

[Get started](#)[Start your workspace](#)

Start your workspace

This page will help you to start a workspace to run your code and experiments on the DSRI in a container.

Introduction to containers

Anything running in DSRI needs to be running in a docker container. Docker containers are namespaces that share the kernel on a linux system, you can see them as a clean minimalist Linux computers with only what you need to run your programs installed. This allows you to completely control the environment where your code runs, and avoid conflicts.

When running experiments we can start from existing images that have been already published for popular data science applications with a web interface. You can use, for example, JupyterLab when running python, RStudio when running R, or VisualStudio Code if you prefer.

Once you access a running container, you can install anything you need like if it was a linux/ubuntu computer (most of them runs with admin privileges), and run anything via the notebook/RStudio /VSCode interface, or the terminal.

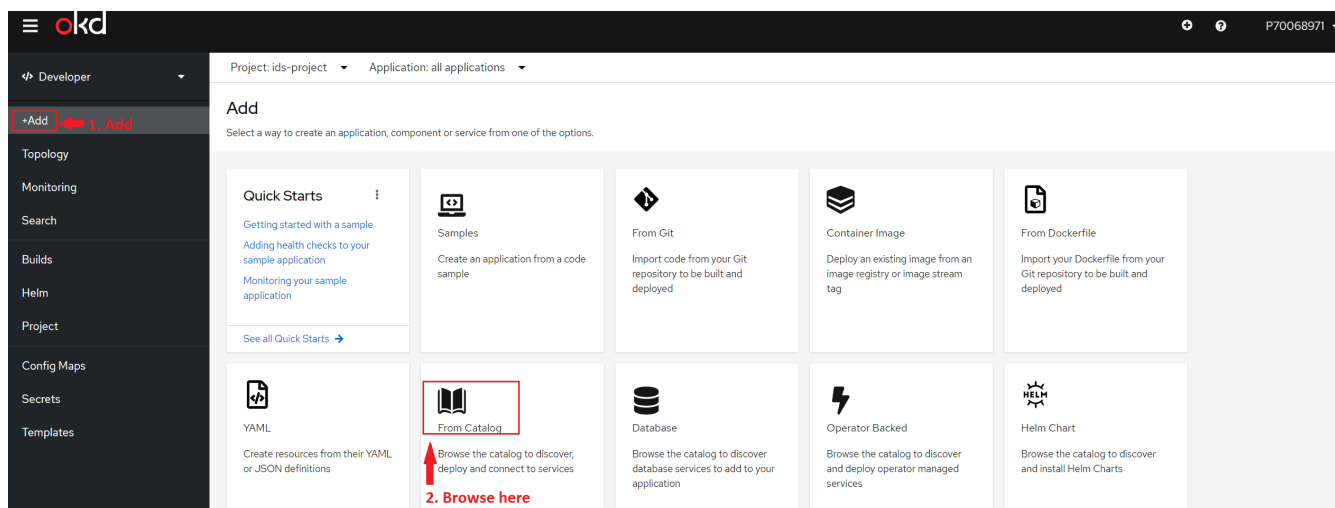
Choose your interface

First step to get your code running on the DSRI is to pick the base interface you want to use to access your workspace on the DSRI.

We prepared generic Docker images for data science workspaces with your favorite web UI pre-installed to easily deploy your workspace. So you just need to choose your favorite workspace, start the container, access it, add your code, and install your dependencies.

1. Login to the DSRI dashboard
2. Select your project, or create one with a meaningful short name representing your project, e.g.
`workspace-yourname`
3. Go to the **+Add** page, and select to add **From Catalog**





4. Search for templates corresponding to the application you want to deploy among the one described below (make sure the filter for templates is properly checked).

JupyterLab: Access and run your code using the popular Jupyter notebooks, with kernel for python, java, R, julia. It also provides a good web interface to access the terminal, upload and browse the files.

VisualStudio Code: Your daily IDE, but in your browser, running on the DSRI.

RStudio: R users favorite's.

The terminal: For people who are used to the terminal and just want to run scripts, it provides smaller and more stable images, which makes installation and deployment easier. You can use the **Ubuntu** template to start a basic ubuntu image and access it from the terminal.

Any web interface: You can easily run and access most programs with a web interface on the DSRI. You can use the template **Custom workspace** if your application is exposed on port 8888. Otherwise visit the page [Anatomy of a DSRI application](#) for more details.

Desktop interface: there is the possibility to start a container as a Linux operating system with a graphical desktop interface. It can be useful to deploy software like Matlab, but the setup can be a bit more complex. You will get an Ubuntu computer with a basic Desktop interface, running on the DSRI, that you can access directly in your web browser. The desktop interface is accessed through a web application by using noVNC, which exposes the VNC connection without needing a VNC client.

! MORE APPLICATIONS

You can also find more documentation on the different applications that can be deployed from the DSRI under **Deploy applications** in the menu on the left.

Start your workspace

Once you chose your favorite way to run your experiments, you can click on the application you want to use for your workspace. Checkout the description to learn more details about the application that will be deployed.

Then click on **Instantiate Template**, and fill the parameters, such as the password to access the web UI. Note that the application name needs to be unique in the project. Finally click on the **Create** button.

Instantiate Template

Namespace *

ids-project

Application name *

rstudio-root

Must be unique in the project. It will be used to generate the application URL.

RStudio password

rstudio#8913

The password of the RStudio user.

Storage size *

10Gi

Size of the storage used for the notebook (approximate).

Number of threads for OpenBLAS

1

Restricting the number of thread allocated to OpenBLAS can speed up computations using OpenBLAS (leave empty for default 64)

Application Docker image *

ghcr.io/maastrichtu-ids/rstudio:latest

See <https://github.com/MaastrichtU-IDS/rstudio> for more details and to customize the image

Create Cancel

RStudio
R STUDIO ROOT PERSISTENT
[View documentation](#) [Get support](#)

Start RStudio with the rstudio user which has root permissions (sudo).

Use the /home/rstudio folder (workspace of the RStudio UI) to store your data in a persistent storage automatically created.

This template uses rocker RStudio image hosted in the GitHub Container Registry.
Visit <https://github.com/MaastrichtU-IDS/rstudio> for more details and to customize the image
Visit original rocker image at <https://hub.docker.com/r/rocker/rstudio/>

You need root containers enabled (aka. anyuid) in your project to start this application.

The following resources will be created:

- DeploymentConfig
- ImageStream
- PersistentVolumeClaim
- Route
- Secret
- Service

You should see your application in your project dashboard, it can take a few seconds to a few minutes to pull the docker image and start the application.

Once the application has started you will be able to access it by clicking on its circle, then click the **Route**, that has been automatically generated for the web interface, in the Resources tab.

! CHECK THE WORKSHOP

For a more detailed tutorial, you can follow the [workshop to start Data Science applications on the DSRI](#)

Upload your code and data

We recommend you to use `git` to clone your project code in your workspace, as it helps sharing and managing the evolution of your project.

It will be preinstalled in most images, otherwise you can install it easily with `apt-get install git`

With web interface like JupyterLab, VisualStudio Code and Rstudio you can easily upload sma

medium size files directly through the UI with a drag and drop.

Otherwise you can use the terminal, install the `oc` client, and use the `oc cp` or `oc rsync` commands to upload large files to your workspace on the DSRI. See the Upload data page for more details.

Install your dependencies

Once the workspace is started, you can install the different dependencies you need to run your experiments.

It is recommended to save all the commands you used to install the different requirements in a script (e.g. `install.sh`). This will insure you can reinstall the environment easily and faithfully if the container is restarted. You can also use them to create a Docker image with everything prepared for your application.

Most containers for science are based on debian/ubuntu, so you can install new packages with `apt-get`:

```
apt-get update
apt-get install -y build-essentials wget curl
```

Run your code

You can use your web interface to run your code as you like to do: notebooks, rstudio, execute via VSCode

Note that for jobs which are running for a long time the web UI is not always the best solution, e.g. Jupyter notebooks can be quite instable when running a 30 min codeblock.

A quick solution for that is to run your code in scripts, using the bash terminal. You can use the `nohup` prefix, and `&` suffix to run your script in the background, so that you can even disconnect, and come back later to check the results and logs.

For example with a python script, you would do:

```
nohup python my_script.py &
```

The script will run in the background, and all terminal output will be stored in the file `nohup.out`

You can also check if the process is currently running by typing `ps aux` or `top`

You can kill the process by getting the process ID (PID) using the previous commands, and then:

```
kill -9 PID
```

Stop your application

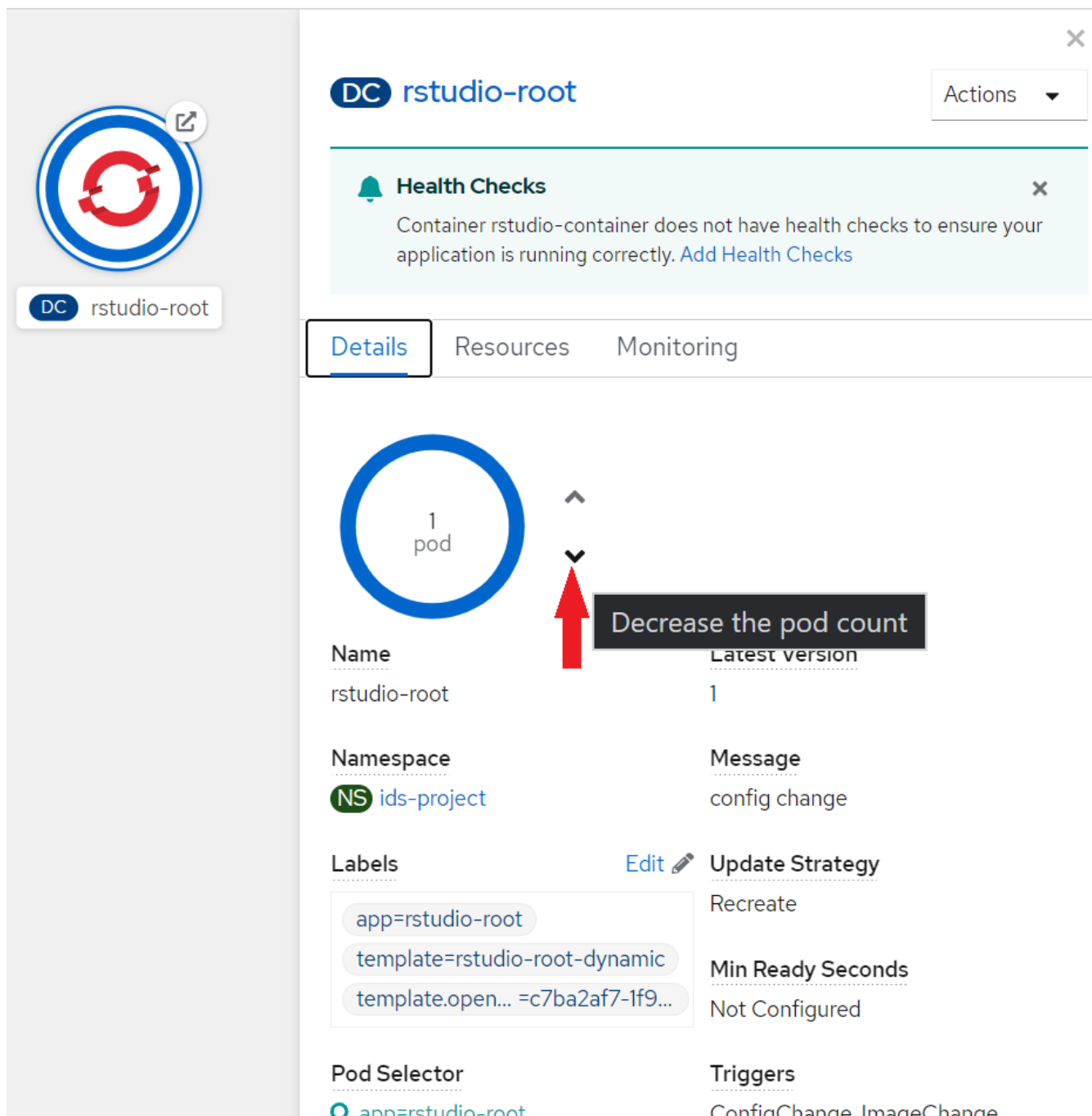
When you are not using your application anymore you can stop the pod. If you are using a Dynamic or Persistent storage you can restart the pod and continue working with all your data in the same state as you left it.

DO NOT WASTE RESOURCES

Please think of stopping applications you are not using to avoid consuming unnecessary resources.

On the **Topology** page click on the down arrow  next to the number of pods deployed.





DC rstudio-root

Actions

Health Checks

Container rstudio-container does not have health checks to ensure your application is running correctly. [Add Health Checks](#)

Details Resources Monitoring

1 pod

Decrease the pod count

Name
rstudio-root

Latest version
1

Namespace
NS ids-project

Message
config change


Labels
app=rstudio-root
template=rstudio-root-dynamic
template.open... =c7ba2af7-1f9...

Edit Update Strategy
Recreate

Min Ready Seconds
Not Configured

Pod Selector
Q app=rstudio-root

Triggers
ConfigChange, ImageChange

You can then restart the pod by clicking the up arrow 

Note that starting more than 1 pod will not increase the amount of resources you have access to, most of the time it will only waste resources and might end up in weird behavior on your side. The web UI will randomly assign you to 1 of the pod started when you access it. This only works for clusters with multiple workers, such as Apache Flink and Spark. Or if you connect directly to each pod with the terminal to run different processes.

Optional: define a docker image

Once you have tested your workspace and you know how to set it up it can be helpful to define a `Dockerfile` to build and publish a Docker image with everything directly installed (instead of

installing your requirements after starting a generic workspace)

1. Start from an existing generic Docker image, depending on the base technologies you need, such as Debian, Ubuntu, Python, JupyterLab, VisualStudio Code, RStudio...
2. Add your source code in the Docker image using `ADD . .` or `COPY . .`
3. Install dependencies (e.g. `RUN apt-get install gfortran`)
4. Define which command to run when starting the container (e.g. `ENTRYPOINT["jupyter", "lab"]`)

Here is a simple example `Dockerfile` for a python application:

```
# The base image to start from, choose the one with everything you need installed
FROM python:3.8

# Change the user and working directory to make sure we are using root
USER root
WORKDIR /root

# Install additional packages
RUN apt-get update && \
    apt-get install build-essentials

# This line will copy all files and folder that are in the same folder as the
# Dockerfile (usually the code you want to run in the container)
ADD . .

# This line will install all the python packages described in the requirements.txt
# of your source code
RUN pip install -r requirements.txt && \
    pip install notebook jupyterlab


# Command to run when the container is started, here it starts JupyterLab as a
# service
ENTRYPOINT [ "jupyter", "lab" ]
```

Here are some examples of `Dockerfile` for various type of web applications:

- [Custom JupyterLab](#) based on the official [jupyter/docker-stacks](#)
- [Custom RStudio](#)
- [VisualStudio Code server](#)
- [Python web app](#)

See the guide to [Publish a Docker image](#) for more details on this topic.



 [Edit this page](#)

*Last updated on **Dec 6, 2022** by **Vincent Emonet***

