Artículo
Alberto Fuentes · 8 jun, 2022 Lectura de 4 min

Open Exchange

Cómo hacer que Github ejecute tus pruebas unitarias

Hola desarrolladores!

Este es otro artículo para simplificar la vida de los desarrolladores. Hablamos de hacer que GitHub ejecute tus pruebas unitarias (*unittest*) con cada *push* que hagas a tu repositorio simplemente añadiendo un fichero. Gratis :). En GitHub Cloud. Suena genial, ¿no?

Es factible, y además muy sencillo. El mérito es para @Dmitry Maslennikov (y su repo), el gestor de paquetes ZPM y las GitHub Actions. Vamos a ver cómo funciona todo en conjunto.



Vamos a partir de un repositorio que tenga algunas pruebas unitarias, por ejemplo la plantilla básica para IRIS.

Asumimos también que cargas el código a un contenedor IRIS utilizando ZPM. Si no es así, échale un vistazo a este video para aprender a hacerlo.

En este repositorio en particular, se hace con la siguiente línea y la presencia de module.xml:

```
    zpm "load /home/irisowner/irisbuild/ -v":1:1
```

A continuación, añadamos un escenario GitHub Actions que llevará a cabo la construcción de la imagen y la ejecución de las pruebas unitarias:

```
name: unittest

on:
  push:
    branches:
      - master
      - main
  pull_request:
    branches:
      - master
      - main
  release:
    types:
      - released

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Build and Test
        uses: docker/build-push-action@v2
        with:
          context: .
          push: false
          load: true
          tags: ${{ github.repository }}:${{ github.sha }}
          build-args: TESTS=1
```

Este escenario construye una imagen docker utilizando el Dockerfile del repositorio con cada **push** o **pull-request** a las ramas **master** o **main**.

Hay una línea adicional que pasa la variable TESTS=1 al Dockerfile:

```
          build-args: TESTS=1
```

Este argumento se evalúa en el Dockerfile y en el que caso TESTS=1, ejecuta las pruebas del paquete zpm del repositorio:

```
  ([ $TESTS -eq 0 ] || iris session iris -U $NAMESPACE "##class(%ZPM.PackageManager).Shell(\"test $MODULE -v -only\",1,1)") && \
```

Esta línea comprueba el parámetro $TESTS y si no es igual a 0, abre una sesión en el $NAMESPACE (IRISAPP en este caso) y ejecuta el comando ZPM:

```
test $MODULE -v -only
```

-only es una opción que ejecuta los tests únicamente sin realizar la carga del módulo (dado que ya ha sido cargado previamente en el docker build del escenario).

El nombre del módulo se establece via $MODULE="dc-sample-template" en este caso:

```
ARG MODULE="dc-sample-template"
```

El comando que ejecutará todas las pruebas unitarias se menciona en el módulo ZPM. En este caso, lo configuramos con esta línea:

```
    <UnitTest Name="/tests" Package="dc.sample.unittests" Phase="test"/>
```

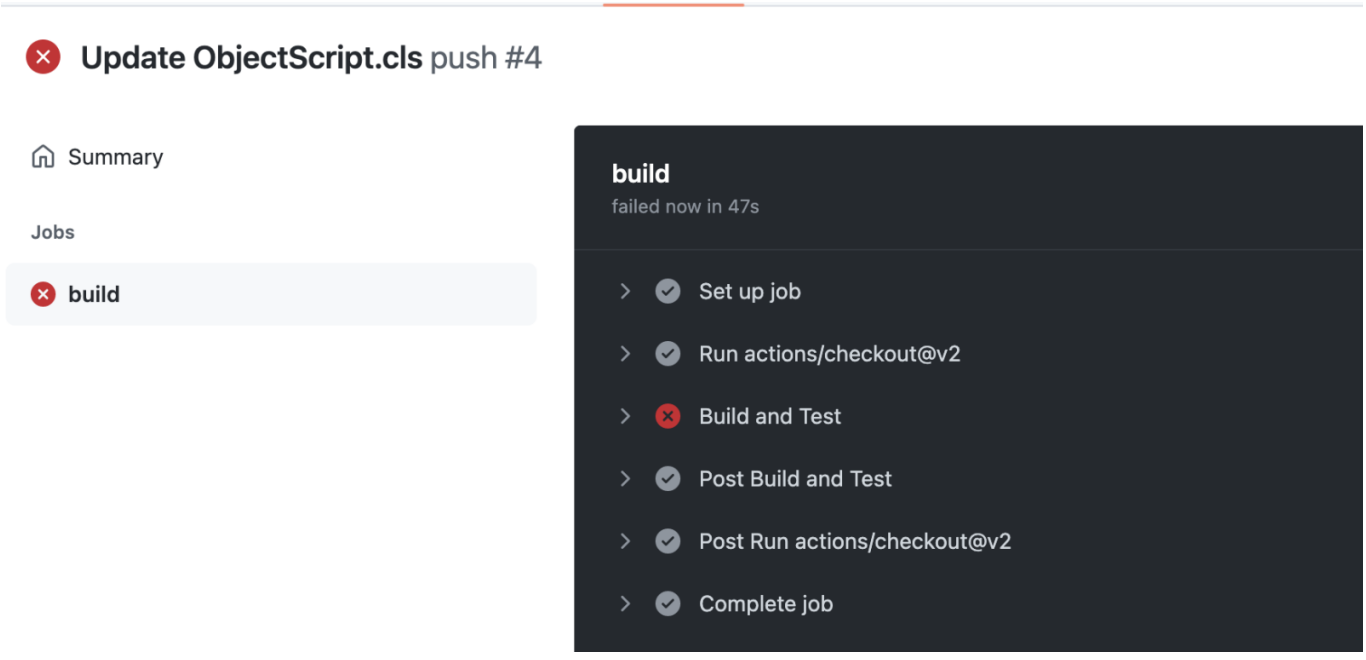
que significa que las pruebas unitarias están en el directorio /tests del repositorio y que el recurso es el paquete de clases
dc.sample.unitests que tiene dos clases.

Este escenario construirá una imagen del repositorio en GitHub cloud y ejecutará las pruebas unitarias en cada push o pull-request que se haga en la rama master!

Vamos a ver cómo funciona.

El método Test42 espera que el método dc.sample.ObjectScript.Test() devuelva 42.

Cambiemos la línea a 43 y enviemos un pull-request. Podemos ver (en la sección Actions) que el pull request inició un GitHub Action, la construcción falló:



Y los detalles de la fase dicen que el Test42() falla en el AssertEquals 42 = 43:

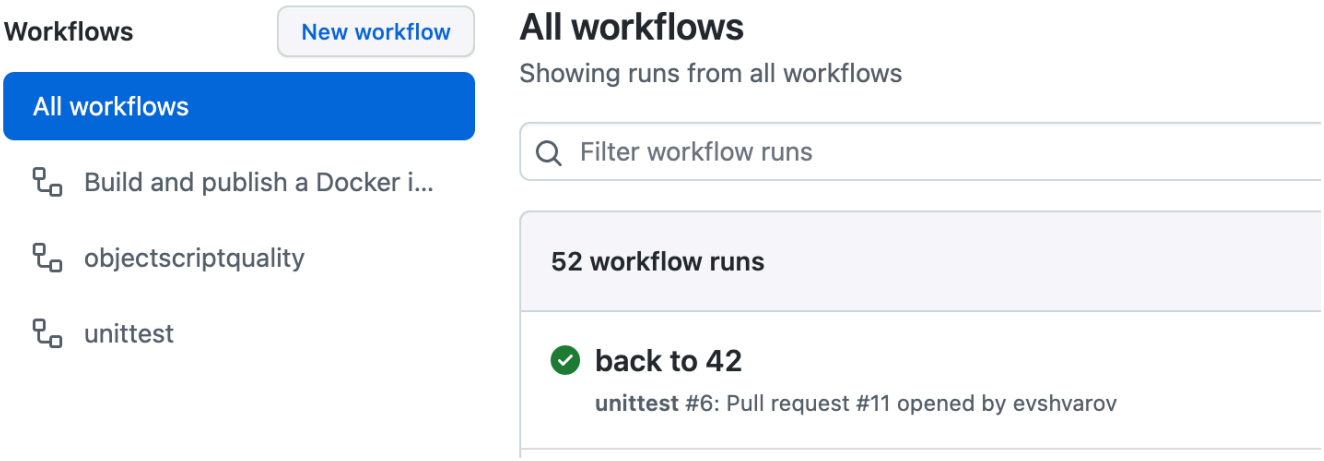
```
#8 9.273     dc.sample.unittests.TestCreateRecord begins ...
#8 9.274       TestCreateRecord() begins ...
#8 9.274         AssertStatusOK:CreateRecord (passed)
#8 9.274         AssertEquals:Test string = Test string (passed)
#8 9.274         LogMessage:Duration of execution: .000271 sec.
#8 9.274       TestCreateRecord passed
#8 9.275     dc.sample.unittests.TestCreateRecord passed
#8 9.275     dc.sample.unittests.TestObjectScript begins ...
#8 9.275       Test42() begins ...It works!
#8 9.275
#8 9.275 AssertEquals:42 = 43 (failed)  <<==== **FAILED**   (root):dc.sample.unittests.TestObjectScript:Test42
#8 9.275         LogMessage:Duration of execution: .000062 sec.
#8 9.275       Test42 failed
#8 9.276     dc.sample.unittests.TestObjectScript failed
#8 9.276   Skipping deleting classes
#8 9.276   (root) failed
#8 9.378
#8 9.378 Use the following URL to view the result:
#8 9.378 http://127.0.0.1:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=1&$NAMESPACE=IRISAPP
#8 9.378 Some tests FAILED in suites:
#8 9.378
#8 9.472 [dc-sample-template]   Test FAILURE
#8 9.473 ERROR! 1 assertion(s) failed.
#8 ERROR: executor failed running [/bin/sh -c iris start IRIS &&        iris session IRIS < iris.script &&     ([ $TESTS -eq 0 ] ||
iris session iris -U $NAMESPACE "##class(%ZPM.PackageManager).Shell(\"test $MODULE -v -only\",1,1)") &&     iris stop IRIS quietly]:
exit code: 1
```

Github también envía notificaciones de emails de los CI fallidos.

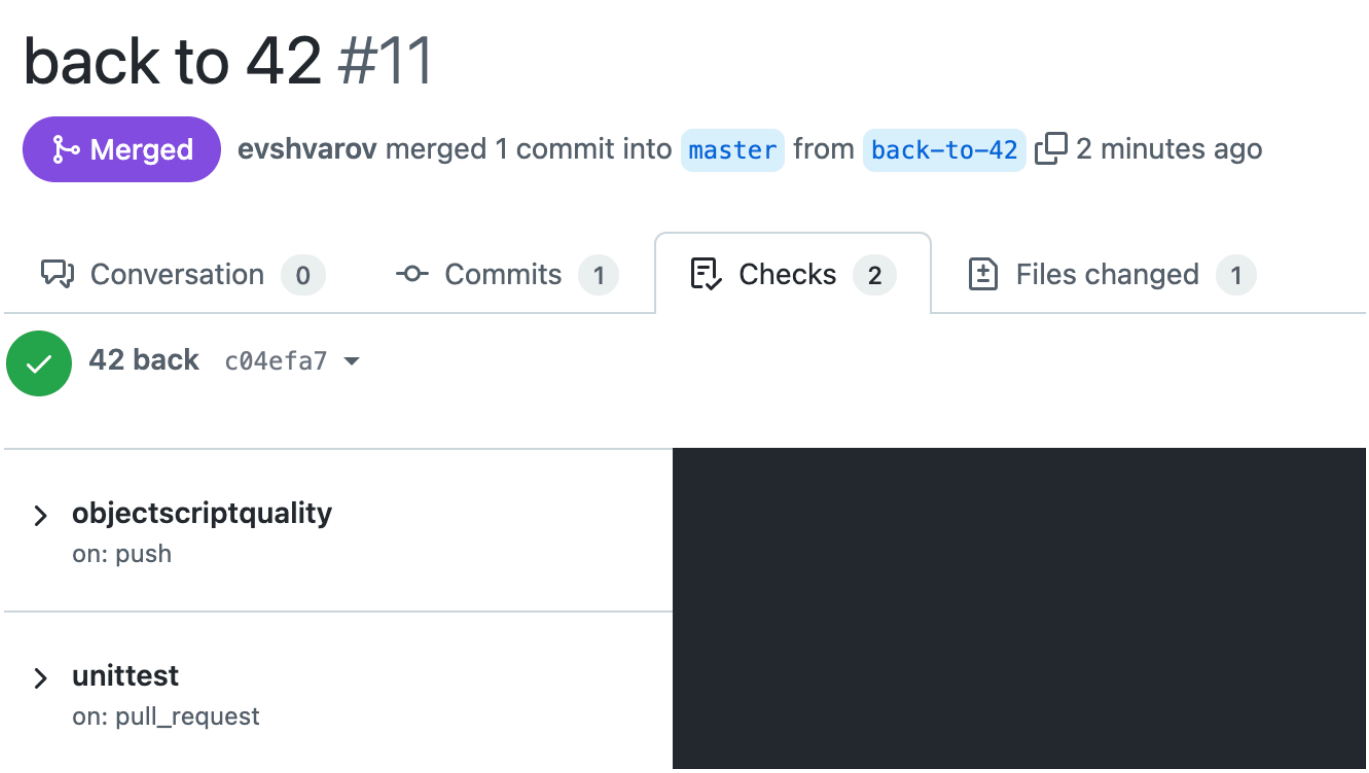
Nota: para ejecutar las pruebas unitarias en local:

USER>zpm test "module-name"

Y si cambiamos de nuevo el valor de la variable a 42, las pruebas resultarán correctas!



Y si es un Pull-Request puedes ver el resultado del CI en la sección especial Checks:



Y ya estaría :)

En resumen, hacer que GitHub construya imágenes docker y ejecute tus pruebas unitarias en tus proyectos IRIS con cada push o pull request requiere un escenario Github Actions para CI, que será el mismo en cada proyecto, y tres líneas en el Dockerfile:

ZPM $MODULE name - tendrás que cambiar esto en cada proyecto,

$TEST como parámetro,

and la línea RUN TEST.

Y por supuesto, utilizar el gestor de paquetes ZPM!

Happy coding!

P.D. la imagen de la entrada está relacionada con uno de mis escritores favoritos de ciencia ficción, la historia de Robert Sheckley "Something for Nothing" - aquí lo tenéis en audio también, disfrutadlo!

#DevOps #Docker #Entorno de desarrollo #GitHub #InterSystems Package Manager (IPM) #Prueba #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/c%C3%B3mo-hacer-que-github-ejecute-tus-pruebas-unitarias