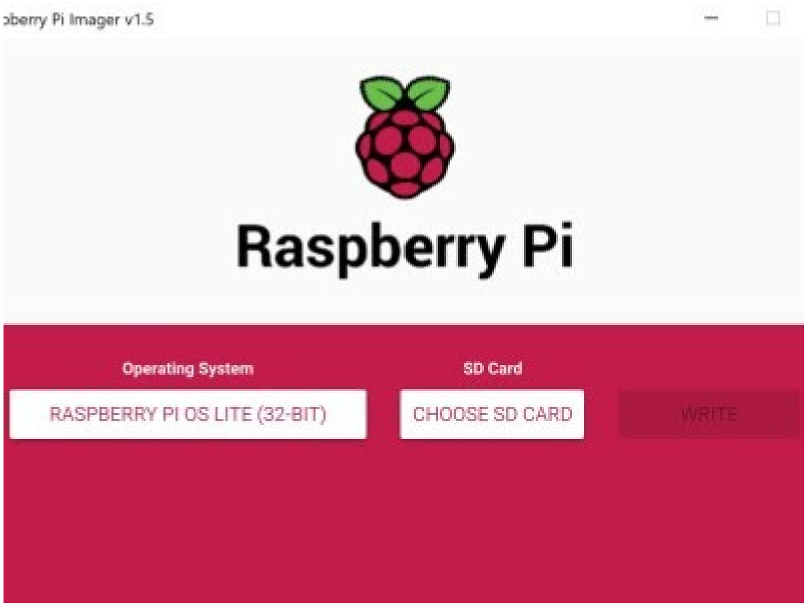
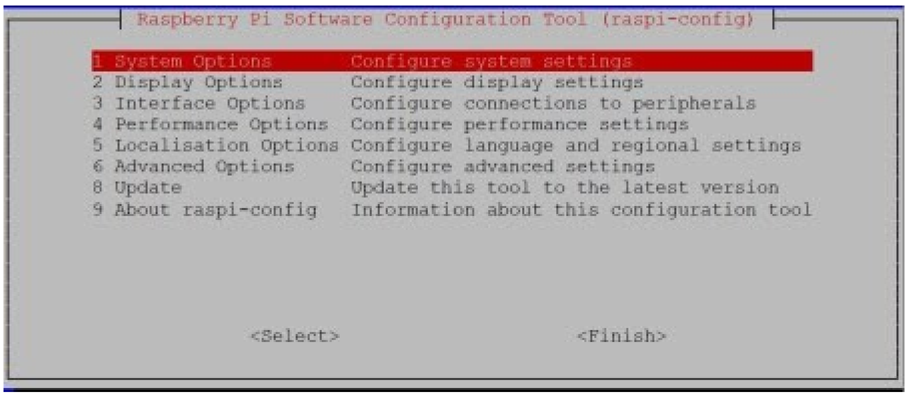
Prepare the Raspberry Pi

Even if most of the process can be automated with Ansible, we still need at least one working Raspberry Pi to configure, so, we need to start by loading a fresh image of Raspberry Pi OS Lite (32-bit) onto our SD card (I recommend at least 32GB class 10 HC1).

Download “Raspberry Pi Imager” from the official site, install it in your system, execute it, select Raspberry Pi OS Lite (32-bit) as the “Operating System”, choose your “SD Card” and press “Write”



Once the process is done, you have a clean installation of Raspberry Pi OS Lite in your micro SD card. Insert the card into your Raspberry Pi, plug a keyboard and a screen, and turn it on so you can set some basic configurations with `raspi-config`.



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

             <Select>                       <Finish>
```

Log into the system (the default user and password are `pi` and `raspberry` respectively), run `sudo raspi-config`, and perform the following tasks:

- Optionally, change the password for the “pi” user (System Options –> Password)
- Enable the SSH server (Interface Options –> SSH)
- Reduce GPU memory to 16MB (Performance Options –> GPU memory)
- Set your locale settings (Localization Options)
- Expand the filesystem to use the full capacity of your SD card (Advanced Options –> Expand Filesystem)
- Disable Predictable Network Interface Names (Advanced Options –> Network Interface

Names)

When you are done, exit the `raspi-config` tool and reboot your Pi with `sudo reboot now`.

Next, you have to create an SSH key, even if it is possible to use Ansible by providing a password for your hosts interactively, a more convenient and secure way to do it is to use an SSH key pair. To create one run the following commands in the terminal:

```
# Create the .ssh folder
cd ~
mkdir .ssh
cd .ssh
# Create the authorized_keys file
touch authorized_keys

# Set proper access permissions
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys

# Create the key pair
ssh-keygen
# Add the new key to the authorized_keys file
cat id_rsa.pub >> authorized_keys

# If you use Putty as your SSH client (Windows), generate a ppk key,
otherwise, ignore these steps
sudo apt install putty-tools
puttygen id_rsa -o id_rsa.ppk
```

Copy the id_rsa and id_rsa.pub (and id_rsa.ppk if applicable) files to the machine you are going to use to execute the playbooks, I like to use `sftp` but you can do it the way you prefer (e.g. with a USB memory). After you have copied the key files, delete them from your Pi with `rm ~/.ssh/id_rsa*`

The next step is to set a static IP so you can always know where to reach your Pi, edit the `dhcpcd.conf` file accordingly to your own needs with `sudo nano /etc/dhcpcd.conf`.

This is a sample IP configuration for the ethernet interface:

```
# Sample static IP configuration
interface eth0
static ip_address=192.168.0.10/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1 8.8.8.8
```

Now your Pi is ready to go, disconnect the keyboard and monitor, plug the Pi into your network (I recommend using a wired connection), and reboot it.

Install Ansible

In theory, you could install Ansible on the Raspberry Pi itself and run the playbooks directly on it with `connection: local` but it wouldn't be practical, it is better to run the playbooks from a computer other than your Pi.

For installing Ansible, if you have a Unix based system (i.e. Linux, macOS) at your disposal, you are golden, installing the latest Ansible version is very simple.

```
# Install pip3
sudo apt install python3-pip
# Install the latest Ansible version with pip3
sudo pip3 install ansible
```

If you are on Windows, sadly, there is no way to run Ansible natively, your best bet is to enable WSL (Windows Subsystem for Linux), install a Linux distribution from the Microsoft Store (I recommend Ubuntu), and execute the previous steps in your Linux VM.

A detailed guide for installing WSL can be found here and more information about installing Ansible can be found on the official documentation site.

Also, since we are going to be installing several packages from source on a low power SBC, some of these tasks are going to take a while and the `SSH` connection might get automatically closed due to inactivity, to prevent this situation, activate the sending of “keep-alive” packets to the server by editing the `ssh_config` file with `sudo nano /etc/ssh/ssh_config` and adding this two lines under `Host *`:

```
    ServerAliveInterval 300
    ServerAliveCountMax 2
```

> ⚠️ Is very important not to skip the previous step, otherwise, the ssh connection is going to silently fail while running long tasks in Ansible and you are going to be waiting pointlessly with your Raspberry Pi doing nothing in reality.

Download and Configure the Playbooks

The playbooks are on a public repository on GitHub, you can clone the repository with these commands:

```
# Install git if you don't have it already
sudo apt install git
# Clone the latest commit from the repository
git clone https://github.com/andresrcs/raspberry_pi_playbooks.git --depth 1
```

To configure the playbooks, you first need to define your “inventory”, this is a list of the hosts you are going to connect to, edit the `raspberry_pi_playbooks/inventory.ini` file and add the IP of your Raspberry Pi(s):

```
[raspberries] # This represents a group of Raspberries
raspberry_01 ansible_host=192.168.0.10 # This is the hostname and IP
for an individual Raspberry Pi
# You can add as many Raspberries as you need
```

Then set common variables for your `[raspberries]` group by editing the `raspberry_pi_playbooks/group_vars/raspberries.yml` file, the most important thing to check here is the path to your ssh key.

```
---
ansible_user: pi # This is the default user for Raspberry Pi OS
ansible_become_method: sudo
ansible_python_interpreter: /usr/bin/python3 # It also works with
```

```
python2 but is not recommended.
ansible_ssh_private_key_file: ~/.ssh/raspberrypi_key # The location of
the ssh key in your local PC.
```

Now, if you want to change the default installation settings, you can do it by editing the variables on the `raspberry_pi_playbooks/vars/config_vars.yml` file, although, the default options are fine for most use case scenarios, including reasonable security settings to use on real-world applications. The only things you most certainly want to change here are the email address for security notifications and the password for the PostgreSQL main user.

```
---
# System Configurations ##############################
##########################

# Swap parameters
swap_file_path: '/var/swap.1'
# Use any of the following suffixes
# c=1
# w=2
# b=512
# kB=1000
# K=1024
# MB=1000*1000
# M=1024*1024
# xM=M
# GB=1000*1000*1000
# G=1024*1024*1024
swap_file_size: '3GB'
swappiness: '10'

# Disable Wifi interface?
disable_wifi: true

# Security Configurations ##############################
########################

# Ports to open
exposed_ports:
  - { rule: 'allow', port: 22, proto: 'tcp' }   # ssh
  - { rule: 'allow', port: 22, proto: 'udp' }
  - { rule: 'allow', port: 80, proto: 'tcp' }   # http
  - { rule: 'allow', port: 5432, proto: 'tcp' }   # PostgreSQL
  - { rule: 'allow', port: 5432, proto: 'udp' }

# Notification email for fail2ban
send_email: true
fail2ban_email: your_email@something.com

# Password for the main PostgreSQL user
postgres_password: 'very_secure_password'

# Access rules for PostgreSQL
```

```
postgresql_rules:
  - { contype: local, users: all, address: samehost, method: trust }
  - { contype: local, users: postgres, address: samehost, method: trust
}
  - { contype: host, users: all, address: "{{
ansible_default_ipv4.network }}/24", method: trust }
  - { contype: host, users: all, address: 0.0.0.0/0, method: password }

# Main Software Versions ##############################
#########################

# R version to install
r_version: '4.0.3'

# shiny-server version to install
shiny_server_version: 'v1.5.16.958'

# rstudio-server version to install
rstudio_version: 'v1.3.1093'
```

Run the Playbooks

In theory, you could simply access the `raspberry_pi_playbooks` folder and run the `main.yml` playbook to install everything at once, but this would take a very, very, long time to complete so I think is not practical, the real use for this playbook is to verify the correct configuration of your server if you have made some changes afterward since the tasks are "idempotent", which means that can be applied multiple times without making changes beyond the initial application.

```
cd raspberry_pi_playbooks
ansible-playbook main.yml
```

To mitigate this issue and to make the installation process more flexible, I have divided the process into three individual playbooks:

```
cd raspberry_pi_playbooks
ansible-playbook install_basic_services.yml
ansible-playbook install_shiny_server.yml
ansible-playbook install_rstudio_server.yml
```

Each one still takes a long time to complete but it is a more reasonable waiting time (this is not entirely true, installing RStudio server may take up to 15 hours, depending on your specific setup), also, if you decide that you only need Shiny server or RStudio server but not the other, or you already have the support services you need to be installed, you can run only the playbooks you want.

If you want to update something in the future, like R, RStudio, or Shiny server (I can't guaranty that is going to work out of the box), you can simply change the version in the config file and run the specific part of the playbook by taking advantage of the defined "tags".

For example, this will only install the R version defined in the config file and nothing more:

```
cd raspberry_pi_playbooks
ansible-playbook install_basic_services.yml --tags "r"
```

The available tags are:

- `secure`: Set security settings on the server
- `swap`: Add swap memory to the server
- `nginx`: Install and configure Nginx + PHP
- `postgresql`: Install and configure PostgreSQL
- `r`: Install R from source
- `disable_wifi`: Disable Wifi and Bluetooth module
- `shiny-server`: Install shiny-server from source
- `configure_shiny`: Configure shiny-server
- `rstudio`: Install RStudio server from source
- `configure_rstudio`: Configure RStudio server

You can also go the other way and skip specific parts of the playbooks by using the `--skip-tags` option, for example, if you don't need PostgreSQL, you can avoid installing it by running the playbook this way:

```
cd raspberry_pi_playbooks
ansible-playbook install_basic_services.yml --skip-tags "postgresql"
```

After successfully running all the playbooks, you will have a fully functional installation ready to be used, so you could simply open an RStudio session at http://your_server_ip/rstudio and/or publish your Shiny apps in the `/srv/shiny-server` folder and access them at http://your_server_ip/shiny/your_app_name.

Final Notes

Even with the automation, installing natively is still going to take much longer than, for example, using containers, but since it is automatized, you can simply let it running and come back later, and you will enjoy the simpler manageability of a native installation.

Also, be aware that the installation process is going to push your Raspberry Pi to its limits so you can expect some random errors, but that doesn't mean it's not going to work, just restart your Pi, rerun the problematic playbook and see if it works the second time. I have done my best to make this functional in the long run by setting specific software versions but It is still possible that a system library update might break the playbooks, If you find any issues, please go to the GitHub repository for the playbooks and file an issue, or even better, if you know how to fix it, make a pull request.

As a final note, I want to point out that this approach can be further extended accordingly to your own needs, for example, you can implement automatic restoration of your server including deployed content. I have an additional playbook to restore the applications, services, and databases I normally run on my Raspberry Pi, so in the event of a catastrophic failure, I can get back to a working state in a short period of time and with little effort, compared with doing a manual restoration of the system.