Build and deploy in Kubernetes a Serverless Workflow application using the Kogito Serverless Workflow Operator

This document describes how to build and deploy your workflow application using a local Kubernetes cluster, such as Minikube, along with the Kogito Serverless Operator.

Using the Kogito Serverless Operator will let you both to build and deploy your Kogito Serverless Workflow application so that you only need the workflow definition.

If you hvae got already a container built and pushed on a container registry and you want simply deploying it on Kubernetes, you can do it without the operator following the guide Deploying your Serverless Workflow application on Kubernetes.

The Kogito Serverless Operator is currently in an alpha version and under active development and at the moment is supporting Serverless Workflow definition that are using:

- Functions
- States
  - Supported State types:
  - Switch including dataConditions
  - Inject including data with a transition
- Operation including Actions with functionRef with arguments
- KeepActive
- AutoRetries
- ExpressionsLang (jq or jsonpath)

Prerequisites

- A workflow definition.
- A Kubernetes cluster with admin privileges (if you haven't got one prepare and use a local Minikube instance).
- `kubectl` command-line tool is installed. Otherwise, Minikube handles it.

Prepare a Minikube instance

```
minikube start --cpus 4 --memory 4096 --addons registry --addons metrics-server
--insecure-registry "10.0.0.0/24" --insecure-registry "localhost:5000"
```

To speed up, you can increase cpus and memory options. For example, use `--cpus`

`12 --memory 16384`.

If it does work with the default driver, aka `docker`, you can try to start with the `podman` driver:

```
minikube start [...] --driver podman
```

There are some issues with the `crio` container runtime and Kaniko that the operator is using. Reference: ISSUE-2201

Setup Kogito Serverless Operator

In order to have an up and running instance of the Kogito Serverless Operator you can use the following command:

```
kubectl create -f https://raw.githubusercontent.com/kiegroup/kogito-serverless-
operator/main/operator.yaml
```

Follow the deployment of the Kogito Serverless Operator:

+

```
kubectl get pod -n kogito-serverless-operator-system --watch
```

You can also follow the operator’s log:

+

```
kubectl logs deployment/kogito-serverless-operator-controller-manager -n kogito-
serverless-operator-system -f
```

Once the operator is running it will watch for new custom resources so that you can prepare your environment to be ready to build new Serverless Workflow based on the definitions you will send to the operator.

Prepare for the build

Follow these steps to create a container that you can than deploy as a Service on Kubernetes or KNative.

- Create a namespace for the building phase

Let’s create a new namespace that will hold all the resources that we (or the operator) will create (pods, deployments, services, secretes, config map and custom resources) in this guide.

```
kubectl create namespace kogito-workflows
```

- Create a secret for the container registry authentication

```
kubectl create secret docker-registry regcred --docker-server=<registry_url> --docker
-username=<registry_username> --docker-password=<registry_password> --docker
-email=<registry_email> -n kogito-workflows
```

or you directly import your local docker config into your kubernetes cluster:

```
kubectl create secret generic regcred --from
-file=.dockerconfigjson=${HOME}/.docker/config.json
--type=kubernetes.io/dockerconfigjson -n kogito-workflows
```

- Create a Platform containing the configuration (i.e. registry address, secret) for building your workflows

You can find a basic Platform CR example in the config folder that you can simply apply to configure your operator.

```
kubectl apply -f config/samples/sw.kogito_v1alpha08_kogitoserverlessplatform.yaml -n
kogito-workflows
Note: In this Custom Resource, spec.platform.registry.secret is the name of the secret
you created just before.
```

You can also update "on-the-fly" the platform CR registry field with this command (change <YOUR_REGISTRY>):

```
cat config/samples/sw.kogito_v1alpha08_kogitoserverlessplatform.yaml | sed "s|address:
.*|address: <YOUR_REGISTRY>"
```

You can retrieve easly the Cluster IP address of the minikube internal registry using this command:

```
kubectl get svc registry -n kube-system -ojsonpath='{.spec.clusterIP}'
```

Build and deploy your Serverless Workflow application

Now we can send to the Operator our Custom Resource that includes the Serverless Workflow definition.

If you would like to deploy a simple Serverless Workflow application you can use custom resource you can find in the samples folder that is defining the Kogito Serverless Workflow Greeting example.

```
kubectl apply -f config/samples/sw.kogito_v1alpha08_kogitoserverlessworkflow.yaml -n
kogito-workflows
```

You can check the logs of the build of your workflow via:

```
kubectl logs kogito-greeting-builder -n kogito-workflows
```

The final pushed image should be printed into the logs at the end of the build.

In order to check that the Kogito Serverless Workflow Greeting is up and running let's try to perform a test HTTP call.

```
POD=$(kubectl get pod -n kogito-workflows -l app=greeting -o
jsonpath="{.items[0].metadata.name}")
kubectl exec -n kogito-workflows --stdin --tty $POD -- /bin/bash
curl -X POST -H 'Content-Type:application/json' -H 'Accept:application/json' -d
'{"name": "John", "language": "English"}' http://localhost:8080/jsongreet
```

If everything is working well you should receive a response like this:

```
{"id":"b5fbfaa3-b125-4e6c-9311-fe5a3577efdd","workflowdata":{"name":"John","language"
:"English","greeting":"Hello from JSON Workflow, "}}
```