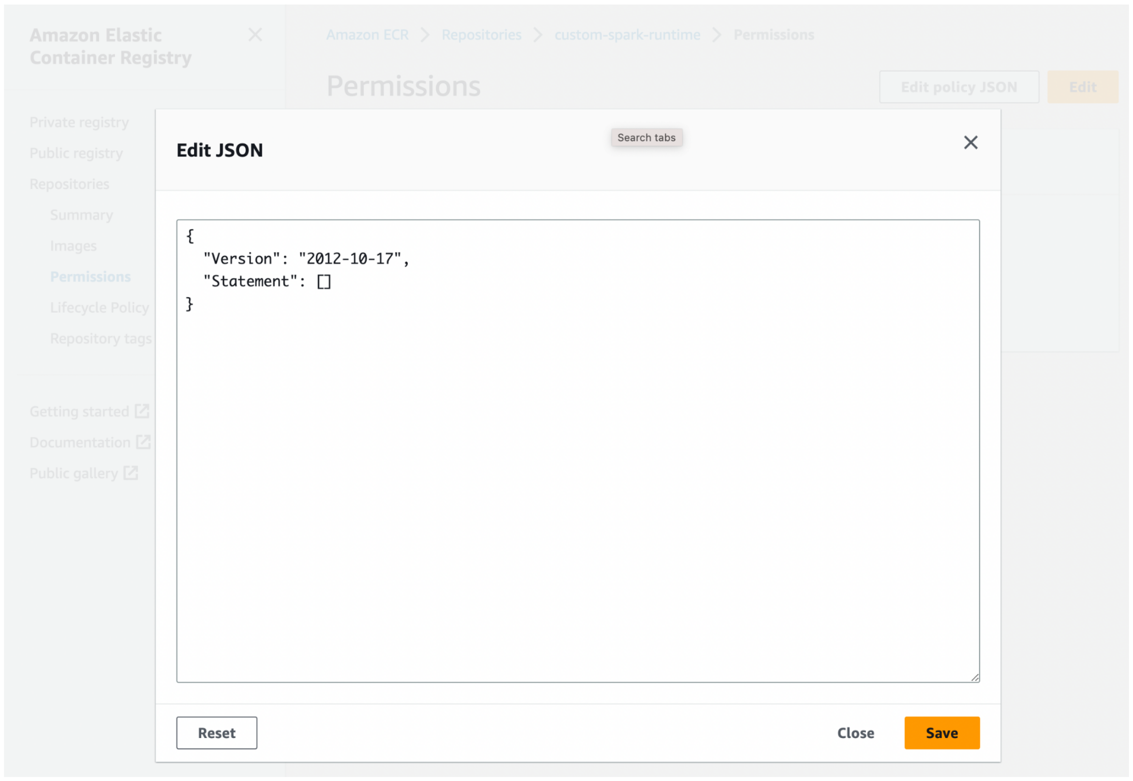
Below steps can be followed to update the ECR permissions to allow the access for both Sales and Finance teams' services.

Steps:

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



3. Click on Edit Policy Json. This will show 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 simply copy and paste the below JSON policy document (Update the required AWS account ids in this document)

Unset
```
{
  "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 on Save button.
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.

Note : Ensure 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 itself. There is no need to provide any credentials to access the ECR from these AWS accounts.

Troubleshooting

Error: Custom image resource with missing or wrong credentials

Creating a custom image resource with missing or wrong credentials should result in the below error which can be seen in the logs or in Kubernetes pod events.

Example

```
Unset
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
```