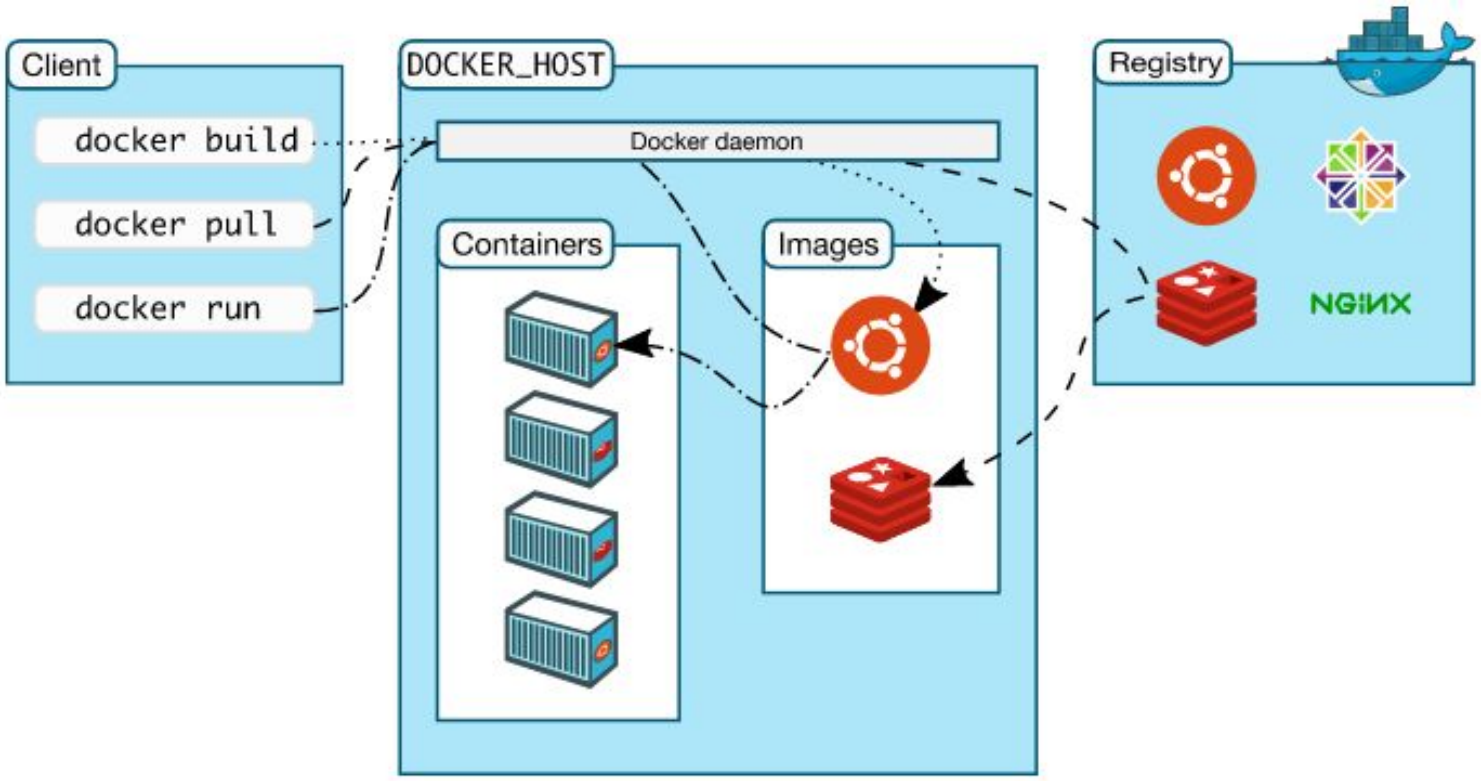
Kubernetes

Solar Team

Prepare Your Environments

- Windows/MacOS, Ram 16
- Docker Desktop + enable Kubernetes
- The Kubernetes command-line tool, **kubectl**.
- VPN Connection

Docker Architecture



- Containers are a good way to bundle and run your applications.
- How can we deal with large numbers of container and distributed issues?

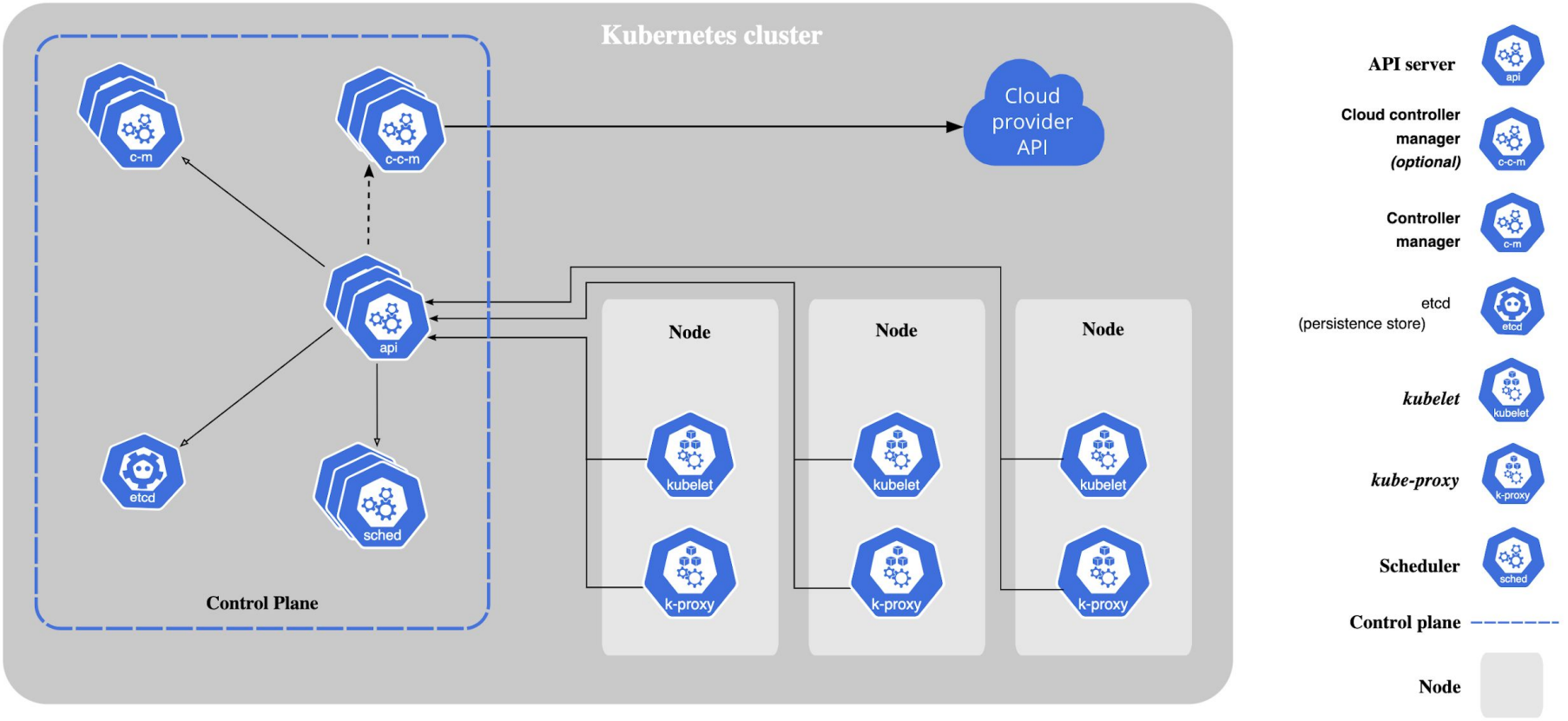
Kubernetes Overview

- An **open source container orchestration platform** designed for running distributed applications and services at scale.
- The core functionality is scheduling workloads in containers across your infrastructure.
- Google open-sourced the Kubernetes project in 2014.
- Known as K8s.
- Lastest version is v1.20 (February 2021)
- https://kubernetes.io/

Why do we need Kubernetes?

- Kubernetes provides you with:
  - **Service discovery and load balancing**
  - **Storage orchestration**
  - **Automated rollouts and rollbacks**
  - **Automatic bin packing**
  - **Self-healing**
  - **Secret and configuration management**

Kubernetes Architecture



kubectl

- **kubectl** is the command-line tool to interact with the Kubernetes clusters.
- It makes the necessary Kubernetes API calls for you.
- kubectl [command] [TYPE] [NAME] [flags]
- Example:
  - kubectl version --short
  - kubetclt get pods
  - kubectl get po
  - kubectl get po etcd-docker-desktop --namespace kube-system
  - kubectl get svc #(or service or services)
  - kubectl get po --namespace kube-system
  - kubectl describe namespaces default
  - kubectl cluster-info --kubeconfig ~/.kube/config.180
  - kubectl run nginx --image=nginx --replicas=3 --labels=app=web-proxy --dry-run -o=yaml

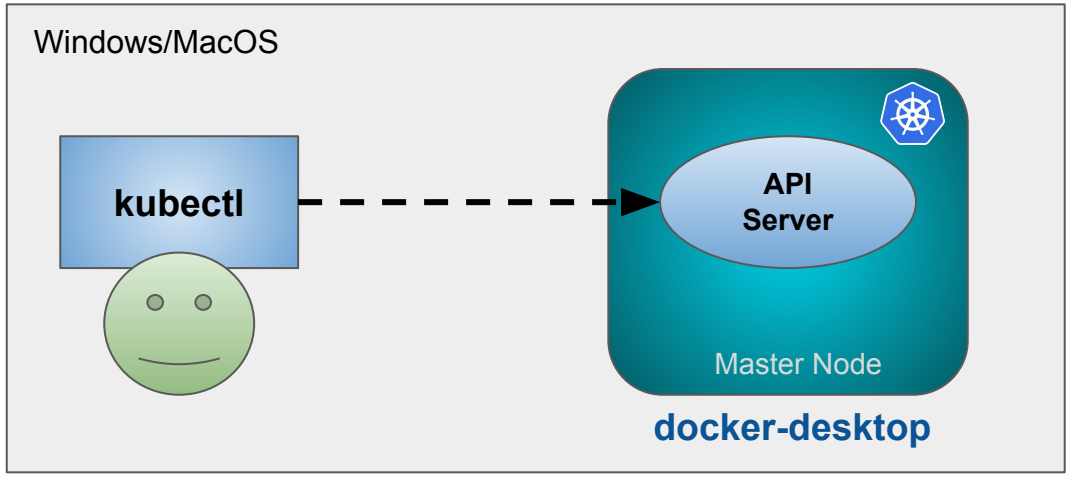
Kubernetes Dashboard

- **For kube 1.14**
  - kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml
- **Access Dashboard**
  - kubectl proxy
  - http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/
- **Get Token**
  - kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')
- **Another tool is Lens (Kubernetes IDE).**

Kubernetes API

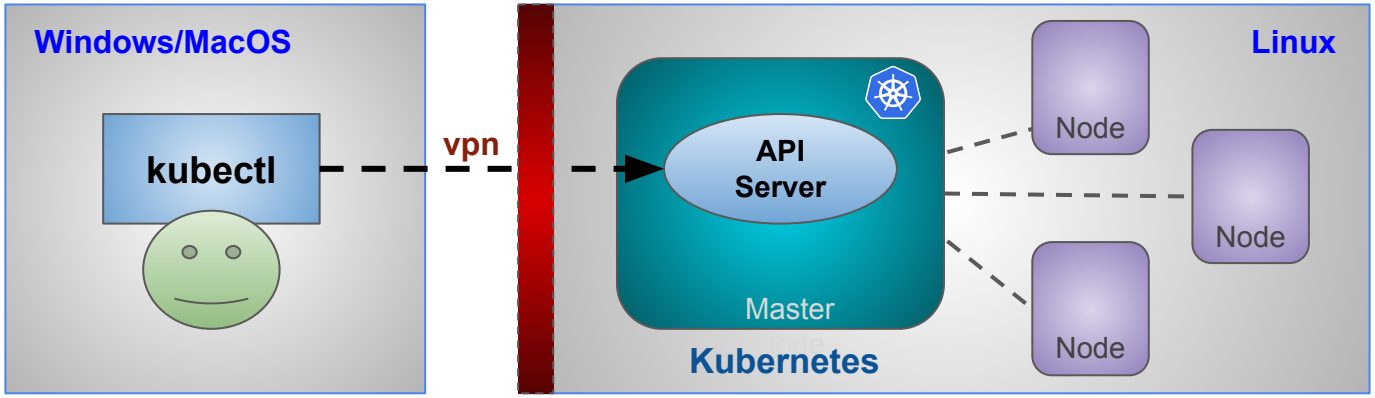
- The core of Kubernetes' control plane is the **API server**.
- The API server exposes an HTTP API.
- Used for communication between user (query and manipulate the kubernetes objects), internal components and external components.
- Most operations can be performed through the **kubectl** and other tool like **kubeadm**.
- Can also access the API directly using REST calls.
- Provided client library for several programming languages like Go, Java, Python, etc.
- Kubernetes stores the serialized state of objects by writing them into **etcd**.
- Support both response format, json and protobuf.

Access Local Kubernetes Cluster



- Start docker and enable local kubernetes.
- Open a terminal console then try these commands.
  - kubectl cluster-info # Print the address of the master and cluster services
  - kubectl config current-context # view the current context
  - kubectl config view # Display merged kubeconfig settings or a specified kubeconfig file
- To access api server.
  - kubectl proxy # default port is 8001
  - Open this url in browser - http://localhost:8001/api/v1/
  - kubectl proxy --port=8080 # change proxy port to 8080

Access Remote Kubernetes Cluster



- Access a remote cluster with Kubectl
  - Copy Kubernetes config file from the master node to for example **(user-home)/.kube/config.180**
  - **Solution #1** - replace local config file with remote cluster config file.
  - **Solution #2** - setting **KUBECONFIG** environment variable.
    - export KUBECONFIG=~/.kube/config.180
  - **Solution #3** - using --kubeconfig option.
    - kubectl cluster-info --kubeconfig ~/.kube/config.180
  - **Solution #4** - using proxy (access kubernetes api)
    - kubectl proxy
    - Or using kubectl proxy --kubeconfig ~/.kube/config.180

Use with caution!

Resource Short-names

Short Name	Full Name	Short Name	Full Name
csr	certificatesigningrequests	**pvc**	persistentvolumeclaims
cs	componentstatuses	**pv**	persistentvolumes
cm	configmaps	**po**	pods
ds	daemonsets	**pdb**	poddisruptionbudgets
deploy	deployments	**psp**	podsecuritypolicies
ep	endpoints	**rs**	replicasets
ev	events	**rc**	replicationcontrollers
hpa	horizontalpodautoscalers	**quota**	resourcequotas
ing	ingresses	**sa**	serviceaccounts
limits	limitranges	**svc**	services
ns	namespaces		
no	nodes		

Basic Commands

- **kubectl version --short** - Print the client and server version information for the current context
- **kubectl options** - Print a list of global command-line options (applies to all commands).
- **kubectl api-versions** - Print the supported **API versions on the server**, in the form of "group/version".
- **kubectl api-resources** - Print the supported API resources on the server.
- **kubectl proxy** - Creates a proxy server or application-level gateway between localhost and the Kubernetes API Server.
- **kubectl explain** - List the fields for supported resources.

Kubernetes Release Notes

- https://v1-18.docs.kubernetes.io/docs/setup/release/notes/

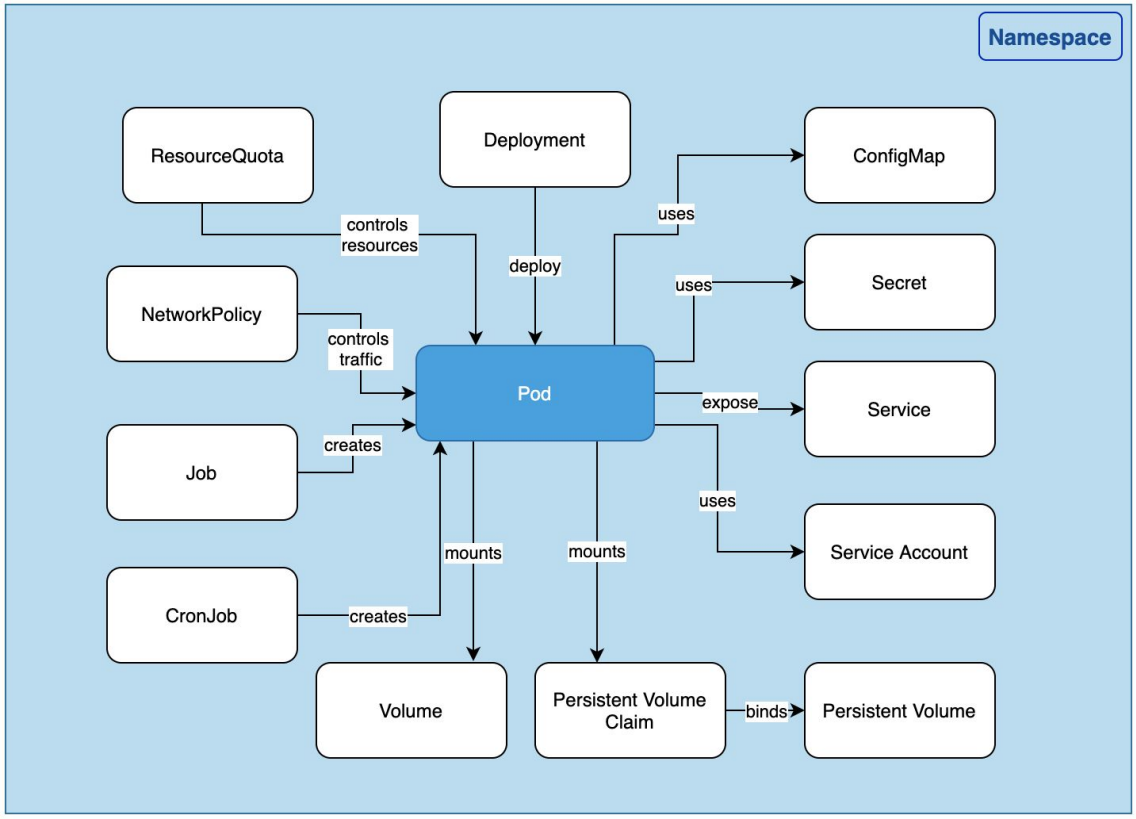
  - kube-apiserver no longer serves the following deprecated APIs: (#85903, @liggitt) * All resources under `apps/v1beta1` and `apps/v1beta2` - use `apps/v1` instead * `daemonsets`, `deployments`, `replicasets` resources under `extensions/v1beta1` - use `apps/v1` instead * `networkpolicies` resources under `extensions/v1beta1` - use `networking.k8s.io/v1` instead * `podsecuritypolicies` resources under `extensions/v1beta1` - use `policy/v1beta1` instead

kubeconfig And Context

- **Kubeconfig files** keep an organize information about clusters, users, namespaces, and authentication mechanisms.
- The kubectl command-line tool uses kubeconfig files to find the information it needs to choose a cluster and communicate with the API server of a cluster.
- The kubectl looks for a file named config in the $HOME/.kube directory.
- Specify other kubeconfig files by setting the **KUBECONFIG** environment variable or by setting the **--kubeconfig** flag.
- **Context** element in a kubeconfig file is used to group access parameters under a convenient name.
- Each context has three parameters: **cluster, namespace, and user.**
- By default, the kubectl command-line tool uses parameters from the current context to communicate with the cluster.

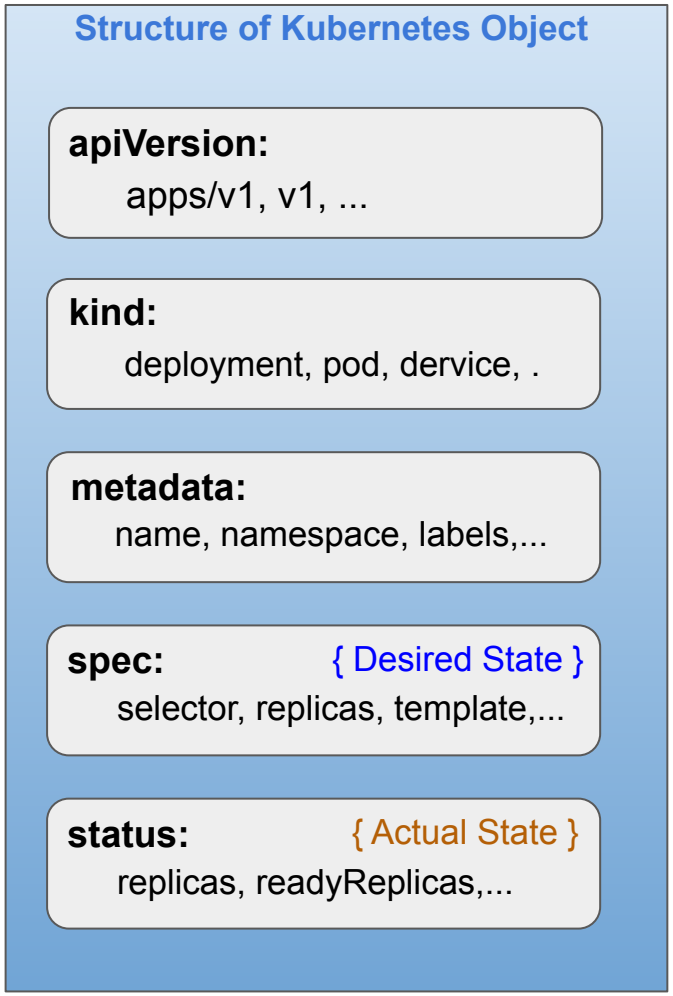
Kubernetes Primitives(Resources)

- The basic building blocks in the Kubernetes architecture for creating and operating an application on the platform.



Some of Kubernetes Primitives

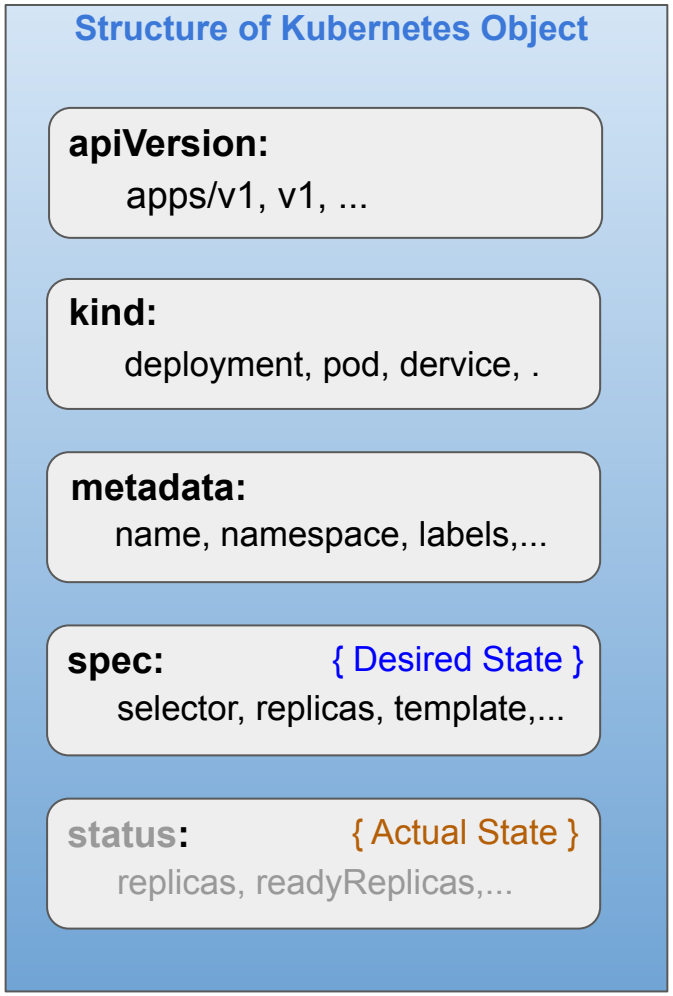
Kubernetes Object



- Kubernetes objects are **persistent entities** in the Kubernetes system.
- Kubernetes uses these entities to **represent the state** of your cluster.
- Kubernetes uses the **YAML file** for declaring.
- A Kubernetes object is a "record of intent"
- Use the **kubectl** command-line to work(create, modify, delete, etc.) with Kubernetes objects.

https://v1-18.docs.kubernetes.io/docs/setup/release/notes/

Kubernetes Object Structure



- **API version** - Which version of the Kubernetes API you're using to create this object.
- **Kind** - What kind of object you want to create e.g., a Pod or a Service.
- **Metadata** - Data that helps uniquely identify the object, including a **name** string, **UID**, and optional namespace.
- **The specification ("spec" for short)** - What state you desire for the object. Which image should run in the container, or which environment variables should be set for?
- **Status** - Describes the actual state of an object.

Kubernetes Object Management

- Using **kubectl** command-line tool supports several different ways to create and manage Kubernetes objects.
- Two approach - **Imperative approach** and **Declarative approach**.
- **Warning:** Use only one technique, mixing techniques for the same object results in undefined behavior.

Imperative approach

- Two kinds of Imperative approaches, **Imperative commands** and **Imperative object configuration**
- **Imperative commands** does not require a manifest definition(YAML file).
  - Use the kubectl run or kubectl create command to create an object on the fly.
  - kubectl create deployment nginx --image nginx
  - kubectl run web --image=nginx --restart=Never --port=80 # **run-web_get.yml**
- **Imperative object configuration** creates objects from a manifest file (YAML file)
  - kubectl create -f nginx.yaml
  - kubectl delete -f nginx.yaml -f redis.yaml
  - kubectl replace -f nginx.yaml # Update the objects defined in a configuration file by overwriting the live configuration:
- **The recommended way for development environments.**
- **Dry run**. Print the corresponding API objects without creating them.
  - kubectl create nginx --image=nginx **--dry-run**

Declarative approach

- **Declarative object configuration -** create, update, and delete operations are automatically detected per-object by kubectl.
  - kubectl apply -f configs/
  - kubectl diff -f configs/
  - kubectl apply -R -f configs/ # Recursively process directories.
  - **This approach is a recommended way for production environments.**

Namespace

Namespaces

- **Namespace** is a **virtual cluster concept** in Kubernetes.
- Kubernetes supports multiple virtual clusters backed by the same physical cluster.
- Provide **isolated resources that are spread across teams and projects** - via resource quotas, network policies.
- Names of resources need to be **unique within a namespace**, but not across namespaces.
- Four initial namespaces are **default**, **kube-system**, **kube-public** and **kube-node-lease.**
- **Not all objects are in a namespace** for example Node, PersistentVolume and Namespace itself.
- Prefix “**kube-**” is reserved for Kubernetes system namespaces.

Namespace Commands

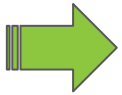
- Viewing namespaces
  - kubectl get namespace
- Setting the namespace for a request
  - kubectl run nginx --image=nginx --namespace=<insert-namespace-name-here>
  - kubectl get pods --namespace=<insert-namespace-name-here>
- Setting the namespace preference
  - kubectl config set-context --current --namespace=<insert-namespace-name-here>
  - kubectl config view --minify | grep namespace # verify it
- List objects in a namespace
  - kubectl api-resources --namespaced=true
- List objects not in a namespace
  - kubectl api-resources --namespaced=false

Create A Namespace Example

kubectl create namespace dev --dry-run -o=yaml > namespace-dev.yml

```yaml
apiVersion: v1
kind: Namespace
metadata:
 creationTimestamp: null
 name: dev
spec: {}
status: {}
```

namespace-dev.yml



kubectl get ns dev -o yaml > namespace-dev_get.yml

```yaml
apiVersion: v1
kind: Namespace
metadata:
 annotations:
   kubectl.kubernetes.io/last-applied-configuration: |
     {"apiVersion":"v1","kind":"Namespace",
     "metadata":{"annotations":{},"name":"dev"}}
 creationTimestamp: "2021-02-28T08:31:45Z"
 name: dev
 resourceVersion: "638999"
 selfLink: /api/v1/namespaces/dev
 uid: 641f333d-799f-11eb-bd28-025000000001
spec:
 finalizers:
 - kubernetes
status:
 phase: Active
```

namespace-dev_get.yml

Deployment

Deployment

- A Deployment is responsible for creating and updating instances of your application (Pods).
- Deployment adds several important features for a production environment.
  - Scalability
  - Update strategy
  - Historical
- Deployment creates:
  - A **ReplicaSet** to ensure that the desired number of Pods is always available,
  - Specifies a strategy to replace Pods (such as RollingUpdate).

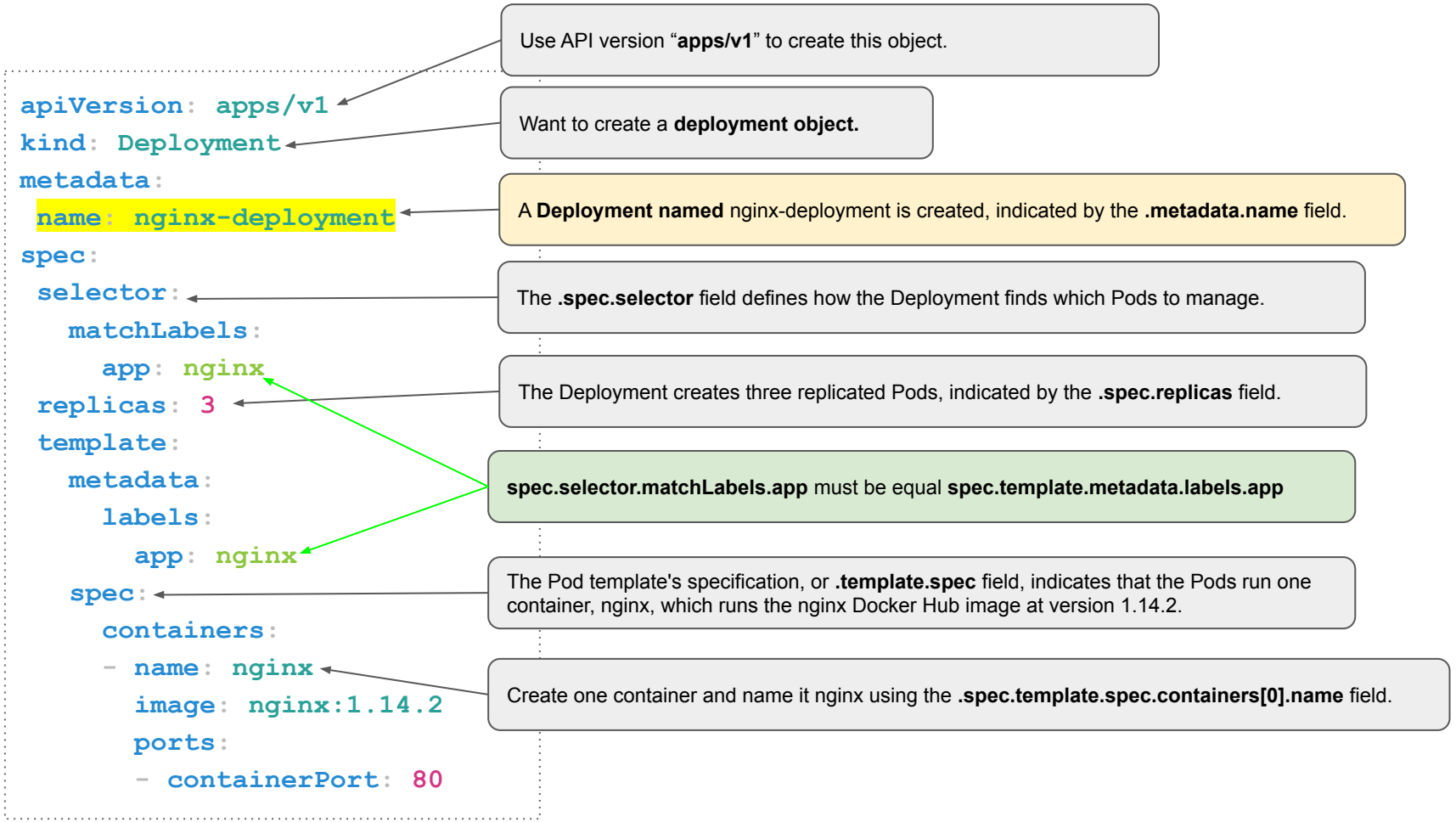
Create Deployment(Imperative Way)

- kubectl create deployment web-server --image=nginx
- kubectl get deployment web-server -o yaml > web-server_get.yml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2021-03-02T13:48:40Z"
  generation: 1
  labels:
    app: web-server
  name: web-server
  namespace: default
  resourceVersion: "752957"
  selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/web-server
  uid: fee753c6-7b5d-11eb-bd28-025000000001
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: web-server
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: web-server
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 1
  conditions: …
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
  updatedReplicas: 1
```

web-server_get.yml

Basic Deployment Manifest



nginx-deployment.yml

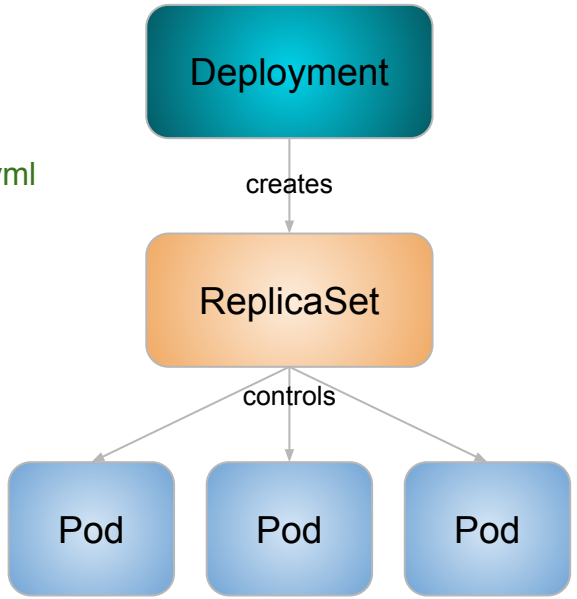
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.20/

Create Deployment from YAML (Declarative Way)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

nginx-deployment.yml

kubectl create -f nginx-deployment.yml




```
kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           13m
```

```
kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-756d9fd5f9   3         3         3       4h10m
```

```
kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-756d9fd5f9-2fkx6   1/1     Running   0          13m
nginx-deployment-756d9fd5f9-9pl8x   1/1     Running   0          13m
nginx-deployment-756d9fd5f9-vjq6p   1/1     Running   0          33s
```

Compare Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

nginx-deployment.yml

kubectl get deployment nginx-deployment
-o=yaml > nginx-deployment_get.yml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "2"
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"name":
  creationTimestamp: "2021-03-01T10:08:58Z"
  generation: 3
  labels:
    app: nginx
  name: nginx-deployment
  namespace: default
  resourceVersion: "693345"
  selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/nginx-deployment
  uid: 23a62fa5-7a76-11eb-bd28-025000000001
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector: …
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec: …
status:
  availableReplicas: 3
  conditions:
  - lastTransitionTime: "2021-03-01T10:22:24Z" …
  - lastTransitionTime: "2021-03-01T10:08:58Z" …
  observedGeneration: 3
  readyReplicas: 3
  replicas: 3
```

ReplicaSet

- Replicate a guaranteed number of Pods with the same configuration.
- Deployment is the higher-level concept that manages the ReplicaSet internally with no involvement required by the end user.
- ReplicaSets provide self-healing capabilities.

Lab: Deployment

- kubectl create -f nginx-deployment.yml Or kubectl apply -f nginx-deployment.yml
- kubectl get deployments,pods,replicasets
- kubectl describe deployment.apps/nginx-deployment
- kubectl set image deployment nginx nginx=nginx:1.19.2
- kubectl rollout history deployment nginx
- kubectl get deployments,pods,replicasets
- kubectl get pods -A
- kubectl describe deployment nginx
- kubectl describe deployment.app/nginx-deployment
- kubectl rollout history deployments nginx-deployment --revision=2
- kubectl port-forward <POD> 8002:80
- kubectl rollout undo deployment my-deploy --to-revision=1

nginx-deployment.yml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
 name: nginx
 annotations:
   app-domain: "infra"
spec:
 selector:
   matchLabels:
     app: nginx
 replicas: 2
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: nginx:1.14.2
       ports:
       - containerPort: 80
```

Names and UIDs

- Each object in your cluster has a **Name** that is unique for that type of resource.
- Every Kubernetes object also has a **UID**(systems-generated uuid string) that is unique across your whole cluster.
- For example, you can only have one Pod named myapp-1234 within the same namespace, but you can have one Pod and one Deployment that are each named myapp-1234.
- For non-unique user-provided attributes, Kubernetes provides **labels** and **annotations**.
- A client-provided string that refers to an object in a resource URL, such as /api/v1/pods/**some-name**.

Label

- Labels are **key/value pairs** that are attached to objects, such as deployment.
- Label **key must be unique** for a given object.
- Labels can be attached to objects at creation time and subsequently added and modified at any time.
- Can manually set and automatically set by Kubernetes itself.
- Used to organize and to locate subsets of objects.
- ReplicaSet monitor the pod by using labels.

```
"metadata": {
  "labels": {
    "key1" : "value1",
    "key2" : "value2"
  }
}
```

Recommended Labels

- There are recommended labels. They make it easier to manage applications but aren't required.
- Shared labels and annotations share a common prefix: **app.kubernetes.io**.
- Labels without a prefix are private to users.

Key	Description	Example	Type
`app.kubernetes.io/name`	The name of the application	`mysql`	string
`app.kubernetes.io/instance`	A unique name identifying the instance of an application	`mysql-abcxzy`	string
`app.kubernetes.io/version`	The current version of the application (e.g., a semantic version, revision hash, etc.)	`5.7.21`	string
`app.kubernetes.io/component`	The component within the architecture	`database`	string
`app.kubernetes.io/part-of`	The name of a higher level application this one is part of	`wordpress`	string
`app.kubernetes.io/managed-by`	The tool being used to manage the operation of an application	`helm`	string

Label Examples

Metadata **{Deployment}**

Name: graviteeio-apim3x-portal
Namespace: gravitee
Creation time: Nov 11, 2020
Age: 3 months

UID: 1f6345f5-0013-4026-aa58-44d4c51048d8

Labels
app.kubernetes.io/component: portal
app.kubernetes.io/instance: graviteeio-apim3x
app.kubernetes.io/managed-by: Helm
app.kubernetes.io/name: apim3
app.kubernetes.io/version: 3.3.3
helm.sh/chart: apim3-3.0.13
Show less

Annotations
deployment.kubernetes.io/revision: 1
meta.helm.sh/release-name: graviteeio-apim3x
meta.helm.sh/release-namespace: gravitee

Metadata **{Service}**

Name: graviteeio-apim3x-portal
Namespace: gravitee
Creation time: Nov 11, 2020
Age: 3 months

UID: 4672e013-2ca3-437c-a81c-5124e88c56a9

Labels
app.kubernetes.io/component: portal
app.kubernetes.io/instance: graviteeio-apim3x
app.kubernetes.io/managed-by: Helm
app.kubernetes.io/name: apim3
app.kubernetes.io/version: 3.3.3
helm.sh/chart: apim3-3.0.13
Show less

Annotations
meta.helm.sh/release-name: graviteeio-apim3x
meta.helm.sh/release-namespace: gravitee

Metadata **{Pod}**

Name: graviteeio-apim3x-portal-869fd8b9-m5kln
Namespace: gravitee
Creation time: Nov 11, 2020
Age: 3 months

UID: 35bc760c-1079-44f9-935f-6734d44e0d74

Labels
app.kubernetes.io/component: portal
app.kubernetes.io/instance: graviteeio-apim3x
app.kubernetes.io/name: apim3
pod-template-hash: 869fd8b9

Annotations
chaos.alpha.kubernetes.io/enabled: false
checksum/config
cni.projectcalico.org/podIP: 172.20.58.4/32
cni.projectcalico.org/podIPs: 172.20.58.4/32

Lable Command Examples

- kubectl label -h
- # Update pod 'foo' with the label 'unhealthy' and the value 'true'.
  - kubectl label pods foo unhealthy=true
- # Update all pods in the namespace
  - kubectl label pods --all status=unhealthy
- # Update pod 'foo' with the label 'status' and the value 'unhealthy', overwriting any existing value.
  - kubectl label --overwrite pods foo status=unhealthy
- # Update pod 'foo' by removing a label named 'bar' if it exists (Does not require the --overwrite flag.)
  - kubectl label pods foo bar-
- Some options:
  - --overwrite=false
  - --record=false

Field Selectors (Selection Via Fields)

- Feature that let us select kubernetes resources based on the value of one or more **resource fields**.
- Supported field selectors vary by Kubernetes resource type.
- All resource types support the **metadata.name** and **metadata.namespace** fields.
- Can use the =, ==, and != operators.
- There are a limited number of fields that can be used for selection.
- Examples of field selector queries:
  - metadata.name=my-service
  - metadata.namespace!=default
  - status.phase=Pending
- Using **--field-selector** option in kubectl command.
  - kubectl get pods --field-selector status.phase=Running
  - kubectl get pods --field-selector=status.phase!=Running,spec.restartPolicy=Always #Chained selectors
  - kubectl get statefulsets,services --all-namespaces --field-selector metadata.namespace!=default #Multiple resource types

Label Selector (Selection Via Labels)

- Labels do not provide uniqueness.
- The label selector is the core grouping primitive in Kubernetes.
- Some Kubernetes objects, such as **replicaset**, also use label selectors to specify sets of pods.

```
apiVersion: apps/v1
kind: Deployment
...
spec:
 replicas: 1
 selector:
   matchLabels:
     app: web-server
```

- We can identify a set of objects by using **--selector** (-l for short) option in kubectl.
  - kubectl get deploy --selector app=nginx #Equality-based
  - kubectl get all --selector app=nginx --show-labels #Equality-based
  - kubectl get pods -l 'environment in (production),tier in (frontend)' #Set-base selector

Two Types of Selectors

- The API currently supports two types of selectors: equality-based and set-based.
- **Equality-based**
  - Equality- or inequality-based requirements allow **filtering by label keys and values.**
  - Three kinds of operators are admitted **=,==,!=.**
  - environment = production # selects all resources with key equal to environment and value equal to production
  - tier != frontend # selects all resources with key equal to tier and value distinct from frontend, and all resources with no labels with the tier key.
  - environment=production,tier!=frontend # using comma
- **Set-based**
  - Allow filtering keys according to a **set of values.**
  - Three kinds of operators are supported: in,notin and exists (only the key identifier).
  - Examples:
    - environment in (production, qa) #selects all resources with key equal to environment and value equal to production or qa.
    - tier notin (frontend, backend) #selects all resources with key equal to tier and values other than frontend and backend, and all resources with no labels with the tier key.
    - partition #selects all resources including a label with key partition; no values are checked.
    - !partition #selects all resources without a label with key partition; no values are checked.
    - partition,environment notin (qa)

Annotation

- Annotations are not used to **identify** and **select** objects.
- Annotations are used to provide additional information about object.
- Can manually set and automatically set by Kubernetes itself.
- More flexible than label about values format.
- Kubernetes reserves all labels and annotations in the **kubernetes.io** namespace for examples:
  - alpha.kubernetes.io/provided-node-ip: "10.0.0.1"
  - ingressclass.kubernetes.io/is-default-class: "true"
- Some examples of information that could be recorded in annotations:
  - Build, release, or image information like timestamps, release IDs, git branch and registry address.
  - Phone or pager numbers of persons responsible.
  - Client libraries or tools information.
- Use **kubectl annotate** command to update the annotations on one or more resources

Annotation Examples

- Show annotation of objects
  - kubectl get pod <POD NAME> -o jsonpath='{.metadata.annotations}'
  - kubectl get deployment nginx-deployment -o yaml| less
- Set annotation to object examples
  - # Update all pods in the namespace
    - kubectl annotate pods --all description='my frontend running nginx' # be careful
  - # Update pod 'foo' with the annotation 'description' and the value 'my frontend running nginx', overwriting any existing value.
    - kubectl annotate --overwrite pods foo description='my frontend running nginx'
  - # Update pod 'foo' by removing an annotation named 'description' if it exists.
    - kubectl annotate pods foo description-

Metadata as Data and Instructions