Kubernetes API Versioning, K8s Extension, Certification Tips & Helm Introduction.

starts at 9:05 pm

AGENDA

1. Admission Controller
    1. Types
2. API Version
3. CRD / CR
4. CKAD Exam tips
    1. Tips
    2. Killer.sh Overview
    3. Sample Questions

```
spec:
 securityContext:
  readOnlyRootFilesystem: true
 containers:
  - name: app
    image: nginx
```

Admission Controller

→ validate

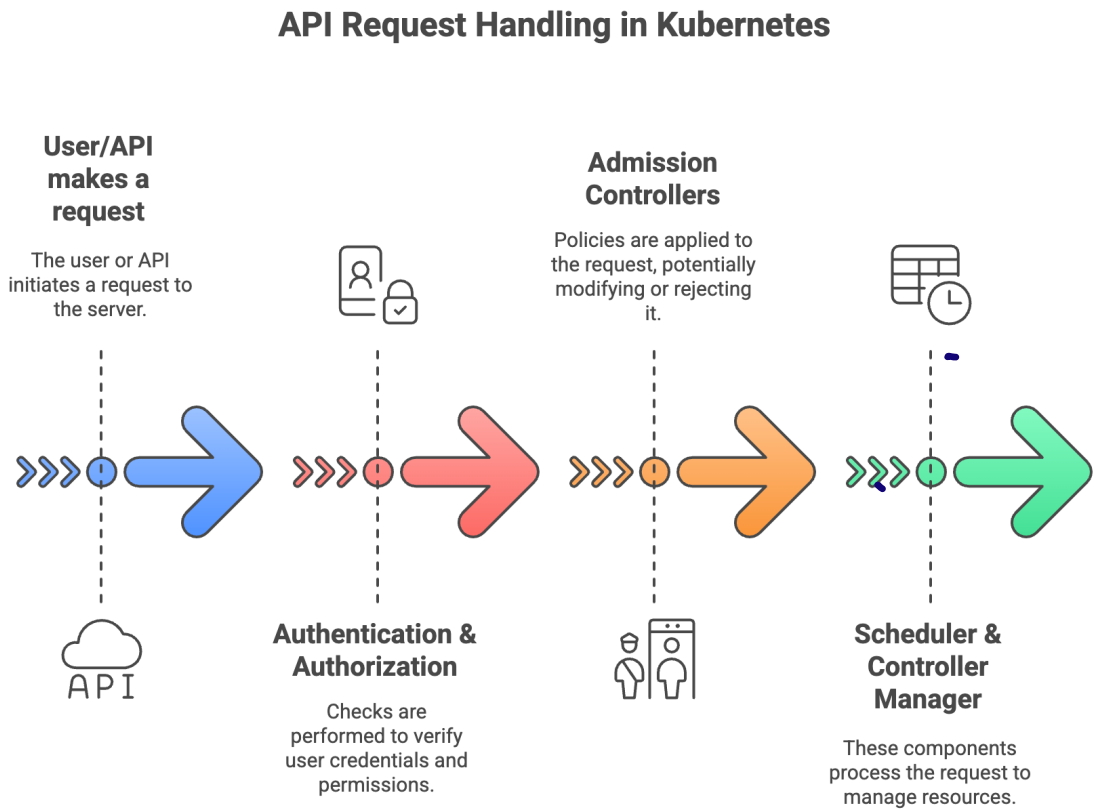
→ modify or

→ reject API request.

→ before getting applied to the cluster.

They intercept requests to the Kubernetes API server before an object is created,

but after the request is authenticated and authorized.



Types of Admission Controllers.

1. Mutating
2. Validating

↓

× can't modify object.

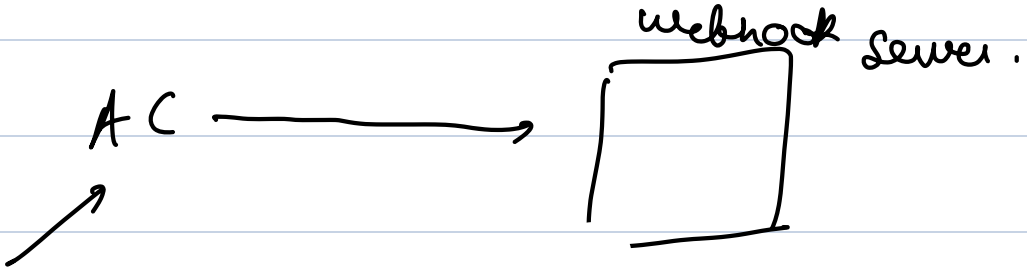
Resource Quota ?

↓

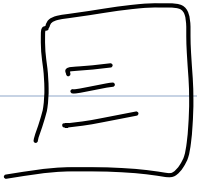
validating AC.

→ Mutating AC

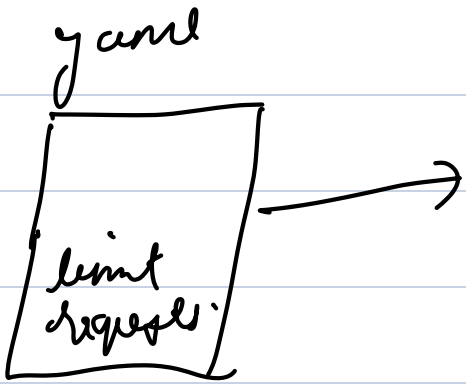
Admission Controller	Description
MutatingAdmissionWebhook	Calls an external webhook to modify API requests before storing them.
AlwaysPullImages	Forces all pods to pull images from the registry instead of using cached images.
DefaultStorageClass	Automatically assigns a default storage class to Persistent Volume Claims (PVCs) if none is specified.
LimitRanger	**Adds default resource requests and limits** if they are missing in a pod/container specification.



yaml.



No requests or limits



Validating AC

Admission Controller	Description
ValidatingAdmissionWebhook	Calls an external webhook to validate API requests before accepting them.
ResourceQuota	Enforces namespace-level quotas (CPU, memory, object count).
NodeRestriction	Restricts kubelet from modifying critical node and pod objects.

→ Dual Purpose. (Both mutating and validating)

Limit Ranger →

Adds default resource requests/limits if missing (mutates) and rejects pods that exceed limits (validates).

→ Demo

```
docker exec -it admission-demo-control-plane bash -c \
"sed -i 's/\(--enable-admission-plugins=[^ ]*\)/\1,LimitRanger,AlwaysPullImages/' /etc/kubernetes/manifests/kube-
apiserver.yaml"
```

restart Api-server after this
↓
delete the pod.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: pod-resource-limits
  namespace: limitranger-demo
spec:
  limits:
  - type: Container
    default:
      cpu: "500m"
      memory: "256Mi"
    defaultRequest:
      cpu: "250m"
      memory: "128Mi"
```

→ API Versions

v1 v2 v3 . . .

deprecated (still available → not suggested)
Removed. (not compatible)

→ apiVersion: group / version.

API groups.

API Group	Description	Example Resources
"" (core API, empty string)	Core API with fundamental resources	pods, nodes, services, secrets, configmaps
`apps`	Manages workloads and deployments	deployments, statefulsets, daemonsets
`batch`	Handles batch job workloads	jobs, cronjobs
`networking.k8s.io`	Controls networking-related resources	networkpolicies, ingress, ingressclasses
`storage.k8s.io`	Manages storage provisioning	storageclasses, volumeattachments
`certificates.k8s.io`	Handles certificate signing requests (CSRs)	certificatesigningrequests (CSR)
`admissionregistration.k8s.io`	Controls admission webhooks	mutatingwebhookconfigurations, validatingwebhookconfigurations

Alpha

Beta

Stable.

| **Alpha** | Experimental, may change anytime, not recommended for production. |

| --- | --- |

| **Beta** | More stable, but still subject to change, recommended for testing. |

| **Stable (GA)** | Fully supported and stable, safe for production use. |

Evolution of Deployments

1.8 → apps / v1 beta1

1.9 → apps / v1 beta2

1.10 → apps / v1

1.15

Removed.?

networking.k8s.io/v1beta1

V.1.19 → Kubernetes.

(deprecated)

v.1.22 Kubernetes.
(Removed.)

v 1.24

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
spec:
  rules:
    - host: my-app.example.com
      http:
        paths:
          - path: /
            backend:
              serviceName: my-service
              servicePort: 80
```

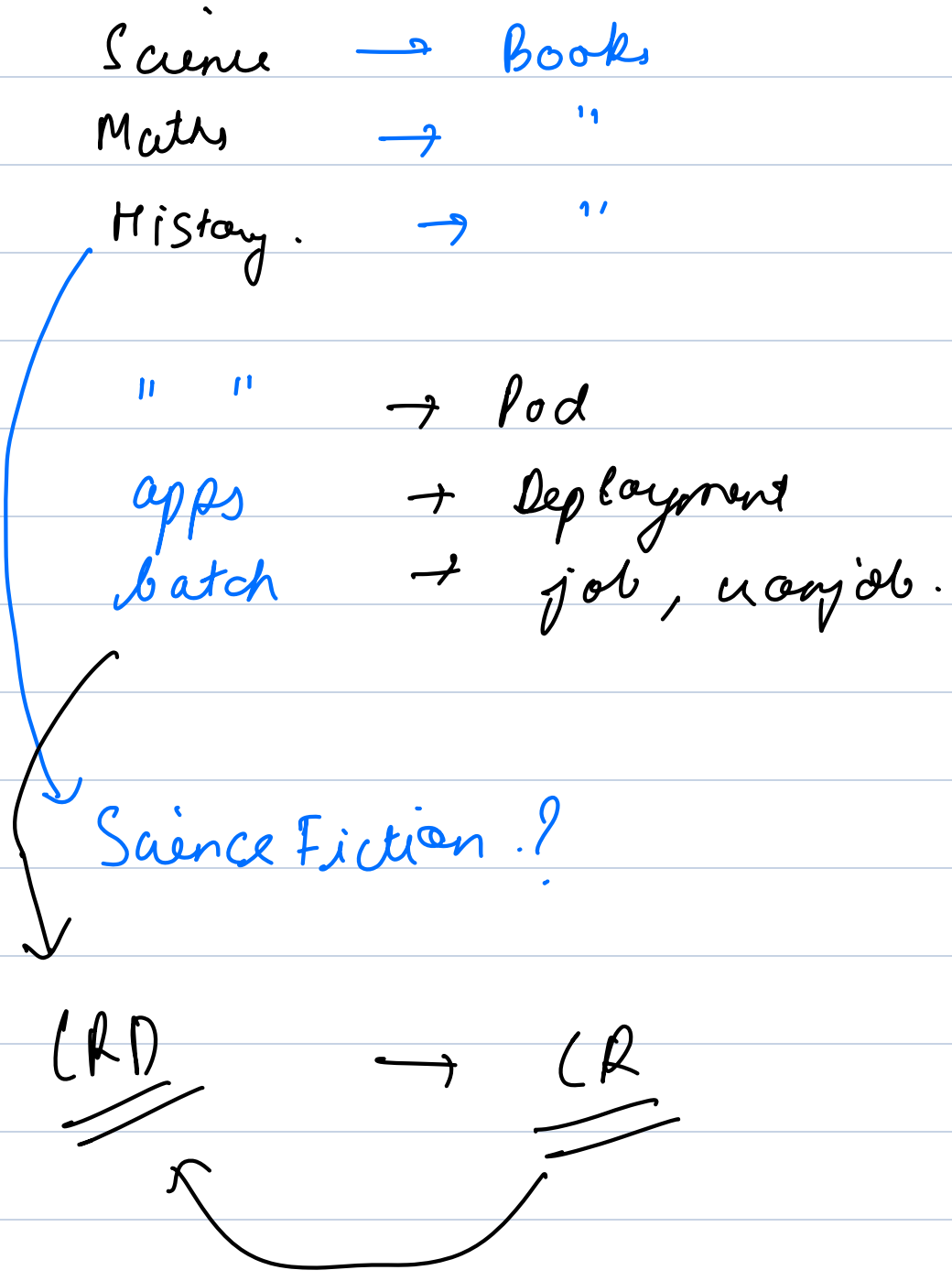
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
```

1.24

kubectl convert

Break → 10:15 pm

Custom Resource Definition & Custom Resources



A **Custom Resource Definition (CRD)** is a way to extend Kubernetes with your **own custom resources**, just like built-in ones (`Pods`, `Deployments`, `Services`, etc.).

(template) → CRD.

It **defines the structure** of a new resource type that Kubernetes should recognize.

v 1.10 → Ingress didn't exist

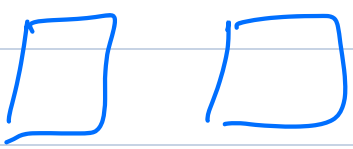
custom nginx , haproxy .

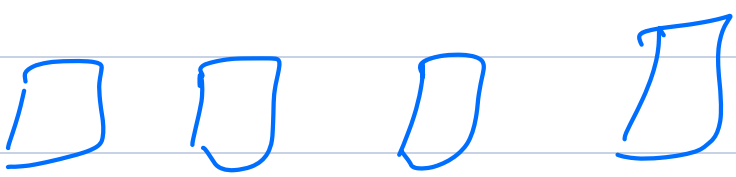
- **Kubernetes v1.1 (2015):** Ingress was **introduced as a built-in resource** under `networking.k8s.io/v1beta1`.

- **Kubernetes v1.19 (2020):** Ingress **graduated to GA** as `networking.k8s.io/v1`.

CRD → fruits
CR → apple.

Horizontal Pod Autoscaler. Autoscaling.





CRD

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: fruits.mygroup.example.com
spec:
  group: mygroup.example.com
  scope: Namespaced
  names:
    plural: fruits
    singular: fruit
    kind: Fruit
    shortNames:
      - fr
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
```

```
        type: object
        properties:
          color:
            type: string
          taste:
            type: string
```

`metadata.name` → The **fully qualified name** of the CRD.

- Follows the format: **`<plural>.<group>`** → `fruits.mygroup.example.com`.
- This is **how Kubernetes uniquely identifies this CRD**.

`group: mygroup.example.com` → Defines the **API group** where this resource will belong.

`scope: Namespaced` → Means that **each "Fruit" resource will exist within a specific namespace**.

- If set to `Cluster`, it would be **cluster-wide** instead.

- `plural: fruits` → The plural name when interacting with multiple resources (`kubectl get fruits`).
- `singular: fruit` → The singular form (`kubectl get fruit apple`).
- `kind: Fruit` → The **Kubernetes Kind** for this resource (`kubectl get Fruit`).
- `shortNames: ["fr"]` → A shorthand for easier use (`kubectl get fr`).

- `name: v1` → Specifies that this **version of the CRD is v1**.

- `served: true` → Kubernetes **accepts requests** for this version.

- `storage: true` → This version is the **primary version stored in etcd**.

- `schema: openAPIV3Schema` → Defines the **structure of the resource using OpenAPI v3 schema**. (Some standard)

- `type: object` → The CRD is an **object** with multiple fields.

- `properties: spec` → The `spec` field contains the actual **custom fields** of the resource.

- `properties:`

  - `color: type: string` → **Defines a "color" field**, which must be a string (e.g., `"red"`).

  - `taste: type: string` → **Defines a "taste" field**, which must also be a string (e.g., `"sweet"`).

apiVersion: mygroup.example.com/v1

kind: Fruit

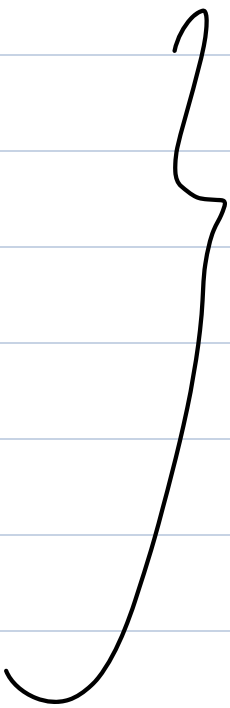
metadata:

 name: apple

spec:

 color: red

 taste: sweet



Custom Resource.

Tips for CKAD exam

Create a Pod

kubectl run nginx-pod --image=nginx

Create a Deployment

kubectl create deployment nginx-deployment --image=nginx

Exposing a Deployment creates a Service

kubectl expose deployment nginx-deployment --port=80 --target-port=80

Scaling a Deployment

kubectl scale deployment nginx-deployment --replicas=3

Updating a Resource Can be Asked.

kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1

Deleting a Resource

kubectl delete pod nginx-pod

Using Aliases for Speed

alias k='kubectl'

alias kgp='kubectl get pods'

alias kgn='kubectl get nodes'

alias kaf='kubectl apply -f'

Using Dry-Run to Generate YAML

kubectl create deployment nginx-deployment --image=nginx --dry-run=client -o yaml > deployment.yaml

Displays additional information.

kubectl get pods -o wide

kubectl get pods -o json

kubectl get pods -o yaml

Filtering Output with `--field-selector`:

kubectl get pods --field-selector status.phase=Running

Using `jq` for JSON Processing:

- `jq` is a powerful tool for processing JSON output. You can use it to filter and format JSON data from `kubectl`.

kubectl get pods -o json | jq '.items[] | {name: .metadata.name, namespace: .metadata.namespace,

status: .status.phase}'

Custom Columns:

- The `-o custom-columns` option allows you to specify which columns to display.

kubectl get pods -o custom-columns=POD_NAME:.metadata.name,STATUS:.status.phase

Using Label Selectors:

kubectl get pods -l app=my-app

Sorting and Watching Resources:

kubectl get pods --sort-by=.metadata.name

kubectl get pods --watch

Using Grep for Quick Searches

kubectl get pods -o wide | grep Running

What is kill.sh

→ CKAD Question - 1

In namespace **mct**, create a **PersistentVolume** named **mct-pv**. It should have:

- **Capacity**: 100Mi

- **Access mode**: ReadWriteOnce

- **HostPath**: `/tmp/data`

- **No Storage Class Name** defined

In the same namespace **mct**, create a **PersistentVolumeClaim** named **mct-pvc**. It should:

- **Request Storage**: 100Mi

- **Access Mode**: ReadWriteOnce

- **Note**: The PVC should bind to the PV correctly.

In the same namespace **mct**, create a **Pod** named **nginx-mct** using:

- **Image**: nginx

- **Container Name**: mct-container

- **Mount Path**: `/tmp/projectdata`

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mct-pv
  namespace: mct
spec:
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/tmp/data"
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mct-pvc
  namespace: mct
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi


apiVersion: v1
kind: Pod
metadata:
  name: nginx-mct
  namespace: mct
spec:
  containers:
    - name: mct-container
      image: nginx
      volumeMounts:
```

```
      - mountPath: "/tmp/projectdata"
        name: mct-storage
  volumes:
    - name: mct-storage
      persistentVolumeClaim:
        claimName: mct-pvc
```

CKAD question - 2

Developers occasionally need to submit pods that run periodically.

Task

Follow the steps below to create a pod that will start at a predetermined time and which runs to completion only once each time it is started:

Create a YAML formatted Kubernetes manifest that runs the following shell command:

```
date
```

in a single **busybox** container. The command should run **every minute** and must **complete within 22 seconds** or be terminated by Kubernetes. The **CronJob name and container name** should both be **hello**.

Create the resource in the above manifest and verify that the job executes successfully at least once.

→ activeDeadlineSeconds → 22

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"  # Runs every minute
  jobTemplate:
    spec:
      activeDeadlineSeconds: 22  # Terminate if not completed within 22s
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              command: ["date"]
          restartPolicy: Never
```

A Kubernetes administrator wants to ensure that all newly created pods specify resource limits before they are scheduled. Which admission controller helps enforce this policy?

A. ResourceQuota

B. **LimitRanger**

C. MutatingAdmissionWebhook

1. Finalisers

2. Topology Constraints

3. DEMO of OIDC

4. Cordon Uncordon

    1. Drain Node

5. HPA VPA

6. Istio

7. Linkerd

8. Kyverno

9. Gatekeeper

10. Endpoint Slices

upgrading cluster

Bkp etcd.

Prometheus Rules.