Grendel: Bare Metal Provisioning System for High Performance Computing

Andrew E. Bruno
aeb Bruno2@buffalo.edu

Center for Computational Research
University at Buffalo
Buffalo, New York

Salvatore J. Guercio
sguercio@buffalo.edu

Center for Computational Research
University at Buffalo
Buffalo, New York

Doris Sajdak
djm29@buffalo.edu

Center for Computational Research
University at Buffalo
Buffalo, New York

Tony Kew
tonykew@buffalo.edu

Center for Computational Research
University at Buffalo
Buffalo, New York

Matthew D. Jones
jonesm@buffalo.edu

Center for Computational Research
University at Buffalo
Buffalo, New York

ABSTRACT

The Center for Computational Research (CCR) at the University at Buffalo has developed Grendel: a fast, easy to use, bare metal provisioning system for High Performance Computing (HPC). Grendel simplifies network booting racks of compute nodes by providing a robust PXE boot server, rest API, and node management in a single binary for easy installation. In this paper, we describe CCR's HPC network architecture and how Grendel was used to provision the center's Linux based compute clusters. We also present some modern features built into Grendel including automatic host discovery, deploying Live OS images to bare metal compute nodes, and delivering kernel, initramfs, and other provisioning assets using access tokens and trusted HTTPS.

CCS CONCEPTS

• **Social and professional topics** → *Computing equipment management*; • **Networks** → *Network management*; • **Information systems** → *Computing platforms*; • **Computer systems organization** → *Client-server architectures*.

KEYWORDS

bare metal provisioning, HPC administration, network booting, linux clusters

ACM Reference Format:

Andrew E. Bruno, Salvatore J. Guercio, Doris Sajdak, Tony Kew, and Matthew D. Jones. 2020. Grendel: Bare Metal Provisioning System for High Performance Computing. In *Practice and Experience in Advanced Research Computing (PEARC '20)*, July 26–30, 2020, Portland, OR, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3311790.3396637>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
PEARC '20, July 26–30, 2020, Portland, OR, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6689-2/20/07.
<https://doi.org/10.1145/3311790.3396637>

1 INTRODUCTION

The Center for Computational Research (CCR) provides a high performance computing (HPC) environment and on-premise cloud infrastructure to support research at the University at Buffalo. CCR's compute resources [4] include three HPC clusters containing a heterogeneous array of hardware including over 1,500 nodes with over 32,500 processor cores, various flavors of GPUs, Infiniband, OmniPath, and Ethernet networks, a 3PB GPFS scratch system, and a 1.7PB EMC Isilon enterprise storage system. Five system administrators support this infrastructure that provides more than 1 PFlop/s of peak performance compute capacity. High performance computing clusters can be challenging to build, operate, and maintain especially in large heterogeneous environments. After the hardware hits the floor, getting up and running can be a daunting task particularly for small to medium-sized research centers with minimal support staff or funding for commercial products. Automated, network based operating system (OS) installation and configuration is a requirement for any HPC cluster both for efficiency and to ensure consistency across the cluster. The Pre-Boot Execution Environment (PXE) specification [15] defines the process of booting an operating system over a network and involves a coordinated dance of several networking protocols. In HPC, compute nodes are commonly PXE booted to perform an automated installation of Linux using available methods depending on the distribution: Kickstart (RedHat) or Preseed (Debian). In this type of deployment, called Stateful (or diskful), the OS is installed to each compute node's local disk. In the case of Stateless (or diskless) deployments, an image of the OS is created and deployed to each compute node via PXE booting an initramfs, ramdisk, or a filesystem mounted via NFS (nfsroot [11]). Setting up a PXE boot server often requires installing and configuring several different network services such as a DHCP, TFTP and DNS. Bare metal provisioning systems are tools that provide the necessary infrastructure for PXE booting along with managing hosts and operating system images.

Efforts such as OpenHPC [16] aim to build a reference collection of HPC software components and best practices for use in administering HPC clusters. The recommended systems specifically designed for HPC provisioning such as xCAT [5] and Warewulf [6] are large feature rich systems with steep learning curves. Similar

tools exist outside of HPC such as Foreman [10], Cobbler [9], and OpenStack Ironic [7] but are more focused on managing generic data center resources or in the case of Ironic require an OpenStack cloud environment. Prior to deploying Grendel, CCR used Foreman to provision its HPC clusters.

Bare metal provisioning systems can be challenging to administer. Many systems are designed to auto-configure back-end services such as ISC DHCPD, BIND, and `in.tftpd` to support the network boot provisioning process. This requires a manual or automatic sync process to update configurations and restart each of the distinct services. PXE and TFTP support is often provided by auto-generating `pxelinux.cfg` files naming them after the hosts' MAC address. This is required because the `in.tftpd` service doesn't know about the hosts configured in DHCP as they are completely separate services. Most tools also require running external database engines such as MySQL or Postgresql which is another service that needs to be administered. xCAT has a concept of management and service nodes that can be deployed in a hierarchical cluster for scalability but comes with the cost of added administrative complexity.

In this paper we present Grendel, a bare metal provisioning system specifically tailored for HPC Linux clusters. Grendel aims to be a modern, easy to use alternative for organizations that don't require the exascale capabilities of xCAT or Warewulf and just want to network boot a rack of servers quickly and with minimal setup. Grendel provides an implementation of the network protocols required to provision Linux systems over a network in a single binary. This simplifies installation and deployment as there is only a single service to manage. Grendel also provides a CLI and rest API for managing hosts and OS images. Grendel aims to fill the gap between the large feature complete systems like xCat and Warewulf and more general life cycle management systems like Foreman and Cobbler that lack features necessary for HPC. The following sections will describe a high-level overview of Grendel architecture, a basic HPC network architecture for building out clusters, and how Grendel was used to provision CCR's computational resources.

2 GRENDAL OVERVIEW

Grendel is written in Go and uses an embedded ACID compliant key/value database for managing hosts and OS images. Grendel can serve all networking protocols required for network booting: PXE, DHCP, ProxyDHCP, TFTP, HTTPS, DNS. iPXE [14], an open source network boot firmware is bundled with Grendel and chainloaded via TFTP during the provisioning process. Grendel includes a command line interface (CLI) and rest API for managing hosts using a `nodeset` syntax commonly used in HPC environments. Other features include auto-discovering hosts, controlling BMC/iDRAC interfaces using RedFish and IPMI, and anaconda kickstart template files. A high level overview of the network boot process supported by Grendel is shown in Figure 1.

A simple HPC cluster setup consists of a rack of nodes configured to PXE boot connected to an Ethernet switch. The nodes send DHCP boot requests to a DHCP server in order to obtain the NBP (network bootstrap program) via the PXE boot process. Grendel chainloads the iPXE firmware and boots an OS image consisting of a Linux kernel and set of `initrd`'s. Grendel uses a simple JSON based file format for defining nodes and boot images. Assuming Grendel is

installed on the same subnet as the nodes or the switch is setup to relay DHCP packets to the Grendel server, nodes can be booted simply by creating two JSON files and running Grendel as shown in Figure 2.

Grendel's host management features are specifically designed for managing clusters of compute nodes. Adding, updating, and provisioning commands in Grendel can be performed on a set of nodes using a `nodeset` syntax comparable to what's found in popular HPC tools like ClusterShell [17]. For example, we can set a rack of 100 nodes `cpn-d13-01` through `cpn-d13-100` to provision on next boot and then reboot them using their baseboard management controller (BMC) from a single Grendel command:

```
$ grendel bmc netboot --reboot cpn-d13-[01-100]
```

Grendel can be deployed in several different ways depending on which services are required. Any combination of the supported network services can be enabled/disabled. For example if you already have an existing DHCP server, Grendel can run as a ProxyDHCP server only and will serve PXE boot requests but not provide any DHCP leases. Used in combination with the built-in HTTP server, Grendel can serve up the provisioning assets such as iPXE boot scripts, kernels, `initrds`, and anaconda kickstart files.

3 BASIC HPC NETWORK ARCHITECTURE

The design and implementation of an HPC cluster depends on many factors and each cluster will most certainly have a unique set of requirements. In this section we describe the basic Ethernet network architecture CCR has used to provision many of its clusters. CCR uses a basic "Top-of-rack (ToR) switching" network architecture design in a two-tier topology consisting of a single core and edge tiers, as shown in Figure 3. In this type of design, compute nodes, switches, PDUs, and other components are located within the same (or adjacent rack) and are connected to an in-rack network switch commonly located at the top or in the middle of the rack. The in-rack network switch is then connected to a core aggregation switch (or switches) that connect the rest of the data center [1]. Once all the physical components are racked, cabled and powered on, the individual compute nodes need an operating system installed and configured. In order to install the OS over the network, we first need to obtain basic information about each compute node such as hostname, IP address, and MAC Address of the primary network interface. Below are the details of CCR's IP addressing and host naming conventions and how we use Grendel to automate what was previously a labor intensive process.

3.1 Host Naming Scheme

CCR uses a custom host naming scheme that maps hostnames to physical locations within the data center. This enables admins to easily locate nodes for service. The host naming convention is as follows: `ROLE-RACK-UNIT[-SLOT]`. Where `ROLE` is a three letter prefix that defines the type of component: `cpn`=compute node, `swe`=ethernet switch, `pdu`=power distribution unit. Following the `ROLE` is the physical rack in which they are located. Racks are named based on their location (row/aisle) in the data center. CCR's server room is laid out in a grid format mapped out in a X-Y coordinate system based on the floor tiles. One axis is labeled A-Z, the

Figure 1: High level overview of network booting using Grendel. 1. Compute node issues DHCP PXE boot request, Grendel replies with static lease and TFTP location of NBP iPXE firmware. 2. Compute node requests NBP file via TFTP and chainloads iPXE firmware. 3. iPXE firmware is loaded and issues HTTPS request for iPXE boot script. Boot script provides location of kernel/initrds which are fetched via HTTPS. Compute node is then booted and any other assets for example anaconda kickstart files or Live images are fetched over HTTPS. 4. Grendel also serves forward and reverse DNS records for any hosts configured with a fully qualified domain name (FQDN).

other is labeled numerically which provides an alphanumeric designation for each rack on the floor, similar to data center standards such as ANSI/TIA-606-B and TIA-942. For example, rack "d13" is located in row "D" aisle "13." The last component of the name is the unit within the rack followed by an optional "slot." The slot is for compute node chassis that have more than one node within the same rack unit. For example, the Dell C6420s are 2U servers with 4 compute nodes (2 in each rack unit). A complete example of a one node unit would be "cpn-d13-18," which is a compute node located in rack d13 unit 18. An example of a multi-unit node or blade server might be "cpn-d13-20-01" and "cpn-d13-20-02" - one physical server with 2 blades/nodes.

3.2 IP Addressing Scheme

To assign IP addresses to each compute node the following scheme is used by CCR: 10. [Net]. [Switch]. [Port]. Where Net is the class B subnet for each high level grouping of services: 10.64.0.0/16 = HPC compute nodes and 10.16.0.0/16 = BMC management network, and 10.32.0.0/16 = Core services. This allows for easily

```
# images.json
[{
  "name": "flatcar",
  "kernel": "flatcar_production_pxe.vmlinuz",
  "initrd": [
    "flatcar_production_pxe_image.cpio.gz"
  ],
  "cmdline": "flatcar.autologin"
}]

# hosts.json
[{
  "name": "cpn-d13-18",
  "provision": true,
  "boot_image": "flatcar",
  "interfaces": [
    {
      "fqdn": "cpn-d13-18.ccr.buffalo.edu",
      "ip": "10.64.25.36",
      "mac": "41:69:AE:E6:61:06"
    }
  ]
}]
```

```
$ grendel serve --hosts hosts.json --images images.json
```

Figure 2: Simple network booting of nodes with Grendel. Shows JSON format for boot images and hosts and the basic command to run all Grendel services.

defining network based ACLs to control access to/from compute nodes and various other network segments. As an example, all compute servers reside on 10.64.0.0/16, a management network 10.16.0.0/16 is used for the BMC devices for out of band management of the nodes, and 10.32.0.0/16 supports core services like DHCP, DNS, and LDAP.

Each ToR switch will have a unique number based on the port its plugged into on the core switch. For example, the ToR switch in rack d13 is plugged into port 25 on the core switch so we'll number this 25. Each switch will have its own VLANID and subnet for each network. Continuing with our example, the compute network switch number 25 will have VLANID 1025 and subnet 10.64.25.0/24. For the BMC network, switch number 25 will have VLANID 3025 and subnet 10.16.25.0/24.

Each node in rack d13 will be assigned an IP address based on the port its plugged into on the ToR switch. As a complete working example, the compute node physically located in rack d13 unit 18 plugged into port 36 on ToR switch 25 would have IP address 10.64.25.36.

3.3 Auto-Discovery of Hosts

Because CCR's naming and IP addressing scheme depends on knowing the exact location of nodes in a rack, it's required that we capture the MAC address of each server and assign the appropriate location-based hostname and IP address prior to the provisioning process. After the compute nodes are powered on for the first time

Figure 3: CCR ToR network architecture and the basic IP and host name addressing scheme

they will typically attempt to network boot via PXE (though this is dependent on how the BIOS is configured in the factory). The ToR switches are configured to relay DHCP packets to one or more DHCP servers in the data center. At CCR, we configure all ToR switches to relay DHCP requests to our Grendel server. Grendel can now be used to automatically harvest network information for each compute node. Grendel queries the ToR switch using SNMP to obtain a table of MAC address, port number, and VLANID. Using the mapping of hostname to port number on the switch we created when physically cabling each rack, we automatically load the new hosts into Grendel. A detailed example is provided in Appendix A. If we're not concerned with mapping host names to physical locations or don't have a particular IP addressing scheme, we can use Grendel's DHCP server to auto-discover hosts. In this case, we assume we have a rack of hosts continuously sending out DHCP boot requests. Grendel can be run to capture the MAC addresses of the requests and assign them a hostname and IP address in a contiguous manner. Grendel also supports importing hosts from an existing `dhcpd.leases` file. This was especially useful for migrating over racks that were provisioned using CCR's old Foreman system. We used the `dhcpd.leases` file created by Foreman's DHCP smart proxy to import directly into Grendel.

4 OS IMAGE PROVISIONING

Once the nodes have been auto-discovered, assigned a hostname, IP address, and loaded into Grendel an OS image is installed next time the node PXE boots. There are a few approaches to this commonly found in HPC. Diskful image provisioning includes deploying an OS image that is installed directly to the nodes' disks. Stateless provisioning involves deploying OS images that are not installed on disk but rather reside completely in memory or use `nfsroot`.

Grendel is entirely agnostic to how images or hosts are installed as it supports any kind of kernel/initrd network booting. This allows Grendel to be readily adopted into any existing methodology used to deploy OS images. You define your boot images using a simple JSON file with the paths to the kernel and initrd files and Grendel will PXE boot the nodes. You can also define custom kickstart files that will be served by Grendel for automated diskful installations.

The approach CCR uses for deploying OS images leverages LiveOS images. We use a tool called `mkosi` [8] to build bare metal images which uses `systemd-nspawn` and `yum` installroot under the hood, similar to the approach discussed in [2] with the major difference being the way we package the image. Instead of using a CPIO based `initramfs` image, we build a SquashFS image containing a root filesystem image. This image within an image format is described further here [12], and is similar to the Live CD's shipped by CentOS and other RedHat based systems. Support for PXE booting Live images is provided by `dracut`'s `livenet` feature. `Dracut` [13] is a tool used to create `initramfs` images and the `livenet` module allows retrieving a rootfs over HTTP. This method speeds up the boot process for large images by leveraging HTTP transport and is supported in both CentOS and Debian based distributions. At CCR, we PXE boot compute nodes using our custom built live image by adding `dmsquash-live` and `livenet` `dracut` modules to the `initramfs`. The URL to the live image served from Grendel is defined via kernel command line arguments within the iPXE boot script. A detailed example is provided in Appendix B. For diskful deployments, we kickstart nodes using the same Live image by modifying our kickstart template in Grendel to install from the live image as opposed to a package based install from `yum` repositories. This allows us to test out development images on nodes without kickstarting them and then use those same images to kickstart nodes once they're ready for production. To date we've tested building CentOS 7, CentOS 8, Ubuntu 18.04 (Bionic Beaver), Ubuntu 20.04 (Focal Fossa) images and successfully deployed these to compute nodes using Grendel. The workflow CCR uses to build and update OS images and their related kernels is open source and available at <https://github.com/ubccr/grendel-images>.

5 AUTHENTICATED AND TRUSTED PROVISIONING

TFTP (Trivial File Transfer Protocol) is used for downloading the initial boot strap program during PXE. This protocol has no built in support for encryption or authentication and is inherently an insecure protocol. HTTP is often used during the provisioning process to transfer assets such as the iPXE boot script, kernel, `initrd`, and kickstart files. In HPC, the provisioning process is typically performed on a trusted internal network where encryption and authentication are not always strictly necessary or required. Grendel can optionally be configured to authenticate requests to deter unauthorized access. Grendel implements all network protocols required to PXE boot nodes providing direct control over every aspect of the provisioning process. On the back-end, Grendel authenticates all requests to TFTP and HTTP using Branca tokens [3]. For each DHCP PXE boot request a Branca token is generated with an expiry time of 1 hour by default allowing only TFTP requests with a valid token to be served. The iPXE boot firmware will only be

chain loaded to authorized clients with valid Branca tokens using TFTP. In addition, provisioning assets such as kernels, initrds, and kickstart files served over HTTP are also authenticated in the same manner.

The iPXE boot firmware can optionally be built with HTTPS support and include the certificate authority (CA) used for the Grendel server certificate allowing trusted HTTPS for the remainder of the boot process. The custom iPXE firmware will request an iPXE boot script over HTTPS (instead of HTTP) along with the kernel and initrds. Additionally, iPXE also supports code signing, allowing the authenticity and integrity of files downloaded by iPXE to be verified using certificates. For instance, kernel and initramfs images can be digitally signed and Grendel configured to use the `imgverify` iPXE command to verify the files downloaded by iPXE are authentic. Additionally, when kickstarting nodes the anaconda initramfs can be augmented to include the same CA allowing the entire kickstart process to be served over trusted HTTPS.

6 CONCLUSIONS

Grendel is an easy-to use and administer bare metal provisioning system designed specifically for HPC. It fills the gap between commercially available systems that might be unaffordable and existing opensource large-scale projects that might require too much configuration and maintenance for an understaffed HPC center. Grendel is open source software and combines the services required for network installation of operating systems into one binary, making it easy to deploy at any size HPC center. Grendel's flexible architecture allows for the integration of existing services that may already be deployed at an HPC center making it ideal for updating and improving current provisioning workflows without having to reconstruct the whole process. While Grendel was specifically designed for HPC environments, it has potential use cases in any data center environment with its ability to improve the provisioning process by scaling down the setup and management requirements. Grendel is released under the GPLv3 license and available at <https://github.com/ubccr/grendel>.

7 FUTURE WORK

Grendel is a new project under heavy development. Future work includes adding features for node groups, custom host attributes, and developing plugins for more BMC, PDU, and switch devices. A web based front end is also planned to allow managing Grendel using a web browser. Future work will also include additional enhancements for stateless image and post installation modification, while preserving the simplicity of Grendel's approach.

ACKNOWLEDGMENTS

This material is supported by resources at the Center for Computational Research (<https://buffalo.edu/ccr>) at the University at Buffalo. Partial support through NSF Award 1724891: "MRI: Acquisition of High Performance Computing Infrastructure to Support Computational and Data-Enabled Science and Engineering" and NIH Award S10OD024973: "High Performance Data and Computing Infrastructure".

REFERENCES

- [1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.
- [2] John Blass and John Roberts. 2018. Stateless Provisioning: Modern Practice in HPC. In *In HPCSYSPROS18: HPC System Professionals Workshop*. Dallas, TX. https://github.com/HPCSYSPROS/Workshop18/tree/master/Stateless_Provisioning_Modern_Practice_in_HPC
- [3] Branca. 2020. Branca. <https://branca.io/>.
- [4] Center for Computational Research, University at Buffalo. 2020. UB CCR Support Portfolio. <http://hdl.handle.net/10477/79221>.
- [5] Bruce Potter Andy Wray et al. Egan Ford, Jarrod Johnson. 2020. xCAT. <http://xcata.org/>.
- [6] Gregory M. Kurtzer et al. 2020. Warewulf. <http://warewulf.lbl.gov/>.
- [7] Julia Greger et al. 2020. Ironic OpenStack project. <https://github.com/openstack/ironic>.
- [8] Lennart Poettering et al. 2020. mkosi. <https://github.com/systemd/mkosi>.
- [9] Michael DeHaan et al. 2020. Cobbler. <https://cobbler.github.io>.
- [10] Foreman. 2020. Foreman. <https://www.theforeman.org/>.
- [11] Nico Schottelius Horms Gero Kuhlmann, Martin Mares and Chris Novakovic. 2018. Mounting the root filesystem via NFS (nfsroot). <https://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt>. Online; accessed: 2020-02-16.
- [12] Harald Hoyer. 2013. dracut. http://www.kernel.org/pub/linux/utils/boot/dracut/dracut.html#_booting_live_images. Online; accessed: 2020-02-16.
- [13] Harald Hoyer. 2013. dracut. <http://www.kernel.org/pub/linux/utils/boot/dracut/dracut.html>. Online; accessed: 2020-02-16.
- [14] iPXE. 2020. iPXE. <https://ipxe.org/>.
- [15] M. Johnston and Ed. S. Venaas. 2006. *Dynamic Host Configuration Protocol (DHCP) Options for the Intel Preboot eXecution Environment (PXE)*. RFC 4578. RFC Editor. <https://doi.org/10.17487/rfc4578>
- [16] Karl W Schulz, C Reese Baird, David Brayford, Yiannis Georgiou, Gregory M Kurtzer, Derek Simmel, Thomas Sterling, Nirmala Sundararajan, and Eric Van Hensbergen. 2016. Cluster computing with OpenHPC. In *In HPCSYSPROS16: HPC System Professionals Workshop*. <http://hdl.handle.net/2022/21082>
- [17] Stéphane Thiell, Aurélien Degrémont, Henri Doreau, and Aurélien Cedeyn. 2012. ClusterShell, a scalable execution framework for parallel tasks. In *Linux Symposium*. Citeseer, 77.

A HOST DISCOVERY USING RACK SWITCH

Here we provide an example of switch based automatic host discovery method supported in Grendel. After physically cabling rack d13 we create a simple text file of hostname to port number mapping:

```
# d13-ports.txt
cpn-d13-18 36
cpn-d13-19 34
```

The MAC address table for the ToR switch in d13 is queried using SNMP. An entry in the MAC address table looks this:

```
{
  "entry-type": "dynamic",
  "if-name": "ethernet1/1/36",
  "mac-addr": "41:69:ae:e6:61:06",
  "status": "ACTIVE",
  "vlan": "vlan1025"
}
```

The port number is obtained by parsing the "if-name" and the VLANID is stored in the "vlan" attribute. At CCR we use the following convention: each ToR switch is numbered and corresponds to a /24 subnet and a tagged VLAN of 1000 + switch number. To automatically import hosts into Grendel's database the following command is used:

```
$ grendel discover switch \
-s 10.64.0.0 \
-e swe-d13-22.cb1s.ccr.buffalo.edu \
-m d13-ports.txt \
-d ccr.buffalo.edu
```

Where `-s 10.64.0.0` is the network for compute nodes, `-e` is the hostname of the ToR switch, `-m` is the host-to-port mapping, and `-d` is the domain to use for FQDNs. Using the MAC address entry above, Grendel computes the IP address `10.64.25.36` from the subnet `10.64.0.0`, switch number `25` based on the VLANID - `1000`, and port number `36`. Grendel parses the hostname from the hostname-to-port text file and generates the following JSON to be imported into Grendel's key/value database:

```
{
  "name": "cpn-d13-18",
  "interfaces": [
    {
      "fqdn": "cpn-d13-18.ccr.buffalo.edu",
      "ip": "10.64.25.36",
      "mac": "41:69:AE:E6:61:06"
    }
  ]
}
```

B BOOTING LIVEOS IMAGES

To build a Live OS image we use `mkosi` [8]. `mkosi` is a wrapper around `yum` `--installroot` and `systemd-nspawn`. It supports several output formats for the resulting image including a plain directory containing the root file system. All the code for building images for use with Grendel can be found at <https://github.com/ubccr/grendel-images>. Here we provide an example boot image JSON file and resulting iPXE script generated by Grendel for booting live OS images. After building our SquashFS live image from the `mkosi` output directory we add the `dracut-livenet` module to `initramfs.img` and create a boot image JSON file that looks like this:

```
{
  "name": "centos7",
  "kernel": "/var/grendel/centos7/vmlinuz",
  "liveimg": "/var/grendel/centos7/squashfs.img",
  "initrd": [
    "/var/grendel/centos7/initramfs.img"
  ]
}
```

Where `liveimg` is the SquashFS live image we built from the output of `mkosi`. We add the `dracut livenet` module to the `initramfs.img` using post install hooks provided by `mkosi`. The post install runs a command equivalent to the following:

```
$ dracut --add "livenet"
    --force initramfs.img KERNEL_VERSION
```

When a compute node PXE boots, Grendel will generate an iPXE script to instruct the node to boot the Live OS:

```
#!ipxe
kernel --name kernel
    http://192.168.10.254/boot/file/kernel
initrd --name initrd0
    http://192.168.10.254/boot/file/initrd-0
boot kernel initrd=initrd0
root=live:http://192.168.10.254/boot/file/liveimg
```

Additionally, we can kickstart a compute node using the same Live OS image in diskful deployments. The iPXE script generated by Grendel to instruct the compute node to kickstart:

```
#!ipxe
kernel --name kernel
    http://192.168.10.254/boot/file/kernel
initrd --name initrd0
    http://192.168.10.254/boot/file/initrd-0
boot kernel initrd=initrd0
ks=http://192.168.10.254/_/kickstart
network ksdevice=bootif ks.device=bootif
inst.stage2=http://192.168.10.254/repo/
```

And the corresponding anaconda kickstart file generated by Grendel:

```
install
liveimg --url="http://192.168.10.254/boot/file/liveimg"
lang en_US.UTF-8
selinux --disabled
keyboard us
timezone --utc America/New_York
skipx
network --bootproto dhcp
    --hostname cpn-d13-18.ccr.buffalo.edu
    --device=de:ad:be:ef:12:8c
```