

第二周作业

1. 基于dockerfile, 实现分层构建的nginx业务镜像
2. 基于docker实现对容器的CPU和内存的资源限制
3. 部署http协议的harbor镜像仓库
4. 基于docker-compose实现对nginx+tomcat web服务的单机编排
5. 扩展作业:

5.1 掌握containerd的安装

5.2 基于nerdctl拉取镜像和创建容器

1. 基于dockerfile, 实现分层构建的nginx业务镜像

```
1 #清除历史容器
2 root@castillo:~# docker stop `docker ps | awk '{if (NR>1){print $1}}'`
3 root@castillo:~# docker rm `docker ps -a | awk '{if (NR>1){print $1}}'`
```

1.1. 编写二进制安装nginx_dockerfile

```
1 FROM ubuntu:22.04
2 MAINTAINER "jinliucastillo"
3
4 ADD sources.list /etc/apt/sources.list
5
6 RUN apt update && apt install -y iproute2 ntpdate tcpdump telnet traceroute nfs-
  kernel-server nfs-common lrzsz tree openssl libssl-dev libpcre3 libpcre3-dev zlib1g-
  dev ntpdate tcpdump telnet traceroute gcc openssh-server lrzsz tree openssl libssl-
  dev libpcre3 libpcre3-dev zlib1g-dev ntpdate tcpdump telnet traceroute iotop unzip
  zip make
7
8 ADD nginx-1.22.1.tar.gz /usr/local/src/
9 RUN cd /usr/local/src/nginx-1.22.1 && ./configure --prefix=/apps/nginx && make &&
  make install && ln -sv /apps/nginx/sbin/nginx /usr/bin
10 RUN groupadd -g 2088 nginx && useradd -g nginx -s /usr/sbin/nologin -u 2088 nginx &&
  chown -R nginx.nginx /apps/nginx
11 ADD nginx.conf /apps/nginx/conf/
12 ADD frontend.tar.gz /apps/nginx/html/
13
14 EXPOSE 80 443
15 CMD ["nginx","-g","daemon off;"]
```

1.2. 构建nginx镜像

```

1 root@castillo:/opt/cicd# docker build -t harbor.jinliu.net/myserver/nginx:v1 .
2 root@castillo:/opt/cicd# docker images
3 REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
4 harbor.jinliu.net/myserver/nginx  v1          5f227fbc596b     About a minute ago  470MB
5 ubuntu               22.04       9d28ccdc1fc7     11 months ago    76.3MB
6 root@castillo:/opt/cicd# docker run -dit -p 80:80 harbor.jinliu.net/myserver/nginx:v1

```

1.3. 提交镜像

```

1 root@castillo:/opt/cicd# docker commit -m "nginx image v2" -a "jinliu castillo" -c
  "EXPOSE 80 443" 2330728e0b6e harbor.jinliu.net/myserver/nginx:v2
2 sha256:e6954854a75e849fc0cb4164da30f2b5dc52d8ac58344e76014eacaf7de2ef3f
3 root@castillo:/opt/cicd# docker images
4 REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
5 harbor.jinliu.net/myserver/nginx  v2          e6954854a75e     6 seconds ago    514MB
6 harbor.jinliu.net/myserver/nginx  v1          5f227fbc596b     21 minutes ago   470MB
7 ubuntu               22.04       9d28ccdc1fc7     11 months ago    76.3MB
8 root@castillo:/opt/cicd#

```

2. 基于docker实现对容器的CPU和内存的资源限制

2.1. 理解

◆ 对于Linux 主机，如果没有足够的内容来执行其他重要的系统任务，将会抛出 OOM (Out of Memory Exception,内存溢出、内存泄漏、内存异常)，随后系统会开始杀死进程以释放内存，凡是运行在宿主机的进程都有可能被 kill，包括Dockerd和其它的应用程序，如果重要的系统进程被Kill,会导致和该进程相关的服务全部宕机。

```

Oct 10 17:09:28 docker-server1 kernel: [57402.792264] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=systemd-
journal.service,mems_allowed=0,global_om,task_memcg=/system.slice/docker-81aca8d9f526f25eea0015af00271475c9e5a794b762fa70dfe7af2f7efaff64.scope,task=stress-ng-
vm,pid=41298,uid=0
Oct 10 17:09:28 docker-server1 kernel: [57402.792303] Out of memory: Killed process 41298 (stress-ng-vm) total-vm:530572kB, anon-rss:314588kB,
file-rss:4kB, shme
m-rss:24kB, UID:0 pgtables:668kB oom_score_adj:1000
Oct 10 17:09:29 docker-server1 kernel: [57403.278032] sshd invoked oom-killer: gfp_mask=0x1100cca(GFP_HIGHUSER_MOVABLE), order=0,
oom_score_adj=0
Oct 10 17:09:29 docker-server1 kernel: [57403.278038] CPU: 0 PID: 39756 Comm: sshd Not tainted 5.15.0-43-generic #46-Ubuntu
Oct 10 17:09:29 docker-server1 kernel: [57403.278040] Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS
6.00 07/22/2020
Oct 10 17:09:29 docker-server1 kernel: [57403.278041] Call Trace:
Oct 10 17:09:29 docker-server1 kernel: [57403.278042] <TASK>
Oct 10 17:09:29 docker-server1 kernel: [57403.278044] show_stack+0x52/0x58
Oct 10 17:09:29 docker-server1 kernel: [57403.278048] dump_stack_lvl+0x4a/0x5f
Oct 10 17:09:29 docker-server1 kernel: [57403.278050] dump_stack+0x10/0x12
Oct 10 17:09:29 docker-server1 kernel: [57403.278052] dump_header+0x53/0x224
Oct 10 17:09:29 docker-server1 kernel: [57403.278054] oom_kill_process.cold+0xb/0x10
Oct 10 17:09:29 docker-server1 kernel: [57403.278055] out_of_memory+0x106/0x2e0
Oct 10 17:09:29 docker-server1 kernel: [57403.278058] __alloc_pages_slowpath.constprop.0+0x97a/0xa40

```

```

1 #查看内核支持情况
2 [root@k8s-c-jl-01 ~]# docker info
3 Client:
4 Context:      default
5 Debug Mode:  false
6 Plugins:
7   app: Docker App (Docker Inc., v0.9.1-beta3)
8   buildx: Docker Buildx (Docker Inc., v0.9.1-docker)
9   scan: Docker Scan (Docker Inc., v0.21.0)
10
11 Server:

```

```
12 Containers: 0
13   Running: 0
14   Paused: 0
15   Stopped: 0
16 Images: 0
17 Server Version: 20.10.21
18 Storage Driver: overlay2
19   Backing Filesystem: xfs
20   Supports d_type: true
21   Native Overlay Diff: true
22   userxattr: false
23 Logging Driver: json-file
24 Cgroup Driver: systemd
25 Cgroup Version: 1
26 Plugins:
27   Volume: local
28   Network: bridge host ipvlan macvlan null overlay
29   Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
30 Swarm: inactive
31 Runtimes: io.containerd.runtime.v1.linux runc io.containerd.runc.v2
32 Default Runtime: runc
33 Init Binary: docker-init
34 containerd version: 1c90a442489720eec95342e1789ee8a5e1b9536f
35 runc version: v1.1.4-0-g5fd4c4d
36 init version: de40ad0
37 Security Options:
38   seccomp
39   Profile: default
40 Kernel Version: 3.10.0-957.el7.x86_64
41 Operating System: CentOS Linux 7 (Core)
42 OSType: linux
43 Architecture: x86_64
44 CPUs: 2
45 Total Memory: 7.795GiB
46 Name: k8s-c-jl-01
47 ID: NA77:3K7L:FKMS:5JW5:6KMC:LOZI:RQ6G:EZMM:ZZFI:BEJL:PGCG:RVDG
48 Docker Root Dir: /var/lib/docker
49 Debug Mode: false
50 Registry: https://index.docker.io/v1/
51 Labels:
52 Experimental: false
53 Insecure Registries:
54   192.168.31.113
55   harbor.jinliu.net
56   127.0.0.0/8
57 Registry Mirrors:
58   https://rlyvli78.mirror.aliyuncs.com/
59 Live Restore Enabled: false
60
61 [root@k8s-c-jl-01 ~]#
```

2.2. 触发条件

- oom的触发条件是：系统无法给应用程序分配物理内存。
- 无法分配到物理内存并不意味着物理内存已耗尽，也可能是其它缘由，如下：

-----内存划分过小，剩余内存都是内存碎片，无法分配一个稍大的内存。

```

1  #物理内存阈值设置
2  [root@k8s-c-j1-01 ~]# cat /proc/sys/vm/min_free_kbytes
3  67584
4  [root@k8s-c-j1-01 ~]#
5
6  #该阈值是Linux最低剩余多少空闲物理内存（Kbytes）给内核使用；当可用物理内存低于这个参数时，系统触发缓存回收机制，以释放内存，直到可用物理内存大于这个值。该值不能设置的过小，原因如下：
7  --当该值设置的过小，并且内存资源使用过量时，没有及时kill进程释放内存资源，如果设置了swap，当物理内存过低，降低内存页的命中率，也会导致内存频繁换页，频繁将内存页写入swap分区（磁盘），导致系统运行卡顿。设置过小也可能出现驱动加载时分配内存失败，无法加载驱动。测试发现：过小会增加系统死机的概率，当内存太少，内核容易出现内存不足以处理异常事件，导致直接死机，无任何反应，例如：串口无任何输出。
8  #该值也不能设置过大，原因如下：
9  --设置过大，会导致应用层能使用的物理内存减少，导致程序被kill掉。

```

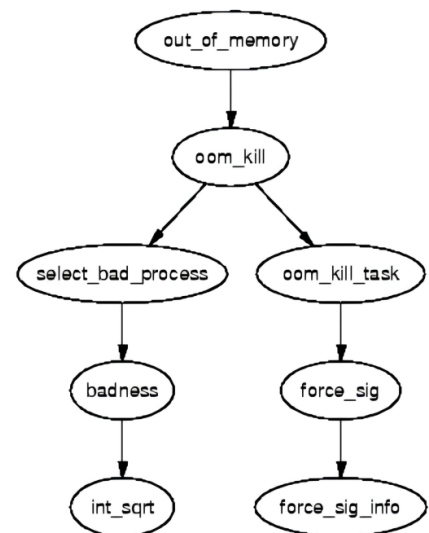
2.3. oom和内存缓存

为尽量避免oom，需应用设计合适逻辑进行内存释放或刷盘。或加入类似redis中间件

2.4. oom优先级机制

◆OOM优先级机制:

- linux会为每个进程算一个分数，最终他会将分数最高的进程kill。
- /proc/PID/oom_score_adj #范围为-1000到1000，值越高越容易被宿主机kill掉，如果将该值设置为-1000，则进程永远不会被宿主机kernel kill。
- /proc/PID/oom_adj #范围为-17到+15，取值越高越容易被干掉，如果是-17，则表示不能被kill，该设置参数的存在是为了和旧版本的Linux内核兼容。
- /proc/PID/oom_score #这个值是系统综合进程的内存消耗量、CPU时间(utime + stime)、存活时间(uptime - start time)和oom_adj计算出的进程得分，消耗内存越多得分越高，越容易被宿主机kernel强制杀死。



◆物理内存限制参数：

- `-m` or `--memory` #限制容器可以使用的最大内存量，如果设置此选项，最小存值为4m（4兆字节）。
- `--memory-swap` #容器可以使用的交换分区大小，必须要在设置了物理内存限制的前提才能设置交换分区的限制
- `--memory-swappiness` #设置容器使用交换分区的倾向性，值越高表示越倾向于使用swap分区，范围为0-100，0为能不用就不用，100为能用就用。
- `--kernel-memory` #容器可以使用的最大内核内存量，最小为4m，由于内核内存与用户空间内存隔离，因此无法与用户空间内存直接交换，因此内核内存不足的容器可能会阻塞宿主机资源，这会对主机和其他容器或者其他服务进程产生影响，因此不要设置内核内存大小。
- `--memory-reservation` #允许指定小于`--memory`的软限制，当Docker检测到主机上的争用或内存不足时会激活该限制，如果使用`--memory-reservation`，则必须将其设置为低于`--memory`才能使其优先。因为它是软限制，所以不能保证容器不超过限制。
- `--oom-kill-disable` #默认情况下，发生OOM时，kernel会杀死容器内进程，但是可以使用`--oom-kill-disable`参数，可以禁止oom发生在指定的容器上，即 仅在已设置`-m` / `--memory`选项的容器上禁用OOM，如果`-m` 参数未配置，产生OOM时，主机为了释放内存还会杀死系统进程。

2.5. 测试内存限制

```
1 root@castillo:/opt/cicd# docker run -it --rm --name jinliu-c1 lorel/docker-stress-ng
--vm 2 --vm-bytes 256M
2 stress-ng: info: [1] defaulting to a 86400 second run per stressor
3 stress-ng: info: [1] dispatching hogs: 2 vm
4
5 root@castillo:~# docker stats
6 CONTAINER ID   NAME          CPU %      MEM USAGE / LIMIT   MEM %      NET I/O
7 5dd8008128b8   jinliu-c1     198.06%    515.8MiB / 3.832GiB  13.15%     796B / 0B
8 2330728e0b6e   sleepy_johnson 0.00%     15.47MiB / 3.832GiB  0.39%     11MB /
110kB  0B / 121MB    2
9
10 root@castillo:~# docker run -it --rm -m 256m --name jinliu-c2 lorel/docker-stress-ng
--vm 2 --vm-bytes 256M
11 Given value bytes is not a valid decimal for the vm option
12 root@castillo:~
13 root@castillo:~# docker run -it --rm -m 256m --name jinliu-c2 lorel/docker-stress-ng
--vm 2 --vm-bytes 256M
14 stress-ng: info: [1] defaulting to a 86400 second run per stressor
15 stress-ng: info: [1] dispatching hogs: 2 vm
16
17
18 root@castillo:~# docker stats
19 CONTAINER ID   NAME          CPU %      MEM USAGE / LIMIT   MEM %      NET I/O
20 2330728e0b6e   sleepy_johnson 0.00%     15.47MiB / 3.832GiB  0.39%     11MB /
110kB  0B / 121MB    2
21 716e0f480798   jinliu-c2     17.54%    255.9MiB / 256MiB   99.95%     656B / 0B
99.5MB / 3.86GB  5
```

2.6. 测试cpu限制

✓ 注：CPU资源限制是将分配给容器的2核心分配到了宿主机每一核心CPU上，也就是容器的总CPU值是在宿主机的每一个核心CPU分配了部分比例。

- Tasks: 288 total, 10 running, 278 sleeping, 0 stopped, 0 zombie
- %Cpu0 : 51.2 us, 0.0 sy, 0.0 ni, 48.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
- %Cpu1 : 26.4 us, 23.4 sy, 0.0 ni, 50.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
- %Cpu2 : 23.1 us, 27.7 sy, 0.0 ni, 49.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
- %Cpu3 : 18.3 us, 31.3 sy, 0.0 ni, 50.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

```
1 root@castillo:~# docker run -it --rm --name jjinliu-c1 --cpus 2 lorel/docker-stress-ng
--cpu 4 --vm 4
2 stress-ng: info: [1] defaulting to a 86400 second run per stressor
3 stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm
4 ^Cstress-ng: info: [1] successful run completed in 91.84s
5 root@castillo:~#
```

2.7. 将容器运行到指定CPU

```
1 root@castillo:~# docker run -it --rm --name jinliu-c1 --cpus 1 --cpuset-cpus 1
lorel/docker-stress-ng --cpu 2 --vm 2
2 stress-ng: info: [1] defaulting to a 86400 second run per stressor
3 stress-ng: info: [1] dispatching hogs: 2 cpu, 2 vm
4
5 root@castillo:~# docker stats
6 CONTAINER ID   NAME          CPU %       MEM USAGE / LIMIT   MEM %       NET I/O
7 2330728e0b6e   sleepy_johnson 0.00%       15.47MiB / 3.832GiB  0.39%       11MB /
110kB  0B / 121MB    2
8 6fbae7fcc895   jinliu-c1      100.35%     519.9MiB / 3.832GiB  13.25%      726B / 0B
0B / 0B      7
9
10 root@castillo:/opt/cicd# top
11 top - 17:08:28 up 10 days,  2:41,  8 users,  load average: 0.67, 1.56, 1.21
12 Tasks: 140 total,  5 running, 135 sleeping,  0 stopped,  0 zombie
13 %Cpu0  :  0.3 us,  0.3 sy,  0.0 ni, 99.0 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
14 %Cpu1  : 85.4 us, 14.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
15 MiB Mem :  3924.0 total, 1197.7 free,   587.1 used, 2139.2 buff/cache
16 MiB Swap:  3925.0 total, 3924.7 free,    0.3 used. 3044.8 avail Mem
17
18      PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
19   32574 root      20   0   6908   2496    672 R   25.0   0.1   0:01.65 stress-ng-cpu
20   32576 root      20   0   6908   2496    672 R   25.0   0.1   0:01.66 stress-ng-cpu
21   32578 root      20   0 268408 150288    540 R   25.0   3.7   0:01.65 stress-ng-vm
22   32579 root      20   0 268408 152136    540 R   25.0   3.8   0:01.65 stress-ng-vm
```

2.8. 对比两个容器cpu限制情况

```
1 #C1
2 root@castillo:~# docker run -it --rm --name jinliu-c1 --cpus 1 --cpu-shares 800
  lorel/docker-stress-ng --cpu 1 --vm 2
3 stress-ng: info: [1] defaulting to a 86400 second run per stressor
4 stress-ng: info: [1] dispatching hogs: 1 cpu, 2 vm
5
6 root@castillo:~# docker stats
7 CONTAINER ID   NAME          CPU %       MEM USAGE / LIMIT   MEM %       NET I/O
8 2330728e0b6e   sleepy_johnson 0.00%       15.47MiB / 3.832GiB  0.39%       11MB /
  110kB  0B / 121MB  2
9 5640562a51c1   jinliu-c1      97.83%      517.8MiB / 3.832GiB  13.20%      796B / 0B
  0B / 0B   6
10
11 root@castillo:~# top
12 top - 17:13:30 up 10 days,  2:46, 10 users,  load average: 1.23, 1.55, 1.34
13 Tasks: 147 total,  5 running, 142 sleeping,  0 stopped,  0 zombie
14 %Cpu(s): 51.0 us,  0.5 sy,  0.0 ni, 48.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
15 MiB Mem :  3924.0 total,  997.4 free,   787.3 used,  2139.3 buff/cache
16 MiB Swap:  3925.0 total,  3924.7 free,    0.3 used.  2844.5 avail Mem
17
18      PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM    TIME+  COMMAND
19    32745 root        20   0 268408 262544   596 R   49.5   6.5   0:37.75 stress-ng-vm
20    32746 root        20   0 268408 262544   596 R   25.2   6.5   0:18.99 stress-ng-vm
21    32742 root        20   0   6908   2432    604 R   24.9   0.1   0:19.01 stress-ng-cpu
22    2596 root        20   0 1356280 58640 35812 S    0.3   1.5   1:34.86 containerd
23    10621 castillo    20   0   17300   8072   5660 S    0.3   0.2   0:01.02 sshd
24
25 #C2
26 root@castillo:~# docker run -it --rm --name jinliu-c2 --cpus 1 --cpu-shares 400
  lorel/docker-stress-ng --cpu 1 --vm 2
27 stress-ng: info: [1] defaulting to a 86400 second run per stressor
28 stress-ng: info: [1] dispatching hogs: 1 cpu, 2 vm
29
30 root@castillo:~# docker stats
31 2330728e0b6e   sleepy_johnson 0.00%       15.47MiB / 3.832GiB  0.39%       11MB /
  110kB  0B / 121MB  2
32 5640562a51c1   jinliu-c1      100.22%     517.8MiB / 3.832GiB  13.20%      866B / 0B
  0B / 0B   6
33 03dd3139eb2d   jinliu-c2      98.85%      314.3MiB / 3.832GiB  8.01%       656B / 0B
```

```

0B / 0B      6
34
35 root@castillo:~# top1
36 top - 17:15:15 up 10 days,  2:47, 10 users,  load average: 3.74, 2.09, 1.54
37 Tasks: 153 total,   7 running, 146 sleeping,   0 stopped,   0 zombie
38 %Cpu(s): 99.0 us,   0.7 sy,   0.0 ni,   0.3 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
39 MiB Mem :  3924.0 total,   466.8 free,  1315.5 used,   2141.7 buff/cache
40 MiB Swap:  3925.0 total,  3924.7 free,    0.3 used.  2314.1 avail Mem
41
42      PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM     TIME+ COMMAND
43    32831 root      20   0  268408 262528    580 R   39.3   6.5   0:15.76 stress-ng-vm
44    32742 root      20   0    6908   2432    604 R   37.7   0.1   0:50.53 stress-ng-cpu
45    32746 root      20   0  268408 262544    596 R   32.3   6.5   0:48.67 stress-ng-vm
46    32745 root      20   0  268408 262544    596 R   30.0   6.5   1:21.84 stress-ng-vm
47    32828 root      20   0    6908   2456    628 R   29.3   0.1   0:12.53 stress-ng-cpu
48    32832 root      20   0  268408 262528    580 R   29.0   6.5   0:12.90 stress-ng-vm
49     4154 root      20   0 1595372  84380 50980 S    0.3   2.1   1:29.97 dockerd
50    31232 root      20   0   712656  11152  8296 S    0.3   0.3   0:03.21 containerd-
shim

```

2.9. systemd验证

```

1 root@castillo:~# docker ps
2 CONTAINER ID   IMAGE                                COMMAND                                  CREATED
3 5565cf04c53b   lorel/docker-stress-ng             "/usr/bin/stress-ng ..."           38
seconds ago    Up 36 seconds                        jinliu-cl
4 2330728e0b6e   harbor.jinliu.net/myserver/nginx:v1 "nginx -g 'daemon of..."           2 hours
ago           Up 2 hours                          0.0.0.0:80->80/tcp, :::80->80/tcp, 443/tcp  sleepy_johnson
5 root@castillo:~# ps -ef |grep nginx
6 root          31251    31232  0 15:31 pts/0    00:00:00 nginx: master process nginx -g
daemon off;
7 nobody        31505    31251  0 15:39 pts/0    00:00:00 nginx: worker process
8 root          33005    32653  0 17:21 pts/8    00:00:00 grep --color=auto nginx
9 root@castillo:~# cat /proc/31251/cpuset
10 /system.slice/docker-

```



```
2330728e0b6e9110151f74b714365945522e66e8c03d723f875e09c112748d6c.scope
11 root@castillo:~# cat /sys/fs/cgroup/system.slice/docker-
2330728e0b6e9110151f74b714365945522e66e8c03d723f875e09c112748d6c.scope/cpu.max
12 max 100000
13 root@castillo:~# cat /sys/fs/cgroup/system.slice/docker-
2330728e0b6e9110151f74b714365945522e66e8c03d723f875e09c112748d6c.scope/memory.max
14 max
15 root@castillo:~#
```

3. 部署http协议的harbor镜像仓库

3.1. 优点

- 基于角色的访问控制：用户与Docker镜像仓库通过“项目”进行项目管理，可以对不同的账户设置不同的权限，以实现权限的精细管控。
- 镜像复制：镜像可以在多个Registry实例中复制（同步），可以实现高性能、高可用的镜像服务。
- 图形化用户界面：用户可以通过浏览器来浏览，管理当前Docker镜像仓库，管理项目和镜像等。
- AD/LDAP 支：Harbor可以集成企业内部已有的AD/LDAP，用于鉴权认证管理。
- 审计管理：所有针对镜像仓库的操作都可以被记录追溯，用于审计管理。
- 国际化：已拥有英文、中文、德文、日文和俄文等多语言支持版本。
- RESTful API：提供给管理员对于Harbor更多的操控,使得与其它管理软件集成变得更容易。
- 部署简单：提供在线和离线两种安装工具，也可以安装到vSphere平台(OVA方式)虚拟设备。

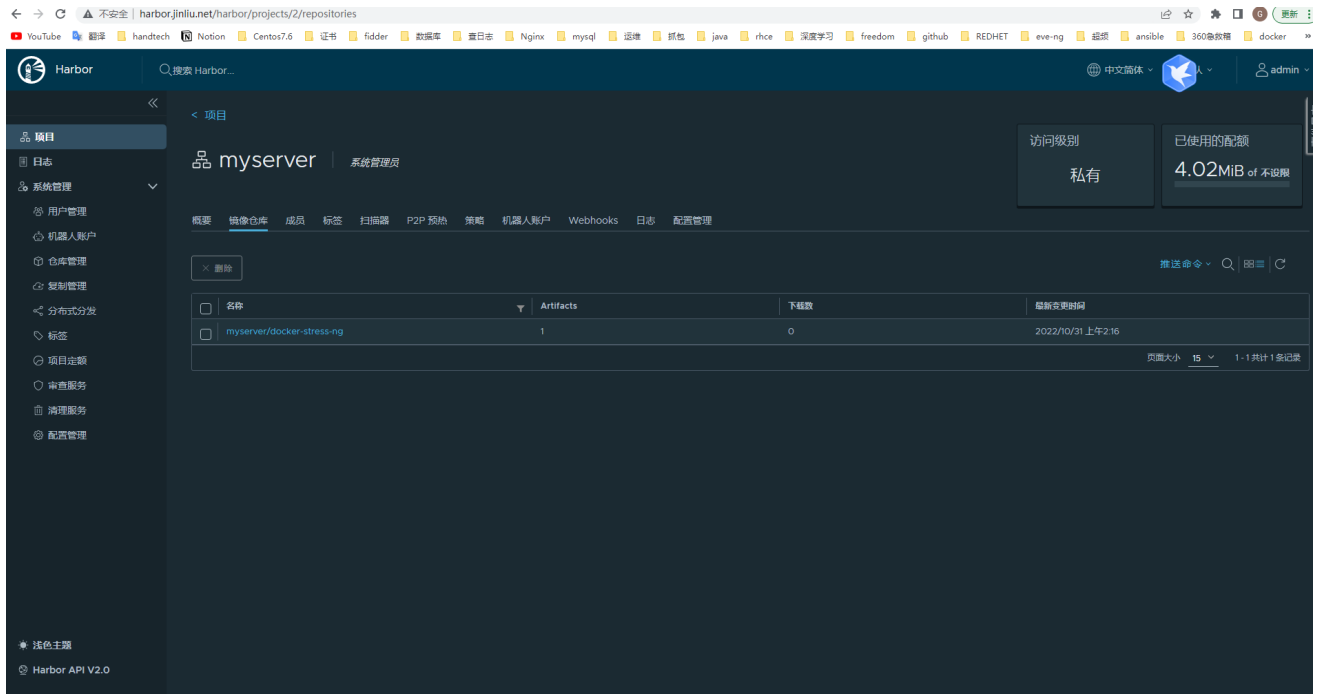
3.2. 安装harbor

```
1 #修改域名
2 root@castillo:/apps# ls -l
3 total 739540
4 -rw-rw-r-- 1 castillo castillo 757278514 Oct 30 14:48 harbor-offline-installer-
v2.6.1.tgz
5 drwxr-xr-x 2 root      root          4096 Oct 30 14:30 nginx
6 root@castillo:/apps# tar -zxf harbor-offline-installer-v2.6.1.tgz
7 root@castillo:/apps# cd harbor
8 root@castillo:/apps/harbor# vim harbor.yml
9 root@castillo:/apps/harbor# grep -A 10 "jinliu" harbor.yml
10 hostname: harbor.jinliu.net
11
12 # http related config
13 http:
14   # port for http, default is 80. If https enabled, this port will redirect to https
   port
15   port: 80
16
17 # https related config
```

```
18 https:
19   # https port for harbor, default is 443
20   port: 443
21 root@castillo:/apps/harbor#
22
23 #配置本地域名解析
24 root@castillo:/apps/harbor# cat /etc/hosts
25 127.0.0.1 localhost
26 127.0.1.1 castillo
27
28 # The following lines are desirable for IPv6 capable hosts
29 ::1          ip6-localhost ip6-loopback
30 fe00::0 ip6-localnet
31 ff00::0 ip6-mcastprefix
32 ff02::1 ip6-allnodes
33 ff02::2 ip6-allrouters
34
35 192.168.31.113 harbor.jinliu.net
36 root@castillo:/apps/harbor#
37
38 #配置域名信任
39 root@castillo:/apps/harbor# cat /etc/docker/daemon.json
40 {
41   "registry-mirrors": ["https://rlyvli78.mirror.aliyuncs.com"],
42   "insecure-registries": ["harbor.jinliu.net", "192.168.31.113"]
43 }
44 root@castillo:/apps/harbor#
45
46 #安装
47 root@castillo:/apps/harbor# bash install.sh
48
49 #登录
50 root@castillo:/apps/harbor# docker login harbor.jinliu.net
51 Username: admin
52 Password:
53 WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
54 Configure a credential helper to remove this warning. See
55 https://docs.docker.com/engine/reference/commandline/login/#credentials-store
56
57 Login Succeeded
58 root@castillo:/apps/harbor#
```

3.3. 推进镜像

```
1 root@castillo:/apps/harbor# docker tag lorel/docker-stress-ng:latest
harbor.jinliu.net/myserver/docker-stress-ng:latest
2 root@castillo:/apps/harbor# docker push harbor.jinliu.net/myserver/docker-stress-
ng:latest
3 The push refers to repository [harbor.jinliu.net/myserver/docker-stress-ng]
4 5f70bf18a086: Pushed
5 ea580b0285fe: Pushed
6 6102f0d2ad33: Pushed
7 latest: digest: sha256:5768a5d8e196be8c8fabbd04d937aabe1407f397b2f12e1fea2573e4b9d9bef
size: 1563
8 root@castillo:/apps/harbor#
```



4. 基于docker-compose实现对nginx+tomcat web服务的单机编排

4.1. docker-compose介绍

- 1 #compose、machine 和 swarm 是docker 原生提供的三大编排工具。简称docker三剑客。
- 2 #Docker Compose能够在 Docker 节点上，以单引擎模式(Single-Engine Mode)进行多容器应用的部署和管理。多数的现代应用通过多个更小的微服务互相协同来组成一个完整可用的应用。
- 3 #部署和管理繁多的服务是困难的。而这正是 Docker Compose 要解决的问题。Docker Compose 并不是通过脚本和各种冗长的 docker 命令来将应用组件组织起来，而是通过一个声明式的配置文件描述整个应用，从而使用一条命令完成部署。应用部署成功后，还可以通过一系列简单的命令实现对其完整声明周期的管理。甚至，配置文件还可以置于版本控制系统中进行存储和管理。

4.1.1. yml文件

```

1  #Docker Compose 的 YAML 文件包含 4 个一级 key:version、services、networks、volumes
2
3  --#version 是必须指定的, 而且总是位于文件的第一行。它定义了 Compose 文件格式 (主要是 API) 的版本。注意, version 并非定义 Docker Compose 或 Docker 引擎的版本号。
4
5  --#services 用于定义不同的应用服务。例如定义了两个服务:一个名为 mysql-r数据库服务以及一个名为 eureka-r的微服。Docker Compose 会将每个服务部署在各自的容器中。
6
7  --#networks 用于指引 Docker 创建新的网络。默认情况下, Docker Compose 会创建 bridge 网络。这是一种单主机网络, 只能够实现同一主机上容器的连接。当然, 也可以使用 driver 属性来指定不同的网络类型。
8
9  --#volumes 用于指引 Docker 来创建新的卷。

```

```

1  #示例
2  version: '3.6' #版本号, https://docs.docker.com/compose/compose-file/compose-versioning/
3  services: #定义服务
4    nginx-server: #当前容器的服务名
5      image: nginx:1.22.0-alpine #镜像名称
6      container_name: nginx-web1 #r容器名称
7      expose: #声明端口映射
8        - 80
9        - 443
10     ports: #定义端口映射
11       - "80:80"
12       - "443:443"

```

4.1.2. 命令

```

1  -f, --file FILE           #指定compsoe 文件, 默认为docker-compose.yml
2  -p, --project-name NAME   #指定功能名称, 默认为当前所在的目录名
3  --profile NAME           #指定一个服务名称, 只执行docker-compose.yml中某些profile 匹配名称的容器
4
5  # docker-compose --profile frontend up -d
6  -c, --context NAME       #指定Dockerfile的文件路径, 也可以是链接到git仓库的url
7  --verbose                #显示更多输出信息
8  --log-level LEVEL        #定义日志级别, DEBUG, INFO, WARNING, ERROR, CRITICAL
9  --ansi (never|always|auto) #定义什么时候显示ANSI控制字符
10 --no-ansi                #不显示ANSI字符, 已废弃
11 -v, --version             #显示docker-compose的版本信息
12 -H, --host HOST           #连接到docker主机, 默认为本机
13 --tls                     #启用tls
14 --tlscacert               #tls ca私钥
15 --tlscert                 #tls ca公钥
16 --tlskey                  #服务私钥
17 --tlsverify               #服务公钥
18 --skip-hostname-check     #不对证书校验 (跳过检查证书签发的主机名)
19 --env-file PATH           #指定文件向容器添加环境变量

```

```

1  #build #构建或重新构建服务、在修改Dockerfile后, 可以通过docker-compose build重新构建镜像并重建服

```

务

```
2 #bundle #从当前docker compose文件生成一个以当前目录为名称的从Compose文件生成一个分布式应用程序捆
   绑包 (DAB)、已废弃
3 --config -q #验证docker-compose.yml文件, 没有错误不输出任何信息
4 #create #创建服务,但是容器不启动
5 --down #停止并删除资源, 默认会删除容器和网络
6 #events #从容器接收实时事件, 可以指定json日志格式, 如
7 # docker-compose events --json
8 #exec #基于service进入指定容器执行命令
9 # docker-compose ps --services
10 # docker exec -it nginx-web1 sh
11 --help #显示帮助细信息
12 --images #列出本地镜像
13 --kill #强制终止运行中的容器
14 # docker-compose kill -s SIGKILL nginx-server #SIG是信号名的通用前缀, KILL是指让一个进程立即
   终止的动作, 合并起来SIGKILL就是发送给一个进程使进程立即终止的信号。
15 --logs #基于service查看容器的日志
16 # docker-compose logs --tail="10" -f nginx-server
17 #pause #暂停服务
18 # docker-compose ps --service
19 # docker-compose pause nginx-server
20 #命令选项, 需要在docker-compose.yml文件目录执行, 带#的不常用
21 #port #基于service查看容器的端口绑定信息
22 # docker-compose port --protocol=tcp nginx-server 80
23 --ps #列出容器信息
24 --pull #拉取镜像
25 #push #上传镜像
26 #restart #重启服务
27 --rm #删除已经停止的服务
28 #run #一次性运行容器, 等于docker run --rm
29 --scale #设置指定服务运行的容器个数
30 # docker-compose scale nginx-server=2 #动态伸缩每个service的副本数, 不能指定容器名和端口映射
31 --start #启动服务
32 # docker-compose start nginx-server
33 --stop #停止服务
34 # docker-compose stop nginx-server
35 --top #显示容器运行状态
36 # docker-compose top
37 --unpause #取消暂定状态中的server
38 # docker-compose unpause nginx-server
39 --up #创建并启动容器, 通常配合-d参数在后台运行容器
40 # docker-compose up -d
41 --version #显示docker-compose版本信息
```

- **ps**: 列出所有运行容器

```
1 docker-compose ps
```

- **logs**: 查看服务日志输出

```
1 docker-compose logs
```

- **port**: 打印绑定的公共端口，下面命令可以输出 eureka 服务 8761 端口所绑定的公共端口

```
1 docker-compose port eureka 8761
```

- **build**: 构建或者重新构建服务

```
1 docker-compose build
```

- **start**: 启动指定服务已存在的容器

```
1 docker-compose start eureka
```

- **stop**: 停止已运行的服务的容器

```
1 docker-compose stop eureka
```

- **rm**: 删除指定服务的容器

```
1 docker-compose rm eureka
```

- **up**: 构建、启动容器

```
1 docker-compose up
```

- **kill**: 通过发送 SIGKILL 信号来停止指定服务的容器

```
1 docker-compose kill eureka
```

- **pull**: 下载服务镜像
- **scale**: 设置指定服务运行容器的个数，以 service=num 形式指定

```
1 docker-compose scale user=3 movie=3
```

- **run**: 在一个服务上执行一个命令

```
1 docker-compose run web bash
```

4.2. nginx+tomcat web服务的单机编排

4.2.1. 准备镜像

```

1 [root@k8s-c-jl-01 ~]# docker images
2 REPOSITORY                                TAG          IMAGE ID      CREATED        SIZE
3 harbor.jinliu.net/myserver/tomcat         v1           fb5657adc892  10 months ago  680MB
4 harbor.jinliu.net/myserver/nginx          1.20.1       c8d03f6b8b91  12 months ago  133MB
5 [root@k8s-c-jl-01 ~]#

```

4.2.2. 准备文件以及挂载目录

```

1 [root@k8s-c-jl-01 docker-compose]# tree .
2 .
3 ├── docker-compose.yml
4 └── nginx
5     ├── nginx.conf
6     └── www
7         ├── images
8         │   └── 1.jpg
9         └── index.html
10
11 3 directories, 4 files
12 [root@k8s-c-jl-01 docker-compose]#

```

4.2.3. docker-compose.yml

```

1 ##version版本设置很重要!!!! 需要和docker-compose版本一致, 否则docker-compose无法解析。例如
2 docker-compose版本为1.18.0, 那么docker-compose.yml中version需要设置成'2'。
3 [root@k8s-c-jl-01 docker-compose]# vim docker-compose.yml
4 version: '2'
5 services:
6   tomcat01:
7     container_name: tomcat01
8     image: harbor.jinliu.net/myserver/tomcat:v1
9     restart: always
10    ports:
11      - 8080
12  tomcat02:
13    container_name: tomcat02
14    image: harbor.jinliu.net/myserver/tomcat:v1
15    restart: always
16    ports:
17      - 8080
18  nginx:
19    container_name: www-nginx
20    image: harbor.jinliu.net/myserver/nginx:1.20.1
21    restart: always
22    ports:
23      - 80:80
24    volumes:
25      - /opt/docker-compose/nginx/www:/usr/share/nginx/html/
26      - /opt/docker-compose/nginx/nginx.conf:/etc/nginx/nginx.conf
27    links:
28      - tomcat01

```

4.2.4. nginx.conf

```
1 [root@k8s-c-jl-01 nginx]# cat nginx.conf | grep -v "#" |grep -v ^$
2 worker_processes 2;
3 events {
4     worker_connections 1024;
5 }
6 http {
7     include mime.types;
8     default_type application/octet-stream;
9     sendfile on;
10    keepalive_timeout 65;
11    upstream tomcat {
12        server tomcat01:8080;
13        server tomcat02:8080;
14    }
15    server {
16        listen 80;
17        server_name localhost;
18        location / {
19            root html;
20            index index.html index.htm;
21        }
22        location /myapp {
23            proxy_pass http://tomcat;
24        }
25        error_page 500 502 503 504 /50x.html;
26        location = /50x.html {
27            root html;
28        }
29    }
30 }
31 [root@k8s-c-jl-01 nginx]#
```

4.2.5. 启动部署


```

1 [root@k8s-c-j1-01 docker-compose]# docker-compose up -d
2 Creating tomcat02 ... done
3 Creating www-nginx ... done
4 Creating tomcat02 ...
5 Creating www-nginx ...
6 [root@k8s-c-j1-01 docker-compose]# docker-compose ps
7      Name                                Command                                State                                Ports
8  -----
9 tomcat01      catalina.sh run                                Up                                0.0.0.0:49153-
>8080/tcp,:::49153->8080/tcp
10 tomcat02      catalina.sh run                                Up                                0.0.0.0:49154-
>8080/tcp,:::49154->8080/tcp
11 www-nginx     /docker-entrypoint.sh nginx ...      Up                                0.0.0.0:80->80/tcp,:::80->80/tcp

```

4.2.6. 测试页面

```

1 #由于没有资源所有访问404，但实际部署成功。
2 [root@k8s-c-j1-01 docker-compose]# ip -4 a |grep "eth0"
3 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
  default qlen 1000
4      inet 192.168.31.127/24 brd 192.168.31.255 scope global noprefixroute eth0

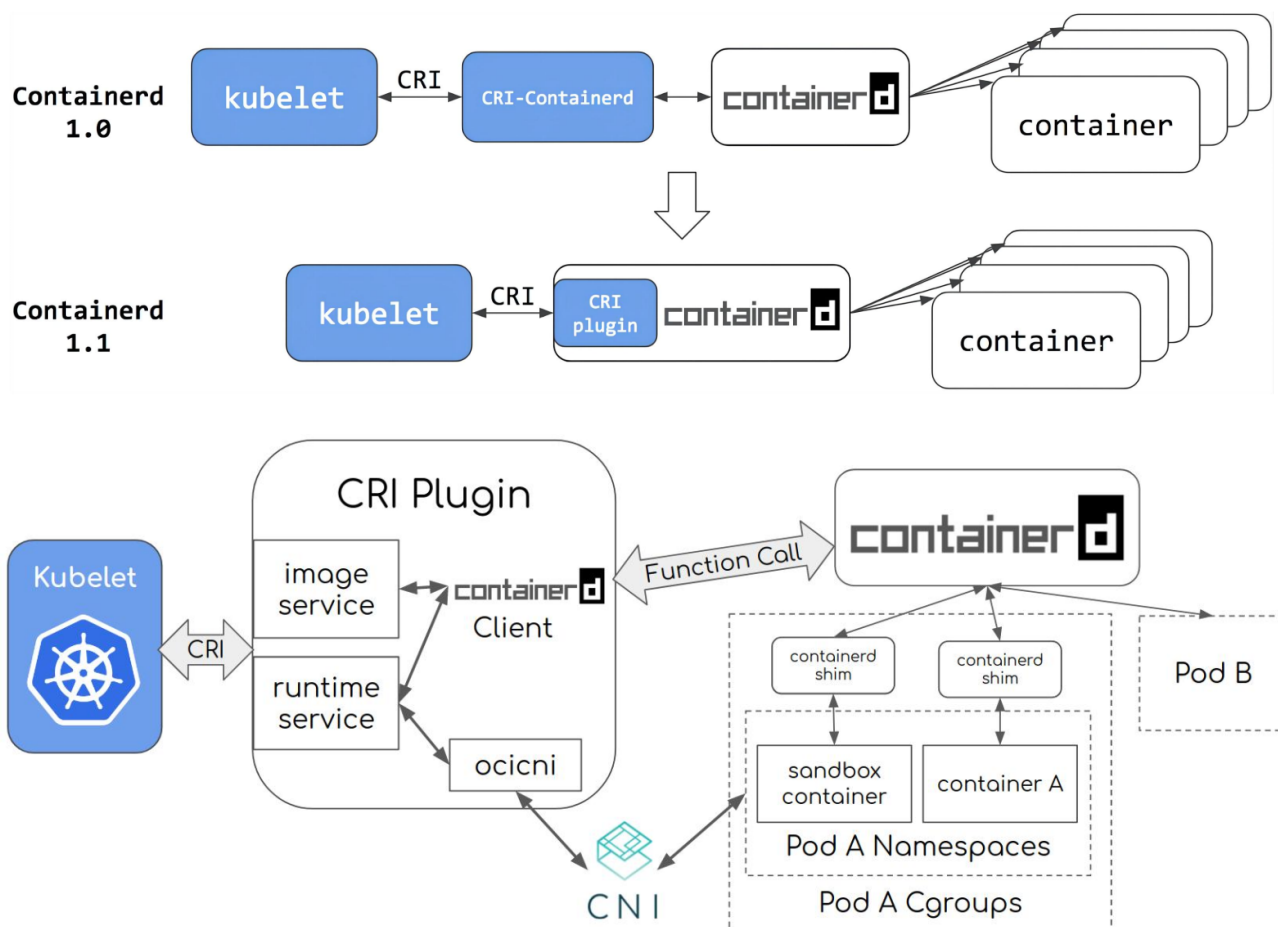
```



5. 掌握containerd的安装

5.1. 简介

- 1 #Containerd在v1.0及以前版本(目前最新v1.6.8)将dockershim和docker daemon替换为cri-containerd+containerd，而在1.1版本时直接将cri-containerd内置在Containerd中，简化为一个CRI插件，用于实现和kubelet的对接。
- 2 #Containerd内置的CRI(container runtime interface)插件实现了Kubelet CRI接口中的Image Service和Runtime Service，通过内部接口管理容器和镜像，并通过CNI(container network interface)插件给Pod配置网络。



5.2. centos安装containerd

```

1 [root@k8s-c-jl-01 docker-compose]# yum list containerd.io --showduplicates
2 Loaded plugins: fastestmirror
3 Loading mirror speeds from cached hostfile
4 * base: mirrors.aliyun.com
5 * epel: ftp.riken.jp
6 * extras: mirrors.aliyun.com
7 * updates: mirrors.aliyun.com
8 Installed Packages
9 containerd.io.x86_64
10                                     1.6.9-3.1.el7
11                                     @docker-ce-stable
12 Available Packages
13 containerd.io.x86_64
14                                     1.2.0-1.2.beta.2.el7
15                                     docker-ce-stable
16 containerd.io.x86_64
17                                     1.2.0-2.0.rc.0.1.el7
18                                     docker-ce-stable
19 containerd.io.x86_64
20                                     1.2.0-2.2.rc.2.1.el7
21                                     docker-ce-stable
22 containerd.io.x86_64

```

		1.2.0-3.el7	
			docker-ce-stable
15	containerd.io.x86_64	1.2.2-3.el7	
			docker-ce-stable
16	containerd.io.x86_64	1.2.2-3.3.el7	
			docker-ce-stable
17	containerd.io.x86_64	1.2.4-3.1.el7	
			docker-ce-stable
18	containerd.io.x86_64	1.2.5-3.1.el7	
			docker-ce-stable
19	containerd.io.x86_64	1.2.6-3.3.el7	
			docker-ce-stable
20	containerd.io.x86_64	1.2.10-3.2.el7	
			docker-ce-stable
21	containerd.io.x86_64	1.2.13-3.1.el7	
			docker-ce-stable
22	containerd.io.x86_64	1.2.13-3.2.el7	
			docker-ce-stable
23	containerd.io.x86_64	1.3.7-3.1.el7	
			docker-ce-stable
24	containerd.io.x86_64	1.3.9-3.1.el7	
			docker-ce-stable
25	containerd.io.x86_64	1.4.3-3.1.el7	
			docker-ce-stable
26	containerd.io.x86_64	1.4.3-3.2.el7	
			docker-ce-stable
27	containerd.io.x86_64	1.4.4-3.1.el7	
			docker-ce-stable
28	containerd.io.x86_64	1.4.6-3.1.el7	
			docker-ce-stable
29	containerd.io.x86_64	1.4.8-3.1.el7	
			docker-ce-stable
30	containerd.io.x86_64	1.4.9-3.1.el7	
			docker-ce-stable
31	containerd.io.x86_64		

```

1.4.10-3.1.el7
docker-ce-stable
32 containerd.io.x86_64
1.4.11-3.1.el7
docker-ce-stable
33 containerd.io.x86_64
1.4.12-3.1.el7
docker-ce-stable
34 containerd.io.x86_64
1.4.13-3.1.el7
docker-ce-stable
35 containerd.io.x86_64
1.5.10-3.1.el7
docker-ce-stable
36 containerd.io.x86_64
1.5.11-3.1.el7
docker-ce-stable
37 containerd.io.x86_64
1.6.4-3.1.el7
docker-ce-stable
38 containerd.io.x86_64
1.6.6-3.1.el7
docker-ce-stable
39 containerd.io.x86_64
1.6.7-3.1.el7
docker-ce-stable
40 containerd.io.x86_64
1.6.8-3.1.el7
docker-ce-stable
41 containerd.io.x86_64
1.6.9-3.1.el7
docker-ce-stable
42 [root@k8s-c-jl-01 docker-compose]# yum install containerd.io-1.6.9 -y

```

5.3. 修改默认配置

```

1 [root@k8s-c-jl-01 docker-compose]# containerd config default >
/etc/containerd/config.toml
2 [root@k8s-c-jl-01 docker-compose]# cat /etc/containerd/config.toml |grep -E -n
'sandbox_|docker\.io|rlyvli78'
3 61:     sandbox_image = "registry.aliyuncs.com/google_containers/pause:3.7"
4 154:     [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
5 155:         endpoint = ["https://rlyvli78.mirror.aliyuncs.com"]
6 [root@k8s-c-jl-01 docker-compose]#

```

5.4. 启动

```

1 [root@k8s-c-jl-01 docker-compose]# systemctl enable containerd --now
2 Created symlink from /etc/systemd/system/multi-user.target.wants/containerd.service to
/usr/lib/systemd/system/containerd.service.
3 [root@k8s-c-jl-01 docker-compose]#

```

5.5. 更新runc

```
1 [root@k8s-c-j1-01 ~]# wget
   https://github.com/opencontainers/runc/releases/download/v1.1.4/runc.amd64
2 [root@k8s-c-j1-01 ~]# > yes |cp -rf runc.amd64 /usr/bin/runc
3 cp: overwrite '/usr/bin/runc'? [root@k8s-c-j1-01 ~]#
4 [root@k8s-c-j1-01 ~]# chmod a+x /usr/bin/runc
5 [root@k8s-c-j1-01 ~]# runc -v
6 runc version 1.1.4
7 commit: v1.1.4-0-g5fd4c4d
8 spec: 1.0.2-dev
9 go: go1.18.7
10 libseccomp: 2.3.1
11 [root@k8s-c-j1-01 ~]#
```

5.6. 客户端使用

1 #containerd相比docker多了一个命名空间的逻辑概念，ctr命令默认是在default命名空间里，当使用crictrl命令的时候，都是在k8s.io这个命名空间里的，所以不指定namespace会发现看到的镜像、容器等内容不一样。

5.6.1. ctr

```
1 #拉取镜像
2 [root@k8s-c-j1-01 ~]# ctr images pull docker.io/library/nginx:1.20.2
3 docker.io/library/nginx:1.20.2:
   resolved          |+++++|
4 index-sha256:38f8c1d9613f3f42e7969c3b1dd5c3277e635d4576713e6453c6193e66270a6d:
   done              |+++++|
5 manifest-sha256:a76df3b4f1478766631c794de7ff466aca466f995fd5bb216bb9643a3dd2a6bb:
   done              |+++++|
6 layer-sha256:e8750203e98541223fb970b2b04058aae5ca11833a93b9f3df26bd835f66d223:
   done              |+++++|
7 layer-sha256:50836501937ff210a4ee8eedcb17b49b3b7627c5b7104397b2a6198c569d9231:
   done              |+++++|
8 config-sha256:0584b370e957bf9d09e10f424859a02ab0fda255103f75b3f8c7d410a4e96ed5:
   done              |+++++|
9 layer-sha256:d838e0361e8efc1fb3ec2b7aed16ba935ee9b62b6631c304256b0326c048a330:
   done              |+++++|
10 layer-sha256:214ca5fb90323fe769c63a12af092f2572bf1c6b300263e09883909fc865d260:
   done              |+++++|
11 layer-sha256:fcc7a415e354b2e1a2fcf80005278d0439a2f87556e683bb98891414339f9bee:
   done              |+++++|
12 layer-sha256:dc73b4533047ea21262e7d35b3b2598e3d2c00b6d63426f47698fe2adac5b1d6:
   done              |+++++|
13 elapsed: 13.4s
   total: 24.2 M (1.8 MiB/s)
14 unpacking linux/amd64
```

```

sha256:38f8c1d9613f3f42e7969c3b1dd5c3277e635d4576713e6453c6193e66270a6d...
15 done: 3.164961912s
16 [root@k8s-c-jl-01 ~]# ctr images ls
17 REF                                     TYPE
    DIGEST                                     SIZE
    PLATFORMS
    LABELS
18 docker.io/library/nginx:1.20.2
   application/vnd.docker.distribution.manifest.list.v2+json
   sha256:38f8c1d9613f3f42e7969c3b1dd5c3277e635d4576713e6453c6193e66270a6d 54.1 MiB
   linux/386,linux/amd64,linux/arm/v5,linux/arm/v7,linux/arm64/v8,linux/mips64le,linux/ppc64le,linux/s390x -
19 [root@k8s-c-jl-01 ~]#
20
21 #运行容器并使用宿主机网络
22 [root@k8s-c-jl-01 docker-compose]# ctr run -t --net-host
   docker.io/library/nginx:1.20.2 test-container
23 /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
   configuration
24 /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
25 /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
26 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
   /etc/nginx/conf.d/default.conf
27 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
   /etc/nginx/conf.d/default.conf
28 /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
29 /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
30 /docker-entrypoint.sh: Configuration complete; ready for start up
31 2022/11/05 06:58:38 [notice] 1#1: using the "epoll" event method
32 2022/11/05 06:58:38 [notice] 1#1: nginx/1.20.2
33 2022/11/05 06:58:38 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
34 2022/11/05 06:58:38 [notice] 1#1: OS: Linux 3.10.0-957.el7.x86_64
35 2022/11/05 06:58:38 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1024:1024
36 2022/11/05 06:58:38 [notice] 1#1: start worker processes
37 2022/11/05 06:58:38 [notice] 1#1: start worker process 31
38 2022/11/05 06:58:38 [notice] 1#1: start worker process 32

```

5.6.2. nerdctl

```

1 #一个兼容docker的containerd的客户端,https://github.com/containerd/nerdctl

```

```

1 [root@k8s-c-jl-01 ~]# wget
   https://github.com/containerd/nerdctl/releases/download/v0.23.0/nerdctl-0.23.0-linux-
   amd64.tar.gz
2 [root@k8s-c-jl-01 ~]# tar xvf nerdctl-0.23.0-linux-amd64.tar.gz -C /usr/bin/
3 nerdctl
4 containerd-rootless-setuptool.sh
5 containerd-rootless.sh
6 [root@k8s-c-jl-01 ~]#

```

5.6.3. cni

```

1 [root@k8s-c-jl-01 ~]# wget
https://github.com/containernetworking/plugins/releases/download/v1.1.1/cni-plugins-
linux-amd64-v1.1.1.tgz
2 [root@k8s-c-jl-01 ~]# mkdir /opt/cni/bin -pv
3 mkdir: created directory '/opt/cni'
4 mkdir: created directory '/opt/cni/bin'
5 [root@k8s-c-jl-01 ~]# tar xvf cni-plugins-linux-amd64-v1.1.1.tgz -C /opt/cni/bin/
6 ./
7 ./macvlan
8 ./static
9 ./vlan
10 ./portmap
11 ./host-local
12 ./vrf
13 ./bridge
14 ./tuning
15 ./firewall
16 ./host-device
17 ./sbr
18 ./loopback
19 ./dhcp
20 ./ptp
21 ./ipvlan
22 ./bandwidth
23 [root@k8s-c-jl-01 ~]#

```

5.6.4. 创建容器并指定端口

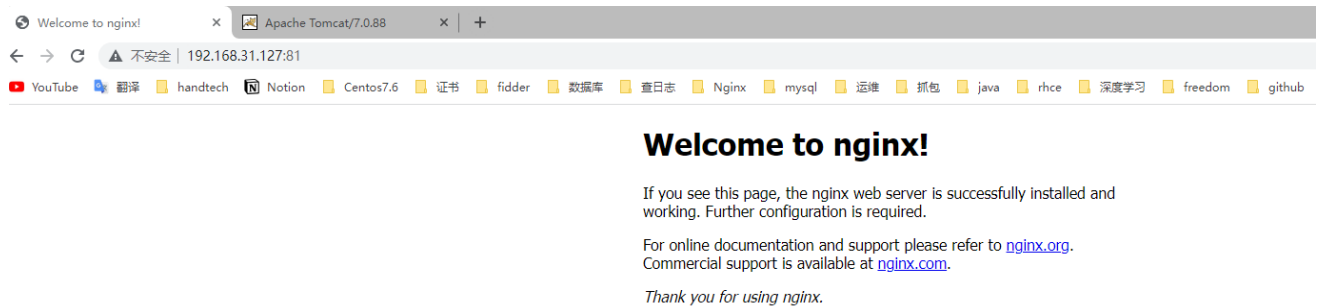
```

1 #nginx
2 [root@k8s-c-jl-01 ~]# nerdctl run -d -p 81:80 --name=nginx-web1 --restart=always
nginx:1.20.2
3 0ca0d45033844bf05df524ed5f4fdd214e41085f75e828451f4d81410f2c0663
4 [root@k8s-c-jl-01 ~]# nerdctl ps
5 CONTAINER ID      IMAGE                                     COMMAND                                CREATED
6 0ca0d4503384      docker.io/library/nginx:1.20.2         "/docker-entrypoint...."             9 seconds
ago      Up      0.0.0.0:81->80/tcp      nginx-web1
7
8 #tomcat
9 [root@k8s-c-jl-01 ~]# nerdctl run -d -p 8080:8080 --name=tomcat-web --restart=always
tomcat:7.0.88-alpine
10 docker.io/library/tomcat:7.0.88-alpine:
resolved      |+++++|
11 index-sha256:fbea9924c4e324b2eee05fbb4317d0fd768f15387f8c77a5d6ee4c4cbdc5a0bc:
done          |+++++|
12 manifest-sha256:3a1bfd26628b05080841b61ce33a581b137243ef77967a86a7e0ab06148d6a8f:
done          |+++++|
13 config-sha256:2c15647bb9a382cf774b7906e6489e2f5491ac6e66593c6a66d6c0d61989c8f1:
done          |+++++|
14 layer-sha256:44e6d293805b58dc5838472788c76c025026ac2c14044d598127d06066a25d5f:
done          |+++++|
15 layer-sha256:cbee32cff593b645ca1694f052ad0545fe5a7bfd1fdc9087624013627ddb041c:

```

```
done | ++++++ |
16 layer-sha256:6334d5b42b354493d40f92ab8d15c307273f5ff11f70421d1bddacc91416adb8:
done | ++++++ |
17 layer-sha256:911c6d0c7995e5d9763c1864d54fb6deccda04a55d7955123a8e22dd9d44c497:
done | ++++++ |
18 layer-sha256:3ae691a546b360850dc64c3f25ebab76406ea37a6abaf5a00109627dcfea839f:
done | ++++++ |
19 layer-sha256:4001add52a901363833fa030b1451162d380a5423906a45f64e090a5ee9efb74:
done | ++++++ |
20 elapsed: 18.1s
total: 73.8 M (4.1 MiB/s)
21 5f22d4ea208ecb3ad072657f54b42355113308a6f983f1eb8e5a08f7e7e55e27
22 [root@k8s-c-jl-01 ~]# nerdctl ps
23 CONTAINER ID      IMAGE                                     COMMAND
24 0ca0d4503384      docker.io/library/nginx:1.20.2         "/docker-entrypoint...." 7
minutes ago      Up          0.0.0.0:81->80/tcp      nginx-web1
25 5f22d4ea208e      docker.io/library/tomcat:7.0.88-alpine  "catalina.sh run"
31 seconds ago    Up          0.0.0.0:8080->8080/tcp  tomcat-web
26 [root@k8s-c-jl-01 ~]#
```

5.6.5. 验证




Welcome to nginx!Apache Tomcat/7.0.88

← → ↻ 不安全 | 192.168.31.127:8080

YouTube 翻译 handtech Notion CentOS7.6 证书 fiddler 数据库 日志 Nginx mysql 运维 抓包 java rhce 深度学习 freedom github REDHET


Home Documentation Configuration Examples Wiki Mailing ListsFind Help

Apache Tomcat/7.0.88



SOFTWARE FOUNDATION
http://www.apache.org/

If you're seeing this, you've successfully installed Tomcat. Congratulations!



Recommended Reading:

[Security Considerations HOW-TO](#)[Manager Application HOW-TO](#)[Clustering/Session Replication HOW-TO](#)

Server Status

Manager App

Host Manager

Developer Quick Start

[Tomcat Setup](#)[First Web Application](#)

[Realms & AAA](#)[JDBC DataSources](#)

[Examples](#)

[Servlet Specifications](#)[Tomcat Versions](#)

Managing Tomcat

For security, access to the [manager webapp](#) is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 7.0 access to the manager application is split between different users.
[Read more...](#)

[Release Notes](#)[Changelog](#)[Migration Guide](#)[Security Notices](#)

Documentation

[Tomcat 7.0 Documentation](#)[Tomcat 7.0 Configuration](#)[Tomcat Wiki](#)

Find additional important configuration information in:

```
$CATALINA_HOME/WORKING.txt
```

Developers may be interested in:

[Tomcat 7.0 Bug Database](#)[Tomcat 7.0 JavaDocs](#)[Tomcat 7.0 SVN Repository](#)

Getting Help

FAQ and Mailing Lists

The following mailing lists are available:

[tomcat-announce](#)
Important announcements, releases, security vulnerability notifications. (Low volume).

[tomcat-users](#)
User support and discussion

[taglibs-user](#)
User support and discussion for [Apache Taglibs](#)

[tomcat-dev](#)
Development mailing list, including commit messages

Other Downloads

[Tomcat Connectors](#)[Tomcat Native](#)[Taglibs](#)[Deployer](#)

Other Documentation

[Tomcat Connectors mod_jk Documentation](#)[Tomcat Native](#)[Deployer](#)

Get Involved

[Overview](#)[SVN Repositories](#)[Mailing Lists](#)[Wiki](#)

Miscellaneous

[Contact](#)[Legal](#)[Sponsorship](#)[Thanks](#)

Apache Software Foundation

[Who We Are](#)[Heritage](#)[Apache Home](#)[Resources](#)

Copyright ©1999-2022 Apache Software Foundation All Rights Reserved

S