Discussing Docker. Pros and Cons.

blog.philipphauer.de/discussing-docker-pros-and-cons/

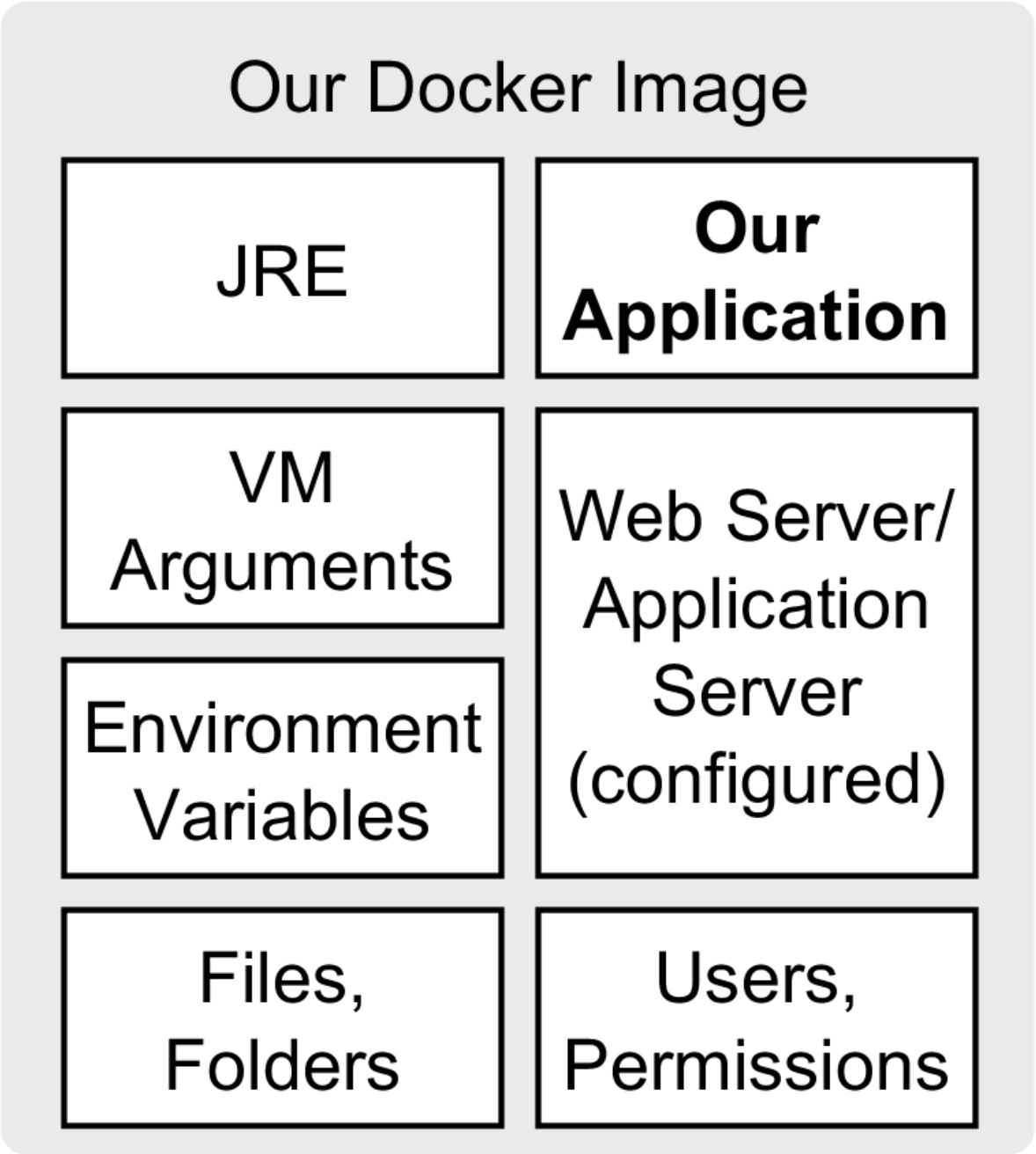
October 25, 2015

Docker allows us to easily create reproducible environments for our application. We automate the setup of the environment and eliminate manual error-prone tasks. This way we reduce the risks and the reliability of the deployment process. But there are also challenges and domains, where the usage of Docker can be difficult. This post discusses several advantages of Docker and points out some drawbacks.

Advantages

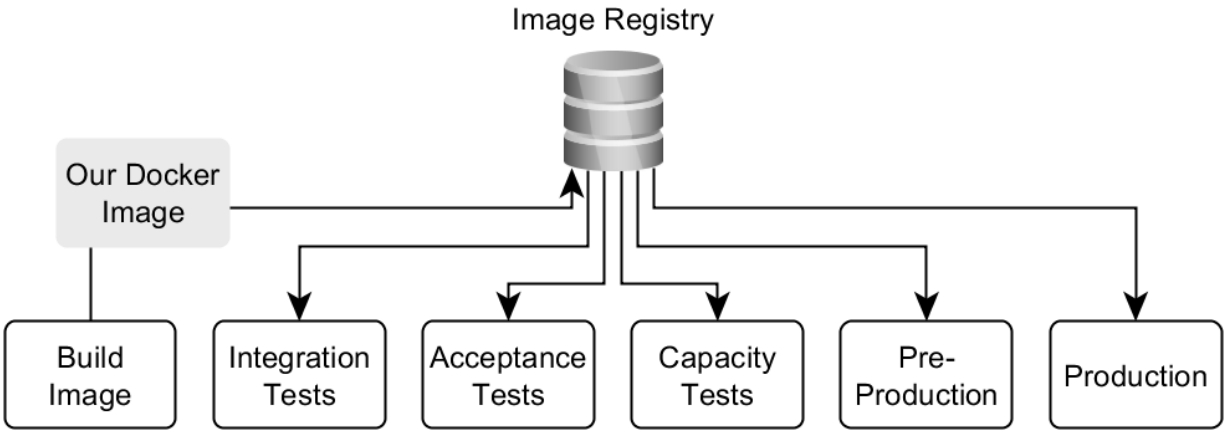
Advantages for the Development Team

- Full control over the execution environment of the application. This includes the necessary infrastructure like the JRE, the application server, VM arguments and other environment variables the application needs to run. The infrastructure can be changed easily and independently by the development team, because they don’t have to wait for the operations team to change the environment.



A Docker image contains everything our application needs to run (except external resources like the database).

- Reduced Risk. The tests run against the same image that will finally run on the production server. This increases the reliability of the tests. Moreover, releases become less scary, because we just run the same image on the production server as we did for the tests or the other stages of the build pipeline (e.g. pre-production-system, acceptance tests, capacity tests etc.).



Once the Docker image is built, we use the image in all stages of the (continuous) delivery pipeline. This increases the reliability of our delivery.

Advantages for the Operations Team

- Less effort for maintaining environments for the application. The creation of an environment is automated. Hence, less manual actions are necessary. This reduces the risks and increases the reliability.
- Manually maintaining a consistent environment on multiple servers is error-prone and can be a nightmare. With Docker it's easy to create several instances of an environment, because we just have to execute the image on the servers. This way it is easy to add further nodes to a cluster and to scale horizontally.
- Easy updating of an existing application and environment. Using traditional approaches for setting up an environment (like install scripts) run into trouble when there is already an existing environment. Considering the update path in the script can be a very complex task (e.g. checking the existence of files and clean up unused files). With Docker we don't have to take an existing environment into account (except for the database). We just stop the running container and start the new updated one. This simplifies the setup because we always start with an empty environment.

Infrastructure as Code

When using Docker (and other tools like Chef or Puppet) to set up our environment, we benefit from the "Infrastructure-as-Code" approach. The infrastructure is not created manually any longer but is a result of an automated process. We describe our infrastructure explicitly. In case of Docker, the Dockerfile is our documentation describing our environment. This leads to the following benefits:

- We can put our Dockerfile under version control and track changes made to the environment. This way, we always know what application release belongs to which version of the environment.
- We get a reliable documentation of our infrastructure. We always know what our applications need to run and what is currently installed in the environment.
- Moreover, the documentation is always up to date, because we need to change it in order to change the environment.

- Therefore, setting up a new environment is easy, because we know what we need to install.

Comparing to other Automation Tools

With tools like Chef or Puppet you can also achieve a lot of the advantages of Docker (like reproducible environments, infrastructure documentation and easy setup of a new node). However, when we use virtualization approaches (like Docker) we gain the following advantages:

- When we want to implement Continuous Delivery, we need to set up multiple instances of our environment for each stage of the delivery pipeline (commit stage, acceptance test, capacity tests, pre-production and production). Using physical machines for this purpose is not practicable.
- The installation of the application and the necessary environment becomes easy, because we always start with an empty VM (in fact Docker doesn't start a real VM, see below). The old VM can be removed completely and the new one is set up.

Lightweight Virtualization

For this reasons, Chef or Puppet are often used in conjunction with virtualization: a virtual machine is started and Chef or Puppet is used to set up the environment within the VM. When using Docker instead of a real VM we have the following benefits:

- Lightweight virtualization. Starting a Docker container is much faster than starting a VM, because no guest operating system has to be booted. This reduces the overhead. The several containers share the kernel of the host operating system, but have their own file system, users, network and processes. From the perspective of the host operating system we just start another process, when we run a Docker container. This significantly speeds up the startup of a container while still providing a good isolation of the containers.

Disadvantages and Challenges

After praising Docker (too much?), let's consider the drawbacks and challenges when using Docker:

- Increased complexity due to an additional layer. This affects not only the deployment but also the development and build.
- In addition, managing a huge amount of containers is challenging – especially when it comes to clustering containers. Tools like Google Kubernetes and Apache Mesos can help here.
- The containers share the same kernel and are therefore less isolated than real VMs. A bug in the kernel affects every container.
- Docker bases on Linux Containers (LXU), which is a Linux technology. Therefore, we can't run Docker on other systems and our container is always a Linux system. But Boot2Docker enables the usage of Docker on Windows and Mac OS X by using

VirtualBox. The Docker client runs on the host OS and communicates with the Docker daemon inside the VirtualBox. Unfortunately, this is less comfortable and makes daily use clumsy and more complicated than running Docker natively.

- Introducing Docker can be a demanding and time-consuming task. We have to evaluate if it is worth the effort and the increased complexity. In a past project of mine, the management rejected Docker with the argument, that 3 nodes (production, pre-production and experimental) are not enough to justify the effort… However, when you have a cluster of nodes (when scaling horizontally) or use Continuous Delivery you need an approach to reproduce environments and Docker is brilliant in this. Moreover, Continuous Delivery reduces the time-to-market (new features can be brought to production faster because the deployment and environment setup is automated), which can be used to convince the management.
- I also made the experience that there are also reservations about Docker in strictly regulated domains (like the banking sector):
    - To run a container you need root rights. This can be a problem for some companies, in which the colleagues that are supposed to run and update the application on the production server must not have root rights.
    - It is unclear if the usage of Docker agrees with certain security standards which have to be fulfilled (like PCI).
    - In some companies, there is the questionable rule, that only software from official/trusted sources can be installed on the machines. For instance, Docker is not included in Red Hat Enterprise Linux 6 and therefore needs to be installed from docker.com, which is an “untrusted source”.