

Artículo

[Mathew Lambert](#) · 15 jul, 2020 · Lectura de 4 min

Historia del proyecto isc-tar

¡Hola desarrolladores!

Aquí podéis ver el [anuncio](#) del proyecto isc-tar de [@Dmitriy Maslennikov](#). En ocasiones, la historia de porque se ha llegado a un resultado es igual o más interesante que el resultado: cómo se construyó, cómo funciona y qué sucede en torno al proyecto. Esta es la historia:

- Cómo desarrollar este proyecto
- Cómo probarlo
- Cómo lanzar nuevas versiones para publicar
- Cómo automatizar todo lo anterior
- Integración continua

Os hablaré de todo eso.

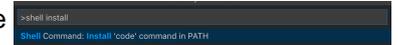
Desarrollo

Últimamente, mis herramientas favoritas son [Docker](#) y [VSCode](#). Uso Docker para empezar cualquiera de mis proyectos en su propio entorno y VSCode es sencillamente el mejor editor existente, el cual uso junto con la extensión [vscode-objectsript](#). En cualquier nuevo proyecto, uso por lo menos esas dos herramientas. También soy usuario de macOS, por lo que todos los siguientes pasos fueron probados y deberían funcionar bien en macOS y quizás también en Linux, pero podría ser distinto para Windows.

- Antes de nada, clonamos nuestro proyecto en algún sitio

```
$ git clone git@github.com:daimor/isc-tar.git
```

- Yo tengo el código de comando en mi intérprete, para poder abrir fácilmente cualquier carpeta desde mi terminal con el código. Si quieres esta funcionalidad, puedes configurarla desde la consola de comandos. Para que quede disponible el comando, tendrás que actualizar tu intérprete

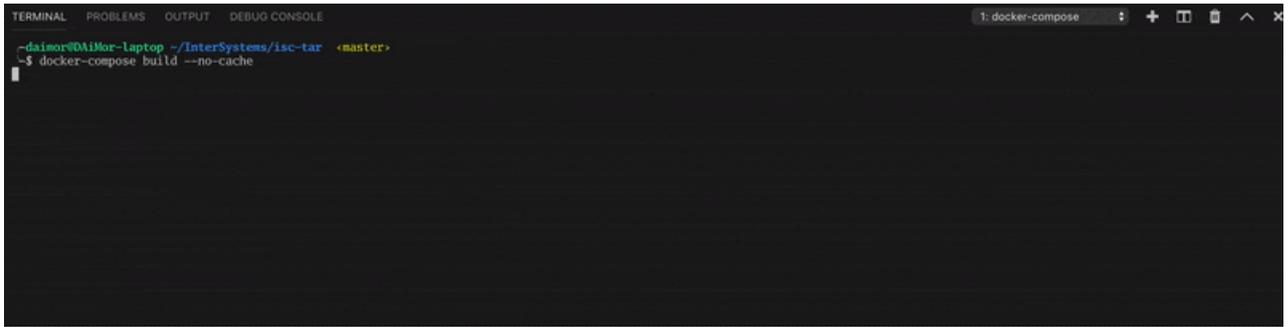


- Abre el editor solo con el repositorio clonado

```
$ code isc-tar
```

- VSCode tiene la opción de un terminal integrado. Ahora puedes abrirlo sin perder tu terminal, que estará muy cerca siempre. Podemos usar el terminal para configurar el entorno del proyecto con los dos comandos siguientes:

```
$ docker-compose build --no-cache  
$ docker-compose up -d
```



```
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
~daimor@DAIMor-laptop ~/InterSystems/isc-tar <master>
~$ docker-compose build --no-cache
```

esto puede tardar un poco.

- Y ya puedes escribir código. Este repositorio contiene la configuración para VSCode, por lo que ya está preconfigurado para usar este IRIS

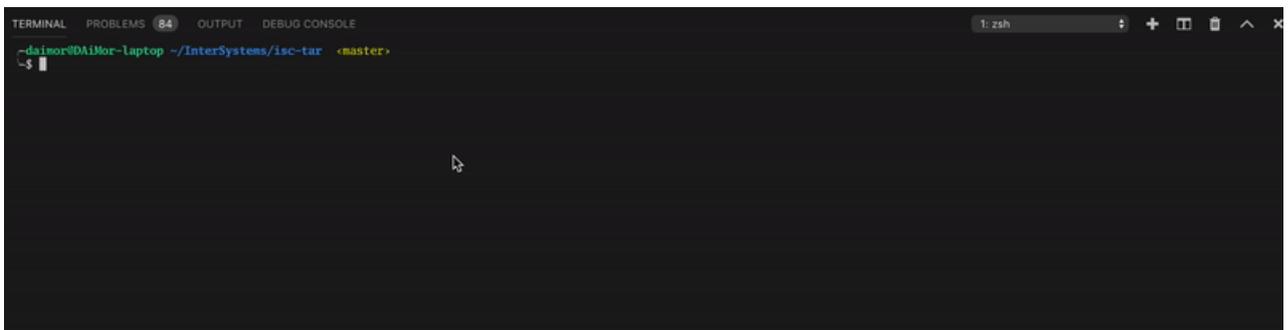
Pruebas

El proyecto tiene un par de pruebas que puedes realizar:

- en el terminal, ve a la sesión de IRIS

```
$ docker-compose exec iris iris session iris
```

```
USER>do ##class(%UnitTest.Manager).RunTest()
```



```
TERMINAL PROBLEMS 84 OUTPUT DEBUG CONSOLE
~daimor@DAIMor-laptop ~/InterSystems/isc-tar <master>
~$
```

Por cierto, ¿has notado los 3 "iris" en el comando? A mí no me gusta tener que repetir las cosas, y hay que escribir "iris" tres veces para lanzar el intérprete de IRIS. La primera vez es para el nombre del servicio en docker-compose, la segunda es el comando iris dentro del contenedor (que sustituye a ccontrol que se usaba en Caché), y la tercera es para el nombre de la instancia. Me gustaría que InterSystems añadiera un comando session más corto, para eliminar esta innecesaria repetición de la palabra "iris". Debería ser algo como esto

```
docker-compose exec iris session
```



```
~daimor@DAIMor-laptop ~/InterSystems/isc-tar <master>
~$ docker-compose exec iris session
Node: 6142138b10a0, Instance: IRIS
USER>
```

Como se comentó en el [anuncio](#), esta herramienta también podría funcionar en versiones anteriores de Caché y Ensemble. Pero no todas esas versiones soportan UDL como formato de importación. Por lo tanto, necesitamos XML, que se puede usar siempre, tanto en IRIS como en Caché/Ensemble. Afortunadamente, tengo una sola clase y puedo hacer una exportación simple.

```
do $system.OBJ.Export("%zUtils.FileBinaryTar.cls", "/opt/zUtils.FileBinaryTar.xml", "
```

```
/diffexport")
```

Y lamentablemente, por algún motivo desconocido, no puedo usar un útil [qualifier](#) /exportversion=2010.1

```
USER>do $system.OBJ.Export("%zUtils.FileBinaryTar.cls", "/opt/zUtils.FileBinaryTar.xml", "/diffexport/exportversion=2010.1")
```

```
Exporting to XML started on 03/16/2019 09:18:35
Exporting class: %zUtils.FileBinaryTar
ERROR #5126: XML export version '2010.1' not supported, supports 2010.1 and onwards.
Errors detected during export.
```

En el XML exportado, necesito sustituir el encabezado Export para que también pueda reconocerlo Caché.

```
<Export generator="IRIS" version="26">
```

Y para eso uso el comando sed.

```
sed -i.bak 's/^<Export generator="IRIS" .*$/<Export generator="Cache" version="25"/>/g' /opt/zUtils.FileBinaryTar.xml
```

Ahora se puede usar tanto en Caché como en IRIS.

Automatización

Para ejecutar pruebas y compilar versiones, es necesario recordar e invocar comandos largos, por lo que sería bueno simplificarlo lo máximo posible. Y Makefile es perfectamente adecuado para ello.

```
APP_NAME = isc-tar
IMAGE ?= daimor/$(APP_NAME)

SHELL := /bin/bash

.PHONY: help build test release

help: ## This help.
    @awk 'BEGIN {FS = ".*?## "} /^[a-zA-Z_]+:.*?## / {printf "\033[36m%-30s\033[0m %s\n", $$1, $$2}' $(MAKEFILE_LIST)

.DEFAULT_GOAL := help

build: ## Build the image
    docker build -t $(IMAGE) .

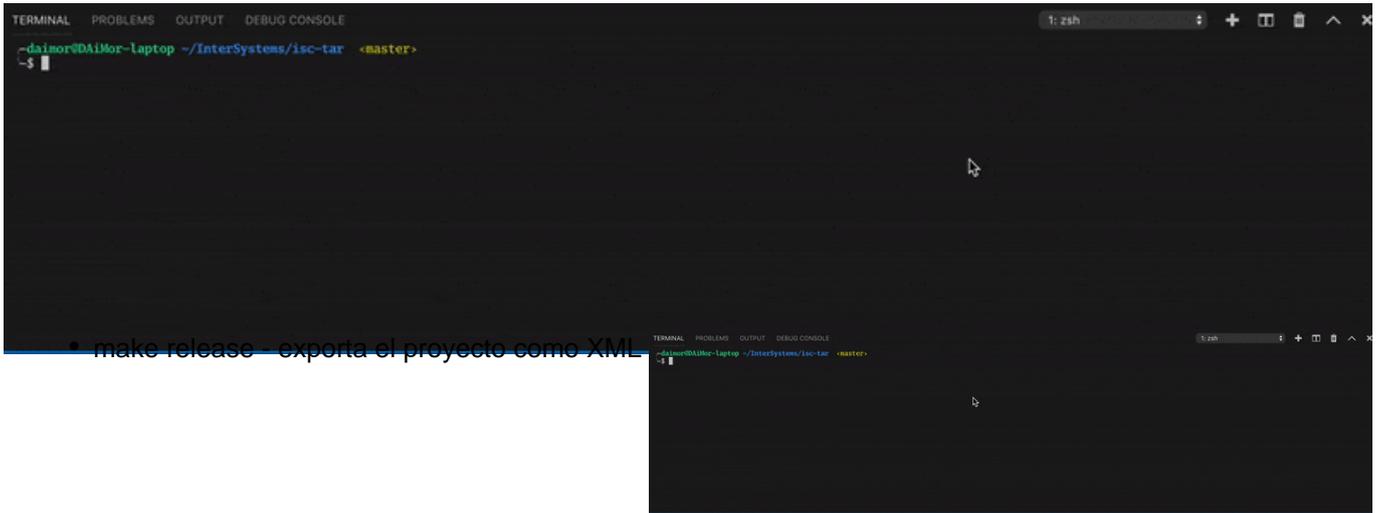
test: build ## Run UnitTests
    docker run --rm -i -v `pwd`/tests:/opt/tests -w /opt --entrypoint /tests_entrypoint.sh $(IMAGE)

release: clean build ## Export as XML
    docker run --rm -i -v `pwd`/out:/opt/out -w /opt --entrypoint /build_artifacts.sh $(IMAGE)

clean:
    -rm -rf out
```

Mi Makefile es bastante simple. Tengo dos recetas principales

- make test - ejecuta Unit Tests



- make release - exporta el proyecto como XML

Estas dos recetas dependen de build, que compila la imagen de IRIS con el proyecto dentro.

Integración continua

Por supuesto, el artículo no estaría completo sin la integración continua, en breve se publicará la segunda parte, y estoy seguro de que os sorprenderá cómo se hizo.

[#DevOps](#) [#Docker](#) [#Entorno de desarrollo](#) [#VSCode](#) [#InterSystems IRIS](#)

URL de fuente: <https://es.community.intersystems.com/post/historia-del-proyecto-isc-tar>