Abstract

To develop and build a modern cloud applications or DevOps implementation, both Docker and Kubernetes have revolutionized the era of software development and operations. Though both these technologies are unique and serve different purposes, they unify the process of software development and integration. This project focuses on building cloud infrastructure to spin up a kubernetes cluster and deploy a containerized react JS application that supports scaling and rolling updates. Docker is used to build and ship applications as containers. Containers use less space, are reliable and are very fast Kubernetes is an automated container orchestration tool that helps in managing containers, deployment and scaling platform. Using Azure Cloud Platform to deploy containers on Kubernetes Engine enables rapid application development and management. Terraform is used to create all the necessary resources onto Azure, Terraform helps maintain infrastructure as code. All this process is automated using a shell script. Finally the web application is exposed through the external ip of the load balancer.

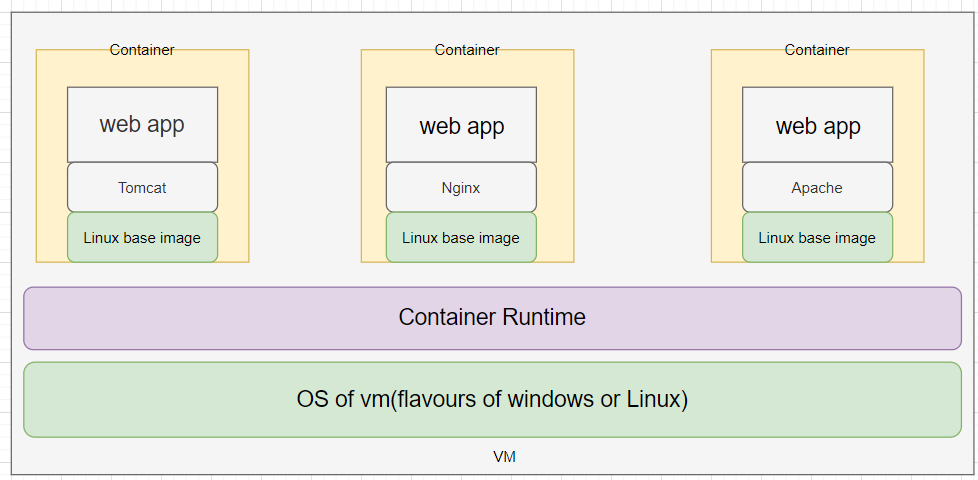
Introduction

Docker and kubernetes have taken the era of cloud computing by storm. This paper discuses about how kubernetes and docker is used to deploy applications in a very short time and how applications on the cloud have 0 downtime, how application updates are performed and how the CI/CD pipeline can be automated using tools like Jenkins and Jenkins x. Traditionally in order to deploy any application on the cloud vms were used. But the drawback was that it was not possible to deploy multiple applications with different dependencies on the same vm. And another major problem was that applications were not working the same in all environments. It was also difficult to maintain configurations in case multiple micro services were used. There may be several micro services that build one complete application in such cases these micro services or containers must work in harmony and in a well-organized manner where one container can talk to another. Kubernetes comes into play to achieve such an environment, kubernetes is a container orchestration platform and is supported by almost all cloud providers. This paper focuses about AKS which is Azure Kubernetes Service nothing but Kubernetes running in Azure. Once application is deployed in AKS and exposed it can be accessed across the globe and the application is always up and running that is no or zero downtime all this may seam like a lengthy process but once all this is automated in the CI/CD pipeline all this can be done with a single command.

Docker and container orchestration

1. Docker:

Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines.



1. Containers:

Application and all its dependencies put together are known as a container. Multiple containers can be mad to run on a single VM as long as they belong to the same kernel either Windows/Linux. Containers can be built as a windows container or a Linux container. And applications will behave the exact same way in all environments. It is also very easy to maintain micro services along with their configurations.

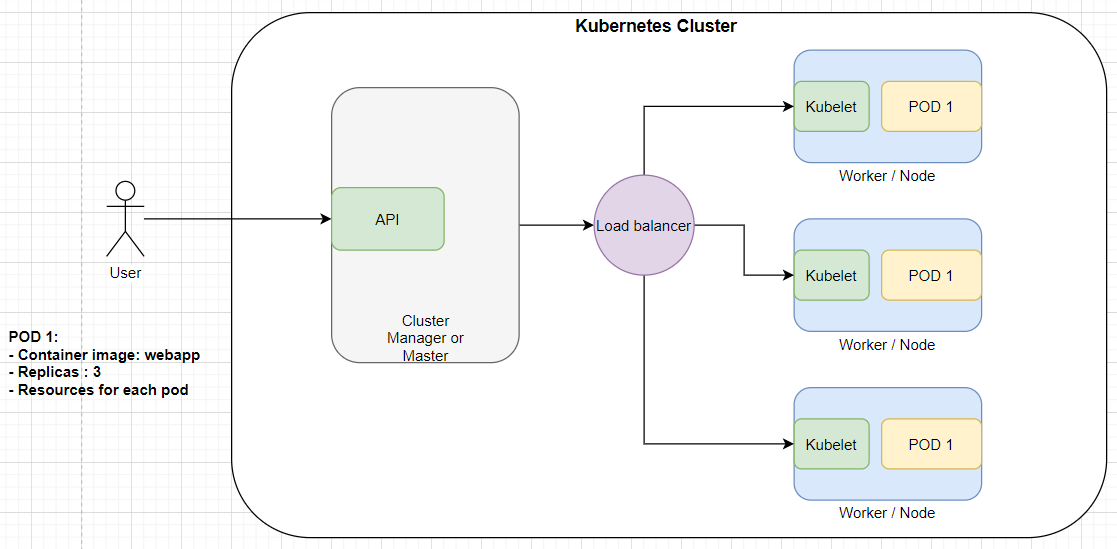
Container Orchestration

1. Introduction to kubernetes:

Kubernetes is a system for running and coordinating containerized application across a cluster of machines. K8S can be used to manage the lifecycle of containerized applications, scalability and also ensures high availability. We can define how our application should interact with other applications and with the outside world. We can scale applications up or down, perform rolling updates and switch traffic between different versions of the application to test features. Kubernetes is supported in almost all cloud providers for example AKS (Azure Kubernetes Service) in AWS is called EKS and on Google cloud Platform it is known as GKS.

1. Kubernetes Architecture:

Kubernetes puts together individual physical or virtual machines into a cluster using a shared network to communicate with each other. All kubernets components and workloads are configured in the cluster.



There are two major components of a kubernetes cluster, the master and the nodes. The **master** is like the brain of the k8s cluster it exposes API for the clients, checking the health of the nodes, splitting up assigned tasks and managing communication between components. It takes care of handling deployments and managing worker nodes. Azure provides cluster manager for free of cost.

**Nodes** are machines in which the work loads are run i.e. nodes are vms in which actual deployment happens. Applications are run as containers, so each node must have a container runtime. Kublet is a service that is running in the nodes that is responsible for taking up tasks from the master and creates or destroys containers accordingly. K8s is responsible for maintaining desired state configuration.

To deploy an application YAML files are used. These files contain information like what to create and how it should be managed in other words this file is the desired state configuration. The master node takes up this plan and decides how to run it in the existing infrastructure.

1. Master Server Components:
2. Etcd: etcd is used to store configuration data that can be accessed by all nodes of the cluster. It helps in maintaining cluster state. It helps services to configure according to up-to-date information.
3. Kube api server: It helps to assign workloads to nodes analyzing the current infrastructure and places work on acceptable node.
4. Cloud controller manger: K8s can be deployed in many different environments and can interact with various infrastructure providers to manage state of resources. Cloud controller manager acts as a bridge that allows K8s to interact with various providers with different features and APIs, this allows K8s to update its state information according to information gathered from the cloud provider.
5. Node server components:
6. Container runtime: Container Runtime is a mandatory component in every node that is made available when docker is installed. It helps managing the containers.
7. Kubelet: Kubelet is a service that runs in all nodes that acts as a contact point for the master. Kubelet receives commands from the master and begins the work, it also interacts with etcd to maintain states. Kubelet process takes responsibility for maintaining state of work in the node. It controls the container runtime to launch and destroy containers as needed.
8. Kube-proxy: Kube-proxy navigates requests to the correct containers in the node it can perform a basic level of load balancing. It makes sure that services are available to other components.
9. Kubernetes Objects and workloads :

Kubernetes uses additional layers of abstraction over containers to provide features like scaling, life cycle management.

1. Pods: A pod is the most basic unit of kubernetes. A pod is generally one or more tightly coupled containers that are to be deployed in the same node. Pods are fundamental units of deployment. Container instances are not deployed directly, when pods are deployed container instances are a part of that pod.
2. Replication set and replication controller: These are used to horizontally scale applications. Replication defines a pod template and parameters to deploy identical pods. It can be used to either increase or decrease the number of running nodes. Controller is responsible for maintaining the specified number of pods or instances running. If one instance fails controller automatically start new pod to meet the desired state configuration.
3. Service: A service groups logical collections of pods to represent them as a single entity. This allows us to deploy a service that can monitor all containers of a particular type. IP address remains the same regardless of changes pods it route to. In order to give access to a pod to other applications or the end user a service has to be configured. the **Load Balancer** service type creates an external load balancer to route to the service using a cloud provider’s Kubernetes load balancer integration. The cloud controller manager will create the appropriate resource and configure it using the internal service service addresses.
4. Deployments: Deployment is the easiest and most used resource for deploying your application. It is a Kubernetes controller that matches the current state of your cluster to the desired state mentioned in the Deployment manifest. e.g. If you create a deployment with 1 replica, it will check that the desired state of ReplicaSet is 1 and current state is 0, so it will create a ReplicaSet, which will further create the pod. If you create a deployment with name **counter**, it will create a ReplicaSet with name **counter-<replica-set-id>**, which will further create a Pod with name **counter-<replica-set->-<pod-id>.** Deployments are usually used for stateless applications. However, you can save the state of deployment by attaching a Persistent Volume to it and make it stateful, but all the pods of a deployment will be sharing the same Volume and data across all of them will be same.

Azure kubernetes services

Azure kubernetes service paves a simple way to deploy and manage a kubernetes cluster. AKS abstracts all the complexities and operational overheads of maintaining a kubernets cluster as azure takes care of most of the respossibilities. As a hosted Kubernetes service, Azure handles critical tasks like health monitoring and maintenance for you. The Kubernetes masters are managed by Azure. You only manage and maintain the agent nodes. As a managed Kubernetes service, AKS is free - you only pay for the agent nodes within your clusters, not for the masters. You can create an AKS cluster in the Azure portal, with the Azure CLI, or template driven deployment options such as Resource Manager templates and Terraform. When you deploy an AKS cluster, the Kubernetes master and all nodes are deployed and configured for you. Additional features such as advanced networking, Azure Active Directory integration, and monitoring can also be configured during the deployment process. Windows Server containers support is currently in preview in AKS.

Features of AKS

1. identity and security management :

To limit access to cluster resources, AKS supports Kubernetes role-based access control (RBAC). RBAC lets you control access to Kubernetes resources and namespaces, and permissions to those resources. You can also configure an AKS cluster to integrate with Azure Active Directory (AD). With Azure AD integration, Kubernetes access can be configured based on existing identity and group membership. Your existing Azure AD users and groups can be provided access to AKS resources and with an integrated sign-on experience.

1. Integrated logging and monitoring:

To understand how your AKS cluster and deployed applications are performing, Azure Monitor for container health collects memory and processor metrics from containers, nodes, and controllers. Container logs are available, and you can also review the Kubernetes master logs. This monitoring data is stored in an Azure Log Analytics workspace, and is available through the Azure portal, Azure CLI, or a REST endpoint.

1. Auto scaling

To keep up with application demands in Azure Kubernetes Service (AKS), you may need to adjust the number of nodes that run your workloads. The cluster autoscaler component can watch for pods in your cluster that can't be scheduled because of resource constraints. When issues are detected, the number of nodes in a node pool is increased to meet the application demand. Nodes are also regularly checked for a lack of running pods, with the number of nodes then decreased as needed. This ability to automatically scale up or down the number of nodes in your AKS cluster lets you run an efficient, cost-effective cluster.

AKS cluster can be scaled in two ways

* The **cluster autoscaler** watches for pods that can't be scheduled on nodes because of resource constraints. The cluster then automatically increases the number of nodes.
* The **horizontal pod autoscaler** uses the Metrics Server in a Kubernetes cluster to monitor the resource demand of pods. If an application needs more resources, the number of pods is automatically increased to meet the demand.

Both the horizontal pod autoscaler and cluster autoscaler can also decrease the number of pods and nodes as needed. The cluster autoscaler decreases the number of nodes when there has been unused capacity for a period of time. Pods on a node to be removed by the cluster autoscaler are safely scheduled elsewhere in the cluster. The cluster autoscaler may be unable to scale down if pods can't move, such as in the following situations:

* A pod is directly created and isn't backed by a controller object, such as a deployment or replica set.
* A pod disruption budget (PDB) is too restrictive and doesn't allow the number of pods to be fall below a certain threshold.
* A pod uses node selectors or anti-affinity that can't be honored if scheduled on a different node.

The cluster autoscaler uses startup parameters for things like time intervals between scale events and resource thresholds. These parameters are defined by the Azure platform, and aren't currently exposed for adjustment.

The cluster and horizontal pod autoscalers can work together, and are often both deployed in a cluster. When combined, the horizontal pod autoscaler is focused on running the number of pods required to meet application demand. The cluster autoscaler is focused on running the number of nodes required to support the scheduled pods.

Spinning up an AKS cluster

Azure service principle:

To access any application registered in the Azure active directory a service principle must be created. The service defines security rules and access restriction for users accessing the application. Service principle takes care of authentication via login and authorization when accessing he resources. In order to use terraform to deploy resources into azure from our local environment a service principle must be created though which we access and deploy the resources. These resources are secured by the Azure AD tenant.

Terraform:

Terraform can be run from your local machine or from azure cloud shell. In either ways it works the same. When creating any resource on the cloud using terraform, terraform takes you one step further ahead. Before actually deploying the resources into Azure, terraform plan gives you an over view of what changes will we made if the script is applied. After going through the plan and there is no errors or conflicts we can then apply the changes. Terraform also helps create multiple resources using a single terraform file. We don’t have to worry about what order the resources should be specified in order to create the resources sequentially. Terraform will automatically detect the dependencies and start deploying the resources either parallel or in a sequential manner.

Spinning up the cluster using terraform:

In order to spin up an AKS cluster using terraform form a local machine, Initial setup must be made. I used WSL throughout this project to work with azure to make any deployments and to access my cluster. First of all a storage account in which a storage container must be created where the terraform state files will be stored, Kube config file will also be taken from tfstates .Terraform must be installed and added to the bin folder and verify terraform installation using terraform –version then create a empty directory where all the terraform scripts will be contained. Once the directory is created it has to be initialized so that required plugins will be available and once all this is done we are ready to deploy our terraform scripts. Copy all the terraform files to the directory. Use terraform plan to see if there will not be any error in creating the resources, if there are no errors then we can apply the terraform plan. This will take about 10 to 15 minutes for the cluster to be setup.

Accessing the cluster:

To access the cluster Kubectl must be installed and configured. Kubectl is a command line tool for accessing a kubernetes cluster. Get the kubeconfig file from the storage account created and save is in ~/.kube/config file so that kubectl can access it. The kubeconfig file contains the required information to access the cluster using kubectl. To verify our cluster is created successfully and the nodes are running use the command kubectl get nodes. Once all the nodes are running we are ready to deploy our application. Kubectl can be used to create all necessary deployments and get any information about our cluster.

Deploying the application:

A containerized application can be deployed into kubernetes creating a deployment and a service type load balancer. By writing a YAML file we can deploy resources into the kubernetes cluster. In the project I have created a deployment and service which is type load balancer. Deployment is responsible for creating replica sets which in turn maintain the desired state configuration in this case the number of replicas specified is 3. The service logically groups these pods across the nodes in the cluster and connects them together through a load balancer. The external ip of the load balancer is given to the end user to access the application.