

# Working with Databases

## Objectives

- Build containerized databases.

## Stateful Database Containers

Database containers typically maintain state because they must persist data. For this reason, they are called *stateful containers*. These containers differ from other types of containerized software, such as web APIs or proxies, which do not need to maintain state and therefore are called stateless containers.

### NOTE

Database containers can be stateless if there is no need to persist data. Memory-based caches or ephemeral testing databases are examples of stateless database containers.

Compared to stateless containers, stateful containers present the following challenges:

### Portability

The ability to move the container between different environments. Creating a container in the new environment is no longer enough and you must provide the container with up-to-date data.

### Scalability

The ability to improve performance by increasing the number of replicas of a container. Many database technologies are not designed to have data operated by more than one database process at a time.

### Availability

The ability to keep the system working in the event of one container crashing as long as that container has another replica running. Even though database systems include configurations for high availability, Podman alone does not provide the features needed for this type of configuration.

To achieve portability with stateful containers, you must provide them with a valid state across different environments. For example, your containers might need sample data for development and testing environments. In production, however, your containers typically require real, up-to-date data.

Scalability and availability are not requirements for development or testing environments. They are addressed in production environments by using features offered by orchestration systems such as Red Hat OpenShift, along with specialized database operators.

# Good Practices for Database Containers

The following practices provide benefits when you containerize database processes:

## Use the **VOLUME** instruction in Containerfiles

When you build a custom database container image, use the **VOLUME** instruction to create mount points at the database storage directories. Mount points avoid using the copy-on-write (COW) file system, which does not perform well with stateful data.

When you create a container from an image that uses the **VOLUME** instruction, Podman creates an anonymous volume and attaches the volume to the container.

## Mount the data directory to a named volume

Use a named volume to mount the data directory of your database to add data persistence across container recreations. Volumes are also more portable than bind mounts because the source directory does not need to exist in the host machine.

## Create a database network

If only containerized applications access your database, then there is no need to expose your database to the host network. You can omit exposing the database port and use a Podman network to access the database.

This provides better isolation for the database, because only applications that share the network have access to the database.

# Import Database Data

Setting up the development environment sometimes requires populating the database with sample data for development purposes. Testing environments might require larger data sets than development environments, or even production-like data.

Depending on the database container that you choose, there might be different approaches to load the database with data.

## Database Containers with Data-loading Features

Database container images usually configure a directory where you can place scripts to initialize the database. You can mount your database scripts into that directory. The database container executes the scripts at container creation or at container start.

You can also migrate the data from a running database. Some database containers include an export feature to extract the data from the container while the database is running.

## Load Data with a Database Client

You can also load the data by using a database client that is compatible with your database server. Provide the client with a file containing the data to load and the configuration to connect to the database server.

The container that runs the database might already include that client. In that case, you must provide the database scripts to the container. You can copy the scripts into the container by using the `podman cp` command.

```
[user@host ~] podman cp SQL_FILE TARGET_DB_CONTAINER:CONTAINER_PATH
```

The preceding command copies *SQL\_FILE*, the file containing the data, from the host to the container.

After you have copied the scripts into the container, run the database client command to load the data. For example, for a PostgreSQL database the following command executes the *SQL\_FILE* to create and populate the database.

```
[user@host ~] podman exec -it DATABASE_CONTAINER \  
    psql -U DATABASE_USER -d DATABASE_NAME \  
        -f CONTAINER_PATH/SQL_FILE
```

If a database image does not include this feature, then you can create a container that includes a database client, and use this container to load the data.

For example, the following command creates an ephemeral container to load data into a PostgreSQL database:

```
[user@host ~] podman run -it --rm \  
    -e PGPASSWORD=DATABASE_PASSWORD \  
    -v ./SQL_FILE:/tmp/SQL_FILE:Z \  
    --network DATABASE_NETWORK \  
    registry.redhat.io/rhel8/postgresql-12:1-113 \  
    psql -U DATABASE_USER -h DATABASE_CONTAINER \  
        -d DATABASE_NAME -f /tmp/SQL_FILE
```

This command uses the SELinux option `z` to set the SELinux context for the container to have access to the host *SQL\_FILE*.

The command also uses the *DATABASE\_NETWORK* network, which allows the `psql` client in this container to use the *DATABASE\_CONTAINER* name as the hostname for the database container. For the DNS to work, the *DATABASE\_NETWORK* network must have DNS enabled.

## Export Database Data

To export the database data, you can use database backup commands present in the database container image. For example, MySQL provides the `mysqldump` command and PostgreSQL provides the `pg_dump` command.

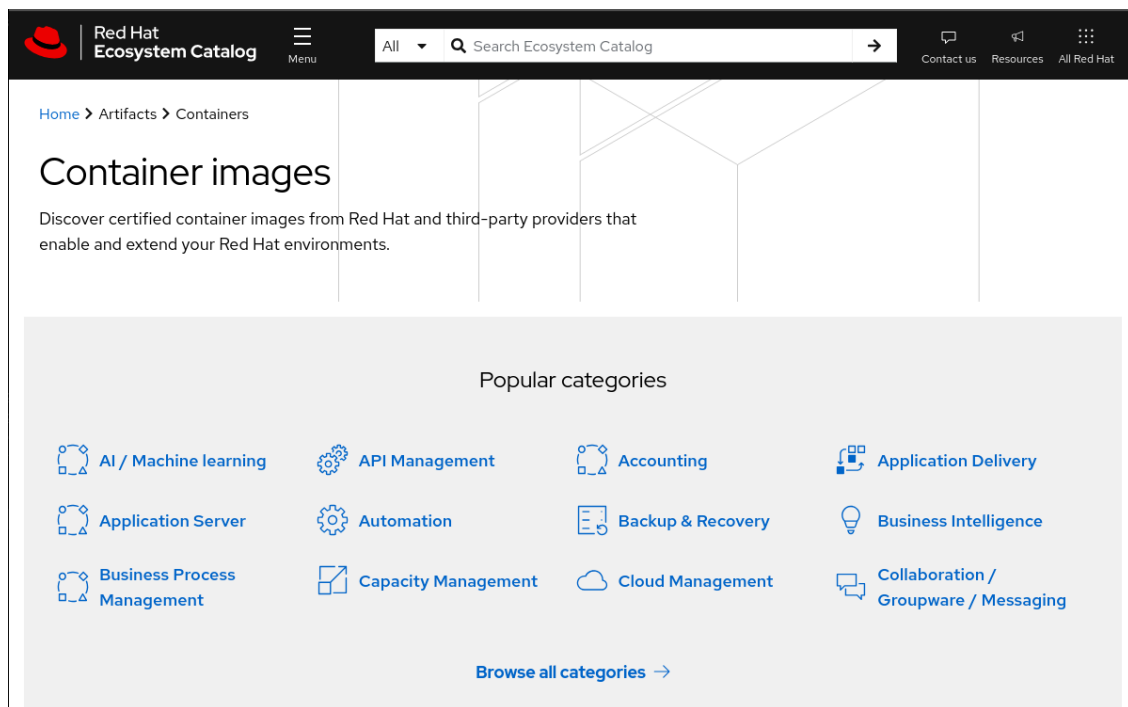
You can run the following command in a PostgreSQL container to export the database called *DATABASE* to a *BACKUP\_DUMP* file.

```
[student@workstation ~]$ podman exec POSTGRESQL_CONTAINER \  
pg_dump -Fc DATABASE -f BACKUP_DUMP  
no output expected
```

Then, you can copy the dump file out of the database container to import the data in a new container.

## Red Hat Database Containers

The registries in the Red Hat Ecosystem Catalog contain a list of database container images supported by Red Hat.



Find the database products containerized by Red Hat and third-party vendors by selecting the Database & Data Management category.

Container results by Red Hat (19 results)

Sort by: Relevance 1 - 19 of 19

Provider: Red Hat Category: Database & Data Management Clear filters

Provider

Category

Search Category

Clear

☐ Data Store

☒ Database & Data Management

☐ Developer Tools

☐ Identity Management

☐ Integration

☐ Logging & Metrics

☐ Management

☐ Messaging

**Red Hat**

**memcached 1.5**

**A** rhel8/memcached

by Red Hat

High-performance memory object caching system

Container image

1.5

amd64

Published 1 hour ago

**Red Hat**

**PostgreSQL 16**

**B** rhel8/postgresql-16

by Red Hat

PostgreSQL is an advanced Object-Relational database management system

Container image

1-42

s390x

Published 4 days ago

**Red Hat**

**Mariadb 10.11**

**B** rhel8/mariadb-1011

by Red Hat

MariaDB 10.11 SQL database server

Container image

latest

s390x

Published 4 days ago

After picking the database container, pull the container image and follow the usage instructions.

For example, if you select the `rhel9/mariadb-105` image, then you can pull the image by navigating to the `Get this image` tab and following the instructions under the `Using podman` login section. The usage instructions are available on the `overview` tab.

Home > Artifacts > Containers > All containers

Builder image

**Mariadb 10.5**

rhel9/mariadb-105

Architecture: amd64 Tag: 9.5-1741891134 Repository structure: Single-stream

Provided by **Red Hat**

9.5-1741891134 1-1741891134 latest 9.5 1 more

Overview Security Technical Information Packages Dockerfile Get this image

**Description**

This container image provides a containerized packaging of the MariaDB mysqld daemon and client application. The mysqld server daemon accepts connections from clients and provides access to content from MySQL databases on behalf of the clients. You can find more information on the MariaDB project

**Published**

4 days ago

**Release category**

## REFERENCES

[Red Hat Ecosystem Catalog](#)

[Red Hat Container Registry Authentication](#)

[Source for the PostgreSQL Container](#)

[Source for the Mariadb Container](#)

[Source for the Mongodb Container](#)

[Source for the Redis Container](#)