Article
Dmitry Maslennikov · Apr 19, 2017 6m read

Containerization Caché - let's add our application

In my previous article, I gave an example of how to get the own image with InterSystems Caché. Now it's time to launch a project with container.

To launch a Caché project in a container we will use an image from the previous article as a source for the new one.

But before it can be used, it must be published publicly or privately in the repository. In our company, we use GitLab as a storage for sources for all of our projects. And it can be used as a registry for Docker images as well. We must have authorization to push our image to any registry.

To authorize on a default registry, use this command.

```
docker login
```

Or with an address of the registry, which will be used as a storage for an image.

```
docker login registry.gitlab.com
```

Authorization is also needed to fetch images from the registry on another server (e.g. if they are not public).

Build the image from the previous article with a new name, which could include registry server name, if it is not hub.docker.com, used by default. Followed by name of user or organization, and name of the particular image with version.

```
docker build -t daimor/ensemble:2016.2 .
```

As you may already know, the release of new Ensemble version 2017.1 was announced since the previous article was published. Here is the command to build with version 2017.1 container.

```
docker build -t daimor/ensemble:2017.1 --build-arg WRC_USERNAME=****** --build-arg WR
C_PASSWORD=****** --build-arg cache=ensemble-2017.1.0.792.0 -f Dockerfile.WRC .
```

Push our image to the registry after build.

```
docker push daimor/ensemble:2016.2
docker push daimor/ensemble:2017.1
```

Version 2017.1 will make as a latest as well.

```
docker tag daimor/ensemble:2017.1 daimor/ensemble:latest
docker push daimor/ensemble:latest
```

Installing application.

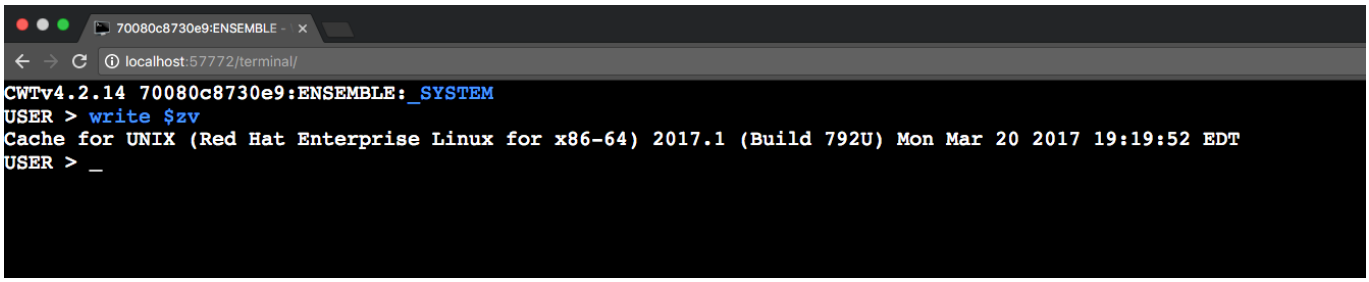
Now it's time to install an application that will live in our Caché container. Let's start with a fairly simple application to install - Caché WebTerminal.

```
# Use our image with Ensemble as a source, we can particular version, or just latest,
 which will be used as well if omit version.
FROM daimor/intersystems-ensemble:latest
# FROM daimor/intersystems-ensemble:2016.2
# FROM daimor/intersystems-ensemble:2017.1
# FROM daimor/intersystems-ensemble equal to FROM daimor/intersystems-ensemble:latest
# Version of WebTerminal to install
ARG TerminalVersion=4.2.14
# Create temporary folder
RUN mkdir /tmp/webterminal \
# Download installation xml from github
 && curl http://intersystems-ru.github.io/webterminal/files/WebTerminal-
v$TerminalVersion.xml -o /tmp/webterminal/webterminal.xml \
# Start Caché Instance
 && ccontrol start $ISC_PACKAGE_INSTANCENAME \
# Generate login and password for csession if needed, and Load downloaded xml with co
mpilation
# WebTerminal will be installed during compilation process
 && printf "_SYSTEM\n$ISC_PACKAGE_USER_PASSWORD\n" \
 |  csession $ISC_PACKAGE_INSTANCENAME -UUSER "##class(%SYSTEM.OBJ).Load(\"/tmp/webte
rminal/webterminal.xml\",\"cdk\")" \
# Stop Caché instance
 && ccontrol stop $ISC_PACKAGE_INSTANCENAME quietly \
# Clean Temporary folder
 && rm -rf /tmp/webterminal/
```

Build and launch.

```
docker build -t terminal .
docker run -d -p 57772:57772 --name webterminal terminal
```

And finally, we are ready to open our application by link 'localhost:57772/terminal/'



It is also possible to get access to csession inside of our container.

```
> docker exec -it webterminal csession ensemble
Node: 34272368af61, Instance: ENSEMBLE
Username: _system
Password: *********
USER>
USER>w $zv
Cache for UNIX (Red Hat Enterprise Linux for x86-64) 2017.1 (Build 792U) Mon Mar 20 2
```

```
017 19:19:52 EDT
USER>
```

Advanced installation

Thanks to @Nikita.Savchenko who developed WebTerminal and made it so easy to install. Let's install another app which cals for a few more steps to be taken.

Next I will install DeepSeeWeb which depends on another project MDX2JSON and which should be installed before DeepSeeWeb obviously.

First we should prepare Installer Manifest. Installer, by the way, is quite simple: prepare namespace, load sources for both of projects, and launch its installers.

```
Class DSWMDX2JSON.Installer
{
XData setup [ XMLNamespace = INSTALLER ]
{
<Manifest>
    <Var Name="Namespace" Value="MDX2JSON"/>
    <Var Name="Import" Value="0"/>
    <If Condition='(##class(Config.Namespaces).Exists("${Namespace}")=0)'>
      <Log Text="Creating namespace ${Namespace}" Level="0"/>
      <Namespace Name="${Namespace}" Create="yes" Code="${Namespace}" Ensemble="" Dat
a="${Namespace}">
        <Configuration>
          <Database Name="${Namespace}"
            Dir="${MGRDIR}/${Namespace}"
            Create="yes"
            Resource="%DB_${Namespace}"
            PublicPermissions="RW"
            MountAtStartup="true"/>
        </Configuration>
      </Namespace>
      <Log Text="End Creating namespace ${Namespace}" Level="0"/>
    </If>
    <Namespace Name="${Namespace}">
      <Import File="/tmp/deps/Cache-MDX2JSON-
master/MDX2JSON/" Flags="ck" Recurse="1"/>
    </Namespace>
    <Namespace Name="${CURRENTNS}">
      <Import File="/tmp/deps/Cache-MDX2JSON-
master/MDX2JSON/Installer.cls.xml" Flags="ck"/>
      <Import File="/tmp/deps/deepseeweb.xml" Flags="ck"/>
      <RunInstall Class="MDX2JSON.Installer" Method="setup"/>
      <RunInstall Class="DSW.Installer" Method="setup"/>
    </Namespace>
</Manifest>
}
ClassMethod setup(
    ByRef pVars,
    pLogLevel As %Integer = 3,
    pInstaller As %Installer.Installer,
    pLogger As %Installer.AbstractLogger
  ) As %Status [ CodeMode = objectgenerator, Internal ]
{
```

```
    do %code.WriteLine($char(9)_"set pVars(""CURRENTCLASS"")=""_%classname_"""")
    do %code.WriteLine($char(9)_"set pVars(""CURRENTNS"")=""_$namespace_"""")
  #; Let our XGL document generate code for this method.
  Quit ##class(%Installer.Manifest).%Generate(%compiledclass, %code, "setup")
}
}
```

Save this class as Installer.cls. Then we need a CacheObject Script which will call that installer. Save it as install.scr

```
// install charset for CSP files as recommended
set ^%SYS("CSP","DefaultFileCharset")="utf-8"
// Load Installer
do $system.OBJ.Load("/tmp/deps/Installer.cls","ck")
// Setup
do ##class(DSWMDX2JSON.Installer).setup(.vars,3)
halt
```

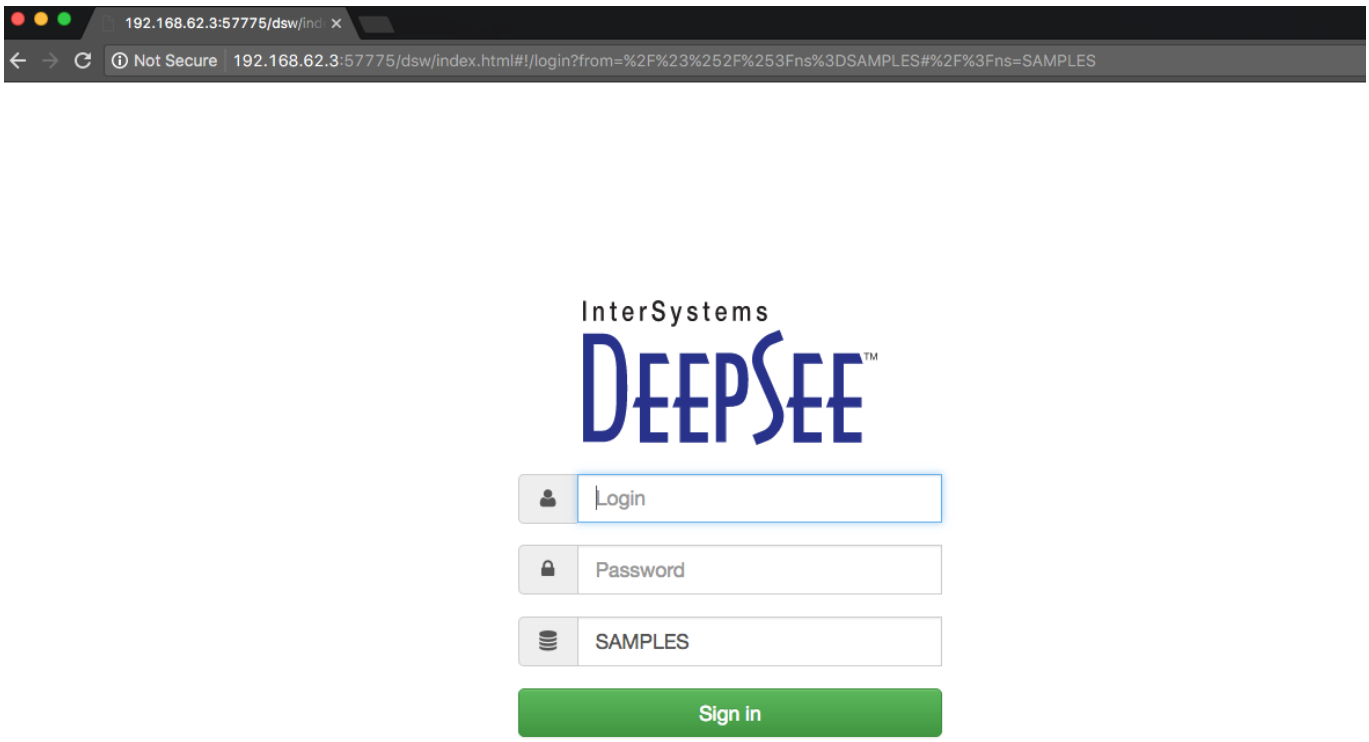
And finally our Dockerfile

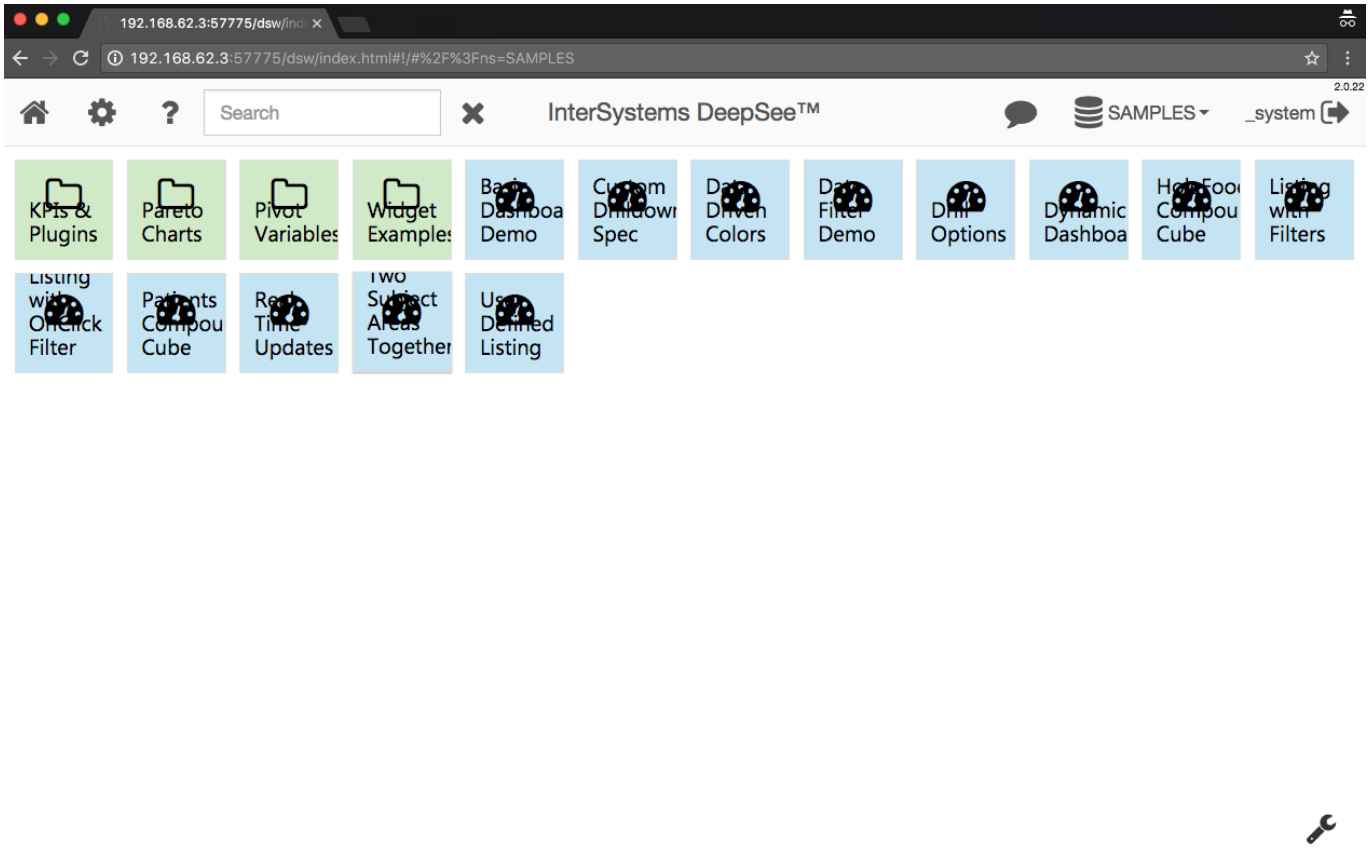
```
FROM daimor/intersystems-ensemble:latest
# version DeepSeeWeb
ARG DSW_VERSION=2.0.22
COPY cache.key /opt/ensemble/mgr/
COPY install.scr /tmp
COPY Installer.cls /tmp/deps/
# Temporary folder
RUN mkdir -p /tmp/deps \
 && cd /tmp/deps \
 # Download MDX2JSON, just master branch from github as archive
 && curl -L -q https://github.com/intersystems-ru/Cache-
MDX2JSON/archive/master.tar.gz | tar xvfzC - . \
 # Download DeepSeeWeb from releases
 && curl -L -q https://github.com/intersystems-ru/DeepSeeWeb/releases/download/${DSW_
VERSION}/DSW.Installer.${DSW_VERSION}.xml -o deepseeweb.xml \
 # Start Caché
 && ccontrol start ensemble \
 # add login and password for csession in our installer script
 && sed -i "1s/^/_SYSTEM\n$ISC_PACKAGE_USER_PASSWORD\n/" /tmp/install.scr \
 # run install script
 && csession ensemble < /tmp/install.scr \
 # Sstop Cache
 && ccontrol stop ensemble quietly \
 # clean temporary folder
 && rm -rf /tmp/deps
WORKDIR /opt/deepsee
```

And everything is ready to build and run

```
docker build -t deepseeweb .
docker run -d -p 57775:57772 deepseeweb
```

We are now ready to open Installed DeepSeeWeb.





That's it. As a result we have the docker image which includes Web Terminal and DeepSee Web. Sources are available on github.

#Cloud #Containerization #Docker #System Administration #Terminal #Caché

Source URL: https://community.intersystems.com/post/containerization-cach%C3%A9-lets-add-our-application