Study of Ansible as an automation tool for Site Reliability

CSI ZG628T: Dissertation

by

Baskar Balasubramanian

2019HT66015

Dissertation work carried out at: IBM India, Chennai

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)

April 2021

Birla Institute of Technology & Science, Pilani
Work-Integrated Learning Programmes Division
Second Semester 2020-2021

CSI ZG628T : Dissertation Pre-Final Report

ID No.	**: 2019HT66015**
NAME OF THE STUDENT	**:** BASKAR BALASUBRAMANIAN
EMAIL ADDRESS	**:** 2019HT66015@wilp.bits-pilani.ac.in
STUDENT'S EMPLOYING ORGANIZATION & LOCATION	**:** IBM India
SUPERVISOR	**:** Swetha J, SAP Technical Architect, IBM
SUPERVISOR'S EMPLOYING ORGANIZATION & LOCATION	**:** IBM India, Chennai
SUPERVISOR'S EMAIL ADDRESS	**:** swethaj.iyer@in.ibm.com
ADDITIONAL EXAMINER	**:** Raghu Srinivasan, Senior Technical Staff Member - Technology Architect Lead Client Transformation SRE

ADDITIONAL EXAMINER'S EMAIL ADDRESS: sriniv@us.ibm.com

BITS FACULTY	**:** Rajesh Kumar, Assistant Professor, Dept of Computer Science
BITS FACULTY EMAIL ADDRESS	**:** rajesh.k@pilani.bits-pilani.ac.in
DISSERTATION TITLE	**:** Study of Ansible as an automation tool for Site Reliability

Table of Contents

1. ABSTRACT

Reliability of a website is a quality that is measured based on user experience. It appears to be simple phenomenon from the outside, but it takes constant effort in steering the moving parts of the systems and software behind the scene. Automation, monitoring and alerting of the system functions and effective communication among the systems and support personnel are few of the foundational requirements that attempts to solve the problem of site reliability.

The choice of tools used to meet the above requirements becomes crucial to deploy such an eco-system. In this dissertation work, there is an attempt to study Ansible, a Python module, as the Automation tool and how it helps to eliminate the toil [4] due to tasks that are manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows. The work also involves studying the interfacing capabilities of ansible with tools that helps in monitoring the system metrics and visualize the data captured for analysis. The monitoring tool under consideration is Prometheus while the visualization tool considered is Grafana.

The use-case for the project involves automating the deployment of docker container, running a TCP Echo Server, hosted on top of a Linux Virtual Machine. The operations of Linux VM, Docker container and TCP Echo Server are all automated using Ansible as the automation tool. A TCP Echo client runs anywhere within or outside Linux VM, can be used to send TCP echo messages to the TCP Server, to test the application functionality.

This effort demonstrates the value of automating the repeatable task of deploying the docker containers with a hosted server, in a large scale environment, such as a cloud data center. The use-case also demonstrates the value of using monitoring and analytics tools such as Prometheus and Grafana, respectively, that can interface with Ansible in the deployed environment. The scope of system parameters captured for monitoring and analysis with the tools is restricted to the TCP statistics such as the number of connections, state of the connections, etc. There will be a detailed analysis on how Ansible automation combined with monitoring and analytics of the system, can address the problem of site reliability.

The extensibility of the project is such that, TCP Echo Server can be replaced with any other server like web servers, application servers, etc during study. Having the server as TCP Echo Server can be of much use for students with academic interest on the study. It can be shown that it is simple to develop, modify, build the code base for the TCP Echo Server and monitor the system characteristics that are fundamental to study any server that run over a TCP protocol.

On the other hand, the time needed to build, develop or extend web server or application servers, with voluminous code base is time consuming. In this case, complex system characteristics involving application layer protocols like HTTP, TLS, SOAP, etc are generally considered for study. This study with TCP Echo Server is analogous to study of the implementation of Data Encryption Standard (DES) in field of Cryptography, the fundamental block cipher algorithm, before studying the more complex algorithms like Advanced Encryption Standard. With this approach, the dissertation gives a way for a wide range of options for future study by keeping this project as the reference study on Site Reliability of web sites.

2. INTRODUCTION

2.1 BACKGROUND

2.1.1 SITE RELIABILITY

In the current world, there are innumerable number of websites and applications that are hosted in remote computers and accessed by users across the world through Internet that forms a communication medium and backbone of the computer network. A **Site** can be defined as any useful application or software available for use over computer networks which is accessible over the Internet or private interconnected networks.

Users experience in using the sites to access application is based on various parameters, which makes the user to make repeatable use of the websites to realize the benefits of the application. Users have various expectations which inclue the site to be available whenever they want to access, cater to the need of the user irrespective of the number of concurrent users using the system, irrespective of the geography of the server hosting the application, tolerant response times, etc.

The above parameters are measured in terms of the site's,

- availability (how much time the application is available for use?)
- scalability (how flexible is the system when there is a need to address an increase in the number of users or resource requirements?)
- recoverability (how quickly the system can recover from a failure?)
- maintainability (how effectively application changes can be incorporated?)
- security (What is the level confidentiality and integrity that the system provides to user's data within the systems and the network?).
- elasticity (how robust the system responds to sudden surge or drop in the processing load?).
- economic value (what is the cost savings for the IT service provider?).

From the perspective of the service provider, they would want to ensure they are able to address all the above expectations and many more, to provide the best experience to the end users. In the same way, users of the sites would fall back to the websites that are able to meet their expectations. Such sites are reliable from user experience perspective, which is the primary goal for anyone providing information services. If such reliable services are realized with the hosted websites, then the sites are meant to have an added quality called as **Site Reliability**. The art of practicing the principles to meet the expectations from reliability perspective can be named as **Site Reliability Engineering**.

The following are considered as some of the important principles of Site Reliability engineering

- **Automation** of tasks that are manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows. In IT industry terms this could be called as *eliminating the toil or backlogs*.

- **Measurement and Interpretation of the system data** which is essential in a system that automatically adjusts its resources and configurations, there by meet the demands of the end users. This could be termed as the *Observability* principle in Site Reliability.

- **Alerting** the support personnel and experts and **effective communication** among them, about the system malfunctioning and take corrective actions for speedy recovery. This functionality is normally categorized as *Event Management or Incident Management* based on the severity of the issue.

In order to practice the above, service providers have to choose the tools wisely, that could be integrated and interfaced with the core systems that serves the user requests. For example there are **automation tools** like *ansible, chef, puppet, etc* that can automate complex IT system tasks with simple yet feature rich modules. There are open source tools like *Prometheus, Logstash, etc* that can **capture the metrics** captured from the functioning system and other open source tools like *Grafana, Kibana, etc* that can **interpret the data** captured as useful information for analysis.

2.2 INTRODUCTION TO THE TECHNOLOGICAL TERMS

2.2.1 OPERATING SYSTEMS TERMS

2.2.1.1 Linux and it's flavors

Linux is a free software built and ported as the operating system binary for various processor architectures.
Initial version of Linux was developed by Linus Torvalds. Linux is one of the largest free software projects actively developed by the Linux community. [5]

Linux System Base (LSB) comprises of kernel and shell utilities that forms the core of linux operting system. There are various flavors of Linux developed on top of the LSB to give added features to the Linux OS. Ubuntu, RedHat, openSUSE, debian are well known Linux flavors to name a few. In this project, redhat flavour of Linux has been used for host machine as well as the virtual machine that hosts the docker container.

2.2.1.2 Virtualization

Virtualization of operating system means emulating operating systems to share the computing resources like processor, memory, storage, I/O system and networks. Virtualization could be either directly done over the hardware using the drivers named as hypervisors or emulated over a host operating system which manages the system resources.

2.2.1.3 libvirt

libvirt is a tool kit comprising of set of operating system libraries for virtualization management system. There are server side and client side components to it. The server side component manages the virtualized guest operating systems by starting, stopping, pausing, unpausing the virtual OS. The client libraries and utilities that connect to the server side component to issue tasks and collect information about the configuration and resources of the host system and guests. [6][7]

2.2.1.4 KVM

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc

KVM is open source software. The kernel component of KVM is included in mainline Linux while The userspace component of KVM is included in mainline QEMU. [8]

2.2.1.5 QEMU

QEMU is a generic and open source machine emulator and virtualizer. When used as a machine emulator, QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC). When used as a virtualizer, QEMU achieves near native performance by executing the guest code directly on the host CPU. QEMU supports virtualization when executing under the Xen hypervisor or using the KVM kernel module in Linux. [9]

Note: In this project, a RHEL OS is virtualized using qemu-kvm emulator. **qemu-kvm** is an open source virtualizer that provides hardware emulation for the KVM hypervisor. qemu-kvm acts as a virtual machine monitor together with the KVM kernel modules, and emulates the hardware for a full system such as a PC and its associated peripherals. [10]

2.2.1.6 docker

Docker is written in the Go Programming Language, it takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called namespaces to provide the isolated workspace, which is called as the container. Docker provides the ability to package and run an application in a loosely isolated container environment. A container is a runnable instance of an image, while image is a read-only template with instructions for creating a Docker container. [20]

2.2.2 COMPUTER NETWORKING TERMS

2.2.2.1 TCP (Transmission Control Protocol)

Internet Engineering Task Force's RFC (RFC793) [11] mentions about TCP as below

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks.

TCP is based on concepts first described by Cerf and Kahn in https://tools.ietf.org/html/rfc793#ref-1 [11]
The TCP fits into a layered protocol architecture just above a basic Internet Protocol (IP) which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes".

2.2.2.2 TCP Echo Server

TCP Echo Server is a preliminary application written using C programming language. It demonstrates TCP Server design using socket programming. It uses TCP sockets to listen to the requests from a TCP Echo client.

2.2.2.3 TCP Echo Client

TCP Echo Client is basically connects to the Echo Server using socket programming with C language.

Note: In this project TCP Echo Server and Echo Client have been developed and compiled using the instructions from the book "The Linux Programming Interface" authored by Michael Kerrisk [12]

2.2.3 PROGRAMMING LANGUAGES

2.2.3.1 C

C is a popular programming language created by Dennis Ritchie and Ken Thompson of AT&T Bell Laboratories in 1970s. C programs are pre-processed, assembled, compiled and linked to result in an executable, which can be run directly on the processor. Popular operating systems like UNIX and Linux, programming languages like Java and Python uses C programming language. In this project, TCP Echo Server and TCP Echo client have been developed using C programming language and compiled using gcc (GNU Compiler Collection) compiler. [13]

2.2.3.2 Python

Python is the most popular programming language in the world where the programs are interpreted before executed by the processor. The language is interactive and object-oriented. Python is a free software and distributed under an Open Source license. Python is bundled with a large number of modules which includes the core package and extensions. This project uses Ansible which is built with Python. [14]

2.2.4 OPEN SOURCE TOOLS USED IN THE PROJECT

2.2.4.1 Ansible

Ansible is a free software originally written by Michael DeHaan. It Is released under the terms of the GPLv3 license. It uses the python interpreter, providing an automation platform for a wide range of modules to automate applications and computing systems and infrastructure deployment and maintenance. Ansible uses SSH for network connections, with no agents to install on remote systems. [1]

2.2.4.2 Prometheus

Prometheus is a system and service monitoring system. It collects metrics from configured targets at given intervals, evaluates the rule expressions, displays the results, and can trigger alerts when specified conditions are observed [16]. It is 100% open source, available under Apache 2 License and development is completely a community driver project. Prometheus is designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems. [21]

2.2.4.3 Grafana

Grafana is an analytics and visualization web application. It is used to visualize the data captured with monitoring systems in the form of charts and graphs by creating monitoring dashboards [18]. It is used in combination with time-series databases such as InfluxDB, Prometheus, etc to visualize metrics, logs and traces. Written in Go language, Grafana is an open source project [19].

The below table provides the list of open source tools along with the download page and license information. Please refer to the introduction to the terminologies / literature references section to understand more about the tools.

Sl. No	Name	Description	Source Code Webpage	License
1	Kernel Virtual Machine(kvm)	Full virtualization solution for Linux on x86	https://git.kernel.org/pub/scm/virt/kvm/kvm.git/snapshot/kvm-for-linus.tar.gz	GNU General Public License version 2 only (GPL-2.0)
2	QEMU	Open source machine & userspace emulator and virtualizer.	https://gitlab.com/qemu-project/qemu	GNU General Public License, version 2
3	libvirt	C API and bindings to other languages for managing virtualization technologies	https://gitlab.com/libvirt/libvirt	GNU Lesser General Public License, version 2.1 (or later)
4	Ansible	IT Automation Platform to deploy and maintain applications and systems	https://github.com/ansible/ansible	GNU General Public License v3.0 or later
5	Prometheus	Systems and Service Monitoring system	https://github.com/prometheus/prometheus	Apache License 2.0
6	Grafana	Observability and Data Visualization platform	https://github.com/grafana/grafana	Apache 2.0 License
7	Node exporter	Metrics exporter for operating system and hardware metrics plugged into prometheus	https://github.com/prometheus/node_exporter	Apache License 2.0

Table 1.1 – Open Source Softwares

3. PROJECT OBJECTIVE AND SCOPE

3.1 PROJECT OBJECTIVE

This project concentrates mainly on the SRE principle **eliminating toil using ansible automation.** [1] [4]
The project also involves studying the **interfacing capabilities of ansible with tools that monitor the service level metrics.** [1][2]

3.1.1 WHY CHOOSE ANSIBLE FOR THIS PROJECT?

Using Python as the interpreter, ansible is basically a python module and classified as configuration management tool, supporting traditional IT and cloud based system automation. This helps to automate tasks on the website hosting systems. The extent of automation possible with Ansible seems to be having a larger scope including Observability and event management, against simply using it for configuration management.

- Ansible is chosen for study as it is a free software (released under the terms of the GPLv3 license) with community based development.
- It has a rich set of modules providing extensive scope for experimentation on IT systems.
- Unlike other open source tools, it is a simple automation tool using SSH (Secure Shell) protocol to perform tasks on the remote hosts, without dependency of agent processes on the hosts.
- Interfacing capabilities with monitoring and analytic tools, is another area of interest.
- Ansible modules not only caters to needs of automating, but also interfacing with various tools that orchestrate together to provide infrastructure solutions at large.

This research work provides an opportunity to
- Study the internals of how ansible modules interact with operating system libraries
- Develop an automation solution with ansible and interface the solution with monitoring and analytic tools.
- Practice the principles that eventually results in Site Reliability.

3.2 SCOPE OF WORK

The scope of this project is to
- Study, practice and document how Ansible would help to automate system functions and reduce toil the with its configuration management & provisioning modules. Refer Use Case 1 for the details of the automation.
- Understand and implement the potential to interface [2] with Prometheus and Grafana for service metrics observability. Refer to Use Case 2 for the details on monitoring and visualization of the metrics.

There are two broad use cases that would be worked upon as part of the project listed here.

3.2.1 USE CASE 1 – AUTOMATION TO REDUCE TOIL WITH ANSIBLE

Automation of the following with Ansible

- Boot a Linux virtual machine from a pre-built image repository
- Install Docker on top of the VM
- Spin a Docker container within the VM
- Host a TCP Echo Server within the container

- Demonstrate TCP Client-Server communication between TCP Client and TCP Echo Server.

3.2.2 USE CASE 2 - CAPTURING THE SYSTEM AND TCP METRICS WITH PROMETHEUS AND VISUALIZE THE METRICS WITH GRAFANA FOR ANALYSIS

- Push infrastructure operational time-series data with **Prometheus**

Reliability of a system depends on the ability to capture the system metrics and use it to analyze the system health. In this project, TCP Echo Server health check metrics and general VM host metrics are collected using metrics collected by the node_exporter tool. A wide variety of data is collected using node_exporter tool with respect to the OS and Hardware. We can also capture customized metrics using textfile collector feature available within node_exporter. The following are the metrics collected:

- TCP Echo Server Running Status (True / False) (Custom metric)
- TCP Echo Server Socket Listen Status (True / False) (Custom metric)
- VM Memory Usage (Memory Used/Memory Free) (Default metric)
- Disk I/O Statistics of VM (read/write I/O) (Default metric)
- Disk Space Usage within VM (Free/Used Bytes) (Default metric)
- Network Usage (Bytes Received / Transmitted) (Default metric)

- Visualization of observable data with **Grafana**

Time series metrics captured with Prometheus, can be visualized within a sophisticated dashboard facility available in Grafana. Graphical representation gives a meaningful understanding of the metrics which can be used for analyzing the behavior of the system under prevailing conditions.

Anomalies in the data can be defined and alerts generated based on the definition. The alerts can be sent to the support personnel via communication channels like mail, chat applications, so that corrective action can be performed. In this project, the use-case involves sending a mail when the TCP Echo Server is not reachable or TCP sockets changes its ‘Listen’ status.

3.2.3 ASSUMPTIONS FOR THE USE CASES:

1) Use case assumes the availability of the operating system image in the local system storage. The **pre-built OS image** is used to create the VM for the experiments. Building an image from the scratch is not within the scope in this project. However automation for building an image is a prospective extension to the project in the future. The automation only involves managing the life-cycle of the operating system image.

2) **Network interfaces** are **well defined** and configured on both host and virtual operating systems., so that the development work in the project involves only at the application layer and TCP layers of the Network protocol stack.

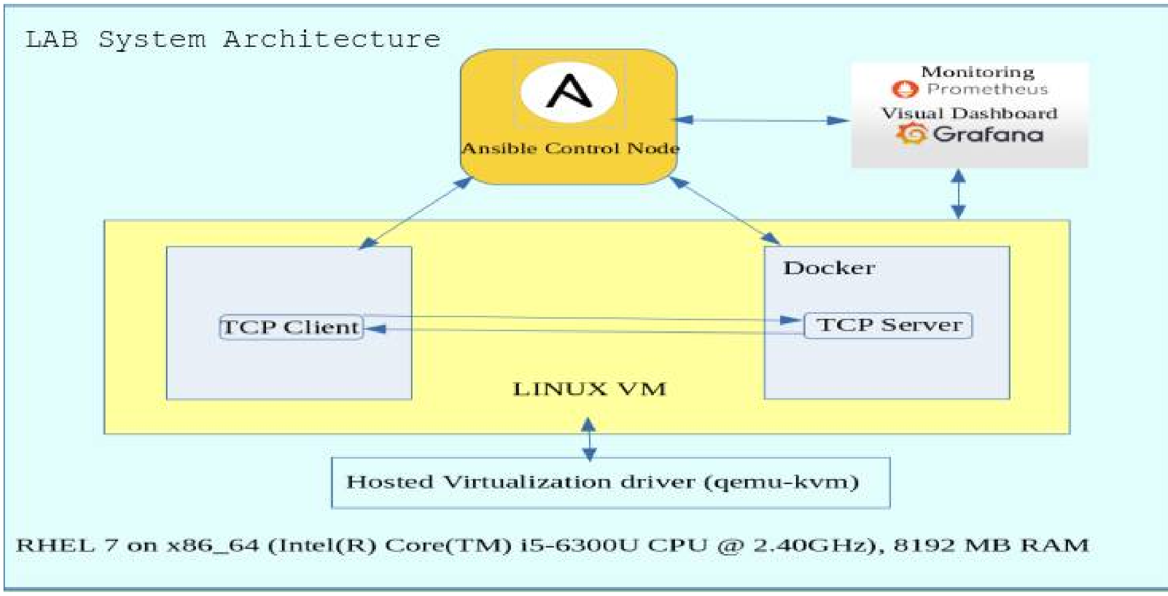
3) VM would be readily **connecting to the tool repositories**, which includes RHEL repositories, Prometheus and Grafana download locations, etc.

4) Network **firewall rules** are **enabled** so that access to the TCP ports used for prometheus (9090), node_exporter (9100) and grafana (3000) is available to the host Linux machine.

5) **SMTP** configuration is **setup** on the Linux VM in order to send the alert messages from Grafana to the support personnel.

4. SETTING THE STAGE FOR THE PROJECT

4.1 LAB ENVIRONMENT



started

ate the

4.2

Fig 1. LAB SYSTEM

The experimental setup for the project needs a host system that would run the desired processes for the project.

The host system used for lab hardware consists of

- A Dual core processor using **x86_64** architecture (Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz)
- 8 GB RAM
- sufficient storage (>30 GB)

The operating system software managing the system resources is Redhat Linux release 3.10.0-1127.19.1.**el7**.x86_64

4.3 VIRTUALIZATION ON THE HOST SYSTEM

The RHEL operating system consists of the kernel module of **kvm (kernel virtual machine) hypervisor** which helps to realize full virtualization. In this project, the scope of virtualization is constrained to the host based virtualization of the Linux OS using kvm. If the implementation of project work is using hypervisors other than kvm, then virtualization drivers need to be chosen appropriately. Other hypervisors are out of scope in this project.

Emulation of the hardware for KVM hypervisor and virtual machine monitoring function is performed by **qemu-kvm** package.
The version and release details of qemu-kvm package used in the host system is as given below.
Version : 1.5.3
Release : 175.el7_9.3

libvirt toolkit is used for Management of virtual machine monitoring system. It facilitates the API which can interact with virtualization capabilities exposed by qemu-kvm. The version and release details of libvirt toolkit used in the host system is given below.
Version : 4.5.0
Release : 36.el7_9.3

4.4 ANSIBLE ENGINE AND ANSIBLE MODULES

Ansible Engine, identified as **Ansible Control Node** in the lab environment, is built from a group of python modules. Using python as interpreter, Ansible consists of python modules, useful in automating the **system configuration management**. In this project, various **ansible modules** are used to perform the following:

- Boot a pre-built image of RHEL as a virtual machine
- Install docker and start docker as a system service.
- Perform docker operations to pull or build docker images
- Use the docker image, to build and run a docker container with a TCP Echo Server process.
- Install and configure monitoring tool prometheus and node_exporter
- Install and configure data visualizer tool grafana for the analysis of the system state.

The above tasks are explained in detail in the following sections.

4.4.1 BOOT A PRE-BUILT RHEL VIRTUAL MACHINE

Ansible's **virt module** can be used to manage virtual machines supported by 'libvirt' API. This module can be used to create/destroy, define/undefine, pause/unpause, shutdown/start/stop virtual machines. The module uses a libvirt connection uri to connect to the virtual machine instance to perform the tasks on the VM. By default, the libvirt connection uri is "*qemu:///system*". 'virt' module can connect to any VM managed by qemu, to execute any command like stop,start,shutdown,etc.

Examples of using virt module with ansible to boot, stop and shutdown the a virtual machine using default libvirt connection uri is provided below.

```
#boot/start the VM  Virtual_Client_RHEL_7-KVM
ansible <host_name> -m virt -a "name=Virtual_Client_RHEL_7-KVM command=start”

#stop the VM  Virtual_Client_RHEL_7-KVM
ansible <host_name> -m virt -a "name=Virtual_Client_RHEL_7-KVM command=stop”

#shutdown the VM  Virtual_Client_RHEL_7-KVM
ansible <host_name> -m virt -a "name=Virtual_Client_RHEL_7-KVM command=shutdown”
```

It should be noted that **sshd service must be started** on the host where ansible is performing the tasks. Ansible uses an ssh connection to perform the remote tasks on the host. Usually a ping test is done using ansible **ping module**, to verify ansible connectivity to the hosts on which the automation is needed.

4.4.2 INSTALL DOCKER

Ansible's **yum module** is useful to install, upgrade, downgrade, remove and list linux packages with yum package manager. By providing the name of the linux package while using 'yum' module, the linux package can be installed on the host ansible is connecting to. The below example show how docker package can be installed on a linux host that ansible connects with.

```
#install latest docker package
ansible -i ./hosts ${IP_rhel} -m yum -a "name=docker state=latest"
```

Ansible's **ansible.builtin.service_facts module** can be used fetch list of all systemd linux services configured on the remote host. By using these facts, ansible scripts is developed to verify if docker service is started or not. If not started, docker service is started on the remote host using ansible **command module**.

It should be noted that, using ansible **systemd module** is recommended while operating with system services.
Example invocation using systemd module with ansible is given below:
ansible <host> -m systemd -a "name=docker.service state=started"

4.4.3 DOCKER OPERATIONS

Once the docker service is started, docker images can be built and containers can be run using the images. In this project, **centos** docker image pulled from docker.io repository using ansible **command module** as given below.

```
 tasks:
 -  name: build docker image for centos
    command: /usr/bin/docker pull centos
```

The above way of defining tasks in ansible is used when a list of ansible tasks is executed in order by ansible-playbook utility. Refer to the ansible-playbook web page [26] provided in the 'Literature References' section.

A docker container running TCP Echo server is built using the centos docker image.A Dockerfile is used to provide the instructions to the docker service while building a docker container. The below command would create a docker container

```
-  name: build docker image for tcpserver
   command: /usr/bin/docker build -t tcpserver /home/baskar/ansible/TCP/app
```

On successfully building the docker container with TCP Echo Server, docker container is started, which initiates an isolated docker container environment running the TCP Echo Server on the desired port number. In the project ansible command module is used to run the TCP Echo server with container name tcpserver_1 and using TCP port 4305. Refer to the next section 4.1.3.4 to understand more about the nature of TCP Echo Server and Echo Client.

```
- name: Run docker container for tcpserver
  command: /usr/bin/docker run --network host --name tcpserver_1 -d tcpserver
```

The ' –network host ' option provided in the command above ensures that the docker container uses the host network interface of the virtual machine. While using the host network interface, the virtual network bridge used by virtual machine. This would enable the external clients to send requests to the docker container. For example, a TCP Echo Client can communicate with TCP Echo Server running within the docker container which in-turn run within the virtual machine.

By default docker would be connected to a bridge network named usually as ‘docker0’. This is usually recommended network interface used when docker containers would communicate among each others.

It should be noted that python module **docker** can be used to perform all the docker operations mentioned above. Ansible modules make use of the python docker API to perform docker operations. For example, docker images can be built using **docker_image** module and docker container management can be done using **docker_container** module.

4.4.4 TCP ECHO SERVER AND TCP ECHO CLIENT

The **TCP Echo Server** is an **iterative server**, where it will be accepting only one client connection at a time. The server handles one client at a time, processing that client’s requests completely, before proceeding to the next client. In general, iterative TCP Echo Server is used when there is simple request-response exchange between client and server. Alternatively, when there is significant processing time is needed to process client request, **concurrent** TCP Echo Server would be implemented. This project considers the most simplest form of a client-server architecture, the iterative TCP Echo server. The binary executable name of the TCP Echo Server used in the project is **echoser.** Usage of the server program is as given below

Usage: <binary_location>/echoser <Server Port>

As already mentioned in the project abstract, the project aims at fundamental study of site reliability with minimal set of functionality for the deployed server. This gives an opportunity to perform research in terms of fundamental study of TCP/IP communication, unlike the webservers and application servers where the code base voluminous with research focussed on the application layers of the network architecture.

TCP Echo Client sends requests to the TCP Echo Server. TCP Echo Client has a simple design, which is programmed to communicate to the TCP Echo Server using the IP address of the server and port where the server socket listens to the client’s connect requests. The client program can be invoked either from within the Linux VM or from outside the VM, as long as the firewall for the server port is opened. The binary executable name of the TCP Echo Server used in the project is **echocli.** Usage of the client program is as given below

Usage: <binary_location>/echocli <Server_IP> <any_dummy_word> <Server_Echo_Port>

Refer to Appendix A to understand the design and code for both TCP Client and Server.

4.4.5 INSTALL AND CONFIGURE MONITORING TOOLS (PROMETHEUS & NODE_EXPORTER)

4.4.5.1 INSTALLATION OF THE MONITORING TOOLS

Prometheus and Node Exporter binaries can downloaded from a download URL using **get_url** ansible module.
Ansible playbook contains the following task to install prometheus.

The version of prometheus can be changed with the usage of standard variables prometheus_version, os_name and arch.

```
- name: copy prometheus to dest server
  get_url:
```

```
        url: "https://github.com/prometheus/prometheus/releases/download/v{{ prometheus_version }}/prometheus-{{ prometheus_version }}.{{ os_name }}-{{ arch}}.tar.gz"
        dest: "{{ prometheus_install_loc }}/prometheus-{{ prometheus_version }}.{{ os_name }}-{{ arch }}.tar.gz"
```

Similarly node_exporter binary can be downloaded from the download url by providing the version information for node_exporter which includes node_exporter_version, os_name and arch variables.

```
    - name: copy node exporter to dest server
      get_url:
        url: "https://github.com/prometheus/node_exporter/releases/download/v{{ node_exporter_version }}/node_exporter-{{ node_exporter_version }}.{{ os_name }}-{{ arch }}.tar.gz"
        dest: "{{ prometheus_install_loc }}/node_exporter-{{ node_exporter_version }}.{{ os_name }}-{{ arch }}.tar.gz"
```

4.4.5.2 SCRAPE SYSTEM METRICS WITH prometheus.yml

Prometheus collects system metrics by itself and also by integrating with other monitoring tools. In this project, prometheus is integrated with node_exporter. This helps prometheus in scraping the OS metrics collected by node exporter, and monitor the metrics as time series data.

The configuration of prometheus to collect the metrics is done using the prometheus.yml file, is placed in the directory where prometheus is installed. The prometheus.yml file is copied to the Linux virtual machine, using ansible **copy module** as given below**.**

```
    - name: copy the prometheus.yml
      copy:
        src: prometheus/prometheus.yml
        dest: "{{ prometheus_install_loc }}/prometheus-{{ prometheus_version }}.{{ os_name }}-{{ arch }}/prometheus.yml"
```

node_exporter is configured as a 'job' in prometheus.yml file. Multiple instances of node_exporter is configured as 'targets'. The targets corresponds to a host name or IP address followed by a TCP port number where prometheus can scrape the system metrics from node_exporter. This enables prometheus to collect metrics from any number of targets where monitoring is required.

The lab system runs one single instance of node_exporter to monitor the Linux system metrics mentioned in the Use case 2. The prometheus.yml configuration for node_exporter job and instance is provided as given below.

```
scrape_configs:
  - job_name: 'prometheus'

    static_configs:
    - targets: ['localhost:9090']

  - job_name: node
    static_configs:
    - targets: ['localhost:9100']
```

Here, job_name 'node' corresponds to the node_exporter installed in the Linux virtual machine. The OS system metrics are available at port '9100' which can be scraped by prometheus for collecting metrics as time series data.

4.4.5.3 ENABLE THE MONITORING TOOLS AS LINUX SYSTEM SERVICES

Both prometheus and node_exporter are enabled as linux system services. A **promethus.service** file and **node_exporter.service** file are copied to the linux virtual machine */etc/systemd/system/* location to install them as linux services.

```
    - name: copy prometheus service file
      copy:
       src: prometheus/prometheus.service
```

```
    dest: /etc/systemd/system/prometheus.service

- name: copy node_exporter service file
  copy:
    src: prometheus/node_exporter.service
    dest: /etc/systemd/system/node_exporter.service
```

Ansible's **systemd** module can be now used to run both the monitoring tools as linux services.

On enabling and running the monitoring services using the corresponding ansible tasks, prometheus console can be accessed from a browser using the http URL **http://<*prometheus_host*>:<*prometheus_http_port*>**

prometheus_host and *prometheus_http_port* information is available in prometheus.yml file under the configuration where **job_name** is **'prometheus'.** The default prometheus_http_port is 9090.

```
scrape_configs:
  - job_name: 'prometheus'

    static_configs:
- targets: ['localhost:9090']
```

4.4.6 INSTALL AND CONFIGURE GRAFANA FOR VISUAL DASHBOARDS

Grafana is available as a linux package under the repository URL https://packages.grafana.com/oss/rpm. Grafana provides with a standard grafana.repo file which can be used as a repository file to download and install. The package installer tool 'yum' available within RHEL, can use this repository file to download and install grafana on the Linux virtual machine.

Ansible's **yum module** refers to the grafana.repo file, to download and install grafana on virtual machine. It is required that the grafana.repo file is available under /etc/yum.repos.d directory for yum package to look for the repository information.

```
- name: copy the grafana repo file to dest server
  copy:
     src: grafana/grafana.repo
     dest: /etc/yum.repos.d/grafana.repo

- name: install grafana from /etc/yum.repos.d/grafana.repo using yum
  yum:
    name: grafana
    state: present
```

Note: It is also possible to download grafana from a download URL using ansible's **get_url module** and install it in the VM.

Installation of the grafana package is followed by enabling grafana-server to run as linux system service using ansible **systemd module**.

```
- name: Start grafana service
  systemd:
      name: grafana-server
      state: started
      enabled: yes
```

The version of grafana used in the project is **grafana-7.4.3.linux-amd64.**

4.5 VISUALIZE THE SYSTEM METRICS USING GRAFANA WITH PROMETHEUS AS DATA SOURCE

A detailed explanation of the automation of various tasks in this project, where booting a pre-built RHEL VM, installing docker, configuring a centos docker image, spinning up a centos docker container with a hosted TCP Echo Server, installation and configuration of prometheus, node_exporter and grafana, was explained in section 4.4 and its subsections.

It can be noticed that ansible provides with a rick set of modules to automate the above mentioned tasks, which includes the modules **virt** for virtual machine related operations with libvirt API, **yum** for installing linux packages, **systemd** for linux services management, **docker** for docker operations, **get_url** for download files using an URL, **copy** to push files from source to destination and **file** for operations on the operating system files. There are also other modules such as **command** and **shell** which have been used for performing few tasks that are performed on the virtual machine.

Using the above automation, a linux VM with a docker container hosting a TCP Echo Server can be quickly deployed for used to realize a TCP Echo Client – Server application. The automation can be modified to quickly spin up other applications such as web servers, application servers, etc by creating a *Dockerfile* corresponding to the application. This automation solutions tries to address the site/system reliability goal or tenet named as 'elimination of toil'.

Installation and configuration of prometheus, node_exporter and grafana in the above sections 4.4.5 and 4..4.6 respectively. Prometheus, node_exporter and grafana takes care of the monitoring and visualization of the system data trying to address the site/system reliability requirement for achieving 'observability' of the system.

Prometheus scrapes OS metrics by integrating with node_exporter and the metrics can be viewed with prometheus console http://<prometheus_host>:<prometheus_http_port>. The time series data can be viewed in a tabular format and graphical format.

Visualization of the metrics on prometheus console is raw in nature i.e. either tabular or graphical view. It also doesn't provide a facility to view the metrics in a consolidated view where multiple metrics can be visualized in a single screen. An appealing visualization of prometheus data can be realized by using Grafana's visual dashboards. With Grafana, a consolidated view of the system metrics can be configured by designing dashboards based on the need. For this, prometheus must be defined as a datasource configuration in Grafana. It is a simple configuration where prometheus host name and http port number is provided in the datasource configuration. Grafana has the capability of integrating with prometheus to display the collected time series data.

Apart from defining the dashboards from scratch, there are also official and community built grafana dashboards in JSON format files. These files can be imported into Grafana configuration to view them as visual dashboards. These dashboards are available in the webpage https://grafana.com/grafana/dashboards. In this project a preconfigured **Node Exporter and Quickstart Dashboard** has been used to capture the following metrics from prometheus. [28]

- CPU Usage
- Load Average
- Memory Usage
- Disk I/O
- Disk Usage
- Network Received

- Network Transmitted

4.5.1 VISUALIZE CUSTOMIZED METRICS FOR TCP ECHO SERVER USING NODE EXPORTER COMMAND LINE FLAG –collector.textfile.directory

Prometheus scrapes the useful system metrics collected by node_exporter. node_exporter comes with a set of collectors, each representing a group of system data as metrics. The collectors are denoted as --collector.<collector_type> where collector type could denote a particular type of system information like cpu information, memory information file system information, network status information, etc. There are group of collectors that are enabled by default and few collectors need to be explicitly enabled. The collectors are passed as command line flags when the node_exporter starts.

Example:

<node_exporter_path>/node_exporter --collector.disable-defaults --collector.cpu --collector.meminfo --collector.netstat

When node_exporter is started as shown above, it can be noted that the all the default collectors are disabled using --collector.disable-defaults and then specific collectors related to cpu information, memory information and network statistics information only are enabled.

The application in this project used is a TCP Echo Server. The server listens on a TCP port to accept the TCP Echo client connections. Hence the primary purpose of monitoring should ensure the TCP Echo server is healthy. This project is designed to check the following two basic information on the server.

- TCP Echo Server Running Status – up (1) or down (0)
- TCP Echo Server Socket State – Listening (1) or Not listening (0)

There are various other factors that can be considered which could determine the health of the server like the response time, number of clients connections waiting, etc. But the scope of this project is restricted to demonstrate the running status and server socket listen status. Monitoring of other parameters can be easily extended by understanding the implementation of these two metrics.

In prometheus, metrics are identified with 'job_name' of the instance at the start of the metric name. The format is like <prometheus_job_name>_<collected_metric_name>. For example, to denote a metric from node_exporter, node_<collected_metric_name> is used where 'node' is the job_name for node_exporter used in prometheus configuration. The following are some of the examples of metric names, collected by node_exporter.

node_memory_Active_bytes – Number of bytes used in the main memory
node_filesystem_avail_bytes – Number of available bytes in the file system
node_network_receive_bytes_total – Number of total received bytes by network interfaces

The list of all metrics collected by node_exporter can be checked using the http URL http://<node_exporter_host>:9100/metrics where 9100 is the default port number used by node_exporter. Prometheus scrapes this information from node_exporter by using this URL.

If there are metrics which are not available in the node_exporter default metrics list, then we can create custom metrics by creating entries in files with .prom extension in the folder specified by the collector flag –collector.textfile.directory
The data available in the .prom files would be available as metrics for node_exporter process. Prometheus can scrape these metrics when it is integrated with node_exporter.

In this project there are two of .prom files created in the –collector.textfile.directory /var/lib/node_exporter/textfile_collector. These files are run_status.prom and socket_state.prom. The metric names captured in these two files are **my_tcp_echo_server_running** and **my_tcp_echo_server_socket_listening** respectively.

If the TCP Echo Server is up and running then the metric value for my_tcp_echo_server_running would be ‘1’. If TCP Echo Server process is not reachable or stopped then the metric value for my_tcp_echo_server_running would be ‘0’. Similarly when the TCP socket for TCP Echo Server is in ‘LISTEN’ status, then the metric value of my_tcp_echo_server_socket_listening is ‘1’ and if the socket state change something other than ‘LISTEN’ then metric value of my_tcp_echo_server_socket_listening would be ‘0’. Since the - - collector.textfile.directory flag is passed to the node_exporter process, these metrics are monitored and ollected by node_exporter every 300 ms.

Prometheus would scrape these metrics from the node_exporter http port (9100). Since Grafana is configured with Prometheus as data source, these custom metrics are available in the Grafana dashboards. Dashboard panels are created one for TCP Echo Server Running status and another panel for TCP Echo Server Listening Status monitoring.

5. EXPERIMENTS WITH IMPLEMENTATION OF USE CASES

5.1 USE CASE 1 - DEPLOYMENT OF TCP ECHO SERVER WITHIN DOCKER CONTAINER

5.1.1 USE CASE FLOW DIAGRAM

5.1.2 RUNNING THE USE CASE IMPLEMENTATION

As mentioned in the scope of work, **step 1 of the use case 1** in the project work involves starting a VM using ansible. Ansible’s virt module can make use of the libvirt library and use qemu-kvm driver to start virtual machines.

Step 2 of the use case 1 involves installing a docker container which hosts the TCP Echo server. The docker images used in the project includes a centos docker image. The docker container application - TCP Echo server is hosted within the centos container image

Step 3 of the use case 1 starts the TCP Echo server within the docker container and use a TCP echo client to communicate with the TCP Echo server.
Ansible modules are used to

- install docker and control the docker operations
- Start the TCP Echo Server within the docker container

The following video demonstrates the Use Case 1 where a TCP Echo server is hosted and started within a docker container within the RHEL virtual machine. A shell script named as *vm_play.sh* is run, which internally calls a series of ansible command line tasks and playbooks to complete the execution. Refer to Appendix A for the detailed source code for the *vm_play.sh* script.

A TCP client process connects to the TCP server to the IP address of the container and a specific port (4305). It is shown from the demonstration that the TCP Echo Client is able to send messages to the Echo Server and the Echo Server is able to echo back the message to the Echo client.

Installing Docker image with TCP Echo Server hosted, starting a docker container and TCP Echo Server started within the docker container is also demonstrated in the video. Copying docker files for TCP Echo Server and Client and performing docker operations.

TCP Echo Client program is used to test the communication with the TCP Echo Server and its functionality.

5.1.3 BENEFITS OF AUTOMATING THE VM AND DOCKER OPERATIONS USING ANSIBLE

Automation of deploying the TCP Echo Server in the docker container within a VM is a significant improvement over the manual installation of VM followed by installing the docker container and then deploying the TCP Echo server within the container.

This effort would help to spend more time on the actual project work related to monitoring, data gathering and analysis, drastically reducing the time to deploy the TCP Echo Server application compared to manual deployment.

In any commercial or enterprise environment, where the system resources scale to the level of traditional or cloud data centers, this ansible automation would be useful, where Linux virtual machines and docker containers can be spun instantaneously catering to the scalability and elasticity requirements for the hosted web applications.

5.2 USE CASE 2 - COLLECTING SYSTEM METRICS AND MONITORING THE ENVIRONMENT

In the Use Case 2 TCP and system metrics are measured and collected using monitoring and the data collected is used for interpretation in a visual dashboard. **Monitoring** tools used are **prometheus and node_exporter** while the **visual dashboards** are created using **Grafana.**

6. A STUDY OF SITE RELIABILITY OF THE LAB ENVIRONMENT

6.1 REDUCING TOIL WITH ANSIBLE AUTOMATION

6.2 OBSERVABILITY WITH PROMETHEUS AND GRAFANA

8. LITERATURE REFERENCES

[1] Ansible references – https://github.com/ansible/ansible, https://docs.ansible.com/
[2] Integration with DevOps tools: https://www.ansible.com/integrations/devops-tools
[3] Site Reliability Engineering principles and practises – https://sre.google/sre-book/part-II-principles/, https://sre.google/sre-book/part-III-practices/
[4] Eliminating Toil - https://sre.google/sre-book/eliminating-toil
[5] GitHub page for Linux - https://github.com/torvalds/linux
[6] Linux manual pages - https://www.kernel.org/doc/man-pages/
[7] libvirt reference - https://libvirt.org/
[8] Linux KVM website - https://www.linux-kvm.org/page/Main_Page
[9] QEMU wiki - https://wiki.qemu.org/Main_Page
[10] QEMU website - https://www.qemu.org/
[11] TCP RFC 793 - https://tools.ietf.org/html/rfc793
[12] The Linux Programming Interface book by Michael Kerrisk - https://www.man7.org/tlpi/
[13] GNU Compiler Collection (GCC) webpage - http://gcc.gnu.org/onlinedocs/gcc/
[14] Python website - https://www.python.org/
[15] Visualize Prometheus data with Grafana - https://www.openlogic.com/blog/how-visualize-prometheus-data-grafana
[16] GitHub page for Prometheus - https://github.com/prometheus/prometheus
[17] Prometheus web site - https://prometheus.io/
[18] Wikipedia page for Grafana - https://en.wikipedia.org/wiki/Grafana
[19] GitHub page for Grafana - https://github.com/grafana/grafana
[20] Overview of Docker - https://docs.docker.com/get-started/overview
[21] Overview of Prometheus - https://prometheus.io/docs/introduction/overview/
[22] How to make docker container use local network interface - https://docs.docker.com/network/network-tutorial-host/
[23] docker module in ansible - https://docs.ansible.com/ansible/2.9/modules/list_of_cloud_modules.html#docker
[24] ansible systemd module - https://docs.ansible.com/ansible/2.9/modules/systemd_module.html#systemd-module
[25] Working with Ansible Playbooks - https://docs.ansible.com/ansible/latest/user_guide/playbooks.html
[26] Michael Kerrisk, The Linux Programming Interface, 2021, O'Reilly Media, Inc. Chapter 60. "SOCKETS: SERVER DESIGN"
[27] Michael Kerrisk, Source Code for The Linux Programming Interface - https://man7.org/tlpi/code/index.html
[28] Node Exporter and Quick Start Dashboard for Grafana - https://grafana.com/grafana/dashboards/13978?pg=dashboards&plcmt=featured-sub1

APPENDIX A – Design and Source Code

PARTICULARS OF SUPERVISOR AND ADDITIONAL EXAMINER

Supervisor's Name:	Swetha J
Supervisor's Email:	swethaj.iyer@in.ibm.com
Supervisor's Qualification:	M.S in Systems Engineering from BITS, Pilani
Supervisor's Designation & Address:	SAP Technical Architect at IBM with 13 years of experience in SAP and Infrastructure.

Additional Examiner's Name:	Raghu Srinivasan
Additional Examiner's Email:	sriniv@us.ibm.com
Additional Examiner's Qualification:	B.E. Instrumentation and Technology 15 years of experience as IT Architect 6 years of experience as Senior Technical Staff Member
Additional Examiner's Designation & Address:	Senior Technical Staff Member (STSM)- Technology Architect Lead Client Transformation SRE (Site Reliability Engineer) Progressive Hybrid Services Integration, IBM Services