National College of Ireland

Automate Provisioning and Orchestration of Cloud Infrastructure using AWX

MSc Research Project
Cloud Computing

Sivaraj Thiyagarajan
Student ID: x20182473

School of Computing
National College of Ireland

Supervisor: Dr. Majid Latifi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sivaraj Thiyagarajan
Student ID:	x20182473
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Majid Latifi
Submission Due Date:	31/01/2022
Project Title:	Automate Provisioning and Orchestration of Cloud Infrastructure using AWX
Word Count:	6501
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Automate Provisioning and Orchestration of Cloud Infrastructure using AWX

Sivaraj Thiyagarajan
x20182473

Abstract

Hybrid-cloud infrastructure adaption is a popular trend in the recent era. As infrastructure grows, the need to design and develop an effective method to automate provisioning and control multiple clouds becomes crucial, as the manual method faces complexities in resource provisioning, application deployment, monitoring, and centralized control. This paper examines the concepts of infrastructure as code (IAC) and the advantages of using it instead of the manual method. Compares and contrasts the features and capabilities of commercially licensed IAC tools with the open-source GUI-based Ansible Works (AWX) tool. This Paper introduces a centralized architecture that includes on-premise and public, private cloud infrastructures using AWX. The study also involves the design, development of individual infrastructure using Ansible playbooks and deploying servers and clusters (Kubernetes, Hadoop) in Amazon web services (AWS) and Google Cloud Platform (GCP) using AWX. Our goal is to integrate automation in resource provisioning and administration of multiple infrastructures using a single tool (AWX) to handle all infrastructure administration tasks. This study also compares the implementation of similar infrastructure using manual method and evaluates the efforts, time, and cost, also discuss how this approach could help enterprises with hybrid infrastructure to establish efficient centralized resource management.

1 Introduction

Cloud computing delivers highly available, elastic, scalable, authentic services to the customers, Due to its features, more organizations migrate their workloads to the cloud. The public cloud provider offers a variety of services to fulfill customer needs. Unlike the traditional data center model, the cloud provides an easy and rapid implementation of cloud resources, which helps businesses to bring the product to the market quickly. Cloud resources are billed based on utilization and resources can be terminated when they are no longer needed. Enterprises can easily scale the resources if their business grows, without investing huge upfront costs. Hybrid infrastructure helps enterprises meet business compliance and audit requirements by maintaining confidential data and workloads on-premises.

Provision and control of cloud resources are handled manually using the cloud provider console. Handling single cloud infrastructure will not create major operational overhead. But infrastructure involving multiple clouds and hybrid infrastructure will be more complex to handle, requires additional support staff to maintain the infrastructure, which creates additional costs to businesses. Tomarchio et al. (2020) illustrated the need

of implementing the cloud resource orchestration frameworks (CROF) in the environment involving multiple clouds, for effective life-cycle management of cloud resources from resource implementation to the monitoring phase. The use of agile methodologies and the DevOps framework in service delivery minimizes the amount of time it takes to deploy and test software, allowing businesses to get services to market faster. The IAC approach enables this rapid transition in provisioning and orchestration schedule reduction.

Infrastructure as a code is an efficient process of creating and controlling the On-premise and cloud infrastructure using codes. Lavriv et al. (2018) used the concept of Infrastructure as a Code to design a template for creating cloud resources. It's just a bunch of code that tells cloud providers what to do. The Infrastructure as a Service (IaaS) model provides compute, storage, and networking considerably more accessible well as offers unrivaled automation capabilities. It helps to control the present state of the infrastructure while reducing the time it takes to create.

Open Source Software (OSS) has dominated the technology industry for the past two decades. Collaboration and innovation are fostered through open-source projects. OSS pursues a community-centric strategy. Updates and distribution of codes are possible. It fosters open participation in code testing as well as quick issue resolution, resulting in more reliable software. Popular Open-source tools include Kubernetes, Origin Community Distribution (OKD), GitHub, and AWX. According to research, open-source software is more secure than vendor proprietary software.

Singh et al. (2016) denotes, Ansible is a well-known IAC tool for DevOps automation and configuration management. It uses Playbooks created using yet another Markup language (YAML), which can be used to provision and orchestrate servers, networking devices, and applications. Compared to other IAC tools, Ansible delivers simple, modular, flexible, and efficient command-line interface (CLI) functions. But managing multiple playbooks using CLI leads to administrative challenges and a lack of control. GUI-based Ansible tool help overcome all major challenges faced in CLI-based Ansible. AWX is a well-known GUI-based Ansible tool built over Ansible engine to provide advanced features such as Web user Interface, Role-based access control (RBAC), REST API, Job scheduling, and integration of multiple clouds and on-premise infrastructure, centralized control, and configuration management could be configured by customized dashboard creation.

In this paper, I propose a novel approach to automate resource provisioning and orchestration in public, private, and hybrid cloud environments using a graphical open-source IAC tool. This approach uses an Open-source project Ansible Worx to combine private cloud, public cloud (AWS, GCP), and on-premises infrastructure into the IAC tool and deploy infrastructure resources using YAML codes to deploy, control, and orchestrate repeated system administration tasks to enhance overall cloud automation and management using a GUI based IAC tool.

1.1 Research Question

Complexity in resource provisioning, orchestration, and centralized Infrastructure administration are the major concerns in Organizations employing Multi and hybrid cloud Infrastructures using manual methods. Therefore there is a research question as follows:

RQ: To what extent does a single Open-source IAC tool effectively handle centralized resource provisioning and orchestration for enterprises using multi-cloud and hybrid infrastructures?

1.2 Research Objectives

The following Research Objectives are considered to address the above mentioned RQ:

1. Understanding the complexities and variability concerning manual and automated cloud deployments.
2. Analyse challenges, emerging trends, and interoperability of Infrastructure as a code in multi-cloud infrastructures.
3. Design Open-source IAC architecture for centralized control and administration.
4. Develop complex cluster environments using Yet Another Markup Language (YAML) codes for Amazon Web Services (AWS) and Google Cloud Platform (GCP) Cloud infrastructure.
5. Automate provisioning and orchestrating cluster environments using AWX.
6. Integrate Github to enable continuous integration (CI) and continuous delivery (CD) using AWX.
7. Create centralized control and dashboards for overall infrastructure management.
8. Evaluating the implementation time and costs between Manual and Automated IAC cloud deployment methods.

1.3 Contribution

As a Contribution, this research introduces a centralized architecture using a web-based graphical IAC tool (AWX) that can provision and administrate the on-premise and multiple cloud infrastructures. To begin, multiple difficulties of resource orchestration in the current cloud ecosystem are explored and I create a comprehensive taxonomy of desired properties and dimensions that may be used to characterize Infrastructure as a code by thoroughly analyzing recently published literature. I examine a variety of traditional infrastructure scenarios and design, develop and deploy the same architecture with centralized control using the AWX. This will assist the reader in understanding not only the merits of each architecture and implementation but also the unsolved difficulties that must be addressed in the future.

1.4 Document Structure

The rest of the paper is organized as follows. In Section 2, I discuss the related works conducted by various researchers under the same context, specification analysis of different IAC tools available, and its functionalities compared. I explain the detailed methodology and the techniques used to materialize the research objective in Section 3. Design details, architectural diagrams are discussed in Section 4. I have provided a detailed implementation strategy for each scenario in Section 5. Section 6 evaluates the time metrics while implementing the different case studies and cost comparison of commercial tools, and, I analyze the potential benefits and the advantages of using AWX over the manual method by comparing the results based on time and cost benefits. In Section 7, I conclude the potential benefits of integrating GUI-based IAC tool over the traditional method and how AWX improve overall productivity, centralized control, and infrastructure automation.

2 Related Work

Cloud resource orchestration, as described in Section 1, Introduction, is concerned with the discovery, selection, allocation, and administration of cloud resources. In this section, I looked at the most recent literature in the fields indicated, I was looking for suggestions, frameworks, prototypes, and commercially licensed products that address the challenges. I discovered that several researchers have already published papers related to our research topic. Each of these studies lists and categorizes a variety of projects that fall under the broad tent of cloud resource orchestration, whether they are proficient or smaller proposals focusing on a specific set of orchestration characteristics. I compare on-premise and multi-cloud frameworks. Review various methods, studies, and protocols used. How the software and hardware as codes are defined. Also, discuss the business standard frameworks for IAC, and comparative study of use cases, proven methods, issues, and best practices for implementing infrastructure resources.

2.1 Challenges in Manual Cloud Provisioning and Control

As cloud computing gains traction in the IT industry, the change from monolithic to microservices design, as well as the shift from largely virtual machines (VMs) to a cloud-native architecture, has made it far more critical to automate infrastructure to respond quickly, academics and practitioners are reporting an increasing number of concerns and challenges. Traditional deployment and control will not be an easy task to achieve the centralized administration, where each cloud infrastructure needs to be managed individually via vendor console. Common challenges faced in multi-cloud infrastructure are provisioning, application deployment, load balancing, costs, and monitoring. Ghanam et al. (2019) denoted the critical challenges faced by enterprises involving hybrid infrastructure. The author analyzed the major issues such as data management, service management, quality, economic challenges, security, and privacy. Barhate and Dhore (2018) mentioned that the interoperability (ability to work with each other, when multiple cloud systems are involved) and Cloud brokering concerns caused by parallel access to services from different providers cannot be overlooked in the review of resource orchestration when multiple cloud infrastructures are involved. Implementing open standards at all levels in the cloud is one strategy to solve the interoperability challenge. The cloud computing industry is pursuing several cloud standardization initiatives. These efforts to standardize focus on challenges such as workload management, data migration, and user authentication, among others. IT teams may recover control of their multi-cloud ecosystems by using cloud automation and orchestration technologies that take advantage of scheduling capabilities. Provisioning servers, backing up data, and controlling unused processes and resources are all duties handled by cloud automation. Low-level tasks across different platforms can then be streamlined into more complicated processes via cloud orchestration, freeing the team from manually meddling with everyday chores. Cloud orchestration may assist ensure that cost overruns and under-utilization of services do not damage the bottom line once the development, security, and operations teams have full visibility and can discover any weaknesses in the infrastructure. Cloud orchestration and automation services offered by cloud providers are proprietary, which doesn't support other vendor services to orchestrate.

2.2 On-premise and Cloud Frameworks

Cloud computing has gotten a lot of attention in the last decade because of its unique properties such as easy adaptability, flexibility, and supportability for a wide range of applications and software. Banking and finance sector businesses need to maintain on-premise infrastructure to maintain confidential business data and sensitive information related to the transactions, to comply with Payment Card Industry (PCI) audit requirements. Applications requiring low latency, high throughput, and very-high Input-Output Systems (IOPS) are hosted and controlled in-house datacenters.

According to Sun et al. (2016), private cloud infrastructure offers outstanding security, privacy, scalability, and performance. Firms can host a private cloud on a vendor's facility or in their data centers. According to Hassan et al. (2019) organizations evaluate factors such as elasticity, security, compliance, storage, data sensitivity, scaling, and cost before adopting the cloud. The public cloud offers huge clusters of resources for big data applications on a pay-per-use basis, Setting up similar infrastructure in a traditional datacenter model requires huge cost and time. According to Ardagna (2015), a public cloud infrastructure hosted in a single vendor creates several issues such as Vendor dependability, supportability and higher costs. Ardagna et al. (2012) in their work denotes multi-cloud infrastructures helps control vendor lock-in, hardware complexity, and quality issues. To assist operators and developers, they suggest the MODA CLOUDS (Model-driven development) approach, which defines the standards for developing and operating applications in multi-cloud setups.

2.3 Orchestration and Provision of Cloud Infrastructures

The process of allocating cloud resources and services to enterprises depending on their needs is known as cloud provisioning. Typically, cloud provisioning is managed by the IT employees of the organization, who connect into the vendor-provided console to deploy cloud resources. To install cloud resources and services in multi-cloud and private cloud situations, the administrator must log in to each vendor's cloud console. Kritikos et al. (2019) discussed the framework to perform business migration using cross-level orchestration for cloud adaption and monitoring. Bousselmi et al. (2014) compare and contrast the various technique of orchestration, they discuss software as a service orchestration SAAS(O) and the infrastructure as a service orchestration IAAS(O) strategy, as well as the problems that came with it, and determined that both techniques were effective in implementing resources. On-demand dynamic provisioning, manual provisioning, and post-sales provisioning are the three different delivery strategies. Provisioning and coordinating cloud resources in multi-cloud and hybrid configurations have become increasingly complex and difficult to handle as the catalog of different cloud provider services has grown. Tomarchio et al. (2020) defines the cloud resource orchestration framework (CROFs) to manage the entire life-cycle from resource implementation to the control phase. Tamburri et al. (2019) introduced an industrially state-of-the-art standardization that uses standard notation to automate the deployment of technology-independent and various cloud-supported apps. Specifies infrastructure as a topology, allowing for faster reuse. As illustrated by Ranjan et al. (2015), the primary goal of orchestration is to ensure that services are delivered consistently to meet the Quality of Service (QoS) requirements.

2.4 DevOps and Infrastructure as a Code Tools

Software engineering methodologies are used in DevOps (Development and Operations) to cascade development and operational tasks and improve overall deliverables. Automated deployments, multi-cloud deployments, disposable environments, immutable infrastructure, security automation, disaster recovery and backups, and blue-green deployments are all major infrastructure as a code use cases. DevOps integrates and reuses conventional development tools (For example, code management, code revision, code versioning). Artac et al. (2017) connect Devops strategies to the OASIS "Topology and Orchestration Specification for Cloud Applications" blueprint (TOSCA). Shvetcova et al. (2020) present Coloni architecture tool that provides application infrastructure in a hybrid cloud environment using TOSCA templates and creates Ansible playbooks to deploy complex cloud topologies efficiently. Sandobalin et al. (2020) contrasted the Model-driven (Argon) and code-centric (Ansible) IAC tools. The results demonstrate Ansible's ability to deploy software and infrastructure services quickly. The usefulness and specifications of various IAC tools were compared. Guerriero et al. (2019) denotes, Ansible is the only tool among the other tools, used by 70 percent of enterprises. It also compares the functionality of each tool in detail, including extensibility, automation, usability, coding, maturity, and interoperability, and concludes that each tool has its own set of functionality and can be chosen based on use cases. Figure 1 shows the comparison study of different IAC tools in the market based on its architecture, methods used for propagation of change, language used to develop, programming approach, interoperability with other operating system flavors, authentication methodology, integrity, and confidentiality. This functionality helps understand businesses to adopt the right tool to fulfill their infrastructure needs.

	Puppet	**Chef**	**Ansible**	**Terraform**
Type	Configuration Management	Configuration Management	Configuration Management	Orchestration
Infrastructure	Mutable	Mutable	Mutable	Immutable
Architecture	Client-Server	Client-Server	Server-Only	Server-Only
Source Code	Open Source	Open Source	Open Source	Open Source
Changes propagation method	Pull	Pull	Push	Push
Programming Language	Puppet DSL	Ruby DSL	YAML	HashiCorp Configuration Language
Programming Language Approach	Declarative	Imperative	Declarative and Imperative	Declarative
Interoperability	Master - Unix, Agents - Unix/Windows	Server - Unix, Clients - Unix/Windows	Server - Unix, supports configuration of Windows	Vast majority of operating systems
Accountability	Activity service API	Ruby based Chef Log Resource class	Ansible built-in Logs	Terraform built-in Logs
Authentication	Certificate-based	Certificate-based	Cloud built-in API	Cloud built-in API
Availability	Multi-master architecture. If Primary master fails, Secondary master takes its place.	Contains primary server as well as backup server.	Contains primary instance which is substituted by secondary instance.	Contains single active node that is substituted by secondary node.
Integrity	Periodic client configuration validation	Periodic client configuration validation	N/A	N/A
Confidentiality	HTTPS	HTTPS	SSH	Cloud build-in API

Figure 1: IAC Tools Comparison

2.5 Orchestration and provisioning using Ansible

Ansible is a popular open-source IAC tool in recent years, with the most forks on GitHub thanks to its simple structure, human-readable code, and ease of creation, manipulation, and execution. Ansible allows you to use YAML playbooks to provision complicated cluster systems like OpenStack, Kubernetes, Hadoop, ECS, and more. Kokuryo et al. (2020) explore the Ansible terms such as task, module, role, and Playbook, They examined the invalid module, incorrect application of imperative modules, practices to be avoided, and best methods for the proper execution of the task. Palma et al. (2020) analyzed a python-based package (Ansible METRICS) that evaluates the usage of the Ansible command-line and illustrates the execution and operation workflow. The program uses the YAML file as input and stores it as an index list, then extracts the plays and executes the jobs as defined in the input, allowing the play to be implemented in the infrastructure. Masek et al. (2018) created a framework for their university systems using Ansible and Java, which provide web UI to operate infrastructure via smart devices. Pieplu (2018) implemented application automation using Ansible and Jenkins in a CI/CD pipeline. Results demonstrate fast, efficient, and optimized hosting of web applications using Ansible. Juopperi (2017) used the Chatops (a recent add-on in DevOps function, in which development updates to the service may be made via chat service) was created and provisioned in AWS cloud architecture. Kaush and Gupta (2017) implemented OpenStack cluster using the Ansible and noted that the manual method for deploying OpenStack takes several days. They noted developing the complete cluster using YAML helps to understand the errors during the dry-run stage.

2.6 Impact of Open-source Software's IT Industry and AWX

Adopting the enterprise-level paid multi-cloud management software adds a huge burden to the overall IT budget. Open-source software (OSS) would be the best alternative. Reshad and Sinha (2020) discusses the factors that promote open source software solutions from business and technical perspectives. The authors denote the open-source projects are more secure and reliable, as more users test the source code which in-turns provide great chances of frequent checks and bug fixes. Khandelwal and Kumar (2020) compares the features of OSS and closed source software based on development, flexibility, innovation, usability, cost, security and argues open-source software can be a viable solution for businesses looking to save proprietary software license prices while simultaneously avoiding the penalties and legal consequences that come with unauthorized use.

Ansible Worx (AWX) is an open-source community project (managed by Ansible-galaxy) that is upstream of Red Hat's Ansible Tower product. which encourages community participation and encourages developers to enhance and repair bugs. AWX can do automation via a restful API, plan and automate a workflow, establish a dashboard for access reporting and monitoring the success and failures of activities, jobs, projects, inventories, and even configure device monitoring. The integrated RBAC (Role-based access control) can be used to configure access delegation. AWX addresses Ansible tool difficulties such as managing various environments and performing scheduled operations, as well as access control complexity. Complex clusters and Autoscaling Cloud resources can be implemented as different projects, and resource management can be handled efficiently. Table 1 summarises the related work carried out by several researchers in the field of IAC, as well as a comparison study of its methodology, merits, and demerits.

Table 1: Summary of Related Works

Related Work	**Methodologies**	**Advantages**	**Limitations**
Ghanam et al. (2019)	Challenges in multi cloud setups	Hybrid cloud framework	Interoperability, Service and Economic challenges
Ardagna et al. (2012)	MOdel-Driven Approach for the design and execution of applications on multiple Clouds	Monitor and redeploy by observing performance	Complex for multi cloud setups
Bousselmi et al. (2014)	Comparison of Infrastructure as a service (IAAS) and Software as a service SAAS approach	Efficient provisioning using SAAS	challenges in IAAS provisioning
Reshad and Sinha (2020)	Opensource source solutions(OSS)	Flexibility, security in OSS	Interoperability and support availability
Tomarchio et al. (2020)	Systematic review of Cloud resource Orchestration (CRO) Framework	Open issues in multi-cloud environments	Challenges in traditional provisioning methods
Tamburri et al. (2019)	Topology and Orchestration Specification for cloud application (TOSCA)	Declarative modeling	Challenges to achieving Quality of Service
Shvetcova et al. (2020)	Clouni proposed TOSCA Architecture	Deploy complex cloud topology	Challenges in TOSCA deployments
Sandobalin et al. (2020)	Model-driven (Argon) and code-centric (Ansible)	Usefulness and quality of code-centric (IAC) deployments	Drawbacks of Argon model
Guerriero et al. (2019)	Evaluation of various IAC tools	Best practices, development, evolution of IAC tools	Challenges in integration of codes
Palma et al. (2020)	Ansible METRICS Analyzer	Develop quality codes	Problematic metrics in code development
Pieplu (2018)	Deployment of Cloud scenarios using Ansible and GIT	Methodology to deploy for entire application stack	Challenges in manual implementation
Juopperi (2017) Kaush and Gupta (2017)	Deployment of openstack cluster automation using Ansible	Deployment time reduction	Challenges in configuring complex codes for clusters

3 Proposed Methodology

This section describes the research methods used to achieve the aforementioned research objectives. The proposed approach is designed to address the practical challenges such as resource provisioning, application deployment, visibility, control, governance, compliance, cost management faced in manual implementation methods. I performed a detailed literature review of existing deployment strategies and methods for commencing any complex cluster infrastructures by making an in-depth analysis of each stage in the deployment lifecycle. This was taken to obtain a broad understanding of existing standards, methodologies, complexities, and pitfalls in the existing approach and to develop an advanced automated solution to deliver the desired infrastructure.

As shown in the Figure 2, the proposed research methodology comprises of five steps, 1) Requirement Analysis 2) Design, 3) Develop, 4) Deploy, 5) Evaluate

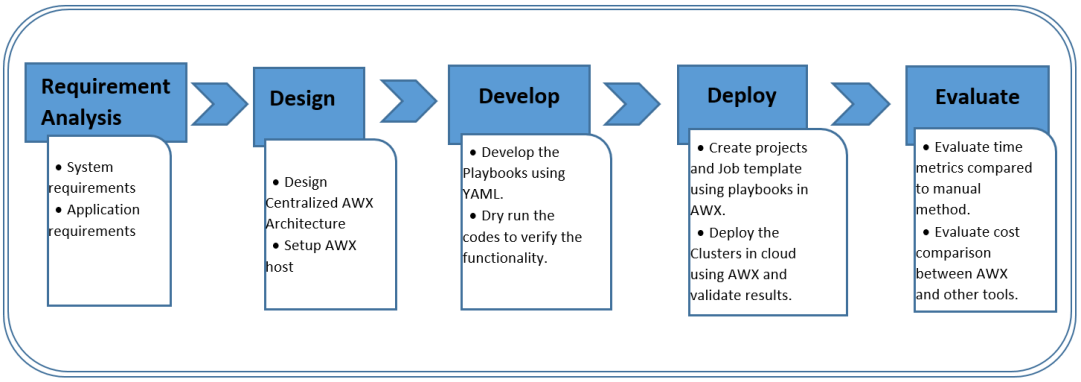

Figure 2: Proposed Research Methodology

- **Requirement Analysis:** Analyzed existing frameworks, standards followed in cloud orchestration and provisioning. Collect information related to application compatibility and the system level requirements to set up the AWX server and implement the server with all required packages. Cloud service providers information and the services required to be deployed as a part of the research are analyzed. create user accounts and security keys from cloud service provider console.

- **Design:** Design centralized architecture using AWX by integrating AWS and GCP cloud infrastructure along with the in-house servers. Design separate architecture for Amazon instance and Google cloud instances, Kubernetes, Hadoop cluster environment, define each cloud component involved in implementing the resources and denote the connectivity between each component in the architecture.

- **Develop:** This phase is related to code development, where the major challenge of multi-cloud infrastructure lies in interoperability between On-premise to cloud infrastructure and between different cloud service providers. Application interoperability, platform interoperability, and management interoperability. A detailed review of the requirements is examined based on the individual cluster scenarios and used appropriate Ansible modules to support the configuration are developed to overcome interoperability and coding dependencies. Develop the individual Ansible playbooks for each cluster and the resource defined in the design section using

YAML codes. Ansible roles are used to define specific tasks. Hadoop deployment playbook consists of five playbooks, the master playbook is defined with four child plays (ec2 instance creation, Hadoop master node creation, Hadoop slave node creation, Hadoop client creation). Kubernetes cluster deployment playbook consists of a master and child plays for deploying ec2 instance, Kubernetes master and slave servers in Amazon cloud. The playbooks are designed to execute entire cluster components in sequential order defined in the master playbook. Test the developed codes using dry-run functionality. A fully developed and tested playbook allows administrators to deploy the cloud resources remotely.

- **Deploy:** Create an AWX host by installing the required packages. Configure cloud credentials in the AWX console for each cloud service provider account, create individual projects for each cluster, and create job templates using the playbooks created on the development stage on the AWX console. Configure GitHub credentials to deploy the playbooks directly from the Git repository. Deploy Kubernetes, Hadoop cluster environments directly by executing each job template. Validate the cloud resources by login into the cloud console for verification.

- **Evaluate:** Compare and assess the metrics for manual versus AWX-based deployments gathered throughout the deploy phase. List the drawbacks of the manual technique and compare them to the advantages and benefits of using AWX. Compare the expenses and benefits of utilizing the AWX tool to the IAC commercial tools.

4 Design Specification

A high-level architecture diagram of the centralized resource provisioning and management using AWX is illustrated in the Figure 3. It shows the in-build features of the tool, the connectivity between the AWX and the public, private, and on-premise data center. Details of the individual components of AWX are described below.

- **Ansible -** Simple software engine developed in python to automate provisioning, configuration management, orchestration, and application deployments using YAML codes. playbooks were created using these codes for deployment.

- **Modules and Roles -** Pre-defined functionality to perform a specific task, there are 2500+ modules available in Ansible Galaxy (Ansible repository) to perform various automation tasks. Roles are Used to load related files, variables, tasks, and artifacts in a defined file structure, which helps to define the requirements in a defined structure.

- **API -** Representational State Transfer (REST) Application program Interface (API) performs HTTP requests using GET, POST, DELETE and PUT data types to access data for processing and the plugins handle service automation.

- **CMDB (Inventory) -** Complete configuration management database of hosts, devices, credentials, roles, users, details about Role-based access control (RBAC), operating system, patch level, etc are stored to keep track of entire assets in the enterprise.

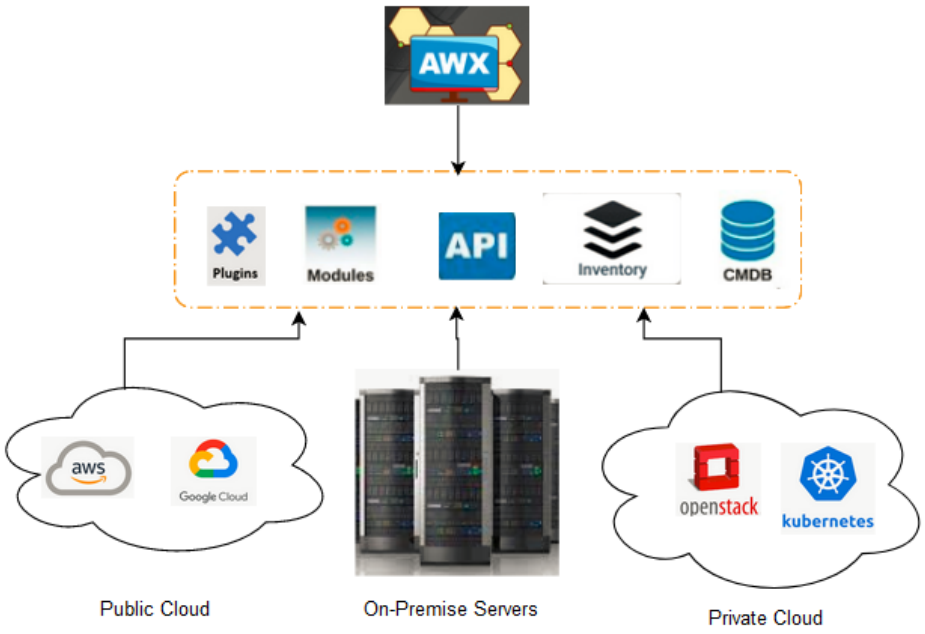

Figure 3: Proposed AWX Centralized Architecture

Each play must be created using variables such as the operating system, packages, and configuration settings. On the AWX server, completely developed code will be placed in the /var/lib/AWX directory. I configured the master playbook by connecting individual plays that have all the dependencies needed to perform the main task effectively.

Figure 4 depicts the Ansible Worx's task workflow and component interconnection within the system. Job templates handle playbooks from different projects and execution can be done manually or by scheduling. A complete record of the job history, execution summary, and configuration management are maintained within the AWX.

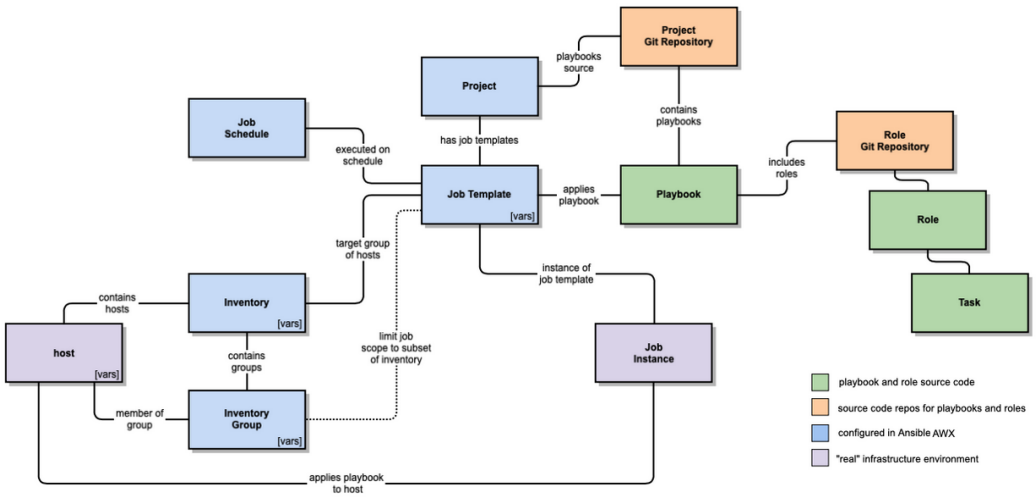

Figure 4: AWX Workflow Diagram

4.1 Design on-premise and cloud servers using AWX

In this section, I designed and created a playbook to deploy two EC2 instances along with an AWS S3 storage bucket. Another playbook is created to deploy a compute instance along with web-server packages in the GCP cloud. Figure 5 shows the architecture diagram of the proposed EC2 deployment with S3 storage on a VPC within the AWS cloud service and a compute instance deployed with Nginx web server service. Cloud components for this design are AWS (virtual private cloud, Amazon internet gateway, elastic compute service, security group, simple storage service) and GCP cloud (compute instance, external network address translator)

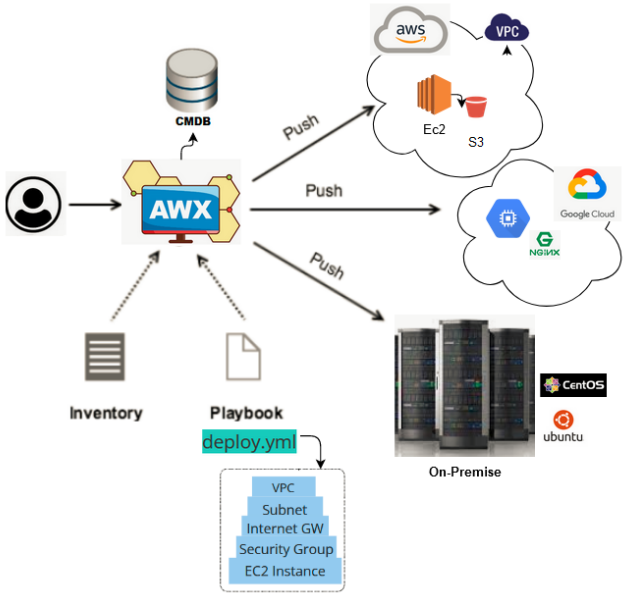

Figure 5: On-Premise, AWS and GCP cloud architecture

I installed two virtual machines on the host machine (using Oracle Virtual Box) as on-premise servers. Separate playbooks were created for system administration tasks, such as package installation, package removal, uptime, and filesystem status checks.

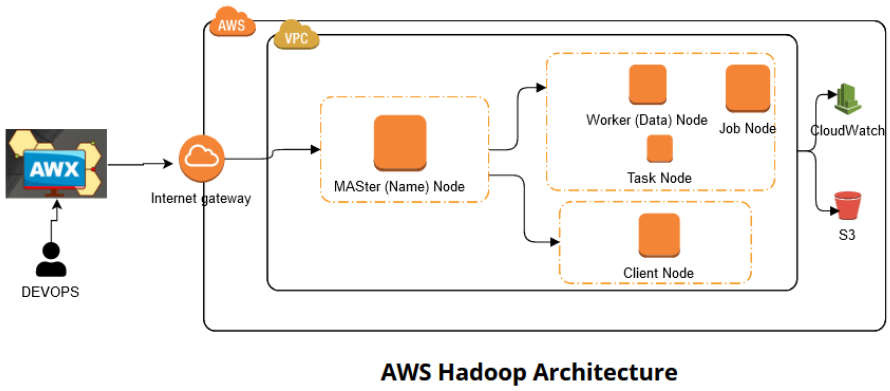

Figure 6: Multi-Node Hadoop Architecture

4.2 Design Multi-node Hadoop Cluster Using AWX

This section covers the architectural design of a Hadoop cluster in the AWS cloud. Hadoop is a scalable distributed computing software used to process a very large volume of data. Hadoop cluster deployment is very complex to implement. Developed the end-to-end YAML code and create the project, template, and deployment using job template in AWX for automated provisioning. Figure 6 shows the deployed Hadoop master and worker nodes using EC2 instance in AWS. Amazon cloud components used are Amazon Virtual Private Cloud (VPC), Elastic Compute Service (EC2). Hadoop components and the connectivity between each component are shown in the figure and the details of individual Hadoop cluster components are as follows. Masternode (Control and manage the Hadoop distributed file-system storage and the other nodes in the cluster). Worker node(Perform data processing as well as create, delete data blocks as per master-node instructions). Task and Job Node(Perform parallel processing for computational power); Client node (Submit Mapreduce jobs, Perform data loading to the Hadoop cluster and fetch results).

I develop the playbooks using required Ansible modules to configure a new amazon VPC, network subnet, network gateway, security group, and create compute instances and map with the network, the Hadoop cluster variables can be either declared hardcoded or by specifying using a separate YAML file. The playbook is designed in such a way that the entire cluster will be deployed on execution. Hadoop deployment consists of four playbooks, for deploying EC2, master, slave, and client node, all the four playbooks are configured to execute in sequential order using the master file (hadoopclusterdeploy.yml), all dependency packages and the parameters are defined in the variable file will be installed at the time of implementation.

```
[root@AWX aws_hadoop]#
[root@AWX aws_hadoop]#
[root@AWX aws_hadoop]# tree
.
├── ansible.cfg
├── ansible.pem
├── cred.yml
├── hadoopcluster_deploy.yml
└── roles
    ├── ec2
    │   ├── tasks
    │   │   └── main.yml
    │   └── vars
    │       └── main.yml
    ├── hadoop_client
    │   ├── tasks
    │   │   └── main.yml
    │   ├── templates
    │   │   └── core-site.xml.j2
    │   └── vars
    │       └── main.yml
    ├── hadoop_master
    │   ├── files
    │   │   └── core-site.xml
    │   ├── tasks
    │   │   └── main.yml
    │   ├── templates
    │   │   └── hdfs-site.xml.j2
    │   └── vars
    │       └── main.yml
    └── hadoop_slave
        ├── tasks
        │   └── main.yml
        ├── templates
        │   ├── core-site.xml.j2
        │   └── hdfs-site.xml.j2
        └── vars
            └── main.yml

17 directories, 17 files
[root@AWX aws_hadoop]#
[root@AWX aws_hadoop]#
[root@AWX aws_hadoop]#
[root@AWX aws_hadoop]#
```

```
# ec2 deploy tasks
- name: Install boto & boto3 on local system
  pip:
    name: "{{ item }}"
    state: present
  loop: "{{ python_pkgs }}"

- name: Create Security Group for Hadoop Cluster
  ec2_group:
    name: "{{ sg_name }}"
    description: Security Group to allow all port
    region: "{{ region_name }}"
    aws_access_key: "{{ access_key }}"
    aws_secret_key: "{{ secret_key }}"
    rules:
    - proto: all
      cidr_ip: 0.0.0.0/0
    rules_egress:
    - proto: all
      cidr_ip: 0.0.0.0/0

- name: Deploy four EC2 instances on AWS
  ec2:
    key_name: "{{ keypair }}"
    instance_type: "{{ instance_flavour }}"
    image: "{{ ami_id }}"
    wait: true
    group: "{{ sg_name }}"
    count: 1
    vpc_subnet_id: "{{ subnet_name }}"
    assign_public_ip: yes
    region: "{{ region_name }}"
    state: present
    aws_access_key: "{{ access_key }}"
    aws_secret_key: "{{ secret_key }}"
    instance_tags:
      Name: "{{ item }}"
  register: ec2
  loop: "{{ instance_tag }}"

- name: Add 1st instance to host group namenode
  add_host:
    hostname: "{{ ec2.results[0].instances[0].public_ip }}"
    groupname: namenode
```

Figure 7: Ansible playbook structure and EC2 deploy Playbook

4.3 Design Kubernetes Cluster using AWX

Kubernetes is a powerful container-orchestration system used for efficient use of computing resources, applications can be containerized and deployed as pods in slave nodes and can scale resources when required in automated configuration settings. Deployment of fully functional Kubernetes clusters needs more work hours to implement, I developed the playbooks by defining the requirements and dependencies in YAML codes. The architectural diagram of the Kubernetes cluster is shown in Figure 8. Let us discuss the use of individual components below.

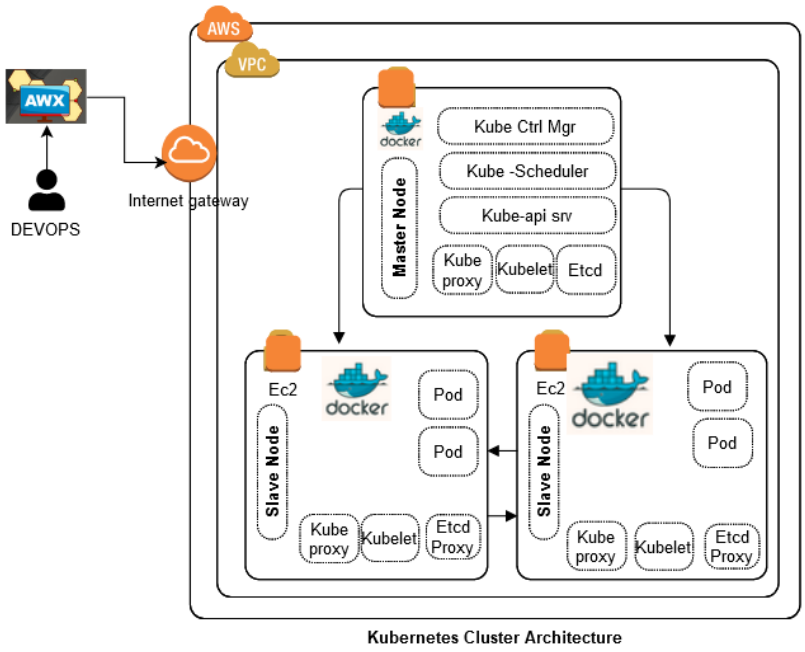

Figure 8: Kubernetes Cluster Architecture

- MasterNode - controls and manages the worker nodes
- Slave node - worker node where pods are running
- Kube Control Manager - Controlling cluster and its functions.
- Kube Scheduler - assign pods to the node
- Kube Proxy - maintain the network on nodes
- Kubelet - node agent communicate with control plane
- Pod - execution unit

Kubernetes cluster deployment consists of three playbooks, (1) deploying the Virtual private cloud, Security group, EC2 instance, (2) deploying the Kubernetes and docker packages, creating Kubernetes cluster, and generating authentication token for the slave nodes to join the cluster (3) install Kubernetes packages on slave nodes and add the nodes to the cluster.

5 Implementation

This section discusses the implementation of cloud resources and clusters, as designed in the previous section. create a user account in AWS and GCP with privileges to deploy cloud resources. Create host machine in Oracle VirtualBox with 2 Cores, 4G Memory and install centos 8.4 Operating system, configure python, pip, boto, Ansible, Docker, and AWX (17.1.0) packages on it. Configure the below steps on the AWX console:

1. Create the organization and inventory. Configure credentials for AWS, GCP, and local machines. Create a project folder for each cluster setup and map the related playbook.

2. Create individual Job templates and map the required credential and project for each deployment.

3. Configure the dynamic inventories for automatic discovery of cloud servers.

5.1 Provision Cloud Servers in AWS and GCP using AWX

Execute the EC2 deployment job from AWX console to implement compute instances as shown in Figure 9 to execute.

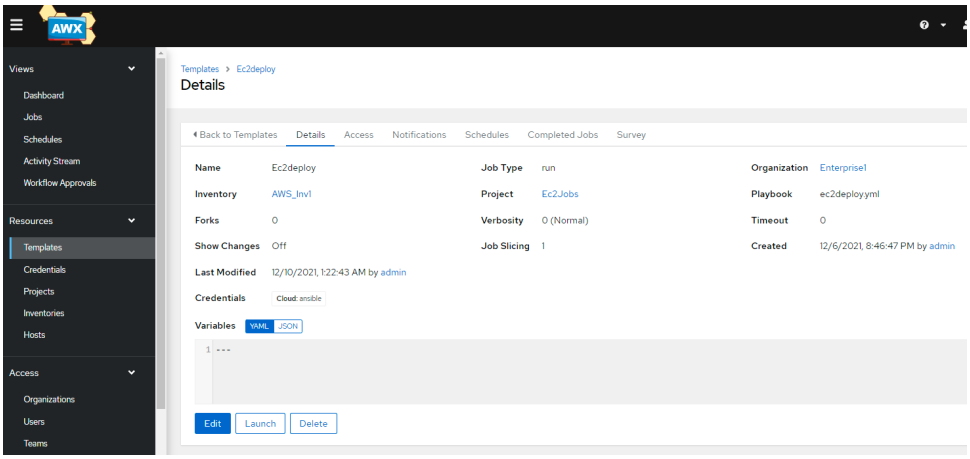

Figure 9: AWS Ec2 Deploy Job

Similarly, execute the GCP compute job to deploy the webserver from the AWX console. Job execution is similar to ec2 deployment. Verify the job completion status. Figure 10 shows the server status on GCP console.

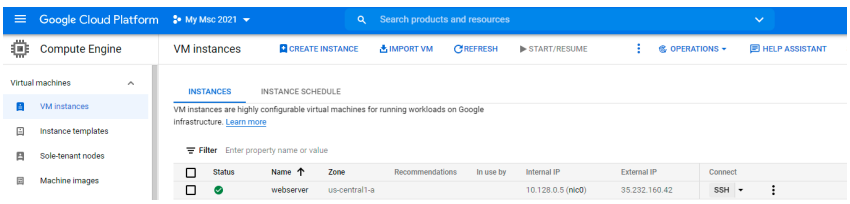

Figure 10: GCP instance deployed

5.2 Deployment of Multi-Node Hadoop Cluster using AWX

Let's provision the three-node Hadoop cluster using the playbook created on the design and deploy section, the playbooks configured to install three nodes in AWS, Once the servers are deployed, playbook update the hostnames as master, slave, upon completion of first, the second playbook will automatically trigger, it downloads and install Hadoop and java packages on these nodes and configure the Hadoop application by updating the configuration file defined in the playbook. Validate the final setup either by login into the AWS console or ssh into the cluster. The Job execution is shown below in the Figure 11

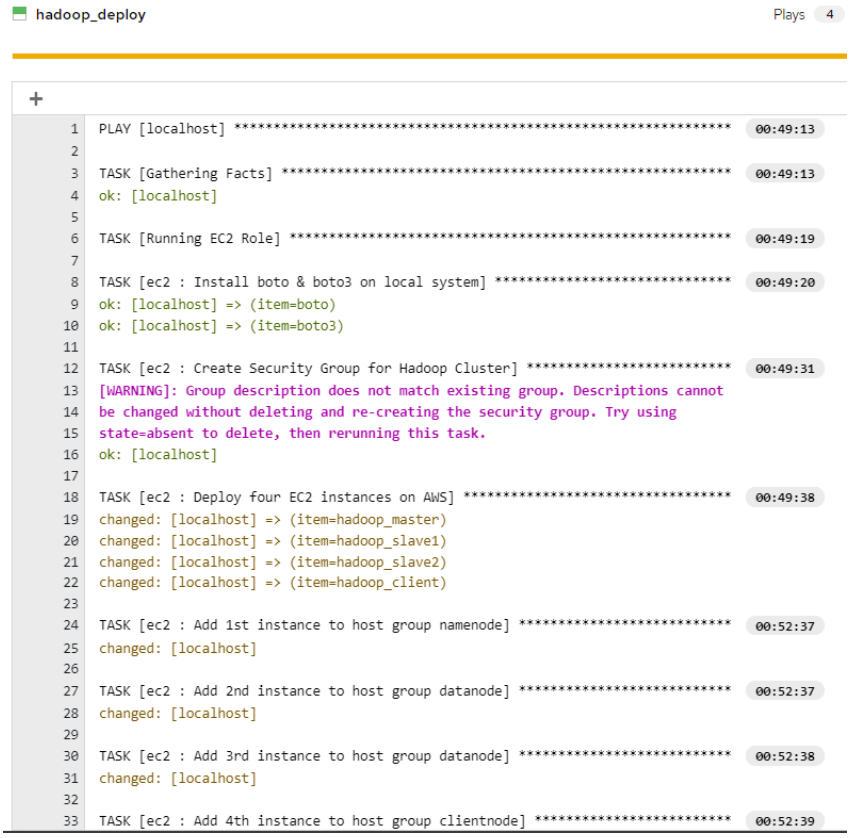

Figure 11: Ansible Playbook Execution Output

Figure 12 shows the provisioned resources in AWS console. Each server hostname is automatically set as configured in the playbook, each server is assigned with a public IP address, servers can be accessed externally using this IP.

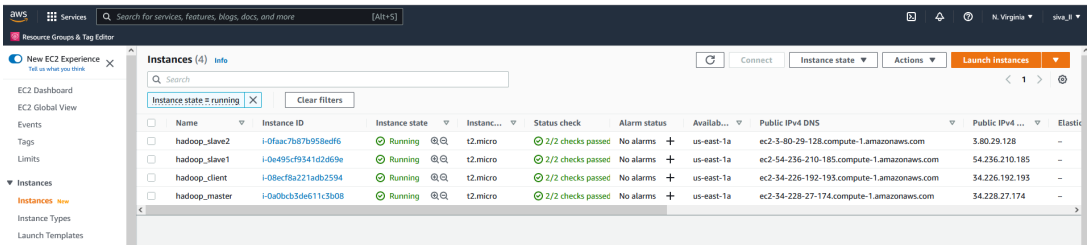

Figure 12: Hadoop Cluster deployed in AWS

5.3 Implementation of Kubernetes Cluster Orchestration using AWX

Deployment steps are similar for any system provisioning in AWX, A significant role is played by the Ansible playbook, as the complete set of instructions required for a specific cluster setup is defined in the form of codes, once the job template is triggered in AWX, the step-by-step execution of the instruction begins. Kubernetes cluster deployment job deploys four AWS ec2 instances and installs the docker and kubeadm packages on all the nodes. Upon completion of the playbook install and configure the Kubernetes cluster and install Flannel network service in the master node and create a security token for the slaves to connect to the master node. Similarly, the slave nodes are installed with the packages and access token and join to master to form a complete cluster. The entire setup is deployed directly from AWX, Cluster can be managed and controlled. Figure 13 shows the last lines of the execution log, It shows the total number of changes made on each server. Where the changes denote the individual tasks that were configured in the playbook to execute during deployment.

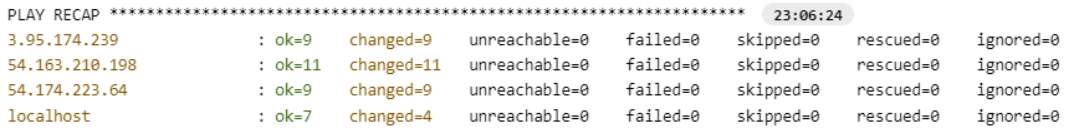

Figure 13: Kubernetes Playbook Execution Output

Figure 14 demonstrates that the control-plane is installed on the master node and all of the needed namespace services are running in the cluster. As a result, the Ansible playbook and AWX were used to set up a sophisticated Kubernetes cluster. AWX provides dynamic inventory functionality to automatically fetch the newly installed server information to its database.

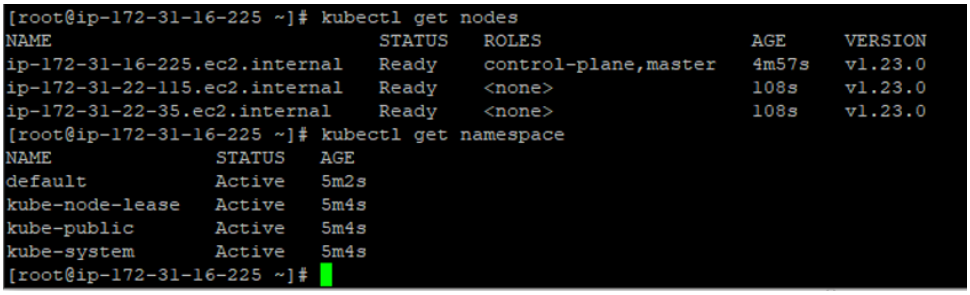

Figure 14: Kubernetes cluster validation

Additional playbooks are created for routine system administration tasks. These playbooks can be used to perform day-to-day operational tasks and can be configured as scheduled jobs to run on defined time. Jobs such as package install, security patching, server reboot, and cloud server removal are configured and placed under the operations project folder.

6 Evaluation

The experimental study of the proposed model and the key findings of the research work are presented in this evaluation section. The goal of this research project is to illustrate the overall IT infrastructure orchestration and management can be done using a single tool. The experiment involves the on-premise, public cloud, and private cloud infrastructures. I have taken standalone virtual machines for on-premise infrastructure and AWS, GCP cloud infrastructures for the research. Amazon Linux2 Operating system used for both Hadoop and Kubernetes cluster environments. Redhat Linux 8 operating system is used for AWS standalone web server setup and Centos7 is used for GCP webserver setup.

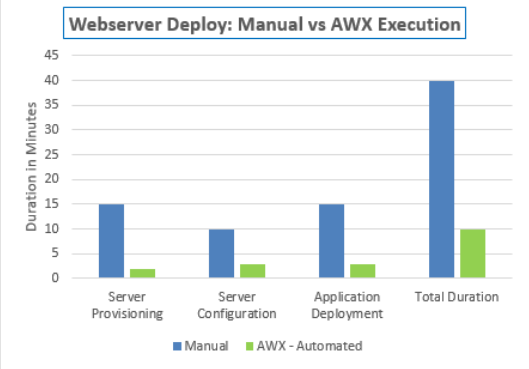

Figure 15: Time comparison of webserver deployment

6.1 Compare time taken to implement resources in GCP and AWS using manual and AWX

As denoted in the implementation phase the webservers and clusters are deployed in AWS and the GCP cloud environments using the playbooks created, from the AWX console. An investigation was conducted based on time duration taken for manual and automated AWX deployment, by taking into account all required dependency creations (VPC creation, security group, gateway) as a part server deployment procedure. Duration of each task plotted based on completion of each task in order, server provisioning task involves the creation of network and install of an operating system. Server configuration tasks involve up-gradation of operating system and installation of services, Application deployment tasks involve package installation, the configuration of specific settings to make the application work as expected.

Figure 15 depicts the experiment's findings, it shows the server provision task is 650 percent faster than the manual method, and the overall deployment of web server setup using AWX method is 300 percent faster than the manual deployment method in case of single server implementation with a simple application deployment without any complex configuration.

Figure 16 shows the time duration comparison of Hadoop and Kubernetes cluster deployment using manual and the automation method.Only difference in overall time duration is that the Hadoop cluster involves 4 servers, but the Kubernetes cluster consists of only 3 nodes. Execution time is plotted in the chart. Results show that the overall Hadoop cluster implementation using the AWX method is 600 percent faster than the

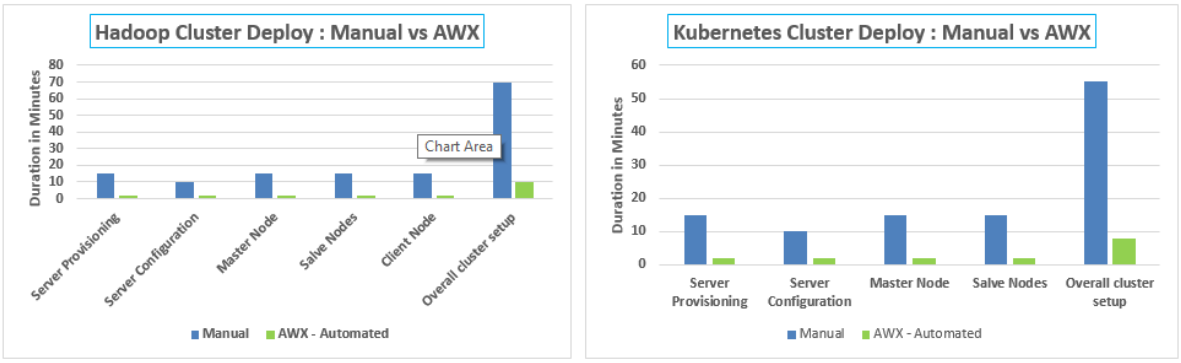

Figure 16: Time comparison of cluster deployment

manual method and the Kubernetes cluster implementation is 587 percent faster than manual deployment. Hence it's evident that using infrastructure as a coding methodology reduces the overall deployment time in any deployment. It is required to be noted that the deployment time reduces significantly if the installation involves complex configuration and more servers using the IAC tools.

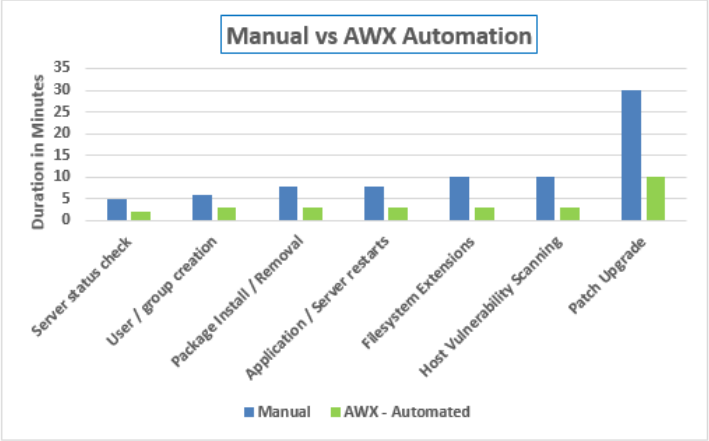

Figure 17: Time comparison of system administration tasks

6.2 Compare time taken to perform system administration tasks using manual and AWX

In this section, I evaluated the time taken to perform different system administration tasks using both methods. I have created a few playbooks for individual package installation, package removal, OS patching, and server status check. These basic tasks are carried out by the system administrator to keep the environment stable. These tasks are executed against in-house servers and the cloud servers and then taken to complete these tasks as captured and plotted in the Figure 17. The time duration is captured based on the single server execution of each task. It shows the patch upgrade task is 200 percent faster as compared to the manual method, other tasks look 150 - 250 percent faster compared to manual execution, it should be noted that execution of these tasks in multiple servers

take the same duration by using the AWX since AWX offers the parallel execution of tasks, which greatly reduce overall execution time.

Figure 18 shows the license cost comparison of Enterprise version of GUI based IAC solution offer by different vendors over Open-source AWX. Approximate cost of license is 100 dollars per node for a year.

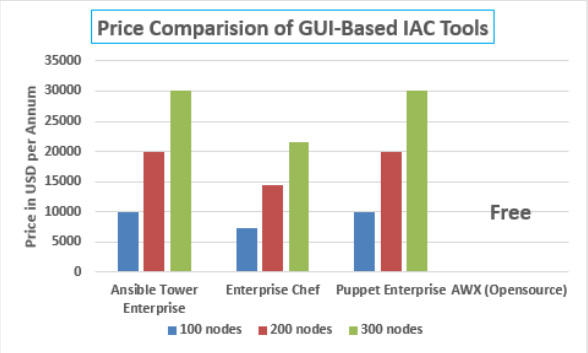

Figure 18: Price comparison of Enterprise version of IAC Tools

6.3 Discussion

This research evaluates the ability and interoperability of the AWX IAC tool in multi-cloud environments and provides centralized administrative control. Simple web servers and complex cluster environments are deployed to demonstrate the functionality of the tool. The experiments demonstrate the use of the IAC tool to reduce overall implementation time. The above experiment could be enhanced by involving several complex deployment tasks and advanced cloud components. Also, implementation of workflow template by combining several jobs template could be developed to orchestrate robust activity. This study analyzed the benefits of GUI-based Open source tools and their use-cases in overall system administration and orchestration.

7 Conclusion and Future Work

Automation plays a major role in recent infrastructure provisioning and administration, manual deployment methods are more complex. To some extend, CLI-based IAC solves the problem, however, deploying GUI-based automation to handle centralized deployment and management is mandated. This paper presents a centralized architecture using the AWX by integrating in-house, multi-cloud infrastructures. Results show deployment tasks take 60 to 70 percent less time compared to manual techniques. The proposed solution helps overcome delays in manual processing and control challenges. AWX provides centralized control for job execution, role-based access control, and workflow automation. On the other side, AWX Tool is a community-driven open source project, even though Ansible has strong community support, more expertise is needed to troubleshoot if an issue arises, also the development of the Ansible playbooks needs significant time and knowledge. In future work, I intend to explore the possibilities of the Ansible with AWX in the field of server-less computing, micro-services-based solutions, Fog, and Edge computing.

References

Ardagna, D. (2015). Cloud and multi-cloud computing: Current challenges and future applications, pp. 1–2.

Ardagna, D., Nitto, E. D., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C. S., Petcu, D., Gericke, A. and Sheridan, C. (2012). Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds, *2012 4th International Workshop on Modeling in Software Engineering, MiSE 2012 - Proceedings* pp. 50–56.

Artac, M., Borovssak, T., Nitto, E. D., Guerriero, M. and Tamburri, D. A. (2017). Devops: Introducing infrastructure-as-code, *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017* pp. 497–498.

Barhate, S. M. and Dhore, M. P. (2018). Hybrid cloud: A solution to cloud interoperability, *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018* pp. 1242–1247.

Bousselmi, K., Brahmi, Z. and Gammoudi, M. M. (2014). Cloud services orchestration: A comparative study of existing approaches, *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014* pp. 410–416.

Ghanam, Y., Ferreira, J. and Maurer, F. (2019). Emerging issues amp; challenges in cloud computing—a hybrid approach, *Journal of Software Engineering and Applications* **05**: 923–937.

Guerriero, M., Garriga, M., Tamburri, D. A. and Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry, *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019* pp. 580–589.

Hassan, W., Chou, T.-S., Pagliari, L., Pickard, J. and Tamer, O. (2019). Is public cloud computing adoption strategically the way to go for all the enterprises?, pp. 310–320.

Juopperi, M. (2017). Deployment automation with chatops and ansible.

Kaush, S. and Gupta, S. (2017). Implementation of open stack through ansible.

Khandelwal, A. and Kumar, A. (2020). (pdf) impact of open source software in research, *Scientific Papers* .

Kokuryo, S., Kondo, M. and Mizuno, O. (2020). An empirical study of utilization of imperative modules in ansible, *Proceedings - 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS 2020* pp. 442–449.

Kritikos, K., Zeginis, C., Iranzo, J., Gonzalez, R. S., Seybold, D., Griesinger, F. and Domaschka, J. (2019). Multi-cloud provisioning of business processes, *Journal of Cloud Computing* **8**.

Lavriv, O., Klymash, M., Grynkevych, G., Tkachenko, O. and Vasylenko, V. (2018). Method of cloud system disaster recovery based on" infrastructure as a code" concept, pp. 1139–1142.

Masek, P., Stusek, M., Krejci, J., Zeman, K., Pokorny, J. and Kudlacek, M. (2018). Unleashing full potential of ansible framework: University labs administration, *Conference of Open Innovation Association, FRUCT* **2018-May**: 144–150.

Palma, S. D., Nucci, D. D. and Tamburri, D. A. (2020). Ansiblemetrics: A python library for measuring infrastructure-as-code blueprints in ansible, *SoftwareX* **12**: 100633.

Pieplu, R. (2018). Ground control segment automated deployment and configuration with ansible and git, p. 2337.

Ranjan, R., Benatallah, B., Dustdar, S. and Papazoglou, M. P. (2015). Cloud resource orchestration programming: Overview, issues, and directions, *IEEE Internet Computing* **19**: 46–56.

Reshad, A. and Sinha, S. (2020). Open source software solution for small and medium enterprises, *International Journal of Computer Sciences and Engineering* .

Sandobalin, J., Insfran, E. and Abrahao, S. (2020). On the effectiveness of tools to support infrastructure as code: Model-driven versus code-centric, *IEEE Access* **8**: 17734–17761.

Shvetcova, V., Borisenko, O. and Polischuk, M. (2020). Using ansible as part of tosca orchestrator, *Proceedings - 2020 Ivannikov Ispras Open Conference, ISPRAS 2020* pp. 109–114.

Singh, N. K., Thakur, S., Chaurasiya, H. and Nagdev, H. (2016). Automated provisioning of application in iaas cloud using ansible configuration management, *Proceedings on 2015 1st International Conference on Next Generation Computing Technologies, NGCT 2015* pp. 81–85.

Sun, A., Ji, T., Yue, Q. and Xiong, F. (2016). Iaas public cloud computing platform scheduling model and optimization analysis, *International Journal of Communications, Network and System Sciences* **4**(12): 803.

Tamburri, D. A., den Heuvel, W. J. V., Lauwers, C., Lipton, P., Palma, D. and Rutkowski, M. (2019). Tosca-based intent modelling: goal-modelling for infrastructure-as-code, *Software-Intensive Cyber-Physical Systems* **34**: 163–172.

Tomarchio, O., Calcaterra, D. and Di Modica, G. (2020). Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks, *Journal of Cloud Computing* **9**(1): 1–24.