

Integration of systems as microservices

Considering you have already implemented an interesting system that you would like to deploy as a microservice in Heureka! is simple! You can use the technologies you know. At the very beginning you have to setup a microservice environment that is very similar to the production environment. In contrast to a production system, the integration of a new microservice requires to have access to


- the APIs of the already existing microservices
- the NATS message broker
- the microservice routing proxy to add your new service to the architecture

You can setup the environment by using the [Heureka! console](#). Pull it from github and follow the [steps described by the readme](#) (How to add a new microservice).

Afterwards, you can start your local applications as part of the microservice configuration. You will have access to the internal docker network, although you are not deploying your applications using docker (consider the [configuration of local applications](#)).

Available APIs

You are running a current microservice configuration, including [drops](#) and [arise](#). Most important the development of a new microservice is the usage of the [drops API](#) to read users and their social structure and the [OAuth-Handshake to initiate a shared session](#). Using the default configuration of a newly created microservice, the Drops API will be available at `172.3.0.2:9000` from the internal docker network or at `localhost:9000/drops` sending the request through the nginx proxy.



Please, take a look at these interfaces to discover available data objects. We strongly suggest to not inspect the database structures, since there could be different technologies in use and it would break the concept of microservices to directly use the databases of other MS.

Just as well, the [NATS](#) message broker is used by the Heureka! microservice configuration to exchange messages between the services. It is available at `172.3.150.1:4222` from the internal network or at `localhost:4222` sending the request through the nginx proxy.

Integration in the architecture

The two main challenges to integrate a new service are

- Having a shared session
- Develop UI elements that are reusable by other services

Currently, the shared session challenge is addressed by the Heureka!-OAuth2 handshake. The [OAuth2 handshake HowTo](#) explains the integration in the shared session. The [widgets HowTo](#) explains the implementation of reusable UI elements and also their usage.

You can develop the UI by the technologies you know! Thus, you will not be limited and can use the framework of your choice. Just understand and use the [existing widgets](#) to extend your UI by elements provided by the other microservices. Additionally, you can use a basic CSS to style your microservice interface the same way as the rest of the application and you can [implement your own reusable user interface elements](#) as widgets that can be used by other microservice developers.

Docker integration

The setup of a new microservice as part of the Heureka! architecture is described in the [README of the Heureka! CLI](#).

There is no need to run your application(s) as a docker container. You can just run your applications on distinct ports and configure the Nginx to point on these ports.

If any additional service is required (e.g. database), just set it up as a docker container and initiate a connection by its IP address.

Consider every hard coded HTTP call! If there are some calls addressing `localhost`, you probably will have to replace them, since `localhost` addresses the docker container of your application. So, to address other microservices, you should use the internal IP address of the corresponding docker container. Services that are deployed next to the web server of your application should also be deployed in docker containers having an internal IP address.

Consider that your microservice will be available behind a specific path configured in the Nginx. For some frameworks it is required to configure the path name as a base path. If it is required and not properly configured, you will mostly see a whitescreen.

Basic styling

Additionally, you have to use the shared CSS by adding

```
<link rel="stylesheet" type="text/css" media="screen" href="/dispenser/css/vca.css">
```

to your HTML files.

You can use basic page elements to replicate the corporate design that is already implemented. These are delivered as widgets by `vca-widget-base`.

Showing the navigation

The navigation in the default layout is shown as part of a header and a footer. These elements are implemented as widgets ([heureka-widget-navigation](#)) and can be integrated into the page that is rendered by your application. See the [HowTo widgets](#) article for a detailed explanation. Subsequently is shown an example for a [vue.js](#) application:

```
<template>
  <div id="ms-frontend-app">
    <WidgetTopNavigation />
    <div id="content">
      <router-view/>
    </div>
    <WidgetBottomNavigation />
  </div>
</template>

<script>
import { WidgetTopNavigation, WidgetBottomNavigation } from 'heureka-widget-navigation-2021';
import 'heureka-widget-navigation-2021/dist/heureka-widget-navigation-2021.css'

export default {
  name: 'ms-frontend-app',
  components: { WidgetTopNavigation, WidgetBottomNavigation }
}
</script>

<style lang="less">
#arise {
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  color: #2c3e50;
  flex: 1;
  display: flex;
  flex-direction: column;
  min-height: min-content;
}
#content {
  flex-grow: 1;
  flex-shrink: 0;
  display: flex;
  overflow: auto;
}
</style>
```

Integration in the navigation

Cloning the [Heureka! console](#) will create the `<path-to-heureka-console>/microservices/ms-<name>/docker-conf/navigation/GlobalNav.json` and `<path-to-heureka-console>/microservices/ms-<name>/docker-conf/navigation/noSignIn.json`. Change the `GlobalNav.json` (menue after a successful login) to add a menue entry for pages of your new microservice and to the `noSignIn.json`, if the entry should be available without an established session.

Subsequently, you have to reload your navigation by calling `GET http://<your-host>/dispenser/navigation/init`.

Integration in documentation

First of all: Use your repository and the readme of your microservice' project to document specific functions of your service. If you want to write an How-To or any other more general documentation, you can simply add a [Markdown](#) file to the [pages](#) directory of the [documentation repository](#) of the Heureka! project.

The documentation uses [GRAV CMS](#) to generate this documentation. Thus, also the [images](#) and the [languages](#) directories in the repository are used and could be filled with content.

Grav was </> with  by [Trilby Media](#).