Kubernetes for HPC Administration

ANONYMOUS AUTHOR(S)

HPC software stacks require administrators to deploy and maintain a complex collection of software. Historically this was done on bare-metal nodes, which present challenges when maintaining a coherent suite of software. This paper describes hosting administrative HPC software on a Kubernetes platform, along with auxilary tooling for storage, IP/DNS management, metrics and logging. Since Kubernetes exclusively runs container images, individual components of the software stack run in simplified, reproducible environments. Altogether, the platform can replicate most of the advantages of a bare-metal environment, while improving on reproducibility, stability, uptime, and the quality of telemetry.

Additional Key Words and Phrases: Kubernetes, HPC, DevOps, Cluster, Administration, Container, Orchestration

ACM Reference Format:
Anonymous Author(s). 2021. Kubernetes for HPC Administration. 1, 1 (September 2021), 0 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software stacks used in HPC Administration require numerous components, such as batch schedulers, telemetry and cron tasks. For many years this was done on bare-metal nodes, resulting in an unwieldy collection of stateful services, which are difficult to maintain, update, or recover in the case of a failure. In recent years, Kubernetes[19] has become a popular industry platform for deploying containerized workloads. It consists of a set of services installed on several nodes to produce a virtual cluster, and it has a demonstrated track record of strengths such as support for high availability, automatically scaling workloads, integration with storage systems, and a large ecosystem of third party software. While intended for cloud environments where scalability and uptime are maximized, uses for these capabilities can be found in many other spaces. A bare-metal Kubernetes cluster was deployed to host and manage administrative services commonly deployed on HPC clusters. The following describes some of the implementation options, the choices we made, and some additional insight into the architecture.

2 PRIOR WORK

2.1 Bare-Metal Provisioning

The simplest strategy for deploying software stacks is to use a bare-metal node, meaning services are installed directly on top of a host operating system (OS) without any additional abstractions. Industry has produced various tools over the years to manage bare-metal deployments. Tools such as Puppet[29] and Chef[3] can assist in the creation of specialized OS images, rollout of updates and patches, and stand up instances of services on running clusters. Ansible[1] is popular python-based tool that uses YAML files to define operations on collections of servers.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under contract DE-NA0003525.

2.2 Virtual Machines

Virtual Machines (VM) present another strategy for provisioning workloads, wherein system resources are abstracted away by a hypervisor layer which sits beneath the guest OS. Inside the VM's containment, a guest operating system, complete with its own kernel, can run the administrative workloads. VMs allow for software control over resource allocation, secure isolation, image state snapshotting, and faster power cycle times. They are ubiquitous across industry in cloud products, as well as *on-premesis* deployments such as OpenStack[25]. VMs are often used in combination with the previously mentioned bare-metal provisioning tools. In single instance use-cases, products like VMWare[39] and Virtualbox[26] provide desktop user interfaces to manage instances; and tools like Vagrant[37] are designed to provide a meta-wrapper for VM environments to improve environmental interoperability.

2.3 Docker Swarm

Docker Swarm is a multi-node version of the Docker runtime built into the Docker Engine software distribution. Given nodes with docker already installed, it is trivial to set up and supports multi-master runtimes for high-availability. For a time this tool was increased in popularity in lockstep with Docker containers, but was ultimately supplanted by Kubernetes as the dominant multi-node container orchestration platform. This was largely due to limited functionality, limited customizability, and limited support for third party extensions. Among its advantages over Kubernetes is that the environment is much more streamlined, as it's built into the Docker engine so all of the dependencies are met as soon as Docker is present. It also uses Docker Compose syntax to define software stacks, which produces a much more compact YAML syntax than the kubernetes equivalent.

Docker Swarm merits particular mention in this analysis because it was initially chosen as the container orchestration platform for hosting the HPC stack. As the software stacks grew more sophisticated, the challenges for delivering services with the desired characteristics grew more difficult. Eventually the decision was made to switch to Kubernetes, then an up-and-coming platform which was becoming very popular in industry. Although the switch was not trivial, as the ecosystem is far more complex than Swarm, its fast development cycle, rich feature set, and ubiquitous industry support justified the migration.

3 IMPLEMENTATION CHOICES

3.1 Kubernetes: Git, YAML, and Containers

Managing the multitude of services and configurations that sustain a cluster is a challenging and error-prone process. Kubernetes provides mechanisms for configuring the cluster in a reproducible and trackable way. At the lowest level, intructions are communicated to an API server using a RESTful API[41]. Kubernetes provides *kubectl*, a tool for interacting with the API server over commandline. *kubectl*[18] can also ingest YAML files for more complex workflows. Kubernetes relies on standardized YAML constructs to define the components of a runtime such as the network, storage infrastructure, configuration files, and lifecycle of each workloads. Thus Git can be used to version control almost every part of Kubernetes' configuration. In practice, one initial shortcoming was that sensitive information commonly found in configurations (such as passwords for SQL databases) could not be kept the Git repository due to the risk of the server hosting the upstream Git project leaking information to unauthorized users. This was solved by using a Mozilla tool called SOPS[35] to encrypt sensitive values, and a decrypting PGP[9] key was distributed to authorized developers. SOPS stores encrypted secrets in YAML files with a Git-compatible string encoding. Common tools such as Kustomize[20] (integrated into *kubectl*) and Helm support SOPS, with the consequence that adopting the encrypted format does not significantly change the command line workflow when using these tools. While this solution worked well for a project with just a few developers, larger projects may do better using a much more sophisticated tool such as HashiCorp Vault[38].

Administrators spend considerable amounts of time maintaining bare-metal software stacks due to conflicts with the host operating system or the dependencies of other software also running on the system. Industry's answer to this was the *container*, which cleverly encapsulates much of a host operating system's environment without the overhead of a virtual machine. Kubernetes expects all workloads to run in container images, which ensures compatibility of the software with every node in the Kubernetes cluster.

The containers used in this deployment came from two origins. The first are developed by third parties and hosted on external registries (such as Quay.io or Docker Hub). The remaining containers were in-house creations and initially pushed into a docker registry deployed inside the Kubernetes cluster. However, the internal registry presented a bootstrapping conflict in subsequent re-deployments of the cluster as many services are not able to start until their respective containers were built and pushed into the internal registry. It also restricted developers who wanted to access or update container images in use by the Kubernetes cluster. Finally, Docker registry's security architecture was more primitive than other platforms, simply requiring a bearer token with no automatic way to update, track, or invalidate keys. Later on the container images were moved into a corporate Gitlab registry, which had the benefit of a more sophisticated security model and it allowed the images to be available to be pulled without depending on the availability of an internal service.

3.2 Provisioning

Many Kubernetes provisioning tools were researched before deploying the Kubernetes cluster. Kops[17] and Terraform[36] could be eliminated without experimentation because they are intended for clouds and not for provisioning a bare-metal cluster. k3s[16], Minikube[40], and Docker Desktop[7] were eliminated because they were not designed for highly available and/or multi-node production environments. Other providers were vendor specific, such as Openshift[33] from Red Hat and MicroK8s[23] from Canonical, which was not preferred in this use case.

Of the remaining options, Kubeadm[4] and Kubespray[6] were selected as the best choices for this use case. Kubeadm was tried first, because it was a first-party tool and had the simplest workflow since all of the functionality was wrapped up in a single command-line tool. In practice, however, it has several shortcomings. First, it does not have out-of-the-box support for provisioning multiple masters in a highly available configuration as it expects to be provided an external load balancer for traffic to the API server. A Kubernetes load balancer can be thought of as a reverse-proxy that distributes traffic across reachable API hosts. Expecting a load balancer as part of the environment is a reasonable requirement when provisioning with a cloud provider, but that isn't the case for a bare-metal system. Moreover, it does not automatically handle many of the idiosyncratic local steps required to tune a bare-metal node into working Kubernetes node, such as disabling swap or modifying various sysctl IP rules. Due to these shortcomings, Kubespray was finally selected for provisioning the cluster. Kubespray is an open source python project that uses Ansible to provision a Kubernetes cluster out of a list of bare-metal nodes. It was a bit more difficult to manage because the configuration sprawls across many files, but unlike Kubeadm it correctly tunes system settings for the common Linux distributions. Kubespray also implements a reverse proxy on each node so communication with the highly available API server does not require an external load balancer.

3.2.1 Kustomize and Helm. Initially, all software was deployed using Kustomize[20]. Kustomize is a template-free method for deploying Kubernetes workloads. It integrates modifications to a YAML configuration by *scoping*, wherein the tool is provided a list of directories full of YAML files, and the contents of the successive directory's YAML files layers its changes onto the previous one. This is a good tool for rapidly developing or modifying a service, since all of the relevant components are laid out in plain YAML in a directory tree. Kustomize does not offer any mechanism for versioning, which is a disadvantage for environments that prefer stability over running the latest developed version of the configuration. Its ability to rollback changes is also limited to the existing declarative semantics provided by Kubectl.

Helm[28] is a self-described "package manager" for Kubernetes. *Packages* consist of tarballs of Jinja2 templated YAML files in a directory tree. Jinja2 is mostly used for text substitution, but it also supports templated loops and conditional logic. Unlike Kustomize, Helm has semantics for *installing*, *upgrading*, and *deleting* deployments, which are tracked using annotations in the upstream YAML bodies. The user may make changes to a Helm package installation at pre-defined locations in the templates with command line flags or a *values.yaml* file. In simple use cases Helm's templating system works well, although it becomes much more unwieldy than Kustomize scopes when more complex changes to the package are required. Since all of the components of a stack are packaged together, this tool has a much stronger use case than Kustomize when the stack is provided by a third party. Future revisions of the cluster plan to make regular use of third party Helm charts in specific cases where the deployment does not change often and no large changes to the deployment have to be made.

3.3 Storage

Many administrative services require persistent filesystem storage; a requirement which exceeds the scope of a container. Data stored in the container's filesystem will be lost if the container is restarted, which may happen from crashes, redeployment, or routine pod migration. Some services are stateless and are not affected by a lack of storage. In some cases a deployment needs a private space for data that should not be directly accessible to other services, such as the backing store for a SQL database. In other cases a deployment may need to attach itself to an existing filesystem to do an operation, such as reading or updating the contents of a */home* NFS mountpoint. Kubernetes supports *provisioners*, which automatically create and mount volumes on demand. The design decisions regarding stateful services were made with the backing filesystem stores in mind.

For services that need a private share, Kubernetes was configured to interface with an existing external Ceph[2] cluster to dynamically create Rados Block Device (RBD) volumes. Ceph RBDs are a form of network-attached storage that mount as block devices on the client node's filesystem, unlike NFS which manifests as a POSIX filesystem mount. A deployment that needs a volume defines a *PersistentVolumeClaim* (PVC) with the volume's desired characteristics. The first time Kubernetes handles a new RBD, the provisioner will format it with a filesystem, then mount it in a place that the client pod can access. A deployment may be designed to share the RBD-backed volume between multiple pods, such a file server pod and an update cron pod. This presents a problem, because a Kubernetes pod may schedule anywhere in the cluster but Ceph forbids mounting RBDs onto multiple host nodes due to the lack of synchronization primitives for block devices. This can be avoided by configuring the pods with an *affinity*, so they always colocate on the same host.

HPC clusters often have multiple filesystem mounts. When a deployment mounts an extant file system, the PVC is provided with details about the filesystem, such as the host address, subpath, keys, and read-write permissions. Kubernetes can (in some cases requiring third party provisioners) mount filesystems such as CephFS, NFS and GPFS.

4 DEPLOYMENTS

4.1 Static Web Pages

Static web pages were used in several places as views into file trees. Each deployment follows the same trivial template; an nginx container – with a few lines of configuration to change the URL sub-path – serves a web page through the reverse proxy which presents the data mounted from a persistent volume into the container. In cases where the data in the mount changes over time, a small container for exporting metrics to Prometheus was included. These views have several uses, such as organizing data for users to access with a browser, or for hosting package manager repository mirrors for operating systems such as Fedora, Ubuntu, Debian, and Centos. For the latter case, Cron Pods are also defined to periodically update the contents of volumes.

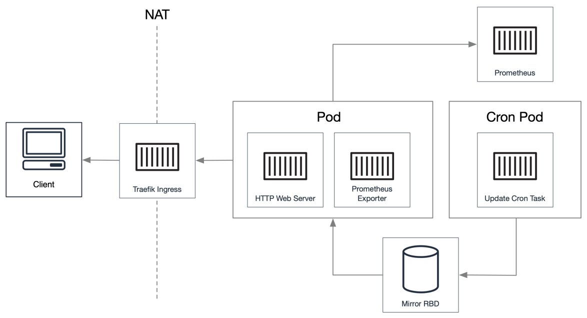

Fig. 1. Diagram of the relationships between components in an instance of a Static Web Page deployment.

4.2 Slurm

Batch schedulers are common in multi-tenant HPC environments because they are specially tuned to maximize throughput of heterogeneous workloads. Slurm is currently the most popular batch scheduling tool deployed on HPC systems. Slurm requires three daemons to operate: a Slurm Controller (*slurmctld*), a Slurm Database (*slurmdbd*), and a Slurm Worker (*slurmd*). In this deployment *slurmd* runs as a systemd service on the bare-metal worker nodes. While *slurmd* could in theory run in pods within the Kubernetes ecosystem, this design choice reflects the preference that tenants of this HPC run codes on bare-metal nodes. *slurmctld* and *slurmdbd* run in Kubernetes pods. Each service was provisioned a volume: one for *slurmctld*'s transaction cache, and one for *slurmdbd*'s SQL database.

Service and Ingress objects were configured to allow for external traffic to *slurmctld* and *slurmdbd*, which are required for communication with user CLI commands and worker nodes. Slurm accounting, which is required for features like fair share scheduling, needs an up-to-date registry of usernames on the system. A separate accounting cron pod periodically runs to collect a user list from the LDAP server and update the Slurm Database.

4.3 Jupyterlab

Jupyterlab[30] is a popular browser-based IDE with tools for writing and executing interactive notebooks. Without administrator support, a user may run a Jupyterlab notebook on a HPC cluster by creating a python virtual environment, installing the Jupyterlab tooling, creating a single-node allocation, launching Jupyterlab on the worker node, forwarding Jupyterlab server ports to the local machine using an SSH tunnel, and finally connecting to the session in a local browser

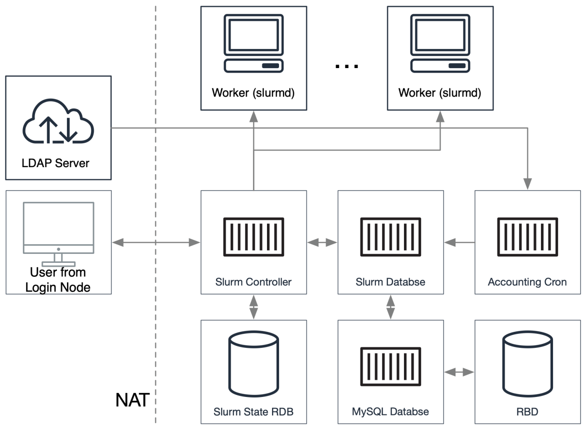

Fig. 2. Diagram of the relationships between components in Slurm deployment.

by accessing the local port. This task is at odds with the simplicity of the Jupyterlab interface and can be improved greatly by cluster-wide tooling.

Jupyterhub[30] provides a common browser-based interface for cluster users to authenticate and launch Jupyterlab sessions. The Kubernetes environment hosts a Jupyterhub pod. In this deployment, user credentials are authenticated by communication with an external LDAP server. Jupyterhub was configured to use two plugins, *batchspawner*[14] and *wrapspawner*[15] to create sessions that provision workloads on Slurm worker nodes. Batchspawner manages Jupyterhub interactions with Slurm with the aid of a templated batch script. ProfilesSpawner adds a dropdown to the UI, from which the user selects from a set of pre-configured launch options, such as deployments with additional RAM or GPU acceleration.

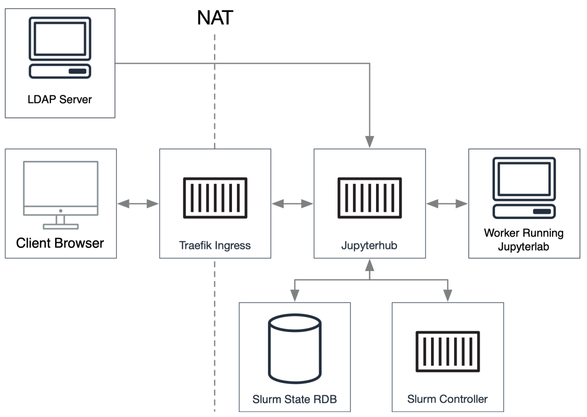

Fig. 3. Diagram of the relationships between components in the Jupyterhub deployment.

4.4 Telemetry

Clusters are generally opaque without specialized monitoring software. Over the years industry developed tooling to improve visibility, particularly into cloud environments.

4.4.1 Logs. Prior to the Kubernetes deployment, log aggregation was accomplished using a *syslog-ng* service running on a bare-metal node. This was risky, because logs were being saved to a local disk, so a downtime to the node would result in gaps in log collection, and the loss of a disk would result in the loss of all stored logs. The Kubernetes replacement mounts a dynamically allocated RBD volume for log storage. In a new final step, the logs are forwarded to a corporate Splunk instance where they are indexed, and queried on-demand to populate dashboards. The pipeline (except for ingesting logs into Splunk) replicates the behavior of the previous pipeline, with the advantage that the workload runs independently of a single node and the logs are stored in a filesystem that will not disappear due to a local disk failure.

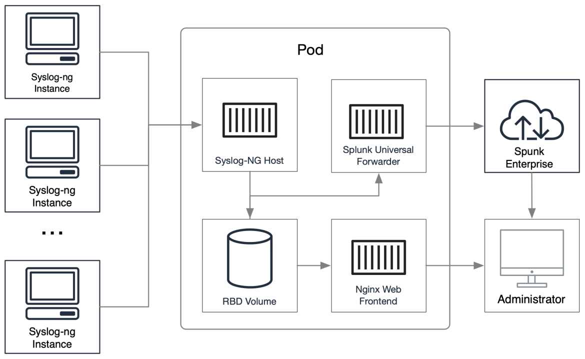

Fig. 4. Diagram of the relationships between components in a log processing deployment.

4.4.2 Metrics. Metrics consist of periodically measured numerical data; the collecting, storing, and displaying of which has been the focus of numerous specialized tools. These tools have gone through several phases in which one solutions becomes quite popular, then supplanted by a new one in a few years time. Among these include Ganglia Monitoring System[8], Graphite[11], and Nagios[24]. After a more detailed investigation of several different options, which included Ovis[27] and InfluxDB[12], the cluster was instrumented using Prometheus[31] and Grafana[10]. Prometheus is a time series database that periodically scrapes endpoints over HTTP and stores the results in a time-series database. Grafana is a web-based dashboard for visualizing metrics from a variety of sources, although it has become the *de facto* Prometheus frontend.

Prometheus ships with mechanisms for automatically detecting and collecting metrics from services in the Kubernetes cluster. Many Kubernetes-native deployments will annotate their service definitions so that Prometheus will detect and scrape them automatically. Bare-metal worker nodes running outside of the Kubernetes environment provide metrics by running *node-exporter*[32] as a systemd service. *node-exporter* hosts an HTTP server on the local node, which Prometheus must be configured to scrape. Ad hoc Grafana dashboards were developed to present consolidated views of node utilization, Slurm status, network switch traffic, filesystem state, and errors. Grafana usually stores this in a local filesystem, but in the interest of making the application stateless and portable, the dashboards were exported as JSON templates and stored in Kubernetes configmaps.

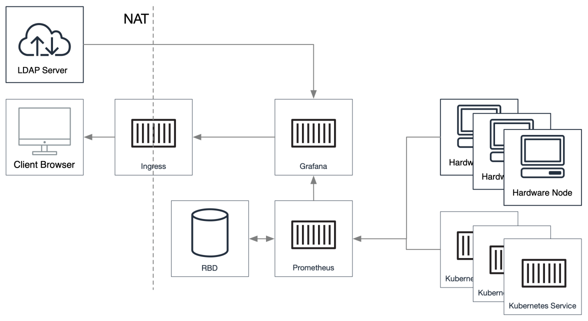

Fig. 5. Diagram of the relationships between Kubernetes components in a metrics processing deployment.

4.5 SSH Reverse Proxy

Clusters often have redundant login nodes, which provide a multi-tenant space for users to enter the cluster. Administrators may configure the DNS to round-robin between the IP addresses of the login nodes with the advantage that users only need to try to connect to a single host-name and get rudimentary load balancing via round-robin. This has two downsides; cluster administrators must have some control over the DNS (which may not be possible depending on corporate policy), and when a login node fails, new connection attempts have a 1/N chance of failing.

Another solution is to use a reverse-proxy, which has a single ingress address to the wider network and internally determines routing decisions to the login nodes. This Kubernetes implementation used HAProxy, which has support for tunable round-robin connections for better load balancing, and can automatically remove unresponsive nodes from the pool.

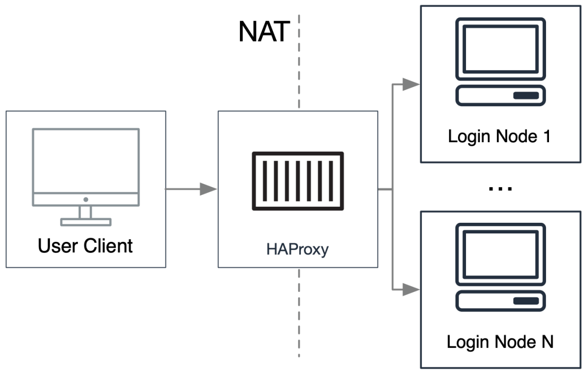

Fig. 6. Diagram of the relationships between components composing the SSH reverse proxy.

4.6 External IP and DNS Management

Prior to porting administrative services to Kubernetes, it could be assumed that each service was reachable at a known IP address and hostname because they ran on bare-metal nodes. Kubernetes' traffic routing created a private network that made services unreachable to external nodes. Even if the IP address could be known and reachable, Kubernetes pods launch on nodes arbitrarily, which would make the address unstable. Moving external services into Kubernetes required a solution that can create a stable IP address and hostname for services that could resemble the previous behavior of bare-metal hosts.

MetalLB[22] is a project that can create dynamic IP address assignments for Kubernetes services. When a Kubernetes deployment defines a service of type *LoadBalancer*, MetalLB will select an unused IP address from a pre-configured pool and promulgate the identity to other (non-Kubernetes) nodes on the network using ARP broadcasts. When an external node tries to send traffic to that IP address, the ARP configuration causes it to route a Kubernetes node where is can be further routed to the host with the target pod. Should the need arise, services can be designed to ensure a specific IP is chosen in the allocation by specifying pools with only one IP in the range.

With a solution to IP assignment in view, there is still the issue of DNS. Ideally a DNS name could be paired to an IP address as services are created, without the need to manually update entries in a DNS server outside of Kubernetes. *ExternalDNS*[21] is a project that interacts with Kubernetes API to discover services in which the author has annotated a domain name, and updates a DNS-compatible store with the IP-name mapping. CoreDNS was chosen for the backing DNS, and an instance was created in Kubernetes. All worker nodes were directed to route traffic requests to it by updating their `/etc/resolv.conf`, and CoreDNS was configured to upstream the the previous DNS so the prior mappings would not be lost.

4.7 Traffic Management

The Kubernetes platform hosts numerous web pages. Most of the pages are available behind the same domain, but unlike a conventional website, URL sub-paths are not routed to the same web service, but an arbitrary number of web servers running in separate containers. Traffic routing decisions have to be handled by a reverse proxy. Traefik was chosen for this deployment, but Nginx is another popular choice. Traefik has several ways to configure routing so that it can correctly map URL paths or sub-domains to the correct container. In this case, *IngressRoute* YAML stubs were written and included with the deployment over every service that required routing rules.

5 CONCLUSION

Administrators have a long history of building and maintaining HPC systems. One of the challenging components in this process has always been the maintenance and stability of the administrative software stack. With the steady advance of novel orchestration technologies such as Kubernetes, the tooling has made it possible to turn a handful of bare-metal nodes into a dynamically scaling, self-healing cluster whose entire software stack can be tracked using modern version control solutions. Finally, choosing the Kubernetes platform also opens access to an enormous collection of cloud-native software. All of these factors work together to make a strong case to use containerization and Kubernetes for HPC administrative workloads.

A TABLE OF CLI TOOLS

Tool	Description
helm[28]	A command-line tool that terms itself as a 'package manager for Kubernetes'. Provides semantics for *installing*, *updating*, and *removing* software.
helmfile[34]	Command-line tool that allows groups multiple helm installations into a single yaml file
helm-diff[5]	Helm plugin for previewing changes to a helm installation
helm-secrets[13]	Helm plugin that interfaces with SOPS
kubespray[6]	Ansible-based project for provisioning a bare-metal Kubernetes cluster
kubectl[18]	First party commandline tool for interacting with a Kubernetes cluster
kustomize[20]	First party zero-template tool for layering YAML configuration; integrated into *kubectl*
SOPS[35]	Command-line tool for creating and editing encrypted versions of files containing sensitive information

REFERENCES

[1] 2021. Ansible is Simple IT Automation. https://www.ansible.com. (2021).
[2] 2021. Ceph.io - Home. https://ceph.io/en/. (2021). Accessed 2021-08-12.
[3] 2021. Chef Software DevOps Automation Tools & Solutions | Chef. https://www.chef.io. (2021). Accessed 2021-09-14.
[4] 2021. Creating a cluster with kubeadm. https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/. (2021). Accessed 2021-08-12.
[5] 2021. databus23/helm-diff. https://github.com/databus23/helm-diff. (2021). Accessed 2021-08-12.
[6] 2021. Deploy a Production Ready Kubernetes Cluster. https://kubespray.io/. (2021). Accessed 2021-08-12.
[7] 2021. Docker Desktop for Mac and Windows. https://www.docker.com/products/docker-desktop. (2021). Accessed 2021-08-12.
[8] 2021. Ganglia Monitoring System. http://ganglia.sourceforge.net. (2021). Accessed 2021-09-14.
[9] 2021. GNU Privacy Guard. https://gnupg.org. (2021). Accessed 2021-08-12.
[10] 2021. Grafana: The open observability platform | Grafana Labs. https://grafana.com/. (2021). Accessed 2021-08-12.
[11] 2021. Graphite. https://graphiteapp.org. (2021). Accessed 2021-09-14.
[12] 2021. InfluxDB Time Series Platform | InfluxData. https://www.influxdata.com/products/influxdb/. (2021). Accessed 2021-09-14.
[13] 2021. jkroepke/helm-secrets. https://github.com/jkroepke/helm-secrets. (2021). Accessed 2021-08-12.
[14] 2021. jupyterhub/batchspawner. https://github.com/jupyterhub/batchspawner. (2021). Accessed 2021-09-14.
[15] 2021. jupyterhub/wrapspawner. https://github.com/jupyterhub/wrapspawner. (2021). Accessed 2021-09-14.
[16] 2021. K3s: Lightweight Kubernetes. https://k3s.io. (2021). Accessed 2021-08-12.
[17] 2021. kOps - Kubernetes Operations. https://kops.sigs.k8s.io. (2021). Accessed 2021-08-12.
[18] 2021. kubectl. https://kubernetes.io/docs/reference/kubectl/kubectl. (2021). Accessed 2021-08-12.
[19] 2021. Kubernetes. https://kubernetes.io. (2021). Accessed 2021-08-12.
[20] 2021. Kubernetes Native Config Management. https://kustomize.io. (2021). Accessed 2021-08-12.
[21] 2021. kubernetes-sigs/external-dns. https://github.com/kubernetes-sigs/external-dns. (2021). Accessed 2021-08-12.
[22] 2021. MetalLB :: MetalLB, bare metal load-balancer for Kubernetes. https://metallb.universe.tf/. (2021). Accessed 2021-08-12.
[23] 2021. MicroK8s - Zero-ops Kubernetes for developers, edge and IoT. https://microk8s.io. (2021). Accessed 2021-08-12.
[24] 2021. Nagios - The Industry Standard in IT Infrastructure Monitoring. https://www.nagios.org. (2021). Accessed 2021-09-14.
[25] 2021. Open Source Cloud Computing Software - OpenStack. https://www.openstack.org. (2021). Accessed 2021-09-14.
[26] 2021. Oracle VM Virtualbox. https://www.virtualbox.org. (2021). Accessed 2021-09-14.
[27] 2021. ovis-hpc/ovis. https://github.com/ovis-hpc/ovis. (2021). Accessed 2021-09-14.
[28] 2021. The Package Manager for Kubernetes. https://helm.sh. (2021). Accessed 2021-08-12.
[29] 2021. Powerful Infrastructure Automation and Delivery | Puppet. https://puppet.com. (2021). Accessed 2021-09-14.
[30] 2021. Project Jupyter | Jupyterhub. https://jupyter.org/hub. (2021). Accessed 2021-09-14.
[31] 2021. Prometheus - Monitoring system time series database. https://prometheus.io/. (2021). Accessed 2021-08-12.
[32] 2021. prometheus/node_exporter. https://github.com/prometheus/node_exporter. (2021). Accessed 2021-09-14.

[33] 2021. Red Hat OpenShift makes container orchestration easier. https://www.redhat.com/en/technologies/cloud-computing/openshift. (2021). Accessed 2021-08-12.
[34] 2021. robol/helmfile. https://github.com/roboll/helmfile. (2021). Accessed 2021-08-12.
[35] 2021. SOPS: Secrets OPerationS. https://github.com/mozilla/sops. (2021). Accessed 2021-08-12.
[36] 2021. Terraform by Hashicorp. https://www.terraform.io. (2021). Accessed 2021-08-12.
[37] 2021. Vagrant by Hashicorp. https://www.vagrantup.com. (2021). Accessed 2021-09-14.
[38] 2021. Vault by Hashicorp. https://www.vaultproject.io/. (2021). Accessed 2021-08-19.
[39] 2021. VMware - Delivering a Digital Foundation to Business. https://www.vmware.com. (2021). Accessed 2021-09-14.
[40] 2021. Welcome! | Minikube. https://minikube.sigs.k8s.io/docs/. (2021). Accessed 2021-08-12.
[41] 2021. What is a restful API. https://www.redhat.com/en/topics/api/what-is-a-rest-api. (2021). Accessed 2021-09-14.