Sure! Here's a **simple explanation** of what Andrew Mallett is demonstrating in this part of the course:

**🛠️ What’s being done here?**

You're learning how to **set up an SSH server** on one virtual machine (ubuntu2) and use another machine (ubuntu1) to **connect to it remotely** using SSH.

This is a hands-on guide where:

* ubuntu2 is the **server** (where you’ll configure SSH)
* ubuntu1 is the **client** (where you’ll connect from)

**🧑‍💻 What are the steps being shown?**

1. **Start both machines** – ubuntu1 (client) and ubuntu2 (server) must be running.
2. **Create a new user** on ubuntu2:
   * Command: sudo useradd -m -s /bin/bash tux
   * This creates a user named tux with a home directory and bash shell.
3. **Set a password** for the user:
   * Command: sudo passwd tux

**🔐 Check and configure SSH settings**

Now on ubuntu2, you're configuring the **SSH server** so it behaves the way you want.

**Check the current SSH config:**

* Command: sudo sshd -T | grep -E '(password|permit)'
  + This shows current settings for:
    - PasswordAuthentication – allows logging in with a password
    - PermitRootLogin – controls whether root can log in via SSH

**What do we want?**

* Allow password logins for now (PasswordAuthentication yes)
* **Block root login** completely (PermitRootLogin no), for better security

**Edit the SSH config file:**

* File: /etc/ssh/sshd\_config
* Use a text editor to:
  + Make sure PasswordAuthentication yes is set (so you can log in as tux)
  + Change PermitRootLogin to no (to stop root logins)

**Restart SSH to apply changes:**

* Command: sudo systemctl restart sshd

Now your SSH server (ubuntu2) is configured safely to:

* Let user tux log in with a password
* **Not** allow root logins at all

**✅ What’s next?**

You’ll switch to ubuntu1 (client) to test connecting to ubuntu2 using SSH and eventually set up **key-based authentication** (so you won’t need passwords anymore).

**💡 In simple terms:**

You're getting your ubuntu2 system ready so someone (you!) can **remotely log in** using SSH as a safe, normal user (tux) — **not** as root, and with password login allowed just for now. Later, you'll improve security further by using SSH keys.

Let me know if you'd like help setting up or testing the SSH connection from ubuntu1!

**🧠 First, What is DNS?**

DNS stands for **Domain Name System**.

Think of DNS like the **contacts app** on your phone. Instead of remembering phone numbers, you save names like "Mom" or "Pizza Hut." DNS does the same for the internet:

* You type google.com
* DNS finds its IP address (like 142.250.180.78)
* Your computer uses the IP address to talk to Google's servers

**📘 What Are We Learning in This Lesson?**

You're learning how to:

1. **Understand how DNS works on Ubuntu 20.04**
2. **Turn your Ubuntu machine into a DNS server**
3. **Make another Ubuntu machine use your DNS server**
4. **Check DNS settings**
5. **Change which DNS servers Ubuntu uses**

**🖥️ Ubuntu’s Default DNS Behavior**

Every Ubuntu 20.04 system already has a small **built-in DNS server** using a service called systemd-resolved.

**Here's what it does:**

* Listens on 127.0.0.53 (that’s your local machine only)
* Uses **port 53**, which is the default DNS port
* Helps your system resolve names like ubuntu.com to IP addresses

**Files & Tools involved:**

* /etc/resolv.conf: This file shows which DNS server your computer is using
* resolvectl: A command that shows detailed DNS info (like IPs, DNS servers used, etc.)
* dig: A tool you can install and use to test DNS lookups

**🔧 Goal: Make Ubuntu2 a Real DNS Server**

**You have 2 machines:**

* **Ubuntu1** = DNS client
* **Ubuntu2** = DNS server (it gives answers to Ubuntu1)

We’ll install a DNS server software called **BIND9** on Ubuntu2.

**⚙️ Step-by-Step Process (Simplified)**

**✅ Step 1: Understand What You Already Have**

On Ubuntu, run this command to check what DNS is doing:

bash

CopyEdit

resolvectl status

You'll see:

* Which DNS servers you're using
* Which interfaces (like eth0) are using them

You can also check:

bash

CopyEdit

cat /etc/resolv.conf

⚠️ This file is a *symlink*, meaning it points to another file. Ubuntu uses it to store current DNS settings.

**✅ Step 2: Test DNS Resolution**

Use the dig command like this:

bash

CopyEdit

dig google.com

It shows:

* Which DNS server was used
* How fast the response was
* The IP address of google.com

**🎯 End Goal: What are we trying to do?**

We want to:

✅ **Set up a secure DNS server** on **Ubuntu 2**,  
✅ So that only **trusted machines** (like Ubuntu 1 and localhost) can ask it DNS questions (queries),  
✅ And **block everyone else** from using our DNS server.

**🧠 Think of it like this:**

You’re setting up a **phone directory** (DNS server) in your house (Ubuntu 2).

* You want your **family** (Ubuntu 1) to be able to look up phone numbers.
* But you **don’t want strangers** knocking on your door asking for phone numbers.

So, you create a **guest list** of people who are allowed to ask questions.

That guest list is called an **ACL (Access Control List)** — a list of IP addresses that are **trusted**.

Of course! 😊 Here's a **very simple explanation** of what you just read — summarizing the **end of your DNS module**:

**🧠 What Was This Module About?**

You’ve just learned **how DNS (Domain Name System)** works on Linux, and how to **set up your own DNS server**.

**✅ Here’s What You Learned in Simple Steps:**

**1. Systemd-resolved is your system’s default DNS service**

* Ubuntu already has a built-in DNS tool called systemd-resolved.
* You can **check it with:**
* resolvectl status

**2. You tested DNS lookups with dig**

* This command checks how names are being resolved (turned into IP addresses).
* If you only want the IP and not the full output:
* dig example.com +short

**3. You learned about /etc/resolv.conf**

* This file tells your system **which DNS servers to use**.
* But nowadays, it's often **just a shortcut (symbolic link)** to another file managed by systemd or Netplan.

**4. You installed your own DNS server (BIND9)**

* bind9 is a real DNS server, like what companies use.
* You learned to:
  + Install it
  + Configure it in /etc/bind/named.conf
  + Check its status with:
  + sudo systemctl status bind9

**5. You created your own DNS zones**

* A **zone** is like your **own little DNS database** (for example, .local domain).
* You created:
  + A **forward zone** (name → IP)
  + A **reverse zone** (IP → name)

**6. You did reverse lookups**

* Using this command:
* dig -x 192.168.56.101
* This tells you which hostname matches a given IP.

**🧾 Big Picture Summary:**

| **You did this** | **Why it's useful** |
| --- | --- |
| Used built-in DNS | Good for learning how Ubuntu handles DNS |
| Used dig | To test name resolution |
| Changed /etc/resolv.conf | To control which DNS your system uses |
| Installed BIND9 | So you have your **own DNS server** |
| Created zones | So you can **name your own computers** on your local network |
| Tested everything | To make sure it works! |

**🎉 What’s Next?**

You’re now ready to **use your DNS server** — for example, when setting up an **Apache web server**, which is coming in the next module.

That means you'll be able to:

* Host websites like ubuntu2.local
* Use DNS instead of typing IPs

Let me know if you want a **visual diagram** or a **cheat sheet** to summarize all of this. You're doing great! 🚀

**What is a DNS record and why do you need it?**

When you install Apache on your server (like **ubuntu2**), your web server has an **IP address** — for example: 192.168.1.100.

But IP addresses are hard to remember, so we use **domain names** (like www.local or myserver.local) to access the server.

**DNS records** link these domain names to IP addresses or other names.

**Common DNS records you’ll use for Apache:**

* **A record:** Links a domain name directly to an IP address.  
  Example:  
  ubuntu2.local → 192.168.1.100
* **CNAME record:** An alias that points one name to another name.  
  Example:  
  www.local → ubuntu2.local

**Example setup step-by-step:**

**1. You have Apache running on ubuntu2 with IP 192.168.1.100.**

**2. Set an A record in your DNS zone file:**

In your DNS server config (usually a file like /etc/bind/zones/db.local), add:

css

CopyEdit

ubuntu2.local. IN A 192.168.1.100

This means ubuntu2.local points to your server’s IP.

**3. Set a CNAME record so you can use www.local as an alias:**

In the same file, add:

lua

CopyEdit

www.local. IN CNAME ubuntu2.local.

This means www.local is just another name for ubuntu2.local.

**4. Update the serial number in the DNS zone file (important for changes to take effect):**

Example:

pgsql

CopyEdit

@ IN SOA ns.local. admin.local. (

4 ; serial number — increase this number each time you change the file

3600 ; refresh

1800 ; retry

604800 ; expire

86400 ; minimum TTL

)

Make sure to **increase the serial number** each time you edit the file (e.g., from 3 to 4).

**5. Check your DNS zone for errors:**

bash

CopyEdit

sudo named-checkzone local /etc/bind/zones/db.local

**6. Reload your DNS server:**

bash

CopyEdit

sudo rndc reload local.

This tells DNS to update with your new records.