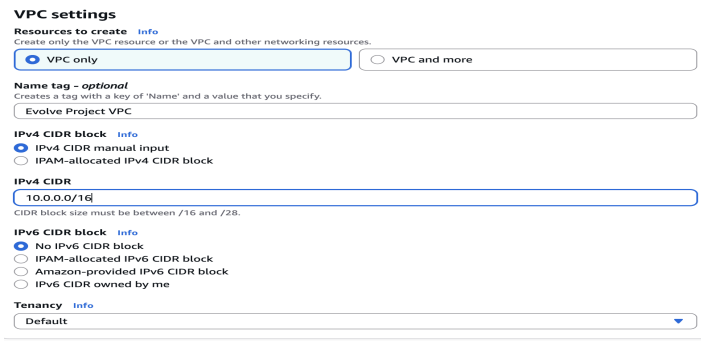
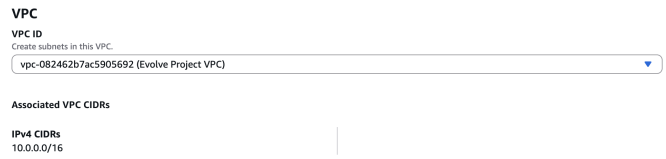
Setting Up VPC

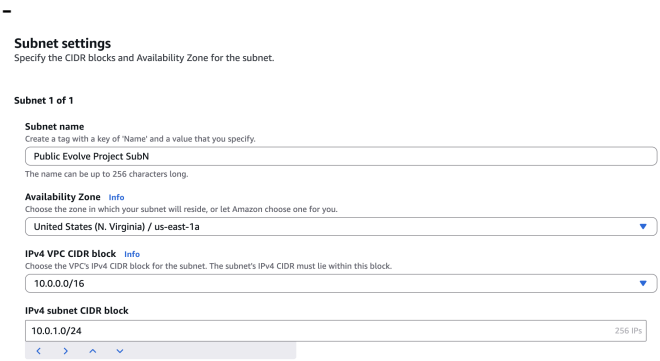
-



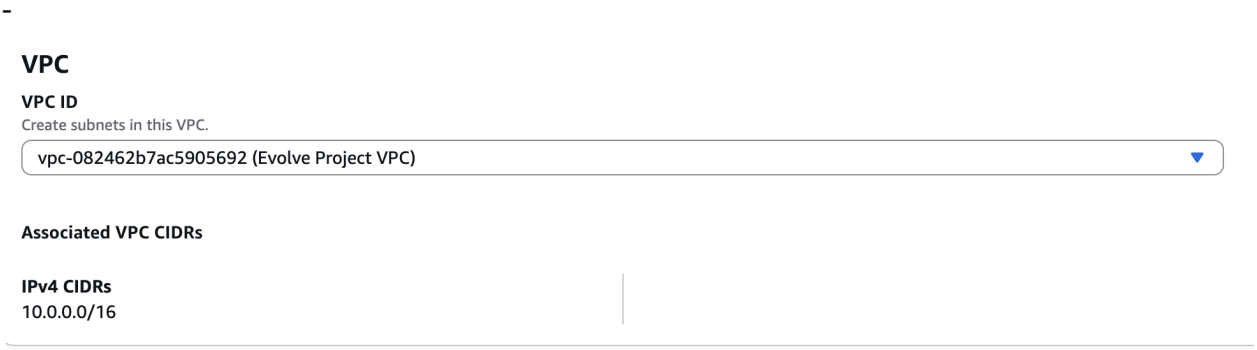
-



-



-



-

Subnet settings
Specify the CIDR blocks and Availability Zone for the subnet.

Subnet 1 of 1

Subnet name
Create a tag with a key of 'Name' and a value that you specify.
Private Evolve Project SubN
The name can be up to 256 characters long.

Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.
United States (N. Virginia) / us-east-1a

IPv4 VPC CIDR block Info
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.
10.0.0.0/16

IPv4 subnet CIDR block
10.0.2.0/24 256 IPs

-

Create internet gateway Info

An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.

Internet gateway settings

Name tag
Creates a tag with a key of 'Name' and a value that you specify.
Evolve Project IGW

-

Attach to VPC (igw-04e163f6ac12d4b31) Info

VPC
Attach an internet gateway to a VPC to enable the VPC to communicate with the internet. Specify the VPC to attach below.

Available VPCs
Attach the internet gateway to this VPC.
vpc-082462b7ac5905692

-

Create route table Info

A route table specifies how packets are forwarded between the subnets within your VPC, the internet, and your VPN connection.

Route table settings

Name - *optional*
Create a tag with a key of 'Name' and a value that you specify.
Evolve Project RT

VPC
The VPC to use for this route table.
vpc-082462b7ac5905692 (Evolve Project VPC)

-

Edit routes

Destination	Target	Status
10.0.0.0/16	local	Active
0.0.0.0/0	Internet Gateway: igw-04e163f6ac12d4b31	–

Add route

-

Edit subnet associations

Change which subnets are associated with this route table.

Available subnets (1/2)
Filter subnet associations

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR	Route table ID
Private Evolve Project SubN	subnet-0cf75e7c709f1c4e8	10.0.2.0/24	–	Main (rtb-047926c65df2bb6df)
Public Evolve Project SubN	subnet-0d58c8c6e51509986	10.0.1.0/24	–	Main (rtb-047926c65df2bb6df)

Selected subnets
subnet-0d58c8c6e51509986 / Public Evolve Project SubN

-

Edit subnet settings Info

Subnet

Subnet ID
subnet-0d58c8c6e51509986

Name
Public Evolve Project SubN

Auto-assign IP settings Info
Enable AWS to automatically assign a public IPv4 or IPv6 address to a new primary network interface for an instance in this subnet.

☑ Enable auto-assign public IPv4 address Info

☐ Enable auto-assign customer-owned IPv4 address Info
Option disabled because no customer owned pools found.

Step-by-Step VPC Setup

-Step 1: Create the VPC
I created a custom Virtual Private Cloud (VPC) in AWS named Evolve Project VPC using the IP address range 10.0.0.0/16. A VPC is like a private data center in the cloud — it gives you full control over your virtual network, including IP addresses, subnets, routing, and access control. The /16 CIDR block gives me a large address space (65,536 IPs) that I can divide into smaller subnets for different parts of my infrastructure, like web servers and databases.

-Step 2: Create Subnets
Inside my VPC, I created two subnets:

A public subnet named Public Evolve Project SubN with IP range 10.0.1.0/24 — this is where I will place the web server that needs internet access.

A private subnet named Private Evolve Project SubN with IP range 10.0.2.0/24 — this is where I will place the database server, which should not be accessible from the public internet.

Subnets are smaller pieces of the VPC that let you separate and control traffic between different server groups. Using separate subnets for public and private resources improves both organization and security.

-Step 3: Create and Attach an Internet Gateway
Next, I created an Internet Gateway (IGW) called Evolve Project IGW. An IGW is like a virtual router that allows resources inside a VPC (like my public web server) to send and receive data from the internet. After creating the IGW, I attached it to my VPC, which is necessary before any public subnet can use it. Without this step, even public subnets would be unable to reach the internet.

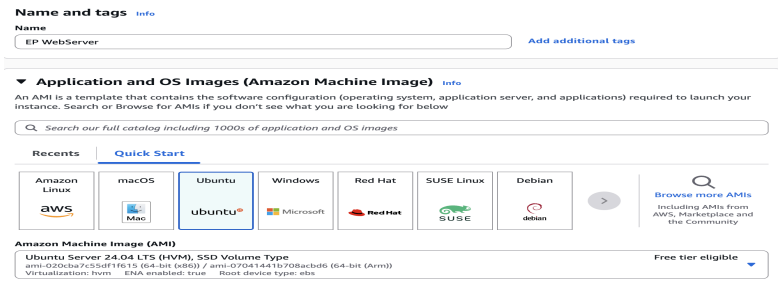
-Step 4: Create Route Tables
I then created a route table named Evolve Project RT to control how network traffic flows within the VPC. In this route table, I added a rule that sends all outbound traffic (0.0.0.0/0) to the Internet Gateway. This is what enables internet access for anything inside the public subnet.

After creating the route table and adding the route, I associated it with the public subnet only. This means that the web server can access the internet (and be accessed), but the private subnet remains isolated — which is ideal for protecting backend services like databases.

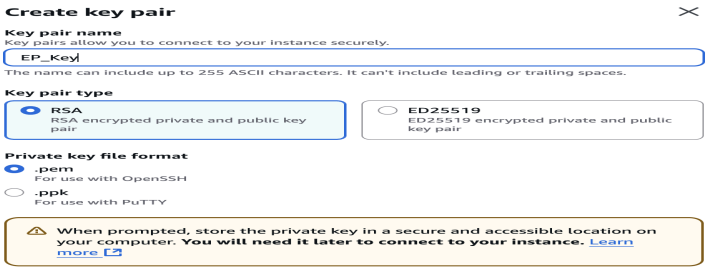
-Step 5: Enable Auto-Assign Public IP for Public Subnet
Finally, I updated the settings on the public subnet to automatically assign a public IP address to any virtual machine (EC2 instance) launched inside it. This is critical because without a public IP, the web server wouldn't be reachable from the internet — even if it's in a public subnet. This setting ensures that as soon as I launch the web server, it will have an internet-accessible IP address without needing manual setup.

EC2 Instance Setup & Launch

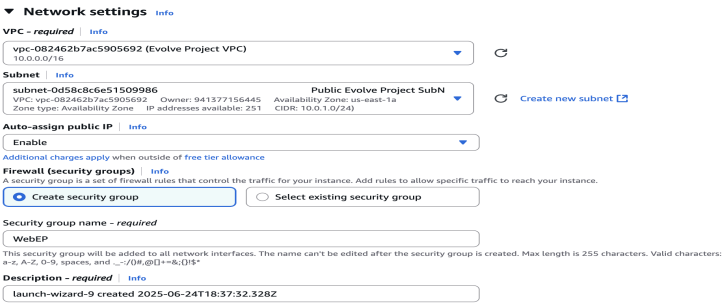
-



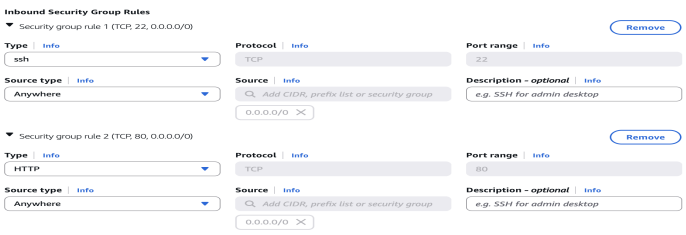
-



-



-



-

Name and tags Info

Name

EP DBServer

Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents | Quick Start

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Linux | Debian

Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type — Free tier eligible
ami-020cba7c55df1f615 (64-bit (x86)) / ami-07041441b708acbd6 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Instance type Info | Get advice

Instance type

t2.micro — Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand RHEL base pricing: 0.026 USD per Hour
On-Demand Linux base pricing: 0.0116 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

EP_Key

Create new key pair

Network settings Info

VPC - *required* Info

vpc-082462b7ac5905692 (Evolve Project VPC)
10.0.0.0/16

Subnet Info

subnet-0cf73e7c709f1c4e8 — Private Evolve Project SubN
VPC: vpc-082462b7ac5905692 Owner: 941377156445 Availability Zone: us-east-1a
Zone type: Availability Zone IP addresses available: 251 CIDR: 10.0.2.0/24)

Create new subnet

Auto-assign public IP Info

Disable

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group | Select existing security group

Security group name - *required*

DBEP

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and _-:/()#,@[]+=&;{}!$*

Description - *required* Info

launch-wizard-9 created 2025-06-24T18:44:27.198Z

Inbound Security Group Rules

Security group rule 1 (TCP, 22, 0.0.0.0/0) — Remove

Type	Protocol	Port range
ssh	TCP	22

Source type	Source	Description - *optional*
Anywhere	Add CIDR, prefix list or security group — 0.0.0.0/0	e.g. SSH for admin desktop

Security group rule 2 (ICMP, All, 0.0.0.0/0) — Remove

Type	Protocol	Port range
All ICMP - IPv4	ICMP	All

Source type	Source	Description - *optional*
Anywhere	Add CIDR, prefix list or security group — 0.0.0.0/0	e.g. SSH for admin desktop

Step-byStep EC2 Setup & Launch

Instance 1: Public Ubuntu Web Server

-Step 1: Access the EC2 Dashboard

I opened the EC2 (Elastic Compute Cloud) dashboard in AWS. EC2 is Amazon's virtual server service — it lets you launch and run cloud-based servers. From here, I clicked "Launch Instance" to start creating a new virtual machine (VM).

-Step 2: Launch an Instance

I initiated the process of launching a new EC2 instance, which involved choosing a name, operating system, and machine type. This instance will act as the public-facing web server in my project.

-Step 3: Configure Instance Details

Name: EP WebServer
AMI: Ubuntu 18.04 LTS
Type: t2.micro
Key Pair: Created (EP_Key)

I named this instance EP WebServer and selected the Ubuntu 18.04 LTS image, which is a lightweight, stable version of the Linux operating system. I used the free-tier t2.micro instance type, which is ideal for small-scale environments like this one. I also created a key pair, which acts like a private "digital key" that allows me to connect securely to this server via SSH (Secure Shell).

-Step 4: Set Network and Subnet

VPC: Evolve Project VPC
Subnet: Public Evolve Project SubN
Auto-assign Public IP: Enabled

I placed this web server in the Public Evolve Project SubN, which is connected to the internet through an Internet Gateway. I enabled automatic assignment of a public IP address so the instance can be accessed from outside AWS — this is necessary for me to connect to the server and for the public to access the website it will host.

-Step 5: Configure Firewall (Security Group)

Name: WebEP
Rules Added:
SSH (Port 22) from Anywhere
HTTP (Port 80) from Anywhere

I created a security group called WebEP, which acts like a firewall for this instance. I allowed SSH access (port 22) only from Anywhere, so I can connect to the server. I also allowed HTTP traffic (port 80) from anywhere, since this is a public web server that needs to serve webpages to anyone on the internet.

-Step 6: Launch Instance
After reviewing all the settings, I launched the instance. AWS then allocated resources, assigned it an IP address, and placed it into the VPC and subnet I configured. The instance status changed to “running,” meaning the virtual server is now active and ready for SSH login or web hosting configuration.

Instance 2: Private Ubuntu Database Server
-Step 1: Launch Second EC2 Instance
I began creating a second EC2 instance. This one will serve as the backend database server, which should not be accessible from the public internet. It will only accept connections from the web server inside the VPC.

-Step 2: Configure Instance Details
Name: EP DBServer
AMI: Ubuntu 18.04 LTS
Type: t2.micro
Key Pair: Same as Web Server (EP_Key)

I named this instance EP DBServer and selected the same Ubuntu operating system and instance type as the web server. Using the same key pair allows me to SSH into both servers from my local machine, although the DB server won’t have public access.

-Step 3: Set Network and Subnet
VPC: Evolve Project VPC
Subnet: Private Evolve Project SubN
Auto-assign Public IP: Disabled (default)

I placed the database server into the Private-DB-Subnet, which is not connected to the internet. This makes it a secure backend server that can only be accessed from within the VPC. Since it doesn’t need to talk to the internet directly, I left the public IP disabled, which keeps it completely hidden from external networks.

-Step 4: Configure Security Group Rules
Name: DBEP
Rules Added:
SSH (Port22) from Anywhere
ICMP (Echo Request, Ping) from anywhere

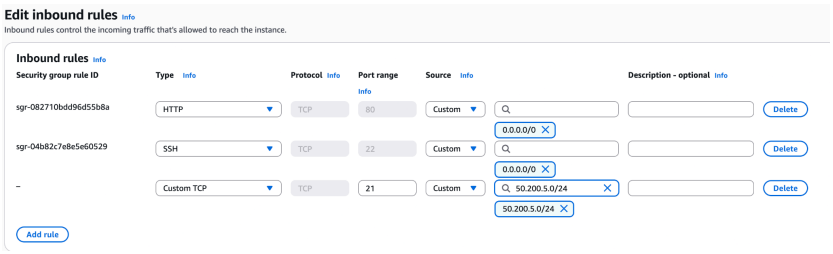
I created a Security Group named DBSG for the database server to define what kinds of network traffic are allowed into the instance. Even though this server is in a private subnet and not reachable from the internet directly, AWS still uses these firewall rules to control communication from within the VPC.

I added two inbound rules to DBSG:

SSH (Port 22) from Anywhere — This allows me to connect to the database server using SSH. While this would normally be restricted for security reasons, it was temporarily opened to allow initial configuration. Later, I'll update it to only allow SSH access from the Web Server.

ICMP (Ping) from Anywhere — This enables basic network testing using the ping command. ICMP Echo Request is a common tool used to check if the instance is reachable within the network. Like SSH, this will later be tightened to only allow ping from the Web Server's IP or security group.

-Step 5: Launch Instance
After configuring the instance details, network settings, and security group rules, I launched the EP DBServer instance. AWS automatically provisioned virtual hardware, assigned a private IP address, and placed the instance into the private subnet of my custom VPC. Since it's in a private subnet with no public IP, it's completely isolated from the internet — exactly as intended. The instance is now running and ready to be connected to from within the VPC, specifically by the public web server for backend database communication.
-

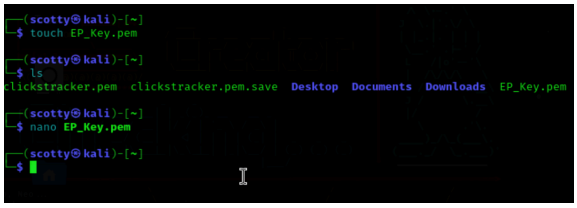


-Step 6: Corrected Security Group to Add FTP Access
After reviewing the traffic configuration requirements, I realized that I had not originally included the FTP rule for the Web Server. To fix this, I returned to the AWS EC2 dashboard, selected the Web Server's security group (WebEP), and edited the inbound rules.

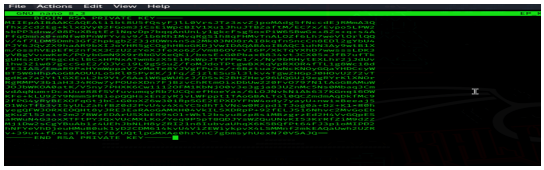
I added a new rule using Custom TCP, specifying port 21, with a source of 50.200.5.0/24. This allows FTP traffic from a specific external network range, as required by the project. This change ensures that trusted external users can access the server using the FTP protocol for file transfer, while still restricting access from the general public.

LAMP Stack

-



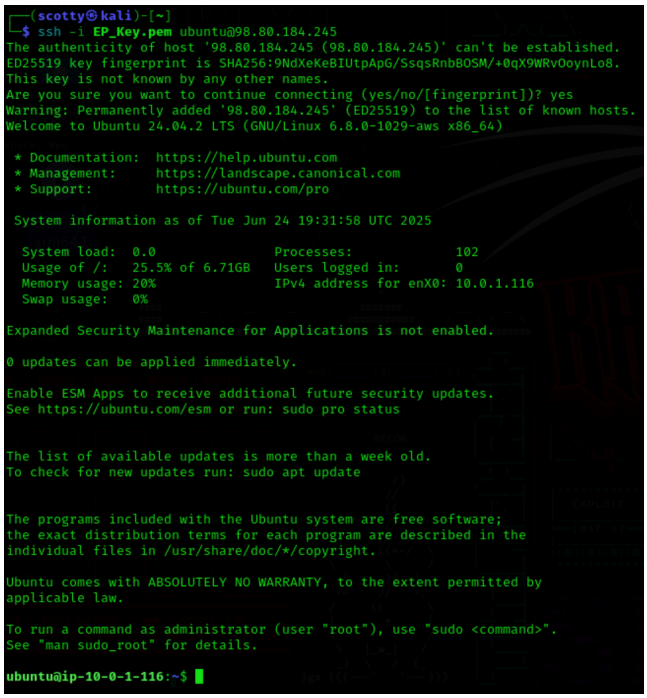
-



-


(scotty㉿kali)-[~]
$ chmod 400 EP_Key.pem

-


(scotty㉿kali)-[~]
$ ssh -i EP_Key.pem ubuntu@98.80.184.245
The authenticity of host '98.80.184.245 (98.80.184.245)' can't be established.
ED25519 key fingerprint is SHA256:9NdXeKeBIUtpApG/SsqsRnbBOSM/+0qX9WRvOoynLo8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '98.80.184.245' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1029-aws x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/pro
System information as of Tue Jun 24 19:31:58 UTC 2025
System load: 0.0 Processes: 102
Usage of /: 25.5% of 6.71GB Users logged in: 0
Memory usage: 20% IPv4 address for enX0: 10.0.1.116
Swap usage: 0%
Expanded Security Maintenance for Applications is not enabled.
0 updates can be applied immediately.
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
The list of available updates is more than a week old.
To check for new updates run: sudo apt update
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
ubuntu@ip-10-0-1-116:~$

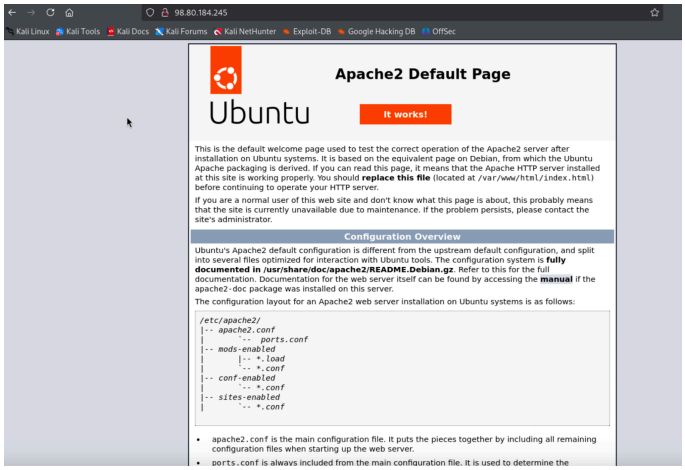
-


ubuntu@ip-10-0-1-116:~$ sudo apt update

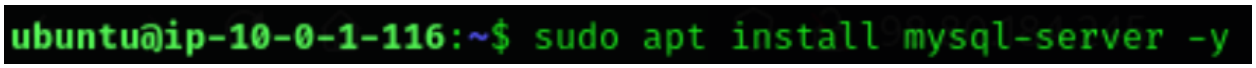
-


ubuntu@ip-10-0-1-116:~$ sudo apt install apache2 -y

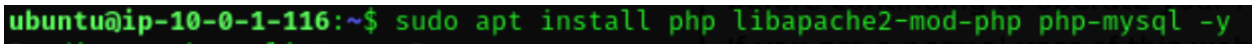
-



-

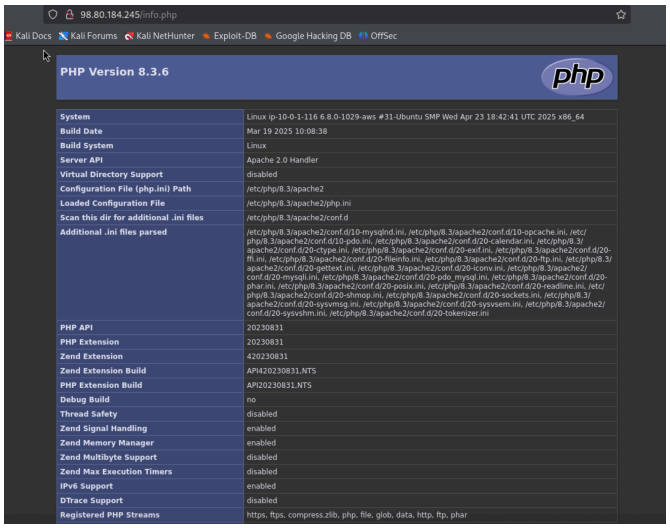


-



-

-



LAMP Ste-by-Step Setup

-Step 1: Create and Save the Private Key File

So first i created a key file using “touch” calling the file “EP_Key.pem” in my personal Kali

Then I nano’d into the EP_Key.pem file and added my key pair

-Step 2: Secure the Key File
Command: chmod 400 mykey.pem
Description: Restricted the key file's permissions to read-only for the owner, which is required by SSH for security.

-Step 3: Connect to the EC2 Instance Using SSH
Command: ssh -i EP_Key.pem ubuntu@90.80.184.245
Description: Initiated a secure connection to the EC2 server using the private key and default Ubuntu username.

-Step 4: Update the Package Index
Command: sudo apt update
Description: Refreshed the list of available packages on the server.

-Step 5: Install Apache Web Server
Command: sudo apt install apache2 -y
Description: Installed Apache to serve web pages on the EC2 instance.

-Step 6: Confirm Apache is Running
Searched: http//:90.80.184.245/

-Step 7: Install MySQL Server
Command: sudo apt install mysql-server -y
Description: Installed the MySQL database server, required for dynamic websites and applications.

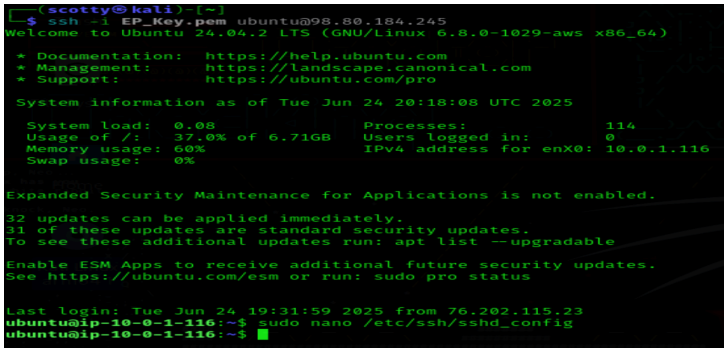
-Step 9: Install PHP and Modules
Command: sudo apt install php libapache2-mod-php php-mysql -y
Description: Installed PHP and required modules to allow Apache to process PHP files and connect to MySQL.

-Step 10: Create a PHP Info Page
Command: sudo nano /var/www/html/info.php
Content: <?php phpinfo(); ?>
Description: Created a simple PHP page to confirm PHP is working correctly on the server.

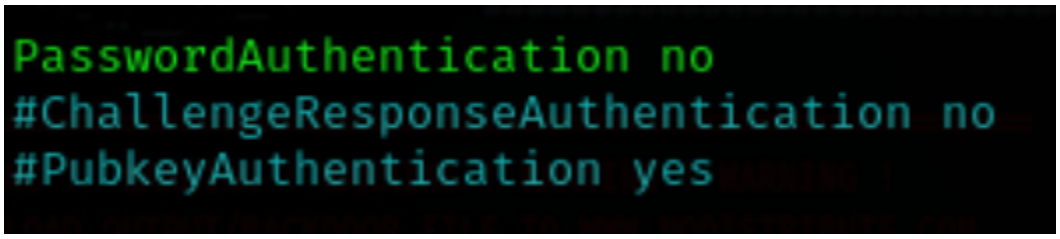
-Step 11: Test the Page in Browser
Searched: http://<your-ec2-public-ip>/info.php
Description: Opened the PHP test page in a browser. Seeing the “It works!” or PHP info screen confirms successful configuration.

SSH Configuration

-


(scotty㉿kali)-[~]
$ ssh -i EP_Key.pem ubuntu@98.80.184.245
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1029-aws x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/pro
System information as of Tue Jun 24 20:18:08 UTC 2025
System load: 0.08 Processes: 114
Usage of /: 37.0% of 6.71GB Users logged in: 0
Memory usage: 60% IPv4 address for enX0: 10.0.1.116
Swap usage: 0%
Expanded Security Maintenance for Applications is not enabled.
32 updates can be applied immediately.
31 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status
Last login: Tue Jun 24 19:31:59 2025 from 76.202.115.23
ubuntu@ip-10-0-1-116:~$ sudo nano /etc/ssh/sshd_config
ubuntu@ip-10-0-1-116:~$

-


PasswordAuthentication no
#ChallengeResponseAuthentication no
#PubkeyAuthentication yes

-


ubuntu@ip-10-0-1-116:~$ sudo systemctl restart ssh

-


ubuntu@ip-10-0-1-116:~$ sudo nano /etc/ssh/sshd_config

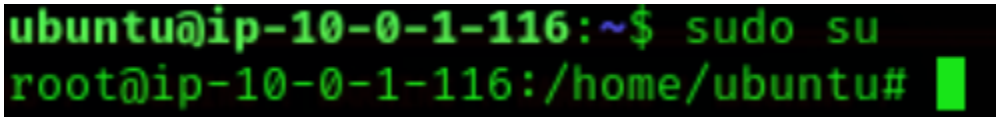
-


#PermitRootLogin yes

-


ubuntu@ip-10-0-1-116:~$ sudo systemctl restart ssh

-


ubuntu@ip-10-0-1-116:~$ sudo su
root@ip-10-0-1-116:/home/ubuntu#

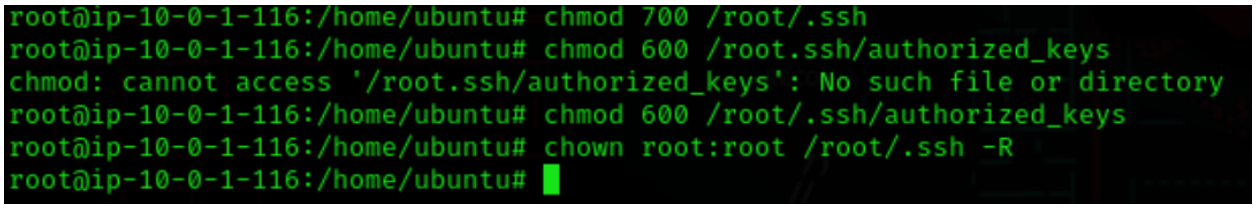
-


root@ip-10-0-1-116:/home/ubuntu# mkdir -p /root/.ssh

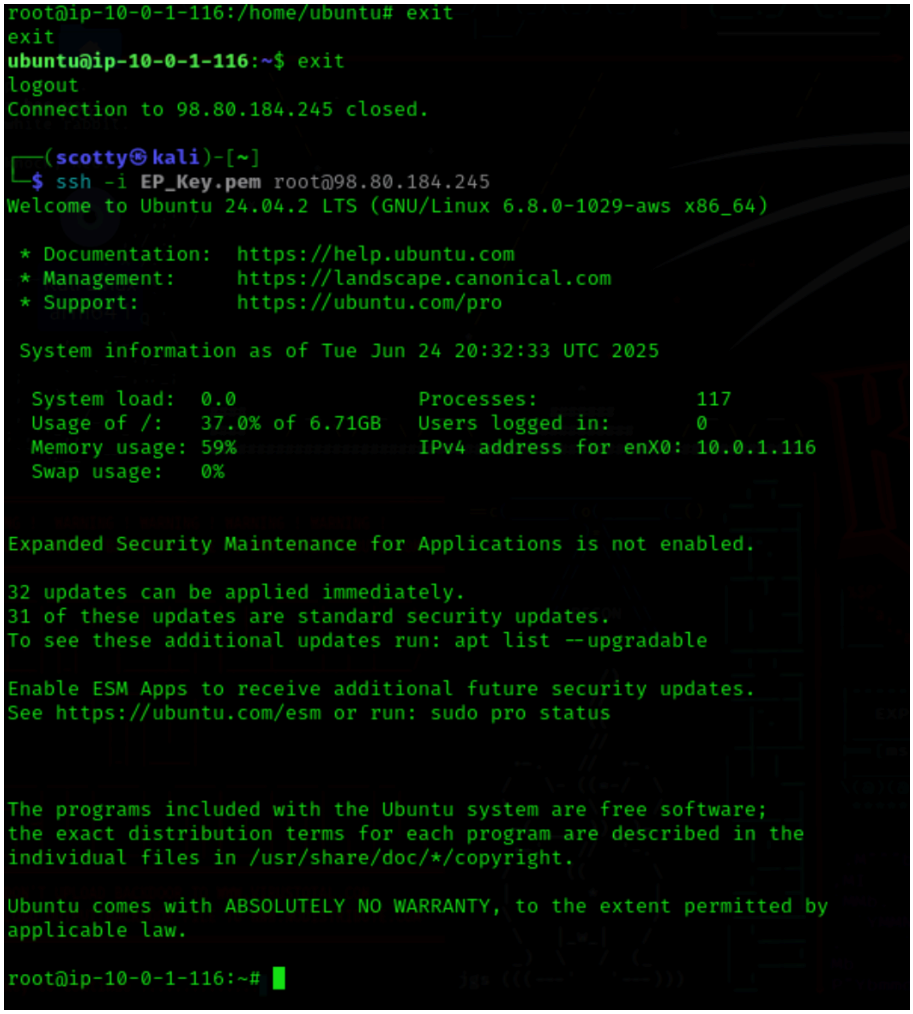
-


root@ip-10-0-1-116:/home/ubuntu# cp /home/ubuntu/.ssh/authorized_keys /root/.ssh/

-


root@ip-10-0-1-116:/home/ubuntu# chmod 700 /root/.ssh
root@ip-10-0-1-116:/home/ubuntu# chmod 600 /root.ssh/authorized_keys
chmod: cannot access '/root.ssh/authorized_keys': No such file or directory
root@ip-10-0-1-116:/home/ubuntu# chmod 600 /root/.ssh/authorized_keys
root@ip-10-0-1-116:/home/ubuntu# chown root:root /root/.ssh -R
root@ip-10-0-1-116:/home/ubuntu#

-



SSH Configuration Step-by-Step Setup

-1. Enabled Public/Private Key Authentication
Instead of logging in with a password (which could be guessed), I configured the server to only accept SSH keys — a highly secure method that requires a unique private file to connect. I edited the server's SSH configuration file to disable passwords and require a key instead. This ensures only someone with the correct private key can log in.

-2. Disabled Password Logins Over SSH
I made sure passwords can no longer be used to log into the server. This protects against brute-force and dictionary attacks, where attackers try hundreds or thousands of possible passwords to gain access. This setting forces the system to only accept secure key-based access.

-3. Allowed Root Login Over SSH
By default, the "root" user — the account with full control over the server — is blocked from logging in remotely. I changed the configuration to permit direct root login over SSH so I could

perform advanced administrative tasks. This step is necessary for the next tasks that require root access.

-4. Copied My Public SSH Key to the Root Account
To allow the root user to log in using my SSH key, I had to place that key into the root user's trusted key file. I did this by copying the existing authorized key file from the regular user to the root's .ssh folder. I also set the correct permissions to prevent unauthorized access.

-5. Logged In as Root to Confirm
Finally, I tested the changes by securely connecting to the server as the root user using my SSH key. This confirmed that all settings were applied correctly and that I now have full secure access as root.