IBM Cloud private 2.1.0.1

Proof of Technology

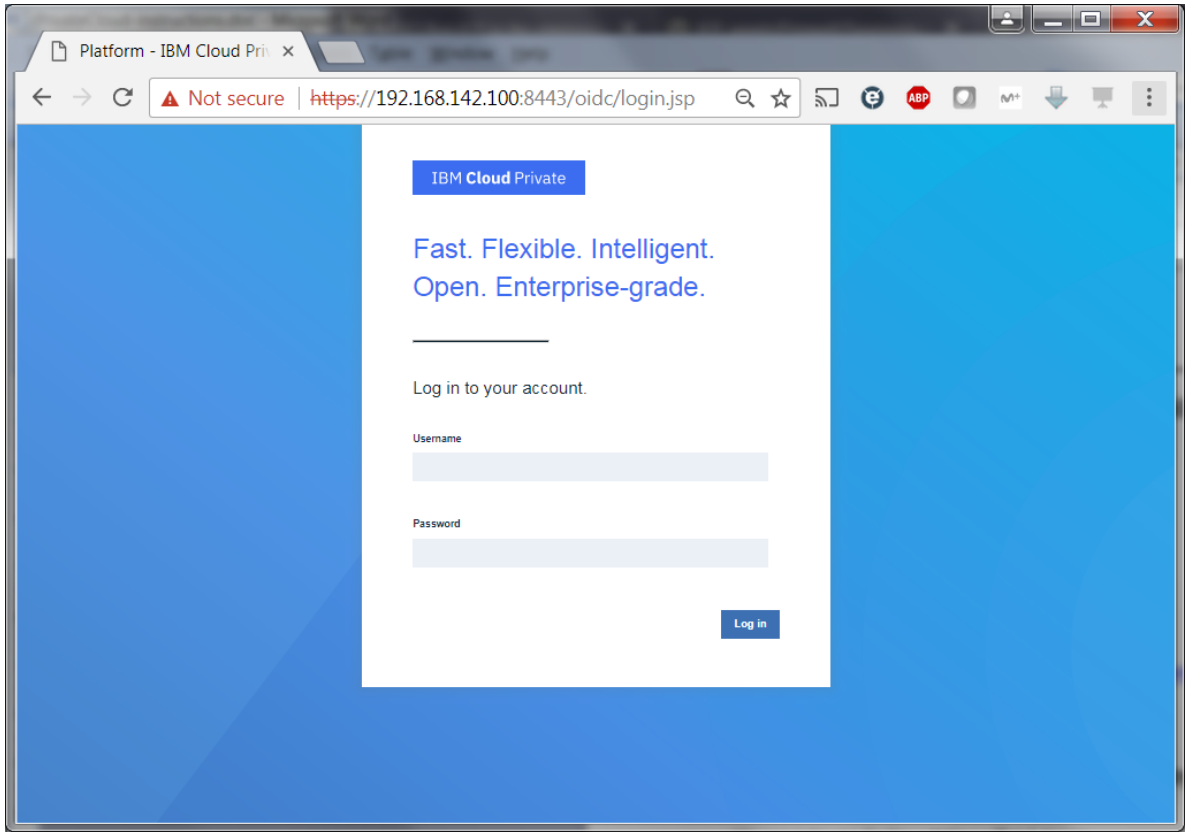
Index

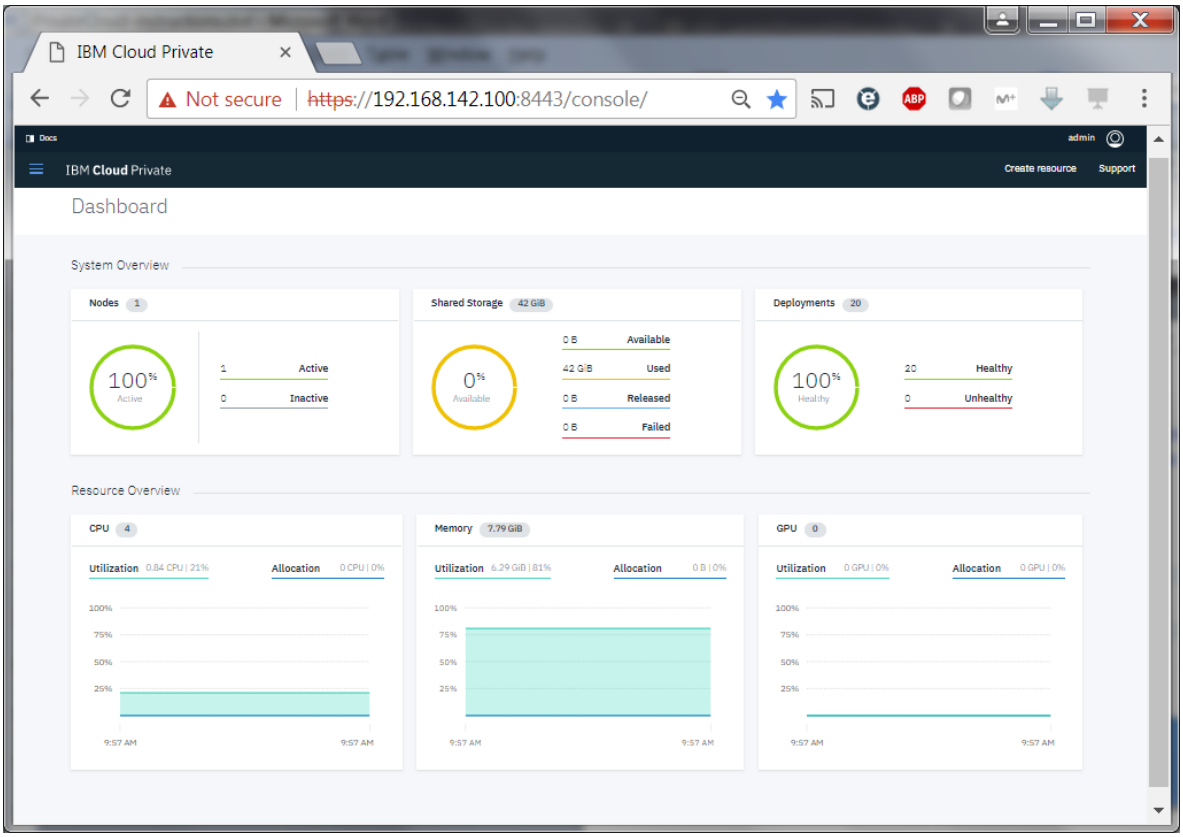
Lab 1: Deploy and expose an Application using IBM Cloud Private console.

IBM Cloud Private host: 192.168.142.100.
Master, Worker and Proxy are in the same computer.

1. Login to the console

Using your browser access to https://192.168.142.100:8443 and login with admin / admin



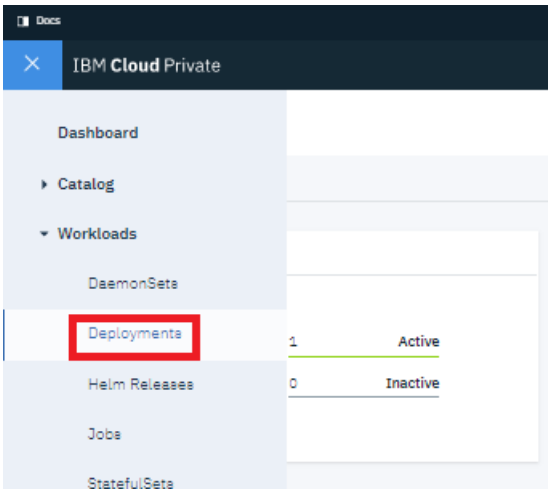


2. Deploy a new application (K8s Deployment)

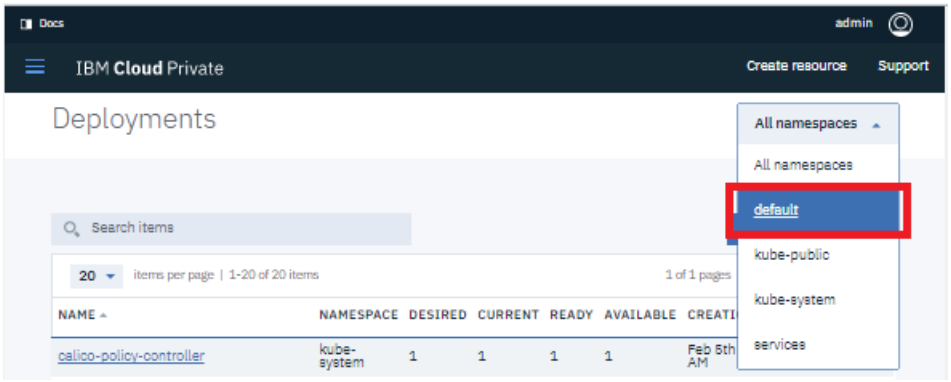
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

We are going to deploy an NGinx official image.

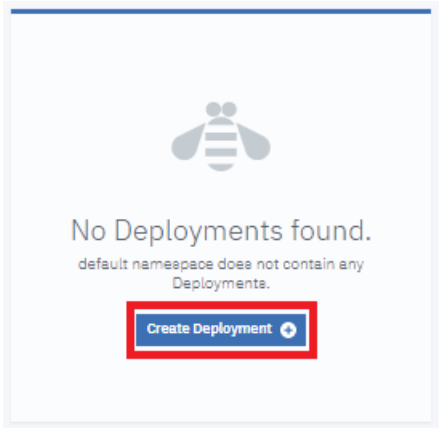
Click on “Deployments”



Select namespace default



And click “Create deployment”



Enter the following information

General:

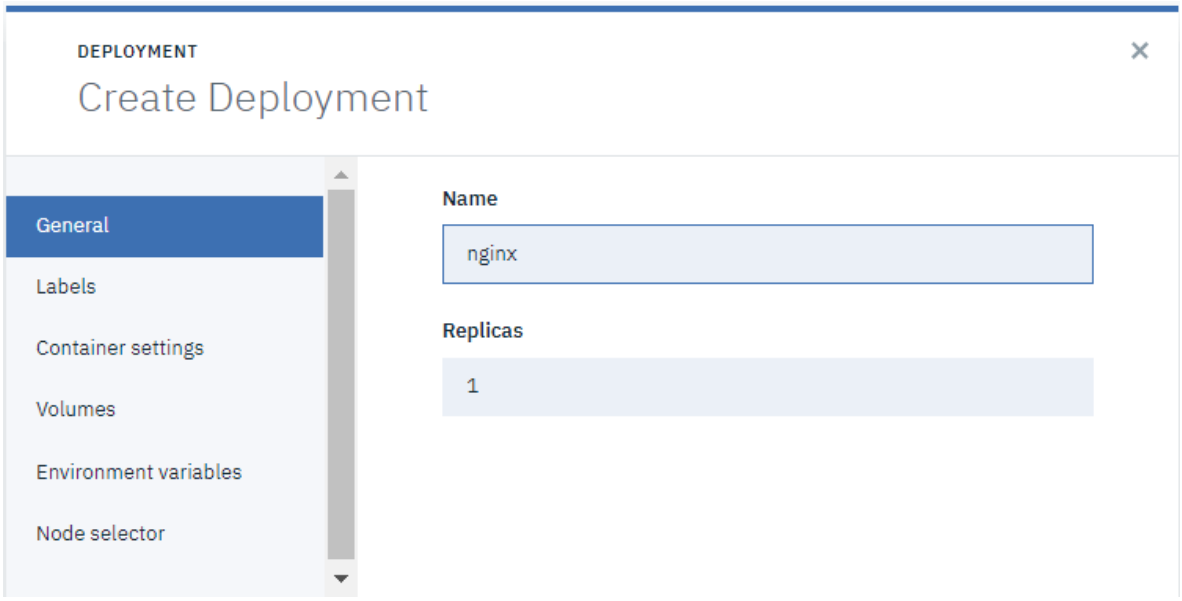
Name: nginx (lower letters)

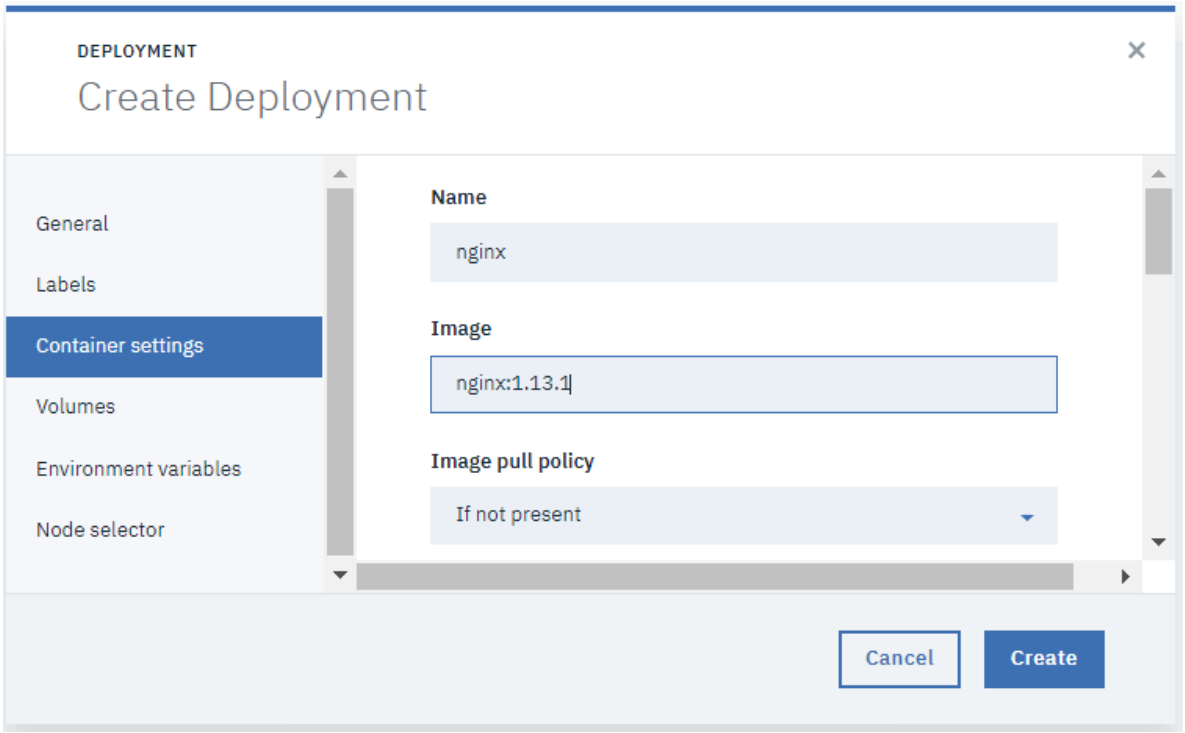
Replicas: 1

Container Settings:

Name: nginx

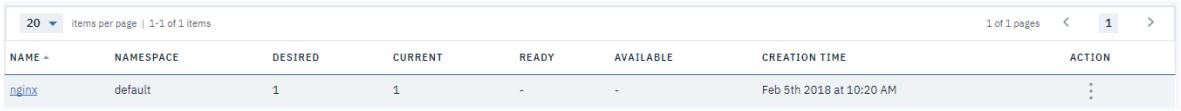
Image: nginx:1.13.1



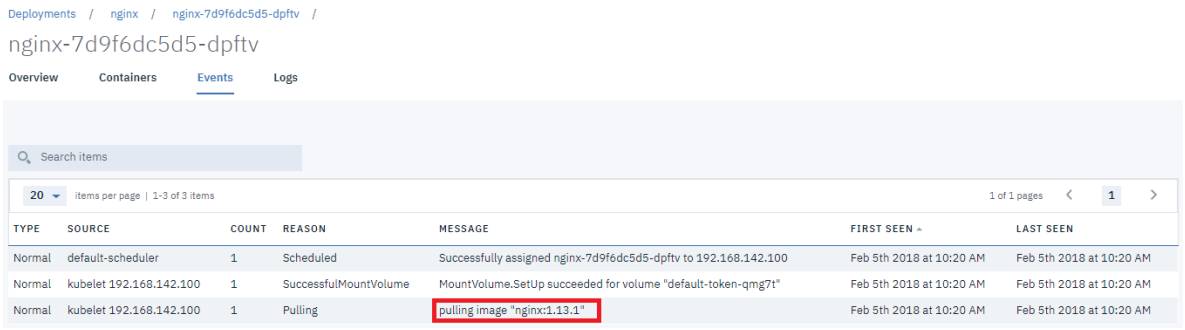


Click “Create”.

The first time the image nginx must be pulled from docker hub and it takes a bit of time to deploy the Application (Deployment)



NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
nginx	default	1	1	-	-	Feb 5th 2018 at 10:20 AM	



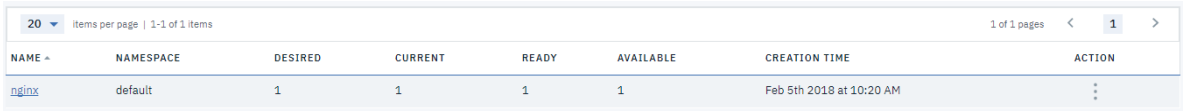
Deployments / nginx / nginx-7d9f6dc5d5-dpftv /

nginx-7d9f6dc5d5-dpftv

Overview Containers Events Logs

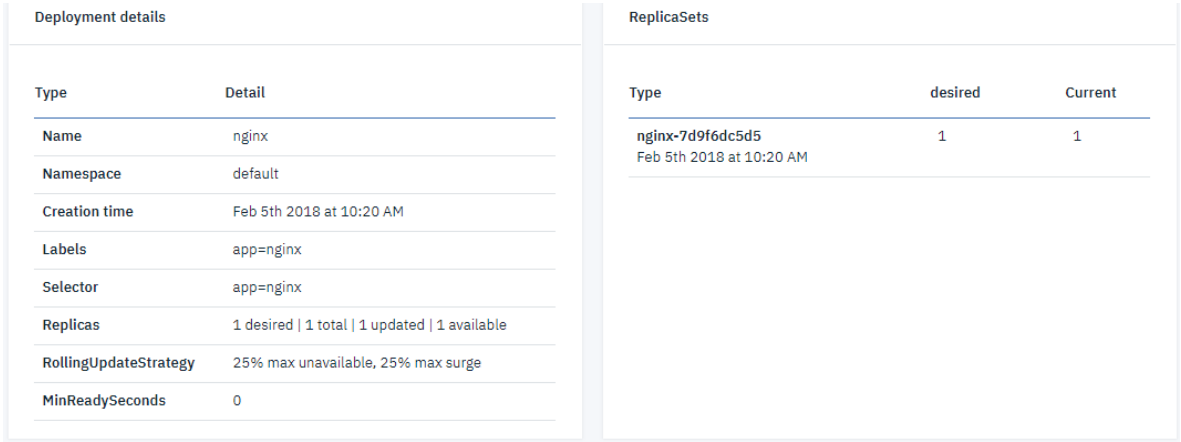
Search items

TYPE	SOURCE	COUNT	REASON	MESSAGE	FIRST SEEN	LAST SEEN
Normal	default-scheduler	1	Scheduled	Successfully assigned nginx-7d9f6dc5d5-dpftv to 192.168.142.100	Feb 5th 2018 at 10:20 AM	Feb 5th 2018 at 10:20 AM
Normal	kubelet 192.168.142.100	1	SuccessfulMountVolume	MountVolume.SetUp succeeded for volume "default-token-qmg7t"	Feb 5th 2018 at 10:20 AM	Feb 5th 2018 at 10:20 AM
Normal	kubelet 192.168.142.100	1	Pulling	pulling image "nginx:1.13.1"	Feb 5th 2018 at 10:20 AM	Feb 5th 2018 at 10:20 AM



NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
nginx	default	1	1	1	1	Feb 5th 2018 at 10:20 AM	

Clicking in the application name nginx the console will show you information about the application, replica set and pods



Deployment details

Type	Detail
Name	nginx
Namespace	default
Creation time	Feb 5th 2018 at 10:20 AM
Labels	app=nginx
Selector	app=nginx
Replicas	1 desired | 1 total | 1 updated | 1 available
RollingUpdateStrategy	25% max unavailable, 25% max surge
MinReadySeconds	0

ReplicaSets

Type	desired	Current
nginx-7d9f6dc5d5 Feb 5th 2018 at 10:20 AM	1	1

Pods

Search items

NAME	NAMESPACE	STATUS	HOST IP	POD IP	READY	START TIME	ACTION
nginx-7d9f6dc5d5-dpftv	default	Running	192.168.142.100	10.1.90.220	1/1	Feb 5th 2018 at 10:20 AM	

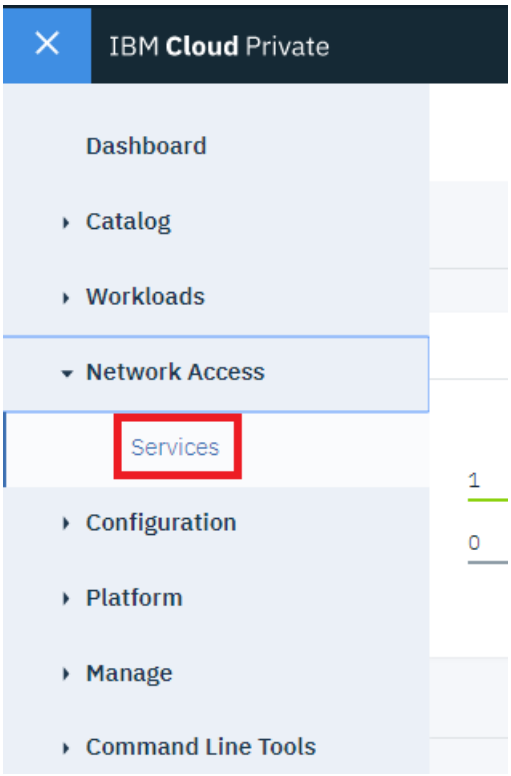
We will use latter the label “app=nginx”, also take note of the pod IP list at the bottom, in this case 10.1.90.220

3. Expose the application (K8s Service)

3.1 ClusterIP

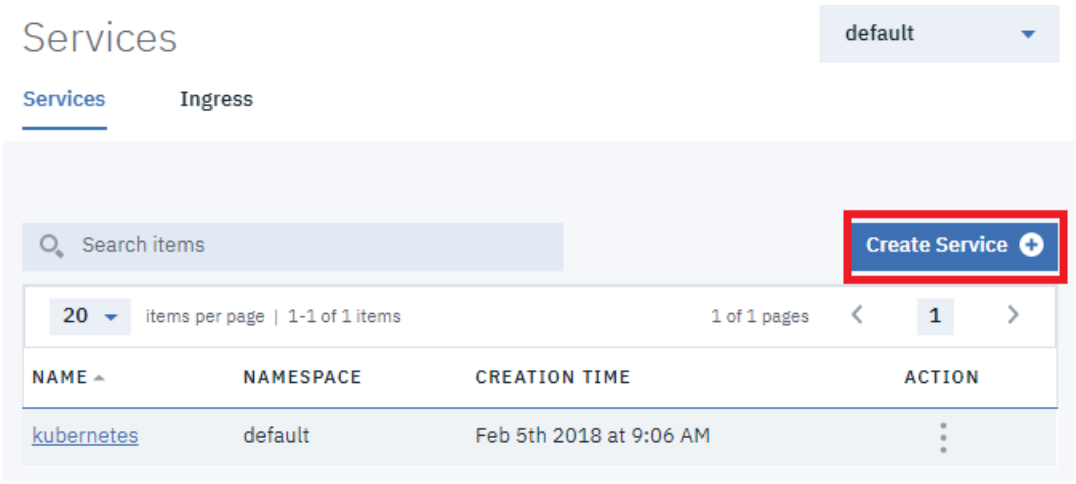
https://kubernetes.io/docs/concepts/services-networking/service/

This application is not exposed yet, now we are going to expose it. Go to Services:



First we are going to expose the application using a kubernetes Service of type “ClusterIP”.

Click on “Create Service”.



Enter the following information

General:

Name: nginxsrv

Type: ClusterIP

Ports:

Name: http

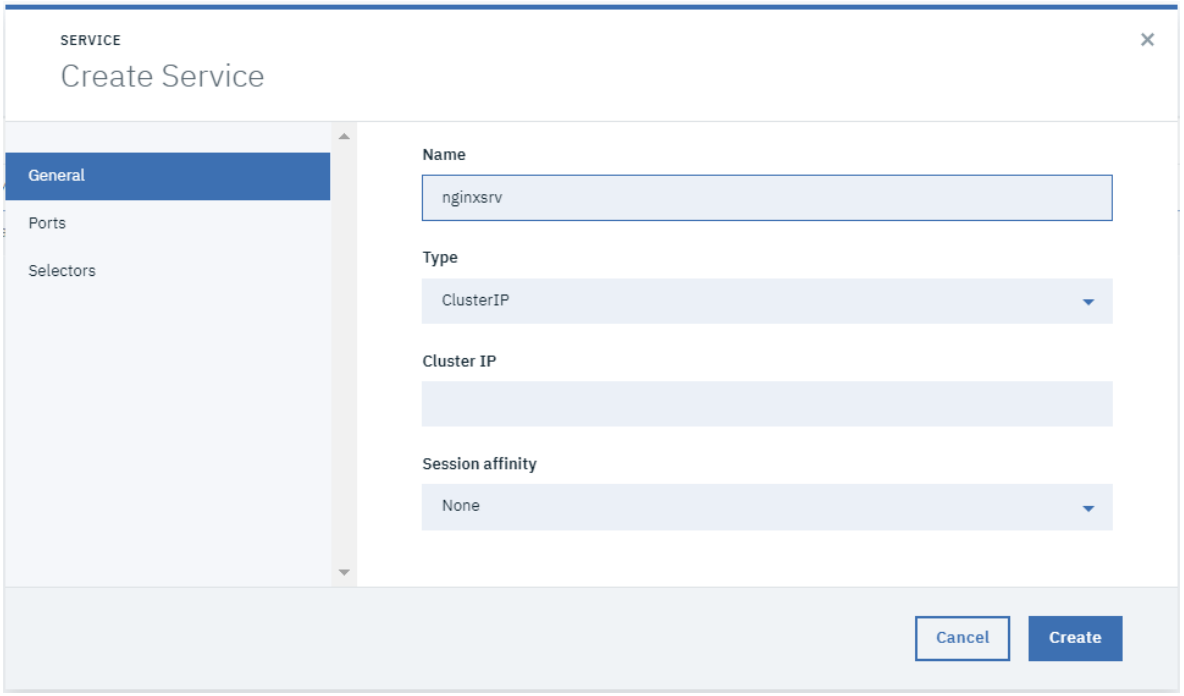
Port: 80

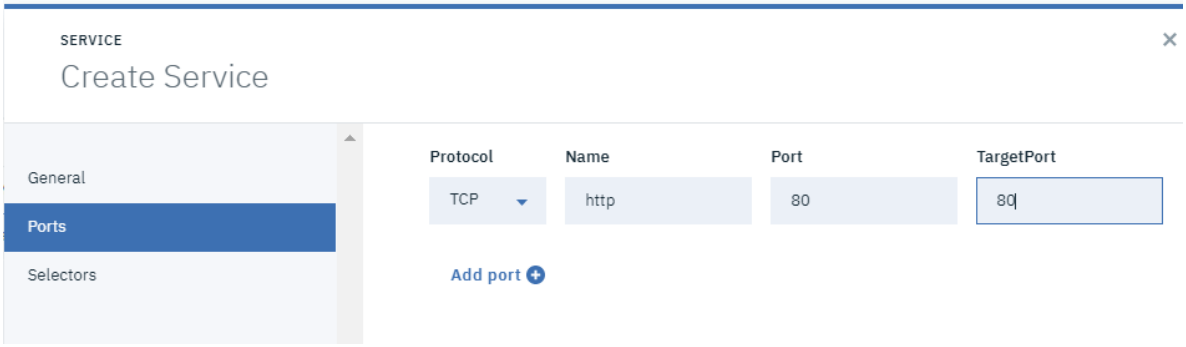
TargetPort: 80

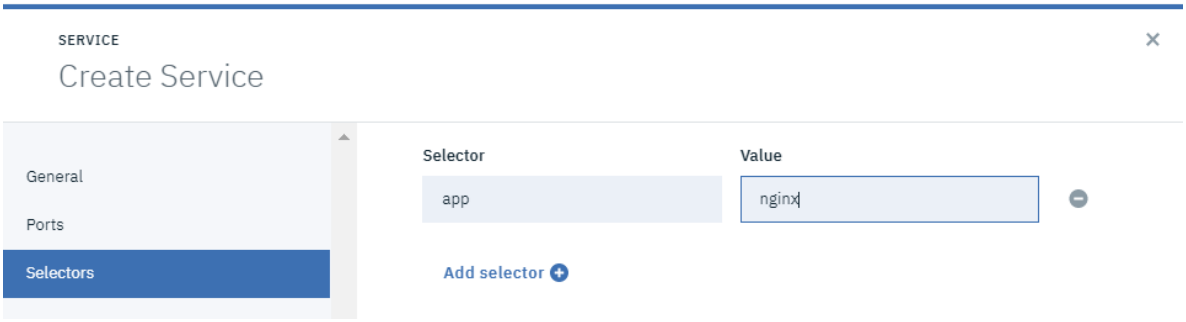
Selectors:

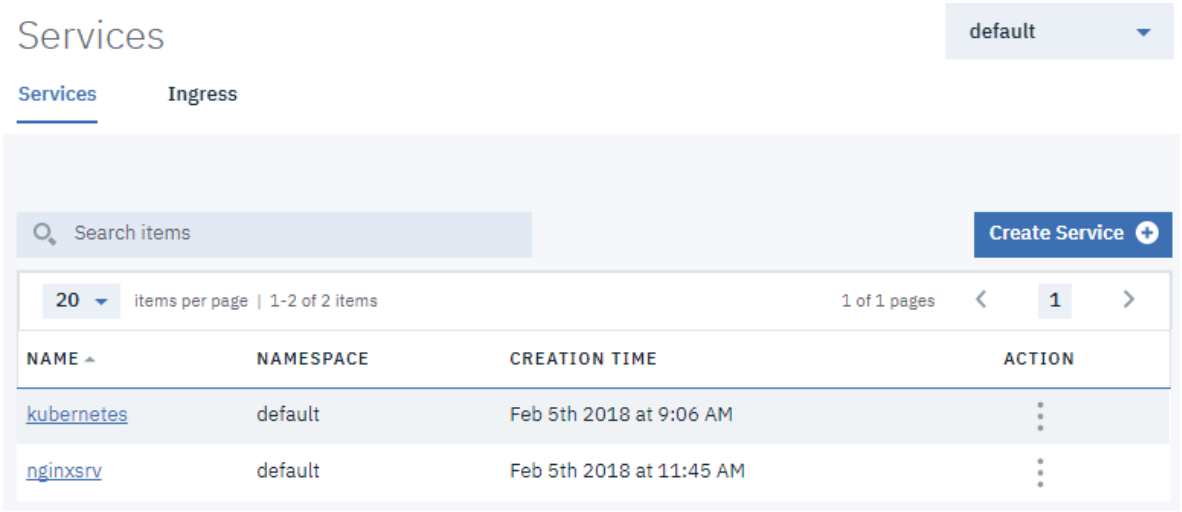
Selector: app

Value: nginx

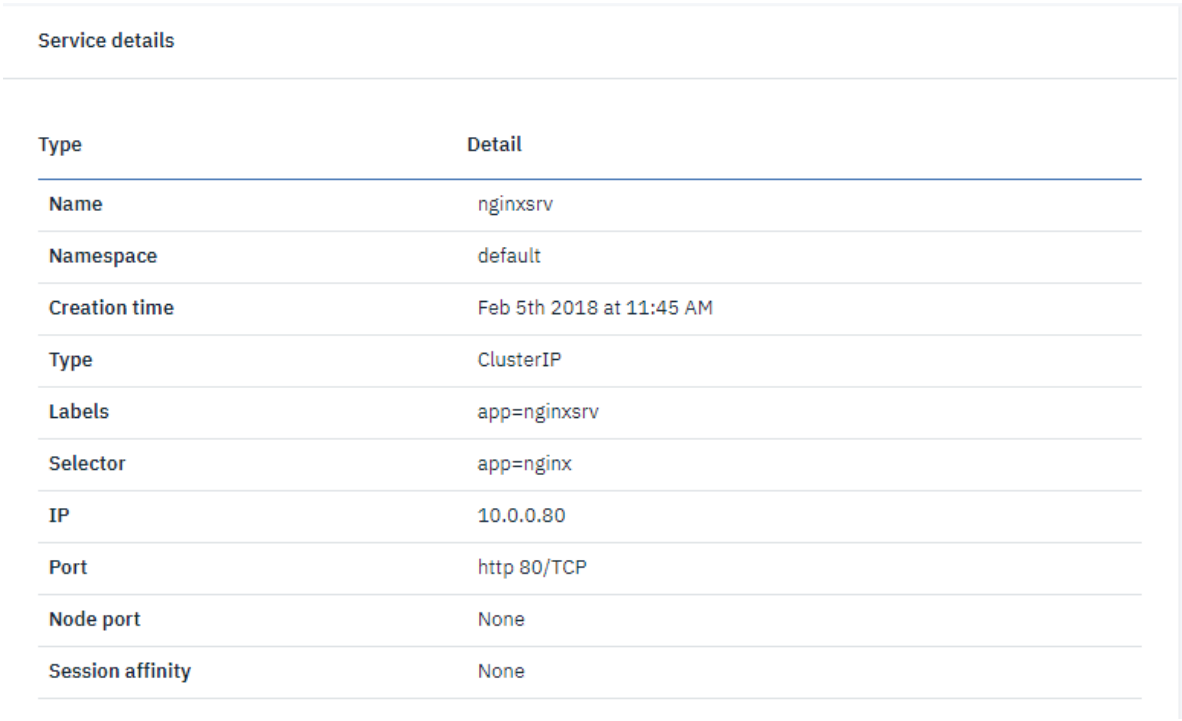

SERVICE
Create Service
General
Ports
Selectors
Name
nginxsrv
Type
ClusterIP
Cluster IP
Session affinity
None
Cancel
Create


SERVICE
Create Service
General
Ports
Selectors
Protocol
Name
Port
TargetPort
TCP
http
80
80
Add port


SERVICE
Create Service
General
Ports
Selectors
Selector
Value
app
nginx
Add selector



If we click on the service “nginxsrv” we get the details:



We will work with the command line latter. If we get the details using the command line we get more information as for example the IPs exposed by the service.

kubectl describe service nginxsrv
Name: nginxsrv
Selector: app=nginx
Type: ClusterIP
IP: 10.0.0.80
Port: http 80/TCP

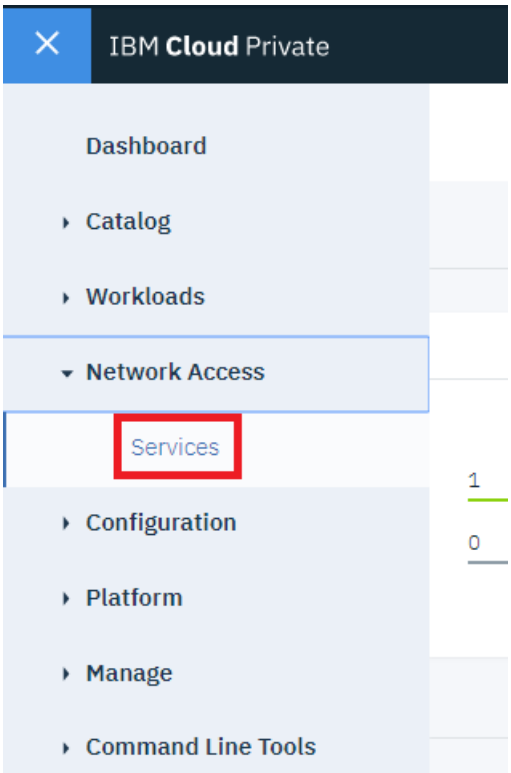
TargetPort: 80/TCP
Endpoints: 10.1.90.220:80
...

As you can see the service exposes by a unique IP 10.0.0.80 or hostname “nginxsrv” the application/deployment “app=nginx” in the IP 10.1.90.220, in case we scale or restart the application the service ip/hostame will be the same and will automatically update the Endpoint IPs.

The type of service “ClusterIP” is used for communication between applications inside the cluster so our application is not public yet. We have two options, create a service of type NodePort instead of type ClusterIP or create an Ingress kubernetes object (https://kubernetes.io/docs/concepts/services-networking/ingress/)

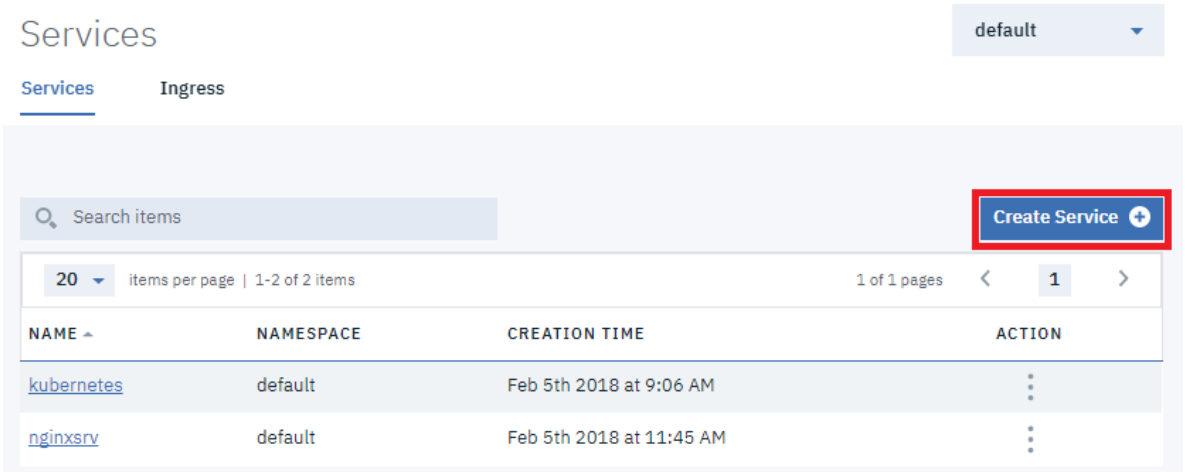
3.2 NodePort

Go to Services:



Now we are going to expose the application/deployment as a service of type “NodePort”.

Click on “Create Service”.



Enter the following information

General:

Name: nginxsrv2

Type: NodePort

Ports:

Name: http

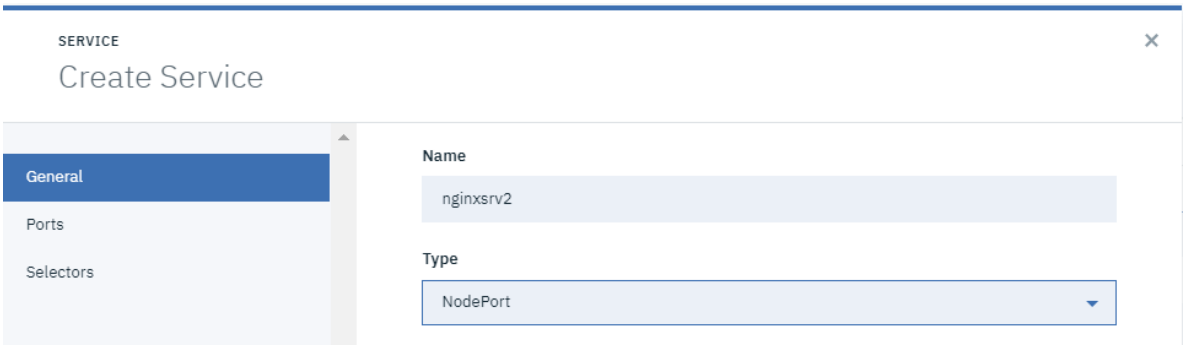
Port: 80

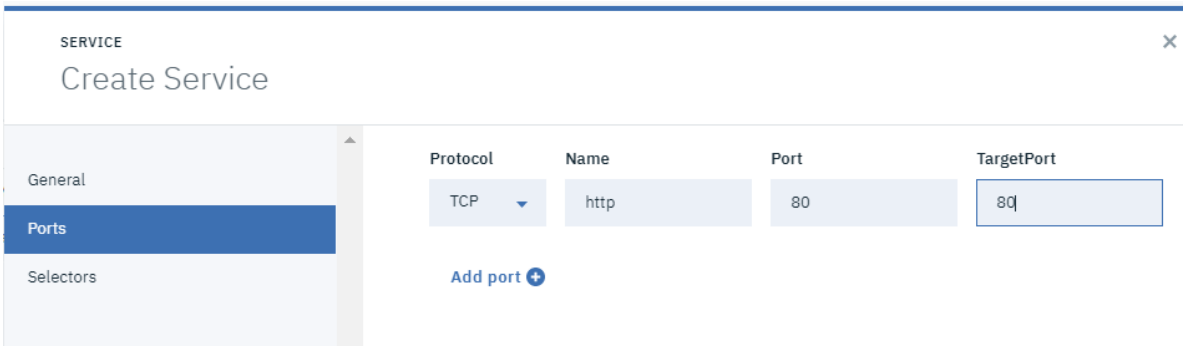
TargetPort: 80

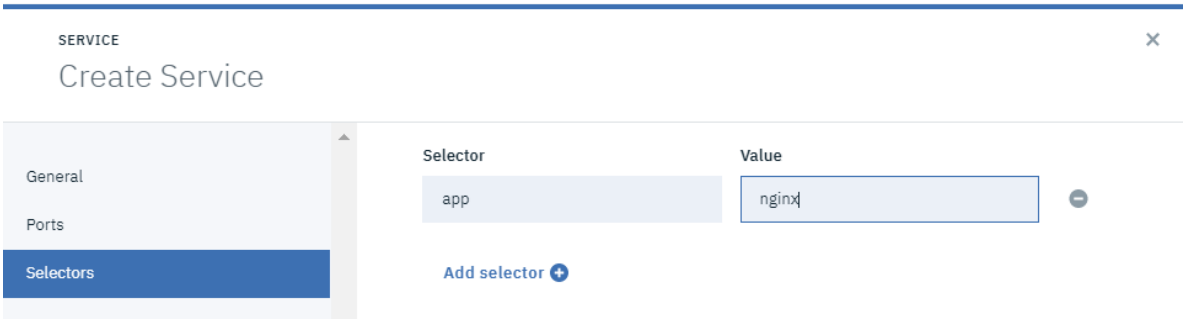
Selectors:

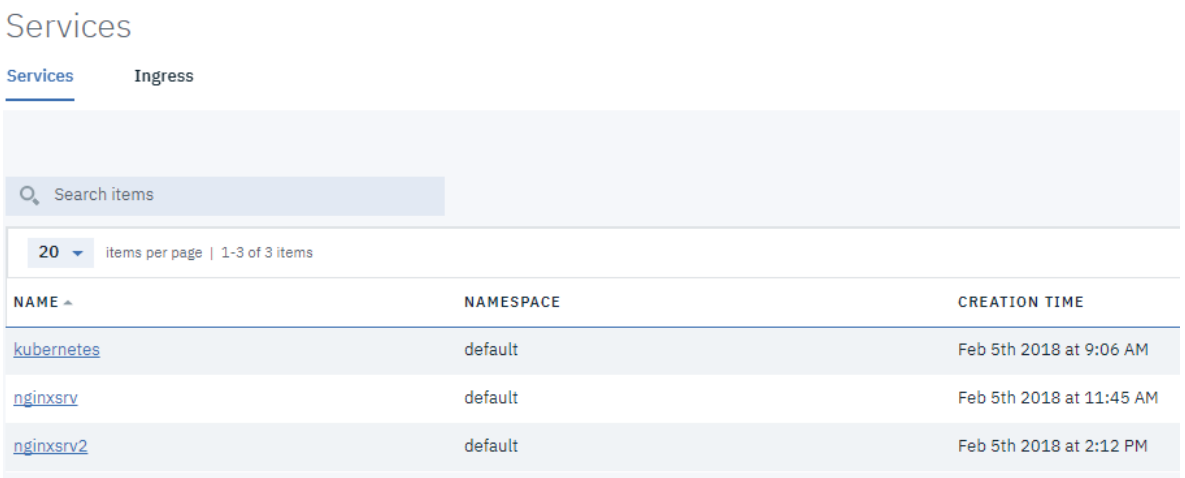
Selector: app

Value: nginx









If we click on the service “nginxsrv2” we get the details:

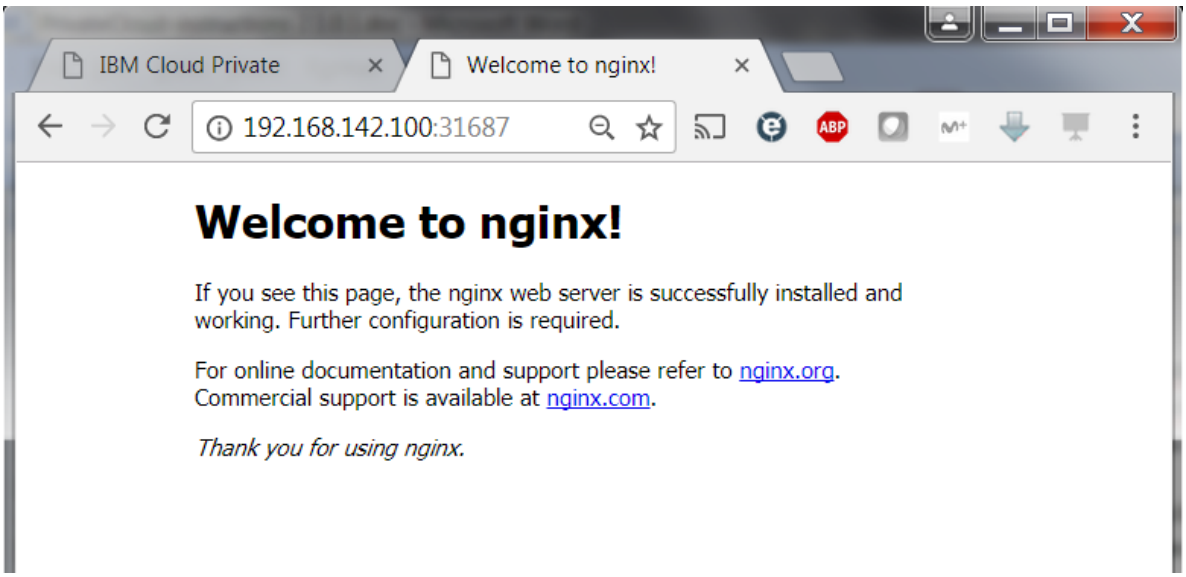
Service details

Type	Detail
Name	nginxsrv2
Namespace	default
Creation time	Feb 5th 2018 at 2:12 PM
Type	NodePort
Labels	app=nginxsrv2
Selector	app=nginx
IP	10.0.0.227
Port	http 80/TCP
Node port	http 31687/TCP
Session affinity	None

A service of type NodePort is a ClusterIP service plus a NodePort. Now the property “Node port” is configured. With NodePort kubernetes enables on the worker and proxy nodes the port “NodePort” to access the Application/Deployment from outside the cluster.

Now you can open a browser and set the URL http://<worker/proxy node>:<node port> and kubernetes will balance the traffic to the internal application/deployment ip:port.

http://192.168.142.100:31687 -> http://10.1.90.220:80



With NodePort we get TCP balancing but no additional features.

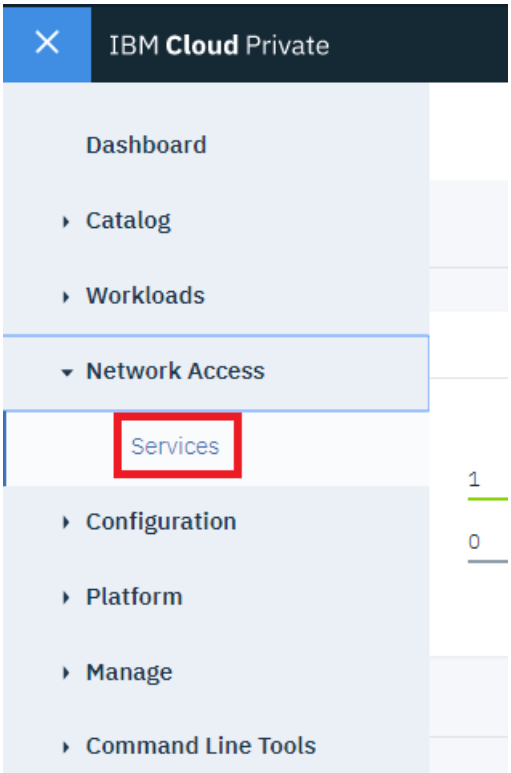
In the next section we will use the kubernetes object “Ingress” that enables a kubernetes internal nginx HTTP server to balance the traffic and provides additional features like SSL management

3.3 Ingress

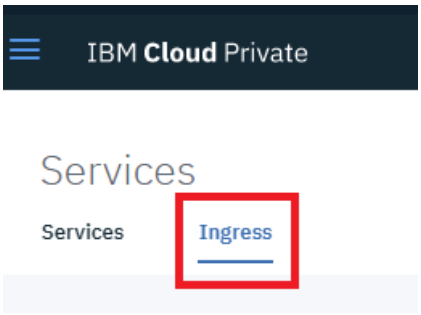
Once we have a service we can create an Ingress object. As explained an Ingress object is use to expose an application to consumers outside the kubernetes cluster. It uses a service to know the IPs of the application/deployment.

The Ingress Controller used by default in ICP is “NGINX Ingress Controller” (https://github.com/kubernetes/ingress-nginx) but there are more ingress controllers available, as for example F5 Ingress Controller

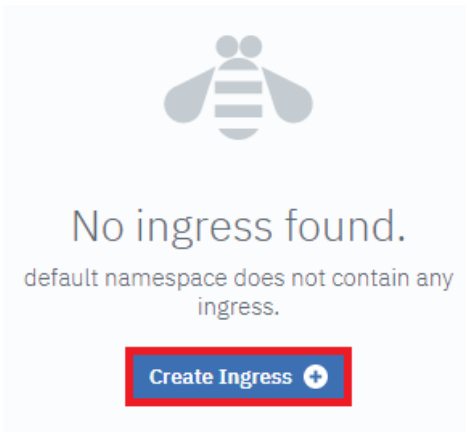
Go to “Services”.



And select Ingress Tab.



Click on “Create Ingress”



General:

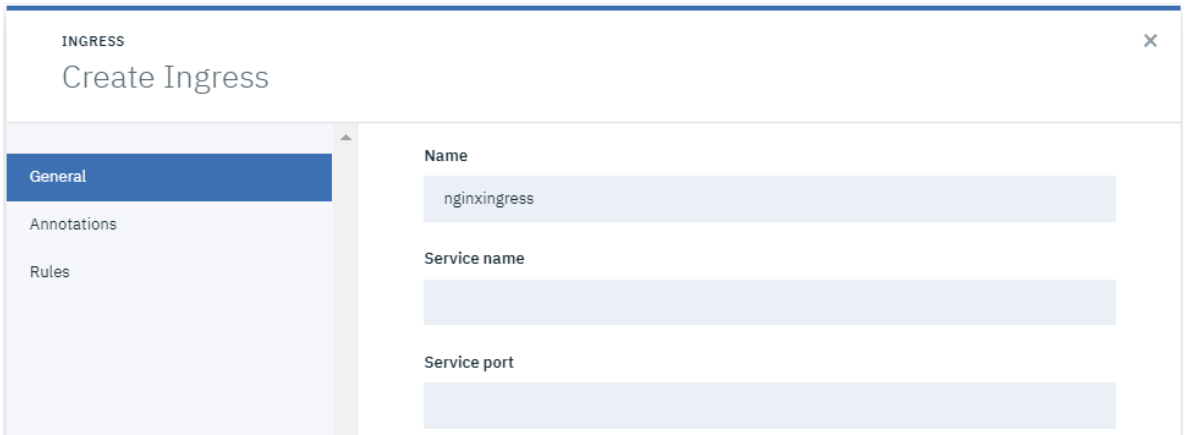
Name: nginxingress

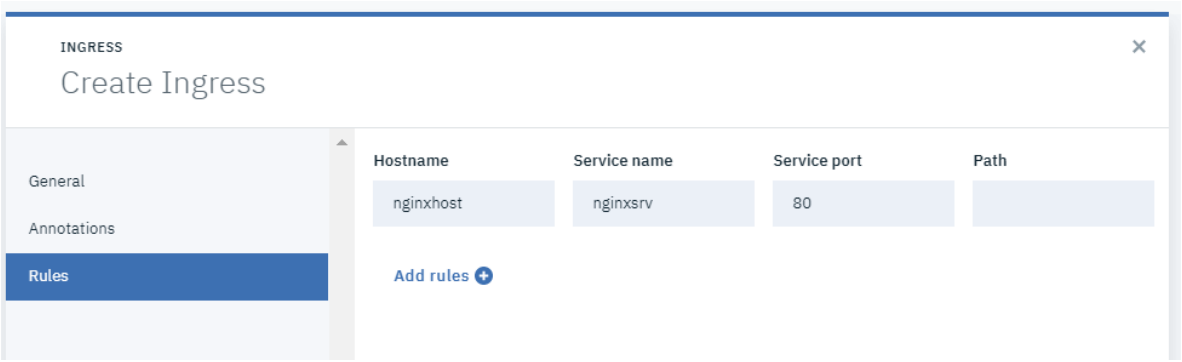
Rules:

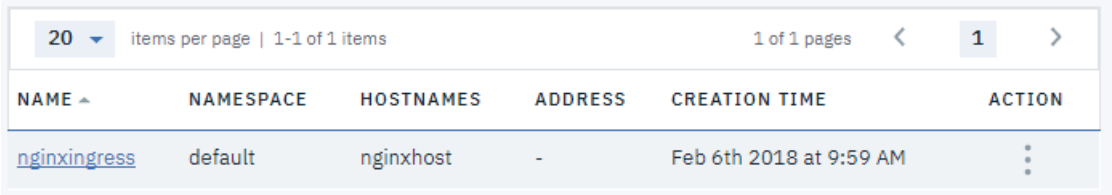
Hostname: nginxhost

Service name: nginxsrv

Service port: 80







kubectl describe ingress nginxingress
Name: nginxingress
Namespace: default
Address: 192.168.142.100
Rules:
Host Path Backends
---- ---- --------
nginxhost nginxsrv:80 (<none>)

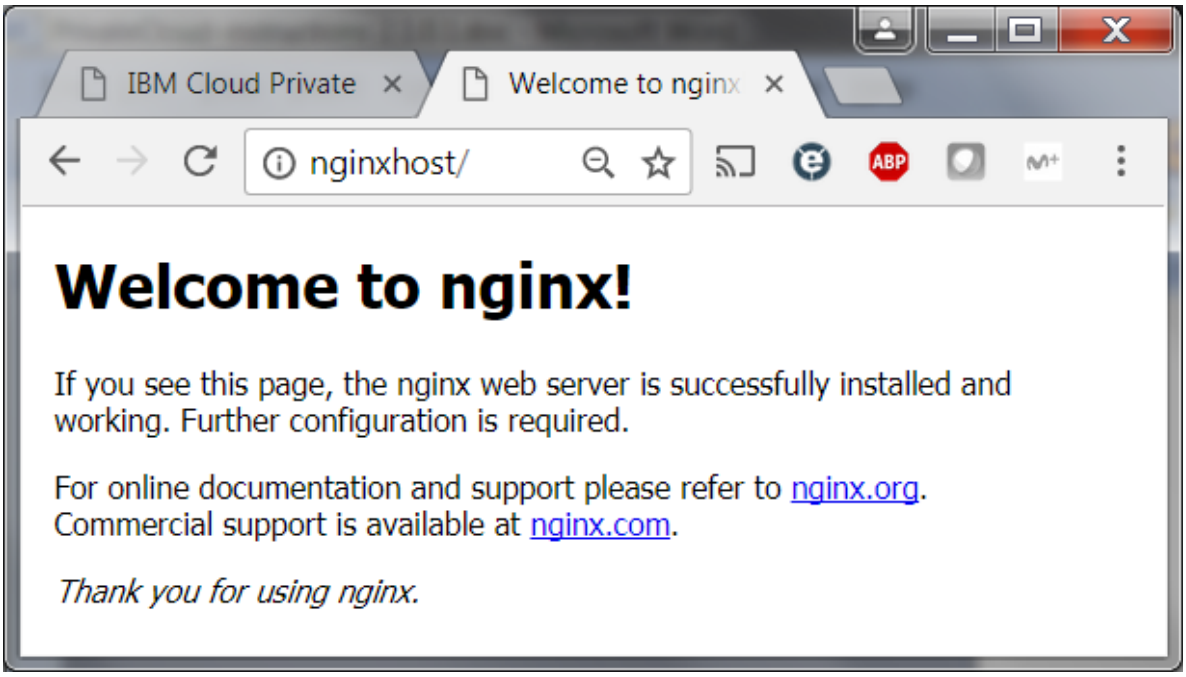
With this Ingress object kubernetes has configured the ingress controller (an internal nginx) to route traffic from proxy nodes to the application/deployment in the cluster.

The “rule” is that all the requests made for the hostname “nginxost” must be routed to the service “nginxsrv”. There are other options to configure the rules, not only by hostname, it is also possible to use the path.

It is also possible to use annotations to configure how the ingress controller manages the requests.

https://github.com/kubernetes/ingress-nginx/blob/master/docs/user-guide/annotations.md

Now if we add the hostname “nginxhost” to the hosts file of the operating system, pointing to the proxy node ip “192.168.142.100”, and access to the url http://nginxhost we will access to our application/deployment.



You can also invoke the application using curl:

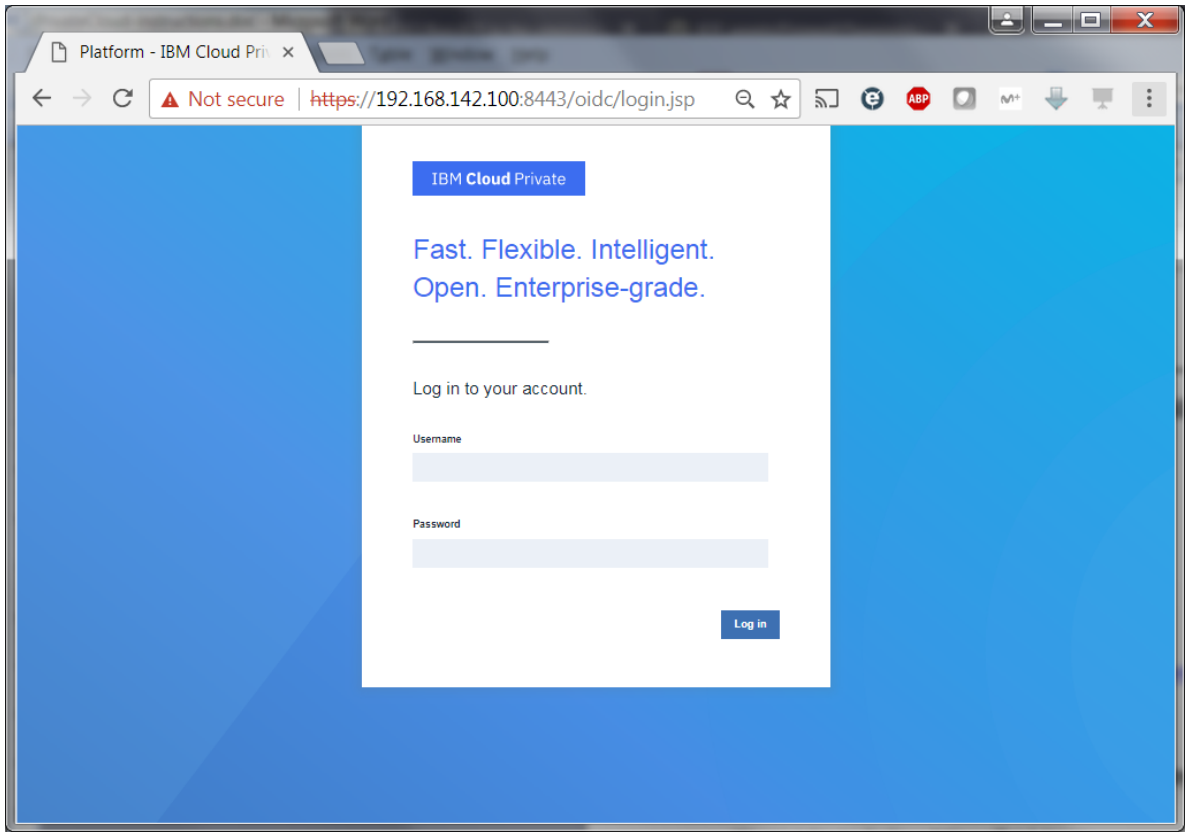
```
osboxes@osboxes:~$ curl -H "Host:nginxhost" 192.168.142.100
<!DOCTYPE html>
<html>
….
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Lab 2: Deploy and expose an Application using K8s CLI

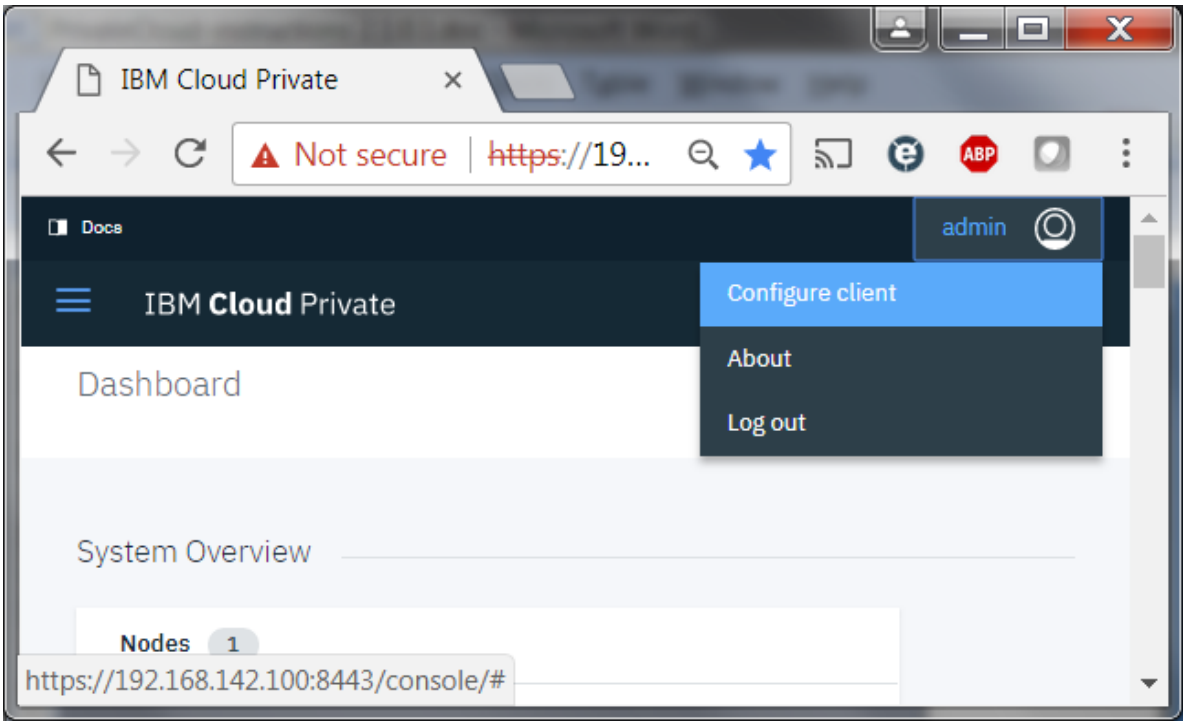
1. Configure Kubectl

Access to IBM Cloud Private console and login with admin / admin

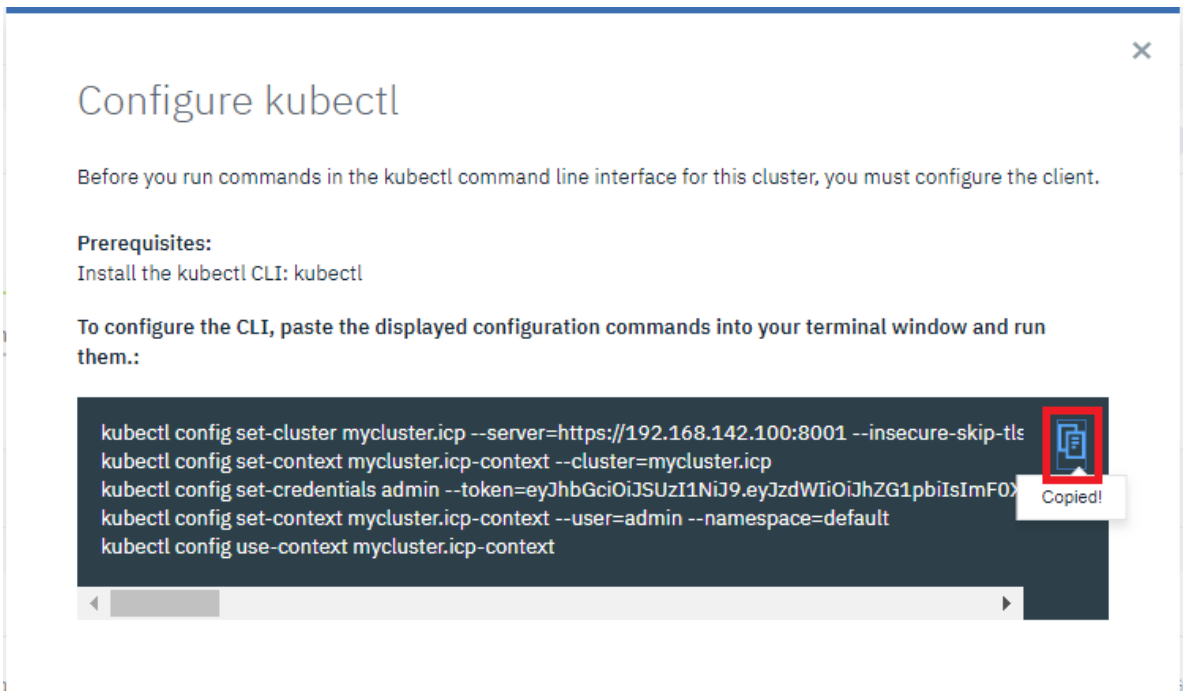
https://192.168.142.100:8443



Show the client configuration



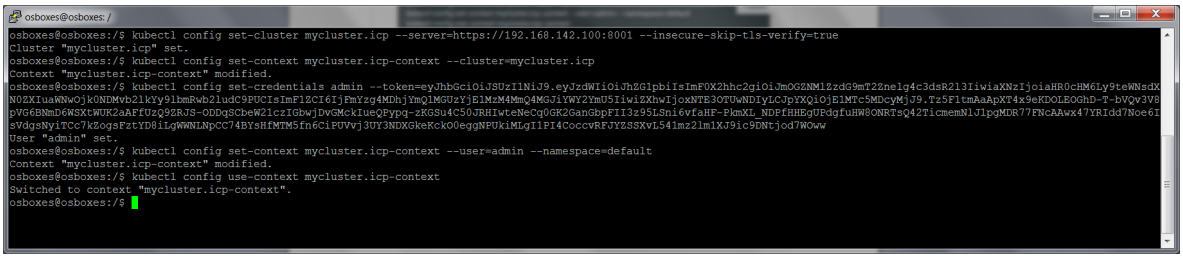
Copy the configuration:



Double check you are in namespace “default” (`kubectl config set-context mycluster.icp-context --user=admin --namespace=default`)

This configuration is used by the k8s CLI (kubectl) to interact with correct kubernetes provider. In this case the configuration points to the IBM Cloud Private installation.

Paste it in the command line terminal. (open a putty for 192.168.142.100 with user osboxes / osboxes.org



It is also possible to get this configuration through command line in case of system integrations.

2. Deploy an application

Now we can execute kubectl commands in name of user “admin”.

List the application deployed in Lab1:

```
osboxes@osboxes:~$ kubectl get deployments
NAME     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx    1         1         1            1           1h
```

List the service deployed in Lab1:

```
osboxes@osboxes:/$ kubectl get service
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)       AGE
kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP       1d
nginxsrv     ClusterIP   10.0.0.80    <none>        80/TCP        22h
nginxsrv2    NodePort    10.0.0.227   <none>        80:31687/TCP   20h

osboxes@osboxes:/$ kubectl get ingress
NAME           HOSTS       ADDRESS         PORTS     AGE
nginxingress   nginxhost   192.168.142.100   80        20m
```

Although you can create an application directly from command line we are going to create a descriptor file. Create a file “nginx-deployment.yml” with the content:

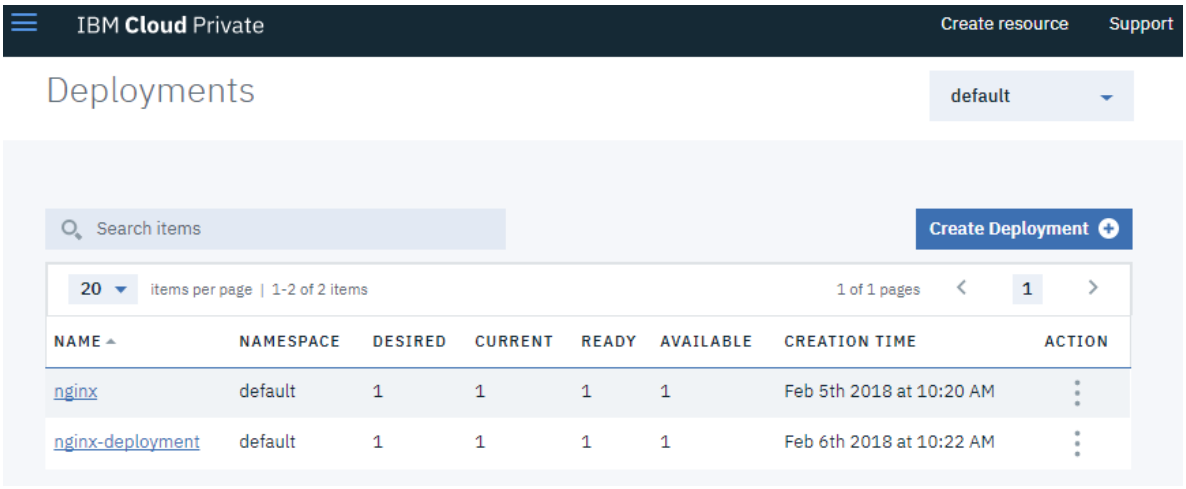
```
osboxes@osboxes:~$ vi nginx-deployment.yml

osboxes@osboxes:~$ cat nginx-deployment.yml
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx-deployment
    spec:
      containers:
      - name: nginx
        image: nginx:1.13.1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
```

Create the deployment using k8s CLI:

```
osboxes@osboxes:~$ kubectl create -f nginx-deployment.yml
deployment "nginx-deployment" created

osboxes@osboxes:~$ kubectl get deployments
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx             1         1         1            1           1h
nginx-deployment  1         1         1            0           9s

osboxes@osboxes:~$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
nginx-1769497579-80krq        1/1       Running   0          1h
nginx-deployment-1290044638-30937   1/1       Running   0          23m
```

If we go the IBM Cloud Private console we will see the new nginx-deployment.



3. Expose the application

Create a file “nginx-deployment.yml” with the content:

osboxes@osboxes:~$ vi nginx-service.yml

osboxes@osboxes:~$ cat nginx-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx-service
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
#    nodePort: 31320
  selector:
    app: nginx-deployment
```

Comments about this file:

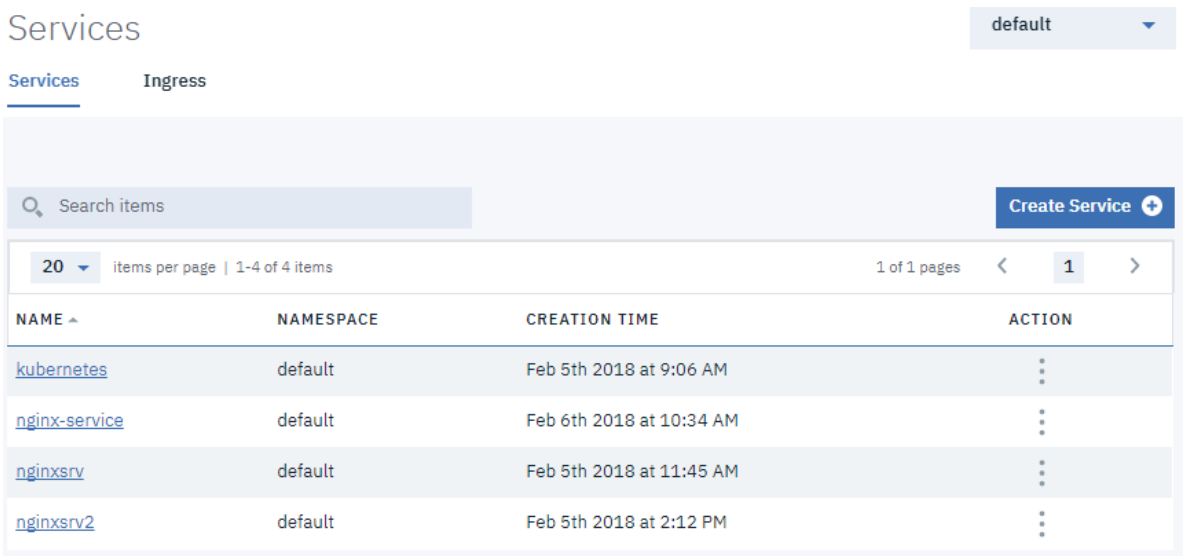
We are going to expose the service as NodePort with the property “type: NodePort”.
Although it is possible to specify a nodePort it is recommended to let K8s to assign it dynamically.
With the selector property we are specifying that the deployment that we are going to expose with this service is the one we created before, it has the property “selector: app: nginx-deployment”

Create the service using K8s CLI:

```
osboxes@osboxes:~$ kubectl create -f nginx-service.yml
service "nginx-service" created

osboxes@osboxes:~$ kubectl get services
NAME          TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)       AGE
kubernetes    ClusterIP  10.0.0.1     <none>        443/TCP       1d
nginx-service NodePort   10.0.0.68    <none>        80:30062/TCP  16s
nginxsrv      ClusterIP  10.0.0.80    <none>        80/TCP        22h
nginxsrv2     NodePort   10.0.0.227   <none>        80:31687/TCP  20h
```

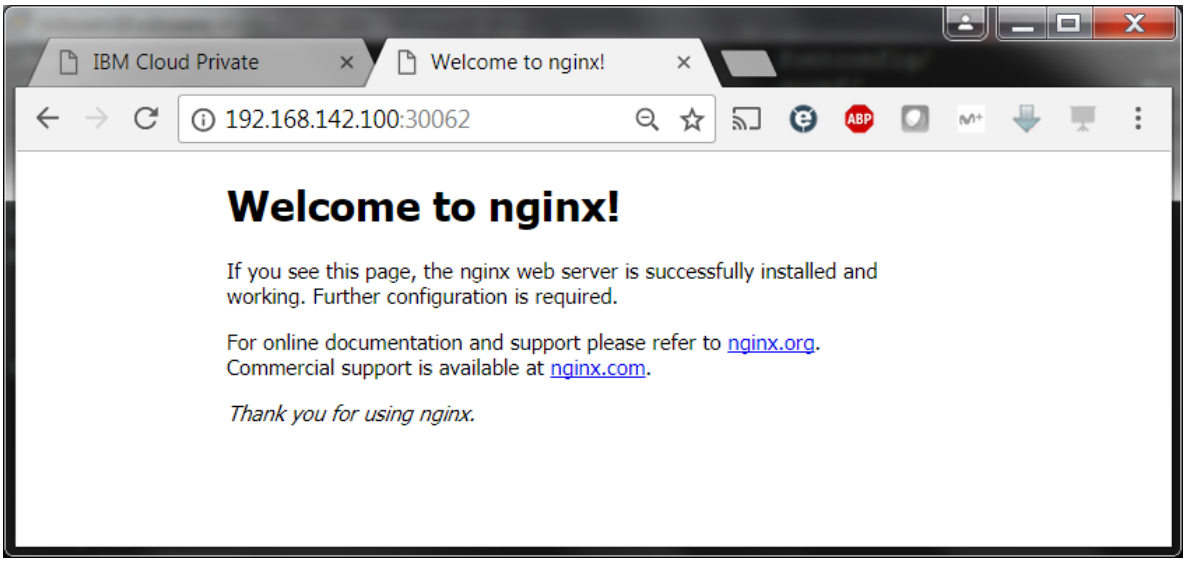
Take note of the port “30062”, it is the port assigned by k8s to our service in the worker nodes.

In the IBM Cloud private console:



Now we can invoke the application (Deployment), this time we have exposed it as NodePort so to invoke it we need to point to <K8s worker IP>:<NodePort>, in our case 192.168.142.100:30062.

IBM Cloud Private also expose that port through the proxy component balancing the load between the worker nodes, in our installation the worker node and the proxy node are in the same computer so the url will be the same 192.168.142.100:30062.



Lab 3. Storage

Create Persistent Volume (PV) and Persistent Volume Claim (PVC)

https://kubernetes.io/docs/concepts/storage/volumes/

During this lab we will configure Storage in IBM Cloud private, this storage will allow to our application to access to a persistent storage.

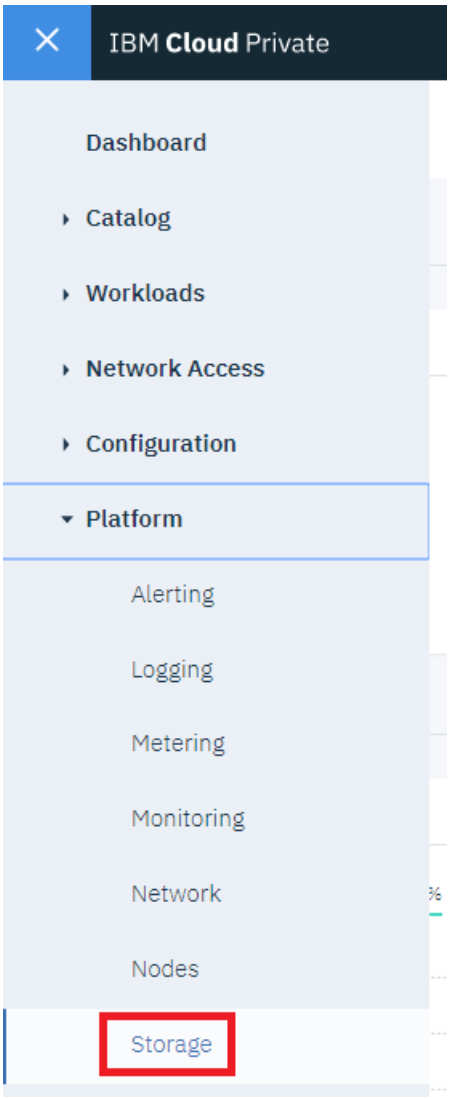
We will use nginx and will load a page we have created.

There are several options for the Storage, like for example NFS, but for this lab we will use the local file system.

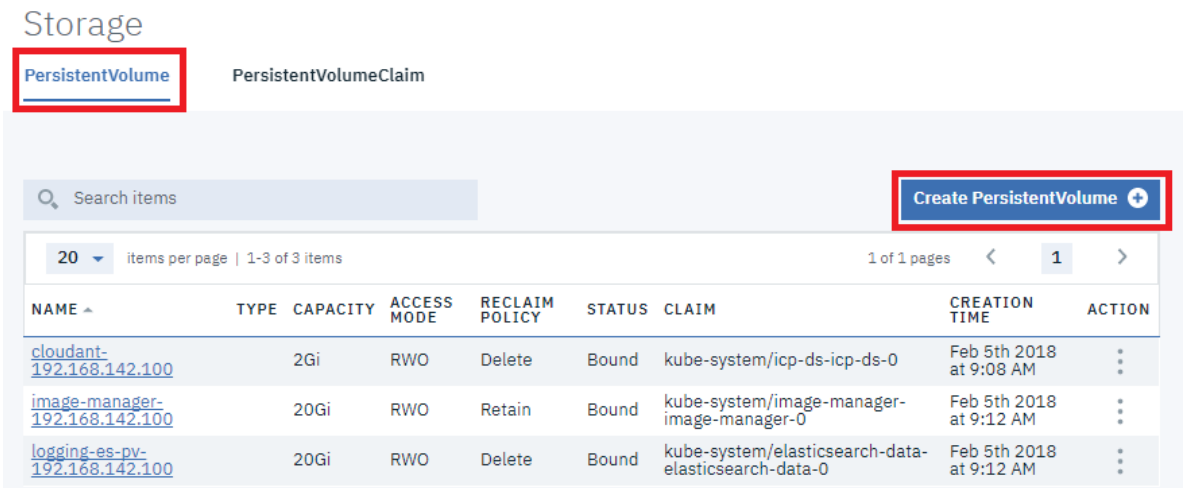
Connect to the IBM Cloud private host and in the command line (putty terminal):

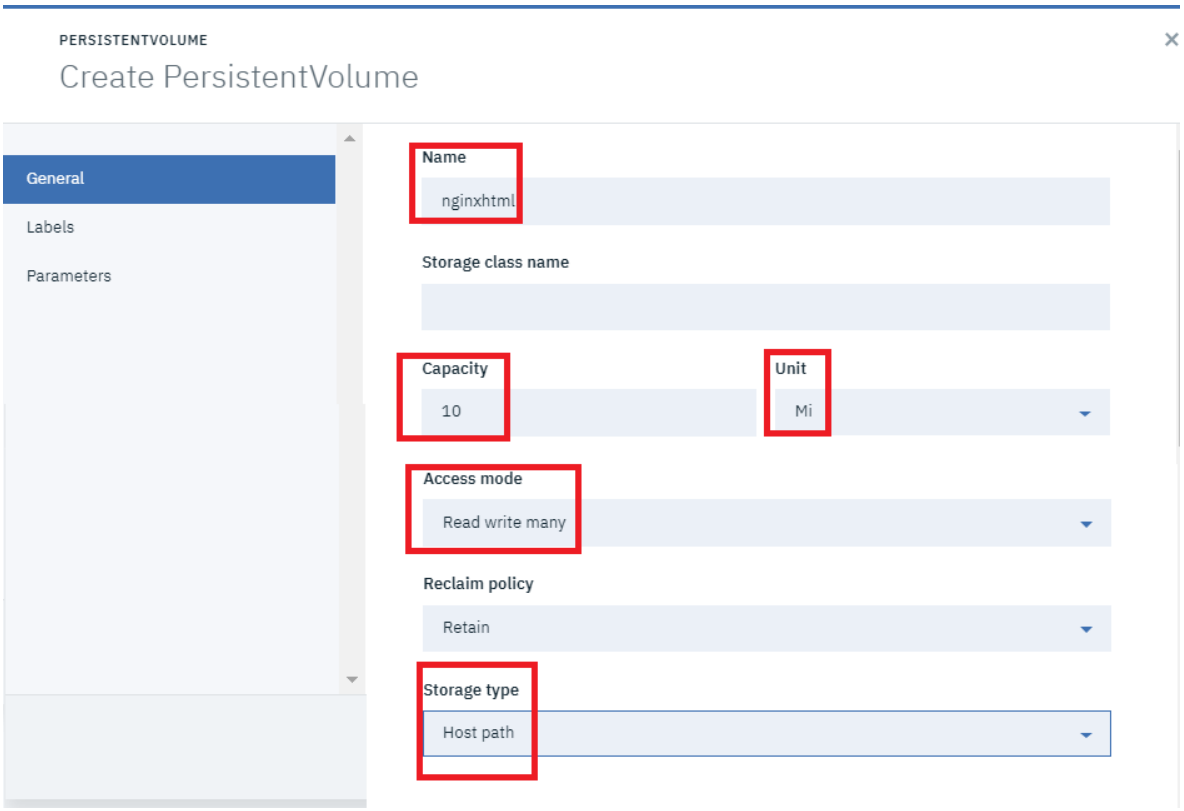
```
sudo mkdir -p /aStorage/nginx
sudo chmod 777 -R /aStorage/
vi /aStorage/nginx/index.html
cat /aStorage/nginx/index.html

<html>
<body>
Hello IBM Cloud private
</body>
</html>
```

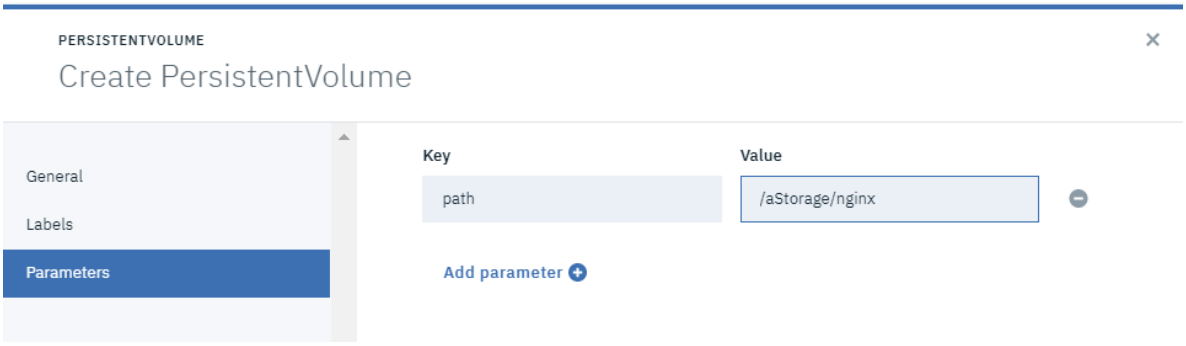
Now go to the IBM Cloud private console, login as admin / admin and go to storage.



Create a new Persistent Volume:



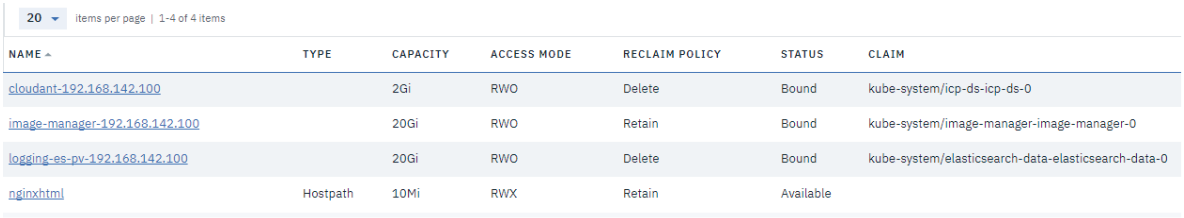




JSON file:


```
{
 "kind": "PersistentVolume",
 "apiVersion": "v1",
 "metadata": {
  "name": "nginxhtml",
  "labels": {}
 },
```

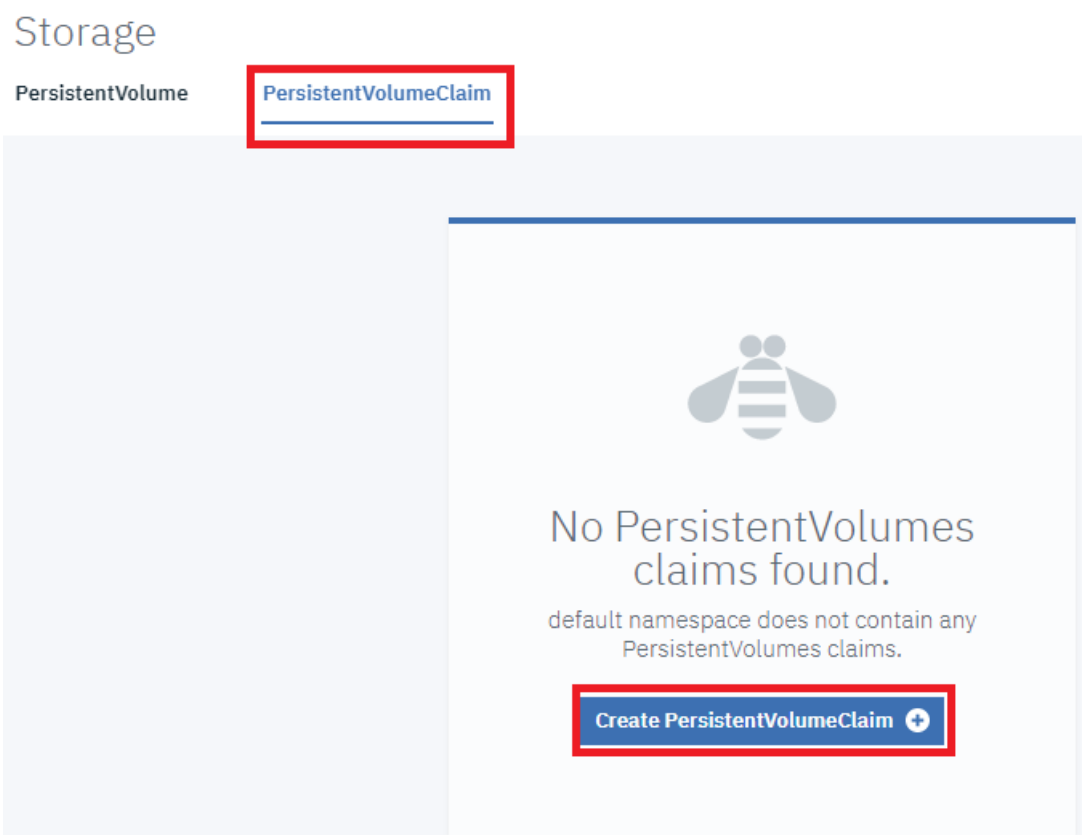
```
 "spec": {
  "capacity": {
   "storage": "10Mi"
  },
  "accessModes": [
   "ReadOnlyMany"
  ],
  "persistentVolumeReclaimPolicy": "Retain",
  "hostPath": {
   "path": "/aStorage/nginx"
  }
 }
}
```

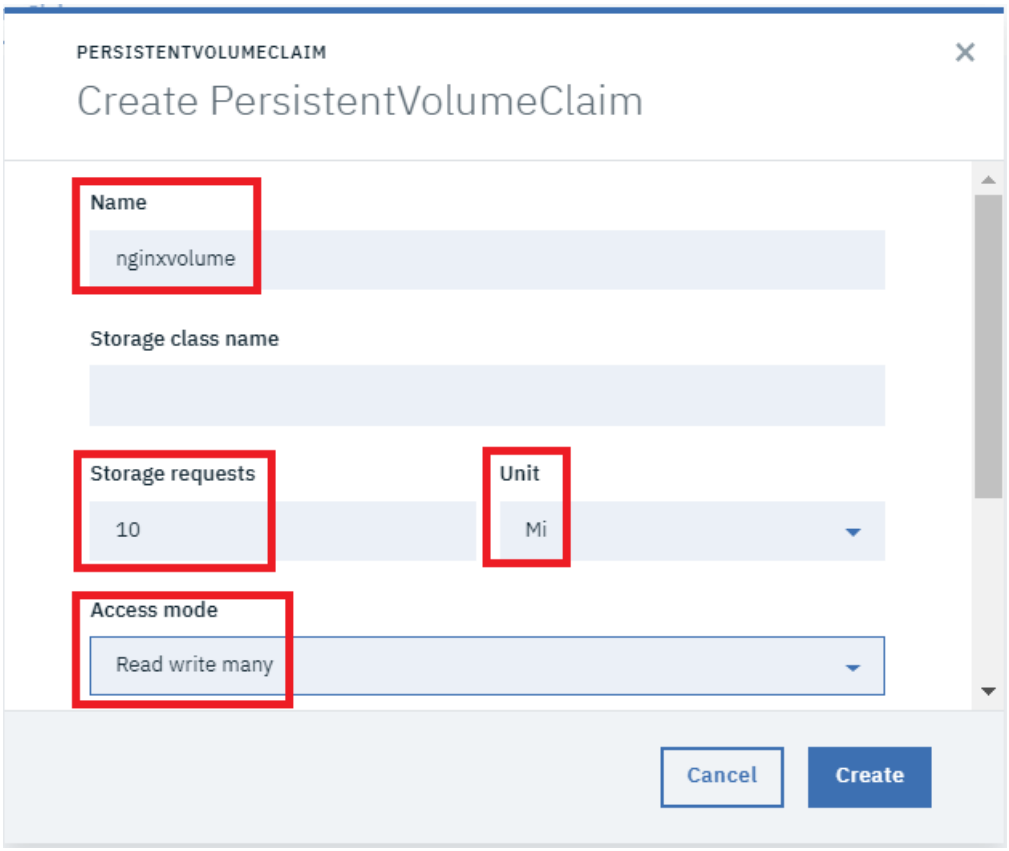


20 items per page | 1-4 of 4 items

NAME	TYPE	CAPACITY	ACCESS MODE	RECLAIM POLICY	STATUS	CLAIM
cloudant-192.168.142.100		2Gi	RWO	Delete	Bound	kube-system/icp-ds-icp-ds-0
image-manager-192.168.142.100		20Gi	RWO	Retain	Bound	kube-system/image-manager-image-manager-0
logging-es-pv-192.168.142.100		20Gi	RWO	Delete	Bound	kube-system/elasticsearch-data-elasticsearch-data-0
nginxhtml	Hostpath	10Mi	RWX	Retain	Available	

Now select the tab PersistentVolumeClaim and create a new Persistent Volume Claim

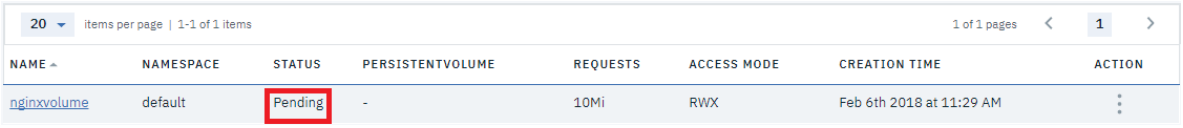

Storage
PersistentVolume
PersistentVolumeClaim
No PersistentVolumes claims found.
default namespace does not contain any PersistentVolumes claims.
Create PersistentVolumeClaim

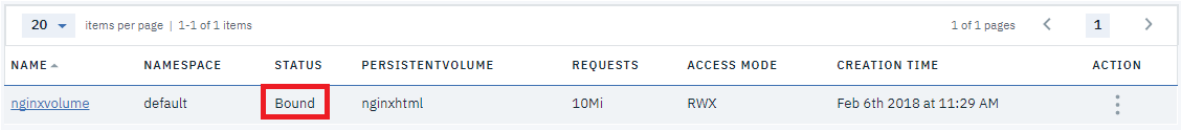


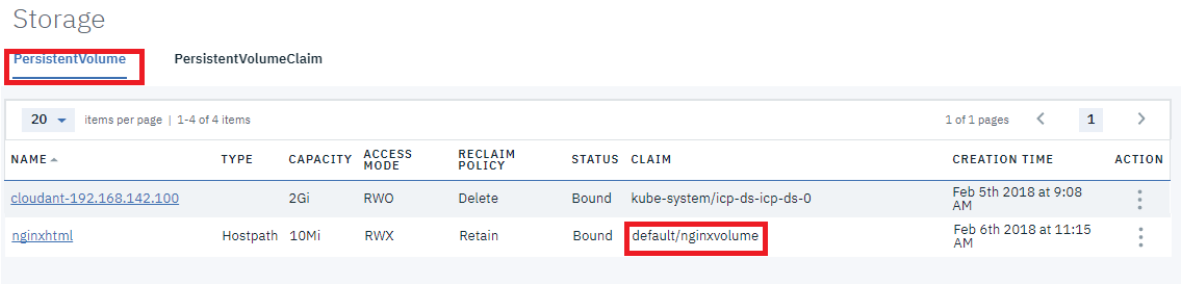
JSON File:

```
{
 "kind": "PersistentVolumeClaim",
 "apiVersion": "v1",
 "metadata": {
  "name": "nginxvolume"
 },
 "spec": {
  "resources": {
   "requests": {
    "storage": "10Mi"
   }
  },
  "accessModes": [
   "ReadOnlyMany"
  ]
 }
}
```

It is possible to use Labels and Storage Classes to specify which Persistent Volume will be bound by the Persistent Volume Claim.

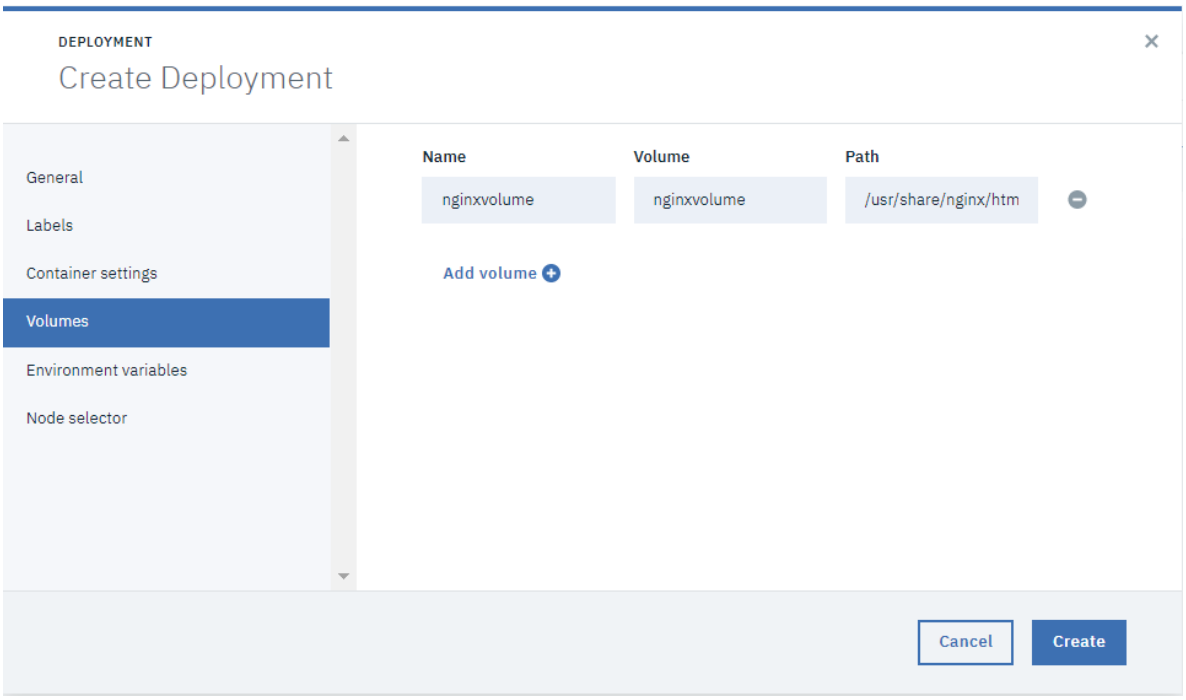






Use the volume in an application.

Creating a new deployment in the UI we can specify the volume claim in the “Volumes” section of a new deployment window:



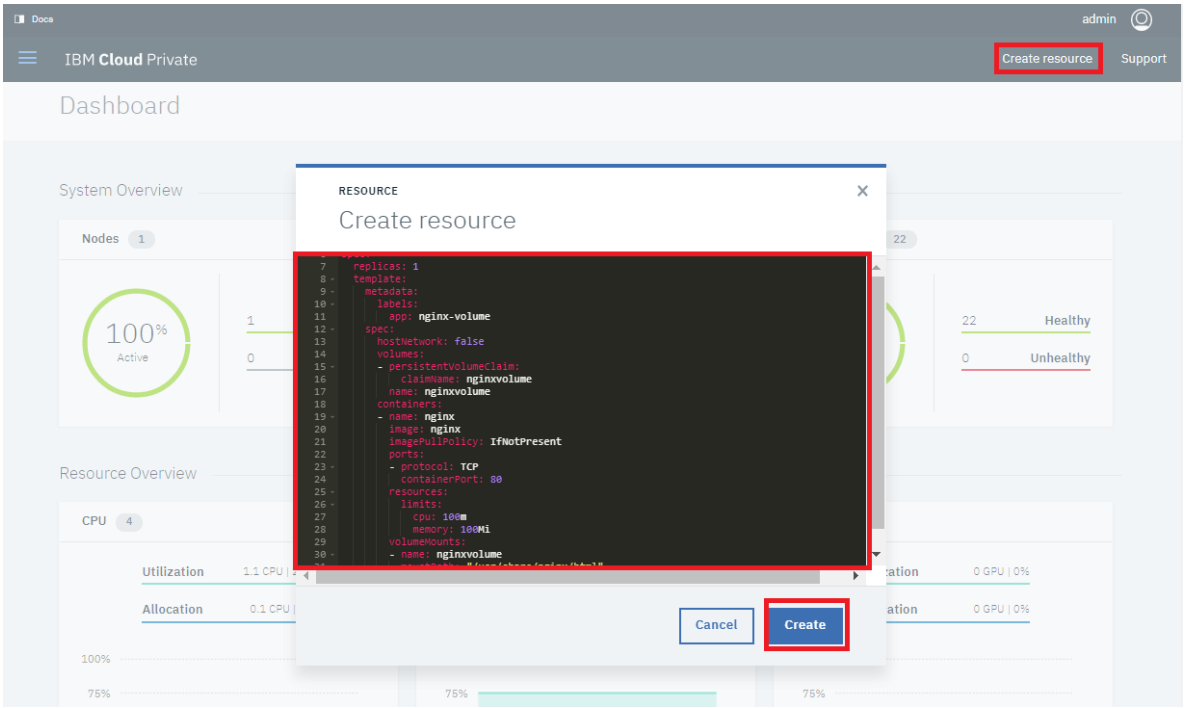
YAML file:

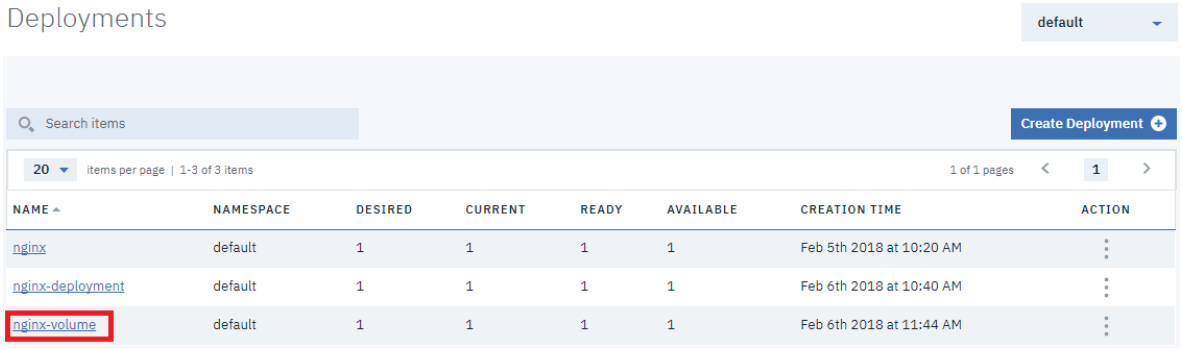
```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
 name: nginx-volume
spec:
 replicas: 1
 template:
  metadata:
   labels:
    app: nginx-volume
  spec:
   hostNetwork: false
   volumes:
   - persistentVolumeClaim:
      claimName: nginxvolume
    name: nginxvolume
   containers:
   - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
```

```
ports:
- protocol: TCP
  containerPort: 80
resources:
  limits:
    cpu: 100m
    memory: 100Mi
volumeMounts:
- name: nginxvolume
  mountPath: "/usr/share/nginx/html"
```

Using the console with the property “Path” or in the yaml file with the property “mountPath” we specify in which path inside the container we are going to mount our storage. In this case is "/usr/share/nginx/html" that is where nginx loads its default web page.

In the console, in the top menu, select create resource, copy the yaml and click “Create”

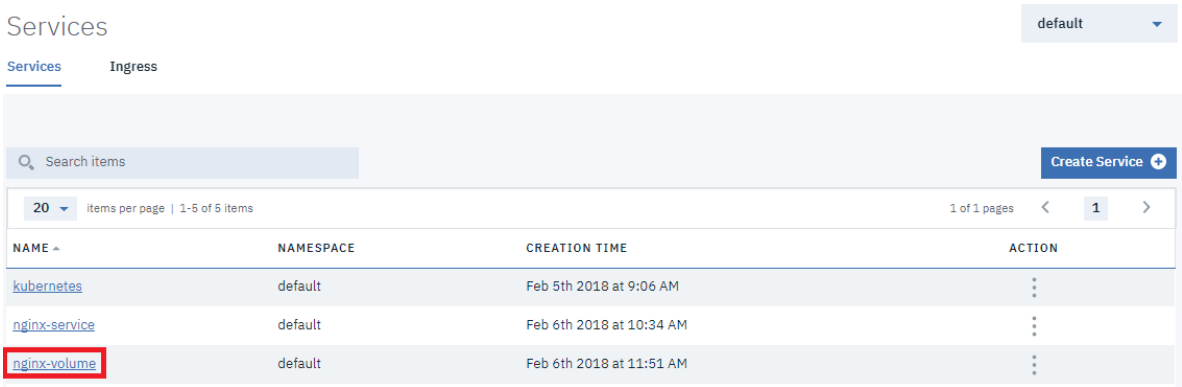




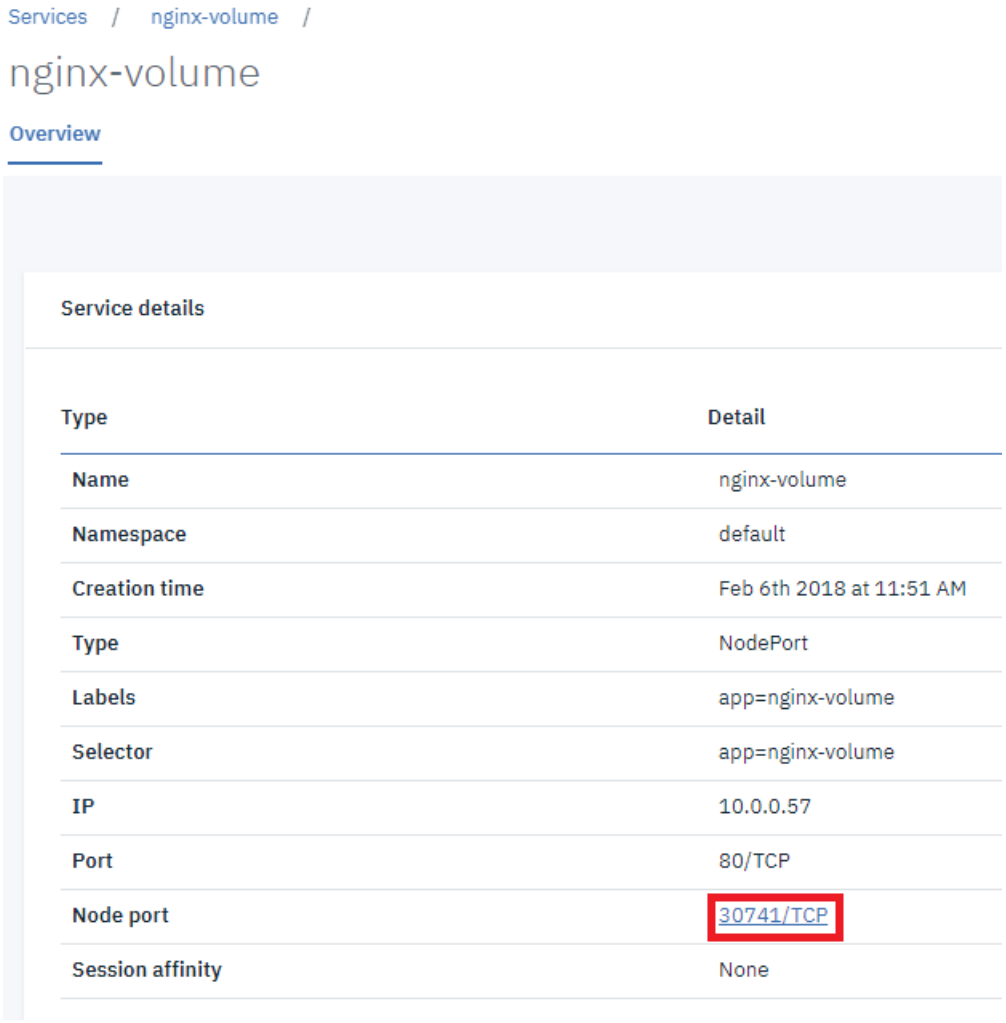
Expose the application:

As before click on “Create resource” and create a servive:

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



Check the node port exposed.



Invoke the application from the command line:

osboxes@bootnode:~$ curl 192.168.142.100:30741
<html>
<body>
Hello IBM Cloud private
</body>
</html>

Make a change in the file and execute it again, you will see your updated file.

Lab 4. Image repository

During these labs we have been using docker hub as image repository, in this lab we are going to create our own docker image and we are going to push it to IBM Cloud Private image repository, then we will use this image to create an application.

This image is a node.js application that we will use in the next lab to demonstrate the out of the box features of Kubernetes as a “docker container manager”.

1. Create a docker image

Prepare the environment

For this lab we provide two files, app.js with the node.js code and package.json to automate the build and execution of the node.js with npm.

In case you don’t have app.js and pacakage.json files in your environment they are in the Appendix section of this document.

Copy / create them in a folder with name “node”.

```
osboxes@osboxes:~$ pwd
/home/osboxes

osboxes@osboxes:~$ mkdir node

osboxes@osboxes:~$ vi node/app.js
osboxes@osboxes:~$ vi node/package.json

osboxes@osboxes:~$ ls node
app.js  package.json
```

Create the dockerfile to create our image.

```
osboxes@osboxes:~$ vi hellonode.dockerfile

osboxes@osboxes:~$ cat hellonode.dockerfile
```

```
FROM debian:8.7

RUN apt-get -y update && apt-get install -y curl

RUN curl -sL https://deb.nodesource.com/setup_6.x | bash -
RUN apt-get install -y nodejs

RUN mkdir /app
ADD ./node/app.js /app
ADD ./node/package.json /app
RUN cd /app && npm install

ENV PORT 8080
EXPOSE 8080
WORKDIR "/app"
CMD ["npm", "start"]
```

With this docker file what we do is to create a new docker image from a from a debian docker image in dockerhub. This is the line “FROM debian:8:1”.

Then we install node and copy our files inside the image.

Finally we say to the image that it has be started with the command /app/npm start

Create the image

At this moment we have these images in our local repository:

```
osboxes@osboxes:~$ sudo docker images | grep -v ibmcom
REPOSITORY                    TAG             IMAGE ID         CREATED          SIZE
nginx                         latest          3f8a4339aadd     5 weeks ago      108MB
hello-world                   latest          f2a91732366c     2 months ago     1.85kB
nginx                         1.13.1          c246cd3dd41d     7 months ago     107MB
```

Create our hellonode docker image.

```
osboxes@osboxes:~$ pwd
/home/osboxes
```

osboxes@osboxes:~$ sudo docker build -t hellonode:1.0 -f hellonode.dockerfile .
...
...
Successfully built 84ed3945077c

If we execute again docker images we will see debian:8.7 and hellonode:1.0

osboxes@osboxes:~$ sudo docker images | grep -v ibmcom

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellonode	1.0	595463fa00bb	22 seconds ago	227MB
nginx	latest	3f8a4339aadd	5 weeks ago	108MB
hello-world	latest	f2a91732366c	2 months ago	1.85kB
nginx	1.13.1	c246cd3dd41d	7 months ago	107MB
debian	8.7	054abe38b1e6	9 months ago	123MB

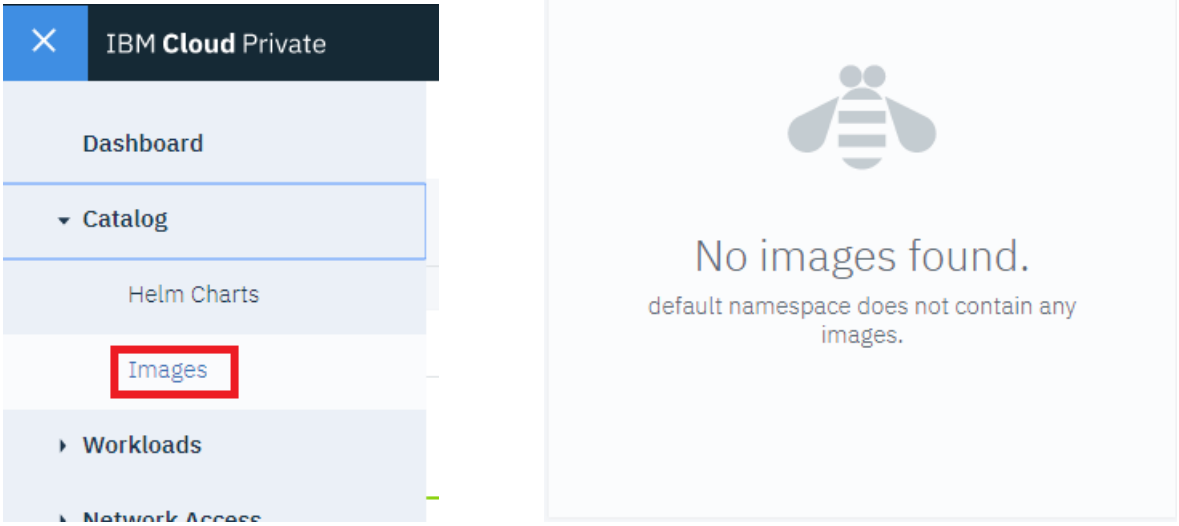
2. Publish the image to IBM Cloud Private

In the IBM Cloud private list the images.

Login in IBM Cloud private.

https://192.168.142.100:8443

List the images:



As this is a fresh installation the user does not have any image yet, all the images used in previous labs where in dockerhub repository.

Using docker push command we will publish hellonode image to the IBM Cloud Private repository.

Tag the image

First we need to tag the docker image, with the tag name docker will know where to push the image.

```
osboxes@osboxes:~$ sudo docker tag hellonode:1.0 mycluster.icp:8500/default/hellonode:1.0

osboxes@osboxes:~$ sudo docker images | grep -v ibmcom
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
hellonode                                 1.0                 595463fa00bb        4 minutes ago       227MB
mycluster.icp:8500/default/hellonode      1.0                 595463fa00bb        4 minutes ago
227MB
nginx                                     latest              3f8a4339aadd        5 weeks ago         108MB
hello-world                               latest              f2a91732366c        2 months ago        1.85kB
nginx                                     1.13.1              c246cd3dd41d        7 months ago        107MB
debian                                    8.7                 054abe38b1e6        9 months ago        123MB
```

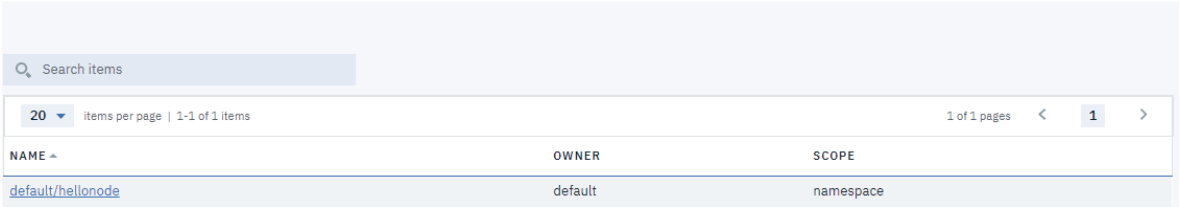
mycluster.icp:8500/ is the hostname pointing to IBM Cloud Private image repository.
default is the namespace
hellonode is the image name
1:0 is the version tag

Login to the repository and push the image

```
osboxes@osboxes:~$ sudo docker login mycluster.icp:8500
Username (devuser): admin
Password:
Login Succeeded

osboxes@osboxes:~$ sudo docker push mycluster.icp:8500/default/hellonode:1.0
```

Now in the IBM Cloud Private console you can see the image.

Images



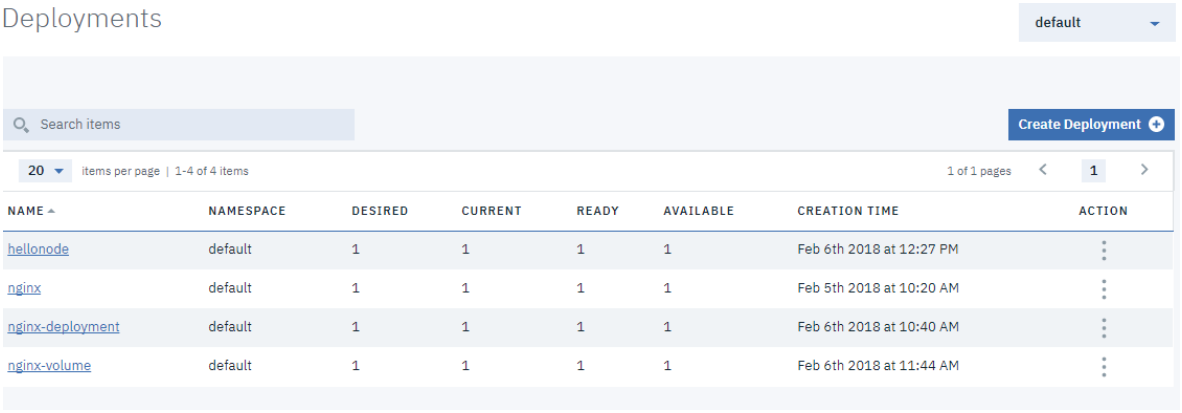
The Scope determines if this image can be seen in any other namespace or just in the current one. By default it is configured with scope “namespace” what means that it can only be used from the current namespace.

3. Create an application

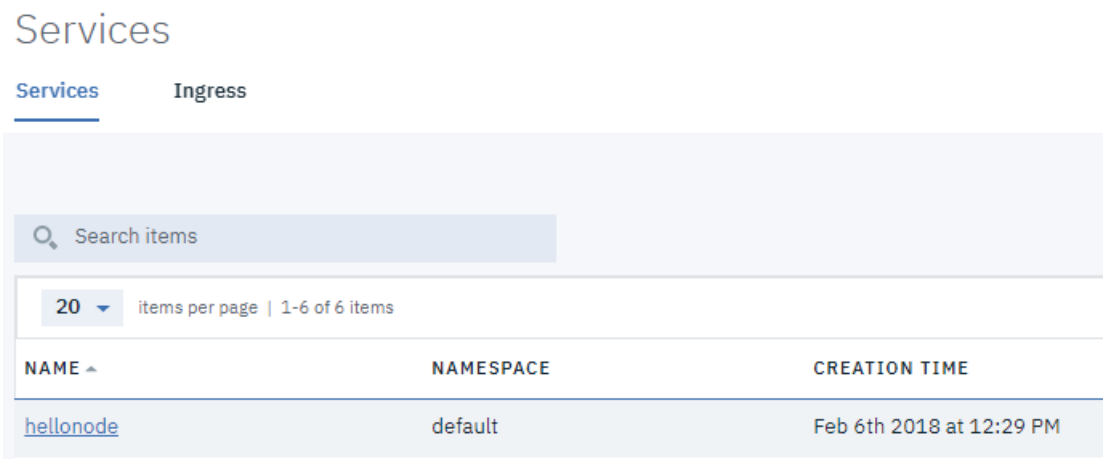
As we did in the previous create a new deployment and service using the “Create Resource” option.

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
 name: hellonode
spec:
 replicas: 1
 template:
  metadata:
   labels:
    app: hellonode
  spec:
   containers:
   - name: hellonode
    image: mycluster.icp:8500/default/hellonode:1.0
    imagePullPolicy: IfNotPresent
```

Pay attention to the image property “image: mycluster.icp:8500/default/hellonode:1.0”, now we are using the internal docker image repository.
Also the property “replicas: 1”, we will use it latter.




```
kind: Service
apiVersion: v1
metadata:
  name: hellonode
  labels:
    app: hellonode
spec:
  type: NodePort
  ports:
  - protocol: TCP
    name: node
    port: 8080
    targetPort: 8080
  selector:
    app: hellonode
  clusterIP: ''
  sessionAffinity: None
```



Services / hellonode /

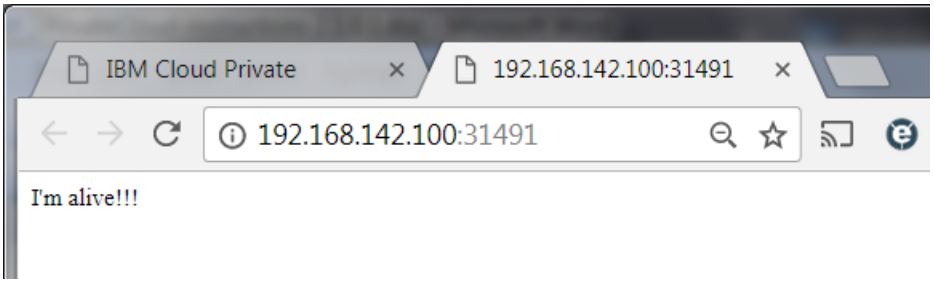
hellonode

Overview

Service details

Type	Detail
Name	hellonode
Namespace	default
Creation time	Feb 6th 2018 at 12:38 PM
Type	NodePort
Labels	app=hellonode
Selector	app=hellonode
IP	10.0.0.128
Port	node 8080/TCP
Node port	node 31491/TCP
Session affinity	None

In this script the port is 31491 but it will be different in each case. It is recommended you take note of your port to use it in the rest of the labs.



Lab 5. Helm Catalog

Helm is a tool that streamlines installing and managing Kubernetes applications. Think of it like apt/yum/homebrew for Kubernetes.

Use Helm to:

Find and use popular software packaged as Kubernetes charts
Share your own applications as Kubernetes charts
Create reproducible builds of your Kubernetes applications
Intelligently manage your Kubernetes manifest files
Manage releases of Helm packages

Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on.

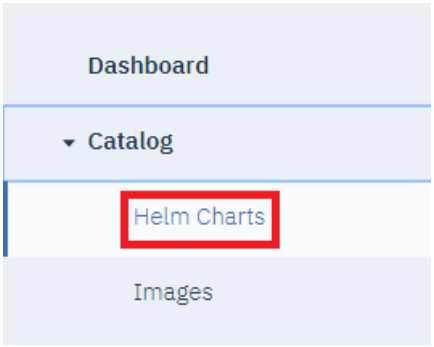
Charts are created as files laid out in a particular directory tree, then they can be packaged into versioned archives to be deployed.

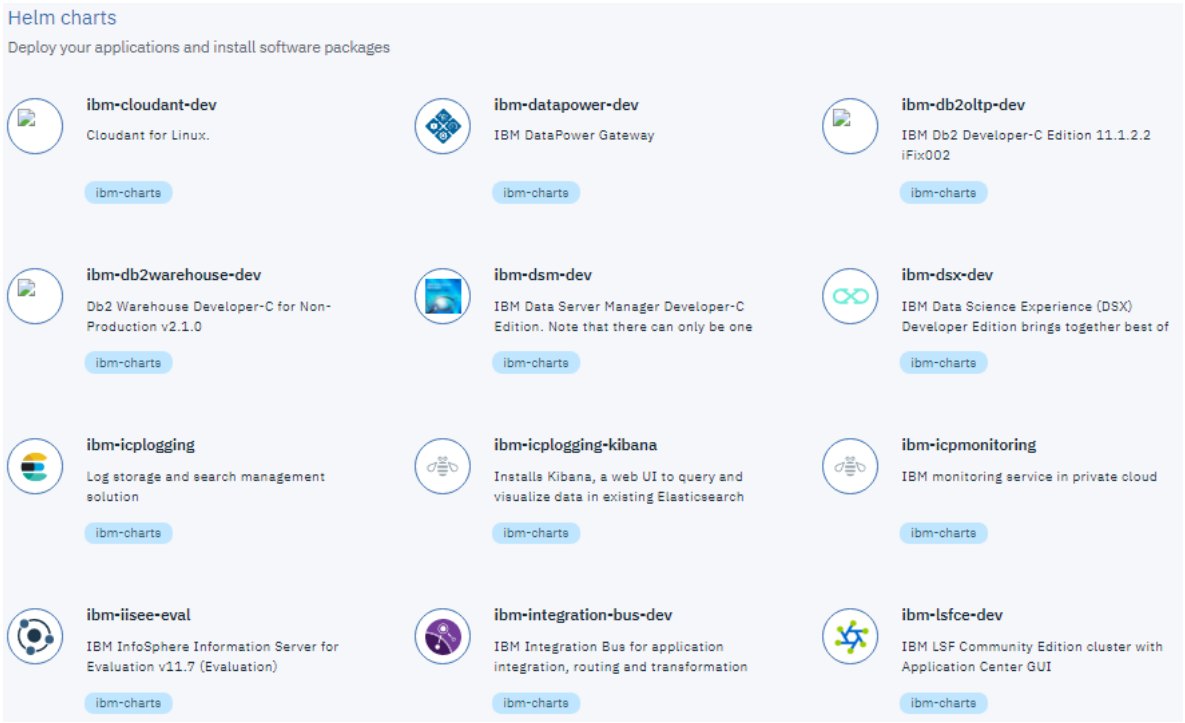
https://github.com/kubernetes/helm/blob/master/README.md
https://github.com/kubernetes/helm/blob/master/docs/charts.md

By default ICP comes with a HELM repository to access and Install IBM software, but it is possible to add external repositories with helm charts for third party software.

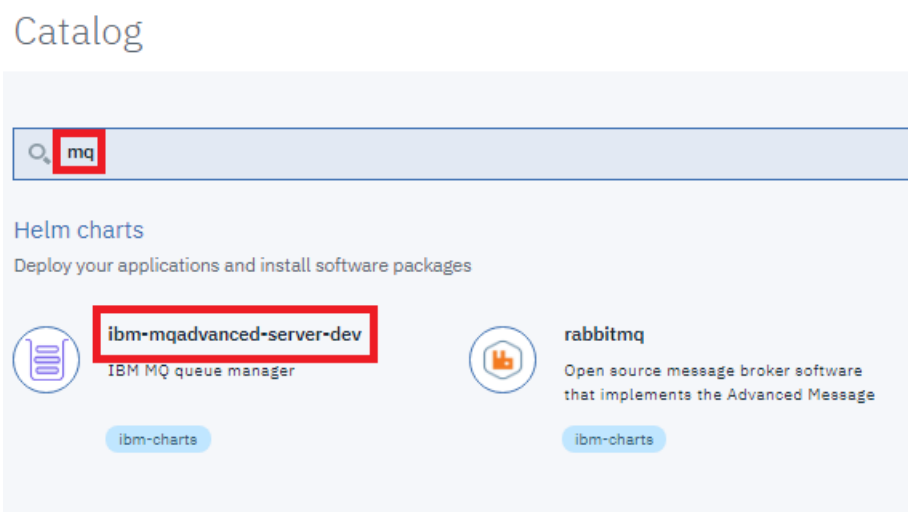
Deploy MQ in IBM Cloud Private

Access to ICP HELM Catalog



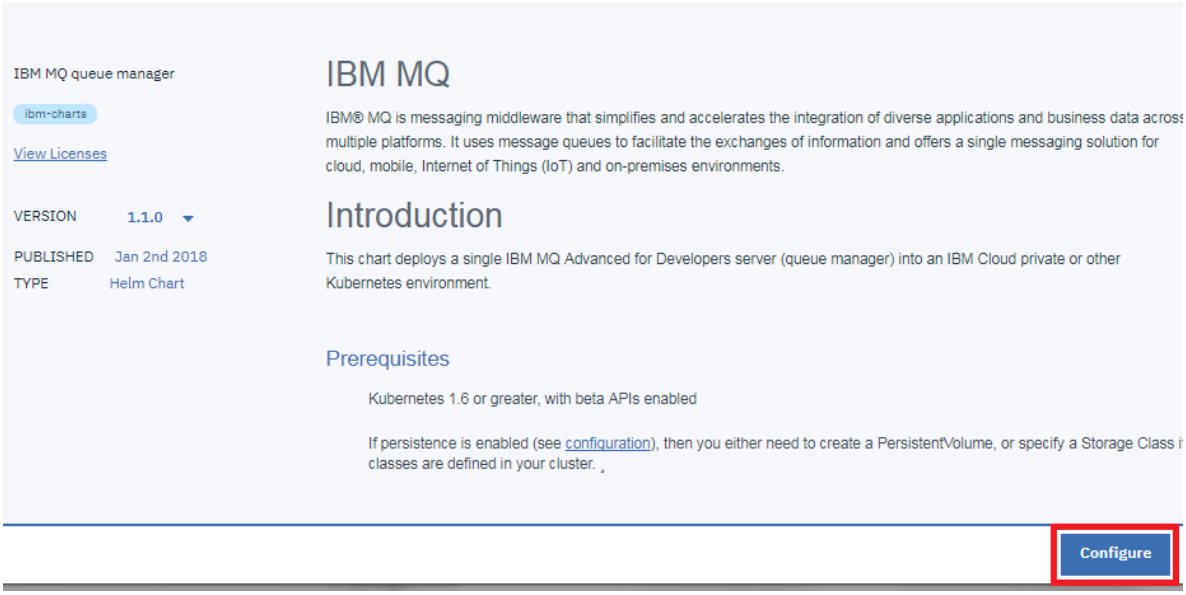


Look for mq and click on the chart



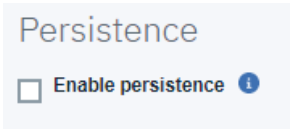
Take a look at the documentation and click on configure

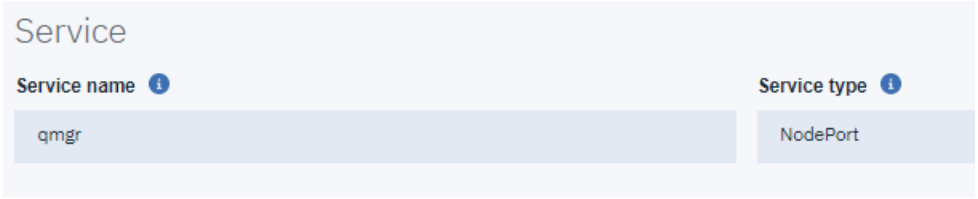
ibm-mqadvanced-server-dev V 1.1.0

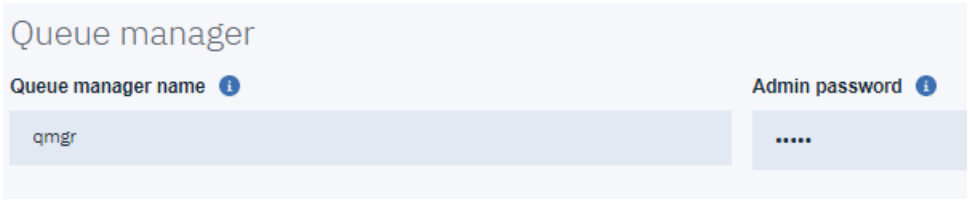


Set this configuration:

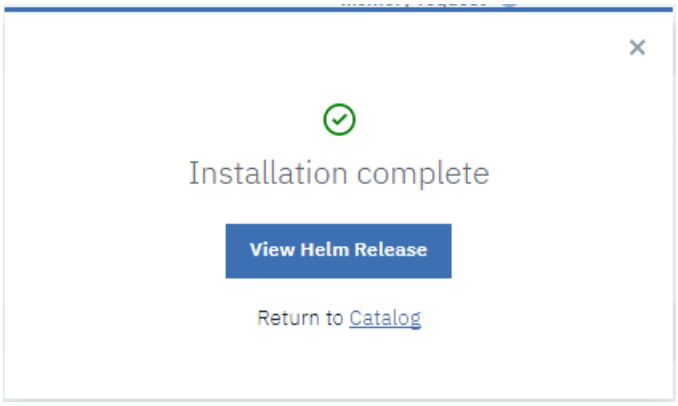
Release name: mymq
Target namespace: default
Check the box "I have read and agreed to the license agreements"
Uncheck the box "Enable persistence"
Service type: NodePort
Queue manager name: qmgr
Admin password: admin



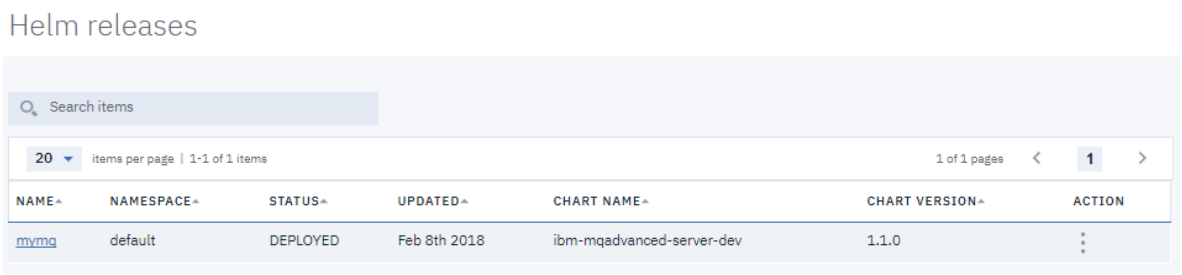




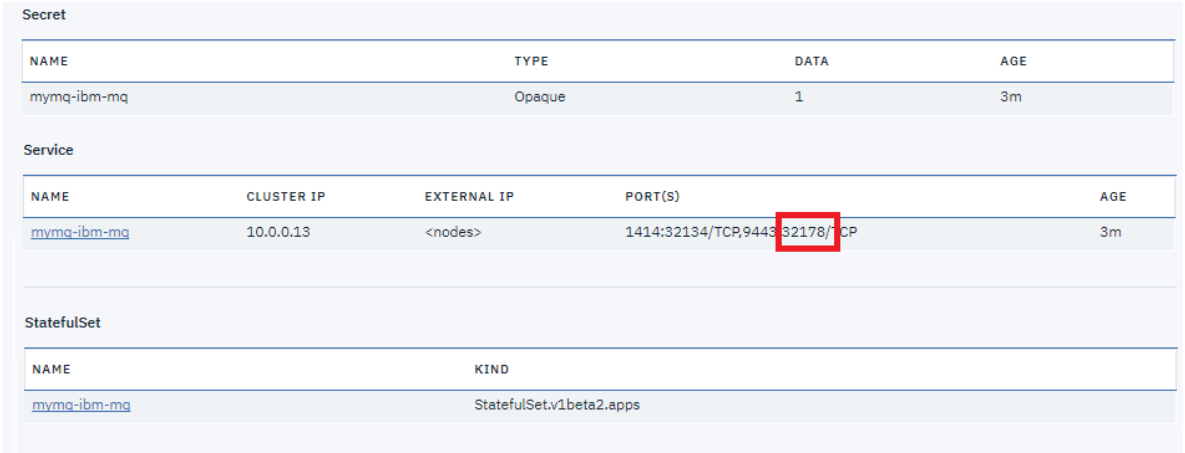
Click Install and when finished click on “View Hel Release”



Click on “mymq”

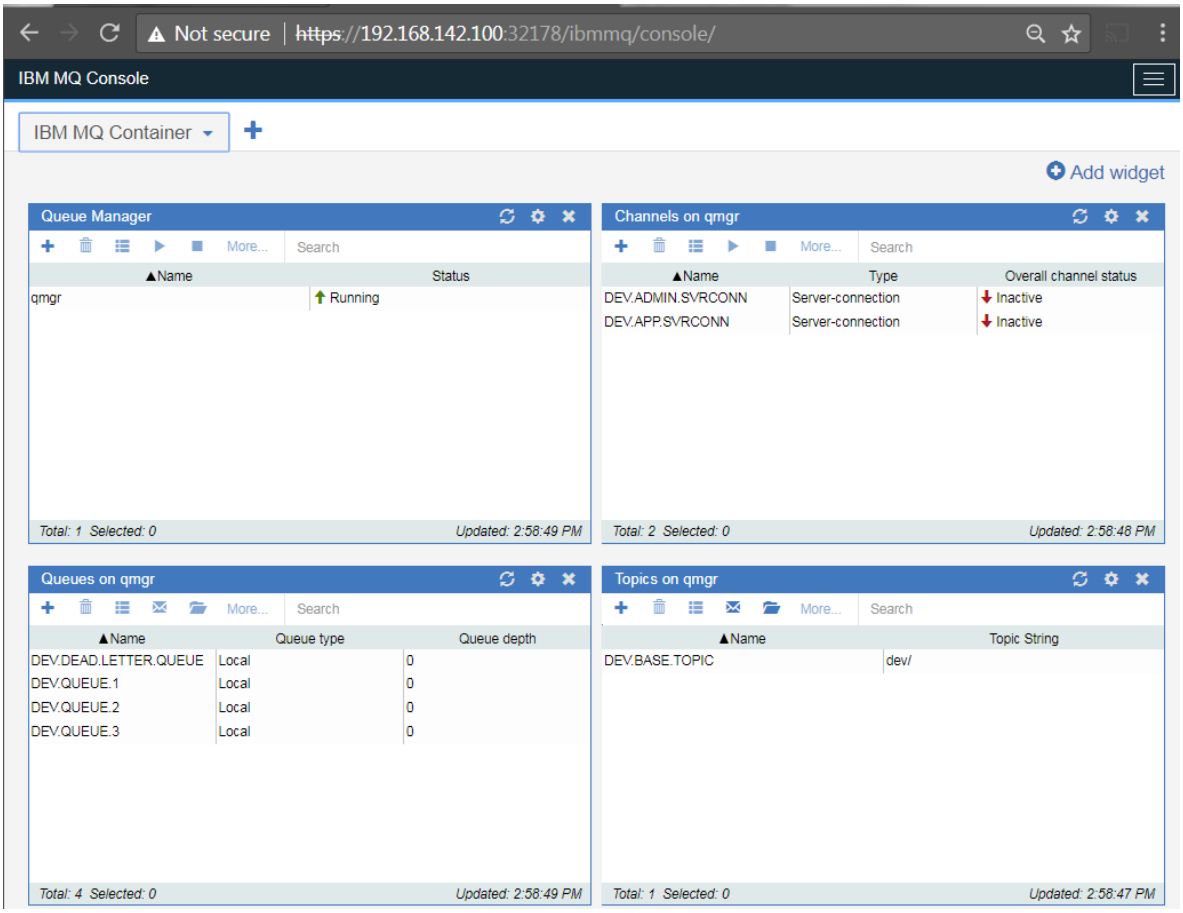


In the installation sever kubernetes objects where created. Take note of the port assigned to 9443, in this case 32178



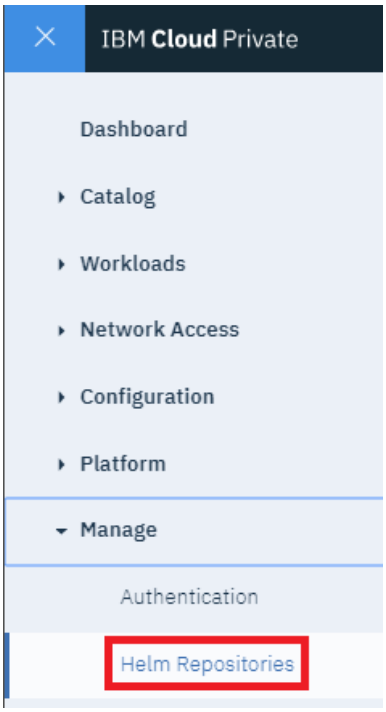
Access to the admin console and login with admin / admin

https://192.168.142.100:32178/ibmmq/console



Adding HELM Repositories

Manage HELM Repositories:

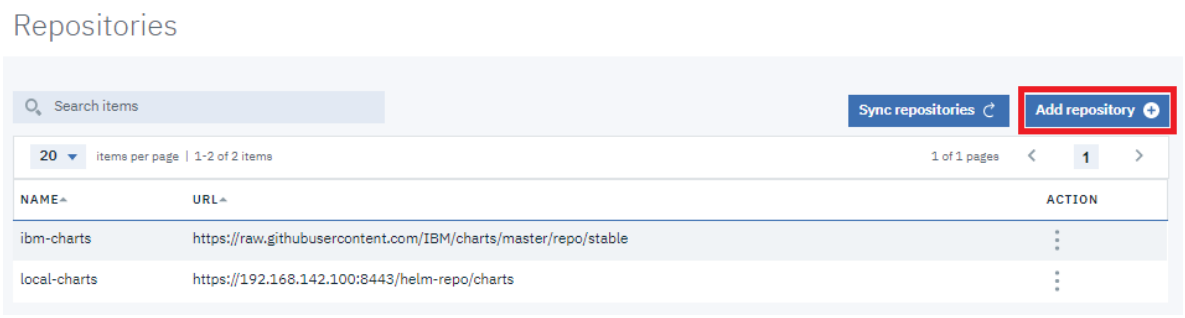


By default there are two repositories.

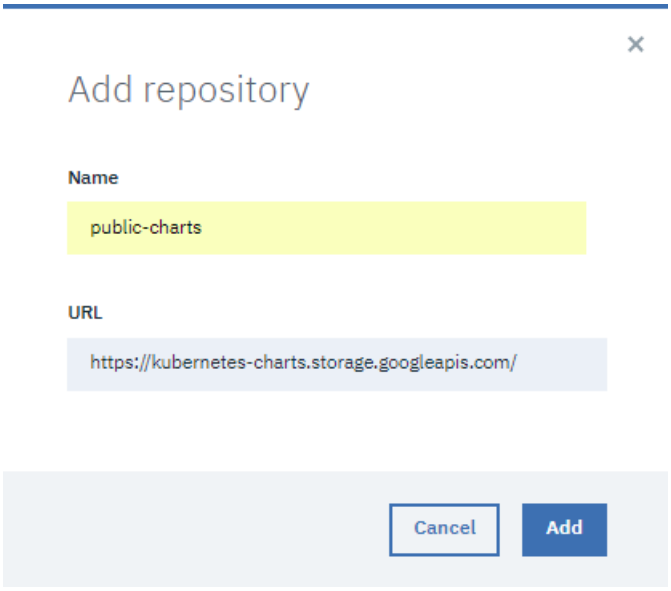
Ibm-charts is the repository we have just used where are all the carts to install IBM software

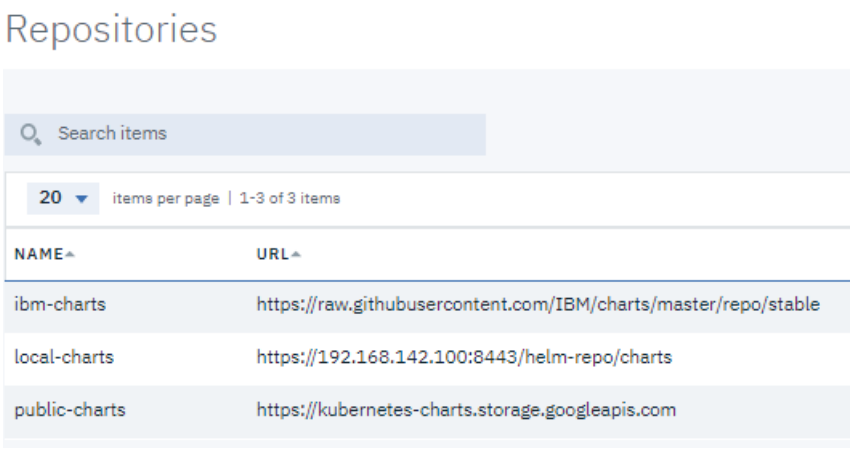
Local-charts is an internal repository where ICP users can register their own charts.

Click on “Add repository” to add an extern repository

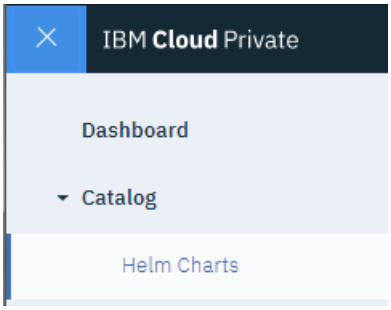


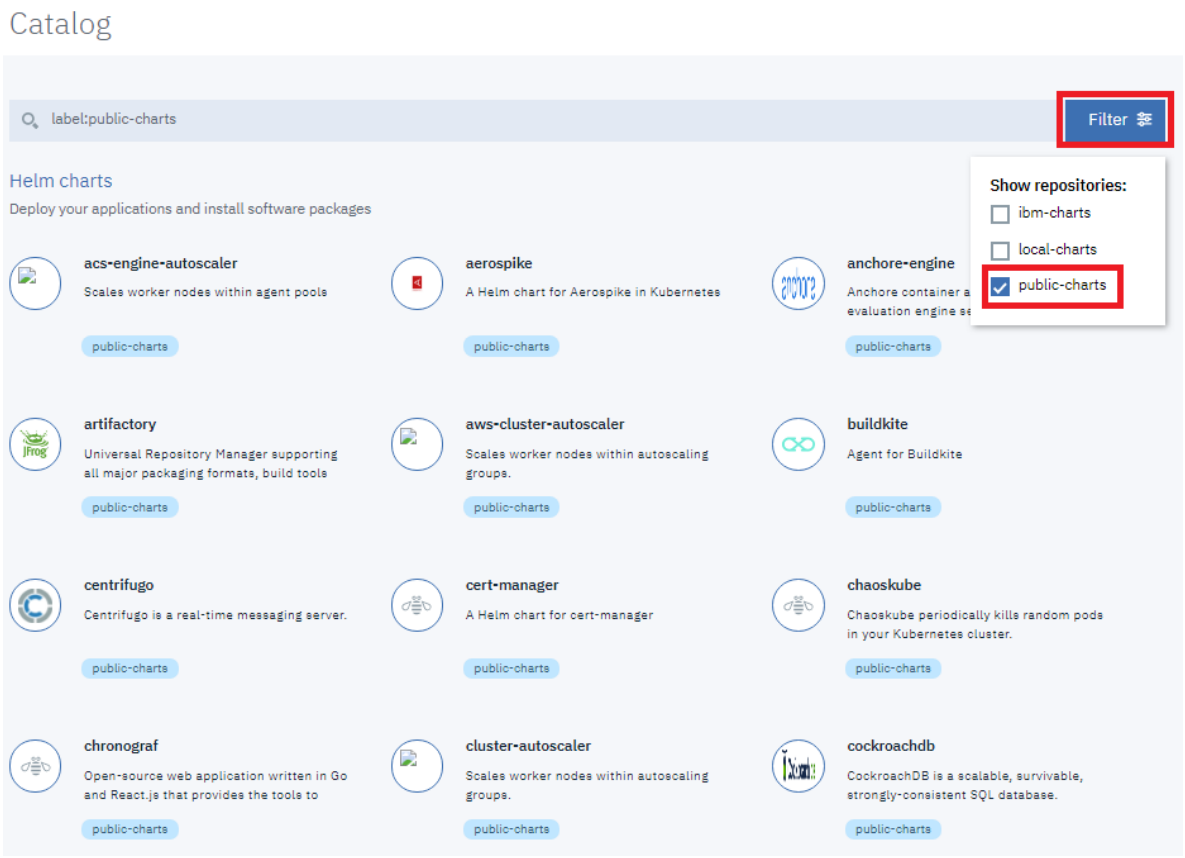
Add the repository https://kubernetes-charts.storage.googleapis.com/





Go to the HELM Catalog and check the new charts.



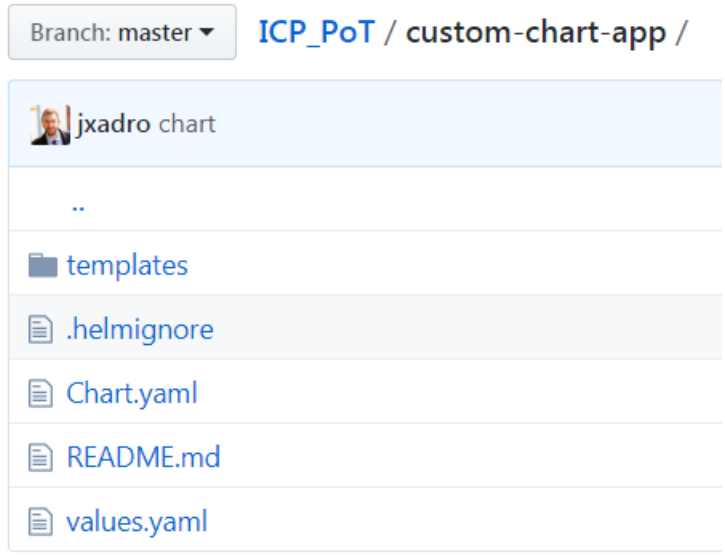


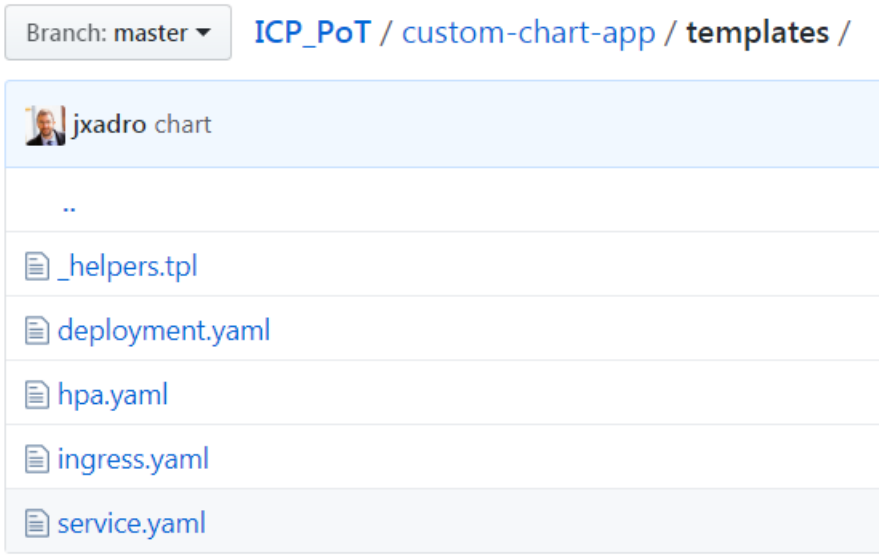
Custom Chart

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_2.1.0/app_center/add_package.htm

A chart is a zip file of kubernetes objects as the files we used in previous labs to create deployment and service objects.

Access to the url https://github.com/jxadro/ICP_PoT/tree/master/custom-chart-app





This is the structure of a chart.

Inside the folder “templates” resides all the objects that are going to be created during the char installation. It can be as many objects as needed. Although in this case there is only one deployment if we would install a microservice application there would be several deployments.

These files use external parameters so the user can set the desired values during the chart installation:

Branch: master ▾ ICP_PoT / custom-chart-app / templates / service.yaml

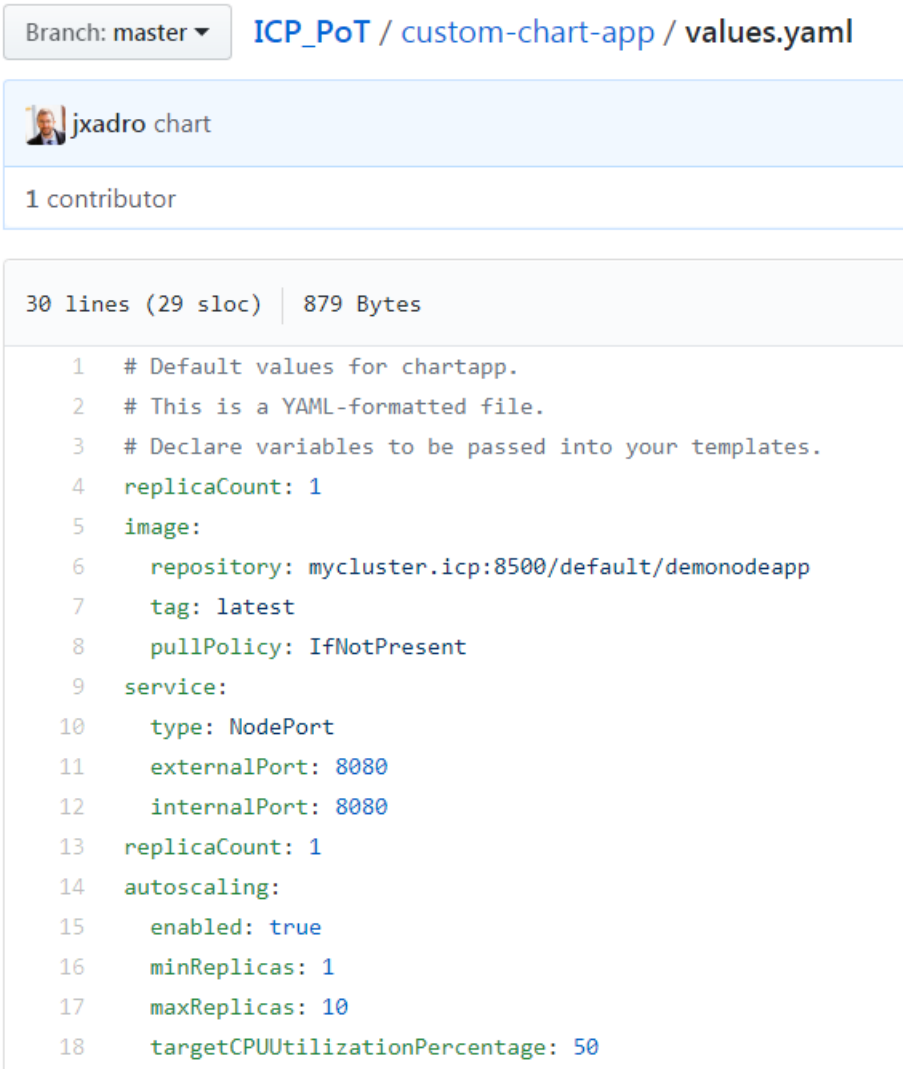
jxadro chart

1 contributor

19 lines (18 sloc) | 494 Bytes

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: {{ .Release.Name }}
5    labels:
6      app: {{ .Release.Name }}
7      chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
8      release: {{ .Release.Name }}
9      heritage: {{ .Release.Service }}
10 spec:
11   type: {{ .Values.service.type }}
12   ports:
13     - port: {{ .Values.service.externalPort }}
14       targetPort: {{ .Values.service.internalPort }}
15       protocol: TCP
16       name: {{ .Values.service.name }}
17   selector:
18     app: {{ .Release.Name }}
```

The default values are taken from the file values.yaml



Lab 6. Configure Kubernetes to manage the application

List the pods.

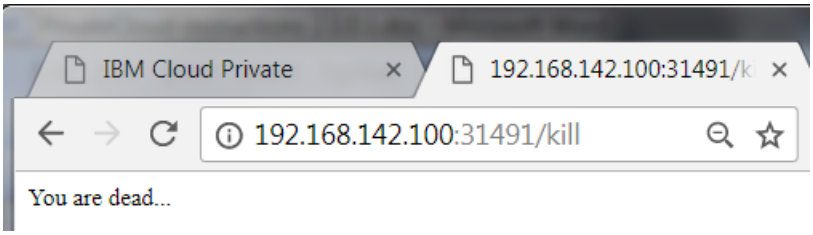
kubectl get pods
NAME READY STATUS RESTARTS AGE
hellonode-546947b56f-hvq77 1/1 Running 0 18m
....

See that the hellonode pod has not been restarted anytime yet. RESTARTS = 0

```
osboxes@osboxes:~$ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
hellonode-546947b56f-hvq77          1/1       Running   0          18m
```

Automatic restarts

Invoke /kill , The /kill method executes the sentence “process.exit();” what makes the node.js process quit and finish the container.



After invoking /kill, if you execute in the command line kubectl get pods, you will see that the hellonode pod has status completed and 0/1 Ready. But as when we deployed the Application we set that we wanted 1 replica active, when kubernetes detects that there are less replicas active than the number configured it automatically restarts the pods.

So if you execute again get pods you will see that the hellonode pod has ready 1/1, status running and 1 restart.

```
osboxes@osboxes:~$ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
hellonode-546947b56f-hvq77          0/1       Completed 0          22m
```

```
osboxes@osboxes:~$ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
hellonode-546947b56f-hvq77          1/1       Running   1          22m
```

In case of deleting the pod, kubernetes will create a new one:

```
osboxes@osboxes:~$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
hellonode-546947b56f-hvq77    1/1       Running   1          26m
osboxes@osboxes:~$ kubectl delete pod hellonode-546947b56f-hvq77
pod "hellonode-546947b56f-hvq77" deleted
osboxes@osboxes:~$ kubectl get pods
NAME                          READY     STATUS              RESTARTS   AGE
hellonode-546947b56f-tmsc7    0/1       ContainerCreating   0          5s
osboxes@osboxes:~$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
hellonode-546947b56f-tmsc7    1/1       Running   0          12s
```

Query application health

We want kubernetes to ask the application its status, if it does not return a 200 OK, Kubernetes will restart the Application.

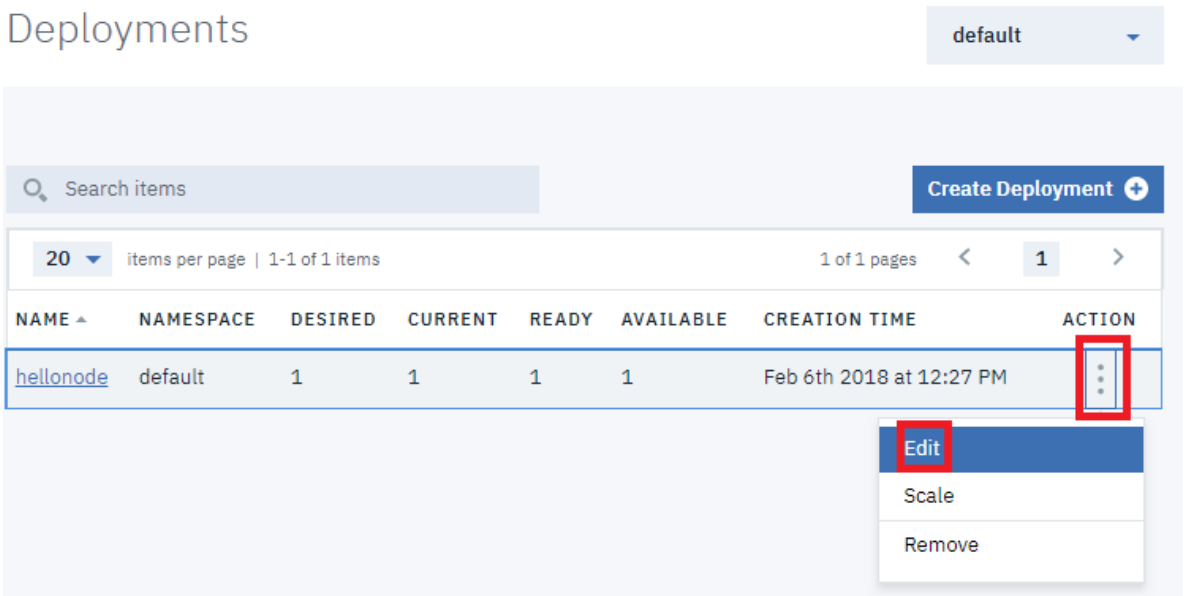
In the node.js application we used to create our hellonode docker images we have two methods:

```
app.get("/infect", function(req, res, next){
        isHealthy = false;

app.get("/health", function(req, res, next){
        if(isHealthy)
                res.send("GREEN");
        else
                res.status(500).send("RED");
```

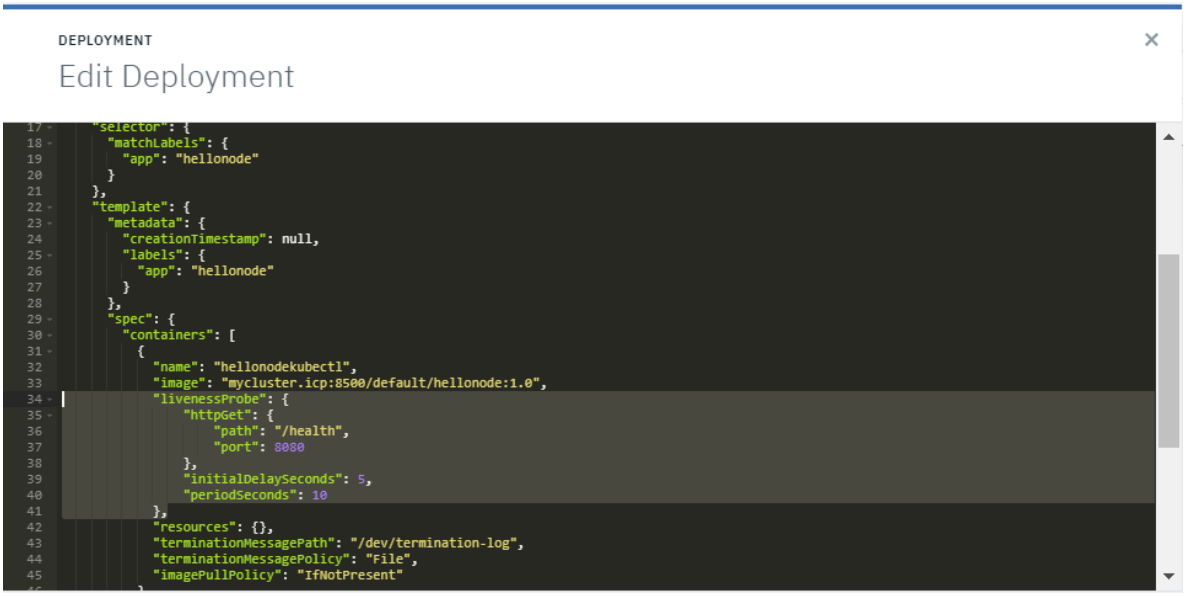
So after invoking the infect method, the app will start returning HTTP 500 code.

First we need to configure our Application to poll the application status.

Edit the hellonode application in the IBM Cloud Private console:



Modify the deployment descriptor to add the element livenessProbe, this element is used by kubernetes to poll the application asking for the status, in case it does not return 200 OK the Application will be restarted.




```
"livenessProbe": {
   "httpGet": {
      "path": "/health",
      "port": 8080
   },
   "initialDelaySeconds": 5,
   "periodSeconds": 10
},
```

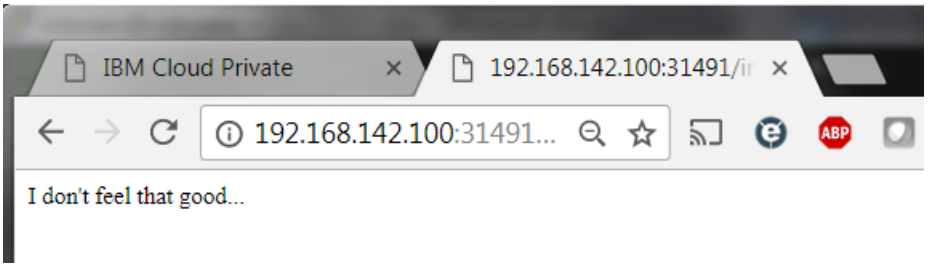
Wit this configuration kubernetes will invoke the operation /heath in periods of 10 seconds, if three consecutives times the application does not returns 200 OK, kubernetes will restart the Application.

In the command line execute kubectl get pods and see the logs

```
osboxes@osboxes:~$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
hellonode-c9576c598-gb917     1/1       Running   0          54s
osboxes@osboxes:~$ kubectl logs -f hellonode-c9576c598-gb917

> HelloKube@1.0.0 start /app
> node app.js

[2018-02-06 11:58:40.566] [INFO] HelloKube - Starting...
[2018-02-06 11:58:40.582] [INFO] HelloKube - HelloKube listening on port 8080
[2018-02-06 11:58:43.573] [INFO] HelloKube - lalala....
[2018-02-06 11:58:44.325] [INFO] HelloKube - operation /health invoked... returning GREEN
[2018-02-06 11:58:46.574] [INFO] HelloKube - lalala....
[2018-02-06 11:58:49.575] [INFO] HelloKube - lalala....
```

Take note of the number of restarts.

Invoke /infect method.



(See that the port does not change after restarting the pods, that is the objective of the kubernetes service objects, they automatically reflect and balance the traffic to the application configured in the service selector)

Look at the logs. After the method /health is invoked three times returning RED the Application will be restarted.

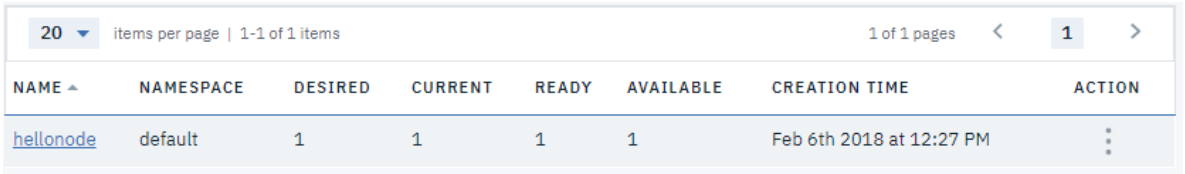
```
[2018-02-06 12:01:29.368] [INFO] HelloKube - operation /infect invoked...
[2018-02-06 12:01:31.855] [INFO] HelloKube - cough...
[2018-02-06 12:01:34.317] [INFO] HelloKube - operation /health invoked... returning RED
[2018-02-06 12:01:34.855] [INFO] HelloKube - cough...
[2018-02-06 12:01:37.856] [INFO] HelloKube - cough...
[2018-02-06 12:01:40.859] [INFO] HelloKube - cough...
[2018-02-06 12:01:43.861] [INFO] HelloKube - cough...
[2018-02-06 12:01:44.318] [INFO] HelloKube - operation /health invoked... returning RED
[2018-02-06 12:01:46.863] [INFO] HelloKube - cough...
[2018-02-06 12:01:49.867] [INFO] HelloKube - cough...
[2018-02-06 12:01:52.869] [INFO] HelloKube - cough...
[2018-02-06 12:01:54.319] [INFO] HelloKube - operation /health invoked... returning RED
osboxes@osboxes:~$
```

```
osboxes@osboxes:~$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
hellonode-c9576c598-gb917   1/1       Running   1          8m
osboxes@osboxes:~$ kubectl logs -f hellonode-c9576c598-gb917

> HelloKube@1.0.0 start /app
> node app.js

[2018-02-06 12:02:04.270] [INFO] HelloKube - Starting...
[2018-02-06 12:02:04.289] [INFO] HelloKube - HelloKube listening on port 8080
[2018-02-06 12:02:04.326] [INFO] HelloKube - operation /health invoked... returning GREEN
[2018-02-06 12:02:07.276] [INFO] HelloKube - lalala....
```

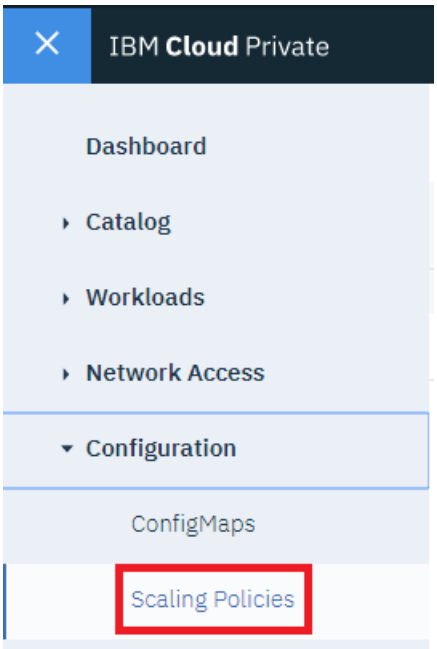
AutoScale

Create a policy to scale your application, you can configure IBM Cloud private to scale the Application based on CPU Usage.

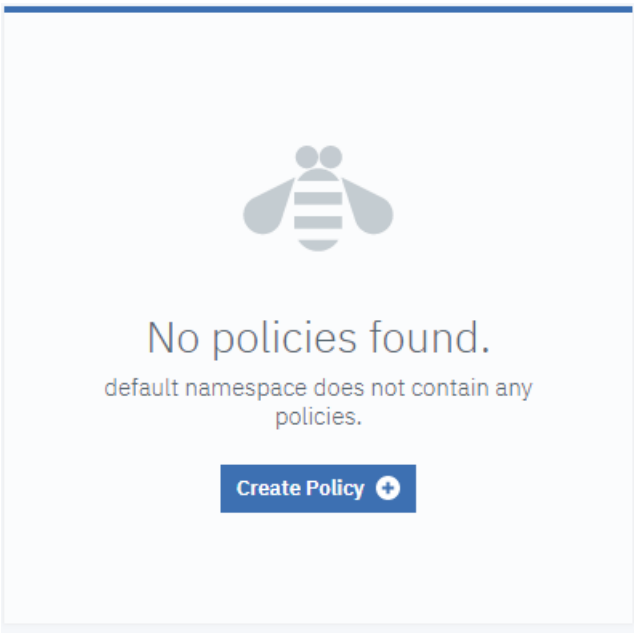
Before creating a new policy take note of the current number of replicas:

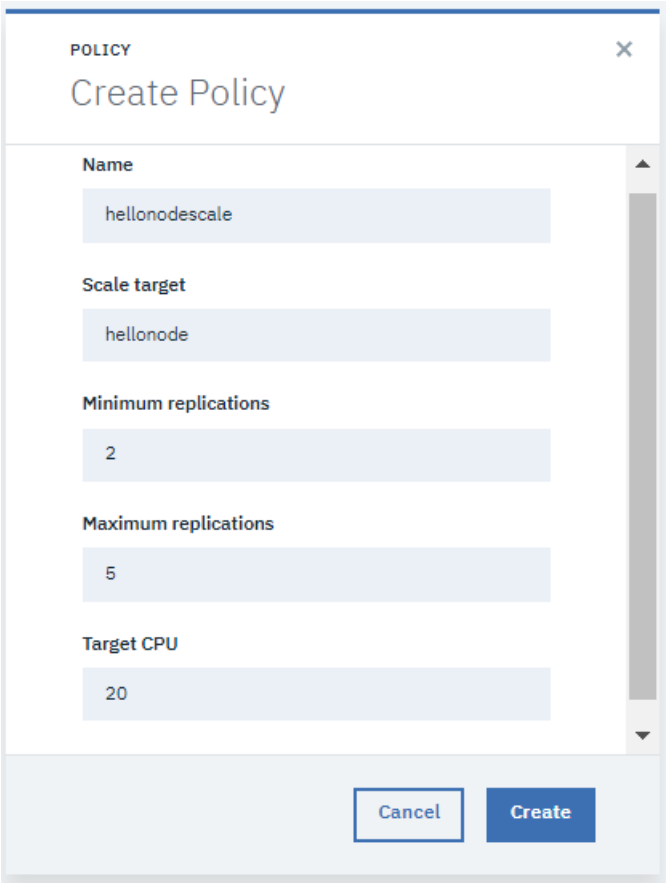


Go to Workload -> Scaling Policies



Create a new policy:

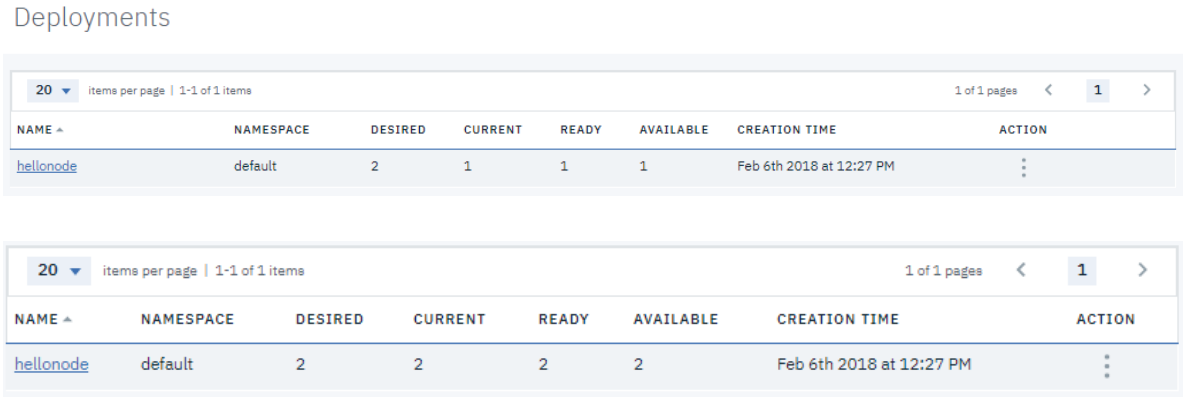




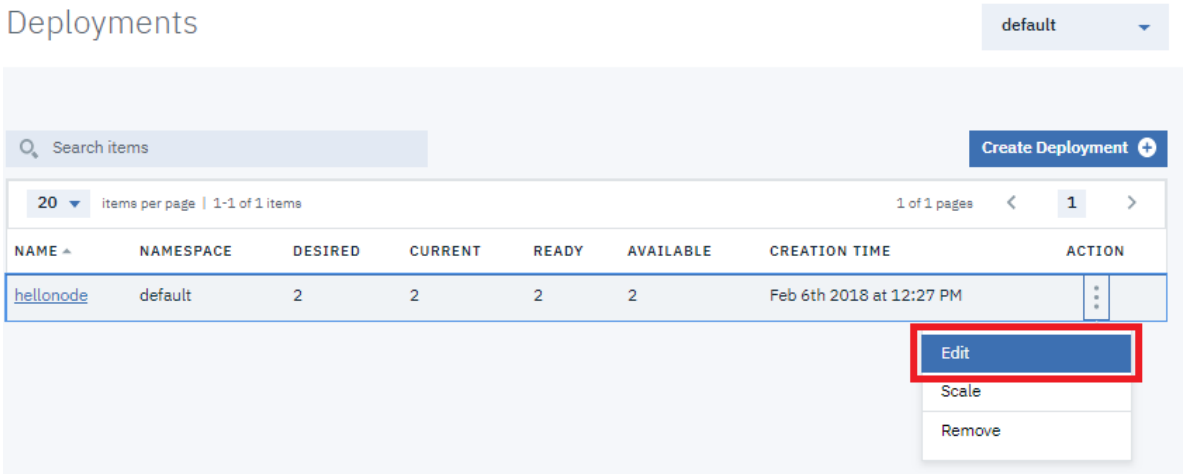
YAML:

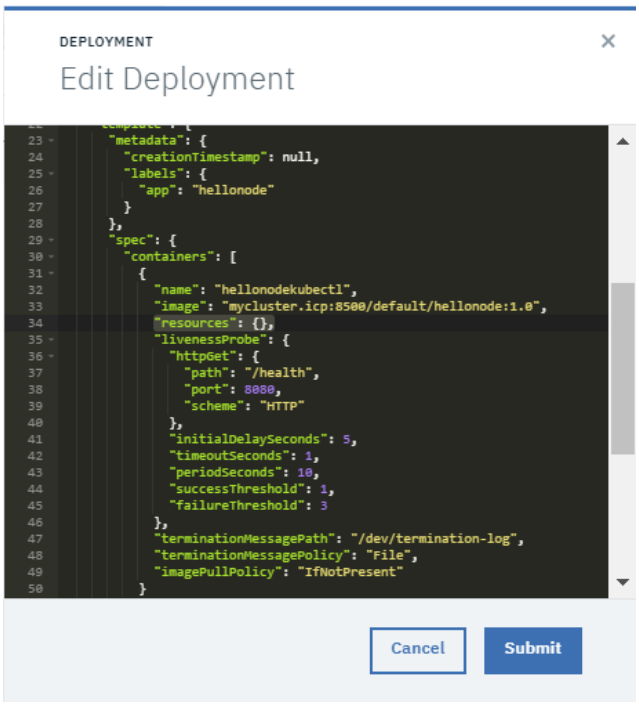
```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hellonodescale
  namespace: default
spec:
  scaleTargetRef:
    kind: Deployment
    name: hellonode
    apiVersion: extensions/v1beta1
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 20
```

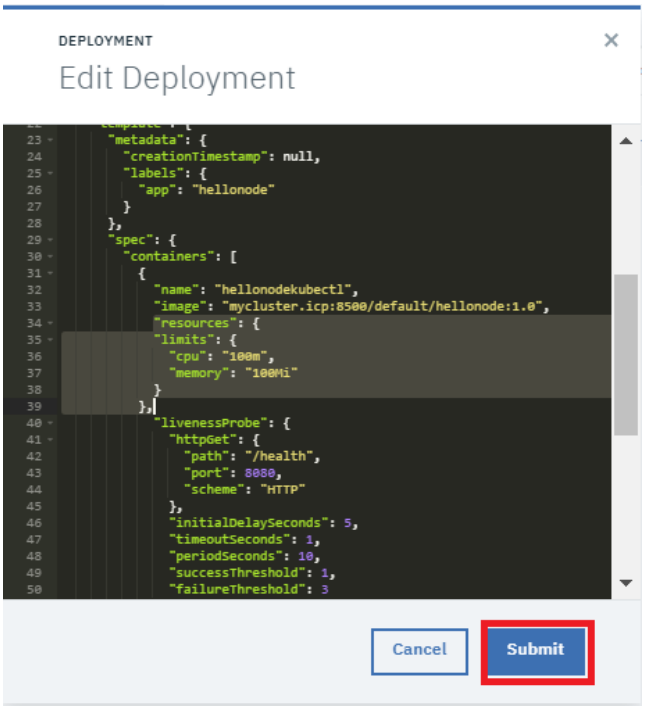
If you go to the deployments you will see that kubernetes is scaling the application to 2 pods as it is the minimum specified in the new policy



Now we are going to query the application in an infinity loop to generate load and make that the CPU consumed by the application grows beyond 20%. But first we need to set the CPU required by the application/deployment.








```
"resources": {
        "requests": {
                "cpu": "100m",
                "memory": "100Mi"
        }
},
```

Query the application.

for ((i=1;10<=100;i++)); do   curl -H "Connection: close" --connect-timeout 1 --connect-timeout 1 http://192.168.142.100:31491/; done

```
osboxes@osboxes:~$ for ((i=1;10<=100;i++)); do   curl -H "Connection: close" --connect-timeout 1 --connect-timeout 1 http://192.168.142.100:31491/; done
I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!
!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm ali
ive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm
```

With the commands “kubectl describe horizontalpodautoscaler hellonodescale” you can get the status of the policy. You can see the current percentage of CPU used and the actions taken.

```
osboxes@osboxes:~$ kubectl describe horizontalpodautoscaler hellonodescale
Name:                                                  hellonodescale
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Wed, 07 Feb 2018 14:48:46 +0000
Reference:                                             Deployment/hellonode
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  46% (46m) / 20%
Min replicas:                                          2
Max replicas:                                          5
Conditions:
  Type            Status  Reason             Message
  ----            ------  ------             -------
  AbleToScale     True    SucceededRescale   the HPA controller was able to update the target scale to 4
  ScalingActive   True    ValidMetricFound   the HPA was able to succesfully calculate a replica count from cpu resource utilization (percentage of request)
```

20 items per page | 1-1 of 1 items 1 of 1 pages 1

NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
hellonode	default	4	4	4	4	Feb 6th 2018 at 12:27 PM	⋮

20 items per page | 1-1 of 1 items 1 of 1 pages 1

NAME	NAMESPACE	DESIRED	CURRENT	READY	AVAILABLE	CREATION TIME	ACTION
hellonode	default	5	5	5	5	Feb 6th 2018 at 12:27 PM	⋮

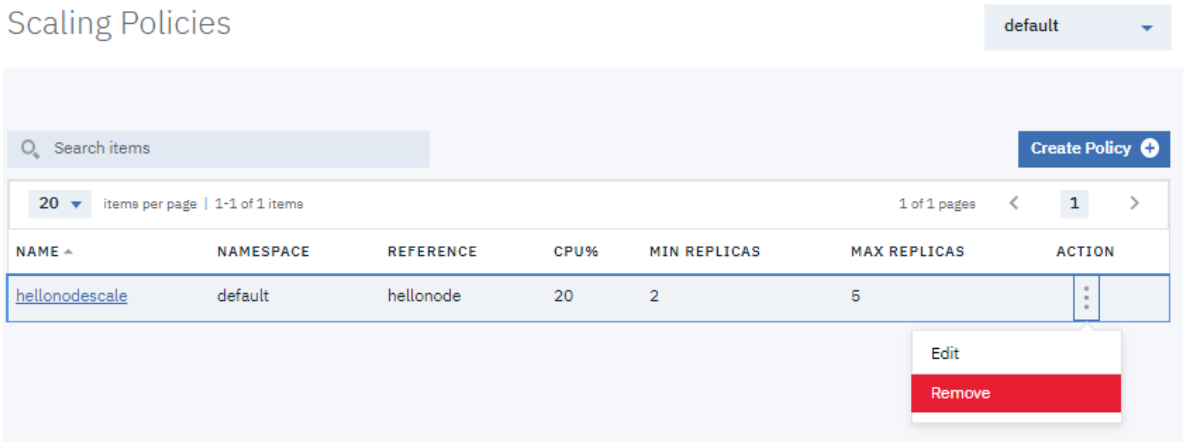
From Kubernetes 1.6 it is possible to user more metrics than CPU % usage.

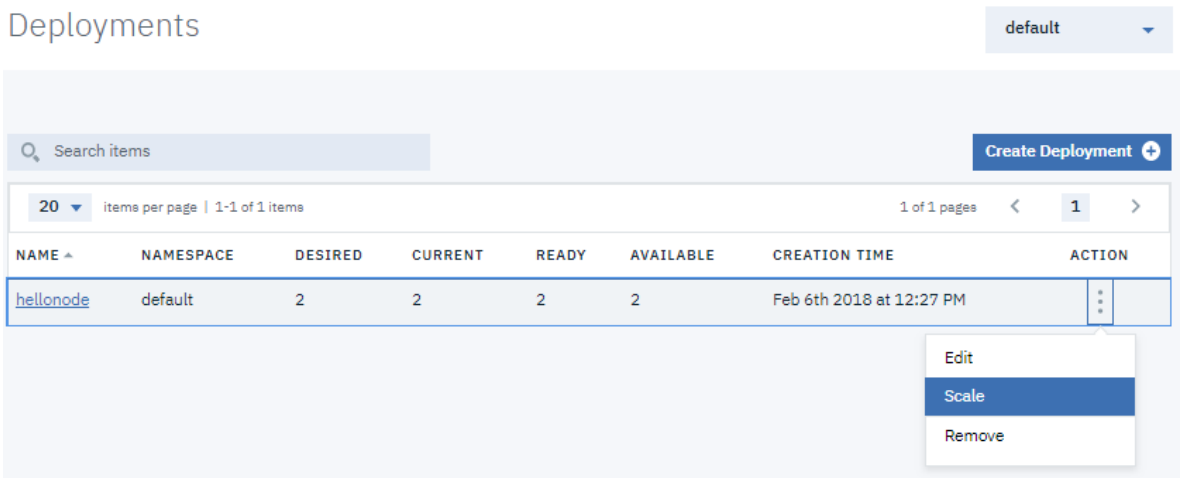
Rollouts. Update without down time.

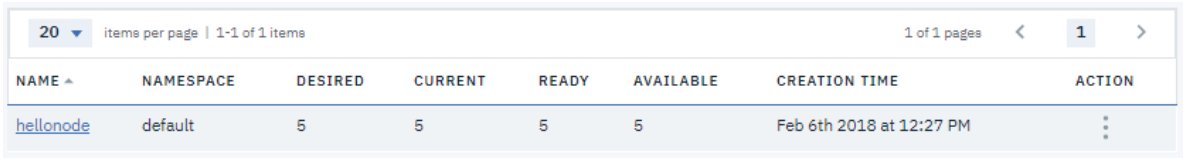
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#updating-a-deployment

In this section we are going to change the docker image version used by an application/deployment. Kubernetes does it progressively without downtime.

Remove the policy we created in the previous lab and set the number of replicas to 5. The biggest number of replicas the smoother the rollout will be.







We are going to create a new version of our hellonode docker image used in Lab3.

Go to the command line and edit the file ./node/app.js

```
osboxes@osboxes:~$ pwd
/home/osboxes
osboxes@osboxes:~$ vi ./node/app.js
osboxes@osboxes:~$ cat ./node/app.js
...
...
app.get("/", function(req, res, next){
      logger.info("operation / invoked...");
```

```
    res.send("I'm alive - 2 !!! ");
});
...
...
```

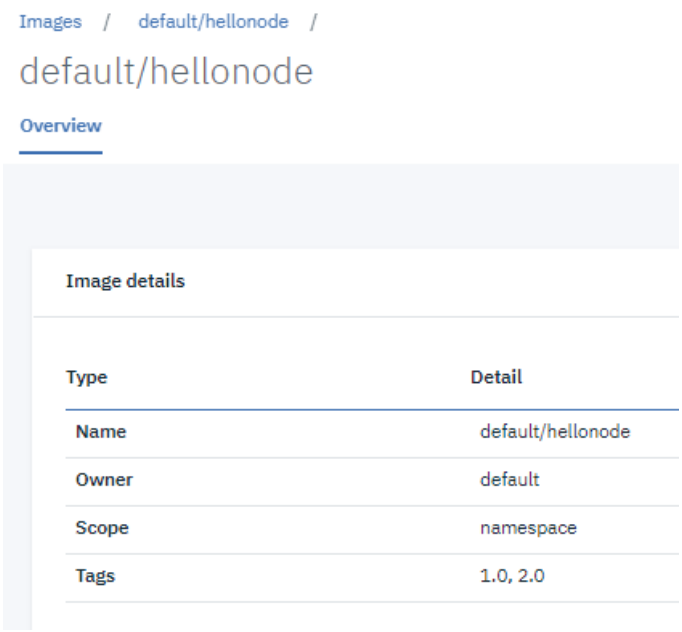
Create the docker image with version tag: 2.0

osboxes@osboxes:~$ sudo docker build -t mycluster.icp:8500/default/hellonode:2.0 -f hellonode.dockerfile .

Push the image to IBM Cloud private image repository:

osboxes@osboxes:~$ sudo docker push mycluster.icp:8500/default/hellonode:2.0

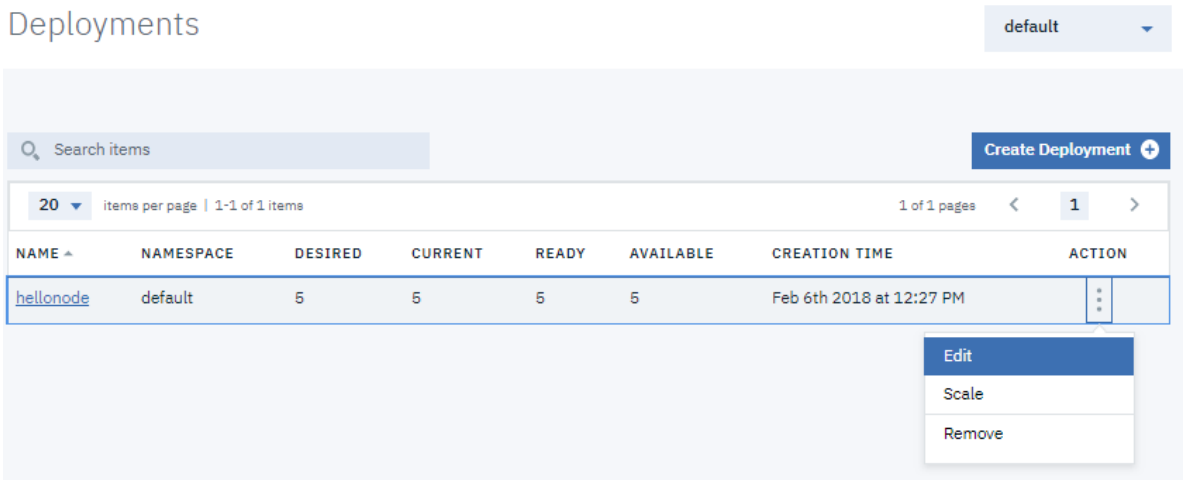
Check the available images in our repository:



To be able to rollout the application without downtime we need to add the property “readinessprobe” to our application/deployment.

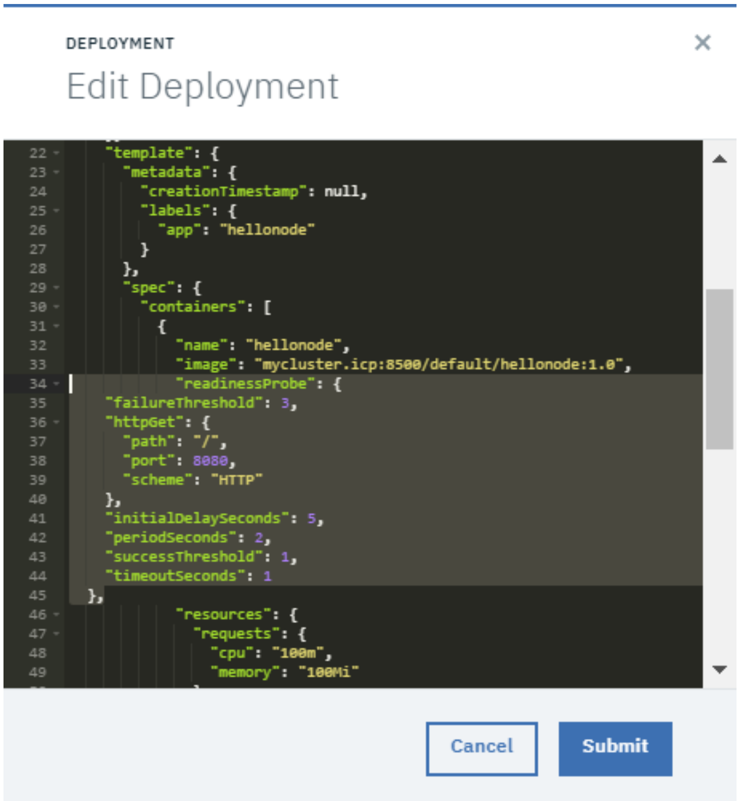
This property tells kubernetes how to check when the application is ready so kubernetes will enable the new pod IP in the service object only when the app is ready and not when just the container is ready.

Edit the Application:



Add the following element:

```
 "readinessProbe": {
    "failureThreshold": 3,
    "httpGet": {
      "path": "/",
      "port": 8080,
      "scheme": "HTTP"
    },
    "initialDelaySeconds": 5,
    "periodSeconds": 2,
    "successThreshold": 1,
    "timeoutSeconds": 1
  }
```



Double check you removed the scaling policy and you scale the application to 5 instances.

Open one terminal and execute the command: kubectl rollout status deployment/hellonode

```
osboxes@osboxes:~$ kubectl rollout status deployment/hellonode
Waiting for rollout to finish: 4 of 5 updated replicas are available...
deployment "hellonode" successfully rolled out
```

You see the result of our last update (rollout).

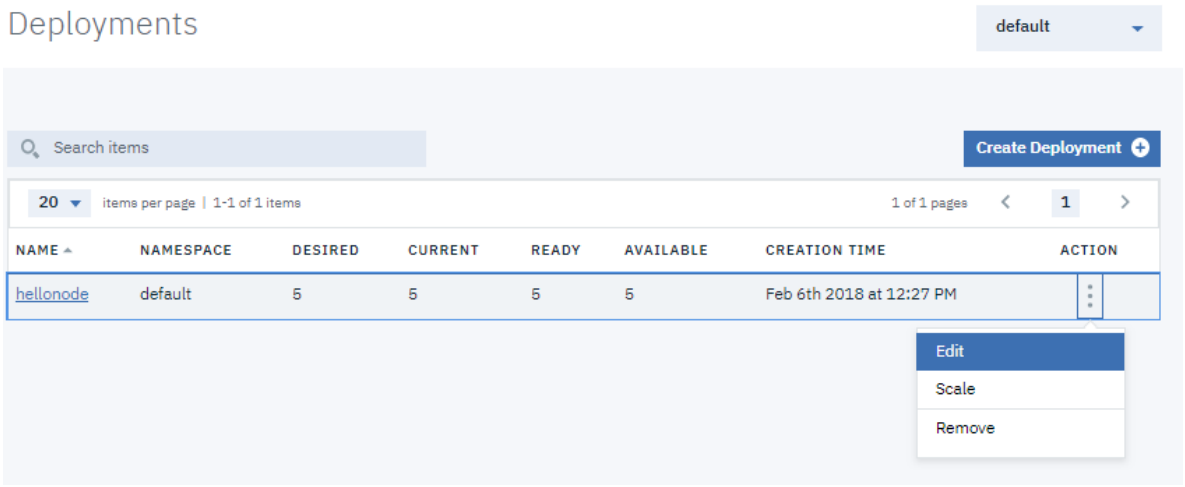
Open other terminal and query the application:

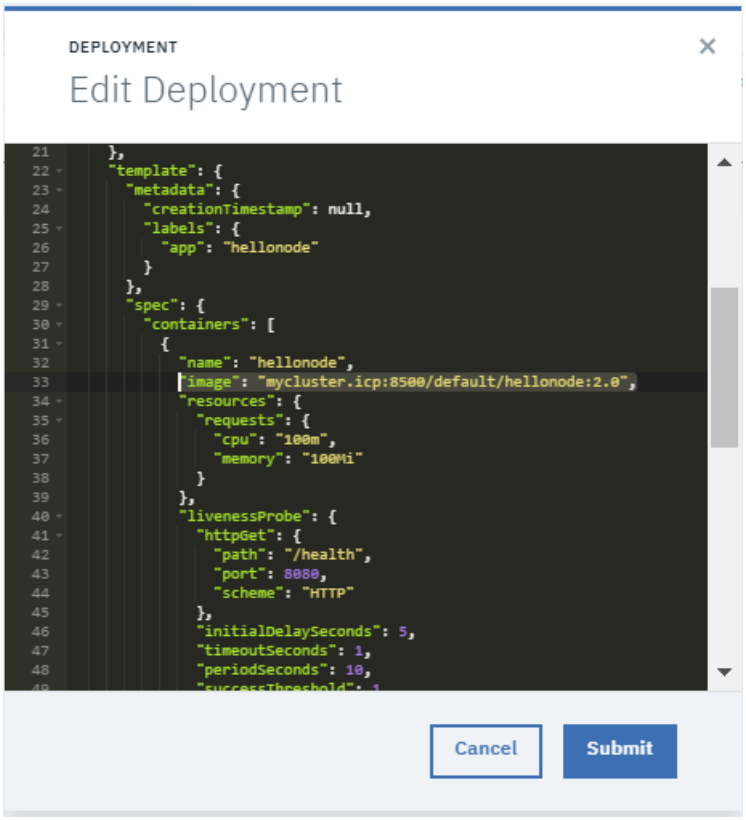
for ((i=1;10<=100;i++)); do   curl -H "Connection: close" --connect-timeout 1 --connect-timeout 1 http://192.168.142.100:31491/; done

```
osboxes@osboxes:~$ for ((i=1;10<=100;i++)); do   curl -H "Connection: close" --connect-timeout 1 --connect-timeout 1 http://192.168.142.100:31491/; done
I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm
!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!I'm alive!!!
```

I'm alive!!! is the only message corresponding to the version 1.

Edit the application/deployment and change the docker image version.

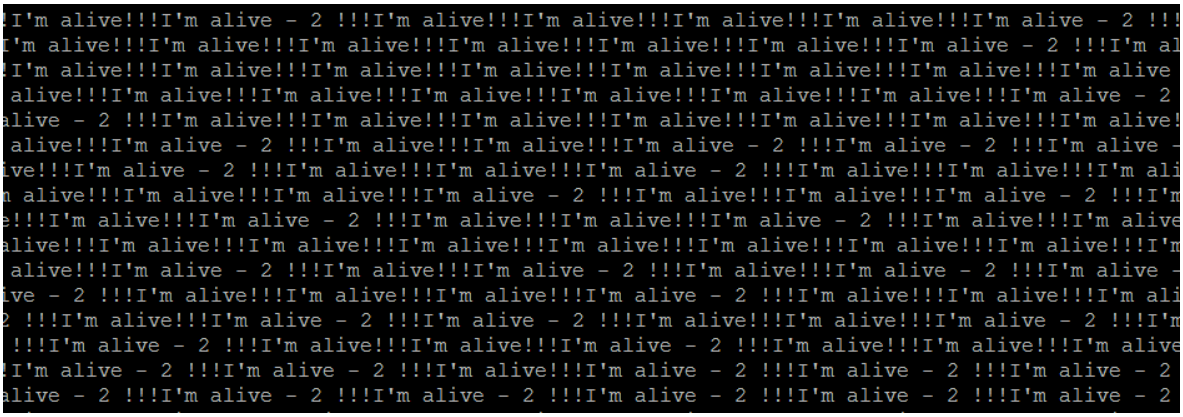


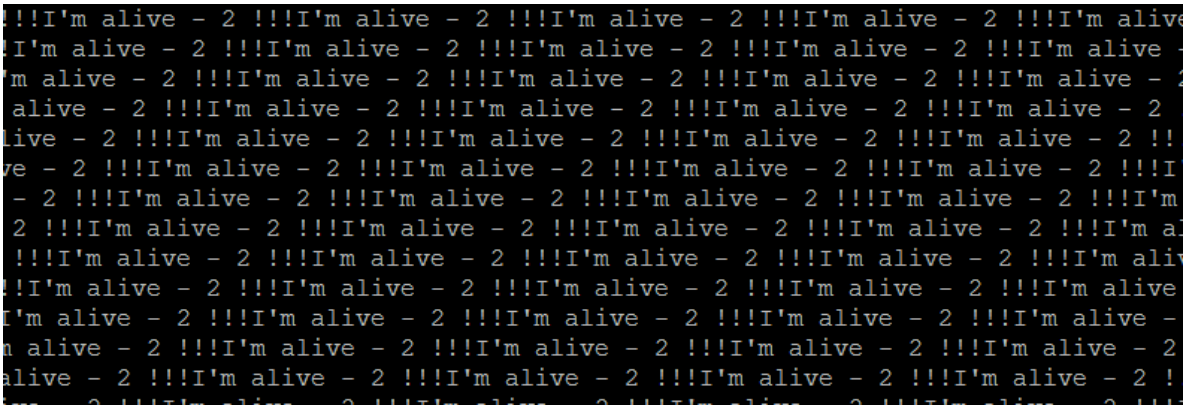


Execute again the command kubectl rollout status deployment/hellonode, you will see how kubernetes rollout your application

```
osboxes@osboxes:~$ kubectl rollout status deployment/hellonode
Waiting for rollout to finish: 2 out of 5 new replicas have been updated...
Waiting for rollout to finish: 2 out of 5 new replicas have been updated...
Waiting for rollout to finish: 2 out of 5 new replicas have been updated...
Waiting for rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for rollout to finish: 1 old replicas are pending termination...
Waiting for rollout to finish: 1 old replicas are pending termination...
Waiting for rollout to finish: 4 of 5 updated replicas are available...
Waiting for rollout to finish: 3 of 5 updated replicas are available...
Waiting for rollout to finish: 4 of 5 updated replicas are available...
deployment "hellonode" successfully rolled out
```

And in the logs you will see first a mix of messages and then only I'm alive -2 !!! messages.





We can control how the rollout update is done with the property "strategy". For example:

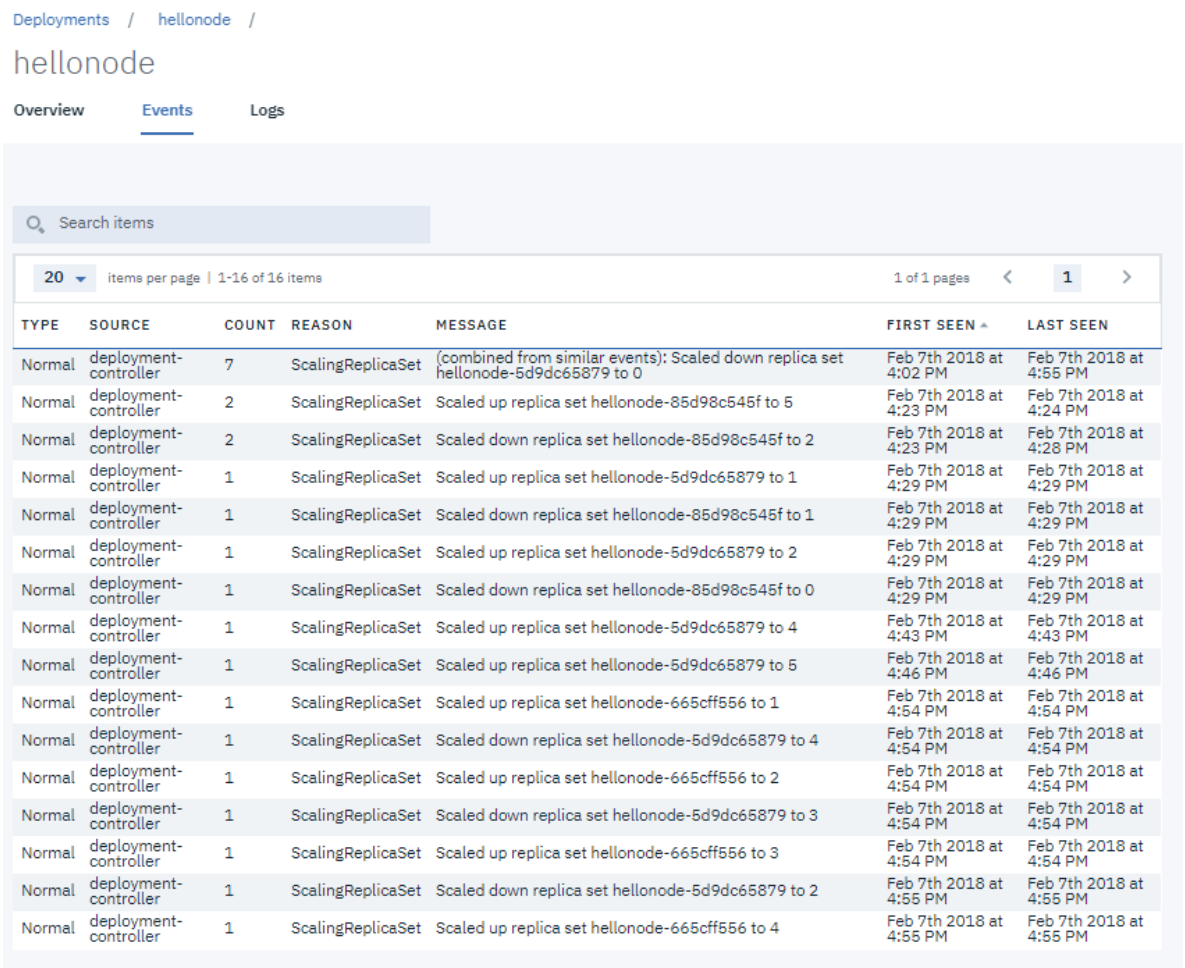
"strategy": {

```
    "type": "RollingUpdate",
    "rollingUpdate": {
      "maxUnavailable": 1,
      "maxSurge": 1
    }
  }
```

This element means that during a rollout as maximum (maxSurge) there can only be 1 more pod than the replicas configured. And that as maximum (maxUnavailabe) there can only be 1 pod less available than the replicas configured.

So in the Application status we will never see more than 6 as current and less than 4 as available.

In the application details:



TYPE	SOURCE	COUNT	REASON	MESSAGE	FIRST SEEN	LAST SEEN
Normal	deployment-controller	7	ScalingReplicaSet	(combined from similar events): Scaled down replica set hellonode-5d9dc65879 to 0	Feb 7th 2018 at 4:02 PM	Feb 7th 2018 at 4:55 PM
Normal	deployment-controller	2	ScalingReplicaSet	Scaled up replica set hellonode-85d98c545f to 5	Feb 7th 2018 at 4:23 PM	Feb 7th 2018 at 4:24 PM
Normal	deployment-controller	2	ScalingReplicaSet	Scaled down replica set hellonode-85d98c545f to 2	Feb 7th 2018 at 4:23 PM	Feb 7th 2018 at 4:28 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-5d9dc65879 to 1	Feb 7th 2018 at 4:29 PM	Feb 7th 2018 at 4:29 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled down replica set hellonode-85d98c545f to 1	Feb 7th 2018 at 4:29 PM	Feb 7th 2018 at 4:29 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-5d9dc65879 to 2	Feb 7th 2018 at 4:29 PM	Feb 7th 2018 at 4:29 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled down replica set hellonode-85d98c545f to 0	Feb 7th 2018 at 4:29 PM	Feb 7th 2018 at 4:29 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-5d9dc65879 to 4	Feb 7th 2018 at 4:43 PM	Feb 7th 2018 at 4:43 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-5d9dc65879 to 5	Feb 7th 2018 at 4:46 PM	Feb 7th 2018 at 4:46 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-665cff556 to 1	Feb 7th 2018 at 4:54 PM	Feb 7th 2018 at 4:54 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled down replica set hellonode-5d9dc65879 to 4	Feb 7th 2018 at 4:54 PM	Feb 7th 2018 at 4:54 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-665cff556 to 2	Feb 7th 2018 at 4:54 PM	Feb 7th 2018 at 4:54 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled down replica set hellonode-5d9dc65879 to 3	Feb 7th 2018 at 4:54 PM	Feb 7th 2018 at 4:54 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-665cff556 to 3	Feb 7th 2018 at 4:54 PM	Feb 7th 2018 at 4:54 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled down replica set hellonode-5d9dc65879 to 2	Feb 7th 2018 at 4:55 PM	Feb 7th 2018 at 4:55 PM
Normal	deployment-controller	1	ScalingReplicaSet	Scaled up replica set hellonode-665cff556 to 4	Feb 7th 2018 at 4:55 PM	Feb 7th 2018 at 4:55 PM

We see the scale up and scale down events.

Lab 7. Logging

By default docker and kubernetes can access to the logs that a container prints to its STDOUT. There are also options to access to internal or custom containers logs.

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_2.1.0/manage_metrics/logging_elk.html

Command line

List the pods.

kubectl get pods

```
osboxes@osboxes:~$ kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
hellonode-665cff556-bs4ds    1/1       Running   0          17h
hellonode-665cff556-f8b7b    1/1       Running   1          17h
hellonode-665cff556-pmn7s    1/1       Running   0          17h
hellonode-665cff556-pv7g2    1/1       Running   0          1m
hellonode-665cff556-qdjhr    1/1       Running   1          17h
```

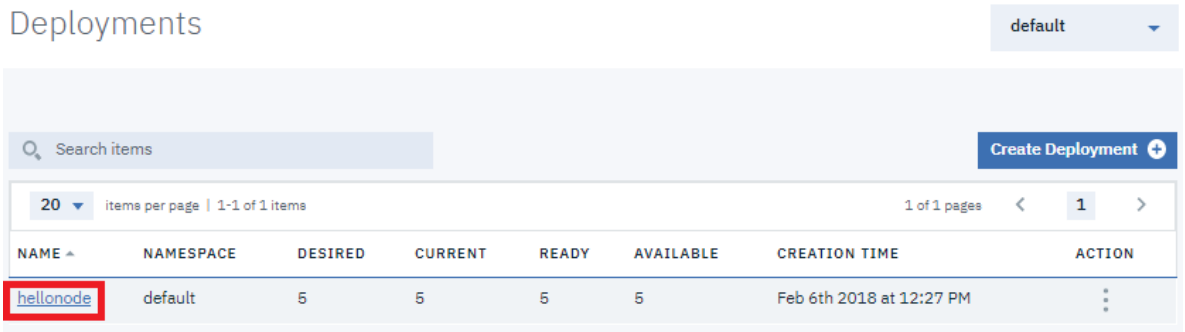
Tail the logs.

kubectl logs –f kubectl logs -f hellonode-665cff556-pv7g2

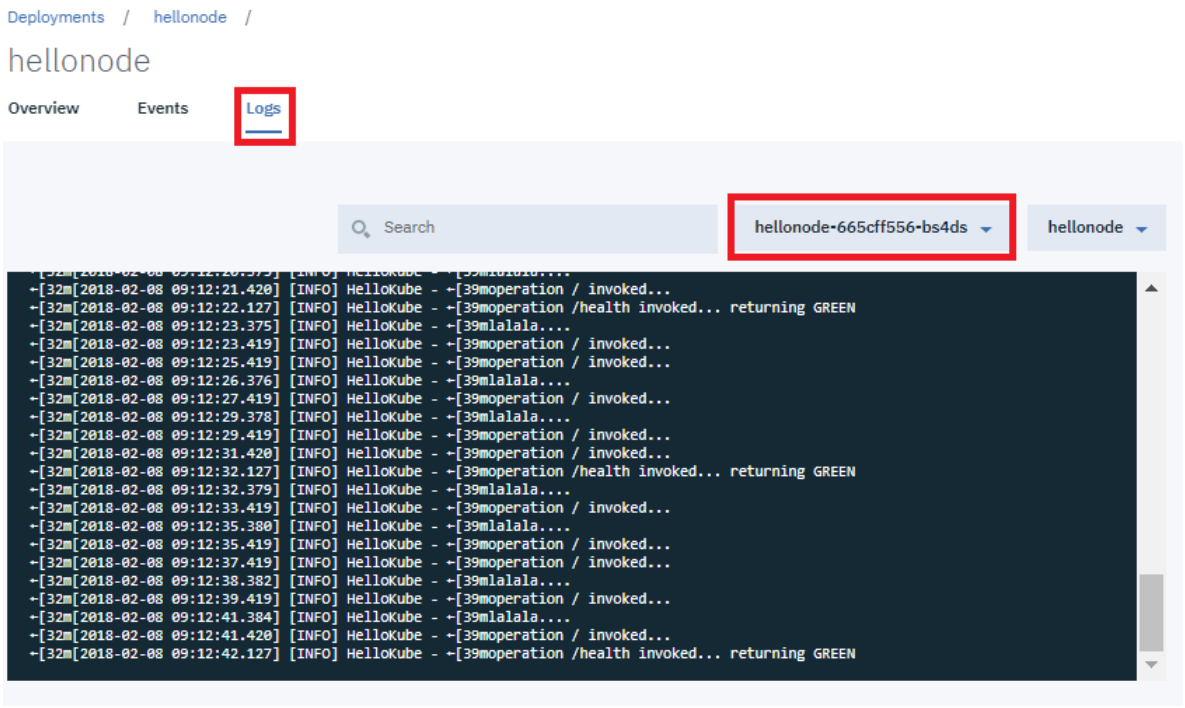
```
osboxes@osboxes:~$ kubectl logs -f hellonode-665cff556-pv7g2

> HelloKube@1.0.0 start /app
> node app.js

[2018-02-08 09:09:49.846] [INFO] HelloKube - Starting...
[2018-02-08 09:09:50.189] [INFO] HelloKube - HelloKube listening on port 8080
[2018-02-08 09:09:51.698] [INFO] HelloKube - operation / invoked...
[2018-02-08 09:09:52.980] [INFO] HelloKube - lalala....
[2018-02-08 09:09:53.693] [INFO] HelloKube - operation / invoked...
[2018-02-08 09:09:55.709] [INFO] HelloKube - operation / invoked...
[2018-02-08 09:09:55.981] [INFO] HelloKube - lalala....
[2018-02-08 09:09:57.692] [INFO] HelloKube - operation / invoked...
[2018-02-08 09:09:58.983] [INFO] HelloKube - lalala....
[2018-02-08 09:09:59.691] [INFO] HelloKube - operation / invoked...
[2018-02-08 09:10:01.690] [INFO] HelloKube - operation / invoked...
[2018-02-08 09:10:01.983] [INFO] HelloKube - lalala....
```

ICP Console

Go to deployments and select your application/deployment.

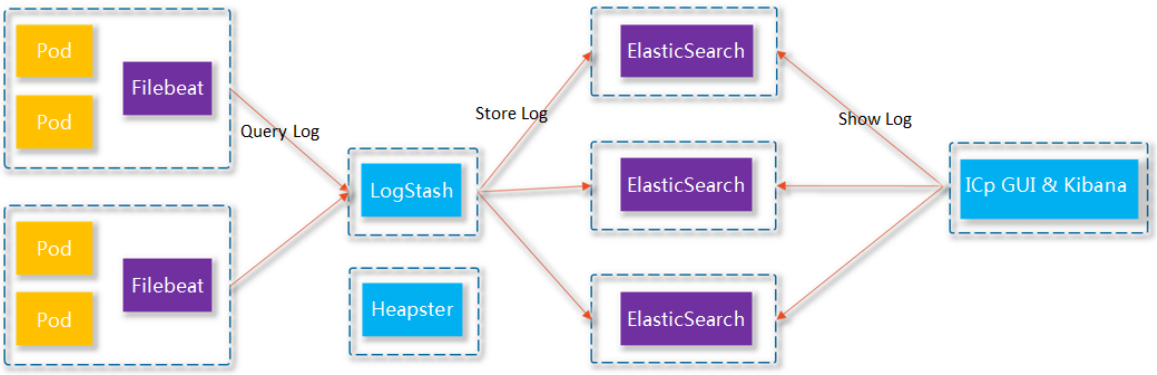


Select the tab “Logs” and review the logs of the pods of your deployment.

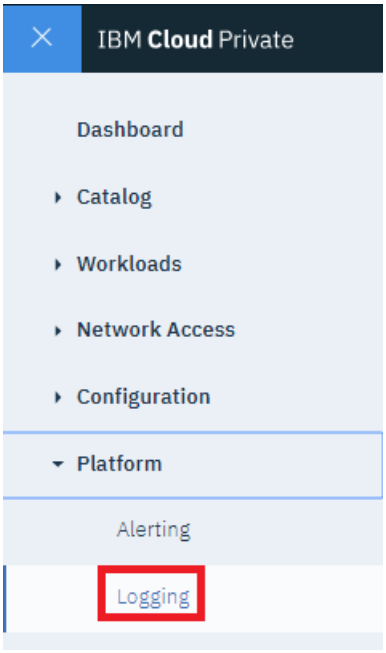


ELK

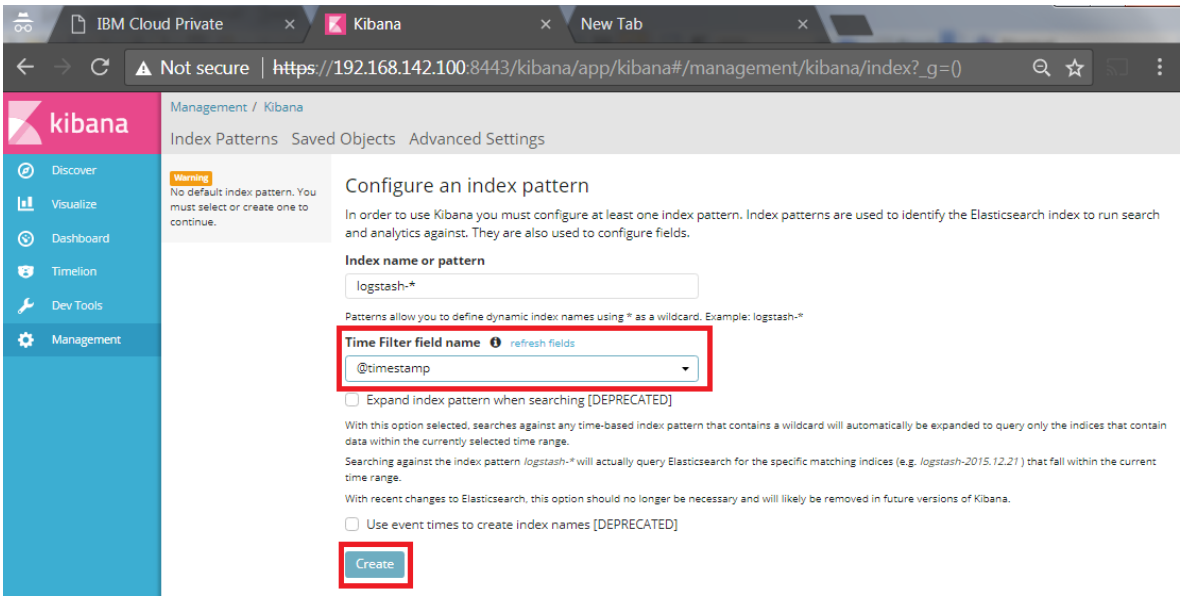
With ICP a you have out of the box an ELK installation to manage your application logs.



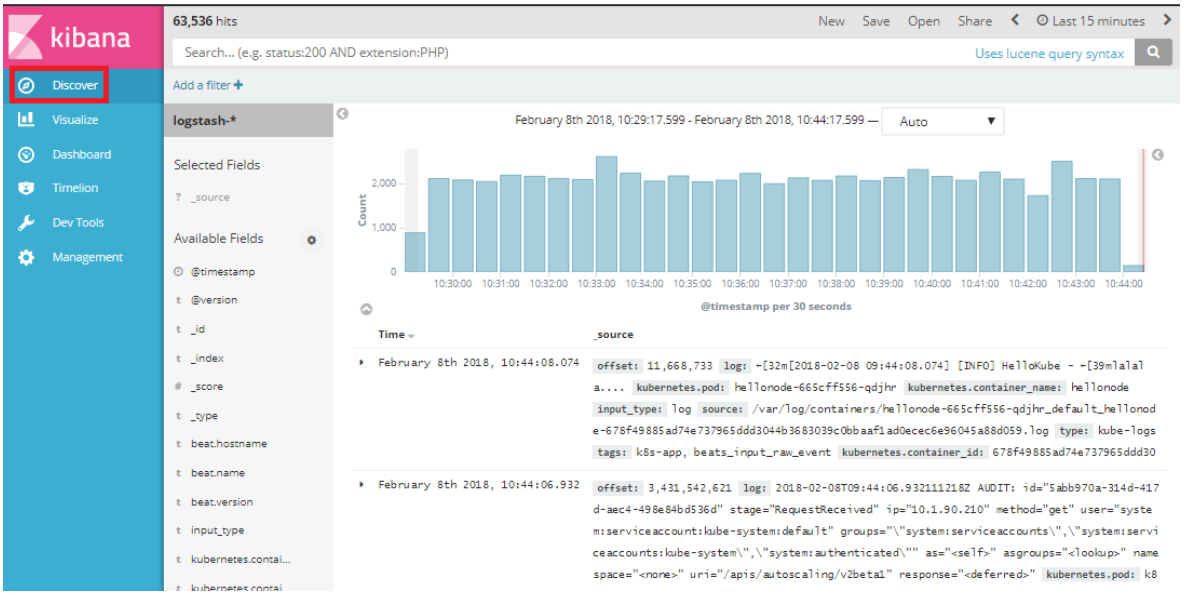
Open Kibana UI



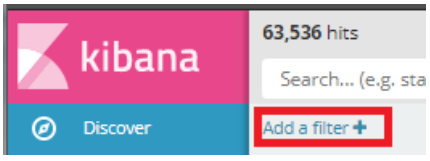
The Kibana console is open. The first action is to configure an index:

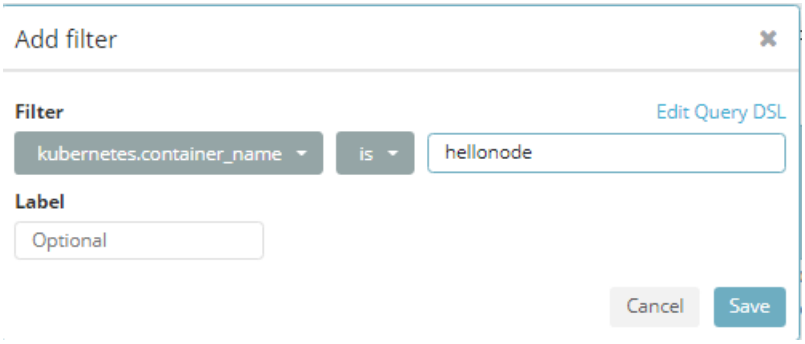


Click on “Discover” and you will see the logs of all the containers in the cluster

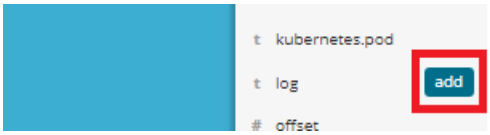


Create a filter

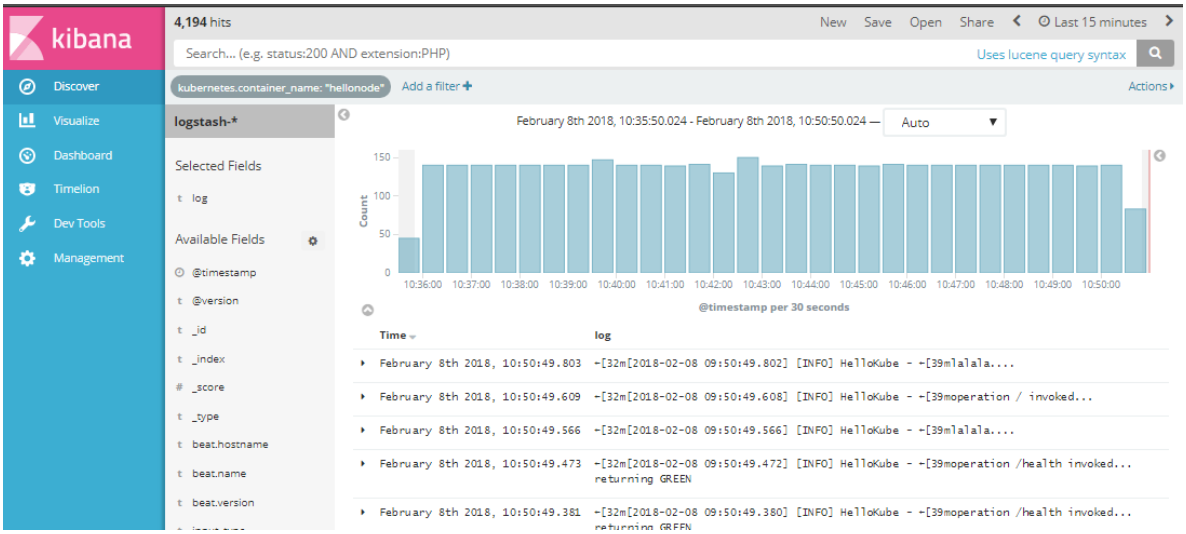




And click “Add” to the log label



Now you see all the logs of you application.



From the helm catalog in ICP you can deploy your own ELK instances with the custom scope and configuration needed in your infrastructure or projects

Lab 8. Monitoring

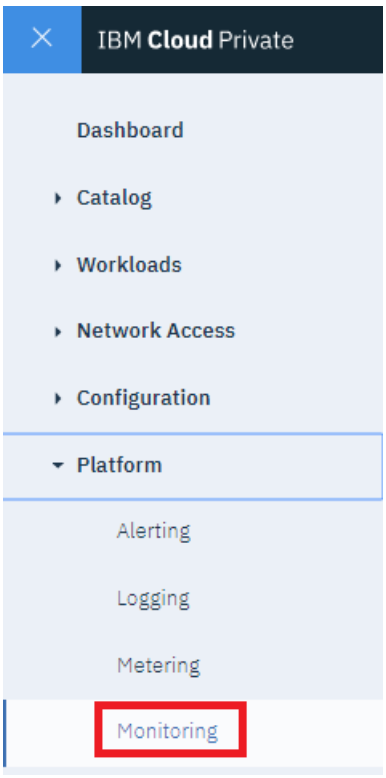
ICP uses out of the box Prometheus and Grafana to monitor the state of your Kubernetes cluster and the containers running on it

https://prometheus.io/

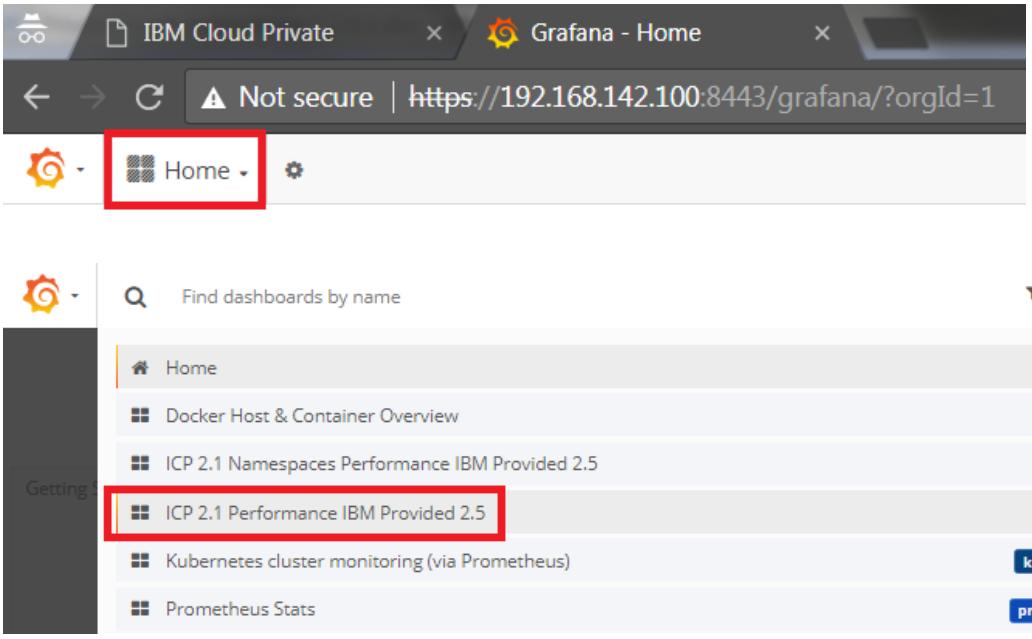
https://grafana.com/

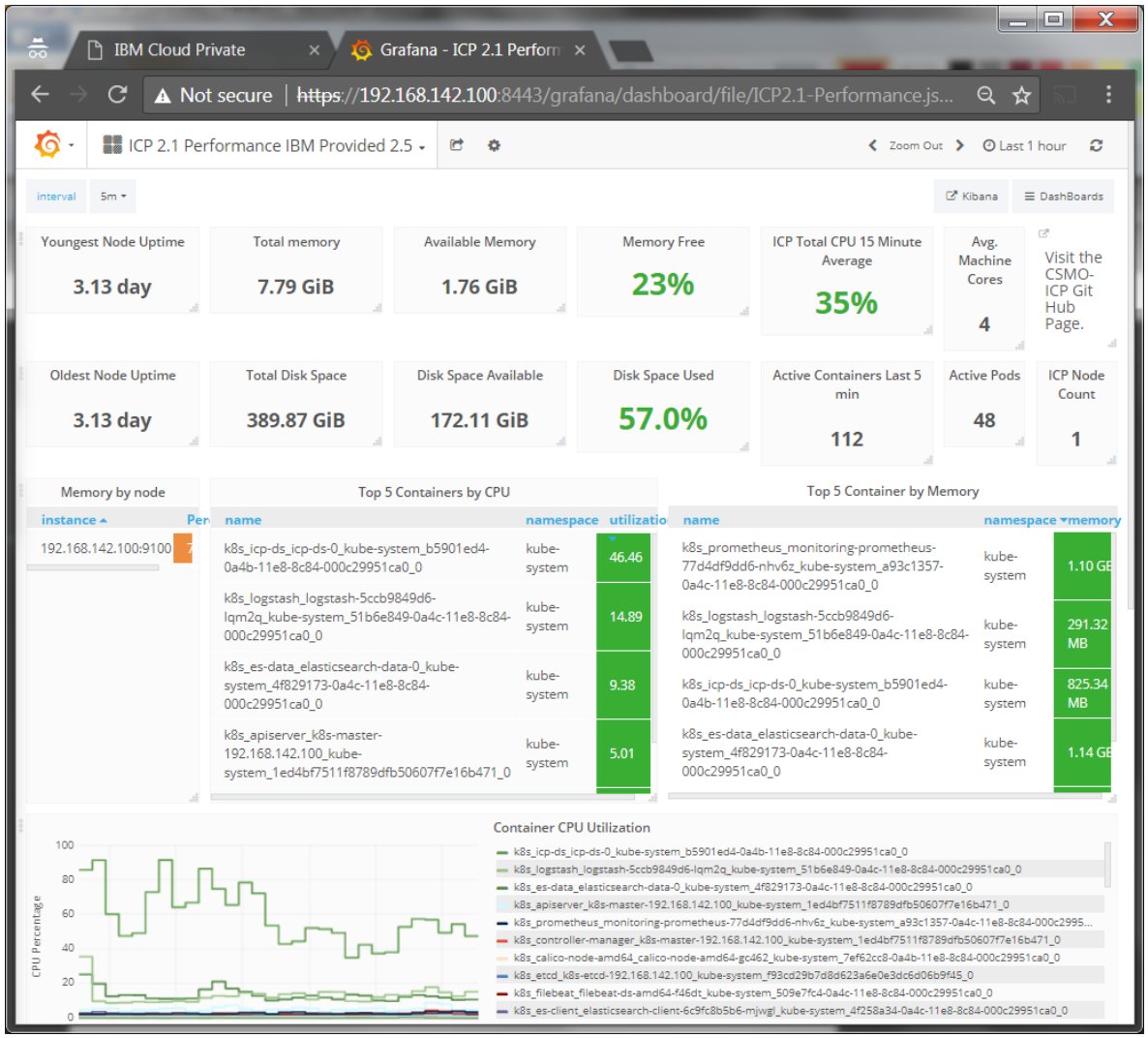
Basically you visualize with Grafana the metrics collected in Prometheus.

Go to Monitoring

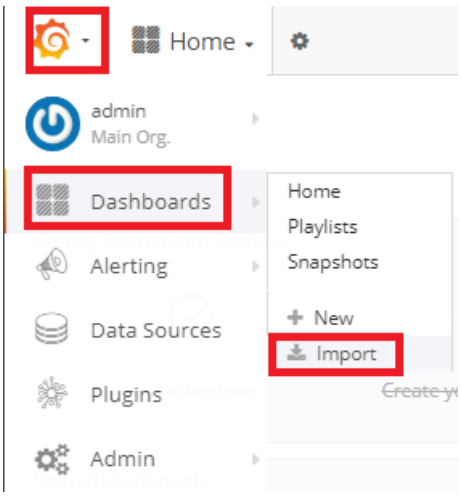


By default there are some dashboards created, access to “ICP 2.1 Performance” dashboard.

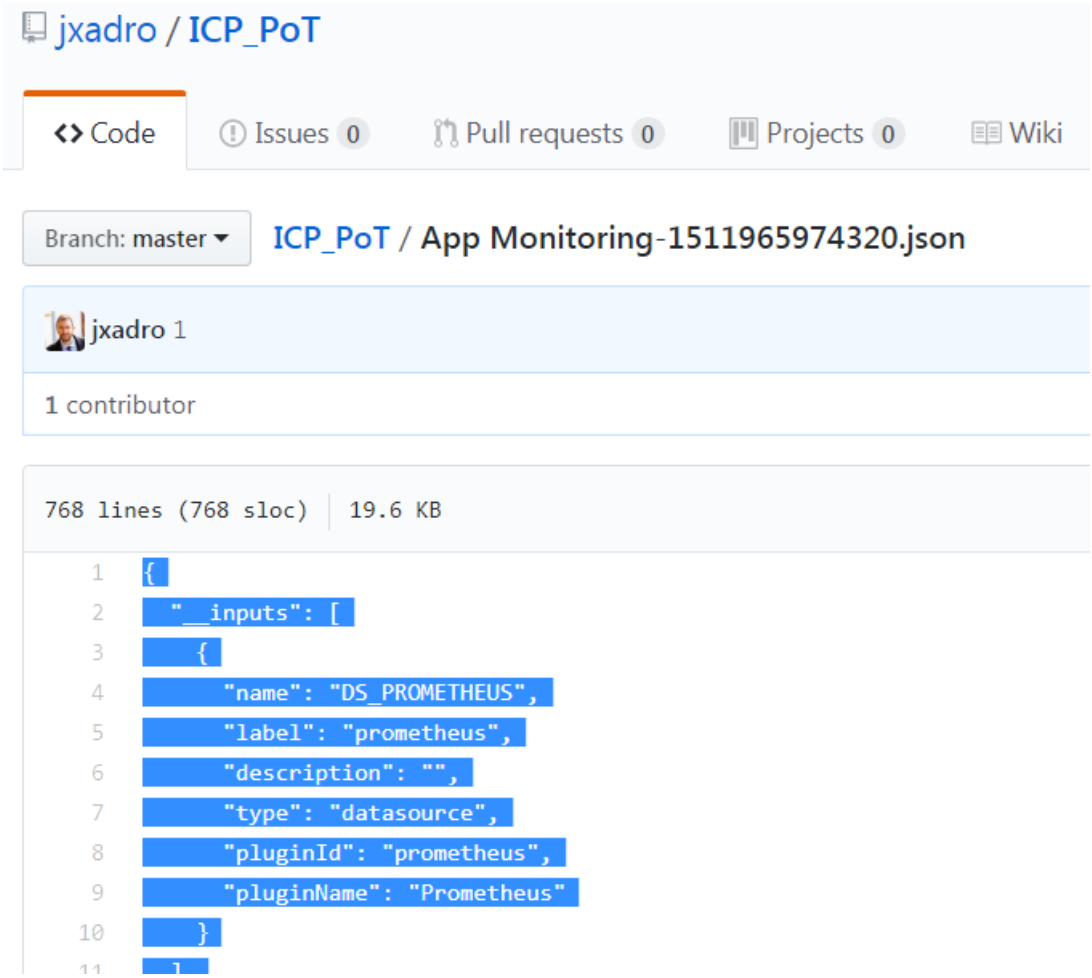




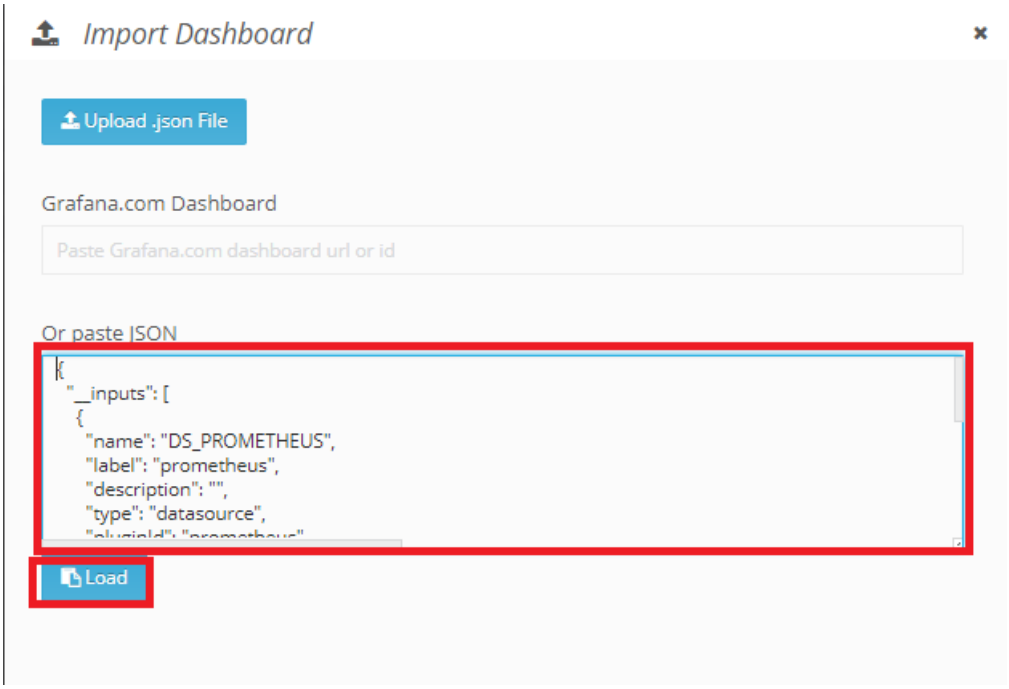
Import a new dashboard.

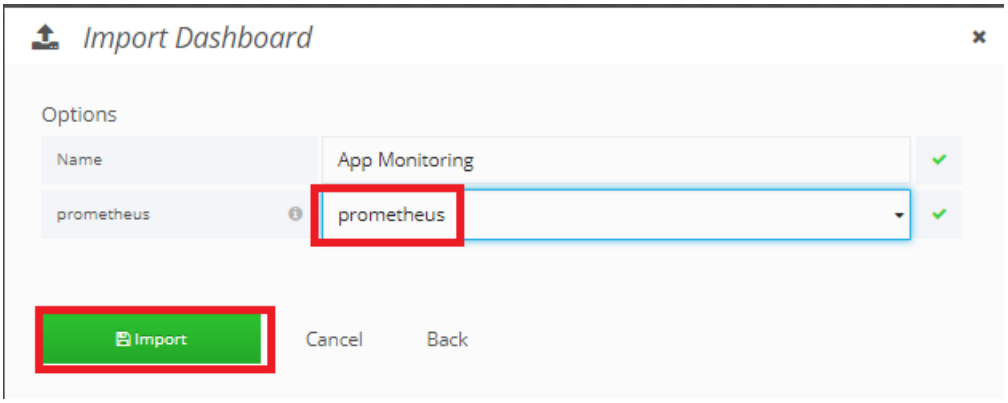


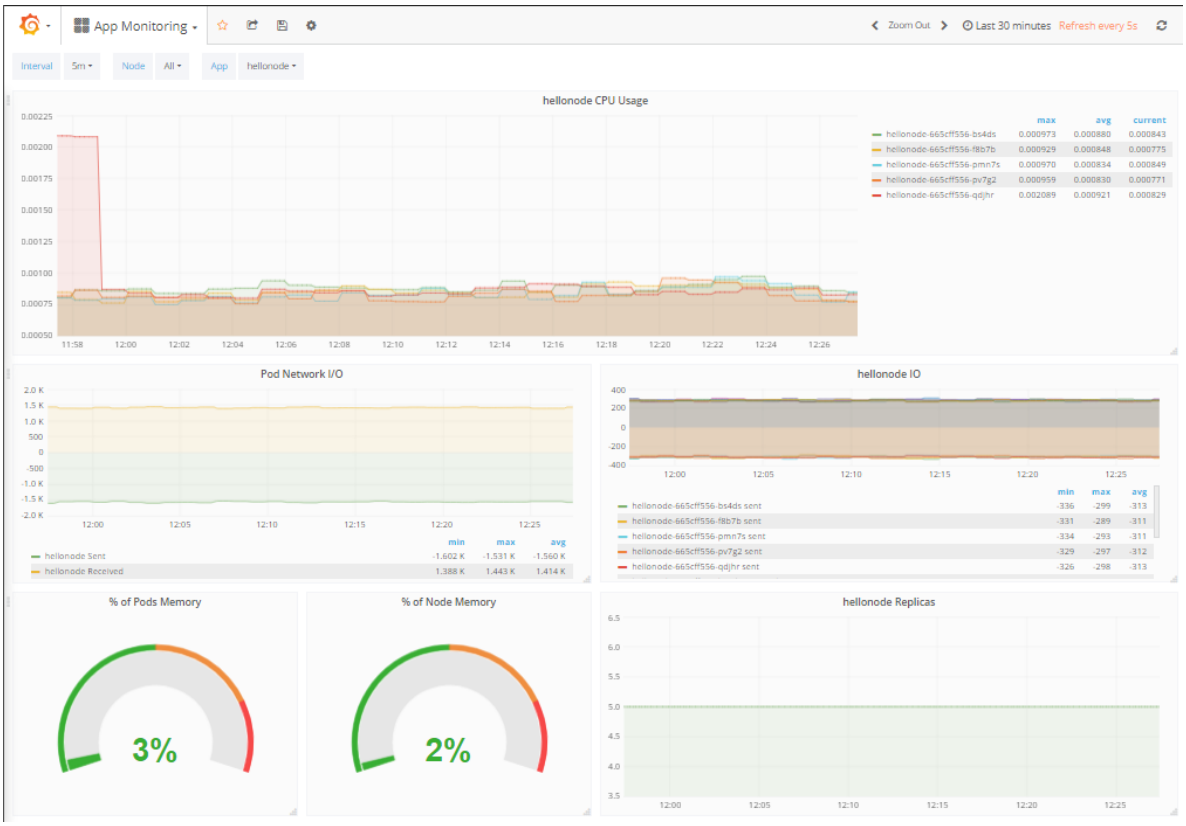
In a browser load the page https://github.com/jxadro/ICP_PoT/blob/master/App%20Monitoring-1511965974320.json and copy the json.



Paste the JSON in the dashboard import section


Import Dashboard
Upload .json File
Grafana.com Dashboard
Paste Grafana.com dashboard url or id
Or paste JSON
{
"__inputs": [
{
"name": "DS_PROMETHEUS",
"label": "prometheus",
"description": "",
"type": "datasource",
Load


Import Dashboard
Options
Name
App Monitoring
prometheus
prometheus
Import
Cancel
Back

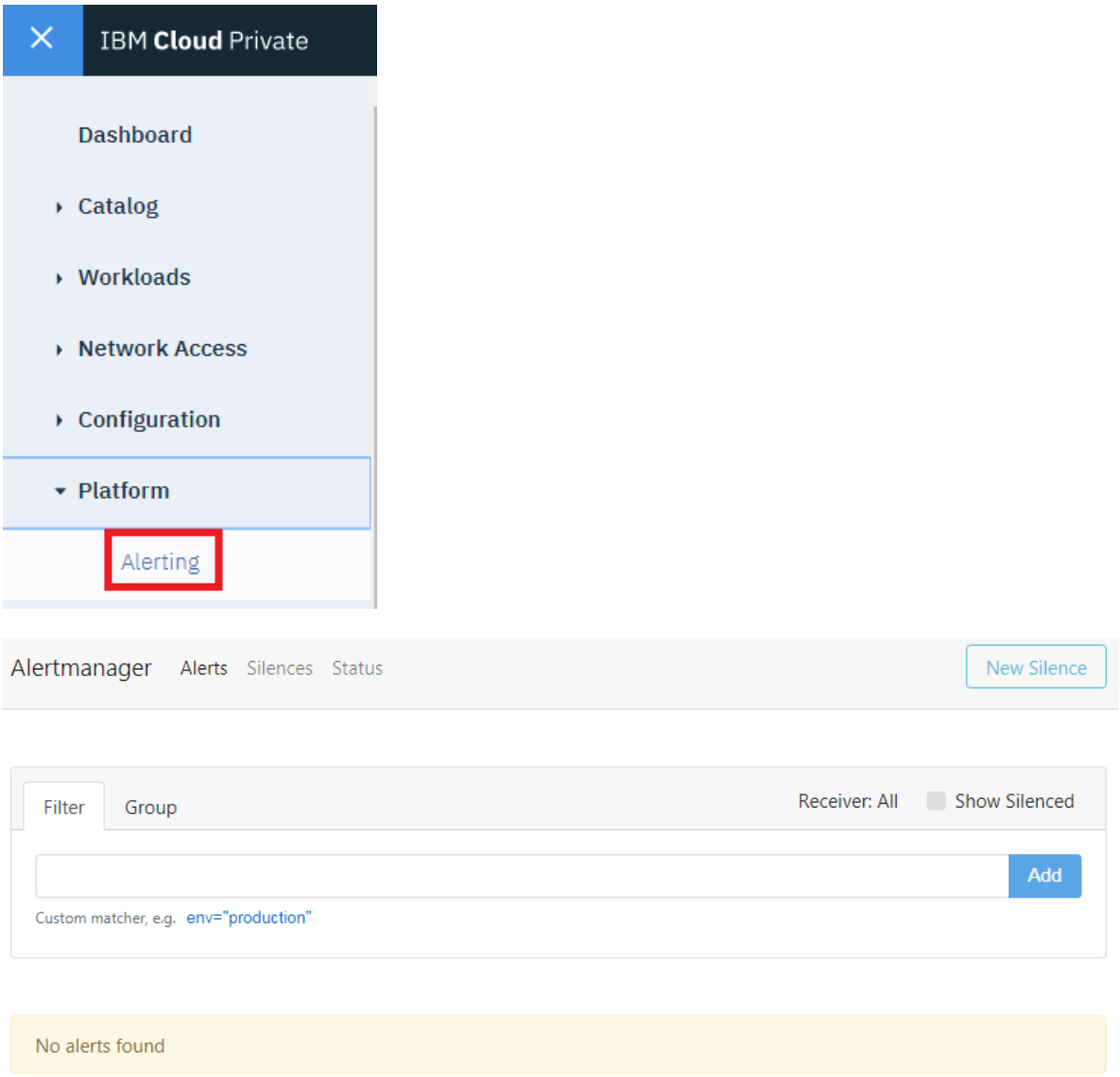


It is also possible to add datasources to Grafana additionally to the default “prometheus” data source.

Lab 9. Alerts

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_2.1.0/manage_metrics/monitoring_service.html

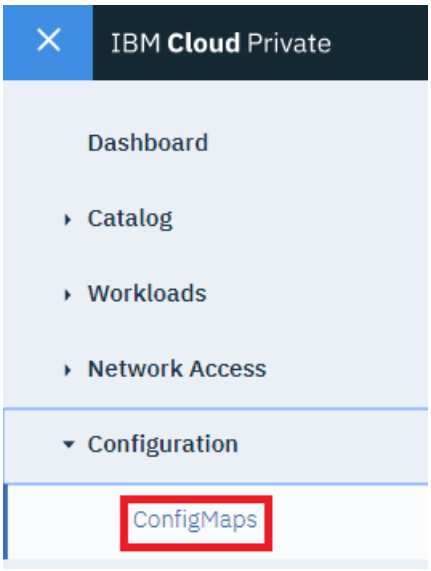
Access to the alerting console.

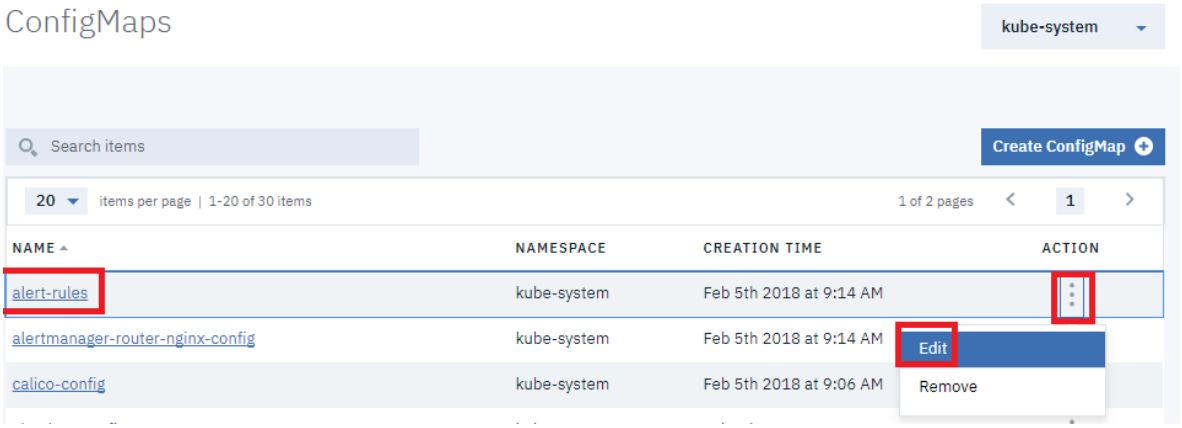


By default there are no alerts defined. You can define alerts using any of the metrics collected by Prometheus.

First we are going to create new alerts and then we will explore how to visualize the available metrics.

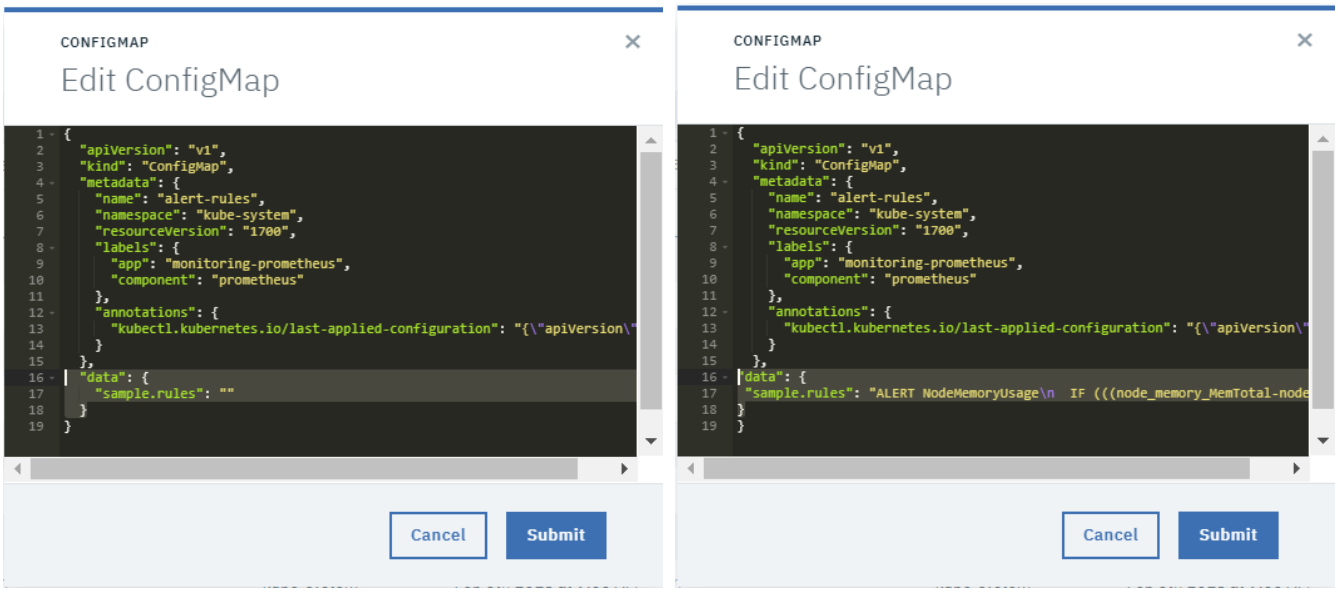
To create new rules we have do it through ConfigMap objects. Edit the rules ConfigMap.



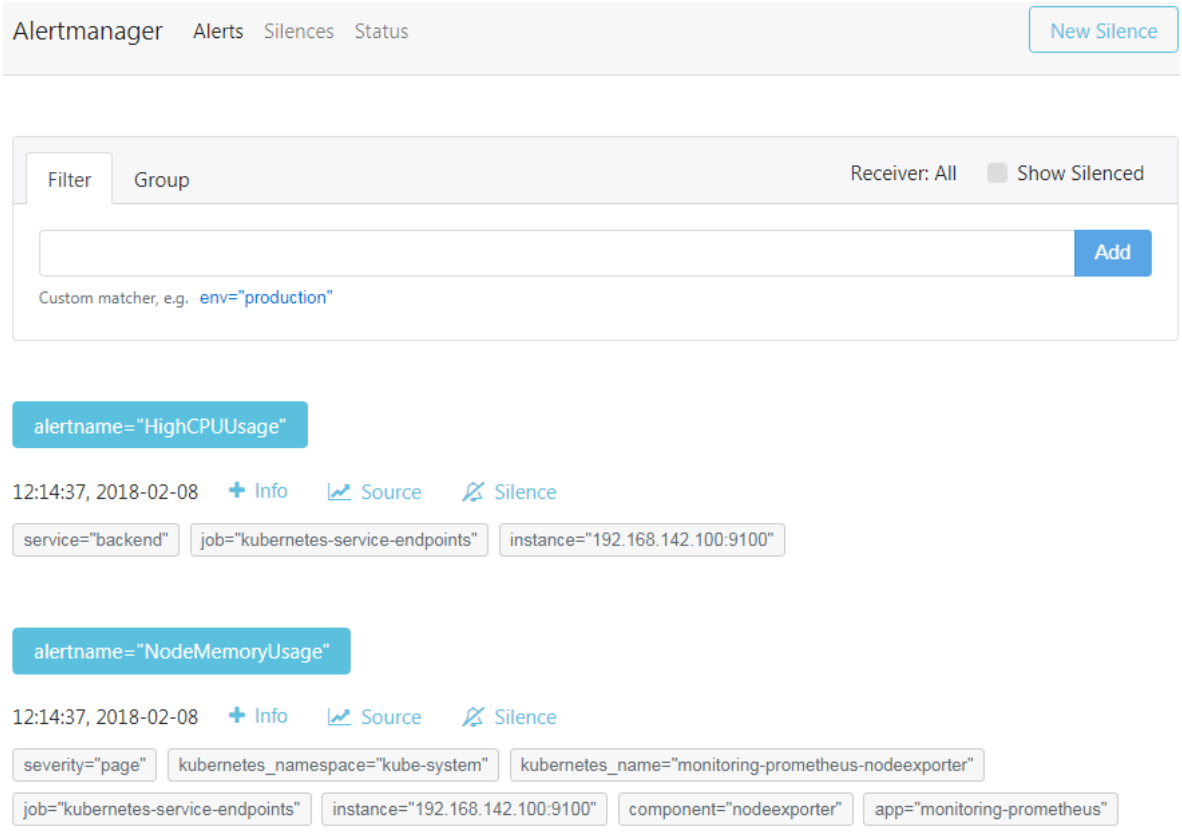


Add the following rules:

```
"data": {
 "sample.rules": "ALERT NodeMemoryUsage\n  IF (((node_memory_MemTotal-
node_memory_MemFree-node_memory_Cached)/(node_memory_MemTotal)*100)) >
25\n  FOR 1m\n  LABELS {\n    severity=\"page\"\n  }\n  ANNOTATIONS {\n
SUMMARY = \"{{$labels.instance}}: High memory usage detected\",\n
DESCRIPTION = \"{{$labels.instance}}: Memory usage is above 75% (current
value is: {{ $value }})\"\n  }\nALERT HighCPUUsage\n  IF
((sum(node_cpu{mode=~\"user|nice|system|irq|softirq|steal|idle|iowait\"})
by (instance, job)) - ( sum(node_cpu{mode=~\"idle|iowait\"}) by
(instance,job)))/(sum(node_cpu{mode=~\"user|nice|system|irq|softirq|steal
|idle|iowait\"}) by (instance, job)) * 100 > 2\n  FOR 1m\n  LABELS { \n
service = \"backend\" \n  }\n  ANNOTATIONS {\n    summary = \"High CPU
Usage\",\n    description = \"This computer has really high CPU usage for
over 10m\",\n  }"
}
```



Now give it some minutes and refresh the alerting console.



To view the metrics you have to access to Prometheus console. By default it is not exposed so we have to create a service to expose it.

Click on “Create resource” and create the following service:

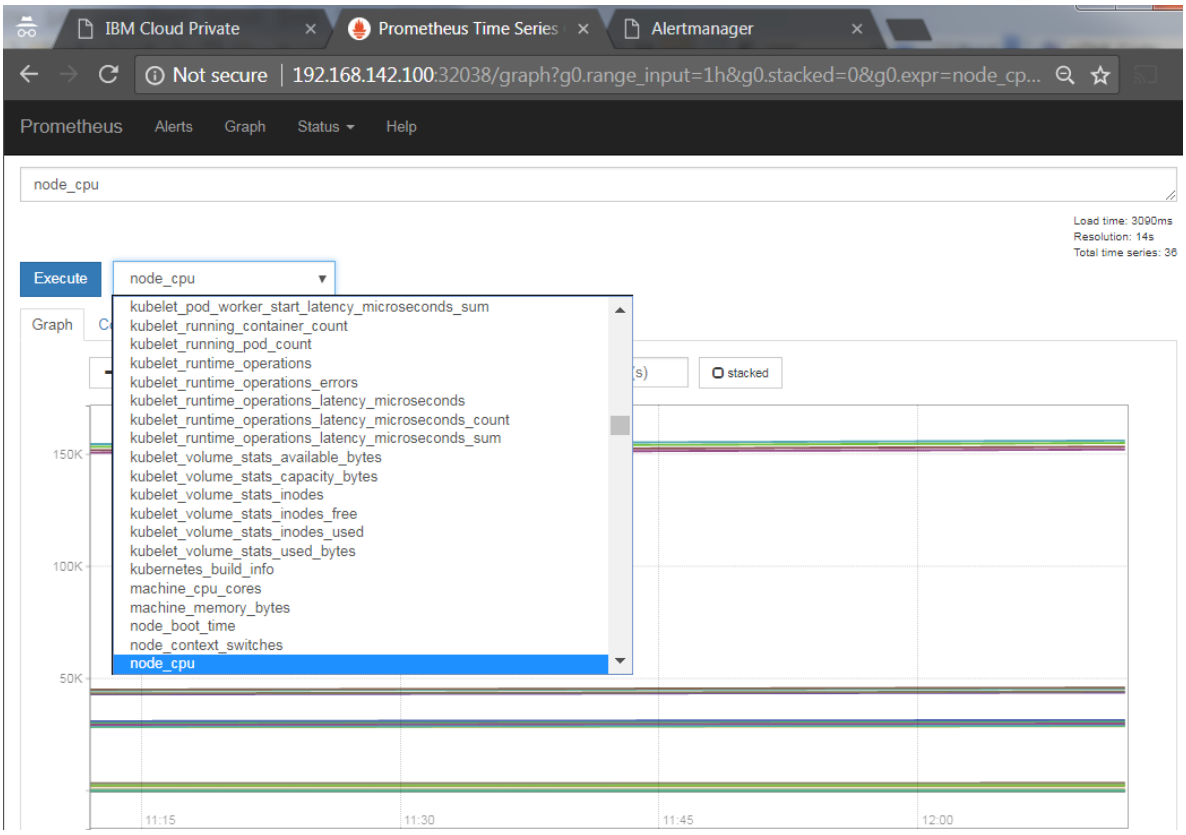

```
apiVersion: v1
kind: Service
metadata:
  name: monitoring-prometheus-nodeport
  namespace: kube-system
  labels:
    app: monitoring-prometheus-nodeport
    component: prometheus
spec:
  ports:
  - name: http
    protocol: TCP
    port: 9090
    targetPort: 9090
  selector:
    app: monitoring-prometheus
    component: prometheus
  type: NodePort
  sessionAffinity: None
```

Review the NodePort assigned:

kubectl get service monitoring-prometheus-nodeport -n kube-system

```
osboxes@osboxes:~$ kubectl get service monitoring-prometheus-nodeport -n kube-system
NAME                              TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
monitoring-prometheus-nodeport    NodePort   10.0.0.78     <none>        9090:32038/TCP   11m
```

Access to the Prometheus console:



Alerts can be pushed to different systems or generally to an HTTP Service.

https://prometheus.io/docs/alerting/configuration/

APPENDIX

Lab3 App.js

```
var express = require('express');
var log4js = require('log4js');
var http = require('http');
var app = express();
var logger = log4js.getLogger('HelloKube');
var isHealthy = true;

logger.info("Starting...");

setInterval(function(){
```

```
        if(isHealthy)
        {
                logger.info("lalala....");
        }
        else
        {
                logger.info("cough...");
        }
},3000);

app.get("/", function(req, res, next){
        logger.info("operation / invoked...");
        res.send("I'm alive!!!");
});

app.get("/env", function(req, res, next){
        res.json(process.env);
});

app.get("/health", function(req, res, next){
        if(isHealthy)
        {
                logger.info("operation /health invoked... returning GREEN");
                res.send("GREEN");
        }
        else
        {
                logger.info("operation /health invoked... returning RED");
                res.status(500).send("RED");
        }
});

app.get("/infect", function(req, res, next){
        logger.info("operation /infect invoked...");
        isHealthy = false;
        res.send("I don't feel that good...");
});

app.get("/kill", function(req, res, next){
        res.send("You are dead...");
        process.exit();
});
```

```
var port = process.env.PORT || 8080;

app.listen(port, function(){
        logger.info('HelloKube listening on port ' + port);
});
```

Lab 3. package.json

```
{
  "name": "HelloKube",
  "main": "app.js",
  "description": "Listeneint API Connect API Events",
  "version": "1.0.0",
  "private": false,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
          "express": "~4.2.0",
          "log4js": "~0.6.15"
  }
}
```