

qc_dev - Quantum Computing Development Image

This image supports the use of a Docker container for the development of Quantum Computing projects in an Ubuntu environment.

Table of Contents

1. Installed core components
 2. Creating a new Quantum Computing development container
 3. Working with an existing Quantum Computing development container
 4. Best practices
-

1. Installed core components

With the following command you can check in detail which software versions are included in the Docker image:

```
apt list --installed
```

Version 1.0.1

Component	Version	Remark	Status
asdf	v0.8.1-a1ef92a		
dos2unix	7.4.0		
G++	9.3.0		
GCC	9.3.0		
Git	2.25.1		
GNU Autoconf	2.69		
GNU Automake	1.16.1		
GNU Make	4.2.1		
htop	2.2.0		
openssl	1.1.1f		
Python	3.9.6		
- jupyterlab	3.1.4	only with vm	
- notebook	6.4.2	only with vm	
- pip	21.1.3		

Component	Version	Remark	Status
Ubuntu	20.04.2 LTS	focal	
Vim	8.2.2269		
wget	1.20.3		

Version 1.0.0

Component	Version	Remark	Status
asdf	v0.8.1-a1ef92a		
cURL	7.77.0		
Docker Compose	1.29.2		
Docker Engine	20.10.7		
dos2unix	7.4.2		
GCC	10.3.0		
Git	2.32.0		
htop	3.0.5		
Python	3.9.6		
- pip	21.1.3		
Ubuntu	20.04.2 LTS	focal	
Vim	8.2.3083		
wget	1.21.1		

2. Creating a new Quantum Computing development container

2.1 Getting started

```
> REM Assumptions:
> REM   - the name of the Docker container should be: my_qc_dev
> REM   - the path the host repository is: //C/projects/my_repro
> REM   - the directory name for this repository inside the container should be:
my_repro_dir
> REM   - you want to use the latest version of the Konnexions development image
> docker run --name my_qc_dev \
    -v //C/projects/my_repro:/my_repro_dir \
    konnexionsgmbh/qc_dev:latest

> REM Stopping the container
> docker stop my_qc_dev
```

```
> REM Restarting the container
> docker start my_qc_dev

> REM Entering a running container
> docker exec -it my_qc_dev bash
```

2.2 Detailed Syntax

A new container can be created with the **docker run** command.

Syntax:

```
docker run -it
    [--name <container_name>] \
    konnexionsgmbh/qc_dev[:<version>]
    [<cmd>]
```

Parameters:

- **container_name** - an optional container identification
- **directory_repository** - an optional host repository directory - the default value is expecting the repository inside the container
- **version** - an optional version number of the image or the constant **latest**
- **cmd** - an optional command to be executed in the container, default is **bash** for running the **bash** shell

Detailed documentation for the command **docker run** can be found [here](#).

Examples:

1. Creating a new Docker container named **my_qc_dev** using a repository inside the Docker container:

```
docker run -it --name my_qc_dev konnexionsgmbh/qc_dev:latest
```

2. Creating a new Docker container named **my_qc_dev** using the host repository of a Windows directory **D:\projects\my_repro**:

```
docker run -it --name my_qc_dev -v //D/projects/my_repro:/my_repro
konnexionsgmbh/qc_dev:latest
```

3. Creating a new Docker container named **my_qc_dev** using the host repository of a Linux directory **/my_repro**:

```
docker run -it --name my_qc_dev -v /my_repro:/my_repro konnexionsgmbh/qc_dev:latest
```

3. Working with an existing Quantum Computing development container

3.1 Starting a stopped container

A previously stopped container can be started with the **docker start** command.

Syntax:

```
docker start <container_name>
```

Parameter:

- **container_name** - the mandatory container identification, that is an UUID long identifier, an UUID short identifier or a previously given name

Detailed documentation for the command **docker start** can be found [here](#).

3.2 Entering a running container

A running container can be entered with the **docker exec** command.

Syntax:

```
docker exec -it <container_name> <cmd>
```

Parameter:

- **container_name** - the mandatory container identification, that is an UUID long identifier, an UUID short identifier or a previously given name
- **cmd** - the command to be executed in the container, e.g. **bash** for running the **bash** shell

Detailed documentation for the command **docker exec** can be found [here](#).

4. Best practices

4.1 Use of a root repository directory on the host computer

If all relevant repositories are located within a common parent directory, then development work in all these repositories can be done within a single Konnexions development container.

Example:

In the following example we assume that the host directory is named **C:\Temp\my_projects** and should be mapped to the **projects** directory in the container.

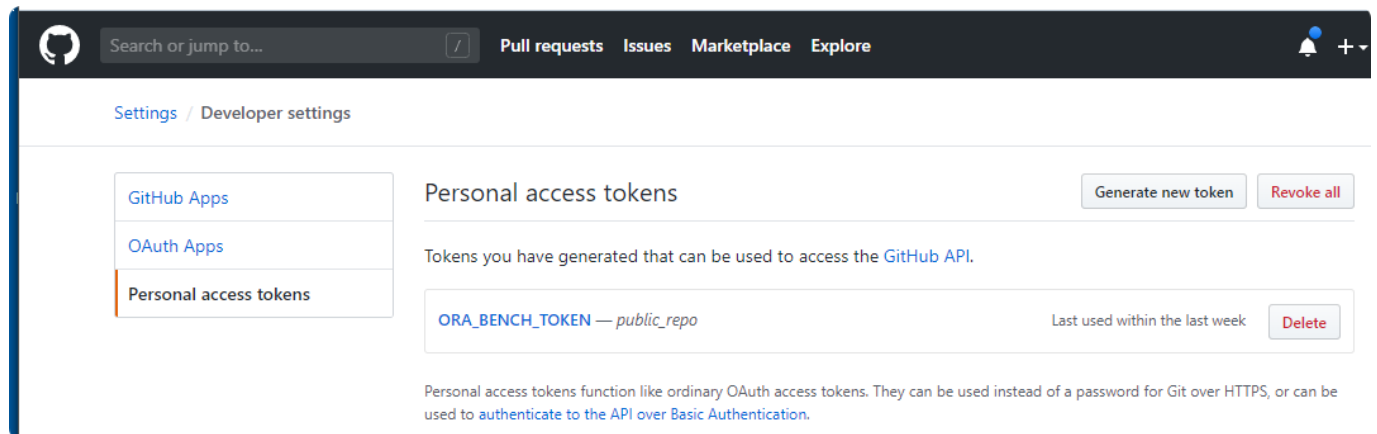
```
>C:\Temp\my_projects>docker run -it --name qc_dev -v
//C:/Temp/my_projects:/projects konnexionsgmbh/qc_dev:latest
root@35b9310932f1:/# cd projects
root@35b9310932f1:/projects# ls -ll
total 0
```

4.2 Use of private GitHub repositories inside the container

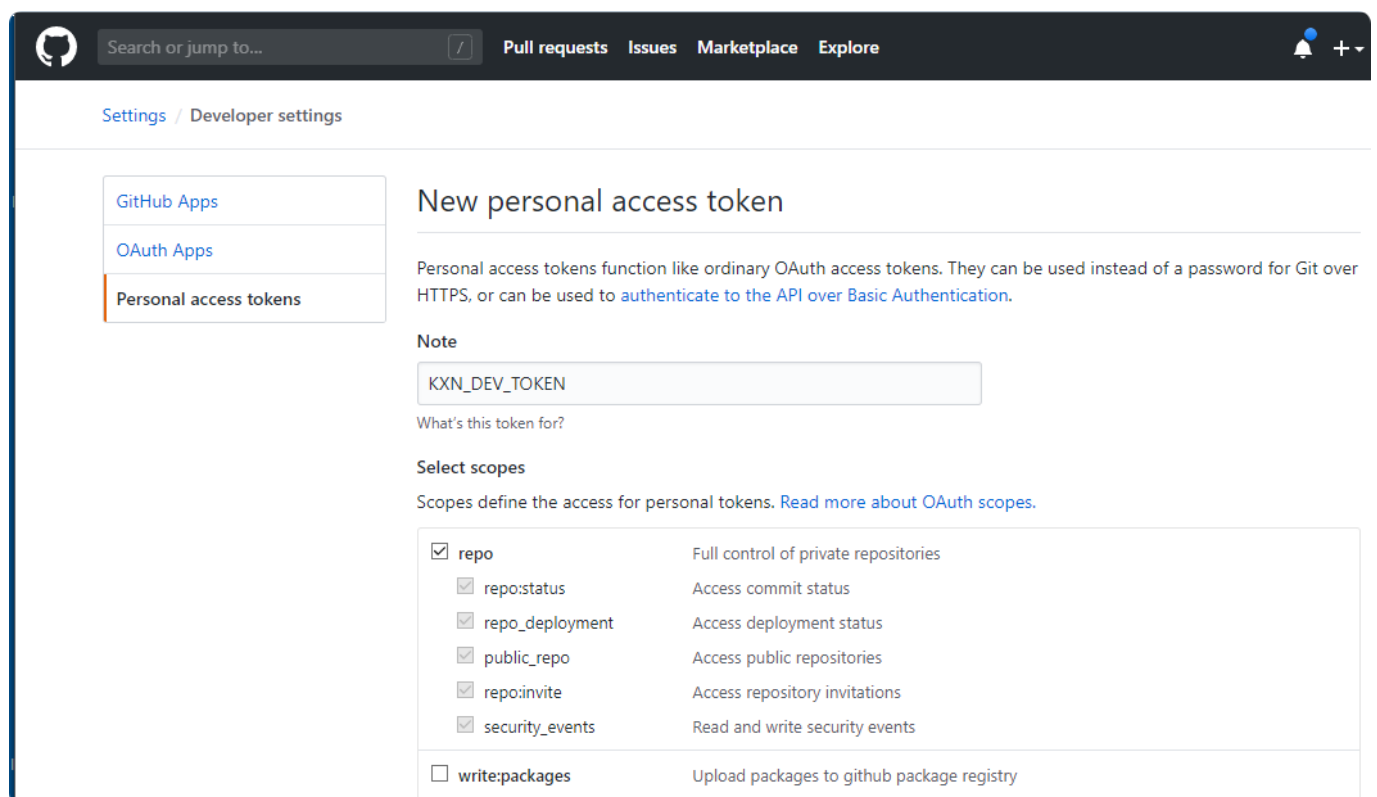
To access private repositories in GitHub, you must first create a new personal access token in GitHub and then add it to your git configuration inside the container.

1. Create a new personal access token in GitHub

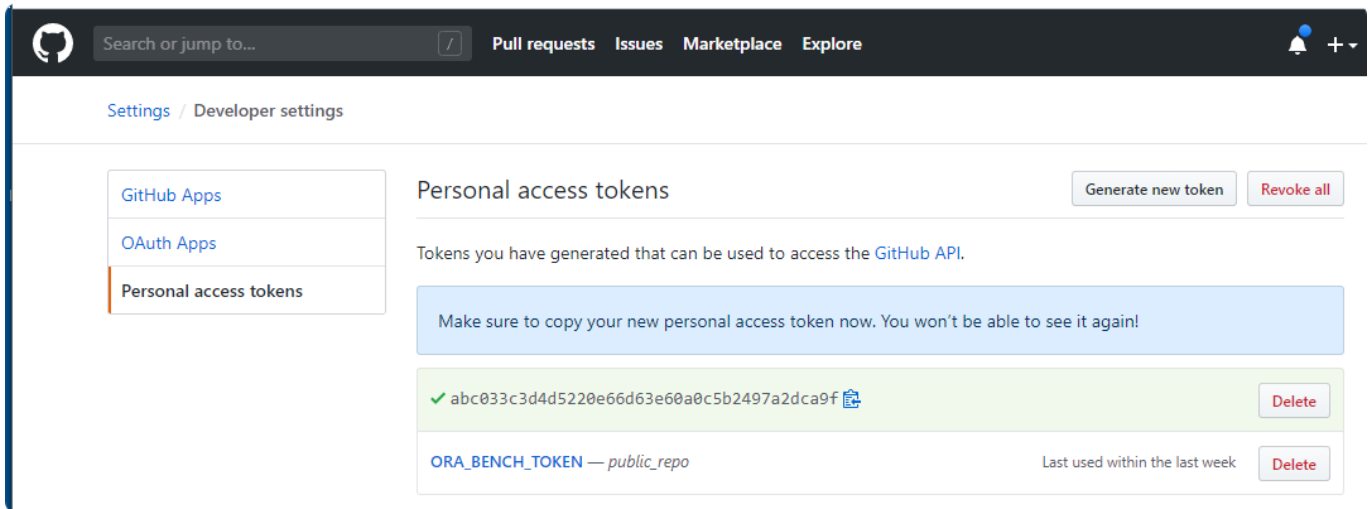
- With the following URL you can create the access token: <https://github.com/settings/tokens>



- Press the button **Generate new token**



- Name the new token, select the scopes and press the button **Generate token**



- Write down the secret code and keep it in a safe place

2. Setting up the Docker container on the host machine

In the following example we assume that the host directory is named `C:\Temp\my_projects` and should be mapped to the `projects` directory in the container.

```
C:\Temp\my_projects>docker run -it --name qc_dev -v //C/Temp/my_projects:/projects
konnexionsgmbh/qc_dev:latest
Unable to find image 'konnexionsgmbh/qc_dev:latest' locally
latest: Pulling from konnexionsgmbh/qc_dev
d51af753c3d3: Pull complete
...
a6bb30d1a5cf: Pull complete
Digest: sha256:5f6d6afc566ef9142d2d85b85dd331c0558eafaaf286179fd0ae787988c1b89b
Status: Downloaded newer image for konnexionsgmbh/qc_dev:latest
```

3. Initial configuration of git in the container

```
root@332206c300f1:/# export XDG_CONFIG_HOME=/projects
root@332206c300f1:/# mkdir -p $XDG_CONFIG_HOME/git/
root@332206c300f1:/# touch $XDG_CONFIG_HOME/git/config
root@332206c300f1:/# touch $XDG_CONFIG_HOME/git/credentials
root@332206c300f1:/# git config --file=$XDG_CONFIG_HOME/git/config
credential.helper 'store --file=/projects/git/credentials'
root@332206c300f1:/# git config --file=$XDG_CONFIG_HOME/git/config user.name "John
Doe"
root@332206c300f1:/# git config --file=$XDG_CONFIG_HOME/git/config user.email
"john.doe@company.com"
root@332206c300f1:/# git config --list --show-origin
file:/projects/git/config      credential.helper=store --
file=/projects/git/credentials
file:/projects/git/config      user.name=John Doe
file:/projects/git/config      user.email=john.doe@company.com
```

4. Verification of the settings

```
root@332206c300f1:/# cat /projects/git/config
[credential]
    helper = store --file=/projects/git/credentials
[user]
    name = John Doe
[user]
    email = john.doe@company.com
```

5. Clone a repository for the first time

When prompted provide your github user name and the new personal access token from (1).

```
root@332206c300f1:/# cd projects
root@332206c300f1:~# git clone https://github.com/KonnexionsGmbH/docker_images
Cloning into 'docker_images'...
Username for 'https://github.com': John Doe
Password for 'https://john.doe@company.com':
abc033c3d4d5220e66d63e60a0c5b2497a2dca9f
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 78 (delta 33), reused 68 (delta 23), pack-reused 0
Receiving objects: 100% (78/78), 167.83 KiB | 867.00 KiB/s, done.
Resolving deltas: 100% (33/33), done.
```

6. Verify if the clone completed with success

```
root@332206c300f1:~# cat /projects/git/credentials
https://John Doe:abc033c3d4d5220e66d63e60a0c5b2497a2dca9f@github.com
```

7. Verification after a restart of the Docker container

```
C:\Temp\my_projects>docker start qc_dev
qc_dev
C:\Temp\my_projects>docker exec -it qc_dev bash
root@332206c300f1:/# export XDG_CONFIG_HOME=/projects
root@332206c300f1:/# git config --list --show-origin
file:/projects/git/config credential.helper=store --
file=/projects/git/credentials
file:/projects/git/config user.name=John Doe
file:/projects/git/config user.email=john.doe@company.com
```

8. Verification after the removal of the Docker container

- Deleting the Docker container and image

```
C:\Temp\my_projects>docker stop qc_dev
qc_dev

C:\Temp\my_projects>docker rm qc_dev
qc_dev

C:\Temp\my_projects>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
konnexionsgmbh/qc_dev latest             51757b5e414e       6 hours ago
3.71GB

C:\Temp\my_projects>docker rmi 51757b5e414e
Untagged: konnexionsgmbh/qc_dev:latest
Untagged:
konnexionsgmbh/qc_dev@sha256:5f6d6afc566ef9142d2d85b85dd331c0558eafaaf286179fd0ae7
87988c1b89b
Deleted: sha256:51757b5e414e5333ace7b163484c06e4685c29312ad09d5d7d648c6936011a60
...
Deleted: sha256:7789f1a3d4e9258fbc5469a8d657deb6aba168d86967063e9b80ac3e1154333f
```

- Recreating the Docker container (and image)

```
C:\Temp\my_projects>docker run -it --name qc_dev -v //C/Temp/my_projects:/projects
konnexionsgmbh/qc_dev:latest
Unable to find image 'konnexionsgmbh/qc_dev:latest' locally
latest: Pulling from konnexionsgmbh/qc_dev
d51af753c3d3: Pull complete
...
a6bb30d1a5cf: Pull complete
Digest: sha256:5f6d6afc566ef9142d2d85b85dd331c0558eafaaf286179fd0ae787988c1b89b
Status: Downloaded newer image for konnexionsgmbh/qc_dev:latest
root@ad1f036bbc44:/# export XDG_CONFIG_HOME=/projects
root@ad1f036bbc44:/# git clone https://github.com/KonnexionsGmbH/docker_images
Cloning into 'docker_images'...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 78 (delta 33), reused 68 (delta 23), pack-reused 0
Receiving objects: 100% (78/78), 167.83 KiB | 895.00 KiB/s, done.
Resolving deltas: 100% (33/33), done.
```

- If we use the same path - where `git/config` and `git/credentials` exist - as in Step 4, `git` access (clone/push/pull) doesn't ask for username/password anymore.