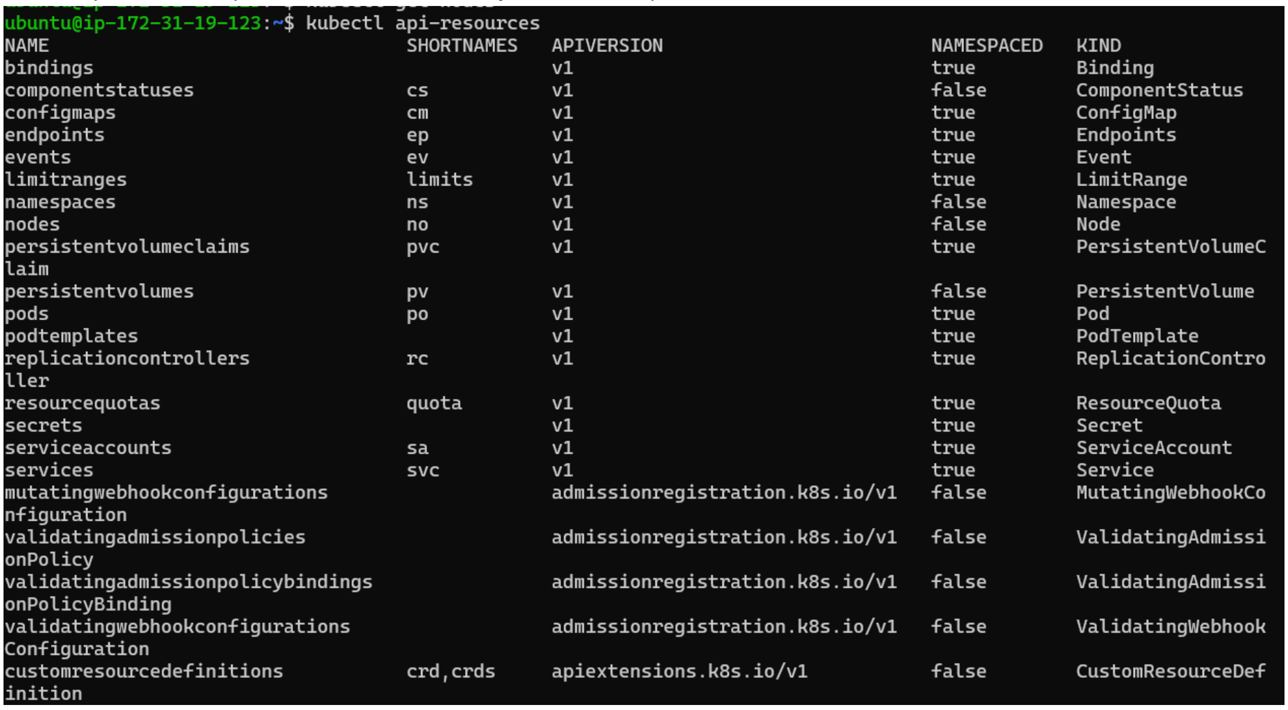
kubeadm setup tricks

* Use the following as user data

```
#!/bin/bash
curl -fsSL https://get.docker.com -o install-docker.sh
sh install-docker.sh
sudo apt-get update
# apt-transport-https may be a dummy package; if so, you can skip that package
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.14/cri-
dockerd_0.3.14.3-0.ubuntu-jammy_amd64.deb
sudo dpkg -i cri-dockerd_0.3.14.3-0.ubuntu-jammy_amd64.deb
```

* Kubeadm has two major operations
  * init: this initializes the k8s cluster and is executed on the master node
  * reset: to remove the cluster we can use reset.
* Refer Here for kubectl cheatsheet
* How kubectl works?
  * watch classroom video

- Kube-api server exposes k8s functionality over rest api



```
ubuntu@ip-172-31-19-123:~$ kubectl api-resources
NAME                              SHORTNAMES   APIVERSION                        NAMESPACED   KIND
bindings                                       v1                                true         Binding
componentstatuses                 cs           v1                                false        ComponentStatus
configmaps                        cm           v1                                true         ConfigMap
endpoints                         ep           v1                                true         Endpoints
events                            ev           v1                                true         Event
limitranges                       limits       v1                                true         LimitRange
namespaces                        ns           v1                                false        Namespace
nodes                             no           v1                                false        Node
persistentvolumeclaims            pvc          v1                                true         PersistentVolumeC
laim
persistentvolumes                 pv           v1                                false        PersistentVolume
pods                              po           v1                                true         Pod
podtemplates                                   v1                                true         PodTemplate
replicationcontrollers            rc           v1                                true         ReplicationContro
ller
resourcequotas                    quota        v1                                true         ResourceQuota
secrets                                        v1                                true         Secret
serviceaccounts                   sa           v1                                true         ServiceAccount
services                          svc          v1                                true         Service
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1   false        MutatingWebhookCo
nfiguration
validatingadmissionpolicies                    admissionregistration.k8s.io/v1   false        ValidatingAdmissi
onPolicy
validatingadmissionpolicybindings              admissionregistration.k8s.io/v1   false        ValidatingAdmissi
onPolicyBinding
validatingwebhookconfigurations                admissionregistration.k8s.io/v1   false        ValidatingWebhook
Configuration
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1           false        CustomResourceDef
inition
```

- Since functionality is exposed over api, we have different client libraries to interact with k8s programatically Refer Here
- We will be using kubectl and kubectl has two modes of working
    - imperative:
        - we build commands to do the work on k8s
    - declarative
        - we create k8s manifests in yaml files and pass it to the kubectl
        - this is recommeded approach.
        - kubectl supports two simple commands if we have manifests
            - apply: to create or update changes `kubectl apply -f <mainfest file/folderpath>`
            - delete: to remove the objects `kubectl delete -f <mainfest file/folderpath>`

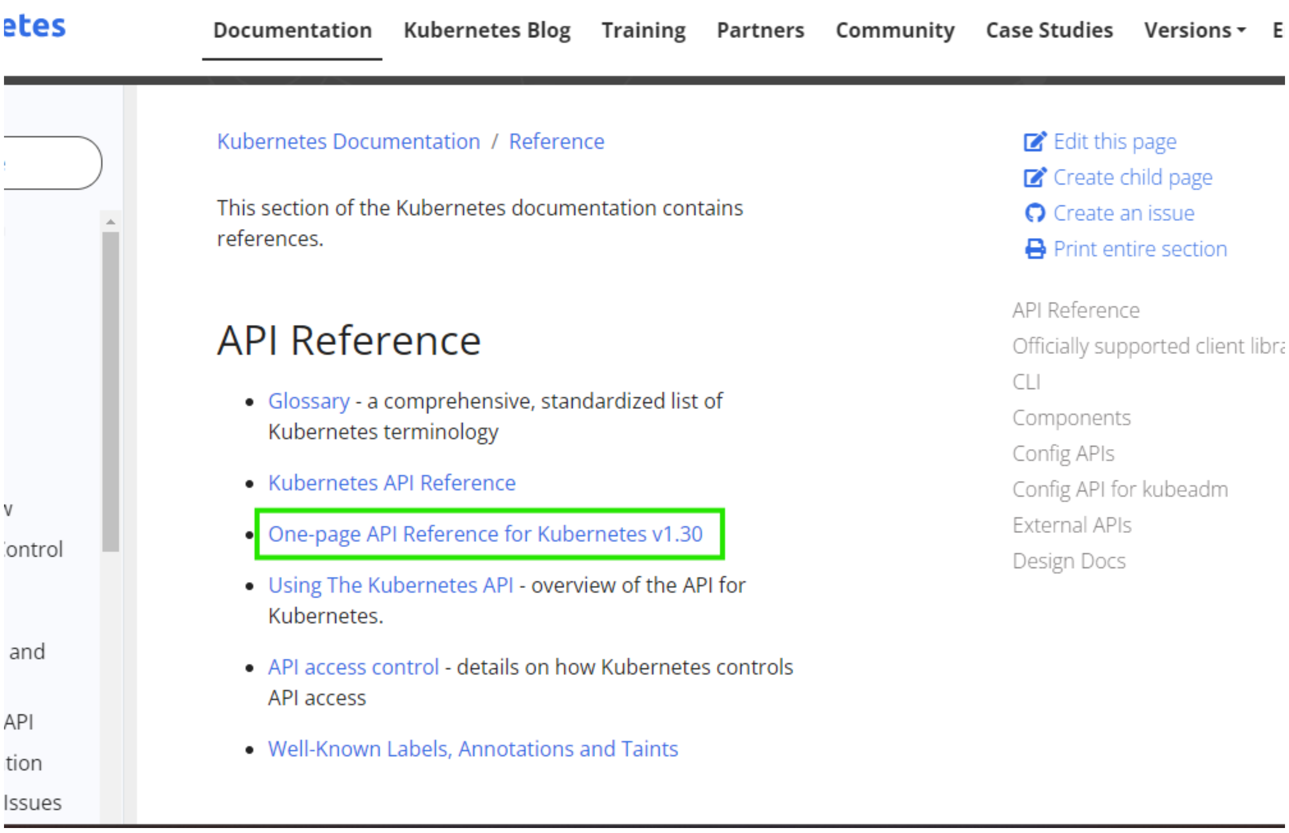
How to write manifest files

- Lets understand manifest file structure
- Generally manifest files have the following structures

```
apiVersion => passed by us
kind  => passed by us
metadata  => passed by us
spec => passed by us
status  => result of execution
```

- apiVersion: Refer Here
- kind: This represents the type of object which we are creating

- metadata:
  - here we provide name, label
- spec: this is specification of what we want
- Navigate to api reference Refer Here

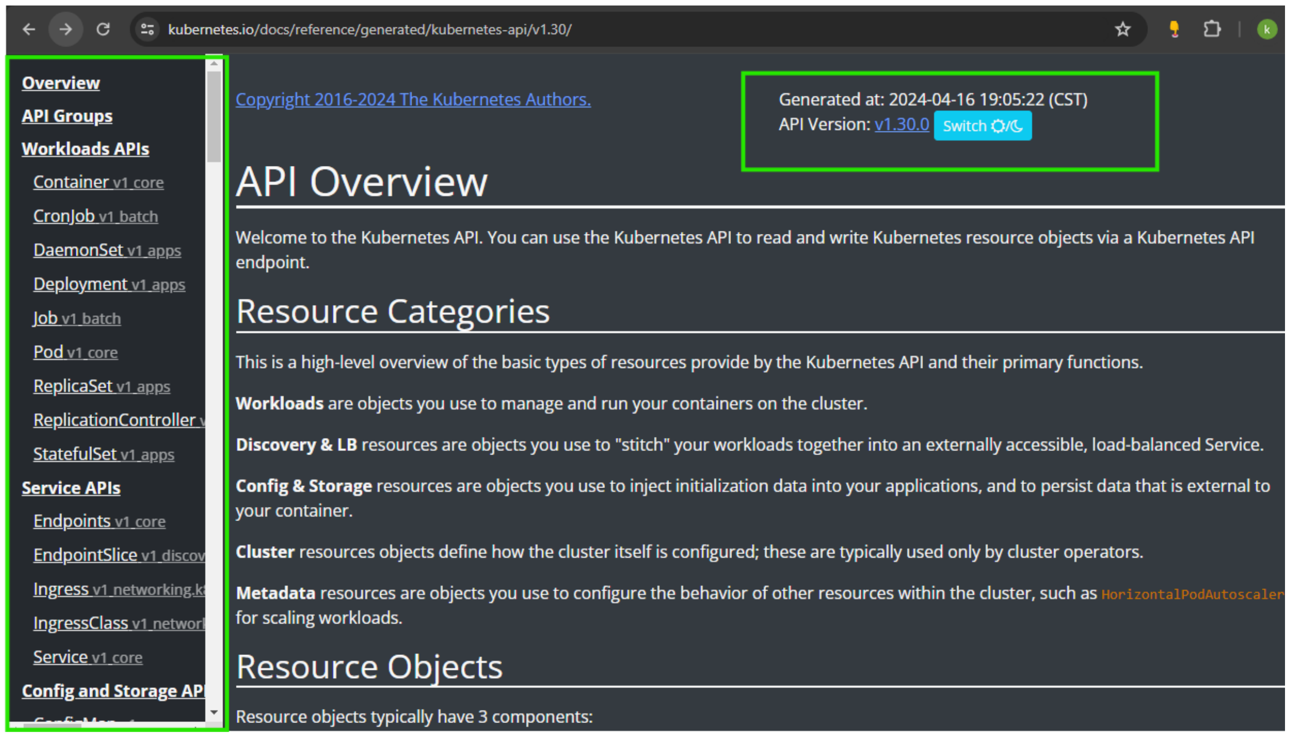


Kubernetes Documentation / Reference

This section of the Kubernetes documentation contains references.

API Reference

- Glossary - a comprehensive, standardized list of Kubernetes terminology
- Kubernetes API Reference
- One-page API Reference for Kubernetes v1.30
- Using The Kubernetes API - overview of the API for Kubernetes.
- API access control - details on how Kubernetes controls API access
- Well-Known Labels, Annotations and Taints



Copyright 2016-2024 The Kubernetes Authors.

Generated at: 2024-04-16 19:05:22 (CST)
API Version: v1.30.0

API Overview

Welcome to the Kubernetes API. You can use the Kubernetes API to read and write Kubernetes resource objects via a Kubernetes API endpoint.

Resource Categories

This is a high-level overview of the basic types of resources provide by the Kubernetes API and their primary functions.

Workloads are objects you use to manage and run your containers on the cluster.

Discovery & LB resources are objects you use to "stitch" your workloads together into an externally accessible, load-balanced Service.

Config & Storage resources are objects you use to inject initialization data into your applications, and to persist data that is external to your container.

Cluster resources objects define how the cluster itself is configured; these are typically used only by cluster operators.

Metadata resources are objects you use to configure the behavior of other resources within the cluster, such as `HorizontalPodAutoscaler` for scaling workloads.

Resource Objects

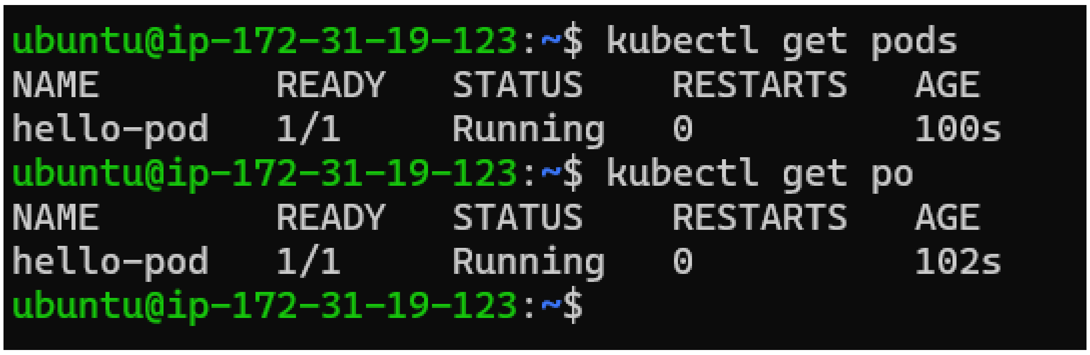
Resource objects typically have 3 components:

Lets write our first pod manifest

- Pod is an atomic unit of creation in k8s which runs container(s) in it
- Lets run nginx container in the Pod
- create a new file with .yaml or .yml extension with following content as discussed in the class

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: hello-pod
spec:
  containers:
    - name: hello-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

- Now apply and get info about pods



```
ubuntu@ip-172-31-19-123:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
hello-pod   1/1     Running   0          100s
ubuntu@ip-172-31-19-123:~$ kubectl get po
NAME        READY   STATUS    RESTARTS   AGE
hello-pod   1/1     Running   0          102s
ubuntu@ip-172-31-19-123:~$
```