Warewulf User Guide

Release 4.7.1

Warewulf Project Contributors

Oct 30, 2026

GETTING STARTED

1 Introduction 3

2 Cluster Provisioning 5

3 Network Planning 7

4 Enterprise Linux Quickstart 9

5 SUSE Quickstart 13

6 Debian Quickstart 17

7 Glossary 21

8 Quick Reference 23

9 Feature Status 29

10 Server Installation 31

11 Controlling Warewulf 35

12 Server Configuration 37

13 Server Routes 45

14 Networking 51

15 Using dnsmasq 55

16 Security 57

17 Bootloaders 59

18 Upgrading Warewulf 65

19 REST API 67

20 Cluster Nodes 69

21 Node Profiles 75

22 Network Interfaces 81

23 IPMI 83
24 Provisioning disks 87
25 OS Images 95
26 Image Kernels 103
27 Syncuser 105
28 SELinux-enabled Images 107
29 Overlays 109
30 Templates 119
31 Troubleshooting 123
32 Known issues 129
33 Contributing 133
34 Development Environment 135
35 Documentation 137
36 Debugging 139
37 wwctl 141
38 wwctl clean 143
39 wwctl configure 145
40 wwctl configure dhcp 147
41 wwctl configure hostfile 149
42 wwctl configure nfs 151
43 wwctl configure ssh 153
44 wwctl configure tftp 155
45 wwctl configure tls 157
46 wwctl configure warewulfd 159
47 wwctl image 161
48 wwctl image build 163
49 wwctl image copy 165
50 wwctl image delete 167
51 wwctl image exec 169
52 wwctl image import 171

53 wwctl image kernels 173
54 wwctl image list 175
55 wwctl image rename 177
56 wwctl image shell 179
57 wwctl image show 181
58 wwctl image syncuser 183
59 wwctl node 185
60 wwctl node add 187
61 wwctl node console 189
62 wwctl node delete 191
63 wwctl node edit 193
64 wwctl node export 195
65 wwctl node import 197
66 wwctl node list 199
67 wwctl node sensors 201
68 wwctl node set 203
69 wwctl node status 205
70 wwctl node unset 207
71 wwctl overlay 209
72 wwctl overlay build 211
73 wwctl overlay chmod 213
74 wwctl overlay chown 215
75 wwctl overlay create 217
76 wwctl overlay delete 219
77 wwctl overlay edit 221
78 wwctl overlay import 223
79 wwctl overlay info 225
80 wwctl overlay list 227
81 wwctl overlay mkdir 229
82 wwctl overlay show 231

83 wwctl power 233
84 wwctl power cycle 235
85 wwctl power off 237
86 wwctl power on 239
87 wwctl power reset 241
88 wwctl power soft 243
89 wwctl power status 245
90 wwctl profile 247
91 wwctl profile add 249
92 wwctl profile delete 251
93 wwctl profile edit 253
94 wwctl profile list 255
95 wwctl profile set 257
96 wwctl profile unset 259
97 wwctl server 261
98 wwctl ssh 263
99 wwctl upgrade 265
100 wwctl upgrade config 267
101 wwctl upgrade nodes 269
102 wwctl version 271
103 v4.6.0 Release Notes 273
104 v4.6.1 Release Notes 281
105 v4.6.2 Release Notes 283
106 v4.6.3 Release Notes 285
107 v4.6.4 Release Notes 287
108 v4.6.5 Release Notes 289
109 v4.7.0 Release Notes 291
110 v4.7.1 Release Notes 295

Welcome to the Warewulf User Guide!

CHAPTER

ONE

INTRODUCTION

Warewulf is an operating system provisioning platform for Linux clusters. Since its initial release in 2001, Warewulf has become the most popular open source and vendor-agnostic provisioning system within the global HPC community. Warewulf is known for its massive scalability and simple management of stateless (disk optional) provisioning.

Warewulf leverages a simple administrative model centralizing administration around virtual OS images which are used to provision cluster nodes. This means you can have hundreds or thousands of cluster nodes all booting and running on the same OS image. As of Warewulf v4, the OS image can be managed using industry-standard container tooling and/or CI/CD pipelines. This can be as simple as DockerHub or your own private GitLab CI infrastructure. With this architecture, Warewulf combines the best of High Performance Computing (HPC), Cloud, Hyperscale, and Enterprise deployment principles to create and maintain large scalable stateless clusters.

Warewulf is used most prominently in High Performance Computing (HPC) clusters, but its architecture is flexible enough to be used in most any clustered Linux environment, including clustered web servers, rendering farms, and even Kubernetes and cloud deployments.

1.1 Warewulf design

Warewulf has had a number of iterations since its inception in 2001, but its design tenets have always remained the same: a simple, scalable, stateless, and flexible provisioning system for all types of clusters.

- **Lightweight**: Warewulf provisions stateless operating system images and then gets out of the way. There are no underlying system dependencies or requisite changes to the provisioned cluster node operating system.
- **Simple**: Warewulf is used by hobbyists, researchers, scientists, engineers and systems administrators alike.
- **Flexible**: Warewulf can address the needs of any environment–from a computer lab with graphical workstations, to under-the-desk clusters, to supercomputing centers providing HPC services to thousands of users.
- **Agnostic**: From the Linux distribution of choice to the underlying hardware, Warewulf is agnostic and standards compliant. From ARM to x86, Atos to Dell, Debian, SUSE, Rocky, CentOS, and RHEL, Warewulf can be used in most any environment.
- **Secure**: Warewulf supports SELinux out-of-the-box. Just install SELinux in your OS image and let Warewulf do the rest!
- **Open Source**: Warewulf is and has always been open source. It can be used in any environment, whether public, private, non-profit, or commercial. And the Warewulf project is always welcoming of contribution from its community of users, with major features often beginning as external contributions.

1.2 Warewulf architecture

Warewulf v4 has a simple but flexible base architecture:

A **Warewulf server** stores information about the cluster and the nodes in it, and provides a command-line interface (wwctl) for managing nodes, their images, and their overlays.

Cluster nodes are defined in a flexible YAML file, including their network configuration and image and overlay assignments.

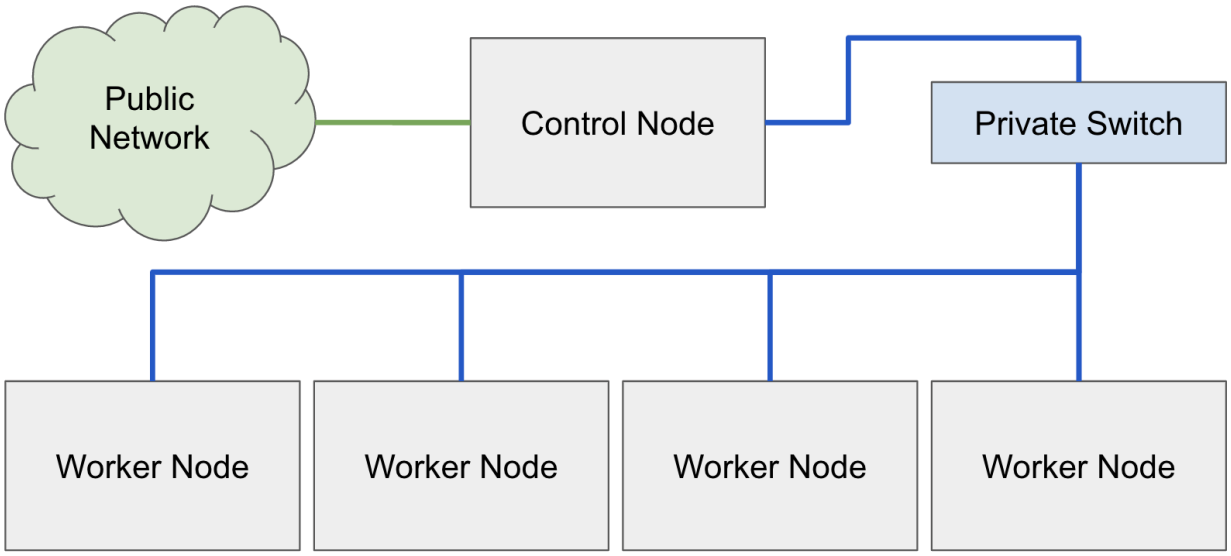
Node profiles provide a flexible abstraction for applying configuration to multiple nodes.

Operating system images provide a bootable image, including the kernel that will be used to boot the cluster node. OS images provide the base operating system and, by default, run entirely in memory. This means that when you reboot the node, the node retains no information about Warewulf or how it booted; but it also means that they return to their initial known-good state.

Overlays customize the provisioned operating system image with static files and dynamic templates applied with the OS image and, optionally, periodically at runtime.

1.3 Beowulf overview

Warewulf is designed to support the original Beowulf Cluster concept. (Thus its name, a soft**WARE** implementation of the beo**WULF**.) The architecture is characterized by a group of similar cluster nodes all connected together using standard commodity equipment on an internal cluster network. The server node (often historically referred to as the "master" or "head" node) is "dual homed" (i.e., it has two network interfaces) with one of these network interfaces attached to an external network and the other connected to the internal cluster network.



This simple topology is the foundation for creating a scalable HPC cluster resource. Even today, almost 30 years after the inception of this architecture, this is the baseline architecture that virtually all HPC systems are built to.

An HPC cluster often includes dedicated storage, scheduling and resource management, monitoring, interactive systems, and other components. For smaller systems, many of these components can be deployed to a single server node; but, as the system scales, it may be better to have groups of nodes dedicated to these different services.

Warewulf is flexible enough to start with a simple "head node" Beowulf style cluster deployment and to grow as needs for the cluster and its environment change.

CHAPTER

TWO

CLUSTER PROVISIONING

Clusters have many scalability factors to consider. Often overlooked among them is "administrative scaling"– the systems administration overhead of a person or team maintaining a large number of systems. While homogeneous configurations do improve administrative scaling, each installed server is still subject to version and configuration drift, eventually becoming a point of discrete administration and debugging. The larger the cluster, the harder this problem is to solve.

This is the problem that Warewulf was created to solve.

2.1 Provisioning Overview

Provisioning is the process of preparing a system for use, typically by providing and configuring an operating system. There are many ways to accomplish this, from copying hard drives, to scripted installs, to automated installs. Each has its place, and there are many tools available to facilitate each method.

Before dedicated cluster provisioning systems, administrators would visit each cluster node and install it from scratch, with an ISO, CD, or USB flash drive. This is obviously not scalable. Because the nodes in a cluster environment are typically identical, it is much more efficient to group sets of nodes together to be provisioned in bulk.

2.2 Why Stateless Provisioning

Warewulf further improves on the automated provisioning process by skipping the installation completely; it boots directly into the runtime operating system without ever doing an installation.

Stateless provisioning means you never have to install another compute node. Think of it like booting a LiveOS or LiveISO on nodes over the network. This means that no node requires discrete administration, but rather the entire cluster is administrated as a single unit. There is no version drift, because it is not possible for nodes to fall out of sync. Every reboot makes it exactly the same as its neighbors.

2.3 Cluster Node Requirements

The only requirement to provision a node with Warewulf is that the node is set to PXE boot. You may need to change the boot order if there is a local disk present and bootable. This is a configuration change you will have to make in the BIOS of the cluster node.

This configuration is different for each vendor platform. For more information, consult your system documentation or contact your hardware vendor support.

Note

Hardware vendors are often able to preconfigure your cluster nodes with values of your choosing. Ask them to provide a text file that includes all of the network interface MAC addresses of the clusters nodes in the order they are racked–this simplifies the process of adding nodes to Warewulf.

2.4 The Provisioning Process

When a cluster node boots from Warewulf, the following process occurs:

1. The system firmware (either BIOS or UEFI) initializes hardware, including local network interfaces.
2. The system uses an in-firmware PXE client to obtain a BOOTP/DHCP address from the network.
3. The DHCP server (hosted either on the Warewulf server or externally) responds with an address suitable for provisioning, along with a "next-server" option directing the cluster node to download (via TFTP) and execute a bootloader (either iPXE or GRUB) with a Warewulf-provided configuration.
4. The bootloader configuration directs the cluster node to download and bootstrap the configured kernel, image, and overlays from the Warewulf (HTTP) server.
    - In a single-stage provisioning configuration, the desired image and overlays are combined and provisioned immediately by the bootloader as the kernel's initial root file system. This is straightforward, but does not work in all environments: some systems have memory layouts that are not handled properly by either iPXE or GRUB for sufficiently large image sizes, leading to strange, unpredictable results.
    - In a two-stage provisioning configuration, a small initial root fs (created by dracut) is provisioned first, and this image uses the provisioned Linux kernel to retrieve and deploy the full image and overlays. Perhaps counter-intuitively, the two-stage provisioning process is often quicker than the single-stage process, because the Linux environment is more I/O efficient than the bootloader itself.
5. Optionally included in a configured overlay, `wwclient` is left resident on the cluster node and periodically refreshes configured runtime overlays.

CHAPTER

THREE

NETWORK PLANNING

A clustered resource depends on a cluster network. This network can be either persistent (it is always "up" even after provisioning) or temporary, only used for provisioning and/or out of band system control and management (e.g., IPMI).

The cluster network must be dedicated to the cluster because Warewulf uses network services (particularly DHCP) which may conflict with services on another mixed-use network. A dedicated cluster network is also important for security, as the cluster network often has an implicit level of trust associated with it.

The Warewulf server is often "dual homed," meaning that it has separate network interfaces connected to each of the cluster network and an external network. But it is also possible for the cluster network to be routable from other, more general-purpose networks.

Many clusters have more than one internal network. This is common for performance critical HPC clusters that implement a high speed and low latency network like InfiniBand. In this case, this network is used for high speed data transfers for inter-process communication between compute nodes and file system IO.

Warewulf will need to be configured to use the private cluster management network. Warewulf will use this network for booting the nodes over PXE. There are three network protocols used to accomplish this DHCP/BOOT, TFTP, and HTTP on port 9873. Warewulf will use the operating system's provided version of DHCP (ISC-DHCP) and TFTP for the PXE bootstrap to iPXE, and then iPXE will use Warewulf's internal HTTP services to transfer the larger files for provisioning.

3.1 Addressing

The addressing scheme of your private cluster network is 100% up to the system integrator, but for large clusters, many organizations like to organize the address allocations. Below is a recommended IP addressing scheme which we will use for the rest of this document.

- 10.0.0.1: Private network address IP
- 255.255.252.0: Private network subnet mask ($10.0.0.0/22$)

Here is an example of how the cluster's address can be divided for a 255 node cluster:

- 10.0.0.1 - 10.0.0.255: Cluster infrastructure including this host, schedulers, file systems, routers, switches, etc.
- 10.0.1.1 - 10.0.1.255: DHCP range for booting nodes
- 10.0.2.1 - 10.0.2.255: Static node addresses
- 10.0.3.1 - 10.0.3.255: IPMI and/or out of band addresses for the compute nodes

CHAPTER

FOUR

ENTERPRISE LINUX QUICKSTART

Deploying Warewulf for Rocky Linux, CentOS, RHEL, and other related distributions.

4.1 Install Warewulf

The preferred way to install Warewulf on Enterprise Linux is using the RPMs published in GitHub releases. For example, to install the v4.7.1 release on Enterprise Linux 9:

```
dnf install https://github.com/warewulf/warewulf/releases/download/v4.7.1/warewulf-4.7.1-1.el9.x86_64.
↪rpm
```

Packages are available for el8 and el9.

4.1.1 Install Warewulf from source

If you prefer, you can also install Warewulf from source.

```
dnf install git
dnf install epel-release
dnf install golang {libassuan,gpgme}-devel unzip tftp-server dhcp-server nfs-utils ipxe-bootimgs-{x86,
↪aarch64}

git clone https://github.com/warewulf/warewulf.git
cd warewulf
PREFIX=/usr/local make defaults
make install
```

Note

Some packages, like libassuan-devel and gpgme-devel, require either PowerTools (EL8) or CodeReady Builder (EL9) repositories.

```
dnf config-manager --set-enabled PowerTools # EL8
dnf config-manager --set-enabled crb # EL9
```

4.2 Configure firewalld

Restart firewalld to register the added service file, add the service to the default zone, and reload.

```
systemctl restart firewalld
firewall-cmd --permanent --add-service=warewulf
firewall-cmd --permanent --add-service=dhcp
firewall-cmd --permanent --add-service=nfs
firewall-cmd --permanent --add-service=tftp
firewall-cmd --reload
```

4.3 Configure Warewulf

Edit the file /etc/warewulf/warewulf.conf and ensure that you've set the appropriate configuration parameters. Here are some of the defaults for reference assuming that 10.0.0.1/22 is the IP address of your cluster's private network interface.

```
ipaddr: 10.0.0.1
netmask: 255.255.252.0
network: 10.0.0.0
warewulf:
  port: 9873
  secure: false
  update interval: 60
  autobuild overlays: true
  host overlay: true
  datastore: /usr/share
  grubboot: false
dhcp:
  enabled: true
  template: default
  range start: 10.0.1.1
  range end: 10.0.1.255
  systemd name: dhcpd
tftp:
  enabled: true
  tftproot: /var/lib/tftpboot
  systemd name: tftp
  ipxe:
    "00:00": undionly.kpxe
    "00:07": ipxe-snponly-x86_64.efi
    "00:09": ipxe-snponly-x86_64.efi
    00:0B: arm64-efi/snponly.efi
nfs:
  enabled: true
  export paths:
  - path: /home
    export options: rw,sync
  - path: /opt
    export options: ro,sync,no_root_squash
  systemd name: nfs-server
image mounts:
- source: /etc/resolv.conf
  dest: /etc/resolv.conf
  readonly: true
paths:
```

(continues on next page)

(continued from previous page)

```
bindir: /usr/bin
sysconfdir: /etc
localstatedir: /var/lib
ipxesource: /usr/share/ipxe
srvdir: /var/lib
firewalldir: /usr/lib/firewalld/services
systemddir: /usr/lib/systemd/system
wwoverlaydir: /var/lib/warewulf/overlays
wwchrootdir: /var/lib/warewulf/chroots
wwprovisiondir: /var/lib/warewulf/provision
wwclientdir: /warewulf
```

Note

The DHCP range from 10.0.1.1 to 10.0.1.255 is dedicated for DHCP during node boot and should not overlap with any static IP address assignments.

4.4 Enable and start the Warewulf service

Warewulf provides a service, warewulfd, which responds to node boot requests.

```
systemctl enable --now warewulfd
```

4.5 Configure system services automatically

There are a number of services and configurations that Warewulf relies on to operate. You can configure all such services with wwctl configure --all.

```
wwctl configure --all
```

Note

If you just installed the system fresh and have SELinux enforcing, you may need to run restorecon -Rv /var/lib/tftpboot/ to label files written to tftpboot.

4.6 Add an OS image

This will pull a basic OS image from Docker Hub and set it for the "default" node profile.

```
wwctl image import docker://ghcr.io/warewulf/warewulf-rockylinux:9 rockylinux-9 --build
wwctl profile set default --image rockylinux-9
```

4.7 Configure the default node profile

In this example, all nodes share the netmask and gateway configuration, so we can set them in the default profile.

```
wwctl profile set -y default --netmask=255.255.252.0 --gateway=10.0.0.1
wwctl profile list
```

4.8 Add a node

Adding nodes can be done while setting configurations in one command. Here we set the IP address of the default interface; and setting the node to be discoverable causes the HW address to be added to the configuration as the node boots.

Node names must be unique. If you are managing multiple clusters with overlapping names, distinguish them using dot notation.

```
wwctl node add n1 --ipaddr=10.0.2.1 --discoverable=true
wwctl node list -a n1
```

The full node configuration comes from both cascading profiles and node configurations which always supersede profile configurations.

4.9 Build overlays

The default configuration should cause node overlays to be built automatically when they are required; but you can build them explicitly, just to be sure.

Warning

Overlay autobuild has been broken at various times prior to v4.5.6; so it's a reasonable practice to rebuild overlays manually after changes to the cluster.

```
# you can also supply an `n1` argument to build for the specific node
wwctl overlay build
```

4.10 Boot

Turn on your compute node and watch it boot!

CHAPTER

FIVE

SUSE QUICKSTART

Deploying Warewulf for openSUSE Leap and SLES 15.

5.1 Install Warewulf and dependencies

```
sudo zypper install -t pattern devel_basis
sudo zypper install go
sudo zypper install tftp dhcp-server nfs-kernel-server

sudo systemctl stop firewalld
sudo systemctl disable firewalld

git clone https://github.com/warewulf/warewulf.git
cd warewulf
PREFIX=/usr SYSCONFDIR=/etc TFTPDIR=/srv/tftproot LOCALSTATEDIR=/var/lib make clean␣
↪defaults
make all
sudo make install
```

The standard configuration template for the dhcpd service is installed at the wrong location, you have to fix this with

```
mv /var/lib/warewulf/overlays/host/etc/dhcp/dhcpd.conf.ww /var/lib/warewulf/overlays/host/etc/
↪dhcpd.conf.ww
```

5.2 Install Warewulf from the open build service

You can also just install the 'warewulf4' package with zypper from the openbuild service. Up to date versions are available on the devel project

https://build.opensuse.org/project/show/network:cluster

5.3 Configure the controller

Edit the file /etc/warewulf/warewulf.conf and ensure that you've set the appropriate configuration parameters. Here are some of the defaults for reference assuming that 192.168.200.1 is the IP address of your cluster's private network interface:

```
ipaddr: 192.168.200.1
netmask: 255.255.255.0
```

(continues on next page)

(continued from previous page)

```
network: 192.168.200.0
warewulf:
  port: 9873
  secure: false
  update interval: 60
  autobuild overlays: true
  host overlay: true
dhcp:
  enabled: true
  range start: 192.168.200.50
  range end: 192.168.200.99
  systemd name: dhcpd
tftp:
  enabled: true
  systemd name: tftp
nfs:
  enabled: true
  export paths:
  - path: /home
    export options: rw,sync
  - path: /opt
    export options: ro,sync,no_root_squash
  systemd name: nfs-server
image mounts:
  - source: /etc/resolv.conf
    dest: /etc/resolv.conf
    readonly: true
```

Note

The DHCP range ends at 192.168.200.99 and as you will see below, the first node static IP address (post boot) is configured to 192.168.200.100.

5.4 Start and enable the Warewulf service

```
# Start and enable the warewulfd service
sudo systemctl enable --now warewulfd
```

5.5 Configure system services automatically

There are a number of services and configurations that Warewulf relies on to operate. If you wish to configure all services, you can do so with the following command. Running wwctl configure individually (omitting the --all) will print help and usage instructions.

Note

If the dhcpd service was not used before you will have to add the interface on which the cluster network is running to the DHCP_INTERFACE in the file /etc/sysconfig/dhcpd.

```
sudo wwctl configure --all
```

5.6 Pull and build the image

This will pull a basic image from Docker Hub and set it in the "default" node profile.

```
$ sudo wwctl image import docker://registry.opensuse.org/science/warewulf/leap-15.4/containers/
↪kernel:latest leap15.4
$ sudo wwctl profile set default --image leap15.4
```

5.7 Set up the default node profile

The --setdefault arguments above will automatically set those entries in the default profile, but if you wanted to set them by hand to something different, you can do the following:

```
sudo wwctl profile set -y -C leap15.4
```

Next we set some default networking configurations for the first ethernet device. On modern Linux distributions, the name of the device is not critical, as it will be setup according to the HW address. Because all nodes will share the netmask and gateway configuration, we can set them in the default profile as follows:

```
sudo wwctl profile set -y default --netname default --netmask 255.255.255.0 --gateway 192.168.200.1
sudo wwctl profile list -a
```

5.8 Add a node

Adding nodes can be done while setting configurations in one command. Here we are setting the IP address of eth0 and setting this node to be discoverable, which will then automatically have the HW address added to the configuration as the node boots.

Node names must be unique. If you have node groups and/or multiple clusters, designate them using dot notation.

Note that the full node configuration comes from both cascading profiles and node configurations which always supersede profile configurations.

```
sudo wwctl node add n0000.cluster --netdev eth0 --ipaddr 192.168.200.100 --discoverable true
sudo wwctl node list -a n0000.cluster
```

5.9 Warewulf Overlays

There are two types of overlays: system and runtime overlays.

System overlays are provisioned to the node before /sbin/init is called. This enables us to prepopulate node configurations with content that is node specific like networking and service configurations.

Runtime overlays are re-applied periodically during the normal runtime of the node. Because these overlays are provisioned at periodic intervals, they are very useful for content that changes, like users and groups.

Overlays are generated from a template structure that is viewed using the wwctl overlay commands. Files that end in the .ww suffix are templates and abide by standard text/template rules. This supports loops, arrays, variables, and functions making overlays extremely flexible.

All overlays are compiled before being provisioned. This accelerates the provisioning process because there is less to do when nodes are being managed at scale.

Here are some of the common overlay commands:

```
sudo wwctl overlay list -l
sudo wwctl overlay list -ls
sudo wwctl overlay edit default /etc/hello_world.ww
sudo wwctl overlay build -a
```

Boot your compute node and watch it boot!

CHAPTER

SIX

DEBIAN QUICKSTART

Deploying Warewulf for Debian 12.

6.1 Install the basic services

```
sudo apt install firewalld nfs-kernel-server tftpd-hpa isc-dhcp-server
```

Note

If you get an error message concerning *isc-dhcp-server.service* you probably need to configure the network interface that isc-dhcp-server will listen to. Run sudo dpkg-reconfigure isc-dhcp-server and enter the name of your cluster's private network interface (e.g. enp2s0). After that, you might also need to run sudo systemctl enable isc-dhcp-server.

6.2 Install Warewulf and dependencies

```
sudo apt install build-essential curl unzip

sudo apt install git golang libnfs-utils libgpgme-dev libassuan-dev

mkdir ~/git
cd ~/git
git clone https://github.com/warewulf/warewulf.git
cd warewulf
git checkout main # or switch to a tag like 'v4.7.1'
make all && sudo make install
```

6.3 Configure firewalld

Restart firewalld to register the added service file, add the service to the default zone, and reload.

```
sudo systemctl restart firewalld
sudo firewall-cmd --permanent --add-service warewulf
sudo firewall-cmd --permanent --add-service dhcp
sudo firewall-cmd --permanent --add-service nfs
sudo firewall-cmd --permanent --add-service tftp
sudo firewall-cmd --reload
```

6.4 Configure the controller

Edit the file /etc/warewulf/warewulf.conf and ensure that you've set the appropriate configuration parameters. Here are some of the defaults for reference assuming that 192.168.200.1 is the IP address of your cluster's private network interface:

```
ipaddr: 192.168.200.1
netmask: 255.255.255.0
network: 192.168.200.0
warewulf:
  port: 9873
  secure: false
  update interval: 60
  autobuild overlays: true
  host overlay: true
dhcp:
  enabled: true
  range start: 192.168.200.50
  range end: 192.168.200.99
  systemd name: isc-dhcp-server
tftp:
  enabled: true
  systemd name: tftpd-hpa
nfs:
  enabled: true
  export paths:
  - path: /home
        export options: rw,sync
  - path: /opt
        export options: ro,sync,no_root_squash
  systemd name: nfs-server
```

Note

The DHCP range ends at 192.168.200.99 and as you will see below, the first node static IP address (post boot) is configured to 192.168.200.100.

6.5 Start and enable the Warewulf service

```
# Start and enable the warewulfd service
sudo systemctl enable --now warewulfd
```

6.6 Configure system services automatically

There are a number of services and configurations that Warewulf relies on to operate. If you wish to configure all services, you can do so with the following command. Running wwctl configure individually (omitting the --all) will print help and usage instructions.

```
sudo wwctl configure --all
```

Note

If you just installed the system fresh and have SELinux enforcing, you may need to reboot the system at this stage to properly set the contexts of the TFTP contents. After rebooting, you might also need to run $ sudo restorecon -Rv /var/lib/tftpboot/ if there are errors with TFTP still.

6.7 Pull and build the image

This will pull a basic image from Docker Hub and set it for the "default" node profile.

```
sudo wwctl image import --build docker://ghcr.io/warewulf/warewulf-debian:12.0 debian-12.0
sudo wwctl profile set default --image=debian-12.0
```

6.8 Set up the default node profile

Node configurations can be set via node profiles. Each node by default is configured to be part of the default node profile, so any changes you make to that profile will affect all nodes.

The following command will set the image we just imported above to the default node profile:

```
sudo wwctl profile set --yes --image debian-12.0 "default"
```

Next we set some default networking configurations for the first ethernet device. On modern Linux distributions, the name of the device is not critical, as it will be setup according to the HW address. Because all nodes will share the netmask and gateway configuration, we can set them in the default profile as follows:

```
sudo wwctl profile set --yes --netdev eth0 --netmask 255.255.255.0 --gateway 192.168.200.1 "default"
```

Once those configurations have been set, you can view the changes by listing the profiles as follows:

```
sudo wwctl profile list -a
```

6.9 Add a node

Adding nodes can be done while setting configurations in one command. Here we are setting the IP address of eth0 and setting this node to be discoverable, which will then automatically have the HW address added to the configuration as the node boots.

Node names must be unique. If you have node groups and/or multiple clusters, designate them using dot notation.

Note that the full node configuration comes from both cascading profiles and node configurations which always supersede profile configurations.

```
sudo wwctl node add n0000.cluster --ipaddr 192.168.200.100 --discoverable
```

At this point you can view the basic configuration of this node by typing the following:

```
sudo wwctl node list -a n0000.cluster
```

To make node changes effective, it is a good practice to update Warewulf overlays with the following command:

```
sudo wwctl overlay build
```

Now, turn on your compute node and watch it boot!

CHAPTER

SEVEN

GLOSSARY

Cluster network
: A dedicated network for the Warewulf cluster. Used for provisioning communication between cluster nodes and the Warewulf server.

External services
: The Warewulf server can configure external services to support the provisioning process. For example, the Warewulf server typically deploys and configures a DHCP server (either ISC DHCP or dnsmasq) and a TFTP server.

Image
: See "Operating System (OS) Image" or "Overlay Image."

Kernel
: In addition to an image, Warewulf also requires a kernel (typically a Linux kernel) in order to provision a node.

  Warewulf (after v4.3.0) automatically provisions a kernel detected and extracted from the image itself. In most cases, kernels may be installed in the image using normal system packages, and no special consideration is necessary.

Node
: Warewulf nodes are the systems that are being provisioned by Warewulf. The roles of these systems could be "compute", "storage", "GPU", "IO", etc.

nodes.conf
: One of two primary Warewulf configuration files, `nodes.conf` is a YAML document which records all configuration parameters for Warewulf's nodes and profiles. It does not contain the images or overlays, but refers to them by name.

  This file is sometimes referred to as the "nodes database" or "node registry."

Operating System (OS) Image
: An operating system image (or OS image) contains a bootable operating system. When provisioning a cluster node, Warewulf combines the OS image with the node's overlay images to provision the complete, configured image.

  OS images may be imported from OCI image registries, OCI image archives, Apptainer sandboxes, and manual chroot directories.

Overlay
: Warewulf overlays provide customization for the provisioned image. Overlays may be configured on nodes or profiles, as either **system** or **runtime** overlays.

  System overlays are applied only once, when a node is first provisioned.

  Runtime overlays are applied when a node is first provisioned and periodically during the runtime of the node. (The default period is 1 minute.)

Warewulf includes a number of **distribution overlays**; but additional **site overlays** can be added to a Warewulf environment.

Overlay Image
An overlay image contains the rendered contents of multiple overlays for a given cluster node. Nodes have a "system overlay" image and a "runtime overlay" image, based on the system and runtime overlays associated with each node.

Overlay images are not managed directly, but are built by the `wwctl overlay build` command.

Profile
Warewulf profiles are abstract nodes that carry the same configuration attributes but do not provision any specific node. Warewulf nodes may then refer to one or more such profiles for their configuration. In this way, profiles provide a simple mechanism for applying configuration to a group of nodes, and this configuration may be mixed with configuration from other profiles.

Server, Warewulf
The Warewulf controller runs the Warewulf daemon (`warewulfd`) and is responsible for the management, control, and administration of the cluster. This system may also sometimes be referred to as the "master," "head," or "admin" node.

A typical Warewulf controller also runs a DHCP service and a TFTP service, and often an NFS service; though these services may be managed separately and on separate servers.

Two-stage boot
A two-stage boot uses an intermediate image (often called "initrd," or "initramfs") to initialize hardware and load the final image. This contrasts with Warewulf's default "single stage" behavior, which effectively uses the final image as a large initial root file system.

The Warewulf two-stage boot process currently supports Dracut-based images.

warewulf.conf
One of two primary Warewulf configuration files, `warewulf.conf` is a YAML document which records all configuration parameters for the Warewulf server and its optional subservices.

wwclient
Warewulf adds a `wwclient` daemon to provisioned nodes. This daemon is responsible for periodically fetching and applying runtime overlays.

wwctl
The main administrative interface for Warewulf is the `wwctl` command, which provides commands to manage nodes, profiles, images, overlays, kernels, and more.

wwinit
Warewulf performs some setup during the provisioning process before control is passed to the provisioned operating system. This process is referred to as "wwinit," and is implemented and configured by a script and overlay of the same name.

CHAPTER

EIGHT

QUICK REFERENCE

A quick reference for common day-to-day Warewulf operations.

8.1 Important paths

```
# Configuration
/etc/warewulf/warewulf.conf       # Main server configuration
/etc/warewulf/nodes.conf          # Node database
/etc/warewulf/auth.conf           # API authentication

# Images (container filesystems)
/var/lib/warewulf/chroots/        # Image root directories

# Overlays
/usr/share/warewulf/overlays/     # Distribution-provided overlays
/var/lib/warewulf/overlays/       # Site-local overlays
/var/lib/warewulf/provision/      # Built overlay images (per-node)

# Logs
journalctl -u warewulfd           # Warewulf daemon logs
```

8.2 Server management

```
# Configure all external services (TLS, warewulfd, TFTP, DHCP, NFS, SSH, hosts)
wwctl configure --all

# Configure individual services
wwctl configure dhcp
wwctl configure tftp
wwctl configure nfs
wwctl configure ssh
wwctl configure hostfile

# Restart the Warewulf daemon
systemctl restart warewulfd
```

8.3 Listing nodes and profiles

```
# List all nodes (summary)
wwctl node list

# List nodes with network interface details
wwctl node list --net

# List nodes with all fields (includes unset/inherited values)
wwctl node list --all n1

# List nodes with IPMI settings
wwctl node list --ipmi

# List specific nodes using hostlist syntax
wwctl node list n[1-10]

# Show current node boot status
wwctl node status

# List all profiles
wwctl profile list

# List all fields of a profile (includes inherited values)
wwctl profile list default --all
```

8.4 Adding a node

```
# Add a single node with an IP address
wwctl node add n1 --ipaddr=10.0.2.1

# Add a range of nodes (IP address auto-increments)
wwctl node add n[2-4] --ipaddr=10.0.2.2

# Add a node with multiple options: image, network interface, kernel args, and profile
wwctl node add n1 \
  --ipaddr=10.0.2.1 \
  --netmask=255.255.255.0 \
  --netdev=eno1 \
  --hwaddr=00:00:00:00:00:01 \
  --image=rockylinux-9 \
  --profile=default \
  --kernelargs="quiet crashkernel=no"

# Un-set a field (revert to profile/default value)
wwctl node unset n1 --image
```

8.5 Network interfaces

```
# Set the primary network interface
wwctl node set n1 \
  --netdev=eno1 \
  --hwaddr=00:00:00:00:00:01 \
  --ipaddr=10.0.2.1 \
  --netmask=255.255.255.0

# Add a secondary network (e.g. InfiniBand)
wwctl node set n1 \
  --netname=infiniband \
  --type=infiniband \
  --netdev=ib0 \
  --ipaddr=10.0.3.1 \
  --netmask=255.255.255.0

# Configure a VLAN interface
wwctl node set n1 \
  --netname=vlan42 \
  --netdev=vlan42 \
  --type=vlan \
  --ipaddr=10.0.42.1 \
  --netmask=255.255.252.0 \
  --nettagadd="vlan_id=42,parent_device=eth0"

# Set DNS on an interface
wwctl node set n1 --nettagadd="DNS1=1.1.1.1"
```

8.6 Images

```
# Import an image from Docker Hub (or any OCI registry)
wwctl image import docker://ghcr.io/warewulf/warewulf-rockylinux:9 rockylinux-9

# Import with registry credentials
wwctl image import \
  --username myuser --password mysecret \
  docker://registry.example.com/myimage:latest myimage

# Import from a local directory or tarball
wwctl image import ./rockylinux-9/ rockylinux-9
wwctl image import rockylinux-9.tar rockylinux-9

# List available images
wwctl image list

# Open an interactive shell inside an image
wwctl image shell rockylinux-9

# Run a single command inside an image (e.g. install packages)
wwctl image exec rockylinux-9 -- dnf -y install vim htop
```

(continues on next page)

(continued from previous page)

```
# Rebuild the image archive (after making changes)
wwctl image build rockylinux-9

# Assign an image to a node or profile
wwctl node set n1 --image=rockylinux-9
wwctl profile set default --image=rockylinux-9
```

8.7 IPMI

```
# Configure IPMI settings on the default profile
wwctl profile set default \
  --ipminetmask=255.255.255.0 \
  --ipmiuser=admin \
  --ipmipass=passw0rd \
  --ipmiinterface=lanplus \
  --ipmiwrite

# Set the IPMI address on a specific node
wwctl node set n1 --ipmiaddr=192.168.2.1

# List IPMI settings
wwctl node list --ipmi

# Power commands
wwctl power status n[1-10]
wwctl power on n1
wwctl power off n1
wwctl power cycle n1
wwctl power reset n1

# Open serial-over-LAN console
wwctl node console n1
```

8.8 Overlays

```
# List all available overlays
wwctl overlay list

# Build overlays for all nodes
wwctl overlay build

# Build overlays for a specific node
wwctl overlay build n1

# Create a new site-local overlay
wwctl overlay create myoverlay

# Import a file from the host into an overlay
wwctl overlay import myoverlay /etc/motd
```

(continues on next page)

(continued from previous page)

```
# Edit a file in an overlay (opens $EDITOR)
wwctl overlay edit myoverlay /etc/motd

# Show an overlay file (optionally rendered for a specific node)
wwctl overlay show myoverlay /etc/motd
wwctl overlay show myoverlay /etc/motd.ww --render=n1

# Assign overlays to all nodes via the default profile
wwctl profile set default \
  --system-overlays="wwinit,wwclient,fstab,hostname,ssh.host_keys,NetworkManager" \
  --runtime-overlays="hosts,ssh.authorized_keys"
```

CHAPTER

NINE

FEATURE STATUS

The following table summarizes the maturity of Warewulf's top-level features. Features are classified as follows:

- **Stable** – Production-ready; the interface is well-tested and unlikely to change in incompatible ways.
- **Preview** – Functional, but may have rough edges, limited testing, or an interface that is still evolving.
- **Incomplete** – Work-in-progress; key functionality or the user-facing interface is not yet complete.

Feature	Status	Notes
iPXE	Stable	Default network bootloader used for PXE booting cluster nodes.
GRUB	Preview	Alternative bootloader; useful for UEFI and Secure Boot scenarios.
Single-stage boot	Stable	Default provisioning mode; the OS image is used directly as the root filesystem.
Two-stage boot	Preview	Uses an intermediate initrd to initialize hardware before loading the final OS image. Requires Dracut-based images.
Nodes	Stable	Core node management (add, remove, configure, list).
Profiles	Stable	Abstract node profiles for sharing configuration across groups of nodes.
Overlays	Stable	Template-based file provisioning applied to OS images at boot and at runtime.
OS images	Stable	Container-based OS image import, management, and provisioning.
Read-only images	Preview	Marking an image read-only to enable future support for image subscriptions and updates.
Kernels	Stable	Kernel extraction from OS images and per-node kernel management.
Disk provisioning	Preview	Partitioning and formatting disks on cluster nodes during provisioning.
Provision-to-disk	Preview	Provisioning an OS image to disk so that the node can subsequently boot without Warewulf.
Resources	Incomplete	Generic node resources; the user-facing interface is not yet complete.
Secure Boot	Preview	UEFI Secure Boot support via signed bootloaders.
dnsmasq	Preview	Using dnsmasq as an alternative to ISC dhcpd and TFTP for DHCP and network boot services.

CHAPTER

TEN

SERVER INSTALLATION

There are multiple methods to install a Warewulf server. This page describes some of those methods.

10.1 Binary RPMs

The Warewulf project builds binary RPMs as part of its CI/CD process. You can obtain them from the GitHub releases page.

10.1.1 Rocky Linux 9

```
# dnf install https://github.com/warewulf/warewulf/releases/download/v4.7.1/warewulf-4.7.1-1.el9.x86_
↪64.rpm
```

10.1.2 openSuse Leap

```
# zypper install https://github.com/warewulf/warewulf/releases/download/v4.7.1/warewulf-4.7.1-1.suse.
↪lp156.x86_64.rpm
```

10.2 Container images

Warewulf can be built in a Linux container. This can be especially useful for testing and development, or to replace traditional package installation. It is also possible to only use the container for building and then install it in the host system afterwards. For that look at the INSTALL, UNINSTALL and PURGE labels inside the Dockerfile

10.2.1 Docker

```
# docker build -t warewulf .
# docker run -d --replace --name warewulf-test --privileged --net=host -v /:/host -v /etc/warewulf:/etc/
↪warewulf -v /var/lib/warewulf/:/var/lib/warewulf/ -e NAME=warewulf-test -e IMAGE=warewulf␣
↪warewulf
```

10.2.2 Systemd-nspawn

Warewulf runs multiple services inside one single container and uses systemd as init system. As such, it might be better to use systemd-nspawn, which was explicitly made to run containers with a full init system.

```
# docker build -t warewulf .
# mkdir warewulf-nspawn
```

(continues on next page)

(continued from previous page)

```
# docker export "$(docker create --name warewulf-test warewulf true)" | tar -x -C warewulf-nspawn
# systemd-nspawn -D warewulf-nspawn/ passwd
# systemd-nspawn -D warewulf-nspawn/ --boot
```

10.3 Compiled from Source

Before you build the Warewulf source code you will first need to install the build dependencies:

- make: This should be available via your Linux distribution's package manager (e.g. dnf install make)
- go: Golang is also available on most current Linux distributions, but you can also install the most recent version.
- Depending on your Linux Distribution, you may need to install other development packages. Typically it is recommended to install the entire development group.

```
dnf groupinstall "Development Tools"
```

Once these dependencies are installed, you can obtain and build the source code.

10.3.1 Release Tarball

The Warewulf project releases source distributions alongside its binary RPMs. You can obtain them from the GitHub releases page.

Select the version you wish to install and download the tarball to any location on the server, then follow these directions making the appropriate substitutions:

```
curl -LO https://github.com/warewulf/warewulf/releases/download/v4.7.1/warewulf-4.7.1.tar.gz
tar -xf warewulf-4.7.1.tar.gz
cd warewulf-4.7.1
make all && sudo make install
```

10.3.2 Git

You can install different versions of Warewulf from its Git tags or branches. The main branch is where most active development occurs, so if you want to obtain the latest and greatest version of Warewulf, this is where to go. But be forewarned, using a snapshot from main is not guaranteed to be stable or generally supported for production.

If you are building for production, it is best to download a release tarball from the main site, the GitHub releases page, or from a Git tag.

```
git clone https://github.com/warewulf/warewulf.git
cd warewulf
git checkout main # or switch to a tag like 'v4.7.1'
make all && sudo make install
```

10.3.3 Runtime Dependencies

In its default configuration, Warewulf requires some operating system provided services. Generally these are provided by your distribution.

- dhcp-server
- tftp-server
- nfs-utils

If you are using an Enterprise Linux compatible distribution you can install them with dnf install dhcp-server tftp-server nfs-utils.

10.4 Building RPM packages from source

You can also build RPM packages from source.

```
dnf -y install epel-release
dnf -y install make mock
git clone git@github.com:warewulf/warewulf.git
(
  cd warewulf
  make clean && make dist warewulf.spec && mock -r rocky+epel-9-$(arch) --rebuild --spec=warewulf.
↪spec --sources=.
)
dnf -y install /var/lib/mock/rocky+epel-9-$(arch)/result/warewulf-*.$(arch).rpm
```

10.5 Starting warewulfd

The Warewulf installation registers the Warewulf service with systemd, so it should be as easy to start/stop/check as any other systemd service:

```
# systemctl enable --now warewulfd
```

CHAPTER

ELEVEN

CONTROLLING WAREWULF

Warewulf's command-line interface is based primarily around the wwctl command. This command has sub-commands for each major component of Warewulf's functionality.

- configure: configures the Warewulf server and its external services
- node: manages nodes in the cluster
- profile: defines common sets of node configuration which can be applied to multiple nodes
- image: configures (node) images
- overlay: manages overlays
- clean: removes the OCI image cache and leftover overlay images from deleted nodes

wwctl also provides additional helpers for interacting with cluster nodes over SSH and IPMI.

- power: turns nodes on and off
- ssh: provides basic parallel ssh functionality

All of these subcommands (and their respective sub-subcommands) have built-in help with either wwctl help or --help.

11.1 Hostlists

Many of the commands (e.g., wwctl node list) support a "hostlist" syntax for referring to multiple nodes at once. Hostlist expressions support both ranges and comma-separated numerical lists.

For example:

- node[1-2] expands to node1 node2
- node[1,3] expands to node1 node3
- node[1,5-6] expands to node1 node5 node6

11.2 Node status

During the whole provisioning process of your nodes, you can check their status through the following command :

```
# wwctl node status
NODENAME          STAGE             SENT                  LASTSEEN (s)
==========================================================================
n1                RUNTIME_OVERLAY   __RUNTIME__.img.gz    16
```

For each node, there are 7 different stages:

- **EFI**
- **IPXE**
- **KERNEL**
- **IMAGE**
- **INITRAMFS**
- **SYSTEM_OVERLAY**
- **RUNTIME_OVERLAY**

You can use the wwctl node status to check communication between the Warewulf server (warewulfd) and the Warewulf client (wwclient).

Note

A provisioning workflow might not use every stage.

11.3 Maintenance

wwctl clean reclaims disk space by removing two categories of data that accumulate over time:

- The OCI blob cache, stored at $cachedir/warewulf (default: /var/cache/warewulf). This cache is populated during wwctl image import and speeds up subsequent imports of images that share layers. It is safe to remove: the next wwctl image import will re-download any needed layers.
- Provisioned overlay images for nodes that have been deleted from the node database. These are stored under $wwprovisiondir/overlays and are no longer needed once the corresponding node is removed.

```
# wwctl clean
```

wwctl clean is particularly useful after deleting a large number of nodes, or when disk space is limited on the Warewulf server. Note that wwctl clean only removes the current cache location ($cachedir/warewulf); if you are upgrading from v4.5.x, the legacy cache at $datastore/oci must be removed manually (see *OCI Blob Cache*).

CHAPTER

TWELVE

SERVER CONFIGURATION

By default, the Warewulf server configuration is located at /etc/warewulf/warewulf.conf. This is a YAML-formatted configuration file used to configure the Warewulf server itself and its external services.

An initial warewulf.conf is packaged with Warewulf. Each section is covered in detail below.

Once Warewulf has been installed and configured:

- run wwctl configure --all to reconfigure external services
- run systemctl restart warewulfd to apply the configuration to the Warewulf server

Re-run both of these commands when making changes to warewulf.conf.

```
ipaddr: 192.168.1.1
netmask: 255.255.255.0
network: 192.168.1.0
warewulf:
  port: 9873
  secure: true
  update interval: 60
  autobuild overlays: true
  host overlay: true
  grubboot: false
dhcp:
  enabled: true
  template: default
  systemd name: dhcpd
tftp:
  enabled: true
  tftproot: /var/lib/tftpboot
  systemd name: tftp
  ipxe:
    00:0B: arm64-efi/snponly.efi
    "00:00": undionly.kpxe
    "00:07": ipxe-snponly-x86_64.efi
    "00:09": ipxe-snponly-x86_64.efi
nfs:
  enabled: true
  systemd name: nfsd
ssh:
  key types:
    - ed25519
    - ecdsa
```

(continues on next page)

(continued from previous page)

```
    - rsa
image mounts:
  - source: /etc/resolv.conf
    dest: /etc/resolv.conf
paths:
  bindir: /usr/bin
  sysconfdir: /etc
  localstatedir: /var/lib
  cachedir: /var/cache
  ipxesource: /usr/share/ipxe
  srvdir: /var/lib
  firewalldir: /usr/lib/firewalld/services
  systemddir: /usr/lib/systemd/system
  datadir: /usr/share
  wwoverlaydir: /var/lib/warewulf/overlays
  wwchrootdir: /var/lib/warewulf/chroots
  wwprovisiondir: /var/lib/warewulf/provision
  wwclientdir: /warewulf
```

12.1 warewulf

```
ipaddr: 192.168.1.1
netmask: 255.255.255.0
network: 192.168.1.0
warewulf:
  port: 9873
  secure: true
  update interval: 60
  autobuild overlays: true
  host overlay: true
  grubboot: false
```

- ipaddr: The Warewulf server address on the cluster network. This configuration must match the server's IP address.

  If ipaddr is specified as a CIDR address, netmask and network may be omitted.

- netmask: The netmask for the cluster network.
- network: The address of the cluster network itself.
- warewulf:port: This is the port that the Warewulf web server will be listening on. It is recommended not to change this so there is no misalignment with node's expectations of how to contact the Warewulf service.
- warewulf:secure: When true, this limits the Warewulf server to only respond to runtime overlay requests originating from a privileged port. This prevents non-root users from requesting the runtime overlay, which may contain sensitive information.

  When true, wwclient uses TCP port 987 by default. (A different port can be specified at wwclient:port.)

  Changing this option requires rebuilding node overlays and rebooting compute nodes to configure them to use a privileged port for wwclient.

- warewulf:secure files: Controls whether the /files/ route requires requests to originate from a privileged port. When unset, this inherits from warewulf:secure.

Set to false when warewulf:secure is true to allow unprivileged clients (e.g. scripts or services not running as root) to fetch files from this route without requiring a privileged source port.

- warewulf:update interval: This defines the frequency (in seconds) with which the Warewulf client on the compute node fetches overlay updates.
- warewulf:autobuild overlays: Controls whether per-node overlays will automatically be rebuilt. (e.g., when an underlying overlay is changed)

  Overlay autobuild is not 100% reliable; but it is particularly useful for building overlays for new nodes.
- warewulf:host overlay: Controls whether the special host overlay is applied to the Warewulf server during configuration. (The host overlay is used to configure external services.)
- warewulf:grubboot: Controls whether iPXE (default) or GRUB is used as the network bootloader.

12.2 dhcp

The DHCP external service can be configured explicitly with wwctl configure dhcp. This (re)writes the DHCP configuration and enables and (re)starts the DHCP service.

```
dhcp:
  enabled: true
  template: default
  systemd name: dhcpd
```

- dhcp:enabled: Whether Warewulf should configure a DHCP server on the cluster network. Set to false when managing DHCP separately.
- dhcp:template An optional DHCP template variable to control the generation of the DHCP template.

  Specifying template: static populates dhcpd.conf with static leases for each host, bypassing the DHCP range. (Run wwctl configure dhcp to update dhcpd.conf when nodes are added, removed, or changed.)
- dhcp:range start and dhcp:range end: Defines a dynamic DHCP range to use when provisioning cluster nodes. This address range must exist in the cluster network defined above. (Otherwise, the DHCP server will fail to start).

  This range should not overlap with IP addresses assigned to nodes in nodes.conf.
- dhcp:systemd name: Identifies the systemd service that manages the DHCP service. Used during wwctl configure dhcp to restart the service.

12.3 tftp

The TFTP external service can be configured explicitly with wwctl configure tftp. This writes the appropriate bootloader executables to the TFTP root directory and enables the TFTP service.

```
tftp:
  enabled: true
  tftproot: /var/lib/tftpboot
  systemd name: tftp
  ipxe:
    00:0B: arm64-efi/snponly.efi
    "00:00": undionly.kpxe
    "00:07": ipxe-snponly-x86_64.efi
    "00:09": ipxe-snponly-x86_64.efi
```

- tftp:enabled: Whether Warewulf should configure a TFTP server on the cluster network. Set to false when managing TFTP separately.
- tftp:tftproot: Identifies the local path being served by the managed TFTP server. Warewulf creates a warewulf/ subdirectory and copies iPXE and/or GRUB bootloader files to this location depending on the server configuration.
- systemd name: Identifies the systemd service that manages the TFTP service. Used during wwctl configure tftp to restart the service.
- ipxe: A map of DHCP option architecture-types to the iPXE binary that should be used for that architecture. iPXE binaries are searched for in paths:ipxesource. By default, these paths correspond to the location of the correct iPXE binary for each architecture in the distribution iPXE packages; but they can be specified explicitly when providing a local iPXE build.

12.4 nfs

The NFS external service can be configured explicitly with wwctl configure nfs. This configures the NFS server (particularly /etc/exports) on the Warewulf server and enables and starts the NFS service.

```
nfs:
  enabled: true
  export paths:
    - path: /home
      export options: rw,sync
    - path: /opt
      export options: ro,sync,no_root_squash
  systemd name: nfsd
```

- nfs:enabled: Whether Warewulf should configure an NFS server on the cluster network. Set to false when not required or when managing NFS separately.
- nfs:export paths: A list of NFS exports to configure on the Warewulf server. Each export defines a path to be exported and the export options for that export.
- systemd name: Identifies the systemd service that manages the NFS service. Used during wwctl configure nfs to restart the service.

12.5 ssh

New in Warewulf v4.5.1

SSH key types to generate during wwctl configure ssh. This creates the appropriate host keys (stored in /etc/warewulf/keys/) and authentication keys for passwordless ssh to cluster nodes. It also installs shell profiles /etc/profile.d/ssh_setup.csh and /etc/profile.d/ssh_setup.sh to initialize authentication keys for new users if and when they log into the Warewulf server.

```
ssh:
  key types:
    - ed25519
    - ecdsa
    - rsa
```

- ssh:key types: Warewulf generates host keys for each listed key type.

The first listed key type is used to generate authentication ssh keys.

12.6 image mounts

A list of paths to temporarily mount from the Warewulf server into an image during wwctl image exec and wwctl image shell, typically to allow them to operate in the host environment prior to deployment.

```
image mounts:
  - source: /etc/resolv.conf
    dest: /etc/resolv.conf
```

- image mounts:source: The path on the Warewulf server to mount into the image.
- image mounts:dest: The path in the image to use for the mount.
- image mounts:readonly: Whether the mount should be read-only (true) or allow writes into the server path (false).
- image mounts:copy: When true, copy files into the image rather than mount. This is useful for initializing files with a starting value from the Warewulf server that should then be maintained as part of the image.

12.7 paths

New in Warewulf v4.5.0

Override paths to images, overlays, and other Warewulf components.

```
paths:
  sysconfdir: /etc
  cachedir: /var/cache
  ipxesource: /usr/share/ipxe
  datadir: /usr/share
  wwoverlaydir: /var/lib/warewulf/overlays
  wwchrootdir: /var/lib/warewulf/chroots
  wwprovisiondir: /var/lib/warewulf/provision
  wwclientdir: /warewulf
```

- paths:sysconfdir: The parent directory for the warewulf configuration directory, which stores warewulf.conf and nodes.conf.
- paths:cachedir: The parent directory for the warewulf cache of OCI images during wwctl image import. The cache is stored at $cachedir/warewulf and can be cleared with wwctl clean.
- paths:ipxesource: Where to get iPXE binaries. These files are copied to warewulf.conf:tftp:tftproot by wwctl configure tftp.
- datadir: Parent directory for distribution overlays and BMC templates.
- paths:wwoverlaydir: Parent directory for site overlays.
- paths:wwchrootdir: Parent directory for Warewulf images.
- paths:wwprovisiondir: The destination for built images and overlay images.
- paths:wwclientdir: Where wwclient looks for its configuration on a provisioned node.

12.8 wwclient

Configuration for the wwclient service on cluster nodes.

```
wwclient:
  port: 987
```

- wwclient:port: The source port used by wwclient. By default an ephemeral port is selected; but warewulf.conf:warewulf:secure: true requires a known privileged port.

  wwclient will use the TCP port "987" by default if secure: true; but, if that port is otherwise in use, a different port may be specified.

12.9 api

New in Warewulf v4.6.1

Configuration for the REST API of the warewulfd service.

```
api:
  enabled: true
  tls: false
  allowed subnets:
    - 127.0.0.0/8
    - ::1/128
```

- api:enabled: Whether the warewulfd service should provide access via a REST interface. Defaults to false.
- api:tls: Whether the REST API requires HTTPS. Defaults to false. Also requires warewulf:tls.
- api:allowed subnets: Which subnets are allowed to access the REST API. By default, only localhost has access. Does not apply to /api/docs.

See *REST API* for authentication and the available endpoints.

12.10 hostfile

There are no explicit "hostfile" configuration options in warewulf.conf; but wwctl configure hostfile updates the Warewulf server's /etc/hosts file to include expected configuration for the server itself as well as the known names of the cluster nodes and their interfaces.

Entries from the Warewulf server's /etc/hosts file are distributed to cluster nodes by the "hosts" overlay.

12.11 logging

You can control the logging verbosity using the WAREWULFD_LOGLEVEL environment variable.

The best place to set this is in /etc/default/warewulfd, which is sourced by the warewulfd.service systemd unit via the EnvironmentFile directive.

By default, the log level is wwlog.INFO (20), and the messages are logged at either INFO or SERV (25).

Setting the log level to WARN (30) should suppress INFO and SERV while still surfacing more important issues.

To do that, execute the following commands:

```
echo WAREWULFD_LOGLEVEL=30 >> /etc/default/warewulfd
systemctl restart warewulfd.service
```

CHAPTER

THIRTEEN

SERVER ROUTES

The Warewulf provisioning daemon, warewulfd, serves all boot and provisioning resources over HTTP. Each resource type has a dedicated route, and nodes are identified by their Warewulf ID (wwid).

{wwid} is typically the node's default MAC address: a colon-separated hexadecimal string, e.g., aa:bb:cc:dd:ee:ff. Dashes are accepted in place of colons and are normalized automatically.

Note

The port warewulfd listens on is configured with warewulf:port in warewulf.conf (default: 9873). When TLS is enabled, a second listener is started on the port configured with warewulf:tls port (default: 9874).

13.1 URL Patterns

Every provisioning route (except /status) supports six equivalent URL patterns for specifying the node identity:

```
/{stage}/{wwid}           # wwid in path
/{stage}?wwid={wwid}      # wwid as query parameter
/{stage}                  # wwid resolved from ARP cache

/provision/{wwid}?stage={stage}
/provision?wwid={wwid}&stage={stage}
/provision?stage={stage}  # wwid resolved from ARP cache
```

The /efiboot/ route is an exception: the path segment contains the boot file name rather than a wwid, and the node is identified via ?wwid= or ARP:

```
/efiboot/{file}                  # file in path, wwid from ARP
/efiboot/{file}?wwid={wwid}      # file in path, explicit wwid
/efiboot?wwid={wwid}&file={file}  # both as query parameters
```

13.2 Common Query Parameters

Most provisioning routes accept the following query parameters:

- wwid: Warewulf ID of the node, typically its default MAC address. Used when the node identity is not embedded in the URL path. Takes priority over ARP lookup; the path always takes priority over the wwid query parameter.
- assetkey: Hardware asset tag. If the node has an AssetKey configured in nodes.conf, the server requires this parameter to match before serving any content. See *Security* below.

- uuid: System UUID of the requesting node. Accepted for logging purposes.
- compress: Compression format for the response. The only supported value is gz. When compress=gz is specified, the server serves a pre-built gzip-compressed version of the file. If no compressed version exists, the server returns 404 Not Found.

13.3 Provisioning Routes

13.3.1 /ipxe/{wwid}

Serves a rendered iPXE boot script for the node identified by {wwid}.

The script is rendered as a Go template from a file in /etc/warewulf/ipxe/. The specific template used is determined by the node's Ipxe field (defaulting to default); for example, a node with Ipxe: dracut receives the template from /etc/warewulf/ipxe/dracut.ipxe.

If the requesting node is not known to Warewulf, the server falls back to serving /etc/warewulf/ipxe/unconfigured.ipxe.

Query parameters: assetkey, uuid

13.3.2 /kernel/{wwid}

Serves the raw kernel binary for the node identified by {wwid}. The kernel is taken from the node's assigned image.

Query parameters: assetkey, uuid, compress

13.3.3 /image/{wwid}

Serves the raw OS image file for the node identified by {wwid}.

Query parameters: assetkey, uuid, compress

13.3.4 /initramfs/{wwid}

Serves the initramfs binary for the node identified by {wwid}. The initramfs is extracted from the node's assigned image based on the node's kernel version. This route is used in two-stage boot configurations. See *Two-stage boot: dracut* for details.

Query parameters: assetkey, uuid, compress

13.3.5 /system/{wwid}

Serves the system overlay image for the node identified by {wwid}. The system overlay is rendered at provisioning time and contains configuration files that are static for the lifetime of the boot.

When autobuild overlays is enabled in warewulf.conf, the server will automatically rebuild the overlay if it is out of date relative to nodes.conf or the overlay source files.

Query parameters: assetkey, uuid, compress

13.3.6 /runtime/{wwid}

Serves the runtime overlay image for the node identified by {wwid}. The runtime overlay is rendered on demand and may contain node-specific secrets. wwclient fetches the runtime overlay periodically during normal operation.

When warewulf:secure is enabled in warewulf.conf, this route requires that the request originate from a privileged TCP port (port number less than 1024). This prevents unprivileged users on a node from retrieving the runtime overlay.

When TLS is enabled in warewulf.conf, this route requires that the request arrive over HTTPS. Plain-HTTP requests are rejected with 403 Forbidden. The HTTPS listener port is configured with warewulf:tls port.

Query parameters: assetkey, uuid, compress

13.3.7 /efiboot/{file}

Serves EFI boot files for GRUB-based booting. The requesting node is identified by ?wwid= (preferred) or, if not supplied, by an ARP lookup of the client's IP address against the kernel's ARP cache (/proc/net/arp). This route is intended for EFI HTTP Boot clients, where the firmware fetches a boot URI from DHCP and cannot perform variable substitution.

Note

On large clusters the kernel's default ARP cache limits may be exceeded, causing node identification to fail. See *ARP Cache Overflow on Large Clusters* for tuning guidance.

The {file} component determines what is served:

- shim.efi: Serves the shim.efi binary extracted from the node's assigned image.
- grub.efi (or grubx64.efi, grubaa64.efi, grubia32.efi, grubarm.efi, grub-tpm.efi): Serves the GRUB EFI binary extracted from the node's assigned image.
- grub.cfg: Serves a rendered GRUB configuration file from /etc/warewulf/grub/grub.cfg.ww. The configuration is rendered as a Go template for the identified node.

Because shim.efi resolves subsequent files relative to its own load URL, GRUB and grub.cfg are also fetched from the /efiboot/ path. The grub.cfg served by this route uses ${net_default_mac} to embed the node's wwid in all further provisioning URLs, directing subsequent requests to the per-node /grub/{wwid} route.

Query parameters: assetkey, uuid, wwid, file

- file: The EFI file to serve (shim.efi, grub.efi, or grub.cfg). Used when the file name is not embedded in the URL path.

Note

/efiboot/ is the recommended route for EFI HTTP Boot clients. For TFTP-booted GRUB clients that know their own wwid, use /grub/{wwid} to fetch the per-node GRUB configuration directly.

13.3.8 /grub/{wwid}

Serves a rendered GRUB configuration file for the node identified by {wwid}. The configuration is rendered from /etc/warewulf/grub/grub.cfg.ww as a Go template. This route is the preferred method for TFTP-booted GRUB clients to fetch their per-node configuration, as the node identity is explicit in the URL rather than resolved via ARP.

Query parameters: assetkey, uuid, wwid

13.3.9 /provision/{wwid}

A legacy dispatcher route. The provisioning stage is determined by the stage query parameter, which is dispatched to the appropriate handler:

- stage=ipxe → /ipxe/
- stage=kernel → /kernel/

- stage=image → /image/
- stage=initramfs → /initramfs/
- stage=system → /system/
- stage=runtime → /runtime/
- stage=grub → /grub/

Query parameters: stage (required), assetkey, uuid, compress

13.4 Status Route

13.4.1 /status

Returns a JSON object containing the last-known provisioning status for all nodes that have contacted the server. No authentication is required.

```
{
  "nodes": {
    "node01": {
      "node name": "node01",
      "stage": "RUNTIME_OVERLAY",
      "sent": "2 kB",
      "ipaddr": "10.0.1.1",
      "last seen": 1712345678
    }
  }
}
```

The stage field reflects the most recent provisioning stage completed for the node. Possible values include IPXE, KERNEL, IMAGE, INITRAMFS, SYSTEM_OVERLAY, RUNTIME_OVERLAY, and EFI.

13.5 REST API

When enabled in warewulf.conf, warewulfd exposes a REST API under /api/. The API provides programmatic access to nodes, profiles, images, and overlays. Interactive documentation is available at /api/docs.

When TLS is enabled, access to the REST API can additionally be restricted to HTTPS-only requests by setting api: tls: true in warewulf.conf.

See *REST API* for full details.

13.6 Files Route

13.6.1 /files/{path}

Serves static files from the warewulf files directory (wwfilesdir in warewulf.conf, defaulting to /var/lib/warewulf/files). Subdirectories are supported.

Every request must identify a node, either via the ?wwid= query parameter (a hardware address) or by ARP cache lookup of the requesting IP. If no node can be identified, the server returns 401 Unauthorized.

When secure is enabled in warewulf.conf, requests must originate from a privileged port (< 1024); otherwise 403 Forbidden is returned. If the node has an AssetKey configured, the ?assetkey= parameter must be present and match; a missing key returns 401 Unauthorized and an incorrect key returns 403 Forbidden.

```
# Place files in the warewulf files directory:
$ cp myfile.txt /var/lib/warewulf/files/
$ mkdir -p /var/lib/warewulf/files/scripts
$ cp setup.sh /var/lib/warewulf/files/scripts/

# Fetch from a compute node:
$ curl http://<server>:9873/files/myfile.txt?wwid=00:00:00:00:00:01
$ curl http://<server>:9873/files/scripts/setup.sh?wwid=00:00:00:00:00:01
```

Directory listing is disabled; requests for a directory path return 404 Not Found.

Template rendering: Adding the ?render query parameter renders the file as a Go template for the identified node, with the same template functions and data available as in overlay templates. Without ?render, files are returned as raw bytes. The path must refer to a .ww template file. If the path does not end in .ww but a .ww-suffixed version exists, that file is used automatically. Using ?render on a path where no .ww file can be found returns 400 Bad Request if the exact file exists, or 404 Not Found if neither form exists.

```
# Place a template in the files directory:
$ echo 'hostname={{ .Hostname }}' > /var/lib/warewulf/files/info.ww

# Fetch the raw template:
$ curl http://<server>:9873/files/info.ww?wwid=00:00:00:00:00:01

# Fetch the rendered template (explicit .ww suffix):
$ curl 'http://<server>:9873/files/info.ww?render&wwid=00:00:00:00:00:01'

# Fetch the rendered template (implicit .ww suffix):
$ curl 'http://<server>:9873/files/info?render&wwid=00:00:00:00:00:01'
```

Query parameters: wwid, assetkey, render

13.7 Security

Several mechanisms are available to restrict access to provisioning routes.

13.7.1 Asset key validation

If a node is configured with an AssetKey in nodes.conf, the Warewulf server will only respond to provisioning requests that include a matching ?assetkey= query parameter. Requests with a missing or incorrect asset key receive 401 Unauthorized. The asset key is typically a hardware-level firmware string (an "asset tag") that is accessible only with root or physical access.

```
# wwctl node set node01 --assetkey "SYSTEM-ASSET-TAG"
```

13.7.2 Secure mode

When warewulf:secure is set to true in warewulf.conf, the /runtime/ route requires that requests originate from a privileged TCP source port (port number less than 1024). Because only processes running as root can bind to privileged ports, this prevents unprivileged users on a cluster node from downloading the runtime overlay.

13.7.3 TLS

When TLS is enabled in warewulf.conf, the /runtime/ route rejects plain-HTTP requests with 403 Forbidden. Runtime overlays must be fetched over HTTPS. Because iPXE and GRUB cannot handle HTTPS, the kernel, image, and system overlay continue to be served over plain HTTP even when TLS is enabled.

See Security for instructions on enabling TLS and generating certificates.

CHAPTER

FOURTEEN

NETWORKING

14.1 Multiple networks

It is possible to configure several networks not just for the nodes but also for the management of dhcpd and tftp. There are two ways to achieve this:

- Add the networks to the templates of dhcpd and/or the dnsmasq template directly.
- Add the networks to a dummy node and change the templates of dhcp and dnsmasq accordingly.

The first method is relatively trivial. The second method is described below.

As the first step, add the dummy node.

```
wwctl node add deliverynet
```

Add the delivery networks to this node.

```
wwctl node set \
  --ipaddr 10.0.20.250 \
  --netmask 255.255.255.0 \
  --netname deliver1 \
  --nettagadd network=10.0.20.0,dynstart=10.10.20.10,dynend=10.10.20.50 \
  deliverynet

wwctl node set \
  --ipaddr 10.0.30.250 \
  --netmask 255.255.255.0 \
  --netname deliver2 \
  --nettagadd network=10.0.30.0,dynstart=10.10.30.10,dynend=10.10.30.50 \
  deliverynet
```

The IP address is used as the network address of the host in the delivery network and an additional tag is used for the definition of the network itself and the dynamic dhcp range. You can check the result with wwctl node list.

```
# wwctl node list -a deliverynet
NODE         FIELD                    PROFILE  VALUE
deliverynet  Id                       --       deliverynet
deliverynet  Comment                  default  This profile is automatically included for each node
deliverynet  ImageName                default  leap15.5
deliverynet  Ipxe                     --       (default)
deliverynet  RuntimeOverlay           --       (hosts,ssh.authorized_keys)
deliverynet  SystemOverlay            --       (wwinit,wwclient,hostname,ssh.host_keys,systemd.
```

(continues on next page)

(continued from previous page)

```
↪netname,NetworkManager)
deliverynet  Root                          --       (initramfs)
deliverynet  Init                          --       (/sbin/init)
deliverynet  Kernel.Args                   --       (quiet crashkernel=no net.ifnames=1)
deliverynet  Profiles                      --       default
deliverynet  PrimaryNetDev                 --       (deliver1)
deliverynet  NetDevs[deliver2].Type        --       (ethernet)
deliverynet  NetDevs[deliver2].OnBoot      --       (true)
deliverynet  NetDevs[deliver2].Ipaddr      --       10.0.30.250
deliverynet  NetDevs[deliver2].Netmask     --       255.255.255.0
deliverynet  NetDevs[deliver2].Tags[dynend]    --       10.10.30.50
deliverynet  NetDevs[deliver2].Tags[dynstart]  --       10.10.30.10
deliverynet  NetDevs[deliver2].Tags[network]   --       10.0.30.0
deliverynet  NetDevs[deliver1].Type        --       (ethernet)
deliverynet  NetDevs[deliver1].OnBoot      --       (true)
deliverynet  NetDevs[deliver1].Ipaddr      --       10.0.20.250
deliverynet  NetDevs[deliver1].Netmask     --       255.255.255.0
deliverynet  NetDevs[deliver1].Primary     --       (true)
deliverynet  NetDevs[deliver1].Tags[network]   --       10.0.20.0
deliverynet  NetDevs[deliver1].Tags[dynend]    --       10.10.20.50
deliverynet  NetDevs[deliver1].Tags[dynstart]  --       10.10.20.10
```

Now the templates of dhcpd and/or dnsmasq must be modified.

```
wwctl overlay edit host etc/dhcpd.conf.ww
wwctl overlay edit host etc/dnsmasq.d/ww4-hosts.ww
```

For the dhcp template you should add following lines

```
{{/* multiple networks */}}
{{- range $node := $.AllNodes}}
{{- if eq $node.Id.Get "deliverynet" }}
{{- range $netname, $netdev := $node.NetDevs}}
# network {{ $netname }}
subnet {{$netdev.Tags.network.Get}} netmask {{$netdev.Netmask.Get}} {
    max-lease-time 120;
    range {{$netdev.Tags.dynstart.Get}} {{$netdev.Tags.dynend.Get}};
    next-server {{$netdev.Ipaddr.Get}};
}
{{- end }}
{{- end }}
{{- end }}
```

and for the dnsmasq the following lines should be added

```
{{/* multiple networks */}}
{{- range $node := $.AllNodes}}
{{- if eq $node.Id.Get "deliverynet" }}
{{- range $netname, $netdev := $node.NetDevs}}
# network {{ $netname }}
dhcp-range={{$netdev.Tags.dynstart.Get}},{{$netdev.Tags.dynend.Get}},{{$netdev.Netmask.Get}},6h
{{- end }}
{{- end }}
```

(continues on next page)

(continued from previous page)

```
{{- end }}
```

Note that the {{- if eq $node.Id.Get "deliverynet" }} is used to identify the dummy host which carries the network information.

CHAPTER

FIFTEEN

USING DNSMASQ

As an experimental feature, it is possible to use dnsmasq instead of the ISC dhcpd server and TFTP server.

In order to keep the configuration manageable, the file /etc/dnsmasq.d/ww4-hosts.conf is created and must be included in the main dnsmasq.conf via the conf-dir=/etc/dnsmasq.d option.

15.1 Installation

Before the installation, make sure that dhcpd and tftp are disabled. You can do that with the commands:

```
systemctl disable --now dhcpd
systemctl disable --now tftp
```

Now you can install dnsmasq.

```
# Rocky Linux
dnf install dnsmasq

# SUSE
zypper install dnsmasq
```

After the installation, instruct warewulf to use dnsmasq as its dhcpd and tftp service. This is done in the server configuration file, typically at /etc/warewulf/warewulf.conf:

```
tftp:
  systemd name: dnsmasq
dhcp:
  systemd name: dnsmasq
```

The configuration of dnsmasq often doesn't need to be changed, as the default configuration includes all files with following pattern /etc/dnsmasq.d/*conf into its configuration. This configuration is created by the overlay template host:/etc/dnsmasq.d/ww4-hosts.conf.ww.

Note

In certain distributions, such as Rocky Linux 9, dnsmasq is configured to listen locally via the interface=lo option by default. Replace this entry in /etc/dnsmasq.conf with the interface associated with your Warewulf network, or remove/comment out the interface option entirely to enable listening on all interfaces.

Once the Warewulf configuration has been updated, re-deploy the configuration and restart warewulfd.

```
wwctl configure --all
systemctl restart warewulfd.service
```

CHAPTER

SIXTEEN

SECURITY

While certain parallelization and high performance library capabilities still require lowering the security threshold within a cluster, Warewulf strives to support good security practices within the cluster wherever possible.

16.1 Provisioning Security

Provisioning is, by default, a relatively "insecure" process: there is generally nothing preventing a user on a cluster node from spoofing a provision request and downloading the OS image and overlays for inspection. If any of these include secrets (e.g., private keys) they are at risk of exposure.

There are multiple ways to secure the Warewulf provisioning process:

- The best way to secure the provisioning process is to dedicate a vLAN specifically for provisioning, and then not make that vLAN available in the provisioned environment. Warewulf can be used in such an environment (without `wwclient`) but you must consult your switch documentation and features to implement a default vLAN for provisioning and to ensure that the runtime operating system is configured for a different tagged vLAN once booted.
- Warewulf can leverage hardware "asset tags" which almost all vendors support. This is a configurable firmware string that is accessible only via root or physical access. During provisioning (as well as post provisioning via `wwclient`) Warewulf sends the detected asset tag to the Warewulf server as a "shared secret" token. If the node is also configured with an `asset key` on the Warewulf server (e.g., via `wwctl node set --assetkey "..."`), the Warewulf server will only respond to requests with a matching asset tag.
- If the Warewulf server is configured with `warewulf:secure: true`, then it will only provide the runtime overlay to a `wwclient` communicating from a privileged (< 1024) TCP port. This prevents unprivileged cluster users from being able to retrieve the runtime overlay.
- When the nodes are booted via `shim` and `grub` Secure Boot can be enabled. This means that the nodes only boot the kernel which is provided by the distributor and also custom complied modules can't be loaded.
- TLS (transport layer security) can be enabled for the Warewulf server. When enabled, HTTPS is used when transferring runtime overlays. The kernel and system image are *always* transferred unencrypted.

16.2 SELinux

The Warewulf server can be run with SELinux enabled in "targeted" and "enforcing" mode.

For more information about running SELinux-enabled cluster OS images, see *SELinux-Enabled Images*.

16.3 firewalld

If the Warewulf server is running firewalld, the following services must be added for them to function:

```
firewall-cmd --permanent --add-service=warewulf
firewall-cmd --permanent --add-service=dhcp
firewall-cmd --permanent --add-service=nfs
firewall-cmd --permanent --add-service=tftp
firewall-cmd --reload
```

Note

The DHCP, TFTP, and NFS services may be managed manually, apart from the Warewulf server. In that case, they may be omitted from the firewalld configuration on the Warewulf server; but they must be accessible from where they are served.

16.4 nftables

If the Warewulf server is running nftables directly, without firewalld, ensure that TCP port 9873 must be permitted for cluster nodes to communicate with the Warewulf server.

```
nft add rule inet filter input tcp dport 9873 accept
nft list ruleset >/etc/nftables.conf
systemctl restart nftables
```

16.5 TLS / HTTPS

TLS can be enabled by setting tls: true in the Warewulf server configuration.

```
warewulf:
  tls: true
  tls port: 9874
```

This enables an HTTPS server on the TLS port (default: 9874). A key and self-signed certificate can be created with wwctl configure tls.

By default, the key and certificate are stored in /etc/warewulf/tls/.

You can also import your own keys with wwctl configure tls --import.

If HTTPS is enabled the delivery of the runtime overlay is disabled over HTTP, and the runtime overlay is only retrieved by wwclient.

To additionally require TLS for access to the REST API, set tls: true under the api: section:

```
api:
  enabled: true
  tls: true
```

When api: tls is set, the REST API rejects plain-HTTP requests.

CHAPTER

SEVENTEEN

BOOTLOADERS

Warewulf uses iPXE as its default network bootloader. As a tech preview, support for GRUB is also available, which adds support for secure boot.

Also as a tech preview, Warewulf may also use iPXE or GRUB to boot a dracut initramfs as an initial stage before loading the image. This is called a two-stage boot.

17.1 Booting with iPXE

The /etc/warewulf/ipxe/ directory contains *text/templates* that are used by the Warewulf configuration process to configure the ipxe service.

Starting in v4.5.0, Warewulf no longer includes an iPXE binary. Instead, by default Warewulf uses the iPXE that comes with the host OS.

Unfortunately, we've encountered a few instances where bugs in the OS-provided iPXE that sometimes make booting a full OS image as an "initrd" unreliable.

Building iPXE locally, using a more recent "version" of the iPXE source code, can alleviate some of these issues.

Another alternative is *Two-stage boot: dracut*, which uses the Linux kernel to load the full OS image, avoiding the issue entirely.

17.1.1 Building iPXE locally

By default (as of v4.5.0) Warewulf packages use iPXE from the host operating system rather than bundling iPXE binaries with Warewulf. However, sometimes the specific build included in the host OS has bugs or missing features, and a local build of iPXE is necessary.

The Warewulf project provides a build-ipxe.sh script to simplify the process of building iPXE locally.

```
# curl -LO https://raw.githubusercontent.com/warewulf/warewulf/main/scripts/build-ipxe.sh
# bash build-ipxe.sh -h
Usage: build-ipxe.sh
       [-h] (help)
TARGETS: bin-x86_64-pcbios/undionly.kpxe bin-x86_64-efi/snponly.efi bin-arm64-efi/snponly.efi
IPXE_BRANCH: master
DESTDIR: /usr/local/share/ipxe
```

Running build-ipxe.sh

The script, by default, builds iPXE for x86_64 BIOS, x86_64 EFI, and arm64 EFI from the master branch on the iPXE project GitHub and stores the resultant builds in /usr/local/share/ipxe/. (These parameters can be adjusted by setting TARGETS, IPXE_BRANCH, and DESTDIR environment variables, with the current values shown in the -h output for reference.)

```
# mkdir -p /usr/local/share/ipxe
# bash build-ipxe.sh
[...]
# ls -1 /usr/local/share/ipxe/
bin-arm64-efi-snponly.efi
bin-x86_64-efi-snponly.efi
bin-x86_64-pcbios-undionly.kpxe
```

Note

Building for aarch64 requires the package gcc-aarch64-linux-gnu.

Build options

By default, build-ipxe.sh enables support for ZLIB and GZIP images, as well as commands for managing VLANs and the framebuffer console. The x86_64 build also enables support for the serial console.

Additional build options can be configured by editing the build-ipxe.sh script. For example, the x86_64 build is configured in the configure_x86_64 function.

```
function configure_x86_64 {
  sed -i.bak \
      -e 's,//\(#define.*CONSOLE_SERIAL.*\),\1,' \
      -e 's,//\(#define.*CONSOLE_FRAMEBUFFER.*\),\1,' \
      config/console.h
  sed -i.bak \
      -e 's,//\(#define.*IMAGE_ZLIB.*\),\1,' \
      -e 's,//\(#define.*IMAGE_GZIP.*\),\1,' \
      -e 's,//\(#define.*VLAN_CMD.*\),\1,' \
      config/general.h
}
```

For example, the imgextract command can be explicitly enabled.

```
function configure_x86_64 {
  sed -i.bak \
      -e 's,//\(#define.*CONSOLE_SERIAL.*\),\1,' \
      -e 's,//\(#define.*CONSOLE_FRAMEBUFFER.*\),\1,' \
      config/console.h
  sed -i.bak \
      -e 's,//\(#define.*IMAGE_ZLIB.*\),\1,' \
      -e 's,//\(#define.*IMAGE_GZIP.*\),\1,' \
      -e 's,//\(#define.*VLAN_CMD.*\),\1,' \
      -e 's,//\(#define.*IMAGE_ARCHIVE_CMD.*\),\1,' \
      config/general.h
}
```

Note

IMG_ARCHIVE_CMD is already enabled by default in the iPXE master branch, but only takes effect when at least one archive image format is configured. This is the case in the default state of build-ipxe.sh, which enables support for ZLIB and GZIP archive image formats.

Configuring Warewulf (>= v4.5.0)

In Warewulf v4.5.0, Warewulf can be configured to use these files using the tftp.ipxe and paths.ipxesource configuration parameters in warewulf.conf.

```
# warewulf.conf
tftp:
  ipxe:
    "00:00": bin-x86_64-pcbios-undionly.kpxe
    "00:07": bin-x86_64-efi-snponly.efi
    "00:09": bin-x86_64-efi-snponly.efi
    "00:0B": bin-arm64-efi-snponly.efi
paths:
  ipxesource: /usr/local/share/ipxe
```

Restart warewulfd following the change to warewulf.conf. Then remove any previously-provisioned files from /var/lib/tftpboot/warewulf/ and use wwctl configure tftp and wwctl configure dhcp to re-provision the TFTP files and update the DHCP configuration.

```
# sudo systemctl restart warewulfd
# rm /var/lib/tftpboot/warewulf/*
# wwctl configure tftp
Writing PXE files to: /var/lib/tftpboot/warewulf
Enabling and restarting the TFTP services
# wwctl configure dhcp
Building overlay for wwctl1: host
Enabling and restarting the DHCP services
```

Configuring Warewulf (< v4.5.0)

Prior to v4.5.0, Warewulf packages included bundled builds of iPXE and did not provide a mechanism for configuring which iPXE to use. To use a custom iPXE before v4.5.0, replace the bundled builds included with Warewulf. After that, remove any previously-provisioned files from /var/lib/tftpboot/warewulf/ and use wwctl configure tftp to re-provision the TFTP files.

```
# cp /usr/local/share/ipxe/bin-arm64-efi-snponly.efi /usr/share/warewulf/ipxe/arm64.efi
# cp /usr/local/share/ipxe/bin-x86_64-efi-snponly.efi /usr/share/warewulf/ipxe/x86_64.efi
# cp /usr/local/share/ipxe/bin-x86_64-pcbios-undionly.kpxe /usr/share/warewulf/ipxe/x86_64.kpxe
# rm /var/lib/tftpboot/warewulf/*
# wwctl configure tftp
Writing PXE files to: /var/lib/tftpboot/warewulf
Enabling and restarting the TFTP services
```

17.2 Booting with GRUB

Support for GRUB as a network bootloader (replacing iPXE) is available in Warewulf as a technology preview.

Instead of the iPXE starter a combination of shim and GRUB can be used with the advantage that secure boot can be used. That means that only the signed kernel of a distribution can be booted. This can be a huge security benefit for some scenarios.

In order to enable the grub boot method it has to be enabled in warewulf.conf.

```
warewulf:
  grubboot: true
```

Nodes which are not known to Warewulf are booted with the shim/grub from the Warewulf server host.

17.2.1 Secure boot

If secure boot is enabled at every step a signature is checked and the boot process fails if this check fails. The shim typically only includes the key for a single operating system, which means that each distribution needs separate shim and grub executables. Warewulf extracts these binaries from the images. If the node is unknown to Warewulf or can't be identified during the TFTP boot phase, the shim/grub binaries of the host in which Warewulf is running are used.

17.2.2 Install shim and efi

shim.efi and grub.efi must be installed in the image for it to be booted by GRUB.

```
# wwctl image shell leap15.5
[leap15.5] Warewulf> zypper install grub2 shim

# wwctl image shell rocky9
[rocky9] Warewulf> dnf install shim-x64.x86_64 grub2-efi-x64.x86_64
```

These packages must also be installed on the Warewulf server host to enable node discovery using GRUB.

17.2.3 HTTP boot

Modern EFI systems have the possibility to directly boot per http. HTTP boot is part of the GRUB boot chain described above, and is not a separate bootloader. The node firmware fetches shim.efi over HTTP instead of TFTP, and the rest of the chain proceeds as it does for a TFTP-booted GRUB node. Warewulf does not serve an iPXE binary over HTTP, so HTTP boot cannot be used to start the iPXE boot method.

The flow diagram is the following:

Warewulf delivers the initial shim.efi and grub.efi via http as taken directly from the node's assigned image.

Note

HTTP boot has the following limitations in the current implementation.

- The DHCP rule that hands out the HTTP boot URI is generated differently by each DHCP backend. With ISC dhcpd (the default) the rule is written only when grubboot is enabled, and it matches any HTTPClient vendor class, so the rule matches clients of any architecture. With dnsmasq the rule is always written, but it matches only the vendor class HTTPClient:Arch:00016, which is the IANA processor architecture type for "x64 uefi boot from http". An aarch64 client booting over HTTP presents architecture type 19, "arm uefi 64 boot from http", and is therefore not matched.
- The HTTP boot URI is built from the server's IPv4 address, so HTTP boot requires an IPv4 address on the provisioning interface and is not usable on an IPv6-only Warewulf server.
- The /efiboot/ route identifies the requesting node from the wwid query parameter or, when that is absent, from a lookup of the client's address in the IPv4 ARP cache. See *Server Routes* for details.

17.3 Two-stage boot: dracut

Some systems, typically due to limitations in their BIOS or EFI firmware, are unable to load an image of a certain size directly with a traditional bootloader, either iPXE or GRUB. As a workaround for such systems, Warewulf can be configured to load a dracut initramfs from the image and to use that initramfs to load the full image.

The two-stage boot is independent of how the first-stage bootloader reaches the node. It runs over the normal network boot path, whether iPXE or GRUB was delivered over TFTP or over HTTP, and it does not require *HTTP boot* to be configured.

Warewulf provides a dracut module to configure the dracut initramfs to load the image. This module is available in the warewulf-dracut subpackage, which must be installed in the image.

With the warewulf-dracut package installed in the image, you can then build an initramfs inside the image.

```
# Enterprise Linux
wwctl image exec rockylinux-9 --build=false -- /usr/bin/dnf -y install https://github.com/warewulf/
↪warewulf/releases/download/v4.7.1/warewulf-dracut-4.7.1-1.el9.noarch.rpm
wwctl image exec rockylinux-9 -- /usr/bin/dracut --force --no-hostonly --add wwinit --regenerate-all

# SUSE
wwctl image exec leap-15 --build=false -- /usr/bin/zypper -y install https://github.com/warewulf/
↪warewulf/releases/download/v4.7.1/warewulf-dracut-4.7.1-1.suse.lp156.noarch.rpm
wwctl image exec leap-15 -- /usr/bin/dracut --force --no-hostonly --add wwinit --regenerate-all
```

Note

In some systems, such as rockylinux:8, it may be necessary to remove /etc/machine-id for dracut to properly generate the initramfs in the location that Warewulf is expecting.

To direct iPXE to fetch the node's initramfs image and boot with dracut semantics, set an IPXEMenuEntry tag for the node.

Note

Warewulf configures iPXE with a template located at /etc/warewulf/ipxe/default.ipxe. Inspect the template to learn more about the dracut booting process.

```
wwctl node set wwnode1 --tagadd IPXEMenuEntry=dracut
```

Note

The IPXEMenuEntry variable may be set at the node or profile level.

Alternatively, to direct GRUB to fetch the node's initramfs image and boot with dracut semantics, set a GrubMenuEntry tag for the node.

Note

Warewulf configures GRUB with a template located at /etc/warewulf/grub/grub.cfg.ww. Inspect the template to learn more about the dracut booting process.

```
wwctl node set wwnode1 --tagadd GrubMenuEntry=dracut
```

Note

The GrubMenuEntry variable may be set at the node or profile level.

During boot, warewulfd will detect and dynamically serve an initramfs from a node's image in much the same way that it can serve a kernel from an image. This image is loaded by iPXE (or GRUB) which directs dracut to fetch the node's image during boot.

The wwinit module provisions to tmpfs. By default, tmpfs is permitted to use up to 50% of physical memory. This size limit may be adjusted using the kernel argument wwinit.tmpfs.size. (This parameter is passed to the size option during tmpfs mount. See tmpfs(5) for more details.)

17.3.1 Network configuration in the dracut initramfs

The :dracut entry passes rd.neednet=1 and an ip= argument for each of the node's network devices, so that the initramfs can fetch the image from the Warewulf server. The autoconfiguration method is chosen per device from the node's configuration:

- a device with an ipaddr6 and no ipaddr gets ip=<device>:auto6, so that the initramfs configures IPv6 (SLAAC, and DHCPv6 if the router advertisement asks for it) and does not wait for a DHCPv4 lease;
- every other device, including dual-stack devices, gets ip=<device>:dhcp.

This matters because on an IPv6-only fabric a request for DHCPv4 is not merely redundant. NetworkManager's initrd generator translates ip=<device>:dhcp into ipv4.method=auto with ipv4.may-fail=no, which makes a DHCPv4 lease a requirement for the connection to activate. Where no DHCPv4 server exists the connection never activates, nothing that waits on the network proceeds, and the boot ends in the dracut emergency shell with FATAL: Unable to load stage: system.

The value a node will be served can be read without booting it.

```
curl -s "http://<server>:<port>/ipxe/<hwaddr>" | grep 'set dracut_net'
```

That prints one line for the :dracut entry and one for :dracut_static. The latter builds its ip= argument from ipaddr, gateway and netmask, so it does not support a node that has only an ipaddr6.

Note

When adjusting the method by hand in a copy of the template, prefer auto6 over dracut's either6. NetworkManager's initrd generator does not recognise either6: it falls back to interpreting the argument positionally, fails to read the device name as an address, and discards the whole ip= argument.

CHAPTER

EIGHTEEN

UPGRADING WAREWULF

New versions of Warewulf might introduce changes to warewulf.conf and nodes.conf. The wwctl upgrade command can help ease the transition between versions.

Note

wwctl upgrade will back up any files before it changes them (to <name>-old) but it is good practice to back up your configuration manually.

```
# wwctl upgrade config
# wwctl upgrade nodes --add-defaults --replace-overlays
```

Both upgrade commands support specifying --output-path=- to print the upgraded configuration file to standard out for inspection before replacing the configuration files.

CHAPTER

NINETEEN

REST API

19.1 Interactive Documentation

The API is disabled by default. Set api:enabled: true in warewulf.conf and restart warewulfd.

Interactive Swagger UI documentation is then served at /api/docs, for example http://localhost:9873/api/docs, with its OpenAPI schema at /api/docs/openapi.json. Substitute the address and warewulf:port of the Warewulf server, or https and warewulf:tls port (default 9874) if api:tls is enabled.

The endpoints themselves require a user in /etc/warewulf/auth.conf. See *Authentication* below.

19.2 Authentication

Authentication is managed at /etc/warewulf/auth.conf. This is a YAML formatted file with a single key: users:, that is a list of user names and passwords able to authenticate to the API.

Warning

Because warewulfd runs as root by default, and because warewulfd can run effectively arbitrary code via overlay templates, API access is tantamount to root access on the Warewulf server. For this reason, the API is only accessible via localhost by default. Still, handle API credentials with care.

```
users:
  - name: admin
    password hash: $2b$05$5QVWDpiWE7L4SDL9CYdi3O/l6HnbNOLoXgY2sa1bQQ7aSBKdSqvsC
```

Passwords are stored as bcrypt2 hashes, which can be generated with mkpasswd.

```
$ mkpasswd --method=bcrypt
Password: # admin
$2b$05$5QVWDpiWE7L4SDL9CYdi3O/l6HnbNOLoXgY2sa1bQQ7aSBKdSqvsC
```

19.3 Node

- GET /api/nodes/: Get nodes
- POST /api/nodes/overlays/build: Build all overlays
- DELETE /api/nodes/{id}: Delete an existing node
- GET /api/nodes/{id}: Get a node

- PATCH /api/nodes/{id}: Update an existing node
- PUT /api/nodes/{id}: Add a node
- GET /api/nodes/{id}/fields: Get node fields
- POST /api/nodes/{id}/overlays/build: Build overlays for a node
- GET /api/nodes/{id}/raw: Get a raw node

19.4 Profile

- GET /api/profiles/: Get node profiles
- DELETE /api/profiles/{id}: Delete an existing profile
- GET /api/profiles/{id}: Get a node profile
- PATCH /api/profiles/{id}: Update an existing profile
- PUT /api/profiles/{id}: Add a profile

19.5 Image

- GET /api/images: Get all images
- DELETE /api/images/{name}: Delete an image
- GET /api/images/{name}: Get an image
- PATCH /api/images/{name}: Update or rename an image
- POST /api/images/{name}/build: Build an image
- POST /api/images/{name}/import: Import an image

19.6 Overlay

- GET /api/overlays/: Get overlays
- DELETE /api/overlays/{name}: Delete an overlay
- GET /api/overlays/{name}: Get an overlay
- PUT /api/overlays/{name}: Create an overlay
- GET /api/overlays/{name}/file: Get an overlay file

CHAPTER

TWENTY

CLUSTER NODES

Warewulf cluster node configuration is persisted in nodes.conf (also known as the "node registry" or "node database"). Editing this file directly is supported; but it is often better to manage it using the wwctl command.

Note

The nodes.conf file is a YAML document that can be edited directly or managed with configuration management; but its internal structure is technically undocumented and subject to change between versions. After Warewulf v4.6.0, the wwctl upgrade nodes command can be used to update a nodes.conf from a previous Warewulf v4 version.

Warning

When nodes.conf is edited directly, warewulfd must be restarted to reflect the changes.

```
systemctl restart warewulfd.service
```

20.1 Adding a Cluster Node

Adding a cluster node is as simple as running wwctl node add.

```
# wwctl node add n1 --ipaddr=10.0.2.1
Added node: n1
```

Several nodes can be added with a node range. In this case, the provided IP address is automatically incremented.

```
# wwctl node add n[2-4] --ipaddr=10.0.2.2
Added node: n2
Added node: n3
Added node: n4

# wwctl node list --net n[1-4]
NODE  NETWORK  HWADDR  IPADDR     GATEWAY  DEVICE
----  -------  ------  ------     -------  ------
n1    default  --      10.0.2.1   <nil>    --
n2    default  --      10.0.2.2   <nil>    --
n3    default  --      10.0.2.3   <nil>    --
n4    default  --      10.0.2.4   <nil>    --
```

20.2 Listing Nodes

Once you have configured one or more nodes, you can list them and their attributes with wwctl node list.

```
# wwctl node list n[1-5]
NODE NAME  PROFILES  NETWORK
---------  --------  -------
n1         default   --
n2         default   --
n3         default   --
n4         default   --
n5         default   --
```

You can also see the node's full attribute list by specifying --all.

```
# wwctl node list --all n1
NODE  FIELD              PROFILE  VALUE
----  -----              -------  -----
n1    Profiles           --       default
n1    Comment            default  This profile is automatically included for each node
n1    Ipxe               default  default
n1    RuntimeOverlay     default  hosts,ssh.authorized_keys
n1    SystemOverlay      default  wwinit,wwclient,fstab,hostname,ssh.host_keys,issue,resolv,udev.netname,
↪systemd.netname,ifcfg,ifupdown,NetworkManager,wicked,ignition
n1    Kernel.Args        default  quiet,crashkernel=no
n1    Init               default  /sbin/init
n1    Root               default  initramfs
n1    Resources[fstab]   default  [{"file":"/home","mntops":"defaults,nofail","spec":"warewulf:/home",
↪"vfstype":"nfs"},{"file":"/opt","mntops":"defaults,noauto,nofail,ro","spec":"warewulf:/opt","vfstype":
↪"nfs"}]
```

20.3 Setting Node Fields

Node fields are set using the wwctl node set command. A list of all available fields is available with wwctl node set --help.

You can also edit nodes as YAML data in an interactive editor using wwctl node edit.

20.3.1 List values

Some node fields, such as overlays and kernel arguments, accept a list of values. These may be specified as a comma-separated list or as multiple arguments.

To include an explicit comma in the value, enclose the value in inner-quotes.

```
wwctl node set n1 \
  --kernelargs 'quiet,crashkernel=no,nosplash' \
  --kernelargs='"console=ttyS0,115200"'
```

20.3.2 Un-setting Node Fields

Node fields can be cleared using the wwctl node unset command. Each field is specified as a boolean flag; the field is cleared when the flag is present.

```
wwctl node unset n1 --image
wwctl node unset n1 --kernelargs
```

To unset fields on a specific network interface, disk, partition, or filesystem, use the scoping flags --netname, --diskname, --partname, and --fsname.

```
wwctl node unset n1 --netname=secondary --ipaddr
```

If a scoping flag is given without any sub-field flags, the entire named sub-entity is removed. For --partname, an optional --diskname scopes the deletion to a specific disk; without it, the named partition is removed from all disks.

```
wwctl node unset n1 --netname=secondary
wwctl node unset n1 --partname=swap --diskname=/dev/vda
```

Tags can be selectively removed with --tag, --nettag, and --ipmitag.

```
wwctl node unset n1 --tag=localtime
wwctl node unset n1 --nettag=DNS1
```

A full list of available flags is shown by wwctl node unset --help.

Note

You can also un-set some fields by setting their value to UNDEF or UNSET with wwctl node set.

```
wwctl node set n1 --image=UNDEF
```

20.4 Configuring an Image

One of the main things to configure for a cluster node is the image that it should provision.

```
wwctl node set n1 \
  --image=rockylinux-9
```

Images are covered in more detail *in their own section.*

20.5 Configuring the Network

By default, network configurations are applied to a "default" network interface.

```
wwctl node set n1 \
  --netdev=eno1 \
  --hwaddr=00:00:00:00:00:01 \
  --ipaddr=10.0.2.1 \
  --netmask=255.255.255.0
```

Network interface configuration is covered in more detail *in its own section.*

20.6 Node Discovery

The MAC / hardware address (--hwaddr) of a cluster node can be automatically discovered by marking the node --discoverable. If a node attempts to provision against Warewulf using an interface that is unknown to Warewulf, its hardware address becomes associated with the first discoverable node. (Multiple discoverable nodes are sorted lexically, first by cluster, then by ID.)

Once a node has been discovered its "discoverable" field is automatically cleared.

20.7 Tags

Cluster nodes support multiple key-value pair tags. Tags may be applied to the node directly, to network interfaces, and even to IPMI interfaces.

```
wwctl node set n1 --tagadd="localtime=UTC"
wwctl node set n1 --nettagadd="DNS1=1.1.1.1"
```

20.8 Resources

Cluster nodes support generic "resources" that may hold arbitrarily complex YAML data. This data, along with tags, may be used by both distribution and site overlays.

```
nodeprofiles:
  default:
    resources:
      fstab:
        - spec: warewulf:/home
          file: /home
          vfstype: nfs
          mntops: defaults
          freq: 0
          passno: 0
        - spec: warewulf:/opt
          file: /opt
          vfstype: nfs
          mntops: defaults,ro
          freq: 0
          passno: 0
```

Resources can only be managed with wwctl node edit.

20.9 Importing Nodes From a File

You can import nodes into Warewulf by using the wwctl node import command. The file used must be in YAML format.

Warning

Importing a node configuration will fully overwrite the existing settings, including any customizations not present in the import file. If the node already exists and you wish to update it, ensure that the import file includes all the options you want to retain.

The YAML file must be a mapping of node names to their attributes, where each node is represented as a dictionary of attributes. To simplify the creation of the YAML file, you can use the wwctl node export command to export the current node configuration to a YAML file. This exported file can serve as a template for creating new nodes.

A minimal example of a YAML file looks like this:

```
n1:
  profiles:
    - default
  image name: rockylinux-9
  ipxe template: default
  kernel:
    args:
      - quiet
      - crashkernel=no
      - nosplash
      - console=ttyS0,115200
  network devices:
    default:
      type: ethernet
      device: eno1
      hwaddr: "00:00:00:00:00:01"
      ipaddr: 10.0.2.1
      netmask: 255.255.255.0
      gateway: 172.16.131.1
      tags:
      DNS1: 1.1.1.1
  primary network: default
```

This can be imported with the following command:

```
wwctl node import /path/to/nodes.yaml
```

CHAPTER

TWENTYONE

NODE PROFILES

Node profiles provide a way to scalably group node configurations together. Instead of redundant configurations for each node, you can set common fields in a profile and then apply one or more profiles to each node.

Profiles may, themselves, reference other profiles, supporting complex mixtures of profile configuration and negation.

21.1 The Default Profile

A default Warewulf installation will come with a single "default" profile pre-defined in nodes.conf.

```
# wwctl profile list
PROFILE NAME  COMMENT/DESCRIPTION
------------  -------------------
default       This profile is automatically included for each node
```

If the default profile exists, each new node automatically includes it when it is added.

You can view the fields of a profile with wwctl profile --all.

```
# wwctl profile list default --all
PROFILE  FIELD             VALUE
-------  -----             -----
default  Profiles          --
default  Comment           This profile is automatically included for each node
default  ClusterName       --
default  ImageName         --
default  Ipxe              default
default  RuntimeOverlay    hosts,ssh.authorized_keys
default  SystemOverlay     wwinit,wwclient,fstab,hostname,ssh.host_keys,issue,resolv,udev.netname,
↪systemd.netname,ifcfg,ifupdown,NetworkManager,wicked,ignition
default  Kernel.Version    --
default  Kernel.Args       quiet,crashkernel=no
default  Init              /sbin/init
default  Root              initramfs
default  PrimaryNetDev     --
default  Resources[fstab]  [{"file":"/home","mntops":"defaults,nofail","spec":"warewulf:/home","vfstype":
↪"nfs"},{"file":"/opt","mntops":"defaults,noauto,nofail,ro","spec":"warewulf:/opt","vfstype":"nfs"}]
```

wwctl node list --all indicates which profile defines each field.

```
# wwctl node list n1 --all
NODE  FIELD             PROFILE  VALUE
```

(continues on next page)

(continued from previous page)

```
----  -----            -------  -----
n1    Profiles         --       default
n1    Comment          default  This profile is automatically included for each node
n1    Ipxe             default  default
n1    RuntimeOverlay   default  hosts,ssh.authorized_keys
n1    SystemOverlay    default  wwinit,wwclient,fstab,hostname,ssh.host_keys,issue,resolv,udev.netname,
↪systemd.netname,ifcfg,ifupdown,NetworkManager,wicked,ignition
n1    Kernel.Args      default  quiet,crashkernel=no
n1    Init             default  /sbin/init
n1    Root             default  initramfs
n1    Resources[fstab] default  [{"file":"/home","mntops":"defaults,nofail","spec":"warewulf:/home",
↪"vfstype":"nfs"},{"file":"/opt","mntops":"defaults,noauto,nofail,ro","spec":"warewulf:/opt","vfstype":
↪"nfs"}]
```

21.2 Setting Profile Fields

(Almost) any node fields can be set on a profile, but some fields don't really make sense anywhere but a node (e.g., --hwaddr and --ipaddr).

```
wwctl profile set default \
  --image=rockylinux-9 \
  --netmask=255.255.255.0
```

21.2.1 Un-setting Profile Fields

Profile fields can be cleared using the wwctl profile unset command, which works the same way as wwctl node unset.

```
wwctl profile unset default --image
wwctl profile unset default --netname=default --netmask
```

21.3 Multiple Profiles

It's possible to create multiple profiles, and even to apply multiple profiles to each node.

```
wwctl profile add net
wwctl profile set net --netmask=255.255.255.0

wwctl profile add image
wwctl profile set image --image=rockylinux-9

wwctl node set n1 --profile="default,net,image"
```

```
# wwctl node list n1 --all
NODE  FIELD                PROFILE  VALUE
----  -----                -------  -----
n1    Profiles             --       default,net,image
n1    Comment              default  This profile is automatically included for each node
n1    ImageName            image    rockylinux-9
n1    Ipxe                 default  default
n1    RuntimeOverlay       default  hosts,ssh.authorized_keys
```

(continues on next page)

(continued from previous page)

```
n1    SystemOverlay             default   wwinit,wwclient,fstab,hostname,ssh.host_keys,issue,resolv,udev.
↪netname,systemd.netname,ifcfg,ifupdown,NetworkManager,wicked,ignition
n1    Kernel.Args               default   quiet,crashkernel=no
n1    Init                      default   /sbin/init
n1    Root                      default   initramfs
n1    NetDevs[default].Netmask  net       255.255.255.0
n1    Resources[fstab]          default   [{"file":"/home","mntops":"defaults,nofail","spec":"warewulf:/home
↪","vfstype":"nfs"},{"file":"/opt","mntops":"defaults,noauto,nofail,ro","spec":"warewulf:/opt","vfstype":
↪"nfs"}]
```

Using multiple profiles makes it easy to work with multiple, heterogeneous groups of cluster nodes and to test new configurations on smaller subsets of nodes. For example, you can use this method to run a different kernel on only a subset or group of cluster nodes without changing any other node attributes.

21.4 Profile Precedence and Combinations of Profiles

In case of scalar fields, the right-most profile in the node's profile list takes precedence. Field values set directly on nodes take precedence over profile field values. However, things behave differently for field types, like lists, where it makes sense to combine their values.

Let's assume we have in addition to the node profile two more profiles: default and profile_a. The field root serves as an example for a scalar field and runtime overlay for list typed fields.

We start with a configuration where the node uses only the default profile.

```
nodeprofiles:
  profile_a:
    runtime overlay:
      - runtime_overlay_from_profile_a
    root: root_from_profile_a
  default:
    runtime overlay:
      - runtime_overlay_from_default
    root: root_from_default
nodes:
 n1:
   profiles:
     - default
   runtime overlay:
     - runtime_overlay_from_node_profile
   root: root_from_node_profile
```

Next we filter the output a little to focus on one field at time.

```
# wwctl node list n1 --all | grep -E '(NODE|----|Root)'
 NODE  FIELD            PROFILE              VALUE
 ----  -----            -------              -----
 n1    Root             SUPERSEDED           root_from_node_profile
```

As expected, the value originates from the node profile because the node profile has highest precedence.

Things are different for the list typed field runtime overlay.

```
# wwctl node list n1 --all | grep -E '(NODE|----|Runtime)'
  NODE  FIELD            PROFILE     VALUE
  ----  -----            -------     -----
  n1    RuntimeOverlay   default,n1  runtime_overlay_from_default,runtime_overlay_from_node_
↪profile
```

Here the value from the node profile is appended to the value of the default profile.

Next we remove the value for root from the node profile and add profile_a to the list of profiles for the node.

```
nodes:
  n1:
    profiles:
      - default
      - profile_a
    runtime overlay:
      - runtime_overlay_from_node_profile
```

First we check the root field and see that it set from profile_a as profile_a is now the profile with highest precedence.

```
# wwctl node list n1 --all | grep -E '(NODE|----|Root)'
  NODE  FIELD            PROFILE             VALUE
  ----  -----            -------             -----
  n1    Root             profile_a           root_from_profile_a
```

The value of runtime overlay is now a combination of the values of all three profiles.

```
# wwctl node list n1 --all | grep -E '(NODE|----|Runtime)'
NODE  FIELD            PROFILE             VALUE
----  -----            -------             -----
n1    RuntimeOverlay   default,profile_a,n1  runtime_overlay_from_default,runtime_overlay_from_
↪profile_a,runtime_overlay_from_node_profile
```

We get the same result when making use of nested profiles, that is when we add profile_a to the list of profiles within the default profile.

```
nodeprofiles:
   default:
     profiles:
       - profile_a
     ...
   ...
nodes:
  n1:
    profiles:
      - default
    ...
```

21.5 Negating Profiles

Profiles may be negated by later profiles. For example, a profile list p2,~p1 adds the profile p2 to a node and removes a previously-applied p1 profile from a node.

```
nodeprofiles:
  p1:
    runtime overlay:
      - runtime_overlay_from_p1
  p2:
    profiles:
      - p1
    runtime overlay:
      - runtime_overlay_from_p2
nodes:
  n1:
    profiles:
      - p2
      - ~p1
```

The value of runtime overlay is then just runtime_overlay_from_p2.

```
# wwctl node list n2 --all
NODE  FIELD          PROFILE  VALUE
----  -----          -------  -----
n2    Profiles       --       p2,~p1
n2    RuntimeOverlay  p2      runtime_overlay_from_p2
```

21.6 Using Profiles Effectively

There are a lot of ways to use profiles to facilitate complex cluster configurations; but they are not required. It is completely possible to not use profiles at all, and to simply set all fields directly on cluster nodes.

If you do use profiles, some fields lend themselves most naturally to being set on profiles. Network subnet masks (--netmask) and gateways (--gateway) are common profile fields, as is --image. Most *IPMI* fields make sense on a profile, and it is also common to configure tags and resources on a profile for easy application to multiple nodes.

Node-specific information, like HW/MAC addresses (--hwaddr) and IP addresses (--ipaddr, --ipmiaddr) should always be put in a node configuration rather than a profile configuration.

CHAPTER

TWENTYTWO

NETWORK INTERFACES

By default, network configurations are applied to a "default" network interface.

```
wwctl node set n1 \
  --netdev=eno1 \
  --hwaddr=00:00:00:00:00:01 \
  --ipaddr=10.0.2.1 \
  --netmask=255.255.255.0
```

Each cluster node can have multiple network interfaces, differentiated by specifying --netname.

```
wwctl node set n1 \
  --netname=infiniband \
  --type=infiniband \
  --netdev=ib1 \
  --ipaddr=10.0.3.1 \
  --netmask=255.255.255.0
```

Warning

Due to the way network interface names are assigned by the Linux kernel, and later reassigned by udev and systemd, the use of eth0, eth1, etc. as interface is strongly discouraged. We recommend the use of the original predictable names assigned to the interfaces (e.g., eno1), as otherwise an interface may fail to be named correct if its desired name conflicts with the kernel-assigned name of another interface during the boot process.

22.1 Network Tags

Each network device can optionally have one or more key-value pair tags.

```
wwctl node set n1 --nettagadd="DNS1=1.1.1.1"
```

22.2 Bonding

Support for bonded / link aggregation network interfaces depends on the network overlay being used.

The ifcfg and NetworkManager overlays can configure a network bond like this:

```
network devices:
  bond0:
```

(continues on next page)

(continued from previous page)

```
    type: Bond
    device: bond0
    ipaddr: 192.168.3.100
    netmask: 255.255.255.0
  en1:
    device: en1
    hwaddr: e6:92:39:49:7b:03
    tags:
      master: bond0
  en2:
    device: en2
    hwaddr: 9a:77:29:73:14:f1
    tags:
      master: bond0
```

22.3 VLAN

You can set the type also to vlan.

Some network configuration systems use the network device name (e.g., of the form eno1.100) to configure VLANs. Other network systems need additional network tags:

- vlan_id: configures the VLAN ID of the interface
- parent_device: configures which physical interface to use

```
wwctl node set \
  --netdev vlan42 \
  --ipaddr 10.0.42.1 \
  --netmask 255.255.252.0 \
  --netname iband \
  --type vlan \
  --nettagadd "vlan_id=42,parent_device=eth0" \
  n001
```

22.4 Static Routes

The included Warewulf network overlays support the configuration of static routes using a network tag of the form route<N>=<dest>,<gateway>.

```
wwctl node set n001 \
  --nettagadd "route1=192.168.2.0/24,192.168.1.254"
```

22.5 Type

When using the NetworkManager overlay template the type attribute determines how the connection is configured. If not set, it defaults to ethernet. To correctly configure IPoIB with NetworkManager it is required that type is set to infiniband.

CHAPTER

TWENTYTHREE

IPMI

Warewulf can use IPMI to control cluster node power state or to connect to a serial console.

23.1 Configuration

Typically, common settings for IPMI interfaces are set on a profile, leaving only the IP address set per-node.

If --ipmiwrite is set to true, the wwinit overlay will write the desired IPMI configuration to the node's BMC during boot.

```
wwctl profile set default \
  --ipminetmask=255.255.255.0 \
  --ipmiuser=admin \
  --ipmipass=passw0rd \
  --ipmiinterface=lanplus \
  --ipmiwrite

wwctl node set n1 \
  --ipmiaddr=192.168.2.1
```

Additionally, a vlan ipmi tag can be used to set the IPMI VLAN ID.

```
wwctl profile set default \
  --ipmitagadd vlan=42
```

A bit-rate ipmi tag can be used to set the Serial over LAN bit rate (defaults to 38.4). Typical options are 19.2, 38.4, and 115.2.

```
wwctl profile set default \
  --ipmitagadd bit-rate=115.2
```

wwctl node list has a specific overview for IPMI settings.

```
# wwctl node list --ipmi
NODE  IPMI IPADDR   IPMI PORT  IPMI USERNAME  IPMI INTERFACE
----  -----------  ---------  -------------  --------------
n1    192.168.1.11  --         hwadmin        lanplus
n2    192.168.1.12  --         hwadmin        lanplus
n3    192.168.1.13  --         hwadmin        lanplus
n4    192.168.1.14  --         hwadmin        lanplus
```

23.2 Power

The wwctl power command can query and set the current power state of cluster nodes.

```
wwctl power status n1 # query the current power status
wwctl power off n1 # power off a cluster node
wwctl power on n1 # power on a cluster node
wwctl power reset n1 # forcibly reboot a node
wwctl power soft n1 # ask a node to shut down gracefully
wwctl power cycle n1 # power a cluster node off, then back on
```

Node ranges are supported; e.g., n[1-10].

23.3 Console

If your node is setup to use serial over lan (SOL), Warewulf can connect a console to the node.

```
# wwctl node console n1
```

23.4 Customization

Warewulf doesn't manage IPMI interfaces directly, but uses ipmitool. This is configured with a template which defines Warewulf's IPMI behavior.

```
{{ $cmd := "ipmitool" }}
{{ if .Interface }}{{ $cmd = cat $cmd "-I" .Interface }}{{ end }}
{{ if .EscapeChar }}{{ $cmd = cat $cmd "-e" .EscapeChar }}{{ end }}
{{ if .Port }}{{ $cmd = cat $cmd "-p" .Port }}{{ end }}
{{ if .Ipaddr }}{{ $cmd = cat $cmd "-H" .Ipaddr }}{{ end }}
{{ if .UserName }}{{ $cmd = cat $cmd "-U" (printf "\"%s\"" .UserName) }}{{ end }}
{{ if .Password }}{{ $cmd = cat $cmd "-P" (printf "\"%s\"" .Password) }}{{ end }}
{{ if eq .Cmd "PowerOn" }}{{ $cmd = cat $cmd "chassis power on" }}
{{ else if eq .Cmd "PowerOff" }}{{ $cmd = cat $cmd "chassis power off" }}
{{ else if eq .Cmd "PowerCycle" }}{{ $cmd = cat $cmd "chassis power cycle" }}
{{ else if eq .Cmd "PowerReset" }}{{ $cmd = cat $cmd "chassis power reset" }}
{{ else if eq .Cmd "PowerSoft" }}{{ $cmd = cat $cmd "chassis power soft" }}
{{ else if eq .Cmd "PowerStatus" }}{{ $cmd = cat $cmd "chassis power status" }}
{{ else if eq .Cmd "SDRList" }}{{ $cmd = cat $cmd "sdr list" }}
{{ else if eq .Cmd "SensorList" }}{{ $cmd = cat $cmd "sensor list" }}
{{ else if eq .Cmd "Console" }}{{ $cmd = cat $cmd "sol activate" }}
{{ end }}
{{- $cmd -}}
```

A different template can be used to change the IPMI behavior using the --ipmitemplate field. Referenced templates must be located in warewulf.conf:Paths.Datadir (/usr/lib/warewulf/bmc/).

All IPMI specific fields are accessible in the template:

Parameter	Template variable
--ipmiaddr	.Ipaddr
--ipminetmask	.Netmask
--ipmiport	.Port
--ipmigateway	.Gateway
--ipmiuser	.UserName
--ipmipass	.Password
--ipmiinterface	.Interface
--ipmiwrite	.Write
--ipmiescapechar	.EscapeChar
--ipmitemplate	.Template

Additionally, the .Cmd variable includes the relevant wwctl power subcommand.

- PowerOn
- PowerOff
- PowerCycle
- PowerReset
- PowerSoft
- PowerStatus
- SDRList
- SensorList
- Console

CHAPTER

TWENTYFOUR

PROVISIONING DISKS

As a tech preview, Warewulf provides structures to define disks, partitions, and file systems. These structures can generate a configuration for Ignition to provision partitions and file systems dynamically on cluster nodes, or with sfdisk, mkfs, and mkswap during boot.

Ignition can, for example, create swap partitions or /scratch file systems.

24.1 Requirements

Partition and file system creation requires both ignition and sgdisk to be installed in the image.

24.1.1 Rocky Linux

```
dnf install ignition gdisk
```

Note

Packages for Ignition are not currently available for Rocky Linux 8, but it is available for Rocky Linux 9 as part of "appstream."

24.1.2 openSuse Leap

```
zypper install ignition gptfdisk
```

24.2 Disks and partitions

A node or profile can have several disks. Each disk is identified by the path to its block device. Each disk holds a map to its partitions and a bool switch to indicate if an existing, non-matching partition table should be overwritten.

Each partition is identified by its label. The partition number can be omitted, but specifying it is recommended as Ignition may fail without it. Partition sizes should also be set (specified in MiB), except for the last partition: if no size is given, the maximum available size is used. Each partition has the switches should_exist and wipe_partition_entry which control the partition creation process (via the --partcreate and --partwipe flags). When omitting a partition number the wipe_partition_entry should be true, as this allows ignition to replace the existing partition.

```
wwctl node set n1 \
  --diskname /dev/vda --diskwipe \
  --partname scratch --partcreate --partwipe --partnumber 1
```

24.3 File systems

File systems are identified by their underlying block device, preferably using the /dev/by-partlabel format. Except for a swap partition, an absolute path for the mount point must be specified for each file system. Depending on the image used, valid formats are btrfs, ext3, ext4, and xfs. Each file system has the switch wipe_filesystem to control whether an existing file system is wiped.

```
wwctl node set n1 \
  --diskname /dev/vda --partname scratch \
  --fsname scratch --fsformat btrfs --fspath /scratch
```

24.4 Boot-time configuration

Ignition uses systemd, as the underlying sgdisk command relies on dbus notifications.

1. ignition-disks-ww4.service uses Ignition to create the specified partitions and file systems.
2. ww4-disks.target depends on a matching .mount unit for each mounted file system.
3. Each .mount creates the necessary mount points in the root file system and mounts the provisioned file systems during boot.

These services and mount units are generated by the ignition overlay and depend on the existence of the file /warewulf/ignition.json, also generated by the ignition overlay.

24.5 Example disk configurations

This command formats a btrfs file system on a "scratch" partition of "vda" and mounts it at /scratch.

```
wwctl node set n1 \
  --diskname /dev/vda --diskwipe \
  --partname scratch --partcreate --partnumber 1 \
  --fsname scratch --fsformat btrfs --fspath /scratch
```

This command adds a swap partition to the "vda" disk.

```
wwctl node set n1 \
  --diskname /dev/vda \
  --partname swap --partsize=1024 --partnumber 2 \
  --fsname swap --fsformat swap --fspath swap
```

24.6 Re-using or wiping disks

For empty disks the desired configuration is created and the filesystems are mounted. If partitions or file systems already exist on the disk, ignition tries to reuse existing file systems by default.

To ignore existing file systems and provision fresh file systems on each boot, specify the --fswipe flag for that filesystem, and --diskwipe for the disk, as necessary.

If you would like to re-use existing partitions but want to replace existing file systems once, you may

- wipe the existing data with tools like wipefs or dd[1]; or

[1] With wipefs you have to remove the filesystem *and* partition information. E.g., use wipefs -fa /dev/vda* to remove all file system information and partition information.

- set the --fswipe flag and remove it after one reboot.

See the upstream ignition documentation for additional information.

24.7 Swap and image memory usage

Warewulf images run entirely in memory. Configuring a local swap partition can allow the kernel to reclaim that RAM for applications — but only under the right conditions. Whether swap can free image memory depends on which root filesystem type the node uses.

24.7.1 tmpfs root

When the root filesystem is tmpfs (the default for two-stage dracut boot, or when --root=tmpfs is set explicitly), the image lives in the page cache. The Linux kernel can swap tmpfs pages to a local swap device exactly as it would any other anonymous memory. Adding swap therefore lets the kernel evict cold image pages to disk and reclaim that RAM for running workloads.

This is the recommended configuration for nodes with large images relative to available RAM.

24.7.2 initramfs root (single-stage boot default)

The default single-stage boot places the image in an initramfs root, which is an instance of ramfs. Unlike tmpfs, ramfs pages are pinned in memory: the kernel will never swap them out. Configuring swap on a node with an initramfs root will **not** free any memory used by the image.

If you are using single-stage boot and want swap to help with image memory, switch to tmpfs root first:

```
wwctl profile set default --root=tmpfs
```

For background on tmpfs NUMA interleaving and size limits, see *tmpfs and NUMA*.

Note

Two-stage dracut boot always uses tmpfs regardless of the --root setting. If you are already using dracut, no additional root configuration is needed.

24.7.3 Example: configuring swap to reclaim image memory

This example provisions a swap partition on the local disk of each node and activates it at boot, enabling the kernel to reclaim RAM occupied by the node image.

1. Configure tmpfs root (skip if using two-stage dracut boot):

```
wwctl profile set default --root=tmpfs
```

2. Add a swap partition to the node or profile disk configuration. This example adds an 8 GiB swap partition as the first partition of /dev/vda:

```
wwctl profile set default \
  --diskname /dev/vda --diskwipe \
  --partname swap --partcreate --partnumber 1 --partsize=8192 \
  --fsname swap --fsformat swap --fspath swap
```

3. Add the required overlays to the system overlay so that the swap partition is formatted and activated at boot. The -O / --system-overlays flag replaces the entire overlay list, so include any overlays already configured for the profile. Use Ignition (recommended for Rocky Linux 9 and openSUSE):

```
wwctl profile set default \
  -O wwinit,wwclient,fstab,hostname,ssh.host_keys,systemd.netname,NetworkManager,ignition,systemd.
↪swap
```

Or, for systems without Ignition (e.g., Rocky Linux 8), use the sfdisk and mkswap overlays instead:

```
wwctl profile set default \
  -O wwinit,wwclient,fstab,hostname,ssh.host_keys,systemd.netname,NetworkManager,sfdisk,mkswap,
↪systemd.swap
```

4. Rebuild the dracut initramfs with the tools needed to provision the disk during the two-stage boot (skip for single-stage boot):

```
# Ignition path
wwctl image exec rockylinux-9 -- /usr/bin/dracut --force --no-hostonly \
  --add wwinit --add ignition --regenerate-all

# sfdisk/mkswap path
wwctl image exec rockylinux-8 -- /usr/bin/dracut --force --no-hostonly \
  --add wwinit --install sfdisk --install blockdev --install udevadm \
  --install mkswap --regenerate-all
```

5. Reboot nodes to apply the new configuration.

24.7.4 Verifying swap is active

After reboot, confirm that the swap device is active:

```
swapon --show
```

Then check the overall memory picture with free -h:

```
$ free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi        12Gi       256Mi       120Mi       2.8Gi       2.8Gi
Swap:          8.0Gi          0Ki       8.0Gi
```

At idle, most of the node's RAM is occupied by the image. The Swap: line shows the available device but near-zero usage at this point.

24.7.5 Confirming that the image gets swapped out

To demonstrate that the kernel evicts image pages to swap when an application needs memory, first note the image size with df -h /:

```
df -h /
```

This shows how much tmpfs space the image occupies — that is the amount of RAM currently holding the image.

Now apply memory pressure using stress-ng (install it in the OS image if not already present). The allocation must exceed **available** RAM — not just total RAM — to force the kernel to evict image pages. Compute the target from MemTotal:

```
stress-ng --vm 1 \
  --vm-bytes $(awk '/MemTotal/{printf "%dM", $2*0.9/1024}' /proc/meminfo) \
  --vm-keep -t 60s &
```

While stress-ng is running, observe memory usage:

```
free -h
```

```
$ free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi        15Gi        32Mi       120Mi       512Mi       192Mi
Swap:          8.0Gi       4.2Gi       3.8Gi
```

The Swap: used value has grown by roughly the size of the image. The kernel has evicted cold image pages to swap, making that RAM available to the application. The application can access the full physical memory of the node, not just what is left over after the image is loaded.

After stress-ng finishes, the evicted image pages remain on swap until they are accessed again, so free -h will continue to show swap usage until the node is under less pressure and pages are faulted back in as needed.

24.7.6 Moving image pages to swap proactively

Rather than simulating a workload, you can instruct the kernel to push image pages to swap directly. On Linux 6.1 and later with cgroup v2, write the desired reclaim size to memory.reclaim:

```
# Request that the kernel reclaim up to 4 GiB from the root cgroup
# (adjust to match your image size shown by df -h /)
echo "4G" > /sys/fs/cgroup/memory.reclaim
```

If the kernel cannot reclaim the full requested amount it returns an error:

```
-bash: echo: write error: Resource temporarily unavailable
```

This is expected and benign. It means the kernel reclaimed as much as it could but the reclaimable pool was exhausted before reaching the target.

The kernel will swap out reclaimable pages until the target is met or exhausted, then stop. Run free -h immediately afterwards to see the reduction in Mem: used and corresponding increase in Swap: used.

This works best when run early in the boot process, before applications start, when nearly all anonymous memory belongs to the image. To automate it, add a systemd unit to a custom overlay that runs at local-fs.target:

```
[Unit]
Description=Reclaim OS image memory to swap
After=local-fs.target
ConditionPathExists=/sys/fs/cgroup/memory.reclaim

[Service]
Type=oneshot
ExecStart=/bin/sh -c 'echo "$(df --output=used / | tail -1)K" > /sys/fs/cgroup/memory.reclaim'

[Install]
WantedBy=multi-user.target
```

This reads the actual image size from df and requests that exact amount be reclaimed, so it adapts automatically as the image grows or shrinks.

24.8 Provision to disk

New in Warewulf v4.6.2

As a tech preview, the Warewulf two-stage boot process can provision the node image to local storage.

Warning

This functionality is a technology preview and should be used with care. Pay specific attention to wipeFilesytem and similar settings.

Note

Warewulf doesn't install a bootloader to the disk or add UEFI entries. Nodes still request an image and configuration from the Warewulf server on every boot.

Note

While provisioning to disk should be possible during a single-stage boot, not all features are available:

- Warewulf does not perform hardware detection to ensure that necessary kernel modules are loaded prior to init.
- Warewulf does not load udev to ensure that /dev/disk/by-* symlinks are available prior to init.

24.8.1 With Ignition

Warewulf needs a prepared file system to deploy the image to. Warewulf can provision this file system using Ignition. To use Ignition, include ignition in your system overlay. The ignition overlay provisions disks during init and, optionally, during the first stage of a two-stage boot. This allows the root file system to be provisioned before the image is loaded.

```
wwctl node set wwnode1 \
  --diskname /dev/vda --diskwipe \
  --partname rootfs --partcreate --partnumber 1 \
  --fsname rootfs --fsformat ext4 --fspath /
```

In order to allow Dracut to provision the disk, partition, and file system, Ignition must be included in the Dracut image.

```
wwctl image exec rockylinux-9 -- /usr/bin/dracut --force --no-hostonly --add wwinit --add ignition --
↪regenerate-all
```

The necessary file system may alternatively be prepared out-of-band.

24.8.2 With sfdisk and mkfs

Systems that do not have access to Ignition (e.g., Rocky Linux 8) can provision the root file system using a combination of sfdisk and mkfs. To use them, include sfdisk and mkfs in your system overlay. The sfdisk and mkfs overlays provision disk and file systems during the first stage of a two-stage boot. This allows the root file system to be provisioned before the image is loaded.

Configure the sfdisk and mkfs overlays using resources:

```
wwctl node set wwnode1 \
  --diskname /dev/vda --diskwipe \
  --partname rootfs --partcreate --partnumber 1 \
  --fsname rootfs --fsformat ext4 --fspath /
```

In order to allow Dracut to provision the disk, partition, and file system, some additional commands must be included in the Dracut image, depending on which functionality is used:

- **sfdisk:** writes the partition table
  - **blockdev:** used to re-read the partition table after writing
  - **udevadm:** used to trigger udev events after writing the partition table
- **mkfs:** formats file systems (may also require file-system-specific commands like mkfs.ext4)
  - **mkfs.ext4**, **mkfs.btrfs**, etc: used by mkfs to format specific file systems
  - **wipefs:** used to determine if a file system already exists

```
wwctl image exec rockylinux-8 -- /usr/bin/dracut --force --no-hostonly \
  --add wwinit \
  --install sfdisk \
  --install blockdev \
  --install udevadm \
  --install mkfs \
  --install mkfs.ext4 \
  --install wipefs \
  --regenerate-all
```

24.8.3 Configuring the root device

Set the desired storage device for the image using the --root parameter.

```
wwctl node set wwnode1 --root /dev/disk/by-partlabel/rootfs
```

24.8.4 Known Problems

If the partition table on the disk isn't properly readable the command sgdisk --zap-all (which is used by Ignition to wipe the partition table) returns with code 2. This is interpreted by Ignition < 2.16.2 as an error, and no partitions or filesystems are created. Since the partition table is still wiped, partitioning and formatting should succeed on the next boot.

CHAPTER

TWENTYFIVE

OS IMAGES

Warewulf operating system (OS) images are a "Virtual Node File System" (VNFS) that serves as an image for cluster nodes. This is similar to a "golden master" image, except that the image source exists mutably within a directory on the Warewulf control node (e.g. a chroot()).

Warewulf OS images have several similarities to Linux containers; so Warewulf v4 integrates directly within the container ecosystem to facilitate the process of image creation and image management: images can be built, for example, with Docker, Podman, or Apptainer, and imported directly from OCI registries or local container image archives. But you can also still build your own chroot directories manually.

25.1 Structure

A Warewulf image is a directory that populates the base runtime root file system of a cluster node. The image source directory must contain a single rootfs directory which represents the actual root directory for the image.

```
/var/lib/warewulf/chroots/rockylinux-9
└── rootfs
    ├── afs
    ├── bin -> usr/bin
    ├── boot
    ├── dev
    ├── etc
    ├── home
    ├── lib -> usr/lib
    ├── lib64 -> usr/lib64
    ├── media
    ├── mnt
    ├── opt
    ├── proc
    ├── root
    ├── run
    ├── sbin -> usr/sbin
    ├── srv
    ├── sys
    ├── tmp
    ├── usr
    └── var
```

25.2 Importing Images

Before any cluster nodes can be provisioned, you must import an image. Images may be imported from an OCI registry, a local OCI archive, or a local directory or Apptainer sandbox.

25.2.1 OCI Registry

You can import OS images from an OCI registry, public or private.

```
# wwctl image import docker://ghcr.io/warewulf/warewulf-rockylinux:8 rockylinux-8
Getting image source signatures
Copying blob d7f16ed6f451 done
Copying config da2ca70704 done
Writing manifest to image destination
Storing signatures
[LOG]       info unpack layer:␣
↪sha256:d7f16ed6f45129c7f4adb3773412def4ba2bf9902de42e86e77379a65d90a984
Updating the image's /etc/resolv.conf
Building image: rockylinux-8
```

Note

Most images in Docker Hub are not "bootable": they typically do not include a kernel, and likely don't include any init system. For this reason, don't expect just any image from DockerHub (e.g. docker://rockylinux or docker://debian) to boot properly with Warewulf.

The Warewulf project maintains a set of example OS images that are configured to boot when used with Warewulf. These images can be imported directly into Warewulf or used as base images for local custom images.

A few environmental variables can be used to control communication with the OCI registry:

```
WAREWULF_OCI_USERNAME
WAREWULF_OCI_PASSWORD
WAREWULF_OCI_NOHTTPS
```

They can be overwritten with --nohttps, --username and --password.

```
# wwctl image import --username tux --password supersecret docker://ghcr.io/privatereg/rocky:8
```

You can also set HTTP_PROXY, HTTPS_PROXY, and NO_PROXY (or their lower-case versions) to use a proxy during wwctl image import.

```
export HTTPS_PROXY=squid.localdomain
wwctl image import docker://ghcr.io/warewulf/warewulf-rockylinux:8
```

See ProxyFromEnvironment for more information.

Note

OCI and ORAS registries typically use HTTPS, so you probably need to set HTTPS_PROXY or https_proxy rather than the HTTP variants.

The above is just an example. Consideration should be done before doing it this way if you are in a security sensitive environment or shared environments as this command line will show up in the process table.

25.2.2 Local OCI Archive

It is also possible to import an image from a local OCI archive. For example, Podman can save a .tar archive of an OCI image.

```
podman save ghcr.io/warewulf/warewulf-rockylinux:8 >rockylinux-8.tar
wwctl image import rockylinux-8.tar rockylinux-8
```

25.2.3 Local Directories and Apptainer Sandboxes

Chroot directories and Apptainer sandbox images can also be imported directly.

```
apptainer build --sandbox ./rockylinux-8/ docker://ghcr.io/warewulf/warewulf-rockylinux:8
wwctl image import ./rockylinux-8/ rockylinux-8
```

25.3 Listing Imported Images

Once the image has been imported, you can list them all with wwctl image list.

```
# wwctl image list
IMAGE NAME
----------
rockylinux-8
```

Additional detail is available using wwctl image list --long, among others. (See --help for more options.)

```
# wwctl image list --long
IMAGE NAME    NODES  KERNEL VERSION      CREATION TIME        MODIFICATION TIME ␣
↪ SIZE
----------    -----  --------------      -------------        -----------------    ----
rockylinux-8  0      4.18.0-553.30.1     11 Feb 25 13:57 MST  11 Feb 25 13:57 MST  1.4 GiB
```

25.4 Modifying Images Interactively

An image that has been imported into Warewulf remains mutable, and can be modified on the Warewulf server. For example, you can "shell" into the image and make changes interactively.

```
# wwctl image shell rockylinux-8
[warewulf:rockylinux-8] /# dnf -y install apptainer
[...]

Installed:
  apptainer-1.3.6-1.el8.aarch64
  fakeroot-1.33-1.el8.aarch64
  fakeroot-libs-1.33-1.el8.aarch64
  fuse3-libs-3.3.0-19.el8.aarch64
  lzo-2.08-14.el8.aarch64
  squashfs-tools-4.3-21.el8.aarch64
```

(continues on next page)

(continued from previous page)

```
Complete!
```

25.4.1 Binding Files and Directories

You can --bind directories from the Warewulf server into the image when using the exec command. This is particularly useful for installing locally-built packages.

```
# wwctl image shell --bind /var/lib/mock/rocky+epel-9-$(arch)/result:/mnt
[warewulf:rockylinux-8] /# dnf -y install /mnt/warewulf-dracut-*.noarch.rpm
```

Note

As with any mount command, both the source and the target must exist. This is why the example uses the /mnt/ directory location, as it is almost always present and empty in every Linux distribution (as prescribed by the LSB file hierarchy standard).

Files may also be automatically bound into the image during wwctl image shell by configuring warewulf.conf:image mounts.

```
image mounts:
- source: /etc/resolv.conf
  dest: /etc/resolv.conf
  readonly: true
```

Note

Instead of readonly: true you can set copy: true. This causes the source file to be copied to the image and removed if it was not modified. This can be useful for files used for registrations.

When the command completes, if anything within the image changed, the image will be rebuilt into a bootable static object automatically. (To skip the automatic image rebuild, specify --build=false.)

If the files /etc/passwd or /etc/group were updated, there will be an additional check to confirm if the users are in sync as described in the *syncuser* section.

25.4.2 Specifying a Prompt

Warewulf sets a custom prompt during a wwctl image shell session. This prompt may be customized using the WW_PS1 variable, which is used to construct the final PS1 variable for the shell.

```
# export WW_PS1="\u@\h:\w\$ "
# wwctl image shell rockylinux-8
[warewulf:rockylinux-8] root@rocky:/$
```

25.4.3 Shell History

By default, Warewulf image shell sessions don't retain history; but you can specify a history file by specifying WW_HISTFILE. Note that this file is stored within the image; you may want to *Excluding Files* it when the image is built.

25.4.4 Running Specific Commands

A single command can also be executed in an image, as an alternative to an interactive shell.

```
wwctl image exec rockylinux-8 -- /usr/bin/dnf -y install apptainer
```

25.5 Building Images

Warewulf images must be built (e.g., with wwctl image build) into compressed images for distribution to cluster nodes during provisioning.

```
# wwctl image build rockylinux-9
Building image: rockylinux-9
Created image for Image rockylinux-9: /var/lib/warewulf/provision/images/rockylinux-9.img
Compressed image for Image rockylinux-9: /var/lib/warewulf/provision/images/rockylinux-9.img.gz
```

25.5.1 Excluding Files

Warewulf can exclude files from an image to prevent them from being delivered to the compute node. This is typically used to reduce the size of the image when some files are unnecessary.

Patterns for excluded files are read from the file /etc/warewulf/excludes in the image itself. For example, the default Rocky Linux images exclude these paths:

```
/boot/
/usr/share/GeoIP
```

/etc/warewulf/excludes supports the patterns implemented by filepath.Match.

25.5.2 Exit Script

Warewulf executes the script /etc/warewulf/image_exit.sh in the image after a wwctl image shell or wwctl image exec. If --build is specified, image_exit.sh is executed before (re)building the image. This is typically used to remove cache or log files that may have been generated by the executed command or interactive session.

For example, the default Rocky Linux images runs dnf clean all to remove any package repository caches that may have been generated.

25.6 Defining New Images

It is absolutely possible to import an OS image into Warewulf and make all changes interactively with wwctl image shell; but it is often better to define new images with a container image definition file. This can be done using the OCI and Singularity (Apptainer) ecosystems.

25.6.1 Podman

An OCI Containerfile can build from an existing container image to add local customizations.

```
FROM ghcr.io/warewulf/warewulf-rockylinux:9

RUN dnf -y install epel-release \
    && dnf -y install apptainer
```

```
# podman build . --file Containerfile --tag custom-image
[...]
Successfully tagged localhost/custom-image:latest

# wwctl image import $(podman image mount localhost/custom-image) custom-image
# podman image unmount localhost/custom-image
```

25.6.2 Apptainer

It is absolutely possible to create an OCI base image from scratch, but it is particularly easy to do with Apptainer.

Consider the following file called warewulf-rockylinux-9.def:

```
Bootstrap: yum
MirrorURL: https://download.rockylinux.org/pub/rocky/9/BaseOS/x86_64/os/
Include: dnf

%post
dnf -y install --allowerasing \
  NetworkManager \
  basesystem \
  bash \
  curl-minimal \
  kernel \
  nfs-utils \
  openssh-server \
  systemd

dnf -y remove \
  glibc-gconv-extra
rm -rf /boot/* /run/*
dnf clean all
```

Warewulf cannot directly import a container image from an Apptainer SIF yet, so an Apptainer image must be built as a *sandbox*.

```
# apptainer build --sandbox warewulf-rockylinux-9 warewulf-rockylinux-9.def
[...]
INFO:    Creating sandbox directory...
INFO:    Build complete: warewulf-rockylinux-9
```

Once a sandbox container image has been built, it can be imported into Warewulf.

```
# wwctl container import ./warewulf-rockylinux-9 rockylinux-9
```

Note

Although warewulf does not currently support importing a SIF directly, a SIF can be converted to a sandbox with Apptainer and then imported into Warewulf.

```
# apptainer build --sandbox my-sandbox my-image.sif
# wwctl container import ./my-sandbox my-image
```

25.7 Duplicating an image

It is possible to duplicate an installed image by using:

```
# wwctl image copy IMAGE_NAME DUPLICATED_IMAGE_NAME
```

This kind of duplication can be useful if you are looking for canary tests.

Note

If an image source includes persistent sockets, these sockets may cause the copy operation to fail.

```
Copying sources...
ERROR  : could not duplicate image: lchown /var/lib/warewulf/chroots/rocky-8/rootfs/run/user/0/
↪gnupg/d.kg8ijih5tq41ixoeag4p1qup/S.gpg-agent: no such file or directory
```

To resolve this, remove the sockets from the image source.

```
find $(wwctl image show rocky-8) -type s -delete
```

25.8 Image Architecture

By default, Warewulf will try to import an image of the same platform (e.g., amd64, arm64) as the local system. To specify the platform to import, either specify WAREWULF_OCI_PLATFORM or use the argument --platform during import.

It is possible to build, edit, and provision images of different architectures (i.e. aarch64) from an x86_64 host by using QEMU. Simply run the appropriate command below based on your image management tools.

```
# docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
# podman run --rm --privileged multiarch/qemu-user-static --reset -p yes
# apptainer run docker://multiarch/qemu-user-static --reset -p yes
```

Then, wwctl image exec will work regardless of the architecture of the image. For more information about QEMU, see their GitHub

Note

When provisioning cluster nodes with a different architecture than the Warewulf server, also use the matching architecture-specific *wwclient* overlay: e.g., wwclient.x86_64 or wwclient.aarch64.

25.9 Read-only images

An image may be marked "read-only" by creating a readonly file in its source directory, typically next to rootfs.

Note

Read-only images are a preview feature primarily meant to enable future support for image subscriptions and updates.

CHAPTER

TWENTYSIX

IMAGE KERNELS

Warewulf nodes require a Linux kernel to boot. As of Warewulf v4.6, the kernel you wish to use must be present in the relevant image. Warewulf locates and provisions the kernel automatically for any node configured to use that image.

You can see what kernels are available in imported images by using the wwctl image kernels command.

```
# wwctl image kernels
Image                Kernel                                          Version          Default   Nodes
-----                ------                                          -------          -------   -----
newroot-test         /boot/vmlinuz-5.14.0-427.37.1.el9_4.aarch64     5.14.0-427.37.1  true      0
newroot-test         /lib/modules/5.14.0-427.37.1.el9_4.aarch64/vmlinuz  5.14.0-427.37.1  false     0
rocky-8              /boot/vmlinuz-4.18.0-372.13.1.el8_6.x86_64      4.18.0-372.13.1  true      2
rocky-8              /lib/modules/4.18.0-372.13.1.el8_6.x86_64/vmlinuz  4.18.0-372.13.1  false     0
rocky-9.3            /lib/modules/5.14.0-362.13.1.el9_3.aarch64/vmlinuz  5.14.0-362.13.1  true      0
rockylinux-9-custom  /lib/modules/5.14.0-427.40.1.el9_4.aarch64/vmlinuz  5.14.0-427.40.1  true      0
```

26.1 Kernel Version

If an image includes multiple kernels, the desired kernel may be selected by specifying the desired version or an explicit path.

```
# wwctl node set n1 --kernelversion=4.18.0-372.13.1
# wwctl node set n1 --kernelversion=/boot/vmlinuz-4.18.0-372.13.1.el8_6.x86_64
```

CHAPTER

TWENTYSEVEN

SYNCUSER

Warewulf's syncuser feature has two distinct parts that work together:

1. The **syncuser command** synchronizes UIDs and GIDs from the Warewulf server into an OS image, updating /etc/passwd, /etc/group, and file ownerships within the image.
2. The **syncuser overlay** merges users and groups from both the OS image and the Warewulf server into the provisioned node's /etc/passwd and /etc/group at runtime.

This is particularly useful when there is no central directory (e.g., an LDAP server). Together, these two parts ensure that UIDs and GIDs are consistent across the server and all cluster nodes.

Note

Some system services (notably "munge") require a user to have the same UID across all nodes.

27.1 Synchronizing UIDs/GIDs in an OS image

The syncuser command updates an OS image so that any user or group present on the Warewulf server has the same UID/GID inside the image. Users and groups that exist only in the image are preserved unless a UID/GID collision is detected. File ownerships within the image are also updated to match.

If there is a mismatch between the server and the image, the import command will generate a warning.

Syncuser may be invoked during image import, exec, shell, or build:

```
# wwctl image import --syncuser docker://ghcr.io/warewulf/warewulf-rockylinux:9 rockylinux-9
# wwctl image exec --syncuser rockylinux-9 -- /usr/bin/echo "Hello, world!"
# wwctl image shell --syncuser rockylinux-9
# wwctl image build --syncuser rockylinux-9
# wwctl image syncuser rockylinux-9
```

After syncuser, /etc/passwd and /etc/group in the image are updated, and permissions on files belonging to these UIDs and GIDs are updated to match.

27.2 The syncuser overlay

The syncuser overlay runs at provisioning time and merges /etc/passwd and /etc/group from both the OS image and the Warewulf server. This makes users defined on the server (but not originally in the image) available on provisioned nodes.

For the overlay to work correctly, the image should also have been prepared with the syncuser command (see above) so that UID/GID values are consistent.

See *syncuser* in the overlays documentation for more detail.

27.3 Node-specific local users and groups

Warewulf supports defining **node-specific local users and groups** through the *resources* section of node definitions. These user and group entries are merged into the syncuser overlay and included in /etc/passwd and /etc/group on the target nodes at provisioning time.

This is useful for applications that require a node-local account (e.g., database, application, or storage service users) without having to maintain centralized identity management.

Note

Users created through this method do not have passwords set. They are intended for service accounts and non-interactive use.

Example:

```
nodes:
  n1:
    resources:
      localgroups:
        - gid: 1002
          members:
            - dbuser
            - dbuserbackup
          name: dbgroup
      localusers:
        - gid: 1001
          home: /
          name: dbuser
          shell: /bin/nologin
          uid: 1001
        - gid: 1005
          home: /
          name: dbuserbackup
          shell: /bin/nologin
          uid: 1005
```

CHAPTER

TWENTYEIGHT

SELINUX-ENABLED IMAGES

Warewulf supports booting SELinux-enabled images, though nodes using SELinux must be configured to use tmpfs for their image file system. (The default initramfs root does not support extended file attributes, which are required for SELinux context labeling.)

```
wwctl profile set default --root tmpfs
```

Note

Versions of Warewulf prior to v4.5.8 also required a kernel argument "rootfstype=ramfs" in order for wwinit to copy the image to tmpfs; but this is no longer required.

Once that is done, enable SELinux in /etc/sysconfig/selinux and install the appropriate packages in the image. An example of such an image is available in the warewulf-node-images repository.

SELinux requires extended attributes, which aren't supported on a default initramfs root. Nodes using SELinux should specify --root=tmpfs.

CHAPTER

TWENTYNINE

OVERLAYS

Warewulf supplements provisioned OS images with an "overlay" system. Overlays are collections of files and *Templates* that are rendered and built per-node and then applied over the image during the provisioning process.

Overlays are the primary mechanism for adding functionality to Warewulf. Much of even core functionality in Warewulf is implemented as distribution overlays, and this flexibility is also available for local, custom overlays. By combining templates with tags, network tags, and resources, the node registry (nodes.conf) can become an expressive metadata store for arbitrary cluster node configuration.

You can list the available overlays with wwctl overlay list, and the files within the overlays with wwctl overlay list --all.

```
# wwctl overlay list --all fstab
OVERLAY NAME  FILES/DIRS    SITE
------------  ----------    ----
fstab         etc/          false
fstab         etc/fstab.ww  false
```

29.1 Overlay Variables

The command wwctl overlay info shows the variables used in an overlay template, along with the help text for each variable.

```
# wwctl overlay info NetworkManager etc/NetworkManager/system-connections/ww4-managed.ww
VARIABLE                      HELP                                          TYPE    OPTION
--------                      ----                                          ----    ------
$netdev.Device                Set the device for given network              string  --netdev
$netdev.Gateway               Set the node's network device gateway         IP      --gateway
$netdev.Hwaddr                Set the device's HW address for given network  string  --hwaddr
$netdev.Ipaddr                IPv4 address in given network                 IP      --ipaddr
$netdev.Ipaddr6               IPv4 address in given network                 IP      --ipaddr
$netdev.Ipaddr6               IPv6 address                                  IP      --ipaddr6
$netdev.MTU                   Set the mtu                                   string  --mtu
$netdev.OnBoot.BoolDefaultTrue  Enable/disable network device (true/false)  WWbool  --onboot
$netdev.Tags
$netdev.Tags.DNSSEARCH
$netdev.Tags.downdelay
$netdev.Tags.master
$netdev.Tags.miimon
$netdev.Tags.mode
$netdev.Tags.parent_device
```

(continues on next page)

(continued from previous page)

```
$netdev.Tags.updelay
$netdev.Tags.vlan_id
$netdev.Tags.xmit_hash_policy
$netdev.Type                    Set device type of given network            string  --type
```

29.1.1 Structure

An overlay is a directory that is applied to the root of a cluster node's runtime file system. The overlay source directory should contain a single rootfs directory which represents the actual root directory for the overlay.

```
/usr/share/warewulf/overlays/issue
└── rootfs
    └── etc
        └── issue.ww
```

29.1.2 Adding Overlays to Nodes

A node or profile can configure an overlay in two different ways:

- An overlay can be configured to apply only during boot, along with the node image. These overlays are called **system overlays**.
- An overlay can be configured to also apply periodically while the system is running. These overlays are called **runtime overlays**.

```
wwctl profile set default \
  --system-overlays="wwinit,wwclient,fstab,hostname,ssh.host_keys,systemd.netname,NetworkManager" \
  --runtime-overlays="hosts,ssh.authorized_keys"
```

Multiple overlays can be applied to a single node, and overlays from multiple profiles are appended together when applied to a single node.

29.1.3 Building Overlays

Overlays are built (e.g., with wwctl overlay build) into compressed overlay images for distribution to cluster nodes. These images typically match these two use cases: system and runtime. As such, each cluster node typically has two overlay images.

```
# wwctl overlay build
Building system overlay image for n1
Created image for n1 system overlay: /var/lib/warewulf/provision/overlays/n1/__SYSTEM__.img
Compressed image for n1 system overlay: /var/lib/warewulf/provision/overlays/n1/__SYSTEM__.img.
↪gz
Building runtime overlay image for n1
Created image for n1 runtime overlay: /var/lib/warewulf/provision/overlays/n1/__RUNTIME__.img
Compressed image for n1 runtime overlay: /var/lib/warewulf/provision/overlays/n1/__RUNTIME__.
↪img.gz
```

Overlay images for multiple nodes are built in parallel. By default, each CPU in the Warewulf server will build overlays independently. The number of workers can be specified with the --workers option.

Warewulf will attempt to build/update overlays as needed (configurable in the warewulf.conf); but not all cases are detected, and manual overlay builds are often necessary.

29.1.4 Creating and Modifying Overlays

You can add a new overlay to Warewulf with wwctl overlay create.

```
wwctl overlay create issue
```

A new overlay is just an empty directory. For it to be useful it needs to contain some files.

For example, wwctl overlay import imports files from the Warewulf server into the overlay.

```
wwctl overlay import --parents issue /etc/issue
```

This imports /etc/issue from the Warewulf server into the new issue overlay.

Note

The issue overlay already existed as a distribution overlay. Creating one shadows the distribution overlay with a new site overlay, allowing for local modification.

Any modification to a distribution overlay first transparently creates a new site overlay and applies any changes there: distribution overlays should always remain unmodified.

You can also edit a new or existing overlay file in an interactive editor.

```
wwctl overlay edit issue /etc/issue
```

Use wwctl overlay show to inspect the content of an overlay file.

```
wwctl overlay show issue /etc/issue
```

Overlay files that end with .ww are templates. You can use wwctl overlay show --render=<node> to show how a given template file would be rendered for distribution to a given cluster node.

```
wwctl overlay delete issue /etc/issue
wwctl overlay import issue /etc/issue /etc/issue.ww
wwctl overlay show issue /etc/issue.ww --render=n1
```

More information about templates is available in *its own section*.

The content of the file for the given overlay is displayed with this command. With the --render option a template is rendered as it will be rendered for the given node. The node name is a mandatory argument to the --render flag. Additional information for the file can be suppressed with the --quiet option.

Note

It is not possible to delete files with an overlay.

Warning

On many modern distributions, /bin, /sbin, /lib, and /lib64 are symlinks into /usr. An overlay that targets one of these directories replaces that symlink with a directory, which can break the node's filesystem in subtle ways (including two-stage Dracut boots failing at switch_root). Target the real /usr paths (e.g., /usr/lib64) instead. See *Overlays and Merged-/usr Symlinks* for details.

29.2 Permissions

Overlay files are distributed to cluster nodes with the same user, group, and mode that they have on the Warewulf server. Use wwctl overlay chown and wwctl overlay chmod to adjust them as necessary.

```
wwctl overlay chown issue /etc/issue.ww root root
wwctl overlay chmod issue /etc/issue.ww 0644
```

29.2.1 Distribution Overlays

Warewulf distinguishes between **distribution** overlays, which are included with Warewulf, and **site** overlays, which are created or added locally. A site overlay always takes precedence over a distribution overlay with the same name. Any modification of a distribution overlay with wwctl actually makes changes to an automatically-generated **site** overlay cloned from the distribution overlay.

Site overlays are often stored at /var/lib/warewulf/overlays/. Distribution overlays are often stored at /usr/share/warewulf/overlays/. But these paths are dependent on compilation, distribution, packaging, and configuration settings.

29.3 wwinit

The **wwinit** overlay performs initial configuration of the Warewulf node. Its *wwinit* script runs before systemd or other init is called and contains all configurations which are needed to boot.

In particular:

- Configure the loopback interface
- Configure the BMC based on the node's configuration
- Update PAM configuration to allow missing shadow entries
- Relabel the file system for SELinux

Other overlays can place scripts in one of two locations for additional pre-init provisioning actions:

- **/warewulf/wwinit.d/:** executed in the initial root final system before the image is loaded into its final location. In a two-stage boot, these scripts are executed in the Dracut initramfs.
- **/warewulf/init.d/:** executed in the final root file system but before calling init.

29.4 wwclient

All configured overlays are provisioned initially along with the OS image itself; but **wwclient** periodically fetches and applies the runtime overlay to allow configuration of some settings without a reboot.

wwclient contacts the ipaddr value from warewulf.conf by default. This can be overridden by specifying a WW_IPADDR environment variable, which can be set via an overlay in /etc/default/wwclient.

The default wwclient overlay contains a wwclient executable compiled for the same architecture as the Warewulf server. Architecture-specific wwclient.aarch64 and wwclient.x86_64 overlays are available as well. This supports using wwclient on cluster nodes with a different architecture than the Warewulf server.

29.5 Network interfaces

Warewulf ships with support for many different network interface configuration systems. All of these are applied by default; but the list may be trimmed to the desired system.

- ifcfg
- ifupdown
- NetworkManager
- wicked
- netplan

Note

The ifupdown overlay was previously named debian.interfaces. The old name is still supported for compatibility, but it is deprecated and will be removed in a future release.

Warewulf also configures both systemd and udev with the intended names of configured network interfaces, typically based on a known MAC address.

- systemd.netname
- udev.netname

Several of the network configuration overlays support netdev tags to further customize the interface:

- **DNS[0-9]*:** one or more DNS servers
- **DNSSEARCH:** domain search path
- **MASTER:** the master for a bond interface

The NetworkManager overlay supports additional netdev tags for advanced interface configuration:

- **parent_device:** the parent device of a vlan interface
- **vlan_id:** the vlan id for a vlan interface
- **downdelay, updelay, miimon, mode, xmit_hash_policy:** bond device settings
- **type:** a secondary interface type, typically used to specify the bond type (e.g., ethernet or infiniband)
- **mtu:** specify the mtu for the interface
- **cloned-mac-address:** set a cloned MAC address for the interface

29.6 Basics

The **hostname** overlay sets the hostname based on the configured Warewulf node name.

The **hosts** overlay configures /etc/hosts to include all Warewulf nodes.

The **issue** overlay configures a standard Warewulf status message for display during login.

The **resolv** overlay configures /etc/resolv.conf based on the value of "DNS" *nettags*. (In most situations this should be unnecessary, as the network interface configuration should handle this dynamically.)

```
wwctl node set n1 --nettagadd="DNS1=1.1.1.1"
wwctl node set n1 --nettagadd="DNS2=1.0.0.1"
```

29.7 fstab

The **fstab** overlay configures /etc/fstab based on the data provided in the "fstab" resource. It also creates entries for file systems defined by Ignition.

```
nodeprofiles:
  default:
    resources:
      fstab:
        - spec: warewulf:/home
          file: /home
          vfstype: nfs
        - spec: warewulf:/opt
          file: /opt
          vfstype: nfs
```

29.8 ssh

Two SSH overlays configure host keys (one set for all nodes in the cluster) and authorized_keys for the root account.

- ssh.authorized_keys
- ssh.host_keys

29.9 syncuser

The **syncuser** overlay updates /etc/passwd and /etc/group at provisioning time to include all users from both the Warewulf server and from the image.

To function properly, wwctl image syncuser (or the --syncuser option during import, exec, shell, or build) must also have been run on the image to synchronize its user and group IDs with those of the server. See *Syncuser* for details on both parts of the syncuser feature.

If a PasswordlessRoot tag is set to "true", the overlay will also insert a "passwordless" root entry. This can be particularly useful for accessing a cluster node when its network interface is not properly configured.

Warning

PasswordlessRoot is not recommended for production; it should only be used during debugging, when normal authentication is not functional.

29.10 ignition

The **ignition** overlay defines partitions and file systems on local disks. Configuration may be provided via native disk, partition, and filesystem fields or via an ignition resource.

```
ignition:
  storage:
    disks:
      - device: /dev/vda
        partitions:
          - label: scratch
```

(continues on next page)

(continued from previous page)

```
            shouldExist: true
            wipePartitionEntry: true
        wipeTable: true
    filesystems:
      - device: /dev/disk/by-partlabel/scratch
        format: btrfs
        path: /scratch
        wipeFilesystem: false
```

If any disk/partition/filesystem configuration is provided for a node with explicit arguments to wwctl <node|profile> set, the ignition resource is ignored.

To use ignition during Dracut (so that the root file system may be provisioned before the image is loaded) include Ignition in the Dracut image.

```
wwctl image exec rockylinux-9 -- /usr/bin/dracut --force --no-hostonly --add wwinit --add ignition --
↪regenerate-all
```

29.11 debug

The **debug** overlay is not intended to be used in configuration, but is provided as an example. In particular, the provided tstruct.md.ww demonstrates the use of most available template metadata.

```
wwctl overlay show --render=<nodename> debug tstruct.md.ww
```

29.12 localtime

The **localtime** overlay configures the timezone of a cluster node to match that of the Warewulf server; alternatively, a different timezone may be specified with a localtime tag.

```
wwctl profile set default --tagadd="localtime=UTC"
```

29.13 sfdisk

The **sfdisk** overlay partitions block devices during wwinit. Configuration may be provided using native disk and partition configuration or via an sfdisk resource.

Multiple devices can be partitioned, with each device provided as an item in sfdisk list.

```
sfdisk:
  - device: /dev/sda
    label: gpt
    partitions:
      - device: /dev/sda1
        name: sfdisk-rootfs
        size: 4194304
      - device: /dev/sda2
        name: sfdisk-scratch
        size: 1048576
      - device: /dev/sda3
```

(continues on next page)

(continued from previous page)

```
    name: sfdisk-swap
    size: 2097152
```

All headers and named partition fields supported by the sfdisk input format are supported in the sfdisk resource.

If any disk/partition configuration is provided for a node with explicit arguments to wwctl <node|profile> set, the sfdisk resource is ignored.

To use the sfdisk overlay, include sfdisk in the Dracut image. Optionally also include blockdev and/or udevadm, to allow the partition table to be re-scanned.

```
wwctl image exec rockylinux-8 -- /usr/bin/dracut --force --no-hostonly --add wwinit --install sfdisk --
↪install blockdev --install udevadm --regenerate-all
```

For more information, see the *Provision to disk* section.

29.14 mkfs

The **mkfs** overlay formats block devices during wwinit. Configuration may be provided using native filesystem fields or via an mkfs resource.

```
mkfs:
  - device: /dev/sda1 # the device to format
    type: xfs # what type of file system to create
    options: -b 1024 # additional options to pass to mkfs
    overwrite: false # whether to overwrite an existing format
    size: 0 # defaults to the full device
```

If any filesystem configuration is provided for a node with explicit arguments to wwctl <node|profile> set, the mkfs resource is ignored.

To use the mkfs overlay, include mkfs and any necessary file-system-specific sub-commands in the Dracut image. Optionally also include wipefs to detect existing file systems.

```
wwctl image exec rockylinux-9 -- /usr/bin/dracut --force --no-hostonly --add wwinit --install mkfs --
↪install mkfs.ext4 --install wipefs --regenerate-all
```

For more information, see the *Provision to disk* section.

29.15 mkswap

The **mkswap** overlay formats block devices during wwinit. Configuration may be provided using native filesystem fields or via a mkswap resource.

```
mkswap:
  - device: /dev/sda2 # the device to format
    overwrite: false # whether to overwrite an existing format
    label: swap # the label to set for the swap device
```

If any filesystem configuration is provided for a node with explicit arguments to wwctl <node|profile> set, the mkswap resource is ignored.

To use the mkswap overlay, include mkswap in the Dracut image. Optionally also include wipefs to detect existing file systems.

```
wwctl image exec rockylinux-9 -- /usr/bin/dracut --force --no-hostonly --add wwinit --install mkswap --
↪regenerate-all
```

For a complete walkthrough of configuring swap to free memory consumed by the in-memory image, including which root filesystem types support swap reclamation, see *Swap and image memory usage*.

29.16 systemd mounts

Two overlays, **systemd.mount** and **systemd.swap**, configure mounted and swap storage based on the configuration of native file system fields. They are often paired with the mkfs and mkswap overlays.

29.17 mig

The **mig** overlay configures NVIDIA MIG (Multi-Instance GPU) settings on nodes with supported NVIDIA GPUs. The **mig** overlay requires the NVIDIA driver and the nvidia-smi tool to be installed and functional on the node.

MIG configuration is based on the gpuMigProfiles tag, which specifies a comma-separated list of MIG profiles IDs or profile short names to configure.

For example, to configure a node with four MIG instances of profile 14 and one instance of profile 15 on all GPUs:

```
wwctl node set n1 --tagadd="gpuMigProfiles=14,14,14,15"
```

The MIG instances can also be defined by their profile short names:

```
wwctl node set n1 --tagadd="gpuMigProfiles=2g.20gb,2g.20gb,2g.20gb,1g.20gb"
```

If different GPUs on a node require different MIG configurations, the gpuMigProfiles tag can be specified as a colon-separated list of comma-separated lists. Commas separate profiles on a single GPU. Colons separate GPUs by index order. A value of 0 indicates no MIG partitioning and uses the full GPU.

For example, to configure a node that has four GPUs, where:

- GPU 0 has three instances of profile 14 and one instance of profile 15
- GPU 1 has two instances of profile 9
- GPU 2 and GPU 3 have one instance of profile 0 (no partitioning):

```
wwctl node set n1 --tagadd="gpuMigProfiles=14,14,14,15:9,9:0:0"
```

This can also be done using profile short names:

```
wwctl node set n1 --tagadd="gpuMigProfiles=2g.20gb,2g.20gb,2g.20gb,1g.20gb:3g.40gb,3g.40gb:0:0"
```

The exact MIG configuration available depends on the specific NVIDIA GPU model present in the node. To see the available MIG profiles for a given GPU model, run:

```
nvidia-smi mig -lgip
```

Refer to NVIDIA's documentation for more information on MIG profiles.

Once configured, the NVIDIA MIG instances can be mapped to Slurm GRES entries backed by /dev/nvidia-caps/capXX device files.

To generate the mapping of MIG instances to /dev/nvidia-caps/capXX device files for Slurm's gres.conf file, use the script provided by this overlay on the node:

```
/usr/bin/sh /usr/local/sbin/mig2gres
```

29.18 host

Configuration files used for the configuration of the Warewulf host / server are stored in the **host** overlay. Unlike other overlays, it *must* have the name host. Existing files on the host are copied to backup files with a wwbackup suffix at the first run. (Subsequent use of the host overlay won't overwrite existing wwbackup files.)

The following services get configuration files via the host overlay:

- ssh keys are created with the scripts ssh_setup.sh and ssh_setup.csh
- hosts entries are created by manipulating /etc/hosts with the template hosts.ww
- nfs kernel server receives its exports from the template exports.ww
- the dhcpd service is configured with dhcpd.conf.ww

CHAPTER

THIRTY

TEMPLATES

Templates (denoted in overlays with a .ww suffix) allow you to create dynamic configuration specifically for the node that it is applied to. Templates have access to all metadata from the node registry (nodes.conf) and much of the server configuration (warewulf.conf), and can also reference and import files from the server file system.

Warewulf uses the text/template engine to facilitate implementing dynamic content. This template format is documented at pkg.go.dev/text/template.

Note

When the template is rendered within a built overlay image, the .ww will be dropped, so /etc/hosts.ww will end up being /etc/hosts.

30.1 Non-Overlay Templates

Most Warewulf templates are included in overlays, but there are a few non-overlay templates as well.

- /etc/warewulf/ipxe/: includes iPXE script templates to direct iPXE during the network boot process.
- /etc/warewulf/grub/: includes GRUB script templates to direct GRUB during the network boot process.
- /usr/share/warewulf/bmc/: includes templates to generate BMC control commands for the wwctl power, wwctl sensor, and wwctl console commands.

30.2 Template documentation

Templates can include documentation to be included in the output of wwctl overlay info.

```
{{/* wwdoc: Your documentation text */}}
```

30.3 Template variables

Overlay templates have access to a number of variables that provide information about the server configuration, the node being provisioned, and all nodes in the cluster. An example of the variables available, and their use, is included with Warewulf in the tstruct.ww template of the debug overlay.

Variables used in an overlay template can be documented by adding a comment to the template with the form {{/* .My.Var: Your help text */}}. Variable help text defined in a comment replaces that variable's default help text in the output of wwctl overlay info.

30.4 Template functions

Warewulf templates have access to a number of functions that assist in creating more dynamic and expressive templates.

30.4.1 Default functions

text/template includes a number of default functions that are available during Warewulf template processing.

30.4.2 Sprig

Supplementing the default functions, Warewulf templates also have access to Sprig functions.

30.4.3 Include

Reads content from the given file into the template. If the file does not begin with / it is considered relative to Paths.Sysconfdir.

```
{{ Include "/root/.ssh/authorized_keys" }}
```

30.4.4 IncludeFrom

Reads content from the given file from the given image into the template.

```
{{ IncludeFrom $.ImageName "/etc/passwd" }}
```

30.4.5 IncludeBlock

Reads content from the given file into the template, stopping when the provided abort string is found.

```
{{ IncludeBlock "/etc/hosts" "# Do not edit after this line" }}
```

30.4.6 ImportLink

Causes the processed template file to become a symlink to the same target as the referenced symlink.

```
{{- ImportLink "/etc/localtime" -}}
```

When paired with file, ImportLink can create multiple symlinks from a single template.

```
{{- file "/tmp/test-link1" -}}
{{- softlink "/tmp/test-target1" -}}
{{- file "/tmp/test-link2" -}}
{{- softlink "/tmp/test-target2" -}}
```

30.4.7 basename

Returns the base name of the given path.

```
{{- range $type, $name := $.Tftp.IpxeBinaries }}
 if option architecture-type = {{ $type }} {
    filename "/warewulf/{{ basename $name }}";
 }
{{- end }}
```

30.4.8 file

Write the content from the template to the specified file name. May be specified more than once in a template to write content to multiple files.

When no file call is present, all template output is written to the **default output path**: the template filename with the .ww suffix stripped (e.g., /etc/hosts.ww → /etc/hosts).

When one or more file calls are present, each call redirects subsequent template output to that named file. Any content rendered before the first file call is not written to disk. A softlink or ImportLink placed before any file call still applies to the default output path and is written to disk even when named files are also present.

```
{{- range $devname, $netdev := .NetDevs }}
{{- $filename := print "ifcfg-" $devname ".conf" }}
{{- file $filename -}}
{{/* content here */}}
{{- end }}
```

30.4.9 softlink

Causes the processed template file to become a symlink to the referenced target.

```
{{- printf "%s/%s" "/usr/share/zoneinfo" .Tags.localtime | softlink -}}
```

When paired with file, softlink can create multiple symlinks from a single template.

30.4.10 readlink

Equivalent to filepath.EvalSymlinks. Returns the target path of a named symlink.

```
{{ readlink /etc/localtime }}
```

30.4.11 IgnitionJson

Generates JSON suitable for use by Ignition to create partitions and file systems, from the data stored in a node's native disks and filesystems fields.

```
{{ IgnitionJson }}
```

30.4.12 abort

Immediately aborts processing the template and does not write a file.

```
{{- abort -}}
```

30.4.13 nobackup

Disables the creation of a backup file when replacing files with the current template.

```
{{- nobackup -}}
```

30.4.14 UniqueField

Returns a filtered version of a multi-line input string. Input is expected to be a field-separated format with one record per line (terminated by \n). Order of lines is preserved, with the first matching line taking precedence.

For example, the following template snippet has been used in the syncuser overlay to generate a combined /etc/passwd.

```
{{
   printf "%s\n%s"
      (IncludeFrom $.ImageName "/etc/passwd" | trim)
      (Include (printf "%s/%s" .Paths.Sysconfdir "passwd") | trim)
   | UniqueField ":" 0 | trim
}}
```

30.4.15 SystemdEscape

Escapes a string for use in a systemd unit file.

Escape rules are documented at systemd.unit.

30.4.16 SystemdEscapePath

Escapes a path for use in a systemd unit file.

```
{{- file (print ($fs.path | SystemdEscapePath) ".mount") -}}
```

Escape rules are documented at systemd.unit.

30.5 Examples

Many example templates are included in the distribution overlays. The debug template also includes a tstruct.ww template that includes much of the available metadata.

```
wwctl overlay show debug tstruct.ww
wwctl overlay show debug tstruct.ww --render=n1
```

30.5.1 Node-Specific Files

Sometimes there is the need to have specific files for each cluster node which can't be generated by a template (e.g., a per-node Kerberos keytab). You can include these files with following template:

```
{{ Include (printf "/srv/%s/%s" .Id "payload") }}
```

CHAPTER

THIRTYONE

TROUBLESHOOTING

31.1 sos

The warewulf-sos package (new in v4.6.1) adds support for gathering Warewulf server configuration information in an sos report.

```
dnf -y install warewulf-sos
sos report # optionally, --enable-plugins=warewulf
```

Note

The warewulf-sos package is not currently built for SUSE.

31.2 warewulfd

The Warewulf server (warewulfd) sends logs to the systemd journal.

```
journalctl -u warewulfd.service
```

To increase the verbosity of the log, specify either --verbose or --debug in the warewulfd OPTIONS.

```
echo "OPTIONS=--debug" >>/etc/default/warewulfd
systemctl restart warewulfd.service
```

31.3 iPXE

If you're using iPXE to boot (the default), you can get a command prompt by pressing with C-b during boot.

From the iPXE command prompt, you can run the same commands from default.ipxe to troubleshoot potential boot problems.

For example, the following commands perform a (relatively) normal Warewulf boot. (Substitute your Warewulf server's IP address in place of 10.0.0.1, update the port number if you have changed it from the default of 9873, and substitute your cluster node's MAC address in place of 00:00:00:00:00:00.)

```
set base http://10.0.0.1:9873
set hwaddr 00:00:00:00:00:00
kernel --name kernel ${base}/kernel/${hwaddr}
imgextract --name image ${base}/image/${hwaddr}?compress=gz
```

(continues on next page)

(continued from previous page)

```
imgextract --name system ${base}/system/${hwaddr}?compress=gz
imgextract --name runtime ${base}/runtime/${hwaddr}?compress=gz
boot kernel initrd=image initrd=system initrd=runtime
```

- The base variable points to warewulfd for future reference. The MAC address is appended to each resource path so that Warewulf knows what image and overlays to provide.
- The kernel command fetches a kernel for later booting.
- The imgextract command fetches and decompresses the images that will make up the booted OS image. In a typical environment this is used to load a minimal "initial ramdisk" which, then, boots the rest of the system. Warewulf, by default, loads the entire image as an initial ramdisk, and also loads the system and runtime overlays at this time.
- The boot command tells iPXE to boot the system with the given kernel and ramdisks.

Note

This example does not provide assetkey information to warewulfd. If your nodes have defined asset tags, provide it in the uri variable for the node you are trying to boot.

For example, you may want to try booting to a pre-init shell with debug logging enabled. To do so, substitute the boot command above.

```
boot kernel initrd=image initrd=system initrd=runtime rdinit=/bin/sh
```

Note

You may be more familiar with specifying init= on the kernel command line. rdinit indicates "ramdisk init." Since Warewulf, by default, boots the OS image as an initial ramdisk, we must use rdinit= here.

31.4 GRUB

If you're using GRUB to boot, you can get a command prompt by pressing "c" when prompted during boot.

From the GRUB command prompt, you can enter the same commands that you would otherwise find in grub.cfg.ww.

For example, the following commands perform a (relatively) normal Warewulf boot. (Substitute your Warewulf server's IP address in place of 10.0.0.1, and update the port number if you have changed it from the default of 9873.)

```
base="(http,10.0.0.1:9873)"
linux "${base}/kernel/${net_default_mac}" wwid=${net_default_mac}
initrd "${base}/image/${net_default_mac}?compress=gz" "${base}/system/${net_default_mac}?
↪compress=gz" "${base}/runtime/${net_default_mac}?compress=gz"
boot
```

- The base variable points to warewulfd for future reference. ${net_default_mac} provides Warewulf with the MAC address of the booting node, so that Warewulf knows what image and overlays to provide it.
- The linux command tells GRUB what kernel to boot, as provided by warewulfd. The wwid kernel argument helps wwclient identify the node during runtime.

- The initrd command tells GRUB what images to load into memory for boot. In a typical environment this is used to load a minimal "initial ramdisk" which, then, boots the rest of the system. Warewulf, by default, loads the entire image as an initial ramdisk, and also loads the system and runtime overlays at this time.
- The boot command tells GRUB to boot the system with the previously-defined configuration.

Note

This example does not provide assetkey information to warewulfd. If your nodes have defined asset tags, provide it in the uri variable for the node you are trying to boot.

For example, you may want to try booting to a pre-init shell with debug logging enabled. To do so, substitute the linux command above.

```
linux "${base}/kernel/${net_default_mac}" wwid=${net_default_mac} debug rdinit=/bin/sh
```

Note

You may be more familiar with specifying init= on the kernel command line. rdinit indicates "ramdisk init." Since Warewulf, by default, boots the OS image as an initial ramdisk, we must use rdinit= here.

31.5 Dracut

By default, dracut simply panics and terminates when it encounters an issue.

Dracut looks at the kernel command line for its configuration. You can configure it for additional logging and to switch to an interactive shell on error:

```
wwctl profile set default --kernelargs=rd.shell,rd.debug,log_buf_len=1M
```

For more information on debugging Dracut problems, see the Fedora dracut problems guide.

31.6 Ignition

If partition creation doesn't work as expected you have a few options to investigate:

- Add systemd.log_level=debug and or rd.debug to the kernelArgs of the node you're working on.
- After the next boot you should be able to find verbose information on the node with journalctl -u ignition-ww4-disks.service.
- You could also check the content of /warewulf/ignition.json.
- You could try to tinker with /warewulf/ignition.json calling

  ```
  /usr/lib/dracut/modules.d/30ignition/ignition \
    --platform=metal \
    --stage=disks \
    --config-cache=/warewulf/ignition.json \
    --log-to-stdout
  ```

  after each iteration on the node directly until you find the settings you need. (Make sure to unmount all partitions if ignition was partially successful.)

- Sometimes you need to add should_exist: "true" for the swap partition as well.

31.7 Overlay Shadowing

When Warewulf introduced the distinction between distribution overlays and site overlays, existing installations that had modified any distribution overlays were left with those modified files in the site overlay directory (typically /var/lib/warewulf/overlays/). Because a site overlay takes complete precedence over a distribution overlay with the same name — with no merging of individual files — the entire distribution overlay is shadowed. Any new files or updates added to the distribution overlay in a subsequent Warewulf upgrade will be hidden as long as a site overlay of the same name exists.

To check whether any distribution overlays are being shadowed by site overlays, use wwctl overlay list, which includes a SITE column:

```
wwctl overlay list
```

Any overlay showing true in the SITE column that you did not intentionally create locally may be unintentionally shadowing its distribution counterpart.

To see which files are present in a site overlay, use the --all flag:

```
wwctl overlay list --all <overlay_name>
```

To see the filesystem paths of the overlays directly, use the --path flag:

```
wwctl overlay list --path
```

If you determine that a site overlay is unintentionally shadowing a distribution overlay, you can restore the distribution overlay by deleting the site overlay. Back up any intentional local modifications first, then delete the site overlay:

```
wwctl overlay delete <overlay_name>
```

wwctl overlay delete only ever deletes site overlays, so this command is safe to run without risk of removing the underlying distribution overlay. After deleting the site overlay, wwctl overlay list should show false in the SITE column for that overlay, confirming that the distribution overlay is now active.

31.8 Overlays and Merged-/usr Symlinks

On many modern Linux distributions, several top-level directories are not real directories but symlinks into /usr:

- /bin → /usr/bin
- /sbin → /usr/sbin
- /lib → /usr/lib
- /lib64 → /usr/lib64

This is known as the "merged /usr" layout. Because Warewulf overlays are applied by walking the overlay's rootfs and creating each directory as a *real* directory on the node, placing files under any of these paths in an overlay replaces the distribution's symlink with a directory. This diverges the node's filesystem from what the image expects and can cause difficult-to-debug failures — most notably, a two-stage (Dracut) boot that fails at the switch_root step, seemingly out of nowhere.

For example, an overlay that adds a compatibility symlink under /lib64:

```
my-overlay
└── rootfs
    └── lib64
        └── libexample.so.1 -> libexample.so.1.2.3
```

will shadow the /lib64 → /usr/lib64 symlink with a directory containing only libexample.so.1, hiding the rest of the system's libraries.

Always target the real ``/usr`` path instead. Rework the overlay to use /usr/lib64 (and likewise /usr/bin, /usr/sbin, and /usr/lib):

```
my-overlay
└── rootfs
    └── usr
        └── lib64
            └── libexample.so.1 -> libexample.so.1.2.3
```

The /lib64 → /usr/lib64 symlink from the image then continues to resolve correctly, and your file appears at both /lib64/libexample.so.1 and /usr/lib64/libexample.so.1.

31.9 Running Containers on Cluster Nodes

Container runtimes such as Podman require filesystem features — most notably OverlayFS support for image storage and container layers — that are not available with the default initramfs root filesystem. To run Podman or similar runtimes on cluster nodes, configure the node or profile to use tmpfs as the root filesystem:

```
# Apply to all nodes via a profile
wwctl profile set default --root=tmpfs

# Or apply to a specific node
wwctl node set <nodename> --root=tmpfs
```

After changing the root filesystem type, reboot the affected nodes to apply the new configuration.

> **Note**
>
> The OS image itself must have Podman (or the desired container runtime) installed. See *OS Images* for guidance on customizing OS images.

For information on tuning tmpfs memory usage and NUMA interleaving behavior, see *tmpfs and NUMA* below.

31.10 tmpfs and NUMA

Warewulf can optionally mount the root filesystem as tmpfs instead of the default initramfs. Warewulf will add mpol=interleave to the mount point which will distribute the memory across all NUMA nodes. This avoids the hotspotting that occurs when the default initramfs stores large OS images on a single NUMA node. To enable this, set the rootfs type to tmpfs:

```
wwctl profile set default --root=tmpfs
```

You may also adjust the tmpfs size via the wwinit.tmpfs.size kernel argument:

```
# Set tmpfs to use maximum 1GB
wwctl profile set default --kernelargs="wwinit.tmpfs.size=1G"
# You can also use a percentage of physical RAM
wwctl profile set default --kernelargs="wwinit.tmpfs.size=25%"
```

By default this is set to 50% of physical RAM. Note that tmpfs is required for SELinux overlays since initramfs cannot preserve SELinux contexts.

Because the root is tmpfs, the kernel can also swap cold image pages to a local swap device, freeing RAM for running workloads. This does not apply to the default initramfs root (single-stage boot), where pages are pinned in memory and cannot be swapped. See *Swap and image memory usage* for a complete walkthrough.

Note

On some systems, it may also be necessary to include the noefi kernel argument. This works around specific EFI firmware bugs that can prevent proper memory release during the transition from initramfs to tmpfs.

31.11 OCI Blob Cache

Warewulf caches OCI image layers on disk to speed up repeated wwctl image import operations. The cache can grow large when many images — or many versions of the same image — are imported over time.

v4.6 and later

The cache is stored at $cachedir/warewulf (default: /var/cache/warewulf on RPM-based distributions). It contains files and directories such as blobs/, index.json, and oci-layout.

Use wwctl clean to remove the cache:

```
# wwctl clean
```

The cache is rebuilt automatically on the next wwctl image import.

v4.5.x and earlier (legacy cache location)

In v4.5.x and earlier releases, the OCI blob cache was stored at $datastore/oci (default: /usr/share/oci in the Rocky Linux RPM packages). This location is not removed by wwctl clean.

If you are upgrading from v4.5.x and want to reclaim the space used by the old cache, you can safely delete this directory manually:

```
# rm -rf /usr/share/oci
```

Adjust the path if your installation used a non-default datastore setting in warewulf.conf.

CHAPTER

THIRTYTWO

KNOWN ISSUES

32.1 SELinux and IPMI Write not Working When Using Two-Stage Boot

The dracut implementation of two-stage boot in versions of Warewulf prior to v4.6.0 bypasses the wwinit process by default, invoking the image's init system directly. While cluster nodes will often still boot mostly successfully this way, features implemented by wwinit will not complete. In particular, SELinux relabeling and IPMI write are not executed.

To ensure that dracut runs the full wwinit process, pass init=/init or init=/warewulf/wwinit on the kernel command line.

```
wwctl profile set default --kernelargs="init=/init"
```

32.2 Images are Read-Only

Warewulf v4.5 uses the permissions on an image's rootfs/ to determine a "read-only" state of the image: if the root directory of the image is u-w, it will be mounted read-only during wwctl image <exec|shell, preventing interactive changes to the image.

In the past, the root directory was u+w, but Enterprise Linux 9.5 (including Red Hat, Rocky, _et al._) includes an update to the filesystem package that marks the root directory u-w. This causes Warewulf images to be "read only" by default.

To mark a Warewulf image as writeable, use *chmod u+w*.

```
chmod u+w $(wwctl image show rockylinux-9.5)
```

This behavior is changed in v4.6 to use an explicit readonly file stored outside of rootfs/.

32.3 Image Sockets Cause Build Failures

If an image source directory includes persistent sockets, these sockets may cause the import operation to fail.

```
Copying sources...
ERROR  : could not import image: lchown ./rockylinux-8/run/user/0/gnupg/d.kg8ijih5tq41ixoeag4p1qup/
↪S.gpg-agent: no such file or directory
```

To resolve this, remove the sockets from the source directory.

```
find ./rockylinux-8/ -type s -delete
```

This issue was fixed in an upstream library and should be resolved in Warewulf v4.6.0.

32.4 Image Size Considerations

OS images can grow quickly as packages and other files are added to them. Even these larger images are often not an issue in modern environments; but some architectural limits exist that can impede the use of images larger than a few gigabytes. Workarounds exist for these issues in most circumstances:

- Warewulf's *two-stage boot support* effectively eliminates this problem by handling the bulk of the image management within Linux. This feature is currently in preview, and is subject to change; but it is likely to become the default boot method in a future release.
- Systems booting in legacy / BIOS mode, being a 32-bit environment, cannot boot an image that requires more than 4GB to decompress. This means that the compressed image and the decompressed image together must be < 4GB. This is typically reported by the system as "No space left on device (https://ipxe.org/34182006)."

  The best work-around for this limitation is to switch to UEFI. UEFI is 64-bit and should support booting significantly larger images, though sometimes system-specific implementation details have led to artificial limitations on image size.
- The Linux kernel itself can only decompress an image up to 4GB due to the use of 32-bit integers in critical sections of the kernel initrd decompression code.

  The best work-around for this limitation is to use an iPXE with support for imgextract. This allows iPXE to decompress the image rather than the kernel.
- Some BIOS / firmware retain a "memory hole" feature for legacy devices, e.g., reserving a 1MB block of memory at the 15MB-16MB address range. This feature can interfere with booting stateless OS images.

  If you are still getting "Not enough memory" or "No space left on device" errors, try disabling any "memory hole" features or updating your system BIOS or firmware.

32.5 ARP Cache Overflow on Large Clusters

On clusters with many nodes, the Linux kernel's default ARP cache limits may be too low, causing the Warewulf server to log:

```
neighbour: arp_cache: neighbor table overflow!
```

The kernel manages the ARP cache with three garbage-collection thresholds:

- net.ipv4.neigh.default.gc_thresh1 – no garbage collection below this count (default: 128)
- net.ipv4.neigh.default.gc_thresh2 – garbage collection is triggered above this count (default: 512)
- net.ipv4.neigh.default.gc_thresh3 – hard limit; entries are dropped above this count (default: 1024)

This is particularly relevant for Warewulf because warewulfd identifies nodes by looking up their IP address in the kernel's ARP cache (/proc/net/arp) during provisioning. If cache entries for cluster nodes are evicted, node identification can fail.

Increase the thresholds on the Warewulf server. As a starting point:

```
sysctl -w net.ipv4.neigh.default.gc_thresh2=1024
sysctl -w net.ipv4.neigh.default.gc_thresh3=2048
```

For larger clusters (hundreds of nodes or more), higher values may be needed:

```
sysctl -w net.ipv4.neigh.default.gc_thresh2=2048
sysctl -w net.ipv4.neigh.default.gc_thresh3=4096
```

To make the change persistent across reboots, create a file in /etc/sysctl.d/:

```
cat > /etc/sysctl.d/90-warewulf-arp.conf << 'EOF'
net.ipv4.neigh.default.gc_thresh2 = 2048
net.ipv4.neigh.default.gc_thresh3 = 4096
EOF
sysctl -p /etc/sysctl.d/90-warewulf-arp.conf
```

CHAPTER

THIRTYTHREE

CONTRIBUTING

Warewulf is an open source project, and we are grateful for any support or contributions. Helping other users, raising issues, writing documentation, and contributing code are all ways to help!

33.1 Join the community

Whether you develop Warewulf or use it to deploy clusters, we hope you'll spread the word! Share your experiences online. Ask your distribution to include support for Warewulf. Consider giving a talk at a conference or meetup!

33.1.1 Warewulf on Slack

Many members of the Warewulf community, including its developers, communicate via Slack. It's a great place to get help with an issue or talk about your deployment.

An invite link is available at *https://warewulf.org/help/ <https://warewulf.org/help/>*.

33.1.2 OpenHPC

OpenHPC includes Warewulf v4 (and Warewulf 3 before it) as a supported cluster management system and deployment strategy. Participating in the OpenHPC community is also a great way to support Warewulf!

33.2 Raise an Issue

For general bugs/issues, you can open an issue at the GitHub repo.

33.3 Contribute to the Code

We use the traditional GitHub Flow to develop. This means that you fork the main repo, create a new branch to make changes, and submit a pull request (PR) to the main branch.

Check out our official CONTRIBUTING.md document.

CHAPTER

THIRTYFOUR

DEVELOPMENT ENVIRONMENT

To develop and test the Warewulf server, you need a single system (typically a virtual machine) to serve as a test server deployment. To actually test provisioning, your development server also needs a dedicated network that it can run DHCP on. This can typically be provisioned as a virtual network bridge in virtual machine software.

Options include:

- KVM / Libvirt
- VirtualBox
- VMWare
- UTM

A Warewulf development environment should likely use Rocky Linux 9 or openSUSE LEAP 15, though there are ongoing development efforts using Debian and Ubuntu as well.

34.1 Compiling Warewulf for a Development Server

```
# Rocky Linux 9
dnf -y install git epel-release golang {libassuan,gpgme}-devel unzip tftp-server dhcp-server nfs-utils ipxe-
↪bootimgs-{x86,aarch64}

git clone https://github.com/warewulf/warewulf.git
cd warewulf
env \
  PREFIX=/opt/warewulf \
  SYSCONFDIR=/etc \
  IPXESOURCE=/usr/share/ipxe \
  WWPROVISIONDIR=/opt/warewulf/provision \
  WWOVERLAYDIR=/opt/warewulf/overlays \
  WWCHROOTDIR=/opt/warewulf/chroots \
  make all
make install
```

These paths balance isolation (e.g., installing binaries in /opt/warewulf/bin/) with integration (e.g., storing configuration in /etc/warewulf/ and using local Dracut and iPXE paths).

After making changes to the source, simply running make install should be enough to update installed binaries.

You should likely also disable any local firewall. Otherwise, consult the general installation guide for configuration details.

```
systemctl disable --now firewalld
```

34.2 Running the Test Suite

Warewulf includes an ever-growing test suite. Alias targets in the Makefile support running it quickly, easily, and consistently.

```
make test
```

Additional tests exist as well to perform various checks on the golang source. These checks are run automatically by GitHub as part of the Warewulf CI process; but it is a good idea to run them locally before submitting a new PR.

```
make vet
make staticcheck
make lint
```

New code, and code changes, should often be accompanied by updates to the test suite.

More information:

- The golang testing package
- Table Driven Tests
- Testify assert
- Warewulf testenv

34.3 Using a Dev Container

Visual Studio Code (VSC) can utilize a Dev Container for a self-contained environment that has all the necessary tools and dependencies to build and test Warewulf. The Dev Container is based on the Rocky 9 image and is built using the devcontainer.json file in the .devcontainer directory of the Warewulf repository. To use this, working Docker/Podman and VSC installations are required. To use the Dev Container, click the "Open a Remote Window" button on the bottom left of the editor (>< icon) and select "Reopen in Container". This will build the container and open a new VSC window with the container as the development environment.

34.4 Using Vagrant and Libvirt

Vagrant can be used to quickly spin up a test/development environment for Warewulf. A Vagrantfile is provided in the vagrant directory of the Warewulf repository. See the README.md for more details.

CHAPTER

THIRTYFIVE

DOCUMENTATION

You can contribute to the documentation by raising an issue to suggest an improvement or by sending a pull request on our GitHub repository.

The current documentation is generated with Sphinx.

For more information on using Git and GitHub to create a pull request suggesting additions and edits to the docs, see the section on contributing to the code. The procedure is identical for contributions to documentation and code.

CHAPTER

THIRTYSIX

DEBUGGING

Whether developing a new feature or fixing a bug, using the automated test suite together with a debugger is a potent combination. This guide here can't substitute for full documentation on a given debugger; but it might help you get started debugging Warewulf.

36.1 Validating the code with vet

The Warewulf Makefile includes a vet target which runs go vet on the full codebase.

```
make vet
```

36.2 Running the Full Test Suite

The Warewulf Makefile includes a test target which runs the full test suite.

```
make test
```

Individual test cases are particularly useful when coupled with a debugger. For example, you can install delve as a regular user directly with Go.

```
$ go install github.com/go-delve/delve/cmd/dlv@latest
```

Visual Studio Code also includes a full-featured golang debugger that includes testsuite integration.

CHAPTER

THIRTYSEVEN

WWCTL

Warewulf Control

37.1 Synopsis

Control interface to the Warewulf Cluster Provisioning System.

37.2 Options

```
-d, --debug                 Run with debugging messages enabled.
-h, --help                  help for wwctl
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

37.3 SEE ALSO

- *wwctl clean* - Clean up
- *wwctl configure* - Manage system services
- *wwctl image* - Operating system image management
- *wwctl node* - Node management
- *wwctl overlay* - Warewulf Overlay Management
- *wwctl power* - Warewulf node power management
- *wwctl profile* - Node configuration profile management
- *wwctl server* - Start Warewulf server
- *wwctl ssh* - SSH into configured nodes in parallel
- *wwctl upgrade* - Upgrade configuration files
- *wwctl version* - Version information

CHAPTER

THIRTYEIGHT

WWCTL CLEAN

Clean up

38.1 Synopsis

This command cleans the OCI cache and removes leftovers from deleted nodes

```
wwctl clean
```

38.2 Options

```
-h, --help   help for clean
```

38.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

38.4 SEE ALSO

- *wwctl* - Warewulf Control

CHAPTER

THIRTYNINE

WWCTL CONFIGURE

Manage system services

39.1 Synopsis

This application allows you to manage and initialize Warewulf dependent system services based on the configuration in the warewulf.conf file.

```
wwctl configure [OPTIONS]
```

39.2 Options

```
-a, --all     Configure all services
-h, --help    help for configure
```

39.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

39.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl configure dhcp* - Manage and initialize DHCP
- *wwctl configure hostfile* - update hostfile on master
- *wwctl configure nfs* - Manage and initialize NFS
- *wwctl configure ssh* - Manage and initialize SSH
- *wwctl configure tftp* - Manage and initialize TFTP
- *wwctl configure tls* - Manage and initialize x509 keys
- *wwctl configure warewulfd* - Enable and start warewulfd

CHAPTER

FORTY

WWCTL CONFIGURE DHCP

Manage and initialize DHCP

40.1 Synopsis

DHCP is a dependent service to Warewulf. This command will configure DHCP as defined in the warewulf.conf file.

```
wwctl configure dhcp [OPTIONS]
```

40.2 Options

```
-h, --help   help for dhcp
```

40.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

40.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYONE

WWCTL CONFIGURE HOSTFILE

update hostfile on master

41.1 Synopsis

Manage the hostfile on the master node

```
wwctl configure hostfile [OPTIONS]
```

41.2 Options

```
-h, --help    help for hostfile
```

41.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

41.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYTWO

WWCTL CONFIGURE NFS

Manage and initialize NFS

42.1 Synopsis

NFS is an optional dependent service of Warewulf, this tool will automatically configure NFS as per the configuration in the warewulf.conf file.

```
wwctl configure nfs [OPTIONS]
```

42.2 Options

```
-h, --help    help for nfs
```

42.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

42.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYTHREE

WWCTL CONFIGURE SSH

Manage and initialize SSH

43.1 Synopsis

SSH is an optionally dependent service for Warewulf, this tool will automatically setup the ssh keys nodes using the 'default' system overlay as well as user owned keys.

```
wwctl configure ssh [OPTIONS]
```

43.2 Options

```
-h, --help                  help for ssh
-t, --keytypes stringArray  ssh key types to be created
```

43.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

43.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYFOUR

WWCTL CONFIGURE TFTP

Manage and initialize TFTP

44.1 Synopsis

TFTP is a dependent service of Warewulf, this tool will enable the tftp services on your Warewulf master.

```
wwctl configure tftp [OPTIONS]
```

44.2 Options

```
-h, --help    help for tftp
-s, --show    Show configuration (don't update)
```

44.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

44.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYFIVE

WWCTL CONFIGURE TLS

Manage and initialize x509 keys

45.1 Synopsis

This application allows you to manage the x509 keys and certificates for Warewulf.

```
wwctl configure tls [OPTIONS]
```

45.2 Options

```
    --export string   Export keys to directory
-f, --force           Enforce creation of keys even if they exist
-h, --help            help for tls
    --import string   Import keys from directory
```

45.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

45.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYSIX

WWCTL CONFIGURE WAREWULFD

Enable and start warewulfd

46.1 Synopsis

Enable and starts the warewulfd service.

```
wwctl configure warewulfd [OPTIONS]
```

46.2 Options

```
-h, --help   help for warewulfd
```

46.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

46.4 SEE ALSO

- *wwctl configure* - Manage system services

CHAPTER

FORTYSEVEN

WWCTL IMAGE

Operating system image management

47.1 Synopsis

Import, manage, and transform OS images as bootable Warewulf images.

47.2 Options

```
-h, --help   help for image
```

47.3 Options inherited from parent commands

```
-d, --debug               Run with debugging messages enabled.
-v, --verbose             Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

47.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl image build* - (Re)build a bootable image
- *wwctl image copy* - Copy an existing image
- *wwctl image delete* - Delete an imported image
- *wwctl image exec* - Run a command inside of a Warewulf image
- *wwctl image import* - Import an image into Warewulf
- *wwctl image kernels* - List available image kernels
- *wwctl image list* - List imported Warewulf images
- *wwctl image rename* - Rename an existing image
- *wwctl image shell* - Run a shell inside of a Warewulf image
- *wwctl image show* - Show root fs dir for image
- *wwctl image syncuser* - Synchronize UIDs/GIDs from the server to an OS image

CHAPTER

FORTYEIGHT

WWCTL IMAGE BUILD

(Re)build a bootable image

48.1 Synopsis

This command will build a bootable image from an imported IMAGE(s).

```
wwctl image build [OPTIONS] IMAGE [...]
```

48.2 Options

```
-a, --all          (re)Build all images
-f, --force        Force rebuild, even if it isn't necessary
-h, --help         help for build
    --syncuser     Synchronize UIDs/GIDs from host to image
```

48.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

48.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FORTYNINE

WWCTL IMAGE COPY

Copy an existing image

49.1 Synopsis

This command will duplicate an imported image.

```
wwctl image copy IMAGE NEW_NAME
```

49.2 Options

```
-b, --build   Build image after copy
-h, --help    help for copy
```

49.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

49.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTY

WWCTL IMAGE DELETE

Delete an imported image

50.1 Synopsis

This command will delete IMAGEs that have been imported into Warewulf.

```
wwctl image delete [OPTIONS] IMAGE [...]
```

50.2 Options

```
-h, --help   help for delete
-y, --yes    Set 'yes' to all questions asked
```

50.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

50.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYONE

WWCTL IMAGE EXEC

Run a command inside of a Warewulf image

51.1 Synopsis

Run a COMMAND inside of a warewulf IMAGE. This is commonly used with an interactive shell such as /bin/bash to run a virtual environment within the image.

```
wwctl image exec [OPTIONS] IMAGE COMMAND
```

51.2 Options

```
-b, --bind stringArray    source[:destination[:{ro|copy}]]
                          Bind a local path which must exist into the image. If destination is not
                          set, uses the same path as source. "ro" binds read-only. "copy" temporarily
                          copies the file into the image.
    --build               (Re)build the image automatically (default true)
-h, --help                help for exec
-n, --node string         Create a read only view of the image for the given node
    --syncuser            Synchronize UIDs/GIDs from host to image
```

51.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

51.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYTWO

WWCTL IMAGE IMPORT

Import an image into Warewulf

52.1 Synopsis

This command will pull and import an image into Warewulf from SOURCE, optionally renaming it to NAME. The SOURCE must be in a supported URI format. Formats are:

- docker://registry.example.org/example:latest
- docker-daemon://example:latest
- file://path/to/archive/tar/ball
- /path/to/archive/tar/ball
- /path/to/chroot/

Imported images are used to create bootable images.

```
wwctl image import [OPTIONS] SOURCE [NAME]
```

52.2 Examples

```
wwctl image import docker://ghcr.io/warewulf/warewulf-rockylinux:8 rockylinux-8
```

52.3 Options

```
-b, --build             Build image after pulling
-f, --force             Remove existing image and import new image with that name
-h, --help              help for import
    --nohttps           Ignore wrong TLS certificates, supersedes env WAREWULF_OCI_NOHTTPS
    --password string   Set password for the access to the registry, supersedes env WAREWULF_OCI_
↪PASSWORD
    --platform string   Set other hardware platform e.g. amd64 or arm64, supersedes env WAREWULF_
↪OCI_PLATFORM
    --syncuser          Synchronize UIDs/GIDs from host to image
-u, --update            Overwrite files in an existing image with the files of remote image
    --username string   Set username for the access to the registry, supersedes env WAREWULF_OCI_
↪USERNAME
```

52.4 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

52.5 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYTHREE

WWCTL IMAGE KERNELS

List available image kernels

53.1 Synopsis

This command lists the kernels that are available in the imported images.

```
wwctl image kernels [OPTIONS]
```

53.2 Options

```
-h, --help    help for kernels
```

53.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

53.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYFOUR

WWCTL IMAGE LIST

List imported Warewulf images

54.1 Synopsis

This command will show you the images that are imported into Warewulf.

```
wwctl image list [OPTIONS]
```

54.2 Options

```
-c, --chroot        show size of chroot
    --compressed    show size of the compressed image
-h, --help          help for list
-k, --kernel        show kernel version
-l, --long          show all
-s, --size          show size information
```

54.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

54.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYFIVE

WWCTL IMAGE RENAME

Rename an existing image

55.1 Synopsis

This command will rename an existing image.

```
wwctl image rename IMAGE NEW_NAME
```

55.2 Options

```
-b, --build   Build image after rename
-h, --help    help for rename
```

55.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

55.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYSIX

WWCTL IMAGE SHELL

Run a shell inside of a Warewulf image

56.1 Synopsis

Run a interactive shell inside of a warewulf IMAGE.

```
wwctl image shell [OPTIONS] IMAGE
```

56.2 Options

```
-b, --bind stringArray    source[:destination[:{ro|copy}]]
                          Bind a local path which must exist into the image.
                          If destination is not set, uses the same path as
                          source. "ro" binds read-only. "copy" temporarily
                          copies the file into the image.
    --build               (Re)build the image automatically (default true)
-h, --help                help for shell
-n, --node string         Create a read only view of the image for the given
                          node
    --syncuser            Synchronize UIDs/GIDs from host to image
```

56.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

56.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYSEVEN

WWCTL IMAGE SHOW

Show root fs dir for image

57.1 Synopsis

Shows the base directory for the chroot of the given image. More information about the image can be shown with the '-a' option.

```
wwctl image show [OPTIONS] IMAGE
```

57.2 Options

```
-a, --all     Show all information about an image
-h, --help    help for show
```

57.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

57.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYEIGHT

WWCTL IMAGE SYNCUSER

Synchronize UIDs/GIDs from the server to an OS image

58.1 Synopsis

Synchronize UIDs and GIDs from the server into an OS image, updating /etc/passwd, /etc/group, and file ownerships within the image. Users and groups that exist only in the image are preserved unless a UID/GID collision is detected.

This command affects the image itself (a one-time operation at build/import time). To also make host users available on provisioned nodes at runtime, add the "syncuser" overlay to the node or profile.

```
wwctl image syncuser [OPTIONS] IMAGE
```

58.2 Options

```
    --build   Build OS image after syncuser is completed
-h, --help    help for syncuser
    --write   Synchronize UIDs/GIDs and write files in OS image (default true)
```

58.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

58.4 SEE ALSO

- *wwctl image* - Operating system image management

CHAPTER

FIFTYNINE

WWCTL NODE

Node management

59.1 Synopsis

Management of node settings. All node ranges can use brackets to identify node ranges. For example: n00[00-4].cluster[0-1] will identify the first 5 nodes in cluster0 and cluster1.

59.2 Options

```
-h, --help   help for node
```

59.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

59.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl node add* - Add new node to Warewulf
- *wwctl node console* - Connect to IPMI console
- *wwctl node delete* - Delete a node from Warewulf
- *wwctl node edit* - Edit node(s) with editor
- *wwctl node export* - Export nodes as yaml to stdout
- *wwctl node import* - Import node(s) from yaml FILE
- *wwctl node list* - List nodes
- *wwctl node sensors* - Show node IPMI sensor information
- *wwctl node set* - Configure node properties
- *wwctl node status* - View the provisioning status of nodes
- *wwctl node unset* - Unset/clear node properties

CHAPTER

SIXTY

WWCTL NODE ADD

Add new node to Warewulf

60.1 Synopsis

This command will add a new node named NODENAME to Warewulf.

```
wwctl node add [OPTIONS] NODENAME
```

60.2 Options

```
    --asset string                the node's Asset tag (key)
-c, --cluster string              cluster group
    --comment string              arbitrary string comment
-e, --discoverable WWbool[=true]  discoverable in given network (true/false)
    --diskname string             set diskdevice name
    --diskwipe                    whether or not the partition tables shall be wiped
    --fsformat string             format of the file system
    --fsname string               set the file system name which must match a partition name
    --fspath string               the mount point of the file system
    --fswipe                      wipe file system at boot
-G, --gateway ip                  the node's IPv4 network device gateway
    --gateway6 ip                 the node's IPv6 network device gateway
-h, --help                        help for add
-H, --hwaddr string               the device's HW address for given network
    --image string                image name
-i, --init string                 the init process to boot the image
-I, --ipaddr ip                   IPv4 address in given network
    --ipaddr6 ip                  IPv6 address in given network
    --ipmiaddr ip                 the IPMI IP address
    --ipmiescapechar string       the IPMI escape character (defaults: '~')
    --ipmigateway ip              the IPMI gateway
    --ipmiinterface string        the node's IPMI interface (defaults: 'lan')
    --ipminetmask ip              the IPMI netmask
    --ipmipass string             the IPMI password
    --ipmiport string             the IPMI port
    --ipmitagadd stringToString   add ipmi tags (default [])
    --ipmitemplate string         template used for ipmi command
    --ipmiuser string             the IPMI username
```

(continues on next page)

(continued from previous page)

```
    --ipmiwrite WWbool[=true]      writing of IPMI configuration (true/false)
    --ipxe string                  the iPXE template name
-A, --kernelargs strings           kernel arguments
    --kernelversion string         kernel version
    --mtu string                   the MTU
-N, --netdev string                the device for given network
-M, --netmask ip                   the network's netmask
    --netname string               network which is modified (default "default")
    --nettagadd stringToString     add network tags (default [])
    --onboot WWbool[=true]         network device (true/false)
    --partcreate                   create the partition if it does not exist
    --partname string              set the partition name so it can be used by a file system
    --partnumber string            the partition number (if not set, next free slot is used)
    --partsize string              the partition size (if not set, maximum possible size is used)
    --parttype string              the partition type GUID
    --partwipe                     if true, Ignition will clobber an existing partition if it does not match the
↪config
    --prefixlen6 string            the network's IPv6 prefix length
-p, --primarynet string            the primary network interface
-P, --profile strings              the node's profile members (comma separated)
    --root string                  the rootfs
-R, --runtime-overlays strings     the runtime overlay
-O, --system-overlays strings      the system overlay
    --tagadd stringToString        add tags (default [])
-T, --type string                  device type of given network
```

60.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

60.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYONE

WWCTL NODE CONSOLE

Connect to IPMI console

61.1 Synopsis

Start a new IPMI console for NODENAME.

```
wwctl node console [OPTIONS] NODENAME
```

61.2 Options

```
-h, --help    help for console
```

61.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

61.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYTWO

WWCTL NODE DELETE

Delete a node from Warewulf

62.1 Synopsis

This command will remove NODE(s) from the Warewulf node configuration.

```
wwctl node delete [OPTIONS] NODE [NODE ...]
```

62.2 Options

```
-h, --help   help for delete
-y, --yes    Set 'yes' to all questions asked
```

62.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

62.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYTHREE

WWCTL NODE EDIT

Edit node(s) with editor

63.1 Synopsis

This command opens an editor for the given nodes.

```
wwctl node edit [OPTIONS] NODENAME
```

63.2 Options

```
-h, --help        help for edit
    --noheader    Do not print header
```

63.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

63.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYFOUR

WWCTL NODE EXPORT

Export nodes as yaml to stdout

64.1 Synopsis

This command exports the given nodes as yaml to stdout.

```
wwctl node export NODENAME
```

64.2 Options

```
-h, --help   help for export
```

64.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

64.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYFIVE

WWCTL NODE IMPORT

Import node(s) from yaml FILE

65.1 Synopsis

This command imports all the nodes defined in a YAML file. It will overwrite nodes with same name.

```
wwctl node import [OPTIONS] FILE
```

65.2 Options

```
-h, --help   help for import
-y, --yes    Set 'yes' to all questions asked
```

65.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

65.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYSIX

WWCTL NODE LIST

List nodes

66.1 Synopsis

This command lists all configured nodes. Optionally, it will list only nodes matching a PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node list [OPTIONS] [PATTERN]
```

66.2 Options

```
-a, --all     Show all node configurations
-h, --help    help for list
-i, --ipmi    Show node IPMI configurations
-j, --json    Show json format
-l, --long    Show long or wide format
-n, --net     Show node network configurations
-y, --yaml    Show yaml format
```

66.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

66.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYSEVEN

WWCTL NODE SENSORS

Show node IPMI sensor information

67.1 Synopsis

Show IPMI sensor information for nodes matching PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node sensors [OPTIONS] PATTERN
```

67.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-F, --full         show detailed output.
-h, --help         help for sensors
-s, --show         only show command which will be executed
```

67.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

67.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYEIGHT

WWCTL NODE SET

Configure node properties

68.1 Synopsis

This command sets configuration properties for nodes matching PATTERN.

Note: use the string 'UNSET' to remove a configuration Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node set [OPTIONS] PATTERN
```

68.2 Options

```
-a, --all                       Set all nodes
    --asset string              the node's Asset tag (key)
-c, --cluster string            cluster group
    --comment string            arbitrary string comment
-e, --discoverable WWbool[=true]   discoverable in given network (true/false)
    --diskdel string            delete the disk from the configuration
    --diskname string           set diskdevice name
    --diskwipe                  whether or not the partition tables shall be wiped
    --fsdel string              delete the fs from the configuration
    --fsformat string           format of the file system
    --fsname string             set the file system name which must match a partition name
    --fspath string             the mount point of the file system
    --fswipe                    wipe file system at boot
-G, --gateway ip                the node's IPv4 network device gateway
    --gateway6 ip               the node's IPv6 network device gateway
-h, --help                      help for set
-H, --hwaddr string             the device's HW address for given network
    --image string              image name
-i, --init string               the init process to boot the image
-I, --ipaddr ip                 IPv4 address in given network
    --ipaddr6 ip                IPv6 address in given network
    --ipmiaddr ip               the IPMI IP address
    --ipmiescapechar string     the IPMI escape character (defaults: '~')
    --ipmigateway ip            the IPMI gateway
    --ipmiinterface string      the node's IPMI interface (defaults: 'lan')
    --ipminetmask ip            the IPMI netmask
```

(continues on next page)

(continued from previous page)

```
    --ipmipass string              the IPMI password
    --ipmiport string              the IPMI port
    --ipmitagadd stringToString    add ipmi tags (default [])
    --ipmitagdel strings           delete ipmi tags
    --ipmitemplate string          template used for ipmi command
    --ipmiuser string              the IPMI username
    --ipmiwrite WWbool[=true]      writing of IPMI configuration (true/false)
    --ipxe string                  the iPXE template name
-A, --kernelargs strings           kernel arguments
    --kernelversion string         kernel version
    --mtu string                   the MTU
    --netdel string                network to delete
-N, --netdev string                the device for given network
-M, --netmask ip                   the network's netmask
    --netname string               network which is modified (default "default")
    --nettagadd stringToString     add network tags (default [])
    --nettagdel strings            delete network tags
    --onboot WWbool[=true]         network device (true/false)
    --partcreate                   create the partition if it does not exist
    --partdel string               delete the partition from the configuration
    --partname string              set the partition name so it can be used by a file system
    --partnumber string            the partition number (if not set, next free slot is used)
    --partsize string              the partition size (if not set, maximum possible size is used)
    --parttype string              the partition type GUID
    --partwipe                     if true, Ignition will clobber an existing partition if it does not match the
↪config
    --prefixlen6 string            the network's IPv6 prefix length
-p, --primarynet string            the primary network interface
-P, --profile strings              the node's profile members (comma separated)
    --root string                  the rootfs
-R, --runtime-overlays strings     the runtime overlay
-O, --system-overlays strings      the system overlay
    --tagadd stringToString        add tags (default [])
    --tagdel strings               delete tags
-T, --type string                  device type of given network
-y, --yes                          Set 'yes' to all questions asked
```

68.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

68.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SIXTYNINE

WWCTL NODE STATUS

View the provisioning status of nodes

69.1 Synopsis

View and monitor the status of nodes as they are provisioned and check in.

```
wwctl node status [OPTIONS] [NODENAME...]
```

69.2 Options

```
-h, --help          help for status
-l, --last          Sort by the last check-in time
-r, --reverse       Reverse the sort order
-t, --time int      Filter by last checkin time (seconds)
-u, --unknown       Only show nodes of unknown status
-U, --update int    Set the update frequency for 'watch' (ms) (default 500)
-w, --watch         Watch the status automatically
```

69.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

69.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SEVENTY

WWCTL NODE UNSET

Unset/clear node properties

70.1 Synopsis

Unsets configuration properties for nodes matching PATTERN.

Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node unset [OPTIONS] PATTERN
```

70.2 Options

```
    --asset              the node's Asset tag (key)
-c, --cluster            cluster group
    --comment              arbitrary string comment
-e, --discoverable       discoverable in given network (true/false)
    --diskname string     disk to modify
    --diskwipe           whether or not the partition tables shall be wiped
-f, --force              Force configuration (even on error)
    --fsformat           format of the file system
    --fsname string       filesystem to modify
    --fspath             the mount point of the file system
    --fswipe             wipe file system at boot
-G, --gateway              the node's IPv4 network device gateway
    --gateway6           the node's IPv6 network device gateway
-h, --help               help for unset
-H, --hwaddr               the device's HW address for given network
    --image              image name
-i, --init             the init process to boot the image
-I, --ipaddr             IPv4 address in given network
    --ipaddr6            IPv6 address in given network
    --ipmiaddr            the IPMI IP address
    --ipmiescapechar      the IPMI escape character (defaults: '~')
    --ipmigateway         the IPMI gateway
    --ipmiinterface      the node's IPMI interface (defaults: 'lan')
    --ipminetmask         the IPMI netmask
    --ipmipass           the IPMI password
    --ipmiport           the IPMI port
```

(continues on next page)

(continued from previous page)

```
    --ipmitag strings     Unset IPMI tags
    --ipmitemplate        template used for ipmi command
    --ipmiuser            the IPMI username
    --ipmiwrite           writing of IPMI configuration (true/false)
    --ipxe                the iPXE template name
-A, --kernelargs          kernel arguments
    --kernelversion       kernel version
    --mtu                 the MTU
-N, --netdev              the device for given network
-M, --netmask             the network's netmask
    --netname string      network which is modified (default "default")
    --nettag strings      Unset network tags
    --onboot              network device (true/false)
    --partcreate          create the partition if it does not exist
    --partname string     partition to modify (requires --diskname)
    --partnumber          the partition number (if not set, next free slot is used)
    --partsize            the partition size (if not set, maximum possible size is used)
    --parttype            the partition type GUID
    --partwipe            if true, Ignition will clobber an existing partition if it does not match the config
    --prefixlen6          the network's IPv6 prefix length
-p, --primarynet          the primary network interface
-P, --profile             the node's profile members (comma separated)
    --root                the rootfs
-R, --runtime-overlays    the runtime overlay
-O, --system-overlays     the system overlay
    --tag strings         Unset tags
-T, --type                device type of given network
-y, --yes                 Set 'yes' to all questions asked
```

70.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

70.4 SEE ALSO

- *wwctl node* - Node management

CHAPTER

SEVENTYONE

WWCTL OVERLAY

Warewulf Overlay Management

71.1 Synopsis

Management interface for Warewulf overlays

71.2 Options

```
-h, --help    help for overlay
```

71.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

71.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl overlay build* - (Re)build node overlays
- *wwctl overlay chmod* - Change file permissions in an overlay
- *wwctl overlay chown* - Change file ownership within an overlay
- *wwctl overlay create* - Initialize a new Overlay
- *wwctl overlay delete* - Delete Warewulf Overlay or files
- *wwctl overlay edit* - Edit or create a file within a Warewulf Overlay
- *wwctl overlay import* - Import a file into a Warewulf Overlay
- *wwctl overlay info* - Show variables for a template file in an overlay
- *wwctl overlay list* - List Warewulf Overlays and files
- *wwctl overlay mkdir* - Create a new directory within an Overlay
- *wwctl overlay show* - Show (cat) a file within a Warewulf Overlay

CHAPTER

SEVENTYTWO

WWCTL OVERLAY BUILD

(Re)build node overlays

72.1 Synopsis

This command builds overlays for given nodes.

```
wwctl overlay build [OPTIONS] NODENAME...
```

72.2 Options

```
-h, --help          help for build
    --workers int   The number of parallel workers building overlays (<=0 indicates 1 worker per CPU)
```

72.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

72.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYTHREE

WWCTL OVERLAY CHMOD

Change file permissions in an overlay

73.1 Synopsis

Changes the permissions of a single FILENAME within an overlay. You can use any MODE format supported by the chmod command.

```
wwctl overlay chmod [OPTIONS] OVERLAY_NAME FILENAME MODE
```

73.2 Examples

```
wwctl overlay chmod default /etc/hostname.ww 0660
```

73.3 Options

```
-h, --help    help for chmod
```

73.4 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

73.5 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYFOUR

WWCTL OVERLAY CHOWN

Change file ownership within an overlay

74.1 Synopsis

This command changes the ownership of a FILE within the system or runtime OVERLAY_NAME to the user specified by UID. Optionally, it will also change group ownership to GID.

```
wwctl overlay chown [OPTIONS] OVERLAY_NAME FILE UID[:GID]
```

74.2 Options

```
-h, --help    help for chown
```

74.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

74.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYFIVE

WWCTL OVERLAY CREATE

Initialize a new Overlay

75.1 Synopsis

This command creates a new empty overlay with the given OVERLAY_NAME.

```
wwctl overlay create [OPTIONS] OVERLAY_NAME
```

75.2 Options

```
-h, --help   help for create
```

75.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

75.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYSIX

WWCTL OVERLAY DELETE

Delete Warewulf Overlay or files

76.1 Synopsis

This command will delete FILEs within OVERLAY_NAME or the entire OVERLAY_NAME if no files are listed. Use with caution!

```
wwctl overlay delete [OPTIONS] OVERLAY_NAME [FILE [FILE ...]]
```

76.2 Options

```
-f, --force      Force deletion of a non-empty overlay
-h, --help       help for delete
-p, --parents    Remove empty parent directories
```

76.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

76.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYSEVEN

WWCTL OVERLAY EDIT

Edit or create a file within a Warewulf Overlay

77.1 Synopsis

This command will open the FILE for editing or create a new file within the OVERLAY_NAME. Note: files created with a '.ww' suffix will always be parsed as Warewulf template files, and the suffix will be removed automatically.

```
wwctl overlay edit [OPTIONS] OVERLAY_NAME FILE
```

77.2 Options

```
-h, --help          help for edit
-m, --mode int32    Permission mode for directory (default 493)
-p, --parents       Create any necessary parent directories
```

77.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

77.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYEIGHT

WWCTL OVERLAY IMPORT

Import a file into a Warewulf Overlay

78.1 Synopsis

This command imports the FILE into the Warewulf OVERLAY_NAME. Optionally, the file can be renamed to NEW_NAME

```
wwctl overlay import [OPTIONS] OVERLAY_NAME FILE [NEW_NAME]
```

78.2 Options

```
-h, --help          help for import
-o, --overwrite     Overwrite file if exists
-p, --parents       Create any necessary parent directories
    --workers int   The number of parallel workers building overlays (<=0 indicates 1 worker per CPU)
```

78.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

78.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

SEVENTYNINE

WWCTL OVERLAY INFO

Show variables for a template file in an overlay

79.1 Synopsis

This command will show the variables for a given template file in a given overlay.

```
wwctl overlay info [flags] OVERLAY_NAME FILE_PATH
```

79.2 Options

```
-h, --help   help for info
```

79.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

79.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

EIGHTY

WWCTL OVERLAY LIST

List Warewulf Overlays and files

80.1 Synopsis

This command displays information about all Warewulf overlays or the specified OVERLAY_NAME. It also supports listing overlay content information.

```
wwctl overlay list [OPTIONS] OVERLAY_NAME
```

80.2 Options

```
-a, --all     List the contents of overlays
-h, --help    help for list
-l, --long    List 'long' of all overlay contents
-p, --path    Show the absolute path to the overlay
```

80.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

80.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

EIGHTYONE

WWCTL OVERLAY MKDIR

Create a new directory within an Overlay

81.1 Synopsis

This command creates a new directory within the Warewulf OVERLAY_NAME.

```
wwctl overlay mkdir [OPTIONS] OVERLAY_NAME DIRECTORY
```

81.2 Options

```
-h, --help          help for mkdir
-m, --mode int32    Permission mode for directory (default 493)
```

81.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

81.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

EIGHTYTWO

WWCTL OVERLAY SHOW

Show (cat) a file within a Warewulf Overlay

82.1 Synopsis

This command displays the contents of FILE within OVERLAY_NAME.

```
wwctl overlay show [OPTIONS] OVERLAY_NAME FILE
```

82.2 Options

```
-h, --help            help for show
-q, --quiet           do not print information if multiple, backup files are written
-r, --render string   node used for the variables in the template
```

82.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

82.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management

CHAPTER

EIGHTYTHREE

WWCTL POWER

Warewulf node power management

83.1 Synopsis

This command controls the power state of nodes.

83.2 Options

```
-h, --help    help for power
```

83.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

83.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl power cycle* - Power cycle the given node(s)
- *wwctl power off* - Power off the given node(s)
- *wwctl power on* - Power on the given node(s)
- *wwctl power reset* - Issue a reset to node(s)
- *wwctl power soft* - Gracefully shuts down the given node(s)
- *wwctl power status* - Show power status for the given node(s)

CHAPTER

EIGHTYFOUR

WWCTL POWER CYCLE

Power cycle the given node(s)

84.1 Synopsis

This command cycles power for a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power cycle [OPTIONS] [PATTERN ...]
```

84.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-h, --help         help for cycle
-s, --show         only show command which will be executed
```

84.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

84.4 SEE ALSO

- *wwctl power* - Warewulf node power management

CHAPTER

EIGHTYFIVE

WWCTL POWER OFF

Power off the given node(s)

85.1 Synopsis

This command will shutdown power to a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power off [OPTIONS] [PATTERN ...]
```

85.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-h, --help         help for off
-s, --show         only show command which will be executed
```

85.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

85.4 SEE ALSO

- *wwctl power* - Warewulf node power management

CHAPTER

EIGHTYSIX

WWCTL POWER ON

Power on the given node(s)

86.1 Synopsis

This command will power on a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power on [OPTIONS] [PATTERN ...] [flags]
```

86.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-h, --help         help for on
-s, --show         only show command which will be executed
```

86.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

86.4 SEE ALSO

- *wwctl power* - Warewulf node power management

CHAPTER

EIGHTYSEVEN

WWCTL POWER RESET

Issue a reset to node(s)

87.1 Synopsis

This command will issue a reset to a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power reset [OPTIONS] [PATTERN ...]
```

87.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-h, --help         help for reset
-s, --show         only show command which will be executed
```

87.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

87.4 SEE ALSO

- *wwctl power* - Warewulf node power management

CHAPTER

EIGHTYEIGHT

WWCTL POWER SOFT

Gracefully shuts down the given node(s)

88.1 Synopsis

This command uses the operating system to shut down the set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power soft [OPTIONS] [PATTERN ...]
```

88.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-h, --help         help for soft
-s, --show         only show command which will be executed
```

88.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

88.4 SEE ALSO

- *wwctl power* - Warewulf node power management

CHAPTER

EIGHTYNINE

WWCTL POWER STATUS

Show power status for the given node(s)

89.1 Synopsis

This command displays the power status of a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power status [OPTIONS] [PATTERN ...]
```

89.2 Options

```
    --fanout int   how many command should be executed in parallel (default 50)
-h, --help         help for status
-s, --show         only show command which will be executed
```

89.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

89.4 SEE ALSO

- *wwctl power* - Warewulf node power management

CHAPTER

NINETY

WWCTL PROFILE

Node configuration profile management

90.1 Synopsis

Management of node profile settings

90.2 Options

```
-h, --help    help for profile
```

90.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

90.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl profile add* - Add a new node profile
- *wwctl profile delete* - Delete a node profile
- *wwctl profile edit* - Edit profile(s) with editor
- *wwctl profile list* - List profiles and configurations
- *wwctl profile set* - Configure node profile properties
- *wwctl profile unset* - Unset/clear profile properties

CHAPTER

NINETYONE

WWCTL PROFILE ADD

Add a new node profile

91.1 Synopsis

This command adds a new named PROFILE.

```
wwctl profile add PROFILE
```

91.2 Options

```
-c, --cluster string              cluster group
    --comment string              arbitrary string comment
    --diskname string             set diskdevice name
    --diskwipe                    whether or not the partition tables shall be wiped
    --fsformat string             format of the file system
    --fsname string               set the file system name which must match a partition name
    --fspath string               the mount point of the file system
    --fswipe                      wipe file system at boot
-G, --gateway ip                  the node's IPv4 network device gateway
    --gateway6 ip                 the node's IPv6 network device gateway
-h, --help                        help for add
-H, --hwaddr string               the device's HW address for given network
    --image string                image name
-i, --init string                 the init process to boot the image
-I, --ipaddr ip                   IPv4 address in given network
    --ipaddr6 ip                  IPv6 address in given network
    --ipmiaddr ip                 the IPMI IP address
    --ipmiescapechar string       the IPMI escape character (defaults: '~')
    --ipmigateway ip              the IPMI gateway
    --ipmiinterface string        the node's IPMI interface (defaults: 'lan')
    --ipminetmask ip              the IPMI netmask
    --ipmipass string             the IPMI password
    --ipmiport string             the IPMI port
    --ipmitagadd stringToString   add ipmi tags (default [])
    --ipmitemplate string         template used for ipmi command
    --ipmiuser string             the IPMI username
    --ipmiwrite WWbool[=true]     writing of IPMI configuration (true/false)
    --ipxe string                 the iPXE template name
```

(continues on next page)

(continued from previous page)

```
-A, --kernelargs strings          kernel arguments
    --kernelversion string        kernel version
    --mtu string                  the MTU
-N, --netdev string               the device for given network
-M, --netmask ip                  the network's netmask
    --netname string              network which is modified (default "default")
    --nettagadd stringToString    add network tags (default [])
    --onboot WWbool[=true]        network device (true/false)
    --partcreate                  create the partition if it does not exist
    --partname string             set the partition name so it can be used by a file system
    --partnumber string           the partition number (if not set, next free slot is used)
    --partsize string             the partition size (if not set, maximum possible size is used)
    --parttype string             the partition type GUID
    --partwipe                    if true, Ignition will clobber an existing partition if it does not match the
↪config
    --prefixlen6 string           the network's IPv6 prefix length
-p, --primarynet string           the primary network interface
-P, --profile strings             the node's profile members (comma separated)
    --root string                 the rootfs
-R, --runtime-overlays strings    the runtime overlay
-O, --system-overlays strings     the system overlay
    --tagadd stringToString       add tags (default [])
-T, --type string                 device type of given network
```

91.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

91.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

CHAPTER

NINETYTWO

WWCTL PROFILE DELETE

Delete a node profile

92.1 Synopsis

This command deletes the node PROFILE. You may use a pattern for PROFILE.

```
wwctl profile delete [OPTIONS] PROFILE
```

92.2 Options

```
-h, --help   help for delete
-y, --yes    Set 'yes' to all questions asked
```

92.3 Options inherited from parent commands

```
-d, --debug                Run with debugging messages enabled.
-v, --verbose              Run with increased verbosity.
    --warewulfconf string  Set the warewulf configuration file
```

92.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

CHAPTER

NINETYTHREE

WWCTL PROFILE EDIT

Edit profile(s) with editor

93.1 Synopsis

This command opens an editor for the given profiles.

```
wwctl profile edit [OPTIONS] PROFILE
```

93.2 Options

```
-h, --help        help for edit
    --noheader    Do not print header
```

93.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

93.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

CHAPTER

NINETYFOUR

WWCTL PROFILE LIST

List profiles and configurations

94.1 Synopsis

This command will display configurations for PROFILE.

```
wwctl profile list [OPTIONS] [PROFILE ...]
```

94.2 Options

```
-a, --all     Show all profile configurations
-h, --help    help for list
-j, --json    Show profile configurations via json format
-y, --yaml    Show profile configurations via yaml format
```

94.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

94.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

CHAPTER

NINETYFIVE

WWCTL PROFILE SET

Configure node profile properties

95.1 Synopsis

This command sets configuration properties for the node PROFILE(s).

Note: use the string 'UNSET' to remove a configuration

```
wwctl profile set [OPTIONS] [PROFILE ...] [flags]
```

95.2 Options

```
-c, --cluster string              cluster group
    --comment string              arbitrary string comment
    --diskdel string              delete the disk from the configuration
    --diskname string             set diskdevice name
    --diskwipe                    whether or not the partition tables shall be wiped
    --fsdel string                delete the fs from the configuration
    --fsformat string             format of the file system
    --fsname string               set the file system name which must match a partition name
    --fspath string               the mount point of the file system
    --fswipe                      wipe file system at boot
-G, --gateway ip                  the node's IPv4 network device gateway
    --gateway6 ip                 the node's IPv6 network device gateway
-h, --help                        help for set
-H, --hwaddr string               the device's HW address for given network
    --image string                image name
-i, --init string                 the init process to boot the image
-I, --ipaddr ip                   IPv4 address in given network
    --ipaddr6 ip                  IPv6 address in given network
    --ipmiaddr ip                 the IPMI IP address
    --ipmiescapechar string       the IPMI escape character (defaults: '~')
    --ipmigateway ip              the IPMI gateway
    --ipmiinterface string        the node's IPMI interface (defaults: 'lan')
    --ipminetmask ip              the IPMI netmask
    --ipmipass string             the IPMI password
    --ipmiport string             the IPMI port
    --ipmitagadd stringToString   add ipmi tags (default [])
    --ipmitagdel strings          delete ipmi tags
```

(continues on next page)

(continued from previous page)

```
    --ipmitemplate string          template used for ipmi command
    --ipmiuser string              the IPMI username
    --ipmiwrite WWbool[=true]      writing of IPMI configuration (true/false)
    --ipxe string                  the iPXE template name
-A, --kernelargs strings           kernel arguments
    --kernelversion string         kernel version
    --mtu string                   the MTU
    --netdel string                network to delete
-N, --netdev string                the device for given network
-M, --netmask ip                   the network's netmask
    --netname string               network which is modified (default "default")
    --nettagadd stringToString     add network tags (default [])
    --nettagdel strings            delete network tags
    --onboot WWbool[=true]         network device (true/false)
    --partcreate                   create the partition if it does not exist
    --partdel string               delete the partition from the configuration
    --partname string              set the partition name so it can be used by a file system
    --partnumber string            the partition number (if not set, next free slot is used)
    --partsize string              the partition size (if not set, maximum possible size is used)
    --parttype string              the partition type GUID
    --partwipe                     if true, Ignition will clobber an existing partition if it does not match the␣
↪config
    --prefixlen6 string            the network's IPv6 prefix length
-p, --primarynet string            the primary network interface
-P, --profile strings              the node's profile members (comma separated)
    --root string                  the rootfs
-R, --runtime-overlays strings     the runtime overlay
-O, --system-overlays strings      the system overlay
    --tagadd stringToString        add tags (default [])
    --tagdel strings               delete tags
-T, --type string                  device type of given network
-y, --yes                          Set 'yes' to all questions asked
```

95.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

95.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

CHAPTER

NINETYSIX

WWCTL PROFILE UNSET

Unset/clear profile properties

96.1 Synopsis

Unsets configuration properties for the specified PROFILE(s).

```
wwctl profile unset [OPTIONS] PROFILE...
```

96.2 Options

```
-c, --cluster            cluster group
    --comment            arbitrary string comment
    --diskname string    disk to modify
    --diskwipe           whether or not the partition tables shall be wiped
-f, --force              Force configuration (even on error)
    --fsformat           format of the file system
    --fsname string      filesystem to modify
    --fspath             the mount point of the file system
    --fswipe             wipe file system at boot
-G, --gateway            the node's IPv4 network device gateway
    --gateway6           the node's IPv6 network device gateway
-h, --help               help for unset
-H, --hwaddr             the device's HW address for given network
    --image              image name
-i, --init               the init process to boot the image
-I, --ipaddr             IPv4 address in given network
    --ipaddr6            IPv6 address in given network
    --ipmiaddr           the IPMI IP address
    --ipmiescapechar     the IPMI escape character (defaults: '~')
    --ipmigateway        the IPMI gateway
    --ipmiinterface      the node's IPMI interface (defaults: 'lan')
    --ipminetmask        the IPMI netmask
    --ipmipass           the IPMI password
    --ipmiport           the IPMI port
    --ipmitag strings    Unset IPMI tags
    --ipmitemplate       template used for ipmi command
    --ipmiuser           the IPMI username
    --ipmiwrite          writing of IPMI configuration (true/false)
```

(continues on next page)

(continued from previous page)

```
    --ipxe               the iPXE template name
-A, --kernelargs          kernel arguments
    --kernelversion       kernel version
    --mtu                the MTU
-N, --netdev              the device for given network
-M, --netmask              the network's netmask
    --netname string      network which is modified (default "default")
    --nettag strings      Unset network tags
    --onboot             network device (true/false)
    --partcreate          create the partition if it does not exist
    --partname string     partition to modify (requires --diskname)
    --partnumber          the partition number (if not set, next free slot is used)
    --partsize           the partition size (if not set, maximum possible size is used)
    --parttype            the partition type GUID
    --partwipe            if true, Ignition will clobber an existing partition if it does not match the config
    --prefixlen6         the network's IPv6 prefix length
-p, --primarynet           the primary network interface
-P, --profile            the node's profile members (comma separated)
    --root               the rootfs
-R, --runtime-overlays    the runtime overlay
-O, --system-overlays     the system overlay
    --tag strings         Unset tags
-T, --type                device type of given network
-y, --yes                Set 'yes' to all questions asked
```

96.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

96.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

CHAPTER

NINETYSEVEN

WWCTL SERVER

Start Warewulf server

97.1 Synopsis

Start Warewulf server

```
wwctl server [OPTIONS]
```

97.2 Options

```
-h, --help   help for server
```

97.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

97.4 SEE ALSO

- *wwctl* - Warewulf Control

CHAPTER

NINETYEIGHT

WWCTL SSH

SSH into configured nodes in parallel

98.1 Synopsis

Easily ssh into nodes in parallel to run non-interactive commands Node patterns are a comma-separated list of individual patterns. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl ssh [OPTIONS] NODE_PATTERN COMMAND
```

98.2 Options

```
-n, --dryrun        Show commands to run
-f, --fanout int    How many connections to run in parallel (default 32)
-h, --help          help for ssh
    --rsh string    Path to use for RSH/SSH command (default "/usr/bin/ssh")
-s, --sleep int     Seconds to sleep inbetween processes
```

98.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

98.4 SEE ALSO

- *wwctl* - Warewulf Control

CHAPTER

NINETYNINE

WWCTL UPGRADE

Upgrade configuration files

99.1 Synopsis

Upgrade warewulf.conf or nodes.conf from a previous version of Warewulf 4 to a format supported by the current version.

99.2 Options

```
-h, --help    help for upgrade
```

99.3 Options inherited from parent commands

```
-d, --debug               Run with debugging messages enabled.
-v, --verbose             Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

99.4 SEE ALSO

- *wwctl* - Warewulf Control
- *wwctl upgrade config* - Upgrade an existing warewulf.conf
- *wwctl upgrade nodes* - Upgrade an existing nodes.conf

WWCTL UPGRADE CONFIG

Upgrade an existing warewulf.conf

100.1 Synopsis

Upgrades warewulf.conf from a previous version of Warewulf 4 to a format supported by the current version.

```
wwctl upgrade config [OPTIONS]
```

100.2 Options

```
-h, --help                  help for config
-i, --input-path string     Path to a legacy warewulf.conf
-o, --output-path string    Path to write the upgraded warewulf.conf to
```

100.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

100.4 SEE ALSO

- *wwctl upgrade* - Upgrade configuration files

CHAPTER

ONE

WWCTL UPGRADE NODES

Upgrade an existing nodes.conf

101.1 Synopsis

Upgrades nodes.conf from a previous version of Warewulf 4 to a format supported by the current version.

```
wwctl upgrade nodes [OPTIONS]
```

101.2 Options

```
    --add-defaults               Configure a default profile and set default node values
-h, --help                       help for nodes
-i, --input-path string          Path to a legacy nodes.conf
-o, --output-path string         Path to write the upgraded nodes.conf to
    --replace-overlays           Replace 'wwinit' and 'generic' overlays with their split replacements
    --with-warewulfconf string   Path to a legacy warewulf.conf
```

101.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

101.4 SEE ALSO

- *wwctl upgrade* - Upgrade configuration files

CHAPTER

TWO

WWCTL VERSION

Version information

102.1 Synopsis

This command will print the Warewulf version.

```
wwctl version [flags]
```

102.2 Options

```
-f, --full   List all compiled in variables.
-h, --help   help for version
```

102.3 Options inherited from parent commands

```
-d, --debug                 Run with debugging messages enabled.
-v, --verbose               Run with increased verbosity.
    --warewulfconf string   Set the warewulf configuration file
```

102.4 SEE ALSO

- *wwctl* - Warewulf Control

CHAPTER

THREE

V4.6.0 RELEASE NOTES

v4.6.0 is a significant upgrade, with many changes relative to the v4.5.x series.

Particularly significant changes, especially those affecting the user interface, are described below. Additional changes not impacting the user interface are listed in the CHANGELOG.

103.1 Documentation

The user documentation has been significantly refactored and re-written. The majority of changes mentioned here should be documented in more detail there, as well, and the reorganization and deduplication supports better documentation maintenance in the future.

The documentation also now includes complete and automatically-generated references for all wwctl commands, subcommands, and options.

103.2 Upgrade

Warewulf v4.6.0 adds the wwctl upgrade command to assist with upgrading from previous versions of Warewulf v4. This command updates existing configuration files for use with the current version.

There are two subcommands:

- wwctl upgrade config updates warewulf.conf.
- wwctl upgrade nodes updates nodes.conf.

Both of these will attempt to update their respective configuration file in-place, retaining a copy of the previous version with a -old suffix. Alternatively, you can see what each command will do by specifying an --output-path=- option, to direct the output of the command to "standard out."

wwctl upgrade nodes additionally requires two options to be specified:

- --add-defaults adds default settings to the default profile when those settings are absent. If you *do not* wish to add defaults, specify --add-defaults=false.

  For more information, see the section on the default profile, below.

- --replace-overlays replaces any reference to the "generic" or "wwinit" overlays with a new set of overlays that replace their behavior. Because an overlay named "wwinit" is present in both the legacy and the upgraded state, --replace-overlays is **not** idempotent, and should only be used once. If you *do not* wish to replace overlays, specify --replace-overlays=false.

  For more information, see the section on overlays, below.

103.3 The default profile

At various points Warewulf v4 has had a number of built-in default settings. These settings were once "compiled in," and more recently were moved to a dedicated defaults.conf file. In v4.6.0 these defaults have been moved to the default profile, and are included in nodes.conf for new installations.

A legacy configuration from a previous Warewulf installation can be updated to include recommended defaults using wwctl upgrade nodes --add-defaults. (For more information, refer to the section on upgrades, above.)

If the default overlay exists, it will be automatically (and explicitly) included by new nodes created with wwctl node add. It is otherwise not "special," and may be removed if a different organization is preferred.

A few wwctl commands have previously had --setdefault options to automatically update the default profile: these options have been removed in v4.6.0.

103.4 Images

One of the more visible changes to Warewulf in v4.6.0: "containers" have been renamed to "images" (more specifically, "node images") throughout the interface, documentation, and even code. This decision (requested by the user community) is meant to alleviate confusion regarding whether Warewulf "containers" are "real" containers running on a container runtime with potential performance and operational consequences.

Warewulf "containers" have never been "virtualized" or executed with a container runtime. Rather, the name "container" was selected to imply the integration in v4 with the container ecosystem of tooling for defining, building, storing, and testing node images. But this terminology ended up causing persistent confusion, so a more industry-standard "node image" terminology has been adopted instead.

The wwctl container command is retained as an alias for the new wwctl image command. The variables .Container and .ContainerName are also retained as overlay template variables. These backwards-compatibility retentions will continue to work through the v4.6.x series.

There are smaller changes to the image system, as well:

- wwctl image shell now supports a WW_HISTFILE environment variable to save shell history *inside* the image.
- wwctl image shell now supports a WW_PS1 environment variable to specify the prompt for the interactive shell. The default prompt has also been updated to indicate the current directory.
- wwctl image import now supports --username and --password parameters for authenticating to a secure OCI registry.
- wwctl image import now supports a --nohttps parameter to use HTTP, rather than HTTPS, when importing an image from an OCI registry.
- wwctl image import now supports a --platform parameter to specify a different target architecture (e.g.., for importing an aarch64 image into an x86_64 Warewulf server). This simplifies importing images in a multi-architecture environment.
- wwctl image <exec|shell|copy> all now support a --build flag to control whether the image should be automatically rebuilt after the operation. (For exec and shell the default value is "true", and may be disabled with --build=false. For copy the default value is "false", and may be enabled with --build or --build=true.)
- Warewulf v4.5 used the permissions on an image's rootfs/ directory to determine a "read-only" state of the image. This behavior is now replaced with a sentinel readonly file stored alongside rootfs/ in the image "chroot" directory. (For more information, see the "known issues" section in the Warewulf documentation.)

103.5 Kernels

Warewulf v4.6.0 removes the wwctl kernel command, and all its subcommands, along with the wwctl <node|profile> <add|set> --kerneloverride parameter. All kernels are now provisioned from an associated node image. If more than one kernel is present in the image, Warewulf uses the highest-version, non-debug kernel; but an explicit kernel version or kernel path can be specified with wwctl <node|profile> <add|set> --kernelversion.

wwctl image kernels provides a new interface to show what kernels are available in each image, along with information regarding the detected version, whether the kernel is the "default" for the image, and how many nodes are configured to use it. (If no version is specified, the detected kernel version is provided to overlay templates as .Kernel.Version).

Kernel arguments are also now represented as a list, rather than as a flat string. This allows kernel arguments to be combined from various levels (e.g., profiles and the node) without having to re-specify the full argument list. However, this also means that kernel arguments must be explicitly *negated* to remove them from prior specification. (For example, you might need to specify both ~crashkernel=no and crashkernel=512MB.) List arguments to wwctl <node|profile> <add|set> may be comma-separated; so arguments that *contain* a comma must now be quoted on the command-line. (e.g., wwctl profile set default --cluster oso --kernelargs 'console=tty0,"console=ttyS0,115200"')

103.6 Overlays

wwctl overlay build has been enhanced to build overlays in parallel, and has also been made significantly more efficient. As a result, building overlay images for large clusters now takes significantly less time. By default, the number of parallel workers is equal to the number of CPUs on the Warewulf server; this can be adjusted with a new wwctl overlay <import|build> --workers=0 parameter.

The "wwinit" and "generic" overlays have been split into multiple overlays based on discrete functionality. Their equivalents may be substituted using wwctl upgrade nodes --replace-overlays. (See the section on upgrading above.) This supports more precise removal of default overlay functionality from a given node or profile by removing only a subset of the default overlays. (For example, you may wish to include only one of the network management overlays, NetworkManager, ifcfg, wicked, or debian.interfaces.)

Overlays have been further separated into "distribution" and "site" overlays. All overlays provided with Warewulf are "distribution" overlays, and should not be modified. New overlays, and modifications to distribution overlays, are stored as "site" overlays. Site overlays are retained between Warewulf upgrades, and take precedence over a distribution overlay of the same name.

wwctl overlay build --host and --nodes have been removed to clarify that the host overlay is not "built." To support development and debugging of the host overlay, wwctl overlay show --render=host now renders overlay templates as they would be applied to the Warewulf server. #623

There are smaller changes to the overlay system, as well:

- wwctl <node|profile> <add|set> [--system-overlays|--runtime-overlays] replaces --wwinit and --runtime, respectively. (The original flags are retained, but deprecated.)
- wwctl overlay show --render can now accept the path to a template without its .ww suffix.

103.7 Templates

Overlay templates now have access to the full suite of Sprig template functions. Use of the local tr and slice template functions in the distribution overlays has been replaced with their Sprig equivalents (replace and substr, respectively).

An additional template function, *UniqueField*, was added to facilitate removing duplicate passwd and group entries in the syncuser overlay. (For more information, see the section on syncuser, below.)

A set of new template functions, *ImportLink*, *softlink*, and *readlink*, add support for creating symbolic links from overlay templates.

The new *localtime* overlay configures the timezone of a cluster node.

103.8 Network Overlays

The network overlays now support *VLAN tagging*, and *static routes*, and have improved support for configuring a *network bond.* They also now support specifying a *DNS search path.*

Note

Not all functionality is supported by all network overlays.

There are smaller changes to the network overlays, as well:

- The NetworkManager overlay now prevents interfaces without a specified Ipaddr from activating DHCP.
- The NetworkManager overlay now only marks interfaces "unmanaged" if they have neither a Device name nor an Hwaddr specified.

103.9 Profiles

Node profiles now support profiles themselves, allowing for complex nested hierarchies of profiles.

```
nodeprofiles:
  default:
    profiles:
      - rocky
      - net
  rocky:
    image name: rockylinux-9
  net:
    network devices:
      default:
        netmask: 255.255.255.0
        gateway: 192.168.1.1
nodes:
  n1:
    profiles:
      - default
    network devices:
      default:
        ipaddr: 192.168.1.101
```

103.10 Resources

Resources are similar to tags except that their value is an arbitrary data structure rather than just a string. This data is represented as YAML data in nodes.conf, and these data structures may then be referenced by overlay templates to implement more expressive cluster behavior.

Resources can currently only be defined with wwctl <node|profile> edit, or by editing nodes.conf directly.

Note

Resources are defined only at the root of nodes (and profiles), not on network interfaces and IPMI interfaces.

The premiere use of resources is in the refactoring of NFS client configuration.

103.11 NFS Mounts

Cluster node NFS mounts are no longer configured in warewulf.conf. In stead, a new fstab overlay configures NFS (or any other) mounts on cluster nodes based on an fstab resource definition.

```
nodeprofiles:
  default:
    resources:
      fstab:
        - spec: warewulf:/home
          file: /home
          vfstype: nfs
          mntops: defaults,nofail
        - spec: warewulf:/opt
          file: /opt
          vfstype: nfs
          mntops: defaults,noauto,nofail,ro
```

103.12 Syncuser

"Syncuser" has always been optional, but the output of certain commands has been updated to no longer imply that not running syncuser is an error condition. The wwctl image build --syncuser now explicitly opts-in to automatic syncuser during image build, and the wwctl image syncuser --write parameter is now automatically enabled. (Specify --write=false to disable.)

Some syncuser functionality is now implemented in a new syncuser overlay. While this overlay *is* supplied by wwctl upgrade nodes --replace-overlays, it is not included by default in the initial nodes.conf in new deployments.

There are smaller changes to the syncuser, as well:

- The syncuser overlay now looks for the passwd and group databases in sysconfdir, rather than explicitly in /etc/. This change is primarily to support testing; but it does mean that if sysconfdir is a path other than /etc/ then these databases must be provided explicitly (e.g., by copying them or symlinking them into sysconfdir).
- The syncuser overlay now skips duplicate users and groups when generating synchronized passwd and group databases.

103.13 Network Boot and wwinit

The network boot and wwinit process have been made more consistent and verbose for both iPXE and GRUB methods. Additional output and logging provides more information about each step of the process as it happens to aid in troubleshooting. And available network boot options are now presented using an iPXE menu, allowing a specific method to be selected without using a custom iPXE script.

Utilizing the new iPXE menu, specifying an IPXEMenuEntry tag on a cluster node now selects the boot method to use, similar to the previously-existing GrubMenuEntry. The dracut.ipxe script has now been merged into the default iPXE script, and specifying IPXEMenuEntry=dracut now replaces specifying a discrete dracut iPXE template.

An issue that prevented nodes from booting in some circumstances with the Warewulf server configured in "secure" mode have also been resolved: now, if the runtime overlay cannot be downloaded during boot, boot proceeds regardless, and wwclient applies the runtime overlay after boot when it is able to control its source port.

103.14 IPMI

The IPMI system has been refactored to use templates to define the required IPMI template from the cluster node configuration. This is expected to support additional BMC implementation in the future.

103.15 CLI

There have been many enhancements to the wwctl command:

wwctl has been updated to use a different table-formatting library that produces more natural output without extraneous whitespace padding.

wwctl has been updated to add hostlist support to wwctl node and wwctl overlay build. Hostlists have also been enhanced to support comma-separated hostlist patterns. (e.g., n[1-2],n5,n[8-9]) Other pattern formats (regular expressions and globs) are no longer supported.

wwctl has been updated to add "tab completions" for additional parameters.

wwctl <node|profile> list [--yaml|--json] generates machine-readable output in YAML and JSON format, and wwctl node export has been updated to match, including indicating node IDs.

wwctl now returns a non-zero exit code on error.

There are smaller changes to wwctl, as well:

- wwctl <node|profile> list --fullall has been removed.
- wwctl clean removes the OCI cache and vestigial overlay images from deleted nodes.
- wwctl container exec no longer requires a double hyphen (-- --) before flags.

103.16 Debian/Ubuntu

Warewulf v4.6.0 does not yet fully support Debian or Ubuntu; but there have been multiple improvements towards future support:

- warewulfd can now detect Ubuntu-style Dracut initrd images.
- A new netplan overlay adds support for modern Debian/Ubuntu network configuration.
- Multiple internal shell scripts have been updated for POSIX compatibility to support internal use of shells other than Bash.

103.17 Server

The Warewulf server daemon (warewulfd) has been refactored to more closely behave like a 12-factor app. As such, the ability to daemonize has been removed (as have the daemon management commands, wwctl server <start,stop, status,restart,reload>). The server now always runs in the foreground and logs to stdout rather than to /var/log/warewulfd.log or syslog.

The warewulfd.service systemd unit has been updated to read environment variables from /etc/default/warewulfd, and now references an OPTIONS environment variable to supply additional arguments to the wwctl server command. (e.g., OPTIONS=--debug)

wwctl auto-detects some network settings if they are not specified in warewulf.conf. These settings are now written back to warewulf.conf after auto-detection. The ipaddr field of warewulf.conf can now also handle a CIDR-formatted address, which internally populates the netmask and network fields. These network fields are also provided to overlay templates in CIDR format as IpCIDR and NetworkCIDR fields.

A new warewulfd API endpoint at /overlay-file/{overlay}/{path...}?render={id} supports fetching (and rendering) arbitrary overlay files.

There are smaller changes to the server, as well:

- wwctl configure ssh now generates ed25519 keys by default.

103.18 DHCP Server

The Warewulf server's external DHCP service now more flexibly accounts for the presence or absence of an address range. wwctl configure dhcp now generates a DHCP configuration without a defined range, generating as much of the subnet and range definition as possible, for either a "default" configuration or a "static" configuration.

103.19 For Warewulf Developers

Finally, there are a number of changes that really only matter to Warewulf developers:

The minimum Go version is now 1.22.9, as required by updated dependencies.

Warewulf v4.6.0 includes a significant refactor of the internal datastructures that represent cluster nodes. The NodeInfo structure (in-memory-only) has been merged with NodeConf, the YAML-backed data structure. In its place, a new Field system supports tracks the source of node fields while values are merged from profiles for use explicitly during wwctl node list --all.

The primary Warewulf Makefile has been enhanced with target help: just run make to see a list and descriptions of notable targets.

The official Warewulf RPM spec file has been updated to recommend the installation of ipmitool. It also simplifies the permissions of installed files, and omits the gRPC API by default.

The GitHub CI process now runs "staticcheck," and problems highlighted by it have been resolved. Recent problems in the nightly build workflow have also been resolved.

A Visual Studio Code "development container" definition is now included in the repository.

CHAPTER

FOUR

V4.6.1 RELEASE NOTES

v4.6.1 is a regularly-scheduled minor release in the v4.6.x series.

Significant changes are described below. Additional changes are listed in the CHANGELOG.

104.1 aarch64 packages

We're now building and publishing aarch64 packages with GitHub releases.

104.2 New REST API

Possibly the largest change in v4.6.1 is the addition of a new *REST API*. This API is optionally served by warewulfd at /api, and is disabled by default.

104.3 Changed JSON output

The REST API returns JSON, which has brought additional attention to the JSON output previously only output by wwctl <node|profile> list --json. This output has been updated such that the JSON object properties match field names used in nodes.conf. JSON output also now omits empty or unspecified values.

104.4 New command-line options

A couple new wwctl options have been added:

- wwctl overlay import --overwrite overwrites existing overlay files during import.
- wwctl node import --yes skips the interactive confirmation dialog and assumes a "yes" answer.

104.5 New tags

A new vlan IPMI tag configures the IPMI interface to use the specified vlan during ipmiwrite.

```
wwctl profile set default --ipmitagadd=vlan=100
```

104.6 Customize wwclient connection to Warewulf server

In some circumstances (e.g., when isolating compute nodes from the provisioning network) it may be useful for wwclient to connect to a different IP address than the default Warewulf server address. To support this, wwclient will connect to the address specified in the environment variable WW_IPADDR, if present.

This variable can be specified in /etc/default/wwclient with an overlay.

104.7 Distribution-specific fixes

Downstream SUSE packages have historically set a kernel argument net.ifnames=1 to enable predictable network interfaces. This argument is now set in the initial nodes.conf and during wwctl upgrade nodes --add-defaults.

104.8 Upgrade fixes and default behaviors

v4.6.0 moved default settings from defaults.conf (and, before that, compiled-in constants) with settings on the "default" profile in the initial nodes.conf, and added wwctl upgrade nodes --add-defaults to add default settings to an existing nodes.conf. However, this removal of default settings caused surprising deficiencies in some cases; and, in others --add-defaults simply failed to set proper defaults.

- wwctl upgrade nodes --add-defaults now sets a default iPXE template. If no default template is set, warewulfd looks for a template named "default".
- If no init is specified, wwinit now looks for /sbin/init, /etc/init, and /bin/init.
- wwctl upgrade nodes --replace-overlays now avoids adding the same overlay multiple times to the same node or profile.
- If, during wwctl overlay build, a node has an empty system or runtime overlay, a warning is printed.

104.9 New SOS plugin

A new warewulf-sos subpackage installs an SOS plugin to gather logs and other state from the Warewulf server for filing a support request. Currently only available in Enterprise Linux distributions.

```
sos report --enable-plugins warewulf
```

104.10 Misc. bug fixes

- Fixed a panic in warewulfd when a network device isn't configured explicitly on a node.
- wwctl <power|node console|node sensors> now better handles missing required IPMI fields.
- Field names containing periods (e.g., NetDev[eth0.100]) are now displayed properly in wwctl <node|profile> list.
- Properly quote a specified escape character during ipmitool commands.
- Update related nodes and profiles when renaming an image.
- wwctl <node|profile> set now properly handles "UNDEF" and "UNSET" to remove a defined field value.
- Configure the GRUB bootloader to sleep and reboot on certain errors (to allow error messages to be read).
- wwctl node import can now import new nodes, not just update existing nodes.
- Fixed a panic during wwctl node list --ipmi for nodes with no IPMI configuration.
- Fixed processing of --verbose and --debug when starting warewulfd.

CHAPTER

FIVE

V4.6.2 RELEASE NOTES

v4.6.2 is a regularly-scheduled minor release in the v4.6.x series. It also includes a preview implementation of the ability to provision the OS image to a local disk.

Significant changes are described below. Additional changes are listed in the CHANGELOG.

105.1 Provisioning to disk

v4.6.2 includes a preview implementation of the ability to provision the node image to a local disk. This includes the ability to provision disks earlier in the boot process, primarily during the first phase of a two-phase boot using dracut.

Disks may be provisioned using ignition, as before; but may now also be provisioned using new sfdisk, mkfs, and mkswap overlays, and to mount configured file systems using systemd.mount and systemd.swap overlays. These overlays may also each be configured using overlay-specific "resources", which may provide additional control compared to Warewulf's built-in disk, partition, and file system configuration.

To support more complex partition layouts, wwctl <node|profile> set --parttype has been added to specify non-default partition types. This is particularly useful when creating an EFI partition, which may be used in a future release to configure serverless local booting.

Finally, wwctl profile set has been fixed to now properly support configuration of disks, partitions and file systems.

For more information, see the *Provision to disk* section.

105.2 REST API

v4.6.2 continues development of the REST API, properly handling attempts to create a node that already exists, and fetching uid, gid, and mode permissions for overlay files. The previous gRPC-based API servers and client have also been removed.

105.3 Other fixes

- NetworkManager now waits for interfaces to come online before provisioning with Dracut.
- The udev.netname overlay now properly handles InfiniBand interfaces in more cases.
- Fixed a regression in SELinux support by restoring the /run mount during wwinit.
- GRUB now properly reports errors and reboots when an error occurs.
- Fixed IPMI VLAN configuration.
- Preserve existing permissions during wwctl overlay edit.
- Fix CSV processing during wwctl node import --csv.

CHAPTER

SIX

V4.6.3 RELEASE NOTES

v4.6.3 is a regularly-scheduled minor release in the v4.6.x series.

Significant changes are described below. Additional changes are listed in the CHANGELOG.

106.1 REST API

v4.6.3 continues development of the REST API, adding support for the If-None-Match: * header to prevent overwriting an existing entity.

Other new functionality in the REST API includes:

- GET /api/nodes/{id}/overlays returns built time metadata for a node's system and runtime overlay images.
- PUT /api/overlays/{name}/file?path={path} creates files in overlays.
- DELETE /api/overlays/{name}/file?path={path} delete files in overlays.
- DELETE /api/overlays/{name}?force=true deletes overlays that are in use.

106.2 IPv6 Support

IPv6 is now supported during the iPXE network boot process, one step towards improving overall IPv6 support.

106.3 Overlay Autobuild

The Warewulf server (when "autobuild overlays" is enabled) now automatically rebuilds overlays after node discovery, which resolves an issue where overlays were out-of-sync with the newly-discovered configuration. This is one step in a more general effort to improve overlay autobuild so that manual overlay builds are less often required.

106.4 Warewulf server configuration

The wwctl configure command can now enable and (re)start the warewulf server daemon itself to apply updated configuration, either as part of wwctl configure -a or wwctl configure warewulfd.

106.5 Ubuntu and Debian Support

The netplan overlay is now much more feature-complete, with relative parity to the other network configuration overlays. This is one step in a more general effort to support Warewulf in Ubuntu and Debian environments.

106.6 Other fixes

- Fixed wwctl upgrade nodes to properly handle kernel argument lists.
- Fixed a panic during wwctl overlay edit due to missing reexec.Init().
- Fixed handling of comma-separated mount options in fstab and ignition overlays.
- Fixed a race condition in wwctl overlay edit that led to changes not being properly detected and applied.

CHAPTER

SEVEN

V4.6.4 RELEASE NOTES

v4.6.4 is a regularly-scheduled minor release in the v4.6.x series.

Significant changes are described below. Additional changes are listed in the CHANGELOG.

107.1 EL10 and dnsmasq support

To support the ongoing development and upcoming release of OpenHPC 4, Warewulf v4.6.4 adds initial support and automatic CI/CD builds for EL10. EL10 does not include the ISC DHCP server, so EL10 packages depend on and use dnsmasq by default.

Warewulf v4 has included some support for dnsmasq since v4.4.0, but its use by default in EL10 will bring additional visibility. For now, this includes:

- A fix for a bug that prevented proper handling of iPXE files when using dnsmasq.

107.2 wwclient

v4.6.4 makes a number of enhancements to `wwclient`, the daemon that periodically re-applies the runtime overlay to cluster nodes.

- `wwclient` now places files from the runtime overlay atomically, such that any application concurrently reading an updated file will only ever see the complete previous or updated content.
- `wwclient` now skips updating files that do not appear to have been modified, as judged by file size and modification time.
- `wwclient --once` prompts wwclient to run once, which is useful both for testing and for dispatching updates more dynamically (e.g., between compute jobs).
- `wwclient --debug` now generates more and more useful debug output.

107.3 wwctl configure warewulfd

v4.6.4 adds the ability to enable and (re)start the warewulf server daemon itself to apply updated configuration, either as part of `wwctl configure -a` or `wwctl configure warewulfd`. This only takes effect if a `systemd name` for Warewulf is specified in `warewulf.conf`.

```
warewulf:
  systemd name: warewulfd
```

New installations will include `systemd name: warewulfd` by default.

107.4 Network configuration fixes

- The NetworkManager overlay now omits the [ethernet] section for non-ethernet interfaces.
- The NetworkManager overlay now sets ipv4:method=disabled if no address or route is specified.
- The ifcfg overlay now sets SLAVE=yes when MASTER is set. #1980

107.5 Other fixes

- Prevent brp-mangle-shebangs from changing files in overlays. (e.g., /bin/sh to /usr/bin/sh)
- Updated golang BuildRequires in the RPM specfile to 1.22.
- Fixed unsetting boolean options with wwctl.

CHAPTER

EIGHT

V4.6.5 RELEASE NOTES

v4.6.5 is a regularly-scheduled minor release in the v4.6.x series.

Significant changes are described below. Additional changes are listed in the CHANGELOG.

108.1 wwctl overlay info

A new wwctl overlay info command lists variables used by an overlay template, including the wwctl <node|profile> set argument used to set that variable.

```
# wwctl overlay info ifcfg /etc/sysconfig/network-scripts/ifcfg.ww
VARIABLE                OPTION      TYPE                    HELP
--------                ------      ----                    ----
$netdev.Device          --netdev    string                  Set the device for given network
$netdev.Gateway         --gateway   IP                      Set the node's IPv4 network device gateway
$netdev.Hwaddr          --hwaddr    MAC                     Set the device's HW address for given network
$netdev.Ipaddr          --ipaddr    IP                      IPv4 address in given network
$netdev.Ipaddr6         --ipaddr6   IP                      IPv6 address in given network
$netdev.MTU             --mtu       uint                    Set the mtu
$netdev.Netmask         --netmask   IP                      Set the networks netmask
$netdev.OnBoot          --onboot    wwtype.WWbool           Enable/disable network device (true/false)
$netdev.Tags                        map[string]string
$netdev.Tags.DNSSEARCH              string
$netdev.Tags.master                 string
$netdev.Type            --type      string                  Set device type of given network
.NetDevs                            map[string]*node.NetDev
```

Other variables and the overlay itself can also be documented explicitly within a template.

```
{{/* wwdoc: Configures chronyd to synchronize time with a specific time server. */}}
{{/* .Tags.ntp_server: the NTP server to synchronize with */}}
```

108.2 wwctl image import –update

The wwctl image import command has included an --update option since v4.6.0, but it has not been functional. The option now imports an image on-top-of an existing image and is meant to be used to update an existing image with new files from an updated upstream or external image (e.g., by importing a new version of an image from an OCI registry).

108.3 wwclient

One of the last real impediments to multi-architecture support in Warewulf has been the wwclient daemon which, though distributed to cluster nodes, has been built only for the architecture of the Warewulf server. Now, new wwclient.aarch64 and wwclient.x86_64 overlays provide architecture-specific builds of wwclient for aarch64 and x86_64 nodes, respectively.

wwclient has also been enhanced with IPv6 support, and now properly handles recently-used addresses when restarting.

108.4 Network management

A number of network management improvements have been made. Most deal with improving IPv6 support. The debian.interfaces overlay has been renamed to ifupdown to better reflect its support for both Debian, Ubuntu, and Alpine systems. The netplan and NetworkManager overlay have improved support for bonded interfaces; and a new systemd-networkd overlay adds support for the systemd networking daemon.

108.5 Packaging

The openEuler 24.03 RPM package now uses dnsmasq by default.

108.6 Development

Work has resumed on support for using Vagrant to initialize a basic testing / development environment with libvirt.

108.7 warewulf-dracut

A bug between warewulf-dracut and the ignition overlay previously prevented non-root filesystems from mounting when provisioning the image to disk. A fix in warewulf-dracut prevents ignition from running twice, allowing all file systems to mount properly. The wwinit dracut module also no longer runs the wwinit module if root is not set to root=wwclient*.

These fixes require installing the updated warewulf-dracut package in the OS image.

108.8 Other fixes

- Enhanced wwctl configure tftp to manage the SELinux context of TFTP directory. #1997
- Enhanced overlay templates to support absolute paths with {{ file }}. #2055
- Fix ImageDelete API not returning error when checking if image is used by nodes/profiles. #1705
- Fix filesystem overwrite/force behavior in mkfs overlay. #2028
- Write $tftpdir/warewulf/grub.cfg to tftproot as configured in warewulf.conf. #2055
- Automatically create a GPT label when sfdisk overlay wipes disks. #2025

CHAPTER

NINE

V4.7.0 RELEASE NOTES

v4.7.0 is a significant upgrade, with many changes relative to the v4.6.x series.

Particularly significant changes, especially those affecting the user interface, are described below. Additional changes are listed in the CHANGELOG.

Running wwctl upgrade should be sufficient when upgrading from v4.6.x and previous releases, but back up your configuration (nodes.conf and warewulf.conf) when upgrading.

109.1 Documentation

The user documentation has received a substantial audit and update for v4.7.0:

- An initial one-page quick reference guide.
- New documentation describing the status of various top-level Warewulf features.
- New documentation for the wwctl clean command.
- New documentation on reclaiming RAM by moving the image to swap.
- New documentation on configuring the ARP cache for large clusters.
- Expanded troubleshooting documentation for container runtimes.
- New documentation on detecting shadowed overlays.
- A new section on multiple-network server configurations, moved from getting-started to a dedicated server networking section.
- Audit and corrections to documentation, cobra help text, and log messages.
- Clarified functionality of syncuser commands and the syncuser overlay.
- The documentation now distinguishes more clearly between OS images and overlay images.
- A new troubleshooting section describes how to detect shadowed overlays.

109.2 wwctl unset

The wwctl set command was previously enhanced with automatic type-checking to ensure that input field values were valid for their target field. However, this broke the ability to unset some fields with a sentinel UNDEF value (e.g., because UNDEF is not a valid IP address).

A new wwctl <node|profile> unset command provides explicit removal of individual configuration fields, replacing the use of empty-string assignment or --del flags on wwctl <node|profile> set. The set and unset commands are now implemented in a unified fashion using cobra's cmd.Flags().Changed() to apply only explicitly-set fields.

wwctl <node|profile> set --partdel and wwctl <node|profile> unset --partname now scope partition deletion to a specific disk when --diskname is provided; without --diskname, the partition is removed from all disks (the previous behavior). A new --partwipe flag has also been added to wwctl <node|profile> set.

109.3 Refactored Server Routes

The warewulfd HTTP API has been refactored to use a set of dedicated, purpose-specific server routes in place of the previous /provision/{hwaddr}?stage=X interface. The new routes are:

- /ipxe/ for iPXE scripts
- /kernel/ for kernel images
- /image/ for node images
- /initramfs/ for initramfs images
- /system/ for system overlays
- /runtime/ for runtime overlays
- /grub/ for GRUB configuration
- /efiboot/ for EFI boot files

The iPXE, GRUB, and dracut boot scripts and templates have been updated to use the new routes, and wwclient now uses the /runtime/ route for runtime overlay downloads. A new wwinit.server kernel parameter is used for dracut/wwinit boot. (The previous wwinit.uri parameter, as well as the /provision/ route, remains supported for backward compatibility.)

109.4 New Files Route

A new /files/ server route serves static files from the Warewulf files directory (typically /var/lib/warewulf/files/). The route supports optional template rendering and an optional .ww suffix, allowing both raw and template-rendered files to be served.

A new warewulf:secure files configuration option in warewulf.conf controls whether the /files/ route requires requests from a privileged port, independently of the global warewulf:secure setting.

109.5 TLS Support

Warewulf v4.7.0 adds TLS support to warewulfd and the REST API. A new wwctl configure tls command generates and configures TLS keys and certificates for the Warewulf server.

109.6 Removed Sub-Overlay Support

The /overlay-file/ route, the ?overlay= query parameter on /system/ and /runtime/, and the --overlay (-O) and --output (-o) flags on wwctl overlay build have all been removed.

109.7 Internal gRPC Removal

The gRPC API libraries and protobuf types have been removed from Warewulf. The node, profile, and image functions previously implemented via gRPC have been refactored to use the same internal interfaces used by the REST API, simplifying the codebase and reducing the number of dependencies.

109.8 Network Overlays

- New chrony overlay to configure chronyd.
- The netplan overlay can now configure now emits default routes.
- The syncuser overlay now supports defining explicit local users and groups.
- A new mig overlay configures NVIDIA MIG devices.
- The hosts overlay has been added to the default system overlay list.

109.9 Template Functions and Whitespace

The {{ file }}, {{ softlink }}, and {{ ImportLink }} template functions have been refactored to use state-based routing instead of sentinel strings. Whitespace-trimming syntax (e.g., {{- file "name" -}}) now correctly creates all named files and symlinks.

109.10 Security

Multiple theoretical path-traversal vulnerabilities in overlay handling have been fixed. (These vulnerabilities are not believed to have been exploitable, but have been addressed as a matter of best practice.)

The assetkey field is no longer leaked into wwclient logs.

The minimum Go version has been increased to 1.25.5 to pick up multiple stdlib CVE fixes, and many other library dependencies have been updated. See CHANGELOG for the full list.

109.11 Development and CI

- A new CI integration test provisions QEMU VMs via Warewulf on Rocky Linux 10, with matrix builds adding SUSE Linux 16.0 (Leap) coverage.
- The userdocs CI job has been sped up.

109.12 Other Fixes

- Include API configuration during wwctl upgrade config.
- The comment field is no longer inherited from profiles by nodes.
- Fix the requisite dependency between the ignition disk target and the ignition service.
- Allow whitespace to be trimmed for wwdoc comments.
- Improved error handling for the /newroot mount during single-stage boot, and bugfixes for command-line arguments during single-stage image unpacking.
- Fixed incorrect help docs for wwctl overlay chown.
- Raised the open file limit for the warewulfd service to match systemd precedence.
- A missing goto has been added to default.ipxe to fix an iPXE menu fallthrough bug.
- The Warewulf REST API now returns HTTP 409 when attempting to create an overlay that already exists.
- Runtime overlay download failure during dracut/wwinit boot is now non-fatal: the node continues to boot and wwclient retries the download at runtime.

V4.7.1 RELEASE NOTES

v4.7.1 is a bug-fix release in the v4.7.x series.

Significant changes are described below. Additional changes are listed in the CHANGELOG.

110.1 wwctl power could act on the Warewulf server

wwctl power (cycle, off, on, reset, soft, and status) ran ipmitool without a -H argument for any node with no ipmi: ipaddr: configured. Without an explicit host, ipmitool falls back to the local BMC, so a command intended for a node could power off or reset the Warewulf server itself.

Such nodes are now skipped, and the command exits non-zero.

110.2 wwclient could fail with an illegal instruction

wwclient is distributed as a static binary built on the Warewulf server, but was compiled without pinning GOAMD64. The Go toolchain selected a microarchitecture level based on the build host, and the resulting binary could fail with an illegal instruction on older nodes.

wwclient is now built with GOAMD64=v1 and runs on any 64-bit x86 processor.

110.3 Image and overlay corruption with large inode numbers

When building node images and overlays on a file system with 64-bit inode numbers, cpio truncated those numbers to 32 bits. Distinct files could collide on the truncated value and be written as hardlinks to each other, silently corrupting the resulting image.

Warewulf now passes --renumber-inodes to cpio where the installed version supports it (GNU cpio 2.13 and later).

110.4 Kernel version detection

Kernel versions whose RPM release field contains a version-like suffix after the dist tag, such as 5.14.0-687.10.1.el9_8.0.1, were mis-detected: Warewulf reported 8.0.1 rather than 5.14.0-687.10.1, because ParseVersion returned the last regular expression match found in the kernel path. This also caused the wrong kernel to be selected as the default. #2199

110.5 IPv6-only servers and nodes

Several independent defects prevented Warewulf from operating on IPv6-only deployments.

The two-stage dracut boot rendered ip=<device>:dhcp for every device, which NetworkManager's initrd generator turns into a mandatory DHCPv4 lease. The connection never activated and the initramfs never fetched the image. A device with ipaddr6 and no ipaddr now renders ip=<device>:auto6.

wwctl node status now falls back to ipaddr6 when ipaddr is unset, brackets IPv6 literals in the status URL, and bounds the request with a timeout.

The IPv4-only sections in the dnsmasq host overlay template are now guarded on ipaddr. This prevents an erroneous empty listen-address= line, the IPv4 boot block with blank http:// boot URLs, and an invalid dhcp-range=,static, ,6h.

The NetworkManager overlay no longer generates invalid [ipv4] sections for IPv6-only interfaces. (gateway=, dns=, dns-search=, and route lines were previously emitted even under method=disabled.)

110.6 DSA host keys removed from the defaults

dsa has been removed from the default ssh: key types in warewulf.conf. On EL9 and other systems with OpenSSH 8.7p1 or later, sshd silently skips DSA host keys, which could leave nodes with no usable host keys at all. #1185

Sites that have customized ssh: key types in warewulf.conf are unaffected.

110.7 provision-to-disk filesystem mounting

Non-root filesystems are now mounted before the image is extracted during provision-to-disk, so files are written to the correct partitions rather than all landing on the root filesystem. #2147

This fix adds a /warewulf/mounts file to the wwinit overlay and a corresponding step to the wwinit dracut module. **Nodes must rebuild their initramfs** with the updated warewulf-dracut package for it to take effect.

110.8 wwclient.service

The wwclient systemd unit dropped the obsolete PIDFile directive, which has no meaning for a Type=notify service, and replaced TimeoutSec=60 with TimeoutStartSec=infinity, TimeoutStopSec=90, Restart=on-failure and StartLimitIntervalSec=0 to accommodate wwclient's own retry logic. #2203

A node that cannot reach the Warewulf server now continues retrying rather than failing the unit after 60 seconds.

110.9 Other fixes

- wwctl node import now reloads warewulfd, matching every other mutating wwctl node command. Previously imported nodes were invisible to wwctl node status, PXE, and wwclient until a reload or restart.
- wwctl profile set --nettagadd no longer silently ignores the given tags when the network device already exists.
- Overlay templates that create multiple files now create the parent directories for those files if they do not yet exist.
- Overlay templates that create symbolic links now replace an existing link, and an existing file, rather than failing. #2252
- Fix MTU configuration with the wicked overlay. #2248
- curl is no longer silenced when downloading with dracut, making boot failures easier to diagnose. #2200
- Fix the example genders.ww template, which still used a .Get accessor API that no longer exists.

110.10 Dependencies

Dependencies have been updated, including go-jose 4.1.4 to address CVE-2026-34986. See the CHANGELOG for the full list.