Linux:

1. Why /etc/resolv.conf and /etc/hosts files are used?

**/etc/resolv.conf:** It is used to configure DNS name servers as it contains the details of the nameserver i.e., details of your DNS server. The DNS server is then used to resolve the hostname of the IP address.   
  
**/etc/hosts:** It is used to map or translate any hostname or domain name to its relevant IP address.

**Case insensitive search :**The -i option enables to search for a string case insensitively in the give file. It matches the words like “UNIX”, “Unix”, “unix”. 

**$grep -i "UNix" geekfile.txt**

**Display the file names that matches the pattern :**We can just display the files that contains the given string/pattern. 

Suppose you want to find all words with the letters "qu" in them. Typing:

|  |
| --- |
| $ **grep 'qu\*' list** |

**$grep -l "unix" \***

**or**

**$grep -l "unix" f1.txt f2.txt f3.xt f4.txt**

**ls -lh**

total 148

-rw------- 1 uucp bin 0 Nov 26 09:25 aculog

-r--r--r-- 1 root other 342K Nov 26 13:56 lastlog

-rw-r--r-- 1 root root 20K Nov 26 13:55 messages

-rw-r--r-- 1 root bin 3.3K Nov 26 13:56 utmpx

-rw-r--r-- 1 adm adm 19K Nov 26 13:56 wtmpx

I have a text file containing the word 'entropy' in this directory, is there anything like SEARCH?

*Solution*: yes, try

|  |
| --- |
| $ grep -l 'entropy' \* |

###### You are supposed to print the content of directory in long format listing, showing hidden/dot files. How will you achieve this?

**Ans:** We need to use option ‘**-a**‘ (list hidden files) and ‘**-l**‘ (long listing) together with command ‘**ls**‘.

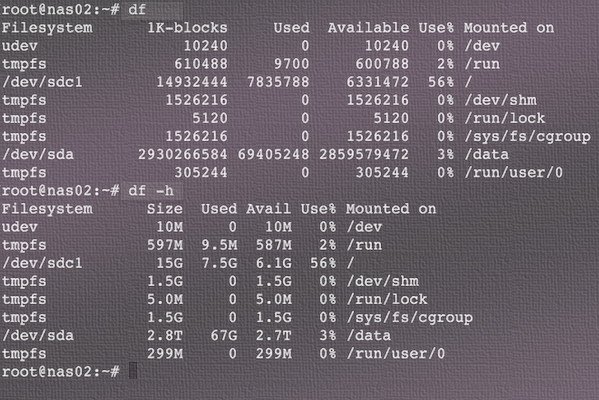
###### ls -h displays file size in KB/MB

# ls -la

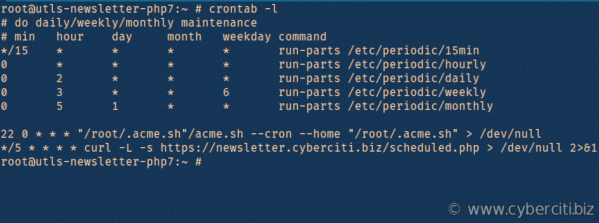
linux check disk space:

1. **df -H**

Sample outputs:

~~[](https://www.cyberciti.biz/faq/linux-check-disk-space-command/df-command-outputs/)~~

### How to Display all jobs in cron / crontab

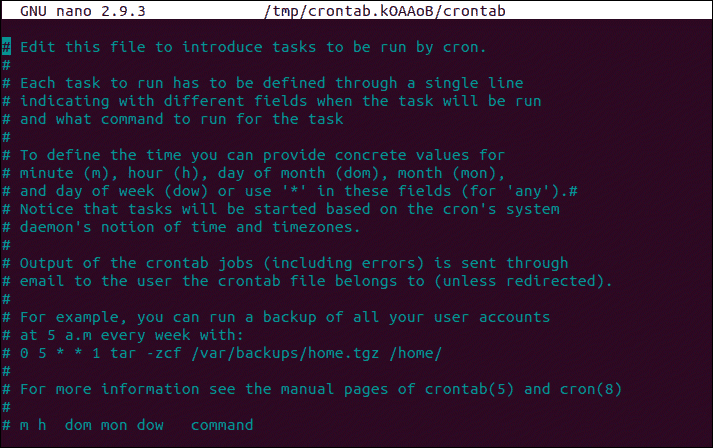
I run the following as root user:  
crontab -l  


Add a cron job:

1) edit cron file:

crontab –e

You can add any number of scheduled tasks, one per line.



**Replacing all the occurrence of the pattern in a line :**The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

**$sed 's/unix/linux/g' geekfile.txt**

**AWK Operations:**   
(a) Scans a file line by line   
(b) Splits each input line into fields   
(c) Compares input line/fields to pattern   
(d) Performs action(s) on matched lines

**Print the lines which match the given pattern.**

$ awk '/manager/ {print}' employee.txt

**Output:**

ajay manager account 45000

varun manager sales 50000

amit manager account 47000

awk '{print $1,$4}' employee.txt

**Output:**

ajay 45000

sunil 25000

varun 50000

amit 47000

tarun 15000

deepak 23000

sunil 13000

satvik 80000

PING (Packet Internet Groper) command is used to check the network connectivity between host and server/host. This command takes as input the IP address or the URL and sends a data packet to the specified address with the message “PING” and get a response from the server/host this time is recorded which is called latency.

**EX: ping** [**www.geeksforgeeks.org**](http://www.geeksforgeeks.org)

To stop pinging we should use ctrl+c otherwise it will keep on sending packets. 

A picture containing text

Description automatically generated

**Nslookup**(stands for “Name Server Lookup”) is a useful command for getting information from DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS related problems.

**Options of nslookup command:**

* **nslookup google.com :**  
  nslookup followed by the domain name will display the “A Record” (IP Address) of the domain. Use this command to find the address record for a domain. It queries to domain name servers and get the details.  
  Text

  Description automatically generated
* **nslookup 192.168.0.10 :** Reverse DNS lookup

You can also do the reverse DNS look-up by providing the IP Address as argument to nslookup.  
Text

Description automatically generated

Terraform:

Common commands:

* **terraform init**: Prepare your working directory for other commands
* **terraform validate**: Check whether the configuration is valid
* **terraform plan**: Show changes required by the current configuration
* **terraform apply**: Create or update infrastructure
* **terraform destroy**: Destroy previously-created infrastructure

Kubernetes:

### ****What is Kubelet?****

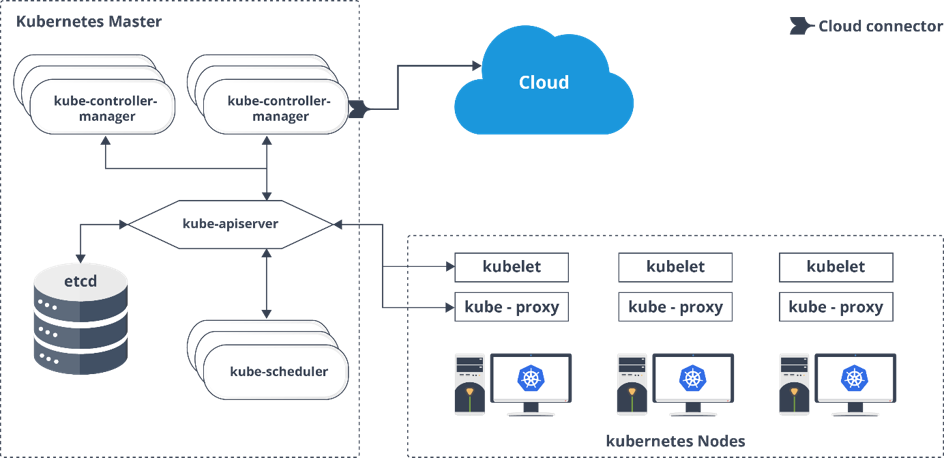
This is an agent service which runs on each node and enables the slave to communicate with the master. So, Kubelet works on the description of containers provided to it in the PodSpec and makes sure that the containers described in the PodSpec are healthy and running.

### kube-proxy

[kube-proxy](https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/) maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster. It is essentially the core networking component of Kubernetes and is responsible for **ensuring that communication is maintained efficiently across all elements of the cluster.**

### ****What are the different components of Kubernetes Architecture?****

The [Kubernetes Architecture](https://www.edureka.co/blog/kubernetes-architecture/) has mainly 2 components – the master node and the worker node. As you can see in the below diagram, the master and the worker nodes have many inbuilt components within them. The master node has the kube-controller-manager, kube-apiserver, kube-scheduler, etcd. Whereas the worker node has kubelet and kube-proxy running on each node.



**What is the role of kube-apiserver and kube-scheduler?**

**Api-server:**

The API server is a component of the Kubernetes [control plane](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane) that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

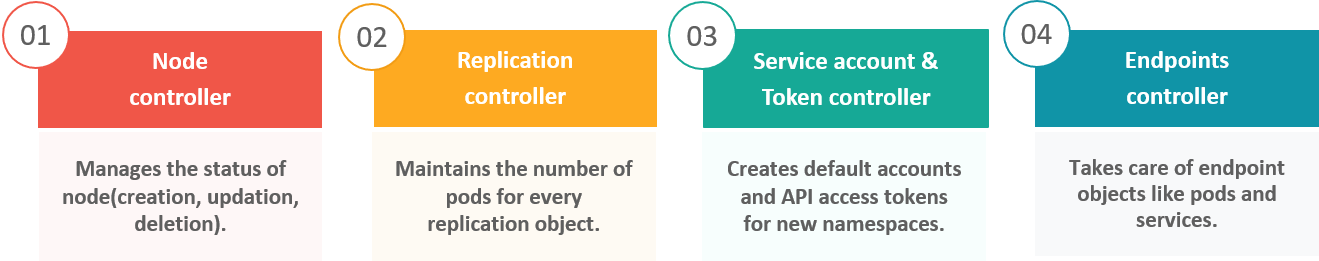
The main implementation of a Kubernetes API server is [kube-apiserver](https://kubernetes.io/docs/reference/generated/kube-apiserver/). kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

Kube-scheduler:

Control plane component that watches for newly created [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) with no assigned [node](https://kubernetes.io/docs/concepts/architecture/nodes/), and selects a node for them to run on.

**Q5.  Can you brief about the Kubernetes controller manager?**

Multiple controller processes run on the master node but are compiled together to run as a single process which is the Kubernetes Controller Manager. So, Controller Manager is a daemon that embeds controllers and does namespace creation and garbage collection. It owns the responsibility and communicates with the API server to manage the end-points.

So, the different types of controller manager running on the master node are :  


In Kubernetes, controllers are **control loops that watch the state of your cluster, then make or request changes where needed**. Each controller tries to move the current cluster state closer to the desired state.

**Q6.  What is ETCD?**

Kubernetes uses etcd **to store all its data** – its configuration data, its state, and its metadata

## **Question 8. What is the difference between a daemonset, a deployment, and a replication controller?**

DaemonSet is a controller similar to ReplicaSet that ensures that the pod runs on all the nodes of the cluster. If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod. A [daemonset](https://www.bluematador.com/blog/an-introduction-to-kubernetes-daemonsets)ensures that all nodes you select are running exactly one copy of a pod.

A Deployment provides declarative updates for [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) and [ReplicaSets](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/" \t "_blank). You describe a desired state in a Deployment, and the Deployment [Controller](https://kubernetes.io/docs/concepts/architecture/controller/) changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

The replication controller specifies how many exact copies of a pod should be running in a cluster. It differs from a deployment in that it does not offer pod health checks, and the rolling update process is not as robust.

## States of a Pod

Through its lifecycle, a Pod can attain following states:

**Pending:**The pod is accepted by the Kubernetes system but its container(s) is/are not created yet.

**Running:**The pod is scheduled on a node and all its containers are created and at-least one container is in Running state.

**Succeeded:** All container(s) in the Pod have exited with status 0 and will not be restarted.

**Failed:** All container(s) of the Pod have exited and at least one container has returned a non-zero status.

**CrashLoopBackoff:** The container fails to start and is tried again and again.

**Init Containers**

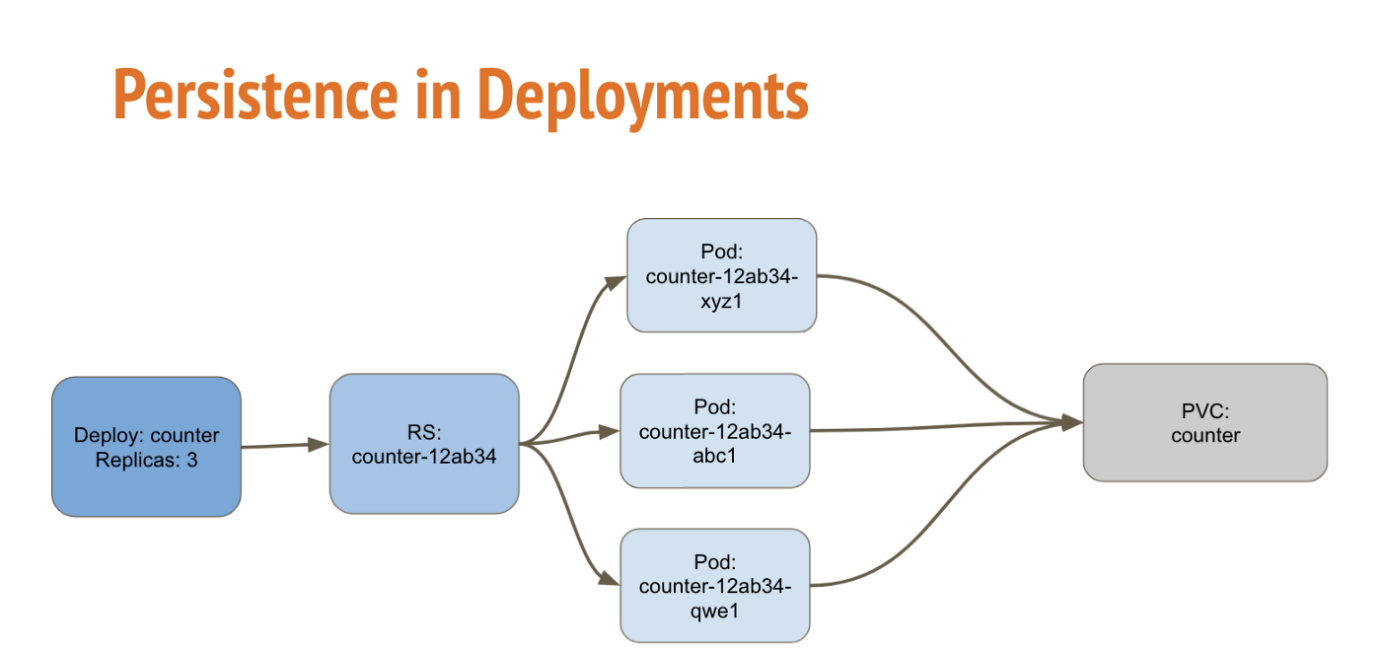
Init containers are containers which are run before the main application container gets started. They have two important characteristics:

* They always run to completion.
* Each init container must complete before the next one is started.

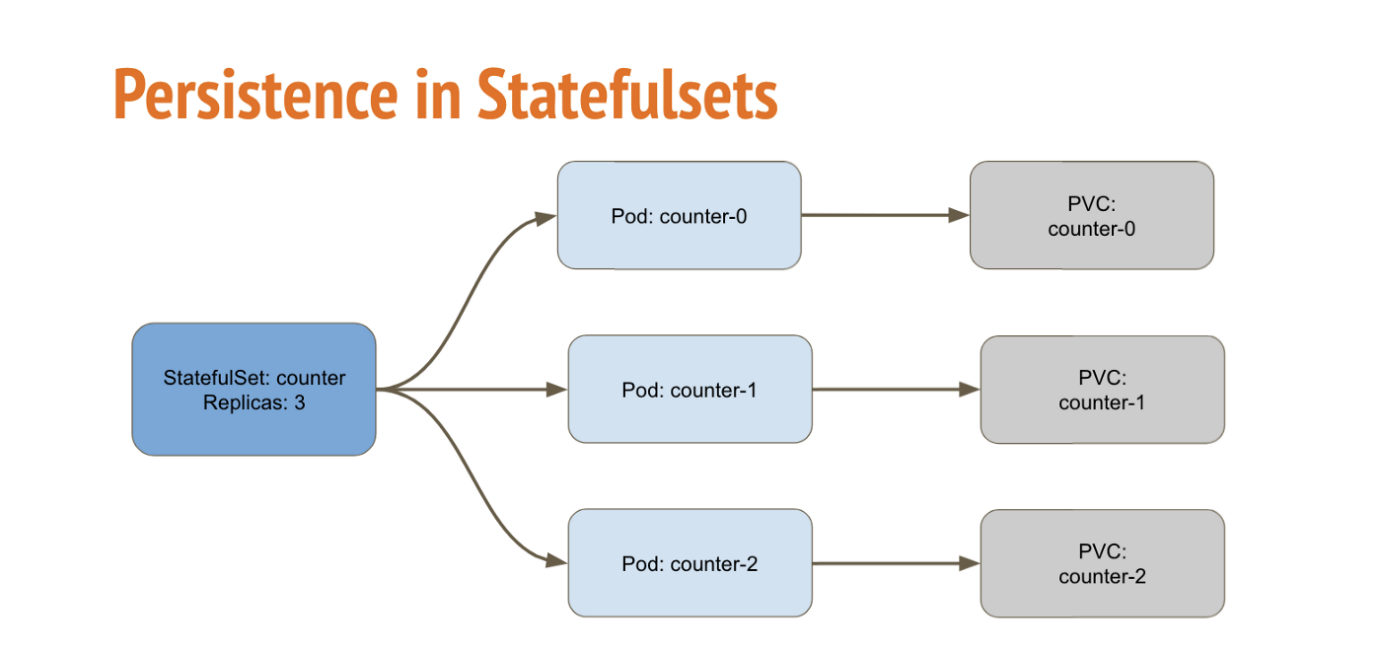
Init containers can be useful when some initial actions need to be run before the main container in the pod starts.

For example: copying config files and updating config values. Init containers use different Linux namespaces, so they have a different filesystem view so they can be given access to secrets which may not be desirable for sharing within the app container.

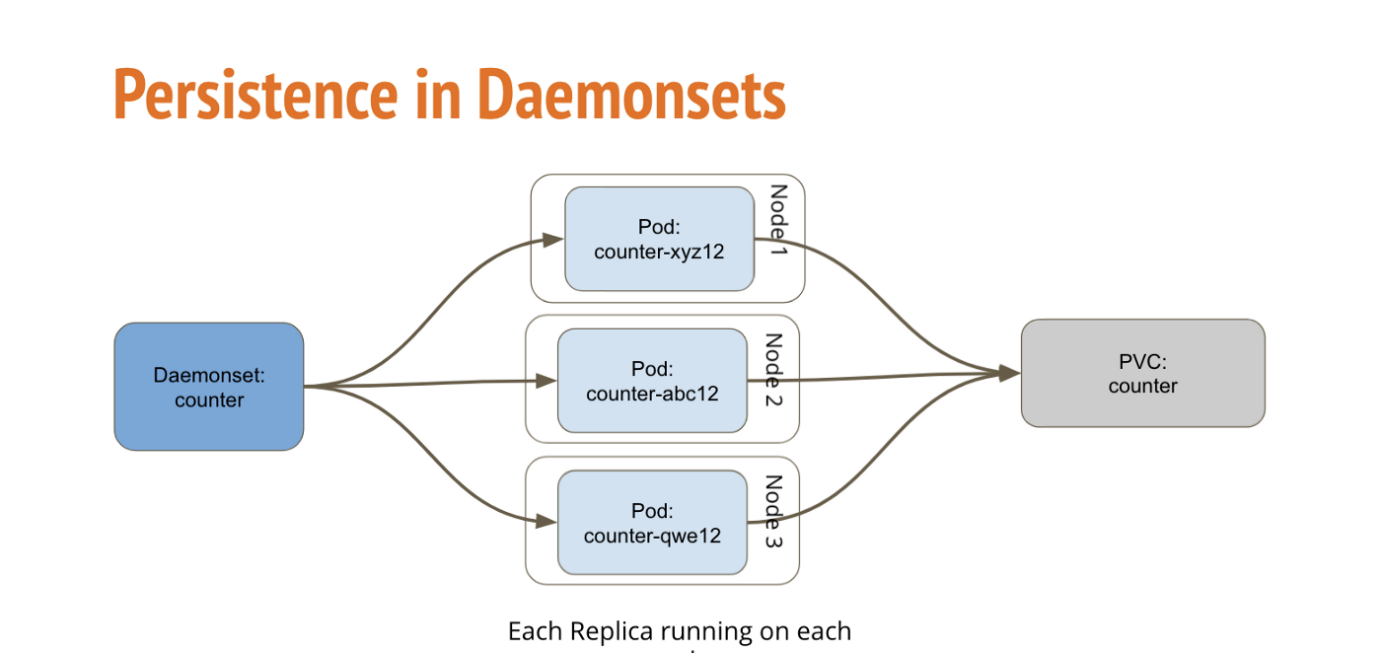
Deployment is a resource to deploy a stateless application, if using a PVC, all replicas will be using the same Volume and none of it will have its own state.

[](https://i.stack.imgur.com/oGm0m.png)

Statefulsets is used for Stateful applications, each replica of the pod will have its own state, and will be using its own Volume.

[](https://i.stack.imgur.com/5pfeD.png)

DaemonSet is a controller similar to ReplicaSet that ensures that the pod runs on all the nodes of the cluster. If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod.

[](https://i.stack.imgur.com/cBVFi.png)

I have written about the detailed differences between Deployments, StatefulSets & Daemonsets, and how to deploy a sample application using these Resources [K8s: Deployments vs StatefulSets vs DaemonSets](https://medium.com/stakater/k8s-deployments-vs-statefulsets-vs-daemonsets-60582f0c62d4).

## **What are Kubernetes Services?**

A **Kubernetes service** is a logical abstraction for a deployed group of pods in a cluster (which all perform the same function).

Since pods are ephemeral, a service enables a group of pods, which provide specific functions (web services, image processing, etc.) to be assigned a name and unique IP address (clusterIP). As long as the service is running that IP address, it will not change. Services also define policies for their access.

## **In Kubernetes, what is the difference between a service and a deployment?**

In [Kubernetes](https://www.vmware.com/topics/glossary/content/kubernetes), a deployment is a method of launching a pod with containerized applications and ensuring that the necessary number of replicas is always running on the cluster.

On the other hand, a service is responsible for exposing an interface to those pods, which enables network access from either within the cluster or between external processes and the service.

## **What are the components of a Kubernetes services?**

Kubernetes services connect a set of pods to an abstracted service name and IP address. Services provide discovery and routing between pods. For example, services connect an application front-end to its backend, each of which running in separate deployments in a cluster. Services use labels and selectors to match pods with other applications. The core attributes of a Kubernetes service are:

* A label selector that locates pods
* The clusterIP IP address and assigned port number
* Port definitions
* Optional mapping of incoming ports to a targetPort

Services can be defined without pod selectors. For example, to point a service to another service in a different namespace or cluster.

## **What are the types of Kubernetes services?**

* **ClusterIP**. Exposes a service which is only accessible from within the cluster.
* **NodePort**. Exposes a service via a static port on each node’s IP.
* **LoadBalancer**. Exposes the service via the cloud provider’s load balancer.
* **ExternalName**. Maps a service to a predefined externalName field by returning a value for the CNAME record.

### What is the Kubernetes ClusterIP service?

ClusterIP is the default type of service, which is used to expose a service on an IP address internal to the cluster. Access is only permitted from within the cluster.

#### **What is a Kubernetes Headless Service?**

Services that do not need load balancing and only expose a single IP can create a ‘headless’ service by specifying “none” as the clusterIP.

Headless services can be defined with selectors, in which case endpoint records are created in the API that modify the DNS to return addresses that point to pods that are exposing the service. Headless services without selectors don’t create endpoint records. The DNS system configures either the CNAME record or a record for endpoints with the same name as the service.

### What is a Kubernetes NodePort service?

NodePorts are open ports on every cluster node. Kubernetes will route traffic that comes into a NodePort to the service, even if the service is not running on that node. NodePort is intended as a foundation for other higher-level methods of ingress such as load balancers and are useful in development.

### What is a Kubernetes ExternalName service?

ExternalName services are similar to other Kubernetes services; however, instead of being accessed via a clusterIP address, it returns a CNAME record with a value that is defined in the externalName: parameter when creating the service.

### What is a Kubernetes Load Balancer service?

For clusters running on[public cloud](https://www.vmware.com/topics/glossary/content/public-cloud) providers like AWS or Azure, creating a load LoadBalancer service provides an equivalent to a clusterIP service, extending it to an external load balancer that is specific to the cloud provider. Kubernetes will automatically create the load balancer, provide firewall rules if needed, and populate the service with the external IP address assigned by the cloud provider.

## **How do Kubernetes services work?**

Services simply point to pods using labels. Since services are not node-specific, a service can point to a pod regardless of where it runs in the cluster at any given moment in time. By exposing a service IP address as well as a DNS service name, the application can be reached by either method as long as the service exists.

## **How do you define a Kubernetes service?**

Services are defined in YAML, as are all Kubernetes objects. Suppose you deployed pods running a back-end service to process data coming from a web front end. To expose a service named ‘service-backend’ on the deployment ‘deployment-backend’ you would use:

**apiVersion:**v1  
**kind:**Service  
**metadata:**  
**name:**service-backend  
**spec:  
ports:**- **port:**4000  
**protocol:**TCP  
**targetPort: 333**  
**selector:  
run:**deployment-backend  
**type:**ClusterIP

The service ‘service-backend’ will be created, and any pod in the cluster can access it on their port 333 via [http://service-backend:4000](http://service-backend:4000/), or at the cluster’s IP address using port 4000.

Kubernetes services can also be created using the ‘kubectl expose’ command, which does not require a YAML file. The same service can be created using the command:

kubectl expose deployment deployment-backend  - - port=333- - target-port=4000    - - name=service-backend

## **How do you access a Kubernetes service?**

There are two ways to discover a Kubernetes service:

* **DNS (most common)**: The DNS method is the recommended method of discovering services. To use this method, a DNS server must first be installed on the cluster. The DNS server monitors the Kubernetes API, and when a new service is created its name becomes available for easy resolution for requesting applications.
* **ENV variable**: This method relies on the kubelet adding environment variables for each active service for every node a pod is running on.

### Overview of Kubernetes Services

Kubernetes [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) are mortal. Pods in fact have a [lifecycle](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/). When a worker node dies, the Pods running on the Node are also lost. A [ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/) might then dynamically drive the cluster back to desired state via creation of new Pods to keep your application running. As another example, consider an image-processing backend with 3 replicas. Those replicas are exchangeable; the front-end system should not care about backend replicas or even if a Pod is lost and recreated. That said, each Pod in a Kubernetes cluster has a unique IP address, even Pods on the same Node, so there needs to be a way of automatically reconciling changes among Pods so that your applications continue to function.

A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable a loose coupling between dependent Pods. A Service is defined using YAML [(preferred)](https://kubernetes.io/docs/concepts/configuration/overview/#general-configuration-tips) or JSON, like all Kubernetes objects. The set of Pods targeted by a Service is usually determined by a *LabelSelector* (see below for why you might want a Service without including selector in the spec).

Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a type in the ServiceSpec:

* *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
* *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.
* *LoadBalancer* - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.
* *ExternalName* - Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up. This type requires v1.7 or higher of kube-dns, or CoreDNS version 0.0.8 or higher.

**The following are some of the Kubernetes Objects:**

* pods.
* Namespaces.
* ReplicationController (Manages Pods)
* DeploymentController (Manages Pods)
* StatefulSets.
* DaemonSets.
* Services.
* ConfigMaps.

A persistent volume is a piece of storage in a cluster that an administrator has provisioned. It is a resource in the cluster, just as a node is a cluster resource. A persistent volume is a volume plug-in that has a lifecycle independent of any individual pod that uses the persistent volume. This API object captures details of implementation if the storage like NFS or a cloud-provider-specific storage system.

## **Why data persistence is important**

[Containers](https://www.netapp.com/devops-solutions/what-are-containers) were developed to be stateless, ephemeral, lightweight tools, only megabytes in size, to speed application launch. However, this design is problematic when data needs to persist after the container goes away. And this problem increases as the scale of container management deployments with tools like Docker, Mesosphere, and Kubernetes grows.

Users are now executing hundreds (sometimes thousands) of nodes in clusters, making data management even more difficult given the unpredictable lifecycles of containers.

To ensure that data persists well beyond a container’s lifecycle, the best practice is to separate data management from containers. There are three approaches to data persistence in a container environment:

* Storage plug-ins
* Data volume containers
* Building a local directory mount into the container as a data directory

## Why does a CrashLoopBackOff occur?

* The application inside the container keeps crashing
* [Some type of parameters of the pod or container have been configured incorrectly](https://stackoverflow.com/questions/41604499/my-kubernetes-pods-keep-crashing-with-crashloopbackoff-but-i-cant-find-any-lo)
* [An error has been made when deploying Kubernetes](https://stackoverflow.com/questions/35537834/debugging-a-container-in-a-crash-loop-on-kubernetes)

## How to Limit Container Resource Usage in Kubernetes

Within Kubernetes, containers are scheduled as pods. By default, a pod in Kubernetes will run with no limits on CPU and memory in a default namespace. This can create several problems related to contention for resources, the two main ones being:

1. There is no control of how much resources each pod can use. Some images might be more resource heavy or have certain "minimum resource" requirements that we would like to see guaranteed.
2. When different teams run different projects on the same cluster, there is no control how much resources each team can use.

These issues can be addressed respectively in Kubernetes in the following way:

1. Developers can control the amount of CPU and memory resources per pod or container by setting [resource requests and limits](https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/) in the pod configuration file.
2. Cluster administrators can create namespaces for different teams and set [resource quota](https://v1-9.docs.kubernetes.io/docs/concepts/policy/resource-quotas/) (defined by a **ResourceQuota**object) per namespace. This limits the amount of objects that can be created in a namespace, as well as the total amount of resources that may be consumed by pods in that namespace.

Kubernetes questions:

1. Architecture of Kubernetes
2. Explain more about etcd
3. What is replica set?
4. What is deamonset?
5. What is namespace?
6. What are the kubectl commands you have used in production level env?
7. Difference between stateful and stateless in Kubernetes.
8. What is ingress
9. What is ingress controller
10. What is load balancer
11. Difference between ingress and load balancer
12. Can 2 containers have same IP? – no, only 1 IP per pod
13. How to enable communication only between pods and not to outside world? – cluster IP concept
14. How to increase node size? – we cannot vertically scale a node, we have to create another node with higher size and the delete the old node.
15. How to restrict a node from assigning further pods? – “cordon” concept
16. How to assigned pods to specific nodes? – “nodeselector” label for basic features and “affinity” for enhanced features
17. What is configmaps?
18. What is secret?
19. What is PV and PVC?
20. What is packet manager? What is HELM?
21. How to revert a deployment? –

kubectl rollout undo daemonset <daemonset-name> --to-revision=<revision>

1. How to check CPU and memory in pods:

$ kubectl top pod --namespace web-app

NAME CPU (CORES) MEMORY (BYTES)

nginx-84ac2948db-12bce 14m 1Mi

nginx-84ac2948db-bc9ae 14m 1Mi

### [View metric snapshots using kubectl top](https://www.datadoghq.com/blog/how-to-collect-and-graph-kubernetes-metrics/#view-metric-snapshots-using-kubectl-top)

Once Metrics Server is deployed, you can retrieve compact metric snapshots from the Metrics API using kubectl top. The kubectl top command returns current CPU and memory usage for a cluster’s pods or nodes, or for a particular pod or node if specified.

For example, you can run the following command to display a snapshot of near-real-time resource usage of all cluster nodes:

kubectl top node

NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

gke-john-m-research-2-default-pool-42552c4a-fg80 89m 9% 781Mi 67%

gke-john-m-research-2-default-pool-42552c4a-lx87 59m 6% 644Mi 55%

gke-john-m-research-2-default-pool-42552c4a-rxmv 53m 5% 665Mi 57%

This output shows three worker nodes in a GKE cluster. Each line displays the total amount of CPU (in cores, or in this case m for millicores) and memory (in MiB) that the node is using, and the percentages of the node’s allocatable capacity those numbers represent. Likewise, to query resource utilization by pod in the web-app namespace, run the command below (note that if you do not specify a namespace, the default namespace will be used):

COPY

kubectl top pod --namespace web-app

NAME CPU(cores) MEMORY(bytes)

nginx-deployment-76bf4969df-65wmd 12m 1Mi

nginx-deployment-76bf4969df-mmqvt 16m 1Mi

1. *# List PersistentVolumes sorted by capacity*

kubectl get pv --sort-by=.spec.capacity.storage

1. How to select data from sql without duplicates:

Select distinct username from userdata where userstatus=’active’ order by id

1. Move resources to a new resource group:

When you move a resource, you change its resource ID.

The following scenarios aren't yet supported for moving VMs:

-Virtual Machine Scale Sets with Standard SKU Load Balancer or Standard SKU Public IP can't be moved.

-Virtual machines in an existing virtual network can't be moved to a new subscription when you aren't moving all resources in the virtual network.

-Virtual machines created from Marketplace resources with plans attached can't be moved across subscriptions. For a potential workaround, see Virtual machines with Marketplace plans.

-Virtual machines in an availability set can't be moved individually.

linux questions:

1. Give at least 5 linux commands
2. What is grep, sed and AWK commands?

Terraform:

***resource “aws\_instance” “testinstance” {***

***ami = “ami — 028598a84ca601344”***

***instance\_type = “m5.4xlarge”***

***subnet\_id = “subnet-06a4dd555ee331cac”***

***associate\_public\_ip\_address “false”***

***vpc\_security\_group\_ids = [ “sg-0b7a71dca0a95842f” ]***

***key\_name “testinstance”***

***tags {***

***Name = “testinstance”***

***}***

***}***

Kubernetes:

Pod definition YAML:

apiVersion: v1

kind: Pod

metadata:

 name: rss-site

 labels:

   app: web

spec:

**containers:**

**- name: front-end**

**image: nginx**

**ports:**

**- containerPort: 80**

**- name: rss-reader**

**image:** nickchase/rss-php-nginx:v1

**ports:**

**- containerPort: 88**

Deployment YAML:

apiVersion: apps/v1

kind: Deployment

metadata:

name: rss-site

labels:

app: web

spec:

replicas: 2

selector:

matchLabels:

app: web

template:

metadata:

labels:

app: web

spec:

containers:

- name: front-end

image: nginx

ports:

- containerPort: 80

- name: rss-reader

image: nickchase/rss-php-nginx:v1

ports:

- containerPort: 88