Quick Refresher

Pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: helloapi

labels:

name: helloapi

spec:

containers:

- name: helloapi

image: gcr.io/google-samples/hello-app:1.0

Hellodeploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

spec:

replicas: 3

selector:

matchLabels:

app: hello

template:

metadata:

labels:

app: hello

spec:

containers:

- name: hello

image: gcr.io/google-samples/hello-app:2.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: hello-service

spec:

selector:

app: hello

ports:

- port: 80

targetPort: 8080

Helloingress.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

kubernetes.io/ingress.class: kong

name: hello-ingress

spec:

rules:

- http:

paths:

- backend:

service:

name: hello-service

port:

number: 80

path: /hellodemo

pathType: Prefix

**Probes**

[Configure Liveness, Readiness and Startup Probes | Kubernetes](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/)

kubectl apply -f <https://k8s.io/examples/pods/probe/exec-liveness.yaml>

kubectl describe pod liveness-exec

Create a hello pod

hellopod.yaml

apiVersion: v1

kind: Pod

metadata:

name: hellopod

labels:

name: hellopod

spec:

containers:

- name: hellopod

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

**livenessProbe:**

**httpGet:**

**path: /health**

**port: 8080**

ports:

- containerPort: 8080

kubectl apply -f hellopod.yaml

kubectl describe pod hellopod

Let’s update to see a Readiness failure => that means service is not ready to accept traffic

hellopod.yaml

apiVersion: v1

kind: Pod

metadata:

name: hellopod

labels:

name: hellopod

spec:

containers:

- name: hellopod

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health

port: 8080

**readinessProbe:**

**httpGet:**

**path: /health**

**port: 8081**

ports:

- containerPort: 8080

This must fail and you must see

PS C:\Users\Siva\Downloads\astaradv2> **kubectl get pods**

NAME READY STATUS RESTARTS AGE

hellopod **0/1**  Running 0 11s

Warning Unhealthy 3s (x5 over 22s) kubelet **Readiness probe failed:** Get "http://10.1.0.125:8081/health": dial tcp 10.1.0.125:8081: connect: connection refused

Fixing the readiness probe and updating the hellopod.yaml

hellopod.yaml

apiVersion: v1

kind: Pod

metadata:

name: hellopod

labels:

name: hellopod

spec:

containers:

- name: hellopod

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health

port: 8080

**readinessProbe:**

**httpGet:**

**path: /health**

**port: 8080**

ports:

- containerPort: 8080

kubectl delete -f .\hellopod.yaml

kubectl apply -f .\hellopod.yaml

In .net 6.0

[Health checks in ASP.NET Core | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/health-checks?view=aspnetcore-6.0)

hellodeploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

spec:

replicas: 3

selector:

matchLabels:

app: hello

template:

metadata:

labels:

app: hello

spec:

containers:

- name: hello

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

**readinessProbe:**

**httpGet:**

**path: /health**

**port: 8080**

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: hello-service

spec:

selector:

app: hello

ports:

- port: 80

targetPort: 8080

kubectl apply -f hellodeploy.yaml

hellodeploy-2-failure.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

spec:

replicas: 3

selector:

matchLabels:

app: hello

template:

metadata:

labels:

app: hello

spec:

containers:

- name: hello

image: gcr.io/google-samples/hello-app:2.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

**readinessProbe:**

**httpGet:**

**path: /health**

**port: 8081**

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: hello-service

spec:

selector:

app: hello

ports:

- port: 80

targetPort: 8080

kubectl apply –f hellodeploy-2-failure.yaml

PS C:\Users\Siva\Downloads\astaradv2> kubectl get pods

NAME READY STATUS RESTARTS AGE

hello-6999d6c954-4d6qg 1/1 Running 0 96s

hello-6999d6c954-lkcm2 1/1 Running 0 96s

hello-6999d6c954-mc5vn 1/1 Running 0 96s

**hello-79d9fd6-54z4w 0/1 Running 0 13s**

kubectl describe pod <<pod id>>

Warning Unhealthy 3s (x5 over 22s) kubelet Readiness probe failed: Get "http://10.1.0.125:8081/health": dial tcp 10.1.0.125:8081: connect: connection refused

Init Container [Service dependencies can be managed by these]

initpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: hellopod

labels:

name: hellopod

spec:

containers:

- name: hellopod

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

**initContainers:**

**- name: init-greeting-service**

**image: busybox:1.28**

**command: ['sh', '-c', "until nslookup greeting-service.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done"]**

To address this we need to create a Greeting Service

greeting-service.yaml

apiVersion: v1

kind: Service

metadata:

name: greeting-service

spec:

selector:

app: greeting

ports:

- port: 80

targetPort: 5000

kubectl apply -f .\greeting-service.yaml

kubectl get pods

Multiple service dependencies

apiVersion: v1

kind: Pod

metadata:

name: hellopod

labels:

name: hellopod

spec:

containers:

- name: hellopod

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

**initContainers:**

**- name: init-greeting-service**

**image: busybox:1.28**

**command: ['sh', '-c', "until nslookup greeting-service.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done"]**

**- name: init-db-service**

**image: busybox:1.28**

**command: ['sh', '-c', "until nslookup db-service.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done"]**

kubectl delete -f .\initpod.yaml

pod "hellopod" deleted

kubectl delete -f .\greeting-service.yaml

service "greeting-service" deleted

kubectl apply -f .\initpod.yaml

pod/hellopod created

kubectl get pods

NAME READY STATUS RESTARTS AGE

hellopod 0/1 Init:0/2 0 2s

Create the greeting-service

Create the DB service…

db-service.yaml

apiVersion: v1

kind: Service

metadata:

name: db-service

spec:

selector:

app: db-service

ports:

- port: 3306

targetPort: 3306

Create both services for your initpod to launch

To view the logs of init containers

kubectl logs hellopod -c init-greeting-service

kubectl logs hellopod -c init-db-service

Create a Deployment with init containers

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

spec:

replicas: 3

selector:

matchLabels:

app: hello

template:

metadata:

labels:

app: hello

spec:

**initContainers:**

**- name: init-greeting-service**

**image: busybox:1.28**

**command: ['sh', '-c', "until nslookup greeting-service.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done"]**

**- name: init-db-service**

**image: busybox:1.28**

**command: ['sh', '-c', "until nslookup db-service.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for myservice; sleep 2; done"]**

containers:

- name: hello

image: gcr.io/google-samples/hello-app:1.0

resources:

limits:

memory: "128Mi"

cpu: "500m"

readinessProbe:

httpGet:

path: /health

port: 8080

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: hello-service

spec:

selector:

app: hello

ports:

- port: 80

targetPort: 8080

Order of booting

1. Pods are scheduled to run on Nodes
2. Pods will try to execute Init containers in order mentioned in the YAML
3. The Pod will not initialise unless the dependencies are resolved
4. When resolved the Pod move to POD initialising state
5. POD Successfully run [or waits for dependencies till they are resolved ]

Check storage options in Kubernetes

kubectl get pv,pvc,sc

volumepod.yaml

apiVersion: v1

kind: Pod

metadata:

name: multi-container-example

spec:

containers:

- name: nginx

image: nginx:stable-alpine

ports:

- containerPort: 80

**volumeMounts:**

**- name: datadirectory**

**mountPath: /usr/share/nginx/html**

**readOnly: true**

- name: content

image: alpine:latest

**volumeMounts:**

**- name: datadirectory**

**mountPath: /html**

**command: ["/bin/sh", "-c"]**

args:

- while true; do

echo $(date)"<br />" >> /html/index.html;

sleep 5;

done

**volumes:**

**- name: datadirectory**

**emptyDir: {}**

kubectl exec -it multi-container-example -- sh

curl localhost

cd /usr/share/nginx/html

echo "Test" > test.html

sh: can't create test.html: Read-only file system

exit

kubectl exec -it multi-container-example -c content -- sh

cd /html

cat index.html

echo “Test” > index.html

exit

# Persistent Volumes

Static Provisioning of a Persistent Volume

pvhostpath.yaml

kind: PersistentVolume

apiVersion: v1

metadata:

name: pv-sc-example

labels:

mydata: pvvol

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteMany

persistentVolumeReclaimPolicy: Delete

hostPath:

type: DirectoryOrCreate

path: "/c/k8sdata"

========================================================

We will use dynamic provisioning with PVC

pvchostpath.yaml

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: **html**

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 500Mi

kubectl apply -f .\pvchostpath.yaml

kubectl get pv,pvc --show-labels

writer.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: writer

spec:

replicas: 1

selector:

matchLabels:

app: writer

template:

metadata:

labels:

app: writer

spec:

containers:

- name: content

image: alpine:latest

**volumeMounts:**

**- name: html**

**mountPath: /html**

command: ["/bin/sh", "-c"]

args:

- while true; do

date >> /html/index.html;

sleep 5;

done

volumes:

- name: html

**persistentVolumeClaim:**

**claimName: html**

kubectl apply -f .\writer.yaml

kubectl describe pod <<pod name>>

reader.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: reader

spec:

replicas: 3

selector:

matchLabels:

app: reader

template:

metadata:

labels:

app: reader

spec:

containers:

- name: nginx

image: nginx:stable-alpine

ports:

- containerPort: 80

**volumeMounts:**

**- name: html**

**mountPath: /usr/share/nginx/html**

**readOnly: true**

volumes:

- name: html

**persistentVolumeClaim:**

**claimName: html**

---

apiVersion: v1

kind: Service

metadata:

name: reader

spec:

type: NodePort

selector:

app: reader

ports:

- protocol: TCP

port: 80

targetPort: 80

nodePort: 32300

kubectl apply -f .\reader.yaml

Go to browser on localhost:32300

Delete the POD

kubectl delete -f .\writer.yaml

Give some time recreate the deployment

kubectl apply -f .\writer.yaml StatefulSets

The StatefulSet controller is tailored to managing Pods that must persist or maintain state. Pod identity including hostname, network, and storage can be considered persistent.

They ensure persistence by making use of three things:

* The StatefulSet controller enforced predictable naming, and ordered provisioning/updating/deletion.
* A headless service to provide a unique network identity.
* A volume template to ensure stable per-instance storage.

### Exercise: Managing StatefulSets

Objective: Create, update, and delete a StatefulSet to gain an understanding of how the StatefulSet lifecycle differs from other workloads with regards to updating, deleting and the provisioning of storage.

1. Create StatefulSet sts-example using the yaml block below or the manifest manifests/sts-example.yaml.

manifests/sts-example.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: sts-example

spec:

replicas: 3

revisionHistoryLimit: 3

selector:

matchLabels:

app: stateful

serviceName: app

updateStrategy:

type: OnDelete

template:

metadata:

labels:

app: stateful

spec:

containers:

- name: nginx

image: nginx:stable-alpine

ports:

- containerPort: 80

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: www

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: hostpath

resources:

requests:

storage: 200Mi

Command

$ kubectl create -f manifests/sts-example.yaml

1. Immediately watch the Pods being created.

$ kubectl get pods --show-labels --watch

Unlike Deployments or DaemonSets, the Pods of a StatefulSet are created one-by-one, going by their ordinal index. Meaning, sts-example-0 will fully be provisioned before sts-example-1 starts up. Additionally, take notice of the controller-revision-hash label. This serves the same purpose as the controller-revision-hash label in a DaemonSet or the pod-template-hash in a Deployment. It provides a means of tracking the revision of the Pod Template and enables rollback functionality.

1. More information on the StatefulSet can be gleaned about the state of the StatefulSet by describing it.

$ kubectl describe statefulset sts-example

Within the events, notice that it is creating claims for volumes before each Pod is created.

1. View the current Persistent Volume Claims.

$ kubectl get pvc

The StatefulSet controller creates a volume for each instance based off the volumeClaimTemplate. It prepends the volume name to the Pod name. e.g. www-sts-example-0.

1. Update the StatefulSet's Pod Template and add a few additional labels.

$ kubectl apply -f manifests/sts-example.yaml --record

< or >

$ kubectl edit statefulset sts-example --record

1. Return to watching the Pods.

$ kubectl get pods --show-labels

None of the Pods are being updated to the new version of the Pod.

1. Delete the sts-example-2 Pod.

$ kubectl delete pod sts-example-2

1. Immediately get the Pods.

$ kubectl get pods --show-labels --watch

The new sts-example-2 Pod should be created with the new additional labels. The OnDelete Update Strategy will not spawn a new iteration of the Pod until the previous one was deleted. This allows for manual gating the update process for the StatefulSet.

1. Update the StatefulSet and change the Update Strategy Type to RollingUpdate.

$ kubectl apply -f manifests/sts-example.yaml --record

< or >

$ kubectl edit statefulset sts-example --record

1. Immediately watch the Pods once again.

$ kubectl get pods --show-labels --watch

Note that the Pods are sequentially updated in descending order, or largest to smallest based on the Pod's ordinal index. This means that if sts-example-2 was not updated already, it would be updated first, then sts-example-1 and finally sts-example-0.

1. Delete the StatefulSet sts-example

$ kubectl delete statefulset sts-example

1. View the Persistent Volume Claims.

$ kubectl get pvc

Created PVCs are NOT garbage collected automatically when a StatefulSet is deleted. They must be reclaimed independently of the StatefulSet itself.

1. Recreate the StatefulSet using the same manifest.

$ kubectl create -f manifests/sts-example.yaml --record

1. View the Persistent Volume Claims again.

$ kubectl get pvc

Note that new PVCs were NOT provisioned. The StatefulSet controller assumes if the matching name is present, that PVC is intended to be used for the associated Pod.

Summary: Like many applications where state must be taken into account, the planning and usage of StatefulSets requires forethought. The consistency brought by standard naming, ordered updates/deletes and templated storage does however make this task easier.

### Exercise: Understanding StatefulSet Network Identity

Objective: Create a *"headless service"* or a service without a ClusterIP (ClusterIP=None) for use with the StatefulSet sts-example, then explore how this enables consistent service discovery.

1. Create the headless service app using the app=stateful selector from the yaml below or the manifest manifests/service-sts-example.yaml.

manifests/service-sts-example.yaml

apiVersion: v1

kind: Service

metadata:

name: app

spec:

clusterIP: None

selector:

app: stateful

ports:

- protocol: TCP

port: 80

targetPort: 80

Command

$ kubectl create -f manifests/service-sts-example.yaml

1. Describe the newly created service

$ kubectl describe svc app

Notice that it does not have a clusterIP, but does have the Pod Endpoints listed. Headless services are unique in this behavior.

1. Query the DNS entry for the app service.

$ kubectl exec sts-example-0 -- nslookup app.default.svc.cluster.local

An A record will have been returned for each instance of the StatefulSet. Querying the service directly will do simple DNS round-robin load-balancing.

1. Finally, query one of the instances directly.

$ kubectl exec sts-example-0 -- nslookup sts-example-0.app.default.svc.cluster.local

This is a unique feature to StatefulSets. This allows for services to directly interact with a specific instance of a Pod. If the Pod is updated and obtains a new IP, the DNS record will immediately point to it enabling consistent service discovery.

Summary: StatefulSet service discovery is unique within Kubernetes in that it augments a headless service (A service without a unique ClusterIP) to provide a consistent mapping to the individual Pods. These mappings take the form of an A record in format of: <StatefulSet Name>-<ordinal>.<service name>.<namespace>.svc.cluster.local and can be used consistently throughout other Workloads.

Clean Up Command

kubectl delete svc app

kubectl delete statefulset sts-example

kubectl delete pvc www-sts-example-0 www-sts-example-1 www-sts-example-2

**Let’s create a MYSQL**

[**https://kubernetes.io/docs/tasks/run-application/run-single-instance-stateful-application/**](https://kubernetes.io/docs/tasks/run-application/run-single-instance-stateful-application/)

MSSQL

**MSSQL :** [**ABC2018/db-statefulset.yml at master · NileshGule/ABC2018 · GitHub**](https://github.com/NileshGule/ABC2018/blob/master/Kubernetes/TechTalksDB/db-statefulset.yml)

MySQL STS

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql-sts

spec:

replicas: 1

revisionHistoryLimit: 3

selector:

matchLabels:

app: mysql-sts

serviceName: app

updateStrategy:

type: OnDelete

template:

metadata:

labels:

app: mysql-sts

spec:

containers:

- name: mysql-sts

image: mysql:5.7

resources:

limits:

memory: "1024Mi"

cpu: "500m"

env:

- name: MYSQL\_ROOT\_PASSWORD

value: rootroot

ports:

- containerPort: 3306

volumeMounts:

- mountPath: /docker-entrypoint-initdb.d

name: mysql-initdb

volumeClaimTemplates:

- metadata:

name: mysql-initdb

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: hostpath

resources:

requests:

storage: 200Mi

---

apiVersion: v1

kind: Service

metadata:

name: mysql-sts

spec:

clusterIP: None

selector:

app: mysql-sts

ports:

- protocol: TCP

port: 3306

targetPort: 3306

From application

apiVersion: apps/v1

kind: Deployment

metadata:

name: flask-mysql

spec:

selector:

matchLabels:

app: flask-mysql

template:

metadata:

labels:

app: flask-mysql

spec:

containers:

- name: flask-mysql

image: shivaspk/flask-mysql

env:

- name: MYSQL\_DATABASE\_USER

value: root

- name: MYSQL\_DATABASE\_PASSWORD

value: rootroot

- name: MYSQL\_DATABASE\_DB

value: cloudenabled

- name: MYSQL\_DATABASE\_HOST

value: mysql-sts

- name: MYSQL\_DATABASE\_PORT

value: '3306'

resources:

limits:

memory: "512Mi"

cpu: "500m"

ports:

- containerPort: 5000

---

apiVersion: v1

kind: Service

metadata:

name: flask-mysql

spec:

type: NodePort

selector:

app: flask-mysql

ports:

- port: 80

targetPort: 5000

nodePort: 31500

MSSQL [Microsoft SQL Server]

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: sqlserverstatefulset

spec:

serviceName: db-mssql

replicas: 1

selector:

matchLabels:

app: db

updateStrategy:

type: OnDelete

template:

metadata:

labels:

app: db

spec:

terminationGracePeriodSeconds: 10

containers:

- name: sql2019

image: mcr.microsoft.com/mssql/server:2019-latest

imagePullPolicy: IfNotPresent

**resources:**

**requests:**

**memory: "2Gi"**

**limits:**

**memory: "3Gi"**

ports:

- name: dbport

containerPort: 1433

env:

- name: ACCEPT\_EULA

value: "Y"

- name: SA\_PASSWORD

value: January2018

# valueFrom:

# secretKeyRef:

# name: sqlsecret

# key: sapassword

- name: MSSQL\_PID

value: Developer

volumeMounts:

- name: mssqldb

mountPath: /var/opt/mssql

volumeClaimTemplates:

- metadata:

name: mssqldb

spec:

accessModes: ["ReadWriteOnce"]

storageClassName: hostpath

resources:

requests:

storage: 500Mi

---

apiVersion: v1

kind: Service

metadata:

name: db-mssql

spec:

selector:

app: db

ports:

- protocol: TCP

port: 1433

targetPort: 1433

type: NodePort

QoS for Kubernetes

kubectl create ns qos-example

kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod.yaml --namespace=qos-example

kubectl describe pod -n qos-example

**QoS Class: Guaranteed**

**Burstable QoS**

**kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod-2.yaml --namespace=qos-example**

**BestEffort QoS**

**kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod-3.yaml --namespace=qos-example**

**Once done delete the Namespace**

**kubectl delete ns qos-example**

DOWNWARD API

kubectl apply -f <https://k8s.io/examples/pods/inject/dapi-envars-pod.yaml>

Stateful set of MSSQL Server

sqlsecret.yaml

apiVersion: v1

kind: Secret

metadata:

name: mssql-creds

type: Opaque

data:

password: SmFudWFyeTIwMTg=

username: U0E=

To update the secret value you need to get the base 64 value

sqlsecret.yaml

apiVersion: v1

kind: Secret

metadata:

name: mssql-creds

type: Opaque

data:

password: SmFudWFyeTIwMjI=

username: U0E=

kubectl describe secret mssql-creds

mssqlsts.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: sqlserverstatefulset

spec:

serviceName: db-mssql

replicas: 1

selector:

matchLabels:

app: db

updateStrategy:

type: OnDelete

template:

metadata:

labels:

app: db

spec:

terminationGracePeriodSeconds: 10

containers:

- name: sql2019

image: mcr.microsoft.com/mssql/server:2019-latest

imagePullPolicy: IfNotPresent

resources:

requests:

memory: "2Gi"

limits:

memory: "3Gi"

ports:

- name: dbport

containerPort: 1433

env:

- name: ACCEPT\_EULA

value: "Y"

- name: SA\_PASSWORD

**valueFrom:**

**secretKeyRef:**

**name: mssql-creds**

**key: password**

- name: MSSQL\_PID

value: Developer

volumeMounts:

- name: mssqldb

mountPath: /var/opt/mssql

volumeClaimTemplates:

- metadata:

name: mssqldb

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: hostpath

resources:

requests:

storage: 500Mi

---

apiVersion: v1

kind: Service

metadata:

name: db-mssql

spec:

selector:

app: db

ports:

- protocol: TCP

port: 1433

targetPort: 1433

**nodePort: 32433**

type: NodePort

**EmployeeAPI.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: empapi

spec:

selector:

matchLabels:

app: empapi

template:

metadata:

labels:

app: empapi

spec:

containers:

- name: empapi

image: shivaspk/empapi:latest

env:

**- name: DbServer**

**value: "db-mssql"**

**- name: DbPort**

**value: "1433"**

**- name: DbUser**

**valueFrom:**

**secretKeyRef:**

**name: mssql-creds**

**key: username**

**- name: Password**

**valueFrom:**

**secretKeyRef:**

**name: mssql-creds**

**key: password**

**- name: Database**

**value: "EMSdb"**

resources:

limits:

memory: "256Mi"

cpu: "500m"

---

apiVersion: v1

kind: Service

metadata:

name: empapi

spec:

type: NodePort

selector:

app: empapi

ports:

- port: 80

targetPort: 80

nodePort: 31700

Go to <http://localhost:31700/swagger/index.html>

Self Learning

[RabbitMQ Cluster Kubernetes Operator Quickstart — RabbitMQ](https://www.rabbitmq.com/kubernetes/operator/quickstart-operator.html)

Deploy Rabbitmq Operator

kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml"

**kubectl api-resources**

rabbitmq-cluster.yaml

apiVersion: rabbitmq.com/v1beta1

kind: RabbitmqCluster

metadata:

labels:

app: rabbitmq

annotations:

some: annotation

name: rabbitmqcluster-sample

kubectl apply -f .\rabbitmq-cluster.yaml

kubectl get svc,secret

**kubectl port-forward rabbitmqcluster-sample-server-0 15672**

<http://localhost:15672>

kubectl get secret rabbitmqcluster-sample-default-user -o yaml

Decode username and password, use that for login

Helm 3

[Helm](https://helm.sh/)

Download latest version: [Releases · helm/helm · GitHub](https://github.com/helm/helm/releases)

./helm.exe version

./helm.exe repo add stable https://charts.helm.sh/stable

./helm repo add kong <https://charts.konghq.com>

./helm repo update

./helm install kong/kong --generate-name

./helm list

kubectl get pods

curl localhost

./helm uninstall kong-1653961732

====

Deploy a Prometheus Operator

.\helm.exe repo add stable <https://charts.helm.sh/stable>

.\helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

.\helm repo update

.\helm.exe install my-monitor prometheus-community/prometheus

**.\helm.exe uninstall my-monitor**

**./helm uninstall my-monitor**

Enable metrics

[Monitoring .NET Core applications on Kubernetes | Red Hat Developer](https://developers.redhat.com/blog/2020/08/05/monitoring-net-core-applications-on-kubernetes#exposing_metrics_from__net_core)

Add Nuget package

prometheus-net.AspNetCore

Startup.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.OpenApi.Models;

using Prometheus;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace metricsapi

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "metricsapi", Version = "v1" });

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

**// if (env.IsDevelopment())**

**// {**

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "metricsapi v1"));

**//for metrics**

**app.UseHttpMetrics();**

**//}**

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

**//for metrics**

**endpoints.MapMetrics();**

});

}

}

}

Build a docker image

docker build -t <<id>>/metricsapi .

docker push <<id>>/metricsapi

Create a Deployment and Service

apiVersion: apps/v1

kind: Deployment

metadata:

name: metricsapi

spec:

selector:

matchLabels:

app: metricsapi

template:

metadata:

labels:

app: metricsapi

spec:

containers:

- name: metricsapi

image: shivaspk/metricsapi

resources:

limits:

memory: "128Mi"

cpu: "500m"

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: metricsapi

spec:

type: NodePort

selector:

app: metricsapi

ports:

- port: 80

targetPort: 80

nodePort: 31900

Install prometheus operator

git clone <https://github.com/prometheus-operator/kube-prometheus.git>

cd .\kube-prometheus\

kubectl create -f manifests/setup

kubectl create -f manifests/

kubectl get pods -n monitoring

kubectl --namespace monitoring port-forward svc/prometheus-k8s 9090:9090

If error then do this

kubectl port-forward services/prometheus-k8s 9000:9090 8080:8080 -n monitoring

Create a Service Monitor

apiVersion: monitoring.coreos.com/v1

kind: ServiceMonitor

metadata:

labels:

k8s-app: prometheus-example-monitor

name: prometheus-example-monitor

spec:

endpoints:

- interval: 30s

port: 80-tcp

scheme: http

selector:

matchLabels:

app: metricsapi

kubectl apply -f .\servicemonitor.yaml

Invoke <http://localhost:31900/WeatherForecast>

Clean up:

cd .\kube-prometheus\

kubectl delete-f manifests/

Web Api for ASP.NET

[Create web APIs with ASP.NET Core | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-6.0)

Entity Framework Core: [Overview of Entity Framework Core - EF Core | Microsoft Docs](https://docs.microsoft.com/en-us/ef/core/)

Start if you are totally new: [Tutorial: Create a web API with ASP.NET Core | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio)

**PetDBContext.cs**

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PetsApi.Models

{

public class PetDBContext: DbContext

{

public PetDBContext(DbContextOptions<PetDBContext> options)

: base(options)

{

}

public DbSet<Pet> Pets { get; set; }

}

}

**Pet.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PetsApi.Models

{

public class Pet

{

public int petId { get; set; }

public String name { get; set; }

public String breed { get; set; }

}

}

**Startup.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.OpenApi.Models;

using PetsApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PetsApi

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

**services.AddDbContext<PetDBContext>(opt =>**

**opt.UseInMemoryDatabase("Pets"));**

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "PetsApi", Version = "v1" });

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

**app.UseDeveloperExceptionPage();**

**app.UseSwagger();**

**app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "PetsApi v1"));**

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

Update the Startup.cs to have Health Check support for Kubernetes Readiness and Liveness Probes

**Startup.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.OpenApi.Models;

using PetsApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PetsApi

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddDbContext<PetDBContext>(opt =>

opt.UseInMemoryDatabase("Pets"));

**services.AddHealthChecks();**

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "PetsApi", Version = "v1" });

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "PetsApi v1"));

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

**endpoints.MapHealthChecks("/health/readiness");**

**endpoints.MapHealthChecks("/health/liveness");**

});

}

}

}

To check the specific service

[GitHub - Xabaril/AspNetCore.Diagnostics.HealthChecks: Enterprise HealthChecks for ASP.NET Core Diagnostics Package](https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks)

**services.AddHealthChecks().AddSqlServer(Configuration["Data:ConnectionStrings:Sql"]);**

Configuring the readiness and liveness on pet api

apiVersion: apps/v1

kind: Deployment

metadata:

name: petapiprobes

spec:

replicas: 3

selector:

matchLabels:

app: petapiprobes

template:

metadata:

labels:

app: petapiprobes

spec:

containers:

- name: petapiprobes

image: **shivaspk/petapiprobes**

resources:

limits:

memory: "256Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health/liveness

port: 80

initialDelaySeconds: 5

readinessProbe:

httpGet:

path: /health/readiness

port: 80

initialDelaySeconds: 5

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: petservice

spec:

selector:

app: petapiprobes

ports:

- port: 80

targetPort: 80

Docker file:

#See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile to build your images for faster debugging.

FROM mcr.microsoft.com/dotnet/aspnet:5.0-alpine AS base

WORKDIR /app

EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:5.0-alpine AS build

WORKDIR /src

COPY ["PetsApi.csproj", "."]

RUN dotnet restore "./PetsApi.csproj"

COPY . .

WORKDIR "/src/."

RUN dotnet build "PetsApi.csproj" -c Release -o /app/build

FROM build AS publish

RUN dotnet publish "PetsApi.csproj" -c Release -o /app/publish

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "PetsApi.dll"]

For Adding metrics

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.OpenApi.Models;

using PetsApi.Models;

using Prometheus;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PetsApi

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddDbContext<PetDBContext>(opt =>

opt.UseInMemoryDatabase("Pets"));

services.AddHealthChecks();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "PetsApi", Version = "v1" });

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "PetsApi v1"));

**app.UseMetricServer();**

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

**endpoints.MapMetrics();**

endpoints.MapHealthChecks("/health/readiness");

endpoints.MapHealthChecks("/health/liveness");

});

}

}

}

Dependency is prometheus-net.AspNetCore in NuGet package and proceed.

.NET core with SQL Server on Docker

[GitHub - shivaspk/dotnetsqlcompose: This is a sample dotnet with sqlserver docker project](https://github.com/shivaspk/dotnetsqlcompose)

Entity Framework: [What is Entity Framework? (entityframeworktutorial.net)](https://www.entityframeworktutorial.net/what-is-entityframework.aspx)

The complete project with EF is available:

EMS system using Entity framework!

<https://drive.google.com/file/d/1rnZbYwIn5CX-aL_1jhDtA_VkMbV4E2HN/view?usp=sharing>

All yamls available at : <https://drive.google.com/drive/folders/1RSOaKEmpg0pIHHe9_4mp2EZ9rdpIjBU1?usp=sharing>

employeedeploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: empapi

spec:

selector:

matchLabels:

app: empapi

template:

metadata:

labels:

app: empapi

spec:

containers:

- name: empapi

image: shivaspk/empapi:latest

env:

- name: DbServer

value: "db-mssql"

- name: DbPort

value: "1433"

- name: DbUser

value: "SA"

- name: Password

value: "January2018"

- name: Database

value: "EMSdb"

resources:

limits:

memory: "256Mi"

cpu: "500m"

initContainers:

- name: init-db-mssql

image: busybox:1.28

command: ['sh', '-c', "until nslookup db-mssql.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo waiting for mydb; sleep 2; done"]

---

apiVersion: v1

kind: Service

metadata:

name: empapi

spec:

type: NodePort

selector:

app: empapi

ports:

- port: 80

targetPort: 80

nodePort: 31700

mssqlsts.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: sqlserverstatefulset

spec:

serviceName: db-mssql

replicas: 1

selector:

matchLabels:

app: db

updateStrategy:

type: OnDelete

template:

metadata:

labels:

app: db

spec:

terminationGracePeriodSeconds: 10

containers:

- name: sql2019

image: mcr.microsoft.com/mssql/server:2019-latest

imagePullPolicy: IfNotPresent

resources:

requests:

memory: "2Gi"

limits:

memory: "3Gi"

ports:

- name: dbport

containerPort: 1433

env:

- name: ACCEPT\_EULA

value: "Y"

- name: SA\_PASSWORD

value: January2018

# valueFrom:

# secretKeyRef:

# name: sqlsecret

# key: sapassword

- name: MSSQL\_PID

value: Developer

volumeMounts:

- name: mssqldb

mountPath: /var/opt/mssql

volumeClaimTemplates:

- metadata:

name: mssqldb

spec:

accessModes: ["ReadWriteOnce"]

storageClassName: hostpath

resources:

requests:

storage: 500Mi

---

apiVersion: v1

kind: Service

metadata:

name: db-mssql

spec:

selector:

app: db

ports:

- protocol: TCP

port: 1433

targetPort: 1433

nodePort: 32433

type: NodePort

sqlsecret.yaml

apiVersion: v1

kind: Secret

metadata:

name: mssql-creds

type: Opaque

data:

password: SmFudWFyeTIwMjI=

username: U0E=

mssqlsecret.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: sqlserverstatefulset

spec:

serviceName: db-mssql

replicas: 1

selector:

matchLabels:

app: db

updateStrategy:

type: OnDelete

template:

metadata:

labels:

app: db

spec:

terminationGracePeriodSeconds: 10

containers:

- name: sql2019

image: mcr.microsoft.com/mssql/server:2019-latest

imagePullPolicy: IfNotPresent

resources:

requests:

memory: "2Gi"

limits:

memory: "3Gi"

ports:

- name: dbport

containerPort: 1433

env:

- name: ACCEPT\_EULA

value: "Y"

- name: SA\_PASSWORD

valueFrom:

secretKeyRef:

name: mssql-creds

key: password

- name: MSSQL\_PID

value: Developer

volumeMounts:

- name: mssqldb

mountPath: /var/opt/mssql

volumeClaimTemplates:

- metadata:

name: mssqldb

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: hostpath

resources:

requests:

storage: 500Mi

---

apiVersion: v1

kind: Service

metadata:

name: db-mssql

spec:

selector:

app: db

ports:

- protocol: TCP

port: 1433

targetPort: 1433

nodePort: 32433

type: NodePort

employeeapiwithsecret.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: empapi

spec:

selector:

matchLabels:

app: empapi

template:

metadata:

labels:

app: empapi

spec:

containers:

- name: empapi

image: shivaspk/empapi:latest

env:

- name: DbServer

value: "db-mssql"

- name: DbPort

value: "1433"

- name: DbUser

valueFrom:

secretKeyRef:

name: mssql-creds

key: username

- name: Password

valueFrom:

secretKeyRef:

name: mssql-creds

key: password

- name: Database

value: "EMSdb"

resources:

limits:

memory: "256Mi"

cpu: "500m"

---

apiVersion: v1

kind: Service

metadata:

name: empapi

spec:

type: NodePort

selector:

app: empapi

ports:

- port: 80

targetPort: 80

nodePort: 31700

CORS for REST APIs

Add the following dependency to the project

Microsoft.AspNetCore.Cors

In Startup.cs

In Services add the cors service

services.AddControllers();

**services.AddCors();**

Do this in configure method just after app.UseRouting();

**app.UseCors(builder =>**

**{**

**builder**

**.AllowAnyOrigin()**

**.AllowAnyMethod()**

**.AllowAnyHeader();**

**});**

To consume the REST API create a HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Employee API</title>

<script>

var requestOptions = {

method: 'GET',

redirect: 'follow'

};

fetch("http://localhost:5000/api/Pets", requestOptions)

.then(response => response.text())

.then(result => {console.log(result);

document.getElementById("output").innerHTML = result;

})

.catch(error => console.log('error', error));

</script>

</head>

<body>

<div id="output"></div>

</body>

</html>

GET and POST of Pets

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Pets API</title>

<script>

function getPets(){

var requestOptions = {

method: 'GET',

redirect: 'follow'

};

fetch("http://localhost:5000/api/Pets", requestOptions)

.then(response => response.text())

.then(result => {console.log(result);

document.getElementById("output").innerHTML = result;

})

.catch(error => console.log('error', error));

}

function addPet(){

var myHeaders = new Headers();

myHeaders.append("Content-Type", "application/json");

var body = JSON.stringify({

"petId": 0,

"name": "Rocky",

"breed": "Lab"

});

var requestOptions = {

method: 'POST',

headers: myHeaders,

body: body,

redirect: 'follow'

};

fetch("http://localhost:5000/api/Pets", requestOptions)

.then(response => response.text())

.then(result => {console.log(result);

document.getElementById("output").innerHTML = result;

})

.catch(error => console.log('error', error));

}

</script>

</head>

<body>

<button onclick="getPets()">Get Pets</button>

<button onclick="addPet()">Add Pet</button>

<div id="output"></div>

</body>

</html>

Cors for k8s deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: petcors

spec:

selector:

matchLabels:

app: petcors

template:

metadata:

labels:

app: petcors

spec:

containers:

- name: petcors

image: shivaspk/petsapi-probes-metrics-cors

resources:

limits:

memory: "256Mi"

cpu: "500m"

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: petservice

spec:

type: NodePort

selector:

app: petcors

ports:

- port: 80

targetPort: 80

nodePort: 31555

On the pets.html

Change the endpoints to: <http://localhost:31555/api/Pets>

Docker compose local testing

version: '3.4'

services:

ms-sql-server:

image: mcr.microsoft.com/mssql/server:2019-latest

environment:

ACCEPT\_EULA: "Y"

SA\_PASSWORD: "January2022"

MSSQL\_PID: Express

ports:

- "**14331**:1433"

emsapi-app:

build: .

ports:

- "9090:80"

environment:

**DbServer: "ms-sql-server"**

**DbPort: "1433"**

**DbUser: "SA"**

**Password: "January2022"**

**Database: "EMSdb"**

depends\_on:

ms-sql-server:

condition: service\_started