Kubernetes Components and their functionalities

1. **Pods** - In a docker environment, the smallest unit you'd deal with is a container. In the Kubernetes world, you'll work with a pod and a pod consists of one or more containers. You cannot deploy a bare container in Kubernetes without it being deployed within a pod.
2. **Replica Sets** - Replica Sets are a level above pods that ensures a certain number of pods are always running. A Replica Set allows you to define the number of pods that need to be always running. If a pod crashes, it will be recreated to get back to the desired state.
3. **DemonSets** – ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created. Some typical uses of a DaemonSet are:
    - running a cluster storage daemon on every node
    - running a logs collection daemon on every node
    - running a node monitoring daemon on every node

    In a simple case, one DaemonSet, covering all nodes, would be used for each type of daemon. A more complex setup might use multiple DaemonSets for a single type of daemon, but with different flags and/or different memory and cpu requests for different hardware types.
4. **Deployments** - Deployments ensure safe rollout of new versions of our pods and without outages. They also make it possible to rollback a deployment if there is some terrible issue with the new version. Deployments are a construct a level above replica sets and manage the replica set objects. So, Deployments manage replica sets and replica sets manage pods and pods manage containers.
5. **Services and Labels** - Kubernetes Services tie our pods together and provide a front-end resource to access. You can think of them like a load balancer that automatically knows which servers it is trying to load balance. Services give us a static resource to access that abstracts the pods behind them. Service sits in front of our pods and distributes requests to pods.
6. **Labels** - are just a key value pair, or tag, that provides metadata on our objects. Using Labels and Tags, Services will be able to identify which pods it should be providing a front-end for.
7. **Selectors** - A label selector can be used to organize Kubernetes resources that have labels. We could use labels to match the frontend service with a backend pod automatically by using a selector. An **equality-based selector** defines a condition for selecting resources that have the specified label value. A **set-based selector** defines a condition for selecting resources that have a label value within the specified set of values.
8. **Kube-API-Server** – acts as front end for the Kubernetes control plane. It exposes the Kubernetes API so that all Kubernetes resources can communicate with each other.
9. **etcd** – is consistent and highly-available key value store used as Kubernetes backing store for all cluster data. It stores all the masters and worker node information.
10. **kube-scheduler** – is responsible for distributing containers across multiple nodes. It watches for newly created pods no assigned node and selects a node for them to run on.
11. **kube-controller-managers** – are responsible for identifying and responding to nodes, containers, or endpoint failures. They make decisions to create new containers in such failures. There are several controllers available in Kubernetes:
    - **Node Controller** – is responsible for identifying and responding to node failures.
    - **Replication Controller** – is responsible for maintaining correct number of pods for every replication controller object in the system.
    - **Endpoints Controller** – populates the Endpoints objects e.g., joins Services and Pods.
    - **Ingress Controller** – is a reverse proxy and load balancer. It adds a layer of obstruction to traffic routing, accepting traffic from outside the Kubernetes and load balances the traffic to Pods running inside the nodes.
    - **Service Account and Token Controller** – is responsible for creating default accounts and API Access for new namespaces.
12. **cloud-controller-manager** – is a Kubernetes control plane component that embeds cloud-specific control logic. It only runs controllers that are specific to cloud provider:
    - **Node Controller** – is responsible for checking the cloud provider to identify whether node has been deleted in the cloud after it stops responding.
    - **Route Controller** – is responsible for setting up routes in the underlying cloud infrastructure
    - **Service Controller** – is responsible for creating, updating, and deleting cloud provider Load Balancer.
    - **Ingress Controller** – is a reverse proxy and load balancer. It adds a layer of obstruction to traffic routing, accepting traffic from outside the Kubernetes and load balances the traffic to Pods running inside the nodes.
13. **Container Runtime** – is the underlying software where we run all the Kubernetes components (Docker, rkt, container-d)
14. **Kubelet** – is the agent that runs on every node in the cluster. It is responsible for making sure that containers are sunning in a Pod on a node.

15. **Kube-Proxy** – is a network proxy that runs on each node in the cluster. It maintains network rules on nodes. These network rules allow network communication to the Pods from network sessions inside or outside of the cluster.
16. **Endpoints** – is a resource to track the IP addresses of the objects or Pods which are dynamically assigned to it and which works as a service selector which matches a pod label by adding the IP addresses to the endpoints and these points can be viewed using software kubectl get endpoints.
17. **Service Publishing** – there are three options for publishing our services for accessing our applications:
    - **ClusterIP** – Whenever a service is created a unique IP address is assigned to the service and that IP address is called the ClusterIP. But the ClusterIP is not accessible from outside of the Kubernetes cluster. This is an internal IP only meaning that other pods can use a Services ClusterIP to communicate between each other.
    - **NodePort** – exposes a service on each node's IP address on a specific port. NodePort does not replace ClusterIP however, all it does is direct traffic to the ClusterIP from outside the cluster. NodePort must be within the 30000-32767 port range.
    - **LoadBalancer** – works only in cloud platform and won't work in on-premises. What it would do is spin up a load balancer in the cloud and point the load balancer to your service. This would allow you to use port 443 for example on your load balancer and direct traffic to one of those 30000 or higher ports.
18. **Namespaces** – is a mechanism for isolating groups of resources or projects in a single cluster.
19. Context – is used to group access parameters under an easily recognizable name in a kubeconfig file, the file which is used to configure access to clusters. It is the connection to a particular cluster used by kubectl. This concept only applies in the place where the kubectl command is run.
20. **DNS** – Kubernetes creates DNS records for Services and Pods. You can contact Services with consistent DNS names instead of IP addresses. Kubernetes publishes information about Pods and Services which is used to program DNS. Kubelet configures Pods' DNS so that running containers can lookup Services by name rather than IP.
21. **ConfigMaps** – is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.
22. **Secrets** – A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.
23. **Persistent Volumes** – is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.
24. **PersistentVolumeClaim –** is a request for storage by a user. It is like a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes).
25. **CSI Controller –** The Container Storage Interface is a standard for exposing arbitrary block and file storage system to containerized workloads on Container Orchestration Systems like Kubernetes.
26. **StorageClass** - provides a way for administrators to describe the "classes" of storage they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators. Kubernetes itself is unopinionated about what classes represent. This concept is sometimes called "profiles" in other storage systems.
27. **StatefulSets** – StatefulSet is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of Pods, *and provides guarantees about the ordering and uniqueness* of these Pods. Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.
28. **Role Based Access Control (RBAC)** – Kubernetes RBAC is a key security control to ensure that cluster users and workloads have only the access to resources required to execute their roles. It is important to ensure that, when designing permissions for cluster users, the cluster administrator understands the areas where privilege escalation could occur, to reduce the risk of excessive access leading to security incidents.
29. **Helm Charts** – Helm uses a packaging format called *charts*. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on.
30. **Taints and Tolerations** – Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints.
31. **Network Policies** – are an application-centric construct which allow you to specify how a pod is allowed to communicate with various network "entities" (we use the word "entity" here to avoid overloading the more common terms such as

"endpoints" and "services", which have specific Kubernetes connotations) over the network. NetworkPolicies apply to a connection with a pod on one or both ends and are not relevant to other connections.

32. **Pod Autoscaling** – there are three Kubernetes Autoscaling methods:
    - Horizontal Pod Autoscaling (HPA)
    - Vertical Pod Autoscaling (VPA)
    - Cluster Autoscaling

    And there are two autoscaling layers:
    - Pod-based scaling - supported by Horizontal Autoscaling and the newer Vertical Pod Autoscaling
    - Node-based scaling – supported by the Cluster Autoscaling
33. **Liveness and Readiness Probes** – Liveness and Readiness probes are used to control the health of an application running inside a Pod's container. Both are very similar in functionality, and usage.
34. **Validating Admission Controllers** - is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized. Admission controllers may be *validating*, *mutating*, or both.
35. **Jobs and CronJobs** – Kubernetes Jobs are used to constructing transitory pods that do the duties that have been allocated to them. CronJobs do the same function, except they run tasks on a predefined schedule. Jobs are essential in Kubernetes for conducting batch processes or significant ad-hoc actions.

Creating AWS Elastic Kubernetes Service.

1. Create Terraform Providers

    - Terraform Settings Block
    - Terraform AWS Provider
    - Terraform Kubernetes Provider
    - Terraform Kubectl Provider
    - Terraform Random Provider
    - Terraform Null Provider
    - Terraform Helm Provider
    - Terraform HTTP Provider

2. Create Terraform Backend for Terraform State File

    - Create S3 Bucket
    - Create DynamoDB Table with LockID
    - Create Terraform Backend Block

3. Create VPC using Terraform

    - Create Public and Private Subnets
    - Create Route Tables, Routes, and Route table Associations
    - Create NAT Gateway with Elastic IP. EKS Worker Nodes in private subnet will connect to EKS Cluster API Server Endpoint via NAT Gateway without leaving AWS platform.
    - Create Internet Gateway and attaching the IGW to the VPC

4. Create Bastion Host in Public Subnet using Terraform

    - Create Dynamic Terraform AWS Data latest AMI resource
    - Create Bastion Host EC2 Instance
    - Create Bastion Host Elastic IP
    - Create Bastion Host Security Group
    - Create SSH-key and terraform exec-provisioner and file resources to copy ssh-keys to remote /tmp/ folder
    - SSH to remote node-group and verify kubelet-config.json in /etc/Kubernetes/kubelet/kubelet-config.json and kubeconfig in /var/lib/kubelet/kubeconfig. Get EKS Cluster API server endpoint url and verify the connection using

nslookup or wget <api endpoint url>. Also check connection to ECR using following command **ps -ef | grep kube**, and look for --pod-infra-container-image=…etc.

5. Create Dockerfile and ECR using Terraform

    - Create Amazon Container Registry (ECR)
    - Create appropriate IAM Roles and Policies for EKS Worker Nodes to be able to pull the Docker Image
    - Create Dockerfile and push the Docker Image to ECR

6. Create AWS EKS Cluster using Terraform

    - EKS Cluster – EKS Cluster will be created in AWS managed account and EKS VPC. It is called EKS Control Plane
    - EKS Cluster IAM Roles and Policies
      eks_master_role | eks-AmazonEKSClusterPolicy | eks-AmazonEKSVPCResourceController
    - EKS Cluster Network Interfaces – are created in public subnets automatically when EKS Cluster is provisioned with Cluster endpoint access private ENABLED
    - If Cluster endpoint access private only ENABLED, EKS Cluster Control Plane will create Network Interfaces in our AWS account in public or in private subnet which will allow node-groups to connect to EKS Cluster API server through Network Interfaces
    - EKS Cluster Security Groups – are created in public subnets automatically when EKS Cluster is provisioned and associated to EKS Cluster Network Interfaces

7. Create AWS EKS Node Group using Terraform

    - EKS Node Group
    - EKS Node Group IAM Roles and Policies
      eks_nodegroup_role | eks-AmazonEKSWorkerNodePolicy | eks-AmazonEKS_CNI_Policy | eks-AmazonEC2ContainerRegistryReadOnly
    - EKS Node Group Security Groups – are created automatically when EKS Node Group is provisioned and associated to EKS Node Group Network Interfaces
    - EKS Node Group Network Interfaces – are created automatically when EKS Node Group is provisioned with Cluster endpoint access public and private ENABLED, and it will communicate with EKS Cluster Control Plane API Server Endpoint
    - If Cluster endpoint access private only ENABLED, EKS Cluster Control Plane will create Network Interfaces in our AWS account in public or in private subnet which will allow node-groups to connect to EKS Cluster API server through Network Interfaces

8. Install Kubectl CLI

    - Install Kubectl CLI on local computer
    - Configure Kubeconfig which will update **/Users/ali/.kube/config** file:
      aws eks --region <region> update-kubeconfig --name <cluster_name>
      aws eks --region us-east-1 updatekubeconfig --name my-eks-cluster

36.

9.
10.
11.
12.
13.
14.

15.
16.
17. Kubernetes Deployment and Service using YAML
18. Terraform Kubernetes Provider - Kubernetes Deployment & Service
19. Terraform Remote State Storage - AWS S3 & DynamoDB
20. AWS EKS IAM Roles for Service Accounts (IRSA) using Terraform
21. AWS EKS EBS CSI Driver Install with Self-Managed AddOn Option using Terraform
22. AWS EKS EBS Demo using k8s YAML (UserMgmt WebApp with MySQL DB)
23. AWS EKS EBS Demo using k8s Terraform (UserMgmt WebApp with MySQL DB)
24. AWS EKS EBS Volumes Retain and Resize Settings
25. AWS EBS CSI EKS Add-On
26. Provision AWS IAM Admin User as EKS Admin
27. Provision AWS IAM Basic User as EKS Admin
28. Provision of AWS Users (Admin & Basic) as EKS Admins using Terraform
29. Provision EKS Admins using IAM Roles & IAM Groups
30. Provision EKS Admins using IAM Roles & IAM Groups using Terraform
31. Provision EKS ReadOnly User using IAM Roles, Groups & k8s CR, CRB
32. Provision EKS Developer Users using IAM Roles, Groups & k8s R, RB
33. AWS Load Balancer Controller Install using Terraform Helm Provider
34. Ingress Basics - Automate with Terraform
35. Ingress Context Path based Routing - Automate with Terraform
36. Ingress SSL and SSL Redirect - Automate with Terraform
37. Install ExternalDNS using Terraform Helm Provider
38. Ingress with ExternalDNS - Automate with Terraform
39. Kubernetes LB Service with ExternalDNS - Automate with Terraform
40. Ingress Name based Virtual Host Routing- Automate with Terraform
41. Ingress SSL Discovery Host
42. Ingress SSL Discovery TLS
43. Ingress Groups - Automate with Terraform
44. Ingress Target Type IP - Automate with Terraform
45. Ingress Internal Load Balancer - Automate with Terraform
46. Ingress Cross Namespaces - Automate with Terraform
47. AWS Network Load Balancer with AWS Load Balancer Controller
48. AWS NLB TLS, External DNS with AWS LBC - Automate with Terraform
49. AWS NLB Internal LB with AWS LBC - Automate with Terraform
50. AWS EKS Fargate Profiles using Terraform
51. Run EKS Workloads on AWS Fargate - Automate with Terraform
52. AWS Fargate Only EKS Cluster using Terraform
53. AWS EFS CSI Controller Install using Terraform Helm Provider
54. AWS EFS Static Provisioning - Automate with Terraform
55. AWS EFS Dynamic Provisioning - Automate with Terraform
56. AWS EFS File System Mount for Fargate Workloads
57. Kubernetes Cluster Autoscaler Controller Install
58. Kubernetes Cluster Autoscaler Controller Test
59. Kubernetes Horizontal Pod Autoscaling with Terraform
60. Kubernetes Vertical Pod Autoscaling with Terraform
61. AWS EKS Monitoring and Logging with kubectl

62. AWS EKS Monitoring and Logging with Terraform

Kubernetes Concepts Covered

1. Kubernetes Deployments
2. Kubernetes Pods
3. Kubernetes Service of Type LoadBalancer
4. Kubernetes Service of Type ClusterIP
5. Kubernetes Ingress Service
6. Kubernetes Ingress Class
7. Kubernetes Storage Class
8. Kubernetes Storage Persistent Volume
9. Kubernetes Storage Persistent Volume Claim
10. Kubernetes RBAC
11. Kubernetes Role
12. Kubernetes Role Binding
13. Kubernetes Cluster Role
14. Kubernetes Cluster Role Binding
15. Kubernetes Cluster Autoscaler
16. Kubernetes Vertical Pod Autoscaler
17. Kubernetes Horizontal Pod Autoscaler
18. Kubernetes DaemonSets
19. Kubernetes Namespaces
20. Kubernetes Service Accounts
21. Kubernetes Groups
22. Kubernetes ConfigMaps
23. Kubernetes Requests and Limits
24. Kubernetes Worker Nodes

Terraform Concepts covered

1. Settings Block
2. Providers Block
3. Multiple Providers usage
4. Dependency Lock File Importance
5. Resources Syntax and Behavior
6. Resources Meta-Argument - depends_on
7. Resources Meta-Argument - count
8. Resources Meta-Argument - for_each
9. Resources Meta-Argument - lifecycle
10. Input Variables - Basics
11. Input Variables - Assign When Prompted
12. Input Variables - Assign with terraform.tfvars
13. Input Variables - Assign with auto tfvars
14. Input Variables - Lists
15. Input Variables - Maps
16. File Function
17. Output Values

18. Local Values
19. Datasources
20. Backends - Remote State Storage
21. File Provisioner
22. remote-exec Provisioner
23. local-exec Provisioner
24. Null Resource
25. Modules from Public Registry
26. element function
27. Remote State Datasource
28. Terraform Datasources

Terraform Providers used

1. AWS Terraform Provider
2. Kubernetes Terraform Provider
3. Kubectl Terraform Provider
4. HTTP Terraform Provider
5. Null Terraform Provider
6. Helm Terraform Provider

What will students learn in your course?

- You will build a AWS VPC 3-Tier network with Terraform
- You will build a AWS EKS Cluster with Public and Private Node Groups with Terraform
- You will implement a simple kubernetes deployment and service using Terraform Kubernetes Provider
- Understand and Implement Terraform Remote State Datasource
- You will learn AWS EKS IRSA (IAM Roles for Service Accounts Concept) in detail and implement it with Terraform
- You will master Kubernetes Storage concepts with AWS EBS CSI Controller by automating the whole install process with Terraform
- You will master AWS EKS IAM Concepts with 7 detailed Demo
- You will learn to implement AWS Load Balancer Controller Install on AWS EKS Cluster with Terraform
- You will learn to implement 14 Ingress Service Demos (AWS Application Load Balancer) using Terraform Kubernetes Provider
- You will learn to implement 3 Kubernetes Service Demos for AWS Load Balancer Controller
- You will master the AWS Fargate Concepts with 3 demos including running all workloads of EKS Cluster on AWS Fargate (Fargate Only EKS Cluster)
- You will install Kubernetes Cluster Autoscaler on AWS EKS Cluster with Terraform and Test it
- You will implement Horizontal and Vertical Pod Autoscaler Concepts with Terraform
- You will learn to implement AWS EKS Monitoring and Logging using kubectl and Terraform

Github Repositories used for this course

- Terraform on AWS EKS Kubernetes IaC SRE- 50 Real-World Demos
- Course Presentation
- Kubernetes Fundamentals
- **Important Note:** Please go to these repositories and FORK these repositories and make use of them during the course.