Restricted

Containerization for Research Collaboration: Platform Independent Economics

Venkat Balasubramanian, Danielle V. Handel, Anson T. Y. Ho, Kim P. Huynh, David T. Jacho-Chávez, Carson H. Rea

Abstract

We present containerization as a tool to facilitate collaboration between researchers across varying institutions and platforms. Docker aids in construction and deployment of containers, and we introduce the relevant concepts and tools for the interested researcher. Advantages of using containers in economic research include enhanced reproducibility, efficiency, portability, and ease of collaboration. Mirroring the empirical use case of Handel et al. (2021a), we use containerization in tandem with cloud computing resources to expedite a computationally intensive spatial analysis mapping access to financial services in Canada.

Keywords: Containerization; Docker; Spark, Azure, AWS.

JEL codes: A11; C87; C88.

Contents

1. Introduction

Research that supports the functions of central banks increasingly necessitates collaboration across banks, universities, and other stakeholders, requiring more advanced technology to effectively communicate. We have identified a suite of tools that can facilitate rapid, reproducible research across time zones and infrastructures. Our previous work highlights the utility of cloud platforms in research, focusing particularly on Azure Databricks and the automation of resource handling through Spark. We note that cloud computing enables researchers to easily leverage high performance computing for computationally intensive tasks (Handel et al., 2021a). However, there is an obvious practical barrier to implementing this approach: the ability to use platforms like Databricks is conditional on institutional access to cloud platform memberships. This may not be available for all collaborators, or preferred vendors may differ across institutions. In this paper, we present containerization as another collaboration tool with no barriers to access and further demonstrate its compatibility with cloud platforms.

The rest of this paper is organized as follows: Section 2 describes containers and their uses, Section 3 introduces Docker, provides an example to illustrate the construction of Docker containers, and demonstrates the portability of containers by leveraging cloud platforms. Section 4 briefly revisits the empirical example following Handel et al. (2021a) executed with containerization, and Section 5 concludes by highlighting the advantages and considerations associated with containerization for research collaboration.

2. Containers

Containers are standalone, executable packages of software that include everything needed to run an application: code, runtime, system tools, system libraries and settings. With all dependencies included, containers act to create an environment isolated from the user's host operating system, meaning that program functionality and output are not dependent on particular settings and software versions that may differ across collaborators. Containers and virtual machines (see Handel et al., 2021b) have similar functionality. However, in the case of containers, the virtualization occurs at the operating system level, as opposed to the hardware level. Consequently, containers are lighter and faster to use. As noted by Boettiger (2015), this enables researchers to run even 100's of containers on a standard laptop. Figures 1a and 1b illustrate the architecture used by containers (a) as compared with that of virtual machines (b). Note that, for containers, the applications sit on top of the shared container engine (Docker in this case), which in turn uses the host operating system.

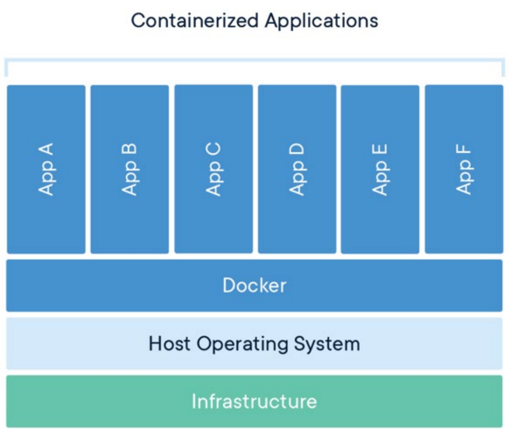


Figure 1a: Container ecosystem

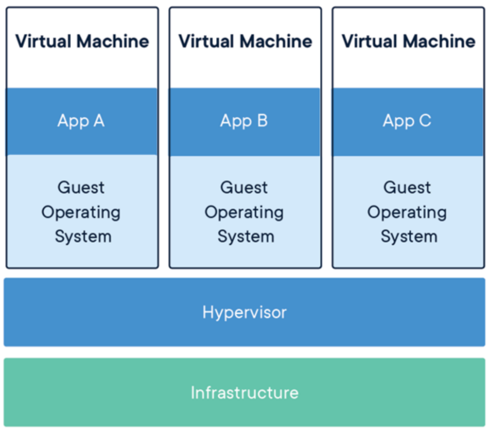


Figure 1b: Virtual machine ecosystem

3. Docker

Docker is an open-source project that streamlines the construction and use of containers. While the official Docker documentation provides a complete and robust set of directions for getting started in implementing containerization, we will briefly introduce the relevant concepts. Researchers can use platforms like Docker for the simplified management and deployment of Windows- and Linux-based container instances using the tools in Table 1.

Table 1

Docker tools

Docker CLI	A command line interface for managing container instances
Dockerfile	Text-based file with a list of instructions for image assembly
Image	Container template and metadata (how the application is **stored**)
Container	A running instance of an image (how the application is **run**)
Build	Command line action to construct an image according to a Dockerfile
Docker Hub	A cloud registry for distributing and storing Docker images

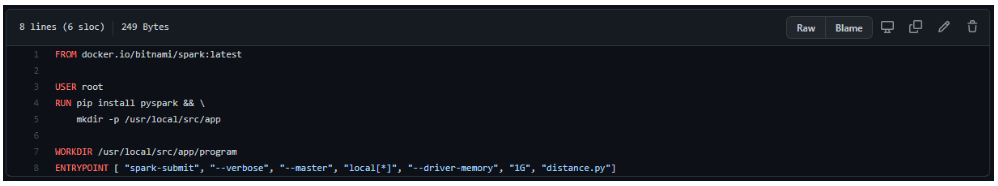
Users leverage these tools to create self-contained packages by first creating a text-based script with the instructions to make an image in a Dockerfile, which requires some basic knowledge of shell scripts. These instructions may include loading a pre-built image from Docker Hub and adding additional data and dependencies manually. Using the command line, a user then builds the image from the Dockerfile specifications using the Docker build command. The Docker up command is used to create a running instance and launch the program in a container. To facilitate exchange, researchers can push their images complete with code, input data, and any dependencies to Docker Hub, where they can then be pulled down by collaborators.

3.1 Constructing and deploying a Docker container image

The easiest way to construct a container is using prebuilt images, which can be found in and pulled from Docker Hub. The use of prebuilt images allows developers to avoid unnecessary building, expediting the creation of the final product. The use of prebuilt images also reduces deployment latency while increasing the success rate.

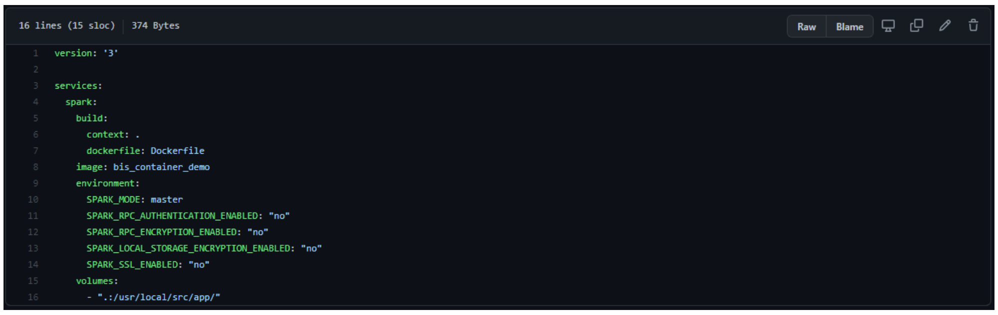
Popular prebuilt environments include Postgres, Ubuntu, and Python, each having over one billion downloads. By pulling one of these images, say Python, the created container would install everything needed to execute code written in Python, even if the local machine does not have it installed. All future containers that attempt to pull the Python image would no longer require the download, expediting the deployment process. Consequently, a container's initial deployment may take the longest compared to subsequent uses. This holds true for any of the over 8 million prebuilt images found on Docker Hub. Figure 2a displays the Dockerfile that will be utilized in the use case located in Section 4. As outlined, line 1 of the Dockerfile begins "FROM docker.io…" which indicates that the container will be pulling a pre-build image to be used, in this case Spark. Likewise, Figure 2b displays the accompanying .yml file that will be called when the user inputs "docker-compose build" or "docker-compose up."

Another option to consider when building a container is whether to include end-to-end platform capabilities. Containers can be set up in such a way so that a user can take full advantage of an IDE or Jupyter Notebook alongside the working environment. Further, a working file system can be mounted. The setup required is beyond the scope of the paper but demonstrates the extent to which containerization can supplement a workflow.



```
1  FROM docker.io/bitnami/spark:latest
2
3  USER root
4  RUN pip install pyspark && \
5      mkdir -p /usr/local/src/app
6
7  WORKDIR /usr/local/src/app/program
8  ENTRYPOINT [ "spark-submit", "--verbose", "--master", "local[*]", "--driver-memory", "1G", "distance.py"]
```

Figure 2a: Dockerfile



```
version: '3'

services:
  spark:
    build:
      context: .
      dockerfile: Dockerfile
    image: bis_container_demo
    environment:
      SPARK_MODE: master
      SPARK_RPC_AUTHENTICATION_ENABLED: "no"
      SPARK_RPC_ENCRYPTION_ENABLED: "no"
      SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED: "no"
      SPARK_SSL_ENABLED: "no"
    volumes:
      - ".:/usr/local/src/app/"
```

Figure 2b: docker-compose.yml

3.2 Container management alternatives and supplements

Singularity is an alternative platform for the construction and deployment of containers. It allows for the same functionality as Docker, specifically optimized for the deployment on high-performance computing (HPC) clusters. The Docker Registry is also accessible by Singularity, enabling the same expedited build process. Docker images can be loaded and implemented in Singularity containers without having to go through the installation process for Docker. Singularity is not compatible with Windows, limiting its functionality to Mac OS and Linux. To demonstrate the full potential of platform independent computing, our use case implements Docker.

Kubernetes supplements containerization by managing and automating services and workloads. Kubernetes is a portable, open-source framework which orchestrates containers with the aim of providing a smooth expedited workflow. Kubernetes distributes network traffic to stabilize app deployment, allows for storage to be automatically mounted, and enables self-healing by restarting, adding, or killing containers as needed. These management services, along with all others Kubernetes provides, are all aimed towards the facilitation of app deployment, so focus can be placed on the research rather than the infrastructure.

3.3 Containers on the cloud

While containers can be built and run locally, the same can be done on virtual machines, independent of platforms. Azure and AWS, products of Microsoft and Amazon respectively. Virtual machines have the same functionality regardless of platform, although Azure requires a Windows operating system. It is worth noting that while entirely possible, the construction and running of containers is not optimized for Windows. In fact, our use case makes use of both Azure and AWS to highlight equal compatibility. If using AWS, selecting Ubuntu as the processor is likely the easiest and most compatible with containerization, as Linux comes out of the box with everything required for virtualization and Git. For instructions on how to create a virtual machine, see Handel et al. (2021b), which uses AWS hosted VMs to support a virtual econometrics laboratory.

4. Empirical use case

We present an empirical use case for containerization by completing an exercise which maps consumer access to financial services in Canada as introduced by Handel et al. (2021a). We use Canadian postal codes to their closest bank branch and examine trends in access. Given consumer reliance on physical bank branches for the purchase of complex financial products (Mintel, 2018) or first-time banking interactions, this provides vital insight into the larger concern of consumer access to financial services. The Bank of Canada is also particularly interested in how consumers are connected to the supply of cash, in which physical bank branches play a large role (Chen & Strathearn, 2020).

The consumer location data we use are shape files of the 24,000 Canadian Postal Codes, which we obtain from Statistics Canada. We use the Statistics Canada Postal Code conversion files to link this data with demographic information from the 2016

Restricted

Canadian Census. Data on the locations of the 14,000 physical bank branches is obtained from the Financial Institutions File from Payments Canada.

Canadian postal codes are compact regions, often comprising a single block. So, this turns out to be a relatively computationally intensive task, with a high volume of postal codes and bank branches. We complete this task by deploying a Docker container including Python code with PySpark and an optimized SQL nearest-location finding algorithm for managing the large volume of input data, which are also hosted on the container. See Handel et al. (2021a) for a brief discussion of the empirical findings. To address the computational needs, we also deploy the container on the cloud using an Amazon AWS EC2 instance running a Linux operating system and a Microsoft Azure virtual machine running Windows 10.

Before containers can be constructed or deployed on any machine, Docker must first be installed. While the deployment of containers will be consistent between platforms, the installation process for Docker itself may differ because it must be done on the hardware level like any normal operation. As a result, the installation process for Windows may require additional steps to properly structure the backend. Installation guides for Mac, Windows, and Linux can be found on Docker's website. In this demonstrational use case, both virtual machines where completely bare, aside from the installation of Docker and Git.

The process for executing the code was identical between the machines once the machines were properly set up. The appropriate command line interface (CLI) was opened with administrative rights. The necessary GitHub repository was cloned using the "git clone" command[1]. The command "docker-compose build" initialized the Docker image. Finally, "docker-compose up" ran the image and exited upon success.

5. Conclusion and Considerations

We are not the first to suggest implementing containerization into economics research. The American Economic Association (AEA) Data Editor highlights the advantages of using containers for academic research, focusing on their implications for reproducibility and urging researchers to make use of containers when they submit data and code to the AEA's set of high impact journals[2] (AEA Data Editor, 2021). As outlined by (Boettiger, 2015), using containers eliminates the role of dependencies and software versions and imprecise documentation in creating discrepancies in program output across researchers.

Containers have ability to be pulled from a registry like Docker Hub and used on any local machine regardless of host environment, which highlights their portability. As noted earlier, containers may also be used on virtual machines on cloud computing platforms such as Azure and AWS EC2, which can aid in the management of computational resources for especially intensive tasks. Also contributing to their portability is their size, which is an order of magnitude smaller than virtual machines. As discussed earlier, containers require significantly less space and time to operate and start up than virtual machines, motivating their use for collaboration among

[1] We direct readers to TK to find all inputs and scripts necessary for replicating this exercise on their own machine. Once Docker is installed, our pre-built container can be deployed using only 2 commands.

[2] See the AEA Data Editor GitHub page for more detailed instructions on using Docker for replication

researchers with varying levels of resource constraints. These advantages in reproducibility, portability, and efficiency position containerization as a powerful tool for research collaboration.

References

AEA Data Editor. (2021, November 21). *Use of Docker for Reproducibility in Economics*. Office of the AEA Data Editor. https://aeadataeditor.github.io/posts/2021-11-16-docker

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1), 71–79. https://doi.org/10.1145/2723872.2723882

Chen, H., & Strathearn, M. (2020, February 6). *A Spatial Model of Bank Branches in Canada (No. 2020–4).* Bank of Canada. https://doi.org/10.34989/swp-2020-4

Handel, D., Ho, A., Huynh, K., Jacho-Chavez, D., & Rea, C. (2021, October). Cloud Computing Research Collaboration: An Application to Access to Financial Services. *Data Science in Central Banking: Machine Learning Applications*. IFC and Bank of Italy Workshop on "Data Science in Central Banking," virtual event hosted by the Bank of Italy.

- . (2021). Econometrics Pedagogy and Cloud Computing: Training the Next Generation of Economists and Data Scientists. *Journal of Econometric Methods*, 10(1), 89–102. https://doi.org/10.1515/jem-2020-0012

Mintel, "The Branch Banking Experience - Canada - February 2018," Technical Report, Mintel February 2018.