基于Docker的微服务自动化测试

第一天：Docker微服务测试启动篇

> 核心内容：docker run

主要内容：

- Docker是什么
- 测试工程师为什么要使用Docker
- 如何安装Docker
- Docker中的仓库、镜像、容器
- Docker常用命令

了解docker吗?

微服务架构 -> 建立docker基础上

对软件测试提出新的挑战

本次训练营：

- 自动化构建多功能、分布式的而测试环境
- 容器和测试，为开发团队交付结果进行质量保障

收获：

1. 构建selenium测试环境

   特点：

   1. 分布式执行
   2. 视频回放

   ```
   docker-compose up
   ```

2. 容器化测试技术

   通过docker 去测试 docker

   ```
   docker-compose up --exit-code-from beifan
   ```

训练营的安排

基于docker自动化测试，大致分为三个阶段：

- 把项目装进docker，测试docker中的服务
- 把用例装进docker，测试外部服务
- 项目和用例都装进docker，用docker测试docker

第一天：

- docker是什么
- docker怎么安装
- docekr常用命令

第二天：

- docker镜像怎么来的
- dockerfile语法
- API接口从开发、测试、部署全过程

第三天：

- docker容器通信方式
- docker-compose编排容器
- 容器编排：微服务测试架构

> 核心内容： docker run

1. Docker

> 利用 Docker 的快速传送、测试和部署代码

举个例子：

搭建测试环境：

1. 搞一台服务器
2. 安装操作系统
3. 安装依赖服务（数据库、缓存、文件存储、日志收集、数据分析、。。。）
4. 上传项目代码
5. 调整配置 （麻烦事最多）
6. 项目启动成功

有了docker：

1. 搞一台服务器
2. 安装docker
3. 启动docker， 完成

可用把docker 看做一个虚拟机，把软件和软件依赖，统一安装其中，实现：

一次部署，到处运行

2. docker 和软件测试有什么关系

微服务 + 容器化 == 大厂标配

==========

开发 -> 测试 -> 运维

==========

1. 开发团队交付是docker，测试团队要对docker进行测试
2. 快速搭建测试环境

3. 安装Docker

1. 虚拟机 请用 ubuntu 20.04

一键安装脚本

```
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
```

2. 腾讯云

腾讯云提供Ubuntu + Docker的镜像

点击购买分分钟就绪：https://curl.qcloud.com/Znt2mOk2

3. 设置国内镜像

docker 和github 类似，主站都国外

```
vim /etc/docker/daemon.json
```

```
{
    "registry-mirrors": [
        "https://registry.docker-cn.com",
        "https://kfwkfulq.mirror.aliyuncs.com",
        "https://pee6w651.mirror.aliyuncs.com",
        "https://2lqq34jg.mirror.aliyuncs.com"
    ]
}
```

4. 免去sudo

需要root权限，但是如果使用sudo 命令，环境变量会变化

```
sudo groupadd docker              # 创建新用户组
sudo usermod -aG docker $USER     # 当前用户加入用户组
newgrp docker                     # 刷新用户组权限
```

5. hello world

```
docker run hello-world
```

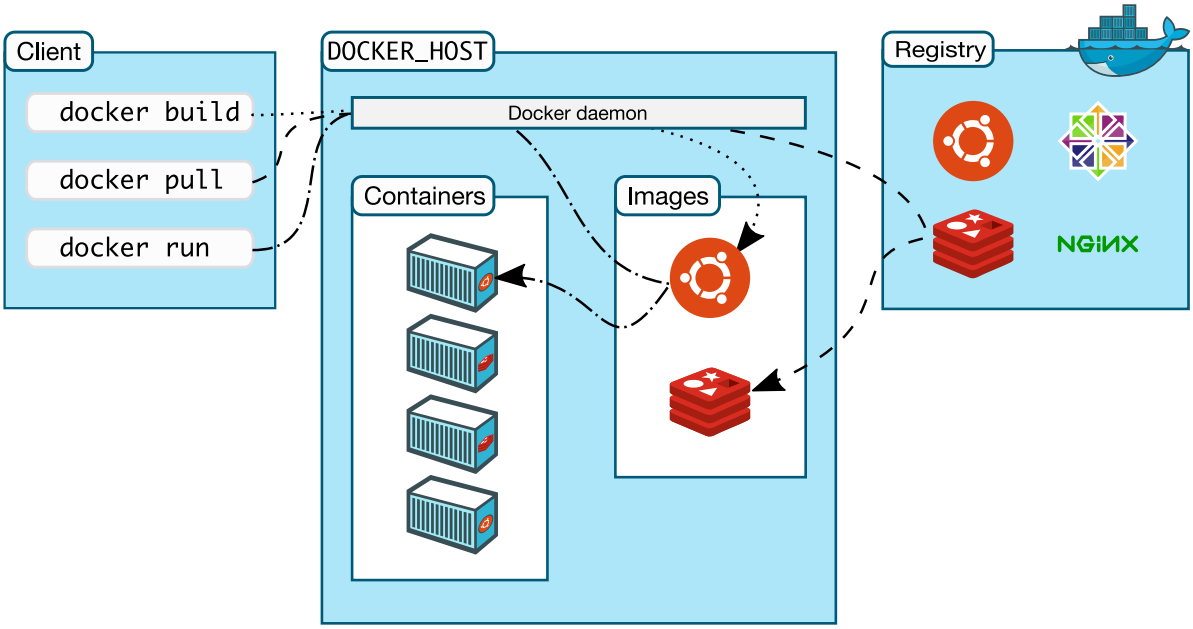
docker安装成功了

目标：搭建一个带视频回放功能selenium测试环境

4. Docker常用命令

docker 是C/S结构，

- docker client 发送指令
- docker server 接收指定，管理镜像、容器、系统资源



docker 的命令，通过`docker --help`查看

```
$ docker --help

Usage:  docker [OPTIONS] COMMAND

```

从docker 1.13开始，把繁多的命令，划入几个子命令中（把命令进行分组）

1. 镜像

镜像操作，在`images`字命令

```
docker iamge --help
```

```
$ docker image --help

Usage:  docker image COMMAND

Manage images

Commands:
  build       构建镜像
  history     查看镜像构建历史
  inspect     洞察：查看详情
  load        导入
  ls          列出
  prune       移除未使用的
  pull        从仓库拉取
  push        退到仓库
  rm          删除
  save        导出


```

```
docker image pull hello-world  # 从仓库拉取镜像
docker image ls                # 列出所有的镜像
docker image inspect api:v2    # 查看镜像的详细信息
docker image save hello-world -o hello.img # 导出镜像为文件
docker image rm hello-world:latest      # 删除镜像
docker image ls                # 检查是否删除成功
docker image load -i hello.img  # 导入镜像
```

没有hello 镜像了，怎么获取?

2. 网络

环境部署的三个时代：

- 同一个服务器，部署多个网站，如果一个被入侵，全体都挂掉
- 同一个服务器，安装多个虚拟机，虚拟机里部署网站
- 同一个服务器，多个容器（默认就是隔离）

镜像就绪之后，开始“网络管理”，因为测试环境和测试框架，必须打破隔离

网络管理的命令在 network 子命令

```
$ docker network --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     让容器加入一个网络
  create      创建新的网络
  disconnect  让容器退出一个网络
  inspect     洞察
  ls          列出所有
  prune       删除所有未使用的网络
  rm          删除指定的网络

```

```
docker network create my_net  # 创建新的网络
```

3. 文件存储（volume）

卷（volume）是Docker持久化工具，文件存储

```
$ docker volume --help

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes

```

创建 + 查看 +删除

4. 容器

最核心的内容终于来了

```
docker container --help

Usage:  docker container COMMAND

Manage containers

```

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
```

其中的`run`命令

```
$ docker container run --help

Usage:  docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
      --add-host list                  Add a custom host-to-IP mapping (host:ip)
  -a, --attach list                    Attach to STDIN, STDOUT or STDERR
      --blkio-weight uint16            Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)
      --blkio-weight-device list       Block IO weight (relative device weight) (default [])
      --cap-add list                   Add Linux capabilities
      --cap-drop list                  Drop Linux capabilities
      --cgroup-parent string           Optional parent cgroup for the container
      --cgroupns string                Cgroup namespace to use (host|private)
```

```
                                    'host':    Run the container in the
Docker host's cgroup namespace
                                    'private': Run the container in its
own private cgroup namespace
                                    '':        Use the cgroup namespace
as configured by the
                                               default-cgroupns-mode
option on the daemon (default)
      --cidfile string              Write the container ID to the file
      --cpu-period int              Limit CPU CFS (Completely Fair
Scheduler) period
      --cpu-quota int               Limit CPU CFS (Completely Fair
Scheduler) quota
      --cpu-rt-period int           Limit CPU real-time period in
microseconds
      --cpu-rt-runtime int          Limit CPU real-time runtime in
microseconds
  -c, --cpu-shares int              CPU shares (relative weight)
      --cpus decimal                Number of CPUs
      --cpuset-cpus string          CPUs in which to allow execution (0-
3, 0,1)
      --cpuset-mems string          MEMs in which to allow execution (0-
3, 0,1)
  -d, --detach                      Run container in background and
print container ID
      --detach-keys string          Override the key sequence for
detaching a container
      --device list                 Add a host device to the container
      --device-cgroup-rule list     Add a rule to the cgroup allowed
devices list
      --device-read-bps list        Limit read rate (bytes per second)
from a device (default [])
      --device-read-iops list       Limit read rate (IO per second) from
a device (default [])
      --device-write-bps list       Limit write rate (bytes per second)
to a device (default [])
      --device-write-iops list      Limit write rate (IO per second) to
a device (default [])
      --disable-content-trust       Skip image verification (default
true)
      --dns list                    Set custom DNS servers
      --dns-option list             Set DNS options
      --dns-search list             Set custom DNS search domains
      --domainname string           Container NIS domain name
      --entrypoint string           Overwrite the default ENTRYPOINT of
the image
  -e, --env list                    Set environment variables
      --env-file list               Read in a file of environment
variables
      --expose list                 Expose a port or a range of ports
      --gpus gpu-request            GPU devices to add to the container
('all' to pass all GPUs)
      --group-add list              Add additional groups to join
      --health-cmd string           Command to run to check health
      --health-interval duration    Time between running the check
(ms|s|m|h) (default 0s)
      --health-retries int          Consecutive failures needed to
report unhealthy
```

```
      --health-start-period duration   Start period for the container to
initialize before starting
                                       health-retries countdown (ms|s|m|h)
(default 0s)
      --health-timeout duration        Maximum time to allow one check to
run (ms|s|m|h) (default 0s)
      --help                           Print usage
  -h, --hostname string                Container host name
      --init                           Run an init inside the container
that forwards signals and reaps processes
  -i, --interactive                    Keep STDIN open even if not attached
      --ip string                      IPv4 address (e.g., 172.30.100.104)
      --ip6 string                     IPv6 address (e.g., 2001:db8::33)
      --ipc string                     IPC mode to use
      --isolation string               Container isolation technology
      --kernel-memory bytes            Kernel memory limit
  -l, --label list                     Set meta data on a container
      --label-file list                Read in a line delimited file of
labels
      --link list                      Add link to another container
      --link-local-ip list             Container IPv4/IPv6 link-local
addresses
      --log-driver string              Logging driver for the container
      --log-opt list                   Log driver options
      --mac-address string             Container MAC address (e.g.,
92:d0:c6:0a:29:33)
  -m, --memory bytes                   Memory limit
      --memory-reservation bytes       Memory soft limit
      --memory-swap bytes              Swap limit equal to memory plus
swap: '-1' to enable unlimited swap
      --memory-swappiness int          Tune container memory swappiness (0
to 100) (default -1)
      --mount mount                    Attach a filesystem mount to the
container
      --name string                    Assign a name to the container
      --network network                Connect a container to a network
      --network-alias list             Add network-scoped alias for the
container
      --no-healthcheck                 Disable any container-specified
HEALTHCHECK
      --oom-kill-disable               Disable OOM Killer
      --oom-score-adj int              Tune host's OOM preferences (-1000
to 1000)
      --pid string                     PID namespace to use
      --pids-limit int                 Tune container pids limit (set -1
for unlimited)
      --platform string                Set platform if server is multi-
platform capable
      --privileged                     Give extended privileges to this
container
  -p, --publish list                   Publish a container's port(s) to the
host
  -P, --publish-all                    Publish all exposed ports to random
ports
      --pull string                    Pull image before running
("always"|"missing"|"never") (default "missing")
      --read-only                      Mount the container's root
filesystem as read only
```

```
      --restart string                 Restart policy to apply when a container exits (default "no")
      --rm                             Automatically remove the container when it exits
      --runtime string                 Runtime to use for this container
      --security-opt list              Security Options
      --shm-size bytes                 Size of /dev/shm
      --sig-proxy                      Proxy received signals to the process (default true)
      --stop-signal string             Signal to stop a container (default "SIGTERM")
      --stop-timeout int               Timeout (in seconds) to stop a container
      --storage-opt list               Storage driver options for the container
      --sysctl map                     Sysctl options (default map[])
      --tmpfs list                     Mount a tmpfs directory
  -t, --tty                            Allocate a pseudo-TTY
      --ulimit ulimit                  Ulimit options (default [])
  -u, --user string                    Username or UID (format: <name|uid>[:<group|gid>])
      --userns string                  User namespace to use
      --uts string                     UTS namespace to use
  -v, --volume list                    Bind mount a volume
      --volume-driver string           Optional volume driver for the container
      --volumes-from list              Mount volumes from the specified container(s)
  -w, --workdir string                 Working directory inside the container
```

重点内容命令

- run （从镜像）创建，并启动容器
- stop 停止容器
- kill 强杀容器
- restart 重启容器
- logs 查看容器日志

举个例子

```
docker container run api:v2 # 启动容器
docker stop 46fd5f09b45b    # 让容器优雅退出
docker kill 46fd5f09b45b    # 强行终止容器
docker restart 46fd5f09b45b  # 重启容器
docker logs 46fd5f09b45b  # 查看容器控制台输出
```

`run` 命令的重点参数

```
docker run
   --name my_hello # 指定容器名称
   --rm   # 结束后自动删除
   --net my_net  #指定容器加入的网络
   --volume ${PWD}:/tmp # 把当前目录挂载到容器中
   -it      # 进入容器内容，挂载终端
   -d       # 让容器后台运行
api:v2
bash       # 容器要执行的命令
```

搭建支持视频回放的UI测试环境

0. 建立测试网络

```
docker network create test_net
```

1. 搭建selenium grid 容器

```
docker run --net test_net  --name my_slenium --rm -d  -p 4444:4444 -p 6900:5900 --shm-size="2g" selenium/standalone-chrome:4.1.0-20211123
```

2. 启动UI录制容器

```
docker run --net test_net  --name my_vidoe --rm -d -v /tmp/videos:/videos selenium/video:ffmpeg-4.3.1-20211123
```

下一步

构建自己的镜像，把测试框架，测试用例，装入其中

答疑

镜像到容器就是部署的过程嘛

是启动的过程，构建镜像的适合，已经完成部署的大部分工作

docker 和虚拟机有什么区别

1. docker 轻量，灵活，共享系统内核
2. 虚拟机完全模拟硬件，可用装各种系统 linux ->windows
3. docker只能跑同样的内核 linux -> linux

镜像和容器 有什么区别

类 VS 对象

程序文件（占用硬盘） VS 进程（占用的内存+cpu）

用的什么虚拟机

Vbox + ubuntu 20.04

UI自动化时，运行的是客户端的浏览器吗

不是的，一些都在容器中，客户端啥也没有

服务器性能有要求么

取决于你要运行的容器，一般我们容器会控制大小， 2核2G 一般够用