Digging for Bedrock: How Linux Containers Were Shaped

95,000,000 BC to today

75TH CONGRESS
1st Session

SENATE

DOCUMENT
No. 80

COLORADO-BIG THOMPSON PROJECT

SYNOPSIS OF REPORT

ON

COLORADO-BIG THOMPSON PROJECT, PLAN OF DEVELOPMENT AND COST ESTIMATE PREPARED BY THE BUREAU OF RECLAMATION, DEPARTMENT OF THE INTERIOR

PRESENTED BY MR. ADAMS

JUNE 15, 1937—Ordered to be printed without illustrations

UNITED STATES
GOVERNMENT PRINTING OFFICE
WASHINGTON : 1937

HORSETOOTH RESERVOIR

The proposed Horsetooth Reservoir will occupy a valley 6 miles long and from one-quarter to three-quarters miles wide, extending in a north-south direction, formed by the erosion of soft red beds of Lykens formation between harder ridges of Lyons on the west and Dakota sandstone on the east. There are three natural outlets to the east through the Dakota hogback, namely, Soldier, Dixon, and

Spring Canyons, which are the sites of three proposed dams of the same names. The fourth proposed dam, Horsetooth, will cross the valley at the north end on a low saddle separating the valley from drainage to the north into the Poudre River. The outlet will be through the Horsetooth Dam saddle. There are no outlets through the other dams. The proposed water surface is at 5,400 feet in elevation which gives a capacity of 96,756 acre-feet. The area flooded will be 1,513 acres. The outlet capacity was designed for 1,200 cubic feet per second with reservoir full. This large capacity is necessary as the irrigation use requires that the entire amount of supplemental water be delivered at a rate that would supply it in 60 days.

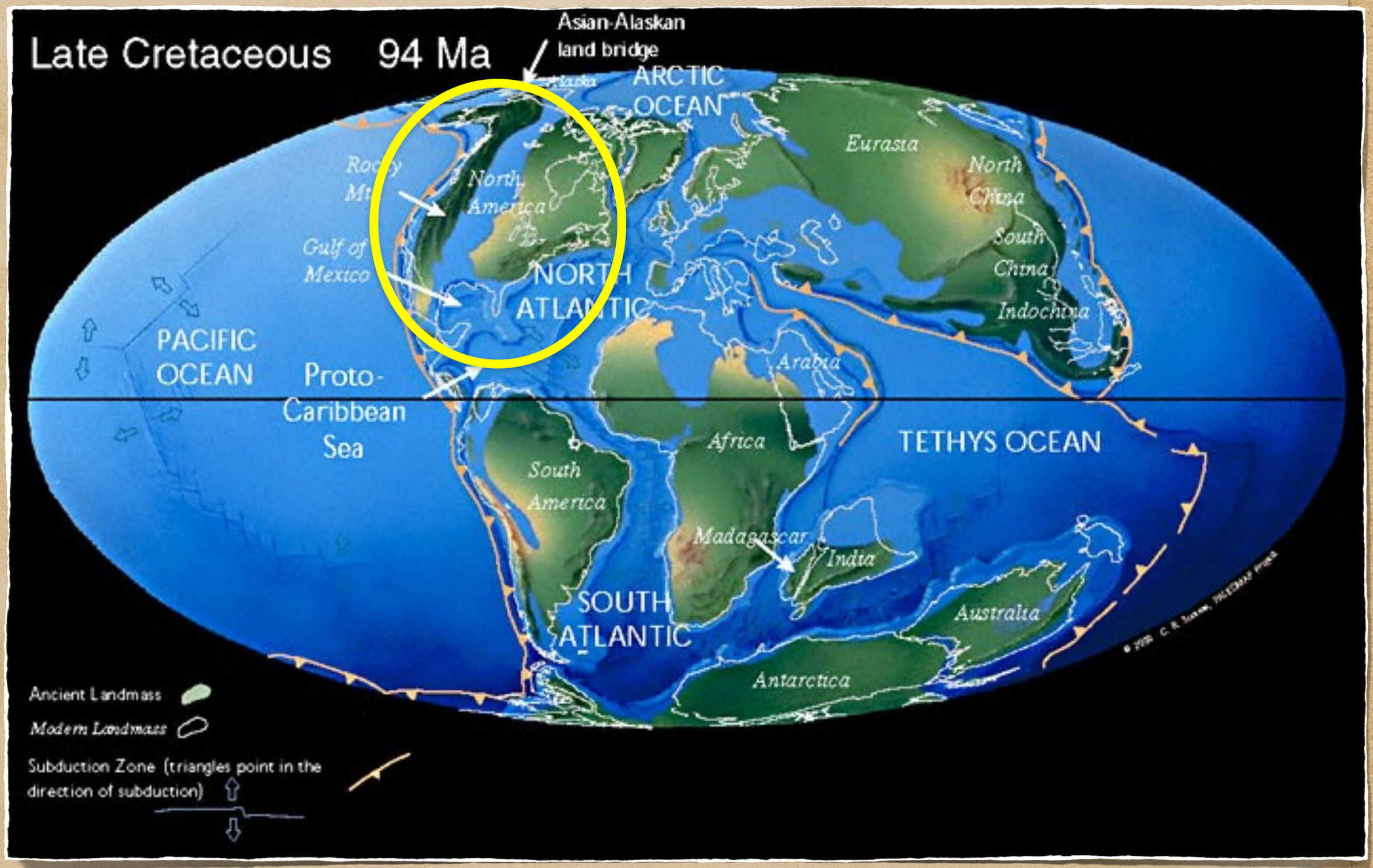
The advantages of a reservoir at this point are: It is high enough to supply all users from the main Cache La Poudre River and is located close to it. It takes the place of 6 miles of canal through rough country and allows a canal of 250 cubic second-feet to be constructed from the Big Thompson instead of one for 1,000 cubic feet per second.

The estimated cost of the reservoir is $3,625,021.

POUDRE FEEDER CANAL





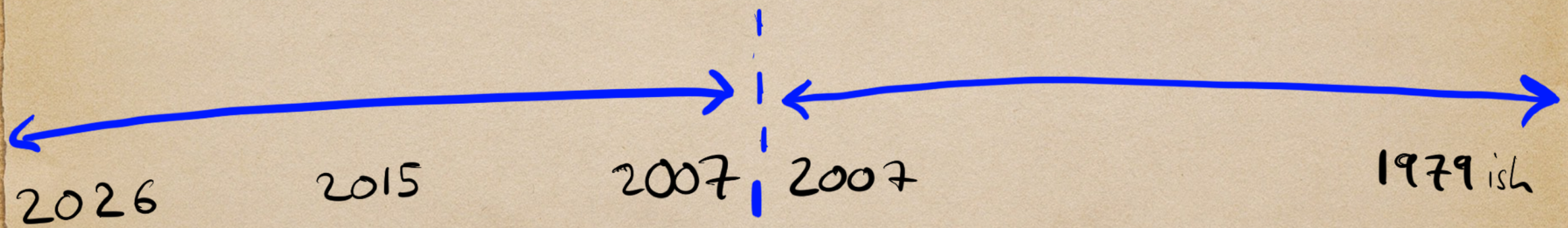

Late Cretaceous 94 Ma
Asian-Alaskan land bridge
Alaska
ARCTIC OCEAN
Eurasia
North China
South China
Indochina
Rocky Mts.
North America
Gulf of Mexico
NORTH ATLANTIC
PACIFIC OCEAN
Proto-Caribbean Sea
Arabia
Africa
TETHYS OCEAN
South America
Madagascar
India
SOUTH ATLANTIC
Australia
Antarctica
Ancient Landmass
Modern Landmass
Subduction Zone (triangles point in the direction of subduction)

batch jobs; both predated Linux control groups. Borg shares machines between these two types of applications as a way of increasing resource utilization and thereby reducing costs. Such sharing was possible because container support in the Linux kernel was becoming available (indeed, Google contributed much of the container code to the Linux kernel), which enabled better isolation between latency-sensitive user-facing services and CPU-hungry batch processes.

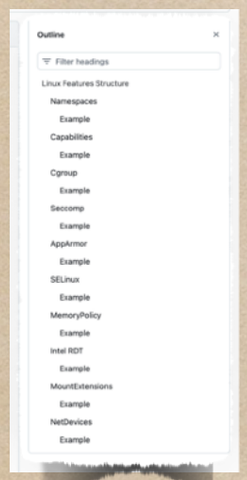
https://spawn-queue.acm.org/doi/pdf/10.1145/2898442.2898444 - "Borg, Omega, and Kubernetes", 2016

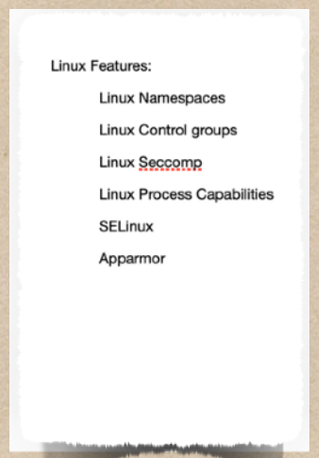
How did Docker build the right thing for isolation tools?

Where did the isolation tools come from?



How did Docker build the right thing for isolation tools?

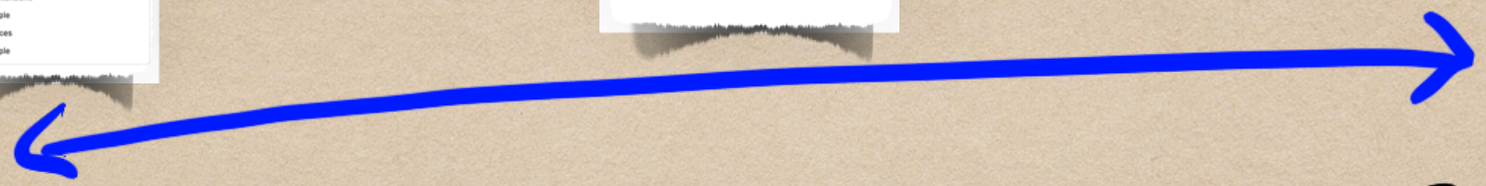




what is now known as containers because now you hear, "Oh, containers existed the whole time and the Docker guys just slapped a nice UI on it and some marketing." Totally not true, first of all. But it was a gradual step by step process of all of us reaching the design that exists today. Definitely, for me, that was the thread for everything I did starting from right after school.

I quickly became obsessed with that tech, cgroups. I think cgroups maybe existed in a very rudimentary form, but when I started playing with containers on Linux you had to patch the kernel and it was either Vserver which was the oldest patch that I know of, a lightweight server, lightweight

It's still a fuzzy concept to explain now, but that was just such an exciting gut feeling, a nerd's gut feeling that there was something really cool there and I want to do more of that. That was 2007 to, I guess, now. That's the thread.



2026

2015

2007

This is great, because it's simple. At least, it *seems* simple. Each container was fully contained (so to speak) within `/var/lib/dotcloud/<containername>`, so you could move a container simply by copying that directory to another machine. Of course, copying this directory *while the container is running* requires extra precautions; but there was something satisfying and UNIX-y in the fact that a container was just a directory, after all.

But when you have thousands of containers (dotCloud peaked above 100,000 containers) running across hundreds of nodes, and your users constantly deploy and scale services, you need an orchestrator. More specifically, you need to automate *resource scheduling*.

Outline

Filter headings

Linux Features Structure

- Namespaces
  - Example
- Capabilities
  - Example
- Cgroup
  - Example
- Seccomp
  - Example
- AppArmor
  - Example
- SELinux
  - Example
- MemoryPolicy
  - Example
- Intel RDT
  - Example
- MountExtensions
  - Example
- NetDevices
  - Example

2026

Linux Features:

- Linux Namespaces
- Linux Control groups
- Linux Seccomp
- Linux Process Capabilities
- SELinux
- Apparmor

2015

Linux Features:

- Linux Namespaces
- Linux Control groups
- Linux Seccomp
- Linux Process Capabilities
- SELinux
- Apparmor

2015

what is now known as containers because now you hear, "Oh, containers existed the whole time and the Docker guys just slapped a nice UI on it and some marketing." Totally not true, first of all. But it was a gradual step by step process of all of us reaching the design that exists today. Definitely, for me, that was the thread for everything I did starting from right after school.

I quickly became obsessed with that tech, cgroups. I think cgroups maybe existed in a very rudimentary form, but when I started playing with containers on Linux you had to patch the kernel and it was either Vserver which was the oldest patch that I know of, a lightweight server, lightweight

It's still a fuzzy concept to explain now, but that was just such an exciting gut feeling, a nerd's gut feeling that there was something really cool there and I want to do more of that. That was 2007 to, I guess, now. That's the thread.

- https://www.heavybit.com/library/podcasts/the-kubelist-podcast/ep-36-the-docker-story-with-solomon-hykes

It was a direct Heroku competitor, so we handled deployment and hosting of the application all in one, super easy, we'd scale it and monitor it, blah, blah, blah. Kind of hands off. The big differentiator was we did it for many languages, not just Ruby, because at the time Heroku was Ruby only and there were a bunch of Heroku clones but they were always for only one language because there were basically just fancy managed hosting. So the whole stack was completely hardwired for that one language.

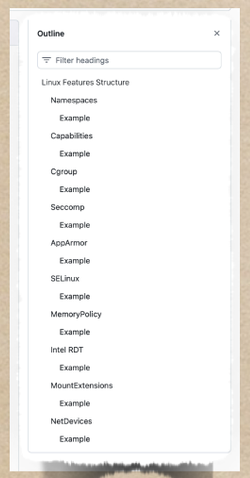
- https://www.heavybit.com/library/podcasts/the-kubelist-podcast/ep-36-the-docker-story-with-solomon-hykes

But when you have thousands of containers (dotCloud peaked above 100,000 containers) running across hundreds of nodes, and your users constantly deploy and scale services, you need an orchestrator. More specifically, you need to automate *resource scheduling*.

This is great, because it's simple. At least, it *seems* simple. Each container was fully contained (so to speak) within `/var/lib/dotcloud/<containername>`, so you could move a container simply by copying that directory to another machine. Of course, copying this directory *while the container is running* requires extra precautions; but there was something satisfying and UNIX-y in the fact that a container was just a directory, after all.

- https://jpetazzo.github.io/2017/02/24/from-dotcloud-to-docker/

How did Docker build the right thing for isolation tools?



Linux Features:

- Linux Namespaces
- Linux Control groups
- Linux Seccomp
- Linux Process Capabilities
- SELinux
- Apparmor

hat is now known as containers because now you hear, "Oh, containers existed the whole time nd the Docker guys just slapped a nice UI on it and some marketing." Totally not true, first of all, ut it was a gradual step by step process of all of us reaching the design that exists today. efinitely, for me, that was the thread for everything I did starting from right after school.

quickly became obsessed with that tech, cgroups. I think cgroups maybe existed in a very udimentary form, but when I started playing with containers on Linux you had to patch the kernel d it was either Vserver which was the oldest patch that I know of, a lightweight server, lightwei

It's still a fuzzy concept to explain now, but that was just such an exciting gut feeling, a nerd's gut feeling that there was something really cool there and I want to do more of that. That was 2007 to, I guess, now. That's the thread.

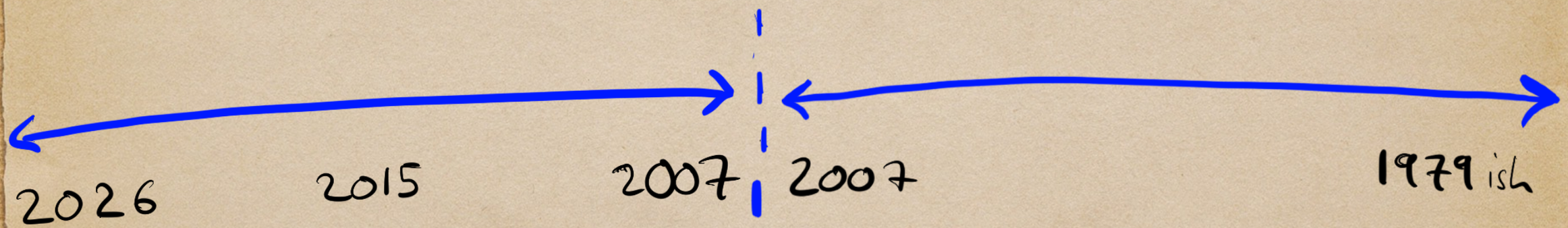


is is great, because it's simple. At least, it *seems* simple. Each containe as fully contained (so to speak) within `/var/lib/dotcloud/<containername>` o you could move a container simply by copying that directory to anothe achine. Of course, copying this directory *while the container is runnin* equires extra precautions; but there was something satisfying and UNIX-y i e fact that a container was just a directory, after all.

But when you have thousands of containers (dotCloud peaked above 100,000 containers) running across hundreds of nodes, and your users constantly deploy and scale services, you need an orchestrator. More specifically, you need to automate *resource scheduling*.

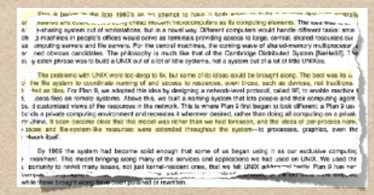
How did Docker build the right thing for isolation tools?

Where did the isolation tools come from?



Where did the isolation tools come from?





To make this happen I need to solve the challenging problem of how to refactor the UNIX/Linux API so that we can have multiple instances of the global Linux namespaces. The Plan 9 inspired mount/filesystem namespace has already proved how this can be done and is slowly proving useful.

Development of Jails

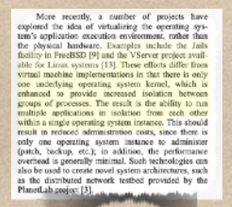
A fellow named Derrick T. Woolworth contacted me about something in FreeBSD I have long since forgotten, and as we exchanged emails he complained about the fact that different customers in his webhotel needed different versions of apache, mysql, perl etc., and that his forced him to run many machines, each almost idle, just for these different software loads.

Thinking about this I realized that chroot(8) could be pretty trivially extended to become light-weight virtual "machines" and proposed that Derricks company could fund the development.

The actual development consisted of five parts:

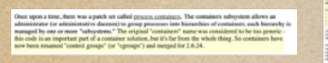
- Making sure you don't escape the chroot/jail
- Restricting process visibility
- Deciding what "root" can and cannot do in a jail
- Teach certain device drivers about jails
- Giving each jail it's own IP

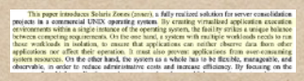




Currently I have identified ten separate namespaces in the kernel. The filesystem mount namespace, uts namespace, the SYS V IPC namespace, network namespace, the pid namespace, the uid namespace, the security namespace, the security keys namespace, the device namespace, and the time namespace.











1.2 Jails

Outside the realm of high perfomance computing people have been restricting their server application to chroot jails for years. The problems with chroot jails have become well understood and people have begun fixing them. First BSD jails, and then Solaris containers are some of the better known examples.

An Evening with Berferd
In Which a Cracker is Lured, Endured, and Studied

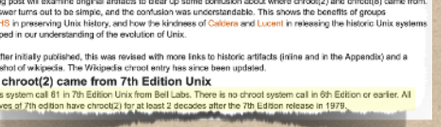
Bill Cheswick

AT&T Bell Laboratories

Abstract

On 7 January 1991 a cracker, believing he had discovered the famous sendmail DEBUG hole in our Internet gateway machine, attempted to obtain a copy of our password file. I sent him one.

For several months we led this cracker on a merry chase in order to trace his location and learn his techniques. This paper is a chronicle of the cracker's "successes" and disappointments, the bait and traps used to lure and detect him, and the chroot "jail" we built to watch his activities.


Warner's Random Hacking Blog

A Diary of Warner's Hacking Projects and other random thoughts

20200527

Whither chroot?

Chroot Origins

This blog post will examine original artifacts to clear up some confusion about where chroot(2) and chroot(8) came from. The answer turns out to be simple, and the confusion was understandable. This shows the benefits of groups like TUHS in preserving Unix history, and how the kindnesses of Caldera and Lucent in releasing the historic Unix systems has helped in our understanding of the evolution of Unix.

EDIT: After initially published, this was revised with more links to historic artifacts (inline and in the Appendix) and a screen shot of wikipedia. The Wikipedia chroot entry has since been updated.

tl;dr: chroot(2) came from 7th Edition Unix

chroot is system call 61 in 7th Edition Unix from Bell Labs. There is no chroot system call in 6th Edition or earlier. All derivatives of 7th edition have chroot(2) for at least 2 decades after the 7th Edition release in 1979.

2007

1979 ish

Subject [patch00/05]: Containers(V2)- Introduction
From Rohit Seth <>
Date Tue, 19 Sep 2006 19:16:08 -0700

Containers:

Commodity HW is becoming more powerful. This is giving opportunity to run different workloads on the same platform for better HW resource utilization. To run different workloads efficiently on the same platform, it is critical that we have a notion of limits for each workload in Linux kernel. Current cpuset feature in Linux kernel provides grouping of CPU and memory support to some extent (for NUMA machines).

For example, a user can run a batch job like backup inside containers. This job if run unconstrained could step over most of the memory present in system thus impacting other workloads running on the system at that time. But when the same job is run inside containers then the backup job is run within container limits.

We use the term container to indicate a structure against which we track and charge utilization of system resources like memory, tasks etc for a workload. Containers will allow system admins to customize the underlying platform for different applications based on their performance and HW resource utilization needs. Containers contain enough infrastructure to allow optimal resource utilization without bogging down rest of the kernel. A system admin should be able to create, manage and free containers easily.

At the same time, changes in kernel are minimized so as this support can be easily integrated with mainline kernel.

The user interface for containers is through configfs. Appropriate file system privileges are required to do operations on each container. Currently implemented container resources are automatically visible to user space through /configfs/container/<container_name> after a container is created.

Signed-off-by: Rohit Seth <rohitseth@google.com>

https://lkml.org/lkml/2006/9/19/283 - Rohit Seth introducing proto-cgroups

From: menage@google.com
To: akpm@linux-foundation.org, dev@sw.ru, xemul@sw.ru, serue@us.ibm.com, vatsa@in.ibm.com, ebiederm@xmission.com, haveblue@us.ibm.com, svaidy@linux.vnet.ibm.com, balbir@in.ibm.com, pj@sgi.com, cpw@sgi.com
Subject: [PATCH 00/10] Containers(V10): Generic Process Containers
Date: Tue, 29 May 2007 06:01:04 -0700
Cc: ckrm-tech@lists.sourceforge.net, linux-kernel@vger.kernel.org, containers@lists.osdl.org, mbligh@google.com, rohitseth@google.com, devel@openvz.org

```
This is an update to my multi-hierarchy multi-subsystem generic
process containers patch. Changes since V9 (April 27th) include:

- The patchset has been rebased over 2.6.22-rc2-mm1

- A lattice of lists linking tasks to their css_groups and css_groups
to their containers has been added to support more efficient iteration
across the member tasks of a container.

- Support for the cpusets "release agent" functionality has been added
back in; this is based on a workqueue concept similar to the changes
that Cliff Wickman has been pushing for supporting CPU hot-unplug.

- Several uses of tasklist_lock replaced by reliance on RCU

- Misc cleanups

- Tested with a tweaked version of PaulJ's cpuset_test script

Still TODO:

- decide whether "Containers" is an acceptable name for the system
given its usage by some other development groups, or whether something
else (ProcessSets? ResourceGroups? TaskGroups?) would be better. I'm
inclined to leave this political decision to Andrew/Linus once they're
happy with the technical aspects of the patches.

- add a hash-table based lookup for css_group objects.

- use seq_file properly in container tasks files to avoid having to
 allocate a big array for all the container's task pointers.

- lots more testing

- define standards for container file names

--

Generic Process Containers
--------------------------
```

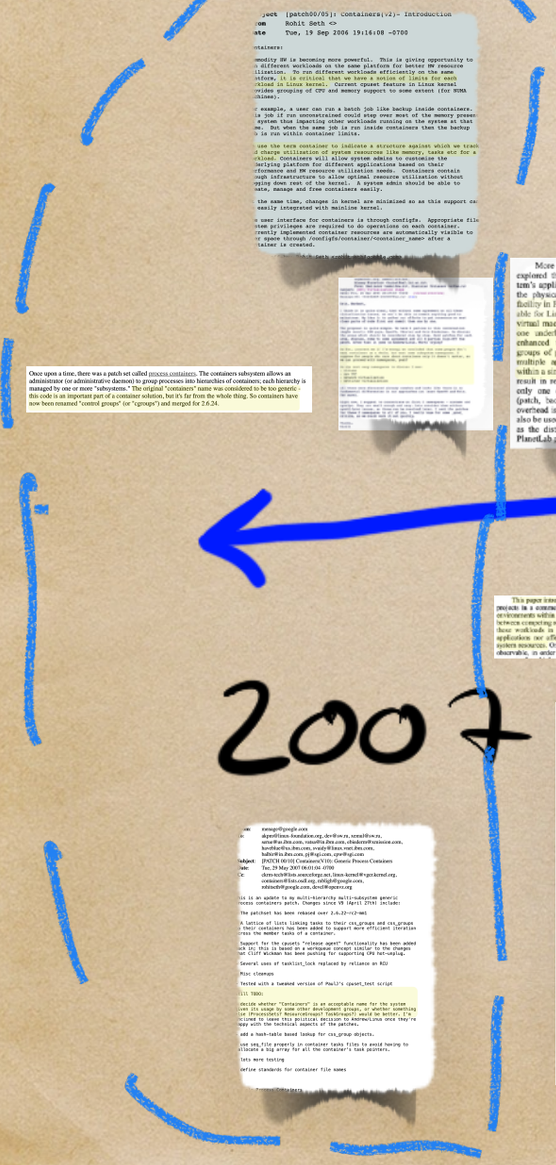
https://lwn.net/Articles/236032/ - let's not call them containers?

Once upon a time, there was a patch set called process containers. The containers subsystem allows an administrator (or administrative daemon) to group processes into hierarchies of containers; each hierarchy is managed by one or more "subsystems." The original "containers" name was considered to be too generic - this code is an important part of a container solution, but it's far from the whole thing. So containers have now been renamed "control groups" (or "cgroups") and merged for 2.6.24.

https://lwn.net/Articles/256389/ - let's not call them containers!

cgroups

Where did the isolation tools come from?



akpm@osdl.org, sam@vilain.net,
Alexey Kuznetsov <kuznet@ms2.inr.ac.ru>,
Pavel Emelianov <xemul@sw.ru>, Stanislav Protassov <st@sw.ru>
Subject: [RFC] Virtualization steps
Date: Fri, 24 Mar 2006 20:19:59 +0300 [thread overview]
Message-ID: <44242A3F.1010307@sw.ru> (raw)

Eric, Herbert,

I think it is quite clear, that without some agreement on all these
virtualization issues, we won't be able to commit anything good to
mainstream. My idea is to gather our efforts to get consensus on most
clean parts of code first and commit them one by one.

The proposal is quite simple. We have 4 parties in this conversation
(maybe more?): IBM guys, OpenVZ, VServer and Eric Biederman. We discuss
the areas which should be considered step by step. Send patches for each
area, discuss, come to some agreement and all 4 parties Sign-Off the
patch. After that it goes to Andrew/Linus. Worth trying?

So far, (correct me if I'm wrong) we concluded that some people don't
want containers as a whole, but want some subsystem namespaces. I
suppose for people who care about containers only it doesn't matter, so
we can proceed with namespaces, yeah?

So the most easy namespaces to discuss I see:
- utsname
- sys IPC
- network virtualization
- netfilter virtualization

all these were discussed already somehow and looks like there is no
fundamental differencies in our approaches (at least OpenVZ and Eric,
for sure).

Right now, I suggest to concentrate on first 2 namespaces - utsname and
sysvipc. They are small enough and easy. Lets consider them without
sysctl/proc issues, as those can be resolved later. I sent the patches
for these 2 namespaces to all of you. I really hope for some _good_
critics, so we could work it out quickly.

Thanks,
Kirill

https://lore.kernel.org/all/44242A3F.1010307@sw.ru/ - Kirill Korotaev namespaces email

Multiple Instances of the Global Linux Namespaces

Eric W. Biederman
Linux Networx
ebiederman@lnxi.com

Abstract

Currently Linux has the filesystem namespace for mounts which is beginning to prove useful. By adding additional namespaces for process ids, SYS V IPC, the network stack, user ids, and probably others we can, at a trivial cost, extend the UNIX concept and make novel uses of Linux possible. Multiple instances of a namespace simply means that you can have two things with the same name.

For servers the power of computers is growing, and it has become possible for a single server to easily fulfill the tasks of what previously required multiple servers. Hypervisor solutions like Xen are nice but they impose a performance penalty and they do not easily allow resources to be shared between multiple servers.

For clusters application migration and preemption are interesting cases but almost impossibly hard because you cannot restart the application once you have moved it to a new machine, as usually there are resource name conflicts.

For users certain desktop applications interface with the outside world and are large and hard to secure. It would be nice if those applications could be run on their own little world to limit what a security breach could compromise.

Several implementations of this basic idea have been done succsessfully. Now the work is to create a clean implementation that can be merged into the Linux kernel. The discussion has begun on the *linux-kernel* list and things are slowly progressing.

1 Introduction

1.1 High Performance Computing

I have been working with high performance clusters for several years and the situation is painful. Each Linux box in the cluster is refered to as a node, and applications running or queued to run on a cluster are jobs.

Jobs are run by a batch scheduler and, once launched, each job runs to completion typically consuming 99% of the resources on the nodes it is running on.

In practice a job cannot be suspended, swapped, or even moved to a different set of nodes once it is launched. This is the oldest and most primitive way of running a computer. Given the long runs and high computation overhead of HPC jobs it isn't a bad fit for HPC environments, but it isn't a really good fit either.

Linux has much more modern facilities. What prevents us from just using them?

https://www.kernel.org/doc/ols/2006/ols2006v1-pages-101-112.pdf

To make this happen I need to solve the challenging problem of how to refactor the UNIX/Linux API so that we can have multiple instances of the global Linux namespaces. The Plan 9 inspired mount/filesystem namespace has already proved how this can be done and is slowly proving useful.

1.2 Jails

Outside the realm of high perfomance computing people have been restricting their server application to chroot jails for years. The problems with chroot jails have become well understood and people have begun fixing them. First BSD jails, and then Solaris containers are some of the better known examples.

Currently I have identified ten separate namespaces in the kernel. The filesystem mount namespace, uts namespace, the SYS V IPC namespace, network namespace, the pid namespace, the uid namespace, the security namespace, the security keys namespace, the device namespace, and the time namespace.

https://www.kernel.org/doc/ols/2006/ols2006v1-pages-101-112.pdf

This paper introduces Solaris Zones (*zones*), a fully realized solution for server consolidation projects in a commercial UNIX operating system. By creating virtualized application execution environments within a single instance of the operating system, the facility strikes a unique balance between competing requirements. On the one hand, a system with multiple workloads needs to run those workloads in isolation, to ensure that applications can neither observe data from other applications nor affect their operation. It must also prevent applications from over-consuming system resources. On the other hand, the system as a whole has to be flexible, manageable, and observable, in order to reduce administrative costs and increase efficiency. By focusing on the

https://www.usenix.org/legacy/event/lisa04/tech/full_papers/price/price.pdf - Solaris Zones: Operating System Support for Consolidating Commercial Workloads, 2004

More recently, a number of projects have explored the idea of virtualizing the operating system's application execution environment, rather than the physical hardware. Examples include the Jails facility in FreeBSD [9] and the VServer project available for Linux systems [13]. These efforts differ from virtual machine implementations in that there is only one underlying operating system kernel, which is enhanced to provide increased isolation between groups of processes. The result is the ability to run multiple applications in isolation from each other within a single operating system instance. This should result in reduced administration costs, since there is only one operating system instance to administer (patch, backup, etc.); in addition, the performance overhead is generally minimal. Such technologies can also be used to create novel system architectures, such as the distributed network testbed provided by the PlanetLab project [3].

https://www.usenix.org/legacy/event/lisa04/tech/full_papers/price/price.pdf - Solaris Zones: Operating System Support for Consolidating Commercial Workloads, 2004

Date: Sat, 24 Feb 2001 23:16:10 -0500 (EST)
From: Alexander Viro <viro@math.psu.edu>
To: linux-fsdevel@vger.kernel.org
Subject: [PATCH][CFT] per-process namespaces for Linux

He's back. And this time he's got a chainsaw.

Yes, folks. We got per-process namespaces. Working. With proper behaviour on exit(), yodda, yodda. Enjoy. Help with testing would be more than welcome.

Current patch is on ftp.math.psu.edu/pub/viro/namespaces-S2.gz
It's against 2.4.2.

Contents:
* proper refcounting of struct super_block
* GC for vfsmounts (finally)
* fix for races between get_super() and umount()
* SMP-safe lock_super()
* general cleanup of fs/super.c
* "lazy" option for umount() (detach from mountpoint now, do the rest when it will cease to be busy - use MNT_DETACH in 'flags' argument to get that behaviour).
* Plan 9 per-process namespaces (sans unions so far)
* large cleanup of boot process (ramdisk handling, etc.)

Variant without namespaces (they were the last part) is in the same directory, called s_lock-S2.gz.

rfork.c (in the same place) will copy a namespace and start shell in it. Use for testing... It's an equivalent of rfork(RFNAMEG) on Plan 9.

One detail - patch requires ramfs built into the kernel (boot process cleanup part needs that).

It works here (ran for about 12 hours with no problems). It's _NOT_ for inclusion into 2.4. Some pieces might go (get_super() races have to be fixed, after all), but most of this stuff is 2.5 fodder. However, it seems to be working. No doubt there are bugs and it's far from being a final version. I would call it _very_ early beta. Please, help with testing.

Comments on the code/design/amount of dope it took to write the thing (zero, actually) are welcome. I _will_ document it, but it's still not in the final form. Pretty close to it, hopefully, but...

I'm more than willing to answer questions on the design of the thing - just ask. So far that's the best I can do - all documentation is a pile of notes + CVS log.

Cheers,
Al

PS: hopefully - back for good.

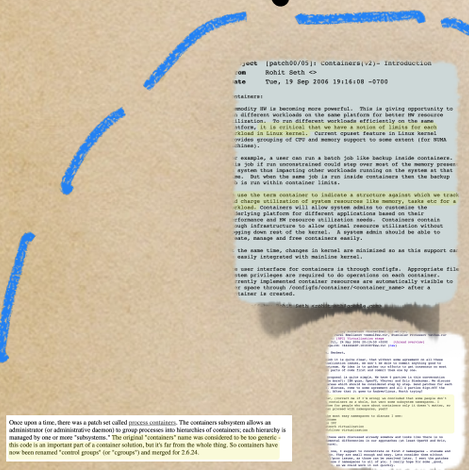
-

https://lwn.net/2001/0301/a/namespaces.php3

Cgroups

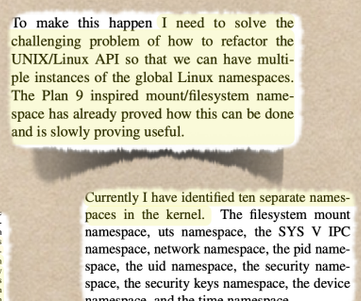
Linux Namespaces

Where did the isolation tools come from?



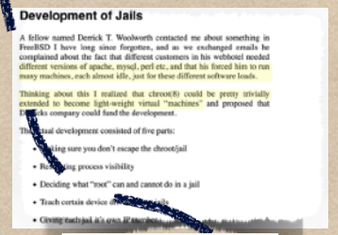
Once upon a time, there was a patch set called process containers. The containers subsystem allows an administrator (or administrative daemon) to group processes into hierarchies of containers; each hierarchy is managed by one or more "subsystems." The original "containers" name was considered to be too generic - this code is an important part of a container solution, but it's far from the whole thing. So containers have now been renamed "control groups" (or "cgroups") and merged for 2.6.24.

More recently, a number of projects have explored the idea of virtualizing the operating system's application execution environment, rather than the physical hardware. Examples include the Jails facility in FreeBSD [9] and the VServer project available for Linux systems [13]. These efforts differ from full virtual machine implementations in that there is only one underlying operating system kernel, which is enhanced to provide increased isolation between groups of processes. The result is the ability to run multiple applications in isolation from each other within a single operating system instance. This should result in reduced administration costs, since there is only one operating system instance to administer (patch, backup, etc.); in addition, the performance overhead is generally minimal. Such technologies can also be used to create novel system architectures, such as the distributed network testbed provided by the



To make this happen I need to solve the challenging problem of how to refactor the UNIX/Linux API so that we can have multiple instances of the global Linux namespaces. The Plan 9 inspired mount/filesystem namespace has already proved how this can be done and is slowly proving useful.

Currently I have identified ten separate namespaces in the kernel. The filesystem mount namespace, uts namespace, the SYS V IPC namespace, network namespace, the pid namespace, the uid namespace, the security namespace, the security keys namespace, the device namespace, and the time namespace.



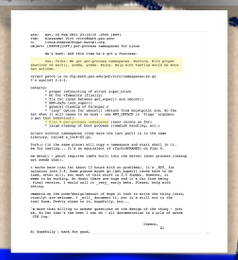
Development of Jails

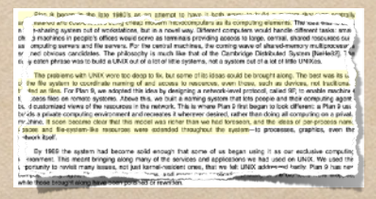
A fellow named Derrick T. Woolworth contacted me about something in FreeBSD I have long since forgotten, and as we exchanged emails he complained about the fact that different customers in his webhotel needed different versions of apache, mysql, perl etc., and that his forced him to run many machines, each almost idle, just for these different software loads.

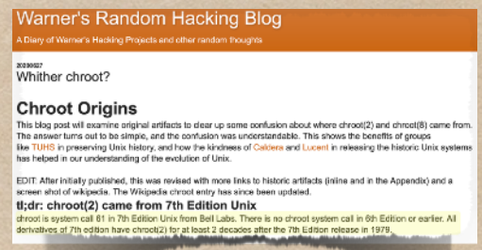
Thinking about this I realized that chroot(8) could be pretty trivially extended to become light-weight virtual "machines" and proposed that FreeBSDs company could fund the development.

The initial development consisted of five parts:

- Making sure you don't escape the chroot/jail
- Restricting process visibility
- Deciding what "root" can and cannot do in a jail
- Teach certain device drivers about jails
- Giving each jail its own IP number







Warner's Random Hacking Blog

A Diary of Warner's Hacking Projects and other random thoughts

20090327

Whither chroot?

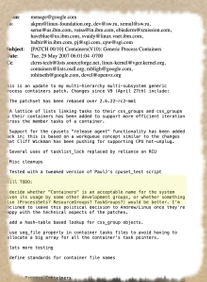
Chroot Origins

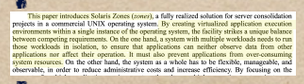
This blog post will examine original artifacts to clear up some confusion about where chroot(2) and chroot(8) came from. The answer turns out to be simple, and the confusion was understandable. This shows the benefits of groups like TUHS in preserving Unix history, and how the kindness of Caldera and Lucent in releasing the historic Unix systems has helped in our understanding of the evolution of Unix.

EDIT: After initially published, this was revised with more links to historic artifacts (inline and in the Appendix) and a screen shot of wikipedia. The Wikipedia chroot entry has since been updated.

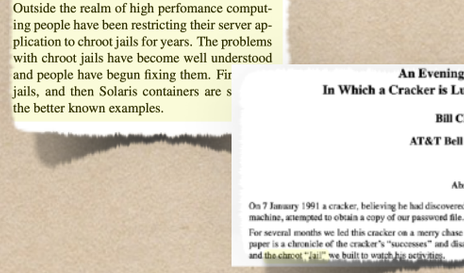
tl;dr: chroot(2) came from 7th Edition Unix

chroot is system call 61 in 7th Edition Unix from Bell Labs. There is no chroot system call in 6th Edition or earlier. All derivatives of 7th edition have chroot(2) for at least 2 decades after the 7th Edition release in 1979.





This paper introduces Solaris Zones (*zones*), a fully realized solution for server consolidation projects in a commercial UNIX operating system. By creating virtualized application execution environments within a single instance of the operating system, the facility strikes a unique balance between competing requirements. On the one hand, a system with multiple workloads needs to run those workloads in isolation, to ensure that applications can neither observe data from other applications nor affect their operation. It must also prevent applications from over-consuming system resources. On the other hand, the system as a whole has to be flexible, manageable, and observable, in order to reduce administrative costs and increase efficiency. By focusing on the



1.2 Jails

Outside the realm of high perfomance computing people have been restricting their server application to chroot jails for years. The problems with chroot jails have become well understood and people have begun fixing them. Fir
jails, and then Solaris containers are s
the better known examples.



Multiple Instances of the Global Linux Namespaces

Eric W. Biederman
Linux Networx
ebiederman@lnxi.com

Abstract

Currently Linux has the filesystem namespace for mounts which is beginning to prove useful. By adding additional namespaces for process ids, SYS V IPC, the network stack, user ids, and probably others we can, at a trivial cost, extend the UNIX concept and make novel uses of Linux possible. Multiple instances of a namespace simply means that you can have two things with the same name.

For servers the power of computers is growing, and it has become possible for a single server to easily fulfill the tasks of what previously required multiple servers. Hypervisor solutions like Xen are nice but they impose a performance penalty and they do not easily allow resources to be shared between multiple servers.

For clusters application migration and preemption are interesting cases but almost impossibly hard because you cannot restart the application once you have moved it to a new machine, as usually there are resource name conflicts.

For users certain desktop applications interface with the outside world and are large and hard to secure. It would be nice if those applications could be run on their own little world to limit what a security breach could compromise.

Several implementations of this basic idea have been done successfully. Now the work is to create a clean implementation that can be merged into the Linux kernel. The discussion has begun on the *linux-kernel* list and things are slowly progressing.

1 Introduction

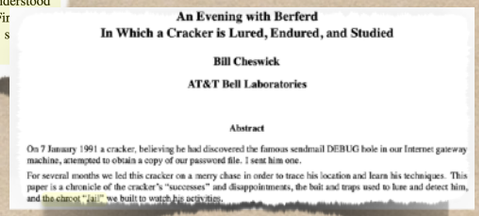
1.1 High Performance Computing

I have been working with high performance clusters for several years and the situation is painful. Each Linux box in the cluster is reduced to as a node, and applications running or queued to run on a cluster are jobs.

Jobs are run by a batch scheduler and, once launched, each job runs to completion typically consuming 99% of the resources on the nodes it is running on.

In practice a job cannot be suspended, swapped, or even moved to a different set of nodes once it is launched. This is the oldest and most primitive way of running a computer. Given the long runs and high computation overhead of HPC jobs it isn't a bad fit for HPC environments, but it isn't a really good fit either.

Linux has much more modern facilities. What prevents us from just using them?



An Evening with Berferd
In Which a Cracker is Lured, Endured, and Studied

Bill Cheswick

AT&T Bell Laboratories

Abstract

On 7 January 1991 a cracker, believing he had discovered the famous sendmail DEBUG hole in our Internet gateway machine, attempted to obtain a copy of our password file. I sent him one.

For several months we led this cracker on a merry chase in order to trace his location and learn his techniques. This paper is a chronicle of the cracker's "successes" and disappointments, the bait and traps used to lure and detect him, and the chroot "jail" we built to watch his activities.

2007

1979 ish

```
# Make a directory to jail
; mkdir jail

# Install a userland
; sudo bsdinstall jail /home/gwyn/jail

# Start the jail.
; sudo jail -c name=jail host.hostname=jail.playtechnique.io \
    ip4.addr=192.168.50.55 \
    path=/home/gwyn/jail \
    command=/bin/sh
```

Development of Jails

A fellow named Derrick T. Woolworth contacted me about something in FreeBSD I have long since forgotten, and as we exchanged emails he complained about the fact that different customers in his webhotel needed different versions of apache, mysql, perl etc, and that his forced him to run many machines, each almost idle, just for these different software loads.

Thinking about this I realized that chroot(8) could be pretty trivially extended to become light-weight virtual "machines" and proposed that Derricks company could fund the development.

The actual development consisted of five parts:

- Making sure you don't escape the chroot/jail
- Restricting process visibility
- Deciding what "root" can and cannot do in a jail
- Teach certain device drivers about jails
- Giving each jail it's own IP number.

https://phk.freebsd.dk/sagas/jails/ - "Jails – High value but shitty Virtualization" - Poul Henning Kamp

However, UNIX-style access control makes it notoriously difficult to compartmentalise functionality. While mechanisms such as chroot(2) provide a modest level compartmentalisation, it is well known that these mechanisms have serious shortcomings, both in terms of the scope of their functionality, and effectiveness at what they provide [CHROOT].

In the case of the chroot(2) call, a process's visibility of the file system name-space is limited to a single subtree. However, the compartmentalisation does not extend to the process or networking spaces and therefore both observation of and interference with processes outside their compartment is possible.

To this end, we describe the new FreeBSD "Jail" facility, which provides a strong partitioning solution, leveraging existing mechanisms, such as chroot(2), to what effectively amounts to a virtual machine environment. Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition.

- https://phk.freebsd.dk/pubs/sane2000-jail.pdf "Jails: Confining the Omnipotent Root", Poul-Henning Kamp & Robert Watson

Plan 9 began in the late 1980's as an attempt to have it both ways: to build a system that was centrally administered and cost-effective using cheap modern microcomputers as its computing elements. The idea was to build a time-sharing system out of workstations, but in a novel way. Different computers would handle different tasks: small, cheap machines in people's offices would serve as terminals providing access to large, central, shared resources such as computing servers and file servers. For the central machines, the coming wave of shared-memory multiprocessors seemed obvious candidates. The philosophy is much like that of the Cambridge Distributed System [NeHe82]. The early catch phrase was to build a UNIX out of a lot of little systems, not a system out of a lot of little UNIXes.

The problems with UNIX were too deep to fix, but some of its ideas could be brought along. The best was its use of the file system to coordinate naming of and access to resources, even those, such as devices, not traditionally treated as files. For Plan 9, we adopted this idea by designing a network-level protocol, called 9P, to enable machines to access files on remote systems. Above this, we built a naming system that lets people and their computing agents build customized views of the resources in the network. This is where Plan 9 first began to look different: a Plan 9 user builds a private computing environment and recreates it wherever desired, rather than doing all computing on a private machine. It soon became clear that this model was richer than we had foreseen, and the ideas of per-process name spaces and file-system-like resources were extended throughout the system—to processes, graphics, even the network itself.

By 1989 the system had become solid enough that some of us began using it as our exclusive computing environment. This meant bringing along many of the services and applications we had used on UNIX. We used this opportunity to revisit many issues, not just kernel-resident ones, that we felt UNIX addressed badly. Plan 9 has new compilers, languages, libraries, window systems, and many new applications. Many of the old tools were dropped, while those brought along have been polished or rewritten.

- https://doc.cat-v.org/plan_9/4th_edition/papers/9, "Plan 9 From Bell Labs"

Warner's Random Hacking Blog

A Diary of Warner's Hacking Projects and other random thoughts

20200627

Whither chroot?

Chroot Origins

This blog post will examine original artifacts to clear up some confusion about where chroot(2) and chroot(8) came from. The answer turns out to be simple, and the confusion was understandable. This shows the benefits of groups like TUHS in preserving Unix history, and how the kindness of Caldera and Lucent in releasing the historic Unix systems has helped in our understanding of the evolution of Unix.

EDIT: After initially published, this was revised with more links to historic artifacts (inline and in the Appendix) and a screen shot of wikipedia. The Wikipedia chroot entry has since been updated.

tl;dr: chroot(2) came from 7th Edition Unix

chroot is system call 61 in 7th Edition Unix from Bell Labs. There is no chroot system call in 6th Edition or earlier. All derivatives of 7th edition have chroot(2) for at least 2 decades after the 7th Edition release in 1979.

https://bsdimp.blogspot.com/2020/06/whither-chroot.html?m=1 - "Whither Chroot", Warner Losh, my personal hero

An Evening with Berferd
In Which a Cracker is Lured, Endured, and Studied

Bill Cheswick

AT&T Bell Laboratories

Abstract

On 7 January 1991 a cracker, believing he had discovered the famous sendmail DEBUG hole in our Internet gateway machine, attempted to obtain a copy of our password file. I sent him one.

For several months we led this cracker on a merry chase in order to trace his location and learn his techniques. This paper is a chronicle of the cracker's "successes" and disappointments, the bait and traps used to lure and detect him, and the chroot "Jail" we built to watch his activities.

https://www.cheswick.com/ches/papers/berferd.pdf - "An Evening with Berferd", Bill Cheswick

Where did the isolation tools come from?

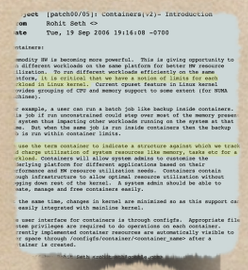
Cgroups

Linux Namespaces

Jails + Plan 9

2007

1979 ish

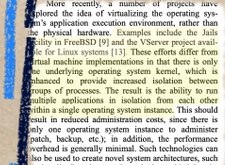




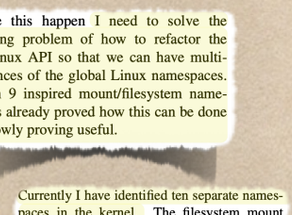
Once upon a time, there was a patch set called process containers. The containers subsystem allows an administrator (or administrative daemon) to group processes into hierarchies of containers; each hierarchy is managed by one or more "subsystems." The original "containers" name was considered to be too generic - this code is an important part of a container solution, but it's far from the whole thing. So containers have now been renamed "control groups" (or "cgroups") and merged for 2.6.24.



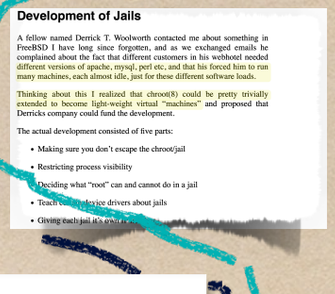
To make this happen I need to solve the challenging problem of how to refactor the UNIX/Linux API so that we can have multiple instances of the global Linux namespaces. The Plan 9 inspired mount/filesystem namespace has already proved how this can be done and is slowly proving useful.



More recently, a number of projects have explored the idea of virtualizing the operating system's application execution environment, rather than the physical hardware. Examples include the Jails facility in FreeBSD [9] and the VServer project available for Linux systems [13]. These efforts differ from virtual machine implementations in that there is only one underlying operating system kernel, which is enhanced to provide increased isolation between groups of processes. The result is the ability to run multiple applications in isolation from each other within a single operating system instance. This should result in reduced administration costs, since there is only one operating system instance to administer (patch, backup, etc.); in addition, the performance overhead is generally minimal. Such technologies can also be used to create novel system architectures, such as the distributed network testbed provided by the [illegible] project [3].



Currently I have identified ten separate namespaces in the kernel. The filesystem mount namespace, uts namespace, the SYS V IPC namespace, network namespace, the pid namespace, the uid namespace, the security namespace, the security keys namespace, the device namespace, and the time namespace.



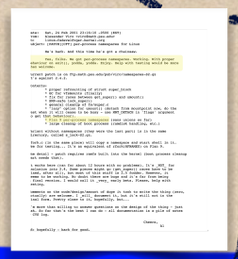
Development of Jails

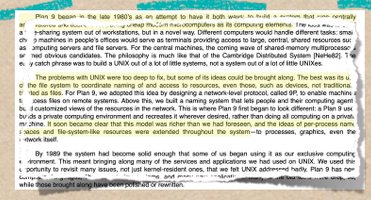
A fellow named Derrick T. Woolworth contacted me about something in FreeBSD I have long since forgotten, and as we exchanged emails he complained about the fact that different customers in his webhotel needed different versions of apache, mysql, perl etc, and that his forced him to run many machines, each almost idle, just for these different software loads.

Thinking about this I realized that chroot(8) could be pretty trivially extended to become light-weight virtual "machines" and proposed that Derricks company could fund the development.

The actual development consisted of five parts:

- Making sure you don't escape the chroot/jail
- Restricting process visibility
- Deciding what "root" can and cannot do in a jail
- Teach [illegible] device drivers about jails
- Giving each jail it's own [illegible]

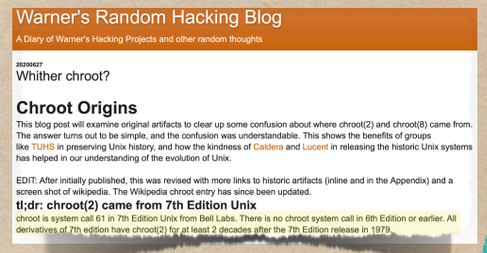




Plan 9 began in the late 1980's as an attempt to have it both work [illegible] computers as its computing elements. The [illegible] sharing system out of workstations, but in a novel way. Different computers would handle different tasks: small machines in people's offices would serve as terminals providing access to large, central, shared resources such as computing servers and file servers. For the central machines, the coming wave of shared-memory multiprocessor had obvious candidates. The philosophy is much like that of the Cambridge Distributed System [Needham82]. The catch phrase was to build a UNIX out of a lot of little systems, not a system out of a lot of little UNIXes.

The problems with UNIX were too deep to fix, but some of its ideas could be brought along. The best was its use of the file system to coordinate naming of and access to resources, even those, such as devices, not traditionally [illegible] ked as files. For Plan 9, we adopted this idea by designing a network-level protocol, called 9P, to enable machines to access files on remote systems. Above this, we built a naming system that lets people and their computing agents build customized views of the resources in the network. This is where Plan 9 first began to look different: a Plan 9 user builds a private computing environment and recreates it wherever desired, rather than doing all computing on a private machine. It soon became clear that this model was richer than we had foreseen, and the ideas of per-process name spaces and file-system-like resources were extended throughout the system—to processes, graphics, even the network itself.

By 1989 the system had become solid enough that some of us began using it as our exclusive computing environment. This meant bringing along many of the services and applications we had used on UNIX. We used this opportunity to revisit many issues, not just kernel-resident ones, that we felt UNIX addressed badly. Plan 9 has new [illegible] and [illegible] while those brought along have been polished or rewritten.



Warner's Random Hacking Blog

A Diary of Warner's Hacking Projects and other random thoughts

20200627

Whither chroot?

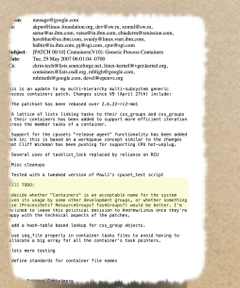
Chroot Origins

This blog post will examine original artifacts to clear up some confusion about where chroot(2) and chroot(8) came from. The answer turns out to be simple, and the confusion was understandable. This shows the benefits of groups like TUHS in preserving Unix history, and how the kindness of Caldera and Lucent in releasing the historic Unix systems has helped in our understanding of the evolution of Unix.

EDIT: After initially published, this was revised with more links to historic artifacts (inline and in the Appendix) and a screen shot of wikipedia. The Wikipedia chroot entry has since been updated.

tl;dr: chroot(2) came from 7th Edition Unix

chroot is system call 61 in 7th Edition Unix from Bell Labs. There is no chroot system call in 6th Edition or earlier. All derivatives of 7th edition have chroot(2) for at least 2 decades after the 7th Edition release in 1979.





Multiple Instances of the Global Linux Namespaces

Eric W. Biederman
Linux Networx
ebiederman@lnxi.com

Abstract

Currently Linux has the filesystem namespace for mounts which is beginning to prove useful. By adding additional namespaces for process ids. SYS V IPC. the network stack, user ids, and probably others we can, at a trivial cost, extend the UNIX concept and make novel uses of Linux possible. Multiple instances of a namespace simply means that you can have two things with the same name.

For servers the power of computers is growing, and it has become possible for a single server to easily fulfill the tasks of what previously required multiple servers. Hypervisor solutions like Xen are nice but they impose a performance penalty and they do not easily allow resources to be shared between multiple servers.

For clusters application migration and preemption are interesting cases but almost impossibly hard because you cannot restart the application once you have moved it to a new machine, as usually there are resource name conflicts.

For users certain desktop applications interface with the outside world and are large and hard to secure. It would be nice if these applications could be run on their own little world to limit what a security breach could compromise.

Several implementations of this basic idea have been done successfully. Now the work is to create a clean implementation that can be merged into the Linux kernel. The discussion has begun on the *linux-kernel* list and things are slowly progressing.

1 Introduction

1.1 High Performance Computing

I have been working with high performance clusters for several years and the situation is painful. Each Linux box in the cluster is referred to as a node, and applications running or queued to run on a cluster are jobs.

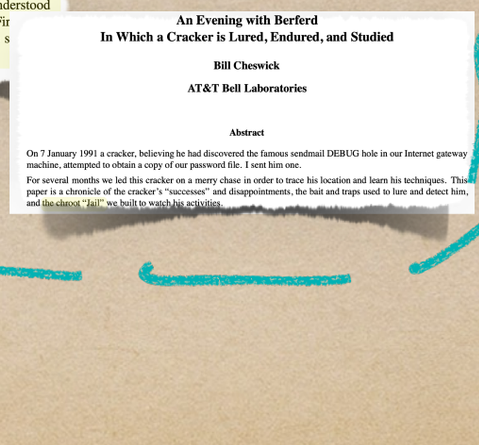
Jobs are run by a batch scheduler and, once launched, each job runs to completion typically consuming 99% of the resources on the nodes it is running on.

In practice a job cannot be suspended, swapped, or even moved to a different set of nodes once it is launched. This is the oldest and most primitive way of running a computer. Given the long runs and high computation overhead of HPC jobs it isn't a bad fit for HPC environments, but it isn't a really good fit either.

Linux has much more modern facilities. What prevents us from just using them?

2 Jails

Outside the realm of high perfomance computing people have been restricting their server application to chroot jails for years. The problems with chroot jails have become well understood and people have begun fixing them. First jails, and then Solaris containers are s[illegible] the better known examples.



This paper introduces Solaris Zones (zones), a fully realized solution for server consolidation projects in a commercial UNIX operating system. By creating virtualized application execution environments within a single instance of the operating system, the facility strikes a unique balance between competing requirements. On the one hand, a system with multiple workloads needs to run those workloads in isolation, to ensure that applications can neither observe data from other applications nor affect their operation. It must also prevent applications from over-consuming system resources. On the other hand, the system as a whole has to be flexible, manageable, and observable, in order to reduce administrative costs and increase efficiency. By focusing on the

An Evening with Berferd
In Which a Cracker is Lured, Endured, and Studied

Bill Cheswick

AT&T Bell Laboratories

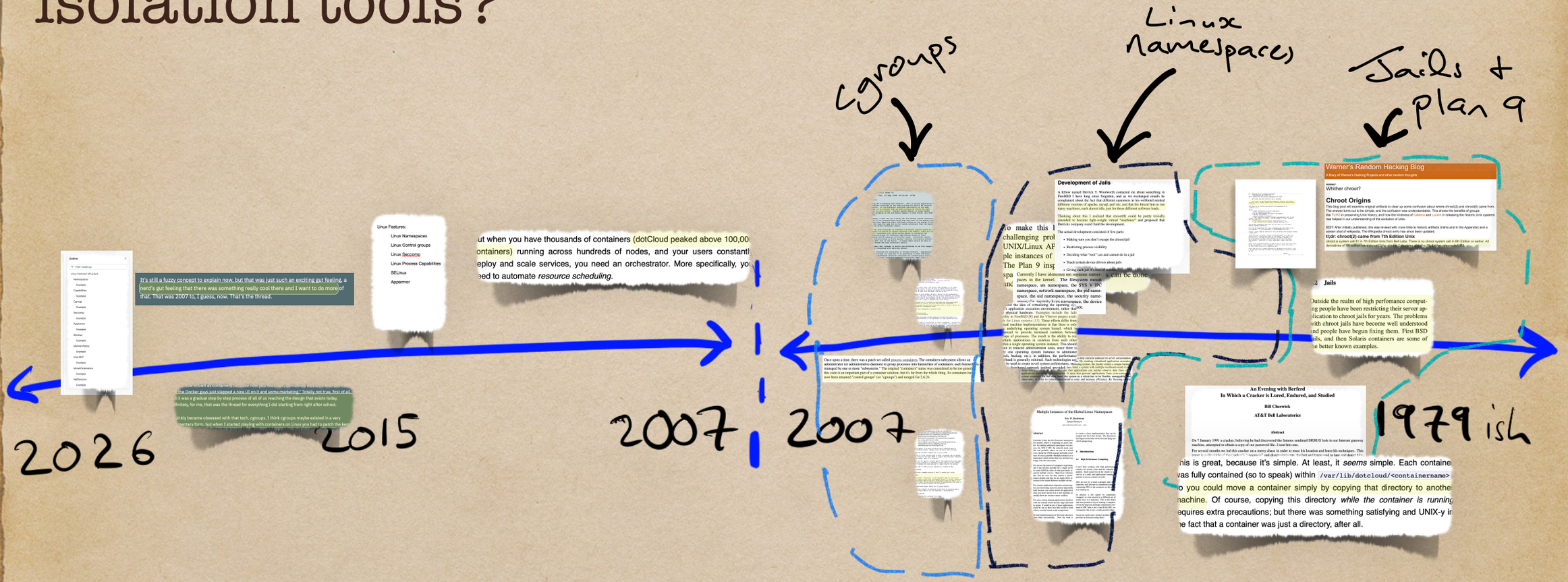
Abstract

On 7 January 1991 a cracker, believing he had discovered the famous sendmail DEBUG hole in our Internet gateway machine, attempted to obtain a copy of our password file. I sent him one.

For several months we led this cracker on a merry chase in order to trace his location and learn his techniques. This paper is a chronicle of the cracker's "successes" and disappointments, the bait and traps used to lure and detect him, and the chroot "jail" we built to watch his activities.

How did Docker build the right thing for isolation tools?

Where did the isolation tools come from?





Gwendolyn James

One on One teaching:

- Linux
- Shell Scripting
- Programming
- Infrastructure as Code
- Deploying agents and applications at scale

gwyn@playtechnique.io

https://playtechnique.io

References

https://media.rainpos.com/108/senate_document_80_clean.pdf - The Senate Report on the Colorado-Big Thomson Project, which created Horsetooth Reservoir

http://www.scotese.com/earth.htm - The Ages of the Earth imagery

https://github.com/opencontainers/runtime-spec/blob/main/features-linux.md - Linux Containers Features Structure

https://github.com/moby/moby/issues/9015 - Docker Registry v2 proposal

https://queue.acm.org/detail.cfm?id=2898444 - Borg, Omega and Kubernetes

References

https://pages.cs.wisc.edu/~stjones/proj/vm_reading/ibmrd2505M.pdf - "The Origin of the VM/370 Time-sharing System"

https://www.heavybit.com/library/podcasts/the-kubelist-podcast/ep-36-the-docker-story-with-solomon-hykes - The Docker Story, kubelist podcast, 2023

https://jpetazzo.github.io/2017/02/24/from-dotcloud-to-docker/ - From dotcloud to docker

https://lore.kernel.org/all/44242A3F.1010307@sw.ru/ - Kirill Korotaev namespaces email

https://lkml.org/lkml/2006/9/19/283 - Rohit Seth introducing proto-cgroups

References

https://lwn.net/Articles/236032/ - let's not call them containers?

https://lwn.net/Articles/256389/ - let's not call them containers!

https://phk.freebsd.dk/pubs/sane2000-jail.pdf "Jails: Confining the Omnipotent Root", Poul-Henning Kamp & Robert Watson

https://www.kernel.org/doc/ols/2006/ols2006v1-pages-101-112.pdf - "Multiple Instances of the Global Linux Namespaces", Eric Biederman, 2006

https://lwn.net/2001/0301/a/namespaces.php3 - the first Linux namespace

https://doc.cat-v.org/plan_9/4th_edition/papers/9, "Plan 9 From Bell Labs"

References

https://phk.freebsd.dk/sagas/jails/ - "Jails – High value but shitty Virtualization" - Poul Henning Kamp

https://bsdimp.blogspot.com/2020/06/whither-chroot.html - "Whither chroot" - Warner Losh

https://www.cheswick.com/ches/papers/berferd.pdf - "An Evening with Berferd", Bill Cheswick