Creating Connectors for AWS Marketplace

You can develop your own custom connector software, and then place it on AWS Marketplace to sell to AWS Glue customers.

Overview of Creating Connectors for AWS Marketplace

AWS Glue Custom Connectors allows you to discover and subscribe to more than 50 connectors on AWS Marketplace. You can also use AWS Glue Spark runtime interfaces to plug-in connectors built for Apache Spark Datasource, Athena federated query, and JDBC APIs. This chapter helps you to build and test custom connectors; and deploy them for connectivity with AWS Glue Spark applications.

1. Create a custom connector as described in Step 1: Developing Marketplace Connectors.
2. Package your connector as described in Step 2: Packaging Marketplace Connectors.
3. Test your connector as described in Step 3: Testing Marketplace Connectors.
4. Validate your connector as described in Step 4: Validating Marketplace Connectors.
5. Create a deep link for your connector. This link is used in the AWS Marketplace listing. See Step 5: Creating a Deployment Link for Marketplace Connectors.
6. Use Markdown to create a **Usage Instructions** document to include with the connector on AWS Marketplace.
7. After you've completed the above steps, contact your assigned Technical Account Manager (TAM) to update your product. After your product is updated, you should be able to see the new deployment template at the end of the subscription workflow for your product in AWS Marketplace.

Step 1: Developing Marketplace Connectors

You can create connectors for JDBC, Spark, or Athena data sources. Each connector type has different requirements.

- To create a connector for Open Spark data stores, see Developing Spark connectors
- To create a connector for Amazon Athena data stores, see Developing Athena connectors
- To create a connector for JDBC data stores, see Developing JDBC connectors

Step 2: Packaging Marketplace Connectors

This section describes how to create and publish a container product with the required connector jars to AWS Marketplace.

1. Setup AWS Command Line Interface (AWS CLI). Refer to Installing, updating, and uninstalling the AWS CLI in the *AWS Command Line Interface User Guide* for the instructions.

   You can install either of the following versions:

   - AWS CLI version 2
   - AWS CLI version 1 v1.17.10 or later

2. Install Docker Engine, as described in "Install Docker Engine" in the *Docker Engine* online documentation.

3. Create and start a docker image, as described in "Orientation and setup" in the Docker online documentation.

4. Prepare the `config` file to provide metadata about the connector that you're publishing. A sample file is provided below. All the keys in the example below are **required**.

```
  {
  "releasetimestamp": "2020-12-21 12:00:00",
  "connectiontype": "MARKETPLACE",
  "classname": "partner.jdbc.salesforce.salesforcedriver",
  "publishername": "partner",
  "connectortype": "JDBC",
  "version": "19.0.7362.0",
  "description": "Partner JDBC Driver for Salesforce",
  "supportinformation": "Please check for this driver's online help."
}
```

5. Download the shell script for creating and publishing a container with the required connector JAR files and `config` file for the connector JAR files.

   Tip

   Copy and run the script in a new folder to avoid conflicts, and for the script to run efficiently.

   This script creates a new container with the necessary JAR files and then pushes the container to the newly created Amazon Elastic Container Registry (Amazon ECR) repository.

```
#!/usr/bin/env bash
# Script to create new container with the supplied jars and to push the
container to the created ECR repository

 # Steps to run the script :
 # 1) chmod +x ./container_setup.sh
 # 2) ./container_setup.sh --jar-file-paths [Paths to jar file] --jar-folder-
paths [Paths to folder where jars are present] --aws-region [AWS Region] --
```

```
ecr-repo-name [ECR repo name] --aws-account-id [AWS Account ID] --aws-cli-
profile [Local AWS CLI profile name]

 # Example 1: ./container_setup.sh --aws-account-id [AWS Account ID] --ecr-
repo-name connector-jars --aws-region us-west-1 --jar-file-paths
/Users/AWS/Desktop/local_repo/partner.jdbc.salesforce.jar --jar-folder-paths
/Users/AWS/Desktop/local_repo/
 # Example 2: ./container_setup.sh --aws-account-id [AWS Account ID] --ecr-
repo-name connector-jars --aws-region us-west-1 --jar-file-paths
partner.jdbc.salesforce.jar --jar-folder-paths /Users/AWS/Desktop/local_repo/
--aws-cli-profile user1
 # Example 3: ./container_setup.sh --aws-account-id [AWS Account ID] --ecr-
repo-name connector-jars --aws-region us-west-1 --jar-file-paths
/Users/AWS/Desktop/local_repo/partner.jdbc.salesforce.jar,partner.jdbc.salesf
orce.jar --jar-folder-paths /Users/AWS/Desktop/local_repo/ --aws-cli-profile
user1
 set -x

 ecr_repo_name=""
 aws_region=""
 aws_account_id=""
 jarfile_paths=""
 jarfolder_paths=""
 aws_cli_profile=""
 config_path=""

 # call_help() function to give example of how to give the parameters
call_help(){
  echo "
     Steps to run the script :

     1) chmod +x ./container_setup.sh
     2) ./container_setup.sh --jar-file-paths [Paths to jar file] --jar-
folder-paths [Paths to folder where jars are present] --aws-region [AWS
Region] --ecr-repo-name [ECR repo name] --aws-account-id [AWS Account ID]

     Example 1: ./container_setup.sh --aws-account-id [AWS Account ID] --ecr-
repo-name connector-jars --aws-region us-west-1 --jar-file-paths
/Users/AWS/Desktop/local_repo/partner.jdbc.salesforce.jar --jar-folder-paths
/Users/AWS/Desktop/local_repo/

     Example 2: ./container_setup.sh --aws-account-id [AWS Account ID] --ecr-
repo-name connector-jars --aws-region us-west-1 --jar-file-paths
partner.jdbc.salesforce.jar --jar-folder-paths /Users/AWS/Desktop/local_repo/

     Example 3: ./container_setup.sh --aws-account-id [AWS Account ID] --ecr-
repo-name connector-jars --aws-region us-west-1 --jar-file-paths
/Users/AWS/Desktop/local_repo/partner.jdbc.salesforce.jar,partner.jdbc.salesf
orce.jar --jar-folder-paths /Users/AWS/Desktop/local_repo/ --aws-cli-profile
user1

     Note: You have a choice to give only --jar-file-paths or --jar-folder-
paths or you can give both the parameters. But one parameter is required.

     --jar-file-paths        Optional Field      Paths to jar file (Refer
Note) (Can give multiple jar file paths eg: /usr/jarfile1,/usr/jarfile2)
```

```
     --jar-folder-paths      Optional Field      Paths to folder where jar is
present (Refer Note) (Can give multiple jar folder paths eg:
/usr/jarfolder1/,/usr/jarfolder2/,/usr/jarfolder3)
     --aws-account           Required Field      AWS Account ID
     --ecr-repo-name         Required Field      ECR repository name
     --aws-region            Required Field      AWS Region
     --aws-cli-profile       Optional Field      Local AWS CLI profile name
(If no AWS CLI profile name given, it will take the default AWS CLI profile)
     --config-path           Required Field      config file that describes
key attributes about the connector jar file, read the usage guide doc to see
how to prepare the config file.
  "
 }

 # Call help menu if --help passed
 if [[ (${#@} == 1) && ("$1" == "--help") ]]; then
   call_help
   exit 1
 fi

  # To create a Dockerfile
cat > Dockerfile <<- "EOF"
FROM amazonlinux
COPY ./connector_jars  ./jars
EOF


 # Create a new directory for the connector jars
 mkdir -p connector_jars

 # Check the total number of parameters passed to the shell script
 if [[ (${#@} != 10 && ${#@} != 12) && (${#@} != 14) ]]; then
   echo "Invalid parameters given"
   call_help
   exit 1
 fi

 # Checking and Initialising variables based on given parameters and flags
 while [[ $# -ne 0 ]]; do
   if [ "$1" == "--ecr-repo-name" ] ; then
     ecr_repo_name="$2"
     shift 2
   elif [ "$1" == "--aws-region" ] ; then
     aws_region="$2"
     shift 2
   elif [ "$1" == "--aws-account-id" ] ; then
     aws_account_id="$2"
     shift 2
   elif [ "$1" == "--jar-file-paths" ] ; then
     jarfile_paths="$2"
     shift 2
   elif [ "$1" == "--jar-folder-paths" ] ; then
     jarfolder_paths="$2"
     shift 2
   elif [ "$1" == "--aws-cli-profile" ] ; then
     aws_cli_profile="$2"
     shift 2
```

```
  elif [ "$1" == "--config-path" ] ; then
    config_path="$2"
    shift 2
  else
    echo "Invalid Parameters"
      call_help
      exit 1
  fi
done

re='^[0-9]+$'

# Check the parameters passed

# To check whether the parameter contains only number, then it AWS Account
ID
if [[ $aws_account_id =~ $re ]]; then
  if [[ ${#aws_account_id} = 12 ]]; then
      echo "Accepted ${aws_account_id} as AWS Account ID"
  else
      echo "Given ${aws_account_id} is an invalid AWS Account ID, expected a
12-digit number"
      exit 1
  fi
fi

# To check whether the parameter is a path to a connector jars file and
copying connector jars to connector_jars folder
IFS=','
read -a str_arr_folder <<< "$jarfolder_paths"
for folder_name in "${str_arr_folder[@]}";
do
  if [[ -d $folder_name ]]; then
    folder_path_length=${#folder_name}
    suffix=${folder_name:folder_path_length-1:folder_path_length}
    slash="/"
    if [[ $suffix == $slash ]]; then
      cp $folder_name*.jar ./connector_jars
    else
      cp $folder_name/*.jar ./connector_jars
    fi
  else
    echo "$folder_name is not a valid Directory"
    exit 1
  fi
done
IFS=' '

# To check whether the parameter is a path to a connector jars folder
IFS=','
read -a str_arr_file <<< "$jarfile_paths"
for file_name in "${str_arr_file[@]}";
do
  if [[ -f $file_name ]]; then
    size=${#file_name}
    check=${file_name:size-4:size}
    name=".jar"
```

```
    if [[ $check == $name ]]; then
      cp $file_name ./connector_jars
    fi
  else
    echo "$file_name is not a valid File"
    exit 1
  fi
done
IFS=' '

# check if --config-path is supplied, copy the config file into the dest
folder if yes.
if [ -z "$config_path" ]
then
  echo "--config-path is required to run the script, exiting..."
  exit 1
else
  cp $config_path ./connector_jars/config.json
fi


# To check whether the given parameter is an AWS Region
IFS='-'
read -a split_region <<< "$aws_region"
if [[ (${#split_region[*]} == 3 && ${#split_region[2]} == 1) &&
(${#split_region[0]} == 2 && ${split_region[2]} =~ $re) ]]; then
  IFS=' '
  echo "Accepted $aws_region as AWS Region"
else
  echo "Given $aws_region is an Invalid AWS Region"
  exit 1
fi

# To Authenticate to your default registry
if [ -z "$aws_cli_profile" ]
then
  # docker_login=$(aws ecr get-login --no-include-email --region
$aws_region)
  # ret=$(docker_login)
  aws ecr get-login-password --region $aws_region | docker login --username
AWS --password-stdin $aws_account_id.dkr.ecr.$aws_region.amazonaws.com
else
  aws ecr get-login-password --region $aws_region --profile $aws_cli_profile
| docker login --username AWS --password-stdin
$aws_account_id.dkr.ecr.$aws_region.amazonaws.com
fi

# If login succeeded, we get an error code 0
if [ $? -eq "0" ]
then
   echo "Login Succeeded"
   aws ecr describe-repositories --repository-names $ecr_repo_name --region
$aws_region || aws ecr create-repository --repository-name $ecr_repo_name --
region $aws_region
   docker build -t connector:latest .
   docker tag connector
$aws_account_id.dkr.ecr.$aws_region.amazonaws.com/$ecr_repo_name
```

```
    docker push
$aws_account_id.dkr.ecr.$aws_region.amazonaws.com/$ecr_repo_name
 else
    echo "Login Failed due to Invalid AWS Credentials or Invalid AWS Region"
 fi
```

6. You can run the following command to see the usage examples for the script and how to run the script to publish the docker image to Amazon ECR:

   ```
   bash container_setup.sh --help
   ```

7. Create, test, and publish your container product to AWS Marketplace. Refer to the instructions in Getting started with container products in the *AWS Marketplace Seller Guide*.

Step 3: Testing Marketplace Connectors

Perform the integration tests to test a connector against most AWS Glue features locally before releasing your connector to the AWS Glue ETL connector marketplace.

Refer to the Marketplace Connector Integration Tests Guide on GitHub, which shows you how to:

- Set up the tool
- Configure each test
- Run each test

The results of these tests are using in to provide the validation information for your connector.

Step 4: Validating Marketplace Connectors

You must validate your connector against AWS Glue supported features in AWS Glue job system before you publish it to AWS Marketplace.

Refer to the Marketplace Connector Validation Guide on GitHub.

1. Complete the steps in the previous section, Step 3: Testing Marketplace Connectors, to perform the validation tests.

2. After you finish the validation testing, complete the steps in the **Reporting** section of the Marketplace Connector Validation Guide.

Step 5: Creating a Deployment Link for Marketplace Connectors

At the end of a customer's subscription workflow in the AWS Marketplace, they need a way to activate the connector they just purchased in AWS Glue Studio. This section describes how to prepare a deep link URL, which redirects the user back to the AWS Glue Studio console to activate the connector.

Please follow these steps carefully and reach out to your AWS contact if you have any questions.

1. Gather the information required for the deep link URL. The deep link URL is an absolute URL that points to an endpoint provided by AWS Glue Studio. The base url is:

   ```
   https://console.aws.amazon.com/gluestudio/home#/connector/add-connection?PARAMETERS
   ```

   There are several parameters you need to append to the base URL so that your connector can be integrated with AWS Glue Studio correctly.

Connector-related parameters:

- **connectorName** (required): The name of your connector. You can also include your company name, if you want. The value should be an alphanumeric string that uses a space character as a delimiter. This field is editable by the end-user.

  ```
  connectorName="Virtual Company Simple DB"
  ```

- **connectorType** (required): The interface type that your connector is built upon. The accepted values are `Spark`, `Athena`, or `Jdbc`. This field is not editable by the end-user.

  ```
  connectorType="Jdbc"
  ```

- **connectorDescription** (optional): A brief summary about this connector that contains only alphanumeric letters, spaces, and the characters ",", ";", or ".". We recommend limiting this field to 50 words or less. This field is editable by the end-user.

  ```
  connectorDescription="A simple description"
  ```

- **connectorUrl** (required): The URL for the corresponding Amazon Elastic Container Registry (Amazon ECR) image that contains your connector.

  Important

  Use the Amazon ECR that AWS Marketplace provides you with after your initial image is copied into their account. **DO NOT** use the URL for the image in your own account.

See Publishing container products in the *AWS Marketplace Seller Guide* for details about when you will get the final URL.

```
connectorUrl="https://mp-account-#.dkr.ecr.us-east-
1.amazonaws.com/product_id/container_group_id/myconnectorimage:version_
title-latest"
```

- **connectorVersion** (required): The version of your connector. The value should contain only numeric letters delimited by ".". The length limit for this field is 36 characters. This field is not editable by the end-user.

  ```
  connectorVersion="7.5.5"
  ```

- **connectorClassName** (required): For JDBC connectors, this field should be the class name of your JDBC driver. For Spark connectors, this field should be the format when loading spark data source with the `format` operator.

  ```
  connectorClassName="some.class.name"
  ```

Connection-related parameters:

- **connectionAccessJdbcURLFormat** (required for JDBC connectors): The JDBC URL templates supported by your connector. If you support more than one template, you can reuse this keyword to specify each of them. Also, each template should start with a name followed by an equal sign. For example, if your connector supports a template that uses a username and password for authentication, then you can put:

  ```
  username_password=jdbc:virtualcomp:simpleDB:user=${Username};password=$
  {Password}
  ```

  The first token, `username_password`, can be any non-empty string. This represents the title presented to the user from a drop-down component in the AWS Glue Studio console. You should make this title easy to read and identifiable. If there is more than one template, make sure this title is unique, for example:

  ```
  username_password_template1= ...
  username_password_template2= ...
  ```

  For the JDBC URL template, you can insert placeholders for parameter values that user has to fill in at runtime for their specific data store. You should also include a usage guide on your product page to call out these mandatory fields and what values are expected for them. Each placeholder variable should be enclosed in curly braces and prefixed with a dollar sign, for example `${RuntimeKey}`.

  If your JDBC URL template contains user name and password parameters, then the placeholder variables for them should *always* be exactly `Username` and `Password`. This is required for AWS Glue Studio to extract them correctly and apply the proper encryption.

A full example looks like this:

```
connectionAccessJdbcURLFormat="username_password=jdbc:virtualcomp:simpl
eDB:user=${Username};password=${Password}"

connectionAccessJdbcURLFormat="oauth=jdbc:virtualcomp:simpleDB:OAuthSet
tingsLocation=${OAuthSettingsLocation};InitiateOAuth=REFRESH;"
```

- **connectionAccessJdbcURLParamsDelimiter** (required for JDBC connectors): For JDBC connectors, this field specifies the delimiter used to separate parameters in the JDBC URL template. This enables users to add additional JDBC parameter key-value pairs in the AWS Glue Studio console.

```
connectionAccessJdbcURLParamsDelimiter=";"
```

2. After you have all the parameters ready, you can join them together with an ampersand (`&`) and append the result to the base URL. The outcome should look like this (with no spaces between each parameter query):

```
https://console.aws.amazon.com/gluestudio/home#/connector/add-
connection
?connectorName="Virtual Company Simple DB"
&connectorType="Jdbc"
&connectorDescription="A virtual company connector that connects to
Simple DB"
&connectorUrl="https://mp-account-#.dkr.ecr.us-east-
1.amazonaws.com/product_id/container_group_id/myconnectorimage:version_
title-latest"
&connectorVersion="7.5.5"
&connectorClassName="virtualcomp.db.simpledb.driver"
&connectionAccessJdbcURLFormat="username_password=jdbc:virtualcomp:simp
leDB:user=${Username};password=${Password}"
&connectionAccessJdbcURLFormat="oauth=jdbc:virtualcomp:simpleDB:OAuthSe
ttingsLocation=${OAuthSettingsLocation};InitiateOAuth=REFRESH;"
&connectionAccessJdbcURLParamsDelimiter=";"
```

3. Use an URI encoder such as the one at https://toolbox.googleapps.com/apps/encode_decode/ to encode the parameter values. This helps to avoid any character escaping issues. Using the example in the previous step, the encoded URL string should look like this (but as a one-line string):

```
https://console.aws.amazon.com/gluestudio/home#/connector/add-
connection?connectorNa
me=%22Virtual%20Company%20Simple%20DB%22&connectorType=%22Jdbc%22&conne
ctorDescription=%22A%20vir
tual%20company%20connector%20that%20connects%20to%20Simple%20DB%22&conn
ectorUrl=%22https://mp-acc
ount-#.dkr.ecr.us-east-
1.amazonaws.com/product_id/container_group_id/myconnectorimage:
version_title-
latest%22&connectorVersion=%227.5.5%22&connectorClassName=%22virtualcom
p.db.simpledb
```

```
.driver%22&connectionAccessJdbcURLFormat=%22username_password=jdbc:virt
ualcomp:simpleDB:user=$%7B
Username%7D;password=$%7BPassword%7D%22&connectionAccessJdbcURLFormat=%
22oauth=jdbc:virtualcomp:
simpleDB:OAuthSettingsLocation=$%7BOAuthSettingsLocation%7D;InitiateOAu
th=REFRESH;%22&connection
AccessJdbcURLParamsDelimiter=%22%3B%22
```

Note

Only the values of the parameters are encoded. You should only encode the values of the parameter (the value to the right of each assignment operator '=').

4. Update the deployment template field in your Product Load Form (PLF) with this deep link URL.

   The PLF is an Excel spreadsheet. Review Getting started with container products in the *AWS Marketplace Seller Guide* to see where to download the PLF.

   After you download the PLF, search within the spreadsheet for `deployment`. You should see the following columns:

   - **Container Image Set 1: Compatible AWS Services**: Enter **`AWS Glue`**.
   - **Container Image Set 1: Deployment Template Text 1**: Enter **`Activate connector in AWS Glue Studio`**.
   - **Container Image Set 1: Deployment Template URL 1**: Paste in the deeplink URL you constructed in this procedure.