SRPCE

NAME:- Akash R

ROLL.NO:- 2

REG.NO:- 422021104002

SEMESTER:-5

DEPARTMENT:-CSE

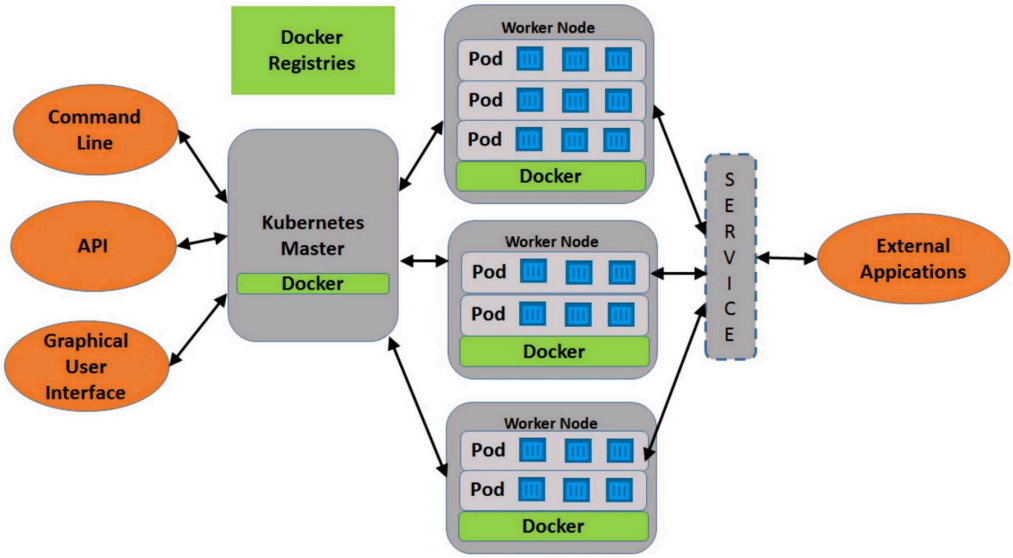
SUBJECT:- cloud Application development

SESSION:- 2021-2025

Introduction

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes deploys containers as “pods” (one or more related containers) across one or more “worker nodes” (such as Linux instances running Docker) in a Kubernetes cluster. Pods can be assigned to logical “services” to allow access to the applications running in the pods from outside of the Kubernetes environment. A “master node” (which can be a single instance or multiple instances clustered for high availability) is the control point for a Kubernetes environment.

The following is a high level view of the general Kubernetes architecture:



The Master node provides the management/user interface for orchestrating the environment. This is where deployments are created, which use Docker registries (public or private) to retrieve the required images. The worker nodes run the application containers in pods, which is a logical grouping of related containers. The pods can be scaled across nodes as needed or desired. Services are logical definitions that provide access to the applications within the pods from external applications. Replicating pods across nodes and front-ending them with service definitions provides high availability deployment of PAAS and SAAS applications in a Kubernetes environment.

Command line and REST API interfaces are provided, to allow Kubernetes to be integrated with automated processes. A user interface (the Kubernetes dashboard) is available for users, with authorization and grouping (via “namespaces”) for access to resources than can be deployed.

More information on Kubernetes can be found at https://kubernetes.io/, where documentation and code can be obtained.

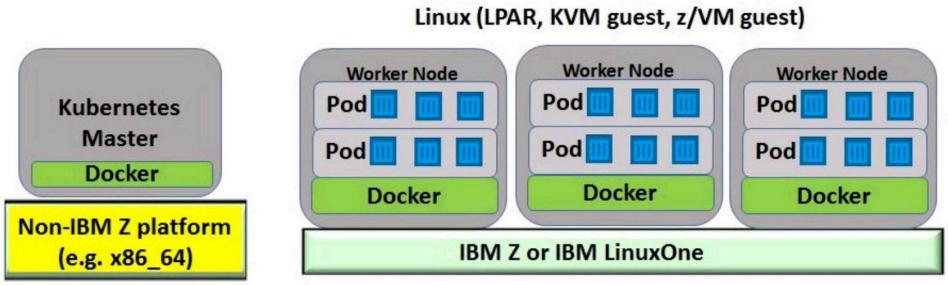
Kubernetes Linux on Z configurations

(**note:** when this paper uses “**Linux on Z**” it refers to Linux running on either the **IBM Z** or **IBM LinuxONE** mainframe platforms)

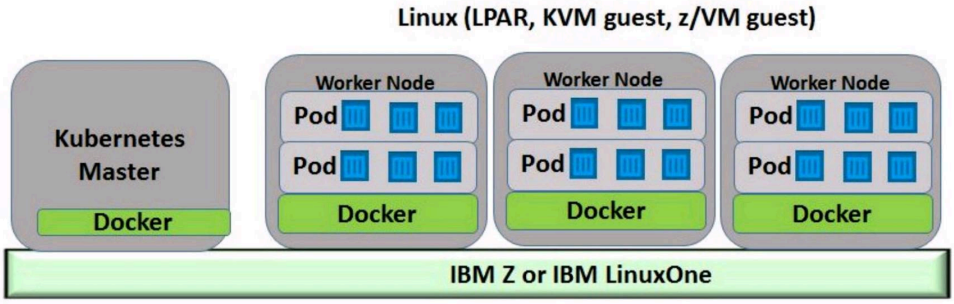
From a Linux on Z perspective Kubernetes can be implemented in two basic configurations:

 ©2018 IBM Corporation

1) Manage to: The Kubernetes master nodes runs on either x86-64 or Unix platforms, with the worker nodes on Linux on Z:



2) Manage-from: The entire Kubernetes infrastructure runs on Linux on Z:



For either configuration, for the Linux on Z nodes:

- The Linux hosting the Kubernetes function can be running in a dedicated LPAR, as a guest under z/VM, or as a guest under KVM.
- The nodes do not have to reside on the same physical platform; they can be spread across multiple physical platforms as long as there is IP connectivity between them.

The Kubernetes documentation for the installation and implementation process is at https://kubernetes.io/docs/setup/; however, there is little information specific to implementation when using Linux on Z; this paper will highlight some of the issues, considerations, and workarounds the author found when attempting to implement both of the above architectures.

What is described in this white paper is not the only method, or perhaps the best method, for implementing Kubernetes in a Linux on Z platform in a private environment. However it may be useful for a "fast start" to get it up and running.

Selecting the installation method

There are many documented ways to install Kubernetes in a private environment, from building its components from scratch to using commercial hosted solutions. The options are best described at https://kubernetes.io/docs/setup/pick-right-solution/

For these examples the **kubeadm** installation method was used. The kubeadm toolkit is used to quickly get a Kubernetes cluster up and running. It uses many defaults for the configuration but does provide some flexibility for customizing the installation. Although kubeadm's overall feature state as of this white paper is **Beta,** it will soon be graduated to **General Availability (GA)** during 2018.

 ©2018 IBM Corporation

The most challenging part of using kubeadm is installing the networking component; kubeadm does not do this, and while the command to do this is simple, the issue if finding a networking solution that is supported via Linux on Z. This will be covered in more detail later in the paper.

The kubeadm installation process is documented in detail at:

- https://kubernetes.io/docs/setup/independent/install-kubeadm/ (installing the kubeadm toolkit on the nodes, including node prerequisite information)
- https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/ (using the installed kubeadm toolkit to create the Kubernetes cluster)

This paper is not intended to repeat the documented installation steps, but to highlight where the installation steps have unique considerations for the Linux on Z platform.

kubeadm toolkit platform requirements and installation

For Linux on Z the supported Linux distributions for kubeadm are Red Hat and Ubuntu.

With kubeadm, Kubernetes components are downloaded from the internet as Docker images and run as Docker containers. So the basic prerequisite is (a) connectivity to the internet, and (b) A support version of Docker pre-installed on all master and worker nodes.

The three major components of the kubeadm toolkit are

- **kubeadm**: the command used to initialize the master node and to join worker nodes to the master node.
- **kubelet**: the Kubernetes agent that runs on all nodes and provides the control interface to the node.
- **kubectl:** the Kubernetes command line interface

These are installed from the distribution repositories. For a Linux on Z node a compatible (s390x) repository is required. For Ubuntu, using the documented instructions will automatically use the right repository. For Red Hat, replace the string **x86_64** with **s390x** in the URL named in the documented instructions to access the correct repository.

Kubernetes does not like swap space defined on the nodes. While flags can be added when installing to ignore the presence of swap, it was easier to simply define Linux instances without swap space to use for the nodes.

An issue was encountered that was only discovered after the networking was installed, related to the cgroups definition in Docker and kubelet (which must match). By default the **cgroups** value for both was set to **systemd**; to resolve the issue the cgroups setting for both was changed to **cgroupfs**.

Installing the master node

The **kubeadm init** command run on the master node will install the Kubernetes master node functions. It will also perform a prerequisite check for missing Linux packages - some will be WARNING, which can be ignored. However, the ERROR messages must be resolved before kubeadm init will run successfully.

There are many options available for the kubeadm init command, which are documented at https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-init/. One useful option is **--dry-run**. This displays the actions kubeadm will perform to set up the master node, but will not actually perform them.

 ©2018 IBM Corporation

If kubeadm init is successfully run, it will display the command **kubeadm join** and the command parameters to execute on worker nodes to connect them to the master node. HOWEVER, do not run these yet. The network component must be installed and customized first.

Setting up the Kubernetes network

Kubernetes requires a network plug-in, as each deployed pod (a group of one or more Docker containers) instance requires an IP address, and each service (which clusters pods and optionally exposes them for external access) requires an IP address. Along with that the various networking configurations for routing is also performed by the network plug-in.

The supported network plug-ins are shown in the kubeadm installation documentation. The tricky aspect for these examples was finding s390x Docker images for the supported network plug-ins. Otherwise, the documented network plug-in installation process will deploy x86_64 architecture images onto the Linux on Z nodes, which simply will not run containers from non-s390x architectures.

To enable networking in these environments, these steps had to be performed:

1) Download the .yaml file associated with the network plug-in, to determine the Docker images used by the plug-in.

   The network plug-in is installed using the command **kubectl apply -f <name of yaml file>.** The kubeadm documentation identifies the Kubernetes yaml files used for deploying each network plug-in option. By looking at the .yaml file for a plug-in, one can see the Docker images that will be used for that plug-in.

2) Locate, or (if necessary) build a corresponding s390x Docker image for each of the images identified in the file.

   More than likely the images identified are going to be x86_64 images. So for the desired networking option, the corresponding 390x images will have to be located, or built.

   In these examples, the **kube-router** network plug-in was chosen. It may not be the best option, but it required the least number of Docker images:

   - **busybox** - this is a common container and an s390x version is available on Docker Hub. There was no need to rebuild this Docker image.
   - **cloudnativelabs/kube-router**: A s390x version of this container was not found, so a s390x version was built from the source (https://github.com/cloudnativelabs/kube-router)

3) Install the s390x Docker images on the Linux on Z nodes, and tag them to match the image name in the .yaml file

   The s390x network plug-in Docker images are manually loaded onto the Linux on Z nodes:

   - For the manage-to environment (master node not on Linux on Z), this step is performed on the Linux on Z worker nodes.
   - For the manage-from environment (master node on Linux on Z), this step is performed on the Linux on Z master node and Linux on Z worker nodes

 ©2018 IBM Corporation

In addition, the default image name:tag information may not match what is in the .yaml file. So after loading the images on the Linux on Z nodes, they were tagged with the names in the .yaml file. For example, by default the s390x busybox Docker image is named s390x/busybox:latest. The name:tag value of busybox:latest was added to this image. Why this is done will be explained in step 5 below.

4) Install the gcr.io/google_containers/kube-proxy-s390x Docker image, and add a name:tag of gcr.io/google_containers/kube-proxy-amd64:latest to it.

   During the kubeadm installation process it was observed that an x86_64 version of the gcr.io/google_containers/kube-proxy Docker image was being installed on the Linux on Z nodes. As a workaround the gcr.io/google_containers/kube-proxy-s390x was manually installed and tagged with the x86_64 name, so that the installation process would see it as already installed.

5) Update the **imagePullPolicy** parameter in the .yaml file to **IfNotPresent**

   The imagePullPolicy parameter specifies when Kubernetes will retrieve an image from the designated repository defined in the .yaml file. It is changed to ifNotPresent to avoid overlaying the s390x images with x86_64 images. When the .yaml file is executed with **imagePullPolicy: ifNotPresent** defined, Kubernetes will check for Docker image names on the node that match the names in the .yaml file. Because it will see the images that have been tagged with the .yaml file image name, it will use them and not attempt to pull images from the repository.

6) Execute the network plug-in .yaml file to install and customize the network component

   Now the kubectl apply -f <network plug-in .yaml file> can be executed on the master node - BE SURE TO USE THE FILE THAT WAS DOWNLOADED AND UPDATED, NOT THE FILE AT THE DOCUMENTED URL.

   - If the master node is on x86_64, it will pull down the required images.
   - If the master node is on Linux on Z, it will use the images that were already loaded into Docker on the master node.

Adding the worker nodes

After the network plug-in is installed as described in the previous setup, worker nodes can now be added by executing the kubeadm join command that was output from the kubeadm init execution. Since the network plug-in modules have already been installed, kubeadm will use what is there to create the required Docker containers to support the networking functions. In this environment, in both the manage-from and manage-to configurations, the nodes were fully online and useable in less than 5 minutes.

Optional step - Kubernetes Dashboard

Kubernetes provides an optional graphical user interface called the Kubernetes dashboard. Documentation for it, including installation, is found at https://github.com/kubernetes/dashboard. That link also contains a pointer to the .yaml file used to deploy it.

Because it uses the Kubernetes API server, which runs on the master node, the dashboard Docker image must also be installed on the master node. The yaml file must be downloaded and updated as follows:

- If the master node is Linux on Z, change the **image: value** to **k8s.gcr.io/kubernetes-dashboard-s390x:v1.8.2** (as of this white paper there was not yet a version 1.9.x Docker image for the dashboard; that may change in the future).

- Add the following lines to the .yaml file under the serviceAccountName line - these are required so that it installs on the master node:

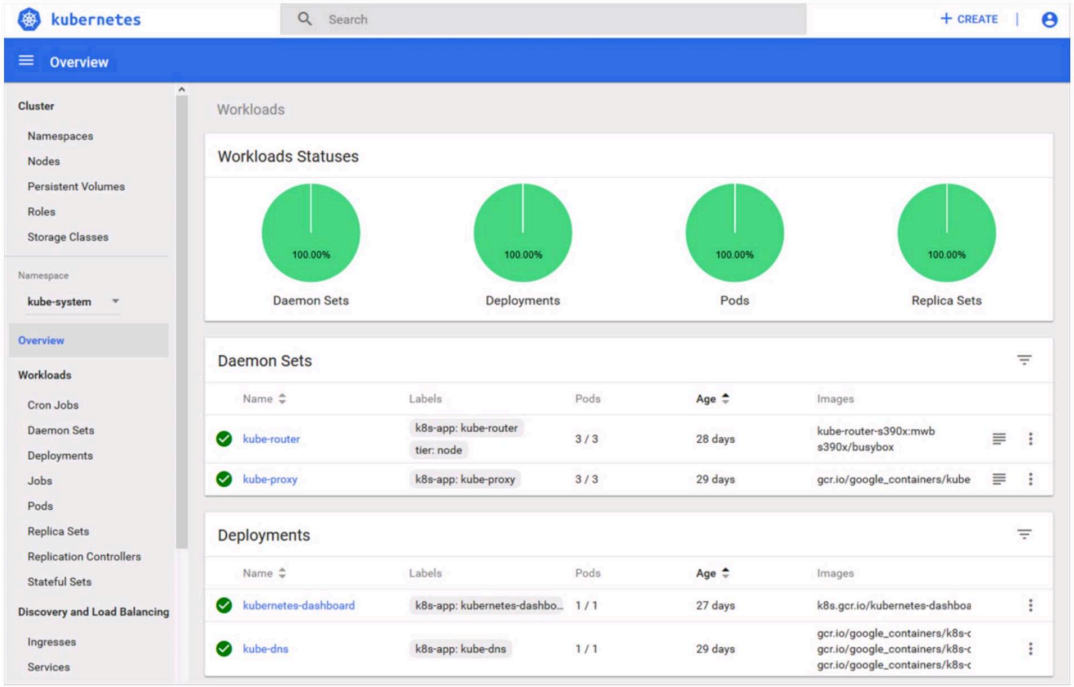
```
tolerations:
- key: node-role.kubernetes.io/master
  effect: NoSchedule
nodeSelector:
  node-role.kubernetes.io/master: ""
```

- Change the spec: type: value to NodePort so that the dashboard can be accessed outside of the cluster

Once the .yaml file is updated, the dashboard can be deployed with the command

kubectl apply -f <name of updated .yaml file>

After authentication for the dashboard is defined (beyond the scope of this document, see https://github.com/kubernetes/dashboard/wiki/Access-control for details), it is available for use:



Summary and Acknowledgements

There are a number of different ways for installing a Kubernetes cluster. The lack of specific documentation when a Linux on Z environment is involved (for either manage-from or manage-to) can be a challenge in properly setting up and configuring Kubernetes. This paper addresses the issues for using the kubeadm method for quickly getting a Kubernetes environment for Linux on Z - either manage from or manage to - up and running; this information can be evaluated for applicability to other installation methods.

Special thanks to the following people who reviewed this document and provided input:

- Mike Sine, IBM Washington Systems Center

 ©2018 IBM Corporation