4. E. Nezhnikova. IOP Conf. Series: Earth and Environmental Science 90 012161 (2017) doi:10.1088/1755-1315/90/1/012161

MICROSERVICES VIRTUALIZATION

Khmyl V.A.

Belarusian State University of Informatics and Radioelectronics, Minsk, Belarus

Microservices is the most popular architecture style and effective management of them makes development faster and deployment easier.Docker offers solution to virtualize microservices based on containers. Containers make microservices lightweight, scalable, independent and more stable.

For the past couple of decades, the most common way for enterprise software to be developed and sold was as monolithic packaged applications and even larger platforms – all fully built and integrated in a big chunk. Today, due in large part to the growth of cloud computing, agile development processes, performance improvements, and the need for scalable, lightweight applications and enterprise architecture flexibility, the movement has been toward microservices and containerization.

Microservices is an approach to application development in which a large application is built as a suite of modular, single-use services. The idea is that microservices should focus on one component of the application and do that one thing exceptionally well. Each module supports a specific business goal and uses a simple, well-defined interface to communication with other modules (Figure 1.1).

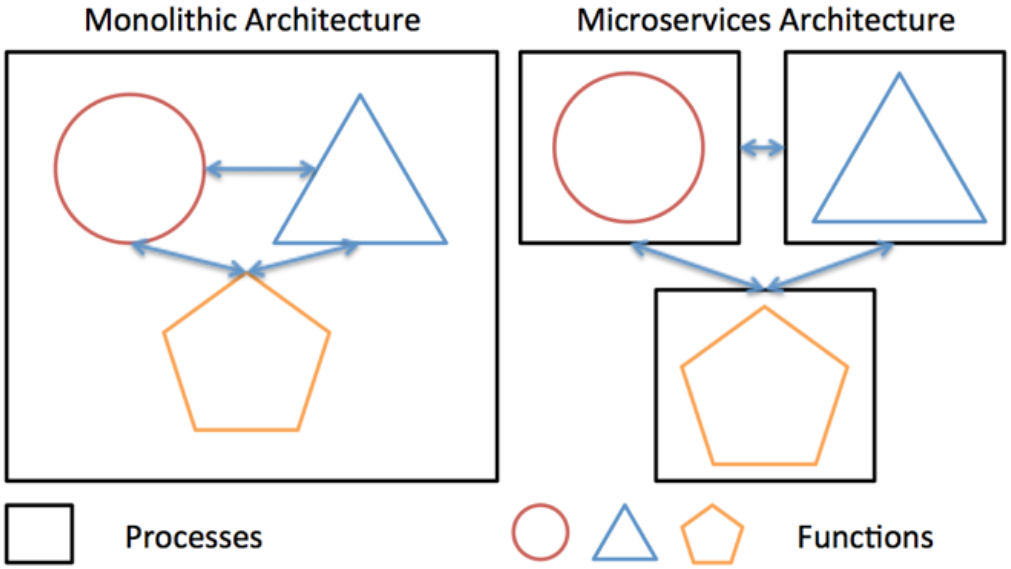

Figure 1.1 – Architecture styles of applications

Containers are the tools and methodology used to organize and develop microservices. Container-based virtualization uses a single kernel to run multiple instances of an operating system. A container is an isolated part of system that includes everything it needs to run: code base, system tools and libraries, configuration and setting. Each instance runs in a completely isolated environment, so there is no risk that one container can gain access to another's files. This allows for different teams to work on different microservices simultaneously.

Microservices approach offers several benefits, including the ability to scale individual microservices, keep the codebase easier to understand and test, and enable the use of different programming languages, databases, and other tools for each microservice. Although this technique solves many problems, it also has several disadvantages [1].

Some problems in a microservice architecture that you can face are:

Once your number of microservices grow, it can be hard to keep track of them;

You will need to consider things such as: how to handle the communication between microservices, handle errors to avoid disrupting other microservices, and add more test cases in each component;

Finding and tracing the bugs/errors in your application;

Microservices could consume more resources compared to a monolithic app.

Docker is an excellent tool that can make managing and deploying of microservices easier. Each microservice can be further broken down into processes running in separate Docker containers, which can be specified with Dockerfiles and Docker Compose configuration files. Specifying an environment in this way also makes it easy to link microservices together to form a larger application. In a way, Virtual machines are the precursors to Docker containers. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer (Figure 1.2).

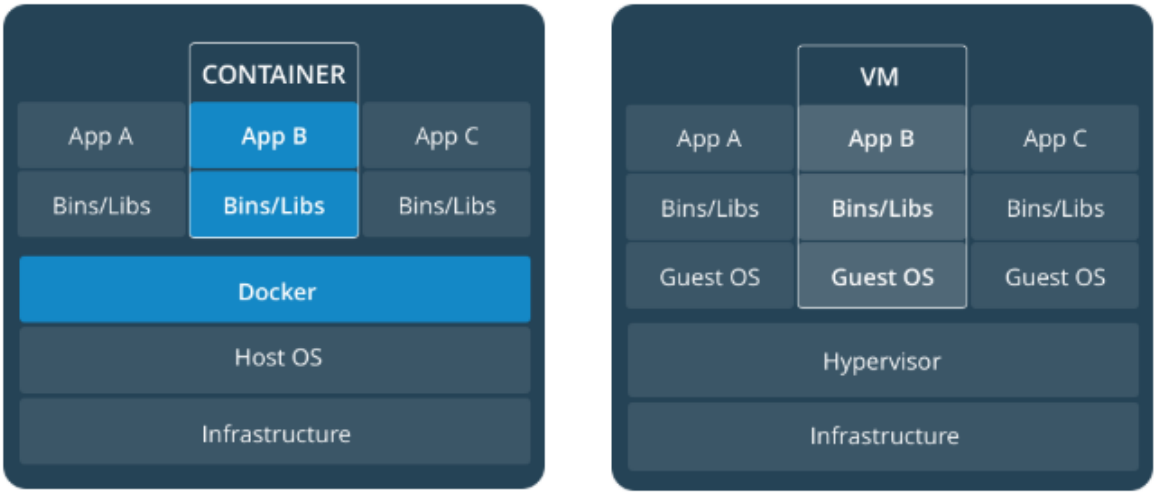

Figure 1.2 – Comparison of virtual machines and docker containers

A Dockerfile is a file with instructions for how Docker should build your image. A Docker image is an executable package that includes everything needed to run an application. The Dockerfile refers to a base image that is used to build the initial image layer. Popular official base images include python, ubuntu, and alpine. Additional layers can then be stacked on top of the base image layers, according to the instructions in the Dockerfile. Each layer is read only, except the final container layer that sits on top of the others. This is a small volume layer containing a program that will run in a container. The Dockerfile tells Docker which layers to add and in which order to add them. Layer are just files with the changes since the previous layer. A Dockerfile instruction is a capitalized word at the start of a line followed by its arguments. Each line in a Dockerfile can contain an instruction. Instructions are processed from top to bottom when an image is built. The simplest instruction can be the following:

FROM ubuntu:18.04
COPY . /app

Only the instructions FROM, RUN, COPY, and ADD create layers in the final image. Other instructions (ENV, ARG, EXPOSE, VOLUME, LABEL, ENTRYPIONT) configure things, add metadata, or tell Docker to do something at run time, such as expose a port or run a command.

If you want other people to be able to make containers from your image, you send the image to a container registry. Docker Hub is the largest registry and the default [2].

For developers who are starting to build their applications, they should decide whether it would be beneficial to them to use a microservices architecture rather than a monolithic one. They should consider the long-term usability and scalability of their application. Managing multiple microservices can be challenging, but useful tools such as Docker, Kubernetes, etc. help developers minimize disadvantages of microservices. By introducing Docker in your development, you will get all the benefits of using microservices and containers: stable working, independent deployment, application portability, resource utilization, vertical and horizontal scalability.

Above were considered the basic principles of microservices and Docker containerization which will help to get into this topic deeper or decide to use them. Docker is easily integrated and can be adopted by technical and business needs, that’s why it becomes more popular and widely used.

References:

1. S. Newman, “Building Microservices: Designing Fine-Grained Systems”, 1st edition, O'Reilly Media, February 2015.
2. Docker Docs [Internet resource]. – Available at: https://docs.docker.com (Accessed 15 March 2019).