Introducing RunX

- A new OCI-compatible containers runtime to start containers as Xen VMs
- Written for Embedded
  - Very simple
  - Minimal overhead
  - Real-Time support
  - Accelerators support
  - Secure by Default
- New project started under the Linux Foundation Edge (LF-Edge) umbrella
  - Early collaboration with Zededa
  - Permissive license (Apache v2)
  - Open to contributions from the start
  - All development using a public mailing list: https://lists.lfedge.org/g/eve-runx
Introducing RunX

Container Orchestration Framework (e.g. Kubernetes)

Container

containerd

containerd “shim”

RunX

1

xl create

2

Linux

Xen

VM

Container

ramdisk

kernel
RunX: implementation choices

- Easy to Build: minimal build dependencies
  - gcc, make, go

- Easy to Run: minimal runtime dependencies
  - (in addition to Xen,) bash, jq, socat, daemonize

- No in-guest agents: minimal runtime overhead
  - Provides a minimal Linux kernel and Busybox-based ramdisk for booting regular containers as VMs
  - Pristine container environment

- Tiny Micro-VMs optimized for embedded
  - A minimal environment
  - No device emulation
  - No in-guest firmware or bootloaders

- OCI Runtime Spec compliant
  - Developed together with ContainerD
  - Should work with any container engines
RunX (Cross)Build

- Cross-build requirements:
  - cross-compilation toolchain
    - e.g. Linaro: https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu
  - golang compiler (soon to be removed)
    - distro golang package expected to work

```bash
$ export ARCH=aarch64
$ export GOROOT=/usr/lib/go-1.10
$ export CROSS_COMPILE=/path/to/aarch64-linux-gnu-
$ ./build.sh
```
RunX Runtime

- Copy runX and /usr/share/runX to target
- Enable it in containerd’s config.toml

```
[plugins.linux]
  runtime="/usr/sbin/runX"
```
Yocto + RunX

- meta-virtualization provides the recipes for RunX, Xen and supporting components (containerd, etc)

- Why use Yocto / OE to build / deploy RunX?
  - Leverage the Yocto / OE core values
  - Transition path from development to production
  - Active community and integration with BSPs (e.g. meta-xilinx)
  - Multiconfig builds
    - Build host + guests + containers + firmware in a single platform

- Note: development and build directly with upstream projects is always possible
Yocto + RunX: Simplified build

- Use master branches (there are no stable/released variants yet)

```
$ git clone -b master http://git.yoctoproject.org/git/poky
$ git clone -b master http://git.openembedded.org/meta-openembedded
$ git clone -b master https://git.yoctoproject.org/git/meta
$ git clone -b master https://github.com/Xilinx/meta
$ . ./oe-init-build-env zcu102-zynqmp
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-oe)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-filesystems)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-python)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-openembedded/meta-networking)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-virtualization)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-xilinx/meta-xilinx-bsp)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-xilinx/meta-xilinx-contrib)
$ bitbake-layers add-layer $(readlink -f $PWD/../meta-xilinx/meta-xilinx-standalone)
```
Yocto + RunX: Simplified setup

- Local build setup (will eventually be in a xen distro config)

```bash
$ cat <<EOF >> conf/local.conf
MACHINE ??= "zcu102-zynqmp"
DISTRO = "poky"
BBMULTICONFIG ?= "pmu"
do_image[mcdepends] = "multiconfig::pmu:pmu-firmware:do_deploy"

IMAGE_FSTYPES += "tar.gz cpio.gz.u-boot jffs2"
DISTRO_FEATURES_append=" xen virtualization vmsep"
IMAGE_INSTALL_append = " busybox xen-tools zlib-dev runx"
IMAGE_INSTALL_append += " virtual/containerd virtual/runc"
ASSUME_PROVIDED += "iasl-native"
PACKAGECONFIG_remove_pn-xen += " sdl"
PREFERRED_PROVIDER_awaiter-native = "xilinx-awaiter-native"
PREFERRED_PROVIDER_awaiter-qemu-native = "nativesdk-qemu-xilinx"
BUILDHISTORY_FEATURES ?= "image package sdk"
QB_DEFAULT_KERNEL="none"
QB_MEM = "-m 4096"
EOF

$ cat << EOF > conf/multiconfig/pmu.conf
MACHINE="microblaze-pmu"
DISTRO="xilinx-standalone"
TMPDIR="${TOPDIR}/pmutmp"
EOF
```
Yocto + RunX: build steps

Note: not all changes are merged upstream, but will be shortly

```bash
# download and extract pmu files (optional: only if not using multiconfig):

# copy files to deploy dir:
$ cp pmufw.elf build/tmp/deploy/images/zcu102-zynqmp/pmu-zcu102-zynqmp.bin
$ cp pmufw.elf build/tmp/deploy/images/zcu102-zynqmp/pmu-zcu102-zynqmp.elf
$ cp pmu-rom.elf $BUILDDIR/tmp/deploy/images/zcu102-zynqmp/pmu-rom.elf
$ cp system.dtb $BUILDDIR/tmp/deploy/images/zcu102-zynqmp/system.dtb

# build the image(s)
$ bitbake core-image-minimal
$ bitbake xen-image-minimal
```
Yocto + RunX: runtime steps

- Note: some on target configuration may be required (and will be automated in the future)

```
# core-image-minimal as a sanity test. Works out of deployed artifacts by default:

$ runqemu core-image-minimal slirp nographic

# xen-minimal: requires manual u-boot config, or boot.scr support from image builder
(https://gitlab.com/ViryaOS/imagebuilder)

$ runqemu xen-image-minimal nographic slirp
```
RunX: Traditional Containers

- **Container Orchestration Framework** (e.g. Kubernetes)
- **Container**
- **containerd**
- **containerd “shim”**
- **Linux**
- **Xen**
- **RunX**
  - `xl create`

- VM
  - **Container**
  - **ramdisk**
  - **kernel**
RunX: Containers with a Kernel

OCI Image Spec Extensions:
- KERNEL
- RAMDISK

Container Orchestration Framework (e.g. Kubernetes)

Tarball w/ kernel, ramdisk, & rootfs

containerd
containerd “shim”

RunX

xl create

Linux

Xen
RunX: Baremetal and RTOS Containers

Container Orchestration Framework (e.g. Kubernetes)

RTOS packaged as a container

containerd

containerd “shim”

1

RunX

2

xl create

RTOS

VM

Linux

Xen

© Copyright 2020 Xilinx
RunX: Containers with a Kernel

- Support containers that come with their own Kernel and/or Ramdisk
  - a specific version of the Linux kernel
  - a specific kernel configuration
  - LinuxRT

- Non-Linux OSes
  - RTOSes
  - Baremetal applications
  - VxWorks

- Kernel and Ramdisk are advertised using new OCI Image flags
  - TBD; currently Implemented using Environmental Variables
    - RUNX_KERNEL
    - RUNX_RAMDISK
  - Work with CNCF to standardize the new labels
RunX: Device Assignment

Container Orchestration Framework (e.g. Kubernetes)

Tarball w/ kernel, ramdisk, & rootfs

containerd

containerd “shim”

RunX + extra args

Linux

Xen

VM

Container

Provided ramdisk

Provided kernel

Heterogeneous HW Resource

© Copyright 2020 Xilinx
Device Assignment

- Device Assignment support via XLCONF
  - Appends configuration options to the xl config file
  - It can be used for anything from device assignment to changing vcpus and memory configurations
  - It can be used to set real-time configurations
  - It is set by the user/admin (not by the container)
Vision: Accelerators & FPGAs

Container Orchestration Framework (e.g. Kubernetes)

Tarball w/ kernel, ramdisk, & rootfs

Device Config

containerd

containerd “shim”

RunX
[extra args] or config.json

Linux

Xen

VM

OCI Image Spec Extensions:
- RESORUCE = DEVICE_DATA
- KERNEL
- RAMDISK

Provided ramdisk

Provided kernel

Heterogeneous HW Resource

© Copyright 2020 Xilinx
Vision: Accelerators & FPGAs

1. Containers come with their own accelerator's binaries and data
   - FPGA bitstreams
   - Co-Processor Kernels
   - AIE Kernels

2. ContainerD calls to a service to program the accelerators

3. RunX assigns the accelerator's resources to the VM
Demo
RunX RTOS and Device Assignment Demo

Baremetal Container with access to the physical TTC timer