

---

# **Warewulf User Guide**

***Release 4.6.5***

**Warewulf Project Contributors**

**Feb 23, 2026**



## GETTING STARTED

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Cluster Provisioning</b>	<b>5</b>
<b>3</b>	<b>Network Planning</b>	<b>7</b>
<b>4</b>	<b>Enterprise Linux Quickstart</b>	<b>11</b>
<b>5</b>	<b>SUSE Quickstart</b>	<b>15</b>
<b>6</b>	<b>Debian Quickstart</b>	<b>19</b>
<b>7</b>	<b>Glossary</b>	<b>23</b>
<b>8</b>	<b>Server Installation</b>	<b>25</b>
<b>9</b>	<b>Controlling Warewulf</b>	<b>29</b>
<b>10</b>	<b>Server Configuration</b>	<b>31</b>
<b>11</b>	<b>Using dnsmasq</b>	<b>37</b>
<b>12</b>	<b>Security</b>	<b>39</b>
<b>13</b>	<b>Bootloaders</b>	<b>41</b>
<b>14</b>	<b>Upgrading Warewulf</b>	<b>47</b>
<b>15</b>	<b>REST API</b>	<b>49</b>
<b>16</b>	<b>Cluster Nodes</b>	<b>51</b>
<b>17</b>	<b>Node Profiles</b>	<b>57</b>
<b>18</b>	<b>Network Interfaces</b>	<b>61</b>
<b>19</b>	<b>IPMI</b>	<b>63</b>
<b>20</b>	<b>Provisioning disks</b>	<b>67</b>
<b>21</b>	<b>Node Images</b>	<b>71</b>
<b>22</b>	<b>Image Kernels</b>	<b>79</b>

23 Syncuser	81
24 SELinux-enabled Images	83
25 Overlays	85
26 Templates	95
27 Troubleshooting	101
28 Known issues	105
29 Contributing	107
30 Development Environment	109
31 Documentation	111
32 Debugging	113
33 wwctl	115
34 wwctl clean	117
35 wwctl configure	119
36 wwctl configure dhcp	121
37 wwctl configure hostfile	123
38 wwctl configure nfs	125
39 wwctl configure ssh	127
40 wwctl configure tftp	129
41 wwctl configure warewulfd	131
42 wwctl image	133
43 wwctl image build	135
44 wwctl image copy	137
45 wwctl image delete	139
46 wwctl image exec	141
47 wwctl image import	143
48 wwctl image kernels	145
49 wwctl image list	147
50 wwctl image rename	149
51 wwctl image shell	151
52 wwctl image show	153

53	wwctl image syncuser	155
54	wwctl node	157
55	wwctl node add	159
56	wwctl node console	161
57	wwctl node delete	163
58	wwctl node edit	165
59	wwctl node export	167
60	wwctl node import	169
61	wwctl node list	171
62	wwctl node sensors	173
63	wwctl node set	175
64	wwctl node status	179
65	wwctl overlay	181
66	wwctl overlay build	183
67	wwctl overlay chmod	185
68	wwctl overlay chown	187
69	wwctl overlay create	189
70	wwctl overlay delete	191
71	wwctl overlay edit	193
72	wwctl overlay import	195
73	wwctl overlay info	197
74	wwctl overlay list	199
75	wwctl overlay mkdir	201
76	wwctl overlay show	203
77	wwctl power	205
78	wwctl power cycle	207
79	wwctl power off	209
80	wwctl power on	211
81	wwctl power reset	213
82	wwctl power soft	215

83	wwctl power status	217
84	wwctl profile	219
85	wwctl profile add	221
86	wwctl profile delete	223
87	wwctl profile edit	225
88	wwctl profile list	227
89	wwctl profile set	229
90	wwctl server	231
91	wwctl ssh	233
92	wwctl upgrade	235
93	wwctl upgrade config	237
94	wwctl upgrade nodes	239
95	wwctl version	241
96	v4.6.0 Release Notes	243
97	v4.6.1 Release Notes	251
98	v4.6.2 Release Notes	255
99	v4.6.3 Release Notes	257
100	v4.6.4 Release Notes	259
101	v4.6.5 Release Notes	261

Welcome to the Warewulf User Guide!





## 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 node images which are used to provision out to the cluster nodes. This means you can have hundreds or thousands of cluster nodes all booting and running on the same node image. As of Warewulf v4, the node 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 principals 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 support SELinux out-of-the-box. Just install SELinux in your node 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.

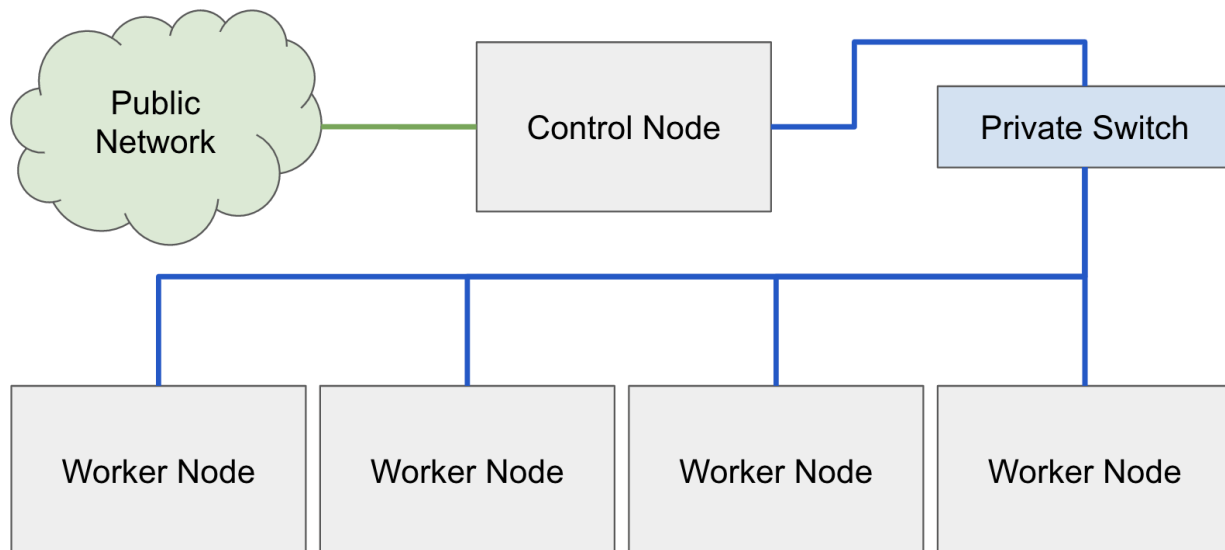
**Node profiles** provide a flexible abstraction for applying configuration to multiple nodes.

**Node images** provide a bootable operating system image, including the kernel that will be used to boot the cluster node. Node images provide a 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 node 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.

## 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 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.

## **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

## 3.2 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 first 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 host in the delivery network and an additional tags is used for 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.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)
```

(continues on next page)

(continued from previous page)

```

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 }}
{{- end }}

```

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





## 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 the RPMs published in [GitHub releases](#). For example, to install the v4.6.5 release on Enterprise Linux 9:

```
dnf install https://github.com/warewulf/warewulf/releases/download/v4.6.5/warewulf-4.
↳ 6.5-1.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
  tftpboot: /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
```

(continues on next page)

(continued from previous page)

```

    readonly: true
paths:
    bindir: /usr/bin
    sysconfdir: /etc
    localstatedir: /var/lib
    ipxesource: /usr/share/ipxe
    srvdir: /var/lib
    firewallddir: /usr/lib/firewalld/services
    systemdmdir: /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 `q`tftpboot``.

## 4.6 Add a base node image

This will pull a basic node 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!

## 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/tftp 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
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 individually (omitting the `--all`) will print a help and usage instructions.

**Note:** If the `dhcpcd` 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/dhpcd`.

```
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!



## 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 'v4.6.5'
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 individually (omitting the `--all`) will print a 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!

## 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** The node images that Warewulf manages and provisions. Images may be imported from OCI image registries, OCI image archives, Apptainer sandboxes, and manual chroot directories.

Warewulf images are maintained as an uncompressed “virtual node file system” (VNFS, sometimes also referred to as a “chroot”). These virtual file systems are then built as single-file images which may be used to provision a node.

**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.”

**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.

**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.

## SERVER INSTALLATION

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

### 8.1 Binary RPMs

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

#### 8.1.1 Rocky Linux 9

```
# dnf install https://github.com/warewulf/warewulf/releases/download/v4.6.5/warewulf-4.6.5-1.el9.x86_64.rpm
```

#### 8.1.2 openSuse Leap

```
# zypper install https://github.com/warewulf/warewulf/releases/download/v4.6.5/warewulf-4.6.5-1.suse.lp155.x86_64.rpm
```

### 8.2 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](#)

#### 8.2.1 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
```

## 8.2.2 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
# 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
```

## 8.3 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.

### 8.3.1 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/v4.6.5/warewulf-4.6.5.
↪tar.gz
tar -xf warewulf-4.6.5.tar.gz
cd warewulf-4.6.5
make all && sudo make install
```

### 8.3.2 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 'v4.6.5'
make all && sudo make install
```



### 8.3.3 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`.

## 8.4 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
```

## 8.5 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
```



## CONTROLLING WAREWULF

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

- `configure`: configures the Warewulf server and its external services
- `node`: manages nodes in the cluster
- `profiles`: defines common sets of node configuration which can be applied to multiple nodes
- `image`: configures (node) images
- `overlays`: manages overlays

`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`.

### 9.1 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`

### 9.2 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 4 different stages:

- **IPXE**
- **KERNEL**
- **SYSTEM\_OVERLAY**
- **RUNTIME\_OVERLAY**

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

## SERVER CONFIGURATION

By default, the Warewulf server configuration is located at `/etc/warewulf/warewulf.conf`. This is a YAML-formatted configuration file used by to configured 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
  tftpboot: /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
    - rsa
    - dsa
image mounts:
```

(continues on next page)

(continued from previous page)

```

- 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
  firewallddir: /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

```

## 10.1 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: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.

## 10.2 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.

## 10.3 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
  tftpboot: /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:tftpboot`: 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.

## 10.4 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.

## 10.5 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
    - dsa
```

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

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



## 10.6 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.

## 10.7 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`.
- `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.

## 10.8 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.

## 10.9 api

*New in Warewulf v4.6.1*

Configuration for the REST API of the `warewulfd` service.

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

- `api:enabled`: Whether the `warewulfd` service should provide access via a REST interface.
- `api:allowed subnets`: Which subnets are allowed to access the REST API. By default, only localhost has access.

## 10.10 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.

## 10.11 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
```

## 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 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.

### 11.1 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
```



## 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.

### 12.1 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 node 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 compiled modules can’t be loaded.

### 12.2 SELinux

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

For more information about running SELinux-enabled cluster node images, see [SELinux-Enabled Images](#).

## 12.3 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.

---

## 12.4 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
```

## 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.

### 13.1 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. In stead, 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.

#### 13.1.1 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 -l /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.ipxsource` 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:
    ipxsource: /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
```

## 13.2 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.

### 13.2.1 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.

### 13.2.2 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.

### 13.2.3 HTTP boot

Modern EFI systems have the possibility to directly boot per http. 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.

## 13.3 Two-stage boot: dracut

Some systems, typically due to limitations in their BIOS or EFI firmware, are unable to load 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.

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/v4.6.5/warewulf-dracut-4.6.5-1.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/v4.6.5/warewulf-dracut-4.6.5-1.suse.lp155.
↳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.)



## 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.



## REST API

On-line documentation for the API is available at `/api/docs`.

### 15.1 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/16HnbNOLoXgY2sa1bQQ7aSBKdSqvsC
```

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

```
$ mkpasswd --method=bcrypt
Password: # admin
$2b$05$5QVWDpiWE7L4SDL9CYdi3O/16HnbNOLoXgY2sa1bQQ7aSBKdSqvsC
```

### 15.2 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

## 15.3 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

## 15.4 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

## 15.5 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



## 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 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
```

### 16.1 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>    --
```

## 16.2 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  wwininit,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"}]
```

## 16.3 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`.

### 16.3.1 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"'
```

### 16.3.2 Un-setting Node Fields

To un-set a field value, set the value to UNDEF.

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

## 16.4 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*.

## 16.5 Configuring the Network

By default, network configurations are applied to a “default” network interface.

```
wwctl node set n1 \
  --netdev=enol \
  --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*.

## 16.6 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.

## 16.7 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"
```

## 16.8 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`.

## 16.9 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
```

(continues on next page)

(continued from previous page)

```
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
```



## 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.

### 17.1 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  wwininit,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"}]

```

## 17.2 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

```

## 17.3 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"

```

**Note:** If two profiles set the same field, the right-most profile in the node's list takes precedence. Field values set directly on nodes take precedence over profile field values.

```

# 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
n1    SystemOverlay                 default  wwininit,wwclient,fstab,hostname,ssh.host_keys,
↪ issue,resolv,udev.netname,systemd.netname,ifcfg,ifupdown,NetworkManager,wicked,
↪ ignition

```

(continues on next page)



(continued from previous page)

```

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.

## 17.4 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.

## 17.5 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.



## NETWORK INTERFACES

By default, network configurations are applied to a “default” network interface.

```
wwctl node set n1 \  
  --netdev=enol \  
  --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 \  
  --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., `enol`), 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.

### 18.1 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"
```

### 18.2 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:  
    type: Bond  
    device: bond0
```

(continues on next page)

(continued from previous page)

```
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
```

## 18.3 VLAN

You can set the type also to `vlan`.

Some network configuration systems use the network device name (e.g., of the form `en01.100`) 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
```

## 18.4 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"
```

## IPMI

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

### 19.1 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=passwd0rd \  
  --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

## 19.2 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]`.

## 19.3 Console

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

```
# wwctl node console n1
```

## 19.4 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`





## 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.

---

**Note:** Warewulf is not currently able to provision the node image onto an explicitly provisioned root file system.

---

### 20.1 Requirements

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

#### 20.1.1 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.”

---

#### 20.1.2 openSuse Leap

```
zypper install ignition gptfdisk
```

### 20.2 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. 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 --partnumber 1
```

## 20.3 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
```

## 20.4 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.

## 20.5 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
```

## 20.6 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`<sup>1</sup>; or
- set the `--fswipe` flag and remove it after one reboot.

See the [upstream ignition documentation](#) for additional information.

## 20.7 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 `wipeFilesystem` 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.

### 20.7.1 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.

<sup>1</sup> 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.

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

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

## 20.7.2 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 wwininit \  
--install sfdisk \  
--install blockdev \  
--install udevadm \  
--install mkfs \  
--install mkfs.ext4 \  
--install wipefs \  
--regenerate-all
```

## 20.7.3 Configuring the root device

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

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

## NODE IMAGES

Warewulf node images are a “Virtual Node File System” (VNFS) that serves as a base 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 node 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.

### 21.1 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
```

## 21.2 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.

### 21.2.1 OCI Registry

You can import node 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 a base image from DockerHub (e.g. `docker://rockylinux` or `docker://debian`) to boot properly with Warewulf.

The Warewulf project maintains a set of [example node 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 image.

---

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 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 wil show up in the process table.

### 21.2.2 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
```

### 21.2.3 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
```

## 21.3 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
```

## 21.4 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!
```

### 21.4.1 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.

### 21.4.2 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:/$
```



### 21.4.3 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.

### 21.4.4 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
```

## 21.5 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
```

### 21.5.1 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`.

### 21.5.2 Exit Script

Warewulf executes the script `/etc/warewulf/image_exit.sh` in the image after a `wwctl image shell` or `wwctl image exec` and prior to (re)building the final node image for delivery. 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.

## 21.6 Defining New Images

It is absolutely possible to import a base 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) ecoystems.

### 21.6.1 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
```

### 21.6.2 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 --allowdowngrading \
    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
[...]
```

(continues on next page)

(continued from previous page)

```
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
```

## 21.7 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.kg8ijih5tq4lixoeag4plqp/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
```

## 21.8 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`.

---

## 21.9 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.

---

## 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
```

### 22.1 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
```



## SYNCUSER

Warewulf can optionally synchronize UIDs and GIDs from the Warewulf server to an image. This can be particularly useful when there is no central directory (e.g., an LDAP server).

---

**Note:** Some system services (notably “munge”) require a user to have the same UID across all nodes.

---

Combined with the “syncuser” overlay, Warewulf syncuser also supports defining local users on the Warewulf server for synchronization to cluster nodes.

If there is 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.





## SELINUX-ENABLED IMAGES

Warewulf supports booting SELinux-enabled images, though nodes using SELinux must be configured to use tmpfs for their image file system. (“ramfs,” often used by default, 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 node 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 `initrootfs`. Nodes using SELinux should specify `--root=tmpfs`.



## OVERLAYS

Warewulf supplements provisioned node 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 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
```

### 25.1 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
```

(continues on next page)

(continued from previous page)

```

$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
$netdev.Tags.updelay
$netdev.Tags.vlan_id
$netdev.Tags.xmit_hash_policy
$netdev.Type                    Set device type of given network      string_
↪ --type

```

### 25.1.1 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

```

### 25.1.2 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="wwininit,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.

### 25.1.3 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 node 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.

### 25.1.4 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=nl
```

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.

---

## 25.2 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
```

### 25.2.1 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.

## 25.3 wwininit

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 file 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`.

## 25.4 wwclient

All configured overlays are provisioned initially along with the node 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.

## 25.5 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`

---

**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

## 25.6 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"
```

## 25.7 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
```

## 25.8 ssh

Two SSH overlays configure host keys (one set for all node in the cluster) and `authorized_keys` for the root account.

- `ssh.authorized_keys`
- `ssh.host_keys`

## 25.9 syncuser

The **syncuser** overlay updates `/etc/passwd` and `/etc/group` to include all users on 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 have also been run on the image to synchronize its user and group IDs with those of the server.

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.

## 25.10 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
            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
```

## 25.11 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
```

## 25.12 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"
```

## 25.13 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
      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.

## 25.14 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.

## 25.15 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
```

## 25.16 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.

## 25.17 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`



## 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](https://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`.

---

### 26.1 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.

### 26.2 Template documentation

Templates can include documentation to be included in the output of `wwctl overlay info`.

```
{{/* wwdoc: Your documentation text */}}
```

### 26.3 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`.

## 26.4 Template functions

Warewulf templates have access to a number of functions that assist in creating more dynamic and expressive templates.

### 26.4.1 Default functions

`text/template` includes a number of [default functions](#) that are available during Warewulf template processing.

### 26.4.2 Sprig

Supplementing the default functions, Warewulf templates also have access to [Sprig functions](#).

### 26.4.3 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" }}
```

### 26.4.4 IncludeFrom

Reads content from the given file from the given image into the template.

```
{{ IncludeFrom $.ImageName "/etc/passwd" }}
```

### 26.4.5 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" }}
```

### 26.4.6 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" }}
```

### 26.4.7 basename

Returns the base name of the given path.

```
{{- range $type, $name := $.Tftp.IpxeBinaries }}
  if option architecture-type = {{ $type }} {
    filename "/warewulf/{{ basename $name }}";
  }
{{- end }}
```

### 26.4.8 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.

```
{{- range $devname, $netdev := .NetDevs }}
  {{- $filename := print "ifcfg-" $devname ".conf" }}
  {{ file $filename }}
  {{/* content here */}}
{{- end }}
```

### 26.4.9 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.

### 26.4.10 readlink

Equivalent to `filepath.EvalSymlinks`. Returns the target path of a named symlink.

```
{{ readlink /etc/localtime }}
```

### 26.4.11 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 }}
```

### 26.4.12 abort

Immediately aborts processing the template and does not write a file.

```
{{ abort }}
```

### 26.4.13 nobackup

Disables the creation of a backup file when replacing files with the current template.

```
{{ nobackup }}
```

### 26.4.14 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
}}
```

### 26.4.15 SystemdEscape

Escapes a string for use in a systemd unit file.

Escape rules are documented at [systemd.unit](#).

### 26.4.16 SystemdEscapePath

Escapes a path for use in a systemd unit file.

```
{{ file (print ($fs.path | SystemdEscapePath) ".mount") }}
```

Escape rules are documented at [systemd.unit](#).

## 26.5 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
```



### 26.5.1 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") }}
```



## TROUBLESHOOTING

### 27.1 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.

---

### 27.2 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
```

### 27.3 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 uri http://10.0.0.1:9873/provision/00:00:00:00:00:00
kernel --name kernel ${uri}?stage=kernel
imgextract --name image ${uri}?stage=image&compress=gz
imgextract --name system ${uri}?stage=system&compress=gz
```

(continues on next page)

(continued from previous page)

```
imgextract --name runtime ${uri}?stage=runtime&compress=gz
boot kernel initrd=image initrd=system initrd=runtime
```

- The `uri` variable points to `warewulfd` for future reference. This includes the cluster node's MAC address 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 noe 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 node image as an initial ramdisk, we must use `rdinit=` here.

## 27.4 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`.

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.)

```
uri="(http,10.0.0.1:9873)/provision/${net_default_mac}"
linux "${uri}?stage=kernel" wwid=${net_default_mac}
initrd "${uri}?stage=image&compress=gz" "${uri}?stage=system&compress=gz" "${uri}?
↪stage=runtime&compress=gz"
boot
```

- The `uri` 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 "${uri}?stage=kernel" wwid=${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 node image as an initial ramdisk, we must use `rdinit=` here.

---

## 27.5 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](#).

## 27.6 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.

## 27.7 Running Containers on Cluster Nodes

Some container runtimes, notably Podman, require file system features that are not available in `initrootfs`. Cluster nodes using Podman (and some other container runtimes) should be configured with `--root=tmpfs`.

## 27.8 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 node 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.

---

**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`.

---

## KNOWN ISSUES

### 28.1 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"
```

### 28.2 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/`.

## 28.3 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.
→kg8ijih5tq41lixoeag4p1qup/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.

## 28.4 Image Size Considerations

Node 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 node 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.



## 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!

### 29.1 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!

#### 29.1.1 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/>>.

#### 29.1.2 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!

### 29.2 Raise an Issue

For general bugs/issues, you can open an issue [at the GitHub repo](#).

### 29.3 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.



## 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.)

### 30.1 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
```

## 30.2 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 go lang 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 go lang testing package](#)
- [Table Driven Tests](#)
- [Testtft assert](#)
- [Warewulf testenv](#)

## 30.3 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.

## 30.4 Using Vagrant and Libvirt

Vagrant can be used to quickly spin up a test/development environment for Warewulf. A *Vagrantfile* is provided in `vagrant` directory of the Warewulf repository. See the [README.md](#) for more details.

## 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.



## 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.

### 32.1 Validating the code with vet

The Warewulf Makefile includes a `vet` target which runs `go vet` on the full codebase.

```
make vet
```

### 32.2 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.





Warewulf Control

## 33.1 Synopsis

Control interface to the Warewulf Cluster Provisioning System.

## 33.2 Options

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-h, --help</code>	help <b>for</b> <code>wwctl</code>
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

## 33.3 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



## WWCTL CLEAN

Clean up

### 34.1 Synopsis

This command cleans the OCI cache and removes leftovers from deleted nodes

```
wwctl clean
```

### 34.2 Options

```
-h, --help    help for clean
```

### 34.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 34.4 SEE ALSO

- [\*wwctl\*](#) - Warewulf Control



## WWCTL CONFIGURE

Manage system services

### 35.1 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]
```

### 35.2 Options

```
-a, --all      Configure all services  
-h, --help    help for configure
```

### 35.3 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
```

### 35.4 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 warewulfd* - Enable and start warewulfd



## WWCTL CONFIGURE DHCP

Manage and initialize DHCP

### 36.1 Synopsis

DHCP is a dependent service to Warewulf. This command will configure DHCP as defined in the warewulf.conf file.

```
wwctl configure dhcp [OPTIONS]
```

### 36.2 Options

```
-h, --help    help for dhcp
```

### 36.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 36.4 SEE ALSO

- *wwctl configure* - Manage system services





## WWCTL CONFIGURE HOSTFILE

update hostfile on master

### 37.1 Synopsis

Manage the hostfile on the master node

```
wwctl configure hostfile [OPTIONS]
```

### 37.2 Options

```
-h, --help    help for hostfile
```

### 37.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 37.4 SEE ALSO

- *wwctl configure* - Manage system services



## WWCTL CONFIGURE NFS

Manage and initialize NFS

### 38.1 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]
```

### 38.2 Options

```
-h, --help    help for nfs
```

### 38.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 38.4 SEE ALSO

- *wwctl configure* - Manage system services



## WWCTL CONFIGURE SSH

Manage and initialize SSH

### 39.1 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]
```

### 39.2 Options

```
-h, --help                help for ssh
-t, --keytypes stringArray ssh key types to be created
```

### 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 configure* - Manage system services



## WWCTL CONFIGURE TFTP

Manage and initialize TFTP

### 40.1 Synopsis

TFTP is a dependent service of Warewulf, this tool will enable the tftp services on your Warewulf master.

```
wwctl configure tftp [OPTIONS]
```

### 40.2 Options

```
-h, --help    help for tftp  
-s, --show    Show configuration (don't update)
```

### 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





## WWCTL CONFIGURE WAREWULFD

Enable and start warewulfd

### 41.1 Synopsis

Enable and starts the warewulfd service.

```
wwctl configure warewulfd [OPTIONS]
```

### 41.2 Options

```
-h, --help    help for warewulfd
```

### 41.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 41.4 SEE ALSO

- *wwctl configure* - Manage system services



## WWCTL IMAGE

Operating system image management

### 42.1 Synopsis

Starting with version 4, Warewulf uses images to build the bootable node images. These commands will help you import, manage, and transform images into bootable Warewulf images.

### 42.2 Options

```
-h, --help    help for image
```

### 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* - 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* - Synchronizes user in image

## WWCTL IMAGE BUILD

(Re)build a bootable image

### 43.1 Synopsis

This command will build a bootable image from an imported IMAGE(s).

```
wwctl image build [OPTIONS] IMAGE [...]
```

### 43.2 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
```

### 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 image* - Operating system image management



## WWCTL IMAGE COPY

Copy an existing image

### 44.1 Synopsis

This command will duplicate an imported image.

```
wwctl image copy IMAGE NEW_NAME
```

### 44.2 Options

```
-b, --build    Build image after copy  
-h, --help    help for copy
```

### 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 image* - Operating system image management





## WWCTL IMAGE DELETE

Delete an imported image

### 45.1 Synopsis

This command will delete IMAGES that have been imported into Warewulf.

```
wwctl image delete [OPTIONS] IMAGE [...]
```

### 45.2 Options

```
-h, --help    help for delete  
-y, --yes     Set 'yes' to all questions asked
```

### 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 image* - Operating system image management



## WWCTL IMAGE EXEC

Run a command inside of a Warewulf image

### 46.1 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
```

### 46.2 Options

<code>-b, --bind stringArray</code>	<code>source[:destination[:{ro copy}]]</code> Bind a local path which must exist into the image. If ↵
<code>↵destination is not</code>	<code>set</code> , uses the same path <b>as</b> source. <code>"ro"</code> binds read-only.
<code>↵"copy" temporarily</code>	copies the file into the image.
<code>--build</code>	(Re)build the image automatically (default true)
<code>-h, --help</code>	help <b>for</b> exec
<code>-n, --node string</code>	Create a read only view of the image <b>for</b> the given node
<code>--syncuser</code>	Synchronize UIDs/GIDs <b>from host</b> to image

### 46.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

## 46.4 SEE ALSO

- *wwctl image* - Operating system image management

## WWCTL IMAGE IMPORT

Import an image into Warewulf

### 47.1 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]
```

### 47.2 Examples

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

### 47.3 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, superseedes env WAREWULF_OCI_
↪ NOHTTPS	
--password string	Set password for the access to the registry, superseedes env_
↪ WAREWULF_OCI_PASSWORD	
--platform string	Set other hardware platform e.g. amd64 or arm64, superseedes_
↪ env WAREWULF_OCI_PLATFORM	
--syncuser	Synchronize UIDs/GIDs from host to image

(continues on next page)

(continued from previous page)

```
-u, --update          Overwrite files in an existing image with the files of remote_  
↪image  
  --username string   Set username for the access to the registry, superseedes env_  
↪WAREWULF_OCI_USERNAME
```

## 47.4 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
```

## 47.5 SEE ALSO

- *wwctl image* - Operating system image management

## WWCTL IMAGE KERNELS

List available image kernels

### 48.1 Synopsis

This command lists the kernels that are available in the imported images.

```
wwctl image kernels [OPTIONS]
```

### 48.2 Options

```
-h, --help    help for kernels
```

### 48.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 48.4 SEE ALSO

- *wwctl image* - Operating system image management





## WWCTL IMAGE LIST

List imported Warewulf images

### 49.1 Synopsis

This command will show you the images that are imported into Warewulf.

```
wwctl image list [OPTIONS]
```

### 49.2 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
```

### 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



## WWCTL IMAGE RENAME

Rename an existing image

### 50.1 Synopsis

This command will rename an existing image.

```
wwctl image rename IMAGE NEW_NAME
```

### 50.2 Options

```
-b, --build    Build image after rename  
-h, --help    help for rename
```

### 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



## WWCTL IMAGE SHELL

Run a shell inside of a Warewulf image

### 51.1 Synopsis

Run a interactive shell inside of a warewulf IMAGE.

```
wwctl image shell [OPTIONS] IMAGE
```

### 51.2 Options

<code>-b, --bind stringArray</code>	<code>source[:destination[:{ro copy}]]</code> Bind a local path which must exist into the image. If destination <b>is not set</b> , uses the same path <b>as</b> source. <b>"ro"</b> binds read-only. <b>"copy"</b> temporarily copies the file into the image.
<code>--build</code>	(Re)build the image automatically (default true)
<code>-h, --help</code>	help <b>for</b> shell
<code>-n, --node string</code>	Create a read only view of the image <b>for</b> the given node
<code>--syncuser</code>	Synchronize UIDs/GIDs <b>from host</b> to image

### 51.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

## 51.4 SEE ALSO

- *wwctl image* - Operating system image management

## WWCTL IMAGE SHOW

Show root fs dir for image

### 52.1 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
```

### 52.2 Options

```
-a, --all      Show all information about an image  
-h, --help    help for show
```

### 52.3 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
```

### 52.4 SEE ALSO

- *wwctl image* - Operating system image management





## WWCTL IMAGE SYNCUSER

Synchronizes user in image

### 53.1 Synopsis

Synchronize the uids and gids from the host to the image. Users/groups which are only present in the image will be preserved if no uid/gid collision is detected. File ownerships are also changed.

```
wwctl image syncuser [OPTIONS] IMAGE
```

### 53.2 Options

```
--build    Build image after syncuser is completed
-h, --help  help for syncuser
--write     Synchronize uis/gids and write files in image (default true)
```

### 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



## WWCTL NODE

Node management

### 54.1 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.

### 54.2 Options

```
-h, --help    help for node
```

### 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* - 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 ADD

Add new node to Warewulf

### 55.1 Synopsis

This command will add a new node named NODENAME to Warewulf.

```
wwctl node add [OPTIONS] NODENAME
```

### 55.2 Options

--asset string	Set the node's Asset tag (key)
-c, --cluster string	Set cluster group
--comment string	Set arbitrary string comment
-e, --discoverable WWbool[=true]	Make discoverable <b>in</b> given network (true/false)
--diskname string	<b>set</b> diskdevice name
--diskwipe	whether <b>or not</b> the partition tables shall be wiped
--fsformat string	<b>format</b> of the file system
--fsname string	<b>set</b> 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	Set the node's IPv4 network device gateway
--gateway6 ip	Set the node's IPv6 network device gateway
-h, --help	help <b>for</b> add
-H, --hwaddr string	Set the device's HW address for given network
--image string	Set image name
-i, --init string	Define the init process to boot the image
-I, --ipaddr ip	IPv4 address <b>in</b> given network
--ipaddr6 ip	IPv6 address <b>in</b> given network
--ipmiaddr ip	Set the IPMI IP address
--ipmiescapechar string	Set the IPMI escape character (defaults: '~')
--ipmi gateway ip	Set the IPMI gateway
--ipmi interface string	Set the node's IPMI interface (defaults: 'lan')
--ipminetmask ip	Set the IPMI netmask
--ipmipass string	Set the IPMI password
--ipmiport string	Set the IPMI port
--ipmitagadd stringToString	add ipmi tags (default [])
--ipmitemplate string	template used <b>for</b> ipmi command
--ipmiuser string	Set the IPMI username

(continues on next page)

(continued from previous page)

--ipmiwrite WWbool[=true]	Enable the write of ipmi configuration (true/false)
--ipxe string	Set the iPXE template name
-A, --kernelargs strings	Set kernel arguments
--kernelversion string	Set kernel version
--mtu string	Set the mtu
-N, --netdev string	Set the device <b>for</b> given network
-M, --netmask ip	Set the networks netmask
--netname string	network which <b>is</b> modified (default "default")
--nettagadd stringToString	add network tags (default [])
--onboot WWbool[=true]	Enable/disable network device (true/false)
--partcreate	Create partition <b>if</b> it does <b>not</b> exist
--partname string	<b>set</b> the partition name so it can be used by a file_
↪system	
--partnumber string	Set the partition number, <b>if not</b> set next free_
↪slot <b>is</b> used	
--partsize string	Set the size of the partition, <b>if not</b> set maximal_
↪possible size <b>is</b> used	
--parttype string	Set the partition <b>type</b> GUID
--prefixlen6 string	Set the networks IPv6 prefix length
-p, --primarynet string	Set the primary network interface
-P, --profile strings	Set the node's <b>profile members</b> (comma separated)
--root string	Define the rootfs
-R, --runtime-overlays strings	Set the runtime overlay
-O, --system-overlays strings	Set the system overlay
--tagadd stringToString	add tags (default [])
-T, --type string	Set device <b>type</b> of given network

## 55.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

## 55.4 SEE ALSO

- *wwctl node* - Node management

## WWCTL NODE CONSOLE

Connect to IPMI console

### 56.1 Synopsis

Start a new IPMI console for NODENAME.

```
wwctl node console [OPTIONS] NODENAME
```

### 56.2 Options

```
-h, --help    help for console
```

### 56.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 56.4 SEE ALSO

- [\*wwctl node\*](#) - Node management





## WWCTL NODE DELETE

Delete a node from Warewulf

### 57.1 Synopsis

This command will remove NODE(s) from the Warewulf node configuration.

```
wwctl node delete [OPTIONS] NODE [NODE ...]
```

### 57.2 Options

```
-h, --help    help for delete  
-y, --yes     Set 'yes' to all questions asked
```

### 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 node* - Node management



## WWCTL NODE EDIT

Edit node(s) with editor

### 58.1 Synopsis

This command opens an editor for the given nodes.

```
wwctl node edit [OPTIONS] NODENAME
```

### 58.2 Options

```
-h, --help      help for edit  
--noheader     Do not print header
```

### 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 node* - Node management



## WWCTL NODE EXPORT

Export nodes as yaml to stdout

### 59.1 Synopsis

This command exports the given nodes as yaml to stdout.

```
wwctl node export NODENAME
```

### 59.2 Options

```
-h, --help    help for export
```

### 59.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 59.4 SEE ALSO

- [\*wwctl node\*](#) - Node management



## WWCTL NODE IMPORT

Import node(s) from yaml FILE

### 60.1 Synopsis

This command imports all the nodes defined in a YAML file. It will overwrite nodes with same name.

```
wwctl node import [OPTIONS] FILE
```

### 60.2 Options

```
-h, --help    help for import  
-y, --yes     Set 'yes' to all questions asked
```

### 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





## WWCTL NODE LIST

List nodes

### 61.1 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 pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node list [OPTIONS] [PATTERN]
```

### 61.2 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
```

### 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

## WWCTL NODE SENSORS

Show node IPMI sensor information

### 62.1 Synopsis

Show IPMI sensor information for nodes matching PATTERN. Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node sensors [OPTIONS] PATTERN
```

### 62.2 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
```

### 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



## WWCTL NODE SET

Configure node properties

### 63.1 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 pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl node set [OPTIONS] PATTERN
```

### 63.2 Options

-a, --all	Set all nodes
--asset string	Set the node's Asset tag (key)
-c, --cluster string	Set cluster group
--comment string	Set arbitrary string comment
-e, --discoverable WWbool[=true]	Make 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
-f, --force	Force configuration (even on error)
--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	Set the node's IPv4 network device gateway
--gateway6 ip	Set the node's IPv6 network device gateway
-h, --help	help for set
-H, --hwaddr string	Set the device's HW address for given network
--image string	Set image name
-i, --init string	Define the init process to boot the image
-I, --ipaddr ip	IPv4 address in given network
--ipaddr6 ip	IPv6 address in given network
--ipmiaddr ip	Set the IPMI IP address
--ipmiescapechar string	Set the IPMI escape character (defaults: '~')
--ipmigateway ip	Set the IPMI gateway

(continues on next page)

(continued from previous page)

--ipmiinterface string	Set the node's IPMI interface (defaults: 'lan')
--ipminetmask ip	Set the IPMI netmask
--ipmipass string	Set the IPMI password
--ipmiport string	Set the IPMI port
--ipmitagadd stringToString	add ipmi tags (default [])
--ipmitagdel strings	delete ipmi tags
--ipmitemplate string	template used <b>for</b> ipmi command
--ipmiuser string	Set the IPMI username
--ipmiwrite WWbool[=true]	Enable the write of ipmi configuration (true/false)
--ipxe string	Set the iPXE template name
-A, --kernelargs strings	Set kernel arguments
--kernelversion string	Set kernel version
--mtu string	Set the mtu
--netdel string	network to delete
-N, --netdev string	Set the device <b>for</b> given network
-M, --netmask ip	Set the networks netmask
--netname string	network which <b>is</b> modified (default "default")
--nettagadd stringToString	add network tags (default [])
--nettagdel strings	delete network tags
--onboot WWbool[=true]	Enable/disable network device (true/false)
--partcreate	Create partition <b>if</b> it does <b>not</b> exist
--partdel string	delete the partition <b>from the</b> configuration
--partname string	<b>set</b> the partition name so it can be used by a file_
↪system	
--partnumber string	Set the partition number, <b>if not</b> set next free_
↪slot <b>is</b> used	
--partsize string	Set the size of the partition, <b>if not</b> set maximal_
↪possible size <b>is</b> used	
--parttype string	Set the partition <b>type</b> GUID
--prefixlen6 string	Set the networks IPv6 prefix length
-p, --primarynet string	Set the primary network interface
-P, --profile strings	Set the node's <b>profile members</b> (comma separated)
--root string	Define the rootfs
-R, --runtime-overlays strings	Set the runtime overlay
-O, --system-overlays strings	Set the system overlay
--tagadd stringToString	add tags (default [])
--tagdel strings	add tags
-T, -- <b>type</b> string	Set device <b>type</b> of given network
-y, --yes	Set ' <b>yes</b> ' to <b>all</b> questions asked

## 63.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

## 63.4 SEE ALSO

- *wwctl node* - Node management





## WWCTL NODE STATUS

View the provisioning status of nodes

### 64.1 Synopsis

View and monitor the status of nodes as they are provisioned and check in.

```
wwctl node status [OPTIONS] [NODENAME...]
```

### 64.2 Options

-h, --help	help <b>for</b> status
-l, --last	Sort by the last check- <b>in</b> time
-r, --reverse	Reverse the sort order
-t, --time <i>int</i>	Filter by last checkin time (seconds)
-u, --unknown	Only show nodes of unknown status
-U, --update <i>int</i>	Set the update frequency <b>for</b> 'watch' (ms) (default 500)
-w, --watch	Watch the status automatically

### 64.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf <i>string</i>	Set the warewulf configuration file

### 64.4 SEE ALSO

- *wwctl node* - Node management



## WWCTL OVERLAY

Warewulf Overlay Management

### 65.1 Synopsis

Management interface for Warewulf overlays

### 65.2 Options

```
-h, --help    help for overlay
```

### 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* - 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

## WWCTL OVERLAY BUILD

(Re)build node overlays

### 66.1 Synopsis

This command builds overlays for given nodes.

```
wwctl overlay build [OPTIONS] NODENAME...
```

### 66.2 Options

```
-h, --help          help for build
-o, --output string  Do not create an overlay image for distribution but write to
                    the given directory. An overlay must also be ge given.
↳to use this option.
-O, --overlay strings Build only specific overlay(s)
    --workers int     The number of parallel workers building overlays (<=0.
↳indicates 1 worker per CPU)
```

### 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 overlay* - Warewulf Overlay Management



## WWCTL OVERLAY CHMOD

Change file permissions in an overlay

### 67.1 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
```

### 67.2 Examples

```
wwctl overlay chmod default /etc/hostname.ww 0660
```

### 67.3 Options

```
-h, --help    help for chmod
```

### 67.4 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

## 67.5 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management



## WWCTL OVERLAY CHOWN

Change file ownership within an overlay

### 68.1 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]
```

### 68.2 Options

```
-h, --help    help for chown
```

### 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 overlay* - Warewulf Overlay Management



## WWCTL OVERLAY CREATE

Initialize a new Overlay

### 69.1 Synopsis

This command creates a new empty overlay with the given OVERLAY\_NAME.

```
wwctl overlay create [OPTIONS] OVERLAY_NAME
```

### 69.2 Options

```
-h, --help    help for create
```

### 69.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 69.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management



## WWCTL OVERLAY DELETE

Delete Warewulf Overlay or files

### 70.1 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 ...]]
```

### 70.2 Options

```
-f, --force      Force deletion of a non-empty overlay
-h, --help      help for delete
-p, --parents    Remove empty parent directories
```

### 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 overlay* - Warewulf Overlay Management



## WWCTL OVERLAY EDIT

Edit or create a file within a Warewulf Overlay

### 71.1 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
```

### 71.2 Options

```
-h, --help          help for edit
-m, --mode int32    Permission mode for directory (default 493)
-p, --parents       Create any necessary parent directories
```

### 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 overlay* - Warewulf Overlay Management





## WWCTL OVERLAY IMPORT

Import a file into a Warewulf Overlay

### 72.1 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]
```

### 72.2 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)
```

### 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



## WWCTL OVERLAY INFO

Show variables for a template file in an overlay

### 73.1 Synopsis

This command will show the variables for a given template file in a given overlay.

```
wwctl overlay info [flags] OVERLAY_NAME FILE_PATH
```

### 73.2 Options

```
-h, --help    help for info
```

### 73.3 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
```

### 73.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management



## WWCTL OVERLAY LIST

List Warewulf Overlays and files

### 74.1 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
```

### 74.2 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
```

### 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



## WWCTL OVERLAY MKDIR

Create a new directory within an Overlay

### 75.1 Synopsis

This command creates a new directory within the Warewulf OVERLAY\_NAME.

```
wwctl overlay mkdir [OPTIONS] OVERLAY_NAME DIRECTORY
```

### 75.2 Options

```
-h, --help          help for mkdir  
-m, --mode int32    Permission mode for directory (default 493)
```

### 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





## WWCTL OVERLAY SHOW

Show (cat) a file within a Warewulf Overlay

### 76.1 Synopsis

This command displays the contents of FILE within OVERLAY\_NAME.

```
wwctl overlay show [OPTIONS] OVERLAY_NAME FILE
```

### 76.2 Options

-h, --help	help <b>for</b> show
-q, --quiet	do <b>not print</b> information <b>if</b> multiple, backup files are written
-r, --render string	node used <b>for</b> the variables <b>in</b> the template

### 76.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 76.4 SEE ALSO

- *wwctl overlay* - Warewulf Overlay Management



## WWCTL POWER

Warewulf node power management

### 77.1 Synopsis

This command controls the power state of nodes.

### 77.2 Options

<code>-h, --help</code>	help <b>for</b> power
-------------------------	-----------------------

### 77.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 77.4 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)



## WWCTL POWER CYCLE

Power cycle the given node(s)

### 78.1 Synopsis

This command cycles power for a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power cycle [OPTIONS] [PATTERN ...]
```

### 78.2 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
```

### 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 power* - Warewulf node power management



## WWCTL POWER OFF

Power off the given node(s)

### 79.1 Synopsis

This command will shutdown power to a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power off [OPTIONS] [PATTERN ...]
```

### 79.2 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
```

### 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 power* - Warewulf node power management





## WWCTL POWER ON

Power on the given node(s)

### 80.1 Synopsis

This command will power on a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power on [OPTIONS] [PATTERN ...] [flags]
```

### 80.2 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
```

### 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 power* - Warewulf node power management



## WWCTL POWER RESET

Issue a reset to node(s)

### 81.1 Synopsis

This command will issue a reset to a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power reset [OPTIONS] [PATTERN ...]
```

### 81.2 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
```

### 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 power* - Warewulf node power management



## WWCTL POWER SOFT

Gracefully shuts down the given node(s)

### 82.1 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 pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power soft [OPTIONS] [PATTERN ...]
```

### 82.2 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
```

### 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 power* - Warewulf node power management



## WWCTL POWER STATUS

Show power status for the given node(s)

### 83.1 Synopsis

This command displays the power status of a set of nodes specified by PATTERN. Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl power status [OPTIONS] [PATTERN ...]
```

### 83.2 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
```

### 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 power* - Warewulf node power management





## WWCTL PROFILE

Node configuration profile management

### 84.1 Synopsis

Management of node profile settings

### 84.2 Options

```
-h, --help    help for profile
```

### 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* - Warewulf Control
- *wwctl profile add* - Add a new node profile
- *wwctl profile delete* - Delete a node profile
- *wwctl profile edit* - Edit node(s) with editor
- *wwctl profile list* - List profiles and configurations
- *wwctl profile set* - Configure node profile properties



## WWCTL PROFILE ADD

Add a new node profile

### 85.1 Synopsis

This command adds a new named PROFILE.

```
wwctl profile add PROFILE
```

### 85.2 Options

-c, --cluster string	Set cluster group
--comment string	Set arbitrary string comment
--diskname string	set diskdevice name
--diskwipe	whether <b>or not</b> 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	Set the node's IPv4 network device gateway
--gateway6 ip	Set the node's IPv6 network device gateway
-h, --help	help <b>for</b> add
-H, --hwaddr string	Set the device's HW address for given network
--image string	Set image name
-i, --init string	Define the init process to boot the image
-I, --ipaddr ip	IPv4 address <b>in</b> given network
--ipaddr6 ip	IPv6 address <b>in</b> given network
--ipmiaddr ip	Set the IPMI IP address
--ipmiescapechar string	Set the IPMI escape character (defaults: '~')
--ipmigateway ip	Set the IPMI gateway
--ipmiinterface string	Set the node's IPMI interface (defaults: 'lan')
--ipminetmask ip	Set the IPMI netmask
--ipmipass string	Set the IPMI password
--ipmiport string	Set the IPMI port
--ipmitagadd stringToString	add ipmi tags (default [])
--ipmitemplate string	template used <b>for</b> ipmi command
--ipmiuser string	Set the IPMI username
--ipmiwrite WWbool[=true]	Enable the write of ipmi configuration (true/false)
--ipxe string	Set the iPXE template name

(continues on next page)

(continued from previous page)

-A, --kernelargs strings	Set kernel arguments
--kernelversion string	Set kernel version
--mtu string	Set the mtu
-N, --netdev string	Set the device <b>for</b> given network
-M, --netmask ip	Set the networks netmask
--netname string	network which <b>is</b> modified (default "default")
--nettagadd stringToString	add network tags (default [])
--onboot WWbool[=true]	Enable/disable network device (true/false)
--partcreate	Create partition <b>if</b> it does <b>not</b> exist
--partname string	<b>set</b> the partition name so it can be used by a file_
↪system	
--partnumber string	Set the partition number, <b>if not set next</b> free slot_
↪ <b>is</b> used	
--partsize string	Set the size of the partition, <b>if not set</b> maximal_
↪possible size <b>is</b> used	
--parttype string	Set the partition <b>type</b> GUID
--prefixlen6 string	Set the networks IPv6 prefix length
-p, --primarynet string	Set the primary network interface
-P, --profile strings	Set the node's <b>profile members</b> (comma separated)
--root string	Define the rootfs
-R, --runtime-overlays strings	Set the runtime overlay
-O, --system-overlays strings	Set the system overlay
--tagadd stringToString	add tags (default [])
-T, -- <b>type</b> string	Set device <b>type</b> of given network

## 85.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

## 85.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

## WWCTL PROFILE DELETE

Delete a node profile

### 86.1 Synopsis

This command deletes the node PROFILE. You may use a pattern for PROFILE.

```
wwctl profile delete [OPTIONS] PROFILE
```

### 86.2 Options

```
-h, --help    help for delete  
-y, --yes     Set 'yes' to all questions asked
```

### 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 profile* - Node configuration profile management



## WWCTL PROFILE EDIT

Edit node(s) with editor

### 87.1 Synopsis

This command opens an editor for the given profiles.

```
wwctl profile edit [OPTIONS] NODENAME
```

### 87.2 Options

```
-h, --help      help for edit  
--noheader     Do not print header
```

### 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 profile* - Node configuration profile management





## WWCTL PROFILE LIST

List profiles and configurations

### 88.1 Synopsis

This command will display configurations for PROFILE.

```
wwctl profile list [OPTIONS] [PROFILE ...]
```

### 88.2 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
```

### 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 profile* - Node configuration profile management



## WWCTL PROFILE SET

Configure node profile properties

### 89.1 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]
```

### 89.2 Options

-c, --cluster string	Set cluster group
--comment string	Set arbitrary string comment
--diskdel string	delete the disk <b>from the</b> configuration
--diskname string	<b>set</b> diskdevice name
--diskwipe	whether <b>or not</b> the partition tables shall be wiped
--fsdel string	delete the fs <b>from the</b> configuration
--fsformat string	<b>format</b> of the file system
--fsname string	<b>set</b> the file system name which must match a <b>␣</b>
↪partition name	
--fspath string	the mount point of the file system
--fswipe	wipe file system at boot
-G, --gateway ip	Set the node's <b>IPv4 network device gateway</b>
--gateway6 ip	Set the node's <b>IPv6 network device gateway</b>
-h, --help	help <b>for set</b>
-H, --hwaddr string	Set the device's <b>HW address for given network</b>
--image string	Set image name
-i, --init string	Define the init process to boot the image
-I, --ipaddr ip	IPv4 address <b>in</b> given network
--ipaddr6 ip	IPv6 address <b>in</b> given network
--ipmiaddr ip	Set the IPMI IP address
--ipmiescapechar string	Set the IPMI escape character (defaults: '~')
--ipmigateway ip	Set the IPMI gateway
--ipmiinterface string	Set the node's <b>IPMI interface</b> (defaults: 'lan')
--ipminetmask ip	Set the IPMI netmask
--ipmipass string	Set the IPMI password
--ipmiport string	Set the IPMI port
--ipmitagadd stringToString	add ipmi tags (default [])

(continues on next page)

(continued from previous page)

--ipmitagdel strings	delete ipmi tags
--ipmitemplate string	template used <b>for</b> ipmi command
--ipmiuser string	Set the IPMI username
--ipmiwrite WWbool[=true]	Enable the write of ipmi configuration (true/false)
--ipxe string	Set the iPXE template name
-A, --kernelargs strings	Set kernel arguments
--kernelversion string	Set kernel version
--mtu string	Set the mtu
--netdel string	network to delete
-N, --netdev string	Set the device <b>for</b> given network
-M, --netmask ip	Set the networks netmask
--netname string	network which <b>is</b> modified (default "default")
--nettagadd stringToString	add network tags (default [])
--nettagdel strings	delete network tags
--onboot WWbool[=true]	Enable/disable network device (true/false)
--partcreate	Create partition <b>if</b> it does <b>not</b> exist
--partdel string	delete the partition <b>from the</b> configuration
--partname string	<b>set</b> the partition name so it can be used by a file_
↪system	
--partnumber string	Set the partition number, <b>if not set</b> next free slot_
↪ <b>is</b> used	
--partsize string	Set the size of the partition, <b>if not set</b> maximal_
↪possible size <b>is</b> used	
--parttype string	Set the partition <b>type</b> GUID
--prefixlen6 string	Set the networks IPv6 prefix length
-p, --primarynet string	Set the primary network interface
-P, --profile strings	Set the node's <b>profile members</b> (comma separated)
--root string	Define the rootfs
-R, --runtime-overlays strings	Set the runtime overlay
-O, --system-overlays strings	Set the system overlay
--tagadd stringToString	add tags (default [])
--tagdel strings	add tags
-T, -- <b>type</b> string	Set device <b>type</b> of given network
-y, --yes	Set ' <b>yes</b> ' to <b>all</b> questions asked

## 89.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

## 89.4 SEE ALSO

- *wwctl profile* - Node configuration profile management

## WWCTL SERVER

Start Warewulf server

### 90.1 Synopsis

Start Warewulf server

```
wwctl server [OPTIONS]
```

### 90.2 Options

```
-h, --help    help for server
```

### 90.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 90.4 SEE ALSO

- [\*wwctl\*](#) - Warewulf Control



## WWCTL SSH

SSH into configured nodes in parallel

### 91.1 Synopsis

Easily ssh into nodes in parallel to run non-interactive commands Node patterns are a comma-separated list of individual pattern. Each pattern can either be a full node name or a node range like node[01-03,05].

```
wwctl ssh [OPTIONS] NODE_PATTERN COMMAND
```

### 91.2 Options

-n, --dryrun	Show commands to run
-f, --fanout <i>int</i>	How many connections to run <b>in</b> parallel (default 32)
-h, --help	help <b>for</b> ssh
--rsh <i>string</i>	Path to use <b>for</b> RSH/SSH command (default <code>"/usr/bin/ssh"</code> )
-s, --sleep <i>int</i>	Seconds to sleep inbetween processes

### 91.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf <i>string</i>	Set the warewulf configuration file

### 91.4 SEE ALSO

- *wwctl* - Warewulf Control





## WWCTL UPGRADE

Upgrade configuration files

### 92.1 Synopsis

Upgrade warewulf.conf or nodes.conf from a previous version of Warewulf 4 to a format supported by the current version.

### 92.2 Options

<code>-h, --help</code>	help <b>for</b> upgrade
-------------------------	-------------------------

### 92.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 92.4 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

### 93.1 Synopsis

Upgrades warewulf.conf from a previous version of Warewulf 4 to a format supported by the current version.

```
wwctl upgrade config [OPTIONS]
```

### 93.2 Options

-h, --help	help <b>for</b> config
-i, --input-path string	Path to a legacy warewulf.conf
-o, --output-path string	Path to write the upgraded warewulf.conf to

### 93.3 Options inherited from parent commands

-d, --debug	Run <b>with</b> debugging messages enabled.
-v, --verbose	Run <b>with</b> increased verbosity.
--warewulfconf string	Set the warewulf configuration file

### 93.4 SEE ALSO

- *wwctl upgrade* - Upgrade configuration files



## WWCTL UPGRADE NODES

Upgrade an existing nodes.conf

### 94.1 Synopsis

Upgrades nodes.conf from a previous version of Warewulf 4 to a format supported by the current version.

```
wwctl upgrade nodes [OPTIONS]
```

### 94.2 Options

<code>--add-defaults</code>	Configure a default profile <b>and</b> set default node_
<code>↪values</code>	
<code>-h, --help</code>	help <b>for</b> nodes
<code>-i, --input-path string</code>	Path to a legacy nodes.conf
<code>-o, --output-path string</code>	Path to write the upgraded nodes.conf to
<code>--replace-overlays</code>	Replace 'wwinit' <b>and</b> 'generic' overlays <b>with</b> their_
<code>↪split replacements</code>	
<code>--with-warewulfconf string</code>	Path to a legacy warewulf.conf

### 94.3 Options inherited from parent commands

<code>-d, --debug</code>	Run <b>with</b> debugging messages enabled.
<code>-v, --verbose</code>	Run <b>with</b> increased verbosity.
<code>--warewulfconf string</code>	Set the warewulf configuration file

### 94.4 SEE ALSO

- *wwctl upgrade* - Upgrade configuration files



## WWCTL VERSION

Version information

### 95.1 Synopsis

This command will print the Warewulf version.

```
wwctl version [flags]
```

### 95.2 Options

```
-f, --full    List all compiled in variables.  
-h, --help    help for version
```

### 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\*](#) - Warewulf Control





## 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](#).

### 96.1 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, sub-commands, and options.

### 96.2 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.

## 96.3 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.

## 96.4 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 in stead.

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.)

## 96.5 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"'`)

## 96.6 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]` re-places `--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.

## 96.7 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.

## 96.8 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.

## 96.9 Profiles

Node profiles now support profiles themselves, allowing for complex nested hierarchies of nested 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
```

## 96.10 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.

## 96.11 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
```

## 96.12 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.

## 96.13 Network Boot and wwininit

The network boot and wwininit 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.

## 96.14 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.

## 96.15 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 return 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.

## 96.16 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.

## 96.17 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.

## 96.18 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.

## 96.19 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.



## 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](#).

### 97.1 aarch64 packages

We're now building and publishing aarch64 packages with GitHub releases.

### 97.2 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 `warewulf` at `/api`, and is disabled by default.

### 97.3 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.

### 97.4 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.

## 97.5 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
```

## 97.6 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.

## 97.7 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`.

## 97.8 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.

## 97.9 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
```

## 97.10 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`.



## 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 node image to a local disk.

Significant changes are described below. Additional changes are listed in the [CHANGELOG](#).

### 98.1 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 additional control compared to Warewulf’s built-in disk, partition, and file system configuration.

To support more complex partition layouts, `wwctl <node|partition> 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.

### 98.2 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.

## 98.3 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`.

## 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](#).

### 99.1 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.

### 99.2 IPv6 Support

IPv6 is now supported during the iPXE network boot process, one step towards improving overall IPv6 support.

### 99.3 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.

## 99.4 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`.

## 99.5 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.

## 99.6 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.



## 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](#).

### 100.1 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.

### 100.2 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.

### 100.3 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.

## 100.4 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

## 100.5 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`.

## 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](#).

### 101.1 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.

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 */}}
```

### 101.2 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).

### 101.3 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.

## 101.4 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.

## 101.5 Packaging

The openEuler 24.03 RPM package now uses `dnsmasq` by default.

## 101.6 Development

Work has resumed on support for using Vagrant to initialize a basic testing / development environment with libvirt.

## 101.7 warewulf-dracut

A bug between `warewulf-dracut` and the `ignition` overlay previously caused 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 node image.

## 101.8 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 `tftpboot` as configured in `warewulf.conf`. #2055
- Automatically create a GPT label when `sfdisk` overlay wipes disks. #2025