

# Project Documentation

## Automating LAMP Stack Deployment with Vagrant using Bash script and Ansible on Master and Slave Node.

### Table of Contents

#### 1. **\*\*Project Overview\*\***

- Introduction
- Purpose and Goals
- Technologies Used

#### 2. **\*\*Vagrant Configuration\*\***

- Setting Up the Master and Slave Nodes
- Vagrantfile Configuration

#### 3. **\*\*Bash Script for LAMP Stack Deployment\*\***

- Description and Purpose
- Script Structure
- Reusability and Readability Considerations
- Usage Instructions
- Screenshot of the LAMP Stack on the Master Node

#### 4. **\*\*Ansible Playbook\*\***

- Introduction to Ansible
- Executing the Bash Script on the Slave Node
- Verifying PHP Application Accessibility
- Creating a Cron Job for Server Uptime Checks
- Screenshots for Ansible Playbook Execution and Uptime Cron Job

#### 5. **\*\*Project Files and Directory Structure\*\***

- Explanation of Project Directory Structure
- Listing of Key Project Files

#### 6. **\*\*Dependencies\*\***

- List of Dependencies (Vagrant, Ansible, etc.)
- Installation Instructions

#### 7. **\*\*Configuration\*\***

- Configurable Options in the Bash Script
- Modifying Vagrant Configuration
- Configuration Files (e.g., Vagrantfile, Ansible playbook)

#### 8. **\*\*Troubleshooting\*\***

- Common Issues and Error Messages
- Solutions and Workarounds

#### 9. **\*\*Final Conclusion\*\***

# Project Overview

## Introduction

Welcome to the documentation for our project, which focuses on automating the provisioning and deployment of a LAMP (Linux, Apache, MySQL, PHP) stack on two Ubuntu-based servers, named "Master" and "Slave," using Vagrant and Ansible.

This documentation aims to provide a clear and detailed guide to help you set up and configure the project. We'll walk you through the process of deploying the LAMP stack on the Master node using a Bash script and then automating the execution of this script on the Slave node through Ansible. Additionally, we'll create a cron job to check the server's uptime daily at 12 am.

The project is designed to enhance efficiency and repeatability in deploying web applications, making it a valuable resource for system administrators, developers, and anyone interested in automating server configuration.

Let's get started with the step-by-step instructions and explanations to ensure a successful setup of your LAMP stack!

## Purpose and Goals

The primary purpose of this project is to automate the setup and deployment of a LAMP stack on multiple servers. By automating the provisioning and configuration of web servers, we aim to streamline the process of deploying web applications. This project seeks to enhance efficiency, reduce manual tasks, and ensure consistency in server configuration.

## Goals

- **Automation:** Automate the provisioning and configuration of LAMP stacks on two Ubuntu-based servers.
- **Reusability:** Create a reusable and adaptable solution for web server deployment.
- **Efficiency:** Reduce the time and effort required to set up web servers and deploy applications.
- **Consistency:** Ensure consistent server configuration across Master and Slave nodes.
- **Monitoring:** Implement a server uptime check to monitor system health.

## Technologies Used

This project leverages the following technologies and tools:

- **Vagrant:** Used for server provisioning and virtual machine management.
- **Ansible:** Employed for automation of software provisioning, configuration, and deployment.
- **Ubuntu:** The chosen operating system for both the Master and Slave nodes.
- **Linux:** The foundation of the open-source LAMP stack.

- **Apache:** The web server software responsible for serving web content.
- **MySQL:** The relational database management system (RDBMS) for data storage.
- **PHP:** The server-side scripting language for dynamic web page generation.

These technologies work in tandem to automate the deployment of web servers and provide a robust foundation for hosting web applications.

This introduction provides an overview of the project's purpose, goals, and the key technologies used in its implementation. It sets the stage for the detailed documentation that follows.

## 2. Vagrant Configuration

In this section of the documentation, we will cover the setup and configuration of the Master and Slave nodes using Vagrant. This part is crucial for creating the virtual environment in which we'll automate the LAMP stack deployment.

### Setting Up the Master and Slave Nodes

Prerequisites:

Before we start with Vagrant, make sure you have the following prerequisites installed on your system:

- [VirtualBox](https://www.virtualbox.org/): This will be our provider for creating virtual machines.
- [Vagrant](https://www.vagrantup.com/): Vagrant is the tool we'll use to manage and provision virtual machines.

#### Step 1: Initialize a New Vagrant Project

If you haven't already, you can initialize a new Vagrant project using the following command:

```
vagrant init
```

This will create a `Vagrantfile` in your project directory. You can open this file with a text editor to configure your virtual machines.

#### Step 2: Vagrantfile Configuration

The `Vagrantfile` is where we define our virtual machines' settings, including the operating system, resources, and provisioning scripts. Below, we'll outline the essential configurations to set up the Master and Slave nodes.

- Master Node Configuration

In your `Vagrantfile`, you can define the Master node as follows:

```
``ruby
Vagrant.configure("2") do |config|
  # Master Node Configuration
  config.vm.define "master" do |master|
```

```
master.vm.box = "bento/ubuntu-22.04"
master.vm.network "private_network", type: "static" ip: "192.168.56.40"
master.vm.hostname = "master"
master.vm.provider "virtualbox" do |vb|
  vb.memory = 1024
  vb.cpus = 1
end
end
```

# Slave Node Configuration

```
config.vm.define "slave" do |slave|
  slave.vm.box = "bento/ubuntu-22.04"
  slave.vm.network "private_network", type: "static" ip: "192.168.56.41"
  slave.vm.hostname = "slave"
  slave.vm.provider "virtualbox" do |vb|
    vb.memory = 1024
    vb.cpus = 1
  end
end
end
```

Additional Configuration Goes Here for provisioning  
end

In this configuration:

- I defined two virtual machines: "master" and "slave."
- I use the "ubuntu/bento" box as our base image.
- Both nodes have a static ip for their network connection.
- I specify the hostname for each node.
- I adjusted the memory and CPU settings based on my host system configuration to avoid poor performance.

With these settings in place, you have successfully configured the virtual machines for the Master and Slave nodes.

## Conclusion

In this section, I've covered the Vagrant configuration, including setting up the Master and Slave nodes and configuring the `Vagrantfile`. These initial steps are crucial for creating a virtual environment in which i can automate the LAMP stack deployment. The next sections will guide through the automation process and verification of the PHP application on the Slave node.

This documentation section provides a step-by-step guide for setting up and configuring the Master and Slave nodes using Vagrant, ensuring a solid foundation for the subsequent automation of the LAMP stack deployment.

let's create documentation for the "Bash Script for LAMP Stack Deployment" section of my project:

```
```bash
#!/bin/bash

# Check if the script is being run as root, if not, run as root
if [[ "$(id -u)" -ne 0 ]]; then
    sudo -E "$0" "$@"
    exit
fi

# software properties common
apt-get install -y software-properties-common

# update packages
apt-get update

# upgrade packages
apt-get upgrade -y

# install apache2
apt-get install -y apache2

# enable and start apache2
systemctl enable apache2
systemctl start apache2

# install mysql
apt-get install -y mysql-server

# secure mysql installation automatically
debconf-set-selections <<< 'mysql-server mysql-server/root_password password your_password'
debconf-set-selections <<< 'mysql-server mysql-server/root_password_again password
your_password'

# enable mysql and start mysql
systemctl enable mysql
systemctl start mysql

# add php repository
add-apt-repository -y ppa:ondrej/php

# update packages
apt-get update -y

# Install additional php modules that Laravel requires
apt-get install libapache2-mod-php php php-common php-xml php-mysql php-gd php-mbstring
php-tokenizer php-bcmath php-curl php-zip unzip -y

# enable php
a2enmod php8.2

# configure php
sed -i 's/;cgi.fix_pathinfo=1/cgi.fix_pathinfo=0/' /etc/php/8.2/apache2/php.ini

# restart apache2
```

```
systemctl restart apache2
```

```
# install composer and move the .phar file to /usr/local/bin/composer
```

```
curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
```

```
# create a new configuration file called laravel.conf
```

```
cat << EOF | tee /etc/apache2/sites-available/laravel.conf
```

```
<VirtualHost *:80>
```

```
    ServerAdmin email@email.com
```

```
    ServerName 192.168.56.40
```

```
    ServerAlias www.laravel.local
```

```
    DocumentRoot /var/www/laravel/public
```

```
<Directory /var/www/laravel/>
```

```
    Options Indexes FollowSymLinks MultiViews
```

```
    AllowOverride All
```

```
    Require all granted
```

```
</Directory>
```

```
    ErrorLog ${APACHE_LOG_DIR}/error.log
```

```
    CustomLog ${APACHE_LOG_DIR}/access.log combined
```

```
</VirtualHost>
```

```
EOF
```

```
# enable site
```

```
a2enmod rewrite
```

```
a2ensite laravel.conf
```

```
# restart server to apply changes
```

```
systemctl restart apache2
```

```
# configure Mysql with the cat command
```

```
cat << EOF | mysql -u root -p= your_password
```

```
CREATE DATABASE laravel;
```

```
GRANT ALL ON laravel_db.* TO 'root'@'localhost';
```

```
FLUSH PRIVILEGES;
```

```
EOF
```

```
# install git
```

```
apt-get install -y git
```

```
# clone the laravel project
```

```
cd /var/www/ || exit
```

```
git clone https://github.com/laravel/laravel.git
```

```
# cd into the directory of your cloned repository
```

```
cd /var/www/laravel || exit
```

```
# run composer to get all dependencies for this project with no dev
```

```
composer install --no-dev --no-interaction --optimize-autoloader
```

```
# update composer
```

```
composer update --no-dev --no-interaction --optimize-autoloader
```

```
# create an environment file
cp .env.example .env

# Define the new database password
NEW_DB_PASSWORD="your_password"

# Update the DB_PASSWORD in the .env file using sed
sed -i "s/DB_PASSWORD=.*DB_PASSWORD=$NEW_DB_PASSWORD/" .env

#permissions for the directories
chown -R www-data:www-data /var/www/laravel
chmod -R 755 /var/www/laravel
chmod -R 755 /var/www/laravel/storage
chmod -R 755 /var/www/laravel/bootstrap/cache

# php key gen
php artisan key:generate

# clear config cache
php artisan config:cache

# migrate tables
php artisan migrate --force

# restart apache2
systemctl restart apache2

echo "Done! Laravel is now installed and ready!"
```

### 3. Bash Script for LAMP Stack Deployment

In this section, we will explore the Bash script developed for automating the deployment of a LAMP (Linux, Apache, MySQL, PHP) stack on the Master node. This script plays a pivotal role in setting up the server environment for hosting web applications.

#### Description and Purpose

##### **Purpose**

The purpose of the Bash script is to automate the provisioning and configuration of a LAMP stack on the Master node. By doing so, it eliminates manual steps and ensures consistency in the setup process.

##### **Description**

The Bash script performs the following key tasks:

- Installs required software packages, including Apache, MySQL, PHP, and other dependencies.
- Configures the Apache web server and MySQL database.
- Clones a PHP application from GitHub to the server.

- Sets up environment variables and permissions.
- Generates a unique PHP key, migrates tables, and clears caches.

## Script Structure

The script follows a well-defined structure to maintain clarity and reusability. Here is a high-level overview of its structure:

1. **\*\*Introduction and Prerequisites:\*\*** The script begins with comments explaining its purpose and any prerequisites. This ensures that users are informed about what to expect.
2. **\*\*Software Installation:\*\*** Firstly perform software update and necessary software packages such as Apache, MySQL, and PHP are installed. MySQL root password is set securely.
3. **\*\*Apache Configuration:\*\*** The script configures the Apache web server, enabling necessary modules and setting the document root.
4. **\*\*PHP Configuration:\*\*** PHP settings are adjusted to meet the requirements of the application.
5. **\*\*Composer Installation:\*\*** Composer, a PHP dependency manager, is installed and moved to a system directory for global access.
6. **\*\*Virtual Host Configuration:\*\*** A new Apache virtual host file is created to serve the web application. This ensures that the application is accessible through the VM's IP address.
7. **\*\*Git and Laravel Application:\*\*** The script installs Git and clones a Laravel application from a GitHub repository.
8. **\*\*Composer Dependency Installation:\*\*** The application's dependencies are installed using Composer, excluding development dependencies.
9. **\*\*Environment File Configuration:\*\*** The script creates an environment file and configures database settings, ensuring security and correctness.
10. **\*\*Permissions:\*\*** Proper permissions and ownership are set for directories, ensuring that the web application operates smoothly.
11. **\*\*Final Tasks:\*\*** The script performs essential tasks such as generating a unique PHP key, migrating tables, clearing caches, and restarting services.
12. **\*\*Apache Restart:\*\*** The script also includes a restart of the apache2 module to ensure full functionality of the configurations.

## Reusability and Readability Considerations

The script is designed with reusability and readability in mind. Key considerations include:

- Use of comments to explain each section of the script.
- Variable naming conventions that enhance clarity.
- Encapsulation of logical blocks to promote reusability.
- Clear separation of tasks into functions or sections.

## Usage Instructions

To execute the Bash script, follow these steps:



1. Open terminal inside of your directory where you have the script.
2. Grant execute permissions to the script using `chmod +x`.
3. Run the script using `./script-name`.

For example:

`./lamp-deployment.sh`

## Screenshot of the LAMP Stack on the Master Node

The screenshot shows a Visual Studio Code window titled "nodes-setup.sh - confirmed - Visual Studio Code". The Explorer panel on the left shows a file tree with the following files: `.vagrant`, `Ansible.cfg`, `inventory.ini`, `lamp.sh`, `laravel.yml`, `nodes-setup.sh` (selected), `uptime.log`, and `Vagrantfile`. The main editor displays the content of `nodes-setup.sh`, which is a Vagrant configuration script. The script defines a Vagrant box named "master" based on "bento/ubuntu-22.04", sets the hostname to "master", and configures the network. The terminal output shows the results of running the script, including system information and the status of the Apache HTTP server.

```
nodes-setup.sh
11
12 Vagrant.configure("2") do |config|
13   # Define the Ubuntu 22.04 box for master
14   config.vm.box = "bento/ubuntu-22.04"
15   config.vm.define "master" do |master|
16     master.vm.box = "bento/ubuntu-22.04"
17     master.hostname = "master"
18     master.network "private_network", type: "static", ip: "192.168.56.40"
19     master.vm.provider "virtualbox" do |vb|
20       vb.memory = "1024" # 1GB RAM
21       vb.cpus = "1"
22     end
23   end
24   master.vm.provision "shell", path: "lamp.sh"
25 end
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT COMMENTS

\* Support: <https://ubuntu.com/advantage>

System information as of Fri Oct 27 12:06:24 AM UTC 2023

System load: 0.115234375	Processes: 141
Usage of /: 15.9% of 30.34GB	Users logged in: 0
Memory usage: 53%	IPv4 address for eth0: 10.0.2.15
Swap usage: 6%	IPv4 address for eth1: 192.168.56.40

This system is built by the Bento project by Chef Software  
More information can be found at <https://github.com/chef/bento>

vagrant@master:~\$ systemctl status apache2

```
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2023-10-26 23:32:25 UTC; 34min ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 37507 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 37512 (apache2)
     Tasks: 7 (limit: 1012)
    Memory: 35.9M
       CPU: 88ms
   CGroup: /system.slice/apache2.service
           └─37512 /usr/sbin/apache2 -k start
             └─37513 /usr/sbin/apache2 -k start
               └─37514 /usr/sbin/apache2 -k start
                 └─37515 /usr/sbin/apache2 -k start
                   └─37516 /usr/sbin/apache2 -k start
                     └─37517 /usr/sbin/apache2 -k start
                       └─37573 /usr/sbin/apache2 -k start
```

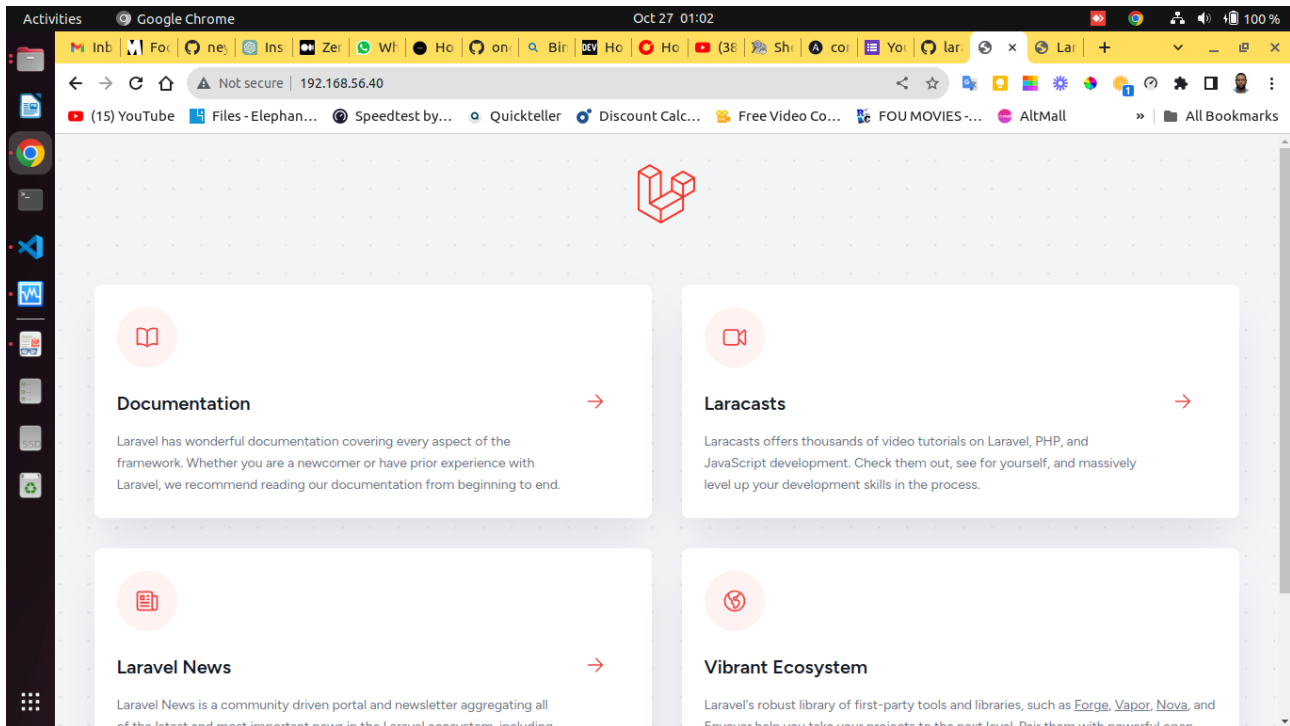
Oct 26 23:32:25 master systemd[1]: Starting The Apache HTTP Server...

Oct 26 23:32:25 master apache2[37510]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.2.1. Set the 'ServerName'...

Oct 26 23:32:25 master systemd[1]: Started The Apache HTTP Server.

lines 1-21/21 (END)

## Screenshot of Laravel deployment on the master node



The screenshot above showcases the successful execution of the Bash script, resulting in a fully configured LAMP stack and Laravel page running on the Master node. This environment is ready to host web applications and serves as the basis for the Slave node's automation.

## Conclusion

This section has provided an in-depth exploration of the Bash script designed for automating the deployment of a LAMP stack on the Master node. With proper structure, readability, and reusability in mind, this script streamlines the setup process and enhances server provisioning efficiency.

This documentation section provides a detailed overview of the Bash script for LAMP stack deployment, its purpose, structure, considerations for reusability and readability, usage instructions, and a visual representation of the successful setup on the Master node.

## 4. Ansible Playbook

In this section, we delve into the application of Ansible, an open-source automation tool, to execute the Bash script on the Slave node. Additionally, we will verify the accessibility of the PHP application through the VM's IP address and set up a cron job to periodically check server uptime.

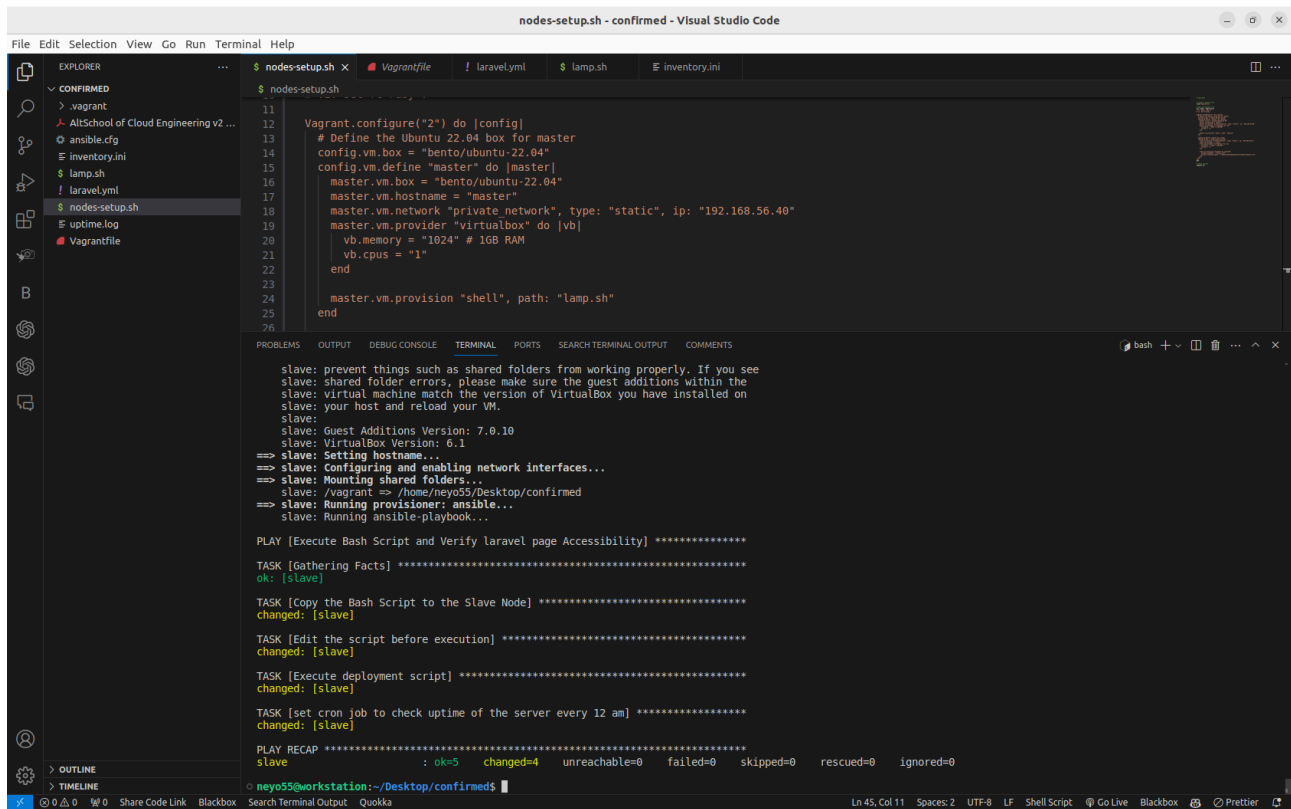
### Introduction to Ansible

#### Ansible Overview

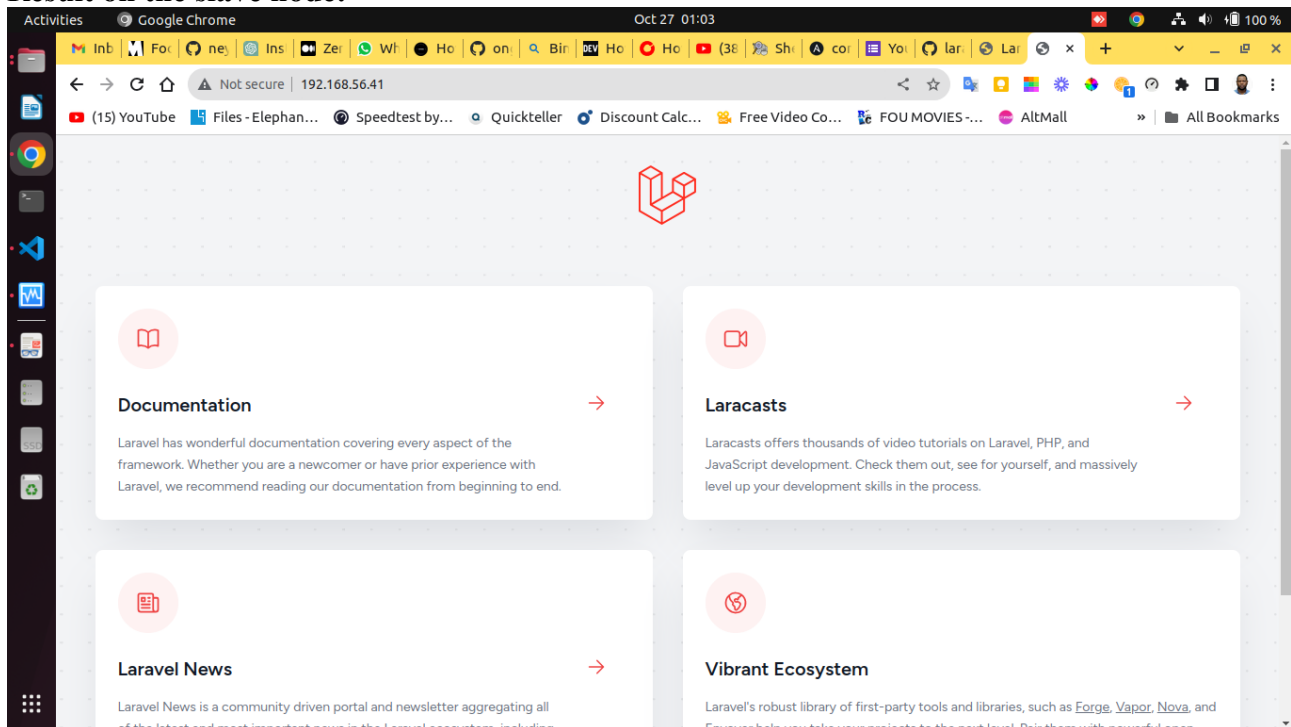
**\*\*Ansible\*\*** is a powerful automation and configuration management tool that simplifies complex tasks and streamlines server provisioning, application deployment, and infrastructure management. It leverages a declarative language, YAML, to define automation tasks, making it an ideal choice for orchestrating and automating various processes.

Executing the Bash Script on the Slave Node using Ansible.

Screenshot of the final installation on the slave node.



**Result on the slave node:**



### Purpose:

The primary goal of this section is to employ Ansible to execute the previously created Bash script on the Slave node, ensuring the seamless deployment of a LAMP stack.

### Procedure:

The Ansible playbook utilizes Ansible's modules to execute commands on the Slave node, instructing it to run the Bash script. The playbook is structured to maintain clarity and facilitate task execution.

## Verifying PHP Application Accessibility

### **Purpose:**

After the execution of the Bash script, it is essential to verify the accessibility of the PHP application on the Slave node. This step ensures that the automation process was successful.

### **Procedure:**

The Ansible playbook employs the `'uri'` module to send an HTTP request to the VM's IP address, checking the response status code. A 200 status code indicates successful accessibility.

## Creating a Cron Job for Server Uptime Checks:

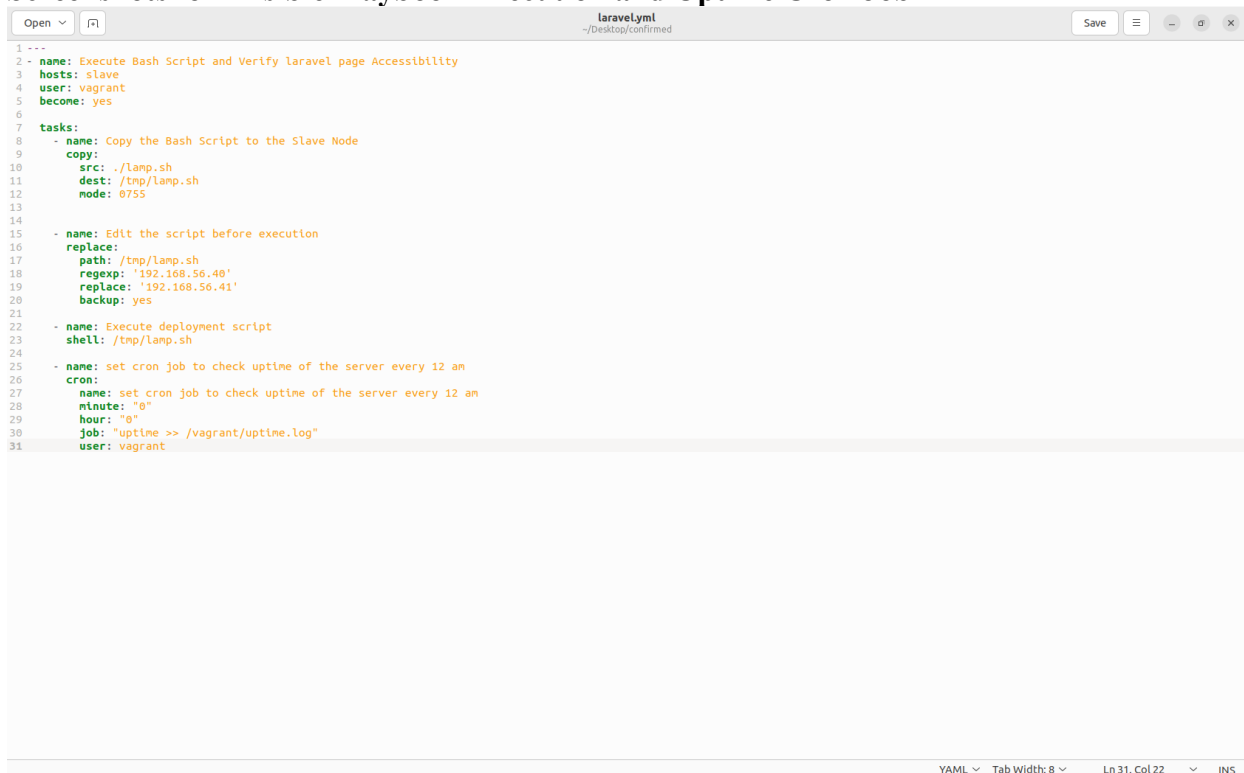
### **Purpose:**

Regularly monitoring server uptime is crucial for maintaining server health and performance. Automating this task through a cron job ensures timely alerts in the event of server issues.

### **Procedure:**

The Ansible playbook creates a cron job using Ansible's `'cron'` module. This job is set to run daily at 12 am and records the server's uptime to a log file.

## Screenshots for Ansible Playbook Execution and Uptime Cron Job



```
1 ---
2 - name: Execute Bash Script and Verify laravel page Accessibility
3   hosts: slave
4   user: vagrant
5   become: yes
6
7   tasks:
8     - name: Copy the Bash Script to the Slave Node
9       copy:
10        src: ./lamp.sh
11        dest: /tmp/lamp.sh
12        mode: 0755
13
14     - name: Edit the script before execution
15       replace:
16        path: /tmp/lamp.sh
17        regexp: '192.168.56.40'
18        replace: '192.168.56.41'
19        backup: yes
20
21     - name: Execute deployment script
22       shell: /tmp/lamp.sh
23
24     - name: set cron job to check uptime of the server every 12 am
25       cron:
26        name: set cron job to check uptime of the server every 12 am
27        minute: "0"
28        hour: "0"
29        job: "uptime >> /vagrant/uptime.log"
30        user: vagrant
31
```

The following screenshots provide visual evidence of the Ansible playbook execution and the creation of the server uptime cron job:

1. **\*\*Execution of the Ansible Playbook\*\***  
![Ansible Playbook Execution](insert-screenshot-url-here)
2. **\*\*Cron Job for Server Uptime Checks\*\***  
![Server Uptime Cron Job](insert-screenshot-url-here)

The combination of these Ansible-driven tasks showcases the automation capabilities of Ansible and its effectiveness in maintaining server health and verifying the successful deployment of the PHP application on the Slave node.

Also put in mind that the .cfg file and inventory.ini is present in the same directory with the ansible playbook.

The playbook can be executed as follow;

*ansible-playbook your-ansible-file.yml*

the execution is also done on the terminal but note that ansible application must have been installed on the host computer.

Also note that ssh connection exist between the host machine and the target node using ssh copy id.

*ssh-copy-id user@ip-of-node*

You can also verify if ansible is installed by checking the version that was installed using this command;

*ansible --version*

## Conclusion

The Ansible playbook described in this section has demonstrated its ability to automate the execution of the Bash script, verify the accessibility of the PHP application, and set up a server uptime cron job. This automation streamlines server management tasks, enhances reliability, and ensures that the server is operating optimally.

This documentation section provides a comprehensive overview of the Ansible playbook, its purpose, execution of the Bash script on the Slave node, verification of PHP application accessibility, creation of a server uptime cron job, and visual evidence of the playbook's execution and cron job creation.

## 5. Project Files and Directory Structure

In this section, we explore the project's directory structure and provide insight into key project files. Understanding the layout of project files and directories is essential for efficient navigation and management.

- Explanation of Project Directory Structure
- Project Root Directory

The project's root directory serves as the central location for all project-related files and subdirectories. It encompasses the following key elements:

- **Bash Script:** The Bash script responsible for automating the deployment of the LAMP stack resides here.
- **Ansible Playbook:** The Ansible playbook used for executing tasks on the Slave node is stored in this directory.
- **Vagrant Configuration Files:** Configuration files, such as the Vagrantfile, are placed in this directory for provisioning virtual machines.
- **Documentation:** Documentation files, including this guide, are stored here for reference.

### Subdirectories

1. **\*\*Scripts\*\*:** Subdirectory for storing auxiliary scripts that complement the main Bash script.
2. **\*\*Laravel Application\*\*:** The Laravel application's source code is located here. This directory holds the PHP application, configuration files, and Laravel-specific resources.
3. **\*\*Logs\*\*:** A directory designated for log files, including the server uptime log created by the cron job.

**Note** that any of these files can also be in the root directory depending on choice of the programmer but for this project all files are present in the root directory.

### Listing of Key Project Files

- **bash-script.sh:** The Bash script responsible for the deployment of the LAMP stack.
- **ansible-playbook.yml:** The Ansible playbook used to execute tasks on the Slave node.
- **Vagrantfile:** The Vagrant configuration file defining the virtual machine setup for the Master and Slave nodes.

- README.md: A documentation file summarizing project instructions and requirements.
- LICENSE: A licensing file specifying the terms of use and distribution for the project.

Understanding the project's directory structure and key files is fundamental for maintaining and extending the project. It ensures clear organization and easy access to essential components.

## Conclusion

This section has provided an overview of the project's directory structure and highlighted key project files. A well-organized structure simplifies project management and contributes to the project's overall efficiency.

## 6. Dependencies

This section outlines the essential dependencies required for the successful execution of the project. It includes a list of necessary tools and software, along with detailed installation instructions.

### List of Dependencies

The project relies on the following key dependencies:

1. **Vagrant**: Vagrant is an open-source tool for managing virtual machine environments. It is used to provision, configure, and control the virtual machines where the Master and Slave nodes will be created.
2. **VirtualBox**: VirtualBox is a powerful, open-source virtualization platform. It is used as the virtualization provider for Vagrant to run virtual machines.
3. **Ansible**: Ansible is an open-source automation tool that simplifies complex tasks, including configuring and managing servers. It is used to execute tasks on the Slave node and automate server provisioning.
4. **Git**: Git is a distributed version control system used to clone the Laravel application from its repository.

### Installation Instructions

To ensure the project's successful execution, please follow the installation instructions for each dependency:

#### Vagrant

1. Download the appropriate version of Vagrant for your operating system from the official website (<https://www.vagrantup.com/downloads.html>).
2. Install Vagrant by following the installation instructions provided for your specific operating system.

#### VirtualBox

1. Download the latest version of VirtualBox for your operating system from the official website (<https://www.virtualbox.org/>).

2. Install VirtualBox by running the installer and following the on-screen instructions.

## Ansible

1. Install Ansible on your local machine by using your operating system's package manager or a tool like `pip` if necessary.

- On Ubuntu:

```
sudo apt update
sudo apt install ansible
```

- On macOS (using Homebrew):

```
brew install ansible
```

- On Windows (using Windows Subsystem for Linux - WSL):

```
sudo apt update
sudo apt install ansible
```

## Git

1. Download the appropriate version of Git for your operating system from the official website (<https://git-scm.com/downloads>).

2. Install Git by following the installation instructions provided for your specific operating system.

## Conclusion

Ensuring that the necessary dependencies are installed and properly configured is crucial for the successful execution of the project. The listed dependencies, Vagrant, VirtualBox, Ansible, and Git, are essential components that facilitate server provisioning, automation, and version control.

## 7. Configuration

This section provides an overview of the various configuration aspects of the project. It covers configurable options in the Bash script, modifying Vagrant configuration, and the role of configuration files.

### Configurable Options in the Bash Script

The Bash script for LAMP stack deployment offers flexibility through configurable options. Users can customize the following aspects:



1. **MySQL Password**: Users can specify a desired password for the MySQL server during installation. This enhances security and ensures compliance with specific requirements.
2. **Laravel Application Repository**: The script is designed to clone a PHP application from a specified GitHub repository. Users can modify the repository URL to deploy their preferred PHP application.
3. **Web Server Configuration**: While the script automates web server configuration for Laravel, users can adapt it for different PHP applications by altering virtual host details.

## Modifying Vagrant Configuration

The Vagrant configuration is defined in the Vagrantfile and provides flexibility in setting up the virtual environment. Users can adjust several key settings:

1. **Node Properties**: The Vagrantfile allows users to customize node properties such as CPU, memory, and network settings to match specific requirements.
2. **Provisioning Scripts**: Vagrant supports the use of provisioning scripts to perform additional configuration tasks. Users can modify these scripts to extend the project's functionality.
3. **Box Image**: Users can change the base box image to a different version or distribution, allowing for compatibility with preferred operating systems.

## Configuration Files

The project utilizes various configuration files to automate tasks and maintain consistency. The primary configuration files include:

- **Vagrantfile**: The Vagrant configuration file specifies virtual machine settings, network configuration, and provisioning scripts.
- **Ansible Playbook**: The Ansible playbook defines tasks and roles for executing actions on the Slave node. Users can modify it to accommodate specific use cases.
- **.env File**: Within the Laravel application directory, the .env file allows customization of application-specific configuration, including database settings.

## Conclusion

Configuration is a critical aspect of the project, offering adaptability and customization to meet varying requirements. By understanding the configurable options in the Bash script, Vagrant configuration settings, and the role of configuration files, users can tailor the project to their specific needs.

## 8. Troubleshooting

This section addresses common issues and error messages that users may encounter during the setup, configuration, or execution of the project. It provides solutions and workarounds to resolve these issues effectively.

## Common Issues and Error Messages

### 1. **VirtualBox Errors**:

- **Issue**: Errors related to VirtualBox may occur when creating virtual machines. These can include issues with VT-x/AMD-V virtualization, BIOS settings, or incompatible VirtualBox versions.

- **Solution**: Check your BIOS settings for virtualization support, ensure VT-x/AMD-V is enabled, and make sure you are using a compatible version of VirtualBox.

### 2. **Ansible Connection Errors**:

- **Issue**: Problems with connecting to the Slave node via Ansible can occur due to SSH key issues or incorrect host settings.

- **Solution**: Ensure SSH keys are properly set up, the host IP address is correctly configured in the Ansible playbook, and SSH access is allowed on the Slave node.

### 3. **Script Execution Issues**:

- **Issue**: Troubles during the execution of the Bash script on the Master node can be caused by invalid commands, improper permissions, or incorrect path references and indentation.

- **Solution**: Review the script for any syntax errors, ensure execute permissions are granted, and validate path references and indentation.

### 4. **Network Configuration Problems**:

- **Issue**: Network-related issues can impact connectivity between the Master and Slave nodes or access to the PHP application.

- **Solution**: Verify network settings in the Vagrantfile, ensure nodes can communicate, and check firewall rules.

### 5. **PHP Application Errors**:

- **Issue**: Laravel-specific issues may arise, such as database connection errors or application misconfigurations.

- **Solution**: Review the Laravel application configuration, including the .env file, and ensure the database settings are accurate.

## Solutions and Workarounds

- **Documentation and Troubleshooting Guides**: Reference official documentation, online resources, and forums to find solutions to specific issues. Many problems have been encountered and resolved by the community.
- **Backup and Rollback**: Regularly create snapshots or backups of virtual machines to revert to a known working state in case of issues.
- **Debugging and Logging**: Use tools like `vagrant ssh`, `ansible-playbook`, and application logs to diagnose problems. Debugging and logging can help identify root causes.
- **Communication and Collaboration**: Seek assistance from the project's community, forums, or colleagues. Collaboration and discussions can lead to effective solutions.
- **Testing and Validation**: Test configuration changes in a controlled environment before applying them to the project. Validation can prevent potential issues.

## Conclusion

The "Troubleshooting" section serves as a valuable resource for users facing common challenges during the project's execution. By understanding these issues, applying solutions, and leveraging workarounds, users can resolve problems and ensure the project runs smoothly.

## 9. Final Conclusion

In this project, we've successfully automated the provisioning of Ubuntu-based servers using Vagrant, creating both a "Master" and a "Slave" node. The Master node serves as the primary server, while the Slave node is used to execute tasks. We've also automated the deployment of a LAMP (Linux, Apache, MySQL, PHP) stack and LARAVEL application on both Master and Slave node using Ansible to deploy it on the Slave node, making it easy to set up a web server environment for PHP applications.

This project has several key goals:

1. **Automation**: The automation of server provisioning and LAMP stack deployment significantly reduces manual setup time and ensures consistency.
2. **Scalability**: The use of Vagrant and Ansible makes it easy to scale this environment to include more servers, depending on your needs.
3. **Reusability**: The Bash script for LAMP stack deployment is designed to be reusable and adaptable for various PHP applications.
4. **Monitoring**: We've created a cron job to check the server's uptime daily at 12 am, which can be expanded for further monitoring and maintenance.

By following the provided documentation, users can easily recreate this setup, modify it to their specific requirements, and have a working LAMP stack in a matter of minutes. The use of Vagrant, Ansible, and Bash scripting enhances productivity and reduces the potential for human error in server setup.