ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

Build Minimal Docker Container Using Golang

[1] Biradar Sangam.M, [2] R.Shekhar
[1][2] Department of Computer Science & Engineering, Alliance University, Bangalore, INDIA

Abstract: **- Docker container is developed in Go Programming language. Container internal is encapsulated from Linux kernel feature i.e. namespaces and cgroup. Namespace isolation allows a server to isolate a process so that can't see certain portion of overall system. Control groups as name its control the process, memory and CPU. It will track used memory. Container shares the same host kernel but they have their own virtualized network adapter and file system. Container allow for efficient application deployment and management. Go language is provide system programming support to the container. Container is the lightweight and portable encapsulation of an environment in which to run application. Container is created from images. It has all binaries and dependencies need to run application. Containerization is the new way to build, ship and deploy applications.**

Keywords**: Go,Container,namespaces,cgroups.**

I. INTRODUCTION

Go is the compiled and statically typed programming language. It provides garbage collection, type safety, dynamic-typing capability, many advanced built-in types such as variable arrays and key-value maps. Go support for environment adopting patterns similar to dynamic language, It is built in the concurrency support: channel, Routine and lightweight processes. And we are using Docker technologies its platform for the developer and sysadmins to build containerized application and build, ship and run distributed application anywhere. Docker container enables apps to be quickly assembled from components and the friction between development, QA and production environment. As a result, it can ship faster and run the same app, unchanged, on laptop, data center, VMs and any cloud. Linux containers provide a virtualized environment for processes on Linux servers with less overhead than virtual machines. System administrators can deploy containers quickly while using fewer server resources. Multiple technologies come together to make containers possible on Linux, but the majority of the work is centered on a concept called namespace isolation. Namespace isolation allows a server to isolate a process so that it cannot see certain portions of the overall system. For example, process ID (PID) isolation can be used to make a process think that it is the only process running on the server. It would have no access to know that other processes exist, and it would not have the ability to send signals to any of those processes. In addition, a container could run its own in it process as PID 1 while the host system sees that process as an entirely different PID.[4]

Linux container provides the capability of implementing all the processes in an isolated fashion by using kernel namespaces and control groups. Each Linux container can view only its filesystem and process space. Each container is provided an isolated environment with its own computing resources and networking stack. The main advantage of Linux containers is that it doesn't need to run a full Linux OS. Routed-based and bridge-based namespaces are two configurations of LXC which allows communicating with outside world. LXC, Docker, Warden are some of the management tools available for Linux Container. [5]A Linux container is a set of processes that are isolated from the rest of the machine. A container can encapsulate any application dependency. For example, if a website relies on a particular version of the PHP scripting language, the container can encapsulate that version. As a result, multiple versions of the same scripting language can co-exist in the same environment - without the administrative overhead of a complete software stack, including the OS kernel. Containerized applications perform about as well as applications deployed on bare metal.

Today's applications are more complex, and yet they must be developed more quickly. These trends increase demands on infrastructure, IT teams, and processes. IT departments are struggling to find ways to:

- Lower costs by helping teams do more with the same staff size
- Respond more quickly to new business requirements
- Keep systems and data secure
- Adopt innovative development and hosting methods using existing infrastructure

Linux kernel namespaces are the fundamental building block of containers on Linux. The idea of namespaces as a logical construct to deal with scope or segmentation is a common idea in computer science.57 For Operating Systems, Plan 9 introduced 58 in 1992 the idea of namespaces, among other interesting concepts such as network or union filesystems and many other computing

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

advancements outside containers. In Linux, kernel namespaces form a foundational isolation layer that allows for the implementation of Linux containers by creating different user land views. The Namespaces in Operation series on Linux Weekly News by Michael Kerrisk offers a great overview and explores each namespace. The Resource Management: Linux kernel namespaces and cgroups presentation by Rami Rosen offers a long and in-depth exploration of namespaces and cgroups. Readers interested in additional background and information should start with these resources. [3] Control groups: control group let you implement mattering and limiting the resources use by processor. In control groups, each subsystem has own hierarchy it like tree with node and each process belong to node.

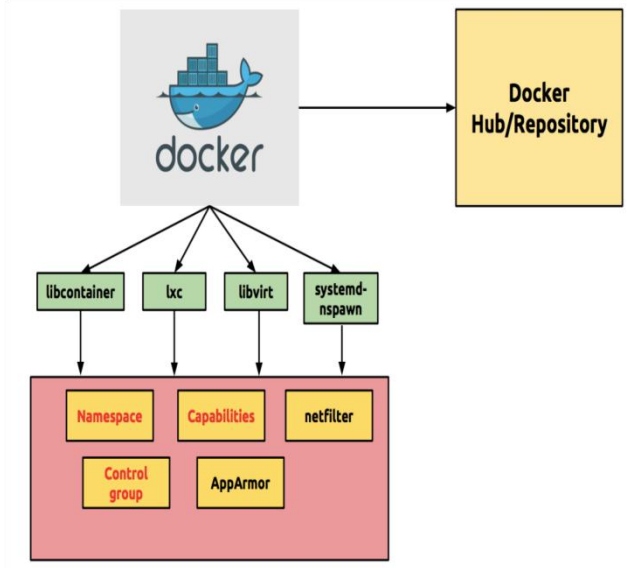

Fig.1.0 Docker Internal

II. METHODOLOGY

A. NAMESPACES

• IPC NAMESPACES: System V IPC objects and POSIX message queues can utilize their own namespace. As with other namespaces, CLONE_NEWIPC provides a method for creating objects in an IPC namespace which are visible to all other processes that are members of that namespace, but are not visible to processes in other IPC namespaces. This is typically used for shared memory segments. This isolation helps from some IPC related attacks and Denial of Service scenarios.[3]

• MOUNT NAMESPACES: The mount namespace via CLONE_NEWNS is the oldest and only namespace introduced in the 2.4 kernel. The mount namespace provides a process, or group there of treated as container, with a specific view of the system's mounted filesystems. This view can range from mount paths, physical or network drives, or advanced features such as union filesystems, bind mounts, or overlay filesystems (where some section of the host filesystem is directly accessible, yet other reads or writes stop at container bound- aries.) The mount namespace can also indirectly secure other namespaces by restricting access to the hosts' mounted instance of /proc, which would violate the PID namespace constraints.[3]

• PID NAMESPACES: (CLONE_NEWPID, Linux 2.6.24) isolate the process ID number space. In other words, processes in different PID namespaces can have the same PID. One of the main benefits of PID namespaces is that containers can be migrated between hosts while keeping the same process IDs for the processes inside the container. PID namespaces also allow each container to have its own init (PID 1), the "ancestor of all processes" that manages various system initialization tasks and reaps orphaned child processes when they terminate. From the point of view of a particular PID namespace instance, a process has two PIDs: the PID inside the namespace and the PID outside the namespace on the host system. PID namespaces can be nested: a process will have one PID for each of the layers of the hierarchy starting from the PID namespace in which it resides through to the root PID namespace. A process can see (e.g., view via /proc/PID and send signals with kill()) only processes contained in its own PID namespace and the namespaces nested below that PID namespace.[2]

```
root@autb:~# go run main.go run echo sangam
Running [echo sangam]
sangam
root@autb:~# go run main.go run  ps
Running [ps]
  PID TTY          TIME CMD
 2731 pts/1    00:00:00 go
 2744 pts/1    00:00:00 main
 2747 pts/1    00:00:00 bash
 5351 pts/1    00:00:00 bash
11229 pts/1    00:00:00 go
11242 pts/1    00:00:00 main
11246 pts/1    00:00:00 exe
11249 pts/1    00:00:00 ps
```

Fig.1.1 PID

• NETWORKING NAMESPACES:(CLONE_NEWNET, started in Linux 2.4.19 2.6.24 and largely completed by about Linux 2.6.29) provide isolation of the system resources associated with networking. Thus, each network namespace has its own network devices, IP addresses, IP routing tables, /proc/net directory, port numbers, and so on. Network namespaces make containers useful from a networking perspective: each container can have its own

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

(virtual) network device and its own applications that bind to the per-namespace port number space; suitable routing rules in the host system can direct network packets to the network device associated with a specific container. Thus, for example, it is possible to have multiple containerized web servers on the same host system, with each server bound to port 80 in its (per-container) network namespace. [2]

• UTS NAMESPACES:(CLONE_NEWUTS, Linux 2.6.19) isolate two system identifiers nodename and domainname returned by the uname() system call; the names are set using the sethostname() and setdomainname() system calls. In the context of containers, the UTS namespaces feature allows each container to have its own hostname and NIS domain name. This can be useful for initialization and configuration scripts that tailor their actions based on these names. The term "UTS" derives from the name of the structure passed to the uname() system call: struct utsname. The name of that structure in turn derives from "UNIX Time-sharing System".[2]

Fig1.2 Hostname

• USER NAMESPACE: (CLONE_NEWUSER, started in Linux 2.6.23 and completed in Linux 3.8) isolate the user and group ID number spaces. In other words, a process's user and group IDs can be different inside and outside a user namespace. The most interesting case here is that a process can have a normal unprivileged user ID outside a user namespace while at the same time having a user ID of 0 inside the namespace. This means that the process has full root privileges for operations inside the user namespace, but is unprivileged for operations outside the namespace.[2]

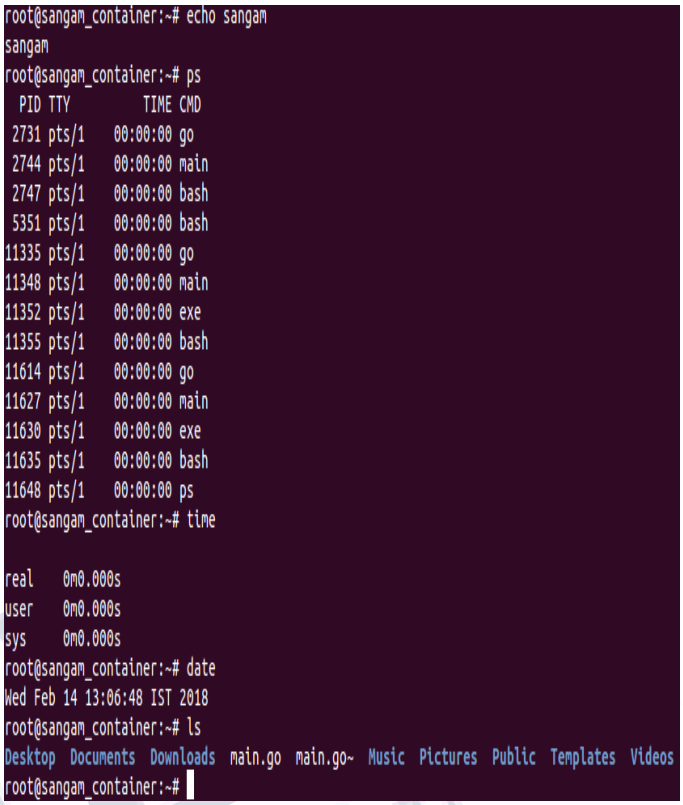

Fig.1.3 /bin/bash

B. Cgroups

Control Groups (cgroups) are a mechanism for applying hardware resource limits and access controls to a process or collection of processes. The cgroup mechanism and the related subsystems provide a tree-based hierarchical, inheritable and optionally nested mechanism of resource control. To put it simply, cgroups isolate and limit a given resource over a collection of processes to control performance or security. Cgroups can generally be thought of as implementing traditional ulimits/rlimits, but now operating across groups of tasks or users. A new, more powerful and more easily-configured alternative to ulimits/rlimits. To silence the naysayers and doubt over code bloat or added complexity.[3] The Cgroups (control groups) subsystem is a Resource Management and Resource Accounting / Tracking solution, providing a generic process grouping framework.

Its handles resources such as memory, CPU, networking and more. Memory Cgroup: accounting We will count how much memory used by each process we will track every single memory page.

Memory Cgroups: limits

Each group can have its own limits there are two type of limits soft limits and hard limit. Limits can set different

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

kind of memory i.e. physical memory, kernel memory and total memory.

C. Layered Filesystems

Namespaces and CGroups are the isolation and resource sharing sides of containerisation. They're the big metal sides and the security guard at the dock. Layered Filesystems are how we can efficiently move whole machine images around: they're why the ship floats instead of sinks. At a basic level, layered filesystems amount to optimising the call to create a copy of the root filesystem for each container.[1]

D. Security responsibility

Developers appreciate containers because they can package their application, test it alongside its libraries, and verify that it will work in production. Operations teams appreciate containers because they get the applications in a cohesive package along with their dependencies and configurations. [4]

III. IMPLEMNETATION

We are using following packages and method of built in go functions:

- fmt
- io/util
- os
- os/exec
- path/filepath
- syscall

fmt- package fmt implements formatted I/O with function analogous to c's printf and scanf.[7]

io/util: package ioutil implements some I/O utility functions.

OS: package os provides a platform-independent interface to operating system functionality. The design is unix-like, although the error handling is Go-like; failing calls return calls return values of type error rather than error numbers. Often, more information is available within the error. The os interface is intended to be uniform across all operating system. features not generally available appear in the system-specific package syscall.[6]

OS/EXEC: package exec runs external commands. It wraps on os. Start Process to make it easier to remap stdin and stdout, connet I/O with pipes, and do other adjustments.[9]

Unlike the "system" library call from c and other language, the OS/exec package intentionally does not invoke the system shell and does not expand any glob patterns or handler other expansions, pipelines, or redirections typically done by shell. The package behaves more like C's "exec" family of functions. To expand glob patterns, either call the shell directly, taking care to escape any dangerous input, or use the path/filepath package's Glob function. To expand enivironment variable, use pacakage os's ExpandEnv.[9]

Path/filepath: Package filepath implement utility routines for manipulating filename paths in way compatible with the target operating system-defined file paths.

The filepath package uses either forword slashes or backslashes, depending on the operating system. To process paths such as URLs that always use forward slashes regardless of the operating system.[10]

Syscall: package syscall contains an interface to the low-level operating system primitives. The details vary depending on the underlying system, and by default, go doc with display the syscall documentation for the current system. if you want godoc to display syscall documentation for another system.[11]

Requirements:

Operating system: FreeBSD 9.3 and later, Linux 2.6 and later, mac os 10.8 and later, windows XP sp2 and later

Tools: VSCODE editor or any other alternative editor

Installation of golang on Linux:

https://golang.org/dl/ the installer as per the operating system

useful command to run the project:

go run < one or more as per program filename > to run the go program

go run < filename> run bin/bash to check container is working or not

```
root@autb:~# go run main.go run /bin/bash
Running [/bin/bash]
root@autb:~# hostname
autb
root@autb:~# gedit main.go
root@autb:~# go run main.go run /bin/bash
Running [/bin/bash]
root@sangam_container:~# ls
Desktop  Documents  Downloads  main.go  main.go~  Music  Pictures  Public  Templates  Videos
root@sangam_container:~# ls /proc
1      11383  1261  1472  1742  22499  27862  40    768    buddyinfo    kmsg          sys
10     11396  12613  1489  1746  22501  27863  41    771    bus          kpagecount    sysrq-trigger
1015   11399  1270  15    1749  23     27864  42    772    cgroups      kpageflags    sysvipc
1026   1140   1272  157   1774  2317   27891  43    798    cmdline      latency_stats  timer_list
10328  11403  1280  158   18    2320   28     4370  8      consoles     loadavg       timer_stats
1033   11431  12897  159   1863  2321   288    4378  80     cpuinfo      locks         tty
1052   11467  12903  16    1911  2387   29     4379  81     crypto       mdstat        uptime
1057   1151   12906  1640  1939  24     3      44    82     devices      meminfo       version
1067   1171   12909  1642  2     2449   30     45    83     diskstats    misc          version_signature
10929  1186   12917  1643  20    25     3046   46    849    dma          modules       vmallocinfo
10985  1190   12919  1644  2025  26     31     496   853    driver       mounts        vmstat
11     1193   12921  1645  21    2682   3157   5     859    execdomains  mtrr          zoneinfo
11067  1194   13     1647  2114  27     3161   5351  860    fb           net
11150  1196   1338   1657  2131  27268  32     577   864    filesystems  pagetypeinfo
11169  12     134    1686  2135  2731   33     578   9      fs           partitions
11195  1200   135    1690  2151  2744   332    58    924    interrupts   sched_debug
11311  1204   136    1695  2161  2747   34     61    935    iomem        schedstat
11334  1209   137    17    2163  27854  35     655   940    ioports      scsi
11335  1212   138    1711  2171  27855  36     683   942    ipmi         self
11348  1213   139    1720  2188  27856  37     684   949    irq          slabinfo
11352  1215   140    1725  22    27859  372    7     957    kallsyms     softirqs
11355  1220   141    1733  2221  27860  38     702   acpi   kcore        stat
1136   1236   1470   1738  2228  27861  4      704   asound  key-users   swaps
```

Fig.1.4 Process list

IV. CONCUSION

we are successfully executed the minimal container in golang and how they made it from namespaces, Cgroups, filesystem etc.we developed container to run application on own environments. the major role of container build it once and run on any platform and anywhere. Docker technology is the future of the software development.so we created minimal container in golang. Container is like package we can put all dependencies or needed resources so its help in software production.

REFERENCES

[1] Build a container golang https://www.infoq.com /articles /build-a-container-golang

[2] Michael Kerrisk. Namespaces in operation, part 1: the namespaces API January 4, 2013

[3] Aaron Grattafiori.Understanding and Hardening Linux Containers June 29, 2016 – Version 1.1

[4] Major Hayden,Securing Linux Containers GIAC (GCUX) Gold Certification, July 26, 2015

[5] Kotikalapudi sai venkat naresh, comparing live migration linux containers and kernel virtual machine, Feb 17

[6] OS package golag https://golang.org /pkg /os /

[7] Fmt package golang https://golang.org/pkg/fmt/

[8] IO utiil package https://golang.org /pkg /io /ioutil/

[9] Exec package golang https://golang.org/pkg/os/exec/

[10] Filepath package golang https://golang.org /pkg /path /filepath/

[11] System call package golang https://golang.org /pkg /syscall /