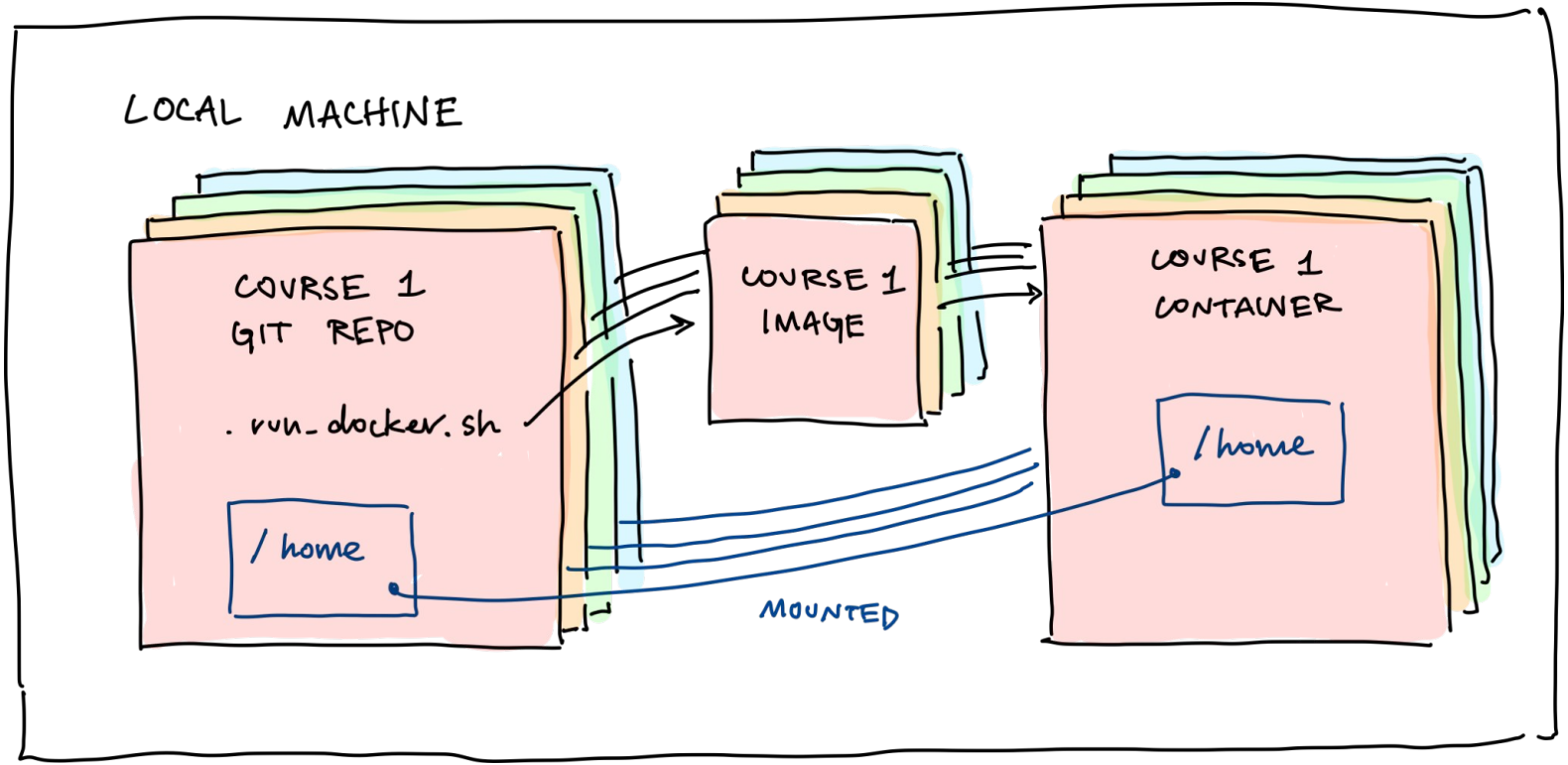
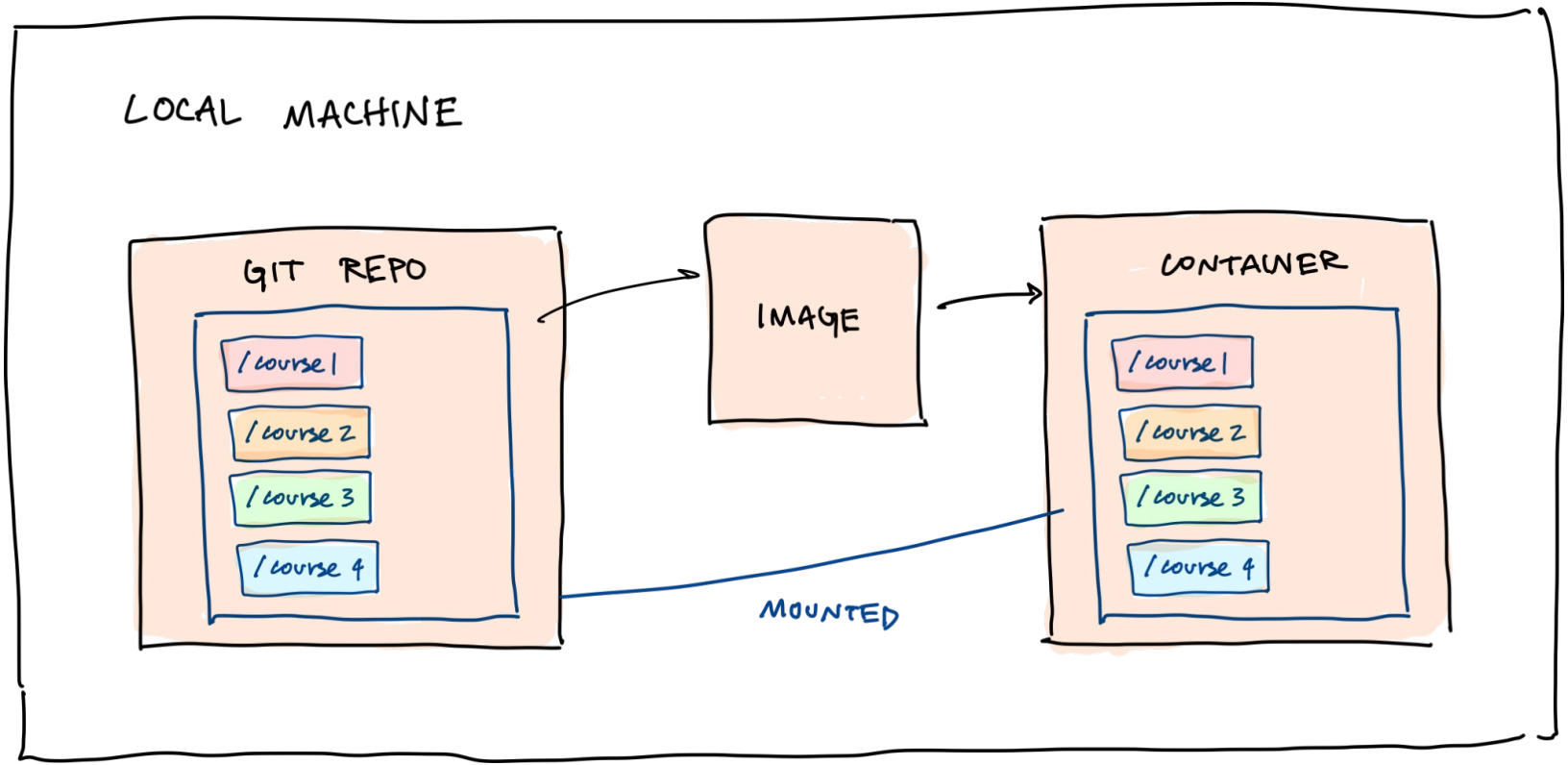
Common Course Containers


LOCAL MACHINE
COURSE 1
GIT REPO
. run_docker.sh
/ home
COURSE 1
IMAGE
COURSE 1
CONTAINER
/ home
MOUNTED


LOCAL MACHINE
GIT REPO
/course1
/course2
/course3
/course4
IMAGE
CONTAINER
/course1
/course2
/course3
/course4
MOUNTED

25% of CS courses provide containers

Course	Dev environment
CSCI 0300 (Computer Systems)	Docker Container
CSCI 1670 (Operating Systems)	Podman/Docker Container
CSCI 1660 (Computer Security)	Docker Container
CSCI 1680 (Computer Networks)	Docker Container
CSCI 1270 (Database Management Systems)	Docker Container
CSCI 1515 (Applied Cryptography)	Codespaces, Docker Container
CSCI 0220 (Discrete Structures and Probability)	Codespaces, Docker Container
CSCI 1260 (Compilers)	Codespaces, Docker Container
CSCI 1810 (Computational Molecular Biology)	Docker Container
CSCI 1951Y (Computer Architecture)	Docker Container
CSCI 1675 (High-Performance Network Systems)	Docker Container

Containers across CS courses are already very **similar**.

Course	Dev environment	OS Version
CSCI 0300 (Computer Systems)	Docker Container	ubuntu:noble ubuntu:jammy (Podmanfile)
CSCI 1670 (Operating Systems)	Podman/Docker Container	debian:12.2
CSCI 1660 (Computer Security)	Docker Container	ubuntu:jammy
CSCI 1680 (Computer Networks)	Docker Container	ubuntu:jammy
CSCI 1270 (Database Management Systems)	Docker Container	ubuntu:jammy
CSCI 1515 (Applied Cryptography)	Codespaces, Docker Container	ubuntu:focal
CSCI 0220 (Discrete Structures and Probability)	Codespaces, Docker Container	ubuntu:jammy
CSCI 1260 (Compilers)	Codespaces, Docker Container	debian:stable
CSCI 1810 (Computational Molecular Biology)	Docker Container	ubuntu:noble
CSCI 1951Y (Computer Architecture)	Docker Container	ubuntu:jammy
CSCI 1675 (High-Performance Network Systems)	Docker Container	n/a

Course	Dev environment	OS Version
CSCI 0300 (Computer Systems)	Docker Container	ubuntu:noble ubuntu:jammy (Podmanfile)
CSCI 1670 (Operating Systems)	Podman/Docker Container	debian:12.2
CSCI 1660 (Computer Security)	Docker Container	ubuntu:jammy
CSCI 1680 (Computer Networks)	Docker Container	ubuntu:jammy
CSCI 1270 (Database Management Systems)	Docker Container	ubuntu:jammy
CSCI 1515 (Applied Cryptography)	Codespaces, Docker Container	ubuntu:focal
CSCI 0220 (Discrete Structures and Probability)	Codespaces, Docker Container	ubuntu:jammy
CSCI 1260 (Compilers)	Codespaces, Docker Container	debian:stable
CSCI 1810 (Computational Molecular Biology)	Docker Container	ubuntu:noble
CSCI 1951Y (Computer Architecture)	Docker Container	ubuntu:jammy
CSCI 1675 (High-Performance Network Systems)	Docker Container	n/a

Majority of courses use Ubuntu…

Course	Dev environment	OS Version
CSCI 0300 (Computer Systems)	Docker Container	ubuntu:noble ubuntu:jammy (Podmanfile)
CSCI 1670 (Operating Systems)	Podman/Docker Container	debian:12.2
CSCI 1660 (Computer Security)	Docker Container	ubuntu:jammy
CSCI 1680 (Computer Networks)	Docker Container	ubuntu:jammy
CSCI 1270 (Database Management Systems)	Docker Container	ubuntu:jammy
CSCI 1515 (Applied Cryptography)	Codespaces, Docker Container	ubuntu:focal
CSCI 0220 (Discrete Structures and Probability)	Codespaces, Docker Container	ubuntu:jammy
CSCI 1260 (Compilers)	Codespaces, Docker Container	debian:stable
CSCI 1810 (Computational Molecular Biology)	Docker Container	ubuntu:noble
CSCI 1951Y (Computer Architecture)	Docker Container	ubuntu:jammy
CSCI 1675 (High-Performance Network Systems)	Docker Container	n/a

… or a compatible version of Debian

Course	Dev environment	OS Version
CSCI 0300 (Computer Systems)	Docker Container	ubuntu:noble ubuntu:jammy (Podmanfile)
CSCI 1670 (Operating Systems)	Podman/Docker Container	debian:12.2
CSCI 1660 (Computer Security)	Docker Container	ubuntu:jammy
CSCI 1680 (Computer Networks)	Docker Container	ubuntu:jammy
CSCI 1270 (Database Management Systems)	Docker Container	ubuntu:jammy
CSCI 1515 (Applied Cryptography)	Codespaces, Docker Container	ubuntu:focal
CSCI 0220 (Discrete Structures and Probability)	Codespaces, Docker Container	ubuntu:jammy
CSCI 1260 (Compilers)	Codespaces, Docker Container	debian:stable
CSCI 1810 (Computational Molecular Biology)	Docker Container	ubuntu:noble
CSCI 1951Y (Computer Architecture)	Docker Container	ubuntu:jammy
CSCI 1675 (High-Performance Network Systems)	Docker Container	n/a

Systems courses already use a similar setup…

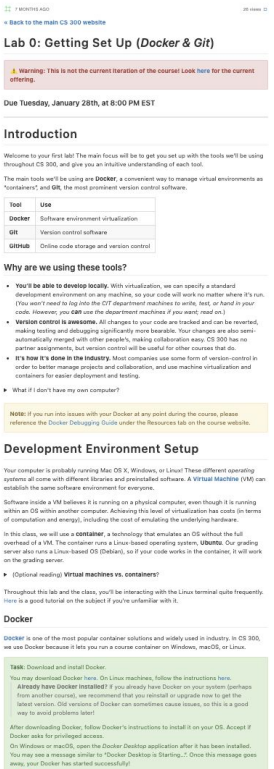
Course	Dev environment	OS Version
CSCI 0300 (Computer Systems)	Docker Container	ubuntu:noble ubuntu:jammy (Podmanfile)
CSCI 1670 (Operating Systems)	Podman/Docker Container	debian:12.2
CSCI 1660 (Computer Security)	Docker Container	ubuntu:jammy
CSCI 1680 (Computer Networks)	Docker Container	ubuntu:jammy
CSCI 1270 (Database Management Systems)	Docker Container	ubuntu:jammy
CSCI 1515 (Applied Cryptography)	Codespaces, Docker Container	ubuntu:focal
CSCI 0220 (Discrete Structures and Probability)	Codespaces, Docker Container	ubuntu:jammy
CSCI 1260 (Compilers)	Codespaces, Docker Container	debian:stable
CSCI 1810 (Computational Molecular Biology)	Docker Container	ubuntu:noble
CSCI 1951Y (Computer Architecture)	Docker Container	ubuntu:jammy
CSCI 1675 (High-Performance Network Systems)	Docker Container	n/a

… which has been adopted by courses like 1515 and 1270

There are a lot of courses that use containers, but…

1. Setting up takes a long time
2. Technical limitations with running multiple containers
3. Containers can take a lot of storage
4. Students often need additional support with containers

1. **Setting up** takes a long time



« Back to the main CS 300 website

Lab 0: Getting Set Up (*Docker & Git*)

⚠ Warning: This is not the current iteration of the course! Look here for the current offering.

Due Tuesday, January 28th, at 8:00 PM EST

Introduction

Welcome to your first lab! The main focus will be to get you set up with the tools we'll be using throughout CS 300, and give you an intuitive understanding of each tool.

The main tools we'll be using are **Docker**, a convenient way to manage virtual environments as "containers," and **Git**, the most prominent version control software.

Tool	Use
Docker	Software environment virtualization
Git	Version control software
GitHub	Online code storage and version control

Why are we using these tools?

- **You'll be able to develop locally.** With virtualization, we can specify a standard development environment on any machine, so your code will work no matter where it's run. (*You won't need to log into the CIT department machines to write, test, or hand in your code. However, you can use the department machines if you want; read on.*)
- **Version control is awesome.** All changes to your code are tracked and can be reverted, making testing and debugging significantly more bearable. Your changes are also semi-automatically merged with other people's, making collaboration easy. CS 300 has no partner assignments, but version control will be useful for other courses that do.
- **It's how it's done in the industry.** Most companies use some form of version-control in order to better manage projects and collaboration, and use machine virtualization and containers for easier deployment and testing.

▸ What if I don't have my own computer?

Note: If you run into issues with your Docker at any point during the course, please reference the Docker Debugging Guide under the Resources tab on the course website.

Development Environment Setup

Your computer is probably running Mac OS X, Windows, or Linux! These different *operating systems* all come with different libraries and preinstalled software. A Virtual Machine (VM) can establish the same software environment for everyone.

Software inside a VM believes it is running on a physical computer, even though it is running within an OS within another computer. Achieving this level of virtualization has costs (in terms of computation and energy), including the cost of emulating the underlying hardware.

In this class, we will use a **container**, a technology that emulates an OS without the full overhead of a VM. The container runs a Linux-based operating system, **Ubuntu**. Our grading server also runs a Linux-based OS (Debian), so if your code works in the container, it will work on the grading server.

▸ (Optional reading) **Virtual machines vs. containers?**

Throughout this lab and the class, you'll be interacting with the Linux terminal quite frequently. Here is a good tutorial on the subject if you're unfamiliar with it.

Docker

Docker is one of the most popular container solutions and widely used in industry. In CS 300, we use Docker because it lets you run a course container on Windows, macOS, or Linux.

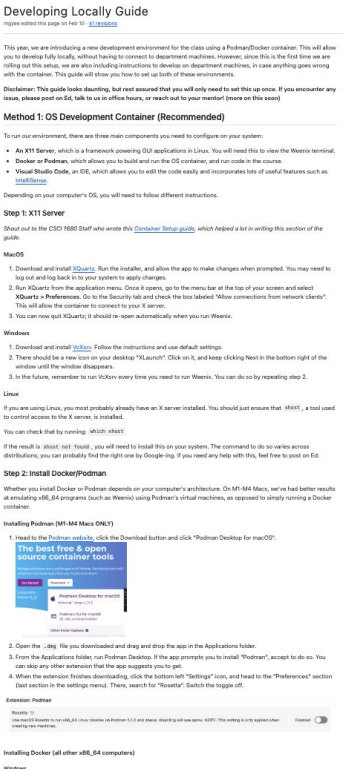
Task: Download and install Docker.

You may download Docker here. On Linux machines, follow the instructions here.

Already have Docker installed? If you already have Docker on your system (perhaps from another course), we recommend that you reinstall or upgrade now to get the latest version. Old versions of Docker can sometimes cause issues, so this is a good way to avoid problems later!

After downloading Docker, follow Docker's instructions to install it on your OS. Accept if Docker asks for privileged access.

On Windows or macOS, open the *Docker Desktop* application after it has been installed. You may see a message similar to "Docker Desktop is Starting...". Once this message goes away, your Docker has started successfully!



Developing Locally Guide

This year, we are introducing a new development environment for the class using a Podman/Docker container. This will allow you to develop fully locally, without having to connect to department machines. However, since this is the first time we are rolling out this setup, we are also including instructions to develop on department machines, in case anything goes wrong with the container. This guide will show you how to set up both of these environments.

Disclaimer: This guide looks daunting, but rest assured that you will only need to set this up once. If you encounter any issue, please post on Ed, talk to us in office hours, or reach out to your mentor! (more on this soon)

Method 1: OS Development Container (Recommended)

To run our environment, there are three main components you need to configure on your system:

- **An X11 Server**, which is a framework powering GUI applications in Linux. You will need this to view the Weenix terminal.
- **Docker or Podman**, which allows you to build and run the OS container, and run code in the course.
- **Visual Studio Code**, an IDE, which allows you to edit the code easily and incorporates lots of useful features such as IntelliSense.

Depending on your computer's OS, you will need to follow different instructions.

Step 1: X11 Server

Shout out to the CSCI 1680 Staff who wrote this Container Setup guide, which helped a lot in writing this section of the guide.

MacOS

1. Download and install XQuartz. Run the installer, and allow the app to make changes when prompted. You may need to log out and log back in to your system to apply changes.
2. Run XQuartz from the application menu. Once it opens, go to the menu bar at the top of your screen and select **XQuartz > Preferences**. Go to the Security tab and check the box labeled "Allow connections from network clients". This will allow the container to connect to your X server.
3. You can now quit XQuartz; it should re-open automatically when you run Weenix.

Windows

1. Download and install VcXsrv. Follow the instructions and use default settings.
2. There should be a new icon on your desktop "XLaunch". Click on it, and keep clicking Next in the bottom right of the window until the window disappears.
3. In the future, remember to run VcXsrv every time you need to run Weenix. You can do so by repeating step 2.

Linux

If you are using Linux, you most probably already have an X server installed. You should just ensure that `xhost` , a tool used to control access to the X server, is installed.

You can check that by running `which xhost`

If the result is `xhost not found` , you will need to install this on your system. The command to do so varies across distributions; you can probably find the right one by Google-ing. If you need any help with this, feel free to post on Ed.

Step 2: Install Docker/Podman

Whether you install Docker or Podman depends on your computer's architecture. On M1-M4 Macs, we've had better results at emulating x86_64 programs (such as Weenix) using Podman's virtual machines, as opposed to simply running a Docker container.

Installing Podman (M1-M4 Macs ONLY)

1. Head to the Podman website, click the Download button and click "Podman Desktop for macOS".

2. Open the `.dmg` file you downloaded and drag and drop the app in the Applications folder.
3. From the Applications folder, run Podman Desktop. If the app prompts you to install "Podman", accept to do so. You can skip any other extension that the app suggests you to get.
4. When the extension finishes downloading, click the bottom left "Settings" icon, and head to the "Preferences" section (last section in the settings menu). There, search for "Rosetta". Switch the toggle off.

Extension: Podman

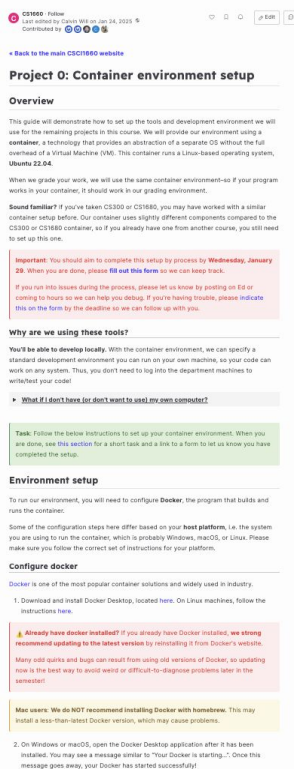
Rosetta

Use macOS Rosetta to run x86_64 Linux binaries on Podman 5.1.0 and above, disabling will use qemu. NOTE: This setting is only applied when creating new machines.

Disabled

Installing Docker (all other x86_64 computers)

Windows



CS1660 · Follow
Last edited by Calvin Wit on Jan 24, 2025
Contributed by

« Back to the main CSCI1660 website

Project 0: Container environment setup

Overview

This guide will demonstrate how to set up the tools and development environment we will use for the remaining projects in this course. We will provide our environment using a **container**, a technology that provides an abstraction of a separate OS without the full overhead of a Virtual Machine (VM). This container runs a Linux-based operating system, **Ubuntu 22.04**.

When we grade your work, we will use the same container environment–so if your program works in your container, it should work in our grading environment.

Sound familiar? If you've taken CS300 or CS1680, you may have worked with a similar container setup before. Our container uses slightly different components compared to the CS300 or CS1680 container, so if you already have one from another course, you still need to set up this one.

Important: You should aim to complete this setup by process by **Wednesday, January 29**. When you are done, please **fill out this form** so we can keep track.

If you run into issues during the process, please let us know by posting on Ed or coming to hours so we can help you debug. If you're having trouble, please indicate this on the form by the deadline so we can follow up with you.

Why are we using these tools?

You'll be able to develop locally. With the container environment, we can specify a standard development environment you can run on your own machine, so your code can work on any system. Thus, you don't need to log into the department machines to write/test your code!

▸ **What if I don't have (or don't want to use) my own computer?**

Task: Follow the below instructions to set up your container environment. When you are done, see this section for a short task and a link to a form to let us know you have completed the setup.

Environment setup

To run our environment, you will need to configure **Docker**, the program that builds and runs the container.

Some of the configuration steps here differ based on your **host platform**, i.e. the system you are using to run the container, which is probably Windows, macOS, or Linux. Please make sure you follow the correct set of instructions for your platform.

Configure docker

Docker is one of the most popular container solutions and widely used in industry.

1. Download and install Docker Desktop, located here. On Linux machines, follow the instructions here.

⚠ **Already have docker installed?** If you already have Docker installed, **we strong recommend updating to the latest version** by reinstalling it from Docker's website.

Many odd quirks and bugs can result from using old versions of Docker, so updating now is the best way to avoid weird or difficult-to-diagnose problems later in the semester!

Mac users: We do NOT recommend installing Docker with homebrew. This may install a less-than-latest Docker version, which may cause problems.

2. On Windows or macOS, open the Docker Desktop application after it has been installed. You may see a message similar to "Your Docker is starting...". Once this message goes away, your Docker has started successfully!

Steps to set up Dev-Environment and docker image:
(This guide uses parts of the setup guide from CS1660 and CS0300. Feel free to check out their guide for additional debugging tips during the dev-environment setup process as well. Big thanks to Professor DeMarinis for his help.)

Why dev-environment?
Last year, every student was allowed to do local development on their native systems, but this ended up causing setup problems for each assignment for the entire year. As such, we decided to try using a dev-environment this year with linux. This way, bugs with setup will hopefully be avoidable or at least standardized between different operating systems. An additional benefit of this is that once you set up your dev-environment, Go, Git, and SQLite should automatically be installed in your dev-environment.

Download Docker

Download and install Docker Desktop, located here. On Linux machines, follow the instructions here.

Already have docker installed? If you already have Docker installed, we recommend updating to the latest version by reinstalling it from Docker's website. Old Docker can sometimes cause issues.

Note: if docker asks for privileged access, say yes.

Mac users: We do NOT recommend installing Docker with homebrew. This may install a less-than-latest Docker version, which may cause problems.

Windows Only: Install WSL

To run the following steps in this setup, you will need to set up Windows Subsystem for Linux (WSL). WSL should already be enabled after you install Docker, but you may still need to install a Linux distribution. This will run in an actual Linux VM, and you will run your Docker container within that VM (turtles all the way down for you!).

1. **Do I have a Linux distribution (Linux distro) installed?**
   - Run *wsl -l -v* in the Command Prompt or Powershell. If there is only "Docker Desktop" and "Docker Desktop Data", you do not have a Linux distribution installed. Proceed to step 2.
   - Otherwise, you have a Linux distro installed. Proceed to step 3.
2. **Install a Linux Distribution.**
   - Run *wsl --set-default-version 2* to ensure Ubuntu will be installed under WSL 2.
   - Install "Ubuntu 22.04" from Microsoft Store.
   - Click "Open" after Ubuntu is downloaded. A terminal will open and guide you through the installation process.
3. **Ensure your Linux Distribution runs on WSL 2.**
   - From the output of *wsl -l -v*, find out if your Linux distro is using WSL 1 or WSL 2. If it's WSL1:
     - Run *wsl --set-version <distro name> 2* to update your distro to use WSL 2.

1. **Setting up** takes a long time

Course #1

1. Install Docker or Podman
2. Clone course repository
3. Download image
4. Run container
5. Set up text editor
6. Set up git credentials

Course #2

2. Clone course repository
3. Download image
4. Run container
5. Set up text editor
6. Set up git credentials

Course #3

2. Clone course repository
3. Download image
4. Run container
5. Set up text editor
6. Set up git credentials

1. **Setting up** takes a long time

Course #1

1. Install Docker or Podman
2. Clone course repository
3. Download image
4. Run container
5. Set up text editor
6. Set up git credentials

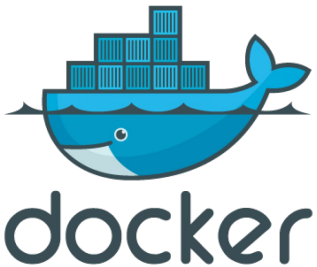
Course #2

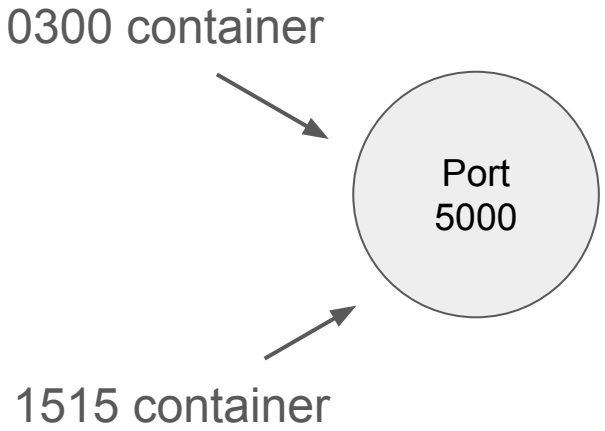
4. Run container
5. Clone course repository
6. Download packages

Course #3

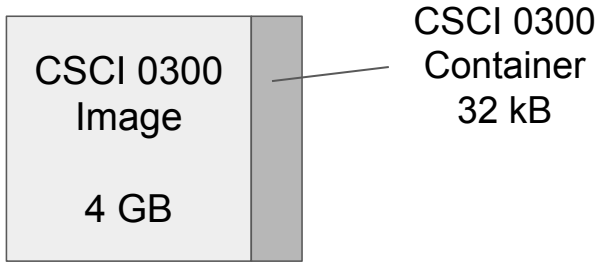
4. Run container
5. Clone course repository
6. Download packages

2. Technical limitations with running **multiple containers**

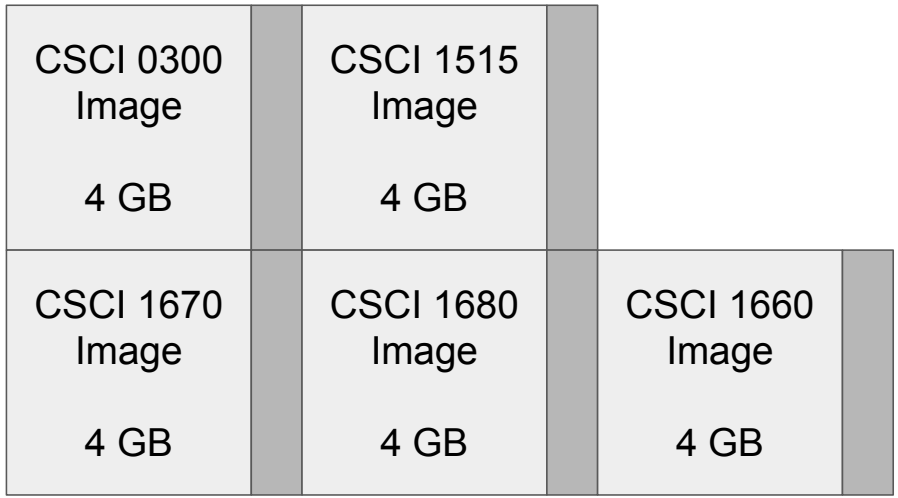




3. Containers can take a lot of **storage**

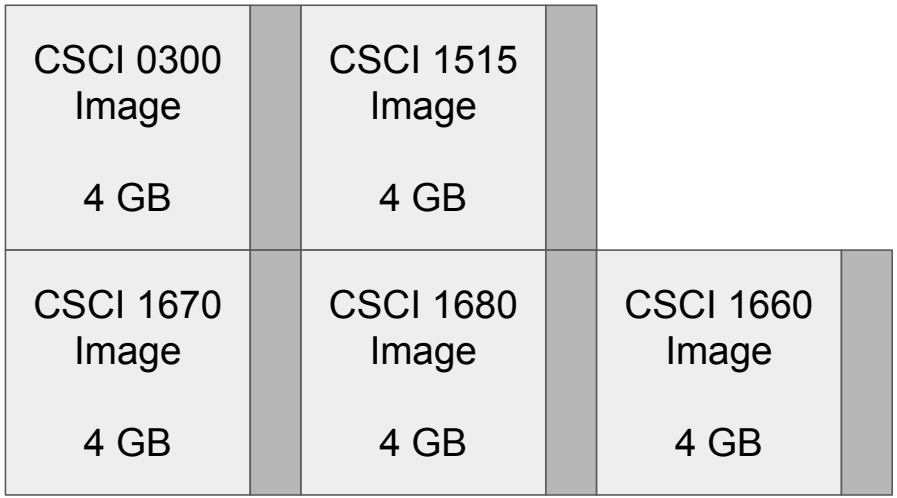


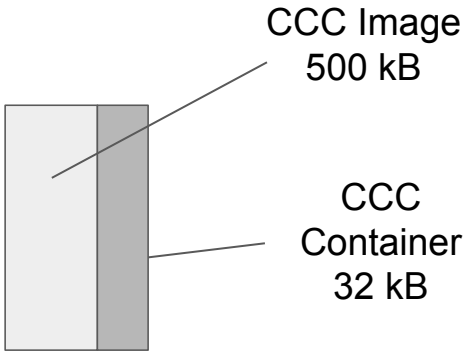
3. Containers can take a lot of **storage**



3. Containers can take a lot of **storage**

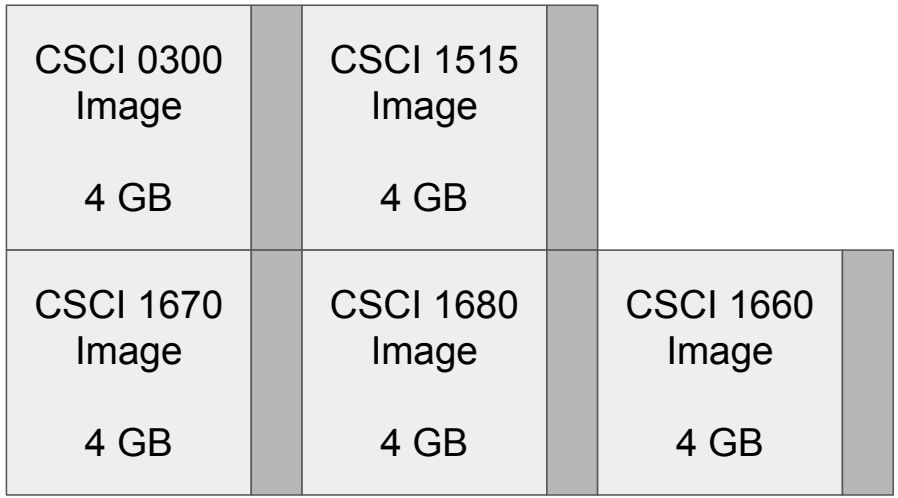
The CCC takes advantage of **shared resources**

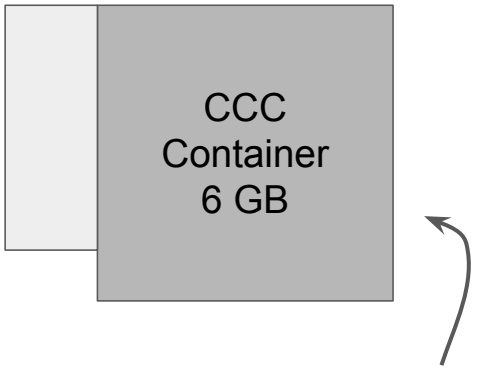




3. Containers can take a lot of **storage**

The CCC takes advantage of **shared resources**





CSCI 0300, 1670, 1680, 1515, 1270

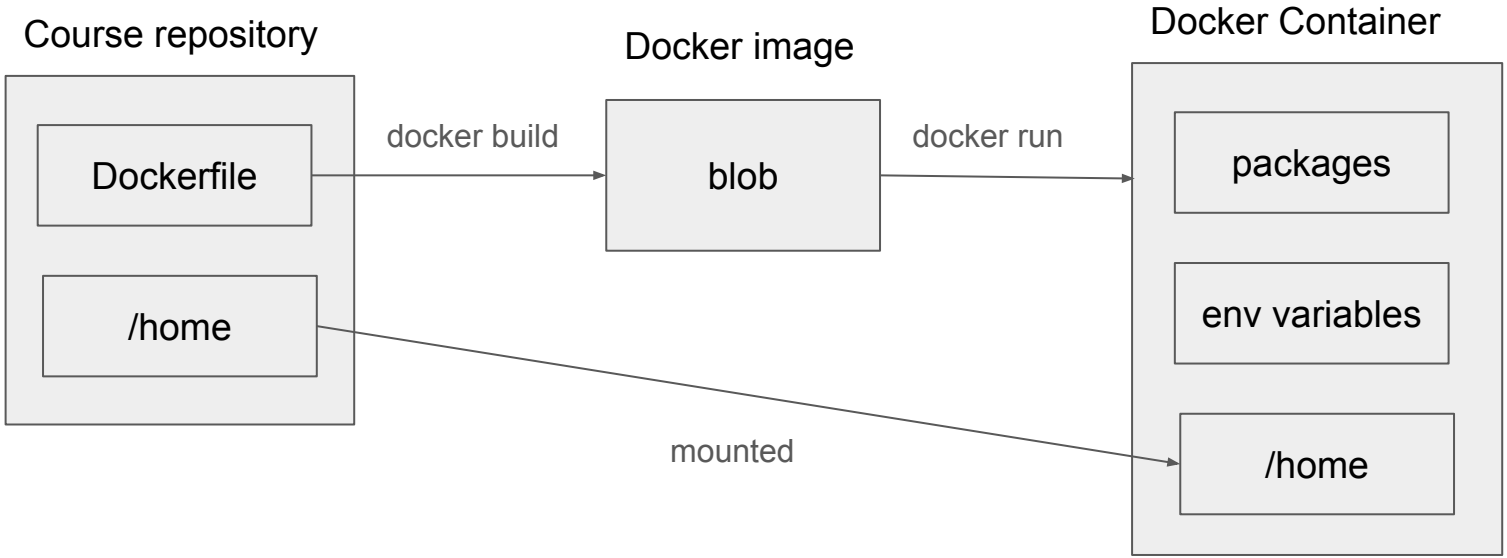
4. Students often need **additional support** with containers

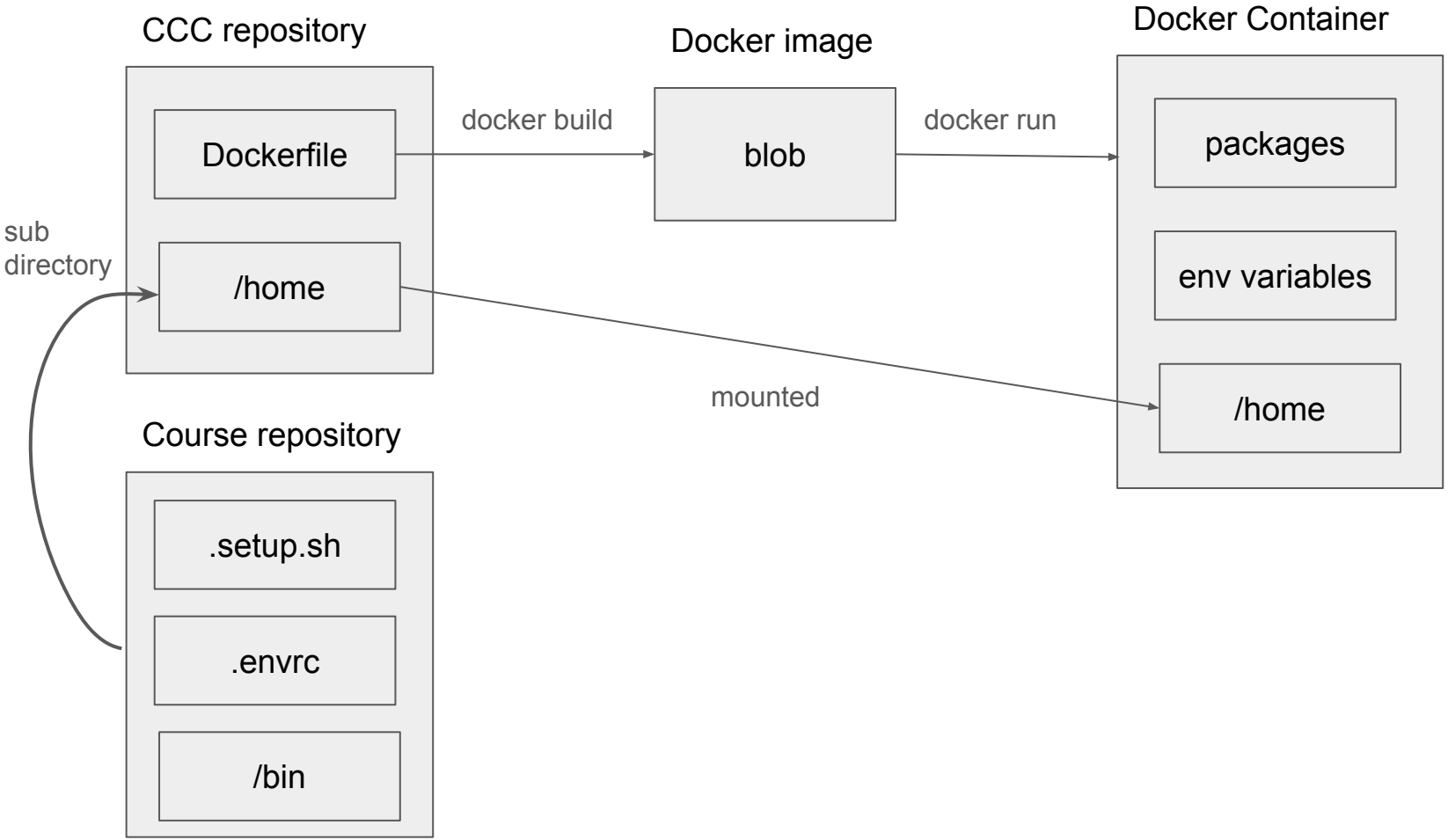
   SPOCs, Sunlab Consultants, and TStaff can help with container problems

Common Course Container

1. Students build **one image** and **one container**
2. Course staff describe packages and environment variables in their repository
3. The `ccc` handles download, setup and isolation *inside of the container*

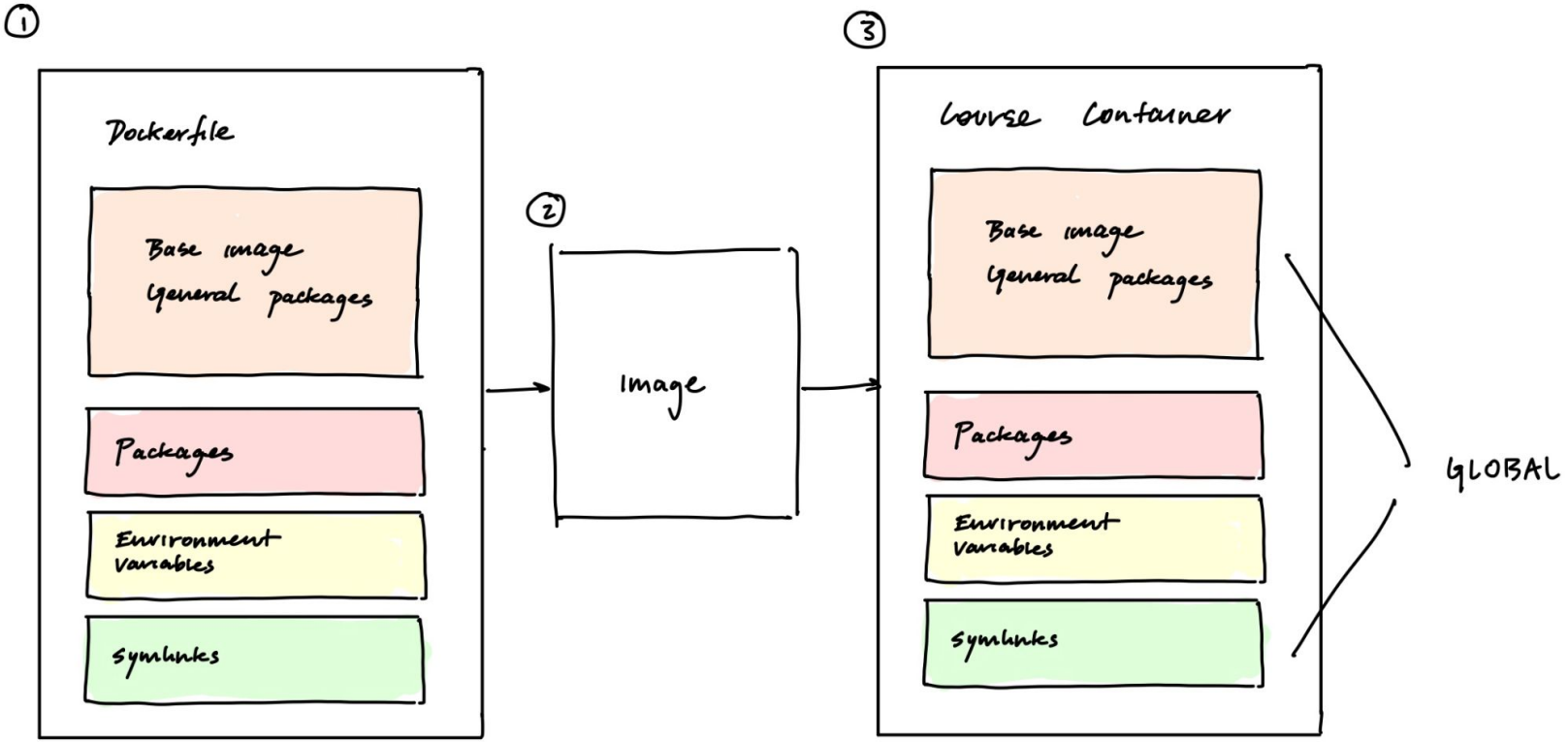
This has been tested it on CSCI 0300, 1670, 1680, and 1515 assignments

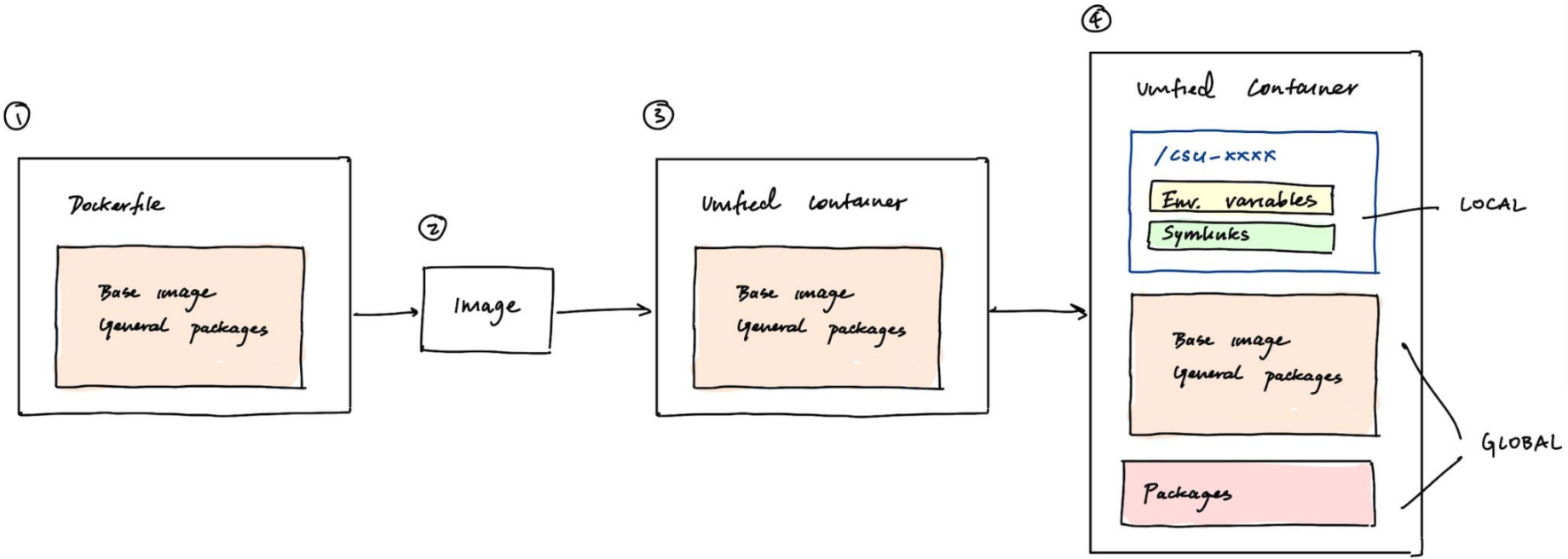

Course repository
Dockerfile
/home
docker build
Docker image
blob
docker run
Docker Container
packages
env variables
/home
mounted


CCC repository
Dockerfile
sub
directory
/home
docker build
Docker image
blob
docker run
Docker Container
packages
env variables
/home
mounted
Course repository
.setup.sh
.envrc
/bin

To port to the CCC, convert Dockerfile to bash script

1. Packages can be specified in setup.sh
2. Environment variables can be specified in .envrc
3. Symlinks can be handled with a local /bin directory


1
Dockerfile
Base image
General packages
Packages
Environment
Variables
Symlinks
2
Image
3
Course Container
Base image
General packages
Packages
Environment
Variables
Symlinks
GLOBAL


1
Dockerfile
Base image
General packages
2
Image
3
Unified container
Base image
General packages
4
Unified Container
/CSU-XXXX
Env. variables
Symlinks
LOCAL
Base image
General packages
Packages
GLOBAL