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

Josh Burks
HPC Systems Analyst
Research Technology Office
Arizona State University

Presented at SC24 November 19, 2024

ASU Research Computing
Arizona State University

Let’s dive into

Warewulf @ ASU

3 Custers – 759 Nodes Total

- Sol – 212 Nodes (Flagship Cluster, Top500 – #438)
- Phoenix – 521 Nodes (Heterogeneous Cluster)
- Aloe – 26 Nodes (Secure Environment)

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

Image source: https://github.com/warewulf/warewulf/blob/main/warewulf-logo.png

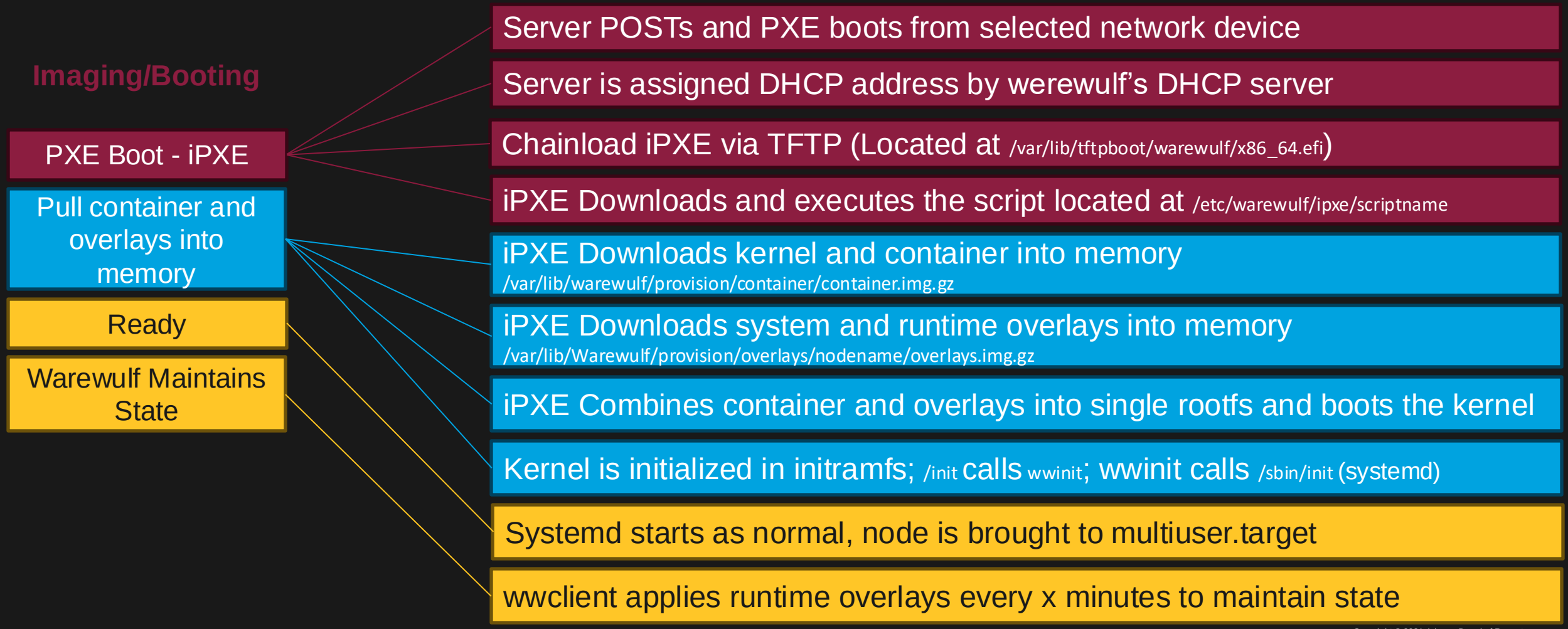
Why did we move to Warewulf?

- **Consistency**
  - **Minimizes configuration drift for predictable performance.**
- **Scalability**
  - **Streamlined growth of compute resources.**

Stateful Provisioning with Cobbler+Salt

Stateless Provisioning with Warewulf

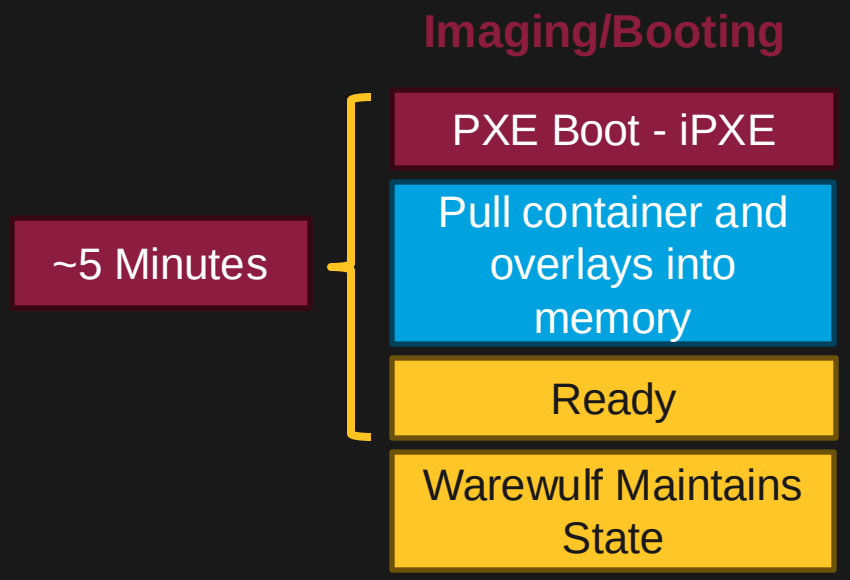
Stateless Provisioning with Warewulf



What makes ASU's deployment innovative?

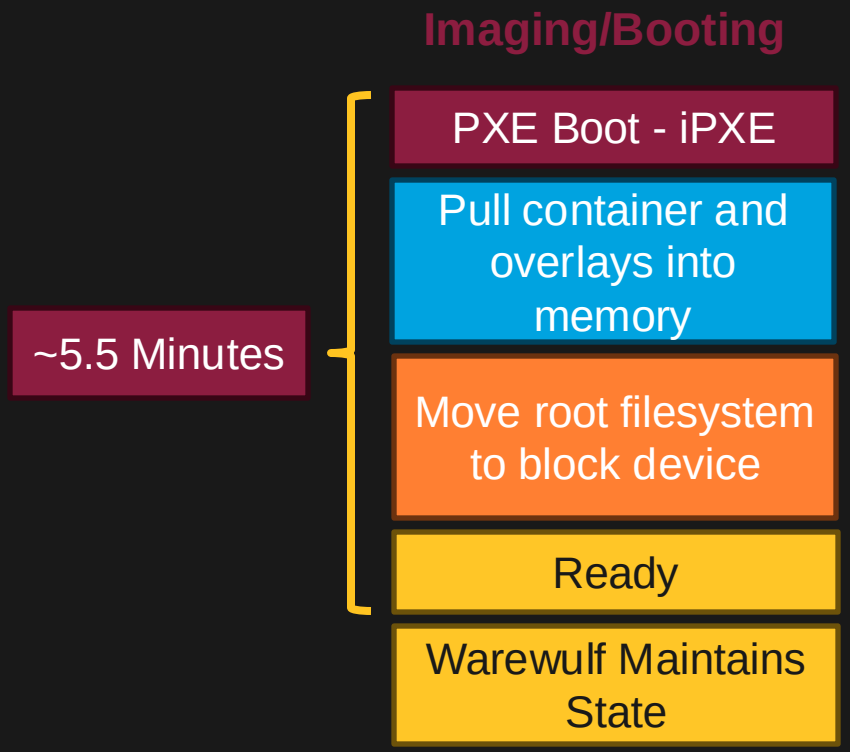
ASU Knowledge Enterprise
Arizona State University

Stateless Provisioning with Warewulf



Stateless Provisioning with Warewulf

Utilizing disks to reduce memory footprint



Stateless Provisioning with Warewulf

Imaging/Booting

- PXE Boot - iPXE
  - 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
- Pull container and overlays into memory
  - iPXE Downloads kernel and container into memory
    /var/lib/Warewulf/provision/container/prod.img.gz
  - iPXE Downloads system and runtime overlays into memory
    /var/lib/Warewulf/provision/overlays/nodename/overlays.img.gz
  - iPXE Combines container and overlays into single rootfs and boots the kernel
  - Kernel is initialized in initramfs; /init is called
- Move root filesystem to block device
  - /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
- Ready
- Warewulf Maintains State
  - wwclient applies runtime overlays every x minutes to maintain state

Moving the Root Filesystem to Disk

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

Moving the Root Filesystem to Disk

Simplified Example. See the full script on GitHub
github.com/jeburks2/warewulf-extras/

```
#init
[…]
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/wwinit
else
        echo b > /proc/sysrq-trigger || /sbin/reboot
```

Measuring Memory Metrics

8 GiB Container	X	Sol	Phx	Total
		212 Nodes totaling 114 TiB RAM	521 Nodes totaling 86 TiB RAM	733 Nodes totaling 200 TiB RAM
		WW containers use 1.7 TiB (1.5 %)	WW containers use 4.1 TiB (4.7%)	WW containers use 5.8 TiB (2.9%)

By moving the rootfs to disk, we give 5.8 TiB back to users for computing

Measuring Memory Metrics

```
WWROOT=tmpfs

df -h /
Filesystem      Size  Used Avail Use% Mounted on
wwroot           47G  7.9G   39G  17% /

free -h
               total        used        free      shared  buff/cache   available
Mem:            93Gi       1.0Gi        84Gi       7.8Gi       7.9Gi        83Gi
Swap:            0B          0B          0B
```

83 GiB Free

```
WWROOT=xfs

df -h /
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        56G  8.4G   48G  15% /

free -h
               total        used        free      shared  buff/cache   available
Mem:            93Gi       957Mi        91Gi        13Mi       453Mi        91Gi
Swap:            0B          0B          0B
```

91 GiB Free

Up to ~9.6% more memory available for jobs

Container Building Best Practices

Pull a base container and shell into it to make changes

```
wwctl container import docker://ghcr.io/Warewulf/Warewulf-node-images/rocky-linux:8.9 my-first-container
wwctl container shell my-first-container
```

Recommended: Write a Containerfile to build a container

```
FROM ghcr.io/warewulf/warewulf-rockylinux:9.2

RUN dnf -y install \
        gcc \
        python3-devel \
        kernel-{core,devel,headers,modules-extra} \
        ...
    && dnf clean all

RUN /mnt/mlnxofedinstall --distro rhel8.2 --skip-repo --kernel --hpc)
RUN /mnt/NVIDIA-Linux-x86_64-550.78.run –s –k 4.18-5.13 –systemd ––no-dkms
```

Example only. Cut down snippit from https://github.com/warewulf/warewulf-node-images/blob/main/examples/rockylinux-9-mofed/Containerfile

Benefits of Using a Containerfile

- Reproducible recipe of how your production container is built
- Changes to the container trackable with Git
- Instead of upgrading packages or drivers in chroot, can build new container

Innovating Container Builds with Make

- Use a Makefile to streamline the process building (multiple) containers
- 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 containers
  - i.e. driver versions, repo information, and package lists
- Example: Building containers for CUDA (x86_64 and aarch64) and ROCM

Innovating Container Builds with Make

Simplified Example. See the full script on GitHub
github.com/jeburks2/warewulf-extras/

```
NVIDIA_VERSION ?= 555.42.02
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:O \
                --tag $(TAG):$(DATE)
        @podman save $(TAG):$(DATE) --output $(TAG).$(DATE).tar
        @echo "wwctl container 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:O \
                --tag $(TAG):$(DATE)
        @podman save $(TAG):$(DATE) --output $(TAG).$(DATE).tar
        @echo "wwctl container import --syncuser $(PWD)/$(TAG).$(DATE).tar $(TAG).$(DATE)" >> $(INSTALL_TMP)
```

Multi-arch Container 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 container file that is as similar as possible

Multi-arch Container 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 unit file)

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

Rendered out for each node

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

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

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
- Do not modify built in “generic” or “wwinit” overlays, create your own
- 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 containers that are as generic as possible
- Use Containerfiles to generate your container
- Use Make to build multiple containers

Next Steps for ASU

- Compare provisioning speeds using Warewulf server on a physical node vs a virtual machine
- Optimize stateless disk-full deployments using Dracut
- Expand contributions to the Warewulf project

Questions?

Contact Info

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

Supplemental Material
github.com/jeburks2/warewulf-extras

ASU Knowledge Enterprise
Arizona State University

Check out Arizona Research Computing Booth #4315
Check out ASU's HPC Dashboard Presentation Friday, 9am B312-B313A