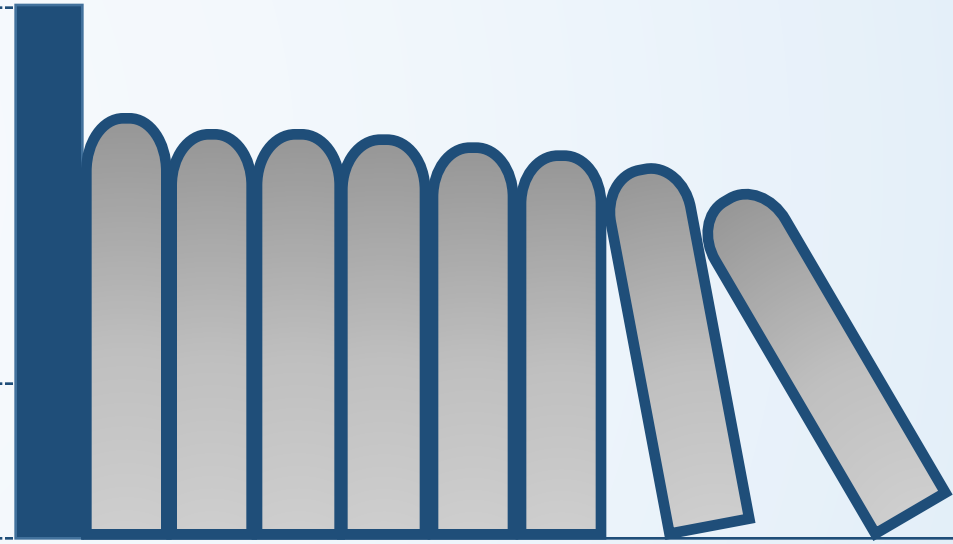
Docker存储

主讲人：宋小金



目录

1 预期收获

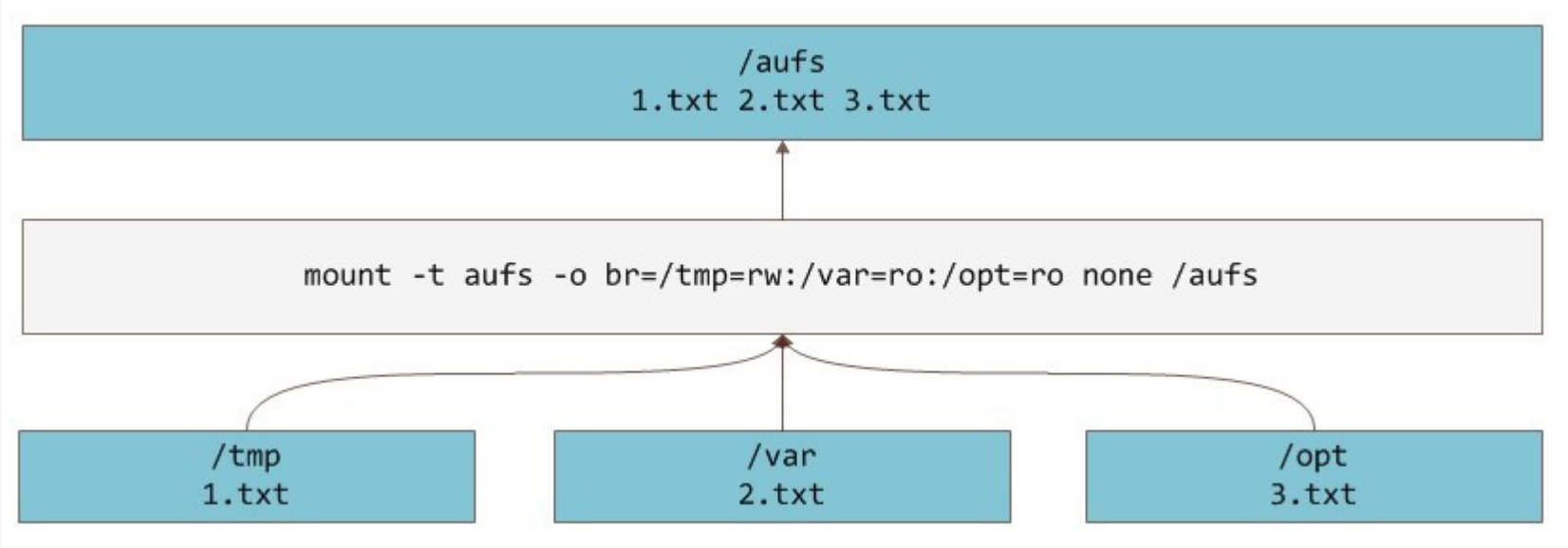
- 分析Storage Driver及选型
- 了解Docker容器安全
- Capabilities权限

Docker Storage Driver分析及选型

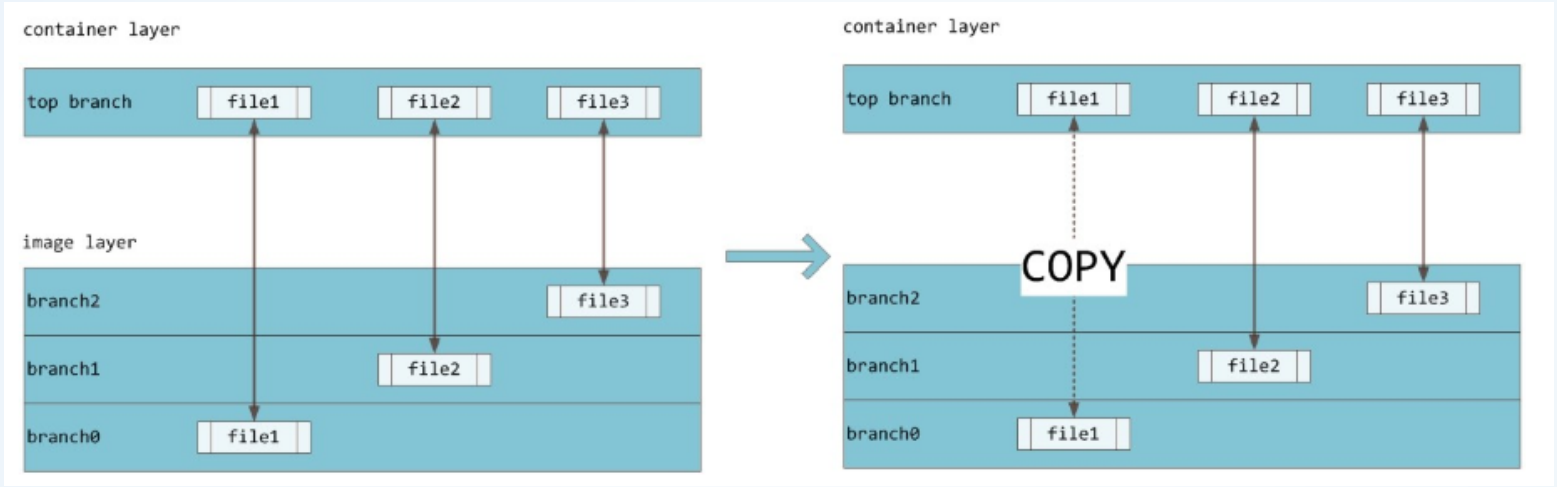
- Aufs
- Btrfs
- Device mapper
- Overlay/Overlay2
- ZFS

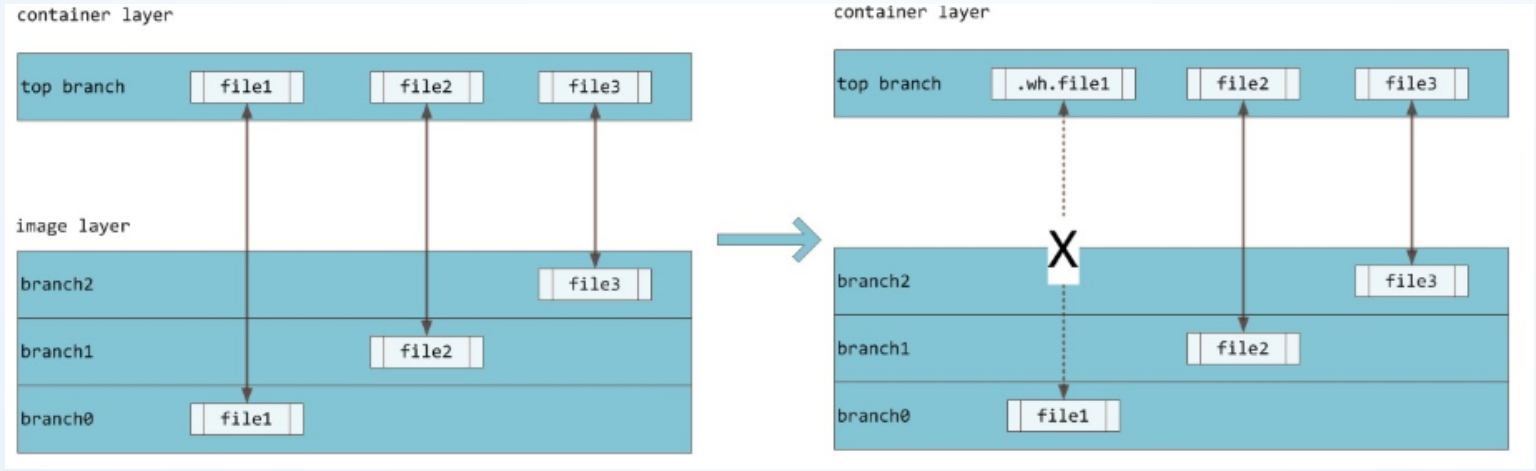
AUFS

AUFS是一种Union File System，所谓UnionFS就是把不同物理位置的目录合并mount到同一个目录中。UnionFS的一个最主要的应用是，把一张CD/DVD和一个硬盘目录给联合 mount在一起，然后，你就可以对这个只读的CD/DVD上的文件进行修改（当然，修改的文件存于硬盘上的目录里）



AUFS的文件操作


container layer
top branch
file1
file2
file3
image layer
branch2
file3
branch1
file2
branch0
file1
container layer
top branch
file1
file2
file3
COPY
branch2
file3
branch1
file2
branch0
file1

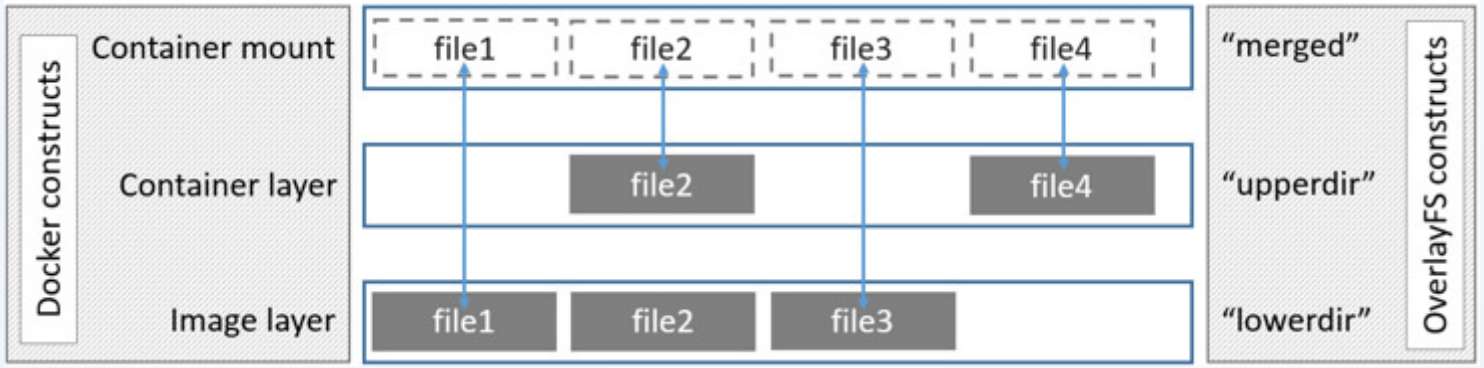

container layer
top branch
file1
file2
file3
image layer
branch2
file3
branch1
file2
branch0
file1
container layer
top branch
.wh.file1
file2
file3
X
branch2
file3
branch1
file2
branch0
file1

AUFS的缺点

- 读取文件，层多，查找性能低，底层只读，浪费空间
- 读写文件，要拷贝到读写层，大文件代价大
- 删除文件，实际是标记隐藏，浪费空间
- AUFS没入内核，仅Ubuntu支持，CentOS等不支持
- 大量文件读写性能差，造成层更多（解决办法:重要数据直接使用 -v 参数挂载）

Overlay/Overlay2

Overlay是一个类AUFS 的现代联合文件系统，更快并实现更简单，Docker为Overlay提供了一个存储驱动程序。 内核提供的文件系统，Overlay和Overlay2是docker提供的存储驱动，OverlayFS将单个Linux主机上的两个目录合并成一个目录。这些目录被称为层，统一过程被称为联合挂载。OverlayFS关联的底层目录称为lowerdir，对应的高层目录称为upperdir。合并过后统一视图称为merged



Overlay/Overlay2

Overlay通过三个概念来实现它的文件系统：一个“下层目录（lower-dir）”，一个“上层目录（upper-dir）”，和一个做为文件系统合并视图的“合并（merged）”目录。受限于只有一个“下层目录”，需要额外的工作来让“下层目录”递归嵌套（下层目录自己又是另外一个overlay的联合），或者按照Docker的实现，将所有位于下层的内容都硬链接到“下层目录”中。正是这种可能潜在的inode爆炸式增长（因为有大量的硬连接）阻碍了很多人采用Overlay

- 内核要求高（4.0以及以上的）
- 与aufs类似，通过使用磁盘上相同的共享库，让分散的容器实现内存共享
- 硬链接的实现方式导致inode耗尽，Overlay2通过每层的lower文件解决了此问题
- 出现的较晚，代码库还比较年轻，成熟度不如其它storage driver高，docker1.12开始支持，docker-ce18.03默认标配

Overlay/Overlay2的读写操作

- 读取文件
  - 文件之存在于container layer中：直接从upperdir中读
  - 文件不在container layer中：那就从lowerdir中读，会耗费一点性能
  - 文件在container layer和image layer中都存在：从upperdir中读文件
- 删除文件和目录：
  - 删除文件：和aufs一样，相应的whiteout文件被创建在upperdir。并不删除容器层(lowerdir)中的文件，whiteout文件屏蔽了它的存在。
  - 删除文件夹：删除一个文件夹时，一个“遮挡目录”(opaque dir)被创建在upperdir中，它的作用与whitout文件一样，屏蔽了lowerdir中文件夹的存在

Overlay/Overlay2的读写操作

- 读写文件
  - 第一次写文件：文件不在container layer(upperdir)中，overlay driver调用Copy-Up操作将文件从lowerdir读到upperdir中，然后对文件的副本做出修改。 Copy-Up操作工作在文件层面，不是块层面（硬连接或直接使用lower层，不是真拷），这意味着对文件的修改需要将整个文件拷贝到upperdir中。索性下面两个事实使这一操作的开销很小
  - copy-up仅发生在文件第一次被修改时，此后对文件的读写都直接在upperdir中进行
  - overlayfs中仅有两层，这使得文件的查找效率很高(相对于aufs)

Docker存在的安全问题

- 内核漏洞（Kernel exploits）攻击内核，影响所有容器
- 有毒镜像（Poisoned images）官方hub出现过挖矿镜像
- Docker漏洞（Docker exploits）Docker Daemon会被攻击
- 隔离不彻底（Namespace）非完全隔离，部分是共享的
- 资源未限制（Cgroup）容器资源独占会影响其它容器
- 限制能力（Capabilities）开放给进程的权限设置不合理

Docker安全的最佳实践（一）

- 限制节点的root访问
  - 用户不需要登录主机，限制有权访问主机的用户数量会减少攻击面
- 禁止远程访问docker daemon
  - 不要启用远程socket，如果必须则使用证书
- 尽可能避免运行Privileged容器
  - 特权容器可以访问主机namespace，有控制主机的权限
- 容器UID管理
  - 在容器中尽量少用root权限，应该在dockerfile中用USER声明运行容器的用户

Docker安全的最佳实践（二）

- 关注镜像安全
  - 只运行信任安全的镜像，数字签名来确认仓库镜像的起源和完整性
- 限制重启
  - 容器如果不停地重启,也会消耗很多资源,造成DoS攻击,这个主要是在启动命令行时使用--restart=on-failure:10这样的参数
- 限制容器的网络
  - 容器应向外暴露尽可能少的端口，容器间的通信最好是需要才是连通的
- 限制资源
  - 防止CPU，内存被耗量，引发Ddos

Docker安全的最佳实践（三）

镜像仓库安全

- 通信安全，SSL证书
- 镜像安全扫描【开源方案/商业方案】
- 镜像签名校验
- 基础镜像自己定制
- 镜像及内部软件基线控制

Docker Capabilities

- Linux Capabilities，Linux把超级用户不同单元的权限分开，可以单独的开启和禁止，称为能力(capability)，可以将能力赋给普通的进程
- 在Docker容器中，配置权限，如修改网卡配置
- 在docker run命令中，可以通过--cap-add和--cap-drop来添加linux Capabilities
- Kubernetes对Pod的定义中，可以add/drop Capabilities在Pod.spec.containers.sercurityContext.capabilities中添加要add的Capabilities list和drop的Capabilities list

默认开启的Capabilities

Docker's capabilities	Linux capabilities	Capability Description
SETPCAP	CAP_SETPCAP	Modify process capabilities.
MKNOD	CAP_MKNOD	Create special files using mknod(2).
AUDIT_WRITE	CAP_AUDIT_WRITE	Write records to kernel auditing log.
CHOWN	CAP_CHOWN	Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	CAP_NET_RAW	Use RAW and PACKET sockets.
DAC_OVERRIDE	CAP_DAC_OVERRIDE	Bypass file read, write, and execute permission checks.
FOWNER	CAP_FOWNER	Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	CAP_FSETID	Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	CAP_KILL	Bypass permission checks for sending signals.
SETGID	CAP_SETGID	Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	CAP_SETUID	Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	CAP_NET_BIND_SERVICE	Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	CAP_SYS_CHROOT	Use chroot(2), change root directory.
SETFCAP	CAP_SETFCAP	Set file capabilities.

默认关闭的Capabilities

Docker's capabilities	Linux capabilities	Capability Description
SYS_MODULE	CAP_SYS_MODULE	Load and unload kernel modules.
SYS_RAWIO	CAP_SYS_RAWIO	Perform I/O port operations (iopl(2) and ioperm(2)).
SYS_PACCT	CAP_SYS_PACCT	Use acct(2), switch process accounting on or off.
SYS_ADMIN	CAP_SYS_ADMIN	Perform a range of system administration operations.
SYS_NICE	CAP_SYS_NICE	Raise process nice value (nice(2), setpriority(2)) and change the nice value for arbitrary processes.
SYS_RESOURCE	CAP_SYS_RESOURCE	Override resource Limits.
SYS_TIME	CAP_SYS_TIME	Set system clock (settimeofday(2), stime(2), adjtimex(2)); set real-time (hardware) clock.
SYS_TTY_CONFIG	CAP_SYS_TTY_CONFIG	Use vhangup(2); employ various privileged ioctl(2) operations on virtual terminals.
AUDIT_CONTROL	CAP_AUDIT_CONTROL	Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.
MAC_OVERRIDE	CAP_MAC_OVERRIDE	Allow MAC configuration or state changes. Implemented for the Smack LSM.
MAC_ADMIN	CAP_MAC_ADMIN	Override Mandatory Access Control (MAC). Implemented for the Smack Linux Security Module (LSM).
NET_ADMIN	CAP_NET_ADMIN	Perform various network-related operations.
SYSLOG	CAP_SYSLOG	Perform privileged syslog(2) operations.
DAC_READ_SEARCH	CAP_DAC_READ_SEARCH	Bypass file read permission checks and directory read and execute permission checks.
LINUX_IMMUTABLE	CAP_LINUX_IMMUTABLE	Set the FS_APPEND_FL and FS_IMMUTABLE_FL i-node flags.

Kubernetes SecurityContext

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
spec:
  containers:
  - name: friendly-container
    image: "alpine:3.4"
    command: ["/bin/echo", "hello", "world"]
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
        drop:
        - KILL
```

课程回顾

已学知识要点

- *了解了Docker Storage Driver及选型*
- *了解Docker容器安全*
- *了解Docker Capabilities*