記事
Toshihiko Minamoto · 2022年12月21日 8m read

Kubeless を使って FasS モードで InterSystems IRIS を実行する

> FaaS（Function as a Service）は、アプリケーションの機能を開発、実行、および管理するプラットフォームを提供するクラウドコンピューティングサービスのカテゴリです。アプリの開発と起動に一般的に関連するインフラストラクチャの複雑な構築や保守がありません。
> このモデルに従ったアプリケーションの構築は、「サーバーレス」アーキテクチャを実現する方法の 1 つであり、通常、マイクロサービスアプリケーションを構築する際に使用されています。
>
> Wikipedia

FaaS は、クラウドでワークロードを実行するための非常に一般的なアプローチで、開発者はコードを書くことに専念することができます。

この記事では、InterSystems IRIS のメソッドを FaaS 方式でデプロイする方法を説明します。

Kubernetes をインストール

まず、**Kubernetes 1.16** をインストールします。
多数のガイドが用意されているため、ここには書き写しませんが、私は minicube を使用しています。 Kubernetes の実行に minicube を使用する場合、以下のコマンドを実行するだけです。

```
minikube start --kubernetes-version=v1.16.1
```

Kubeless をインストール

次に、Kubeless をインストールします。 Kubeless は Kubernetes ネイティブのサーバーレスフレームワークで、基盤のインフラストラクチャの構築を気にすることなく、小さなコードをデプロイできます。 Kubernetes リソースを利用して、自動スケール、API
ルーティング、監視、トラブルシューティングなどの機能を提供します。

```
kubectl create ns kubeless
kubectl create -f https://github.com/kubeless/kubeless/releases/download/v1.0.8/kubeless-v1.0.8.yaml
kubectl get pods -n kubeless
```

出力は以下のようになります。

```
NAME READY STATUS RESTARTS AGE
 kubeless-controller-manager-666ffb749-26vhh 3/3 Running 0 83s
```

また、Kubeless クライアントをインストールする必要もあります（kubectl のあるインスタンスにインストール）。 これはこちらで入手できます。 Linux でのインストールは、以下のように単純です。

```
sudo install kubeless /usr/local/bin/kubeless
```

Kubeless をテスト

まず、単純な Python 関数をデプロイして、Kubeless が動作することを確認しましょう。

test.py を作成します。

```
def hello(event, context):
  return event['data']
```

関数環境についての詳細は、こちらのドキュメント をお読みください。一般的に関数は、以下のデータを伴うイベントとコンテキストの 2 つの引数を受け入れます。

```
event:
  data:                                          # Event data
    foo: "bar"                                   #
 The data is parsed as JSON when required
  event-id: "2ebb072eb24264f55b3fff"             # Event ID
  event-type: "application/json"                 # Event content type
  event-time: "2009-11-10 23:00:00 +0000 UTC"    # Timestamp of the event source
  event-namespace: "kafkatriggers.kubeless.io"   # Event emitter
  extensions:                                    # Optional parameters
    request: ...                                 # Reference to the request received
    response: ...                                # Reference to the response to send

          # (specific properties will depend on the function language)
context:
    function-name: "pubsub-nodejs"
    timeout: "180"
    runtime: "nodejs6"
    memory-limit: "128M"
```

次に、関数とランタイムを含むファイルを指定して、hello 関数をデプロイします。

```
kubeless function deploy hello --runtime python3.7 --from-
file test.py --handler test.hello
kubeless function ls hello
```

では、テストしましょう。

```
kubeless function call hello --data 'Hello world!'
```

答えとして *Hello World!* を受け取るはずです。

IRIS 構成を追加

次に、InterSystems IRIS 関数ハンドラーを追加する必要があります。編集する kubeless 構成を開きましょう。

```
kubeless get-server-config
kubectl get -n kubeless configmaps -o yaml > configmaps.yaml
kubectl edit -n kubeless configmaps
```

以下のエントリを runtime-images 配列に追加して保存します。

```
{"ID": "iris","depName": "","fileNameSuffix": ".cls","versions": [{"images": [{"image
": "eduard93/kubeless-iris-
runtime:latest","phase": "runtime"}],"name": "iris2022.1","version": "2022.1"}]}
```

変更が適用されるように、Kubeless コントローラーを再起動します。

```
kubectl delete pod -n kubeless -l kubeless=controller
```

IRIS 関数 CRD を構築して公開

では、InterSystems IRIS に最初の関数を書き込みましょう。

```
Class User.Test {

ClassMethod hi(event, context) As %Status
{
    if $isObject(event) {
        write event.Text + event.Text
    } else {
        write "HELLO FROM IRIS"
    }
    quit $$$OK
}
}
```

次に、関数 CRD を構築する必要があります。

以下は、テンプレートです。

function.yaml

そして、以下の内容を入力します。

- *name*: 関数名（kubeless）
- *handler*: class.namemethod（InterSystems IRIS）
- *function の本体*: 最後に追加します（タブを忘れずに！）

すると、CRD は以下のようになります。

functiondemo.yaml

これは簡単に自動化できます。 Linux では以下を実行します。

```
sed 's/!name!/iris-
demo/; s/!handler!/User_Test.hi/' function.yaml > function_demo.yaml
sed  's/^/     /'  User.Test.cls >> function_demo.yaml
```

Windows では以下は実行します（PowerShell）。

```
Get-Content function.yaml | ForEach-Object { $_ -replace "!handler!", "User_Test.hi"
-replace "!name!", "iris-demo" } | Set-Content function_demo.yaml
"    " + [string]((Get-Content User.Test.cls)  -join "`r`n    ") | Add-
Content function_demo.yaml
```

Kubeless にこの CRD を公開する必要があります。

```
kubectl apply -f function_demo.yaml
```

IRIS 関数をテスト

まず、関数がデプロイされて準備が完了していることを確認しましょう（初回時は、準備に数分かかることがあります）。

```
kubeless function ls
```

次にこれを呼び出します。

```
kubeless function call iris-demo --data '{"Text":123}'
```

Windows を使用している場合は、以下のように関数を呼び出します（エスケープ付きの二重引用符を使ったその他すべての呼び出しでも共通です）。

```
kubeless function call iris-demo --data '{\"Text\":123}'
```

どちらで操作しているかに関わらず、*123* が数値であるため、レスポンスは *456* になります。

HTTP アクセス

Kubeless には HTTP アクセスも提供されています。 これをテストするには、以下の kubectl proxy コマンドを使用します。

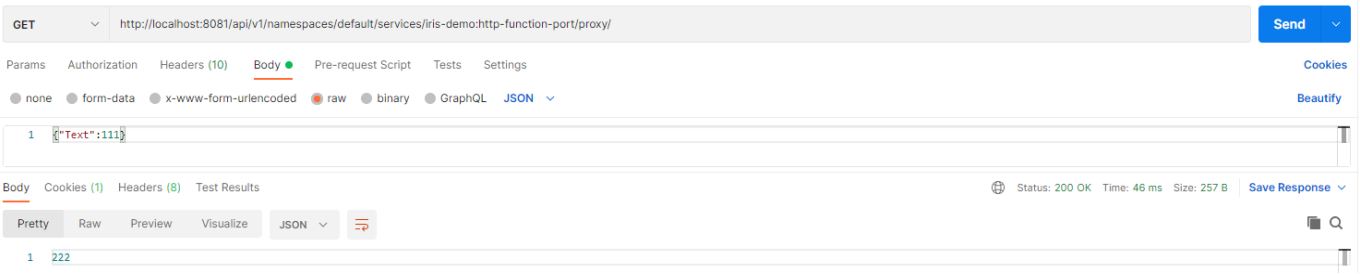
```
kubectl proxy -p 8081
```

次に、このリクエストを任意の REST API クライアントを使って送信します。

```
GET http://localhost:8081/api/v1/namespaces/default/services/iris-demo:http-function-port/proxy/

{"Text":111}
```

Postman では以下のように表示されます。



次に、これをインターネットに公開しましょう。

これには 2 つのアプローチがあります。
イングレスをこちらに説明されているとおりに構成するのがお勧めです。

また、関数サービスをパッチすることができます。

```
kubectl get svc
kubectl patch svc iris-demo -p '{"spec": {"type": "LoadBalancer"}}'
kubectl get svc
```

クリーンアップ

デプロイした関数呼び出しを削除するには、以下のようにします。

```
kubectl delete -f function_demo.yaml
```

まとめ

これは紛れもなくコンセプトの証明であり、本番級のソリューションではありませんが、このアプローチでは、サーバーレスの FaaS アプローチを使って、InterSystems IRIS ワークロードを実行できることが分かりました。

リンク

- Minicube
- Kubeless
- InterSystems IRIS ランタイム

#Docker #クラウド #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/kubeless-%E3%82%92%E4%BD%BF%E3%81%A3%E3%81%A6-fass-%E3%83%A2%E3%83%BC%E3%83%89%E3%81%A7-intersystems-iris-%E3%82%92%E5%AE%9F%E8%A1%8C%E3%81%99%E3%82%8B