This article shows you a very basic deployment into multiple namespaces within a cluster.

Remember a namespace is a virtual cluster within a cluster.

Choice of creating dedicated cluster vs a namespace is a pure design decision and thus it is not good vs best but a practice that overall architect wants to follow.

Following are the benefits having dedicated cluster vs creating different namespaces in a single cluster where largescale applications are deployed together.

[Benefits]

Prerequisites for this lab:

* minikube – is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes. [https://minikube.sigs.k8s.io/docs/start/]. Follow the installation instructions directly from minikube official site [minikube is not currently offered into organization, but it is very lightweight and easy to get the concepts straight]
* node
* Docker

Usecase:

Create a simple hello world application into node, test it locally and deploy it to a container in a dedicated namespace called DEV environment. Once tested to development environment, deploy it to a new namespace called UAT.

Update the code and repeat the cycle. All this will be done using single container.

Conclusion: Containers gives benefits over traditional bare-metal hardware or VM by quickly spinning up the environment (namespace).

Note: This demonstration shows you very basic commands and how it interacts with the inputs. As a best practice you should create configuration files (yaml) for each configuration.

Let’s start…..

Create a simple node application

nano server.js

server.js

var http = require('http');

var handleRequest = function(request, response) {

console.log('Received request for URL: ' + request.url);

response.writeHead(200);

response.end('Hello Kubernetes World!');

};

var www = http.createServer(handleRequest);

[www.listen(8080)](http://www.listen(8080));

save and test

node server.js [will open a new browser and application webpage should display the ‘Hello Kubernetes World’]

Text

Description automatically generated

Let’s create a docker file (What is a docker file and what it does is beyond the scope of this lab)

echo 'FROM node:14

>

> EXPOSE 8080

>

> COPY server.js .

>

> CMD node server.js' >

Start the minikube

minikube start --cpus=4 --memory [memory, e.g. 1990]--driver=docker

You can define number of CPUs based on your docker configuration. If you get the error of exceeding the number of CPUs, then goto docker and change the default configuration (Docker > Preferences > Resources)

minikube start --cpus=4 --memory=1990 --driver=docker

😄 minikube v1.21.0 on Darwin 11.4

✨ Using the docker driver based on user configuration

👍 Starting control plane node minikube in cluster minikube

🚜 Pulling base image ...

🔥 Creating docker container (CPUs=4, Memory=1990MB) ...

🐳 Preparing Kubernetes v1.20.7 on Docker 20.10.7 ...

▪ Generating certificates and keys ...

▪ Booting up control plane ...

▪ Configuring RBAC rules ...

🔎 Verifying Kubernetes components...

▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5

🌟 Enabled addons: storage-provisioner, default-storageclass

❗ /usr/local/bin/kubectl is version 1.17.3, which may have incompatibilites with Kubernetes 1.20.7.

▪ Want kubectl v1.20.7? Try 'minikube kubectl -- get pods -A'

🏄 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

Since we are running docker driver, we need to set some environment variables

minikube docker-env

*export DOCKER\_TLS\_VERIFY="1"*

*export DOCKER\_HOST="tcp://127.0.0.1:55003"*

*export DOCKER\_CERT\_PATH="/Users/bipinkalawade/.minikube/certs"*

*export MINIKUBE\_ACTIVE\_DOCKERD="minikube"*

*# To point your shell to minikube's docker-daemon, run:*

*# eval $(minikube -p minikube docker-env)*

eval $(minikube -p minikube docker-env)

Checking the current namespace

k get namespaces

NAME STATUS AGE

default Active 67s

kube-node-lease Active 69s

kube-public Active 69s

kube-system Active 70s

(k = kubectl – you can alias the command in your .bash\_profile {alias k=kubectl})

Let’s create a namespace dev

kubectl create namespace dev

namespace/dev created

Let’s check if new namespace has been created.

kubectl get namespaces

NAME STATUS AGE

default Active 66m

dev Active 65m

istio-system Active 64m

kube-node-lease Active 66m

kube-public Active 66m

kube-system Active 66m

Let’s create a new image

docker build -t hello-kube:v1 .

*Sending build context to Docker daemon 3.072kB*

*Step 1/4 : FROM node:14*

*14: Pulling from library/node*

*bfde2ec33fbc: Pull complete*

*787f5e2f1047: Pull complete*

*7b6173a10eb8: Pull complete*

*dc05be471d51: Pull complete*

*55fab5cadd3c: Pull complete*

*bd821d20ef8c: Pull complete*

*fb63ab1f1488: Pull complete*

*809893a7b9ac: Pull complete*

*9e84a31df1a6: Pull complete*

*Digest: sha256:52fe2e3604ca3d7d0bc017c0d60df5baaf54c3e261bab8334b0657f923092523*

*Status: Downloaded newer image for node:14*

*---> c47df8afdb4a*

*Step 2/4 : EXPOSE 8080*

*---> Running in 04021739ac89*

*Removing intermediate container 04021739ac89*

*---> bd4307aac2e3*

*Step 3/4 : COPY server.js .*

*---> b3c0ccc976bd*

*Step 4/4 : CMD node server.js*

*---> Running in 3f6644d8923a*

*Removing intermediate container 3f6644d8923a*

*---> d35d6370b44f*

*Successfully built d35d6370b44f*

*Successfully tagged hello-kube:v1*

docker images

*REPOSITORY TAG IMAGE ID CREATED SIZE*

*hello-kube v1 d35d6370b44f 5 seconds ago 944MB*

*node 14 c47df8afdb4a 7 days ago 944MB*

*istio/proxyv2 1.10.1 5c66e8ac89a7 2 weeks ago 282MB*

*istio/pilot 1.10.1 07d6b563f74b 2 weeks ago 217MB*

*quay.io/kiali/kiali v1.34 1d3ab1649f0b 5 weeks ago 194MB*

*k8s.gcr.io/kube-proxy v1.20.7 ff54c88b8ecf 5 weeks ago 118MB*

*k8s.gcr.io/kube-controller-manager v1.20.7 22d1a2072ec7 5 weeks ago 116MB*

*k8s.gcr.io/kube-apiserver v1.20.7 034671b24f0f 5 weeks ago 122MB*

*k8s.gcr.io/kube-scheduler v1.20.7 38f903b54010 5 weeks ago 47.3MB*

*gcr.io/k8s-minikube/storage-provisioner v5 6e38f40d628d 2 months ago 31.5MB*

*kubernetesui/dashboard v2.1.0 9a07b5b4bfac 6 months ago 226MB*

*k8s.gcr.io/etcd 3.4.13-0 0369cf4303ff 9 months ago 253MB*

*k8s.gcr.io/coredns 1.7.0 bfe3a36ebd25 12 months ago 45.2MB*

*kubernetesui/metrics-scraper v1.0.4 86262685d9ab 15 months ago 36.9MB*

*k8s.gcr.io/pause 3.2 80d28bedfe5d 16 months ago 683kB*

Let’s create a deployment for our application and deploy it into namespace dev

kubectl run hello-kube --image=hello-kube:v1 --image-pull-policy=Never --port=8080 --namespace dev

*kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.*

*deployment.apps/hello-kube created*

[Ignore the warning for now]

kubectl get deployments

No resources found in default namespace.

Since the “default” namespace is a default namespace there are no resources found. So we have to specify namespace parameter.

kubectl get deployments --namespace dev

*NAME READY UP-TO-DATE AVAILABLE AGE*

*hello-kube 1/1 1 1 19s*

Let’s expose the service to outer world (remember to provide the namespace, we can set the default namespace but for this lab will keep passing that as a parameter)

kubectl expose deployment hello-kube --type=LoadBalancer --namespace dev

*service/hello-kube exposed*

Let’s verify the service and describe it.

kubectl get services --namespace dev

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*hello-kube LoadBalancer 10.102.36.89 <pending> 8080:30926/TCP 89m*

kubectl describe service --namespace dev

*Name: hello-kube*

*Namespace: dev*

*Labels: run=hello-kube*

*Annotations: <none>*

*Selector: run=hello-kube*

*Type: LoadBalancer*

*IP: 10.102.36.89*

*Port: <unset> 8080/TCP*

*TargetPort: 8080/TCP*

*NodePort: <unset> 30926/TCP*

*Endpoints: 172.17.0.7:8080*

*Session Affinity: None*

*External Traffic Policy: Cluster*

*Events: <none>*

Let’s run the service, it will open into default browser

minikube service hello-kube --namespace dev

|-----------|------------|-------------|---------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|------------|-------------|---------------------------|

| dev | hello-kube | 8080 | http://192.168.49.2:30926 |

|-----------|------------|-------------|---------------------------|

🏃Starting tunnel for service hello-kube.

|-----------|------------|-------------|------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|------------|-------------|------------------------|

| dev | hello-kube | | http://127.0.0.1:51979 |

|-----------|------------|-------------|------------------------|

🎉Opening service dev/hello-kube in default browser...

❗Because you are using a Docker driver on darwin, the terminal needs to be open to run it.

Graphical user interface, text, application

Description automatically generated

Remember the port that was allocated by Kubernetes service (not the 8080)

Once the results are satisfactory then now it’s time to move the details to UAT.

First create a namespace UAT

kubectl create namespace uat

namespace/uat created

create a deployment for uat

kubectl run hello-kube --image=hello-kube:v1 --image-pull-policy=Never --port=8080 --namespace uat

*kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.*

*deployment.apps/hello-kube created*

[ignore the warning]

Let’s expose the UAT service

kubectl expose deployment hello-kube --type=LoadBalancer --namespace uat

service/hello-kube exposed

Let’s analyze the describe the service.

kubectl get service --namespace uat

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

hello-kube LoadBalancer 10.101.185.107 <pending> 8080:30911/TCP 46s

kubectl describe service --namespace uat

*Name: hello-kube*

*Namespace: uat*

*Labels: run=hello-kube*

*Annotations: <none>*

*Selector: run=hello-kube*

*Type: LoadBalancer*

*IP: 10.101.185.107*

*Port: <unset> 8080/TCP*

*TargetPort: 8080/TCP*

*NodePort: <unset> 30911/TCP*

*Endpoints: 172.17.0.9:8080*

*Session Affinity: None*

*External Traffic Policy: Cluster*

*Events: <none>*

Let’s run the service in uat

minikube service hello-kube --namespace uat

|-----------|------------|-------------|---------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|------------|-------------|---------------------------|

| uat | hello-kube | 8080 | http://192.168.49.2:30911 |

|-----------|------------|-------------|---------------------------|

🏃 Starting tunnel for service hello-kube.

|-----------|------------|-------------|------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|------------|-------------|------------------------|

| uat | hello-kube | | http://127.0.0.1:60480 |

|-----------|------------|-------------|------------------------|

🎉 Opening service uat/hello-kube in default browser...

❗ Because you are using a Docker driver on darwin, the terminal needs to be open to run it.

Rectangle

Description automatically generated with medium confidence

Confirm the associated port for the uat service, which suggests that UAT is now running the same version of code.

Now, let’s move to some updates to our application.

nano server.js

changes the response, save it and test it.

Graphical user interface, text, application, email

Description automatically generated

Let’s create a new image with version 2

docker build -t hello-kube:v2 .

Sending build context to Docker daemon 3.072kB

Step 1/4 : FROM node:14

14: Pulling from library/node

bfde2ec33fbc: Pull complete

787f5e2f1047: Pull complete

7b6173a10eb8: Pull complete

dc05be471d51: Pull complete

55fab5cadd3c: Pull complete

bd821d20ef8c: Pull complete

fb63ab1f1488: Pull complete

809893a7b9ac: Pull complete

9e84a31df1a6: Pull complete

Digest: sha256:52fe2e3604ca3d7d0bc017c0d60df5baaf54c3e261bab8334b0657f923092523

Status: Downloaded newer image for node:14

---> c47df8afdb4a

Step 2/4 : EXPOSE 8080

---> Running in aa86faad0743

Removing intermediate container aa86faad0743

---> 0ec1fa5bde64

Step 3/4 : COPY server.js .

---> 36ca4fc912a0

Step 4/4 : CMD node server.js

---> Running in 1556bddb666e

Removing intermediate container 1556bddb666e

---> b956c9541f7b

Successfully built b956c9541f7b

Successfully tagged hello-kube:v2

Verify the image is now created

docker images

*REPOSITORY TAG IMAGE ID CREATED SIZE*

*hello-kube v2 7a785422b733 6 seconds ago 944MB*

*hello-kube v1 c540ce59bca7 7 minutes ago 944MB*

*node 14 c47df8afdb4a 7 days ago 944MB*

*k8s.gcr.io/kube-proxy v1.20.7 ff54c88b8ecf 5 weeks ago 118MB*

*k8s.gcr.io/kube-controller-manager v1.20.7 22d1a2072ec7 5 weeks ago 116MB*

*k8s.gcr.io/kube-apiserver v1.20.7 034671b24f0f 5 weeks ago 122MB*

*k8s.gcr.io/kube-scheduler v1.20.7 38f903b54010 5 weeks ago 47.3MB*

*gcr.io/k8s-minikube/storage-provisioner v5 6e38f40d628d 2 months ago 31.5MB*

*kubernetesui/dashboard v2.1.0 9a07b5b4bfac 6 months ago 226MB*

*k8s.gcr.io/etcd 3.4.13-0 0369cf4303ff 9 months ago 253MB*

*k8s.gcr.io/coredns 1.7.0 bfe3a36ebd25 12 months ago 45.2MB*

*kubernetesui/metrics-scraper v1.0.4 86262685d9ab 15 months ago 36.9MB*

*k8s.gcr.io/pause 3.2 80d28bedfe5d 16 months ago 683kB*

Verify the container name so we can deploy it to the same container.

kubectl describe deploy hello-kube --namespace dev | grep Name

*Name: hello-kube*

*Namespace: dev*

Now, set the image to newer version and deploy to the container where namespace is dev.

k set image deployment/hello-kube hello-kube=hello-kube:v2 --namespace dev

deployment.apps/hello-kube image updated

[k is just an alias to Kubernetes which is set to my bash\_profile]

Let’s test it

set image deployment/hello-kube hello-kube=hello-kube:v2 --namespace dev

deployment.apps/hello-kube image updated

ToyTron-Machine360:hello-kube bipinkalawade$ minikube service hello-kube --namespace dev

|-----------|------------|-------------|---------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|------------|-------------|---------------------------|

| dev | hello-kube | 8080 | http://192.168.49.2:32056 |

|-----------|------------|-------------|---------------------------|

🏃 Starting tunnel for service hello-kube.

|-----------|------------|-------------|------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|------------|-------------|------------------------|

| dev | hello-kube | | http://127.0.0.1:55835 |

|-----------|------------|-------------|------------------------|

🎉 Opening service dev/hello-kube in default browser...

❗ Because you are using a Docker driver on darwin, the terminal needs to be open to run it.

Graphical user interface, text, application

Description automatically generated

This concluded the lab. What we have seen here is how quickly and effectively we can deploy our code and test the same onto different environment.

Now, there is lot going behind the scene such as service management, resource (cpu, memory, i/o) allocation. This should be considered in to a design.

NEXT >>

Deploy Istio onto minikube and enable kiali and other dashboard.

Installation of Istio is a separate process and two useful links to follow. This lab will help to get going with both (post installation and setup)

Download Istio [https://istio.io/latest/docs/setup/getting-started/#download] and follow the steps. Make sure you add Istio into your existing PATH as you can run istioctl from different terminal windows.

Istion setup [https://istio.io/latest/docs/setup/platform-setup/minikube/]. This setup is specific to minikube but there are articles where we can enable the same for other services.

We will be using profile demo, some articles suggests that demo profile should give kiali and Prometheus by default. I had to enable it manually and shown in the steps below.

We are going to apply the same to the namespace we created recently, instead of default.

istioctl manifest apply --set profile=demo --namespace dev

*This will install the Istio 1.10.1 demo profile with ["Istio core" "Istiod" "Ingress gateways" "Egress gateways"] components into the cluster. Proceed? (y/N) y*

*✔ Istio core installed*

*✔ Istiod installed*

*✔ Ingress gateways installed*

*✔ Egress gateways installed*

*✔ Installation complete Thank you for installing Istio 1.10. Please take a few minutes to tell us about your install/upgrade experience! https://forms.gle/KjkrDnMPByq7akrYA*

Make sure pod is running.

kubectl get pods --namespace dev

*NAME READY STATUS RESTARTS AGE*

*hello-kube-6846c8bc54-bjkwl 1/1 Running 0 14m*

Let’s label our namespace

kubectl label namespace dev istio-injection=enabled

*namespace/dev labeled*

Let’s review this.

kubectl get namespace dev --show-labels

*NAME STATUS AGE LABELS*

*dev Active 27m istio-injection=enabled*

Istio comes with some sample installations addons. These are not meant to use in production as these are the samples only and more sophisticated versions can made for real deployment (dev to prod).

Let’s deploy Kiali. [Kiali is a management console for an Istio-based service mesh.]

kubectl apply -f ${ISTIO\_HOME}/samples/addons/kiali.yaml

customresourcedefinition.apiextensions.k8s.io/monitoringdashboards.monitoring.kiali.io created

serviceaccount/kiali created

*configmap/kiali created*

*clusterrole.rbac.authorization.k8s.io/kiali-viewer created*

*clusterrole.rbac.authorization.k8s.io/kiali created*

*clusterrolebinding.rbac.authorization.k8s.io/kiali created*

*role.rbac.authorization.k8s.io/kiali-controlplane created*

*rolebinding.rbac.authorization.k8s.io/kiali-controlplane created*

*service/kiali created*

*deployment.apps/kiali created*

*monitoringdashboard.monitoring.kiali.io/envoy created*

*monitoringdashboard.monitoring.kiali.io/go created*

*monitoringdashboard.monitoring.kiali.io/kiali created*

*monitoringdashboard.monitoring.kiali.io/micrometer-1.0.6-jvm-pool created*

*monitoringdashboard.monitoring.kiali.io/micrometer-1.0.6-jvm created*

*monitoringdashboard.monitoring.kiali.io/micrometer-1.1-jvm created*

*monitoringdashboard.monitoring.kiali.io/microprofile-1.1 created*

*monitoringdashboard.monitoring.kiali.io/microprofile-x.y created*

*monitoringdashboard.monitoring.kiali.io/nodejs created*

*monitoringdashboard.monitoring.kiali.io/quarkus created*

*monitoringdashboard.monitoring.kiali.io/springboot-jvm-pool created*

*monitoringdashboard.monitoring.kiali.io/springboot-jvm created*

*monitoringdashboard.monitoring.kiali.io/springboot-tomcat created*

*monitoringdashboard.monitoring.kiali.io/thorntail created*

*monitoringdashboard.monitoring.kiali.io/tomcat created*

*monitoringdashboard.monitoring.kiali.io/vertx-client created*

*monitoringdashboard.monitoring.kiali.io/vertx-eventbus created*

*monitoringdashboard.monitoring.kiali.io/vertx-jvm created*

*monitoringdashboard.monitoring.kiali.io/vertx-pool created*

*monitoringdashboard.monitoring.kiali.io/vertx-server created*

Since kiali use Prometheus as well, we need to install that as well. Otherwise, the dashboard will give some errors.

Without Prometheus:

istioctl dashboard kiali

*http://localhost:20001/kiali*

Graphical user interface, application

Description automatically generated

Let’s apply that configuration as well. Again this is for sample only and not meant to use for production.

kubectl apply -f ${ISTIO\_HOME}/samples/addons/prometheus.yaml

*serviceaccount/prometheus created*

*configmap/prometheus created*

*clusterrole.rbac.authorization.k8s.io/prometheus created*

*clusterrolebinding.rbac.authorization.k8s.io/prometheus created*

*service/prometheus created*

*deployment.apps/prometheus created*

Let’s try one more time.

Graphical user interface, application

Description automatically generated

It’s a great place to start working with your environment. As you can see all namespaces we have created are available there. You can explore observability.

Graphical user interface, application

Description automatically generated

Since we are not generating any traffic, there are some details missing. But it also give some details about pod and what is missing.

Graphical user interface, application

Description automatically generated

Common Errors I encountered:

Broken pipe – terminate the current session ^c and run istioctl dashboard command again.

Port 8080 being used – Find the process running the port 8080, then stop it or kill it.

Some other useful resources

Kubernetes cheatsheet [https://kubernetes.io/docs/reference/kubectl/cheatsheet/]