
Kiosc Documentation

Release 0.3.0

Oliver Stolpe

Apr 29, 2022

INTRODUCTION

1	What Kiosk is and what it is not	3
1.1	Overview	3
1.2	Installation	4
1.3	Interface	6
1.4	Roles	8
1.5	Cookbook	9
1.6	Containers	17
1.7	Container Templates	31
1.8	Small Files	37
1.9	Overview	38
1.10	Commands	42
1.11	Periodic Tasks	42
1.12	Overview	43
1.13	Containers	44
2	Indices and tables	47

Kiosc is a web application to control Docker containers that run a webserver.

It was conceived to facilitate presenting analysis results interactively (but is not limited to that) by running Docker images that are constricting any application that offers a web interface. Any image can be loaded, and Kiosc manages the access to the web interface of the application.

WHAT KIOSC IS AND WHAT IT IS NOT

Kiosc is a web interface for

- loading Docker images and creating containers from it,
- controlling the state of Docker containers,
- controlling access to containers based on the user management provided by SODAR,
- providing access to the app running in a container.

Kiosc is NOT

- a tool to create Docker images,
- for running Docker images without webserver (you can do that, but you won't benefit from it),
- for directly loading or sharing data (this has to be managed by the Docker image).

Note: You can find the official version of this documentation at readthedocs.io. If you view these files on GitHub, beware that their renderer does not render the ReStructuredText files correctly and content may be missing.

1.1 Overview



1.1.1 General Idea

Kiosc was developed to share analysis results with customers and collaborators that are best displayed using an app – an interactive client-side tool that is based on a webserver. The idea was to bundle the webserver in a custom-build Docker image with an application of choice. Upon starting, the container loads the data by setting the environment variables or passing a parameter to the command when starting the Docker container.

Kiosc takes the role of providing functionality to create, configure, manage and control Docker containers from such Docker images, allowing to set up the environment variables or the start command of the container, and to give access to the web interface of the container using a reverse proxy. Technically, Kiosc can be used to pull and start any Docker image, however, one would not benefit from that as it is specifically designed for Docker images hosting a web server.

A typical workflow scheme is then as follows:

1. Kiosc launches a previously configured docker container
2. The container downloads and initializes necessary data objects and starts a web server on a specified port serving the preconfigured app
3. Kiosc allows the access to the webserver via reverse proxy.
4. Users can navigate to the app served by the container from the Kiosc entry page.

1.1.2 SODAR Universe

Kiosc is based on the SODAR core framework and be linked to an upstream SODAR instance to receive projects, users and role assignments. Based on the project information, Docker containers can be created from available Docker images and shared with collaborators and customers.

1.1.3 Technical Description

- Docker environment
- Technical description of networks
- Reverse proxy

1.2 Installation

We ship Kiosc as a Docker container, and provide Docker compose file to start also other required containers. This part describes how to use Kiosc as a Docker container, and also the manual is based on this.

Disclaimer: It is possible to run Kiosc as is (mode `host`), but this requires additional work to set up the database and scheduler. This is not described in this manual. Also, this has impact on how the Docker containers the user creates are organized and presented. This is also not described in this manual.

The Kiosc Docker container is served via Github Container Registry (*gchr*).

1.2.1 Docker compose

Set up the Docker compose by cloning the repository:

```
$ git clone https://github.com/bihealth/kiosc-docker-compose.git
$ cd kiosc-docker-compose
```

Initialize the folder structure required. Among others, the database will be stored in there, such it is available after restarting the container:

```
$ bash init.sh
```

Copy the `env.example` file to `.env`:

```
$ cp env.example .env
```

Here you can change Kiosc and Django parameters. Most of them are set to reasonable defaults, but changing the `DJANGO_SECRET_KEY` is a good idea:


```
DJANGO_SECRET_KEY=CHANGEMEchangemeCHANGEMEchangemeCHANGEMEchangemeCH
```

When done, start the Docker containers:

```
$ docker-compose up
```

The Kiosc installation can now be reached by accessing `localhost` with your browser.

Configuration

Kiosc can be configured via environment variables. Docker compose can digest a `.env` file. It is a good idea to leave the values as they are.

The environment file could look like this:

```
# Postgres configuration -----
POSTGRES_USER=kiosc
POSTGRES_PASSWORD=password
POSTGRES_DB=kiosc
POSTGRES_HOST=postgres

# Kiosc configuration -----
DATABASE_URL="postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${POSTGRES_HOST}/${
↪ ${POSTGRES_DB}}"

DJANGO_ALLOWED_HOSTS="*"
DJANGO_SECRET_KEY="CHANGEMEchangemeCHANGEMEchangemeCHANGEMEchangemeCH"
DJANGO_SETTINGS_MODULE="config.settings.production"

PROJECTROLES_SITE_MODE=SOURCE

CELERY_BROKER_URL="redis://redis:6379/0"

KIOSC_SERVER_VERSION=main-0
KIOSC_NETWORK_MODE=docker-shared
KIOSC_DOCKER_NETWORK=kiosc-net
KIOSC_DOCKER_WEB_SERVER=kiosc-web
kIOSC_DOCKER_ACTION_MIN_DELAY=1
KIOSC_DOCKER_MAX_INACTIVITY=1
KIOSC_DOCKER_ACTION_MIN_DELAY=7
KIOSC_EMBEDDED_FILES=1
```

Note that setting `PROJECTROLES_SITE_MODE=TARGET` requires an upstream SODAR instance that is running in `SOURCE` mode and that the Kiosc instance is registered to. If no SODAR instance is available or connecting Kiosc to the SODAR instance is not intended, set the `PROJECTROLES_SITE_MODE=SOURCE`. Further description of the `SOURCE/TARGET` mode can be found in the [SODAR Core documentation](#).

Optionally the LDAP can be configured with up to two LDAP servers:

```
# LDAP configuration -----
```

(continues on next page)

(continued from previous page)

```

ENABLE_LDAP=1
AUTH_LDAP_SERVER_URI=...
AUTH_LDAP_BIND_PASSWORD=...
AUTH_LDAP_BIND_DN=...
AUTH_LDAP_USER_SEARCH_BASE=...
AUTH_LDAP_USERNAME_DOMAIN=...
AUTH_LDAP_DOMAIN_PRINTABLE=...

ENABLE_LDAP_SECONDARY=1
AUTH_LDAP2_SERVER_URI=...
AUTH_LDAP2_BIND_PASSWORD=...
AUTH_LDAP2_BIND_DN=...
AUTH_LDAP2_USER_SEARCH_BASE=...
AUTH_LDAP2_USERNAME_DOMAIN=...
AUTH_LDAP2_DOMAIN_PRINTABLE=...

```

If the KIOSC_ environment variables are not set, Kiosc selects the defaults as stated in the following table.

Environment variable	Default	Description
KIOSC_NETWORK_MODE	host	Can be host or docker-shared. Indicates whether installation runs in a Docker environment or not.
KIOSC_DOCKER_NETWORK	kiosc-net	Name of the Docker network for the users Docker containers.
KIOSC_DOCKER_WEB_SERVER	kiosc-web	Name of the web server Docker container.
KIOSC_DOCKER_ACTION_MIN_DELAY	1	Min delay in seconds for Docker container actions.
KIOSC_DOCKER_MAX_INACTIVITY	7	Max threshold for inactive running Docker containers in days.
KIOSC_EMBEDDED_FILES	True	Enable the feature to upload small files to Kiosc that can be served to the Docker containers.

1.3 Interface

When accessing the Kiosc web interface, one is greeted with the Kiosc logo and a login form. You need to have an account with Kiosc or the LDAP must be configured to be able to log in with your institute account. Approach your system administrator about that matter if you are unsure.

KIOSC Beta Extra Link Help

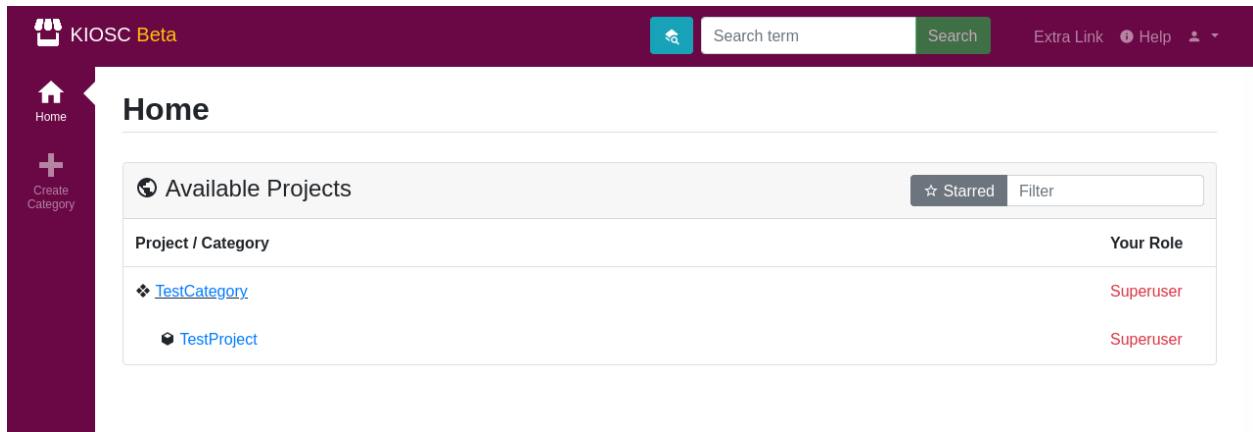
Login

Please log in.

user

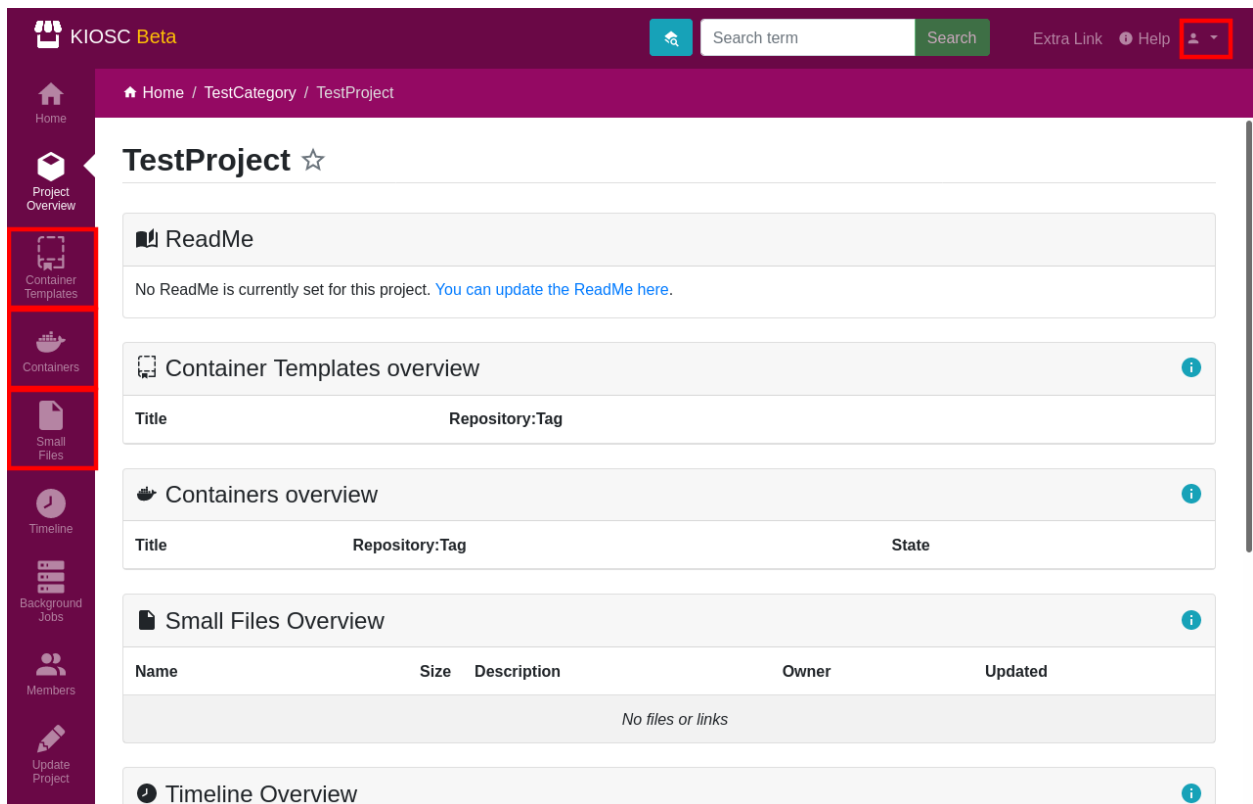
Login

Once logged in, you will see an overview of all the projects you are assigned to, alike SODAR. If you do expect to have access to a project you do not have access to, ask the leader or delegate of that project to grant you access to that project. If the Kiosc instance is linked to a SODAR instance, the access is set in SODAR and must then be synchronized to Kiosc by the administrator.

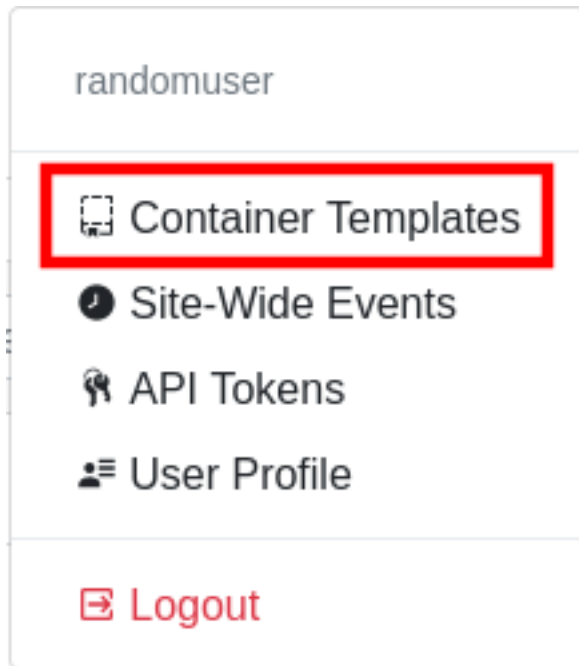


To be able to access the Kiosc apps, click on a project. On the left-hand side you will have access to multiple apps, three of them are of interest:

1. **Containers** for creating and controlling Docker containers.
2. **Container Template** for creating templates for Docker containers.
3. **Small Files** for uploading smaller files that the containers can then access.



Additionally, in the top-right corner is a drop-down menu for account settings and site apps. This gives access to the site-wide container template app. This hosts container templates that are accessible site-wide and not project-wide.



1.4 Roles

Kiosc provides the same roles as SODAR Core as it is based on this framework. However, this section will clarify what each role is allowed to do within Kiosc.

Contents

- *Roles*
 - *Containers*
 - *Container Templates*
 - * *Project-wide*
 - * *Site-wide*

1.4.1 Containers

Role	Create/Update	Delete	Start	Stop	(Un)pause	View
Administrator	OK	OK	OK	OK	OK	OK
Owner	OK	OK	OK	OK	OK	OK
Delegate	OK	OK	OK	OK	OK	OK
Contributor	OK	OK	OK	OK	OK	OK
Guest			(OK)*			OK

* *Guests can't start the container directly, but do so indirectly by requesting to view a not running container.*

1.4.2 Container Templates

Project-wide

Role	Create/Update	Delete	Copy*/Duplicate	View
Administrator	OK	OK	OK	OK
Owner	OK	OK	OK	OK
Delegate	OK	OK	OK	OK
Contributor	OK	OK	OK	OK
Guest				OK

* *Copy project-wide or site-wide templates.*

Site-wide

Role	Create/Update	Delete	Duplicate	View
Administrator	OK	OK	OK	OK
Owner				OK
Delegate				OK
Contributor				OK
Guest				OK

1.5 Cookbook

Contents

- *Cookbook*
 - *Guest accessing the web interface of a container*
 - *Create a container running ...*
 - * *Shiny (using environment variables)*
 - * *Dash (using environment variables)*
 - * *seaPiper*
 - * *cellxgene (using a command)*
 - * *cellxgene (using a command with small files)*
 - * *ScElvis (using a command and environment variables)*

1.5.1 Guest accessing the web interface of a container

You are a guest of a project. You can list the containers of the project, and access the details of each container. You can't change the status of a container, except indirectly by viewing a container that is not running. Probably you like to access the web interface provided by the container.

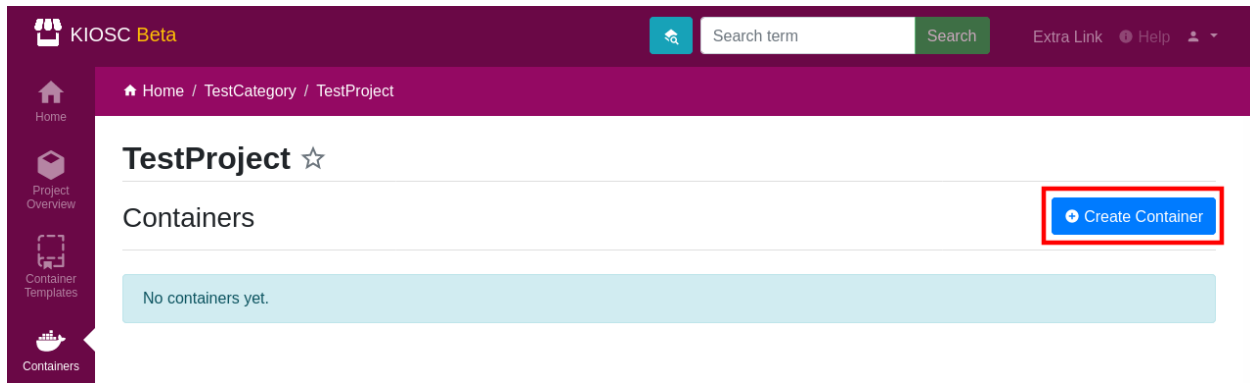
To proceed, click on a project and then select the **Container** app. This will display a list of all containers in the project. On the right-hand side of each container is a button with an eye icon. The button might be either gray-outlined with a crossed-out eye, or with a blue background with an open eye. The crossed-out eye indicates that the container is not running and this will also be reflected in the state. The blue open eye indicates that the container is available. No matter the state, clicking the icon will open the web interface provided by the container. The difference is that in the crossed-out state Kiosc tries to start the container before accessing the web interface which might take some time while in the running state the web interface will be displayed immediately.

1.5.2 Create a container running ...

To create a container switch to the **Containers** app



and select **Create Container**. This will be the starting point for the following tutorials.



After the creation of the container you will be redirected the details of the container. The state will be set to `initial` which indicates that there is the container object but no actual Docker container (yet). You can find the operations menu (cog icon) on the top right of the details page. Open the dropdown menu by clicking the cog icon and select **Start**, or click the crossed-out eye icon to start and access the container directly.

Shiny (using environment variables)

Shiny example

Dataset iris

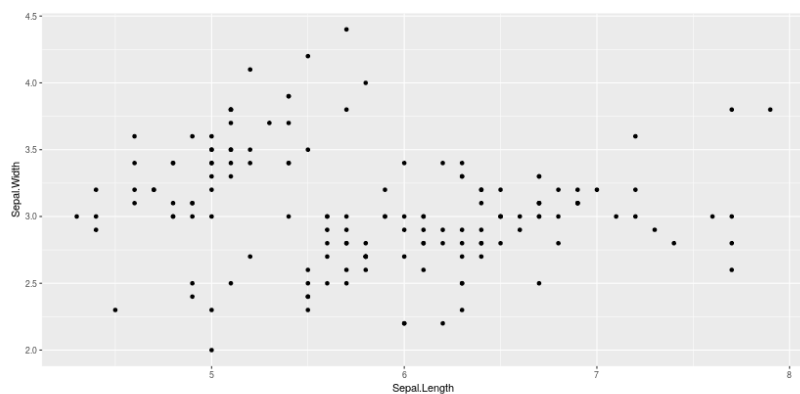
X covariate
 Sepal.Length

Y covariate
 Sepal.Width

Color by
 NA

Symbol by
 NA

Trellis (facet) by
 NA



For this tutorial we provide you with a pre-build [Docker image with a Shiny application](#). Use the linked repository as a base to create your own Docker image.

This example sets up a simple Shiny application loading the popular iris dataset. The data set is loaded by setting the dataset variable in the environment. Fill out the following fields and click **Create**:

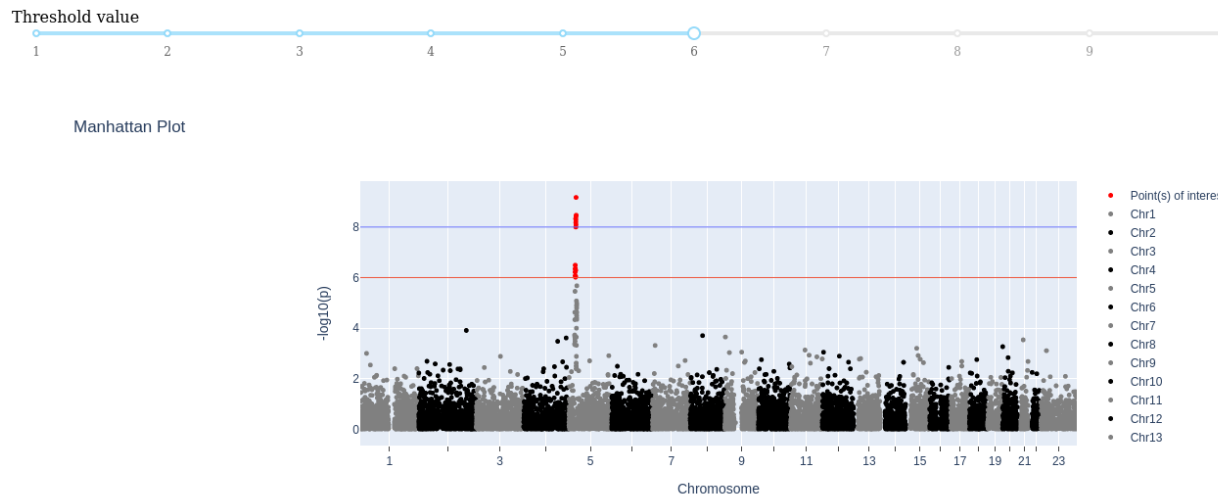
Title	<i>Set a unique title that helps you identify the container easily.</i>
Repository	ghcr.io/bihealth/kioscshinytest
Tag	latest
Container Port	8080
Environment	{"title": "Kiosc Shiny App example", "dataset": "iris"}

The **Environment** field should contain a [JSON object literal](#), which corresponds to a Python dictionary with the exception that only double quotes are allowed, or nothing.

The value in the **Environment** field will be transformed and passed to the environment of the container. In the above example, the Docker container will hold two environment variables. Imagine that inside the container the following lines will be performed upon start:

```
$ export title="Kiosc Shiny App example"
$ export dataset=iris
```

Dash (using environment variables)



For this tutorial we provide you with a pre-build Docker image with a Dash application. Use the linked repository as a base to create your own Docker image.

In this example we are running a Dash application. As we are behind a reverse proxy, the Dash application needs some tweaks to make it load all scripts and stylesheets into the container when started. The Dash application was extended by accepting an environmental variable named `PUBLIC_URL_PREFIX`, and for this to work, you have to set up this environment variable and set it to the value `__KIOSC_URL_PREFIX__`. This acts as a place holder that is substituted with the path to the container how it is known to the outside. Fill out the following fields and click **Create**:

Title	<i>Set a unique title that helps you identify the container easily.</i>
Repository	ghcr.io/bihealth/kiosc-example-dash
Tag	main-0
Container Port	8050
Environment	{"PUBLIC_URL_PREFIX": "__KIOSC_URL_PREFIX__"}

The **Environment** field should contain a **JSON object literal**, which corresponds to a Python dictionary with the exception that only double quotes are allowed, or nothing.

The value in the **Environment** field will be transformed and passed to the environment of the container. In the above example, the Docker container will hold two environment variables. Imagine that inside the container the following lines will be performed upon start:

```
$ export PUBLIC_URL_PREFIX=containers/proxy/abcdef123...
```

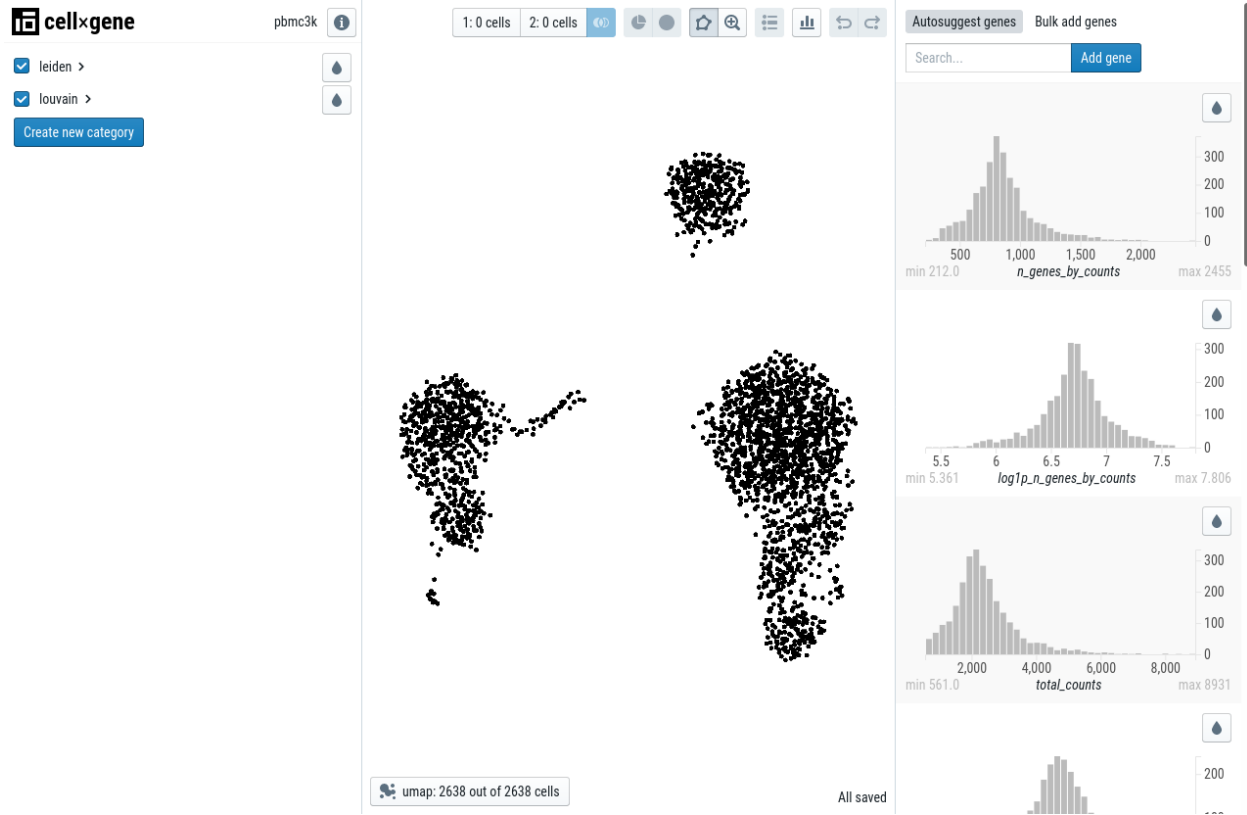

seaPiper

For this tutorial we provide you with a pre-build Docker image with a seaPiper application. Use the linked repository as a base to create your own Docker image.

seaPiper is based on Shiny. Fill out the following fields and click **Create**:

Title	<i>Set a unique title that helps you identify the container easily.</i>
Repository	ghcr.io/bihealth/kiosc-seapiper-demo
Tag	latest
Container Port	8080

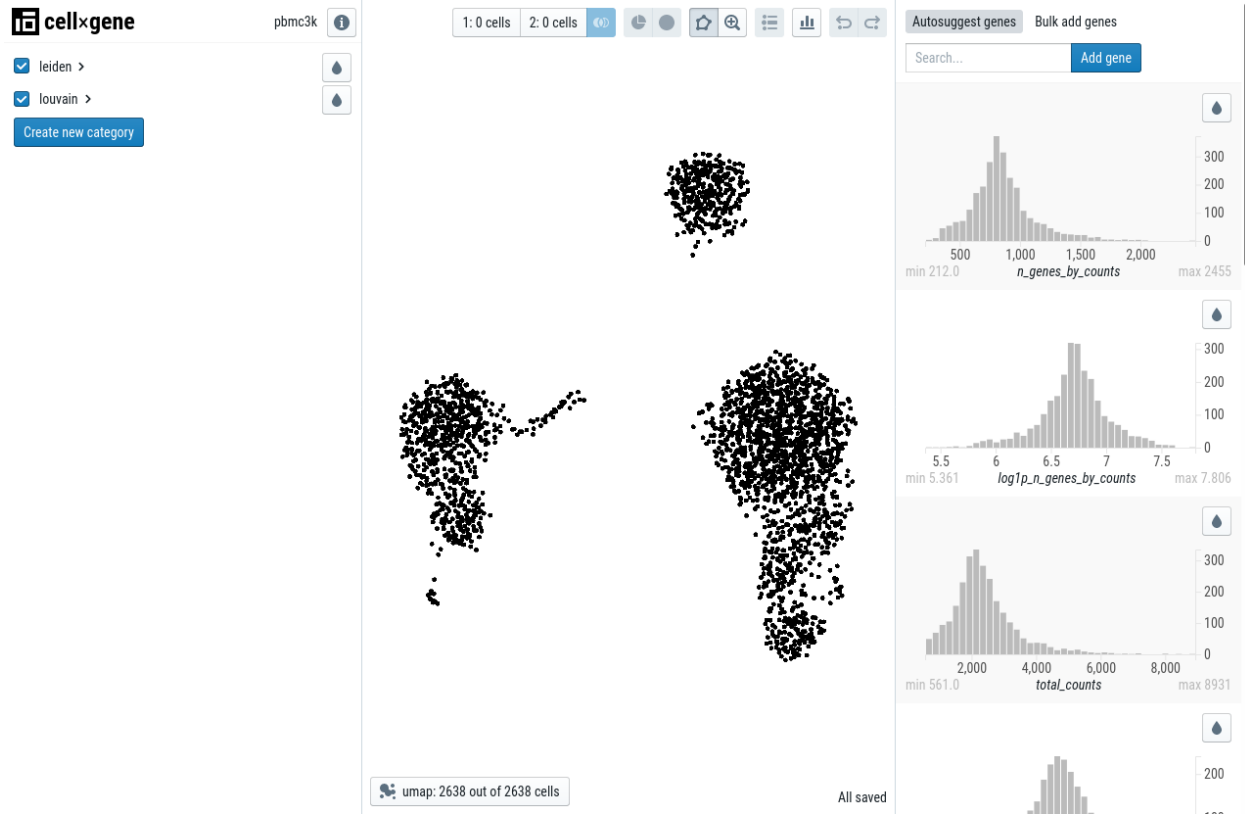
cellxgene (using a command)



This example takes a publicly available container and passes a command that is run when starting the container. In this case, the cellxgene application is started immediately when running the container. The data is loaded by passing the data URL to the command. Fill out the following fields and click **Create**:

Title	<i>Set a unique title that helps you identify the container easily.</i>
Repository	quay.io/biocontainers/cellxgene
Tag	1.0.0--pyhdfd78af_0
Container Port	8050
Command	cellxgene launch https://cellxgene-example-data.czi.technology/pbmc3k.h5ad -p 8050 --host 0.0.0.0 --verbose

cellxgene (using a command with small files)



This example is the same as above but using a file uploaded to Kiosc. A command to copy-and-paste can't be provided as the link to the file depend on the UUID that is randomly created. To get the file into Kiosc, download the file from the official server and upload it to Kiosc:

1. Download [example data](#).
2. Go to a Kiosc project and select the *Small Files app*.
3. Upload the `pbmc3k.h5ad` file. It is now available during container creation.

Now continue with the container creation. To make use of the uploaded file, when inserting the command, place the cursor at the mentioned position in the command, select the file and click *Insert*.

Files

/pbmc3k.h5ad

Insert

Command

cellxgene launch | -p 8050 --host 0.0.0.0 --verbose

This will place a link at the cursor position.

Files

/pbmc3k.h5ad

Insert

Command

```
cellxgene launch http://kiosc-web:8080/containers/file/serve/424e4362-01b8-4fe6-b24c-1262cf4f148c/pbmc3k.h5ad -p 8050 --host 0.0.0.0 --verbose
```

Title	<i>Set a unique title that helps you identify the container easily.</i>
Repository	quay.io/biocontainers/cellxgene
Tag	1.0.0--pyhdfd78af_0
Container Port	8050
Command	cellxgene launch <PLACE_CURSOR_HERE_BEFORE_INSERTING_FILE> -p 8050 --host 0.0.0.0 --verbose
Files	/pbmc3k.h5ad

ScElvis (using a command and environment variables)



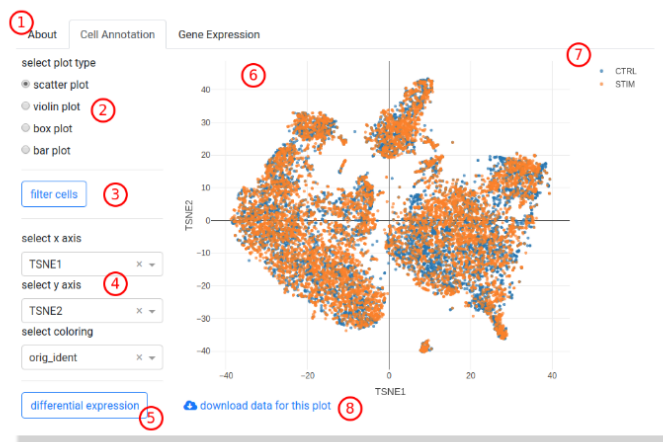
ScElvis

ScElvis is a web app for the visualization and interactive exploration of single-cell transcriptomic data.

Quickstart

- Start by selecting the dataset you want to work with from the **Go To** menu on the top right
- check out our [Documentation](#), in particular the [Tutorial](#), or the movie on our [github page](#)
- if you find ScElvis useful for your research, please consider citing our [paper](#)

Main user interface



This example sets up the ScElvis. ScElvis is based on Dash. For this to work, you have to set up two environment variables, `SCELVIS_URL_PREFIX` helps the application alter the URL path to load scripts and style sheets into the container and `SCELVIS_DATA_URL` sets the data that is to be loaded into the container. Fill out the following fields and click **Create**:

Title	<i>Set a unique title that helps you identify the container easily.</i>
Repository	ghcr.io/bihealth/scelvis
Tag	v0.8.6
Container Port	8050
Environment	{"SCELVIS_URL_PREFIX": "__KIOSC_URL_PREFIX__", "SCELVIS_DATA_SOURCES": "https://cellxgene-example-data.czi.technology/pbmc3k.h5ad"}
Command	scelvis run

The **Environment** field should contain a [JSON object literal](#), which corresponds to a Python dictionary with the exception that only double quotes are allowed, or nothing.

The value in the **Environment** field will be transformed and passed to the environment of the container. In the above example, the Docker container will hold two environment variables. Imagine that inside the container the following lines will be performed upon start:

```
$ export SCELVIS_URL_PREFIX=containers/proxy/abcdef123...
$ export SCELVIS_DATA_SOURCES=https://cellxgene-example-data.czi.technology/pbmc3k.h5ad
```

In addition to the user defined variables, the `title`, `description` and `container_port` are also exposed as environment variables to the Docker container (as `TITLE`, `DESCRIPTION` and `CONTAINER_PORT` respectively):

```
$ export TITLE="Some unique title"
$ export DESCRIPTION="Some description"
$ export CONTAINER_PORT=8050
```

1.6 Containers

Container objects hold the information to create and afterwards to control the underlying Docker containers. All information the user enters is used during the creation of the container (which happens when the container is started). They also hold information about the current state and include the logs reported by any process associated with the container.

1.6.1 Overview

Find the Containers icon in the left-hand menu to open the Container app. This will list all available containers and offers the menus to create new containers, control and delete containers and show details including its logs.

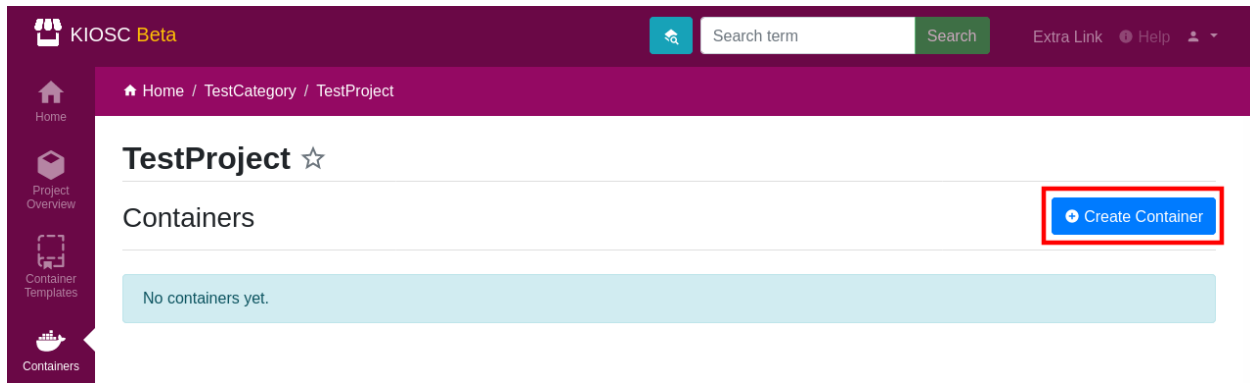


1.6.2 Create

Contents

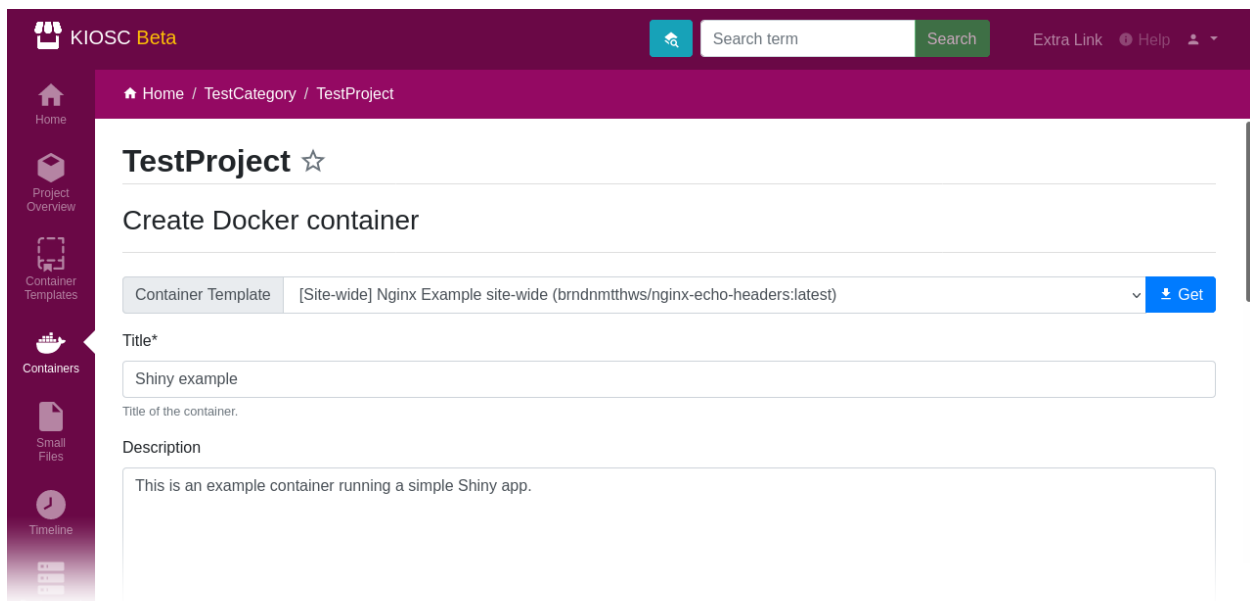
- *Create*
 - *Container templates*
 - *Environment*
 - *Environment secret keys*
 - *Container path*
 - *Timeout*
 - *Heartbeat URL (inactive)*
 - *Files*
 - *Max retries*
 - *Inactivity threshold*

Click the **Create Container** button to enter the form for creating a new container object. This does not create a Docker container yet but only gathers information. The actual Docker container is created when starting the container.



Fill in at least the mandatory fields, marked with a star (*). Some of them are pre-filled with a reasonable default value. Change only if required. Others like Title, Repository, Tag and Container Port have to be set by the user. Below is a detailed description of each form field. In the example screenshots, we set up a Shiny app.

Fill in a reasonable title that helps you identify the container. The title must be unique. A description is helpful, but not required.



Fill in the repository, tag and container port.

Home / TestCategory / TestProject

Repository*

ghcr.io/bihealth/kioscshinytest

The repository/name of the image.

Tag*

latest

The tag of the image.

Container port*

8080

Server port within the container

Container path

Click the **Create** button to create the container object. This does not create the actual Docker container yet.

Timeline

Background Jobs

Members

Update Project

The command to execute

Cancel Create

Set the content for your footer in include/_footer.html. KIOSC v0.1.0 / SODAR Core v0.10.5+10.g36ed460

Container templates

To make use of the container templates, select a template from the top-hand dropdown menu and click **Get**. This will populate all form fields that are set in the template with you create form. Anything you already entered will be overwritten. The prefix [Site-wide] or [Project-wide] indicates whether this template is either a site-wide or a project-wide template.

Environment

Environment variables can be specified using a JSON dictionary. Top-level keys in the dictionary become the environmental variables visible to the app launched in the container:

```
{
  "ID": "My container",
  "LIST": [ "A", "B", "C" ]
}
```

Given the above example, two environment variables will be defined: `ID` and `LIST`. The contents of `ID` will be `My container`; the contents of `LIST` will be `['A', 'B', 'C']`. Note that the double quotes will be changed to single quotes.

These variables are available to the web app of the container, and can be used to specify e.g. a data source or other parameters for the container web app.

In addition to the user defined variables, the `title`, `description` and `container_port` are also exposed as environment variables to the Docker container (as `TITLE`, `DESCRIPTION` and `CONTAINER_PORT` respectively). The complete list looks like this:

```
{
  "ID": "My container",
  "LIST": [ "A", "B", "C" ],
  "TITLE": "Some title",
  "DESCRIPTION": "Some description",
  "CONTAINER_PORT": 8080,
}
```

Environment secret keys

Environment secret keys is a comma-separated list of sensitive keys to environment variables that have to have a corresponding key defined in the JSON dictionary in the `environment` field. Those variables will be masked when editing them or viewing the details of the container.

Container path

The container path is the folder structure appended to the web address of the container.

Timeout

The timeout is set in seconds and is set as the time limit for any Docker action (start/stop/etc..) to complete.

Heartbeat URL (inactive)

The heartbeat URL can be used to check whether the container app runs correctly. (Feature is currently inactive)

Files

This dropdown provides the files that were uploaded to Kiosc via the `Small Files` app to the project the current container is created in.

To get the internal link to the file the container then can access, click `Insert` and the link will be appended to the `command` field.

Max retries

Maximal number of retries for an action in case of failure. If an action (e.g. starting a container) fails, it will be retried this many times.

Inactivity threshold

Number of days the container is allowed to run without proxy access. If this threshold is hit, the container will be stopped.

1.6.3 Details

Click on the title of a container to access its details and the logs. You will also be forwarded to the details once you created a container object. A detailed description of all the fields can be found below.

Contents

- *Details*
 - *Initial container*
 - *Running container*
 - *Fields*
 - * *Environment & Environment Secret Keys*
 - * *State*
 - * *Last action*
 - * *Date of latest Docker log*
 - * *Logs*

Initial container

When you created a container, the detail page will provide you with information about the container object. Please note that the container is still in initial state and not running yet. Thus, the button to access the proxy (eye icon) is not active.

KIOSC Beta

Home / TestCategory / TestProject

TestProject ☆

Container Shiny example

View Controls Container List

Description

This is an example container running a simple Shiny app.

Details

SODAR UUID	34c33253-3285-4797-9f92-effac26e09a1
Date Created	2021-10-26 19:20
Date Modified	2021-10-26 19:20
Image ID	no value
Container ID	
Repository:Tag	ghcr.io/bihealth/kioscshinytest:latest
Container IP	
Container Port	8080
Container Path	no value

In the bottom of the page the logs are displayed. In this case the logs only contain one entry, indicating that the object has been created.

KIOSC Beta

Home / TestCategory / TestProject

Command no value

Environment

```
{
  "dataset": "iris",
  "title": "Kiosc Shiny App example"
}
```

Environment Secret Keys no value

Max Retries 5

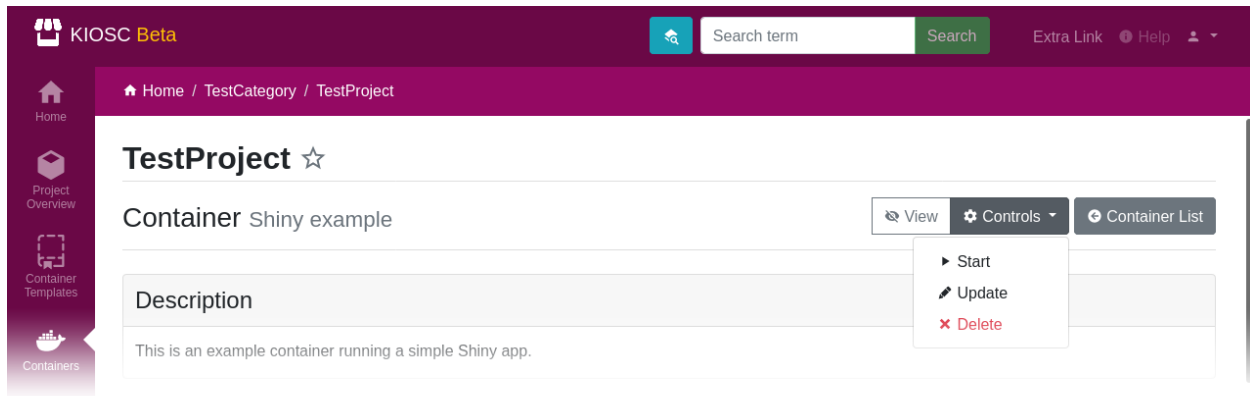
Inactivity Threshold [days] 7

State initial

Logs

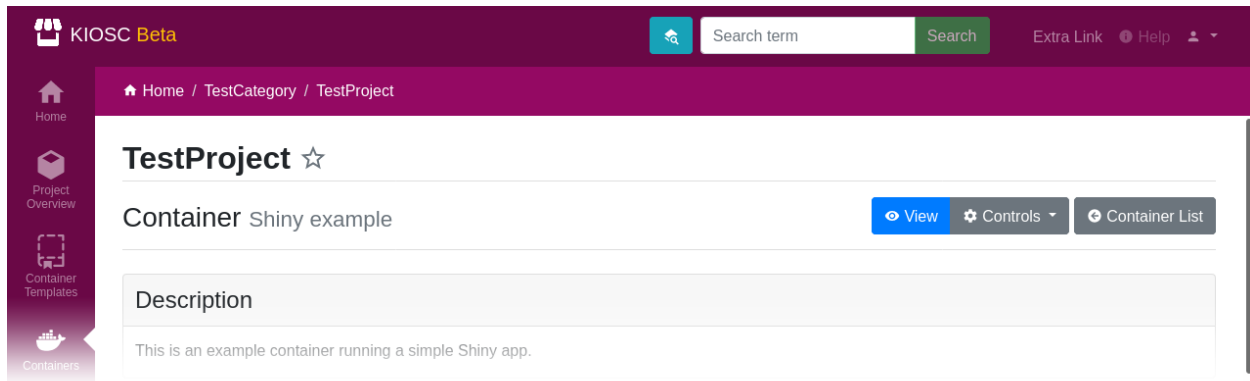
```
[2021-10-26 19:20:42 INFO admin] (Object) Created
```

To start the container, open the operator menu located on the right-hand side of the title and click the **Start** item. The Docker container will be created and started. This menu also provides you with the options to edit (**Update**) or **Delete** the container.



Running container

Once the container is running, the detail page for the container changes. The operator menu will change its entries, the button to access the proxy server with the eye icon will turn blue and the state of the container will be set to **running**.



The logs will be updated and now contain the logs coming from the Docker container.

KIOSC Beta

Home / TestCategory / TestProject

Environment Secret Keys	no value
Max Retries	5
Inactivity Threshold [days]	7
State	running
Last action	start (0/5)

Logs

```
[2021-10-26 19:20:42 INFO admin] (Object) Created
[2021-10-26 19:22:00 INFO admin] (Action) Start
[2021-10-26 19:22:01 INFO admin] (Task) Pulling image ...
[2021-10-26 19:22:02 INFO admin] (Docker) Pulling from bihealth/kioscshinytest
[2021-10-26 19:22:02 INFO admin] (Docker) Digest: sha256:dc7a7625f5a8260eb01d52de2168dd8ae979bc658c7b7a3d006735207bcad033
[2021-10-26 19:22:02 INFO admin] (Docker) Status: Image is up to date for ghcr.io/bihealth/kioscshinytest:latest
[2021-10-26 19:22:02 INFO admin] (Task) Pulling image succeeded
[2021-10-26 19:22:02 INFO admin] (Task) Starting ...
[2021-10-26 19:22:03 INFO admin] (Task) Starting succeeded
[2021-10-26 19:22:05 INFO anonymous] (Docker)
[2021-10-26 19:22:05 INFO anonymous] (Docker) Listening on http://0.0.0.0:8080
```

When a user accesses the container via the proxy URL (which is triggered by clicking the button with the eye icon), this will also be displayed in the logs. Look out for an entry that is provided by (Proxy), starting with Accessing [...].

KIOSC Beta

Home / TestCategory / TestProject

Inactivity Threshold [days] 7

State **running**

Last action start (0/5)

Logs

```
[2021-10-26 19:22:02 INFO admin] (Docker) Pulling from bihealth/kioscschinytest
[2021-10-26 19:22:02 INFO admin] (Docker) Digest: sha256:dc7a7625f5a8260eb01d52de2168dd8ae979bc658c7b7a3d006735207bcad033
[2021-10-26 19:22:02 INFO admin] (Docker) Status: Image is up to date for ghcr.io/bihealth/kioscschinytest:latest
[2021-10-26 19:22:02 INFO admin] (Task) Pulling image succeeded
[2021-10-26 19:22:02 INFO admin] (Task) Starting ...
[2021-10-26 19:22:03 INFO admin] (Task) Starting succeeded
[2021-10-26 19:22:05 INFO anonymous] (Docker)
[2021-10-26 19:22:05 INFO anonymous] (Docker) Listening on http://0.0.0.0:8080
[2021-10-26 19:22:51 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:51 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:51 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
[2021-10-26 19:22:52 INFO admin] (Proxy) Accessing http://ec9525f2bd5a:8080
```

Fields

Environment & Environment Secret Keys

Names of sensitive environment variables can be entered in the `Environment secret keys` field.

If you have set an `environment` and registered `environment_secret_keys`, the value of the corresponding items in the environment dictionary are displayed in Kiosk as `<masked>`, indicating that they are available to the system but are not displayed for security reasons. However, they will still be visible in plain in the container environment.

State

The current state is presented and highlighted:

- **initial**, indicating that the database object has been created but no actual Docker container exists yet.
- **running**
- **failed**, indicating that something went wrong
- **exited**
- **paused**

If there is a small bell icon next to the state, this indicates that the last user action and the current state of the Docker container do not match.

Last action

The last action performed on the container of any user is displayed, if available. If there is an inconsistency found between the actual Docker state and the last user action (indicated by the bell icon right to the state), a cron job running every few minutes tries to perform the last known issued user action. The first number next to the action is a counter, indicating how many times it tried to re-perform the action, with the maximal limit indicated by the second number.

Date of latest Docker log

When a Docker log has been fetched in the past, this date indicates the timestamp of the latest Docker log and synchronisation of the Docker state. Docker logs are not displayed immediately in the log file but fetched by a background process every few minutes. This line is missing when there are no fetched Docker logs.

Logs

The logs will update themselves every half minute. As described above, Docker logs are also fetched only every few minutes from the Docker container, thus there can be a bit of latency until logs are displayed.

The log window combines logs from multiple sources. The structure of a log entry is:

```
[YYYY-MM-DD HH:MM:SS <LOG_LEVEL> <USER>] (<PROCESS>) <MESSAGE>
```

For example:

```
[2021-09-08 22:57:26 INFO anonymous] (Task) Syncing last registered container state_
↔(running) with current Docker state (exited)
```

Currently the following sources can contribute to the log:

- **Task:** Logs reported by automatically running background tasks. Usually they are issued by **anonymous**.
- **Docker:** Logs reported by Docker for this container. They are fetched every half minute, so they might not appear immediately.
- **Action:** Any action the user issues on the container.
- **Proxy:** Issued when accessing the proxy.
- **Object:** Issued when changes in the database object are made that represents the Container in Kiosc.

1.6.4 Access & Controls

Contents

- *Access & Controls*
 - *Access (via Proxy)*
 - *Controls*
 - * *Start*
 - * *Stop*
 - * *Pause*

- * *Unpause*
- * *Restart*
- * *Update*
- * *Delete*

Access (via Proxy)

The web application running inside of the container can be accessed when clicking the button with the eye icon. A grey and crossed-out eye indicates that the container is currently not running. A click on the button will start the container and access the web application afterwards.



A blue button with an eye icon indicates that the container is running. The access will happen immediately when clicking the button.



Controls

The Controls dropdown menu (cog icon) comprises multiple actions that can be issued on a container, displayed depending on the state the container is currently in. In the details page this menu is presented by the cog icon + Controls, while in the list this is presented by the cog icon only.

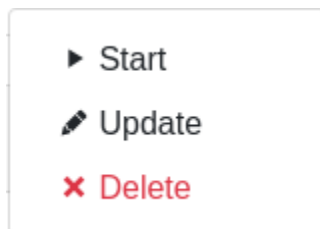
The Controls button on the details page:



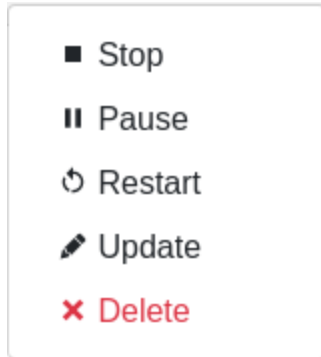
The Controls button on the container list:



When a container is stopped, the selection includes the actions Start, Update and Delete, given the permissions.



When a container is running, the selection includes the actions Stop, Pause, Restart, Update and Delete, given the permissions.



Start

Create a container from a Docker image and start it. If the image isn't cached yet, it is pulled from the specified repository. An existing container is always wiped before performing the starting action.

Internally, the following cadence is performed:

```
docker rm
docker pull
docker create
docker run
```

The state should be **running** when performed successfully.

Stop

Stop a running Docker container. Only available when Docker container state is reported as running.

Internally, a `docker stop` is performed.

The state should be **exited** when performed successfully.

Pause

Pause a running Docker container. Only available when Docker container state is reported as running.

Internally, a `docker pause` is performed.

The state should be **paused** when performed successfully.

Unpause

Unpause a paused Docker container. Only available when Docker container state is reported as paused.

A `docker unpause` is performed.

The state should be **running** when performed successfully.

Restart

Restart a running container. Only available when Docker container state is reported as running.

Internally, the following cadence is performed:

```
docker stop
docker rm
docker pull
docker create
docker start
```

(It's NOT a `docker restart` as the name would suggest.)

The state should be **running** when performed successfully.

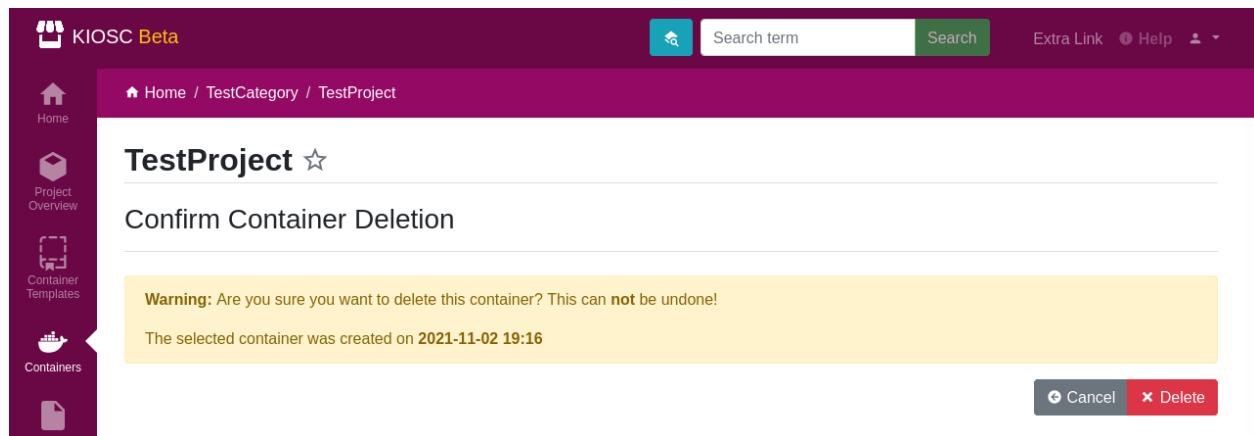
Update

This leads to the form to update the setting of the current container. Please note that values of items in the `environment` dictionary are displayed as `<masked>` if listed in the `environment_secret_keys`. When left as `<masked>`, the value itself will not change. To set a new value, simply change the value.

If the Docker container state is reported as running, a restart as described above will be performed to account for the changes.

Delete

This makes sure that the associated Docker container is not running and stops it if necessary, and deletes the Docker container as well as the database object. This action can't be undone.



1.7 Container Templates

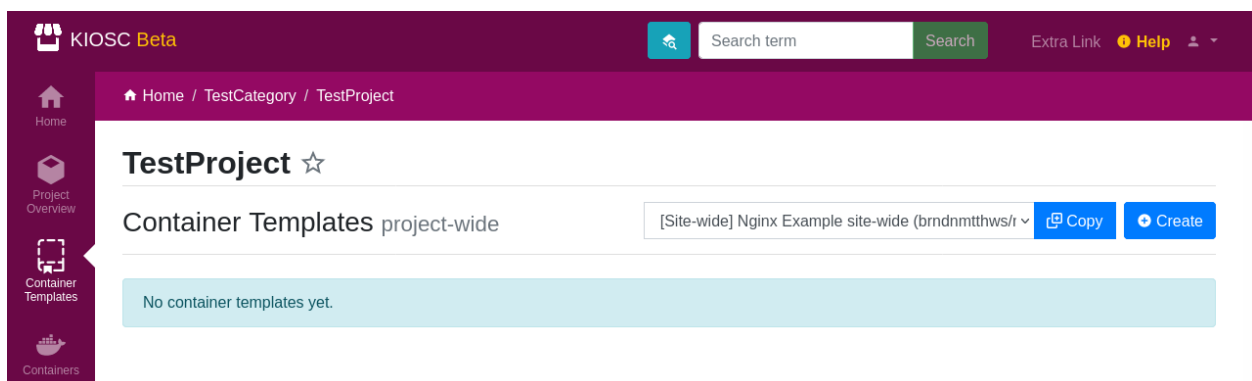
Container templates were conceived to facilitate the creation of containers by offering default settings that can be copied during the creation of an actual container. Container templates themselves can be duplicated and copied to different projects. They do exist as a project-wide for every user, and as site-wide only available for the administrators.

1.7.1 Overview

Find the **Container Templates** icon in the left-hand menu to open the Container Template app. This will list all available container templates and offers the menus to create, update and delete templates.

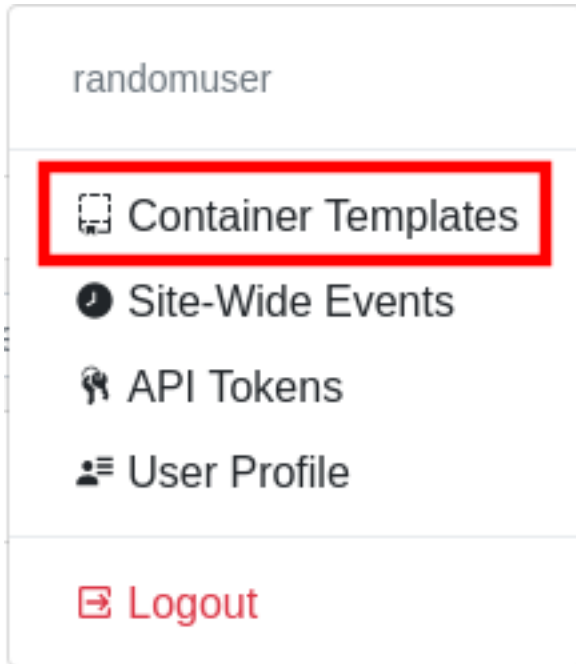


Note that the new templates will be visible in all your projects on the KIOSC site.



Site-wide templates

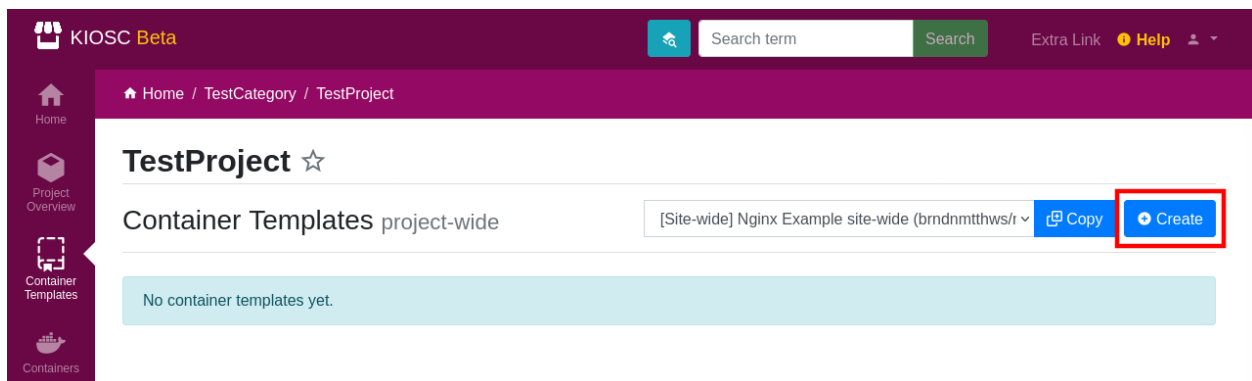
The site-wide container templates can be found in the settings menu.



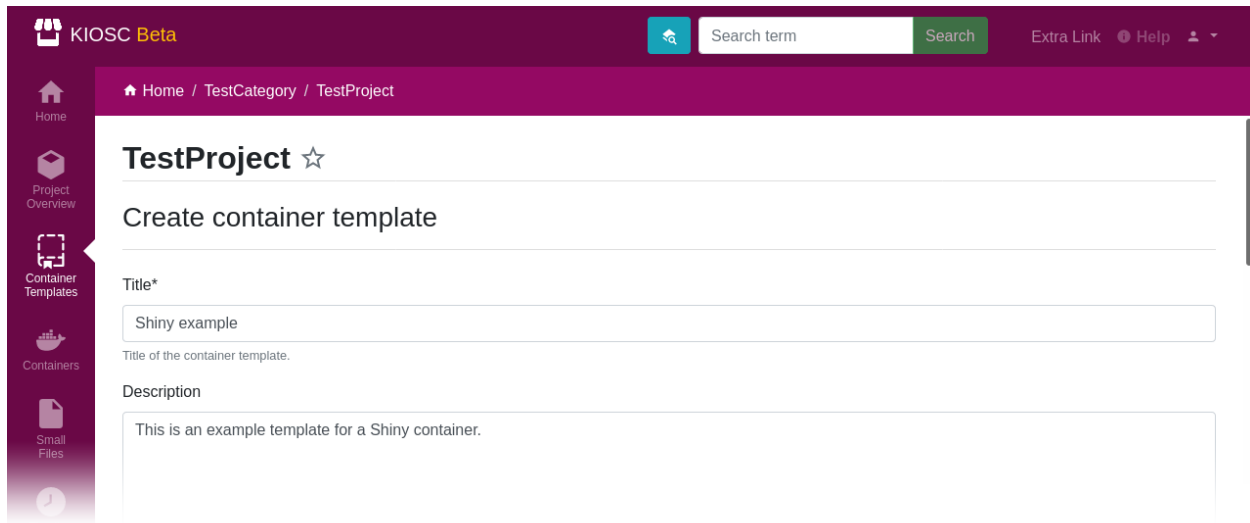
The difference between project-wide templates and site-wide templates is that only administrators can create site-wide templates, and as the name suggests are available on the entry site, i.e. can be used in every project. Project-wide templates are created in a specific project but can be accessed in every project the user has access to.

1.7.2 Create

Click **Create** to enter the form to create a template.



The only mandatory field is **Title**, which is also unique. Everything else can be left out, although some values should be set as the template makes no sense otherwise.



KIOSC Beta

Home / TestCategory / TestProject

TestProject ☆

Create container template

Title*

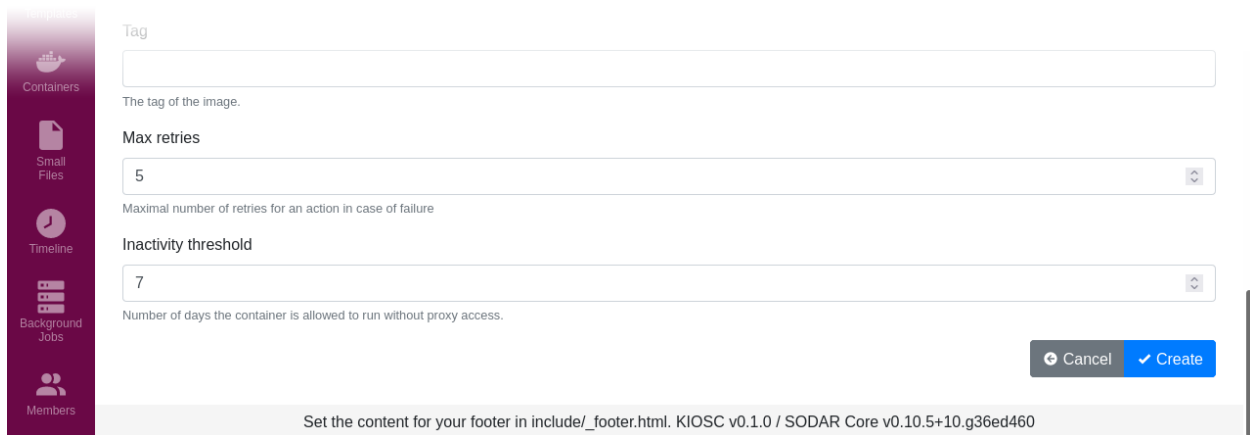
Shiny example

Title of the container template.

Description

This is an example template for a Shiny container.

Create the template by clicking Create.



KIOSC Beta

Home / TestCategory / TestProject

TestProject ☆

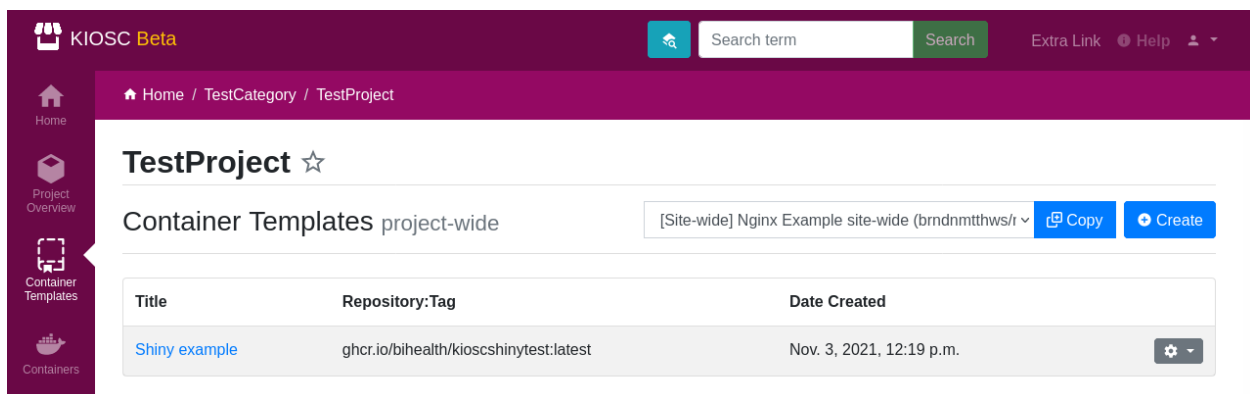
Container Templates project-wide

[Site-wide] Nginx Example site-wide (brndnmtthws/...) [Copy](#) [Create](#)

Title	Repository:Tag	Date Created
Shiny example	ghcr.io/bihealth/kioscshinytest:latest	Nov. 3, 2021, 12:19 p.m.

Set the content for your footer in include/_footer.html. KIOSC v0.1.0 / SODAR Core v0.10.5+10.g36ed460

In the overview, the container template will be listed.



KIOSC Beta

Home / TestCategory / TestProject

TestProject ☆

Container Templates project-wide

[Site-wide] Nginx Example site-wide (brndnmtthws/...) [Copy](#) [Create](#)

Title	Repository:Tag	Date Created
Shiny example	ghcr.io/bihealth/kioscshinytest:latest	Nov. 3, 2021, 12:19 p.m.

1.7.3 Details

Click on the title of a container template in the overview to enter the details of a container template. These are the same details that need to be provided when creating a container.

The screenshot shows the Kiosc Beta interface. The top navigation bar is purple with the Kiosc Beta logo, a search bar, and links for Extra Link, Help, and a user profile. The left sidebar contains icons for Home, Project Overview, Container Templates (selected), Containers, Small Files, Timeline, Background Jobs, and Members. The main content area has a breadcrumb trail: Home / TestCategory / TestProject. Below this, the title 'TestProject' is followed by a star icon. The subtitle is 'Container Template Shiny example'. A button labeled 'Container Template List' is in the top right. The 'Description' section contains the text: 'This is an example template for a Shiny container.' The 'Details' section is a table with the following data:

SODAR UUID	73e188e3-7711-4b6d-9a74-5ec83552da3e
Date Created	2021-11-03 12:19
Date Modified	2021-11-03 12:19
Repository:Tag	ghcr.io/bihealth/kioscshinytest:latest
Container Port	80
Container Path	no value

1.7.4 Copy

Select a template from the top-hand dropdown menu and click Copy to make a copy of the selected template.

The screenshot shows the Kiosc Beta interface. The top navigation bar is purple with the Kiosc Beta logo, a search bar, and links for Extra Link, Help, and a user profile. The left sidebar contains icons for Home, Project Overview, Container Templates (selected), Containers, Small Files, Timeline, Background Jobs, and Members. The main content area has a breadcrumb trail: Home / TestCategory / TestProject. Below this, the title 'TestProject' is followed by a star icon. The subtitle is 'Container Templates project-wide'. A dropdown menu is open, showing '[Site-wide] Nginx Example site-wide (brndnmtthws/)' and a 'Copy' button. A 'Create' button is also visible. Below the dropdown, a light blue box contains the text: 'No container templates yet.'

A new template with the same title will show up in the list, only that the title is extended by (Copy).

KIOSC Beta

Home / TestCategory / TestProject

Successfully created container template 'Nginx Example site-wide (Copy)' from 'Nginx Example site-wide'

TestProject ☆

Container Templates project-wide

[Site-wide] Nginx Example site-wide (brndnmtthws/t) [Copy](#) [Create](#)

Title	Repository:Tag	Date Created	
Nginx Example site-wide (Copy)	brndnmtthws/nginx-echo-headers:latest	Nov. 3, 2021, 4:49 p.m.	
Shiny example	ghcr.io/bihealth/kioscshinytest:latest	Nov. 3, 2021, 12:19 p.m.	

In the dropdown menu all templates are listed the current user has access to, plus the site-wide templates. This enables the user to copy templates from other projects.

1.7.5 Controls

The Controls dropdown menu (cog icon) comprises multiple actions that can be issued on template.

Update

Duplicate

Delete

Contents

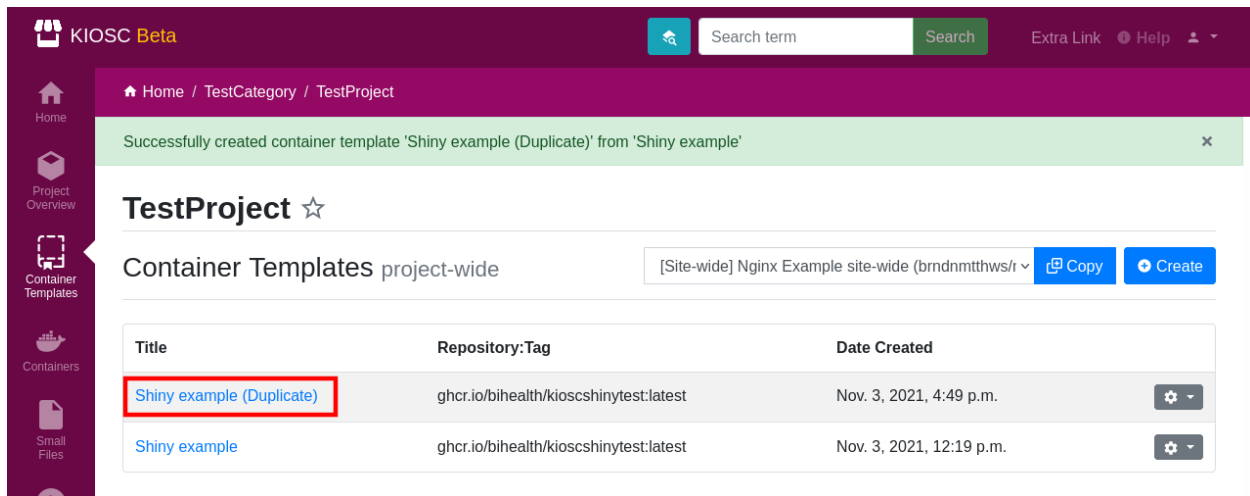
- *Controls*
 - *Update*
 - *Duplicate*
 - *Delete*

Update

Click this to enter the update form of the template.

Duplicate

Duplicate is similar to the Copy action, only that you will make a copy or duplicate of the template this menu is referring to. A new template with the same title will show up in the list, only that the title is extended by (Duplicate). The result is otherwise the same as when the user would make a copy of that template with the Copy action. This means it is just a shortcut to copy templates around within a template, while the Copy action is designed to copy templates across projects or to copy site-wide templates.



KIOSC Beta

Home / TestCategory / TestProject

Successfully created container template 'Shiny example (Duplicate)' from 'Shiny example'

TestProject ☆

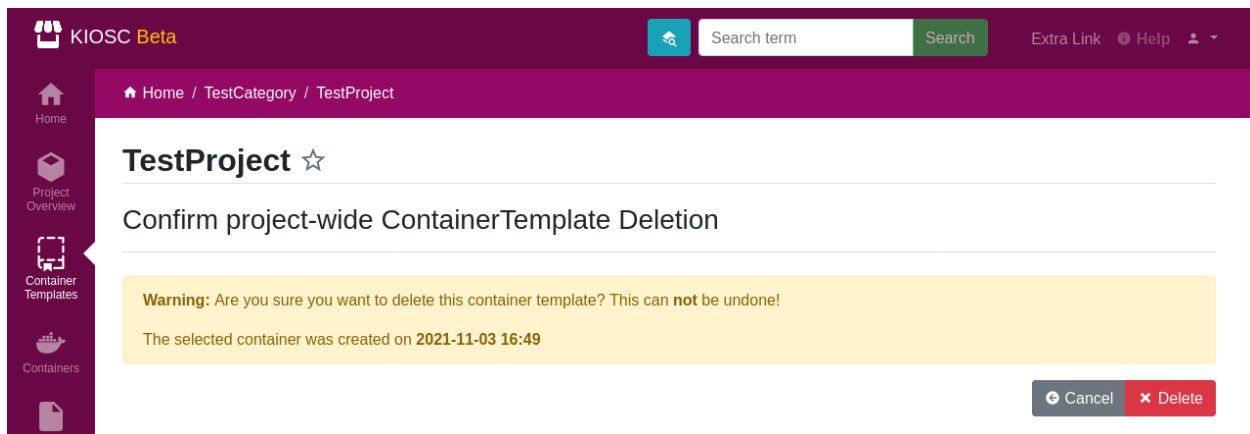
Container Templates project-wide

[Site-wide] Nginx Example site-wide (brndmtthws/t) Copy Create

Title	Repository:Tag	Date Created
Shiny example (Duplicate)	ghcr.io/bihealth/kioscshinytest:latest	Nov. 3, 2021, 4:49 p.m.
Shiny example	ghcr.io/bihealth/kioscshinytest:latest	Nov. 3, 2021, 12:19 p.m.

Delete

Delete deletes the template. This action needs a confirmation and can't be undone.



KIOSC Beta

Home / TestCategory / TestProject

TestProject ☆

Confirm project-wide ContainerTemplate Deletion

Warning: Are you sure you want to delete this container template? This can **not** be undone!

The selected container was created on 2021-11-03 16:49

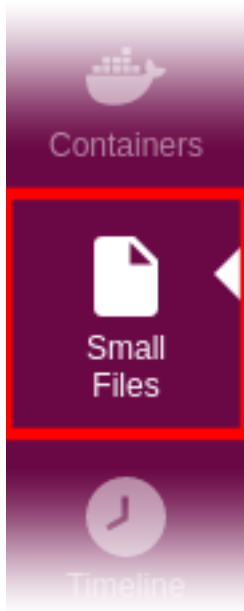
Cancel Delete

1.8 Small Files

1.8.1 Overview

Kiosc has activated an app provided by SODAR Core that allows the user to upload files of smaller size (several MB, but not GB).

Find the **Small Files** icon in the left-hand menu to open the Small Files app. This will list all uploaded files and allow to upload or delete files.




Those files then can be accessed by the containers created. For this to work, a dropdown menu of files uploaded to a project is provided during container creation that allows to insert an internal link to that file into the `command` field. This link can only be accessed by containers that are associated running inside the same project the files is associated with.

The intention is that the application running inside of the container uses links to load the files anyway, and those links now can be replaced with internal links to files that are uploaded to Kiosc. This also requires the application inside the container to accept links as file source, another way to provide the file is not possible.

For further description on how to use the app, please have a look at the [part in the SODAR Core manual](#).

When accessing the container form, the uploaded files are available in a dropdown, and a link will be inserted into the `command` field when **Insert** is clicked.

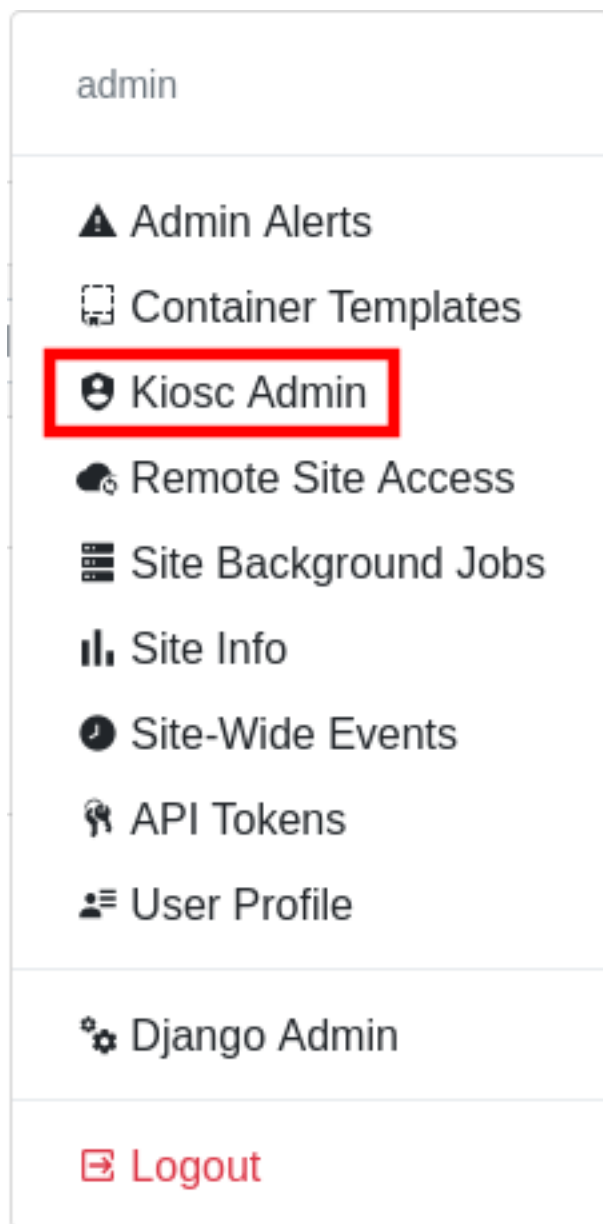
Files	/pbmc3k.h5ad	▼	 Insert
-------	--------------	---	--

1.9 Overview

Contents

- *Overview*
 - *Containers*
 - *Not in Kiosc*
 - *Other Docker Entities*

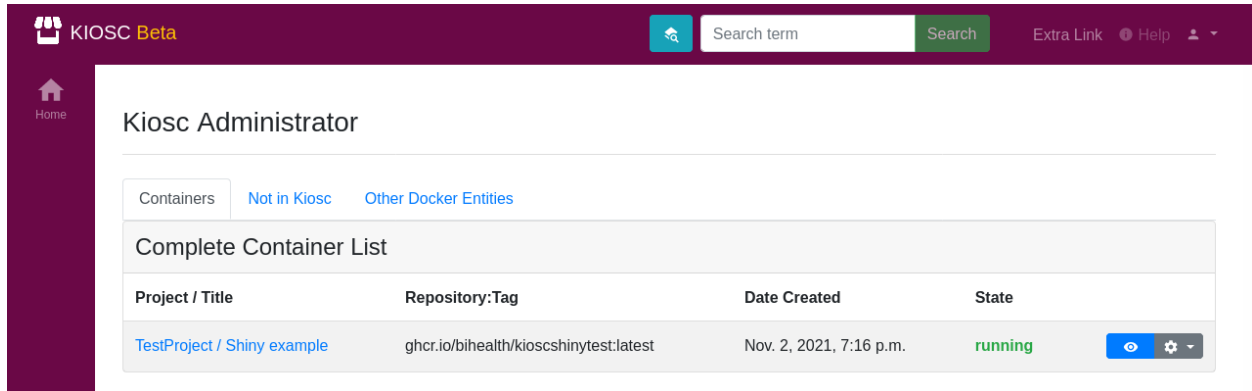
To access the administration interface for Kiosc containers (which is not to confuse with the Django administration interface), click on the user menu in the top right corner and choose *Kiosc Admin*.



This will open a page with three tabs: **Containers**, **Not in Kiosc** and **Other Docker Entities**.

1.9.1 Containers

This tab lists all available containers that were created in Kiosc, with their current status and the control menu to adjust the status of the container, or delete it.

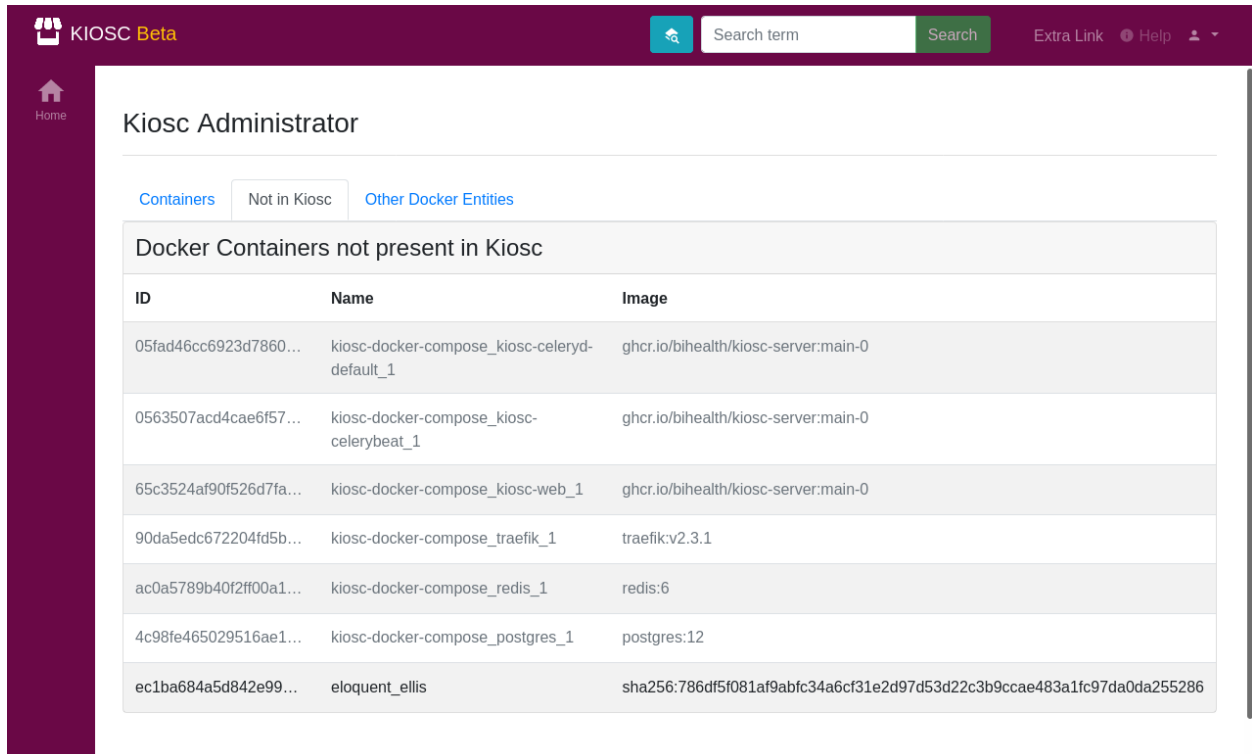


The screenshot shows the Kiosc Administrator web interface. The top navigation bar is dark purple with the Kiosc Beta logo, a search bar, and links for Extra Link, Help, and a user profile. The left sidebar has a 'Home' link. The main content area is titled 'Kiosc Administrator' and features three tabs: 'Containers' (selected), 'Not in Kiosc', and 'Other Docker Entities'. Below the tabs is a section titled 'Complete Container List' containing a table with the following data:

Project / Title	Repository:Tag	Date Created	State	
TestProject / Shiny example	ghcr.io/bihealth/kioscshinytest:latest	Nov. 2, 2021, 7:16 p.m.	running	

1.9.2 Not in Kiosc

This tab lists all running Docker containers that are not connected to a container object in Kiosc. This is meant to list orphaned Docker containers, but it will also keep a list of Docker containers that are part of the Kiosc server itself. They are grayed out.



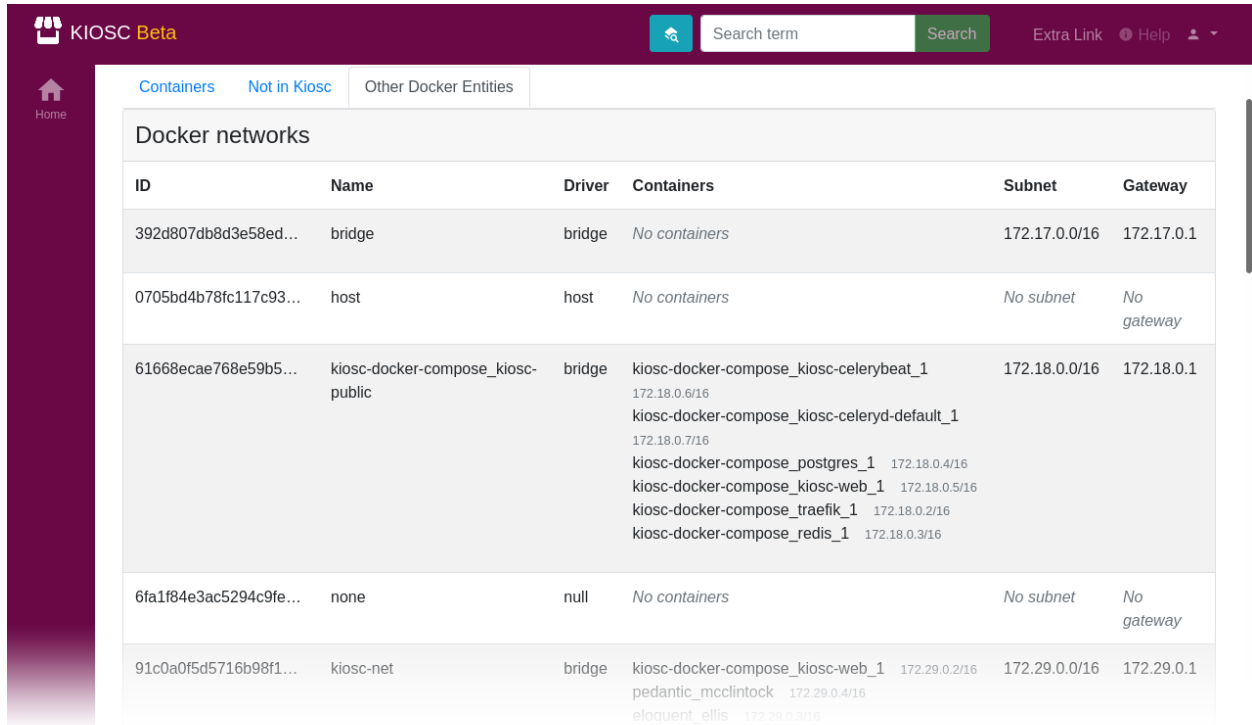
The screenshot shows the Kiosc Administrator web interface with the 'Not in Kiosc' tab selected. The main content area is titled 'Kiosc Administrator' and features three tabs: 'Containers', 'Not in Kiosc' (selected), and 'Other Docker Entities'. Below the tabs is a section titled 'Docker Containers not present in Kiosc' containing a table with the following data:

ID	Name	Image
05fad46cc6923d7860...	kiosc-docker-compose_kiosc-celeryd-default_1	ghcr.io/bihealth/kiosc-server:main-0
0563507acd4cae6f57...	kiosc-docker-compose_kiosc-celerybeat_1	ghcr.io/bihealth/kiosc-server:main-0
65c3524af90f526d7fa...	kiosc-docker-compose_kiosc-web_1	ghcr.io/bihealth/kiosc-server:main-0
90da5edc672204fd5b...	kiosc-docker-compose_traefik_1	traefik:v2.3.1
ac0a5789b40f2ff00a1...	kiosc-docker-compose_redis_1	redis:6
4c98fe465029516ae1...	kiosc-docker-compose_postgres_1	postgres:12
ec1ba684a5d842e99...	eloquent_ellis	sha256:786df5f081af9abfc34a6cf31e2d97d53d22c3b9ccae483a1fc97da0da255286

1.9.3 Other Docker Entities

This tab lists other Docker entities that are not containers and comprises three lists, **Docker networks**, **Docker volumes** and **Docker images**.

Docker networks list among other the identifier and the connected containers as well as the subnet and gateway of the network.



The screenshot shows the KIOSC Beta web interface. The top navigation bar includes the KIOSC Beta logo, a search bar, and links for Extra Link, Help, and a user profile. The left sidebar has a Home link. The main content area is divided into three tabs: Containers, Not in Kiosk, and Other Docker Entities. The 'Other Docker Entities' tab is active, displaying a table titled 'Docker networks'.

ID	Name	Driver	Containers	Subnet	Gateway
392d807db8d3e58ed...	bridge	bridge	No containers	172.17.0.0/16	172.17.0.1
0705bd4b78fc117c93...	host	host	No containers	No subnet	No gateway
61668ecae768e59b5...	kiosc-docker-compose_kiosc-public	bridge	kiosc-docker-compose_kiosc-celerybeat_1 172.18.0.6/16 kiosc-docker-compose_kiosc-celeryd-default_1 172.18.0.7/16 kiosc-docker-compose_postgres_1 172.18.0.4/16 kiosc-docker-compose_kiosc-web_1 172.18.0.5/16 kiosc-docker-compose_traefik_1 172.18.0.2/16 kiosc-docker-compose_redis_1 172.18.0.3/16	172.18.0.0/16	172.18.0.1
6fa1f84e3ac5294c9fe...	none	null	No containers	No subnet	No gateway
91c0a0f5d5716b98f1...	kiosc-net	bridge	kiosc-docker-compose_kiosc-web_1 172.29.0.2/16 pedantic_mcclintock 172.29.0.4/16 eloquent_ellis 172.29.0.3/16	172.29.0.0/16	172.29.0.1

Docker images list the identifier and the repository and tag name.

Home	
Docker images	
ID	Repository
sha256:db00b4541a8cd987e0b69d9335a49b284bf...	ghcr.io/bihealth/kiosc-server:main-0
sha256:9c3a0a891e52f5680e13693cf7ad78a3151...	
sha256:786df5f081af9abfc34a6cf31e2d97d53d22c...	ghcr.io/bihealth/kiosc-seapiper-demo:latest
sha256:4b615687ea3e055dac965057aba64b938c...	
sha256:d7a0f776c201e5d8b5a9cfc759841173e88...	ghcr.io/bihealth/kiosc-example-dash:main-0
sha256:25c18f76624c36f99de68c9bcb30b2dc14f...	ghcr.io/bihealth/kiosc-example-dash:master-0
sha256:effb55a7fc0456d829d1f7a9284c6a094b65...	ghcr.io/bihealth/kioscshinytest:latest
sha256:2bd0edd01153938fc4733c8137945dc4bb8...	ghcr.io/bihealth/kiosc-server:dev-0
sha256:9a84fe8f1cdfc6effe617f351c32e7454eb69...	ghcr.io/bihealth/docker-shiny-example:1-0
sha256:aa4d65e670d6518e5da96ca9d1a76370a9...	redis:6
sha256:0f698a6badfa74ce8a84a7c5b593348578b...	postgres:12

Docker volumes list the identifier and mountpoint.

Home	
Docker volumes	
Name	Mountpoint
008e2184010dec873c12...	/var/lib/docker/volumes/008e2184010dec873c12ade13b842062eb995d7cf1ce9adfae9aefc553548dde/_data
1dd6d4e2399ad5e4b85c...	/var/lib/docker/volumes/1dd6d4e2399ad5e4b85c5858e36c7b5d55cc4b4aa68bd391369d153e1b7812cc/_data
659957e74c24a96b565d...	/var/lib/docker/volumes/659957e74c24a96b565d338a39a461918ca859266fc273ab8478c09341b9c253/_data
6dc49770784dd11dfe49...	/var/lib/docker/volumes/6dc49770784dd11dfe492c29c82c667d84e7280fdeb24cf99acf6ee2d64180e4/_data
bc21decc97593598c906...	/var/lib/docker/volumes/bc21decc97593598c9060a0492411d27316c47a13f32543505b332f3853b9cb/_data
f6815d13158e4d38607d...	/var/lib/docker/volumes/f6815d13158e4d38607d92bcc7c874cd16b9cd7df945a09133690f3a4122724d/_data
0fea814e291d6ee0aac9...	/var/lib/docker/volumes/0fea814e291d6ee0aac981ab44017670e7eb7ca0c6a66f25580235535d170ca2/_data
4ecd5cc433691bd48a65...	/var/lib/docker/volumes/4ecd5cc433691bd48a65e5d3a121a7505d50aa3cf2560980df3212218b872ef3/_data
534c6db8510aec9d1625...	/var/lib/docker/volumes/534c6db8510aec9d16259badb08b1e3c9f820e91a16ba7db285a7689f45a12b5/_data
839855b5499fe60e3a52...	/var/lib/docker/volumes/839855b5499fe60e3a5249c47a3e84369d049154689d4aa02118333f1fe039bc/_data
8b29b0981f2020232adb...	/var/lib/docker/volumes/8b29b0981f2020232adb6e931100c1210515bac5756fecb9c8460470ca8a7b20/_data

1.10 Commands

Contents

- *Commands*
 - *Remove Stopped Containers*
 - *Stop All Containers*
 - *Stop Unused Containers*

1.10.1 Remove Stopped Containers

Usage: `python manage.py remove_stopped --remove`

This command removes all of the stopped containers. To prevent accidentally deleting containers, the `--remove` parameter has to be provided. Omitting this parameter only dry-runs the command.

1.10.2 Stop All Containers

Usage: `python manage.py stop_all`

This command sets all containers to `exited` status, no matter their current state.

1.10.3 Stop Unused Containers

Usage: `python manage.py stop_unused`

This command stops all containers that haven't been accessed by the reverse proxy for a defined period of time (the parameter can be set in the container object itself, but there is an upper limit of 7 days).

1.11 Periodic Tasks

The user has no influence on periodic tasks, but the periodic tasks in return influence the user experience.

They are designed to keep the Docker containers consistent with the information the user enters into the web interface, and to fetch logs and information from the Docker containers.

Contents

- *Periodic Tasks*
 - *Get logs and status from Docker container*
 - *Synchronize Docker container state with last user action*
 - *Stop inactive containers*
 - *Synchronize with upstream SODAR instance (if configured)*

1.11.1 Get logs and status from Docker container

Runs every 30 seconds.

This task fetches the logs that are provided by the Docker container, which includes logs from whatever runs inside of the Docker container. It also sets the Docker container status in the container database object, which means that the Docker container can change without the users intention (e.g. in case the Docker container exists unexpectedly).

1.11.2 Synchronize Docker container state with last user action

Runs every minute.

This task synchronizes the last user action performed on the container with the actual Docker container status if and only if this information differs. For example, if the Docker container exited for whatever reason but the last user action was to start the container, the task tries to run the Docker container.

1.11.3 Stop inactive containers

Runs every day at 1:11am.

This task stops inactive containers that were not accessed by the proxy for a defined period of time. This can be set by the user for each container individually, but there is maximum of 7 days. If the user omits the setting, it defaults to the 7 days maximum.

1.11.4 Synchronize with upstream SODAR instance (if configured)

Runs every five minutes.

This task synchronizes the projects and users with the upstream SODAR instance. Only if this site is in target mode.

1.12 Overview

1.12.1 Requirements

A token is required to validate against the REST API. When logged in to Kiosc, you can find the **API Tokens** link in your user dropdown menu in the top right corner of the site. Select **Create Token** from the **Token Operations** dropdown to create a new token. You will only see the token once, so make sure to copy it to clipboard at this point. Deleting existing tokens can be done from the token list.

1.12.2 Legend

The token used in the following examples is 1234567890abcdef but the generated token will be much longer.

The project SODAR UUID used in the following examples is 00000000-0000-0000-0000-000000000000.

The container SODAR UUID used in the following examples is cccccccc-cccc-cccc-cccc-cccccccccccc.

1.13 Containers

Contents

- *Containers*
 - *Create Container*
 - *Delete Container*
 - *List Containers*
 - *Container Details*
 - *Start Container*
 - *Stop Container*

1.13.1 Create Container

Create container in a project.

URL	/containers/api/create/<PROJECT_SODAR_UUID>
Method	POST
Data type	JSON dictionary

Example cURL:

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: token_
↪1234567890abcdef" --data '{"title": "Nginx echo headers", "repository": "brndnmtthws/
↪nginx-echo-headers", "tag": "latest"}' https://kiosc.bihealth.org/containers/api/
↪create/00000000-0000-0000-0000-000000000000
```

1.13.2 Delete Container

Delete container from a project given a container SODAR UUID.

URL	/containers/api/delete/<CONTAINER_SODAR_UUID>
Method	DELETE

Example cURL:

```
curl -X DELETE -H "Authorization: token 1234567890abcdef" http://kiosc.bihealth.org/
↪containers/api/delete/cccccccc-cccc-cccc-cccc-cccccccccccc
```


1.13.3 List Containers

List all containers available in a project.

URL	/containers/api/<PROJECT_SODAR_UUID>
Method	GET

Example cURL:

```
curl -H "Authorization: token 1234567890abcdef" http://kiosc.bihealth.org/containers/api/
↪000000000-0000-0000-0000-000000000000
```

1.13.4 Container Details

Show details and logs of a given container.

URL	/containers/api/detail/<CONTAINER_SODAR_UUID>
Method	GET

Example cURL:

```
curl -H "Authorization: token 1234567890abcdef" http://kiosc.bihealth.org/containers/api/
↪detail/cccccccc-cccc-cccc-cccc-cccccccccccc
```

1.13.5 Start Container

Start a given container.

URL	/containers/api/start/<CONTAINER_SODAR_UUID>
Method	GET

Example cURL:

```
curl -H "Authorization: token 1234567890abcdef" http://kiosc.bihealth.org/containers/api/
↪start/cccccccc-cccc-cccc-cccc-cccccccccccc
```

1.13.6 Stop Container

Stop a given container.

URL	/containers/api/stop/<CONTAINER_SODAR_UUID>
Method	GET

Example cURL:

```
curl -H "Authorization: token 1234567890abcdef" http://kiosc.bihealth.org/containers/api/
↪stop/cccccccc-cccc-cccc-cccc-cccccccccccc
```


INDICES AND TABLES

- `genindex`
- `search`