How to: Proxmox

What is Proxmox?

Introduction

Virtualization is a technology that allows you to create and run multiple "virtual" computers inside a single physical computer. Instead of needing multiple separate machines, virtualization lets you run multiple operating systems (like Windows, Linux, or macOS) on the same hardware at the same time. At its core, virtualization uses special software called a hypervisor to create and manage virtual machines (VMs). The hypervisor acts as a middleman between the physical hardware and the virtual machines, making sure each VM gets the resources (CPU, RAM, storage, etc.) it needs. □ ‹ □ •□ •□ ›ა

A Proxmox server is a powerful open-source Type 1 hypervisor (also known as "bare-metal" hypervisor) that allows you to run multiple virtual machines (VMs) and containers on a single physical server. It combines two main technologies: KVM (Kernel-based Virtual Machine) for full virtualization and LXC (Linux Containers) for lightweight container-based virtualization. Proxmox is widely used for enterprise-level virtualization, home labs, and small to medium-sized businesses looking for an efficient and cost-effective way to manage virtualized workloads.

One of the awesome things about Proxmox is its ability to cluster (ᒧ✧▽✧)ᒧ. Clustering is the process of grouping multiple independent servers (or nodes) into a single logical unit. This allows you to manage them together as if they were a single entity, making tasks like resource allocation, failover, and load balancing much easier. In Proxmox, clustering involves connecting multiple Proxmox Virtual Environment (PVE) nodes together, enabling them to communicate and share resources. Proxmox uses its PVE Cluster feature to integrate and synchronize nodes into one unified management interface.

Minimum Requirements:

- **CPU:** 64-bit processor with virtualization support
- **RAM:** 2 GB (absolute minimum; recommended is at least 8 GB)
- **Storage:** 16 GB+ boot drive (SSD recommended for performance)
- **Network:** 1 Gbps network adapter
- **OS:** Proxmox VE is installed as a bare-metal hypervisor (based on Debian Linux)

Capabilities of Proxmox VE

- Virtualization Options
- Web-Based Management Interface
- Storage Flexibility
- Networking Features
- High Availability and Clustering

- Backup and Disaster Recovery

Last Updated: 2/20/2025
BY: Lilian

1.0 Installation

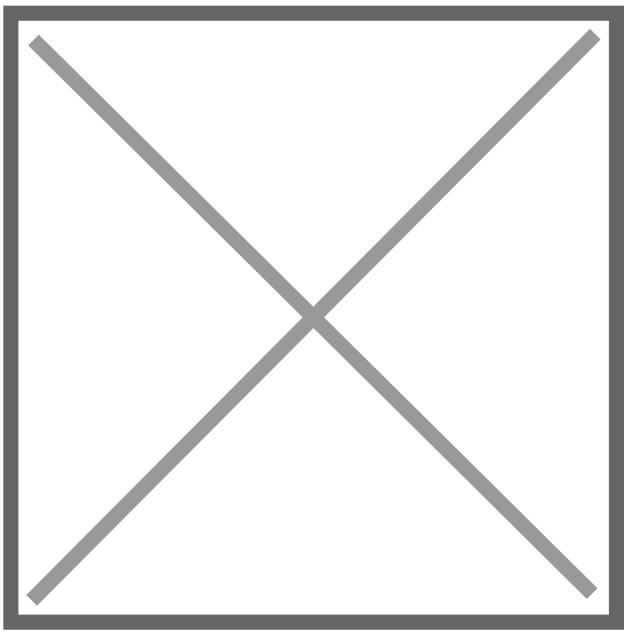
Materials / Pre-Setup:

- USB Flash drive
- Machine with minimum requirements
  - *To make a Proxmox Cluster - 2 minimum, but 3 machines are preferred (for fault tolerance and high availability)*
- Understand the schema of your network *(i.e. IP subnet, default gateway, etc.)*

Installing Proxmox

1. Download Proxmox Virtual Environment onto any system that you're working with
2. Create a bootable USB download using a USB Creation Tool (multiple approaches stated below)
   - Rufus (Windows)
   - Ventoy (Windows, macOS, Linux)
   - UNetbootin (Windows, macOS, Linux)

EXAMPLE USING RUFUS:



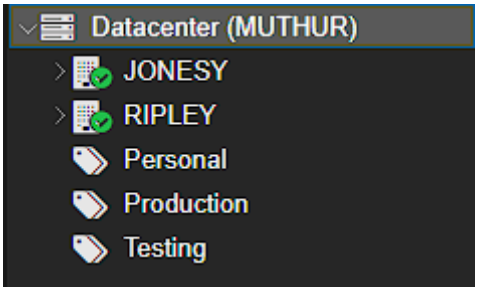
3. Insert the newly created USB bootable into the machine that will act as a designated Proxmox server and proceed with installation as you would any other OS installation (may have to change BIOS settings to boot to USB first)

Installer Configurations

Once you get to the Proxmox Installer, you'll be prompted with multiple things - examples/suggestions are given below:

- Choose "Install Proxmox VE (Terminal UI)"
- Hostname (FQDN): [PVE NAME].local *(use different names if you're trying to cluster - doesn't matter what you name it)*
- IP Address (CIDR): [IP ADDRESS CIDR] *(e.g. 10.0.0.100/16)*
- Gateway Address: [DEFAULT GATEWAY] *(e.g. 10.0.0.1)*
- DNS Server Address: [DNS SERVER] *(i.e. 10.0.0.1 or public DNS servers: Cloudflare - 8.8.8.8, Google - 1.1.1.1)*

EXAMPLE OF WHAT IT WILL LOOK LIKE AFTER CONFIGURATIONS:



In this Proxmox example, the node names follows a theme from the Alien franchise

Last Updated: 2/22/2025
Contributors: Lilian, Vivian

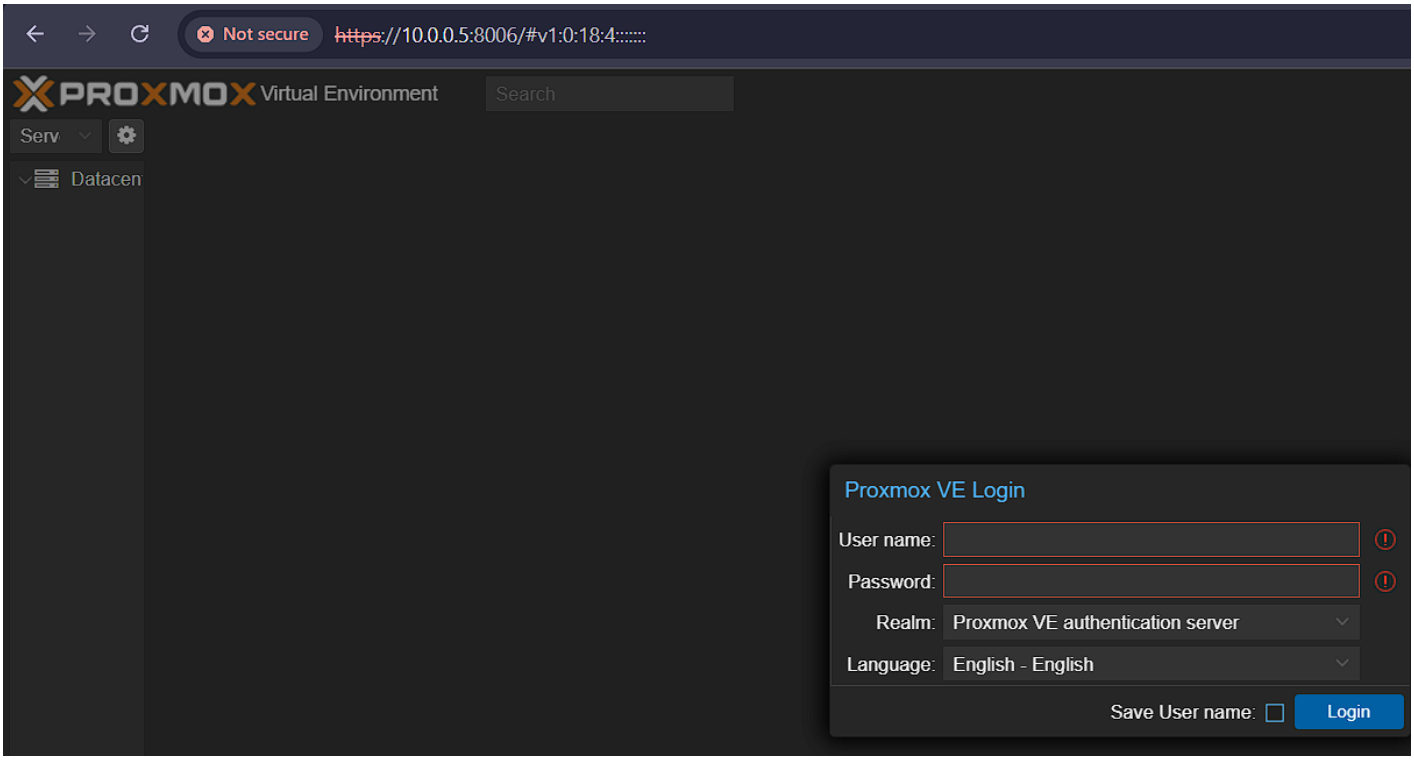
2.0 Proxmox Users and Groups

Adding New Users

1. Access the Proxmox GUI by entering the static IP you gave it into a web browser appending :8006 (the port) to it as well
   - The default login for Proxmox is: root
   - The password is what you set during installation

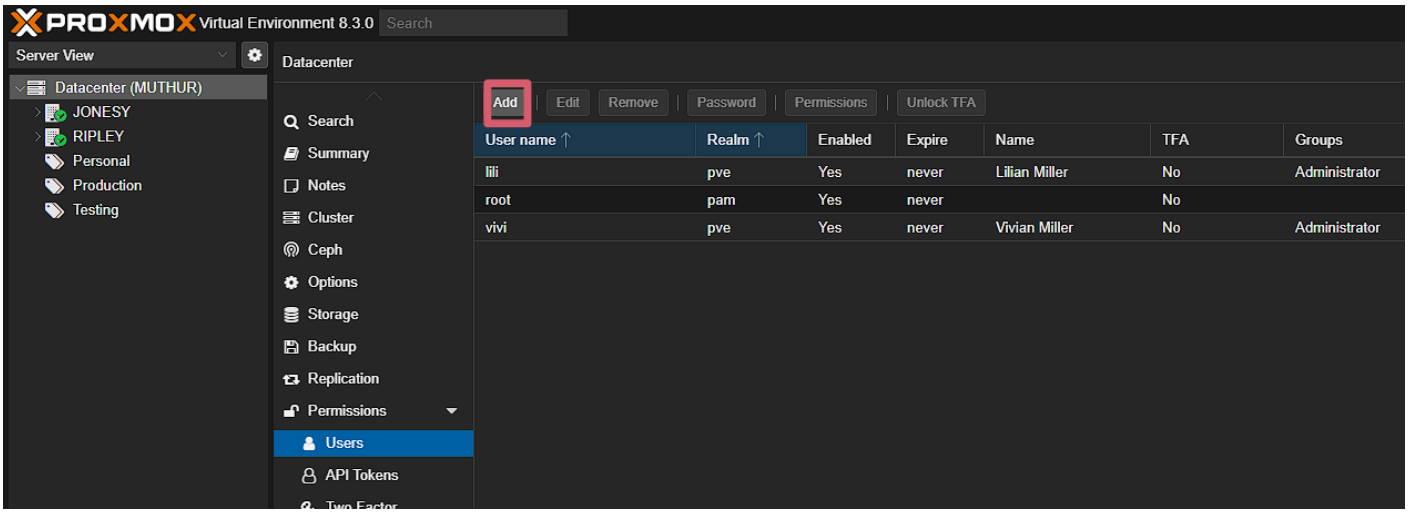
   > **For security reasons, change/disable the root account and remove it from the Administrator group after you've made a user account**

EXAMPLE OF THE STATIC IP AS 10.0.0.5:8006 AND CORRECTLY GIVING THE PROXMOX LOGIN PAGE:



2. Click on Datacenter > Users > Add
   - Allow user to set their own username and password
   - Appoint user into appropriate group

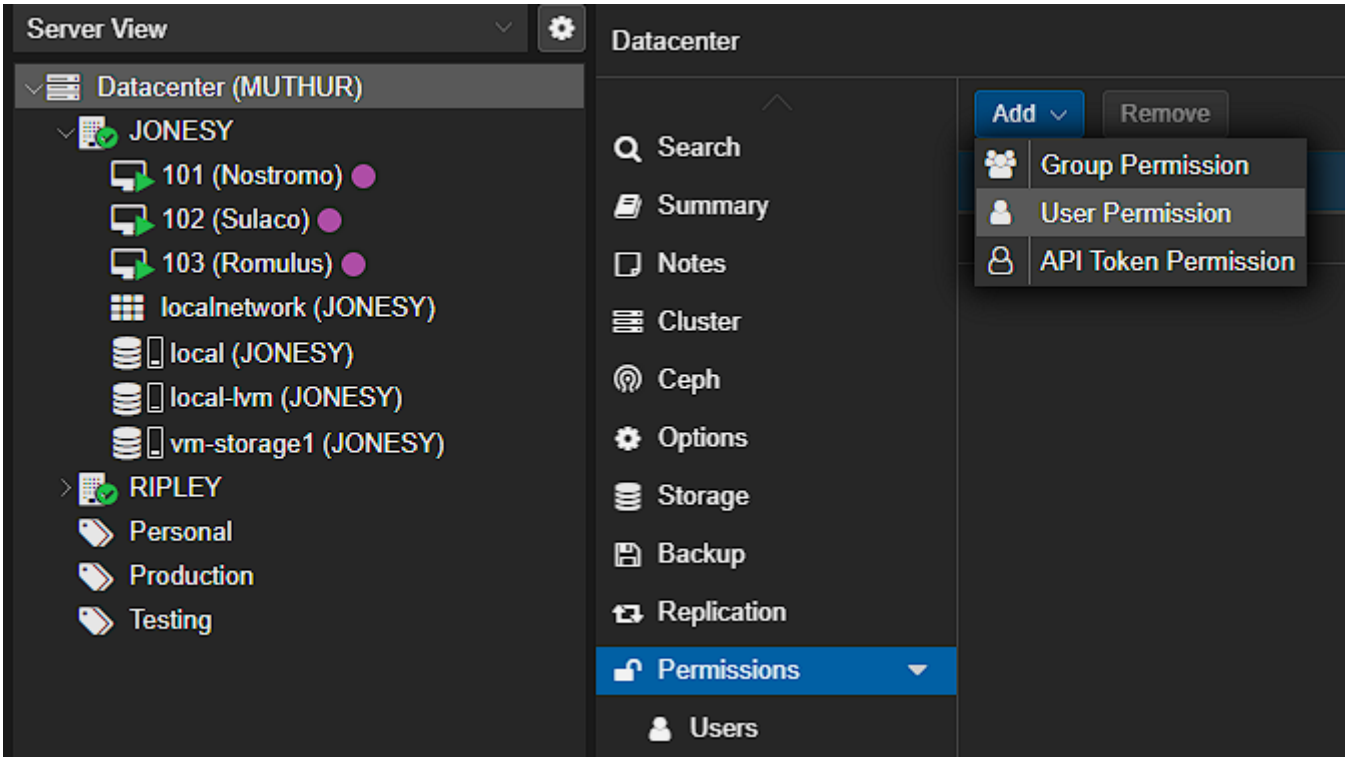
IMAGE SHOWING PLACE TO ADD USERS:



Group Permissions

1. Click on Datacenter > Groups > Add
   - Apply Administrator group on the Datacenter

IMAGE SHOWING WHERE TO CUSTOMIZE GROUP AND USER PERMISSIONS:



Last Updated: 2/22/2025
Contributors: Lilian, Vivian

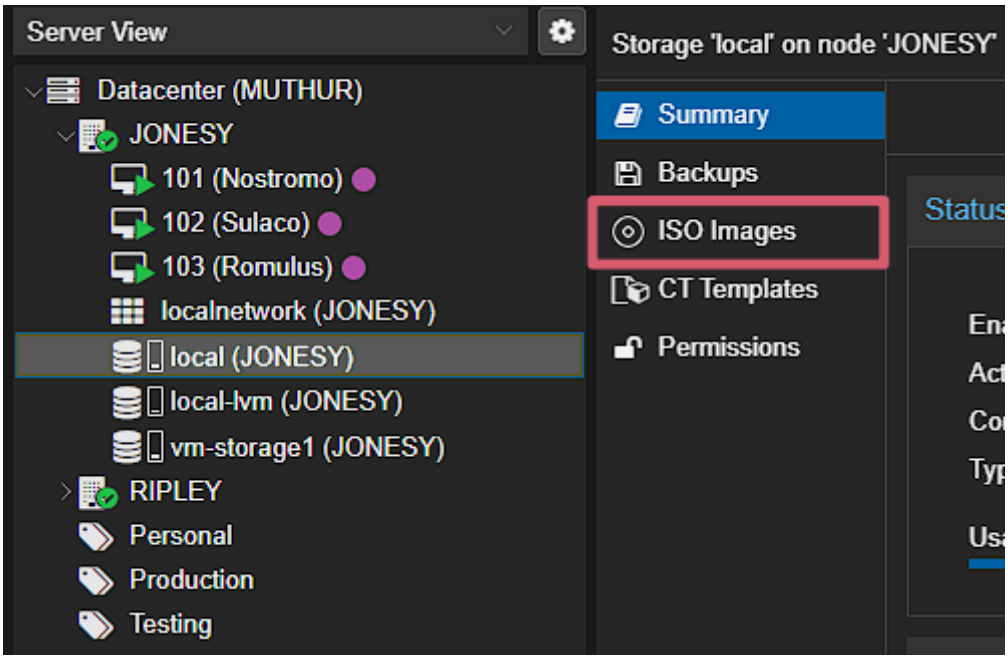
3.0 Create a VM

If the ISO for the operating system that you desire is not already available in the existing catalog of ISO Images, you must download it. Please do look first, to see if it is already there.

1. Download ISO images of VMs you'd like to virtualize:
   - Ubuntu Server
   - Ubuntu Desktop
   - Windows Desktop
2. Upload ISOs to Proxmox
   - Click on a node > local (node name) > ISO Images > Upload

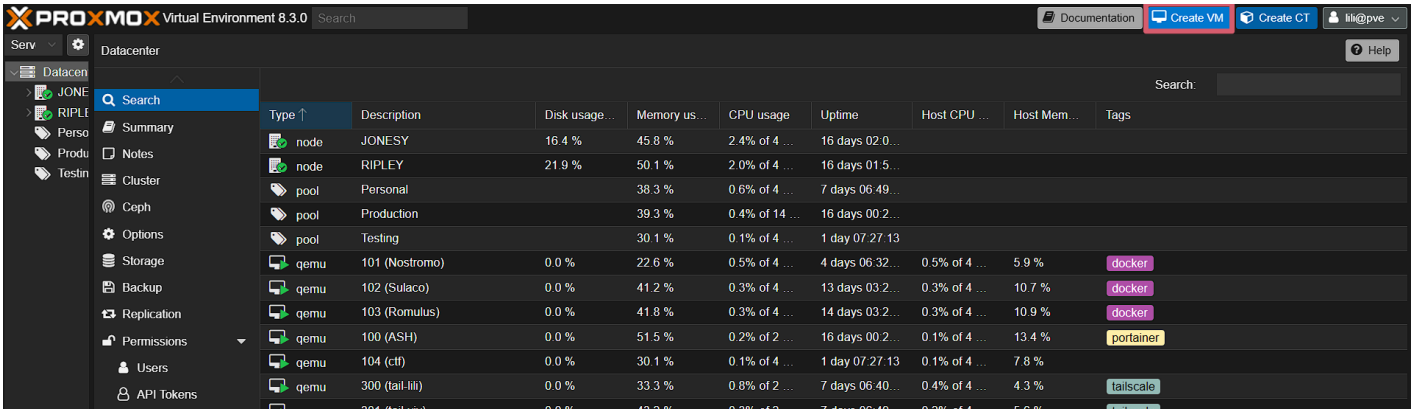
     > **You will have to repeat the upload on each node you have**

IMAGE SHOWING WHERE TO UPLOAD ISO IMAGES:



3. Create a new Ubuntu Server VM on the Proxmox server
   - Create a VM on the top right of the Proxmox GUI
   - Allocate enough resources *(i.e. CPU, storage, memory, etc.)* for your use case

IMAGE SHOWING WHERE TO CREATE A VM:



Last Updated: 2/20/2025

Contributors: Lilian, Vivian

4.0 Clustering

Once you have your Proxmox nodes initialized, it's quite easy to get them clustered together.

Creating a Cluster

1. Choose the machine you want to be the primary node for the cluster
2. Click on the "Datacenter" at the top left of the interface
3. Click on "Cluster"
4. Choose "Create Cluster"
5. After that is done creating, click on "Join Information" and then "Copy Information"

Joining a Cluster

1. On another machine, click on "Join Cluster"
2. Paste the join information from the other node into the box
3. Fill out any other necessary boxes and join.

Last Updated: 6/20/25
Contributors: Lilian

Dump of Keiths notes: Proxmox

This is a dump of my notes from the last few years

proxmox cloud init notes

Cloud init in proxmox by default offers basic functionality to create a user, add a password for that user, decide if the machine should be updated and import ssh keys and doing basic network config. This is good enough to get started using ansible but cloud init is capable of so much more including setting up repositorys, instaling packages, running arbitrary commands to configure the system, perform advanced network setup and setup many users on the system by default and much much more.

I am largely using information from this article https://dustinrue.com/2020/05/going-deeper-with-proxmox-cloud-init/ to get this project off the ground. I have been struggling to understand how to get this working for the last while and I think this is going to be a great feather for my toolbox.

First we started by editing the Cephfs volume storage at the datacenter level to allow for snippets. This allows the file system to use the cloud init file that we create to apply it to the cloud init disk we already have "installed" on one of our templates.

Next we create a file in the /mnt/pve/Cephfs/snippets/ directory that will be our cloud init user file. This will include things like creating our user, adding ssh keys, and installing packages. If you plan to use a more complicated netplan file than just turning on DHCP for the first network interfaces you can also setup a network cloud init file. You could create a meta file as well but Im not sure why that would be nessicary.

A few things I would like to know and I will test as I go though this. If I leave some fields blank can I still use the basic config information to apply things like the system hostname? That would be helpful to know.

I am going to be working off examples from the cloud init documentation for this. I would like to have my systems at least have a bare minimum config of being setup to work with ansible and also to have mdns installed. below is a cloud init file that should allow that. Im also going to create an additional user to get into the system. it doesnt seem like it will use the cloud init from the default user and it also seems like the network setup is fucked as well

To select a custom cloud init file for a specific VM you use the command `qm set <VMID> --cicustom "user=local:snippets/user.yaml`

```
#cloud-config


# Install additional packages on first boot
#
```

```
# Default: none
#
# if packages are specified, then package_update will be set to true
#
# packages may be supplied as a single package name or as a list
# with the format [<package>, <version>] wherein the specific
# package version will be installed.
packages:
 - libnss-mdns
 - qemu-guest-agent

# A common use-case for cloud-init is to bootstrap user and ssh
# settings to be managed by a remote configuration management tool,
# such as ansible.
#
# This example assumes a default Ubuntu cloud image, which should contain
# the required software to be managed remotely by Ansible.
#

ssh_pwauth: false
users:
- name: borg
  gecos: Ansible User
  groups: users,admin,wheel
  sudo: ALL=(ALL) NOPASSWD:ALL
  shell: /bin/bash
  ssh_authorized_keys:
    - "ssh-rsa YOUR KEY HERE"
```

This all ended up working pretty well with a few caveats. I don't think you can create a user named admin on an Ubuntu machine. At least the system failed to create that user. There are a few other important things I learned.

First, you can only use either use the builtin cloud init tools from the proxmox UI **or** you can use custom cloud init tools using the `cicustom` command. This is slightly unfortunate because I cannot quite figure out how to pass in the name of the machine used by proxmox to the custom cloud init file at the buildtime for the new vm. this isnt a huge issue but it does mean a tweak in how the systems are spun up and setup. the systems all come out with the same host name if useing cicustom.

I think that this is the best solution for some projects. If a template is setup to generate specificly configured systems (like the tailscale templates) the cicustom command makes more sense so you

can get a basic config and

templates

https://technotim.live/posts/cloud-init-cloud-image/

Follow the above guide!

I will use the **fart** net for only local system networking

The rest of this tutorial seems do-able. Ill probably want to do this at home.

SDN vs pfsense and or opensense

The process for setting up pfsense or opensense in proxmox and then messing with it looks something like this...

first install opnsense, add multiple network adapters before logging in. one of those adapters will be setup using just a default bridge to the existing lan network and one of the bridges will use the same interface but will have a vlan tag on the interface. this way all traffic is created inside this vlan.

the next step is to install an ubuntu desktop system on the proxmox host as well and setup 1 network interface there without vlan tag untill after ubuntu is installed. then add the vlan tag that matches the interface on the pfsense vms lan interface. this way youll be able to get access to the web console.

When trying to figure all this stuff out I was unclear on how connections should work between machines and software defined networks.

lxc containers

just reading though the lxc contianers documentation on proxmoxes wiki, key take aways are such

- containers are lightweight (duh)

- no live migration due to technical limitations (migrations are handled by restarting the contianer but this is quick because containers are light)

- only linux systems can be emulated (containers)

- for security reasons containers run seperate namespaces and some syscalls are not allowed in containers (im not sure I fully understand this point but it sounds like podman where it locks stuff down more than docker for security)

- You can use proxmox ve firewall and high avaliblity framework with containers

- the goal of lxc containers to to provide benifits of a vm without additional overhead. LXC containers should be thought of as system containers instead of application containers (docker syle containers (generaly))

- Proxmox recomends if you want to run applications like you would with docker to do it in a vm with docker installed so all syscalls and other stuff like live migration is avalible along with stronger isolation from the host system.

all in all, I could be somewhat wrong about LXC containers and if we want to spin up quick crash and trash systems we could consider them as viable test platforms. the live migration being lost is a bit of a bummer.

https://pve.proxmox.com/wiki/Linux_Container

Tailscale templates

A bit more on the ansible stuff

[[Tailscale Ansible Automation]]

The steps I have got though to setup tail-scale Templates for Proxmox.

- create fresh Ubuntu VM

- user is called 'administrator'

- password will be kept for admin

 users setup for guests will have sudo access only for tailscale functions

XOXO- QEMU guest agent installed

- tailscale installed

- subnet router rules applied to system

- udp optimizations setup

- avahi-daemon setup for mdns id of systems

- machine id removed

- ssh host information and keys removed

- cleared out Ethernet device classifications

Onboarding a user:

- create user information for the user in the vault warden, they need to add the install script per linux server onboard steps from tailscale

- Get them to setup tailscale account and one node on their stuff they plan to use for their microspace stuff

- split DNS setup for yeticraft.net (thanks josh)

- need to know what IP range they will be allowed to access

- need to know if they need additional network interfaces to make that happen

- will want their email address to send them the link to authorize their account

- I should figure out how to email...
https://docs.ansible.com/ansible/latest/collections/community/general/mail_module.html

Does the user need access to the system???

Need to experiment...

I don't think so... If we put in the correct information from an admin perspective we should be able to run tailscale up --advertise routes 192.168.x.0/x with all stuff included. Then we paste them the login link. I think that there is a way to

Need to talk over with the fellas if this seems like a safe and or sane way of doing stuff...

avahi-daemon (mDNS linux client) installed to get hostname. tailscale nodes will be updated based on hostname

The link below is the fix for ssh not working (when you wipe out template keys for the template they need to be setup again on the new vm)

https://www.reddit.com/r/Proxmox/comments/sgyv24/ssh_after_clone_doesnt_work/

Vm's that are full clones can be independent of the the Ceph cluster. We are probably going to go with linked clones. Not sure how that works on the backside but it does seem to be functional

Things an admin will need to do.

- create machine based on template

  - users should have a tag with there name on it and

  - select linked clone (unless putting on system not in Ceph. Probably don't do that)

  - add to remote-access pool (haven't created this yet)

  - wait for VM to be created

  - select correct network interfaces before brining system up (this will dictate the IP address the user gets)

- run `sudo dpkg-reconfigure openssh-server` to generate fresh keys

- import ssh keys (gh or other)

- change hostname to correct numerical designation `hostnamectl hostname tailscale-node#` replace # with number

- reboot system to change hostname

- add new host to new guest inventory section

- set administrator user as a no password user (reworking the template may be necessary for this)

- run ansible playbook to setup system (Crowdsec and other stuff)

- walk user though getting authkey for a Linux server. This is not an awesome way to go about this. I have a feeling that getting input from the fellas could lead to better solutions...

-

- tailscale up --advertise-routes *route ip based on allowed addresses*

- ~~give user the login link (need to figure out how long this will wait) I could potentially automate this bit with ansible and email... That would be cool~~

- move to regular guest inventory

- ensure nymph can get to the system for regular updating

- ensure the user can get to services they expect they should be able to get to.

- Will need to implement a full follow along guide for user.

Today I got this mostly working. I don't really know if there are better ways I could do this for now but this seems like an excellent start.

I am going to do a second version. There are a few things I need to get correct

---

After review with the u-space team this is how I'm gonna tweak stuff

A better procedure generally would be such

- user installs tailscale, enorles at least one system, setup split dns

- enroll user in bitwarden (get auth key for user along with having them generate passwords for whatever services they will be using) we should encourage users to use the vault-warden account for all passwords they generate for the project they are in. That way resets are easier for us. This is ultimately where LDAP and SSO will eventually come into play

- create VM for users access including correct network interfaces. There's a bunch of steps here, will outline later

- add VM to ansible creation inventory

- run initial install playbook

- when user has their auth key in the system, paste into ansible var for auth key as part of playbook to spin up system

- the tailscale join should be the last playbook. Should include the subnet routing to users expected resources. (Nextcloud, dev environment like coder, help desk software, redmine, oodoo, WordPress site backend, ECT.)

- user needs to enable subnet routes in the admin console and also the host system the are using (--accept-routes for Linux. Assuming that windows is a toggle. Android auto forwards afaik)

- check with user that they can login to their services

Steps I need to complete.

- Build a fresh template. I think I'm going to try and automate the install and setup of system variables like the tailscale install, setting up the hostname, configuring to use mDNS and other items at the top of this note can all be automated and we can use the cloud init tools to create the VM. Use the techno tim video to get this VM setup.

- probably need to implement logging push up to network monitoring tools (this is something Josh/garth should be consulted on)

- harden nymph

https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-amd64.img

Stuff for the ansible playbook

Tailscale optimizations

https://tailscale.com/kb/1320/performance-best-practices

for setting up nodes with this new setup.

1. go to template 8000

2. create a clone

3. target node: mss1/2

4. VM ID: leave alone (let it auto populate)

5. Name: tailscale-node#

6. resource pool: tail-nodes

7. mode: full clone

8. target storage: same as source

9. create clone

10. do not turn on vm once its cloned!

11. set up network devices as needed

12. turn on system

13. get ip address for host (mdns will be installed)

14. ensure nymph can contact hosts

15. run config ansible playbooks (what do those playbooks do?)

Lets talk about password manager integration...

I have yet again, bounced off of working with a password manager and ansible. the issue ive got is that all the implementations I can think of seem incredibly fragile and dificult to setup. I also realised some of the limitations of the community edition of vault warden. my main issue is the lack of granularity when it comes to password user control.

It seems that we would have to add users to the vaults that they need access to, (part of that is sending them an invite link) Once they are in the system they would have access to both their vault and the organization they need access to. they would then add their auth tolken to their personal account and then share it with an admin user?

---

roleplay

Here is my tabletop of a user and admin interaction

In this scenario, a user needs to gain access to several development machines as well as nextcloud for document storage and redmine for project managment. the admin has access to the proxmox cluster to provision this new user a system with access to their cluster of systems. the admin also has access to bitwarden. the project lead sends a message to admin group to establish users in this group.

Project Manager:

Good morning, I need users1, user2, user3 and user4 to have access to my workgroup X. please enroll them. here are their email addresses

Admin Managment:

Check rodger, those users will be added

Admin: *send email to users*

Good morning, Im going to help enroll you in our system. you will recive an invite to join the bitwarden account with access to your workgroups vault. please create your account. Once complete, i will send you an enrolement invite to join the u-space organization along with adding you to relevant password vaults for your work group

Once your account is created you'll also create a tailscale account using this link https://tailscale.com/ you will need to provide an identity provider such as github or google. If you already have a tailscale account you can skip this step.

Once you have tailscale setup on your devices of choice, You will need to go to the settings menu in the tailscale admin dashboard. then select keys, and then generate an auth token. your key for

this account should not be re-usable. do not select any of the switches.

That auth-tolken will then be saved under your name in a note with the "enroll tolken vault collection"

From there, I will create your machines. Please reach out to me if you have troubles signing up for tailscale or the bitwarden account

Breif overview of steps

1. sign up for yeticraft vault warden

3. sign up for tailscale

4. admin will enroll you in enroll vault collection and other relevant vaults for your work group

5. generate an auth key and save it to your vault

6. send that item to admin@admin.example

Your remote access will be established to the systems in work group X. if you need additional access please talk to your project manager about provisioning those resources.

Users: do the tasks as outlined

Admin:

- create systems. one for each user. add users systems network interfaces to software defined network for that work group. create entrys for those users in the ansible inventory. then get into the bitwarden account and get the auth tolkens. these should be added to an ansible vault and mapped to the system ID they belong to.

- configure ansible vault with auth tolkens and ip addresses for internal ip's of the systems

- running the playbook to enrole new users will run any hardening steps required, configure a strong administrator password, install tailscale and authorize the system for the users network as well as allowing the subnets needed for the workgroup.

A few notes

Im going to spend some time working on play optimization steps. for example, stripping out the gather facts step if it is not required, dont try to install shit if its already there and dont create users that already exist.

This explains the check_mode tool in the users module which is helpful

https://stackoverflow.com/questions/75211712/ansible-user-module-check-if-a-user-exists-or-not

checkout these modules for checking if stuff is already there

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/package_facts_module.html

---

So how did this all end up?

Well It seems that things worked pretty well. Currently this has all largely come together. What I have now is a system that will allow many users access to internal services for projects. This system still needs a bit of refinement as I had to do a good amount of tweaking to get everything working. I think that I need to run though this whole process again now that the ansible scripts are working. I may also move my access into the microspace to that method and see how it works to access internal services. Everything internal to the machine that Im on should be fairly fast given that there's no real network connections.

As I finished up this project a project came to my attention to host a tailscale proxy system that proxy's all the services in its docker network. This project is fairly easy but I'm not sure if it would work for our needs over all since it seems like its to allow one user to get access to resources. I am going to do a deeper dive into this TSDProxy tool and see if it can be applied to our usecase. It could be that for users who dont need access to full virtual machines and only need access to internal services like nextcloud this could be a more appropriate tool to use.

I have gotten tsdproxy to work, The next step is to see if I can get multiple tailscale accounts link up to a single tailscale proxy stack. that way it can be much more efficent to give access to services for many users we can just setup

Somehow your template got all fucked up. gonna try again using

https://dev.to/minerninja/create-an-ubuntu-cloud-init-template-on-proxmox-the-command-line-guide-5b61

https://blog.themaxtor.boo/post/2024-10-12-how-to-create-proxmox-template-with-cloud-image/

https://blog.themaxtor.boo/post/2024-10-19-how-to-create-vm-with-cloud-init-in-proxmox/

I was able to re-establish the Ubuntu-cloud template. this was an excellent starting point and i was very glad to have it back. I was finally able to crack being able to add customized cloud init user files.

I think the problem I was running into had several causes. Mostly though I think the main issue was that I was incorrectly formatting the cloud init files and expecting things that were not possible. It seems like you can either use the values entered in the proxmox ui for cloud init or you can use the ci-custom command to overwrite those things with a customized version. You only get to use one. This means that I will end up tweaking the procedure for an admin to spin up the tailscale nodes significantly. the new procedure looks like this once all data is gathered from the user.

1. select the Tailscale-Template and create a clone

2. select mss1/mss2

3. leave vmid alone

4. give name of ts-node# (number of the tailscale node your adding)

5. add to Tailscale-Node Resource pool

6. select Full Clone mode

7. leave target striagre at same as source

8. leave format as QEMU image format (qcow2)

9. Wait for system lock to go away

10. Do not start system yet

11. add system to relevant network (default is simple LAN (fartnet), this may or may not be an appropriate option for each user, that system should have some sort of dhcp method for assigning ip addresses.

12. Tag each system with the tsnode tag, the users name as a tag, and the project they are on

13. start new system

14. you can watch the system go though its first boot process, wait for it to finish this process or at least wait for 10 minutes

15. reboot system once cloud init has run

16. the system should now report its ip address to the summary page using the qemu-guest-agent, enter this ip address into the ansible playbook for these new machines in the new guests section.

17. enter user auth tokens into the ansible vault in the keys feild (It is very likely that this portion could be automated)

Ansible, tailscale, proxmox automated templates

Stuff for the ansible playbook

Tailscale optimizations generally, mostly for subnet routers

https://tailscale.com/kb/1320/performance-best-practices

Tailscale subnet router setup

https://tailscale.com/kb/1019/subnets

The vault look-ups

this project helped me get a better idea on how ansible looks up variables. this diagram is how I understand this as working.

![[Drawing 2024-12-11 17.47.07.excalidraw.png]]

basicly the way I understand this working is the vars file is used to map in variables from diffrent places including the inventory file and also the vault file. kinda brings everything together.

This was the main addition that this playbook added to my knowledge of ansible. Learning to work with variables and pull information from different sources will be incredibly handy going forward. As far as I can tell this method of pulling variables works but could be turned into a oneliner and hopping over the vars file? It is possible this is wrong as Im still learning about how this all works.

This is the rest of the ansible script written for this project. Beyond the use of vars files the main things I think are cool about this playbook is the use of registering facts about the packages installed using the ansible.builtin.package_facts module to pull information about the systems targeted. I also register the outputs of certain plays to check if other plays should run. this helps when running this play against systems after they have been configured once.

```YAML
---

- name: preliminary steps

  hosts: new_guests

  become: true

  vars_files:

    - ./vars.yaml

./vault.yaml

tasks:
```

```
- name: Set a hostname

  ansible.builtin.hostname:

    name: "{{ inventory_hostname }}"

- name: gather facts about packages installed

  ansible.builtin.package_facts:

    manager: auto

- name: install packages

  ansible.builtin.apt:

    upgrade: full

- name: Check if reboot is required

```

ansible.builtin.stat:

path: /var/run/reboot-required

register: reboot_required_file

- name: Reboot systems to apply kernel updates

ansible.builtin.reboot:

when: reboot_required_file.stat.exists == true

- name: setup crowdsec

hosts: new_guests

become: true

vars_files:

```
    - ./vars.yaml

    - ./vault.yaml




  tasks:




-name: Crowdsec repo's are installed via script

ansible.builtin.shell: curl -s https://install.crowdsec.net | sh

register: my_output # <- Registers the command output.

changed_when: my_output.rc != 0 # <- Uses the return code to define when the task has changed.

when: "'crowdsec' is not in ansible_facts.packages"

-name: Crowdsec install

 
```

ansible.builtin.apt:

package:

- crowdsec

update_cache: true

when: "'crowdsec' is not in ansible_facts.packages"

- name: Crowdsec install

ansible.builtin.apt:

package:

- crowdsec-firewall-bouncer-iptables

update_cache: true

when: "'crowdsec' is not in ansible_facts.packages"

- name: Enroll in crowdsec console

  ansible.builtin.command: sudo cscli console enroll -n {{ inventory_hostname }} -e context clz8lrn840007lb085o6va59z

  register: my_output # <- Registers the command output.

  changed_when: my_output.rc != 0 # <- Uses the return code to define when the task has changed.

  when: "'crowdsec' is not in ansible_facts.packages"

- name: Add linux collection

  ansible.builtin.command: sudo cscli collections install crowdsecurity/linux

  register: my_output

  changed_when: my_output.rc != 0

when: "'crowdsec' is not in ansible_facts.packages"

- name: Restart crowdsec post enrollment to get stuff working

ansible.builtin.service:

  name: crowdsec

  state: restarted

when: "'crowdsec' is not in ansible_facts.packages"

- name: install and enroll tailscale on hosts

  hosts: new_guests

  become: true

  vars_files:

```
    - ./vars.yaml


    - ./vault.yaml




  tasks:


-name: install and enroll host in tailscale

ansible.builtin.shell: curl -fsSL https://tailscale.com/install.sh | sh && sudo tailscale up --auth-key="{{ tailscale_auth_key }}"

when: "'tailscale' is not in ansible_facts.packages"

-name: setup subnet routers pt 1

ansible.builtin.shell: echo 'net.ipv4.ip_forward = 1' | sudo tee -a /etc/sysctl.d/99-tailscale.conf

# when: "'tailscale' is not in ansible_facts.packages"

-name: setup subnet routers pt 2

 
```

ansible.builtin.shell: echo 'net.ipv6.conf.all.forwarding = 1' | sudo tee -a /etc/sysctl.d/99-tailscale.conf

when: "'tailscale' is not in ansible_facts.packages"

- name: setup subnet routers pt 3

ansible.builtin.shell: sudo sysctl -p /etc/sysctl.d/99-tailscale.conf

when: "'tailscale' is not in ansible_facts.packages"

- name: subnet router optimizations

ansible.builtin.shell: printf '#!/bin/sh\n\nethtool -K %s rx-udp-gro-forwarding on rx-gro-list off \n' "$(ip -o route get 1.1.1.1 | cut -f 5 -d " ")" | sudo tee /etc/networkd-dispatcher/routable.d/50-tailscale && sudo chmod 755 /etc/networkd-dispatcher/routable.d/50-tailscale

- name: advertise routes for systems

ansible.builtin.shell: tailscale up --advertise-routes "{{ subnets }}"

```
```

This is the vars file included with this project. the commented out lines are from earlier iterations of this project that are helpful to understand how things are looked up using variables

```YAML
# ansible_password: "{{ server_passwords.admin }}"

# ansible_become_password: "{{ server_passwords.admin }}"

tailscale_auth_key: "{{ individual_sys[server_id].key}}"

# new_borg_password: "{{ individual_sys[server_id].borg_pass }}"
```

This is the vault file that I created and is useful for understanding how auth-keys are looked up while stored in this secure format. It is sanitized and the keys used are only useful once anyway

```YAML
individual_sys:

  ts-node1:
```

key: tskey-auth-kfjkdeadfkeeCNTRL-ueoiruyaoieuryaiosudfyoaisudyfudh

ts-node2:

key: tskey-auth-kfjkdeadfkeeCNTRL-ueoiruyaoieuryaiosudfyoaisudyfudh

ts-node3:

key: tskey-auth-kfjkdeadfkeeCNTRL-ueoiruyaoieuryaiosudfyoaisudyfudh

```

---

This is the inventory used for this project, Useful for understanding vars and the workflow with this script

```YAML

### uspace stuff

## Once configured, hosts from new_guests get moved to guests

```
you should be able to switch hosts from ip addresses to mdns names

Systems configured like this could also be swapped over to mainline inventorys

##

new_guests:

  hosts:

    tailscale-node3:

    # ansible_host: tailscale-node1.local

    ansible_host: 10.0.0.13

    server_id: ts-node6

    subnets: 192.168.4.0/24

  vars:
```

```
    #ansible_user: administrator

    ansible_user: borg




guests:

  hosts:

    tailscale-node1:

    ansible_host: tailscale-node1.local

   # ansible_host: 10.0.0.11

    server_id: ts-node1

    subnets: 192.168.0.0/24



    tailscale-node2:
```

```
    ansible_host: tailscale-node2.local

  # ansible_host: 10.0.0.12

    server_id: ts-node2

    subnets: 192.168.0.0/24



  vars:

    ansible_user: borg
```

I did do a bit of work for a previous version of this project before i figured out custom cloud init files. below are those files. at this time I was creating a borg user, setting a borg user password, and targeting an admin account that required a password for running sudo commands. I was also passing in the whole enrollment command instead of just an auth-key which is much more cumbersome. ultimately this ended up being unnecessary.

vars:

```yaml
```

ansible_password: "{{ server_passwords.admin }}"

ansible_become_password: "{{ server_passwords.admin }}"

tailscale_auth_command: "{{ individual_sys[server_id].command}}"

new_borg_password: "{{ individual_sys[server_id].borg_pass }}"

```

vault:

```YAML

server_passwords:

admin: password123456!@#$

individual_sys:

tsn1:

command: curl -fsSL https://tailscale.com/install.sh | sh && sudo tailscale up --auth-key=tskey-auth-k8iqCx4jQV11CNTRL-xW3gNm6FuKf7ra49zQ32SfTKRJnVCGbC

borg_pass: ez8aaBkNxRyKGeR9zhdxLRQGoJfoMEEiRuptxamr

tsn2:

command: curl -fsSL https://tailscale.com/install.sh | sh && sudo tailscale up --auth-key=tskey-auth-kBxMEyjHu111CNTRL-3JzrF8UmkRacSdd4yHbuKav7udHYBkeua

borg_pass: aoQCCYphdFvBhzxxnVQW4C6XfrtDtePKUeAhUxj9

```

here's the playbook I wrote for this its basically the same as one I ultimately used with a few slight tweaks. It also includes the steps to bootstrap a borg user.

```YAML

---

- name: setup apt-cache proxy

hosts: new_guests

become: true

vars_files:

- ./vars.yaml

- ./vault.yaml

tasks:

- name: remove apt-cache proxy info

ansible.builtin.file:

path: /etc/apt/apt.conf.d/00proxy

state: absent

register: proxy

- name: reboot all hosts to apply changes

```
ansible.builtin.reboot:

when: proxy.changed

- name: setup mdns and install general use packages

hosts: new_guests

become: true

vars_files:

- ./vars.yaml

- ./vault.yaml

tasks:

- name: gather facts about packages installed
```

```
ansible.builtin.package_facts:

manager: auto

- name: install packages

ansible.builtin.apt:

pkg:

- libnss-mdns

- qemu-guest-agent

update_cache: true

when: "'libnss-mdns' is not in ansible_facts.packages"




- name: apply hardening

hosts: new_guests
```

become: true

vars_files:

- ./vars.yaml

- ./vault.yaml

tasks:

- name: Crowdsec repo's are installed via script

ansible.builtin.shell: curl -s https://install.crowdsec.net | sh

register: my_output # <- Registers the command output.

changed_when: my_output.rc != 0 # <- Uses the return code to define when the task has changed.

when: "'crowdsec' is not in ansible_facts.packages"

```
- name: Crowdsec install

ansible.builtin.apt:

package:

- crowdsec

update_cache: true

when: "'crowdsec' is not in ansible_facts.packages"



- name: Crowdsec install

ansible.builtin.apt:

package:

- crowdsec-firewall-bouncer-iptables
```

update_cache: true

when: "'crowdsec' is not in ansible_facts.packages"

- name: Enroll in crowdsec console

ansible.builtin.command: sudo cscli console enroll -n {{ inventory_hostname }} -e context clz8lrn840007lb085o6va59z

register: my_output # <- Registers the command output.

changed_when: my_output.rc != 0 # <- Uses the return code to define when the task has changed.

when: "'crowdsec' is not in ansible_facts.packages"

- name: Add linux collection

ansible.builtin.command: sudo cscli collections install crowdsecurity/linux

register: my_output

changed_when: my_output.rc != 0

when: "'crowdsec' is not in ansible_facts.packages"

- name: Restart crowdsec post enrollment to get stuff working

ansible.builtin.service:

name: crowdsec

state: restarted

when: "'crowdsec' is not in ansible_facts.packages"

- name: Create borg user

hosts: new_guests

become: true

vars_files:

- ./vars.yaml

- ./vault.yaml

tasks:

- name: Check if users exist

ansible.builtin.user:

name: borg

check_mode: true

register: test_users

```
- name: Set user password from vault based on host index for new user borg

ansible.builtin.user:

name: borg

shell: /bin/bash

groups: sudo

password: "{{ new_borg_password }}"

append: true

create_home: yes

no_log: true

when: "test_users is false"



- name: Add borg user to sudoers with no password
```

ansible.builtin.copy:

src: /home/borg/tailscale-project/borgaddin

dest: /etc/sudoers.d/

owner: root

group: root

mode: "0600"

when: "test_users is false"

- name: Turn off cloud init

ansible.builtin.file:

path: /etc/cloud/cloud-init.disabled

mode: "0600"

```
state: touch



- name: Restart cloud-init

ansible.builtin.service:

name: cloud-init

state: restarted



- name: Upload SSH key

ansible.posix.authorized_key:

user: borg

key: "{{ lookup('file', '/home/borg/.ssh/id_rsa.pub') }}"

state: present
```

```
when: "test_users is false"




- name: install and enroll tailscale on hosts

hosts: new_guests

become: true

vars_files:

- ./vars.yaml

- ./vault.yaml




tasks:

- name: install and enroll host in tailscale

ansible.builtin.shell: "{{ tailscale_auth_command }}"
```

when: "'tailscale' is not in ansible_facts.packages"

- name: setup subnet routers pt 1

ansible.builtin.shell: echo 'net.ipv4.ip_forward = 1' | sudo tee -a /etc/sysctl.d/99-tailscale.conf

when: "'tailscale' is not in ansible_facts.packages"

- name: setup subnet routers pt 2

ansible.builtin.shell: echo 'net.ipv6.conf.all.forwarding = 1' | sudo tee -a /etc/sysctl.d/99-tailscale.conf

when: "'tailscale' is not in ansible_facts.packages"

- name: setup subnet routers pt 3

ansible.builtin.shell: sudo sysctl -p /etc/sysctl.d/99-tailscale.conf

when: "'tailscale' is not in ansible_facts.packages"

- name: advertise routes for systems

ansible.builtin.shell: tailscale up --advertise-routes "{{ subnets }}"

```

I also wrote a few scripts to test things...

```YAML

### This playbook is just a tester to see how the when flag(?)

### it also checks out the package_facts module

### This playbook is useful for figuring out

- name: check if packages exist, skip or run

hosts: new_guests

become: true

gather_facts: false

```
tasks:

- name: gather facts about packages installed

ansible.builtin.package_facts:

manager: auto

- name: run arbitrary command if a specific package is installed

ansible.builtin.debug:

msg: this is a test1

when: "'libnss-mdns' is in ansible_facts.packages"

- name: run an arbitrary command if a package is not installed

ansible.builtin.debug:

msg: this is another more diffrent test
```

when: "'nmap' is not in ansible_facts.packages"

- name: run arbitrary command if a specific package is installed

ansible.builtin.debug:

msg: this is a test2

when: "'libnss-mdns' is not in ansible_facts.packages"

- name: run an arbitrary command if a package is not installed

ansible.builtin.debug:

msg: this is another more diffrent test2

when: "'nmap' is in ansible_facts.packages"

```

```YAML

```

This script sets a hostname on systems targeted

matches hostnames to the inventory hostname because why not

- name: set hostname for hosts

hosts: targeted systems

become: true

tasks:

- name: Set a hostname

ansible.builtin.hostname:

name: "{{ inventory_hostname }}"
```

Backup server setup

Once proxmox ve is installed on a system and proxmox backup server is installed on a separate server (ideally on the same subnet) you can get the two working together by adding proxmox backup server as a storage volume in the proxmox data center. Follow along with the required information and things will get backed up. From there you can save backups directly or you can setup scheduling. The scheduling is where I got to and stopped. We will likely want to do some resource pooling and decide who and what gets backed up where.

LXC container for lightweight remote access

I am running a lxc container for my access to the makerspace. here are the steps I took to stand up the lxc container. You can use this as a guide for standing up your own personal LXC container. you dont have to do tailscale on there but this has some info on securing the container.

1. login as root/PAM auth on the proxmox cluster

2. get to the shell on a particular system

3. run a proxmox convience script to standup a very basic ubuntu lxc container

4. once the container is stood up and running get into the root console and do the following

1. change password for root `passwd` to something secure and your own

2. modify system to prevent auto-login using the command below and remove the `--auto-login root` portion of the line that is there.

```
nano /etc/systemd/system/container-getty@1.service.d/override.conf
```

3. reboot the container

4. login to your root user

5. run `wget https://github.com/YOUR-GITHUB-USERNAME.keys` to pull down your ssh keys

6. modify your /etc/ssh/sshd_config file to allow for root login via ssh

7. verify you can ssh to the system

8. modify system hostname using proxmox gui under container > DNS > hostname

9. reboot to apply hostname

10. once all that works install tailscale the normal way you do that on linux servers.

1. setup subnet routing

2. **TURN OFF SUBNET ROUTING ONCE IT WORKS** we should be using the wireguard VPN for access. this is a backup in the case the wireguard is acting up

5. once the system is configured, your backdoor now should work just fine. I recomend not going crazy with this system.