MY472 - Week 11
Cloud Computing + Docker

Thomas Robinson

Outline

- Cloud computing
- Containerization & Docker
- Guided coding
  1. Running an EC2 instance
  2. Running Docker containers locally

This week's seminar:

- Building a shiny app

Cloud computing basics

A definition

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.” (From *NIST Definition of Cloud Computing*)

Possible use cases in data science

- Continuous scraping or API requests
- Hosting and querying (very large) databases
- Training machine learning models, e.g. in NLP, deep and reinforcement learning
- “Embarrassingly parallel” tasks/simulations
- Hosting web applications (like Shiny Apps)
- …

Won’t be able to cover even 1/100000th of possible use-cases

- Highlight the basic *mechanics*
- Leave it to you to explore further avenues

Virtualisation

- In cloud computing, computers are virtualised
  - Similar idea to virtual machine on your computer but readily scalable
  - Data centers use hardware to host a number of virtual machines
    - Centers are geolocated
    - Choice can depend on latency and legal issues
- Configuration of *virtual machine* is called an **instance**
  - Number of (v)CPUs
  - Amount of RAM
  - Amount/type of storage capacity
  - Base operating system

Advantages of cloud computing

- Scalability (on demand)
  - Many services let you purchase capacity near instantly
  - You can choose the hardware configuration *that suits the task at hand*
  - Suitable for massive tasks that are simply infeasible on a personal computer
- Costs
  - Only pay for what you use
- Ease
  - You can get on with your day while your code runs!

Disadvantages of cloud computing

- Scalability
  - More “compute” does not necessarily mean faster execution
- Costs
  - Can be expensive e.g. for frequent computations
- Ease
  - Requires careful testing before committing to costly servers
- Security and legal compliance
  - Connection via the cloud can open vulnerabilities
  - GDPR requirement of encryption, data security, data physical location

Amazon AWS Elastic Cloud Compute (EC2)

One of most popular cloud computing services:

- Reasonably cheap (including free options)
- Good control of configuration
- Complementary products (like storage)

We will use **t2.micro** instances this week (free for first 12 months)

- “micro” because it has 1 vCPU and only 1GB RAM
- After 12 months, this server costs $0.0116 per hour to run (US-East-1)

AWS offers general-purpose and compute/memory-optimized instances

- Including GPUs (at a cost)

Amazon AWS EC2 online resources

AWS Free Tier

Instance types

On demand pricing by instance type

(Amazon Machine) Images

We want to initialise our instance with some basic configuration

- An operating system
- Any specific applications

Amazon “images” provide templates for configuring the system

- Can be stored and loaded quickly
- Can be application-specific, e.g. launching an RStudio server

A good starting point is the default Amazon Linux image:

- Open-source so no additional costs to run
- Loads of support online

Step 1: Launching the instance

- In the EC2 dashboard click “Launch instance”
- Choose Amazon Linux 2 AMI
- Choose **t2.micro** instance
- Create key pair for SSH access (or choose existing one)

Note:

- Default user name for Amazon Linux 2 AMI is “ec2-user”
- (See documentation for other AMIs)

Step 2: Connecting to the instance, setting swap memory

- Connect to EC2 instance via command line (more info)
  - Mac/linux: `chmod 400 yourkeyname.pem` and `ssh -i "path/to/key/yourkeyname.pem" ec2-user@public-instance-dns`
  - Windows: Use PuTTY - tutorial as pdf on course page
  - Linux console via the browser (simpler): Click on instance -> Connect -> EC2 Instance Connect -> Connect
- NOTE: for small instances with only 1GB of ram, create swap memory (reference) to install some larger R packages without maxing out memory

```
sudo /bin/dd if=/dev/zero of=/var/swap.1 bs=1M count=2048
sudo /sbin/mkswap /var/swap.1
sudo /sbin/swapon /var/swap.1
sudo sh -c 'echo "/var/swap.1 swap swap defaults 0 0 " >> /etc/fstab'
```

Step 3: Installing Linux libraries and R

- Before installing R packages, we need to install additional Linux libraries
  - `sudo yum install libcurl-devel openssl-devel libxml2-devel`
- Then install R
  - `sudo amazon-linux-extras install R4`
- Afterwards you can open R session by typing `R` at the command line
- More usually on the cloud, you run entire R scripts non-interactively
  - We'll demonstrate this in a few slides times!

Step 4: Copying files to and from the EC2 instance

- We can copy files from and to the EC2 via the command line (`scp` command)
- Or use a file transfer programme (guide here)
- Cyberduck is a good option
  - Choose “Open connection”
  - SFTP (secure file transfer protocol)
  - Enter the public DNS for server,
  - User name is “ec2-user” and use your .pem as SSH private key

Step 5: Running R, R scripts and installing packages

Useful commands:

- `ls` shows all folders and files in the current directory
- `cd path/to/some/folder` goes to folder
- `Rscript myscript.R` runs R script in current folder
- `Rscript myscript.R &` runs R script in the background while the shell is open

When you have many packages to install:

- Write an install R script
- Upload to server
- Run using `Rscript`

Persisting sessions

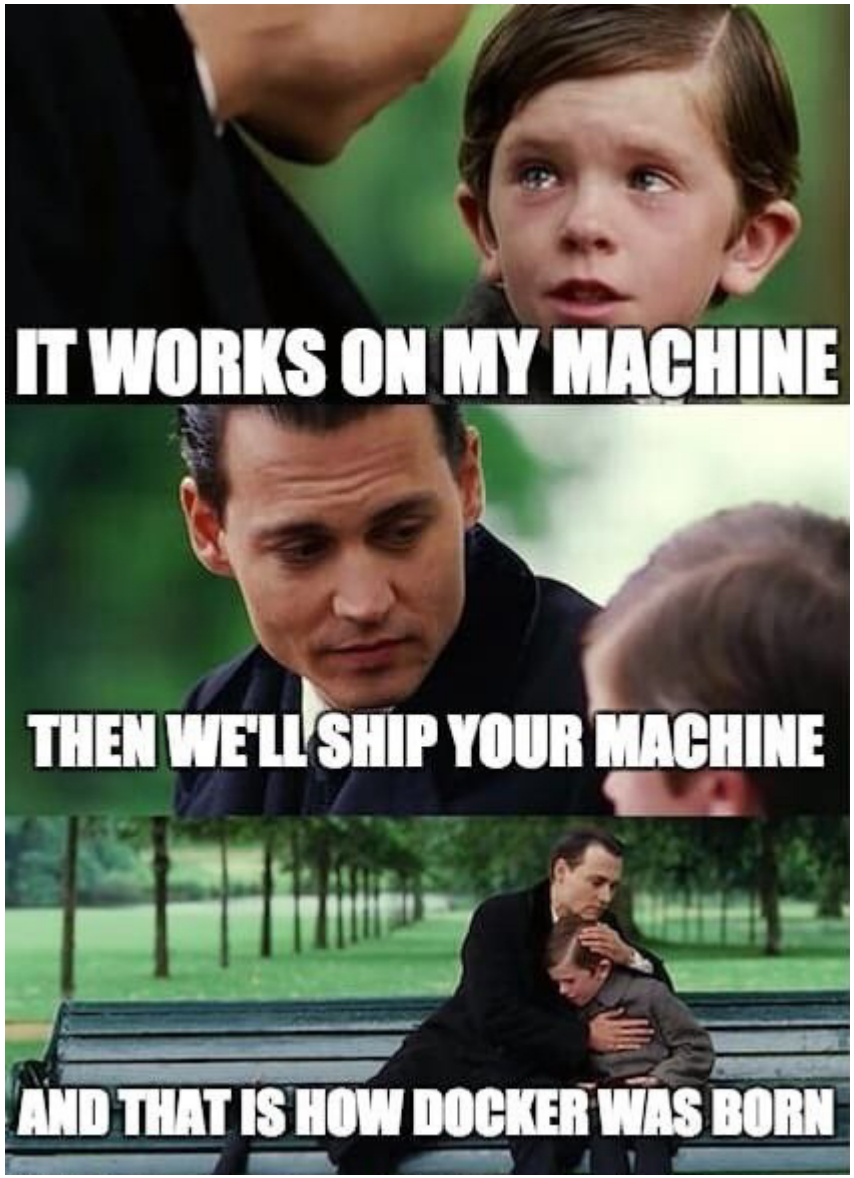
Common issue:

- Set an R script running
- Lose connection (close terminal/laptop/internet connection)
- Log back in and find the script has terminated

Why? Your session terminates when the SSH connection closes

- Use `screen -S screen_name` to start a persistent session
- Run your code in this screen
  - Exit the screen with Ctrl + a + d
  - Reconnect with `screen -r screen_name`
  - "Kill" the screen with Ctrl + a + k (then type 'y' to confirm)

Containerization and Docker


IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

A (definitely not personal) anecdote

You're an academic who has just had a paper accepted at a journal!

- The journal requires that all your results replicate **exactly**
- You last ran the code to generate the figures 2-3 years ago
- The same code no longer generates the same results

What do you do!?

- ~~"Dear Head of Department, I am not worthy of this profession…"~~
- ~~Acquire a time machine~~
  - No wait, maybe…

Failures to replicate

Code often fails to replicate when:

- It relies on randomisation, and the RNG changes (e.g. R 3.5 -> 3.6)
- Some package functions get updated (without telling you)
- Cross-OS inconsistencies (in packages, e.g. BART)

We want to create a “container” of OS + applications + packages

- Hold fixed all versions etc.
- Hold fixed underlying computation regardless of hardware
- Load this setup whenever we want

We could do this with AWS, if we were careful

- But remember, AMIs get updated

Docker

A platform for running code/application in “a loosely isolated environment”

- Lightweight(ish)
- Self-contained
- Consistent

Sign up for a Docker account here

- Then download Docker Desktop here

Images + Containers

Docker also uses images

- The ``stack" of OS + apps you want
- Lots of pre-written docker images

A **container** is the runnable instances of your image:

- Can be run locally or on the cloud
- Can be run on any OS
- (Almost) isolated from the rest of your system

Basic Docker workflow

1. **Pull** a Docker image
   - Go to Docker Hub or Docker Desktop to search available images
   - At the command line, use `docker pull`:
     - E.g. `docker pull rocker/r-ver` loads a pre-built R image
2. **Run** the image as a container:
   - `docker run -ti rocker/r-ver`
     - `-ti`: makes your container interactive (`i`) with a shell terminal (`t`)
     - Many more arguments to change runtime

Closing/stoppping/removing a container

Close the container by pressing Ctrl + a + d

- Exits the contianer, but it is still running

Stop the container running:

- Verify which container is running: `docker ps -a`
- Copy the name of the server
- Run `docker stop <container_name>`

Remove the container from your system:

- Run `docker rm <container_name>`

Alternatively, we can use `--rm` flag to close the container on exit by default:

- E.g. `docker run -ti --rm rocker/r-ver`

Mounting volumes

Our container is isolated from the rest of our system:

- What happens in the container, stays in the container

But sometimes useful to read/write to a specific part of our own system

- We do so by **mounting** a storage volume *when we run* the container
  - E.g. `docker run -ti –rm -v “$(pwd)”:/home rocker/r-ver
    - Format is `<path/on/host>:<path/in/container>`
    - `"$(pwd)"` is our local working directory

Writing & building custom images

Sometimes, we want to configure our own image:

- E.g. add specific package requirements
  - Maybe even specific versions of packages
- Add custom setup steps

We write a `Dockerfile` that Docker uses to configure a new image

- Key commands include:
  - `FROM`: adapt an existing image
  - `RUN`: call specific scripts or commands
  - `ENV`: set environment variables
  - `WORKDIR`: change your default working directory

Building Dockerfiles

Once we are happy with our configuration, we **build** the image with:

```
docker build -t my472 .
```

- `-t` allows you to give your image a 'tag' (in this case `my472`)
- `.` tells Docker that your Dockerfile is in the current directory

The first build can take some time

- Docker caches *steps* to speed-up rebuilding
- If you want to add new packages, can be better to add new steps
- Then consolidate when finished

Running RStudio docker container

Our example Dockerfile sets up an RStudio server app:

- We add specific arguments to interact with the container via our browser:
- `docker run -ti --rm -v "$(pwd)":/home/rstudio -p 8787:8787 my472`
  - `-p 8787:8787`: links a host port to a port inside the container
  - `-ti`, `--rm`, and `-v` as before

Using Docker in the cloud

The same isolated environment can be run on any system (with Docker)

- Use AWS EC2 instance as host
- Pull and run a Docker container on server instance
    - Ensures *exact* environment at any point if using the same image

If you have built a custom image and want to access it on the cloud:

- Use `docker push` to send your image to Docker Hub
- More info here

The end

Quick reminders:

- Use free tier instances for testing/learning whenever possible
- **Stop/terminate them after use**
- Keep an eye on costs via e.g. `Services` -> `Cost Explorer` and `Services` -> `Billing`

This term:

- Data munging -> online data scraping/storage/access -> cloud computing
- We’ve practiced fundamental DS skills: each and every one can be extended
- Keep practicing with them!

```
   *    *  ()   *   *
*        * /\         *
      *   /i\\    *  *
    *     o/\\  *      *
 *       ///\i\    *
   *    /i//\*\      *
        /o/*\\i\   *  *
    * /*////\\\\i\*
   * /i///*/\\\\\o\   *
  *    *   ||     *
```

Have a lovely holiday!