RDKB Containerization in RPI - User Manual - 2019 M3

This document explains about how to build RDKB containerization image and use it. In addition it also has some useful commands which can be used to handle the containers

Procedure to create new container with Build steps

The code is developed and changes made in necessary files for containerization.

Below are the list of steps that needs to performed for creating a container for any component or application

1. **Code Sync /Download**
2. **Creation of XML file**
3. **Include XML file in container generator recipe - lxc-container-generator-native.bbappend**
4. **Providing File permission for the containers**
5. **Allowing D-Bus socket to access the containers (Specific to RDK-B Architecture)**
6. **Building the lxc image**
7. **Flashing the container image**

Code Sync/Download

To download code, following commands are needed to be executed

Code sync

```
$ repo init -u <url> -m <manifest file> -b <branch>
$ repo sync -j4 --no-clone-bundle

#Container Branch
$ repo init -u https://code.rdkcentral.com/r/manifests -b rdkb-container -m rdkb-container.xml
$ repo sync -j4 --no-clone-bundle
```

Creation of XML file

Create a *.xml file on the following path **meta-cmf-raspberrypi/recipes-containers/lxc-container-generator/files/xml/*.xml**

This xml file would describe the following parameters for an container generation

1. Launcher Name
2. Application Name with path
3. lxc config file creation
4. rootfs creation

meta-cmf-raspberrypi/recipes-containers/lxc-container-generator/files/xml/lxc_conf_Psm.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<CONTAINER  SandboxName="CONTAINER_FOLDER_NAME">
    <LxcParams>
        <LauncherName>"LAUNCHER_SCRIPT_NAME"</LauncherName>
        <ExecName>"APPLICATION_NAME_WITH_PATH"</ExecName>
        <ExecParams>"COMMAND_LINE_ARGUMENTS"</ExecParams>
                <SystemdNotify create="yes">
                        <PidFile>"PID_FILE_WITH_PATH"</PidFile>
                </SystemdNotify>
        <StopFunction enable="true"></StopFunction>
    </LxcParams>
        <LxcConfig>
-        <UserName>"USER_NAME"</UserName>
        <GroupName>"GROUP_NAME"</GroupName>
        <CGroupSettings>
            <DeviceCgroup>
                <DevicesDeny>a</DevicesDeny>
                <AllowDefaultDevices enable="yes"/>
            </DeviceCgroup>
        </CGroupSettings>
                <Environment>
                   <Variable>DBUS_SESSION_BUS_ADDRESS=unix:path=/var/run/dbus/system_bus_socket</Variable>
                </Environment>

        <Network type="none"></Network>
                <Dbus enable="true"></Dbus>
        <Rootfs create="yes">
            <MountPoints>
<!-- /bin -->
                <Entry type="file">
                    <Source>/bin/sh</Source>
                    <Destination>bin/sh</Destination>
                    <Options>ro,bind,nosuid,nodev</Options>
                </Entry>

<!-- /proc -->
                <Entry type="dir">
                    <Source>proc</Source>
                    <Destination>proc</Destination>
                    <FsType>proc</FsType>
                    <Options>defaults,noexec,nosuid,nodev,hidepid=2</Options>
                </Entry>

            </MountPoints>

            <LibsRoBindMounts>
                <Entry>ld</Entry>
                <Entry>libtr181</Entry>
                <Entry>libxml2</Entry>
                <Entry>libz</Entry>
                <Entry>libccsp_common</Entry>
            </LibsRoBindMounts>
        </Rootfs>
</LxcConfig>
</CONTAINER>
```

Example File: lxc_conf_Psm.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<CONTAINER  SandboxName="PSMSSP">
    <LxcParams>
```

```
        <LauncherName>PsmSsp</LauncherName>
        <ExecName>/usr/bin/PsmSsp</ExecName>
        <ExecParams>-subsys eRT.</ExecParams>
                <SystemdNotify create="yes">
                        <PidFile>/var/tmp/PsmSsp.pid</PidFile>
                </SystemdNotify>
        <StopFunction enable="true"></StopFunction>
    </LxcParams>
        <LxcConfig>
-        <UserName>psm</UserName>
        <GroupName>psm</GroupName>
        <CGroupSettings>
            <DeviceCgroup>
                <DevicesDeny>a</DevicesDeny>
                <AllowDefaultDevices enable="yes"/>
            </DeviceCgroup>
        </CGroupSettings>
                <Environment>
                   <Variable>DBUS_SESSION_BUS_ADDRESS=unix:path=/var/run/dbus/system_bus_socket</Variable>
                </Environment>


        <Network type="none"></Network>
                <Dbus enable="true"></Dbus>
        <Rootfs create="yes">
            <MountPoints>
<!-- /bin -->
                <Entry type="file">
                    <Source>/bin/sh</Source>
                    <Destination>bin/sh</Destination>
                    <Options>ro,bind,nosuid,nodev</Options>
                </Entry>
                <Entry type="file">
                    <Source>/usr/bin/PsmSsp</Source>
                    <Destination>usr/bin/PsmSsp</Destination>
                    <Options>ro,bind,nosuid,nodev</Options>
                </Entry>

                <Entry type="file">
                    <Source>/bin/touch</Source>
                    <Destination>bin/touch</Destination>
                    <Options>ro,bind,nosuid,nodev</Options>
                </Entry>
                <Entry type="file">


<!--rdklogs-->
                <Entry type="dir">
                                        <Source>/rdklogs</Source>
                                        <Destination>rdklogs</Destination>
                                        <Options>rw,bind,noexec,nosuid</Options>
                 </Entry>

            </MountPoints>

            <LibsRoBindMounts>
                <Entry>ld</Entry>
                <Entry>libtr181</Entry>
                <Entry>libxml2</Entry>
                <Entry>libz</Entry>
                <Entry>libccsp_common</Entry>
                <Entry>libsyscfg</Entry>
                <Entry>libsysevent</Entry>
                                ...
                <Entry>liblzma</Entry>
                <Entry>libdl</Entry>
                <Entry>libtinfo</Entry>

            </LibsRoBindMounts>
        </Rootfs>
</LxcConfig>
```

```
</CONTAINER>
```

Include XML file in container generator recipe - lxc-container-generator-native.bbappend

Include the **lxc_conf_<NAME>.xml** file on the bb file **meta-cmf-raspberrypi/recipes-containers/lxc-container-generator/lxc-container-generator-native.bbappend**

Adding XML into recipie

```
#For example, Adding lxc_conf_Psm.xml to the recipie

SRC_URI_append = "${@bb.utils.contains('DISTRO_FEATURES', 'lxc-secure-containers-br', ' file://xml/lxc_conf_Psm.
xml ', '', d)}"

do_install_append () {
    ${@bb.utils.contains('DISTRO_FEATURES', 'lxc-secure-containers-br', ' install_lxc_config secure
lxc_conf_Psm.xml ', '', d)}
}
```

Providing File permission for the containers

Provide user permission for the new container to run as unprivileged , in **meta-cmf-raspberrypi/recipes-core/images/add-users-groups-file-owners-and-permissions-broadband.inc**

Following changes need to be added, new container which will create userid and groupid for container about to create with necessary permission

Adding user and permission

```
EXTRA_USERS_PARAMS += "\
    useradd -u <uid> -G dbusgrp                              -r -s /bin/false <container_user_name>           ; \

   ROOTFS_CHOWN_SETCAP += " -o <container_user_name>:<container_group_name> -m o-rwx /usr/bin
/application_name     \n"

-------------------------------------------------------------------------------------------------------------------
-----
#Adding PSM permission

 EXTRA_USERS_PARAMS += "\
    useradd -u 703 -G dbusgrp                              -r -s /bin/false psm           ; \

   ROOTFS_CHOWN_SETCAP += " -o psm:psm -m o-rwx /usr/bin/PsmSsp     \n"
```

Allowing D-Bus socket to access the containers (Specific to RDK-B Architecture)

Dbus socket should allow the newly created conatiner to access system bus, to do so ,we need to add the conatiner user name to **system.conf**

Adding user and permission

```
#Add the new user name in below line ,

sed -i '/allow user/c\<deny user="*"/>\n<allow user="ccspcr"/>\n<allow user="psm"/>\n<allow user="pandm"
/>\n<allow user="ccspwifi"/>\n<allow user="USER_NAME"/>\n<allow user="ccsplmlite"/>\n<allow user="root"/>' ${D}
/usr/share/dbus-1/system.conf
```

Building the lxc image

Compile/build

Go to the <*workspace*>

```
$ cd <workspace>
```

execute the following command:

```
$ source meta-cmf-raspberrypi/setup-environment
```

A list is obtained, select the following option from among the list:

```
12) meta-cmf-raspberrypi/conf/machine/raspberrypi-rdk-broadband-lxc.conf
```

The path is automatically directed to:

```
<workspace>/build-raspberrypi-rdk-broadband-lxc/
```

Use the following command to compile the complete code:

Build generation

```
$ bitbake rdk-generic-broadband-lxc-image
```

To compile a specific component container use the below command:

```
$ bitbake <component> -c compile -f
```

Ex:

```
$ bitbake ccsp-dmcli -c compile -f
```

The above command will compile the dmcli component container.

Image path

Once the code generation for containerization is compiled, the image for the same is found under the path:

```
$ cd <workspace>/build-raspberrypi-rdk-broadband-lxc/tmp/deploy/images/raspberrypi-rdk-broadband-lxc/
```

The image to be considered:

<ImageName.rootfs.rpi-sdimg>

Ex Path:

```
$ cd container-work/build-raspberrypi-rdk-broadband-lxc/tmp/deploy/images/
```

Image:

raspberrypi-rdk-broadband-lxcrdk-generic-broadband-lxc-image_default_20190327101556.rootfs.rpi-sdimg

Flashing the container image

Command to flash the image

Generated image has to be flashed to an SD card using this command in local PC:

```
$ sudo dd if=<path to ImageName.rootfs.rpi-sdimg> of=<path to SD card space> bs=4M
```

Ex:

```
$ sudo dd if=rdkb-generic-broadband-image_default_20181026100202.rootfs.rpi-sdimg of=/dev/sdd bs=4M
```

The SD card is inserted to the Raspberry Pi board and booted to check for containers created.

The Raspberry Pi board is connected to the PC via a USB to serial converter and the logs can be checked in console or can be connected via HDMI cable to a TV and logs will be shown in the terminal

Steps/commands to be executed to check for successful container generation

1.Check for the container generated in the following path:

```
  /container/<container_name>/
```

Three folders will be generated under *<container_name>:*

i. conf -- This folder contains the *lxc.conf* file.

ii. launcher -- This folder contains the script to execute the container.

iii. rootfs – This folder contains all the root files which includes *bin*, *sbin*, *usr*, *lib*, etc.

The above folders will be created automatically based on the xml files written in the workspace.

XML File path

```
<workspace>/meta-cmf-raspberrypi/recipes-containers/lxc-container-generator/files/xml/<container.xml>
```

Ex:

```
container-work/meta-cmf-raspberrypi/recipes-containers/lxc-container-generator/files/xml/lxc_conf_CcspPandMSsp.xml
```

The above xml file is written for “P and M” container and will be responsible for the generation of the 3 folders mentioned above.

2.Check all the processes running in regard to lxc.

```
$ ps -Af | grep lxc
```

The Pi terminal should show logs similar to the following:

5530	root	0.00	/usr/bin/lxc-execute -n PSMSSP -f /container/PSMSSP/conf/lxc.conf -- /usr/bin/PsmSsp -subsys eRT .
5534	psm	0.00	/init.lxc.static –gid 705 –uid 705 -- /usr/bin/PsmSsp -subsys eRT .
6433	root	0.00	/usr/bin/lxc-execute -n CCSPPANDM -f /container/CCSPPANDM/conf/lxc.conf -- /usr/bin/gw_prov_utopia
6435	pandm	0.00	/init.lxc.static –gid 706 –uid 706 -- /usr/bin/gw_prov_utopia
6520	root	0.00	/usr/bin/lxc-execute -n DBUS -f /container/DBUS/conf/lxc.conf -- /usr/bin/dbus-daemon –system –nofork –nopidfile –systemd-activation
6532	dbus	0.00	/init.lxc.static –gid 703 –uid 703 -- /usr/bin/dbus-daemon –system –nofork –nopidfile –systemd-activation
6574	root	0.00	/usr/bin/lxc-execute -n CCSPCR -f /container/CCSPCR/conf/lxc.conf -- /usr/bin/CcspCrSsp -subsys eRT .
6577	ccspcr	0.00	/init.lxc.static –gid 704 –uid 704 -- /usr/bin/CcspCrSsp -subsys eRT .
6737	root	0.00	grep lxc

WiFi Container :: Manual Start

Due to some limitations in invoking wifi driver with container permission, ccspwifi container has to be run manually in latest environment with below procedure

After checking pandm container status which has to be up and running, run the following commands

- ```
  $ sh /lib/rdk/wifi_container.sh
  ```

  - this command may return error which can be ignored
- ```
  $ sh /container/CCSPWIFI/launcher/CcspWifiSsp.sh start
  ```

Please wait for few seconds/mins till log "Enter WiFi Loop" is observed in the console

- ```
  $ lxc-attach -n CCSPWIFI -f /container/CCSPWIFI/conf/lxc.conf
  ```

After execution of above command shell will change to ccspwifi
Inside container shell please run following command

- ```
  $ hostapd -B /nvram/hostapd0.conf
  ```

Now Check (in host shell) if CCSPWIFI container is running with corresponding name

Now, can able to connect wifi client(Laptop or mobile) using default ssid: RPI3_RDKB-AP0 and password: rdk@1234

Invoking containers manually in Raspberry Pi terminal

1. Run respective container scripts to execute individual container when container fails in boot up process:

```
$ sh /container/<Container Name>/launcher/<container.sh> start
```

Ex:

```
$ sh /container/PSMSSP/launcher/PsmSsp.sh start
```

The above command can be executed to run PSM container and see its logs in Console.

2. Run command/commands inside the container specified by the Name when the container is already running:

```
$ lxc-attach -n CCSPWIFI -f /container/CCSPWIFI/conf/lxc.conf
```

Ex:

```
$ lxc-attach -n CCSPWIFI -f /container/CCSPWIFI/conf/lxc.conf
```

The above command can be executed to run the commands specified by lxc.conf file.

Containers and their respective files to be referred in Raspberry Pi terminal

DBUS

i. Service file used:

```
/lib/systemd/system/dbus.service
```

ii. Directory name:

```
/container/DBUS/
```

iii. Conf file used for it:

```
/container/DBUS/conf/lxc.conf
```

iv. Service handled by this file:

DBUS acts as a message bus. DBUS container contains commands which initialize the message bus and connects all the components to each other for communication. Dbus path differs for each component.

PSM

i. Service file used:

```
/lib/systemd/system/PsmSsp.service
```

ii. Directory name:

```
/container/PSMSSP/
```

iii. Conf file used for it:

```
/container/PSMSSP/conf/lxc.conf
```

iv. Service handled by this file:

Persistent Storage Manager container stores current/default system configurations and retrieves them from backup files when required on behalf of other component containers.

P and M

i. Service file used:

```
/lib/systemd/system/CcspPandMSsp.service
```

ii. Directory name:

```
/container/CCSPPANDM/
```

iii. Conf file used for it:

```
/container/CCSPPANDM/conf/lxc.conf
```

iv. Service handled by this file:

Provisioning and Management container acts an an interface and responds to any query or setting of variables. It keeps a track of changes in attributes and notifies to other component containers.

This container also handles Gateway provisioning for provisioning LAN and DOCSIS which includes IP addressing, routing and other gateway initialisation. It initiates registration of all servies on boot up.

WiFi Agent

i. Service file used:

```
/lib/systemd/system/ccspwifiagent.service
```

ii. Directory name:

```
/container/CCSPWIFI/
```

iii. Conf file used for it:

```
/container/CCSPWIFI/conf/lxc.conf
```

iv. Service handled by this file:

WiFi Agent container provides each component container, the access to data model APIs and driver specific APIs to manage WiFi specific parameters through a component specific Abstraction layer. It helps to manage and configure the gateway's wifi access point interface.

CR

i. Service file used:

```
/lib/systemd/system/CcspCrSsp.service
```

ii. Directory name:

```
/container/CCSPCR/
```

iii. Conf file used for it:

```
/container/CCSPCR/conf/lxc.conf
```

iv. Service handled by this file:

Component registrar acts as a centralized container for registration of all the component containers with their respective name, version, dbus path and namespace.

Useful commands for container execution

lxc-execute

This command is used to quickly launch a container in an isolated environment. It mainly runs the specified command into the container via intermediate process lxc-init (forwards the received signal to starting command). Lxc-execute uses the configurations specified by the lxc-create process.

Ex:

```
$ lxc-execute -n DBUS -f /container/DBUS/conf/lxc.conf

$ lxc-execute -n PSMSSP -f /container/PSMSSP/conf/lxc.conf

$ lxc-execute -n CCSPPANDM -f /container/CCSPPANDM/conf/lxc.conf

$ lxc-execute -n CCSPWIFI -f /container/CCSPWIFI/conf/lxc.conf

$ lxc-execute -n CCSPCR -f /container/CCSPCR/conf/lxc.conf
```

lxc-attach

This command attaches to the container namespace and run a specified command inside, the already executing container.

Ex:

```
$ lxc-attach -n DBUS -f /container/DBUS/conf/lxc.conf

$ lxc-attach -n PSMSSP -f /container/PSMSSP/conf/lxc.conf

$ lxc-attach -n CCSPPANDM -f /container/CCSPPANDM/conf/lxc.conf

$ lxc-attach -n CCSPWIFI -f /container/CCSPWIFI/conf/lxc.conf

$ lxc-attach -n CCSPCR -f /container/CCSPCR/conf/lxc.conf
```

hostapd

hostapd is software daemon used for turning the normal network interface to access point.

Ex:

```
$ hostapd -B /nvram/hostapd0.conf
```

Usage of Container based RDKB gateway

Host PC has to be connected with the Raspberry Pi board via USB to Ethernet adaptor or by WiFi

WiFi client should be connected to the WiFi Access point RPi using the SSID(Default: RPI3-RDKB-AP0) and password(Default: rdk@1234) given.

Host based management can be done via WebUI by opening http://10.0.0.1/ in connected client device