CLOUD CONCEPTS

(DEV-OPS TOOL: Kubernetes [AWS-EKS])

https://kubernetes.io/docs/concepts/

Kubernetes: Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

Traditional Deployment

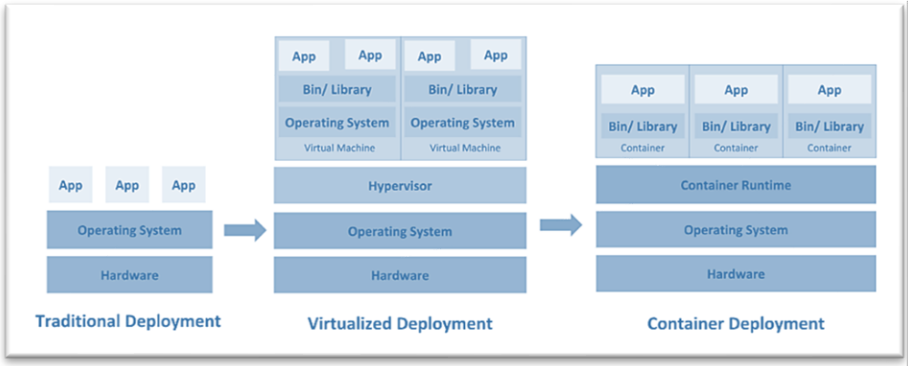
- Organisation ran applications on physical server.
- No way to define resource boundaries for applications causing resource allocation issues.
- Problem: If multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform.
- Solution: Run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

Virtualized Deployment

- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU.
- Allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.
- Allows better utilization of resources in a physical server
- Allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more.
- Virtualization present a set of physical resources as a cluster of disposable virtual machines.
- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container Deployment

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications.
- Containers are considered lightweight.
- Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more.
- They are decoupled from the underlying infrastructure and are portable across clouds and OS distributions.



Containers have become popular because they provide extra benefits, such as:

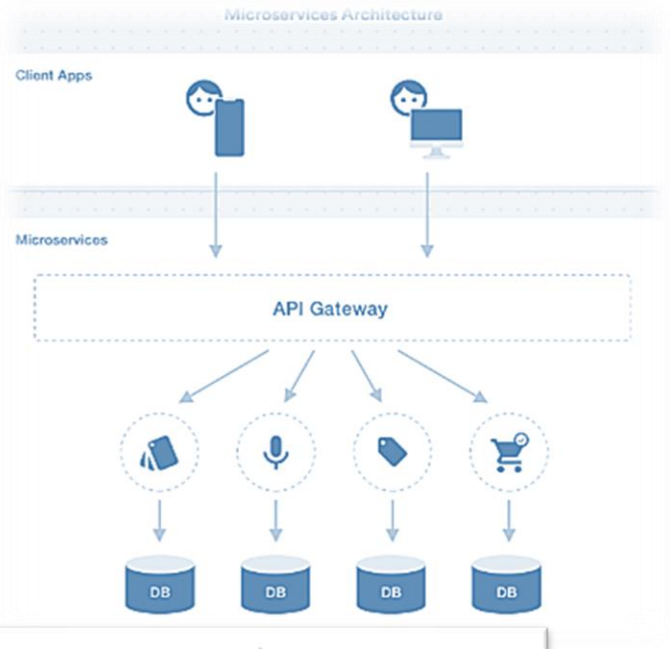
- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability: not only surfaces OS-level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- Resource isolation: predictable application performance.
- Resource utilization: high efficiency and density.

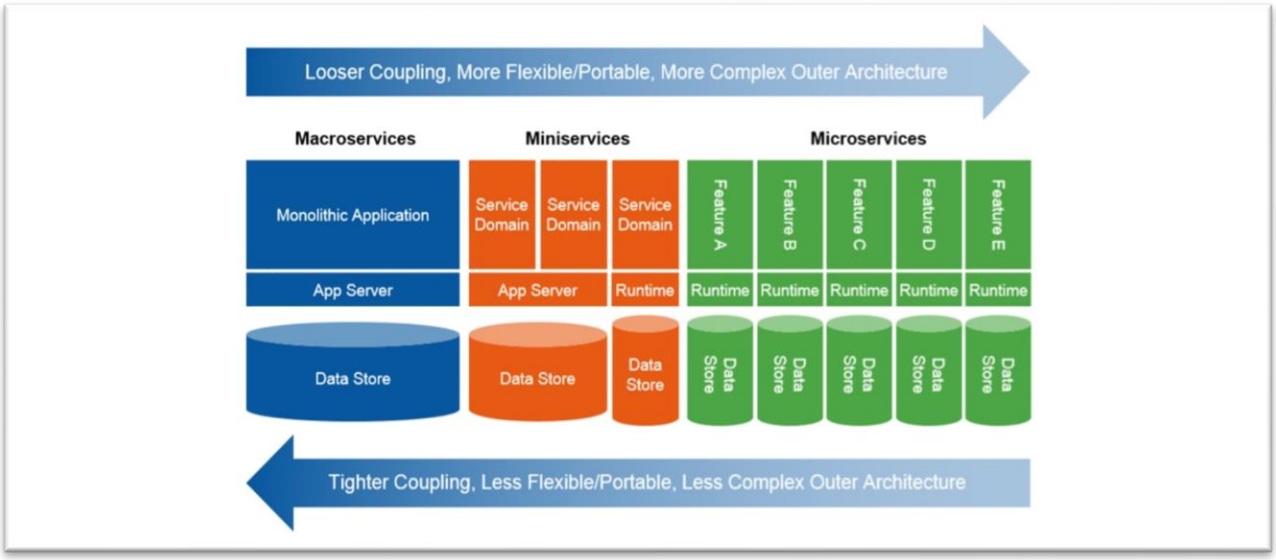
In cloud computing, a **containerized application** refers to an app that has been specially built using cloud-native architecture for running within containers. A container can either host an entire application or small, distributed portions of it known as **microservices**.

- ✓ Loosely coupled.
- ✓ Highly maintainable
- ✓ Organized around business capabilities.
- ✓ Independently deployable
- ✓ Owned by a small team.

E & OE

Handouts: Drakhshan Bokhat





- Problem with Containers:

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Managing this problem through a system is a better solution. And the solution is Kubernetes. Kubernetes can easily manage deployment for your system.

- Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more.

Kubernetes provides you with:

- **Service discovery and load balancing:** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes can load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration:** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks:** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- **Automatic bin packing:** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
- **Self-healing:** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management:** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.
- **Batch execution:** In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.
- **Horizontal scaling:** Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
- **IPv4/IPv6 dual stack:** Allocation of IPv4 and IPv6 addresses to Pods and Services
- **Designed for extensibility:** Add features to your Kubernetes cluster without changing upstream source code.

- **Kubernetes is NOT:**

- × a traditional, all-inclusive PaaS (Platform as a Service) system (operates at the container level rather than at the hardware level).
- × is not monolithic (default solutions are optional and pluggable).
- × does not limit the types of applications supported (support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads).
- × does not deploy source code.
- × does not build your application (Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures).
- × does not provide application-level services, *such as middleware (for example, message buses), data-processing frameworks (for example, Spark), databases (for example, MySQL), caches, nor cluster storage systems (for example, Ceph) as built-in services. Such components can run on Kubernetes, and/or can be accessed by applications running on Kubernetes through portable mechanisms, such as the Open Service Broker.*
- × does not dictate logging, monitoring, or alerting solutions (provides some integrations as proof of concept, and mechanisms to collect and export metrics).
- × does not provide nor mandate a configuration language/system (provides a declarative).

- ✗ does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.

> **Kubernetes is not a mere orchestration system. In fact, it eliminates the need for orchestration. The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C. In contrast, Kubernetes comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state. It shouldn't matter how you get from A to C. Centralized control is also not required. This results in a system that is easier to use and more powerful, robust, resilient, and extensible.**

- Kubernetes objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster.
    - What containerized applications are running (and on which nodes)
    - The resources available to those applications
    - The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance.
  - A Kubernetes object is a "record of intent"--*once the object is created, the Kubernetes system will constantly work to ensure that object exists.*
  - By creating an object, you're effectively telling the Kubernetes system what you want your cluster's workload to look like; this is your cluster's ***desired state***.
  - Kubernetes API is used to work with Kubernetes objects—to create, modify, or delete an object.
  - Every Kubernetes object includes two nested object fields to govern the object's configuration:
    - object spec
    - object status
  - For objects that have a spec: when you create the object you must set its *desired state*.
  - Object status describes the *current state* of the object, supplied, and updated by the Kubernetes system and its components.
  - The Kubernetes control plane continually and actively manages every object's actual state to match the desired state you supplied.

  - ✓ Whenever an object is created in Kubernetes, the object spec *that describes its desired state*, as well as *some basic information about the object (such as a name)* must be provided.
  - ✓ Kubernetes API is used to create objects. An API request must include object spec as JSON in the request body. The information is provided in a file known as a ***manifest***.
    - By convention, manifests are YAML (you could also use JSON format).
    - Tools such as kubectl convert the information *from a manifest into JSON or another supported serialization format* when making the API request over HTTP.

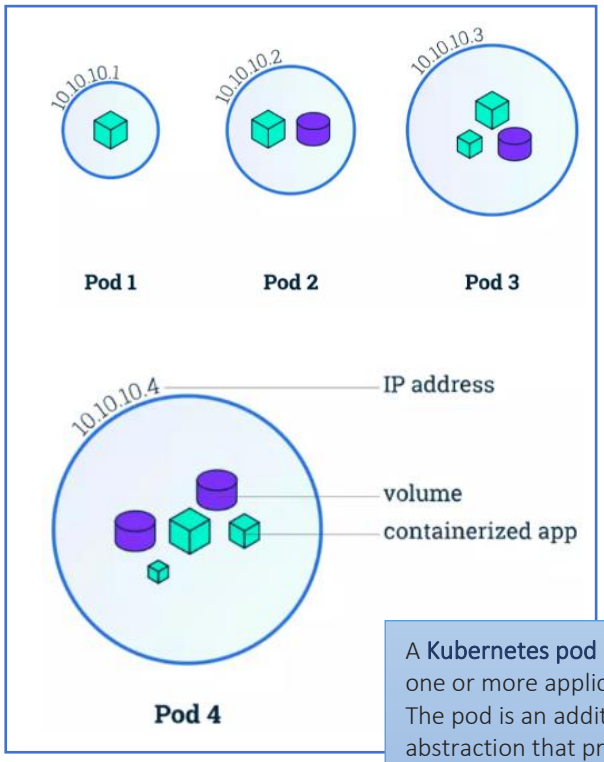
For example: in Kubernetes, a Deployment is an object that can represent an application running on your cluster. When you create the Deployment, you might set the Deployment spec to specify that you want three replicas of the application to be running. The Kubernetes system reads the Deployment spec and starts three instances of your desired application--updating the status to match your spec. If any of those

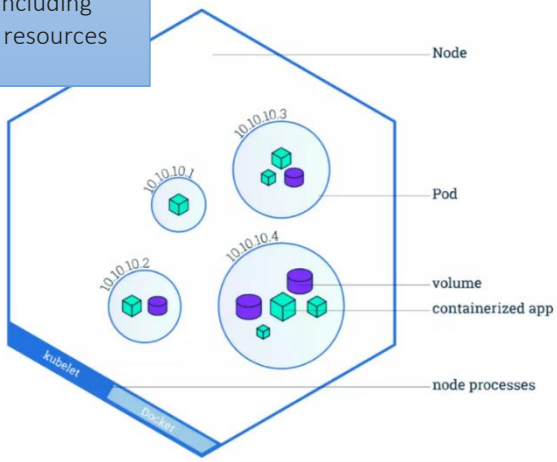
instances should fail (a status change), the Kubernetes system responds to the difference between spec and status by making a correction--in this case, starting a replacement instance.

- ✹ When you deploy a Kubernetes, you get a cluster.

- A Kubernetes cluster consists of a set of worker machines called nodes.
  - Nodes run containerized applications.
  - Every cluster has at least one worker node.
    - The worker node hosts the Pods that are the components of the application workload.
- The control plane manages the worker nodes and the Pods in the cluster.
  - In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

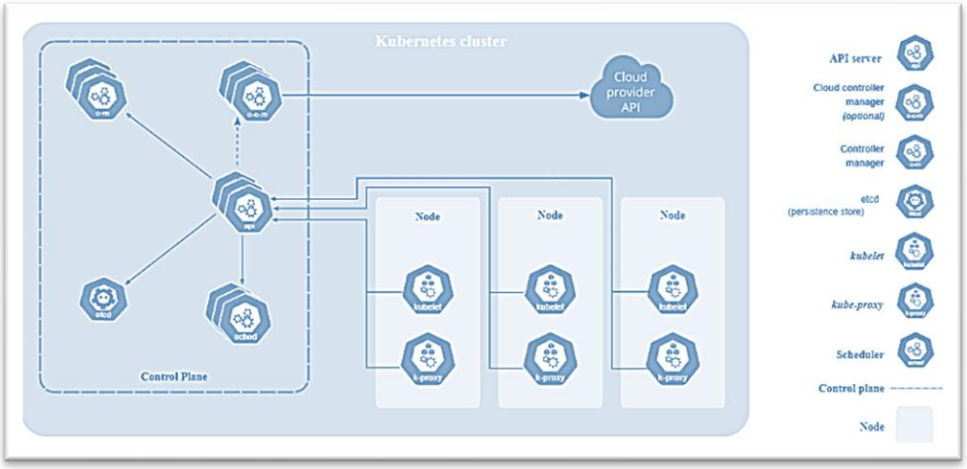
A Kubernetes node is either a virtual or physical machine that one or more Kubernetes pods run on. It is a worker machine that contains the necessary services to run pods, including the CPU and memory resources they need to run.





A Kubernetes pod is a collection of one or more application containers. The pod is an additional level of abstraction that provides shared storage (volumes), IP address, communication between containers, and hosts other information about how to run application containers.

The components of a Kubernetes cluster:



Control Plane Components

- Make global decisions about the cluster (for example, scheduling)
- Detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).
- Can be run on any machine in the cluster. *For simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine.*

Node Components

- Run on every nod
- Maintain running pods
- Provide the Kubernetes runtime environment.

Add-Ons

- Use Kubernetes resources to implement cluster features.
- Namespaced resources for addons belong within the kube-system namespace.

Control Plane Components

- **kube-apiserver** is a component of the Kubernetes control plane that exposes the Kubernetes API i.e. the front end for the Kubernetes control plane. It is designed to scale horizontally—scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.
- **etcd** is a consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
- **kube-scheduler** watches for newly created Pods with no assigned node, and selects a node for them to run on. Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.
- **kube-controller-manager** runs the controller processes. Each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process**.** There are many different types of controllers: like **Node controller** (responsible for noticing and responding when nodes go down), **Job controller** (watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion)

Node Components

- **kubelet** is an agent that runs on each node in the cluster and makes sure that containers are running in a Pod. It also ensures that the containers described in PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
- **kube-proxy** is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. IT maintains network rules on nodes to allow network communication to your Pods from network sessions inside or outside of your cluster.kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.
- **Container runtimem** is a fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

Add-Ons

- **DNS** is strictly required, all Kubernetes clusters should have cluster DNS i.e. a DNS Server which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.
- **Web UI (Dashboard)** is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.
- **Container Resource Monitoring** records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.
- **Cluster-level Logging** is responsible for saving container logs to a central log store with search/browsing interface.
- **Network Plugins** are software components that implement the container network interface (CNI) specification. They are responsible for allocating IP addresses to pods and enabling them to communicate with each other within the cluster.

cloud-controller-manager embeds cloud-specific control logic that link your cluster into your cloud provider's API and separates out the components that interact with that cloud platform from components that only interact with your cluster. It only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager. As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

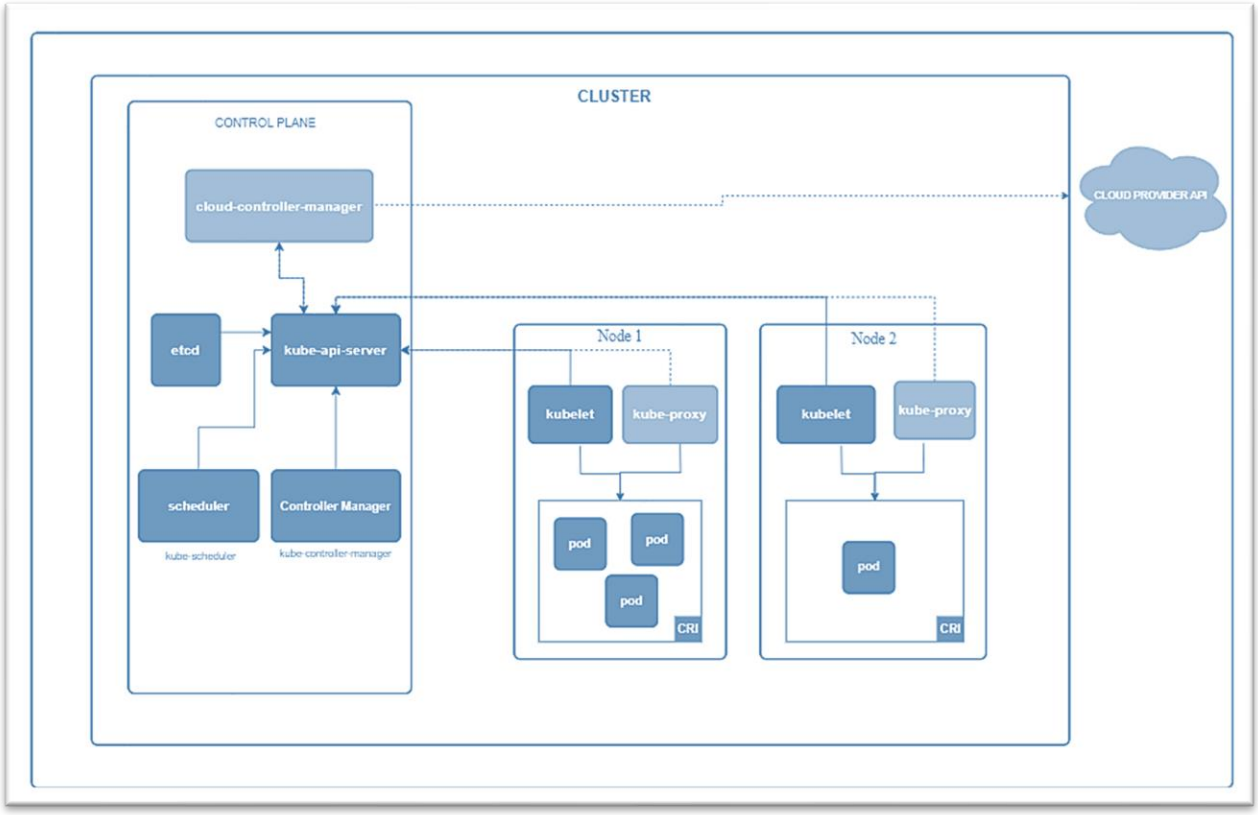
The following controllers can have cloud provider dependencies:

Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.

Route controller: For setting up routes in the underlying cloud infrastructure.

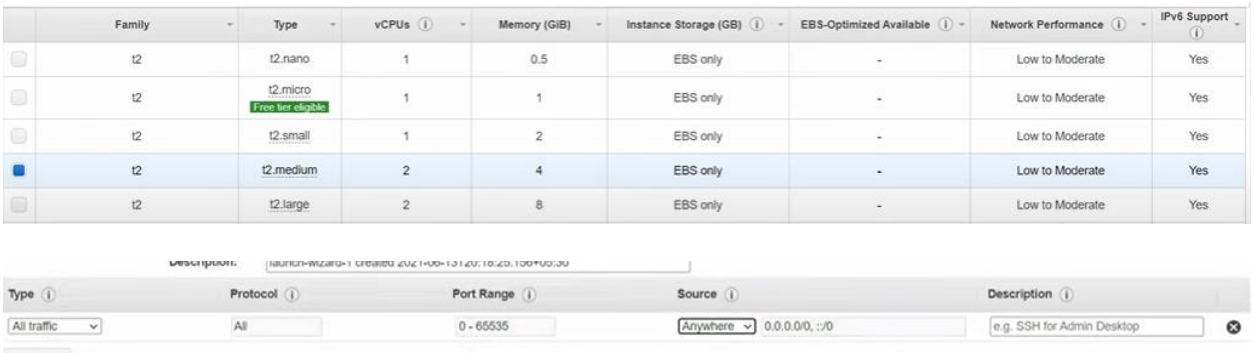
Service controller: For creating, updating, and deleting cloud provider load balancers.

Kubernetes Cluster Architecture on Cloud:



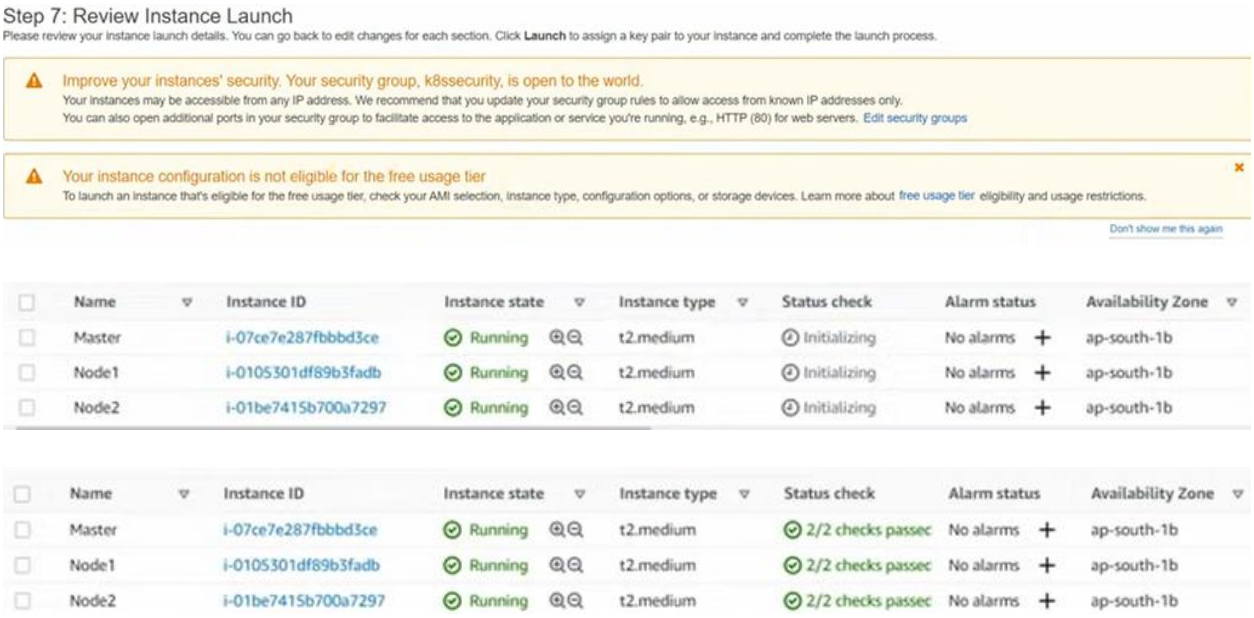
Lab (Setting up K8 cluster (Kubernetes) on AWS)

1. Launch three EC2 instances (Ubuntu OS) with default configurations except for the ones listed below. One is master node and others are slave nodes (minions). For master node minimum requirement is 4GB RAM and 2 CPUs. Select t2 medium (little bill will be generated). You can create all the three nodes with this specification or just a master node. You can select your VPC or default. Attach security group with all ports enabled (*can be specified as per requirements*).



	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
☐	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
☐	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
☐	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
☑	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
☐	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes

Type	Protocol	Port Range	Source	Description
All traffic	All	0 - 65535	Anywhere 0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop



2. Login to master node using putty or secure shell (SSH must be enabled in security group).
3. Login to other two worker nodes in same way separately.
4. Run the command following commands in all three nodes.

```
ubuntu@ip-172-31-6-165:~$ sudo su
root@ip-172-31-6-165:/home/ubuntu# apt-get update
```

5. Install https package for secure intra cluster communication on all three nodes. (control plane to individual nodes (pods)).

```
root@ip-172-31-6-165:/home/ubuntu# apt-get install apt-transport-https
```

6. Install docker on all the three nodes and check the version using docker -- version command. (*ref: docker manual*)

```
root@ip-172-31-6-165:/home/ubuntu# apt install docker.io -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

```
docker --version
```

7. Start and enable the docker service on all the three nodes using the following commands.

```
root@ip-172-31-6-165:/home/ubuntu# systemctl start docker
root@ip-172-31-6-165:/home/ubuntu#
```

```
root@ip-172-31-6-165:/home/ubuntu# systemctl start docker
root@ip-172-31-6-165:/home/ubuntu# systemctl enable docker
Synchronizing state of docker.service with SysV init with /l
Executing /lib/systemd/systemd-sysv-install enable docker
root@ip-172-31-6-165:/home/ubuntu#
```

8. Install the package from google cloud to get the **gpg key** that will enable the communication of other nodes with master node (intra cluster communication). It will be used with the source key on the node i.e., it will send the signed information to our host and hence information is exchange (after key agreement). Run the command on all three nodes to get the key and add it.

```
root@ip-172-31-6-165:/home/ubuntu# sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | s
udo apt-key add
```

9. Install the Kubernetes package using file (create a file using nano or pico command and paste the link there). Do this on all the three nodes.

```
root@ip-172-31-6-165:/home/ubuntu# nano /etc/apt/sources.list.d/kubernetes.list
```

10. After installing the package update the packages on all nodes and install all the components required to run Kubernetes on all the nodes.

```
root@ip-172-31-6-165:/home/ubuntu# apt-get update
```

```
root@ip-172-31-6-165:/home/ubuntu# apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

11. Bootstrap the master node. After running the command to initialize the K8s cluster, you will get the following output. The three commands are required to run on master node while the code is

to connect any of the worker node with the master. (Note: the code must be in single line without breaks/spaces when pasted)

```
root@ip-172-31-6-165:/home/ubuntu# kubeadm init
```

```
To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.6.165:6443 --token kl9fhu.co2n90v3rxtqllrs \
        --discovery-token-ca-cert-hash sha256:b0f8003d23dbf445e0132a53d7aa1922bdef8d553d9eca06e65
3e7c0
```

12. Run the following command on master node.

```
root@ip-172-31-6-165:/home/ubuntu# mkdir -p $HOME/.kube
root@ip-172-31-6-165:/home/ubuntu# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@ip-172-31-6-165:/home/ubuntu# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- First command will create .kube and its parent directory [-p flag is for parent directory]
- Copy the configuration file from .conf to config.
- Provide permissions to user.

13. Run the command to download the flannel packages. The cluster will be created.

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml
```

```
root@ip-172-31-6-165: /home/ubuntu
root@ip-172-31-6-165:/home/ubuntu# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/
er/Documentation/kube-flannel.yml
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.2
podsecuritypolicy.policy/psp.flannel.unprivileged created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
root@ip-172-31-6-165:/home/ubuntu# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/
er/Documentation/k8s-manifests/kube-flannel-rbac.yml
Warning: rbac.authorization.k8s.io/v1beta1 ClusterRole is deprecated in v1.17+, unavailable in v1.22+
e rbac.authorization.k8s.io/v1 ClusterRole
clusterrole.rbac.authorization.k8s.io/flannel configured
Warning: rbac.authorization.k8s.io/v1beta1 ClusterRoleBinding is deprecated in v1.17+, unavailable in
22+; use rbac.authorization.k8s.io/v1 ClusterRoleBinding
clusterrolebinding.rbac.authorization.k8s.io/flannel configured
```

14. Now run the command you get from the master node and run it on the other two nodes. You will see the connection is established.

```
root@ip-172-31-15-102:/home/ubuntu# kubeadm join 172.31.6.165:6443 --token kl9fhu.co2n90v3rxtqllrs
overy-token-ca-cert-hash sha256:b0f8003d23dbf445e0132a53d7aa1922bdef8d553d9eca06e65c928322b3e7c0
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recomm
driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-confi
aml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

15. To check the connected nodes with the master run the following command.

```
root@ip-172-31-6-165:/home/ubuntu# kubectl get nodes
NAME                STATUS     ROLES                  AGE     VERSION
ip-172-31-15-102    Ready      <none>                 74s     v1.21.1
ip-172-31-3-98      NotReady   <none>                 28s     v1.21.1
ip-172-31-6-165     Ready      control-plane,master   5m15s   v1.21.1
```

```
root@ip-172-31-6-165:/home/ubuntu# kubectl get nodes
NAME                STATUS   ROLES                  AGE     VERSION
ip-172-31-15-102    Ready    <none>                 93s     v1.21.1
ip-172-31-3-98      Ready    <none>                 47s     v1.21.1
ip-172-31-6-165     Ready    control-plane,master   5m34s   v1.21.1
```

Lab (Amazon Elastic Kubernetes Service)