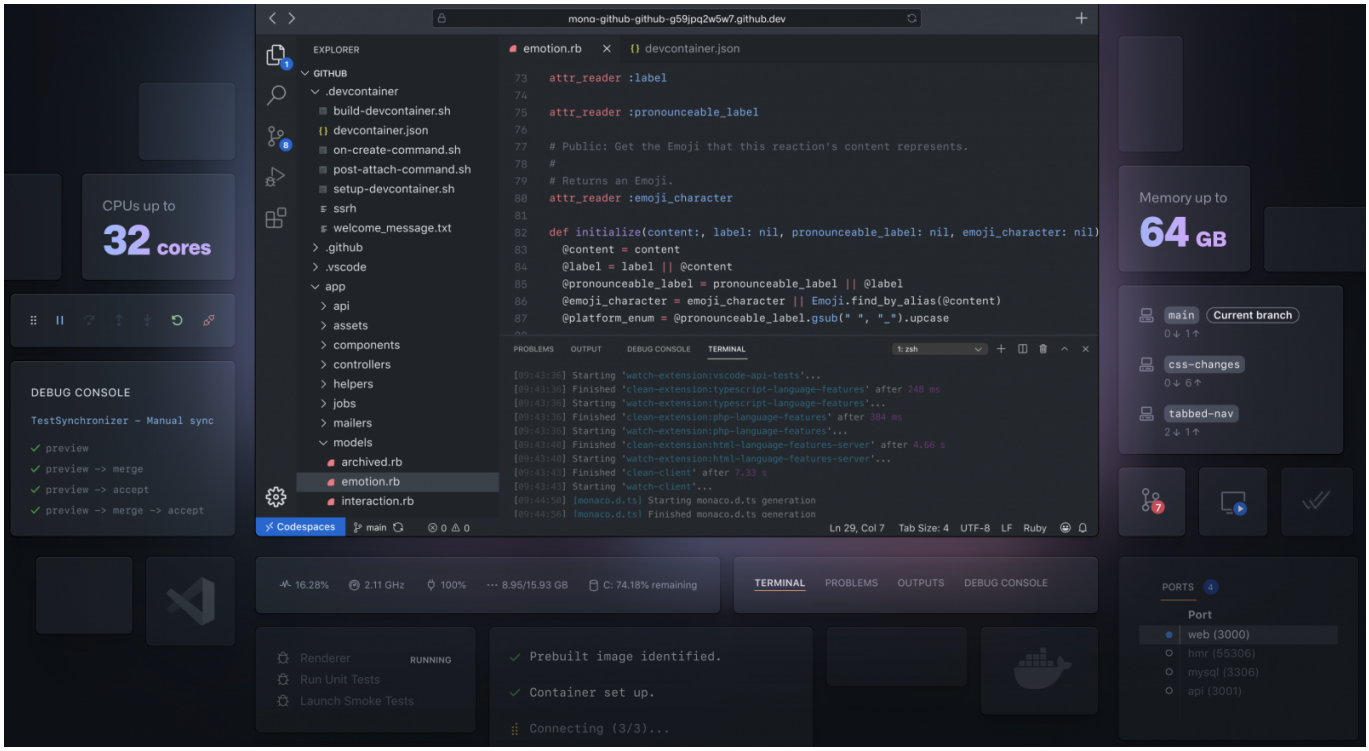
Article
Dmitry Maslennikov · Aug 20, 2021 6m read

GitHub Codespaces with IRIS

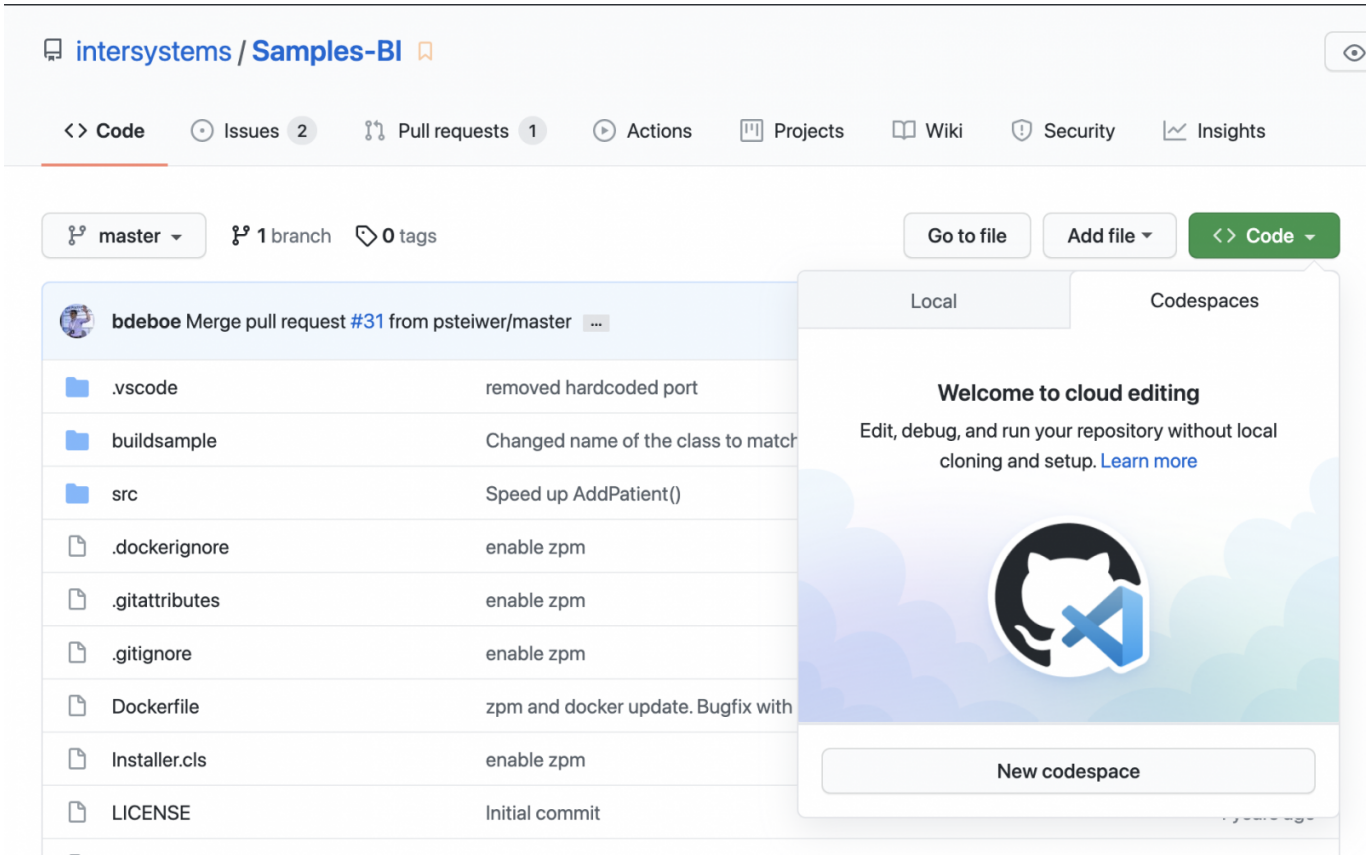
Some time ago GitHub, has announced the new feature, GitHub Codespaces. It gives an ability to run VSCode in the browser, with almost the same power as it would run locally on your machine, but also with a power of clouds, so, you are able to choose the machine type with up to 32 CPU cores and 64 GB of RAM.



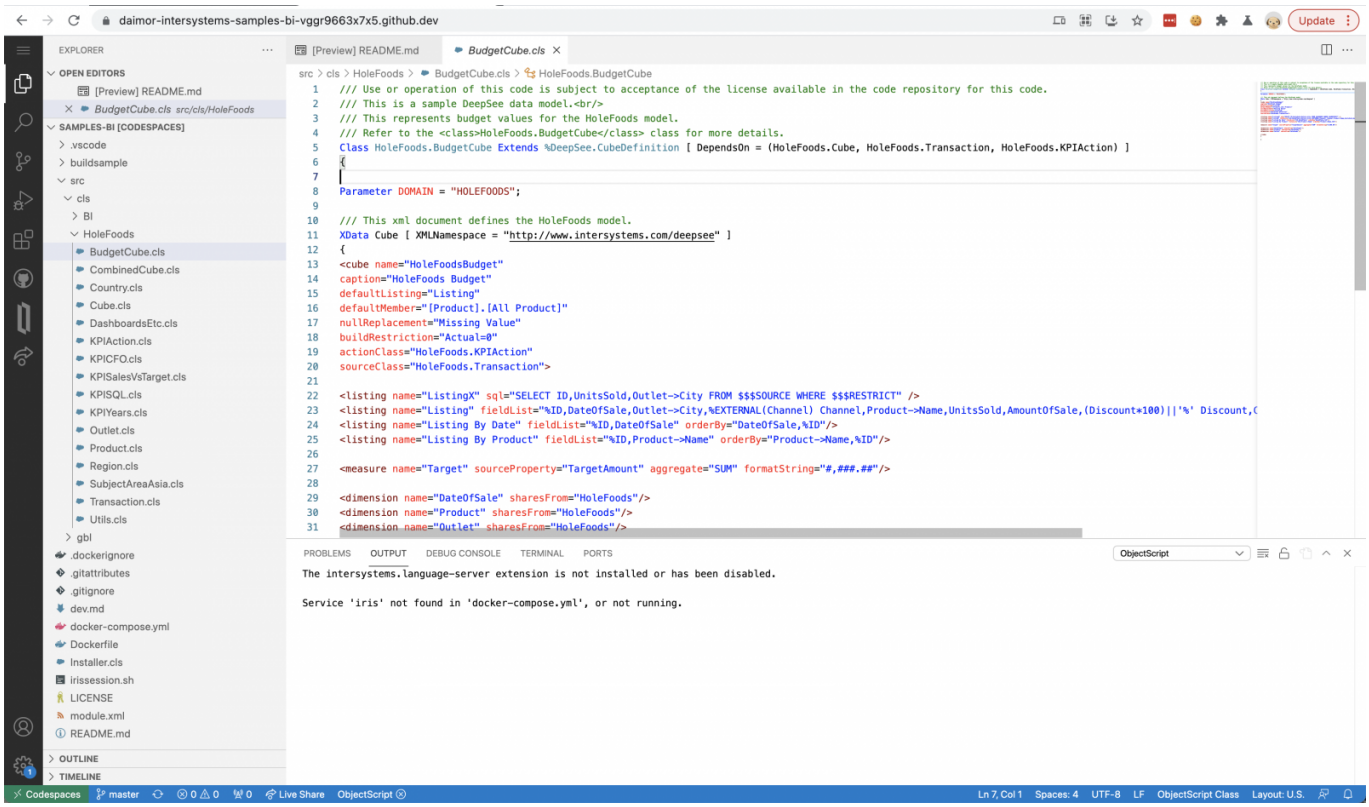
Looks impressive, is not it? But how it could help us, to work with projects driven by InterSystems IRIS? Let's have a look, how to configure it for us.

Simple start for any repo

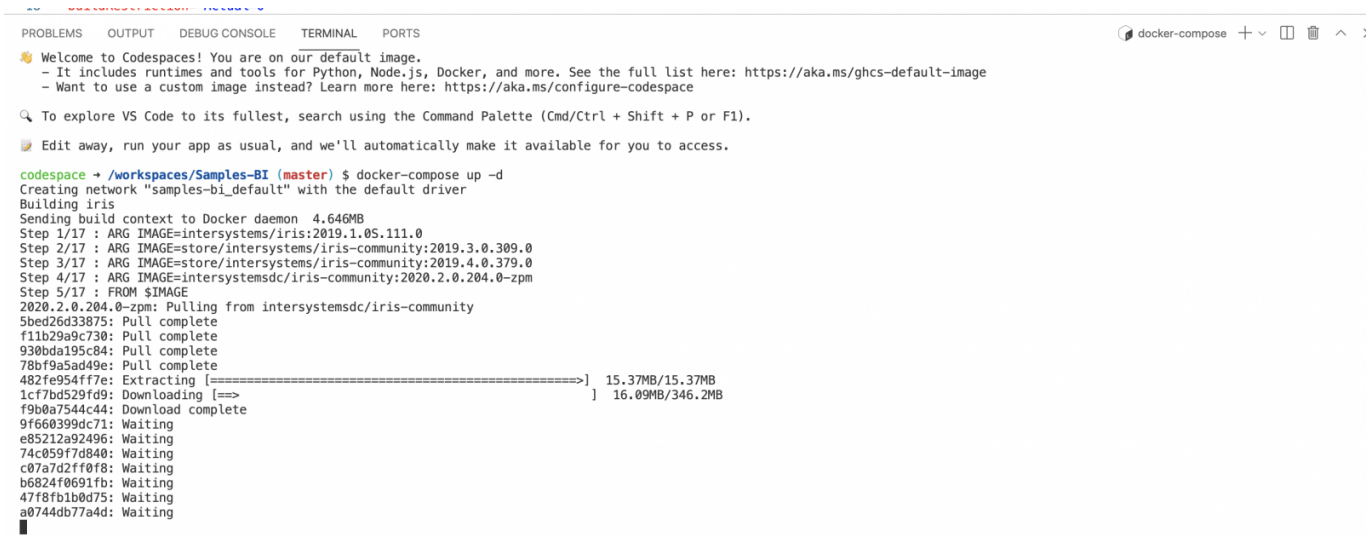
With this feature, you'll be able to edit any repository in the cloud, by Code button. Please note, that this feature is still in beta, and may not be available for everyone, and after a beta period, will be available only for paid accounts.



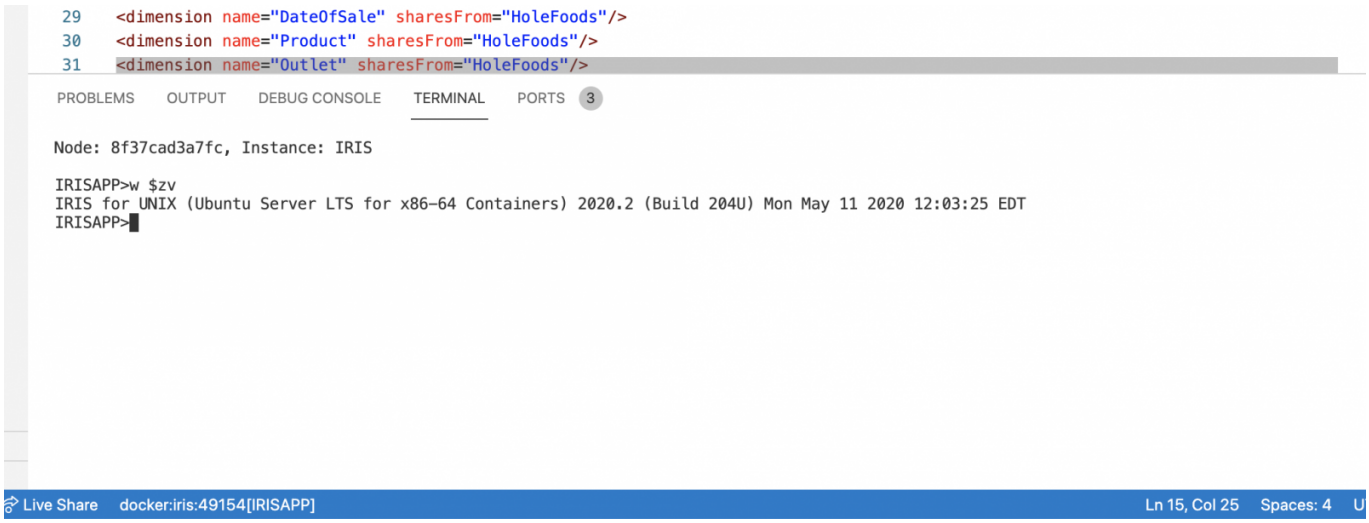
So, this, repository has not been specially prepared for Codespaces at all, just for VSCode. Press New codespace button, to create an environment just for you.



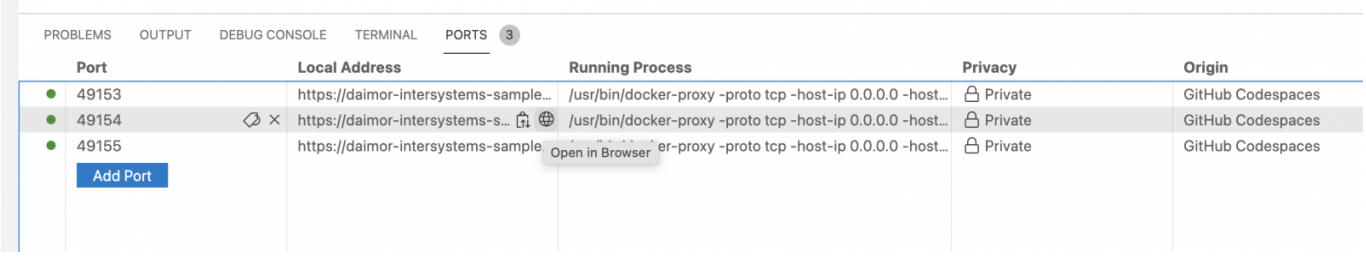
In my case, in the previous usage of codespaces, I've already installed the ObjectScript extension and enabled it globally by prompt from Codespaces. So, it installs it to me every time, and ObjectScript code already highlighted. But thee IRIS there is not available, yet. Let's start it with docker-compose.



After that, now we are able to connect to IRIS Terminal, and compile the code



Ports from docker-compose are automatically recognized and can be opened in the browser, so, it's also possible to open System Management Portal



Usage with prepared repository

We managed, to run VSCode and IRIS in the cloud, but we had to do some things manually, to get get it ready. But, it's also possible to make your repository ready for development right after the start.

It is possible, with devcontainer.json. I will do an example based on one of my recent projects Realworld. This

project is quite complex, it has a backend and frontend and uses docker-compose to start all it together.

devcontainer may use docker-compose as well, so, my config appeared to be like this.

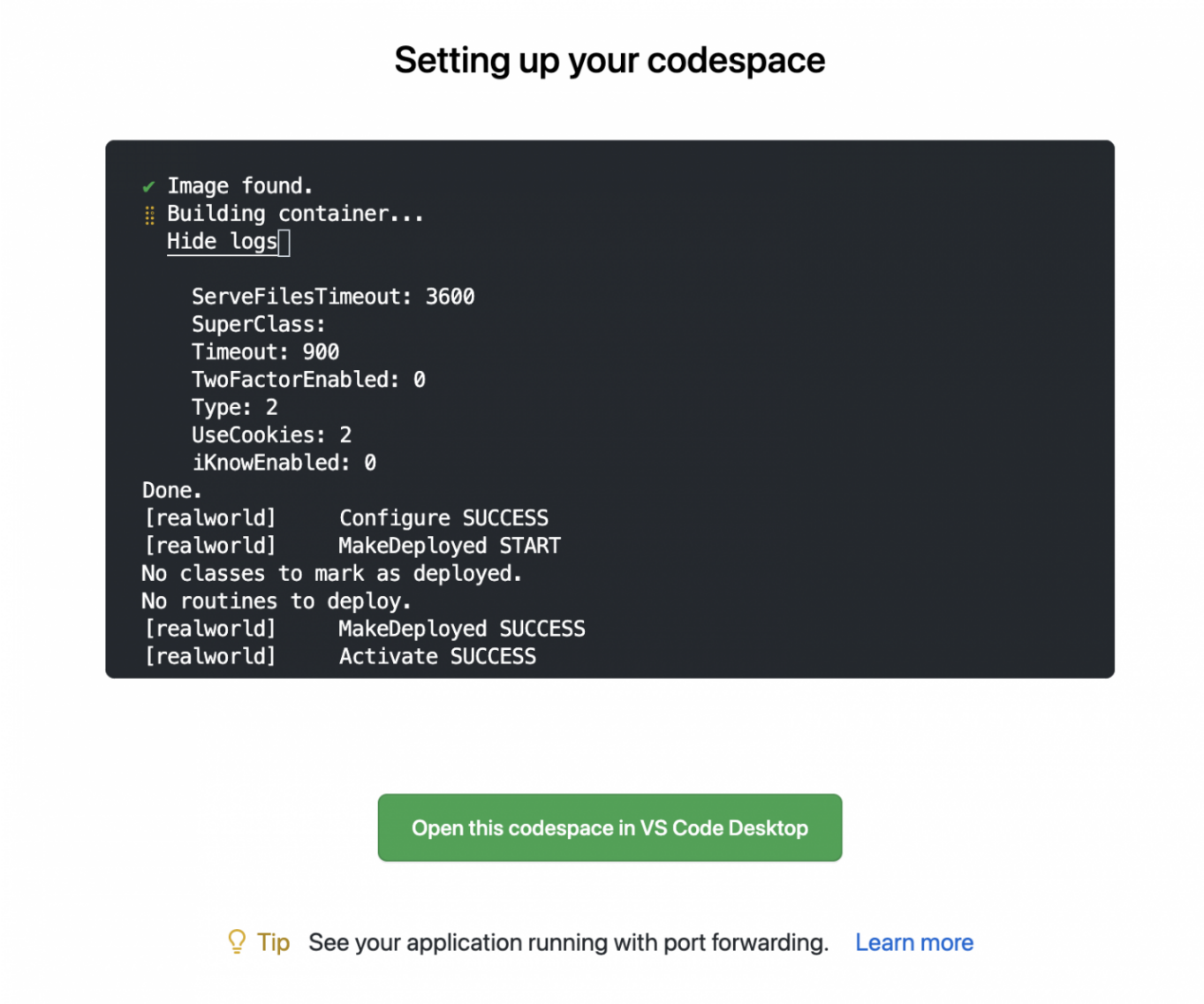
```
{
  "name": "IRIS RealWorld example",
  "dockerComposeFile": "docker-compose.yml",
  "service": "server",
  "extensions": [
    "intersystems-community.vscode-objectscript"
  ],
  "forwardPorts": [
    80,
    52773
  ],
  "workspaceFolder": "/home/irisowner/conduit",
  "remoteUser": "irisowner",
  "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/"
      }
    }
  }
}
```

There are many things configured

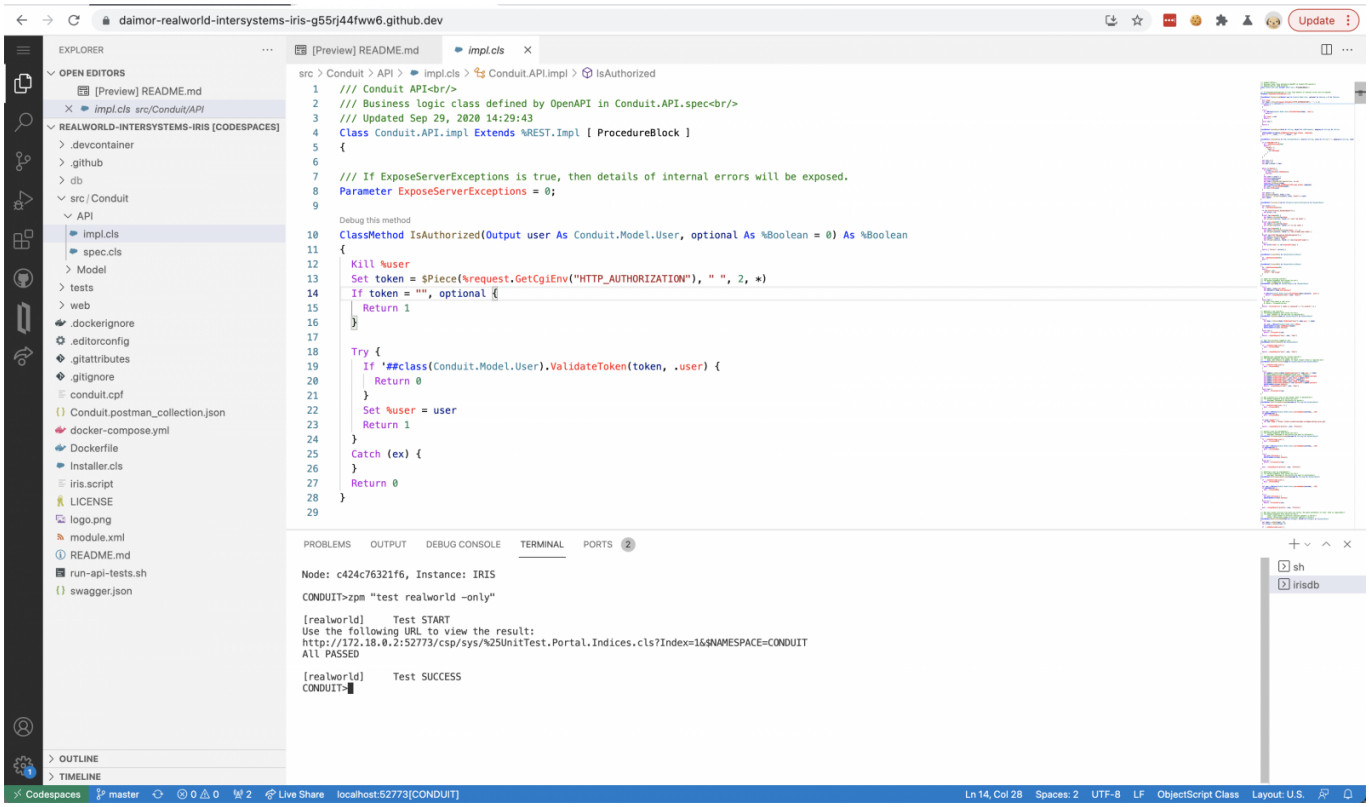
- path to custom docker-compose.yml, especially for Codespaces
- name for the main service, where the development
- list of extensions installed by default in VSCode
- ports which have to be published, in this case, web server port for IRIS and for frontend

- path to working directory
- user inside the container, as we are entering IRIS container, we need user irisowner
- in docker-compose our IRIS container confiigured to not use default entrypoiny iris-main, but just sleep with with infinity, and after start the environment we have to start our IRIS
- and finally, the settings for VSCode, also can be configured here, it's a machine level of settings. Which for sure can be overrided or appended with .vscode/settings.json

Starting Codespaces for such repository, will take a bit more time, as it will needs to build all the necessary containers and start them. GitHub says, that it will be possible to prebake such images after any push to the repo, so, such start will be faster.



And when it started, no more actions needed, it's ready for development



This project has an option to test REST API with prepared Postman tests, so, I've installed npm and newman inside the backend container with IRIS. And it's possible to run this tests there. All passed, well done.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2

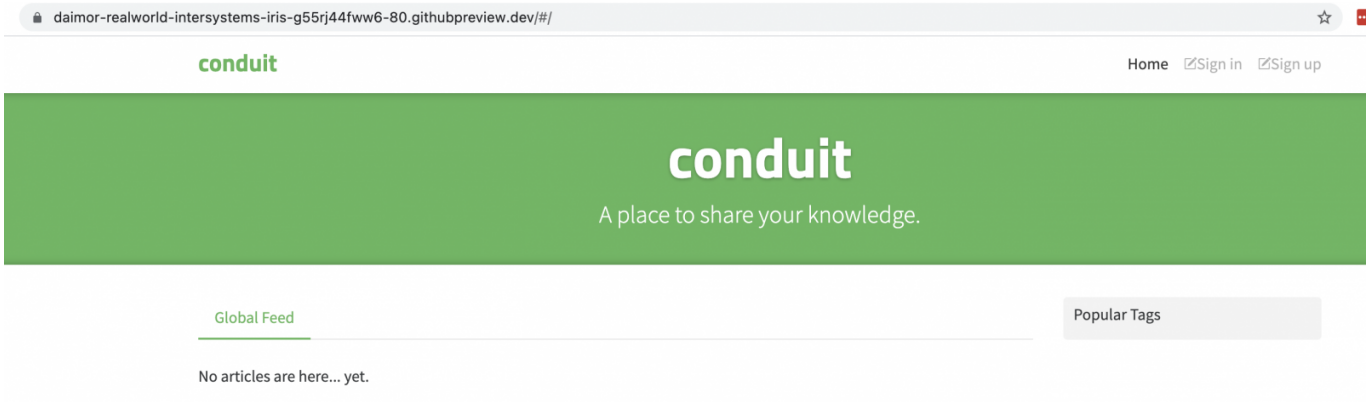
```
  ✓  Response contains "profile" property
  ✓  Profile has "username" property
  ✓  Profile has "bio" property
  ✓  Profile has "image" property
  ✓  Profile has "following" property
  ✓  Profile's "following" property is false

❑ Tags
↳ All Tags
  GET http://localhost:52773/conduit/tags [200 OK, 282B, 6ms]
  ✓  Response code is 200 OK
  ✓  Response contains "tags" property
  ✓  "tags" property returned as array
```

	executed	failed
iterations	1	0
requests	31	0
test-scripts	46	0
prerequest-scripts	17	0
assertions	280	0
total run duration: 16.8s		
total data received: 7.8KB (approx)		
average response time: 18ms [min: 5ms, max: 110ms, s.d.: 26ms]		

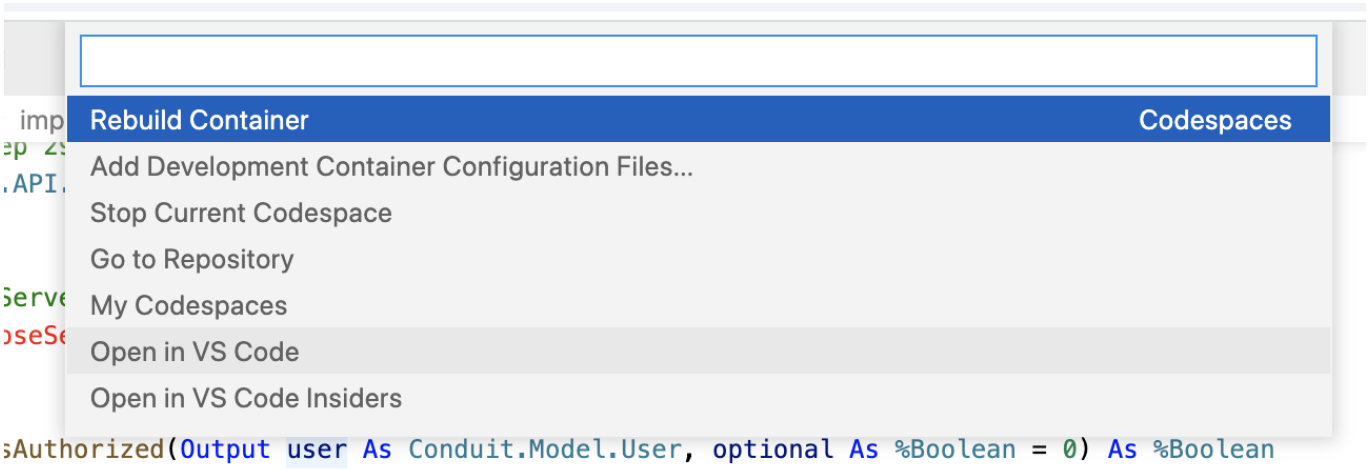
```
irisowner@c424c76321f6:/workspaces/realworld-intersystems-iris$
```

Live Share localhost:52773[CONDUIT]

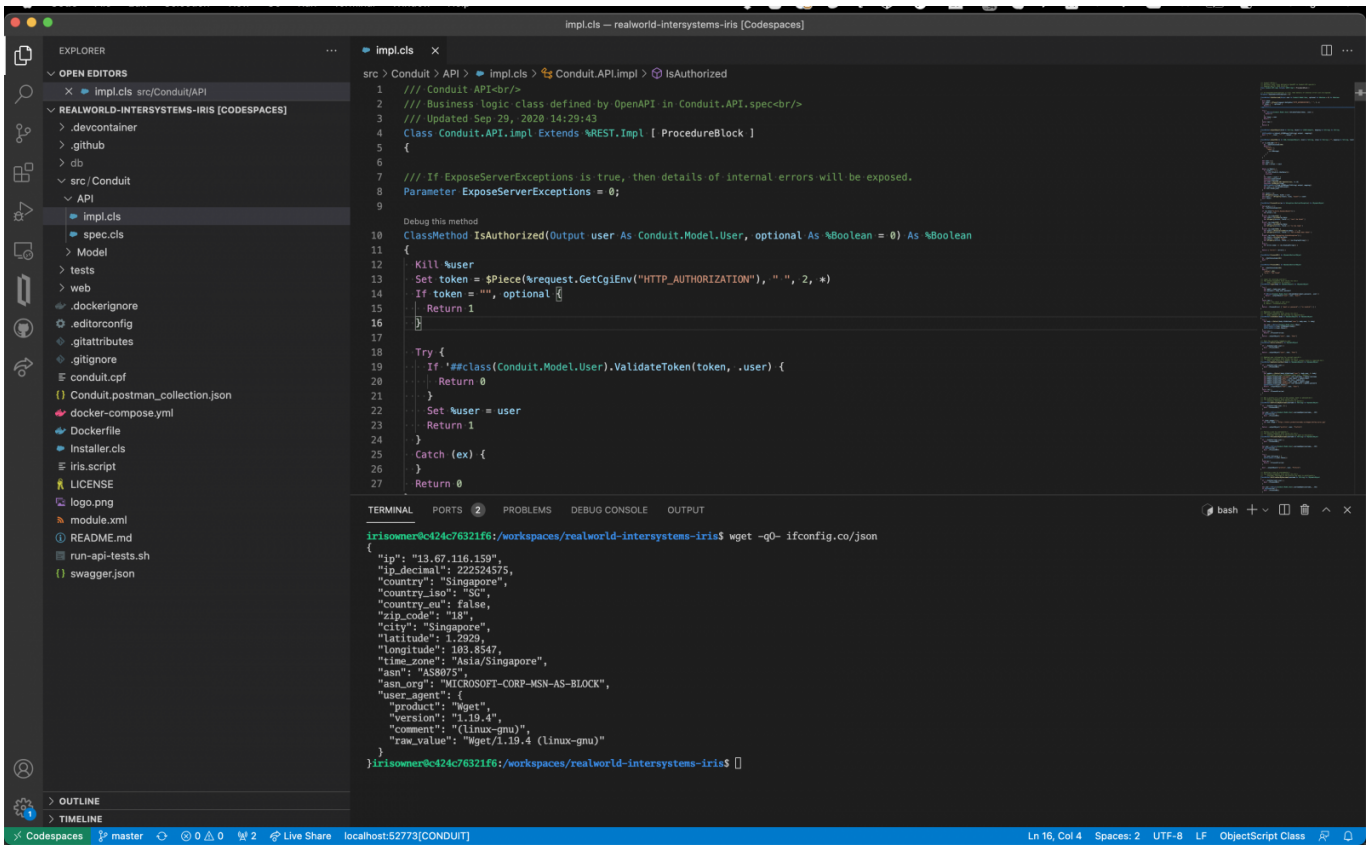
And frontend part is available



GitHub allows to connect to Codespaces, from the local VSCode as well. When you press by green Codespaces in the corner, you may choose to open in VC Code (GitHub Codespaces extension have to be installed)



And here it is, it's the same project, opened with your local VSCode, but running in the cloud, as you may see the result of ifconfig, I'm definetly not in Singapore, right now.



In browser but without GitHub Codespaces

What about if you don't have access to Codespaces feature, or don't want to use it by this way, but still would like to try VSCode in the browser.

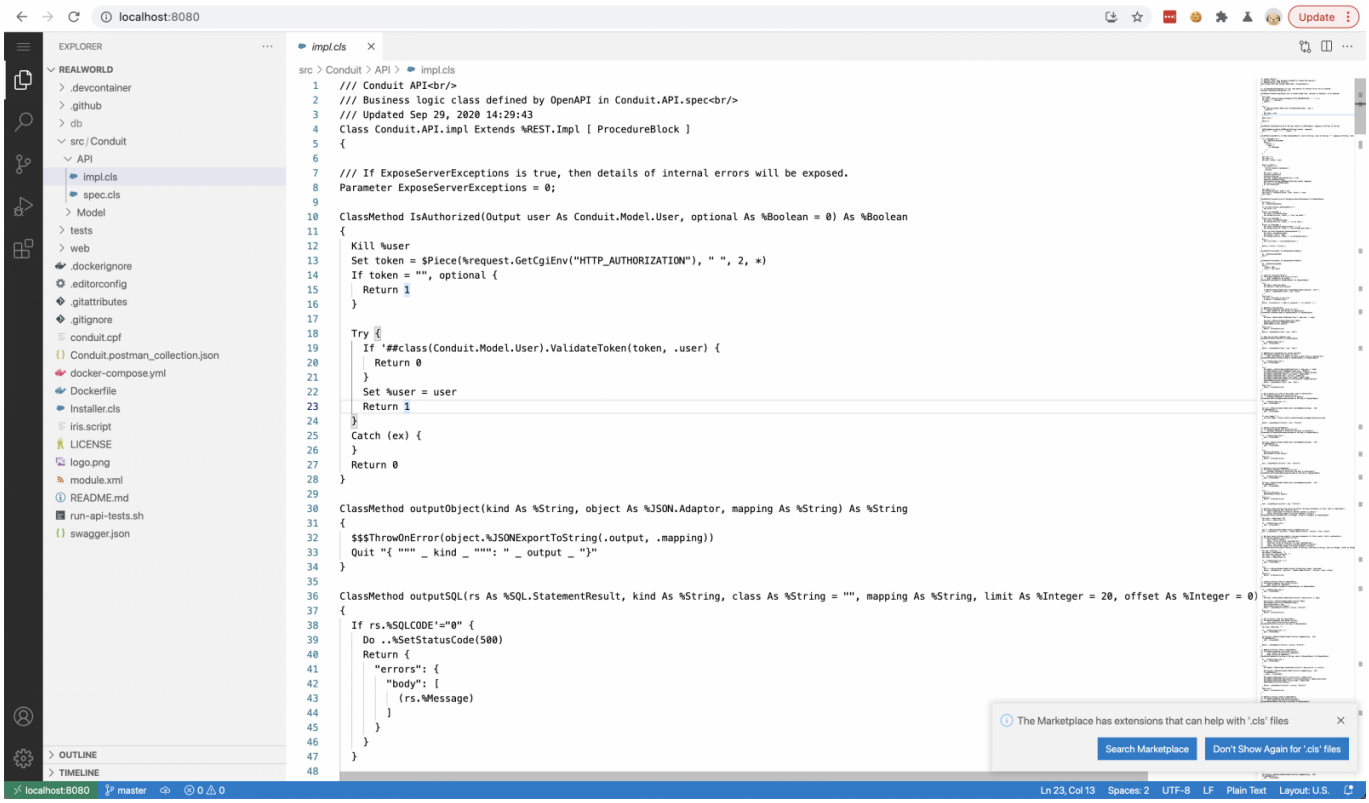
Well, it's possible with another project [code-server]

You can simply run this VSCode with this command

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

It will run, the default version of VSCode, with no folders mapped inside, just mount any folder, and set it as the workdir, and you will see it inside.

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



It's default VSCode, with no ObjectScript Extension installed. It has a limitation with extensions, it does not have access to original VSCode marketplace, instead it uses another place, open-vsx.org, and the main ObjectScript extension is avalable there as well.

With Dockerfile like this, we can bake our own Code Server, with anything installed there, as well as some extensions already installed

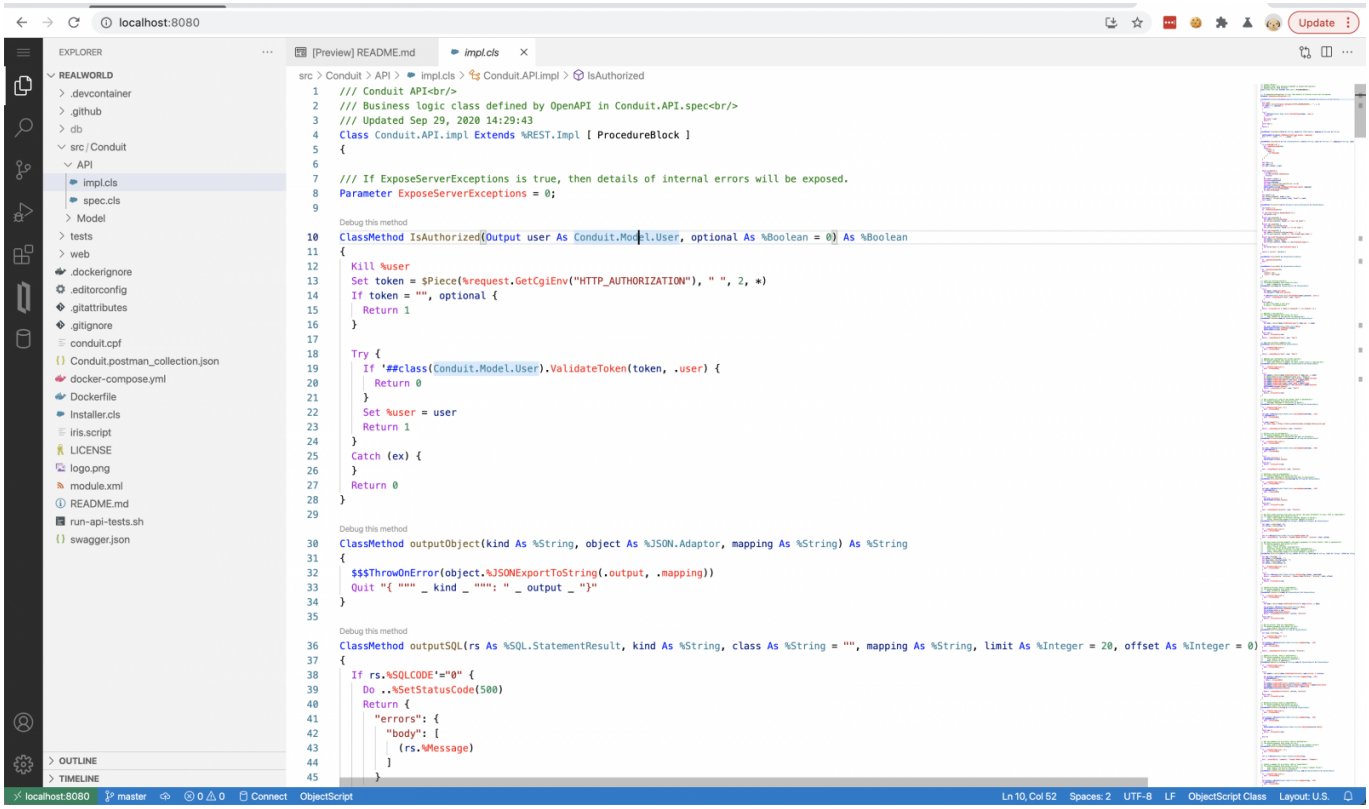
```
FROM codercom/code-server

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" ]
```

You may define, some default settings.json for user-level and if some extensions you need not available on open-vsx, download them manually, place them to extensions folder next to Dockerfile, and you'll get them installed as well.

Now you are able to run new code-server with all extensions you need installed

```
docker run -it
-p 8080:8080 -v `pwd`:/opt/rea
lworld -w /opt/realworld caretdev/code-server --auth=none
```

And Syntax highlighting already there, the only thing is left, to run IRIS itself, and it can be done with extended docker-compose, where code-server will be just as another service next to IRIS



#Best Practices #Development Environment #InterSystems IRIS #VSCode

Source URL:https://community.intersystems.com/post/github-codespaces-iris