Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

# Docker and Kubernetes support for HTM

Shahab Uddin
shahab.uddin@stud.fra-uas.de

***Abstract*— Loosely-coupled distributed systems organized as collections of so-called cloud-native microservices are able to adapt to traffic in very fine-grained and flexible ways. For this purpose, the cloud-native microservices exploit containerization such as Docker and container management systems such as Kubernetes.**

**Docker is a latest technology that allows development teams to build, manage, and secure apps anywhere. Docker makes it possible to get far more apps running on the same old servers and it also makes it very easy to package and ship programs. It's not possible to explain what Docker is without explaining what containers are. Containers, however, use shared operating systems. This means they are much more efficient than hypervisors in system resource terms. Instead of virtualizing hardware, containers rest on top of a single Linux instance.**

***Keywords*— *docker, containers, hypervisors virtual machines, kubernestes, docker daemon, virtualization***

## I. Introduction

The goal of this project was to containerize the current implementation of HTML algorithm. The primary advantage of containerized application is to deploy the application on any supported container runtime such as Docker, CRI-O, Containerd and many more. Containerized application can also be deployed on public, private or hybrid cloud environments. Docker is a widely used platform for developers and sysadmins to develop, ship, and run applications, anywhere. Docker is popular because it has revolutionized development. Docker and the containers have revolutionized the software industry and in five short years their popularity as a tool and platform has skyrocketed. The main reason is that containers create vast economies of scale. Systems that used to require expensive, dedicated hardware resources can now share hardware with other systems. Containers are self-contained and portable. If a container works on one host, it will work just as well on any other, as long as that host provides a compatible runtime.

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Kubernetes architecture is based on running clusters that allow containers to run across multiple machines and environments. Each cluster typically consists of two classes of nodes:

**Worker nodes**, which run the containerized applications.

**Control plane nodes**, which control the cluster.

The control plane basically acts as the orchestrator of the Kubernetes cluster and includes several components—the API server (manages all interactions with Kubernetes), the control manager (handles all control processes), cloud controller manager (the interface with the cloud provider's API), and so forth. Worker nodes run containers using container runtimes such as Docker. Pods, the smallest deployable units in a cluster hold one or more app containers and share resources, such as storage and networking information.

## II. Methodology

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so we don't need to rely on what's installed on the host (operating system).

In order to run Docker containers, the very first step is to create a Dockerfile. This Dockerfile container the instructions and commands to use a base image, either an operating system or any other binaries required by the application. This Dockerfile contains the command used to create the complete package and also an entry point of the executable.

Here in this project, I have two main applications to containerize. First is neocortexapi, which is used to run the experiments and generate SDR values. Second is the draw_figure python application, which takes the SDR values as an input and generate the Column Activity Diagram.

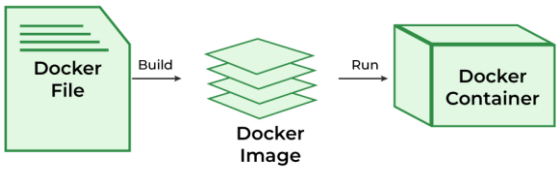


Fig. 1. Docker container creation process

Command line arguments are a powerful way to pass parameters to a program when it is executed. They allow users to customize the behavior of the program without modifying its source code. Command-line arguments are used extensively in the Unix/Linux command-line interface and are also commonly used in scripting languages like C#, Python, Java and more.

Compared to other types of user input like prompts and web forms, command-line arguments are more efficient and can be automated easily. They can also be used to run scripts in batch mode, without the need for user interaction.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is perfect for storing temporary data. For example, temporary data can be user-generated data, such as a submitted form on a website. JSON can also be used as a serialization data format for any programming language to provide a high level of interoperability.

A JSON object data type is a set of name or value pairs inserted between {} (curly braces). The keys must be strings and separated by a comma and should be unique.

{ "Influencer" : { "name" : "Jaxon" , "age" : "42" , "city" , "New York" } }

For example in this project JSON is being used to pass the parameters through Command Line Interface

D:\ShahabUddin\neocortexapi\source> docker run a203 MSL fileName2058.txt [{\"key\":\"S1\",\"values\":[1.0,2.0,3.0,4.0,5.0,6.0,7.0]},{\"key\":\"S2\",\"values\":[10.0,11.0,12.0,13.0,14.0,15.0,16.0]}]

Json data format is widely used data exchange format for different kind of applications.

## III. Results

This section describes the results of this project that includes Dockerfile for containerization and related changes in console application to take algorithm name and input values as Command Line Interface arguments.

The conditional statements in console application startup helps to run the specific algorithm when running the container from the Docker image. This also ensures that each algorithm is running in separate space of container resources and does not interfere with other algorithm running in different containers.

```
if (algorithmName == "SPL")
{
    Console.WriteLine(">> Running SpatialPatternLearning Experiment");

    //Starts experiment that demonstrates how to learn spatial patterns.
    SpatialPatternLearning experiment = new SpatialPatternLearning();
    experiment.Run();
}
else if (algorithmName =="MSSL" )
{
    Console.WriteLine(">> Running MultiSimpleSequenceLearning Experiment");

    var sequenceList = JsonConvert.DeserializeObject<List<Sequence>>(inputSequences);
    Dictionary<string, List<double>> sequences = new Dictionary<string, List<double>>();
    foreach (var request in sequenceList)
    {
        sequences.Add(request.Key, request.Values);
    }

    RunMultiSimpleSequenceLearningExperiment(sequences);
}
else if (algorithmName == "MSL")
{
    Console.WriteLine(">> Running MultiSequenceLearning Experiment");

    var sequenceList = JsonConvert.DeserializeObject<List<Sequence>>(inputSequences);
    Dictionary<string, List<double>> sequences = new Dictionary<string, List<double>>();
    foreach (var request in sequenceList)
    {
        sequences.Add(request.Key, request.Values);
    }

    RunMultiSequenceLearningExperiment(sequences);
}
```

Fig. 2. Command line arguments for different types of experiments

Following are the commands used in current project Dockerfiles

FROM command is used to define the base image either the SDK of .net8 or runtime of .net8

WORKDIR instruction sets the current working directory for subsequent instructions

COPY command is used during Docker image building to duplicate local files and directories into the Docker image

RUN is the central executing directive for Dockerfiles

ENTRYPOINT is to set up images for specific applications or services

```
Dockerfile
# Fetch base image from DockerHub

# Runtime environment image
FROM mcr.microsoft.com/dotnet/runtime:8.0 AS runtime-env
WORKDIR /app


# Build environment image
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build-env
WORKDIR /src

# Copy everything
COPY . .

# Restore dependent projects
RUN dotnet restore NeoCortexApi/NeoCortexApi.csproj
RUN dotnet restore NeoCortexArrayLib/NeoCortexArrayLib.csproj
RUN dotnet restore NeoCortexEntities/NeoCortexEntities.csproj
RUN dotnet restore NeoCortexUtils/NeoCortexUtils.csproj
RUN dotnet restore GridCell/GridCell.csproj
RUN dotnet restore Samples/NeoCortexApiSample/NeoCortexApiSample.csproj

# Restore solution
RUN dotnet restore NeoCortexApi.sln

WORKDIR "/src/"

RUN dotnet build Samples/NeoCortexApiSample/NeoCortexApiSample.csproj -c Debug -o /app/build
FROM build-env AS publish

RUN dotnet publish Samples/NeoCortexApiSample/NeoCortexApiSample.csproj -c Debug -o /app/publish
FROM runtime-env AS final

WORKDIR /app
COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "NeoCortexApiSample.dll"]
```

Fig. 3. Dockerfile for neocortexapi console application

```
Dockerfile
1  FROM python:3.12.1
2
3  # Copy everything
4  COPY . ./
5
6  # RUN python -m ensurepip
7  RUN pip install  plotly
8  # RUN pip install matplotlib
9  # RUN pip install numpy
10
11 WORKDIR  /ColumnActivityDiagram
12
13 # Run hello.py when the container launches
14 ENTRYPOINT ["python", "draw_figure.py"]
15
16 # Pass arguments while running the container from this Dockerfile
17 # CMD ["-fn", "sampleOne.txt", "-gn", "A2024123", "-mc", "19", "-ht", "8", "-yt",
18 # "yaxis", "-xt", "xaxis", "-min", "50", "-max", "4000", "-st", "'single column'",
19 # "-fign", "CortialColumn", "-a"]
20
```

Fig. 4. Dockerfile for draw_figure

### A. *Column Activity Diagrams*

Python project takes SDR values as input and generates Column Activity Diagrams.

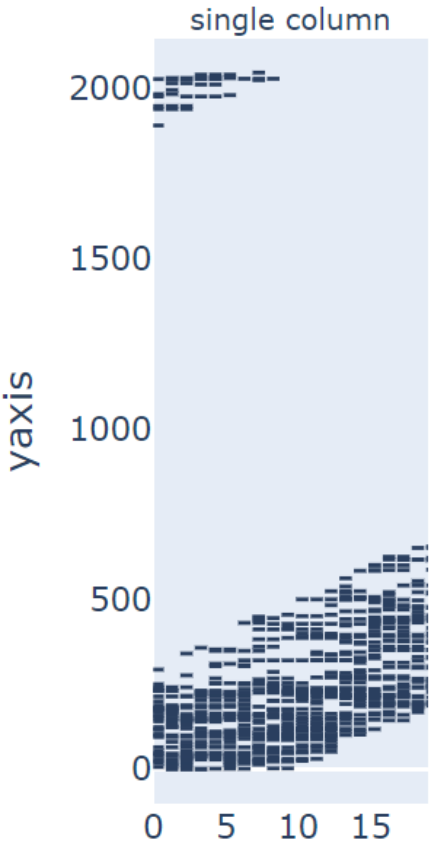


Fig. 5. Column activity diagram generated n container

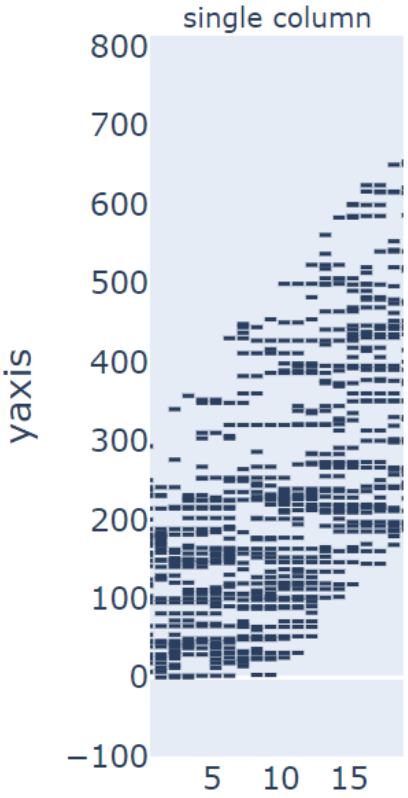


Fig. 6. Colume activity diagram (zoomed)

## IV. Containerization

### A. *Containerization*

Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure. Traditionally, to run any application on your computer, you had to install the version that matched your machine's operating system. For example, you needed to install the Windows version of a software package on a Windows machine. However, with containerization, you can create a single software package, or container, that runs on all types of devices and operating systems.

### B. *Use Cases Of Containerization*

Following are the widely adopted use cases of containerization to build and deploy modern applications

- Cloud migration: Cloud migration, or the lift-and-shift approach, is a software strategy that involves encapsulating legacy applications in containers and deploying them in a cloud computing environment. Organizations can modernize their applications without rewriting the entire software code.
- Adoption of microservice architecture in software: Organizations seeking to build cloud applications with microservices require containerization technology. The microservice architecture is a software development approach that uses multiple, interdependent software components to deliver a functional application. Each microservice has a unique and specific function. A modern cloud application consists of multiple microservices. For

example, a video streaming application might have microservices for data processing, user tracking, billing, and personalization. Containerization provides the software tool to pack microservices as deployable programs on different platforms.

- IoT devices: Internet of Things (IoT) devices contain limited computing resources, making manual software updating a complex process. Containerization allows developers to deploy and update applications across IoT devices easily.

### C. *Benefits Of Containerization*

Developers use containerization to build and deploy modern applications because of the following advantages.

- Portability: Software developers use containerization to deploy applications in multiple environments without rewriting the program code. They build an application once and deploy it on multiple operating systems. For example, they run the same containers on Linux and Windows operating systems. Developers also upgrade legacy application code to modern versions using containers for deployment.
- Scalability: Containers are lightweight software components that run efficiently. For example, a virtual machine can launch a containerized application faster because it doesn't need to boot an operating system. Therefore, software developers can easily add multiple containers for different applications on a single machine. The container cluster uses computing resources from the same shared operating system, but one container doesn't interfere with the operation of other containers.
- Fault tolerance: Software development teams use containers to build fault-tolerant applications. They use multiple containers to run microservices on the cloud. Because containerized microservices operate in isolated user spaces, a single faulty container doesn't affect the other containers. This increases the resilience and availability of the application.
- Agility: Containerized applications run in isolated computing environments. Software developers can troubleshoot and change the application code without interfering with the operating system, hardware, or other application services. They can shorten software release cycles and work on updates quickly with the container model.

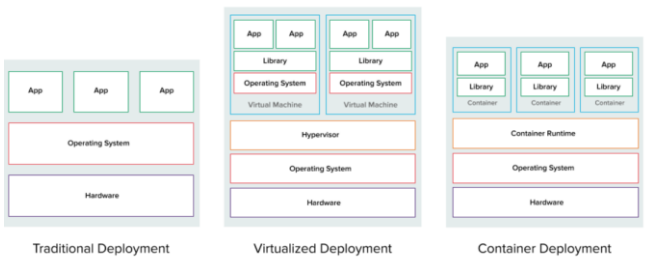


Fig. 7. Deployment methos comparison

## V. DOCKER

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

### A. *How Docker works*

Docker works by providing a standard way to run your code. Docker is an operating system for containers. Similar to how a virtual machine virtualizes (removes the need to directly manage) server hardware, containers virtualize the operating system of a server. Docker is installed on each server and provides simple commands you can use to build, start, or stop containers.

### B. *When to use Docker*

You can use Docker containers as a core building block creating modern applications and platforms. Docker makes it easy to build and run distributed micro services architectures, deploy your code with standardized continuous integration and delivery pipelines, build highly-scalable data processing systems, and create fully-managed platforms for your developers.

### C. *Why use Docker*

Using Docker lets you ship code faster, standardize application operations, seamlessly move code, and save money by improving resource utilization. With Docker, you get a single object that can reliably run anywhere. Docker's simple and straightforward syntax gives you full control. Wide adoption means there's a robust ecosystem of tools and off-the-shelf applications that are ready to use with Docker.

You can use Docker containers as a core building block creating modern applications and platforms. Docker makes it easy to build and run distributed microservices architecures, deploy your code with standardized continuous integration and delivery pipelines, build highly-scalable data processing systems, and create fully-managed platforms for your developers.

### D. *Docker Key Components*

- Dockerfile: Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- DockerImage: A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run.
- DockerContainer: A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies. This container runtime runs on the engine on a server, machine, or cloud instance. The engine runs multiple containers depending on the

underlying resources available. A Docker container is a self-contained, runnable software application or service. On the other hand, a Docker image is the template loaded onto the container to run it, like a set of instructions.

- Docker Commands: Docker provides commands that are used on command line interface or build pipelines to interact with Docker runtime. For example, PS command is used to check running containers. IMAGE command is used to check build images in the platform. Similarly RUN command is used to run the container.

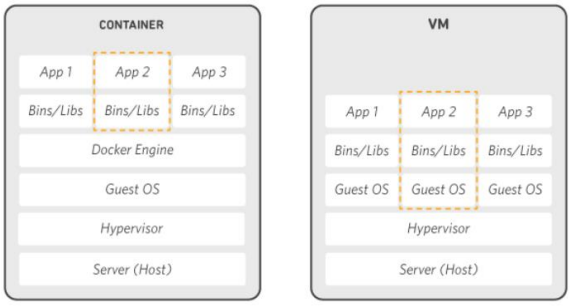


Fig. 8. Architecture of Hypervisor & Docker

## VI. KUBERNETES

Kubernetes (also known as k8s or "kube") is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

Originally developed and designed by engineers at Google as the Borg project, Kubernetes was donated to the Cloud Native Computing Foundation (CNCF) in 2015.

### *A. Kubernetes Cluster*

A working Kubernetes deployment is called a cluster, which is a group of hosts running Linux® containers. You can visualize a Kubernetes cluster as two parts: the control plane and the compute machines, or nodes.

Each node is its own Linux environment, and could be either a physical or virtual machine. Each node runs pods, which are made up of containers.

The control plane is responsible for maintaining the desired state of the cluster, such as which applications are running and which container images they use. Compute machines actually run the applications and workloads. The control plane takes commands from an administrator (or DevOps team) and relays those instructions to the compute machines.

This handoff works with a multitude of services to automatically decide which node is best suited for the task. Services decouple work definitions from the pods and automatically get service requests to the right pod—no matter where it moves in the cluster or even if it's been replaced. It allocates resources and assigns the pods in that node to fulfill the requested work.

Kubernetes runs on top of an operating system and interacts with pods of containers running on the nodes.

The desired state of a Kubernetes cluster defines which applications or other workloads should be running, along with which images they use, which resources should be made available to them, and other such configuration details.

There is little change to how you manage containers using this type of infrastructure. Your involvement just happens at a higher level, giving you better control without the need to micromanage each separate container or node.

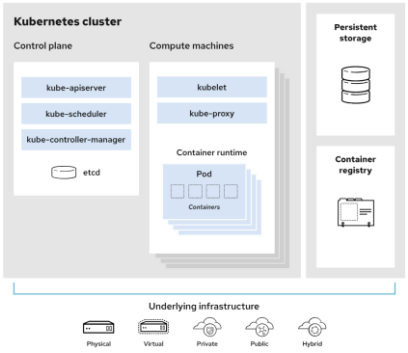


Fig. 9. Kubernetes Architecture

Where you run Kubernetes is up to you. This can be on bare metal servers, virtual machines (VMs), public cloud providers, private clouds, and hybrid cloud environments. One of Kubernetes' key advantages is it works on many different kinds of infrastructure.

## VII. DISCUSSION

These Dockerfile and containerization techniques forms the basis of deployment of Docker containers on any platform either on-premises or any public or private cloud platform such Azure Kubernetes Service (AKS), Amazon Elastic Kubernetes Service (Amazon EKS)

I can create an Azure Container Registry (ACR) to store Docker images.

Build the Docker image and tag it with the ACR's login server, image name, and version.

Push the Docker image to the ACR.

Create a Kubernetes deployment file that specifies the Docker image to use and any other desired configurations.

Apply the Kubernetes deployment file to AKS cluster using the kubectl apply command.

These similar steps can be followed to deploy on any cloud Kubernetes platform.

## REFERENCES

Following are the articles and sources helpful to understand Hypervisor, Containerization and Docker technologies.

[1] Estrada, Z. et al. 2014, A performance evaluation of sequence alignment software in virtualized environments, in: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGrid), IEEE, 2014.

[2] Hwang, J., Zeng S. and Wood, T. 2013, A component-based performance comparison of four hypervisors, in: IFIP/IEEE

International Symposium on Integrated Network Management, (IM 2013), IEEE, 2013.

[3] Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P. 2019, Cloud Container Technologies: A State-of-the-Art Review, IEEE Transactions on Cloud Computing, Vol. 7, No. 3.

[4] Soltesz, S. et al. 2007, Container-based operating system virtualization: a scalable, high-performance alternative tohypervisors, SIGOPS Oper. Syst. Rev. 41 (3) 275–287.

[5] Varrette, S. et al. 2013, HPC performance and energy-efficiency of Xen, KVMand VMware hypervisors, in: 25th International Symposium on Computer Architecture and High Performance Computing, (SBAC-PAD), IEEE, 2013.

[6] Abdellatief, E., Abdelbaki, N. 2013, Performance evaluation and comparison of the top market virtualization hypervisors, in: 2013 8th International Conference on Computer Engineering & Systems, (ICCES), IEEE.

[7] Brogi A. et al. 2016, SeaClouds: An Open Reference Architecture for Multi-Cloud Governance. Cham, Switzerland: Springer, pp. 334–338.

[8] Zhanibek Kozhirbayev and Richard O. Sinnott 2017, A performance comparison of container-based technologies for the Cloud, Future Generation Computer Systems 68, pp 175–182.

[9] Subutai Ahmad, "Real-Time Multimodal Integration in a Hierarchical Temporal Memory Network", 2015

[10] Jeff Hawkins and Dileep George, "Hierarchical Temporal Memory", 2009

[11] Jeff Hawkins and Subutai Ahmad, "Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex", 2016