|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Bharat College of Arts, Commerce and Science, Badlapur(w)** |  | **2016-17** |  |
|  |  |  |  |  |
|  |  |  |  |  |

**F.Y.B.Sc. Computer Science [SEM-I]**

**USCS103: FOSS**

**Practical-10**

**CONTAINERIZATION**

**What is Containerization?**

Containerization is a lightweight alternative to full machine[virtualization](http://www.webopedia.com/TERM/V/virtualization.html) that involves encapsulating an application in a container with its own operating environment. This provides many of the benefits of loading an application onto a [virtual machine](http://www.webopedia.com/TERM/V/virtual_machine.html), as the application can be run on any suitable physical machine without any worries about dependencies.

Containerization has gained recent prominence with the open-source [Docker](http://www.webopedia.com/TERM/D/docker.html). Docker containers are designed to run on everything from physical computers to virtual machines, [bare-metal](http://www.webopedia.com/TERM/B/bare_metal.html)servers, [OpenStack](http://www.webopedia.com/TERM/O/openstack.html) [cloud](http://www.webopedia.com/TERM/C/cloud.html) clusters, public instances and more

**What is Docker?**

* Docker is an [open-source](https://en.wikipedia.org/wiki/Open-source) project that automates the deploymentof [Linux](https://en.wikipedia.org/wiki/Linux) [applications](https://en.wikipedia.org/wiki/Application_software)

Inside [software containers](https://en.wikipedia.org/wiki/Software_container).

* Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, and system libraries – anything you can install on a server.
* Docker provides an additional layer of abstraction and automation of [operating-system-level virtualization](https://en.wikipedia.org/wiki/Operating-system-level_virtualization) on [Linux](https://en.wikipedia.org/wiki/Linux).Docker uses the resource isolation features of the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel) such as [cgroups](https://en.wikipedia.org/wiki/Cgroups) and kernel [namespaces](https://en.wikipedia.org/wiki/Linux_namespaces), and a [union-capable file system](https://en.wikipedia.org/wiki/Union_mount) such as[aufs](https://en.wikipedia.org/wiki/Aufs) and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining [virtual machines](https://en.wikipedia.org/wiki/Virtual_machine).
* The Linux kernel's support for namespaces mostlyisolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting, including the CPU, memory, block I/O and network.
* Since version 0.9, Docker includes the libcontainer [library](https://en.wikipedia.org/wiki/Library_(computing)) as its own way to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via [libvirt](https://en.wikipedia.org/wiki/Libvirt), [LXC](https://en.wikipedia.org/wiki/LXC) (Linux Containers) and[systemd-nspawn](https://en.wikipedia.org/wiki/Systemd-nspawn).

In a nutshell, docker as a project offers you the complete set of higher-level tools to carry everything that forms an application across systems and machines - virtual or physical - and brings along loads more of great benefits with it.

Docker achieves its robust application (and therefore, process and resource) containment via Linux Containers (e.g. namespaces and other kernel features). Its further capabilities come from a project's own parts and components, which extract all the complexity of working with lower-level linux tools/APIs used for system and application management with regards to securely containing processes.

**Step 1. Install Docker for Windows**

1. Double-click InstallDocker.msi to run the installer.

If you haven’t already downloaded the installer (InstallDocker.msi), you can get it [here](https://download.docker.com/win/stable/InstallDocker.msi). It typically downloads to your Downloads folder, or you can run it from the recent downloads bar at the bottom of your web browser.

1. Follow the install wizard to accept the license, authorize the installer, and proceed with the install.

You will be asked to authorize Docker.app with your system password during the install process. Privileged access is needed to install networking components, links to the Docker apps, and manage the Hyper-V VMs.

1. Click Finish on the setup complete dialog to launch Docker.

**Step 2. Start Docker for Windows**

When the installation finishes, Docker starts automatically.

The whale in the status bar indicates that Docker is running, and accessible from a terminal.

If you just installed the app, you also get a popup success message with suggested next steps, and a link to this documentation.

When initialization is complete, select About Docker from the notification area icon to verify that you have the latest version.

Congratulations! You are up and running with Docker for Windows.

**Step 3. Check versions of Docker Engine, Compose, and Machine**

Start your favorite shell (cmd.exe, PowerShell, or other) to check your versions of docker and docker-compose, and verify the installation.

**Step 4. Explore the application and run examples**

The next few steps take you through some examples. These are just suggestions for ways to experiment with Docker on your system, check version information, and make sure docker commands are working properly.

1. Open a shell (cmd.exe, PowerShell, or other).
2. Run some Docker commands, such as dockerps, docker version, and docker info.

Note: The outputs above are examples. Your output for commands like docker version and docker info will vary depending on your product versions (e.g., as you install newer versions).

1. Run docker run hello-world to test pulling an image from Docker Hub and starting a container.
2. Try something more ambitious, and run an Ubuntu container in a Bash shell.

Type exit to stop the container and close the Bash shell.

1. For the pièce de résistance, start a Dockerized webserver with this command:

This will download the nginx container image and start it. Here is the output of running this command in a powershell

1. Point your web browser at http://localhost to display the start page.

(Since you specified the default HTTP port, it isn’t necessary to append: 80 at the end of the URL.)

1. Run dockerps while your webserver is running to see details on the container.
2. PS C:\Users\samstevens>dockerps
3. CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
4. NAMES
5. dfe13c68b3b8 nginx"nginx -g 'daemon off"3 days ago Up 45 seconds 0.0.0.0:80->80/tcp, 443/tc
6. p webserver
7. Stop or remove containers and images.

The nginx webserver will continue to run in the container on that port until you stop and/or remove the container. If you want to stop the webserver, type: docker stop webserver and start it again with docker start webserver.

To stop and remove the running container with a single command, type: dockerrm -f webserver. This will remove the container, but not the nginx image. You can list local images with docker images. You might want to keep some images around so that you don’t have to pull them again from Docker Hub. To remove an image you no longer need, use dockerrmi<imageID>|<imageName>. For example, dockerrminginx.

**Docker Settings**

When Docker is running, the Docker whale is displayed in the system tray. If it is hidden, click the up arrow in the tray to show it.

To get a popup menu with application options, right-click the whale:

The Settings dialogs provide options to allow Docker auto-start, automatically check for updates, share local drives with Docker containers, enable VPN compatibility, manage CPUs and memory Docker uses, restart Docker, or perform a factory reset.

**How to Use Docker: Creating Your First Docker Container**

**Creating your first Docker container**

* Docker creates virtual containers. Docker's container system is very efficient because it works with commits. This saves space, and allows you to see changes to the container. For example, if you install Apache in a container, you can create a commit with the name "Installed Apache" so you know exactly what happened.
* The first thing we'll do is pull from a repository. Say that you want to install Ubuntu in a container, you can pull Ubuntu from the repository:

docker pull ubuntu

* Be patient, as this can take a while. After everything has been downloaded, you can create a container with this OS:
* docker run -i -t ubuntu /bin/bash
* Or with Debian, for example:
* docker run -i -t debian /bin/bash
* If it can't find the OS (not pulled yet) it will automatically pull it from Docker Hub.

Effectively, you now have a container! You are running bash in the slimmed down container that is managed by Docker. Try running some common Linux commands to get a feel for the environment.

* When you type exit to exit the container and return to your main OS, all of your changes will be gone. To save changes to a container, we use commits.
* Commits

When you create a Docker container, its hostname is automatically generated. For example, when I create a new Ubuntu container, I might get the hostnamef7943e42aff0. This is the name that Docker has given to your container.

* Install what you want on it, and make sure everything works. Then exit your Docker container:
* exit

We now need to commit; otherwise, all of your changes will be lost.

docker commit -a "William E." -m "Installed Apache" f7943e42aff0 apachesnapshot

The -a switch can be used to properly determine who authored that commit (who made the changes in the container). -m is the commit message. Thef7943e42aff0 is the hostname of my container. In your case it will differ, as Docker generates them randomly. apachesnapshot is the name of your image.

You can view a list with all of the images on your local machine. The newest ones are at the top.

docker images

In order to start your Docker container with the changes, run:

docker run -t -i apachesnapshot /bin/bash

**Using Dockerfiles**

Dockerfiles can be used to make images with applications already installed. This makes it convenient to start a container without having to run a specific command. For example, if we want to create an image with the file ~/file.txt already created, we would use the following Dockerfile:

FROM ubuntu:14.04

MAINTAINER William E. <william@localhost>

RUN touch ~/file.txt

In order to create a Docker container with this Dockerfile, make a folder for your Dockerfile on your local machine (I used ~/files). Put the contents of your Dockerfile in a file called Dockerfile. You can now create an image with it by running:

docker build -t="test" .

This creates a Docker image from your Dockerfile script. You can now run your container. test is the same value as test in the docker build command.

docker run -t -i test /bin/bash

When the bash shell opens, you'll see that the ~/file.txt has already been created.

This is just a taste of the powerful environments that you can create using Docker. The Docker official manual goes into much further depth on these topics. At this point, you should be able to experiment running existing containers and begin to start imaging your own.

**How Docker works and how to make it work on windows**

Before jumping into the installation process, let’s see how Docker works in order to understand what we’ll need to do to make it work on Windows.  
We must first view Docker as both a server and a client.

The server lets us build, download, start and stop images or containers.  
The client is just a command line tool that will allow us to communicate with the API of the server.

Here is what it looks like on Linux:

Boot2docker

Since we aren’t using Linux, it is not possible ([yet](http://azure.microsoft.com/blog/2014/10/15/new-windows-server-containers-and-azure-support-for-docker/)) to use Docker natively. We are going to need some sort of lightweight VM that emulates a Docker Host.

This is what Boot2docker is for.

Boot2docker is a lightweight Linux distribution based on Tiny Core Linux made specifically to run Docker containers. It runs completely from RAM, weighs ~27MB and boots in ~5s

Source : [boot2docker.io](http://boot2docker.io/)

Basically, Boot2Docker will encapsulate our Docker server into a virtual machine and let us access it through the Windows Docker client.

**Installation**

* First, download and run the latest version of Boot2docker [here](https://github.com/boot2docker/windows-installer/releases/tag/v1.3.0). The Windows version of Boot2docker contains the Docker for Windows installer. It will install a set of tools (VirtualBox, Boot2docker ISO, MSYS-git UNIX tools) to help us run Docker.
* We now have a new shortcut on the Desktop called Boot2Docker Start. This shortcut is a shell script that will initiate the Boot2docker virtual machine and also the docker client. It will ask you for a ssh passphrase; you can either type one or just hit enter.  
  Boot2docker then starts initializing the VM and the Docker client. However, unlike Docker on OSX, we can’t use the Docker client directly from Windows but only from the Boot2docker VM.  
  We are now logged into the boot2docker VM and ready to use Docker.

**Running Docker containers**

Now that the install process is finally finished, we have the ability to launch containers, pull images, etc. as if we were using Docker on Linux.

First, type :

|  |  |
| --- | --- |
| 1 | docker |

to ensure that everything is working.

Boot2docker comes with a sample image named hello-world that lets new users quickly launch a container. Let’s give it a try :

|  |  |
| --- | --- |
| 1 | docker run hello-world |

We should see a “hello world” message printed on the screen.

**Port Redirection**

Redirecting ports with Boot2docker is really easy. The command is the same as on Linux, i.e. by using the -p flag like this :

|  |  |
| --- | --- |
| 1 | docker run -p PUBLIC\_PORT:PRIVATE\_CONTAINER\_PORT IMAGE CMD |

For instance, if we wanted to run the redis image on port 6379, we would run :

|  |  |
| --- | --- |
| 1 | docker run -p 6379:6379 redis |

The only thing that differs from running this command on Windows instead of Linux is that we will need the Boot2docker IP address in order to access that redis server.

In order to get it we can run the following command :

|  |  |
| --- | --- |
| 1 | boot2docker ip |

**Sharing a Windows folder**

This part will explain how to properly set up a shared folder between Windows and Boot2docker. However, as this feature is not officially included we will need to install additional tools.

First, we need to get cifs-utils on the Boot2docker VM

|  |  |
| --- | --- |
| 1. wget<http://distro.ibiblio.org/tinycorelinux/5.x/x86/tcz/cifs-utils.tcz> 2. tce-load -i cifs-utils.tcz |  |

Then, we create a folder

mkdir/mnt/sharefolder

This will be used as a mount point for our shared folder.

We will then need to create and share a new folder on Windows. To keep it simple, let’s just share the folder with ourselves.

Finally, we mount the shared folder on the Boot2docker VM

|  |  |
| --- | --- |
| 1 | sudo mount -t cifs //WINDOWS\_IP/shared /mnt/sharefolder -o username=WINDOWS\_USERNAME |

We will be asked to enter our Windows password and the folder will be mounted.

This will allow us to create volume containers (instructions [here](https://github.com/boot2docker/boot2docker#folder-sharing)) or share Dockerfiles between Windows and the VM for instance.