

Research Track I – Course Introduction

Carmine Tommaso Recchiuto

Research Track

Why?

- ✓ Hands-on experiences are required for engineering education.
- ✓ It may be seen how the first step of your Master Thesis work. Research Track I will put the basis for letting you working with robots, giving practical fundamentals in different contexts.
- ✓ Research Track I & Research Track 2.



Course Contents (RT1)



- ✓ Basic skills (I): Linux, git, documentation, Docker
- ✓ Basic skills (II): C++, Python

[Assignment I: 19/10] Deadline: 16/11

- ✓ Robot Operating System (I): nodes, topics, messages, services
- ✓ Robot Operating System (II): additional functionalities, custom messages and services, actions.

[Assignment I: 23 or 30/11] Deadline: 21/12

- ✓ Simulation environments: Gazebo
- ✓ Visualization tool: RViz
- ✓ ROS and navigation – mapping and motion planning: gmapping, MoveBase, laser sensors, obstacle avoidance

- Final evaluation: written exam (open-ended / closed –ended questions + practical exercises)

Methodology

Teaching methods consist of:

- Frontal lessons with practical examples (Wednesday 14:00 – 16:00) in presence and online via the TEAMS platform.
- There will be room for some exercises during classes
- An additional class will take place on a Friday morning (November or December)



Assignments

- 1st assignment: an exercise involving a simple robot simulator in python
- 2nd assignment: a simple software architecture in ROS, involving the usage of the publish/subscribe and client/server architecture

Evaluation

Each assignment must be done individually and it should be:

- submitted as a separate Git repository with: the ROS-based code
a report as a README file
- The final exam will have some questions related to the assignments done, and to general aspects seen during the course
- In the final exam, you will also required to do some simple exercises based on the contents of the course (in particular related to the last weeks of the course)
- The evaluation will analyze different aspects of the development of the systems (architecture design, robot behaviour, structure and clarity of the code, structure and clarity of the repository, the ability to motivate the proposed solutions and discuss drawbacks and benefits)



GIT – Distributed Version Control Systems

Carmine Tommaso Recchiuto

Version Control System?

A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time. Developers can review project history to find out:

Which changes were made?
Who made the changes?
When were the changes made?
Why were changes needed?



Distributed Version Control System?

A distributed version control system (DVCS) allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes.

Unlike once popular centralized version control systems, DVCSs don't need a constant connection to a central repository.

Developers can work anywhere and collaborate asynchronously from any time zone.



Distributed Version Control System!

Without version control, team members are subject to redundant tasks, slower timelines, and multiple copies of a single project.

To eliminate unnecessary work, DVCSs give each contributor a unified and consistent view of a project, surfacing work that's already in progress.

Seeing a transparent history of changes, who made them, and how they contribute to the development of a project helps team members stay aligned while working independently.



GIT

Git is an example of a distributed version control system (DVCS) commonly used for open source and commercial software development.

Git is the most-used VCS in the world!

Git is commonly used for both open source and commercial software development, with significant benefits for individuals, teams and businesses.



Some Terminology

Repository: encompasses the entire collection of files and folders associated with a project, along with each file's revision history.

Branching: divergence from the main line of development. It's a method to update the code without messing with the one of the main line.

Merge: the contents of a source branch are integrated with a target branch. In this process, only the target branch is changed.

Fork: copy of the repository. Forking a repository allows you to freely experiment with changes without affecting the original project.



Some History

1990 – Concurrent Versions System (CVS): revision control system with a client-server architecture.

The server stores the current version of the project and its history, and clients connect to the server in order to “check out” a copy of the project.

It supported distributed operations (the server only accepts changes made to the most recent version of a file). The CVS client may handle conflicts.



Some History

2000 – Subversion (SVN): open source version control system (it fixed the bugs and supplied some features missing in CVS).

New features:

- ***Branching***
- Possibility to rename or move files
- Versioning of symbolic links
- Versioning for directories, renames and file metadata



Some History

2000 – Bitkeeper: commercial software for version control.

The first, real, distributed system: repositories could be **forked** and **merged** easily (Subversion could do forks and merges only as major, time-consuming operations)

It has been used for the development of the source code of the Linux kernel.



Git

2006 – GIT: free and open source DCVS

Git lets developers see the entire timeline of their changes, decisions, and progression of any project in one place. From the moment they access the history of a project, the developer has all the context they need to understand it and start contributing.



Why Git?

With Subversion, you have a Problem: if the SVN Repository is in a location you can't reach (e.g., you don't have internet at the moment), you cannot commit. If you want to make a copy of your code, you have to literally copy/paste it.

With Git, you do not have this problem. Your local copy is a repository, and you can commit to it and get all benefits of source control. When you regain connectivity to the main repository, you can commit against it.

Git

2006 – GIT: free and open source DCVS

Git tracks **content rather than files**.

Branches are lightweight and merging is *easy*,

Git repositories are much **smaller in file size** than Subversion repositories.
There's only one ".git" directory, as opposed to dozens of ".svn"
Repositories.

Git facilitates this through the use of topic branches: lightweight pointers to commits in history that can be easily created and deprecated when no longer needed.



Git - Handbook



Basic GIT Commands:

git init initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control

git clone creates a local copy of a project that already exists remotely. The clone includes all the project's files, history, and branches

git add stages a change. Git tracks changes to a developer's codebase, but it's necessary to stage and take a snapshot of the changes to include them in the project's history. This command performs staging, the first part of a two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project's history. Staging and committing separately gives developers complete control over the history of their project without changing how they code

Git - Handbook



Basic GIT Commands:

git commit saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that's been staged with `git add` will become a part of the snapshot with `git commit`

git status shows the status of changes as untracked, modified, or staged.

git branch shows the branches being worked on locally.

git merge merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the main branch for deployment.

Git - Handbook



Basic GIT Commands:

git pull updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.

git push updates the remote repository with any commits made locally to a branch

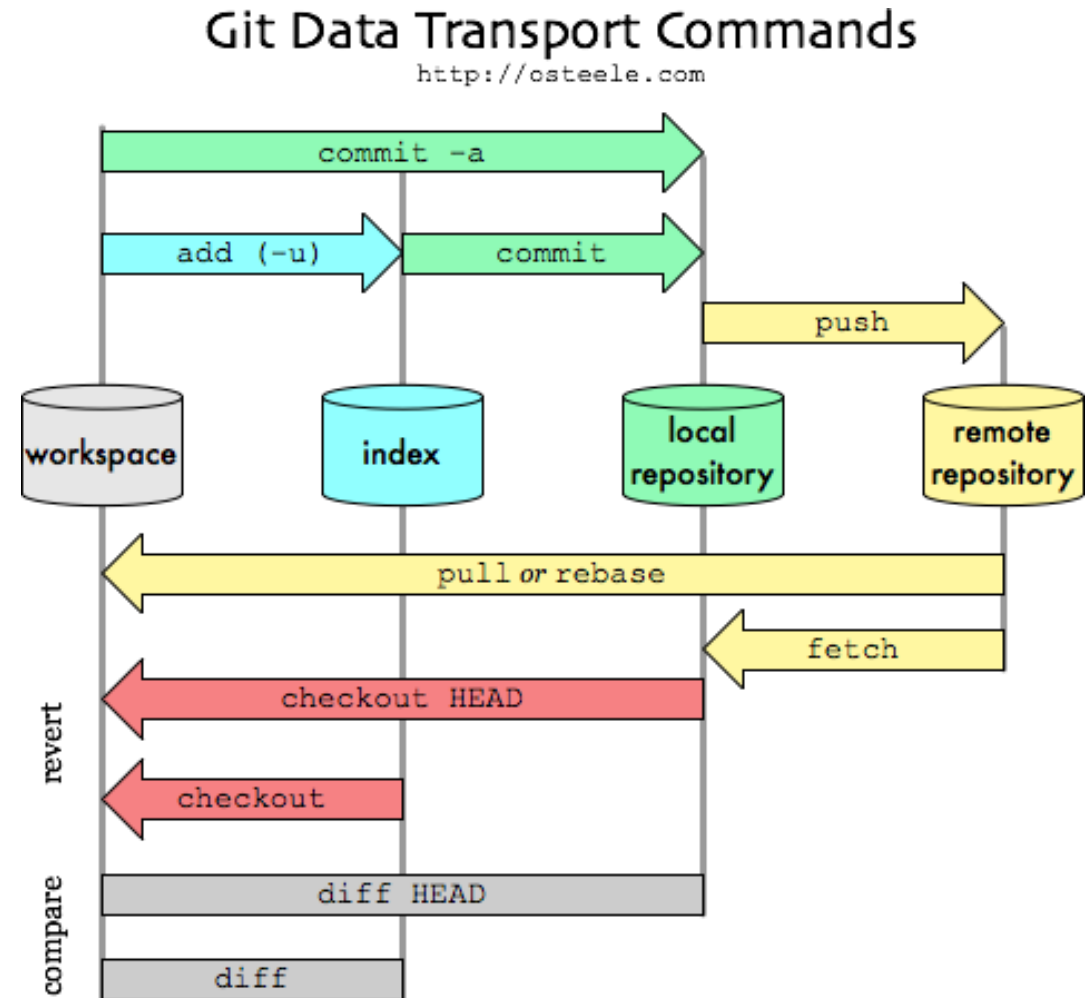
git fetch gathers any commits from the target branch that do not exist in your current branch and stores them in your local repository. However, it does not merge them with your current branch.

Git - Handbook

git checkout 'checks out' of an existing branch and view another branch of code. Checkout is essential for working on a new branch, existing branch, or remote branch

git diff runs a diff functions between different data sources

Please refer to <https://git-scm.com/docs> for a full handbook about all git commands



Example

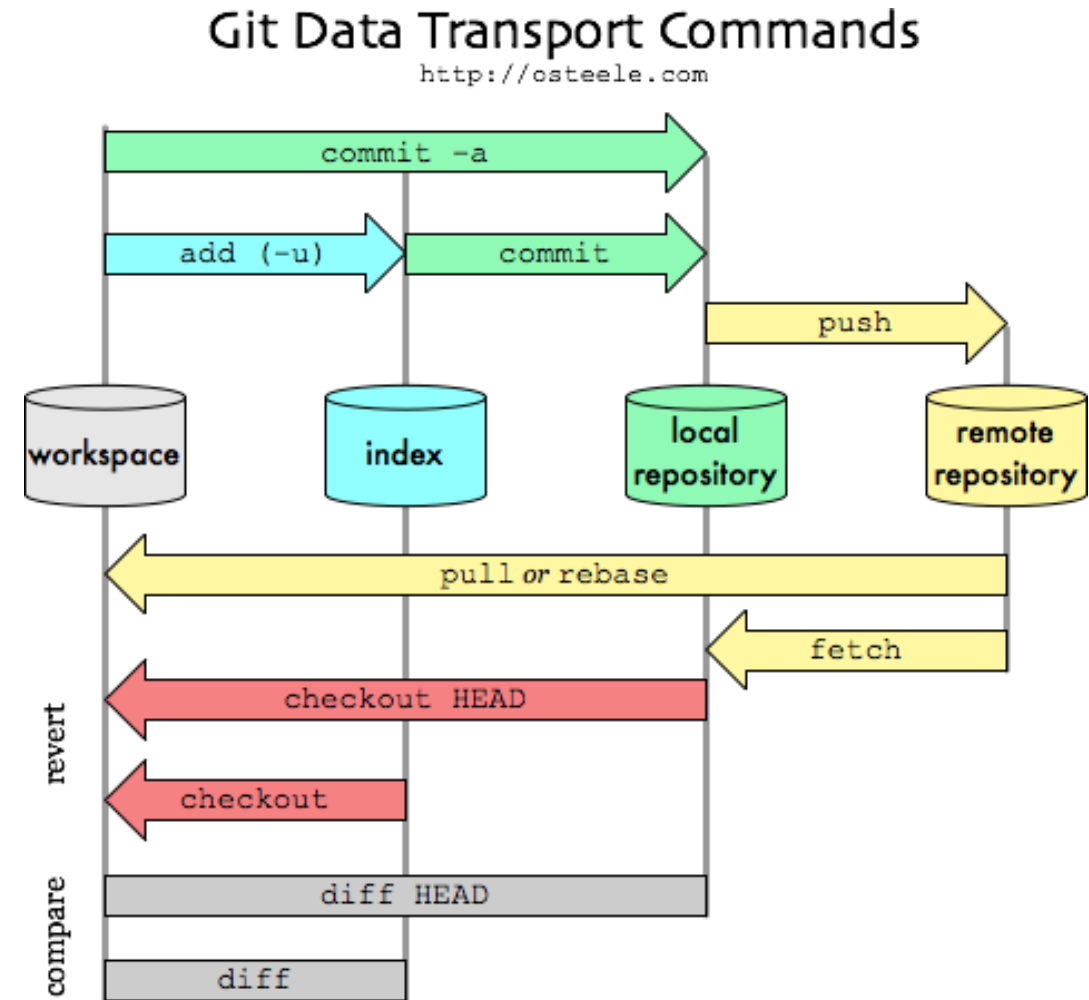
<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git> -> first install Git!

```
$ cd project  
$ git init
```

Git will reply
Initialized empty Git repository in .git/

Next, tell Git to take a snapshot of the contents of all files under the current directory (note the .), with git add:

```
$ git add .
```



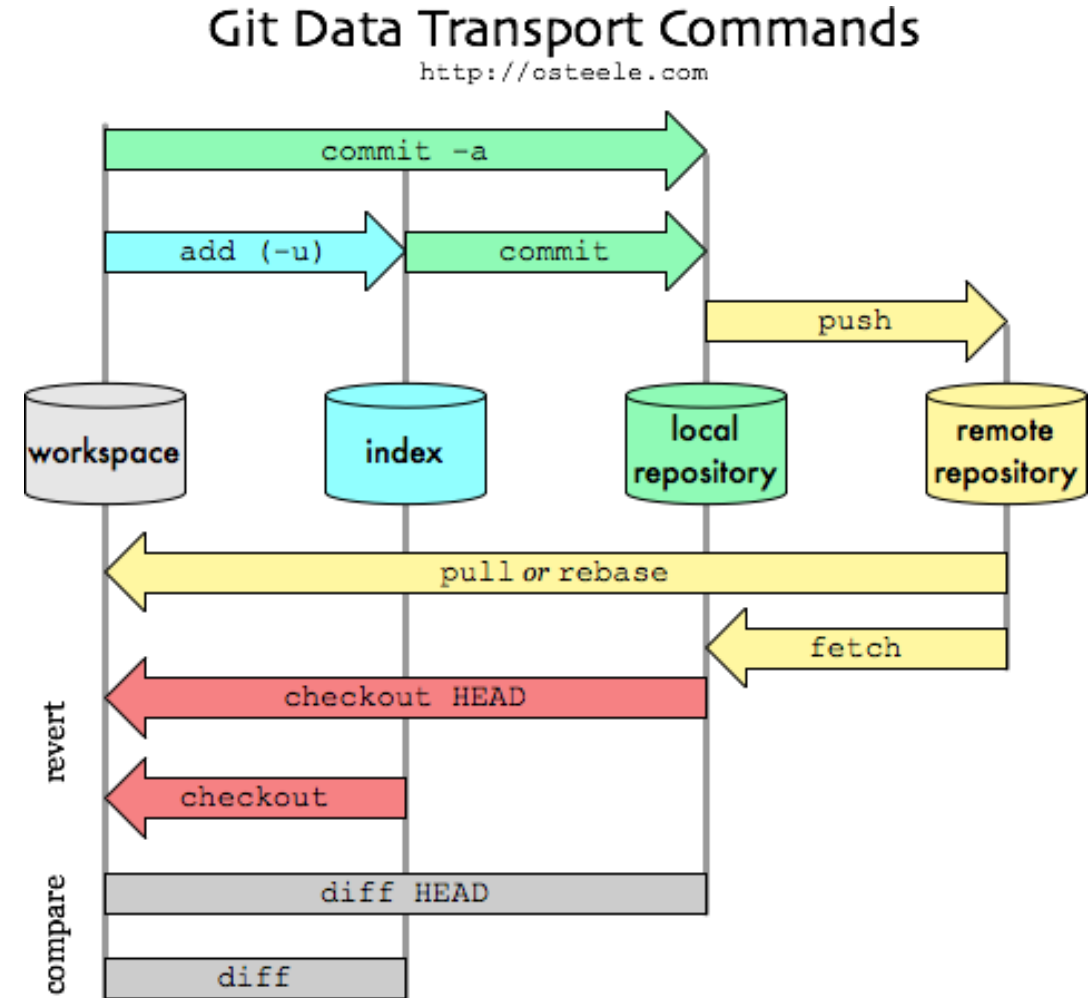
Example

This snapshot is now stored in a temporary staging area which Git calls the "index". You can permanently store the contents of the index in the repository with git commit:

```
$ git commit
```

This will prompt you for a commit message. By default, the vi editor will be used to write the commit message.

You've now stored the first version of your project in Git!



Git Hosting Repositories

However, everything is still in your local repo! To work with a remote repository, we need a Git hosting repository.

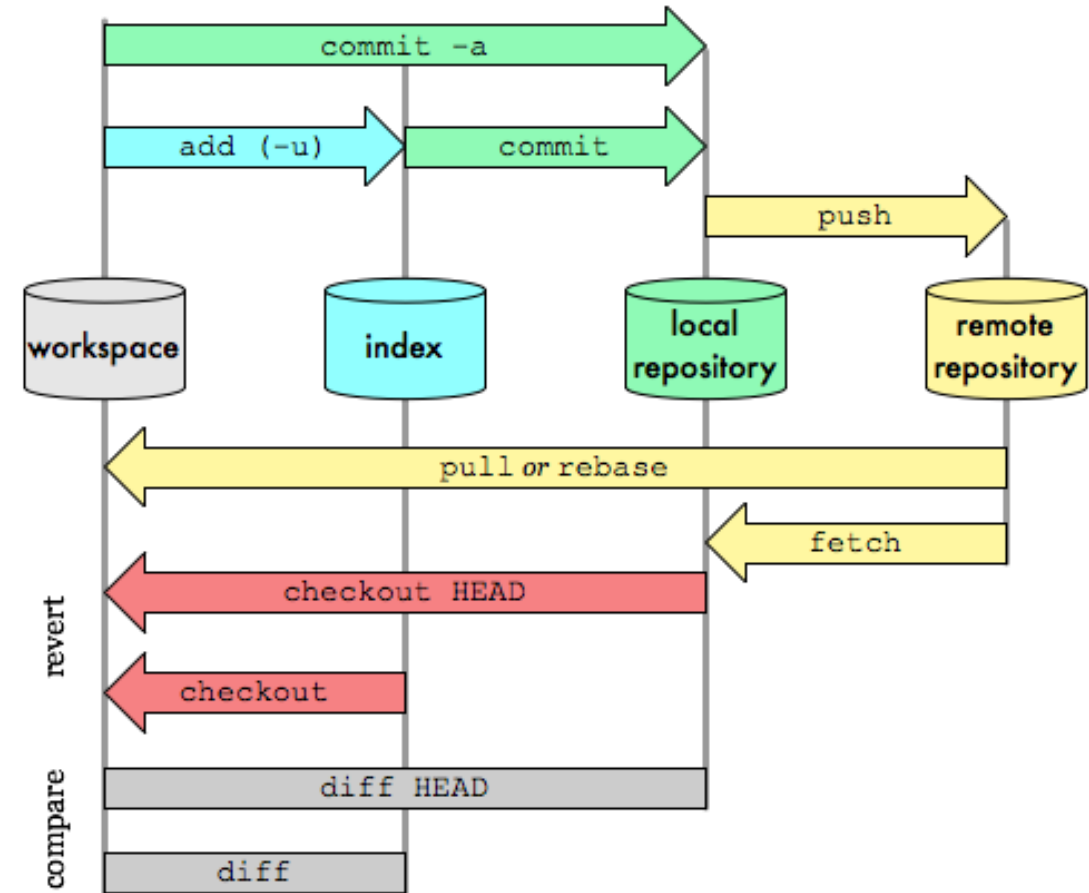
Git hosting repositories provide developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, code review, or the use of a collection of free and for-purchase apps in the Marketplace.

Some examples are GitHub, GitLab, BitBucket, ...

Let's give a look to GitHub.

Git Data Transport Commands

<http://osteele.com>



GitHub

A repository is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. A README, or a file with information about your project, IS HIGHLY RECOMMENDED!

[Create your account at https://github.com/](https://github.com/)

Add a new repository: <https://github.com/new>

Now we can add the contents of our local repository



Example

We can also add a README.md (md stands for MarkDown, a markup language)

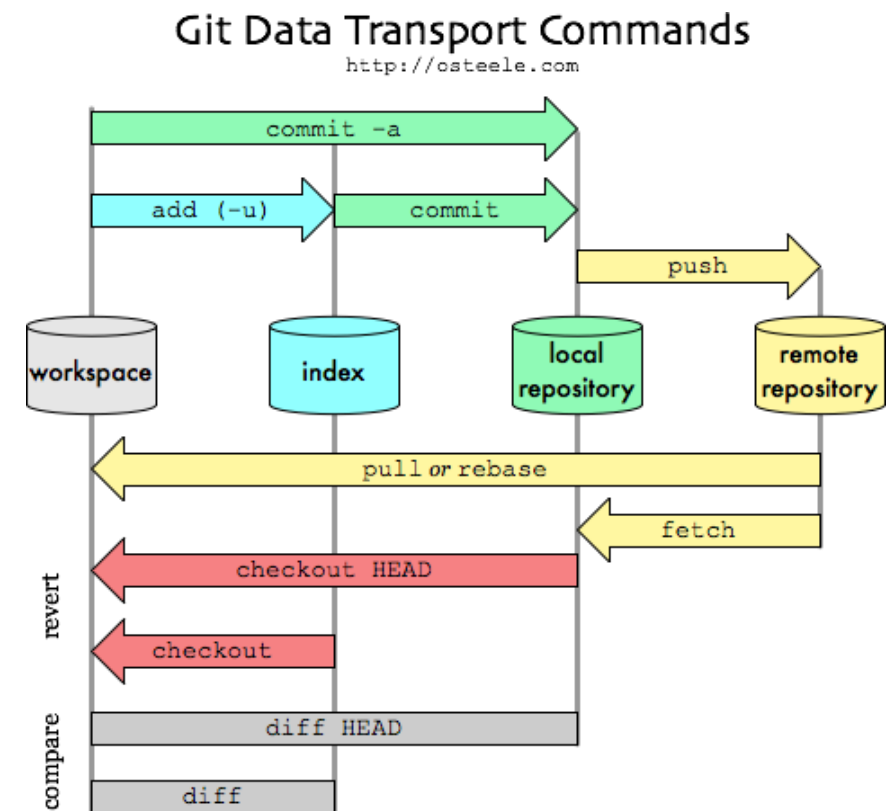
```
$ echo "# fantastic-guide" >> README.md  
$ git add README.md  
$ git commit -m "first commit"
```

Now let's set the current branch where we are working as the Master branch

```
$ git branch -M main
```

Finally, we need to associate a remote repository to our local repository. We use the command:

```
$ git remote add origin <remote_repo_url>
```



Example

Now we need to update the remote repository with the content of the local repository

\$ git push -u origin main

Important: to push something to your repo, you need at first to generate a token: <https://github.com/settings/tokens>

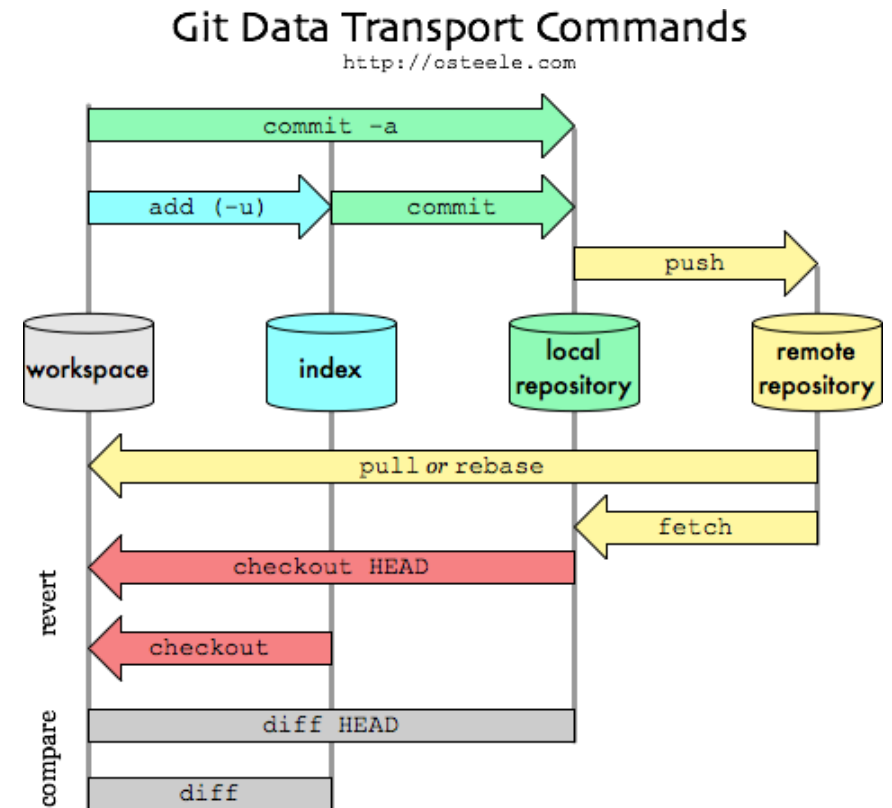
The remote repository is now updated. Every collaborators may consequently upload its local workspace by doing

\$ git clone <remote_repo_url>

Or just

\$ git pull

After the clone, a plain git fetch without arguments will update all the remote-tracking branches, and a git pull without arguments will in addition merge the remote master branch into the current master branch.



Example (2)

So, what if I want to contribute to an existing Git repository?

- Download the repository to your machine

```
$ git clone <remote_repo_url>
```

- Move into the local workspace

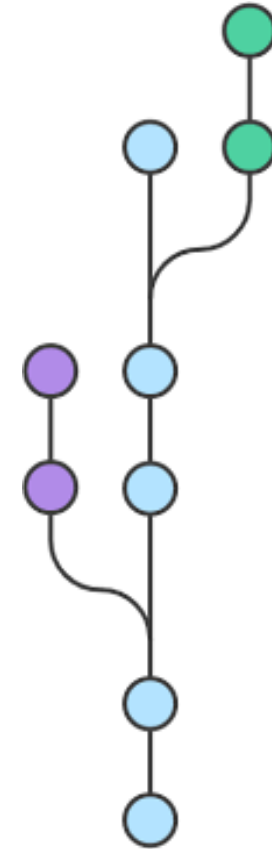
```
$ cd <local_dir_name>
```

- Create a new branch to store any new changes

\$ git branch my-branch

- Switch to that branch (line of development)

\$ git checkout my-branch



Example (2)

Make your changes!

- Stage the changed files

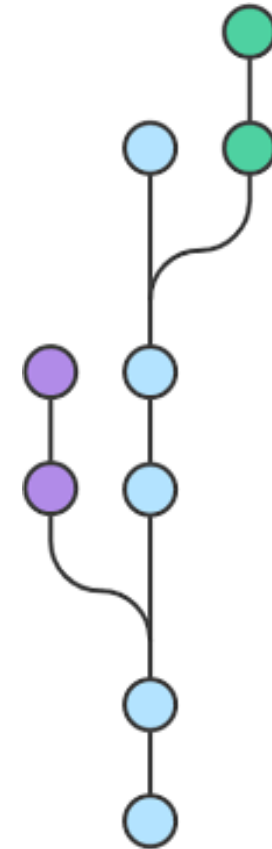
```
$ git add <file1_name> <file2_name>
```

- Take a snapshot of the staging area (and consequently update your local repository)

```
$ git commit -m "my_commit"
```

- Push changes to github

```
$ git push --set-upstream origin my-branch
```



Example (3)

What if I eventually want to merge my changes into the Master branch?

- Switch to the Master branch

\$ git checkout main

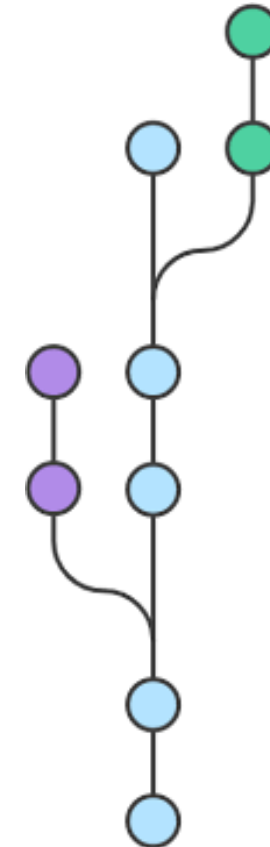
- Merge the branch my-branch on the Master branch

\$ git merge my-branch

- Push changes to github

\$ git push

Now all changes are also visible in the Master branch.



Git Workflows

Git Workflows: recipes or recommendation for how to use Git to accomplish work in a consistent and productive manner.

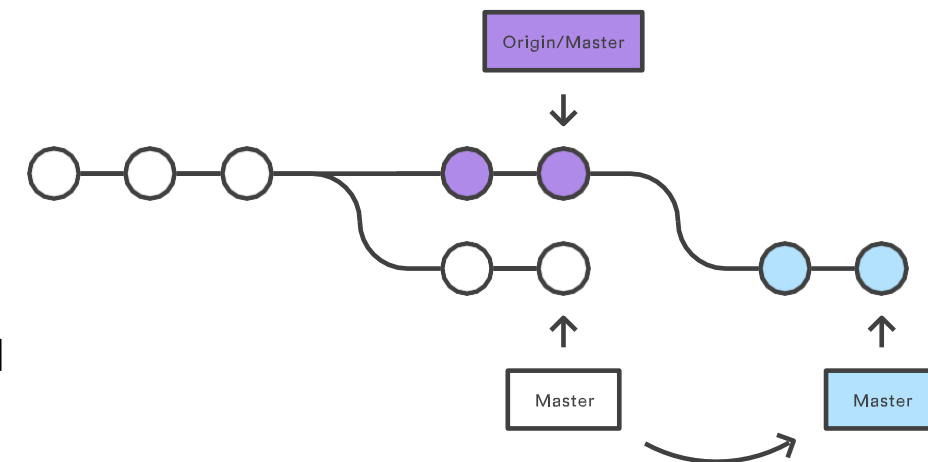
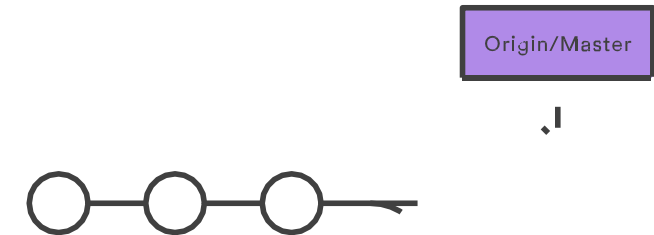
<https://www.atlassian.com/git/tutorials/comparing-workflows>

Examples:

Centralized Workflow

The Centralized Workflow uses a central repository to serve as the single point-of-entry for all changes to the project (only one branch: Master)

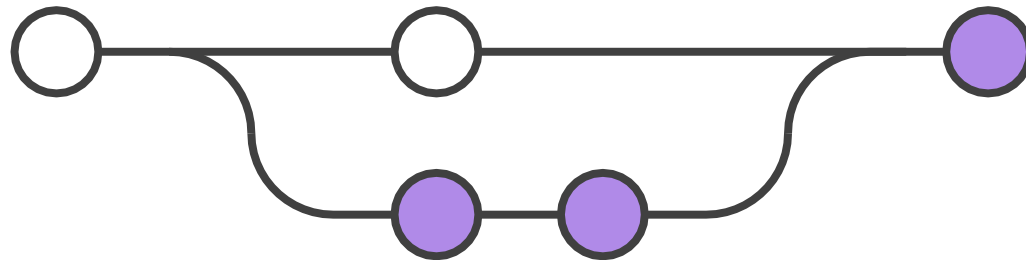
Developers start by cloning the central repository. In their own local copies of the project, they edit files and commit changes; however, these new commits are stored locally - they're completely isolated from the central repository. This lets developers defer synchronizing upstream until they're at a convenient break point.



Git Workflows

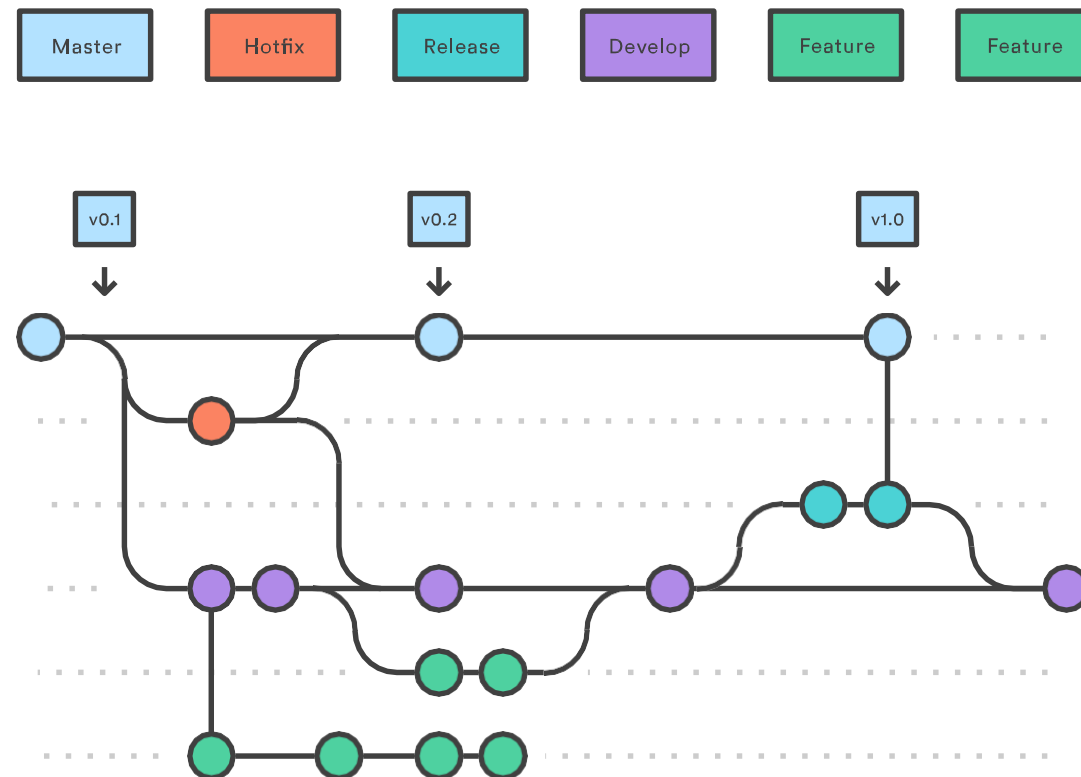
Feature Branch Workflow: all feature development should take place in a dedicated branch instead of the Master branch. This makes it easy for multiple developers to work on a particular feature without disturbing the main codebase.

A merge with the Master branch is done by means of a pull request (in the Git GUI). When a pull request is accepted, the feature is merged into the stable project (Master branch).



Git Workflows

Gitflow Workflow: based on the concept of «releases». It assigns very specific roles to different branches and defines how and when they should interact.



Git GUI Client

In particular when using a workflow involving branches, it may be useful to adopt a Git GUI Client, to have a clear idea of what it is going on.



A GIT GUI Client may be extremely useful to manage multiple branches and to solve conflicts.

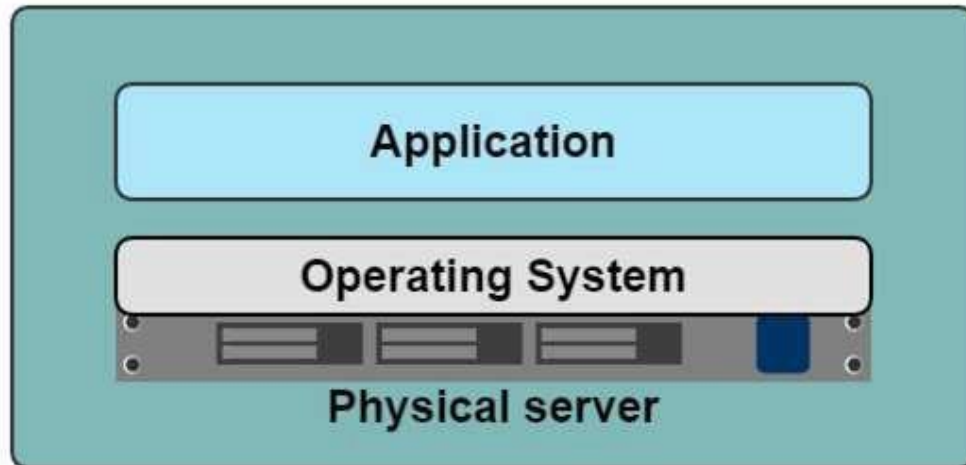
Docker – Virtual Machines and Containers

Carmin Tommaso Recchiuto

Limitations of application deployment

In the Dark Ages

One application on one physical server

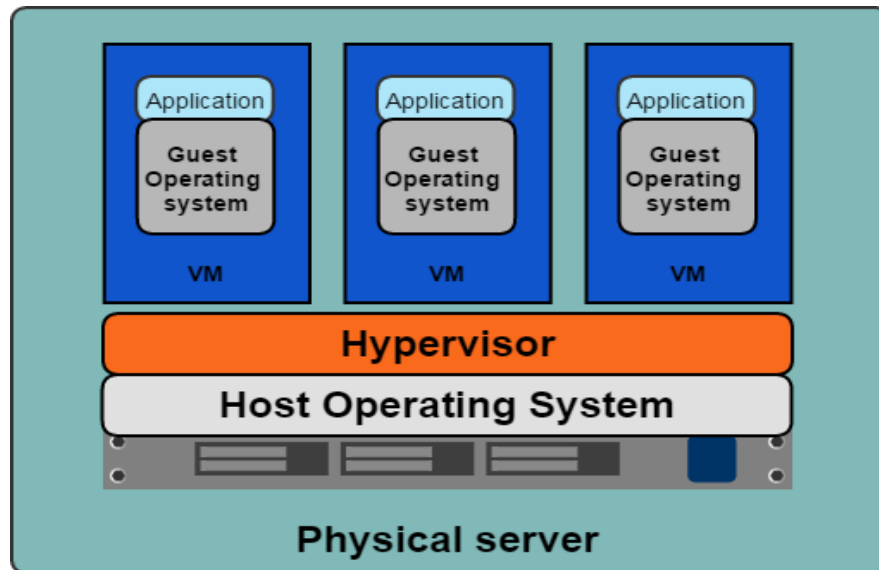


- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Difficult to migrate
- Vendor lock in

Hypervisor-based Virtualization

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)



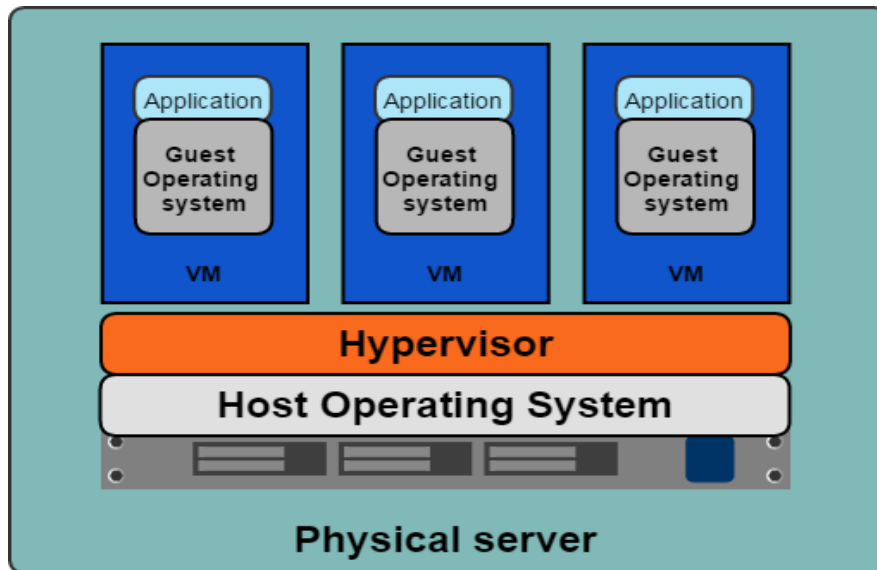
- Better resource pooling
 - One physical machine divided into multiple virtual machines
- Easier to scale
- VMs in the cloud
 - Rapid elasticity
 - Pay as you go model



Hypervisor-based Virtualization

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)

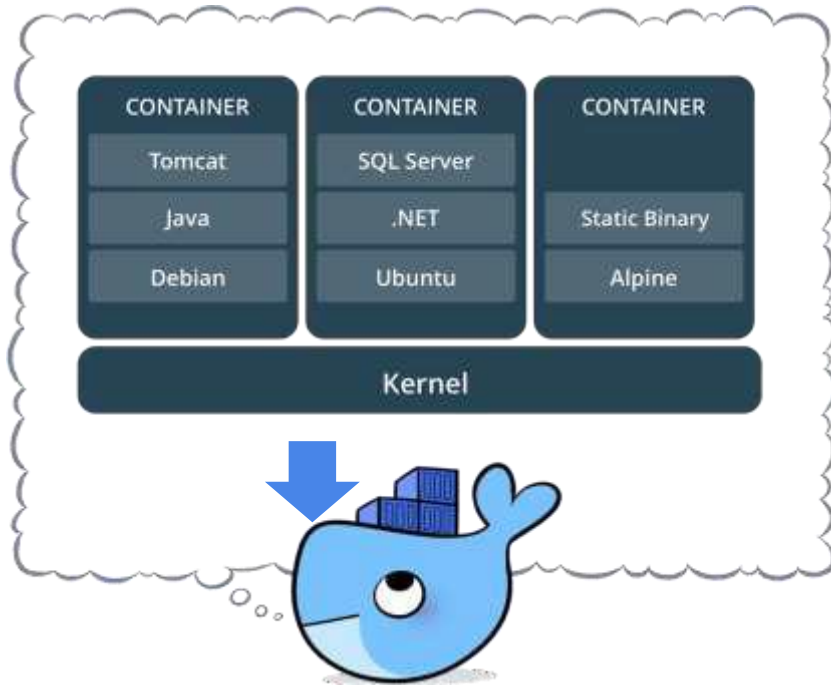


Limitations!

- Each VM stills requires
 - CPU allocation
 - Storage
 - RAM
 - An entire guest operating system
- The more VMs you run, the more resources you need
- Guest OS means wasted resources

Containers

- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works with all major Linux and Windows Server



Containers sit on top of a physical server and its host OS—for example, Linux or Windows.

Each container shares the host OS kernel and, usually, the binaries and libraries, too.

Shared components are read-only.

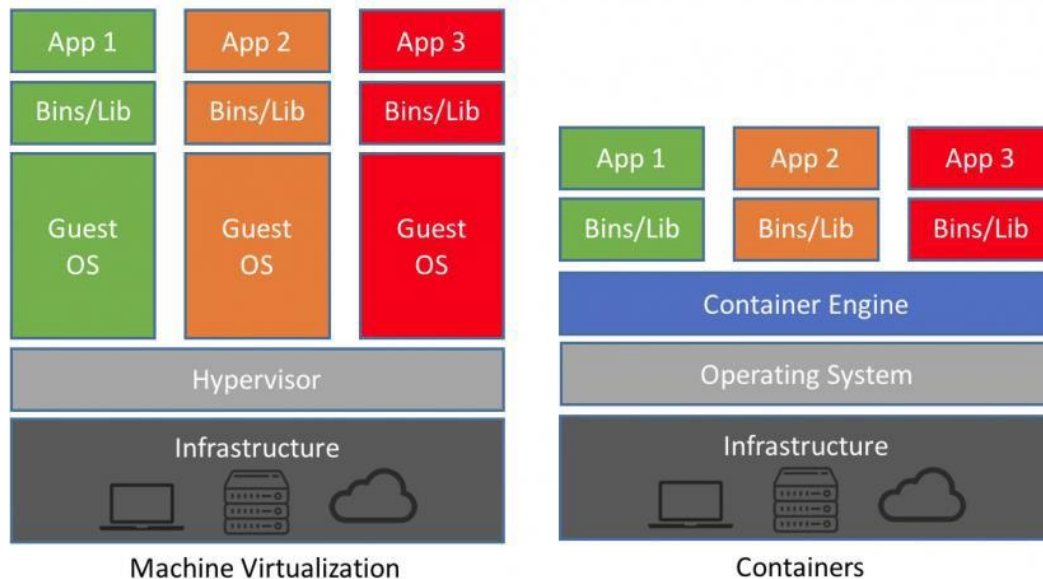
Containers are thus exceptionally “light”—they are only megabytes in size and take just seconds to start, versus gigabytes and minutes for a VM.

Containers

Each virtual machine runs a unique guest operating system, thus VMs with different operating systems can run on the same physical server

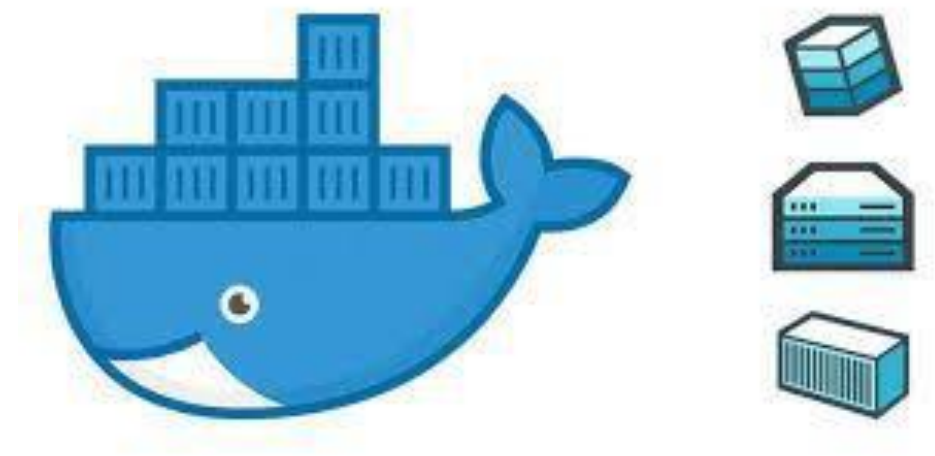
Each VM has its own binaries, libraries, and applications that it services, and the VM may be many gigabytes in size.

Containers reduce management overhead. Because they share a common operating system, only a single operating system needs care and feeding for bug fixes, patches, and so on.



Docker

- ✓ Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. In other words, it is a software platform for building containers: it allows applications to use the same kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer.
- ✓ Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.



Docker



Image

The basis of a Docker container. The content at rest.



Container

The image when it is 'running.' The standard unit for app service



Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.



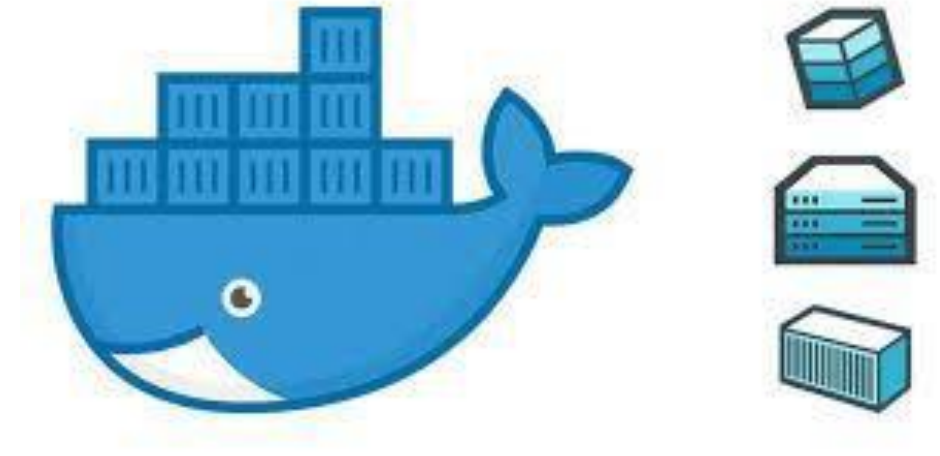
Registry

Stores, distributes and manages Docker images



Control Plane

Management plane for container and cluster orchestration



Its mission is basically: make it easy to package and ship code!

Dockerfile

- ✓ A *Dockerfile* is a text file that contains a bunch of instructions which informs Docker HOW the Docker image should get built.
- ✓ You can relate it to cooking. In cooking you have recipes. A recipe lets you know all of the steps you must take in order to produce whatever you're trying to cook.
 - The act of cooking is building the recipe.
 - A Dockerfile is a recipe for building Docker images



Dockerfile - example

```
$ git clone https://github.com/dockersamples/node-bulletin-board
```

```
$ cd node-bulletin-board/bulletin-board-app
```

```
$ docker build --tag bulletinboard:1.0 .
```

You'll see Docker step through each instruction in your Dockerfile, building up your image as it goes. If successful, the build process should end with a message ***Successfully tagged bulletinboard:1.0***

We can now run the following command to start a container based on this new image:

```
$ docker run --publish 8000:8080 --detach --name bb bulletinboard:1.0
```



Dockerfile - example

The dockerfile defined in this example takes the following steps:

Start FROM the pre-existing node:current-slim image. This is an official image, built by the node.js vendors and validated by Docker to be a high-quality image containing the Node.js Long Term Support (LTS) interpreter and basic dependencies.

Use WORKDIR to specify that all subsequent actions should be taken from the directory /usr/src/app in your image filesystem (never the host's filesystem).

COPY the file package.json from your host to the present location (.) in your image (so in this case, to /usr/src/app/package.json)

RUN the command npm install inside your image filesystem (which will read package.json to determine your app's node dependencies, and install them)

COPY in the rest of your app's source code from your host to your image filesystem.



Dockerfile - example

```
$ docker run --publish 8000:8080 --detach --name bb bulletinboard:1.0
```

--publish asks Docker to forward traffic incoming on the host's port 8000 to the container's port 8080. Containers have their own private set of ports, so if you want to reach one from the network, you have to forward traffic to it in this way. Otherwise, firewall rules will prevent all network traffic from reaching your container, as a default security posture.

--detach asks Docker to run this container in the background.

--name specifies a name with which you can refer to your container in subsequent commands, in this case bb.

Visit your application in a browser at localhost:8000. You should see your bulletin board application up and running.

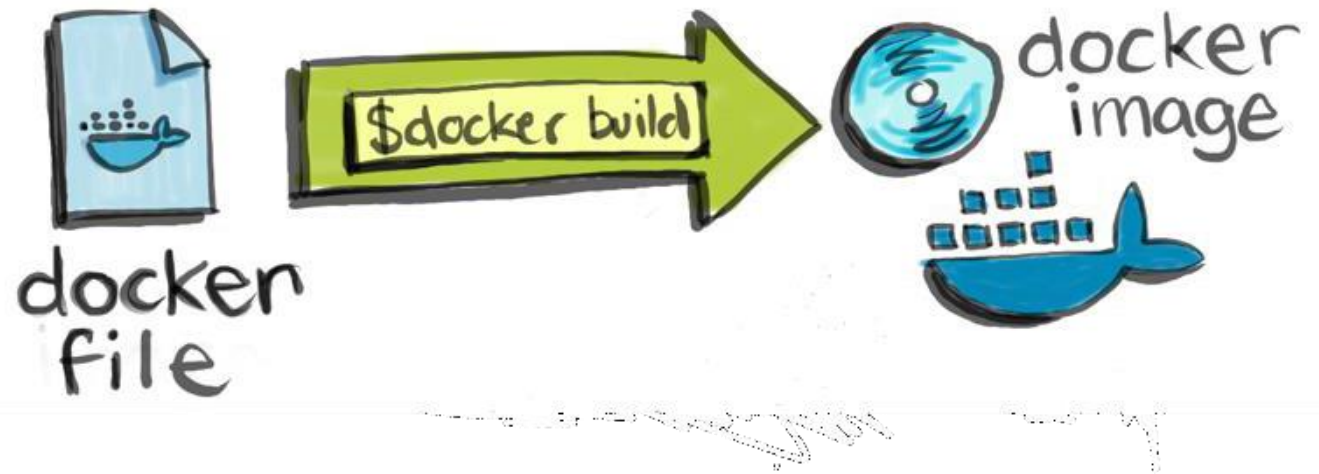


Images and Containers

- ✓ A Docker image gets built by running a Docker command (which uses a Dockerfile)
- ✓ A Docker image is a file, comprised of multiple layers, used to execute code in a Docker container.

\$ docker images lists all the images in your system

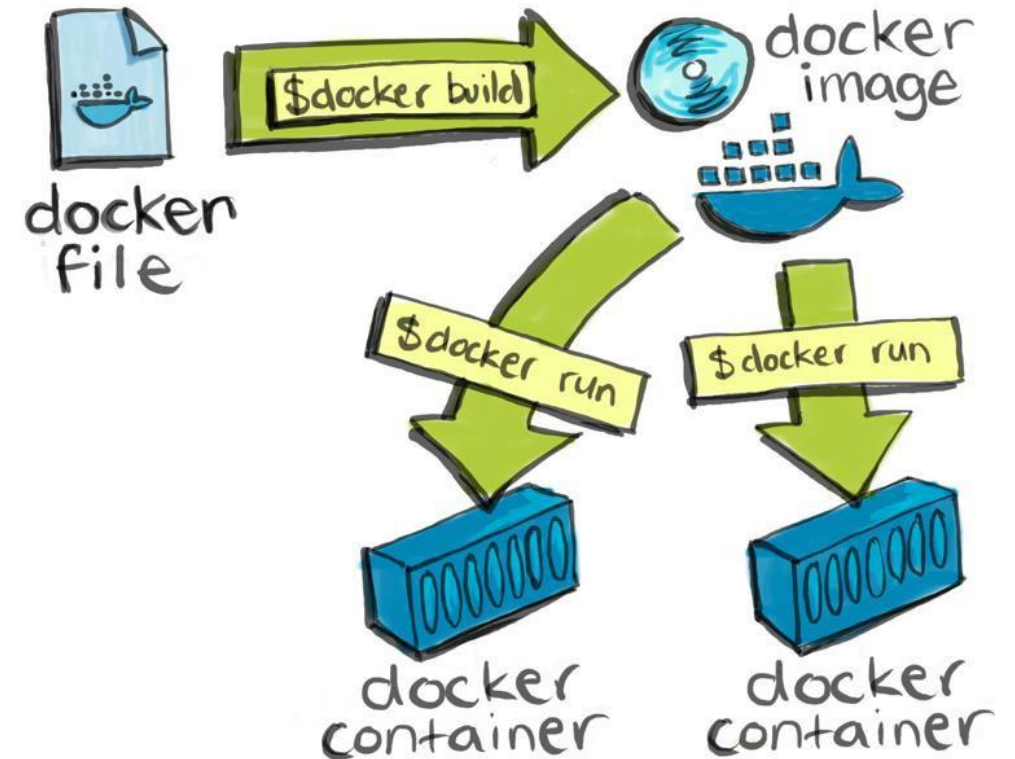
\$ docker rmi <IMAGE ID> may be used to remove a image



Images and Containers

- ✓ The act of running a Docker image creates a Docker container
- ✓ You can think of a Docker image as a class, where as a Docker container is an instance of that class.
- ✓ When a new container is created from an image, a writable layer is also created. This layer is called the container layer, and it hosts all changes made to the running container.

\$ docker run ... creates a container starting from one image



Images and Containers

How to save a modified containers?

Well, you don't need to do that. Once you have done, you may run

\$ docker stop <container_name>

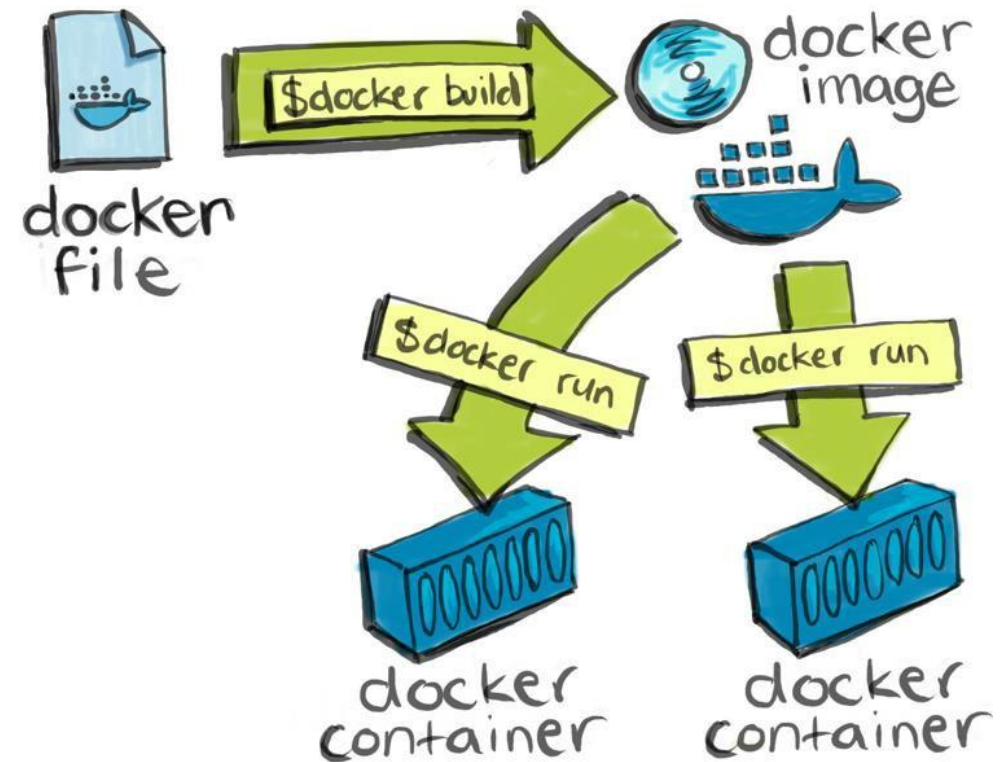
to stop the execution of your container.

When you need again your *living* container, run

\$ docker start <container_name>

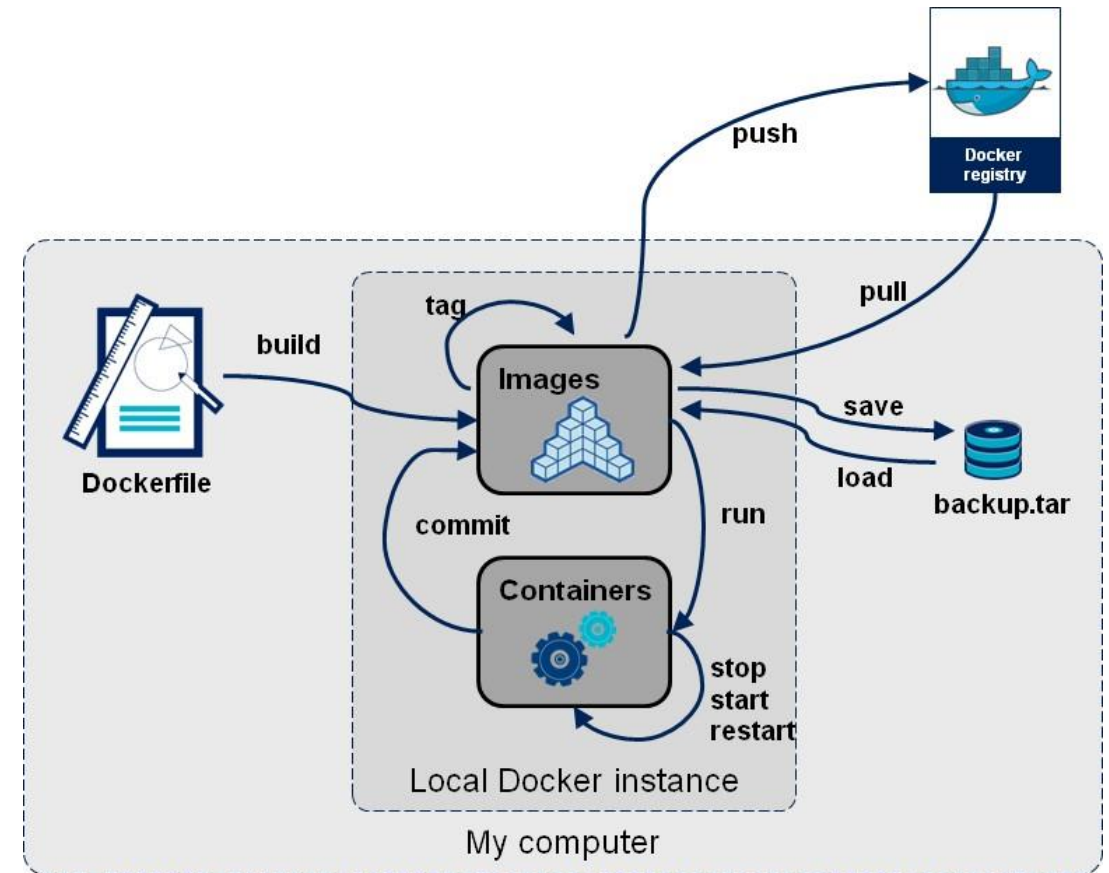
You can also list all containers (active or not)

\$ docker ps -a (docker ps lists only active containers)



Docker Image Repository

- ✓ Docker users store images in private or public repositories and from there can deploy containers, test images and share them. Docker offers Docker Hub, which is a cloud-based registry service that includes private and public image repositories..
- ✓ A user can upload their own custom image to the Docker Hub by using the docker push command. To ensure the quality of community images, Docker reviews the image and provides feedback for the image author before publishing.



References

✓ If you are interested in learning more:

<https://docs.docker.com/>

<https://docker-curriculum.com/>




Research Track - Docker Image


- ✓ Install Docker
 - Follow instructions from:
<https://docs.docker.com/install/>
- ✓ Get the image
 - Run `docker pull carms84/noetic_ros2`
- ✓ Run a container


Follow instructions from
https://hub.docker.com/repository/docker/carms84/noetic_ros2




https://hub.docker.com/repository/docker/carms84/noetic_ros2

 **carms84 / noetic_ros2**



This repository does not have a description 

 Last pushed: an hour ago

Tags and Scans

 VULNERABILITY SCANNING - DISABLED [Enable](#)

This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
 latest		an hour ago	an hour ago

[See all](#)

Docker commands [Public View](#)

To push a new tag to this repository,



```
docker push carms84/noetic_ros2:tagname
```

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available on Pro and Team plans.

[Upgrade to Pro](#) [Learn more](#)

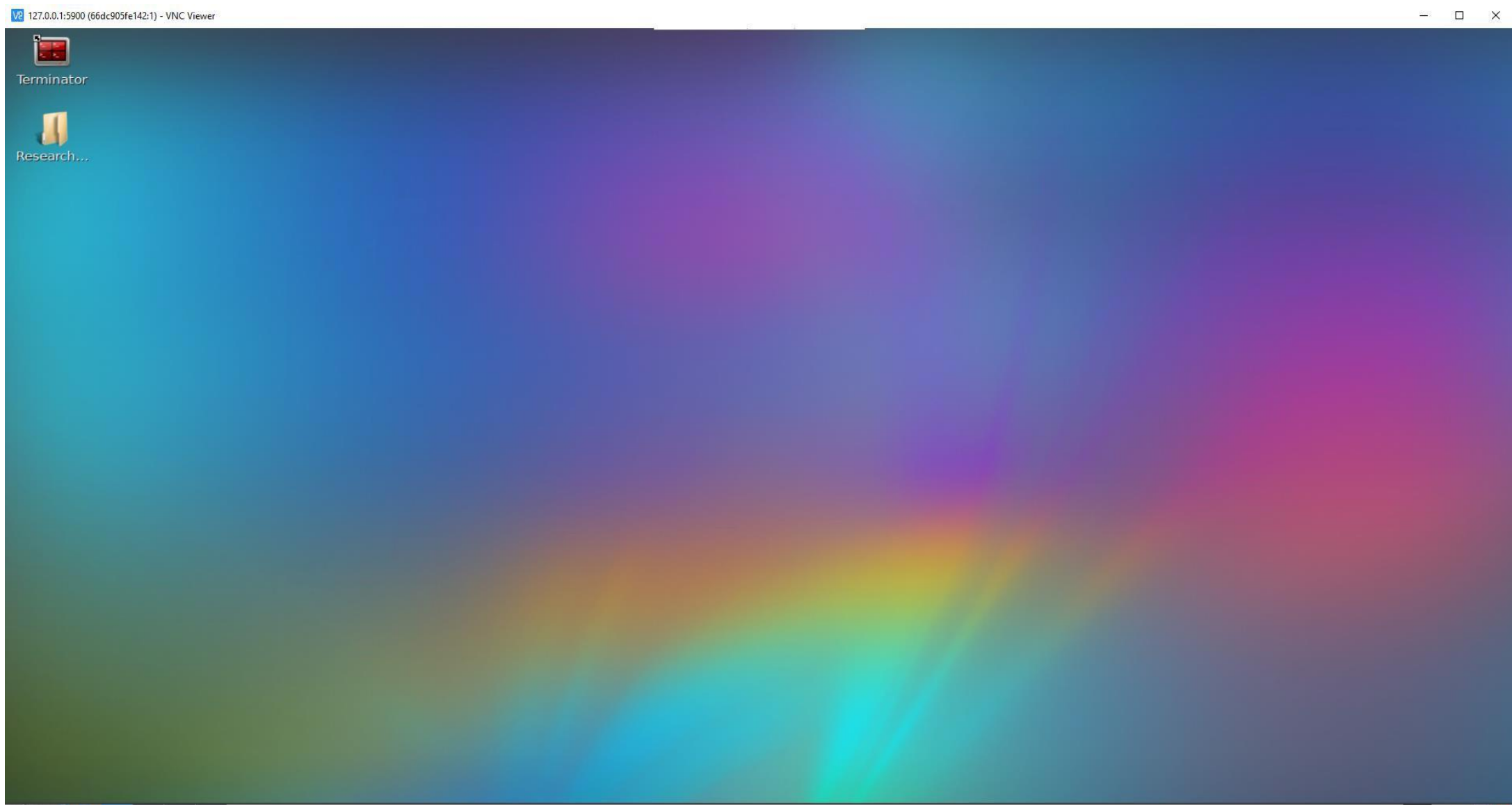
Readme  

Docker container used in the context of robotic programming courses @ Unige

The image has been developed starting from a docker image providing HTML5 VNC interface to access ROS Noetic on Ubuntu 20.04 with the LXDE desktop environment.

Run it by: `docker run -it --name my_ros -p 6080:80 -p 5900:5900 carms84/noetic_ros2`

You may then open a browser and type `http://127.0.0.1:6080/` or use a VNC client (port 5900) to connect to the VNC server.



How to start

✓ If you have already:

- a Native Linux System, with ROS installed
- a Virtual Machine, with Linux-ROS
- a Docker image with Linux-ROS

you may use that.

✓ Otherwise, I suggest to:

- On Windows 10 Pro – Enterprise, Linux, MacOS, install Docker and pull the Docker image of the course
- For other Windows versions, you may:
 - Use Docker Toolbox and the Docker image of the Course **OR**
 - Use a Virtual Machine with Linux and run Docker inside the Virtual Machine.

✓ In this last scenario, follow the instructions to install Docker Toolbox here:

https://docs.docker.com/toolbox/toolbox_install_windows/

✓ Or you may install VirtualBox (<https://www.virtualbox.org/>) and a Linux image for Virtual Box

https://www.virtualbox.org/wiki/Linux_Downloads

Known Issue

If executing: **`docker run -it -p 6080:80 -p 5900:5900 --name my_ros carms84/noetic_ros2`**, you get an error as :

```
INFO exited: xvfb (exit status 1; not expected)
INFO gave up: xvfb entered FATAL state, too many start retries too quickly
INFO exited: pcmanfm (exit status 1; not expected)
INFO gave up: pcmanfm entered FATAL state, too many start retries too quickly
INFO exited: lxpanel (exit status 1; not expected)
INFO gave up: lxpanel entered FATAL state, too many start retries too quickly
INFO exited: lxsession (exit status 1; not expected)
```

run the following commands from your terminal:

```
$ docker start my_ros
$ docker exec -it my_ros /bin/bash
$ rm /tmp/.X1-lock
$ /usr/bin/Xvfb :1 -screen 0 1024x768x16 &
$ export DISPLAY=":1"
$ export HOME="/root"
$ export USER="/root"
$/usr/bin/openbox &
$/usr/bin/lxpanel --profile LXDE &
$/usr/bin/pcmanfm --desktop --profile LXDE &
$ x11vnc -display :1 -xkb -forever -shared -repeat &
```


Images and Containers

Once done all this commands, your container will be visible using a vnc viewer or a browser.

Please notice that this is required only the first time you launch the container. When finished working with it, you can just close it

\$ docker stop my_ros

And start it when you need it again

\$ docker start my_ros