Self-hosting a Secure Home Lab

Issac Kim | Henry Reed | Maya Zeng

Internet: https://github.com/henryreed/scale23x-homelab-workshop
Workshop LAN: http://10.1.1.4:3000/henryreed/scale23x-homelab-workshop

Who We Are

- Issac Kim
  - Cyber Engineer at a non-profit, pentester
  - Not a weeb and not a rhythm game fanatic
- Henry Reed
  - Cyber Engineer at Zetier, Inc., primarily doing vulnerability research on embedded Linux systems
  - Formerly a Project Leader at The Aerospace Corporation
  - Zetier is hiring! Ask me after we finish with the workshop
- Maya Zeng
  - IT Security Analyst
  - I'll be running around to help out
- Oli
  - Emotional Support Quail

Rules of Engagement

- Heed the SCaLE Code of Conduct (https://www.socallinuxexpo.org/scale/23x/code-conduct)
- All of you will have differing technical backgrounds, and some topics will be obvious while others will not be
  - We want you to learn as much as possible. Please ask questions!
  - Even if you feel other students know the answer to your question, the purpose of this workshop is to teach every one of you–your questions are important and will drive the discussion for this class!
- We will have some default passwords to make lives easier
  - Do NOT attack any other student’s systems, to include their laptops and their assigned Proxmox accounts, virtual machines, and other resources, no matter how “easy” it is
- There are maximum 50 accounts. If we run out of accounts, please pair up with a neighbor and work with their system

Outline

Secure Network Architecture

Section 1

Defining Risk

- An effect of uncertainty on or within information and technology; relate to the loss of confidentiality, integrity, or availability of information, data, or information (or control) systems and reflect the potential adverse impacts to organizational operations (i.e., missions, functions, image, or reputation) and assets, individuals, other organizations, and the Nation.
- What are the risks within your homelab?
  - Compromise of sensitive data (private photos, files)
  - Attacks on users or user systems
  - Unintended use of your resources (botnet or malware activity)

Sources: https://csrc.nist.gov/glossary/term/cybersecurity_risk

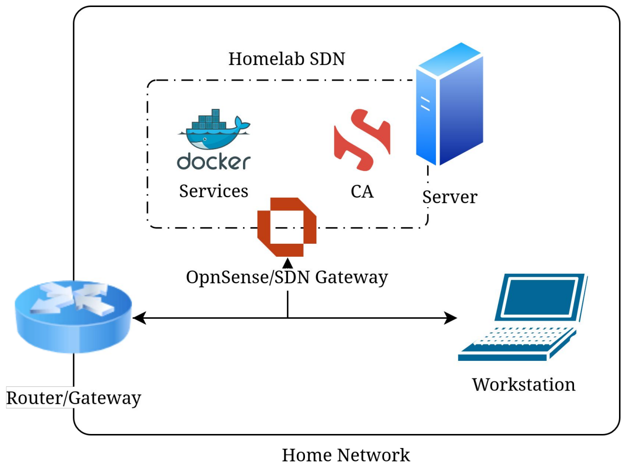
Threat Modeling for your Home Lab

- Understanding your threat model helps you understand what is “good enough”
- An average person’s home lab will have threats of opportunist attackers
  - Automated scanners and throwers for the latest CVE in the news
    - E.g., log4j
  - Manual attacks by opportunists
  - Potential supply-chain attacks via malicious software images or packages
    - E.g., the (failed) xz supply chain attack
- Advanced persistent threats (APTs) are generally not a concern

Security Goals for Your Home Lab

- Popular advice in forums focuses on security doctrine
  - “Never open ports to the Internet”
  - “Just use wg-easy”
  - Very few threads on security testing of your network
- Flip the script
  - Instead of relying on axioms, define your requirements, limit access, add authentication, and test, test, test!

Example of a Good Network Architecture



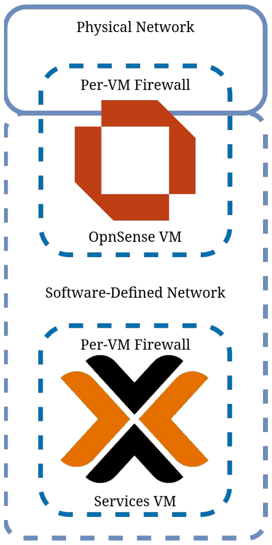
- Services hosted within Proxmox software-defined network (SDN)
- Access services via OpnSense through Network Address Translation (NAT)
- Services may only access update servers; no arbitrary Internet access
  - DNS sinkholing + hostname-based IP whitelisting

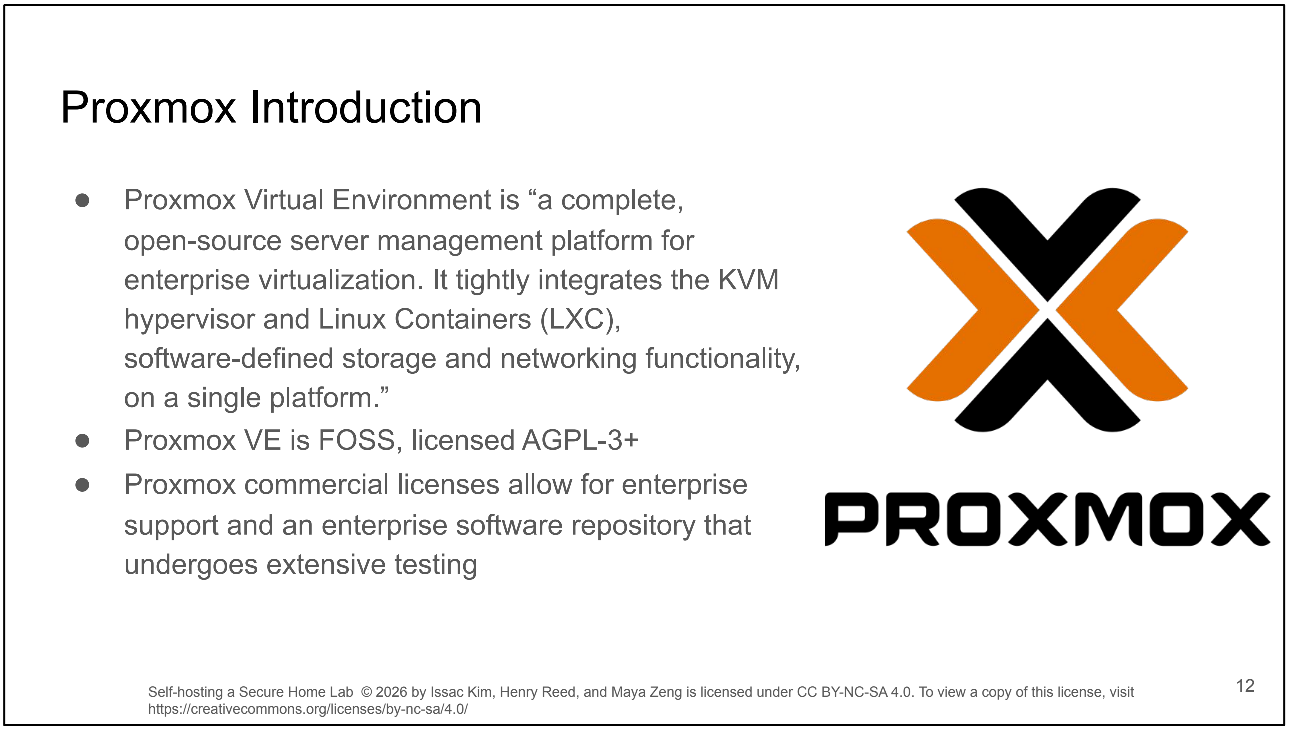
Proxmox: Setting up virtual machines and software-defined networks

Section 2

What is Section 2 About?

- Our services should be hosted behind a firewall that allows us to allow inbound HTTPS, but only selectively allow outbound traffic
- In this section, we will cover how to:
    - Create a software-defined network, where our virtual machine running our services will be
    - Clone a ready-made VM that will be (temporarily) dual-homed, to make OpnSense configuration possible
    - Create an OpnSense VM and configure it





Proxmox Introduction

- Proxmox Virtual Environment is “a complete, open-source server management platform for enterprise virtualization. It tightly integrates the KVM hypervisor and Linux Containers (LXC), software-defined storage and networking functionality, on a single platform.”
- Proxmox VE is FOSS, licensed AGPL-3+
- Proxmox commercial licenses allow for enterprise support and an enterprise software repository that undergoes extensive testing

PROXMOX

Proxmox offers the following description on their website for their Proxmox Virtual Environment (VE) product:

“Proxmox Virtual Environment is a complete, open-source server management platform for enterprise virtualization. It tightly integrates the KVM hypervisor and Linux Containers (LXC), software-defined storage and networking functionality, on a single platform. With the integrated web-based user interface you can manage VMs and containers, high availability for clusters, or the integrated disaster recovery tools with ease.” [1]

Proxmox VE is one of four products offered by Proxmox Server Solutions GmbH. The other three products are Proxmox Backup Server, Proxmox Datacenter Manager, and Proxmox Mailbox Gateway. For the scope of this workshop, we will only cover Proxmox VE. We will also refer to Proxmox VE as simply “Proxmox.”

Proxmox VE has licensing options. Unlicensed Proxmox VE instances receive no enterprise support, and the packages available in the Proxmox repository do not undergo the same level of scrutiny as those in the enterprise subscription [2]. In the many years of use, the authors of this workshop have not ran into any issues with Proxmox’s no-subscription repository packages. We would consider the paid licenses to be entirely optional for a majority of home lab environments.

[1] https://www.proxmox.com/en/products/proxmox-virtual-environment/overview, accessed 22FEB26

[2] https://www.proxmox.com/en/products/proxmox-virtual-environment/pricing, accessed 22FEB26
[3] https://pve.proxmox.com/wiki/Package_Repositories, accessed 22FEB26

Proxmox Virtual Machines and Software-Defined Networks

- Proxmox Virtual Machines
  - Virtual TPMs, which are especially helpful in virtualizing OS’s mandating TPMs (Windows 11)
  - Per-VM firewalls, allowing the hypervisor admin to set firewall rules for a guest that the guest itself cannot control
  - Snapshots, including memory snapshots, and backups
  - High availability, allowing one VM to run on multiple Proxmox hosts such that the virtual machines remains available even if some Proxmox instances fail
- Software-Defined Networks
  - Allows you to create virtual networks without relying on physical routers, switches, firewalls
  - Can be as simple as a virtual LAN, or as complicated as a full BGP-enabled network
  - We will only cover the simple virtual LAN that allows us to segment our services VM from the rest of the physical network

Workshop’s Proxmox

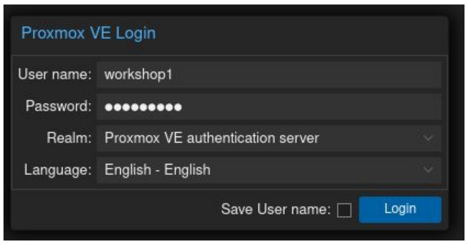
- We have two beefy (for seven years ago) servers running Proxmox VE
- Despite sharing nearly half a terabyte of RAM across them, we will be resource constrained due to the number of workshop attendees
- To limit the attack surface, ideally:
  - Each service would be in its own virtual machine
  - This especially applies to critical services like a certificate authority
  - In this workshop, we will use a single “services” VM that will contain all services
- Software repositories
  - Wherever listed in these slides, please use local (and not Internet-hosted) software repositories; this is to avoid hammering both the SCaLE network and the repos themselves

Creating a Proxmox Software-Defined Network

Lab 2.1

Access Proxmox

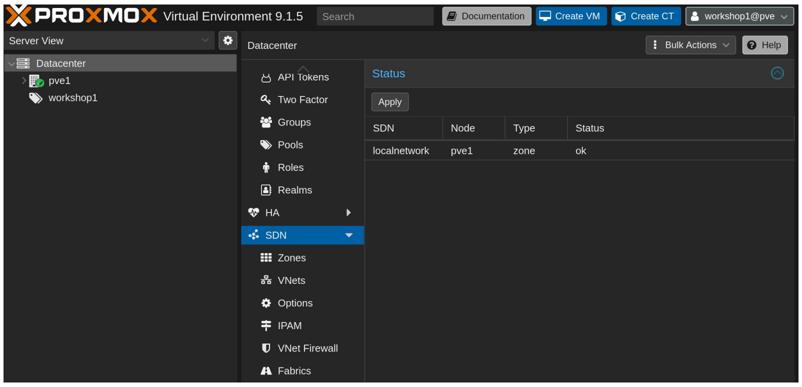
- The workshop Proxmox instance is located at
  - https://10.1.1.2:8006 (for workshop IDs 1-25)
  - https://10.1.1.10:8006 (for workshop IDs 26-50)
- On the login page, change Realm to “Proxmox VE authentication server” as seen below
- You were handed a number when you arrived, that will be your username and password for Proxmox, in the format “workshop#”, e.g. workshop1


Proxmox VE Login

User name: workshop1
Password: ●●●●●●●●●
Realm: Proxmox VE authentication server
Language: English - English

Create an SDN in Proxmox

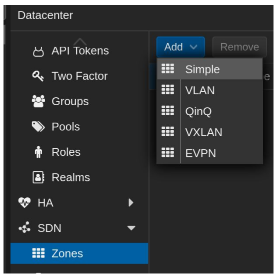
- In the Proxmox web UI, go to the top level Datacenter menu, then scroll down to SDN

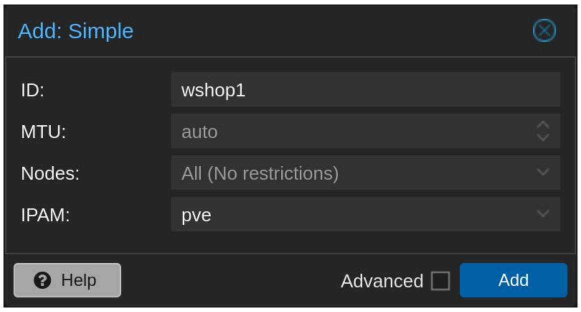


SDN	Node	Type	Status
localnetwork	pve1	zone	ok

Create an SDN in Proxmox (Zones)

- From the Datacenter section, go to SDN -> Zones
- Add a new simple zone, and name it “wshop#”, numbered after your workshop user number

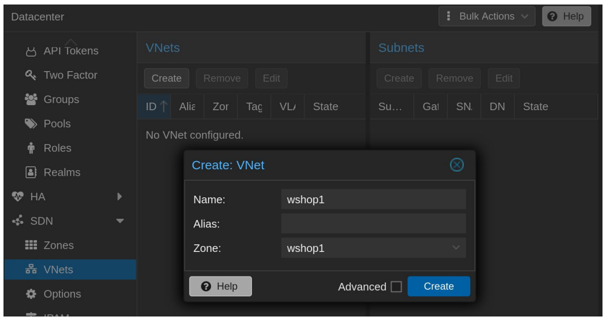




ID: wshop1
MTU: auto
Nodes: All (No restrictions)
IPAM: pve

Create an SDN in Proxmox (VNets)

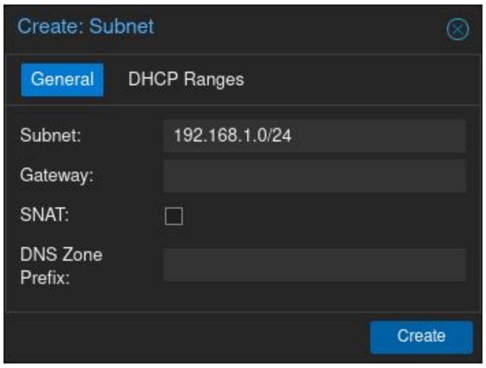
- Next, go to the VNets menu
- Create a new VNet under the zone you just made
  - This can be named anything but we recommend also calling this “wshop#”, where # is your workshop ID



Name: wshop1
Alias:
Zone: wshop1

Create an SDN in Proxmox (Subnets)

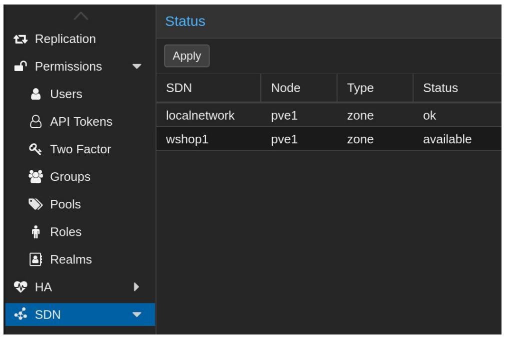
- Click on the subnet you just created, then on the right-most panel, create a new subnet
  - Set Subnet to 192.168.#.0/24, where # is your workshop ID



Subnet: 192.168.1.0/24
Gateway:
SNAT: ☐
DNS Zone Prefix:

Create an SDN in Proxmox (Final Step)

- Go back to the top level SDN menu on the side panel, then hit Apply, then confirm the changes
  - May require a page refresh but you should now see your new SDN



SDN	Node	Type	Status
localnetwork	pve1	zone	ok
wshop1	pve1	zone	available

Cloning a Proxmox Template Virtual Machine

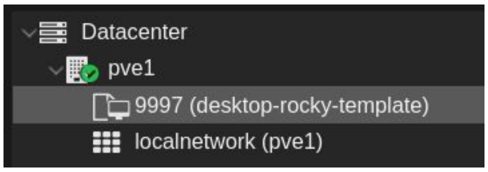
Lab 2.2

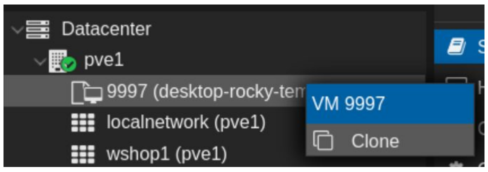
Why?

- Currently, any virtual machine inside the SDN cannot communicate to anything outside the SDN
  - This is intentional
- During initial setup, you'll need a system to work from that has a GUI with which you can access your firewall
  - You may also do a SOCKS5 dynamic proxy via SSH if you'd like
  - This system will be "dual-homed": connected to both the SDN and the lab network
  - Think of it as your network debugger
- We created a Rocky Linux template VM for you to use
  - Username admin, password admin

Cloning the VM Template (1/2)

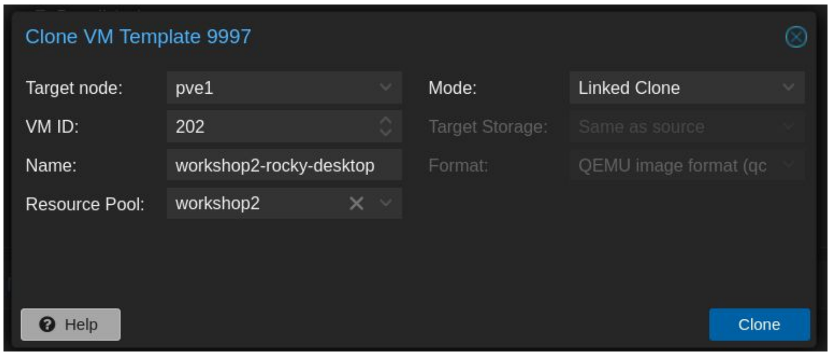
- Right click on 9997 (desktop-rocky-template)
- Select “clone”





Cloning the VM Template (2/2)

- Target node: pve1 if your number is 1-25, or pve2 if your number is 26-50
- VM ID -> First number is your workshop ID number
- Resource Pool -> You HAVE to select one here that matches your username
- Mode -> Linked Clone (help us save on disk space!)

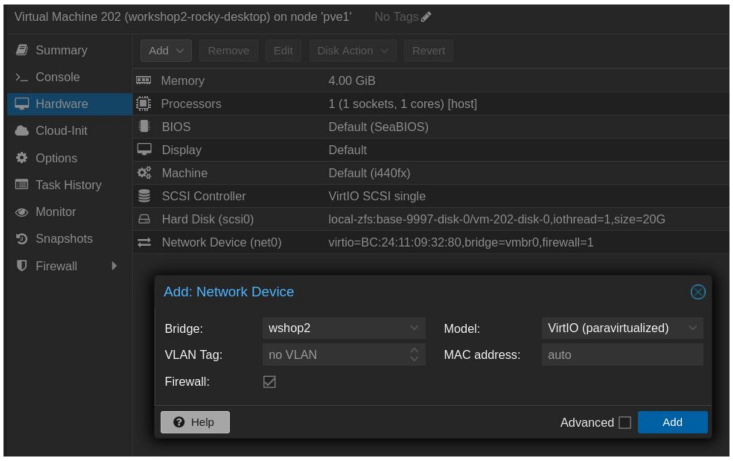


Clone VM Template 9997

Target node:	pve1	Mode:	Linked Clone
VM ID:	202	Target Storage:	Same as source
Name:	workshop2-rocky-desktop	Format:	QEMU image format (qc
Resource Pool:	workshop2		

Dual-Homing the Rocky VM (1/2)

- Hardware -> Add -> Network Device
- For the bridge, select wshop# where # is your workshop ID
- Press "Add"



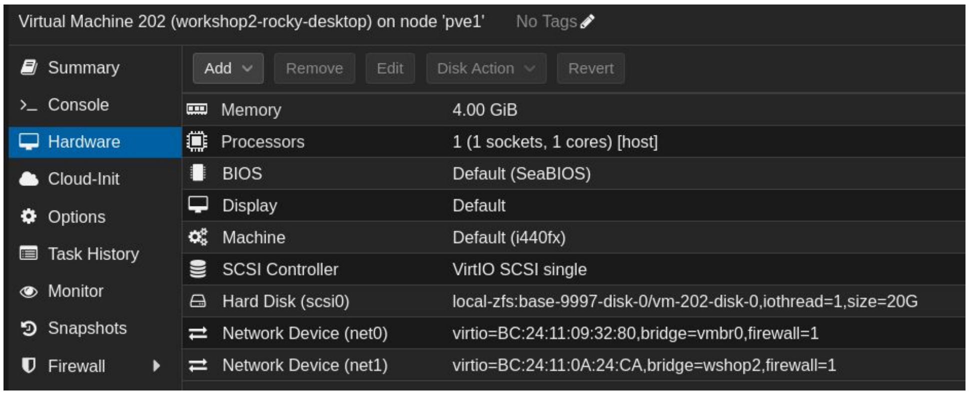
Memory	4.00 GiB
Processors	1 (1 sockets, 1 cores) [host]
BIOS	Default (SeaBIOS)
Display	Default
Machine	Default (i440fx)
SCSI Controller	VirtIO SCSI single
Hard Disk (scsi0)	local-zfs:base-9997-disk-0/vm-202-disk-0,iothread=1,size=20G
Network Device (net0)	virtio=BC:24:11:09:32:80,bridge=vmbr0,firewall=1

Add: Network Device

Bridge: wshop2
Model: VirtIO (paravirtualized)
VLAN Tag: no VLAN
MAC address: auto
Firewall: ✓
Advanced ☐

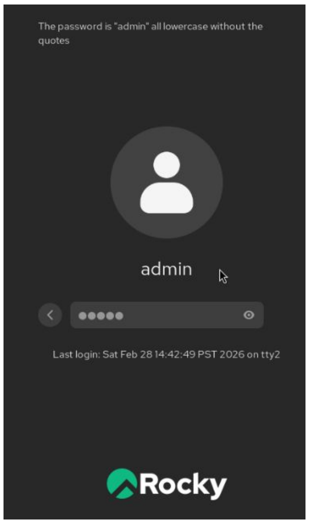
Dual-Homing the Rocky VM (2/2)

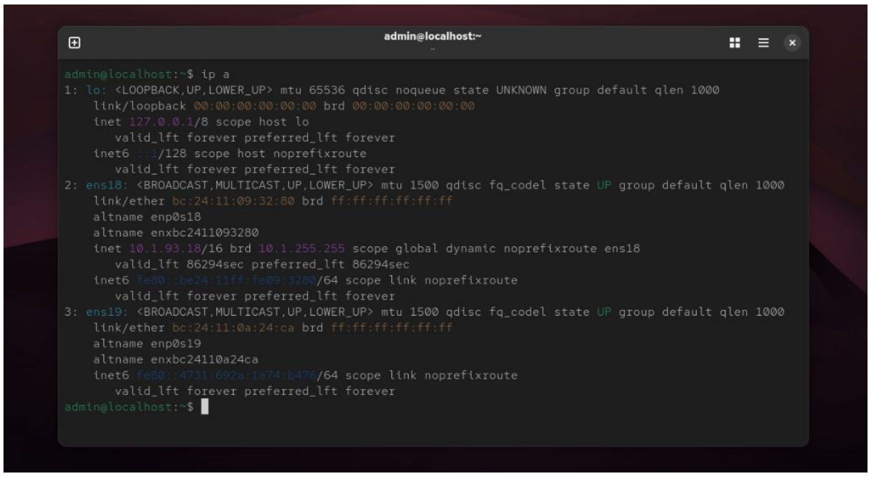
- Your hardware settings should look like this



Memory	4.00 GiB
Processors	1 (1 sockets, 1 cores) [host]
BIOS	Default (SeaBIOS)
Display	Default
Machine	Default (i440fx)
SCSI Controller	VirtIO SCSI single
Hard Disk (scsi0)	local-zfs:base-9997-disk-0/vm-202-disk-0,iothread=1,size=20G
Network Device (net0)	virtio=BC:24:11:09:32:80,bridge=vmbr0,firewall=1
Network Device (net1)	virtio=BC:24:11:0A:24:CA,bridge=wshop2,firewall=1

Start the VM and verify network interfaces show


The password is "admin" all lowercase without the quotes



```
admin@localhost:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:09:32:80 brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    altname enxbc2411093280
    inet 10.1.93.18/16 brd 10.1.255.255 scope global dynamic noprefixroute ens18
       valid_lft 86294sec preferred_lft 86294sec
    inet6 fe80::be24:11ff:fe09:3280/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:0a:24:ca brd ff:ff:ff:ff:ff:ff
    altname enp0s19
    altname enxbc24110a24ca
    inet6 fe80::4731:692a:1a74:b476/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
admin@localhost:~$
```

Creating a Proxmox Virtual Machine

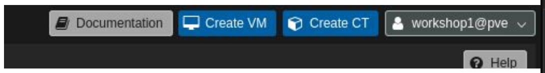
Lab 2.3

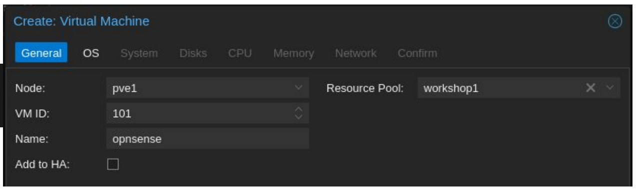
Creating an OpnSense Virtual Machine

- Now that we have a network, we will move to creating our first virtual machine from scratch
  - We’ll go over the OpnSense firewall setup steps in the following labs
- Proxmox supports a variety of operating system families since the virtual machine backend is KVM
- Proxmox also supports LXC for running lightweight containers
- For your own homelab, choose what works best for your workflow
  - In this workshop, we’ll be primarily be covering VMs

Proxmox: Creating VMs (General)

- Start by clicking “Create VM” in the top right of the Proxmox web UI
- Give your VM a unique VM ID, use the number of your workshop user followed by a two digit number.
- Under resource pool, select the pool with your username. Your VM name can be anything, but ideally recognizable.
    - For instance, workshop25 would make their first VM ID #2501

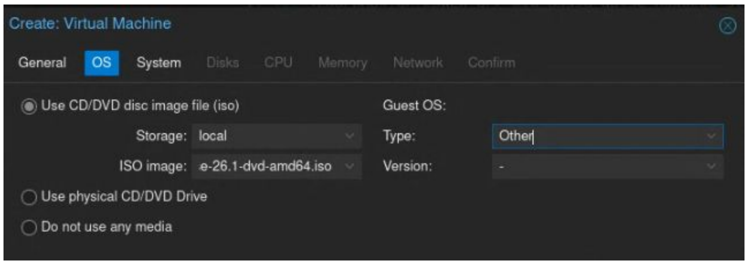




Node: pve1
Resource Pool: workshop1
VM ID: 101
Name: opnsense
Add to HA: ☐

Proxmox: Creating VMs (OS)

- Storage -> “local” storage object
- ISO image -> OS ISO that you want to install
- For Guest OS and OPNsense, change Type to “Other” as OPNsense is not Linux



Create: Virtual Machine

General | OS | System | Disks | CPU | Memory | Network | Confirm

(●) Use CD/DVD disc image file (iso)

Storage: local

ISO image: e-26.1-dvd-amd64.iso

( ) Use physical CD/DVD Drive

( ) Do not use any media

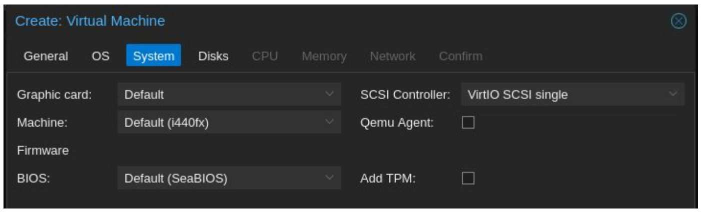
Guest OS:

Type: Other

Version: -

Proxmox: Creating VMs (System)

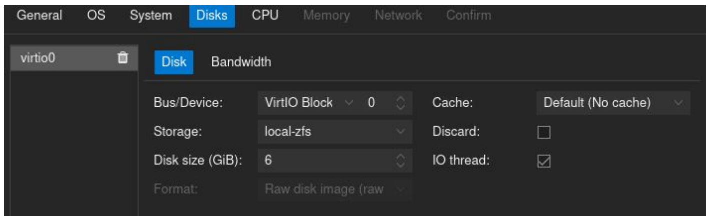
- Leave these as default for this workshop
- On a “production” home lab, you’d likely change to UEFI and enable secure boot here if you have a specific need for those settings
- Qemu agent is not necessary today, but for your own Proxmox cluster, this allows the VM guest to shut down or suspend cleanly when the cluster is shutting down or doing backup tasks on the VM



Graphic card: Default
SCSI Controller: VirtIO SCSI single
Machine: Default (i440fx)
Qemu Agent: ☐
Firmware
BIOS: Default (SeaBIOS)
Add TPM: ☐

Proxmox: Creating VMs (Disks)

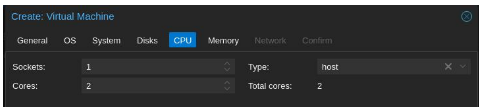
- For Disks configuration, we will change several settings:
  - Bus/Device: VirtIO Device
    - IDE is the Proxmox default since this is compatible with virtually every OS, but most modern OSes support VirtIO, which is more performant for virtualized systems
  - Storage: select the "local-zfs" datastore
  - Disk size: set to 6 GiB unless the VM/service you want to test needs more data. Please do not over-allocate the storage, even if there are a few TB free.



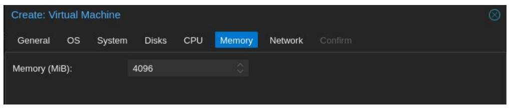
Bus/Device: VirtIO Block 0
Storage: local-zfs
Disk size (GiB): 6
Format: Raw disk image (raw
Cache: Default (No cache)
Discard: ☐
IO thread: ✓

Proxmox: Creating VMs (CPU, Memory, and Network)

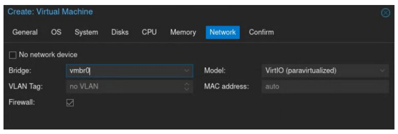
- For CPU, change the following settings:
  - Cores: 2
  - Type: Host (at the bottom of the list)
- For Memory, set to 4096 MiB
- For Network, keep it as vmbr0
  - We will add your software defined network in the next step; the firewall will be dual-homed



Sockets: 1
Cores: 2
Type: host
Total cores: 2



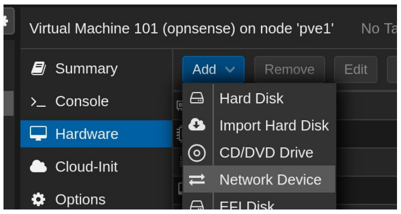
Memory (MiB): 4096

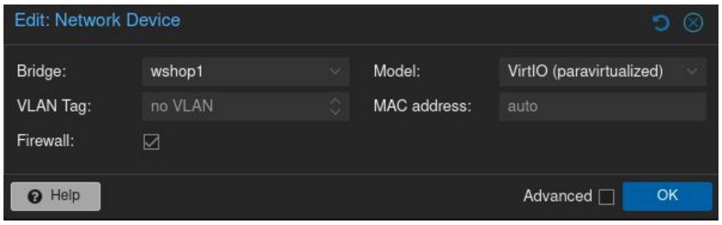


☐ No network device
Bridge: vmbr0
VLAN Tag: no VLAN
Firewall: ☑
Model: VirtIO (paravirtualized)
MAC address: auto

Proxmox: Creating VMs (OPNsense Extra Step)

- For the OPNsense VM, before installing the OS, go to the Hardware menu and add a new network device
- Under Bridge, select the SDN network you created earlier



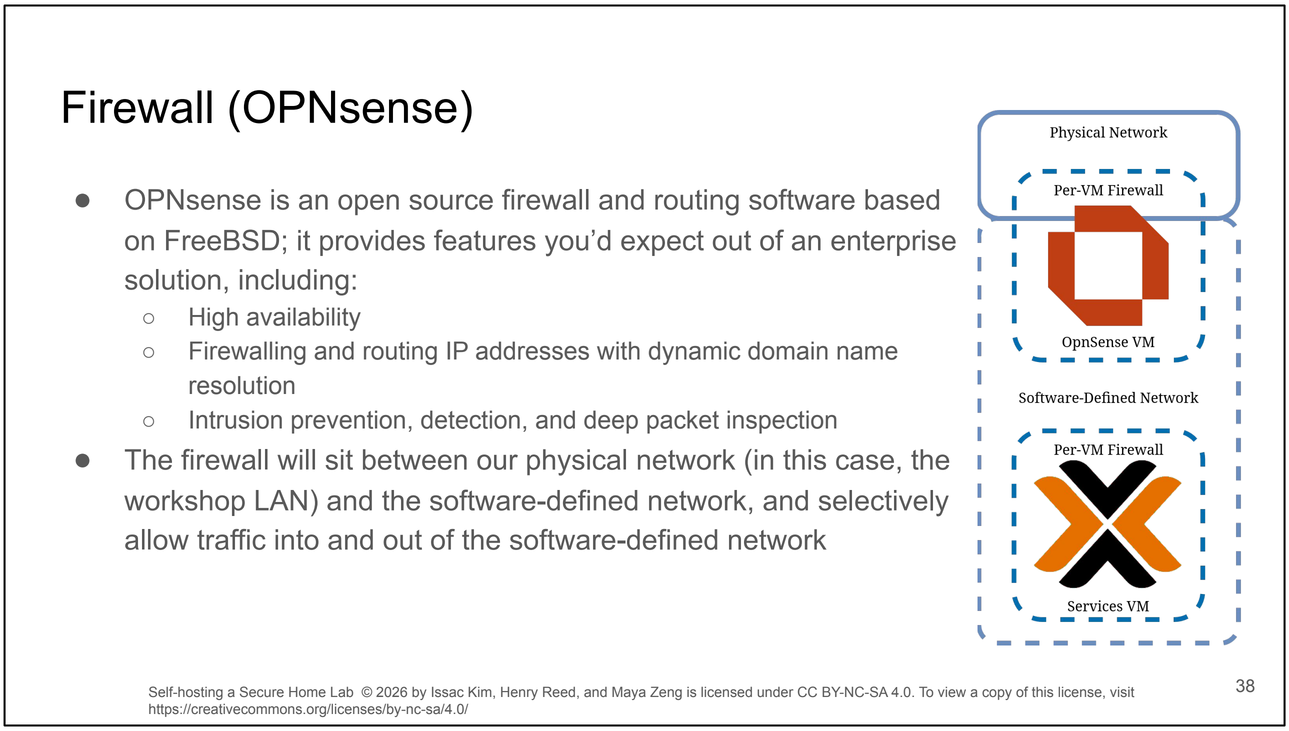

Edit: Network Device

Bridge: wshop1
Model: VirtIO (paravirtualized)
VLAN Tag: no VLAN
MAC address: auto
Firewall: ☑

OPNsense will require two virtual network devices, which will look similar for your own home lab - one for the WAN connection and one for your LAN. In this lab, the “WAN” connection will be the vmbr0 bridge that you already added in the initial VM setup. Here, we’ll add another network device to connect to the SDN you created earlier.

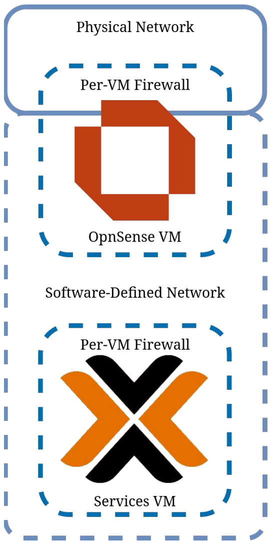
Firewall and DNS Sinkhole

Section 3



Firewall (OPNsense)

- OPNsense is an open source firewall and routing software based on FreeBSD; it provides features you'd expect out of an enterprise solution, including:
  - High availability
  - Firewalling and routing IP addresses with dynamic domain name resolution
  - Intrusion prevention, detection, and deep packet inspection
- The firewall will sit between our physical network (in this case, the workshop LAN) and the software-defined network, and selectively allow traffic into and out of the software-defined network



For more information on OPNsense, check out their website: https://opnsense.org/

Domain Name System

- DNS was originally a plaintext (not encrypted) protocol
  - It’s so old it was invented before any of the authors of this workshop were alive!
- DNSSEC added cryptographic signatures to DNS responses
- DNS over TLS added a traffic encryption layer to DNS
  - 853/TCP with manual CN configuration
  - This is beneficial for privacy, though enterprise security teams often disable this because it blocks visibility into what employees (and potentially malicious actors) are requesting
- DNS over HTTPS added even more obfuscation, making it difficult for bad guys (malware operators) or your good guys (goodware operators/enterprise security) from knowing what domain names you’re requesting

Why DNS Sinkhole?

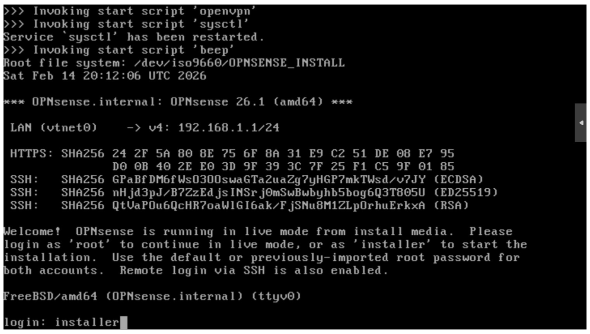
- Malware command and control (C2) can occur over any type of protocol
  - ICMP
  - Raw TCP
  - Raw UDP
  - HTTPS
  - Specific bitflips within your processor causing information-bearing electromagnetic radiation to emanate out your server chassis and into an RF collector outside your building
  - Also includes DNS
- Modern firewalls not only control traffic into and out of your LAN, but they also host services that have unfettered access to the Internet. DNS requests can be made information-bearing, allowing malware to exfiltrate telemetry to and receive commands from an operator

Configuring OPNsense

Lab 3.1

Installing OPNsense

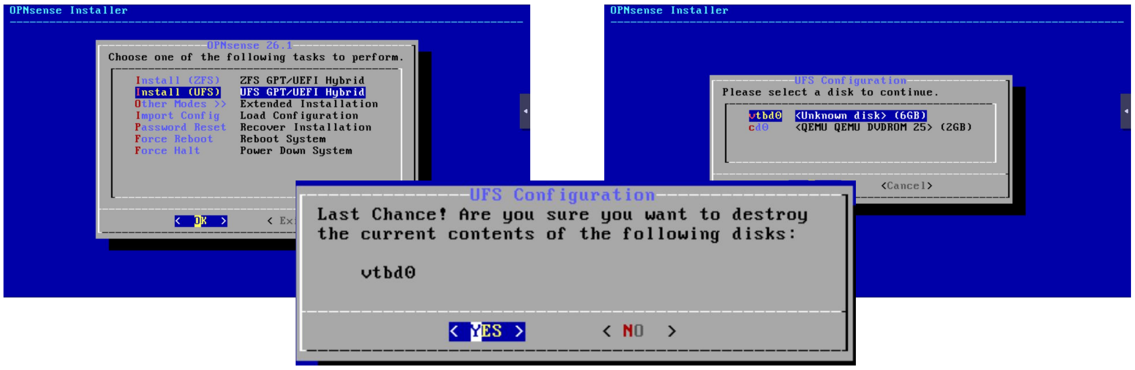
- Start the OPNsense VM you just made from the last lab, and wait for it to boot to the login prompt
  - Log in with the “installer” user, password “opnsense” to install



```
>>> Invoking start script 'openvpn'
>>> Invoking start script 'sysctl'
Service `sysctl' has been restarted.
>>> Invoking start script 'beep'
Root file system: /dev/iso9660/OPNSENSE_INSTALL
Sat Feb 14 20:12:06 UTC 2026

*** OPNsense.internal: OPNsense 26.1 (amd64) ***

 LAN (vtnet0)    -> v4: 192.168.1.1/24

 HTTPS: SHA256 24 2F 5A 80 8E 75 6F 8A 31 E9 C2 51 DE 08 E7 95
               D0 0B 40 2E E0 3D 9F 39 3C 7F 25 F1 C5 9F 01 85
 SSH:   SHA256 GPaBfDM6fWsO3OOswaGTa2uaZg7yHGP7mkTWsd/v7JY (ECDSA)
 SSH:   SHA256 nHjd3pJ/B7ZzEdjsINSrj0mSwBwbyhb5bog6Q3T805U (ED25519)
 SSH:   SHA256 QtVaPOu6QcHR7oaWlGI6ak/FjSNu8M1ZLpOrhuErkxA (RSA)

Welcome!  OPNsense is running in live mode from install media.  Please
login as 'root' to continue in live mode, or as 'installer' to start the
installation.  Use the default or previously-imported root password for
both accounts.  Remote login via SSH is also enabled.

FreeBSD/amd64 (OPNsense.internal) (ttyv0)

login: installer
```

Installing OPNsense

- Select UFS for disk type
- Select vtbd0, the virtual disk created earlier



```
OPNsense Installer

OPNsense 26.1
Choose one of the following tasks to perform.

Install (ZFS)      ZFS GPT/UEFI Hybrid
Install (UFS)      UFS GPT/UEFI Hybrid
Other Modes >>     Extended Installation
Import Config      Load Configuration
Password Reset     Recover Installation
Force Reboot       Reboot System
Force Halt         Power Down System

< OK >     < Ex
```

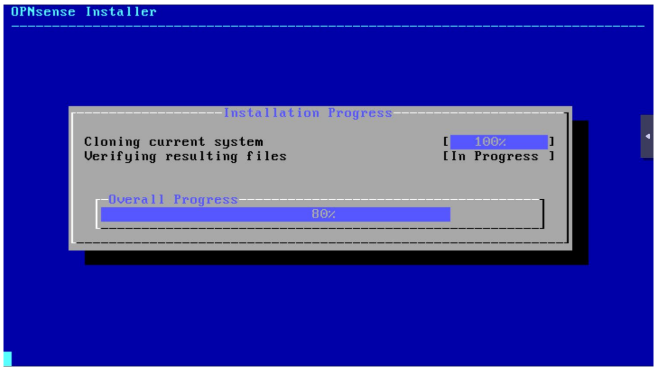
```
OPNsense Installer

UFS Configuration
Please select a disk to continue.

vtbd0     <Unknown disk> (6GB)
cd0       <QEMU QEMU DVDROM 25> (2GB)

<Cancel>
```

```
UFS Configuration
Last Chance! Are you sure you want to destroy
the current contents of the following disks:

    vtbd0

< YES >          < NO >
```

For the first step in installing OPNsense, select UFS for the install type. We would normally recommend ZFS if installed on bare metal or on a mission critical system, but since the virtual machine is already on a VirtIO disk on top of a ZFS backed storage in Proxmox, there's no need to use ZFS for this workshop.

Installing OPNsense

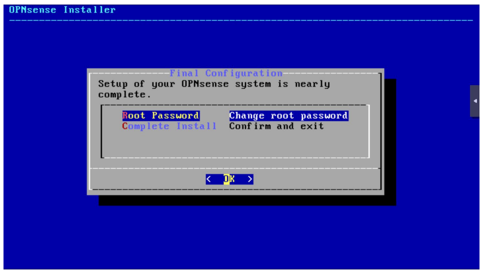
- After disk partitioning, the installer will automatically start the OS install



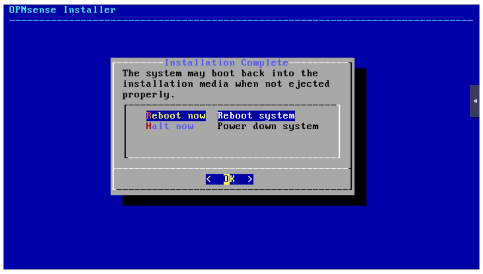
```
OPNsense Installer

                    Installation Progress
Cloning current system                         [    100%    ]
Verifying resulting files                      [In Progress ]

   Overall Progress
                         80%
```

Installing OPNsense

- Set the root user's password when prompted
- Complete install and select reboot now to reboot into the OPNsense install



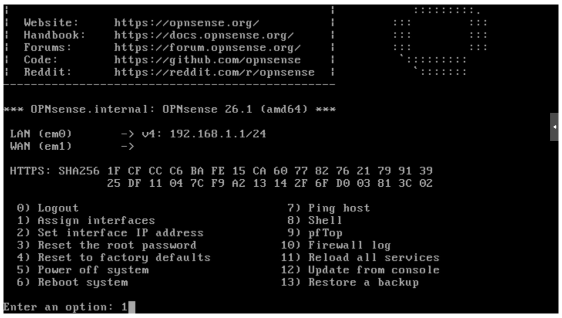
```
OPNsense Installer

Final Configuration
Setup of your OPNsense system is nearly
complete.

Root Password      Change root password
Complete install   Confirm and exit

< OK >
```



```
OPNsense Installer

Installation Complete
The system may boot back into the
installation media when not ejected
properly.

Reboot now   Reboot system
Halt now     Power down system

< OK >
```

Installing OPNsense

- Log into the OPNsense shell interface with the username “root” and the password you set in the installer, then select option 1 to assign interfaces
    - Hit enter twice on the next two options, configure LAGGs and VLANs, we’ll ignore those for this workshop

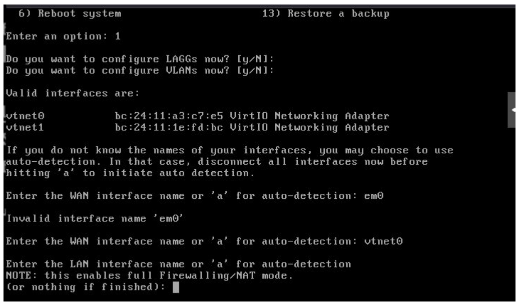


```
|                                                  |          :::::::::.
|  Website:      https://opnsense.org/             |        :::        :::
|  Handbook:     https://docs.opnsense.org/        |        :::        :::
|  Forums:       https://forum.opnsense.org/       |        :::        :::
|  Code:         https://github.com/opnsense       |         `:::::::::
|  Reddit:       https://reddit.com/r/opnsense     |           `:::::::
--------------------------------------------------

*** OPNsense.internal: OPNsense 26.1 (amd64) ***

 LAN (em0)       -> v4: 192.168.1.1/24
 WAN (em1)       ->

 HTTPS: SHA256 1F CF CC C6 BA FE 15 CA 60 77 82 76 21 79 91 39
               25 DF 11 04 7C F9 A2 13 14 2F 6F D0 03 81 3C 02

  0) Logout                              7) Ping host
  1) Assign interfaces                   8) Shell
  2) Set interface IP address            9) pfTop
  3) Reset the root password            10) Firewall log
  4) Reset to factory defaults          11) Reload all services
  5) Power off system                   12) Update from console
  6) Reboot system                      13) Restore a backup

Enter an option: 1
```

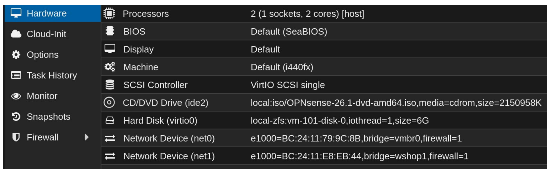
After OPNsense installs and boots to the initial shell, we’ll have to change a few settings so that it uses the correct interfaces for the WAN and LAN connection. For the workshop, we’ll want the “WAN” to be the virtual interface attached to the vmbr0 bridge, and the “LAN” to be your SDN.

Installing OPNsense

- When prompted, set the WAN interface name to the MAC address that matches the vmbr0 virtual interface, which here is vtnet0
- For LAN, this will likely be vtnet1



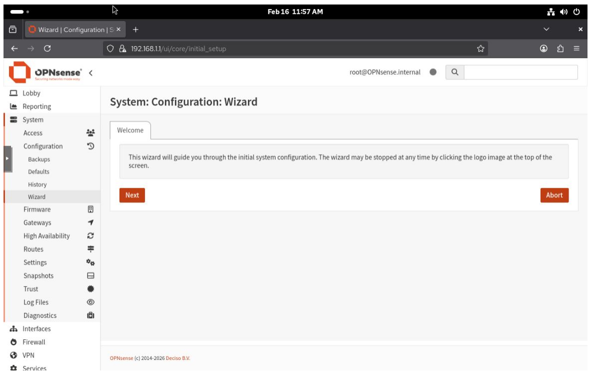
```
  6) Reboot system                     13) Restore a backup

Enter an option: 1

Do you want to configure LAGGs now? [y/N]:
Do you want to configure VLANs now? [y/N]:

Valid interfaces are:

vtnet0           bc:24:11:a3:c7:e5 VirtIO Networking Adapter
vtnet1           bc:24:11:1e:fd:bc VirtIO Networking Adapter

If you do not know the names of your interfaces, you may choose to use
auto-detection. In that case, disconnect all interfaces now before
hitting 'a' to initiate auto detection.

Enter the WAN interface name or 'a' for auto-detection: em0

Invalid interface name 'em0'

Enter the WAN interface name or 'a' for auto-detection: vtnet0

Enter the LAN interface name or 'a' for auto-detection
NOTE: this enables full Firewalling/NAT mode.
(or nothing if finished):
```



Processors	2 (1 sockets, 2 cores) [host]
BIOS	Default (SeaBIOS)
Display	Default
Machine	Default (i440fx)
SCSI Controller	VirtIO SCSI single
CD/DVD Drive (ide2)	local:iso/OPNsense-26.1-dvd-amd64.iso,media=cdrom,size=2150958K
Hard Disk (virtio0)	local-zfs:vm-101-disk-0,iothread=1,size=6G
Network Device (net0)	e1000=BC:24:11:79:9C:8B,bridge=vmbr0,firewall=1
Network Device (net1)	e1000=BC:24:11:E8:EB:44,bridge=wshop1,firewall=1

OPNsense Setup

- From your Rocky desktop VM, open Firefox and go to https://192.168.1.1, then log in to the OPNsense web UI with your root credentials

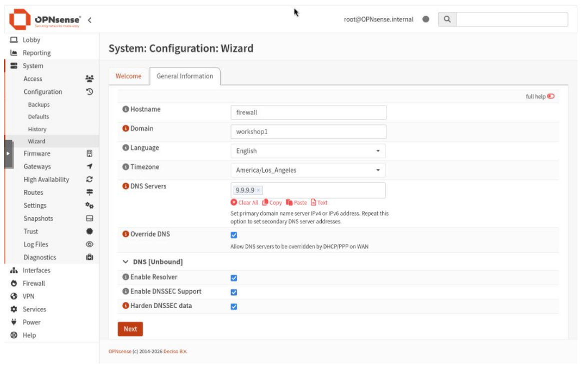


System: Configuration: Wizard

This wizard will guide you through the initial system configuration. The wizard may be stopped at any time by clicking the logo image at the top of the screen.

OPNsense Setup (General Information)

- On the first setup page, set the DNS server to be 9.9.9.9, other settings can be set to your preference
    - The DNS sinkhole setup we will do later will use the host OS as the primary resolver



System: Configuration: Wizard

Hostname	firewall
Domain	workshop1
Language	English
Timezone	America/Los_Angeles
DNS Servers	9.9.9.9 Set primary domain name server IPv4 or IPv6 address. Repeat this option to set secondary DNS server addresses.
Override DNS	✓ Allow DNS servers to be overridden by DHCP/PPP on WAN

DNS [Unbound]

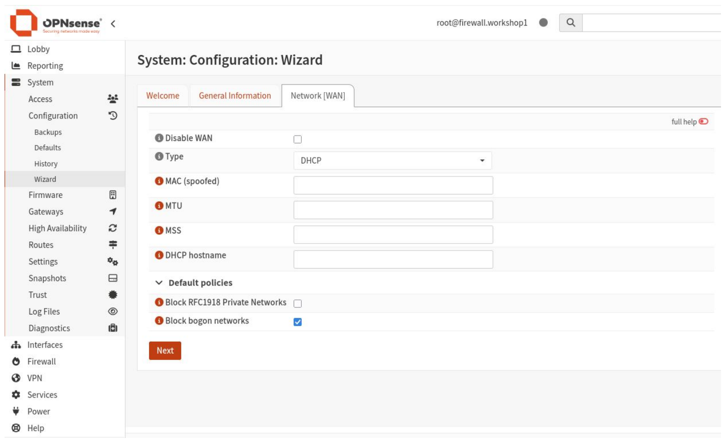
Enable Resolver	✓
Enable DNSSEC Support	✓
Harden DNSSEC data	✓

OPNsense (c) 2014-2026 Deciso B.V.

We will go in depth more into DNS later, but this will provide an initial starting point for Unbound to get up and running. In the next lab, we’ll configure the DNS sinkhole and change some of the default OPNsense Unbound DNS settings.

OPNsense Setup (WAN)

- Uncheck the Block RFC1918 Private Networks option
  - You won't have to do this on your own home network, just for this workshop
  - The WAN IP that OPNsense will get is a 10.x.x.x address, which conflicts with the default policy



System: Configuration: Wizard

Disable WAN	☐
Type	DHCP
MAC (spoofed)	
MTU	
MSS	
DHCP hostname	

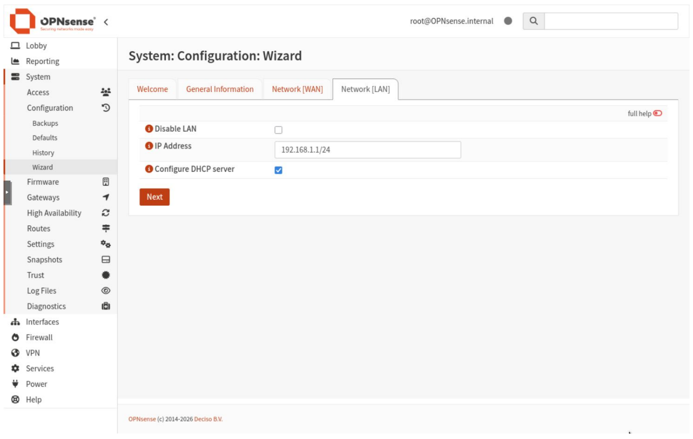
Default policies

Block RFC1918 Private Networks	☐
Block bogon networks	☑

For your own home setup, this page will only need custom settings set for the WAN connection if your ISP has specific requirements or settings that need to be set.

OPNsense Setup (LAN)

- Set the LAN IP to be 192.168.#.1/24, where # is your workshop ID



System: Configuration: Wizard

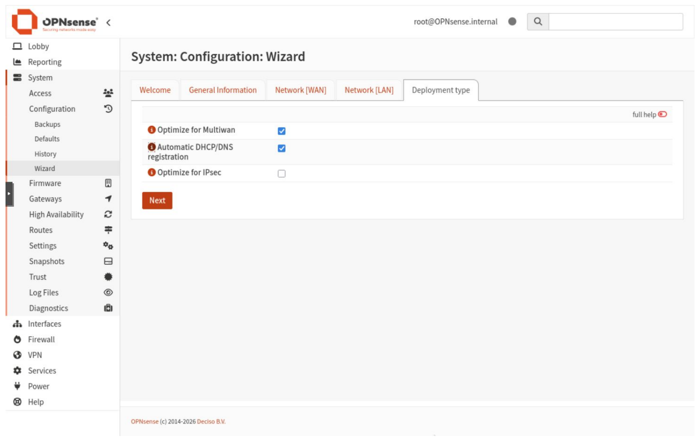
Disable LAN ☐

IP Address 192.168.1.1/24

Configure DHCP server ✓

OPNsense Setup (Deployment Type)

- Everything can be left default here

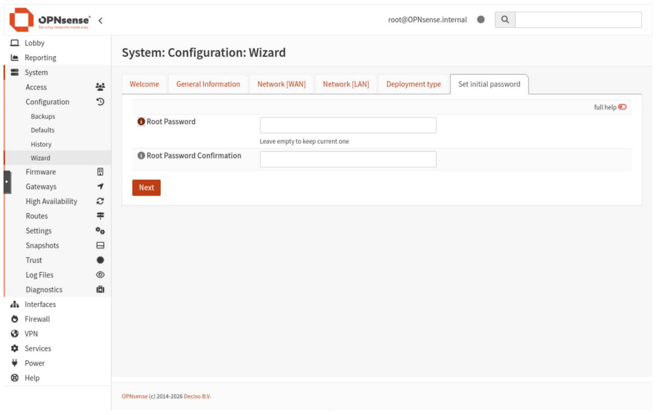


System: Configuration: Wizard

Optimize for Multiwan	✓
Automatic DHCP/DNS registration	✓
Optimize for IPsec	☐

OPNsense Setup (Password)

- You can just hit next on this section



System: Configuration: Wizard

Root Password

Leave empty to keep current one

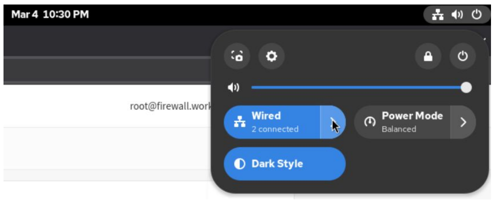
Root Password Confirmation

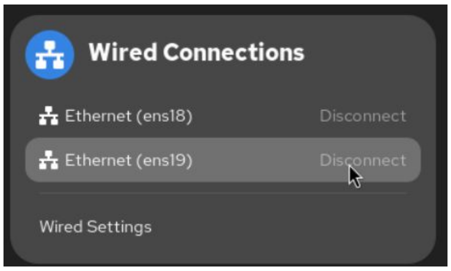
OPNsense (c) 2014-2026 Deciso B.V.

For your own home lab, ideally you would set a randomly generated root password stored securely for future use. For the purposes of convenience in this workshop, you can just stick with something simple or keep the password you used when initially installing OPNsense.

OPNsense Setup (Reconnect Network)

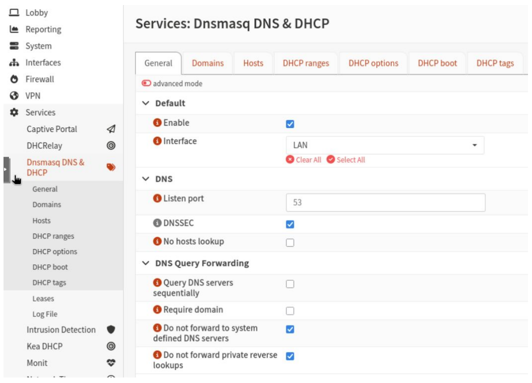
- After you finish the OPNsense wizard, you'll want to bring your Rocky VM interface down and back up to get a new IP
- Click the menu in the top right of Rocky Linux, then the drop down next to Wired
- Click on "Ethernet (ens19)" once to bring it down, then back up
- In the browser, go to https://192.168.#.1, where # is your workshop ID





OPNsense Setup (More DNS Settings)

- After the setup wizard completes, from the main page, go to Services -> Dnsmasq DNS & DHCP -> General
- Several settings to set here:
    - Set listen port to 53
    - Enable DNSSEC
    - Under DNS Query Forwarding, enable:
        - Do not forward to system defined DNS servers
        - Do not forward private reverse lookups
- Scroll down and click Apply



Services: Dnsmasq DNS & DHCP

Default

Enable	✓
Interface	LAN

DNS

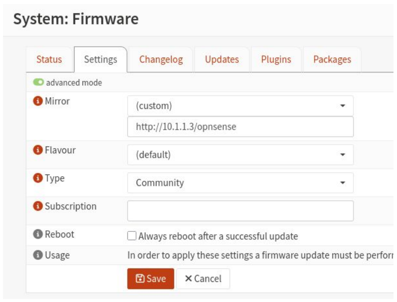
Listen port	53
DNSSEC	✓
No hosts lookup	☐

DNS Query Forwarding

Query DNS servers sequentially	☐
Require domain	☐
Do not forward to system defined DNS servers	✓
Do not forward private reverse lookups	✓

OPNsense Setup (System Update)

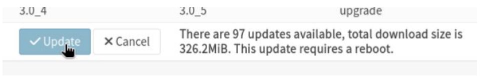
- Next, we'll set up the package repo to pull from the local workshop repo. Go to System -> Firmware -> Settings, and set the Mirror option:
  - Mirror: (custom) http://10.1.1.3/opnsense
- Click Save



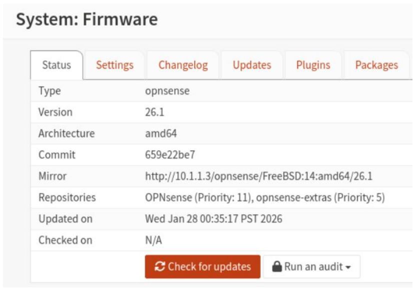
Mirror: (custom) http://10.1.1.3/opnsense
Flavour: (default)
Type: Community
Subscription:
Reboot: ☐ Always reboot after a successful update
Usage: In order to apply these settings a firmware update must be perfor

OPNsense Setup (System Update)

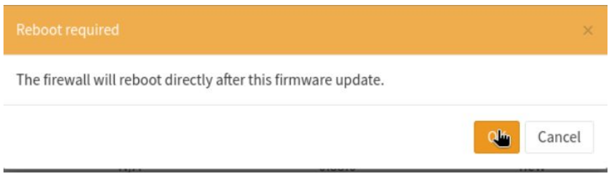
- Please *do not* update the OPNsense VM during the workshop, but if you were to update:
    - Click on the Status tab, and click Check for updates
    - Once the check finishes, scroll to the bottom of the screen then click Update



There are 97 updates available, total download size is 326.2MiB. This update requires a reboot.


System: Firmware

Type	opnsense
Version	26.1
Architecture	amd64
Commit	659e22be7
Mirror	http://10.1.1.3/opnsense/FreeBSD:14:amd64/26.1
Repositories	OPNsense (Priority: 11), opnsense-extras (Priority: 5)
Updated on	Wed Jan 28 00:35:17 PST 2026
Checked on	N/A


Reboot required

The firewall will reboot directly after this firmware update.

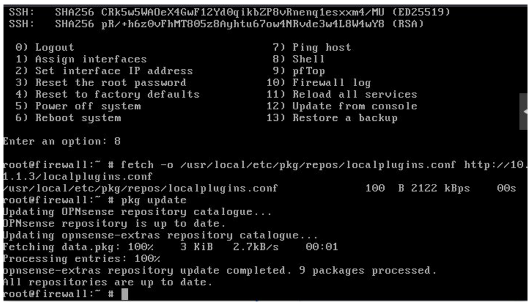
Configuring AdGuard Home for a DNS Sinkhole

Lab 3.2

OPNsense Setup (AdGuard)

- Go to the OPNsense VM console on Proxmox, log in, then choose option 8 to enter the shell
- Run the following commands:

```
# fetch -o /usr/local/etc/pkg/repos/localplugins.conf http://10.1.1.3/localplugins.conf
# pkg update
```



```
 SSH:   SHA256 CRk5w5WAOeX4GwF12Yd0qikbZP8vRnenq1esxxm4/MU (ED25519)
 SSH:   SHA256 pR/+h6z0vFhMT805z8Ayhtu67ow4NRvde3w4L8W4wY8 (RSA)

  0) Logout                              7) Ping host
  1) Assign interfaces                   8) Shell
  2) Set interface IP address            9) pfTop
  3) Reset the root password            10) Firewall log
  4) Reset to factory defaults          11) Reload all services
  5) Power off system                   12) Update from console
  6) Reboot system                      13) Restore a backup

Enter an option: 8

root@firewall:~ # fetch -o /usr/local/etc/pkg/repos/localplugins.conf http://10.
1.1.3/localplugins.conf
/usr/local/etc/pkg/repos/localplugins.conf             100  B 2122 kBps    00s
root@firewall:~ # pkg update
Updating OPNsense repository catalogue...
OPNsense repository is up to date.
Updating opnsense-extras repository catalogue...
Fetching data.pkg: 100%    3 KiB   2.7kB/s    00:01
Processing entries: 100%
opnsense-extras repository update completed. 9 packages processed.
All repositories are up to date.
root@firewall:~ #
```

We'll be using AdGuard home to act as the DNS sinkhole for the home lab. While it can be hosted on a separate device or VM, we'll be installing it directly within OPNsense as a plugin from a community maintained repository.

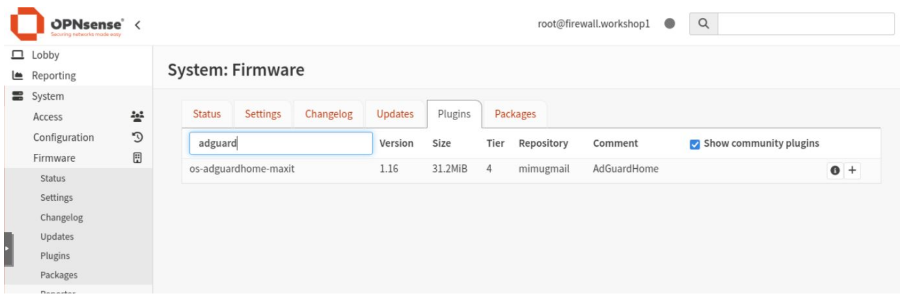
Why AdGuard as the sinkhole instead of the native OPNsense Unbound? Although Unbound has a blocklist and allowlist feature, it does not have a way to block *everything* by default and only permit-by-exception. If you are used to how Pi-hole works, that can also be used as a sinkhole, but for the purposes of this workshop, we're demonstrating the sinkhole capability with AdGuard because it can be installed as a plugin within the OPNsense interface itself and run without setting up an additional device or VM.

The plugins are provided by @mimugmail: https://www.routerperformance.net/opnsense-repo/ and mirrored for this workshop so we don't hammer their repo. In the real world, the command you'll want to use to add the repository is:

```
fetch -o /usr/local/etc/pkg/repos/mimugmail.conf
https://www.routerperformance.net/mimugmail.conf
```

OPNsense Setup (AdGuard)

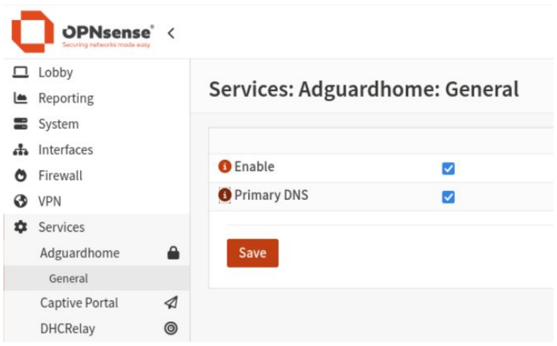
- From the OPNsense web UI, go to System -> Firmware -> Plugins
- Check “Show community plugins”, then search for “adguard”
- Click the + symbol to install AdGuard, then click Install to confirm on the popup



adguard	Version	Size	Tier	Repository	Comment
os-adguardhome-maxit	1.16	31.2MiB	4	mimugmail	AdGuardHome

OPNsense Setup (AdGuard)

- After the plugin installs, refresh the web UI, and go to Services -> Adguardhome
- Enable both the “Enable” option and the “Primary DNS” options

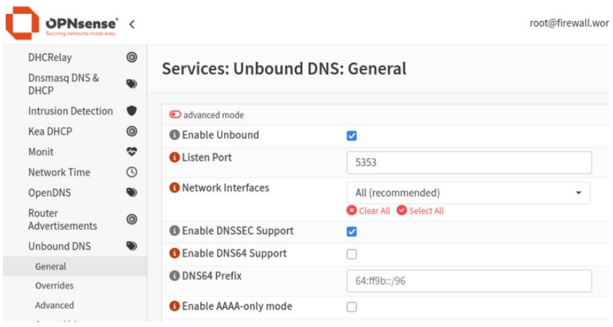


Services: Adguardhome: General

Enable	✓
Primary DNS	✓

OPNsense Setup (Unbound)

- Go to Services -> Unbound DNS -> General
- Change the port to 5353 and enable DNSSEC
- Scroll to the bottom and hit Apply



Services: Unbound DNS: General

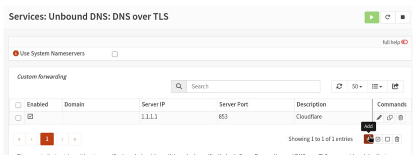
advanced mode

Enable Unbound	✓
Listen Port	5353
Network Interfaces	All (recommended)
Enable DNSSEC Support	✓
Enable DNS64 Support	☐
DNS64 Prefix	64:ff9b::/96
Enable AAAA-only mode	☐

For the sinkhole setup, AdGuard will only be the DNS sinkhole, and all upstream resolution requests will be handled by the OPNsense system's Unbound DNS resolver. Before setting AdGuard up, we want to set Unbound up to not accept requests from anywhere other than itself (preventing sinkhole bypass) and enable both DNSSEC and DNS over TLS for security.

OPNsense Setup (Unbound)

- Go to Services -> Unbound DNS -> DNS over TLS
- Click Add in the bottom right, then use the following settings:
  - Enabled: check this box
  - Server IP: 1.1.1.1
  - Server Port: 853
  - Verify CN: cloudflare-dns.com
  - Description: Cloudflare
- Hit Apply



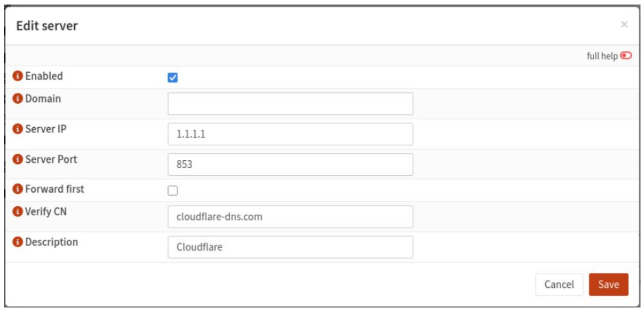
Services: Unbound DNS: DNS over TLS

Use System Nameservers ☐

Custom forwarding

☐	Enabled	Domain	Server IP	Server Port	Description	Commands
☐	✓		1.1.1.1	853	Cloudflare	

Showing 1 to 1 of 1 entries



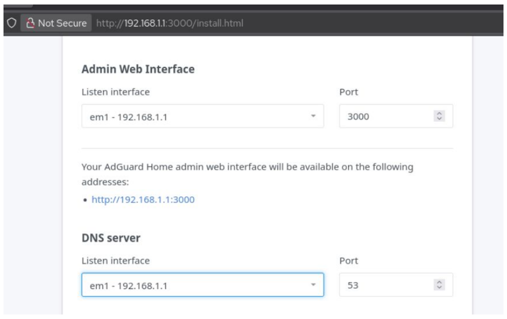
Edit server

Enabled	✓
Domain	
Server IP	1.1.1.1
Server Port	853
Forward first	☐
Verify CN	cloudflare-dns.com
Description	Cloudflare

Next, we'll configure Unbound to use a DNS over TLS server for all upstream requests. As discussed earlier, this prevents a man in the middle from potentially intercepting or monitoring DNS requests. There are other providers for DNS over TLS, but for the purposes of this workshop, we'll use Cloudflare since we can generally rely on it being online.

OPNsense Setup (AdGuard)

- Go to the address http://192.168.#.1:3000
- Start the setup process, and on the first screen, set the following:
    - Admin Web Interface
      Listen Interface: em1 - 192.168.#.1
      Port: 3000
    - DNS server
      Listen Interface: em1 - 192.168.#.1
      Port: 53
- # is your workshop ID


Not Secure http://192.168.1.1:3000/install.html

Admin Web Interface

Listen interface: em1 - 192.168.1.1
Port: 3000

Your AdGuard Home admin web interface will be available on the following addresses:
- http://192.168.1.1:3000

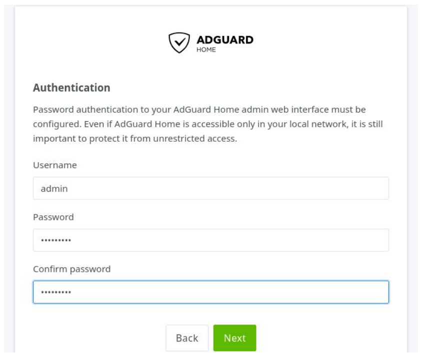
DNS server

Listen interface: em1 - 192.168.1.1
Port: 53

Now that Unbound is configured to run on a different port, we'll set up AdGuard Home. Since this is sharing the main interface as the rest of OPNsense, it needs to stay on its own port. For the DNS server itself, we want it to listen on 53, but only on the LAN interface.

OPNsense Setup (AdGuard)

- Set your username and password
  - Recommend that you use a different set of credentials than what you use to log into OPNsense
  - For this lab, okay to use the same thing for time convenience
- Confirm the next two screens to get to the main dashboard
- Log in with the credentials you just created
- (Side note: If the AdGuard Home page prompts for an update after setup, do NOT update, this will hammer their download servers since we don't have a mirror for AdGuard Home updates.)



Authentication

Password authentication to your AdGuard Home admin web interface must be configured. Even if AdGuard Home is accessible only in your local network, it is still important to protect it from unrestricted access.

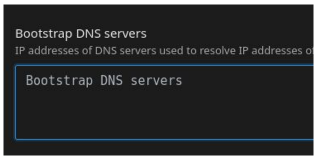
Username: admin

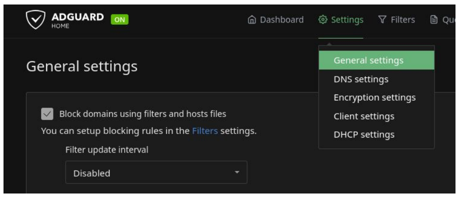
Password: ••••••••••

Confirm password: ••••••••••

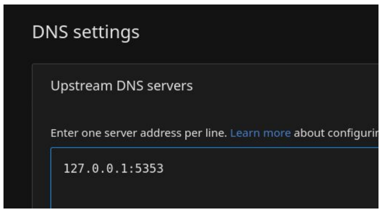
OPNsense Setup (Sinkholing)

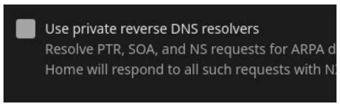
- From the AdGuard home page, go to Settings -> General
    - Set Filter update interval to “Disabled”, then click Save at the bottom
- Next, go to Settings -> DNS settings
    - Set Upstream DNS servers to be: 127.0.0.1:5353
    - Clear the Bootstrap DNS servers list
    - Uncheck the “Use private reverse DNS resolvers” option
    - Click “Test upstreams”, then if it succeeds, click Apply


Bootstrap DNS servers
IP addresses of DNS servers used to resolve IP addresses o
Bootstrap DNS servers



General settings
☑ Block domains using filters and hosts files
You can setup blocking rules in the Filters settings.
Filter update interval
Disabled


DNS settings
Upstream DNS servers
Enter one server address per line. Learn more about configurin
```
127.0.0.1:5353
```

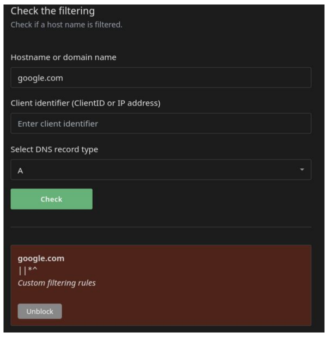

☐ Use private reverse DNS resolvers
Resolve PTR, SOA, and NS requests for ARPA d
Home will respond to all such requests with N

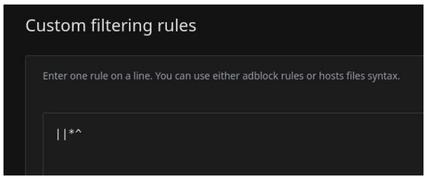
As we talked about earlier, the primary purpose of AdGuard will be to act as a DNS sinkhole, that is, by default, no DNS resolution requests will be allowed out. We’ll want to create a baseline of what good, known DNS requests should be allowed, and build those rules on top of the deny-by-default resolution filters. To supplement this, we’ll keep DNS logging enabled on AdGuard, since it has a convenient dashboard to monitor and ensure requests are being blocked, and also to determine what your home lab needs to have whitelisted.

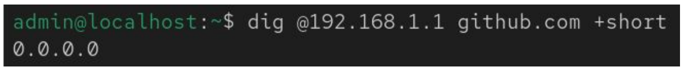
For AdGuard’s DNS settings, we’ll point it to the Unbound DNS resolver built into OPNsense as the only upstream, since we’ll want Unbound to handle the actual requests, as well as any DNS-over-TLS and DNSSEC. This also requires removing the default servers set in AdGuard’s bootstrap DNS server setting, as well as disabling private DNS resolution requests to prevent potential leakage.

OPNsense Setup (Sinkholing)

- Go to Filters -> DNS blocklists, and delete all filter lists currently installed
- Next, go to Filters -> Custom filtering rules
  - Add this rule to the list, then hit apply: ||*^
    - This rule tells AdGuard to block everything
  - Use the tool at the bottom of this page to test the rules and make sure every domain is blocked
  - For extra sanity checking, you can do a manual DNS query from your desktop to verify that any DNS request is blocked


Check the filtering
Check if a host name is filtered.
Hostname or domain name: google.com
Client identifier (ClientID or IP address): Enter client identifier
Select DNS record type: A
Check
google.com
||*^
Custom filtering rules
Unblock


Custom filtering rules
Enter one rule on a line. You can use either adblock rules or hosts files syntax.
```
||*^
```


```
admin@localhost:~$ dig @192.168.1.1 github.com +short
0.0.0.0
```

Next step in setting up the DNS sinkhole is to remove the AdGuard filter lists. Since it'll be blocking everything by default, there's no need to have ad domain lists installed.

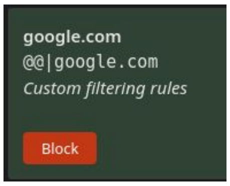
The filter rule to block *everything* with AdGuard is simple, a single glob match will block every domain name request from your LAN. We'll then add rules back in to allow only the domains we want to permit. Order in the list doesn't matter since a permit rule for a specific domain later in the custom filtering rules will still allow that domain to be resolved even if the first rule blocks everything by default.

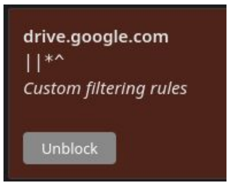
The AdGuard settings page for this rule describes how to unblock a specific domain, but the `@@||domain.com^` rule would unblock every subdomain of domain.com. Instead, if you want to make more granular rules to only allow the base domain or a specific subdomain *only*, you can use the following:
```
@@|domain.com^
@@|subomain.domain.com^
```
The first rule will only allow resolution of strictly the domain.com A record only, while the second will only allow subdomain.domain.com's A record to resolve, while not allowing domain.com to resolve (assuming only one of the two above rules is set). The key here is to use a single pipe (|) character, a double pipe (||) like in the example rule matches on all subdomains of the domain you're trying to whitelist.

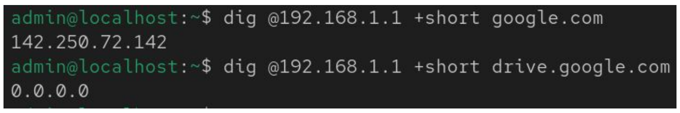
For more information, check https://adguard-dns.io/kb/general/dns-filtering-syntax/ under the Adblock-style syntax section.

OPNsense Setup (Sinkhole Whitelist)

- Since your VMs are using local package repos, this will just be an example to test whitelisting domains
- Under the catch-all rule, add a new rule: `@@|google.com`
  - This will allow *only* google.com to resolve, but not any of its subdomains (like drive.google.com)
  - See the previous slide's notes for more details


google.com
@@|google.com
Custom filtering rules


drive.google.com
||*^
Custom filtering rules


```
admin@localhost:~$ dig @192.168.1.1 +short google.com
142.250.72.142
admin@localhost:~$ dig @192.168.1.1 +short drive.google.com
0.0.0.0
```

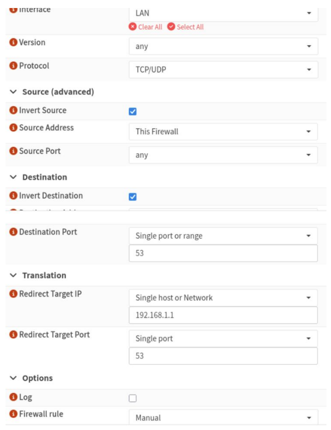
OPNsense Setup (NAT Rules)

- Now that the DNS sinkhole is set up, we want to make sure LAN clients can only use OPNsense and AdGuard as the source for DNS requests
- There are multiple ways to set this up but for this workshop, we'll show you how to do it with NAT
  - It's possible to also use firewall rules to block outbound DNS
- NAT will essentially allow the firewall to "intercept" all outbound DNS requests from the LAN and use AdGuard to resolve these requests

Next, we'll want to make sure the LAN clients can only use OPNsense for DNS. While you'll also be adding permit-by-exception rules for outbound traffic in general, it's still possible that a permitted outbound address allows DNS. The best way to keep DNS within the network has two steps, and the first is to use destination NAT. Network Address Translation (NAT) will rewrite any DNS requests so that they essentially intercepted by OPNsense and sent to AdGuard, then sent back to the LAN client, all transparently.

OPNsense Setup (NAT Rules)

- On the OPNsense web UI, go to Firewall -> NAT -> Destination NAT
- Add a new rule with the following:
  - Interface: LAN
  - Protocol: TCP/UDP
  - Source Address: This Firewall (and check Invert Source)
  - Source Port: any
  - Destination Address: This Firewall (and check Invert Source)
  - Destination Port: 53
  - Redirect Target IP: Single host or Network, 192.168.#.1 (where # is your workshop ID)
  - Redirect Target Port: 53
- Save, then apply the settings

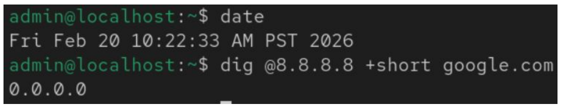


Interface	LAN
Version	any
Protocol	TCP/UDP
Source (advanced)	
Invert Source	✓
Source Address	This Firewall
Source Port	any
Destination	
Invert Destination	✓
Destination Port	Single port or range; 53
Translation	
Redirect Target IP	Single host or Network; 192.168.1.1
Redirect Target Port	Single port; 53
Options	
Log	☐
Firewall rule	Manual

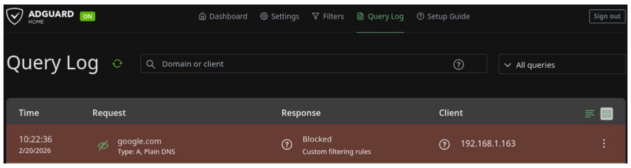
The Destination NAT rule (formerly Port Forwarding) in OPNsense will allow the firewall to essentially rewrite any outbound requests to DNS servers to instead point to AdGuard listening on OPNsense's IP on port 53 instead. This happens transparently to the LAN client requesting DNS, and it will still resolve host names as if nothing is wrong.

OPNsense Setup (NAT Rules)

- To test that the rule works, make a DNS request to a known public DNS server for a domain that you have not whitelisted
  - `dig @8.8.8.8 +short google.com`
- In the AdGuard web UI, go to the Query Log, and you can verify that the DNS request came through AdGuard instead of going to 8.8.8.8



```
admin@localhost:~$ date
Fri Feb 20 10:22:33 AM PST 2026
admin@localhost:~$ dig @8.8.8.8 +short google.com
0.0.0.0
```



Time	Request	Response	Client
10:22:36 2/20/2026	google.com Type: A, Plain DNS	Blocked Custom filtering rules	192.168.1.163

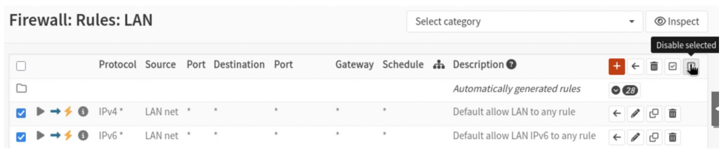
Now that the NAT rule is in place, we'll want to make sure it's actually in place and "intercepting" DNS requests.

OPNsense Setup (Firewall Rules)

- Next, we’ll set up outbound rules for the LAN to only allow traffic to the allow list
- Side note, for your own home lab, you’ll likely need to add many more domains, such as public package repos (Fedora, Rocky, etc.) or Let’s Encrypt domains for automatic certificate provisioning
- From the OPNsense guide, firewall rules for in and out can be best described as:
  - Traffic IN is coming into the firewall interface, while traffic OUT is going out of the firewall interface. In visual terms: [Source] -> IN -> [Firewall] -> OUT -> [Destination].
- Start by going to the Firewall -> Rules -> LAN Page

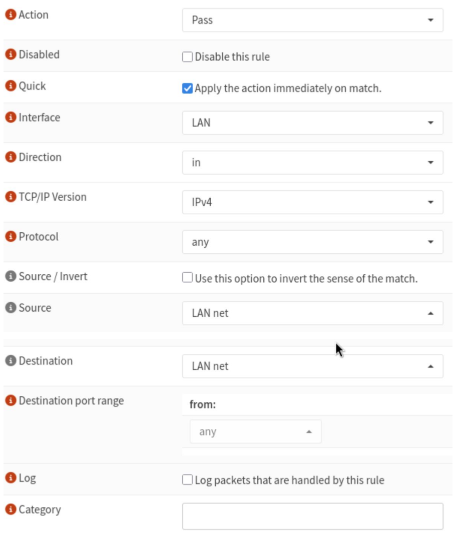
OPNsense Setup (Firewall Rules)

- First, we’ll want to disable the default allow LAN rules
    - Don’t hit apply yet!
- Next, add a rule allowing any LAN net to LAN net traffic
- Apply the rules and make sure you can still connect to the OPNsense web UI



Firewall: Rules: LAN

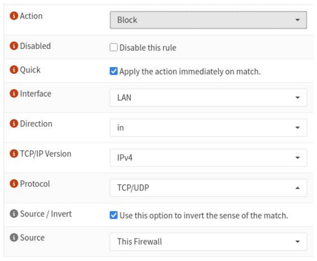
		Protocol	Source	Port	Destination	Port	Gateway	Schedule	Description
☐									
									Automatically generated rules 28
✓	▶ → ⚡ ⓘ	IPv4 *	LAN net	*	*	*	*	*	Default allow LAN to any rule
✓	▶ → ⚡ ⓘ	IPv6 *	LAN net	*	*	*	*	*	Default allow LAN IPv6 to any rule



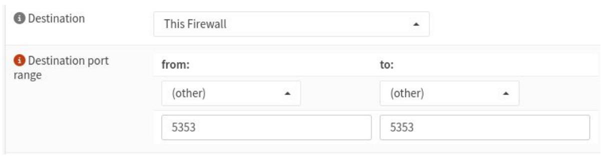
Action	Pass
Disabled	☐ Disable this rule
Quick	✓ Apply the action immediately on match.
Interface	LAN
Direction	in
TCP/IP Version	IPv4
Protocol	any
Source / Invert	☐ Use this option to invert the sense of the match.
Source	LAN net
Destination	LAN net
Destination port range	from: any
Log	☐ Log packets that are handled by this rule
Category	

OPNsense Setup (Firewall Rules)

- Add a new rule to block any IPs other than the firewall itself from using 5353:
  - Action: Block
  - Protocol: TCP/UDP
  - Source: This Firewall (and check Source / Invert)
  - Destination: This Firewall
  - Destination port range: Select (other), then 5353 to 5353



Action	Block
Disabled	☐ Disable this rule
Quick	✓ Apply the action immediately on match.
Interface	LAN
Direction	in
TCP/IP Version	IPv4
Protocol	TCP/UDP
Source / Invert	✓ Use this option to invert the sense of the match.
Source	This Firewall

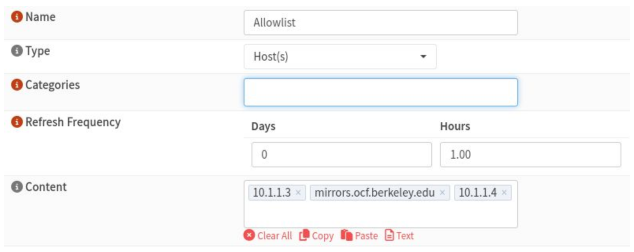


	from:	to:
Destination	This Firewall	
Destination port range	(other)	(other)
	5353	5353

We want to make sure LAN clients can't bypass AdGuard/the DNS sinkhole, so only allow the firewall itself to resolve domains on port 5353 with Unbound.

OPNsense Setup (Firewall Rules)

- Go to Firewall -> Aliases, and click Add
  - Name: Allowlist
  - Type: Host(s)
  - Refresh Frequency: 1 hour
  - Content: 10.1.1.3 and 10.1.1.4 (for the local workshop repos) and mirrors.ocf.berkeley.edu (for testing purposes)
- Scroll to the bottom and click Apply

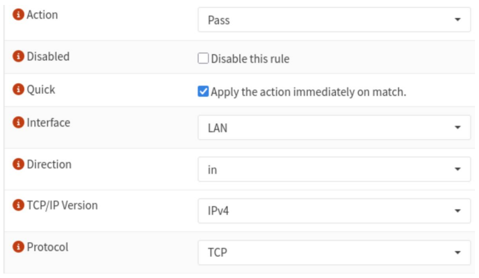


Name	Allowlist
Type	Host(s)
Categories	
Refresh Frequency	Days: 0 Hours: 1.00
Content	10.1.1.3, mirrors.ocf.berkeley.edu, 10.1.1.4

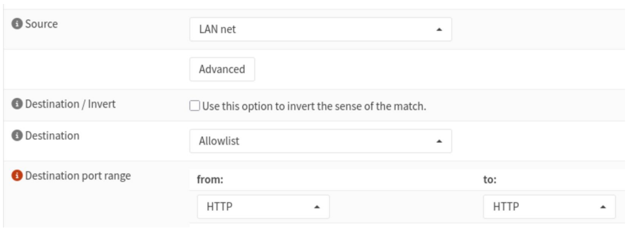
The Aliases feature of OPNsense allows you to allowlist websites based on their domain name. The firewall will automatically refresh IPs of domain names and when inserted into the allowlist firewall rule, will allow that domain name without needing to manually add the IP it resolves to. This is especially necessary for any domain names with load balancing and IPs that change frequently. The refresh frequency can also be set to a smaller interval if you need to frequently re-resolve a domain name to an IP.

OPNsense Setup (Firewall Rules)

- Go back to the LAN firewall rules, and add two new rules:
  - Protocol: TCP
  - Source: LAN net
  - Destination: Allowlist
  - Destination port range: HTTP to HTTP for the first rule, HTTPS to HTTPS for the second



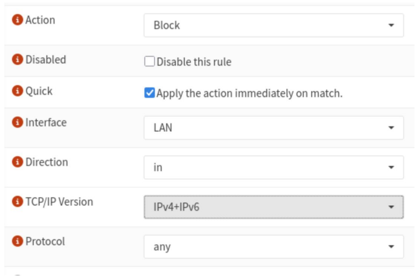
Action	Pass
Disabled	☐ Disable this rule
Quick	✓ Apply the action immediately on match.
Interface	LAN
Direction	in
TCP/IP Version	IPv4
Protocol	TCP



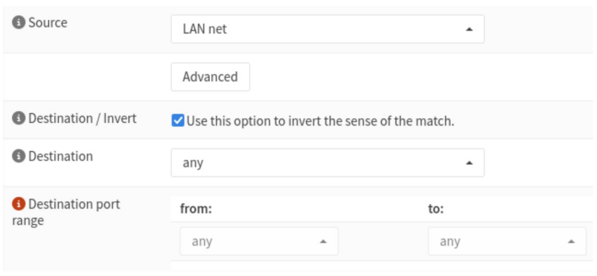
Source	LAN net	
	Advanced	
Destination / Invert	☐ Use this option to invert the sense of the match.	
Destination	Allowlist	
Destination port range	from: HTTP	to: HTTP

OPNsense Setup (Firewall Rules)

- For the final rule, we'll add the block everything else rule:
    - Action: Block
    - TCP/IP Version: IPv4+IPv6
    - Source: LAN net
    - Destination: any



Action	Block
Disabled	☐ Disable this rule
Quick	☑ Apply the action immediately on match.
Interface	LAN
Direction	in
TCP/IP Version	IPv4+IPv6
Protocol	any



Source	LAN net	
	Advanced	
Destination / Invert	☑ Use this option to invert the sense of the match.	
Destination	any	
Destination port range	from: any	to: any

Fedora CoreOS, Docker, and Docker Compose

Section 4

Fedora CoreOS

- Distro designed for containers
- Uses “read-only” images for updates
  - Robust update system
  - Collects and modifies operating system image from Fedora image store, then reboots into it
  - If the collection or modification fails, it does not attempt to reboot
  - Able to easily switch back to an earlier operating system image if current one has any issues
- Uses a *butane* file (yaml) that defines the operating system, which is then converted into an *ignition* file (json) that the installer reads to configure your operating system
- A popular competitor is sidecar [1]

[1] https://www.flatcar.org/

Docker

- Containers using namespaces for compartmentalized service hosting
- Why Docker? Industry standard and convenient
  - Most services have a Docker image and a Docker Compose spec
  - If a Docker image “works on my machine” (i.e., the service’s developer), it will almost certainly work on yours
  - Adds some minimal layer of security and compartmentalization
- Competitors: Podman, BSD Jails
  - Almost no one uses Podman Desktop, even fewer people use Podman Quadlets, even fewer-er people use BSD Jails
  - While nearly all self-hosted hobby services come with Docker images and Docker Compose config files, it’s difficult to find any that support Podman Quadlets or BSD Jails
  - While Podman uses OCI images, its syntax is not fully compatible with Docker, and Red Hat is against supporting Docker Compose with Podman

Docker Compose

- "Docker Compose is a tool for defining and running multi-container applications." [1]
- Allows you to define a yaml file that can be used to create and remove a Docker image at will
- Allows for more streamlined updates
  - You can carve-out portions of the file system to be persistent while also being able to swap images quickly
- You can even save all your yaml files to a Git server, backup your mounts, and quickly recover from a complete OS failure

[1] https://docs.docker.com/compose/

Fedora CoreOS, NSA SELinux, and Docker Volumes

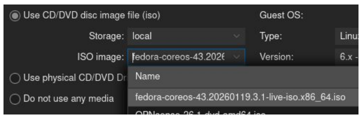
- If you’re using a system with SELinux, like Fedora CoreOS, when you set volumes, use Z (capital)
- This uses SELinux contexts to prevent other containers from reading the contents of a container’s volumes (the container breakout scenario)
- Ideally, you would have individual VMs per service to prevent this
- Many tutorials and guides use Docker volumes (not bind mounts, which also use the “volume” syntax)
  - For ease of editing config files, we’ll be using bind mounts for the labs in this workshop

Installing Fedora CoreOS

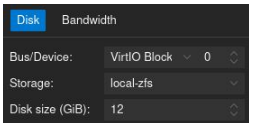
Lab 4.1

Create a CoreOS Virtual Machine

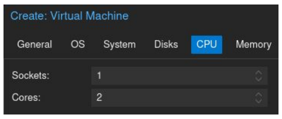
- In Proxmox, create a CoreOS virtual machine by selecting the Fedora CoreOS iso file as the installation image
- VM settings:
  - 12GiB of storage on “local-zfs” (our SSD)
  - 1 CPU socket, 2 CPU cores, type: host
  - 4096 MiB memory
  - Network -> Bridge -> wshop# where # is your workshop ID number



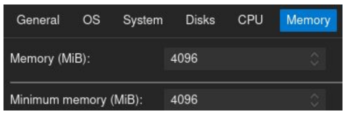
(●) Use CD/DVD disc image file (iso)
Storage: local
ISO image: fedora-coreos-43.2026
( ) Use physical CD/DVD Dr
( ) Do not use any media
Name
fedora-coreos-43.20260119.3.1-live-iso.x86_64.iso
Guest OS:
Type: Linu
Version: 6.x -



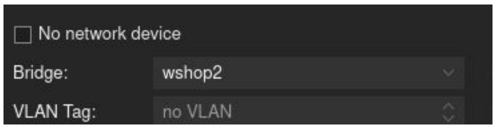
Disk | Bandwidth
Bus/Device: VirtIO Block 0
Storage: local-zfs
Disk size (GiB): 12



Create: Virtual Machine
General | OS | System | Disks | CPU | Memory
Sockets: 1
Cores: 2



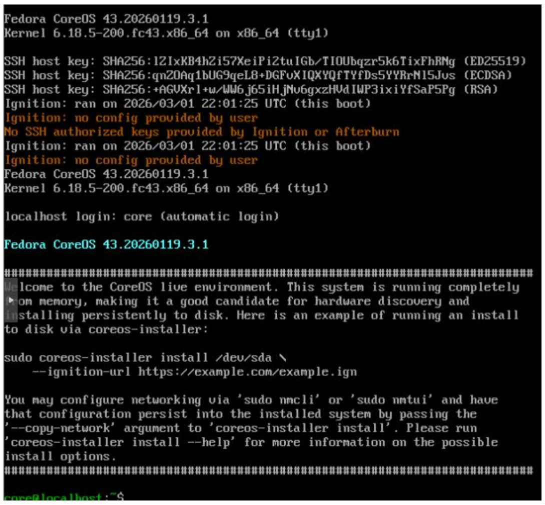
General | OS | System | Disks | CPU | Memory
Memory (MiB): 4096
Minimum memory (MiB): 4096



☐ No network device
Bridge: wshop2
VLAN Tag: no VLAN

Booting into Fedora CoreOS

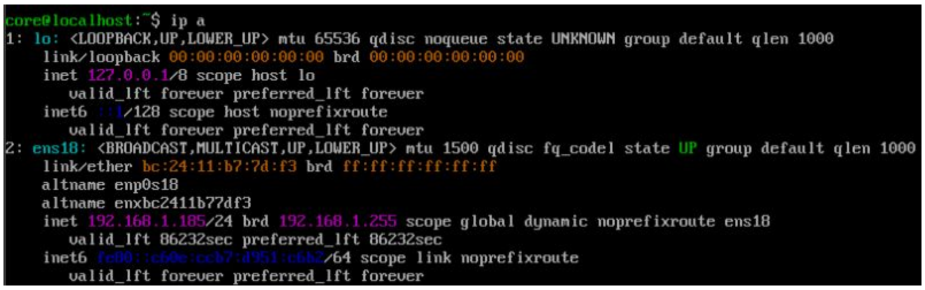
- Unlike most Linux distros, Fedora CoreOS has no graphical installer and requires a JSON configuration file (called “ignition”) to install
- Here, we will configure networking and test it before moving on with the ignition file installation



```
Fedora CoreOS 43.20260119.3.1
Kernel 6.18.5-200.fc43.x86_64 on x86_64 (tty1)

SSH host key: SHA256:lZIxKB4hZi57XeiPi2tuIGb/TIOUbqzr5k6TixFhRNg (ED25519)
SSH host key: SHA256:qnZOAq1bUG9qeL8+DGFvXIQXYQfTYfDs5YYRrN15Jvs (ECDSA)
SSH host key: SHA256:+AGVXrl+w/WW6j65iHjNv6gxzHVdIWP3ixiYfSaP5Pg (RSA)
Ignition: ran on 2026/03/01 22:01:25 UTC (this boot)
Ignition: no config provided by user
No SSH authorized keys provided by Ignition or Afterburn
Ignition: ran on 2026/03/01 22:01:25 UTC (this boot)
Ignition: no config provided by user
Fedora CoreOS 43.20260119.3.1
Kernel 6.18.5-200.fc43.x86_64 on x86_64 (tty1)

localhost login: core (automatic login)

Fedora CoreOS 43.20260119.3.1

############################################################################
Welcome to the CoreOS live environment. This system is running completely
on memory, making it a good candidate for hardware discovery and
installing persistently to disk. Here is an example of running an install
to disk via coreos-installer:

sudo coreos-installer install /dev/sda \
    --ignition-url https://example.com/example.ign

You may configure networking via 'sudo nmcli' or 'sudo nmtui' and have
that configuration persist into the installed system by passing the
'--copy-network' argument to 'coreos-installer install'. Please run
'coreos-installer install --help' for more information on the possible
install options.
############################################################################

core@localhost:~$
```

Setting a Static IP

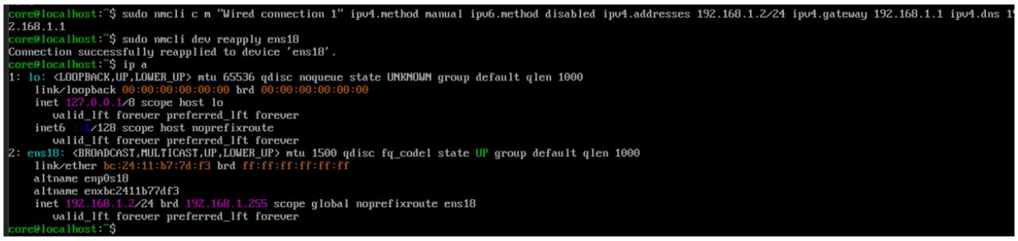
- For services, it’s useful to have known IPs and avoid setting up dynamic DNS
- One way to do this is via static IP addresses in the host; another, with reserved DHCP leases
    - We’ll use the former method
- First, list your network addresses so you can view what your network device is:



```
core@localhost:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:b7:7d:f3 brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    altname enxbc2411b77df3
    inet 192.168.1.185/24 brd 192.168.1.255 scope global dynamic noprefixroute ens18
       valid_lft 86232sec preferred_lft 86232sec
    inet6 fe80::c60e:ccb7:d951:c6b2/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Using nmcli to set a static IP

- Now that we know our network interface is ens18, use the following command to set a static IP address (where # is your workshop ID):
  `nmcli c m "Wired connection 1" ipv4.method manual ipv6.method disabled ipv4.addresses 192.168.#.2/24 ipv4.gateway 192.168.#.1 ipv4.dns 192.168.#.1`
- Then, load the config with `nmcli dev reapply ens18`



```
core@localhost:~$ sudo nmcli c m "Wired connection 1" ipv4.method manual ipv6.method disabled ipv4.addresses 192.168.1.2/24 ipv4.gateway 192.168.1.1 ipv4.dns 1
2.168.1.1
core@localhost:~$ sudo nmcli dev reapply ens18
Connection successfully reapplied to device 'ens18'.
core@localhost:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:b7:7d:f3 brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    altname enxbc2411b77df3
    inet 192.168.1.2/24 brd 192.168.1.255 scope global noprefixroute ens18
       valid_lft forever preferred_lft forever
core@localhost:~$
```

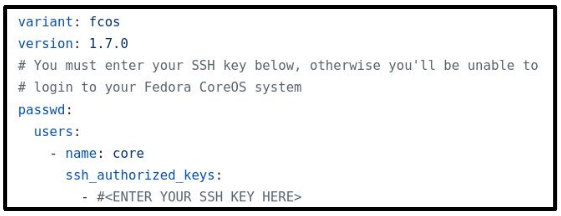
Butane Files

- Yaml files that define a Fedora CoreOS instance
- Multiple specifications exist; the latest at time of writing: v1.7.0 [1]
- Multiple examples of Butane configurations exist [2] [3], though we will focus on a few specific configurations to get us started and continue off manually
- To work with butane files, you will need the butane package
    - Already installed in your Rocky Desktop VM, but if you need to install it on a different host, simply use `sudo dnf install butane`
- Clone the Git repository for this workshop, which will contain the Butane file you will work with
    - git clone http://10.1.1.4:3000/henryreed/scale23x-homelab-workshop
    - Also hosted online: https://github.com/henryreed/scale23x-homelab-workshop
    - Please only use the GitHub link outside of the SCaLE network to avoid getting us IP blocked

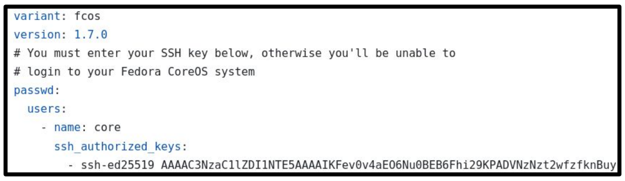
[1] https://coreos.github.io/butane/config-fcos-v1_7/
[2] https://coreos.github.io/butane/examples/
[3] https://docs.fedoraproject.org/en-US/fedora-coreos/producing-ign/

Butane Object: passwd

- The first section, passwd, defines an SSH key for the core user
  - By default, passwords are not set for core
- Before using our Butane file, create an ssh key and add it to the ssh_authorized_keys section so that it looks similar to the second image

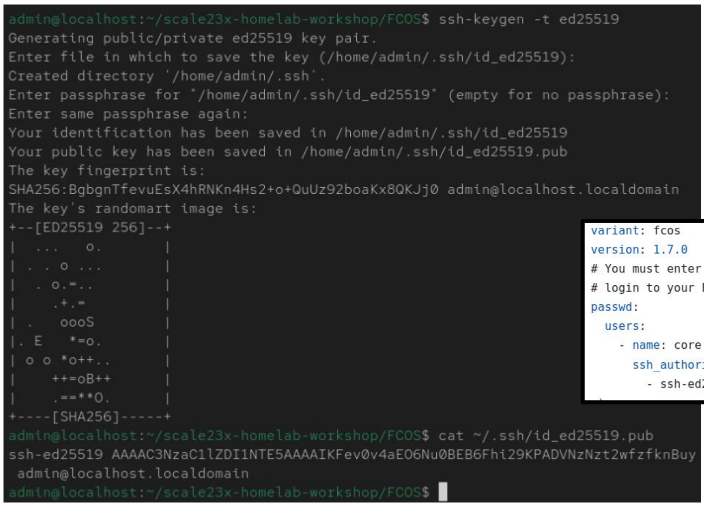


```
variant: fcos
version: 1.7.0
# You must enter your SSH key below, otherwise you'll be unable to
# login to your Fedora CoreOS system
passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - #<ENTER YOUR SSH KEY HERE>
```

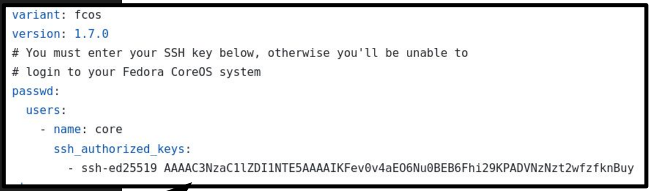


```
variant: fcos
version: 1.7.0
# You must enter your SSH key below, otherwise you'll be unable to
# login to your Fedora CoreOS system
passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKFev0v4aEO6Nu0BEB6Fhi29KPADVNzNzt2wfzfknBuy
```

An Aside: SSH key generation



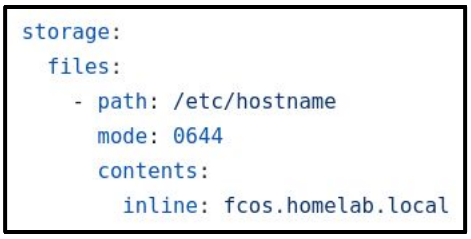
```
admin@localhost:~/scale23x-homelab-workshop/FCOS$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/admin/.ssh/id_ed25519):
Created directory '/home/admin/.ssh'.
Enter passphrase for "/home/admin/.ssh/id_ed25519" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/admin/.ssh/id_ed25519
Your public key has been saved in /home/admin/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:BgbgnTfevuEsX4hRNKn4Hs2+o+QuUz92boaKx8QKJj0 admin@localhost.localdomain
The key's randomart image is:
+--[ED25519 256]--+
|  ...   o.       |
| . . o ...       |
|  . o.=..        |
|    .+.=         |
| .   oooS        |
|. E   *=o.       |
| o o *o++..      |
|    ++=oB++      |
|    .==**O.      |
+----[SHA256]-----+
admin@localhost:~/scale23x-homelab-workshop/FCOS$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKFev0v4aEO6Nu0BEB6Fhi29KPADVNzNzt2wfzfknBuy
 admin@localhost.localdomain
admin@localhost:~/scale23x-homelab-workshop/FCOS$
```



```
variant: fcos
version: 1.7.0
# You must enter your SSH key below, otherwise you'll be unable to
# login to your Fedora CoreOS system
passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKFev0v4aEO6Nu0BEB6Fhi29KPADVNzNzt2wfzfknBuy
```

Butane Object: storage

- Storage allows the operator to define disks, partitions, file systems, and files on the file system
- In our case, we simply define the hostname file



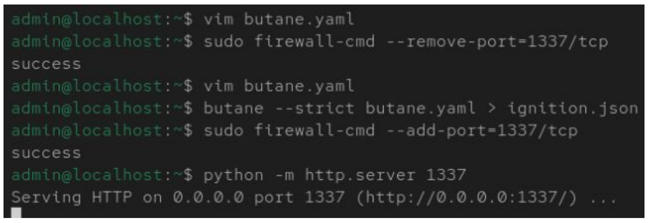
```
storage:
  files:
    - path: /etc/hostname
      mode: 0644
      contents:
        inline: fcos.homelab.local
```

Butane Object: systemd

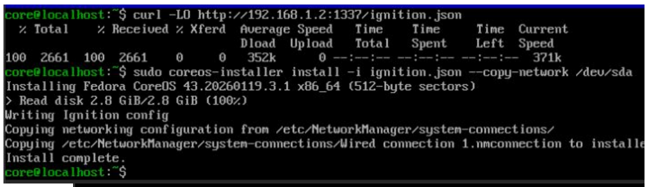
- The systemd component defines systemd unit files on the system
- This is also how we remove and install packages
    - rpm-ostree: the tool that creates an operating system image
    - zincati: the service that fetches OS images and reboots into them for updates
- In our case, we remove the pre-installed non-community edition version of Docker and instead install community editions of Docker and Docker Compose

Converting the Butane File

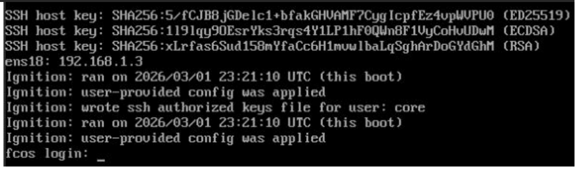
- After adding your SSH key, convert the Butane file to an Ignition file:
  - butane --strict butane.yaml > ignition.json
- Then, use Python's web server to host the ignition file and allow your Fedora CoreOS instance to collect it via curl
- After the installation finishes, reboot the system with `sudo reboot`
- You'll get to a login screen. After the second installation systemd unit file runs, the system will restart for a final time with Docker CE installed



```
admin@localhost:~$ vim butane.yaml
admin@localhost:~$ sudo firewall-cmd --remove-port=1337/tcp
success
admin@localhost:~$ vim butane.yaml
admin@localhost:~$ butane --strict butane.yaml > ignition.json
admin@localhost:~$ sudo firewall-cmd --add-port=1337/tcp
success
admin@localhost:~$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```



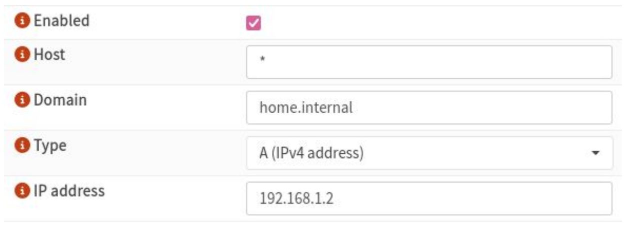
```
core@localhost:~$ curl -LO http://192.168.1.2:1337/ignition.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2661  100  2661    0     0   352k      0 --:--:-- --:--:-- --:--:--  371k
core@localhost:~$ sudo coreos-installer install -i ignition.json --copy-network /dev/sda
Installing Fedora CoreOS 43.20260119.3.1 x86_64 (512-byte sectors)
> Read disk 2.8 GiB/2.8 GiB (100%)
Writing Ignition config
Copying networking configuration from /etc/NetworkManager/system-connections/
Copying /etc/NetworkManager/system-connections/Wired connection 1.nmconnection to installe
Install complete.
core@localhost:~$
```



```
SSH host key: SHA256:5/fCJB8jGDelc1+bfakGHVAMF7CygIcpfEz4vpWVPU0 (ED25519)
SSH host key: SHA256:1191qy9OEsrYks3rqs4Y1LP1hF0QWn8F1VyCoHvUDwM (ECDSA)
SSH host key: SHA256:xLrfas6Sud158mYfaCc6H1mvwlbaLqSghArDoGYdGhM (RSA)
ens18: 192.168.1.3
Ignition: ran on 2026/03/01 23:21:10 UTC (this boot)
Ignition: user-provided config was applied
Ignition: wrote ssh authorized keys file for user: core
Ignition: ran on 2026/03/01 23:21:10 UTC (this boot)
Ignition: user-provided config was applied
fcos login: _
```

Additional Step (DNS)

- We'll want to ensure all your "LAN" hosts can resolve the Core OS services, so we'll set up an explicit DNS entry to resolve `home.internal` and any subdomains
- In the OPNsense web UI, navigate to Services -> Unbound DNS -> Overrides
- Add an entry in the Hosts section
  - Host: *
  - Domain: home.internal
  - Type: A (IPv4 address)
  - IP address: 192.168.#.2 (where # is your workshop ID)



Enabled	✓
Host	*
Domain	home.internal
Type	A (IPv4 address)
IP address	192.168.1.2

Additional Step (Docker Daemon)

- Before we continue, SSH into your newly set up Core OS box and run the following command:
  - `sudo systemctl enable docker`
- This ensures your Docker containers and services will restart automatically after a reboot
  - In general, you'll set containers to restart unless stopped in the Docker compose file

Installing Step CA

Lab 4.2

Step CA

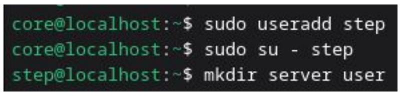
- step-ca is a simple certificate authority server, typically used for automated ACME certificate enrollment
- We'll be using Step CA as our CA to issue user TLS certificates for authentication and mTLS, as well as a CA for server TLS certificates
- In this lab section, we'll focus on installing Step CA and using it to issue TLS certs for the Caddy web server to proxy your self hosted service
    - The next section's labs will show you how to use Step CA and Caddy to do mTLS and user authentication

Step CA Security Considerations

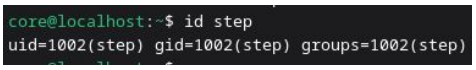
- Ideally on your own home lab you would have the CA on a dedicated VM
- If you have the resources, use a hardware device like a Yubikey as secure storage for the private keys: https://smallstep.com/docs/step-ca/cryptographic-protection/#yubikey-piv
- For this workshop, the CA private key passwords will remain in plain text
    - If you have to host multiple services on one host, the next slide will demonstrate a way to separate services using SELinux and non-privileged service accounts

Step CA Security Considerations

- Create a new user called “step”:
    - `sudo useradd step`
- Note the user’s user ID and group ID numbers:
    - `id step`
- Change to the step user to set up directories for the Docker containers:
    - `sudo su - step`
    - `mkdir server user`



```
core@localhost:~$ sudo useradd step
core@localhost:~$ sudo su - step
step@localhost:~$ mkdir server user
```

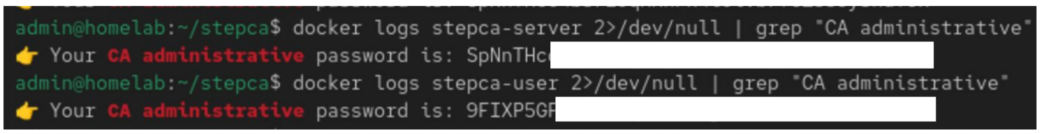


```
core@localhost:~$ id step
uid=1002(step) gid=1002(step) groups=1002(step)
```

This is a mitigation of the idea discussed in the previous slide. In theory, if your hosting resources are limited, and you must serve multiple services on a single host, create a separate service user for each service. These users don’t need any sudo privileges and exist solely to store the service configuration files isolated naturally via Linux file permissions and SELinux confines. While this doesn’t eliminate the potential for compromise completely, in an absolute worst case scenario where an attacker is able to compromise one Dockerized service on the same host, it greatly decreases the chance they can read critical secrets or configuration files from another service.

Step CA (Initialization)

- As the `core` user, grab the premade Docker Compose file for Step CA
  - Make a new directory in the core user’s home directory called “step”
  - Set the UID and GID for the step user you just created in the compose file
- Run docker compose up
  - If there are no errors, hit d to detach
  - Check the log output to save the CA administrative password:
    `docker logs stepca-server 2>/dev/null | grep “CA administrative”`
    `docker logs stepca-user 2>/dev/null | grep “CA administrative”`



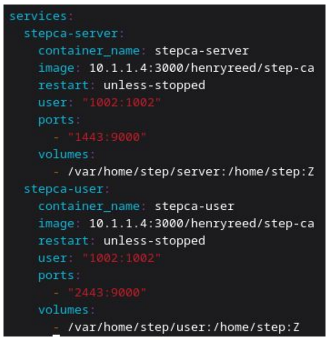
```
admin@homelab:~/stepca$ docker logs stepca-server 2>/dev/null | grep "CA administrative"
👉 Your CA administrative password is: SpNnTHc[redacted]
admin@homelab:~/stepca$ docker logs stepca-user 2>/dev/null | grep "CA administrative"
👉 Your CA administrative password is: 9FIXP5GF[redacted]
```

For your own home lab, you’ll want to store this password safely in a password manager. This password is different from the CA private key password stored on disk, since step-ca abstracts away direct key access by the concept of provisioning users. This is the password for the provisioning user, username “admin” by default. (The default username can be changed on init by setting `DOCKER_STEPCA_INIT_PROVISIONER_NAME` in the Docker compose file before first run.)

For this workshop, you’ll be using the password later to generate an SSH certificate and TLS certificates for your services.

Step CA (Additional Steps)

- Edit the Docker compose file to remove the environment section
  - These are only needed for initialization
- Run `docker compose up -d` to recreate the container

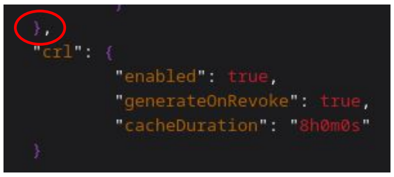


```
services:
  stepca-server:
    container_name: stepca-server
    image: 10.1.1.4:3000/henryreed/step-ca
    restart: unless-stopped
    user: "1002:1002"
    ports:
      - "1443:9000"
    volumes:
      - /var/home/step/server:/home/step:Z
  stepca-user:
    container_name: stepca-user
    image: 10.1.1.4:3000/henryreed/step-ca
    restart: unless-stopped
    user: "1002:1002"
    ports:
      - "2443:9000"
    volumes:
      - /var/home/step/user:/home/step:Z
```

Step CA (CRL Setup)

- We will turn on Certificate Revocation List (CRL) generation in step-ca
- Edit the step-ca config in `~/step/{server,user}/config/ca.json`
- At the bottom level of the JSON, add the following:

```
"crl": {
        "enabled": true,
        "generateOnRevoke": true,
        "cacheDuration": "8h0m0s"
}
```



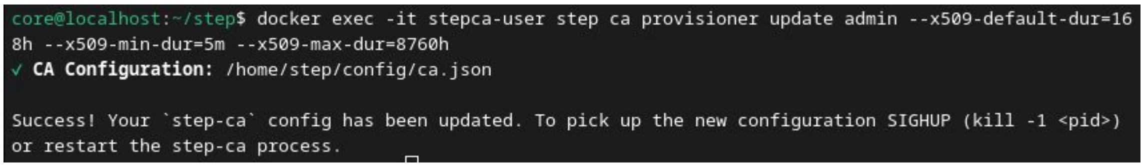
```
},
"crl": {
        "enabled": true,
        "generateOnRevoke": true,
        "cacheDuration": "8h0m0s"
}
```

Don't forget to add a comma on the previous closed JSON block!

Caddy web server does not support CRL checking, so we will be turning on CRL generation in step-ca for use with Keycloak. This ensures users with expired certs can no longer authenticate to the service.

Step CA (Certificate Expiration)

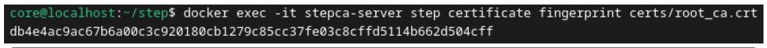
- Step CA has short lifetimes on certificates (by design)
- However, for your home lab, you won't want to regenerate user certs every day
- For the user CA, run the following command, then `docker compose restart`:
  - `docker exec -it stepca-user step ca provisioner update admin --x509-default-dur=168h --x509-min-dur=5m --x509-max-dur=8760h`
- This will make the maximum lifetime of a cert 1 year (8760 hours) for users



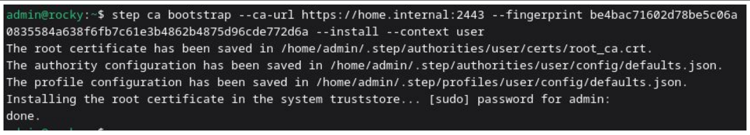
```
core@localhost:~/step$ docker exec -it stepca-user step ca provisioner update admin --x509-default-dur=16
8h --x509-min-dur=5m --x509-max-dur=8760h
✓ CA Configuration: /home/step/config/ca.json

Success! Your `step-ca` config has been updated. To pick up the new configuration SIGHUP (kill -1 <pid>)
or restart the step-ca process.
```

Step CA Management

- The client portion of step-ca, step-cli needs to be bootstrapped
- On CoreOS , from the `~/step` directory, run the following commands:
  - `docker exec -it stepca-server step certificate fingerprint certs/root_ca.crt`
- From your Rocky VM, install step-cli: `sudo dnf install step-cli`
  - `step ca bootstrap --ca-url https://home.internal:2443 --fingerprint (output from above cmd) --install --context server`
- Do the same for the User CA server (`userca.home.internal`)



```
core@localhost:~/step$ docker exec -it stepca-server step certificate fingerprint certs/root_ca.crt
db4e4ac9ac67b6a00c3c920180cb1279c85cc37fe03c8cffd5114b662d504cff
```



```
admin@rocky:~$ step ca bootstrap --ca-url https://home.internal:2443 --fingerprint be4bac71602d78be5c06a
0835584a638f6fb7c61e3b4862b4875d96cde772d6a --install --context user
The root certificate has been saved in /home/admin/.step/authorities/user/certs/root_ca.crt.
The authority configuration has been saved in /home/admin/.step/authorities/user/config/defaults.json.
The profile configuration has been saved in /home/admin/.step/profiles/user/config/defaults.json.
Installing the root certificate in the system truststore... [sudo] password for admin:
done.
```

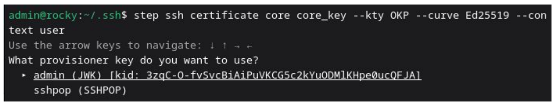
Step-cli is used to manage the Step CA functions, like issuing certificates, which we'll do from the Rocky VM. This also installs the root CA into the system truststore so your system will now be able to browse to self hosted sites without a certificate warning.

Step CA and SSH

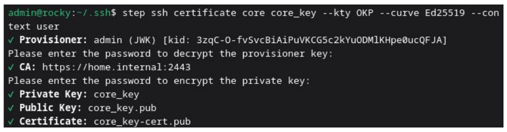
- Our user CA can generate certs for SSH authentication, to test this, we'll generate a cert to SSH from Rocky to CoreOS
- On the CoreOS box as root, copy `/var/home/step/user/certs/ssh_user_ca_key.pub` to `/etc/ssh`
- Create a new file in `/etc/ssh/sshd_config.d` called `10-stepca.conf`
- Insert the following line:
  - `TrustedUserCAKeys /etc/ssh/ssh_user_ca_key.pub`
- Restart the SSH daemon with: `sudo systemctl restart sshd`

Step CA and SSH

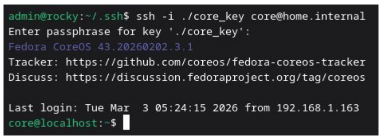
- From the Rocky VM, go to the `~/.ssh` directory
- Run the command `step ssh certificate core id_ed25519 --kty OKP --curve Ed25519 --context user --not-after=8766h`
- Select the provisioner key for admin
  - When prompted, use the user CA administrative password you saved earlier
- Set a password for the private key when prompted



```
admin@rocky:~/.ssh$ step ssh certificate core core_key --kty OKP --curve Ed25519 --con
text user
Use the arrow keys to navigate: ↓ ↑ → ←
What provisioner key do you want to use?
  ▸ admin (JWK) [kid: 3zqC-O-fvSvcBiAiPuVKCG5c2kYuODMlKHpe0ucQFJA]
    sshpop (SSHPOP)
```



```
admin@rocky:~/.ssh$ step ssh certificate core core_key --kty OKP --curve Ed25519 --con
text user
✓ Provisioner: admin (JWK) [kid: 3zqC-O-fvSvcBiAiPuVKCG5c2kYuODMlKHpe0ucQFJA]
Please enter the password to decrypt the provisioner key:
✓ CA: https://home.internal:2443
Please enter the password to encrypt the private key:
✓ Private Key: core_key
✓ Public Key: core_key.pub
✓ Certificate: core_key-cert.pub
```



```
admin@rocky:~/.ssh$ ssh -i ./core_key core@home.internal
Enter passphrase for key './core_key':
Fedora CoreOS 43.20260202.3.1
Tracker: https://github.com/coreos/fedora-coreos-tracker
Discuss: https://discussion.fedoraproject.org/tag/coreos

Last login: Tue Mar  3 05:24:15 2026 from 192.168.1.163
core@localhost:~$ 
```

Caddy web server does not support CRL checking, so we will be turning on CRL generation in step-ca for use with Keycloak. This ensures users with expired certs can no longer authenticate to the service.

Step CA and SSH

- Try SSHing to CoreOS, `ssh -i ~/.ssh/core_ed25519 core@home.internal`
  - If using the Rocky Linux desktop, the step CLI will automatically detect and use the key without needing to specify the private key
- While the private key is password protected, a more secure way to store SSH credentials is on a hardware backed token like a Yubikey
  - We'll cover this in lab 6.4

Installing Bookstack

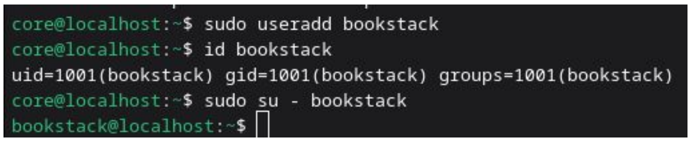
Lab 4.4

Installing Bookstack

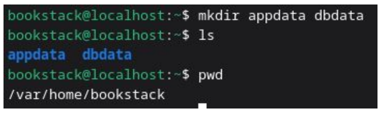
- Bookstack is a simple, self-hosted, easy-to-use platform for organizing and storing information, like a personal wiki
  - https://www.bookstackapp.com/
- This will be your primary “selfhosted” service for this workshop
- Bookstack supports multiple authentication methods (OIDC, SAML, LDAP)
- We’ll be using an adapted version of the official Docker compose file
  - https://codeberg.org/bookstack/devops/src/branch/main/config/lsio-docker/docker-compose.yml

Installing Bookstack

- Create a new user called “bookstack”:
  - `sudo useradd bookstack`
- Note the user’s user ID and group ID numbers:
  - `id bookstack`
- Change to the Bookstack user to set up directories for the Docker containers:
  - `sudo su - bookstack`
  - `mkdir appdata dbdata`



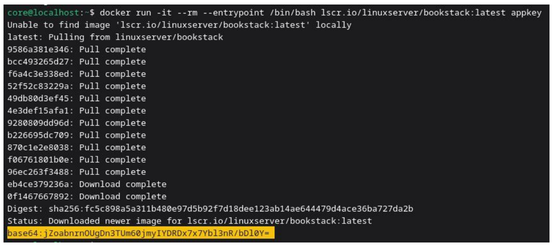
```
core@localhost:~$ sudo useradd bookstack
core@localhost:~$ id bookstack
uid=1001(bookstack) gid=1001(bookstack) groups=1001(bookstack)
core@localhost:~$ sudo su - bookstack
bookstack@localhost:~$
```



```
bookstack@localhost:~$ mkdir appdata dbdata
bookstack@localhost:~$ ls
appdata  dbdata
bookstack@localhost:~$ pwd
/var/home/bookstack
```

Installing Bookstack (Secrets Config)

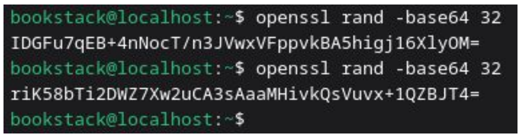
- Next, we'll randomly generate the secrets for Bookstack
- First, as the core user, run the following command to generate a unique app key:
  - `docker run -it --rm --entrypoint /bin/bash 10.1.1.4:3000/henryreed/bookstack:latest appkey`
  - Save the appkey output for the next step



```
core@localhost:~$ docker run -it --rm --entrypoint /bin/bash lscr.io/linuxserver/bookstack:latest appkey
Unable to find image 'lscr.io/linuxserver/bookstack:latest' locally
latest: Pulling from linuxserver/bookstack
9586a381e346: Pull complete
bcc493265d27: Pull complete
f6a4c3e338ed: Pull complete
52f52c83229a: Pull complete
49db80d3ef45: Pull complete
4e3def15afa1: Pull complete
9280809dd96d: Pull complete
b226695dc709: Pull complete
870c1e2e8038: Pull complete
f06761801b0e: Pull complete
96ec263f3488: Pull complete
eb4ce379236a: Download complete
0f1467667892: Download complete
Digest: sha256:fc5c898a5a311b480e97d5b92f7d18dee123ab14ae644479d4ace36ba727da2b
Status: Downloaded newer image for lscr.io/linuxserver/bookstack:latest
base64:jZoabnrnOUgDn3TUm60jmyIYDRDx7x7Ybl3nR/bDl0Y=
```

Installing Bookstack (Secrets Config)

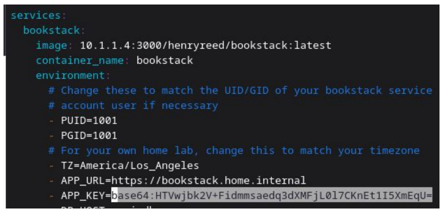
- We’ll need to generate two random passwords, one for the database user, one for the MariaDB root password
  - For simplicity, we’ll use the OpenSSL CLI to generate a random password: `openssl rand -base64 32`
- Replace the placeholder values in the Docker compose file including the app key you generated in the previous slide
  - `DB_PASSWORD` for the Bookstack container and `MYSQL_PASSWORD` for the MariaDB container should be the same



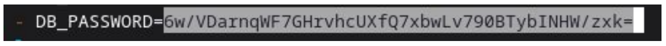
```
bookstack@localhost:~$ openssl rand -base64 32
IDGFu7qEB+4nNocT/n3JVwxVFppvkBA5higj16XlyOM=
bookstack@localhost:~$ openssl rand -base64 32
riK58bTi2DWZ7Xw2uCA3sAaaMHivkQsVuvx+1QZBJT4=
bookstack@localhost:~$
```

Install Bookstack and Caddy

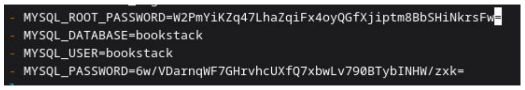
- As the `core` user, create a `bookstack` directory in the home folder, and cd to it
- Copy the `bookstack.yml` file from the git repo into `~/bookstack` and rename it to `docker-compose.yml`
- Replace the placeholder values in the Docker compose files with the secrets you generated in the last two steps



```
services:
  bookstack:
    image: 10.1.1.4:3000/henryreed/bookstack:latest
    container_name: bookstack
    environment:
      # Change these to match the UID/GID of your bookstack service
      # account user if necessary
      - PUID=1001
      - PGID=1001
      # For your own home lab, change this to match your timezone
      - TZ=America/Los_Angeles
      - APP_URL=https://bookstack.home.internal
      - APP_KEY=base64:HTVwjbk2V+Fidmmsaedq3dXMFjL0l7CKnEt1I5XmEqU=
```



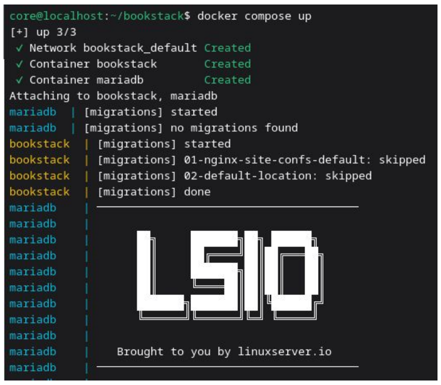
```
- DB_PASSWORD=6w/VDarnqWF7GHrvhcUXfQ7xbwLv790BTybINHW/zxk=
```



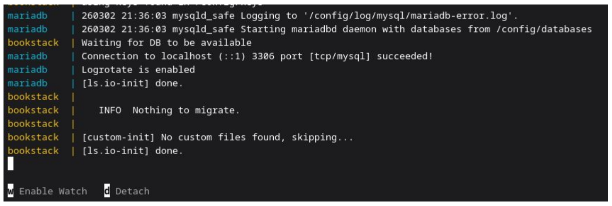
```
- MYSQL_ROOT_PASSWORD=W2PmYiKZq47LhaZqiFx4oyQGfXjiptm8BbSHiNkrsFw=
- MYSQL_DATABASE=bookstack
- MYSQL_USER=bookstack
- MYSQL_PASSWORD=6w/VDarnqWF7GHrvhcUXfQ7xbwLv790BTybINHW/zxk=
```

Install Bookstack

- Bring the Bookstack containers up with `docker compose up`
    - If there were no errors, hit d to detach



```
core@localhost:~/bookstack$ docker compose up
[+] up 3/3
 ✓ Network bookstack_default Created
 ✓ Container bookstack       Created
 ✓ Container mariadb         Created
Attaching to bookstack, mariadb
mariadb   | [migrations] started
mariadb   | [migrations] no migrations found
bookstack  | [migrations] started
bookstack  | [migrations] 01-nginx-site-confs-default: skipped
bookstack  | [migrations] 02-default-location: skipped
bookstack  | [migrations] done
mariadb    | ───────────────────────────────────
mariadb    |
mariadb    |
mariadb    |
mariadb    |
mariadb    |
mariadb    |
mariadb    |
mariadb    |
mariadb    |      Brought to you by linuxserver.io
mariadb    | ───────────────────────────────────
```



```
mariadb    | 260302 21:36:03 mysqld_safe Logging to '/config/log/mysql/mariadb-error.log'.
mariadb    | 260302 21:36:03 mysqld_safe Starting mariadbd daemon with databases from /config/databases
bookstack  | Waiting for DB to be available
mariadb    | Connection to localhost (::1) 3306 port [tcp/mysql] succeeded!
mariadb    | Logrotate is enabled
mariadb    | [ls.io-init] done.
bookstack  |
bookstack  |    INFO  Nothing to migrate.
bookstack  |
bookstack  | [custom-init] No custom files found, skipping...
bookstack  | [ls.io-init] done.

w Enable Watch   d Detach
```

Install Caddy

- Next, we'll do an initial setup of Caddy to reverse proxy your services
- Create a new directory called `caddy` and the subdirectories `conf` and `data` in the CoreOS home directory
- Copy the server CA's public cert into this directory
  - `docker cp stepca-server:/home/step/certs/root_ca.crt ~/caddy/conf`

Install Caddy

- Create a file called `Caddyfile` in the `conf` directory with the below contents
- This will use the ACME server you just set up on Step CA (server CA) to automatically issue TLS certs

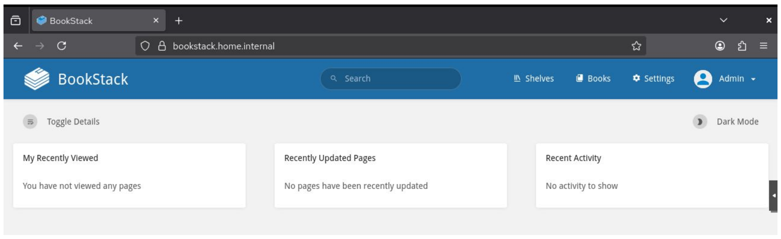
```
{
        email admin@home.internal
        acme_ca https://localhost:1443/acme/acme/directory
        acme_ca_root /etc/caddy/root_ca.crt
}

bookstack.home.internal:443 {
        reverse_proxy http://127.0.0.1:6875 {
}
```

Install Caddy

- Copy the Docker compose file for Caddy from the repo and rename it to `docker-compose.yml`, place it in `~/caddy`
- From the `~/caddy` directory, bring the web server up with `docker compose up -d`
- Test that the reverse proxying works by browsing to https://bookstack.home.internal from the Rocky VM
  - Bootstrapping the step CLI earlier added the server CA to the Rocky system trust, so there should be no certificate errors
- We'll be adding mTLS to Caddy in Lab 6.1

Bookstack Final Steps (for now)

- Log into Bookstack with the initial credentials admin@admin.com/password
- Bookstack is set up!
  - Change the default credentials
  - We’ll go over adding single sign on authentication with OIDC/SAML 2.0 in lab 5.1
  - Feel free to change the settings or test out Bookstack



My Recently Viewed

You have not viewed any pages

Recently Updated Pages

No pages have been recently updated

Recent Activity

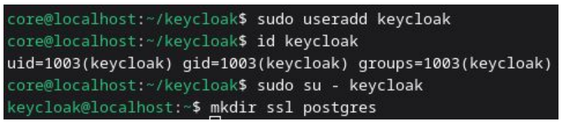
No activity to show

Installing Keycloak

Lab 4.5

Installing Keycloak

- Create a new user called “keycloak”:
  - `sudo useradd keycloak`
- Note the user’s user ID and group ID numbers:
  - `id keycloak`
- Change to the Bookstack user to set up directories for the Docker containers:
  - `sudo su - keycloak`
  - `mkdir postgres`



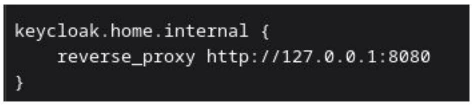
```
core@localhost:~/keycloak$ sudo useradd keycloak
core@localhost:~/keycloak$ id keycloak
uid=1003(keycloak) gid=1003(keycloak) groups=1003(keycloak)
core@localhost:~/keycloak$ sudo su - keycloak
keycloak@localhost:~$ mkdir ssl postgres
```

Installing Keycloak

- As the `core` user, create a `keycloak` directory in the home folder, and cd to it
- Copy the `keycloak.yml` file from the git repo into `~/keycloak` and rename it to `docker-compose.yml`
- We'll need to generate two random passwords, one for the Keycloak admin user, one for the Postgres user password
  - For simplicity, we'll use the OpenSSL CLI to generate a random password:
    `openssl rand -base64 32`
- Replace the placeholder values in the Docker compose file including the app key you generated in the previous slide
  - `KC_DB_PASSWORD` for the Keycloak container and `MYSQL_PASSWORD` for the MariaDB container should be the same

Installing Keycloak

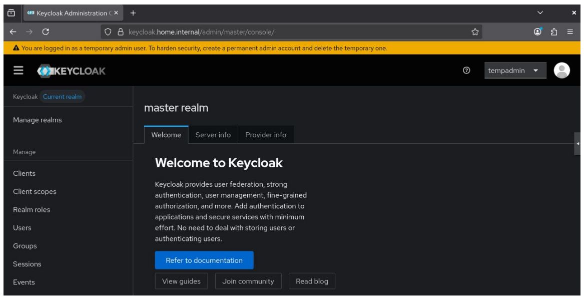
- From the `~/keycloak` directory, bring the service up with `docker compose up`
  - There may be some warnings, but as long as you don't see any errors, you can hit d to detach
- Next, we'll add to the Caddy config to reverse proxy Keycloak to the rest of the LAN
- Go to `~/caddy`, and edit the Caddyfile in `conf` to add the following block
- We will be adding mTLS support in the next section and lab



```
keycloak.home.internal {
    reverse_proxy http://127.0.0.1:8080
}
```

Installing Keycloak

- From your Rocky VM, try going to https://keycloak.home.internal
  - With the Caddy setup we did previously, this new subdomain should have already been issued a cert via ACME from the server CA
- Try logging in with the username tempadmin and the randomly generated password set in the Docker compose file


keycloak.home.internal/admin/master/console/

You are logged in as a temporary admin user. To harden security, create a permanent admin account and delete the temporary one.

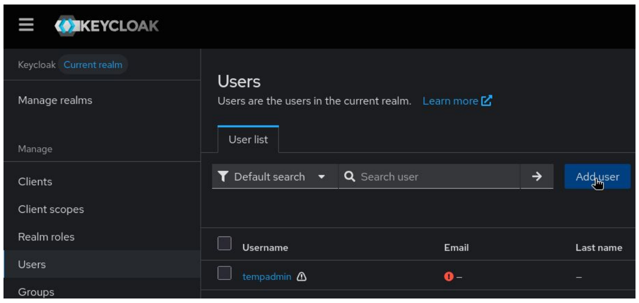
master realm

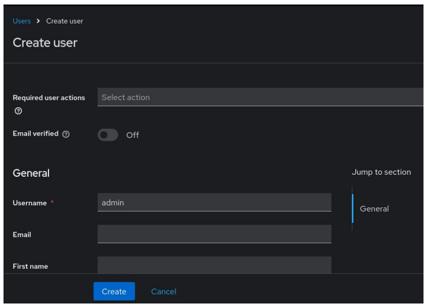
Welcome to Keycloak

Keycloak provides user federation, strong authentication, user management, fine-grained authorization, and more. Add authentication to applications and secure services with minimum effort. No need to deal with storing users or authenticating users.

Keycloak Setup

- As the final step for this lab, we'll create a new admin user and delete the bootstrap user as the web UI recommends
- Go to Users, then click Add User, call it admin, then hit Create

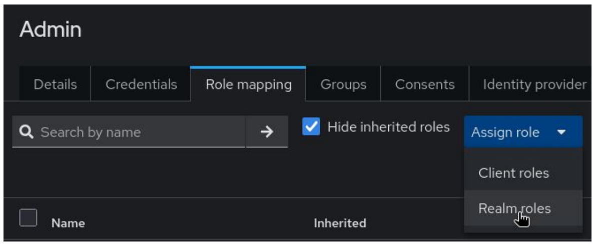


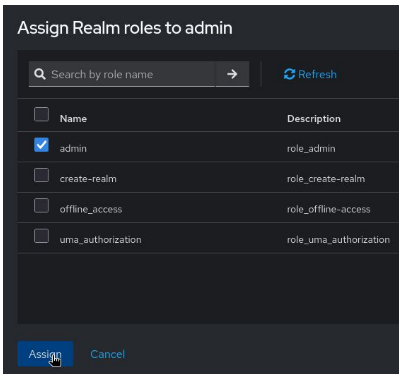


Username: admin

Keycloak Setup

- After creating the new user, go to the Role mapping tab, then click Assign role -> Realm roles
- Check off the “admin” role, then click Assign

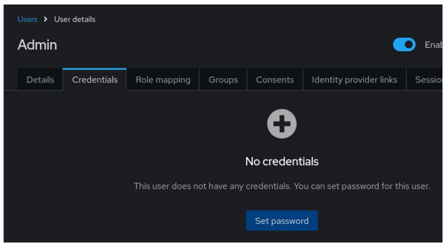




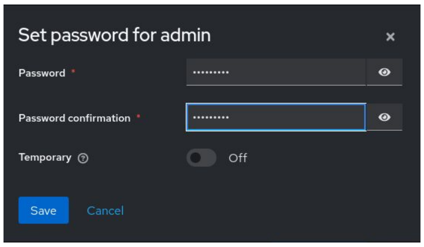
Name	Description
✓ admin	role_admin
create-realm	role_create-realm
offline_access	role_offline-access
uma_authorization	role_uma_authorization

Keycloak Setup

- Next, go to the Credentials tab, and click Set password
- Set a secure password, and toggle the Temporary setting off
  - You would normally use this setting when creating a new user and giving them a temporary password that Keycloak will prompt them to change on first sign on
- From the top right corner, sign out of the temporary admin account



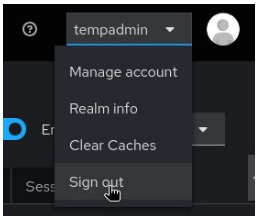
This user does not have any credentials. You can set password for this user.



Password * ·········

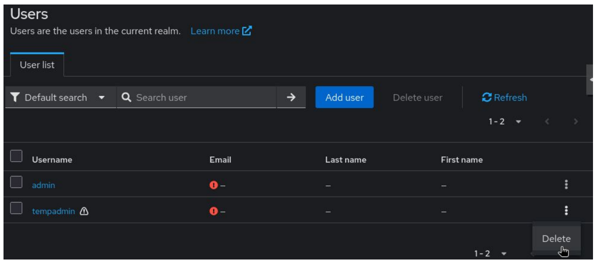
Password confirmation * ·········

Temporary Off



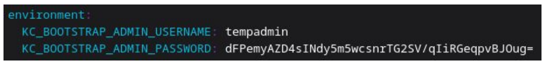
Keycloak Setup

- Log in with the new admin user you just created
- Go to the Users menu again, and delete the temporary admin user
- From the CoreOS host, edit the Docker compose file and remove the bootstrap admin entries under the environment section
    - Run `docker compose up -d` to restart the container with the new compose file


Users

Users are the users in the current realm. Learn more

Username	Email	Last name	First name
admin	–	–	–
tempadmin	–	–	–



```
environment:
  KC_BOOTSTRAP_ADMIN_USERNAME: tempadmin
  KC_BOOTSTRAP_ADMIN_PASSWORD: dFPemyAZD4sINdy5m5wcsnrTG2SV/qIiRGeqpvBJOug=
```

Keycloak Setup

- Keycloak is now set up (for now)!
- What's next for Keycloak in the next few labs?
    - We'll be adding authentication flows for OIDC for Bookstack, and also discussing when you'd need to use SAML 2.0 instead
    - We'll add mTLS to the reverse proxy entry for Keycloak in Caddy, and show you how to pass the certificate to Keycloak to authenticate users

Single Sign On with OIDC and SAML 2.0

Section 5

Creating a Keycloak Realm

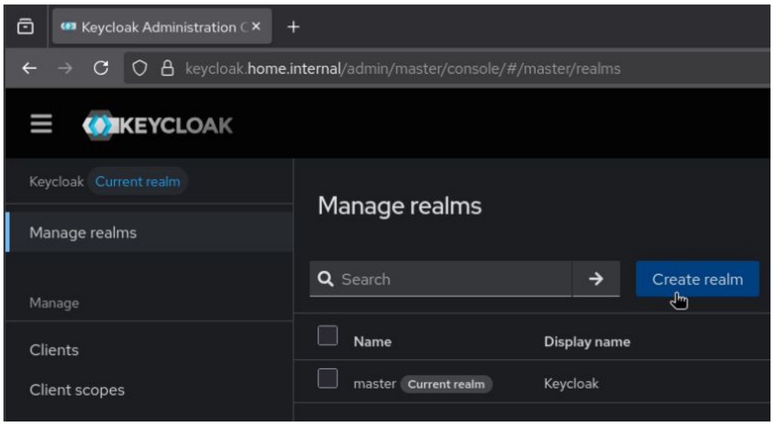
Lab 5.1

What is a realm?

- “A realm is a space where you manage objects, including users, applications, roles, and groups.” [1]
  - “A user belongs to and logs into a realm.” [1]
  - “One Keycloak deployment can define, store, and manage as many realms as there is space for in the database” [1]
- Our general recommendation: keep Keycloak’s admin account in a realm separate from all other realms.
  - Default realm is “master” and displayed as “Keycloak”; do not use these for services or users other than the Keycloak admin user

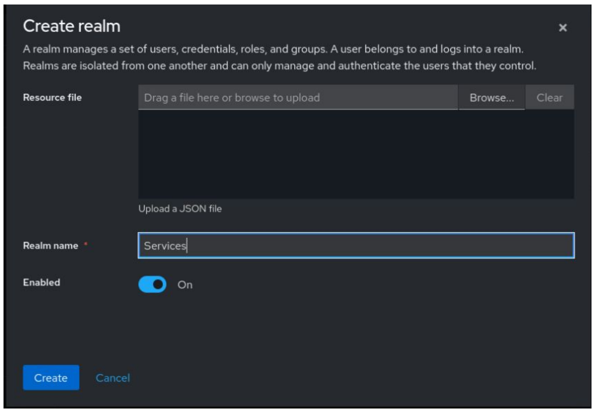
[1] https://www.keycloak.org/docs/latest/server_admin/index.html#_configuring-realms

Create a New Realm (1/2)



From the Keycloak admin panel, select “Manage realms” and then press on “Create realm”

Create a New Realm (2/2)


Create realm

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

Resource file: Drag a file here or browse to upload

Upload a JSON file

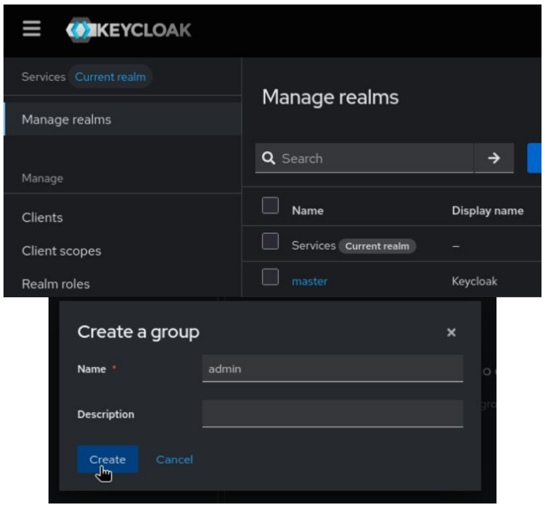
Realm name *: Services

Enabled: On

Give your realm a name then press “create”

Creating a Group in the New Realm

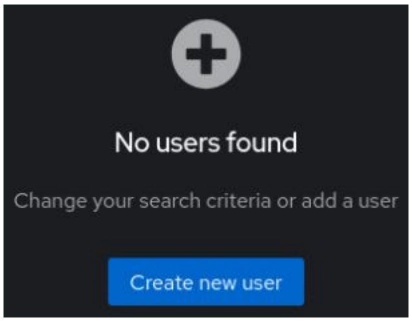
- You will now be in the new realm
- You will notice this via the top left where "Services" (your realm's name) has the blue text "Current realm" next to it, same as in the "Manage realms" tab
- Press on Groups -> Create New Groups
- Name the group "admin" and press "Create"



Name: admin

Creating Users in the New Realm (1/3)

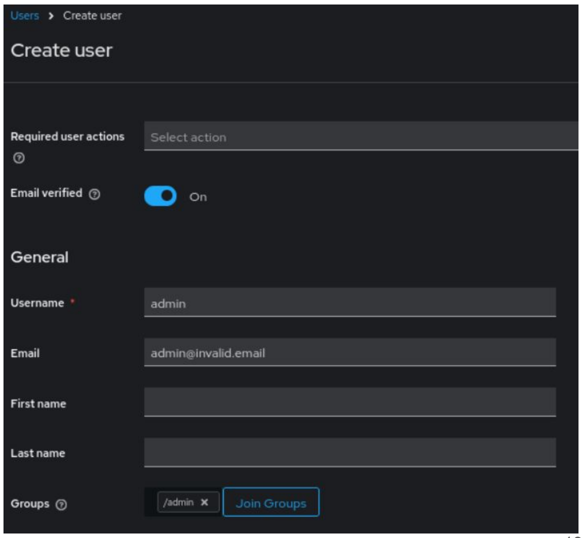
- Press on Users to create new users
- You'll notice the message "No users found"
- Why?
- The admin account is only in the master realm
- If you attach a service to this realm, your Keycloak login won't work
    - This is intentional!


No users found

Change your search criteria or add a user

Creating Users in the New Realm (2/3)

- Fill out the following:
  - Email
  - First name
  - Last name
  - Email Verified -> On
  - Groups -> admin
- Many services expect an email, even if you're not going to need an email service within your lab
- Then, press create!



Required user actions: Select action

Email verified: On

General

Username *: admin

Email: admin@invalid.email

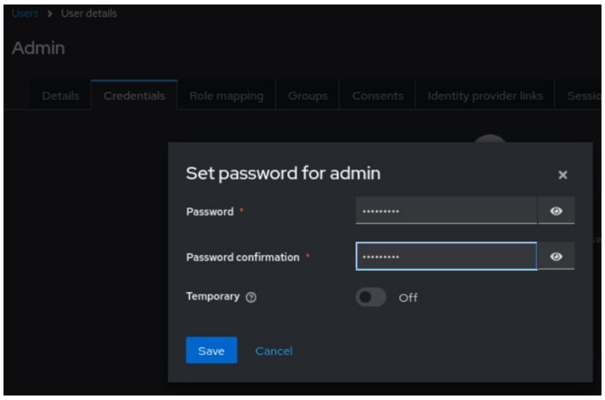
First name:

Last name:

Groups: /admin

Creating Users in the New Realm (3/3)

- Head to Credentials -> Set Password
- Set a password and uncheck "Temporary"
- Then, press save



Set password for admin

Password * ••••••••••

Password confirmation * ••••••••••

Temporary Off

We’re done!

- Quick and easy, right? But what exactly happened?
- Keycloak allows you to login to itself only using the master realm
- The new realm will be used for services like Bookstack and others that support OIDC and SAML 2.0 authentication

Configuring OIDC in Bookstack

Lab 5.2

Introduction

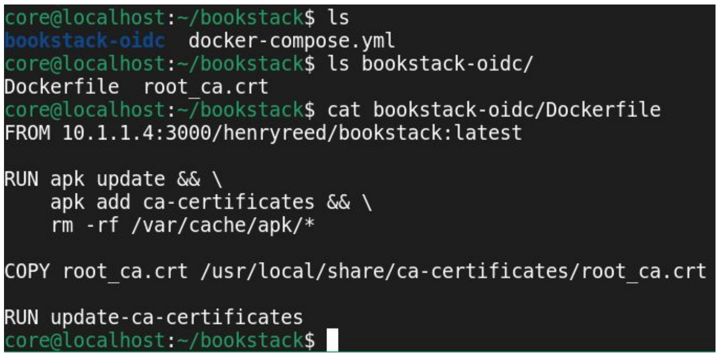
- We’ll now connect Keycloak to Bookstack, so that we only have to login to one account with one password
- With more services, this will translate to other areas of your network, allowing you to login to any number of services only once
- Hence the name: single sign on!
- Note: We refer to Bookstack as a web service (which it is); in Keycloak parlance, the correct term is “client”
    - Keycloak being the service that clients interact with

Steps for Supporting OIDC

1. Bookstack needs to be able to communicate with Keycloak; as such, its Docker image needs to be modified so that the “operating system certificate store” contains our Server CA
2. A “groups” client scope must be added in Keycloak
3. A Bookstack client must be added in Keycloak
4. Bookstack needs to be configured to use OIDC

Add CA Certificate to Bookstack Image (1/2)

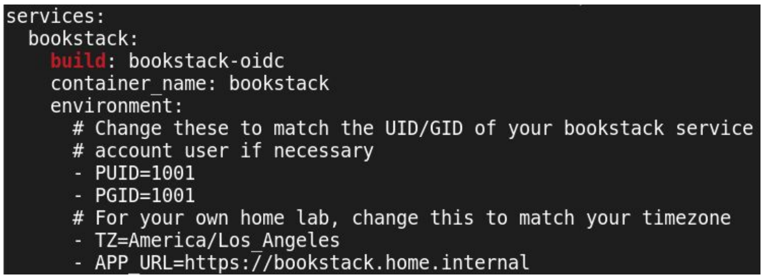
- From the workshop Git repo, place the “bookstack-oidc/Dockerfile” directory and file in the same directory as the Bookstack compose file
- Also copy the root_ca.crt into bookstack-oidc; it should look like the screenshot below



```
core@localhost:~/bookstack$ ls
bookstack-oidc  docker-compose.yml
core@localhost:~/bookstack$ ls bookstack-oidc/
Dockerfile  root_ca.crt
core@localhost:~/bookstack$ cat bookstack-oidc/Dockerfile
FROM 10.1.1.4:3000/henryreed/bookstack:latest

RUN apk update && \
    apk add ca-certificates && \
    rm -rf /var/cache/apk/*

COPY root_ca.crt /usr/local/share/ca-certificates/root_ca.crt

RUN update-ca-certificates
core@localhost:~/bookstack$
```

Add CA Certificate to Bookstack Image (2/2)

- Then, instead of “image:” use the “build:” directive in your compose file, like so:

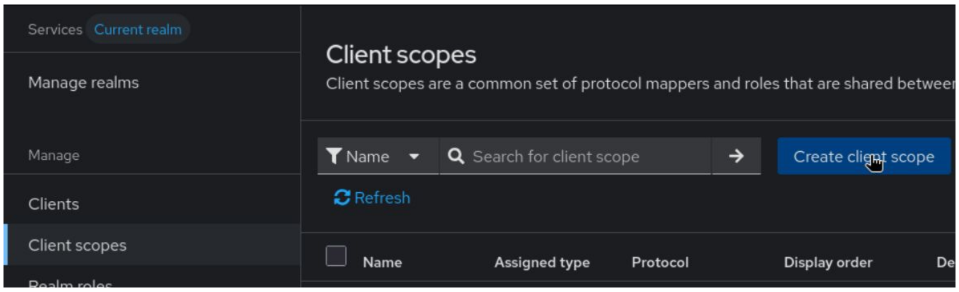


```
services:
  bookstack:
    build: bookstack-oidc
    container_name: bookstack
    environment:
      # Change these to match the UID/GID of your bookstack service
      # account user if necessary
      - PUID=1001
      - PGID=1001
      # For your own home lab, change this to match your timezone
      - TZ=America/Los_Angeles
      - APP_URL=https://bookstack.home.internal
```

- Lastly, run `sudo docker compose down && sudo docker compose up -d`
    - This will rebuild Bookstack and add our Root CA file into its “OS certificate store”

Creating a Client Scope (1/4)

- Most OIDC clients use a *groups* scope to query what groups a user is added to
- Keycloak doesn't include this by default (boo!) so we will add it manually
- Select "Client scopes" -> "Create client scope"

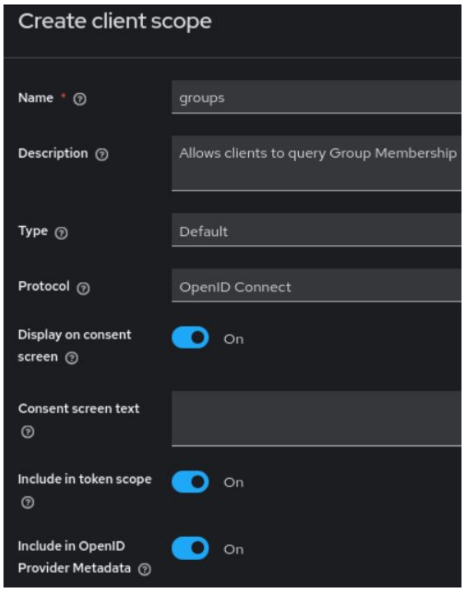


Client scopes

Client scopes are a common set of protocol mappers and roles that are shared betwee

Creating a Client Scope (2/4)

- For name, use “groups” (lowercase)
- Type -> Default
  - This way, all OIDC clients have this client scope automatically
- Enter a good description
- Select “On” for the following:
  - Display on consent screen
  - Include in token scope
  - Include in OpenID Provider Metadata

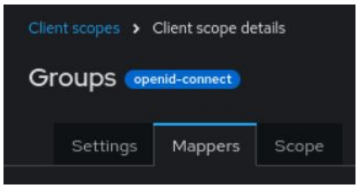


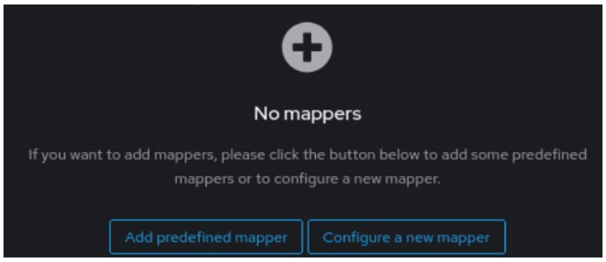
Create client scope

Name *	groups
Description	Allows clients to query Group Membership
Type	Default
Protocol	OpenID Connect
Display on consent screen	On
Consent screen text	
Include in token scope	On
Include in OpenID Provider Metadata	On

Creating a Client Scope (3/4)

- On the next screen, select Mappers -> Configure a new mapper

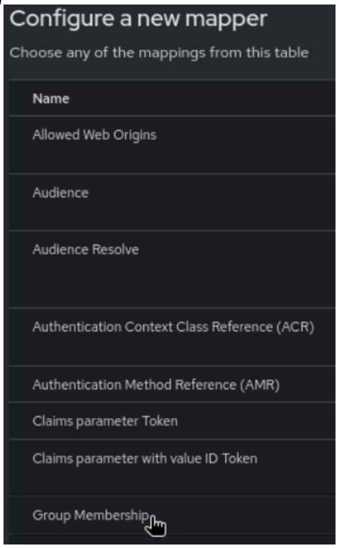




If you want to add mappers, please click the button below to add some predefined mappers or to configure a new mapper.

Creating a Client Scope (4/4)

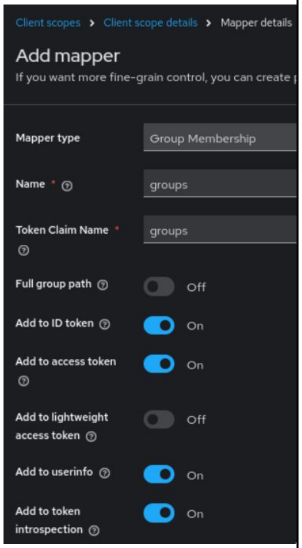
- Select “Group Membership”
- For “Name” and “Token Claim Name” use “groups” all lowercase
- Uncheck “Full group path”
  - Allows “/admin” to become “admin” which is what most web services expect
- Leave all options as is and press save



Configure a new mapper

Choose any of the mappings from this table

Name
Allowed Web Origins
Audience
Audience Resolve
Authentication Context Class Reference (ACR)
Authentication Method Reference (AMR)
Claims parameter Token
Claims parameter with value ID Token
Group Membership



Client scopes > Client scope details > Mapper details

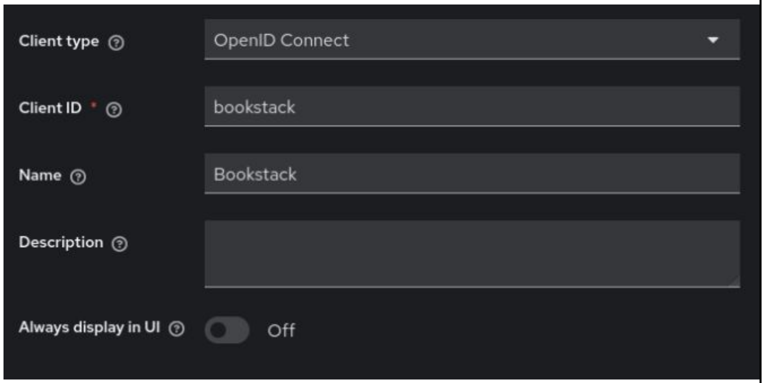
Add mapper

If you want more fine-grain control, you can create

Mapper type	Group Membership
Name *	groups
Token Claim Name *	groups
Full group path	Off
Add to ID token	On
Add to access token	On
Add to lightweight access token	Off
Add to userinfo	On
Add to token introspection	On

Creating a Bookstack Client (1/3)

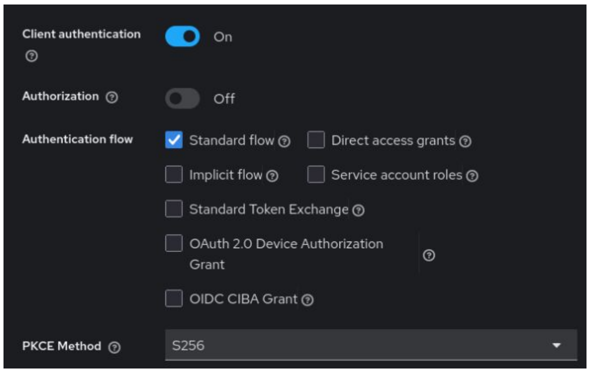
- In Keycloak, navigate to Clients -> Create client
- Give the client an ID (which can be any arbitrary value) and a name



Client type	OpenID Connect
Client ID *	bookstack
Name	Bookstack
Description	
Always display in UI	Off

Creating a Bookstack Client (2/3)

- Select the checkbox next to “Client authentication”
- PKCE Method -> S256
  - This adds an additional cryptographic layer to OIDC authentication
- PKCE support is client dependent
  - Bookstack supports it, services with an older implementation of OIDC do not. Consult your documentation!
- Leave the rest as defaults



Client authentication: On

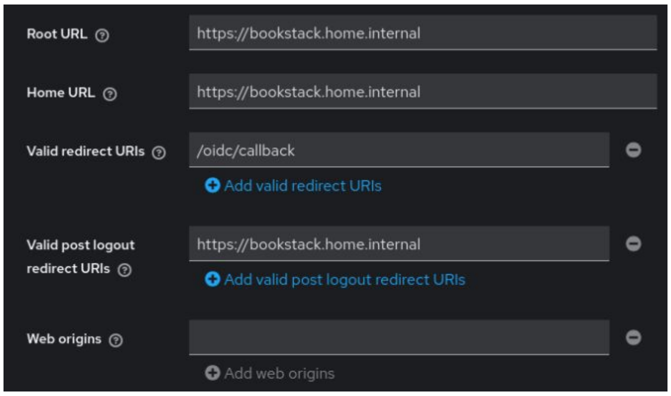
Authorization: Off

Authentication flow: [✓ Standard flow] Direct access grants, Implicit flow, Service account roles, Standard Token Exchange, OAuth 2.0 Device Authorization Grant, OIDC CIBA Grant

PKCE Method: S256

Creating a Bookstack Client (3/3)

- Root, Home, and Valid post logout redirect URLs/URIs are all:
  - https://bookstack.home.internal
- Valid redirect URIs:
  - /oidc/callback
- How do we know these values? [1]
  - These are determined by your service's development team
  - For Bookstack, it's Dan Brown
  - Consult the documentation for your service to get these values!

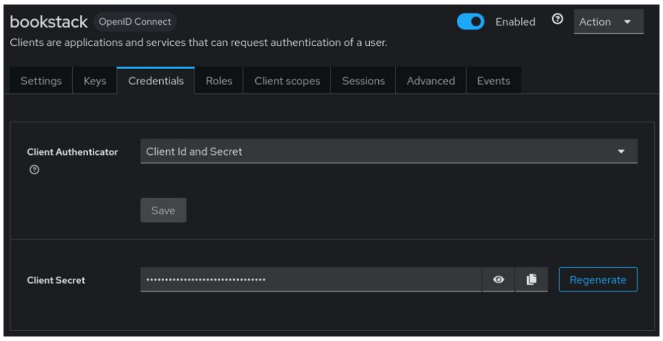


Root URL	https://bookstack.home.internal
Home URL	https://bookstack.home.internal
Valid redirect URIs	/oidc/callback
Valid post logout redirect URIs	https://bookstack.home.internal
Web origins	

[1] https://www.bookstackapp.com/docs/admin/oidc-auth/

Client Secret

- In the next screen, press on “credentials”
- You’ll notice the starred “Client Secret” value
- This will be the secret we’ll add to the Bookstack environment file to allow it to authenticate with Keycloak

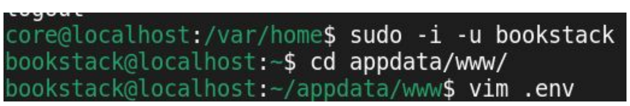


Clients are applications and services that can request authentication of a user.

Client Authenticator: Client Id and Secret

Client Secret: ••••••••••••••••••••••••••••••••

Configuring Bookstack



```
core@localhost:/var/home$ sudo -i -u bookstack
bookstack@localhost:~$ cd appdata/www/
bookstack@localhost:~/appdata/www$ vim .env
```

- Login to your Fedora CoreOS instance, login to the Bookstack user account, then edit the .env file in ~/appdata/www
- Refer to your OIDC environment file example in the workshop Git repo, and change the following:
    - AUTH_AUTO_INITIATE=true
    - OIDC_DISPLAY_NAME_CLAIMS=preferred_username
    - OIDC_CLIENT_ID=bookstack
    - OIDC_CLIENT_SECRET=<CLIENT SECRET FROM KEYCLOAK>
    - OIDC_ISSUER=https://keycloak.home.internal/realms/<REALM NAME YOU USED>
    - OIDC_END_SESSION_ENDPOINT=true

Try to login!

- Bookstack will now use Keycloak’s new realm to log you in
- You will sign in “once” to Keycloak’s new realm, and be able to reuse that same session with other services you attach to Keycloak
- Congratulations: OIDC is now set up

On SAML 2.0

- We won't demo or do a walkthrough of SAML 2.0, but it's important to discuss
- The basic steps are the same: create a SAML 2.0 client, and use your service's instructions to fill in the proper URLs, keys and other details to get things hooked up
- SAML 2.0 uses cryptographic signatures and XML files; Keycloak signs an XML file, your browser transmits it to the service, the service verifies the signature
- It's older, but it's still useful
- **Unlike OIDC, SAML 2.0 does not require the web service to have direct connectivity to Keycloak**
- This is useful in many enterprise environments where the SSO is behind a network that an external service cannot access

Certificate Authorities and Layer 6 Authentication (mTLS)

Section 6

The Benefits (and Beauty) of TLS, and mTLS

- TLS: The cryptographic protocol that is ubiquitous across the web
    - The little lock that lets you know websites are trusted all exist due to TLS
    - This is your client cryptographically verifying the identity of the server
    - The server's identity is signed by a Certificate Authority; the identity is an X.509 certificate file and a private key
- Mutual TLS (mTLS): You verify the server, and the server verifies you
- Why it matters:
    - Assume worst case: your service has an exploitable RCE vulnerability
    - With mTLS, the attacker will be required to key and cert before they can attack the service
- This bears repeating: mTLS happens at the crypto level. Unless someone steals your keys or breaks your crypto, bad guys can't hack your vulnerable service
- Another cool tidbit: if you use an unrelated term for your domain name, the attacker won't even know what service you're running behind mTLS

Smart Cards and mTLS

- Smart cards!
  - National IDs in Belgium, Estonia
  - U.S. Military and federal services (NASA, JPL)
  - Yubikey PIV application
- Smart cards can't leak the private key, and they have PINs, making them cryptographic hardware two-factor authentication devices
- Soft certificates are also an option, at the loss of 2FA

Datum und Ort der Ausstellung / Date and Place of issue
06 01 2020 Eupen
IDBEL000001115<7027<<
"Left my CAC in the bathroom? That was ONE time!"
CONFESSIONS OF A SECRET SQUIRREL
CAREER REINVESTIGATION No. 01202014
ClearanceJobs

Where can you use mTLS?

- Web server
- OpenVPN and similar
- Combined with a smart card, you have hardware-backed 2FA with unguessable keys
- Additionally, cryptographic certificates hold your username, too! Combined with something like Keycloak, you won't need to know your username or password again
- In the next few slides, we'll show you:
    - Web server mTLS
    - Keycloak mTLS

Web Servers and mTLS

- Most web servers have some form of support for mTLS
  - `SSLVerifyClient` in Apache
  - `ssl_verify_client` in NGINX (stream module)
    - As a side note, NGINX in HTTP mode doesn't terminate the connection at the transport layer if client verification fails (it instead sends an HTTP error code but the TLS handshake is still done)
- We'll be adding mTLS onto the Caddy config you made earlier for Bookstack
  - Caddy doesn't support certificate revocation checking without a 3rd party plugin
  - We'll have Keycloak check the status so even if transport layer is established, application level authentication with mTLS will fail

Secure mTLS Key Usage

- Certificates for mTLS are the same as any other user certificate generated by a CA
- It's possible to use your mTLS certificate with a soft private key, though this makes it significantly easier to compromise
- Yubikey PIV is a smart card! The Yubikey shows up as a "smart card reader" to your computer, allowing you to have the same feature set as an actual card but in a convenient form factor
    - If time permits, we'll show how to generate a user cert in lab 6.4 for anyone who brought a Yubikey today

Enabling mTLS in Your Web Server

Lab 6.1

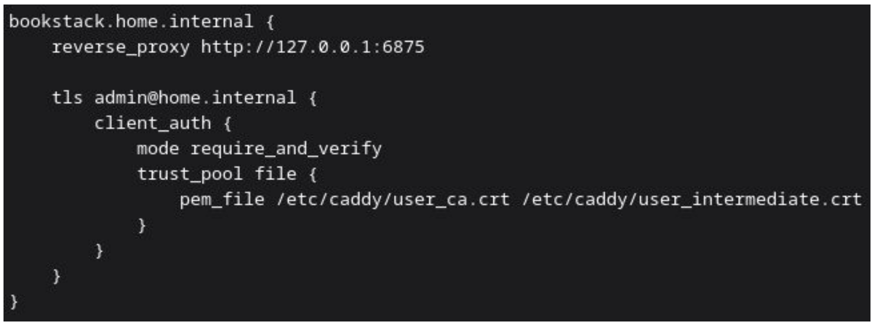
mTLS in Caddy

- Adding mTLS to an existing Caddy config is simple
- On CoreOS, go the `~/caddy` directory, then copy the user CA root and intermediate public certs
  - `docker cp stepca-user:/home/step/certs/root_ca.crt ~/caddy/conf/user_ca.crt`
  - `docker cp stepca-user:/home/step/certs/intermediate_ca.crt ~/caddy/conf/user_intermediate.crt`
  - User certs will be verified against the user CA we set up earlier
- Side note: Although Caddy verifies that the certificate is valid at time of use, it does not support checking revocation status (via CRL or OCSP) without a third-party plugin

Caddy has various options for the tls module, including a client certificate verification mode (https://caddyserver.com/docs/caddyfile/directives/tls). The four modes are:
request: Ask clients for a certificate, but allow even if there isn't one; do not verify it
require: Require clients to present a certificate, but do not verify it
verify_if_given: Ask clients for a certificate; allow even if there isn't one, but verify it if there is
require_and_verify: Require clients to present a valid certificate that is verified

mTLS in Caddy

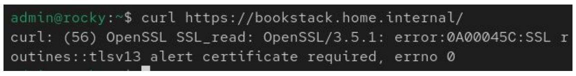
- Edit the `Caddyfile` in `~/caddy/conf`, the Bookstack block should now look like below
- Restart Caddy with docker compose restart



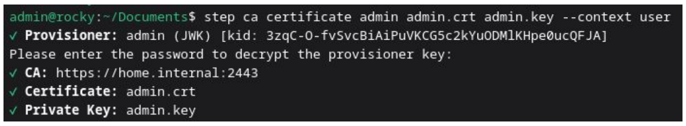
```
bookstack.home.internal {
    reverse_proxy http://127.0.0.1:6875

    tls admin@home.internal {
        client_auth {
            mode require_and_verify
            trust_pool file {
                pem_file /etc/caddy/user_ca.crt /etc/caddy/user_intermediate.crt
            }
        }
    }
}
```

mTLS in Caddy

- If you try to open Bookstack now, the connection will be refused since we don't yet have a cert
- Next, we'll generate a soft cert so Firefox can access Bookstack again



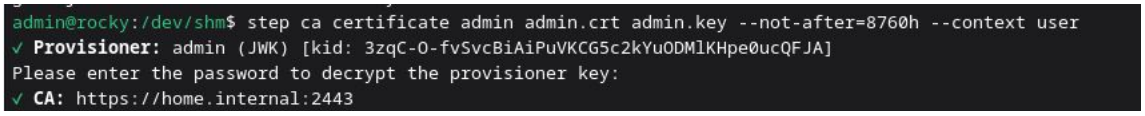
```
admin@rocky:~$ curl https://bookstack.home.internal/
curl: (56) OpenSSL SSL_read: OpenSSL/3.5.1: error:0A00045C:SSL r
outines::tlsv13 alert certificate required, errno 0
```



```
admin@rocky:~/Documents$ step ca certificate admin admin.crt admin.key --context user
✓ Provisioner: admin (JWK) [kid: 3zqC-O-fvSvcBiAiPuVKCG5c2kYuODMlKHpe0ucQFJA]
Please enter the password to decrypt the provisioner key:
✓ CA: https://home.internal:2443
✓ Certificate: admin.crt
✓ Private Key: admin.key
```

Generating a User Cert

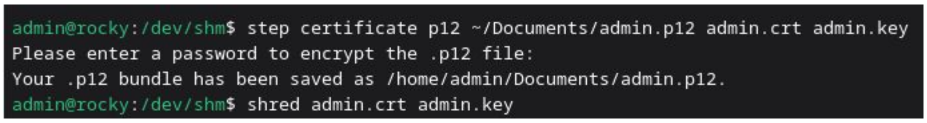
- The ca subcommand for step CLI does not request a password for the cert private key, so we'll create it in memory tmpfs then shred the plaintext key
- On your Rocky VM, go to /dev/shm, then run the command:
  `step ca certificate admin admin.crt admin.key --not-after=8766h --context user`
  - When prompted, provide the user CA administrative password saved earlier
  - By default and design philosophy, Step CA issues short lived certs
  - For the purposes of this workshop and lab, we'll generate one that lasts for a year since you won't want to have to generate your user cert every other day.



```
admin@rocky:/dev/shm$ step ca certificate admin admin.crt admin.key --not-after=8760h --context user
✓ Provisioner: admin (JWK) [kid: 3zqC-O-fvSvcBiAiPuVKCG5c2kYuODMlKHpe0ucQFJA]
Please enter the password to decrypt the provisioner key:
✓ CA: https://home.internal:2443
```

Generating a User Cert

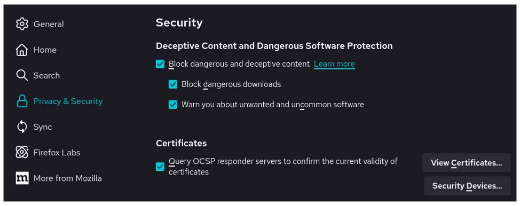
- Bundle the cert and key into a PKCS12 file: `step certificate p12 ~/Documents/admin.p12 admin.crt admin.key`
  - When prompted, provide a password to encrypt the bundle
- Shred the plaintext cert and key: `shred admin.crt admin.key`



```
admin@rocky:/dev/shm$ step certificate p12 ~/Documents/admin.p12 admin.crt admin.key
Please enter a password to encrypt the .p12 file:
Your .p12 bundle has been saved as /home/admin/Documents/admin.p12.
admin@rocky:/dev/shm$ shred admin.crt admin.key
```

Using the User Cert

- Open Firefox, and go to Settings -> Privacy & Security -> Certificates, then click on View Certificates
- Select the Your Certificates tab, then click Import, then select the `admin.p12` bundle we just created, and input its password when prompted



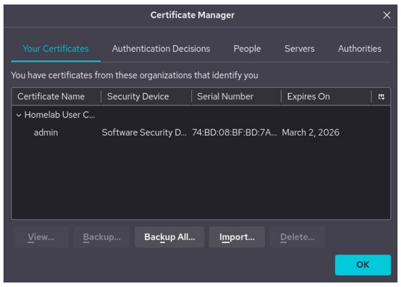
Security

Deceptive Content and Dangerous Software Protection

- [✓] Block dangerous and deceptive content Learn more
  - [✓] Block dangerous downloads
  - [✓] Warn you about unwanted and uncommon software

Certificates

- [✓] Query OCSP responder servers to confirm the current validity of certificates



You have certificates from these organizations that identify you

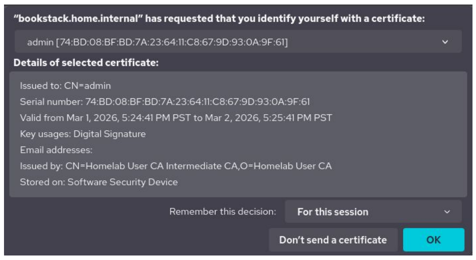
Certificate Name	Security Device	Serial Number	Expires On
Homelab User C...			
admin	Software Security D...	74:BD:08:BF:BD:7A...	March 2, 2026

It's possible to use certs (both software and hardware-backed) with command line tools, but since Bookstack is a web app, we'll show you how to add it to Firefox.

As an example, if you self host a Git server like Forgejo or Gitea, you can protect the publicly exposed web endpoint using mTLS on the web server. In addition, if you don't have SSH open for Git over SSH to reduce your attack surface, you can configure the Git client to use PKI for mTLS; it's easiest with a soft cert, but also possible with a user cert on a Yubikey and some one-time extra configuration (see https://memetichenry.com/books/it-and-devops/page/using-smart-cards-for-remote-git-instances).

Using the User Cert

- Now, try browsing to https://bookstack.home.internal, and Firefox should prompt you to use the certificate you just imported
- You now have the first step in mTLS set up (client cert authentication)!
  - In the next lab, we'll actually use the cert to authenticate you to Bookstack using Keycloak and OIDC



"bookstack.home.internal" has requested that you identify yourself with a certificate:

admin [74:BD:08:BF:BD:7A:23:64:11:C8:67:9D:93:0A:9F:61]

Details of selected certificate:

Issued to: CN=admin
Serial number: 74:BD:08:BF:BD:7A:23:64:11:C8:67:9D:93:0A:9F:61
Valid from Mar 1, 2026, 5:24:41 PM PST to Mar 2, 2026, 5:25:41 PM PST
Key usages: Digital Signature
Email addresses:
Issued by: CN=Homelab User CA Intermediate CA,O=Homelab User CA
Stored on: Software Security Device

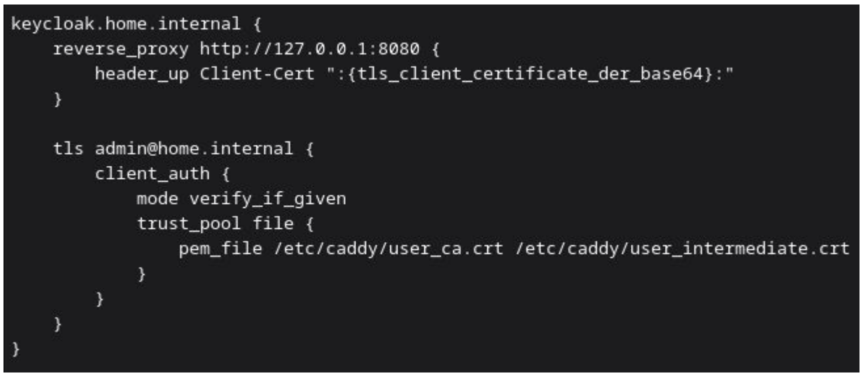
Remember this decision: For this session

Enabling mTLS in Keycloak

Lab 6.2

Keycloak Caddy mTLS

- Next, we'll add to the Caddy config to reverse proxy Keycloak to the rest of the LAN
- Go to `~/caddy`, and edit the Keycloak host block in your Caddyfile in `conf` to look like below:



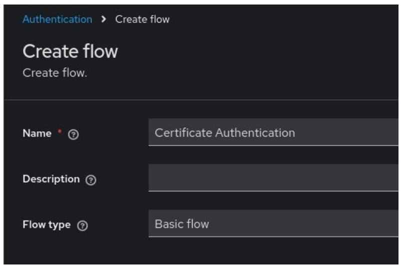
```
keycloak.home.internal {
    reverse_proxy http://127.0.0.1:8080 {
        header_up Client-Cert ":{tls_client_certificate_der_base64}:"
    }

    tls admin@home.internal {
        client_auth {
            mode verify_if_given
            trust_pool file {
                pem_file /etc/caddy/user_ca.crt /etc/caddy/user_intermediate.crt
            }
        }
    }
}
```

The crucial line here is the block within reverse_proxy, header_up. This option grabs the cert from the browser/client and passes it to Keycloak, which is configured to look for the client certificate in the RFC 9440 format (a base64 encoded DER cert). This is set in the Docker compose file for Keycloak. In addition, we need to set the verification mode to "verify_if_given", because OIDC requires that the OIDC client application is able to reach out to the authentication provider. There are ways to provision certs for inter-server communication or spilt DNS, but these would take more time to cover.

The RFC: https://datatracker.ietf.org/doc/rfc9440/

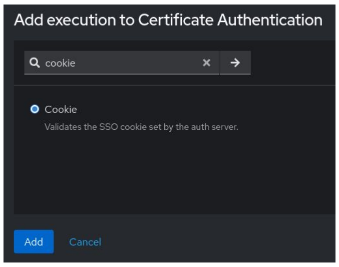
Keycloak Caddy mTLS

- Back in Keycloak, from the left menu, go to Configure -> Authentication -> Flows, and click “Create flow”
- Call it “Certificate Authentication”, then everything else on the first page can be left default
- We’re going to add three execution steps for this flow, starting with cookies
  - On the Flow details screen, click Add execution, then search for “cookie”


Authentication > Create flow

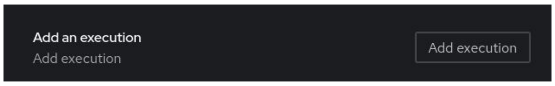
Create flow
Create flow.

Name: Certificate Authentication
Description:
Flow type: Basic flow


Add execution to Certificate Authentication

cookie

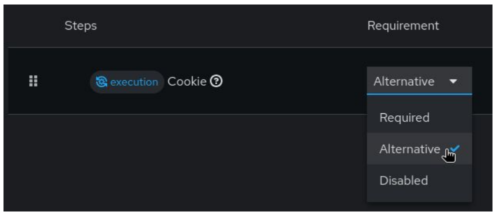
Cookie
Validates the SSO cookie set by the auth server.


Add an execution
Add execution

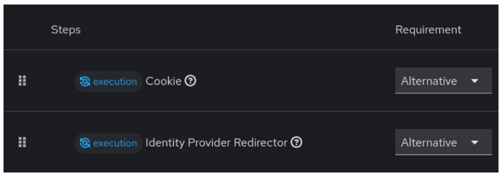
In the next few steps, we’re going to configure Keycloak to read the user certificate passed on from Caddy, to then verify the cert, make sure it’s not revoked (using the CRL), then authenticate the user if everything looked good.

Keycloak Caddy mTLS

- After adding Cookie, set the Requirement field to “Alternative”
- Repeat the previous steps to add another execution step for “Identity Provider Redirector”, and also set the requirement to this one to “Alternative”
- No additional options need to be configured for these two execution steps



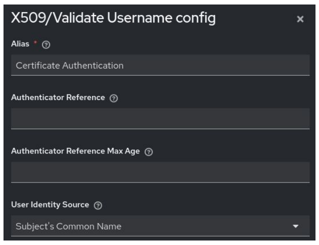
Cookie — Alternative



Steps	Requirement
Cookie	Alternative
Identity Provider Redirector	Alternative

Keycloak Caddy mTLS

- Next, add another execution step for "X509/Validate Username"
  - This step should automatically be added as Required
- Click on the Settings icon to configure this step
- Set the following settings:
  - Alias: Certificate Authentication
  - User Identity Source: Subject's Common Name
  - User mapping method: Username or Email
  - Turn on the CRL Checking Enabled option
  - CRL Path: https://home.internal:2443/crl
  - Turn on the Revalidate Client Certificate option
  - Any other options can be left default

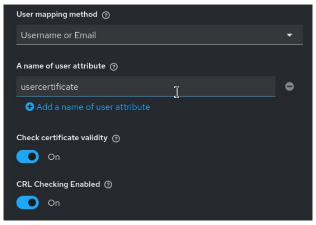

X509/Validate Username config

Alias *: Certificate Authentication

Authenticator Reference:

Authenticator Reference Max Age:

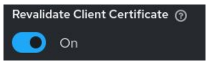
User Identity Source: Subject's Common Name

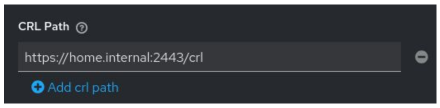

User mapping method: Username or Email

A name of user attribute: usercertificate

Check certificate validity: On

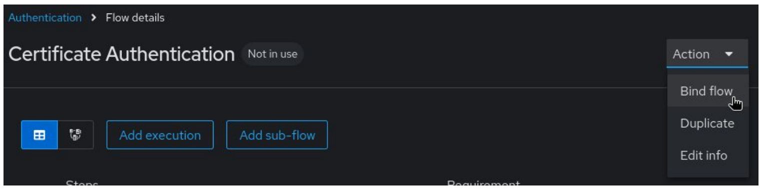
CRL Checking Enabled: On

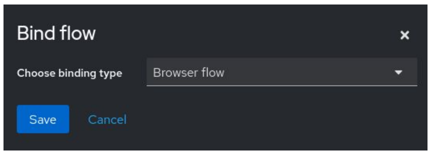

Revalidate Client Certificate: On


CRL Path: https://home.internal:2443/crl

Keycloak Caddy mTLS

- From the Action drop down menu, click Bind flow
- Choose the binding type Browser flow, then click Save





Keycloak Caddy mTLS

- Log out of Bookstack if you're still logged in, and close Firefox to reset the certificate request states
- Open Firefox again, go to Bookstack, and this time, after the browser requests your user certs, it should automatically log you into Bookstack!

Using Yubikey Smart Cards

Lab 6.3 (optional)

Yubikey Intro

- Earlier, we showed you how to do mTLS with a soft cert and key, but what if you want to use something more secure?
- A Yubikey can be your hardware-backed store for private keys for mTLS
  - There are other devices with PIV/PKCS11 compatibility, but since Yubikeys are ubiquitous and have open source library support, we'll focus on it for this workshop
  - It's also possible to use a physical smart card to store your mTLS key and cert
- Download the Yubico Authenticator
  - https://www.yubico.com/products/yubico-authenticator/
- There are also command line tools such as yubikey-manager (ykman)
  - https://github.com/Yubico/yubikey-manager
  - For ease of following along we'll use the Yubico Authenticator

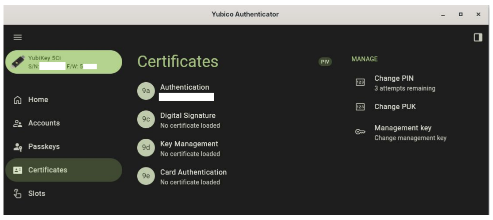
Authenticator devices like Yubikeys and similar devices can serve as two factor authentication devices (something you have). Depending on how they are configured, the device itself can be two factors in one (if you add a PIN, something you know).

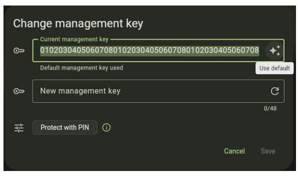
Yubikey Best Practices

- All Yubikeys with PIV application support ship with the same defaults for PIN, PIN Unblocking Key (PUK), and management key
  - The defaults are PIN: 123456, PUK: 12345678, and management key: 010203040506070801020304050607080102030405060708
- The first step you should take with a new Yubikey is to change all three of these
  - Don’t forget the PUK, because after lockout, the only solution to be able to use the PIV application is to reset it (meaning you lose the private key)
- Ideally, have two Yubikeys/PIV devices available
  - Things happen, you might lose one or the PUK gets locked out

Yubico Authenticator

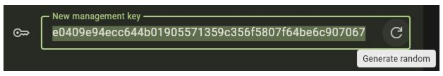
- Plug your new Yubikey into your computer, and open Yubico Authenticator
- On the left menu, select Certificates
- For a new Yubikey, the first thing we want to do is set the PIN, PUK, and management under Manage
  - The current (factory default) PIN is 123456, PUK is 12345678
  - For management key, select the star icon as shown below to insert the factory default
  - Store the management key somewhere safe!





Current management key: 010203040506070801020304050607080102030405060708

Default management key used



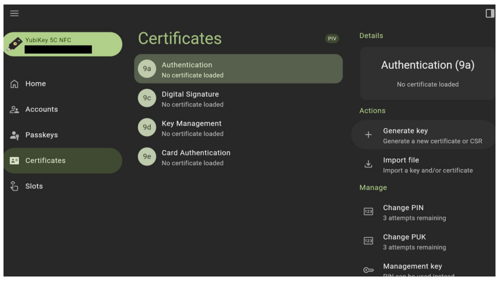
New management key: e0409e94ecc644b01905571359c356f5807f64be6c907067

Provisioning the Yubikey

- The process is slightly different from generating a soft user cert and key
- We'll first use Yubico Authenticator to generate a certificate signing request (CSR)
- Next, use the user CA to sign the CSR and generate the user cert
- Bring the cert back to Yubico Authenticator to load onto the Yubikey
- Test the cert out on Bookstack and Keycloak

Provisioning the Yubikey

- From Yubico Authenticator, go to Certificates, then select slot 9a (Authentication)
- On the right side menu, select “Generate key”, which will prompt for your PIN
- On the Generate key box:
  - For Subject, enter “CN=” followed by the username you’re generating a cert for (e.g., below, we’re requesting a cert for user “abc12345”)
  - Output format: Certificate Signing Request (CSR)
  - In the below box, select ECCP384 as the cryptographic signature algorithm


YubiKey 5C NFC
[redacted]

Home
Accounts
Passkeys
Certificates
Slots

Certificates PIV

9a Authentication
No certificate loaded

9c Digital Signature
No certificate loaded

9d Key Management
No certificate loaded

9e Card Authentication
No certificate loaded

Details

Authentication (9a)
No certificate loaded

Actions

Generate key
Generate a new certificate or CSR

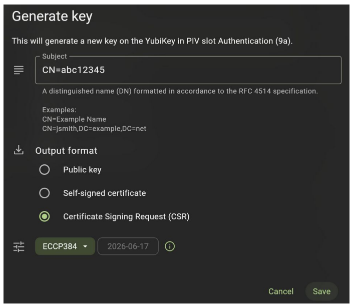
Import file
Import a key and/or certificate

Manage

Change PIN
3 attempts remaining

Change PUK
3 attempts remaining

Management key


Generate key

This will generate a new key on the YubiKey in PIV slot Authentication (9a).

Subject: CN=abc12345

A distinguished name (DN) formatted in accordance to the RFC 4514 specification.

Examples:
CN=Example Name
CN=jsmith,DC=example,DC=net

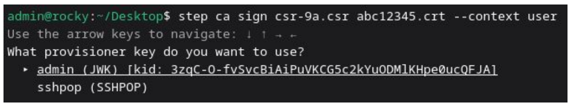
Output format
- Public key
- Self-signed certificate
- [✓ Certificate Signing Request (CSR)]

ECCP384 2026-06-17

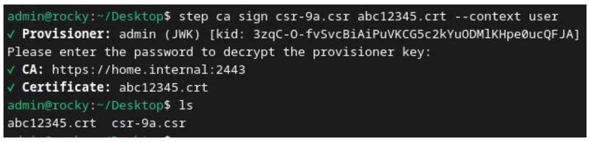
Cancel Save

Provisioning the Yubikey

- Save the CSR file, then transfer it to your Rocky VM
  - This might require jumping to Lab 7.1 to set up port forwarding first
- We'll sign the CSR using step CLI and the user CA context
  - `step ca sign csr-9a.csr [username].crt --context user`
- Transfer the certificate file back to your computer with Yubico Authenticator



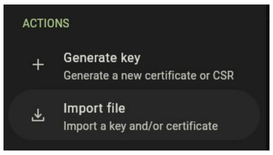
```
admin@rocky:~/Desktop$ step ca sign csr-9a.csr abc12345.crt --context user
Use the arrow keys to navigate: ↓ ↑ → ←
What provisioner key do you want to use?
  ▸ admin (JWK) [kid: 3zqC-O-fvSvcBiAiPuVKCG5c2kYuODMlKHpe0ucQFJA]
    sshpop (SSHPOP)
```

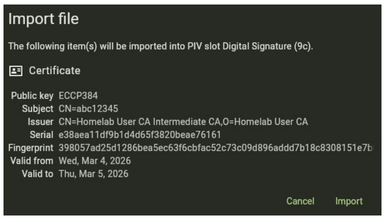


```
admin@rocky:~/Desktop$ step ca sign csr-9a.csr abc12345.crt --context user
✓ Provisioner: admin (JWK) [kid: 3zqC-O-fvSvcBiAiPuVKCG5c2kYuODMlKHpe0ucQFJA]
Please enter the password to decrypt the provisioner key:
✓ CA: https://home.internal:2443
✓ Certificate: abc12345.crt
admin@rocky:~/Desktop$ ls
abc12345.crt  csr-9a.csr
```

Provisioning the Yubikey

- In Yubico Authenticator, return to Certificates -> 9a, and select Import file from the right menu, then select the certificate you just generated
  - (Ignore the 9c slot used in the screenshots, the author did not have a spare Yubikey to use with slot 9a)
- Confirm the import, then your new certificate is ready to use!

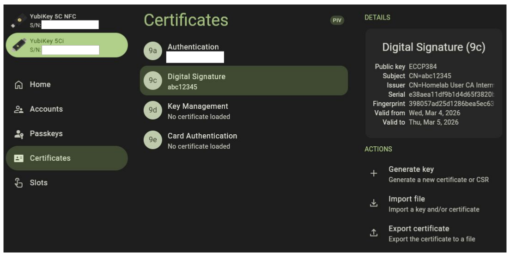

ACTIONS
Generate key
Generate a new certificate or CSR
Import file
Import a key and/or certificate


Import file

The following item(s) will be imported into PIV slot Digital Signature (9c).

Certificate

```
  Public key  ECCP384
     Subject  CN=abc12345
      Issuer  CN=Homelab User CA Intermediate CA,O=Homelab User CA
      Serial  e38aea11df9b1d4d65f3820beae76161
 Fingerprint  398057ad25d1286bea5ec63f6cbfac52c73c09d896addd7b18c8308151e7b
  Valid from  Wed, Mar 4, 2026
    Valid to  Thu, Mar 5, 2026
```



9a Authentication [redacted]
9c Digital Signature abc12345
9d Key Management No certificate loaded
9e Card Authentication No certificate loaded

DETAILS

Digital Signature (9c)

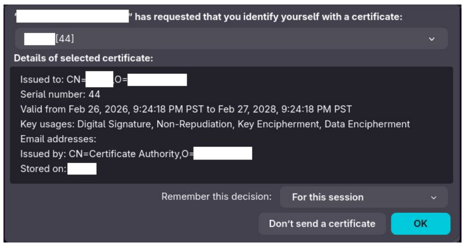
```
  Public key  ECCP384
     Subject  CN=abc12345
      Issuer  CN=Homelab User CA Interm
      Serial  e38aea11df9b1d4d65f3820
 Fingerprint  398057ad25d1286bea5ec63
  Valid from  Wed, Mar 4, 2026
    Valid to  Thu, Mar 5, 2026
```

ACTIONS
Generate key
Generate a new certificate or CSR
Import file
Import a key and/or certificate
Export certificate
Export the certificate to a file

There are some applications that know to look for certificates in other PIV slots, but browsers in general will only pull from 9a.

Using the Yubikey Cert

- To test your Yubikey with Bookstack and Keycloak, you will also need to jump ahead to Lab 7.1 and port forward Caddy to your OPNsense instance's IP
- Most operating systems and browsers will have some form of PIV slot 9a support for cert authentication
  - We won't cover setting this up here, since there are too many OS and browser combinations
- After port forwarding, try browsing to Bookstack or Keycloak with your Yubikey plugged in, and you should now get a prompt to use the cert on your Yubikey!



"[redacted]" has requested that you identify yourself with a certificate:

[redacted][44]

Details of selected certificate:

```
Issued to: CN=[redacted],O=[redacted]
Serial number: 44
Valid from Feb 26, 2026, 9:24:18 PM PST to Feb 27, 2028, 9:24:18 PM PST
Key usages: Digital Signature, Non-Repudiation, Key Encipherment, Data Encipherment
Email addresses:
Issued by: CN=Certificate Authority,O=[redacted]
Stored on:[redacted]
```

Remember this decision: For this session

Compared to when you use the software cert, Firefox now shows the cert stored on the token's name instead of "Software Security Device"
(ignore the redaction, this was one of the author's personal self hosted services to show the Yubikey usage)

As a side note, if you are using a sandboxed browser install on Linux, such as an AppImage, Flatpak, or Snap, it may require additional configuration to access USB devices and/or Udev rules. While the Yubikey presents the cert as a PIV device, it still requires USB permission to access.

Accessing Your Services

Section 7

Accessing Your Services

- Now that you have a selfhosted service at “home”, with an authentication provider, it’s time to expose it to the “internet”
- Of course, during the workshop, that won’t be possible, but we’ll show you how to port forward from your LAN to the WAN, i.e. the network you’re currently connected to
- The brief lab on port forwarding will cover a standard home ISP setup where port forwarding is not blocked
    - There are other options you can consider such as using a VPN back into your home network; we’ll briefly discuss Wireguard on the next slide
    - If your home services are behind NAT, there are options such as using a cloud VPS and a mesh VPN, or a tunneling service

Wireguard VPN

- Wireguard is a fast, modern VPN protocol, with support on many platforms
  - It has native support in the Linux kernel upstream since version 5.6
  - Also a built in remote VPN access option in OpnSense
- OpnSense also supports OpenVPN, but it can be slower due to differences in design (even though both VPN protocols operate over UDP)
- There are other options, such as Tailscale
- OpnSense has built in support for setting up the firewall as a Wireguard server
  - https://docs.opnsense.org/manual/how-tos/wireguard-client.html

Configuring Port Forwarding in the Firewall

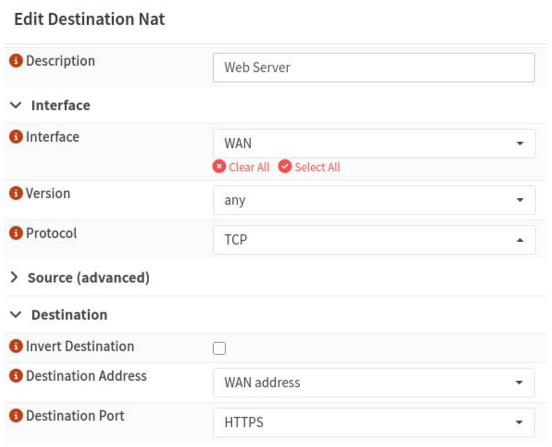
Lab 7.1

Port Forwarding

- A note on port forwarding services - opening up a service on your home IP naturally increases your attack surface
- Adding mTLS like we configured on Caddy mitigates this risk significantly, as it terminates the connection at the transport layer, making scanning and exploitation against the web application (nearly) impossible without compromising your key
- While it’s good protection, keep in mind your own threat model, because even mTLS may not be foolproof

Port Forwarding

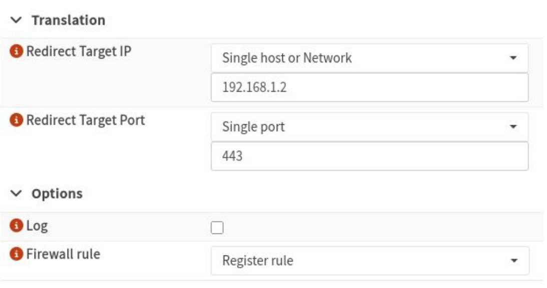
- Log into OpnSense, and go to Firewall -> NAT -> Destination NAT, then add a new rule
- We'll start off by port forwarding your web server:
  - Interface: WAN
  - Version: any
  - Protocol: TCP
  - Destination Address: WAN address
  - Destination Port: HTTPS


Edit Destination Nat

Description	Web Server
Interface	
Interface	WAN
Version	any
Protocol	TCP
Source (advanced)	
Destination	
Invert Destination	☐
Destination Address	WAN address
Destination Port	HTTPS

Port Forwarding

- Continued:
  - Redirect Target IP: 192.168.#.2 (your CoreOS server, # is workshop ID)
  - Redirect Target Port: Single Port, 443
  - Firewall rule: Register rule
- Click Save, then Apply
- Now try browsing to Bookstack or Keycloak from your computer's browser
  - You'll have to add a hosts entry since Caddy will only proxy if you're accessing the hostname, not the IP


Translation

Redirect Target IP	Single host or Network 192.168.1.2
Redirect Target Port	Single port 443

Options

Log	☐
Firewall rule	Register rule

Register rule will automatically create a firewall allow rule for you on the WAN firewall chain so you don't have to manually allow the port through from the outside to the target IP.

Using Nmap to Test Your Network

Lab 7.2

Verifying Home Network Security

- We’ll now test the security considerations that we’ve been building up to this point with mTLS
- Since your service is now port forwarded to the “internet”, you can run a scan against your OPNsense instance to verify that everything is configured properly
- While your attack surface now includes an open port, a potential attacker will not be able to access either Bookstack or Keycloak since they don’t have the certificate to access it
    - Caddy will terminate the connection and not even complete a TLS handshake

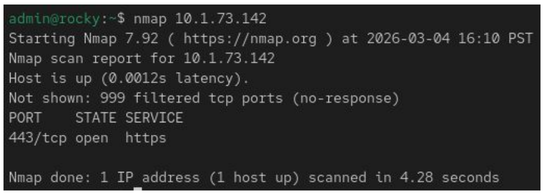
Using Nmap

- Nmap is an open source network scanner, typically used for discovering devices on a network
- Also has a built in script engine and can be used to discover misconfigurations or vulnerabilities in a network
- Used from IT admins to security professionals across the board
- Nmap is best used for scanning TCP ports; due to the nature of UDP, Nmap is unable to reliably scan for open UDP ports, even though it has a toggle for it [1]

[1] https://nmap.org/book/scan-methods-udp-scan.html

Using Nmap

- Grab the WAN IP that OPNsense shows
- First, we're just going to run a standard Nmap scan, which checks the top 1000 ports by default [1] (if you don't pass it any other port parameters otherwise)
  - `nmap [WAN IP]`
- As expected, the only port open is TCP 443 for our web server



```
admin@rocky:~$ nmap 10.1.73.142
Starting Nmap 7.92 ( https://nmap.org ) at 2026-03-04 16:10 PST
Nmap scan report for 10.1.73.142
Host is up (0.0012s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT    STATE SERVICE
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 4.28 seconds
```

When it comes to security on your own home network, it's best to not make any assumptions. You can try a full 1-65534 TCP port scan by running `nmap -p- [IP]`, but we won't do that during the workshop due to the limited infrastructure and potential to slow down the network for everyone here.

[1] https://nmap.org/book/man-port-specification.html

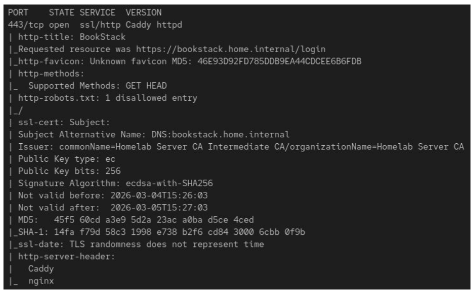
Using Nmap

- Next, we'll try to run a version scan on the open port
  - As previously mentioned, Nmap has a built in script engine with many preexisting scripts (use them with `--script default` or `--script "default AND safe"`)
  - In this case, we will just do a service and version detection (-sV) without running additional scripts [1]
- Since we know the only port open is 443, we'll target just that port
  - `nmap -v -sV -p 443 bookstack.home.internal`
  - -v is verbose output, so you can compare on the next slide what it looks like having mTLS vs on versus not having it on
- You may need to add a temporary hosts entry to be able to scan the domain
  - Caddy reverse proxies based on the hostname required, our configuration won't serve anything for just the IP:443

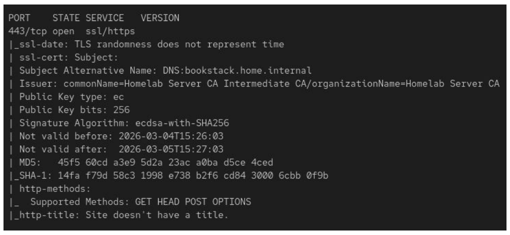
[1] https://nmap.org/book/man-version-detection.html

Using Nmap

- The top screenshot is when mTLS is set to “verify_if_given” mode in Caddy, and the below is how you currently have it set in “require_and verify” mode
- Because the TLS handshake doesn’t complete in “require” mode, Nmap is only able to get basic details about the server cert, but cannot reach the web app like it did in the first screenshot



```
PORT    STATE SERVICE  VERSION
443/tcp open  ssl/http Caddy httpd
| http-title: BookStack
|_Requested resource was https://bookstack.home.internal/login
|_http-favicon: Unknown favicon MD5: 46E93D92FD785DDB9EA44CDCEE6B6FDB
| http-methods:
|_  Supported Methods: GET HEAD
| http-robots.txt: 1 disallowed entry
|_/
| ssl-cert: Subject:
| Subject Alternative Name: DNS:bookstack.home.internal
| Issuer: commonName=Homelab Server CA Intermediate CA/organizationName=Homelab Server CA
| Public Key type: ec
| Public Key bits: 256
| Signature Algorithm: ecdsa-with-SHA256
| Not valid before: 2026-03-04T15:26:03
| Not valid after:  2026-03-05T15:27:03
| MD5:   45f5 60cd a3e9 5d2a 23ac a0ba d5ce 4ced
|_SHA-1: 14fa f79d 58c3 1998 e738 b2f6 cd84 3000 6cbb 0f9b
|_ssl-date: TLS randomness does not represent time
| http-server-header:
|   Caddy
|_  nginx
```



```
PORT    STATE SERVICE   VERSION
443/tcp open  ssl/https
|_ssl-date: TLS randomness does not represent time
| ssl-cert: Subject:
| Subject Alternative Name: DNS:bookstack.home.internal
| Issuer: commonName=Homelab Server CA Intermediate CA/organizationName=Homelab Server CA
| Public Key type: ec
| Public Key bits: 256
| Signature Algorithm: ecdsa-with-SHA256
| Not valid before: 2026-03-04T15:26:03
| Not valid after:  2026-03-05T15:27:03
| MD5:   45f5 60cd a3e9 5d2a 23ac a0ba d5ce 4ced
|_SHA-1: 14fa f79d 58c3 1998 e738 b2f6 cd84 3000 6cbb 0f9b
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Site doesn't have a title.
```

[1] https://nmap.org/book/man-version-detection.html