TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Karl Markus Jõgila 179916IVSB

AUTOMATED INFRASTRUCTURE DEPLOYMENT WITH ANSIBLE FOR SMALL SCALE WEB APP DEVELOPMENT COMPANY

Bachelor's thesis

Supervisor: Siim Vene
MSc

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl Markus Jõgila 179916IVSB

ANSIBLE-PÕHINE AUTOMAATNE TARISTUJUURUTS VÄIKESELE VEEBIARENDUSETTEVÕTTELE

Bakalaureusetöö

Juhendaja: Siim Vene
MSc

Tallinn 2020

Author’s declaration of originality

Author: Karl Markus Jõgila

18.05.2020

Abstract

The aim of the current thesis is to develop a solution, that can be used to automate the deployment of infrastructure in a cloud-based environment for a small-scale web app development company using Ansible. Using configuration management (CM) and the principle of infrastructure as code (IaC) will ensure a quick, secure and consistent infrastructure deployment. The thesis provides insight into mistakes made during infrastructure deployment, lifecycle management and analysis of available infrastructure solutions.

Theoretical analysis will examine the common mistakes made during the infrastructure deployment process and comparison of infrastructure services. During the practical part, an Ansible playbook will be developed, which will be used to deploy the infrastructure.

This thesis is written in English and is 93 pages long, including 5 chapters, 4 figures.

Annotatsioon

Ansible-põhine automaatne taristujuurutus väikesele veebiarendusettevõttele

Käesoleva diplomitöö eesmärk on arendada lahendus, mida saab kasutada automaatseks taristujuurutuseks pilvepõhises keskkonnas väiksele veebiarendusettevõttele kasutades Ansiblet. Kasutades konfiguratsiooni haldust ja infrastruktuur koodina printsiibi põhimõtteid, aitab tagada kiire, turvalise ja konsistense taristujuurutuse. Diplomitöö annab analüüsi infrastruktuuri juurutamisel levinud vigadest ning analüüsi saadaval olevatest infrastruktuuri teenustest.

Teoreetilise analüüsi käigus uuritakse enimlevinuid vigu infrastruktuuri juurutamisel ning analüüsi infrastruktuuri teenustest. Töö praktilises osas arendatakse välja Ansible skript, mida kasutatakse infrastruktuuri juurutamiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 93 leheküljel, 5 peatükki, 4 joonist.

List of abbreviations and terms

AWS	Amazon Web Services
Azure	Microsoft Cloud service offering
API	Application programming interface
HA	High availability
REST	Representational state transfer
VPC	Virtual Private Cloud
Network ACL	Network Access Control List
SQL	Structured Query Language

Table of contents

List of figures

1 Introduction

The work deals with the analysis of automating the process of configuring and deploying the infrastructure needed for a small-scale web application development company. The environment will be deployed in a cloud-based solution. Automating the entire process with a configuration management tool will make the deployment consistent, secure, portable and easy to manage. The work will give an analysis of most common mistakes made during the infrastructure deployment process in small businesses and start-ups and analysis of infrastructure services, that will be implemented in the practical part of the thesis.

The infrastructure in question will contain the following services:

- Web servers
- Databases
- Monitoring
- Backup and recovery

The topicality of the work and the need for such a solution are related to the authors work experience during which the author has experienced consequences of improperly or manually configured infrastructure services in start-ups or already established small businesses.

The result of this work should apply to a starting or already existing web application development company, who would like to increase the security and ease the management of their infrastructure.

The solution will be developed using the best security, coding and infrastructure practices so that it could be implemented for already existing companies working in the web application development field. In addition to that, since the solution will be open-sourced, the author will provide documentation and examples to make the deployment as easy as possible.

The research questions that the author will want to answer during the work will be:

1. What are the most common mistakes made during infrastructure deployments?
2. What are cyber security related mistakes done in infrastructure lifecycle management?
3. What are the optimal software solutions for the infrastructure services (web server, database, monitoring, backup)?

After the questions get answered, the author will develop the solution using the services which proved to be optimal.

2 Description of the problem

2.1 Overview of the situation and goals

Manual infrastructure deployment is a very time-consuming process, during which misconfigurations can happen. These mistakes can lead to security holes or systems not working as they should be. This means more time will be spent in the future debugging or redesigning the entire infrastructure, which could mean possible downtime for prolonged periods.

This thesis will be focusing mainly on start-ups and small businesses because as of 2018, 47% of small businesses in the United States suffered at least one cyber-attack per year. Of those, 44% experienced two, three or four attacks in the same period, and 8% had five or more attacks. The top cyber-related insurance claims were ransomware, hackers and loss or misuse of data. In addition to that, 7 out of 10 businesses are unprepared to deal with a cyber-attack. [9]

The most common mistakes start-ups and small businesses make with cloud infrastructure deployments and lifecycle management are:

- Committing credentials to version control systems like Git. [6]
- Improper access management. [6]
- Backup and restore procedures not correctly configured.
- Patch and vulnerability systems/procedures are not in place.

Automating the deployment will ensure consistency of the environment, reduce time-consumption, eliminate the possibility of misconfigurations and ensure a more secure and sustainable infrastructure. Configuration changes will also be faster and consistent, since the entire environment is described in code and managed by a configuration management tool. [2]

The principles what automating infrastructure deployments will help to achieve:

1. Ability to make improvements continuously, rather than doing everything at once at a specific time.
2. System Administrator(s) spending less time on routine and repetitive tasks.
3. Users can manage and define resources what they need without the help of a system administrator.
4. Systems and services can be recovered quickly and easily.
5. Configuration management tools will ensure consistency of different environments.

The outcome of the practical part of this thesis will help start-ups and small businesses to deploy their infrastructure to the cloud as soon as possible while maintaining the best security and infrastructure practices. Configuring the services in question securely with having no previous experience, would be a challenging task.

2.2 Problems in small-scale IT companies

Globally more than 95% of businesses are considered small (<100 employees). The most challenging tasks for creating IT infrastructure in small-scale IT companies are requirement analysis, financial and budget calculations and dealing with implementation and operational issues. Following that, the company has to understand the product that they are building, and the technology needed for it. [23]

In addition to that, starting a company can be a very stressful process, during which some parts of the process are inevitably rushed and then forgotten about. These parts can become huge problems if they are tied to the IT infrastructure. When the business owner is not an expert in the IT field and does not consider proper IT strategy planning a priority, scaling in the future can become a considerable challenge for the company.

The main security mistakes made in the cloud according to a research conducted by Netskope are:

- Granting public access to storage buckets.

- Using the root account for everyday activities.
- Failing to implement monitoring.
- Integration issues. [10]

When the company does not take these problems seriously, one day, they might be out of business because of a data breach, which could lead to huge fines. In 2017, there were approximately 1579 reported data breaches in the United States. [24]

Data breaches can be the result of a user error. Most typical case scenario is when an operator has misconfigured a platform or a server which has resulted in the ability of an external entity to gain unauthorised access to the data. Data breaches can affect millions of people's personal details, which in Europe, due to GDPR, can lead to massive fines. [24]

As cloud computing often simplifies the process of deploying IT infrastructure, many companies are moving their services to the cloud. However, users must understand the security concepts of their chosen cloud provider. [24]

Since starting companies tend to have limited resources, they leave the deployment and management of new applications in cloud infrastructure to developers. They see it as a less expensive solution compared to hiring an employee, who would be responsible for cloud infrastructure. [15]

When the application starts to gain popularity and becomes one of the primary income sources of the company, the underlying infrastructure becomes neglected as developers are focusing more on the application. Routine processes are left undone, proper monitoring is not implemented, and documentation is written. [15]

In addition to that, should there be problems with interconnectivity between instances or external services, improper network configuration could be done by developers, which could leave the entire infrastructure in a vulnerable state. As mentioned before, one of the worst-case scenarios would be to leave databases or AWS S3 buckets open to the public, which could lead to data breaches.

2.3 Data breaches caused by improperly configured IT infrastructure services

In October 2017, API data, authentication credentials, which included 40,000 plaintext passwords, certificates, decryption keys, configurations, customer information of Accenture customers were left open to public access due to a misconfigured AWS S3 storage buckets. Their customer list included 94 companies on the Fortune Global 100 list. [21][24]

BJC Healthcare reported that an unsecured server was left open to internet access between May 2017 to January 2018. It is reported that the breach affected information about 33,000 patients. Information such as patients driving licenses, insurance details, treatment documents were stored on the server. Personal data such as names, addresses, date of birth, telephone numbers and social security numbers were also vulnerable. [20][24]

2.4 Configuration consistency

As mentioned before, manual deployment of infrastructure services can be a long and complicated task. When deploying multiple environments at once, depending on the size and competence of the team working on it, the possibility of inconsistencies and misconfigurations is present. Undiscovered configuration mistakes, which are not detected on time, can lead to security breaches, systems not working or malfunctioning.

In case of disaster, the recovery of virtual machines or services will be a challenging task if proper backup and disaster recovery procedures are not in place. If the redeployment procedure is done with no automation, the process can take hours or days depending on the size of the infrastructure. In larger redeployments, misconfigurations and inconsistencies are highly likely due to the pressure of getting everything working again. Discovery of the mistakes made during that period can be extremely difficult to find and fix, which can increase the total downtime of the infrastructure.

If a company is using multiple cloud service providers, the importance of consistent infrastructure is critical. It reduces the risk of operational inefficiencies and enables seamless transitions between cloud service providers should there ever be a need to do that.

2.5 Time consumption of deployments

As stated before manual infrastructure deployment can be a very time-consuming process. If talking about the infrastructure that this thesis is focusing on, then the process of deploying that infrastructure manually can take multiple days in multiple environments. It is crucial to document the infrastructure setup as well for future sustainability which will also increase the time consumption of the entire deployment.

This work will focus on deploying the infrastructure in a cloud-based environment. The outcome of the practical part can be modified in the future to deploy the infrastructure in other environments.

3 Analysis of common mistakes made in infrastructure deployments and lifecycle management

The analysis part of this thesis will give an overview of the most commonly made mistakes during infrastructure deployment and lifecycle management. The following problems were chosen due to the cyber security impact they can have on businesses. In addition to that, the author has experienced the consequences of the problems personally during his work experience.

The chapter focuses on the following issues: backup and restore procedures, committing credentials to version control systems, improper access management and patch management.

3.1 Backup and restore procedures

Data is one of the most critical assets regardless of the size of the business, and there is always the risk of losing your data. Whether the data loss occurs because of a hardware failure, cyber-attack or improper access management, it will cause many problems to the company or in the worst-case scenario, shut down the company. Having a good backup strategy in place is essential for data security since there is no guaranteed method of preventing a data breach. [25]

Data loss can occur from:

- Accidental data deletion or modification.
- Hardware failure.
- Improper governance of access rights.
- Lost or stolen devices.

As companies are dependent on their data being present and secured, proper backup and recovery procedures must be implemented and tested regularly. Planning starts with identifying what data needs to be backed up and how regularly backups should happen. Data such as database backups should be backed up at least once a day.

The most common mistakes start-ups and small businesses make with backups are:

- Relying on people making manual backups. [19]
- No recovery procedures present.
- No policies set for cloud storage. [22]
- Encrypting backups.

Relying on people making regular backups is not a sustainable backup strategy. Start-ups and small businesses tend to have one key person who would be responsible for this procedure. Should there be a need to restore data and if the key person happens to be on holiday without having written any documentation on the recovery procedure, the company could be in huge trouble.

As more companies are storing their data in the cloud, proper governance and access management must be implemented. Failing to do that can result in unauthorised access, data breach and data loss. In addition to that, critical storage that is stored in cloud storage should be encrypted by default.

Since the entire infrastructure will be deployed and managed by Ansible, there is no need to make backups of the configuration files. Source code of the applications that are present on the systems should be stored in version control systems, which means there is no need to make backups of those. However, automated database backups have to be configured. Ideally, the automated backup should backup all of the databases that are present on the system and keep them for 30 days. If needed, backups older than 30 days could be stored in long-term storage.

The most efficient way of making a backup of the databases is to use a built-in database export feature. For example, in MySQL databases, the feature is called mysqldump, and in PostgreSQL, the feature is called pg_dump.

3.2 Committing credentials to version control systems

One of the most common ways of storing source code of applications is using public version control systems like GitHub and GitLab. If developers are not paying attention to what they are uploading to these public environments, they can upload secrets by mistake. Secrets such as API keys, encryption keys, OAuth tokens, certificates, PEM files, passwords and passphrases can be used to access other resources which are managed by the company. [18]

Should the repository include secrets or credentials for resources such as AWS S3 buckets, databases, customer relationship management systems or something similar, then it would leave the company in a vulnerable state for a data breach. The only way of removing secrets is to replace the current repository with a new one.

Creating a new repository is essential since there are tools that available such as Gitrob, that are used to find potentially sensitive files pushed to the public repositories on GitHub. It can clone repositories belonging to an organisation down to a configurable depth and iterate through the commit history and in the process, mark potential sensitive files. [16]

As infrastructure as code principle follows practices from software development, the source code of the solution should be stored in a version control system such as Github. During the initial setup of the solution, the user is prompted to enter their cloud service provider authentication credentials, which will be stored in an encrypted form using ansible-vault. The file will not be pushed to the remote repository since the credential file will be ignored by default with. gitignore.

3.3 Improper access management

Proper access control is one of the most critical components of cloud security. There should be minimal access to the cloud platform itself, and those who have access to it should have the minimum access rights to carry out their assignments. To maintain proper security, privileged access protocol must be implemented. [7]

When start-ups or small companies start using AWS S3 storage without prior knowledge, they can configure the storage bucket with too public access rights. One of the biggest mistakes a cloud user can make is configuring their storage bucket with "authenticated users" access, thinking that these are users that belong to the organisation or to the relevant application. That is incorrect, as "authenticated users" refer to anyone with AWS authentication, which is any AWS customer. This misunderstanding can leave storage objects to fully exposed public access. [6]

As developers should have the minimum amount of access to the cloud console, the author will develop a playbook for the creation of developers' accounts. The playbook will create user accounts on the EC2 instances and add their SSH public keys to their account, so users can access the system without using passwords.

3.4 Patch management

The primary purpose of patching is to ensure a stable and secure environment. The objective of a patching strategy is to maximise performance, mitigate security issues and improve system availability. As some systems might be exposed to newly discovered security vulnerabilities, it is critical to address those issues as soon as possible. Otherwise, systems that are dependent on it might perform sub-optimally or deteriorate in terms of performance.

Patching should be done regularly, for example, every 30 days and executed automatically. During the process security and bug-fixes are installed on the system. Patches should be applied first in development or test environments to make sure that systems are not affected by the changes made during the process. Should any issues arise after the process is complete, they could be dealt with, and fixes could be applied before patching happens in the production environment.

One of the main benefits that proper patch management offers is network security. Unfortunately, patch management is usually implemented after a company has experienced a data breach and wants to ensure that other businesses data remains untouched. By securing the network beforehand will help to prevent data theft, legal issues and reputational damage. [1]

Another benefit of patch management is compliance. As the number of security breaches is increasing, the number of regulations that companies must follow is also increasing. That forces companies to implement best security practices. Failure to comply with the regulations can potentially lead to legal penalties. [1]

One of the main mistakes' companies make is not keeping their software up to date. Hackers are always looking for vulnerabilities that can be exploited and used to access the company's internal data and resources. Implementation of patch management on both servers and workstations is an essential part of data security.

The solution will include a playbook, which will patch the instances every 30 days. There will be two separate patching jobs for web servers and database instances. These jobs will be configured to be run every 30 days using Cron on the Ansible control node.

3.5 Instance deployment

The underlying operating system that will be running on the instances is Ubuntu. It is the most popular OS for running websites. It is based on Debian and can be downloaded from Ubuntu's official website. The version that will be used is 20.04 LTS (code name Focal Fossa), which will be receiving updates until 2030. [12] [13]

Automating the deployment of instances helps to ensure the consistency of the environment, eliminates the need of deploying from the AWS Console, reduces the time required to apply the initial configuration on the instance.

In companies where EC2 deployment is an everyday occurrence, it will help to save a considerable amount of time spent on that task. Developers can quickly provision new instances without the need for a system administrator.

3.6 Web server deployment

The main principles that automating the web server deployment will help to achieve:

- Consistency of the configuration files.
- Faster deployment of new web applications.
- Developers being able to deploy new virtual hosts without the need of a system administrator.

While the pros of automating web server related tasks overweigh the cons of configuring everything manually, there still are some tasks that might require manual interference.

For example, adding new configuration file templates, changes to existing configuration files, creation and modifications to the custom configuration files, that are not created using standard deployment templates.

The main requirements for the web server are:

- Lightweight.
- High performance.
- Available from the default repositories.

In this thesis, Nginx will be used as the web server software because of the lightweight structure and ability to handle high traffic websites when compared to Apache. When comparing the complexity of configuring Apache and Nginx, Apache is a clear winner. That makes the automation of Nginx deployments even more critical because configuration mistakes are likely to happen.

3.7 Database deployment

The main principles automating the database deployment will help to achieve:

- New databases are created without using the command line.
- Database replication is set up in a consistent matter.
- During the installation process, unused users and databases are removed by default, during the manual installation process that has to be done manually.
- Database backups are configured by default.

When a new database has to be created, a user has to login into the management database and execute a command to create the database. When replication is also needed, that will require extra steps on two or more database instances. Automating that process will help to save the time spent on the database creation and the replication setup process. In addition to that, the configuration of both the database and the replication will be done consistently.

When there is a need to restore from the backup, that process can also be automated to ensure a consistent database recovery. If there are additional steps that need to be performed before the recovery, the other option is to run the restore command manually, which will be provided by the author.

The main requirements for the database are:

- Relational database structure to prevent data duplication.
- User management is available.
- Easy to configure.
- Data replication must be available.
- Available from default repositories.

The author considered using the following databases: PostgreSQL, MySQL and MariaDB. Out of the three MySQL was chosen. As of March 2020, it is the second most popular database software behind Oracle, while PostgreSQL is 4th MariaDB is 13th. When compared to PostgreSQL, the performance and more straightforward configuration outweigh the advanced features that PostgreSQL offers. In addition to that, Ansible has a module available for setting up MySQL replication. [8] [14]

3.8 Monitoring deployment

The main benefits of automating the monitoring solution deployment and configuration will help to achieve the following principles:

- Quick and consistent deployment.
- Configuration of the server/agents can be managed centrally.
- Adding new hosts will require no manual input.

The main requirements for the monitoring solution are:

- Highly customisable.

- Integration with visualisation tools must be available.

The author considered using the following monitoring solutions: Zabbix, Prometheus and Nagios. Out of the three solutions, Zabbix will be used. The powerful and modern web GUI is outweighing the features of Nagios. Easy and fast configuration process is more important than the performance gains which Prometheus is offering.

3.9 Backup and restore

The main benefits of automating the monitoring solution deployment and configuration will help to achieve the following principles:

- Restore process requires no manual intervention.
- Backups are configured consistently on instances.
- In complex setups, less time spent on backup master/agent configuration debugging.
- Does not rely on people doing backups manually.

While manual backups of databases can be done, it is not a sustainable strategy. People make mistakes and might sometimes forget about making a backup of the database or some other system. Automating that process will ensure that backups will be made regularly and consistently. The solution will automatically configure backups for the database servers. Backups of the configuration files will not be needed, as mentioned before, Ansible will configure the services, and the source code of the playbook is kept on a separate machine and ideally in VCS as well.

In complex backup systems such as Bacula or Bareos, managing configuration files manually can take a lot of time due to the complexity of the system. If there is an issue with the configuration which will cause the backups to fail, debugging that issue can be a very long and stressful task.

The main requirement for the backup solution is:

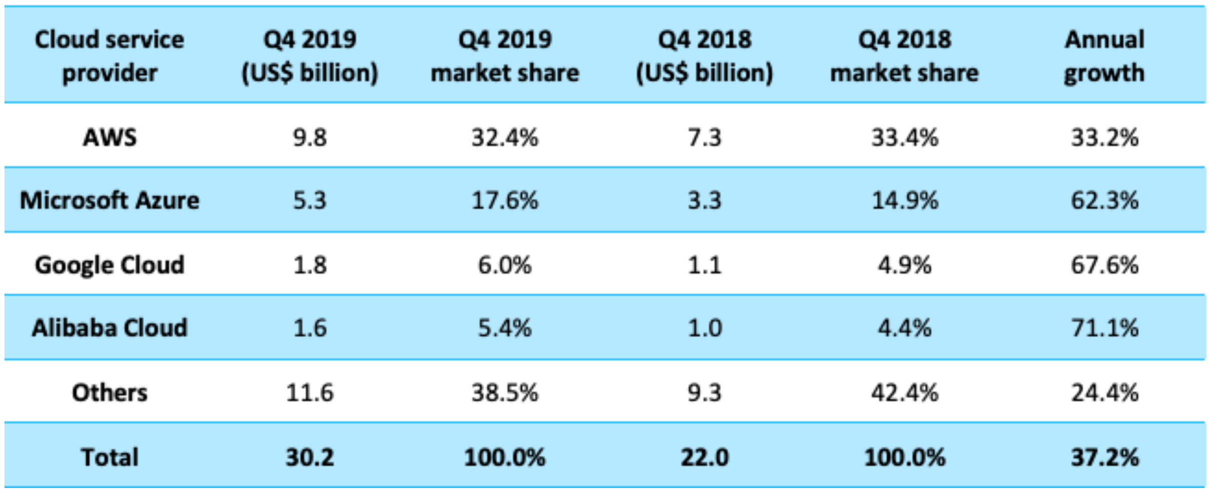
- Cloud storage must be supported.

Borg, Bareos and Duplicity fit the criteria. Duplicity will be used in this thesis because the only purpose of the solution is to store database backups, and the advanced features of Bareos are not needed.

3.10 Cloud infrastructure service providers

Over the past few years usage of cloud computing has grown massively. In 2019, public cloud adoption grew while private cloud use declined. Companies are increasing their investments in the public cloud. With a higher number of enterprises using Azure as their cloud service provider, they are closing the gap between the leader AWS. For the past three years, managing cloud costs has been one of the top priorities for companies using cloud service providers. [3]

As of Q4 2019, the worldwide cloud infrastructure services market reached a record high as spending grew by 37% to over US$30 billion. AWS remained the dominant cloud service provider, accounting for 32% of total spend. Azure increased their share from 15% to 18%. Google cloud was the third-largest service provider with a 6% share. [4]



Cloud service provider	Q4 2019 (US$ billion)	Q4 2019 market share	Q4 2018 (US$ billion)	Q4 2018 market share	Annual growth
AWS	9.8	32.4%	7.3	33.4%	33.2%
Microsoft Azure	5.3	17.6%	3.3	14.9%	62.3%
Google Cloud	1.8	6.0%	1.1	4.9%	67.6%
Alibaba Cloud	1.6	5.4%	1.0	4.4%	71.1%
Others	11.6	38.5%	9.3	42.4%	24.4%
Total	**30.2**	**100.0%**	**22.0**	**100.0%**	**37.2%**

Figure 1 Worldwide cloud infrastructure spending and annual growth (Source: Canalys Cloud Channels Analysis)

The main advantages of using cloud infrastructure services are:

- Flexible up-front investment costs.
- Frequent and easier product upgrades.

- Reduced IT support performed by internal resources.
- Community of users for the latest versions and features.
- Efficient for multi-tenant usage (scalability, recoverability, patching, security). [26]

The main disadvantages of using cloud infrastructure services are:

- Costs can get out of control when proper governance is not implemented.
- A 3rd party could access data.
- Long-term projects costs must be in place.
- During hardware failure, you have no control over the recovery process.
- Significant learning curve, adequately trained personnel is expensive.

As mentioned before, cloud cost management and cloud governance are the top challenges for companies regardless of cloud maturity. Another challenge is managing software licenses that are used in public cloud environments. Specifically, understanding the cost implications of licensed software running in the cloud, ensuring that they are following the rules and the complexity of license rules in the public cloud. [3]

In this thesis, AWS will be used because of the popularity and great integration with Ansible. Configuring the services in question properly while having no previous experience would be a challenging task. There is also an option to use Amazon Lightsail for a deploying a LAMP stack. While being very easy to set up, it does not offer any monitoring or backup services by default. [5]

EC2 compute instances will be used in this thesis to deploy services. Reason being more cost-effective, having control over server-side configuration, possibility to launch playgrounds, where developers can test new solutions, without having to interfere with test and production environments. Should the company reach a point where there is a need to migrate to an on-premise setup or a hybrid-cloud solution, they could use the same solution to configure infrastructure services.

In the AWS marketplace, there are already similar CloudFormation templates available. However, the majority of them are doing the same deployment as Amazon Lightsail, and the same problem persists, backups and monitoring are not implemented by default. In addition to that, CloudFormation templates can only be used in AWS, while the Ansible playbook could be used elsewhere as well.

4 Implementation of the solutions

4.1 AWS infrastructure diagrams

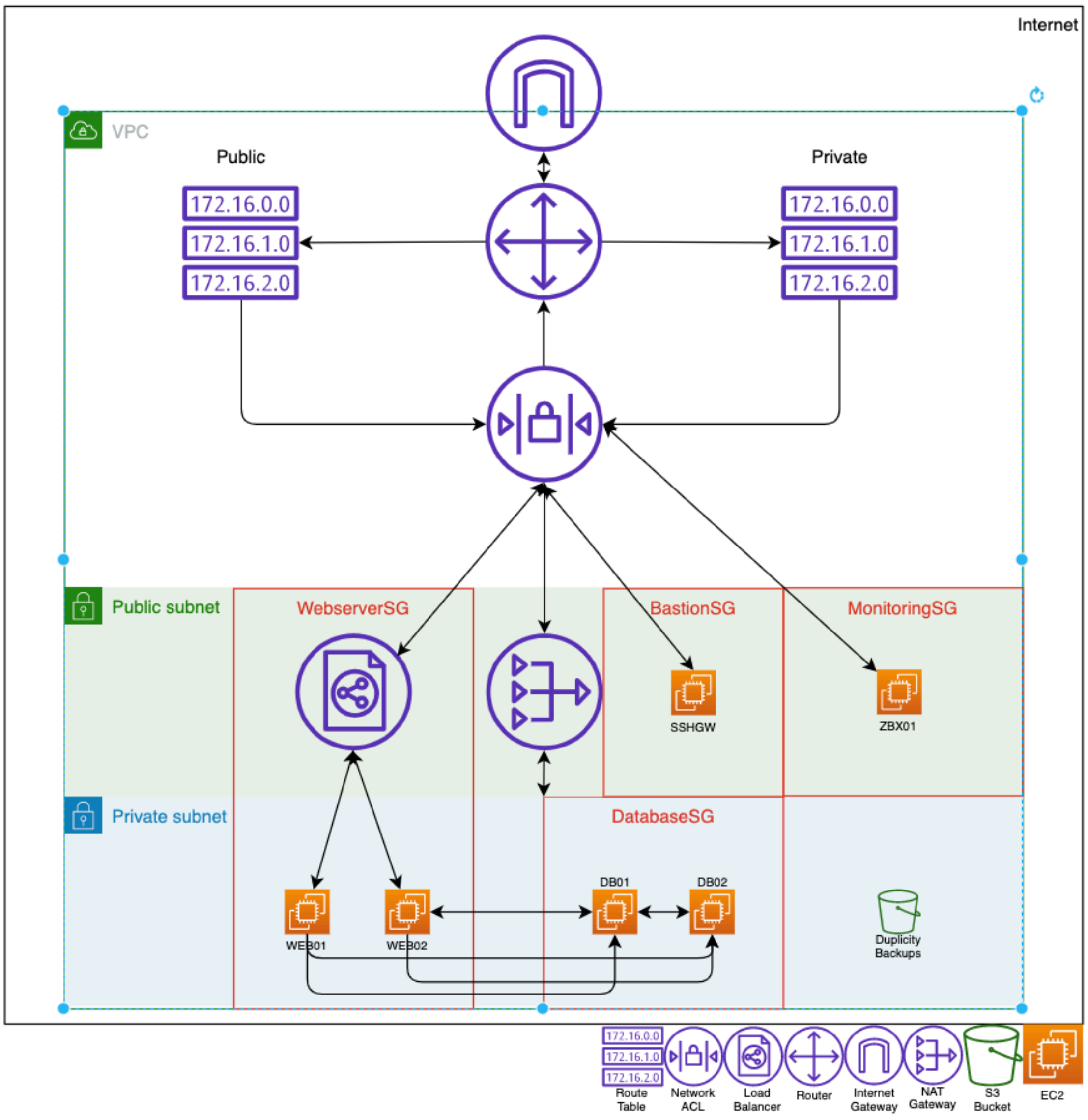


Figure 2 AWS Infrastructure Schema with a load balancer (Source: Author created)

The solution will be able to deploy the infrastructure, which is drawn in the figure above. It consists of 2 web servers located in the private subnet, which are load-balanced by the Elastic Load Balancer. 2 database servers with a master-slave configuration, one monitoring server and 1 Bastion host.

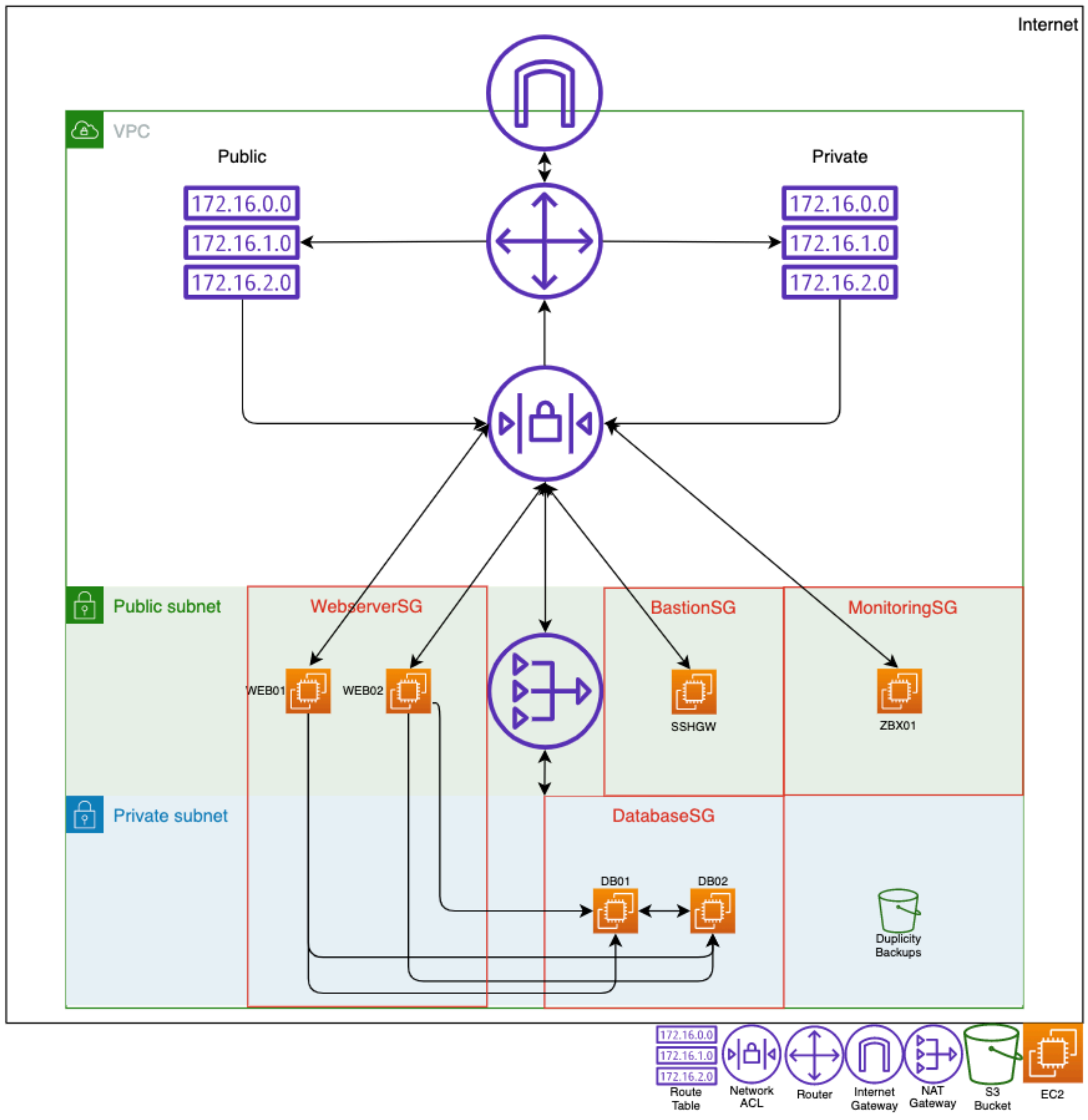


Figure 3 AWS Infrastructure Schema without a load balancer (Source: Author created)

The solution will also be able to deploy the infrastructure, which is drawn in the figure above. This particular set up does not deploy a load balancer for the webservers. Instead, they can be used as two different web servers. Both are placed in the public subnet and have static IP addresses.

4.2 Overview of the automated deployment process in AWS

The automated deployment workflow will be:

1. Virtual Private Cloud is created, during which the default route table, network access control list, and security group are added.

2. Private subnet is created.
3. Public subnet is created and configured. Servers that are connected to that subnet will automatically receive a public IP.
4. Internet Gateway is created and attached to the VPC.
5. Public route table is created, and routes are configured.
6. Public subnet is associated with the public route table.
7. Security group for web servers is created and configured.
8. Security group for database servers is created and configured.
9. Security group for monitoring server(s) is created and configured.
10. NAT Gateway is created and configured for the private subnet.
11. Route to the internet is added to the default route table via NAT Gateway.
12. Bastion instance is deployed and configured with a static IP.
13. Monitoring instance is deployed and configured with a static IP.
14. Database instances are deployed and configured.
15. Web server instances are deployed and configured.
16. S3 bucket for backups is configured.
17. If specified, the load balancer will be configured for the web servers.

4.3 Setting up Ansible Control node

All Ansible playbooks and commands are run from the control node. The node in question must have connectivity to all servers in the infrastructure. The control node can be a laptop, a shared desktop or a virtual machine. However, there is no possibility to use a Windows machine as a control node. The solution, in that case, would be to install a virtualisation software on the machine and launch a virtual machine.

The author in this thesis, is using a virtual machine and the operating system that is used is Ubuntu 18.04. The virtual machine in question has 1 vCPU and 1GB of RAM. To install Ansible and the dependencies on the system using the same operating system, the following commands have to be executed.

```
$ sudo apt update && sudo apt install software-properties-common -y &&
sudo apt-add-repository --yes --update ppa:ansible/ansible && sudo apt
install ansible python-pip python3-pip -y
```

```
$ pip install botocore boto boto3 zabbix-api
```

```
$ pip3 install botocore boto boto3 zabbix-api
```

Once the required packages are installed, the playbook has to be downloaded from github.com. The easiest way to do it is to use git clone command.

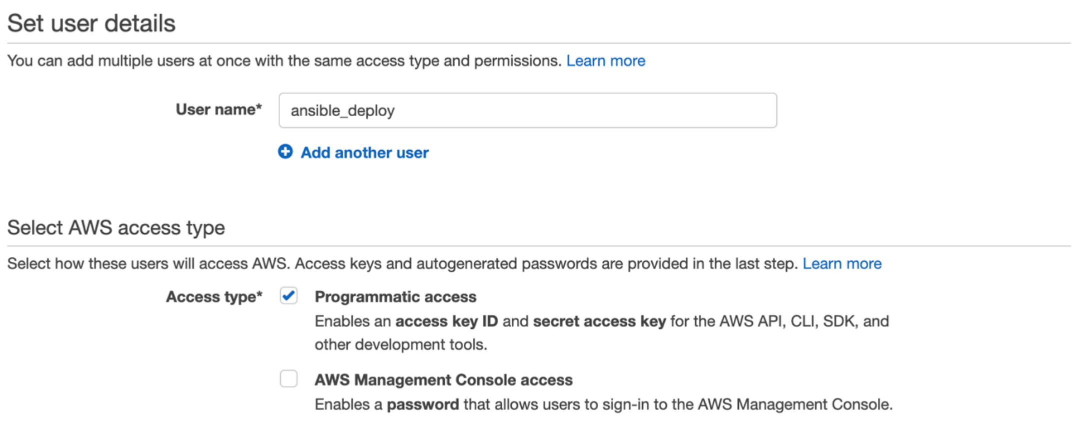
```
$ git clone https://github.com/karlmjogila/thesis-aws.git
```

```
$ cd thesis-aws/
```

After cloning the repository and changing the directory, the user must create an account in the AWS Console. The user must have full access to resources which are used during the deployment. To create the users, log in to AWS console and navigate to IAM -> Users -> Add User.

User creation process:

1. Create a username and make sure access type is “Programmatic access”



Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* ansible_deploy

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* [✓] Programmatic access
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[ ] AWS Management Console access
Enables a **password** that allows users to sign-in to the AWS Management Console.

Figure 4 AWS IAM (Source: Author created)

2. Select the "Attach existing policies directly" tab and choose the following policies: AmazonEC2FullAccess, AmazonVPCFullAccess, AmazonS3FullAccess, AmazonRoute53FullAccess, AWSCertificateManagerFullAccess.
3. Click Next -> Next -> Make sure the correct policies are applied -> Create user.
4. Make sure to download the .csv file containing both access keys and store the credentials in a password manager. They will be needed later.

Back on the control node, execute the following command:

```
$ ansible-vault create keys
```

The user will have to create a password, which will be used to encrypt the credential file. Once the password is created, an editor is opened, and the user has to the store the access keys created in the previous step. The Github repository contains an example credentials file, where the user has to fill in the values. The credentials are stored in a key-value format.

```
AWS_SECRET_ACCESS_KEY: <Secret access key>

AWS_ACCESS_KEY_ID: <Access key id>
```

Once the file is populated with values, exit the editor and execute the following command:

```
$ ansible-playbook site.yml –-ask-vault-pass
```

The user will be asked for the password, which was used to encrypt the credential file. After that, the deployment process will start and once finished; the infrastructure is deployed in AWS.

4.4 Development of the web server deployment

During the development, the author wanted to make sure that new virtual hosts can be added in group variables, after which the configuration and deployment of the virtual host is automatically completed.

The deployment workflow is as follows:

1. Package list is updated on the server.
2. Nginx package is installed.
3. Nginx is configured to start on bootup.

When a new virtual host is added to the list, the webroot directory is created, the configuration file is created, configuration check will be run, and if the check passes, Nginx is reloaded, and the virtual host is active (see Appendix 7 for Nginx deployment). If the load balancer is deployment is not specified, Let's Encrypt certificate will be configured as well (see Appendix 9 for Let's Encrypt deployment).

4.5 Development of the database deployment

During the database deployment, MySQL replication is configured (see Appendix 8 for Ansible database deployment). The deployment workflow is as follows:

1. Package list is updated on the servers.
2. MySQL package is installed on the servers.
3. Service is started and configured to start on boot.
4. MySQL root password is updated.
5. Unused users and databases are removed.
6. Databases and database users are created.
7. Replication users are created.
8. Slave is configured for replication.
9. Replication is started.

4.6 Development of the monitoring deployment

The primary purpose of the monitoring solution is to alert the responsible personnel about unavailable services or high consumption of instance resources. During the deployment, the following tasks are executed:

1. Zabbix repository is added to the server.
2. Package list is updated.
3. Zabbix and MySQL database packages are installed.
4. Zabbix and MySQL services are configured and started automatically on boot.
5. Services are started on the server. (see Appendix 2 for Ansible Zabbix deployment)

Instances will be added to Zabbix using the zabbix_host module in Ansible. [11]

4.7 Development of the database backup deployment

Since the infrastructure is described in code, the backups of the configuration files are not required since, in theory, everything should be managed by Ansible. The only thing that needs to be backed up frequently is the databases. This thesis will be using mysqldump script and duplicity to store the backup in S3 bucket (see Appendix 10 for Ansible backup deployment).

The deployment workflow is as follows:

1. Duplicity with required python modules are installed on database servers.
2. Backup script will be created from a template file and uploaded to the database servers.
3. Cron job is configured to back up the databases once a day.

5 Summary

The goal of the work was to analyse infrastructure services and common mistakes made in infrastructure deployment and lifecycle management to develop a solution, which could be used to automate infrastructure deployments for a small-scale web app development company in a cloud-based environment.

In the analysis section author compared different cloud service providers, analysed common mistakes made in infrastructure deployment and lifecycle management in start-ups and small businesses and analysed infrastructure services to find the optimal ones to be used in the solution.

In the practical section, an Ansible playbook was developed for infrastructure deployment automation, which helps to keep future deployments consistent. Infrastructure services that proved to be the optimal ones in the analysis section were implemented in the solution. Common lifecycle management and deployment mistakes were also handled.

The solution deploys the following services:

- 2 Web servers – Nginx is used as the web server software.
- 2 Database servers – MySQL is used as database software.
- 1 Monitoring server – Zabbix is used as a monitoring solution.
- 1 AWS S3 bucket for database backups – Duplicity is used to sync the data.
- 1 Bastion host for accessing servers in the VPC.

The solution was developed using the best security, infrastructure and coding practices. Documentation with examples is also included for easier customisation. In addition to that, an additional playbook is included to deploy the environment locally on the developer's workstation. The solution is available at https://github.com/karlmjogila/thesis-aws.

References

[1] What Is Patch Management? [Online]. Available: https://consoltech.com/blog/patch-management/

[2] Evaluation of Infrastructure as a Code for Enterprise Automation. 2018 [Online]. Available: https://is.muni.cz/th/liwzz/IaC-Final-el.pdf [Accessed 10 April 2020]

[3] Cloud Computing Trends: 2019 State of the Cloud Survey. 2019 [Online]. Available: https://www.flexera.com/blog/cloud/2019/02/cloud-computing-trends-2019-state-of-the-cloud-survey/ [Accessed 10 April 2020].

[4] Cloud market share Q4 2019 and full-year 2019. 2020 [Online]. Available: https://www.canalys.com/static/press_release/2020/Canalys---Cloud-market-share-Q4-2019-and-full-year-2019.pdf [Accessed 10 April 2020].

[5] Amazon Lightsail. [Online]. Available: https://aws.amazon.com/lightsail/ [Accessed 27 April 2020].

[6] 5 Common Cloud Configuration Mistakes. [Online]. Available: https://www.darkreading.com/cloud/five-common-cloud-configuration-mistakes/a/d-id/1335768 [Accessed 29 April 2020].

[7] 7 Security Mistakes Organisations Make When Adopting Cloud. [Online]. Available: https://www.cloudreach.com/en/resources/blog/7-security-mistakes-organizations-make-when-adopting-cloud/ [Accessed 29 April 2020].

[8] mysql_replication – Manage MySQL replication. [Online]. Available: https://docs.ansible.com/ansible/latest/modules/mysql_replication_module.html [Accessed 6 May 2020].

[9] 2018 HISCOX Small Business Cyber Risk Report [Online]. Available: https://www.hiscox.com/documents/2018-Hiscox-Small-Business-Cyber-Risk-Report.pdf [Accessed 27 April 2020].

[10] Top 10 AWS Security Mistakes and Solutions. [Online]. Available: https://resources.netskope.com/cloud-security-solution-white-papers/top-10-aws-security-mistakes-and-solutions [Accessed 5 May 2020].

[11] zabbix_host – Create/update/delete Zabbix hosts. [Online]. Available: https://docs.ansible.com/ansible/latest/modules/zabbix_host_module.html [Accessed 5 May 2020].

[12] Usage statistics of Linux for websites. [Online]. Available: https://w3techs.com/technologies/details/os-linux [Accessed 25 April 2020].

[13] The Ubuntu lifecycle and release cadence. [Online]. Available: https://ubuntu.com/about/release-cycle [Accessed 25 April 2020].

[14] DB-Engines Ranking. [Online]. Available: https://db-engines.com/en/ranking [Accessed 25 April 2020].

[15] Are Developers Running Your Infrastructure? [Online]. Available: https://www.rhythmictech.com/white-papers/are-developers-running-your-infrastructure/ [Accessed 21 April 2020].

[16] Gitrob: Putting the Open Source in OSINT. [Online]. Available: https://github.com/michenriksen/gitrob [Accessed 29 April 2020].

[17] Redis vs MySQL - A quick database comparison. [Online]. Available: https://tableplus.com/blog/2018/10/redis-vs-mysql-database-comparison.html [Accessed 10 April 2020].

[18] Why (and how) you should manage secrets outside version control. [Online]. Available: https://www.datree.io/resources/secrets-management-aws [Accessed 29 April 2020].

[19] Five common backup mistakes start-ups make. [Online]. Available: https://www.startupdonut.co.uk/blog/18/06/five-common-backup-mistakes-start-ups-make [Accessed 29 April 2020].

[20] BJC Healthcare data breach, 33,000 affected. [Online]. Available: https://www.scmagazine.com/home/security-news/data-breach/bjc-healthcare-data-breach-33000-affected/ [Accessed 20 April 2020].

[21] Accentuate the negative: Accenture exposes data related to its enterprise cloud platform. [Online]. Available: https://www.scmagazine.com/home/security-news/data-breach/accentuate-the-negative-accenture-exposes-data-related-to-its-enterprise-cloud-platform/ [Accessed 20 April 2020].

[22] 10 Data Security Mistakes Startups Can't Afford to Make. [Online]. Available: https://www.stamfordadvocate.com/news/article/10-Data-Security-Mistakes-Startups-Can-t-Afford-8318455.php [Accessed 29 April 2020].

[23] Information Technology Services Issues and Challenges with A Case Study in Small Medium Enterprises. [Online]. Available: https://www.researchgate.net/publication/308009820_Information_Technology_Services_Issues_and_Challenges_with_A_Case_Study_in_Small_Medium_Enterprises [Accessed 20 April 2020].

[24] Data Breaches Caused by Misconfigured Servers. [Online]. Available: https://www.scmagazine.com/home/opinions/data-breaches-caused-by-misconfigured-servers/ [Accessed 20 April 2020].

[25] The Importance of Backup and Recovery. [Online]. Available: https://www.grayanalytics.com/blog/the-importance-of-backup-and-recovery [Accessed 29 April 2020].

[26] Cloud versus On-Premise Computing. [Online]. Available: https://www.scirp.org/html/7-2121263_87661.htm

Appendix 1 – Ansible VPC Deployment and Management Role

vpc.yml:

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Include VPC role
      include_role:
        name: "{{ role_name }}"
      vars:
        create_initial_vpc: True
      loop:
        - vpc
        - bastion
        - le
        - zabbix
        - nginx
        - mysql
      loop_control:
        loop_var: role_name
    - name: Set create_initial_vpc to fact
      set_fact:
        create_initial_vpc: True
```

main.yml:

```
---
- name: Deploy initial VPC
  include_tasks: deploy_vpc.yml
  when: create_initial_vpc is defined and create_initial_vpc ==
True
```

```
- name: Gather VPC information
  include_tasks: get_vpc_information.yml
  when: fetch_vpc_info is defined and fetch_vpc_info == True
```

deploy_vpc.yml:

```
---
# Create VPC
- include: create_vpc.yml
# Create private subnet
- include: create_private_subnet.yml
# Create public subnet
- include: create_public_subnet.yml
# Create internet gateway
- include: create_igw.yml
# Create public route table
- include: create_public_rt.yml
# Create security group for bastion
- include: create_bastion_sg.yml
# Create security group for monitoring
- include: create_monitoring_sg.yml
# Create security group for webservers
- include: create_webserver_sg.yml
# Create security group for database servers
- include: create_db_sg.yml
# Update monitoring security group
- include: update_monitoring_sg.yml
# Update database security group
- include: update_db_sg.yml
# Create NAT gateway for private subnet
- include: create_nat_gw.yml
# Create route table for private subnet
- include: create_private_rt.yml
```

create_vpc.yml:

```
- name: Create VPC
  ec2_vpc_net:
    name: "{{ vpc_name }}"
    cidr_block: "{{ vpc_cidr_block }}"
    region: "{{ AWS_REGION }}"
    state: present
    tags:
      Name: "{{ vpc_name }}"
      Environment: "{{ project_env }}"
  register: deployed_vpc

- name: Assign VPC ID to a variable
  set_fact:
    vpc_id: "{{ deployed_vpc.vpc.id }}"
```

create_private_subnet.yml

```
---
- name: Create private subnet for database servers
  ec2_vpc_subnet:
    state: present
    vpc_id: "{{ vpc_id }}"
    cidr: "{{ private_subnet_cidr_block }}"
    tags:
      Name: "{{ vpc_name }}_private_sn"
  register: private_subnet

- name: Assign private subnet ID to variable
  set_fact:
    private_subnet_id: "{{ private_subnet.subnet.id }}"
```

create_public_subnet.yml

```
---
- name: Create public subnet for webservers servers
  ec2_vpc_subnet:
    state: present
    vpc_id: "{{ vpc_id }}"
    cidr: "{{ public_subnet_cidr_block }}"
    map_public: yes
    tags:
      Name: "{{ vpc_name }}_public_sn"
  register: public_subnet

- name: Assign public subnet ID to variable
  set_fact:
    public_subnet_id: "{{ public_subnet.subnet.id }}"
```

create_igw.yml

```
---
- name: Create internet gateway
  ec2_vpc_igw:
    vpc_id: "{{ vpc_id }}"
    state: present
    tags:
      VPC: "{{ vpc_name }}"
  register: igw

- name: Set internet gateway ID to a variable
  set_fact:
    igw_id: "{{ igw.gateway_id }}"
```

create_public_rt.yml

```
---
- name: Create route table for public subnet
  ec2_vpc_route_table:
    vpc_id: "{{ vpc_id }}"
    subnets: "{{ public_subnet_id }}"
    routes:
      - dest: "0.0.0.0/0"
        gateway_id: "{{ igw_id }}"
    tags:
      Name: "{{ vpc_id }}_public_rt"
```

create_bastion_sg.yml

```
---
- name: Create security group for bastion hosts
  ec2_group:
    name: "{{ vpc_name }}_bastion_sg"
    description: "SG for bastion hosts"
    vpc_id: "{{ vpc_id }}"
    rules:
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: 0.0.0.0/0
    tags:
      Name: "{{ vpc_name }}_bastion_sg"
      Environment: "{{ project_env }}"
  register: bastion_sg

- name: Assign bastion security group properties to facts
  set_fact:
    bastion_sg_id: "{{ bastion_sg.group_id }}"
    bastion_sg_owner: "{{ bastion_sg.owner_id }}"
    bastion_sg_name: "{{ bastion_sg.group_name }}"
```

create_monitoring_sg.yml

```
---
- name: Create security group for monitoring servers
  ec2_group:
    name: "{{ vpc_name }}_monitoring_sg"
    description: "SG for monitoring"
    vpc_id: "{{ vpc_id }}"
    tags:
      Name: "{{ vpc_name }}_monitoring_sg"
      Environment: "{{ project_env }}"
  register: monitoring_sg

- name: Assign monitoring security group properties to facts
  set_fact:
    monitoring_sg_id: "{{ monitoring_sg.group_id }}"
    monitoring_sg_owner: "{{ monitoring_sg.owner_id }}"
    monitoring_sg_name: "{{ monitoring_sg.group_name }}"
```

create_webserver_sg.yml

```
---
- name: Create security group for webservers
  ec2_group:
    name: "{{ vpc_name }}_webserver_sg"
    description: "SG for webservers"
    vpc_id: "{{ vpc_id }}"
    rules:
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      - proto: tcp
```

```
          from_port: 443
          to_port: 443
          cidr_ip: 0.0.0.0/0
        - proto: tcp
          from_port: 10050
          to_port: 10050
          group_id: "{{ monitoring_sg_owner }}/{{ monitoring_sg_id
}}/{{ monitoring_sg_name }}"
        - proto: tcp
          from_port: 22
          to_port: 22
          group_id: "{{ bastion_sg_owner }}/{{ bastion_sg_id }}/{{
bastion_sg_name }}"
    tags:
      Name: "{{ vpc_name }}_webserver_sg"
      Environment: "{{ project_env }}"
  register: webserver_sg

- name: Assign webserver security group properties to facts
  set_fact:
    webserver_sg_id: "{{ webserver_sg.group_id }}"
    webserver_sg_owner: "{{ webserver_sg.owner_id }}"
    webserver_sg_name: "{{ webserver_sg.group_name }}"
```

create_db_sg.yml

```
---
- name: Create security group for db servers
  ec2_group:
    name: "{{ vpc_name }}_database_sg"
    description: "SG for db servers"
    vpc_id: "{{ vpc_id }}"
    tags:
      Name: "{{ vpc_name }}_database_sg"
      Environment: "{{ project_env }}"
  register: database_sg

- name: Assign db servers security group ID to a variable
  set_fact:
    database_sg_id: "{{ database_sg.group_id }}"
    database_sg_owner: "{{ database_sg.owner_id }}"
    database_sg_name: "{{ database_sg.group_name }}"
```

update_monitoring_sg.yml

```yaml
---
- name: Update monitoring security group
  ec2_group:
    name: "{{ vpc_name }}_monitoring_sg"
    description: "SG for monitoring"
    group_id: "{{ monitoring_sg_id }}"
    vpc_id: "{{ vpc_id }}"
    state: present
    rules:
      - proto: tcp
        from_port: 10051
        to_port: 10051
        group_id: "{{ webserver_sg_owner }}/{{ webserver_sg_id
}}/{{ webserver_sg_name }}"
      - proto: tcp
        from_port: 10051
        to_port: 10051
        group_id: "{{ bastion_sg_owner }}/{{ bastion_sg_id }}/{{
bastion_sg_name }}"
      - proto: tcp
        from_port: 10051
        to_port: 10051
        group_id: "{{ database_sg_owner }}/{{ database_sg_id }}/{{
database_sg_name }}"
      - proto: tcp
        from_port: 22
        to_port: 22
        group_id: "{{ bastion_sg_owner }}/{{ bastion_sg_id }}/{{
bastion_sg_name }}"
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 443
        to_port: 443
        cidr_ip: 0.0.0.0/0
```

update_dg_sg.yml

```yaml
- name: Create security group for db servers
  ec2_group:
    name: "{{ vpc_name }}_database_sg"
    description: "SG for db servers"
    vpc_id: "{{ vpc_id }}"
    rules:
      - proto: tcp
        from_port: 3306
        to_port: 3306
        group_id: "{{ webserver_sg_owner }}/{{ webserver_sg_id
}}/{{ webserver_sg_name }}"
      - proto: tcp
        from_port: 10050
        to_port: 10050
        group_id: "{{ monitoring_sg_owner }}/{{ monitoring_sg_id
}}/{{ monitoring_sg_name }}"
      - proto: tcp
        from_port: 22
        to_port: 22
        group_id: "{{ bastion_sg_owner }}/{{ bastion_sg_id }}/{{
bastion_sg_name }}"
      - proto: tcp
        from_port: 3306
        to_port: 3306
        group_id: "{{ database_sg_owner }}/{{ database_sg_id }}/{{
database_sg_name }}"
```

create_nat_gw.yml

```
---
- name: Create NAT gateway for servers in private subnet
  ec2_vpc_nat_gateway:
    state: present
    subnet_id: "{{ private_subnet_id }}"
    wait: true
    if_exist_do_not_create: true
  register: nat_gw

- name: Assign NAT GW id to a variable
  set_fact:
    nat_gateway_id: "{{ nat_gw.nat_gateway_id }}"
```

create_private_rt.yml

```
---
- name: Create private subnet for database servers
  ec2_vpc_subnet:
    state: present
    vpc_id: "{{ vpc_id }}"
    cidr: "{{ private_subnet_cidr_block }}"
    tags:
      Name: "{{ vpc_name }}_private_sn"
  register: private_subnet

- name: Assign private subnet ID to variable
  set_fact:
    private_subnet_id: "{{ private_subnet.subnet.id }}"
```

get_vpc_information.yml:

```
- name: Get VPC information for deployment
  ec2_vpc_net_info:
    filters:
      "tag:Name": "{{ vpc_name }}"
  register: vpc_info

- name: Set VPC id to a variable
  set_fact:
    vpc_id: "{{ vpc_info.vpcs | map(attribute='vpc_id') | join }}"

- name: Get public subnet information for deployment
  ec2_vpc_subnet_info:
    filters:
      vpc-id: "{{ vpc_id }}"
      "tag:Name": "{{ vpc_name }}_public_sn"
  register: public_subnet_info

- name: Set public subnet id to a variable
  set_fact:
    public_subnet_id: "{{ public_subnet_info.subnets |
map(attribute='subnet_id') | join }}"

- name: Get private subnet information for deployment
  ec2_vpc_subnet_info:
    filters:
      vpc-id: "{{ vpc_id }}"
      "tag:Name": "{{ vpc_name }}_private_sn"
  register: private_subnet_info

- name: Set private subnet id to a variable
  set_fact:
    private_subnet_id: "{{ private_subnet_info.subnets |
map(attribute='subnet_id') | join }}"

- name: Get monitoring security group info
  ec2_group_info:
    filters:
      "tag:Name": "{{ vpc_name }}_monitoring_sg"
  register: monitoring_sg_info

- name: Set monitoring security group info to variables
  set_fact:
    monitoring_sg_id: "{{ monitoring_sg_info.security_groups |
map(attribute='group_id') | join }}"
    monitoring_sg_owner: "{{ monitoring_sg_info.security_groups |
map(attribute='owner_id') | join }}"
    monitoring_sg_name: "{{ monitoring_sg_info.security_groups |
map(attribute='group_name') | join }}"

- name: Get webserver security group info
  ec2_group_info:
    filters:
      "tag:Name": "{{ vpc_name }}_webserver_sg"
  register: webserver_sg_info

- name: Set webserver security group info to variables
  set_fact:
    webserver_sg_id: "{{ webserver_sg_info.security_groups |
map(attribute='group_id') | join }}"
    webserver_sg_owner: "{{ webserver_sg_info.security_groups |
map(attribute='owner_id') | join }}"
    webserver_sg_name: "{{ webserver_sg_info.security_groups |
map(attribute='group_name') | join }}"

- name: Get database security group info
  ec2_group_info:
    filters:
      "tag:Name": "{{ vpc_name }}_database_sg"
  register: database_sg_info

- name: Set database security group info to variables
```

```
  set_fact:
    database_sg_id: "{{ database_sg_info.security_groups |
map(attribute='group_id') | join }}"

- name: Get bastion security group info
  ec2_group_info:
    filters:
      "tag:Name": "{{ vpc_name }}_bastion_sg"
  register: bastion_sg_info

- name: Set bastion security group info to variables
  set_fact:
    bastion_sg_id: "{{ bastion_sg_info.security_groups |
map(attribute='group_id') | join }}"
    bastion_sg_owner: "{{ bastion_sg_info.security_groups |
map(attribute='owner_id') | join }}"
    bastion_sg_name: "{{ bastion_sg_info.security_groups |
map(attribute='group_name') | join }}"

- name: Get bastion instance info
  ec2_instance_info:
    filters:
      "tag:Name": "{{ bastion_host_name }}"
      instance-state-name: running
  register: bastion_instance_info

- name: Set bastion host IP to a variable
  set_fact:
    bastion_ip: "{{
bastion_instance_info.instances[0].public_ip_address }}"

- name: Get monitoring instance info
  ec2_instance_info:
    filters:
      "tag:Name": "{{ monitoring_host_name }}"
      instance-state-name: running
  register: monitoring_instance_info

- name: Set monitoring host IP's to a variable
  set_fact:
    monitoring_public_ip: "{{
monitoring_instance_info.instances[0].public_ip_address }}"
    monitoring_private_ip: "{{
monitoring_instance_info.instances[0].private_ip_address }}"
```

```
- name: Get first web server instance info
  ec2_instance_info:
    filters:
      "tag:Name": "{{ webserver_host_name[0] }}"
      instance-state-name: running
  register: webserver1_instance_info

- name: Get second web server instance info
  ec2_instance_info:
    filters:
      "tag:Name": "{{ webserver_host_name[1] }}"
      instance-state-name: running
  register: webserver2_instance_info

- name: Set web server private ip's to fact
  set_fact:
    web1_private_ip: "{{
webserver1_instance_info.instances[0].private_ip_address }}"
    web2_private_ip: "{{
webserver2_instance_info.instances[0].private_ip_address }}"
```

Appendix 2 – Zabbix Deployment and Management Role

monitoring.yml:

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Get VPC information
      include_role:
        name: vpc
      vars:
        fetch_vpc_info: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined

- hosts: monitoring
  become: true
  vars:
    ansible_ssh_common_args: "-o ProxyCommand='ssh -o
StrictHostKeyChecking=no -i {{ private_key_location }} -W %h:%p -q
ubuntu@{{ hostvars['localhost']['bastion_ip'] }}'"
  vars_files:
    - keys
  tasks:
    - name: Patch monitoring host
      include_role:
        name: patching
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Install common packages
      include_role:
        name: common
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Configure monitoring
      include_role:
        name: zabbix
      vars:
        configure_monitoring_host: True
      when: hostvars['localhost']['create_initial_vpc'] is defined

- hosts: monitoring
  become: true
  vars_files:
    - keys
  tasks:
    - name: Configure monitoring host
      include_role:
        name: zabbix
      vars:
        configure_monitoring_host: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined
```

tasks/main.yml:

```
---
- name: Deploy monitoring instance during VPC deploy
  include_tasks: create_instance.yml
  when: create_initial_vpc is defined and create_initial_vpc ==
True

- name: Configure monitoring host
  include_tasks: configure_monitoring.yml
  when: configure_monitoring_host is defined and
configure_monitoring_host == True

- name: Configure agent
  include_tasks: configure_agent.yml
  when: zabbix_agent is defined and zabbix_agent == True
```

tasks/create_instance.yml

```
---
- name: Create monitoring host
  include_role:
    name: provisioning
  vars:
    instance_name: "{{ item }}"
    instance_role: "monitoring"
    subnet_id: "{{ public_subnet_id }}"
    security_group: "{{ monitoring_sg_id }}"
  with_items:
    - "{{ monitoring_host_name }}"

- name: Set monitoring instance id to a fact
  vars:
    fact_string: "{{ item }}_instance_info"
  set_fact:
    "{{ item }}_host_id": "{{
hostvars['localhost'][fact_string].instance_ids | join }}"
    "{{ item }}_host_ip": "{{
hostvars['localhost'][fact_string].instances[0].private_ip_address
}}"
  with_items:
    - "{{ monitoring_host_name }}"

- name: Assign elastic IP to monitoring host
  vars:
    fact_string: "{{ item }}_host_id"
  ec2_eip:
    device_id: "{{ hostvars['localhost'][fact_string] }}"
    in_vpc: true
    state: present
  register: elastic_ip
  with_items:
    - "{{ monitoring_host_name }}"

- name: Set monitoring host IP to a variable
  vars:
    ip_string: "{{ item }}_host_ip"
  set_fact:
    monitoring_private_ip: "{{ hostvars['localhost'][ip_string] }}"
    "{{ item }}_ip": "{{ elastic_ip.results |
map(attribute='public_ip') | join }}"
  with_items:
    - "{{ monitoring_host_name }}"

- name: Add private ip to in-memory group
  add_host:
    hostname: "{{ item }}"
    ansible_host: "{{ monitoring_private_ip }}"
    groups:
      - monitoring
  with_items:
    - "{{ monitoring_host_name }}"
```

tasks/configure_zabbix.yml

```
---
- name: Add GPG key and repository
  include_tasks: configure_repo.yml

- name: Install Zabbix server packages
  apt:
    name: ['zabbix-server-mysql', 'zabbix-frontend-php', 'zabbix-
nginx-conf']
    state: present
    update_cache: yes
    install_recommends: yes

- name: Install Ansible module dependencies
  apt:
    name: ['python3-psycopg2', 'python3-mysqldb', 'mariadb-client']
    state: present
    update_cache: yes

- name: Remove anonymous users
  mysql_user:
    name: ''
    host: "{{ item }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
    state: absent
  with_items:
    - localhost
    - 127.0.0.1
    - ::1

- name: Remove test database if present
  mysql_db:
    name: test
    state: absent
    login_unix_socket: /var/run/mysqld/mysqld.sock

- name: Create MySQL database
  mysql_db:
    name: "{{ zabbix_database_name }}"
    encoding: utf8
    collation: utf8_bin
    state: present
    login_unix_socket: /var/run/mysqld/mysqld.sock
  register: zabbix_db_creation

- name: Create MySQL user
  mysql_user:
    name: "{{ zabbix_database_user }}"
    host: localhost
    password: "{{ zabbix_database_password }}"
    priv: "{{ zabbix_database_name }}.*:ALL,GRANT"
    state: present

- name: Import database
  mysql_db:
    name: "{{ zabbix_database_name }}"
    encoding: utf8
    collation: utf8_bin
    state: import
    target: '/usr/share/doc/zabbix-server-mysql/create.sql.gz'
  when: zabbix_db_creation.changed

- name: Update Admin user password
  vars:
    query: "UPDATE zabbix.users SET passwd=md5('{{
zabbix_admin_password }}') where alias='Admin'"
  command: mysql -u root zabbix --execute "{{ query }}";
  when: zabbix_db_creation.changed

- name: Configure zabbix-server
  template:
    src: zabbix_server.conf.j2
    dest: /etc/zabbix/zabbix_server.conf
    owner: zabbix
    group: zabbix
    mode: 0640

- name: Configure Zabbix PHP file
  template:
    src: zabbix.conf.php.j2
    dest: /usr/share/zabbix/conf/zabbix.conf.php
    owner: www-data
    group: www-data
    mode: 0640
```

```yaml
  notify:
    - Restart server

- name: Upload certificates
  include_role:
    name: le
  vars:
    upload_certificate: True
  when: issue_le_certificate == True

- name: Configure nginx
  template:
    src: nginx.conf.j2
    dest: /etc/zabbix/nginx.conf
    owner: root
    group: root
    mode: 0640
  notify:
    - Restart nginx

- name: Remove default nginx file
  file:
    path: /etc/nginx/sites-enabled/default
    state: absent

- name: Configure PHP-FPM
  template:
    src: php-fpm.conf.j2
    dest: /etc/zabbix/php-fpm.conf
    owner: root
    group: root
    mode: 0640
  notify:
    - Restart php-fpm

- name: Enable services
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  with_items:
    - zabbix-server
    - nginx
    - php7.4-fpm
- name: Set fact
  set_fact:
    dns_ip_value: "{{ monitoring_host_name }}_ip"

- name: Manage DNS entry
  include_tasks: manage_dns_entry.yml
  vars:
    dns_name: "{{ zabbix_webui_url }}"
    dns_ip: "{% if create_initial_vpc is defined and domain_name !=
''  %}{{hostvars['localhost'][dns_ip_value]}}{% else %}{{
hostvars['localhost']['monitoring_public_ip'] }}{% endif %}"
  when: load_balancer_setup is not defined or load_balancer_setup
!= True and manage_dns is defined and manage_dns == True
```

tasks/configure_repo.yml

```
---
- name: Set Zabbix repository URL to a fact
  set_fact:
    zabbix_repo: "http://repo.zabbix.com/zabbix/{{ zabbix_version
}}/{{ ansible_distribution.lower() }} {{
ansible_distribution_release }} main"

- name: Install Zabbix GPG key
  apt_key:
    id: "{{ zabbix_gpg_key_id }}"
    url: http://repo.zabbix.com/zabbix-official-repo.key

- name: Install Zabbix repository
  apt_repository:
    repo: "{{ item }} {{ zabbix_repo }}"
    state: present
  with_items:
    - deb-src
    - deb
```

tasks/configure_monitoring.yml

```
---
# Configure SSH options
- name: Configure SSH
  include_role:
    name: ssh
```

```yaml
# Configure users
- name: Configure users
  include_role:
    name: users

- name: Install and configure Zabbix
  include_tasks: configure_zabbix.yml
```

tasks/configure_agent.yml

```yaml
---
- name: Add GPG key and repository
  include_tasks: configure_repo.yml

- name: Install Zabbix agent
  apt:
    name: zabbix-agent
    update_cache: yes
    state: latest

- name: Install Zabbix API PIP module
  pip:
    name: zabbix-api

- name: Configure Zabbix agent
  template:
    src: zabbix_agentd.conf.j2
    dest: /etc/zabbix/zabbix_agentd.conf
    owner: root
    group: root
    mode: '0644'
  notify:
    - Restart agent

- name: Upload PSK file
  template:
    src: zabbix_agentd.psk.j2
    dest: /etc/zabbix/zabbix_agentd.psk
    owner: root
    group: zabbix
    mode: '0640'
  notify:
    - Restart agent
```

tasks/manage_dns_entry.yml

```
---
- name: Create DNS entry for "{{ dns_name }}"
  route53:
    zone: "{{ domain_name }}"
    record: "{{ dns_name }}"
    type: A
    value: "{{ dns_ip }}"
    ttl: 300
    state: "{{ dns_state | default(‘present’) }}"
    wait: yes
    overwrite: yes
  delegate_to: localhost
  become: no
```

handlers/main.yml

```
- name: Restart agent
  service: name=zabbix-agent state=restarted

- name: Restart server
  service: name=zabbix-server state=restarted

- name: Restart nginx
  service: name=nginx state=restarted

- name: Restart php-fpm
  service: name=php7.4-fpm state=restarted
```

Appendix 3 –User Creation and Management Role

tasks/main.yml

```
---
# Create groups
- include: groups.yml
# Create users on systems
- include: users.yml
```

tasks/groups.yml

```
---
- name: Create groups
  group:
    name: "{{ item.name }}"
    state: "{{ item.state | default('present') }}"
  with_items:
    - "{{ user_groups }}"
```

tasks/users.yml

```
---
# https://docs.ansible.com/ansible/latest/modules/user_module.html
- name: Create user
  user:
    name: "{{ item.name }}"
    state: "{{ item.state | default('present') }}"
    group: "{% if item.admin_user is defined and item.admin_user ==
True %}sudo{% else %}developers{% endif %}"
    home: "{{ item.home | default('/home/'+item.name) }}"
    shell: "/bin/bash"
    password: "{{ item.password }}"
    comment: "{{ item.comment | default('') }}"
    append: yes
  with_items:
    - "{{ users }}"
  when: item.state != 'absent'

- name: Add SSH key to user
  authorized_key:
    user: "{{ item.name }}"
    key: "{{ item.ssh_key }}"
    state: "{{ item.state | default('present') }}"
  with_items:
    - "{{ users }}"
  when: item.state != 'absent'

- name: Delete user
  user:
    name: "{{ item.name }}"
    state: absent
  with_items:
    - "{{ users }}"
  when: item.state == 'absent'
    - "{{ users }}"
  when: item.state != 'absent'

- name: Delete user
  user:
    name: "{{ item.name }}"
    state: absent
  with_items:
    - "{{ users }}"
  when: item.state == 'absent'
```

Appendix 4 –SSH Configuration and Management Role

tasks/main.yml

```yaml
---
- name: Replace default SSH config
  template:
    src: sshd_config.j2
    dest: "/etc/ssh/sshd_config"
    owner: "root"
    group: "root"
    mode: "0644"
  notify:
    - Test SSHD config
    - Restart SSHD
```

handlers/main.yml

```yaml
---
- name: Test SSHD config
  command: sshd -t

- name: Restart SSHD
  service: name="ssh" state="restarted"
```

Appendix 5 – Instance Provisioning Role

tasks/main.yml

```yaml
---
- name: Create {{ instance_name }} instance
  ec2_instance:
    name: "{{ instance_name }}"
    key_name: "{{ key_name }}"
    vpc_subnet_id: "{{ subnet_id }}"
    security_group: "{{ security_group }}"
    instance_type: "{{ instance_type }}"
    image_id: "{{ image_id }}"
    state: running
    wait: yes
    wait_timeout: 60
    tags:
      Role: "{{ instance_role }}"
      Environment: "{{ project_env }}"
  register: instance_info

- name: Set instance information to a fact
  set_fact:
    "{{ instance_name }}_instance_info": "{{ instance_info }}"
```

Appendix 6 – System Patching Role

tasks/main.yml

```
---
#
https://encoretechnologies.github.io/blog/2018/06/ansiblepatchingau
tomation/
- name: Wait for any possibly running unattended upgrade to finish
  raw: systemd-run --property="After=apt-daily.service apt-daily-
upgrade.service" --wait /bin/true

- name: Update packages...
  apt:
    upgrade: dist
    update_cache: yes

- name: Reboot if kernel was updated and reboot is requested by the
system
  shell: 'sleep 5 && /sbin/shutdown -r now'
  args:
    removes: /var/run/reboot-required
  async: 1
  poll: 0
  ignore_errors: true

- name: Waiting for system to come back from reboot...
  wait_for_connection:
    connect_timeout: 20
    sleep: 5
    delay: 5
    timeout: 180
```

Appendix 7 – Nginx Configuration and Management Role

webservers.yml

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Get VPC information
      include_role:
        name: vpc
      vars:
        fetch_vpc_info: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined

- hosts: webserver
  become: true
  vars_files:
    - keys
  vars:
    ansible_ssh_common_args: "-o ProxyCommand='ssh -o
StrictHostKeyChecking=no -i {{ private_key_location }} -W %h:%p -q
ubuntu@{{ hostvars['localhost']['bastion_ip'] }}'"
  tasks:
    - name: Patch webserver hosts
      include_role:
        name: patching
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Install common packages
      include_role:
        name: common
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Configure webservers
      include_role:
        name: nginx
      vars:
        configure_nginx_host: True
      when: hostvars['localhost']['create_initial_vpc'] is defined

- hosts: webserver
  become: true
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Configure webservers
      include_role:
        name: nginx
      vars:
        configure_nginx_host: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined
```

tasks/main.yml

```
---
- name: Deploy nginx instances during VPC deploy
  include_tasks: create_instance.yml
  when: create_initial_vpc is defined and create_initial_vpc ==
True

- name: Configure nginx host
  include_tasks: configure_nginx.yml
  when: configure_nginx_host is defined and configure_nginx_host ==
True
```

tasks/create_instance.yml

```
---
- name: Set subnet to a fact
  set_fact:
    subnet_group: "{% if load_balancer_setup is not defined or
load_balancer_setup != True %}public{% elif load_balancer_setup is
defined and load_balancer_setup == True %}private{% else %}public{%
endif %}"
- name: Create webserver hosts
  include_role:
    name: provisioning
  vars:
    instance_name: "{{ item }}"
    instance_role: "webserver"
    subnet_id: "{% if subnet_group == 'public' %}{{
public_subnet_id }}{% else %}{{ private_subnet_id }}{% endif %}"
    security_group: "{{ webserver_sg_id }}"
  with_items:
    - "{{ webserver_host_name }}"

- name: Set webserver instance ids to a fact
  vars:
    fact_string: "{{ item }}_instance_info"
  set_fact:
    "{{ item }}_host_id": "{{
hostvars['localhost'][fact_string].instance_ids | join }}"
    "{{ item }}_host_ip": "{{
hostvars['localhost'][fact_string].instances[0].private_ip_address
}}"
  with_items:
    - "{{ webserver_host_name }}"

- name: Assign elastic IP to webserver host
  vars:
    fact_string: "{{ item }}_host_id"
  ec2_eip:
    device_id: "{{ hostvars['localhost'][fact_string] }}"
    in_vpc: true
    state: present
  register: elastic_ip
  with_items:
    - "{{ webserver_host_name }}"
  when: subnet_group == 'public'

- name: Set webserver public ip into a fact when present
  vars:
    fact_string: "{{ item }}_instance_info"
  set_fact:
    "{{ item }}_public_ip": "{{
hostvars['localhost'][fact_string].instances[0].public_ip_address
}}"
  with_items:
    - "{{ webserver_host_name }}"
  when: subnet_group == 'public'

- name: Get first web server instance info
  ec2_instance_info:
    filters:
      "tag:Name": "{{ webserver_host_name[0] }}"
      instance-state-name: running
  register: webserver1_instance_info

- name: Get second web server instance info
  ec2_instance_info:
    filters:
      "tag:Name": "{{ webserver_host_name[1] }}"
      instance-state-name: running
  register: webserver2_instance_info

- name: Set web server private ip's to fact
  set_fact:
    web1_private_ip: "{{
webserver1_instance_info.instances[0].private_ip_address }}"
    web2_private_ip: "{{
webserver2_instance_info.instances[0].private_ip_address }}"

- name: Add private ip to in-memory group
  vars:
    ip_string: "{{ item }}_host_ip"
  add_host:
    hostname: "{{ item }}"
    ansible_host: "{{ hostvars['localhost'][ip_string] }}"
    groups:
      - webserver
  with_items:
    - "{{ webserver_host_name }}"
```

tasks/configure_nginx.yml

```
---
```

```yaml
#
https://docs.ansible.com/ansible/latest/modules/zabbix_host_module.
html
# Configure SSH
- name: Configure SSH
  include_role:
    name: ssh

# Configure users
- name: Configure users
  include_role:
    name: users

# Configure Zabbix agent
- name: Configure Zabbix agent
  include_role:
    name: zabbix
  vars:
    zabbix_agent: True

- name: Add host to Zabbix
  zabbix_host:
    server_url: "https://{{ zabbix_webui_url }}"
    login_user: Admin
    login_password: "{{ zabbix_admin_password }}"
    host_name: "{{ inventory_hostname }}"
    status: enabled
    state: present
    tls_psk_identity: "{{ zabbix_tls_identity_name }}"
    tls_connect: 2
    tls_psk: "{{ zabbix_tls_identity_password }}"
    tls_accept: 2
    host_groups:
      - Linux servers
    link_templates:
      - Template OS Linux by Zabbix agent
      - Template App Nginx by Zabbix agent
    interfaces:
      - type: 1
        main: 1
        useip: 1
        ip: "{{ ansible_default_ipv4.address }}"
    validate_certs: no
```

```
# - name: Install nginx packages
- name: Install nginx package
  apt:
    name: nginx
    state: latest
    update_cache: yes

- name: Install PHP packages
  apt:
    name: ['php-fpm', 'php-mysql']
    state: latest
    update_cache: yes

- name: Configure nginx
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/nginx.conf
    owner: root
    group: root
    mode: '0644'
  notify:
    - Restart nginx

- name: Configure default site for zabbix checks
  template:
    src: default.conf.j2
    dest: /etc/nginx/sites-enabled/default
    owner: root
    group: root
    mode: '0644'
  notify:
    - Reload nginx

- name: Manage Elastic Load Balancer
  include_tasks: manage_elb.yml
  when: load_balancer_setup is defined and load_balancer_setup ==
True

- name: Manage vhosts
  include_tasks: manage_vhosts.yml
  when: vhosts is defined and vhosts != ''
```

tasks/manage_dns_entry.yml

```
---
- name: Create DNS entry for "{{ dns_name }}"
```

```
  route53:
    zone: "{{ domain_name }}"
    record: "{{ dns_name }}"
    type: A
    value: "{{ dns_ip }}"
    ttl: 300
    state: "{{ dns_state }}"
    wait: yes
    overwrite: yes
  delegate_to: localhost
  become: no
```

tasks/manage_elb.yml

```
---
# Let's Encrypt certificates does not work with ELB
# - name: Import certificate into aws ACM
#   vars:
#     full_cert_file: "{{ certificate_directory }}/{{ domain_name
}}-fullchain.crt"
#     cert_file: "{{ certificate_directory }}/{{ domain_name
}}.crt"
#     key_file: "{{ certificate_directory }}/{{ domain_name }}.key"
#   shell: >
#     aws acm import-certificate
#     --certificate "file://{{ cert_file }}"
#     --private-key "file://{{ key_file }}"
#     --certificate-chain "file://{{ full_cert_file }}"
#     --region "{{ AWS_REGION }}"
#   register: certificate_id
#   delegate_to: localhost
#   become: no

- name: Set webserver instance-id's to variables
  vars:
    app1_string: "{{ webserver_host_name[0] }}_host_id"
    app2_string: "{{ webserver_host_name[1] }}_host_id"
  set_fact:
    app1_id: "{{ app1_string }}"
    app2_id: "{{ app2_string }}"
  delegate_to: localhost
  become: no
```

```
- name: Create ELB
  ec2_elb_lb:
    name: 'ELB'
    state: present
    region: "{{ AWS_REGION }}"
    instance_ids: ['app1_id', 'app2_id']
    security_group_ids: "{{
hostvars['localhost']['webserver_sg_id'] }}"
    subnets: "{{ hostvars['localhost']['public_subnet_id'] }}"
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
        proxy_protocol: True
      - protocol: https
        load_balancer_port: 443
        instance_protocol: http
        instance_port: 80
        ssl_certificate_id: "{{ ssl_certificate_location }}"
  delegate_to: localhost
  become: no
```

tasks/manage_vhosts.yml

```
---
- name: Create vhost directory
  file:
    path: "/srv/vhosts/{{ item.name }}"
    state: "{{ item.state | default('directory') }}"
    owner: root
    group: developers
    mode: '0775'
  with_items: "{{ vhosts }}"

- name: Create html directory
  file:
    path: "/srv/vhosts/{{ item.name }}/html"
    state: "{{ item.state | default('directory') }}"
    owner: root
    group: developers
    mode: '0775'
  with_items: "{{ vhosts }}"
```

```yaml
- name: Create log directory
  file:
    path: "/srv/vhosts/{{ item.name }}/log"
    state: "{{ item.state | default('directory') }}"
    owner: root
    group: developers
    mode: '0640'
  with_items: "{{ vhosts }}"

- name: Create tmp directory
  file:
    path: "/srv/vhosts/{{ item.name }}/tmp"
    state: "{{ item.state | default('directory') }}"
    owner: root
    group: root
    mode: '0777'
  with_items: "{{ vhosts }}"

- name: Upload certificates
  include_role:
    name: le
  vars:
    upload_certificate: True
  when: issue_le_certificate == True

- name: Create nginx configuration
  template:
    src: vhost.conf.j2
    dest: "/etc/nginx/sites-available/{{ item.name }}.conf"
    owner: root
    group: root
    mode: '0644'
  with_items: "{{ vhosts }}"

- name: Enable nginx configuration
  file:
    src: "/etc/nginx/sites-available/{{ item.name }}.conf"
    dest: "/etc/nginx/sites-enabled/{{ item.name }}.conf"
    state: "{{ item.state | default('link') }}"
  with_items: "{{ vhosts }}"
  notify:
    - Test nginx config

- name: Set instance IP public ip to a variable
```

```
  vars:
    ip_string: "{{ item }}_public_ip"
  set_fact:
    dns_ip_value: "{{ hostvars['localhost'][ip_string] }}"
  when: load_balancer_setup is not defined or load_balancer_setup
!= True
  with_items:
    - "{{ webserver_host_name }}"

- name: Manage DNS entry
  include_tasks: manage_dns_entry.yml
  vars:
    dns_name: "{{ item.name }}"
    dns_ip: "{{ dns_ip_value }}"
    dns_state: "{{ item.state | default('present') }}"
  with_items: "{{ vhosts }}"
  when: load_balancer_setup is not defined or load_balancer_setup
!= True and manage_dns is defined and manage_dns == True
```

handlers/main.yml

```
---
- name: Test nginx config
  shell: nginx -t
  notify:
    - Reload nginx

- name: Reload nginx
  service: name=nginx state=reloaded

- name: Restart nginx
  service: name=nginx state=restarted
```

Appendix 8 – MySQL Configuration and Management Role

dbservers.yml

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Get VPC information
      include_role:
        name: vpc
      vars:
        fetch_vpc_info: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined

- hosts: database
  become: true
  vars_files:
    - keys
  vars:
    ansible_ssh_common_args: "-o ProxyCommand='ssh -o
StrictHostKeyChecking=no -i {{ private_key_location }} -W %h:%p -q
ubuntu@{{ hostvars['localhost']['bastion_ip'] }}'"
  tasks:
    - name: Patch database hosts
      include_role:
        name: patching
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Install common packages
      include_role:
        name: common
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Configure database servers
      include_role:
        name: mysql
      vars:
        configure_database_host: True
      when: hostvars['localhost']['create_initial_vpc'] is defined

- hosts: database
  become: true
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Make sure common packages are installed
      include_role:
        name: common
    - name: Configure database
      include_role:
        name: mysql
      vars:
        configure_database_host: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined
```

tasks/main.yml

```
---
- name: Deploy database instances during VPC deploy
  include_tasks: create_instance.yml
  when: create_initial_vpc is defined and create_initial_vpc ==
True

- name: Configure database host
  include_tasks: configure_database.yml
  when: configure_database_host is defined and
configure_database_host == True
```

tasks/create_instance.yml

```
---
- name: Create database hosts
  include_role:
    name: provisioning
  vars:
    instance_name: "{{ item }}"
    instance_role: "database"
    subnet_id: "{{ private_subnet_id }}"
    security_group: "{{ database_sg_id }}"
  with_items:
    - "{{ database_host_name }}"

- name: Set database instance ids to a fact
  vars:
    fact_string: "{{ item }}_instance_info"
  set_fact:
    "{{ item }}_host_id": "{{
hostvars['localhost'][fact_string].instance_ids | join }}"
    "{{ item }}_host_ip": "{{
hostvars['localhost'][fact_string].instances[0].private_ip_address
}}"
  with_items:
    - "{{ database_host_name }}"

- name: Add private ip to in-memory group
  vars:
    ip_string: "{{ item }}_host_ip"
  add_host:
    hostname: "{{ item }}"
    ansible_host: "{{ hostvars['localhost'][ip_string] }}"
    groups:
      - database
  with_items:
    - "{{ database_host_name }}"
```

tasks/configure_database.yml

```
---
#
https://docs.ansible.com/ansible/latest/modules/zabbix_host_module.
html
# Configure SSH
- name: Configure SSH
  include_role:
    name: ssh

# Configure users
- name: Configure users
```

```
  include_role:
    name: users

# Configure Zabbix agent
- name: Configure Zabbix agent
  include_role:
    name: zabbix
  vars:
    zabbix_agent: True

- name: Add host to Zabbix
  zabbix_host:
    server_url: "https://{{ zabbix_webui_url }}"
    login_user: Admin
    login_password: "{{ zabbix_admin_password }}"
    host_name: "{{ inventory_hostname }}"
    status: enabled
    state: present
    tls_psk_identity: "{{ zabbix_tls_identity_name }}"
    tls_connect: 2
    tls_psk: "{{ zabbix_tls_identity_password }}"
    tls_accept: 2
    host_groups:
      - Linux servers
    link_templates:
      - Template OS Linux by Zabbix agent
      - Template DB MySQL by Zabbix agent
    interfaces:
      - type: 1
        main: 1
        useip: 1
        ip: "{{ ansible_default_ipv4.address }}"
    validate_certs: no

- name: Install MySQL package
  apt:
    name: mysql-server
    state: latest
    update_cache: yes

- name: Install pymysql
  pip:
    name: pymysql
    state: present
```

```yaml
- name: Configure MySQL server
  template:
    src: my.cnf.j2
    dest: /etc/mysql/mysql.conf.d/mysqld.cnf
    owner: root
    group: root
    mode: '0644'
  notify:
    - Restart mysql

- name: Start and enable MySQL database service
  service:
    name: mysql
    state: started
    enabled: yes

- name: Remove anonymous users
  mysql_user:
    name: ''
    host: "{{ item }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
    state: absent
  with_items:
    - localhost
    - 127.0.0.1
    - ::1

- name: Remove test database if present
  mysql_db:
    name: test
    state: absent
    login_unix_socket: /var/run/mysqld/mysqld.sock

- name: Manage databases
  include_tasks: manage_databases.yml
  when: databases is defined and databases != ''
```

tasks/manage_database.yml

```yaml
---
# https://topic.alibabacloud.com/a/managing-mysql-replication-with-
ansible_1_41_30026734.html
- name: Create database
```

```yaml
  mysql_db:
    name: "{{ item.name }}"
    state: "{{ item.state | default('present') }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
  with_items:
    - "{{ databases }}"

- name: Create database user from first web server
  mysql_user:
    name: "{{ item.user | default(item.name) }}"
    password: "{{ item.password }}"
    host: "{{ hostvars['localhost']['web1_private_ip'] }}"
    priv: "{{ item.name }}.*:ALL,GRANT"
    state: "{{ item.state | default('present') }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
  with_items:
    - "{{ databases }}"

- name: Create database user from second web server
  mysql_user:
    name: "{{ item.user | default(item.name) }}"
    password: "{{ item.password }}"
    host: "{{ hostvars['localhost']['web2_private_ip'] }}"
    priv: "{{ item.name }}.*:ALL,GRANT"
    state: "{{ item.state | default('present') }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
  with_items:
    - "{{ databases }}"

- name: Create replication user
  mysql_user:
    name: "{{ item.user_name }}"
    host: "%"
    password: "{{ item.password }}"
    priv: "*.*:REPLICATION SLAVE"
    state: "{{ item.state | default('present') }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
  with_items:
    - "{{ replication_setup }}"
  when: mysql_replication_role == 'master'
  register: create_replication_user

- name: Update authentication type for replication user
  vars:
    query: "ALTER USER {{ item.user_name }} IDENTIFIED WITH
mysql_native_password BY '{{ item.password }}'"
  command: mysql -u root --execute "{{ query }}";
  when: create_replication_user.changed
  with_items:
    - "{{ replication_setup }}"

- name: Configure slave node
  mysql_replication:
    mode: getslave
    login_unix_socket: /var/run/mysqld/mysqld.sock
  ignore_errors: true
  register: slave
  when: mysql_replication_role == 'slave'

- name: Set master's IP and hostname to a variable
  vars:
    master_hostname: "{{ database_host_name[0] }}"
  set_fact:
    master_ip: "{{
hostvars[master_hostname]['ansible_default_ipv4']['address'] }}"
    master_hostname: "{{ master_hostname }}"

- name: Make sure hosts entries are in place
  vars:
    mysql_replication_master_ip:
  lineinfile:
    dest: /etc/hosts
    line: "{{ master_ip }} {{ master_hostname }}"
  when: mysql_replication_role == 'slave'

- name: Get masters replication status
  mysql_replication:
    mode: getmaster
    login_unix_socket: /var/run/mysqld/mysqld.sock
  delegate_to: "{{ master_hostname }}"
  when: mysql_replication_role == 'slave'
  register: replication_status

- name: Change the master in slave
  mysql_replication:
    mode: changemaster
    master_host: "{{ master_ip }}"
    master_user: "{{ replication_setup[0].user_name }}"
    master_password: "{{ replication_setup[0].password }}"
    master_log_file: "{{ replication_status.File }}"
    master_log_pos: "{{ replication_status.Position }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
  when: mysql_replication_role == 'slave'

- name: Start slave in slave and start replication
  mysql_replication:
    mode: startslave
    login_unix_socket: /var/run/mysqld/mysqld.sock
  when: mysql_replication_role == 'slave'
```

handlers/main.yml

```
---
- name: Restart mysql
  service: name=mysql state=restarted
```

Appendix 9 – Automated Let’s Encrypt Certificate Deployment Role

tasks/main.yml

```
---
- name: Issue wildcard during VPC deploy
  include_tasks: issue_certificate.yml
  when: create_initial_vpc is defined and create_initial_vpc ==
True and issue_le_certificate == True and manage_dns is defined and
manage_dns == True

- name: Upload certificate to other instances
  include_tasks: upload_certificate.yml
  when: issue_le_certificate is defined and issue_le_certificate ==
True and upload_certificate is defined and upload_certificate ==
True and manage_dns is defined and manage_dns == True
```

tasks/issue_certificate.yml

```
---
#
https://docs.ansible.com/ansible/latest/modules/acme_account_module
.html
#
https://docs.ansible.com/ansible/latest/modules/acme_certificate_mo
dule.html#acme-certificate-module
#
https://docs.ansible.com/ansible/latest/modules/route53_module.html

- name: Set domain name to a fact
  set_fact:
    le_cert_domain: "{{ domain_name }}"
    wildcard: "*.{{ domain_name }}"

- name: Create private key directory
  file:
    path: '{{ certificate_directory }}'
    state: directory
    owner: ansible
    group: ansible
```

```yaml
- name: Create private key for ACME account
  openssl_privatekey:
    path: "{{ certificate_directory }}/account.key"
    size: 4096

- name: Create private key for certificate account
  openssl_privatekey:
    path: "{{ certificate_directory }}/{{ le_cert_domain }}.key"
    size: 4096

- name: Generate CSR for {{ le_cert_domain }}
  openssl_csr:
    path: "{{ certificate_directory }}/{{ le_cert_domain }}.csr"
    privatekey_path: "{{ certificate_directory }}/{{ le_cert_domain
}}.key"
    common_name: "{{ wildcard }}"

- name: Make sure ACME account exists
  acme_account:
    account_key_src: "{{ certificate_directory }}//account.key"
    acme_version: 2
    acme_directory: 'https://acme-
v02.api.letsencrypt.org/directory'
    state: present
    terms_agreed: yes
    contact:
      - "mailto:{{ acme_account_email }}"

- name: Create challenge for {{ le_cert_domain }}
  acme_certificate:
    modify_account: no
    acme_directory: 'https://acme-
v02.api.letsencrypt.org/directory'
    acme_version: 2
    account_key_src: "{{ certificate_directory }}/account.key"
    src: "{{ certificate_directory }}/{{ le_cert_domain }}.csr"
    cert: "{{ certificate_directory }}/{{ le_cert_domain }}.crt"
    challenge: dns-01
    remaining_days: 60
  register: dns_challenge

- name: Create Route53 DNS entry
  route53:
    zone: "{{ le_cert_domain }}"
    record: '{{ dns_challenge.challenge_data[wildcard]["dns-
01"]["record"] }}'
    type: TXT
    ttl: 60
    state: present
    wait: yes
    value: '"{{ dns_challenge.challenge_data[wildcard]["dns-
01"]["resource_value"] }}"'
    overwrite: yes
  when: dns_challenge.changed

- name: Wait for DNS record to expire
  pause:
    seconds: 120
  when: dns_challenge.changed

- name: Let the challenge be validated and retrieve the cert and
intermediate certificate
  acme_certificate:
    modify_account: no
    account_key_src: "{{ certificate_directory }}/account.key"
    src: "{{ certificate_directory }}/{{ le_cert_domain }}.csr"
    cert: "{{ certificate_directory }}/{{ le_cert_domain }}.crt"
    fullchain: "{{ certificate_directory }}/{{ le_cert_domain }}-
fullchain.crt"
    chain: "{{ certificate_directory }}/{{ le_cert_domain }}-
intermediate.crt"
    challenge: dns-01
    acme_version: 2
    acme_directory: 'https://acme-
v02.api.letsencrypt.org/directory'
    remaining_days: 60
    data: "{{ dns_challenge }}"
  when: dns_challenge is changed
```

tasks/upload_certificate.yml

```
---
- name: Create directory
  file:
    path: "/etc/ssl/{{ domain_name }}"
    state: directory
    owner: root
    group: root
```

```yaml
- name: Upload {{ domain_name }} certificate
  copy:
    src: "{{ certificate_directory }}/{{ domain_name }}.crt"
    dest: "/etc/ssl/{{ domain_name }}/{{ domain_name }}.crt"
    owner: root
    group: root

- name: Upload {{ domain_name }} private key
  copy:
    src: "{{ certificate_directory }}/{{ domain_name }}.key"
    dest: "/etc/ssl/{{ domain_name }}/{{ domain_name }}.key"
    owner: root
    group: root
    mode: '0600'
```

Appendix 10 – Duplicity Database Backup Deployment

backups.yml

```
---
- hosts: database
  become: true
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Configure backups
      include_role:
        name: duplicity
      vars:
        configure_duplicity_backups: True
```

tasks/main.yml

```
---
- name: Configure duplicity backup on database nodes
  include_tasks: configure_duplicity.yml
  when: configure_duplicity_backups is defined and
configure_duplicity_backups == True
```

tasks/configure_duplicity.yml

```
---
- name: Install Duplicity
  apt:
    name: duplicity
    state: latest
    update_cache: yes

- name: Install boto module
  pip:
    name: boto

- name: Upload gpg key generation script
```

```yaml
  template:
    src: gen_gpg.j2
    dest: /root/duplicity_gpg
    owner: root
    group: root
    mode: '0700'

- name: Generate gpg key
  command: 'gpg --batch --generate-key /root/duplicity_gpg'
  args:
    chdir: /root
  become: yes
  become_user: root

- name: Create duplicity credentials file
  template:
    src: duplicity_credentials.j2
    dest: /root/.duplicity
    owner: root
    group: root
    mode: '0600'
  become: yes
  become_user: root

- name: Create backup directory
  file:
    path: /srv/backup/mysql
    state: directory
    owner: root
    group: root
    mode: '0700'

- name: Upload backup script
  template:
    src: backup_script.sh.j2
    dest: /srv/backup/backup_script.sh
    owner: root
    group: root
    mode: '0700'

- name: Configure backups
  cron:
    name: Duplicity backups
    weekday: "*"
    hour: "23"
    minute: "0"
    user: root
    job: "/srv/backup/backup_script.sh {{ s3_bucket_name }}"
    cron_file: ansible_auto_backups
```

Appendix 11 – Common Package Installation Role

tasks/main.yml

```
---
- name: Install packages
  apt:
    name: ['python-apt', 'python3-apt', 'python3-pip', 'vim-nox',
'net-tools', 'mc']
    state: latest
    update_cache: yes
```

Appendix 12 – Bastion Host Deployment and Configuration

bastion.yml

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  environment:
    AWS_ACCESS_KEY_ID: "{{ aws_access_key_id }}"
    AWS_SECRET_ACCESS_KEY: "{{ aws_secret_access_key }}"
    AWS_REGION: "{{ AWS_REGION  }}"
  vars_files:
    - keys
  tasks:
    - name: Get VPC information
      include_role:
        name: vpc
      vars:
        fetch_vpc_info: True
      when: hostvars['localhost']['create_initial_vpc'] is not
defined

- hosts: bastion
  gather_facts: no
  become: true
  tasks:
    - name: Patch bastion host
      include_role:
        name: patching
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Install common packages
      include_role:
        name: common
      when: hostvars['localhost']['create_initial_vpc'] is defined
    - name: Configure bastion host
      include_role:
        name: bastion
      vars:
        configure_bastion_host: True
      when: hostvars['localhost']['create_initial_vpc'] is defined
```

tasks/main.yml

```
---
- name: Create bastion instance during VPC deployment
  include_tasks: create_instance.yml
  when: create_initial_vpc is defined and create_initial_vpc ==
True

- name: Configure bastion host
  include_tasks: configure_bastion.yml
  when: configure_bastion_host is defined and
configure_bastion_host == True
```

tasks/create_instance.yml

```
---
- name: Create bastion host
  include_role:
    name: provisioning
  vars:
    instance_name: "{{ item }}"
    instance_role: "bastion"
    subnet_id: "{{ public_subnet_id }}"
    security_group: "{{ bastion_sg_id }}"
  with_items:
    - "{{ bastion_host_name }}"

- name: Set bastion instance id to a fact
  vars:
    fact_string: "{{ item }}_instance_info"
  set_fact:
    "{{ item }}_host_id": "{{
hostvars['localhost'][fact_string].instance_ids | join }}"
  with_items:
    - "{{ bastion_host_name }}"

- name: Assign elastic IP to bastion host
  ec2_eip:
    device_id: "{{ item }}"
    in_vpc: true
    state: present
  register: elastic_ip
  with_items:
    - "{{ bastion_host_id }}"
- name: Set bastion host IP to a variable
  set_fact:
    "{{ item }}_ip": "{{ elastic_ip.results |
map(attribute='public_ip') | join }}"
  with_items:
    - "{{ bastion_host_name }}"

- name: Add public_ip to in-memory group
  vars:
    ip_string: "{{ item }}_ip"
  add_host:
    hostname: "{{ item }}"
    ansible_host: "{{ hostvars['localhost'][ip_string] }}"
    groups:
      - bastion
  with_items:
    - "{{ bastion_host_name }}"

- name: Create dynamic inventory file
  template:
    src: aws_ec2.j2
    dest: ./aws_ec2.yml
```

tasks/configure_bastion.yml

```
---
# Configure SSH
- name: Configure SSH
  include_role:
    name: ssh

# Configure users and groups
- name: Configure users
  include_role:
    name: users
```