# Anchor: Diskless Cluster Provisioning Using Container Tools

Joseph Voss
*Oak Ridge National Laboratory*
Knoxville, TN
vossjm@ornl.gov

***Abstract*—Large scale compute clusters are often managed without local disks to ease configuration management across several hundred nodes. This diskless management frequently relies on a collection of in-house scripts designed to build client compute images. By leveraging container building tools matured by the tech community we can reduce internal technical debt while allowing cluster installations to be more flexible and resilient. Deploying container images to compute clusters remains an unsolved problem however. To this end we present Anchor, an extensible initrd module designed to boot clusters from an immutable squashfs image with a read-write overlay.**

**The code referenced in this paper is available at https://github.com/olcf/anchor.**

***Index Terms*—diskless, containers, booting, system administration**

## I. Introduction

One of the challenges in large scale installations is to ensure that the entire cluster is in a consistent and known state. A common cluster management strategy involves diskless provisioning, where the nodes hold no local state and instead are re-provisioned from a known image at boot. Typically this is done either by using a copy-on-write file system or a read-only network share. In both of these cases the booted cluster has a strong dependency on an external management node, and thereby has a single point of failure that could bring the entire system down. Additionally, creating and managing these images and related infrastructure has historically been non-trivial.

Several vendors have created their own tools to solve these problems, such as xCAT from IBM or CSM from Cray. At the National Center for Computational Sciences (NCCS), we have managed our commodity clusters using a tool maintained internally called GeDI [1]. However, these tools are different approaches to address the same root problem; creating an OS file system and configuring the nodes on boot. In the years since these technologies were created similar problems were encountered by the developers of linux containers. By borrowing from mature industry container tools to build an OS file system we gain several distinct advantages:

- **Reducing technical debt:** Instead of relying on a system of in house scripts to create chroots and client images, we can leverage existing widely-used and supported technologies and reduce the software we are responsible for maintaining.
- **Centralized image store:** Images do not live (and die) as a file on a single host. Instead they are published to an internal registry and able to be downloaded to another host easily. Image access is controlled through user authentication.
- **Software stacks are shippable:** Since images are in an easily shared format we can publish planned software upgrades ahead of time, allowing users to stage and rebuild their applications if necessary.
- **Images are immutable:** Once an image is created and published it cannot be changed. The image can be tracked by the hash of its contents, always providing a stable image to use for running new machines. This forces immutable infrastructure, ensuring system reliability and consistency.
- **Image builds are completely repeatable:** Each build input and step is tracked through a Dockerfile, which can then be managed through version control software and saved in the image metadata. Output images are tagged with the specific software and inputs they were built with.

## II. Methods

This new deployment method (Anchor) used at NCCS focuses on separating the different stages of cluster management into chunks that can be managed and then scaled up individually. The services required for booting are moved in line with the rest of the services the center provides and intend to use industry standard APIs and protocols wherever possible. These tools need to joined together during the early boot stages of each compute node, and therefore have to be present in the initial RAM disk served over the network. As such, the main software created for this effort is a dracut module to glue together this functionality into our initial RAM disks

At boot this dracut module focuses on bootstrapping trust for the node, fetching a compressed squashfs image, and mounting this read-only image locally with a tmpfs overlay file system. This allows us to have a generic base image configured at boot for a cluster, but still leaves the root file system writable for post-boot configuration steps.

The two steps that interface the most with a center's infrastructure at this stage in boot are trust bootstrapping and image fetching. Trust bootstrapping verifies the client compute node to avoid leaking secrets while distributing the operating system image. After the node is authenticated, images can be downloaded from the image store and mounted.

These steps are intended to be extensible, allows for different centers or teams to pick and choose deployment methods that best fit their use case. Anchor currently supports secret bootstrapping via solving challenges for the automated certificate management engine (ACME) [3] used by Let's Encrypt, or waiting for an SSH connection to manually put a client certificate in place. The image download step can then use that certificate to either download a pre-squashed image from a HTTPS file server, or a un-compressed image from a Docker registry directly and compress it live it during boot.

### A. Building images

Compute node image builds are a perfect example of a process that varies widely from center to center based on their chosen infrastructure and configuration management tools. For the commodity clusters at NCCS we rely heavily on Puppet to both configure our compute images and manage our booted nodes. Prior to this though, we need to have a base OS image to configure.

Every patching cycle we build a new base operating system image using Buildah [2]. The specific process is similar to what GeDi uses for initial chroot creation, except we do not need to handle anything associated with the chroot creation. A simple `buildah mount $(buildah from scratch)` will create a new empty build container and mount it on the host operating system. We then use a separate yum config to point at the latest OS snapshot repositories, change the install root to inside of the build container, and install the base RedHat packages needed for a functional system. This works well to bootstrap the new image and create a foundation for all resulting configuration needed for compute node images.

From here the same base image can pulled from and used as the starting point for each clusters' compute images. Since we use Puppet we just needed to write a build script using Buildah to run Puppet in the container. Puppet then installs and configures everything else needed for the image. Finally the image is then saved to a image repository.

### B. Deployment

*1) Serverless deployment:* On our smaller clusters we host the services needed to boot in our on-site Kubernetes cluster. After POST-ing, a node PXEs from our center-wide DHCP server. This server points the node to Matchbox, a service that maps nodes to different boot scripts based on the machine attributes. From this iPXE boot script the node downloads it's kernel and initial RAM disk containing the anchor dracut module. Once unpacked, the node requests an x509 certificate from our Step-CA server [4], but can communicate with any ACME provider. The ACME provider issues a new host certificate for the node, which is then used to fetch an image from a Docker Registry via a mutual TLS proxy. The mutual TLS proxy verifies that the node's host certificate is correct and that it is authorized to fetch the image that it is requesting. With the image downloaded to the node, the node uses Buildah to extract and mount the container image, and then creates a read-only squashed file system from that image.

*2) Management Server:* The serverless deployment process described above works well for our smaller clusters, but once a cluster has multiple tens of nodes we found it necessary to use a local management server to provision the cluster. While each of the services described above can be scaled easily in Kubernetes, at a certain point we saturate the network links to the Kubernetes client nodes. Additionally scaling services like the Docker registry to serve hundreds of nodes concurrently and efficiently is not trivial [5]. For our local cluster we host an internal DHCP and matchbox server, and instead rely on bootstrapping authentication solely through DNS hostnames. If a node is booting and requests it's iPXE boot script, we start a systemd service to periodically try to SSH into the initial RAM disk and provide it a x509 client certificate. Once the node receives the certificate it stops the SSH server and downloads a squashed image from a file server.

## III. Conclusion

The process described above has been widely adopted by our group with NCCS, and has greatly reduced the amount of maintenance we need to do on our in-house provisioning methods. While this deployment method relies more on our configuration management to finish provisioning each host at boot, the over-all image build and boot process is more straight forward compared our previous solution. Additionally this new deployment tool is flexible and intended to support multiple image build and post boot provisioning strategies. In the future we would like to extend the provided early-boot module to use Trusted Platform Modules (TPMs) and hardware-based encryption keys to bootstrap trust from a node's hardware itself instead of relying on external services. For our management-server-less clusters we would also like to invest more resources into a dedicated and distributed image registry, which would decrease boot times and allow larger clusters to be managed like this.

I would like to thank all of my co-workers who helped make this project a reality.Clay England, thank you for allowing me the freedom to go off and spend weeks if not months re-working how we handle operations within our group. Jesse Hanley and Stefan Maerz, thank you for providing the technical and security background to help get to some solution that actually worked in additional to being performant and secure.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility.

## References

[1] M. Minch, *Generic Diskless Installer*. (2005). Available: https://sourceforge.net/projects/gedi-tools/

[2] *Buildah*. (2020). Available: https://buildah.io

[3] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten. (2019). ”Automatic Certificate Management Environment (ACME)”, RFC 8555. Available: https://tools.ietf.org/html/rfc8555

[4] *Smallstep*. (2020). Available: https://github.com/smallstep/certificates

[5] W. Yiran, C. Gibb. ”P2P Docker Image Distribution in Hybrid Cloud Environment with Kraken”. Presented at KubeCon + CloudNativeCon Europe 2019. Available: https://kccnceu19.sched.com/event/MPcz