Document Generator

Technical Deployment

*Success Solutions  
R&D*

Table of Contents

[Table of Contents 2](#_Toc164341841)

[1 BTP Deployment Guide 3](#_Toc164341842)

[1.1 BTP accesses 3](#_Toc164341843)

[1.1.1 Global Account 3](#_Toc164341844)

[1.1.2 **Subaccounts (multi-Environment)** 3](#_Toc164341845)

[1.2 Success Factors 4](#_Toc164341846)

[1.2 BTP instance log 5](#_Toc164341847)

[2 Preparing the App for Deployment 6](#_Toc164341848)

[Backend Document Generator 6](#_Toc164341849)

[Placeholderengine\_v2 BE 8](#_Toc164341850)

[HromadneGenerovani FrontEnd 9](#_Toc164341851)

[DG -Config / Placeholderengine-V2-config 10](#_Toc164341852)

[3 Logging in and setting CF. 11](#_Toc164341853)

[4 Preparing for deployment (deploy.yaml) 12](#_Toc164341854)

[SF\_URL 13](#_Toc164341855)

[API\_URL 13](#_Toc164341856)

[ROUTE 14](#_Toc164341857)

[COMPANY\_ID 14](#_Toc164341858)

[CLIENT\_KEY 16](#_Toc164341859)

[DOC\_GEN URL / PE\_HOST 17](#_Toc164341860)

[CERTIFICATION AND PRIVATE KEY 18](#_Toc164341861)

[5 Deploying the application 19](#_Toc164341862)

[Backend Deployment 19](#_Toc164341863)

[FRONTEND Deployment 23](#_Toc164341864)

[6 Address Configuration (Provisioning) 24](#_Toc164341865)

[creating an address 24](#_Toc164341866)

[7 Configurations and tests 25](#_Toc164341867)

# 1 BTP Deployment Guide

## BTP accesses

***Each customer following these instructions will be allowed access to their instances which are required to deploy all our applications.***

***All the following steps must be taken for a correct and functional deployment.***

***If the customer does not have sufficient knowledge of working in the BTP Cockpit, it is possible to request a consultation with a developer with whom this configuration will be carried out.***

### Global Account

Admin rights are granted if the customer is interested in the management of his Global and Subaccount by the developers.

**ADMIN**

* only if the customer needs and wants to create and edit a sub account and the entire space.
* This is how your left bar will look when you have admin rights to global account.

**VIEWER**

* It is enough if the customer can prepare the environment himself, so this permission is enough for us to get to the subaccount.
* This is how your left bar will look like when you have viewer rights to global account.

### **Subaccounts (multi-Environment)**

* Here we will be interested in two main factors to set the right approaches.
* Permissions are granted for the given Subaccount on which the application will be deployed (TEST/PROD)

1. Security
2. Cloud Foundry

**SECURITY**

* When the Users tab opens, there is a Create button in the upper right corner, click on it and fill in the form where you enter the email of the developer you want to add.
* in most cases the default identity provider is assigned, unless otherwise specified.
* the Role Collections must also be assigned.

**After creating the user, click on the row that was created with the new user.**

* Expand the overview of the user on the right side of the screen, click on the three dots in the Role Collections section to assign Role Collections
* **When you click on the Role Collections assignment you will see a form like this where you will be given the choice to check every role that will be available**

**Cloud Foundry**

* In Cloud Foundry, click on the Org Members button.

**Once the Org Members tab opens, the procedure is very similar to Security, there is an Add Members button in the top right corner, click on it and fill in the developer's email and you have the option to fill in the rights for the developer.**

* In the E-mails field fill in the email of the developer you want to add
* In Assign Org Roles we assign in most cases all the roles offered by

Once the developers and all the Roles and Role Collections are assigned, we can click on the space where the applications will be deployed.

* **Click on the Add Members** button.
* **You will see a form that you fill out very similarly to all the previous ones.**
* Enter the developer's email.
* assign roles according to the screenshot, in most cases all.
* These permissions are set for the specific space to which the application will be deployed.

## 1.2 Success Factors

It is necessary to enable accesses to SF for TEST/PROD, where in each instance we create API key, Certification, Private Key, copy Company key via Manage OAuth Data

Via Manage Homepage we add a tile for the application to the main page in SF.

Access to SF either as Admin or for instances only - Manage OAuth Data and Manage Homepage

The creation of objects for the application and import is handled by the Consultants.

**More information with print screen are here** [**Guides**](https://successsolutions2.sharepoint.com/:f:/s/RD830/ElwBUfYtdaFLpflqe-shiCAB_KhdgZ_EQS66TEGYzQ7pkQ?e=0CJztB)**.**

## BTP instance log

If we are deploying an application for the first time, we need to make sure that we have logs set up, which then allows us to monitor the deployment process and look for possible deployment errors.

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**Go to BTP to subacc to service and then to the service and instance tab and at the bottom see if any instances are already set up. If there is a login service created in the **Service Application instance**, we can skip this step, just make a note of the name of this service which we then enter into manifest.yaml under service. If the names did not match the logs would not be read.

**If there is nothing in the instances, we have to create the service by clicking on create and selecting the appropriate Service " Application logging instance" and assign the appropriate name (we can select our own) which we will then insert into manifest.yaml and click on create , nothing else needs to be set.**

# Preparing the App for Deployment

***The Document Generator application has 5 parts:***

* DocumentGenerator-Be
* HromadneGenerovani-Fe
* Dg-configurator

Then we need to deploy PlacehlderEngine which inserts metadata into the placeholder to DG

* Placeholderengine\_v2
* Placeholderengine\_v2\_config

## Backend Document Generator

For the backend part of the application, the. env requisites for deploying it is the only requisite for this part.

A screen shot of a computer

Description automatically generatedThe deployment is projected either via **manifest.yaml** or **bitbucket-pipelines.yml** which deploys the Backend part automatically.

**We set several things in bitbucket-pipelines:**

**A close up of a website

Description automatically generatedCf api - The api URL is found from the customer's BTP in their subaccount.**

**Cf auth** - These are the login credentials of the developer or account that has access to the required BTP

**Cf target** - It is login to the global account which is via **-o** and to the space -s to which we will deploy the application.



org name is the acc you connect to and the space name and select which space you want to deploy to

The other set-env settings will be detected after the application is deployed for the first time.

The **$** dollar sign is used for the bitbucket and variables that store the data promoted in the respective pipelines (it is mainly used for passwords and various sensitive information so that they are not visible directly in the code)

This is where you configure the CF login to Sf, the login to global acc and its subacc, and all the settings needed for deployment.

If we don't have pipelines set up, we can deploy via manifest.yaml in the manifest we just have to see if the application name matches and we also have to look at the name of the service log that we either created or was already created in BTP in the previous case **BTP instance log**

**A screen shot of a computer

Description automatically generated**

## Placeholderengine\_v2 BE

This part of the backend of this application is written in Typescript and must be prepared first before deployment.

First, we run the **NPM INSTALL = NPM I** command via terminal to the root directory of the placeholder to install all dependencies and node-modules for a successful build of the application

A screenshot of a computer

Description automatically generated

After installing the module, we run another command NPM RUN BUILD which will take care of the rest, it will build and prepare our application after deployment, if the build goes well a **dist** folder will be created in the application - which is necessary for the working part of the application.

A screen shot of a computer program

Description automatically generated

Then we just check the manifest.yaml file again so that we have the correct application name and our service log. The application name can be found in package.json.

## HromadneGenerovani FrontEnd

This part of the document generator must be prepared for deployment.

In the terminal, navigate to the static directory with the command cd app and then cd static.

Here we install the node helper modules using NPM INSTALL which will then control the build of the application.

A black screen with white text

Description automatically generated

In the same directory we enter **npm run build** which will create the build directory where the base of our application is.

Then we return to the app root directory using cd.. where we create a dist folder and copy the contents of the build folder we created a little while ago into this folder. This will prepare the Frontend application and the next part of the Document Generator

manifest.yaml which then controls the deployment itself will be set up in the next chapter. The application prepared in this way will suffice for now.

## DG -Config / Placeholderengine-V2-config

The config part of the file is set the same as the frontend part, the preparation and setup are the same.

A screen shot of a computer

Description automatically generatedIn each application, we again go to the static folder using the terminal and the command cd app and cd static and, in this directory, we again first install node-modules using **npm install**, which allows us to build the application.

In the next step, again in the terminal in the same folder, enter the **npm run build** command to create a build application.

A screenshot of a computer

Description automatically generatedFor DG config, which is written in React, the build will again appear in the static directory. Go to the app folder again where we create a **dist** file and copy everything that contains the build here.

A computer screen shot of a program

Description automatically generatedFor Placeholder config it behaves slightly differently, the app is in Typescript so when the build is created the dist folder is created at the same time this folder is only moved from the static folder to the app folder.

# Logging in and setting CF.

A screenshot of a computer

Description automatically generatedTo access CF we first need to find out the API to which the respective subaccount is connected, we can find this out from the BTP. The address is needed to connect to the space.

The Api address is the same in most cases, but it is good to make sure you cannot connect to the subaccount without the correct Api address.

To log in to CF you need an E-mail and password to BTP.

A screen shot of a computer

Description automatically generatedAfter connecting to the cloud foundry, you use the **cf login** command and then enter your email and password or **cf login --sso** where you click on the subcommand, and it generates a login code that you enter the terminal.

If you need to log into another Endpoint Api, just type **cf api** and copy the appropriate Endpoint.

You will only see the accounts that are assigned to a given endpoint, you will never see a list of all of them unless you are on their endpoint Api.

# Preparing for deployment (deploy.yaml)

The yaml file controls the deployment of the application, adding details such as how many instances to create, how much memory the application will need and other necessary variables to make the application work.

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

For multi-customer deployments it is a good idea to create your own yaml file to add variables to, this will allow you to not overwrite the root file each time a new deployment is made.

When these variables are added, the information is automatically overwritten in the main file

**!! The names in the created file must match those in brackets ((API\_URL)) it is case sensitive!!**

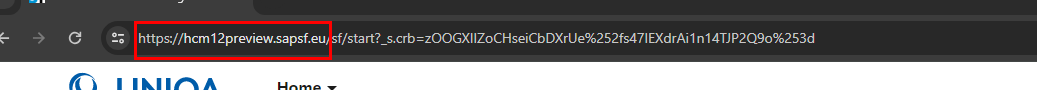
A computer code with red lines

Description automatically generated

In the application in the config.js file you can make sure that you have all the necessary env. that need to be inserted or created if you miss one in the yaml file.

## SF\_URL

What is the SF\_URL I need can be found from the SF address of the customer, after logging into SF we investigate the ulr address and from there we find out what sf url the customer uses.



It is always **https:// <<datacentrum>>.eu** Everything beyond this address is no longer relevant for us.

## API\_URL

A screenshot of a white sheet

Description automatically generatedThis address comes directly from the SF\_URl [on this page](https://help.sap.com/docs/SAP_SUCCESSFACTORS_PLATFORM/364a6d783ef041c786df62eb8c410412/e78cc7b3bc644f15bfc7b051e19c237a.html) to find the corresponding SF\_URL in the third column.

When we find the appropriate address, we focus on the first column DATA CENTER where is the code designation of the address. This is necessary to find the API\_URL [on this page](https://userapps.support.sap.com/sap/support/knowledge/en/2215682), we find the row that corresponds to our code name, and this is our API\_URL.

**PREVIEW** is for the test instance.

**Production** is for production.

A red line with blue text

Description automatically generated

## ROUTE

**This route is created according to the rule:**

**<<application name>> - <<subdomain>>.cfapps <<api endpoint>>**

**the name of the application is found in package.json or directly in manifest.yaml followed by a dash subdomain which is on the BTP of the respective subaccount followed by .cfapps and the noon part of the Api endpoint as seen in the picture**

**A screenshot of a computer

Description automatically generateds**

this way we create the application address which is created both in routes and in APP\_URL

## COMPANY\_ID

Company ID in some cases can be found directly in BTP from the subaccount, but in most cases you can find it either from the client itself or directly in SF when you have access.



After logging into the SF client, enter **Manage Oauth2 Client Application** in the search box.

A screen shot of a computer

Description automatically generated

Here you will see a table of applications logged in, just click on any of the view buttons.

A screenshot of a computer

Description automatically generated

And after clicking you will see a table where the first column shows the CompanyID we need



## CLIENT\_KEY

The client key is retrieved from the same place as the Company\_id again by entering it in the search box in SF Manage OAuth2 Client Application which redirects you to the next page.

Here are 2 points:

1. we can find an already registered application by name (usually when creating it you enter the same name as the application itself, for better searching) and when we click on view we will see in the middle of the page Api Key = CLIENT KEY

2. If the application is not yet registered here, we must first create it, click on REGISTER CLIENT APPLICATION

A screenshot of a computer

Description automatically generatedA screen shot of a computer

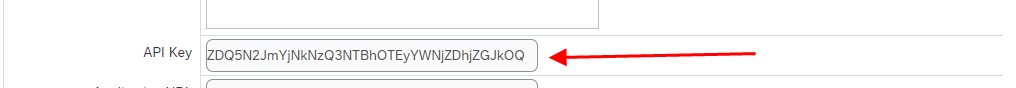
Description automatically generated

**Here we enter the name of the application, we can look in the manifest.yaml and copy the name.**

**Then enter the URL it is our created route that we did in the previous step supplemented with https:// and Certificate, usually we use the same for all applications.**

**We have the certificate stored because the certificate includes the private key which is paired and without these two important things the application will not work.**

**When we fill everything in, we click on Register.**

****

When we open the created and registered application via the view button, we will see a new window with our CLIENT\_KEY = API\_KEY in the middle.

For each part of the application, we have to create a new **CLIENT\_KEY** the only thing that remains is always Certificates, we use the same.

## DOC\_GEN URL / PE\_HOST

This variable specifies the URL of the backend part of the application after its first deployment. The URL is in the BTP in the given space where the application is deployed.

A screen shot of a computer

Description automatically generated

The address is simply copied and added to the manifest.

## CERTIFICATION AND PRIVATE KEY

These keys are necessary for the functioning of the reposted application, they must always be the same and both must be paired together, the certificate does not work without the private key and vice versa.

If we don't have either key the key is registered as an application in Manage OAuth2 Client application in SF where we first create the keys when we create them.

Both keys can be downloaded, after creating the Certificate you must download the private key as well because it is the only way to get it then you will never get it again and you would be forced to make a new certificate.

A screenshot of a computer

Description automatically generated

The keys are added directly to the code, in **APP/SRC/VARSSL**, and this opens certs/cert.pem for the certification and private/private.key for the private key.

We always need to be careful that both keys are copied correctly otherwise the application will give us an error. It is always a good idea to check the first and last few characters of both codes.

# Deploying the application

**The deployment itself is governed by 2 rules:**

1 First we deploy the BE part of the application (because we need their URL addresses to the FE part)

2. Then we deploy the FE

## Backend Deployment

If we have prepared Pipelines, we just need to prepare in variables in bitbucket prepared data that we found here **Preparing for deployment (deploy.yaml) and then add their variables names into pipeline.**

A screenshot of a computer program

Description automatically generated

**we can add a new one:**

**cf sent-env documentgenerator PDF\_TOKEN**

* new function to generate a token to better understand who is generating the document and how much. (only for DG)

**cf sent-env documentgenerator PE\_HOST**

* link to BE url of placeholderEngine(only for DG)

After all the variables are completed, the process of deploying the application to the BTP is automatically started.

The same can be continued for BE placeholder setting pipelines and automatic deployment, but the additional PDF and PE\_host transformations are not used here.

All these variables can then be edited and checked in BTP directly, or edited and restarted without having to re-deploy the application.

If we don't have pipelines ready, we can deploy the BE part of the application without this necessity, just log in to CF via the terminal, check manifest.yaml if the services: are correctly described and then deploy the application via the CF push command.

A screen shot of a computer

Description automatically generated

If everything is fine the application will go through the deployment process, if it is deployed as you see in the yaml file we don't have set env variables which we then have to enter directly in the BTP. So, when we deploy for the first time, it is assumed that at the end of the deployment the terminal will show us that the deployment failed because we have not set the variables.

We go to BTP to the subaccount and space and select the BE we have deployed.

A screenshot of a calendar

Description automatically generated

A screenshot of a computer application

Description automatically generatedAfter clicking, the application will be displayed directly, go to **User\_provide\_variables**

And we will see a blank page where we have no variables, we will enter each variable we need manually. click on create variable.

A screenshot of a chat

Description automatically generated

And gradually we'll create all the transformations we need

A screenshot of a computer

Description automatically generated

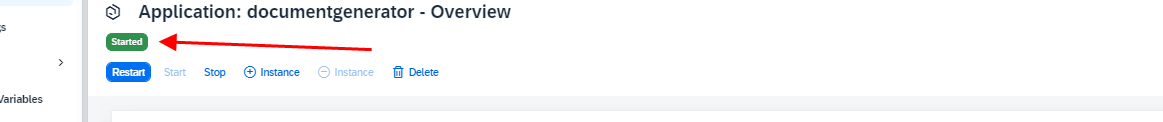
All the variables, where to find how to find or create them are in the chapter **Preparing for deployment (deploy.yaml)**

List of PDF\_TOKEN

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | * KIWITEST | * vkHmyYh9 | | | * 999999997 | | * 08/20/2033 17:20:08 | |
|  | * 2SPROD | * oqMuypTx | | | * 999999949 | | * 08/20/2033 17:19:11 | |
|  | * DEMO2S | * 0pnvAWwE | | | * 999999939 | | * 08/20/2033 17:19:31 | |
|  | * KIWIPROD | * mfb5jFWT | | | * 999999994 | | * 08/20/2033 17:19:54 | |
|  | * ASAHIPROD | * hBeGm5dl | | | * 999997150 | | * 08/20/2033 17:20:25 | |
|  | * ASAHITEST | * hYbcxpKP | | | * 999998912 | | * 08/20/2033 17:20:43 | |
|  | * JUSDAPROD | * xzTAqKP8 | | | * 999999998 | | * 08/20/2033 17:21:28 | |
|  | * JUSDATEST | * F2aW6wV0 | | | * 999999996 | | * 08/20/2033 17:21:47 | |
|  | * UNIQAPROD | | * ug3148g0 | * 999999999 | | * 04/08/2034 22:15:01 | |  |
|  | * UNIQATEST | | * MeSIL2sB | * 999999999 | | * 04/08/2034 22:15:17 | |  |

The same applies to Placeholder, but the variables do not contain PDF\_TOKEN AND PE HOST

After manually adding all variables the application in BTP must be restarted (it doesn't have to be deployed again) and after the restart when everything is set correctly BE should be ready



## FRONTEND Deployment

Deployment parts of Frontend are the last 3 parts of HromadneGenerovani, DG-Config and PlaceholderConfig.

If we have everything prepared according to this chapter **Preparing the App for Deployment** and the manifest.yaml correctly filled according to this chapter **Preparing for deployment (deploy.yaml)**, supplemented by DOCGEN\_URL and PE\_HOST ( which we deployed first )

All we must do is log on to the CF in the given space and deploy all the parts one by one to the BTP.

A screen shot of a computer

Description automatically generated

When we have everything set up and checked, and all deployments have taken place, the green start button is lit for all applications. We can test their functionality. When we start all 3 Frontend applications, and the screen shows this:

A screenshot of a computer error

Description automatically generated

This means that the application is ready, and we are halfway done by this stage. The application is successfully deployed, but until the Provisioning addresses are created, we don't know if it is fully functional.

# Address Configuration (Provisioning)

This step is necessary for the functioning of the application, the created addresses must be sent to the appropriate consultant who oversees the application to the customer.

Addresses are always created 3 and from each frontend. So, for DG we will send 9 https addresses which the consultant will enter SF.

## A screenshot of a computer Description automatically generatedcreating an address

**1- address is for login.**

**https:// <<ROUTE>>/api/login/<<APP\_NAME>>**

**2- address is for logout.**

**https:// <<ROUTE>>/api/logout/<<APP\_NAME>>**

**3 - only adress**

A black screen with red text

Description automatically generated**https:// <<ROUTE>>**

APP name and route can be found either in the manifest of each application or directly in the BTP of each application. So, we must create 3 addresses for Hromadnegenerovani, DG-Config and Placeholderconfig together so 9 addresses.

And send them all to the consultant, watch out for errors, if you create the wrong application, it will not register correctly.

# Configurations and tests

A screenshot of a computer

Description automatically generatedAfter deploying and saving the address to provisioning, the application should start fine like here:

A screenshot of a computer

Description automatically generatedIf the application does not start and either the screen remains white or nothing is displayed, we must look for possible errors. We can look in the BTP logs which may tell us the possible error.

Check all the variables to see if they are entered correctly and there is no error, or if the corresponding Certificates or Private Keys match.

We can try to run the application on the guest locale unless there is a bug in the application itself or a bug in its build, we will test all functionality. If the application is running, there may be an error somewhere in the deployment or in one of the possible versions.

Or all MDF objects or Picklists that are needed for the application to work properly are not specified yet.