Docker : Learning Resources

This is a collection of resources that have been helpful to me in learning Docker. I would not be anywhere without them and wish to thank every single contributor to these resources.

In no particular order of preference:

* [Official Docker Documentation](https://docs.docker.com/)
* [Docker Try It](https://www.docker.com/tryit/)
* [Free Docker Training](https://training.docker.com/) from Docker
* [Awesome Docker](https://github.com/veggiemonk/awesome-docker) : Great list to start with.
* [Docker Cheat Sheet](https://github.com/wsargent/docker-cheat-sheet)
* [Docker How To’s](https://github.com/botchagalupe/DockerDo)
* [Container Tutorials](http://containertutorials.com/) by Rajdeep Dua
* If you want to learn more about Docker files, what has helped me is to simply navigate over to the [Docker Hub](https://hub.docker.com/) and browse through a few popular repositories. Check out their Dockerfile and see what is going on inside.
* [Official Docker Blog](http://blog.docker.com/)
* Sign up for the [Weekly Docker news](https://www.docker.com/newsletter-subscription). Great way to get served the best content around Docker.

# Docker Use Cases

[Go to the profile of Romin Irani](https://rominirani.com/@iromin?source=post_header_lockup)

[Romin Irani](https://rominirani.com/@iromin)Follow

Apr 9, 2015

[**Docker**](https://www.docker.com/)is all the rage of late and it is not easy to comprehend the different areas that you could use it for. It took me a while to understand how it can be applied to my everyday role as a developer and to be frank, I am still learning.

This blog post is a list of use cases that I have found myself applying Docker to and my hope is that it will help you understand it better and trigger your thought processes into using Docker appropriately. Keep in mind that this is not a solution in search of a problem.

Before we get into the use cases, keep repeating this statement in your head a few times : “Docker is a **shipping container system** for applications”. The words in bold are intentional. It is important to understand to some extent not just what shipping container are but the efficiency that they brought to transportation of **any goods**.

When it comes to your application (remember it can **any**application), the shipping container system **abstracts and provides a standard container** in which to run them.

Can I run a Database Server? Yes, you can.  
Can I run my Web application written in Node.js? Yes you can.  
Can I just provide an API stub server while I am busy still writing the details? Yes you can.

**It’s not about your app**. It’s not about what is inside your app to a large extent. Its about **packaging, shipping and delivering your app** in a standard way.

These are use cases that I use regularly and I would like to hear yours too. So here we go (in no order of importance):

### Trying out new software

As a developer, you are always trying out some software or the other. That’s what we live for. It’s not always a pleasant experience to set things up after downloading the software. Time is of essence and sometimes all we are looking for is to fire a few commands and that’s it. The Docker model is a super simplistic way of running software, which behind the scenes takes care of getting the image and running it for you.

It’s not just about new software. As an example, consider that you want to spin up a Database Server (MySQL) quickly on your laptop. Or setup a Redis Server for your team. Docker makes this dead simple. Want to run MySQL Server? All you need to do is : docker run -d -p 3306:3306 tutum/mysql

You could save hours of your time.

### Great for Demos

I often find myself giving a demo or two on the weekends to some group or the other. The software stack for these demos varies big time. I am increasingly finding that Docker images as an ideal way to package and demo these applications. This way I stick to a consistent method to package and demo my software. And it is also a great way for the participants to maybe tweak things a bit and then package the images for others to use.

### Avoiding “It works on my machine” syndrome

If you have been developing software, you know about it. Actually I should have put it in another way i.e. we all have said this to our Testing Team or fellow developers from time to time. But that’s not the point. The point is that can we have a much more reliable, repeatable and standard process so that the experience of just setting up and running our software is straight forward. Docker gives you that container format and runtime in which to make this happy. Give your testing team an image and straightforward Docker commands to run your application. Relieve them of complex setup instructions of going into some file and tweaking a property or two.

Another area where I find this useful is in a training class. If your intent is to demonstrate an application or two, avoid software setup nightmares on all your participant machines by going the Docker way. You will save hours. Granted that this could be done via a VM but Docker makes it simple, lightweight and hey … you can tell the training participants that they are also going to learn about the hottest topic out there in the software world. A win-win for everyone!

### Learning a bit of Linux/Scripting

This might look like a odd reason but this represents a great opportunity for folks not familiar with Linux OS and its scripting languages to get another shot to picking it up. Come to think of it, Linux is important and I am not going to get into the specifics here. If you come from a Windows background, take my advice and get yourself a Linux VM with any of the Cloud Providers. My choice of OS is CoreOS there. But this will really force you to pick up some Linux basics, feel comfortable with the command line and over time begin to appreciate the OS.

### Better use of resources

Compared to VMs, I have found Docker containers lightweight to a large extent. Just the granularity of it is a win for me. I often use it to run several containers on my laptop to demonstrate the simplicity, granularity and footprint.

### Made for Microservices

You are sure to have heard about Microservices, if you have been following any tech news of late. Docker and Microservices play well together. Conceptually thinking, a Microservice would be a logical piece of your application that can run independently and once you have figured that out, Docker plays well here to help you out in packaging this not just as an image but making it easier for your development team , testing team and possibly even your deployment team to just take the image and deploy your microservices based app.

### Porting across Cloud Providers

Most Cloud Providers of note have voiced their full support for Docker. What does this mean for developers. A likelihood that you will be able to move your workloads across different Cloud providers easily. It also makes your job of moving your application into the cloud easily. Why have one way of setting up the software locally and then some other way of setting up / running your software in the cloud. Docker here and Docker there. It should help simplify the last leg in your workflow i.e. deploying much easier and standardized.

### API Endpoints

APIs are the glue between apps and you definitely have either used a REST API or better still even implemented a REST API. The point here is that before we start working on the implementation of the REST API, you would like to define the API contract and publish that document, so that the client side of the equation can start coding against that interface. To take this to the next step, typically the Server side folks implement a dummy API stub layer so that fake or sample responses are returned from the server.

While some of you might end up saying that Docker is an overkill here and that you could do with sample.json files, think about a Docker container that stubs out your API layer and the client team can access to that.

To explain my point better, I will point you to [JSON Server](https://registry.hub.docker.com/u/clue/json-server/), a Docker image that provides REST API mocking based on plain JSON. You get the picture, right?

### Ripe for Tools/Innovation

I don’t think this is a use case but I thought I will just push this in here. Docker is evolving fast. Tools are coming in fast. Have we seen the last of them? Definitely No. There is all the likelihood that the more you use Docker, the more you apply it to your medium to complex applications, it is likely that you will chance about a particular void in the ecosystem. Since it is early days in here, any tools that you build around this are likely to get wide traction in the community. They say it is important to be there early in the game. The current timeline in this whole Containerization World is exactly that. Great opportunities for everyone.

### Your Use Case

This is not my use case but your own. I would love to learn what other use cases you have applied Docker too. Do share those in the comments.

### Some other points

I would like to cover 2 other points that have greatly helped me in my ongoing journey of learning Docker.

First up is the [Docker Hub Registry](https://registry.hub.docker.com/). This is a public repository of Docker images that you can use today. You should check out the official Docker images that have been made available but more than that, look at the top community contributors and the images that they are publishing. You are sure to find some gems in there that could save you and your teams hours.

If you feel that there is particular image that you need but are not sure if someone has created one, hop on to the Docker Hub Registry first. As the saying goes that if you think you have an idea (a new Docker image), there is a great likelihood that someone has already implemented it (provided the Docker image).

The other one is to look at industry news around the IaaS vendors and see how they have been embracing Docker and the ecosystem that has been springing up around it. While I am not a privy to what might be the actual things going on inside their meeting rooms, it is a safe bet that containerization technology is what they have been running all along and it would be a safe bet to understand this technology as a way to delivering your applications not just locally, not just to your cloud vendor of choice but across cloud vendors in a portable way.

Things are moving towards that … and that can only be a good thing.

# Learning Docker ? Move To The Cloud !

[Go to the profile of Romin Irani](https://rominirani.com/@iromin?source=post_header_lockup)

[Romin Irani](https://rominirani.com/@iromin)Follow

Mar 16, 2015



*December 2017: This post is definitely dated by now , not in what it is trying to tell you but the tools that it suggests. It still makes perfect sense to use to the cloud to learn docker. The tools are now different. For e.g. Google Cloud Platform now provides a great option called Google Cloud Shell, which is a curated Admin machine available to you in the cloud that you can access and it has all developer tools installed for you (including Docker). Try it out here:*[*https://cloud.google.com/shell/*](https://cloud.google.com/shell/)

[Docker](https://www.docker.com/) is all the rage and there are valid reasons for it. Every software developer needs to understand this fundamental shift that is happening when it comes to packaging, delivering and running applications as containers. It would definitely pay to start learning this technology. There are enough resources on the Net to do that today and hopefully I will come around with some tutorials that can help demonstrate how useful Docker can be to individual developers like you and me.

The focus of this blog post is different though. It is about which platform/OS should I start learning Docker on. And then I would like to make some points from the perspective of countries/individuals where it is not easy to get high speed internet access all the time for several reasons. My focus is going to be for students who want to get started and have limited resources.

### Which platform : Linux, Mac or Windows?

So first up is which platform ? It is clear from the current state of Docker that it runs natively only on Linux and not on Mac or Windows. Agreed that there are solutions like boot2docker available that make it possible to run on Mac or Windows. But let me throw out Mac from the equation altogether. Nothing against the OS as such but in my experience from where I stay, the developers that I talk to and current circumstances, I believe 1 in 1000 people have a Mac as compared to a Linux or Windows laptop. The simple reason is that of economics.

### boot2docker is good! But keep it on hold for a moment!

Despite that I suggest that you do not do anything locally even via boot2docker. The chief reason (and I won’t go into every detail) is that when building up initial docker images, you have to start with some version of the OS. That is perfectly fine but the problem that comes up is that this initial download of the base OS image that you begin to work with is likely to run into 100’s of MB that need to be downloaded first. This is both time consuming (on slow Internet access) and possibly expensive too for you to upgrade to more expensive plans. As a student of the subject, you want to see results quickly. The First Time To Hello World (FTTHW) has to be as minimal as possible to create the right impact for you. Given that, take my advice that in the initial days of your learning — keep away from this loop.

### Embrace the Cloud

So what does that leave you with? It leaves you with the IaaS providers like AWS, Google Cloud Platform (GCP), Azure and others. Yes, it is going to cost you a bit but believe me it is a very minimal expense to rent out a Linux Box on the high quality, top class networks that your instance is hosted with and blazing fast network access to Docker images.

This is what I have been doing for the past several weeks and have been recommending to all folks that I come across:

1. Go ahead and get yourself an account on [Google Cloud Platform](https://cloud.google.com/). It is easy to get started and you even get a $300 credit for a trial period of 2 months. Trust me $300 is a lot of money for IaaS services , especially if you are looking to learn about the platform and its services.
2. Provision a [Google Compute Engine](https://cloud.google.com/compute/) (GCE) instance. Select one of the [Linux images available](https://cloud.google.com/compute/docs/operating-systems/) and you are all set to install docker.io and move forward with it.
3. As an alternative to **Step 2**, you can directly create a GCE instance of a Container VM image itself so that it is all setup with Docker. Check out the [documentation](https://docs.docker.com/installation/google/) on how to do that.

### Parting thoughts

I believe the above steps will keep the productivity high and keep your costs to a minimum. Once you are done with the instance, you can simply delete it. Once you have your head wrapped around Docker, you will be better prepared to then go ahead and do stuff that you want on your local machine.

Some folks mention that the [online Docker tutorial](https://www.docker.com/tryit/) is good enough. I agree to some extent but there is nothing compared to setting up your own box and playing with Docker on your own. This way you can experiment with other Docker images, build a few yourself and go through the mechanics of Docker present at the [official documentation site](https://docs.docker.com/userguide/). Do try out the [examples](https://docs.docker.com/examples/), especially the ones that are relevant to the software stack that you are working on.

I would love to hear your thoughts on this. It is possible that I might have completely missed out something.

Note: I have taken the example of Google Cloud Platform that I primarily use. Linux Instances are available on any other cloud provider too, so pick your choice.