The Compatible One Application and Platform Service[[1]](#footnote-1) (COAPS) API

User Guide

Using the COAPS API (v1.5.3) to provision and manage applications on Cloud Foundry

Telecom SudParis, Computer Science Department



**Contributors**:

Mohamed Sellami, Telecom SudParis.

Nguyen Ngoc Chan, Telecom SudParis.

Sami Yangui, Telecom SudParis.

Mohamed Mohamed, Telecom SudParis.

Samir Tata, Telecom SudParis.

Contents

[1 Introduction 3](#_Toc356314576)

[2 Building the Cloud Foundry-COAPS API 3](#_Toc356314577)

[3 Deploying the Cloud Foundry-COAPS API 3](#_Toc356314578)

[4 Interacting with the Cloud Foundry-COAPS API 4](#_Toc356314579)

[4.1 Using CURL 4](#_Toc356314580)

[4.2 Using the Web-based client 5](#_Toc356314581)

[5 Handling applications 11](#_Toc356314582)

[5.1 Configuration for remote application storage 11](#_Toc356314583)

[5.2 Configuration for application metadata management 11](#_Toc356314584)

[5.3 Uploading an application on the Cloud 11](#_Toc356314585)

[5.4 An application to handle files stored on the remote server 12](#_Toc356314586)

[5.4.1 Uploading an application: 12](#_Toc356314587)

[5.4.2 Handling applications: 13](#_Toc356314588)

# Introduction

The COAPS API is a generic REST API that allows a seamless interaction with different and heterogeneous PaaS. In this aim, it exposes a generic interface that can be implemented according to the different actions exposed by a PaaS (e.g. Cloud Foundry, Openshift, etc.). In this user guide, we present a COAPS API implementation to interact with a Cloud Foundry PaaS. The different steps to build the *CloudFoundry-COAPS API*, to install it and to test it are detailed below.

# Building the Cloud Foundry-COAPS API

1. First, use GIT to locally clone the project’s code from: <git://gitorious.ow2.org/ow2-compatibleone/coaps.git>. The cloned repository contains:
   1. **coaps/spec**: the COAPS API specifications.
   2. **coaps/ap**i: the java interfaces for the COAPS API. The current api ensures application and environment management operations.
   3. **coaps/core**: this folder contains two subfolders:
      1. **CloudFoundry-api**: the CloudFoundry-COAPS implementation.
      2. **OpenShift-api**: the OpenShift-COAPS implementation.
   4. **coaps/client**: a generic Web-based client to test the different implementations.
   5. **coaps/test-resources**: the application description (i.e manifest) and a Java Web application that can be deployed on Cloud Foundry or Openshift.
2. Build the project using maven (version 3.0.4). Go to the cloned repository’s root directory (i.e. coaps) and run ’’mvn install’’:



This action will build, among others, a Web-based implementation of the CloudFoundry-COAPS API (CF-api.war in the coaps/core/CloudFoundry-api/targetfolder**).**

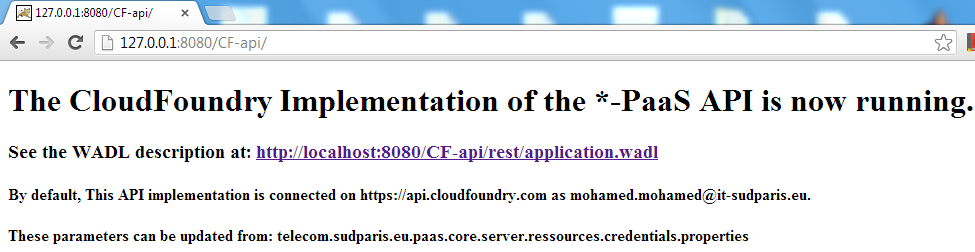
# Deploying the Cloud Foundry-COAPS API

After the maven build, a Web application (i.e. war file) is generated in the CloudFoundry-api sub-folder. This application has to be deployed on a Web server and will act as a REST server which allows interacting with a CloudFoundry PaaS.

1. To deploy the application using the apache Tomcat server, the version 7 using a java 1.6 runtime is required. Deploy the Web implementation of the CoudFoundry-api by copying the CF-api.war from GenericAPI/core/CloudFoundry-api/target to the tomcat’s webapps folder.



1. You can test the deployment of the application by typing http://{hostname}:{portNumber}/CF-api/ in your Web browser. You should get a message indicating that your CloudFoundry implementation is correctly running.



1. By default, the CloudFoundry-COAPS API will interact with the online VMware's Cloud Application Platform (i.e. <http://api.cloudfoundry.com/> ) using a test account. You can sign up for a free Cloud Foundry account on this page: <https://my.cloudfoundry.com/signup>. The connection parameters can be updated by changing the default parameters in the credentials.properties file:
   1. To modify this file, first stop the running tomcat instance and open the CF-api.war file with your favorite compression tool.
   2. Go to WEB-INF/classes/telecom/sudparis/eu/copaas/core/server/ resources/credentials.properties.
   3. Edit the file to specify your credentials. For example:

vcap.target=http://api.cloudfoundry.com

vcap.email=login

vcap.passwd=password

host=api.cloudfoundry.com

api.public.url=http:// {hostname}:{portNumber}/CF-api/rest/

* 1. Save the file and restart tomcat.

**Note**: *if the old credentials persist after a change, try to delete the extracted CF-api folder and any related temporally folders.*

# Interacting with the Cloud Foundry-COAPS API

To invoke the API actions, one can use any REST client (eg. CURL) or our Web-based client (generated in the client sub-folder).

## Using CURL

The list of available operations, their type (i.e. GET, POST, etc.) and invocation path and parameters can be viewed at: http://{hostname}:{portNumber}/CF-api/rest/application.wadl. An example of the provided WADL resources and operations description is given in Figure 1.

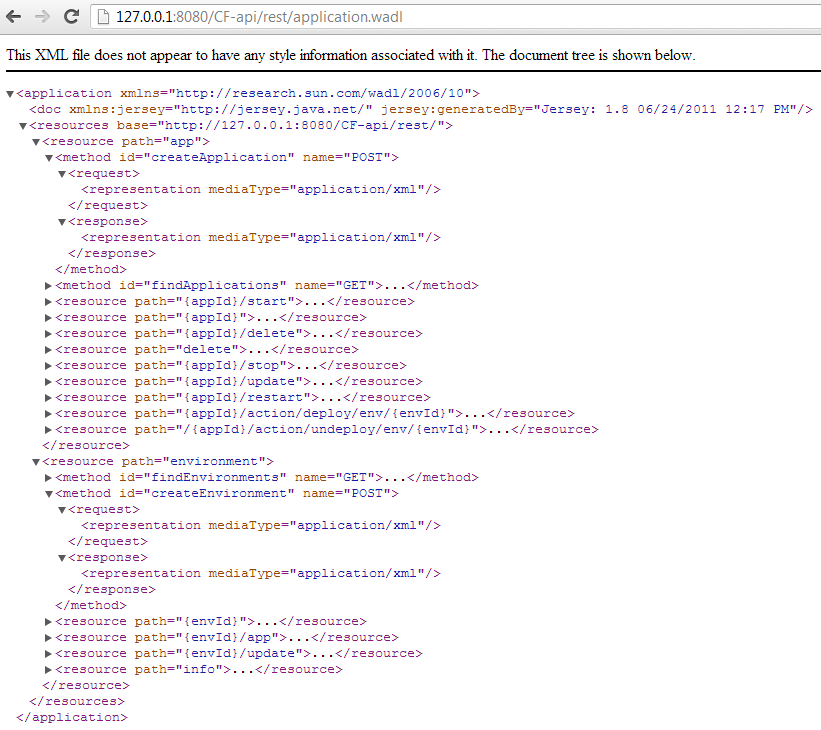


Figure 1 the WADL description of the REST resources and operations offered by the Cloud Foundry implementation

For example:

* to list all available applications on a Cloud Foundry instance we use:



* To create an environment, we have to provide its description file in XML (i.e. its manifest) through a POST:



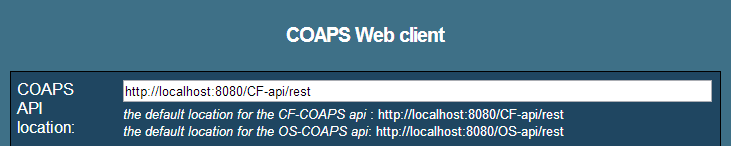
An example of a manifest is provided in coaps/test-resources.

**Note**: *If you use this manifest, do not forget to remove the license block in the beginning*.

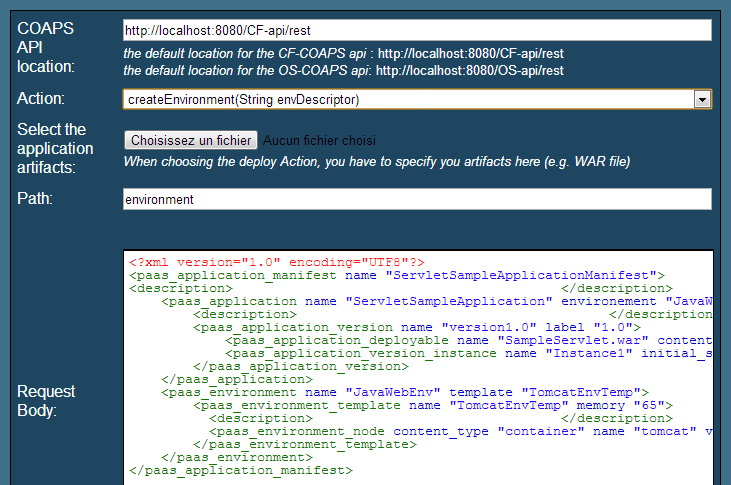
## Using the Web-based client

The web archive (client.war) of the client is available at coaps/client/target/client.war. To deploy the client, copy the client.war to the tomcat’s webapps folder. The client will be available at this address: http://{hostname}:{portNumber}/client/. In the following, we describe an application provisioning use case on Cloud Foundry:

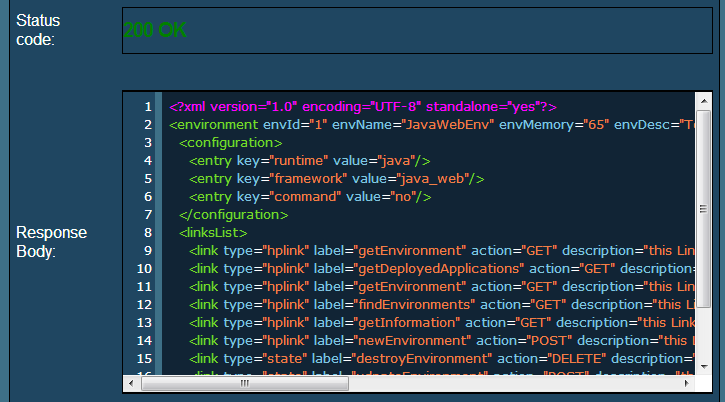
1. Launch the client http://{hostname}:{portNumber}/client/ from a Web browser and specify the location of your CloudFoundry-COAPS API implementation[[2]](#footnote-2)



1. **Create an environment**: select the "create environment" action from the action list and provide the manifest describing, among others, the environment needed by the application to deploy (runtimes, frameworks, etc.). In the body of the request just paste the content of coaps/test-resources/manifest.xml (without the license comment).



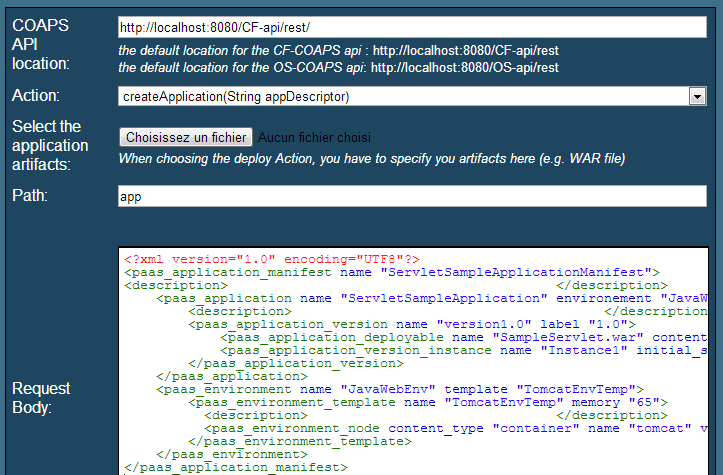
After submitting this request (i.e. click on the submit button), an XML description of the created environment is returned. From this descriptor, you have to ''save'' the returned "envid" that must be provided later to link this environment to the application to deploy (deployApplication action).



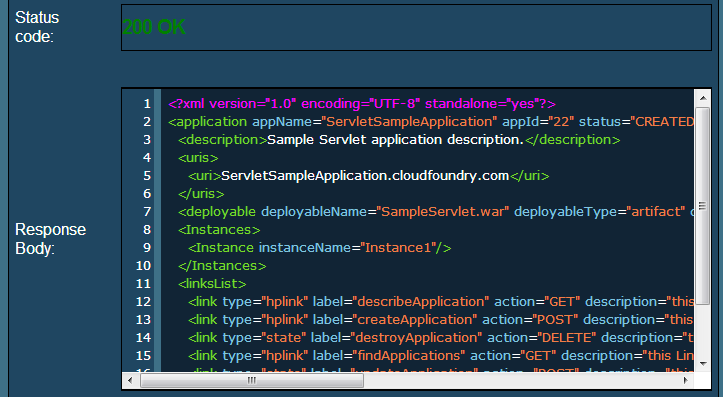
**Note**: If additional nodes are needed, a database for example, the Environment manifest must be updated accordingly. To add a mysql database in the environment for example, add the following element after the existing < paas\_environment\_node > element.

**<** **paas\_environment\_node content\_type="database" name="mysql" version="2.2" provider="CF"/>**

1. **Create the application**: select the "create application" action from the action list and provide the manifest describing the application. In the body of the request just paste the content of coaps/test-resources/manifest.xml (without the license comment).



In the same way as the created environment, an "appid" will be provided by the API.



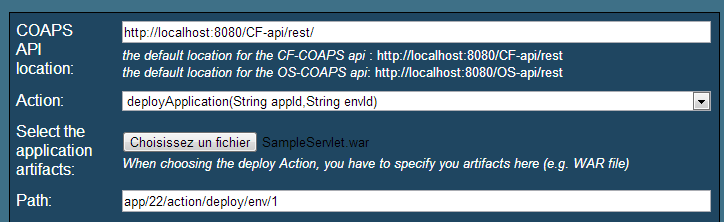
**Note 1**: If needed, you have to modify the manifest to update the name of the application and to specify the path of the application to deploy in the <paas\_application\_deployable> element.

**Note 2**: If additional instances are needed, the manifest must be updated accordingly. To declare an additional instance, add the following element after the existing <paas\_application\_version\_instance> element.

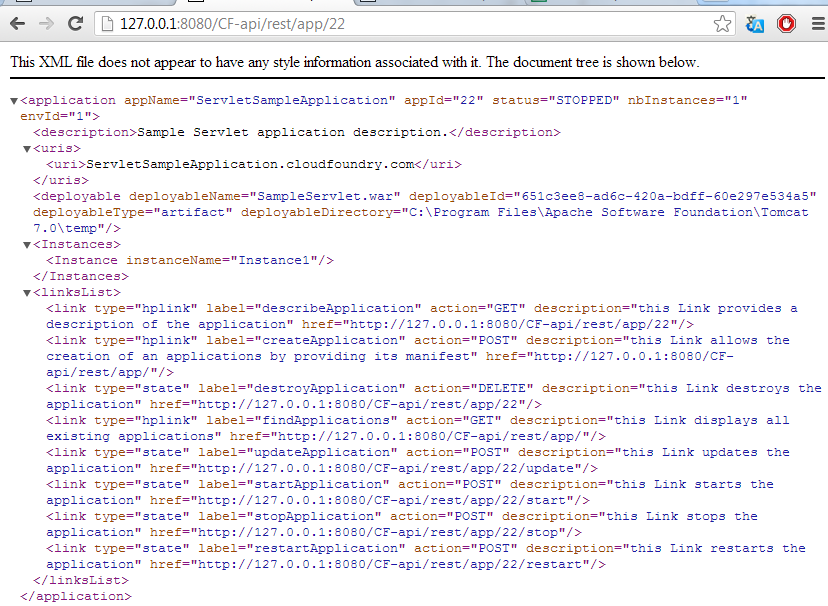
**<paas\_application\_version\_instance name="Instance2" initial\_state="1" default\_instance="false"/>**

**Note 3:** As applications can be stored locally or remotely, we provide two ways to specify the path to an application. Details are presented in section Handling applications.

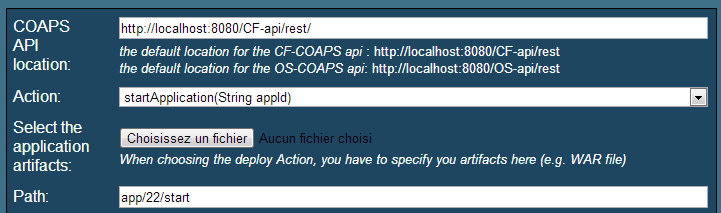
1. **Deploy the application** select the "deploy application" action from the action list while specifying the environment identifier "envid" and the application identifier "appid" in the path. You also have to join the application artifacts to deploy. In this example we are using the SampleServlet.war application.



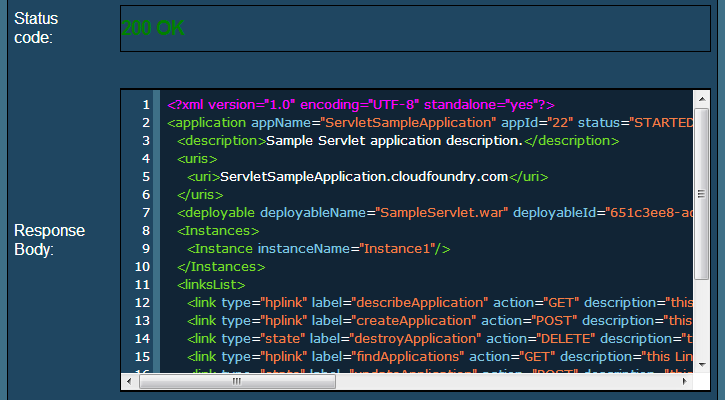
This action will upload and deploy the specified application on the Cloud Foundry PaaS. By default, the state of the application is set to “STOPPED”. As response, an XML descriptor of the deployed application is returned.



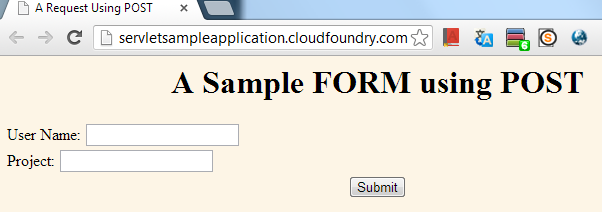
1. **Start the application**: select the "start application" action from the action list and provide the application identifier "appid" in the path.



The application is now started and its access URL <http://ServletSampleApplication.cloudfoundry.com> is available in the returned application description.



The application is now accessible via a browser on the returned URL.

  
**Note**: The provided scenario is a basic example that shows how to deploy a java Web application on a specific environment (java\_web) and start it on one instance. However, for particular needs, more constraints can be described on the manifests (i.e. jredis environment, mysql database, more instances, etc.).

**Note’**: Additional environment and application operations are also offered by the CloudFoundry-COAPS API implementation. For example: stop application, describe/destroy application/environment, etc.

# Handling Applications

The COAPS application provides two ways to upload an application on the Cloud: (1) uploading a local application and (2) uploading a remote application. In the following we present some configurations and method to upload an application on the Cloud.

## Configuration for remote application storage

The remote storage server needs to be configured to support *SSH connection*. The configuration includes the *hostname*, *port*, an account (*username/password*) to remotely access to the server, and a *physical path* to the directory that stores uploaded applications. This server should also provide an HTTP connection to support the application downloading using HTTP GET method. Thus, a *logical path* to the application repository needs also to be specified. All configuration information is declared in the *CF-api/WEB-INF/classes/telecom/sudparis/eu/copaas/core/server/ resources/credentials.properties* file.

For example, the configuration of the remote server used by the COAPS application is as follows:

deployable.host=157.159.249.218

deployable.port=22

deployable.user=ufile

deployable.pwd=1234

deployable.physicalPath=/home/ufile/UploadedFiles/

deployable.logicalPath=/filestore/

## Configuration for application metadata management

We use an XML file to store the information about applications on the remote server. This information includes application ID, description, logical name, local name, remote name, size and method to get the application.

So, we need also to specify the path to the XML file in the *CF-api/WEB-INF/classes/telecom/sudparis/eu/copaas/core/server/ resources/credentials.properties* file. For example:

deployable.fileDB=fileDB.xml

## Uploading an application on the Cloud

There are two ways to upload an application: uploading local application or uploading an application stored on the remote server.

In order to upload a local application, the developer needs to specify the path to the application stored on the local machine in the *manifest.xml* file. For example:

<paas\_application name="ServletSampleApplication" environement="JavaWebEnv">  
 <description>Sample Servlet application description.</description>  
   <paas\_application\_version name="version1.0" label="1.0">  
   <paas\_application\_deployable name="SampleServlet.war" content\_type="artifact" location="C:\Users\sellami\Desktop\Workshop \SampleServlet\deployable" multitenancy\_level="SharedInstance"/>  
   <paas\_application\_version\_instance name="Instance1" initial\_state="1" default\_instance="true"/>  
   </paas\_application\_version>  
</paas\_application>

In order to upload a remote application, the developer needs to specify the ID of the application stored in the remote server. For example:

<paas\_application name="ServletSampleApplication" environement="JavaWebEnv">  
 <description>Sample Servlet application description.</description>  
 <paas\_application\_version name="version1.0" label="1.0">  
 <paas\_application\_deployable name="SampleServlet.war" content\_type="artifact" location="2cf30c86-477d-4a5b-b5d7-66c98d785f7b" multitenancy\_level="SharedInstance"/>  
 <paas\_application\_version\_instance name="Instance1" initial\_state="1" default\_instance="true"/>  
 </paas\_application\_version>  
</paas\_application>

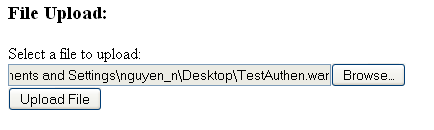
## An application to handle files stored on the remote server

We develop an independent application that allows developers to upload, view and delete files (applications) stored on the remote server. This application is available at http://coaps-marketplace.cloudfoundry.com

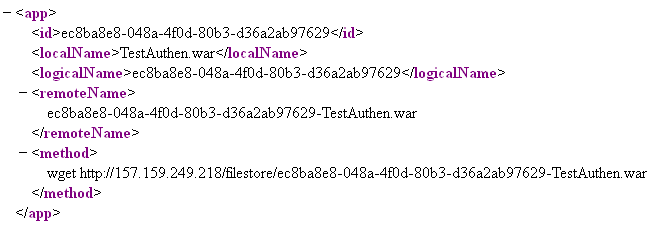
### Uploading an application

**URL:**  http://coaps-marketplace.cloudfoundry.com/fileuploading.jsp

1. The developer browses the application to be uploaded and click on the button Upload File.

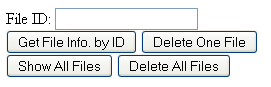


1. The application executes the uploading process and returns the stored metadata:



### Handling applications

**URL:** http://coaps-marketplace.cloudfoundry.com/ filehandling.jsp

****

Using this URL, a developer can:

1. Get information of an application by its ID.
2. Get the list of existing applications on the remote server.
3. Delete an application by its ID.
4. Delete all applications on the remote server.

1. COAPs is proposed to replace \*-PaaS.

   Compatible One Application and Platform Service (COAPS) API and user guide are licensed under a [Creative Commons Attribution 3.0 Unported License](http://creativecommons.org/licenses/by/3.0/) [↑](#footnote-ref-1)
2. Currently we provide only the Cloud Foundry and Openshift implementations. To test the Openshift-PaaS API implementation, deploy the associated war (from coaps/core/OpenShift-api/target**) on tomcat.** [↑](#footnote-ref-2)