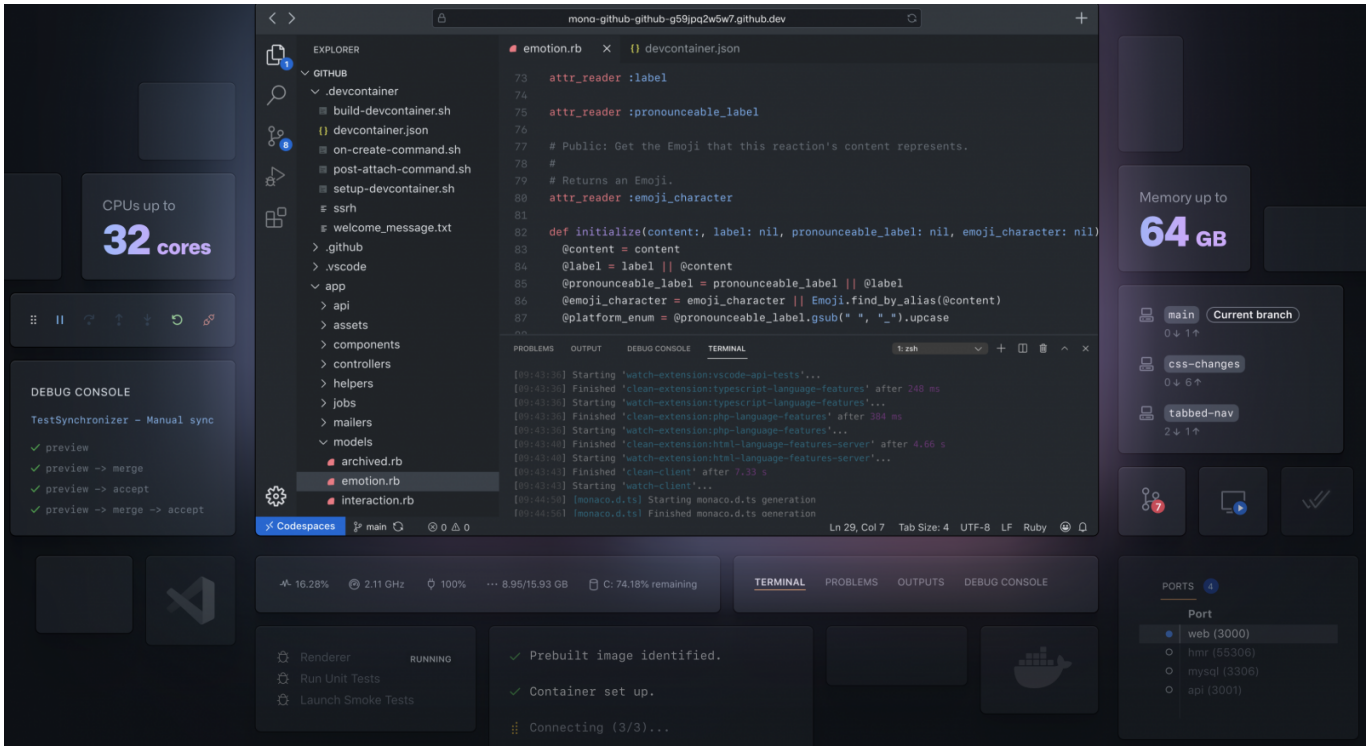
Artículo
Alberto Fuentes · 9 dic, 2021 Lectura de 7 min

Codespaces de GitHub con IRIS

¡Hola a todos! Comparto hoy con vosotros un artículo muy interesante de Dmitry Maslennikov acerca de una nueva funcionalidad de GitHub llamada GitHub Codespaces. Permite ejecutar VSCode en el navegador, con casi la misma potencia con la que se ejecutaría de forma local en tu equipo, pero con la potencia de la nube, por lo que podrás elegir el tipo de equipo con hasta 32 núcleos de CPU y 64 GB de RAM.



Parece impresionante, ¿verdad? Pero, ¿cómo nos podría ayudar a trabajar con proyectos realizados con InterSystems IRIS? Vamos a ver cómo configurarlo.

Un inicio sencillo para cualquier repositorio

Con esta funcionalidad, podrás editar cualquier repositorio en la nube, mediante el botón Code. Ten en cuenta que esta función aún está en fase beta, y podría no estar disponible para todos, y después del periodo beta solo estará disponible para cuentas de pago.

Así que este repositorio no se ha preparado especialmente para Codespaces, solo para VSCode. Haz clic en el botón *New codespace*, para crear un entorno solo para ti.

En mi caso, en el uso anterior de codespaces, ya había instalado la extensión ObjectScript y la había habilitado de forma global mediante línea de comando desde Codespaces. Por lo tanto, me lo instala cada vez, y el código ObjectScript ya se resaltó. Pero IRIS aún no está disponible. Empecemos con docker-compose.

Después de eso, ahora podremos conectarnos al terminal de IRIS, y compilar el código

Los puertos de docker-compose se reconocen automáticamente y se pueden abrir en el navegador, por lo que

también es posible abrir el Portal de Administración del Sistema

Uso con repositorio preparado

Logramos ejecutar VSCode e IRIS en la nube, pero tuvimos que hacer algunas cosas manualmente para que estuviera listo. Pero también es posible hacer que tu repositorio esté listo para el desarrollo desde el inicio.

Esto es posible, con devcontainer.json. Mostraré un ejemplo basado en el proyecto Realworld. Este proyecto es bastante complejo, tiene un backend y un frontend y utiliza docker-compose para empezar todo a la vez.

devcontainer también puede utilizar docker-compose, así que mi configuración es algo así:

```
{
  "name": "IRIS RealWorld example",
  "dockerComposeFile": "docker-compose.yml",
  "service": "server",
  "extensions": [
    "intersystems-community.vscode-objectscript"
  ],
  "forwardPorts": [
    80,
    52773
  ],
  "workspaceFolder": "/home/irisowner/conduit",
  "remoteUser": "irisowner",
  "postCreateCommand": "iris start iris",
  "settings": {
    "terminal.integrated.defaultProfile.linux": "bash",
    "terminal.integrated.profiles.linux": {
      "bash": {
        "path": "bash",
        "icon": "terminal-bash"
      },
      "iris": {
        "path": "iris",
        "args": ["session", "iris"]
      }
    },
    "intersystems.servers": {
      "/ignore": true
    },
    "objectscript.ignoreInstallServerManager": true,
    "objectscript.ignoreInstallLanguageServer": true,
    "objectscript.conn": {
      "active": true,
      "host": "localhost",
      "port": 52773,
      "ns": "CONDUIT",
      "username": "demo",
      "password": "demo",
      "links": {
        "Conduit APP": "http://localhost:80/",
        "Conduit API": "http://${host}:${port}/conduit/"
      }
    }
  }
}
```

Hay varias cosas configuradas:

- La ruta a docker-compose.yml personalizado, en especial para Codespaces
- El nombre para el servicio principal, donde está el desarrollo
- La lista de extensiones instaladas por defecto en VSCode
- Los puertos que se deben publicar, en este caso, el puerto del servidor web para IRIS y para el frontend
- La ruta al directorio de trabajo
- El usuario dentro del contenedor, como estamos entrando en el contenedor de IRIS, necesitamos el usuario irisowner
- En docker-compose nuestro contenedor IRIS se configuró para no utilizar el entrypoint iris-main de forma predeterminada, sino simplemente suspenderlo con Infinity, y después de iniciar el entorno, debemos iniciar nuestro IRIS
- Y por último, la configuración de VSCode, también se puede ajustar aquí, es el nivel de configuración del equipo. Que por supuesto se puede anular o añadir con .vscode/settings.json

Poner en marcha Codespaces para dicho repositorio llevará un poco más de tiempo, ya que necesitará crear todos los contenedores necesarios e iniciarlos. GitHub afirma que será posible preconfigurar dichas imágenes después de realizar cualquier actualización en el repositorio, por lo que el inicio será más rápido.

Cuando comience, no se necesitarán más acciones, estará listo para el desarrollo

Este proyecto tiene una opción para probar la API REST con pruebas preparadas de Postman, así que he instalado npm y newman dentro del contenedor de backend con IRIS. Es posible realizar estas pruebas allí. Todo fue aprobado, bien hecho.

La parte del frontend está disponible

GitHub también permite conectar a Codespaces desde el VSCode local. Cuando se pulsa el área verde en la esquina de Codespaces, se puede elegir abrir en VS Code (deberá estar instalada la extensión GitHub Codespaces)

Y aquí está, el mismo proyecto, abierto con tu VSCode local, pero que se ejecuta en la nube, como se puede ver el resultado de ifconfig, no estoy en Singapur en este momento :)

En el navegador pero sin los Codespaces de GitHub

Qué pasa si no tienes acceso a la funcionalidad de Codespaces o no quieres usarla de esta manera, pero aun así te gustaría probar VSCode en el navegador.

Bien, es posible con otro proyecto code-server

Basta con ejecutar este VSCode con este comando

docker run -it -p 8080:8080 codercom/code-server --auth=none

Se ejecutará la versión predeterminada de VSCode, sin carpetas mapeadas en su interior. Simplemente organiza cualquier carpeta, y establécela como el directorio de trabajo workdir y la verás en su interior.

docker run -it -p 8080:8080 -v ̀pwd:̀/opt/realworld -w /opt/realworld codercom/code-server --auth=none

Es el VSCode predeterminado, sin la extensión instalada de ObjectScript. Tiene una limitación en cuanto a las extensiones, no tiene acceso al *marketplace* original de VSCode, sino que utiliza otro sitio, open-vsx.org, y la extensión principal de ObjectScript también está disponible allí.

Con un Dockerfile como este podríamos hacer nuestro propio Code Server, con cualquier cosa instalada allí, así

como algunas extensiones ya instaladas

```
FROM codercom/code-server

USER root

RUN curl -fsSL https://deb.nodesource.com/setup_15.x | bash - && \
    apt-get install -y jq nodejs python3-pip python3-dev unixodbc-dev && \
    rm -rf /var/lib/apt/lists/* && \
    pip3 install pyodbc && \
    npm install -g yarn && \
  sudo chown -R 1000:1000 /home/coder

COPY extensions extensions

COPY settings.json /root/.local/share/code-server/User/settings.json

ENV SERVICE_URL=https://open-vsx.org/vscode/gallery
ENV ITEM_URL=https://open-vsx.org/vscode/item

RUN \
  code-server --install-extension ms-python.python && \
  code-server --install-extension intersystems-community.vscode-objectscript && \
  find extensions -type f -exec code-server --install-extension {} \;

WORKDIR /opt/intersystems

CMD [ "--auth=none", "--disable-telemetry" ]
```

Puedes definir algunos ajustes predeterminados settings.json para el nivel de usuario y si algunas extensiones que necesita no están disponibles en open-vsx, descárgalas manualmente, colócalas en la carpeta de extensiones junto a Dockerfile, y también las tendrá instaladas.

Ahora puedes ejecutar el nuevo code-server con todas las extensiones instaladas que necesites

```
docker run -it -p 8080:8080 -v `pwd`:/opt/realworld -w /opt/realworld &lt;strong>care
tdev/code-server&lt;/strong> --auth=none
```

Además, ya hay una sintaxis resaltada, lo único que falta es ejecutar el propio IRIS. Esto se puede hacer con docker-compose ampliado, donde el code-server será como un servicio más junto a IRIS

#Entorno de desarrollo #Mejores prácticas #InterSystems IRIS #VSCode

---

URL de fuente:https://es.community.intersystems.com/post/codespaces-de-github-con-iris