RStudio® *Open source & enterprise-ready professional software for R*

RStudio and Docker

"Docker" invokes many different ideas of how infrastructure can be built in the future. Why are people so excited?

Many organizations are turning towards Docker as a solution for reproducibility and scalability. At RStudio, we use Docker every day in production systems for many good reasons. However, we also deploy traditional multi-tenant servers and do data science on the least hyped platform of all, our desktops. In this document, we will share what we've learned by looking at the pros and cons of Docker, where Docker fits with R, and how RStudio is making Docker easier to use with our products.

Understanding the Docker Phenomenon

Isolation

At the foundation of Docker are clever interactions with the Linux kernel based on the idea of sandboxes. These sandboxes ensure an application in a Docker container is isolated from the underlying server and from other applications.

This isolation has nice benefits. For example, two users can run R code on the same server, each in their own container, and not worry about user A's installation of libxml2 breaking user B's analysis. (libxml2 is a system library used by the popular xml2 package).

However, this isolation comes at a cost. Imagine starting each analysis with a brand new computer - you'd spend quite a bit of time installing packages! Luckily Docker supports caching. Organizations can create an "R container" that comes with their favorite R packages pre-installed. However, caching is one of the two hardest problems in Computer Science. Docker is not immune; it is hard to find the right balance between caching and customization.

Even with an optimized container, launching Docker takes more time than starting a new R session. For data scientists, the time between starting a project and writing the first line of code is an important cost. Often dedicated analytic servers outperform containerized deployments by allowing users to create projects with little overhead. For the best of both worlds, it is possible to run RStudio Server Pro inside Docker. This model allows a single container to support multiple users and sessions, requiring the overhead of containerization to be paid just once per project, not once per data scientist's session.

The challenge of balancing caching with isolation is also why we've opted to use a different isolation mechanism in RStudio Connect instead of Docker. Connect uses Linux kernel tools (cgroups) similar to Docker to isolate parts of a user's application from the operating system, and uses packrat to manage R dependencies. The result is an application sandbox, or container, similar to Docker but optimized for data science products like Shiny applications and R Markdown reports.

Reproducibility

Isolating applications from the underlying server facilitates reproducibility, but doesn't guarantee it. In Docker, all the dependencies for an application are captured in the container. These dependencies can be enumerated in code in the form of Dockerfiles. A Dockerfile contains a series of instructions, e.g., install R, install some R packages, etc. When the instructions are executed, you end up with a Docker container. Much like writing down a recipe, the benefit of a Dockerfile is that someone else can follow the instructions and end up with a similar environment.

Dockerfiles do not ensure reproducibility. A Dockerfile contains enough information to create an environment, but not enough information to reproduce an environment. Consider a Dockerfile that contains the command "install.packages('dplyr')". Following this instruction in August 2017 and again in December 2017 will result in two different Docker containers, since the current version changed.

Users often modify their environment after a container has been started. A Dockerfile that includes the instruction "install R" will build a container with R, but then the user will manually install packages inside the container. These manual actions are not captured by the Dockerfile. This scenario doesn't represent a bad setup. Most R users are not expected to write Dockerfiles. A seasoned DevOps team will work to supply R users with a container that serves as a reasonable launching place for further customization.

Since a Dockerfile helps with reproducibility but does not guarantee it, many organizations rely on taking snapshots of the actual containers themselves. These snapshots can be stored, organized, and brought back to life at a later date.

Provisioning containers from Dockerfiles and managing snapshots requires a significant amount of tooling. This tooling can be bought from platform providers, or home-built by a mature DevOps organization, but the engineering investment should not be underestimated. Proper tooling for an analytics workflow centered on Docker will take an order of magnitude more work than supporting a traditional, dedicated, and multi-tenant analytics server.

Organizations interested in getting started with Docker can begin by treating Docker containers as stand-in replacements for virtual machines. For example, it is just as easy to install RStudio Server Pro, Shiny Server Pro, or RStudio Connect inside a Docker container as inside a VM. Teams can start by running a single, long-lasting container and then gradually build tools to support a model that uses containers to isolate and reproduce specific analysis.

Portability and Scalability

Docker containers are also known for portability and scalability.

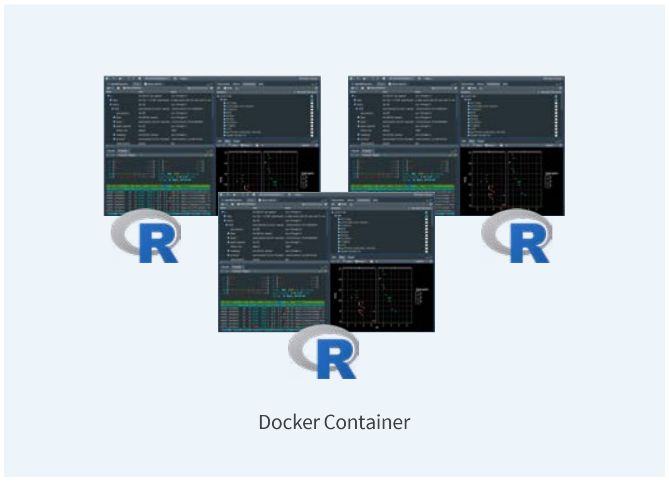
Like virtual machines, Docker containers are easily portable because they isolate the runtime environment of an application from the underlying server. The same Docker container can be run on a physical server or in the cloud. This flexibility saves time for IT organizations tasked with maintaining many applications and servers.

Containers are also easy to clone, theoretically allowing organizations to scale applications that are built on top of containers. In practice, however, scaling also requires careful tooling. For example, R users typically require persistent storage that can be accessed during an analysis. Creating tooling that can quickly scale 100s of R containers while seamlessly giving them equal access to shared storage is no small task.

Docker and RStudio … err I mean R?

Where does rubber meet the road with R and Docker?

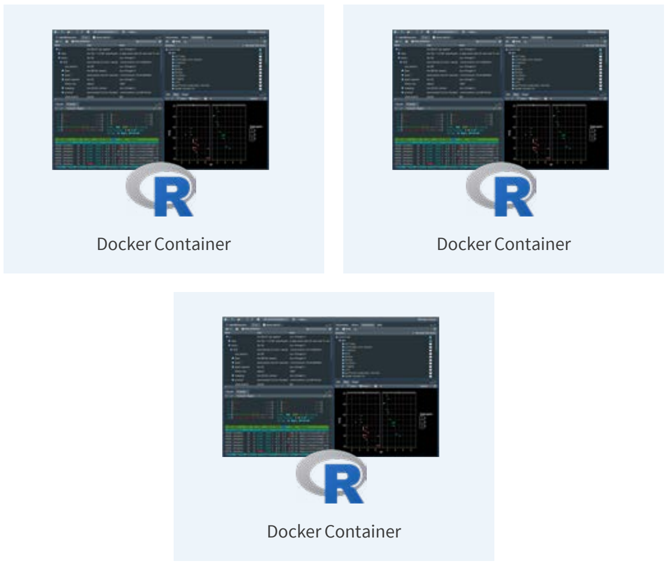
Stage 1: Treat Docker like a VM



Most teams using R with Docker will begin small and grow. To start, many organizations treat a Docker container like a VM. The container can include either the open-source RStudio Server or RStudio Server Pro, and is accessed by many users across different projects. The container is still portable, but will be left running for an extended period of time.

A similar approach works for Shiny applications. RStudio Connect or Shiny Server Pro can be installed into a container or a cluster of containers for high availability. Connect or Shiny Server Pro handles scaling the number of R processes, and Connect additionally provides app isolation and reproduces package libraries.

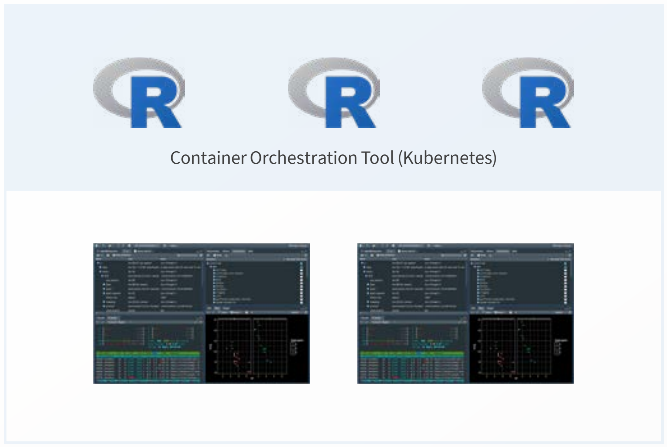
Stage 2: Tooling to support multiple containers



As the organization gains experience, it can build out tools to support multiple containers. The required interface allows an R user to provision a container and access previously saved containers. DevOps teams can work with R users to design reasonable baseline images. Infrastructure can be put in place to allow containers to mount shared resources and access data.

For Shiny applications, this approach translates into a container that includes Shiny Server Pro and a single app. Alternatively, RStudio Connect can be used to achieve isolation, reproducibility, and scalability with or without Docker. However, we do not recommend using one RStudio Connect instance per application.1

Stage 3: Separating R from the RStudio IDE or RStudio Connect



Today, we're proud that the terms "R" and "RStudio" are often used interchangeably. However, as organizations mature further into their use of Docker, there comes a point where conflating the two is problematic. Looking ahead, RStudio will be working in 2018 to allow a third architecture, which separates the R runtime from the RStudio service. This model will help DevOps organizations focus on scaling and managing containers, while allowing users to access backend compute through the familiar RStudio interface.

Connect will also support the ability to launch R processes in separate containers. In this model, Connect is a front-end service responsible for running R in support of Shiny apps, R Markdown documents, and plumber APIs, but Connect is agnostic to where R actually runs. The Connect service could be backed by local R processes or a Kubernetes, Mesos, or Docker Swarm cluster.

Conclusion

Accounting for the benefits and costs, Docker is here to stay and will become a bigger part of computing infrastructure.

How should you use Docker with R? Unfortunately there is not a single answer. At RStudio, we're committed to building products that work inside and outside Docker, in a variety of ways. We're investing in tools today to support your team's future Docker adoption.

If you would like to learn more, or are interested in having a conversation with us on how to think about your deployment model, please reach out to us at sales@rstudio.com.

1. For deployments that increasingly isolate RStudio Server Pro and Shiny Server Pro in containers for specific analyses and move away from multi-tenant servers, our server-dependent published pricing is not a good fit. Please contact our customer success team for server-independent pricing.

About RStudio

RStudio® makes data analysis with R easier and provides powerful tools for sharing reports, dashboards, and interactive Shiny® applications with your entire enterprise.