Comparison of NoSQL on the Cloud Software, Systems and Platforms

J. McFadden
Univ. of Washington: Tacoma
Tacoma, WA
mcfaddja@uw.edu

Y. Tamta
Univ. of Washington: Tacoma
Tacoma, WA
yashaswitamta@gmail.com

J. N. Gandhi
Univ. of Washington: Tacoma
Tacoma, WA
jugalg@uw.edu

April 11, 2017

Project coordinator indicated by *

Abstract

Two different NoSQL database software packages will be implemented in and/or installed on several different types of systems. In turn, these systems will be deployed on several different platforms. This will allow different combinations of software, systems, and platforms to be compared based on both performance, difficulty of setup & maintenance, and both setup & operating costs.

1 Introduction

The software/systems chosen for comparison in this project are two different NoSQL database system. These systems will be deployed/run/operated in several different ways. These include *SaaS*[1] implementations, *containerized* implementations, and *native installations*. The goal of the project is to understand the performance characteristics of each deployment method ***and*** to quantify the costs of each deployment method. These costs will be calculated based on the hourly cost to operate, the initial time & costs required for setup, and the maintenance requirement of a deployment. Additionally, performance of the systems and deployments will be measured using the time required to carry out various database operations, under a set of several different conditions, as well as the CPU, memory, and network loads imposed by the various deployments under the same set of conditions.

2 Systems and Platforms

We will be using two NoSQL database software packages. The first software package is **DynamoDB** from Amazon Web Services (*AWS*), while the second software package will be **Cassandra**, an open-source NoSQL database software package. These software packages will be deployed using several different systems and platforms, as described below.

2.1 Systems

This project will run the software packages on three different systems (*or types of systems*). We have chosen systems which range from hosted *SaaS* through various degrees of virtualization and then all the way to non-virtualized machines. These systems are as follows

A): ***SaaS***

B): **Running inside a *Docker* container**

C): **Running on a dedicated machine**

[1] **SaaS :** Software as a service.

These four systems will be deployed using several different platforms which we will describe in the next part of this section; however, before proceeding to that, we will give a through description of each system listed above.

2.1.1 ***SaaS*** **(code: A)**

In this system paradigm, a particular software package is developed and maintained with the intent of being used by a provider as selling *Software as a Service*. As far as system configuration on our end (*the system-administrator/developer*), paradigm only requires that we select a platform provider and collected the database data so that it can eventually be prepared for deployment.

2.1.2 **Running inside a *Docker* container (code: B)**

For the purposes for this paper and the ensuing project, we will only use **containers** which are created using and for use in *Docker*. This choice was made because of *Docker*'s ubiquity, wide availability, and the ease of access to cloud services designed for and solely dedicated to the deployment of *Docker* containers. This implementation paradigm requires that a *Docker* container be created after which, the software package in question be installed into the container so that it can be run on the operating system employed in the container (*the OS*[2] *was chosen at the time of container creation*). Once the software has been installed, any information for and/or data required by it is either loaded into the container or, in the case of data *only*, the container is "pointed at" the location of the data required by the software. Finally, we must note that, as far as this paper and project are concerned, the use of a *Docker* container as a means of implementation does not imply the use of any particular platform for running the containers, or even a deployment of the *Docker* software/system itself.

2.1.3 **Running on a dedicated machine (code: C)**

The third system paradigm involves acquiring and configuring a dedicated machine which can be either an actual, physical machine or a virtual machine. In either case, the machine will have

[2] **OS:** Operating System

performance specifications and and run an OS appropriate for our planed use. When setting up one of these machines, after configuration and setup of physical/virtual system, the selected OS will be installed and configured for use. Once it has been verified that the OS has been properly installed and configured (*i.e. running stably*) we will install the software selected for testing on that instance, along with any other software or system components required to run that software. After configuring the installed software, the machine will be prepared and configured for use as a server running said software. The final step of this paradigm is preparing the software and system combination for final deployment to the selected platform.

2.2 Platforms

We have chosen four different platforms on which to deploy our systems. The chosen platforms span the range of cloud service paradigms from *SaaS* to *PaaS*[3] to *IaaS*[4]. We list the three platforms, along with two variations on one of the platforms, below

1): **(An) AWS Software Service** (*SaaS*)

2): **Containers** (using *Docker*) **running on**

- **i**): ***AWS's Container Service*** (*PaaS*)
- **ii**): ***AWS EC2 VM with Docker run-time*** (*hybrid-Pass/IaaS*)

3): **AWS EC2 VMs running the software in Linux** (*IaaS*)

4): **A dedicated, non-virtualized server** (*server*)

In the next section, we list which systems will run each software package, along with a explanation why each software-system pairing was chosen. Additionally, we will describe which platforms will be used to deploy each system and why those deployment choices were made. Before proceeding to that, and as with our list of system, we will describe each platform thoroughly.

[3] **PaaS :** Platform as a service.
[4] **IaaS :** Infrastructure as a service.

2.2.1 AWS Software Service (code: 1)

Our first platform we will use for the deployment of some systems is AWS's Software Service platform which provides various software packages which run as a service. This means that Administrator do not have to configure and secure a server and OS on which to run the software, nor do they have to install and manage the running of the software on said server. Instead, all a System Administrator must do is create an instance of the desired Software Service, with the configuration required for the application at hand. Once the instance has been created, it is loaded with any information required and/or data to be used, after which it is immediately available for use by users and other clients.

2.2.2 Containers (code: 2_)

[5]

The next platform we have chosen involves the deployment of the previously described *Docker* containers. We will use two different platforms that implement the *Docker* engine which is actually responsible for running the containers. These "*Docker* Platforms" are

- [*AWS's Container Service (code:* **2i***)*]**:** In this version of the second platform, we will deploy our containers by using AWS's Container Service. When deploying to this platform, an Administrator first creates an appropriately configured instance of the Container Service. Upon the successful creation and launch of the Container Service instance in question, the Administrator simply uploads the container being deployed to the instance and, if a data-source external to the container is required, uploads the required data to the location specified during the creation of the container. Once the container has been uploaded to the instance (*and any required data has been uploaded to the appropriate location*), the Administrator tests the instance to ensure it is functioning as expected. In the last step, the Administrator configures the instance to properly connect with users and/or clients before finally granting them access to the deployed container.

[5] In the code **2**_, we use the "_" after the 2 is a place-holder for an additional code-letter. This is due to the fact that two different versions of and approaches to this platform (*container deployment*) are going to be used.

- [*AWS EC2 VM with Docker run-time (code:* **2ii**)]**:** The other version of the second deployment platform involves deploying our containers using a *Docker* run-time/engine running in a Linux machine being hosted on AWS's EC2 Virtualization Platform. To deploy via this platform, an Administrator begins by creating an EC2 VM instance with performance characteristics necessary for running the container inside the Linux version of *Docker* run-time. Once the EC2 VM instance has been created and successfully launched, the Administrator will install and configure an appropriate version of Linux, followed by installing the *Docker* run-time and any required support software (*such as software for accessing remotely stored data*). After configuring the installation of the *Docker* run-time (*and any support software*), the Administrator will upload the container using *Docker*. Additionally, if a data-source external to the container is employed, up to two additional steps are required and based on whether the data is local or remote to the VM instance. In the case the data-source is local to the VM instance, the Administrator must also configure appropriate storage on the instance, including installing and configuring any necessary software, after which the required data is uploaded to the storage location. Similarly, if the data-source is external to the VM instance, the Administrator must also install and configure the software required for accessing the remote location (*NFS, SAN, Samba, EBS, etc*), followed by configuring the system to access the remote location and the data stored within. Once these steps are complete, the Administrator will be able configure the container to run on the VM instance, as well as configure the OS to allow the container to properly run. These configurations involve ensuring that the container has access to the necessary networks, addresses, and ports and that any local and/or remote resources are both appropriately mounted to the container (*container configuration*) and accessible to the container (*OS configuration*). With these configurations complete, the Administrator is able to launch the container and begin testing it to ensure it functions as expected. Once the container has been determined to be properly functioning, the Administrator

completes any configuration of the EC2 Instance, the VM itself (*OS*), and the container itself which might be necessary for proper user and/or client access of the container. This final configuration is followed, finally, by allowing users and clients to access the container.

2.2.3 AWS EC2 VMs running the software in Linux (code: 3)

Our third deployment platform paradigm also relies on use of AWS's EC2 VM service; however, instead of installing using containers as in **2ii**, this paradigm has the software to be tested natively and directly installed on a Linux OS being run on an AWS EC2 VM instance. As with the previous paradigm involving an AWS EC2 VM the Administrator begins by creating an EC2 VM instance, launching it, and installing an appropriate version of the Linux operating system, with the only difference being the determination of necessary performance characteristics base on natively running the software, and any supporting software, in Linux. Following the installation and configuration of the Linux OS, the Administrator will natively install and then configure the software to be tested, along with any other software required support software (*i.e. remote data access*). These installations are followed by creating and configuring any required local storage space on the instance, to include installing and configuring any software necessary for doing this. In the case where remote storage be used instead of or in addition to local storage, the Administrator must also install and configure the software required for accessing the remote location, followed by configuring the system to access the remote location and the data stored within. Once these data storage/access steps have been completed, the Administrator uploads the required data to the appropriate location or locations, after which they configure both the software being tested, so that it can locate these data-sources, and the OS, so the software can access both the data-sources and any required networks, services, or ports. As before, this configuration step is followed first by testing to ensure the software is accessible and works as desired, and then by the final step, allowing users and clients to access the container.

2.2.4 A dedicated, non-virtualized server (code: 4)

The last deployment platform we have chosen is a dedicated, non-virtualized server. These servers will natively run Linux as their OS and will be dedicated to running and serving the software to be tested and will be setup in a manner similar to that used for the AWS EC2 instances described in **3**. In fact the only differences between the setup process for these physical machines and the setup of the EC2 VMs are the elimination of the need to create and configure an EC2 VM instance and the added requirement to communicate with the administrator of the network on which the machines are to be deployed. This second difference in the setup process arises because we do not have the same control over the network connections for these physical machines as we do with AWS VMs.

3 Deployment

The deployment strategy was developed with the goal of maximizing our ability to compare combinations of software, systems, and platforms. Maximizing our choices for comparison was important because it allowed us to choose combinations which could be effectively measured and where our findings would have "real-world" applications and meaning. We have structured the description of our deployment strategy using the software package being tested as the over-arching descriptor of they deployments (*i.e. breaking all deployment down along the lines of which software package is a particular deployment testing*). This allows us to simplify the entire deployment strategy into a single table for easy viewing. We begin by describing the systems on which we have decided to run *DynamoDB*, which are as follows:

- **AWS's DynamoDB Service:** a *SaaS* approach using AWS's DynamoDB SaaS Cloud Service.
- **A *Docker* container on AWS's Container Service:** a *PaaS* approach using AWS's Container Cloud Service.
- **A *Docker* container on an AWS EC2 VM:** a *hybrid-Pass/IaaS* approach using AWS's EC2 VM Cloud Service.

Continuing, we now describe the systems on which we have decided to run *Cassandra*. These systems are:

- **A *Docker* container on AWS's Container Service:** a *PaaS* approach using AWS's Container Cloud Service.
- **A *Docker* container on an AWS EC2 VM:** a *hybrid-Pass/IaaS* approach using AWS's EC2 VM Cloud Service.
- **A native installation in Linux on an AWS EC2 VM:** an *IaaS* approach using AWS's EC2 VM Cloud Service.

Finally, we have the previously mentioned table.

Cloud Paradigm	**DynamoDB**	**Cassandra**
SaaS	AWS's DynamoDB SaaS Cloud Service	
PaaS	*Docker* container on AWS's Container Service	*Docker* container on AWS's Container Service
hybrid-PaaS/IaaS	A *Docker* container on an AWS EC2 VM instance	A *Docker* container on an AWS EC2 VM on an AWS EC2 VM instance
IaaS		Native installation in Linux on an AWS EC2 VM instance

4 Metrics

We will evaluate each of the software/system/platform combinations based on three metrics. The first two of these metrics are based on performance data which will use data collected during the operation of each of these combinations. Each element of performance data will be collected based on carrying out a specific database operation, while the system is under a given load condition. We will use several different types of database operations (*i.e. insert, update, etc.*) when running these tests. Additionally, each database operation will be run in several different ways, with each having a different level of complexity. Finally, the third and last metric will be based on the cost

to setup, run, and maintain the software, system, and platform used to create and deploy a given combination.