Barcelona GSE
Data Science Center

Introduction to Computing

September, 2018

Goals

This module will provide the tools you need to start being productive using Linux servers.

You will:

- Launch a cloud virtual machine
- Connect via SSH.
- Use the command line
- Write a shell script
- Install programs
- Use Docker

TCP

TCP is a protocol, a “transport layer” representation that abstracts the two-way communication between two computers. It ensures all the “packets” get delivered in the proper order.

HTTP, the protocol of the web, is built on top of TCP.

SSH, “secure shell”, is also built on top of TCP .

SSH can be used to login to another computer and control it via a “shell”, particularly via the a CLI shell.

Launching a Server

Let’s launch a virtual machine on AWS.

We will use SSH to login and control the machine via a CLI.

We will run a Jupyter notebook on this server as well.

The Jupyter notebook creates a web server, which we access over HTTP.

So we will create two TCP “channels” to communicate with our virtual machine both via SSH and via HTTP.

Key pairs

The first TCP channel we will create is to SSH into the server.

Logging in via SSH requires a password or RSA keypair.

For AWS instances, you must use a keypair!

A keypair consists of a private key and a public key.

You can create this keypair, or AWS can create this keypair.

ssh-keygen & ssh-agent

```
eval "$(ssh-agent -s)"
ssh-add [path-to-pem-file]
```

Launching a Server

Demo:

(create a key)

(configure: specs, volumes, security groups)

(connect via SSH)

Shell

There are many shells in the world.

Luckily, most use similar commands.

“Bash” is a very common shell that is available in most Linux distros, as well as in Mac OSX.

All shell CLI’s come with builtin commands. For example:

```
which
pwd
cd
```

Practise:

(Navigating and tab-complete (creating, moving, copying, viewing files)

```
touch
mv
cp
cat
head
tail
```

$HOME

Home contains per-user settings.

It’s alias’d to ~ and available in environment variable: $HOME

Environment Variables

These are just variables, but they live in the “process” (the shell, or whatever is started from the shell)

$HOME is an example of an environment variable.

You can create environment variables, which can be a way to pass information to your program.

```
echo $USER
```

Practise

If you want more practise with shell commands and basic programs:

OverTheWire

Programs

What if you want to create a program?

```
echo "date > foo" > write-date-to-foo.sh
chmod +x write-date-to-foo.sh
./write-date-to-foo.sh
```

You’ve created a program.

Executables

Shell scripting is one way to create programs.

The other way is to create an executable (a.k.a. binary)!

An executable is machine code: instructions that your computer can run.

Machine code is created by compiling source code.

Source code is for humans, machine code is for machines.

Compilers are translators, human > machine!

Shell as a program

Shells, such as bash, are executables that runs on your computer!

```
stdin >  bash  > stdout
       ---^v---
       computer
```

Compiling from source

Demo:

(Compile CPython from source)

https://github.com/bgsedatascience/module-computing/tree/master/compiling

$PATH

Your contains an environment variable called PATH:

```
echo $PATH
```

PATH contains a list of paths, separated by colons:

```
usr/local/bin:/usr/bin:/bin
```

When you enter a command into your shell, (for example "python") it will look for any executables that are in any of the folders listed in PATH.

Installing

Installing a program, therefore, is often nothing more than putting an executable in a folder that is listen in your PATH.

Instead of moving it to one of those folders, you could also create a symbolic link:

```
sudo ln -s /your/program/executable /usr/local/bin
```

Or, conversely, you could modify your PATH to add the folder that holds the executable:

```
export PATH="your/program/:$PATH"
```

Reproducible environments

Setting up a server requires lots of programs.

Every version of a program is different: software is alive!

How can you ensure your code will run on a new server? Or someone else's computer?

Cloud Images

Cloud providers usually provide a way to make an “image” of your server.

AWS calls these AMI (Amazon Machine Image).

This can be a great solution if you do all your work on AWS servers.

Docker

Docker is the name of a popular containerization software.

Containers are like little mini computers that run inside your computer.

If you've heard of virtual machines, it's like that, but without the CPU overhead.

Docker

Demo:

(Install Docker via apt)

https://docs.docker.com/install/linux/docker-ce/ubuntu/

Jupyter Docker Stacks

The folk behind Jupyter have created a set of Docker images that are extremely useful for data science:

https://github.com/jupyter/docker-stacks

We can run the “datascience” image with the following command:

```
sudo docker run -d --name notebook -v $PWD:/home/jovyan/wor
-p 8888:8888 jupyter/datascience-notebook
```

Docker options

-d Run the container “in the background,” so you can do other things in your terminal, or close it, without the container exiting.

—name This is just a name which you can refer to it by later.

-v Containers have their own file system. This command allows you to “mount” a folder from your host computer into a folder in your container, such that you can read and write to files on your computer from the container. It is a mapping “host/folder:docker/folder”

-p Just as computers have numbered ports, so do containers. This option creates a mapping from the host computers port to the containers port. For example, “707:8888” would map the port 707 on the host computer to the port 8888 in the container.

Getting logs

Because we ran the container “in the background” via the -d command, we have to look a bit to find the token that jupyter creates for us:

```
sudo docker logs notebook
```

This will display all the logs that the container named “notebook” has created.

Look at the website for the Jupyter Docker Stacks to see more about all the options they provide! Including setting your own password, and using Jupyter Lab.

Giving Docker to our user

To avoid having to use “sudo” every time we use docker, we can add the current user (ubuntu) to the docker group:

```
sudo usermod -aG docker $USER
```

USER is an environment variable that contains your username, and “docker” is the group we are adding the user to, a group which was created by the docker install process.