CSC 346 - Cloud Computing
01: What is the Cloud? An introduction to Docker

The Cloud

There is no cloud
it's just someone else's computer
The Cloud

- Using someone else’s computers is actually pretty useful

The Data Center

- Not too long ago, pretty much all big applications ran on physical servers in data centers that the Company or University controlled.

- Our Apps pretty much used to be installed on specific physical servers.
- If it was a big app, maybe it was distributed across several physical servers.
The Data Center
• Required a lot of guessing about the future
  • How much memory?
  • How many CPUs?
  • 1 server? 10 servers?
  • If I need more, how long will it take to order them, ship them, rack them, install the app...
  • If I bought too much, what then?

The Data Center
• Also problematic for smaller applications
  • Don’t need a whole server for some smaller apps or sites
  • Can put multiple applications on the same server
  • Difficulties with cross dependencies

The Data Center - Virtual Machines
• Virtual Machines allowed one massive server to host many smaller virtual machines.
  • This solved a lot of problems
  • Isolated applications
  • VMs can be sized to meet the application needs
  • Some overhead for growth
The Data Center - Virtual Machines

- Still had problems
  - Expensive initial purchase
  - Had to guess about future
  - You could expand later, but might not get identical equipment
  - If needs change drastically you may still be caught short on resources

- "Spiky Workloads" are a particular problem for a datacenter
  - If your application gets slammed at particular times (say... for priority registration)

- You have to have enough resources to meet that peak demand year round
  - That costs a lot of money
  - That excess capacity is "wasted" much of the time
  - VMs help some, as that excess capacity can be used by short lived projects
**The Cloud**

- Using someone else’s computers is actually pretty useful
- Instead of purchasing physical hardware, you rent it from someone else
- Costs move from large periodic capital expenses, to smaller monthly operational expenses. (Budget people love this)
  - $100,000 in year 1 - Lasts for 5 years (hopefully)
  - $2000 per month for 5 years
  - A bit more expensive over the long term possibly, but you don’t need $100,000 up front

**The Cloud**

- The biggest advantage of the Cloud is flexibility
  - Instead of paying for peak capacity year round, you can only pay for the 2 week spike
  - So maybe instead of $2000 a month its only $500 most months, and $2,000 that one peak month
  ($27,500 + $10,000 = $37,500)

**The Cloud**

Flexibility enables many different use cases

- Autoscaling - detect when your backend hosts are getting stressed and automatically deploy more backend resources. Get rid of them as the load subsides
- Experiments - deploy additional development environments in parallel to your production environments. Maybe each developer or feature gets dedicated resources
- Try new things faster - you don’t have to wait for new CPUs to be delivered to your datacenter. You can try new resources quickly and relatively cheaply.
The Cloud

Pay Per Hour vs Pay Per Request

- Cloud vendors offer many higher level services that shift the compute calculation.
- Virtual Servers are pay per hour. You pick a configuration, and it costs that much as long as you have the server “on”.
- Other services are pay per request. You configure the service, and then you pay a small fraction for each request the service handles. This can offer tremendous savings for smaller services, but could also benefit large ones.
- Become, hire, or befriend a cloud economist.

The Cloud

Datacenter Application Model

Shifting Skillsets

- Developers
- Systems Administrators
- Datacenter Technicians
- Business Managers
- Cloud Resources

The Cloud

Cloud Application Model

Shifting Skillsets

- Developers
- Business Managers
- Cloud Resources
The Cloud
Shifting Skillsets

What about these folks?

- Developers
- Systems Administrators
- Datacenter Technicians
- Business Managers
- Datacenter Resources

Provide Development Direction
Deliver Code
Deploy Code
Monitor Apps
Maintain Operating Systems
Install and remove hardware
Provide Purchase Direction

• These jobs don't go away.
• Companies still have lots of datacenters
• Cloud Providers have lots of datacenters!
• Migrate to other Jobs

Application Development
It's All About Speed of Deployment

- Research shows one of the best indicators of high performing development teams is how often they deploy new code to production, and how fast they can do this*
- Requires automation at all levels
- Cloud providers are easier to automate
  - API First mentality

*Accelerate: The Science of Lean Software and DevOps, Building and Scaling High Performing Technology Organizations
Forsgren, Nicole; Humble, Jez; Kim, Gene; 2018
Full text available at: O'Reilly Safari Learning Platform Academic
https://arizona-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01UA_ALMA21382514420003843&context=L&vid=01UA
Datacenters
- It was hard. Each new host had to be manually configured, at least initially.
- After initial setup, automation tools like Chef, Puppet, and Ansible could be used to setup a standard application environment, install dependencies, and deploy the application.
- This process was still comparatively slow, taking minutes to hours to complete.
- Operating system maintenance and patching could also be done through these orchestration tools.

Virtual Servers
- Once the VM infrastructure was configured, a “master image” of an application could be built.
- These images could then be deployed multiple times across VM infrastructure to build out the desired capacity.
- Images needed to be kept up to date with security patches still.
- Deploying code meant pushing changes into an existing VM, or re-building the entire VM image.
- Long-lived VMs still need to be managed with orchestration like Puppet, Chef, Ansible.

Cloud Computing with “Traditional” VMs
- Not really much different from VM infrastructure in your own datacenter.
- You’re still responsible for:
  - Building images
  - Operating system updates and patches
  - Application code updates
- It’s still just someone else’s computer
- Faster: No “spare capacity” to maintain yourself.
Containers

Yeah, pretty much Docker

- Containers isolate all dependencies required to run an application process
- Feels like a VM, but the underlying technology is different
- Does not contain a full OS / Kernel
- All containers on a Host share the same underlying Kernel
- Processes are isolated

Containers

- Container images are much smaller than full VM images.
- Host container environment can be run on commodity hardware. Does not require specialized VM infrastructure.
- The same container can run on Linux, Windows, macOS.
- Can run in Google Cloud, AWS, Azure
- Can run on your laptop

Containers & Docker

- Solves the age old problem of:
  "Developer: well, it runs on my laptop.
  Operations: great, give me your laptop, I'll put that into production."

- With Docker, you pretty much can do just that.
Containers & Docker
Key Concepts

- Containers isolate processes
- Containers encapsulate dependencies
- Running containers are ephemeral
- Images are immutable
- Images are composable

Containers & Docker

- Container isolate processes
  - A container is meant to run one process
  - You don't run Apache, Django, and MySQL all in one container
  - Instead have three separate containers
  - Allows each piece to be updated separately

Containers & Docker

- Container images are composable
  - You can start from a “base” image, and build your changes on top of this
  - Allows other teams/companies to be responsible for base configuration
  - You just have to worry about your specific dependencies
  - No limit to how many layers you want to go
Containers & Docker

- Container encapsulate dependencies
  - All the required libraries and code files for your process can be built into the image
  - A Dockerfile is used to define an image
  - Using the Dockerfile you can then build the image

FROM python:3.10
RUN pip install locust beautifulsoup4
RUN mkdir /tests
WORKDIR /tests
CMD ["locust"]

Containers & Docker

- Container images are immutable
  - When you run an image, you create a running container
  - Each time you run a container based off an image it's exactly the same.
  - Analogous to instantiating a class (imperfect)
Containers & Docker

- Running containers are ephemeral
  - You don’t “shut down” a container (at least in production)
  - When a container terminates, all changes to the container filesystem are lost
  - Any data the needs to be persisted must happen outside the container

- Docker gives us a way to have a standard development and evaluation environment across widely varying hosts
- You can turn in Dockerfiles and code files for us to run and test, without having to maintain a full Virtual Machine
- Remember that containers do not save their filesystem! Don’t lose your work!

Why Use Docker for CSC 346?

- You all have different laptops
- Docker allows us to have a standard development and evaluation environment across widely varying hosts
- You can turn in Dockerfiles and code files for us to run and test, without having to maintain a full Virtual Machine
- Remember that containers do not save their filesystem! Don’t lose your work!

UNIX Environments

Specifically, ubuntu/debian linux

- Linux is the default for most cloud hosts
- Linux is cheaper than MS Windows for servers
- Many platforms default to ubuntu, so why fight it
- For example: official python images

```
FROM scratch
ADD rootfs.tar.xz /
CMD ["bash"]
```

```
FROM debian:bullseye
```

```
FROM buildpack-deps:bullseye
```

```
FROM scratch
ADD rootfs.tar.xz /
CMD ["bash"]
```

```
FROM python:3.10
RUN pip install ...
```

```
FROM buildpack-deps:bullseye
ENV PATH /usr/local/bin:$PATH
```
What is UNIX?
- Bell Labs in the early 1970s
- Spawned many Open Source derivatives
  - BSD → Darwin → macOS
  - Linux → Debian → Ubuntu
  - Linux → Android
- Nearly unchallenged in the server / cloud space
  - Great process model
  - Developer friendly
  - Great command line interface

Linux Basics
Files and Directories
- Linux organizes a filesystem based mainly on files and directories
- Directories = Folders - We will not be pedantic about this 😊
- A filesystem is organized into a directory tree
  - Directories = branches   Files = leaves
- A filesystem has a single root directory
- Linux uses the “forward slash”, or just “slash” as the directory delimiter
  /Users/mark/Documents/csc246/01-cloud-docker.key

Linux Basics
Users and Groups
- Files are owned by users
- A ‘root’ user has access to everything
- Users can belong to groups
- File and Directories can have permissions that grant various access to users and groups
- Docker containers run everything inside them as a local root user, this is different from the host’s root user.
Linux Basics

Connecting

- A remote host is usually accessed through a Secure Shell - ssh

$ ssh username@hostname

- A local docker container can be accessed either through the initial run command, or by an exec command.

$ docker run -it --name python python:3.10 bash

Where Am I?

- When you first connect to a linux host, your CLI session will usually start in the user's home directory
- Docker containers usually start at the WORKDIR defined in that Image's Dockerfile
- If WORKDIR is not defined, you'll start at the filesystem root: /
- Use pwd to see your filesystem location (Present Working Directory)
What Stuff Is Here?

- To see the contents of the directory you're in, use the `ls` command (list).

Linux Basics
CLI Arguments

- Most CLI commands support arguments and options. Tells the command to do different things.
- The `ls` command accepts the `-l` option to list files in the long format.

- By default the `-l` long format shows file sizes in bytes.
- Use the `-h` option to show sizes in human readable format.
- Multiple options can be combined with the same dash: `-lh`
Linux Basics
Moving Yourself Around

- To move to a different directory, use the cd command (Change Directory)
- If used without an argument, cd will take you to your home directory.

• To move into another directory contained in the current one, use cd dirname

• To move up a directory, use the special “..” directory

Linux Basics
Core CLI Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td>Prints your present working directory</td>
</tr>
<tr>
<td>ls</td>
<td>Lists the files in your current directory</td>
</tr>
<tr>
<td>ls -lh</td>
<td>Lists the files in your current directory in long form, with human-readable file sizes</td>
</tr>
<tr>
<td>cd [dir]</td>
<td>Change your current working directory to [dir]</td>
</tr>
<tr>
<td>mkdir [dir]</td>
<td>Create a new directory named [dir] inside your current working directory</td>
</tr>
<tr>
<td>mv [from] [to]</td>
<td>Move a file from one location to another if [to] is not already another directory, it renames the file in your current directory</td>
</tr>
<tr>
<td>cp [from] [to]</td>
<td>Copy a file from one location to another</td>
</tr>
<tr>
<td>rm [file]</td>
<td>Delete a file (remove it)</td>
</tr>
</tbody>
</table>
### Linux Basics

#### Core CLI Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cat [file]</code></td>
<td>Prints the full contents of <code>file</code> to the screen</td>
</tr>
<tr>
<td><code>grep [string] [file]</code></td>
<td>Search <code>file</code> for the specified <code>string</code></td>
</tr>
<tr>
<td><code>head -n[count] [file]</code></td>
<td>Print the first <code>n</code> lines of <code>file</code> to the screen</td>
</tr>
<tr>
<td><code>tail -n[count] [file]</code></td>
<td>Print the last <code>n</code> lines of <code>file</code> to the screen</td>
</tr>
<tr>
<td><code>tail -f [file]</code></td>
<td>Print the last few lines of a file to the screen, and continue to follow it as new lines are added</td>
</tr>
<tr>
<td><code>less [file]</code></td>
<td>Prints out the contents of the first page of a file to your screen, and gives you keyboard commands for navigating through the file. Read-only</td>
</tr>
</tbody>
</table>

#### Linux Basics

#### CLI Text Editors

- Popular editors: `vi`, `vim`, `emacs`, `nano`
- All keyboard and text based. No mouse.
- I mostly try and avoid CLI text editors. I like my GUI!
- We’ll see many strategies for avoiding the CLI editors
- When I need to, I mostly use `vim` or `vi` depending on what is available

#### Linux Basics

#### STDOUT, Redirection, and Pipes

- UNIX has a concept of Standard Out (STDOUT) and Standard Error (STDERR)
- By default STDOUT is directed to your terminal screen
- STDOUT can be redirected to other places through

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls -l &gt; output.txt</code></td>
<td>Sends the STDOUT of the <code>ls</code> command to a file named <code>output.txt</code>. If that file exists, it will be overwritten. If the file does not exist, it will be created</td>
</tr>
<tr>
<td><code>ls -l &gt;&gt; output.txt</code></td>
<td>Appends the STDOUT of the <code>ls</code> command to a file named <code>output.txt</code>. If that file exists, it will add new output to the end of the file. If the file does not exist, it will be created</td>
</tr>
<tr>
<td>`python3 ./prog.py</td>
<td>less`</td>
</tr>
</tbody>
</table>
Development Environments
Microsoft VS Code

- Not required, but it's really great
- Free
- GUI Text editor and terminal all in one
- Can open a local folder and use it as a project
- Customizable
- Plugins for just about everything
Development Environments
Microsoft VS Code

• For this class you will need access to Docker Desktop on a computer
• Free for individual and educational uses
• Installers for Windows, Mac, and Linux

https://www.docker.com/products/docker-desktop/
Once you have Docker Desktop installed and running, you should see a window like this.

### Docker

**docker run**

- Starting a new container from an image is done with the `docker run` command:

  ```bash
  $ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
  $ docker run -it python:3.10 bash
  ```

- The `-it` options are for interactive (i) and connect tty terminal (t).
- The image is `python` and the tag for the image is `3.10`, this specifies which version of the image to run.
- Once the container is running, we execute the `bash` command inside it. Since we connected our terminal to this, we should get a command prompt 'inside' the container.
Our First Container

- We can run this command in our terminal.
- Because we have never used the python:3.10 image before, it must be downloaded from hub.docker.com.

Once the image has downloaded, our bash command is executed inside.

You can see our terminal prompt has changed.

- We're root inside the container.

We can use our linux commands here.
- The pwd command shows we're currently at the filesystem root.
- The ls command lists all the files and directories at the root of the filesystem.
- The cd command will take us to root's home directory.
We can exit our container by typing the `exit` command.

This returns us to our host.

We can list all the running or stopped containers with the `docker ps -a` command.

Docker containers are not removed by default.

Remove an exited container with `docker rm [container id]`.

Can also remove containers by name with `docker rm [container name]`.

You can list images you currently have locally with the `docker images` command.
Our First Container

- While still smaller than full Virtual Machine images, docker images can still clutter up your local storage.
- Use `docker rmi [image id]` or `docker rmi [image:tag]` to remove them.

Docker

- Some additional run options
  - The `-name` option sets the friendly name of the container.
  - The `--rm` option automatically removes the container upon exit.

```
$ docker run -it --rm -name python python:3.10 bash
```
Our CLI commands are already getting longer and harder to remember.

Linux offers us a way to wrap up a set of commands into a script file that can be executed.

This works by default for macOS and Linux-based laptops.

Windows uses PowerShell by default and can do similar things.

```bash
$ docker run -it -rm -name python python:3.10 bash
```

With a folder opened in VS Code, click on the new file icon next to the folder name in the Explorer tab.

Type in the name of the new file. For example `run.sh`.

The new file will open in a new tab in the Editor pane.

Instead of having everything on one line, it is often easier to break a command across multiple lines.

Shell commands can be continued to a new line by having a backslash character as the final character on a line.
Automation

Bash Shell Script

- Before you can execute a shell script, you must flag it as executable
- The chmod command lets you change modes on a file
- The +x option adds the execute mode to the file
- Run the command with `./[filename]`