kubernetes

OBJECTS DECLARATIVE

❖ KUBERNETES OBJECTS:

- Kubernetes objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster.
- Specifically, they can describe:
  - What containerized applications are running (and on which nodes)
  - The resources available to those applications
  - The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance.

OBJECT SPEC AND STATUS:

- Every Kubernetes object includes **two nested object fields** that govern the object's configuration: **object spec & object status.**
- For **objects** that have **a spec,** you have to set this when you create the object, providing a description of the characteristics you want the resource to have: its desired state.
- The **status** describes the current state of the object, supplied and updated by the Kubernetes system and its components.

DESCRIBING A KUBERNETES OBJECT:

- When you use the Kubernetes API to create the object (either directly or via kubectl), that API request must include that information as JSON in the request body.
- Most often, you provide the information to kubectl in a **.yaml** file. kubectl converts the information to JSON when making the API request.

➢ OBJECT NAMES AND IDS:

- Each object in your cluster has a Name that is unique for that type of resource.
- Every Kubernetes object also has a UID that is unique across your whole cluster.

NAMES:

- A client-provided string that refers to an object in a resource URL, such as /api/v1/pods/somename.
- Four types of commonly used name constraints for resources.

DNS SUBDOMAIN NAMES:

- Most resource types require a name that can be used as a DNS subdomain name as defined in RFC 1123.
- This means the name must:
  - contain no more than 253 characters
  - contain only lowercase alphanumeric characters, '-' or '.'
  - start with an alphanumeric character
  - end with an alphanumeric character

RFC 1123 LABEL NAMES:

- Some resource types require their names to follow the DNS label standard as defined in RFC 1123.
- This means the name must:
  - contain at most 63 characters
  - contain only lowercase alphanumeric characters or '-'
  - start with an alphanumeric character
  - end with an alphanumeric character

RFC 1035 LABEL NAMES:

- Some resource types require their names to follow the DNS label standard as defined in RFC 1035.
- This means the name must:
  - contain at most 63 characters
  - contain only lowercase alphanumeric characters or '-'
  - start with an alphabetic character
  - end with an alphanumeric character

PATH SEGMENT NAMES:

- Some resource types require their names to be able to be safely encoded as a path segment.
- In other words, the name may not be "." or ".." and the name may not contain "/" or "%".

REQUIRED FIELDS:

- In the .yaml file for the Kubernetes object you want to create, you'll need to set values for the following fields:

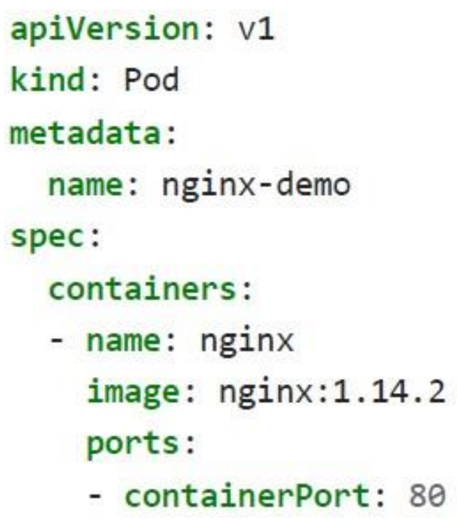
  apiVersion: Which version of the Kubernetes API you're using to create this object

  kind: What kind of object you want to create

  metadata: Data that helps uniquely identify the object, including a name string, UID, and optional namespace

  spec: What state you desire for the object

EXAMPLE: MANIFEST FOR A POD NAMED NGINX-DEMO:



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-demo
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

→ **To preview changes to an object's configuration before applying:**

$kubectl diff -f nginx-demo.yml

→ **To create or update an object:**
$kubectl apply -f nginx-demo.yml
$kubectl get pods

→ **To connect a container in pod:**
$kubectl exec -it nginx-demo – bash

VIEWING AN OBJECT:

- The final step in managing Kubernetes declarative objects is to view the object configuration to ensure that the changes have been applied correctly.
- You can use the kubectl get command to view the current state of an object, and use the -o yaml option to view the object configuration in YAML format:

```
$kubectl get -f <filename|url> -o yaml

$kubectl get -f nginx-demo -o yaml
```

→ **To delete a pod:**

```
$kubectl delete pod nginx-demo

$kubectl get pods
```

EXAMPLE: TWO CONTAINERS IN A POD:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
  labels:
    app: web
spec:
  containers:
    - name: front-end
      image: nginx
      ports:
        - containerPort: 80
    - name: rss-reader
      image: nickchase/rss-php-nginx:v1
      ports:
        - containerPort: 88
```