IT & Software

7 Kubernetes Interview Questions You Should Be Able to Answer

Edward Viaene
Udemy Instructor

Share this article

Interviews that test your extensive Kubernetes knowledge can be intimidating. Although Kubernetes is still fairly young, it has expanded to a massive ecosystem over the last few years.

Knowing what you should learn to be able to answer an interviewer's questions can be especially difficult. The time where interviewers asked "What is Kubernetes?" or "How does Kubernetes compare to Docker Swarm?" are unfortunately long gone. Detailed knowledge about Kubernetes' internals is now the norm.

In this article, I cover some of the harder, deep-dive Kubernetes interview questions that companies may ask you. Learning the answers to the following questions will go a long way to helping you land your next role.

1. What happens when you enter "kubectl get pods"?

Don't underestimate this question. The simple statement, "It lists the pods on the kubernetes cluster," is far from the answer an interviewer wants to hear. Let's go over the steps that are happening when you enter this command, at a high level:

Step 1: Config

When running kubectl, it will first look into the kubernetes configuration file. The default path for the configuration is ~/.kube/config (where ~ is the home directory of the current user).

Make sure you are familiar with this configuration file. The config file can contain multiple clusters, users, and contexts, but there's typically only one context activated (the current context). To see the contexts and which one is current, you can use the command "kubectl config get-contexts."

For someone with real-world kubernetes experience, these commands are for switching from one cluster to another cluster (or for wrapping tools that will do this changing of context for you).

Step 2: Kubernetes API Server

Once the current context is found, the user and server configuration can be retrieved. The server information will then contact the kubernetes API server (running on the kubernetes master), which can be queried to get information, but also to make changes.

It's basically a REST API protected by two-way authentication (called mTLS or "mutual TLS").

Step 3: Authentication

This brings us to the authentication step. First, a TLS handshake will need to take place. To make this happen, you need the Certificate Authority Certificate (CA certificate) which is typically embedded within the kubernetes config (or, there is a reference to the file path).

This certificate is needed to validate the identity of the Kubernetes API server. If someone else attempted a "man in the middle" attack by setting up a fake API server, the verification would fail, because you need the Certificate Authority Key (CA key) to be able to run a Kubernetes server with the same identity.

The CA key is never shared with the clients. The client will have access to only the X.509 CA certificate to be able to validate the identity of the server, not create a server with the same identity.

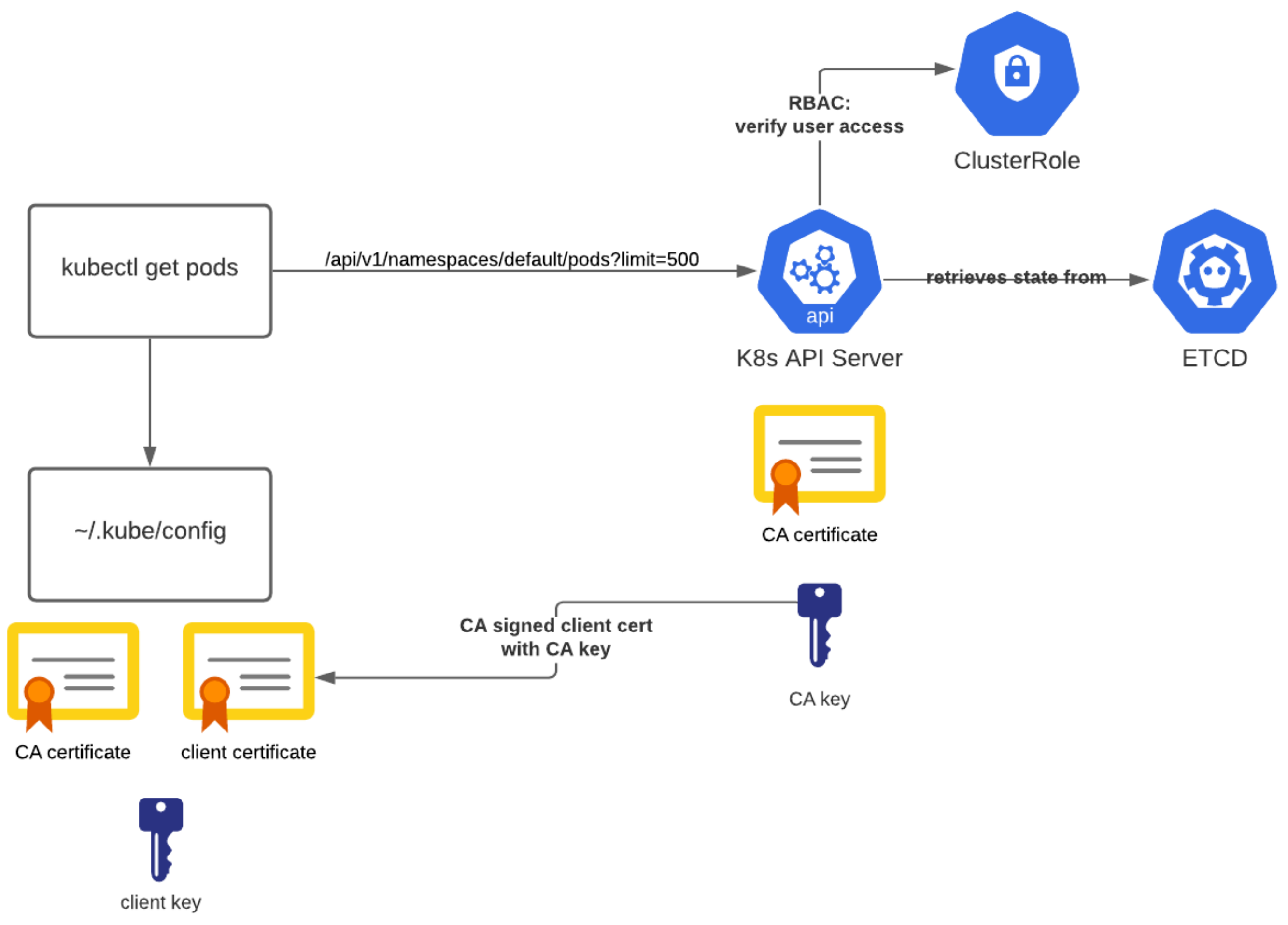
Once kubectl can verify the identity of the Kubernetes API server, it will perform the authentication. For this, we need another X.509 certificate (called the "client certificate") signed by the Certificate Authority (so we couldn't just make up any cert+key pair), and a "client key."

Step 4: REST Endpoint

To get the pods from the Kubernetes API server, kubectl will hit the REST endpoint with the following URL path: /api/v1/namespaces/default/pods?limit=500.

This endpoint will list the pods in the "default" namespace. If you have another default namespace configured, then it will use that name instead of the "default" namespace.

Every REST call will use the CA certificate, client key, and client certificate (as explained earlier) to pass the authentication part of the Kubernetes API server. If one of these elements was missing, instead of a HTTP 200 OK code, you'd get a HTTP 403 Permission Denied error.



Step 5: Authorization

Next to authentication, there is also an authorization part. Authorization is controlled by RBAC and is typically turned on by default in any recent Kubernetes cluster. With RBAC, you control the access of an entity (a user, group, or service account) to specific resources.

For example, let's say you're executing "kubectl get pods" with a user. The user needs to be able to do an HTTP GET request on /api/v1/namespaces/default/pods. You'll need either a ClusterRole (cluster level) or a Role (namespace level) for that. If your user has the predefined cluster-admin role, you'll have access to read/modify any resource.

DevOps students also learn

Docker Kubernetes Jenkins Continuous Integration Docker Containers Terraform

Developer Tools Ansible Automation Amazon AWS Microsoft Azure DevOps Microsoft Azure

Cloud Computing AWS Certification

Deeper dive: additional information

If you outline the above steps in an interview, this would already be a great answer, but there is even more going on than described above. To understand exactly what's going on, you can add the "-v=number" parameter to the kubectl command for log level verbosity.

Adding -v=10 for example will give you trace level logs, showing all the API calls that are being made to the Kubernetes API server.

When enabling tracing, you can see that there is first a call to /api, to discover the possible APIs of the Kubernetes API server. These results are cached in ~/.kube/cache, so you won't see these calls happening every single time.

The APIs of one Kubernetes cluster are not necessarily equal to the APIs provided in another cluster. The Kubernetes API server can be extended with your own APIs, or with APIs of third parties, to add more functionality to Kubernetes. Custom Resource Definitions, which are often used by third parties, will also extend the Kubernetes API.

2. Follow-up question: What protocol is used to communicate with the Kubernetes API Server?

The Kube API Server is using REST, so typically, JSON is for communicating between Kubernetes clients (such as kubectl) and the Kube API Server.

3. Follow-up question: How would you write a Kubernetes client?

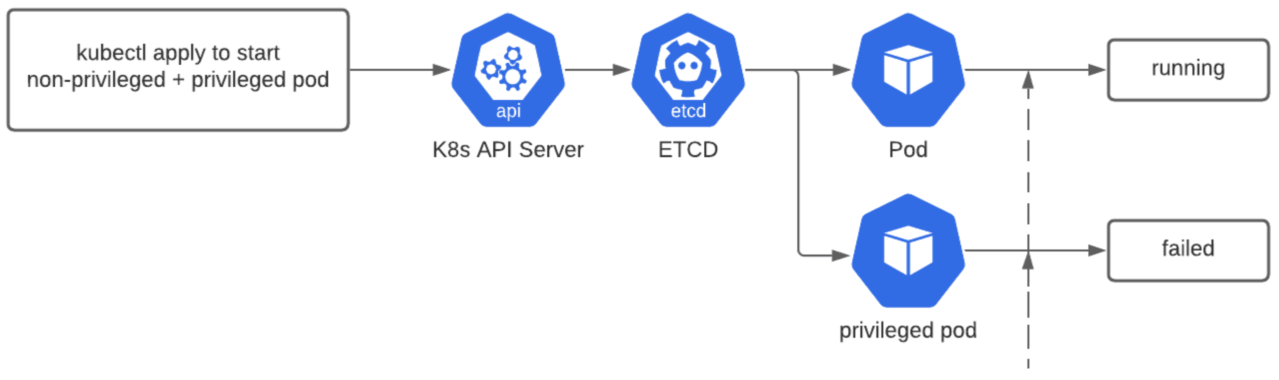
To provide some context, you can write a client to bundle with your CI/CD pipeline to do deployments or to make small changes to your Kubernetes environment. You also are going to want to write a client if you're going to deal with Kubernetes Operators.

To write a Kubernetes client, you need to be able to do REST calls. Any programming language can do REST calls, but it can be cumbersome to write all the calls out manually. That's why Kubernetes maintains a client SDK for the most popular programming languages. You're going to want to use this SDK.

When implementing the SDK, you'll need to handle authentication. The easiest way to do this is by providing a kubectl config file, just like you would when using kubectl. If you're authenticating against a cloud-hosted Kubernetes cluster, the flow might be different, so make sure to check the documentation of your cloud provider.

4. What security measures can be taken to ensure pod-level security?

Best practice says to avoid running containers as root and to avoid accessing the host level filesystem — and those policies should be enforced. One way to do this is through PodSecurityPolicies. This resource can control the security aspects on a pod specification level.



5. A lot of deployments of Kubernetes happen on the cloud. What security measures can be taken there?

All major cloud providers now have a hosted Kubernetes product. Cloud deployments are very different from on-premise deployments, with their own weaknesses that you must anticipate.

Launch with a private control plane

The first step is to ensure the cluster is launched with a private control plane, ensuring nobody from the outside internet can connect to your Kubernetes API server. All major cloud providers now have this option (this wasn't the case when it was originally launched, unfortunately).

Control ingress and egress

Once this is done, you want to ensure ingress/egress traffic is also controlled. You typically want only one way in, so the cloud Load Balancer is going to point to your Kubernetes ingress controller. This can be an nginx, but there are plenty of other third-party solutions available like Traefik, Ambassador, Gloo, Kong, Istio (istio ingress gateway), Linkerd, and others.

For egress, it depends a bit on the cloud provider. On one side you have the cloud-provided constructs like security groups and network firewalls, which can help you on a "cloud network level." On the cluster side, you can already do some of the filtering within the Kubernetes cluster using specific CNIs and NetworkPolicies or with egress gateways that you can find in Istio, or other services mesh products.

Enable identity and access management integrations

With cloud providers, you have the cloud provider's identity and access management products. You'll need to make sure that your Kubernetes install nicely ties into this, to ensure that every pod has its own (temporary) credentials to access cloud services.

In AWS you can tie a pod to a service account, which can be linked to an IAM Role using WebTokens. In Azure you have something similar, using aad-pod-identities.

Create an upgrade strategy

You'll also need to come up with a solid upgrade plan, as Kubernetes tend to release quite often. One upgrade strategy that works great on the cloud is to deploy a new Kubernetes cluster every time there's a new release. This is called a blue/green kubernetes deployment. This can only happen easily if you don't save the state of any app/container within the cluster — the state would need to be saved only in external services (which is very doable on cloud providers).

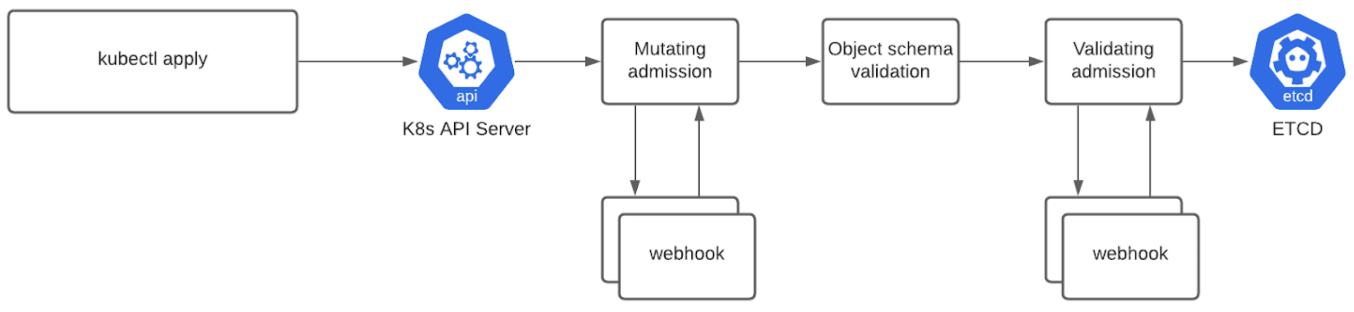
Besides these measures, you'll want to implement all of the typical Kubernetes security controls you'd implement on on-premises clusters, like PodSecurityPolicies.

6. When should you use MutatingWebhooks?

MutatingWebhooks can be used to modify resources when they're sent to the Kubernetes API server.

Courses by Edward Viaene

Let's say you want to deploy an application, so you run a "kubectl apply" using a definition of a deployment object. This deployment object will launch a ReplicaSet that launches the pods (ReplicaSet is the replacement for the older ReplicationController). You could write a MutatingWebhook that injects environment variables into the pod, depending on the annotations the pod provides.

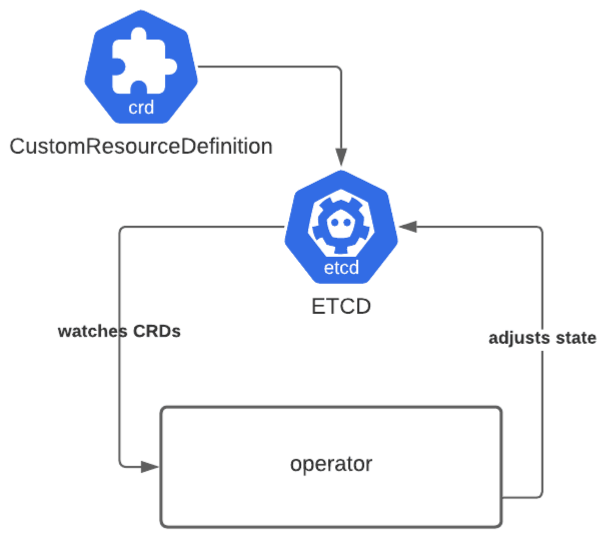


You could make any change you want to a resource definition, at the time that the resource is being created on the cluster. Istio uses MutatingWebhooks, for example, to inject the sidecar to every pod to enable the service mesh.

7. Explain the steps you would take to build an operator on Kubernetes.

The first article of Operators, written in November 2016 by CoreOS, explains that Operators are "a concept to reliably create, configure, and manage complex application instances atop Kubernetes." Operators extend the Kubernetes API to "create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user."

With this in mind, you'll first need to extend the Kubernetes API to create CustomResourceDefinitions. These are custom resources that your operator will look for. Instead of a "Deployment" object, you'll have your own custom object with a name and API that you choose.



You'll now need to write the actual operator itself — either from scratch, or using one of the already-written frameworks that can help you implement this operator pattern.

Once you deploy your new custom object (for example: using kubectl), the operator will be notified and will start to work on your custom implementation (for example: launching a third-party database).

Your operator should also manage this newly created software (for example: restart/recreate pods when the software in the pod malfunctions).

The operator will need to communicate with the Kubernetes API server to listen for changes to your custom objects. It will also need RBAC permissions to do this. You can see that a lot of components of Kubernetes actually interface with the Kubernetes API server, so having a good notion of how the kube-apiserver works is the foundation of understanding Kubernetes.

Read up on Kubernetes to ace your interview

In this article, we've touched on just a few topics (albeit very important ones) of Kubernetes. The above questions will already give you plenty of interesting concepts and ideas that you can explore further, but you should be striving to take your knowledge beyond these questions.

In my Udemy course "The complete Kubernetes Course", you can learn about many Kubernetes concepts in an easy way. Once you have a grasp of the basics, make sure to check out "Advanced Kubernetes Usage" and "On-prem or cloud agnostic Kubernetes".
To fully comprehend all the concepts you will need to have a good look at the official Kubernetes documentation (which is the most authoritative source), to gain a deeper understanding of how Kubernetes really works.

Page Last Updated: December 2020

Edward Viaene
Udemy Instructor

Edward Viaene's Udemy Profile

Edward has been a System Administrator and full stack developer for over 10 years, the typical profile for a DevOps engineer. He has been working in multiple organizations and startups. He cofounded a startup that focuses on applying DevOps and Cloud. Edward has been training people in newer technologies, like Big Data and trained a lot of people working in FTSE 100 & S&P 100 companies. Today he mainly works together with companies to improve their software delivery processes, while coaching and teaching on platforms like
...Read more

Recent Articles by Edward Viaene

- The Best DevOps Tools for Your Application Lifecycle