Deploying a dotnet application to the IBM cloud

[Deploy a .net core application to openshift 2](#_Toc48897851)

[Deploying SQL Server to openshift 4](#_Toc48897852)

[Monitoring and Logging 8](#_Toc48897853)

[Appendix A 11](#_Toc48897854)

[Dockerfile for MSSQL 11](#_Toc48897855)

# Deploy a .net core application to openshift

Deploying a dotnet application to the IBM cloud is straight-forward. There are a few steps to follow, this guide will use Openshift for deployment purposes. This guide assumes that the dotnet project is on a SCM (git) repository. We will use <https://github.com/airazabal/hellosql.git> as the example.

1. Create an IBM account – instructions [here](https://cloud.ibm.com/docs/account?topic=account-account-getting-started).
2. an Openshift cluster must exist or be created. Instructions on creating a cluster on the IBM cloud can be found [here](https://cloud.ibm.com/docs/account?topic=account-manage_resource).
3. Create a new project ([openshift](https://docs.openshift.com/container-platform/4.2/applications/projects/working-with-projects.html))
4. Install CLI (oc – [openshift](https://cloud.ibm.com/kubernetes/clusters/bsi2ardd0aj09569m61g/access?platformType=openshift&region=us-south&resourceGroup=b67e9c8bfbef41b6a4c51c8384c112e8))
5. Install docker
6. In a command window type the following to clone the project and build the image
   1. git clone <https://github.com/airazabal/hellosql.git>
   2. cd hellosql
   3. docker build -t hellosql . (this will build an image with the Dockerfile in that directory)

Now we must login to the cluster by obtaining the token from the dropdown in the openshift console

A screenshot of a cell phone

Description automatically generated

Copy the login token and paste into the command window. Then switch to the previously created project or create a new one

1. oc new-project sentinel-dev or
2. oc project sentinel-dev

Then, create a deployable app by typing

1. oc new-app –docker-image=hellosql

This will create a deployment-config. Now we must tag the docker image (hellosql) and push it to the cloud image registry.

1. HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')

echo $HOST

default-openshift-image-registry.garage-dev-ocp44-vpc-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-east.containers.appdomain.cloud

if the HOST variable is empty, then just create the default route:

oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec":{"defaultRoute":true}}' --type=merge, then run the command above again.

1. docker tag hellosql $HOST/sentinel-dev/hellosql:latest

docker tag hellosql default-openshift-image-registry.garage-dev-ocp44-vpc-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-east.containers.appdomain.cloud/sentinel-dev/hellosql:latest

1. Docker push $HOST/sentinel-dev/hellosql:latest

Now that the image is in the image registry, we must give permission to the pod to run as “anyuid”. This will allow us to NOT run as root (since it is not permitted and root is the default if you don’t specify a user in the pod). If you look at the Dockerfile in the repo, there are some changes to the default Dockerfile as generated by Genexus:

ENV APP\_ROOT=/app

COPY ibm/20200730123835 ${APP\_ROOT}

RUN chmod -R u+x ${APP\_ROOT} && \

chgrp -R 0 ${APP\_ROOT} && \

chmod -R g=u ${APP\_ROOT} /etc/passwd

These changes are such so we can run as non-root and access the files.

1. oc adm policy add-scc-to-user anyuid system:serviceaccount:sentinel-dev:default

Now the image is on the cloud and we can build a new app.

1. oc new-app sentinel-dev/hellosql

--> Found image b712dee (44 hours old) in image stream "sentinel-dev/hellosql" under tag "latest" for "sentinel-dev/hellosql"

\* This image will be deployed in deployment config "hellosql"

\* The image does not expose any ports - if you want to load balance or send traffic to this component

you will need to create a service with 'oc expose dc/hellosql --port=[port]' later

\* WARNING: Image "sentinel-dev/hellosql:latest" runs as the 'root' user which may not be permitted by your cluster administrator

--> Creating resources ...

deploymentconfig.apps.openshift.io "hellosql" created

--> Success

Run 'oc status' to view your app.

oc get pods

NAME READY STATUS RESTARTS AGE

hellosql-1-7w9f6 1/1 Running 0 72m

hellosql-1-deploy 0/1 Completed 0 80m

server-1-deploy 0/1 Completed 0 35m

server-2-deploy 0/1 Completed 0 28m

server-2-xl7lh 1/1 Running 0 28m

oc expose pod/hellosql-1-7w9f6 --port 80

As you can see “hellosql-1-7w9f6” is “running”. You can see the pod’s logs by

1. oc logs hellosql-1-7w9f6

We need to expose the pod with a service and a route so that external clients can get to it. The easiest way to do that is to use the “oc expose” command:

1. oc expose pod/hellosql-1-7w9f6

The service can be seen by typing:

oc get svc

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

hellosql-1-7w9f6 ClusterIP 172.21.41.106 <none> 80/TCP 69m

Note that it shows a CLUSTER-IP, which is an internal IP – i.e., it cannot be accessed from the outside. So, in order to externally expose the pod, we must expose the service.

1. oc expose svc/hellosql-1-7w9f6

And we finally can see the external IP by typing:

1. oc get routes

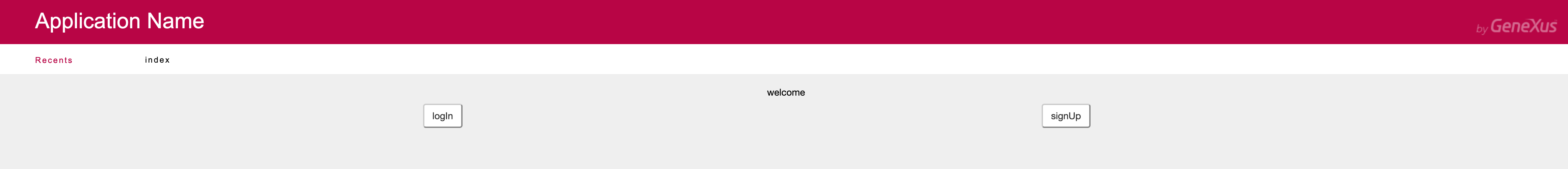
NAME HOST/PORT PATH SERVICES PORT TERMINATION WILDCARD

hellosql-1-7w9f6 hellosql-1-7w9f6-sentinel-dev.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud hellosql-1-7w9f6 80 None

As you can see the long HOST/PORT is the external DNS (IP) we want. To test the pod, you can go to a browser and type:

1. http:// hellosql-1-7w9f6-sentinel-dev.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud/index.aspx

You should see this:



So that wraps up the application, but we still need a DB to connect. The easiest way is to use microsoft’s default sql server images on their repo.

Note: we will use the convention “<service>.<pod\_namespace>.svc.cluster.local” to invoke the sql server pod from the hellosql pod internally. So, the name of the "Connection-Default-Datasource" should be “server.sentinel-dev.svc.cluster.local”. More on this later.

# Deploying SQL Server to openshift

Create a new app with microsoft’s sql server image:

1. oc new-app ACCEPT\_EULA=Y SA\_PASSWORD=MyPass@word MSSQL\_PID=Express mcr.microsoft.com/mssql/server:latest

We are passing the environment variables that the image requires to start up.

As before we can use “oc status” to see what’s being created.

In project sentinel-dev on server https://c106-e.us-south.containers.cloud.ibm.com:32160

http://hellosql-1-7w9f6-sentinel-dev.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud to pod port 80 (svc/hellosql-1-7w9f6)

rc/hellosql-1 runs image-registry.openshift-image-registry.svc:5000/sentinel-dev/hellosql@sha256:177fe23f71352f38169d9ce1014cca6f87abf509179add9ef3fde4f7e6bd3b61

rc/hellosql-1 created 2 hours ago - 1 pod

http://sqlserver-sentinel-dev.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud to pod port 1433-tcp (svc/server)

dc/server deploys istag/server:latest

deployment #2 deployed about an hour ago - 1 pod

deployment #1 deployed about an hour ago

dc/hellosql deploys istag/hellosql:latest

deployment #1 deployed 2 hours ago - 1 pod

7 infos identified, use 'oc status --suggest' to see details.

We now have 2 DeploymentConfigs – one each for each pod. A service was automatically created for the sqlserver (svc/server), because the port (1433) was exported in the image. However, this port exposed by the new-app command is exposed as http. We need a straight tcp port for connecting to the database, so we need to delete the service and recreated as follows:

oc get svc

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

server ClusterIP 172.21.21.242 <none> 1433/TCP 7s

Notice that the type of service is ClusterIP, we need a type of LoadBalancer, which exposes the port externally and does not restrict the protocol to http. Delete the old service first.

oc delete svc/server

Create a new svc by using “oc expose”:

oc get pods

NAME READY STATUS RESTARTS AGE

server-1-deploy 0/1 Completed 0 3m24s

server-1-swhwj 1/1 Running 0 3m22s

oc expose pod server-1-swhwj --port=1433 --type=LoadBalancer --name=sql-server-svc

oc get svc

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

sql-server-svc LoadBalancer 172.21.76.193 169.60.132.139 1433:31136/TCP 69s

Notice that we now have an external ip (169.60.132.139) which we can use directly.

Now we are going to ssh into the server pod and create the “ibm” database, and the “login” table with these scripts:

oc rsh server-1-swhwj bash

/opt/mssql-tools/bin/sqlcmd -U sa -P MyPass@word

>create database ibm;

>go

>use ibm;

>go

>CREATE TABLE [login] ( [id] DECIMAL(10) NOT NULL IDENTITY ( 1 , 1 ),[firstName] NVARCHAR(40) NOT NULL,[lastName] NVARCHAR(40) NOT NULL,[emailAdress] NVARCHAR(40) NOT NULL,[password] NVARCHAR(40) NOT NULL,PRIMARY KEY ( [id] ))

> go

We can use the oc rsh <pod> to ssh into the server pod.

1. oc get pods

NAME READY STATUS RESTARTS AGE

hellosql-1-7w9f6 1/1 Running 0 97m

hellosql-1-deploy 0/1 Completed 0 106m

server-1-deploy 0/1 Completed 0 60m

server-2-deploy 0/1 Completed 0 54m

server-2-xl7lh 1/1 Running 0 54m

1. oc rsh server-2-xl7lh bash

You now should get a “$” – meaning you are logged in to the pod. The mssql command is in “/opt/mssql-tools/bin/sqlcmd”. To run it you need to pass the server name, userid and password, like this:

1. /opt/mssql-tools/bin/sqlcmd -S 127.0.0.1 -U sa -P MyPass@word

You should get a “>1” prompt. You can enter the commands above followed by a “go” command to execute. Type exit twice to return to your command prompt and exit the pod.

1. exit
2. exit

We are ready to access the application again and “signup”. You should see this:

A screenshot of a cell phone

Description automatically generated

And entering a sample login, like this:

A screenshot of a cell phone

Description automatically generated

This confirms access to the ms sql database.

A screenshot of a cell phone

Description automatically generated

# Monitoring and Logging

Monitoring and logging can be enabled in the IBM Cloud Cluster console.

A screenshot of a cell phone

Description automatically generated

The logging will be shown in a separate window where logDNA service will show all the logs configured.

A screenshot of a cell phone

Description automatically generated

Likewise, for monitoring, sysdig will show a separate window:

A screen shot of a video game

Description automatically generated

These two tools can be used to monitor the cluster and all the applications from one place.

Additionally, you can go directly to the openshift console and navigate to the project and look up the pods.

A screenshot of a cell phone

Description automatically generated

And the corresponding logs:

A screenshot of a cell phone

Description automatically generated

You can also ssh (terminal) into the pod to look at local logs. This can be done by selecting the “Terminal” tab.

# Appendix A

## Dockerfile for MSSQL

Normally, in order to get an instance of mssql running on openshift, you can directly use “oc new-app mcr.microsoft.com/mssql/server:latest”. However if you need more control or need to restore a database for example, you can use a Dockerfile to have that level of control.

FROM mcr.microsoft.com/mssql/server:latest

ENV ACCEPT\_EULA Y

ENV MSSQL\_PID Express

ENV SA\_PASSWORD MyPass@word

COPY ./RegistroActividades.bak /opt/local/

COPY ./restoredb.sql /opt/local

USER root

RUN [ "chmod", "-R", "777", "/opt/local" ]

USER mssql

RUN sh -c "/opt/mssql/bin/sqlservr -m -T4022 -T3659 -q'Latin1\_General\_100\_CI\_AI' &" && sleep 30

USER root

RUN kill -9 1 && ps -ef

USER mssql

RUN sh -c "/opt/mssql/bin/sqlservr &" && sleep 30 && /opt/mssql-tools/bin/sqlcmd -U sa -P MyPass@word -i /opt/local/restoredb.sql

RUN ps -ef

ENTRYPOINT [ "/opt/mssql/bin/sqlservr" ]

In this Dockerfile, we load the same image we used in the “oc new-app” command, but we also setup a local directory (/opt/local), then COPY the database backup (RegistroActividades.bak) and then run the sqlservr command to start the database in single user mode (-m) and selecting a new collating mode (Latin1\_General\_100\_CI\_AI). Then we “kill” the existing ms server instance and run it again. This time we want to run the restore command via sqlcmd. The last statement (ENTRYPOINT) is there so when we start the image, it will start the database server.

In order to use this Dockerfile, we can do the following:

docker build -t sqlserver -f Dockerfile.mssql .

The above command assumes the Dockerfile is named “Dockerfile.mssql”. After it builds, then we can tag it and push it.

docker tag sqlserver:latest default-openshift-image-registry.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud/sentinel-dev/sqlserver:latest

and push (make sure you are logged in to docker first - docker login -u $(oc whoami) -p $(oc whoami -t) default-openshift-image-registry.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud)

docker push default-openshift-image-registry.sentinel-test-0143c5dd31acd8e030a1d6e0ab1380e3-0000.us-south.containers.appdomain.cloud/sentinel-dev/sqlserver:latest

finally, we can use the “new-app” command:

oc new-app sentinel-dev/sqlserver:latest