EDSB: Data Science Workflows

Jim Harner, West Virginia University

Oct. 7, 2020

Important links

NISS's Essential Data Science for Business:
https://www.niss.org/events/essential-data-science-business-data-science-workflows

The tutorial content:
RSpark Tutorial: https://github.com/jharner/rspark-tutorial

The RSpark development environment:
RSpark: https://github.com/jharner/rspark

A user environment pulled from Docker Hub:
RSpark Docker: https://github.com/jharner/rsparkdocker

A user environment pulled from Vagrant Cloud:
RSpark Vagrant: https://github.com/jharner/rspark-vagrant

R, Shiny, etc. environments:
Rocker Project: https://www.rocker-project.org

Jupyter environments: Jupyter Docker Stacks:
https://github.com/jupyter/docker-stacks

Tutorial Outline

The tutorial content is based on RSpark Tutorial

- Session 1:
  - Module 1 Fundamentals: s1 workflows; s5 data structure; s6 functions; s7 intro machine learning (ml)
  - Module 2 Data sources: s1 text; s5 databases s7 excel, dates
- Session 2:
  - Module 3 Transformations: s1 data manipulation; s2 dplyr backends; s3 data cleaning
  - Module 4 Hadoop: s1 hadoop; s3: hdfs
  - Module 5 Spark: s1 basics; s2 spark dplyr; s3 spark sql
- session 3:
  - Module 6 Supervised Learning: s1 regression; s2 regularization; s3 slump example; s7 tree models
  - Module 7:Unsupervised Learning: s2 k-means

DevOps for Data Science

Data Science Platforms are complex! How do we solve the data analysis issues of:

- collaboration,
- sharing,
- reliability
- scalability, and
- reproducibility.

DevOps (development + operations) has been revolutionized by containers and methods of orchestrating containers. Specifically, two technologies are changing how complex software systems are built, deployed, and maintained. These are:

- Docker: a technology for building, deploying, and running container images;
- Kubernetes: a technology for deploying, scaling, and managing containerized applications.

Reproducibility

The following variants of reproducibility assume the same underlying hypotheses:

- **Repeatable**: same data, code, and user $\rightsquigarrow$ same results
  Ex.: Repeatable over long time periods
- **Reproducible**: same data and code, but different users $\rightsquigarrow$ same results
  Ex.: Referee for a journal paper
- **Strongly Reproducible**: same data, but different code and users $\rightsquigarrow$ comparable results
  Ex.: Competitions and multiple research teams
- **Replicable**: different data, different code, different user $\rightsquigarrow$ comparable results
  Ex.: Meta analyses
- **Generalizable**: different data, same code, different user $\rightsquigarrow$ comparable results
  Ex.: Productionizing a machine learning algorithm

These definitions (except for the 3rd) are from NSF.

Basic Tools for Reproducibility

Reproducible computing and reporting require the following underlying skills:

- the Linux CLI
- basic bash scripting
- make
- Docker/ Docker Hub
- git/ GitHub
- R/ Python

The following skills are required for certain use cases:

- R package development
- ansible
- Pandoc
- TeX/ LaTeX

Required Environments for Reproducibility

- computing infrastructure
  - Docker for creating images/ containers based on Linux
  - Docker Hub for storing/ deploying pre-built images
  - bash scripts for automating the docker build process
  - version tags to identify the pulled images
- coding/ reporting projects
  - git for version control, tracking changes, branching, etc.
  - GitHub for hosting software projects and for providing version control and collaboration features
  - make for automating the code/ report build process
  - version tags to identify the project state

The version tag of the Docker Hub images should match the version tag of the GitHub project at any given step in the development process for the images and the project repo, respectively.

Creating a Reproducible Workflow

- Build the computing environment first, at least approximately.
  - download a base image from Docker Hub—perhaps just a Linux image, a Rocker image, etc.
  - write a Dockerfile extending the base image, e.g., adding R packages
  - add configuration files as needed
  - write shell scripts to automate building/running containers and pushing images to Docker Hub

Your container (a running instance of an image) must have a development environment for running code using literate programming, e.g., R Markdown.

- Build the coding environment.
  - start a project repository on GitHub, initially with a `README` file
  - clone the GitHub repo to the container running the editing environment
  - use R Markdown for literate programming
  - use `make` to automate the project build process
  - stage, commit, and push to GitHub often

What is Docker?

Docker allows developers, devops, and sysadmins to develop, deploy, and run applications using containers. We call this containerization.

Containers are:

- Flexible
- Lightweight
- Portable
- Scalable

A container runs natively on Linux and shares the kernel of the host machine with other containers. A container runs as a discrete process and thus its memory requirements are nearly equivalent to other executables, i.e., it is lightweight. On the other hand, a virtual machine runs a guest OS which is built on a hypervisor, through which host resources are accessed. Running multiple VMs is very heavy.

Docker Applications

Docker containers run a single service, although there are workarounds. What if you need multiple services, e.g., RStudio and PostgreSQL?

Answer: build a Docker application with `docker-compose`, which builds multiple images (`docker-compose build`) and starts the corresponding containers (`docker-compose up`).

`rspark` is a Docker application with four containers:

- `rstudio` with an RStudio service;
- `postgres` with a PostgreSQL service;
- `hadoop` with a HDFS service;
- `hive` with a Hive data warehouse service

`rstudio` is built on the Rocker Project, a widely-used suite of Docker images with customized R environments for particular tasks. In particular, we pull Docker Hub images from `rocker/verse:3.6.3-ubuntu18.04`.

By default, the containers within a Docker application are on the

Docker Container Architecture

- Images:
  - A set of instructions in a Dockerfile detailing the construction of the container, including all files, dependencies, and code necessary for execution.
  - Instructions in the Dockerfile create the layers of the resulting image. If the build process proceeds to completion, the resulting image can be run utilizing any executable files on its filesystem using `docker exec`, but there is a specified entry point.
- Containers:
  - A running instance of an image.
  - Additional Docker objects specified, such as data volumes, networks, and plugins.

Docker Application Architecture

A Docker application is created by a `docker-compose` file, which declares services, volumes, and networks:

- Services:
  - A container coupled with execution commands and specifications.
  - Ports exposed to services outside the network.
  - A network or collection of networks connecting services within the application environment.
  - Specifications for container behavior and hardware limits.
- Volumes:
  - Bind Mounts: Files or directories on the host machine mounted to one or more Docker containers.
  - Docker Volumes: Filesystem bind mounts that are managed by Docker.

Both volumes and bind mounts allow you to share files between the host machine and container so that you can persist data even after the container is stopped.

Future Work

The `rspark` project is being expanded to better facilitate big data and big compute:

- Arrow

Apache Arrow is a software framework for statistical and machine learning applications that process columnar data. It has a column-oriented memory format that is able to represent flat and hierarchical data for efficient operations on modern CPU and GPU hardware. The Arrow memory format supports zero-copy reads for lightning-fast data access without serialization overhead.

See the talk on Thursday at 11:25 am on Data Science Technology.

- Kubernetes (k8s)

Kubernetes is a platform for running and coordinating related, containerized applications across a cluster of (typically virtual) machines. It manages the complete lifecycles of containerized applications and services

Arrow in R and Spark

The R package `arrow` provides a `dplyr` interface to Arrow Datasets. You can use `open_dataset` to read a directory of data files and `dplyr` for querying.

Spark applications using `arrow` with R (similar with pandas):

- Copying data from R to Spark with `copy_to`

Using `arrow` with `sparklyr` does not require serialization in R or persisting data to disk. The speedup is substantial using `arrow`.

- Collecting data from Spark to R with `collect`

Nice speedup, but not as substantial as above since `sparklyr` already collects data in columnar format.

- Custom transformations using R functions in Spark

`spark_apply` with an R function converts the row format of Spark DataFrames to columnar format in parallel. No serialization or deserialization needed. The speedup is substantial.

A Kubernetes Example using Jupyter

A Container Platform provides a complete solution, e.g., the components needed for teaching a data science program. Increasingly these platforms are being built on Kubernetes.

Example: JupyterHub

JupyterHub allows instances of a single-user Jupyter notebook to be spawned and managed. Two distributions are available:

- The Littlest JupyterHub for 0–100 users on a single machine
- Zero to JupyterHub on Kubernetes for a large number of users and machines

Example: RStudio Server Pro with Launcher and Kubernetes

RStudio Server is not supported, i.e., Launcher is a Pro product.