International
OPEN ACCESS Journal
Of Modern Engineering Research (IJMER)

Container Based Solution for Embedded Linux Products

[1]Saminath, [2]Raghavendra prasad
[1]Saminath, Staff Engineer in Borqs Technologies
[2]Raghavendra prasad, Senior principal Staff Engineer in BorqsTechnologies

ABSTRACT:*The development of container-based Application is widespread across cloud-based solution and open stack connectivity. The containerized application is a stand-alone package which provides virtualization features and is isolated from host platform. A container package consist of Application binary, dependent libraries, configurations and associated resources. This miniature platform improves the portability of application and deployment to any remote system with minimal cost.Application portability in Embedded industry has many challenges, as the devices are running in remotely and upgraded images on-the-fly instead of standalone connected devices. The container separates OS Layer and Targeted Application; it helps to migrate only the application on the fly. Ease of porting avoids the action on hardware and software re-initializing process, which means it decreases impact on application transition.*

KEY WARDS:*LXC:LinuxContainer, RFS: Root File System, OTA :overTheAir*

Date of Submission: 18-07-2021 Date of acceptance: 03-08-2021

I. INTRODUCTION

Containers empower manufacturers to deploy their applications more frequently and independently with large scaling, without any hassle. Virtualization is playing a great role in the growth of cloud computing, and considered as one of the basis technologies for image migration and Application upgradation. Primarily there are two kinds of technologies which offers virtualization services, they are Hypervisor and Container-based solution. Both container and hypervisor are used to create an abstraction layer between Hardware and OS platform in order to effectively use the available resources to support on virtualization. Virtualization technology runs on multiple operating system simultaneously with distribution of single physical hardware system. Virtualization OS layer shares CPUs, Memories, Device I/O, Networks and other sub systems connected in Host system.

Hypervisor is a traditional way of creating virtual environment to communicate with the Hardware directly from OS layer. It permits many Operating systems to run on the same Hardware. The parallel communication allows multiple applications concurrently and improvises the system performance.

Alternative way of enabling virtualization is achieved by Container based solution. It provides light weight components to run on the HOST operating system to manage resources, constraint in mobilize platforms. Container achieves a better performance as compared with traditional virtual machine by using hypervisor-based virtualization. Container orchestration provides facilities for applications to scale widely and deploy over the air. Many cloud service providers like Amazon, Azure are extending their services to container components which benefits using cloud APIs.

Linux container (LXC) and Dockers are increasingly popular on its features and benefits in the recent decades. Docker is supported by multiple platforms like Windows, Linux and OSX. The Docker image is packed with target platform Application and dependent components like libraries and bins. Docker shares the Host environment to access the hardware resources. But Linux container has many components Control groups, Namespace and Network support, which offers an entirely isolated environment from Host system. It preserves network resource access, hardware device communication, Filesystem access and Memory management. It emulates the guest operating system and creates a self-contained environment to run an application. In this work, we are more focusing on LXC based container environment and various features supported by it.

II. LXC COMPONENTS

LXC requires various HOST operating-system components to deploy on it. Namespace mechanism emphasizes on data isolation and it makes an abstraction on resources accessed by any process. The Cgroup mechanism focuses more on performance isolation by limiting the number of resources. Networking component is bringing the networking feature to LXC container and allows access to internet inside the container. LXC component runs only on Linux host platform. All the configuration and features are needed by LXC to be enabled in the Kernel and User space layer. These features should be exposed to container Image layer in run time and applications can utilize it. Two major components are required to run the LXC on Host machine, Container Engine and Container Image. The details of the components are explained in Figure-1.

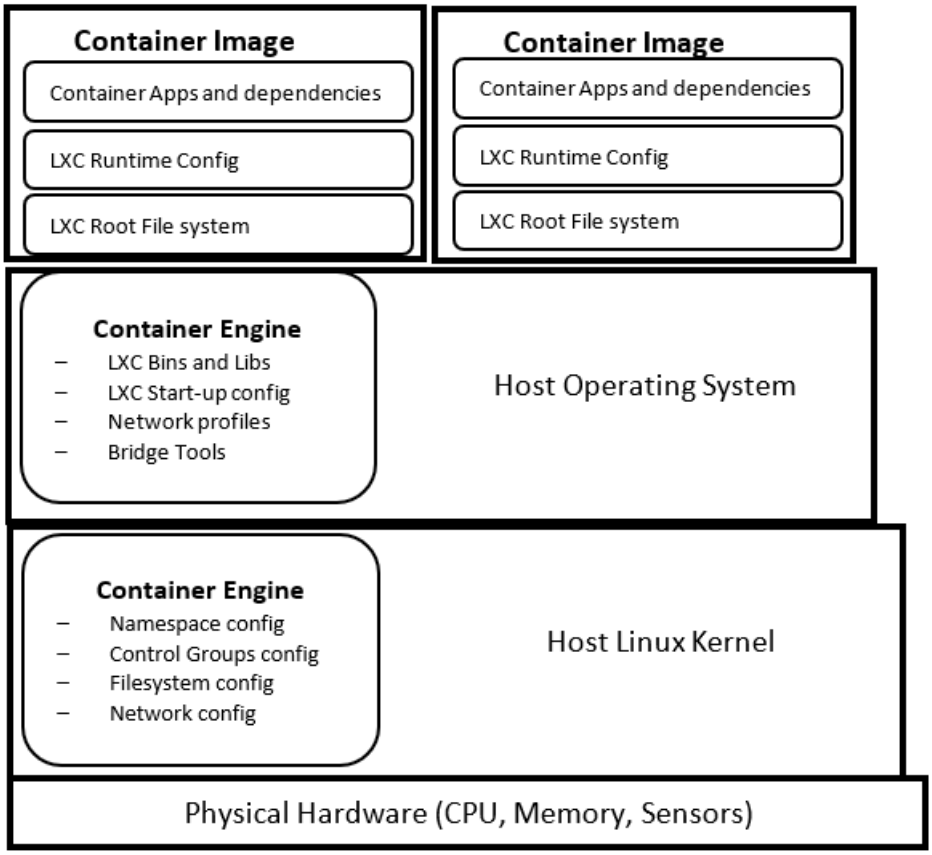

Figure -1: High level LXC container components

2.1 Container Engine

Container engine is the heart of LXC components and it provides life-cycle management of the containers. Container engine configuration is required in kernel and user space layer. Kernel modules limit the resources and isolates the CPU, memory, disk I/O and network's usage by multiple processes. LXC supporting filesystem need to be configured in kernel modules.

Another core component of LXC user-space tool is lxclib, the API library and set of binaries which is used to control containers. A set of templates is used to create container images in runtime. The executables and libraries should be compatible with host platform. LXC binaries use configurations and templates to start the system containers with specific features. Lxc-create utility is use to download lxc image from remote server or host machine. Lxc-start and lxc-stop are used to start and halt the container image. Similarly, lxc-attach utility is used to start the Application inside container, which is isolated from host system.

Templates contain set of APIs to download the container images and initialize the runtime environment of the specific container distribution. LXC configuration files define the behavior of containers like IO ports, networking and filesystem mounting. It brings various network support to container to connect with network interfaces. There are two types of configuration, system-wide configuration for global container features and container specific configuration for detailed container image settings. Configuration allows to amend the container capability. Mount option allows to access the host partition from inside container for file read/write operation.

2.2 Container Image

Container image is composed of root file systems (RFS) with minimal operating system components. There is an official opensource Linux distribution for container images like ubuntu, fedora, etc. Container image should be a light weight package, compatible with host Linux environment and has necessary runtime system tools with standalone executable.

Container image has unique identifier with hashing (sha-256) format, mostly a compressed content. The image container has two partitions, Rootfs and metadata. Rootfs is a minimal file system which primarily contains busybox utility, boot-up scripts, device nodes and container initialization environment. The configuration comprises specific things such as username or UID, working directory, and device ports that needs to be exposed. Metadata contains details about container image, version and compatibility info.

III. CONTAINER COMPONENTS

Chroot is a traditional way of changing the root directory of a process and its children. It is not very secure to start a new root directory and there is no isolation between the chroot environment and processes of the Host environment. Many secured components are added in the LXC engine to provide a secure way of launching container components as like Namespace, Cgroups and privileged container mounting.

Control groups (cgroups) is a Linux kernel feature that can be used to limit and monitor the resource used by every process. Cgroup is a resource controller, it controls the memory, cpu ,pid's and IO entity of the container. Each resource in the cgroup controllers is arranged in a hierarchical manner. High level resource limitation is applicable to child node of the controller in same cgroup.

Namespaces are significant element of the containerization to separating the processes from host. Currently, eight kinds of namespace support the container processes like IPC, cgroup, network, mount, PID, UTS, user, and time. These resources are identical and isolated instance and become available in the global environment.

CRIU (checkpoint and restore) feature helps to live migration of container on the fly, which launches a new dynamic application on the embedded product. It maintains the state of the container before stopping and restores in after migration is performed. CRIU organizes memory pre-copy and memory pre-post during container migration process.

IV. CONTAINER LIFE CYCLE MANAGEMENT

The container image mounts and creates a runtime environment for containerized Application. There are various steps to start the container and link with Host system. Two types of containers will be launched based on system protection. Privileged container and unprivileged container.

The unprivileged container is the safest way to start a container. The mapping of uid and gid of the container is remapped from outside container. Container process cannot be hacked from the external process. In contrast, privileged container is created by root user and accessed by host. The various stages of lxc life cycle are explained in Figure-2.

4.1 Container start

Container is initialized with resource components and it starts by lxc-create command. It uses templates as a parameter to load the container images and it launches the container with a unique ID. Configuration file is parsed and configured with scopes to act like networking, file system mounting. Lxc-ls to list all the running container info, process ID, networks and physical status of container.

4.2 Application start

Application inside the container will be started and access the resource configured by container. Lxc-start or lxc-console with application parameters starts the application inside the container. Multiple containers can be start and supported by Host. Lxc-info provides the memory, cpu and pid list of the specified container. Container Application will use the internet of host network interface or assign dedicated physical network interface to container, decided by the LXC configuration. The Application is able to access the Host file systems, Libraries and File read/write operations.

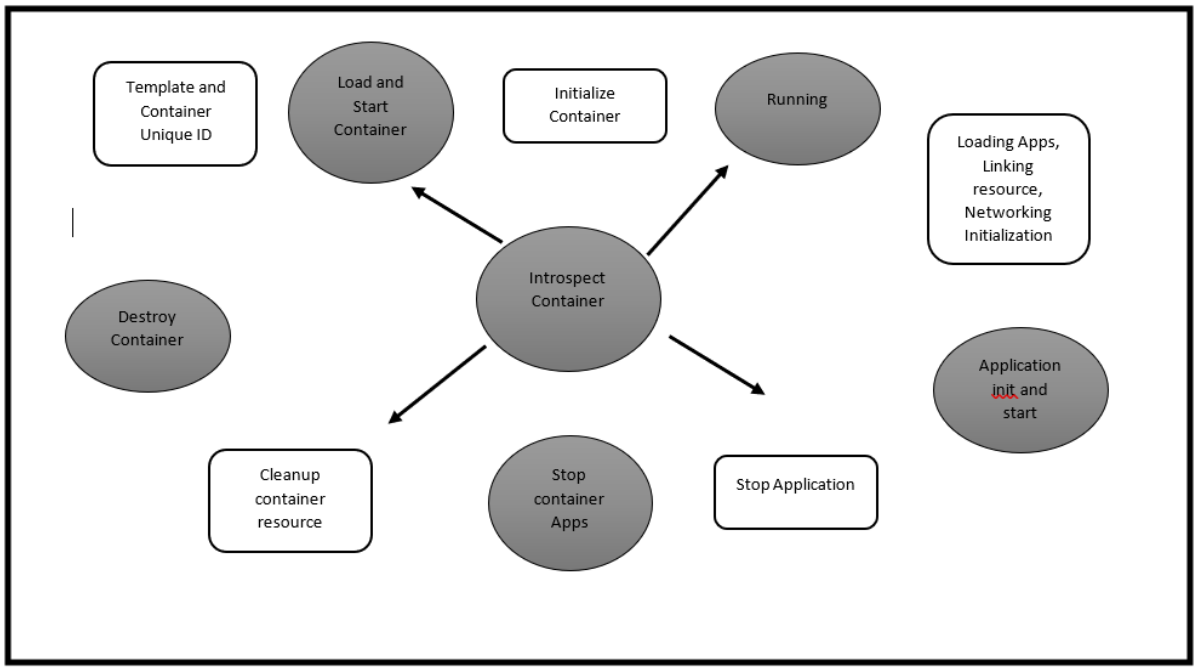

Figure -2: Life cycle management of LXC container

4.3 Application stops and store state.

The Application stopped and it cleans the state by using lxc utilities. Any running container is stop and removed from its state. This action releases the hardware resources and cpu assign to the process. During this time, push new Application inside container mounting place and start the Application. The older configuration, library and binaries will be initialized and start the Application. CRIU helps to maintain the previous container object state.

4.4 Container destroys.

The container data should be cleaned, and delete older contents and configurations. The device initiates the shutdown process or container is going to upgrade with new container images, then we need to do cleanup process. It calls the lxc-stop command to stop running the container. But the lxc configuration and image is still present in Host system. Lxc-destroy completely cleans up the data backed by container and it is removed permanently from HOST. The important lxc commands and usage with container process is described in Table-1.

LXC Commands	Description	LXC Commands	Description
Lxc-create	Creates a container from Image	lxc-info	Container details, like CPU usages, Memory usages, networking Type, IP Address.
lxc-checkconfig	Verifying the Container supporting components	Lxc-ls	List of running container
Lxc-config	Print the Container config	Lxc-attach	Attach to container shell
Lxc-start	Start the container	Lxc-autostart	Start/kill auto start container
Lxc-stop	Stop/halt container	Lxc-console	Launch container console

Table-1 : LXC command and Descriptions

V. DESIGN OF LIVE MIGRATION

Migration of an Application inside a container helps to improve the performance for Application upgrade with minimal impact. In general, during the migration process the energy loss on the device is higher. Application upgrade will stop current state of the device like network connectivity, device peripheral connection, memory clean up, CPU runs maximum clocks, board reinitialize. It will be huge activity from server to re-establish the end-to-end communication.

But in container base Application will reduce all above impact by restarting only initializing Container and inside the application, Host process remains to continue to run on the same state. Container saving impact of application migration in embedded devices. Lxc create will copy the root file system into configured template path (/var/lib/lxc/container/rootfs). Now the complete container image is un-compressed and placed in this filesystem path. When you want to start the container application, lxc-start command changes the state to

container to run and execute the application inside the container. Similarly, we can stop the application and clean the state of container.

The container images should be complained with Host system. Open source Busybox utilities and compiler toolchains able to create the minimal Root file system. Another way, we can use lxc-busybox template to create RFS for target device. Lxc-create passing with busybox template creates binaries, libraries, mount folders, network configuration are used for the target container to start up. The upgraded application will be replaced inside the template path and restarts the container. This will eliminate the unnecessary re-initializing process of system and rebooting all the process list. Embed container is created from any Linux host system. It generates busybox utilities to start a container image.

```
lxc-create -t lxc-busybox -n embed-container
lxc-start -n embed-container
```

VI. EXPERIMENTAL SET-UP

The below experiment set-up is to understand the Live migration of a container image in the embedded product. Container kernel configuration should be enabled on the embedded device, it can verify by lxc-checkconfig. The necessary container user space component is lxc-engine, it should run on the target device, Lxc-image component is Rootfs with target application and dependent component which needs to run on the container. The below table shows minimumlxc configuration required.

```
lxc.network.type = veth
lxc.network.link = bridge0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3f:3c:1e:fe
lxc.network.ipv4 = 192.168.225.2/24
lxc.network.ipv4.gateway = auto
lxc.rootfs = /var/lib/lxc/container-name/rootfs
lxc.haltsignal = SIGUSR1
lxc.rebootsignal = SIGTERM
lxc.utsname = test-con
lxc.tty = 1
lxc.pts = 1
```

Prime lxc user-space components are lxc binaries(lxc-xxx) and libraries (liblxc), templates (lxc-templates) and dependent components. The network virtualization layer helps to define the lxc network interfaces types like bridge, vlan, veth. Template scripts assist to container environment and lxc-busybox initializes with host system. Lxd initialization connects with cloud server using lxc networking configuration. Containers have a set of embedded Linux template and it runs by using the lxc launch command.

```
lxdinit
lxc launch container-name  -t container-template
```

Lxc-start command starts the container and directly connects to one of its terminals. Then we need to run the Application inside the console. Another possibility is to start the container as a daemon and connect it using console by using lxc-console. Lxc-execute solves two step operation into single command to start the container and it launches the Application too.

```
lxc start -n container-name
lxc console -n container-name
lxc execute -n container-name   Container-App
```

The final command is to stop and destroy the container, that is lxc-stop and lxc-destroy commands. These executables are do clean up the backing store by container and it deletes all the container configuration.

```
lxc stop -n container-name
lxc destroy -n container-name
```

The valid Container image is placed in Cloud sever to upgrade the image remotely. This Container image is downloaded into embedded device by using third party OTA client. Now, running container need to stop and

clean the resource allocation by the container. After this, lxc-create need to point with newer container image and start the container by default. We can start the container by using lxc-create command passing the compressed container image.

```
lxc-create -t container-template -n new-container-name -- --imagetar embed-container.tar.gz
```

This command extracts the container image and is copied into path (/var/lib/lxc/new-container-name /rootfs). The busybox is mounted in rootfs location. It has binaries, libraries and the lxc application component. It helps to start the container and launch Application in existing steps. Figure-3 explains the Container commands with runtime sequences. The System's up and down time do the lxc command with the initialization and destroy the operations. OTA update invokes, create the container instance and runs the Application without initialization. With the benefit of LXC, the complexity involves in application migration by VM or Bare meta server is avoided.

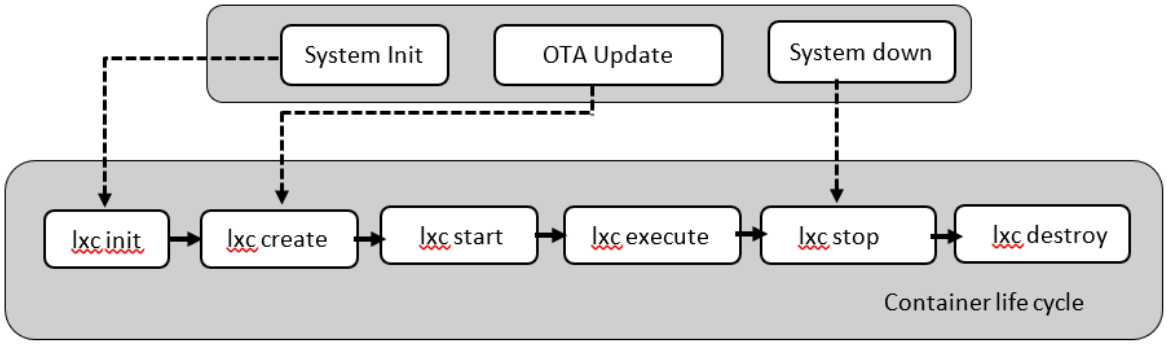

Figure -3: Container Runtime flows

VII. PERFORMANCE AND EFFICIENCY

The major challenge in lxd based container development is not to fit for Embedded product. The GO based programming is heavier for light weight operating systems and minimal foot print devices. On the fly, starved of lxd component is highly challenging during the container upgrade. The proposed components reduce memory used by lxc components in the target device. The auto upgraded related binaries, libraries and configurations are removed. It will do frequent checks on Cloud server for container image and impact processor cycles. Post download process and restarting of container time also reduces. The extraction of container file, mounting container images and starts Container Application are under control.

VIII. CONCLUSION

The major goal of this investigation to compare the performance of container-based virtualization during the process of live migration. The verified procedure indicates that LXC component is better in overall container performance. LXC has patched up its system and has the best result in terms of system performance, its proven that LXC can increase overall system performance effectively. To conclude, container downtime and Total Migration Time are greatly reduced in Live migration and improve overall performance. It allows multiple smaller Application ease of deployment in the Software development. The factors affecting such as network or IPC latencies and configuration could be taken into account when considering the overhead of the containers

REFERENCES

[1]. HarriManninen , 2020, "Evaluation of Container Technologies for an Embedded Linux Device", Thesis to the Degree of Master of Science in Technology at Espoo Institute.
[2]. Pavan Sutha Varma Indukuri , 2016, "Performance comparison of Linux containers (LXC) and OpenVZ during live migration", Thesis to the Faculty of Computing at Blekinge Institute of Technology.
[3]. Jesse Hertz, 2016, "Abusing Privileged and Unprivileged Linux Containers", An NCC Group Publication.
[4]. McCarty. 2020, "A Practical Introduction to Container Terminology." developers.redhat.com. https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction
[5]. Xin Lin, LingguangLei, 2018, "A Measurement Study on Linux Container Security: Attacks and Countermeasures", [https://doi.org/10.1145/3274694.3274720] ACSAC Association for Computing Machinery.
[6]. AdindaRiztia Putri, RendyMunadi, RidhaMuldinaNegara , 2020, "Performance analysis of multi services on container Docker LXC, and LXD", Bulletin of Electrical Engineering and Informatics.
[7]. Scott Murray, 2018, "Building Container Images with OpenEmbedded and the Yocto Project", Konsulko Group.

[8]. Patrick Goemaere, Rajat Ghai, 2018, "Building Container Images with OpenEmbedded and the Yocto Project", A Technical paper prepared for SCTE ISBE and NCTA..
[9]. Lammi, T, 2018 "Feasibility of application containers in embedded real-time Linux,"M.S thesis, Dept. Elect. Eng., Tampere Univ. Technol., Tampere.
[10]. Linux containers. "What's LXC?" linuxcontainers.org. https://linuxcontainers.org/lxc/introduction/ (accessed Jan. 29, 2020).

Saminath, et. al. "Container Based Solution for Embedded Linux Products ." *International Journal of Modern Engineering Research (IJMER)*, vol. 11(07), 2021, pp 27-33.