Deploying production-grade Kubernetes cluster

Vojdan Kjorveziroski*, Panche Ribarski†
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje
Email: *vojdan.kjorveziroski@finki.ukim.mk, †panche.ribarski@finki.ukim.mk

***Abstract*—Deploying a production-grade Kubernetes cluster is a challenging feat, mainly because of all the different services that need to be integrated with each other. We examined Kubespray, which is an open-source project whose purpose is to automate the deployment of stable clusters, as well as ease the future administration and lifecycle management along the way. Additional components can be installed during the deployment process that enrich and improve the cluster capabilities. We used Kubespray and together with other open-sources projects we created production-grade Kubernetes cluster ready for real deployment scenarios.**

***Index Terms*—Kubernetes, K8S, Docker, Kubespray, Helm, ELK, monitoring, automation, containers, orchestration**

I. Introduction

Nowadays there is an increasing trend in the computer engineering world to adopt the cloud as a native place for the execution of many workloads, ranging from high performance computations to hosting lightweight demo student projects made as part of some course assignment. There are many reasons that support this notion of using someone else's infrastructure for your work. One of them is the increased reliability, since the end user is no longer in charge of managing the hardware and affordability, because the primary method of billing is pay-as-you-go, meaning that users are obliged to pay only for the resources that they have used as many cloud providers even have the option of billing per minute or even per second for certain services. Other reason, scalability, in terms of available resources is no longer a problem, additional storage or memory is just a few clicks away. However, many companies and even individuals are either not satisfied with the current offerings from the major cloud providers, have industry or policy bound restrictions that prevent them from using public clouds, think that they can implement a more cost-effective solution by using their own hardware or simply want to have more control over the whole process and learn something new along the way. In all these cases, the usual options of implementing the infrastructure for hosting multiple applications or services is to either deploy individual virtual machines and assign them to different teams or departments or go with the container route, where each application will reside in its own dedicated environment. No matter what option is chosen, some abstraction layer in terms of software will have to be deployed that will need to manage the underlying hardware and allow for easy and convenient provisioning and deprovisioning of either virtual machines or containers. These days containers are the preferred choice for many workloads, mainly because they are lightweight and easily disposable, yet at the same time provide sufficient isolation. Manual deployment of containers quickly becomes a burdensome process, so a container orchestration system is almost always needed. Kubernetes[1][2] has distinguished itself as a leader in this space, mainly because of its extensibility, large community and large feature set. However, all of these benefits result in a complex installation process, large amounts of prerequisites and dependencies, and elaborate lifecycle management. The common solution to the aforementioned problems is to have a well-tested automated solution which can offer a reproducible and stable deployment each time it is invoked. There are multiple such solutions when it comes to the installation of Kubernetes, one being Kubespray[3], an open-source project that is centered around Ansible[4][5] playbooks and roles that provisions production grade clusters with minimal end-user interaction. The purpose of this paper is to explain the benefits of using Kubespray while exploring the customizability that it offers and options for the future lifecycle management of the deployed clusters. First, a general overview of the Kubernetes platform is given, and then the Kubespray project is explained in details, along with possible additions to the cluster that improve its functionality, either deployed by Kubespray itself during the initial setup, or manually at a later point in time.

II. Automated deployment of production grade clusters

Kubernetes is a container orchestration platform that can operate on top of many container runtimes, most popular of which are Docker[6][7] and Rkt. A deployment of an application is done using multiple manifests that represent YAML based text files, where using a predetermined syntax and options the desired environment for the application is described. These manifests are then submitted to the Kubernetes API server and are eventually translated to low level actions such as bringing up a container or issuing a storage request to a remote system. The API server is extensible, so cluster administrators can define custom resource types that include their own logic and interact with systems that are not natively supported.

Since the deployment of a Kubernetes cluster involves many dependencies and is a complicated feat, multiple open source projects exist whose purpose is to make the process easier. One such example is Kubespray, a solution based on Ansible playbooks and roles that automate every aspect of the Kubernetes cluster deployment. The end-user is only required to install the dependencies that include a couple of Python libraries and Ansible itself in order to deploy a cluster.

A. Cluster requirements

Kubernetes clusters are comprised of two types of nodes, masters and workers. Master nodes are hosting the Kubernetes API server and are responsible for managing the cluster. In order to achieve high availability, multiple master nodes can be deployed and an uninterrupted cluster operation will be ensured as long as a majority of monitors can be formed. Workers can be added or removed from the cluster at any point in time, depending on the resource requirements. The worker nodes are directly controlled by the master nodes. As a persistent backend the etcd key-value store is used and all the cluster information including the deployed manifests are stored there. Similar to the Kubernetes API server, etcd can also operate in a high-avialability mode, if more than half of the nodes are available. Since etcd also has official Docker images available, the cluster operator can choose to either run it in a standalone fashion, installed as any other software package or inside a Docker container.

Publishing of web applications is done using an Ingress controller[8] which effectively acts as a reverse proxy to any container that exposes an HTTP port. The configuration of the ingress controller, such as what endpoints to use, whether SSL is enabled, what redirects to use; is done using annotations, simple key value pairs in the YAML file that describes the Ingress resource. Since Kubernetes is such an extensible platform, there are multiple Ingress controllers to choose from, for example NGINX[9] or Traefik[10]. The NGINX ingress controller is natively supported by Kubespray. If an application that does not use the HTTP protocol needs to be exposed to the outside world, then there are multiple options, some of which are binding it to a specific worker node and accessing it through it exclusively or, a more attractive and versatile option is to deploy a load balancer[11] addon which can automatically assign either public or private IP addresses from a given subnet. One such addon is MetalLB[12], which from recently is also supported by Kubespray and can automatically be provisioned. Whenever a LoadBalancer resource is defined within the Kubernetes API, a private IP address from a predefined range is assigned to the service. Furthermode, BGP is also supported by MetalLB, in environments where assignment of public IP addresses is needed.

Networking is a major topic when it comes to Kubernetes cluster deployments, since there are so many options and extensions available. An important thing to note is that the chosen network plugin is responsible for the isolation of the different containers running in the cluster. Kubernetes natively supports a NetworkPolicy[13] resource type that can be specified like all other resources, using a YAML file. With this, granular network rules for an application can be specified, such as what ports are opened and from what other containers or namespaces they can be accessed. More details regarding networking plugins and deployment options are given in the Additional tools section.

B. Initial setup

Once the project has been downloaded from the source code repository, an inventory file needs to be created in which the IP addresses of all the nodes that will take part in the clusters will be entered. Kubespray exploits the concept of groups that Ansible provides, where multiple hosts can be put together in a group so that they can be referenced using a common name, in order to designate the various roles that the nodes will have inside the cluster, such as whether they will be master nodes, worker nodes, etcd nodes, or have some other role. Once the inventory has been defined, the cluster deployment can be customized by editing the default values of the various predefined parameters. Using simple switches and statements, the Kubernetes version can be specified, along with the desired networking plugin, container runtime, ingress controller, and storage addons. Finally, with the execution of the Ansible playbook, Kubespray will perform any prerequisite checks on the nodes and install all of the missing requirements before deploying the cluster. After the script has finished with its execution, the administrator should be able to login into the master nodes and have a fully functional Kubernetes cluster.

C. Lifecycle management

Kubespray would not be so enticing if it did not support some sort of lifecycle management, ensuring that the cluster functions properly after it has been deployed. The following non-distrubtive operations are possible with Kubespray: addition and removal of worker, master and route reflector nodes, API server certificate renewal and rotation, cluster upgrade and cluster tear-down. The most beneficial of these is the automatic cluster upgrade, where not only is the Kubernetes version upgraded, but also all of the prerequisites and addons, such as the etcd store, the container runtime and the networking plugins. Furthermore, the addition and removal of worker nodes is simple, fast, and it does not require user intervention, meaning that it can be automated with some monitoring system, where new nodes would be deployed once the cluster load reaches a certain level.

III. Additional tools

In the Cluster Requirements section, we briefly mentioned some of the Kubernetes addons that are supported by Kubespray. In the following paragraphs we provide a more thorough explanation of some of them.

A. Kubernetes package management

The definition of several YAML manifest files for a deployment of a single service can quickly become tiresome, since most of the time the required changes are very small in comparison to the complete files. Helm[14] is a package manager for Kubernetes that tries to solve this problem. It uses the concept of charts, where a chart represents a service that needs to be deployed, such as a content management system (CMS) application. Using a templating language, a set of default parameters are inserted in the manifest files and the user can choose to override only the desired values.

Helm requires a dedicated component, Tiller, to be installed in the cluster itself, and Kubespray fully supports its complete deployment.

B. Storage

There are many options for integrating the Kubernetes cluster with standalone storage systems. Kubernetes has a resource called a persistent volume claim[15], where such a claim can be created by an application and the default storage provisioner that is deployed in the cluster is required to fulfill the request. The way in which the request will be serviced is up to the plugin, whether a new block device needs to be set up, a shared file system mounted or a new bucket on some object storage created. For test clusters an NFS provisioner[16] might be sufficient, where all the volume claims will be serviced by mounting a specific directory from a given NFS server. However, this is not scalable since the performance of NFS is not optimal for such deployments. A popular option is to integrate the Kubernetes cluster with Ceph[17][18], a storage system that offers both block and object storage and a shared file system. Since Ceph is also a distributed application that pools disks present on different hosts, the end-user needs to decide how the Ceph cluster will be managed. One option is to have a separate Ceph cluster, preferably on bare metal machines, and integrate it with the Kubernetes cluster with volume provisioning plugins. These plugins watch for any new volume claims and execute the necessary commands on the Ceph cluster in order to deploy a new block device or mount the shared file system. Another option is to use Rook[19], a Kubernetes native storage solution that deploys a Ceph cluster inside Kubernetes. The initial deployment and future management of the storage cluster is done through YAML files that are then submitted to the Kubernetes API server. One of the main benefits of Rook is that the operation of the storage system is completely abstracted from the end-user, so no knowledge of Ceph is required. However, this can also be seen as a drawback; Kubernetes upgrades now become much more complicated, since bringing down Kubernetes nodes might bring the whole storage cluster down, so careful planning and testing is needed. On the other hand, while administering a standalone Ceph cluster requires much more resources both in terms of people and hardware, it is a more robust solution and provides a nice separation of concerns, where multiple complex systems are not intertwined with each other.

C. Networking

There are two main types of networking plugins available for Kubernetes and they are layer 2 or layer 3. An example of a layer 2 plugin is Flannel[20], which creates a single flat network shared by all future containers. Another option is to use separate networks and with the help of a routing algorithm bridge these together. A plugin that works in this way is Calico[21] and it uses BGP for routing between the different subnets in the overlay network. The benefits of using Flannel or any other L2 plugin is that it is easier to configure and simpler to administer, but it is not as scalable as using an L3 solution. On the other hand, Calico supports the concept of BGP route reflectors, which can be used in very large clusters. In this mode, instead of forming a BGP full mesh topology, a separate node outside the cluster is configured to run as a standalone BGP route reflector and all the cluster nodes peer only with it. For redundancy, multiple route reflectors can be deployed. Both of these plugins as well as a few others are supported by Kubepsray and the user has an option of manually choosing between them.

D. Monitoring and logging

Monitoring of the Kubenetes cluster and the individual services that are deployed as containers within it is an important task and there are many different solutions that can be implemented. One of the most popular options regarding performance metrics is to use Prometheus[22], a dedicated software which crawls different endpoints that expose system metrics. These metrics can then either be persisted in a separate time series database or cached by Prometheus itself. The visualization of the gathered values can be done using a separate web application application, one example being the widely popular Grafana[23]. When it comes to logging, the usual choice not only when working with Kubernetes, but also in other areas is to use the Elasticsearch, Logstash and Kibana (ELK)[24] stack. An important thing to note is that cluster administrators should not only be interested in the logs from the cluster nodes themselves, but also from the containers that are run within the cluster. In order to achieve this, there are multiple Kubernetes addons that are able to send the log output of the containers either directly to Elasticsearch or to Logstash for further processing. One such example is Fluentd[25]. Once the logs have been persisted, graphical visualizations and dashboards can be created using Kibana, which is a web application that connects to Elasticsearch and provides browsing and visualization options for the gathered logs.

IV. Deployment of production-grade cluster with Kubespray on the Nebula platform

We tested Kubespray in an attempt to bring up a production-grade Kubernetes cluster on virtual machines residing on the OpenNebula cloud computing platform. In this section we describe the deployment process, along with the choices that we have made and additional applications that were installed in order to ensure the reliability and availability of the cluster.

A. Hardware

We have chosen a rather minimal initial setup that will offer the option for non-disruptive upgrades to the cluster and tolerate a single node failure, with the possibility to add new nodes as needed. The storage has been completely decoupled from Kubernetes and an external Ceph cluster is utilized. This setup allows both the computational power to be increased by adding new worker nodes, or to improve the resiliency by deploying additional masters, without the need for any major architectural changes. The separation of storage from the

Purpose	CPU	Memory	Storage
Kubernetes Master #1	4	4GB	40GB
Kubernetes Master #2	4	4GB	40GB
Kubernetes Master #3	4	4GB	40GB
Kubernetes Worker #1	8	8GB	60GB
Kubernetes Worker #2	8	8GB	60GB
Ceph #1	2	4GB	40GB
Ceph #2	2	4GB	40GB
Ceph #3	2	4GB	40GB
Ceph #4	4	4GB	40GB + 1TB
Ceph #5	4	4GB	40GB + 1TB
Ceph #6	4	4GB	40GB + 1TB

cluster allows easy maintenance of both the cluster itself and the underlying virtual machines. Additionally, a catastrophic failure in the Kubernetes cluster will not affect the data, all that will be needed to restore normal operation is to recreate the necessary Kubernetes resources and remap the persistent volumes to the RBD volumes on Ceph. If Rook had been used, then this would have been much more complicated, and a Kubernetes failure would have probably meant corrupton of the data as well.

The specification for the virtual machines are:

B. Storage cluster deployment

The storage cluster has been deployed with the officially supported Ansible playbooks from the Ceph community. The process is very similar to Kubespray, where the cluster "skeleton", the roles for each virtual machine, are given in the inventory files and any further tweaks regarding the operation of the cluster is done by modifying the default Ansible variables. We have provisioned six Ceph nodes, where the first three (1-3) serve as both monitor and manager nodes in order to ensure the high-availability of the cluster and the others as object storage device (OSD) hosts, where, currently, each of them hosts a single OSD, with the option of adding additional ones at a later point in time. Additional monitors or managers can also be added whenever needed.

C. Kubernetes cluster deployment

For the initial deployment, five Kubernetes nodes were deployed, where three of them are masters that also function as etcd servers and the rest are worker nodes. No route reflector was used, since the initial number of nodes is low, but it remains an option for future expansion. At least two additional nodes will be required in order to ensure the high-availability of the route reflector. The Kubernetes addons that were installed by Kubespray are: NGINX ingress as the cluster ingress controller, CoreDNS as the in-cluster DNS server, Calico as the networking plugin, Helm and Tiller with TLS support, Kubernetes dashboard for web access to the cluster resources. The communication between the nodes and the Ceph cluster is done through a private VLAN dedicated to the project. Official support for the MetalLB load balancer was added to Kubespray after the cluster was deployed, so manual installation was needed. Regarding the RADOS block device (RBD) provisioner that allows the mapping of Kubernetes volumes to RBD images, there is an active pull request[26] that is yet to be merged into the main branch of Kubespray at the time of this writing. Once it is approved and merged, that process could be automated as well.

D. Additional tools and addons

In order to be production ready, we needed a way to monitor the cluster resources, receive alerts for any anomalies, and centralize the logging for all the different services that are installed. To achieve this, we deployed Prometheus-Operator in order to monitor the cluster and receive alerts, and the EFK stack (Elasticsearch, Fluentd, Kibana) to centralize the logs.

Prometheus operator[27] is a project that offers Kubernetes native monitoring. Once deployed, it installs Prometheus, Alertmanager[28], and Grafana inside the cluster. Prometheus is responsible for scraping the metrics of the various Kubernetes components, such as the API server, the scheduler and the controller, as well as any additional services that are deployed inside the cluster and expose Prometheus metrics. The metrics are by default persisted inside Prometheus, with the option of using an external database as well. Alertmanager comes preconfigured with some default alerts that are sent when cluster resources are low, or when critical Kubernetes components are unavailable. Grafana comes preloaded with few default dashboards, where various graphs are available that showcase the current cluster state. The cluster administrator can get detailed information on all possible levels, from individual pods, to cluster nodes, to the cluster itself. One benefit of Prometheus operator over an independent deployment is that it installs custom resource definitions[29] (CRDs) inside the cluster, and these can be used for defining YAML files that describe what additional services need to be monitored, the same way that native Kubernetes resources are defined, for example Deployments.

Log management is made possible with the EFK stack, where Fluentd is deployed as a DaemonSet on all of the available Kubernetes nodes. Fluentd tails all of the container logs as well as the host logs, preprocess them, enriches them with Kubernetes specific data, such as the namespace to which the given container belongs, or the node where it runs and then sends these to Elasticsearch for long term storage and indexing. Kibana is a web application that connects to Elasticsearch and can monitor the logs in real-time and even visualize them using different graph types. Since no retention policy can be configured in Elasticsearch, an additional tool needs to be used, in this case Elasticsearch Curator[30] which has the option of deleting indices on predefined conditions, one being time. Each day, a new index is created in Elasticsearch that will contain the logs for that day. Deleting the index removes the logs for that period and frees up the used storage space. Elasticseach Curator works as a Kubernetes Cronjob that can be executed at a desired interval.

V. Conclusion

Kubespray is already a major player in the Kubernetes world and is even one of the officially recommended ways to deploy Kubernetes. The large user base, along with a rich feature set, active development and support for lifecycle management

make it one of the best choices for new deployments. Not all of the Kubernetes extensions mentioned in this paper are natively supported by Kubespray, but the easy customizability of the Ansible playbooks and roles means that end users can extend them with minimal effort.

REFERENCES

[1] M. Lukša, *Kubernetes in Action*. Manning Publications Company, 2018.

[2] "Kubernetes - production-grade container orchestration," https://kubernetes.io/docs/home/, accessed: 2019-04-13.

[3] "Kubespray - deploy a production ready kubernetes cluster," https://kubespray.io/, accessed: 2019-04-13.

[4] L. Hochstein and R. Moser, *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. " O'Reilly Media, Inc.", 2017.

[5] "Ansible documentation," https://docs.ansible.com/, accessed: 2019-04-13.

[6] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[7] "Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries," https://docs.docker.com/, accessed: 2019-04-13.

[8] "Kubernetes ingress - kubernetes api object that manages external access to the services in a cluster," https://kubernetes.io/docs/concepts/services-networking/ingress/, accessed: 2019-04-13.

[9] "Nginx ingress controller for kubernetes," https://github.com/kubernetes/ingress-nginx, accessed: 2019-04-13.

[10] "Traefik - kubernetes ingress controller," https://docs.traefik.io/user-guide/kubernetes/, accessed: 2019-04-13.

[11] "Kubernetes load balancer - kubernetes api object that integrates with external load balancers," https://kubernetes.io/docs/concepts/services-networking/#loadbalancer, accessed: 2019-04-13.

[12] "Load balancer implementation for bare metal kubernetes clusters, using standard routing protocols," https://metallb.universe.tf/, accessed: 2019-04-13.

[13] "Networkpolicy - kubernetes resource which specifies how groups of pods are allowed to communicate with each other and other network endpoints," https://kubernetes.io/docs/concepts/services-networking/network-policies/, accessed: 2019-04-13.

[14] "Helm - the package maanger for kubernetes," https://helm.sh/, accessed: 2019-04-13.

[15] "Kubernetes resource which acts as an abstraction to other storage systems," https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims, accessed: 2019-04-13.

[16] "Kubernetes nfs dynamic provisioner," https://github.com/kubernetes-incubator/external-storage/tree/master/nfs, accessed: 2019-04-13.

[17] N. Fisk, *Mastering Ceph*. Packt Publishing Ltd, 2017.

[18] "Ceph uniquely delivers object, block, and file storage in one unified system," http://docs.ceph.com/docs/master/, accessed: 2019-04-13.

[19] "Rook - storage orchestration for kubernetes," https://rook.io/, accessed: 2019-04-13.

[20] "Flannel - a network fabric for containers, designed for kubernetes," https://github.com/coreos/flannel, accessed: 2019-04-13.

[21] "Calico is an open source networking and network security solution for containers, virtual machines, and native host-based workloads," https://docs.projectcalico.org/v3.6/introduction/, accessed: 2019-04-13.

[22] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[23] "Grafana is an open platform for beautiful analytics and monitoring," https://grafana.com/, accessed: 2019-04-13.

[24] "Elasticsearch, logstash, kibana, open-source logging stack," https://www.elastic.co/elk-stack, accessed: 2019-04-13.

[25] "Fluentd - open source data collector," https://www.fluentd.org/, accessed: 2019-04-13.

[26] "Add rbd provisioner addon support to kubespray," https://github.com/kubernetes-sigs/kubespray/pull/3668#issuecomment-480623695, accessed: 2019-04-13.

[27] "The prometheus operator creates, configures, and manages prometheus monitoring instances." https://coreos.com/operators/prometheus/docs/latest/, accessed: 2019-04-13.

[28] "Alertmanager handles alerts sent by client applications such as the prometheus server," https://prometheus.io/docs/alerting/alertmanager/, accessed: 2019-04-13.

[29] "Custrom resource definitions - kubernetes documentation," https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/#customresourcedefinitions, accessed: 2019-04-13.

[30] "Elasticsearch curator eases the management of elasticsearch indices," https://www.elastic.co/guide/en/elasticsearch/client/curator/current/about.html, accessed: 2019-04-13.