Debugging a NodeJS application while deployed on Kubernetes/Openshift

Introduction

As nodeJS becomes a relevant technology to develop & deploy microservices, it is necessary to find a way to debug such a service while deployed in a multi-containers environment.

Debugging on the dev's PC is easy for a unit service debug, but at integration time, it is often required to debug a deployed service itself.

This work is based on the work done here : https://hub.docker.com/r/glenschler/nodejs-inspector/ that covers how to debug a nodejs embedded in a docker container.

This paper is complementing this approach for debugging on a Kubernetes platform.

NodeJS versions

There are several versions of nodeJS in the nature, and you should first select the right version for your work. The node-inspector (debugger) tool (with the relevant version) will be included within the image you will build with your project in the next steps.

- nodejs-inspector/v5
- nodejs-inspector/v4
- nodejs-inspector/v0.12.LTS
- nodejs-inspector/v0.10.LTS

What to do on your project side ?

Let's suppose you have an application NodeJS, in the **./myapplication** directory.

This application is expected to have a **package.json** to include the dependancies (for npm)

cd ./myapplication

```
git clone https://github.com/jmlambert78/docker-debugnode
```

- This will copy a set of files used during the build of the debuggable docker image for your app.

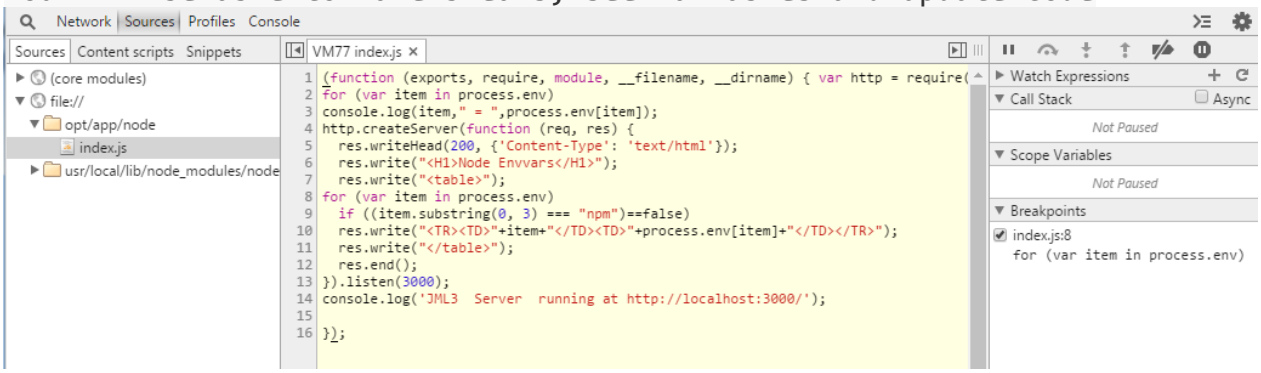
Use the appropriate command to match the node version from this list :

- docker build -t nodeapp-debug:5 --file=./docker-debugnode/debugapp/v5/Dockerfile .
- docker build -t nodeapp-debug:4 --file=./docker-debugnode/debugapp/v4/Dockerfile .
- **docker build -t nodeapp-debug:0.12 --file=./docker-debugnode/debugapp/v0.12.LTS/Dockerfile .**
- docker build -t nodeapp-debug:0.10 --file=./docker-debugnode/debugapp/v0.10.LTS/Dockerfile .

This will build a docker image for you to run

Local docker run option

- docker run --rm --name nodeapp-v0.12 -p 8080:8080 -p 5858:5858 -p 3000:3000 nodeapp-debug:0.12
- 3 ports are exposed/mapped.
    - 3000 is hier for the Node App itself
    - 8080 : nodedebug web interface
    - 5858 : node debugger interface.
- point your chrome browser to the debugger's
    - http://localhost:8080/?port=5858
- You will be able to make breaks, see variables and update code



- Access to your application web interface at the http://localhost:3000 if this port is the one.

Kubernetes deployment & debugging on the platform

To deploy a container image on a K8S platform, you need to use a YAML file to :

- Create a POD
- Create Services to reach the POD instances
- Create a Route to reach the Service (using a URL exposed outside of the platform)

Pod creation

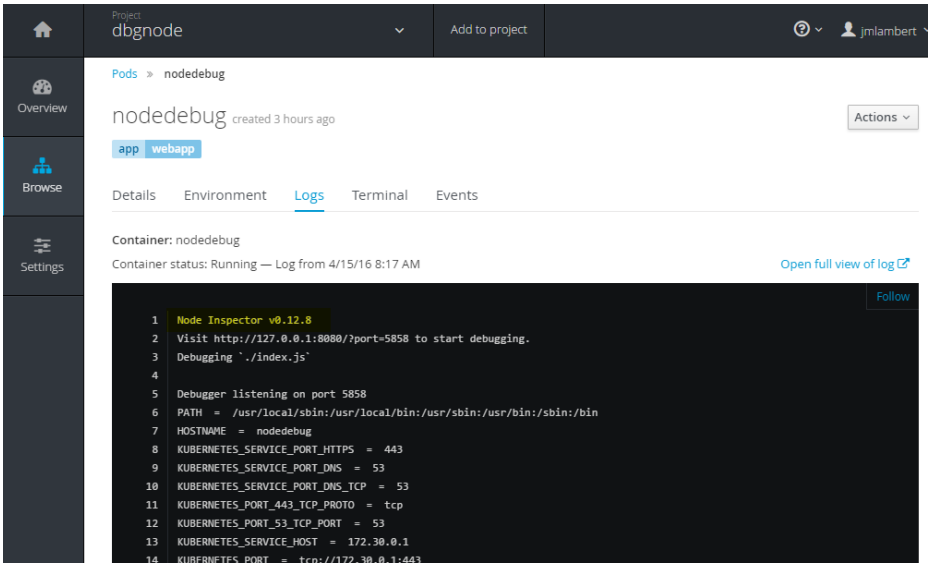
This image is to be deployed through a YAML file that will expose the different ports as below :

```
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "nodedebug",
    "labels": {
      "app": "webapp"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "nodedebug",
        "image": "nodeapp-debug:0.12",
        "ports": [
          {
            "name": "webapp",
            "containerPort": 3000
          },
          {
            "name": "dbgapp",
            "containerPort": 8080
          },
          {
            "name": "debug",
            "containerPort": 5858
          }
        ]
      }
    ]
  }
}
```

With this file in your area, you may deploy it with the command :

oc create –f pod.yaml

Then the Pod is created and you should see the running pod in the platform :



The default launched file is here **index.js**

Service creation

As you need to reach this pod via a service layer, you need to create a service object like the one here :

```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "nodedbg-svc"
    },
    "spec": {
        "selector": {
            "app": "webapp"
        },
        "ports": [
            {
                "name": "dbgapp",
                "protocol": "TCP",
                "port": 8080,
                "targetPort": 8080
            },
            {
                "name": "webapp",
                "protocol": "TCP",
                "port": 3000,
                "targetPort": 3000
            },
            {
                "name": "dbgport",
                "protocol": "TCP",
                "port": 5858,
                "targetPort": 5858
            }
        ]
    }
}
```

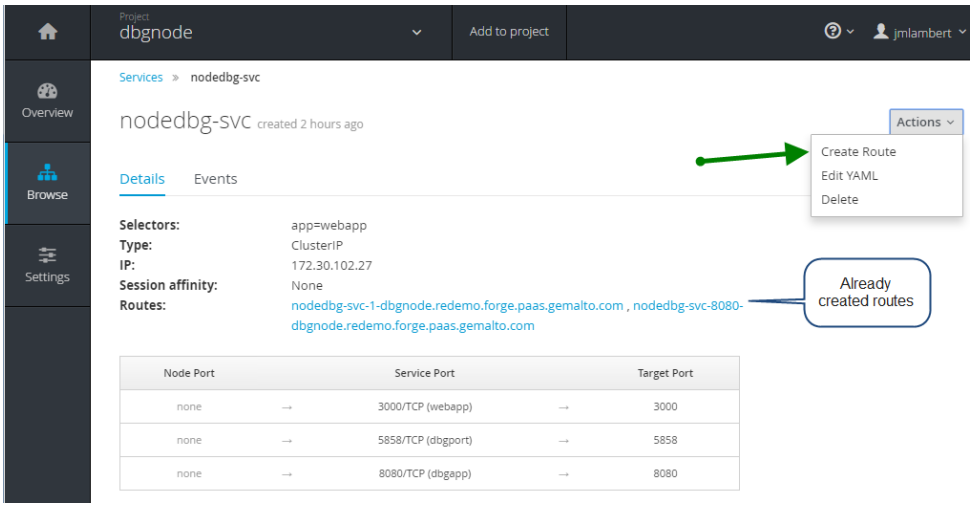
Here we map ports one to one with the Pods

The 2 useful services that you need to access are the webapp and the dbgapp.

Create the services using :

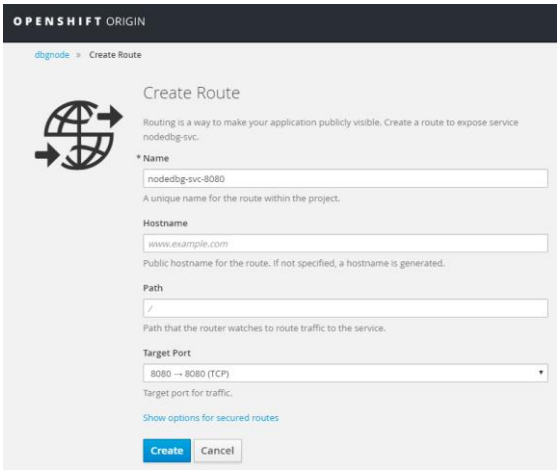
oc create –f svc.yaml

Go to the Openshift console and create a route for each of these ports : (8080 & 3000 in our case)



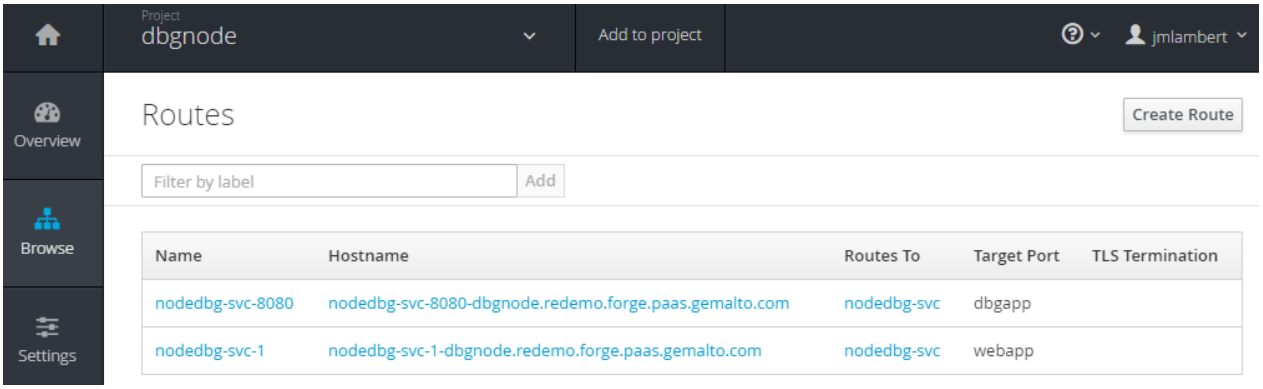
Give a different name to the route object,

Select the port you want to route in the popup.



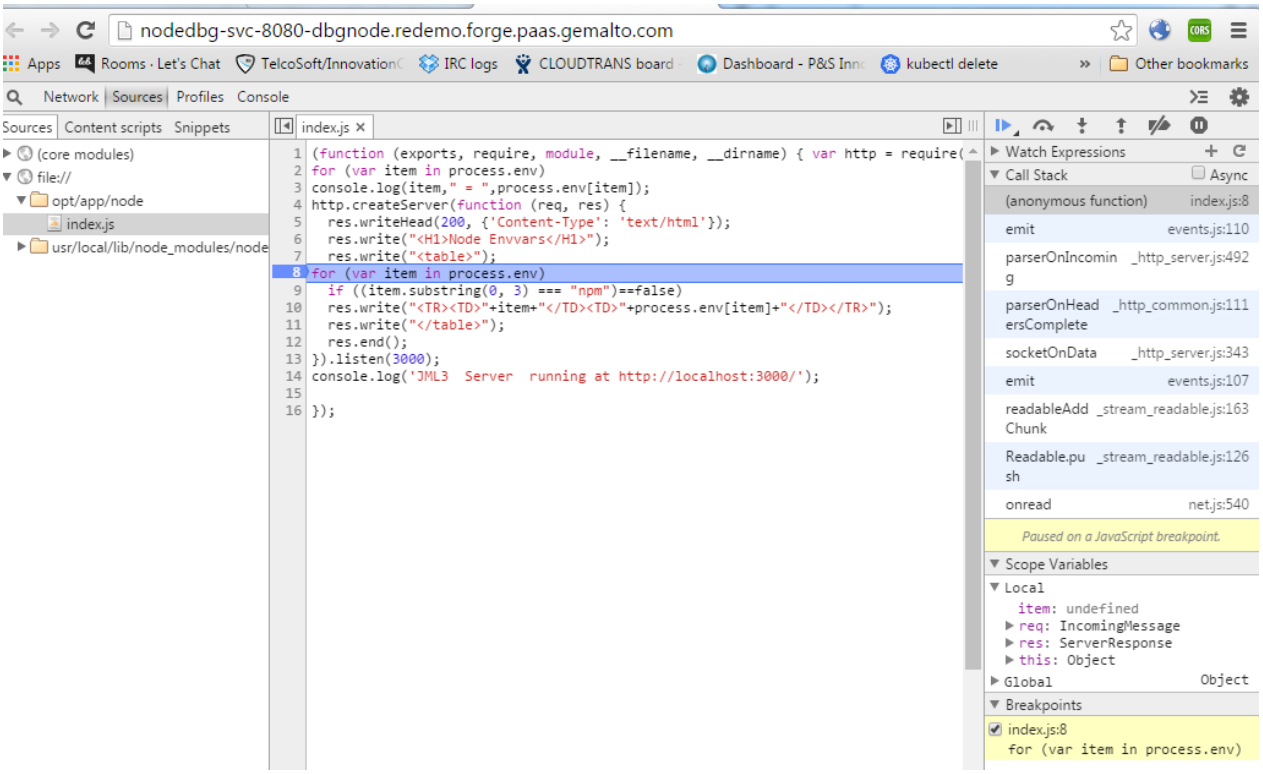
Click on the CREATE to create the new route.

This will allocate a url specific to this namespace, and the service name like :

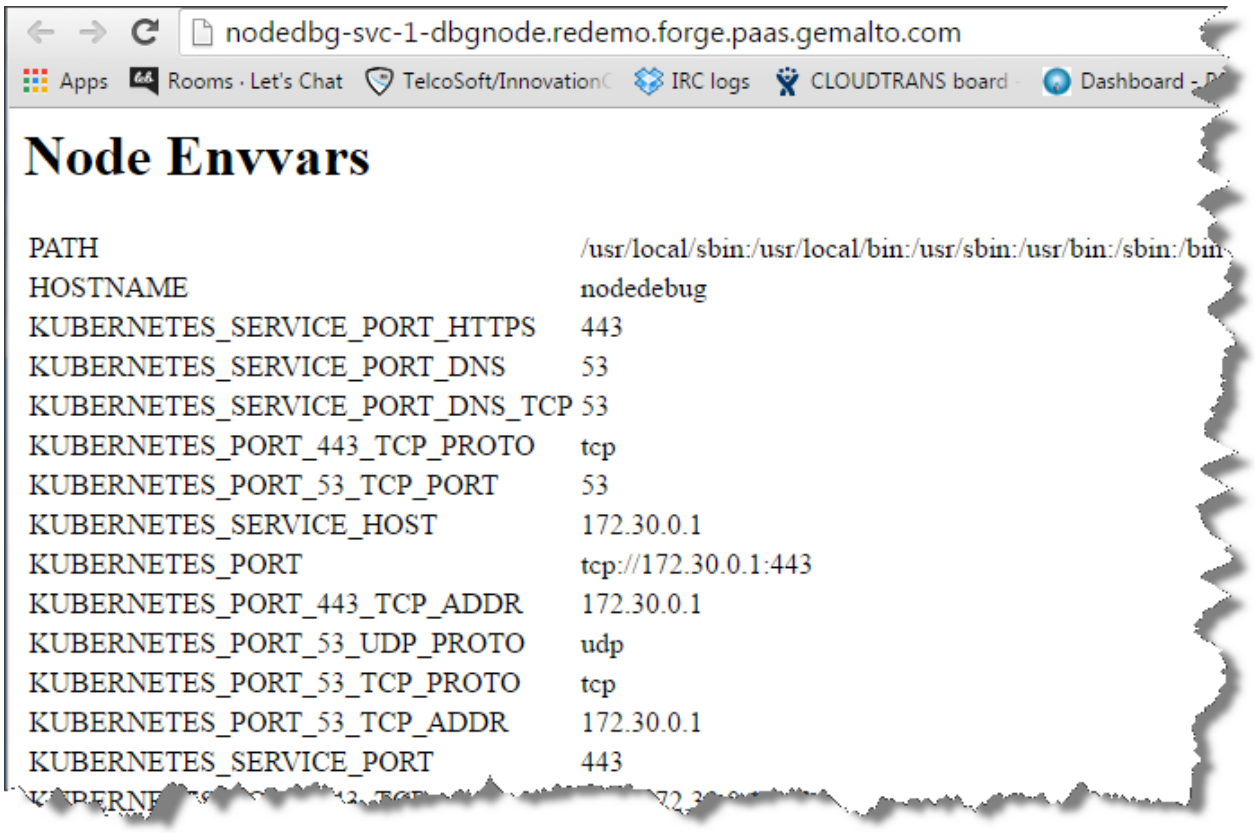


Then you may access to these 2 URL :

The debugging window : with a breakpoint and variables display etc :



And the application side itself :



Conclusion

This process may be included within the Forge pipeline to automatically build the "debug ready" images for the node applications, and allow the debugging at the deployment staging, by adding the right services & routes to debug the right microservices instances.

The additional files (yaml) are located also at the github area :

https://github.com/jmlambert78/docker-debugnode

If you find difficulties, or have other questions, call jean-marc.lambert@gemalto.com