# K8S (6) Stand-Alone Application

Through K8S Dashboard or CLI. Check the demo Below

## Liveness and Readiness Probes:

Allows the kubelet to control the health of the pod's application and force a container restarting if needed.
If you are adding both Readiness Probe and Liveness Probe, give Readiness Probe some time to determine if the application is pass or fail before the Liveness Probe started. Otherwise Liveness Probe might keep force restart the application

## Liveness Probe:

Liveness Probe checks on an application's health, and if the health check fails, kubelet restarts the container automatically.
You can set the liveness probe by
(1) Liveness command
The following command check if file /tmp/healthy is existing

```
# livenessProbe.yaml
# Kubelet will wait for 15 seconds before the first probe (initialDelaySeconds: 15)
# The livenessProbe check the file /tmp/healthy every 5 seconds (periodSeconds: 5)
# If the probe failed one time, the container will be restarted (failureThreshold: 1). Default value is 3
# The container will create /tmp/healthy file and remove it after 30 seconds (Should trigger the probe failure)
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 15
      failureThreshold: 1
      periodSeconds: 5
```

(2) Liveness HTTP request

The following example send HTTP GET request to endpoint /healthz:8080 of the application. If returns failure, the kubelet will restart the container.

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
    httpHeaders:
    - name: X-Custom-Header
      value: Awesome
  initialDelaySeconds: 15
  periodSeconds: 5
```

(3) TCP Liveness probe
The kubelet will attempts to open the TCP Socket to the container. If the socket failed to open, the kubelet will restart the container.

```
livenessProbe:
  tcpSocket:
    port: 8080
  initialDelaySeconds: 15
  periodSeconds: 5
```

# Readiness Probe:

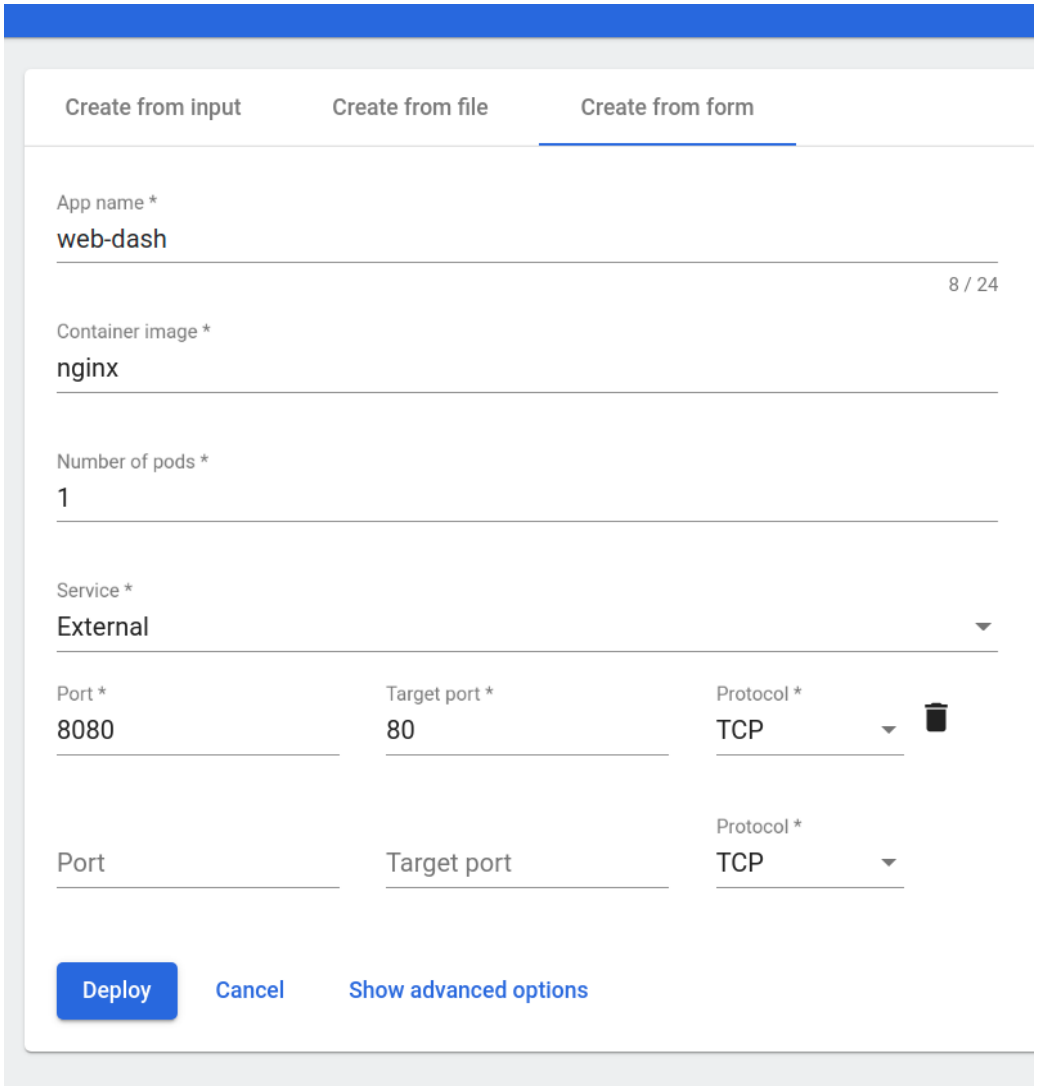
When initialize application, we set a certain condition for the Readiness Probe to check. Application will be READY to serve traffic only when all conditions are met. Usually we user Readiness Probe to verify the dependency service is ready.

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

# Demo of Dashboard:

```
$ minikube dashboard
W1224 11:05:03.147860 19743 main.go:291] Unable to resolve the current Docker CLI
context "default": context "default": context not found: open
/Users/chenyang/.docker/contexts/meta/37a8eec1ce19687d132fe29051dca629d164e2c4958ba1
41d5f4133a33f0688f/meta.json: no such file or directory
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:64949/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```

Open the dashboard in browser and create a new application through form




```
$ kubectl get pod,svc,ep,deploy,rs -l k8s-app=web-dash --show-labels
// By default, the label will be create as k8s-app=[application name]
// We can see the dashboard create the deployment->RS->pods for us
NAME READY STATUS RESTARTS AGE LABELS
pod/web-dash-55b6455445-fd2q2 1/1 Running 0 3m23s k8s-app=web-dash,pod-template-
hash=55b6455445

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
```

```
service/web-dash LoadBalancer 10.99.143.3 <pending> 8080:31854/TCP 3m24s k8s-
app=web-dash

NAME ENDPOINTS AGE LABELS
endpoints/web-dash 10.244.120.78:80 3m23s k8s-app=web-dash

NAME READY UP-TO-DATE AVAILABLE AGE LABELS
deployment.apps/web-dash 1/1 1 1 3m24s k8s-app=web-dash

NAME DESIRED CURRENT READY AGE LABELS
replicaset.apps/web-dash-55b6455445 1 1 1 3m24s k8s-app=web-dash,pod-template-
hash=55b6455445
```

# Demo of CLI:

```
$ kubectl delete deployments web-dash // Clearnup, deleing the deployment will
delete deploy+rs+pods
deployment.apps "web-dash" deleted

$ kubectl get pod,svc,ep,deploy,rs -l k8s-app=web-dash --show-labels // But the
Service is still here. enpoint is empty
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
service/web-dash LoadBalancer 10.99.143.3 <pending> 8080:31854/TCP 6m58s k8s-
app=web-dash

NAME ENDPOINTS AGE LABELS
endpoints/web-dash <none> 6m57s k8s-app=web-dash
```

Create a deployment in Declarative method.

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

```
$ kubectl create -f webserver.yaml // The deployment create RS, RS create pods
deployment.apps/webserver created

$ kubectl get deploy,po,rs
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/webserver 3/3 3 3 19s

NAME READY STATUS RESTARTS AGE
pod/webserver-f7f5c78c5-2dgmr 1/1 Running 0 19s
pod/webserver-f7f5c78c5-6v9mt 1/1 Running 0 19s
pod/webserver-f7f5c78c5-dfdvr 1/1 Running 0 19s

NAME DESIRED CURRENT READY AGE
```

```
replicaset.apps/webserver-f7f5c78c5 3 3 3 20s
```

Alternative, we can create a deployment in Imperative method with the following command:

```
$ kubectl create deployment webserver --image=nginx:alpine --replicas=3 --port=80
```

Next, ew are exposing the our webserver application.

```
# webserver-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: web-service
  labels:
    app: nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

```
$ kubectl create -f webserver-svc.yaml // Create a service with NodePort type
service/web-service created

$ kubectl get deploy,po,rs,svc,ep --show-labels
NAME READY UP-TO-DATE AVAILABLE AGE LABELS
deployment.apps/webserver 3/3 3 3 16m app=nginx

NAME READY STATUS RESTARTS AGE LABELS
pod/webserver-f7f5c78c5-2dgmr 1/1 Running 0 16m app=nginx,pod-template-
hash=f7f5c78c5
pod/webserver-f7f5c78c5-6v9mt 1/1 Running 0 16m app=nginx,pod-template-
hash=f7f5c78c5
pod/webserver-f7f5c78c5-dfdvr 1/1 Running 0 16m app=nginx,pod-template-
hash=f7f5c78c5
```

```
NAME DESIRED CURRENT READY AGE LABELS
replicaset.apps/webserver-f7f5c78c5 3 3 3 16m app=nginx,pod-template-hash=f7f5c78c5

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 11h
component=apiserver,provider=kubernetes
service/web-dash LoadBalancer 10.99.143.3 <pending> 8080:31854/TCP 75m k8s-app=web-
dash
service/web-service NodePort 10.105.228.63 <none> 80:30292/TCP 11m app=nginx

NAME ENDPOINTS AGE LABELS
endpoints/kubernetes 192.168.105.5:8443 11h endpointslice.kubernetes.io/skip-
mirror=true
endpoints/web-dash <none> 75m k8s-app=web-dash
endpoints/web-service 10.244.120.79:80,10.244.120.80:80,10.244.120.81:80 11m
app=nginx
```

Alternative, we can expose a deployment in Imperative method with the following command:

```
$ kubectl expose deployment webserver --name=web-service --type=NodePort
```

At this point, we have service exposing application, routing the request to the pods.
The Service's ClusterIP is 10.105.228.63. The port mapping is **80:31074**. Which means the we have opened  a high-port 31074 on the node. All the request to port 31074 will be routing to  10.105.228.63:80.

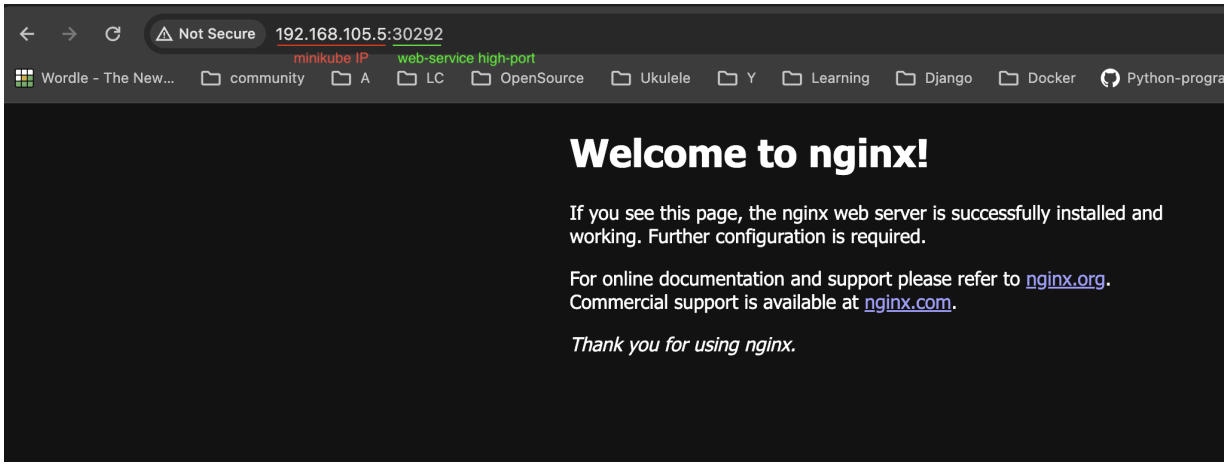
```
$ kubectl get services
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 11h
web-dash LoadBalancer 10.99.143.3 <pending> 8080:31854/TCP 65m -> web-dash is
created by dashboard
web-service NodePort 10.105.228.63 <none> 80:30292/TCP 2m20s
```

In this demo, we create deployment+RS+pods, and creating Service. But he order doesn't matter. The Service will find the pods according to the label.
In this case, there are 3 pods with app=nginx label. Service will routes the request to one of these 3 pods' endpoint.

```
$ minikube ip
192.168.105.5
```

```
$ minikube service web-service --url // Showing the service URL, shoudl be
minikubeIP + high-port
http://192.168.105.5:30292

$ minikube service web-service // open browser for web-service service
|-----------|-------------|-------------|----------------------------|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----------|-------------|-------------|----------------------------|
| default | web-service | 80 | http://192.168.105.5:30292 |
|-----------|-------------|-------------|----------------------------|
🎉  Opening service default/web-service in default browser...
```




# Demo of Liveness Probe:

There is some issue on image. I can't use the [k8s.gcr.io/busybox](k8s.gcr.io/busybox). using busybox instead

```
# livenessProbe.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 15
      failureThreshold: 1
      periodSeconds: 5
```

```
$ kubectl apply -f livenessProbe.yaml
pod/liveness-exec created

$ kubectl get pods liveness-exec -w
// -w flag means watch, the output will automativally refresh
// We can see the pod is restarted many times
NAME READY STATUS RESTARTS AGE
liveness-exec 0/1 ContainerCreating 0 3s
liveness-exec 1/1 Running 0 18s
liveness-exec 1/1 Running 1 (1s ago) 67s
liveness-exec 1/1 Running 2 (1s ago) 2m12s
```

```
$ kubectl describe pod liveness-exec
// You can see there is "Liveness probe failed" in the event section
...
node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Normal Scheduled 5m35s default-scheduler Successfully assigned default/liveness-exec
to minikube
Normal Pulled 5m30s kubelet Successfully pulled image "busybox" in 3.864s (3.864s
including waiting)
Normal Pulled 4m28s kubelet Successfully pulled image "busybox" in 877ms (877ms
including waiting)
Normal Pulled 3m23s kubelet Successfully pulled image "busybox" in 1.253s (1.253s
including waiting)
Normal Created 2m18s (x4 over 5m30s) kubelet Created container liveness
Normal Started 2m18s (x4 over 5m30s) kubelet Started container liveness
Normal Pulled 2m18s kubelet Successfully pulled image "busybox" in 1.219s (1.219s
including waiting)
Warning Unhealthy 104s (x4 over 4m59s) kubelet Liveness probe failed: cat: can't
open '/tmp/healthy': No such file or directory
Normal Killing 104s (x4 over 4m59s) kubelet Container liveness failed liveness
probe, will be restarted
```