Applicable Releases:

**SAP HANA2 SPS02 and above**

This First Guidance Document is part of a series of documents that should help to better understand the various concepts of SAP HANA Data Warehouse SQL approach. The purpose of these documents is to deliver additional information besides SAP Help and Blogs to get a better understanding of the concepts of SAP HANA DW.

For more information or feedback please contact: Sefan.Linders@sap.com

**Version 1.0**

**May 2018**
SAP "How-to" Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

Disclaimer
Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP’s Support Services and may not be modified or altered in any way.
### Document History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>First draft of this guide</td>
</tr>
<tr>
<td>1.0</td>
<td>DW Foundation Backend deploy taken into code</td>
</tr>
<tr>
<td></td>
<td>NPM alternatives added</td>
</tr>
</tbody>
</table>
## Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example Text</strong></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.</td>
</tr>
<tr>
<td>Example text</td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles</td>
</tr>
<tr>
<td>Example text</td>
<td>File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td>Example text</td>
<td>User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td><code>&lt;Example text&gt;</code></td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>

## Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Caution" /></td>
<td>Caution</td>
</tr>
<tr>
<td><img src="image" alt="Note or Important" /></td>
<td>Note or Important</td>
</tr>
<tr>
<td><img src="image" alt="Example" /></td>
<td>Example</td>
</tr>
<tr>
<td><img src="image" alt="Recommendation or Tip" /></td>
<td>Recommendation or Tip</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Introduction ........................................................................................................... 4
2. Continuous Integration toolset ............................................................................. 4
3. Prerequisites .......................................................................................................... 4
4. Assumptions ........................................................................................................... 6
5. Git flow.................................................................................................................... 6
   5.1 Branching model ............................................................................................... 6
   5.2 Main branches .................................................................................................. 7
   5.3 Supporting branches ......................................................................................... 7
   5.4 Naming convention .......................................................................................... 8
   5.5 Automation behavior ....................................................................................... 8
6. HANA XSA configuration ....................................................................................... 9
   6.1 Spaces ............................................................................................................... 9
   6.2 Users ................................................................................................................ 9
7. Bamboo server installation and configuration ..................................................... 9
   7.1 Users ................................................................................................................ 9
   7.2 Support tool installation .................................................................................. 9
      7.2.1 Install Node.JS ........................................................................................... 9
      7.2.2 Install XS command line client ............................................................... 9
      7.2.3 Install MTA Archive Builder .................................................................... 10
8. WebIDE project prerequisites and example ......................................................... 10
   8.1 Adjustment: create module dependency ....................................................... 10
   8.2 Adjustment: add mta.mtaext file ................................................................. 11
9. Configuration Bamboo .......................................................................................... 11
   9.1 Develop branch build plan ............................................................................. 11
      9.1.1 Initialize plan ............................................................................................ 11
      9.1.2 Add task: cleanup previous run ............................................................... 14
      9.1.3 Add task: create MTAR .......................................................................... 15
      9.1.4 Add task: deploy ..................................................................................... 16
      9.1.5 Enable plan ............................................................................................. 18
      9.1.6 Set variables ........................................................................................... 19
      9.1.7 Set artifact definitions ............................................................................ 19
      9.1.8 Demo ........................................................................................................ 20
   9.2 Deploy Develop to TEST_PROD ..................................................................... 23
      9.2.1 Initialize deployment project .................................................................. 23
      9.2.2 Add target environment ......................................................................... 23
      9.2.3 Set variables ............................................................................................ 26
      9.2.4 Add trigger .............................................................................................. 27
      9.2.5 Demo ........................................................................................................ 28
   9.3 Feature branch build plan ............................................................................... 30
      9.3.1 Detect server capabilities ....................................................................... 30
9.3.2 Workaround branch .................................................................................. 30
9.3.3 Clone and configure branch ..................................................................... 30
9.3.4 Demo ........................................................................................................... 36
9.4 Master branch build plan .............................................................................. 37
9.4.1 Demo ........................................................................................................... 40
9.5 Hotfix branch build plan ............................................................................... 40
9.5.1 Demo ........................................................................................................... 42

10. Configuration BitBucket .................................................................................. 44

11. Miscellaneous topics ....................................................................................... 44
11.1 Mail server and notifications ........................................................................ 44
11.2 Side note on CI builds .................................................................................. 44
11.3 Tagging .......................................................................................................... 44
11.4 Versioning ...................................................................................................... 44
11.5 Script re-use and versioning ......................................................................... 45
11.6 Using NPM when no outgoing connection to NPM registry ....................... 45
1. **Introduction**

The HANA SQL Data Warehouse (DW) allows you to organize your Git branches, builds, and deployments, as you see fit. As the HANA SQL DW relies on Git and has command line tools available for building and deploying, you can also use any tool or toolset that support these. However, as there are so many possibilities and therefore choices to make, this guide provides guidance on two topics:

1. How to organize your branches
2. How to automate the build and deploy process using a Continuous Integration (CI) toolset

This document does not treat Continuous Testing.

2. **Continuous Integration toolset**

There are many different software packages for CI. In this guide, we use the tool *Atlassian Bamboo* and Atlassian BitBucket (Git repository management tool) as example. Most information, instructions, and code examples in this guide are applicable to any CI toolset.

The automation and code examples in this guide are largely based on the SAP CI Best Practices guide, but the code examples have been changed and extended to better suit the DW use case. Also, where the CI guide propose a setup for Jenkins, we do that for Bamboo.

This document does not treat integration with CTS+.

3. **Prerequisites**

The instructions and examples in this document are based on the following software products and versions. It might very well work on other versions but that has not been tested.

Assumed to be installed:
- SAP HANA SPS02 with XS Advanced, Web IDE and Data Warehouse Foundation
- Atlassian BitBucket Server v5.6.2
- Atlassian Bamboo v6.3.0 (assuming application link with BitBucket is set up)

Installation of following tools are part of this document:
- SAP MTA Archive Builder 1.0, SP110 Patch 0
- NodeJS v8.9.4
- XSA Command Line client

Assumed to be configured:
- An open connection to NPM registries. If such a connection is not possible, resort to paragraph 11.6. The registries that need to be accessed (by the MTA Archive Builder) are the following:
  - [https://registry.npmjs.org/](https://registry.npmjs.org/)
  - [https://npm.sap.com/](https://npm.sap.com/)

The setup of the guide was done while having full admin rights on the HANA platform and OS, as well as in the BitBucket and Bamboo applications and OS. Both OS tested are SLES 12.1.

---

4. SAP enabled its flagship change and transport system, CTS+, to work with XS Advanced. It is up to you to decide whether to use this tool for deployments, use another tool instead like described in this document, or use them both.
The branches mentioned in this guide should are assumed to exist already in BitBucket, you can also create them as you go through the guide.
4. Assumptions

This guide works under the following assumptions. You can deviate from these but you will have to change the proposed configuration described in this guide accordingly.

- You run two hana systems: development and production.
- BitBucket and Bamboo are already linked as application
- You are running a WebIDE project created as a Data Warehouse Project based on the SAP Data Warehouse Foundation template.

5. GIT flow

We propose a branching model in this chapter that we think suits Data Warehouse development, although you will find this model in generic application development as well.

5.1 Branching model

The organization of branches is according to the following figure

![Proposed Branching Model Diagram](image)

Figure 1: proposed branching model

This model is based on the following blog post, with the difference that we are not using tags, several actions are automated, and the timing can be different. We advise reading this post if you want to inform yourself on the how and why this design: [http://nvie.com/posts/a-successful-git-branching-model/](http://nvie.com/posts/a-successful-git-branching-model/)
5.2 Main branches

The repository has two main branches, which have an infinite lifetime: develop, and master. The master branch represents what is or should soon be running productively. The development branch contains the latest changes for the next release. New features that should go into the next release, are merged to this branch whenever they are ready.

5.3 Supporting branches

Next to development and master, there are supporting branches:

- Feature branches branch off from development, and are used to develop new features. For any feature, or related set of features, you create a new branch. When a feature is finished, the feature branch can be merged into the development branch. Then, the feature branch no longer serves a purpose and can be deleted.
- Hotfix branches branch off from master, and are used to fix production issues. When the hotfix is merged into master, the branch can be deleted.
- Release branches branch off from development. You would create this branch when you want to start developing a new release in development, while the previous release is not yet ready to merge to master. The release branch would then allow you to finish up the first upcoming release. In the model above this branch was not displayed, therefore below an illustration including such release branch.

![Branch model including a release branch](image)

Figure 2: Branch model including a release branch
5.4 Naming convention

This guide assumed your branches have the following name or naming convention.

- Master
- Develop
- Feature/<feature_number_name>
- Hotfix/<hotfix_number_name>
- Release/<release number and or name>

5.5 Automation behavior

In this guide, Bamboo will be set up to automate build and deployment based on changes in any branch. Below this behavior is illustrated. The entire control on moving functionality or fixes to development or production, is entirely managed by the Atlassian toolset. For example, when you want to move functionality from your development system to the production system, you would go to BitBucket and merge the Develop branch to the Master branch. This will automatically trigger a build and deployment of the new functions to production.
6. HANA XSA configuration

6.1 Spaces
XS Advanced Spaces
- On the development system, create the following spaces:
  - DEV_CENTRAL
  - CI
- On the production system, create the following spaces:
  - PROD
  - TEST_PROD
Make sure all spaces have grantor services running to provide authorizations.

6.2 Users
HANA user
- Create a user BAMBOO
- Provide “developer” authorizations for each of the earlier mentioned spaces

7. Bamboo server installation and configuration
This guide assumed you have already set up the Atlassian toolset. This chapter describes the additionally required steps.

7.1 Users
For the Bamboo server, it is assumed you have a user available that can execute OS statements, and which can access the below mentioned NodeJS and MTA Archive builder binaries. For creating this document, we have used user BAMADM which is the linux user which runs the Atlassian Bamboo application.

7.2 Support tool installation
7.2.1 Install Node.JS
Download node-v8.9.4-linux-x64.tar.xz from https://nodejs.org/en/download/ to /tmp

```
md /opt/nodejs
cd /opt/nodejs
tar -xf /tmp/node-v8.9.4-linux-x64.tar.xz
```
Create symbolic link to make future upgrades easier
```
ln -s /opt/nodejs/node-v8.9.4-linux-x64 /opt/nodejs/latest
```
add path to nodejs to /usr/local/bamadm_home/.bashrc
```
PATH=$PATH:/opt/nodejs/latest/bin/
```

7.2.2 Install XS command line client
Download latest version of XS RUNTIME 1, to be found in SAP SMP under SAP HANA PLATFORM EDITION / SAP HANA PLATFORM EDITION 2.0.
```
XS_CLIENT00P_71-70001320.ZIP
```
add path to xsa client to /usr/local/bamadm_home/.bashrc
PATH=$PATH:/opt/nodejs/latest/bin/:/opt/xsa_client/bin/

7.2.3 Install MTA Archive Builder

Download latest version of MTA ARCHIVE BUILDER 1.0 from SAP SMP
Download MTABUILDER110_0-80002501.JAR to /tmp
There is no installation process, just copy into directory for later use

```bash
cd /opt
md mta_archive_builder
mv /tmp/MTABUILDER110_0-80002501.JAR /opt/mta_archive_builder
chmod +x MTABUILDER110_0-80002501.JAR
```

Create symbolic link to make future upgrades easier

```
ln /opt/mta_archive_builder/MTABUILDER110_0-80002501.JAR /opt/mta_archive_builder/latest.jar
```

8. WebIDE project prerequisites and example

This guide assumes you are using a project created as a Data Warehouse Project based on the SAP Data Warehouse Foundation template. This can be created from the Web IDE. In this guide, the example project is called “VerticalLocal”, and contains a few basic DB modules.

8.1 Adjustment: create module dependency

Of the TaskChains and the Persistence (or Database) module, it is usually better if the persistence module is build first. It is more likely that there is an error in the Persistence module, than in the TaskChains module. When the first module (Persistence) fails, the build of the second module (TaskChains) is not started. As roll-back for failures is done on the module level, you prevent the TaskChain module deployment to run out of sync with the Persistence module if the latter has an error but builds after the TaskChain module.

This is the code to add:

```bash
# Added to make sure persistence builds first and fails first
- name: verticalLocal-Persistence
  type: dwf
  parameters:
```

```yaml
- name: verticallocal-TaskChains
type: dwf
parameters:
  app-name: verticallocal-TaskChains
  path: TaskChains
  requires:
# Added to make sure persistence builds first and fails first
  - name: verticallocal-dws
# Added to make sure persistence builds first and fails first
  - name: verticallocal-Persistence
```

```bash
- name: verticalLocal-Woo
  type: dws
```
8.2 Adjustment: add mta.mtaext file

As the different deployments should go to different schema’s, you need to be able to set the schema at deploy time. The mta.mtaext file enables this. The code that we suggest for your builds assume that this file sits in the root of your project. At deploy time, the code renames the schema into what you have chosen as variable for that build or deployment plan.

```yaml
mta.mtaext

| 1 | _schema-version: "3.1.0"
| 2 | ID: verticallocal-mtaext
| 3 | extends: verticallocal
| 4 | resources:
| 5 |   - name: verticallocal-container
| 6 |     parameters:
| 7 |       config:
| 8 |         schema: schema_to_be_replaced_at_deploy

_schema-version: "3.1.0"
ID: verticallocal-mtaext
extends: verticallocal
resources:
  - name: verticallocal-container
    parameters:
      config:
        schema: schema_to_be_replaced_at_deploy
```

9. Configuration Bamboo

9.1 Develop branch build plan

What you will configure is the following:

1. A Develop build plan which creates an executable (the *.mtar file)
2. A deployment to the central development branch. This will not be configured as a deployment project, but included in the build plan. Then, a build will only be marked successful if the deployment is also successful. Most of the expected errors are in the deployment, when the database artefacts are activated in HANA. Also, any testing as part of the build plan can only be done after the database artefacts are activated. Therefore, all builds should include at least one deployment, so that the status reporting in Bamboo, BitBucket and JIRA, are as expected.
3. The build and deploy is triggered by any change in the Develop branch. According to our model, this would be a merge into Develop from either a feature, hotfix, or release branch.

9.1.1 Initialize plan

Follow the screenshots and code snippets to create the build plan for the develop branch.
HANA SQL DW – Continuous Integration with Atlassian Bamboo

![Bamboo interface showing project and build plan configuration]

- **Create project**
  - **Project name**: Vertical.Local
  - **Project key**: VER
  - **Project description**: HANA SQL DW project

- **Configure plan**
  - **Project and plan name**
    - **Project**: Vertical.Local
    - **Plan name**: Development build and deploy
    - **Plan key**: DBD
    - **Plan description**: Build of Develop branch and deploy into central dev schema
  - **Plan access**: Allow all users to view this plan

- **Link repository to new build plan**
  - **Repository host**: Bitbucket Server / Stash
  - **Display name**: Develop
  - **Bitbucket Server / Stash details**
    - **Server**: Bitbucket
    - **Repository**: HANA_SQL DW / Vertical.Local
    - **Branch**: develop
  - **Repository access**: Allow all users to reuse the configuration of this repository
Create plan

Configure tasks

Each plan has a default job when it is created. In this section, you can configure the Tasks for this plan's default job. You can add more jobs to this plan once the plan has been created.

A task is an operation that is run on a Bamboo working directory using an executable. An example of task would be the execution of a script, a shell command, an Ant Task or a Maven goal. Learn more about tasks.

- Source Code Checkout
- Checkout Default Repository

Final tasks: Are always executed even if a previous task fails

Drag tasks here to make them final

Add task

No task selected

Select a task from the list on the left to configure it.

Enable this plan?

- Yes please!

By selecting this option your plan will be available for building and change detection straight away.

do not select this option if you have advanced configuration changes to make after creation.

Create  Cancel
9.1.2 Add task: cleanup previous run

Add task
Search for “shell”
Choose Script

# when a build and deploy finishes, we don’t clean up the workspace.
# this allows issue analysis in case anything goes wrong in the execution plan.
# this means however that in a new run, we do have to delete the old mtar file

# remove created mtar from previous run
if [ `ls -l *.mtar 2>/dev/null | wc -l ` -gt 0 ];
then
    rm *.mtar
    echo "existing mtar(s) deleted"
else
    echo "no existing mtar to delete"
fi
9.1.3 Add task: create MTAR

```bash
# reinitialize bash in case path variable reinitialization is needed
source ~/.bashrc

# create local npmrc file
cat <<EOF > .npmrc
registry=https://registry.npmjs.org/
@sap:registry=https://npm.sap.com/
EOF
echo "end of step: create local npmrc file"

# link to local .npmrc file from submodules
# first extract the different module paths from yaml file
modulePaths=`awk -F: '$1 ~ /path/ { gsub(/s/,"", $2) print $2 }' mta.yaml`
# create links in each module to .npmrc file
for path in $modulePaths; do
    ln -sft $path ../.npmrc
done
echo "end of step: link to local .npmrc file from submodules"

# extract artifact name from yaml file
mtaName=`awk -F: '$1 ~ /^ID/ { gsub(/s/,"", $2) print $2 }' mta.yaml`
echo "end of step: extract artifact name: ($mtaName)"

# safeguard yaml file before adapting it
cp mta.yaml mta.yaml.original

# also set the version number to 0.0.<bamboo build nr> if version is 0.0.1
# the version of the mta will then be made equal to the bamboo build number
# making it easy to analyze what is deployed where
```
HANA SQL DW – Continuous Integration with Atlassian Bamboo

```
sed -i '/version: 0.0.1/c\version: 0.0.'$bamboo_buildNumber'' mta.yaml
# if version != 0.0.1, we want to know the version anyway for file naming
# fetch yaml version from file
yaml_version="$(sed -n -e 's/^version: //p' mta.yaml)"
#echo "yaml version: $yaml_version"
echo "end of step: extract version: ($yaml_version)"

# execute MTA build
java -jar $bamboo_MTA_ARCHIVE_BUILDER_LOCATION --mtar ${mtaName}_${yaml_version}.mtar --build-target=XSA build

# put original mta.yaml back so the repo is restored for following build
cp mta.yaml mta.yaml.edited
cp mta.yaml.original mta.yaml
rm mta.yaml.original

9.1.4 Add task: deploy
```

# reinitialize bash in case path variable reinitialization is needed
source ~/.bashrc

# make sure to exit script with error if any deploy error occurred
# bamboo only looks at final $? status so we use a variable and
# set it after deploy command
exitStatus=0

# put mtar filename in variable (* serves to exclude "./"
mtaFileName="$(find * -type f -iname *.mtar)"

# safeguard yaml extension file before adapting it
cp mta.mtaext mta_edited.mtaext

# set schema name in extension file
sed -i '/schema_to_be_replaced_at_deploy/c\'  
  "schema: "$bamboo_TARGET_SCHEMA"
"mta_edited.mtaext

# set target system
# skip SSL using "--skip-ssl-validation" as we use self-signed certs
xs api $bamboo_XSA_API_ENDPOINT --skip-ssl-validation
echo "end of step: set endpoint"
xs login -u $bamboo_HANA_XSA_USER -p $bamboo_HANA_XSA_PASSWORD -o $bamboo_ORGANIZATION -s $bamboo_CI_SPACE
echo "end of step: login to xs server"

# set deploy starting timestamp in variable for later retrieval of logs
deploy_start_ts=$(date +%Y-%m-%d"%H:%M:%S)
echo "deploy start timestamp: 

# deploy mtar
xs deploy -f ${mtaFileName} -e mta_edited.mtaext
# exit status should depend on successful deploy
# should be placed directly after deploy command
if [ $? != 0 ]; then
  exitStatus=1
  echo "Deploy exit status: $?"
fi

# set exit code
if [ $exitStatus != 0 ]; then
  echo "Deploy failed, hence exit status !=0"
  exit 1
else

May 2018
```bash
   echo "Finish with exit status 0"
   exit 0
```

### 9.1.5 Enable plan

![Create plan interface](image)

**Enable this plan?**

- [x] Yes please!
- [ ] No thanks!

*By selecting this option your plan will be available for building and change detection straight away. do not select this option if you have advanced configuration changes to make after creation.*

[Create] [Cancel]
9.1.6 Set variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI_SPACE</td>
<td>DEV CENTRAL</td>
</tr>
<tr>
<td>HANA_XSA_PASSWORD</td>
<td>—</td>
</tr>
<tr>
<td>HANA_XSA_USER</td>
<td>bamboo</td>
</tr>
<tr>
<td>MTA_ARCHIVE_BUILDER_LOCATION</td>
<td>/opt/mta_archive_builder/latest.jar</td>
</tr>
<tr>
<td>ORGANIZATION</td>
<td>ZEPH</td>
</tr>
<tr>
<td>TARGET_SCHEMA</td>
<td>V1.2</td>
</tr>
<tr>
<td>XSA_API_ENDPOINT</td>
<td><a href="https://example.com/mta">https://example.com/mta</a></td>
</tr>
</tbody>
</table>

9.1.7 Set artifact definitions

Set the artefact definitions, so that these files can be re-used by further deployment in next phases. For example, with the mta_mtar artifact definition, you are storing the build mtar file for later re-use. In our case, that is the deployment to TEST_PROD as described in the next paragraph.
9.1.8 Demo

We make a change in the WebIDE for project VerticalLocal, for table T1, and push to the develop branch.

In Bamboo, a build starts automatically

In the details we can see the log reported live:
Build #7 is now completed

And in BitBucket we can see the build result for this commit as well
### Commits

<table>
<thead>
<tr>
<th>Author</th>
<th>Commit</th>
<th>Message</th>
<th>Commit date</th>
<th>Builds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stefan Linders</td>
<td>1a8d2e6c477</td>
<td>Added T1</td>
<td>4 mins ago</td>
<td>✔️</td>
</tr>
<tr>
<td>Stefan Linders</td>
<td>8a1f412a472</td>
<td>ni/nt�зовfle.edited</td>
<td>Yesterday</td>
<td>✔️</td>
</tr>
<tr>
<td>Stefan Linders</td>
<td>240e94baf69</td>
<td>test1</td>
<td>Yesterday</td>
<td>✔️</td>
</tr>
<tr>
<td>Stefan Linders</td>
<td>z366772b88c</td>
<td>one file generated</td>
<td>2 days ago</td>
<td>✔️</td>
</tr>
<tr>
<td>Stefan Linders</td>
<td>a955d544b7d</td>
<td>copy from other project</td>
<td>2 days ago</td>
<td>✔️</td>
</tr>
<tr>
<td>Stefan Linders</td>
<td>2e56ae83c3f</td>
<td>init</td>
<td>3 days ago</td>
<td>✔️</td>
</tr>
</tbody>
</table>
9.2 Deploy Develop to TEST_PROD

9.2.1 Initialize deployment project

9.2.2 Add target environment
Test on production to early identify environment-related build errors
The code here is the same as in the earlier defined deployment step at § 9.1.4 Add task: deploy.
9.2.3 Set variables

![HANA SQL DW - Continuous Integration with Atlassian Bamboo](image)

**Environment: Test Production**
- Test on production early: identify environment-related build errors.
- **Tasks** define the steps involved in deploying a release to the environment.
- **Other environment settings**
  - These settings are not strictly necessary for your deployment to run, but they can be very helpful and allow you to make Bamboo deployments go just right.
  - **Triggers**, **Agents assignment**, **Notifications**, **Variables**, **Environment permissions**

---

**Edit variables: Test Production**

- Define variables to be used by your deployment tasks. You can use any of the keys below in your task configuration or scripts and it will be automatically substituted with the value.
- You can override global variables by using the same key.
- For task configuration keys, use the syntax `${variableName}`. Bamboo also supports nested variables. For instance, if you set `variableName` = `world`, and `variables.value` = `Hello [bamboo variable name]`, Bamboo will resolve it as `Hello world`. For inline scripts, variables are exposed as shell environment variables which can be accessed using the syntax `BAMBOO_VAR_VARIABLE_NAME` (Linux/OS X) or `%BAMBOO_VAR_VARIABLE_NAME%` (Windows).

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI_SPACE</td>
<td>TEST_PROD</td>
</tr>
<tr>
<td>HANA_XSA_PASSWORD</td>
<td>***************</td>
</tr>
<tr>
<td>HANA_XSA_USER</td>
<td>bamboo</td>
</tr>
<tr>
<td>ORGANIZATION</td>
<td>ZCPH</td>
</tr>
<tr>
<td>TARGET_SCHEMA</td>
<td>TESTPROD_VL</td>
</tr>
<tr>
<td>XSA_API_ENDPOINT</td>
<td><a href="http://047f05a407.mo.sap.corp:30130">http://047f05a407.mo.sap.corp:30130</a></td>
</tr>
</tbody>
</table>

---

Back to deployment project
9.2.4 Add trigger

No automatic triggers defined

- After successful build plan
  - Deployment is started after a plan is successfully built

- After successful stage
  - Deployment is started after a stage is successfully built

- Scheduled
  - Run according to schedule
9.2.5 Demo

When we changed a file in WebIDE, we already saw in § 9.1.8 that a build and deploy to the Develop branch was kicked off. When that build succeeded, the deploy to TEST_PROD, what we configured above, was triggered as well.
HANA SQL DW – Continuous Integration with Atlassian Bamboo

Check later:
9.3 Feature branch build plan

9.3.1 Detect server capabilities

In case GIT is not part yet of the server capabilities, please check this in the settings. Choosing “detect server capabilities” might detect GIT executable automatically. This allows then using “auto merge” functions further up in the configuration of this plan.

9.3.2 Workaround branch

First we will need to create a branch we will not use actively. We call this one “z_bamboo_workaround”, and branch it of Develop, although it does not matter where you branch it off. Later will be explained why you need this branch.

9.3.3 Clone and configure branch

As the plan is pretty similar to the Develop branch plan, we clone the plan and adjust it.
Remove the existing Development branch and instead use the workaround branch
This is where we picked the workaround branch. Reason is that we don’t want a change in the development (or another existing) branch to trigger this plan. We create this plan to automatically track feature branches. This is configured a bit further ahead in this paragraph.
Make sure NOT to enable the branch
After that, go back to “Branches”. We only created this plan branch, to be able to configure it in the “branch” updates section of the branches. With this setting, before a feature is deployed, first the branch gets the updates from the Development branch. Then when the build and deploy is successful, the merged result is merged to the feature branch. This saves the developer work, and it also makes sure that features are tested with the latest commits from Develop, before they are merged to Develop.
Also, you have to add an undeploy task to remove previous CI deploys

```bash
# reinitialize bash in case path variable reinitialization is needed
source ~/.bashrc

# extract artifact name from yaml file
mtaName=`awk -F: ' $0 ~ /^ID/ { print $1 }' mta.yaml`

# set global variable for use in subsequent tasks
echo "end of step: extract artifact name ($mtaName)"

# undeploy
# skip SSL as we use self-signed certs
xs api $bamboo_XSA_API_ENDPOINT --skip-ssl-validation
echo "step set endpoint finished"
xs login -u $bamboo_HANA_XSA_USER -p $bamboo_HANA_XSA_PASSWORD -o $bamboo_ORGANIZATION -s $bamboo_CI_SPACE
echo "login finished"
xs undeploy -f $mtaName --delete-services
# status
```
if [ $? != 0 ]; then
    echo "Undeploy did not succeed"
else
    echo "Undeploy succeeded"
fi

9.3.4 Demo
9.4 Master branch build plan
Delete the Develop branch and add Master
Because our prod space is at the same server as the dev space, as this is a test setup, our schema needs a different name for master than for dev. If you are a different system or tenants for DEV_CENTRAL and PROD, you can keep the schema names the same.

Save trigger
9.4.1 Demo

Prod deploy starts running now

9.5 Hotfix branch build plan

Create

Create plan

Clone plan

Create project

Create deployment project
9.5.1 Demo
HANA SQL DW – Continuous Integration with Atlassian Bamboo

Create a New Remote Branch

Source Branch: develop
Branch Name: hotfix\fix123_contract_name

Bamboo
My Bamboo
Projects
Build dashboard / Vertical/Local / Hotfix CI deploy and test
configuration - Hotfix CI deploy and test

Plan Configuration
- Stages & jobs: 1
  Default Stage
- Default Job: 1
- Branches: 2
  - hotfix\fix123_contract_name
  - master
Create plan branch

Plan branches
- hotfix\fix123_contract_name: Never built
- master: Never built
10. **Configuration BitBucket**

Only allow successful feature and hotfix commits into Develop / Master. This means that the feature or hotfix branch should first have been build and deployed successfully into the CI_BUILD space, before a merge into Master, and with that, deploy into production system is allowed.

11. **Miscellaneous topics**

11.1 **Mail server and notifications**

A mail server is not configured in this guide, but it would make sense to configure this, and set up notifications to let developers and administrators know when a build failed or succeeded.

11.2 **Side note on CI builds**

If a source system, or certain tables in source system, is broken, then the CI builds will fail as they always do a complete build from scratch and therefore check all source system connections. This is a benefit too: you will know very fast if certain sources have changed in an incompatible way, or are not available.

11.3 **Tagging**

Tagging of releases into GIT is not implemented in this guide. There are already enough options to see what is deployed where, and it is too complex for a first simple setup (you will need to setup a private key on remote agent(s)): [https://almfirst.wordpress.com/tagging-from-bamboo/](https://almfirst.wordpress.com/tagging-from-bamboo/)

11.4 **Versioning**

The version of an MTA is defined in the yaml file, and consists of three numbers separated by a dot, for example: 0.3.12. To see which version of an MTA is deployed in a space, type XS MTAS in the command line.

The implementation in this guide does not cover fully automated versioning. This would mean that with each build, the version number is automatically adapted. However, this would require Bamboo to adapt the yaml file and push it back to git, which is not covered due to added complexity and risk.

Implemented in this guide is that when a build takes place, the Bamboo build number is set as the last number. This allows you to match the current version in a system with the build number for the related
branch. The version is only replaced if the yaml version as stored in Git is equal to “0.0.1”, so you can also use your own versioning by manually adapting the yaml file.

### 11.5 Script re-use and versioning

This guide proposes to configure at least four branch types. For each build plan, and the test_prod deployment, very similar code is being used. To re-use this code, you have several options, two examples:

1. You can use bash scripts and start the same scripts from different branch build and deploy plans;
2. You could store the code in a separate Git repository and fetch this code when you start a build.

### 11.6 Using NPM when no outgoing connection to NPM registry

The MTA Archive builder needs outgoing connections to NPM registries. If the automation server where the MTA Archive Builder is installed on does not allow such connections, the first alternative should be to follow paragraph “Setting Up and Preparing the npm Registry” of the CI Best Practise guide at [https://www.sap.com/developer/tutorials/ci-best-practices-xsa.html](https://www.sap.com/developer/tutorials/ci-best-practices-xsa.html) to set up a local NPM registry.

Another work around is to make sure you have all dependent NPM packages stored somewhere already, which you can copy to the right module before you start the build. This will require you to maintain it whenever updates to NPM packages are necessary, and is therefore not the recommended solution. The process would be as follows:

1. Use a server that does have access to the NPM registry (could be your local machine).
2. On that server, run the MTA Archive Builder to build a project that contains all necessary modules that any of your builds are going to need. In this documents example, that is a project that has an HDB module (named Persistence), and two DWF modules named TaskChains and Backend.
3. Inside those project modules, the package.json files contains the pointers to which node modules are needed. The MTA Archive builder downloads these from the NPM registries and places them inside a folder called "node_modules" in each module. In this step, you zip those files into a zip file for each module, and place them at a location from where you can reuse them.

```
zip -r node_modules-Persistence.zip node_modules
cp node_modules-Persistence.zip /tmp/mta_node_modules/
zip -r node_modules-Backend.zip node_modules
 cp node_modules-Backend.zip /tmp/mta_node_modules/
zip -r node_modules-TaskChains.zip node_modules
 cp node_modules-TaskChains.zip /tmp/mta_node_modules/
```

4. In the "create mtar" script after initializing bash, you add the following code

```
# copy needed node modules if there is no outgoing connection to NPM registries
unzip /tmp/mta_node_modules/node_modules-Persistence.zip -d Persistence
unzip /tmp/mta_node_modules/node_modules-Backend.zip -d Backend
unzip /tmp/mta_node_modules/node_modules-TaskChains.zip -d TaskChains
```