Kubernetes

Wednesday, February 23, 2022 1:01 AM

cluster → more than one node / compute / system → come together via network to perform task

in k8 we can have clusters from cloud / legacy (VM)

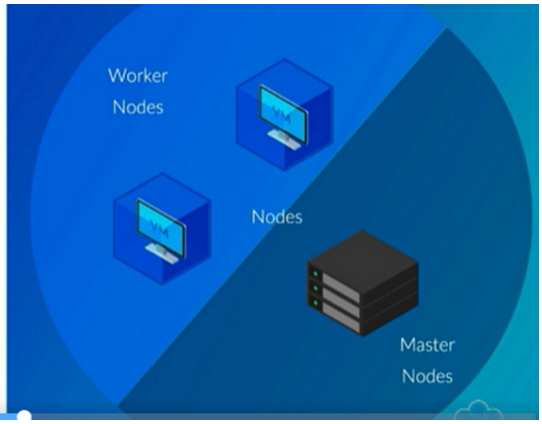
- Controlled — we control but time consuming
- Managed (version is depending on the provider)
  - AKS → azure
  - EKS → aws

other options similar to k8 is docker swarm mode / EKS / DO / AKS

- GitHub - cloudacademy/intro-to-k8s: Assets used for the production of the Introduction to Kubernetes course on Cloud Academy
- GitHub - kubernetes-sigs/metrics-server: Scalable and efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

Kubernetes Architecture

- Cluster refers to all of the machines collectively and can be thought of as the entire running system
- Nodes are the machines in the cluster
- Nodes are categorized as workers or masters
- Worker nodes include software to run containers managed by the Kubernetes control plane
- Master nodes run the control plane
- The control plane is a set of APIs and software that Kubernetes users interact with
- The APIs and software are referred

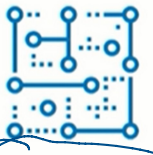


Scheduling

Control plane schedules containers onto nodes

This is not time scheduling

Scheduling decisions consider required CPU and other factors



Scheduling refers to the decision process of placing containers onto node

***

Pods → Group of containers (one or more)

↳ all containers in pod run on the same node.

Kubernetes Pods

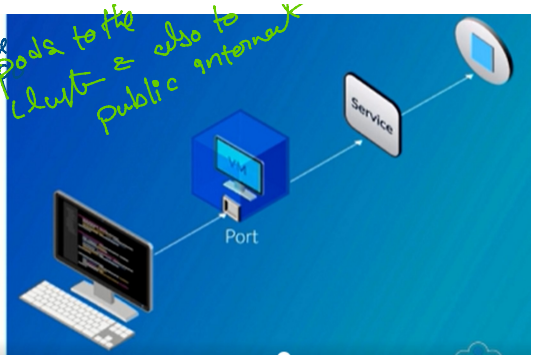
- Groups of containers
- Pods are the smallest building block in Kubernetes

Kubernetes Services → expose Pods to the network & also to public internet

Kubernetes Services

- Services define networking rules for exposing groups of Pods
  - To other Pods
  - To the public Internet

→ expose pods to the cluster & also to public internet



Kubernetes Deployments

→ control rollback & rollout of pods.

- Manage deploying configuration changes to running Pods
- Horizontal scaling

→ Kubernetes API server

Modify cluster state information by sending requests to the Kubernetes API Server

The API Server is a master component that acts as the frontend for the cluster

Interacting with Kubernetes: Method 1

REST API

It is possible but not common to work directly with the API server

You might need to if there is no Kubernetes client library for your programming language

Interacting with Kubernetes: Method 2

Client Libraries

Handles authenticating and managing individual REST API requests and responses

Kubernetes maintains official client libraries

Community-maintained libraries when no official library exists

Interacting with Kubernetes: Method 3

`kubectl`

*** Most commonly used method is this one.

Issue high-level commands that are translated into REST API calls

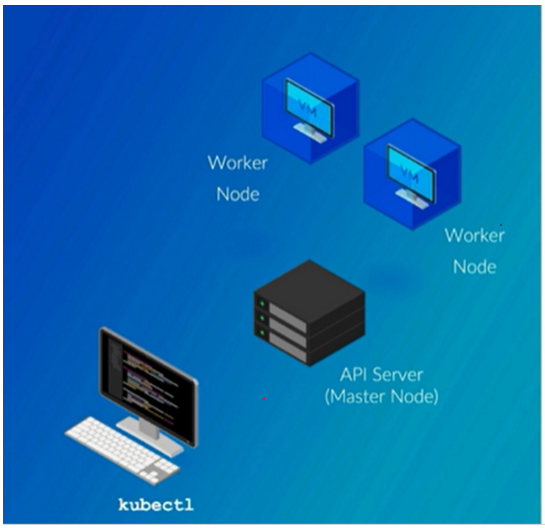
Works with local and remote clusters

kubectl

Kubernetes success correlates with `kubectl` skill

Manages all different types of Kubernetes resources, and provides debugging and introspection features

`kubectl` commands follow an easy to understand pattern



Example **`kubectl`** commands

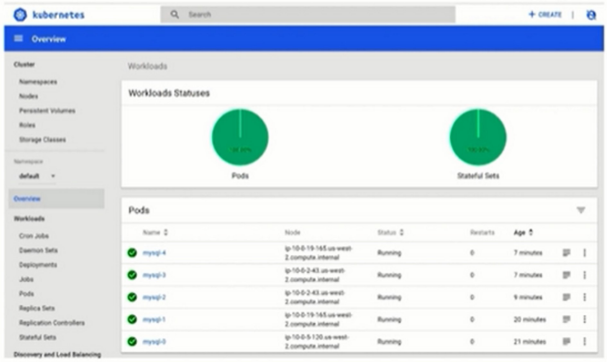
- `kubectl create` to create resources (Pods, Services, etc.)
- `kubectl delete` to delete resources
- `kubectl get` to get list of resources of a given type
  - `kubectl get pods`
- `kubectl describe` to print detailed info about a resource(s)
  - `kubectl describe pod server`
- `kubectl logs` to print container logs

→ commands can be used directly (or) you can also refer to a file called manifest (yaml) where all resources names are declared

Interacting with Kubernetes: Method 4

Web Dashboard

→ optional not all clusters will have this



What Are Pods?

- Basic building block in Kubernetes
- One or more containers in a Pod
- Pod containers all share a container network
- One IP address per Pod → k8s will take care of it.

What's in a Pod Declaration

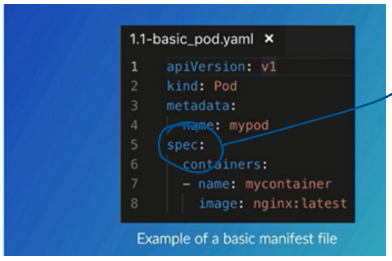
What's in a Pod Declaration

when to restart if fails.

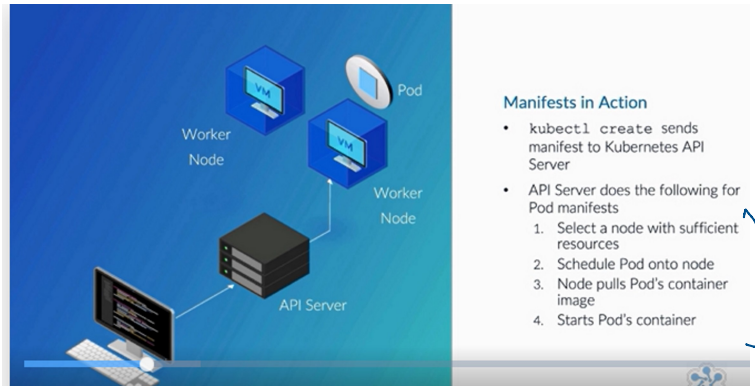
cpu/memory

Manifest Files

- Declare desired properties
- Manifests can describe all kinds of resource
- The spec contains resource-specific properties



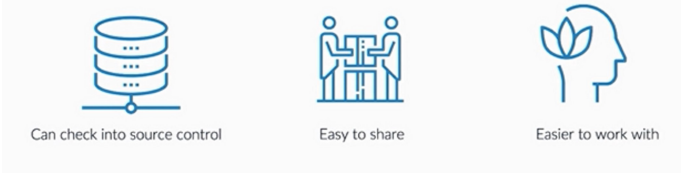
configuration specifier to a resource.



Manifests in Action

- kubectl create sends manifest to Kubernetes API Server
- API Server does the following for Pod manifests
  1. Select a node with sufficient resources
  2. Schedule Pod onto node
  3. Node pulls Pod's container image
  4. Starts Pod's container

Why Use Manifests?




```
admin@Jamess-MBP Cloud Academy % minikube start
😄  minikube v1.12.0 on Darwin 10.15.4
✨  Using the hyperkit driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🔄  Restarting existing hyperkit VM for "minikube" ...
📦  Preparing Kubernetes v1.18.3 on Docker 19.03.12 ...
🔎  Verifying Kubernetes components...
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube"
admin@Jamess-MBP Cloud Academy % kubectl get pods
No resources found in default namespace.
admin@Jamess-MBP Cloud Academy % cd src
admin@Jamess-MBP src % cat 1.1-basic_pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
admin@Jamess-MBP src %
```

apiVersion

must include a name

all manifests will have high level some fields

single container with image & name

```
admin@Jamess-MBP src % kubectl create -f 1.1-basic_pod.yaml
pod/mypod created
admin@Jamess-MBP src %
```

create using a file

describe → to describe details.

Just like docker we need to tell the k8s which port we can be accessible.

```
Namespace:    default
Priority:     0
Node:         minikube/192.168.64.2
Start Time:   Fri, 17 Jul 2020 12:09:23 -0400
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           172.17.0.3
IPs:
  IP:  172.17.0.3
Containers:
  mycontainer:
    Container ID:   docker://dd4aa6999d64bd425d97b09dbca5b93b72ec9c4314c1bad3f593f6579c
707f
    Image:          nginx:latest
    Image ID:       docker-pullable://nginx@sha256:a93c8a0b0974c967aebe868a186e5c205f4d
cb5423a56559f2f9599074bbcd
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 17 Jul 2020 12:11:48 -0400
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-7gjvg (ro)
Conditions:
```

→ port not provided.

```
admin@Jamess-MBP src % cat 1.2-port_pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
    ports:
      - containerPort: 80
admin@Jamess-MBP src %
```

→ Port

K8's cannot update Ports for a running pod.

1) we need to delete, update & recreate.

Kubectl delete pod my pod

(or)

Kubectl delete -f filename

```
Priority:     0
Node:         minikube/192.168.64.2
Start Time:   Fri, 17 Jul 2020 12:19:34 -0400
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           172.17.0.3
IPs:
  IP:  172.17.0.3
Containers:
  mycontainer:
    Container ID:   docker://78edf6a17d358ff9cb33e501c74086a9cf56a80170c66b4a0e0
5c05
    Image:          nginx:latest
    Image ID:       docker-pullable://nginx@sha256:a93c8a0b0974c967aebe868a186e5
cb5423a56559f2f9599074bbcd
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 17 Jul 2020 12:19:36 -0400
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-7gjvg (ro
Conditions:
admin@Jamess-MBP src % curl 172.17.0.3:80
```

→ will not work.

Because pod's ip is in the container network.

```
Name:         mypod
Namespace:    default
Priority:     0
Node:         minikube/192.168.64.2
Start Time:   Fri, 17 Jul 2020 12:19:
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           172.17.0.3
IPs:
```

→ key value pair

pair like region.

1) helps to get pods on that region using label.

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    app: webserver
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
    ports:
      - containerPort: 80
admin@Jamess-MBP src %
```

→ we can have multiple labels.

Its a good idea to set resource requests so that API server knows to which node to schedule based on resource availability.

to

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    app: webserver
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
    resources:
      requests:
        memory: "128Mi" # 128Mi = 128 mebibytes
        cpu: "500m"     # 500m = 500 milliCPUs (1/2 CPU)
      limits:
        memory: "128Mi"
        cpu: "500m"
    ports:
      - containerPort: 80
admin@Jamess-MBP src %
```

minimum resource to schedule pod on node

maximum resource a node can assign to pod.

Pod will be guaranteed this service.

```
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-7gjvg (ro
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-7gjvg:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-7gjvg
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
```

To overcome networking issues K8's provide service

Services

A service defines networking rules for accessing Pods in the cluster and from the internet

Use labels to select a group of Pods

Service has a fixed IP address

Distribute requests across Pods in the group

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: webserver
  name: webserver
spec:
  ports:
  - port: 80
  selector:
    app: webserver
  type: NodePort
```

selector define the labels to match the pod against.

values of pod's container port

How to expose the service.

NodePort → allocate port over this service on each node in the cluster.

By doing this you can reach this service from any node by sending request to this service.

```
admin@Jamess-MBP src % kubectl create -f 2.1-web_service.yaml
service/webserver created
admin@Jamess-MBP src % kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        6d18h
webserver    NodePort    10.100.37.149   <none>        80:32337/TCP   5s
admin@Jamess-MBP src %
```

→ k8's automatically allocated port

→ normally between 30,000 & 32,767

Private IP for each service.

not available for NodePort type

```
admin@Jamess-MBP src % kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        6d18h
webserver    NodePort    10.100.37.149   <none>        80:32337/TCP   5s
admin@Jamess-MBP src % kubectl describe service webserver
Name:                     webserver
Namespace:                default
Labels:                   app=webserver
Annotations:              <none>
Selector:                 app=webserver
Type:                     NodePort
IP:                       10.100.37.149
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32337/TCP
Endpoints:                172.17.0.3:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
admin@Jamess-MBP src %
```

If there were multiple Pods selected by the label,

```
admin@Jamess-MBP src % kubectl describe nodes | grep -i address -A 1
Addresses:
  InternalIP:  192.168.64.2
admin@Jamess-MBP src %
```

internal ip of the nodes inside the cluster

↓ private IP

```
admin@Jamess-MBP src % curl 192.168.64.2:32337
```

↳ lab vm is on the same machine as the node. so it will allow traffic.

↳ will connect to webserver service.

we can try from any node & it will connect to that service.

↳ services expose pods via a static IP address.

↳ may be changing but service will take care of it

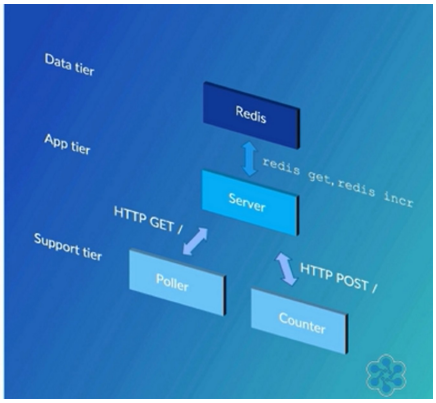
CONCLUSION

- Services expose Pods via a static IP address
- NodePort Services allows access from outside the cluster

Multi-container pods

Our Application

- A counter
- 4 containers split across 3 tiers
- Application tier is a Node.js server container
- Redis data tier storing the counter
- Poller and counter in the support tier
- All containers configured using environment variables



Kubernetes Namespaces

Namespaces separate resources according to users, environments, or applications

Role-based access control (RBAC) to secure access per Namespace

Using Namespaces is a best practice

```
admin@Jamess-MBP src % cat 3.1-namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: microservice
  labels:
    app: counter
admin@Jamess-MBP src % kubectl create -f 3.1-namespace.yaml
namespace/microservice created
```

→ namespace name.

```
admin@Jamess-MBP src % cat 3.2-multi_container.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - name: redis
      image: redis:latest
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 6379

    - name: server
      image: lrakai/microservices:server-v1
      ports:
        - containerPort: 8080
      env:
        - name: REDIS_URL
          value: redis://localhost:6379

    - name: counter
      image: lrakai/microservices:counter-v1
      env:
        - name: API_URL
          value: http://localhost:8080

    - name: poller
      image: lrakai/microservices:poller-v1
      env:
        - name: API_URL
          value: http://localhost:8080
```

→ better always mention specific image than latest

→ we are not mentioning namespace in here because if we do that pod will become less portable.

→ environment variable

→ because containers in pod share same network stack.

```
admin@Jamess-MBP src % kubectl create -f 3.2-multi_container.yaml -n microservice
pod/app created
admin@Jamess-MBP src % kubectl get -n microservice pod app
NAME   READY   STATUS              RESTARTS   AGE
app    0/4     ContainerCreating   0          50s
admin@Jamess-MBP src %
```

→ mentioning name space.

describe gives more details than get

* once running we can see (stderr, stdout) logs on the containers.

```
admin@Jamess-MBP src % kubectl logs -n microservice app counter --tail 10
Incrementing counter by 8 ...
Incrementing counter by 9 ...
Incrementing counter by 3 ...
Incrementing counter by 2 ...
Incrementing counter by 2 ...
Incrementing counter by 9 ...
Incrementing counter by 2 ...
Incrementing counter by 10 ...
Incrementing counter by 6 ...
Incrementing counter by 9 ...
admin@Jamess-MBP src % kubectl logs -n microservice app poller -f
```

→ to follow logs.

Issues with above approach.

- Kubernetes → pods are building blocks
  - It can scale only by increasing number of pods not by increasing containers inside the pods.

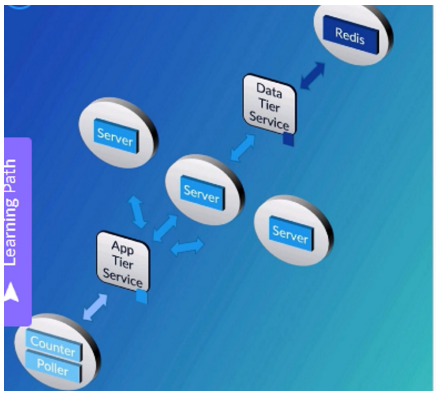
→ This is not what we want

→ They are tightly coupled.

→ They are tightly coupled.

(we should break applications to multiple pods & then scale will be ideal.

Service discovery



Compare with the previous design why this is best

Why Services?

- Supports multi-Pod design
- Provides static endpoint for each tier
- Handles Pod IP changes
- Load balancing

Service Discovery Mechanisms

Environment Variables	DNS
• Services address automatically injected in containers	• DNS records automatically created in cluster's DNS
• Environment variables follow naming conventions based on service name	• Containers automatically configured to query cluster DNS

```
admin@Jamess-MBP src % kubectl create -f 4.1-namespace.yaml
namespace/service-discovery created
admin@Jamess-MBP src % cat
```

→ Creat new namespace.

```
admin@Jamess-MBP src % cat 4.2-data_tier.yaml
apiVersion: v1
kind: Service
metadata:
  name: data-tier
  labels:
    app: microservices
spec:
  ports:
  - port: 6379
    protocol: TCP # default
    name: redis # optional when only 1 port
  selector:
    tier: data
  type: ClusterIP # default
---
apiVersion: v1
kind: Pod
metadata:
  name: data-tier
  labels:
    app: microservices
    tier: data
spec:
  containers:
  - name: redis
```

→ name same for service & pod

→ naming port is mandatory if we have multiple ports. we can create as many resources as we want seperated by 3 - - -

selector tier → data

This is how linking happens.

→ label tier : data

type : cluster ip → default type, creates virtual ip inside cluster for internal access only.

```
in@Jamess-MBP src % kubectl create -f 4.2-data_tier.yaml -n service-discovery
```

Cresources are created in the order they are listed in the file.

Resources are created in the order they are listed in the file.

```
admin@Jamess-MBP src % kubectl describe service -n service-discovery data-tier
Name:              data-tier
Namespace:         service-discovery
Labels:            app=microservices
Annotations:       <none>
Selector:          tier=data
Type:              ClusterIP
IP:                10.98.110.202
Port:              redis  6379/TCP
TargetPort:        6379/TCP
Endpoints:         172.17.0.3:6379
Session Affinity:  None
Events:            <none>
admin@Jamess-MBP src %
```

→ cluster IP

```
    tier: app
apiVersion: v1
kind: Pod
metadata:
  name: app-tier
  labels:
    app: microservices
    tier: app
spec:
  containers:
    - name: server
      image: lrakai/microservices:server-v1
      ports:
        - containerPort: 8080
      env:
        - name: REDIS_URL
          # Environment variable service discovery
          # Naming pattern:
          #   IP address: <all_caps_service_name>_SERVICE_HOST
          #   Port: <all_caps_service_name>_SERVICE_PORT
          #   Named Port: <all_caps_service_name>_SERVICE_PORT_<all_caps_port_name>
          value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_TIER_SERVICE_PORT_REDIS)
          # In multi-container example value was
          # value: redis://localhost:6379
in@Jamess-MBP src %
```

environment variables are auto created by the K8s we just need naming convention proper.

ex:- NAME_SERVICE_HOST

↳ data-tier

↳ hyphen replaced by underscore

→ ALL CAPS.

$(DATA_TIER_SERVICE_HOST)

$(DATA_TIER_SERVICE_PORT_REDIS)

↳ Port name.

To use environment variables the service should be created first before using/defining.

service should also be in same namespace where the env variables are used

```
admin@Jamess-MBP src % cat 4.4-support_tier.yaml
apiVersion: v1
kind: Pod
metadata:
  name: support-tier
  labels:
    app: microservices
    tier: support
spec:
  containers:

    - name: counter
      image: lrakai/microservices:counter-v1
      env:
        - name: API_URL
          # DNS for service discovery
          # Naming pattern:
          #   IP address: <service_name>.<service_namespace>
          #   Port: needs to be extracted from SRV DNS record
```

→ support tier.

↳ K8s create DNS for every service.

```
    env:
      - name: API_URL
        # DNS for service discovery
        # Naming pattern:
        #   IP address: <service_name>.<service_namespace>
        #   Port: needs to be extracted from SRV DNS record
        value: http://app-tier.service-discovery:8080
  - name: poller
    image: lrakai/microservices:poller-v1
    env:
      - name: API_URL
        # omit namespace to only search in the same namespace
```

k8s creates DNS for every service.

if in same namespace no need to mention.

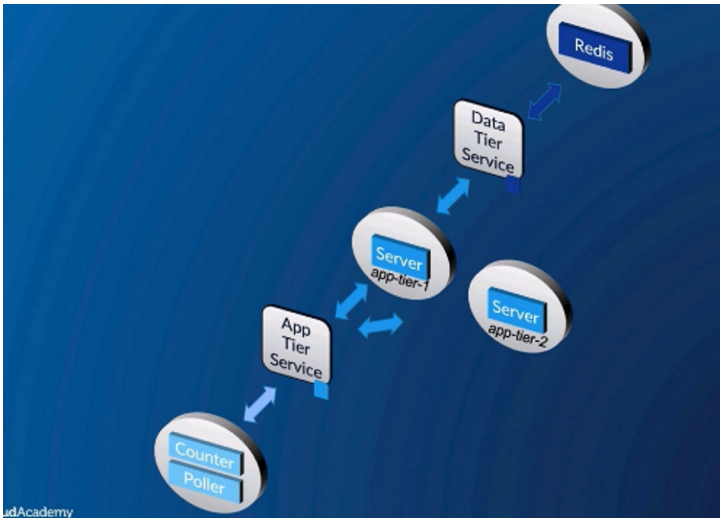
```
  - name: poller
    image: lrakai/microservices:poller-v1
    env:
      - name: API_URL
        # omit namespace to only search in the same namespace
        value: http://app-tier:$(APP_TIER_SERVICE_PORT)
in@Jamess-MBP src %
```

continuation hard code (or) use env. variable.

namespace omitted

DNS does not require underscore or all caps.

```
admin@Jamess-MBP src % kubectl create -f 4.4-support_tier.yaml -n service-discover
pod/support-tier created
in@Jamess-MBP src % kubectl get pods -n service-discovery
E              READY   STATUS    RESTARTS   AGE
-tier          1/1     Running   0          4m3s
a-tier         1/1     Running   0          12m
port-tier      2/2     Running   0          11s
```



lelmn.

→ Till now we created pods directly. normally/ideally we should not creat Pod directly but instead use the higher level to creat pods for example deployment.

Deployments

Represent multiple replicas of a Pod

Describe a desired state that Kubernetes needs to achieve

Deployment Controller master component converges actual state to the desired state

We'll replace the individual pods with deployments that manage the pods for us.

Deployment is a template to create pods.

↳ used to create replicas (↳ are copies of pods & scaled.)

Comparing pod & deployment

no of pods

same

→ instead of v1 its apps/v1 and is deployment

→ spec → contains deployment specific settings and also a pod template with exactly same as pod

To group pods during deployments.

does not need a name for pod here because to create unique name for each pod.

```
admin@Jamess-MBP src % kubectl create -n deployments -f 5.2-data_tier.yaml -f 5.3-app_tier.yaml -f 5.4-support_tier.yaml
```

↳ using multiple files at a time.

```
admin@Jamess-MBP src % kubectl get -n deployments deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
app-tier       1/1     1            1           76s
data-tier      1/1     1            1           76s
support-tier   1/1     1            1           76s
admin@Jamess-MBP src %
```

```
admin@Jamess-MBP src % kubectl -n deployments get pods
NAME                           READY   STATUS    RESTARTS   AGE
app-tier-74cd4c68c9-z99h4      1/1     Running   0          2m24s
data-tier-8646dd765b-k6h98     1/1     Running   0          2m24s
support-tier-997bc57fb-c579t   2/2     Running   0          2m24s
admin@Jamess-MBP src %
```

↳ counts & poller

We can use scale command to scale the replicas.

```
admin@Jamess-MBP src % kubectl scale -n deployments deployments support-tier --replicas=5
deployment.apps/support-tier scaled
admin@Jamess-MBP src %
```

```
admin@Jamess-MBP src % kubectl -n deployments get pods
NAME                           READY   STATUS    RESTARTS   AGE
app-tier-74cd4c68c9-z99h4      1/1     Running   0          4m49s
data-tier-8646dd765b-k6h98     1/1     Running   0          4m49s
support-tier-997bc57fb-42s98   2/2     Running   0          54s
```

containers poller & counter remain same per pod

```
admin@Jamess-MBP src % kubectl -n deployments get pods
NAME                              READY   STATUS    RESTARTS   AGE
app-tier-74cd4c68c9-z99h4         1/1     Running   0          4m49s
data-tier-8646dd765b-k6h98        1/1     Running   0          4m49s
support-tier-997bc57fb-42s98      2/2     Running   0          54s
support-tier-997bc57fb-c579t      2/2     Running   0          4m49s
support-tier-997bc57fb-l91kc      2/2     Running   0          54s
support-tier-997bc57fb-lp56v      2/2     Running   0          54s
support-tier-997bc57fb-mtfrf      2/2     Running   0          54s
```

→ notice containers pod/s & counts remain same per pod because

replicas replicate pods not individual containers inside the pod. ←

Deployments ensure that specified number of replicated pods keep on running

1) Test this by deleting pods.

```
admin@Jamess-MBP src % kubectl delete -n deployments pods support-tier-997bc57fb-mtf
rf support-tier-997bc57fb-lp56v support-tier-997bc57fb-l91kc
pod "support-tier-997bc57fb-mtfrf" deleted
pod "support-tier-997bc57fb-lp56v" deleted
pod "support-tier-997bc57fb-l91kc" deleted
admin@Jamess-MBP src % watch -n 1 kubectl -n deployments get pods
```

→ watch every 1 sec update output.
↳ linux command

```
Every 1.0s: kubectl -n deployments get pods

NAME                              READY   STATUS        RESTARTS   AGE
app-tier-74cd4c68c9-h7hvf         1/1     Running       0          9m37s
data-tier-8646dd765b-wjmjd        1/1     Running       0          9m37s
support-tier-997bc57fb-7597k      2/2     Running       0          3m8s
support-tier-997bc57fb-79gsl      2/2     Running       0          7m59s
support-tier-997bc57fb-8wjnf      2/2     Terminating   0          7m59s
support-tier-997bc57fb-b2f9f      2/2     Terminating   0          3m8s
support-tier-997bc57fb-b8c4q      2/2     Running       0          19s
support-tier-997bc57fb-fggcx      2/2     Running       0          19s
support-tier-997bc57fb-hprh6      2/2     Terminating   0          3m8s
support-tier-997bc57fb-k8kfd      2/2     Running       0          19s
```

→ k8's automatically make sure 5 replicas are running.

```
Every 1.0s: kubectl -n deployments get pods

NAME                              READY   STATUS    RESTARTS   AGE
app-tier-74cd4c68c9-h7hvf         1/1     Running   0          10m
data-tier-8646dd765b-wjmjd        1/1     Running   0          10m
support-tier-997bc57fb-7597k      2/2     Running   0          3m36s
support-tier-997bc57fb-79gsl      2/2     Running   0          8m27s
support-tier-997bc57fb-b8c4q      2/2     Running   0          47s
support-tier-997bc57fb-fggcx      2/2     Running   0          47s
support-tier-997bc57fb-k8kfd      2/2     Running   0          47s
```

```
admin@Jamess-MBP src % kubectl scale -n deployments deployment app-tier --replicas=5

deployment.apps/app-tier scaled
deployment.apps/app-tier scaled
admin@Jamess-MBP src % kubectl -n deployments get pods
NAME                              READY   STATUS    RESTARTS   AGE
app-tier-74cd4c68c9-kd7z4         1/1     Running   0          13s
app-tier-74cd4c68c9-qnrrh         1/1     Running   0          13s
app-tier-74cd4c68c9-rkbxh         1/1     Running   0          13s
app-tier-74cd4c68c9-thh25         1/1     Running   0          13s
app-tier-74cd4c68c9-z99h4         1/1     Running   0          9m48s
data-tier-8646dd765b-k6h98        1/1     Running   0          9m48s
support-tier-997bc57fb-2lptf      2/2     Running   0          3m4s
support-tier-997bc57fb-42s98      2/2     Running   0          5m53s
support-tier-997bc57fb-c579t      2/2     Running   0          9m48s
support-tier-997bc57fb-gpmj5      2/2     Running   0          3m3s
support-tier-997bc57fb-zfcrn      2/2     Running   0          3m3s
admin@Jamess-MBP src %
```

```
admin@Jamess-MBP src % kubectl describe -n deployments service app-tier
Name:              app-tier
Namespace:         deployments
Labels:            app=microservices
Annotations:       <none>
Selector:          tier=app
Type:              ClusterIP
IP:                10.111.185.244
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
Endpoints:         172.17.0.10:8080,172.17.0.11:8080,172.17.0.12:8080 + 2 more...
Session Affinity:  None
Events:            <none>
```

→ all 5 replicas are tracked & load balanced in the service.

CONCLUSION

- Deployments to manage Pods in each tier
- Kubernetes ensures actual state matches desired state
- `kubectl scale` to scale number of replicas
- Services seamlessly support scaling
- Scaling is best with stateless Pods

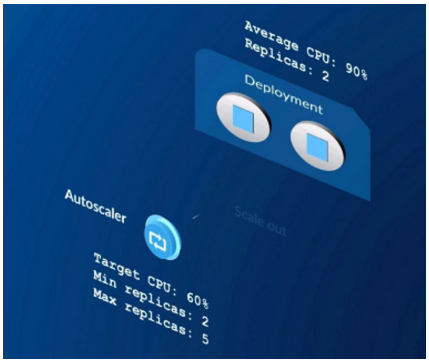
→ even there is any failure(or) manual deletion it ensures to bring back to desired state

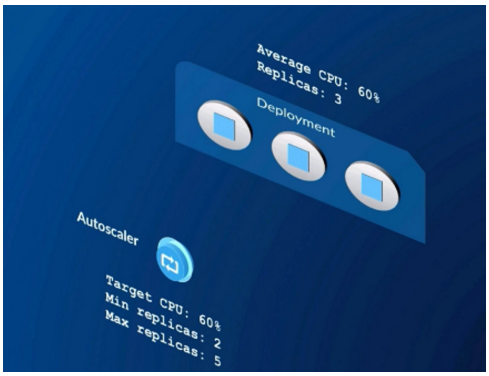
→ we can't scale the data tier in above example.

Autoscaling Deployments

- Scale automatically based on CPU utilization (or custom metrics)
- Set target CPU along with min and max replicas
- Target CPU is expressed as a percentage of the Pod's CPU request

→ If no cpu request is set autoscaling will not take any action.

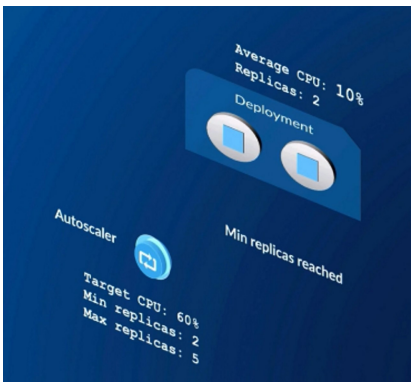




Creates more replicas if average usage exceeds target.

→ Dont creat more replicas than max set

→ Dont decrease less than min set replicas.



Autoscaling depends on metrics being collected on the cluste.

Metrics

- Autoscaling depends on metrics being collected
- Metrics Server is one solution for collecting metrics

→ we have to get metrics server up & running before using autoscaling.

Metrics server created.

Metrics Server is one solution for collecting metrics

Several manifest files are used to deploy Metrics Server (https://github.com/kubernetes-sigs/metrics-server)

autoscaling

→ all folder under metrics server dirs are being created.

```
admin@Jamess-MBP src % kubectl apply -f metrics-server/
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
serviceaccount/metrics-server created
deployment.apps/metrics-server created
service/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
admin@Jamess-MBP src % _
```

```
admin@Jamess-MBP src % kubectl top pods -n deployments
NAME                          CPU(cores)   MEMORY(bytes)
app-tier-74cd4c68c9-h74qz     2m           45Mi
app-tier-74cd4c68c9-hcg26     1m           45Mi
app-tier-74cd4c68c9-p24sr     2m           45Mi
app-tier-74cd4c68c9-pstcr     2m           45Mi
app-tier-74cd4c68c9-sfxgm     2m           46Mi
data-tier-8646dd765b-7f457    2m           2Mi
support-tier-997bc57fb-2dtq2  10m          2Mi
support-tier-997bc57fb-dvlcv  11m          2Mi
support-tier-997bc57fb-gtd2d  12m          2Mi
support-tier-997bc57fb-kvvtr  11m          2Mi
support-tier-997bc57fb-mdlrp  12m          2Mi
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

(m) → milli CPU

1000 Mi = 1 CPU

previous

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-tier
  labels:
    app: microservices
    tier: app
spec:
  replicas: 1
  selector:
    matchLabels:
      tier: app
  template:
    metadata:
      labels:
        app: microservices
        tier: app
    spec:
      containers:
      - name: server
        image: lrakai/microservices:server-v1
        ports:
          - containerPort: 8080


        env:
          - name: REDIS_URL
            # Environment variable service discovery
            # Naming pattern:
            #   IP address: <all_caps_service_name>_SERVICE_H
            #   Port: <all_caps_service_name>_SERVICE_PORT
            #   Named Port: <all_caps_service_name>_SERVICE_P
            value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_T
            # In multi-container example value was
            # value: redis://localhost:6379
admin@Jamess-MBP src %
```

updated

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-tier
  labels:
    app: microservices
    tier: app
spec:
  replicas: 5
  selector:
    matchLabels:
      tier: app
  template:
    metadata:
      labels:
        app: microservices
        tier: app
    spec:
      containers:
      - name: server
        image: lrakai/microservices:server-v1
        ports:
          - containerPort: 8080
        resources:
          requests:
            cpu: 20m # 20 milliCPU / 0.02 CPU
        env:
          - name: REDIS_URL
            # Environment variable service discovery
            # Naming pattern:
            #   IP address: <all_caps_service_name>_SERVICE_H
            #   Port: <all_caps_service_name>_SERVICE_PORT
            #   Named Port: <all_caps_service_name>_SERVICE_P
            value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_T
            # In multi-container example value was
            # value: redis://localhost:6379
```

→ k8s will deploy pods on to each node having min of of 20 milli CPU

```
admin@Jamess-MBP src % kubectl create -f 6.1-app_tier_cpu_request.yaml -n deployments
Error from server (AlreadyExists): error when creating "6.1-app_tier_cpu_request.yaml": services "app-tier" already exists
Error from server (AlreadyExists): error when creating "6.1-app_tier_cpu_request.yaml": deployments.apps "app-tier" already exists
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

→ we can delet & recreat

(oo)

→ use apply command which updaty the deployment.

* (KAD) administrator course will have ideas on diff between apply & creat

```
admin@Jamess-MBP src % kubectl get -n deployments deployments app-tier
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
app-tier   5/5     5            5           13m
admin@Jamess-MBP src %
```

```
admin@Jamess-MBP src % cat 6.2-autoscale.yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: app-tier
  labels:
    app: microservices
    tier: app
spec:
  maxReplicas: 5
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app-tier
  targetCPUUtilizationPercentage: 70

# Equivalent to
# kubectl autoscale deployment app-tier --max=5 --min=1 --cpu-percent=70
admin@Jamess-MBP src % _
```

→ kind for autoscaling

→ lower & upper limit

70%

version

command for above manifest

→ declared.

```
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
app-tier       1/1     1            1           19m
data-tier      1/1     1            1           19m
support-tier   5/5     5            5           19m
```

```
admin@Jamess-MBP src % kubectl api-resources
```

→ gives all resources with their shortnames.

```
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
customresourcedefinitions         crd,crds     apiextensions.k8s.io           false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io         false        APIService
controllerrevisions                            apps                           true         ControllerRevision
daemonsets                        ds           apps                           true         DaemonSet
deployments                       deploy       apps                           true         Deployment
replicasets                       rs           apps                           true         ReplicaSet
statefulsets                      sts          apps                           true         StatefulSet
tokenreviews                                   authentication.k8s.io          false        TokenReview
localsubjectaccessreviews                      authorization.k8s.io           true         LocalSubjectAccessReview
selfsubjectaccessreviews                       authorization.k8s.io           false        SelfSubjectAccessReview
selfsubjectrulesreviews                        authorization.k8s.io           false        SelfSubjectRulesReview
subjectaccessreviews                           authorization.k8s.io           false        SubjectAccessReview
horizontalpodautoscalers          hpa          autoscaling                    true         HorizontalPodAutoscaler
cronjobs                          cj           batch                          true         CronJob
jobs                                           batch                          true         Job
certificatesigningrequests        csr          certificates.k8s.io            false        CertificateSigningRequest
leases                                         coordination.k8s.io            true         Lease
endpointslices                                 discovery.k8s.io               true         EndpointSlice
events                            ev           events.k8s.io                  true         Event
```

→ shortname

```
admin@Jamess-MBP src % kubectl describe -n deployments hpa
Name:                                                  app-tier
Namespace:                                             deployments
Labels:                                                app=microservices
                                                       tier=app
Annotations:                                           <none>
CreationTimestamp:                                     Tue, 04 Aug 2020 12:10:16 +1200
Reference:                                             Deployment/app-tier
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  40% (8m) / 70%
Min replicas:                                          1
Max replicas:                                          5
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from cpu resource utilization (percent
age of request)
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable range
Events:
  Type    Reason             Age   From                       Message
  ----    ------             ----  ----                       -------
  Normal  SuccessfulRescale  107s  horizontal-pod-autoscaler  New size: 2; reason: All metrics below target
  Normal  SuccessfulRescale  77s   horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
admin@Jamess-MBP src % _
```

```
admin@Jamess-MBP src % kubectl get -n deployments hpa
NAME       REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
app-tier   Deployment/app-tier     40%/70%   1         5         1          7m47s
admin@Jamess-MBP src % _
```

```
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl edit -n deployments hpa_
```

(we can update manifest file & use apply)

(or) (use edit command)

opens in VI & we can edit the config like replicas if we want.

```
        f:apiVersion: {}
        f:kind: {}
        f:name: {}
      f:targetCPUUtilizationPercentage: {}
  manager: kubectl
  operation: Update
  time: "2020-08-04T00:10:16Z"
- apiVersion: autoscaling/v1
  fieldsType: FieldsV1
  fieldsV1:
    f:metadata:
      f:annotations:
        .: {}
        f:autoscaling.alpha.kubernetes.io/conditions: {}
        f:autoscaling.alpha.kubernetes.io/current-metrics: {}
    f:status:
      f:currentCPUUtilizationPercentage: {}
      f:currentReplicas: {}
      f:desiredReplicas: {}
      f:lastScaleTime: {}
  manager: kube-controller-manager
  operation: Update
  time: "2020-08-04T00:18:07Z"
name: app-tier
namespace: deployments
resourceVersion: "4191"
selfLink: /apis/autoscaling/v1/namespaces/deployments/horizontalpodautoscalers/app-tier
uid: d8118aba-f4aa-432d-b5cc-67d6e8535e62
spec:
maxReplicas: 5
minReplicas: 2
scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: app-tier
targetCPUUtilizationPercentage: 70
status:
```

In VI

```
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl edit -n deployments hpa
horizontalpodautoscaler.autoscaling/app-tier edited
```

```
Every 1.0s: kubectl get -n deployments deployment

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
app-tier       2/2     2            2           24m
data-tier      1/1     1            1           24m
support-tier   5/5     5            5           24m
```

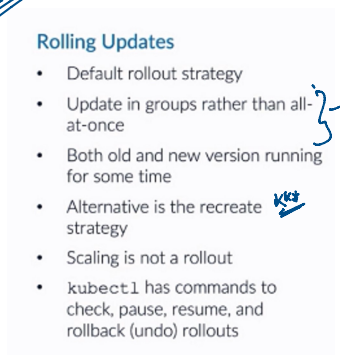
metrics server.

Pre requisitives for autoscaling.

Rolling

0

Strategies!



some thing like some pods to new some old & all new at end.

Rollouts are compatible.

check, pause, resume, and
rollback (undo) rollouts

o autoscaling & rollouts are compatiable.

```
admin@Jamess-MBP src % kubectl delete -n deployments hpa app-tier
horizontalpodautoscaler.autoscaling "app-tier" deleted
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl edit -n deployments deployment app-ti_
```

```
            f:message: {}
            f:reason: {}
            f:status: {}
            f:type: {}
        f:observedGeneration: {}
        f:readyReplicas: {}
        f:replicas: {}
        f:updatedReplicas: {}
    manager: kube-controller-manager
    operation: Update
    time: "2020-08-04T03:01:31Z"
  name: app-tier
  namespace: deployments
  resourceVersion: "1711"
  selfLink: /apis/apps/v1/namespaces/deployments/deployments/ap
  uid: 0350bfcd-9bdf-4960-95a4-e43326b691a3
spec:
  progressDeadlineSeconds: 600
    cas: 10
    ionHistoryLimit: 10
    tor:
    chLabels:
    ier: app
    gy:
    lingUpdate:
    axSurge: 25%
    axUnavailable: 25%
    e: RollingUpdate
    ate:
    data:
    reationTimestamp: null
    abels:
     app: microservices
     tier: app
    ::
    ntainers:
     env:
```

→ change replicas from 2 to "10"

```
  resourceVersion: "1711"
  selfLink: /apis/apps/v1/namespaces/deployments/deployments/app-tier
  uid: 0350bfcd-9bdf-4960-95a4-e43326b691a3
spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      tier: app
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: microservices
        tier: app
    spec:
      containers:
      - env:
        - name: REDIS_URL
          value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_TIER_SERVICE_PORT_REDIS)
        image: lrakai/microservices:server-v1
        imagePullPolicy: IfNotPresent
        name: server
        ports:
        - containerPort: 8080
          protocol: TCP
        resources:
          requests:
            cpu: 20m
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
```

} delete the resources req. to avoid any potential problems which scheduling replicas. If all 10 of the CPU requests can be satisfied.

```
admin@Jamess-MBP src % kubectl edit -n deployments deployment app-tier
deployment.apps/app-tier edited
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % watch -n 1 kubectl get -n deployments deployment app-tier_
```

```
Every 1.0s: kubectl get -n deployments deployment app-tier

NAME       READY   UP-TO-DATE   AVAILABLE   AGE
app-tier   10/10   10           10          8m25s
```

```
dmin@Jamess-MBP src % kubectl edit -n deployments deployment app-tier
```

```
    tier: app
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
```

} these are server added default values for the deployment strategy.

type is rollingupdate

maxUnavailable → how many old pods can be deleted

pods to be ready.

↳ type is rollingupdate

maxUnavailable → how many old pods can be deleted without waiting for new pods to be ready.

maxSurge → how many replicas over the desired total are allowed during a rollout.

```
- name: REDIS_URL
  value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_TIER_SERVICE_PORT_REDIS)
image: lrakai/microservices:server-v1
imagePullPolicy: IfNotPresent
name: cloudacademy
ports:
- containerPort: 8080
```

→ changed name from server to cloudacademy.
↳ non-functional change.

```
admin@Jamess-MBP src % kubectl edit -n deployments deployment app-tier
deployment.apps/app-tier edited
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl rollout -n deployments status deployment app-tier
Waiting for deployment "app-tier" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 2 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 2 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 2 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 1 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 1 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 1 old replicas are pending termination...
for deployment "app-tier" rollout to finish: 8 of 10 updated replicas are available...
for deployment "app-tier" rollout to finish: 9 of 10 updated replicas are available...
nt "app-tier" successfully rolled out
mess-MBP src % _
```

new replicas in

old replicas out.

Learning Path

```
admin@Jamess-MBP src % kubectl edit -n deployments deployment app-tier
deployment.apps/app-tier edited
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl rollout -n deployments pause deployment app-tier
deployment.apps/app-tier paused
admin@Jamess-MBP src % _
```

opened in two terminals to watch the deployment while we run pause command.

```
Waiting for deployment "app-tier" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 6 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 7 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 8 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 9 out of 10 new replicas have been updated...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment spec update to be observed...
Waiting for deployment spec update to be observed...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
```

FEEDBACK

replicas which rollout before pause continue to become ready

where as the other which not yet started will not after pause.

```
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment spec update to be observed...
Waiting for deployment spec update to be observed...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
Waiting for deployment "app-tier" rollout to finish: 3 old replicas are pending termination...
deployment "app-tier" successfully rolled out
admin@Jamess-MBP src %
"Jeremys-MacBook-Pro.l" 15:08 04-Aug-20
```

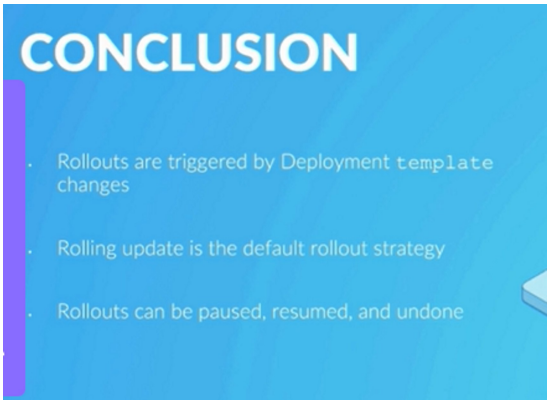
```
admin@Jamess-MBP src % kubectl rollout -n deployments undo deployment app-tier
deployment.apps/app-tier rolled back
admin@Jamess-MBP src % _
```

to rollback the deployment.

```
admin@Jamess-MBP src % kubectl rollout history -n deployments deployment app-tier
deployment.apps/app-tier
REVISION  CHANGE-CAUSE
          <none>
          <none>
          <none>
          <none>

mess-MBP src % _
```

↳ to get all versions of deployments we can use & rollback to any version.



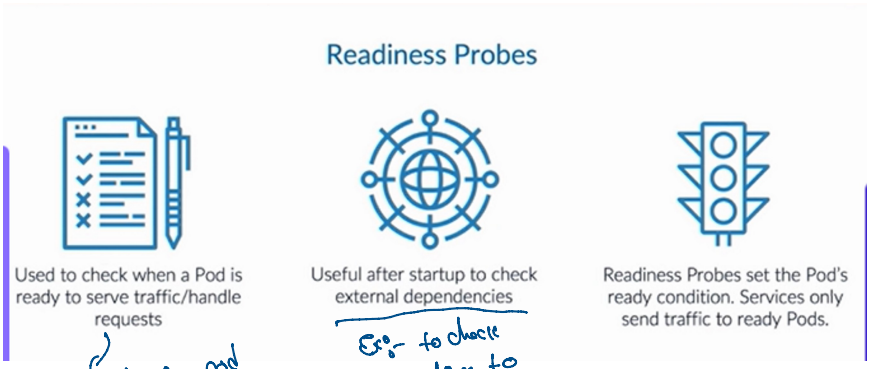
Kubernetes assumes that created containers are immediately ready and the rollout should continue. But this does not work in all cases. We may need to wait for the web server to accept connections. So here's another scenario. Considering an application using a relational database, the containers may start but it will fail until a database and tables are created.

These scenarios must be considered to build reliable applications. This is where probes in init containers come into the picture. So we'll take a look at integrating probes and init containers in our next two lessons.

b

- Rollout depend on container. K8 rollout thinking that container is ready but there are cases where we have to wait like waiting for database.
- another scenario is dead locked container & wanted it to be restarted.

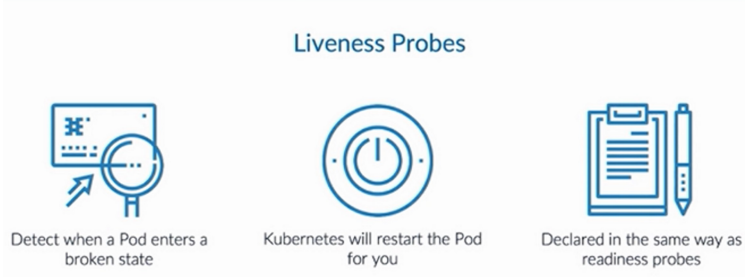
Probes are sometimes referred to healthchecks.



to know if the pod is ready to handle requests

Eg:- to check connection to CMS is working otherwise we can't use token_adapter.

Liveness Probes



- Readeness Probe → stop serving traffic.
- Liveness probe → restarts pod.

Declaring Probes

Probes can be declared in a Pod's containers

All container probes must pass for the Pod to pass

Probe actions can be a command that runs in the container, an HTTP GET request, or opening a TCP socket

By default probes check containers every 10 seconds

We will add readiness and liveness probes to our application

Data Tier (Redis)

- Liveness: Open TCP socket
- Readiness: `redis-cli ping` command

App Tier (Server)

- Liveness: HTTP GET /probe/liveness
- Readiness: HTTP GET /probe/readines

Learning Path

```
  type: ClusterIP # default
---
apiVersion: apps/v1 # apps API group
kind: Deployment
metadata:
  name: data-tier
  labels:
    app: microservices
    tier: data
spec:
  replicas: 1
  selector:
    matchLabels:
      tier: data
  template:
    metadata:
      labels:
        app: microservices
        tier: data
    spec: # Pod spec
      containers:
      - name: redis
        image: redis:latest
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 6379
```

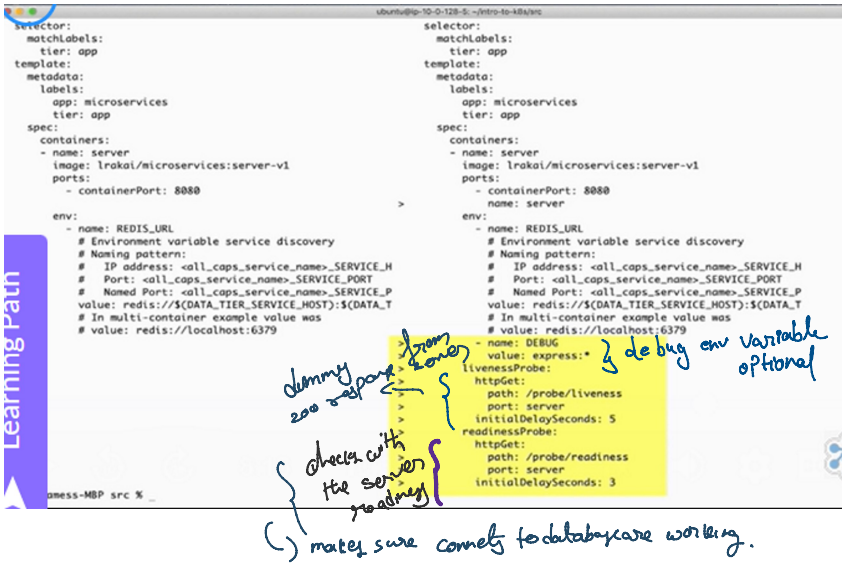
```
  type: ClusterIP # default
---
apiVersion: apps/v1 # apps API group
kind: Deployment
metadata:
  name: data-tier
  labels:
    app: microservices
    tier: data
spec:
  replicas: 1
  selector:
    matchLabels:
      tier: data
  template:
    metadata:
      labels:
        app: microservices
        tier: data
    spec: # Pod spec
      containers:
      - name: redis
        image: redis:latest
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 6379
>         name: redis
>       livenessProbe:
>         tcpSocket:
>           port: redis # named port
>         initialDelaySeconds: 15
>       readinessProbe:
>         exec:
>           command:
>           - redis-cli
>           - ping
>         initialDelaySeconds: 5
```

named port instead of Port to avoid changing

commands as a list

- Setting the initial delay seconds, we give the redis server an adequate time to start.
- Given the consequences of failing a liveness probe is going to be restarting a Pod. It's generally advisable to have the liveness probe at a high delay than the readiness probe.
- By default three sequential probes need to fail before a probe is marked as failed, so that we have some buffer. Kubernetes won't immediately restart the Pod the first time the probe fails, but we can configure it that way if we need to.
- The particular delay depends on our application and how long it reasonably requires to start up. Five seconds should be more than enough to start checking readiness. And by default, we only need to pass a single probe before any traffic is sent to the Pod. Having the readiness initial delay too high will prevent Pods that are able to handle traffic from receiving any.

```
admin@Jamess-MBP src % kubectl create -f 7.2-data_tier.yaml -n probes
service/data-tier created
deployment.apps/data-tier created
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl get deployments -n probes -w
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
data-tier   0/1     1            0           2s
data-tier   1/1     1            1           14s
```

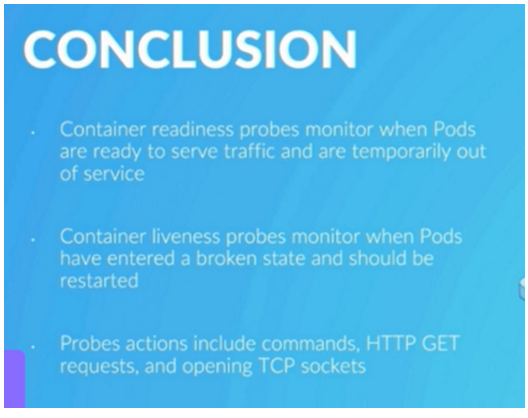



```
admin@Jamess-MBP src % kubectl create -f 7.3-app_tier.yaml -n probes
service/app-tier created
deployment.apps/app-tier created
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl get deployments -n probes app-tier -w
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
app-tier   1/1     1            1           25s


^Cadmin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl get -n probes pods
NAME                        READY   STATUS    RESTARTS   AGE
app-tier-74c849f7c9-wdjn7   1/1     Running   0          65s
data-tier-7557d44554-l6g6h  1/1     Running   0          3m32s
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

```
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl logs -n probes app-tier-74c849f7c9-wdjn7 | cut -d' ' -f5,8-11_
```

```
03:33:52 urlencodedParser  : /probe/liveness
03:33:56 dispatching GET /probe/readiness
03:33:56 query  : /probe/readiness
03:33:56 expressInit  : /probe/readiness
03:33:56 urlencodedParser  : /probe/readiness
03:34:02 dispatching GET /probe/liveness
03:34:02 query  : /probe/liveness
03:34:02 expressInit  : /probe/liveness
03:34:02 urlencodedParser  : /probe/liveness
03:34:06 dispatching GET /probe/readiness
03:34:06 query  : /probe/readiness
03:34:06 expressInit  : /probe/readiness
03:34:06 urlencodedParser  : /probe/readiness
03:34:12 dispatching GET /probe/liveness
03:34:12 query  : /probe/liveness
03:34:12 expressInit  : /probe/liveness
03:34:12 urlencodedParser  : /probe/liveness
03:34:16 dispatching GET /probe/readiness
03:34:16 query  : /probe/readiness
```



→ If we want to test (or) prepare things before container starts & we can do using init.

Motivation for Init Containers

Sometimes you need to wait for a service, downloads, dynamic or decisions before starting a Pod's containers

Prefer to separate initialization wait logic from the container image

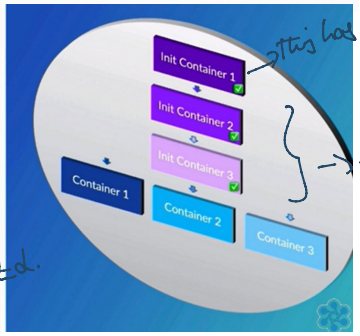
Initialization is tightly coupled to the main application (belongs in the Pod)

Init containers allow you to run initialization tasks before starting the main container(s)

Init Containers

- Pods can declare any number of init containers
- Init containers run in order and to completion
- Use their own images
- Easy way to block or delay starting an application
- Run every time a Pod is created

Restarted.

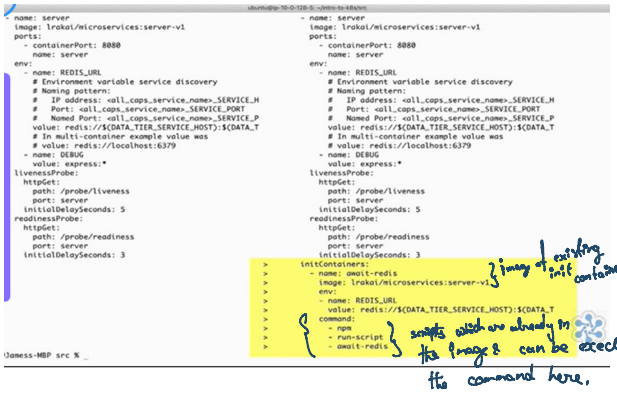


this has to complete before going to next one in the order.

→ no readiness probe for these.

Add an init container to our app tier that will wait for Redis before starting any application servers.

We'll see init containers have the same fields as regular containers except for `readinessProbe`.



```
- name: server
  image: lrakai/microservices:server-v1
  ports:
    - containerPort: 8080
      name: server
  env:
    - name: REDIS_URL
      # Environment variable service discovery
      # Naming pattern:
      #   IP address: <all_caps_service_name>_SERVICE_H
      #   Port: <all_caps_service_name>_SERVICE_PORT
      #   Named Port: <all_caps_service_name>_SERVICE_P
      value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_T
      # In multi-container example value was
      # value: redis://localhost:6379
    - name: DEBUG
      value: express:*
  livenessProbe:
    httpGet:
      path: /probe/liveness
      port: server
    initialDelaySeconds: 5
  readinessProbe:
    httpGet:
      path: /probe/readiness
      port: server
    initialDelaySeconds: 3
```

```
- name: server
  image: lrakai/microservices:server-v1
  ports:
    - containerPort: 8080
      name: server
  env:
    - name: REDIS_URL
      # Environment variable service discovery
      # Naming pattern:
      #   IP address: <all_caps_service_name>_SERVICE_H
      #   Port: <all_caps_service_name>_SERVICE_PORT
      #   Named Port: <all_caps_service_name>_SERVICE_P
      value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_T
      # In multi-container example value was
      # value: redis://localhost:6379
    - name: DEBUG
      value: express:*
  livenessProbe:
    httpGet:
      path: /probe/liveness
      port: server
    initialDelaySeconds: 5
  readinessProbe:
    httpGet:
      path: /probe/readiness
      port: server
    initialDelaySeconds: 3
>   initContainers:
>     - name: await-redis
>       image: lrakai/microservices:server-v1
>       env:
>         - name: REDIS_URL
>           value: redis://$(DATA_TIER_SERVICE_HOST):$(DATA_T
>       command:
>         - npm
>         - run-script
>         - await-redis
```

image of existing init container

scripts which are already in this image & can be executed by the command here.

```
admin@Jamess-MBP src % kubectl get pods -n probes
NAME                              READY   STATUS    RESTARTS   AGE
app-tier-84db4f4665-rg47f         1/1     Running   0          2m58s
data-tier-7557d44554-l6g6h        1/1     Running   0          23m
admin@Jamess-MBP src %
```

```
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl describe pod app-tier-84db4f4665-rg47f -n _
```

```
Host Port:      0/TCP
State:          Running
  Started:      Tue, 04 Aug 2020 15:50:05 +1200
Ready:          True
Restart Count:  0
Liveness:       http-get http://:server/probe/liveness delay=5s timeout=1s period=10s #success=1 #failure=3
Readiness:      http-get http://:server/probe/readiness delay=3s timeout=1s period=10s #success=1 #failure=3
Environment:
REDIS_URL:  redis://$(DATA_TIER_SERVICE_HOST):$(DATA_TIER_SERVICE_PORT_REDIS)
DEBUG:      express:*
Mounts:
/var/run/secrets/kubernetes.io/serviceaccount from default-token-ffpb2 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-ffpb2:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-ffpb2
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  3m43s  default-scheduler  Successfully assigned probes/app-tier-84db4f4665-rg47f to minikube
  Normal  Pulled     3m42s  kubelet, minikube  Container image "lrakai/microservices:server-v1" already present on machine
  Normal  Created    3m42s  kubelet, minikube  Created container await-redis
  Normal  Started    3m42s  kubelet, minikube  Started container await-redis
  Normal  Pulled     3m41s  kubelet, minikube  Container image "lrakai/microservices:server-v1" already present on machine
  Normal  Created    3m41s  kubelet, minikube  Created container server
  Normal  Started    3m41s  kubelet, minikube  Started container server
admin@Jamess-MBP src %
```

both containers started.

connect confirmed

CONCLUSION

- Init containers allow you to perform tasks before main application containers have an opportunity to start
- Useful for checking preconditions and preparing dependencies

Motivation for Volumes

- Sometimes useful to share data between containers in a Pod
- Lifetime of container file systems is limited to the container's lifetime
- Can lead to unexpected consequences if a container restarts

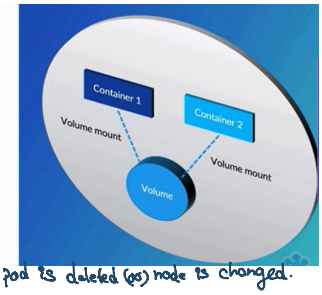
in same Pod network is same

Pod Storage in Kubernetes

- Two high-level storage options: Volumes and Persistent Volumes
- Used by mounting a directory in one or more containers in a Pod
- Pods can use multiple Volumes and Persistent Volumes
- Difference between Volumes and Persistent Volumes is how their lifetime is managed

Volumes

- Volumes are tied to a pod and their lifecycle
- Share data between containers and tolerate container restarts
- Use for non-durable storage that is deleted with the Pod
- Default Volume type is emptyDir
- Data is lost if Pod is rescheduled on a different node



volume will be lost if pod is deleted (or) node is changed.

eg:- Amazon EBS

Persistent Volumes

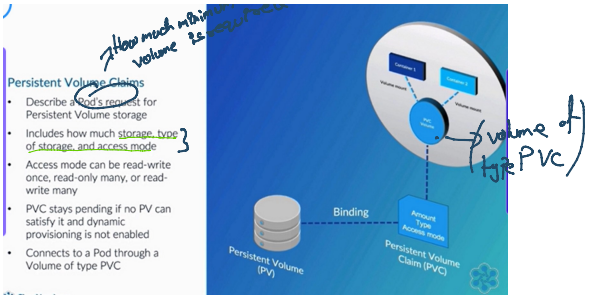
- Independent of Pod's lifetime
- Pods claim Persistent Volumes to use throughout their lifetime
- Can be mounted by multiple Pods on different Nodes if underlying storage supports it
- Can be provisioned statically in advance or dynamically on-demand

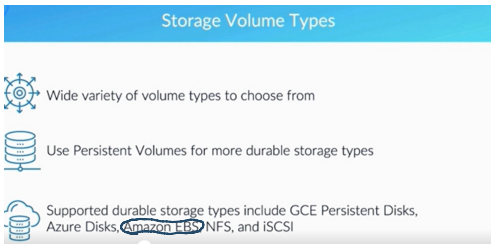
Persistent Volume Claims

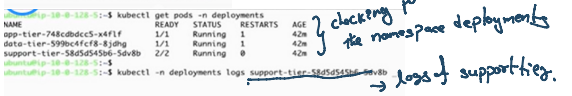
- Describe a Pod's request for Persistent Volume storage
- Includes how much storage, type of storage, and access mode
- Access mode can be read-write once, read-only many, or read-write many
- PVC stays pending if no PV can satisfy it and dynamic provisioning is not enabled
- Connects to a Pod through a Volume of type PVC

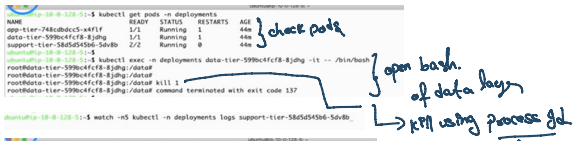
How much minimum size volume is required

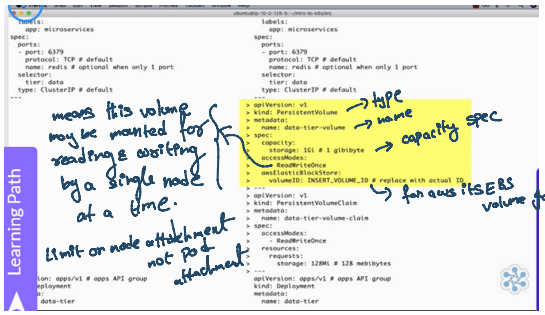
Volume of type PVC

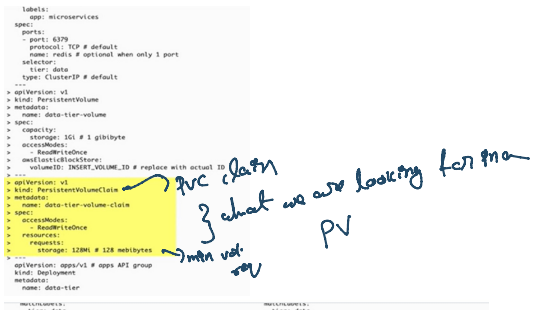








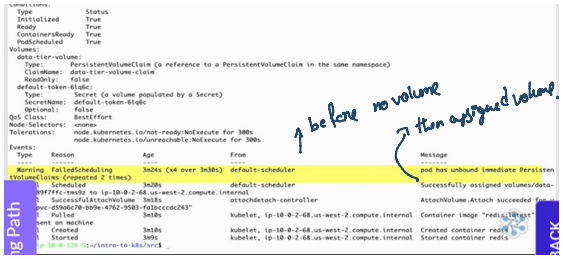


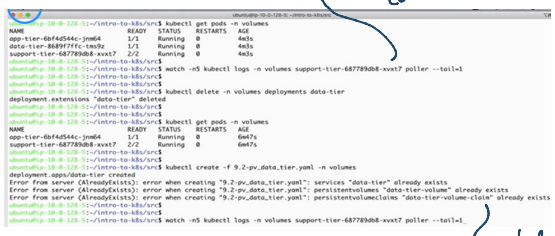



```
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$ cat 9.2-pv_data_tier.yaml | grep INSERT -C5
  capacity:
```

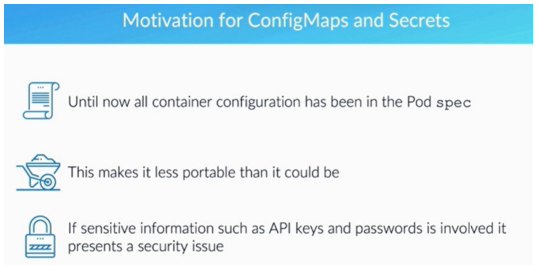
```
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$ cat 9.2-pv_data_tier.yaml | grep INSERT -CS
  capacity:
    storage: 1Gi # 1 gibibyte
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    volumeID: INSERT_VOLUME_ID # replace with actual ID
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-tier-volume-claim
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$ sed -i "s/INSERT_VOLUME_ID/$vol_id/" 9.2-pv_data_tier.yaml
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$ cat 9.2-pv_data_tier.yaml | grep $vol_id -CS
  capacity:
    storage: 1Gi # 1 gibibyte
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    volumeID: vol-0b071a7775995192c # replace with actual ID
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-tier-volume-claim
ubuntu@ip-10-0-128-5:~/intro-to-k8s/src$ _
```

(1) replacing value of EBS





→ data continue as volume is PersistentVolume



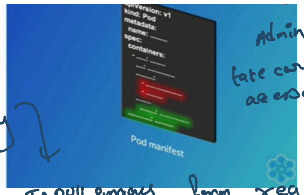
Same

After sensitive data

Admin has to take care of

specs

- Results in easier to manage and more portable manifests
- Both are similar but Secrets are specifically for sensitive data
- There are specialized types of Secrets for storing Docker registry credentials and TLS certs
- We will focus on the generic Secrets



Admin key to take care of accessing.

To pull images from registry.

Using ConfigMaps and Secrets

Data stored in key-value pairs

Pods must reference ConfigMaps and Secrets to use their data

References can be made by mounting Volumes or setting environment variables that

→ as files.

```
admin@Jamess-MBP src % cat 10.1-namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: config
  labels:
    app: counter
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl create -f 10.1-namespace.yaml _
```

} created a namespace.

```
admin@Jamess-MBP src % cat 10.2-data_tier_config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: redis-config
data:
  config: | # YAML for multi-line string
    # Redis config file
    tcp-keepalive 240
    maxmemory 1mb
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

pipe symbol. → notice there is no spec → only key value pairs data values → single key config, can have more than one.

```
  tier: data
template:
  metadata:
    labels:
      app: microservices
      tier: data
  spec: # Pod spec
    containers:
    - name: redis
      image: redis:latest
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 6379
          name: redis
      livenessProbe:
        tcpSocket:
          port: redis # named port
        initialDelaySeconds: 15
      readinessProbe:
        exec:
          command:
          - redis-cli
          - ping
        initialDelaySeconds: 5
```

```
  tier: data
template:
  metadata:
    labels:
      app: microservices
      tier: data
  spec: # Pod spec
    containers:
    - name: redis
      image: redis:latest
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 6379
          name: redis
      livenessProbe:
        tcpSocket:
          port: redis # named port
        initialDelaySeconds: 15
      readinessProbe:
        exec:
          command:
          - redis-cli
          - ping
        initialDelaySeconds: 5
      command:
      - redis-server
      - /etc/redis/redis.conf
      volumeMounts:
      - mountPath: /etc/redis
        name: config
    volumes:
    - name: config
      configMap:
        name: redis-config
        items:
        - key: config
          path: redis.conf
```

Learning Path

by this redis know to start using redis conf.

where redis.conf is located.

Volume config.

above pair to use. → we only defined one in above ie config.

Path of the file that would be mounted with config value. (→ relative to mount point of volume.

Name, declarative way not

```
admin@Jamess-MBP src % kubectl create -n config -f 10.2-data_tier_config.yaml -f 10.3-data_tier.yaml
configmap/redis-config created
service/data-tier created
deployment.apps/data-tier created
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl -n config get pods
NAME                         READY   STATUS    RESTARTS   AGE
data-tier-79775b6c55-kqxnp   0/1     Running   0          22s
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

} created.

} Pod running

```
admin@Jamess-MBP src % kubectl create -n config -f 10.2-data_tier_config.yaml -f 10.3-data_tier.yaml
configmap/redis-config created
service/data-tier created
deployment.apps/data-tier created
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl -n config get pods
NAME                         READY   STATUS    RESTARTS   AGE
data-tier-79775b6c55-kqxnp   0/1     Running   0          22s
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl exec -n config data-tier-79775b6c55-kqxnp -it -- /bin/bash
root@data-tier-79775b6c55-kqxnp:/data#
root@data-tier-79775b6c55-kqxnp:/data#
root@data-tier-79775b6c55-kqxnp:/data# cat /etc/redis/redis.conf
# Redis config file
tcp-keepalive 240
maxmemory 1mb
root@data-tier-79775b6c55-kqxnp:/data#
root@data-tier-79775b6c55-kqxnp:/data# redis-cli CONFIG GET tcp-keepalive
1) "tcp-keepalive"
2) "240"
root@data-tier-79775b6c55-kqxnp:/data#
root@data-tier-79775b6c55-kqxnp:/data# exit
exit
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

} started shell to inspect the effect of config map.

} → content as we specified.

} matches in redis as in file.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  config: |
    # Redis config file
    tcp-keepalive 240
    maxmemory 1mb
kind: ConfigMap
metadata:
  creationTimestamp: "2020-08-03T22:56:25Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
      f:config: {}
    manager: kubectl
    operation: Update
    time: "2020-08-03T22:56:25Z"
  name: redis-config
  namespace: config
  resourceVersion: "1944"
  selfLink: /api/v1/namespaces/config/configmaps/redis-config
  uid: 993a5387-d604-4fd1-b678-8d4849427d96
```

→ to see edit → updated config map.

```
admin@Jamess-MBP src % kubectl edit -n config configmaps
configmap/redis-config edited
admin@Jamess-MBP src %
admin@Jamess-MBP src % watch kubectl exec -n config data-tier-79775b6c55-kqxnp -it -- cat /etc/redis/redis.conf_
```

} volume is reflected with change we made

```
Every 2.0s: kubectl exec -n config data-tier-79775b6c55-kqxnp -it -- cat /etc/redis/red...  Jeremys-MacBook-Pro.local: Tue Aug  4 11:01:28 2020

# Redis config file
tcp-keepalive 500
maxmemory 1mb
```

```
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl exec -n config data-tier-79775b6c55-kqxnp -it -- redis-cli CONFIG GET tcp-keepalive
1) "tcp-keepalive"
2) "240"
admin@Jamess-MBP src %
admin@Jamess-MBP src % _
```

→ not changed

} redis will not be updated as we never updated deployment template (→) we never triggered a rollout.

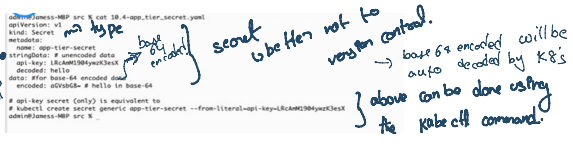
→ rollout with new config

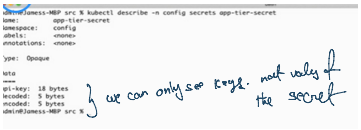
```
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl rollout -n config restart deployment data-tier
deployment.apps/data-tier restarted
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl get pods -n config
NAME                         READY   STATUS    RESTARTS   AGE
data-tier-b545555585-2lpcb   1/1     Running   0          29s
admin@Jamess-MBP src %
admin@Jamess-MBP src %
admin@Jamess-MBP src % kubectl exec -n config data-tier-b545555585-2lpcb -- redis-cli CONFIG GET tcp-keepalive
tcp-keepalive
500
admin@Jamess-MBP src % _
```

→ now its changed

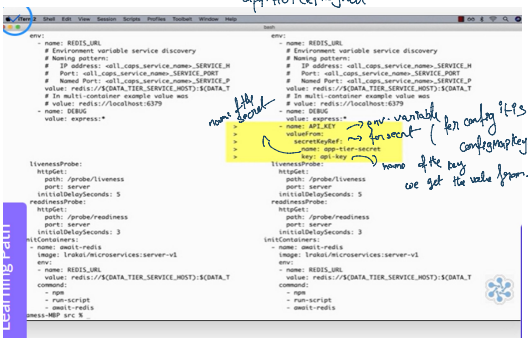
```
admin@Jamess-MBP src % cat 10.4-app_tier_secret.yaml
apiVersion: v1
kind: Secret
```

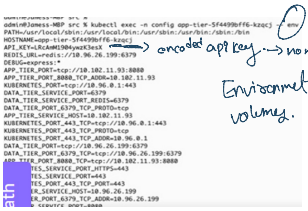
not to be type ... not to ... method.



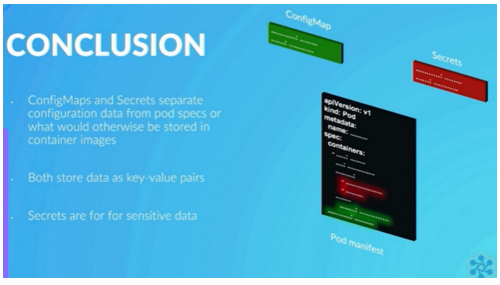


app-tier deployment





Environment variables don't update on fly like the volumes. we need to do so/bout



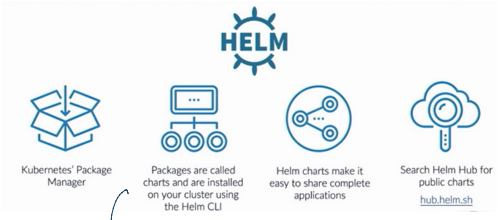
K con

le

Kubernetes Ecosystem

- Vibrant ecosystem around the core of Kubernetes
- We'll only touch on a select few members of the Kubernetes ecosystem

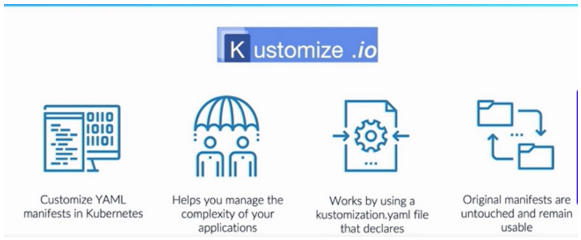
CloudAcademy



Helm has all the resources that are required for an application.

Helm Examples

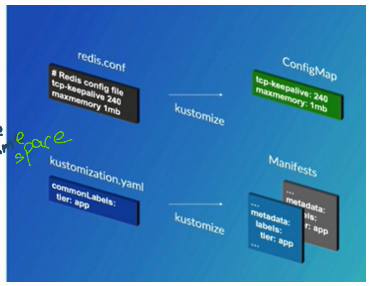
1. Use Helm for our application's Redis data tier
   - Redis charts available
   - Highly-available Redis chart avoids a single point of failure
   - Install with a single command
2. Create a chart for the entire sample application
   - Share it with anyone



Kustomize Examples

- Generating ConfigMaps and Secrets from files
- Configure common fields across multiple resources
- Apply patches to any field in a manifest
- Use overlays to customize base groups of resources

like name space



Using Kustomize

Kustomize is directly integrated with `kubectl`

Include the `--kustomize` or `-k` option to `kubectl create` or `apply`

Prometheus

Open-source monitoring and alerting system

A server for pulling in time series metric data and storing it

Inspired by an internal monitoring tool at Google called borgmon

De facto standard solution for monitoring Kubernetes

↳ some place where ICS came from

Prometheus + Kubernetes

- Kubernetes components supply all their own metrics in Prometheus format
  - Many more metrics than Metrics Server
- Adapter available to autoscale using metrics in Prometheus rather than CPU utilization
- Commonly paired with Grafana for visualizations (left image)
- Define alert rules and send notifications
- Easily installed via Helm chart

Kubeflow

→ for machine learning this is Crood.

Makes deployment of machine learning workflows on Kubernetes simple, scalable, and portable

A complete machine learning stack

Leverage Kubernetes to deploy anywhere, autoscale, etc.

Knative

Knative

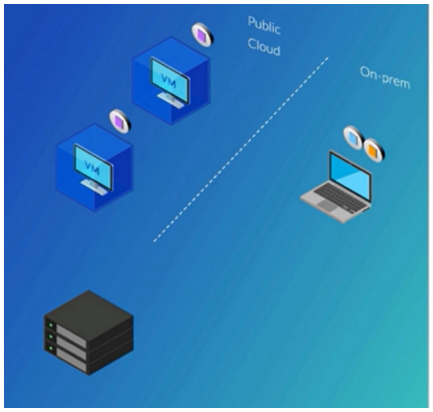
→ Crood for serverless.

Platform for building, deploying, and managing serverless workloads on Kubernetes

Can be deployed anywhere with Kubernetes, avoiding vendor lock-in

Supported by Google, IBM, and SAP

In

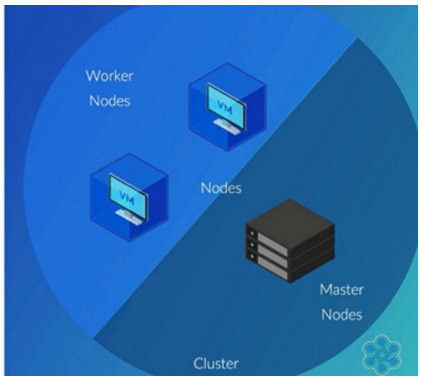


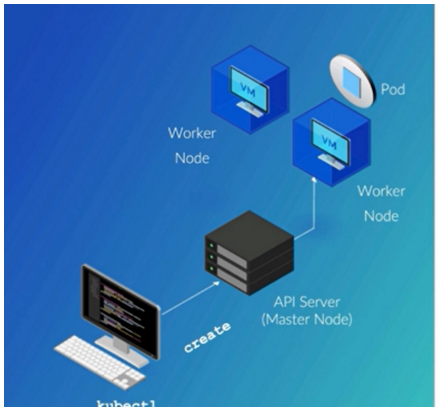
What Kubernetes Is

- Kubernetes is a distributed system
- Machines may be physical, virtual, on-prem, or in the cloud
- Schedules containers on machines
- Moves containers as machines are added/removed
- Can use different container runtimes
- Modular, extensible design

Kubernetes Architecture

- **Cluster** refers to all of the machines collectively and can be thought of as the entire running system
- **Nodes** are the machines in the cluster
- Nodes are categorized as **workers** or **masters**
- **Worker** nodes include software to run containers managed by the Kubernetes control plane
- **Master** nodes run the control plane
- The **control plane** is a set of APIs and software that Kubernetes users interact with
- The APIs and software are referred to as **master components**





Example `kubectl` commands

- `kubectl create` to create resources (Pods, Services, etc.)
- `kubectl delete` to delete resources
- `kubectl get` to get list of resources of a given type
  - `kubectl get pods`
- `kubectl describe` to print detailed info about a resource(s)
  - `kubectl describe pod server`
- `kubectl logs` to print container logs

Kubernetes in Practice

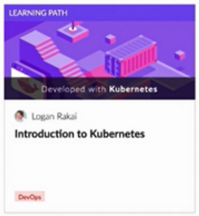
- Kubernetes terminology
- N-tier apps & Service discovery
- Deployment rollouts
- Monitoring containers with probes
- Preparing Pods with init containers
- Persistent data storage
- Separating configuration and sensitive data using ConfigMaps and Secrets

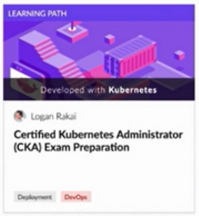
Kubernetes Ecosystem

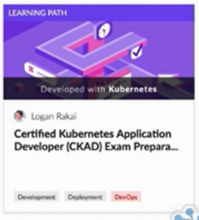
- Vibrant ecosystem around the core of Kubernetes

Where To Go Next?

More Content on Cloud Academy







Where To Go Next?

Convert your own application to run on Kubernetes

Take an application you know well and model it for Kubernetes

Reinforce what we learned and can help you find new solutions to existing challenges

Try packaging your solution as a Helm chart

Where To Go Next?

Participate in the Kubernetes Community

On GitHub
github.com/kubernetes

Join a Kubernetes Special Interest Group (SIG)

Conferences, Slack channels, and Google groups

KubeCon is the flagship Kubernetes conference