DOCKER

by : DISHANK GOEL (18110052)
SHIVAM SAHNI (18110159)

Your code doesn't work

But it works on my machine!

Installed all the dependencies?

What about Configurations
Scalability
Security

Just use DOCKER!

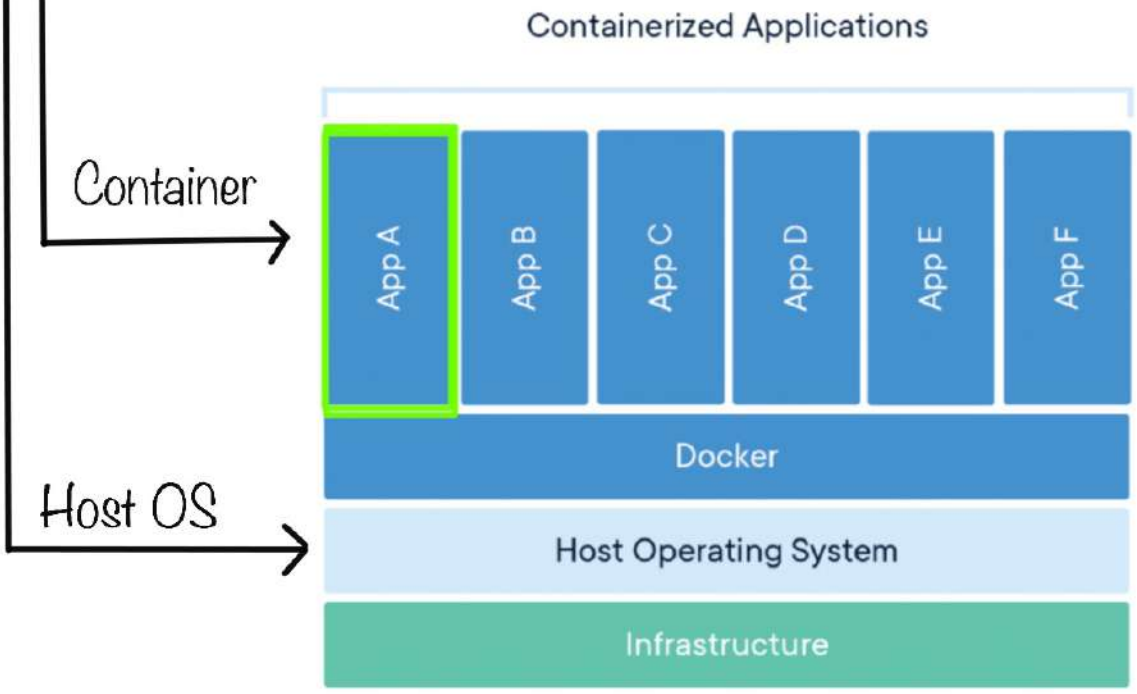
Docker provides an isolated environment to separate your code from the infrastructure

```
~ ls                                                                    INT 4s 07:11:55 PM
bin   cdrom  etc   initrd.img      lib    lost+found  mnt  proc  run   snap  swapfile  tmp  var      vmlinuz.old
boot  dev    home  initrd.img.old  lib64  media       opt  root  sbin  srv   sys       usr  vmlinuz

~ uname -a                                                                          07:11:57 PM
Linux dishank-Lenovo-ideapad-320-15IKB 5.3.0-46-generic #38~18.04.1-Ubuntu SMP Tue Mar 31 04:17:56 UTC 2020 x86_64 x86_64 x86_64 GNU/Li
nux

~ whoami                                                                            07:12:00 PM
dishank

~ docker exec -it trusting_rosalind /bin/bash                                       07:12:13 PM
root@97b57db7c113:/# ls
bin  boot  dev  etc  hello.py  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@97b57db7c113:/# uname -a
Linux 97b57db7c113 5.3.0-46-generic #38~18.04.1-Ubuntu SMP Tue Mar 31 04:17:56 UTC 2020 x86_64 GNU/Linux
root@97b57db7c113:/# whoami
root
root@97b57db7c113:/#
```



Credits: https://www.docker.com/resources/what-container

This seems so cool but how to run a container?

Before running, we need to build a Docker image

```
~/sem5/os/zines/docker docker build -t oszine -f Dockerfile .
Sending build context to Docker daemon  3.584kB
Step 1/3 : FROM python:3
 ---> 6feb119dd186
Step 2/3 : COPY sample_app/ /
 ---> Using cache
 ---> 4c851ed8a29c
Step 3/3 : ENTRYPOINT "/bin/sh"
 ---> Using cache
 ---> a89793a141d7
Successfully built a89793a141d7
Successfully tagged oszine:latest
```

Steps for building a Docker file

- Step1: Pulling an existing image
  (Here python: 3 is an Ubuntu image with python installed)
  The image is fresh with no dependency issues

- Step 2: Copy your code into the image
  Dump all the code you want to use here!

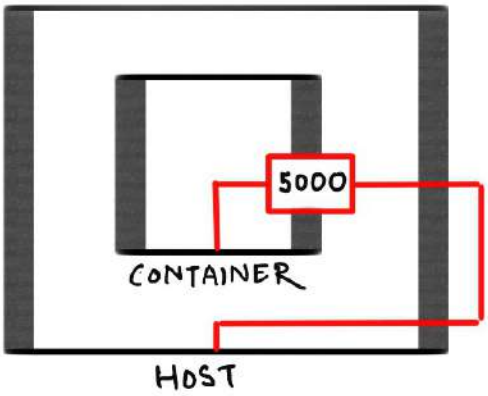
- Step 3: ENTRYPOINT
  Commands the container will run on startup

Great, now that our Docker image is built, let's start a container!

```
~/sem5/os/zines/docker docker run -t -p 5000:5000 -d oszine
84ea7ebebc7ab9eb7f595c8da86b866e89f4388455a141f6d158b57c9f535ede
```

Flags

- `-t` is used to get a terminal
- `-p` allows port forwarding



This allows the host to see the container's contents on port 5000

- `-d` is used to run the container in the background
- Oszine is the name of the image from which the container is run

Our container is now running!

```
~/se/os/zines/docker docker container ls                    INT ✖ 05:24:15 PM
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
84ea7ebebc7a        oszine              "/bin/sh -c \"/bin/sh\""   27 minutes ago      Up 27 minutes       0.0.0
.0:5000->5000/tcp   relaxed_euler
```

Container name

But how do we access the container?

```
~/sem5/os/zines/docker docker exec -it relaxed_euler /bin/bash
root@84ea7ebebc7a:/# ls
bin   dev  hello.py  lib    media  opt   root  sbin  sys  usr
boot  etc  home      lib64  mnt    proc  run   srv   tmp  var
root@84ea7ebebc7a:/#
```

Docker exec

- It executes your given commands inside the container
- Here we execute the /bin/bash command to get an interactive shell