Deployment of Vehicle Detection application on Chameleon clouds

ABHISHEK NAIK[1],* **AND SHREE GOVIND MISHRA**[2],*

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[2] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
*Corresponding authors: absnaik810@gmail.com, shremish@indiana.edu

---

This project focuses on the deployment of Vehicle Detection application on multiple Chameleon clouds using Ansible playbook. It also focuses on the benchmarking of the deployment results and its analysis.

Keywords: Vehicle detection, Ansible, Cloudmesh, OpenCV, Haar Cascades, Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P010/report.pdf

---

INTRODUCTION

Vehicle Detection forms an integral part of the development of new technologies like fully self-driving cars, etc. One of the techniques to perform such detection is by using Haar Cascades [1]. This technique has been applied to vehicle detection by creating a haar-cascade cars.xml file which has been trained using 526 rear-end images of cars. In this project, we would be extending this vehicle detection approach to enable it to run on multiple clouds. This deployment would initially be done on the localhost, followed by deployment on the cloud. Cloudmesh client would be used for cloud management and Ansible scripts would be used for software stack deployment. Appropriate benchmarking would be carried out at each iteration using some benchmarking technique.

SOFTWARE STACK

For this project, we have used the following softwares/applications:

- Ansible [2] Ansible is an open source platform that is used for automation. Configuration management, task automation and application deployment can be carried out using Ansible. In our project, we have used Ansible to manage the configuration and automate the deployment of the software stack onto the clouds. We run Ansible scripts via localhost entering the IP addresses of the cloud's virtual machines (VMs) and the softwares/applications are installed in those VMs. Thus, we are using Ansible to orchestrate the software deployment via playbooks using the inventory.txt file containing the IP addresses.

- Cloudmesh client [3] Cloudmesh is basically used for cloud VM management. It provides easy access to different cloud environments via a command shell and command line. In our project, we have used Cloudmesh client for dynamic management of the cloud VMs. Thus, we have used it to add our SSH keys to a local database and then SSH to the remote VMs on the clouds using various cloudmesh client commands and utilities.

- Python [4] Python is an object oriented, high level programming language. It is an interpreted language and provides built-in data structures with dynamic typing and binding. It has an easy and simple yet intuitive syntax. In our project, we are using Python (and Pip [5]) for the installation and management of softwares written in Python.

- Git [6] Git is a version control system that can be used for tracking and change management when multiple people are involved in a project. It helps control and coordinate the working among a group of people. We have used Git to manage our work within our team. It has helped us by providing a central place wherein we could commit our work and make it easily accessible to others. Also, the Vehicle Detection Application (outlined below) application that we are using in our project has been developed and hosted on Github by the creator Andrews Sobral. We have forked his repository and using his application for the detection of vehicles.

- OpenCV [7] OpenCV is a collection (library) of various functions used to execute computer vision related applications. The Vehicle Detection Application runs on OpenCV. OpenCV includes both, a trainer as well as a detector. It also has many pre-trained classifiers. We are using the Haar classifier [1].

- Vehicle Detection Application [8] The Vehicle Detection

Application has been developed by Andrews Sobral. It basically uses a haar-cascade cars.xml for vehicle detection. The haar-cascade was trained using 526 rear-end images of the cars. The video size used is 360x240 pixels. In our project, we are running this application on the clouds.

DEPLOYMENT

For successful project deployment, the first step was gaining access to the clouds. Cloudmesh client was used for gaining this access. In order to set up this cloudmesh client, we had to configure the cloudmesh.yml file with our credentials and other details. We mainly used Chameleon clouds in our project so we only edited the cloudmesh.yml file parts that corresponded to chameleon clouds. If we were to use Jetstream, we would have required to edit the Jetstream part as well. We had to edit the values in such a way as to ensure that the info command did not yield any (To Be Decided) TBD values. In case it did, then it meant that there were possibly some errors and we had to revisit the cloudmesh.yml file and correct those. If there were no TBD values at all, then it meant that we were good and could now access the clouds. Once we had access to the clouds, we booted a VM and then assigned a floating IP to it. Post this, we used the SSH command to log in into the VM. Once we were within the VM, we could use it as a normal local Ubuntu machine. Although we didn't require it in this project, cloudmesh also provided the functionality to use a different image on the cloud VMs.

When we need a new VM, we simply booted a new VM and assigned a new floating IP address to it. This resulted in allocation of a new VM which we could then use as a new additionL machine. By proper set up in the Ansible scripts, one of these machines could be used as a master and the other one(s) as slaves, if there is a need in some project.

The second step was the identification of the software stack that would be required. In order to understand this, we first deployed the softwares and application on the local machines and then on the clouds. This step helped us understand the softwares needed as well as the dependencies amongst them. Once we identified the softwares, we developed an ansible script to deploy these softwares and applications dynamically in an automated way.

Ansible is an open source automation platform. It mainly uses .yml files for its working. The file 'inventory.txt' contains the details of the hosts (their IP addresses) wherein the software stack is to be installed. Similarly, we can also mention the ansible username in this file. Thus, in our case, the entry in the inventory.txt file is as below:

```
129.114.110.83 ansible_ssh_user=cc
```

The 'playbook.yml' file lists the hosts, variables and the roles. The hosts contains the details of the machine where the softwares are to be installed. The variables section lists the variables that are being used elsewhere, for e.g., 'dwnld_dr' denoting the download directory. The advantage of using variables is that the values do not need to be hard coded and thus they can be changed everywhere with a minor update. The roles section lists the various softwares that need to be installed, in order. In our case, the entry of the playbook.yml file is as below:

```
---
- hosts: all
    vars:
```

```
dwnld_dr: ~/downloads}
ocv_ver: 2.4.13.2}
repository: {{ repository }}
tmp: /tmp/vehicledetection
    roles:
- git
- python
- upgrade
- opencv
- vehicledetection
```

The 'roles' directory contains the details about all the softwares that are to be installed. Since we have installed four softwares, we have four directories within it. Each of these four directories is named after the software that it is supposed to install. Thus, the directory 'git' installs git, 'python' installs python and so on. Each of these directories in turn contain two directories - 'defaults' and 'tasks'. The 'defaults' directory contains a file 'main.yml' that lists the temporary variables like the download directory to be used, the version to be downloaded, etc that is specific to its software. The 'tasks' directory contains a file named 'main.yml' that lists the tasks (i.e., the activities) that need to be carried out step-by-step. These activities are executed in a sequential order. A snapshot of one of the main.yml files used in our project is as below:

```
---
- name: install Git on Ubuntu machine
  become: yes
  apt: name=git state=present
```

Thus, the first statement says that we are installing Git on the Ubuntu. The next statement says that sudo privileges would be required for installation of Git. The last statement specifies the package name and its state. Ansible thus enabled us to dynamically deploy the software stack and configure the system as required for proper installation.

EXECUTION

For the execution of this project, we had set the tasks as below:

Week 1

In the first week, we deployed the vehicle detection application on the localhost by using bare commands. The main aim of this step was to ensure that the application worked. This step was critical in the sense that it helped us understand the various dependencies among the various softwares in the software stack. It also helped us understand the environmental variables like OpenCV_FOUND and OpenCV_DIR that we had to setup, since they are specific to the local system on which the application is run. This step in reality took us more than a week to find out about OpenCV_DIR, OpenCV_FOUND and their specific expected values. Once this step was executed successfully, we wrote down an Ansible script to carry out the software deployment. Since we spent substantial amount of time in debugging the OpenCV_FOUND and OpenCV_DIR errors, we could not write the scripts for the entire software stack. We wrote it only for Git; but it provide us with a good start.

Week 2

In this week, our aim was to reserve and access a Chameleon cloud using cloudmesh client. We then planned to carry out software stack deployment on it, and carry out benchmarking. However, we spent more than four days in trying to debug the

`OpenCV_DIR` and `OpenCV_FOUND` errors that we encountered in the first week. Nonetheless, we found out a solution and tested that the application works as expected. As a consequence of the error we faced, we could not spend much time in developing ansible scripts for further software stack deployment. However, by the end of second week, we had configured our cloudmesh.yml file. We were yet to test it, though.

Week 3

For week 3, we tested our configuration of the cloudmesh.yml file by actually gaining access to the Chameleon clouds. For this, we learned about the various commands that cloudmesh had to offer and booted up a VM in the clouds. Later, we assigned a floating IP address to it and then logged in into the remote VM via SSH. Once we were into the VM, we could operate on it just like a local normal VM. We spent the next couple of days in writing down ansible scripts for software stack deployment. Once this was complete, we ran those scripts on the local host to test if they worked as expected, or if they needed any changes.

Week 4

In this week, we carried out minor changes to the ansible scripts that we had written. We booted up a cloud VM using cloudmesh client and logged in into it. We then ran the ansible scripts from our local hosts by entering the floating IP of our cloud VMs in the inventory.txt file. While the scripts for git ran successfully, the script for OpenCV failed. After much debugging and testing, we found a solution which involved updating the cache (running an equivalent of sudo apt-get update) which resulted in OpenCV being installed successfully on the clouds. We faced another challenge when cloning the Vehicle detection project from github. We circumvented this problem by making the ansible script run the command as a sudo user.

Week 5

In the final fifth week, we aimed to carry out benchmarking and write a report about our observations. We first carried out benchmarking by deploying the software stack on the local machine and taking the readings. Then, we deployed it onto the Chameleon clouds and noted down the observations. Finally we noted down all these observations in a detailed report.

BENCHMARKING RESULTS AND ANALYSIS

For benchmarking, we first deployed the software stack onto the local machine and then deployed it on the clouds. We noted down the observations in each case. The parameters for benchmarking as well as the observed values have been noted below. In order to reduce any errors, we have considered average values. Thus, we installed each software (from scratch) 3 times on the local machine and 3 times on the clouds. Then, we took the average of all these values in order to find out the exact time needed for its deployment on the platform.

Deployment benchmarking

Deployment benchmarking included observing the software deployment times for each of the softwares in the software deployment stack. We started off with the first software, Git. On the local machine, it was installed in 2.567 seconds, while it took 10.447 seconds to be installed on the clouds. The next command, upgrade, took 35.837 seconds on the local machine and 155.03 seconds on the clouds. Python took 4.554 seconds to be installed on the local machine and 30.748 seconds to be

Table 1. Deployment Benchmark

	Local VM avg time	Cloud VM avg time
Git	2.5863333333	10.494
Upgrade	35.531	155.5903333333
Python	4.2873333333	30.5766666667
OpenCV	277.032	1419.5266666667
Vehicle Detection	2.1833333333	11.7166666667

Table 2. Elasticity benchmark

Software/Application	Dependency count
Git	0
Upgrade	NA
Python	4
OpenCV	14
Vehicle Detection	3

installed on the clouds. OpenCV, is a heavy software having lots of dependencies on other softwares. We thus had to install it dependencies first before we could compile and install it. Due to all this, OpenCV took 1420.373 to be installed on the clouds and 276.014 seconds on the local machine. Vehicle detection is a lightweight application and as such, it took 11.987 to install on the clouds and just 2.04 seconds to be installed on the local machine. All these values have been tabulated in Table 1.

Elasticity benchmarking

Elasticity metric aims to test the dependency of one software on the other ones. In other words, it aims to test a software's sensitivity to changes in another software. In our project, we noticed that some of the softwares had a high dependency (and sensitivity), while others had relatively low (or no) dependency. For benchmarking the software elasticity, we referred its ansible script. We counted the number of tasks that had to be successfully executed before the software in question could be considered to be successfully installed. For e.g., in case of the Vehicle Detection application, before running the 'make' command, we need to carry out 3 steps - getting the Vehicle detection package source, changing the directory and running cmake. Thus, the elasticity factor associated with the Vehicle Detection project is 3. We carried out a similar analysis of the other softwares from the software stack. Table 2 lists these observations.

Re-installation benchmarking

Ansible scripts do a good job of installing the softwares. However, even before the softwares/applications are installed, ansible first tests to make sure that the software is not already installed on the system. In case it is, then it simply skips over the installation process of that software and then continues with the next one. This is important, since it does not try to update/re-install the existing softwares since that would lead to conflicts with the existing package versions or features. Also, since the re-installation of packages is just skipped, in case of failures, there is no need to edit the inventory.txt file to deploy only the

failed software - it can be kept as it is and the ansible scripts can be run from scratch. Only the failed software would be installed. This also helps in reducing the installation time.

Workload benchmarking

For Workload benchmarking, we found out the disk utilization of the various softwares from the software stack. While softwares such as Git were pretty light, others like OpenCV were pretty heavy. They not only required a long time to install, but also consumed a lot of disk space. Figures 1 and 2 denote the output of the 'df -h' command on the cloud VMs. As shown in Figure 1, we had used only 1.3 GB of system space before installing OpenCV while we ended up consuming 4.4 GB of it after its installation. The usage thus increased from 7 percent to 24 percent.

Fig. 1. Usage statics before OpenCV installation

Fig. 2. Usage statics after OpenCV installation

Security and trust factors

In our project we have used cloudmesh client which in turn uses the SSH protocol to make secure connections to the VMs on the clouds [9]. SSH is a cryptographic network protocol used for establishing secured connections over the network. It is thus used for access to shell accounts on Unix like operating systems. Since SSH is being used, trustworthy connections can be established between the local machine and the remote VMs on the clouds. Along with this, Ansible also helps in security enforcement. Some of the qualities like being agentless, capability to support SSH, no unnecessary changes, modular structure, etc [10].

Latency and reliability benchmarking

We used the ping facility for benchmarking the latency [11] of the network. Latency measures the time it takes for a packet to reach the cloud VM (with a specific IP) from the local host. We used the following command to test this:

```
ping 129.114.33.88 -c 10
```

In this command, the '-c 10' option makes 10 requests to the cloud VMs. In the end, it displays a summary of the results. We can thus observe that the minimum round trip time (rtt) was 40.499 seconds, while the maximum was 72.198. Along with all this, we also noticed that there was 10 percent packet loss, since out of the 10 packets that were sent, only 9 of them could make it back to the local host. However, repeated experiments showed that the system was indeed reliable and that this was a small exception.

Fig. 3. Latency Test results

CONCLUSION

In this project, we used cloudmesh client which is a client that enables us to access and manage various cloud environments. We used it to gain access to Chameleon cloud VMs by using the SSH protocol. Ansible is an automation platform that helps us in automating the software deployment. We used Ansible scripts to install all our softwares along with the configuration of the project. Once the software stack was installed, we installed the Vehicle detection application on top of it. The Vehicle detection application running on OpenCV used haar-classifiers to detect the vehicles. Once this was done, we did benchmarking of the system. We typically focused on the deployment, elasticity, installation capacity, workload, security, latency and reliability.

ACKNOWLEDGEMENTS

REFERENCES

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. Cambridge, MA: Institute of Electrical and Electronics Engineers (IEEE), Dec. 2001. [Online]. Available: http://wearables.cc.gatech.edu/paper_of_week/viola01rapid.pdf

[2] Wikipedia, "Ansible (software)," Web page, online; accessed 10-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Ansible_(software)

[3] G. von Laszewski, "Cloudmesh/client," Code Repository, accessed: 2017-3-10. [Online]. Available: https://github.com/cloudmesh/client

[4] Python Software Foundation, "What is python? executive summary," Web page, online; accessed 21-Mar-2017. [Online]. Available: https://www.python.org/doc/essays/blurb/

[5] Wikipedia, "pip (package manager)," Web page, online; accessed 20-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Pip_ (package_manager)

[6] Wikipedia, "Git," Web page, online; accessed 19-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Git

[7] Wikipedia, "Opencv," Web page, online; accessed 19-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenCV

[8] A. Sobral, "Vehicle detection by haar cascades with opencv," Code Repository, accessed: 2017-3-10. [Online]. Available: https://github.com/andrewssobral/vehicle_detection_haarcascades

[9] Wikipedia, "Secure shell," Web page, online; accessed 18-Mar-2017. [Online]. Available: https://en.wikipedia.org/wiki/Secure_Shell

[10] Red Hat, Inc., "The inside playbook," Web page, online; accessed 21-Mar-2017. [Online]. Available: https://www.ansible.com/ blog/security-automation

[11] Red Hat, Inc., "The inside playbook," Web page, online; accessed 21-Mar-2017. [Online]. Available: https://www.ansible.com/ blog/security-automation