
timon.vogt@gwdg.de

Sebastian Krey, Freja Nordsiek, Timon Vogt

# Cluster Provisioning Using Warewulf

# Table of contents

# What is a Compute Cluster?

- Contrary to single-machine supercomputers (like Cray-I), a compute cluster consists of a large number (even thousands) of smaller computers ("nodes").
- These nodes work together on big computational tasks (weather simulation, AI training, etc.).
- All nodes are usually connected with a high-speed, dedicated network.
- Individual nodes are (comparably) cheap and easy to replace.

Figure: Image of Emmy, one of the systems of NHR@Göttingen.

# Simple Example: Beowulf

- Original design dates back to 1996 at NASA
- Basic idea: Connect commodity hardware inside a private network
- Goal: Being able to run parallel programs, e.g. using a Message Passing Interface (MPI), to bring down wall-clock times
- One has two
  - types of nodes
  - types of network

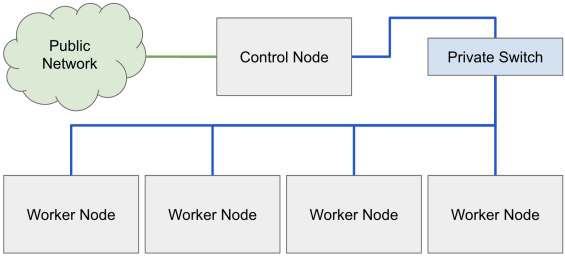



Figure: Simple sketch of the beowulf cluster.
https://warewulf.org/docs/development/contents/background.html

# Interactive: What is the Challenge for Administrators

- What do you think is the challenge for administrators when you scale out such a cluster?
- How would you solve this?

# Problem and Solution for Admins

- One problem is to manage all software stacks on all nodes when scaling out
- Solution: Unlike on your Laptop, one does not use a local installation of the Kernel, OS and other software
- Instead all nodes fetch their images and related software from a single `Control Node`

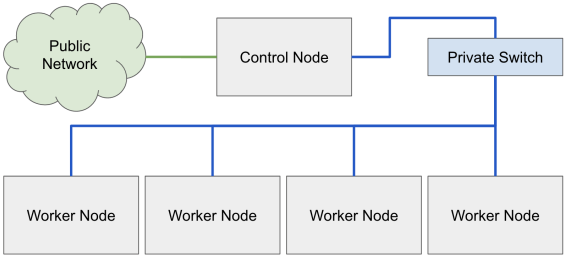



Figure: Simple sketch of the Beowulf cluster concept.
https://warewulf.org/docs/development/contents/background.html

# Problem and Solution for Admins

- This ensures a homogeneous state across all nodes
- And an admin only has to work on a single machine, not on thousands

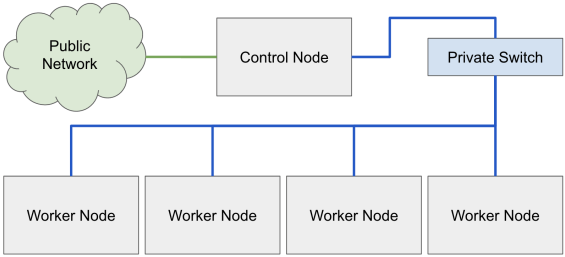



Figure: Simple sketch of the Beowulf cluster concept.
https://warewulf.org/docs/development/contents/background.html

# PXE Boot

- To enable the distribution of the operating system from a single control node, the nodes need to be able to fetch the OS before the actual boot process starts.
- This is different from your laptop, where the bootloader loads the OS from disk, here, it needs to download the OS from a remote location.
- This is done by the `Preboot Execution Environment` (PXE) boot system

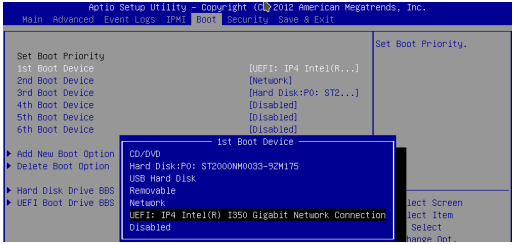



Figure: UEFI boot priorities needed to enable PXE boot.
https://docs.mirantis.com/mcp/q4-18/mcp-deployment-guide/deployment-customizations-guidelines/boot-uefi-pxe.html

# Basic PXE Workflow

- Prerequisites:
  - PXE requires a DHCP and a TFTP server
  - The client requires a PXE-capable network card and appropriate BIOS/UEFI settings
  - Modern, UEFI-based, systems can also boot via HTTP(S) instead of TFTP.

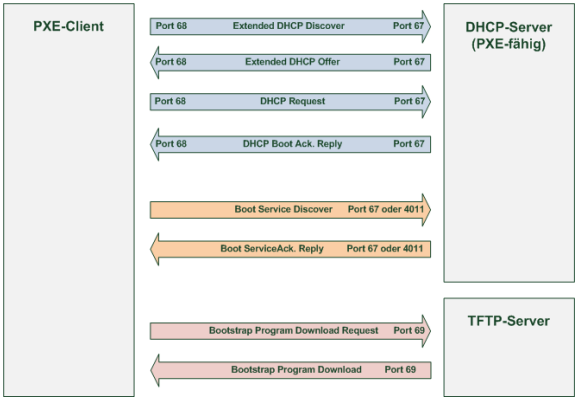



Figure: Simplified workflow of the PXE boot process. https://de.wikipedia.org/wiki/Preboot_Execution_Environment

# Basic PXE Workflow

1. PXE device (network card) initializes the workflow, sends out a DHCP request.
2. A DHCP server replies, sends back
   - IP address
   - Netmask
   - Location of TFTP server and path of file
3. PXE device starts the download from TFTP server
4. The downloaded bootloader is executed in RAM

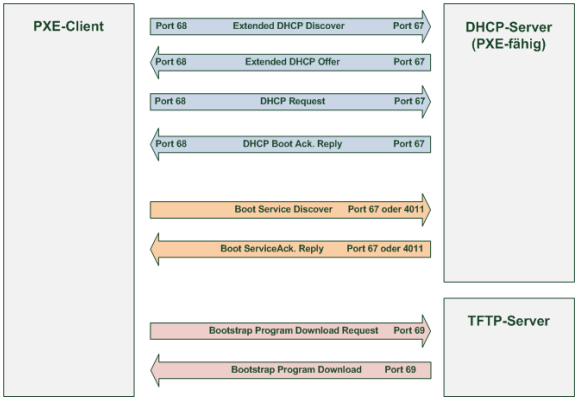



Figure: Simplified workflow of the PXE boot process. https://de.wikipedia.org/wiki/Preboot_Execution_Environment

# Warewulf and iPXE-Boot

- PXE can only download a single file via TFTP. Thus, a second stage is needed for a flexible Linux boot process: iPXE.

  iPXE is a scriptable extension to the regular PXE boot.

- Once the iPXE binary is downloaded, it is executed and takes over the boot process:
  1. iPXE behaves like PXE, thus it also sends out a DHCP request.
  2. The DHCP server recognizes it, responds with the (node-specific) iPXE script.
  3. The script instructs iPXE to download the node's kernel, container, system- and runtime-overlay via http(s).
  4. The script unpacks them and places them into memory.
  5. iPXE executes the kernel and gives it container and overlays as initramfs.
  6. Some pre-configuration of the container is done.
  7. The container's `/sbin/init` is executed $\rightarrow$ the boot process starts.

# Recap: What are we going to do

- We have stateless compute nodes (disk optional)
- And a stateful control node to manage the compute nodes
- We are doing a PXE-boot to fetch an iPXE stack
  - This step is necessary since pure PXE only allows you to download a **single** file via tftp
  - The overall complexity and fine tuning of warewulf requires a bit more

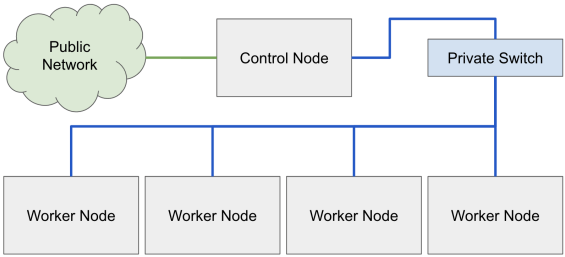



Figure: Simple sketch of the beowulf cluster.
https://warewulf.org/docs/development/contents/background.html

# What Does Warewulf Do?

- Warewulf will be used for the basic PXE-boot and provides
  - the dhcp server
  - the tftp server
- and a stateful control node to manage the compute nodes
- We are using PXE-boot to fetch a iPXE stack
- Network settings, image, kernel are all managed by Warewulf

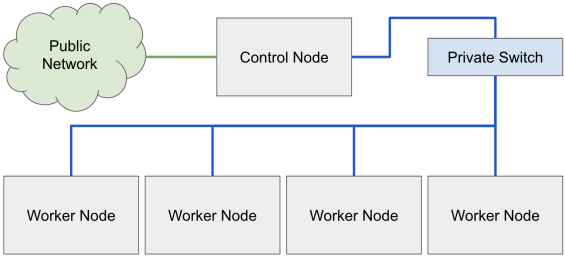



Figure: Simple sketch of the beowulf cluster.
https://warewulf.org/docs/development/contents/background.html

# Warewulf components

What is Warewulf now?

Warewulf consists of:

- **DHCP** server. The DHCP server listens for incoming DHCP requests and initializes the PXE boot process.
- **TFTP** server. The TFTP server enables the download of the iPXE binary at the initial stage.
- **iPXE** binary. The iPXE binary is built together with Warewulf and delivered via TFTP.
- **Warewulfd** server. The Warewulfd server is a simple HTTP server, enabling the download of iPXE script, kernel, overlays, etc.
- A configuration database for all node settings (mainly YAML files)

# Interacting with Warewulf

For interacting with Warewulf (i.e. managing a cluster), Warewulf provides the **wwctl** command for modifying the (YAML-) configuration files conveniently.

**wwctl** can

- add, configure, modify and delete nodes (`wwctl node`).
- add, configure, modify and delete node profiles (`wwctl profile`).
- control, reload and restart the Warewulfd server (`wwctl server`).
- download, update and build images and containers (`wwctl image`).
- create, manage and build overlays (`wwctl overlay`).

# Warewulf Uses Containers?!?

- Yes, basically.
- We are still in HPC; thus the containers have to be booted on bare metal
- Most containers from Docker or Singularity have no kernel and only a lightweight service manager.
  - That works, because the container is supposed to be executed within a container runtime
- However, one can also install the kernel, a full-fledged `systemd` and `core-utils`.
- With that, your container becomes basically a full (although slim) OS.
- You can still work with `docker`, `buildah`, `kiwi`, and many more tools to build your containers:
- You can also convert it into a base image and open a shell and modify/customize an image as you need
- It is basically an uncompressed `chroot`

# Creating Containers from Scratch

- You can bootstrap a mini chroot directory

```
$ sudo dnf install --installroot /tmp/newroot basesystem bash
  ↪ chkconfig coreutils e2fsprogs ethtool filesystem findutils
  ↪ gawk grep initscripts iproute iputils net-tools nfs-utils
  ↪ pam psmisc rsync sed setup shadow-utils rsyslog tzdata
  ↪ util-linux words zlib tar less gzip which util-linux
  ↪ openssh-clients openssh-server dhclient pciutils
  ↪ vim-minimal shadow-utils strace cronie crontabs cpio wget
  ↪ rocky-release dnf
```

- Afterwards, you can import it into warewulf:

```
$ sudo wwctl container import /tmp/newroot containername
```

# Building a Container Using Docker

- You can also directly import containers into Warewulf using any OCI compliant registry:
  ```
  $ sudo wwctl container import
  docker://docker.io/library/debian:latest
  ```
- Alternatively, you can also create your container with a Dockerfile.
- Then, you can build it with docker, with `docker build`
- Warewulf is able to directly import this container:
  ```
  $ sudo wwctl container import
  docker-daemon://<image-name>:<tag> <name>
  ```

# Two Common Strategies for Making and Maintaining the Containers

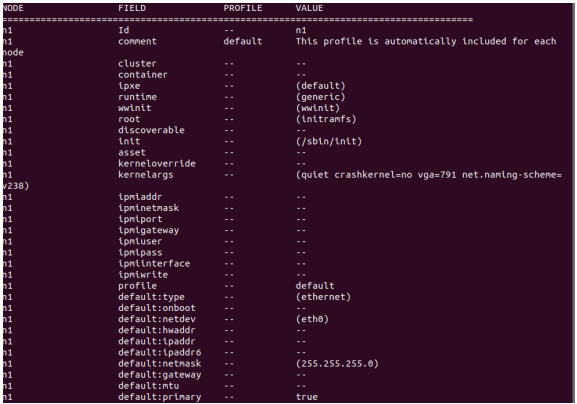
1. Golden images
   - Built by hand in the shell
   - Great option when setting things up that can't be scripted
   - Hard to recreate if lost, especially after many update cycles
   - Documentation is critical to recreate
   - Critical to protect
2. Scripted images
   - Built by a recipe, such as a `Dockerfile` with Docker, Podman, etc.
   - Can rebuild containers more reproducibly
   - The recipe is the instructions how to build
   - Only have to protect the recipes to rebuild (trivial to backup)
   - Some things are hard or impossible to script

# Stateless Provisioning

- Provisioning: Putting an OS onto a system
- As mentioned, directly booting your OS without any installation
- In this case, this is done stateless, i.e. it is provisioned to RAM
- Ongoing discussion about statefulness, e.g. having it on disk
  - Currently there is no need for non-volatile storage
  - Most arguments for statefulness on modern systems not relevant anymore

# Node Settings

- In order to boot a node with warewulf, you need to configure it.
- Warewulfs main job is to hold on to these configuration values, then apply them to nodes as they boot.
- Typical configuration values include container name, IP address, MAC address, kernel parameters, ...

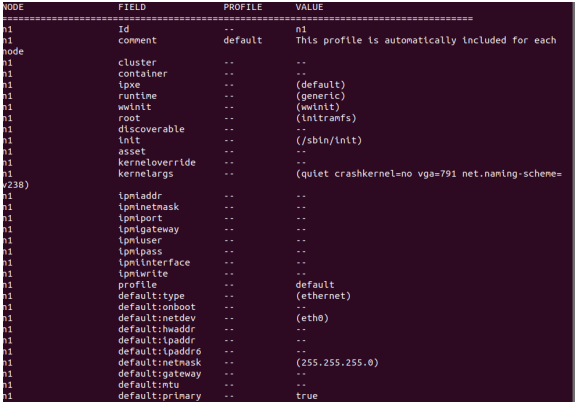


```
NODE                FIELD               PROFILE     VALUE
================================================================================
n1                  Id                  --          n1
n1                  comment             default     This profile is automatically included for each
node
n1                  cluster             --          --
n1                  container           --          --
n1                  ipxe                --          (default)
n1                  runtime             --          (generic)
n1                  wwinit              --          (wwinit)
n1                  root                --          (initramfs)
n1                  discoverable        --          --
n1                  init                --          (/sbin/init)
n1                  asset               --          --
n1                  kerneloverride      --          --
n1                  kernelargs          --          (quiet crashkernel=no vga=791 net.naming-scheme=
v238)
n1                  ipmiaddr            --          --
n1                  ipminetmask         --          --
n1                  ipmiport            --          --
n1                  ipmigateway         --          --
n1                  ipmiuser            --          --
n1                  ipmipass            --          --
n1                  ipmiinterface       --          --
n1                  ipmiwrite           --          --
n1                  profile             --          default
n1                  default:type        --          (ethernet)
n1                  default:onboot      --          --
n1                  default:netdev      --          (eth0)
n1                  default:hwaddr      --          --
n1                  default:ipaddr      --          --
n1                  default:ipaddr6     --          --
n1                  default:netmask     --          (255.255.255.0)
n1                  default:gateway     --          --
n1                  default:mtu         --          --
n1                  default:primary     --          true
```

Figure: Screenshot showing the node attributes which one can set

# Profiles

- For easier configuration, you can group nodes which share the same attributes.
- These groups are called `profiles`.
- Profile settings apply to all nodes with the specific profile.
- Multiple profiles can be applied hierarchically to a node.



```
NODE                FIELD               PROFILE      VALUE
================================================================================
n1                  Id                  --           n1
n1                  comment             default      This profile is automatically included for each
node
n1                  cluster             --           --
n1                  container           --           --
n1                  ipxe                --           (default)
n1                  runtime             --           (generic)
n1                  wwinit              --           (wwinit)
n1                  root                --           (initramfs)
n1                  discoverable        --           --
n1                  init                --           (/sbin/init)
n1                  asset               --           --
n1                  kerneloverride      --           --
n1                  kernelargs          --           (quiet crashkernel=no vga=791 net.naming-scheme=
v238)
n1                  ipmiaddr            --           --
n1                  ipminetmask         --           --
n1                  ipmiport            --           --
n1                  ipmigateway         --           --
n1                  ipmiuser            --           --
n1                  ipmipass            --           --
n1                  ipmiinterface       --           --
n1                  ipmiwrite           --           --
n1                  profile             --           default
n1                  default:type        --           (ethernet)
n1                  default:onboot      --           --
n1                  default:netdev      --           (eth0)
n1                  default:hwaddr      --           --
n1                  default:ipaddr      --           --
n1                  default:ipaddr6     --           --
n1                  default:netmask     --           (255.255.255.0)
n1                  default:gateway     --           --
n1                  default:mtu         --           --
n1                  default:primary     --           true
```

Figure: Screenshot showing some possible Warewulf node attributes

# What are Overlays?

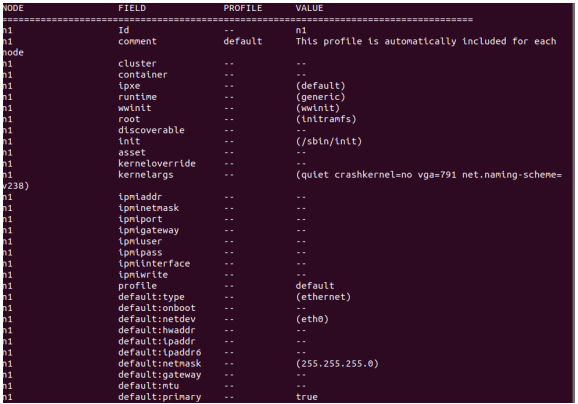
- Usually many nodes use the same stateless image.
- Although this homogeneous state is a general advantage, certain configurations have to be node-specific, like
  - `hostname`
  - network settings: IP addresses, routing setup, network type(e.g. Infiniband, Omni-Path, Ethernet)
  - BMC setup (Redfish or IPMI)
  - Mounted filesystems
- For this, Warewulf offers two different overlay types:
  - `System Overlays` which are applied during the boot of the node.
  - `Runtime Overlays` which are applied periodically while the node is up.
- A node can have multiple of each overlay type. Warewulf combines them into a single file for transmission.

# What are Overlays?

- Both overlay types can be templated.
- During overlay build, Warewulf consider all overlays files ending in `.ww` as templates.
  - Each `.ww` file is processed using Golang's `text/template` engine.
  - Each template has access to all node settings Warewulf has for the node.
  - Warewulf provides a lot of helpful functions for these templates.
- Overlays are combined into `cpio` archive files for every node (think of .zip files)
- After iPXE boots the kernel, the kernel calls the `wwinit` script first, from the `wwinit` overlay.
  - Allows one to modify the rootfs before systemd init is started.

# Applying Overlays

- Custom overlays are stored in `/var/lib/warewulf/overlays`
- Warewulf provides a whole host of overlays, located in `/usr/share/warewulf/overlays`
- You can set the overlays as you set any other attribute, to either nodes directly or to profiles.



```
NODE                FIELD               PROFILE      VALUE
================================================================================
n1                  Id                  --           n1
n1                  comment             default      This profile is automatically included for each
node
n1                  cluster             --           --
n1                  container           --           --
n1                  ipxe                --           (default)
n1                  runtime             --           (generic)
n1                  wwinit              --           (wwinit)
n1                  root                --           (initramfs)
n1                  discoverable        --           --
n1                  init                --           (/sbin/init)
n1                  asset               --           --
n1                  kerneloverride      --           --
n1                  kernelargs          --           (quiet crashkernel=no vga=791 net.naming-scheme=
v238)
n1                  ipmiaddr            --           --
n1                  ipminetmask         --           --
n1                  ipmiport            --           --
n1                  ipmigateway         --           --
n1                  ipmiuser            --           --
n1                  ipmipass            --           --
n1                  ipmiinterface       --           --
n1                  ipmiwrite           --           --
n1                  profile             --           default
n1                  default:type        --           (ethernet)
n1                  default:onboot      --           --
n1                  default:netdev      --           (eth0)
n1                  default:hwaddr      --           --
n1                  default:ipaddr      --           --
n1                  default:ipaddr6     --           --
n1                  default:netmask     --           (255.255.255.0)
n1                  default:gateway     --           --
n1                  default:mtu         --           --
n1                  default:primary     --           true
```

Figure: Screenshot showing some possible Warewulf node attributes

# Questions/Exercise

Any Questions?

Exercise