Article <u>Mikhail Khomenko</u> · Nov 18, 2019 9m read

Open Exchange

Deploying InterSystems IRIS Solution into GCP Kubernetes Cluster GKE Using CircleCI

Most of us are more or less familiar with Docker. Those who use it like it for the way it lets us easily deploy almost any application, play with it, break something and then restore the application with a simple restart of the Docker container.

InterSystems also likes Docker. The InterSystems <u>OpenExchange</u> project contains a number of examples that run InterSystems IRIS images in Docker containers that are <u>easy to download</u> and run. You ' II also find other useful components, such as the <u>Visual Studio IRIS plugin</u>.

It 's easy enough to run IRIS in Docker with additional code for specific use cases, but if you want to share your solutions with others, you 'Il need some way to run commands and repeat them after each code update. In this article, we 'Il see how to us continuous Integration/Continuous Delivery (CI/CD) practices to simplify that process.

Setting Up

We ' II start with simple REST API application based on IRIS. The details of the application can be found in the video <u>Creating REST API with InterSystems IRIS</u>, <u>ObjectScript and Docker</u>. Let ' s see how we could share similar applications with others using_CI/CD.

Initially, we 'Il clone the code into a personal GitHub repository. If you don 't have an account on GitH<u>sbgn up</u> for one. For convenience, add<u>access via SSH</u> so you don 't need to enter a password with each pull or push. Then go to the <u>intersystems-community/objectscript-rest-docker-template</u> project page on GitHub and click the "Use this Template" button to create your own version of the repo based on the template. Give it a name like "my-objectscript-rest-docker-template".

Now pull the project to your local machine:

\$ git clone git@github.com:<youraccount>/my-objectscript-rest-docker-template.git

Next, we ' II add a REST endpoint in the spirit of " hello, world! " .

Endpoints are defined in the src/cls/Sample/PersonREST.cls class. Our endpoint will look like this (defined before the first <Route>):

<Route Url="/helloworld" Method="GET" Call="HelloWorld"/> <Route Url="/all" Method="GET" Call="GetAllPersons"/>

```
•••
```

```
It calls the HelloWorld method:
ClassMethod HelloWorld() As %Status
{
Write "Hello, world!"
Quit $$$OK
}
```

Now we need to consider how this works when pushing to a remote repository. We need to:

- 1. Build a Docker image.
- 2. Save the Docker image.
- 3. Run the container based on this image.

We 'Il use th<u>circleCl</u> service, which is already integrated with GitHub, to build the Docker image. And we 'Il use Google Cloud, which allows you to store Docker images and run containers based on them in Kubernetes. Let 's delve into this a little.

Google Cloud Prerequisites

Let 's assume you 've registered for an account w@bogle Cloud, which provides a free tier of services. Create a project with the name "Development", then create a Kubernetes cluster by clicking the "Create cluster" button:

For the demo, select "Your first cluster" on the left. Choose a newer version of Kubernetes and a machine type of n1-standard-1. For our purposes, one machine should be enough.

Click the Create button, then set up a connection to the cluster. We ' II use the ubectl and gcloud utilities: \$ gcloud init [2] Create a new configuration Configuration name. " development " [2] Log in with a new account Pick cloud project to use configure a default Compute Region and Zone? (Y/n)? y Here europe-west-1b was chosen

\$ gcloud container clusters get-credentials dev-cluster --zone europe-west1-b --project <projectid>

You can get the last command by clicking the "Connect" button:

Check the status from kubectl: \$ kubectl config current-context gkepossible-symbol-254507europe-west1-bdev-cluster

\$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
gke-dev-cluster-pool-2-8096d93c-fw5w Ready <none> 17m v1.14.7-gke.10

Now create a directory called k8s/ under the root project directory to hold the three files that describe the future application in Kubernetes: <u>Namespace</u>, which describes the workspace, <u>Deployment</u>, and <u>Service</u>: \$ cat namespace.yaml apiVersion: v1 kind: Namespace metadata: name: iris

\$ cat deployment.yaml apiVersion: apps/v1beta1 kind: Deployment metadata:

name: iris-rest namespace: iris spec: replicas: 1 strategy: type: Recreate selector: matchLabels: app: iris template: metadata: labels: app: iris spec: containers: - image: eu.gcr.io/iris-rest:v1 name: iris-rest ports: - containerPort: 52773 name: web \$ cat service.yaml apiVersion: v1 kind: Service metadata: name: iris-rest namespace: iris spec: selector: app: iris ports: - protocol: TCP port: 52773 targetPort: 52773 type: LoadBalancer

Send those definitions from your k8s/ directory to the Google Kubernetes Engine (GKE): \$ kubectl apply -f namespace.yaml \$ kubectl apply -f deployment.yaml -f service.yaml

Things won 't be working correctly yet, since we haven 't yet sent the gcr.io/iris-rest:v1 image to the Docker registry, so we see an error: \$ kubectI -n iris get po NAME READY STATUS RESTARTS AGE iris-rest-64cdb48f78-5g9hb 0/1 ErrImagePull 0 50s

\$ kubectl -n iris get svc NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE iris-rest LoadBalancer 10.0.13.219 <pending> 52773:31425/TCP 20s

When Kubernetes sees a LoadBalancer service, it tries to create a balancer in the Google Cloud environment. If it

succeeds, the service will get a real IP address instead of External IP = <pending>.

Before leaving Kubernetes for a bit, let's give CircleCI the ability to push Docker images into the registry and restart Kubernetes deployments by creating a <u>service account</u>. Give your service account EDITOR permission to the project. You 'II find informatio<u>here</u> on creating and storing a service account.

A bit later, when we create and set up the project in CircleCI, you ' II need to add the following three environment variables:

The names of these variables speak for themselves. The value of GCLOUDSERVICEKEY is the JSON structure Google sends you when you press " Create key " and select a key in the JSON format after creating the Service Account:

CircleCI

Let's turn our attention to <u>CircleCl</u> now, where we 'II register using our GitHub account (click Sign Up, then Sign Up with GitHub). After registration, you 'II see the dashboard with projects from your GitHub repository listed on the Add Project tab. Click the Set Up Project button for "my-objectscript-rest-docker-template" or whatever you named the repository created from the objectscript-rest-docker-template repo:

Note: all CircleCI screenshots are made as of October 2019. Changes may occur in new versions.

The page that opens tells you how to make your project work with CircleCI. The first step is to create a folder called .circleci and add a file named config.yml to it. The structure of this configuration file is well described in the <u>official</u> <u>documentation</u>. Here are the basic steps the file will contain:

- 1. Pull the repository
- 2. Build the Docker image
- 3. Authenticate with Google Cloud
- 4. Upload image to Google Docker Registry
- 5. Run the container based on this image in GKE

With any luck, we ' Il find some already created configurations (calle<u>drbs</u>) we can use. There are certified orbs and third-party ones. The certified GCP-GKE <u>orb</u> has a number of <u>limitations</u>, so let's take a third-party orb — <u>duksis</u> — that meets our needs. Using it, the configuration file turns into (replace names — for example, the cluster name — with correct ones for your implementation):

\$ cat .circleci/config.yml
version: 2.1
orbs:
 gcp-gke: duksis/gcp-gke@0.1.9
workflows:
 main:
 jobs:
 - gcp-gke/publish-and-rollout-image:
 google-project-id: GOOGLEPROJECTID
 gcloud-service-key: GCLOUDSERVICEKEY
 registry-url: eu.gcr.io

image: iris-rest tag: \${CIRCLESHA1} cluster: dev-cluster namespace: iris deployment: iris-rest container: iris-rest

The initial configuration of the publish-and-rollout-image task can be viewed on the project page.

We don 't actually need the final three notification steps of this orb, which is good because they won 't work anyway without some additional variables. Ideally, you can prepare your own orb once and use it many times, but we won 't get into that now.

Note that the use of third-party orbs has to be specifically allowed on the "Organization settings" tab in CircleCI:

Finally, it 's time to send all our changes to GitHub and CircleCI: \$ git add .circleci/ k8s/ src/cls/Sample/PersonREST.cls \$ git commit -m "Deploy project to GKE using CircleCI" \$ git push

Let 's check the CircleCI dashboard:

If you forgot to add Google Service Account keys, here 's what you 'll soon see:

So don 't forget to add those environment variables as described in the end of the Google Cloud Prerequisites section. If you forgot, update that information, then click "Rerun workflow."

If the build is successful you ' II see a green bar:

You can also check the Kubernetes pod state separately from the CircleCl Web UI: \$ kubectl -n iris get po -w NAME READY STATUS RESTARTS AGE iris-rest-64chdb48f78-q5sbw 0/1 ImagePullBackOff 0 15m

. . .

iris-rest-5c9c86c768-vt7c9 1/1 Running 0 23s

That last line — 1/1 Running — is a good sign.

Let 's test it. Remember, your IP address will differ from mine. Also, you ' II have to figure out about passwords over HTTP yourself as it 's out of scope for this article. \$ kubectI -n iris get svc NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE iris-rest LoadBalancer 10.0.4.242 23.251.143.124 52773:30948/TCP 18m

\$ curl -XGET -u system:SYS 23.251.143.124:52773/person/helloworld Hello, world!

\$ curl -XPOST -H "Content-Type: application/json" -u <u>system:SYS</u> 23.251.143.124:52773/person/ -d '{"Name":"John Dou"}'

\$ curl -XGET -u <u>s</u>ystem:SYS 23.251.143.124:52773/person/all [{"Name":"John Dou"},]

It seems the application works. You can continue with tests described on project page.

In sum, the combination of using GitHub, CircleCI, and Google Cloud Kubernetes Engine looks quite promising for testing and deployment of IRIS applications, even though it '<u>sot completely free</u>. Also, do not forget that running Kubernetes cluster can gradually eat your virtual (and then real) money. We are not responsible for any charges you may incur.

<u>#Best Practices</u> <u>#Cloud</u> <u>#Containerization</u> <u>#DevOps</u> <u>#Docker</u> <u>#Google Cloud Platform (GCP)</u> <u>#Kubernetes</u> <u>#InterSystems IRIS</u> <u>#Open Exchange</u> <u>Check the related application on InterSystems Open Exchange</u>

Source

URL:https://community.intersystems.com/post/deploying-intersystems-iris-solution-gcp-kubernetes-cluster-gkeusing-circleci