

**2Q) what is the namespace in kubernetes?**

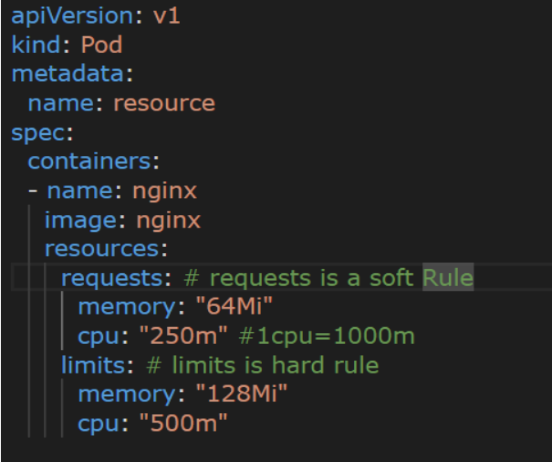
**ANS)**  Namespace is logical isolation in kubernetes.Ideally we will create one namespace for one project. So resources will be to the isolated namspace level.

**3Q) what is the deference between pod and containers?**

**Ans)** pod is a smallest deployble unit in kubernetes. It can have multiple containers. Containers in a podshare the same ip address , volumes. This feature is useful when you want to keep multiple apps together with the same IP,volume, side car containers,proxies etc.

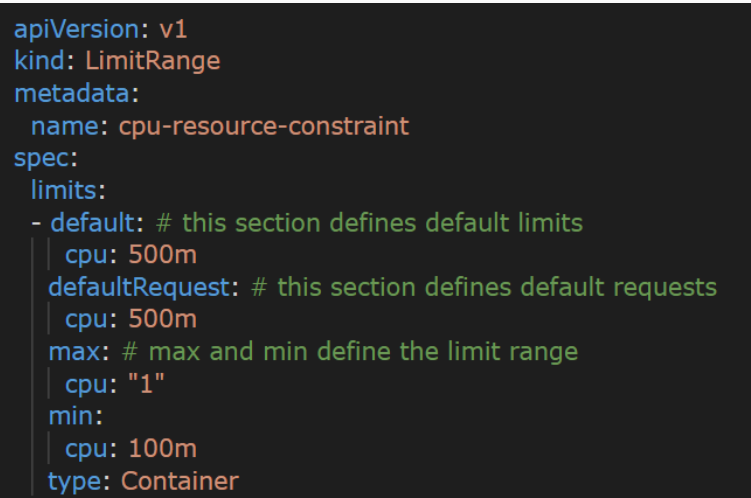
**4Q) how do you restrict pod and containers to use limited host resources?**

Ans) we can limit the resources to the pod by mentioning limits and requests YAML that can restrict containers not to use more CPU and Memory.



**5Q) how can you restrict resources to pod if the engineer forgets to mention them in pod definition?**

**Ans)** we have a resource called LimitRange in kubernetes. As administrators we can restrict the resoures at the namespace level. So if the enginners to forgets to mention limit range will be applied autometically.

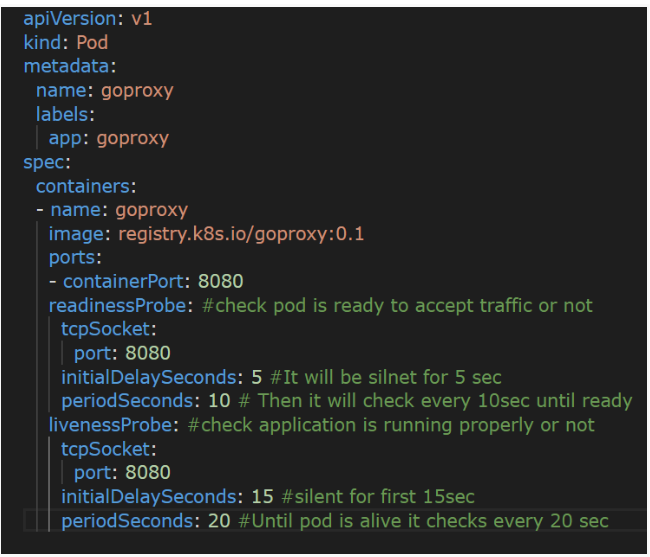


**6Q) how to configure health checks in pod?**

Ans) we can configure using readiness probe and liveness probe.

Readiness is to check at the time of container creation whether pod is ready to take traffic or not.

Liveness probe will kick in when readiness is done. It will be checked at a particular interval to check if the application is working properly or not.



**7Q) what is the image pull policy in pod?**

Ans) we have 3 types of image pull policy are available.

* IfNotPresent – if it is not present then it will pull, if available already it will not pull.
* Always – whether image is available or not it pulls the image.
* Never – if present in host it can run the pod, otherwise error.

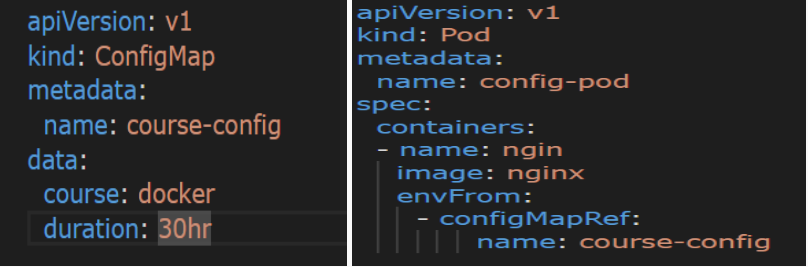
Keep always is the best way.

**8Q) how can you provide env variables to the pod in a better way?**

Ans) we can create configMap resource. We can refer to the configMap in the pod so that the entire config will be loaded at the at the time of container creation.

In future if we need to add more configurations, do the changes in configMap restart the pod so that there is no need to make changes in

Pod definition and deployment.



**9Q) what are the secrets in Kubernetes?**

Ans) secrets are confidential information like DB username, password, etc. we can store this info in secret and refer it to a Pod definition like configMap.

Secrets are base64 encoded in Kubernetes. By default, it is not secure, so we use third party solutions like AWS secret manager and Hashicorp Vault, etc.



**10Q) what is service in Kubernetes? How many types are there?**

Ans) Pods are normally ephemeral in Kubernetes.so pod IP will be changed Every time it is created. To achieve pod to pod communication we can’t rely on IP addresses. Solution is service.

service can have name and port , when a pod is created it can go attached to service itself. It acts as a service mesh.

Multiple pods can be attached to service based on labels selectors, service acts as Loadbalancer between pods. 3 types of services available.

**Cluster IP**: Default type. It can get one IP address. We can configure cluster IP service to achieve pod to pod communication. But it can’t be accessed over the internet.

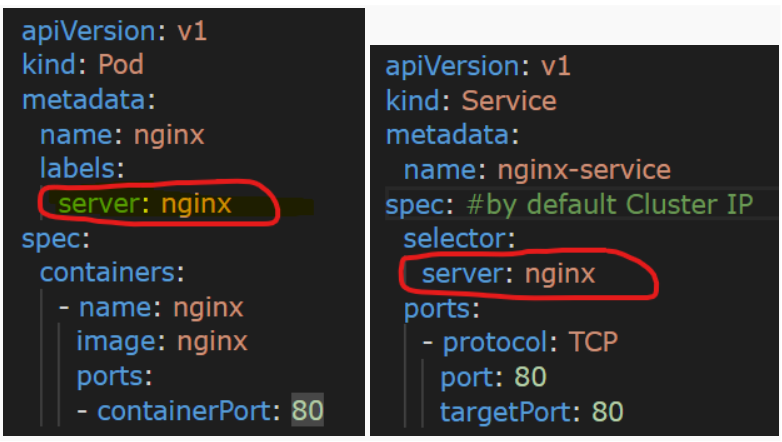
**NodePort:** Nodeport by default creates cluster IP in background. When we say Nodeport a port will be opened on each and every node. This port will be redirected to the cluster IP. NodePort can be accessed over the internet.

**Loadbalancer:** we can create Loadbalancer through cloud providers like

Aws, gcp, azure, etc.

LoadBalancer By default creates NodePort and Cluster IP in the background.

LoadBalancer 🡪NodePort 🡪 Cluster IP

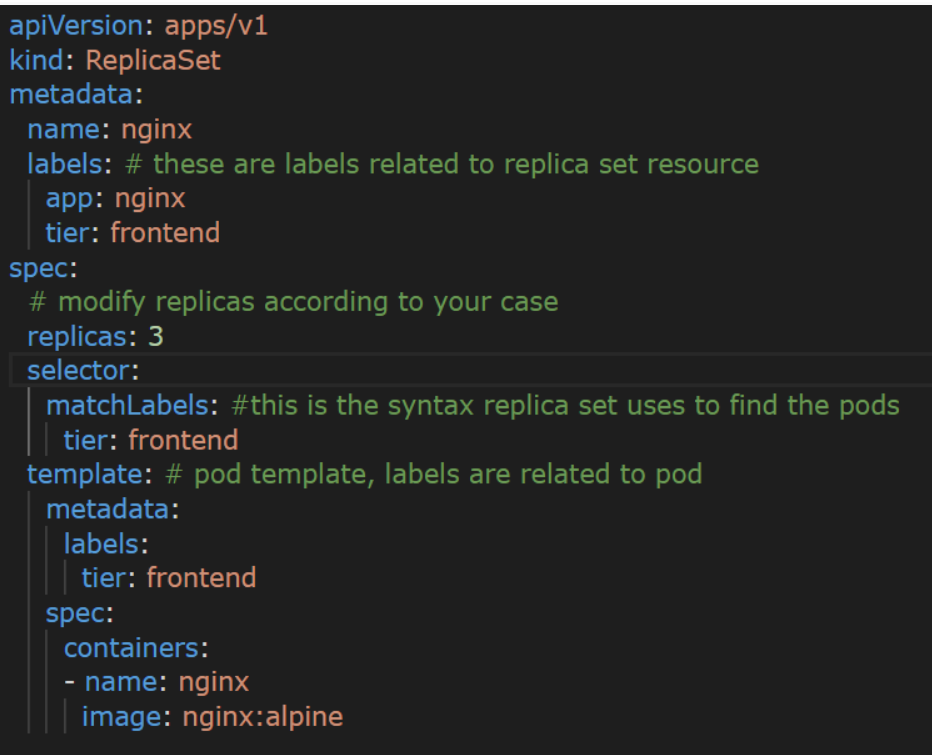


**11Q) What is replicaset in Kubernetes?**

**Ans)** by default it can increase the number of pods, if you want multiple

Instances of pods to serve high traffic we can use replicaset. It guarantees the declared no of pods will run always. It is most important feature of high availability and autoscale.

When using ReplicaSet we no need to have separate pod, it is included Under template section.



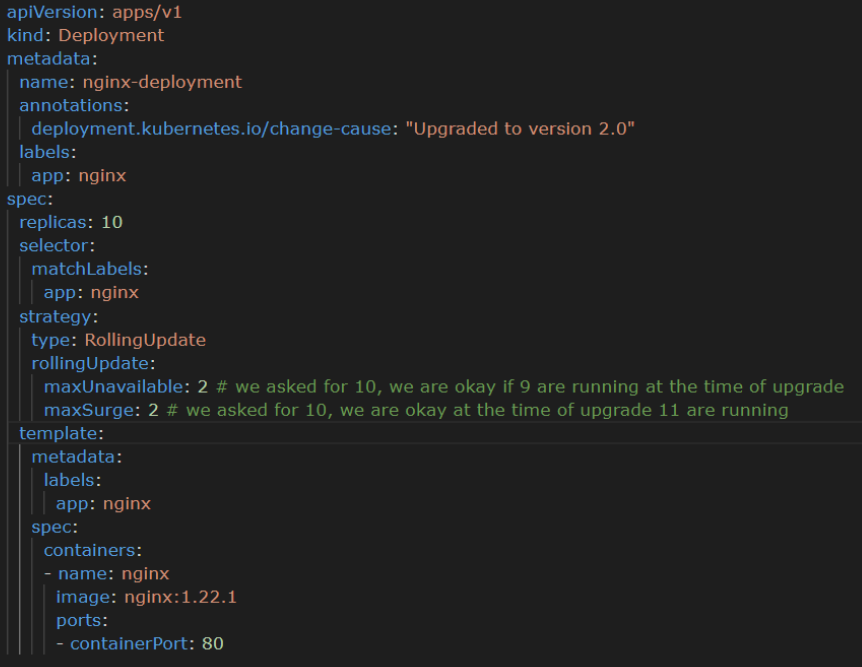
**12Q) what is the deployment in kubernetes?**

Ans) Deployment is most important resource in kubernetes maintain the applications. It ensures zero downtime of application. Deployment

Creates ReplicaSet in background to maintain number of replicas.

We can use deployment for stateless applications. Finally, deployment is the highest resource in kubernetes to maintain

* Availability – Make sure desired number of replicas are always available.
* Scalability – can be used to scale at runtime based on traffic
* Maintainability – we can do the updates with new image versions



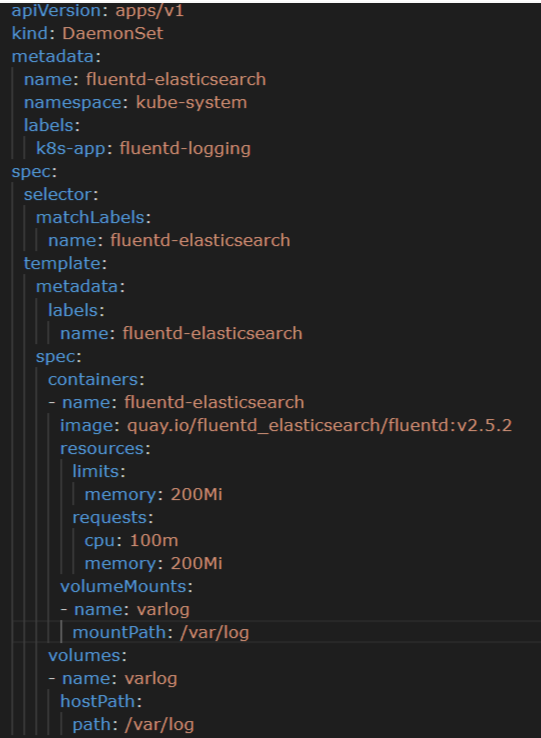
**13Q) what is Daemon set in kubernetes?**

Ans) Daemon set is similar to deployment. But there is only one difference.

Deployment makes pods available in any node.

But Daemon set makes sure one replica of pod runs in each and every node in kubernetes cluster.

This feature is helpful for Cluster level administration like collecting the metrics, logs of all nodes, etc.

****

**14Q) what are the volume types in kubernetes?**

Ans) data is going to store inside the kubernetes cluster

**EmptyDir:** storage is going to be inside pod only. This is completely ephemeral . when pod is deleted this memory also deleted.

So what is the use of creating emptyDir.

We have pod and side car container and side car will access the actual pod logs and send to elastic OpenSearch. For this purpose emptyDir is the solution.

If you create am emptyDir it will create the temporary storage inside the pod. side car also access for it.it can refer pod is deleted after ½ hour so with help of side car those ½ hour pod logs push into the elastic OpenSearch. Once main pod is deleted no need of log file. There will be no logs that is why emptyDir is perfect solution for sidecar.

This is a volume created when pod is created. It is deleted when pod is deleted.

NGINX container is a main container

Sidecar container is filebeat

Filebeat should access NGINX logs and push into AEOS.

Conclusion: emptyDir is memory inside the pod it is used to share the volume b/w multiple containers. simple use case is for a filebeat sidecar

We can define emptyDir volume that can access the nginx logs it will push into AEOS. Not only nginx logs any main container logs. This is about emptyDir.

**Hostpath:** hostpath volumes present many security risks and it is a best practice to avoid The use of hostpath when possible. Because hacker access one pod access that is hostpath access and easily get data of other projects as well.

There is one valid use case for this daemonset. We ware shipping container logs to side car. But what about the host logs . every server have some log files go to any host machine logs /var/log/ here lot of logs available administration purpose analyzing these logs also very important .something called daemonset.

**15Q) what is the deference between daemon set and deployment?**

Ans) If you run daemonset then k8s make sure it run pod each and every node of the cluster.

But deployment is not like that deployment used for the application deployment.

Daemonset used to underlying host log file to external system and daemonset we don’t use it for the application deployment. just use it for the administrative activity.

**16Q) storages in k8s?**

Ans) EBS 🡪 Elastic block storage 🡪 zone level storage

EFS 🡪 Elastic file system 🡪 it can be access anywhere from the internet.

We have 3 options in k8s:

Storageclass

Persistent volume

Persistent volume claim

We have two types of storage options

* Static provisioning – in static provisioning we have to create the disk and attach. Means admin create the disk. Your admin team create pv. thy will inform you. Now you can create pvc . then start using it.
* Dynamic provisioning – dynamic provisioning is nothing but k8 creates itself. You no need to any thing automatically create k8s.

17Q) **Kubernetes images and container errors?**

Kubernetes is a powerful container orchestration platform that allows you to manage and deploy containerized applications at scale. When working with Kubernetes, you may encounter errors related to images and containers. Here are some common issues and their potential solutions:

**1. ImagePullBackOff:**

* **Error Message:** Pods remain in a pending state with an "ImagePullBackOff" error.
* **Cause:** Kubernetes cannot pull the specified container image.
* **Solution:**
  + Check the image name and tag for typos.
  + Ensure the image repository is accessible and the image exists.
  + Verify image credentials if the image repository requires authentication.
  + Check if there are network issues preventing image download.

**2. CrashLoopBackOff:**

* **Error Message:** Pods repeatedly start and crash, entering a "CrashLoopBackOff" state.
* **Cause:** The container encounters an issue that causes it to crash.
* **Solution:**
  + Check the logs of the container using **kubectl logs <pod-name>**.
  + Fix any application-level issues causing the crash.
  + Ensure that required environment variables and configurations are correctly set.
  + Check resource limits (CPU, memory) to ensure the container has enough resources.

**3. Insufficient Resources:**

* **Error Message:** Pods may fail to start due to insufficient resources.
* **Cause:** The pod's resource requests exceed the available resources in the cluster.
* **Solution:**
  + Adjust the resource requests and limits in the pod's YAML configuration.
  + If necessary, scale your cluster or allocate more resources to nodes.

**4. ImageVersionMismatch:**

* **Error Message:** Applications may behave unexpectedly if they rely on specific image versions.
* **Cause:** The deployed application may not be compatible with the used image version.
* **Solution:**
  + Ensure that the image tag/version specified in the deployment matches the expected version.
  + Implement a versioning strategy to control image updates and rollbacks.

**5. Container Runtime Errors:**

* **Error Message:** Errors related to the container runtime, like "ContainerCreating," may occur.
* **Cause:** Issues with the container runtime (e.g., Docker) on cluster nodes.
* **Solution:**
  + Check the status of the container runtime on nodes (**docker ps**, **containerd status**, etc.).
  + Restart the container runtime if necessary.
  + Monitor the system logs for runtime-specific errors.

**6. Resource Exhaustion:**

* **Error Message:** Pods or nodes may become unresponsive due to resource exhaustion.
* **Cause:** Heavy workloads or misconfigurations can lead to resource depletion.
* **Solution:**
  + Implement resource quotas and limits for pods to prevent resource hogging.
  + Monitor cluster resource usage and scale the cluster as needed.

**7. Network Issues:**

* **Error Message:** Communication problems between containers or services.
* **Cause:** Network policies, DNS resolution, or firewall rules may be misconfigured.
* **Solution:**
  + Check network policies and firewall rules.
  + Ensure that DNS resolution is working correctly in your cluster.
  + Test connectivity between pods and services.

Troubleshooting Kubernetes errors related to images and containers requires a deep understanding of your application, cluster configuration, and the Kubernetes ecosystem. Be prepared to examine logs, monitor resource utilization, and iterate on your configuration to resolve issues effectively.

Certainly, let's address the "ImagePullError" and "ContainerCreating" issues in Kubernetes in the format you requested:

**1. ImagePullError:**

* **Error Message:** Pods remain in a pending state with an "ImagePullError" error.
* **Cause:** Kubernetes cannot pull the specified container image.
* **Solution:**
  + Check the image name and tag for typos.
  + Ensure the image repository is accessible and the image exists.
  + Verify image credentials if the image repository requires authentication.
  + Check if there are network issues preventing image download.

**2. ContainerCreating:**

* **Error Message:** Pods are stuck in a "ContainerCreating" state.
* **Cause:** There are various reasons why pods might get stuck in this state. Here are some potential causes and solutions:
  + **Resource Constraints:**
    - **Cause:** The node lacks sufficient resources (CPU, memory) to create the container.
    - **Solution:** Adjust resource requests and limits in the pod's YAML configuration. Ensure that the node has enough resources.
  + **ImagePullBackOff:**
    - **Cause:** The container image specified cannot be pulled, leading to a failed container creation.
    - **Solution:** Refer to the "ImagePullError" section above for solutions.
  + **Network Issues:**
    - **Cause:** Problems with networking may prevent the container from starting.
    - **Solution:** Check network configurations, DNS resolution, and firewall rules. Ensure network connectivity within the cluster.
  + **Pod Configuration Errors:**
    - **Cause:** There might be errors in the pod's configuration, such as incorrect volume mounts or missing environment variables.
    - **Solution:** Review the pod's YAML configuration carefully and correct any errors.
  + **Node Issues:**
    - **Cause:** The node itself might have problems, such as insufficient disk space or other system issues.
    - **Solution:** Check the node's health and resource availability. Troubleshoot node-level issues if necessary.

Remember that the "ContainerCreating" state is a broad category that can have various causes, and the solution depends on the specific circumstances of your deployment. Examining pod logs (**kubectl logs <pod-name>**) and events (**kubectl describe pod <pod-name>**) can provide more detailed information about the problem.

18Q) **Kubernetes pv and pvc and Storageclass errors above format?**

Certainly, let's discuss Kubernetes Persistent Volumes (PV), Persistent Volume Claims (PVC), and StorageClass errors in the format you requested:

**1. PV (Persistent Volume) Errors:**

* **Error Message:** Issues related to Kubernetes Persistent Volumes.
* **Cause:** Various problems can occur with PVs, including:
  + PV not found when claimed.
  + PV capacity exhausted.
  + Incorrect access modes.
  + PV reclaim policy issues.
* **Solution:**
  + Check if the PV exists (**kubectl get pv**) and its status.
  + Ensure the PV capacity meets your PVC requirements.
  + Verify that the access modes (e.g., ReadWriteOnce, ReadOnlyMany) match your PVC.
  + Review the reclaim policy (e.g., Retain, Recycle, Delete) and ensure it aligns with your needs.
  + Troubleshoot and resolve any issues with the underlying storage provider.

**2. PVC (Persistent Volume Claim) Errors:**

* **Error Message:** Problems with Kubernetes Persistent Volume Claims.
* **Cause:** Issues with PVCs can include:
  + PVC not binding to a PV.
  + Insufficient storage resources.
  + Requested StorageClass not found.
* **Solution:**
  + Check if the PVC is bound to a PV (**kubectl get pvc**).
  + Ensure that the requested storage resources in the PVC meet the available PV's capacity.
  + Verify that the StorageClass specified in the PVC exists and is accessible.
  + Check if the PVC is in the correct namespace.
  + Investigate and resolve any issues preventing binding.

**3. StorageClass Errors:**

* **Error Message:** Errors related to Kubernetes Storage Classes.
* **Cause:** Problems with Storage Classes can involve:
  + StorageClass not found.
  + Incorrect provisioner or parameters.
  + Misconfigured default StorageClass.
* **Solution:**
  + Confirm that the StorageClass specified in the PVC exists (**kubectl get storageclass**).
  + Review the provisioner type and parameters to ensure they match your storage solution.
  + Check if a default StorageClass is correctly set and that it meets your requirements.
  + Inspect the StorageClass events for more details on errors.

When troubleshooting issues with PVs, PVCs, and Storage Classes, it's essential to examine Kubernetes events, logs, and describe commands (**kubectl describe pv**, **kubectl describe pvc**, **kubectl describe storageclass**) for specific error messages and details. Additionally, make sure that your cluster's storage infrastructure is properly configured and accessible.

19Q) \*\*1. **What is Storage in Kubernetes?**

* **Abode Explanation:** Storage in Kubernetes refers to how data is stored, accessed, and managed within the Kubernetes container orchestration platform. It enables applications to persist data even if containers or pods are restarted or moved.

\*\*2. **Persistent Volumes (PVs) - Storage Resources**

* **Abode Explanation:** PVs are like predefined storage units or rooms in a building. They provide a way to allocate and manage storage resources independently of applications.

\*\*3. **Persistent Volume Claims (PVCs) - Requesting Storage**

* **Abode Explanation:** PVCs are like reservation requests for storage. Applications or pods can request a certain amount of storage (like reserving a room) using PVCs.

\*\*4. **StorageClasses - Storage Options**

* **Abode Explanation:** StorageClasses are like catalogs of available storage types. They define the characteristics of storage, such as performance (fast or slow), access modes (read/write permissions), and provisioning mechanisms.

\*\*5. **Binding PVCs to PVs - Making the Connection**

* **Abode Explanation:** When a PVC is created, Kubernetes tries to find a matching PV based on size, access mode, and StorageClass. If a suitable PV is available, it binds the PVC to that PV, creating a connection.

\*\*6. **Volumes in Pods - Storage Integration**

* **Abode Explanation:** To use the allocated storage, pods need to integrate it by specifying it as a volume. This is like connecting a storage room to a specific apartment in the building.

\*\*7. **Storage Lifecycle - Using and Releasing**

* **Abode Explanation:** PVs and PVCs have lifecycles. Once a PVC is bound to a PV, the storage is accessible for reading and writing. When it's no longer needed, you can release the claim, allowing Kubernetes to recycle the storage for other use.

\*\*8. **Dynamic Provisioning - On-Demand Storage**

* **Abode Explanation:** Some StorageClasses support dynamic provisioning, which means Kubernetes can automatically create PVs when PVCs are requested. It's like having rooms magically appear when you need them.

\*\*9. **Storage for Stateful Applications**

* **Abode Explanation:** Storage is crucial for stateful applications like databases, ensuring that data persists across pod or node restarts. PVs and PVCs are essential to maintain data integrity.

\*\*10. **Troubleshooting Storage Issues** - **Abode Explanation:** When encountering storage-related problems, you can use **kubectl** commands to inspect, describe, and troubleshoot PVs, PVCs, and StorageClasses. Understanding their configurations is key to resolving issues.

Think of Kubernetes storage as a versatile building with rooms (PVs) available for rent. You request a room (PVC) with specific attributes from a catalog (StorageClass). Once you have your room, you can set it up inside your apartment (pod) to store and retrieve your belongings. Properly managing these storage components ensures your data is safe and accessible in Kubernetes.