Persistent

Working with Docker Containers

What are Containers?

- Virtualization systems like VMware and Xen, allow to run the complete Linux Kernel and operating system on top of virtualization layer known as Hypervisor.
- This approach provides strong isolation as each hosted Kernel sits in separate memory space.
- All containers share a single Kernel, and isolation is implemented within a single Kernel.
- Container is a self contained execution environment.
- The major advantage of this approach is, you don’t need a whole operating system for each isolated function.

History of Containers

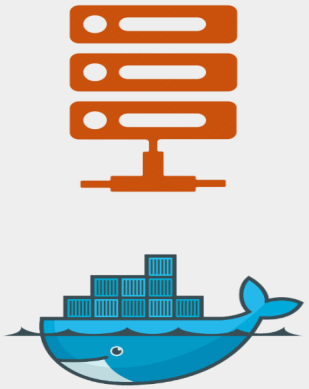
Batch Processing systems	Free BSD, jail command	Solaris Zones, HP-UX containers
• Run a program for a while and then switch to run another program. • Isolation – make sure your program didn't step on another.	• Allows shared hosting providers create separation between the processes.	• Solaris zones is the first major commercial implementation of containers. • HP released HP-UX containers. • 2013, inclusion of namespaces in Linux Kernel.

Creating a container

- Docker “run” command is a combination of two steps into one.
- The first thing is to creates the container from the image, this is accomplished using docker create command.
- The second step docker run does is, it executes the container, this can also be done using docker start command.
  - sudo docker create --name="awesome-service" ubuntu:latest
  - sudo docker ps –a

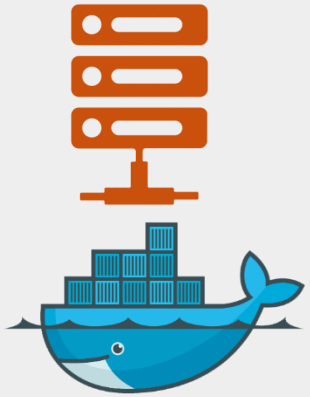
Container Hostnames

- Let’s launch a default container
  - sudo docker run --rm -ti ubuntu:latest /bin/bash
- The output will be root@e0df4a3e4453:/#
- When you see a with a prompt that looks something like root@hostname, it means that you are running a command within the container instead of on the Docker host.
- -rm tells Docker to delete the container when it exists.
- -t tells the docker to allocate pseudo – TTY
- -i indicates this is interactive session, /bin/sh is command to run in container



Container Hostnames

- Let’s check container’s hostname.
  - hostname -f
- The output will be e0df4a3e4453
- This hostname is the container ID which Docker has created.
- To set the hostname specifically, we can use the --hostname argument to pass in a more specific value.
  - docker run --rm -ti --hostname="mycontainer.example.com" ubuntu:latest /bin/bash
- Again for command,
  - hostname –f
    - The output will be mycontainer.example.com



Storage Volumes

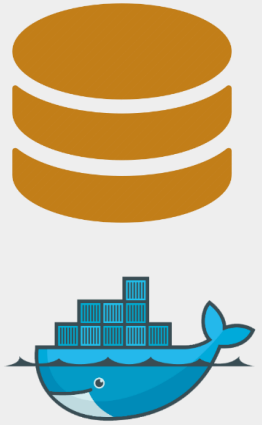
- A Volume is a designated directory in a container.
- It is designed to persist data, independent of the container life cycle
- Volumes are initialized when a container is created. If the container's base image contains data at the specified mount point, that existing data is copied into the new volume upon volume initialization. (Note that this does not apply when mounting a host directory.)
- Data volumes can be shared and reused among containers.
- Changes to a data volume are made directly.
- Changes to a data volume will not be included when you update an image.
- Data volumes persist even if the container itself is deleted.

Storage Volumes…

- Some of the important commands for Docker Volumes
- Adding a data volume –
  - You can add a data volume to a container using the -v flag with the docker create and docker run command.
  - $ docker run -d -P --name web -v /webapp training/webapp python app.py
- Mount a host directory as a data volume - In addition to creating a volume using the -v flag you can also mount a directory from your Docker engine's host into a container.
  - $ docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py
- More information about Data Volumes is available at below link
- https://docs.docker.com/engine/tutorials/dockervolumes/#/data-volumes

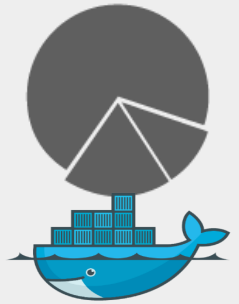
Storage Volumes…

- Mounting storage from the Docker host is not a generally advisable pattern because it ties your container to a particular Docker host for its persistent state.
- We are mounting /mnt/session_data to /data within the container.
  - sudo docker run --rm -ti -v /mnt/session_data:/data ubuntu:latest /bin/bash
  - mount | grep data
- The output will be /dev/xvda1 on /data type ext4 (rw,relatime,discard,data=ordered)
- We can see that the filesystem was mounted read-write on /data



Docker resource Quotas

- In traditional Virtual machines, you can tightly control how much memory, CPU and other resources are allocated to a Virtual machines.
- In case of Docker, Constraints are applied at the time of container creation.
- CPU Shares – Computing power of all CPU cores is considered full pool of shares which is 1024.
- By configuring a container's CPU shares, you can dictate how much time the container gets to use the CPU for Shipping industry solved this problem with standard container.
  - sudo docker run --rm -ti progrium/stress --cpu 2 --io 1 --vm 2 --vm-bytes 128M --timeout 120s
  - top -bn1 | head -n 15

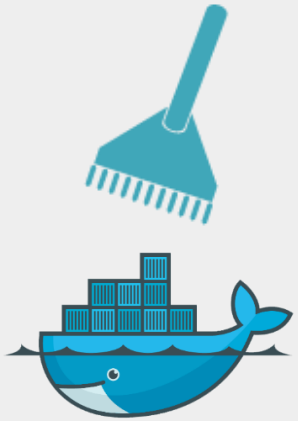


Docker resource Quotas - Memory

- We can control how much memory a container can access in a manner similar to constraining the CPU.
- While constraining the CPU only impacts applications priority for CPU time, memory limit is the hard limit.
- It’s possible to allocate more memory to a container than the system has actual RAM.
- In this case, the container will resort to using swap in the event that actual memory is not available.
  - sudo docker run --rm -ti -m 200m --memory-swap=300m progrium/stress \ --cpu 2 --io 1 --vm 2 --vm-bytes 128M --timeout 120s
- We are telling the kernel that this container can have access to 512 MB of memory and 256 MB of additional swap space.

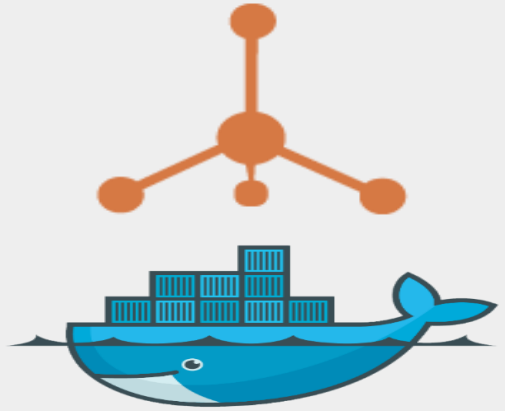
Stopping and cleaning a container.

- Let’s stop a running container.
  - sudo docker stop container id
  - sudo docker ps –a
- Status will be Exited (0) 4 seconds ago.
- You can also pause and unpause a container.
- You can clean a container using the command
  - sudo docker rm container id



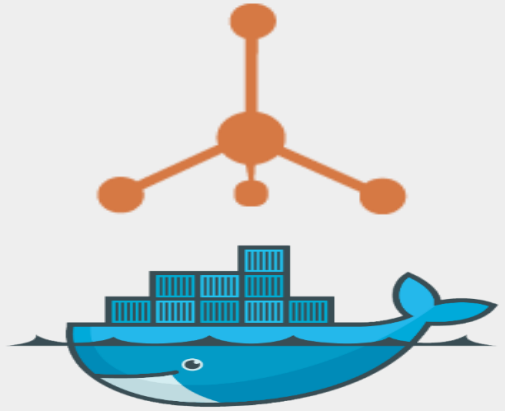
Docker Container Networks.

- Containers have their own network and IP Address.
- When you install Docker, it creates three networks automatically – bridge, none and host.
- The bridge network represents the docker0 network present in all Docker installations.
- The none network adds a container to a container-specific network stack.
- The host network adds a container on the hosts network stack.
- By default Docker containers can make connections to the outside world, but the outside world cannot connect to containers.
- If you want containers to accept incoming connections, you will need to provide special options when invoking docker run.

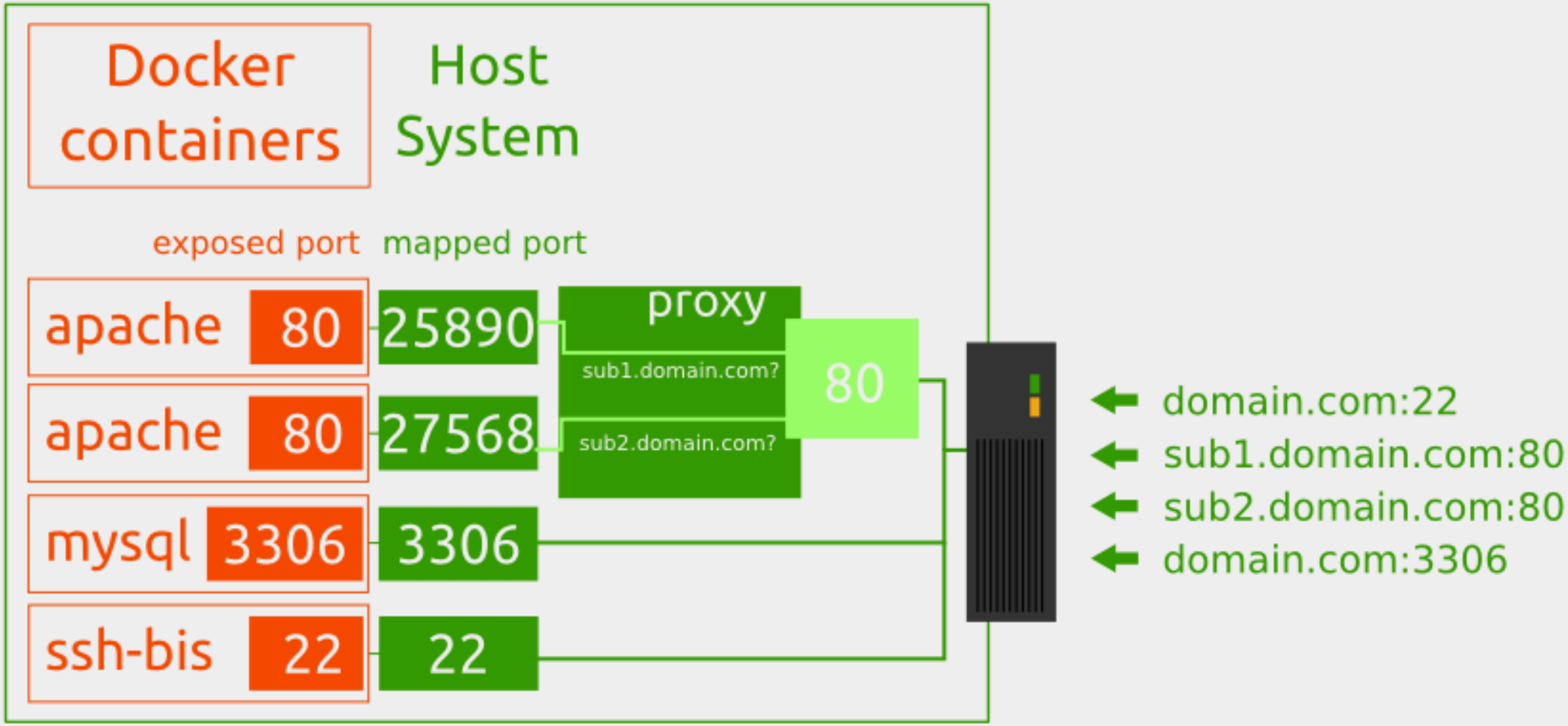


Docker Container Networks...

- Here are the options for exposing container ports
- You can supply -P or --publish-all=true|false to docker run.
- This is a blanket operation that identifies every port with an EXPOSE line in the image's Dockerfile
- Other option is to use
  - --expose <port> commandline flag and maps it to a host port somewhere within an ephemeral port range.



Docker exposing ports



Persistent

Exposing ports in Docker

- There are multiple ways to allow traffic between the containers.
- Option 1 is to use expose.
- You can expose a port via the --expose flag at runtime
- Other option is to include an EXPOSE instruction in the Dockerfile
- Option 2 is to publish ports by using the -p or -P flags in the Docker run string.

Example - Run Nginx in a Docker Container

- Let’s build a concrete application to understand the concepts of mapping the ports and how to link the container with local file system.
- We will try to create a container for Nginx, a popular Reverse Proxy server.
- In this section we'll download the Nginx Docker image and show you how to run the container so it's publicly accessible as a web server.
- By default containers are not accessible from the Internet, so we need to map the container's internal port to the Droplet's port.

Example - Run Nginx in a Docker Container…

- Let’s get the Nginx Docker image.
- Run the following command to get the Nginx Docker image.
  - sudo docker pull nginx
- This downloads all the necessary components for the container.
- Docker will cache these, so when we run the container we don't need to download the container image(s) each time.
- Let's start our Nginx Docker container with this command:
  - sudo docker run --name docker-nginx -p 80:80 nginx

Example - Run Nginx in a Docker Container…

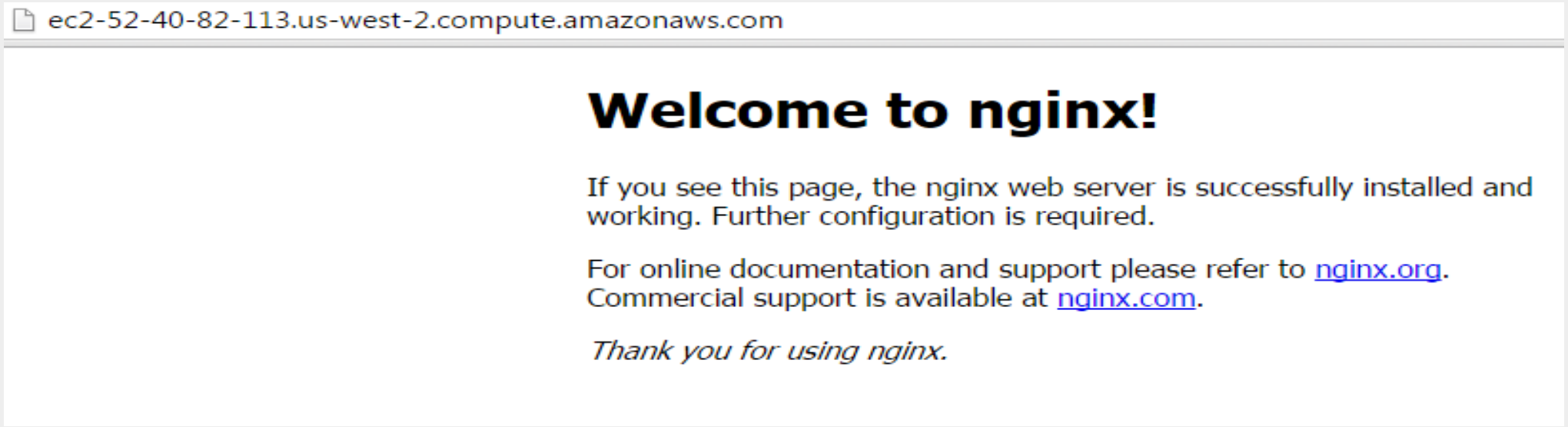
- Let’s analyze this command in more detail
  - sudo docker run --name docker-nginx -p 80:80 nginx
- run is the command to create a new container
- The --name flag is how we specify the name of the container, which is left blank for now.
- -p specifies the port we are exposing in the format of -p local-machine-port:internal-container-port.
- In this case we are mapping Port 80 in the container to Port 80 on the server.
- nginx is the name of the image on dockerhub (we downloaded this before with the pull command, but Docker will do this automatically if the image is missing)

Example – testing Nginx container

- Simple way is to use the CURL command
  - curl hostname
- You will see the HTML text of the Nginx page.
- Better way to test is through the browser.
- Make sure, you have the required ports open (in this case port 80) is open for Nginx to accept the requests.

Example – testing Nginx container…

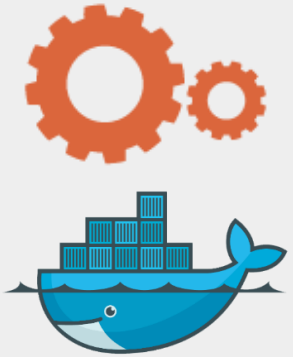
- Let’s hit the Nginx URL through the browser
  - http://ec2-52-40-82-113.us-west-2.compute.amazonaws.com/
- You will see the Nginx landing page.
- We can see the port 80 of the host is getting mapped to port 80 of Nginx container running on the host.



Persistent

Example - Running Nginx in detached mode

- Let's hit the break shortcut CTRL+C to get back to our shell session.
- If you try to load the page now, you'll get a "connection refused" page. This is because we shut down our container. We can verify this with this command:
  - sudo docker ps –a
- We can see that our Docker container has exited.
- Nginx isn't going to be very useful if we need to be attached to the container image for it to work.
- So let’s try to run Nginx in the detached mode.
- Let’s first remove the existing Nginx container.
  - sudo docker rm docker-nginx



Example - Running Nginx in detached mode…

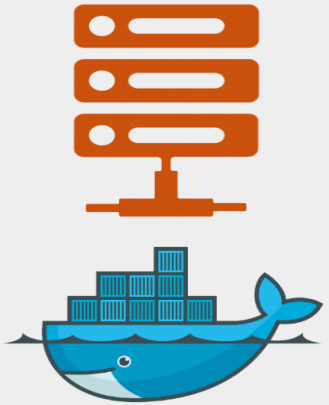
- Let’s run below command
  - sudo docker run --name docker-nginx -p 80:80 -d nginx
- We added the -d flag to run this container in the background.
  - sudo docker ps
- We can see that instead of Exited (0) X minutes ago we now have Up About a minute, and we can also see the port mapping.
- If we go to our server's IP address again in our browser, we will be able to see the "Welcome to nginx!" page again.
- Now we have a running instance of Nginx in a detached container!

Example - Running Nginx in detached mode…

- Let’s stop and remove the container.
  - sudo docker stop docker-nginx
  - sudo docker rm docker-nginx

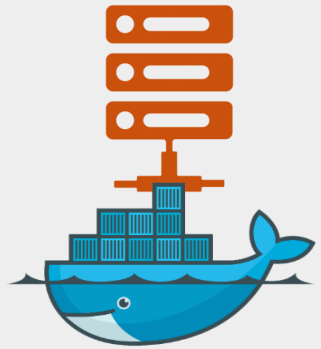
Example – Building a web page to serve Nginx

- Now we'll create a custom index page for our website.
- This setup allows us to have persistent website content that's hosted outside of the (transient) container.
- Let's create a new directory for our website content within our home directory, and move to it, by running the commands shown below.
  - mkdir -p ~/docker-nginx/html
  - cd ~/docker-nginx/html
  - vim index.html



Example – Building a web page to serve Nginx…

- Add the contents to the HTML file and save it.
- We've now got a simple index page to replace the default Nginx landing page.

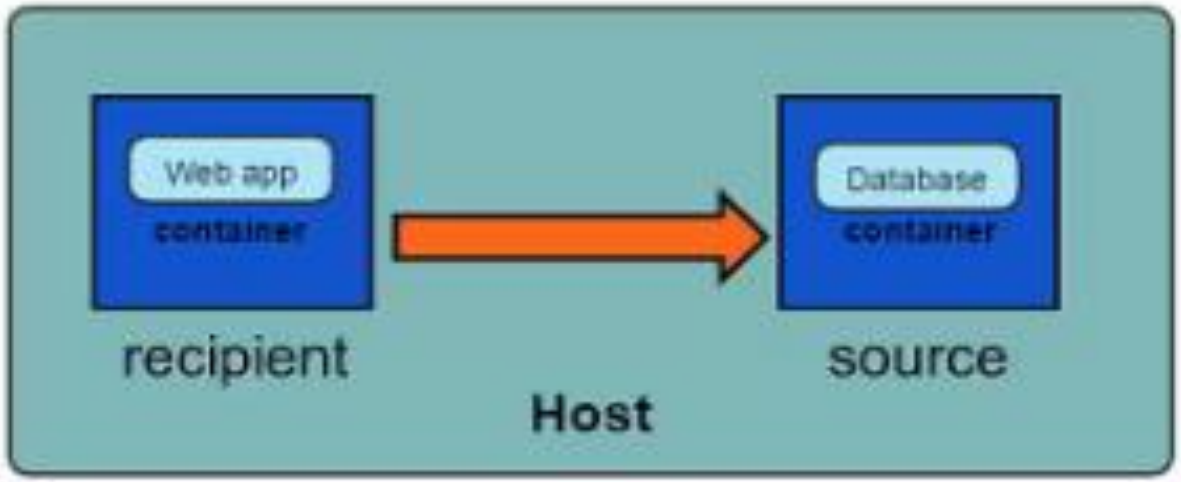


Persistent

Linking Containers

Linking *is a communication method between containers which allows them to securely transfer data from one to another*

- Source and recipient containers
- Recipient containers have access to data on source containers
- Links are established based on container names



Linking the containers…

Uses of Linking

- Containers can talk to each other without having to expose ports to the host
- Essential for micro service application architecture
- Example:
  - Container with Tomcat running
  - Container with MySQL running
  - Application on Tomcat needs to connect to MySQL

Linking the containers…

Creating a Link

1. Create the source container first
2. Create the recipient container and use the `--link` option

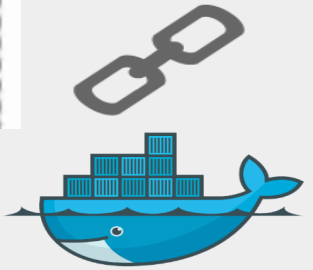
- **Best practice –** give your containers meaningful names

Create the source container using the postgres

```
docker run -d --name database postgres
```

Create the recipient container and link it

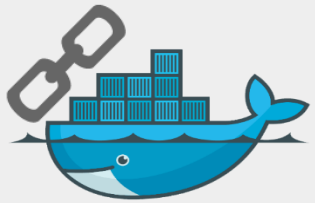
```
docker run -d -P --name website --link database:db nginx
```



Persistent

Example – Linking the container to local file system

- Let’s put it all together.
- We'll start our Nginx container so it's accessible on the Internet over Port 80, and we'll connect it to our website content on the server.
- Docker allows us to link directories from our virtual machine's local file system to our containers.
- The Nginx container is set up by default to look for an index page at /usr/share/nginx/html
- So in our new Docker container, we need to give it access to our files at that location.
- To do this, we use the -v flag to map a folder from our local machine (~/docker-nginx/html) to a relative path in the container (/usr/share/nginx/html).

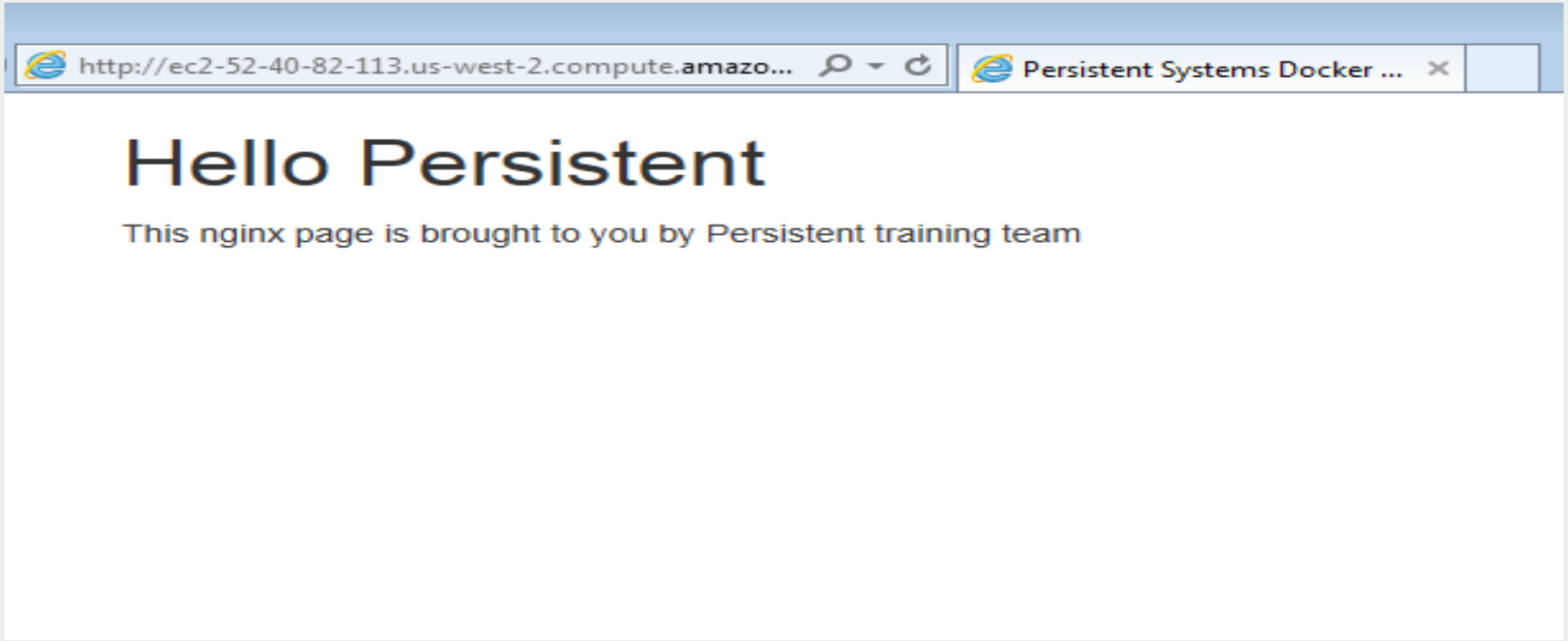


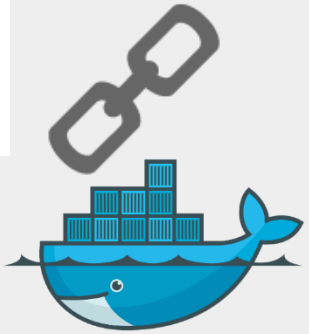
Example – Linking the container to local file system…

- Let’s run the command,
  - sudo docker run --name docker-nginx -p 80:80 -d -v ~/docker-nginx/html:/usr/share/nginx/html nginx
- -v specifies that we're linking a volume
- The part to the left of the : is the location of our file/directory on our virtual machine (~/docker-nginx/html)
- The part to the right of the : is the location that we are linking to in our container (/usr/share/nginx/html)
- After running that command, if you now point your browser to host's IP address, you should see the new HTML page.

Example – Linking the container to local file system…

- Here is the new HTML landing page for Nginx.





Persistent

Reference Material : Websites & Blogs

- https://www.docker.com/
- https://training.docker.com/self-paced-training
- https://www.youtube.com/watch?v=Q5POuMHxW-0
- https://www.digitalocean.com/community/tutorials/how-to-run-nginx-in-a-docker-container-on-ubuntu-14-04

Docker Interactive

Dattatray Kulkarni

dattatray_kulkarni@persistent.co.in

Asif Immanad

asif_immanad@persistent.co.in

Vice President

Shubhangi Kelkar

shubhangi_kelkar@persistent.co.in

Persistent

Thank you!

Persistent University