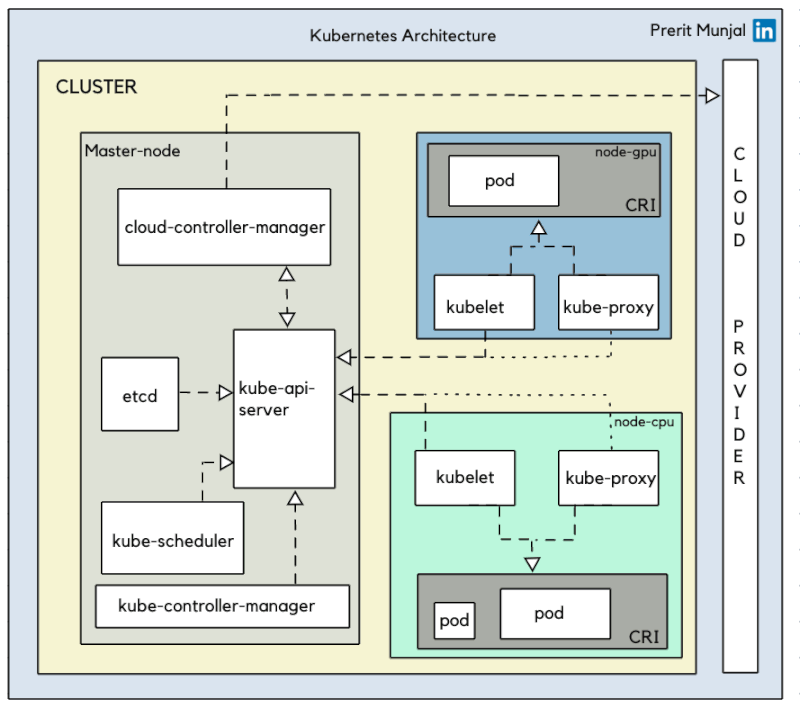
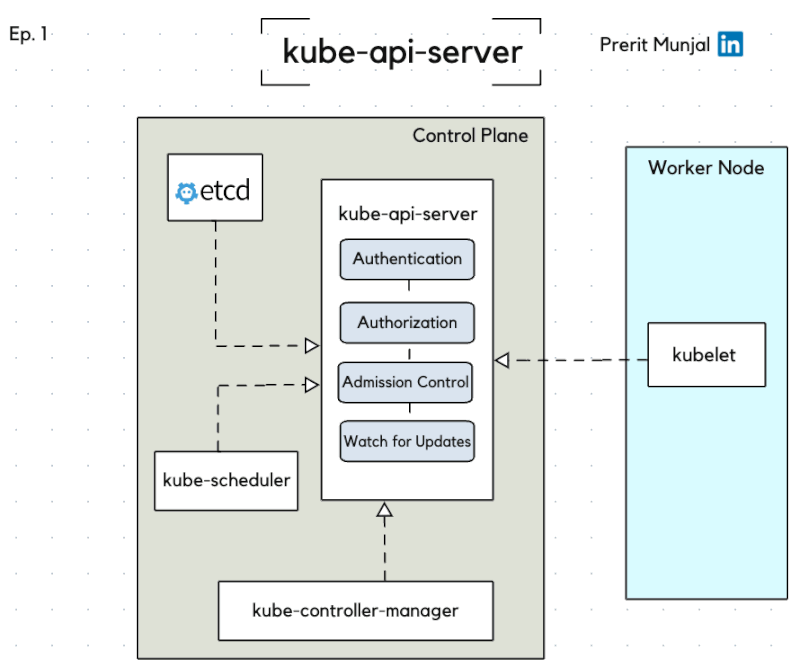
**Interview**

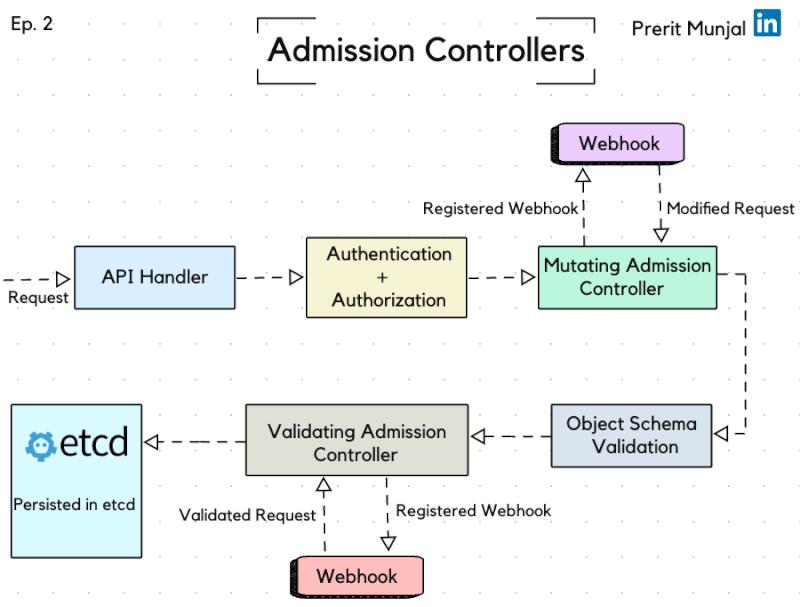
Engineers choking on the Kubernetes Architecture, so here it goes!!

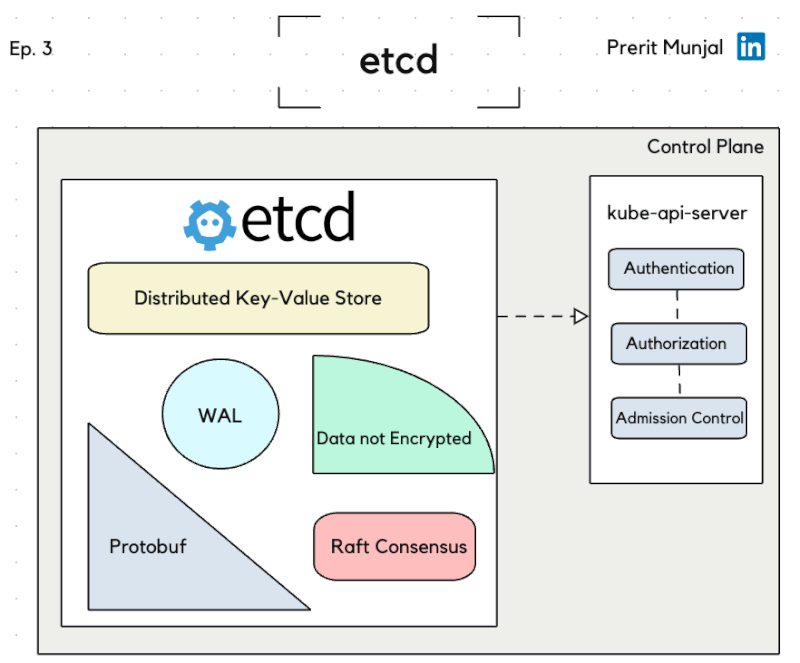
1. 𝗰𝗹𝗼𝘂𝗱-𝗰𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿-𝗺𝗮𝗻𝗮𝗴𝗲𝗿: Links your cluster with the cloud provider API which enables cloud providers to release features at a different pace as compared to the main k8s project. It is also responsible for Node, Route and Service Controller.  
  
2. 𝗲𝘁𝗰𝗱: It is a Multiversion Consistent Immutable Key-Value store for storing cluster data(config, state, metadata) which uses gRPC for its messaging protocol.   
  
Only the API server has the privileges to connect to etcd, the rest of the components need to go through the API server for any state retrieval.  
  
3. 𝗸𝘂𝗯𝗲-𝗮𝗽𝗶-𝘀𝗲𝗿𝘃𝗲𝗿: This is the laziest component of k8s. The main function is to process the API calls. It is designed to scale horizontally.  
  
4. 𝗸𝘂𝗯𝗲-𝘀𝗰𝗵𝗲𝗱𝘂𝗹𝗲𝗿: It watches for schedulable resources and binds them to nodes based on the resource availability by taking multiple factors into account like taints, tolerations, and node affinity.  
  
5. 𝗸𝘂𝗯𝗲-𝗰𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿-𝗺𝗮𝗻𝗮𝗴𝗲𝗿: It implements the core control loops through which it watches for any changes and does the necessary changes. Takes care of garbage collection, and namespace life cycle.  
  
6. 𝗸𝘂𝗯𝗲𝗹𝗲𝘁: Makes sure that the containers are running in a healthy state and manages the node’s resources.  
  
7. 𝗸𝘂𝗯𝗲-𝗽𝗿𝗼𝘅𝘆: It is responsible for all the underlying network configuration on nodes. Example: Service to Pod IP Translation.  
  
8. 𝗖𝗥𝗜: Container Runtime Interface manages the life cycle of containers/images and enables the working of containers on k8s.  
  
9. 𝗣𝗼𝗱𝘀: They are a bunch of processes running till completion. I call them FTC(Functions till completion)  
  
[Cloud Native Computing Foundation (CNCF)](https://www.linkedin.com/company/cloud-native-computing-foundation/)



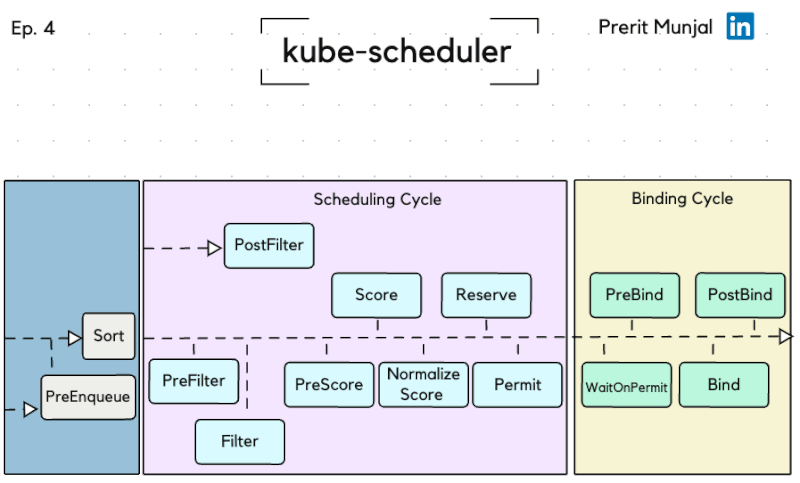
𝗪𝗵𝘆 𝗶𝘀 𝘁𝗵𝗲 𝗸𝘂𝗯𝗲-𝗮𝗽𝗶-𝘀𝗲𝗿𝘃𝗲𝗿 𝘀𝗼 𝗹𝗮𝘇𝘆?  
  
It is an HTTP server storing the states in a database(etcd) and exposes the Kubernetes API. That’s it!!  
  
1. 𝗟𝗼𝗴𝘀 𝗲𝘃𝗲𝗿𝘆𝘁𝗵𝗶𝗻𝗴: As it serves as a middle ground for all the components of a cluster to meet and fetch the request, it logs everything in Audit and Basic Logs.   
  
2. 𝗠𝗮𝗻𝗮𝗴𝗲𝘀 𝗔𝗣𝗜 𝗰𝗮𝗹𝗹𝘀: The entire k8s is built around APIs, so its role is mostly occupied with API discovery and API translation.  
  
3. 𝗔𝘂𝘁𝗵𝗲𝗻𝘁𝗶𝗰𝗮𝘁𝗶𝗼𝗻: As soon as a request/API call reaches the server, it checks for the authenticity of the request by using client certs and tokens.   
  
4. 𝗔𝘂𝘁𝗵𝗼𝗿𝗶𝘇𝗮𝘁𝗶𝗼𝗻: Once the identity is determined, it performs the Authorization using RBAC(Role-based access control)/Webhook/Node/ABAC(Attribute-based access control).  
  
5. 𝗔𝗱𝗺𝗶𝘀𝘀𝗶𝗼𝗻 𝗖𝗼𝗻𝘁𝗿𝗼𝗹: Once the request is Authenticated and Authorized it moves on to the admission controllers(there are many). These controllers check for the changes/actions the request will perform once validated.   
All the admission controllers are called serially each receiving the output of the previous one. If any admission controller finds an error, the request is rejected.  
   
6. 𝗦𝘂𝗽𝗽𝗼𝗿𝘁𝘀 𝘁𝗵𝗲 𝘄𝗮𝘁𝗰𝗵 𝗔𝗣𝗜: Instead of polling the server frequently for updates, it uses the low-latency updates known as Watch.  


𝗔𝗱𝗺𝗶𝘀𝘀𝗶𝗼𝗻 𝗖𝗼𝗻𝘁𝗿𝗼𝗹 𝗶𝘀 𝗷𝘂𝘀𝘁 𝗹𝗶𝗸𝗲 𝘁𝗵𝗲 𝘀𝗰𝗵𝗼𝗼𝗹 𝗮𝗱𝗺𝗶𝘀𝘀𝗶𝗼𝗻 𝗽𝗿𝗼𝗰𝗲𝘀𝘀!  
  
Once the request/API Call is authenticated and authorized, it is sent to the Admission Controller which intercepts the requests before making any solid change in the cluster.   
  
The entire process happens in 2 phases:   
  
1. 𝗠𝘂𝘁𝗮𝘁𝗶𝗻𝗴: Performs changes on objects if specified. Like 𝗗𝗲𝗳𝗮𝘂𝗹𝘁𝗜𝗻𝗴𝗿𝗲𝘀𝘀𝗖𝗹𝗮𝘀𝘀 which will add an ingress class to the Ingress objects.   
  
2. 𝗩𝗮𝗹𝗶𝗱𝗮𝘁𝗶𝗻𝗴: Validates the request after mutation. Like 𝗡𝗮𝗺𝗲𝘀𝗽𝗮𝗰𝗲𝗘𝘅𝗶𝘀𝘁𝘀 which will check if the namespace exists or not.  
  
First, the mutating admission controllers are run followed by the validating admission controllers. If any of them rejects the request, the error is reported to the API server and an error is returned to the end user.  
  
Some controllers can be both mutating and validating like:  
  
𝗟𝗶𝗺𝗶𝘁𝗥𝗮𝗻𝗴𝗲𝗿: This will observe the incoming requests and make sure that it does not violate any limit range values defined in the namespace and set the default value to the objects.  
  
In case you want to extend the admission controllers you can use 𝗠𝘂𝘁𝗮𝘁𝗶𝗻𝗴𝗔𝗱𝗺𝗶𝘀𝘀𝗶𝗼𝗻𝗪𝗲𝗯𝗵𝗼𝗼𝗸 and 𝗩𝗮𝗹𝗶𝗱𝗮𝘁𝗶𝗻𝗴𝗔𝗱𝗺𝗶𝘀𝘀𝗶𝗼𝗻𝗪𝗲𝗯𝗵𝗼𝗼𝗸  
  
Once this request is validated the request, the change is made in etcd and the related API Calls are made to the relevant components.  
  
Some practices use case of these controllers are:  
  
1. Blocking the use of public registries.  
2. Restricting actions like get, and list on objects.  
3. Enforcing security policies on pod.

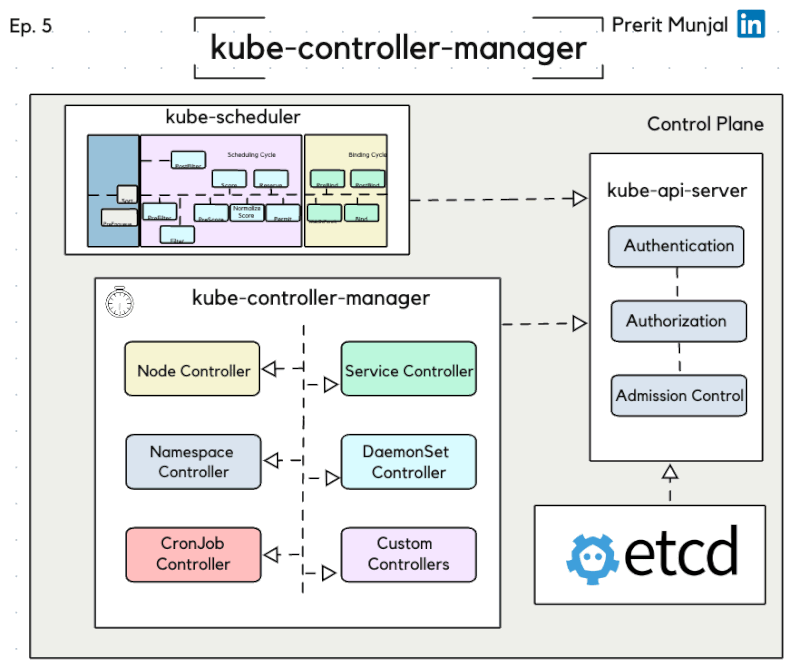
  
𝗗𝗼𝗲𝘀 𝘁𝗵𝗲 𝗻𝗮𝗺𝗲 𝗲𝘁𝗰𝗱 𝗺𝗮𝗸𝗲 𝗮𝗻𝘆 𝘀𝗲𝗻𝘀𝗲?  
  
etcd is not just some ordinary database, it is a distributed, highly consistent key-value store and the only stateful component of the entire k8s components.  
  
The name etcd came from 𝗲𝘁𝗰 which is the location of system configuration files in Linux and 𝗱 which stands for distributed.  
  
1. 𝗕𝘂𝗶𝗹𝘁 𝗼𝗻 𝗥𝗮𝗳𝘁 𝗖𝗼𝗻𝘀𝗲𝗻𝘀𝘂𝘀(out of this post's scope) but on a very high level, all the nodes hold elections. The one with the majority becomes a leader and others become followers.  
First, the Leader receives the data and then replicates it across all the followers. Once all the nodes have the same data, the Log is committed. This is how 𝗟𝗼𝗴 𝗥𝗲𝗽𝗹𝗶𝗰𝗮𝘁𝗶𝗼𝗻 works in etcd.  
  
2. Uses 𝗣𝗿𝗼𝘁𝗼𝗯𝘂𝗳 to deserialise/serialise the data faster.  
  
3. By default, data stored in etcd is 𝗻𝗼𝘁 𝗲𝗻𝗰𝗿𝘆𝗽𝘁𝗲𝗱(but can be encrypted).  
  
4. 𝗦𝘁𝗼𝗿𝗲𝘀 𝗮𝗹𝗹 𝘁𝗵𝗲 𝗱𝗮𝘁𝗮(current state, cluster info, desired state, runtime data).  
  
5. Clubs together a couple of entries and sends to the engine to attain high throughput despite the heavy load, resulting in less 𝗖𝗣𝗨 𝗽𝗿𝗲𝘀𝘀𝘂𝗿𝗲 𝗮𝗻𝗱 𝗹𝗮𝘁𝗲𝗻𝗰𝘆.  
  
6. Uses 𝗪𝗔𝗟 (𝗪𝗿𝗶𝘁𝗲 𝗔𝗵𝗲𝗮𝗱 𝗟𝗼𝗴) which means operations are logged before they happen to ensure high consistency and data integrity.  
  
7. Data is stored in a 𝗺𝘂𝗹𝘁𝗶-𝘃𝗲𝗿𝘀𝗶𝗼𝗻 𝗸𝗲𝘆-𝘃𝗮𝗹𝘂𝗲 𝗱𝗮𝘁𝗮 𝗺𝗼𝗱𝗲𝗹. On a very high level, keys store some value and their subsequent revisions/ versions.  
  
Yes, there are other alternatives like Consul but etcd is every battle tested :)



𝗦𝗰𝗵𝗲𝗱𝘂𝗹𝗶𝗻𝗴 𝗶𝘀 𝗻𝗼𝘁 𝘀𝗼 𝗲𝗮𝘀𝘆 𝗶𝗻 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀!!  
  
𝗸𝘂𝗯𝗲-𝘀𝗰𝗵𝗲𝗱𝘂𝗹𝗲𝗿 is the Control Plane component that manages the scheduling and distribution of Pods on the worker nodes.  
  
The entire scheduling consists of 2 phases:  
𝗦𝗰𝗵𝗲𝗱𝘂𝗹𝗶𝗻𝗴 𝗖𝘆𝗰𝗹𝗲 which finds a feasible node and the 𝗕𝗶𝗻𝗱𝗶𝗻𝗴 𝗖𝘆𝗰𝗹𝗲 which binds a pod to that node.  
  
Each cycle has various 𝗘𝘅𝘁𝗲𝗻𝘀𝗶𝗼𝗻 𝗣𝗼𝗶𝗻𝘁𝘀 𝗼𝗿 𝗣𝗹𝘂𝗴𝗶𝗻𝘀 that keep the scheduler lightweight and efficient:  
  
1. 𝗣𝗿𝗲𝗘𝗻𝗾𝘂𝗲𝘂𝗲: It performs all the necessary checks and segregates pod in the 𝗔𝗰𝘁𝗶𝘃𝗲𝗤, 𝗨𝗻𝘀𝗰𝗵𝗲𝗱𝘂𝗹𝗲𝗮𝗯𝗹𝗲𝗤 or 𝗣𝗼𝗱𝗕𝗮𝗰𝗸𝗼𝗳𝗳𝗤.  
  
2. 𝗦𝗼𝗿𝘁: Sort the pods in the queue.  
  
3. 𝗣𝗿𝗲𝗙𝗶𝗹𝘁𝗲𝗿: Pre-processes the info about the pod, and checks all the preconditions like taints, node selectors.  
  
4. 𝗙𝗶𝗹𝘁𝗲𝗿: Filter out nodes that are not suitable for the pod.  
  
5. 𝗣𝗼𝘀𝘁𝗙𝗶𝗹𝘁𝗲𝗿: This is only run if no feasible node is found, the only function is to kick out some of the lower priority pods and schedule the requested one.  
  
6. 𝗣𝗿𝗲𝗦𝗰𝗼𝗿𝗲: Fetches Information for the scoring tasks.  
  
7. 𝗦𝗰𝗼𝗿𝗲: Rank the nodes that have passed the filtering phase.  
  
8. 𝗡𝗼𝗿𝗺𝗮𝗹𝗶𝘇𝗲𝗦𝗰𝗼𝗿𝗲: Modifies the final score if required and returns an integer value.  
  
9. 𝗥𝗲𝘀𝗲𝗿𝘃𝗲: Once the nodes are ranked, the Node is reserved for a Pod.  
  
10. 𝗣𝗲𝗿𝗺𝗶𝘁: This plugin consists of 3 phases.  
𝗔𝗽𝗽𝗿𝗼𝘃𝗲 which will give a green flag for the binding cycle to happen, 𝗗𝗲𝗻𝘆 which will return the pod to the scheduling queue and 𝗪𝗮𝗶𝘁 which will wait until a plugin approves it.  
  
11. 𝗣𝗿𝗲𝗕𝗶𝗻𝗱: Once the pod is approved, the pod is sent to this plugin to perform the pre-paperwork like creating a network volume and binding it.  
  
12. 𝗕𝗶𝗻𝗱: This plugin performs the heavy work of finally binding a pod with the node.  
  
13. 𝗣𝗼𝘀𝘁𝗕𝗶𝗻𝗱: This extension point marks the end of the binding cycle and declares the result.  
  
If any Binding Cycle Plugin rejects the pod, it is again sent to the scheduling queue.  
  
Scheduling cycles are run serially while  
Binding cycles may run concurrently



𝗔𝗻𝘆 𝗽𝗼𝗱 𝗰𝗮𝗻 𝗴𝗼 𝗥𝗼𝗴𝘂𝗲, 𝗤𝘂𝗶𝘁 𝗼𝗿 𝗷𝘂𝘀𝘁 𝗸𝗶𝗹𝗹 𝗶𝘁𝘀𝗲𝗹𝗳 𝗱𝘂𝗿𝗶𝗻𝗴 𝘁𝗵𝗲 𝗲𝗻𝘁𝗶𝗿𝗲 𝗰𝘆𝗰𝗹𝗲!!  
  
Once the request is:  
1. Authenticated & Authorized.  
2. Passed by the Admission Controllers.  
3. Persisted in etcd.  
4. Got the green flag from all the plugins in the kube-scheduler.  
  
Comes the part of kube-controller-manager.  
  
𝗢𝗯𝘀𝗲𝗿𝘃𝗲𝗿-𝗖𝗼𝗺𝗽𝗮𝗿𝗲-𝗔𝗰𝘁𝗶𝗼𝗻 𝗶𝗳 𝗿𝗲𝗾𝘂𝗶𝗿𝗲𝗱) - 𝗥𝗲𝗽𝗲𝗮𝘁 (𝘁𝗶𝗹𝗹 𝗶𝗻𝗳𝗶𝗻𝗶𝘁𝘆).  
  
It's a Daemon polling the API Server at regular intervals to make sure that the 𝗗𝗲𝘀𝗶𝗿𝗲𝗱 𝘀𝘁𝗮𝘁𝗲(.spec) and the 𝗖𝘂𝗿𝗿𝗲𝗻𝘁 𝘀𝘁𝗮𝘁𝗲(.status) are the same.   
  
Some of the built-in Kubernetes controllers are:  
  
1. 𝗡𝗼𝗱𝗲 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿: Responsible for managing the worker nodes based on the metrics passed by the kubelet.  
  
2. 𝗦𝗲𝗿𝘃𝗶𝗰𝗲 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿: Responsible for configuring and managing all the services.  
  
On a very high level, if the object is found unhealthy it is evicted and replaced by a healthy one. That’s it!!  
  
How does it work internally?  
  
1. 𝗦𝗵𝗮𝗿𝗲𝗱𝗜𝗻𝗳𝗼𝗿𝗺𝗲𝗿: As the role of multiple controllers can overlap, it may happen that some of the controllers are polling the API server continuously for the same objects. So it creates one watch state to the Upstream API server which will be consumed by multiple consumers to reduce the latency and CPU pressure.  
  
2. 𝗪𝗼𝗿𝗸𝗤𝘂𝗲𝘂𝗲: Just like the name its role is to send the tasks to a queue which will be picked and processed once the previous one is executed.



𝗕𝗲 𝗚𝗼𝗼𝗱 𝗶𝗻 𝗖𝗹𝗼𝘂𝗱 𝗙𝘂𝗻𝗱𝗮𝗺𝗲𝗻𝘁𝗮𝗹𝘀!!  
  
But how to check if you are good at it or not??   
  
If you can answer 4-5 of the below questions, then Yes you are good at it.  
  
1. One of our VM is running out of RAM, and the CPU utilisation has crossed more than 95%, what will you do to 𝗺𝗶𝗻𝗶𝗺𝗶𝘀𝗲 𝘁𝗵𝗲 𝗱𝗶𝘀𝗿𝘂𝗽𝘁𝗶𝗼𝗻 𝗮𝗻𝗱 𝘁𝗵𝗿𝗼𝘁𝘁𝗹𝗲?  
  
2. One of our stateless workloads running on a pod is 𝗿𝗲𝗯𝗼𝗼𝘁𝗶𝗻𝗴 𝗮𝘁 𝗲𝘅𝗮𝗰𝘁𝗹𝘆 𝗲𝘃𝗲𝗿𝘆 𝟲 𝗵𝗼𝘂𝗿𝘀, what would be the possible root cause?  
  
3. We are modernising a Recycled-Plastic Warehouse, by installing multiple smoke, and IR sensors which would result in high ingestion of NoSQL data, can you design an architecture for 𝗱𝗮𝘁𝗮 𝘄𝗮𝗿𝗲𝗵𝗼𝘂𝘀𝗶𝗻𝗴 𝗮𝗻𝗱 𝗶𝗺𝗺𝗲𝗱𝗶𝗮𝘁𝗲 𝗱𝗲𝗹𝗶𝘃𝗲𝗿𝘆 𝗼𝗳 𝗻𝗼𝘁𝗶𝗳𝗶𝗰𝗮𝘁𝗶𝗼𝗻𝘀 in case of any mishap?  
  
4. 𝗖𝗮𝗹𝗰𝘂𝗹𝗮𝘁𝗲 𝘁𝗵𝗲 𝗽𝗿𝗶𝗰𝗶𝗻𝗴 𝗳𝗼𝗿 𝗮𝗽𝗽𝗹𝗶𝗰𝗮𝘁𝗶𝗼𝗻 𝗱𝗲𝗽𝗹𝗼𝘆𝗺𝗲𝗻𝘁 𝗳𝗼𝗿 𝗮 𝗺𝗼𝗻𝘁𝗵, there are 2 machines with 2vCPUs, 7.5GB Boot disk, 100GB of external disk, 100k CLASS A and B ops + a MySQL server database with the storage capacity of 1TB and a read replica for HA.  
  
5. In an Event-driven application, subscriber A is 𝗻𝗼𝘁 𝗮𝗯𝗹𝗲 𝘁𝗼 𝗽𝘂𝗹𝗹 𝘁𝗵𝗲 𝗺𝗲𝘀𝘀𝗮𝗴𝗲 from topic B, what would be your initial guess to this problem?  
  
6. 𝗪𝗵𝗮𝘁 𝘀𝗵𝗼𝘂𝗹𝗱 𝘆𝗼𝘂 𝗱𝗼 𝗶𝗻 𝗰𝗮𝘀𝗲 𝗼𝗳 𝗮 𝟰𝟮𝟵 𝗲𝗿𝗿𝗼𝗿? And how will you prevent it in the future?