IBM Content Manager OnDemand Implementing Docker

March 6, 2019

Rob Russell

Software Engineer - Content Manager OnDemand

Table of Contents

Introduction

Overview

With the rise in popularity of containerization, many companies are now looking at how they can benefit from leveraging this relatively new technology to increase portability and scalability of applications and minimize usage of system resources. The purpose of this document is to provide you with the information that can simplify the process of creating a Content Manager OnDemand Docker image in a Linux environment.

This document will not go into the details of Docker itself. It assumes the reader has the necessary skills required to implement Docker solutions. It is recommended you read the document in its entirety before attempting to create your first Content Manager OnDemand Docker image.

The first step guides you through the process of creating a Content Manager OnDemand Docker image. It covers the basics of creating an image with all the necessary software needed to run a Content Manager OnDemand library server. With the image created, the document will then discuss in greater detail key concepts like connecting to a remote database, storage considerations, and running maintenance using the ARSMAINT command.

Software requirements

In order to create your Content Manager OnDemand Docker image, you will need to download and stage the appropriate software for your desired environment. We will focus on using Db2 as the database. At a minimum you will need the following software:

- Database runtime client
- Content Manager OnDemand version 10.1 or later
- Content Manager OnDemand latest fix pack
- GSKIT (included with Content Manager OnDemand)

Refer to the Content Manager OnDemand Detailed System Requirements page for the most updated information about supported software.

Content Manager OnDemand and Docker

Based on Docker best practices and the needs of Content Manager OnDemand, it is likely that you will need to create multiple Docker images to properly implement your full Content Manager OnDemand Docker strategy. Each image will support a different Content Manager OnDemand process in your environment.

It is considered best practice to create Docker images that serve a single purpose. Each container should have only one main concern. Since Content Manager OnDemand is typically installed on multiple different servers, you may find yourself creating multiple Docker images. The following examples are some of the different images you might create:

- As a part of an IBM Content Navigator container - Customers using IBM Content Navigator as their web client would install Content Manager OnDemand to a server running a compatible application server. In this scenario no Content Manager OnDemand database is required; just the Content Manager OnDemand Java APIs to communicate with one or many Content Manager OnDemand servers.

- As an ARSLOAD server - For high volume systems, it might be beneficial to separate the indexing and load processes into their own environment. In such an environment, the Content Manager OnDemand binaries would be installed, the appropriate staging directories would be created, and daemons would be configured to run upon system start up.

- As a stand-alone object server - While this scenario is not as common as it once was, it may still be desirable to have a one or many stand-alone object servers. This could be for the geographical distribution of data or for horizontal scaling.

- As a stand-alone Content Manager OnDemand Web Enablement (ODWEK) application server – When developing Content Manager OnDemand custom applications based on the Content Manager OnDemand Web Enablement Kit, a base install of Content Manager OnDemand is required.

- As a Content Manager OnDemand library server – This is a typical Content Manager OnDemand server requiring access to a database and able to manage its own storage.

Luckily, each of these scenarios have installing Content Manager OnDemand and its appropriate fix pack in common. This document should provide the necessary information to get you started with all of the Content Manager OnDemand Docker scenarios described above and guide you through the most complex of the scenarios, which is setting up a Content Manager OnDemand server with a remote Db2 database.

General guidelines and recommendations

One of the main goals to keep in mind when creating Docker images is keeping your images as immutable and ephemeral as possible. What this means is that your containers should be able to be stopped, destroyed, then rebuilt and replaced with a minimal setup and configuration. Consider upgrading Content Manager OnDemand for example. In the configuration that follows, we have designed our Dockerfile so that it is only necessary to replace the fix pack in the staging directory and re-run the Docker build command. This will then create an updated image with the latest fix pack applied. You can then simply stop your currently running container and restart with the new image and everything should work as previously defined.

Another key aspect of a successful Docker solution is the management of persistent data. It is necessary to isolate persistent data from a container in order to retain the benefits of containerization. If you were to store data inside a typical Docker container, the data would not persist once the container is no longer running. For example, if you were to store Content Manager OnDemand document data inside your container, once the container is stopped the data would be gone for good. To solve this problem, Docker has a couple options to ensure the data can be persist when a container is stopped: volumes and bind mounts. Refer to Docker documentation for more detailed information on both volumes and bind mounts. We will be using both of these in our example.

Development environment

In our Docker development environment, we are running Red Hat Enterprise Linux Server along with the latest version of Docker installed. Content Manager OnDemand and its latest fix pack have been installed locally as well. All the files located in the <Content Manager OnDemand install dir>/config directory are required, and this is the simplest way to get them.

You will also use your local Content Manager OnDemand install to create a Content Manager OnDemand stash file which will contain your Db2 credentials. You will also make a copy of the Content Manager OnDemand configuration files and edit them according to your needs. These files will all be mounted to your container once it is started.

Creating your first Content Manager OnDemand Docker image

The following steps will guide you through the process of creating your first Content Manager OnDemand Docker image. You will need the following software to create this image:

- Content Manager OnDemand v10.1 plus latest fix pack
- Content Manager OnDemand configuration files
- Latest version of Db2 Runtime
- GSKIT that comes bundled with Content Manager OnDemand
- Scripts to perform database creation and ARSSOCKD startup

Once you have all the software gathered, follow the steps below to create your Docker image build environment.

1. Create a staging area for the software you will be installing into your image. In this example, the root directory is defined as */build*. We then create subdirectories for each of the key software components. For example:

   /build/od - This directory should contain the base Content Manager OnDemand v10.1 installation image as well as the latest current Content Manager OnDemand fix pack. It should also contain the Content Manager OnDemand installation response file. The same response file can be used for both the base install and the fix pack install.

   ```
   [root@geeklab /build/od]# ls -l
   odlinux10104.bin
   odlinux.bin
   response.file
   ```

   /build/db2 - This directory should contain the Db2 installation file as well as the Db2 response file for the Db2 Runtime Client.

   ```
   [root@geeklab /build/db2]# ls
   DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar
   db2rtcl.rsp
   ```

   /build/scripts - This directory should contain any scripts that are needed to create Content Manager OnDemand tables (if necessary) and also the necessary scripts to start the ARSSOCKD process. The scripts in this directory are not invoked during the image creation process but rather when a container is created from the image. They will be discussed in a subsequent section of this document.

   /build/gskit - This directory should contain the GSKIT installation files shipped with Content Manager OnDemand.

```
[root@geeklab /build/gskit]# ls
      gskcrypt64-8.0.50.84.linux.x86_64.rpm
      gskssl64-8.0.50.84.linux.x86_64.rpm
```

/build – The /build directory should contain your Dockerfile, entrypoint.sh script and .od_envlist file.

```
root@geeklab /build]# ls -l
total 12
drwxrwxrwx. 2 root root    69 Nov 28 09:39 db2
-rwxrwxrwx. 1 root cigar 1794 Dec 10 20:09 Dockerfile
-rwxr-xr-x. 1 root root  2231 Dec  4 12:48 Dockerfile.curl
-rwxrwxrwx. 1 root cigar 2543 Dec 10 10:18 entrypoint.sh
drwxrwxrwx. 2 root root    94 Nov 28 09:39 gskit
drwxrwxrwx. 2 root root    70 Nov 28 09:39 od
drwxrwxrwx. 2 root root    77 Dec 10 13:51 scripts
```

Dockerfile – A Dockerfile is a file that contains all the instructions that a user would call from the command line to assemble an image. Think of this as your recipe for the image you are building. When the command *docker build* is run, the commands from within the Dockerfile are executed and an image is created. Dockerfiles are typically not very complex but there are some key points to keep in mind when creating them:

> Each command (entry in your Dockerfile) creates a layer. Every layer gets its own unique hash value. Docker caches each step in the build. This greatly speeds up build time because if you run your build a second time, Docker checks its local cache to see if anything in that line has changed. If it hasn't changed, then the local cache is used and that step can be skipped. If it encounters a line that has changed, that command and all subsequent commands are then run.
>
> The takeaway here is to put all lines in your Dockerfile that you don't think will change first. This will speed up your build process significantly.

entrypoint.sh – An entrypoint script is used to initialize stateful data in a container at runtime. This file is copied to the image during the image creation process and invoked when a container is created with the *docker run* command. This script should execute any necessary setup steps before control is passed to the container's long-running process. In our case that long-running process is ARSSOCKD.

.od_envlist – Like the entrypoint.sh file, this file is not used until a container is started. It may contain some of the following information:

```
DB2INSTANCE=archive
DB2INSTOWNER=archive
DB2INST_PASSWORD=password
DBNAME=archive
DB2PATH=/opt/ibm/db2/V11.1_01/
DB2ADDR=172.17.0.2
STORAGE_DIR=/database
```

This information would be necessary for your Content Manager OnDemand container to communicate with a remote instance of Db2.

2. Create response files for both Content Manager OnDemand and Db2 and place them into the appropriate staging directory. A response file is required since installing software to Docker images is not an interactive process.

   Instructions for creating a Content Manager OnDemand response file can be found at:
   Silently installing Content Manager OnDemand
   Instruction for creating a Db2 response file can be found at:
   Sample response files (Linux, UNIX, and Windows)

3. Create a Dockerfile to perform the installation. This file should be in the root of your build directory. Below is a sample Docker file for reference.

```
FROM registry.access.redhat.com/rhel7/rhel

EXPOSE 1445

# Required packages
RUN yum install -y              \
    vi                          \
    ksh                         \
    zip                         \
    unzip                       \
    sudo                        \
    pam                         \
    pam.i686                    \
    passwd                      \
    ncurses-libs.i686           \
    file                        \
    libaio                      \
    libstdc++-devel.i686        \
    numactl-libs                \
    openssl &&                  \
    yum clean all &&            \
    mkdir /tmp/DB2INSTALLER &&  \
    mkdir /var/odsetup


#Copy install files
COPY  od/odlinux.bin                                  \
      od/odlinux10104.bin                             \
      od/response.file                                \
      db2/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar        \
      db2/db2rtcl.rsp                                 \
      gskit/gskcrypt64-8.0.50.84.linux.x86_64.rpm     \
      gskit/gskssl64-8.0.50.84.linux.x86_64.rpm /tmp/


#Install packages. Remove files when complete
RUN /tmp/odlinux.bin -i silent -f /tmp/response.file &&           \
    /tmp/odlinux10104.bin -i silent -f /tmp/response.file &&      \
    rpm -Uv /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm &&         \
    rpm -Uv  /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm  &&         \
    rm -f /tmp/odlinux10104.bin &&                                \
    rm -f /tmp/odlinux.bin &&                                     \
    rm -f /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm  &&          \
    rm -f /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm &&             \
    rm -f /tmp/response.file &&                                   \
    tar -xf /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar -C /tmp/DB2INSTALLER &&
      \
    rm /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar &&               \
    /tmp/DB2INSTALLER/server_awse_o/db2setup -r /tmp/db2rtcl.rsp &&    \
    rm -rf /tmp/DB2INSTALLER


COPY entrypoint.sh /entrypoint.sh
COPY scripts/* /var/odsetup/

ENTRYPOINT ["/entrypoint.sh"]

CMD ["start"]
```

4. Create your Content Manager OnDemand Docker image using the *docker build* command. Below is the abbreviated output generated by the build command:

```
[root@geeklab /build]# docker build -t ondemand:10.1.0.4 .
Sending build context to Docker daemon 2.941 GB
Step 1/9 : FROM registry.access.redhat.com/rhel7/rhel
 ---> 3da40a1670b5
Step 2/9 : EXPOSE 1445
 ---> Using cache
 ---> f6f382a2cfe2
Step 3/9 : RUN yum install -y vi
 ---> Using cache
 ---> c8970099c217
Step 4/9 : COPY od/odlinux.bin od/odlinux10104.bin
 ---> Using cache
 ---> ca8978af9898
Step 5/9 : RUN /tmp/odlinux.bin -i silent -f /tmp/response.file
 ---> Using cache
 ---> 432c82274c9a
Step 6/9 : COPY entrypoint.sh /entrypoint.sh
 ---> Using cache
 ---> 374cb13c711c
Step 7/9 : COPY scripts/* /var/odsetup/
 ---> Using cache
 ---> c964948be1c7
Step 8/9 : ENTRYPOINT /entrypoint.sh
 ---> Using cache
 ---> bca5984e1071
Step 9/9 : CMD start
 ---> Using cache
 ---> 33f73ec2105e
```

```
Successfully built 33f73ec2105e
```

Using the *docker image* command, you should now be able to see your newly created Content Manager OnDemand Docker image with the Db2 Runtime Client.

```
[root@geeklab /build]# docker images ondemand
REPOSITORY       TAG          IMAGE ID            CREATED              SIZE
ondemand         10.1.0.4     33f73ec2105e        20 minutes ago       4.22 GB
```

Starting your Content Manager OnDemand Docker container

In the previous section we laid the ground work for creating a Content Manager OnDemand Docker image. In order to make this image a fully functional Content Manager OnDemand container, we now need to look at some key details that were only briefly introduced above, namely the entrypoint.sh script, persistent volumes, Content Manager OnDemand config files and .od_envlist file.

Content Manager OnDemand storage

Your Content Manager OnDemand container will require access to persistent storage. To accomplish this, Docker volumes are used. Refer to the Docker documentation for more information on volumes.

At a minimum you will need a volume for Content Manager OnDemand document storage.

Create a directory called /data and subdirectories called arscache and arstmp on your Docker host. While arscache must be located on persistent storage, arstmp does not need to be. For this example, we will set up arstmp as persistent. Just keep in mind, it doesn't need to be.

You must ensure the uid number of the user that will run the ARSSOCKD process must match the uid number of the archive user on your Docker host.

Create the archive user on your Docker host and give it a unique uid, for example, 1234. Once the user is created, set the permissions on the Content Manager OnDemand document storage directory called arscache just as you would for a local Content Manager OnDemand installation:

```
chown -R archive:archive /data*
chmod 700 /data/arscache
chown g-s /data/arscache
```

To define the archive user in your Docker container, use the unix *useradd* command. Ensure the command you use to create this user ID also includes the same uid number as the Docker host to ensure arscache permissions are correct. For example:

```
useradd -r -d /home/odadmin odadmin --uid 1234
```

The *useradd* command to create the archive user should be issued in your entrypoint script prior to creating the Db2 instance.

Content Manager OnDemand configuration files

As previously mentioned, Content Manager OnDemand has been installed to the local system. The contents of the config directory were then copied to the /data/odconfig directory. We copy them rather than edit them in place in order to preserve the local original copy.

With the Content Manager OnDemand configuration files staged in our /data/odconfig directory, we need to edit them just as we would for a traditional Content Manager OnDemand installation.

ARS.INI – Update the SRVR_INSTANCE_OWNER and the SRVR_INSTANCE. For this example we are using ARCHIVE for the instance and archive as the instance owner. Also set the SRVR_OD_STASH to reflect the location of your stash file. Ideally this should be placed in the odconfig directory.

ARS.CACHE - Add the appropriate arscache directory to the end of the file. From the mount point above you can see that we are using /data/arscache

ARS.CFG – Update the values for ARS_TMP and ARS_PRINT_PATH to reflect the appropriate directory. Again, this is taken from the mount point above, /data/arstmp. The DB2INSTANCE we are using is archive.

Set environment variables (.od_envlist)

The env-file parameter is a convenient place to store information necessary for connecting to a remote Db2 database. Imagine you are starting multiple Content Manager OnDemand containers for different environments, such as development, test, etc. You can have multiple env-files with connection information for different databases.

When your container is started, the values are set and accessible from within your entrypoint script. Below is a sample env-file that defines the necessary information to connect to a remote Db2 database.

```
[root@geeklab /build]# cat .od_envlist
DB2INSTANCE=archive
DB2INST_PASSWORD=password
DBNAME=archive
DB2PATH=/opt/ibm/db2/V11.1_01/
DB2ADDR=172.17.0.2
STORAGE_DIR=/database
```

Create a stash file

At a minimum, your stash file should contain the Content Manager OnDemand userid and password and the Db2 userid and password. The stash file needs to be created using the ARSSTASH command.

Place the stash file in the /build/odconfig directory. As with the ARS.INI, ARS.CFG and ARS.CACHE files, this file will then be located in /opt/ibm/ondemand/V10.1/config once the container is started. The ARS.INI file should contain the following reference so that Content Manager OnDemand is aware of the file when starting ARSSOCKD:

```
SRVR_OD_STASH=/opt/ibm/ondemand/V10.1/config/<stashfile name>
```

Create entrypoint.sh script

When a Docker container starts, it calls an entrypoint command/script. In order for the Content Manager OnDemand ARSSOCKD process to start successfully, we need to do several things prior to starting it. Remember earlier we said that containers should be able to be destroyed and recreated with very little configuration. This is exactly where the entrypoint script comes in. The entrypoint script, in the case of Content Manager OnDemand, needs to ensure your database is accessible and that the Content Manager OnDemand tables have already been created. If the tables have not been created, the entrypoint script should create them. It should also initialize the system logging facility. Once this has been completed the script can then start ARSSOCKD.

The following is an example in pseudo code of what would be required inside your entrypoint script (assuming an external Db2 external database).

Note: The database should already exist prior to running your container. The values for variables beginning with '$' in the sample commands below are passed in via the –env-file parameter.

Create the Db2 instance owner id and the Db2 client instance:

```
/bin/su -c "$DB2PATH/instance/db2icrt $DB2INSTANCE"
```

Catalog the database node in the Content Manager OnDemand Docker container:

```
/bin/su -c "db2 catalog tcpip node ODINST remote $DB2ADDR server 50000" -
$DB2INSTANCE
```

Catalog the remote Db2 database in the Content Manager OnDemand Docker container:

```
/bin/su -c "db2 catalog database archive as archive at node ODINST" -
$DB2INSTANCE
```

Connect to the remote database:

```
db2 -x "connect to ${DBNAME}"
```

Check the remote database for the Content Manager OnDemand tables. If they do not exist, create them along with the Content Manager OnDemand Logging facility:

```
TABLE=$(db2 -x "select tabname from syscat.tables where
tabschema='${DB2INSTANCE^^}' and tabname='ARSSYS'")

    if [ $TABLE = "" ]; then
        /opt/ibm/ondemand/V10.1/bin/arsdb -tvf
        /opt/ibm/ondemand/V10.1/bin/arssyscr -l
    fi
```

Start ARSSOCKD:

```
/opt/ibm/ondemand/V10.1/bin/arssockd -v -S -I archive
```

The above pseudo code can handle starting a container against a database that has Content Manager OnDemand tables created in it and also one that does not.

Start your Content Manager OnDemand container

With everything now in place, let's look at the command required to start your first Content Manager OnDemand container:

```
[root@geeklab /build]# docker run -dit --name oddocker --env-file .od_envlist
-p 1445:1445 -v /data/arscache:/data/arscache:Z -v
/data/odconfig:/opt/ibm/ondemand/V10.1/config:Z -v
/data/arstmp:/data/arstmp:Z ondemand:10.1.0.4
```

The parameters of this command are as follows:

sudo docker run -dit --name oddocker - Creates the container and attaches the name "oddocker" to it.

--env-file .od_envlist - Passes the contents of the file *.od_envlist* to the container and sets each name value/pair as an environment variable inside the container. These values are then available to the entrypoint.sh script that is invoked when the container is started.

-p 1445:1445 - The *–p* parameter publishes port 1445 as port 1445 to the host running the container.

-*v /data/arscache:/data/arscache* - (Volume option) Mounts the host directory /data/arscache over the destination directory /data/arscache.

-*v /data/odconfig:/opt/ibm/ondemand/V10.1/config:Z* - (Volume option) Mounts the host /data/odconfig directory over the destination /opt/ibm/ondemand/V10.1/config directory.

-v /data/arstmp:/data/arstmp:Z - (Volume option) Mounts the host directory /data/arstmp over the destination directory /data/arstmp.

ondemand:10.1.0.4- The name of the image that the container is based on.

It is important to note that any contents already present in any of the local volumes will persist into the container. In mounting the volumes in the above fashion, they are mounted read/write. This means any content written will persist once the container is stopped.

Running Content Manager OnDemand maintenance

Just like any Content Manager OnDemand system, it is necessary to perform maintenance against your Content Manager OnDemand Docker container using the ARSMAINT and ARSDB commands. There are several ways in which this can be done:

1. By installing cron directly in your Docker image and configuring in your container.
2. By setting up a separate cron-enabled Docker container.
3. By using your Docker host with docker exec or docker run in your container.

The recommended option when dealing with scheduled tasks and Docker containers is the third option. For example, the following entry in your Docker host crontab would automatically start the ARSMAINT program every day at 4 am for the instance named archive against the container name ODDOCKER. The ARSMAINT program will maintain application group data in cache storage, including copying report data to archive storage:

```
      00 4 * * * /usr/bin/docker exec -u archive -d oddocker
/opt/ibm/ondemand/V10.1/bin/arsmaint –cdeimrsv
```

In addition to ARSMAINT, it is also recommended that you run ARSDB to properly maintain Content Manager OnDemand system tables.

The following example is a CRON record that automatically starts the ARSDB program to maintain the Content Manager OnDemand system tables for the instance named archive. The ARSDB program will run twice a month, on the 7th and 14th of each month, beginning at 5 am.

```
00 5 7,14 * * 00 4 * * * /usr/bin/docker exec -u archive -d oddocker
/opt/IBM/ondemand/V10.1/bin/arsdb -mv -I archive >> /tmp/arsdb.log 2>&1
```

Appendix A: Useful Docker commands and links

Useful commands

Stop a Docker container:

```
docker container stop <container name>
```

Start an existing Docker container:

```
docker container start <container name>
```

Remove a stopped Docker container:

```
docker rm <container name>
```

View logs of a Docker container (useful for troubleshooting container startup):

```
docker logs <container name>
```

Show the status of all Docker containers:

```
docker ps –a
```

Enter into a running Docker container in interactive mode:

```
docker exec -ti <container name> bash
```

Get low-level information about Docker containers (such as IP address):

```
docker inspect <container>
```

Useful links

Official Docker documentation:

Docker documentation

Db2 Developer-C Edition available via Docker Hub – A containerized version of Db2:

Full feature, free version for non-production environments. Ideal for developers.

Appendix B: Shrinking your Docker image

You may have noticed in the output of the *docker images* command above that the size of our Content Manager OnDemand image was 4.3GB. The reason the image is so large is because of the layer that is created when the COPY action is executed:

```
#Copy install files
COPY    od/odlinux.bin                                  \
        od/odlinux10104.bin                             \
        od/response.file                                \
        db2/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar        \
        db2/db2rtcl.rsp                                 \
        gskit/gskcrypt64-8.0.50.84.linux.x86_64.rpm     \
        gskit/gskssl64-8.0.50.84.linux.x86_64.rpm /tmp/
```

Even though we remove these files in a later action, they become a permanent part of the layer once the COPY action completes and thus create wasted space.

There is however a way to significantly reduce the size of the image. By combining the COPY and RUN actions into a single command, we can stage the temporary data, run the appropriate install command, and then clean up all temporary data in a single layer.

Here is an alternative version of the Dockerfile that does exactly this. In order to use this Dockerfile you will need to install an HTTP server either on your Docker host or some other server accessible to it. You will then stage all the install images (Db2, Content Manager OnDemand GSKIT, etc.) on this HTTP server. Using curl, the data is retrieved in the RUN action, followed by the install and then removal of the install images. Since we remove the install images as part of the RUN command, we are able to save roughly 3GB of space. Here is the sample Dockerfile:

```
FROM registry.access.redhat.com/rhel7/rhel

MAINTAINER Robert Russell <rob.russell@unicomsi.com>

EXPOSE 1445

# Required packages
RUN yum install -y                  \
    vi                              \
    ksh                             \
    zip                             \
    unzip                           \
    sudo                            \
    pam                             \
    pam.i686                        \
    passwd                          \
    ncurses-libs.i686               \
    file                            \
    libaio                          \
    libstdc++-devel.i686            \
    numactl-libs                    \
    openssl &&                      \
    yum clean all &&                \
    mkdir /tmp/DB2INSTALLER &&      \
    mkdir /var/odsetup

#Install packages. Remove files when complete
RUN curl -o /tmp/odlinux.bin http://localhost/od/odlinux.bin &&         \
    curl -o /tmp/response.file http://localhost/od/response.file &&     \
    curl -o /tmp/odlinux10104.bin http://localhost/od/odlinux10104.bin &&
\
    curl -o /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar
      http://localhost/db2/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar &&     \
    curl -o /tmp/db2rtcl.rsp http://localhost/db2/db2rtcl.rsp  &&       \
    curl -o /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm
http://localhost/gskit/gskcrypt64-8.0.50.84.linux.x86_64.rpm &&         \
    curl -o /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm
http://localhost/gskit/gskssl64-8.0.50.84.linux.x86_64.rpm &&           \
    chmod 777 /tmp/odlinux.bin &&                                       \
    chmod 777 /tmp/odlinux10104.bin &&                                  \
    /tmp/odlinux.bin -i silent -f /tmp/response.file &&                 \
    /tmp/odlinux10104.bin -i silent -f /tmp/response.file &&            \
    rpm -Uv /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm &&               \
    rpm -Uv  /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm  &&               \
    rm -f /tmp/odlinux10104.bin &&                                      \
    rm -f /tmp/odlinux.bin &&                                           \
    rm -f /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm  &&                \
    rm -f /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm &&                   \
    rm -f /tmp/response.file &&                                         \
    tar -xf /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar -C /tmp/DB2INSTALLER &&
      \
    rm /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar &&                     \
    /tmp/DB2INSTALLER/server_awse_o/db2setup -r /tmp/db2rtcl.rsp &&     \
    rm -rf /tmp/DB2INSTALLER

COPY entrypoint.sh /entrypoint.sh
COPY scripts/* /var/odsetup/

ENTRYPOINT ["/entrypoint.sh"]

CMD ["start"]
```

Build this image just as you did using the previous version of the Dockerfile:

```
      root@geeklab /build]# docker build -t ondemand:10.1.0.4 .
```

Comparing these two images, you can see that we are able to reduce the size by nearly 3GB.

```
[root@geeklab /build/scripts]# docker images

REPOSITORY  TAG         IMAGE ID         CREATED           SIZE
ondemand    10.1.0.4    9b5a05c40bfe     50 minutes ago    4.22 GB
od_small    10.1.0.4    83d24b441cca     7 days ago        1.28 GB
```

Appendix C: Exits

If your Content Manager OnDemand system requires the use of user exits like ARSUSEC, ARSLOG, or ARSPRT, you may wish to change the default location in which to find these. To do so you add the ARS_USER_EXITS_DIR parameter to your ARS.CFG file.

If you specify this parameter, all exits must exist in this directory.

For example, set the user exits directory to /data/exits and start your container with this mounted directory.

Create a new directory on your Docker host:

mkdir /data/exits

Add all the required exits to the /data/exits directory.

Add the following line to the ARS.CFG file:

ARS_USER_EXITS_DIR=/data/exits

Run your container with the additional volume mounted.

```
docker run -dit --name oddocker --env-file .od_envlist -p 1445:1445 -v
/data/arscache:/data/arscache:Z -v /data/exits:/data/exits:Z -v
/data/odconfig:/opt/ibm/ondemand/V10.1/config:Z -v
/data/arstmp:/data/arstmp:Z ondemand:10.1.0.4
```

Content Manager OnDemand will now look for all user exits to be located in the /data/exits directory.

Appendix D: IBM Spectrum Protect (TSM)

If your Content Manager OnDemand system requires access to IBM Spectrum Protect (SP; formerly known as Tivoli Storage Manager or TSM), you will need to add the SP API to your Content Manager OnDemand Docker image.

To accomplish this, download the IBM Spectrum Protect Client package from Passport Advantage. Then create a new staging directory in your build directory to store the appropriate SP rpm packages. For example:

```
[root@geeklab /build/sp]# ls
TIVsm-API64.x86_64.rpm
```

Once the data has been staged, you will need to edit your Dockerfile. First, add a line to copy the install binary to your Docker image:

```
#Copy install files
COPY    od/odlinux.bin                                  \
        od/odlinux10104.bin                             \
        od/response.file                                \
        db2/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar        \
        db2/db2rtcl.rsp                                 \
        gskit/gskcrypt64-8.0.50.84.linux.x86_64.rpm     \
        gskit/gskssl64-8.0.50.84.linux.x86_64.rpm       \
        sp/TIVsm-API64.x86_64.rpm /tmp/
```

You will then add the line to perform the installation. The last step is to remove the install image to prevent wasted space in your Docker image.

```
RUN /tmp/odlinux.bin -i silent -f /tmp/response.file &&           \
    /tmp/odlinux10104.bin -i silent -f /tmp/response.file &&      \
    rpm -Uv /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm &&         \
    rpm -Uv /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm  &&          \
    rpm -iv /tmp/TIVsm-API64.x86_64.rpm  &&                       \
    rm -f /tmp/TIVsm-API64.x86_64.rpm &&                          \
    rm -f /tmp/odlinux10104.bin &&                                \
    rm -f /tmp/odlinux.bin &&                                     \
    rm -f /tmp/gskcrypt64-8.0.50.84.linux.x86_64.rpm  &&          \
    rm -f /tmp/gskssl64-8.0.50.84.linux.x86_64.rpm &&             \
    rm -f /tmp/response.file &&                                   \
    tar -xf /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar -C /tmp/DB2INSTALLER &&   \
    rm /tmp/DB2_AWSE_REST_Svr_11.1_Lnx_86-64.tar &&               \
    /tmp/DB2INSTALLER/server_awse_o/db2setup -r /tmp/db2rtcl.rsp && \
    rm -rf /tmp/DB2INSTALLER
```

Before completing the last change to your Dockerfile, create a dsm.sys and dsm.opt file on your local Docker host.

Create a directory to store the dsm.sys and dsm.opt files:

```
mkdir /build/tsmconfig
```

Create the files with the necessary information to connect to a remote SP server. For example:

```
cat /build/tsmconfig/dsm.opt
SERVERNAME ARCHIVE
QUIET


cat /build/tsmconfig/dsm.sys
SERVERNAME             ARCHIVE
COMMMETHOD             TCPIP
TCPPORT                1500
TCPSERVERADDRESS       sp.server.com
COMPRESSION            OFF
```

With the files created, we can now add the appropriate lines to our Docker file to ensure they make it to our Docker image.

Add the following after your RUN Docker statement:

```
# Copy SP options files into default install dir
COPY tsmconfig/dsm.opt /opt/tivoli/tsm/client/api/bin64/
COPY tsmconfig/dsm.sys /opt/tivoli/tsm/client/api/bin64/
RUN chmod a+r /opt/tivoli/tsm/client/api/bin64/dsm.*
```

The last step is to edit your Content Manager OnDemand ARS.CFG file which we have staged in /data/odconfig. Change the parameter ARS_STORAGE_MANAGER from CACHE_ONLY to TSM. You should also change the TSM LOG parameters to point to an appropriate directory. Since we have already created a temp directory for Content Manager OnDemand we will use that. You could, however, create any directory you wish. The TSM section of ARS.CFG should now look like:

```
#######################################
# TSM Parameters (Object Server Only) #
#######################################
DSMSERV_DIR=/opt/tivoli/tsm/server/bin
DSMSERV_CONFIG=/opt/tivoli/tsm/server/bin/dsmserv.opt
DSM_DIR=/opt/tivoli/tsm/client/api/bin64
DSM_CONFIG=/opt/tivoli/tsm/client/api/bin64/dsm.opt
DSM_LOG=/data/arstmp
DSMG_DIR=/opt/tivoli/tsm/client/api/bin64
DSMG_CONFIG=/opt/tivoli/tsm/client/api/bin64/dsm.opt
DSMG_LOG=/data/arstmp
DSMI_DIR=/opt/tivoli/tsm/client/api/bin64
DSMI_CONFIG=/opt/tivoli/tsm/client/api/bin64/dsm.opt
DSMI_LOG=/data/arstmp
```

Use the previously defined Docker build command to create an updated image, now with Spectrum Protect (TSM) support.

```
      docker build -t ondemand:10.1.0.4 .
```

Once your container is started, you can create storages nodes using the OnDemand Administrator client and test your new image. For troubleshooting, refer to the log files which will be located in /data/arstmp.

Appendix D: Oracle database support

Supporting a remote Oracle database is similar to supporting a remote Db2 instance. For example, instead of installing the Db2 Runtime Client you would install the Oracle Client.

The information provided in this document, coupled with support from your Oracle administrator should provide the necessary foundation for configuring a Content Manager OnDemand Docker image with support for Oracle.