

Cloud Computing

A Paper by Ole Methler

Introduction

The so-called “Cloud” is something that almost every person on this planet has engaged with at some point or another on this planet, due to the ubiquity of the internet in recent years. Be it through the streaming of media through services like Netflix or Amazon Prime, the sending of emails, which has become one of the most common types of communication over long distances, with 306 billion emails sent in the last year, or even through working with the cloud directly with data storage and development over the Cloud.

In this paper I will discuss the origins of the term “Cloud” in computing as well as the origins of the technology behind it. I will also look at two concepts that are utilized among others through cloud computing.

What is Cloud Computing?

Definition

Wikipedia Tells us “Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user”

this definition can be separated into two main parts: The on-demand availability of the cloud, as well as the lack of direct active management by the user. The first defines the Cloud as an active part of the System for which the user intends to use it. This means that the average software that uses cloud services integrates the cloud as if it was a part of the system the software is running on. The second part explains the popularity of the Cloud computing model. The user does not actively manage the resources of the Cloud, he simply gives a task to the cloud and receives a result with which he does further work. Especially for tasks with high requirements of computing power or data storage capabilities, this can be essential for a functioning Software.

Types of Cloud Computing

Cloud Computing can be separated into three main types:

- Infrastructure as a Service
- Platform as a Service

- Software as a Service

Infrastructure as a Service, shortened to IaaS, is the largest of the three types and is defined as “The capability [...] to provision processing, storage, networks, and other fundamental computing resources”. IaaS providers grant access to remote computing power, storage and other resources commonly used as extensions to the user’s system. IaaS is very practical for small companies and single users that may not have the physical space, time, or money to facilitate large servers or databanks they themselves manage. This was (until recently) the main thing that cloud computing was known for. Examples of Infrastructure as a Service are Amazon Web Services, Microsoft Azure or Google Compute Engine

Platform as a Service, shortened to PaaS, is defined as “The capability [...] to deploy onto the Cloud infrastructure consumer-created or acquired applications”. PaaS providers function, as the name suggests, as a platform on which the user can create and run their own applications. This is, again, very practical for small companies and single users, with the added benefit of the platforms usually already having some built-in functionality in the form of APIs, programming environments and other tools. This can also be helpful for learning developers. Examples of Platform as a Service are Heroku, Salesforce or Google App Engine

Software as a Service, shortened to SaaS, is defined as “The capability [...] to use the provider’s applications running on a cloud infrastructure” SaaS providers are everywhere nowadays since the Cloud is ubiquitous as well, though SaaS does not really define the product the user receives, but rather the way in which the product is delivered to the user. SaaS products are usually applications that use the cloud in an integral way. It is not uncommon for SaaS products to also use IaaS or PaaS services. Examples of Software as a Service are Dropbox, GoToMeeting or SAP Concur

The Cloud Computing Metaphor

The Term “Cloud” as a Metaphor to describe external resources in computer systems was originally used in diagrams. The cloud there was a Symbol that represented everything outside of the given system that still interacted with it on a common basis.

The History of Cloud Computing

The early development of what we now know as Cloud Computing started in the 1970s when telecommunications companies used so called “time-sharing”, a concept in which computing resources would be shared amongst users. This

practice was extremely practical for companies since most users would not always need a large amount of computing power, but rather small intervals of it. This led to the construction of data centers in which the more demanding tasks within an organization could be computed while the individual machines of the users did not need a lot of power.

This practice eventually spawned the Amazon Web Services, a project by the e-Commerce platform Amazon that was designed to give customers the opportunity to rent computing power based at the facilities of AWS instead of having to build their own infrastructure. This model was one of the first popular IaaS options. In 2008 Google introduced their Google App Engine, which was one of the first PaaS providers. Microsoft followed in 2010 with Microsoft Azure, a service that provides IaaS as well as PaaS and SaaS.

Today Cloud Computing is one of the most important aspects of computing, mainly because of its role in enabling the surge of startup founding in the 2000s and 2010s, as well as the number of different products that were developed involving Cloud Computing on some level

Advantages and disadvantages of Cloud Computing

Containers

What is a Container?

One of the most interesting ways that Cloud Computing has been realized over the past few years has been through the concept of Containers. Containers are usually small systems that, using a software like Docker or Singularity, are packaged into a singular unit that can be interacted with by a main system. These systems do not have to be interacted with in the cloud, though it is a common practice. Containers are also practical because they are inherently independent from the system, they are run on which makes it possible to construct an entire system from Containers which can be scaled up, modified, and optimized extremely easily.

An easy way to think of a container is as a “Black Box” where the user only needs to know the inputs and outputs of the system, but not the inner

workings of it. This is also where the name “Docker” comes from, as one could imagine containers “docking on” to a system to fulfill their tasks while being separate systems in themselves.

How do you build and use a preexisting Container?

Starting to build containers, the first step is usually to install a software that implements containers. The most popular option for this is Docker, though I will be using Sylabs’ Singularity. Singularity is a service that provides the tools necessary to download, build and run Containers of all kinds and is especially optimized for scientific use.

If the user wants to run a certain software that already exists, they can download it or run it directly from Singularity’s Cloud Library, Docker Hub, or from their local system. The command to do this is **Build**. If a user were to build Ubuntu Linux version 18.04 from the Cloud library, the command to do this would be

```
sudo singularity build library://library/default/ubuntu:18.04
```

This command creates a SIF (Singularity Image Format) file in the singularity directory which can be interacted with later. To Run a container, the command **Run** is used. This can be done with a container on the local system, or even one in the library. The command to run the previously mentioned Ubuntu version would be

```
sudo singularity run library://library/default/ubuntu:18.04
```

Containers are usually interacted with through a command-line-interface within the shell of the system that Singularity is installed on. Other applications that implement containers sometimes use a command line within the application, the Docker application for example, but all can usually be interacted with through the shell.

Most containers that in the Container Library or on Docker Hub have a documentation which details how to use the given container. This should always be read before or alongside using the container.

How do you build and use your own Containers?

To build a Container, all that is needed is the files that will make up the functionality of the Container, (scripts, files, etc.) as well as a definition file that will tell Singularity how the Container works.

The picture on the right shows what the definition file for the Ubuntu-Container may look like.

At the top, the keyword **Bootstrap** is used to determine which platform is used as a Base for the container. In this example a Container from the Container Library is used, marked by the keyword **library**

The sections marked by % correspond to different components of the Container

- %setup
 - Commands executed outside the Container when it is first created
- %files
 - The files needed for the Container
- %environment
 - Certain environment variables that need to be set
- %post
 - Commands executed inside the Container after it has been created
- %runscript
 - A script that is executed when the Container is run via the **run** command
- %startscript
 - A script that is executed when the image of the Container is started via the **instance start** command
- %test
 - A script that is meant to test whether the container is set up as intended at the end of the build process
- %labels
 - A bit of space to show general data like the author or the current version
- %help
 - A bit of space for a description of the Container as well as any information a user would need when using the Container

```
Bootstrap: library
From: ubuntu:18.04

%setup
touch /file1
touch ${SINGULARITY_ROOTFS}/file2

%files
/file1
/file1 /opt

%environment
export LISTEN_PORT=12345
export LC_ALL=C

%post
apt-get update && apt-get install -y netcat
NOW=`date`
echo "export NOW=\`${NOW}\`" >> $SINGULARITY_ENVIRONMENT

%runscript
echo "Container was created $NOW"
echo "Arguments received: $*"
exec echo "$@"

%startscript
nc -lp $LISTEN_PORT

%test
grep -q NAME="\Ubuntu\" /etc/os-release
if [ $? -eq 0 ]; then
echo "Container base is Ubuntu as expected."
else
echo "Container base is not Ubuntu."
fi

%labels
Author d@sylabs.io
Version v0.0.1

%help
This is a demo container used to illustrate a def file that uses all
supported sections.
```

No section must be included, but all of them fulfill a distinct purpose that is important to a working container.

How can we use Containers?

Now that we have established how to create and set up Containers, we can discuss some practical applications for the concept. Containers might be thought of as larger functions that are used in code, in the sense that it is usually completely nonsensical to create one to only use it once, but extremely efficient on multiple usage.

And just as with code, we can package an entire system into smaller containers that interact with each other to create the whole thing again. The advantage of doing this is modularity. With a Container-based system, it is easy to swap out certain parts and put in others when needed or scale the system to a higher degree of efficiency. The disadvantage is usually just the time and effort needed to package the separate parts of a system up and make the whole thing work.

Another example is the publishing and distribution of software through Container based software like Singularity or Docker. While not having found an immense amount of mainstream success, these applications are designed to make it possible to easily publish your own software to other users who use the application, making it incredibly easy to distribute your own programs to others.

One last example on why Containers are incredibly practical also concerns the more casual users of computing technology. While most people all over the planet use Windows as an operating system, most computer science happens on Linux-based systems, creating a barrier of entry for newer users since they can not get their hands on certain software if it is not made for Windows. Containers change that with their total self-reliance. If someone has packaged the technology or application into a Container, it can run on any system and be used by anyone, if the Container-providing Application also exists on that system.

MapReduce

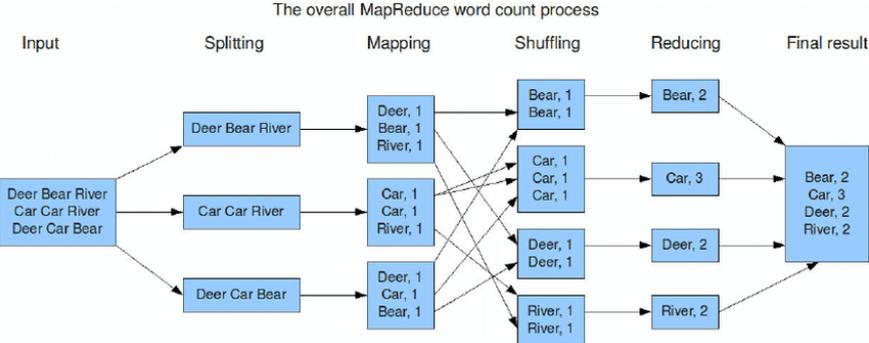
What is MapReduce

One example of such a software would be Hadoop MapReduce, coincidentally another example of a technology that is often used in Cloud Computing. Hadoop is a framework that was developed by Apache as a tool for data Analysis and implements the MapReduce method. Hadoop itself consists of two distinct parts, the MapReduce method, as well as the Hadoop File System (HDFS).

MapReduce is usually used in data analysis to search, organize, filter, or restructure a database through an input. The general idea is that, since vast amounts of data are usually stored in some sort of Cloud database, it would be impractical for a data scientist to process it on their own machine, downloading each chunk of data one by one. Instead, the machine is figuratively brought to the data in the form of the MapReduce algorithm that is implemented in a lot of IaaS services. To use MapReduce, the developer only needs to write a script that implements a few key functions and start the MapReduce function with that script.

MapReduce is generally split into two phases, the Map phase and the Reduce phase, which give MapReduce its name.

The Image on the right shows a wordcount of nine words which are first split into smaller parts, and then mapped, sorting each item into a key-value pair, which is called the Mapping phase. Then the resulting key-value pairs are sorted into clusters of other similar key-value pairs and finally reduced into a result which is useful to the user.



MapReduce Example

To see how the MapReduce method works in practice, we are going to look at an example by the Department of Computer Science and Engineering at the Chinese University of Hong Kong

In this example, we want to count the occurrence of each word in the poem “Youth” by Samuel Ullman. The poem is first brought into the HDFS format to later be processed.

The most important thing to do when using MapReduce is to write the script that will sort through our database. The Script consists in our case of two Java Classes which inherit from the Mapper and Reducer Classes, as well as a Main function. The only thing that these classes need to have is a map or reduce function, which define how the database will later be sorted through and modified. There can also be other methods, variables and code to help the process, but they are not necessary.

In our example, the Mapper only takes one word at a time, and puts them into a key-value pair, as we have seen earlier, with the key being the word and the value being 1. When the Mapper has finished this task, the Reducer takes one key-value pair at a time and adds up all the instances of it in the file, finally counting the result as the value of a single key-value pair.

Now that the script is complete, a few commands need to be executed to run MapReduce.

```
~/Programs/hadoop/sbin/start-all.sh
```

Starts Hadoop so we can use the framework

```
~/Programs/hadoop/bin/hadoop fs -mkdir -p  
/user/bigdata/wordcount/input
```

Creates an input directory for the input file

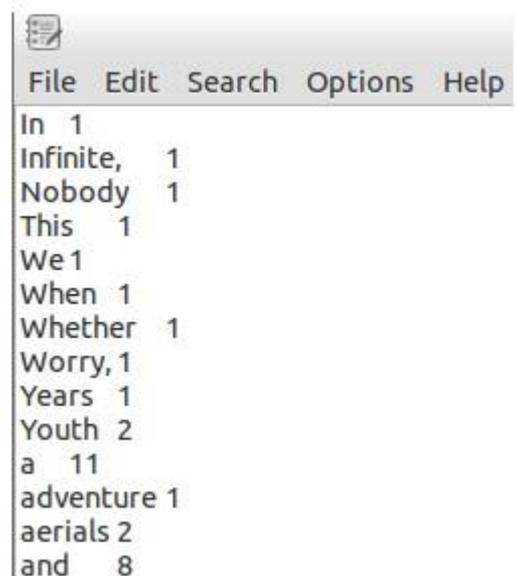
```
~/Programs/hadoop/bin/hadoop fs -put ~/hadoop.txt  
/user/bigdata/wordcount/input
```

Uploads the input text file into the file system, thereby making it accessible to Hadoop

```
~/Programs/hadoop/bin/hadoop jar ~/wordcount.jar  
polyu.bigdata.WordCount /user/bigdata/wordcount/input  
/user/bigdata/wordcount/output
```

Finally executes the code within the Hadoop framework.

After the code has run, the result can be found in a text file. In this example, the use of the MapReduce was not necessary, since a normal application would have also done the job, what Hadoop excels at though is processing vast amounts of data that would be impossible for small scale applications to go through. The separation of the Mapping and Reducing phases also make large amounts of data easier to oversee.



```
File Edit Search Options Help  
In 1  
Infinite, 1  
Nobody 1  
This 1  
We 1  
When 1  
Whether 1  
Worry, 1  
Years 1  
Youth 2  
a 11  
adventure 1  
aerials 2  
and 8
```

The Future of Cloud Computing

Cloud technology for everyone has now existed for a little under 20 years with large developments only happening in the last few years. So, to wrap up this discussion, let's lastly look at some future trends that involve Cloud Computing.

Game Streaming

In November of 2019, Google launched their service "Stadia", which is a SaaS provider specifically for a wide range of games. The idea is that games are streamed directly to the user via the internet, effectively meaning that devices transmit inputs to the provider, which are be processed there. The provider then transmits back the picture data to the user's device. The service was a way to stream games to almost every device, using 5G technology. Its very possible that Game Streaming like the model that Stadia offers will #

Artificial Intelligence

Another area that Cloud Computing will be integrated with a lot is the area of artificial intelligence. Since AI takes up a lot of computing power, a remote access to the AI is very useful. Similarly, to Game Streaming, since the speed of internet connections is becoming faster as time goes, it is conceivable that AIs that already exist might similarly be streamed to user's devices for various purposes.

General Expansion

Cloud Computing is also getting more integrated into areas that usually have nothing to do with it. The store next door might soon have a customer database hosted in the cloud, and most users still have some online backup for their important data. Cloud computing has a very bright future with a lot of opportunities for users and developers.