Integrating Docker containers in Yocto Project®

Sandra Tobajas, Savoir-faire Linux

Yocto Project Virtual Summit Europe, October 29-30, 2020
Presentation

• Sandra Tobajas
  • Product engineer at Savoir-faire Linux in Montreal
  • Consultant in embedded systems and product engineering
  • Worked on some BSPs and Linux OSes on various architectures/platforms
  • I like:
    • to work on low level features (bootloader integration, system programming),
    • to parse protocols,
    • to implement or find network security attacks.
It always starts with a customer’s need

• Worked on different projects with customers
  • Different business and industrial contexts
    • Healthcare, avionics, manufacturing, entertainment, home automation, etc...

• But most of them have this question in common

  “Hello, I have a (new/old) board and I would like to embed my (rich/legacy) application in a Linux firmware, can you help us ?”

• Usage of containers to solve this problem
What is this session about?

- Why running containers in embedded systems
- Production/Industrialization challenges
- Integrating Docker containers within Yocto
Why running containers in embedded systems?
Why running container in embedded systems?

- Application independent of the base OS
Why running container in embedded systems?

- Focus on the application development
  - Platform modularity and application portability
  - Allow asynchronous development cycles
Why running container in embedded systems?

- User-space isolation
- Really low overhead vs full/para virtualization
- Platform modularity and portability
- Speed up application development time
Production/Industrialization challenges
Production/Industrialization/Embedded system challenges

- Hardware resource limitations
- System reliability and data integrity
- Cross-platform development
- System provisioning
- System update
- Container traceability and reproducibility
- License compliance
Production/Industrialization challenges

- **Target requirements**
  - No connectivity on the board
  - Immutable read-only root file system
  - Boot integrity verification

- **Build requirements**
  - Reproducibilty and system provisioning
  - License compliance
  - Yocto build system
  - Containerized build system
    - https://github.com/savoirfairelinux/cqfd
  - Docker images already stored in a registry
Production/Industrialization challenges

• System provisioning
  • Container integration during build time
  • Container engine installed natively in the build system
  • Docker-in-Docker solution
Production/Industrialization challenges

• **Data corruption**
  - Major cause of embedded device failure
  - Avoid writing operations on rootfs
  - Writable storage options
    - Volatile and non volatile writable filesystem

• **Boot integrity verification**
  - Verifying boot sequence loading
  - Ensure the container image integrity at boot
Integrating Docker containers within Yocto
Integrating Docker containers within Yocto

- **Two approaches**
  - Embed Docker archive in the root filesystem
  - Populate the Docker store into the target root filesystem

- **Docker store = /var/lib/docker directory**
Integrating Docker containers within Yocto

- Meta-embedded-container tree
Integrating Docker archive within Yocto

- The distribution file

```plaintext
include conf/distro/poky.conf

DISTRO = "embedded-container-distro"

# Append poky based features
DISTRO_FEATURES_append = " virtualization"
DISTRO_FEATURES_append = " systemd"

# Add these binaries to the HOSTTOOLS_NONFATAL variable to allow them to
# be called from within the recipe build task. They are specified as
# NONFATAL as they are not installed at the same time on the build
# system.
HOSTTOOLS_NONFATAL += "sudo pidof dockerd podman newgidmap newuidmap"

# Use systemd as init manager
VIRTUAL-RUNTIME_init_manager = "systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED += "sysvinit"
VIRTUAL-RUNTIME_initscripts = ""
```
Integrating Docker archive within Yocto

- The image recipe file

```bash
require recipes-core/images/core-image-base.bb

DESCRIPTION = "Core image with embedded container images"

IMAGE_FSTYPES += "squashfs"

WKS_FILE = "embedded-container.wks"

IMAGE_FEATURES_append = "\debug-tweaks \post-install-logging \read-only-rootfs \ssh-server-dropbear"

IMAGE_INSTALL_append = "\container-image \docker"
```
Integrating Docker archive within Yocto

- The image recipe file

```bash
update_fstab_archive() {
    install -d "${IMAGE_ROOTFS}/${datadir}/docker-store"

    cat >> "${IMAGE_ROOTFS}${sysconfdir}/fstab" <<EOF
    tmpfs   ${datadir}/docker-store tmpfs defaults 0 0
    ${datadir}/docker-store /var/lib/docker none noauto,bind 0 2
    /tmp/docker-data /etc/docker none noauto,bind 0 2
    EOF
}

ROOTFS_POSTPROCESS_COMMAND += "${@bb.utils.contains("IMAGE_INSTALL","container-image","update_fstab_image;","",d)}"
ROOTFS_POSTPROCESS_COMMAND += "${@bb.utils.contains("IMAGE_INSTALL","container-archive","update_fstab_archive;","",d)}"
```
Integrating Docker archive within Yocto

- Container-image recipe
  - Refactoring as a class
Integrating Docker archive within Yocto

- Container image manifest
  - The docker registry name
  - The tag
  - Local container name
Integrating Docker archive within Yocto

- Container-archive.bb recipe

```bash
# The order should be:
# 1. do_fetch
# 2. do_pull_image
# 3. do_tag_image
# 4. do_save_image
# 5. do_install
addtask pull_image before do_tag_image after do_fetch
addtask tag_image before do_save_image after do_pull_image
addtask save_image before do_install after do_tag_image
```
Integrating Docker archive within Yocto

- Container-archive.bb recipe

```bash
# Pull the container images from the manifest file.
do_pull_image() {

  [ -f "/${WORKDIR}/${MANIFEST}" ] || bbfatal "${MANIFEST} does not exist"

  # Specify the PATH env variable allowing Bitbake:
  # - to look for podman binary as /usr/bin is not defined in the originally PATH env variable.
  # - to call /usr/bin/newgidmap and /usr/bin/newuidmap binaries which set uid and gid mapping of a user namespace.
  local name version
  while read -r name version _ ; do
    if ! PATH=/usr/bin:${PATH} podman pull "${name}:${version}"; then
      bbfatal "Error pulling ${name}:${version}"
    fi
  done < "${WORKDIR}/${MANIFEST}"
```

Integrating Docker archive within Yocto

- Container-archive.bb recipe

```bash
# Tag the container images with the tag specified in the manifest file.
do_tag_image() {
    [ -f "${WORKDIR}/${MANIFEST}" ] || bbfatal "${MANIFEST} does not exist"
    local name version tag
    while read -r name version tag _; do
        if ! PATH=/usr/bin:${PATH} podman tag "${name}:${version}" "${tag}:${version}"; then
            bbfatal "Error tagging ${name}:${version}"
        fi
    done < "${WORKDIR}/${MANIFEST}"
}
# Introducing Docker archive within Yocto

- **Container-archive.bb**

```bash
# Save the container images.
do_save_image() {
    local name version archive tag
    mkdir -p "${STORE_DIR}"
    while read -r name version tag _; do
        archive="${tag}-${version}.tar"
        if [ -f "${WORKDIR}/${archive}" ]; then
            bbnote "Removing the archive ${STORE_DIR}/${archive}"
            rm "${WORKDIR}/${archive}"
        fi

        if ! PATH=/usr/bin:$PATH podman save --storage-driver overlay "${tag}:${version}" \
                -o "${WORKDIR}/${archive}"; then
            bbfatal "Error saving ${tag} container"
        fi
    done < "${WORKDIR}/${MANIFEST}"
}
```
Integrating Docker archive within Yocto

- Container-archive.bb recipe

```bash
# Install the manifest inside the root filesystem.
do_install() {
    local name version archive tag
    install -d "${D}${datadir}/container-images"
    install -m 0400 "${WORKDIR}/${MANIFEST}" "${D}${datadir}/container-images/"
    while read -r name version tag _; do
        archive="${tag}-${version}.tar"
        [ -f "${WORKDIR}/${archive}" ] || bbfatal "${archive} does not exist"
        install -m 0400 "${WORKDIR}/${archive}" "${D}${datadir}/container-images/"
    done < "${WORKDIR}/${MANIFEST}"

    install -d "${D}${systemd_unitdir}/system"
    install -m 0644 "${WORKDIR}/container-archive.service" "${D}${systemd_unitdir}/system"
    install -d "${D}${bindir}"
    install -m 0755 "${WORKDIR}/container-image.sh" "${D}${bindir}/container-image"
    install -m 0755 "${WORKDIR}/container-load.sh" "${D}${bindir}/container-load"
}
```
Integrating Docker archive within Yocto

• During boot time
  1) Mount the Docker store in a writable partition.
  2) Execute the Docker daemon.
  3) Load the docker archive file in the Docker store using `docker load` command.
  4) Finally, run the Docker image.
Integrating Docker archive within Yocto

- During boot time
  - /var/lib/docker bind mounted on a tmpfs directory
  - /etc/docker mounted on a tmpfs directory

```
root@genericx86-64-# findmnt | tail -4
|--/var/lib
| `--/var/lib/docker    overlay rw,relatime,lowerdir=/var/lib,up...`}
|--/var/cache
| `--/etc/docker        tmpfs[/docker-data] tmpfs rw,nosuid,nodev`
```
Integrating Docker archive within Yocto

• During boot time
  - Container-archive.service

```
[Unit]
Description=Load and start container image at boot
After=mount-noauto.service docker.service
Requires=mount-noauto.service docker.service docker.socket

[Service]
Type=simple
RemainAfterExit=yes
ExecStartPre=/usr/bin/container-load start
ExecStart=/usr/bin/container-image start
ExecStop=/usr/bin/container-image stop
Restart=on-failure

[Install]
WantedBy=multi-user.target
```
Integrating Docker archive within Yocto

- During boot time
  - Container archives present in the rootfs
  - All container images running
Integrating Docker archive within Yocto

- With docker-compose
  - Embedding docker-compose.yml in the rootfs

```bash
# Install the manifest inside the root filesystem.
do_install() {
    local name version archive tag
    install -d "${D}${datadir}/container-images"
    install -m 0400 "${WORKDIR}/${MANIFEST}" "${D}${datadir}/container-images/"
    install -m 0400 "${WORKDIR}/docker-compose.yml" "${D}${datadir}/container-images/"
    while read -r name version tag _; do
        archive="${tag}-${version}.tar"
        [ -f "${WORKDIR}/${archive}" ] || bbfatal "${archive} does not exist"
        install -m 0400 "${WORKDIR}/${archive}" "${D}${datadir}/container-images/"
    done < "${WORKDIR}/${MANIFEST}"

    install -d "${D}${systemd_unitdir}/system"
    install -m 0644 "${WORKDIR}/container-multiple-images.service" "${D}${systemd_unitdir}/system"
    install -d "${D}${bindir}"
    install -m 0755 "${WORKDIR}/container-multiple-images.sh" "${D}${bindir}/container-multiple-images"
    install -m 0755 "${WORKDIR}/container-load.sh" "${D}${bindir}/container-load"
}
```
Integrating Docker archive within Yocto

• With docker-compose
  - Embedding docker-compose.yml in the rootfs
  - Docker-compose up instead of docker run command
Integrating Docker store within Yocto

- **Image recipe**
  - /var/lib/docker mounted as an overlayfs to be writable
  - /etc/docker mounted as a tmpfs

```bash
update fstab_image() {
  install -d "${IMAGE_ROOTFS}/${datadir}/docker-store"
  cat >> "${IMAGE_ROOTFS}/${sysconfdir}/fstab" <<EOF
  tmpfs  ${datadir}/docker-store  tmpfs defaults 0 0
  overlay  /var/lib/docker overlay noauto,rw,relatime,lowerdir=/var/lib/docker,upperdir=${datadir}/docker-store/docker,workdir=${datadir}/docker-store/.docker,x-systemd.requires,mounts-for=${datadir}/docker-store
/$(tmp/docker-data) /etc/docker none noauto,bind 0 2
EOF
}
ROOTFS_POSTPROCESS_COMMAND = "${@bb.utils.contains("IMAGE_INSTALL","container-image", "update_fstab_image;", ",", d)}"
ROOTFS_POSTPROCESS_COMMAND = "${@bb.utils.contains("IMAGE_INSTALL","container-archive","update_fstab_archive;", ",", d)}"
```
Integrating Docker store within Yocto

- Container-image.bb recipe
- Kill Docker daemon
- Clean Docker store
- Start Docker daemon
- Pull Docker images
- Clean tmp directories
- Kill Docker daemon

```bash
do_pull_image() {
    [ -f "${WORKDIR}/${MANIFEST}" ] || bbfatal "${MANIFEST} does not exist"
    [ -n "$(pidof dockerd)" ] && sudo kill "$(pidof dockerd)" && sleep 5
    [ -d "${DOCKER_STORE}" ] && sudo rm -rf "${DOCKER_STORE}"/*

    # Start the dockerd daemon with the driver vfs in order to store the # container layers into vfs layers. The default storage is overlay # but it will not work on the target system as /var/lib/docker is # mounted as an overlay and overlay storage driver is not compatible # with overlayfs.
    sudo /usr/bin/dockerd --storage-driver vfs --data-root "${DOCKER_STORE}" &

    # Wait a little before pulling to let the daemon be ready.
    sleep 5

    if ! sudo docker info; then
        bbfatal "Error launching docker daemon"
    fi

    local name version tag
    while read -r name version tag _; do
        if ! sudo docker pull "${name}:${version}"; then
            bbfatal "Error pulling ${name}"
        fi
    done < "${WORKDIR}/${MANIFEST}"
    sudo chown -R "$USER" "${DOCKER_STORE}"

    # Clean temporary folders in the docker store.
    rm -rf "${DOCKER_STORE}/runtimes"
    rm -rf "${DOCKER_STORE}/tmp"

    # Kill dockerd daemon after use.
    sudo kill "$(pidof dockerd)"
}```
Integrating Docker store within Yocto

- Container-image.bb recipe

```bash
do_install()
{
    install -d "${D}${systemd_unitdir}/system"
    install -m 0644 "${WORKDIR}/container-image.service" "${D}${systemd_unitdir}/system/"

    install -d "${D}${bindir}"      
    install -m 0755 "${WORKDIR}/container-image.sh" "${D}${bindir}/container-image"

    install -d "${D}${datadir}/container-images"  
    install -m 0400 "${WORKDIR}/${MANIFEST}" "${D}${datadir}/container-images/"

    install -d "${D}${localstatedir}/lib/docker"
    cp -R "${DOCKER_STORE}"/* "${D}${localstatedir}/lib/docker/"
}
```

do_pull_image[nostamp] = "1"
do_package_qa[noexec] = "1"
INSANE_SKIP_${PN}_append = "already-stripped"
EXCLUDE_FROM_SHLIBS = "1"
Integrating Docker store within Yocto

• **During boot time**
  1) Mount the Docker store as an overlayfs on a writable partition.
  2) Launch the Docker daemon.
  3) Run the container image.
Integrating Docker store within Yocto

- **During boot time**

```
[Unit]
Description=Load and start container image at boot
After=mount-noauto.service docker.service
Requires=mount-noauto.service docker.service

[Service]
Type=simple
 RemainAfterExit=yes
 ExecStart=/usr/bin/container-image start
 ExecStop=/usr/bin/container-image stop
 Restart=on-failure

[Install]
 WantedBy=multi-user.target
```
## Using a container engine in a Docker container

<table>
<thead>
<tr>
<th>Container engine</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Podman-in-Docker</td>
<td>- standalone process</td>
<td>- we had problems running Podman-in-Docker with newer versions than 1.5.1</td>
</tr>
<tr>
<td></td>
<td>- use of overlay storage driver on large container images</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- rootless</td>
<td></td>
</tr>
<tr>
<td>Docker-in-Docker</td>
<td>- no need to convert the container image</td>
<td>- start the daemon before pulling the image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- sudo privileges to run the Docker daemon</td>
</tr>
</tbody>
</table>
# Integrating Docker containers within Yocto

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Container engines</th>
</tr>
</thead>
</table>
| Integrating Docker image archive in the root filesystem | - easier to track the pulled image,  
- easier integration in Yocto | - longer boot time | • Podman-in-Docker  
• Docker-in-Docker |
| Integrating Docker store in the root filesystem | - faster boot time,  
- ensure the image integrity included in the root filesystem during boot and runtime,  
- minimize writing operation in the writable partition. | - start the daemon before pulling the image  
- sudo privileges to run the Docker daemon | • Docker-in-Docker |
Conclusion

• Blog articles on Savoir-faire Linux blog

• Meta-embedded-containers on GitHub (under MIT license)
  – https://github.com/savoirfairelinux/meta-embedded-containers

• Future works:
  – Docker and Podman as native tools ?
  – Container license compliance ?

• Two approaches made from the customer’s requirements
Thanks for your time
Me, Myself and I

• Sébastien LE STUM
  • Embedded engineer and Director at Savoir-faire Linux
    • Montreal office => Product engineering team
  • Working on solving our customers’ problems
    • Embedded products based on Linux
    • Building firmwares using Buildroot / Yocto / etc...
    • Design and implement feature-rich userspace applications
  • Cybersec background and Linux enthusiast
    • Past contributions to tpm2-tools
    • Developing stuff in Rust (Rust rocks! :-))