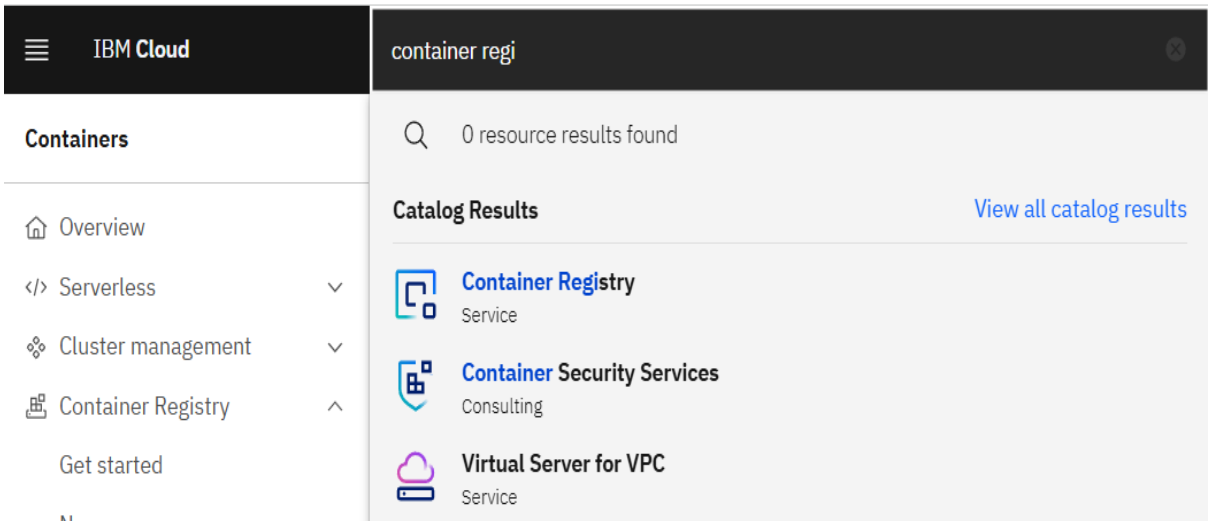
Deploying Dockerized Applications on IBM Cloud Kubernetes with Container Registry

Accessing IBM Cloud Container Registry:

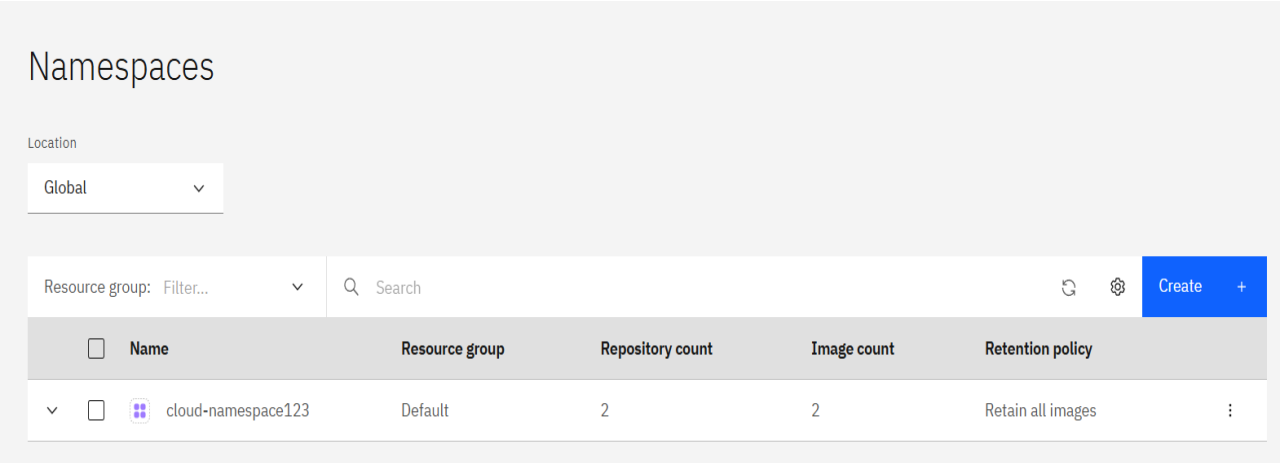
1. **Log in to IBM Cloud Dashboard**:
   - Visit the IBM Cloud Dashboard and log in with your IBM Cloud credentials.
2. **Navigate to Container Registry**:
   - In the **IBM Cloud Dashboard**, click on the **Menu** (three horizontal lines in the top left corner).
   - Under the **"Catalog"** section, search for **"Container Registry"**.
   - Click on **Container Registry** in the results.



Creating a Namespace in IBM Cloud Container Registry:

1. **Go to "Namespaces"**:
   - In the **IBM Cloud Container Registry** dashboard, you'll see a tab called **"Namespaces"**. Click on it to manage your namespaces.
2. **Create a Namespace**:
   - Once in the **Namespaces** tab, click the **Create Namespace** button.

- Enter a **name** for your namespace (it should follow the rules mentioned earlier, e.g., lowercase letters, numbers, and hyphens).
- Click **Create** to create your namespace.



List of Example Valid Namespace Names:

- my-container
- dev-apps
- project-registry
- cloud-namespace01
- web-images

Once your namespace is created, you can start using it to organize your container images within the IBM Cloud Container Registry.

Step 1: Log in to IBM Cloud

1. **Log in to IBM Cloud using your API key:**

If you haven't logged in yet, use the following command. Replace <your-api-key> with your actual IBM Cloud API key.

ibmcloud login --apikey <your-api-key>

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>ibmcloud login --apikey tYKOn6HumQZj-mbctiz_1hbSbdwbCpKEHBoryo8pEKwj
API endpoint: https://cloud.ibm.com
Region: us-south
Authenticating...
OK

Targeted account Santhanam Account (a1e6dcc5c5e741999deea2a9487b9b4c)


API endpoint:     https://cloud.ibm.com
Region:           us-south
User:             santhanaml2605@gmail.com
Account:          Santhanam Account (a1e6dcc5c5e741999deea2a9487b9b4c)
Resource group:   No resource group targeted, use 'ibmcloud target -g RESOURCE_GROUP'
```

This command will authenticate you and log you into your IBM Cloud account.

Step 2: Log in to IBM Cloud Container Registry

1. **Authenticate Docker to IBM Cloud Container Registry:**

Log in to your IBM Cloud Container Registry (icr.io) using the following command:

ibmcloud cr login

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>ibmcloud cr login
Logging 'docker' in to 'icr.io'...
Logged in to 'icr.io'.

OK
```

This will log you into the IBM Cloud Container Registry so you can push and pull container images.

Ensure the JAR file is built first: The Dockerfile is attempting to copy the JAR file from the target directory. This implies that the JAR file should already exist in that directory before you run the docker build command.

If you haven't built the JAR file yet, you'll need to run Maven to build the project first. In your front-service directory, run:

mvn clean package

This will compile your project and create the JAR file in the target directory.

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\front-service>mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< com.ibm.developer:front-service >-------------------
[INFO] Building front-service 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.1.0:clean (default-clean) @ front-service ---
[INFO]
[INFO] --- resources:3.2.0:resources (default-resources) @ front-service ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO] The encoding used to copy filtered properties files have not been set. This means that the same encoding will be used to copy filtered properties fil
es as when copying other filtered resources. This might not be what you want! Run your build with --debug to see which files might be affected. Read more at
 https://maven.apache.org/plugins/maven-resources-plugin/examples/filtering-properties-files.html
[INFO]
[INFO] --- compiler:3.8.1:compile (default-compile) @ front-service ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 3 source files to C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\front-service\target\classes
[INFO]
[INFO] --- resources:3.2.0:testResources (default-testResources) @ front-service ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\front-service\src\test\resources
[INFO]
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ front-service ---
[INFO] No sources to compile
[INFO]
[INFO] --- surefire:2.22.2:test (default-test) @ front-service ---
[INFO] No tests to run.
[INFO]
[INFO] --- jar:3.2.0:jar (default-jar) @ front-service ---
[INFO] Building jar: C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\front-service\target\front-service-0.0.1-SNAPSHOT.jar
[INFO]
```

Step 3: Build Docker Images for Frontend and Backend Services

1. **Navigate to your frontend service directory**:

cd path/to/front-service

2. **Build the frontend Docker image**:

docker build -t icr.io/cloud-namespace123/front-service:latest .

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\front-service>docker build -t icr.io/cloud-namespace123/front-service:latest .
[+] Building 17.1s (13/13) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                   0.1s
 => => transferring dockerfile: 563B                                                                                                   0.0s
 => [internal] load metadata for docker.io/library/adoptopenjdk:11-jre-hotspot                                                        1.0s
 => [internal] load .dockerignore                                                                                                      0.0s
 => => transferring context: 2B                                                                                                        0.0s
 => [builder 1/4] FROM docker.io/library/adoptopenjdk:11-jre-hotspot@sha256:ad6431b2e2a6f8016aa6d79c3f588783af9fdc06cafe131fd0d3faf560914b13  0.0s
 => [internal] load build context                                                                                                      1.7s
 => => transferring context: 18.89MB                                                                                                   1.7s
 => CACHED [builder 2/4] WORKDIR application                                                                                           0.0s
 => [builder 3/4] COPY target/*.jar application.jar                                                                                    1.9s
 => [builder 4/4] RUN java -Djarmode=layertools -jar application.jar extract                                                          11.9s
 => CACHED [stage-1 3/6] COPY --from=builder application/dependencies/ ./                                                               0.0s
 => CACHED [stage-1 4/6] COPY --from=builder application/snapshot-dependencies/ ./                                                      0.0s
 => CACHED [stage-1 5/6] COPY --from=builder application/spring-boot-loader/ ./                                                         0.0s
 => CACHED [stage-1 6/6] COPY --from=builder application/application/ ./                                                                0.0s
 => exporting to image                                                                                                                 0.1s
 => => exporting layers                                                                                                                0.0s
 => => writing image sha256:fe15c3d34380789050fb05a56d91fc61d5b943ac541d541822e0c5cab6f3f602                                           0.0s
 => => naming to icr.io/cloud-namespace123/front-service:latest                                                                        0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/25ubjdwuzyqhfnsz7746wl7nf

 2 warnings found (use docker --debug to expand):
 - WorkdirRelativePath: Relative workdir "application" can have unexpected results if the base image changes (line 2)
 - WorkdirRelativePath: Relative workdir "application" can have unexpected results if the base image changes (line 8)

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

3. **Navigate to your backend service directory**:

cd path/to/backend-service

4. **Build the backend Docker image**:

docker build -t icr.io/cloud-namespace123/backend-service:latest .

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\backend-service>docker build -t icr.io/cloud-namespace123/backend-service:latest .
[+] Building 7.8s (13/13) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                   0.0s
 => => transferring dockerfile: 563B                                                                                                   0.0s
 => [internal] load metadata for docker.io/library/adoptopenjdk:11-jre-hotspot                                                        1.0s
 => [internal] load .dockerignore                                                                                                      0.0s
 => => transferring context: 2B                                                                                                        0.0s
 => [builder 1/4] FROM docker.io/library/adoptopenjdk:11-jre-hotspot@sha256:ad6431b2e2a6f8016aa6d79c3f588783af9fdc06cafe131fd0d3faf560914b13  0.0s
 => [internal] load build context                                                                                                      1.1s
 => => transferring context: 18.89MB                                                                                                   1.1s
 => CACHED [builder 2/4] WORKDIR application                                                                                           0.0s
 => [builder 3/4] COPY target/*.jar application.jar                                                                                    0.6s
 => [builder 4/4] RUN java -Djarmode=layertools -jar application.jar extract                                                           4.7s
 => CACHED [stage-1 3/6] COPY --from=builder application/dependencies/ ./                                                               0.0s
 => CACHED [stage-1 4/6] COPY --from=builder application/snapshot-dependencies/ ./                                                      0.0s
 => CACHED [stage-1 5/6] COPY --from=builder application/spring-boot-loader/ ./                                                         0.0s
 => CACHED [stage-1 6/6] COPY --from=builder application/application/ ./                                                                0.0s
 => exporting to image                                                                                                                 0.0s
 => => exporting layers                                                                                                                0.0s
 => => writing image sha256:888a438eaa539af6aae8e85b89aecd315503997a86b125f45d6810d1e17d8772                                           0.0s
 => => naming to icr.io/cloud-namespace123/backend-service:latest                                                                      0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/mjq2njyh953sritcc8yzy5l5i
```

Step 4: Push Docker Images to IBM Cloud Container Registry

1. **Push the frontend image to IBM Cloud Container Registry**:

docker push icr.io/cloud-namespace123/front-service:latest

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\backend-service>docker push icr.io/cloud-namespace123/front-service
Using default tag: latest
The push refers to repository [icr.io/cloud-namespace123/front-service]
6797061f92a2: Pushed
6e229868a3f3: Pushed
bff96a67d2a7: Pushed
adf78f614550: Pushed
51f4fd87c5a9: Pushed
83b767b06655: Pushed
14fbd8039ba4: Pushed
da55b45d310b: Pushed
latest: digest: sha256:9b5289d3bf9b8f4e5fb067e34886d00fbc776957d47f4e0a879a602875f91059 size: 1996

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\backend-service>ibmcloud cr image-list
Listing images...

Repository                              Tag      Digest         Namespace            Created         Size     Security status
icr.io/cloud-namespace123/front-service  latest   9b5289d3bf9b   cloud-namespace123   4 minutes ago   105 MB   -

OK
```

2. **Push the backend image to IBM Cloud Container Registry**:

docker push icr.io/cloud-namespace123/backend-service:latest

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\backend-service>docker tag backend-service icr.io/cloud
-namespace123/backend-service:latest

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\backend-service>docker push icr.io/cloud-namespace123/backend-service
Using default tag: latest
The push refers to repository [icr.io/cloud-namespace123/backend-service]
5bc8f3426b96: Pushed
6e229868a3f3: Mounted from cloud-namespace123/front-service
bff96a67d2a7: Mounted from cloud-namespace123/front-service
adf78f614550: Mounted from cloud-namespace123/front-service
51f4fd87c5a9: Mounted from cloud-namespace123/front-service
83b767b06655: Mounted from cloud-namespace123/front-service
14fbd8039ba4: Mounted from cloud-namespace123/front-service
da55b45d310b: Mounted from cloud-namespace123/front-service
latest: digest: sha256:556f7ae364b3b179c29ba9cd79f26142920fed2bc4999000b3f6f6d44ec00d70 size: 1996

C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s\backend-service>cd ..
```

Step 5: Deploy Your Application Using Kubernetes

Check if the image pull secret is configured correctly: If your registry is private, you need to ensure that the Kubernetes cluster has access to the IBM Cloud Container Registry using a pull secret.

- You can create a secret using the following command:

kubectl create secret docker-registry ibm-cloud-secret --docker-server=icr.io --docker-username=iamapikey --docker-password=tYKOn6HumQZj-mbctiz_1hbSbdwbCpKEHBoryo8pEKwj --docker-email=divya.adroittech@gmail.com

In this command, you need to replace the following:

1. **ibm-cloud-secret**: Replace with a name for your secret (e.g., my-k8s-secret).
2. **<your-api-key>**: Replace with your actual IBM Cloud API key.
3. **your-email@example.com**: Replace with your IBM Cloud account email.

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>kubectl create secret docker-registry ibm-cloud-secret --docker-server=icr.io --docker-user
name=iamapikey --docker-password=tYKOn6HumQZj-mbctiz_1hbSbdwbCpKEHBoryo8pEKwj --docker-email=divya.adroittech@gmail.com
secret/ibm-cloud-secret created
```

1. **Apply the deployment.yaml file to your Kubernetes cluster**:

Update your deployment.yaml should reference the correct image paths, like this:

- For the frontend service:

  image: icr.io/cloud-namespace123/front-service:latest

- For the backend service:

  image: icr.io/cloud-namespace123/backend-service:latest

Final deployment.yaml file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-service-deployment
  labels:
    app: front-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: front-service
  template:
    metadata:
      labels:
        app: front-service
    spec:
      imagePullSecrets:
        - name: ibm-cloud-secret  # Image pull secret added here
      containers:
      - name: front-service
        image: icr.io/cloud-namespace123/front-service:latest
        imagePullPolicy: Always
        ports:
```

```
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /actuator/health/liveness
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 2
          failureThreshold: 1
        readinessProbe:
          httpGet:
            path: /actuator/health/readiness
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 2
          failureThreshold: 1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-service-deployment
  labels:
    app: backend-service
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: backend-service
  template:
    metadata:
      labels:
        app: backend-service
    spec:
      imagePullSecrets:
        - name: ibm-cloud-secret  # Image pull secret added here
      containers:
      - name: backend-service
        image: icr.io/cloud-namespace123/backend-service:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
        env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: front-service-port
spec:
  type: NodePort
  selector:
    app: front-service
  ports:
    - port: 8080
      name: http
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service-port
spec:
  sessionAffinity: None
  selector:
    app: backend-service
  ports:
    - port: 8080
      targetPort: 8080
      name: http
```

Now, apply the deployment.yaml file to your Kubernetes cluster:

kubectl apply -f path/to/deployment.yaml

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>kubectl apply -f deployment.yaml
deployment.apps/front-service-deployment unchanged
deployment.apps/backend-service-deployment unchanged
service/front-service-port unchanged
service/backend-service-port unchanged
```

This will deploy both the frontend and backend services on your Kubernetes cluster.

Step 6: Verify the Deployment

1. **Check the status of your deployments**:

kubectl get deployments

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>kubectl get deployments
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
backend-service-deployment  1/1     1            1           103m
divya                       1/1     1            1           5d21h
front-service-deployment    1/1     1            1           103m
hel                         1/1     1            1           5d22h
hello1                      1/1     1            1           5d22h
nodeapp-deployment          1/1     1            1           4d22h
```

This will show you the deployments for both frontend and backend services.

2. **Check the status of your pods**:

kubectl get pods

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>kubectl get pods
NAME                                      READY   STATUS    RESTARTS      AGE
backend-service-deployment-76f8997766-6lxhq  1/1     Running   0             29s
divya-78fd96c656-cs9wn                    1/1     Running   3 (91m ago)   5d21h
front-service-deployment-f77fd767-qpb89   0/1     Running   0             15s
hel-799f7ddf44-jnnzc                      1/1     Running   3 (91m ago)   5d22h
hello1-7d4877cbfd-g8w47                   1/1     Running   3 (91m ago)   5d22h
nodeapp-deployment-584987cc94-kg4n6       1/1     Running   2 (91m ago)   4d22h
```

This will show you the running pods for the services.

3. **Verify the services are running**:

kubectl get services

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>kubectl get services
NAME                 TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
backend-service-port ClusterIP     10.106.31.154   <none>        8080/TCP        91m
divya                NodePort      10.109.174.101  <none>        80:30096/TCP    5d21h
front-service-port   NodePort      10.104.224.102  <none>        8080:31618/TCP  91m
hello1               NodePort      10.99.176.176   <none>        80:31329/TCP    5d22h
kubernetes           ClusterIP     10.96.0.1       <none>        443/TCP         5d22h
nodeapp-service      LoadBalancer  10.108.98.106   <pending>     5000:31110/TCP  4d22h
```

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>kubectl get svc
NAME                 TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
backend-service-port ClusterIP     10.106.31.154   <none>        8080/TCP        92m
divya                NodePort      10.109.174.101  <none>        80:30096/TCP    5d21h
front-service-port   NodePort      10.104.224.102  <none>        8080:31618/TCP  92m
hello1               NodePort      10.99.176.176   <none>        80:31329/TCP    5d22h
kubernetes           ClusterIP     10.96.0.1       <none>        443/TCP         5d22h
nodeapp-service      LoadBalancer  10.108.98.106   <pending>     5000:31110/TCP  4d22h
```

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>minikube ip
192.168.49.2
```

Ensure that the front-service-port and backend-service-port are exposed correctly and are accessible.

Step 7: Access Your Services

1. **Get the external IP address or NodePort for your frontend service**:

If you're using a NodePort type service (as defined in your YAML), you can get the external port with:

kubectl get svc front-service-port

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>minikube service front-service-port
|-----------|--------------------|-------------|--------------------------|
| NAMESPACE |        NAME        | TARGET PORT |           URL            |
|-----------|--------------------|-------------|--------------------------|
| default   | front-service-port | http/8080   | http://192.168.49.2:31618 |
|-----------|--------------------|-------------|--------------------------|
* Starting tunnel for service front-service-port.
|-----------|--------------------|-------------|-----------------------|
| NAMESPACE |        NAME        | TARGET PORT |          URL          |
|-----------|--------------------|-------------|-----------------------|
| default   | front-service-port |             | http://127.0.0.1:61640 |
|-----------|--------------------|-------------|-----------------------|
* Opening service default/front-service-port in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

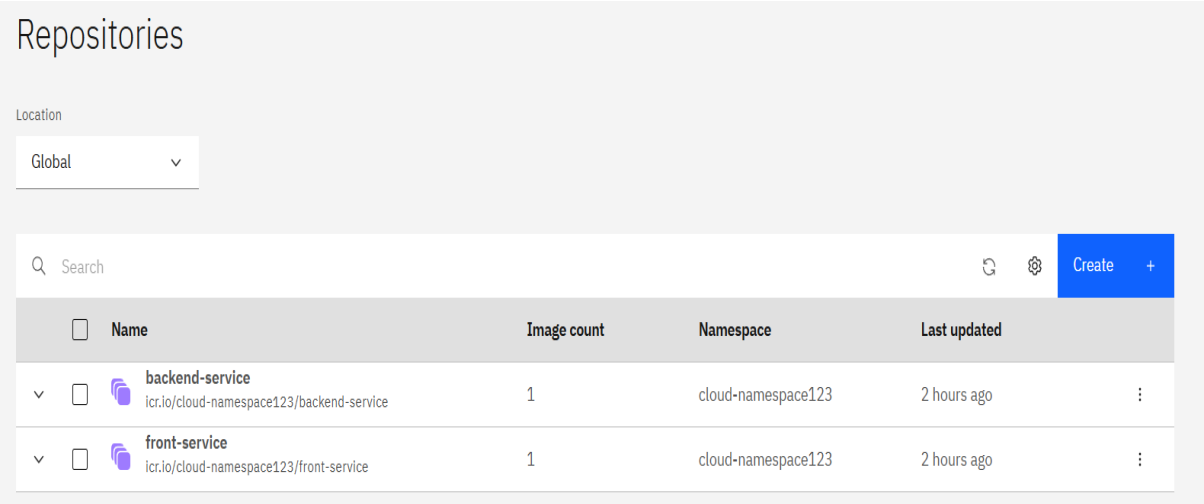
This will show the external port under NODE-PORT. You can access your frontend service using that port.

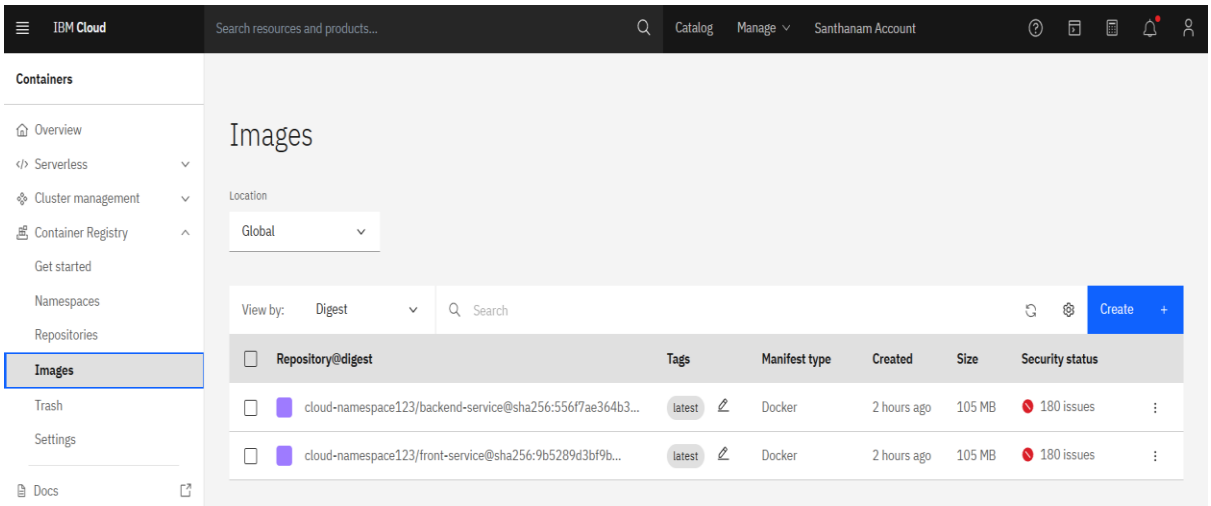
For backend-service:

kubectl get svc backend-service-port

```
C:\Users\ibmtr\Desktop\VTU DevOps\Week 9\managing-cn-apps-on-k8s>minikube service backend-service-port
|-----------|----------------------|-------------|--------------|
| NAMESPACE |         NAME         | TARGET PORT |     URL      |
|-----------|----------------------|-------------|--------------|
| default   | backend-service-port |             | No node port |
|-----------|----------------------|-------------|--------------|
* service default/backend-service-port has no node port
! Services [default/backend-service-port] have type "ClusterIP" not meant to be exposed, however for local development m
inikube allows you to access this !
* Starting tunnel for service backend-service-port.
|-----------|----------------------|-------------|------------------------|
| NAMESPACE |         NAME         | TARGET PORT |          URL           |
|-----------|----------------------|-------------|------------------------|
| default   | backend-service-port |             | http://127.0.0.1:61814 |
|-----------|----------------------|-------------|------------------------|
* Opening service default/backend-service-port in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Verify these changes in IBM CLOUD:





Summary of Commands:

- **Log in to IBM Cloud**: ibmcloud login --apikey <your-api-key>
- **Login to IBM Cloud Container Registry**: ibmcloud cr login
- **Build frontend Docker image**: docker build -t icr.io/cloud-namespace123/front-service:latest .
- **Build backend Docker image**: docker build -t icr.io/cloud-namespace123/backend-service:latest .
- **Push frontend Docker image**: docker push icr.io/cloud-namespace123/front-service:latest
- **Push backend Docker image**: docker push icr.io/cloud-namespace123/backend-service:latest
- **Configure Kubernetes cluster**: ibmcloud ks cluster config --cluster <cluster-name>
- **Deploy to Kubernetes**: kubectl apply -f deployment.yaml
- **Check deployments**: kubectl get deployments
- **Check services**: kubectl get services