

****

**Docker Agile**

Author by: Sanjay C H

**Intro**

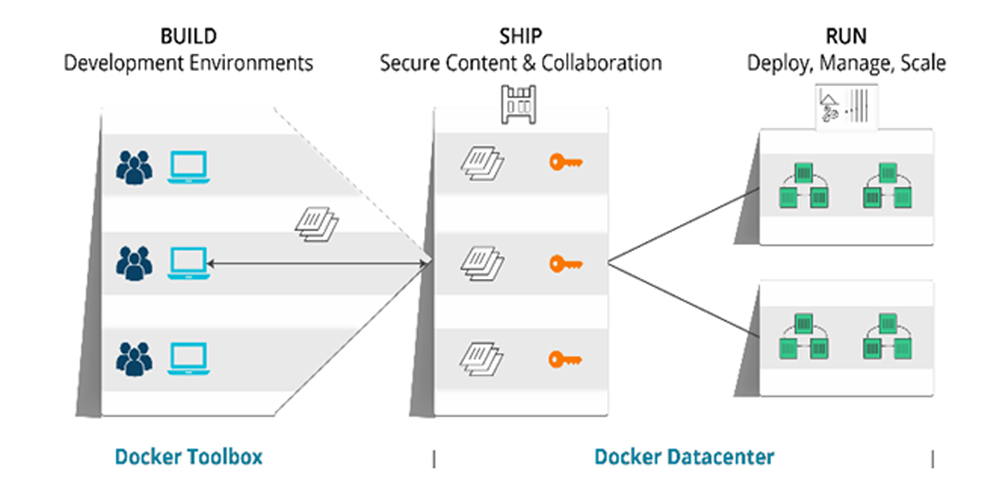
In today’s world, especially in Information Technology, we are seeing continuous changes day-by-day. These changes are related to technology which is making our daily life easier and more convenient by making use of Apps & Websites. Using these we can do anything from any location, for example, we can order food online, buy new clothes, hire a cab, book a hotel, do banking transactions and much more. All we need is, just an Internet connection plus a smartphone or a computer/laptop.

So whenever our demand or requirement changes we have to look for the alternate technologies or we need to enhance the existing ones. And when the new technology emerges, with that the concept plus logic which we have used also needs support, not only by the developing strategy plus by the Infra where the things are hosted and are responsible for creating the better version of Technology.

That’s why the title of this blog is set to **“Why Docker and Agile are a “Good Couple”,**because AGILE is the strategy that is used to plan & develop a technology/software, whereas Docker is the way of deploying this software to a number of Containers, hosted on a single or multiple servers.

To understand all this, first, we need to understand what exactly Agile and Docker is. So, here in this blog, we will make you understand how both of these runs in parallel to support each other in an efficient way.

**What Docker Is?**



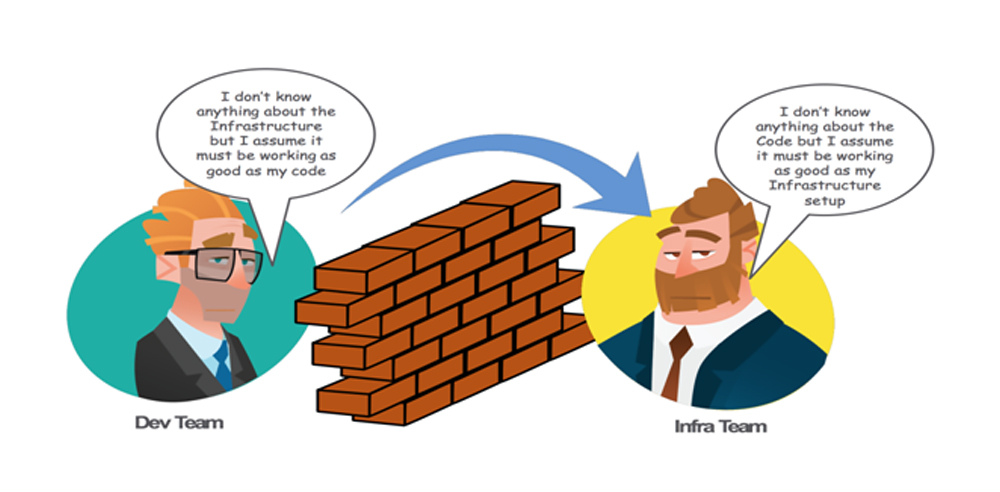
Docker is an open-source platform where one can develop, ship, as well as run his/her applications by using the concept of Containerization. We can also say that it is a container management service. Using docker containers we can easily separate our applications from our infrastructure which in-turn helps us in delivering software quickly. Also by implementing the Docker’s methodology, we can easily reduce the effort from writing the code, to test the code, to make it run in the production.

By adopting the concept of Docker we can easily develop applications, can ship them into containers and finally can deploy them anywhere in our infrastructure. This feature comes as a helping hand for all the developers throughout the world.

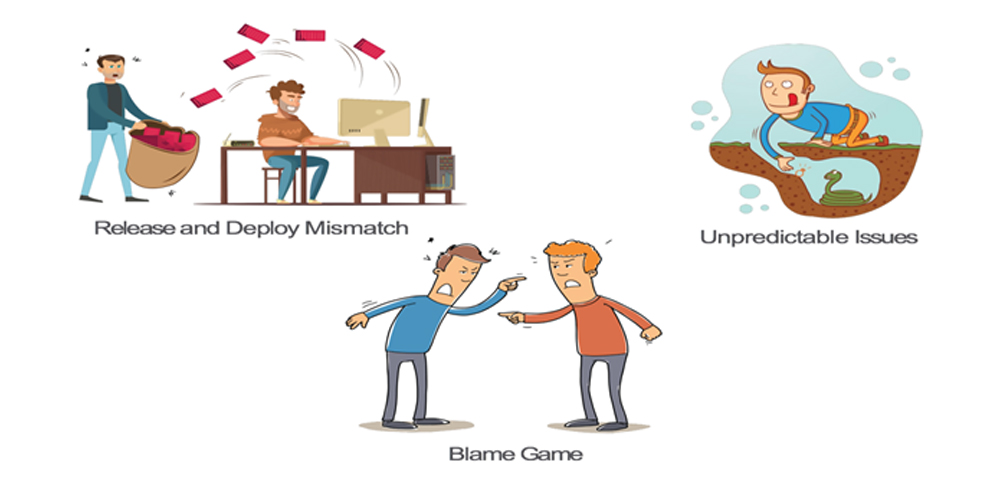
Docker becomes very useful since its launch in 2013, especially in Agile based development projects. Below diagram will help you understand what we have written here about docker.

**Agile collaboration**

To understand this, we need to know what were the challenges that came while developing software using an Agile Methodology. As we know, Agile is the way to follow the best practices in order to create quality software, but all these stages involve people who are working individually, or you can say that who are working in Silos with their own goals/key areas. This situation results in no communication between the teams who are eventually working for a common objective, that is, a reliable software. Look the below picture, where Dev team is not having any idea about the Infrastructure where the code is going to be deployed. Similar is the case with the Infra team



Also, the Agile Development without Docker culture results in many issues as shown in the below picture





To overcome all these situations the need for containerization with Docker arises.

The reason I am only talking about DOCKER instead of Virtualization is because Dev-Infra team found it more easy to understand, easy to implement plus it helps in saving a lot of hardware resources, as we can run a number of Docker containers, in which our software/applications reside. Also, it becomes more easy to monitor the Software health check and the people working in Silos started working as a team. Below image will give you more clarity on what I want to explain to you,

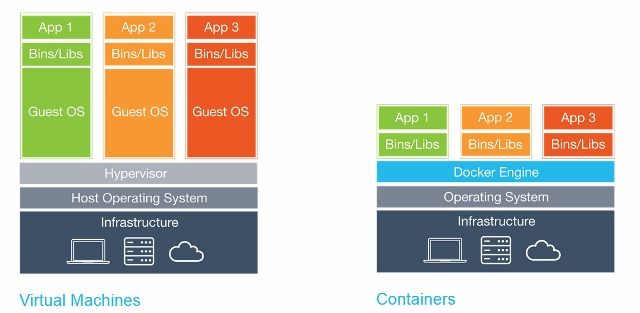
**Docker Enables Agile Software Development**

Docker is the de facto containerization platform, and it has revolutionized how software is packaged, distributed, and deployed. It runs software that is packaged and distributed as Docker images in Docker containers that run on Docker Engine.

Docker has facilitated the adoption of the microservices architecture, which decouples services components and facilitates making iterative changes to software services. In fact, Docker makes software development and deployment more agile.

**Docker Design**

Docker design is sustainable, as it makes a more efficient use of the operating system compared to a virtual machine.

Virtual machines run on top of a hypervisor, which runs on top of an underlying OS. Each VM uses up a whole guest operating system, which is not very efficient or sustainable in terms of resource consumption. 

A Docker container does not make use of a whole operating system, instead only employing a snapshot of the underlying OS kernel, thus making it more lightweight and sustainable in terms of resource consumption. Multiple Docker containers run in isolation, with each having its own file system and networking, on top of a single Docker Engine using the same OS kernel, as illustrated in figure 2.

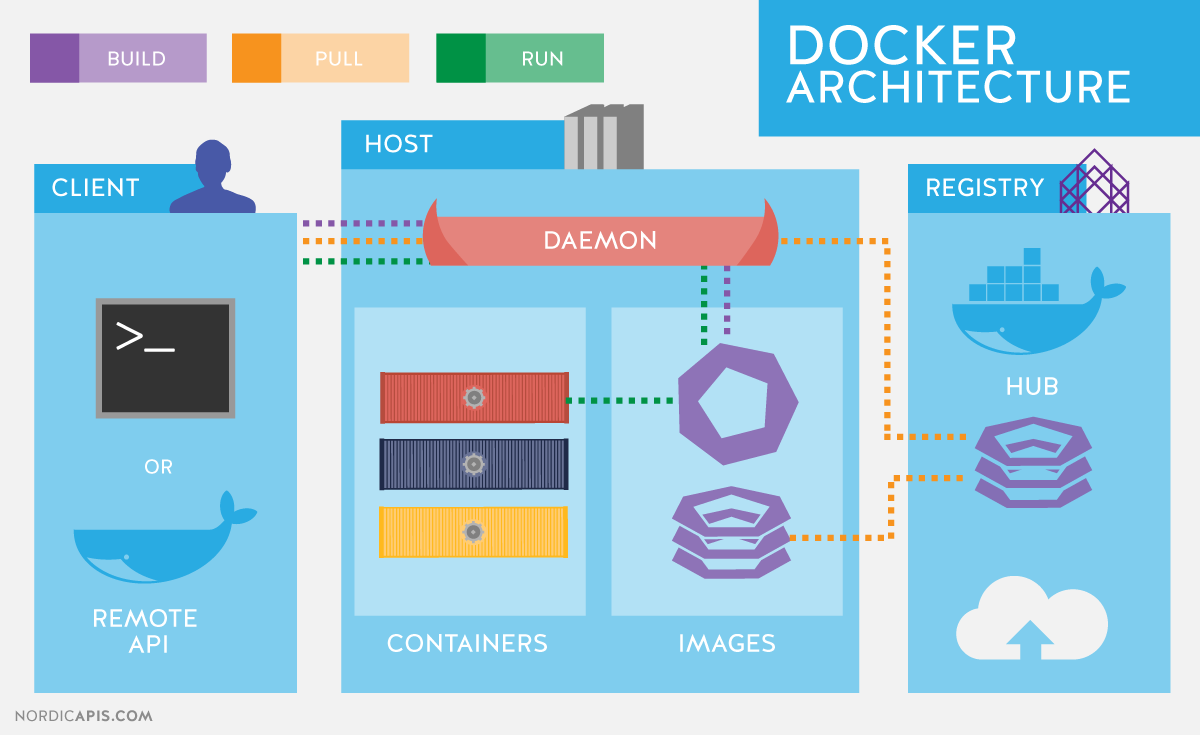
Docker design is simpler, modular, and less resource-intensive, which encourages leaner, more agile development practices.

**Efficient Software Delivery**

Docker delivers pre-packaged software in the form of reusable, modular Docker images. More specifically, a Docker image is built from a Dockerfile, which consists of instructions and commands to run in order to download, install, and run the software.

A Docker image is a set of layers, with each layer representing an instruction or command in a Dockerfile. This takes away the hassle of downloading and installing individual components of software.

Docker images may be pulled or downloaded from a repository, such as Docker Hub, and are provided for Linux and Windows OS and support several different types of architectures, including amd64, arm32v5, arm32v7, arm64v8, i386, ppc64le, s390x, and windows-amd64.



**Working Software**

Docker provides working software in that the software is ready to be run without further configuration. The simple command **docker run <image>** runs the software packaged in a Docker image and all the dependencies packaged with it.

For example, if a software depends on a specific version of Java, the Java version is also downloaded and installed with the other software that is downloaded and installed. Running software from a Docker image.

**Accommodating Changing Requirements**

A Docker image is built from a Dockerfile, which consists of Docker syntax instructions. A Dockerfile gets built into a Docker image with the docker build command, and the image is tagged to distinguish the different builds generated from the same Dockerfile.

If some requirement changes, the Dockerfile could be modified accordingly to generate a new image with a new tag. Consequently, multiple versions of software could be made available using different tags.

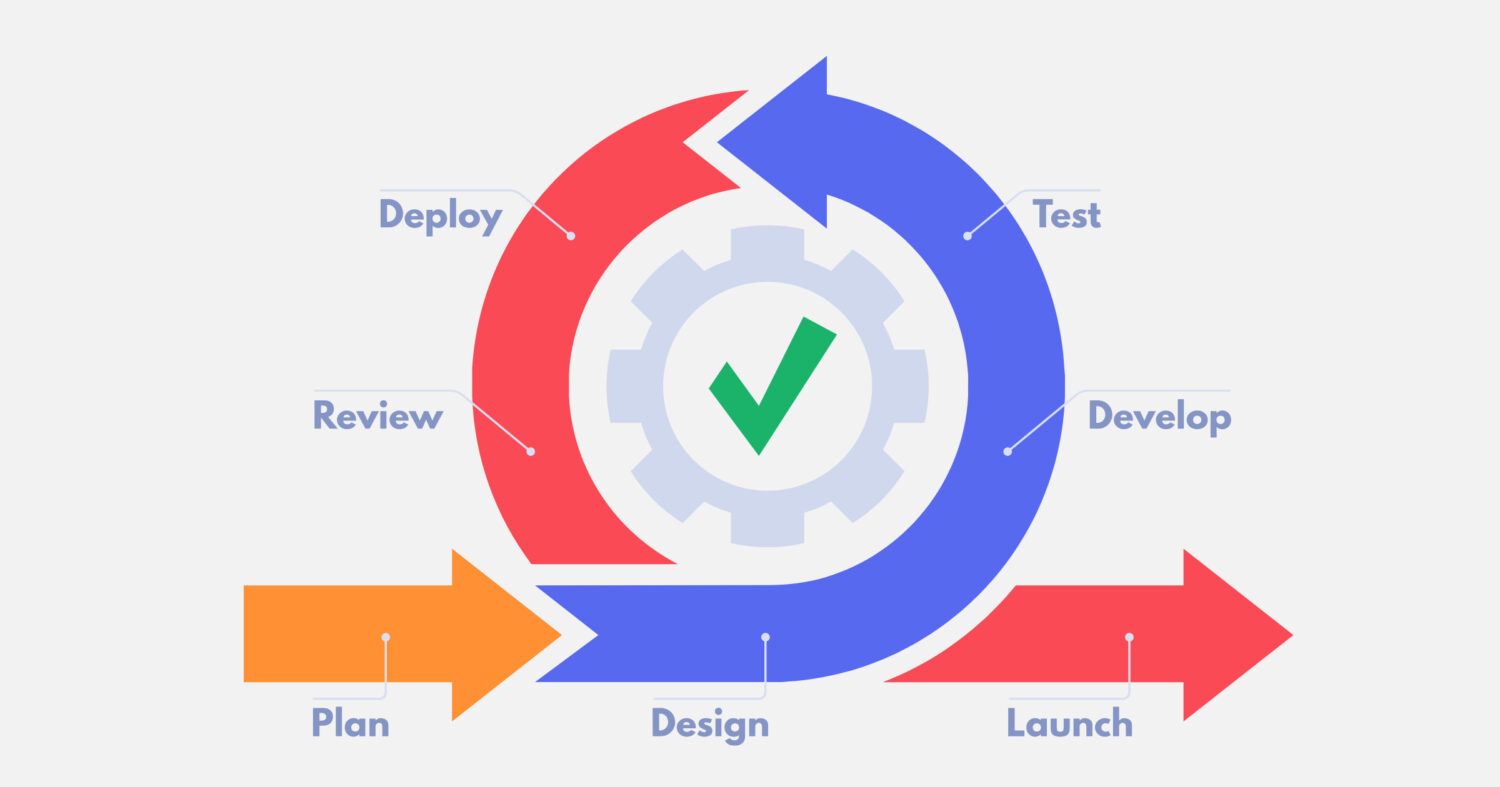
The default tag is “latest,” and a subsequent Docker image built using a tag that already exists overwrites an earlier image with the same tag. Tagged Docker images for three different versions (v1, v2, and v3) of a Dockerfile .

**Iterative, Test-Driven Development**

Because Docker distributes ready-to-install software as pre-packaged Docker images, it supports iterative, test-driven development.

The source code could be hosted on an online repository such as GitHub. A single command docker build creates a Docker image from the source code Dockerfile. The Docker image could be tested in a test environment before deploying in production. A single command docker run deploys the Docker image as running software.

Most Docker container orchestration platforms support rolling upgrades so that software can be updated and deployed iteratively.



**Automation**

Docker lends itself to automation very well. Each of the build, test, and deploy processes could be automated with pipeline-based automation tools such as Jenkins.

Continuous Integration and Continuous Testing

In the context of Docker, continuous integration refers to integrating source code that is checked into a source code control system (like GitHub) into a Docker image continuously with each successive check-in.

Build automation tools like Jenkins could be used to develop a build pipeline that builds source code on GitHub into a new Docker image each time code is committed to GitHub. The Docker image also could be tested continuously using automated tests in the build pipeline. After testing a Docker image, it could be uploaded to a Docker image repository, such as Docker Hub, using the docker push command, and this process can also be automated in the build pipeline.

As a result, the source code for software could be integrated continuously into a usable form of a Docker image.

**Continuous Delivery**

Continuous delivery is the next phase in the software development process. Continuous delivery is defined as making usable software available for deployment without actually deploying the software into production.

Continuous delivery may include deploying software into some staging environment after passing CI and running a suite of tests against the software in that environment. A user or administrator has to approve the software for deployment into production. A build pipeline again could be used for continuous delivery.

Converting a Docker image into production-quality software could involve further testing to sure an image is usable. Some services also require the microservices they depend on to be available in some way to make the service useful.

**Continuous Deployment**

Continuous deployment fully automates software development, testing, and running an application. The usable software is deployed continuously to production without user intervention by using rolling upgrades, as illustrated in figure 8. A build pipeline could be used for continuous deployment as well.

*Collaboration with Software Users*

By automating the Docker build, test, deliver, and deployment processes, it becomes easier to collaborate with your software’s end-users.

End-user production deployments of artifacts that have passed through a continuous delivery cycle are valuable. This gives the development team immediate feedback on the software while waiting for the users to be ready to accept a new release on their terms. Because Docker images are tagged, different end-users could use different versions of the same software customized to their needs.

A multi-branch Jenkins pipeline provides for further collaboration with software end users. For example, some of the branches of the pipeline could be allocated to the software end-user team while the other branches are managed by the software development team. The end-users may suggest changes more frequently than when using a non-Docker application, as it is easier to update software packaged, distributed, and deployed with Docker.

**A Tool for Agile Work**

Docker facilitates modular design for working software, sustainable resource consumption, efficient software delivery, continuous integration, continuous delivery, continuous deployment, and collaboration with end-users, all of which are founding principles of agile software development. In this way, using Docker as your containerization platform can actually help make your software development, testing, delivery, and deployment more agile.