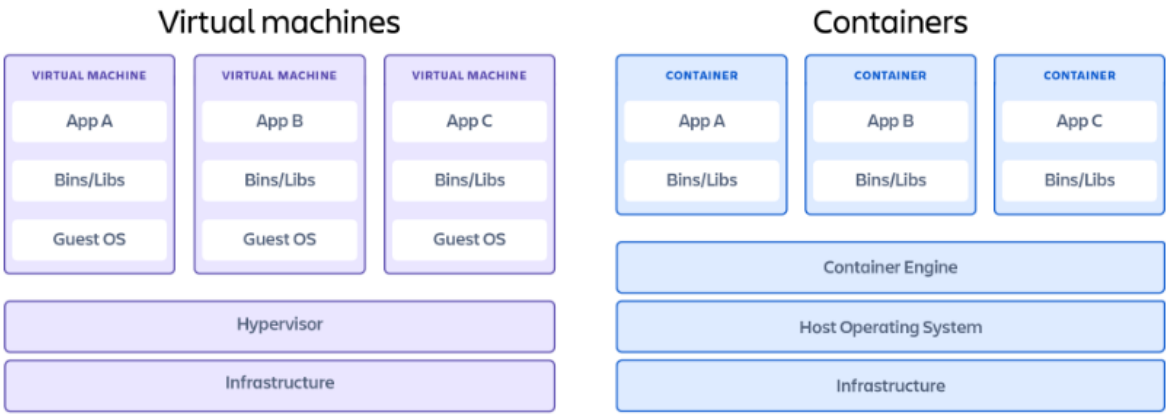
docker

1. 基本介绍

配环境是个很头疼的活，不仅要安装很多奇怪的库，有时候还要卸载一些本机上冲突的包。经常是卸这个，装那个，降版本，装一些已经不维护的库，最后可能还是跑不起来，但你本机的环境肯定已经一团乱麻了，原本自己可以跑的项目，突然就跑不了了，报一段比代码还长的错。能有一个与本机隔离的环境，在上面进行对应的配置，既不会与影响本机环境，又能避免因为冲突导致的各种bug，就能愉快的写代码了。

虚拟机(virtual machine)：虚拟机是大量的软件包，可以完全模拟 CPU、磁盘和网络设备等低级硬件设备。虚拟机还可能包含在模拟硬件上运行的补充软件堆栈。这些硬件和软件包结合在一起，可以生成功能齐全的计算系统快照。

容器(container)：容器是一种快速的打包技术(Package Software into Standardized Units for Development, Shipment and Deployment)，容器是轻量级软件包，包含执行包含的软件应用所需的所有依赖关系。这些依赖关系包括系统库、外部第三方代码包和其他操作系统级应用。容器中包含的依赖关系存在于高于操作系统的堆栈级别中。



虚拟机从底层的软件开始模拟，例如内核，操作系统，创造出一个与本机完全隔离的系统，基本可以说是只有硬件是共用的，其余的各种软件都是不一样的；容器少模拟一些，底层的东西不管，与本机共用，只有在上层的各种安装的库上与本机隔离，做到隔离的同时又轻量化。

1.1 Dockerfile

相当于这个环境的配置清单，有哪些需要的库，需要的配置，如何构建，相当于写的代码

1.2 image

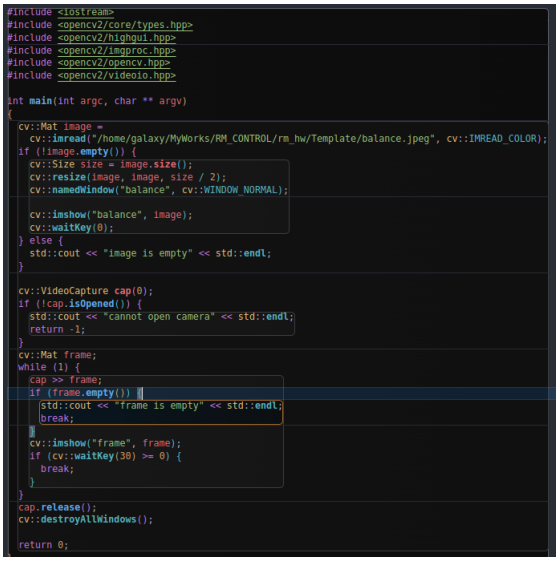
根据Dockerfile构建出一个环境，但这个环境还没有运行，就像是虚拟机的snapshot，相当于把代码编译成可执行文件

1.3 container

根据image，具体运行的环境，可以用一个image，运行多个相同配置的container，但这些container又相互隔离，相当于运行可执行文件

举个例子：

编写的代码就是Dockerfile，根据代码生成的可执行文件就是image，运行可执行文件就相当于运行container，可以同时运行多个相同的内容的代码，但内容上相互隔离。



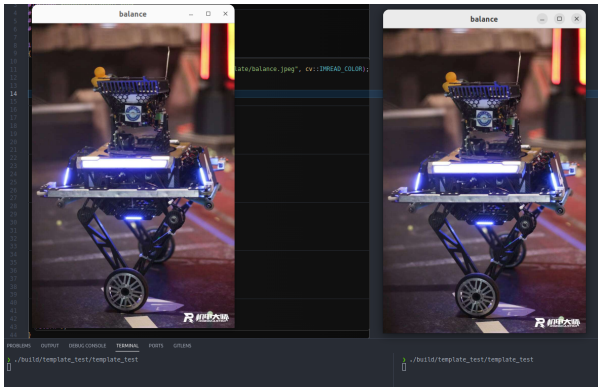
```cpp
#include <iostream>
#include <opencv2/core/types.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/videoio.hpp>

int main(int argc, char ** argv)
{
  cv::Mat image =
    cv::imread("/home/galaxy/MyWorks/RM_CONTROL/rm_hw/Template/balance.jpeg", cv::IMREAD_COLOR);
  if (!image.empty()) {
    cv::Size size = image.size();
    cv::resize(image, image, size / 2);
    cv::namedWindow("balance", cv::WINDOW_NORMAL);

    cv::imshow("balance", image);
    cv::waitKey(0);
  } else {
    std::cout << "image is empty" << std::endl;
  }

  cv::VideoCapture cap(0);
  if (!cap.isOpened()) {
    std::cout << "cannot open camera" << std::endl;
    return -1;
  }
  cv::Mat frame;
  while (1) {
    cap >> frame;
    if (frame.empty()) {
      std::cout << "frame is empty" << std::endl;
      break;
    }
    cv::imshow("frame", frame);
    if (cv::waitKey(30) >= 0) {
      break;
    }
  }
  cap.release();
  cv::destroyAllWindows();

  return 0;
```



2. 基本使用

2.1 安装

官网, 或者fishros yyds

2.2 基本操作

- docker image pull nginx 拉取一个叫nginx的docker image镜像
- docker container stop web 停止一个叫web的docker container容器

2.2.1 container基本操作

命令	docker container run	docker container ls < -a >	docker container stop/start	docker container rm
含义	容器的创建	容器的列出(up)	容器的停止/启动	容器的删除

> docker container --help

2.2.2 image基本操作

命令	docker image pull <:tag>	docker build <:tag>	docker image ls	docker image rm
含义	拉取镜像	构建镜像	镜像的列出	镜像的删除

> docker image --help

2.2.3 Dockerfile

- Dockerfile是用于构建docker镜像的文件
- Dockerfile里包含了构建镜像所需的“指令”

举个例子

```
FROM ubuntu:22.04  # 基于ubuntu22.04的镜像
RUN apt-get update && \ # 构建容器环境的命令
    DEBIAN_FRONTEND=noninteractive apt-get install --no-install-recommends -y
python3.10 python3-pip python3.10-dev
WORKDIR aaa #指定当前的工作目录
COPY hello.py ./ # 添加文件到容器的/aaa目录下
CMD ["python3", "/hello.py"] # 容器启动时默认执行的命令
```

2.3 docker cli

举个例子：

docker run -t -i -d --privileged -v /dev:/dev -p 8765:8765 --name="vision" vision

> 运行一个名为vision的container，
> -t -i -d分别是分配一个伪终端，交互式运行，后台运行，
> --privileged是给予container权限，-v是挂载本机的/dev到container的/dev，-p是端口映射，将本机的8765端口映射到container的8765端口，
> --name是给container命名，vision是container的名字，最后的vision是image的名字

实际操作：

- 根据Dockerfile构建image

> docker build -t docker_aaa:0.1 .

- 根据image运行container

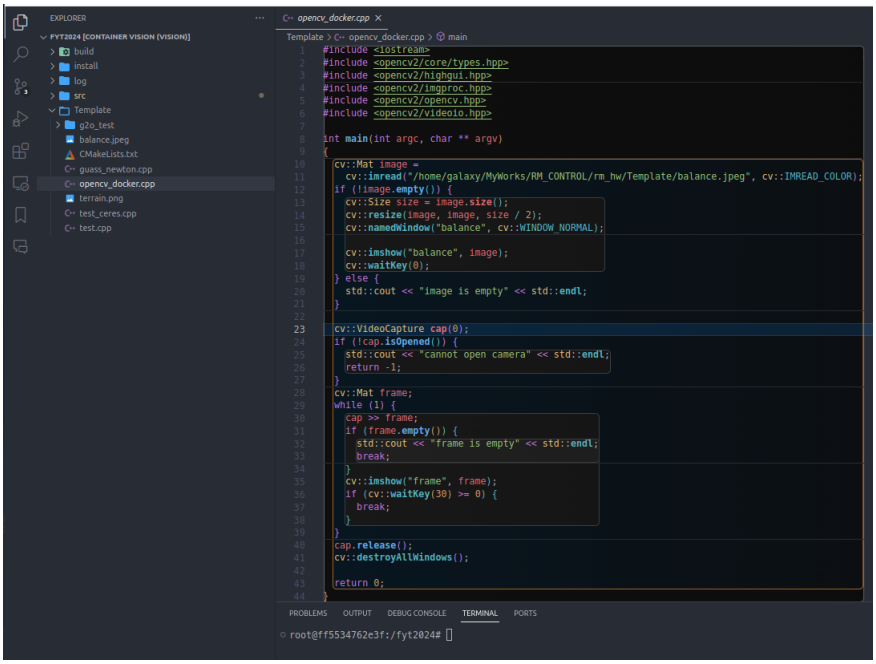
> docker run -itd --name aaa docker_aaa:0.1

- 进入container

> docker exec -it aaa /bin/bash

2.4 vscode + docker

安装 remote-development，ctrl+shift+p，输入 remote，选择show remote menu，attach running container，and then start your coding in container



3.作业

写一个用来构建ros-humble环境的Dockerfile文件，并用该文件构建一个image，最后用该image构建一个container，最后在container中运行ros-humble完成如下功能

- 写一个发布者，同过调用电脑的摄像头，捕获图像发送到话题/image_raw上
- 写一个订阅者，订阅/image_raw话题，打印图像的大小在图片上
- container外用foxglove查看图像
  提交文件程序以及运行结果图片

Reference

- https://github.com/xiaopeng163/docker.tips.git
- https://www.runoob.com/docker/docker-container-usage.html