IBM Db2 11.5.8

for Red Hat OpenShift
2023-02-15

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:
© (your company name) (year).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Contents

Tables

Chapter 1. Db2 on Red Hat OpenShift

Db2® can be deployed in a Red Hat® OpenShift® cluster as a containerized micro-service, or pod, managed by Kubernetes.

Note: While it is possible to deploy the containerized version of Db2 on other Kubernetes-managed container platforms, the documentation focuses on the Red Hat OpenShift deployment.

You deploy Db2 to your OpenShift cluster through a series of API calls to the Db2 Operator. The table below lists the Db2 Operators that have been released in the version 1.x, 2.x, and 3.0 channels, and their supported Db2 versions for deployment on OpenShift.

Attention: The Db2 containerized solution utilizes both OpenShift and Kubernetes. This relationship determines the lifespan of each Db2 containerized solution: When the supporting OpenShift and Kubernetes versions are no longer supported, any Db2 containerized solution that is based on those versions is also no longer supported.

Table 1. Db2 Operators and their associated Db2 engines

Db2 Operator version	Db2 Operator upgrade channel	Db2 Engine version	OCP Version	Container Application Software for Enterprises (CASE) version
1.1.3	1.1	11.5.6.0	4.6	4.0.1
1.1.5	1.1	11.5.6.0-cn2	4.6	4.0.3
1.1.6	1.1	11.5.6.0-cn3	4.6, 4.8	4.0.4
1.1.8	1.1	11.5.6.0-cn5	4.6, 4.8	4.0.6
1.1.9	1.1	11.5.7.0	4.6, 4.8	4.0.7
1.1.10	1.1	11.5.7.0-cn1	4.6, 4.8	4.0.8
1.1.11	1.1	11.5.7.0-cn2	4.6, 4.8	4.0.9
1.1.12	1.1	11.5.7.0-cn3	4.6, 4.8	4.0.10
1.1.13	1.1	11.5.7.0-cn4	4.6, 4.8	4.0.11
2.0.0	2.0	11.5.7.0-cn5	4.6, 4.8, 4.10	4.5.0
2.2.0	2.2	11.5.7.0-cn7	4.6, 4.8, 4.10	4.5.3

The table below lists the Db2 Operator channels and the Db2 engine versions they support.

Table 2. Db2 Operator channels and supported Db2 engine versions

Channel	Db2 engine version
v1.1	11.5.6
v2.x	11.5.7
v110508.0	11.5.8

If you are new to the world of containers and Kubernetes and would like to learn more, please see the Core Concepts section of the Red Hat OpenShift Documentation site.

What's new in Db2 11.5.8 for Red Hat OpenShift

Db2 for Red Hat OpenShift has been enhanced for the 11.5.8 release, with improved security.

In addition to security enhancements and defect corrections, Db2 on OpenShift 11.5.8 includes the following feature enhancements:

- Support for Valero for backup and restore operations.
- Rolling updates for HADR deployments.

Installing Db2

To add a Db2 database to Red Hat OpenShift, you first prepare your Red Hat OpenShift cluster and then install and deploy the database.

Db2 for Red Hat OpenShift supports the following chip sets:

- x86-64
- POWER ppc64le
- IBM Z s390

Installing the Db2 Operator

Db2 11.5.5 and later for Red Hat OpenShift are Operator-enabled installations, allowing you more control over your deployment.

The Db2 Operator is acquired from either the IBM Operator Catalog or the Red Hat Marketplace. The IBM Operator catalog is accessible through your OpenShift UI console, while the Red Hat Marketplace is a web site.

If your target Db2 cluster is disconnected from the internet (air-gapped), you can use the IBM Cloud Pak CLI (cloudctl) command-line utility to install and configure the Db2 Operator. You can run the utility from a bastion host or from a suitably equipped portable device.

Installing the Db2 Operator from the IBM Operator Catalog

IBM provides a catalog of product offerings in the form of a catalog index image. To view IBM offerings in the OpenShift Operator catalog, the catalog index image needs to be enabled. You enable the image on a Red Hat OpenShift cluster through a CatalogSource resource.

Procedure

To enable the IBM Operator Catalog in your OpenShift cluster:

1. From your OpenShift UI console, roll over the + icon on the tool bar and select **Import YAML**.
2. Paste the following YAML content into the space provided:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: ibm-operator-catalog
  namespace: openshift-marketplace
spec:
  displayName: "IBM Operator Catalog"
  publisher: IBM
  sourceType: grpc
  image: icr.io/cpopen/ibm-operator-catalog
  updateStrategy:
    registryPoll:
      interval: 45m
```

3. Click **Create**.

From the navigation panel, under *Provider Type*, you should see an option for *IBM Operators*, from which you can install the Db2 Operator. For the latest Db2 operator version and channel information, see Db2 on Red Hat OpenShift.

What to do next

You can verify the installation by running the following from the command line:

```
oc get CatalogSources ibm-operator-catalog -n openshift-marketplace
```

- If the installation was successful, you should see output that is similar to this:

```
NAME                   DISPLAY                TYPE   PUBLISHER      AGE
ibm-operator-catalog   IBM Operator Catalog   grpc   IBM            50s
```

- If the installation failed, the following message is displayed:

```
Error from server (NotFound): catalogsources.operators.coreos.com
                "ibm-operator-catalog" not found
```

To help resolve this error, run the following command to check the pods and CatalogSource resources in the `openshift-marketplace` namespace:

```
oc get catalogsource,pods -n openshift-marketplace
```

The output should provide information about your CatalogSource resources and status of the pods; for example:

```
NAME                            DISPLAY        TYPE  PUBLISHER  AGE
catalogsource.operators.coreos.com/certified-operators  Certified Operators  grpc  Red Hat
20d
catalogsource.operators.coreos.com/community-operators  Community Operators  grpc  Red Hat
20d
catalogsource.operators.coreos.com/ibm-operator-catalog  IBM Operator Catalog  grpc  IBM
48s
catalogsource.operators.coreos.com/opencloud-operators  IBMCS Operators    grpc  IBM     48s
catalogsource.operators.coreos.com/redhat-marketplace   Red Hat Marketplace  grpc  Red Hat
20d
catalogsource.operators.coreos.com/redhat-operators    Red Hat Operators   grpc  Red Hat   20d
NAME                     READY  STATUS  RESTARTS  AGE
pod/certified-operators-575f586fd8-m2ldh  1/1   Running  0     41m
pod/community-operators-57fd7676ff-sqzgs  1/1   Running  0     15h
pod/ibm-operator-catalog-85b2w       1/1   Running  0     48s
pod/marketplace-operator-5fcf68c65c-l8tcg  1/1   Running  0     5d8h
pod/opencloud-operators-wbb9k        1/1   Running  0     48s
pod/redhat-marketplace-665c9c6db4-hhfkd   1/1   Running  0     41m
pod/redhat-operators-8678ddbc5-6szpp    1/1   Running  0     4d1h
```

Installing the Db2 Operator from the Red Hat Marketplace

The Red Hat Marketplace provides a catalog of Operator products that can be purchased and installed on your Red Hat OpenShift cluster. The *IBM Db2 on Cloud Pak for Data* Operator is provided as a free purchase.

Before you begin

Ensure that you complete the following prerequisite tasks:

- Create an account with Red Hat Marketplace (see Account management for steps on how to register with the Marketplace).
- Set up a working Red Hat OpenShift cluster.
- Register your cluster with the Red Hat Marketplace (see Clusters for steps on how to register your cluster with the Marketplace).
- Create an OpenShift Container Platform (OCP) Project (Namespace) on the OCP cluster.

Procedure

1. Log in to the Red Hat Marketplace.
2. Using the field provided, search for.**Db2**
3. Click the **IBM Db2 on Cloud Pak for Data** product tile to open the product details page.
4. From the *Pricing* tab, click **Get now**.
5. Click **Purchase**.
6. Click **View software**, or select **Workspace** > **My software**, from the menu bar, and then click the **IBM Db2 on Cloud Pak for Data** tile.
7. Click **Install operator** and ensure that the following items are selected:
   - The default selections for *Update channel* and *Approval strategy*.
   - Under *Target clusters*, in the *Name* column, the check box to the left of your registered cluster.
   - From the drop-down list in the *Namespace scope* column, the OCP namespace associated with your cluster. This is the project name that was used for operator deployment from the RHM portal.
8. Click **Install**.

   After a few moments the Db2 Operator should be installed.
9. From the *Operators* tab, select the drop-down list to the far-right of your operator and select **Cluster Console**.
10. From the navigation panel, select **Operators** > **Installed operators**.
11. From the top-left of the *Installed operators* page, expand the *Project* drop-down list and select your project/namespace.
12. From the list of installed Operators, click **IBM Db2** and follow the instructions on the *Db2 Operators* page to install Db2 to your Red Hat OpenShift cluster.

Installing the Db2 Operator from the command line

You can install the Db2 Operator using the IBM Cloud Pak CLI tool.

About this task

The IBM Cloud Pak CLI (cloudctl) provides significant benefits when dealing with air-gapped (disconnected) environments for case management. It provides a common framework for IBM operators around a consistent and optimized air gap install experience, using bastion, non-bastion or portable storage.

You will need to install the cloudctl tool before proceeding.

Procedure

1. Download and extract the Container Application Software for Enterprises (CASE) bundle:
   a. Set up environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment:

      ```
      export CASE_NAME=ibm-db2uoperator
      export CASE_VERSION=<CASE version of latest Db2 Operator>
      export OFFLINECASE=/tmp/cases
      export CASEPATH="https://github.com/IBM/cloud-pak/raw/master/repo/case"
      export OFFLINECASE=${OFFLINECASE}/${CASE_NAME}
      export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz
      ```

   b. Create a directory to save the CASE bundle to a local directory:

      ```
      mkdir -p ${OFFLINECASE}
      ```

   c. Download the CASE bundle:

```
$ cloudctl case save --repo ${CASEPATH} --case ${CASE_NAME} --version ${CASE_VERSION} --
outputdir ${OFFLINEDIR}
Downloading and extracting the CASE ...
- Success
Retrieving CASE version ...
- Success
Validating the CASE ...
- Success
Creating inventory ...
- Success
Finding inventory items
- Success
Resolving inventory items ...
Parsing inventory items
- Success
```

d. Verify that the CASE bundle and images csv have been downloaded:

```
$ ls ${OFFLINECASE}
total 128K
drwxr-xr-x 2 root root    6 Jan 20 11:10 charts/
-rw-r--r-- 1 root root 116K Jan 20 11:10 ibm-db2uoperator-1.0.x.tgz
-rw-r--r-- 1 root root   32 Jan 20 11:10 ibm-db2uoperator-1.0.x-charts.csv
-rw-r--r-- 1 root root 5.2K Jan 20 11:10 ibm-db2uoperator-1.0.x-images.csv
```

e. Extract the CASE bundle:

```
cd ${OFFLINECASE}
tar -xvzf ${CASE_ARCHIVE}
```

2. Install the Db2 catalog:

```
cloudctl case launch                        \
    --case ${OFFLINECASE}                   \
    --namespace openshift-marketplace       \
    --inventory db2uOperatorSetup           \
    --action installCatalog
```

3. Install the Db2 operator:

```
cloudctl case launch                         \
     --case ${OFFLINECASE}                   \
     --namespace ${NS}                       \
     --inventory db2uOperatorStandaloneSetup \
     --action installOperator
```

4. Deploy Db2, using the Db2uCluster API.

Installing the Db2 Operator in an air-gapped environment

There are two ways of installing the Db2 Operator in an air-gapped environment, either through a bastion host or by transferring it to the cluster from a portable device.

Installing the Db2 Operator through a bastion host

You can install the Db2 Operator to an air-gapped Red Hat on OpenShift (RHOS) cluster that uses a bastion host for connections.

Before you begin

Before installing the Db2 Operator on your bastion machine, ensure that it is properly configured by logging onto the machine and performing the following tasks:

- Verify that the bastion machine has access to the following:
  - public internet [to download the requried Container Application Software for Enterprises (CASE) bundle]
  - a target image registry (where the images are mirrored)
  - a target Db2 cluster onto which to install the operator
- Download and install the dependent command line tools:

- oc - For interacting with the OpenShift Cluster
- cloud-pak-cli - For downloading and installing the CASE bundle

Note: Do all of the steps in the following procedure from the bastion machine.

Procedure

1. Download and extract the CASE bundle:
   a. Set up the environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment:

      ```
      export NS=<Namespace of target installation on OpenShift cluster>
      export CASE_NAME=ibm-db2uoperator
      export CASE_VERSION=<CASE version of latest Db2 Operator>
      export OFFLINECASE=/tmp/cases
      export OFFLINECASE=${OFFLINECASE}/${CASE_NAME}
      export CASEPATH="https://github.com/IBM/cloud-pak/raw/master/repo/case"
      export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

      # Details of the target registry to copy to
      export TARGET_REGISTRY_HOST=""          # Target registry host
      export TARGET_REGISTRY_PORT=5000        # Target registry port number
      export TARGET_REGISTRY=${TARGET_REGISTRY_HOST}:${TARGET_REGISTRY_PORT}
      export TARGET_REGISTRY_USER="user"      # Actual username goes here
      export TARGET_REGISTRY_PASSWORD="key"   # Actual API Key goes here
      ```

   b. Create a directory to save the CASE to a local directory:

      ```
      $ mkdir ${OFFLINECASE}
      ```

   c. Download and extract the CASE bundle:

      ```
      $ cloudctl case save --repo ${CASEPATH} --case ${CASE_NAME} --version ${CASE_VERSION} --
      outputdir ${OFFLINEDIR}
      Downloading and extracting the CASE ...
      - Success
      Retrieving CASE version ...
      - Success
      Validating the CASE ...
      - Success
      Creating inventory ...
      - Success
      Finding inventory items
      - Success
      Resolving inventory items ...
      Parsing inventory items
      - Success
      ```

   d. Verify the CASE and images csv has been downloaded:

      ```
      $ ls ${OFFLINECASE}
      total 128K
      drwxr-xr-x 2 root root    6 Jan 20 11:10 charts/
      -rw-r--r-- 1 root root 116K Jan 20 11:10 ibm-db2uoperator-1.0.x.tgz
      -rw-r--r-- 1 root root   32 Jan 20 11:10 ibm-db2uoperator-1.0.x-charts.csv
      -rw-r--r-- 1 root root 5.2K Jan 20 11:10 ibm-db2uoperator-1.0.x-images.csv
      ```

   e. Extract the CASE:

      ```
      cd ${OFFLINECASE}
      tar -xvzf ${CASE_ARCHIVE}
      ```

2. Configure your registry authentication secrets:
   a. Create an authentication secret for target image registry:

      ```
      $ cloudctl case launch              \
          --case ${OFFLINECASE}           \
          --namespace ${NS}               \
      ```

```
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${TARGET_REGISTRY} --user ${TARGET_REGISTRY_USER} --pass $
{TARGET_REGISTRY_PASSWORD}"

The credentials are now saved to `~/.airgap/secrets/<registry-name>.json`
```

3. Copy the images from your saved CASE (images.csv) to the target registry in the air-gapped cluster.

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action mirror-images          \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

4. Configure the air-gapped cluster to use its internal/target image registry:

Warning: Cluster resources must adjust to the new pull secret, which can temporarily limit the usability of the cluster. Authorization credentials are stored in `$HOME/.airgap/secrets` and `/tmp/airgap*` to support this action.

   a. Apply an image source content policy. Doing so causes each worker node to restart:

```
$ cloudctl case launch               \
    --case ${OFFLINECASE}            \
    --namespace ${NS}                \
    --inventory db2uOperatorSetup    \
    --action configure-cluster-airgap \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

   b. Add the target registry to the cluster **insecureRegistries** list if the target registry isn't secured by a certificate. Run the following command to restart all nodes, one at a time:

```
$ oc patch image.config.openshift.io/cluster --type=merge
-p "{\"spec\":{\"registrySources\":{\"insecureRegistries\":[\"${TARGET_REGISTRY_HOST}:$
{TARGET_REGISTRY_PORT}\", \"${TARGET_REGISTRY_HOST}\"]}}}"
```

5. Install the catalog source:

```
cloudctl case launch                       \
    --case ${OFFLINECASE}                  \
    --namespace openshift-marketplace      \
    --inventory db2uOperatorSetup          \
    --action installCatalog
```

6. Install the Db2 Operator:

```
cloudctl case launch                        \
     --case ${OFFLINECASE}                  \
     --namespace ${NS}                      \
     --inventory db2uOperatorStandaloneSetup \
     --action installOperator
```

7. Deploy Db2, using the Db2uCluster API.

Installing the Db2 Operator in an air-gapped OpenShift cluster with no bastion host

You can install the Db2 Operator to an air-gapped Red Hat OpenShift (RHOS) cluster by doing an on-site transfer of the required components from a portable device.

Before you begin

Prepare a portable device, such as a laptop, with the required Container Application Software for Enterprises (CASE) bundle.

- Verify that the portable device has access to the following:
  - public internet (to download CASE and images)
  - a target image registry (where the images will be mirrored)

  - a target OpenShift cluster onto which to install the operator
- Download and install dependent command line tools
  - oc - To interact with your OpenShift cluster
  - cloud-pak-cli - To download and install the CASE bundle

Note: Do all of the steps in the following procedure from the portable device.

Procedure

1. Download and extract the CASE bundle:
   a. Set up environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment.

      ```
      export NS=<Namespace of target installation on OpenShift cluster>
      export CASE_NAME=ibm-db2uoperator
      export CASE_VERSION=<CASE version of latest Db2 Operator>
      export OFFLINECASE=/tmp/cases
      export OFFLINECASE=${OFFLINECASE}/${CASE_NAME}
      export CASEPATH="https://github.com/IBM/cloud-pak/raw/master/repo/case"
      export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

      # Details of the target registry to copy to
      export TARGET_REGISTRY_HOST=""          # Target registry host
      export TARGET_REGISTRY_PORT=5000        # Target registry port number
      export TARGET_REGISTRY=${TARGET_REGISTRY_HOST}:${TARGET_REGISTRY_PORT}
      export TARGET_REGISTRY_USER="user"      # Actual username goes here
      export TARGET_REGISTRY_PASSWORD="key"   # Actual API Key goes here
      ```

   b. Create a directory to save the CASE to a local directory:

      ```
      $ mkdir ${OFFLINECASE}
      ```

   c. Download and extract the CASE bundle:

      ```
      $ cloudctl case save --repo ${CASEPATH} --case ${CASE_NAME} --version ${CASE_VERSION}
      --outputdir ${OFFLINEDIR}
      Downloading and extracting the CASE ...
      - Success
      Retrieving CASE version ...
      - Success
      Validating the CASE ...
      - Success
      Creating inventory ...
      - Success
      Finding inventory items
      - Success
      Resolving inventory items ...
      Parsing inventory items
      - Success
      ```

   d. Verify the CASE and images csv has been downloaded:

      ```
      $ ls ${OFFLINECASE}
      total 128K
      drwxr-xr-x 2 root root    6 Jan 20 11:10 charts/
      -rw-r--r-- 1 root root 116K Jan 20 11:10 ibm-db2uoperator-1.0.x.tgz
      -rw-r--r-- 1 root root   32 Jan 20 11:10 ibm-db2uoperator-1.0.x-charts.csv
      -rw-r--r-- 1 root root 5.2K Jan 20 11:10 ibm-db2uoperator-1.0.x-images.csv
      ```

   e. Extract the CASE:

      ```
      cd ${OFFLINECASE}
      tar -xvzf ${CASE_ARCHIVE}
      ```

2. Copy the images to the local container registry on the portable device:
   a. Set up environment variables.

Review the following parameters for your environment and then run the following commands to set up the environment:

```
export NS=<Namespace of target installation on OpenShift cluster>
export CASE_NAME=ibm-db2uoperator
export CASE_VERSION=<CASE version of latest Db2 Operator>
export OFFLINECASE=/tmp/cases
export OFFLINECASE=${OFFLINECASE}/${CASE_NAME}
export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

# Details of the intermediate registry if not using a Bastion server
export PORTABLE_REGISTRY_HOST=localhost
export PORTABLE_REGISTRY_PORT=5000
export PORTABLE_REGISTRY=${PORTABLE_REGISTRY_HOST}:${PORTABLE_REGISTRY_PORT}
export PORTABLE_REGISTRY_USER="user"      # Actual username goes here
export PORTABLE_REGISTRY_PASSWORD="key"   # Actual API Key goes here
export PORTABLE_REGISTRY_PATH=${OFFLINECASE}/registry
export PORTABLE_STORAGE_LOCATION=""       # Override

# Details of the target registry to copy to
export TARGET_REGISTRY_HOST=""          # Target registry host
export TARGET_REGISTRY_PORT=5000        # Target registry port number
export TARGET_REGISTRY=${TARGET_REGISTRY_HOST}:${TARGET_REGISTRY_PORT}
export TARGET_REGISTRY_USER="user"      # Actual username goes here
export TARGET_REGISTRY_PASSWORD="key"   # Actual API Key goes here
```

b. Set the target registry:

```
export TARGET_REGISTRY=${PORTABLE_REGISTRY}
export TARGET_REGISTRY_USER=${PORTABLE_REGISTRY_USER}
export TARGET_REGISTRY_PASS=${PORTABLE_REGISTRY_PASSWORD}
```

c. Initialize the Docker registry by running the following command:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action init-registry        \
    --args "--registry $PORTABLE_REGISTRY_HOST --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

d. Start the Docker registry by running the following command:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action start-registry       \
    --args "--registry $PORTABLE_REGISTRY --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

3. Configure your registry authentication secrets:

a. Create an authentication secret for target image registry:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${TARGET_REGISTRY} --user ${TARGET_REGISTRY_USER} --pass $
{TARGET_REGISTRY_PASSWORD}"

The credentials are now saved to `~/.airgap/secrets/<registry-name>.json`
```

4. Copy the images from your saved CASE (images.csv) to the target registry in the air-gapped cluster:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action mirror-images          \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

5. Copy the offline case inventory images and registry data folder to the portable storage device:

```
cp -r ${OFFLINECASE} ${PORTABLE_STORAGE_LOCATION}
```

6. Copy the images to the target registry behind the firewall:

   a. Set up environment variables.

   Review the following parameters for your environment and then run the following commands to set up the environment:

```
export NS=<Namespace of target installation on OpenShift cluster>
export CASE_NAME=ibm-db2uoperator
export CASE_VERSION=<CASE version of latest Db2 Operator>
export OFFLINECASE=/tmp/cases
export OFFLINECASE=${OFFLINECASE}/${CASE_NAME}
export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

# Details of the intermediate registry if not using a Bastion server
export PORTABLE_REGISTRY_HOST=localhost
export PORTABLE_REGISTRY_PORT=5000
export PORTABLE_REGISTRY=${PORTABLE_REGISTRY_HOST}:${PORTABLE_REGISTRY_PORT}
export PORTABLE_REGISTRY_USER="user"      # Actual username goes here
export PORTABLE_REGISTRY_PASSWORD="key"   # Actual API Key goes here
export PORTABLE_REGISTRY_PATH=${OFFLINECASE}/registry
export PORTABLE_STORAGE_LOCATION=""       # Override

# Details of the target registry to copy to
export INTERNAL_REGISTRY_HOST=""          # Target registry host
export INTERNAL_REGISTRY_PORT=5000        # Target registry port number
export INTERNAL_REGISTRY=${INTERNAL_REGISTRY_HOST}:${INTERNAL_REGISTRY_PORT}
export INTERNAL_REGISTRY_USER="user"      # Actual username goes here
export INTERNAL_REGISTRY_PASSWORD="key"   # Actual API Key goes here
```

   b. Set the source and target registries.

   The source container registry is now the local registry on the portable device, for example **localhost:5000** and the destination is the registry behind the firewall, for example **10.10.4.6:5000**, or the host and port in your air-gap environment. You need to set up the environment variables, mirror the images, and then install the catalog.

   Set up the environment variables:

```
export SOURCE_REGISTRY=${PORTABLE_REGISTRY}
export SOURCE_REGISTRY_USER=${PORTABLE_REGISTRY_USER}
export SOURCE_REGISTRY_PASS=${PORTABLE_REGISTRY_PASSWORD}

export TARGET_REGISTRY=${INTERNAL_REGISTRY}
export TARGET_REGISTRY_USER=${INTERNAL_REGISTRY_USER}
export TARGET_REGISTRY_PASS=${INTERNAL_REGISTRY_PASSWORD}
```

   Override the registry storage location to point to the location of the portable storage:

```
export PORTABLE_STORAGE_LOCATION=#Provide external storage path here
```

   Copy the offline case inventory images and registry data folder from the portable storage device to the node.

```
cp -r ${PORTABLE_STORAGE_LOCATION} ${OFFLINECASE}
```

   c. Initialize the Docker registry:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action init-registry        \
    --args "--registry $PORTABLE_REGISTRY_HOST --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

   d. Start the Docker registry by running the following command:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action start-registry       \
```

```
    --args "--registry $PORTABLE_REGISTRY --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

7. Configure your registry authentication secrets:
   a. Create an authentication secret for the source image registry:

      Note: If the registry is public, which doesn't require credentials, skip this step.

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${SOURCE_REGISTRY} --user ${SOURCE_REGISTRY_USER} --pass $
{SOURCE_REGISTRY_PASSWORD}"
```

   b. Create an authentication secret for target image registry:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${TARGET_REGISTRY} --user ${TARGET_REGISTRY_USER} --pass $
{TARGET_REGISTRY_PASSWORD}"
```

8. Copy the images from the saved CASE (images.csv) to the target registry in the air-gap environment:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action mirror-images          \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

9. Configure the air-gapped cluster to use its internal/target image registry:

   Warning: Cluster resources must adjust to the new pull secret, which can temporarily limit the usability of the cluster. Authorization credentials are stored in `$HOME/.airgap/secrets` and `/tmp/airgap*` to support this action.

   a. Apply an image source content policy. Doing so causes each worker node to restart:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-cluster-airgap \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

   b. Add the target registry to the cluster **insecureRegistries** list if the target registry isn't secured by a certificate. Run the following command to restart all nodes, one at a time:

```
$ oc patch image.config.openshift.io/cluster --type=merge
-p "{\"spec\":{\"registrySources\":{\"insecureRegistries\":[\"${TARGET_REGISTRY_HOST}:$
{TARGET_REGISTRY_PORT}\", \"${TARGET_REGISTRY_HOST}\"]}}}"
```

10. Install the catalog source:

```
cloudctl case launch                    \
    --case ${OFFLINECASE}               \
    --namespace openshift-marketplace   \
    --inventory db2uOperatorSetup       \
    --action installCatalog
```

11. Install the Db2 Operator:

```
cloudctl case launch                     \
     --case ${OFFLINECASE}               \
     --namespace ${NS}                   \
     --inventory db2uOperatorStandaloneSetup \
     --action installOperator
```

12. Deploy Db2, using the Db2uCluster API.

Watching multiple namespaces from a single namespace installation

With the CN5 release of Db2 11.5.7, you can now choose to watch additional namespaces from a single namespace installation of the Db2 Operator.

About this task

Prior to the 11.5.7-cn5 release of Db2, users had two deployment options for the Db2 Operator:

- Installation at the cluster level, to view all namespaces. This option allows you to create Db2 instances in any namespace within your cluster
- Installation at the namespace level, to view a single namespace. This limits your control to a single namespace.

With the release of Db2 11.5.7-cn5, you can configure a ConfigMap to watch namespaces outside of the one where the Db2 Operator is installed.

Procedure

1. From your OpenShift console, create a ConfigMap named *namespace-scope* in the db2u-operator namespace.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: namespace-scope
  namespace: <db2u_operator_ns>
data:
  namespaces: '<db2u_operator_ns>,<target_ns_1>,<target_ns_2>'
```

2. Edit the OperatorGroup resource `db2u-operator-group`. By adding the target namespaces to this resource, you give the db2u-operator permissions in the target namespaces.

```
oc edit OperatorGroup db2u-operator-group -n ${NAMESPACE}
```

   where NAMESPACE is the location of the installed db2u-operator.

   Edit the `spec.targetNamespaces` field by adding the namespaces to be watched.

```
 targetNamespaces:
    - db2u-operator-ns
    - <target-ns-1>
    - <target-ns-2>
```

3. Save your changes and exit.

   You can now see the target namespaces from your Db2 Operator namespace.

Deploying Db2 on your OpenShift cluster

When you have installed the Db2 Operator to your OpenShift cluster, you use the **db2uCluster** API to deploy Db2. Before doing so, you also need to set up any dedicated nodes within your cluster, accept the license terms, and configure your database storage.

Accepting the Db2 on OpenShift license terms

In order to successfully deploy Db2, the license terms detailed inside the Db2UCluster API need to be reviewed and accepted inside the Db2 Custom Resource.

Storage requirements

The storage class must exist in the cluster or a supported storage class must be provided accordingly. To deploy Db2, you need a supported storage class.

During a Db2 deployment, storage can be dynamically created or pre-created PVs can be specified. Db2 needs the following storage locations:

- System & Backup storage [Shared with RWX]
  - Db2 instance home directory
  - Diagnostic logs
  - Other global configuration directories
  - Backups, Restore or Load locations
- User storage [Exclusive with RWO]
  - Database storage paths
  - Transaction logs

Software-defined	Shared Zone [Meta]	Data Zone [Data]
NFS	Access Mode: RWX	RWX (combined with Meta) or RWO
Portworx 2.7	Shared v4, RWX (based on NFS v4 protocol)	io-profile: db_remote, RWO
OCS 4.6	CephFS, RWX	CephRBD(Block Device), RWO
Spectrum Scale CSI 2.1 or greater	RWX	RWO

Note: In cases where the storage layer supports it, a single storage location, defined as RWX, can be specified. Such a configuration would exhibit degraded performances.

PodSecurityPolicy Requirements

The Db2 deployment is currently only supported on Red Hat OpenShift.

SecurityContextConstraints Requirements

The Db2 deployment requires the following SCC:

```
kind: SecurityContextConstraints
apiVersion: v1
apiGroup: security.openshift.io
metadata:
    name: db2u-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
# privileged container is only needed for the init container that sets the Db2 kernel parameters
allowPrivilegedContainer: true
allowedCapabilities:
- "SYS_RESOURCE"
- "IPC_OWNER"
- "SYS_NICE"
- "CHOWN"
- "DAC_OVERRIDE"
- "FSETID"
- "FOWNER"
- "SETGID"
- "SETUID"
- "SETFCAP"
- "SETPCAP"
- "SYS_CHROOT"
- "KILL"
- "AUDIT_WRITE"
priority: 10
runAsUser:
    type: RunAsAny
```

```
seLinuxContext:
    type: MustRunAs
fsGroup:
    type: RunAsAny
supplementalGroups:
    type: RunAsAny
version: v1
```

Setting up dedicated nodes for your Db2 on Red Hat OpenShift deployment

You can dedicate one or more worker nodes to your Db2 on Red Hat OpenShift database service. The nodes are then used exclusively by the database containers or pods.

About this task

Db2 on Red Hat OpenShift uses the concepts of *taint* and *toleration* to dedicate Db2 to particular nodes. Nodes are tainted and labeled:

- A taint prevents pods from being deployed on a tainted node.
- A label allows for a pod with a matching toleration to be treated as an exception.

Taint and toleration work together to provide node exclusivity to deploying Db2 on particular nodes

Perform these steps for each worker node that you plan to dedicate to the database deployment. The steps use Db2 on Red Hat OpenShift CLI commands. The *node_name* is the name of a worker node on which you plan to host Db2.

Considerations for software-defined storage (such as OpenShift Container Storage)

When storage and compute share the same nodes, some additional considerations apply so that Db2 and software-defined storage can be properly scheduled on the same worker nodes. See the optional Step "2" on page 14 below.

Procedure

1. Retrieve the name of the worker node that you want to dedicate to Db2:

   ```
   oc get nodes
   ```

2. Optional: If you are using OpenShift Container Storage, see the OpenShift documentation for information about dedicating nodes that is specific to that platform.
3. Taint the node with the `NoSchedule` effect and safely evict all of the pods from that node:

   ```
   oc adm taint node node_name icp4data=dedicated_specifier:NoSchedule --overwrite
   oc adm drain node_name
   oc adm uncordon node_name
   ```

   Where *dedicated_specifier* is an identifier that is used to deploy a database only on nodes that have the *dedicated_specifier* label. By default, the *dedicated_specifier* for Db2 is `database-db2oltp`. The `--overwrite` flag is only required if the node is already tainted or labeled and ensures that the taint or label is replaced.
4. Label the node:

   ```
   oc label node node_name icp4data=dedicated_specifier --overwrite
   ```

5. Optional: Verify that the node is labeled:

   ```
   oc get node --show-labels
   ```

Deploying Db2 with limited privileges

You can improve security by running the Db2 container on Red Hat OpenShift with limited privileges.

Changing node settings by using the Node Tuning Operator

You can use the Red Hat OpenShift Node Tuning Operator to set interprocess communication (IPC) kernel parameters if you want to run Db2 on OpenShift with limited privileges.

Before you begin

Decide whether you plan to deploy the services on dedicated nodes. With dedicated nodes, you can limit node tuning to the nodes where the service or services will run.

For more information about setting up dedicated nodes, see Setting up dedicated nodes for your Db2 deployment.

About this task

Complete this task if you plan to install Db2 on an environment where you want to run Db2 with limited privileges.

- What is the Node Tuning Operator?

  The Node Tuning Operator helps you manage node-level tuning by orchestrating the tuned daemon. Tuned is a system tuning service for Linux®. The core of Tuned are profiles, which tune your system for different use cases. In addition to static application of system settings, Tuned can also monitor your system and optimize the performance on-demand based on the profile that is applied.

  Tuned is distributed with a number of predefined profiles. However, it is also possible to modify the rules defined for each profile and customize how and what to tune. Tuned supports various types of system configuration such as sysctl, sysfs, and kernel boot parameters. For more information, see Monitoring and managing system status and performance and The Tuned Project

  The Node Tuning Operator provides a unified management interface to users of node-level **`sysctls`** and gives more flexibility to add custom tuning.

  The operator manages the containerized tuned daemon for Red Hat OpenShift Container Platform as a Kubernetes DaemonSet. It ensures the custom tuning specification is passed to all containerized tuned daemons that run in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

  The Node Tuning Operator is part of a standard Red Hat OpenShift Container Platform installation. For more information, see the Red Hat OpenShift documentation:

  - Red Hat OpenShift version 4.6
  - Red Hat OpenShift version 4.8
  - Red Hat OpenShift version 4.10

Procedure

You can employ the Node Tuning Operator by using one of the following methods:

Creating a custom resource file
The custom resource method requires you to manually compute all required IPC kernel parameters.

Creating a shell script
The shell script enables you to generate a YAML file that you can install, deploy, and run on the target OpenShift cluster.

Creating a custom resource file

1. Create a Tuned custom resource file.

   Use the following guidance to adjust the contents of the custom resource file:

   - You must compute the values for the IPC kernel parameters that are denoted by < ...>. Use the formulas in Kernel parameter requirements (Linux).

- Use the `memory.resource` limit that you plan to apply to the deployment as `size of RAM` if your Kubernetes worker node pool is heterogeneous.
- The match label `icp4data` and the corresponding value is only required for dedicated deployments. In that case the IPC kernel tuning is applied only on the labeled worker nodes.
- The inheritance option, `include=openshift-node`, implements the following inheritance chain: `openshift-db2u-ipc <- openshift-node <- openshift <- virtual-host`.

  The custom resource injects the IPC sysctl changes on top of the default tuned profile settings on the OpenShift worker nodes.

The following sample YAML file describes the basic structure that is needed to create the custom resource for a Node Tuning Operator instance that can tune IPC kernel parameters.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: db2u-ipc-tune
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - name: openshift-db2u-ipc
    data: |
      [main]
      summary=Tune IPC Kernel parameters on OpenShift nodes running Db2U engine PODs
      include=openshift-node

      [sysctl]
      kernel.shmmni = <shmmni>
      kernel.shmmax = <shmmax>
      kernel.shmall = <shmall>
      kernel.sem = <SEMMSL> <SEMMNS> <SEMOPM> <SEMMNI>
      kernel.msgmni = <msgmni>
      kernel.msgmax = <msgmax>
      kernel.msgmnb = <msgmnb>

  recommend:
  - match:
    - label: node-role.kubernetes.io/worker
    - label: icp4data
      value: database-db2wh
    priority: 10
    profile: openshift-db2u-ipc
```

2. Save the custom resource as a YAML file. For example: `/tmp/Db2UnodeTuningCR.yaml`.
3. Log in to the cluster as a cluster administrator and create the custom resource:

```
oc create -f /tmp/Db2UnodeTuningCR.yaml
```

It might take a few minutes for the custom resource to be created and for the custom IPC tuned profile to be applied on the worker nodes.

Creating a shell script

The following sample shell script can be used to:

- Generate a YAML file that you can install and deploy and run on the target OpenShift cluster.
- Delete the custom resource and clean up deployed tuned profiles.

The sample assumes that the script is saved as `/root/script/crtNodeTuneCR.sh`.

```
#!/bin/bash

# Compute IPC kernel parameters as per IBM Documentation topic
# https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.db2.luw.qb.server.doc/doc/
c0057140.html
# and generate the Node Tuning Operator CR yaml.

tuned_cr_yaml="/tmp/Db2UnodeTuningCR.yaml"
mem_limit_Gi=0
node_label=""
cr_name="db2u-ipc-tune"
cr_profile_name="openshift-db2u-ipc"
```

```
cr_namespace="openshift-cluster-node-tuning-operator"
create_cr="false"
delete_cr="false"

usage() {
    cat <<-USAGE #| fmt
    Usage: $0 [OPTIONS] [arg]

    OPTIONS:
    =======
    * -m|--mem-limit mem_limit  : The memory.limit (Gi) to be applied to Db2U deployment.
    * [-l|--label node_label]   : The node label to use for dedicated Cp4D deployments.
    * [-f|--file yaml_output]   : The NodeTuningOperator CR YAML output file. Default /tmp/
Db2UnodeTuningCR.yaml.
    * [-c|--create]             : Create the NodeTuningOperator CR ${cr_name} using the
generated CR yaml file.
    * [-d|--delete]             : Delete the NodeTuningOperator CR ${cr_name}.
    * [-h|--help]               : Display the help text of the script.
USAGE
}

[[ $# -lt 1 ]] && { usage && exit 1; }

while [[ $# -gt 0 ]]; do
    case "$1" in
        -f|--file) shift; tuned_cr_yaml=$1
        ;;
        -m|--mem-limit) shift; mem_limit_Gi=$1
        ;;
        -l|--label) shift; node_label=$1
        ;;
        -c|--create) create_cr="true"
        ;;
        -d|--delete) delete_cr="true"
        ;;
        -h|--help) usage && exit 0
        ;;
        *) usage && exit 1
        ;;
    esac
    shift
done

((ram_in_BYTES=mem_limit_Gi * 1073741824))
((ram_GB=ram_in_BYTES / (1024 * 1024 * 1024)))
((IPCMNI_LIMIT=32 * 1024))
tr ' ' '\n' < /proc/cmdline | grep -q ipcmni_extend && ((IPCMNI_LIMIT=8 * 1024 * 1024))

#
### ============== functions ================ ###
#
# Compute the required kernel IPC parameter values
compute_kernel_ipc_params() {
    local PAGESZ=$(getconf PAGESIZE)

    # Global vars
    ((shmmni=256 * ram_GB))
    shmmax=${ram_in_BYTES}
    ((shmall=2 * (ram_in_BYTES / PAGESZ)))
    ((msgmni=1024 * ram_GB))
    msgmax=65536
    msgmnb=${msgmax}
    SEMMSL=250
    SEMMNS=256000
    SEMOPM=32
    SEMMNI=${shmmni}

    # RH bugzilla https://access.redhat.com/solutions/4968021. Limit SEMMNI, shmmni and msgmni
to the max
    # supported by the Linux kernel -- 32k (default) or 8M if kernel boot parameter
'ipcmni_extend' is set.
    ((SEMMNI=SEMMNI < IPCMNI_LIMIT ? SEMMNI : IPCMNI_LIMIT))
    ((shmmni=shmmni < IPCMNI_LIMIT ? shmmni : IPCMNI_LIMIT))
    ((msgmni=msgmni < IPCMNI_LIMIT ? msgmni : IPCMNI_LIMIT))
}

# Generate NodeTuning Operator YAML file
gen_tuned_cr_yaml() {
    # Generate YAML file for NodeTuning CR and save as ${tuned_cr_yaml}
    cat <<-EOF > ${tuned_cr_yaml}
apiVersion: tuned.openshift.io/v1
kind: Tuned
```

```
metadata:
  name: ${cr_name}
  namespace: ${cr_namespace}
spec:
  profile:
  - name: ${cr_profile_name}
    data: |
      [main]
      summary=Tune IPC Kernel parameters on OpenShift nodes running Db2U engine PODs
      include=openshift-node

      [sysctl]
      kernel.shmmni = ${shmmni}
      kernel.shmmax = ${shmmax}
      kernel.shmall = ${shmall}
      kernel.sem = ${SEMMSL} ${SEMMNS} ${SEMOPM} ${SEMMNI}
      kernel.msgmni = ${msgmni}
      kernel.msgmax = ${msgmax}
      kernel.msgmnb = ${msgmnb}

  recommend:
  - match:
    - label: node-role.kubernetes.io/worker
EOF

    # Add the optional dedicated label into match array
    if [[ -n "${node_label}" ]]; then
        cat <<-EOF >> ${tuned_cr_yaml}
    - label: icp4data
      value: ${node_label}
EOF
    fi

    # Add the priority and profile keys
    cat <<-EOF >> ${tuned_cr_yaml}
    priority: 10
    profile: ${cr_profile_name}
EOF

    [[ "${create_cr}" == "true" ]] && return
    cat <<-MSG
================================================================================
* Successfully generated the Node Tuning Operator Custom Resource Definition as
  ${tuned_cr_yaml} YAML with Db2U specific IPC sysctl settings.

* Please run 'oc create -f ${tuned_cr_yaml}' on the master node to
  create the Node Tuning Operator CR to apply those customized sysctl values.
================================================================================
MSG
}

create_tuned_cr() {
    echo "Creating the Node Tuning Operator Custom Resource for Db2U IPC kernel parameter
tuning ..."
    oc create -f ${tuned_cr_yaml}
    sleep 2

    # List the NodeTuning CR and describe
    oc -n ${cr_namespace} get Tuned/${cr_name}
    echo ""

    echo "The CR of the Node Tuning Operator deployed"
    echo "-------------------------------------------"
    oc -n ${cr_namespace} describe Tuned/${cr_name}
    echo ""
}

delete_tuned_cr() {
    echo "Deleting the Node Tuning Operator Custom Resource used for Db2U IPC kernel parameter
tuning ..."
    oc -n ${cr_namespace} get Tuned/${cr_name} --no-headers -ojsonpath='{.kind}' | grep -iq
tuned || \
        { echo "No matching CR found ..." && exit 0; }
    oc -n ${cr_namespace} delete Tuned/${cr_name}
    echo ""
    sleep 2

    # Get the list of containerized tuned PODs (DaemonSet) deployed on the cluster
    local tuned_pods=( $(oc -n ${cr_namespace} get po --selector openshift-app=tuned --no-
headers -ojsonpath='{.items[*].metadata.name}') )
    # Remove the tuned profile directory deployed on those PODs
    for p in "${tuned_pods[@]}"; do
```

```
        echo "Removing the installed tuned profile ${cr_profile_name} on POD: $p"
        oc -n ${cr_namespace} exec -it $p -- bash -c "rm -fr /etc/tuned/${cr_profile_name}"
    done
    echo ""
}

#
### ================== Main =================== ###
#

[[ "${delete_cr}" == "true" ]] && { delete_tuned_cr && exit 0; }

compute_kernel_ipc_params

gen_tuned_cr_yaml

[[ "${create_cr}" == "true" ]] && create_tuned_cr
```

What to do next

After you install Db2, complete Configuring Db2 to be deployed with limited privileges.

Configuring Red Hat OpenShift to set IPC kernel parameters

In a limited privileged deployment of Db2, you must enable unsafe sysctls to meet the enforced minimum values for interprocess communication (IPC) kernel parameters.

Procedure

Follow the procedure Enabling unsafe sysctls in the OpenShift documentation.

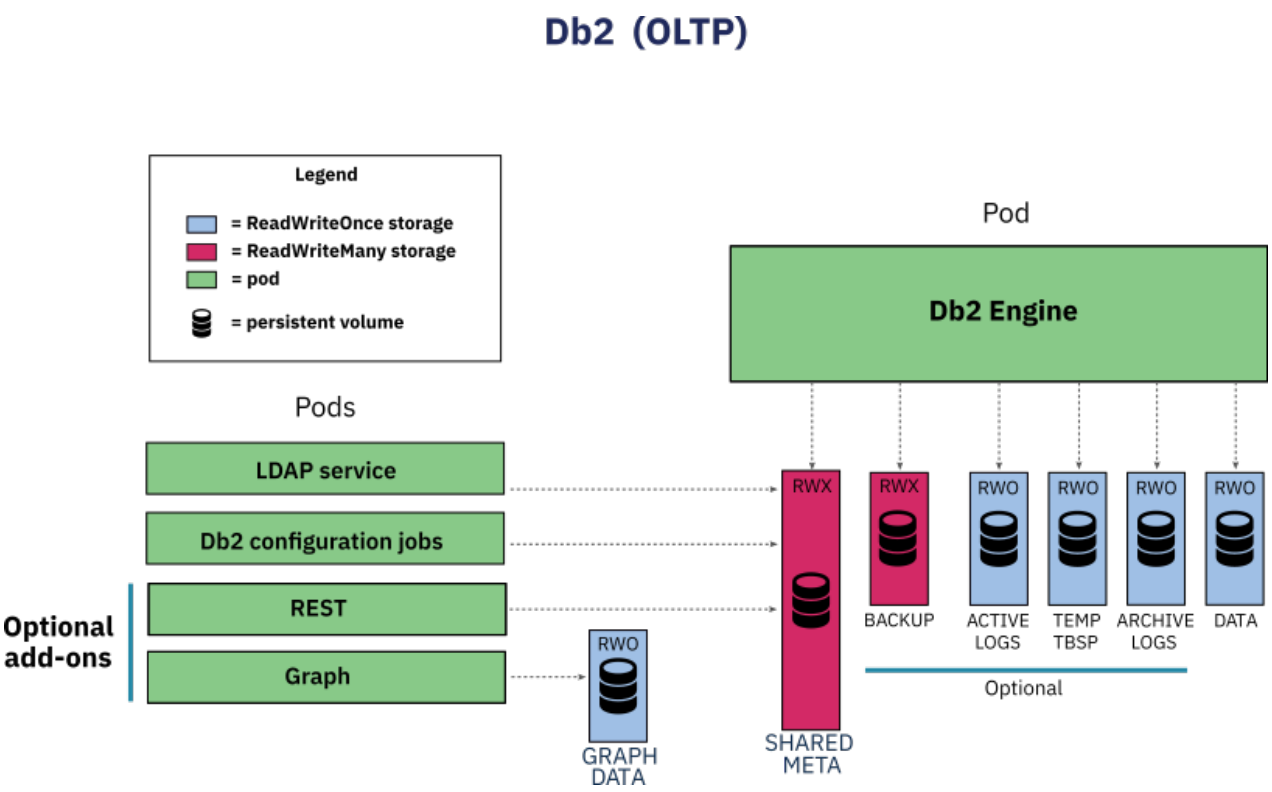
Replace the example in Step 2 for creating a KubeletConfig Custom Resource with the following values:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl
  kubeletConfig:
    allowedUnsafeSysctls:
      - "kernel.msg*"
      - "kernel.shm*"
      - "kernel.sem"
```

Configuring database storage for Db2 on OpenShift

Db2 supports a variety of both traditional and software-defined storage types. You define your storage types during database provisioning.

The following diagram shows the supported storage types.



For a full listing of supported storage types, see “Certified storage options for Db2” on page 20.

Notes:

- When provisioning the database, for user data (the main database data), select ReadWriteOnce (RWO) access mode for your storage.
- Select 4K sector size during provisioning if your storage supports 4K (this setting is mandatory for Portworx). Db2 also supports 4K for OpenShift Container Storage.
- Appropriate host-attached SAN storage is usually the better performing option for OLTP workloads. For a comparison of NAS versus SAN storage, see Storage technologies for IBM Db2 Warehouse.
- If you are using Security Enhanced Linux, see one of the following topics:
  - hostPath: Labeling hostPath storage for SELinux
  - NFS: NFS storage requirements for Db2
- For more information about configuring dynamic provisioning for NFS storage for testing, see the External Storage topic in the `kubernetes-incubator` repository on GitHub.
- For more information about configuring a hostPath for data storage, see HostPathVolumeSource in the Red Hat OpenShift documentation.

Certified storage options for Db2

Db2 on Red Hat OpenShift provides a broad range of certified storage options.

Certified storage options for Db2 shows the certified storage options and their deployment requirements.

Table 3. Certified storage options for Db2

Certified storage	Deployment requirements
IBM Spectrum® Scale Container Storage Interface 2.0 or higher For more information, see IBM Spectrum Scale.	**Split storage:** **System/backup data** • ibm-spectrum-scale-csi storage class • ReadWriteMany (RWX) **User data** • ibm-spectrum-scale-csi storage class • 4K sector size • ReadWriteOnce (RWO)
Portworx 2.7	**Split storage:** **System/backup data** • portworx-db2-rwx-sc storage class • ReadWriteMany (RWX) **User data** • portworx-db2-rwo-sc storage class • You must specify 4K block size. • ReadWriteOnce (RWO)
OpenShift Container Storage 4.6 For more information, see the OpenShift documentation.	**Split storage:** **System/backup data** • ocs-storagecluster-cephfs storage class • ReadWriteMany (RWX) **User data** • ocs-storagecluster-ceph-rbd storage class • 4K sector size • ReadWriteOnce (RWO)
IBM® Cloud File Storage	ibmc-file-gold-gid storage class
Traditional storage: • NFS • Local storage	• "NFS storage requirements for Db2" on page 21 • "Requirements for Db2 on SELinux" on page 22

NFS storage requirements for Db2

The Db2 service has several prerequisites for NFS storage.

Configuring NFS with Dell EMC Isilon

Here is a sample configuration to deploy Db2 with Dell EMC Isilon.

Disable root and non-root mapping

You must configure a mapping for the NFS export options. Specify both **Do not map root users** and **Do not map non root users**.

Mount access to subdirectories

Specify **Enable mount access to subdirectories**.

NFS storage class

The following example shows a recommended NFS storage class for Dell EMC Isilon with the correct mount options:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-nfs-storage
mountOptions:
- v3
- nolock
parameters:
  archiveOnDelete: "false"
provisioner: fuseim.pri/ifs
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

Note: When using Dell EMC Isilon, you must set the NFS version to v3.

Configuring NFS with Trident

You can configure Db2 NFS storage on Red Hat OpenShift to use the Trident provisioner from NetApps.

About this task

For more information on setting up Trident, see these pages:

- Kubernetes StorageClass objects
- Backend configuration options
- Securing NFS access using export policies

Procedure

1. Add the mount options that are described in Requirements for NFS 4.2.
2. Define an ONTAP-NFS export policy rule to:
   - Include the Red Hat OpenShift cluster domain as the `client identifier`.
   - Set Superuser security types: `any`.
3. Create a back-end configuration that is associated with this export policy rule.
4. Create a storage class to use this specific back-end configuration.

Requirements for Db2 on SELinux

Before you deploy a Db2 service on Security Enhanced Linux (SELinux) in enforcing mode, additional steps are required for setting up persistent storage.

Some functionality, such as support for Db2 external table load, use FIFO pipes and need the following SELinux modules in order to allow for the correct file permissions.

Requirements for NFS 4.2

NFS 4.2 supports volume relabeling on the client-side mount. You can specify the mount options either in the storage class if you are dynamically provisioning the volume, or in the persistent volume if you are manually provisioning the volume.

The following key and values must be added to the storage definition that you are using for Db2:

```
mountOptions:
- v4.2
- context="system_u:object_r:container_file_t:s0"
```

- If you are using a persistent volume for Db2 storage, add these values under the spec key of the PersistentVolume object.
- If you are using a storage class for Db2 storage, add the values as a top-level key of the StorageClass object.

Tip: To enable 4.2 on your NFS server or to check whether it is enabled, see How to enable NFS v4.2 on RHEL7.

Requirements for NFS 4.1 and earlier
For NFS 4.1 and earlier you must create an SELinux policy module and install it on all of the nodes that are hosting Db2.

1. Add the following code to the db2u-nfs.te file and save the file to your desired location:

```
module db2u-nfs 1.0;
require {
  type nfs_t;
  type container_t;
  class fifo_file { create open read unlink write ioctl getattr setattr };
}
allow container_t nfs_t:fifo_file { create open read unlink write ioctl getattr setattr };
```

2. Transform the db2u-nfs.te file into the db2u-nfs.mod module file by following these steps:

   a. Ensure that the checkpolicy package is installed. If the package is not installed you will get this error:

   ```
   -bash: checkmodule: command not found
   ```

   To install the package, run this command:

   ```
   sudo yum install -y checkpolicy
   ```

   b. Run the following command to transform the db2u-nfs.te file:

   ```
   checkmodule -M -m -o db2u-nfs.mod db2u-nfs.te
   ```

3. Compile the module file db2u-nfs.mod into the policy package file db2u-nfs.pp:

```
semodule_package -o db2u-nfs.pp -m db2u-nfs.mod
```

4. Install the policy package:

```
semodule -i db2u-nfs.pp
```

5. To confirm that the package was installed, you can run the following command:

```
semodule -l
```

The SE Linux module db2u-nfs should be shown.

Note: Only Step 4 is necessary to install the SELinux policy. You can perform the first three steps on a single node and transfer the policy package file db2u-nfs.pp to all of the other nodes and install it.

Requirements for OpenShift Container Storage on SELinux
To use OpenShift Container Storage, you must create an SELinux policy module and install it on all of the nodes that are hosting Db2.

1. Add the following code to the db2u-cephfs.te file and save the file to your desired location:

```
module db2u-cephfs 1.0;
require {
type cephfs_t;
type container_t;
class fifo_file { create open read unlink write ioctl getattr setattr };
}
#============= container_t ==============
allow container_t cephfs_t:fifo_file { create open read unlink write ioctl getattr setattr };
```

2. Transform the db2u-cephfs.te file into the db2u-cephfs.mod module file:

```
# checkmodule -M -m -o db2u-cephfs.mod db2u-cephfs.te
```

3. Compile the module file db2u-cephfs.mod into the policy package file db2u-cephfs.pp:

```
# semodule_package -o db2u-cephfs.pp db2u-cephfs.mod
```

4. Install the policy package:

```
# semodule -i db2u-cephfs.pp
```

5. To confirm that the package was installed, you can run the following command:

```
semodule -l
```

   The SE Linux module db2u-cephfs should be shown.

Note: Only Step 4 is necessary to install the SELinux policy. You can perform the first three steps on a single node and transfer the policy package file db2u-nfs.pp to all of the other nodes and install it.

Requirements for Portworx on SELinux
To use Portworx, you must create an SELinux policy module and install it on all of the nodes that are hosting Db2.

1. Add the following code to the db2u-nfs.te file and save the file to your desired location:

```
module db2u-nfs 1.0;

require {
  type nfs_t;
  type container_t;
  class fifo_file { create open read unlink write ioctl getattr setattr };
}
allow container_t nfs_t:fifo_file { create open read unlink write ioctl getattr setattr };
```

2. Transform the db2u-nfs.te file into the db2u-nfs.mod module file:

```
# checkmodule -M -m -o db2u-nfs.mod db2u-nfs.te
```

3. Compile the module file db2u-nfs.mod into the policy package file db2u-nfs.pp:

```
# semodule_package -o db2u-nfs.pp -m db2u-nfs.mod
```

4. Install the policy package:

```
# semodule -i db2u-nfs.pp
```

5. To confirm that the package was installed, you can run the following command:

```
semodule -l
```

   The SE Linux module db2u-nfs should be shown.

Note: Only Step 4 is necessary to install the SELinux policy. You can perform the first three steps on a single node and transfer the policy package file db2u-nfs.pp to all of the other nodes and install it.

Labeling hostPath storage for SELinux
If you are using hostPath storage with Security Enhanced Linux (SELinux) in enforcing mode, you must label the storage path on all nodes where your Red Hat OpenShift database service runs.

Run the following commands to apply the container_file_t SELinux label to the storage path on all nodes. For dedicated deployments, run the commands on all labeled nodes. For non-dedicated deployments, run them on all worker nodes.

```
# chmod 777 storage_path
# semanage fcontext -a -t container_file_t "storage_path(/.*)?"
# restorecon -Rv storage_path
```

Replace *storage_path* with the path to your persistent volume.

Configuring storage for comparable performance with Db2 on-premises

You can deploy Db2 to use a local volume for its user data to achieve performance on Red Hat OpenShift that is comparable with Db2 on-premises.

About this task

By default, the Db2 transaction log and database partition path are saved under the user storage location, which might use OpenShift Container Storage, Portworx, or some other storage technology where network latency can slow performance, especially for large-volume workloads such as those incurred during loading operations.

In order to achieve performance that is similar to an on-premises deployment, follow these steps to deploy Db2 by using a local volume for its user data (the main database data).

Note: Local volumes are subject to the availability of the underlying node. If the node that is hosting the volume goes down, application downtime will result until the node comes back online.

Procedure

1. On your OpenShift cluster, taint and label a node that uses a local disk. You will deploy Db2 on this node.
2. If you are using hostPath, run the following OpenShift command to add a cluster-role policy for the zen-databases-sa service account to allow use of local persistent volumes:

   ```
   oc adm policy add-cluster-role-to-user system:controller:persistent-volume-binder
   system:serviceaccount:project:zen-databases-sa
   ```

   Where *project* is the OpenShift project for the `zen-database-sa` service account.
3. Create the local storage path and label it for SELinux on the node where you will deploy Db2.
4. In the Red Hat OpenShift web console, when deploying Db2 click **Deploy database on dedicated nodes** and enter the node label from Step 1 in the **Value for node label** box.
5. In the console storage area, use one of the following options to specify a local volume for user storage:

Option	Steps
Local persistent volume storage class	In the web console, specify **User storage** > **Create new storage** > **User storage template** > **Storage class**. For an example of a storage class that targets a defined local volume, see Persistent storage using local volumes in the OpenShift documentation. The associated `storageClassName` is created and can be specified as the storage class to target your local volumes. Whether you choose to use a disk directly, or create logical volumes, the settings `volumeMode: Filesystem` and `accessModes: ReadWriteOnce` are required. This option is most likely the easiest to manage.
Self-created PVC	Specify **User storage** > **Use existing claim**. With this option, you must ensure that the PVC targets the local storage path on the node. Some examples of the existing claim are a PVC of type hostPath, or an NFS PVC that points to the dedicated node with a hostPath. You can also create a `LocalVolume` PVC. When using local volumes, you can install the Local Storage Operator to help you create the PVs in your definition.
hostPath	Specify **User storage** > **Create new storage** > **Storage volume type** > **hostPath** This option creates a hostPath persistent volume claim (PVC) that uses the host path that you specify as a persistent volume.

Option	Steps
NFS	Specify **User storage** > **Create new storage** > **Storage volume type** > **NFS** This option creates an NFS PVC that uses the NFS server IP address (specify the same address as the dedicated node). For **NFS server path**, specify the host path.

6. Optional: You can also use logical volume management (LVM) devices or a volume group (VG) to improve local storage performance. The following example includes the necessary steps:

```
# Commands to run on corresponding dedicated worker nodes. In this example we will be
referencing two local disks to create a Volume Group and Logical Volume
LOCAL_DISK_DEVICE_1=""
LOCAL_DISK_DEVICE_2=""
LOCAL_DISK_1="/dev/${LOCAL_DISK_DEVICE_1}"
LOCAL_DISK_2="/dev/${LOCAL_DISK_DEVICE_2}"

VG=db2u_vg
LV1="db2u_lv_1"
# Please specify the unit for the lvm size we will be creating. For example, specifying a
100 Gigabyte requires LV_SIZE="100G"
LV_SIZE=""

 # Clean the drive or device
sudo dd if=/dev/zero of=/dev/${LOCAL_DISK_DEVICE_1} bs=1M count=2
sudo dd if=/dev/zero of=/dev/${LOCAL_DISK_DEVICE_2} bs=1M count=2
# Initialize the physical volumes for later use by the Logical Volume Manager
pvcreate ${LOCAL_DISK_1}
pvcreate ${LOCAL_DISK_2}
# Create volume group from one or more physical volumes
vgcreate ${VG} ${LOCAL_DISK_1} ${LOCAL_DISK_2}
# Create a logical volume in an existing volume group
lvcreate --name ${LV1} -L ${LV_SIZE} ${VG}

# The device path we want to reference in this example for referencing this lvm we created
would be:
devicePaths:
  - /dev/db2u_vg/db2u_lv_1

# Display information about logical volumes, volume groups, and physical volumes
lvm lvs
vgdisplay db2u_vg
pvdisplay
```

Deploying Db2 using the Db2uCluster custom resource

Once the Db2 Operator is installed, the Db2uCluster custom resource (CR) provides the interface required to deploy Db2. This CR is supported by an OpenShift Custom Resource definition.

Accessing the Db2uCluster custom resource

You deploy Db2 by running Db2uCluster CR commands through a YAML script. You can do this in several ways:

- Through the Red Hat OpenShift console.
- Through the Red Hat OpenShift command-line tool.
- Through the command-line tool of a Kubernetes cluster.

To use the Red Hat OpenShift console, go to **Installed Operators** > **IBM Db2** and click the **Db2uCluster** tab. Through this tab page you create your Db2 instance, either by following the on-screen instructions of the form view, or by entering and running the YAML script.

The following sections cover CR options that can be included in the YAML file. An example of a completed Db2uCluster CR is also included.

Configure the Db2 Version

Specifies the version of the Db2 database based on the Db2 operator installed.

```
spec:
  version: "s11.5.8.0-cn<container layer release number>"
```

For a list of Db2 Operator versions, see Table 1 of Db2 on Red Hat OpenShift.

Configure the database name

Specifies the name of the desired Db2 database.

The following example shows how to configure the database name:

```
spec:
  environment:
    database:
      name: bludb
```

Deploy on a dedicated node

Specifies how to target labels on specific nodes for dedicated deployments. Deploying on dedicated nodes is a best practice in production. See Setting up dedicated nodes for your Db2 deployment.

The following example shows how to deploy a dedicated node:

```
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: database
            operator: In
            values:
            - db2u-affinity
  tolerations:
  - effect: NoSchedule
    key: database
    operator: Equal
    value: db2u-affinity
```

Deploying Db2 with a custom service account

A service account is an OpenShift Container Platform account that allows a component to directly access the API. You can set parameters in your CR to create the Db2 instance with a custom service account.

The following example shows the syntax for identifying your custom service account in the CR:

```
spec:
  account:
    serviceAccountName: ${SERVICE_ACCOUNT}
```

Configure memory and CPU consumption

When deploying Db2 using the Db2 Operator, you have the ability to assign a CPU and Memory profile. This will assign CPU/MEM values to the container running the Db2 Common SQL Engine.

The following example shows how set memory and CPU limits for your Db2 instance:

```
spec:
  podConfig:
    db2u:
      resource:
        db2u:
          limits:
```

```
            cpu: 2
            memory: 4Gi
```

Configure storage

Configure separate storage areas for the following storage categories:

- `meta` shared storage volume for db2 meta data.
- `data` non-shared storage volume for database storage.
- `backup` shared storage volume for backing up the database (optional).
- `activelogs` non-shared storage volume for transactional logs (optional). For more information, see Creating separate storage for database transaction logs.
- `tempts` non-shared storage volume for temporary table spaces (optional). For more information, see Creating separate storage for temporary table spaces.

Note: With certain storage solutions, the same storage class can be specified for both.

For example,

```
spec:
  storage:
  - name: meta
    spec:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 100Gi
      storageClassName: ocs-storagecluster-cephfs
    type: create
  - name: data
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 100Gi
      storageClassName: ocs-storagecluster-ceph-rbd
    type: template
  - name: activelogs
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 100Gi
      storageClassName: ocs-storagecluster-ceph-rbd
    type: template
  - name: backup
    spec:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 100Gi
      storageClassName: ocs-storagecluster-cephfs
    type: create
  - name: tempts
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 100Gi
      storageClassName: ocs-storagecluster-ceph-rbd
    type: template
```

See Certified storage options for Db2 for a full list of for supported storage solutions.

Use persistent storage claims

Existing persistent volume claims can be used also for deployment.

The following example shows the adding of an existing single, shared volume claim:

```
storage:
  - claimName: metapvc
    name: <shared-pvc-name>
    spec:
      resources: {}
    type: existing
```

The following example shows the adding of four existing and separate volume claims:

```
storage:
  - claimName: <meta-pvc-name>
    name: meta
    spec:
      resources: {}
    type: existing
  - claimName: <meta-pvc-name>
    name: data
    spec:
      resources: {}
    type: existing
  - claimName: <backup-pvc-name>
    name: backup
    spec:
      resources: {}
    type: existing
  - claimName: <activelogs-pvc-name>
    name: activelogs
    spec:
      resources: {}
    type: existing
  - claimName: <tempts-pvc-name>
    name: tempts
    spec:
      resources: {}
    type: existing
```

Enabling 4K support

When your Db2 on OpenShift deployment is configured to use either OpenShift Container Storage (OCS) or Portworx container storage (PX), ensure that you have enabled 4K support.

The following example code shows the 4K support set to **ON**:

```
spec:
  environment:
    ...
    instance:
      registry:
        DB2_4K_DEVICE_SUPPORT: "ON"
```

Disabling LDAP

The LDAP directory service is enabled by default. The following example shows how to disable LDAP:

```
spec:
  environment:
    ldap:
      enabled: false
```

Disabling the Node Port service

By default, the creation of the Db2u instance activates a `nodeport` service, for moving external data into your cluster. The following example shows how to disable this behavior.

```
spec:
  environment:
```

```
    database:
      disableNodePortService: true
```

Deploying a Db2 instance with limited privileges

You can set parameters in your CR to create the Db2 instance with limited privileges. This option improves security. The following example shows how to set limited privileges:

```
spec:
  account:
    privileged: false
    restricted: true
```

Overriding the default database settings

You can override the default database settings for your Db2 instance.

CAUTION: You should only override the default database settings if you are aware of the risks involved. If you are at all unsure, accept the default settings.

The following example shows how to set the database values for your Db2 instance:

```
spec:
  environment:
    database:
      settings:
        dftTableOrg: "COLUMN"
        dftPageSize: "32768"
        encrypt: "NO"
        codeset: "UTF-8"
        territory: "US"
        collation: "IDENTITY"
```

Overriding the Db2 database configuration (dbConfig) settings

You can override the default database configuration settings for your Db2 instance.

CAUTION: You should only override the dbConfig settings if you are aware of the risks involved. If you are at all unsure, accept the default settings.

The following example shows how to set the dbConfig values fpr your Db2 instance:

```
spec:
  environment:
    database:
      dbConfig:
        LOGPRIMARY: "50"
        LOGSECOND: "35"
        APPLHEAPSZ: "25600"
        STMTHEAP: "51200 AUTOMATIC"
```

Setting the Db2 registry variable

You can override the default Db2 registry variable values for your Db2 instance.

CAUTION: You should only override the Db2 registry variable settings if you are aware of the risks involved. If you are at all unsure, accept the default settings.

The following example shows how to set the Db2 registry variable values for your Db2 instance:

```
spec:
  environment:
    instance:
      registry:
        DB2_ATS_ENABLE: "NO"
        DB2_OBJECT_STORAGE_SETTINGS: "OFF"
        DB2_DISPATCHER_PEEKTIMEOUT: "2"
        DB2_COMPATIBILITY_VECTOR: "ORA"
```

Example of a complete Db2uCluster CR

The following example shows the CR code to deploy a Db2u Cluster. The CR creates a Db2OLTP instance with the following configuration:

- Database name: BLUDB.
- 4 CPUs.
- 16 GB of memory.
- 5 storage volumes (`meta`, `data`, `backup`, `activelogs`, and `tempts`).
- DB2 4K SUPPORT enabled.
- LDAP disabled.
- Privileged instance.

```
apiVersion: db2u.databases.ibm.com/v1
kind: Db2uCluster
metadata:
  name: db2oltp-test
  namespace: db2u
spec:
  account:
    privileged: true
  environment:
    database:
      name: bludb
    dbType: db2oltp
    ldap:
      enabled: false
  license:
    accept: true
  podConfig:
    db2u:
      resource:
        db2u:
          limits:
            cpu: "4"
            memory: 16Gi
  size: 1
  storage:
  - name: meta
    spec:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 10Gi
      storageClassName: ocs-storagecluster-cephfs
    type: create
  - name: data
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
      storageClassName: ocs-storagecluster-ceph-rbd
    type: template
  - name: backup
    spec:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 1Gi
      storageClassName: ocs-storagecluster-cephfs
    type: create
  - name: activelogs
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
      storageClassName: ocs-storagecluster-ceph-rbd
    type: template
```

```
  - name: tempts
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
      storageClassName: ocs-storagecluster-ceph-rbd
    type: template
  version: s11.5.8.0
```

Creating separate storage for database transaction logs

You can dedicate a separate storage area for database transaction logs on the Db2 pod to reduce I/O bottlenecks and improve performance.

Procedure

Follow one of these procedures depending on whether you are creating log storage for a new deployment or an existing deployment:

- **New deployment**

  a. Create a storage volume for transaction logs by specifying the storage volume in the Db2 cluster custom resource definition (CR). Use the name `activelogs` for the storage volume, and for `storageClassName` specify the name of the storage class that you are using for data storage (the example below uses `ocs-storagecluster-ceph-rbd` for OpenShift Container Platform).

  The following example custom resource definition creates four storage areas for Db2 for meta, data, backup, and active logs storage. The active logs storage definition is highlighted at the end of the CR:

```
apiVersion: db2u.databases.ibm.com/v1
kind: Db2uCluster
metadata:
  name: db2ucluster-sample
spec:
  license:
    accept: true
  account:
    privileged: true
    imagePullSecrets:
      - ibm-registry
  version: "11.5.6.0"
  size: 1
  podConfig:
    db2u:
      resource:
        db2u:
          requests:
            cpu: 2
            memory: 4Gi
          limits:
            cpu: 2
            memory: 4Gi
  environment:
    dbType: db2oltp
    database:
      name: sampledb
      settings:
        dftPageSize: "16384"
        encrypt: "NO"
    instance:
      password: cicdtest
      registry:
        DB2_4K_DEVICE_SUPPORT: "ON"
      dbmConfig:
        DIAGSIZE: "100"
  addOns:
    rest:
      enabled: false
    graph:
      enabled: false
  storage:
```

```
    - name: meta
      type: "create"
      spec:
        storageClassName: "ocs-storagecluster-cephfs"
        accessModes:
          - ReadWriteMany
        resources:
          requests:
            storage: 10Gi
    - name: data
      type: "template"
      spec:
        storageClassName: "ocs-storagecluster-ceph-rbd"
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 100Gi
    - name: backup
      type: "create"
      spec:
        storageClassName: "ocs-storagecluster-cephfs"
        accessModes:
          - ReadWriteMany
        resources:
          requests:
            storage: 100Gi
    - name: activelogs
      type: "template"
      spec:
        storageClassName: "ocs-storagecluster-ceph-rbd"
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
```

b. When you deploy the Db2 service, on the **Transaction logs storage** page of the web console specify the following values:

- **Create new storage**
- **Use storage template**
- **Storage class**: ocs-storagecluster-ceph-rbd
- **Size**: Your chosen value between 1 and 1,000 GiB.

  You can calculate a recommended space for active log storage by using the value of several Db2 configuration parameters. The size should be greater than `LOGFILSIZ (in 4K pages) * (LOGPRIMARY + LOGSECONDARY).` The following default values are set in the container:

```
Log file size (4KB)          (LOGFILSIZ) = 50000
Number of primary log files      (LOGPRIMARY) = 20
Number of secondary log files    (LOGSECOND) = 30
```

  So a calculation that uses the default values would be `(50000*4) * (20 + 30) KB = 10` GB. A recommended allocation for active log storage based on this calculation might be 20 GiB to allow extra space for large workloads.

- **Existing deployment**

a. Patch the `db2ucluster` object to add the path to the transaction logs:

```
oc patch db2ucluster db2oltp-test --type 'json' -p '[{ "op": "add", "path": "/spec/
storage/-", "value": {"name": "activelogs", "spec": {"accessModes": ["ReadWriteOnce"],
"resources": {"requests": {"storage": "20Gi"}},"storageClassName": "ocs-storagecluster-
ceph-rbd"},"type": "create"} }]'
```

In the example, a volume with the name `activelogs` will be added to the Db2 engine pod. In the example, the `db2ucluster` name is db2oltp-test, storage size for the transaction log path is 20 Gi, and the storage class that is used is ocs-storagecluster-ceph-rbd. Update the values based on your environment.

When db2ucluster is patched, the Db2 pods restart and you can see the mounted path as /mnt/logs/active/. You can confirm the pod is mounted by looking up the engine pod description or by running the mount command in the container.

b. While logged in as db2inst1, run a shell function in the pod to set the log path:

```
/bin/bash -c "source /db2u/scripts/include/db2_functions.sh &&
update_transactional_logpath dbname"
```

Where *dbname* is the name of the database for which you are enabling transaction log storage.

c. To check whether the function ran successfully, run this command to confirm the new log path:

```
db2 get db cfg for bludb | grep -i 'Path to log files'
```

Creating separate storage for database archive logs

You can dedicate a separate storage area for database archive logs on the Db2 pod to reduce I/O bottlenecks and improve performance.

About this task

The method that you use to create storage for database archive logs depends on whether you are doing a new deployment of Db2 on OpenShift, or updating an existing deployment.

Creating archive log storage for a new deployment of Db2 on OpenShift

You can dedicate a separate storage area for database archive logs when deploying Db2 on OpenShift for the first time. This separate storage area can reduce I/O bottlenecks and improve performance.

About this task

Procedure

In your Db2 custom resource (CR) definition, add the storage volume: .

- For the storage volume, use archivelogs.
- For **storageClassName**, use the name of the storage class that you are using for data storage. The example below uses ocs-storagecluster-ceph-rbd for the OpenShift Container Platform.

Example

The following example shows the command syntax of a CR definition that is used to create five storage areas for Db2, including archive logs. The archive logs storage definition is located at the end of the CR:

```
apiVersion: db2u.databases.ibm.com/v1
kind: Db2uCluster
metadata:
  name: db2ucluster-sample
spec:
  license:
    accept: true
  account:
    privileged: true
    imagePullSecrets:
      - ibm-registry
  version: "11.5.6.0"
  size: 1
  podConfig:
    db2u:
      resource:
        db2u:
          requests:
            cpu: 2
            memory: 4Gi
          limits:
            cpu: 2
```

```
          memory: 4Gi
environment:
  dbType: db2oltp
  database:
    name: sampledb
    settings:
      dftPageSize: "16384"
      encrypt: "NO"
  instance:
    password: cicdtest
    registry:
      DB2_4K_DEVICE_SUPPORT: "ON"
    dbmConfig:
      DIAGSIZE: "100"
addOns:
  rest:
    enabled: false
  graph:
    enabled: false
storage:
  - name: meta
    type: "create"
    spec:
      storageClassName: "ocs-storagecluster-cephfs"
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 10Gi
  - name: data
    type: "template"
    spec:
      storageClassName: "ocs-storagecluster-ceph-rbd"
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 100Gi
  - name: backup
    type: "create"
    spec:
      storageClassName: "ocs-storagecluster-cephfs"
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 100Gi
  - name: activelogs
    type: "template"
    spec:
      storageClassName: "ocs-storagecluster-ceph-rbd"
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
  - name: archivelogs
    type: "template"
    spec:
      storageClassName: "ocs-storagecluster-ceph-rbd"
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
```

Creating archive log storage for an existing deployment of Db2 on OpenShift

You can dedicate a separate storage area for database archive logs on your Db2 on OpenShift deployment. This separate storage area can reduce I/O bottlenecks and improve performance.

About this task

By default, the archive logs of the database are stored on the same file system as the database. As a result, the file system might run out of space as the database and archive logs continue to grow.

To separate the archive logs from the database file system, you need to create a persistent volume.

Procedure

1. Exec into the Db2 pod.
2. Temporarily stop your Db2 on OpenShift instance:

```
wvcli system disable
```

3. Make a copy of the Db2 StatefulSet:

```
oc get sts c-<db2_instance_id-db2u> -o yaml > c-<db2_instance_id>-db2u.sts.bak
cp c-<db2_instance_id>-db2u.sts.bak c-<db2_instance_id>-db2u.sts
```

   You can find the correct StatefulSet (`sts`) in the project:

```
oc get sts | grep <db2_instance_id>
```

   where <db2_instance_id> is the name of your deployment.

   You can retrieve the deployment name from your db2cluster instance:

```
oc get db2ucluster
```

4. Open the `c-<db2_instance_id>-db2u.sts` file and add two new sections, *volumeMounts* and *volumeClaimTemplates*. Replace *size* and *storageclass_name* with appropriate values:

```
volumeMounts:
 - mountPath: /mnt/logs/archive
   name: archivelogs

volumeClaimTemplates:
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: archivelogs
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <size>
    storageClassName: <storageclass_name>
    volumeMode: Filesystem
```

5. Delete the Db2 StatefulSet and recreate it with the updated `c-<db2_instance_id>-db2u.sts` file.
6. Delete the Db2 pods so that they recycle and inherit the new StatefulSet definition:

```
oc delete sts c-<db2_instance_id>-db2u --cascade=false
oc create -f c-<db2_instance_id>-db2u.sts
# delete all db2 pods, below example only includes Db2 pod with suffix -0.
oc delete pod c-<db2_instance_id>-db2u-0
```

7. Open the `c-<db2_instance>-db2dbconfig` ConfigMap and replace `LOGARCHMETH1 DISK:/mnt/bludata0/db2/archive_log` with `LOGARCHMETH1 DISK:/mnt/logs/archive`.
8. When the db2u-0 pod and all other db2u pods are in 1/1 ready state, run the **exec** command on the Db2 pod, and then run the following commands:

```
chown -R db2inst1:db2iadm1 /mnt/logs/archive/
su - db2inst1 -c "/db2u/scripts/apply-db2cfg-settings.sh"
```

   When you query LOGARCHMETH1,

```
db2 get db cfg | grep -i logarchmeth1
```

   you will see that it is using your new persistent volume mount:

```
 First log archive method                (LOGARCHMETH1) = DISK:/mnt/logs/archive/
```

Creating separate storage for temporary table spaces

You can dedicate a separate storage area for temporary table spaces on the Db2 pod to reduce I/O bottlenecks and improve performance.

Procedure

1. Create a storage volume for temporary table spaces by specifying the storage volume in the Db2 cluster custom resource definition (CR). Use the name `tempts` for the storage volume, and for `storageClassName` specify the name of the storage class that you are using for data storage (the example below uses `ocs-storagecluster-ceph-rbd` for OpenShift Container Storage).

   The following example custom resource definition creates four storage areas for Db2 for meta, data, backup, and temporary table spaces. The `tempts` storage definition is highlighted at the end of the CR:

```
apiVersion: db2u.databases.ibm.com/v1
kind: Db2uCluster
metadata:
  name: db2ucluster-sample
spec:
  license:
    accept: true
  account:
    privileged: true
    imagePullSecrets:
      - ibm-registry
  version: "11.5.6.0"
  size: 1
  podConfig:
    db2u:
      resource:
        db2u:
          requests:
            cpu: 2
            memory: 4Gi
          limits:
            cpu: 2
            memory: 4Gi
  environment:
    dbType: db2oltp
    database:
      name: sampledb
      settings:
        dftPageSize: "16384"
        encrypt: "NO"
    instance:
      password: cicdtest
      registry:
        DB2_4K_DEVICE_SUPPORT: "ON"
      dbmConfig:
        DIAGSIZE: "100"
  addOns:
    rest:
      enabled: false
    graph:
      enabled: false
  storage:
    - name: meta
      type: "create"
      spec:
        storageClassName: "ocs-storagecluster-cephfs"
        accessModes:
          - ReadWriteMany
        resources:
          requests:
            storage: 10Gi
    - name: data
      type: "template"
      spec:
        storageClassName: "ocs-storagecluster-ceph-rbd"
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 100Gi
    - name: backup
      type: "create"
```

```
      spec:
        storageClassName: "ocs-storagecluster-cephfs"
        accessModes:
          - ReadWriteMany
        resources:
          requests:
            storage: 100Gi
- name: tempts
      type: "template"
      spec:
        storageClassName: "ocs-storagecluster-ceph-rbd"
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
```

2. When you deploy the Db2 service, on the **Temporary table space storage** page of the web console specify the following values:
   - **Create new storage**
   - **Use storage template**
   - **Storage class**: ocs-storagecluster-ceph-rbd
   - **Size**: Your chosen value between 1 and 1,000 GiB.

Upgrading and updating your Db2 on Red Hat OpenShift cluster

Your Db2 on Red Hat OpenShift deployment is auto-upgradable and auto-updatable by design. The automation to upgrade or update your Db2 server is fully encapsulated by the container automation.

Upgrading the Db2 Operator

Because of the automatic control that your Db2 Operator has on the configuration of your Db2 on OpenShift deployment, the upgrade and update options available to you are determined by the Operator.

The table below lists the Db2 Operators that have been released in the version 1.x, 2.x, and 3.0 channels, and their supported Db2 versions for deployment on OpenShift.

Table 4. Db2 Operators and their associated Db2 engines

Db2 Operator version	Db2 Operator upgrade channel	Db2 Engine version	OCP version	Container Application Software for Enterprises (CASE) version
1.1.9	1.1	11.5.7.0	4.0.7	4.0.7
1.1.10	1.1	11.5.7.0-cn1	4.0.8	4.0.8
1.1.11	1.1	11.5.7.0-cn2	4.0.9	4.0.9
1.1.12	1.1	11.5.7.0-cn3	4.0.10	4.0.10
1.1.13	1.1	11.5.7.0-cn4	4.0.11	4.0.11
2.0.0	2.0	11.5.7.0-cn5	4.5.0	4.5.0
2.2.0	2.2	11.5.7.0-cn7	4.6, 4.8, 4.10	4.5.3

The table below lists the Db2 Operator channels and the Db2 engine versions they support.

Table 5. Db2 Operator channels and supported Db2 engine versions

Channel	Db2 engine version
v1.1	11.5.6

Table 5. Db2 Operator channels and supported Db2 engine versions (continued)	
Channel	**Db2 engine version**
v2.x	11.5.7
v110508.0	11.5.8

Upgrading the Db2 Operator

You can upgrade the Db2 Operator using the IBM Cloud Pak CLI tool.

About this task

The IBM Cloud Pak CLI (cloudctl) provides significant benefits when dealing with air-gapped (disconnected) environments for case management. It provides a common framework for IBM operators around a consistent and optimized air gap install experience, using bastion, non-bastion or portable storage.

You will need to install the cloudctl tool before proceeding.

Procedure

1. Download and extract the Container Application Software for Enterprises (CASE) bundle:
   a. Set up environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment:

      ```
      export CASE_NAME=ibm-db2uoperator
      export CASE_VERSION=<CASE version of latest Db2 Operator>
      export OFFLINEDIR=/tmp/cases
      export CASEPATH="https://github.com/IBM/cloud-pak/raw/master/repo/case"
      export OFFLINECASE=${OFFLINEDIR}/${CASE_NAME}
      export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz
      ```

   b. Create a directory to save the CASE bundle to a local directory:

      ```
      mkdir -p ${OFFLINEDIR}
      ```

   c. Download the CASE bundle:

      ```
      $ cloudctl case save --repo ${CASEPATH} --case ${CASE_NAME} --version ${CASE_VERSION} --
      outputdir ${OFFLINEDIR}
      Downloading and extracting the CASE ...
      - Success
      Retrieving CASE version ...
      - Success
      Validating the CASE ...
      - Success
      Creating inventory ...
      - Success
      Finding inventory items
      - Success
      Resolving inventory items ...
      Parsing inventory items
      - Success
      ```

   d. Verify that the CASE bundle and images csv have been downloaded:

      ```
      $ ls ${OFFLINEDIR}
      total 128K
      drwxr-xr-x 2 root root    6 Jan 20 11:10 charts/
      -rw-r--r-- 1 root root 116K Jan 20 11:10 ibm-db2uoperator-1.0.x.tgz
      -rw-r--r-- 1 root root   32 Jan 20 11:10 ibm-db2uoperator-1.0.x-charts.csv
      -rw-r--r-- 1 root root 5.2K Jan 20 11:10 ibm-db2uoperator-1.0.x-images.csv
      ```

   e. Extract the CASE bundle:

```
cd ${OFFLINEDIR}
tar -xvzf ${CASE_ARCHIVE}
```

2. Install the Db2 catalog:

```
cloudctl case launch                        \
    --case ${OFFLINECASE}                   \
    --namespace openshift-marketplace       \
    --inventory db2uOperatorSetup           \
    --action installCatalog
```

3. Install the Db2 operator:

```
cloudctl case launch                         \
     --case ${OFFLINECASE}                   \
     --namespace ${NS}                       \
     --inventory db2uOperatorStandaloneSetup \
     --action installOperator
```

4. Verify the installation:

```
oc get CatalogSources ibm-db2uoperator-catalog -n openshift-marketplace
```

When the catalog source is updated, the Db2 Operator is automatically upgraded if the `installPlanApproval` property is set to `Automatic` in your Db2 Operator subscription.

Check the setting of the `installPlanApproval` property in your Db2 Operator subscription:

```
 $ oc get subscription ibm-db2uoperator-catalog-subscription -oyaml | grep
installPlanApproval
```

Note: The Db2 Operator `installPlanApproval` property is set to `Automatic` by default.

If the `installPlanApproval` is not set to `Automatic`, upgrade the Db2 Operator by manually approving the upgrades. You can find the steps to manually approve the upgrades in the CustomResourceDefinitions > Subscription page of the *Operator Lifecycle Manager* site, or from the *Operator Details* page of the Db2 Operator on the OpenShift web console.

Upgrading the Db2Cluster instance

You can upgrade your Db2 server by editing the source YAML of the custom resource (CR) that is associated with your Db2 on OpenShift cluster.

About this task

Note: If you have configured HADR for your Db2 on OpenShift deployment, do not proceed. See Upgrading HADR Db2 deployments.

When you have updated the CR version, the Db2 operator automatically reconciles all the objects that were created and managed by the Db2 operator when the CR was first created. The Db2 Operator also edits the Db2 on OpenShift **statefulset** to point to the new images, causing your Db2 engine pods to be recreated using the new images. This process automatically updates or upgrades Db2 on your OpenShift cluster.

Procedure

1. From the console of your OpenShift cluster, select the **Db2u Cluster** tab.
2. From the table, locate the custom resource (CR) that is linked to your Db2 deployment and, from the far-right column, click the ellipses (...) and select **Edit Db2uCluster**.
3. In the text box that appears, go through the YAML file and change the value of `version`, under `spec`, to the target version number:

```
spec:
  version: "11.5.<x>.<y>-cn<z>>"
```

where <x> is the latest mod pack number, <y> is the latest fix pack number, and <z> is the latest container layer enhanced release number.

4. Save your changes.

Upgrading HADR Db2 on OpenShift deployments

You can upgrade the Db2® server by editing the source YAML of the custom resource (CR) that is associated with your Db2 on OpenShift® cluster. When HADR is enabled, the upgrade steps vary depending on the Db2 on OpenShift release to which you are upgrading.

Before you begin

- Upgrade your Db2 Operator to the intended version.
- The primary and principal standby databases must be in PEER state before you start the upgrade.

  Note: This upgrade procedure does not apply when upgrading to previous versions, such as Db2 11.5.7.0. When upgrading to previous versions, you need to first stop your HADR deployment, upgrade your Db2uCluster, and then reconfigure your HADR deployment.

About this task

If you are upgrading to the Db2 on OpenShift 11.5.8 release (including CNs), there is a slight interruption when processing is switched from one database to the other. The duration of the interruption is the time it takes for ACR to switch between the primary and standby connection. The length of this duration depends on how you have configured automatic client reroute (ACR).

Note: This upgrade procedure keeps at least one HADR database available throughout the upgrade process. In previous upgrades to HADR deployments of Db2 on OpenShift 11.5.7, the databases were unavailable during the upgrade process.

Procedure

1. Modify environment and version variables for your deployments:

   ```
   DB2UCLUSTER_PRIMARY="db2oltp-crd-hadr-primary"
   DB2UCLUSTER_STANDBY="db2oltp-crd-hadr-standby"
   PROJECT_PRIMARY="zen"
   PROJECT_STANDBY="zen"
   DB2_VERSION="s11.5.8.0"
   ```

   `Db2uCluster` names can be obtained with the `oc get db2ucluster` command, in the namespace where the deployments are located. The `DB2_VERSION` is the version number to which you want to upgrade. For more information, see Upgrading and updating your Db2 on Red Hat OpenShift cluster.

2. Run the stop governor command on your deployments.

   Note: Governor does not run on auxiliary standbys. This step isn't necessary on auxiliary deployments.

   Run the command on your principal standby deployment first, then your primary deployment:

   ```
   oc exec -it c-${DB2UCLUSTER_STANDBY}-db2u-0 -n ${PROJECT_STANDBY} -- sv stop governor
   ```

   ```
   oc exec -it c-${DB2UCLUSTER_PRIMARY}-db2u-0 -n ${PROJECT_PRIMARY} -- sv stop governor
   ```

3. Upgrade your standby deployment.
4. Patch the `Db2uCluster` custom resource to the new version:

   ```
   oc patch db2ucluster ${DB2UCLUSTER_STANDBY} -n ${PROJECT_STANDBY} --type merge -p
   "{\"spec\":{\"version\":\"${DB2_VERSION}\"}}"
   ```

   The `Db2uCluster` returns READY when the standby deployment upgrade is complete.

5. Return the status:

```
oc get db2ucluster ${DB2UCLUSTER_STANDBY} -n ${PROJECT_STANDBY}
```

6. If there are auxiliary deployments in the HADR topology, upgrade them one at a time. Repeat this step and wait for each deployment to return a READY status.
7. Do a role switch to the standby database to set it as the new primary database:

```
oc exec -it c-${DB2UCLUSTER_STANDBY}-db2u-0 -n ${PROJECT_STANDBY} -- su - db2inst1
```

```
db2 takeover hadr on db ${DBNAME}
```

8. Upgrade the new standby (former primary) deployment.
9. Patch the `Db2uCluster` custom resource to the new version:

```
oc patch db2ucluster ${DB2UCLUSTER_PRIMARY} -n ${PROJECT_PRIMARY} --type merge -p
"{\"spec\":{\"version\":\"${DB2_VERSION}\"}}"
```

   The `Db2uCluster` returns READY when the standby deployment upgrade is complete.
10. Return the status:

```
oc get db2ucluster ${DB2UCLUSTER_PRIMARY} -n ${PROJECT_PRIMARY}
```

11. Update the databases.
12. Run the `db2u_update.sh` update script on the current primary (former standby) deployment with the `-db` parameter:

```
oc exec -it c-${DB2UCLUSTER_STANDBY}-db2u-0 -n ${PROJECT_STANDBY} -- bash
```

```
su - db2inst1 -s /bin/bash -c '/db2u/scripts/db2u_update.sh -db'
```

13. Wait for the log replay to complete.
14. Ensure that the HADR pair has reached PEER state again by running the following command and reviewing the status:

```
oc exec -it c-${DB2UCLUSTER_STANDBY}-db2u-0 -n ${PROJECT_STANDBY} -- manage_hadr -status
```

15. Deactivate, then activate the database in the current standby (former primary) deployment:

```
oc exec -it c-${DB2UCLUSTER_PRIMARY}-db2u-0 -n ${PROJECT_PRIMARY} -- su - db2inst1
```

```
db2 deactivate db ${DBNAME}
```

```
db2 activate db ${DBNAME}
```

   If there are auxiliary deployments in the HADR topology, repeat this step to deactivate and activate each deployment.
16. Perform a role switch to revert back to the original primary deployment by running the following command:

```
oc exec -it c-${DB2UCLUSTER_PRIMARY}-db2u-0 -n ${PROJECT_PRIMARY} -- su - db2inst1
```

```
db2 takeover hadr on db ${DBNAME}
```

17. Deactivate, then activate the database in the standby deployment by running the following command:

```
oc exec -it c-${DB2UCLUSTER_STANDBY}-db2u-0 -n ${PROJECT_STANDBY} -- su - db2inst1
```

```
db2 deactivate db ${DBNAME}
```

```
db2 activate db ${DBNAME}
```

18. Start the governor on the primary deployment:

```
oc exec -it c-${DB2UCLUSTER_PRIMARY}-db2u-0 -n ${PROJECT_PRIMARY} -- su - db2inst1
```

```
${HOME}/governor/governor.sh start
```

19. Start the governor on the standby deployment:

```
oc exec -it c-${DB2UCLUSTER_STANDBY}-db2u-0 -n ${PROJECT_STANDBY} -- su - db2inst1
```

```
${HOME}/governor/governor.sh start
```

Your HADR Db2 on OpenShift deployment is now upgraded to the 11.5.8.0 release.

Upgrading the Db2 Operator in an air-gapped environment

There are two ways of upgrading the Db2 Operator in an air-gabbed environment, either through a bastion host or by transferring it to the cluster from a portable device.

Upgrading the Db2 Operator through a bastion host

You can upgrade the Db2 Operator when installed in an air-gapped OpenShift cluster that uses a bastion host for connections.

Before you begin

Before upgrading the Db2 Operator on your bastion machine, ensure that it is properly configured by logging onto the machine and performing the following tasks:

- Verify that the bastion machine has access to the following:
  - public internet [to download the required Container Application Software for Enterprises (CASE) bundle]
  - a target image registry (where the images are mirrored)
  - a target OpenShift cluster onto which to install the operator
- Download and install the dependent command line tools:
  - oc - For interacting with the OpenShift Cluster
  - cloud-pak-cli - For downloading and installing the CASE bundle

Note: Do all of the steps in the following procedure from the bastion machine.

Procedure

1. Download and extract the CASE bundle:
   a. Set up the environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment:

```
export NS=<Namespace of target installation on OpenShift cluster>
export CASE_NAME=ibm-db2uoperator
export CASE_VERSION=<CASE version of latest Db2 Operator>
export OFFLINEDIR=/tmp/cases
export OFFLINECASE=${OFFLINEDIR}/${CASE_NAME}
export CASEPATH="https://github.com/IBM/cloud-pak/raw/master/repo/case"
export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

# Details of the target registry to copy to
export TARGET_REGISTRY_HOST=""          # Target registry host
export TARGET_REGISTRY_PORT=5000        # Target registry port number
export TARGET_REGISTRY=${TARGET_REGISTRY_HOST}:${TARGET_REGISTRY_PORT}
```

```
export TARGET_REGISTRY_USER="user"      # Actual username goes here
export TARGET_REGISTRY_PASSWORD="key"   # Actual API Key goes here
```

b. Create a directory to save the CASE to a local directory:

```
$ mkdir ${OFFLINEDIR}
```

c. Download and extract the CASE bundle:

```
$ cloudctl case save --repo ${CASEPATH} --case ${CASE_NAME} --version ${CASE_VERSION} --
outputdir ${OFFLINEDIR}
Downloading and extracting the CASE ...
- Success
Retrieving CASE version ...
- Success
Validating the CASE ...
- Success
Creating inventory ...
- Success
Finding inventory items
- Success
Resolving inventory items ...
Parsing inventory items
- Success
```

d. Verify the CASE and images csv has been downloaded:

```
$ ls ${OFFLINEDIR}
total 128K
drwxr-xr-x 2 root root    6 Jan 20 11:10 charts/
-rw-r--r-- 1 root root 116K Jan 20 11:10 ibm-db2uoperator-1.0.x.tgz
-rw-r--r-- 1 root root   32 Jan 20 11:10 ibm-db2uoperator-1.0.x-charts.csv
-rw-r--r-- 1 root root 5.2K Jan 20 11:10 ibm-db2uoperator-1.0.x-images.csv
```

e. Extract the CASE:

```
cd ${OFFLINEDIR}
tar -xvzf ${CASE_ARCHIVE}
```

2. Configure your registry authentication secrets:

a. Create an authentication secret for target image registry:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${TARGET_REGISTRY} --user ${TARGET_REGISTRY_USER} --pass $
{TARGET_REGISTRY_PASSWORD}"

The credentials are now saved to `~/.airgap/secrets/<registry-name>.json`
```

3. Copy the images from your saved CASE (images.csv) to the target registry in the air-gapped cluster.

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action mirror-images          \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

4. Configure the air-gapped cluster to use its internal/target image registry:

Warning: Cluster resources must adjust to the new pull secret, which can temporarily limit the usability of the cluster. Authorization credentials are stored in `$HOME/.airgap/secrets` and `/tmp/airgap*` to support this action.

a. Apply an image source content policy. Doing so causes each worker node to restart:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
```

```
    --inventory db2uOperatorSetup     \
    --action configure-cluster-airgap \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

b. Add the target registry to the cluster **insecureRegistries** list if the target registry isn't secured by a certificate. Run the following command to restart all nodes, one at a time:

```
$ oc patch image.config.openshift.io/cluster --type=merge
-p "{\"spec\":{\"registrySources\":{\"insecureRegistries\":[\"${TARGET_REGISTRY_HOST}:$
{TARGET_REGISTRY_PORT}\", \"${TARGET_REGISTRY_HOST}\"]}}}"
```

5. Install the catalog source:

```
cloudctl case launch                        \
    --case ${OFFLINECASE}                   \
    --namespace openshift-marketplace       \
    --inventory db2uOperatorSetup           \
    --action installCatalog
```

6. Install the Db2 Operator:

```
cloudctl case launch                         \
     --case ${OFFLINECASE}                   \
     --namespace ${NS}                       \
     --inventory db2uOperatorStandaloneSetup \
     --action installOperator
```

7. Verify the installation:

```
oc get CatalogSources ibm-db2uoperator-catalog -n openshift-marketplace
```

When the catalog source is updated, the Db2 Operator is automatically upgraded if the `installPlanApproval` property is set to `Automatic` in your Db2 Operator subscription.

Check the setting of the `installPlanApproval` property in your Db2 Operator subscription:

```
 $ oc get subscription ibm-db2uoperator-catalog-subscription -oyaml | grep
installPlanApproval
```

Note: The Db2 Operator `installPlanApproval` property is set to `Automatic` by default.

If the `installPlanApproval` is not set to `Automatic`, upgrade the Db2 Operator by manually approving the upgrades. You can find the steps to manually approve the upgrades in the CustomResourceDefinitions > Subscription page of the *Operator Lifecycle Manager* site, or from the *Operator Details* page of the Db2 Operator on the OpenShift web console.

Upgrading the Db2 Operator in an air-gapped OpenShift cluster with no bastion host

You can upgrade the Db2 Operator in an air-gapped OpenShift cluster by doing an on-site transfer of the required components from a portable device.

Before you begin

Prepare a portable device, such as a laptop, with the required Container Application Software for Enterprises (CASE) bundle.

- Verify that the portable device has access to the following:
  - public internet (to download CASE and images)
  - a target image registry (where the images will be mirrored)
  - a target OpenShift cluster onto which to install the operator
- Download and install dependent command line tools
  - oc - To interact with your OpenShift cluster
  - cloud-pak-cli - To download and install the CASE bundle

Note: Do all of the steps in the following procedure from the portable device.

Procedure

1. Download and extract the CASE bundle:
   a. Set up environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment.

      ```
      export NS=<Namespace of target installation on OpenShift cluster>
      export CASE_NAME=ibm-db2uoperator
      export CASE_VERSION=<CASE version of latest Db2 Operator>
      export OFFLINEDIR=/tmp/cases
      export OFFLINECASE=${OFFLINEDIR}/${CASE_NAME}
      export CASEPATH="https://github.com/IBM/cloud-pak/raw/master/repo/case"
      export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

      # Details of the target registry to copy to
      export TARGET_REGISTRY_HOST=""          # Target registry host
      export TARGET_REGISTRY_PORT=5000        # Target registry port number
      export TARGET_REGISTRY=${TARGET_REGISTRY_HOST}:${TARGET_REGISTRY_PORT}
      export TARGET_REGISTRY_USER="user"      # Actual username goes here
      export TARGET_REGISTRY_PASSWORD="key"   # Actual API Key goes here
      ```

   b. Create a directory to save the CASE to a local directory:

      ```
      $ mkdir ${OFFLINEDIR}
      ```

   c. Download and extract the CASE bundle:

      ```
      $ cloudctl case save --repo ${CASEPATH} --case ${CASE_NAME} --version ${CASE_VERSION}
      --outputdir ${OFFLINEDIR}
      Downloading and extracting the CASE ...
      - Success
      Retrieving CASE version ...
      - Success
      Validating the CASE ...
      - Success
      Creating inventory ...
      - Success
      Finding inventory items
      - Success
      Resolving inventory items ...
      Parsing inventory items
      - Success
      ```

   d. Verify the CASE and images csv has been downloaded:

      ```
      $ ls ${OFFLINEDIR}
      total 128K
      drwxr-xr-x 2 root root    6 Jan 20 11:10 charts/
      -rw-r--r-- 1 root root 116K Jan 20 11:10 ibm-db2uoperator-1.0.x.tgz
      -rw-r--r-- 1 root root   32 Jan 20 11:10 ibm-db2uoperator-1.0.x-charts.csv
      -rw-r--r-- 1 root root 5.2K Jan 20 11:10 ibm-db2uoperator-1.0.x-images.csv
      ```

   e. Extract the CASE:

      ```
      cd ${OFFLINEDIR}
      tar -xvzf ${CASE_ARCHIVE}
      ```

2. Copy the images to the local container registry on the portable device:
   a. Set up environment variables.

      Review the following parameters for your environment and then run the following commands to set up the environment:

      ```
      export NS=<Namespace of target installation on OpenShift cluster>
      export CASE_NAME=ibm-db2uoperator
      export CASE_VERSION=<CASE version of latest Db2 Operator>
      export OFFLINEDIR=/tmp/cases
      export OFFLINECASE=${OFFLINEDIR}/${CASE_NAME}
      ```

```
export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

# Details of the intermediate registry if not using a Bastion server
export PORTABLE_REGISTRY_HOST=localhost
export PORTABLE_REGISTRY_PORT=5000
export PORTABLE_REGISTRY=${PORTABLE_REGISTRY_HOST}:${PORTABLE_REGISTRY_PORT}
export PORTABLE_REGISTRY_USER="user"      # Actual username goes here
export PORTABLE_REGISTRY_PASSWORD="key"   # Actual API Key goes here
export PORTABLE_REGISTRY_PATH=${OFFLINECASE}/registry
export PORTABLE_STORAGE_LOCATION=""       # Override

# Details of the target registry to copy to
export TARGET_REGISTRY_HOST=""          # Target registry host
export TARGET_REGISTRY_PORT=5000        # Target registry port number
export TARGET_REGISTRY=${TARGET_REGISTRY_HOST}:${TARGET_REGISTRY_PORT}
export TARGET_REGISTRY_USER="user"      # Actual username goes here
export TARGET_REGISTRY_PASSWORD="key"   # Actual API Key goes here
```

b. Set the target registry:

```
export TARGET_REGISTRY=${PORTABLE_REGISTRY}
export TARGET_REGISTRY_USER=${PORTABLE_REGISTRY_USER}
export TARGET_REGISTRY_PASS=${PORTABLE_REGISTRY_PASSWORD}
```

c. Initialize the Docker registry by running the following command:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action init-registry        \
    --args "--registry $PORTABLE_REGISTRY_HOST --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

d. Start the Docker registry by running the following command:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action start-registry       \
    --args "--registry $PORTABLE_REGISTRY --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

3. Configure your registry authentication secrets:

a. Create an authentication secret for target image registry:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${TARGET_REGISTRY} --user ${TARGET_REGISTRY_USER} --pass $
{TARGET_REGISTRY_PASSWORD}"

The credentials are now saved to `~/.airgap/secrets/<registry-name>.json`
```

4. Copy the images from your saved CASE (images.csv) to the target registry in the air-gapped cluster:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action mirror-images          \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

5. Copy the offline case inventory images and registry data folder to the portable storage device:

```
cp -r ${OFFLINECASE} ${PORTABLE_STORAGE_LOCATION}
```

6. Copy the images to the target registry behind the firewall:

a. Set up environment variables.

Review the following parameters for your environment and then run the following commands to set up the environment:

```
export NS=<Namespace of target installation on OpenShift cluster>
export CASE_NAME=ibm-db2uoperator
export CASE_VERSION=<CASE version of latest Db2 Operator>
export OFFLINEDIR=/tmp/cases
export OFFLINECASE=${OFFLINEDIR}/${CASE_NAME}
export CASE_ARCHIVE=${CASE_NAME}-${CASE_VERSION}.tgz

# Details of the intermediate registry if not using a Bastion server
export PORTABLE_REGISTRY_HOST=localhost
export PORTABLE_REGISTRY_PORT=5000
export PORTABLE_REGISTRY=${PORTABLE_REGISTRY_HOST}:${PORTABLE_REGISTRY_PORT}
export PORTABLE_REGISTRY_USER="user"      # Actual username goes here
export PORTABLE_REGISTRY_PASSWORD="key"   # Actual API Key goes here
export PORTABLE_REGISTRY_PATH=${OFFLINECASE}/registry
export PORTABLE_STORAGE_LOCATION=""       # Override

# Details of the target registry to copy to
export INTERNAL_REGISTRY_HOST=""          # Target registry host
export INTERNAL_REGISTRY_PORT=5000        # Target registry port number
export INTERNAL_REGISTRY=${INTERNAL_REGISTRY_HOST}:${INTERNAL_REGISTRY_PORT}
export INTERNAL_REGISTRY_USER="user"      # Actual username goes here
export INTERNAL_REGISTRY_PASSWORD="key"   # Actual API Key goes here
```

b. Set the source and target registries.

The source container registry is now the local registry on the portable device, for example **localhost:5000** and the destination is the registry behind the firewall, for example **10.10.4.6:5000**, or the host and port in your air-gap environment. You need to set up the environment variables, mirror the images, and then install the catalog.

Set up the environment variables:

```
export SOURCE_REGISTRY=${PORTABLE_REGISTRY}
export SOURCE_REGISTRY_USER=${PORTABLE_REGISTRY_USER}
export SOURCE_REGISTRY_PASS=${PORTABLE_REGISTRY_PASSWORD}

export TARGET_REGISTRY=${INTERNAL_REGISTRY}
export TARGET_REGISTRY_USER=${INTERNAL_REGISTRY_USER}
export TARGET_REGISTRY_PASS=${INTERNAL_REGISTRY_PASSWORD}
```

Override the registry storage location to point to the location of the portable storage:

```
export PORTABLE_STORAGE_LOCATION=#Provide external storage path here
```

Copy the offline case inventory images and registry data folder from the portable storage device to the node.

```
cp -r ${PORTABLE_STORAGE_LOCATION} ${OFFLINECASE}
```

c. Initialize the Docker registry:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action init-registry        \
    --args "--registry $PORTABLE_REGISTRY_HOST --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

d. Start the Docker registry by running the following command:

```
cloudctl case launch              \
    --case ${OFFLINECASE}         \
    --inventory db2uOperatorSetup \
    --action start-registry       \
    --args "--registry $PORTABLE_REGISTRY --user $PORTABLE_REGISTRY_USER --pass
$PORTABLE_REGISTRY_PASSWORD --dir $PORTABLE_REGISTRY_PATH"
```

7. Configure your registry authentication secrets:

a. Create an authentication secret for the source image registry:

Note: If the registry is public, which doesn't require credentials, skip this step.

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${SOURCE_REGISTRY} --user ${SOURCE_REGISTRY_USER} --pass $
{SOURCE_REGISTRY_PASSWORD}"
```

b. Create an authentication secret for target image registry:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-creds-airgap \
    --args "--registry ${TARGET_REGISTRY} --user ${TARGET_REGISTRY_USER} --pass $
{TARGET_REGISTRY_PASSWORD}"
```

8. Copy the images from the saved CASE (images.csv) to the target registry in the air-gap environment:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action mirror-images          \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

9. Configure the air-gapped cluster to use its internal/target image registry:

Warning: Cluster resources must adjust to the new pull secret, which can temporarily limit the usability of the cluster. Authorization credentials are stored in `$HOME/.airgap/secrets` and `/tmp/airgap*` to support this action.

a. Apply an image source content policy. Doing so causes each worker node to restart:

```
$ cloudctl case launch              \
    --case ${OFFLINECASE}           \
    --namespace ${NS}               \
    --inventory db2uOperatorSetup   \
    --action configure-cluster-airgap \
    --args "--registry ${TARGET_REGISTRY} --inputDir ${OFFLINECASE}"
```

b. Add the target registry to the cluster **insecureRegistries** list if the target registry isn't secured by a certificate. Run the following command to restart all nodes, one at a time:

```
$ oc patch image.config.openshift.io/cluster --type=merge
-p "{\"spec\":{\"registrySources\":{\"insecureRegistries\":[\"${TARGET_REGISTRY_HOST}:$
{TARGET_REGISTRY_PORT}\", \"${TARGET_REGISTRY_HOST}\"]}}}"
```

10. Install the catalog source:

```
cloudctl case launch                    \
    --case ${OFFLINECASE}               \
    --namespace openshift-marketplace   \
    --inventory db2uOperatorSetup       \
    --action installCatalog
```

11. Install the Db2 Operator:

```
cloudctl case launch                     \
     --case ${OFFLINECASE}               \
     --namespace ${NS}                   \
     --inventory db2uOperatorStandaloneSetup \
     --action installOperator
```

12. Verify the installation:

```
oc get CatalogSources ibm-db2uoperator-catalog -n openshift-marketplace
```

When the catalog source is updated, the Db2 Operator is automatically upgraded if the `installPlanApproval` property is set to `Automatic` in your Db2 Operator subscription.

Check the setting of the `installPlanApproval` property in your Db2 Operator subscription:

```
 $ oc get subscription ibm-db2uoperator-catalog-subscription -oyaml | grep
installPlanApproval
```

Note: The Db2 Operator `installPlanApproval` property is set to `Automatic` by default.

If the `installPlanApproval` is not set to `Automatic`, upgrade the Db2 Operator by manually approving the upgrades. You can find the steps to manually approve the upgrades in the CustomResourceDefinitions > Subscription page of the *Operator Lifecycle Manager* site, or from the *Operator Details* page of the Db2 Operator on the OpenShift web console.

Upgrading your Db2 Community Edition license certificate key

Db2 for Red Hat OpenShift can be upgraded from the Community Edition by applying a new license activation key.

About this task

You can upgrade your Db2 for OpenShift Community Edition license to any of the Db2 11.5.5 production licenses:

Table 6. Processor and memory limits per Db2 server or pureScale cluster

Db2 edition	**Processor cores**	**Allowed instance memory**
Advanced	Unlimited	Unlimited
Standard	Up to 16	128 GB
Community	Up to 4	16 GB

License keys for the Advanced and Standard editions can be downloaded from the IBM Passport Advantage site:

- Db2 Advanced Edition: `db2adv_vpc.lic`
- Db2 Standard Edition: `db2std_vpc.lic`

Note: You must be entitled to use a license before you can download it. Ensure that you or your organization has purchased the required license.

Procedure

1. Encode your Db2 license to base64 by running the following command::

   ```
   LICENSE_KEY="./db2adv_vpc.lic"
   cat ${LICENSE_KEY} | base64 | tr -d '\n'
   ```

   Save the encoded output, as you will add it to your YAML file later in this procedure.
2. Log in to your Red Hat OpenShift cluster and use the command line interface (CLI) to get the name of your db2ucluster instance:

   ```
   # oc get db2ucluster
   NAME                          STATE   AGE
   db2ucluster-sample            Ready   1d

   # INSTANCE_NAME="db2ucluster-sample"
   ```
3. Edit the instance.

   ```
   oc edit db2ucluster ${INSTANCE_NAME}
   ```

4. In edit mode, locate the `license` key under `spec` in the YAML file and add the encoded string from step 1 as a value key, as shown below:

```
spec:
  license:
    value: <ENCODED STRING FROM STEP 1 GOES HERE>
```

5. Save and close the YAML file.

```
# oc edit db2ucluster ${INSTANCE_NAME}
db2ucluster.db2u.databases.ibm.com/db2ucluster-sample edited
```

6. Scale down the Db2 `statefulset` to 0:

```
oc scale --replicas=0 statefulset c-${INSTANCE_NAME}-db2u
```

7. Scale up the Db2 `statefulset` back to 1 for the new license to take effect:

```
oc scale --replicas=1 statefulset c-${INSTANCE_NAME}-db2u
```

8. Once the new Db2 pod is ready, verify the updated Db2 license:

```
oc exec -it  c-${INSTANCE_NAME}-db2u-0 -- su - db2inst1 -c "db2licm -l"
```

What to do next

To upgrade processor speed and memory capacity for your new Db2 server, see Scaling up Db2.

To apply annotations for your licensed Db2 instance to your db2cluster instance, see “Injecting license annotations into a Db2 on OpenShift deployment” on page 51.

Injecting license annotations into a Db2 on OpenShift deployment

You can annotate the details of your Db2 edition license for license utilization reporting. Annotations are made to the db2ucluster instance of your Db2 on OpenShift deployment.

About this task

The IBM License Service is used to report license utilization in dynamic cloud environments, measuring license consumption of IBM Containers. To enable license tracking for containerized Db2, additional annotations must be added to the db2ucluster custom resource (CR) at the time of deployment, or when the license is upgraded.

To apply the annotation for your Db2 edition, you must update the `spec.podConfig.db2u.annotations` fields of your `db2ucluster` instance.

Note: Applying the annotation to your Db2 on OpenShift instance requires that the instance be restarted. As a result, your instance is unavailable during this time.

Table 7. Product annotations for different Db2 editions

Db2 edition	CCID	Product ID	Product Name	Product Metric	Product Charged Containers
Advanced	CC029674	2a175af984234bf6bf6a681494f88b4b	IBM Db2 Advanced Edition VPC Option	VIRTUAL_PROCESSOR_CORE	All
Standard	CC029676	237beb0aed3a4083af3e17f37f4b7a15	IBM Db2 Standard Edition VPC Option	VIRTUAL_PROCESSOR_CORE	All

Table 7. Product annotations for different Db2 editions (continued)					
Db2 edition	**CCID**	**Product ID**	**Product Name**	**Product Metric**	**Product Charged Containers**
Community	N/A	b1a38d065bfc40478cacdb2bee266a4d	IBM Db2 Community Edition	FREE	All

Procedure

1. From the console of your OpenShift cluster, identify the name of your db2ucluster instance and store it in an environment variable:

```
oc get db2ucluster
```

```
DB2UCLUSTER_NAME="db2ucluster-sample"
```

2. Set the appropriate environment variables for your licensed Db2 edition:

   For Db2 Advanced Edition:

```
PRODUCT_ID="2a175af984234bf6bf6a681494f88b4b"
PRODUCT_NAME="IBM Db2 Advanced Edition VPC Option"
PRODUCT_METRIC="VIRTUAL_PROCESSOR_CORE"
PRODUCT_CHARGED_CONTAINERS="All"
```

   For Db2 Standard Edition:

```
PRODUCT_ID="237beb0aed3a4083af3e17f37f4b7a15"
PRODUCT_NAME="IBM Db2 Standard Edition VPC Option"
PRODUCT_METRIC="VIRTUAL_PROCESSOR_CORE"
PRODUCT_CHARGED_CONTAINERS="All"
```

   For Db2 Community Edition:

```
PRODUCT_ID="b1a38d065bfc40478cacdb2bee266a4d"
PRODUCT_NAME="IBM Db2 Community Edition"
PRODUCT_METRIC="FREE"
PRODUCT_CHARGED_CONTAINERS="All"
```

3. From the OpenShift terminal, apply the annotations to the db2ucluster instance:

```
oc patch db2ucluster $DB2UCLUSTER_NAME --type merge -p "{\"spec\":{\"podConfig\":{\"db2u\":
{\"annotations\":
{\"productID\":\"$PRODUCT_ID\",\"productName\":\"$PRODUCT_NAME\",\"productMetric\":\"$PRODUCT
_METRIC\",\"productChargedContainers\":\"$PRODUCT_CHARGED_CONTAINERS\"}}}}}"
```

4. Verify that the annotations are applied correctly to the `spec.podConfig.db2u.annotations` field of your db2ucluster instance:

```
oc get db2ucluster $DB2UCLUSTER_NAME -ojsonpath="{.spec.podConfig.db2u.annotations}"
```

5. Once the Db2 instance is ready, verify that the annotations were applied:

```
oc get po c-$DB2UCLUSTER_NAME-db2u-0 -ojsonpath='{.metadata.annotations}'
```

Example

The following code shows how the verification of the `spec.podConfig.db2u.annotations` change might look:

```
{"productChargedContainers":"All","productID":"237beb0aed3a4083af3e17f37f4b7a15","productMetric"
:"VIRTUAL_PROCESSOR_CORE","productName":"IBM Db2 Standard Edition VPC Option"}
```

Administering Db2

The following information describes common tasks that an administrator performs to maintain a Db2 database in production.

Note: You can use the Db2 Data Management Console to administer, monitor, manage, and optimize the performance of Db2 databases. If you plan to use the console to manage your Db2 on Red Hat OpenShift configuration, you must install the console separately.

Loading data into Db2 on Red Hat OpenShift

To load data into Db2 on Red Hat OpenShift, the data must be in a location that is visible from within the Db2 container. Supported loading methods differ for the web console and command line.

Web console

The Db2 console allows to load Data from CSV files, Amazon S3, and Cloud Object Storage.

Command line

Loading methods that are supported by commands differ depending on whether you are using traditional storage or software-defined storage such as Red Hat OpenShift Container Storage or Portworx.

Creating a staging area

Place the data, for example from flat files, in a folder that is visible under the **System Storage** area that you defined when you deployed Db2. System storage is shared. User data storage is not shared. Inside the Db2 container, the location for system storage is mounted as `/mnt/blumeta0/` and you can create a staging area by running the following commands:

```
mkdir /mnt/blumeta0/load
chmod 777 /mnt/blumeta0/load
```

Using OpenShift to copy data to the staging area

You can use an OpenShift command to load data into the staging area. For example:

```
oc rsync source-directory db2oltp-1592325602159-db2u-0:/mnt/blumeta0/load
```

Using an NFS mount or container-defined storage tools to copy data to the staging area

NFS

The system storage area, exposed as an NFS persistent volume, can be directly mounted through an NFS mount.

Portworx

Portworx has an import command that loads files into the volume from a directory. You can transfer flat files to the Portworx volume from a directory. The directory should be accessible by the server where the Portworx volume is located.

For example, the following command imports data from a Portworx volume `myVol` that is located in the */path/to/files* directory:

```
pxctl volume import --src /path/to/files myVol
```

For more information, see Create and manage volumes using pxctl in the Portworx documentation.

Load copy

Loaded data is copied by default to the `/mnt/bludata0/db2/copy` directory inside the Db2 container. After the loading process is completed, you can delete the data from this directory.

For more information, see Loading data

Using multiple databases in a Db2 deployment

You can run more than one database in a single Db2 deployment on Red Hat OpenShift by using a tool that enables you to add or drop all of the databases under that specific deployment.

About this task

You run the `manage_databases` tool after deploying the Db2 service. The following limitations apply:

- A maximum of eight databases are supported.
- All databases share the same Db2 instance registry and database manager configuration, and the basic characteristics of all the databases that you create by default use the settings in the `Db2uCluster` custom resource definition (CRD). But you can specify database-specific customizations by using an option in the tool to override the CRD.

Note: The `manage_databases` tool must be run under the `db2inst1` user ID.

Procedure

Use these commands with the `manage_databases` tool to add or drop databases within a single Db2 deployment:

Operation	Command
Add	You provide a list of database names and use the `--add` or `-a` option: `oc exec -it service_name -- manage_databases --dblist "database1_name,database2_name" --add` • *service_name*: The unique identifier for the Db2 deployment within Red Hat OpenShift, for example `c-db2oltp-1619534024988588-db2u-0`. • *database_name*: The name that you assign to each database.
Drop	You provide a list of database names and use the `--drop` or `-d` option: `oc exec -it service_name -- manage_databases --dblist "database1_name,database2_name" --drop`

To specify a custom configuration for a database, you specify the `--db-cfg-overrides` option followed by list of database-specific overrides in the following format:

```
'database1_name:parameter1@value|database2_name:parameter1@value,parameter2@value'
```

For example, the following command adds the databases `mydb1` and `mydb2` to the Db2 service deployment and specifies custom overrides that set the LOGARCHMETH1 configuration parameter to OFF for `mydb1` and the DFT_EXTENT_SZ and DBHEAP parameters to 128 and 1500 for `mydb2`:

```
oc exec -it c-db2oltp-1618958410448015-db2u-0 -- manage_databases --dblist "mydb1,mydb2" --db-
cfg-overrides "'mydb1:LOGARCHMETH1@OFF|mydb2:DFT_EXTENT_SZ@128,DBHEAP@1500'" --add
```

You can use the tool to display all databases within the deployment by name, alias, and local directory from the system database directory by using the `--show` or `-s`)option, for example:

```
oc exec -it c-db2oltp-1619534024988588-db2u-0 -- manage_databases --show
```

You can use the `--verbose` or `-v` option at the end of any command to provide additional debugging information in the output.

The `--help` or `-h` option provides command help.

Advanced configuration for demanding workloads

You can manually update the CRI-O configuration of the Red Hat OpenShift system where Db2 is running with or without Machine Config to handle advanced configurations and demanding workloads.

CRI-O is the OCI (Open Container Initiative) implementation of the Kubernetes Container Runtime Interface (CRI). CRI is an alternative to using Docker. To update OCP to use Db2, open the OCP console and follow the instructions from the console to change the CRI-O configuration.

Changing the CRI-O with Machine Config for Db2

You can use the Machine Config operator to manually update the OpenShift system where Db2 is running by using the OCP console or CLI.

About this task

CRI-O allows Kubernetes to use any Open Container Initiative (OCI)-compliant runtime to run pods. These pods share a namespace and live in the same control group (`cgroup`).

The Machine Config operator focuses on the operating system itself, managing updates to **`systemd`**, **`CRI-O/kubelet`**, **`kernel`**, **`NetworkManager`**, and the like. It also offers a **`MachineConfig`** CRD that can write configuration files onto the host.

Procedure

Set CRI-O max PIDs limit to 16384 by creating a **`ContainerRuntimeConfig`** resource.

The following example of a ContainerRuntimeConfig resource sets the PIDs limit to 16k on all worker nodes:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: crio-pids-limit
spec:
 machineConfigPoolSelector:
   matchLabels:
     pools.operator.machineconfiguration.openshift.io/worker: ''
 containerRuntimeConfig:
   pidsLimit: 16384
```

Changing the CRI-O without Machine Config for Db2

Some installations of OpenShift might not use the Machine Config operator. If your environment does not use the operator, you can still change the CRI-O configuration for Db2.

About this task

CRI-O allows Kubernetes to use any Open Container Initiative (OCI)-compliant runtime to run pods. These pods share a namespace and live in the same control group (`cgroup`).

Procedure

1. Set CRI-O max `Pid ID` to 16k or higher.
   a) Log in to the cluster by using the OpenShift CLI, which can be downloaded from https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/.
   b) From the console, click your user, and then click **Copy Login Command**.

      A new tab opens, and you might need to log in again.
   c) Click **Display Token** and copy the token.
   d) Log in from the console.

```
oc login --token=xxxxxxx --server=https://api.example.com:5402
```

e) Run **oc get no** to list the nodes in your cluster.

```
NAME       STATUS   ROLES           AGE  VERSION
10.0.0.1   Ready    master, worker   1d   v1.16.2
10.0.0.2   Ready    master, worker   1d   v1.16.2
10.0.0.3   Ready    master, worker   1d   v1.16.2
```

f) On each of the nodes, run the **oc debug** command.

```
oc debug node/node_name
```

Run the following command to check the value of the **pids_limit** on each node.

```
chroot /host
grep -i "pids_limit[ ]*="  /etc/crio/crio.conf
```

The output of the command shows you the current value:

```
pids_limit = 231205
```

If the line is not commented out and the value is greater than 16384, move to the next node.

If the line is commented out or the value is less than 16384, edit the file.

i) Open `/etc/crio/crio.conf` with a text editor, for example with Vim.

```
vi /etc/crio/crio.conf
```

ii) Find the line `pids_limit = xxx` (it might start with #) and change it to `pids_limit = 16384`.

g) When all of the nodes are done, restart each node one at a time. Check to see that everything is in good health by running the following command:

```
oc get no -l node-role.kubernetes.io/worker -o name
```

2. Enable **container_manage_cgroup** on the worker nodes.

Important: If your platform is OCP 4.6, then you must follow Step 2 in "Changing the CRI-O with Machine Config for Db2" on page 55 instead.

Using oc debug

Run the following command to set **container_manage_cgroup** to `true` on each of the worker nodes. For clusters with 10+ nodes, use **daemonset**.

```
oc get no -l node-role.kubernetes.io/worker --no-headers -o name | xargs -I {} --  oc
debug {} -- bash -c 'chroot /host setsebool -P container_manage_cgroup true'
```

Using a daemon set

a. Create a service account with the name `set-container-manage-cgroup` by running the following command:

```
oc create -f - <<EOFapiVersion: v1kind: ServiceAccountmetadata:  name: set-container-
manage-cgroup  namespace: kube-systemEOF
```

b. Give the service account privileged security context constraints (SCC) by running the following command:

```
oc adm policy add-scc-to-user privileged system:serviceaccount:kube-system:set-
container-manage-cgroup
```

c. Create the daemon set and set the **selinux boolean container_manage_cgroup** to true by running the following command:

```
oc create -f - << EOF
apiVersion: apps/v1
kind: DaemonSet
metadata:
   name: set-container-manage-cgroup
   namespace: kube-system
   labels:
     tier: management
     app: set-container-manage-cgroup
spec:
   selector:
     matchLabels:
       name: set-container-manage-cgroup
   template:
     metadata:
       labels:
         name: set-container-manage-cgroup
     spec:
       serviceAccountName: set-container-manage-cgroup
       initContainers:
         - command:
             - sh
             - -c
             - echo 1 > /sys1/fs/selinux/booleans/container_manage_cgroup; echo 1 > /
sys1/fs/selinux/commit_pending_bools;
           image: alpine:3.6
           imagePullPolicy: IfNotPresent
           name: sysctl
           resources: {}
           securityContext:
             privileged: true
           volumeMounts:
             - name: modifysys
               mountPath: /sys1
       containers:
         - resources:
             requests:
               cpu: 0.01
           image: alpine:3.6
           name: sleepforever
           command: ["/bin/sh", "-c"]
           args:
             - >
               while true; do
                 sleep 100000;
               done
       volumes:
         - name: modifysys
           hostPath:
             path: /sys
EOF
```

Note: The **container_manage_cgroup** is set to the default value after a restart.

Updating the Db2 password secrets

The Db2 administration accounts on your Red Hat OpenShift cluster are protected by password secrets. During deployment of Db2, these password secrets are automatically generated and securely stored for these accounts. These accounts are used by Db2 to handle administrative tasks on the database. If you need to change them to comply with specific password regulations, or if your security situation changes, you can use this method to update the password secrets at any point in time.

About this task

These commands update the passwords for both the Db2 instance user account, and the Db2 admin account, which is kept in the local Db2 LDAP service. This change must be run on the master node, and requires the **kubectl** or **oc** command. You need a user account with sufficient authority to run the **kubectl patch** and **kubectl delete** commands in the namespace where your Db2 instance is running.

Procedure

1. Assign the namespace to a variable:

```
NAMESPACE=
```

2. Run the following **kubectl** command to get the database instance identifier:

```
kubectl get pods -n NAMESPACE | grep db2oltp
```

Replace *NAMESPACE* with the namespace where your database instance is running on the cluster. The command returns a string that contains the instance identifier number:

```
c-db2oltp-1605722434029496-db2u-0
```

In this example, the instance identifier of the database is **1605722434029496**.

3. Run the following **kubectl** commands to update the password in the secret object.

```
kubectl patch -n NAMESPACE $(kubectl get secret -n NAMESPACE -o name | grep "DB2UCLUSTER_ID-
instancepassword") \
-p $"{\"data\":{\"password\": \"$(echo NEW_PASSWORD | base64)\"}}"
```

Replace *NAMESPACE* with the namespace where your Db2 instance is running and *DB2UCLUSTER_ID* with the numerical identifier that was returned from the previous step. Replace *NEW_PASSWORD* with the new password for the Db2 instance.

4. Restart the affected cluster pods.

In the following commands, replace *NAMESPACE* with the namespace where your Db2 instance is running and *DB2UCLUSTER_ID* with the numerical identifier that was returned from the earlier command.

- Restart the Db2 database pods:

```
kubectl delete -n NAMESPACE $(kubectl get po -n NAMESPACE -o name | grep -E
"DB2UCLUSTER_ID-db2u-[0-9]")
```

- Restart the Db2 tools pod:

```
kubectl delete -n NAMESPACE $(kubectl get po -n NAMESPACE -o name | grep "DB2UCLUSTER_ID-
db2u-tools")
```

Exec into the Db2 pod

To do some common administration tasks, you must be able to run the exec command on the Db2 pod.

Before you begin

In the following procedure, the placeholder <namespace> is used represent the namespace used for the respective database deployments.

Procedure

1. Find the name of your Db2 pod for your database deployment:

```
db2_podname=$(oc -n <namespace> get po --selector name=dashmpp-head-0)
```

or

```
db2_podname=$(kubectl -n <namespace> get po --selector name=dashmpp-head-0)
```

2. Run the **exec** command to exec into the Db2 pod:

```
oc -n namespace exec -it ${db2_podname} -- bash
```

or

```
kubectl -n namespace exec -it ${db2_podname} -- bash
```

Stopping and starting a Db2 instance

Stop and start a Db2 instance inside a container to perform maintenance tasks.

About this task

Db2 comes with a built-in High Availability (HA) solution that monitors and triggers recovery actions if there is a problem with Db2 database services. The built-in HA monitoring must be temporarily disabled before you stop and start Db2.

Procedure

1. "Exec into the Db2 pod" on page 58.
2. Temporarily disable the built-in HA:

   ```
   sudo wvcli system disable -m "Disable HA before Db2 maintenance"
   ```

3. Become the Db2 instance owner:

   ```
   su - ${DB2INSTANCE}
   ```

4. Run the **db2stop** command and perform the maintenance tasks that you want.
5. When the maintenance tasks are completed, restart Db2 with the **db2start** command, and activate the database (for example, with the command `db2 activate db BLUDB`).
6. As the root user, re-enable the built-in HA monitoring:

   ```
   sudo wvcli system enable -m "Enable HA after Db2 maintenance"
   ```

7. Confirm that the built-in HA monitoring is active:

   ```
   sudo wvcli system status
   ```

   ```
   sudo wvcli system devices
   ```

   Tip: You might have to run these commands several times before the HA status is shown as active.

Monitoring Db2 startup after rebooting a node

When a node is restarted, confirm that Db2 successfully starts up by monitoring the startup progress.

About this task

Db2 should automatically start up after a Kubernetes worker node is started and the kubelet starts all the pods on the node. You can run a series of commands to monitor the startup progress.

Procedure

1. Run the command `kubectl get nodes` and check the output to verify the following:
   - The Kubernetes worker nodes that were rebooted are in a ready state.
   - All nodes with the etcd operator role are in a ready state.
2. Monitor the Db2 pod startup progress:

   ```
   kubectl -n <namespace> get po | grep db2u
   ```

3. Check that all Db2 database services pods are in a running state:

```
kubectl -n <namespace> get po --selector type=engine
```

4. Confirm that the Db2 instance is healthy.
   a) “Exec into the Db2 pod” on page 58.
   b) Become the Db2 instance owner:

```
su - ${DB2INSTANCE}
```

   c) Run the following commands to validate the health of the Db2 instance:

```
db2gcf -s   # Should show 'Active'
```

```
db2pd - -member all | grep Member   # Should show an entry for each Db2 partition
```

```
db2pd -db bludb - -member all | grep Member   # Should show an entry for each Db2
partition
```

```
db2 list active databases   # Should show BLUDB database active
```

```
db2 connect to BLUDB; db2 connect reset # Confirm you can connect to the database
```

Locating the keystore for a Db2 database

You can use a database manager configuration parameter to locate the encryption keystore for a Db2 database on Red Hat OpenShift.

About this task

The keystore is used for Db2 native encryption for data at rest. You might need to know the keystore location if you are performing a backup of the database with the intention of restoring it on a different deployment. You can also back up the keystore separately and not necessarily as part of a restore operation.

Procedure

1. Exec into the Db2 pod.
2. Become the Db2 instance owner:

```
su - ${DB2INSTANCE}
```

3. Find the KEYSTORE_LOCATION database manager configuration parameter:

```
db2 get dbm cfg | grep KEYSTORE_LOCATION
```

   The command returns a value that is similar to the following example:

```
Keystore location   (KEYSTORE_LOCATION) = /mnt/blumeta0/db2/keystore/keystore.p12
```

Backing up and restoring Db2

You can use the **db2 backup** and **db2 restore** commands to back up and restore your integrated Db2 instance.

Important notes:

- IBM recommends that you run full online backups of your entire database on a daily basis.
- Do not stop the container while a backup or a restore operation is in progress.

- Use the **db2 backup** and **db2 restore** commands to run full online backup and restore operations on a Db2 database.
- To restore the database in a different deployment or cluster, ensure that you export the encryption keystore with your backup images.

For more information about Db2 backup and restore operations, see:

- Db2 backup command documentation
- Db2 restore command documentation

Preparing to back up or restore Db2

Before you can run the db2 backup or db2 restore commands, you need to identify the node where the Db2 database service is deployed.

Procedure

1. To identify the Db2 container, run the following command on any node in your cluster:

   ```
   kubectl get pods -n namespace | grep db2u
   ```

   Replace *namespace* with the cluster namespace where the database was deployed.

   This command returns a list of one or more nodes for the Db2 database configuration. For a single-node configuration, the catalog node is always the same as physical node.

   A sample SMP configuration:

   ```
   db2oltp-1556786589-db2u-0                                 1/1     Running     0          3h54m
   ```

2. Exit that bash shell and open a new shell to the single node that hosts your Db2 database service.

   ```
   kubectl exec -it -n namespace db2oltp-node -- bash
   ```

   Replace *db2oltp-node* with the name of the node that you found in the previous step.

What to do next

After you open a bash shell, you can proceed with "Backing up a Db2 database offline" on page 63 or "Restoring Db2 by using commands" on page 71.

Backing up a Db2 database

You can backup your Db2 database on Red Hat OpenShift either in online or offline mode.

Performing a snapshot backup with Db2 container commands

You can run container commands against your Db2 on OpenShift instance to suspend write operations and take a snapshot backup of a Db2 database.

About this task

Snapshot backups have significant advantages over traditional backups. They are completed quickly, regardless of database size, and they allow for fast recovery. Also, snapshots do not affect many database operations, except momentarily pausing operations that perform writes.

Procedure

1. Suspend operations on the database:

   ```
   oc exec -it c-db2oltp-1611589677085880-db2u-0 -- manage_snapshots --action suspend
   ```

The **manage_snapshots** command might prompt you with a message that is similar to the example below:

```
The authenticity of host '[c-db2oltp-1611589677085880-db2u-0.c-db2oltp-1611589677085880-db2u-
internal]:50022 ([10.254.13.10]:50022)' can't be established.
RSA key fingerprint is SHA256:R9/ve/iF4j+2BKunEqEo5PGN19UoCYBsnbHWmEmlrbU.
RSA key fingerprint is MD5:72:cb:84:3a:fa:36:0c:ed:45:96:1b:cd:f9:d4:67:99.
Are you sure you want to continue connecting (yes/no)?
```

If you see this message, type `yes` in the shell prompt to enable the script to update the `known_hosts` file to enable password-less SSH access.

2. Confirm that Db2 high availability monitoring is disabled and the database is suspended by issuing the following command:

```
oc exec -it c-db2oltp-1611589677085880-db2u-0 -- wvcli system status
```

3. Take the snapshot of the persistent volume claim and volume claim templates at the storage layer. The steps differ depending on your storage provider:
   - OpenShift Container Storage
   - Portworx
   - IBM Spectrum Scale CSI
   - IBM Cloud File Storage
4. Resume writes to the database:

```
oc exec -it c-db2oltp-1611589677085880-db2u-0 -- manage_snapshots --action resume
```

5. Confirm that Db2 high availability monitoring is re-enabled and the database is taken out of write suspend by issuing the following command:

```
oc exec -it c-db2oltp-1611589677085880-db2u-0 -- wvcli system status
```

What to do next

For instructions on the recommended restore option, see “Using the restore script” on page 69.

Backing up a Db2 database online

For a Db2 deployment, the method for the *BLUDB* database backup is to use a Single System View (SSV) backup. This strategy helps you back up all database partitions simultaneously, including the catalog partition. SSV backups provide a single time-stamp for all database partitions, making the recovery simpler. The db2 `backup` command must be run from the node where the Db2 database service is deployed.

Before you begin

- For an online backup, the database is online and accessible to applications during backup. Changes that are made to the database during the backup process are included in the backup.
- You must identify the Db2 catalog node and have an SSH connection to the catalog node. For more information, see “Preparing to back up or restore Db2” on page 61.

Procedure

1. From a bash shell on the catalog node, switch to the database instance owner and create the directory to hold the backup images:

```
su - db2inst1
mkdir /backup_dir/backup_nnn
```

Where *backup_dir* is the directory that you create to hold backup images and *nnn* is an incremental value that puts each backup in a separate subdirectory. The backups can be put into any directory on

the cluster that is shared by the container nodes and has sufficient space to hold the backups. If you back up into a different directory, alter these commands as necessary.

2. Connect to the database.

```
db2 connect to BLUDB
```

3. Run the online backup:

```
db2 backup db BLUDB on all dbpartitionnums online to backup_dir include logs without
prompting
```

Where *backup_dir* is the full path to the directory that you created in Step 1.

The online backup is run and returns a successful result:

```
Part  Result
----  ------------------------------------------------------------------------
0000  DB20000I  The BACKUP DATABASE command completed successfully.
0001  DB20000I  The BACKUP DATABASE command completed successfully.
0002  DB20000I  The BACKUP DATABASE command completed successfully.

Backup successful. The timestamp for this backup image is : 20190523204048
```

The database backup is complete.

What to do next

For instructions on the recommended restore option, see “Using the restore script” on page 69.

Backing up a Db2 database offline

For a Db2 deployment, the method for the *BLUDB* database backup is to use a Single System View (SSV) backup. This strategy helps you back up all database partitions simultaneously, including the catalog partition. SSV backups provide a single time-stamp for all database partitions, making the recovery simpler. The db2 backup command must be run from the node where the Db2 database service is deployed.

Before you begin

- For an offline backup, the database is offline and inaccessible to applications during backup, but there are no ongoing transactions that need to be added to the backup.
- You must identify the Db2 catalog node and have an SSH connection to the catalog node. For more information, see “Preparing to back up or restore Db2” on page 61.

Procedure

1. From a bash shell on the catalog node, switch to the database instance owner and create the directory to hold the backup images:

```
su - db2inst1
mkdir /backup_dir/backup_nnn
```

Where *backup_dir* is the directory that you create to hold backup images and *nnn* is an incremental value that puts each backup in a separate subdirectory. The backups can be put into any directory on the cluster that is shared by the container nodes and has sufficient space to hold the backups. If you back up into a different directory, alter these commands as necessary.

2. Temporarily disable the built-in HA:

```
sudo wvcli system disable -m "Disable HA before Db2 maintenance"
```

3. Connect to the database:

```
db2 connect to BLUDB
```

4. Find all the applications that are connected to Db2:

   ```
   db2 list applications
   ```

   This command returns a list of all currently connected applications. You can either stop all the connections by closing the applications, or you can enter the following command to disconnect all connections:

   ```
   db2 force application all
   ```

5. Terminate the database:

   ```
   db2 terminate
   ```

6. Stop the database:

   ```
   db2stop force
   ```

   Ensure that the command completes on all nodes.

7. Ensure that all Db2 interprocess communications are cleaned for the instance:

   ```
   ipclean -a
   ```

8. Turn off all communications to the database by setting the value of the DB2COMM variable to null:

   ```
   db2set -null DB2COMM
   ```

9. Restart the database in restricted access mode:

   ```
   db2start admin mode restricted access
   ```

10. Run the offline backup with the command:

    ```
    db2 backup db BLUDB on all dbpartitionnums to backup_dir
    ```

    Where *backup_dir* is the full path to the directory you created in Step 1.

    The offline backup is run and returns a successful result:

    ```
    Part  Result
    ----  ------------------------------------------------------------------------
    0000  DB20000I  The BACKUP DATABASE command completed successfully.
    0001  DB20000I  The BACKUP DATABASE command completed successfully.
    0002  DB20000I  The BACKUP DATABASE command completed successfully.

    Backup successful. The timestamp for this backup image is : 20190523210916
    ```

11. Halt the restricted access mode:

    ```
    db2stop force
    ```

12. Ensure that all Db2 interprocess communications are cleaned for the instance:

    ```
    ipclean -a
    ```

13. Reinitialize the Db2 communication manager to accept database connections:

    ```
    db2set DB2COMM=TCPIP,SSL
    ```

14. Restart the database for normal operation:

    ```
    db2start
    ```

15. Activate the database:

    ```
    db2 activate db bludb
    ```

16. As the root user, re-enable the built-in HA monitoring:

```
sudo wvcli system enable -m "Enable HA after Db2 maintenance"
```

17. Confirm that the built-in HA monitoring is active:

```
sudo wvcli system status
sudo wvcli system devices
```

18. Connect to the database:

```
db2 connect to BLUDB
```

What to do next

For instructions on the recommended restore option, see “Using the restore script” on page 69.

Copying a Db2 backup to a different instance or remote cluster

To copy a backup to a different instance, you need to compress your backup images and add them to the backup directory you want to restore from.

Before you begin

You must complete the following tasks:

1. Backup a Db2 database offline. See Backing up a Db2 database offline for more details.
2. Restore your backup by using the restore script. See Using the restore script for more details.

Procedure

1. On the catalog node, compress your backup images and keystore with the following command:

   Important: You need to provide your keystore to be able to restore from a backup.

   ```
   su - db2inst1
   cd /backup_dir
   tar -cvzf backup_nnn.tgz -C /backup_dir/backup_nnn .

   # Examine the tar file and ensure it contains the backup images and keystore files
   tar -ztvf backup_nnn.tgz
   ```

   Perform an initial hash check with the following command and keep note of the output:

   ```
   sha1sum backup_nnn.tgz
   ```

2. Exit the session and copy the `tar` locally using the following command:

   ```
   mkdir /local/backup/dir/backup_nnn
   oc rsync <db2 backup catalog pod>:/backup_dir/backup_nnn.tgz /local/backup/dir/backup_nnn
   ```

   You can now safely delete the backup `tar` you created on the backup instance.

3. Copy the `tar` and add it to the backup directory of the instance you want to restore. Refer to the following example:

   ```
   oc rsync /local/backup/dir/backup_nnn <db2 restore catalog pod>:/backup_dir
   ```

   Perform another hash check with the following command:

   ```
   sha1sum backup_nnn.tgz
   ```

   Compare the output of this hash check with the output of the initial hash check and confirm they match.

   After confirming that the hash checks match, you can safely delete the backup `tar` you created on the backup instance.

4. From the catalog node of the restored instance, extract the `tar` to the backup directory. Refer to the following example:

```
su - db2inst1
sudo tar -xvzf backup_dir/backup_nnn/backup_nnn.tgz -C backup_dir/backup_nnn
```

   You can now safely delete the backup `tar` you copied to the restore instance.

What to do next

Complete the steps to restore an external database. See Using the restore script for more details.

Restoring a Db2 database

You can restore your Db2 database on Red Hat OpenShift either from an offline backup or online backup.

Note: To restore the database in a different deployment or cluster, ensure that you export the encryption keystore with your backup images.

Performing a snapshot restore with Db2 container commands in Db2

You can restore a snapshot backup of your Db2 database by putting your database in maintenance mode and restoring your volume snapshots.

About this task

Important: In running these steps, you might be at risk of losing your data unless you take the mentioned precautions.

Procedure

1. Put your cluster in maintenance mode:

```
DB2U_CLUSTER=$(oc get db2ucluster --no-headers | head -n1 | awk '{print $1}'); echo
$DB2U_CLUSTER
```

```
oc annotate db2ucluster $DB2U_CLUSTER db2u.databases.ibm.com/maintenance-pause-
reconcile=true --overwrite
```

2. Scale down your deployment:

```
NUM_REPLICAS=$(oc get sts ${DB2U_STS} -ojsonpath={.spec.replicas})
```

```
DB2U_STS=$(oc get sts --selector="app=${DB2U_CLUSTER},type=engine" --no-headers | awk
'{print $1}'); echo $DB2U_STS
```

```
ETCD_STS=$(oc get sts --selector="app=${DB2U_CLUSTER},component=etcd" --no-headers | awk
'{print $1}'); echo $ETCD_STS
```

```
TOOLS_DEPLOY=$(oc get deploy --selector="app=${DB2U_CLUSTER},role=tools" --no-headers | awk
'{print $1}'); echo $TOOLS_DEPLOY
```

```
oc scale sts ${DB2U_STS} --replicas=0
```

```
oc scale sts ${ETCD_STS} --replicas=0
```

```
oc scale deploy ${TOOLS_DEPLOY} --replicas=0
```

3. Copy the existing PVC definitions:

```
oc label pvc c-${DB2U_CLUSTER}-meta app=${DB2U_CLUSTER}
```

```
oc label pvc c-${DB2U_CLUSTER}-backup app=${DB2U_CLUSTER}
```

```
oc label pvc c-${DB2U_CLUSTER}-share app=${DB2U_CLUSTER}
```

```
for PVC in $(oc get pvc -l app=$DB2U_CLUSTER --no-headers | awk '{print $1}'); do oc get pvc
$PVC -oyaml > ${PVC}_copy.yaml ; done
```

4. Set your existing PVs reclaim policy to `retain` in case something goes wrong:

```
for PV in $(oc get pv --no-headers | grep $DB2U_CLUSTER | awk '{print $1}'); do oc patch pv
$PV -p "{\"spec\":{\"persistentVolumeReclaimPolicy\":\"Retain\"}}" ; done
```

   Important: Manually verify that each PV is set to `retain` or you are at risk of losing your data.

5. Delete the existing PVCs:

```
oc delete pvc -l app=${DB2U_CLUSTER}
```

   CAUTION: Do this step only if you are confident that you are able to restore your volumesnapshots, and your PV reclaim policies are set to `retain`. If you are unsure, try the next step instead by restoring your volumesnapshots to dummy PVCs with different names first.

6. Restore your volumesnapshot to PVCs with the same name and namespace that you copied in step "3" on page 66.

   The steps will differ depending on your provider:

   - OpenShift® Data Foundation
   - Portworx
   - IBM Spectrum® Scale CSI
   - IBM® Cloud File Storage

7. Scale up your deployment:

```
oc scale deploy ${TOOLS_DEPLOY} --replicas=1
```

```
oc scale sts ${ETCD_STS} --replicas=${NUM_REPLICAS}
```

```
oc scale sts ${DB2U_STS} --replicas=${NUM_REPLICAS}
```

8. Bring the database out of write-suspend after restoring:

```
CATALOG_POD=$(oc get po -l name=dashmpp-head-0,app=${DB2U_CLUSTER} --no-headers | awk
'{print $1}')
```

```
oc exec -it ${CATALOG_POD} -- manage_snapshots --action restore
```

9. Exit maintenance mode:

```
oc annotate db2ucluster $DB2U_CLUSTER db2u.databases.ibm.com/maintenance-pause-
reconcile=true --overwrite
```

Restoring Db2 from an online backup using commands

This process restores a database that was backed up by using the **db2 backup** command. The restored database starts back the same state that it was in when the online backup was made.

Before you begin

- You must have an existing Db2 backup that is accessible on the Red Hat OpenShift system.
- You must identify the Db2 catalog node and data nodes and have an SSH connection open to each node. For more information, see "Preparing to back up or restore Db2" on page 61.

About this task

For a Db2 deployment, the **db2 restore** command can be started only from the catalog node but each available multiple logical node (MLN) participates in the restore operation.

Restriction: A full database restore operation can be run only in offline mode.

Procedure

1. From a bash shell on each of the nodes specified in the db2nodes.cfg file, switch to the database instance owner. Confirm that the directory that contains the database backup exists and is mounted. For example:

   ```
   su - db2inst1
   cd /backup_dir/backup_nnn
   ```

   Where *backup_dir* is the directory that you create to hold backup images and *nnn* is an incremental value that was used to put each backup in a separate subdirectory. The backups can be in any directory on the cluster that is shared by the container nodes and has sufficient space to hold the backups. If you restore a backup from a different directory location, alter these commands as necessary.
2. Temporarily disable the built-in HA:

   ```
   sudo wvcli system disable -m "Disable HA before Db2 maintenance"
   ```
3. Connect to the database:

   ```
   db2 connect to BLUDB
   ```
4. On the catalog node, find all the applications that are connected to Db2:

   ```
   db2 list applications
   ```

   This command returns a list of all currently connected applications. You can either stop all the connections by closing the applications, or you can enter the following command to disconnect all connections:

   ```
   db2 force application all
   ```
5. Terminate the database:

   ```
   db2 terminate
   ```
6. Stop the database:

   ```
   db2stop force
   ```

   Ensure that the command completes on all nodes.
7. Ensure that all Db2 interprocess communications are cleaned for the instance:

   ```
   ipclean -a
   ```
8. Turn off all communications to the database by setting the value of the DB2COMM variable to null:

   ```
   db2set -null DB2COMM
   ```
9. Restart the database in restricted access mode:

   ```
   db2start admin mode restricted access
   ```
10. Run the restore operation:
    a) On the catalog node, run the following command:

```
db2 RESTORE DATABASE BLUDB FROM backup_dir TAKEN AT backup_image_timestamp INTO BLUDB
REPLACE EXISTING WITHOUT PROMPTING
```

Where *backup_dir* is the full path to the directory where your backup images are located, and *backup_image_timestamp* is the associated timestamp on those backup image files.

b) After the commands are complete, rollforward the database by running the following command on the catalog node:

```
db2 rollforward db BLUDB to end of backup on all dbpartitionnums and stop
```

11. Stop the database:

```
db2stop force
```

Ensure that the command completes on all nodes.

12. Ensure that all Db2 interprocess communications are cleaned for the instance:

```
ipclean -a
```

13. Reinitialize the Db2 communication manager to accept database connections:

```
db2set DB2COMM=TCPIP,SSL
```

14. Restart the database for normal operation:

```
db2start
```

The online database restore operation is complete.

15. Activate the database:

```
db2 activate db bludb
```

16. Re-enable the Wolverine high availability monitoring process:

```
wvcli system enable -m "Enable HA after Db2 maintenance"
```

17. Connect to the database:

```
db2 connect to BLUDB
```

What to do next

For more information about advanced command options, see the RESTORE DATABASE command

Restoring Db2 from an offline backup

If you created an offline backup, you can use a provided script file to restore Db2 backup images (including different form factor or version), or you can use restore commands when you are restoring from a Db2 backup in Red Hat OpenShift at a similar version.

Using the restore script

You can use a provided script file to restore Db2 backup images, with or without encryption.

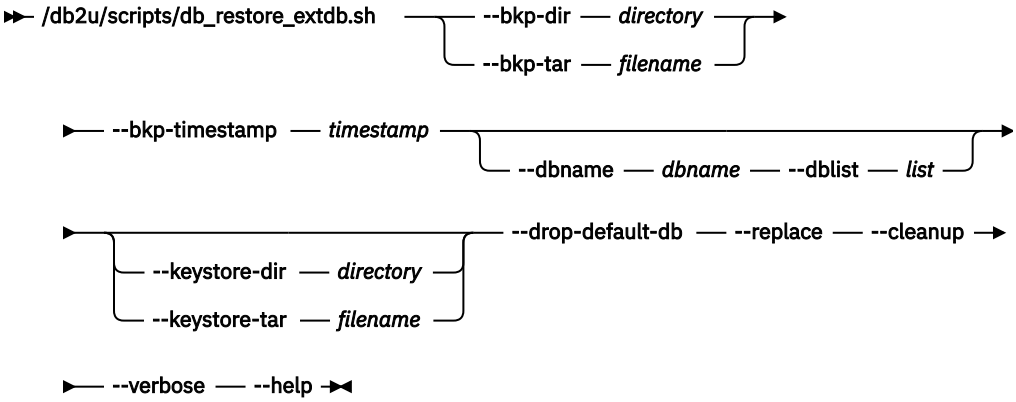
The restore script can restore:

- An encrypted backup
- An unencrypted backup
- An external Db2 database from the same or different form factor (such as a bare metal Db2 deployment within the same platform family)
- The backup from an earlier Db2 version

The following sections provide usage information for the `db_restore_extdb.sh` script, and details on using the script to restore with different options.

Note: You can only use the script to restore full offline backup images. If you are using incremental/online backup images, use the native Db2 restore commands.

Syntax



Options:

`--bkp-dir` ***directory***
Specifies the directory where the backup images are located. Choose this option or the **`--bkp-tar`** option.

`--bkp-tar` ***filename***
Specifies a `.tar` file that contains the backup images.

`--bkp-timestamp` ***timestamp***
The timestamp of the backup image to use for the restore.

`--dbname` ***dbname***
The name of the database to use when restoring the backup image.

`--dblist`
A comma-separated list of database names to use when restoring the backup image.

`--keystore-dir` ***directory***
The directory where the backup native encryption keystore is located. Choose this option or the **`--keystore-tar`** option.

`--drop-default-db`
Drops the existing databases.

`--replace`
Replaces the existing database.

`--keystore-tar`
Specifies a `.tar` file that contains the backup native encryption keystore.

`--cleanup`
Specifies to clean up all backup images.

`--verbose`
Use this flag to view explicit commands that the script is performing or to view errors.

`--help`
Use this flag to display help for the tool.

Restoring an encrypted backup

To restore an encrypted backup, follow these steps:

1. Save the Db2 database backup images and the native encryption keystore in a single location. For example, `/mnt/blumeta0/db2/backup/db2ubkp`, which is equivalent to `${BACKUPDIR}/db2ubkp`.
2. The Db2 instance owner/group must have read-write access to the backup directory where the backup images and keystore files are saved. You can set the required mode-bits and permissions by issue a command:

   ```
   chmod 755 ${BACKUPDIR}/db2ubkp
   chown db2inst1:db2iadm1 ${BACKUPDIR}/db2ubkp
   ```

3. Run the following command to restore the backup:

   ```
   db_restore_extdb --bkp-dir ${BACKUPDIR}/db2ubkp --dbname dbname --bkp-timestamp timestamp --
   replace --keystore-dir ${BACKUPDIR}/db2ubkp
   ```

   Replace *dbname* with the case-sensitive database name, and *timestamp* with the timestamp of the backup image. You can obtain the output log for the restore command by running `cat ${SUPPORTDIR}/db_restore_extdb.log`.

Restoring an unencrypted backup

To restore an unencrypted backup, follow these steps:

1. Save the Db2 database backup images in a single location. For example, `/mnt/blumeta0/db2/backup/db2ubkp`, which is equivalent to `${BACKUPDIR}/db2ubkp`.
2. The Db2 instance owner/group must have read-write access to the backup directory where the backup images and keystore files are saved. You can set the required mode-bits and permissions by issue a command:

   ```
   chmod 755 ${BACKUPDIR}/db2ubkp
   chown db2inst1:db2iadm1 ${BACKUPDIR}/db2ubkp
   ```

3. Run the following command to restore the backup:

   ```
   db_restore_extdb --bkp-dir ${BACKUPDIR}/db2ubkp --dbname dbname --bkp-timestamp timestamp --
   replace
   ```

   Replace *dbname* with the case-sensitive database name, and *timestamp* with the timestamp of the backup image. You can obtain the output log for the restore command by running `cat ${SUPPORTDIR}/db_restore_extdb.log`.

Restoring an external database

The backup image (and keystore, if encrypted) can also be restored from an external source. For example, you might want to re-initialize Db2 by using a database from an on-premises deployment.

You can use the **--dbname** *dbname* option to set the database name to the same as the source in this scenario.

Restoring Db2 by using commands
This process restores a database that was backed up by using the **db2 backup** command. The restored database starts back the same state that it was in when the backup was made.

Before you begin

- You must have an existing Db2 backup that is accessible on the Red Hat OpenShift cluster.
- You must identify the Db2 catalog node and data nodes and have an SSH connection open to each node. For more information, see "Preparing to back up or restore Db2" on page 61.

About this task

The restore script is the preferred strategy to restore because it provides:

- A simple flow
- A restore strategy for full offline backup images
- Native support for restoring encrypted databases

The commands in this procedure are illustrative only and are only applicable when you are restoring from a Db2 backup in Red Hat OpenShift at a similar version. Further commands are required for restoring an encrypted database. For most restore operations, the best practice is to use the procedure in "Using the restore script" on page 69.

Note: If your backup and restore strategy is based on full offline backup images, you can use a provided script for restoring instead of using the process in this topic. These commands are only applicable when you are restoring from a Db2 backup in Red Hat OpenShift at a similar version. Otherwise, further commands are required, as detailed in ../../../com.ibm.db2.luw.admin.ha.doc/doc/t0006242.ditaUsing restore. For most restore operations, the best practice is to use the procedure in "Using the restore script" on page 69.

For a Db2 deployment, the **db2 restore** command can be started only from the catalog node, but each available multiple logical node (MLN) participates in the restore operation.

Restriction: A full database restore operation can be run only in offline mode.

Procedure

1. Exec into the Db2 pod.
2. Temporarily disable the built-in HA:

   ```
   sudo wvcli system disable -m "Disable HA before Db2 maintenance"
   ```

3. From a bash shell on each of the nodes specified in the db2nodes.cfg file, switch to the database instance owner. Confirm that the directory that contains the database backup exists and is mounted. For example:

   ```
   su - db2inst1
   cd /backup_dir/backup_nnn
   ```

   Where *backup_dir* is the directory that you create to hold backup images and *nnn* is an incremental value that was used to put each backup in a separate directory. The directory that is shown here is only an example. The backups can be in any directory on the cluster that is shared by the container nodes and has sufficient space to hold the backups. If you restore a backup from a different directory location, alter these commands as necessary.
4. Connect to the database:

   ```
   db2 connect to BLUDB
   ```

5. On the catalog node, find all the applications that are connected to Db2:

   ```
   db2 list applications
   ```

   This command returns a list of all currently connected applications. You can either stop all the connections by closing the applications, or you can enter the following command to disconnect all connections:

   ```
   db2 force application all
   ```

6. Terminate the database:

   ```
   db2 terminate
   ```

7. Stop the database:

```
db2stop force
```

Ensure that the command completes on all nodes.

8. Ensure that all Db2 interprocess communications are cleaned for the instance:

```
ipclean -a
```

9. Turn off all communications to the database by setting the value of the DB2COMM variable to null:

```
db2set -null DB2COMM
```

10. Restart the database in restricted access mode:

```
db2start admin mode restricted access
```

11. Run the restore operation. On the catalog node, run the following command:

```
db2 RESTORE DATABASE BLUDB FROM backup_dir TAKEN AT backup_image_timestamp INTO BLUDB
REPLACE EXISTING WITHOUT ROLLING FORWARD
```

Where *backup_dir* is the full path to the directory where your backup images are located, and *backup_image_timestamp* is the associated timestamp on those backup image files.

12. Stop the database:

```
db2stop force
```

Ensure that the command completes on all nodes.

13. Ensure that all Db2 interprocess communications are cleaned for the instance:

```
ipclean -a
```

14. Reinitialize the Db2 communication manager to accept database connections:

```
db2set DB2COMM=TCPIP,SSL
```

15. Restart the database for normal operation:

```
db2start
```

The offline database restore operation is complete.

16. Activate the database:

```
db2 activate db bludb
```

17. Re-enable the Wolverine high availability monitoring process:

```
wvcli system enable -m "Enable HA after Db2 maintenance"
```

18. Connect to the database:

```
db2 connect to BLUDB
```

What to do next

For more information about advanced command options, see the RESTORE DATABASE command

Backing up and restoring Db2 with IBM Spectrum Protect (Tivoli Storage Manager)

You can back up Db2 on Red Hat OpenShift by using IBM Spectrum Protect, formerly known as Tivoli Storage Manager.

Before you begin

Ensure that the SELinux module is installed on all worker nodes where Db2 is installed. For more details, see "Requirements for Db2 on SELinux" on page 22.

Procedure

Downloading the IBM Spectrum Protect client

1. Go to Install the UNIX and Linux backup-archive clients and click the client installation option that is specific to your architecture. For example, *Installing the Linux x86_64 client*. Follow the links to the download page.
2. Copy the `8.x.x.x-TIV-TSMBAC-LinuxX86.tar` file into the Db2 catalog node, where 8.x.x.x represents the client version

```
DB2U_POD=$(oc get po --no-headers -l name=dashmpp-head-0 | awk '{print $1}')
oc cp 8.x.x.x-TIV-TSMBAC-LinuxX86.tar $DB2U_POD:/tmp/
```

3. From within the pod, extract the contents of the compressed file and navigate to the target directory:

```
tar -xvf 8.x.x.x-TIV-TSMBAC-LinuxX86.tar -C target-directory
```

4. As a root user, install the GSKit packages:

```
sudo rpm -U gskcrypt64-8.x.x.x.linux.x86_64.rpm gskssl64-8.x.x.x.linux.x86_64.rpm
```

5. As a root user, install the IBM Spectrum Protect API:

```
sudo rpm -ivh TIVsm-API64.x86_64.rpm
```

6. As a root user, install the backup-archive client components:

```
sudo rpm -ivh TIVsm-BA.x86_64.rpm
```

Configuring the IBM Spectrum Protect client and server

Perform the following steps for each node in your configuration:

7. Add the following environment variables to the db2inst1 user profile:

```
cat <<EOF > ${BLUMETAHOME}/db2inst1/sqllib/userprofile
export GSK_STRICTCHECK_CBCPADBYTES=GSK_TRUE
DSMI_DIR=/opt/tivoli/tsm/client/api/bin64
DSMI_CONFIG=/opt/tivoli/tsm/client/api/bin64/dsm.opt
DSMI_LOG=$DIAGPATH
export DSMI_DIR DSMI_CONFIG DSMI_LOG
EOF
```

8. Create the directory structure and assign ownership of the IBM Spectrum Protect API password directory to db2inst1:

```
TSMDIR="/mnt/blumeta0/tsm"
TSM_API_DIR="${TSMDIR}/api"
TSM_BA_DIR="${TSMDIR}/ba"
mkdir -p ${TSM_API_DIR}/$(hostname -s)
mkdir -p ${TSM_BA_DIR}/$(hostname -s)
mkdir -p ${TSM_API_DIR}/$(hostname -s)/cred_store
sudo chown db2inst1:db2iadm1 ${TSM_API_DIR}/$(hostname -s)/cred_store
```

9. Create the configuration files (`dsm.opt` and `dsm.sys`) for the API:

Command to create `dsm.opt`:

```
cat << EOF >> api-directory/node-name/dsm.opt
servername server-name
EOF
```

- *api-directory*: The path to the directory structure that you created for the API. In this setup, it is `/mnt/blumeta0/tsm/api`.
- *node-name*: The name of the Db2 pod, for example `c-db2oltp-1609890842138524-db2u-0`.
- *server-name*: The name of the IBM Spectrum Protect server.

Command to create `dsm.sys`:

```
cat << EOF >> api-directory/node-name/dsm.sys
servername server-name
tcpserveraddress server-address
commmethod communication-method
passwordaccess generate
tcpport tcp-port
ssl yes|no
nodename node-name
passworddir api-directory/node-name/cred_store
errorlogname /mnt/blumeta0/db2/log/db2-dsmerror.log
EOF
```

The *api-directory* is `/mnt/blumeta0/tsm/api`.

- *server-address* is the IP address of the IBM Spectrum Protect server.
- *communication-method* would typically be tcpip.

10. If you want to back up the file system, create the configuration files (`dsm.opt` and `dsm.sys`) for the IBM Spectrum Protect backup archive (BA). This step is not needed if you only want to back up the database.

    Command to create `dsm.opt`:

    ```
    cat << EOF >> BA-directory/node-name/dsm.opt
    servername server-name
    EOF
    ```

    - *BA-directory*: The path to the directory structure that you created for BA. In this setup, it is `/mnt/blumeta0/tsm/ba`.

    Command to create `dsm.sys`:

    ```
    cat << EOF >> BA-directory/node-name/dsm.sys
    servername server-name
    tcpserveraddress server-address
    commmethod communication-method
    passwordaccess generate
    tcpport tcp-port
    ssl yes|no
    nodename node-name
    errorlogname /mnt/blumeta0/db2/log/db2-dsmerror.log
    EOF
    ```

11. Create symlinks to the IBM Spectrum Protect configuration files:

    ```
    ##Create symlink to API client configuration files
    sudo ln -sf tsmdir/api/$(hostname -s)/dsm.opt /opt/tivoli/tsm/client/api/bin64/dsm.opt
    sudo ln -sf tsmdir/api/$(hostname -s)/dsm.sys /opt/tivoli/tsm/client/api/bin64/dsm.sys
    ##Create symlink to BA client configuration files
    sudo ln -sf tsmdir/ba/$(hostname -s)/dsm.opt /opt/tivoli/tsm/client/ba/bin/dsm.opt
    sudo ln -sf tsmdir/ba/$(hostname -s)/dsm.sys /opt/tivoli/tsm/client/ba/bin/dsm.sys
    ```

    - *tsmdir*: The directory where you installed the IBM Spectrum Protect client (`/mnt/blumeta0/tsm`).

Registering the client with the server

12. Register your client with the IBM Spectrum Protect server by running the following command on the server in the admin mode:

```
register node client-hostname passw0rd backdel=yes maxnummp=10 SESSIONSECurity=transitional
```

- *client-hostname*: The hostname of the system where you installed the IBM Spectrum Protect client.

Authenticating

13. Switch to db2inst1 user:

```
su - db2inst1
```

14. Set up your password:

```
/mnt/blumeta0/home/db2inst1/sqllib/adsm/dsmapipw
```

When you successfully set up the password, your client is configured with the IBM Spectrum Protect server.

Running the backup

15. Issue one of the following commands to back up Db2 on the IBM Spectrum Protect server:

```
Online
db2 "backup db bludb on all dbpartitionnums online use tsm open <1-6> sessions include logs
without prompting"
Incremental
db2 "backup db bludb on all dbpartitionnums online incremental use tsm open <1-6> sessions
include logs without prompting"
Offline
db2 "backup db bludb on all dbpartitionnums use tsm open <1-6> sessions"
```

Restoring

16. Issue one of the following commands to restore Db2:

```
Online
- Head Node
db2_all '<<+0< db2 restore database bludb use tsm taken at timestamp logtarget
overflow_logpath replace existing without prompting'
- Data Nodes
db2_all '<<-0<|| db2 restore database bludb use tsm taken at timestamp logtarget
overflow_logpath replace existing without prompting'
Rollforward
db2 "rollforward db bludb to end of backup and stop overflow log path (overflow_logpath)
noretrieve"
Incremental
- Head Node
db2_all '<<+0< db2 restore database bludb incremental automatic use tsm taken at timestamp
logtarget overflow_logpath replace existing without prompting'
- Data Nodes
db2_all '<<-0<|| db2 restore database bludb incremental automatic use tsm taken at
timestamp logtarget overflow_logpath replace existing without prompting'
Rollforward
db2 "rollforward db bludb to end of backup and stop overflow log path (overflow_logpath)
noretrieve"
Offline
- Head Node
db2_all '<<+0< db2 restore database bludb use tsm taken at timestamp without rolling
forward without prompting'
- Data Nodes
db2_all '<<-0<|| db2 restore database bludb use tsm taken at timestamp without rolling
forward without prompting'
```

Velero Backup and Restore

Use this procedure to backup and restore the latest development version of Velero.

Note: Official Velero installation instructions can be found here.

The following procedures can be used to backup and restore your Velero implementation.

Install the db2u custom Velero plugin:

1. Copy the image tag from the list of releases, where **appVersion** is the installed version. Example: s11.5.8.0

```
VELERO_PLUGIN=$(oc get cm db2u-release -n $NAMESPACE -o=jsonpath='{.data.json}' | jq
'.databases.db2u."<appVersion>".images.veleroplugin'); echo $VELERO_PLUGIN
```

2. Add the plugin to the velero deployment.

```
velero plugin add $VELERO_PLUGIN
```

Perform a backup:

1. Identify the instance you wish to backup.

```
DB2U_CLUSTER=$(oc get db2ucluster --no-headers | head -n1 | awk '{print $1}'); echo
$DB2U_CLUSTER
DB2U_POD=$(oc get po --selector="app=${DB2U_CLUSTER},name=dashmpp-head-0" --no-headers | awk
'{print $1}'); echo $DB2U_POD
```

2. Put the cluster in maintenance mode.

```
oc -n $NAMESPACE annotate db2ucluster $DB2U_CLUSTER db2u.databases.ibm.com/maintenance-pause-
reconcile=true --overwrite
```

3. Suspend writes.

```
oc exec -it ${DB2U_POD} -- ksh -lc 'manage_snapshots --action suspend --retry 0'
```

4. Take the Velero backup using the label identifier, along with the velero configurable parameters. Example: **--default-volumes-to-restic**

```
velero backup create $BACKUP_NAME --selector formation_id=$DB2U_CLUSTER ...
```

5. Check the status of your backup and wait for it to complete.

```
velero backup describe $BACKUP_NAME
```

6. Take the cluster out of maintenance mode and resume writes.

```
oc -n standalone-rc annotate db2ucluster $DB2U_CLUSTER db2u.databases.ibm.com/maintenance-
pause-reconcile- --overwrite

oc exec -it ${DB2U_POD} -- ksh -lc 'manage_snapshots --action resume --retry 0'
```

Perform the restore:

1. Scale down the db2u-operator so it doesn't reconcile resources while you are restoring.

```
oc scale deployment db2u-operator-manager -n $OPERATOR_NAMESPACE --replicas=0
```

2. Restore from the backup.

```
velero restore create --from-backup $BACKUP_NAME
```

3. Wait for your restore to complete.

```
velero restore get
```

4. Identify the instance after the resources have been restored.

```
DB2U_CLUSTER=$(oc get db2ucluster --no-headers | head -n1 | awk '{print $1}'); echo
$DB2U_CLUSTER
DB2U_POD=$(oc get po --selector="app=${DB2U_CLUSTER},name=dashmpp-head-0" --no-headers | awk
'{print $1}'); echo $DB2U_POD
```

5. When your pods are ready, initiate the restore.

```
oc exec -it ${DB2U_POD} -- ksh -lc 'manage_snapshots --action restore --retry 0'
```

6. Scale up the operator and take your cluster out of maintenance mode.

```
oc scale deployment db2u-operator-manager -n $OPERATOR_NAMESPACE --replicas=1

oc -n standalone-rc annotate db2ucluster $DB2U_CLUSTER db2u.databases.ibm.com/maintenance-
pause-reconcile- --overwrite
```

Db2 high availability disaster recovery (HADR)

Db2 on Red Hat OpenShift supports Db2 high availability disaster recovery (HADR) to protect against data loss on various topologies, including primary and standby on the same or different Red Hat OpenShift clusters and within the same or different projects (namespaces).

HADR provides a high availability solution for both partial and complete site failures by replicating data changes from a source database, called the primary database, to the target databases, called the standby databases. Db2 on Red Hat OpenShift supports up to three remote standby servers.

Before you begin: To set up a Db2 HADR configuration on Red Hat OpenShift, you must install and deploy two or more Db2 database services that are at the same release level. Use the same database name for the deployed service instances.

Part of the setup process involves sharing the database backup image and keystore tar file between the primary and standby databases. If the databases are in the same OpenShift project in a single OpenShift cluster, it can be beneficial to use a backup volume that is shared between the primary and standby databases. In this case, you need to create the volume when you deploy Db2. If the databases are on different OpenShift projects and/or clusters, the databases can be deployed without shared volumes, and you can use rsync to copy assets.

Restriction: HADR for multiple databases in a single Db2 deployment is not supported. To enable HADR for multiple databases, create separate deployments for each database.

For full HADR documentation, see High availability disaster recovery (HADR).

Prerequisites for configuring HADR for Db2 on OpenShift

Before configuring HADR for the Db2 service, you must deploy two or more databases and set the same password for primary and standby deployments.

Before you set up a Db2 HADR configuration on Red Hat OpenShift, be sure that you do the following tasks:

- Install and deploy two or more Db2 database deployments that are at the same release level. These deployments will operate as the primary and standby databases. Use the same database name for all of the deployed databases that you will use in the Db2 HADR configuration.
- Ensure that the primary and the standby deployments have the same password. See “Updating the Db2 password secrets” on page 57 for details on updating passwords for the Db2 deployments.
- Ensure the database configuration value for LOGARCHMETH1 is not set to OFF.

Configuring and starting HADR

You can configure high availability disaster recovery (HADR) for a Db2 instance in a single Red Hat OpenShift project or in different OpenShift projects.

Setting up an etcd store for HADR

You must set up an etcd store to enable automated failover in a Db2 High Availability Disaster Recovery (HADR) configuration on Red Hat OpenShift.

Db2 HADR supports automated failover by using a mechanism called Governor. Governor relies on an etcd key-value store to keep track of HADR state information, such as the current leader (primary

node). Automated failover is only supported between the primary and principal standby, so a single etcd endpoint must be shared between the two deployments.

Note: If using the built-in etcd store with the etcd pod, create a network policy to allow communication from all deployments with the etcd pod. For more information, see Creating a network policy for built-in etcd.

The external etcd store can be a cloud provided service, deployed on the OpenShift® cluster, or hosted on-premise.

Important: For production deployments, a three-node etcd topology is required.

If etcd is installed on the same OpenShift cluster, it should not be co-located on the same worker nodes as the Db2 deployments. This can be achieved using pod anti-affinity in the etcd deployment.

Using the built-in etcd store (non-production, single OpenShift cluster environments only)

Db2 on Red Hat OpenShift includes a built-in etcd store. For development-only environments with all databases on the same OpenShift cluster, you can use the built-in etcd store from one of the deployments as the etcd endpoint for governor and HADR.

Important: The built-in etcd store is not to be used in production environments.

If you use the built-in etcd store with Db2 on Red Hat OpenShift, it might affect how Db2 is deployed. If the deployment is on a dedicated node, the etcd pod must be detached and moved to a different node. This node must not be the same as the node where the primary or standby deployments are running. This restriction ensures that if those nodes are shut down, etcd remains available.

To move the etcd pod:

1. Label and taint the desired node for etcd. This must be different from the node that the db2 deployments are running on. See Setting up dedicated nodes for your Db2 deployment for the required steps, but use a different label.
2. Scale down the etcd statefulset to 0:

   ```
   oc scale sts c-db2oltp-1573141715-etcd --replicas=0
   ```

3. Edit the etcd stateful set to match the new label.
4. Scale up the etcd statefulset to 1:

   ```
   oc scale sts c-db2oltp-1573141715-etcd --replicas=1
   ```

5. Determine the etcd endpoint:

   If the databases are all in the same OpenShift project:
   From the infrastructure or master node, discover the etcd service endpoint of the primary deployment, for example:

   ```
   oc get svc | grep etcd
   db2oltp-primary-etcd ClusterIP None <none> 2380/TCP,2379/TCP 5h
   db2oltp-standby-etcd ClusterIP None <none> 2380/TCP,2379/TCP 4h
   ```

   Assuming that your primary deployment name is db2oltp-primary, then the etcd endpoint to use is db2oltp-primary-etcd:2379 (port 2379 is the etcd client port).

   If the databases in different OpenShift projects:
   From the infrastructure or master node, discover the etcd service endpoint of the primary deployment in its OpenShift project, for example:

   ```
   oc get svc | grep etcd
   db2oltp-primary-etcd ClusterIP None <none> 2380/TCP,2379/TCP 5h
   ```

Assuming that your primary deployment name is db2oltp-primary, and the OpenShift project is zen, then the etcd endpoint to use is db2oltp-primary-etcd.zen:2379 (port 2379 is the etcd client port).

HADR with multiple standbys

Db2 on Red Hat OpenShift supports HADR with multiple standby databases. Using multiple standbys, you can have your data in more than two sites, which provides improved data protection with a single technology.

HADR allows you to have up to three standby databases in your setup. One of these databases must be designated as the principal HADR standby database; any other standby database is an auxiliary HADR standby database.

For full documentation on this configuration, see HADR multiple standby databases.

Restrictions

- Databases must be designated during configuration as primary, principal standby, and auxiliary standby. These roles cannot be changed without re-configuration
- Automated failover via Governor is only supported between the primary and principal standby databases
- Failover to an auxiliary standby database must be initiated manually.

Configuring with the Db2 HADR API

You can set up a Db2 high availability disaster recovery (HADR) configuration between databases that are on the same cluster and in the same Red Hat OpenShift project.

Before you begin

- Ensure you have the etcd endpoint (IP/name and port) available for HADR setup. See "Setting up an etcd store for HADR " on page 78.
- Part of the setup process involves sharing the database backup image and keystore tar file between the primary and standby databases, as described in Step "5" on page 81. You can perform this step either by creating a dynamically provisioned persistent volume or by using rsync to copy assets. If you are deploying Db2 for the purpose of using HADR, it can be beneficial to use a volume that is shared between the primary and standby databases. In this case, you need to create the volume when you deploy Db2. If you are setting up HADR on existing databases, using **rsync** might be preferable.

Important: When you start an HADR configuration, the backup images that are in the ${BACKUPDIR} path are deleted. To preserve existing backup images, move the images to a subdirectory under ${BACKUPDIR}. For example, ${BACKUPDIR}/backup_001.

About this task

After both Db2 services are deployed, follow these steps to set up the HADR configuration by using the Db2 HADR API, which is implemented by using an OpenShift custom resource definition (CRD).

Note: This procedure can only be used for two deployments that are in the same OpenShift project in the same cluster.

Procedure

1. Create a YAML file for the HADR custom resource definition on one of the master nodes and save the file, for example as /tmp/db2u_hadr.yaml.

   The following example shows the minimum fields that are required. You can set additional options by using the Db2 HADR API.

```
apiVersion: db2u.databases.ibm.com/v1alpha1
kind: Db2uHadr
metadata:
```

```
  name: example-hadr
spec:
  primary:
    db2uCluster: "db2u-oltp-1"
  standby:
    db2uCluster: "db2u-oltp-2"
  etcd:
    name: "cloud-etc-svc"
    port: 3056
```

You can use the **oc get Db2uCluster** command to get the names for the db2uCluster keyword.

Note: You can exclude the etcd fields in a development-only environment. In this case, the HADR setup uses the primary deployment's built-in etcd. See “Setting up an etcd store for HADR ” on page 78 for important considerations regarding this step.

2. Run the following command to create the CRD.

```
oc create -f /tmp/db2u_hadr.yaml
```

3. List the CRD:

```
oc get Db2uHadr
```

4. Use the following command to monitor the status of the CRD:

```
oc get db2uhadr db2u-oltp-hadr -oyaml | awk '/^status:/,/hadrSettings:/'
```

Here is an example status:

```
status:
  conditions:
  - lastTransitionTime: "2020-10-21T23:16:33Z"
    message: Creating Db2u HADR configuration file on Standby
    status: "False"
    type: Creating Db2u HADR configuration files
  - lastTransitionTime: "2020-10-21T23:17:37Z"
    message: Done
    status: "True"
    type: Setting up Primary database copy for HADR
```

After you see a similar status message, proceed to the next step.

5. Share the database backup image and keystore tar file between the primary and standby databases. You can perform this step in two ways:

- **Create a dynamically provisioned persistent volume**

  This volume is shared between the primary and standby databases. This volume is used by the CRD to share the database backup and keystore tar file.

- **Use rsync to copy assets to the standby deployment**

  When `type: Setting up Primary database copy for HADR` reaches the status `"True"`, start copying the backup image and keystore tar file into the backup directory on the standby deployment.

  For example, the following two commands copy the assets from a directory on the host called `/tmp/hadr` on the primary database to the standby database:

  ```
  oc rsync path_to_backup /tmp/hadr
  oc rsync /tmp/hadr/ path_to_backup
  ```

Results

When the CRD status phase shows as `Completed`, the HADR setup is complete, automatic client reroute is enabled, and HADR is started.

You can periodically issue the **oc get db2uhadr** command to monitor the status. The following examples show the command results, and then the CRD status when HADR setup is complete:

```
NAME             HADR PRIMARY              HADR STANDBY              STATE      AGE
db2u-oltp-hadr   db2u-oltp-1               db2u-oltp-1               Complete   7m8s
```

```
status:
  conditions:
  - lastTransitionTime: "2020-11-12T20:10:47Z"
    message: Creating Db2u HADR configuration file on Standby
    status: "False"
    type: Creating Db2u HADR configuration files
  - lastTransitionTime: "2020-11-12T20:12:54Z"
    message: Done
    status: "True"
    type: Enabling ACR feature for HADR
  - lastTransitionTime: "2020-11-12T20:12:56Z"
    message: Done
    status: "True"
    type: Setting up Db2u HADR completed
  - lastTransitionTime: "2020-11-12T20:11:33Z"
    message: Done
    status: "True"
    type: Setting up Primary database copy for HADR
  - lastTransitionTime: "2020-11-12T20:12:10Z"
    message: Done
    status: "True"
    type: Setting up Standby database copy for HADR
  - lastTransitionTime: "2020-11-12T20:12:46Z"
    message: Done
    status: "True"
    type: Starting HADR on the Primary database copy
  - lastTransitionTime: "2020-11-12T20:10:45Z"
    message: Setting up Db2u HADR completed
    status: "False"
    type: latestPhase
  hadrSettings:
```

Using the Db2 HADR API
The Db2 HADR API provides the interface required to configure and start HADR for two databases that are on the same cluster and in the same Red Hat OpenShift project. This API is supported by an OpenShift custom resource definition (CRD).

Configuring the primary and standby databases

When you set up HADR, you must define a primary and standby database. At a minimum, you must specify the **Db2uCluster** name for both the primary and standby. You can find these names by using the **oc get Db2uCluster** command.

You can also define the HADR services and specific endpoints to use for the HADR databases. If the services and endpoints do not exist, they will be created for you.

For example,

```
primary:
   db2uCluster: "db2u-oltp-primary"
   service:
     name: "db2u-oltp-primary-hadr-svc"
     port: 60006
standby:
   db2uCluster: "db2u-oltp-standby"
   service:
     name: "db2u-oltp-standby-hadr-svc"
     port: 60007
```

Configuring etcd

The Governor service that enables automatic failover of the HADR configuration relies on an etcd key-value store to keep track of the HADR state information, such as the current leader (primary node). A single etcd endpoint must be shared between the two HADR deployments. If these fields are not

provided, the default etcd setup is the primary deployment's built-in etcd cluster, but this is meant for development-only environments. Use this setting to define your own etcd cluster.

For example,

```
etcd:
   name: "cloud-etc-svc"
   port: 3056
```

Configuring HADR database configuration parameters

When configuring HADR, you can specify the **hadr_timeout**, **hadr_syncmode**, and **hadr_peer_window** configuration parameters. See HADR configuration parameters for more information.

For example:

```
dbConfig:
   timeOut: 90
   syncMode: "NEARSYNC"
   peerWindow: 120
```

Configuring HADR features

When configuring HADR, you can optionally disable automatic client reroute or enable reads on standby. You cannot use automatic client reroute if you enable reads on standby.

Note: Client reroute does not differentiate between writable databases (primary and standard databases) and read-only databases (standby databases). Configuring client reroute between the primary and standby might route applications to the database on which they are not intended to be run.

For example, the following example shows client reroute disabled and reads on standby enabled:

```
features:
   enableAutomaticClientReroute: false
   enableReadsOnStandby: true
```

Example of a complete Db2 HADR custom resource definition

The following example shows a complete Db2 HADR custom resource definition:

```
apiVersion: db2u.databases.ibm.com/v1alpha1
kind: Db2uHadr
metadata:
  name: db2u-oltp-hadr
spec:
  primary:
    db2uCluster: "db2u-oltp-primary"
    service:
      name: "db2u-oltp-primary-hadr-svc"
      port: 60010
  standby:
    db2uCluster: "db2u-oltp-standby"
    service:
      name: "db2u-oltp-standby-hadr-svc"
      port: 60012
  etcd:
    name: "cloud-etc-svc"
    port: 3056
  dbConfig:
    timeOut: 90
    syncMode: "NEARSYNC"
    peerWindow: 120
  features:
    enableAutomaticClientReroute: false
    enableReadsOnStandby: true
```

Deploying a Db2 HADR configuration by using a shared volume for backup storage
When you set up an HADR configuration, you can copy assets between the primary and standby databases by using a shared persistent volume claim for backup storage.

About this task

This process creates a dynamically provisioned persistent volume that is shared between the primary and standby databases. This volume is used by the Db2 HADR custom resource definition to share the database backup and keystore tar file between the two deployments.

Procedure

1. Deploy the primary database with a backup storage area.
2. Once the primary database has been deployed, determine the name of the backup PVC created for the primary deployment.

   This can be done using:

   ```
   oc get pvc | grep -backup
   ```

   The PVC name should contain the name of the primary Db2uCluster with the **-backup** suffix.
3. Deploy the standby database using the existing backup storage persistent volume claim (PVC) from the primary database. For example, the storage section of the Db2uCluster CR would look as follows:

   ```
   - claimName: c-db2oltp-crd-hadr-primary-backup
     name: backup
     spec:
        resources: {}
     type: existing
   ```

   Ensure you use the name of the PVC determined previously as the *claimName*.

Configuring with HADR scripts

When you configure Db2 high availability disaster recovery (HADR) in different Red Hat OpenShift projects, you create services to expose the HADR endpoints and then configure and start HADR.

Creating services to expose the HADR endpoints
To specify connection information for the remote source or remote target database in Db2 High Availability Disaster Recovery (HADR), you must create and specify service endpoints to expose the Db2 HADR ports.

Unlike traditional (on-premise) Db2 HADR, you cannot use pod IP address to define the HADR remote server because in Red Hat OpenShift whenever a pod is rescheduled the pod/container IP address changes. Instead, you must expose the HADR connection information as an OpenShift service that can then be referenced by the remote HADR copy to set the database configuration parameters **hadr_remote_host** and **hadr_remote_svc**. The OpenShift cluster DNS then resolves the service to the correct active pod, regardless of which worker node it is scheduled on.

You can use the provided script to generate the required service definitions for HADR.

Depending on the HADR topology, the create HADR services script will generate the appropriate service definitions:

Single cluster (single or multiple projects)
A ClusterIP type service for each database in its own OpenShift project

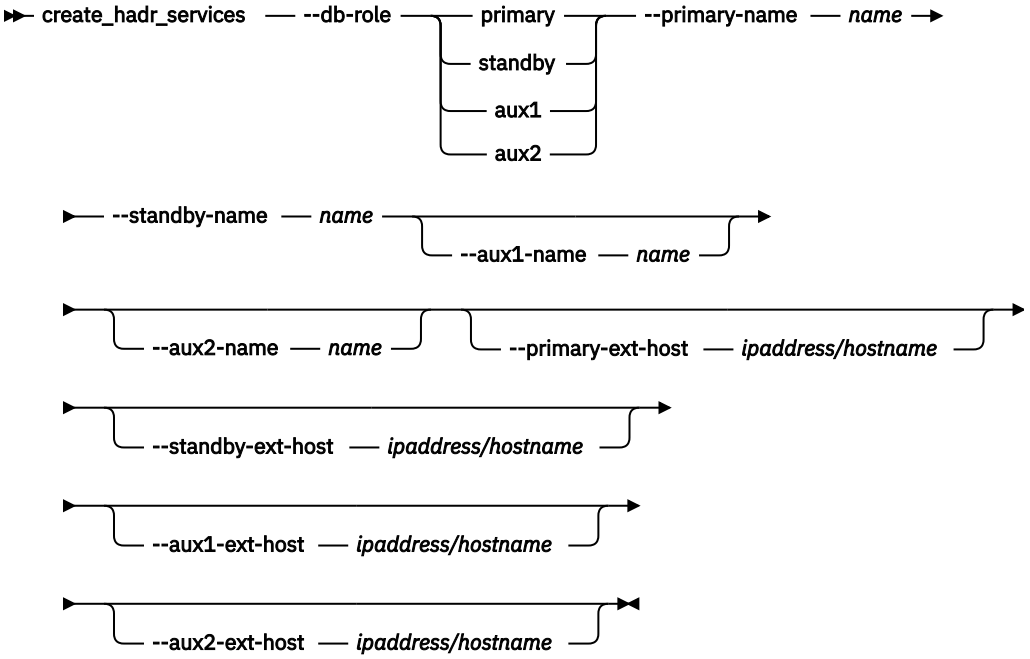
Different clusters

- A NodeIP type service for each database in its own OpenShift project
- ExternalName services corresponding to every database, in each OpenShift project plus cluster

Important: This script needs to be executed on every database in the HADR configuration with the appropriate parameters corresponding to your HADR topology.

Note that this script only generates the YAML definitions. You can use the **oc apply -f** command to create the services directly from the output, or if the output is redirected to a file, from that file.

Syntax



Parameters

--db-role
The HADR role for the current database, either primary, standby, aux1, or aux2.

--primary-name
The name of the primary Db2 cluster.

--standby-name
The name of the standby Db2 cluster.

--aux1-name
The name of the auxiliary1 Db2 cluster.

--aux2-name
The name of the auxiliary2 Db2 cluster.

--primary-ext-host
The external IP address or hostname of the primary database cluster. Required if the topology includes multiple clusters, for example virtual IP, ELB, infra node with external-facing Ingress Controller.

--standby-ext-host
The external IP address or hostname of the standby database cluster. Required if the topology includes multiple clusters.

--aux1-ext-host
The external IP address or hostname of the auxiliary1 database cluster. Required if the topology includes multiple clusters.

--aux2-ext-host
The external IP address or hostname of the auxiliary2 database cluster. Required if the topology includes multiple clusters.

Before you start

Note the names of the Db2uCluster custom resources that correspond to the primary and standby databases. Designate an HADR role for each database:

- primary => primary
- principal standby => standby
- auxiliary 1 => aux1
- auxiliary 2 => aux2

As noted previously, automated failover is only supported between the primary and principal standby databases. As such, these roles are designated in the HADR configuration and cannot be switched without reconfiguring HADR.

```
oc get db2ucluster
NAME STATE AGE
db2oltp-primary Ready 6h26m
db2oltp-standby Ready 6h26m
db2oltp-aux Ready 6h26m
```

Creating HADR services for a single cluster topology
You can create HADR services for a single cluster topology, whether databases are in a single project or multiple OpenShift projects.

About this task

If your configuration includes only the primary database and a single standby database (no auxiliary standbys) in the same OpenShift project, you might want to use the HADR API instead of the HADR scripts.

Procedure

1. Generate the HADR service definitions using the `create_hadr_services` script on the primary database pod.

   Ports are always set up for primary and three standby databases, regardless of the configuration.

   ```
   oc exec -it c-db2oltp-primary-db2u-0 -- create_hadr_services --db-role primary --primary-
   name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux

   apiVersion: v1
   kind: Service
   metadata:
     name: c-db2oltp-primary-hadr-svc
   spec:
     selector:
       app: db2oltp-primary
       type: engine
     ports:
       - name: db2u-hadrp
         port: 60006
         targetPort: 60006
       - name: db2u-hadrs
         port: 60007
         targetPort: 60007
       - name: db2u-hadra1
         port: 60008
         targetPort: 60008
       - name: db2u-hadra2
         port: 60009
         targetPort: 60009
     type: ClusterIP
   ---
   ```

2. Switch to the project that the primary database is in, and use the **`oc apply -f`** command directly on the output to create the k8s services:

```
oc project zen
oc exec -it c-db2oltp-primary-db2u-0 -- create_hadr_services --db-role primary --primary-
name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux | oc apply -f
-
```

3. Verify that the service was created in the same OpenShift project as the Db2 deployment:

```
oc get svc | grep hadr-svc
#Output
c-db2oltp-primary-hadr-svc    ClusterIP   172.30.77.20  <none>   60006/TCP,60007/TCP,60008/
TCP,60009/TCP        26s
```

4. Repeat steps 1-3 for each standby database, ensuring that you use the appropriate value for **--db-role** and that you are in the project that corresponds to the database:

```
oc project zen
oc exec -it c-db2oltp-standby-db2u-0 -- create_hadr_services --db-role standby --primary-
name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux | oc apply -f
-

oc project zen-dr
oc exec -it c-db2oltp-aux-db2u-0 -- create_hadr_services --db-role aux1 --primary-name
db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux | oc apply -f -
```

Creating HADR services for a multiple cluster topology
You can create HADR services for a multiple cluster topology.

About this task

The examples in this procedure are for an HADR setup with a primary and two standby databases. The primary and principal standby are in cluster1, and the auxiliary standby is in cluster2.

Procedure

1. Generate the HADR service definitions using the `create_hadr_services` script on the primary database pod.

   Ports are always set up for primary and three standby databases, regardless of the your configuration.

```
oc exec -it c- db2oltp-primary -db2u-0 -- create_hadr_services --db-role primary
--primary-name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux --
primary-ext-host api.cluster1.ibm.com --standby-ext-host api.cluster1.ibm.com --aux1-ext-
host api.cluster2.ibm.com

apiVersion: v1
kind: Service
metadata:
  name: c-db2oltp-primary-hadr-svc
spec:
  selector:
    app: db2oltp-primary
    type: engine
  ports:
    - name: db2u-hadrp
      port: 60006
      targetPort: 60006
    - name: db2u-hadrs
      port: 60007
      targetPort: 60007
    - name: db2u-hadra1
      port: 60008
      targetPort: 60008
    - name: db2u-hadra2
      port: 60009
      targetPort: 60009
  type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: c-db2oltp-primary-hadr-svc-ext
spec:
  type: ExternalName
```

```
  externalName: api.cluster1.ibm.com
---
apiVersion: v1
kind: Service
metadata:
  name: c-db2oltp-standby-hadr-svc-ext
spec:
  type: ExternalName
  externalName: api.cluster1.ibm.com
---
apiVersion: v1
kind: Service
metadata:
  name: c-db2oltp-aux-hadr-svc-ext
spec:
  type: ExternalName
  externalName: api.cluster2.ibm.com
---
```

2. Switch to the project that the primary database is in, and use the **oc apply -f** command directly on the output to create the k8s services:

```
oc project zen
oc exec -it c- db2oltp-primary-db2u-0 -- create_hadr_services --db-role primary --
primary-name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux --
primary-ext-host api.cluster1.ibm.com --standby-ext-host api.cluster1.ibm.com --aux1-ext-
host api.cluster2.ibm.com | oc apply -f -
```

3. Verify that the services were created:
   - One NodePort service that matches the Db2 deployment
   - Multiple ExternalName services – one for each database in the HADR configuration

```
oc get svc | grep hadr-svc
# Output:
c-db2oltp-aux-hadr-svc-ext       ExternalName   <none>     api.cluster2.ibm.com   <none>   9s
c-db2oltp-primary-hadr-svc        NodePort       172.30.77.20     <none>   60006:32457/
TCP,60007:31243/TCP,60008:30374/TCP,60009:30977/TCP   2m15s
c-db2oltp-primary-hadr-svc-ext    ExternalName   <none>  api.cluster1.ibm.com   <none>    9s
c-db2oltp-standby-hadr-svc-ext    ExternalName   <none>     api.cluster1.ibm.com   <none>
```

4. Repeat steps 2 and 3 for each standby database, ensuring that you use the appropriate value for **--db-role** and that you are in the project that corresponds to the database:

```
# Create services for principal standby database in cluster1
oc project zen
oc exec -it c- db2oltp-standby-db2u-0 -- create_hadr_services --db-role standby --
primary-name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux --
primary-ext-host api.cluster1.ibm.com --standby-ext-host api.cluster1.ibm.com --aux1-ext-
host api.cluster2.ibm.com | oc apply -f -

# Check services on cluster1
oc get svc | grep hadr-svc
# Output:
c-db2oltp-aux-hadr-svc-ext       ExternalName   <none>           api.cluster2.ibm.com
<none>                                                                    9s
c-db2oltp-primary-hadr-svc        NodePort       172.30.77.20
<none>                            60006:32457/TCP,60007:31243/TCP,60008:30374/
TCP,60009:30977/TCP                   2m15s
c-db2oltp-primary-hadr-svc-ext    ExternalName   <none>           api.cluster1.ibm.com
<none>                                                                    9s
c-db2oltp-standby-hadr-svc        NodePort       172.30.247.77
<none>                            60006:32649/TCP,60007:31384/TCP,60008:30473/
TCP,60009:30652/TCP                   1m10s
c-db2oltp-standby-hadr-svc-ext    ExternalName   <none>           api.cluster1.ibm.com
<none>

# Create services for auxiliary standby database in cluster2
oc project zen
oc exec -it c- db2oltp-aux-db2u-0 -- create_hadr_services --db-role aux1 --primary-
name db2oltp-primary --standby-name db2oltp-standby --aux1-name db2oltp-aux --primary-
ext-host api.cluster1.ibm.com --standby-ext-host api.cluster1.ibm.com --aux1-ext-host
api.cluster2.ibm.com | oc apply -f –

# Check services on cluster2
oc get svc | grep hadr-svc
# Output:
```

```
c-db2oltp-aux-hadr-svc        NodePort       172.30.241.25
<none>                            60006:34578/TCP,60007:31546/TCP,60008:30698/
TCP,60009:30448/TCP                  10s
c-db2oltp-aux-hadr-svc-ext       ExternalName   <none>           api.cluster2.ibm.com
<none>                                                                  9s
c-db2oltp-primary-hadr-svc-ext    ExternalName   <none>            api.cluster1.ibm.com
<none>                                                                  9s
c-db2oltp-standby-hadr-svc-ext    ExternalName   <none>            api.cluster1.ibm.com
<none>
```

Determining the NodePorts for a multiple cluster configuration
If the HADR databases are in different OpenShift clusters, you must have NodePort-type services for deployments to communicate with each other.

About this task

For details on creating these services, see “Creating services to expose the HADR endpoints” on page 84.

The specific NodePorts that are assigned to each service/database are required for the following HADR configuration steps:

- Configure the HADR NodePorts with an Ingress Controller.
- Set up the HADR configuration.

To determine the HADR NodePort that is associated with a Db2 deployment:

Procedure

1. Describe the HADR service that corresponds to the database deployment.
2. Find the NodePort that corresponds to the port for the database HADR role:
   - Primary - db2u-hadrp
   - Principal Standby – db2u-hadrs
   - Auxiliary Standby 1 – db2u-hadra1
   - Auxiliary Standby 2 – db2u-hadra2

   In the following example, the `c-db2oltp-primary-hadr-svc` service corresponds to the primary database, and the NodePort for the primary deployment is 32457.

```
oc describe svc c-db2oltp-primary-hadr-svc
#Output:
Name:                     c-db2oltp-primary-hadr-svc
Namespace:                zen
Labels:                   <none>
Annotations:              <none>
Selector:                 app=db2oltp-primary,type=engine
Type:                     NodePort
IP:                       172.30.77.20
Port:                     db2u-hadrp  60006/TCP
TargetPort:               60006/TCP
NodePort:                 db2u-hadrp  32457/TCP
Endpoints:                10.254.27.150:60006
Port:                     db2u-hadrs  60007/TCP
TargetPort:               60007/TCP
NodePort:                 db2u-hadrs  31243/TCP
Endpoints:                10.254.27.150:60007
Port:                     db2u-hadra1  60008/TCP
TargetPort:               60008/TCP
NodePort:                 db2u-hadra1  30374/TCP
Endpoints:                10.254.27.150:60008
Port:                     db2u-hadra2  60009/TCP
TargetPort:               60009/TCP
NodePort:                 db2u-hadra2  30977/TCP
Endpoints:                10.254.27.150:60009
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

In the following example, the `c-db2oltp-standby-hadr-svc` service corresponds to the principal standby database, and the NodePort for the standby deployment is 31384.

```
oc describe svc c-db2oltp-standby-hadr-svc
# Output:
Name:                     c-db2oltp-standby-hadr-svc
Namespace:                zen
Labels:                   <none>
Annotations:              <none>
Selector:                 app=db2oltp-standby,type=engine
Type:                     NodePort
IP:                       172.30.247.77
Port:                     db2u-hadrp  60006/TCP
TargetPort:               60006/TCP
NodePort:                 db2u-hadrp  32649/TCP
Endpoints:                10.254.21.80:60006
Port:                     db2u-hadrs  60007/TCP
TargetPort:               60007/TCP
NodePort:                 db2u-hadrs  31384/TCP
Endpoints:                10.254.21.80:60007
Port:                     db2u-hadra1  60008/TCP
TargetPort:               60008/TCP
NodePort:                 db2u-hadra1  30473/TCP
Endpoints:                10.254.21.80:60008
Port:                     db2u-hadra2  60009/TCP
TargetPort:               60009/TCP
NodePort:                 db2u-hadra2  30652/TCP
Endpoints:                10.254.21.80:60009
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Setting up Ingress HA proxy for different OpenShift clusters
If the databases in your HADR configuration are on different OpenShift clusters, and you use an external infrastructure node to route external Db2 traffic into the cluster, you need to configure an external-facing Ingress Controller to route the traffic to the OpenShift nodes.

Procedure

1. Note down the NodePorts for the deployments on the OpenShift cluster by using "Determining the NodePorts for a multiple cluster configuration" on page 89.
2. Create the Ingress Controller by following the steps in "Configuring the Db2 NodePort with an Ingress Controller" on page 119.

Creating a network policy for built-in etcd for Db2
To use the built-in etcd store from one of the Db2 deployments in your HADR configuration, you must create a network policy to allow other database deployments to communicate with the etcd pod.

About this task

Creating a network policy will allow incoming connections to the etcd pod. For more details on the built-in etcd store, see Setting up an etcd store for HADR.

Important: The built-in etcd store is not to be used in production environments.

Procedure

1. Create variables referencing the *DB2UCLUSTER* name and *NAMESPACE* being used. Use the corresponding *DB2UCLUSTER* with the database deployment containing the built-in etcd store you are using.

```
export DB2UCLUSTER=<db2ucluster_name>
export NAMESPACE=<namespace_of_db2ucluster>
```

2. Create a network policy for the etcd pod by modifying, then running the following script.

```
cat <<EOF | oc apply -f -
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
metadata:
  name: c-${DB2UCLUSTER}-etcd-ext
  namespace: ${NAMESPACE}
spec:
  ingress:
  - ports:
    - port: 2379
      protocol: TCP
  podSelector:
    matchLabels:
      formation_id: ${DB2UCLUSTER}
      component: etcd
  policyTypes:
  - Ingress
EOF
```

Results

You can now use the network policy to access etcd for automated failover.

What to do next

See "Setting up the HADR configuration" on page 91.

Setting up the HADR configuration

After you configure the service endpoints, you can quickly set up the high availability disaster recovery (HADR) configuration by running a script that performs the setup steps.

Before you begin

- Determine the HADR ports corresponding to each database.
  - In a topology with a single OpenShift cluster, the ports for each database are:
    - Primary: 60006
    - Standby: 60007
    - Aux1: 60008
    - Aux2: 60009
  - In a multiple cluster topology, use "Determining the NodePorts for a multiple cluster configuration" on page 89.
- If databases are in different OpenShift projects on the same cluster, also determine the IP addresses for each HADR service.
- Ensure you have the etcd endpoint (IP address/hostname and port) available for HADR setup. See "Setting up an etcd store for HADR " on page 78.

  For a development-only environment with all databases on a single OpenShift cluster, you can use the built-in etcd store of the primary database.

About this task

The script:

- Generates and sets the Db2 HADR database configuration parameters
- On the primary database, generates an offline backup of the database and the encryption keystore.
- On the standby databases, restores the offline backup image and encryption keystore that were copied from the primary database.

For full command syntax and parameter descriptions, see "setup_config_hadr" on page 92.

To set up HADR, follow these steps:

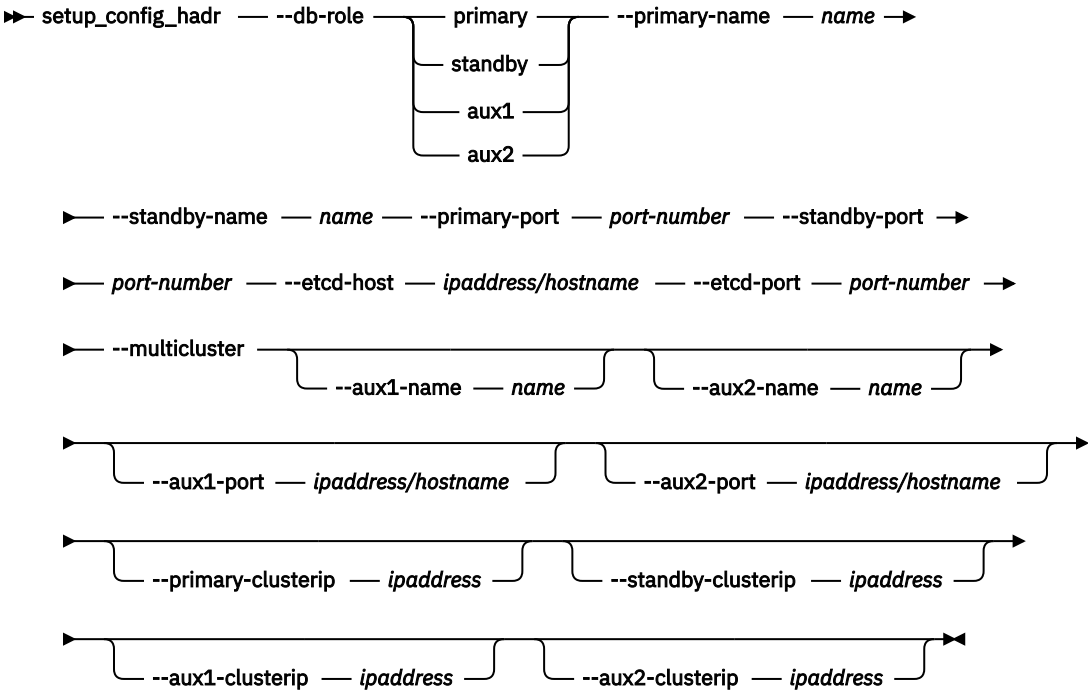
Procedure

1. Set up HADR by using the `setup_config_hadr` script on the primary database pod.
2. Copy the database backup image and keystore file in the backup storage area (`/mnt/backup/`) from the primary database to the standby database or databases by using rsync.

   Note: If the backup volume is shared between the primary and standby databases, you can skip this step.
3. Set up HADR using the `setup_config_hadr` script on each standby database pod.

setup_config_hadr
Use the `setup_config_hadr` script to set up a high availability disaster recovery (HADR) configuration.

Syntax



Parameters

--db-role
The HADR role for the current database, either primary, standby, aux1, or aux2.

--primary-name
The name of the primary Db2 cluster.

--standby-name
The name of the standby Db2 cluster.

--primary-port
The HADR primary port of the clusterIP or NodePort service that corresponds to the primary database.

--standby-port
The HADR standby port of the clusterIP or NodePort service that corresponds to the standby database.

--etcd-host
The etcd cluster hostname or IP address.

--etcd-port
The etcd port.

--multicluster
A flag that specifies that the databases are on different clusters. If not specified, the default is a single cluster topology.

--aux1-name
The name of the auxiliary1 Db2 cluster.

--aux2-name
The name of the auxiliary2 Db2 cluster.

--aux1-port
The HADR aux1 port of the clusterIP or NodePort service that corresponds to the aux1 database.

--aux2-port
The HADR aux2 port of the clusterIP or NodePort service that corresponds to the aux2 database.

--primary-clusterip
The ClusterIP of the primary database HADR service (c-*primary-Db2-cluster*-hadr-svc). In an IBM Cloud environment, this is the fully qualified domain name of the primary db2u-0 pod (c-*primary-Db2-cluster*-db2u-0.c-*primary-Db2-cluster*-db2u-internal.primary-project.svc.cluster.local).

--standby-clusterip
The ClusterIP of the standby database HADR service (c-*standby-Db2-cluster*-hadr-svc). In an IBM Cloud environment, this is the fully qualified domain name of the standby db2u-0 pod (c-*standby-Db2-cluster*-db2u-0.c-*standby-Db2-cluster*-db2u-internal.standby-project.svc.cluster.local).

--aux1-clusterip
The ClusterIP of the aux1 database HADR service (c-*aux1-Db2-cluster*-hadr-svc). In an IBM Cloud environment, this is the fully qualified domain name of the aux1 db2u-0 pod (c-*aux1-Db2-cluster*-db2u-0.c-*aux1-Db2-cluster*-db2u-internal.aux1-project.svc.cluster.local).

--aux2-clusterip
The ClusterIP of the aux2 database HADR service (c-*aux2-Db2-cluster*-hadr-svc). In an IBM Cloud environment, this is the fully qualified domain name of the aux2 db2u-0 pod (c-*aux2-Db2-cluster*-db2u-0.c-*aux2-Db2-cluster*-db2u-internal.aux2-project.svc.cluster.local).

Example: Setting up HADR in a single OpenShift project
This example has one primary, one principal standby, and one auxiliary standby database, all located in the same OpenShift project.

Because the services were created by using "Setting up the HADR configuration" on page 91, the ports for each database are:

- Primary: 60006
- Standby: 60007
- Aux1: 60008

1. Run the following command to invoke the `setup_config_hadr` script on the primary database Db2 pod:

```
oc exec -it c-db2oltp-primary-db2u-0 -- setup_config_hadr --db-role primary --primary-name
db2oltp-primary --standby-name db2oltp-standby --primary-port 60006 --standby-port 60007 --
aux1-name db2oltp-aux --aux1-port 60008 --etcd-host my-etcd-client.my-etcd --etcd-port 2379
```

The output of the command contains the HADR configuration settings:

```
################################################################################################
###        Creating HADR configuration file for HADR setup.                              ###
################################################################################################

    Database role -- primary
    primary database formation_id -- db2oltp-primary
```

```
HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-primary-db2u-0|c-db2oltp-primary-hadr-svc
HADR_LOCAL_SVC 60006|60006
HADR_REMOTE_HOST c-db2oltp-standby-hadr-svc
HADR_REMOTE_SVC 60007
HADR_TARGET_LIST c-db2oltp-standby-hadr-svc:60007|c-db2oltp-aux-hadr-svc:60008
etcd: my-etcd-client.my-etcd:2379
```

If the setup is successful, the following message displays at the end of the script output:

```
################################################################################################
###        The HADR setup command completed the primary node setup.          ###
################################################################################################
```

2. Copy the database backup image and keystore file in the backup storage area (`/mnt/backup/`) from the primary database to the standby databases by using rsync.

```
# Copy from primary database to a directory on the host called /tmp/hadr
oc rsync c-db2oltp-primary-db2u-0:/mnt/backup/ /tmp/hadr

# Copy from the /tmp/hadr directory on the host to the standby databases
oc rsync /tmp/hadr/ c-db2oltp-standby-db2u-0:/mnt/backup/
oc rsync /tmp/hadr/ c-db2oltp-aux-db2u-0:/mnt/backup/
```

3. Run the following command to invoke the `setup_config_hadr` script on the principal standby database Db2 pod:

```
oc exec -it c-db2oltp-standby-db2u-0 -- setup_config_hadr --db-role standby --primary-name
db2oltp-primary --standby-name db2oltp-standby --primary-port 60006 --standby-port 60007 --
aux1-name db2oltp-aux --aux1-port 60008 --etcd-host my-etcd-client.my-etcd --etcd-port 2379
```

The output of the command:

```
################################################################################################
###        Creating HADR configuration file for HADR setup.                  ###
################################################################################################

    Database role -- standby
    primary database formation_id -- db2oltp-standby

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-standby-db2u-0|c-db2oltp-standby-hadr-svc
HADR_LOCAL_SVC 60007|60007
HADR_REMOTE_HOST c-db2oltp-primary-hadr-svc
HADR_REMOTE_SVC 60006
HADR_TARGET_LIST c-db2oltp-primary-hadr-svc:60006|c-db2oltp-aux-hadr-svc:60008
etcd: my-etcd-client.my-etcd:2379
```

4. Run the following command to invoke the `setup_config_hadr` script on the auxiliary standby database Db2 pod:

```
oc exec -it c-db2oltp-aux-db2u-0 -- setup_config_hadr --db-role aux1 --primary-name db2oltp-
primary --standby-name db2oltp-standby --primary-port 60006 --standby-port 60007 --aux1-name
db2oltp-aux --aux1-port 60008 --etcd-host my-etcd-client.my-etcd --etcd-port 2379
```

The output of the command:

```
################################################################################################
###        Creating HADR configuration file for HADR setup.                  ###
################################################################################################

    Database role -- aux1
    aux1 database formation_id -- db2oltp-aux

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
```

```
HADR_SYNCMODE SUPERASYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-aux-db2u-0|c-db2oltp-aux-hadr-svc
HADR_LOCAL_SVC 60008|60008
HADR_REMOTE_HOST c-db2oltp-primary-hadr-svc
HADR_REMOTE_SVC 60006
HADR_TARGET_LIST c-db2oltp-standby-hadr-svc:60007|c-db2oltp-primary-hadr-svc:60006
etcd: my-etcd-client.my-etcd:2379
```

Example: Setting up HADR in a single OpenShift cluster across multiple OpenShift projects
This example has one primary and one principal standby in different OpenShift projects within the same cluster.

Because the services were created by using "Setting up the HADR configuration" on page 91, the ports for each database are:

- Primary: 60006
- Standby: 60007

The IP addresses for each database are:

- Primary: 172.30.77.20
- Standby: 172.30.247.77

In the example below, the IP address of the primary database is 172.30.77.20:

```
oc get svc | grep hadr-svc
# Output:
c-db2oltp-primary-hadr-svc        ClusterIP   172.30.77.20     <none>        60006/TCP,60007/
TCP,60008/TCP,60009/TCP                                          26s

oc describe svc c-db2oltp-primary-hadr-svc
# Output:
Name:                     c-db2oltp-primary-hadr-svc
Namespace:                zen
Labels:                   <none>
Annotations:              <none>
Selector:                 app=db2oltp-primary,type=engine
Type:                     ClusterIP
IP:                       172.30.77.20
Port:                     db2u-hadrp  60006/TCP
TargetPort:               60006/TCP
Endpoints:                10.254.27.150:60006
Port:                     db2u-hadrs  60007/TCP
TargetPort:               60007/TCP
Endpoints:                10.254.27.150:60007
Port:                     db2u-hadra1  60008/TCP
TargetPort:               60008/TCP
Endpoints:                10.254.27.150:60008
Port:                     db2u-hadra2  60009/TCP
TargetPort:               60009/TCP
Endpoints:                10.254.27.150:60009
Session Affinity:         None
Events:                   <none>
```

1. Run the following command to invoke the `setup_config_hadr` script on the primary database Db2 pod:

   ```
   oc exec -it c-db2oltp-primary-db2u-0 -- setup_config_hadr --db-role primary --primary-
   name db2oltp-primary --standby-name db2oltp-standby --primary-port 60006 --standby-port
   60007 --primary-clusterip 172.30.77.20 --standby-clusterip 172.30.247.77 --etcd-host my-etcd-
   client.my-etcd --etcd-port 2379
   ```

   The output of the command contains the HADR configuration settings:

   ```
   ################################################################################################
   ###        Creating HADR configuration file for HADR setup.                                  ###
   ################################################################################################

       Database role -- primary
       primary database formation_id -- db2oltp-primary
   ```

```
HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-primary-db2u-0|172.30.77.20
HADR_LOCAL_SVC 60006|60006
HADR_REMOTE_HOST 172.30.247.77
HADR_REMOTE_SVC 60007
HADR_TARGET_LIST 172.30.247.77:60007
etcd: my-etcd-client.my-etcd:2379
```

If the setup is successful, the following message displays at the end of the script output:

```
################################################################################################
###        The HADR setup command completed the primary node setup.          ###
################################################################################################
```

2. Copy the database backup image and keystore file in the backup storage area (`/mnt/backup/`) from the primary database to the standby databases by using rsync.

```
# Copy from primary database to a directory on the host called /tmp/hadr
oc rsync c-db2oltp-primary-db2u-0:/mnt/backup/ /tmp/hadr

# Copy from the /tmp/hadr directory on the host to the standby database
oc rsync /tmp/hadr/ c-db2oltp-standby-db2u-0:/mnt/backup/
```

3. Run the following command to invoke the `setup_config_hadr` script on the standby database Db2 pod:

```
oc exec -it c-db2oltp-standby-db2u-0 -- setup_config_hadr --db-role standby --primary-
name db2oltp-primary --standby-name db2oltp-standby --primary-port 60006 --standby-port
60007 --primary-clusterip 172.30.77.20 --standby-clusterip 172.30.247.77 --etcd-host my-etcd-
client.my-etcd --etcd-port 2379
```

The output of the command:

```
################################################################################################
###        Creating HADR configuration file for HADR setup.                  ###
################################################################################################

    Database role -- standby
    primary database formation_id -- db2oltp-standby

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-primary-db2u-0|172.30.247.77
HADR_LOCAL_SVC 60007|60007
HADR_REMOTE_HOST 172.30.77.20
HADR_REMOTE_SVC 60006
HADR_TARGET_LIST 172.30.77.20:60006
etcd: my-etcd-client.my-etcd:2379
```

Example: Setting up HADR in multiple OpenShift clusters
This example has one primary and one principal standby in the same OpenShift project within the same cluster, and one auxiliary standby in a different OpenShift cluster (disaster site).

Using “Determining the NodePorts for a multiple cluster configuration” on page 89 configuration, the ports for each database in the example are:

- Primary: 32457
- Standby: 31384
- Aux1: 32649

To set up HADR for databases that are in different OpenShift clusters:

1. Run the following command to invoke the `setup_config_hadr` script on the primary database Db2 pod:

```
oc exec -it c-db2oltp-primary-db2u-0 -- setup_config_hadr --db-role primary --primary-name
db2oltp-primary --standby-name db2oltp-standby --primary-port 32457 --standby-port 31384 --
aux1-name db2oltp-aux --aux1-port 32649 --etcd-host my-etcd-client.my-etcd --etcd-port 2379
--multicluster
```

   The output of the command contains the HADR configuration settings:

```
################################################################################################
###        Creating HADR configuration file for HADR setup.                  ###
################################################################################################

    Database role -- primary
    primary database formation_id -- db2oltp-primary

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-primary-db2u-0|c-db2oltp-standby-hadr-svc-ext
HADR_LOCAL_SVC 60006|32457
HADR_REMOTE_HOST c-db2oltp-standby-hadr-svc-ext
HADR_REMOTE_SVC 31384
HADR_TARGET_LIST c-db2oltp-standby-hadr-svc-ext:31384|c-db2oltp-aux-hadr-svc-ext:32649
etcd: my-etcd-client.my-etcd:2379
```

   If the setup is successful, the following message displays at the end of the script output:

```
################################################################################################
###        The HADR setup command completed the primary node setup.          ###
################################################################################################
```

2. Copy the database backup image and keystore file in the backup storage area (`/mnt/backup/`) from the primary database to the standby databases by using rsync. An extra step is required here to copy from Cluster1 (where the primary database is located) to Cluster2 (where the auxiliary standby database is located).

```
# Copy from primary database to a directory on the host called /tmp/hadr
oc rsync c-db2oltp-primary-db2u-0:/mnt/backup/ /tmp/hadr

# Copy from the /tmp/hadr directory on the host to the principal standby database
oc rsync /tmp/hadr/ c-db2oltp-standby-db2u-0:/mnt/backup/

# After copying /tmp/hadr/ from Cluster1 to Cluster2
oc rsync /tmp/hadr/ c-db2oltp-aux-db2u-0:/mnt/backup/
```

3. Run the following command to invoke the `setup_config_hadr` script on the principal standby database Db2 pod:

```
oc exec -it c-db2oltp-standby-db2u-0 -- setup_config_hadr --db-role standby --primary-name
db2oltp-primary --standby-name db2oltp-standby --primary-port 32457 --standby-port 31384 --
aux1-name db2oltp-aux --aux1-port 32649 --etcd-host my-etcd-client.my-etcd --etcd-port 2379
--multicluster
```

   The output of the command:

```
################################################################################################
###        Creating HADR configuration file for HADR setup.                  ###
################################################################################################

    Database role -- standby
    primary database formation_id -- db2oltp-standby

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-standby-db2u-0|c-db2oltp-primary-hadr-svc-ext
HADR_LOCAL_SVC 60007|31384
```

```
HADR_REMOTE_HOST c-db2oltp-primary-hadr-svc-ext
HADR_REMOTE_SVC 32457
HADR_TARGET_LIST c-db2oltp-primary -hadr-svc-ext:32457|c-db2oltp-aux-hadr-svc-ext:32649
etcd: my-etcd-client.my-etcd:2379
```

4. Run the following command to invoke the `setup_config_hadr` script on the auxiliary standby database Db2 pod:

```
oc exec -it c-db2oltp-aux-db2u-0 -- setup_config_hadr --db-role aux1 --primary-name db2oltp-
primary --standby-name db2oltp-standby --primary-port 32457 --standby-port 31384 --aux1-
name db2oltp-aux --aux1-port 32649 --etcd-host my-etcd-client.my-etcd --etcd-port 2379 --
multicluster
```

The output of the command:

```
##############################################################################################
###        Creating HADR configuration file for HADR setup.                  ###
##############################################################################################

    Database role -- aux1
    primary database formation_id -- db2oltp-aux

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-aux-db2u-0|c-db2oltp-aux-hadr-svc-ext
HADR_LOCAL_SVC 60008|32649
HADR_REMOTE_HOST c-db2oltp-primary-hadr-svc-ext
HADR_REMOTE_SVC 32457
HADR_TARGET_LIST c-db2oltp-standby-hadr-svc-ext:31384|c-db2oltp-primary-hadr-svc-ext:32457
etcd: my-etcd-client.my-etcd:2379
```

Example: Setting up HADR in a single IBM Cloud cluster
This example has one primary and one principal standby in different OpenShift projects within the same cluster on IBM Cloud. The same logic applies for deployments in a single OpenShift project, except the project names would be the same for primary and standby.

Note the project for each deployment. In this example:

- Primary: project *primary-project*
- Standby: project *standby-project*

The ports for each database:

- Primary: 60006
- Standby: 60007

Instead of using IP addresses, use the fully qualified domain name for each db2u-0 pod instead:

- Primary: `c-db2oltp-primary-db2u-0.c-db2oltp-primary-db2u-internal.primary-project.svc.cluster.local`
- Standby: `c-db2oltp-standby-db2u-0.c-db2oltp-standby-db2u-internal.standby-project.svc.cluster.local`

Procedure

1. Set up HADR using the `setup_config_hadr` script on the primary database db2u pod:

```
oc exec -it c-db2oltp-primary-db2u-0 -- setup_config_hadr --db-role primary --primary-
name db2oltp-primary --standby-name db2oltp-standby --primary-port 60006 --standby-port
60007 --primary-clusterip c-db2oltp-primary-db2u-0.c-db2oltp-primary-db2u-internal.primary-
project.svc.cluster.local --standby-clusterip c-db2oltp-standby-db2u-0.c-db2oltp-standby-
db2u-internal.standby-project.svc.cluster.local --etcd-host my-etcd-client.my-etcd --etcd-
port 2379
```

The HADR configuration settings are in the script output:

```
##############################################################################################
###        Creating HADR configuration file for HADR setup.                                ###
##############################################################################################

    Database role -- primary
    primary database formation_id -- db2oltp-primary

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-primary-db2u-0|c-db2oltp-primary-db2u-0.c-db2oltp-primary-db2u-
internal.primary-project.svc.cluster.local
HADR_LOCAL_SVC 60006|60006
HADR_REMOTE_HOST c-db2oltp-standby-db2u-0.c-db2oltp-standby-db2u-internal.standby-
project.svc.cluster.local
HADR_REMOTE_SVC 60007
HADR_TARGET_LIST c-db2oltp-standby-db2u-0.c-db2oltp-standby-db2u-internal.standby-
project.svc.cluster.local:60007
etcd: my-etcd-client.my-etcd:2379
```

If the setup is successful, you should see the following message at the end of the script output:

```
##############################################################################################
###        The HADR setup command completed the primary node setup.                        ###
##############################################################################################
```

2. Copy the database backup image and keystore file in the backup storage area (`/mnt/backup/`) from the primary database to the standby database by using rsync:

```
## Copy from primary database to a directory on the host called /tmp/hadr
oc rsync c-db2oltp-primary-db2u-0:/mnt/backup/ /tmp/hadr

## Copy from the /tmp/hadr directory on the host to the standby database
oc rsync /tmp/hadr/ c-db2oltp-standby-db2u-0:/mnt/backup/
```

3. Set up HADR by using the `setup_config_hadr` script on the standby database db2u pod:

```
oc exec -it c-db2oltp-standby-db2u-0 -- setup_config_hadr --db-role standby --primary-
name db2oltp-primary --standby-name db2oltp-standby --primary-port 60006 --standby-port
60007 --primary-clusterip c-db2oltp-primary-db2u-0.c-db2oltp-primary-db2u-internal.primary-
project.svc.cluster.local --standby-clusterip c-db2oltp-standby-db2u-0.c-db2oltp-standby-
db2u-internal.standby-project.svc.cluster.local --etcd-host my-etcd-client.my-etcd --etcd-
port 2379
```

Script output:

```
##############################################################################################
###        Creating HADR configuration file for HADR setup.                                ###
##############################################################################################

    Database role -- standby
    primary database formation_id -- db2oltp-standby

HADR_REMOTE_INST db2inst1
HADR_TIMEOUT 120
HADR_SYNCMODE NEARSYNC
HADR_PEER_WINDOW 120
LOGINDEXBUILD ON
HADR_LOCAL_HOST c-db2oltp-standby-db2u-0|c-db2oltp-standby-db2u-0.c-db2oltp-standby-db2u-
internal.standby-project.svc.cluster.local
HADR_LOCAL_SVC 60007|60007
HADR_REMOTE_HOST c-db2oltp-primary-db2u-0.c-db2oltp-primary-db2u-internal.primary-
project.svc.cluster.local
HADR_REMOTE_SVC 60006
HADR_TARGET_LIST c-db2oltp-primary-db2u-0.c-db2oltp-primary-db2u-internal.primary-
project.svc.cluster.local:60006
etcd: my-etcd-client.my-etcd:2379
```

Modifying HADR configuration parameters
The `setup_config_hadr` script that is used when setting up an HADR configuration sets the HADR configuration parameters.

About this task

Some of the parameters are static, such as **`hadr_local_host`**, **`hadr_remote_host`**, and **`hadr_target_list`**. These parameters should not be modified after the script is run. Other parameters are set with default values but can be adjusted by using the steps below.

The HADR configuration parameters that can be modified are:

hadr_syncmode

- Default for primary and principal standby database: NEARSYNC
- Default for auxiliary standby databases: ASYNC

hadr_timeout

- Default: 120 seconds

hadr_peer_window

- Default: 120 seconds

See the Db2 HADR configuration parameter documentation for more information.

Note: You should only change HADR configuration parameters before "Starting an HADR configuration" on page 101, or when HADR is stopped.

Procedure

1. Exec into the Db2 pod and switch to the database instance owner:

   ```
   oc exec -it c-db2oltp-primary-db2u-0 -- bash
   su - db2inst1
   ```

2. Update the desired database configuration parameter by using the **`db2 update`** command.

   The structure of the command is:

   ```
   db2 update db cfg for dbname using param value
   ```

   For example:

   ```
   db2 update db cfg for BLUDB using HADR_TIMEOUT 180
   ```

Enabling reads on standby
You can enable the reads on standby feature on your HADR standby database by using the DB2_HADR_ROS registry variable.

About this task

For full documentation, see Enabling reads on standby.

Important:

- The reads on standby feature should be enabled prior to starting HADR.
- You cannot use automatic client reroute (ACR) if you enable reads on standby.

Procedure

1. Exec into the Db2 pod and switch to the database instance owner:

```
oc exec -it c-db2oltp-primary-db2u-0 -- bash
su - db2inst1
```

2. Use the **db2set** command to enable reads on standby:

```
db2set DB2_HADR_ROS=ON
```

3. Restart the Db2 database.

```
db2stop
ipclean -a
db2start
```

Starting an HADR configuration
To start a high availability disaster recovery (HADR) configuration on Db2, use the `manage_hadr` tool.

Before you begin

- Create services to expose the HADR endpoints.
- Set up the HADR configuration.

Important: When you start an HADR configuration, the backup images that are in the ${BACKUPDIR} path are deleted. To preserve existing backup images, move the images to a subdirectory under ${BACKUPDIR}. For example, ${BACKUPDIR}/backup_001.

Procedure

1. Start HADR on the standby database pod:

   Note: Use **-start_as standby** for both the principal and auxiliary standby databases.

```
oc exec -it db2oltp-1573144443-db2u-0 -- manage_hadr -start_as standby
```

2. Start HADR on the primary database pod:

```
oc exec -it db2oltp-1573141715-db2u-0 -- manage_hadr -start_as primary
```

What to do next

Connect clients to the HADR configuration.

Reinitializing an HADR configuration to resolve errors in Db2
You can reinitialize a Db2 High Availability Disaster Recovery (HADR) configuration to resolve an error condition that prevents the primary and standby databases from connecting and achieving a peer state.

About this task

For various reasons, an HADR configuration can end up in an error state. In these situations, usually one copy of the database (primary or standby) is working correctly while the other copy is corrupted.

For example, after a worker node reboot, the old primary can sometimes fail to re-integrate if the peer window expires and then a subsequent takeover by force is issued by the HADR automation (governor). In such a scenario you will find a log entries in the governor log (`/var/log/governor/governor.log`) that are similar to the following example on the new primary (old standby):

```
2020-04-01 18:35:38,832 INFO 8991-47423382027648: child(13084) executing db2 takeover hadr on
db BLUDB by force peer window only
2020-04-01 18:35:39,084 INFO 8991-47423382027648: SQL1770N  Takeover HADR cannot complete.
Reason code = "9".

2020-04-01 18:35:39,085 INFO 8991-47423382027648: we have the mandate to force takeover
(window=300)
2020-04-01 18:35:39,086 INFO 8991-47423382027648: Result of DNS resolution of remote endpoint:
10.130.0.39
```

```
2020-04-01 18:35:40,096 INFO 8991-47423382027648: child(13151) executing db2 takeover hadr on
db BLUDB by force

....

2020-04-01 18:35:54,686 INFO 8991-47423382027648: using cached role(PRIMARY) as of
0.369333982468 seconds ago (threshold 1)
2020-04-01 18:35:54,687 INFO 8991-47423382027648:
                            db2 role is PRIMARY,
                            db2 connect status is DISCONNECTED,
                            db2 state is DISCONNECTED
```

On the old primary (currently disconnected standby), you will see governor logs inside the Db2 database pod that are similar to the following:

```
2020-04-01 18:36:01,690 INFO 2668-47200027639168: db2 state is LOCAL_CATCHUP
2020-04-01 18:36:01,690 INFO 2668-47200027639168: startup: waiting on db2 to become peer with
primary (waited 20 secs)
2020-04-01 18:36:11,701 INFO 2668-47200027639168: Calling db2pd
2020-04-01 18:36:12,260 INFO 2668-47200027639168: db2pd returned
2020-04-01 18:36:12,262 INFO 2668-47200027639168:
Database BLUDB not activated on database member 0 or this database name cannot be found in the
local database directory.

Option -hadr requires -db <database> or -alldbs option and active database.

2020-04-01 18:36:12,262 INFO 2668-47200027639168: db2 state is None
2020-04-01 18:36:12,262 INFO 2668-47200027639168: startup: waiting on db2 to become peer with
primary (waited 30 secs)
```

The old primary never integrates as the new standby after the rebooted host comes back online. In this situation, the only option is to reinitialize the HADR system by using the following procedure.

Procedure

1. Stop HADR on the standby pod:

   ```
   oc exec -it <standby_pod> -- manage_hadr -stop
   ```

2. Stop HADR on the primary pod:

   ```
   oc exec -it <primary_pod> -- manage_hadr -stop
   ```

3. Set up the HADR on the primary and standby pods by using the`setup_config_hadr` script.
   See “setup_config_hadr” on page 92.

4. Start HADR services on the standby pod:

   ```
   oc exec -it <standby_pod> -- manage_hadr -start_as standby
   ```

5. Start HADR services on the primary pod:

   ```
   oc exec -it <primary_pod> -- manage_hadr -start_as primary
   ```

6. To verify that the HADR setup completed as expected, run the `-status` command:

   ```
   oc exec -it <primary/standby pod> -- manage_hadr -status
   ```

Managing HADR with CLI commands

You can use Db2 HADR CLI commands to show the HADR status, manually initiate a takeover, stop HADR, and enable the HADR Automatic Client Reroute feature

Showing the HADR status

You can monitor the state of your HADR configuration by running the **--manage_hadr** command with the **-status** option in the Db2 engine pod.

For example:

```
oc exec -it c-db2oltp-1616468282925295-db2u-0 -- manage_hadr -status
```

The following example shows the output from the primary database with an HADR setup that includes two standbys; note the second status section:

```
################################################################################
###           Db2 Advanced Edition high availability and                 ###
###             disaster recovery (HADR) management                   ###
################################################################################


Running HADR action -status on the database BLUDB ...
################################################################################################
###                          The HADR status summary                                         ###
################################################################################################
Database Member 0 -- Database BLUDB -- Active -- Up 0 days 00:00:39 -- Date
2021-05-28-03.47.31.838856

####### Primary - Standby 1 ######
                            HADR_ROLE = PRIMARY
                        HADR_SYNCMODE = NEARSYNC
                           HADR_STATE = PEER
                  PRIMARY_MEMBER_HOST = c-db2oltp-crd-hadr-primary-db2u-0|172.30.174.210
                  STANDBY_MEMBER_HOST = 172.30.247.77
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 05/28/2021 03:46:54.501837 (1622173614)
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 264947956
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 264947956
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 264947956
                       PEER_WINDOW_END = 05/28/2021 03:49:31.000000 (1622173771)

####### Primary - Standby 2 ######
                            HADR_ROLE = PRIMARY
                        HADR_SYNCMODE = SUPERASYNC
                           HADR_STATE = REMOTE_CATCHUP
                  PRIMARY_MEMBER_HOST = c-db2oltp-crd-hadr-primary-db2u-0|172.30.174.210
                  STANDBY_MEMBER_HOST = 172.30.105.147
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 05/28/2021 03:46:55.280238 (1622173615)
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 264947956
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 264947956
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 264947956


GOVERNOR: RUNNING
The HADR action -status was successful
```

Manually initiating an HADR takeover

To initiate a graceful takeover of the standby database, use the **--manage_hadr** command with the **-takeover** option in the Db2 engine pod. To switch to an auxiliary database, use the optional **aux** and **force** parameters with the **-takeover** option.

HADR offers two types of takeover: role switch and failover. Role switch, sometimes called graceful takeover or non-forced takeover, can be performed only when the primary is available and it switches the role of primary and standby. This method can be used to switch database roles to perform scheduled maintenance of the primary database/cluster.

Failover, or forced takeover, can be performed when the primary is not available. Governor will automatically fail over from primary to principal standby if a primary failure occurs. If multiple standbys

are set up, a manual failover can be performed to the auxiliary standby in the event that the primary and principal standby are unavailable.

The following example shows the use of the **--manage_hadr** command with the **-takeover** option:

```
oc exec -it c-db2oltp-1616469166354348-db2u-0 -- manage_hadr -takeover
```

The command output shows the current primary database and then the new primary database (leader). The following example shows the output when issuing the takeover from the principal standby:

```
################################################################################
###           Db2 Advanced Edition high availability and               ###
###             disaster recovery (HADR) management                 ###
################################################################################


Running HADR action -takeover on the database BLUDB ...
################################################################################
###                       Running takeover HADR on BLUDB                     ###
################################################################################
Executing HADR role-switch via Db2 Governor ...
current leader is c-db2oltp-1616468282925295-db2u-0
new leader is c-db2oltp-1616469166354348-db2u-0


################################################################################
###             The manage_hadr command completed an HADR takeover.          ###
################################################################################


The HADR action -takeover was successful
```

To validate that the takeover was successful, use the **--manage_hadr** command with the **-status** option to verify that the standby database now has the primary HADR role, and that the two databases are connected.

Graceful takeover to auxiliary standby

A graceful takeover (role-switch) to the auxiliary standby might be required if the primary and principal standby clusters are undergoing maintenance.

The following example shows the use of the **--manage_hadr** command with the **-takeover aux** option:

```
oc exec -it c-aux-db2u-0 -- manage_hadr -takeover aux
```

This command ensures that Governor does not try to automatically failover back to the primary or principal standby, and then executes a takeover to the auxiliary standby database. The following example shows the output when issuing a takeover from the auxiliary standby:

```
################################################################################
###           Db2 Advanced Edition high availability and               ###
###             disaster recovery (HADR) management                 ###
################################################################################


Running HADR action -takeover on the database BLUDB ...
################################################################################
###        Running auxiliary standby takeover HADR on BLUDB                  ###
################################################################################
Executing HADR takeover to auxiliary standby ...
2021-05-28 03:52:36,203 INFO: using etcd3 as truth_manager
2021-05-28 03:52:36,247 INFO: Attempt connect to: 172.30.139.33:2379
2021-05-28 03:52:36,250 INFO: Established new connection to: 172.30.139.33:2379
status: <etcd3.client.Status object at 0x1513ee625c10>
2021-05-28 03:52:36,250 INFO: verify endpoint success
2021-05-28 03:52:36,250 INFO: Using force_etcd ip: dummy
################################################################################
###                       Running takeover HADR on BLUDB                     ###
################################################################################
takeover hadr on db BLUDB
DB20000I  The TAKEOVER HADR ON DATABASE command completed successfully.
```

```
################################################################################
###           The manage_hadr command completed an HADR takeover.            ###
################################################################################

The HADR action -takeover was successful
```

Forced takeover (failover) to auxiliary standby

A forced takeover (failover) to the auxiliary standby might be required if both the primary and principal standby clusters are unavailable.

The following example shows the use of the **--manage_hadr** command with the **-takeover aux force** option:

```
oc exec -it c-aux-db2u-0 -- manage_hadr -takeover aux force
```

This command ensures that Governor does not try to automatically failover back to the primary or principal standby, and then executes a takeover by force to the auxiliary standby database. The following example shows the output when issuing a forced takeover from the auxiliary standby:

```
################################################################################
###            Db2 Advanced Edition high availability and                    ###
###              disaster recovery (HADR) management                         ###
################################################################################

Running HADR action -takeover on the database BLUDB ...
################################################################################
###         Running auxiliary standby takeover HADR on BLUDB                 ###
################################################################################
Executing HADR takeover to auxiliary standby ...
2021-05-28 03:55:41,617 INFO: using etcd3 as truth_manager
2021-05-28 03:55:41,665 INFO: Attempt connect to: 172.30.139.33:2379
2021-05-28 03:55:41,669 INFO: Established new connection to: 172.30.139.33:2379
status: <etcd3.client.Status object at 0x1505ab0c6cd0>
2021-05-28 03:55:41,669 INFO: verify endpoint success
2021-05-28 03:55:41,669 INFO: Using force_etcd ip: dummy
################################################################################
###                        Running takeover HADR on BLUDB                    ###
################################################################################
takeover hadr on db BLUDB by force
DB20000I  The TAKEOVER HADR ON DATABASE command completed successfully.

################################################################################
###           The manage_hadr command completed an HADR takeover.            ###
################################################################################

The HADR action -takeover was successful
```

Restarting HADR after a failover

After a failover (forced takeover), HADR is stopped on the former primary database. This database might need to be manually reintegrated as a standby.

This procedure assumes that a new primary is currently running, (that takeover to an original standby was performed when the original primary became unavailable).

Restarting HADR on the designated auxiliary standby database

An auxiliary standby database might have been the primary database because of maintenance or unavailability of the primary and principal standby databases. When the primary and/or principal standby databases became available, and a takeover was executed back to one of those two databases, HADR is automatically stopped on the auxiliary database.

About this task

Because Governor, the automated failover mechanism, is not running on auxiliary databases, HADR must be explicitly restarted by using the **manage_hadr** tool with the **-start_as standby** option.

The **-status** option with the **manage_hadr** tool displays the following message to indicate that HADR is not started:

```
oc exec -it c-${aux1_id}-db2u-0 -n ${aux1_project} -- manage_hadr -status
Output:

################################################################################
###            Db2 Advanced Edition high availability and            ###
###              disaster recovery (HADR) management                 ###
################################################################################


Running HADR action -status on the database BLUDB ...
Unable to query HADR status.


################################################################################
* The requested HADR operation failed.
* Correct the errors and rerun the manage_hadr command.
################################################################################
```

Procedure

Use the **manage_hadr** tool to start HADR as standby.

```
oc exec -it c-db2-aux-db2u-0 -- manage_hadr -start_as standby
```

Restarting HADR on the primary or principal standby databases after failover to an auxiliary standby database
With Governor running on the designated primary and principal standby databases, HADR is automatically restarted after a failover to another database. But in a scenario where the database unexpectedly becomes unavailable (for example, a site outage), the old and new primaries' log streams might diverge, resulting in HADR failing to start.

About this task

You would see this message in the db2 diaglog (located in the Db2u pod in `${DIAGPATH}/NODE000`):

```
MESSAGE : ADM12500E  The HADR standby database cannot be made consistent with
          the primary database. The log stream of the standby database is
          incompatible with that of the primary database. To use this database
          as a standby, it must be recreated from a backup image or split
          mirror of the primary database.
```

If this scenario occurs, you should make an online backup from the current primary database and restore it to the standby database that is failing to start. You can do this by manually taking a backup from the current primary and re-running the `setup_config_hadr` script with **--db-role standby**.

Procedure

1. Stop HADR on the database that cannot reintegrate:

   ```
   oc exec -it c-db2-primary-db2u-0 -- manage_hadr -stop
   ```

2. Determine the database that is the current primary by using the **manage_hadr** tool with **-status** option.

   In the following example, `db2oltp-aux` is the current primary database, after a forced takeover. Note that `HADR_ROLE = PRIMARY`.

   ```
   oc exec -it c-db2oltp-aux-db2u-0 -- manage_hadr -status

   # Output:
   ################################################################################
   ###            Db2 Advanced Edition high availability and            ###
   ###              disaster recovery (HADR) management                 ###
   ################################################################################
   ```

```
Running HADR action -status on the database BLUDB ...
################################################################################
###                        The HADR status summary                           ###
################################################################################
Database Member 0 -- Database BLUDB -- Active -- Up 0 days 00:00:39 -- Date
2021-05-28-03.47.31.838856

####### Primary - Standby 1 ######
                            HADR_ROLE = PRIMARY
```

3. Exec into current primary database Db2 pod and switch to the database instance owner:

```
oc exec -it c-db2oltp-aux-db2u-0
su - db2inst1
```

4. Initiate an online backup of the database to the backup location (${BACKUPDIR} (/mnt/backup):

```
db2 backup db BLUDB online to ${BACKUPDIR}
```

5. Copy the keystore into the backup location:

```
tar -cjvf ${BACKUPDIR}/keystore.tar -C ${KEYSTORELOC}.
```

6. Update permissions on the backup directory so the Db2 instance owner/group has read-write access:

```
sudo chmod 755 -R  /mnt/backup
```

7. Copy the Db2 backup file from the current primary database to the standby database:

```
## Copy from current primary database to a directory on the host called /tmp/hadr
oc rsync c-db2oltp-aux-db2u-0:/mnt/backup/ /tmp/hadr
```

8. Run the setup_config_hadr script again to restore the database.
   - Use **standby** for the **--db-role** to ensure that the database is reconfigured as a standby.
   - If the database that is being reinitialized is the former primary database, use the designated principal standby as primary, and use the designated primary as standby, leaving the auxiliary standby databases as auxiliaries:

     ```
     oc exec -it c-db2oltp-primary-db2u-0 -- setup_config_hadr --db-role standby --primary-
     name db2oltp-standby --standby-name db2oltp-primary --primary-port 31384 --standby-port
     32457 --aux1-name db2oltp-aux --aux1-port 32649 --etcd-host my-etcd-client.my-etcd --etcd-
     port 2379 –multicluster
     ```

   - If the database that is being reinitialized is the former principal standby database, use the same parameters as used in the original setup:

     ```
     oc exec -it c-db2oltp-standby-db2u-0 -- setup_config_hadr --db-role standby --primary-
     name db2oltp-primary --standby-name db2oltp-standby --primary-port 32457 --standby-port
     31384 --aux1-name db2oltp-aux --aux1-port 32649 --etcd-host my-etcd-client.my-etcd --etcd-
     port 2379 --multicluster
     ```

9. Exec into the Db2 pod again, and as the Db2 instance owner, check the HADR configuration by setting HADR_LOCAL_SVC.

```
db2 get db cfg for bludb | grep -i hadr_local_svc

# Output:
HADR local service name           (HADR_LOCAL_SVC) = 60007|31384
```

10. Verify that the first port number is correct for its designated original role:
    - Primary: 60006
    - Standby: 60007
    - Aux1: 60008
    - Aux2: 60009

If incorrect, edit the HADR configuration setting HADR_LOCAL_SVC so that it uses the correct port. Only update the first port number, and use the existing value for the second port number:

```
db2 “update db cfg for BLUDB using HADR_LOCAL_SVC 60006|31384”
```

11. Exit the pod, and start HADR on the database as a standby:

```
oc exec -it c-db2oltp-primary-db2u-0 -- manage_hadr -start_as standby
```

Stopping HADR

To stop the HADR process on the database and to stop the Governor automated failover mechanism, run the **manage_hadr** command with the **-stop** option.

The following example shows the command:

```
oc exec -it c-db2oltp-1616469166354348-db2u-0 -- manage_hadr -stop
```

Enabling HADR Automatic Client Reroute

You can enable the Db2 HADR Automatic Client Reroute (ACR) feature by running the **manage_hadr** command with the **-enable_acr** option.

You must run this command on both the primary and standby deployments. If all client connections are coming from within the same OpenShift cluster, use the `engn-svc` service name and port of the other deployment. If client connections are external, use the host name or IP address of the Db2 instance, and the `NodePort` of the `engn-svc` service of the other deployment.

Internal clients

On clients within your OpenShift cluster, run the following command to enable HADR automatic client reroute:

```
oc exec -it primary-db2-svc-id-db2u-0 -- manage_hadr -enable_acr standby-db2-svc-id-db2u-engn-
svc:port
oc exec -it standby-db2-svc-id-db2u-0 -- manage_hadr -enable_acr primary-db2-svc-id-db2u-engn-
svc:NodePort
```

Where *db2-svc-id* is the Db2 service identifier on Red Hat OpenShift that is associated with the primary and standby deployments. Use the **oc get service** command to find the exact values.

For example, if your primary deployment is db2oltp-1616468282925295 and your standby deployment is db2oltp-1616469166354348, the commands would look like this:

```
oc exec -it c-db2oltp-1616468282925295-db2u-0 -- manage_hadr -enable_acr c-
db2oltp-1616469166354348-db2u-engn-svc:50000
oc exec -it c-db2oltp-1616469166354348-db2u-0 -- manage_hadr -enable_acr c-
db2oltp-1616468282925295-db2u-engn-svc:50000
```

External clients

On external clients, run the following command to enable HADR automatic client reroute:

```
oc exec -it primary-db2oltp-svc-id-db2u-0 -- manage_hadr -enable_acr
host_name_or_IP_address:NodePort
oc exec -it standby-db2oltp-svc-id-db2u-0 -- manage_hadr -enable_acr
host_name_or_IP_address:NodePort
```

Where *host_name_or_IP_address* is the host name or IP address of the Db2 instance. Use the **oc get service** command to find the *NodePort* value for the service. For example, if your primary deployment is db2oltp-1616468282925295 and your standby deployment is db2oltp-1616469166354348, the commands would look like this:

```
oc exec -it c-db2oltp-1616468282925295-db2u-0 -- manage_hadr -enable_acr 9.121.221.159:31900
oc exec -it c-db2oltp-1616469166354348-db2u-0 -- manage_hadr -enable_acr 9.121.221.159:31806
```

When ACR is enabled, follow the instructions at "Connecting clients to the HADR configuration" on page 120.

For more information, see Db2 automatic client reroute with HADR.

Scaling up Db2

You can run a script to add more memory and CPU to the Db2 service on Red Hat OpenShift to support high-availability or increase processing capacity.

Procedure

1. If you have deployed a Db2uCluster CR, use the following command to edit the OpenShift® Db2uCluster CR for the Db2 pod to set higher memory and CPU limits:

   In the following example, we patch the Db2ucluster custom resource with these entries to set CPU to 12 vCPU and memory to 36 Gi, where DB2UCLUSTER is the name of your Db2uCluster custom resource:

   ```
   oc patch db2ucluster <DB2UCLUSTER> --type merge --patch '{"spec": {
   "podConfig": {
       "db2u": {
           "resource": {
               "db2u": {
                   "limits": {
                       "cpu": "8",
                       "memory": "16Gi"
                   }
               }
           }
       }
   }
   }}'
   ```

   If you have deployed a Db2uInstance CR, use the following command to patch the Db2uInstance custom resource with these entries to set CPU to 12 vCPU and memory to 36 Gi, where DB2UINSTANCE is the name of your Db2uInstance custom resource:

   ```
   oc patch db2uinstance <DB2UINSTANCE> --type merge --patch '{"spec": {
   "podTemplate": {
       "db2u": {
           "resource": {
               "db2u": {
                   "limits": {
                       "cpu": "8",
                       "memory": "16Gi"
                   }
               }
           }
       }
   }
   }}'
   ```

   After the patch command, the db2 pods will restart. Wait for 1/1 Ready state of the restarted db2 pods before proceeding to the next step.
2. "Exec into the Db2 pod" on page 58.
3. Disable the Wolverine high availability monitoring process:

   ```
   sudo wvcli system disable -m "Disable HA before Db2 maintenance"
   ```
4. On the head pod of your Db2 cluster, run the following series of commands to:
   - Create the `update_mem.sh` script file.
   - Deactivate and stop Db2.
   - Disable any remote database connections.
   - Update the Db2 **`instance_memory`** configuration parameter to use the normalized percentage value that was derived from the higher system memory limit.

- Re-run the **autoconfigure** command to start using updated **instance_memory** percentage and reapply any overrides.
- Re-enable Db2 remote connections, start Db2, and activate the database.

```
cat <<EOF > /db2u/tmp/update_mem.sh
#!/bin/bash
source /etc/profile
source /db2u/scripts/include/common_functions.sh
source /db2u/scripts/include/db2_functions.sh
# Get PID1 environ required for access OS envvar MEMORY_LIMIT
export_pid1_env
source /db2u/scripts/include/db2_memtune_functions.sh
# Deactivate database and stop Db2
db2 terminate
deactivate_multiple_db
db2stop force
rah 'ipclean -a'
# Disable remote connections
db2set DB2COMM -null
# Update the normalized instance_memory % value derived from higher MEMORY_LIMIT
rm -f /mnt/blumeta0/SystemConfig/instancememory
# db2start prior to activate db bludb and connection within db2_autoconf.clp
db2start
set_instance_memory
db2stop
db2start
# Re-run autoconfigure to start using updated instance_memory %, and reapply any overrides
run_autoconfigure
apply_cfg_setting_to_db2 "-all"
# Re-enable Db2 remote connections, start Db2 and activate db
db2stop force
rah 'ipclean -a'
db2set DB2COMM=TCPIP,SSL
db2start
activate_multiple_db
EOF
```

5. Set permissions on the `update_mem.sh` script and run the script:

```
chmod +x /db2u/tmp/update_mem.sh
su - db2inst1 -c "/db2u/tmp/update_mem.sh"
```

   The script runs the function `export_pid1_env`, which parses the PID 1 environment file `/proc/1/env` to get the value of the operating system memory limit environment variable and make the needed adjustments.
6. Re-enable the Wolverine high availability monitoring process:

```
sudo wvcli system enable -m "Enable HA after Db2 maintenance"
```

Enabling the Db2 Spatial Extender

You can enable the Db2 Spatial Extender component to store geospatial data in special data types and provide a spatial grid index to improve performance.

Procedure

1. Exec into the Db2 pod:

```
oc exec -ti c-db2ucluster-sample-db2u-0 bash
```

2. Become the Db2 instance owner:

```
su - db2inst1
```

3. Enable Spatial Extender by running the following command:

```
db2se enable_db database_name
```

What to do next

For more information, see Db2 Spatial Extender.

Gathering information for Db2 support

You can gather information about your Db2 environment to be used by IBM Support, including running a support tool and viewing diagnostic logs.

Using the db2support tool

You can run a command to gather information about your Db2 environment to be used by IBM Support for Db2.

Procedure

1. Exec into the Db2 pod.
2. Become the Db2 instance owner:

```
su - ${DB2INSTANCE}
```

3. Run the **db2support** tool:

```
db2support
```

   The resulting `db2support.zip` file is saved in the current directory, `/mnt/blumeta0/home/db2inst1`.
4. Exit the pod and copy the zip file out of the Db2 pod:

```
oc cp c-db2wh-1611278932361-db2u-0:/mnt/blumeta0/home/db2inst1/db2support.zip
```

Viewing log files

In general, there are two categories of Db2 log files, container logs and Db2 diagnostic logs.

Procedure

1. To view Db2 container log files, do the following:
   a) Get the names of all Db2 database service pods:

```
kubectl -n <namespace> get po --selector type=engine
```

   b) Use `kubectl logs` to view the Db2 database container logs:

```
kubectl -n <namespace> logs <Db2 engine POD name> | more -
```

2. To view Db2 diagnostic log files, do the following:
   a) "Exec into the Db2 pod" on page 58.
   b) Go to the Db2 diagnostic logs folder `/mnt/blumeta0/db2/log/`.

      Tip: The OS environment variable DIAGPATH defines the Db2 diagnostic logs folder.
   c) Sort all the `db2diag.*.log` files by timestamp and view/tail the most recent one:

```
ls -latr ${DIAGPATH}/db2diag.*.log
```

Enabling Db2 monitoring

You can use the db2mon tool to collect monitoring data on your Db2 databases in Red Hat OpenShift.

About this task

A db2mon.sh shell script provides a sampling of data collection and analysis. The script also checks that the prerequisites for db2mon are met. The `db2mon.sh` script is packaged with Db2 and can be found in the `${BLUMETAHOME}/db2inst1/sqllib/samples/perf/` directory.

For more information, see Collecting and reporting performance monitor data with db2mon.

Procedure

1. Log into the Db2 pod:

```
oc exec -it db2u-pod -- bash
```

2. Log into the Db2 instance as the db2inst1 user:

```
su - db2inst1
```

3. Navigate to the folder where the `db2mon.sh` script is present

```
cd ${BLUMETAHOME}/db2inst1/sqllib/samples/perf/
```

4. Issue the following command as the db2inst1 user to run the `db2mon.sh` script:

```
sh db2mon.sh dbname > /tmp/db2mon.out
```

Where *dbname* is the name of the database to be monitored.

By default, the script collects MON_GET data over a 30-second period. The script can take a second optional argument to collect data over a different period, which is specified in seconds. For example, to produce an online report that collects performance data for 120 seconds:

```
db2mon.sh dbname 120 > /tmp/db2mon-120s.out
```

5. To copy the report from the Db2 container to the host, you can run the following command:

```
oc cp db2u-podname:/tmp/output-filename destination-path
```

Where *destination-path* is the path on the host where you want to copy the report to.

Changing the name of the default Db2 database

To change the name of your default Db2 database, complete the following steps.

Procedure

1. Run the following command, substituting the new default database name for MYDB:

```
oc patch db2ucluster -n <PROJECT> <DB2UCLUSTER-NAME> --type merge -p '{"spec":{"environment":
{"database":{"name":"MYDB"}}}}'
```

2. Verify that the corresponding db2uconfig configMap reflects the changed DBNAME value:

```
oc get cm -n <PROJECT> -o yaml c-<DB2UCLUSTER-NAME>-db2uconfig | grep DBNAME
```

3. Delete one or more corresponding Db2 engine pods to enable the configMap changes to take effect:

```
oc delete pods -n <PROJECT> c-<DB2UCLUSTER-NAME>-db2u-0
```

Managing Db2 transaction logs

When your instance is running many transactions, your archive log directory can fill and you might have to remove log files to free up space.

About this task

You can either manually delete unneeded log files and directories or use the **prune history** command.

If the Db2 instance is failing because of a lack of space, prior to pruning logs you can edit the associated PVC and increase the size that is allocated. The storage class that you are using must have `allowVolumeExpansion` set to `True`.

Note: Before you remove log directories, make sure that no backup or restore operation is running. You can use the **db2 list utilities** command to confirm.

Procedure

Use one of these methods to manage log files:

Manually delete unneeded log directories and files

By default, Db2 archives transaction logs to disk with the following settings:

```
First log archive method (LOGARCHMETH1): DISK:/mnt/bludata0/db2/archive_log/
Number of primary log files (LOGPRIMARY): 20
Number of secondary log files (LOGSECOND): 30
```

The following formula is used to determine how many logs are kept in each multiple logical node (MLN):

```
(log primary + log secondary) * 2
```

So by default, (20 + 30) * 2 or 100 logs are kept.

To maintain this number of logs, assume that you have a deployment of one MLN. Archived logs are kept in sub-directories whose name starts with "C" in the `/mnt/bludata0/db2/archive_log/db2inst1/BLUDB/NODE0000/LOGSTREAM0000` directory.

So if directories from `C0000000` to `C0000005` exist, you would remove `C0000000` to `C0000004`.

Then, within the `C0000005` directory, you would keep only the latest 100 log files. The file names begin with "S." If the `C0000005` directory has `S0000000.LOG` through `S0000200.LOG`, you would keep `S0000101.LOG` through `S0000200.LOG`.

You then repeat this process for each MLN, NODE0000 to NODE*xxxx*.

To remove the log directories, use the following command while logged in as the db2inst1 user:

```
sudo rm -r directory
```

Use the Db2 `prune history` command

Find the timestamp of the logs that you want to prune (up to and including) by running the following command:

```
db2 list history all for database-name | egrep "B P|B D|X D"
```

The command returns the timestamps of the following storage items:

```
X D:log archive
B P:tablespace backup
B D:full backup
```

The basic prune history command, db2 `prune history` *`timestamp`*, allows Db2 to maintain a restore set. If you use db2 `prune history` *`timestamp`* `with force option` Db2 does not

keep the restore set. You can also use command parameters such as `AND DELETE` to specify that associated log archives are physically deleted when the history file entry is removed, and `LOGFILE PRIOR TO` *log-file-name* to specify the log files to delete by file name.

For more details, see the following topics:

- PRUNE HISTORY/LOGFILE command
- Hands-on example for PRUNE HISTORY command

Connecting to Db2

You can securely connect clients, applications, and users to the Db2 service in Red Hat OpenShift.

Clients

You can remotely catalog the Db2 database and connect your applications to the Db2 database server.

Cataloging the database

You can remotely catalog the Db2 database by running CLP or CLPPlus commands.

The following variables are used in the catalog commands:

- *host_name_or_IP_address* – The host name or IP address of the Red Hat OpenShift instance.
- *db2_ssl_port* – The port number of the Db2 secure sockets layer (SSL) instance.
- *db2_port* – The port number of the Db2 instance.
- *database_name* – The name of the Db2 database.
- *database_alias_name* – The alias name of the remote Db2 database.
- *server_alias_name* – The alias for the remote server of the Db2 instance.
- *authentication_method* – The authentication method for connecting to the remote database. For user name and password, use SERVER_ENCRYPT.

CLP

You can catalog the remote database by using the CLP either with or without SSL:

SSL

```
db2 catalog tcpip node server_alias_name remote host_name_or_IP_address server
db2_ssl_port security SSL
db2 catalog db database_name as database_alias_name at node server_alias_name
authentication authentication_method
```

Non-SSL

(Only supported for user name and password)

```
db2 catalog tcpip node server_alias_name remote host_name_or_IP_address server db2_port
db2 catalog db database_name as database_alias_name at node server_alias_name
authentication authentication_method
```

CLPPlus

For CLPPlus to connect to a Db2 server by using Red Hat OpenShift authentication, you must configure a data source name (DSN) first in a `db2dsdriver.cfg` configuration file by running the following command:

```
db2cli writecfg add -dsn dsn_alias -database database_name -host host_name_or_IP_address
-port db2_ssl_port -parameter "parameter"
```

Where *parameter* is SecurityTransportMode=SSL – Used for SSL connections

The `db2dsdriver.cfg` configuration file is an XML file, typically located in the `sqllib/cfg` directory, that contains a list of DSN aliases and their properties. The following example of a `db2dsdriver.cfg` configuration file shows the configurations that are used to establish a

connection to a database service instance. The configuration file provides the DSN alias, the database name, the host name (or IP address), and the **Authentication** and **SecurityTransportMode** parameter values:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<configuration>
    <dsncollection>
        <dsn alias="<data_source_name>" name="database_name" host="host_name_or_IP_address"
port="db2_ssl_port">
            <parameter name="Authentication" value="authentication_method"/>
            <parameter name="SecurityTransportMode" value="SSL"/>
        </dsn>
    </dsncollection>
    <databases>
        <database name="database_name" host="host_name_or_IP_address" port="db2_ssl_port"/>
    </databases>
</configuration>
```

For more details about connecting to DSN aliases with CLPPLUS, see DSN aliases in CLPPlus.

Connecting applications to the Db2 database server

To connect your applications via JDBC/ODBC to the Db2 database server, you must install IBM Data Server driver packages.

Procedure

1. Download the driver packages from the Download Fix Packs by version for IBM Data Server Client Packages web page. Choose a download link from the Version 11.5 column.
2. After you download the required package, choose one of the following methods to install those drivers. The method that you choose depends on the database connectivity method that is used by the client application.
   - For ODBC/CLI applications, see Installing the IBM Data Server Driver for ODBC and CLI software on Linux and UNIX operating systems.

     For instructions on how to configure the ODBC/CLI driver setting after you install the driver packages, see Configuring the IBM Data Server Driver for ODBC and CLI.
   - For JDBC/SQLJ applications, see Installing the IBM Data Server Driver for JDBC and SQLJ on Db2 on Linux, UNIX, and Windows systems.

Managing users with LDAP in a Db2 on OpenShift cluster

Managing users with the internal LDAP deployment

You can use the optional LDAP service that's included as part of the Db2 for Red Hat OpenShift offering to add and manage users for your Db2 instance.

Adding Db2 Users

1. Run the following commands to setup a terminal session with the tools pod. Set the variables PROJECT and RELEASE_NAME accordingly.

   ```
   PROJECT=""
   RELEASE_NAME=""
   ldap_pod=$(oc get po -n ${PROJECT} -o name | grep ${RELEASE_NAME}-ldap)
   ```

2. Run the script, **addLdapUser.py**, to add an LDAP user. If the password argument is not provided, a prompt will be provided to enter the password securely.

   ```
   oc rsh ${ldap_pod} /opt/ibm/ldap_scripts/addLdapUser.py
   ```

   Usage: `addLdapUser.py [-h] -u USERNAME [-p PASSWORD] -r {admin,user}`

where:

- `-h, --help` displays the help message and exit option
- `-u USERNAME, --username USERNAME` defines the username for the new LDAP user (default: None)
- `-p PASSWORD, --password PASSWORD` defines the password for the new LDAP user (default: Prompt if not specified)
- `-r {admin,user}, --roletype {admin,user}` defines the role for the new LDAP user (admin or user)(default: None)

3. Verify the newly created LDAP user ID and credential by following these steps:
   a. Exit from the LDAP pod.

   ```
   exit
   ```

   b. Log in to the Db2 pod.

   ```
   oc rsh db2u-deployment-db2u-0 /bin/bash
   ```

   c. Verify that the new LDAP user exists.

   ```
   id ldap-user
   ```

   d. Log in to a Db2 instance.

   ```
   su - db2inst1
   ```

   e. Connect to a database by using the newly created LDAP user ID:

   ```
   db2 connect to bludb user ldap_user using ldap_password
   ```

Changing a Db2 user's password

You can change the password of an existing LDAP user by running the script, **changePassword.py**.

```
oc rsh ${ldap_pod} /opt/ibm/ldap_scripts/changePassword.py
```

Usage: `changePassword.py [-h] -u USERNAME [-cp CURRENTPASSWORD] [-np NEWPASSWORD]`

where:

- `-h, --help` displays help and exit option
- `-u USERNAME, --username USERNAME` defines the username for the new LDAP user (default: None)
- `-cp CURRENTPASSWORD, --currentpassword CURRENTPASSWORD` defines the current password for the LDAP user (default: prompt if not specified)
- `-np NEWPASSWORD, --newpassword NEWPASSWORD` defines the new password for the LDAP user (default: Prompt if not specified)

Deleting a Db2 user

You can delete an existing LDAP user by running the script, **removeLdapUser.py**

```
oc rsh ${ldap_pod} /opt/ibm/ldap_scripts/removeLdapUser.py
```

Usage: `removeLdapUser.py [-h] -u USERNAME`

where:

- `-h, --help` displays help and exit option

- `-u USERNAME, --username USERNAME` defines the username for the LDAP user to be removed (default: None)

Connecting to an external LDAP store

By default, Db2 for Red Hat OpenShift uses a self-contained OpenLDAP server for authentication and authorization. However, you can use an external OpenLDAP server instead.

Procedure

1. Create LDAP entries for the following groups:

   bluadmin
   This is the group for Db2 administrators. The value of its **CN** attribute (the full or common name) must be **bluadmin**.

   bluusers
   This is the group for Db2 users. The value of its **CN** attribute must be **bluusers**.

   Note:

   - Both groups must have the same location, that is, with the exception of their CN attributes, the DNs of the two groups must be identical.
   - For each entry, specify **objectclass: top** and **objectclass: posixGroup** attributes.

   For example:

   - For the **bluadmin** group:

     ```
     dn: cn=bluadmin,ou=groups,dc=example,dc=com
     objectClass: top
     objectClass: posixGroup
     cn: bluadmin
     gidNumber: unique_bluadmin_group_ID
     ```

   - For the **bluusers** group:

     ```
     dn: cn=bluusers,ou=groups,dc=example,dc=com
     objectClass: top
     objectClass: posixGroup
     cn: bluusers
     gidNumber: unique_bluusers_group_ID
     ```

2. Create an LDAP entry for the **bluadmin** user, who must be part of the **bluadmin** group.

   For this entry, specify **objectclass** attributes of **account**, **posixAccount**, and **top**. For example:

   ```
   dn: uid=bluadmin,ou=users,dc=example,dc=com
   uid: bluadmin
   cn: bluadmin
   objectClass: account
   objectClass: posixAccount
   objectClass: top
   loginShell: /bin/bash
   uidNumber: unique_bluadmin_user_ID
   gidNumber: unique_bluadmin_group_ID

   gecos: bluadmin
   ```

3. Modify the **bluadmin** group so that it includes a set of **memberuid** attributes for the **bluadmin** user.

   For the first **memberuid** attribute, use the value of the **bluadmin** user's **uid** attribute. For the second **memberuid** attribute, use the value of the **bluadmin** user's **dn** attribute. For example:

   ```
   dn: cn=bluadmin,ou=groups,dc=example,dc=com
   changetype: modify
   add: memberuid
   memberuid: bluadmin
   memberuid: uid=bluadmin,ou=users,dc=example,dc=com
   ```

4. Ensure that the host name of the external OpenLDAP server is resolvable from all Db2 pods.

5. Configure Db2 to act as a client to an external OpenLDAP server. When creating the Db2u Cluster instance, fill out the corresponding LDAP keys accordingly. For example, in YAML format:

```
spec:
  environment:
    ldap:
      admin: ""
      adminGroup: "bluadmin"
      blueAdminPassword: "password123"
      domain: "dc=somedomain,dc=com"
      enabled: true
      password: ""
      port: "389"
      server: "my-ldap-server.example.com"
      userGroup: "bluusers"
```

Note: If you specify a group base DN or user base DN:

- The group base DN must be at the same location as (that is, must be in the same directory as) the **bluadmin** and **bluusers** groups.
- The user base DN is the same DN that you specified for the **bluadmin** user in step 2, but without the **uid=bluadmin** attribute.

You can use the **adminGroup**, **userGroup**, and **admin** values to override the default names for the administrative group (default is **bluadmin**), user group (default is **bluusers**), and administrative user (default is **bluadmin**). For example, you might want to use different groups and users depending on whether your system is a production or test system. All other requirements for these groups and user remain unchanged.

6. If needed, create additional Db2 administrators by adding them to the **bluadmin** group, and create additional Db2 users by adding them to the **bluusers** group. Use the same sort of approach that you used for the **bluadmin** user in steps “2” on page 117 and “3” on page 117. The **uid** and **uidNumbers** attribute values of each administrator and user must be unique.

Db2 ports and services

You can use the Db2 NodePort service or an external-facing Ingress Controller to connect applications to Db2 on Red Hat OpenShift.

Retrieving the Db2 port number

You can use the Db2 `NodePort` service to connect JDBC or ODBC applications to Db2 on Red Hat OpenShift.

You can find the `NodePort` by using the web console or commands:

Web console

After you deploy the database, the console displays the `NodePort` that applications should use for client connections in the **JDBC Connection URL** field of the **Access information** section of the database details page for both SSL and non-SSL connections.

Commands

The command differs for SSL and non-SSL ports:

SSL port

```
oc get svc -n project db2_service_name -o jsonpath='{.spec.ports[?(@.name=="ssl-
server")].nodePort}'
```

Non-SSL port

```
oc get svc -n project db2_service_name -o jsonpath='{.spec.ports[?(@.name=="legacy-
server")].nodePort}'
```

Where:

- *project* is the OpenShift project where Db2 is deployed.

- *service_name* is the unique identifier that is assigned to each Db2 deployment. The service name always starts with `c-db2oltp`, for example `c-db2oltp-1605022957148004-db2u-engn-svc`.

For more information, see Configuring ingress cluster traffic using a NodePort.

Configuring the Db2 NodePort with an Ingress Controller

If you use an external infrastructure node to route external Db2 traffic into the Red Hat OpenShift cluster, the cluster might be in a private zone and you need to configure an external-facing Ingress Controller to route the traffic to the OpenShift nodes.

About this task

Because Db2 is externally exposed through a NodePort, the Ingress Controller also needs to expose the NodePort in order to allow traffic into the cluster.

The configuration below is only applicable with an HAProxy Ingress Controller. For more detail about configuring networking, see Understanding networking in the OpenShift documentation.

Procedure

1. On the infrastructure node, open the HAProxy configuration file located at `/etc/haproxy/haproxy.cfg`.
2. Modify the `haproxy.cfg` file to include the OpenShift NodePort:

```
frontend db2
        bind *:Db2 NodePort
        default_backend db2u
        mode tcp
        option tcplog
backend db2u
        balance source
        mode tcp
        server master0 Master0-privateIP:Db2 NodePort check
        server master1 Master1-privateIP:Db2 NodePort check
        server master2 Master3-privateIP:Db2 NodePort check
```

3. Reload HAProxy:

```
systemctl reload haproxy
```

Removing non-SSL ports

You can edit the Db2 configuration to remove non-SSL ports from your deployment to prevent non-SSL connections to the service and guarantee the highest security.

Procedure

1. Run the following command to edit the Db2 service configuration:

```
oc edit svc -n project c-service_name-db2u-engn-svc
```

   Where:

   - *project* is the name of the Red Hat OpenShift project where Db2 is deployed.
   - *service_name* is the identifier for the Db2 service instance, for example `c-db2oltp-1605022957148004-db2u-engn-svc`.

2. Remove the following block from the `spec.ports` section:

```
- name: legacy-server
    nodePort: 30279
    port: 50000
    protocol: TCP
    targetPort: 50000
```

3. Save the service.

Connecting clients to the HADR configuration

To connect to a Db2 high availability disaster recovery (HADR) configuration on Red Hat OpenShift, external clients can use the hostname and IP address of the master node and the `NodePort` service that is associated with the `${SVCENAME}-db2u-engn-svc` service as you would in a non-HADR deployment.

Prerequisites

1. Enable support for the Db2 HADR Automatic Client Reroute (ACR) feature by running the following command against the primary deployment. For example, if your primary deployment is `db2oltp-1573144443`:

```
oc exec -it db2oltp-1573144443-db2u-0 -- manage_hadr --enable_acr db2u-engn-svc:[port]
```

2. Discover the service name and the associated `NodePort` values for the Db2 service on the primary and standby deployments. For example, you can get the `db2u-engn-svc` value from the primary and standby HADR deployments and note down the service `NodePort` values.

Connection methods

The connection method differs for command-line interface (CLI) or ODBC applications and Java™ applications that use JDBC:

CLI or ODBC applications

If the HADR alternate servers are configured by using **`manage_hadr -enable_acr`**, then Db2 Automatic Client Reroute automatically routes the connection to the standby database by using the `ClusterIP` service (intra-cluster deployment) or `ExternalName` service (inter-cluster deployment).

You must set the following parameters in the `db2dsdriver.cfg` file:

- **Database section**
  - **name:** primary
  - **hostname:** IP address/hostname (if DNS resolution works) of the master node (or load balancer if the master is not exposed outside of the cluster).
  - **port:** Primary HADR `db2u-engn-svc` service `NodePort`
- **<acr> ... </acr> section**
  - **enableAcr:** true
  - **maxAcrRetries:** 30
  - **acrRetryInterval:** 2
  - **enableseamlessACR:** true
  - **enableAlternateServerListFirstConnect:** true
- **alternateserverlist section**
  - **name:** standby
  - **hostname:** IP address/hostname (if DNS resolution works) of the master node (or load balancer if the master is not exposed outside of the cluster).
  - **port:** Standby HADR `db2u-engn-svc` service `NodePort`

For applications with their own `odbc.ini` configuration files, add the same keywords to the respective DSN sections.

The following is an example `db2dsriver.cfg`:

```
<configuration>
  <dsncollection>
    <dsn alias="bludb" name="bludb" host="c-db2oltp-1628012388316353-db2u-engn-svc"
```

```
port="50000">
    </dsn>
  </dsncollection>
  <databases>
    <database name="bludb" host="c-db2oltp-1628012388316353-db2u-engn-svc" port="50000">
      <acr>
        <parameter name="enableAcr" value="true"/>
        <parameter name="maxAcrRetries" value="100"/>
        <parameter name="acrRetryInterval" value="2"/>
        <parameter name="enableAlternateServerListFirstConnect" value="true"/>
        <alternateserverlist>
          <server name="standby" hostname="c-db2oltp-1628012684942463-db2u-engn-svc"
port="50000"/>
        </alternateserverlist>
      </acr>
    </database>
  </databases>
</configuration>
```

The hostname of the primary and standby instances must be recorded in a reliable DNS, or in a UNIX environment the hostname should be added to the `/etc/hosts` file. If the client is within the cluster, use the fully qualified domain name (FQDN) of the `db2u-engn-svc` along with the IP address, and if outside the cluster use the external route that was created to connect to the client. For example:

```
172.30.127.129 c-db2oltp-1628012388316353-db2u-engn-svc.zen.svc.cluster.local
172.30.138.32 c-db2oltp-1628012684942463-db2u-engn-svc.zen.svc.cluster.local
```

You might also need to configure TCP/IP kernel parameters. This is a system-level configuration and is outside the scope of Db2. So, system administrators should configure the settings based on their needs. Suggestions for kernel parameters:

- **net.ipv4.tcp_retries2** = 2: The **tcp_retries2** value tells the kernel how many times to retry before killing a live TCP connection. To make a permanent change so that the value persists on node reboot, use this command: `sysctl -w net.ipv4.tcp_retries2=2 >> /etc/sysctl.conf`.
- The following kernel parameter values should be considered depending on the kind of workload and the environment:
  - **net.ipv4.tcp_keepalive_time** = 300
  - **net.ipv4.tcp_fin_timeout** = 30
  - **net.ipv4.tcp_keepalive_intvl** = 30
  - **net.ipv4.tcp_keepalive_probes** = 3

For more information, see the following topics:

- Configuration of Db2 automatic client reroute support for applications other than Java
- HOWTO: Configure non-Java for automatic client reroute

Java applications

- If you use a DriverManager interface for connections, set the **clientRerouteAlternateServerName** and **clientRerouteAlternatePortNumber** properties.
- If you use a DataSource interface for connections, you might also need to configure JNDI for automatic client reroute by using a DB2ClientRerouteServerList instance to identify the primary server and alternate server.

You must set the following JDBC driver parameters:

- **serverName:** IP address/hostname (if DNS resolution works) of the master node (or load balancer if master is not exposed outside the cluster)
- **portNumber:** Primary HADR `db2u-engn-svc` service `NodePort`
- **clientRerouteAlternateServerName:** IP address/hostname (if DNS resolution works) of the master node (or load balancer if Master is not exposed outside the cluster)
- **clientRerouteAlternatePortNumber:** Standby HADR `db2u-engn-svc` service NodePort

- **enableSeamlessFailover:** true
- **maxRetriesForClientReroute:** 30
- **retryIntervalForClientReroute:** 2

For more details, see the following topics:

- Configuration of Db2 on Linux, UNIX, and Windows systems automatic client reroute support for Java clients
- FAQ: How to configure the JDBC driver for automatic client re-route

Configuring TLS client connections with Db2

Use transport layer security (TLS) to create secure connections from Db2 clients to the integrated Db2 database server deployed on Red Hat OpenShift.

About this task

A Db2 deployment on Red Hat OpenShift contains self-signed TLS support for connections to the Db2 database. This task outlines how to extract the client certificate and enable TLS support for any Db2 client or application that uses IBM Data Server Drivers.

For a detailed description of TLS and how it works in the context of a Db2 client connection, see TLS configuration of Db2.

Procedure

1. SSH into the master node (`master-1`) on your Red Hat OpenShift cluster:

   ```
   ssh account@MASTER-1-IP
   ```

   Replace *account* with the user account that has permission to run **`kubectl`** commands.
2. Get the list of active Db2 pods from the master node.

   ```
   kubectl get pods –all-namespaces | grep -E “servicename-db2u-[0-9]+”
   ```

   Replace *servicename* with the name of your Db2 service. For example, if your service name was *mpp2*:

   ```
   kubectl get pods --all-namespaces | grep -E "mpp2-db2u-[0-9]+"
   ```

   A sample output from the command:

   ```
   zen            mpp2-db2u-0                 1/1     Running           0          20h
   zen            mpp2-db2u-1                 0/1     Running           1          20h
   zen            mpp2-db2u-2                 0/1     Running           1          20h
   ```
3. After you have the name of the Db2 pod, you can copy the self-signed TLS certificate chain. As the TLS certificates are saved in your persistent volume, this procedure can be run from any running Db2 pod.

   On OpenShift

   ```
   oc -n project cp servicename-db2u-0:/mnt/blumeta0/db2/ssl_keystore/rootCA.pem .
   ```

   On Kubernetes-based cluster:

   ```
   kubectl -n namespace cp servicename-db2u-0:/mnt/blumeta0/db2/ssl_keystore/rootCA.pem .
   ```

   Replace *project* or *namespace* with the project or namespace where your Db2 database is deployed. Replace *servicename* with the name of your Db2 service.
4. Copy the Db2 TLS certificate chain over to the system that contains your Db2 client application. The procedure to install the TLS certificate depends on the method that the application uses to connect to the Db2 database.

a) For non-Java clients such as CLI/CLP, ODBC, and .Net, see Configuring TLS support in non-Java Db2 clients

b) For Java applications that use JDBC or JCC connections, see Configuring the Java Runtime Environment to use TLS

5. You need to find the TLS NodePort on your cluster that is used by the Db2 database.

   OpenShift

   ```
   oc -n [project] get svc | grep db2u-engn-svc
   ```

   On Kubernetes-based cluster:

   ```
   oc -n [namespace] get svc | grep db2u-engn-svc
   ```

   Replace *project* or *namespace* with the project or namespace where your Db2 database is deployed. Consider the following example output:

   ```
   mpp2-db2u-engn-svc    NodePort    10.0.86.99     <none>        50000:32209/
   TCP,50001:31050/TCP      20h
   ```

6. Configure your database client application to use that NodePort value when it connects to the database with the installed TLS certificate.

   Using the previous example, you would configure your client application to use `10.0.86.99` as the IP address and port `31050` to connect to the Db2 database server that is running on the Red Hat OpenShift cluster.

Db2 database applications

You can build applications that access and use the data in your Db2 service on Red Hat OpenShift.

Db2 REST service

You can set up your Db2 service so that application programmers can create Representational State Transfer (REST) endpoints that can be used to interact with Db2.

Each endpoint is associated with a single SQL statement. Authenticated users of web, mobile, or cloud applications can use these REST endpoints from any REST HTTP client without having to install any Db2 drivers.

The Db2 REST server accepts an HTTP request, processes the request body, and returns results in JavaScript Object Notation (JSON).

Enabling the Db2 REST interface

You can enable the Db2 REST interface for an existing Db2 deployment on Red Hat OpenShift.

Procedure

1. During or after your provisioning of the Db2 service instance, you can edit the `db2ucluster` custom resource. Find your respective `db2ucluster` resource by running the following command:

   ```
   [root@deploysquad2-inf ~]# oc get db2ucluster
   ```

   The command results will match your deployment ID, in the example below `db2oltp-1603819662989`:

   ```
   NAME                  STATE   AGE
   db2oltp-1603819662989   Ready   34m
   ```

2. Run the following command to edit the add-ons file:

   ```
   oc edit db2ucluster deployment_ID
   ```

Set the **rest** value to `enabled: true`. The edited file should look like the following example:

```
addOns:
  rest:
    enabled: true
```

3. Run the following command to view the REST service and deployment information:

```
oc get formations.db2u.databases.ibm.com db2oltp-1603819662989 -o
go-template='{{range .status.components}}{{printf "%s,%s,%s\n" .kind .name .status.state}}
{{end}}' | column -s, -t
```

The results should look similar to the following example:

```
service               c-db2oltp-1603819662989-db2u-rest-svc          Creating
Deployment            c-db2oltp-1603819662989-rest                   Creating
```

4. Run the following command to view the REST pod as part of your deployment:

```
oc get pods |grep db2
```

Here is an example with the REST pod highlighted:

```
c-db2oltp-1603819662989-db2u-0                                  1/1     Running     0
39m
c-db2oltp-1603819662989-etcd-0                                  1/1     Running     0
39m
c-db2oltp-1603819662989-instdb-df7hh                            0/1     Completed   0
40m
c-db2oltp-1603819662989-ldap-64bc6d58dc-ms29x                   1/1     Running     0
40m
c-db2oltp-1603819662989-rest-66f7fcb7d4-98fbh                   1/1     Running     0
86s
c-db2oltp-1603819662989-restore-morph-bkfpr                     0/1     Completed   0
39m
c-db2oltp-1603819662989-tools-57bc7d4ddf-wz8fr                  1/1     Running     0
40m
db2u-operator-5999cd944b-dwgrr                                 1/1     Running     0          17m
```

Note the REST pod name, which is used to run documented REST commands.

What to do next

After enabling the Db2 REST interface, you can authenticate your TLS certification for Db2 REST.

Enabling TLS certificate based authentication for the Db2 REST service

To use TLS certificate based authentication in a Db2 instance in OpenShift, you must create a secret to provide your certificate to the Db2 REST service. Follow these steps:

Procedure

1. Encode your certificate in base64, replacing myCert.pem with the fully qualified path to your certificate:

```
REST_CERT=$(base64 myCert.pem | tr -d '[:space:]')
```

2. Create a secret:

```
cat << EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: db2rest-internal-tls
  namespace: zen
type: Opaque
data:
  ca.crt: $(echo -n $REST_CERT)
EOF
```

3. Add secret to deployment:
   Run the following command and note the REST deployment name:

   ```
   oc get deployment | grep rest
   ```

   Next, run:

   ```
   oc edit deployment <REST_DEPLOYMENT_NAME>
   ```

   In the volumeMounts section add:

   ```
   - mountPath: /secrets/db2ssl
     name: db2ssl
     readOnly: true
   ```

   In the volumes section, add:

   ```
   - name: db2ssl
     secret:
       defaultMode: 420
       optional: false
       secretName: db2rest-internal-tls
   ```

   Save and exit editing the REST deployment. The existing REST pod will be terminated and a new one is created with the certificate shared with the pod.

Creating the route for the Db2 REST service

After you enable the Db2 REST service, you need to create a route to expose the REST server to clients that reside outside of the cluster.

Before you begin

Before you create the route, you need the name of the Db2 REST service. Run the following command to find the service name:

```
oc get svc |grep db2u-rest-svc
```

The result should look similar to the following example:

```
c-db2oltp-1621977491245678-db2u-rest-svc   NodePort    172.30.84.121   <none>
50050:31812/TCP   27m
```

In the example, the Db2 REST service name is `c-db2oltp-1621977491245678-db2u-rest-svc`.

Procedure

1. Run the following command to create the route:

   ```
   oc create route passthrough rest --service rest_service_name
   ```

   Using the above example, *rest_service_name* would be `c-db2oltp-1621977491245678-db2u-rest-svc`.
2. To view the route that you created, run the following command:

   ```
   oc get routes
   ```

   The result should look similar to the following example:

   ```
   NAME        HOST/PORT                                          PATH
   SERVICES                                  PORT                 TERMINATION
   WILDCARD
   cpd         cpd-zen.apps.db2restgraph.cp.fyre.ibm.com                ibm-nginx-
   svc                              ibm-nginx-https-port   passthrough/Redirect   None
   ```

```
rest        rest-zen.apps.db2restgraph.cp.fyre.ibm.com              c-
db2oltp-1621977491245678-db2u-rest-svc    rest-server            passthrough            None
```

Example

The following example shows the URL for a Db2® REST server: https://rest-zen.apps.db2restgraph.cp.fyre.ibm.com/docs.

What to do next

After you create the route, you can use the REST interface with Db2 on Red Hat OpenShift.

Using the REST interface with Db2 on Red Hat OpenShift

Using the Db2 REST interface is extensively documented in the Db2 documentation. However, some differences exist in usage methods between Db2 on Red Hat OpenShift (RHOS) and the standalone Db2.

Enablement
The biggest difference is in enablement. To enable the REST interface on Red Hat OpenShift, follow the steps in "Enabling the Db2 REST interface" on page 123.

Db2 REST documentation that can be used as is
The following topics apply as is to the REST interface on Red Hat OpenShift:

- Creating applications that use REST services
- Listing jobs
- Describing a REST SQL service
- Listing REST services
- Granting and revoking permissions for a REST service
- Updating or deleting a REST service

Db2 REST documentation that requires some changes for Red Hat OpenShift
The following topics apply to the REST interface on Red Hat OpenShift, but with some differences:

- Defining a REST SQL service: The sentence, "Before you define a service, you must first set up the REST server metadata, either by activating and initialing REST service capability or by issuing the following REST call" does not apply to the REST interface on Red Hat OpenShift because the service is enabled differently. You must issue the REST call that is described in "Enabling the Db2 REST interface" on page 123.
- Operating a REST server: Replace all commands that begin with **`docker exec containername`** with the Red Hat OpenShift command **`oc exec podname`**.
- Authenticating REST commands: If you are authenticating against the Red Hat OpenShift database instance, you do not need to specify valid values for *dbHost*, *dbName*, and *dbPort* because they are pre-specified in the Db2 REST implementation on Red Hat OpenShift. However, you must pass some values when authenticating or the request is rejected. For *dbHost* and *dbName*, if you pass empty strings the pre-specified values are used. For *dbPort*, if you pass -1 the pre-specified values are used. The following example would be valid:

  ```
  json = {
  "dbParms": {
  "dbHost": "",
  "dbName": "",
  "dbPort": -1,
  "isSSLConnection": true,
  "username": username,
  "password": password
  },
  "expiryTime": "24h"
  }
  ```

  If you wish to authenticate with a different database instance, you can specify the parameters for that database instance as necessary.

Db2 Graph

Utilizing the Apache Tinkerpop graph analytics framework, Db2 Graph transforms and optimizes gremlin queries for analyzing data in your Db2 database. Using a Graph overlay file that defines each row in a table as either a vertex or an edge, Db2 Graph is able to expose your Db2 data to graph queries natively and without third party software.

Important: Db2 Graph is not FIPS 140-2 compliant. In Db2 versions 11.5.7 and earlier, Db2 Graph does not function in a FIPS 140-2 environment. If Db2 Graph is deployed in a FIPS 140-2 environment in versions 11.5.8 or later, it will operate in a mode that disables FIPS enforcement for the execution of Db2 Graph processes.

To meet the challenge of analyzing rapidly growing graph and network data created by modern applications many different graph database implementations are emerging. They mainly target low-latency graph queries, such as finding the neighbors of a vertex with certain properties and retrieving the shortest path between two vertices.

Although many of the graph databases handle graph-only queries well, they fall short for real life applications involving graph analysis. Graph queries are not all that one does in an analytics workload. They are often only a part of an integrated heterogeneous analytics pipeline, which can include SQL, machine learning, graph, and other analytics. Graph queries need to be synergistic with other analytics.

Unfortunately, most existing graph databases are stand-alone and cannot easily integrate with other analytics systems:

- Customers need to create and maintain data transformation, export, and loading jobs
- The time to export and load data is time that could be spent analyzing the data to gain insights
- Access control and auditing become problematic when there are two copies of the data
- Custom views of graphs require complex logic to create and maintain, as underlying data changes

Db2 Graph solves these challenges by letting you run Gremlin queries on your existing relational data structure to perform graph analytics without requiring any changes to the underlying database structure.

How Db2 Graph works

Db2 Graph is written with the Apache Tinkerpop graph analytics framework. It can transform and optimize Gremlin queries into SQL statements, which get efficiently processed in Db2 over a JDBC connection. It works by creating a virtual graph model that defines each row in a table as either a vertex or an edge.

This means that data already stored in Db2 can be exposed for graph queries without:

- exporting the data
- transforming the data
- loading the data into a separate graph analytics application

Because the graph queries are running on the data stored in Db2, any new or updated data is made available for graph queries immediately allowing for graph analytics to be performed on transactional data in real time. The vertex and edge definitions are not limited to tables. Custom views in Db2 can be used to define vertexes or edges as well. Allowing for instantaneous customization of a graph with no need to maintain complex logic to create or modify edges to get different graph views.

There are also a number of security and audit processes that become much easier with Db2 Graph. Since the data is stored in Db2, and does not need to be exported or transformed, existing practices for data security and auditing can remain in place. Db2 Graph is viewed as another client accessing the data.

Db2 Graph is preconfigured with the Gremlin console and Gremlin server.

Note: For full details about using Db2 Graph service, see IBM Db2 Graph.

Running IBM Db2 Graph on a Red Hat OpenShift cluster

You can run IBM Db2 Graph as a service within a Red Hat OpenShift cluster. To set up your Db2 Graph service, you must make changes to the `db2ucluster` custom resource, either during or after you provision the Db2 service instance.

Procedure

1. Find your respective **`db2ucluster`** resource by running the following command:

   ```
   oc get db2ucluster
   ```

   The command results match your deployment ID, in the example db2wh-1603819662989:

   ```
   NAME                  STATE   AGE
   db2wh-1603819662989   Ready   34m
   ```

2. Run the following command to edit the **`db2ucluster`** custom resource:

   ```
   oc edit db2ucluster deployment_ID
   ```

   Set the graph value to `enabled: true`. The edited resource should look like the following example:

   ```
   addOns:
     graph:
       enabled: true
   ```

   Add a storage class for graph. Under the **storage:** heading, you can define the graph storage class, allocating the store size and `storageClassName` as applicable for your environment. In this example, 5 GB of manage-nfs-storage are allocated to IBM Db2 Graph:

   ```
    - name: graph
      spec:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 5Gi
        storageClassName: managed-nfs-storage
      type: create
   ```

3. Run the following command to view the Graph service and deployment information:

   ```
   oc get formations.db2u.databases.ibm.com db2wh-1603819662989 -o
   go-template='{{range .status.components}}{{printf "%s,%s,%s\n" .kind .name .status.state}}
   {{end}}' | column -s, -t
   ```

   The results should look similar to the following example:

   ```
   service               c-db2wh-1603819662989-db2u-graph-svc         Creating
   Deployment            c-db2wh-1603819662989-graph                  Creating
   ```

4. Run the following command to view the Graph pod as part of your deployment:

   ```
   oc get pods |grep db2
   ```

   Here is an example with the Graph pod highlighted:

   ```
   c-db2wh-1603819662989-db2u-0                                  1/1     Running     0          50m
   c-db2wh-1603819662989-etcd-0                                  1/1     Running     0          50m
   c-db2wh-1603819662989-graph-5d4b8b694c-nd4jw                  1/1     Running     0          56s
   c-db2wh-1603819662989-instdb-df7hh                            0/1     Completed   0          51m
   c-db2wh-1603819662989-ldap-64bc6d58dc-ms29x                   1/1     Running     0          51m
   c-db2wh-1603819662989-rest-66f7fcb7d4-98fbh                   1/1     Running     0          12m
   c-db2wh-1603819662989-restore-morph-bkfpr                     0/1     Completed   0          50m
   c-db2wh-1603819662989-tools-57bc7d4ddf-wz8fr                  1/1     Running     0          51m
   ```

Take note of the Graph pod name as it will be referred to as *GRAPH_POD_NAME* and used to execute commands in the remainder of the documentation.

5. After your initial deployment, query the logs and take note of the password presented for the **graph_admin** and **graph_user** accounts. You will need the password in order to log into the IBM Db2 Graph user interface or to programmatically access IBM Db2 Graph.

   Run the following command to view the Graph pod deployment logs:

   ```
   oc logs <GRAPH_POD_NAME>
   ```

   ```
   ################################################################################################
   ###  Db2 Graph passwords for users graph_admin and graph_user have been          ###
   ###  initialized to: 8VmO@b8nCLFTGruQPOb1lS7c6velm                               ###
   ###  To change the initial passwords, use the manage user command.              ###
   ################################################################################################

   Starting the Db2 Graph server...
   Db2GRAPH-1000I The Db2 Graph server is running.
   ################################################################################################
   ###  IBM Db2 Graph was started successfully.                                     ###
   ################################################################################################
   * To return to the host after viewing and accepting the license, press
     ww
     Ctrl+p followed by Ctrl+q.

   * To manage the Db2 Graph server, issue the following command for more information:
     [ docker | oc | kubectl -n <NS> ] exec -it <graph container name> manage
   ################################################################################################
   ```

   To change the randomly generated default password for either **graph_admin** or **graph_user**, execute the following command:

   ```
   oc exec -it GRAPH_POD_NAME manage user
   ```

   Select the **graph_admin** or **graph_user** account and enter a password. If you want to change both passwords, repeat the process for the other user. When you have completed changing passwords, run the following command to restart IBM Db2 Graph:

   ```
   oc exec -it GRAPH_POD_NAME manage restart
   ```

What to do next

After setting up Db2 Graph for your Red Hat Open Shift cluster, you can .

Enabling TLS certificate based authentication for Db2 Graph

To use TLS certificate based authentication in a Db2 instance in OpenShift, you must create a secret to provide your certificate to the Db2 Graph service. Follow these steps:

Procedure

1. Encode your certificate in base64, replacing myCert.pem with the fully qualified path to your certificate:

   ```
   GRAPH_CERT=$(base64 myCert.pem | tr -d '[:space:]')
   ```

2. Create a secret:

   ```
   cat << EOF | oc apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: db2graph-internal-tls
     namespace: zen
   type: Opaque
   data:
   ```

```
  ca.crt: $(echo -n $GRAPH_CERT)
EOF
```

3. Add secret to deployment:
   Run the following command and note the graph deployment name:

   ```
   oc get deployment | grep graph
   ```

   Next, run:

   ```
   oc edit deployment <GRAPH_DEPLOYMENT_NAME>
   ```

   In the volumeMounts section add:

   ```
   - mountPath: /secrets/db2ssl
     name: db2ssl
     readOnly: true
   ```

   In the volumes section, add:

   ```
   - name: db2ssl
     secret:
       defaultMode: 420
       optional: false
       secretName: db2graph-internal-tls
   ```

   Save and exit editing the Graph deployment. The existing Graph pod will be terminated and a new one is created with the certificate shared with the pod.

Creating a custom route to IBM Db2 Graph

After enabling Db2 Graph you can create and customize the route by which users access the Db2 Graph User Interface (UI) and Server.

Before creating the route we need the name of the Db2 Graph service. Run the following command to find the service name:

```
oc get svc | grep db2u-graph-svc
```

The result should look similar to the following example:

```
c-db2wh-1621022757650703-db2u-graph-svc                NodePort    172.30.237.60
<none>        8182:30403/TCP,3000:30412/TCP               62s
```

In the example above the Db2 Graph service name is:

```
c-db2wh-1621022757650703-db2u-graph-svc
```

Note: The Db2 Graph service name is referenced as GRAPH_SERVICE_NAME in the topics Creating a route for the Db2 Graph User Interface and Creating a route for the Db2 Graph Server.

Creating a route for the IBM Db2 Graph User Interface

For the Db2 Graph user interface (UI) to be accessed from clients external to the cluster you need to create a route to expose this service.

Procedure

1. Execute the following command to create the route:

   ```
   oc create route passthrough graph --service GRAPH_SERVICE_NAME --port graph-ui
   ```

2. Query the routes to see the route you have created. Execute the following command:

   ```
   oc get routes
   ```

The result should look similar to the following example:

```
NAME    HOST/PORT                                          PATH
SERVICES                                  PORT                  TERMINATION
WILDCARD
cpd     cpd-zen.apps.deploysquad5.cp.fyre.ibm.com          ibm-nginx-
svc                              ibm-nginx-https-port   passthrough/Redirect   None
graph   graph-zen.apps.deploysquad5.cp.fyre.ibm.com         c-db2wh-1621022757650703-db2u-
graph-svc   graph-ui              passthrough            None
```

The following example shows the URL for a Db2 Graph UI:

```
https://graph-zen.apps.deploysquad5.cp.fyre.ibm.com
```

Creating a route for the IBM Db2 Graph server

To make use of Db2 Graph programmatically from clients that reside outside of the cluster, you need to create a route to expose the service.

Procedure

1. Execute the following command to create the route:

```
oc create route passthrough graph-wss --service GRAPH_SERVICE_NAME --port graph-server
```

2. Query the routes to see the route you have created. Execute the following command:

```
oc get routes
```

The result should look similar to the following example:

```
NAME    HOST/PORT                                                PATH
SERVICES                            PORT                     TERMINATION
WILDCARD
cpd     cpd-zen.apps.deploysquad5.cp.fyre.ibm.com            ibm-nginx-
svc                              ibm-nginx-https-port      passthrough/Redirect        None
graph   graph-wss-zen.apps.deploysquad5.cp.fyre.ibm.com      c-db2wh-1621022757650703-db2u-
graph-svc   graph-server              passthrough                None
```

Using the above example to programmatically access the Db2® Graph server:

- Use `graph-wss-zen.apps.deploysquad5.cp.fyre.ibm.com` as the *graphServerHostname*.
- Use 443 as the *graphServerPort*.

Db2 security on Red Hat OpenShift

Db2 on Red Hat OpenShift supports Transport Layer Security (TLS) to encrypt data in transit.

In addition, client-server communications are fully encrypted at both the network and disk level.

Db2 SCC capabilities

When you install the Db2 service, custom security context constraints (SCC) are automatically created for the service.

You can also manually create an SCC, service account, role, and role binding for the Db2 service.

The default, automatically created Db2 SCCs have the following capabilities:

SYS_RESOURCE
Allows manipulation of reservations, memory allocations, and resource limits. Maximum memory allocation is still constrained by the memory cgroup (memcg) limit, which cannot be overridden by this sys-capability. The Db2 database engine needs this sys-capability to increase the resource limits (IE.ulimits).

IPC_OWNER
Bypasses permission checks for operations on IPC objects. Even when the IPC kernel parameters are set to maximum values on the hosts/worker nodes, the Db2 engine still tries to dynamically throttle those values. This system capability is provided in addition to sharing IPC namespace with the host.

SYS_NICE
Allows changing process priorities. Because each container has its own PID namespace, this capability applies to that container only. The Db2 database engine relies on process thread prioritization to ensure that Work Load Management (WLM) and Fast Communications Manager (FCM) processing is prioritized over generic agent work.

CHOWN
Necessary to run **chown** to change ownership of files/directories in persistent volumes.

DAC_OVERRIDE
Bypasses permission checks for file read, write, and execute.

FSETID
Prevents the clearing of the setuid and setgid mode bits when a file is modified.

FOWNER
Bypasses permission checks on operations that normally require the file system UID of the process to match the UID of the file (for example, chmod(2), utime(2)), excluding those operations that are covered by CAP_DAC_OVERRIDE and CAP_DAC_READ_SEARCH.

SETGID
Necessary to run Db2 engine processes with escalated group privileges.

SETUID
Necessary to run Db2 engine processes with escalated user privileges.

SETFCAP
Used to set capabilities on files.

SETPCAP
Used to set capabilities on processes.

SYS_CHROOT
Necessary to use the **chroot** command.

KILL
Bypasses permission checks for sending signals. Necessary for signal handling during process management.

AUDIT_WRITE
Required to write records to the kernel auditing log when SELinux is enabled.

Manually creating an SCC, service account, role, and role binding

If you do not want Db2 to automatically create custom Security Context Constraints (SCC) when you deploy the service on Red Hat OpenShift, you can manually create the SCC, service account, role, and role binding within Red Hat OpenShift.

Before you begin

Define the following parameters in OpenShift:

- SCC_NAME
- PROJECT
- SERVICE_ACCOUNT
- ROLE_NAME
- ROLEBINDING_NAME

When you run the commands below, the values that you assign to these parameters will automatically be used in the commands in place of `${SCC_NAME}`, `${PROJECT}`, `${SERVICE_ACCOUNT}`, `${ROLE_NAME}`, and `${ROLEBINDING_NAME}`.

For more information, see Managing security context constraints and Understanding and creating service accounts in the OpenShift documentation.

About this task

Creating the SCC and service account prior to deploying Db2 gives you full control over the security specifications. The following procedure shows you how to create the required objects in two ways:

- Privileged account without enabling unsafe sysctls
- Limited privileged account with unsafe sysctls enabled

Procedure

Run one of the following commands to create either a privileged account or unprivileged account:

- **Privileged account without enabling unsafe sysctls**

```
oc apply -n ${PROJECT} -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ${SERVICE_ACCOUNT}
  namespace: ${PROJECT}
EOF

oc apply -n ${PROJECT} -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ${ROLE_NAME}
  namespace: ${PROJECT}
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  - pods
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - apps
  resources:
  - statefulsets
  - deployments
  - replicasets
  verbs:
  - get
  - list
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - patch
  - watch
  - list
  - update
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - create
  - update
- apiGroups:
  - db2u.databases.ibm.com
  resources:
```

```
  - recipes
  verbs:
  - watch
  - get
  - update
  - create
  - patch
  - list
  - delete
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - buckets
  verbs:
  - patch
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - backups
  verbs:
  - patch
  - delete
  - list
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - formations
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - pods/exec
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - watch
  - list
  - get
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - watch
  - list
  - get
EOF

oc apply -n ${PROJECT} -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ${ROLEBINDING_NAME}
  namespace: ${PROJECT}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ${ROLE_NAME}
subjects:
- kind: ServiceAccount
  name: ${SERVICE_ACCOUNT}
  namespace: ${PROJECT}
EOF

oc apply -n ${PROJECT} -f - <<EOF
allowHostDirVolumePlugin: true
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: true
allowedCapabilities: []
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  ranges:
```

```
  - max: 1000
    min: 1000
  type: MustRunAs
groups: []
kind: SecurityContextConstraints
metadata:
  name: ${SCC_NAME}
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- SETUID
- SETGID
- MKNOD
- ALL
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:${PROJECT}:${SERVICE_ACCOUNT}
volumes:
- configMap
- downwardAPI
- emptyDir
- hostPath
- persistentVolumeClaim
- projected
- secret
EOF
```

- **Limited privileged account with unsafe sysctls enabled**

```
oc apply -n ${PROJECT} -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ${SERVICE_ACCOUNT}
  namespace: ${PROJECT}
EOF

oc apply -n ${PROJECT} -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ${ROLE_NAME}
  namespace: zen
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  - pods
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - apps
  resources:
  - statefulsets
  - deployments
  - replicasets
  verbs:
  - get
  - list
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - patch
  - watch
  - list
  - update
- apiGroups:
  - ""
  resources:
```

```
  - secrets
  verbs:
  - get
  - create
  - update
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - recipes
  verbs:
  - watch
  - get
  - update
  - create
  - patch
  - list
  - delete
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - buckets
  verbs:
  - patch
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - backups
  verbs:
  - patch
  - delete
  - list
- apiGroups:
  - db2u.databases.ibm.com
  resources:
  - formations
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - pods/exec
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - watch
  - list
  - get
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - watch
  - list
  - get
EOF

oc apply -n ${PROJECT} -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ${ROLEBINDING_NAME}
  namespace: ${PROJECT}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ${ROLE_NAME}
subjects:
- kind: ServiceAccount
  name: ${SERVICE_ACCOUNT}
  namespace: ${PROJECT}
EOF


oc apply -n ${PROJECT} -f - <<EOF
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
```

```
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: []
allowedUnsafeSysctls:
- kernel.shmmni
- kernel.shmmax
- kernel.shmall
- kernel.sem
- kernel.msgmni
- kernel.msgmax
- kernel.msgmnb
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  ranges:
  - max: 1000
    min: 1000
  type: MustRunAs
groups: []
kind: SecurityContextConstraints
metadata:
  name: zen-c-db2u-oltp-restr-scc
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- SETUID
- SETGID
- MKNOD
- ALL
runAsUser:
  type: MustRunAs
  uid: 500
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:${PROJECT}:${SERVICE_ACCOUNT}
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
EOF
```

What to do next

Deploy Db2 with the custom service account, SCC, role, and role binding that you created.

Roles and permissions required for Db2 on Red Hat OpenShift

To install and use the Db2 service on Red Hat OpenShift, you must have certain roles and permissions on the Red Hat OpenShift platform.

The following roles and permissions are needed:

Install the Db2 operator
To install the Db2 operator, which is required to install Db2, you need the Red Hat OpenShift cluster administrator role.

Create a Db2 instance
To create a Db2 instance, you need the OpenShift Project Administrator role.

Use Db2 databases
To use Db2 databases, you need different roles depending on the task. Table 8 on page 138 shows the role descriptions and names and the permissions that they include. To learn more about the authorities for database user and database administrator, see GRANT (database authorities) statement and Authorities overview.

Table 8. Required roles for database operations

Role	Name	Permission
Database user	User	CONNECT, CREATETAB, LOAD, BINDADD, IMPLICIT_SCHEMA
Database administrator	Admin	SECADM, DBADM WITH DATAACCESS, CREATE_EXTERNAL_ROUTINE
Custom definition	UserDefined	None by default [1]

1. The UserDefined role grants no authorities to the user be default. Database administrators can perform Db2 GRANT statements to give users who have this role the required authorities.

Role-binding access control

The db2u ServiceAccount and associated db2u-role Role are necessary for pod-to-pod control and communication for a successful deployment. The resources and verbs are outlined in the following example:

```
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log", "pods/exec"]
  verbs: ["get", "list", "patch", "watch", "update", "create"]

- apiGroups: [""]
  resources: ["services"]
  verbs: ["get", "list"]

- apiGroups: ["batch", "extensions"]
  resources: ["jobs", "deployments"]
  verbs: ["get", "list", "watch", "patch"]
```

Hostpath requirements

The `/proc` volume must be mounted into an init container to either set or validate the required IPC kernel parameters for Db2. Hostpath volumes are also supported for single-node Db2 deployments.

Uninstalling Db2 for Red Hat OpenShift

A project administrator can uninstall Db2 from a Red Hat OpenShift cluster.

Before you begin

To complete this task, you must be an administrator of the project (namespace) where Db2 is installed.

Before you uninstall , ensure that the machine from which you will run the commands meets the following requirements:

- Can connect to the cluster.
- Has the Red Hat OpenShift command line interface (oc)

Removing Db2 will remove all of the data that is associated with the instance.

Procedure

From your installation node:

1. Log in to your Red Hat OpenShift cluster as a project administrator:

   ```
   oc login OpenShift_URL:port
   ```

2. Retrieve all objects of the type `db2cluster` and then delete them:

```
oc get db2ucluster
oc delete db2ucluster [clustername]
```

Where *clustername* is the value given to the named `db2cluster` object.

The Db2 instance has now been removed from the Red Hat OpenShift cluster.

Configuring ID mapping on NFS 4

You can create a daemon set to configure ID mapping for Db2 on NFS 4.

About this task

These steps also enable `no_root_squash` in the IBM Cloud environment. For more details, see Implementing no_root_squash for NFS in the IBM Cloud documentation.

Procedure

You can use a daemonset or commands to configure ID mapping:

- **To use a daemonset**

  a. Create a service account called `norootsquash` by running the following command:

  ```
  oc create -f - << EOF
  apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: norootsquash
    namespace: kube-system
  EOF
  ```

  b. Give the service account privileged security context constraints (SCC) by running the following command:

  ```
  oc adm policy add-scc-to-user privileged system:serviceaccount:kube-system:norootsquash
  ```

  c. Create the daemon set by running the following command:

  ```
  export DOMAIN_NAME=<>

  oc create -f - << EOF
  apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: norootsquash
    namespace: kube-system
    labels:
      tier: management
      app: norootsquash
  spec:
    selector:
      matchLabels:
        name: norootsquash
    template:
      metadata:
        labels:
          name: norootsquash
      spec:
        serviceAccountName: norootsquash
        initContainers:
          - resources:
              requests:
                cpu: 0.1
            securityContext:
              privileged: true
            image: alpine:3.6
            name: unrootsquash
            command: ["chroot", "/host", "sh", "-c"]
            args:
              - >
                grep "^Domain = ${DOMAIN_NAME}" /etc/idmapd.conf;
  ```

```
              if [ "\$?" -ne "0" ] ; then
                sed -i 's/.*Domain =.*/Domain = ${DOMAIN_NAME}/g' /etc/idmapd.conf;
                nfsidmap -c;
                rpc.idmapd
              fi;
          volumeMounts:
            - name: host
              mountPath: /host
      containers:
        - resources:
            requests:
              cpu: 0.1
          image: alpine:3.6
          name: sleep
          command: ["/bin/sh", "-c"]
          args:
            - >
              while true; do
                sleep 100000;
              done
      volumes:
        - hostPath:
            path: /
            type: Directory
          name: host
EOF
```

If you want the daemonset pods to run on a set of nodes, you can add the node selector to the YAML. For more information, refer to Assigning Pods to Nodes

If your nodes are tainted (for dedicated Db2 Warehouse deployments, we add taints to the node), you can add the corresponding tolerations for those taints. For more information, refer to Taints and Tolerations

- **To use commands**

  Run the following command to perform the same task as the daemonset. The command takes about 30 seconds per node. Note that these settings do not apply to new worker nodes, so you must add them.

```
oc get no -l node-role.kubernetes.io/worker --no-headers -o name | xargs -I {} --  oc
debug {} -- chroot /host sh -c 'grep "^Domain = ${DOMAIN_NAME}" /etc/idmapd.conf || ( sed
-i "s/.*Domain =.*/Domain = slnfsv4.com/g" /etc/idmapd.conf; nfsidmap -c; rpc.idmapd )'
```

Note: The `DOMAIN_NAME` for `ibm-cloud-file-storage` would be `slnfsv4.com`