Maximizing Memory: Arizona State University's Disk-full Take on Warewulf

Josh Burks

HPC Systems Analyst Research Technology Office Arizona State University

Presented at Dell XL, April 1, 2025





Warewulf @ ASU

3 Custers - 759 Nodes Total

- Sol 270 Nodes (Flagship Cluster, Top500 #436)
- Phoenix 545 Nodes (Heterogeneous Cluster)
- Aloe 26 Nodes (Secure Environment)

Compute nodes, login nodes (VMs), and Slurm node (VM) are all booted with Warewulf



Why did we move to Warewulf?

Consistency

 Minimizes configuration drift for predictable performance.

Scalability

• Streamlined growth of compute resources.

Stateful Provisioning with Cobbler+Salt

Imaging

PXE Boot - iPXE

Cobbler Kickstart Install Base OS

Reboot to local drive

Ansible Installs Salt

Salt States Run

~2 Hours

DNF packages, conf files, drivers

Reboot

Ready

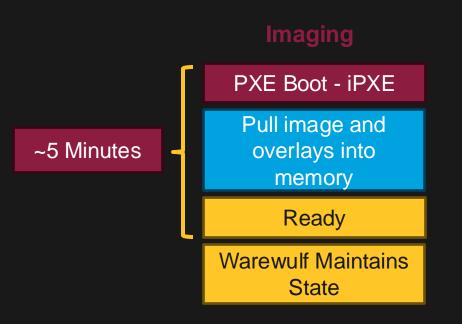
Booting

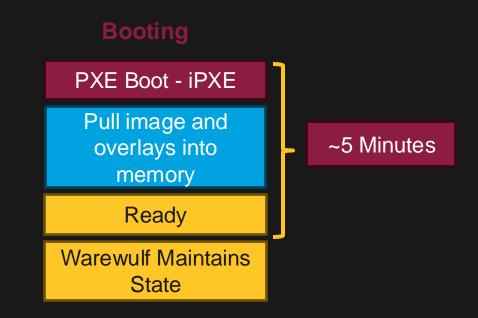
Boot Local Drive

Ready

Salt Maintains State

~3 Minutes





Imaging/Booting

PXE Boot - iPXE

Pull image and overlays into memory

Ready

Warewulf Maintains
State

Server POSTs and PXE boots from selected network device

Server is assigned DHCP address by werewulf's DHCP server

Chainload iPXE via TFTP (Located at /var/lib/tftpboot/warewulf/x86_64.efi)

iPXE Downloads and executes the script located at /etc/warewulf/ipxe/scriptname

iPXE Downloads kernel and image into memory /var/lib/warewulf/provision/image/image.img.gz

iPXE Downloads system and runtime overlays into memory /var/lib/Warewulf/provision/overlays/nodename/overlays.img.gz

iPXE Combines image and overlays into single rootfs and boots the kernel

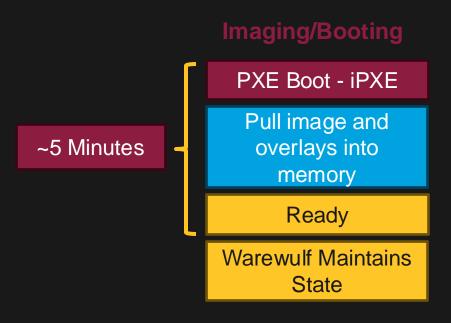
Kernel is initialized in initramfs; /init calls wwinit; wwinit calls /sbin/init (systemd)

Systemd starts as normal, node is brought to multiuser.target

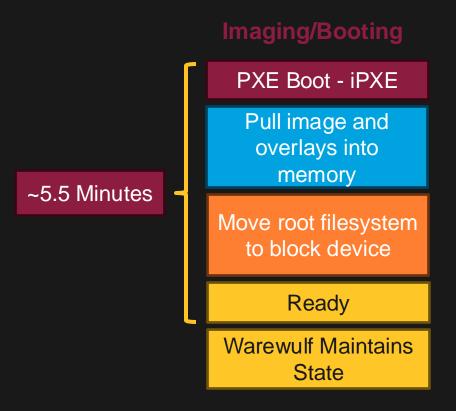
wwclient applies runtime overlays every x minutes to maintain state

What makes ASU's deployment innovative?





Utilizing disks to reduce memory footprint



Imaging/Booting

PXE Boot - iPXE

Pull image and overlays into memory

Move root filesystem to block device

Ready

Warewulf Maintains
State

Server POSTs and PXE boots from selected network device

Server is assigned DHCP address by werewulf's DHCP server

Chainload iPXE via TFTP (Located at /var/lib/tftpboot/Warewulf/x86_64.efi)

iPXE Downloads and executes the script located at /etc/Warewulf/ipxe/script.ipxe

iPXE Downloads kernel and image into memory /var/lib/Warewulf/provision/image/prod.img.gz

iPXE Downloads system and runtime overlays into memory /var/lib/Warewulf/provision/overlays/nodename/overlays.img.gz

iPXE Combines image and overlays into single rootfs and boots the kernel

Kernel is initialized in initramfs; /init is called

/init formats block device; moves rootfs; calls /switch_root

wwinit runs pre-systemd scripts; calls /sbin/init (systemd)

systemd starts as normal and node is brought to multiuser.target

wwclient applies runtime overlays every x minutes to maintain state

Moving the Root Filesystem to Disk

echo b > /proc/sysrq-trigger | /sbin/reboot

```
#init
. /warewulf/config
echo "Warewulf v4 is now booting: $WWHOSTNAME"
mkdir /proc /dev /sys /run 2>/dev/null
mount -t proc proc /proc
mount -t devtmpfs devtmpfs /dev
mount -t sysfs sysfs /sys
mount -t tmpfs tmpfs /run
if test "$WWROOT" = "initramfs"; then
       exec /warewulf/wwinit
elif test "$WWROOT" = "tmpfs"; then
       mkdir /newroot
       mount wwroot /newroot -t tmpfs
       tar -cf - --exclude ./proc --exclude ./sys --exclude ./dev --exclude ./newroot . | \
       tar -xf - -C /newroot
       mkdir /newroot/proc /newroot/dev /newroot/sys /newroot/run 2>/dev/null
       exec /sbin/switch root /newroot /warewulf/wwinit
else
```

Moving the Root Filesystem to Disk

```
#init
                                                  Simplified Example. See the full script on GitHub
                                                            github.com/jeburks2/warewulf-extras/
elif test "$WWROOT" = "xfs"; then
       PATH=$PATH:/sbin
       mkdir /newroot
       modprobe nvme
       nvme="/dev/nvme0n1"
       nvme p1="${nvme}p1"
       sleep 1 #Give kernel time to create block devices
       parted -s $nvme mklabel gpt
       parted -s -a optimal $nvme mkpart primary 0% 100%
       mkfs.xfs -f $nvme_p1 2> /dev/null
       mount $nvme p1 /newroot
       tar -cf - --exclude ./proc --exclude ./sys --exclude ./dev --exclude ./newroot . \
       tar --warning=no-timestamp -xf - -C /newroot
       mkdir /newroot/proc /newroot/dev /newroot/sys /newroot/run 2>/dev/null
       exec /sbin/switch_root /newroot /warewulf/wwprescripts
else
       echo b > /proc/sysrq-trigger | /sbin/reboot
```

Measuring Memory Metrics

10 GiB Container Sol 270 Nodes totaling 143 TiB RAM Phx 545 Nodes totaling 87 TiB RAM

Total
733 Nodes totaling
230 TiB RAM

WW Images use 2.7 TiB (1.9 %)

WW Images use 5.3 TiB (6.1%)

WW Images use 8.0 TiB (3.5%)

By moving the rootfs to disk, we are giving 8 TiB of memory back to researchers

Measuring Memory Metrics

```
WWR00T=tmpfs
df -h /
                                                                 83 GiB Free
Filesystem
                Size Used Avail Use% Mounted on
wwroot
                 47G
                     7.9G
                             39G
                                 17% /
free -h
                                                                       available
              total
                                       free
                                                 shared
                                                         buff/cache
                           used
               93Gi
                          1.0Gi
                                       84Gi
                                                  7.8Gi
                                                               7.9Gi
                                                                            83Gi
Mem:
Swap:
                 0B
                             0B
                                         0B
WWROOT=xfs
df -h /
                                                                 91 GiB Free
                      Used Avail Use% Mounted on
Filesystem
/dev/sda1
                 56G
                      8.4G
                              48G 15% /
free -h
              total
                                                          buff/cache
                                                                        available
                           used
                                        free
                                                  shared
               93Gi
                                        91Gi
                                                    13Mi
                                                                453Mi
Mem:
                          957Mi
                                                                             91Gi
                 0B
                              0B
                                          0B
Swap:
```

Up to ~9.6% more memory available for jobs

Image Building Best Practices

Pull a base image and shell into it to make changes

```
wwctl image import docker://ghcr.io/warewulf/warewulf-node-images/rocky-linux:8.10 my-image
wwctl image shell my-image
```

Recommended: Write a Containerfile to build an image

Benefits of Using a Containerfile

Reproducible recipe of how your production image is built

. Changes to the image are trackable with Git

Instead of upgrading packages or drivers in chroot, can build

new image – provides backout plan or any changes

Innovating Image Builds with Make

- Use a Makefile to streamline the process building (multiple) images
- Makefile can setup the build environment for you
 - i.e. download drivers, copy /etc/passwd and /etc/groups into the cwd
- Makefile can control variables to produce similar images
 - i.e. driver versions, repo information, and package lists
- Example: Building images for CUDA (x86_64 and aarch64) and ROCM

Innovating Image Builds with Make

```
Simplified Example. See the full script on GitHub
NVIDIA VERSION ?= 555.42.02
                                                                   github.com/jeburks2/warewulf-extras/
MLX VERSION ?= 23.10-3.2.2.0
cuda: TAG := sol-x86-rocky8-cuda-$(NVIDIA VERSION)
        @podman build $(PODMAN ARGS) \
                --file ./Containerfile.cuda \
                --build-arg NVIDIA VERSION=$(NVIDIA VERSION) \
                --build-arg MLX VERSION=$(MLX VERSION)-rhel8.10-x86 64 \
                --volume $(PWD):/mnt:0 \
                --tag $(TAG):$(DATE)
        @podman save $(TAG):$(DATE) --output $(TAG).$(DATE).tar
        @echo "wwctl image import --syncuser $(PWD)/$(TAG).$(DATE).tar $(TAG).$(DATE)" >> $(INSTALL_TMP)
gracehopper: TAG := sol-arm-rocky9-cuda-$(NVIDIA_VERSION)
        @podman build $(PODMAN_ARGS) \
                --file ./Containerfile.gracehopper \
                --build-arg NVIDIA VERSION=$(NVIDIA_VERSION) \
                --build-arg MLX VERSION=$(MLX_VERSION)-rhel9.4-aarch64 \
                --volume $(PWD):/mnt:0 \
                --tag $(TAG):$(DATE)
        @podman save $(TAG):$(DATE) --output $(TAG).$(DATE).tar
        @echo "wwctl image import --syncuser $(PWD)/$(TAG).$(DATE).tar $(TAG).$(DATE)" >> $(INSTA
```

Multi-arch Image Management

Warewulf can easily manage nodes with different

Bootstrap your head node with QEMU and binutils to run multi-arch

sudo podman run --rm --privileged multiarch/qemu-user-static --reset -p yes

Build wwclient for different arches

git clone https://github.com/warewulf/Warewulf ; cd warewulf
GOARCH=arm64 PREFIX=/ make wwclient

Use a ContainerFile that is as similar as possible

Multi-arch Image Management

Add cpuArch tags to your nodes

```
wwctl node set gracehopper --tagadd=cpuArch=aarch64
wwctl profile set baseline --tagadd=cpuArch=x86_64
```

Template files based off arch (slurmd.service.ww unit file)

```
ExecStart=/packages/apps/slurm-{{ .Tags.cpuArch }}/sbin/slurmd --systemd $SLURMD_OPTIONS
```

Rendered out for each node

```
ExecStart=/packages/apps/slurm-aarch64/sbin/slurmd --systemd $SLURMD_OPTIONS
ExecStart=/packages/apps/slurm-x86_64/sbin/slurmd --systemd $SLURMD_OPTIONS
```

DNS Management

Warewulf automatically handles DNS on its nodes via /etc/hosts

```
wwctl overlay show --render sc001 hosts /etc/hosts.ww
10.139.121.1 sc001 sc001.sol.rc.asu.edu
10.139.121.2 sc002 sc002.sol.rc.asu.edu
10.139.121.3 sc003 sc003.sol.rc.asu.edu
10.139.121.4 sc004 sc004.sol.rc.asu.edu
10.139.121.5 sc005 sc005.sol.rc.asu.edu
[...]
```

But what about non-Warewulf booted nodes?

DNS Management with Technitium API

```
#!/usr/bin/env python3
import subprocess, requests, csv, dns.resolver
DNS SERVER = "192.168.120.100"
                                             Simplified Example. See the full script on GitHub
resolver = dns.resolver.Resolver()
                                                       github.com/jeburks2/warewulf-extras/
resolver.nameservers = [DNS SERVER]
cmd = "wwctl node list --net | grep -i ethernet | awk '{print $1\",\"$4}'"
[...] # cmd to CSV using StringIO; Processing to set ip, fqdn, zone
current_ip = resolver.query(fqdn, "A")[0].to_text()
if current ip != ip:
  requests.post(f"http://{DNS_SERVER}:5380/api/zones/records/add", params={
  "token": "your api key",
  "domain": fqdn,
  "zone": zone,
  "type": "A",
  "ipAddress": ip,
})
```

Best Practices & Lessons Learned

- Use git to version control overlays
- Be very mindful with overlay pathing and permissions
- Put as much as possible in overlays
- Warewulf can set IPMI at boot with IPMITool – take advantage of this
- Use static DHCP template, with "deny unknown"

- Use git to version control your Container Files
- Create images that are as generic as possible
- Use Containerfiles to generate your node images
- Use Make to build multiple images

Next Steps for ASU

- Expand contributions to the Warewulf project
- Optimize stateless disk-full deployments using Dracut
- Explore NetBox API to have nodes register in NetBox on boot

Questions?

Contact Info

Josh Burks
HPC Systems Analyst
Arizona State University
josh.burks@asu.edu

Supplemental Material

github.com/jeburks2/warewulf-extras



