HA450

SAP HANA 2.0 SPS05 - Application Development for SAP HANA

PARTICIPANT HANDBOOK
INSTRUCTOR-LED TRAINING

Course Version: 17
Course Duration: 4 Day(s)
Material Number: 50155527

SAP Copyrights, Trademarks and Disclaimers

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation	
Demonstration	
Procedure	
Warning or Caution	
Hint	
Related or Additional Information	
Facilitated Discussion	
User interface control	*Example text*
Window title	*Example text*

Contents

Course Overview

TARGET AUDIENCE

This course is intended for the following audiences:

- Developer

UNIT 1 Introducing Application Development in SAP HANA

UNIT OBJECTIVES

- Describe prerequisite skills for SAP HANA application development
- Describe the basic concepts of application architecture in SAP HANA
- Describe the tools used by the application developer in SAP HANA
- Introduce SAP HANA express edition
- Describe the information sources for SAP HANA developers

Unit 1
Lesson 1

Describing Prerequisite Skills for SAP HANA Application Development

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe prerequisite skills for SAP HANA application development

Describing Prerequisite Skills for SAP HANA Application Development

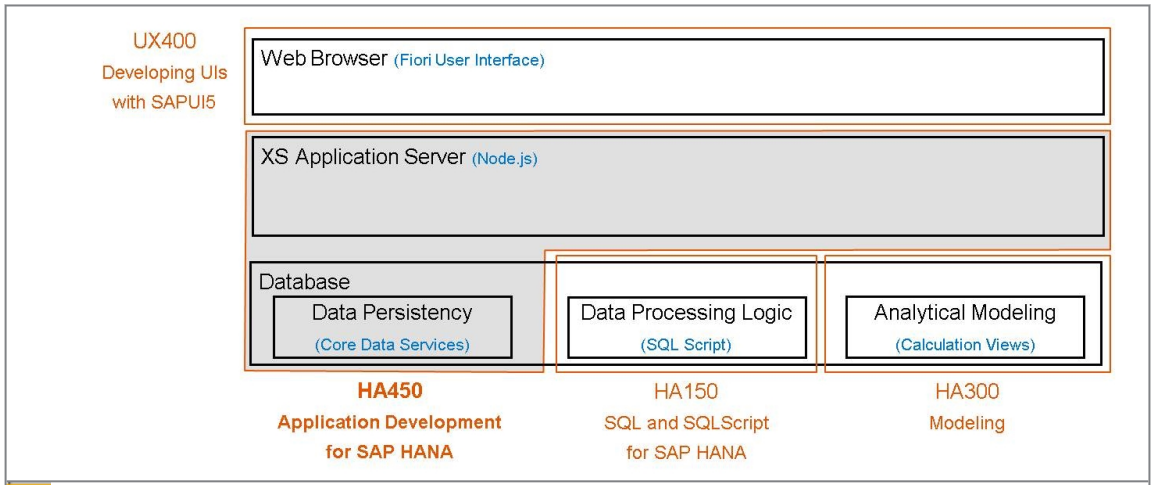


Figure 1: Courses You Need to Develop a Full-Stack Application in SAP HANA

A typical web application developed in SAP HANA is made of 3 tiers:

- A database layer, in the SAP HANA database
- An application server layer, coded using Node.js (other languages are also possible)
- A user interface, built using Fiori technologies, typically using the SAPUI5 JavaScript library

Course HA450 is the starting point to application development, covering the creation of the core application, the database persistency and the integration of the various application layers.

Other courses will then allow the developer to complete his/her skill-set an be able to develop the "full-stack" application.

In particular:

- Course HA150 deep dives the development of SQLScript stored procedures in the SAP HANA database.

- Course HA300 deep dives the development of the Analytical views you may need to integrate within your applications.
- The "SAPUI5 and SAP Fiori" learning journey covers the development of Fiori user interfaces. Out of this, course UX400 is probably the one you should choose to approach UI development with SAPUI5.

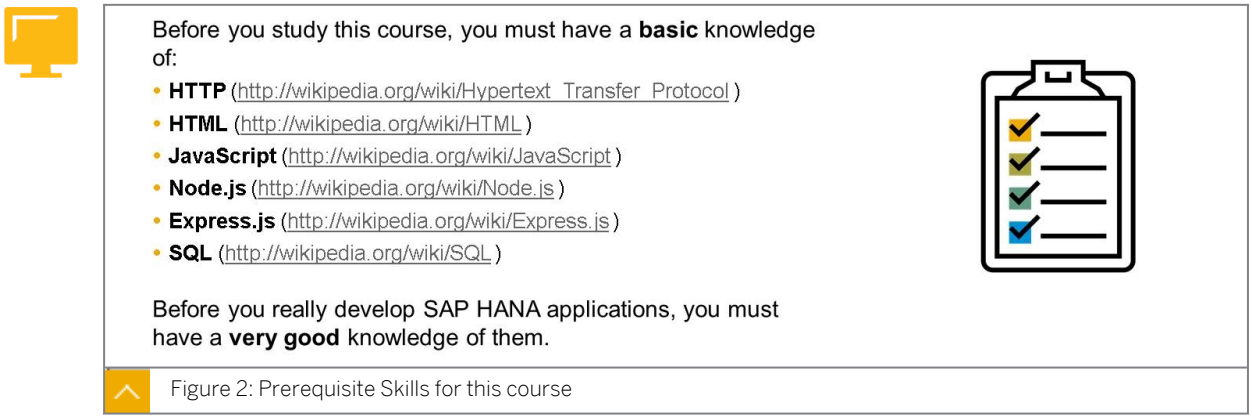


Figure 2: Prerequisite Skills for this course

SAP software is based on several open standards, in particular with respect to development languages and communication protocols.

The knowledge of these is a prerequisite for this course and it is a must-have to develop applications in SAP HANA.

In particular you must have a good knowledge of:

- HTTP (http://wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- HTML (http://wikipedia.org/wiki/HTML)
- JavaScript (http://wikipedia.org/wiki/JavaScript)
- Node.js (http://wikipedia.org/wiki/Node.js)
- Express.js (http://wikipedia.org/wiki/Express.js)
- SQL (http://wikipedia.org/wiki/SQL)

LESSON SUMMARY

You should now be able to:

- Describe prerequisite skills for SAP HANA application development

Unit 1
Lesson 2

Introducing the Application Architecture in SAP HANA

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Describe the basic concepts of application architecture in SAP HANA

Introducing XS Advanced and Cloud Foundry

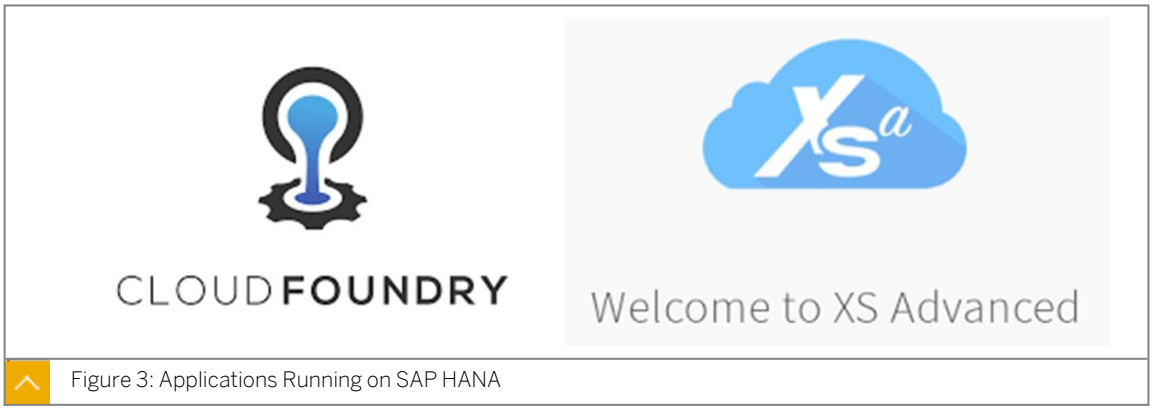


Figure 3: Applications Running on SAP HANA

Alongside the database, SAP HANA provides an application server named SAP HANA Extended Application Services, Advanced Model (commonly known as XS Advanced).

XS Advanced is a renamed and enhanced version of Cloud Foundry. Cloud Foundry is an open-source, multi-cloud application platform governed by the Cloud Foundry Foundation.

Cloud Foundry is currently the application service of choice for the SAP Cloud Platform, while XS Advanced is provided with the on premises version of SAP HANA.

In most cases, you will find that an application running on XS Advanced will also run in Cloud Foundry on the SAP Cloud Platform.

Describing Application Layers and Programming Languages in the SAP HANA Application

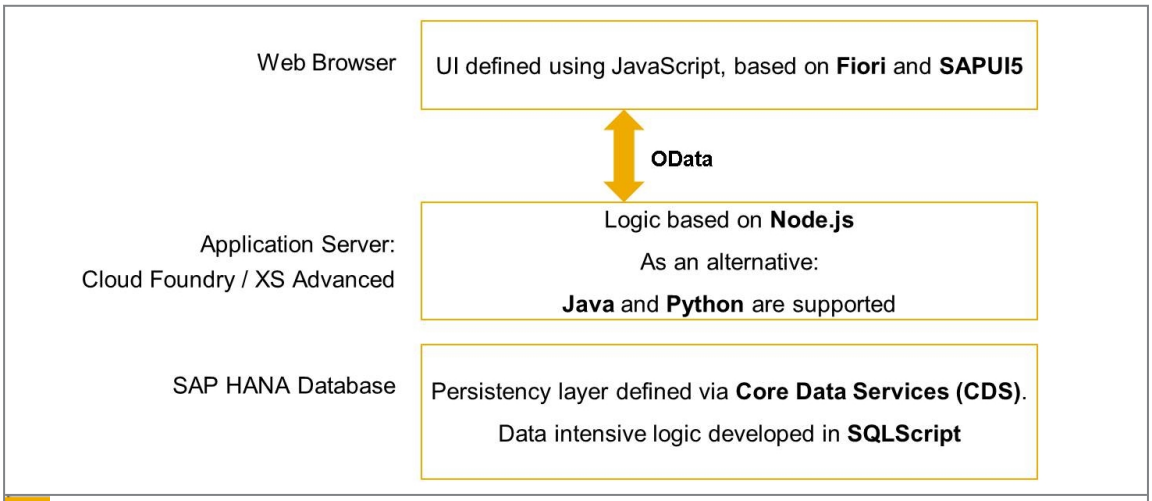


Figure 4: The Full-Stack Application

The full-stack application is made of the following three layers:

- The UI layer: This layer uses the SAPUI5 JavaScript library and runs in the web browser.
- The application server layer: The application logic that runs in this layer can be written in different languages, such as, Node.js, Java, or Python.
- The database layer: The persistency layer is defined via Core Data Services (CDS). The data intensive logic is developed in SQLScript and runs within the SAP HANA Database.

Describing the Runtime Application Architecture in SAP HANA

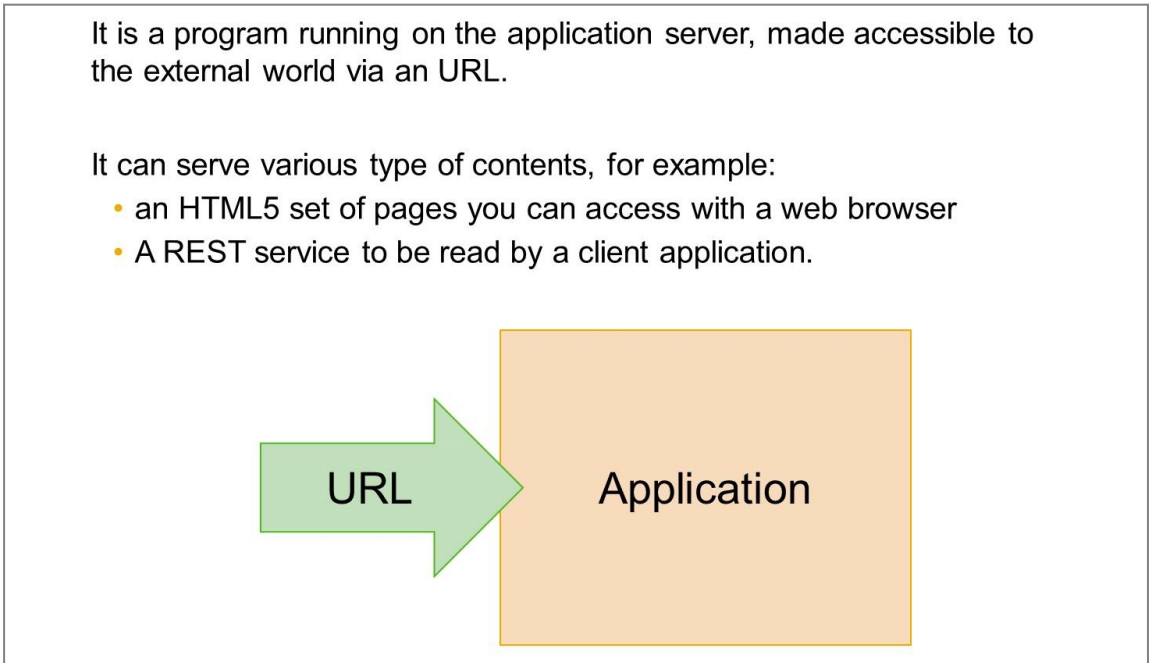


Figure 5: The Cloud Foundry Application

The Cloud Foundry Application is a program running on the application server. The external world can access Cloud Foundry via a URL.

The application can provide via the url various types of content, such as the following:

- A set of HTML5 pages that you can access from a web browser
- A REST service to be read by a client application

Figure 6: The Buildpack

Applications are deployed to the target platform by using the push operation of the platform API. For this reason, in Cloud Foundry parlance, applications are "pushed" to the platform.

Pushing an application works as follows:

1. The application files are uploaded to the platform.
2. Programs called buildpacks are executed to create archives that create the self-contained and ready-to-run executable applications.

Some of the most typical activities executed by buildpacks include downloading any required libraries and other dependencies, and configuring the application. Different buildpacks exist for the different target runtime environments, such as HTML5, Node.js, Java, Python, and SAP HANA database.

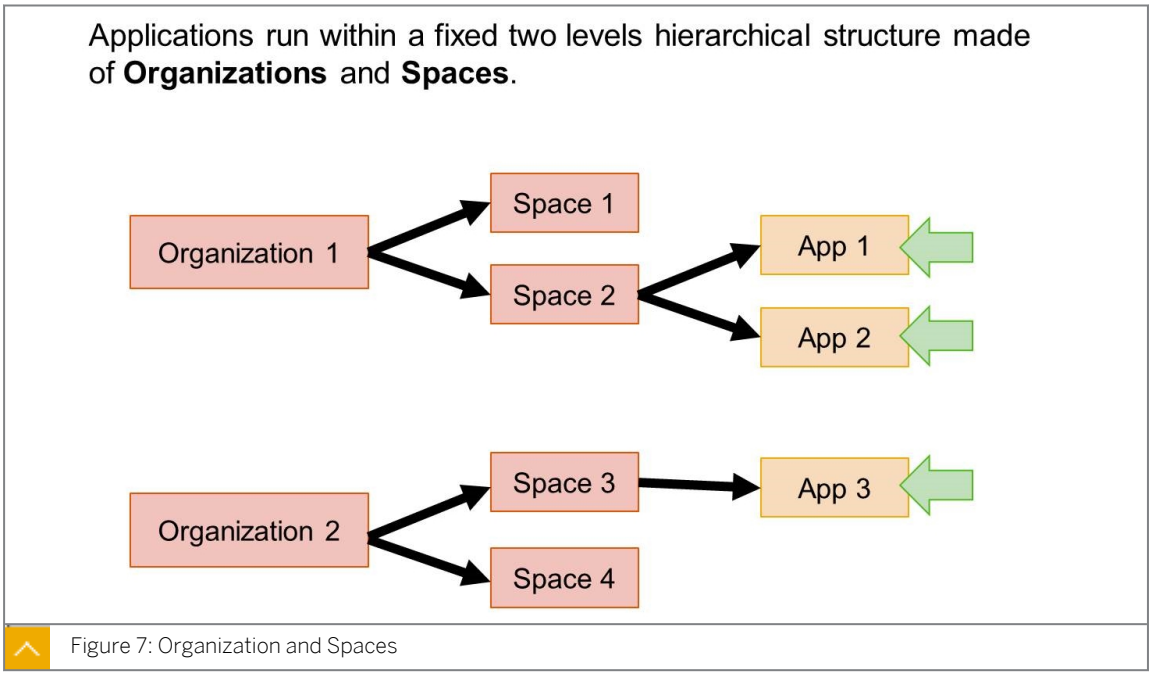


Figure 7: Organization and Spaces

Applications run within a fixed two level hierarchical structure made of **Organizations** and **Spaces**.

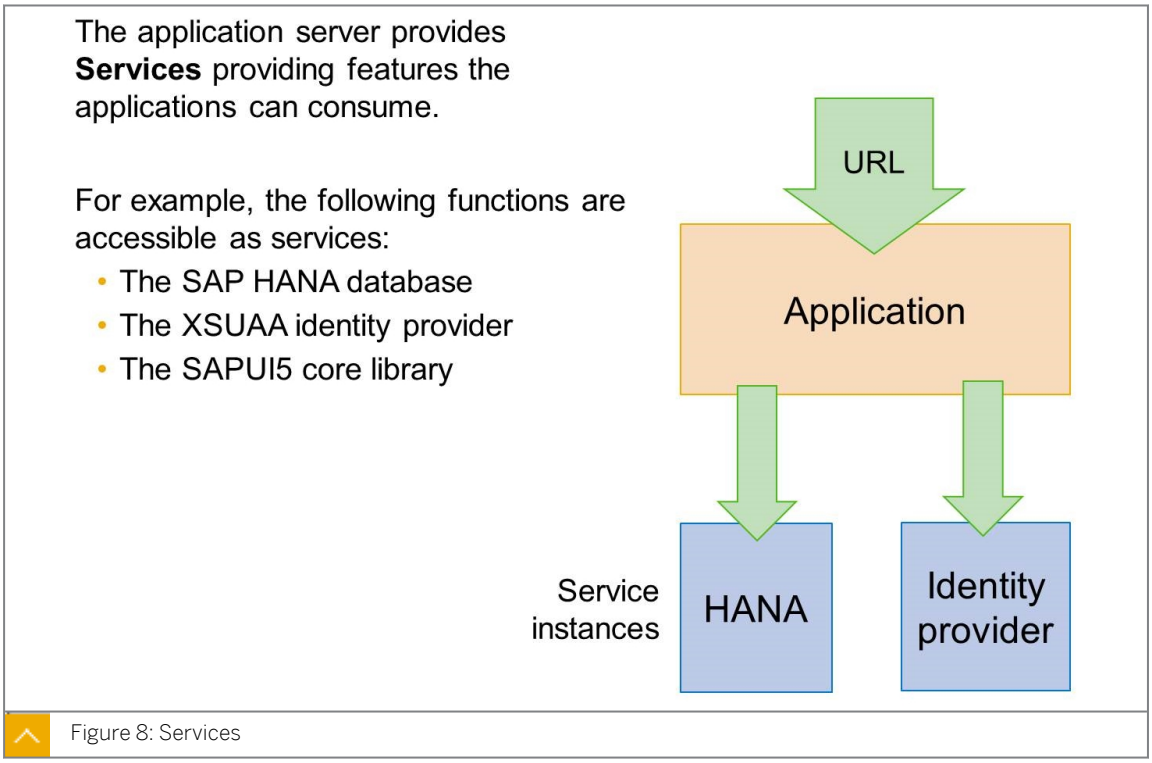


Figure 8: Services

The application server provides **Services**. Services provide features that the applications can consume.

For example, the following functions are accessible as services:

- The SAP HANA database
- The XSUAA identity provider

- The SAPUI5 core library

LESSON SUMMARY

You should now be able to:

- Describe the basic concepts of application architecture in SAP HANA

Unit 1
Lesson 3

Describing the Application Development Tools in SAP HANA

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the tools used by the application developer in SAP HANA

Describing the Application Development Tools

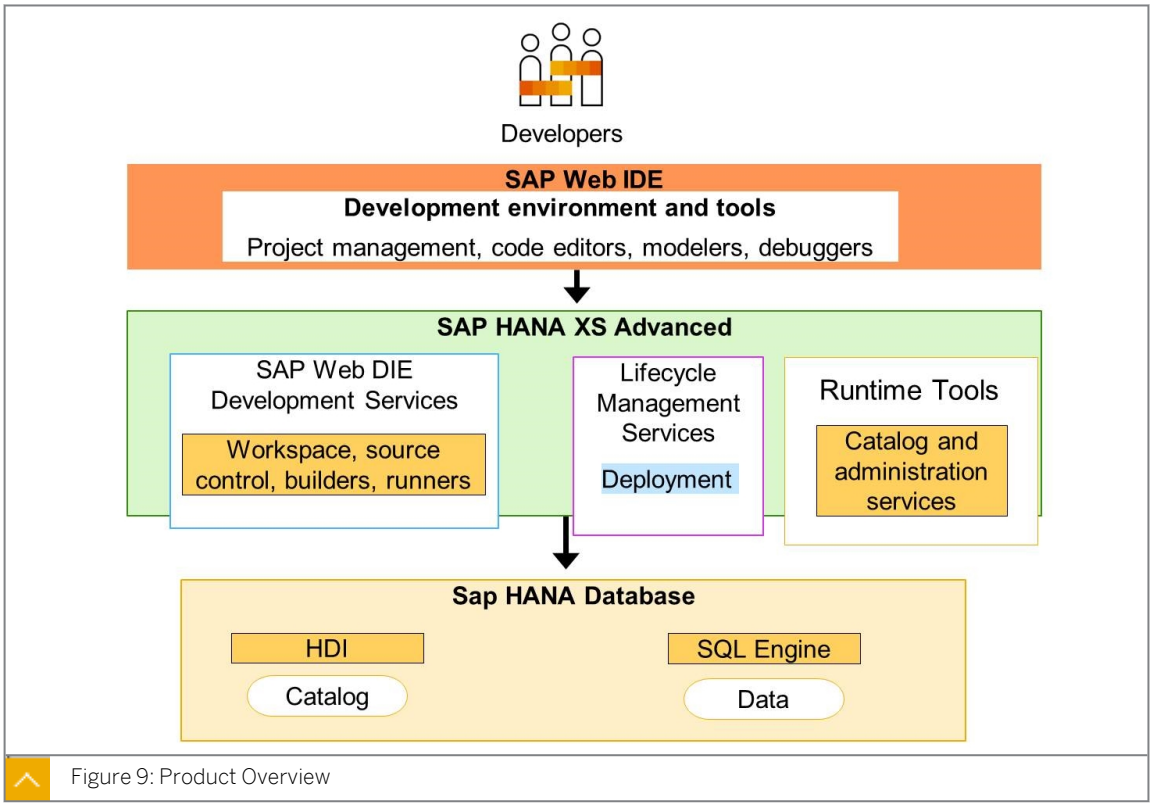


Figure 9: Product Overview

The SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) for the development of SAP HANA based applications. It is comprised of web-based or mobile UIs, business logic, and extensive SAP HANA data models.

The SAP Web IDE for SAP HANA supports developers who use XS Advanced, by providing a variety of tools.

These include the following:

- Syntax-aware editors for code
- SAP HANA artifacts

- Graphical editors for CDS data models
- Calculation views
- Inspection tools
- Testing tools
- Debugging tools

The SAP Web IDE for SAP HANA works with the SAP HANA Runtime Tools (HRTT), the SAP HANA Deployment Infrastructure (HDI), Application Lifecycle Management tools (ALM), and the XS Advanced runtime platform, as shown in the figure above.

To facilitate the development of multi-target applications (MTAs) for XS Advanced, the SAP Web IDE for SAP HANA provides the following features and tools:

- An integrated workspace
- Central management of application projects
- A comprehensive suite of the most up-to-date development tools
- Wizards and code templates to help you get started quickly and easily
- The automatic update, in real time, of dependencies between the development and build artifacts
- Build, run, and deployment tools

The SAP Web IDE for SAP HANA is designed to guide you through the development process required to deploy MTAs. This process includes the following high-level steps:

1. Set up an application project.
2. Develop your application modules.

   For example, for your data model, your application business logic, and your client user interface.
3. Build your application modules.
4. Run and debug your application modules.
5. Package the modules into an MTA.
6. Deploy the application to XS Advanced.

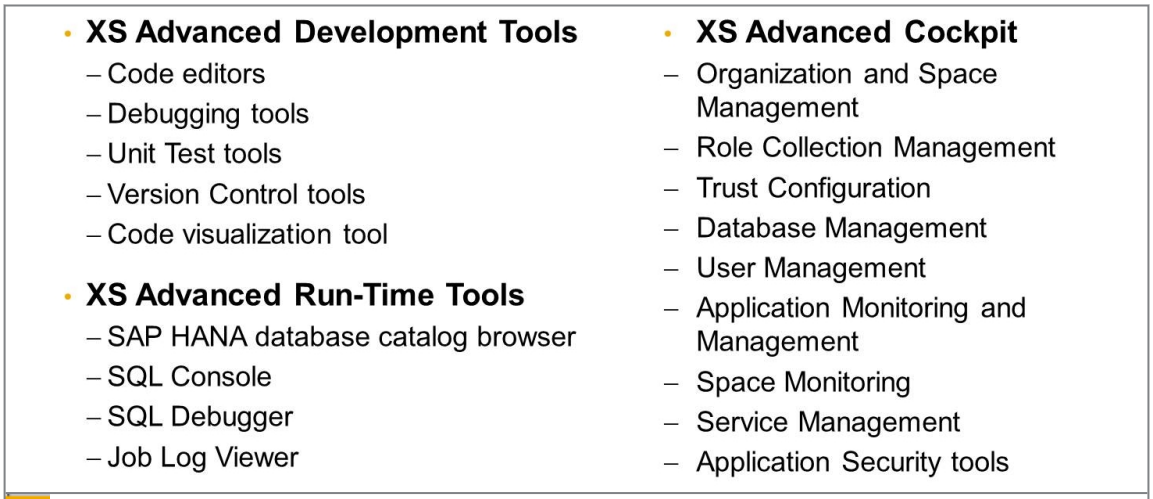


Figure 10: XS Advanced Application Development Tools

XS Advanced provides a selection of development and administration tools that can be used to help in the various phases of the design time development and runtime administration of XS Advanced. These tools include the following:

- Development tools
- Runtime tools
- Administration tools

Describing the XS Advanced Command Line Client

XS Advanced Command-Line Client

The XS Advanced Command-Line Client enables access to the XS Advanced runtime, which allows you to do the following:

- Deploy and manage applications
- Create and manage new users
- Maintain organizations and spaces

  In the XS Advanced runtime, an organization is like a team and a space is an area that assigned developers can use to develop applications.
- Maintain selected aspects of the runtime space to which you are assigned.
- Assign authorizations to the new users in organizations and spaces.

The XS Advanced Command-Line Client is installed by default on the SAP HANA server, but needs to be downloaded and installed locally to be able to connect from your development machine.

(xs.onpremise.runtime.client_<*platform*>-<*version*>.zip) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal (more info in SAP note 2242468)

Installation on your local development machine can be checked with the following command:

xs help

xs –v

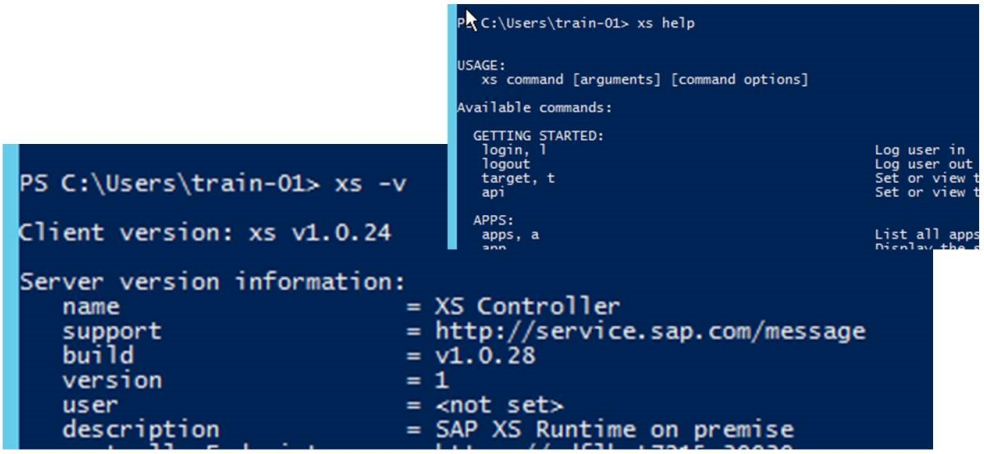


Figure 11: XS Advanced Command-Line Client

You can use the XS Advanced Command-Line Client to perform a wide-variety of developer and administrator-related tasks. For example, as a developer, you can use the XS Advanced Command-Line Client to connect to the XS Advanced runtime installed on the SAP HANA server, log on as a specific user, and deploy and manage your applications. If you have the correct role assigned, you can also maintain selected aspects of the runtime space you are assigned to. As an administrator, and with the appropriate roles assigned, you can use the XS Advanced Command-Line Client to create and manage new users, maintain organizations and spaces, and assign authorizations to new users in organizations and spaces.

Describing the XS Advanced Cockpit

The URLs of the XS Advanced Tools are “installation dependent” and can be either found using the command **xs app xsa-admin –urls** with the command-line client or by connecting to the XS Advanced Controller at port **3<instance>30** via web browser and navigating from there to the applications.

Figure 12: XS Advanced Cockpit: Administration and Monitoring Tools

A number of administration tools are available to maintain and manage the various components of the XS Advanced model runtime environment. The XS Advanced Cockpit contains the following tools:

- System-wide tools
- Space-related tools
- Application-related tools

System-Wide Tools

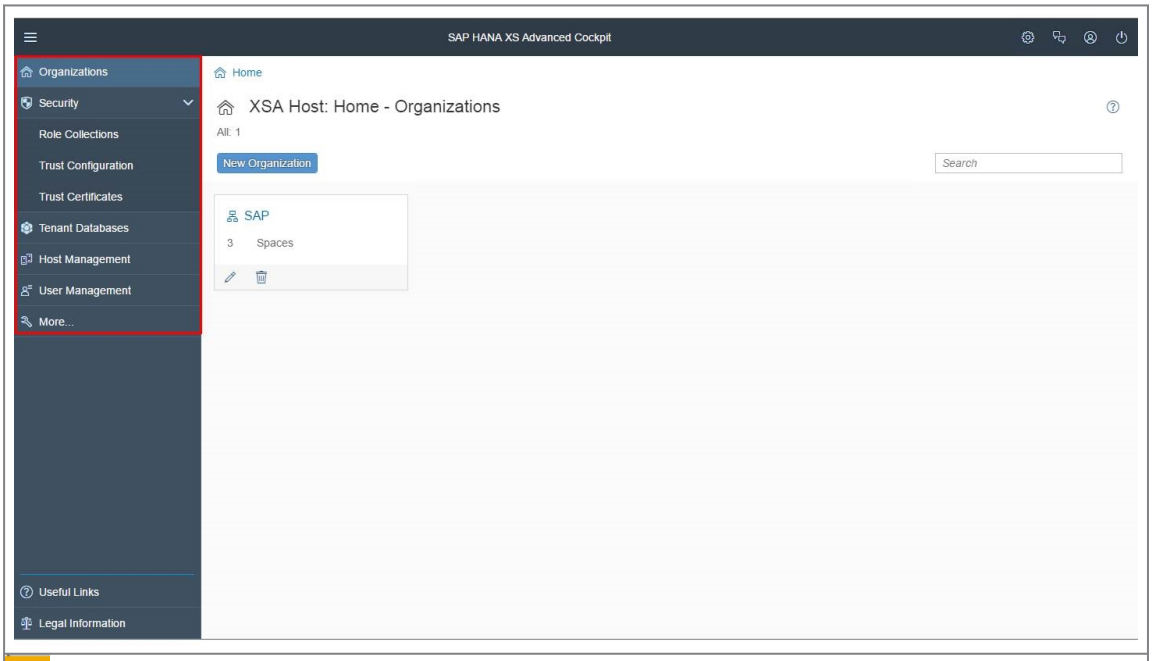


Figure 13: System-Wide Tools

The following are system-wide tools:

- Organization and Space Management

  View all existing organizations in the XS Advanced system, create new ones, and edit or delete them. For each organization, the contained spaces can be reviewed and maintained.

- Role Collection Manager

  Manage role collections for the entire system.

- Trust Configuration and Certificates

  Maintain identity providers used for Single Sign On (SSO) and manage system certificates.

- Tenant Databases

  Manage, maintain, and configure SAP HANA tenant databases for use with XS Advanced model applications.

- Host Management

  View the SAP HANA hosts that are pinned to XS Advanced applications or spaces.

- User Management

  Maintain and manage database and business users for XS Advanced. You can also view and manage role collections assigned to the users.

Space-Related Tools

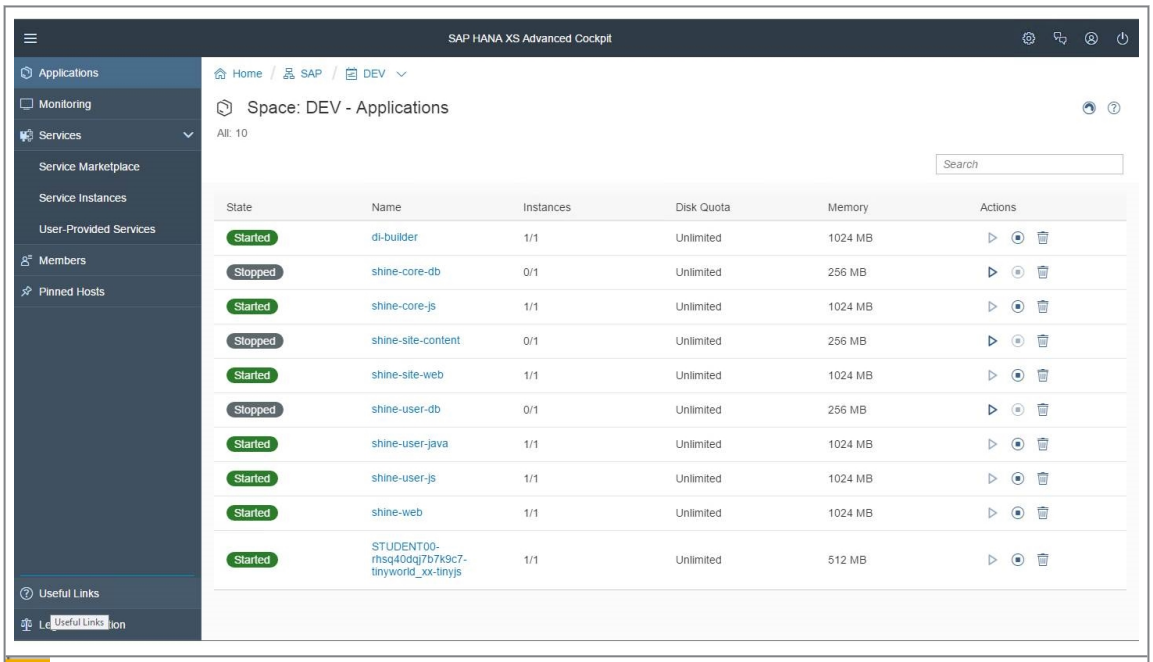


Figure 14: Space-Related Tools

The following are space-related tools:

- Application List and Monitor

  Manage all applications assigned to the current space. Here, you can start, stop, delete, access, and monitor the applications.

- Space Monitoring

  Monitor the system resources used by the applications running in the current space.

- Service Management

  List all available services for the space in the *Service Marketplace* and manage created service instances of the current space.

- Space Members

  Shows a list of the members of the space and their roles. A manager of the space can add and remove users, and select or deselect roles.

- Pinned Hosts

  Shows a list of the SAP HANA hosts that are pinned to XS Advanced applications or spaces.

Application-Related Tools

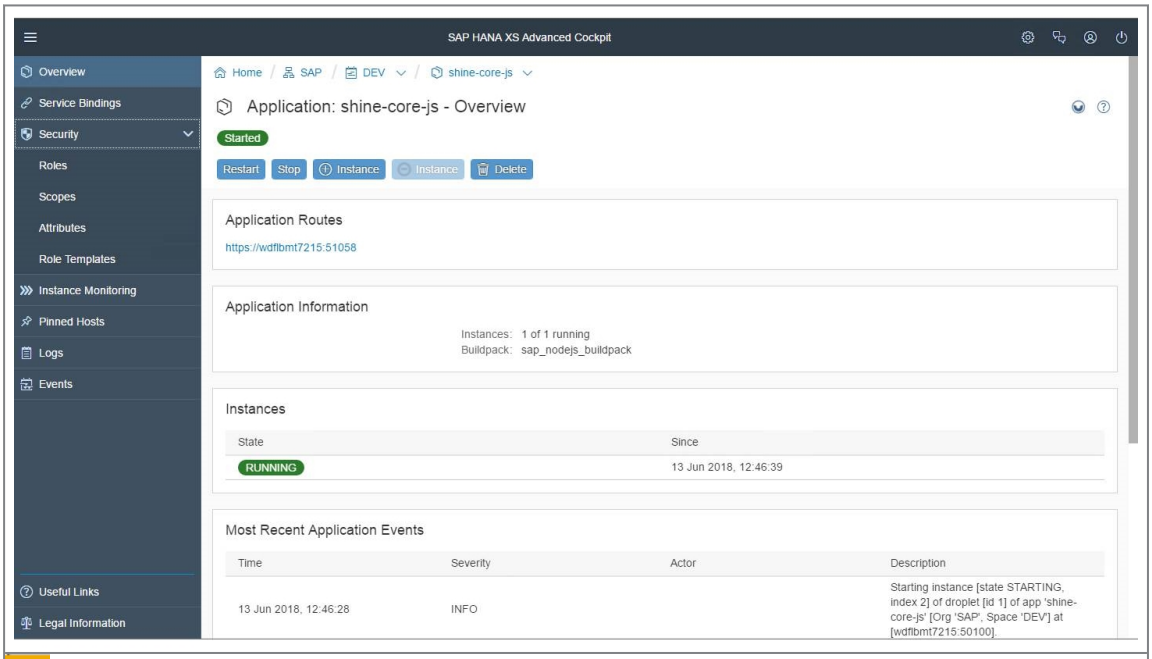


Figure 15: Application-Related Tools

The following are application-related tools:

- Overview

  The application details are displayed such as routes, events, and running instances. It's possible to start, restart, stop, and delete applications as well as add additional application instances.

- Service Bindings

  Lists the service instances bound to the application including service instance name and service plan and the specific service credentials.

- Application Security

  Manage the application-related roles, scopes, attributes, and role templates. The role templates are usable in the system-wide role collection manager.

- Instance Monitoring

  Monitor the system resources used by each instance of the application and its processes.

- Logs

  Monitor the system resources used by each instance of the application and its processes.

- Events

  Review the application events. These are records of the sequence of actions that you perform on the application. They include a timestamp, severity, the actor, and a description to analyze the state of an application though its lifecycle.

Describing the SAP Web IDE for SAP HANA

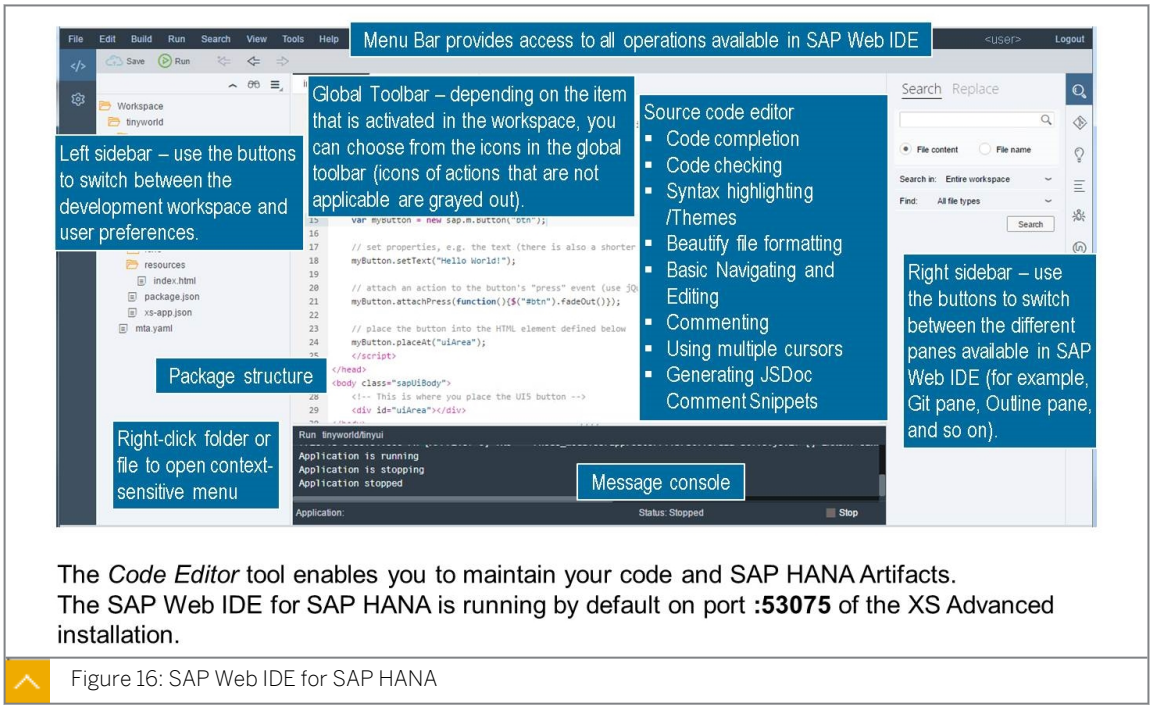


Figure 16: SAP Web IDE for SAP HANA

The SAP Web IDE for SAP HANA provides customizable editors in which you edit the code for your applications.

There are various editors to support different types of files, including:

- JavaScript
- XML
- xsodata
- CDS
- hdbprocedure

When you open a file, it displays in the appropriate editor for that file type. All of the editors support code completion.

Note the following suggestions for working with the Code Editor tool:

- Configuring the code editor

  Define the appearance and behavior of the code editor, and whether to autosave all changes in the SAP Web IDE for SAP HANA.

- Working in the code editor

  Use keyboard shortcuts and context menus to easily edit and navigate through your code and code comments.

- Generating JSDoc comment snippets

  You can generate a snippet for JavaScript function declaration that creates a template for documenting the function.

- Using code completion

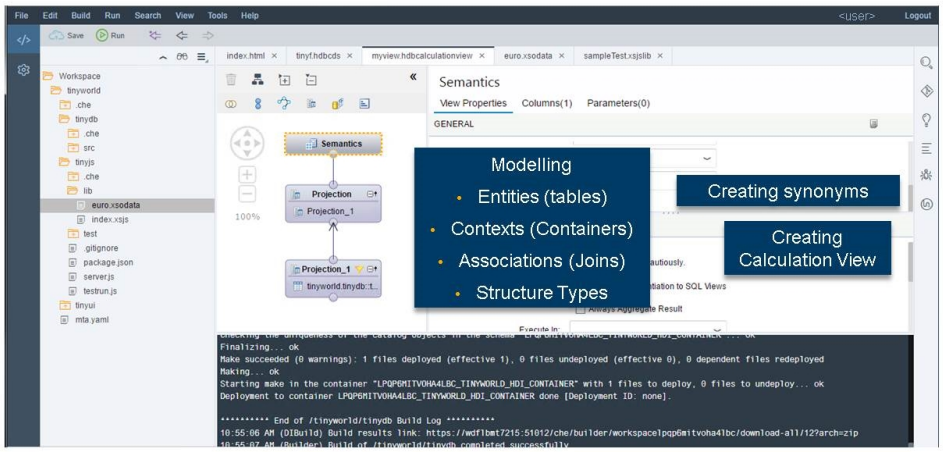
  The code completion feature assists you when you are writing your code, by preventing typos and other common mistakes.

- Checking code

  The SAP Web IDE for SAP HANA performs code checking, also known as validation, and displays errors as annotations.

- Locating objects in code

  The code editor allows you to locate objects or definitions of objects in code.



SAP Web IDE for SAP HANA (XS advanced model) offers graphical tools to data modelers to model artifacts that define data persistence objects using Core Data Services (CDS), Calculation Views and Synonyms.

Figure 17: SAP Web IDE for SAP HANA: Graphical Editor

The graphical editor can be used for the following:

- Creating Synonyms

  Synonyms are design-time files in the repository that provide data independence. After creating a synonym, you can bind this synonym with tables in different schemas.

- CDS Artifacts

  In the current version of the XS Advanced model, you can only use the graphical editor to model the following artifacts and elements:

  - Entities (tables)
  - Contexts (Containers)
  - Associations (Joins)
  - Structure Types
  - Calculation Views

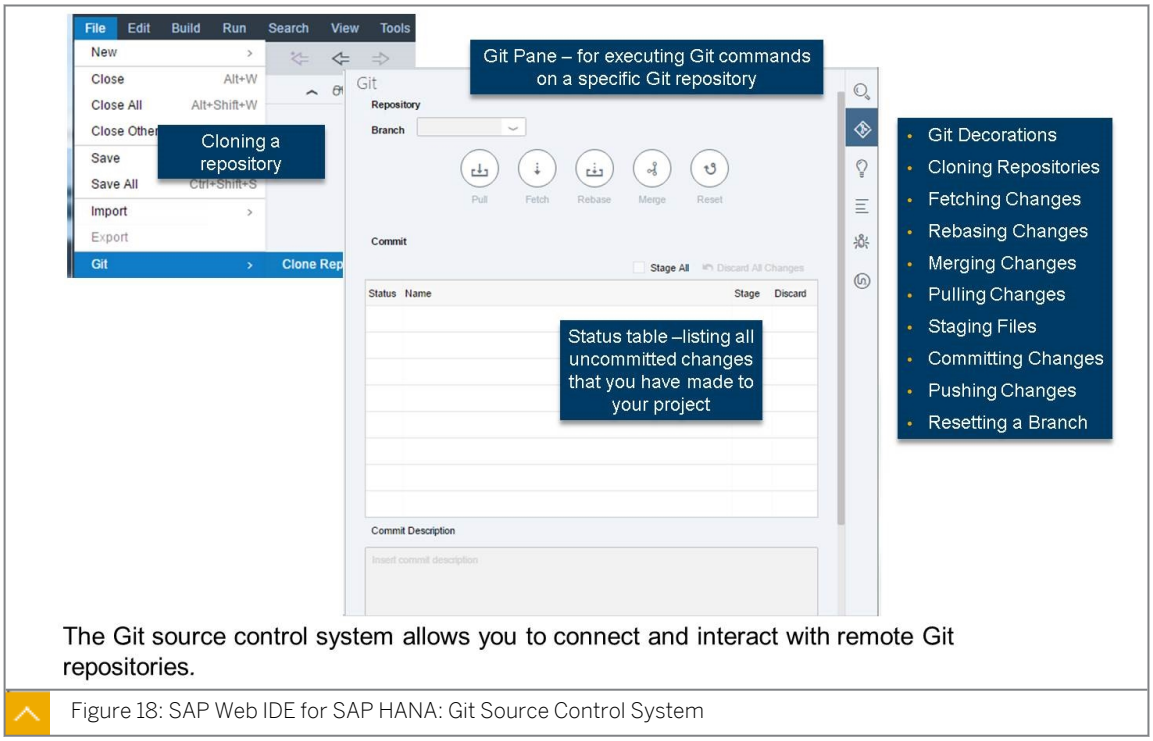


Figure 18: SAP Web IDE for SAP HANA: Git Source Control System

The SAP Web IDE for SAP HANA provides a graphical user interface for managing your source using Git.

It provides the following features:

- Decorations

  Any change in a file's status is reflected in the workspace by decorations.

- Cloning Repositories

  You can clone an existing Git repository into your workspace.

- Fetching Changes

  Fetching enables you to download objects and references from another repository into your local repository. You can then merge or rebase the changes into your project.

- Rebasing Changes

  Rebasing enables you to take all committed changes from one branch and incorporate them into a different branch.

- Merging Changes

  You can incorporate all changes from one branch into another, with a single commit.

- Pulling Changes

  Pulling is the same as fetching and merging. Pulling enables you to download objects and references from another repository into your local repository, and then merge the changes into your project.

- Staging Files

  The status table shows changed files, and lets you select files to stage.

- Committing Changes

You can commit changes to the repository locally.

- Pushing Changes

  The push option incorporates all un-synced, committed changes into the remote branch of the currently checked out local branch. The number of un-synced committed changes is displayed next to the repository name. All tags created within the open repository are pushed.

- Resetting a Branch

  You can delete all new objects and references that were added to an existing local branch to make it identical to its remote branch.

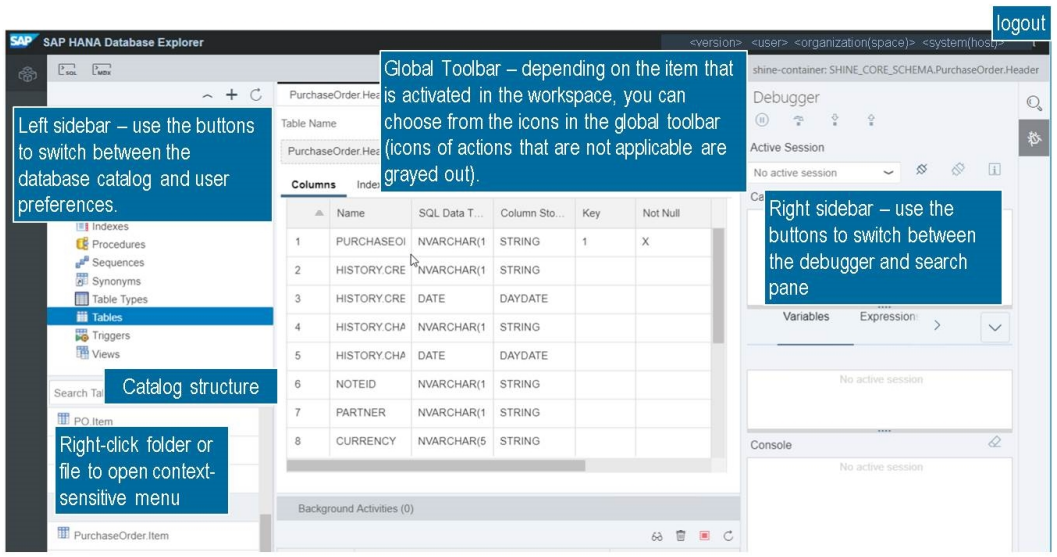


Figure 19: SAP Web IDE for SAP HANA: Database Explorer

The SAP HANA Database Explorer is part of the SAP Web IDE for SAP HANA and provides a set of functions to access the SAP HANA database.

The SAP HANA Database Explorer can either be connected to an isolated database container or to an entire database using the credentials of a database user. While a connection to a single container will allow you to only see the container content which corresponds to a database schema, a "normal" database connection makes it possible to explore multiple schemas as long as the connected user has the right to see them.

LESSON SUMMARY

You should now be able to:

- Describe the tools used by the application developer in SAP HANA

Introducing the SAP HANA Express Edition

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Introduce SAP HANA express edition

Introducing the SAP HANA Express Edition

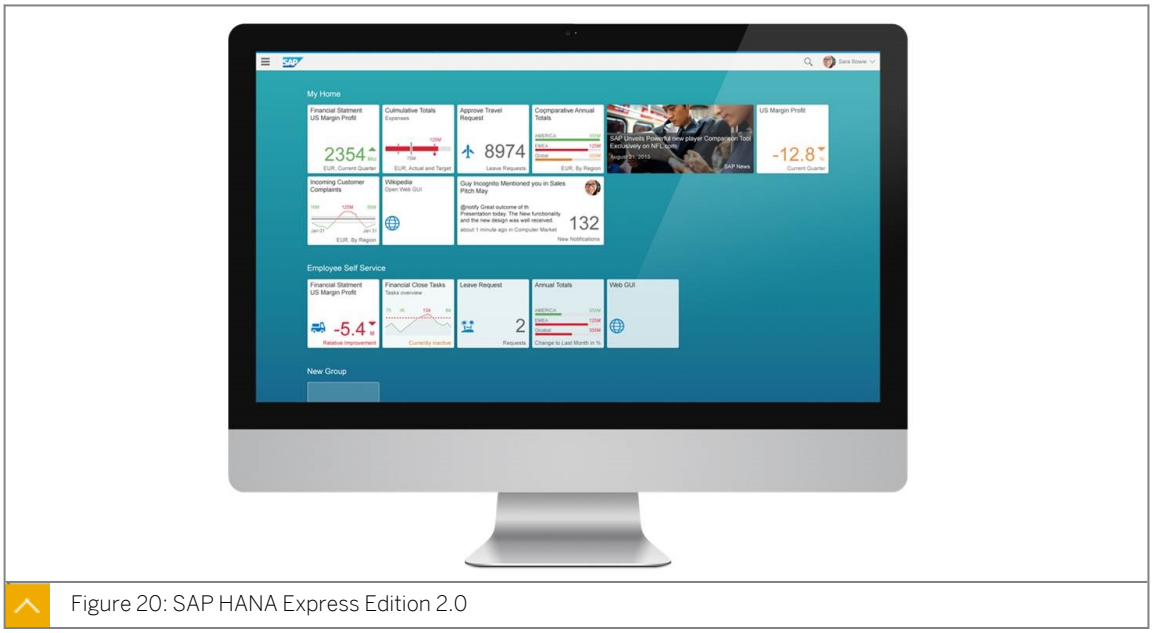


Figure 20: SAP HANA Express Edition 2.0

SAP HANA express edition 2.0 is a streamlined version of SAP HANA, optimized for fast and continuous application development. It can be used on a laptop, personal computer, server, or in the cloud, with flexible development and deployment options.

SAP HANA express edition 2.0 is based on SAP HANA SPS 2. It provides 32 GB of memory, which can be expanded to 128 GB for a fee.

SAP HANA express edition 2.0 contains all of the essential SAP HANA features, but has a smaller footprint. There is no need for hardware certification; a minimum 16 GB memory is recommended. It also allows production use.

The installation options for SAP HANA express edition 2.0 are as follows:

- Virtual machine packages for Windows, Mac OS X, or other virtual machine supported operating systems
- Binary package for Linux (SUSE and Red Hat)
- Cloud appliance library (CAL), available through cal.sap.com

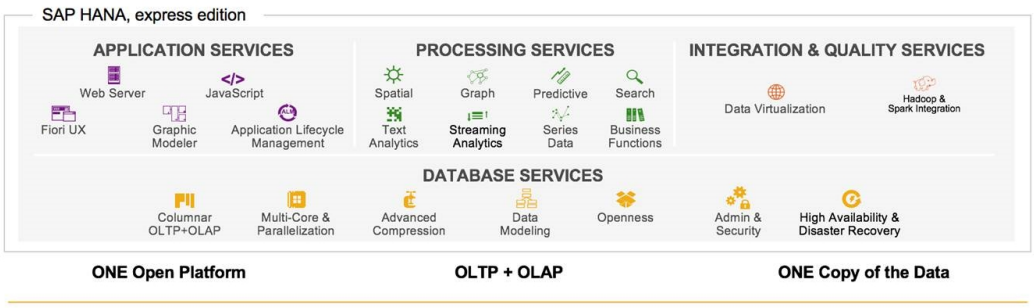
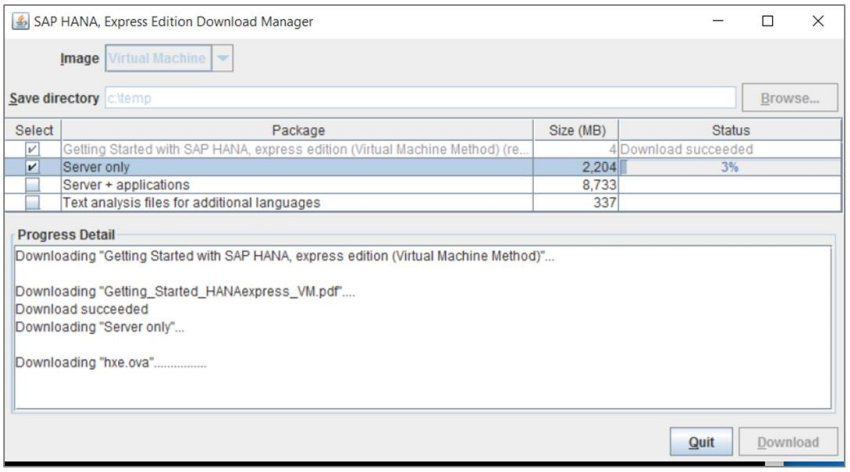


Figure 21: SAP HANA Express Edition 2.0: Feature-Rich Edition of SAP HANA



Fast and simple model to download and install SAP HANA, express edition.

Figure 22: Simplified Downloading and Uploading

The SAP HANA Lifecycle Management (HDBLCM) tool can download updates automatically and perform an in-place upgrade for SAP HANA 2.0 and later releases. The Download Manager allows users to separately download the SAP HANA Client Installation, and the SAP HANA XS Client Installation.

Feature Description:

- SAP HANA, express edition is deployable in a variety of public cloud providers

Details:

- **SAP CAL delivery** for AWS, Azure.
- **Smaller footprint** also enables **self installation** in cloud provider of choice (using binary installation)
- **New Tutorials** for running HXE in cloud providers.

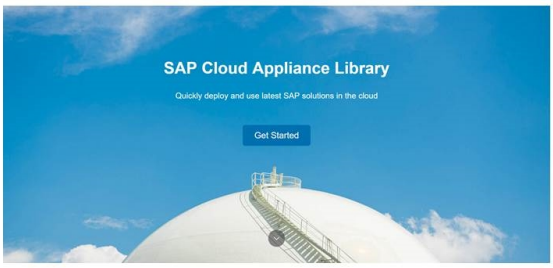


Figure 23: Cloud Convergence

Feature Description:

- A set of features allowing the system to capture and transfer usage and diagnostic data unobtrusively to SAP product development

Details:

- Collects **anonymized usage data** (without any personal information) from HANA Express instances for improving SAP HANA.
- **Commands to enable, disable** this functionality have been added to SAP HANA

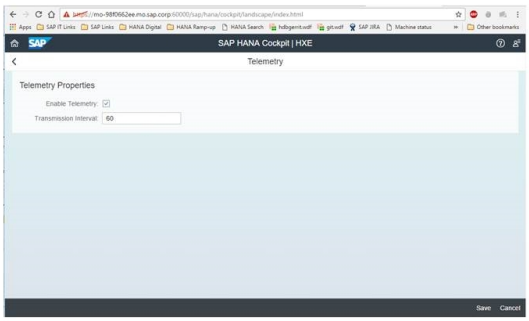


Figure 24: Telemetry

LESSON SUMMARY

You should now be able to:

- Introduce SAP HANA express edition

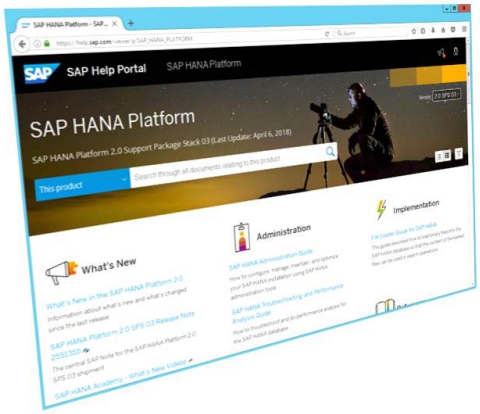
Describing the Information Sources for SAP HANA Developers

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the information sources for SAP HANA developers

Documentation, References, and Information Sources



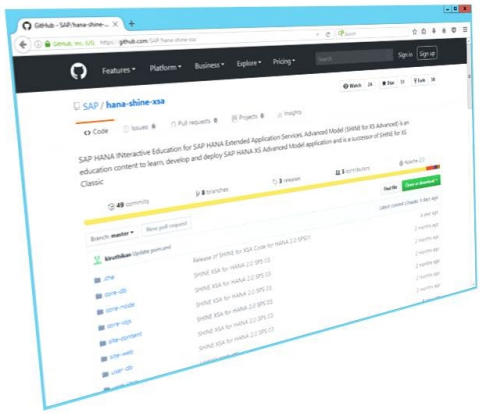
SAP Help Portal

Official SAP Help Documentation:
https://help.sap.com

All HANA Related guides:
https://help.sap.com/hana

- Developer Information Map
- Developer Guide for XS Advanced Model
- SQL and System Views Reference
- SQLScript Reference
- Security Guide

Figure 25: SAP HANA Developer Guides and Documentation



SAP HANA Interactive Education

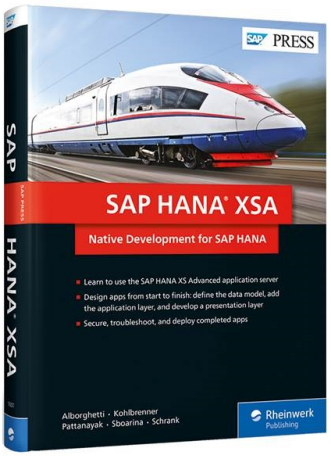
SAP HANA INteractive Education for SAP HANA Extended Application Services, Advanced Model (SHINE for XS Advanced) is an education content to learn, develop and deploy SAP HANA XS Advanced Model application

Clone it directly from GitHub to the Web IDE and deploy it to your XS Advanced system.

Many working code examples to get inspired…

https://github.com/SAP/hana-shine-xsa

Figure 26: SAP HANA Interactive Education (SHINE)



SAP HANA XSA

Native Development for SAP HANA

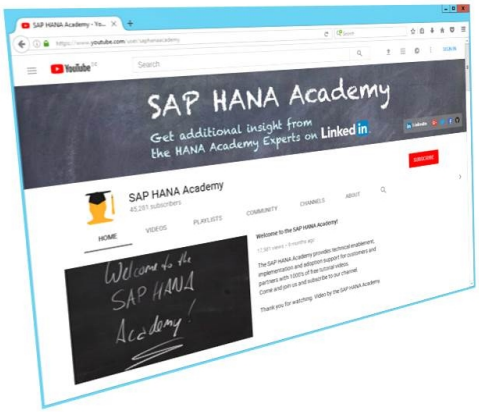
written by Francesco Alborghetti, Jonas Kohlbrenner, Abani Pattanayak, Dominik Schrank, Primo Sboarina

SAP-Press, 600 pages, 2018

ISBN 978-1-4932-1602-4

https://www.sap-press.com/sap-hana-xsa_4500/

Figure 27: SAP HANA XSA – Native Development for SAP HANA



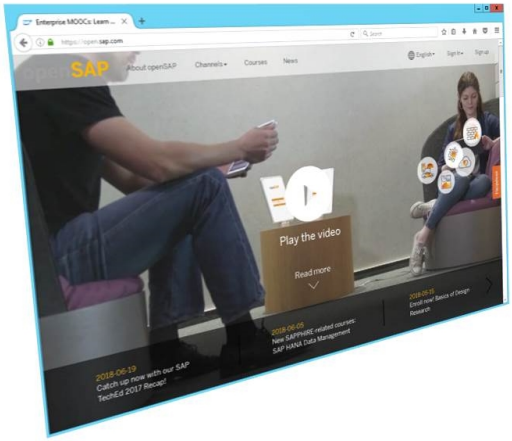
SAP HANA Academy

The SAP HANA Academy provides Technical Enablement, Implementation and Adoption Support for Customers and Partners with 1000's of free tutorial videos.

With dedicated Playlist for XS Advanced development, Predictive Analytics and many more…

https://www.youtube.com/user/saphanaacademy

Figure 28: SAP HANA Academy



openSAP

provides free Massive Open Online Courses (MOOCs) to everyone interested in learning about SAP's latest innovations and how to survive in the digital economy

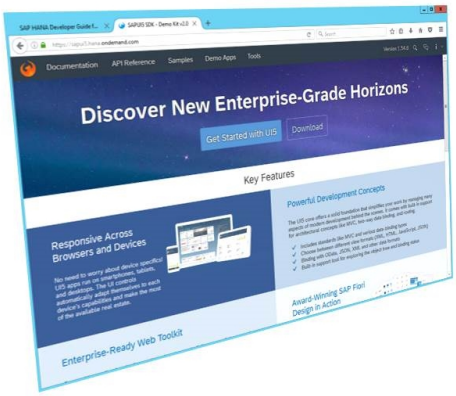
https://open.sap.com/

Native HANA development Courses:
https://open.sap.com/courses/hana6
https://open.sap.com/courses/hana5

SAPUI5 Courses:
https://open.sap.com/courses/ui51
https://open.sap.com/courses/3dmv1

JAVA Object Oriented Programming:
https://open.sap.com/courses/java1

Figure 29: openSAP Online Courses



SAPUI5 References

https://sapui5.hana.ondemand.com/

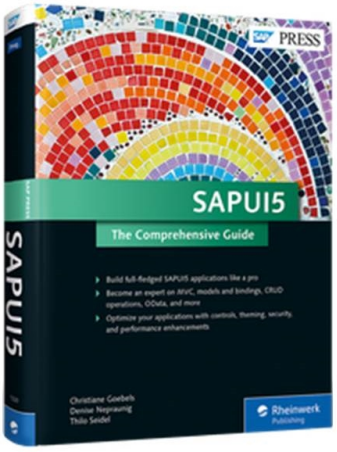
Official Documentation:
https://sapui5.hana.ondemand.com/#/topic

API Reference:
https://sapui5.hana.ondemand.com/#/api

UI Control samples:
https://sapui5.hana.ondemand.com/#/controls

Tools like Icon Explorer, UI5 Inspector etc…
https://sapui5.hana.ondemand.com/#/tools

Figure 30: SAPUI5 References



SAPUI5

The Comprehensive Guide

written by Christiane Goebels, Denise Nepraunig, Thilo Seidel

SAP-Press, 672 pages, 2016

ISBN 978-1-4932-1320-7

https://www.sap-press.com/sapui5_3980/

Figure 31: SAPUI5 – The Comprehensive Guide

LESSON SUMMARY

You should now be able to:

- Describe the information sources for SAP HANA developers

Learning Assessment

1. What are the prerequisite skills needed to develop SAP HANA applications, if they are based on a Fiori UI and use Node.js in the application server?

   Choose the correct answers.

   - ☐ **A** JavaScript language
   - ☐ **B** ABAP language
   - ☐ **C** HTTP protocol
   - ☐ **D** SQL language
   - ☐ **E** SAP Screen Personas

2. In XS Advanced or Cloud Foundry, what are the two levels of the hierarchy that contain the applications ?

   Choose the correct answers.

   - ☐ **A** Entity
   - ☐ **B** Space
   - ☐ **C** Container
   - ☐ **D** Organization

3. Which of the following statements are true about the SAP Web IDE for SAP HANA?

   Choose the correct answers.

   - ☐ **A** It is a browser-based IDE.
   - ☐ **B** It provides wizards and code templates to help you get started more quickly and easily.
   - ☐ **C** It does not provide debugging facilities to test your applications.
   - ☐ **D** You can package modules into a multi-target application.

4. Which of the following tasks can be performed from the XS Advanced Command Line Interface?

   Choose the correct answers.

   - ☐ **A** As a developer, you can connect to the XS Advanced runtime installed on SAP HANA.
   - ☐ **B** As a developer, you can log on as a specific user.
   - ☐ **C** As a developer, you can deploy and manage your applications.
   - ☐ **D** As an administrator, you can create and manage users, organizations, and spaces.

5. Which of the following statements are true about SAP HANA Express Edition?

   Choose the correct answers.

   - ☐ **A** It is free to use – up to 32 GB memory use, and expandable to 128 GB for a fee.
   - ☐ **B** It can only run on certified hardware.
   - ☐ **C** It can be used in production scenarios.
   - ☐ **D** It is packed with essential SAP HANA features, yet utilizes a smaller footprint.

6. Which information source is based on YouTube videos?

   Choose the correct answer.

   - ☐ **A** SAP Help Portal
   - ☐ **B** SAP HANA Interactive Education
   - ☐ **C** openSAP
   - ☐ **D** SAP HANA Academy

Learning Assessment - Answers

1. What are the prerequisite skills needed to develop SAP HANA applications, if they are based on a Fiori UI and use Node.js in the application server?

   Choose the correct answers.

   - [X] A JavaScript language
   - [ ] B ABAP language
   - [X] C HTTP protocol
   - [X] D SQL language
   - [ ] E SAP Screen Personas

   That is correct! The prerequisite skills that you need to have before you study SAP HANA application development cover HTTP, HTML, JavaScript, Node.js, Express.js and SQL.

2. In XS Advanced or Cloud Foundry, what are the two levels of the hierarchy that contain the applications ?

   Choose the correct answers.

   - [ ] A Entity
   - [X] B Space
   - [ ] C Container
   - [X] D Organization

   That is correct! The two levels are Organization and Space.

3. Which of the following statements are true about the SAP Web IDE for SAP HANA?

   Choose the correct answers.

   [X] A It is a browser-based IDE.

   [X] B It provides wizards and code templates to help you get started more quickly and easily.

   [ ] C It does not provide debugging facilities to test your applications.

   [X] D You can package modules into a multi-target application.

   That is correct! The SAP Web IDE for SAP HANA allows you to run and debug your applications.

4. Which of the following tasks can be performed from the XS Advanced Command Line Interface?

   Choose the correct answers.

   [X] A As a developer, you can connect to the XS Advanced runtime installed on SAP HANA.

   [X] B As a developer, you can log on as a specific user.

   [X] C As a developer, you can deploy and manage your applications.

   [X] D As an administrator, you can create and manage users, organizations, and spaces.

   That is correct! All of the tasks listed can be performed by the respective users. The XS Advanced Command Line Interface (XS CLI) is a versatile tool for all types of users.

5. Which of the following statements are true about SAP HANA Express Edition?

   Choose the correct answers.

   [X] A It is free to use – up to 32 GB memory use, and expandable to 128 GB for a fee.

   [ ] B It can only run on certified hardware.

   [X] C It can be used in production scenarios.

   [X] D It is packed with essential SAP HANA features, yet utilizes a smaller footprint.

   That is correct! SAP HANA Express Edition is a streamlined version of SAP HANA, optimized for fast and continuous application development. It is available for free up to 32 GB memory use on a laptop, personal computer, server, or available in the cloud, with flexible development and deployment options.

6. Which information source is based on YouTube videos?

   Choose the correct answer.

   - [ ] A SAP Help Portal
   - [ ] B SAP HANA Interactive Education
   - [ ] C openSAP
   - [x] D SAP HANA Academy

   That is correct! The SAP HANA Academy information source is based on YouTube videos.

UNIT 2 Developing a Basic Multi-Target Application

UNIT OBJECTIVES

- Describe the basic concepts about the MTA development project
- Describe the information contained in the MTA Development Descriptor mta.yaml file
- Describe introductory concepts required to use the Node.js module in the SAP Web IDE for SAP HANA
- Create, run, export, and deploy a Node.js module saying Hello World
- Debugging Node.js code using the SAP Web IDE debugger

Introducing the Multi-Target Application

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the basic concepts about the MTA development project

Introducing the Multi-Target Application Development Project

It is a good practice to use separate applications for the 3 tiers: front-end, application and database. So a full-stack application is typically made of the combination of multiple applications and service instances.

That's why SAP extended the Cloud Foundry standard, introducing the concept of **Multi-Target Application**.

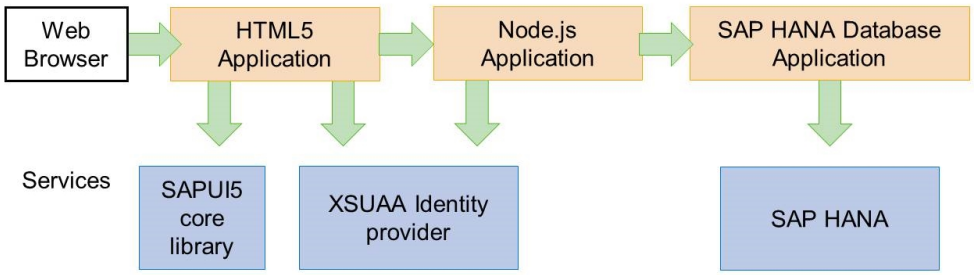


Figure 32: The Multi-Target Application

It is a good practice to use separate applications for the three tiers: front-end, application, and database. Therefore, a full-stack application is typically made of the combination of multiple applications and service instances.

Because of this, SAP extended the Cloud Foundry standard by introducing the concept of a multi-target application (MTA).

Development tools provided by SAP allow you to manage multiple applications (as "modules") in a unique development project, and deploy them via a unique archive file.

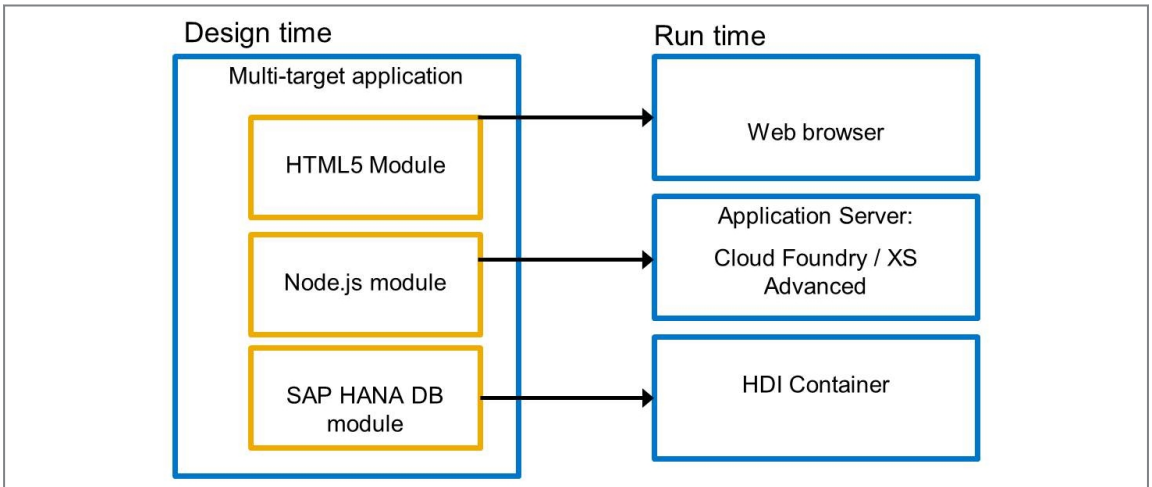


Figure 33: The MTA in the SAP Web IDE for SAP HANA

In the SAP Web IDE for SAP HANA, you create a development project called Multi-Target Application (MTA) project.

Within the MTA project, you create modules. Different type of modules may exist, for example, HTML5, Java, Node.js, or SAP HANA Database.

When deployed, every module of the project will become a separate application in Cloud Foundry or XS Advanced, and each module will have its own buildpack, development language, and runtime environment.

For example:

- HTML5 modules are served as static files and executed in the Web Browser
- Node.js and Java modules are executed in the Application Server (Cloud Foundry / XS Advanced).
- SAP HANA database modules generate database objects in the SAP HANA database.

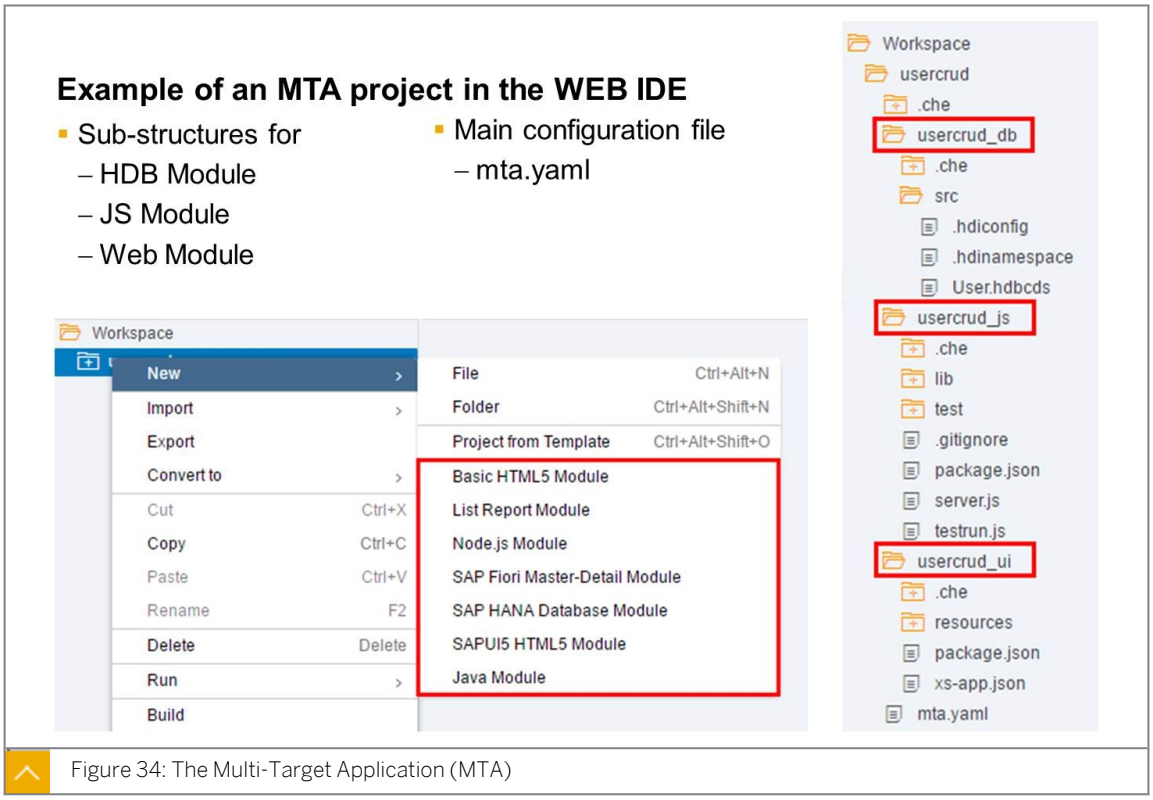


Figure 34: The Multi-Target Application (MTA)

LESSON SUMMARY

You should now be able to:

- Describe the basic concepts about the MTA development project

Describing the MTA Development Descriptor File mta.yaml

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the information contained in the MTA Development Descriptor mta.yaml file

Describing the MTA Development Descriptor File

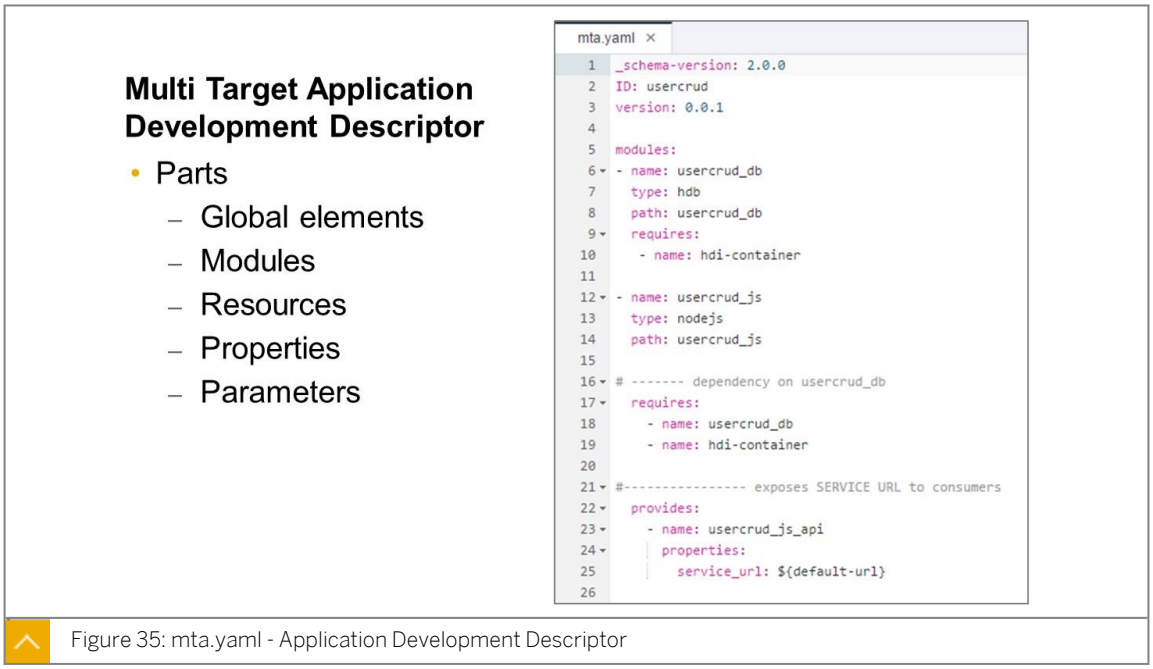


Figure 35: mta.yaml - Application Development Descriptor

MTAs are defined in a design time development descriptor. The development descriptor (mta.yaml) is used to define the elements and dependencies of an XS Advanced compliant MTA.

The MTA development descriptor (mta.yaml) is used by SAP HANA GUI tools to generate the deployment descriptor (mtad.yaml), which is required to deploy an MTA to the XS Advanced runtime.

The MTA development descriptor contains the following parts:

- Global elements

  These include the application identifier and version, a description (optional), copyrights, author, and so on.

- Modules

Modules created in the application, such as the SAP HANA database module and the Node.js module, including name, type, path, and requirements on other modules.

- Resources

  Dependent services which are not provided by the application, such as XS Advanced User Account and Authentication (XSUAA), XS Advanced HANA Deployment Infrastructure (XSHDI) container, and XSJob-Scheduler.

- Properties

  These can be specified when the value has to be determined during the deployment, for example, for generated URLs of other services or API keys.

- Parameters

  Reserved variables of a module, which can be accessed by other modules, for example, user, app-name, default-host, or default-uri using the placeholder notation. Parameters can be read-only or read/write enabled.

Global Elements

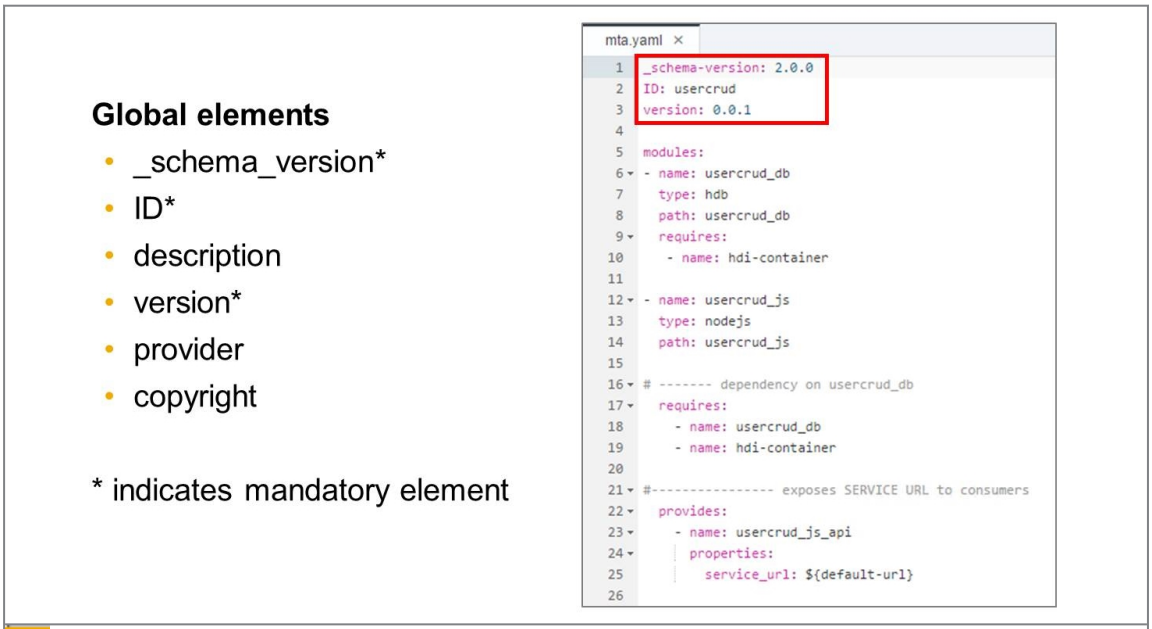


Figure 36: mta.yaml - Global Elements

- _schema-version

  Specifies the version of the MTA descriptor in the following schema: <major>.<minor>.<patch> Indicating a major version is enough

- ID

  Mandatory string to identify the application

- Description

  Optional description text

- Version

  Mandatory version of the application: <major>.<minor>.<patch>

- Provider

  Optional string to specify the organization providing the application

- Copyright

  Optional copyright information

Modules

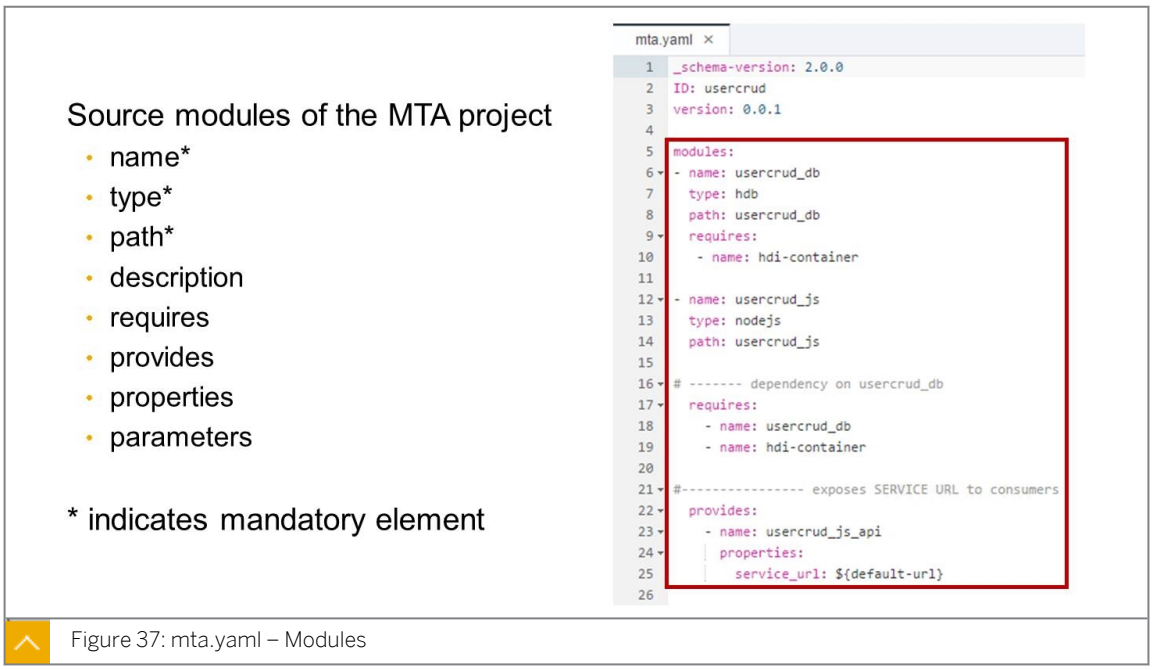


Figure 37: mta.yaml – Modules

Within the MTA development descriptor, the modules element declares the source modules of the MTA project.

- Name

  Mandatory name of the module. Unique in the descriptor file

- Type

  Mandatory content of the module, for example, HDB, Node.js, JAVA, HTML5

- Path

  Mandatory file system path starting from the applications root director

- Description

  Optional description text

- Requires

  Optional section containing required sources of other modules

- Provides

  Optional section containing configuration data used by other modules

- Properties

  Optional named variable containing application-specific configuration data

- Parameters

  Optional named variable to be used by the deployer, for example, the amount of memory for the module.

Here is the list of available Module Types:

Module Type	Platform	Result
nodejs javascript.nodejs	XS/CF	Node.js run time
java	XS/CF	Java run time
java.tomcat	XS/CF	Tomcat run time of sap_java_buildpack
java.tomee	XS/CF	TomEE run time of sap_java_buildpack
python	XS/CF	Python run time
com.sap.xs.hdi	XS/CF	HDI content activation
html5	XS/CF	Node.js runtime for application router

Figure 38: mta.yaml - Module Types

Resources

Dependency declaration on external resources

- name*
- type*
- description
- properties
- parameters

* indicates mandatory element

mta.yaml

```
31
32 - name: usercrud_ui
33   type: html5
34   path: usercrud_ui
35
36 # -- requires usercrud_js service URL
37   requires:
38     - name: usercrud_js_api
39       group: destinations
40       properties:
41         name: usercrud_js_be
42         url:  ~{service_url}
43
44
45 resources:
46 - name: hdi-container
47   type: com.sap.xs.hdi-container
48
```

Figure 39: mta.yaml – Resources

A resource is something that is required by a module of the MTA at runtime, or at deployment time, but not provided inside the MTA. More precisely, an MTA descriptor declares a resource dependency, not the resource itself. Sometimes they are referred to as Backing Services.

- Name

  Mandatory name of the module, unique in the descriptor file.

- Type

  Mandatory content of the resource, for example, com.sap.xs.uaa or com.sap.xs.hdi-container

- Description

Optional description text

- Properties

  Optional named variable containing application-specific configuration data

- Parameters

  Optional named variable to be used by the deployer, for example, the amount of memory for the module

Here is the list of available Resource Types:

Resource Type	Service	Service Plan	Created Service
com.sap.xs.hana-sbss	hana	sbss	Service broker security
com.sap.xs.hana-schema	hana	schema	Plain schema
com.sap.xs.hana-securestore	hana	securestore	SAP HANA secure store
com.sap.xs.hdi-container	hana	hdi-shared	HDI container
com.sap.xs.jobscheduler	jobscheduler	default	Job Scheduler
com.sap.xs.uaa	xsuaa	default	Global UAA service
com.sap.xs.fs	fs-storage	free	File system storage
com.sap.xs.uaa-devuser	xsuaa	devuser	Development-user UAA service
com.sap.xs.uaa-space	xsuaa	space	UAA service for a space
com.sap.xs.sds	sds	default	Smart data streaming

Figure 40: mta.yaml – Resource Types

Properties

Values specified during deployment

- <key>: <value>
- use defined parameters via the placeholder ($) notation *${<parameter>}*
- refer to property values via *~{<property>}*

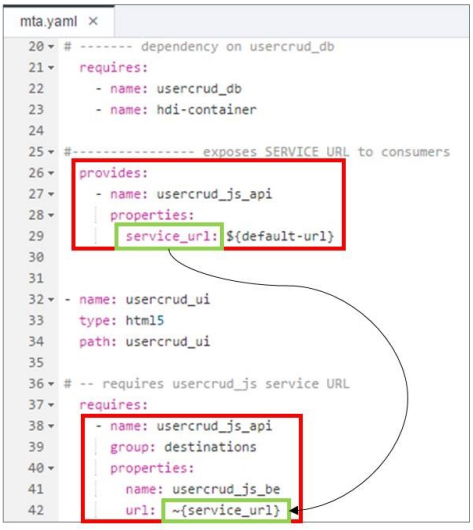


Figure 41: mta.yaml – Properties

Module properties are (possibly structured) key-value pairs that must be injected (made available) to the respective module at runtime. No assumption is made on how the MTA deployer injects the values. Many operating systems and languages, such as C, Java, node.js, and Python, support a concept of environment variables, but other methods will do also. For example, JNDI in the Java context.

> Example: JNDI in the Java context.
>
> In this example, the HTML5 module requires the Node.js module, which provides the service url that the HTML5 front-end is connecting to. The Node.js module defines the property *service_url* and uses a system placeholder *"${default-url}"* to fill with its URL that is generated during deployment. The HTML5 module refers to this property *"service_url"* and defines itself a property "url" to be used by the front-end.

Parameters

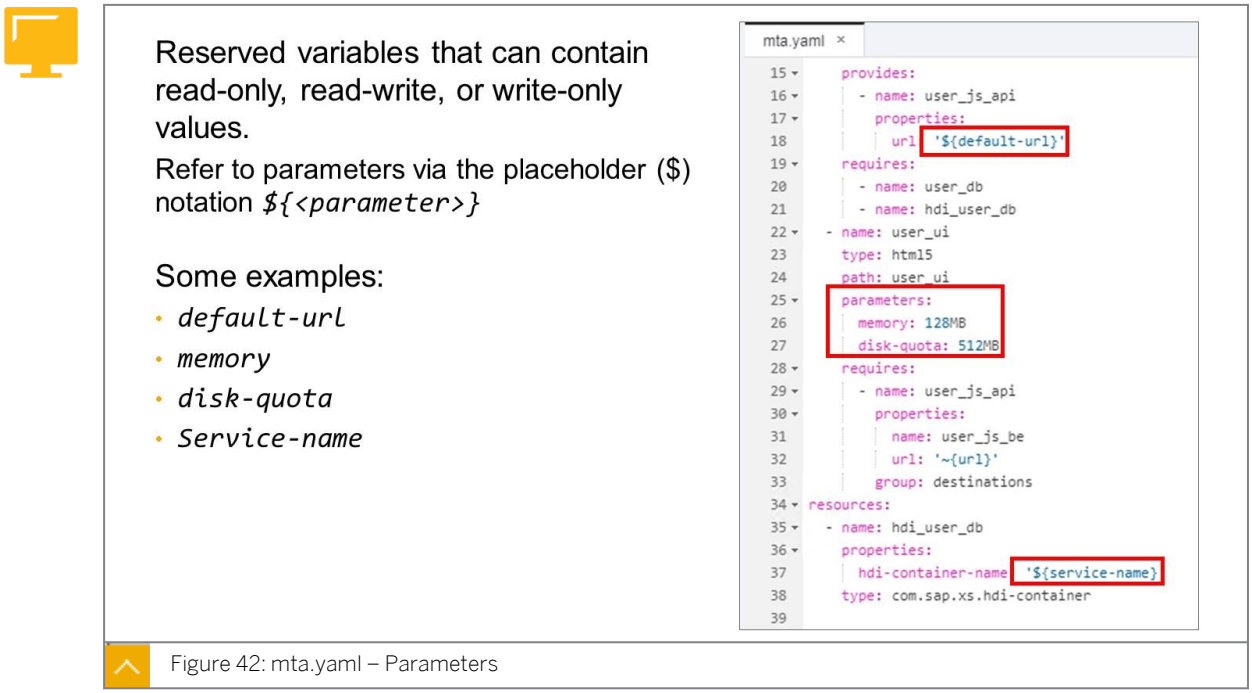


Figure 42: mta.yaml – Parameters

Parameters are reserved variables which influence the behavior during the deployment process and/or during runtime. A parameter can either be read-only, which is the case for most system parameters, read/write, or write-only.

In the figure above we specify the *memory* and *disk-quota* parameters for the user_ui module, which advises the deployer to grant 128 MB of memory and 512 MB of disk-space to the application.

In addition, we refer to the *service-name* and *default-url* parameters which are filled by the system, using the placeholder notation *${<parameter_name>}*. During the deployment, the parameter value is determined and the placeholder is replaced with the actual value.

Here is a list of commonly used parameters:

Parameter	Scope	Read-Only	Default Value	Example
app-name	modules	Yes	Application name in XS advanced or Cloud Foundry environment, based on the module name (with or without a name-space prefix)	ui5_user_list.user_js
default-url	modules	Yes	The modules default URL, composed as ${protocol}://${default-uri} under which it is reachable	${protocol}://${default-uri}
disk-quota	modules		-1, or as specified in module-type. Note Requires a unit of measurement (M, MB, G, or GB) in upper or lower case.	disk-quota: 1G
memory	modules		256M, or as specified in module-type	memory: 128M
service-name	resources	Yes	The name of the service to be created for this resource in the XS advanced or Cloud Foundry environment, based on the resource name (with or without a name-space prefix)	ui5_user_list.hdi_user_db

Figure 43: mta.yaml – Parameter Examples

In the figure above you can find some of the available parameters. You can find the full list in the *MTA Deployment Descriptor Syntax* section of the *SAP HANA Developer Guide*: https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/4050fee4c469498ebc31b10f2ae15ff2.html

MTA Editor

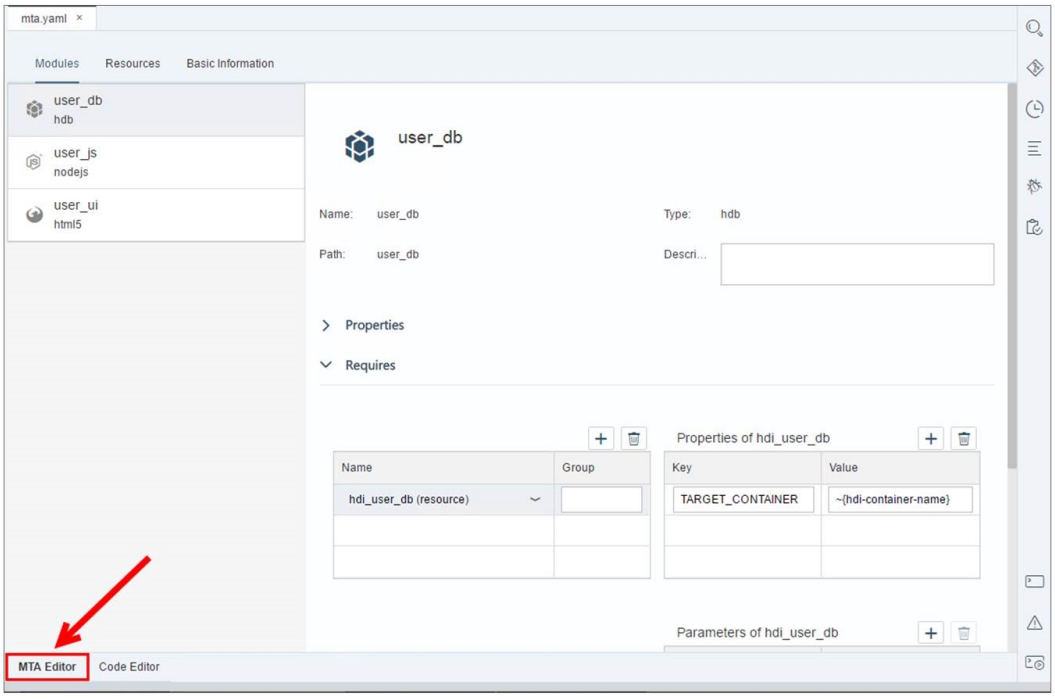


Figure 44: MTA Editor

As an alternative to the code editor for the MTA file, the SAP Web IDE for SAP HANA also provides the possibility to use the MTA editor to make changes. You can find the form-based editor on the bottom of the screen when editing an mta.yaml file.

LESSON SUMMARY

You should now be able to:

- Describe the information contained in the MTA Development Descriptor mta.yaml file

Introducing the Node.js Module

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe introductory concepts required to use the Node.js module in the SAP Web IDE for SAP HANA

Introducing the Node.js Runtime

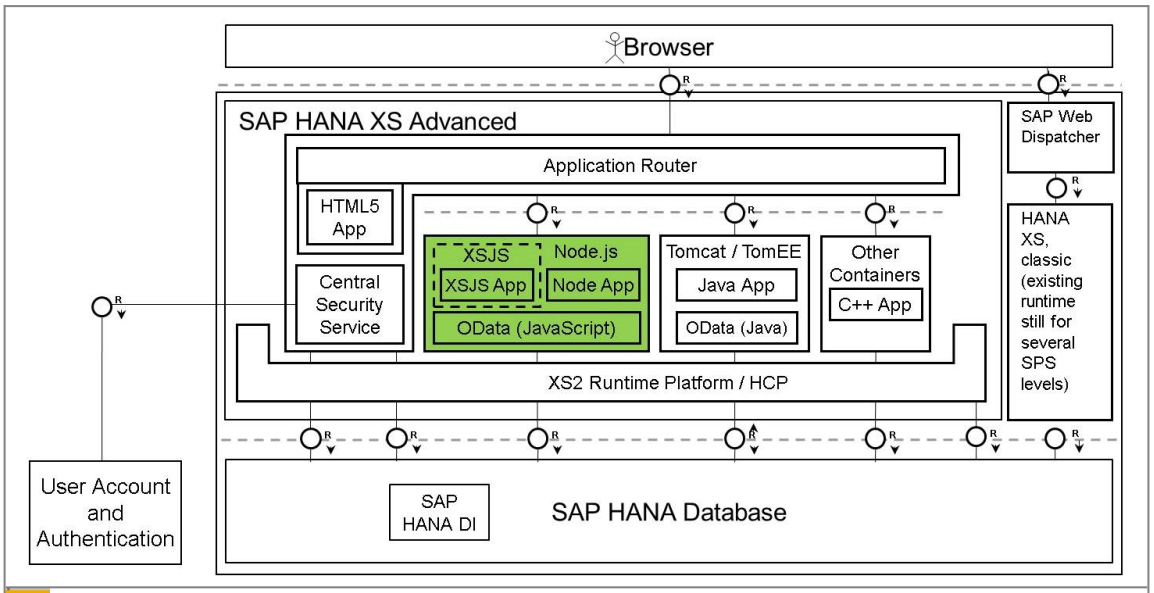


Figure 45: Architecture Overview: Node.js Runtime

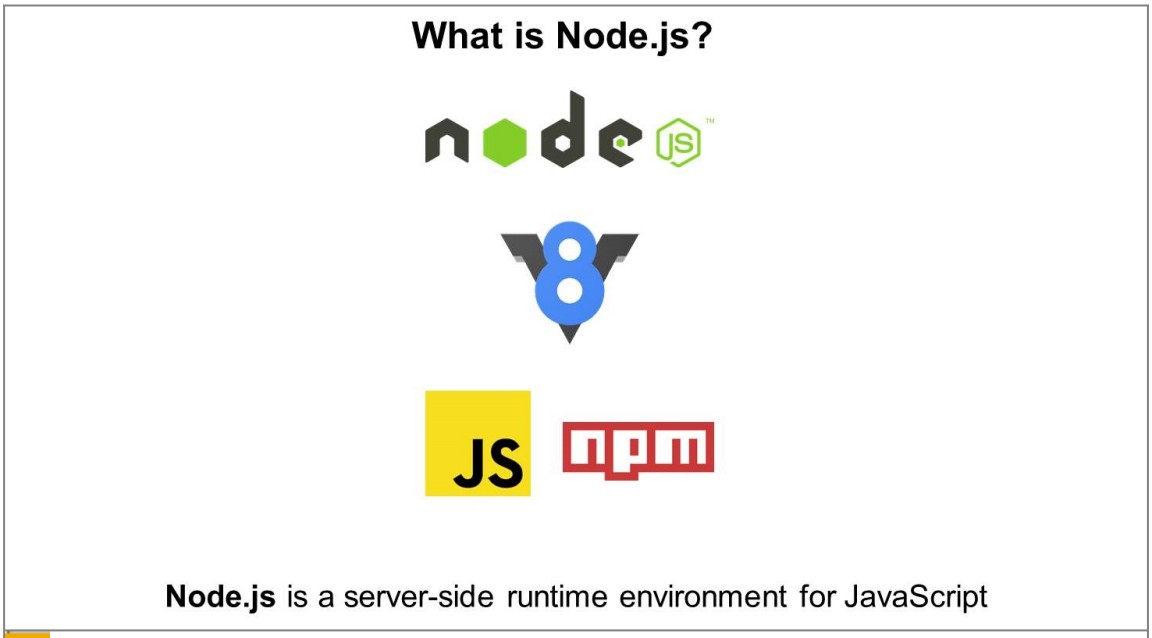


Figure 46: Node.js Characteristics

What is Node.js?

The following list describes Node.js:

- Node.js is a server-side runtime environment for JavaScript.
- Node.js is built on Google's V8 JavaScript engine.
- Node.js uses an asynchronous, event-driven programming model
- Node.js is single threaded and highly scalable
- Node.js is best suited for web applications
- Node.js is open-source with a huge community
- Node.js is has many available libraries (Node Modules) via NPM library

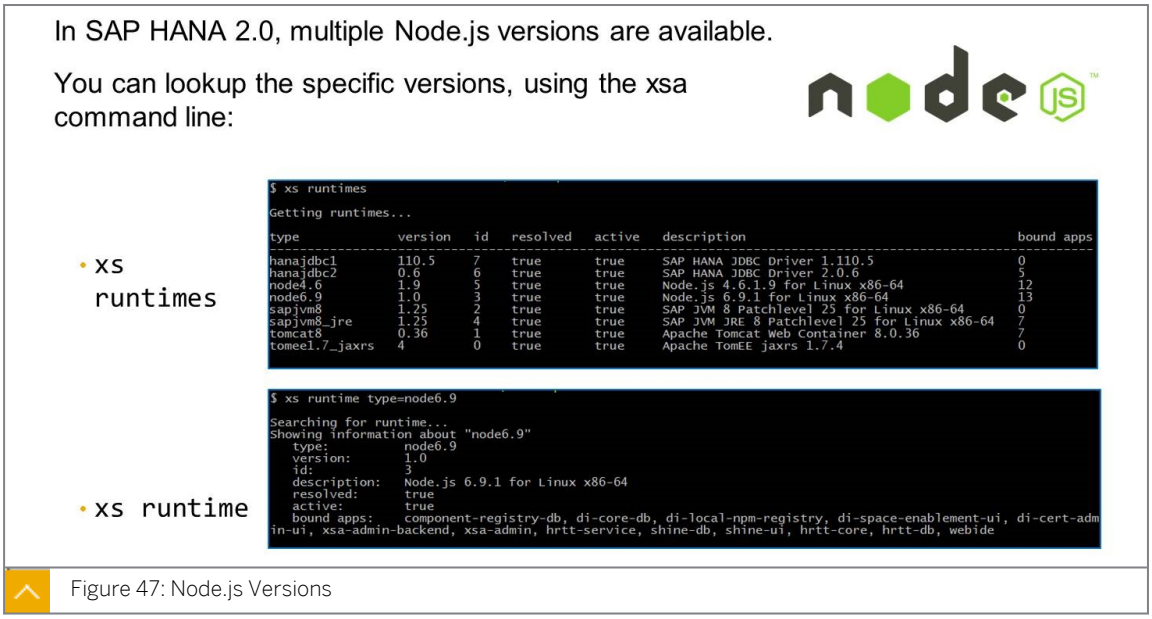


Figure 47: Node.js Versions

As of SAP HANA 2.0 SP01, XS Advanced contains multiple Node.js runtimes.

The version of all runtimes can be looked up, using the following XS Advanced client command: `xs runtimes`

Synchronous versus Asynchronous Processing

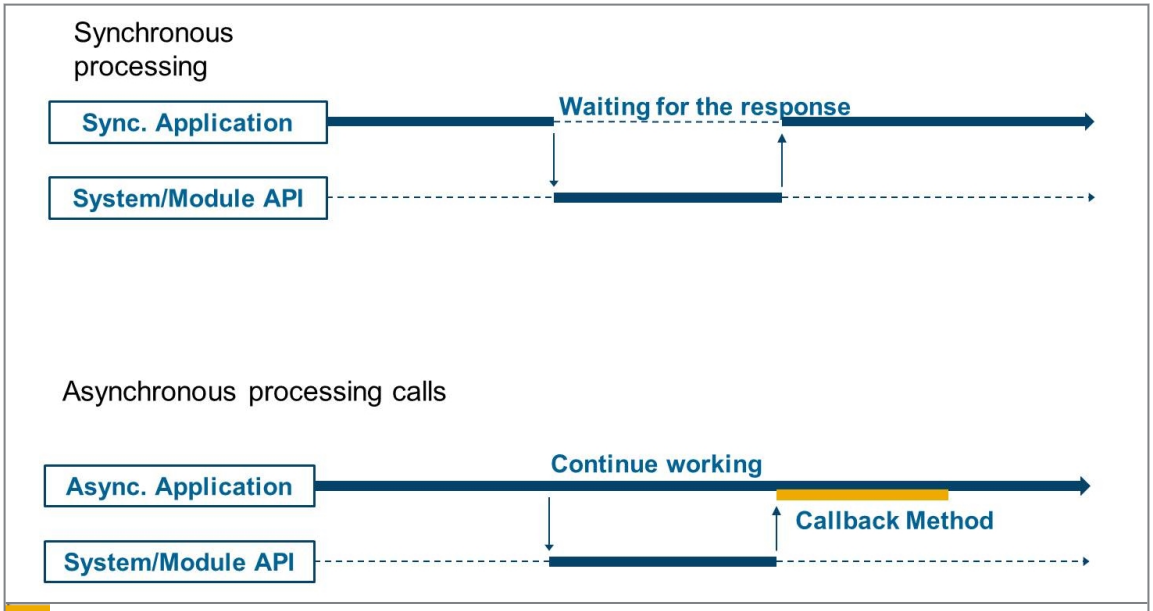


Figure 48: Synchronous versus Asynchronous Processing

Synchronous processing stops the execution until a response is retrieved from a called API. After the response is retrieved, the program continues.

Asynchronous processing calls an API but doesn't wait for the response. The response is processed later by the calling program using a callback method.

Node.js Event Loop

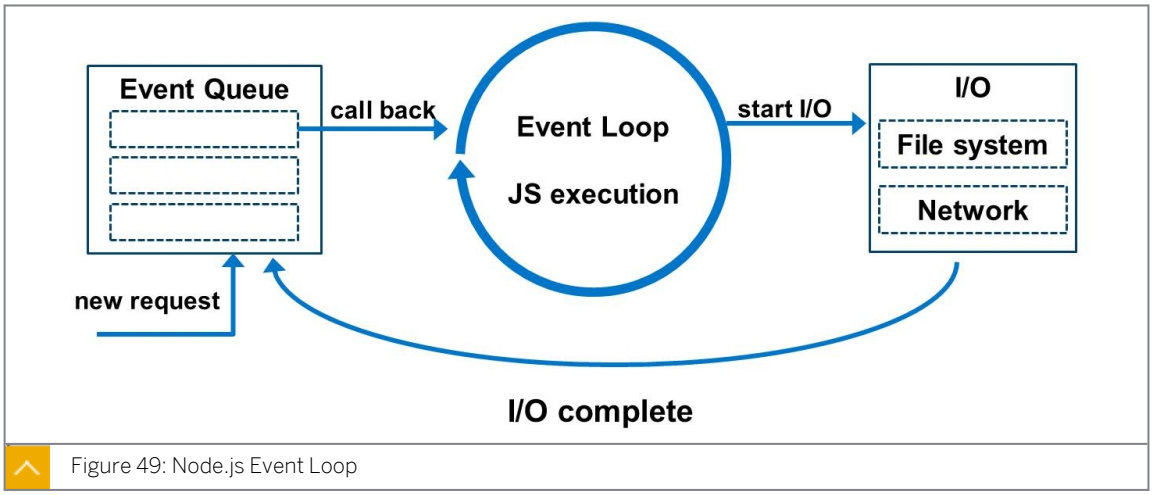


Figure 49: Node.js Event Loop

Node.js uses an asynchronous and non-blocking programming model.

The event loop executes tasks from the event queue and starts the callbacks.

If the queue is empty, the event loop process stops and gives back system resource.

Figure 50: Synchronous vs. Asynchronous Processing

Node modules run in an asynchronous mode by default. If you need synchronous processing, you can use the XSJS layer or a Node library.

Creating a Node.js Module in the SAP Web IDE for SAP HANA

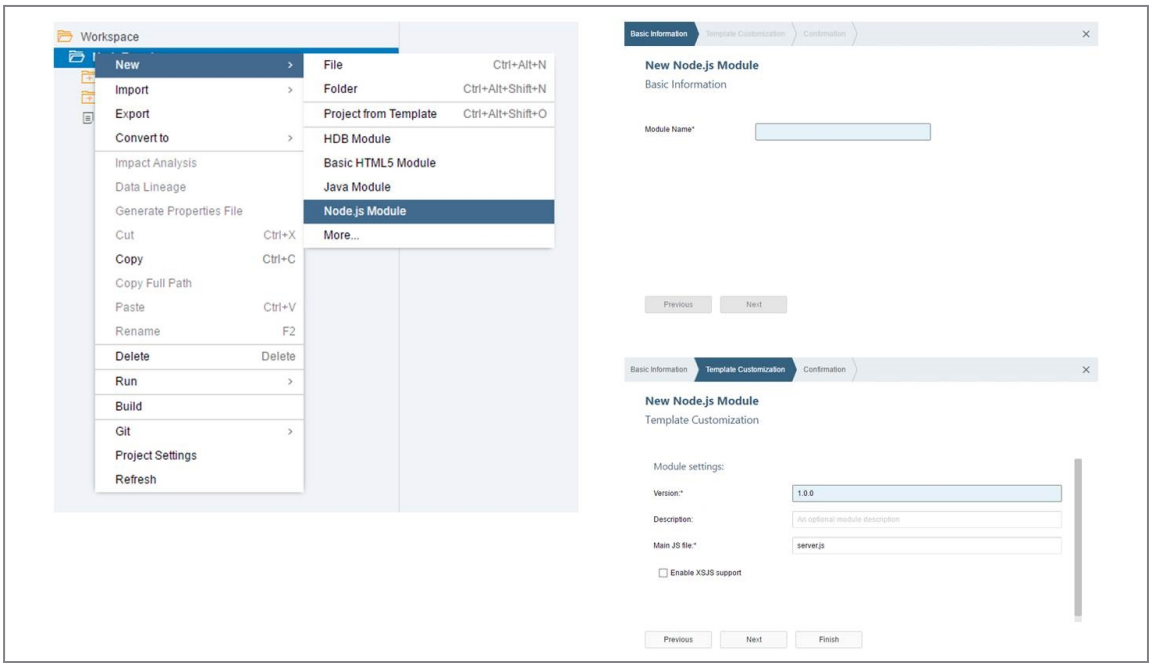


Figure 51: Create a Node.js (JavaScript) Module

The SAP Web IDE for SAP HANA offers a wizard to create a new Node.js module. During the setup you can enable XSJS support to use the XSJS compatibility layer.

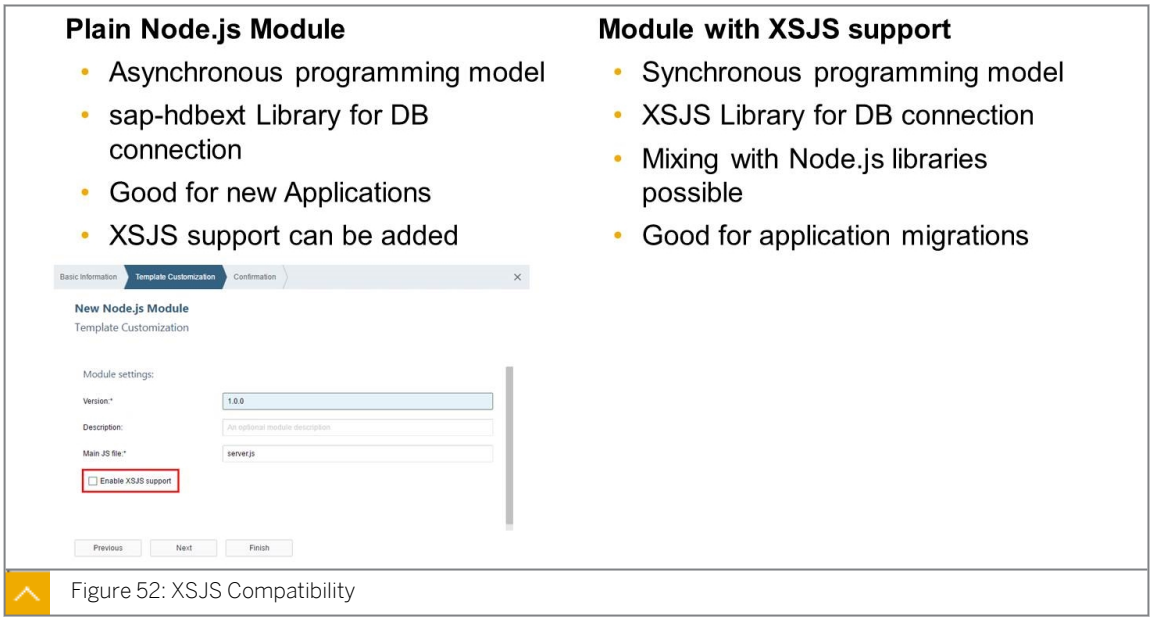


Figure 52: XSJS Compatibility

During the creation of the Node.js module, you can decide to enable the XSJS support. In this case, some special files will already be set up. However, the XSJS support can be added later if necessary.

While the recommendation for new applications is to start with a plain Node.js module, you can decide to include XSJS files if you need synchronous processing or if you already have libraries you want to reuse.

Describing the Application Package Descriptor File package.json

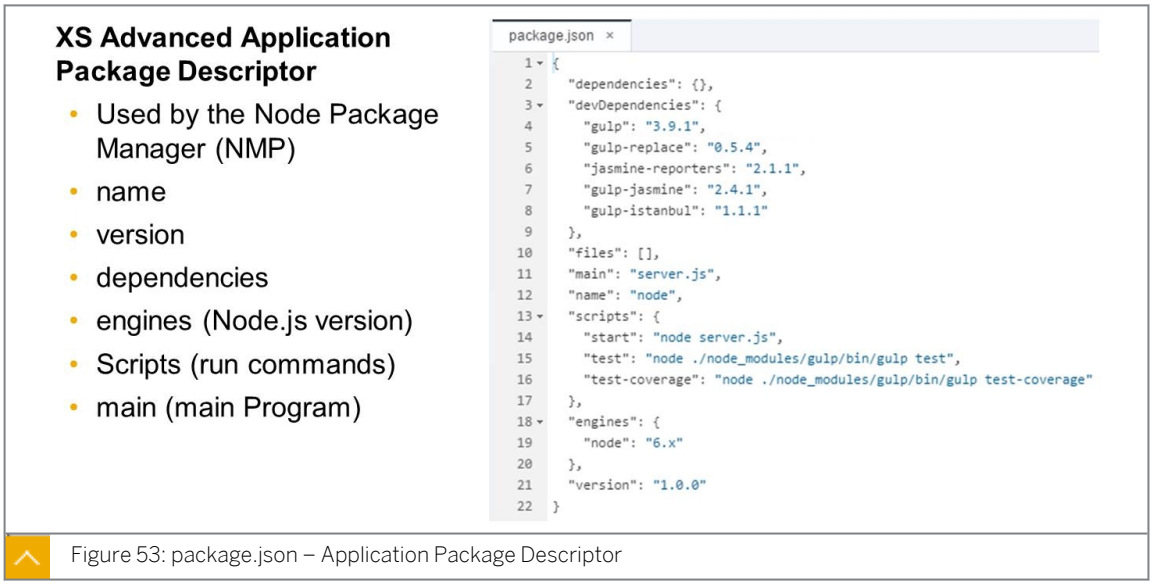


Figure 53: package.json – Application Package Descriptor

The build, deployment, and runtime dependencies of a JavaScript application are described in the package.json file. The package.json file is mandatory for JavaScript applications and it must be located in the JavaScript application's source-file folder.

As well as the application name and version, dependencies to other Node.js modules, the Node.js version, run scripts, and the main program are configured.

The scripts section contains the different run commands, which are executed for different tasks.

Application Package Descriptors

*package.json

```
{
    "name": "<application name>",
    "description": "<application description>",
    "private": true,
    "version": "1.3.1",
    "repository": {
        "url": "git@github.acme.com:xs-samples/nodejs-hello-world.git"
    },
    "dependencies": {
        "sap-xsjs": "^1.0.5",
        "approuter": "^2.3.1"
    },
    "engines": {
        "node": ">=0.10.x <0.12"
    },
    "scripts": {
        "start": "node --max-old-space-size=400 --expose-gc main.js"
    }
}
```

Figure 54: package.json – Application Package Descriptors

The application package descriptors are as follows:

- name

  The name of the JavaScript application whose package prerequisites and dependencies are defined in this package description.

- description

  A short description of the JavaScript application, whose package prerequisites and dependencies are defined in this package description.

- private

  Use the private property to indicate if access to the package specified in name is restricted. Private packages are not published by npm.

- version

  The version of the JavaScript application, whose package prerequisites and dependencies are defined in this package description.

- repository

  The absolute path to the repository used by the JavaScript application, whose package prerequisites and dependencies are defined in the package description.

- dependencies

  A list of dependencies that apply to the JavaScript application, whose package prerequisites and dependencies are defined in the package description.

- engines

  The runtime engines used by the application specified in the name property.

- scripts

  The script containing the command used to start the JavaScript application, along with any additional (required) options and parameters.

Semantic Versioning

```
"version": "<Major_Version>.<Minor_Version>.<Patch_Version>"
```

Version	
<1.2.3	Less than
<=1.2.3	Less than or equal to
>1.2.3	Grater than
>=1.2.3	Grater than or equal to
=1.2.3 / 1.2.3	Exact version. DEFAULT if nothingspecified
1.X / 1.x / 1.*	Any minor version of 1 (e.g. 1.2.3 or 1.8)
>1.2.3 <=2.3.4	RANGE: Greater than AND less than or equal to
1.2.3 || >=2.3.4 <3.0.0	Exactly 1.2.3 OR (grater than or equal to AND Less than)
~1.2.3	Allows patch level changes: >=1.2.3 <1.3.0
~1.2	Allows patch level changes; equivalent to 1.2.x
~1	Allows minor version changes: >=1.0.0 <2.0.0; equivalent to 1.x
^1.2.3	Allows versions with the same most left non zero digit: >=1.2.3 <2.0.0
^0.0.3	Allows >=0.0.3 <0.0.4

https://docs.npmjs.com/misc/semver

Figure 55: semver – Semantic Versioning

When defining dependencies and runtime engines in the package description, you can specify a range of versions. For example: >1.0.3, <=1.2.5, ^1.0.5 (compatible with version 1.0.5), or 1.2.x (any 1.2 version), or 1.1.0 - 1.3.12 (any version including or between the specified versions).

Start Commands

Figure 56: Start Commands

In the package.json file, start commands are included in the scripts section. Every command can be used in the SAP Web IDE for SAP HANA to run a certain part of the node.js application.

The figure above shows the start command which runs the file server.js, the test command which runs the gulp task test, and the test-coverage command which runs the gulp task test-coverage.

Describing Which Actions are Executed in XS Advanced When You Run an Application

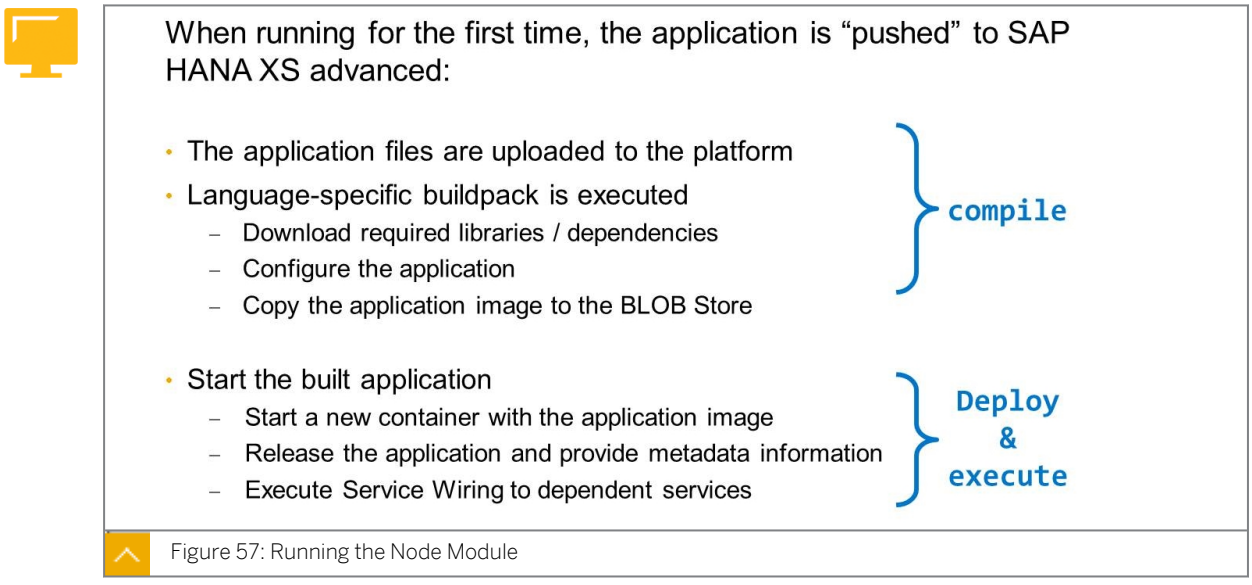


Figure 57: Running the Node Module

Applications are deployed to the target platform by using the push operation of the platform API. For this reason, in Cloud Foundry parlance, applications are "pushed" to the platform. Pushing an application works as follows:

1. The application files are uploaded to the platform.

2. Buildpacks are executed to create archives, that create the self-contained and ready-to-run executable applications (downloading any required libraries and other dependencies, configuring the application). Different buildpacks exist for the different target runtime environments, such as Java or JavaScript/ Node.js.

3. Applications are started as separate processes. At runtime, the applications need the connection information for the service instances to which they are bound. The applications obtain this information from process-specific environment variables, which are resolved by the platform in a process known as service wiring.

   Bindings can only be created between applications and service instances in the same space.

Running a Node.js Module in the SAP Web IDE for SAP HANA

Running the Node Module

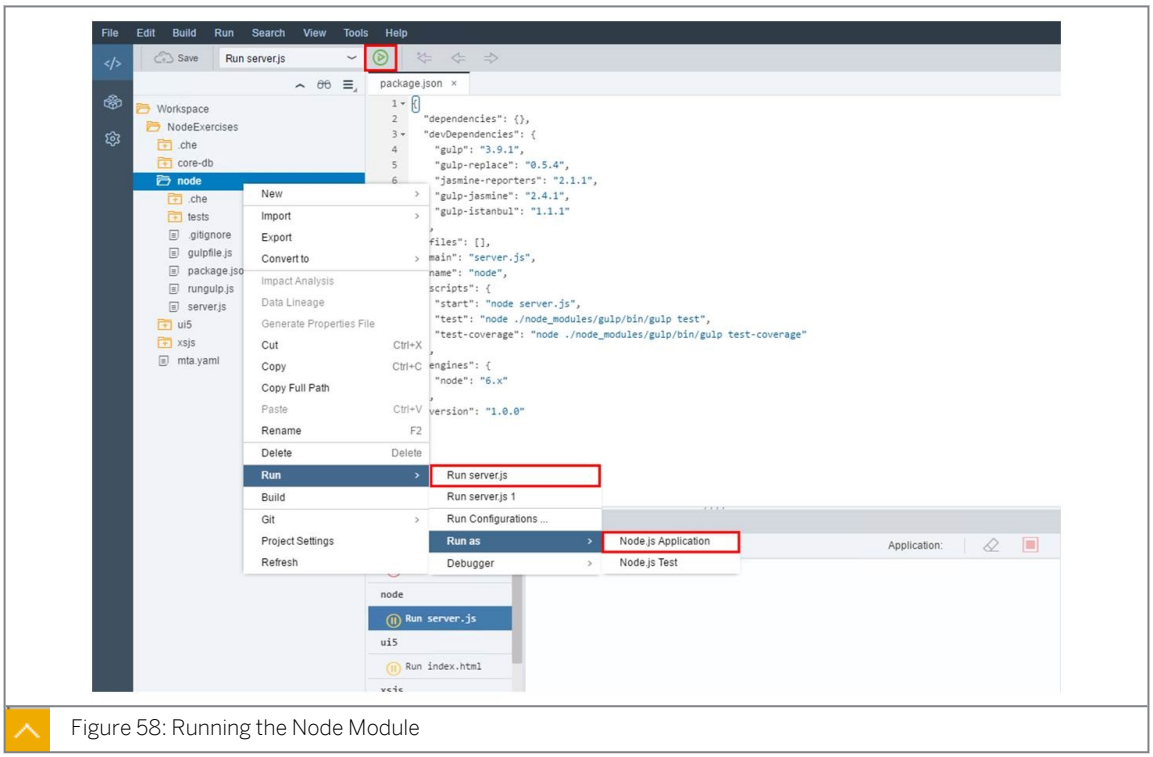


Figure 58: Running the Node Module

The node module can be run from the context menu of the module, from the window menu, or using the green run button in the toolbar. The different run configurations can be selected from the toolbar.

The build process completes automatically when starting the module. It is not necessary to execute it separately.

Run Console

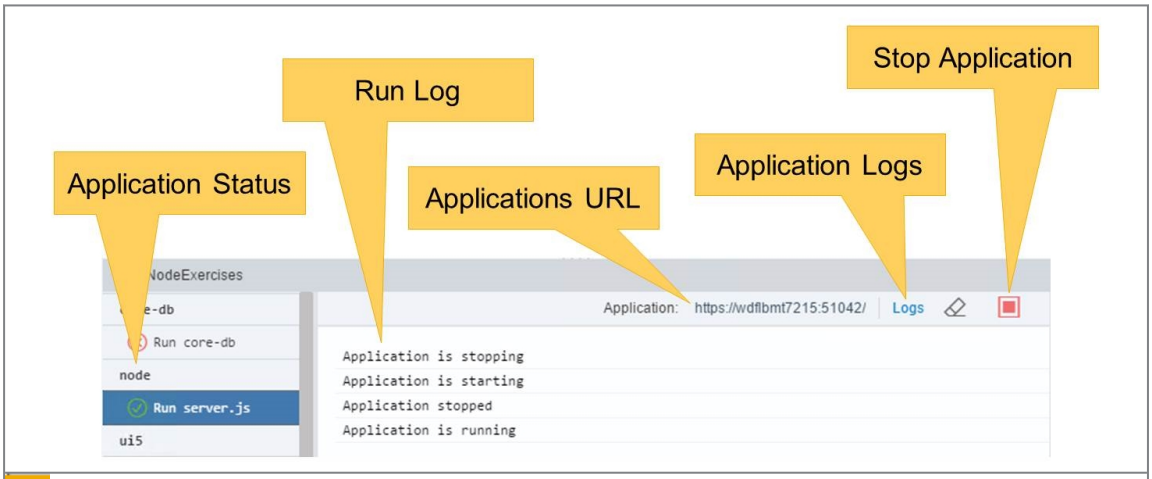


Figure 59: Run Console

The *Run* console in the SAP Web IDE for SAP HANA shows information about the application. It allows you to stop the application and allows navigation to the started app.

Application Status

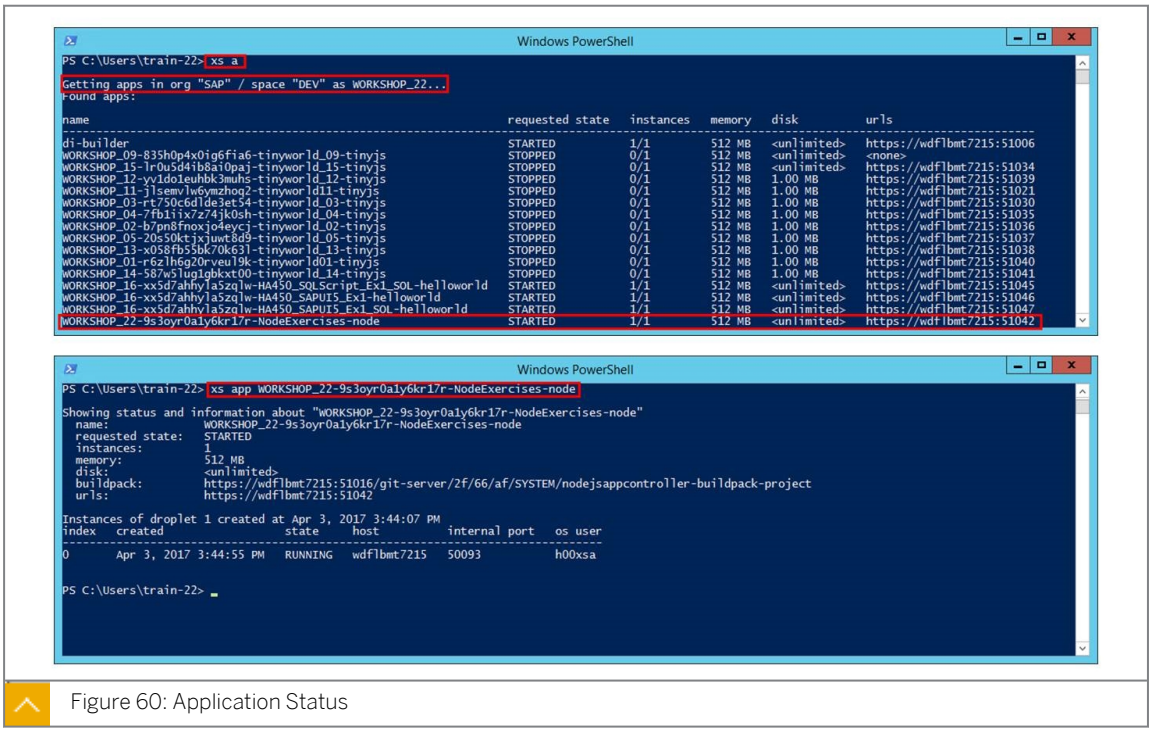


Figure 60: Application Status

Using the XS Advanced command line interface, the application status can be shown.

Using the Node.js Run Configurations in the SAP Web IDE for SAP HANA

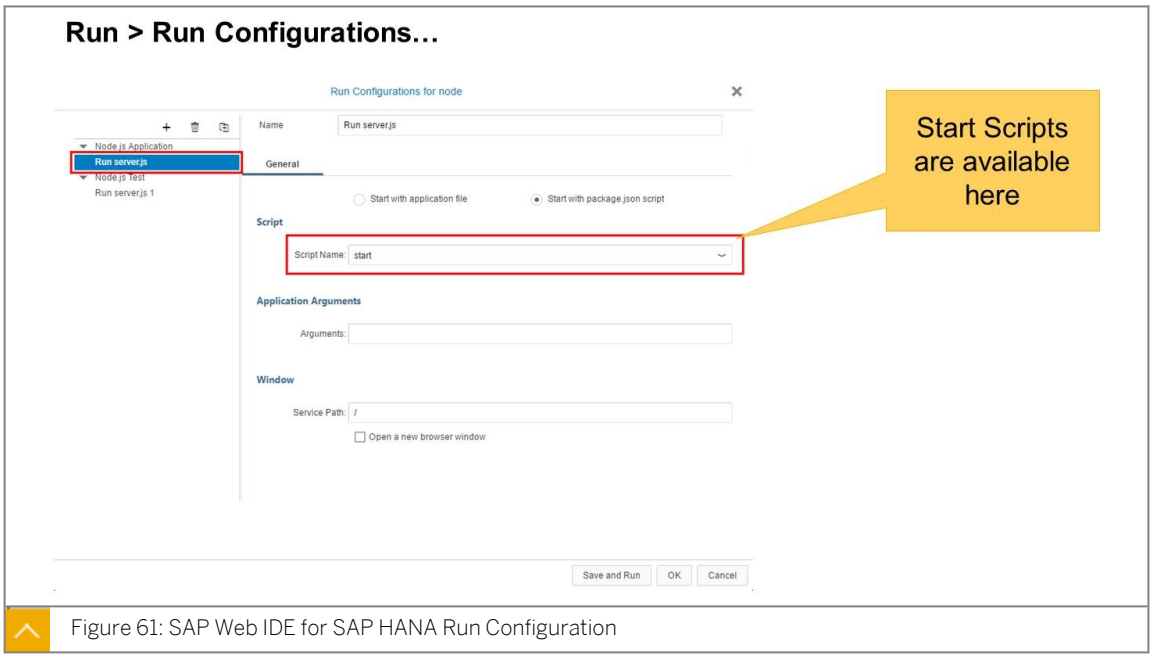


Figure 61: SAP Web IDE for SAP HANA Run Configuration

In the SAP Web IDE for SAP HANA run configurations, the start scripts are available for you to execute. The different run configurations can be used in the SAP Web IDE for SAP HANA to start the node application.

LESSON SUMMARY

You should now be able to:

- Describe introductory concepts required to use the Node.js module in the SAP Web IDE for SAP HANA

Creating and Deploying a Basic Node.js Module

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Create, run, export, and deploy a Node.js module saying Hello World

Creating the Node.js Module

Node.js module
- Basic files
  - package.json
  - server.js

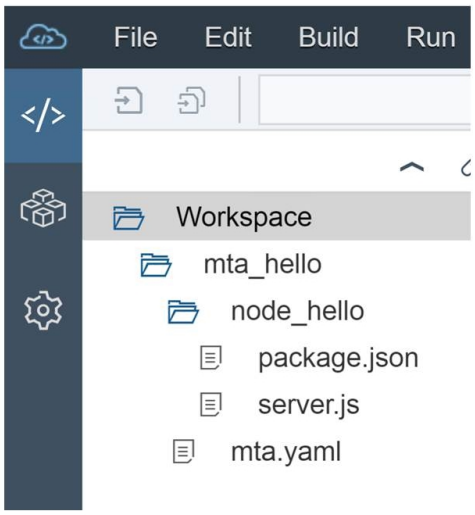


Figure 62: Basic Node.js Module

A minimal Node.js module contains two files:

- package.json : The package descriptor file. It describes the module characteristics. It must exist in every Node.js module within the application project.
- server.js : The JavaScript program that starts the server. The default name is server.js, nevertheless it may change based on the name defined within the package.json file.

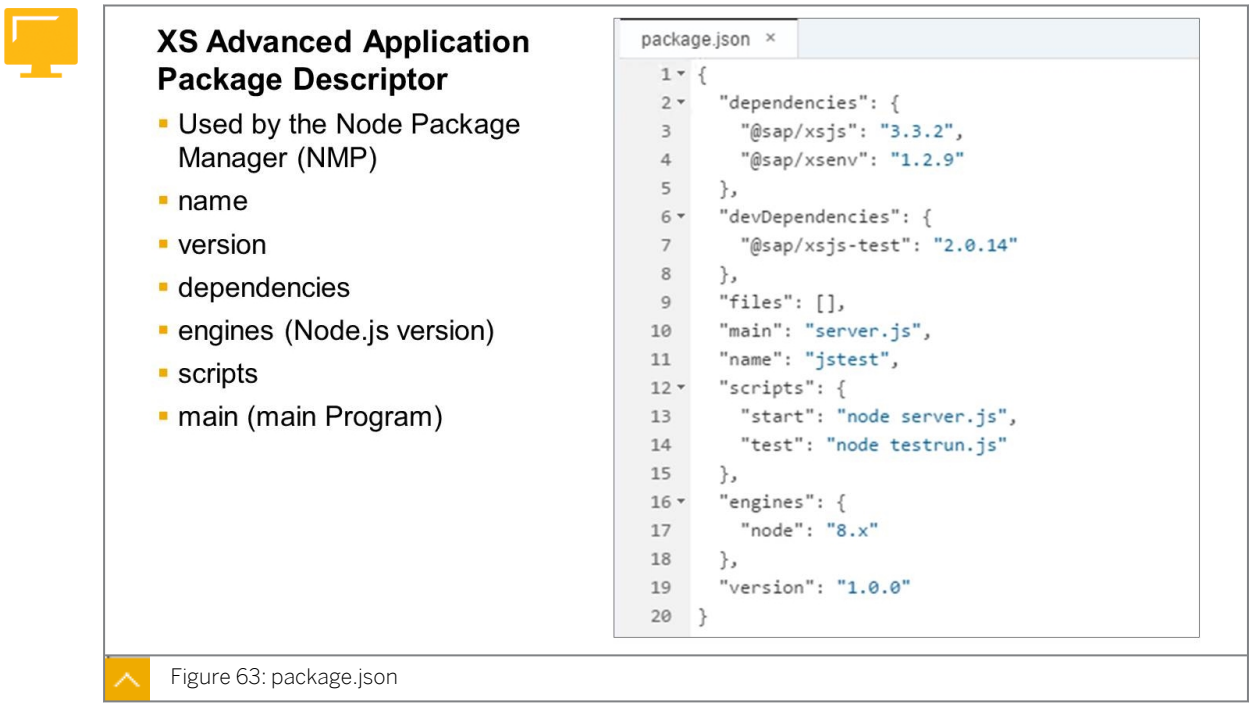


Figure 63: package.json

The build, deployment, and runtime dependencies of a JavaScript application are described in the package.json file. The package.json file is mandatory for JavaScript applications and it must be located in the JavaScript application's source-file folder.

As well as the application name and version, dependencies to other Node.js modules, the Node.js version, run scripts, and the main program are configured.

Using Express.js

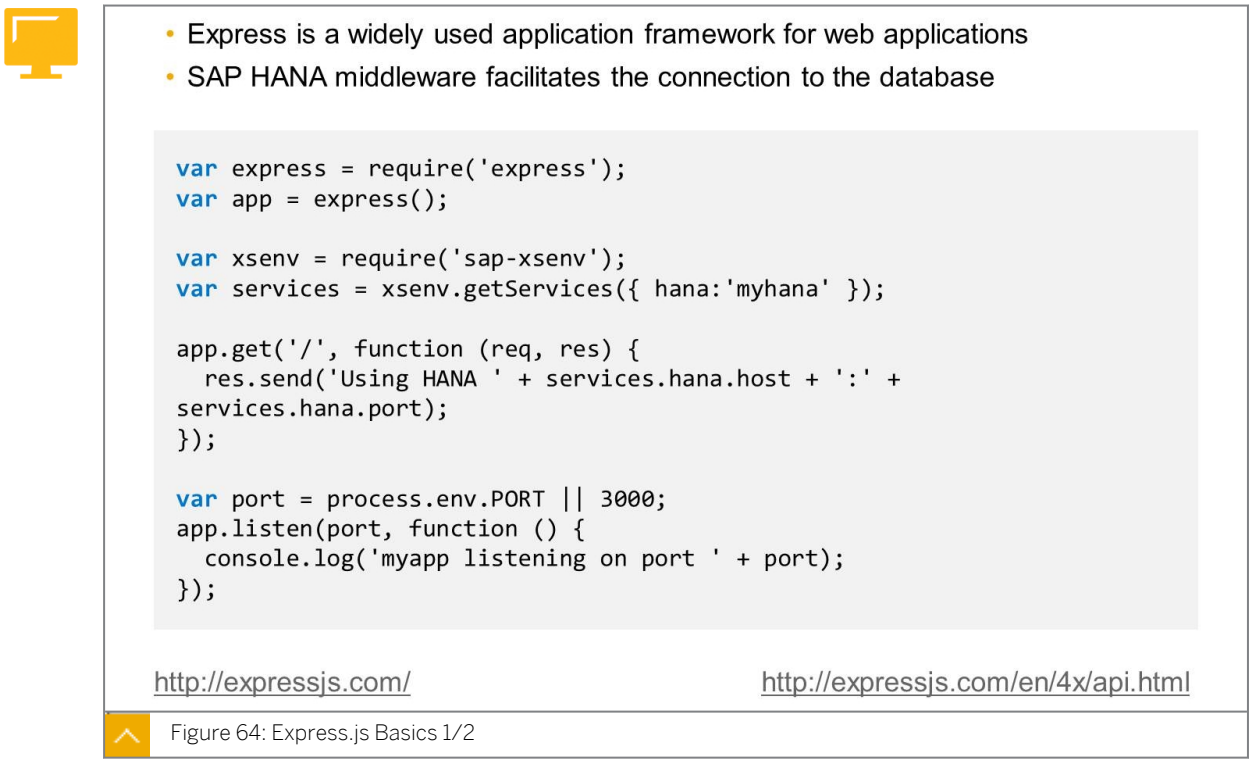


Figure 64: Express.js Basics 1/2

The express module is included into the program and an express application is created using the *express()* function.

The xsenv module is included, and the bound SAP HANA service instance is retrieved.

Using the*get()* method, a function is declared, which will be executed every time the root path of the application ("/") is accessed using a *GET* request.

Inside the function, the req (request) object contains passed payload from the client. The res (response) object is used to return information to the client.

Finally, the application is registered to an application port, on which it listens for the defined request methods.

HTTP Methods:

```
var app = express();
app.<METHOD>(path, callback
[, callback...])
```

METHOD stands for an HTTP method on which the Express framework should react. E.g.:

- delete
- get
- post
- put
- search

http://expressjs.com/en/4x/api.html

Callback Function:

```
app.get("/", function (req, res, next){
});
```

Request object examples:

- req
- req.body
- req.params
- req.param(<param name>)
- req.query

Response object examples:

- res.send(<body>): sends http response
- res.code(<status>): set http status
- res.json(<body>): sends json response
- res.set(<field>,<value>): set header field
- res.type(<type>): set response content type

If the callback doesn't end the request cycle, i.e. by sending back a resonse to the client, it has to call the next() function to pass processing on to the next registered function.

Figure 65: Express.js Basics 2/2

The complete API reference of the express framework can be reviewed at the following location: http://expressjs.com/en/4x/api.html

Creating the Hello World Node.js module

package.json

```
{
    "name": "node_hello",
    "version": "1.0.0",
    "dependencies": {
        "express": "^4.0.0"
    },
    "scripts": {
        "start": "node server.js"
    },
    "engines": {
        "node": "8.x"
    }
}
```

package.json is a standard Node.js file, commonly known as *package descriptor*.

Name and version of the Node.js module.

Dependencies from other Node.js modules.

Command to start the application.

Used Node.js engine version.

Figure 66: The Package Descriptor file for the Node.js application

package.json is a standard Node.js file, commonly known as the package descriptor

It is used to:

- List the packages your project depends on.
- Specify versions of a package that your project can use using semantic versioning rules.
- Make your build reproducible, and therefore easier to share with other developers.

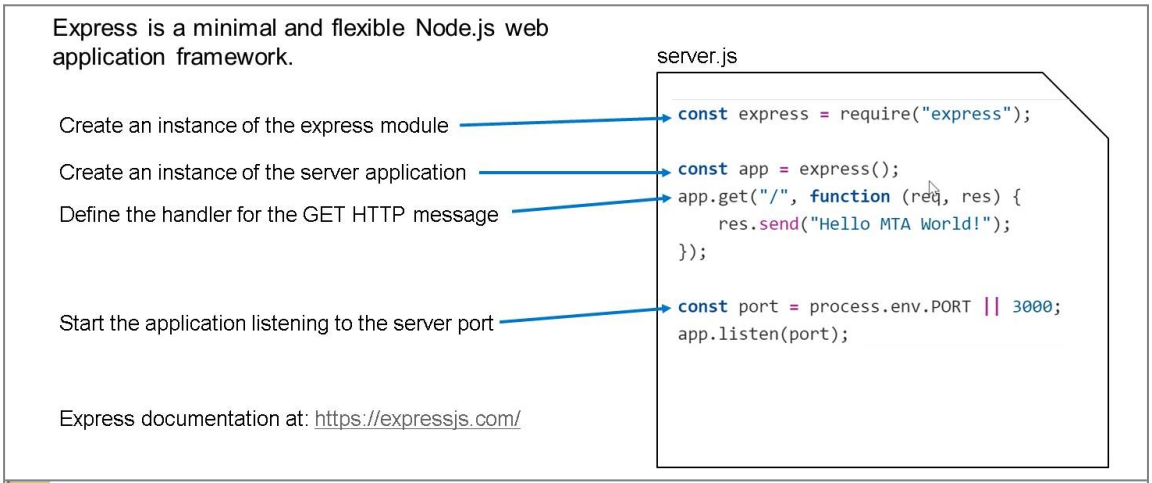


Figure 67: The server.js file for the Hello World application

Express is a minimal and flexible Node.js web application framework.

It's widely used and open-source.

The example shown in the figure uses Express to create a basic Hello World application.

LESSON SUMMARY

You should now be able to:

- Create, run, export, and deploy a Node.js module saying Hello World

Debugging the Node.js Code

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Debugging Node.js code using the SAP Web IDE debugger

Debugging Node.js code

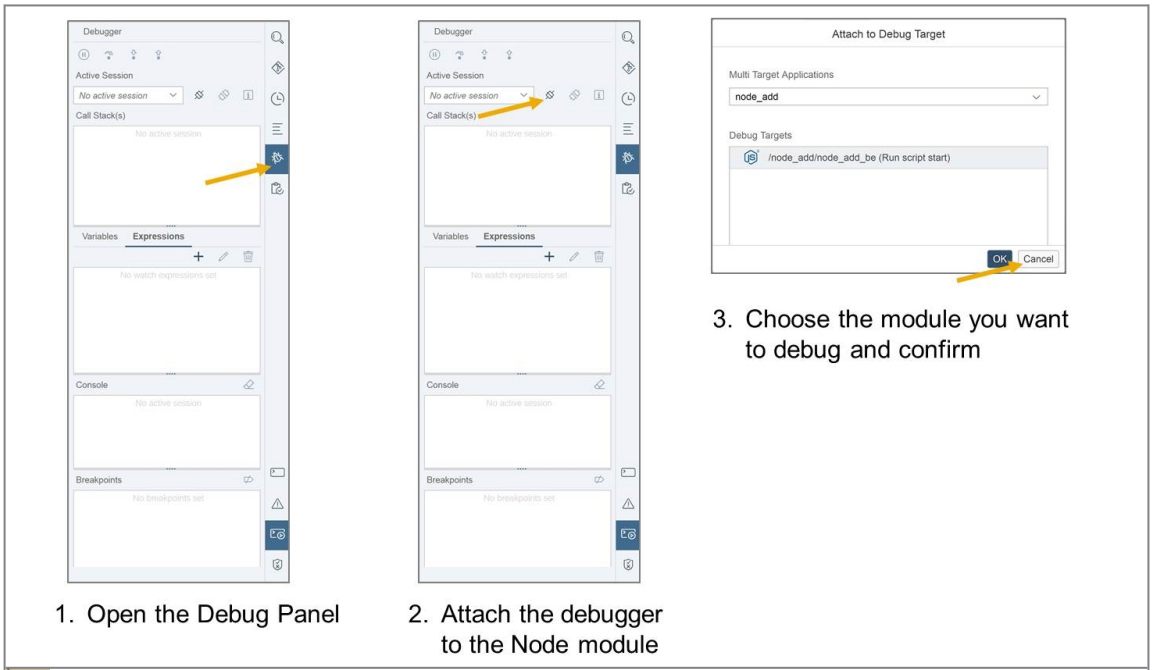


Figure 68: Enable the Debugger (1 of 2)

You can debug programs directly from the SAP Web IDE for SAP HANA. The debugger can be attached to a running Node.js module. After this is done, the debugger will be able to stop program execution at the breakpoints.

To attach the debugger:

1. Open the debug panel on the left side of the SAP Web IDE for SAP HANA.
2. Choose the *Connect* button at the top, and choose which node module to which you want to connect the debugger.

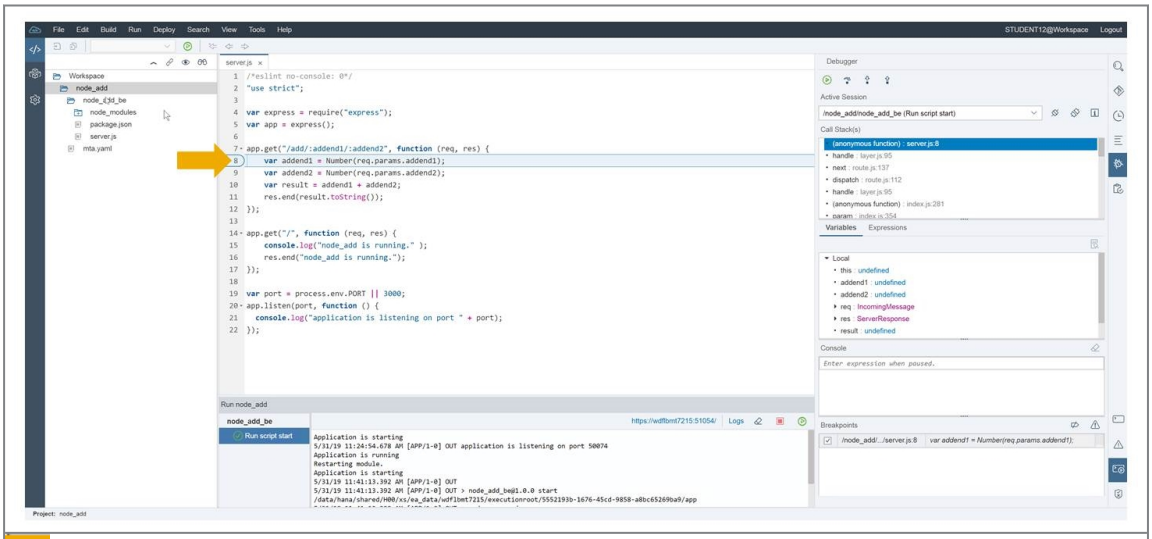


Figure 69: Enable the Debugger (2 of 2)

After you have successfully attached the debugger to the Node.js module, you can set breakpoints by selecting the line numbers to the left of the code.

If you execute your application using the applications link in the *Run* console, the debugger stops when it reaches the breakpoints that you set before.

Use the buttons on top of the debug panel to resume, step over, step in, step out, or deactivate all breakpoints. You can also detach the debugger from your Node.js module.

LESSON SUMMARY

You should now be able to:

- Debugging Node.js code using the SAP Web IDE debugger

Learning Assessment

1. What is the origin of the name “Multi-target Application”?

   Choose the correct answer.

   - ☐ **A** It is an application made to run on multiple front-end devices.
   - ☐ **B** It is an application made of multiple modules running in different runtime environments.
   - ☐ **C** It is an application targeting multiple user profiles.
   - ☐ **D** It is an application made of modules that can be reused across applications.

2. What is Node.js?

   Choose the correct answer.

   - ☐ **A** It is a scripting language specialized in the execution of data intensive calculations.
   - ☐ **B** It is a client-side JavaScript library, providing components to design the user interface.
   - ☐ **C** It is a server-side runtime environment for JavaScript.

3. In SAP HANA, the debugger needs to be attached to a Node.js module before it stops at any breakpoint.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

Learning Assessment - Answers

1. What is the origin of the name “Multi-target Application”?

   Choose the correct answer.

   - [ ] A It is an application made to run on multiple front-end devices.
   - [x] B It is an application made of multiple modules running in different runtime environments.
   - [ ] C It is an application targeting multiple user profiles.
   - [ ] D It is an application made of modules that can be reused across applications.

   That is correct! The name “Multi-target Application” comes from the fact that a multi-target application is made of multiple modules running in different runtime environments.

2. What is Node.js?

   Choose the correct answer.

   - [ ] A It is a scripting language specialized in the execution of data intensive calculations.
   - [ ] B It is a client-side JavaScript library, providing components to design the user interface.
   - [x] C It is a server-side runtime environment for JavaScript.

   That is correct! Node.js is a server-side runtime environment for JavaScript.

3. In SAP HANA, the debugger needs to be attached to a Node.js module before it stops at any breakpoint.

   Determine whether this statement is true or false.

   - [x] True
   - [ ] False

   This is correct! In SAP HANA, the debugger needs to be attached to a Node.js module before it stops at any breakpoint.

UNIT 3 Creating the Persistence Data Model Using Core Data Services

UNIT OBJECTIVES

- Describe the main features of the SAP HANA Database module
- Explain the basic concepts of Core Data Services
- Create a Core Data Services Entity, converted at runtime into a database table
- Use context, association, and view in Core Data Services

Introducing the SAP HANA Database Module

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Describe the main features of the SAP HANA Database module

Introducing the SAP HANA Database module

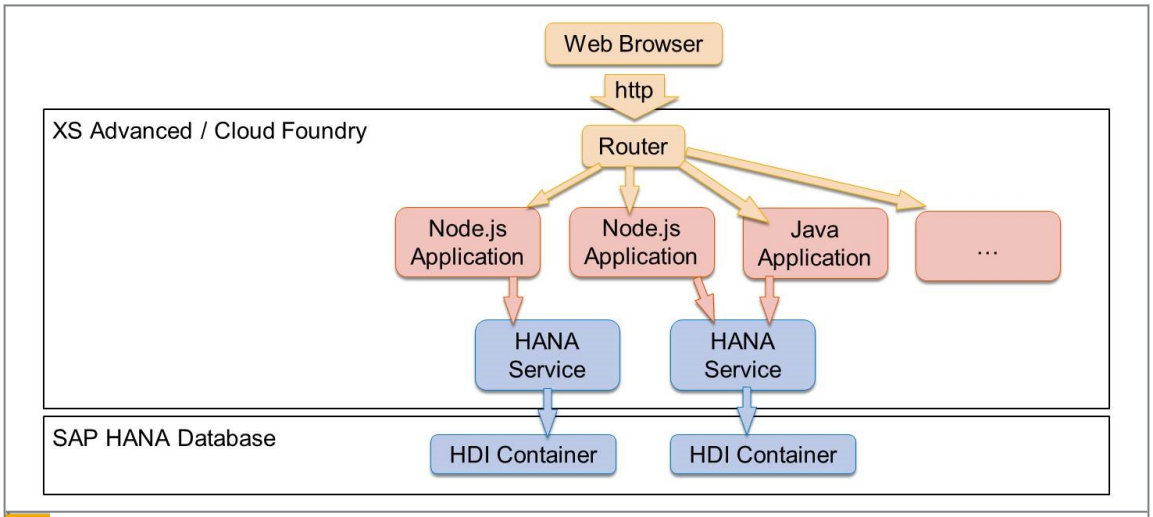


Figure 70: Runtime Structure of the Full-Stack Application

At runtime, the full-stack MTA is commonly made in the following way:

1. The user interface is written using HTML5 and rendered/executed in a web browser.
2. The HTML5 program communicates with the router via the HTTP protocol. The router is a program running on the server. It does not execute business logic, it is just a dispatcher of HTTP messages.
3. The router dispatches the HTTP message to the proper application, running on the server. This application is built out of the corresponding Node.js (Java, Python, and so on) module, coded by the developer within the MTA project in the SAP Web IDE for SAP HANA.
4. The Node.js application communicates with the database via a SAP HANA service, that is connected to a SAP HANA Deployment Infrastructure (HDI) container stored in the database.
5. Database objects (Tables, Views, Procedures, and so on) are stored in the HDI container.
6. The procedures in the database use the SQLScript (alternatively the R) language, that is specialized in highly parallel and data intensive processing.

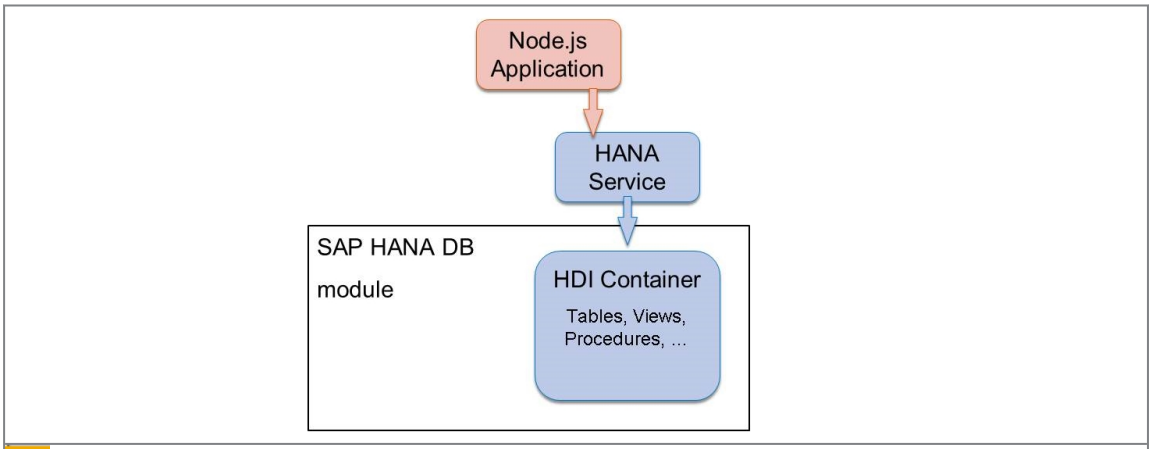


Figure 71: The SAP HANA Database Module

At design time, in the SAP Web IDE for SAP HANA, an MTA project is created.

The database content of the MTA is defined using the SAP HANA database module.

This module contains the design time definitions of all the objects to be created in the database, for example:

- Tables and Views
- Calculation Views
- Procedures

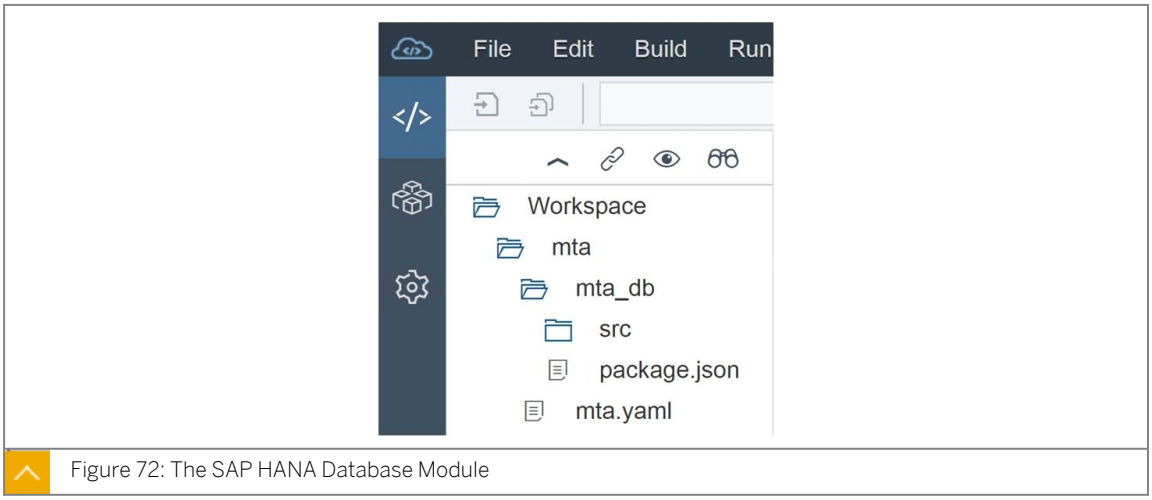


Figure 72: The SAP HANA Database Module

A minimal SAP HANA database module contains:

- An src folder, made to store the design time definitions of the database objects.
- A minimal package.json file for configuration, for example, to set the version and the options of the deploy program.

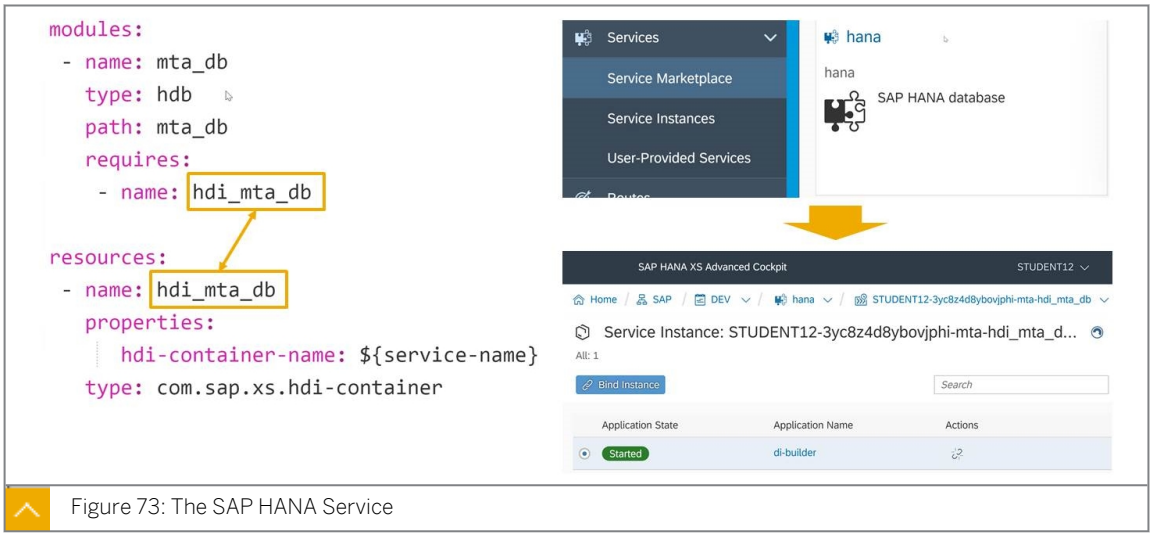


Figure 73: The SAP HANA Service

A service of type HANA (technical name com.sap.xs.hdi-container) needs to exist for the HDI container to be accessible from XS Advanced / Cloud Foundry. When you create a new SAP HANA database module, a service is also added by default to the mta.yaml file, together with the configuration required for the service to be accessible by the module.

When the module is built, a service instance (of type HANA) is created in XS Advanced. This service instance is connected to the HDI container that is created in the SAP HANA database.

Using the Hidden Configuration Files in the SAP HANA Database Module

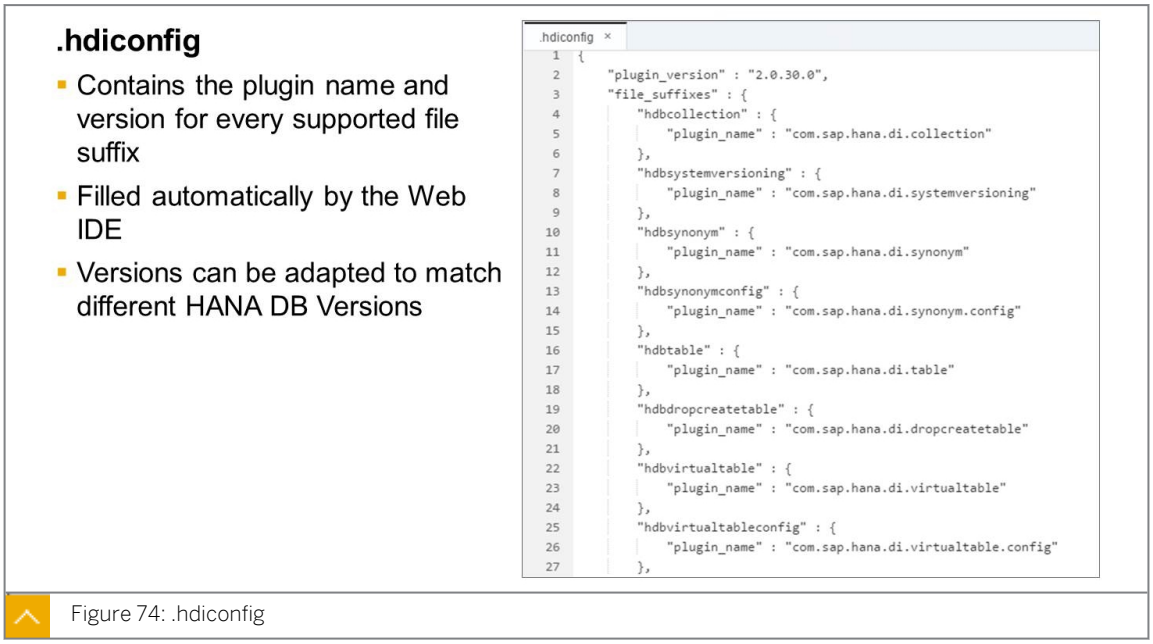


Figure 74: .hdiconfig

The database artifact .hdiconfig specifies the plug-ins and the version to use when deploying a particular type of database artifact to an HDI container. The artifact type is defined in the file suffix, for example, .hdbcds or .hdbsynonym. The plug-in definition ensures that the appropriate runtime object is created from the specified design time artifact.

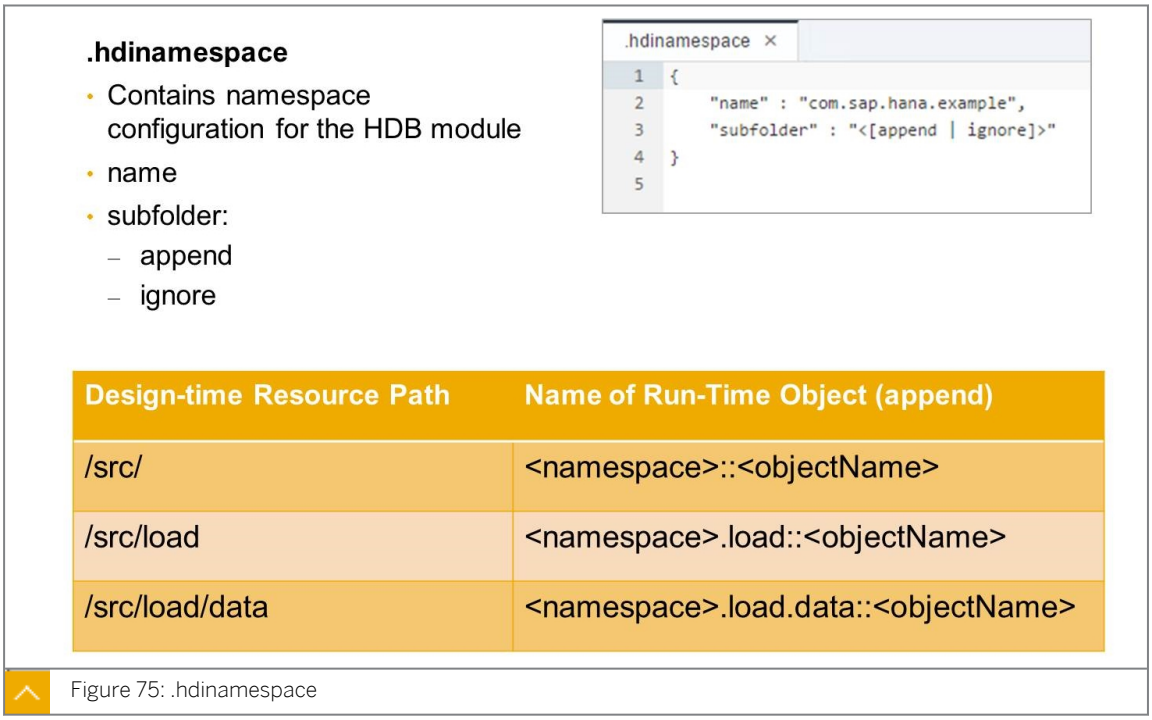


Figure 75: .hdinamespace

In SAP HANA HDI, name-space rules are defined in one or more file resources named .hdinamespace. The file resources must be located in the design time folder to which the naming rules apply, or the root folder of a hierarchy to which the naming rules apply. The content of the .hdinamespace file is specified according to the JSON format.

You can change the path defined in the name and, for the sub-folder, you can choose between append (add sub-folder name to object name space) and ignore (do not add sub-folder name to the object name space).

LESSON SUMMARY

You should now be able to:

- Describe the main features of the SAP HANA Database module

Introducing Core Data Services

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Explain the basic concepts of Core Data Services

Introducing Core Data Services

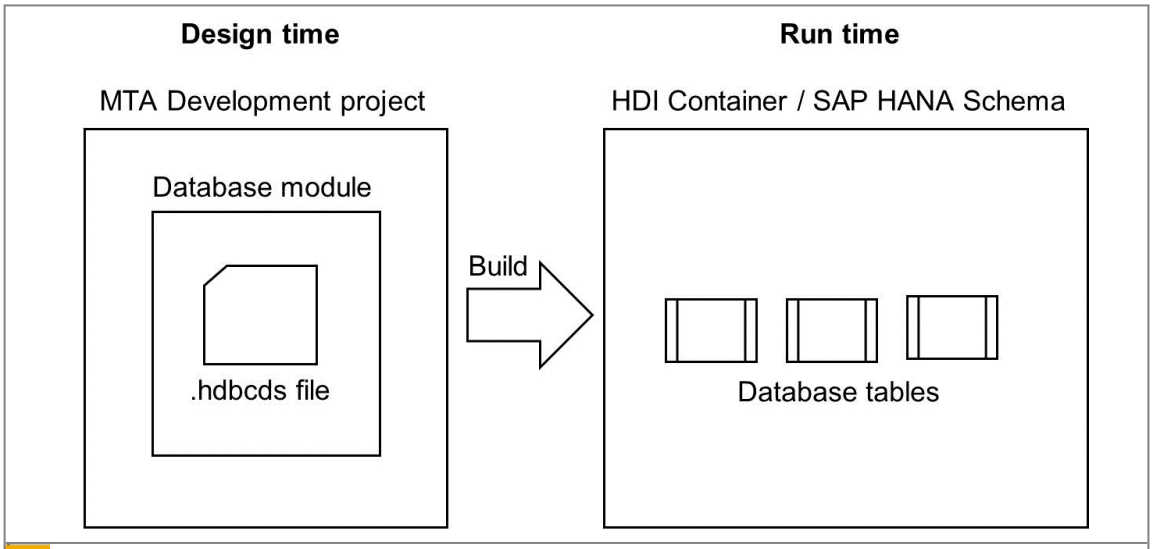


Figure 76: Core Data Services

Core Data Services (CDS) is a technique used to create the persistency layer (the database layer) of a software application.

During software development the data model is defined via a Data Definition Language within a file with extension .hdbcds.

When the application is built and then deployed, a set of database tables is generated in the database reproducing the data structure described in the file.

If the .hdbcds file is modified and the program is re-built, the created database objects are automatically changed to realize the new model.

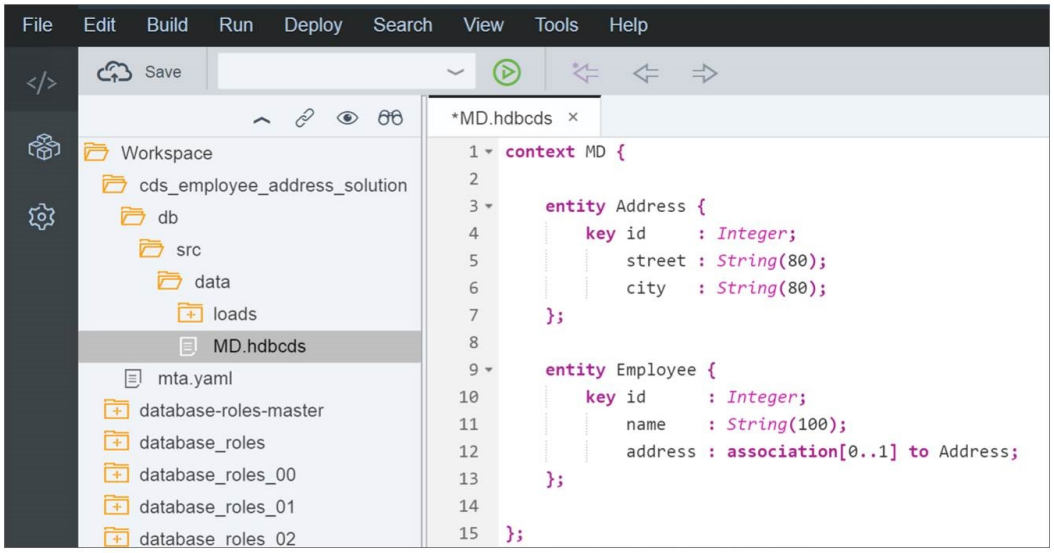


Figure 77: Example of .hdbcds file

The figure above is an example of a .hdbcds file, created in the SAP Web IDE for SAP HANA.

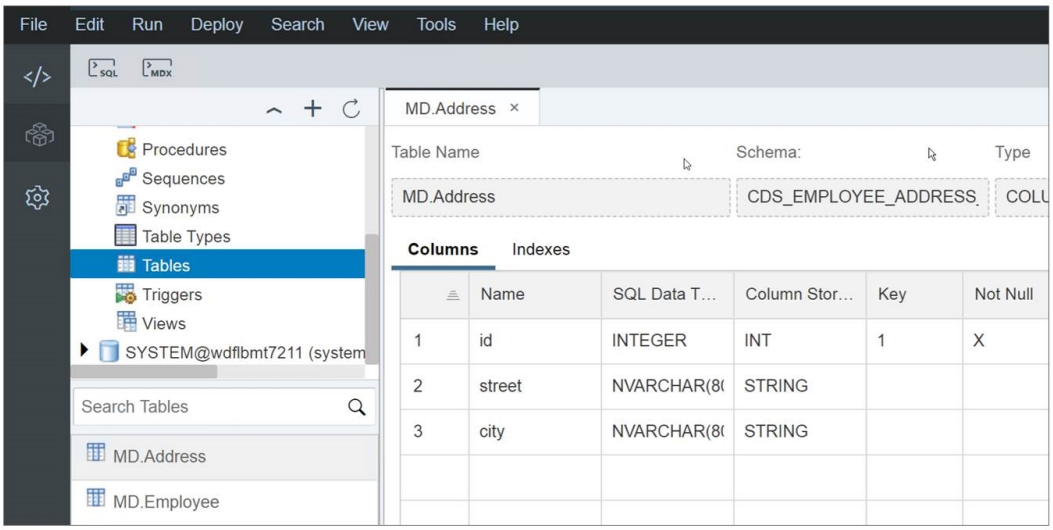


Figure 78: The Created Tables

The figure above shows the created database tables that you can display with the Database Explorer.

Graphical CDS Editor in the SAP Web IDE for SAP HANA

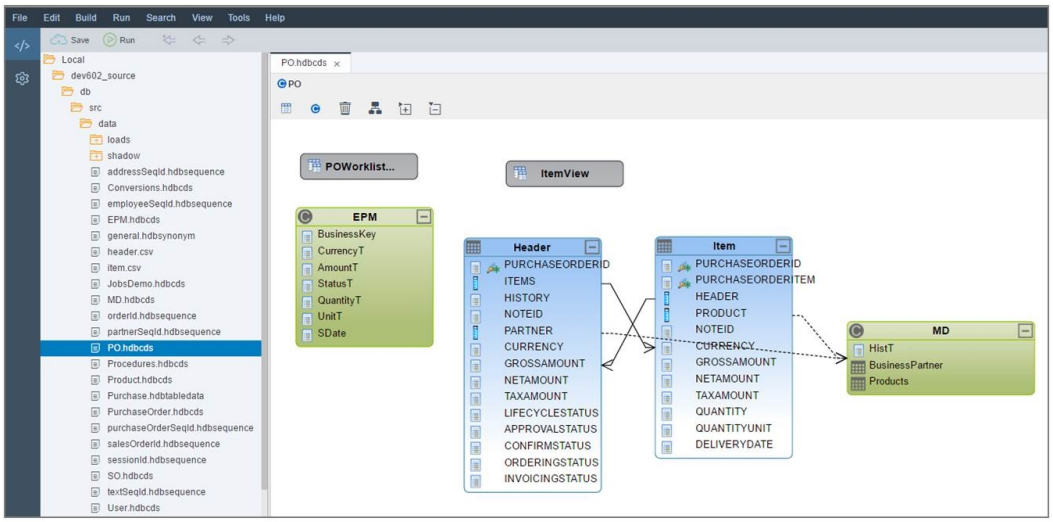


Figure 79: Graphical CDS Editor in the SAP Web IDE for SAP HANA

The SAP Web IDE for SAP HANA enables the creation and modification of CDS files, via a graphical editor. As well as specifying the entities, you can specify associations, views, contexts, and structures.

LESSON SUMMARY

You should now be able to:

- Explain the basic concepts of Core Data Services

Using the Core Data Services Entity

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Create a Core Data Services Entity, converted at runtime into a database table

Using the CDS Entity

```
entity Employee {
    key id   : Integer;
        name : String(100);
};
```

Table Name	Schema:	Type
Employee	MTA_HDI_DB_1	TABLE

Columns Indexes

	Name	SQL Data Type	Column Sto…	Key
1	id	INTEGER	INT	1
2	name	NVARCHAR(100)	STRING	

Figure 80: CDS Entity

The CDS Entity triggers (upon build) the generation of a database table with the corresponding name and structure.

LESSON SUMMARY
You should now be able to:

- Create a Core Data Services Entity, converted at runtime into a database table

Using the CDS Context, Association, and View

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use context, association, and view in Core Data Services

CDS Context

```
context HumanResources {
    context Recruitment {
        entity Employee {
            key id   : Integer;
                name : String(100);
        }
    }
};
```

Figure 81: CDS Context

The CDS context is a logical container that you can use to group multiple CDS objects (that is, CDS Entities) in the same .hdbcds file.

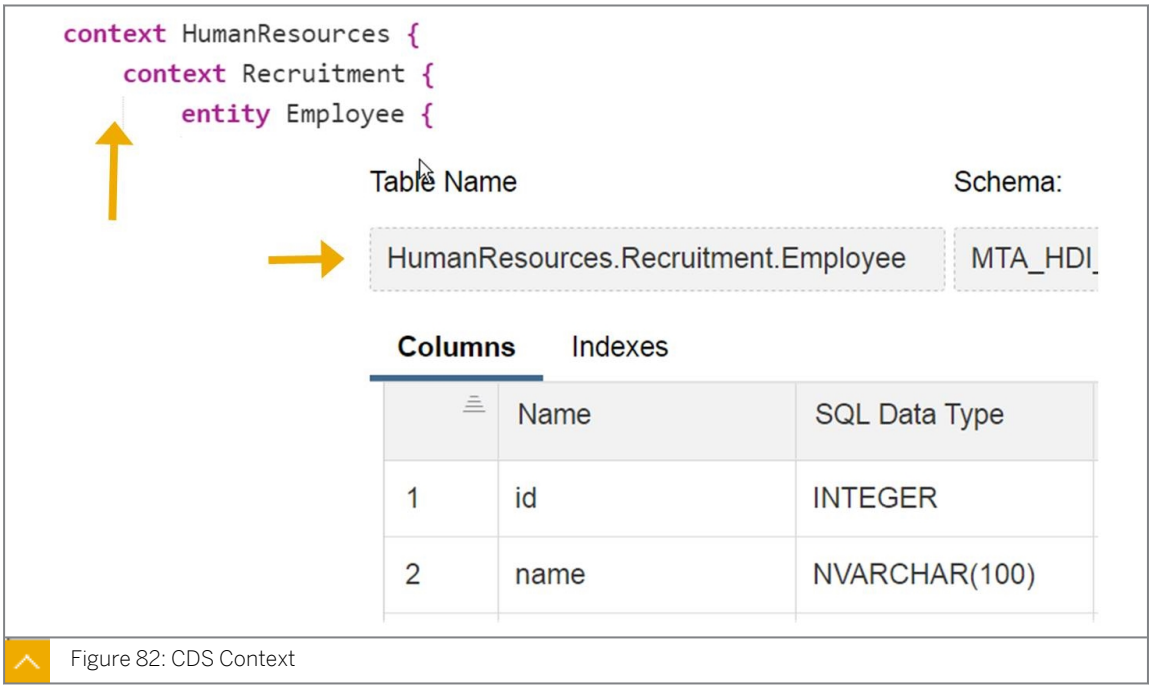


Figure 82: CDS Context

The container name is reflected in the name of the generated table. In the above figure, the container, *HumanResources*, contains a container, *Recruitment*, that contains an entity, *Employee*.

As a consequence, the table generated for the entity, *Employee*, in the database has the name *HumanResources.Recruitment.Employee*.

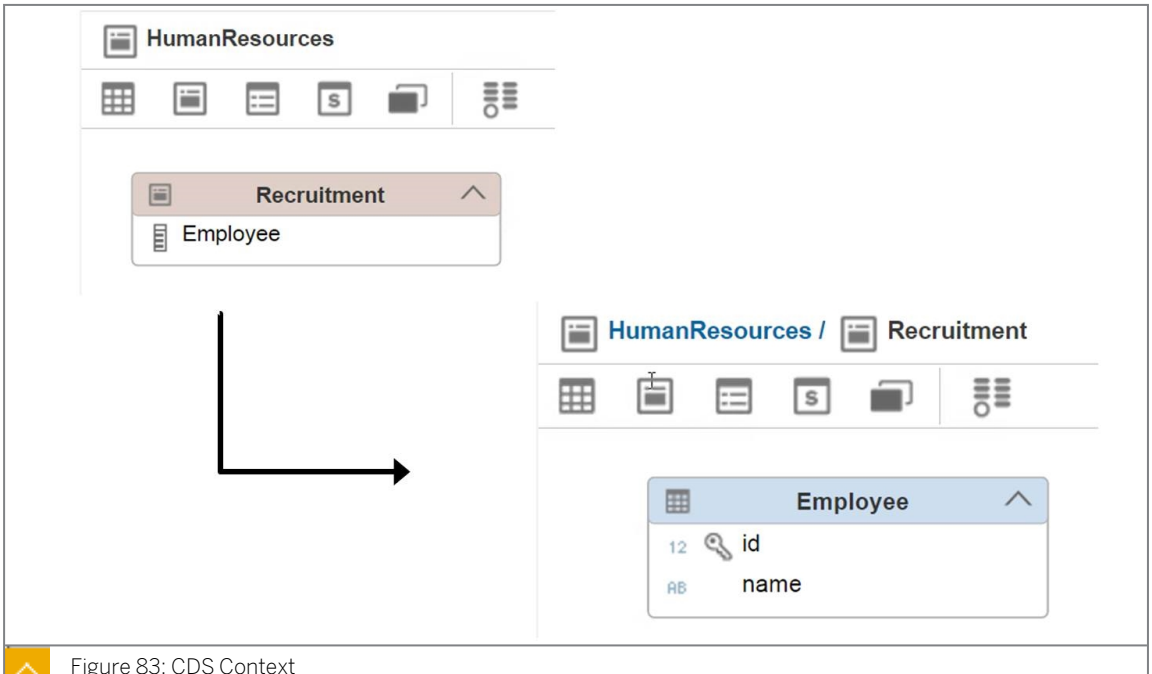


Figure 83: CDS Context

In the graphical CDS editor, you can navigate containers as a hierarchy of folders. The current container appears in the higher part of the screen.

CDS Association

```
context MD {
    entity Address {
        key id      : Integer;
            street  : String(80);
            city    : String(80);
    };
    entity Employee {
        key id      : Integer;
            name    : String(100);
            address : association to Address;
    };
}
```

Figure 84: CDS Association

Associations define relationships between entities.

Associations are specified by adding an element to a source entity that has an association type that points to a target entity and is complemented by optional information defining cardinality and which keys to use.

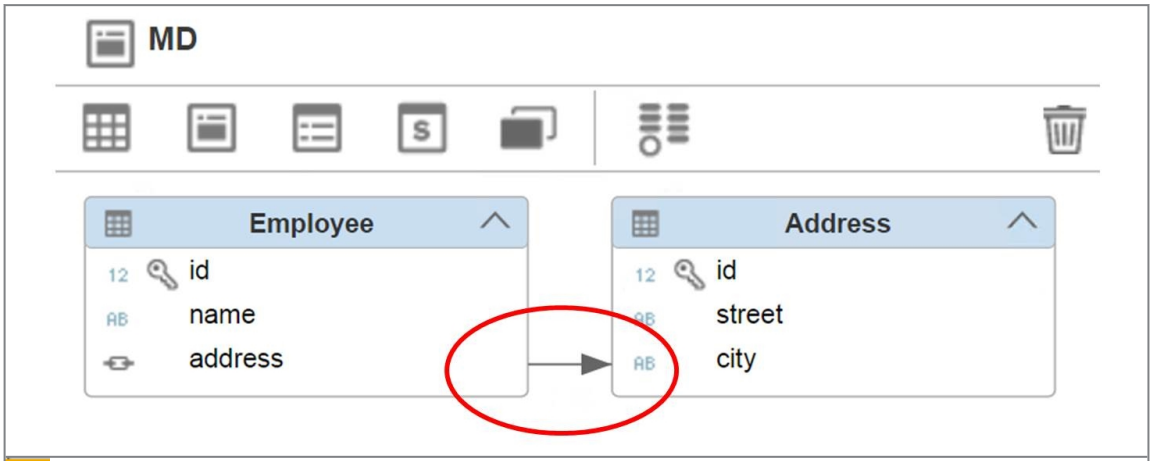


Figure 85: Association in the Graphical CDS Editor

In the graphical CDS editor, the association appears as an arrow.

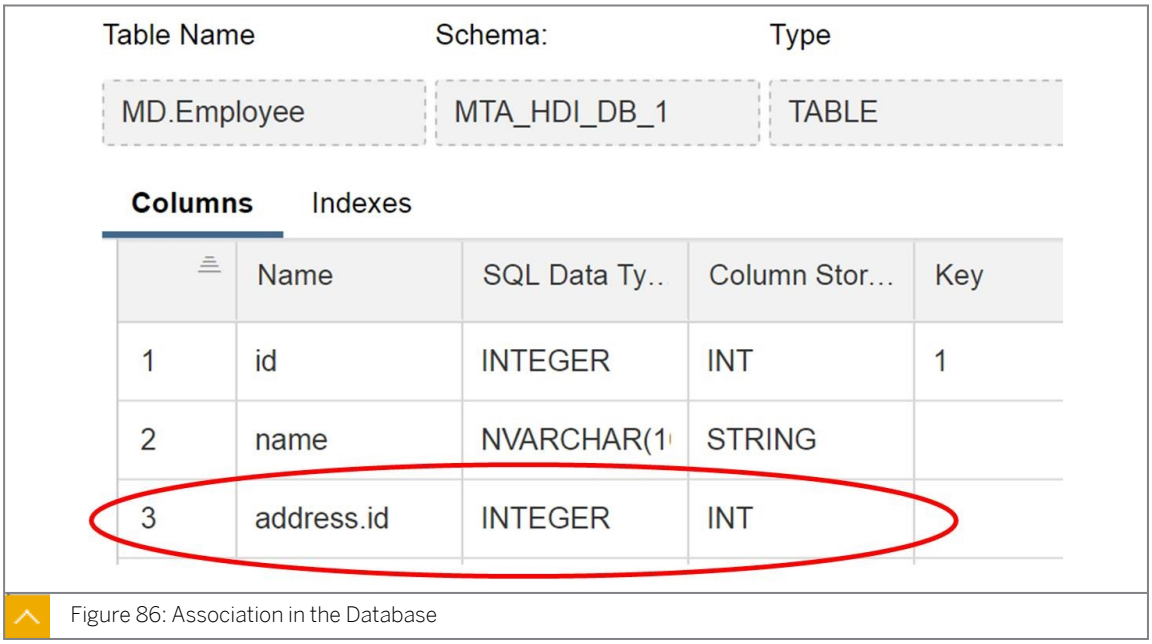


Table Name: MD.Employee | Schema: MTA_HDI_DB_1 | Type: TABLE

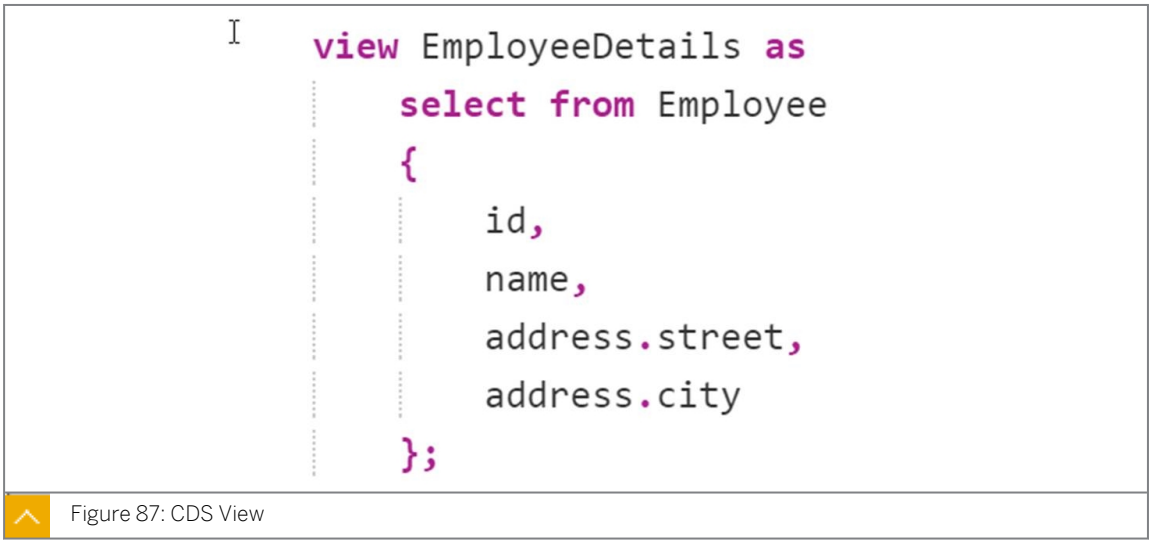
Columns | Indexes

	Name	SQL Data Ty..	Column Stor...	Key
1	id	INTEGER	INT	1
2	name	NVARCHAR(1	STRING	
3	address.id	INTEGER	INT	

Figure 86: Association in the Database

In the generated database table, the key fields are reported as a reference to the remote table.

CDS View



```
view EmployeeDetails as
    select from Employee
    {
        id,
        name,
        address.street,
        address.city
    };
```

Figure 87: CDS View

A view is a virtual table based on the dynamic results returned in response to an SQL statement. XS Advanced enables you to use CDS syntax to create a database view as a design time file. You can use this design time view definition to generate the corresponding catalog object when you deploy the application that contains the view-definition artifact (or just the application's database module).

LESSON SUMMARY

You should now be able to:

- Use context, association, and view in Core Data Services

Learning Assessment

1. In the SAP Web IDE for SAP HANA, when you create a new SAP HANA database module, what is added to the mta.yaml file?

   Choose the correct answers.

   - ☐ **A** A module of type HDB
   - ☐ **B** A route to the SAP HANA database
   - ☐ **C** A service of type *com.sap.xs.hdi-container*
   - ☐ **D** A service of type *OData*

2. In the SAP Web IDE for SAP HANA, you want to use Core Data Services to define the persistency layer. Which extension do you use for the design time file?

   Choose the correct answer.

   - ☐ **A** .hdbddl
   - ☐ **B** .hdbtable
   - ☐ **C** .hdbcds

3. Upon build, the CDS entity triggers the generation of a database view with the corresponding name and structure.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

4. CDS associations define:

   Choose the correct answer.

   - ☐ **A** Relationships between entities.
   - ☐ **B** Relationships between columns of the same entity.
   - ☐ **C** Distribution of entity data across multiple database tables.

Learning Assessment - Answers

1. In the SAP Web IDE for SAP HANA, when you create a new SAP HANA database module, what is added to the mta.yaml file?

   Choose the correct answers.

   - [X] A A module of type HDB
   - [ ] B A route to the SAP HANA database
   - [X] C A service of type *com.sap.xs.hdi-container*
   - [ ] D A service of type *OData*

   That is correct! In the SAP Web IDE for SAP HANA, when you create a SAP HANA database module, a SAP HANA database module and a *com.sap.xs.hdi-container* service are added to the mta.yaml file.

2. In the SAP Web IDE for SAP HANA, you want to use Core Data Services to define the persistency layer. Which extension do you use for the design time file?

   Choose the correct answer.

   - [ ] A .hdbddl
   - [ ] B .hdbtable
   - [X] C .hdbcds

   This is correct! You use the .hdbcds extension for the design time file.

3. Upon build, the CDS entity triggers the generation of a database view with the corresponding name and structure.

   Determine whether this statement is true or false.

   - [ ] True
   - [X] False

   That is correct! The CDS entity triggers the generation of a database table with the corresponding name and structure.

4. CDS associations define:

   Choose the correct answer.

   - [X] A Relationships between entities.
   - [ ] B Relationships between columns of the same entity.
   - [ ] C Distribution of entity data across multiple database tables.

   That is correct! CDS associations define relationships between entities.

UNIT 4

Creating the Analytical Data Model Using Calculation Views

Lesson 1

UNIT OBJECTIVES

- Describe the basic features of Calculation Views

Unit 4
Lesson 1

Introducing Calculation Views

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the basic features of Calculation Views

Key Vocabulary of Modeling

Before introducing calculation views in SAP HANA, you must become familiar with some key concepts that are frequently used when reporting on financial or operational data, such as the following:

- Measure
- Attribute
- Dimension
- Star schema
- Hierarchy
- Semantics

Measure and Attribute

When you report on data, you have to distinguish between the following important concepts:

- Measure
- Attribute

Table 1: Measure Versus Attribute

	Measure	Attribute
Definition	A numeric value, such as a price, quantity, or volume, on which you can process arithmetic or statistics operations, such as sum, average, top N values, and calculations.	An element that is used to describe a measure.

	Measure	Attribute
Examples	• Number of products sold • Unit Price • Total Price	• Product ID • Product Name • Customer ID • Customer Name • Sales Organization • Sales Org. Country • Sales Org. Region • Currency

Attributes are used to filter or aggregate the measures, in order to answer questions such as the following:

- What are the total sales originating from *Sales Org.* located in the EMEA region?
- What is the sales revenue generated by the product *Cars*?

Note:
One key objective of modeling in SAP HANA is to create a relevant association between attributes and measures to fulfill a particular reporting requirement.

Dimension

In a number of cases, analyzing the measures is easier if you group attributes together by dimension.

In the examples shown in the figure below, the sales organization is treated as a dimension, with the following associated attributes:

- Country
- Region

Similarly, a Product ID dimension could be associated with several attributes, such as product name, product category, or supplier.

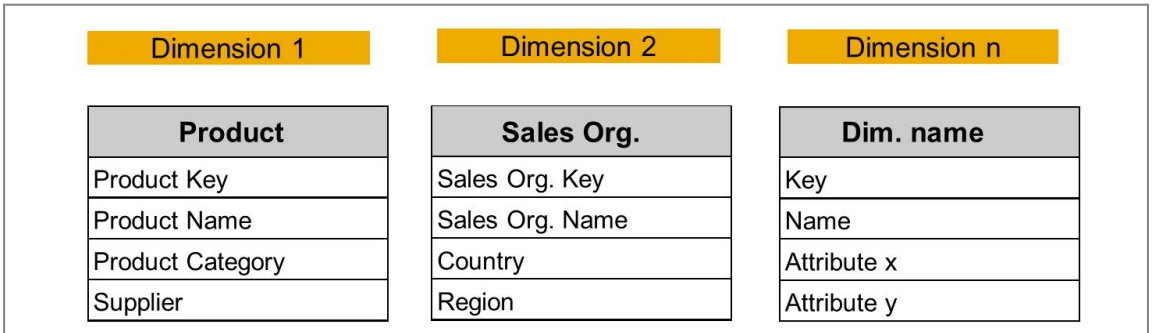


Figure 88: Dimensions

Star Schema

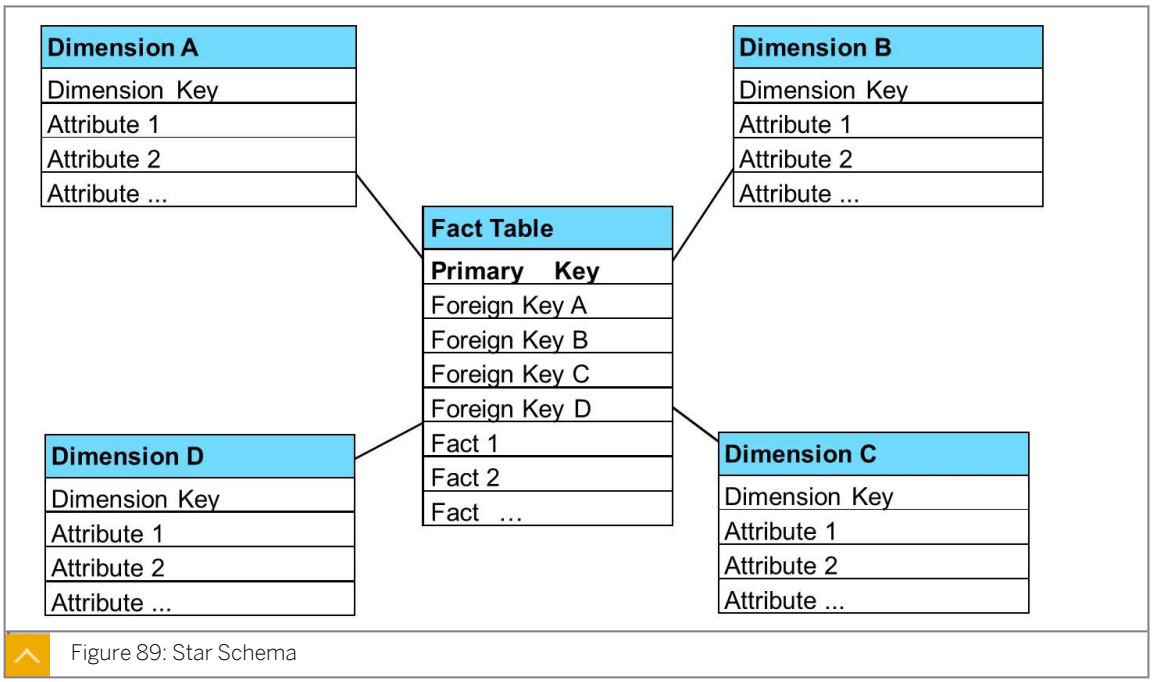


Figure 89: Star Schema

A star schema consists of one fact table that references one or several dimension tables.

The fact table contains facts, or measures, as well as the keys used to identify, for each dimension, which dimension member corresponds to each fact.

Each dimension contains attributes, such as the name of the dimension member, a description, and other attributes that can be used to filter the dimension members, build a hierarchy, and perform specific calculations.

Note:
The term *fact table* is used in a generic way. Later in the course, you will learn that calculation views in SAP HANA allow you to create this fact table.

Hierarchy

A hierarchy is a structured representation of an organization, a list of products, the time dimension, and so on, by levels.

It is used to navigate the entire set of members with more ease (the location of the company, the products, or the days, weeks, months, or years) when analyzing the data.

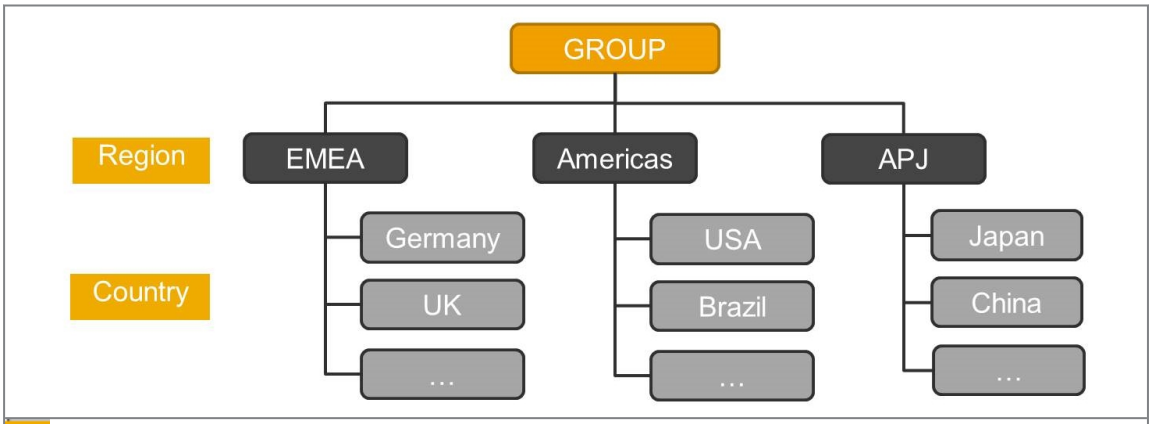


Figure 90: The Hierarchy Concept

In the figure above, you can see a geographical representation of the structure of a company by regions and countries. The level of detail (granularity) of the hierarchy will depend on how the company wants to analyze its data, and also on its design. For example, the geographical hierarchy can be based on where the subsidiaries of a company are located.

In the figure above, the top node of the hierarchy, *GROUP*, represents the entire company.

Note:
The list of products that a company sells can be organized into a hierarchy, by classifying the products by product area or product type.

Semantics

The term **semantics** is sometimes used to describe what a piece of data means, or relates to. A piece of numeric data that you report can be of different types. Here are a few examples:

- A monetary value

  For example, the total amount of sales orders.

  In this case, you might need a dimension to specific the currency (for example USD, EUR, or GBP,), if it is not implicit.

- A number of items

  For example, a number of sales orders, or a number of calls to support services.

- A weight, volume, distance, or a compound of these measures

  For example, the payload-distance in freight transportation.

  You often need to specify the unit in which the data is expressed.

- A percentage

  For example, a discount rate, or a tax rate.

Note:
Even if the semantics are not always expressed explicitly in the data repository itself, it is very important to know and understand the semantics of any measure to avoid misinterpretation of the data or irrelevant calculation.

For example, before summing up the amount of sales orders, you must make sure that they are all expressed in the same currency.

As another example, you have to be careful with aggregations when using columns that contain ratios.

Calculation Views in SAP HANA

Calculation views are used in SAP HANA to create a virtual data model, based on the data that resides in the SAP HANA database.

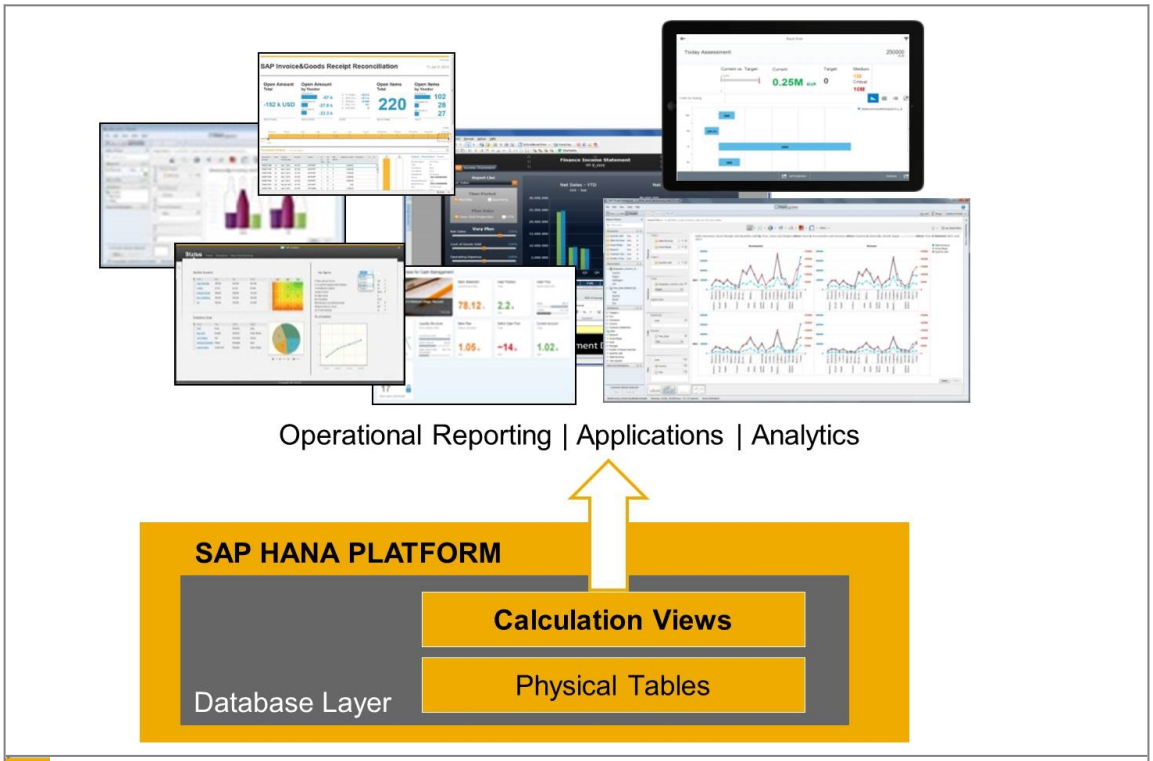


Figure 91: Calculation Views

The purpose of calculation views is to organize the data from the individual transactional tables, and to perform a variety of data calculations, in order to get a relevant and meaningful set of measures and dimensions or attributes to answer a specific reporting need. You can make the data more meaningful than it is in the source tables by customizing the column names, assigning label columns to key columns, and calculating additional attributes.

Benefits of Calculation Views

The main benefits of using calculation views in SAP HANA are as follows:

- All calculations are performed on the fly within the database engines.

No aggregates need to be pre-calculated, and the front-end reporting tools delegate most of the data processing workload (filtering, aggregation, calculations) to the SAP HANA in-memory engines in order to achieve very high performance.

- Reusability

  Each calculation view can be used (referenced) by other calculation views.

- Flexibility

  Calculation views provide a number of features that make them very flexible. For example, defining hierarchies, filtering data, generating prompts for variables and input parameters, and performing currency conversion.

- Adaptability

  An SAP HANA calculation view can adapt its behavior to the list of columns that are selected or projected on top of it. For example, the granularity of a Rank node can be determined dynamically depending on whether you query results by country or by country and customer.

- Easy to transport

  SAP HANA provides powerful tools to transport calculation models between different SAP HANA databases, for instance, to install SAP-delivered calculation models, or transport your own calculation models between your development, quality assurance, and production landscapes.

Note:
These benefits are illustrated in the following lessons of this unit, and in the following units.

Design Time Versus Runtime Calculation Views

In SAP HANA, calculation views, like a large variety of development objects, can exist both as a design time and runtime object.

- Design Time View

  The design time view is the object that you create and modify with a graphical or text-based editor.

- Runtime View

  The runtime view is a database object, more specifically a column view, that is used when you preview the data within SAP HANA, execute a SQL query, or execute a query on top of the calculation view with an external tool, such as BI tools, Microsoft Excel, and so on.

Caution:
A view cannot show its data before a runtime version of the view is successfully created in the SAP HANA database. Similarly, any change to a design time view is only visible in a query after the runtime view has been successfully updated.

Analytical Versus Transactional Requirements

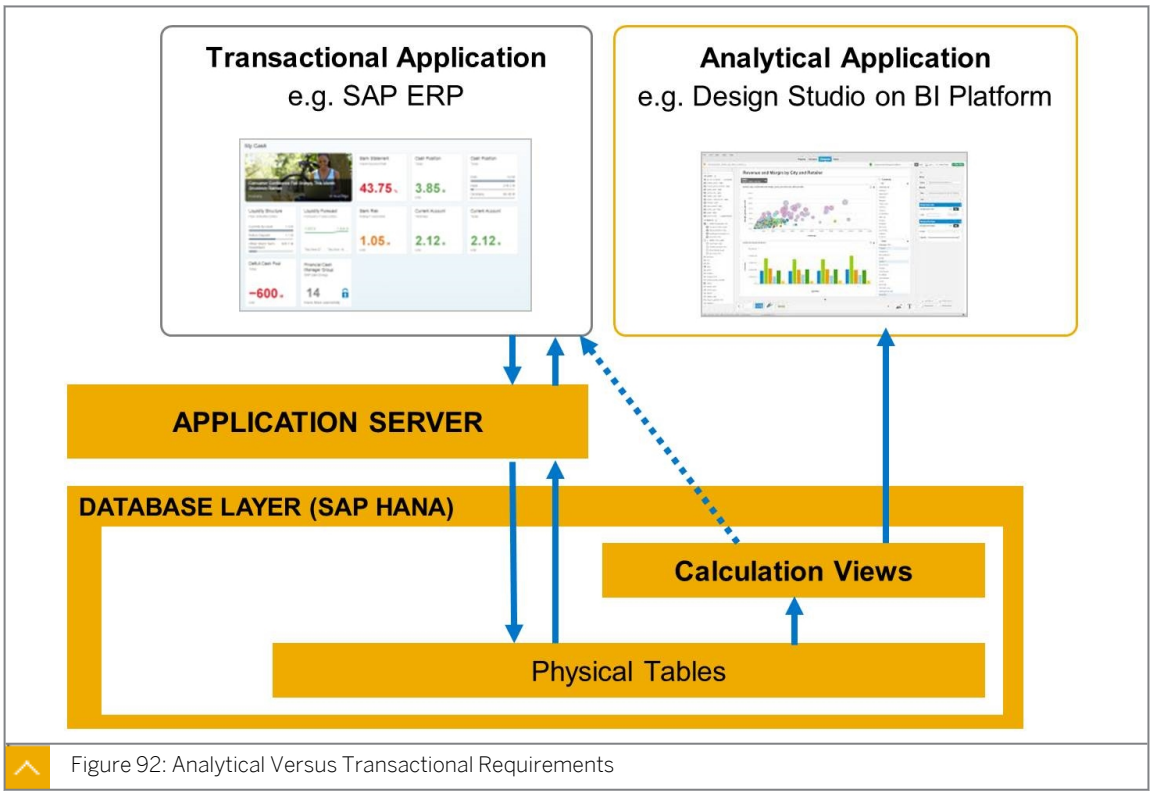


Figure 92: Analytical Versus Transactional Requirements

In transactional applications, such as SAP ERP, the underlying data (stored in physical tables) is generally handled by the application server. This layer is necessary to handle the business process, however complex it can be, while still ensuring data consistency at any time when updating multiple tables and checking the authorizations of the user.

On the other hand, calculation views are used only to retrieve data, without making any changes to the source transactional data.

For this reason, as shown in the figure above, the analytical process can bypass the application server, which makes it even faster. The reporting tools directly query the SAP HANA calculation views, where data is calculated on-the-fly.

Note:
In some cases, the transactional application can also leverage the real-time capabilities of SAP HANA calculation views by displaying inside the transactional application (for example in the SAP Fiori launchpad) real-time analytical data that is relevant for the end user.

Calculation Views and the SAP HANA Engines

The index server of SAP HANA provides several engines to process queries against calculation views.

Main SAP HANA Engines for Executing Calculation Views

The following are the main SAP HANA engines for executing calculation views:

- Join Engine
- OLAP Engine
- Calculation Engine

The distribution of the overall view execution between these engines is complex and out of the scope of this course.

However, you can keep in mind that optimization processes exist in order to delegate the execution of a calculation view to the most relevant engine.

Overview of Calculation View Optimization

When executing a query against a calculation view, SAP HANA uses optimization processes in order to achieve the best possible performance. The optimization processes are enhanced over time, with each release of SAP HANA.

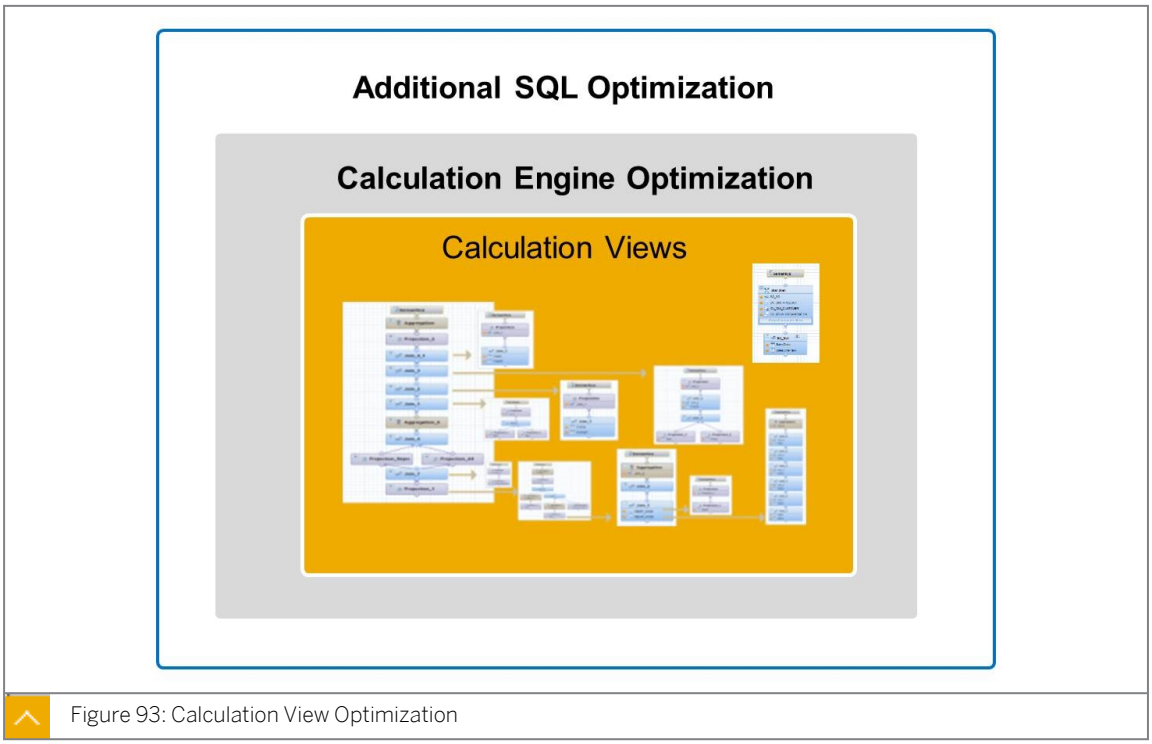


Figure 93: Calculation View Optimization

In SAP HANA SPS10 and above, when you query a calculation view, the optimization is made in two main steps:

- The initial calculation engine optimization generates a single SQL statement across a stacked model, which is then passed to the SQL optimizer.
- The SQL optimizer adds additional optimizations and delegates operations to the best database execution operator.

  For example, it delegates that the execution of the star schema joins onto the online analytical processing (OLAP) engine whenever it is possible.

Querying Calculation Views Data

When building a calculation view, the SAP Web IDE for SAP HANA allows you to preview the data of the view or to execute a custom query on top of the view. It is important to understand the difference between these approaches.

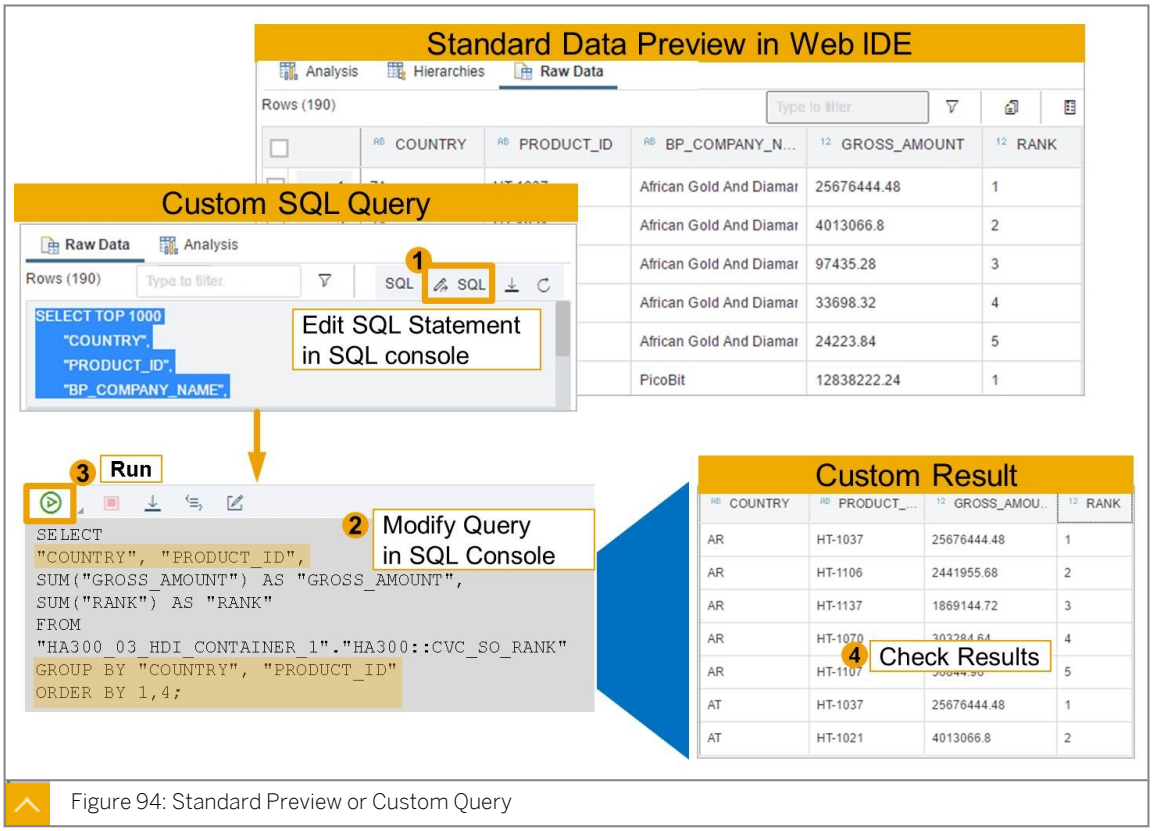


Figure 94: Standard Preview or Custom Query

Standard Data Preview

With the standard data preview, you select all the columns that are included in the semantics of the calculation view (provided that they are not hidden).

You can move the columns (using drag and drop), apply a temporary filter on one or several attribute columns, and order the result set by one (and only one) column.

Custom SQL Query

An alternative to the standard data preview is to execute a custom SQL query. As shown in the figure above, you can modify the default SQL statement corresponding to the data preview in an SQL console. For example, you can change the selected columns or the GROUP BY clause, and order the result set by one or more columns.

This is particularly useful to perform a thorough test of calculation views with a complex scenario (stacked calculation views, counter measures, join between several aggregation nodes with different *GROUP BY* columns).

Standard Data Preview Features

Even though the data preview editor is not a reporting tool, it still offers analysis functionality that can be useful during modeling or troubleshooting. It is comprised of the tabs shown in the following table, each offering specific capabilities:

Table 2: Data Preview Tabs

Tab	Displays	Use Case
Raw Data	All data	Basic display of contents
Analysis	Selected attributes and measures in tables or graphs	Profiling and analysis

Tab	Displays	Use Case
Hierarchies	Hierarchy tree with basic navigation	Navigating hierarchies in DIMENSION or CUBE Calculation Views without additional front-end tool

A setting in the SAP Web IDE for SAP HANA defines whether a filter must be set before fetching view or table data. To check or change this setting, choose *Tools → Preferences → Data Preview*.

If the option is selected, the default data preview is not executed immediately when you choose *Data Preview*.

Deferred Default Query Execution

Deferring the default query execution can be useful in the following situations:

- When the user wants to apply additional criteria to the data preview.

  For example, to set an additional filter on one or several columns (measures or attributes).

- When the user wants to execute a custom query derived from the standard data preview query and does not need to execute the standard data preview.

Note:
Even with the option *Require that at least one filter be set before fetching view or table data* selected, it is still possible to execute the data preview without any filter. A dialog box is displayed to confirm that you want to fetch the data anyway.

If you are testing a very complex view, and you want to temporarily reduce the result set to one specific order ID. Executing the default query in this case is not useful, and might cause you to lose time. Instead you can perform the following steps:

1. In the SAP Web IDE for SAP HANA preferences, select the *Require that at least one filter be set before fetching view or table data* option.
2. Define a filter on the *ORDERID* column.
3. Select *Refresh Data*.

LESSON SUMMARY
You should now be able to:

- Describe the basic features of Calculation Views

Learning Assessment

1. What do you use calculation views for?

   Choose the correct answer.

   - [ ] A To code using the R programming language.
   - [ ] B To provision data from remote sources.
   - [ ] C To resolve complex mathematical equations.
   - [ ] D To read data in real time for analytical reporting.

Learning Assessment - Answers

1. What do you use calculation views for?

   Choose the correct answer.

   - [ ] A To code using the R programming language.
   - [ ] B To provision data from remote sources.
   - [ ] C To resolve complex mathematical equations.
   - [x] D To read data in real time for analytical reporting.

   That is correct! You use calculation views to read data in real time for analytical reporting.

UNIT 5 Processing Data with SQLScript

UNIT OBJECTIVES

- Explain the basic concepts of SQLScript
- Create an SQLScript procedure
- Debug SQLScript using the SAP Web IDE for SAP HANA

Unit 5
Lesson 1

Introducing SQLScript

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Explain the basic concepts of SQLScript

What is SQLScript?

SQLScript Extends Standard SQL

SQLScript is a set of extensions added on top of standard SQL. It is used to exploit the specific features of SAP HANA.

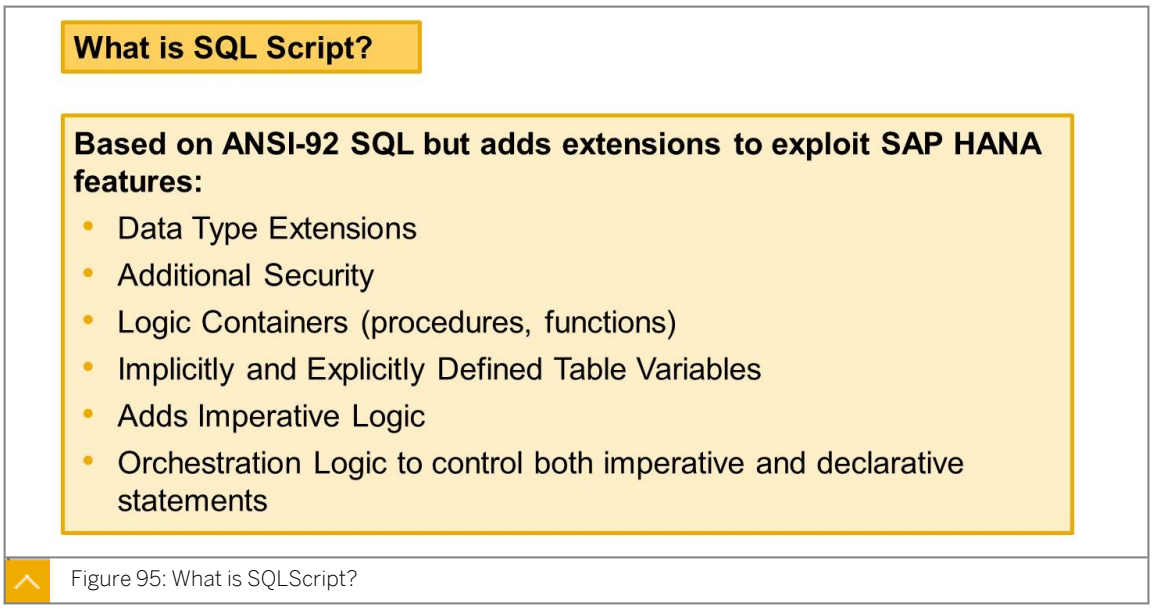


Figure 95: What is SQLScript?

By using these extensions, SQLScript allows much more pushdown of the data–intensive processing to the SAP HANA database, which otherwise would have to be done at the application level.

Applications benefit most from the potential of SAP HANA when they perform as many data-intensive computations in the database as possible. This avoids loading large amounts of data into an application server separate from SAP HANA, and leverages fast column operations, query optimization, and parallel execution. This can be achieved to a certain extent if the applications use advanced SQL statements. However, sometimes you may want to push more logic into the database than is possible when using individual SQL statements, or make the logic more readable and maintainable. Therefore, SQLScript has been introduced to assist with this task.

SQLScript Definition and Goal

- SQLScript is defined as follows:
  - The language to write stored procedures and user-defined functions in SAP HANA
  - An extension of ANSI SQL
- The main goal of SQLScript is to allow the execution of data intensive calculations within SAP HANA. This is helpful for the following reasons:
  - It eliminates data transfer between database and application tiers.
  - It executes calculations in the database layer to benefit from fast column operations, query optimization, and parallel execution.

SQLScript Advantages

Compared to standard SQL, SQLScript provides the following advantages:

- Using SQLScript, complex logic can be broken down into smaller chunks of code. This encourages a modular programming style which means better code reuse. Standard SQL only allows the definition of SQL views to structure complex queries, and SQL views have no parameters.
- SQLScript supports local variables for intermediate results with implicitly-defined types. With standard SQL, it would be required to define globally visible views even for intermediate steps.
- SQLScript has flow control logic such as if-then-else clauses that are not available in standard SQL.
- Stored procedures can return multiple results, while a standard SQL query returns only one result set.

Pushing the processing to SAP HANA is beneficial because there are lots of opportunities for SAP HANA to optimize the execution with in–memory, parallel processing.

Standard SQL does not provide sufficient syntax to push many calculations to the database and as a result, the application layer has to take on this duty. This means huge amounts of data must be copied between the database server and the application server.

SQL	SQLScript
Offload very limited functionality into the database using SQL. Most of the application logic is normally executed on an application server	SQLScript uses the SQL extensions (for the SAP HANA database), allowing developers to push data-intensive logic to the database, better perfomance
SQL views cannot be parameterized, which limits their reuse	Re-usable
Do not have features to express business logic (for example a complex currency conversion)	Provide superior optimization possibilities
SQL query can only return one result at a time	Implement algorithms using a set-oriented paradigm and not using a one record at a time paradigm. (Imperative logic is required like iterative approximation algorithms)
SQL is a declarative language	It is possible to mix ímperative constructs (procedural language) known from stored procedures with declarative ones

Figure 96: Standard SQL VS. SQLScript

It is important to remember that SQLScript is not a full application programming language such as ABAP or C++. However, SQLScript can reduce the need to program data intensive tasks in the application layer by providing imperative language. Imperative language allows the developer to add very precise control-flow logic, for example, to the read tables, one record at a time and process each record before returning to read the next record. Standard SQL does not allow this, and only provides a way to write set-based logic that returns complete data sets from an instruction. That is why standard SQL needed to be extended to add more programmatic control in the database layer.

The cost of using the SQLScript imperative language elements is that you potentially break the automatic optimization by SAP HANA. This is because you introduce dependencies in your logic. For example, before you can apply a discount, you first need to read through all customer sales records line by line, check if they are eligible records, and sum the total sales to look up a discount table based on spend amount. With SQLScript you are able to take over the control of the logic flow to make sure each step happens in the right order. However, the parallelization potential is limited if queries, that should otherwise be able to execute independently, get caught up in this sequenced logic.

Declarative VS. Imperative Logic

Declarative VS. Imperative

SQL is a descriptive language that is sometimes called a declarative language.

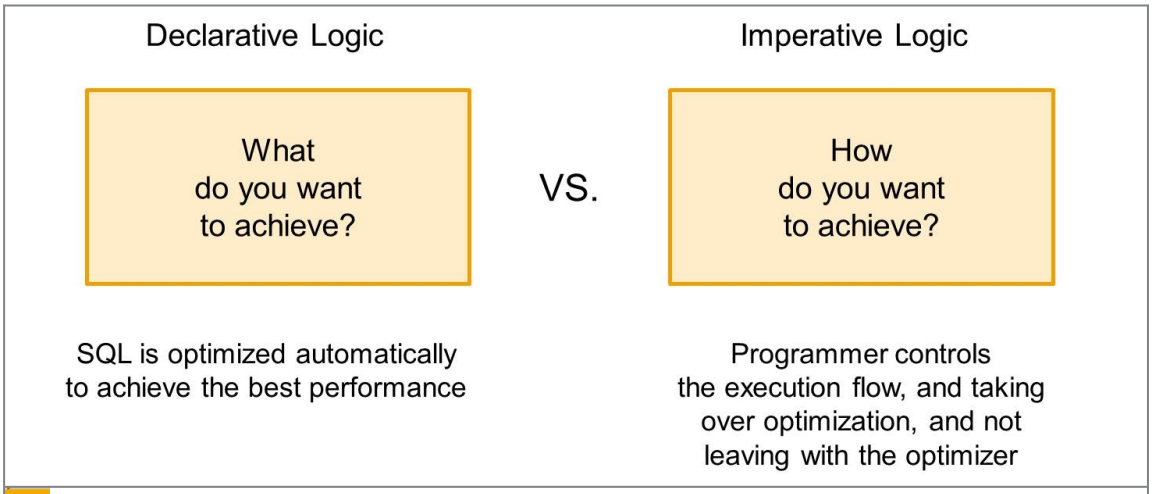
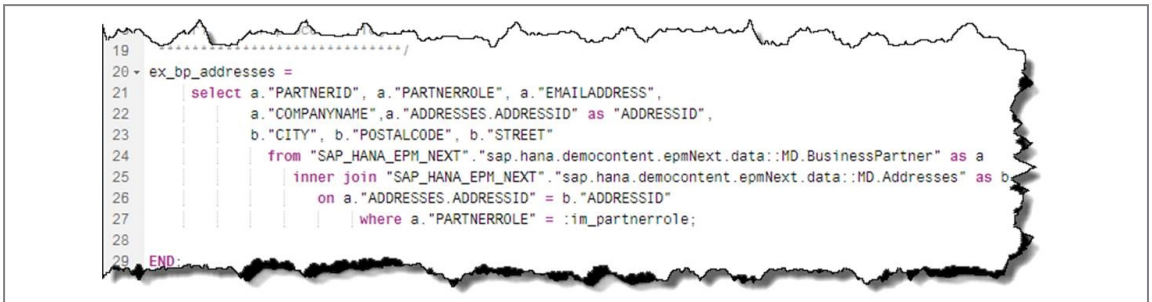


Figure 97: SQLScript Adds Imperative Logic

SQLScript is written in either **Procedures** or **Functions**. These are XS Advanced source objects that are part of a complete SAP HANA application.



```
19
20 ex_bp_addresses =
21     select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS",
22            a."COMPANYNAME",a."ADDRESSES.ADDRESSID" as "ADDRESSID",
23            b."CITY", b."POSTALCODE", b."STREET"
24              from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.BusinessPartner" as a
25                inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Addresses" as b
26                   on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
27                      where a."PARTNERROLE" = :im_partnerrole;
28
29 END;
```

Figure 98: Declarative Logic

Declarative logic allows the developer to declare the data selection via SELECT statements, as follows:

- The developer defines the what.
- The engine defines the how, and executes accordingly.

This enables massive parallelization.

```
17  declare lv_category nvarchar(40) := null;
18  declare lv_discount decimal(15,2) := 0;
19
20  lt_product = select PRODUCTID, CATEGORY, PRICE
21               from "sap.hana.democontent.epmNext.data::MD.Products"
22                    where PRODUCTID = :im_productid;
23
24  select CATEGORY into lv_category from :lt_product;
25
26  if :lv_category = 'Notebooks' then
27       lv_discount := .20;
28  elseif :lv_category = 'Handhelds' then
29       lv_discount := .25;
30  elseif :lv_category = 'Flat screens' then
31       lv_discount := .30;
32  elseif :lv_category like '%printers%' then
33       lv_discount := .30;
34  else
35       lv_discount := 0.00;  -- No discount
36  end if;
37
38  ex_product_sale_price =
39          select PRODUCTID, CATEGORY, PRICE,
40               PRICE - (PRICE * :lv_discount) as "SALEPRICE"
41                    from :lt_product;
42  END;
```

Figure 99: Imperative Logic

Imperative logic allows the developer to control the flow of the logic within SQLScript using the following:

- Scalar variable manipulation
- DDL/DML logic
- WHILE loops
- Branching logic based on some conditions, for example IF/ELSE

Logic is executed exactly as scripted (procedurally).

Code Pushdown

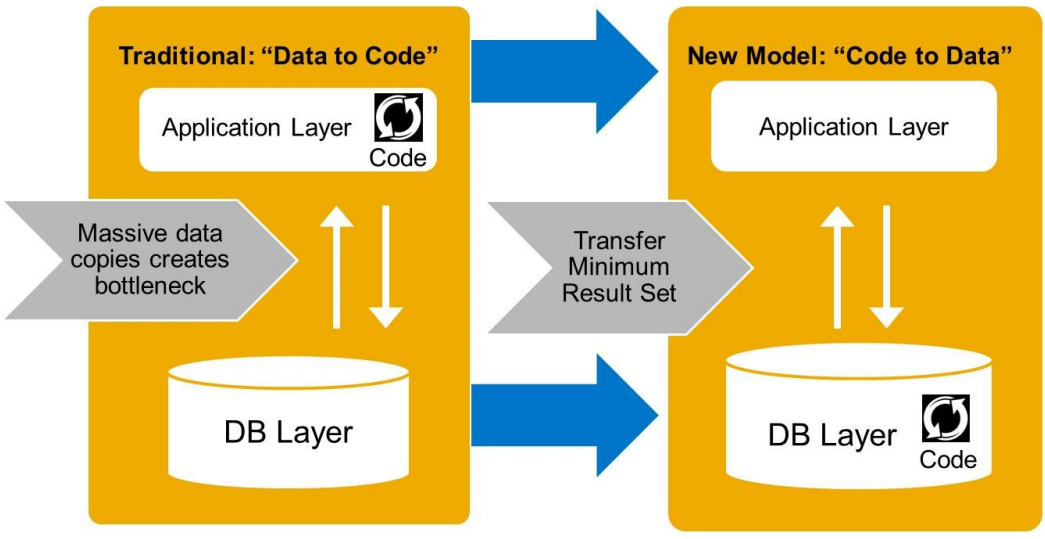


Figure 100: Traditional Programming Model Versus New Programming Model

We need to change the way we think about application development. Previously, for example in ABAP, we did a SELECT * INTO TABLE and brought thousands of rows back to the

application server, then looped over it and did some processing. The new model suggests a different approach, where we take the data-intensive logic and process that logic in the database closer to the data, and then only send back to the application layer what is necessary for presentation to the end user for further processing. This is referred to as code pushdown. The SAP Business Suite on SAP HANA does this in a couple of ways, for example, in ABAP, we can now leverage SAP HANA views by exposing them to the ABAP dictionary, via external views. We can also expose SQLScript procedures and call them directly from ABAP, using the CALL DATABASE PROCEDURE statement.

LESSON SUMMARY

You should now be able to:

- Explain the basic concepts of SQLScript

Creating an SQLScript procedure

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create an SQLScript procedure

Describing the Structure of a Database Procedure

get_product_sale_price.hdbpro... ×

```
1  PROCEDURE "get_product_sale_price" (
2      IN im_productid NVARCHAR(10),
3      OUT ex_product_sale_price "Procedures.tt_product_sale_price" )
4     LANGUAGE SQLSCRIPT
5     SQL SECURITY INVOKER
6     --DEFAULT SCHEMA <default_schema_name>
7     READS SQL DATA AS
8  BEGIN
9     /***********************************
10       Write your procedure logic
11    ***********************************/
12 declare lv_category nvarchar(40) := null;
13 declare lv_discount decimal(15,2) := 0;
14 declare lv_price decimal(15,3) := 0;
15 declare ex_found boolean;
16
17 lt_product = select PRODUCTID, CATEGORY, PRICE
18                 from "MD.Products"
19                         where PRODUCTID = :im_productid;
20 if ( :lt_product.SEARCH("PRODUCTID", :im_productid, 1) ) > 0 then
21   ex_found = true;
22  else
23   ex_found = false;
24  end if;
```

Figure 101: Procedure

In a database management system (DBMS), a stored procedure is a set of SQL statements with an assigned name that is stored in the database.

A stored procedure is used to consolidate and centralize data intensive logic. It can accept import parameters and return export parameters.

At design time, it is defined via the .hdbprocedure artifact.

Procedure Name

Corresponds to the file name.

get_product_sale_price.hdbpro...

```
1  PROCEDURE "get_product_sale_price" (
2      IN im_productid NVARCHAR(10),
3      OUT ex_product_sale_price "Procedures.tt_product_sale_price" )
4    LANGUAGE SQLSCRIPT
5    SQL SECURITY INVOKER
6    --DEFAULT SCHEMA <default_schema_name>
7    READS SQL DATA AS
8  BEGIN
```

Figure 102: Procedure Name

Interface (Input and Output Parameters)

- Developer can define both input parameters with default values as well as output parameters. Parameters can reference simple types, spatial types, in-line table types, or global types defined via CDS
- DEFAULT keyword sets the default initial value for scalar and table parameters
- DEFAULT EMPTY initializes table IN/OUT parameters

get_product_sale_price.hdbpro...

```
1  PROCEDURE "get_product_sale_price" (
2      IN im_productid NVARCHAR(10),
3      OUT ex_product_sale_price "Procedures.tt_product_sale_price" )
4    LANGUAGE SQLSCRIPT
5    SQL SECURITY INVOKER
6    --DEFAULT SCHEMA <default_schema_name>
7    READS SQL DATA AS
8  BEGIN
```

Figure 103: Interface (Input and Output Parameters)

Header Properties

A developer can set language(SQLScript/R), security(invoker/definer), read/write access, and encryption.

get_product_sale_price.hdbpro...

```
1  PROCEDURE "get_product_sale_price" (
2      IN im_productid NVARCHAR(10),
3      OUT ex_product_sale_price "Procedures.tt_product_sale_price" )
4    LANGUAGE SQLSCRIPT
5    SQL SECURITY INVOKER
6    --DEFAULT SCHEMA <default_schema_name>
7    READS SQL DATA AS
8  BEGIN
```

Figure 104: Header Properties

Script Body

The developer writes the body of the script between the BEGIN and END statements.

get_bp_addresses_by_role.hd...

```
1  PROCEDURE "get_bp_addresses_by_role" ( in im_partnerrole nvarchar(3) DEFAULT '1',
2      out ex_bp_addresses TABLE ( "ADDRESSID" NVARCHAR(10) , "CITY" VARCHAR(40),
3                                  "POSTALCODE" NVARCHAR(10), "STREET" NVARCHAR(60) )
4  )
5    LANGUAGE SQLSCRIPT
6    SQL SECURITY INVOKER
7    --DEFAULT SCHEMA <default_schema_name>
8    READS SQL DATA AS
9  BEGIN
10 lt_employees = select "PARTNERID", "PARTNERROLE", "EMAILADDRESS", "COMPANYNAME", "ADDRESSES.ADDRESSID"
11 FROM "MD.BusinessPartner" where "PARTNERROLE" = :im_partnerrole ;
12
13 ex_bp_addresses =  MAP_MERGE(:lt_employees,
14                        "get_address_func"(:lt_employees."ADDRESSES.ADDRESSID"));
15
16 END
```

Figure 105: Script Body

LESSON SUMMARY

You should now be able to:

- Create an SQLScript procedure

Debugging SQLScript

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Debug SQLScript using the SAP Web IDE for SAP HANA

Debugging SQLScript

To use the SQLScript debugger, proceed as follows:

1. From the Database Explorer, select the procedure to be debugged.

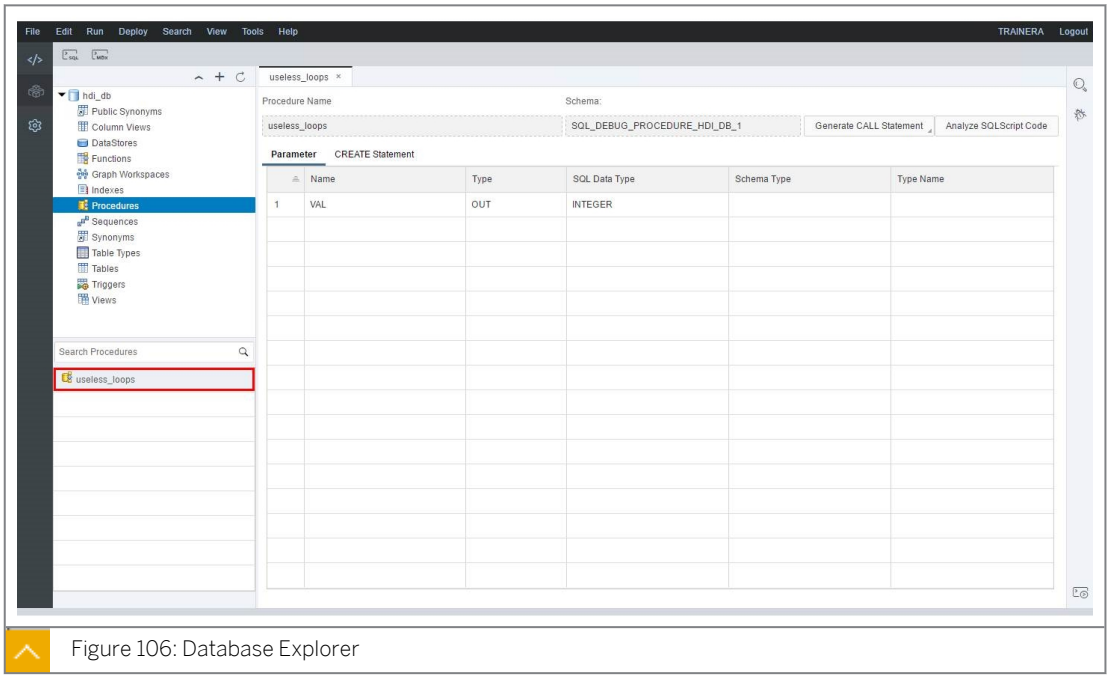


Figure 106: Database Explorer

2. To open the debug panel, choose either *View* → *Debugger* from the menu bar or click the *Debugger* icon on the right side of the screen.

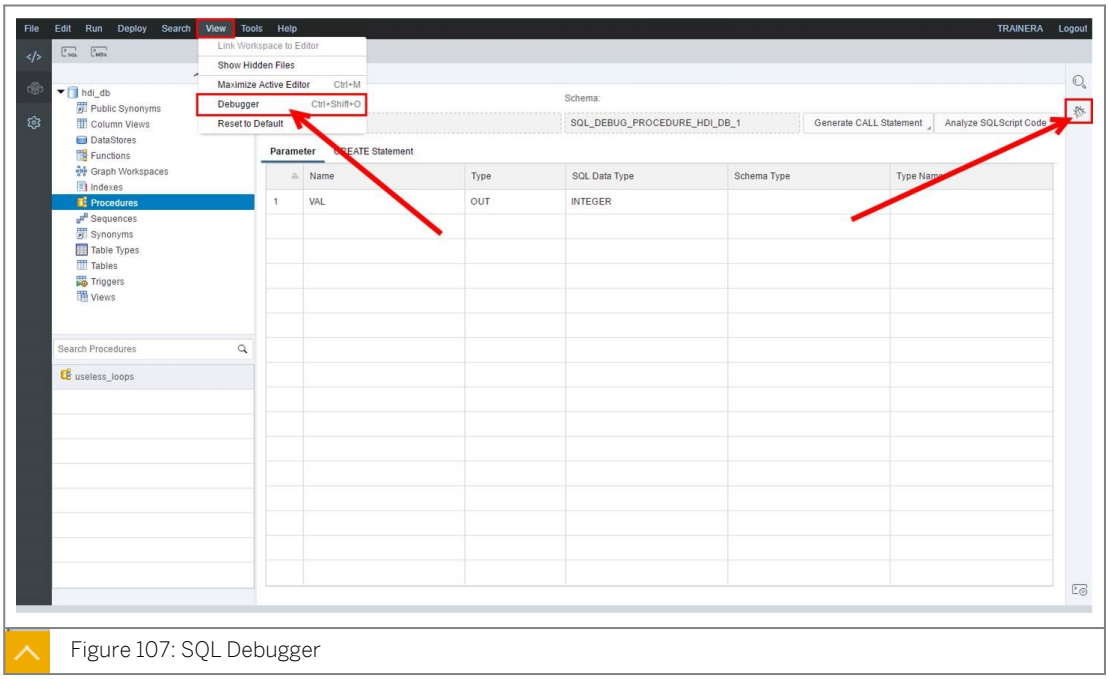


Figure 107: SQL Debugger

3. Use the *Attach Debugger* icon (plug icon next to the *Active Session* dropdown box) to link the debugger to your session.

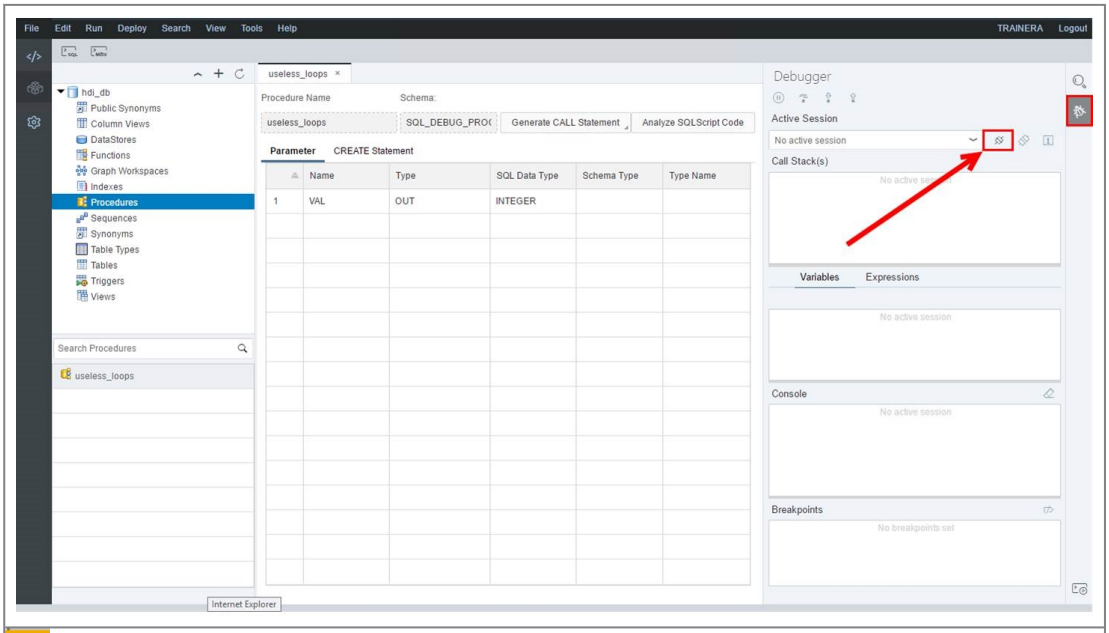


Figure 108: SQL Debugger Perspective

4. Specify the debug target which is the database connection you want to use and confirm by selecting *OK*.

   You will see a success message appear in the upper right corner of the SAP Web IDE for SAP HANA.

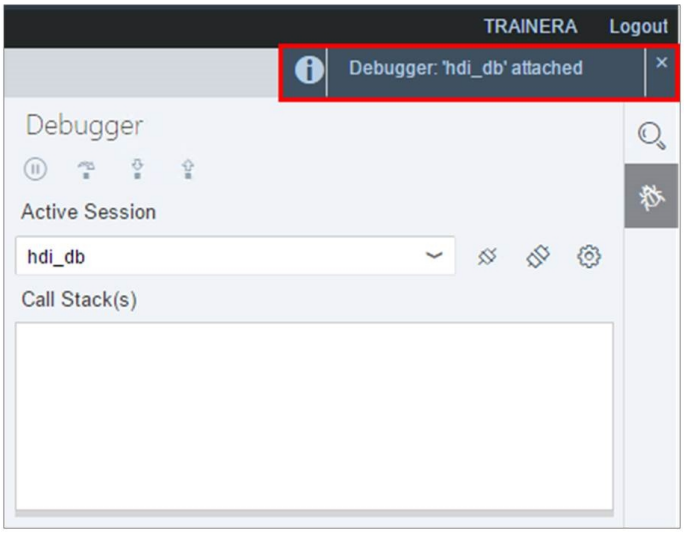


Figure 109: Attach Debugger Success Message

5. a
6. Open the procedure for debugging from the context menu: *Context Menu → Open for Debugging*.
7. Set a breakpoint by clicking the line numbers.

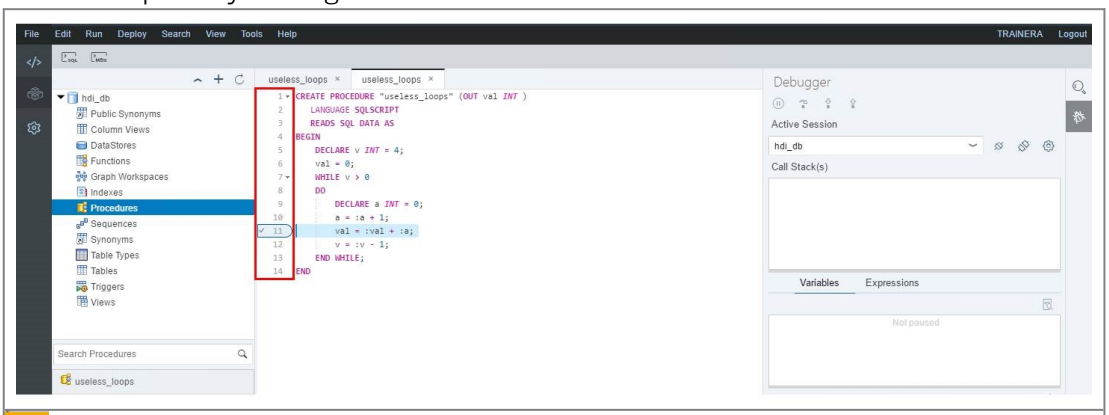


Figure 110: Setting a Breakpoint

8. From the *Context Menu* of the procedure use the *Generate CALL Statement* function to execute the procedure.
9. The SAP Web IDE for SAP HANA will switch automatically to the procedure after a breakpoint is hit.
10. Use the controls in the *Debugger* panel to *Resume*, *Step Over*, *Step In*, or *Step Out*.

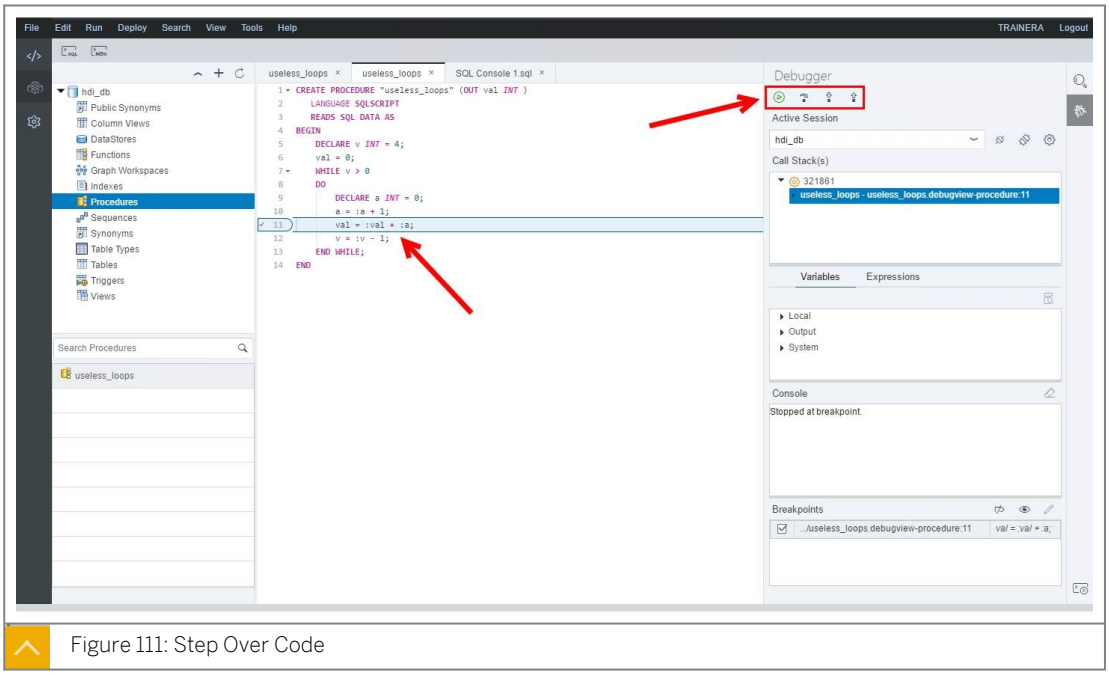


Figure 111: Step Over Code

LESSON SUMMARY

You should now be able to:

- Debug SQLScript using the SAP Web IDE for SAP HANA

Learning Assessment

1. Compared to standard SQL, which of the following are the advantages of SQLScript?

   Choose the correct answers.

   - ☐ **A** It returns a single set.
   - ☐ **B** It enables complex logic to be broken up into smaller chunks of code that enables modular programming, reuse, and a better understanding by functional abstraction.
   - ☐ **C** It has control logic, such as if/else.
   - ☐ **D** One can make use of table variables structure complex SQL statements.

2. In SAP HANA, you create a stored procedure. What can you set in the header properties?

   Choose the correct answers.

   - ☐ **A** Global variables
   - ☐ **B** Read/write access
   - ☐ **C** Storage type
   - ☐ **D** Programming language
   - ☐ **E** Security

3. In the SAP Web IDE for SAP HANA, what features are provided in the SQLScript *debugger* pane?

   Choose the correct answers.

   - ☐ **A** Runtime analysis
   - ☐ **B** Expressions
   - ☐ **C** Runtime code change
   - ☐ **D** Watchpoints
   - ☐ **E** Console

Learning Assessment - Answers

1. Compared to standard SQL, which of the following are the advantages of SQLScript?

   Choose the correct answers.

   - [ ] A It returns a single set.
   - [X] B It enables complex logic to be broken up into smaller chunks of code that enables modular programming, reuse, and a better understanding by functional abstraction.
   - [X] C It has control logic, such as if/else.
   - [X] D One can make use of table variables structure complex SQL statements.

   That is correct! SQLScript has numerous advantages over standard SQL, such as returning multiple results, complex logic, table variables, parallel execution, and control logic.

2. In SAP HANA, you create a stored procedure. What can you set in the header properties?

   Choose the correct answers.

   - [ ] A Global variables
   - [X] B Read/write access
   - [ ] C Storage type
   - [X] D Programming language
   - [X] E Security

   This is correct! In the header properties, a developer can set language (SQLScript/R), security (invoker/definer), read/write access, and encryption.

3. In the SAP Web IDE for SAP HANA, what features are provided in the SQLScript *debugger* pane?

   Choose the correct answers.

   - [ ] A Runtime analysis
   - [x] B Expressions
   - [ ] C Runtime code change
   - [x] D Watchpoints
   - [x] E Console

   That is correct! Expressions, watchpoints, and the console are provided.

UNIT 6 Using Database Security

UNIT OBJECTIVES

- Describe basic concepts about authorizations in the SAP HANA database
- Understand the HDI Security Concepts

Unit 6
Lesson 1

Introducing Authorization in SAP HANA

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe basic concepts about authorizations in the SAP HANA database

Describing Basic SAP HANA Database Security Concepts

Before taking a closer look at authorizations in the context of HDI containers, it is important to understand the basic authorization entities and the relationships between them in the SAP HANA database.

The basic authorization entities are defined as follows:

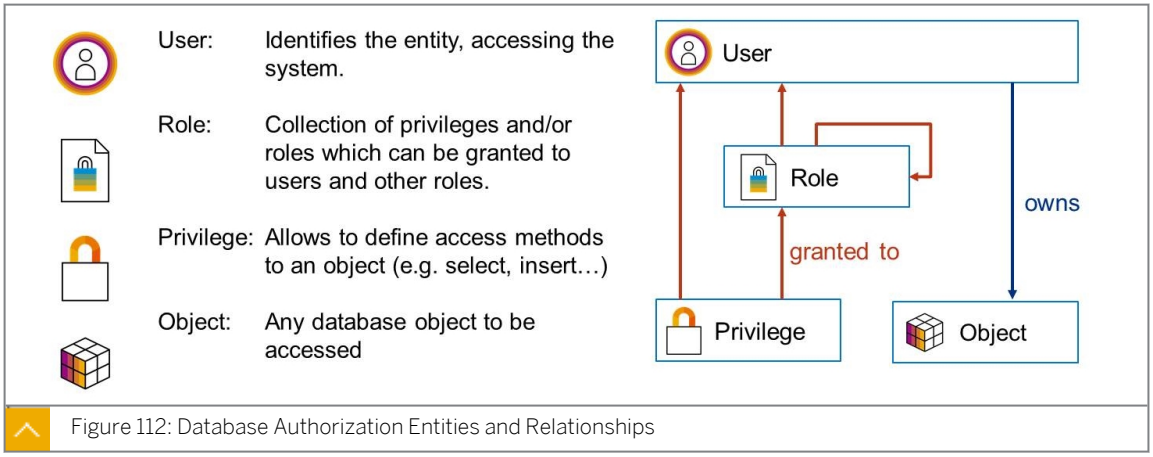


Figure 112: Database Authorization Entities and Relationships

User

Identifies the entity, accessing the system. While on the system side this always translates to a User, outside the system it could be used by a real person but also by a machine Technical User.

Object

Any object in the database which might be accessed by a user or another object. An object is owned by the user who defined it.

Privilege

Privileges are used to define allowed operations, carried out on an object. For example, `SELECT` on a database table. They can be granted to a user directly or to a role.

Role

Roles are collections of privileges and / or other roles. They can be granted to a user or to another role (nesting).

Let's elaborate a bit further on the ownership concept of database objects, such as tables, views or schemas, and the implications of this concept.

The owner of an object is the database user who creates the object. Generally speaking, only the Object Owner can initially access the object and grant privileges on the object to others. However, all objects are located in schemas and the schemas themselves are objects with an owner. The schema owner may be different to the owners of objects contained in the schema. How this works in the context of HDI containers and who the Object Owners are, is described in detail later in the course.

The figure, *Owner Versus Schema 2: How SAP HANA Handles Ownership of Database Objects*, shows the following example:

- User BILL owns the schema *BILL* and the schema *WDF*.
- User JIM owns the schema *JIM*.
- As the owners can grant privileges to others, user BILL grants user JIM the privilege to create objects in the schema *BILL*.
- User JIM creates the table *ORDERS* inside the schema *BILL* (table name = BILL.ORDERS)

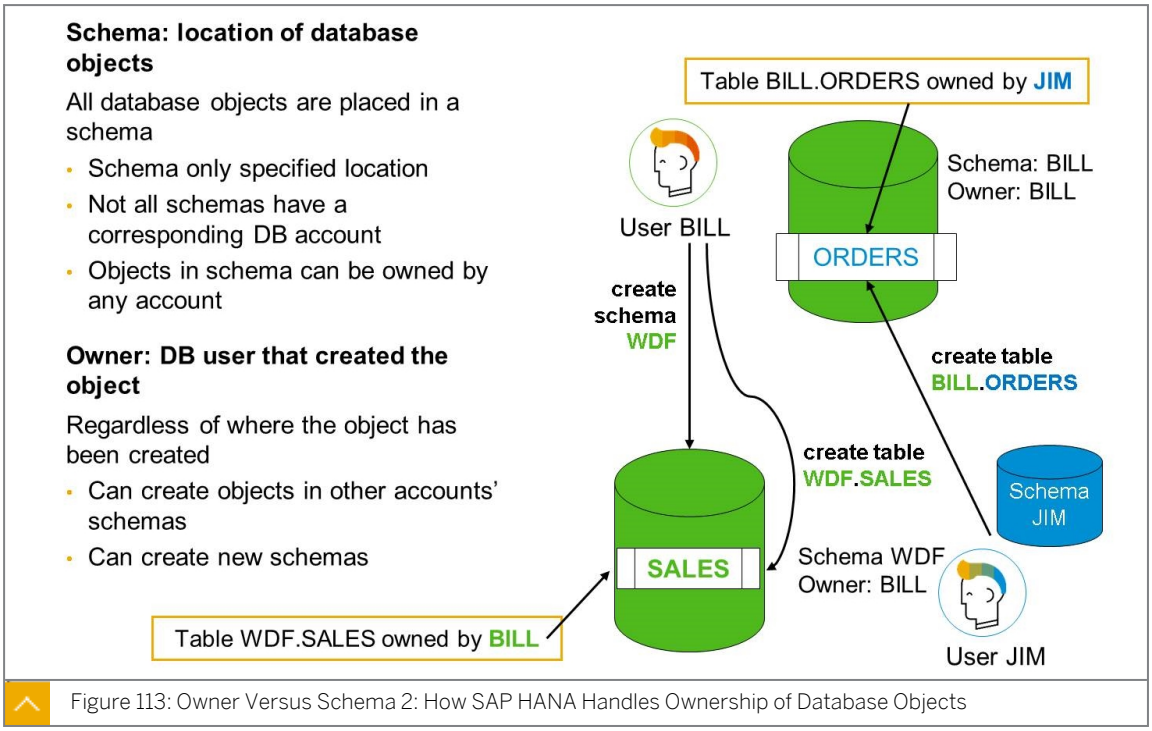


Figure 113: Owner Versus Schema 2: How SAP HANA Handles Ownership of Database Objects

Based on the example above, you need to understand which objects can be accessed by which users. The following rules apply:

- Objects can be accessed by the owner of the object.
- Objects can be accessed by the owner of the schema in which the object is located.
- Objects can be accessed by users to whom the owner of the object has granted privileges.
- Objects can be accessed by users to whom the owner of the parent schema has granted privileges.

Taking this into consideration, you can conclude that user JIM can only access schema *JIM* and the table *BILL.ORDERS*. On the other hand, user BILL has access to schema *WDF* and schema *BILL*. Notice that user BILL also has access to all the content of those schemas, and so, user BILL also has access to table *BILL.ORDERS*.

Granting Access to a Database Object

The following are the two possible options to grant a privilege:

- Granting a privilege to a user or role

  This option allows user BILL to grant a privilege to user JIM. User JIM is not allowed to further extend this privilege to others by default.

- Granting a privilege to a user or role with GRANT/ADMIN option

  This option allows user BILL to grant a privilege to user JIM. User JIM is allowed to extend this privilege to others. For example, user JIM can extend the same privilege to user HASSO or the role ADMIN.

> Note:
> To grant a privilege, the user granting the privilege either needs to be the owner of the object (owner of the object related to the privilege) or should have the privilege assigned to them with the GRANT/ADMIN option.

In the following example, user BILL granted a privilege to user JIM with the GRANT/ADMIN option. Afterwards, user JIM granted the same privilege to user HASSO and the role ADMIN.

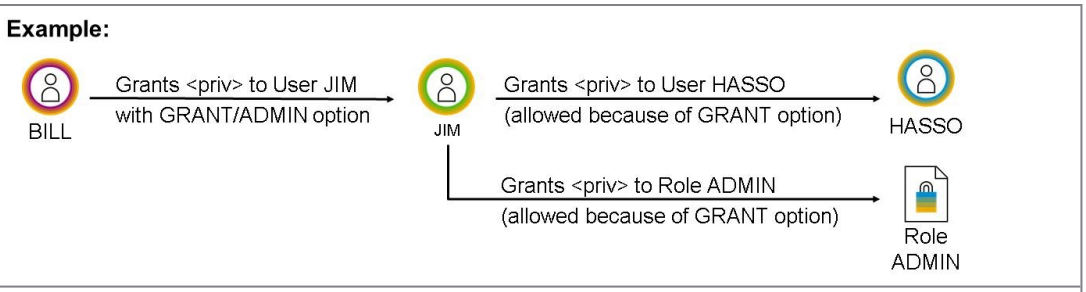


Figure 114: Granting Privileges to Users and Roles

In the case of roles, these are owned by the database user who creates them. To create a role, a database user needs to be granted all the privileges which should be included.

You can analyze the ownership of the objects in your SAP HANA system by using the system view named OWNERSHIP, SCHEMAS, or ROLES depending on the type of object you would like to analyze.

Information on object ownership can be found in system views

For catalog objects (e.g. tables, views, procedures)

- View “OWNERSHIP”
- E.g. `select * from "PUBLIC"."OWNERSHIP" where OWNER_NAME = 'RICHARD'`
- This view does not list schemas or roles; it does list analytical privileges

For schemas

- View “SCHEMAS”
- E.g. `select * from schemas where schema_name='_SYS_BIC'`

For Roles there is no actual ownership, but there is a creator:

- View “ROLES”
- E.g. `select * from roles where ROLE_NAME like '%::%';`
- Note: dropping the creator of a role does _not_ drop the role

Figure 115: Finding Ownership Information

Now, let us take a look at the implications of the ownership concept in the case when one of the users needs to be dropped. The following rules apply:

- You can only drop a user if he owns no objects (except their own schema).
- If the user owns further objects, the only option is to drop the user with the “cascade” option. This means that all the objects owned by the user are dropped and all the privileges on these objects are revoked.
- If one of the objects belonging to the user to be dropped is a schema, all of the objects located in that schema are also dropped (even if they are owned by a different user).

Be careful when dropping DB users

If user owns no objects (except its own schema)

- Simple drop sufficient

If user owns further objects

- Need to drop with “cascade” option `drop user <name> cascade`
- Drops all objects owned by this account

Examples:

`drop user JIM cascade`

- Removes user JIM
- Removes schema JIM and table BILL.ORDERS

`drop user BILL cascade`

- Removes user BILL
- Removes schemas BILL and WDF with all contents

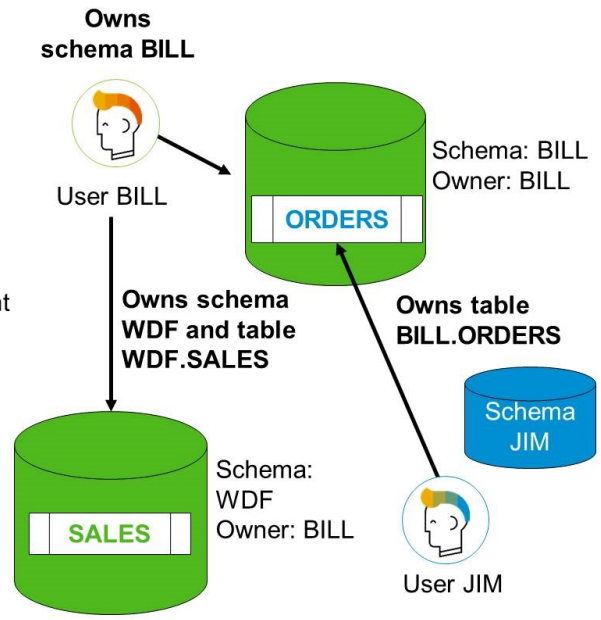


Figure 116: Dropping of Database Users: The Impact of Dropping with Cascade

Going back to the example above, let's analyze what would happen if you drop each of the users. The following rules apply:

- In both cases you need to use the cascade option as both users own other objects beside their own schema.
- If you drop user JIM with the cascade option, the schema *JIM* and the table *BILL. ORDERS* are also dropped.
- If you drop user BILL with the cascade option, the schema *WDF* and its content (table *WDF.SALES*) are dropped. Also, the schema *BILL* and the table *BILL. ORDERS* (owned by JIM) are dropped.

The deletion of users also has an impact on the privileges that the user granted to others. When a user is dropped, all the privileges granted by the user to others (users or roles) are also revoked.

So, how do you overcome the problems which emerge due to the Object Owner concept? You can use the HDI.

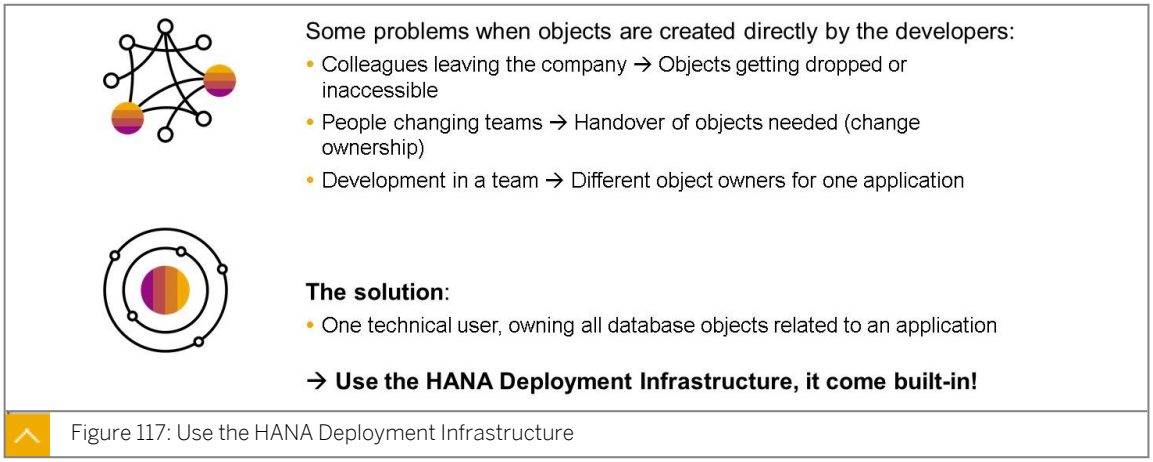


Figure 117: Use the HANA Deployment Infrastructure

In the example above, BILL and JIM left the company and therefore their SAP HANA users needed to be dropped. To avoid the dependent objects to be dropped as well, it is necessary to decouple the developed objects from their users.

This can be achieved by changing the ownership of an object to a different user before dropping the users. However, the better solution is to let one, possibly technical user, create the database objects right from the beginning.

By using the HDI, objects no longer belong to the developer, but to a technical user. The next Lesson will show how this works.

LESSON SUMMARY

You should now be able to:

- Describe basic concepts about authorizations in the SAP HANA database

Unit 6
Lesson 2

HDI Container Security Concepts

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand the HDI Security Concepts

HDI Container Isolation

As introduced earlier, the HDI is a service layer on top of the SAP HANA database that simplifies the deployment of SAP HANA database artifacts by providing a declarative approach for defining database objects and ensuring a consistent deployment into the database.

HDI containers are the place where those deployed database objects live on the database. In principle, every container relates to a database schema which is created by the HDI deployer along with several technical users who own the schema and the objects inside.

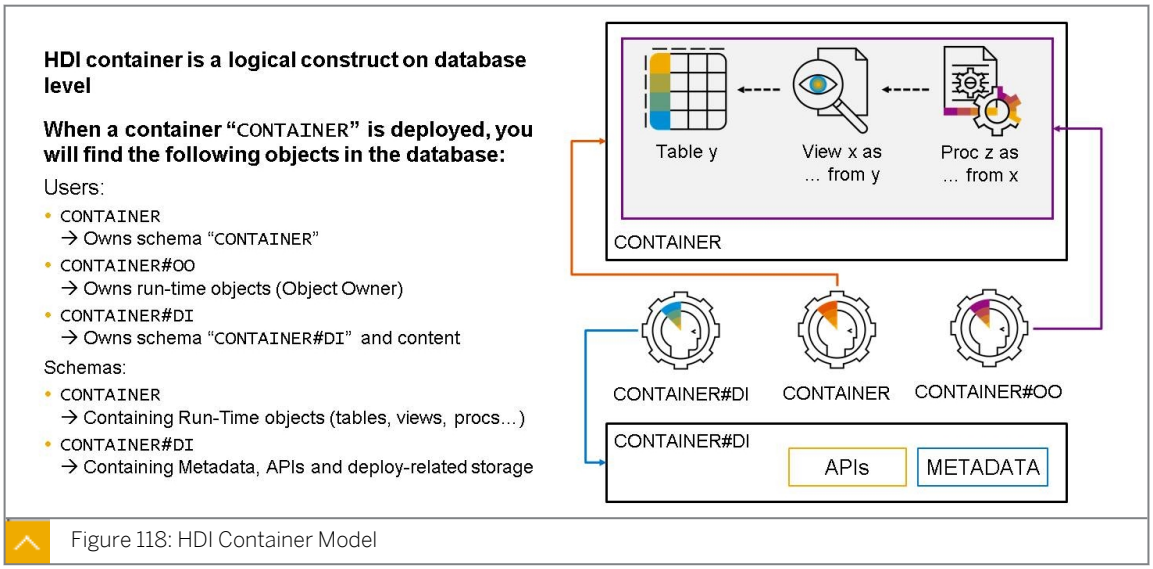


Figure 118: HDI Container Model

When a container is deployed, several objects are created on the database. One of these objects is the schema, containing the developed objects such as tables, views, procedures, and so on. This schema has the same name as the HDI container and is owned by a corresponding user with the same name, for example, the name `"CONTAINER"`in the figure above. In addition to that, a user with the suffix `#OO`, the Object Owner, is used to create the runtime objects.

At database level, the container has an additional schema for deployment related storage, metadata, and APIs identified by the suffix `#DI`, for example, `"CONTAINER#DI"` in the figure above.

HDI Container Isolation

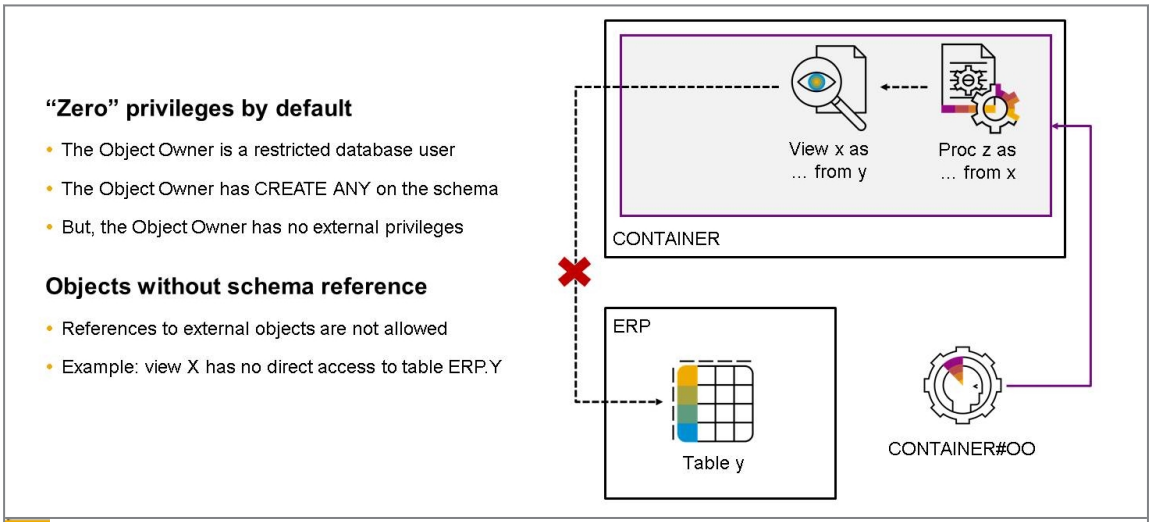


Figure 119: HDI Container Isolation

The Object Owner is created as a restricted database user which can't be used to log on to the database.

On the container schema, the user is granted the `CREATE ANY` privilege in order to create the runtime objects inside. However, apart from that, the Object Owner doesn't have privileges on external objects outside of the container.

It is also important to know, that it is not allowed to directly reference an external object from within an HDI container. If you try to access an external object from inside of a container, for example, the table *ERP.Y* in the figure above, the deployment of the HDI container will fail.

In the example above, view X has no direct access to table *ERP.Y*. The deployment of a view trying to access that object will fail, because the object can not be referenced. In addition, the Object Owner does not have the necessary privileges to read from it.

Accessing an HDI Container

Now that you know how object ownership works in an HDI environment, let's elaborate on how to access the deployed container.

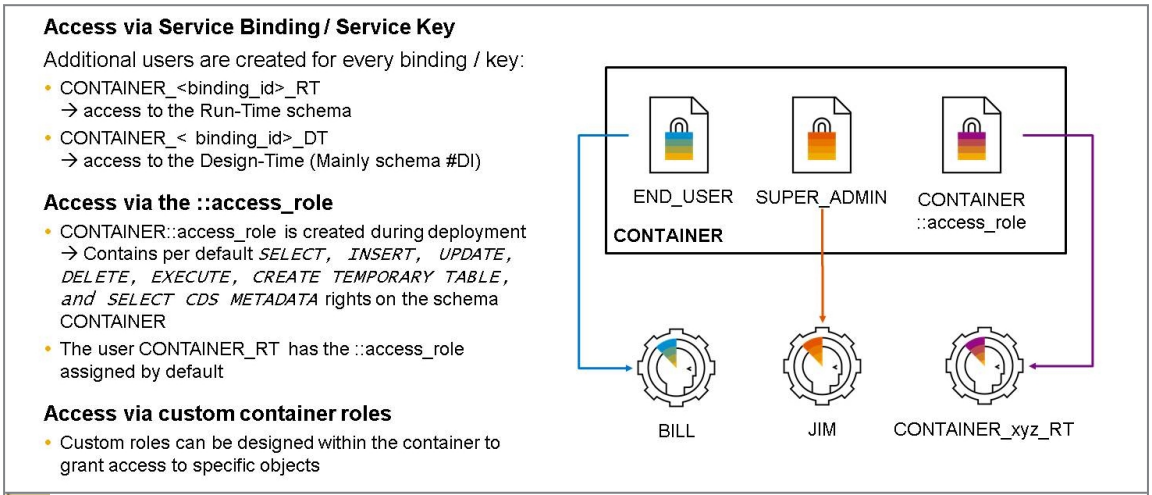


Figure 120: Accessing an HDI Container

Generally speaking, there are three ways to access an HDI container:

- **Service Binding / Service Key**

  In a micro-service oriented architecture, the container is usually accessed via a service binding between the consuming service, for example, the Node.js module in the previous lesson, and the providing service, for example, the HDI container shown in the figure above.

- **Default ::access_role**

  Access to the container can also be given to a database user by granting the automatically generated `*::access_role`. The role contains `SELECT`, `INSERT`, `UPDATE`, `DELETE`,`EXECUTE`,

  `CREATE TEMPORARY TABLE`, and `SELECT CDS METADATA` privileges on the container schema by default.

- **Custom Container Roles**

  Just like adapting the default access role, `*.hdbrole` files can be used to define roles with access to the container artefacts. The roles can then be granted to any database user.

Service Binding

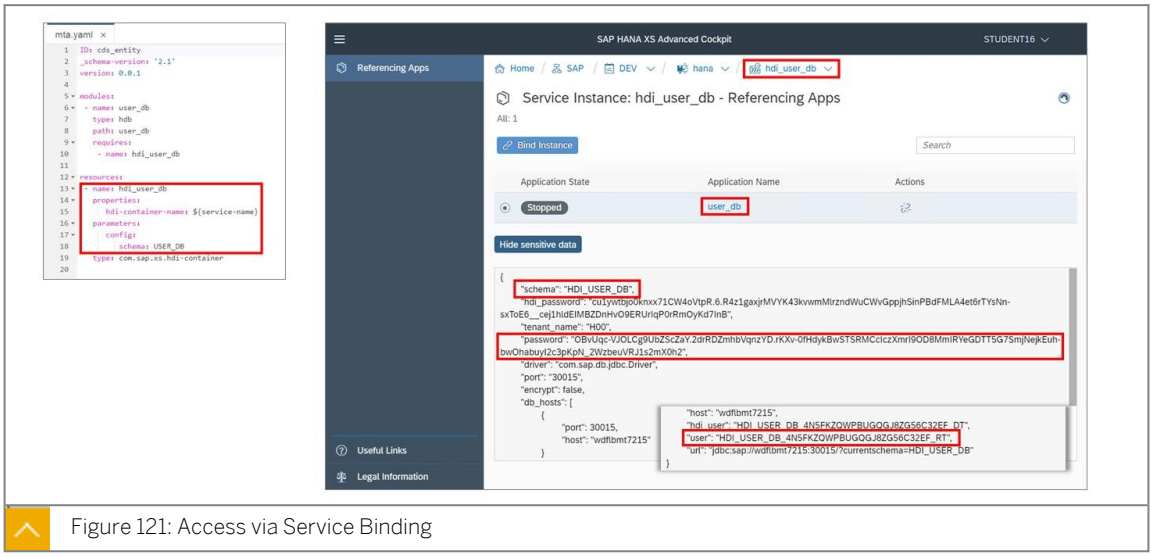


Figure 121: Access via Service Binding

The service binding is usually done automatically when deploying an application which has a dependency to a service it wants to consume.

In the example above, the HDI container `hdi_user_db` is required by the SAP HANA database module `user_db`. When the MTA is deployed, the service binding is created. The service binding contains all of the necessary information to access the container, for example, the `_RT` user and the actual schema name of the container.

Service Keys

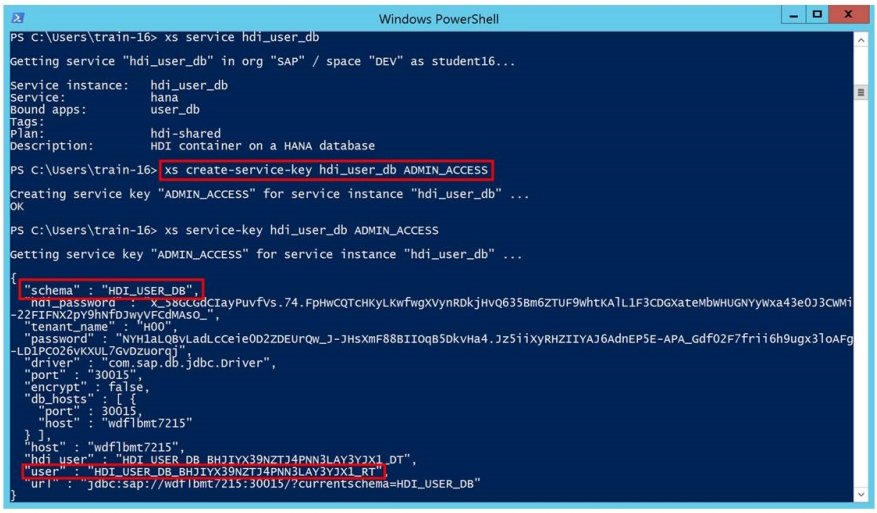


Figure 122: Access via Service Key

A service key is basically a binding which has been created manually. In the terminal window shown above, you can see the creation of a service key with the `xs create-service-key` command and the created key with the necessary connection and user information.

Default ::access_role

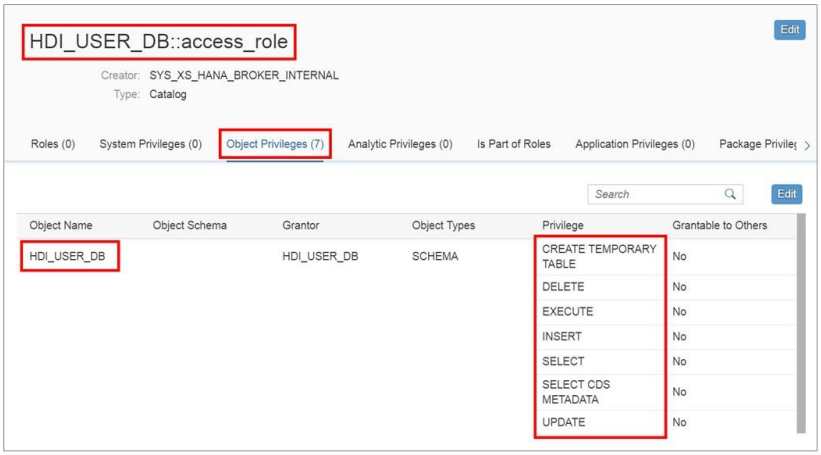


Figure 123: The Default ::access_role in the SAP HANA Cockpit

The default `::access_role` role is generated automatically during the deployment and contains the privileges that are shown in the figure above. The role can subsequently be granted to a database user which is then able to access the objects inside the HDI container schema.

If necessary, the role content can be adapted by using the `src/defaults/default_access_role.hdbrole` role in the HDI module. The privileges mentioned there will then be added to the default access role.

Creating Custom Container Roles

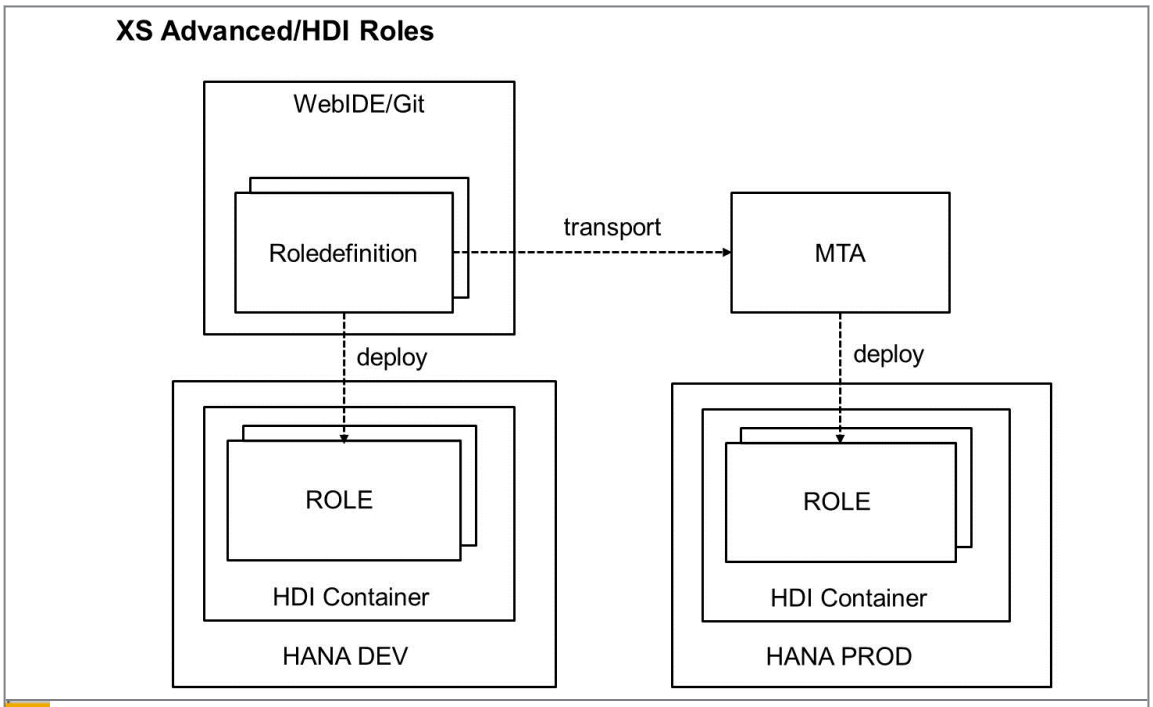


Figure 124: Design Time Database Roles and the Transport

HDI roles are created using the SAP Web IDE for SAP HANA. You can create them as design time development artifacts (text files with `.hdbrole` extension) within an MTA development project.

When the project is deployed, the runtime database roles are created.

Any change and re-build of the existing design time roles in the project will cause an automatic re-generation of the runtime role, in line with the new design. This means that any modification made to the role in the meantime will be overwritten.

When you transport the role to a new system, the design time role is transferred and then built on the target system. In this way, the runtime version is aligned to the one in the original system.

Advantages of HDI roles created via design-time artifacts

- Design-time roles are transportable
- Role creation and role ownership are de-couple from role granting
- Using a grant procedure, the grantor doesn't need to have rights to grant roles

Important properties of HDI roles created via design-time artifacts

- A grantor can grant/revoke any designed role
- Developers can edit any role in their project
- Role creation and deployment are separated

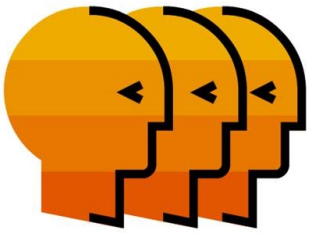


Figure 125: Properties of HDI Roles Created via Design Time Artifacts

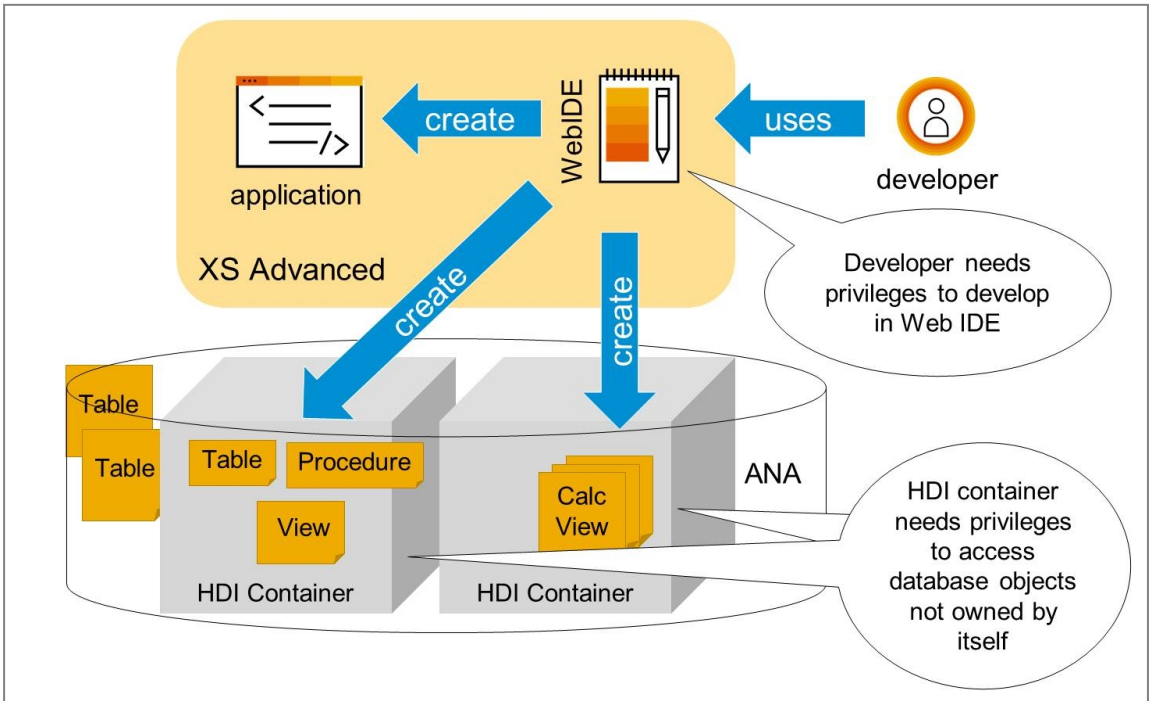


Figure 126: Developer Scenarios XS Advanced

Creating HDI roles is a software development activity.

The author needs to have development authorizations.

The Object Owner (`#OO` user) of the HDI container needs additional privileges to access any external objects outside of the container which they do not own. The next lesson will show how this works in detail.

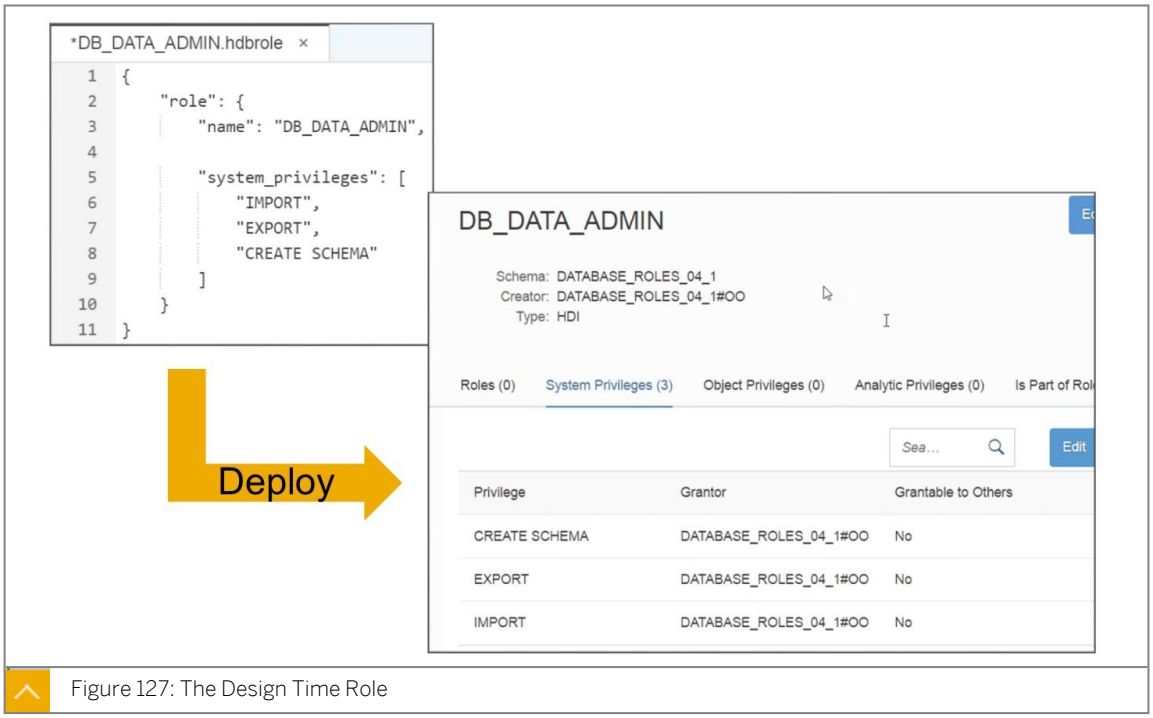


Figure 127: The Design Time Role

The design time role is a text file with the `.hdbrole` extension which has a JSON type structure.

The file content describes the role name and the list of privileges to be included in the role.

When the runtime role has been generated during the deployment process, it can be granted to any database user which can then access the referenced objects.

LESSON SUMMARY

You should now be able to:

- Understand the HDI Security Concepts

Learning Assessment

1. Who is the Object Owner in SAP HANA?

   Choose the correct answer.

   - ☐ A The creator of the schema containing the object
   - ☐ B The user who created the object
   - ☐ C The technical user _SYS_REPO
   - ☐ D The administrator of the HDI Container group

2. Waldemar creates a plain database schema and grants Constanze the `CREATE ANY` privilege on that schema. Who can access the table `DUMMY` which Constanze creates in Waldemars schema?

   Choose the correct answers.

   - ☐ A Waldemar, because he own the schema
   - ☐ B Constanze, because she owns the table
   - ☐ C Every user having the`CREATE ANY` privilege
   - ☐ D Every user owning an object in Waldemars schema

3. Marta creates an HDI container `PORTEMONNAIE` and deploys it in space `COAT_POCKET`. Which user owns the created schema `PORTEMONNAIE`?

   Choose the correct answer.

   - ☐ A `Marta`, because she deployed the container.
   - ☐ B `Marta`, because she created the container.
   - ☐ C `Portemonnaie`, which was created during deployment.
   - ☐ D `Portemonnaie#OO`, which was created during the deployment.

4. Eva creates an HDI container `Franntz`. It only contains a view with the statement `SELECT BRICKS FROM MILANO.DUOMO`. Why does the deployment fail?

   Choose the correct answers.

   - ☐ A Schema references are not allowed in HDI containers.
   - ☐ B `Franntz#OO` wasn't granted privileges on `MILANO.DUOMO`.
   - ☐ C `EVA` is not the object owner of `MILANO.DUOMO`.
   - ☐ D Tables need to be consumed via calculation views.

5. How can you access the schema `WALLDORF` of HDI container `WALLDORF`?

   Choose the correct answers.

   - ☐ A By assigning your user the role `WALLDORF::access_role`.
   - ☐ B By using the credentials of user `WALLDORF#OO`.
   - ☐ C By creating a service key and using the generated `WALLDORF_..._DT` user.
   - ☐ D By creating a service binding to the deploy service and using the generated credentials.

Learning Assessment - Answers

1. Who is the Object Owner in SAP HANA?

   Choose the correct answer.

   - [ ] A The creator of the schema containing the object
   - [x] B The user who created the object
   - [ ] C The technical user _SYS_REPO
   - [ ] D The administrator of the HDI Container group

   This is correct! The Object Owner is the user who created the object.

2. Waldemar creates a plain database schema and grants Constanze the `CREATE ANY` privilege on that schema. Who can access the table `DUMMY` which Constanze creates in Waldemars schema?

   Choose the correct answers.

   - [x] A Waldemar, because he own the schema
   - [x] B Constanze, because she owns the table
   - [ ] C Every user having the`CREATE ANY` privilege
   - [ ] D Every user owning an object in Waldemars schema

   This is correct! Waldemar can access, since he own the schema. Constanze can access too, since she owns the table.

3. Marta creates an HDI container `PORTEMONNAIE` and deploys it in space `COAT_POCKET`. Which user owns the created schema `PORTEMONNAIE`?

   Choose the correct answer.

   - [ ] A `Marta`, because she deployed the container.
   - [ ] B `Marta`, because she created the container.
   - [x] C `Portemonnaie`, which was created during deployment.
   - [ ] D `Portemonnaie#OO`, which was created during the deployment.

   That is correct! User `Portemonnaie` owns the created schema.

4. Eva creates an HDI container `Franntz`. It only contains a view with the statement `SELECT BRICKS FROM MILANO.DUOMO`. Why does the deployment fail?

   Choose the correct answers.

   - [X] A Schema references are not allowed in HDI containers.
   - [ ] B `Franntz#OO` wasn't granted privileges on `MILANO.DUOMO`.
   - [ ] C `EVA` is not the object owner of `MILANO.DUOMO`.
   - [ ] D Tables need to be consumed via calculation views.

   That is correct! Schema references are not allowed in HDI containers.

5. How can you access the schema `WALLDORF` of HDI container `WALLDORF`?

   Choose the correct answers.

   - [X] A By assigning your user the role `WALLDORF::access_role`.
   - [ ] B By using the credentials of user `WALLDORF#OO`.
   - [ ] C By creating a service key and using the generated `WALLDORF_..._DT` user.
   - [ ] D By creating a service binding to the deploy service and using the generated credentials.

   That is correct! You access the schema `WALLDORF` of HDI container `WALLDORF` by assigning your user the role `WALLDORF::access_role`.

UNIT 7

Accessing Database Objects Across Schemas and Containers

UNIT OBJECTIVES

- Describe how to access a remote SAP HANA schema

Accessing a Remote SAP HANA Schema

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe how to access a remote SAP HANA schema

Accessing a Remote SAP HANA Schema

Accessing External Objects

Assign access privileges

- .hdbgrants defines SELECT ON ERP.Y grant (with Grant Option) to the Object Owner
- A User-Provided-Service (UPS) contains credentials for a granting user (ERP_GRANTOR) → Granting user needs privileges with grant option
- Grant is executed during deployment using the User-Provided-Service, assigning privileges to Object Owner

Synonyms, referencing external objects

- Synonyms (.hdbsynonym) point to external objects
- Container Objects can access external object through synonym

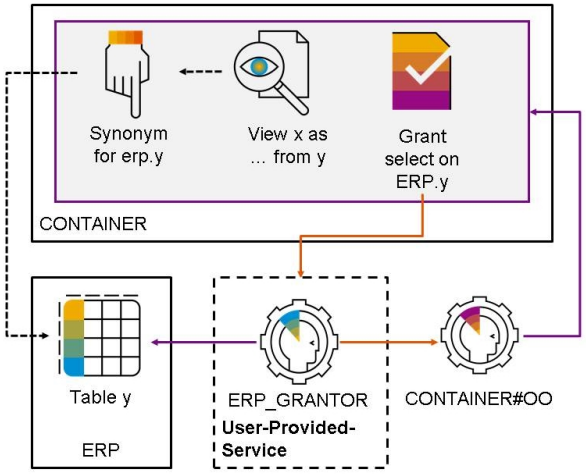


Figure 128: Accessing External Objects

Breaking the Isolation

- To make a remote object, for example, a database table, accessible within your SAP HANA database module, you need to:
  - Identify/create a user (granting user) which has the necessary privileges for the external object and has the ability to grant these privileges (Grant Option).
  - In XS Advanced, create a user-provided service which contains the granting user's credentials.
  - In your MTA development project, add the user-provided service as a required resource in the mta.yaml file.
  - In your MTA development project's SAP HANA database module, create a .hdbgrants file that triggers the privilege assignment to the Object Owner of the container.

- In your SAP HANA database module, create a synonym for every required remote object, for example, tables, views, and so on.
- In your SAP HANA database module, create a view that refers to the synonym.

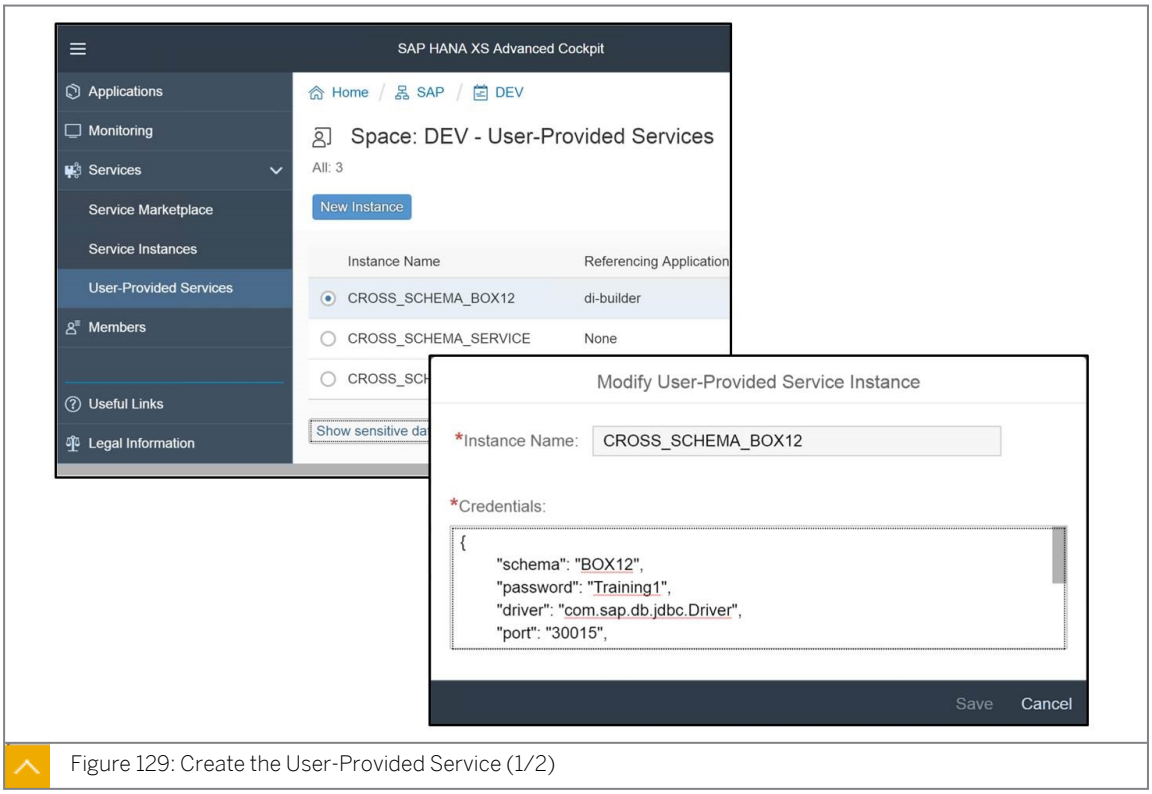


Figure 129: Create the User-Provided Service (1/2)

In the XS Advanced Cockpit, you can create a user-provided service that accesses the remote database and schema via the driver named `com.sap.db.jdbc.Driver`.

Provide the required connection information, for example, `driver, host, port, schema, user, password`, in the *Credentials* field.

```
{
        "driver": "com.sap.db.jdbc.Driver",
        "host": "localhost",
        "port": "30015",
        "schema": "BOX12",
        "user": "STUDENT12",
        "password": "Training1",
        "tags": ["hana"]
}
```

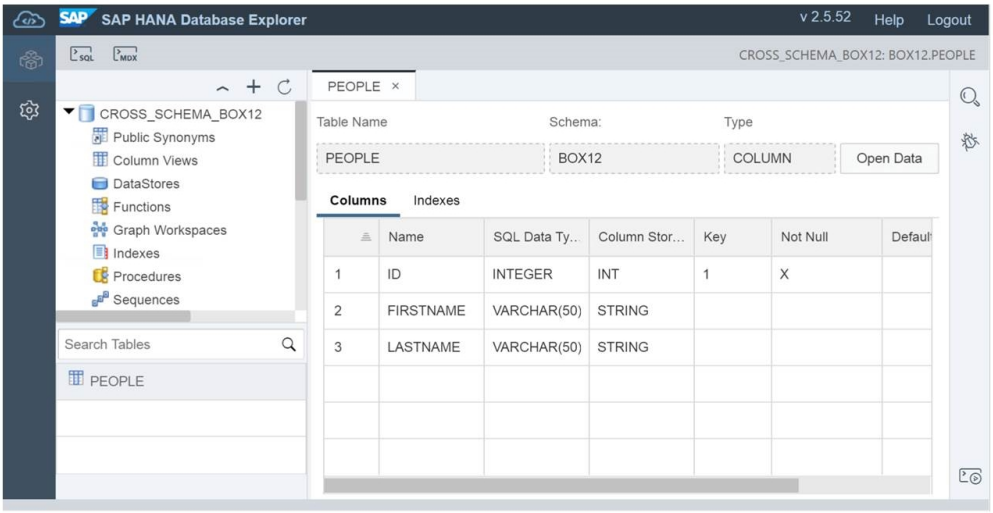


Figure 130: Create the User-Provided Service (2/2)

To verify that the service is working properly, you can access it in the HDI containers list in the SAP HANA Database Explorer and browse the database objects in the remote system.

```
modules:
 - name: db
   type: hdb
   path: db
   requires:
    [...]
    - name: cross-container-service-1
      group: SERVICE_REPLACEMENTS
      properties:
        key: ServiceName_1
        service: ~{the-service-name}

resources:
    [...]
 - name: cross-container-service-1
   parameters:
      service-name: CROSS_SCHEMA_BOX12
   properties:
      the-service-name: ${service-name}
   type: org.cloudfoundry.existing-service
```

Figure 131: Declare the Service in the mta.yaml File

To make the service available within your MTA project, you need to declare it within the mta.yaml file.

To use the service within your MTA module, you need to declare it as required in the module section of the mta.yaml file.

File *anyname.hdbgrants*

```
"CROSS_SCHEMA_BOX12": {
  "object_owner" : {
    "schema_privileges":[
      {
        "reference":"BOX12",
        "privileges_with_grant_option":["SELECT METADATA",
                        "SELECT CDS METADATA",
                        "SELECT"]
      }
    ]
  },
```

Figure 132: Grant Proper Authorizations for the Object Owner

The Object Owner (#OO user) of your module is the owner of any object in the HDI container of the module, nevertheless the Object Owner has no access to any object within other schemas.

Defining a .hdbgrants file within your project, triggers the assignment of the required additional authorizations to the Object Owner during the project deployment.

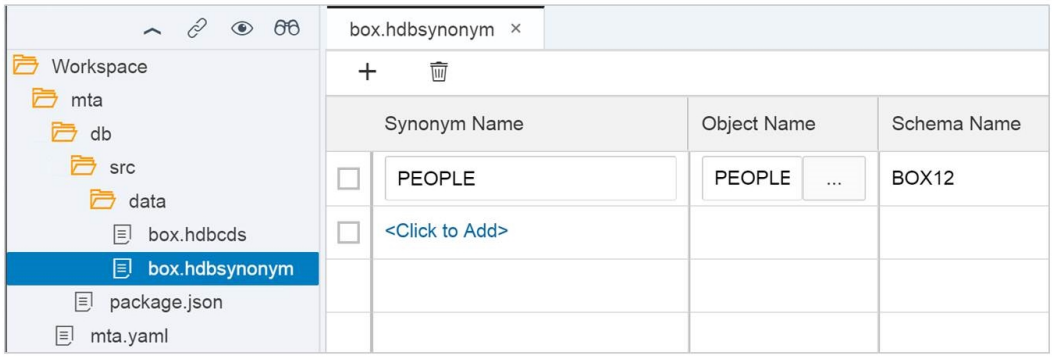


Figure 133: Create a Synonym for Every Required Remote Object

Create a synonym (.hdbsynonym file) for every object, for example, tables, views, and so on, to be accessed in the remote database schema.

```
using PEOPLE as P;

context box {
   define view peopleView as
      select from P {
      P.ID as id,
      P.FIRSTNAME as firstname,
      P.LASTNAME as lastname
      } ;
};
```

Figure 134: Create the Database View That Refers to the Synonym

Define a view using CDS. At the beginning of the .hdbcds file, include a "using" statement for any synonym that you are using in the file.

LESSON SUMMARY

You should now be able to:

- Describe how to access a remote SAP HANA schema

Learning Assessment

1. You are developing the persistency layer for a multi-target application. You want to create a database view that reads data from a table in a remote SAP HANA schema. What do you need to use?

   Choose the correct answer.

   - [ ] A A remote desktop connection
   - [ ] B A user-defined service
   - [ ] C An OData service
   - [ ] D A schema synonym

Learning Assessment - Answers

1. You are developing the persistency layer for a multi-target application. You want to create a database view that reads data from a table in a remote SAP HANA schema. What do you need to use?

   Choose the correct answer.

   - [ ] A A remote desktop connection
   - [x] B A user-defined service
   - [ ] C An OData service
   - [ ] D A schema synonym

   That is correct! You need to use a user-defined service.

UNIT 8

Accessing Database Objects from the Node.js Application

UNIT OBJECTIVES

- Run SQL in the Database with Node.js

Running SQL in the Database with Node.js

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Run SQL in the Database with Node.js

Running SQL in the Database with Node.js

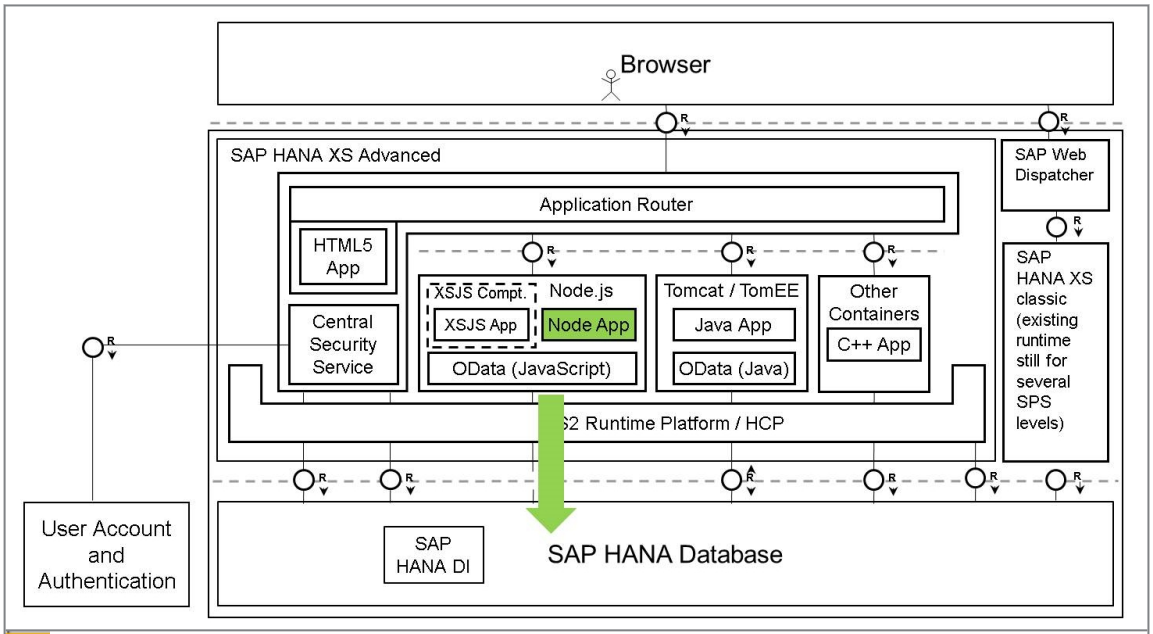


Figure 135: Connecting to the Database

As of SAP HANA 2.0 SP01, XS Advanced contains two Node.js runtimes: Version 4 and 6. The version of all runtimes can be looked up, using the following xs command: xs runtimes.

Process Web Requests with Express

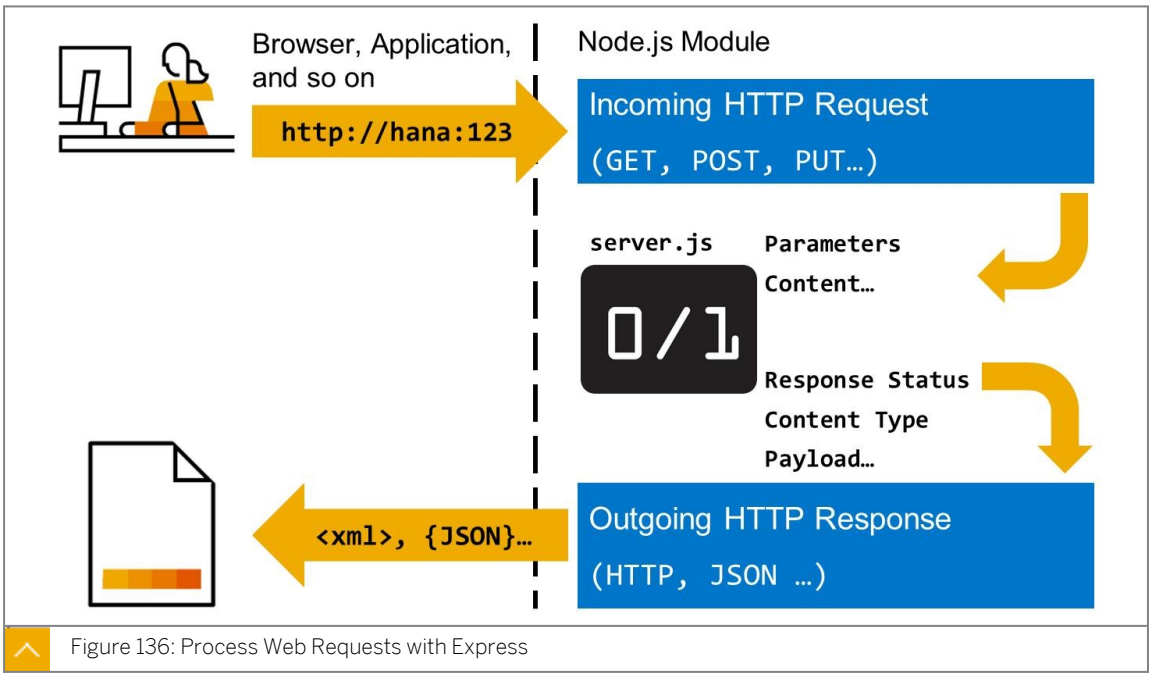


Figure 136: Process Web Requests with Express

Requests to a node service occur via the HTTP protocol. An application such as an SAPUI5 front-end can initiate these requests.

The node module receives the incoming HTTP request via the application router and executes a defined method, depending on the request method and registered application path.

The program can access the information sent in the HTTP request. For example, the parameters, and the request content.

During the server program execution, the HTTP response is prepared. It is then sent back to the requester once the program execution terminates.

Back on the client side, the requester can further process the HTTP response.

Connecting to the Database

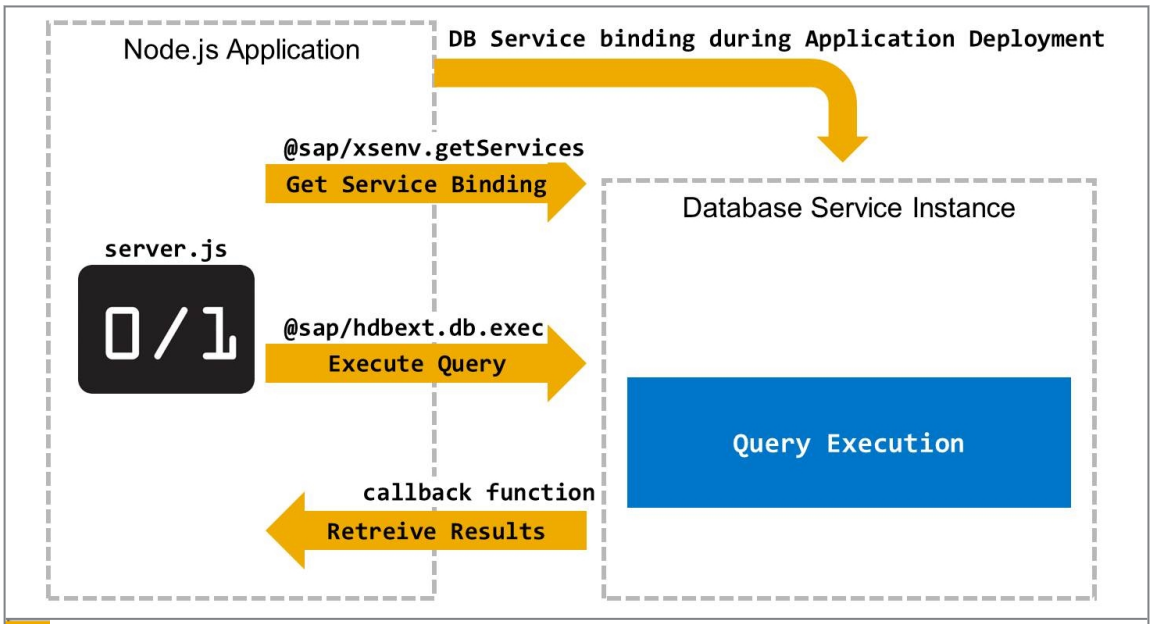


Figure 137: Connecting to the Database

To access SAP HANA database content, such as tables, procedures, and views, from a Node.js module, the *@sap/hdbext* module is used.

During the application deployment, the information in the Application Deployment Descriptor (mtad.yaml) is used to bind the database service instance to the Node.js module.

The binding information is available in the environment variables of the application. The module *@sap/xsenv* is used to retrieve the service instance from the environment and passed on to the module *sap-hdbext*.

The module, *@sap/hdbext*, sends the query to the database and provides a callback function to retrieve the results.

Consume XS Advanced Services

xsenv.getServices(query)

```
var xsenv = require("@sap/xsenv");
var services = xsenv.getServices({
        hana: {tag: "hana"}
});
```

Query Value	Description
{string}	Matches the service with the same service instance name (name property). Same as { name: '<string>' }.
{object}	All properties of the given object should match corresponding service instance properties as they appear in VCAP_SERVICES
{function}	A function that takes a service object as argument and returns true only if it is considered a match

Property	Description
name	The name of the service instance; the name you use to bind the service
label	Service name - the name shown by the command: "xs marketplace"
tag	Should match any of the service tags
plan	The name of the service instance plan specified in the command: "xs create-service"

Figure 138: Consume XS Advanced Services

SXS Advanced provides a convenient package (*@sap/xsenv*) for Node.js applications, which can be used to read bound services. To use the *@sap/xsenv* package, you have to add it to your application's dependencies, which are specified in a corresponding package.json file.

Created Service Bindings

```
xs env <applicationName>
```


```
"VCAP_APPLICATION" : {
  "start" : "2017-04-08 13:23:39 +0000",
  "application_id" : "d87ddf41-923a-4f42-8dcc-01859289b3c3",
  "instance_id" : "d87ddf41-923a-4f42-8dcc-01859289b3c3",
  "space_id" : "f0d533a2-aa1a-44fc-b18f-5d331314d22c",
  "application_name" : "WORKSHOP_22-9s3oyr0a1y6kr17r-NodeExercises-node",
  "organization_name" : "SAP",
  "space_name" : "DEV",
  "started_at_timestamp" : "1491657819850",
  "started_at" : "2017-04-08 13:23:39 +0000",
  "state_timestamp" : "1491582545409",
  "full_application_uris" : [ "https://wdflbmt7215:51042" ],
  "application_uris" : [ "wdflbmt7215:51042" ],
  "uris" : [ "wdflbmt7215:51042" ],
  "version" : "ba87b1ea-d802-4d65-bfc5-cd360e6ba88d",
  "application_version" : "ba87b1ea-d802-4d65-bfc5-cd360e6ba88d"
},
"VCAP_SERVICES" : {
  "hana" : [ {
    "name" : "WORKSHOP_22-9s3oyr0a1y6kr17r-NodeExercises-hdi-container",
    "label" : "hana",
    "tags" : [ "hana", "database", "relational" ],
    "plan" : "hdi-shared",
    "credentials" : {
      "schema" : "AX5NDGV8VFWURTZF_NODEEXERCISES_HDI_CONTAINER",
      "hdi_password" : "Aa_9886415090703918701901245091162290667106616331268209839911976390 2",
      "password" : "Aa_1898580113515010596028802613192171310651489609971337012590053404 3",
      "driver" : "com.sap.db.jdbc.Driver",
      "port" : "30015",
      "host" : "wdflbmt7215",
      "db_hosts" : [ {
        "port" : 30015,
        "host" : "wdflbmt7215"
      } ],
      "hdi_user" : "SBSS_3146812504268378378774408360067547590232517339421479794430698009 3",
      "user" : "SBSS_9239385546000130001272951017363024455974646978803872141314522360 6",
      "url" : "jdbc:sap://wdflbmt7215:30015/?currentschema=AX5NDGV8VFWURTZF_NODEEXERCISES_HDI_CONTAINER"
    }
  } ]
}
```


Figure 139: Created Service Bindings

Service bindings for an application can be retrieved, using the command `XS ENV <APPLICATIONNAME>`.

The attributes of a bound service can be used in a service query.

Connecting to the Database

- @sap/hdbext provides an Express middleware which allows easy access to a connection pool in an Express based application.
- In the background a connection pool is created.
- The connection is automatically returned to the pool when the express request is closed or finished.

```
var hdbext = require("@sap/hdbext");
app.use("/", hdbext.middleware(services.hana));
app.get("/", function (req, res, next) {
  req.db.exec("SELECT COUNT(EMPLOYEEID) AS COUNT FROM \"MD.Employees\"",
  function (err, rows) {
    if (err) { return next(err); }
    res.send("Current employees: " + rows[0].COUNT);
  });
});
```

Figure 140: Connecting to the Database

@sap/hdbext is a small Node.js package, which extends the functionality of the *hdb* package. hdb is a JavaScript client for Node.js, which implements the SAP HANA database SQL command network protocol.

hdbext.middleware connects to SAP HANA automatically on each access to the specified path, – `/` in this case.

Afterwards the connection is available in req.db. This is the client object of SAP HANA database driver. The connection is closed automatically at the end of the request.

The result set is passed to the callback function and available in the rows array. In the example shown in the figure above, the first record is sent back to the client via the response object.

LESSON SUMMARY

You should now be able to:

- Run SQL in the Database with Node.js

Learning Assessment

1. Which Node.js module provides a database connection?

   Choose the correct answer.

   - [ ] A @sap/xsenv
   - [ ] B express
   - [ ] C jest
   - [ ] D @sap/hdbext

Learning Assessment - Answers

1. Which Node.js module provides a database connection?

   Choose the correct answer.

   - [ ] A @sap/xsenv
   - [ ] B express
   - [ ] C jest
   - [x] D @sap/hdbext

   That is correct! *@sap/hdbext* provides an express middleware which allows access to a database connection pool.

UNIT 9 Exposing Data as OData Services

UNIT OBJECTIVES

- Describe basic concepts about OData services
- Create a simple OData service, using XSODATA to expose a single Entity Set
- Describe the use of keys and associations in OData services, created with XSODATA

Unit 9
Lesson 1

Introducing OData Services

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe basic concepts about OData services

Introducing OData Services

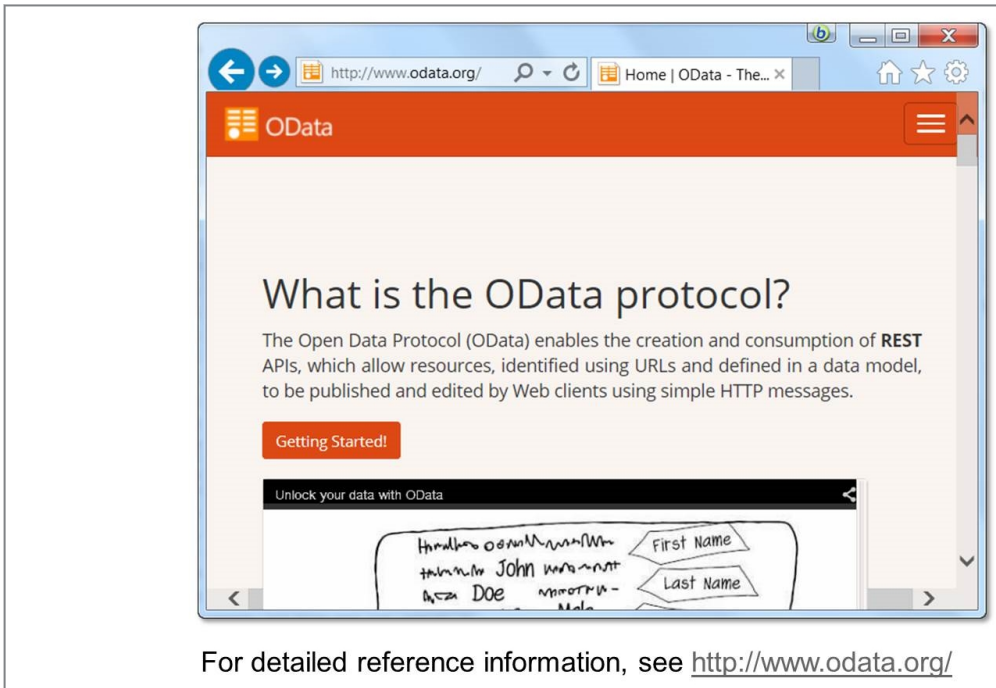


Figure 141: OData Protocol

Open Data Protocol (OData) is an OASIS standard that defines the best practice for building and consuming RESTful APIs. OData helps you to focus on your business logic, while building RESTful APIs, without having to worry about the approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats, query options, and so on. It is an open standard, defined by the OASIS consortium.

OData also guides you in tracking changes, defining functions or actions for reusable procedures, sending asynchronous or batch requests, and so on. Additionally, OData provides a facility for extension to fulfill any custom needs of your RESTful APIs.

OData RESTful APIs are easy to consume. The OData metadata, a machine-readable description of the data model of the APIs, enables the creation of powerful generic client proxies and tools. Some of these can help you interact with OData, even without knowing anything about the protocol.

OData Basics

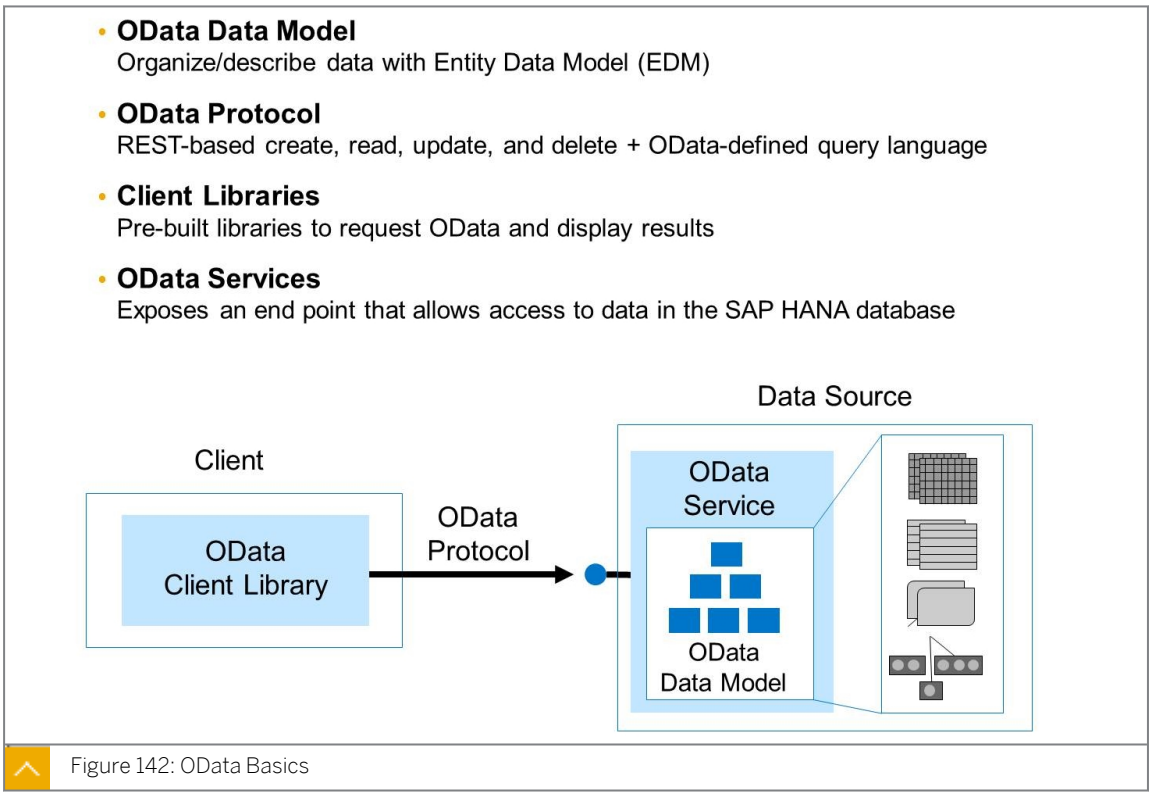


Figure 142: OData Basics

OData is a resource-based web protocol for querying and updating data. OData defines operations on resources using HTTP commands (for example, GET, PUT, POST, and DELETE) and specifies the uniform resource indicator (URI) syntax to use to identify the resources.

Data is transferred over HTTP using the Atom or JSON format.

The main aim of OData is to define an abstract data model and a protocol, which, combined, enable any client to access data exposed by any data source. Clients might include web browsers, mobile devices, business-intelligence tools, and custom applications, for example, custom applications written in programming languages such as PHP or Java. Data sources can include databases, content-management systems, the Cloud, or custom applications, for example, custom applications written in Java.

The OData approach to data exchange involves the following elements:

- OData data model

  Provides a generic way to organize and describe data. OData uses the Entity 1 Data Model (EDM).

- OData protocol

  Enables a client to query an OData service. The OData protocol is a set of interactions, which includes the usual REST-based create, read, update, and delete operations, along with an OData-defined query language. The OData service sends data in either of the following ways:

  - XML-based format defined by Atom/AtomPub
  - JSON

- OData client libraries

  Enable access to data via the OData protocol. Since most OData clients are applications, pre-built libraries for making OData requests and getting results reduces and simplifies work for the developers who create those applications.

  A broad selection of OData client libraries are already widely available, for example: Android, Java, JavaScript, PHP, Ruby, and the best known mobile platforms.

- OData services

  Expose an end point that allows access to data in the SAP HANA database. The OData service implements the OData protocol (using the OData Data Services runtime) and uses the Data Access layer to map data between its underlying form (database tables, spreadsheet lists, and so on) and a format that the requesting client can understand.

Introducing OData Services in SAP HANA

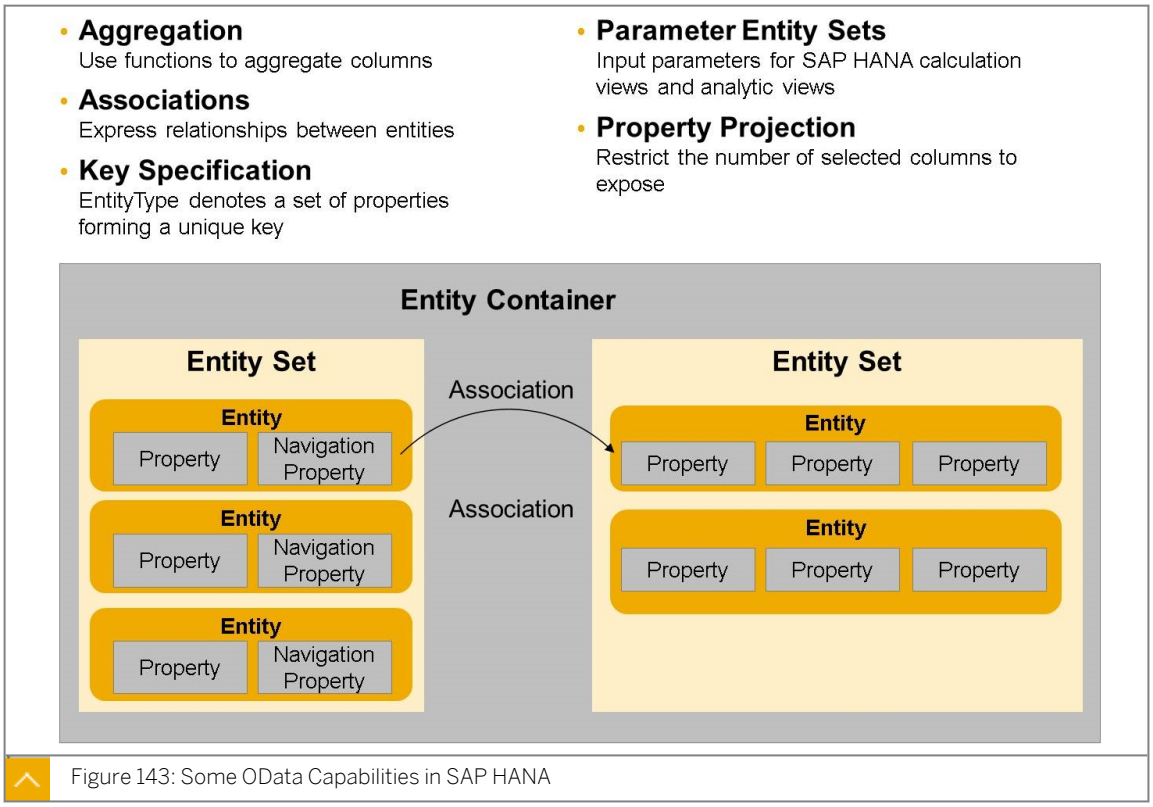


Figure 143: Some OData Capabilities in SAP HANA

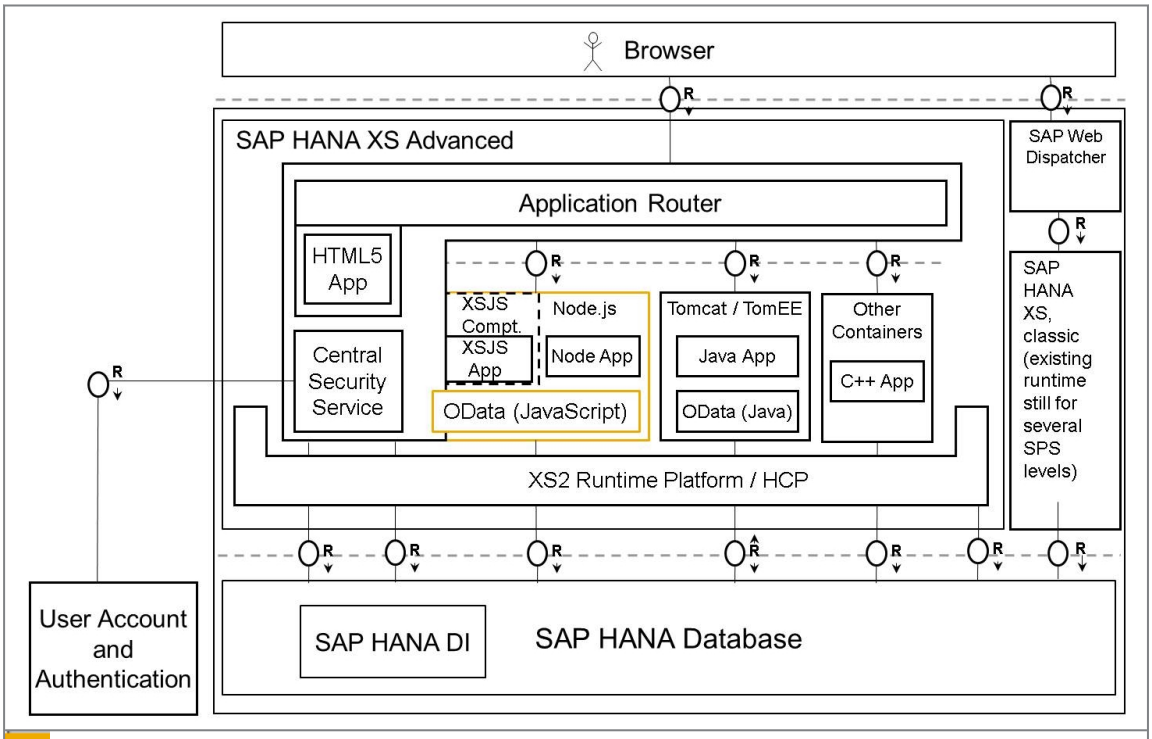


Figure 144: OData on XS Advanced: Architecture

To expose database information via OData in XS Advanced, you have to use the Node.js module of the MTA.

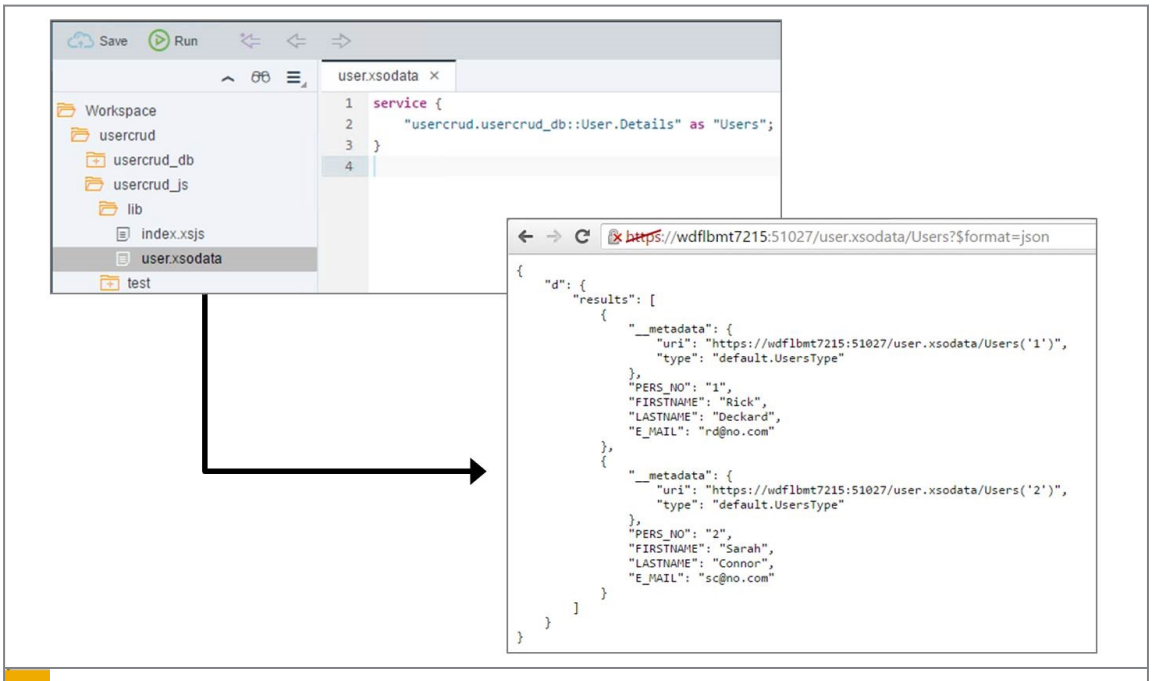


Figure 145: Define an OData Service Using XSODATA

An OData service is defined in a text file, with the file extension .xsodata, for example, user.xsodata. It must contain at least the entry `service {}`, which generates an OData service with an empty service catalog and an empty metadata file.

In the example above, the table, *usercrud.usercrud_db::User.Details*, is directly exposed and accessible via OData.

Note the appended format url parameter to request the JSON format, instead of the ATOM (XML) format.

By default, the created OData service is write-enabled, so a user with proper authorizations on the destination table can use it to create, change, or delete table lines.

It is possible to forbid data modification via explicit options:

```
service { "sap.test::myTable"
    create forbidden
    update forbidden
    delete forbidden;
}
```

Figure 146: Read-Write

Custom Exits

Custom exits can be created to complete, override, default create, update, and delete service routines.

There are two types of exits:

- Validation Exits

  These exits are for validation of input data and data consistency checks. They can be executed before or after the change operation, or before or after the commit operation.

- Modification Exits

  You can define custom logic to create, update, or delete an entry in an entity set. If a modification exit is specified, it is executed instead of the generic actions provided by the OData infrastructure.

LESSON SUMMARY

You should now be able to:

- Describe basic concepts about OData services

Exposing an OData Entity Set with XSODATA

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create a simple OData service, using XSODATA to expose a single Entity Set

Exposing an OData Entity Set with XSODATA

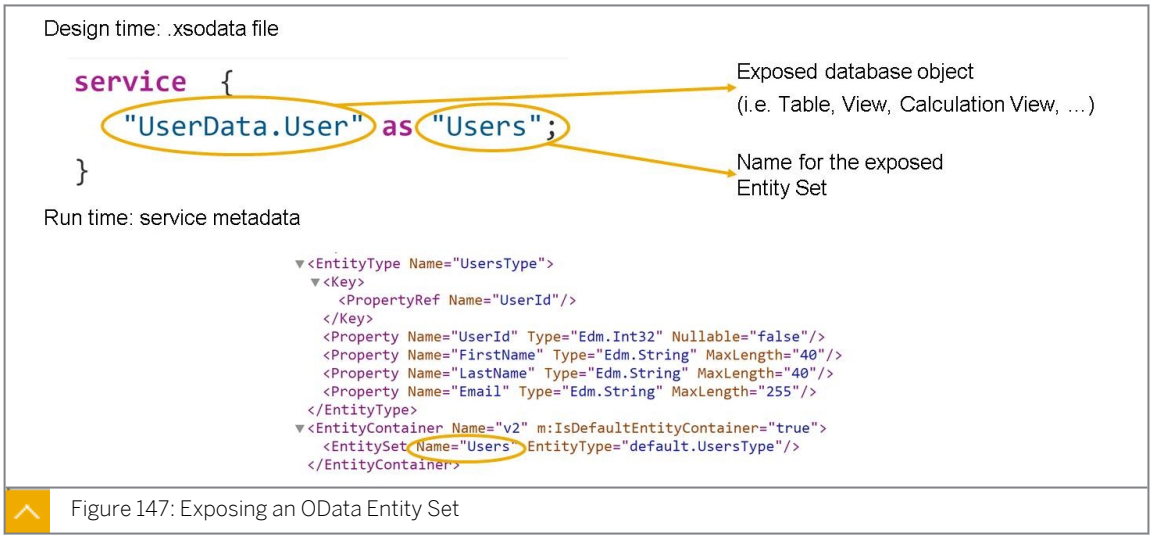


Figure 147: Exposing an OData Entity Set

At design time, within a node.js module, you create an .xsodata file where you indicate the exposed database object, for example, table, view, calculation view, and so on, and the name of the Entity Set in the service.

On build, the application provides an OData service, exposing the object content under the given entity set name.

LESSON SUMMARY

You should now be able to:

- Create a simple OData service, using XSODATA to expose a single Entity Set

Unit 9
Lesson 3

Using OData Key and Association in XSODATA

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the use of keys and associations in OData services, created with XSODATA

OData Key Specification in XSODATA

```
service {
    "sample.odata::view"
        as "MyView" key ("ID","Text");
}
```

Figure 148: OData Key Specification

The OData specification requires an EntityType to denote a set properties forming a unique key.

In SAP HANA, only tables can have a unique key; the primary key. For all other (mostly views) objects you need to specify a key for the entity.

OData Association in XSODATA

```
service {
    "sample.odata::customer" as "Customers";
    "sample.odata::order" as "Orders";

    association "Customer_Orders"
        with referential constraint
        principal "Customers"("ID") multiplicity "1"
        dependent "Orders"("CustomerID") multiplicity "*";
}
```

Figure 149: OData Association

The definition of an association requires you to specify a name, which references two exposed entities and whose columns keep the relationship information. To distinguish the ends of the association, you must use the keywords: principal and dependent. In addition, it is necessary to denote the multiplicity for each end of the association.

The association in the example above with the name Customer_Orders defines a relationship between the table customer, identified by its EntitySet name Customers, on the principal end, and the table order, identified by its entity set name Orders, on the dependent end. Involved columns of both tables are denoted in braces ({}) after the name of the corresponding entity set. The multiplicity keyword on each end of the association specifies their cardinality, in this example, one-to-many.

The `with referential constraint` syntax ensures that the referential constraint check is enforced at design time, for example, when you activate the service definition in the SAP HANA repository. The referential constraint information appears in the metadata document.

Navigation Properties

```
service {
    "sample.odata::customer" as "Customers"
        navigates ("Customer_Orders" as "HisOrders");

    "sample.odata::order" as "Orders";

    association "Customer_Orders"
        principal "Customers"("ID") multiplicity "1"
        dependent "Orders"("CustomerID") multiplicity "*";
}
```

Figure 150: Navigation Properties

By only defining an association, it is not possible to navigate from one entity to another. Associations need to be bound to entities by a Navigation Property. You can create Navigation Properties by using the keyword *navigates*.

The example in the figure above says that it is possible to navigate from Customers over the association Customer_Order via the Navigation Property named *HisOrders*.

LESSON SUMMARY

You should now be able to:

- Describe the use of keys and associations in OData services, created with XSODATA

Learning Assessment

1. What is OData?

   Choose the correct answer.

   - ☐ **A** A software tool
   - ☐ **B** An SAP design guideline
   - ☐ **C** A protocol
   - ☐ **D** A document format

2. Which file extension do you use for the definition file for an OData service?

   Choose the correct answer.

   - ☐ **A** .data
   - ☐ **B** .odata
   - ☐ **C** .xsodata

3. Which keywords are used in an XSODATA association definition?

   Choose the correct answers.

   - ☐ **A** Dependent
   - ☐ **B** Key
   - ☐ **C** Principal
   - ☐ **D** Join

Learning Assessment - Answers

1. What is OData?

   Choose the correct answer.

   - [ ] A A software tool
   - [ ] B An SAP design guideline
   - [x] C A protocol
   - [ ] D A document format

   That is correct! OData is an ISO/IEC approved protocol, defined by the OASIS consortium.

2. Which file extension do you use for the definition file for an OData service?

   Choose the correct answer.

   - [ ] A .data
   - [ ] B .odata
   - [x] C .xsodata

   That is correct! The file extension that you need to use for the definition file for an OData service is .xsodata

3. Which keywords are used in an XSODATA association definition?

   Choose the correct answers.

   - [x] A Dependent
   - [ ] B Key
   - [x] C Principal
   - [ ] D Join

   That is correct! The keywords used in a XSODATA association definition are: principal and dependent.

UNIT 10 Integrating HTML5 Modules Using the Router

UNIT OBJECTIVES

- Create and run an HTML5 module saying Hello World
- Configure the Router to forward HTTP messages to the Node.js back end module
- Configure the Router to replace at runtime a placeholder within an html file

Unit 10
Lesson 1

Creating a Basic HTML5 Module

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create and run an HTML5 module saying Hello World

Introducing the HTML5 module

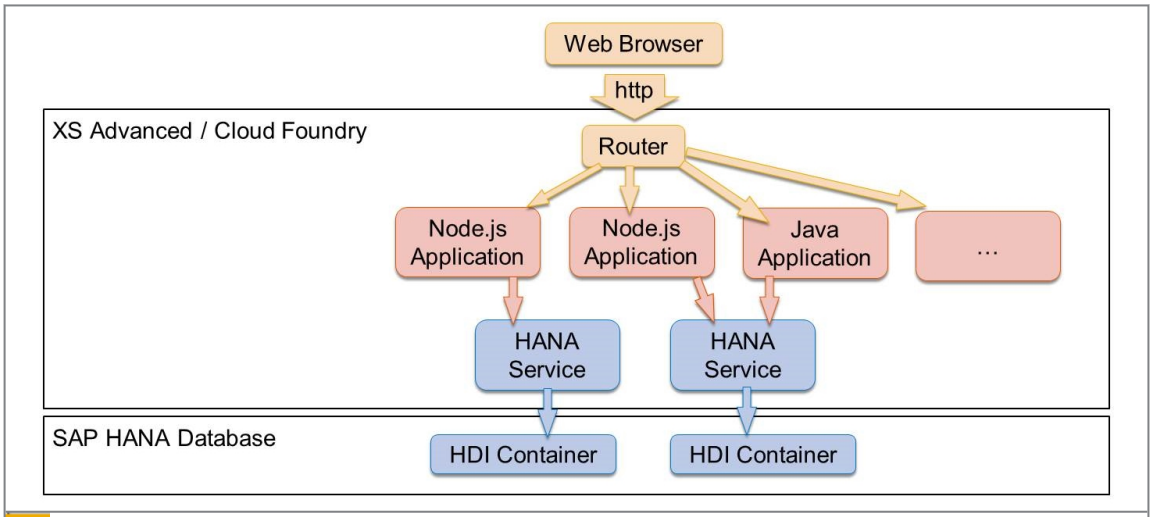


Figure 151: Runtime Structure of the Full-Stack Application

At runtime, the full-stack, MTA is commonly made in the following way:

1. The user interface is written using HTML5 and rendered and executed in a web browser.
2. The HTML5 program communicates with the router via the HTTP protocol. The router is a program running on the server. It does not execute business logic, instead it is just a dispatcher of HTTP messages.
3. The router dispatches the HTTP message to the proper application, running on the server. This application is built out of the corresponding Node.js (Java, Python, ...) module, coded by the developer within the MTA project in the SAP Web IDE for SAP HANA.
4. The Node.js application communicates with the database via a SAP HANA service, that is connected to an HDI container stored in the database.
5. Database objects, for example, tables, views, and procedures, are stored in the HDI container.
6. The procedures in the database use the SQLScript (alternatively the R) language, that is specialized in highly parallel and data intensive processing.

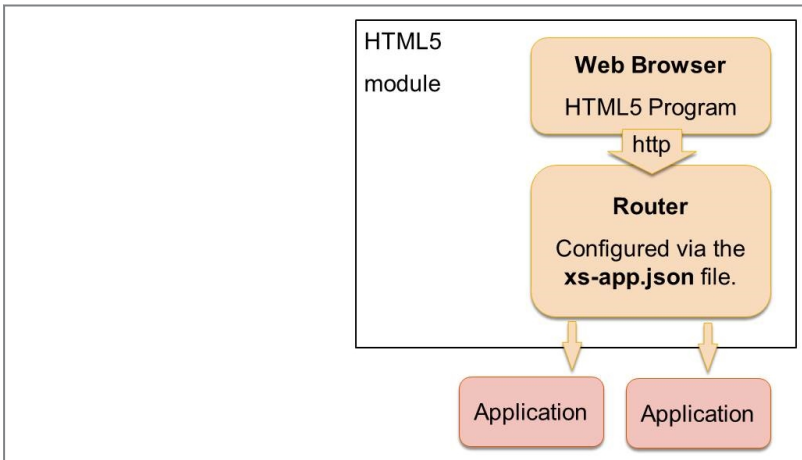


Figure 152: The HTML5 Module

At design time, an MTA project is created in the SAP Web IDE for SAP HANA.

The front-end of the MTA is defined using the HTML5 module.

This module is used to store:

- The HTML5 user interface
- The configuration file for the router xs-app.json

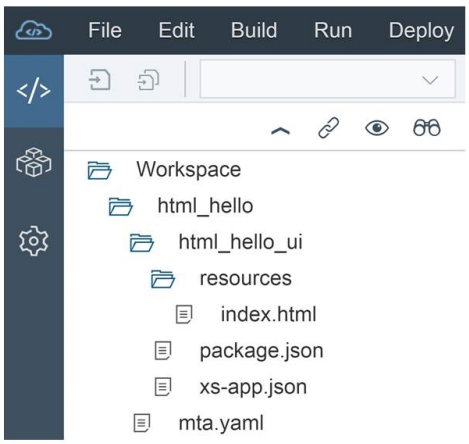


Figure 153: The HTML5 Module

A minimal HTML5 module contains:

- An index.html file: containing the static HTML code to be uploaded to the browser
- The xs-app.json file: configuring the router
- A minimal package.json file: for configuration, for example, to set the router version

LESSON SUMMARY

You should now be able to:

- Create and run an HTML5 module saying Hello World

Configuring the Router for HTTP Message Forwarding

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Configure the Router to forward HTTP messages to the Node.js back end module

Configuring the Router for HTTP message forwarding

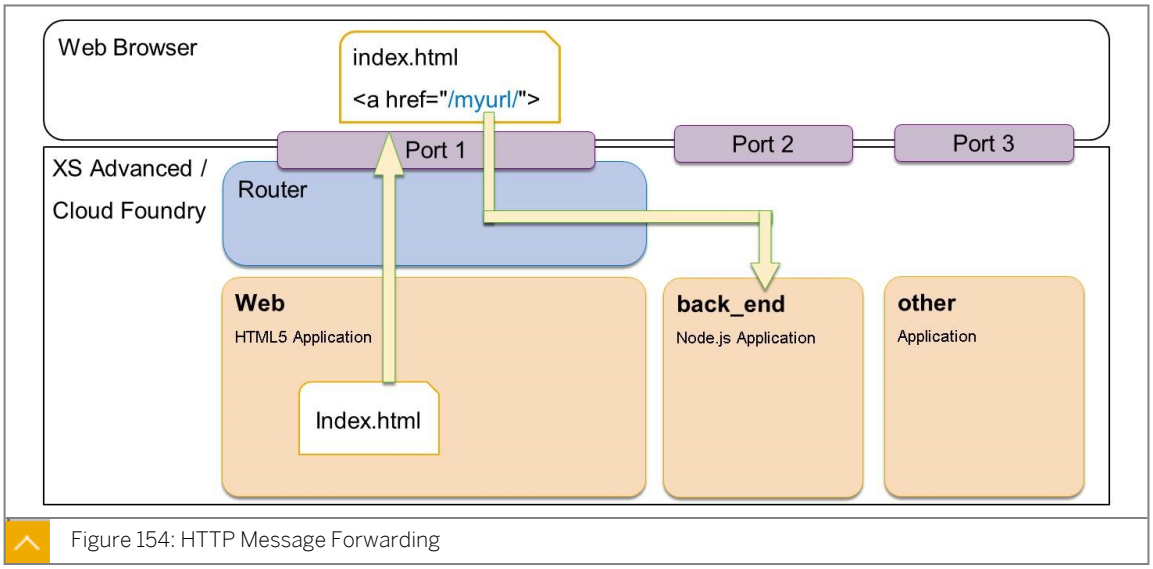


Figure 154: HTTP Message Forwarding

You create an MTA project in the SAP Web IDE for SAP HANA that is made of multiple modules. Typically, the UI is defined within an HTML5 module, while the back-end is a Node.js module.

When you build the modules, every one of them generates an XS Advanced/Cloud Foundry application, communicating initially via a specific HTTP port.

Within HTML5 modules, you have the possibility to configure the router, so that HTTP messages coming from the web browser are dispatched to other modules, for example, to the back-end module.

To execute such a configuration, you need to modify:

- The mta.yaml file, at multi-target application level
- The xs-app.json file, at HTML5 module level

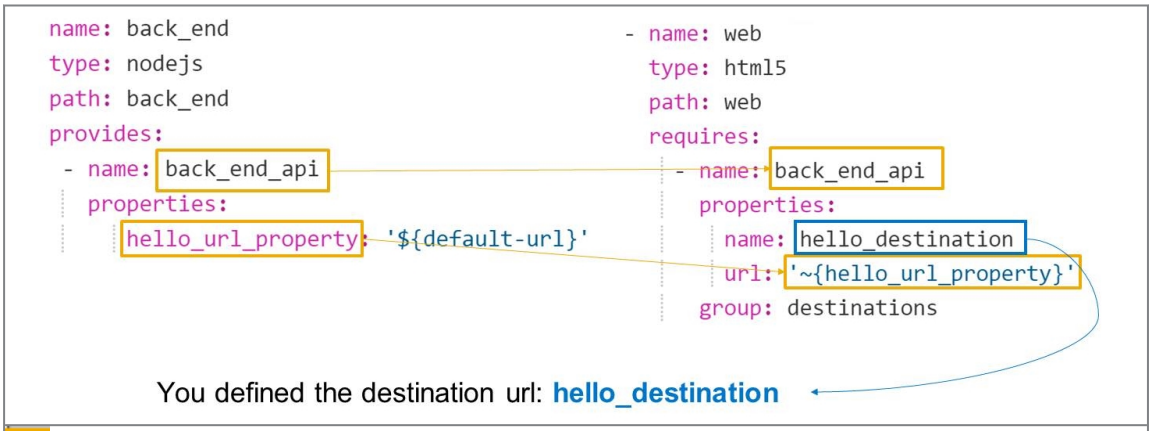


Figure 155: Define a Destination in the mta.yaml File

In the mta.yaml file, you define a destination containing the URL of the back_end module.

The URL is provided as a property, for example hello_url_property, of the back_end module.

On the HTML5 module side, a property of group destinations, that is, hello_destination, is created and valued using the first property, that is, "~{hello_url_property}".

Figure 156: Define a Route in the xs-app.json File

In the xs-app.json file, you define the route.

The route determines that any HTTP message corresponding to the source is forwarded to the given destination.

The source is a regular expression, that is, it can use special characters to identify families of different URLs.

The rules for regular expressions are taken from the JavaScript programming language.

You can find a tutorial at: https://www.w3schools.com/jsref/jsref_obj_regexp.asp

LESSON SUMMARY

You should now be able to:

- Configure the Router to forward HTTP messages to the Node.js back end module

Configuring the Router for Placeholders Replacement

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Configure the Router to replace at runtime a placeholder within an html file

Configuring the Router for placeholders replacement

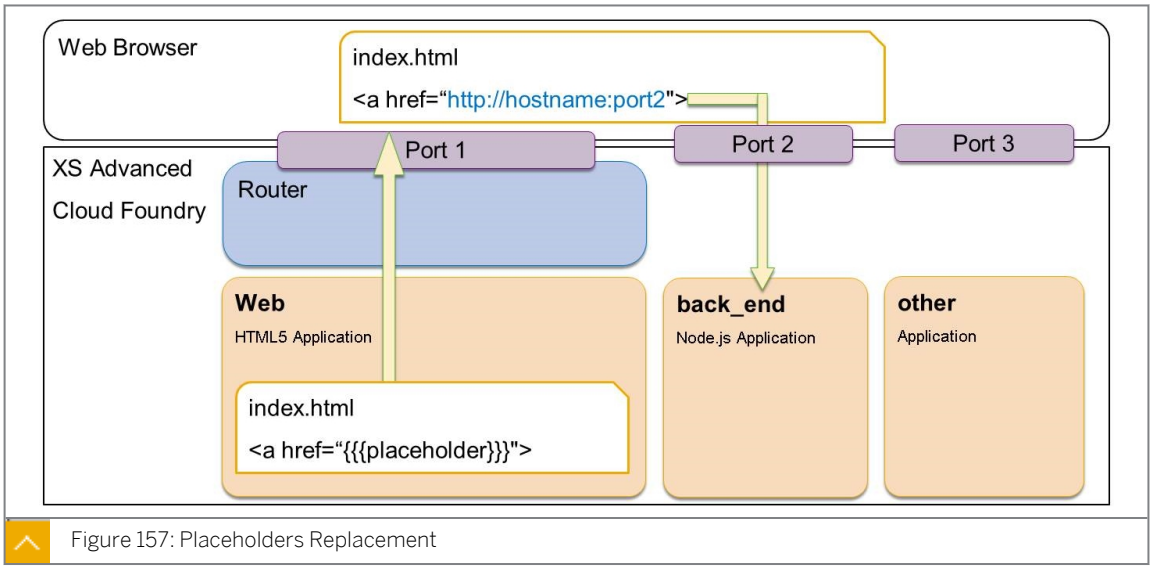


Figure 157: Placeholders Replacement

You create an MTA project, made of multiple modules, in the SAP Web IDE for SAP HANA.

When you build the modules, every one of them generates an XS Advanced/Cloud Foundry application, communicating initially via a specific HTTP port.

The port for every application is determined at deploy time, so it is not known by the developer.

Within HTML5 modules, you have the possibility to configure the router, so that HTML files contain placeholders that are replaced at runtime, just before the file is served to the web browser.

This way you can refer to the "actual" URLs of current applications or services within your HTML files.

To execute such a configuration, you need to modify:

- The mta.yaml file, at multi-target application level
- The xs-app.json file, at HTML5 module level

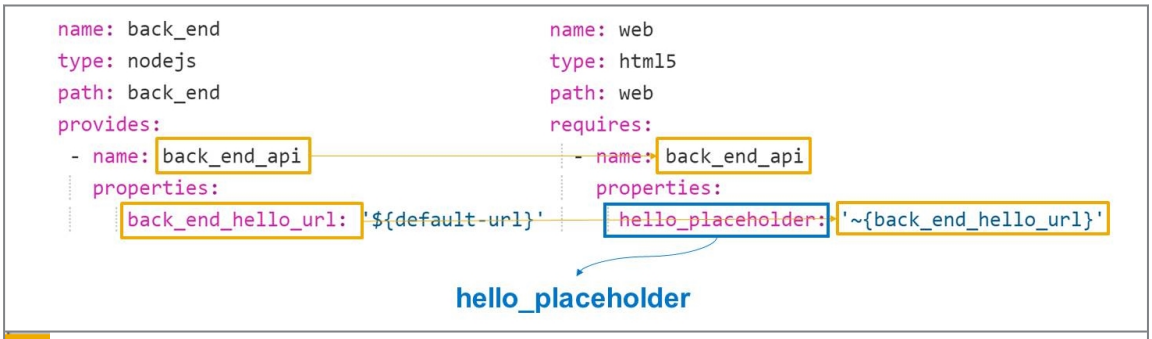


Figure 158: In the mta.yaml File

In the mta.yaml file, you define a property back_end_hello_url containing the URL of the back_end module.

In the web module, use the previous property to populate another property hello_placeholder.

The second property will be used for placeholder replacement in the index.html file.

Figure 159: Define a Route in the xs-app.json File

In the xs-app.json file, you define the route.

The route determines the following details of placeholder replacement:

- Source:

The source identifies the URLs that are subject to the replacement.

The source is a regular expression, that is, it can use special characters to identify families of different URLs.

The rules for regular expressions are taken from the JavaScript programming language.

You can find a tutorial at: https://www.w3schools.com/jsref/jsref_obj_regexp.asp

- localDir:

The local folder, within the HTML5 module that contains the files that are subject to replacement.

- replace.pathSuffixes:

Within the folder, only some files are subject to replacement, based on the suffixes indicated in this list.

- replace.vars:

The list of placeholders to be replaced.

Original code

```
<!DOCTYPE HTML>
<html>
<body>
<p>Router Placeholder Replacement</p>
<p><a href="{{{hello_placeholder}}}">Hello</a></p>
</body>
</html>
```

... at run time

```
<!DOCTYPE HTML>
<html>
<body>
<p>Router Placeholder Replacement</p>
<p><a href="https://wdflbmt7195.wdf.sap.corp:51040">Hello</a></p>
</body>
</html>
```

Figure 160: Enter Placeholders in the index.html File

Insert the placeholders in the HTML files.

At runtime, the router replaces the placeholders. Then, the file is served to the web browser.

LESSON SUMMARY

You should now be able to:

- Configure the Router to replace at runtime a placeholder within an html file

Learning Assessment

1. Which part of a full-stack application is defined within the HTML5 module?

   Choose the correct answers.

   - [ ] A The router configuration
   - [ ] B The HTTP services
   - [ ] C The security role templates
   - [ ] D The user interface

2. Which file contains route definitions?

   Choose the correct answer.

   - [ ] A mta.yaml
   - [ ] B package.json
   - [ ] C security.json
   - [ ] D xs-app.json

3. In your HTML page, you want to have a placeholder replaced at runtime with the actual URL of the back-end application. Which design time files do you modify to obtain this?

   Choose the correct answers.

   - [ ] A package.json
   - [ ] B security.json
   - [ ] C mta.yaml
   - [ ] D xs-app.json

Learning Assessment - Answers

1. Which part of a full-stack application is defined within the HTML5 module?

   Choose the correct answers.

   - [X] A The router configuration
   - [ ] B The HTTP services
   - [ ] C The security role templates
   - [X] D The user interface

   That is correct! The HTML5 module defines the user interface and the router configuration.

2. Which file contains route definitions?

   Choose the correct answer.

   - [ ] A mta.yaml
   - [ ] B package.json
   - [ ] C security.json
   - [X] D xs-app.json

   That is correct! Routes are defined in the xs-app.json file.

3. In your HTML page, you want to have a placeholder replaced at runtime with the actual URL of the back-end application. Which design time files do you modify to obtain this?

   Choose the correct answers.

   - [ ] A package.json
   - [ ] B security.json
   - [X] C mta.yaml
   - [X] D xs-app.json

   That is correct! You need to modify the mta.yaml and the xs-app.json files.

UNIT 11 Defining the Application Security

UNIT OBJECTIVES

- Describe basic concepts of application security in XS Advanced
- Create the user with authorization for development in the SAP Web IDE for SAP HANA
- Create the security concept within an HTML5 module

Unit 11
Lesson 1

Introducing Application Security in XS Advanced

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe basic concepts of application security in XS Advanced

Introducing application security in XS Advanced

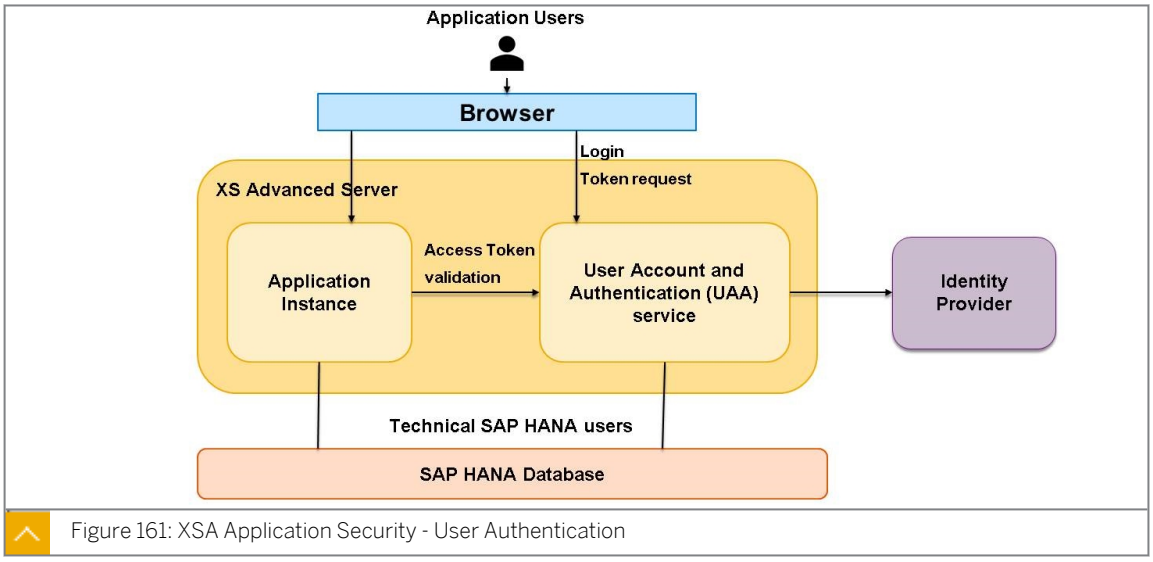


Figure 161: XSA Application Security - User Authentication

In XS Advanced, the central service for the management and authentication of users is the User Account and Authentication (UAA) service. (https://help.sap.com/viewer/6b94445c94ae495c83a19646e7c3fd56/2.0.04/en-US/7e87c8bc1bf64bfdb462b715f18ea05d.html)

The personal data used to identify the application users is kept in a user store like an Identity Provider (IdP) and the logon requests are managed by the XS Advanced User Account and Authentication service (UAA). Appropriate roles also need to be granted to the application users to define their permissions.

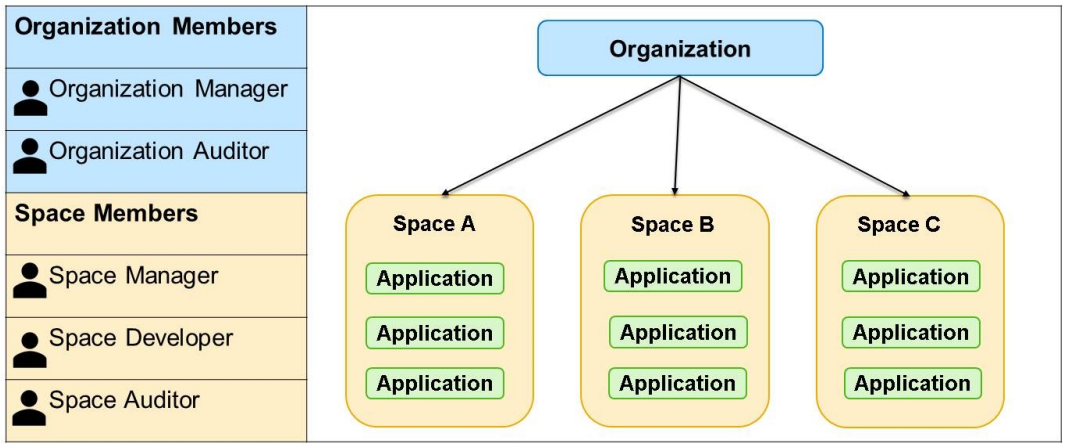


Figure 162: XS Advanced Development Areas

To group together the XS Advanced applications the followings hierarchical structures are used:

- Organizations
- Spaces

The users can be added to the organizations and to the spaces. This assigns the users predefined roles.

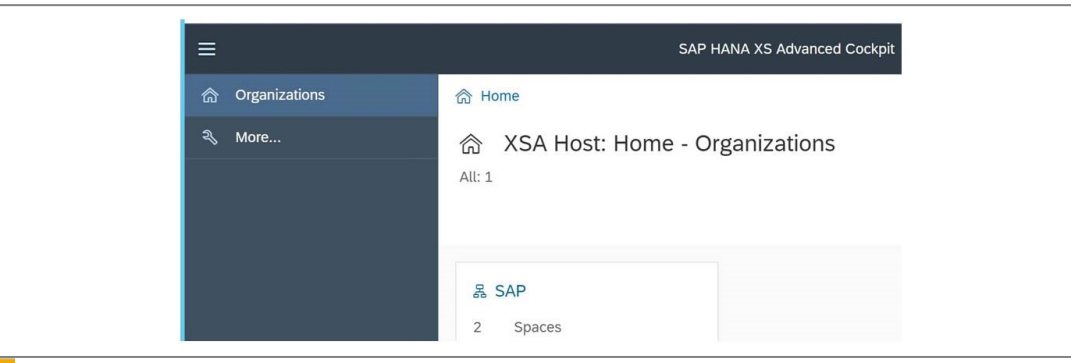


Figure 163: XS Advanced Organizations

An organization is a top-level account which logically groups together some spaces and to which information like application domains and server certificates are related. Organizations can reflect the organizational structure of a company and can be used to expose a group of applications with a specific domain name.

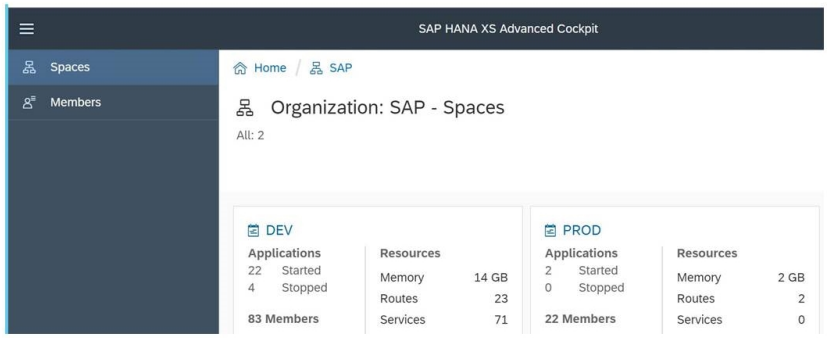


Figure 164: XS Advanced Spaces

Each XS Advanced application belongs to a specific space, which is also a trust zone.

Applications have to use resources created in their space. Application processes are also isolated at the space level.

Spaces can be used to separate the objects of different development teams or to separate applications currently in development from those already tested.

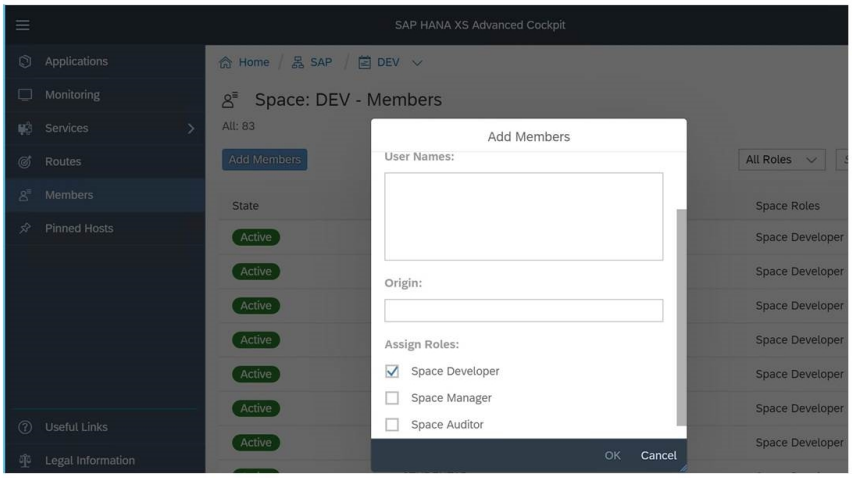


Figure 165: XS Advanced Space Members

Each user can be added to a space as a member by the manager of the space.

The manager also provides the user with their appropriate role for the space.

LESSON SUMMARY

You should now be able to:

- Describe basic concepts of application security in XS Advanced

Creating the User with Authorization for Development in SAP Web IDE for SAP HANA

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create the user with authorization for development in the SAP Web IDE for SAP HANA

Creating a user with authorization for development in the SAP Web IDE

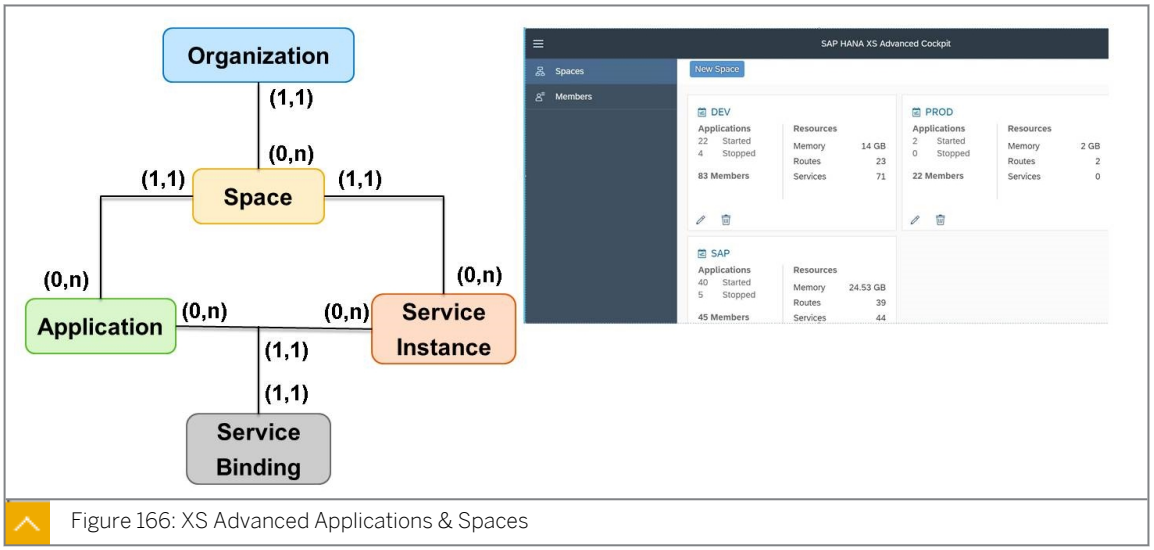


Figure 166: XS Advanced Applications & Spaces

Different XS Advanced applications can be coupled together to deploy them to the same space. Indeed, each application must be deployed to an existing space.

The applications belonging to the same space may share common resources like data storage, user authorizations, and passwords.

To achieve the isolation of the spaces, they have to be mapped to different OS users.

As previously mentioned, some spaces that are logically linked together can be grouped into an organization to be managed and administrated in a collective way.

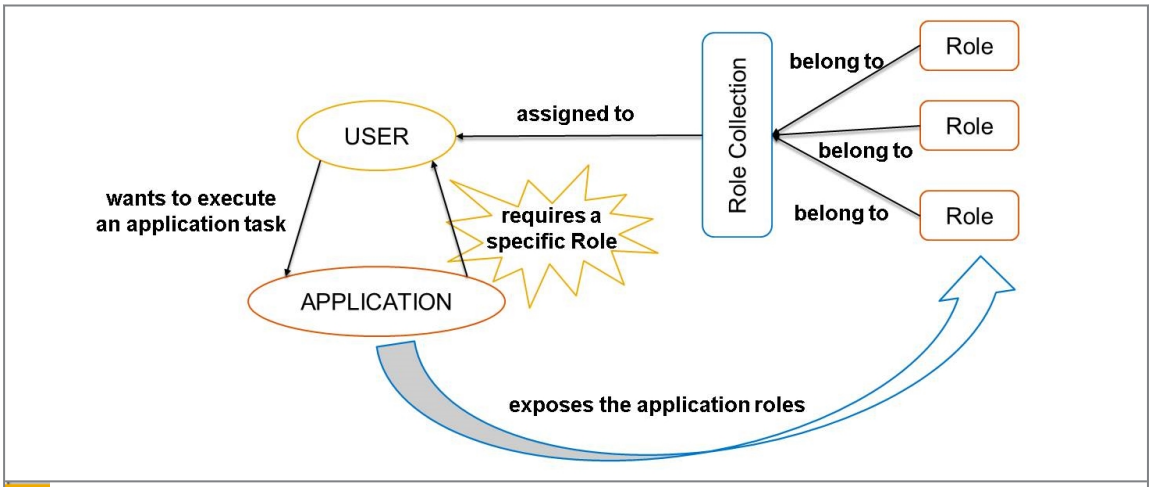


Figure 167: XS Advanced Application Roles

In XS Advanced , in order to perform a specific task of an application, a user needs to have the specific role required by the application.

However, a role can not be assigned directly to a user, it must be assigned to a role collection and then this role collection can be assigned to the user.

You can use role collections to differentiate between different kinds of authorizations, that is, between the view authorizations and the modify authorizations.

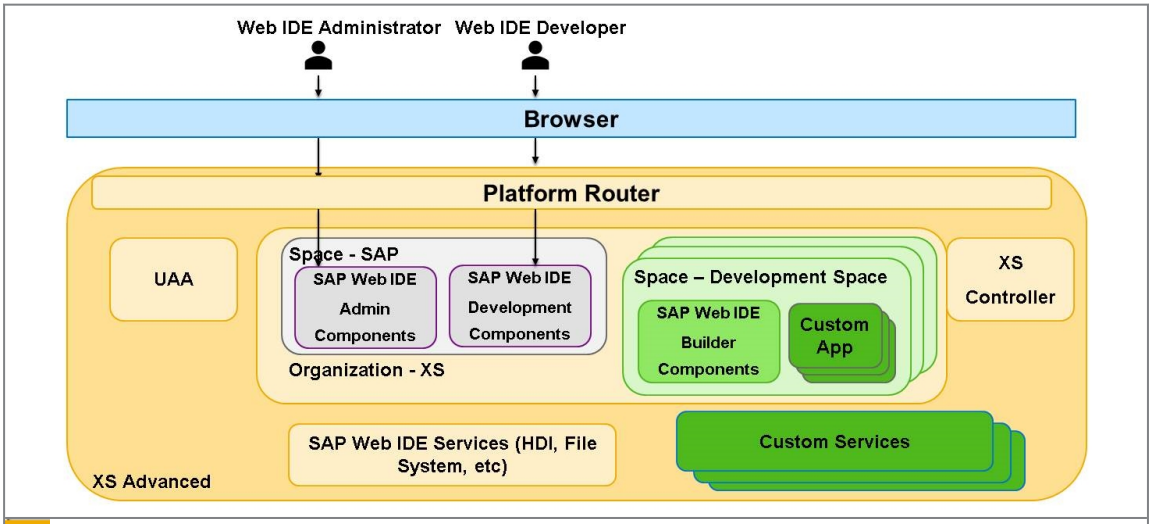


Figure 168: SAP Web IDE for SAP HANA Users

As an example of the XS Advanced application, let's have a close look at the SAP Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is a browser-based integrated environment to develop XS Advanced applications.

To ensure the isolation of the development environment, the SAP Web IDE for SAP HANA supports multiple spaces in which developers can build and run their applications.

Developers who use the same space can access each others application artifacts.

All SAP Web IDE for SAP HANA components, except the Builder component, are installed in the predefined SAP space, whereas the Builder component is installed in each space used for development.

The SAP Web IDE for SAP HANA is installed on top of XS Advanced and uses The XS Advanced User Account and Authorization service (UAA) and the application router to manage user logon and logout requests.

The SAP Web IDE for SAP HANA supports the following user types:

- SAP Web IDE developer

An SAP HANA database user who has additional permissions to create, modify, build, and run applications in the SAP Web IDE for SAP HANA . These developers are assigned to personal SAP Web IDE for SAP HANA workspaces, where they manage their own application artifacts, such as projects, modules, and files. Workspace access is granted only to the workspace owner.

- SAP Web IDE administrator

An SAP HANA database user who has additional permissions to perform administration tasks for the SAP Web IDE for SAP HANA .

User Type	Role/Role Collections	Description	User Type	Role/Role Collections	Description
Developer	XS_CONTROLLER_USER role collection	Grants read-write permissions within the assigned organization or space.	Administrator	XS_CONTROLLER_USER role collection	Grants read-write permissions within the assigned organization or space.
	SpaceDeveloper role	Assigned per space in XS Advanced. Enables users to access the shared resources of the space, and to deploy, build, and run applications.		SpaceDeveloper role	Assigned per space in XS Advanced. Enables users to access the shared resources of the space, and to deploy, build, and run applications.
	A role collection containing the WebIDE_Developer role	Enables users to develop applications using SAP Web IDE.		A role collection containing the WebIDE_Administrator role template	Enables users to access the SAP Web IDE administration tools, such as SSL management and space enablement.

Figure 169: SAP Web IDE for SAP HANA User Roles

The table lists the required roles and role collections for a developer and an administrator of the SAP Web IDE for SAP HANA.

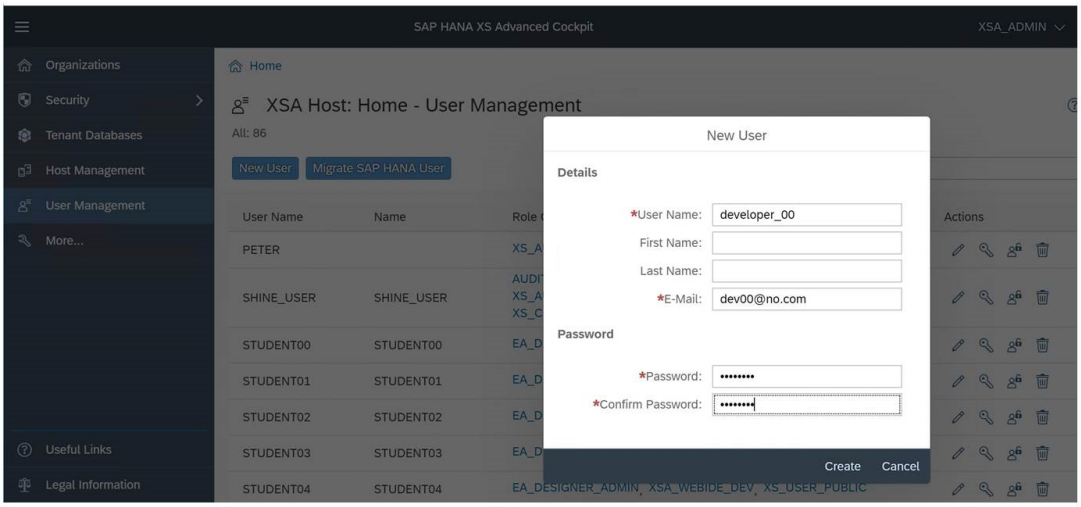


Figure 170: XS Advanced Cockpit (xsa-cockpit) - User Creation

If you access the XS Advanced Cockpit, as an administrator with the appropriate authorizations, under "User Management", you can maintain and manage database and business users for XS Advanced.

You can perform the following actions:

- Create New User: Create a new database and business user
- Migrate HANA User: Upgrade an existing database user to XS Advanced business user
- Edit User: Edit details for a user like name and email address
- Change Password: Change the XS Advanced password for a user
- Assign Role Collections: Add or remove role collection assignments for a user
- Delete User: Delete a user

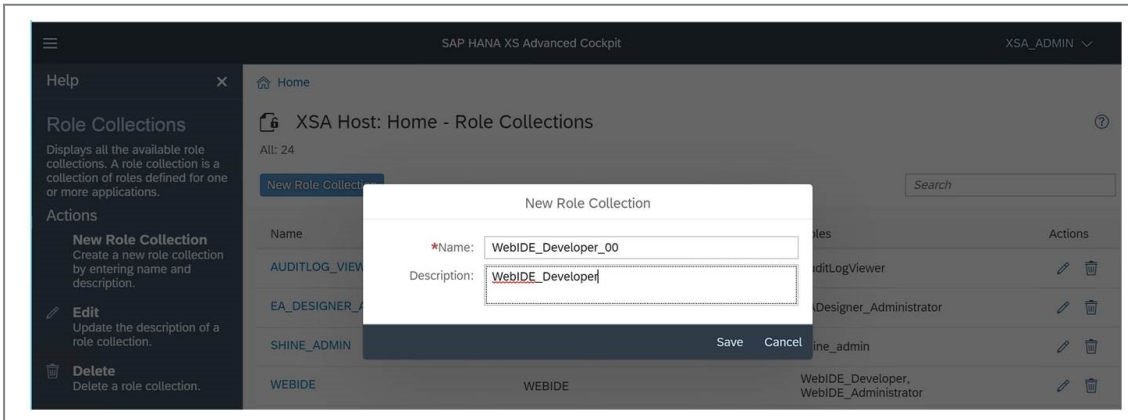


Figure 171: XS Advanced Cockpit (xsa-cockpit) - Role Collections

In the XS Advanced Cockpit, as an administrator, you can view all of the available role collections (collections of roles defined for one or more applications) and you can perform the following actions under Security > Role Collections:

- Create a New Role Collection: Create a new role collection by entering name and description
- Edit: Update the description of a role collection
- Delete: Delete a role collection

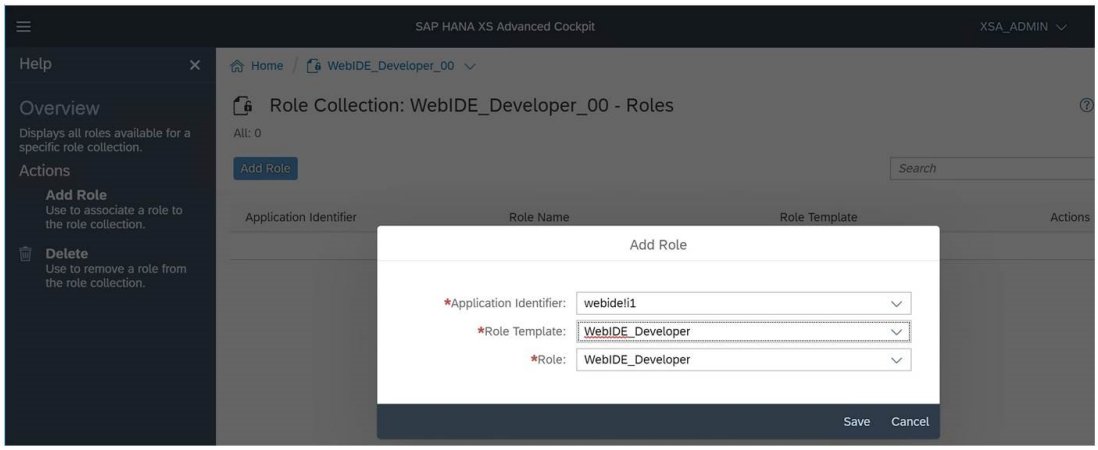


Figure 172: XS Advanced Cockpit (xsa-cockpit) - Add Roles

Inside a role collection in the XS Advanced Cockpit you can:

- Add Role: Associate a role to the role collection
- Delete: Remove a role from the role collection

To identify the role to be added to the role collection, you are, first of all, requested to provide the identifier of the application you are interested in.

Then you have to choose between the role templates and the roles exposed by this application.

A role-template is a description of a role and of the attributes that apply to it.

If a role template does not contain any attributes, then a role is generated automatically with the same name as the role template, as shown in the figure above.

Otherwise, a role based on a role template can be generated using the XS administration tools for XS Advanced.

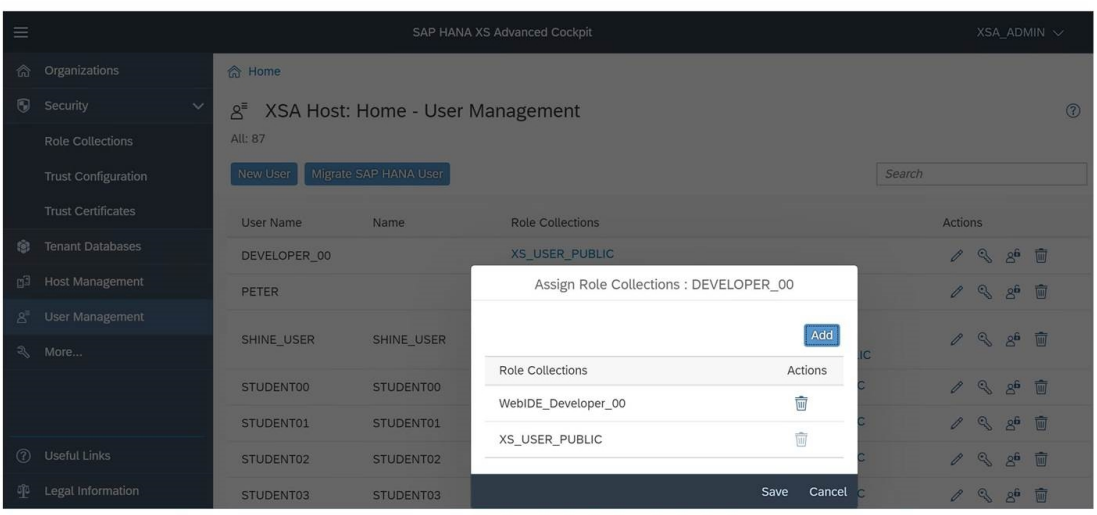


Figure 173: XS Advanced Cockpit (xsa-cockpit) - Assign Role Collection

Return to "User Managent" in the XS Advanced Cockpit, to assign the newly created role collection to the user.

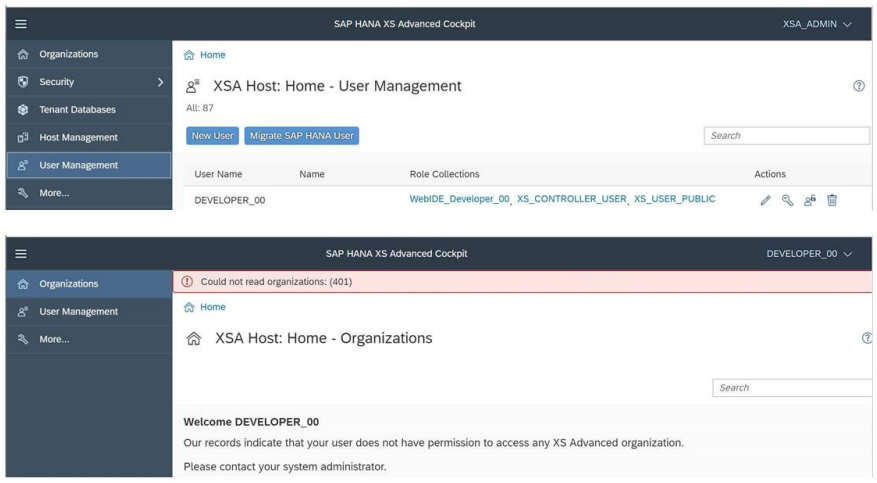


Figure 174: Access the XS Advanced Application

When the user has the necessary role collections assigned to them, they can execute the XS Advanced application, performing only the actions to which they are authorized.

What is missing for our user to be able to develop with the SAP Web IDE for SAP HANA is to be assigned as a member to a space, that is, DEV, as Space Developer.

LESSON SUMMARY

You should now be able to:

- Create the user with authorization for development in the SAP Web IDE for SAP HANA

Unit 11
Lesson 3

Creating the Security Concept Within an HTML5 Module

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create the security concept within an HTML5 module

Creating the security concept within an HTML5 module

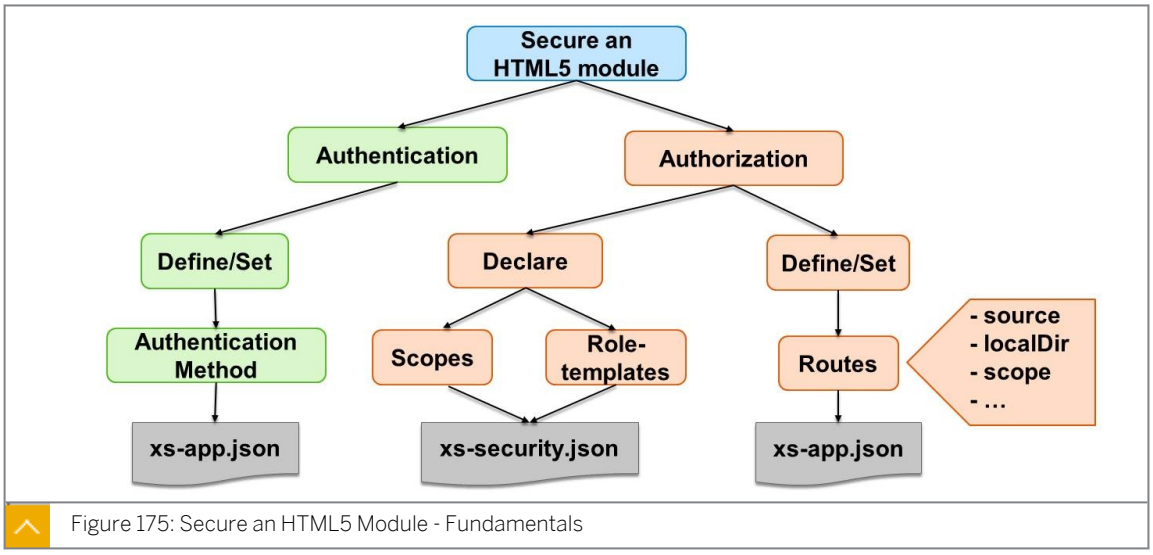
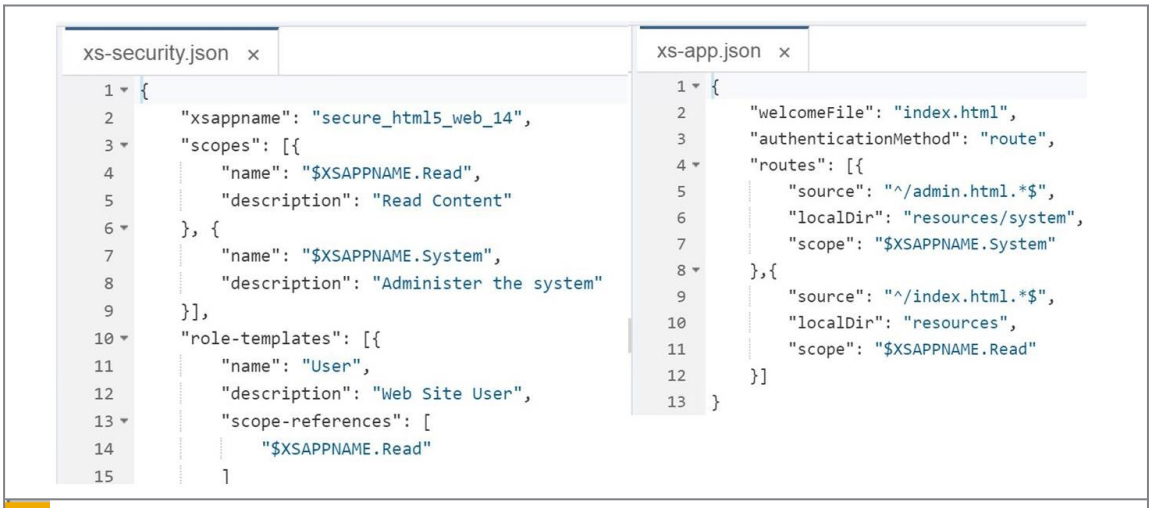


Figure 175: Secure an HTML5 Module - Fundamentals

Let's assume that you want to secure a simple XS Advanced application made of a single HTML5 module with two web pages. One page is for the business users who insert, delete, and modify transactional data. The other page is for the administrators who have the rights to define and modify special settings of the application itself.

In this case, you have to define how your application's users are identified and which roles they need to access to each page.



xs-security.json

```
{
    "xsappname": "secure_html5_web_14",
    "scopes": [{
        "name": "$XSAPPNAME.Read",
        "description": "Read Content"
    }, {
        "name": "$XSAPPNAME.System",
        "description": "Administer the system"
    }],
    "role-templates": [{
        "name": "User",
        "description": "Web Site User",
        "scope-references": [
            "$XSAPPNAME.Read"
        ]
```

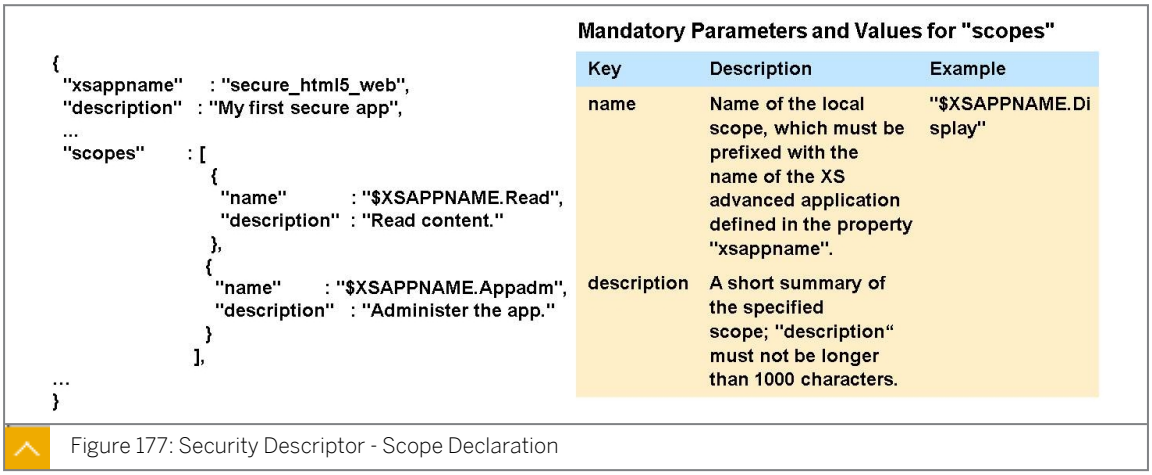
xs-app.json

```
{
    "welcomeFile": "index.html",
    "authenticationMethod": "route",
    "routes": [{
        "source": "^/admin.html.*$",
        "localDir": "resources/system",
        "scope": "$XSAPPNAME.System"
    },{
        "source": "^/index.html.*$",
        "localDir": "resources",
        "scope": "$XSAPPNAME.Read"
    }]
}
```

Figure 176: Security Descriptor and Application Descriptor Files

In the XS Advanced Programming Model, the entry point for each business application is the application router , which routes incoming requests to the appropriate microservices. The application router needs the configuration information contained into the application descriptor, xs-app.json, which is a JSON-formatted file.

The security descriptor file, xs-security.json, is the central component for the configuration and maintenance of application security; its content describes any required user authorizations using scopes, such as view and edit, attributes, such as, dynamically defined cost center, department, and so on, and role templates (used to build roles). Authorization scopes enable you to restrict access to resources based on user permissions specified in role templates.



```
{
 "xsappname"     : "secure_html5_web",
 "description"   : "My first secure app",
 ...
 "scopes"        : [
                    {
                     "name"         : "$XSAPPNAME.Read",
                     "description"  : "Read content."
                    },
                    {
                     "name"         : "$XSAPPNAME.Appadm",
                     "description"  : "Administer the app."
                    }
                   ],
...
}
```

Mandatory Parameters and Values for "scopes"

Key	Description	Example
name	Name of the local scope, which must be prefixed with the name of the XS advanced application defined in the property "xsappname".	"$XSAPPNAME.Display"
description	A short summary of the specified scope; "description" must not be longer than 1000 characters.	

Figure 177: Security Descriptor - Scope Declaration

Scopes are used to define the authorizations required by the users to access the features provided by specific XS advanced applications, for example, view, change, or delete.

They are defined in an application's security descriptor file, xs-security.json, and created when the application is deployed or updated. In the example in the figure above, you can see the scopes $XSAPPNAME.Read and $XSAPPNAME.Appadm. Each scope has, at least, a name and a short description.

In addition to the so-called "local" (application-specific) scopes, it is also possible to define "foreign" scopes, which are valid for other applications, too. Foreign scopes are not provided by the application itself; they are checked by other sources outside the context of the application.

Scopes are non-mandatory properties of the application router that define the authorization scopes required to access the target paths. After a successful authentication, the application router starts an authorization check based on scopes. Scopes can also be checked programmatically by the application itself. The authorization checks can be performed declaratively, for example, by the application router or in the Java runtime container, or programmatically, for example, in either the Java or the Node.js runtime containers.

By declaring the scopes of your application in the security descriptor file, xs-security.json, assigned to the application itself, the functional authorization scopes for the application are communicated to the User Account and Authentication service (UAA).

Each scope name must be unique in the context of the current XS Advanced UAA installation. So, "local" scopes should be prefixed with the variable <$XSAPPNAME>, which, at runtime, is replaced with the local application name. The "name" can contain up to 193 characters and must not start with a period '.' (dot), and cannot include any characters included in the so-called "blacklist". Permitted characters include: 'a'-'z', 'A'-'Z', '0'-'9', '_', '-', '\', '/', ':', and '.'

The xs-security.json file uses JSON notation to define the security options for an application.

The scopes are assigned to the users by means of security roles.

```
{
"xsappname"     : "secure_html5_web",
"description"   : "My first secure app",
 ...
"role-templates": [
                    {
                      "name"           : «User",
                      "description"    : «Web site user.",
                      "scope-references" : [
                            "$XSAPPNAME.Read"
                                          ]
                    },
                    {
                      "name"           : «Administrator",
                      "description"    : «App Administrator",
                      "scope-references" : [
                            "$XSAPPNAME. Appadm",
                                          ]
                    }
                  ]
}
```

Role-Template Parameters

Key	Description	Example
name	The name of the role to build from the role template	“Viewer”
description	A short summary of the role to build	“View all books”
scope-references	The security (authorization) scope to apply to the application-related role	“$XSAPPNAME.Display”
attribute-references	One or more attributes to apply to the built role	[“Country”, “CostCenter”]

Figure 178: Security Descriptor - Role-Template Declaration

At design time, when the authorization scopes for your XS Advanced applications are defined in the application's security descriptor, xs-security.json, you, as developer, need to provide a reference for them in a role-template.

A role-template defines the authorization scopes that can be associated, for example, with a given position in a company.

After the deployment of these authorization artifacts, the administrators use the role templates provided by the developers to build roles, which are aggregated in role collections and the role collections are assigned, in turn, to business users.

In the application-security file, xs-security.json, you can define multiple role-templates.

The role-templates property enables you to define an array listing one or more roles with its own scopes and attributes. If you want to know more about attributes, please refer to the SAP HANA Developer Guide for XS Advanced Model.

In the example in the figure above, we have the scopes User and Administrator. The maximum length of an XS Advanced role-template name is 64 characters and can be made of the following characters "aA"-"zZ", "0"-"9", "_" (underscore).

```
{
  "welcomeFile": "index.html",
  "authenticationMethod": "route",
  ...
}
```

Property	Type	Mandatory	Values
authenticationMethod	String	No	The following authentication methods are allowed: • route Authentication type is defined in the route configuration • none Disables authentication for all routes

Figure 179: Application Descriptor - Authentication Method

The application descriptor file, xs-app.json, has the configuration information used by the application router; it contains the mandatory and the optional properties in a JSON-compliant syntax.

The top-level property, "authenticationMethod", identifies the method used to authenticate user requests. It can be set to route or none.

If "authenticationMethod" is set to none, logon with User Account and Authentication (UAA) is disabled. If "authenticationMethod" is set to route, the authentication type is defined in the route configuration.

```
{
  "welcomeFile": "index.html",
  ...
  "routes": [
          {
           "source": "^/admin.html.*$",
           "localDir": "resources/system",
           "scope": "$XSAPPNAME.Appadm",
          },
          {
           "source": "^/index.html.*$",
           "localDir": "resources",
           "scope": "$XSAPPNAME.Read",
          },
         ],
}
```

Some Routes Properties

Property	Type	Mandatory	Description
source	RegEx	Yes	Regular expression that matches an incoming request path. To ensure the RegEx matches the complete path, wrap it with ^ and $, i.e. "^/sap/ui5/1(.*)$". A request matches a particular route if its path contains the specified regular expression.
scope	String	No	The authorization scope required to access the target path. The scope itself is defined in the application's security descriptor (xs-security.json).
localDir	String	No	The directory from which application router serves static content (for example, from the application's web/ module)

Figure 180: Application Descriptor - Routes

The routes determine, with the specified paths, which incoming requests should be forwarded to which destination (applications/microservices). Therefore, the routes must be defined for your applications.

A path is considered public, if it does not require authentication. This is the case for routes with "authenticationType: none", or if authentication is disabled completely via the top level property "authenticationMethod: none".

For every destination, there can be more than one route.

The application routes are defined in the descriptor file, xs-app.json, a JSON-formatted file that is mandatory for the approuter.

The properties target, destination, and localDir are optional. However, at least one of them must be defined.

The properties destination and localDir cannot be used together in the same route.

If there is no route defined for serving static content localDir, a default route is created for resources.

In the example in the figure above, we have defined one route for the page admin.html with the scope $XSAPPNAME.Appadm and one for the page index.html with the scope $XSAPPNAME.Read.

```
ID: secure_html5
_schema-version: '2.1'
version: 0.0.1

modules:
 - name: secure_html5_web
   type: html5
   path: secure_html5_web
   requires:
    - name: secure_html5_##_uaa

resources:
 - name: secure_html5_##_uaa
   type: com.sap.xs.uaa
   parameters:
     path: xs-security.json
```

```
1  ID: secure_html5
2  _schema-version: '2.1'
3  version: 0.0.1
4
5  modules:
6   - name: secure_html5_web
7     type: html5
8     path: secure_html5_web
9     requires:
10      - name: secure_html5_XX_uaa
11 resources:
12  - name: secure_html5_XX_uaa
13    type: com.sap.xs.uaa
14    parameters:
15      path: xs-security.json
```

Figure 181: Deployment descriptor- Specifies the Dependencies

The deployment descriptor specifies the dependencies required to ensure a successful deployment to the XS Advanced runtime.

Authentication and authorization are managed using the XS Advanced User Account and Authentication (UAA) service. In order to bind your application modules to an instance of the XS Advanced UAA service, you need to define a new UAA service. You can perform this configuration in the application's central deployment descriptor file, mta.yaml.

You need to add a new service instance to the resources section and to the requires section of the mta.yaml application descriptor file.

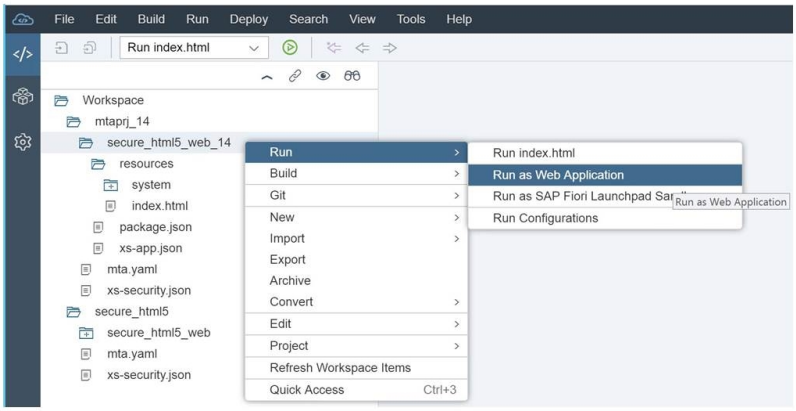


Figure 182: Application Security Test

At this point, the XS Advanced UAA is aware of the role-templates defined in the application's security descriptor , xs-security.json. However, since the required scopes and roles are not yet assigned to a user, the user does not yet have the permissions required to access the advertised application end points.

You need to create a new role based on the role-template specified in the application's xs-security.json file, add the role to a role collection (either existing or newly created), and assign a user to the role collection.

Logging into the application as the user with the assigned role collection ensures that the configured authorizations are assigned to the logged in user.

LESSON SUMMARY

You should now be able to:

- Create the security concept within an HTML5 module

Learning Assessment

1. Which resources do you find in XS Advanced spaces?

   Choose the correct answers.

   - ☐ **A** Application routes
   - ☐ **B** Application domains
   - ☐ **C** Memory
   - ☐ **D** Services
   - ☐ **E** Server certificates

2. How can you assign the authorizations needed to perform specific tasks executing XS Advanced applications?

   Choose the correct answer.

   - ☐ **A** You can add the necessary roles to a role collection and then assign this role collection to the user.
   - ☐ **B** You can directly assign the necessary roles to the user.
   - ☐ **C** You can add the necessary permissions to a role collection and then assign this role collection to the user.
   - ☐ **D** You can directly assign the necessary permissions to the user.

3. You are setting up security for an HTML application. Where do you link an HTML page with the role needed to visualize it?

   Choose the correct answer.

   - ☐ **A** In the scope definition.
   - ☐ **B** In a route definition.
   - ☐ **C** In the role definition.
   - ☐ **D** In the authentication method definition.

Learning Assessment - Answers

1. Which resources do you find in XS Advanced spaces?

   Choose the correct answers.

   - [X] A Application routes
   - [ ] B Application domains
   - [X] C Memory
   - [X] D Services
   - [ ] E Server certificates

   That is correct! Memory, application routes and services are resources shared in each XS Advanced space.

2. How can you assign the authorizations needed to perform specific tasks executing XS Advanced applications?

   Choose the correct answer.

   - [X] A You can add the necessary roles to a role collection and then assign this role collection to the user.
   - [ ] B You can directly assign the necessary roles to the user.
   - [ ] C You can add the necessary permissions to a role collection and then assign this role collection to the user.
   - [ ] D You can directly assign the necessary permissions to the user.

   This is correct! To assign the necessary authorization to execute specific tasks of an XS Advanced application to a user, you can add the necessary roles to a role collection and then assign this role collection to the user.

3. You are setting up security for an HTML application. Where do you link an HTML page with the role needed to visualize it?

   Choose the correct answer.

   - [ ] A In the scope definition.
   - [x] B In a route definition.
   - [ ] C In the role definition.
   - [ ] D In the authentication method definition.

   This is correct! When you define a route, you can specify the HTML page in the source property and a scope in the related property. The scope will be then referred by a role-template, which in turn will be used for the role creation.

UNIT 12 Creating the User Interface Using UI5

UNIT OBJECTIVES

- Explain what UI5 is
- Describe the structure of an elementary UI5 application
- Create the UI using the SAP Fiori Master-Detail template

Unit 12
Lesson 1

Introducing UI5

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Explain what UI5 is

Introducing UI5

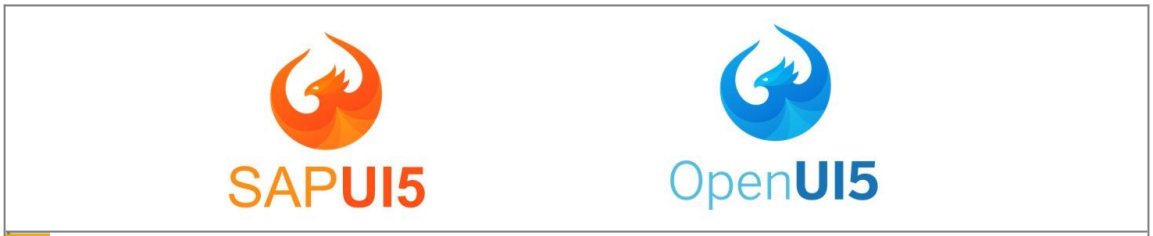


Figure 183: UI5

UI5 is a JavaScript application framework designed to build cross-platform, responsive, enterprise-ready applications. It is based on HTML5 and JavaScript and runs in the web browser. You can access it at:

https://sapui5.hana.ondemand.com

Responsive Across Browsers and Devices

UI5 apps run on smartphones, tablets, and desktops. The UI controls automatically adapt themselves to each device's capabilities and make the most of the available real estate.

Powerful Development Concepts

The UI5 core offers a solid foundation that simplifies your work by managing many aspects of modern development behind the scenes. It comes with built-in support for architectural concepts like MVC, two-way data binding, and routing. The UI5 core has the following features and capabilities:

- Includes standards like MVC and various data-binding types
- Choose between different view formats (XML, HTML, JavaScript, JSON)
- Binding with OData, JSON, XML and other data formats
- Built-in support tool for exploring the object tree and binding status

Enterprise-Ready Web Toolkit

UI5 comes with all features needed to cover most current application requirements, with standards high enough to be delivered in standard SAP solutions. UI5 has the following features and capabilities:

- Translation and internationalization support
- Extensibility concepts at code and application level
- High contrast theme to aid visually impaired users

Award-Winning SAP Fiori Design in Action

UI5 applications benefit from a consistent design language and predefined UX patterns across all frontend features. Use a predefined theme or try our online theming tool to create a custom theme.

OpenUI5 is an open-source project maintained by SAP SE, providing UI5 under the Apache 2.0 license and open to contributions.

You can access it at:

https://openui5.org

https://github.com/SAP/openui5

LESSON SUMMARY

You should now be able to:

- Explain what UI5 is

Describing the Structure of an Elementary UI5 Application

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Describe the structure of an elementary UI5 application

Describing the Structure of an Elementary UI5 Application

`index.html` is the default starting point of the application. It contains 3 main areas:

- Bootstrap - Contains configuration including the libraries to be loaded, resource root location, and theme.
- Application – Initial construction of application including `placeAt` method to attach application into primary UI Area
- UI-Area – Contains primary UI anchor.

Best practice keeps minimal coding in `index.html`

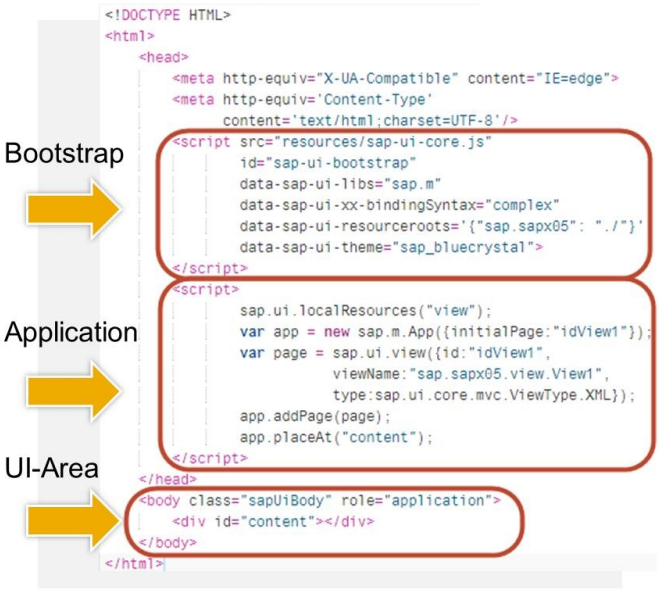


Figure 184: Index HTML

LESSON SUMMARY
You should now be able to:

- Describe the structure of an elementary UI5 application

Unit 12
Lesson 3

Creating the UI Using the SAP Fiori Master-Detail Template

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create the UI using the SAP Fiori Master-Detail template

Describing the Fiori Master-Detail Template

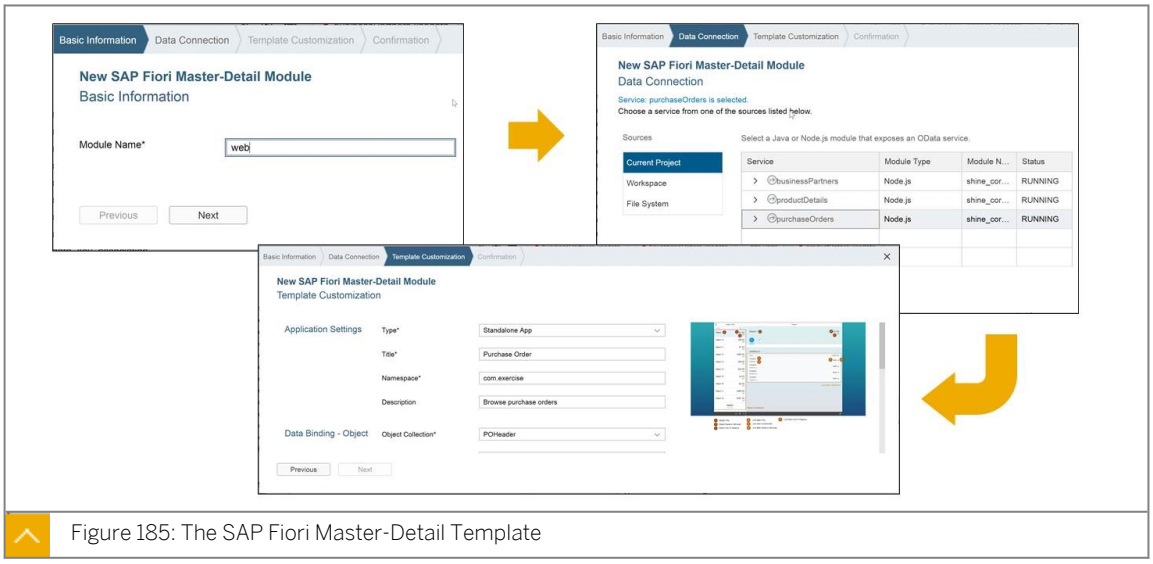


Figure 185: The SAP Fiori Master-Detail Template

The SAP Fiori Master-Detail template generates a SAPUI5 app displaying master-detail data, for example Order-Item.

The application is generated based on a pre-existing OData service providing the data.

A wizard guides you along the creation process. The wizard identifies the OData services, it reads the metadata, and allows you to personalize the application based on the collected information.

You don't need to provide any technical detail.

The final result is a Fiori-compliant application, connected to the OData service, capable to browse the master detail-data, which is a good starting point to further develop your application.

LESSON SUMMARY

You should now be able to:

- Create the UI using the SAP Fiori Master-Detail template

Learning Assessment

1. What is OpenUI5?

   Choose the correct answer.

   - ☐ **A** An extensibility framework for UI5
   - ☐ **B** A platform providing free video courses
   - ☐ **C** An open-source project
   - ☐ **D** A guideline for an optimal user experience

2. You are creating the index.html file of an UI5 application. What does the bootstrap contain?

   Choose the correct answer.

   - ☐ **A** The initial construction of the application
   - ☐ **B** The libraries to be loaded
   - ☐ **C** The primary UI anchor

3. What is the SAP Fiori Master-Detail template?

   Choose the correct answer.

   - ☐ **A** A Master-Detail application, used to generate a documentation template for your application
   - ☐ **B** A demo application, part of SHINE (SAP HANA Interactive Education)
   - ☐ **C** A guideline document that explains the best practice to create a Fiori compliant Master-Detail application
   - ☐ **D** A wizard that generates a UI5 Master-Detail application based on an existing OData service

Learning Assessment - Answers

1. What is OpenUI5?

   Choose the correct answer.

   - [ ] A An extensibility framework for UI5
   - [ ] B A platform providing free video courses
   - [x] C An open-source project
   - [ ] D A guideline for an optimal user experience

   This is correct! OpenUI5 is an open-source project.

2. You are creating the index.html file of an UI5 application. What does the bootstrap contain?

   Choose the correct answer.

   - [ ] A The initial construction of the application
   - [x] B The libraries to be loaded
   - [ ] C The primary UI anchor

   That is correct! The bootstrap contains the libraries to be loaded, the resource root location, the theme, and so on.

3. What is the SAP Fiori Master-Detail template?

   Choose the correct answer.

   - [ ] A A Master-Detail application, used to generate a documentation template for your application
   - [ ] B A demo application, part of SHINE (SAP HANA Interactive Education)
   - [ ] C A guideline document that explains the best practice to create a Fiori compliant Master-Detail application
   - [x] D A wizard that generates a UI5 Master-Detail application based on an existing OData service

   That is correct! The SAP Fiori Master-Detail template is a wizard that generates a UI5 Master-Detail application based on an existing OData service.

UNIT 13

Using the SAP Cloud Application Programming Model

Lesson 1

UNIT OBJECTIVES

- Use the SAP Cloud Application Programming model

Unit 13
Lesson 1

Using the SAP Cloud Application Programming Model

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Use the SAP Cloud Application Programming model

Introducing the SAP Cloud Application Programming model

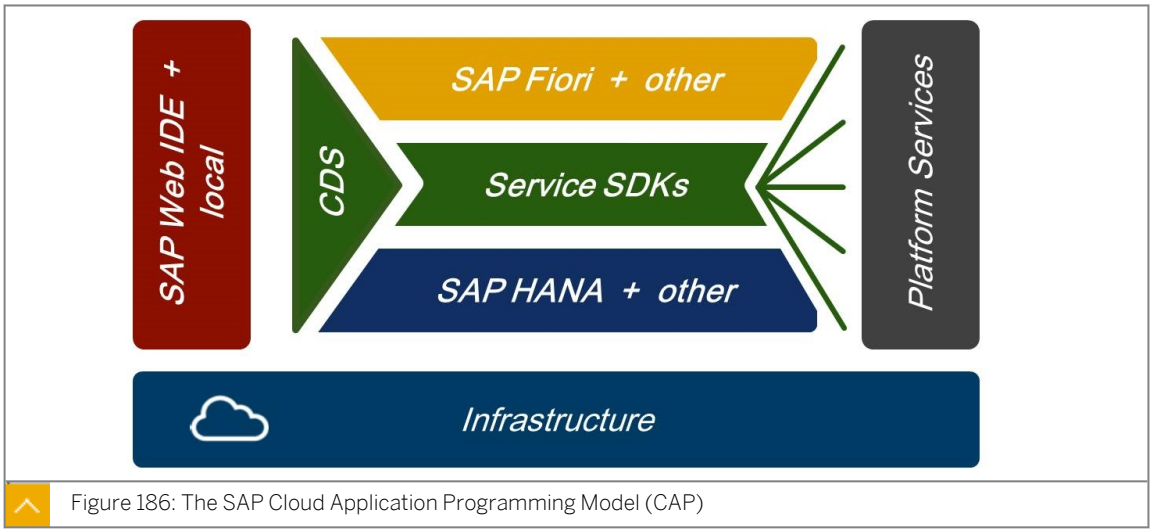


Figure 186: The SAP Cloud Application Programming Model (CAP)

The SAP Cloud Application Programming Model (CAP) is the go-to programming model for business applications on SAP HANA and SAP Cloud Platform.

The CAP model is based on Core Data Services (CDS), to define database model, services, and UI with a unified formal language.

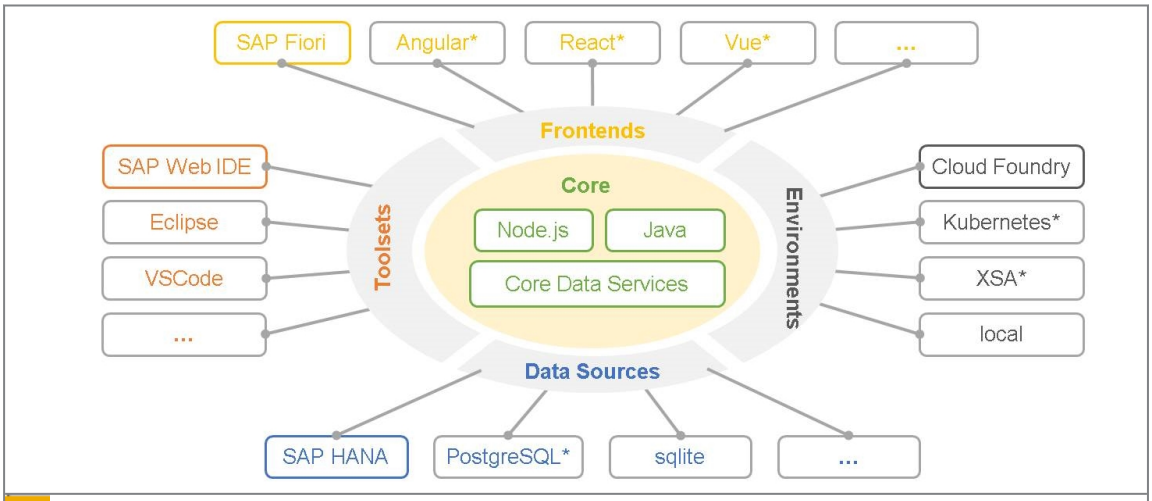


Figure 187: Flexibility and Openness

CAP abstracts the definition of the business logic, giving the chance of choice for:

- The UI technology
- The database
- The runtime environment
- The development environment

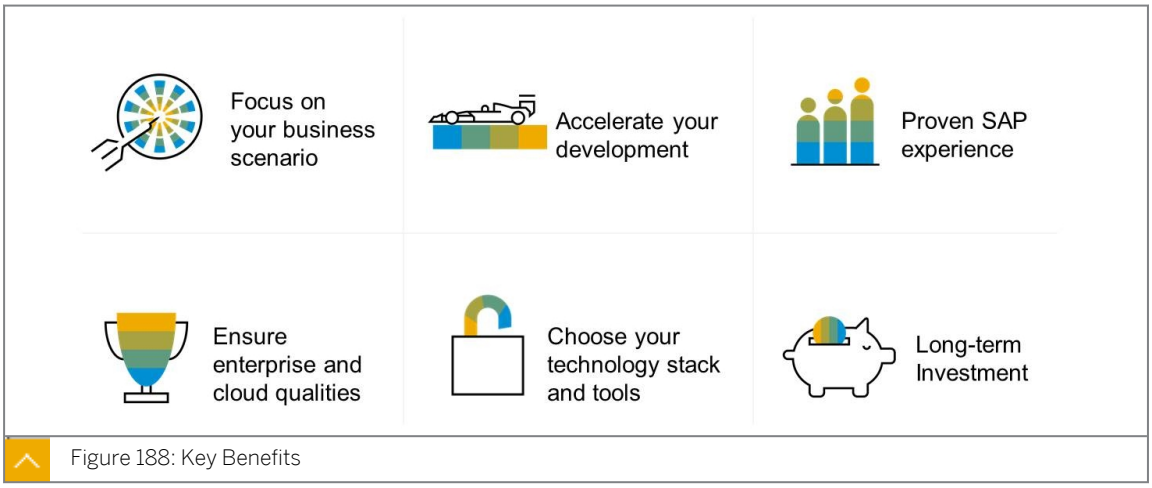


Figure 188: Key Benefits

The key benefits are:

- Focus on your business scenario

Write your business logic and let the framework help you manage auditing, authorizations, configurations, etc., thus creating a more comprehensive and concise code, increasing application adaptability and reducing maintenance efforts. Automate tedious tasks → multi-tenancy, extensibility, authorization, audit logging, data privacy, localization, platform integration, and so on.

- Accelerate your development

With CDS you can quickly create data models and services that are ready to run.In addition, the SAP Web IDE for SAP HANA helps you to jump start your development with dedicated templates and tools.

- Proven SAP experience

The application programming model is based on proven SAP technologies used internally in SAP, such as CDS, SAP HANA, and SAP Fiori.

- Ensure enterprise and cloud qualities

Built-in support for authorization, audit logging, configuration, tenant isolation, and additional required features for cloud and enterprise applications.

- Choose your technology stack and tools

The application programming model offers a recommended set of tools and technologies. However, as a flexible and open framework, it allows you to choose your preferred data source, environment, front-end technology, and toolset.

- Long-term Investment

CDS is not dependent on a specific technology or platform. The development effort of today will remain relevant for future technologies.

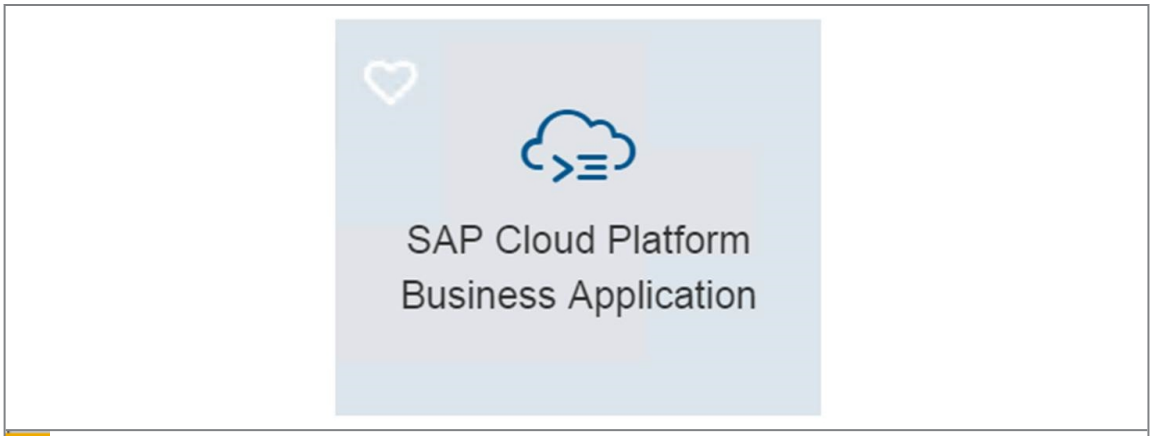


Figure 189: The SAP Web IDE for SAP HANA Template for CAP

If you want to create an application based on CAP, you need to use a specific project template within the SAP Web IDE for SAP HANA.

This template is named SAP Cloud Platform Business Application. Other templates, for example, the Multi-target Application, don't support the new model.

db/data-model.cds

```
namespace my.bookshop;
using { Country, managed } from '@sap/cds/common';

entity Books {
  key ID : Integer;
  title  : localized String;
  author : Association to Authors;
  stock  : Integer;
}
```

Figure 190: Defining Database Model

srv/cat-service.cds

```
using my.bookshop from '../db/data-model';

service CatalogService {
  entity Books @readonly as projection on bookshop.Books;
  entity Authors @readonly as projection on bookshop.Authors;
  entity Orders @insertonly as projection on bookshop.Orders;
}
```

Figure 191: Defining the OData Service

srv/cat-service-fiori.cds

```
annotate CatalogService.Books with @(
  UI.LineItem: [
    {$Type: 'UI.DataField', Value: ID},
    {$Type: 'UI.DataField', Value: title},
    {$Type: 'UI.DataField', Value: stock},
    {$Type: 'UI.DataField', Value: "author/name"},
  ],
```

Figure 192: Defining the Fiori UI

LESSON SUMMARY

You should now be able to:

- Use the SAP Cloud Application Programming model

Learning Assessment

1. You are programming using the SAP Cloud Application Programming Model. You need to define an OData service. Which is the extension of the file you create?

   Choose the correct answer.

   - ☐ A xsodata
   - ☐ B odata
   - ☐ C hdbcds
   - ☐ D cds

Learning Assessment - Answers

1. You are programming using the SAP Cloud Application Programming Model. You need to define an OData service. Which is the extension of the file you create?

   Choose the correct answer.

   - [ ] A xsodata
   - [ ] B odata
   - [ ] C hdbcds
   - [x] D cds

   That is correct! The OData service is defined via a .cds file.

UNIT 14 Managing Source Code Using Git (optional)

Lesson 1

UNIT OBJECTIVES

- Use the Native Git Integration of the SAP Web IDE for SAP HANA

Unit 14
Lesson 1

Working with GIT Within SAP Web IDE for SAP HANA

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Use the Native Git Integration of the SAP Web IDE for SAP HANA

Overview of Git

Why Should You Use a Version Control System?

Think about an XS Advanced application. This application, with its different modules (SAP HANA database modules, UI, and control flow modules), consists of a number of files, organized in different folders.

Now, suppose the last released version of the application is V2.1.0, and you are currently preparing new features for the upcoming minor release V2.2.0, which is scheduled to release in a few weeks. You might have imported the MTA project in your SAP Web IDE for SAP HANA workspace and started developing the new features that are planned for V2.2.0. This development means that there are new files, as well as modifications to existing files, and maybe the deletion of a few existing files as well.

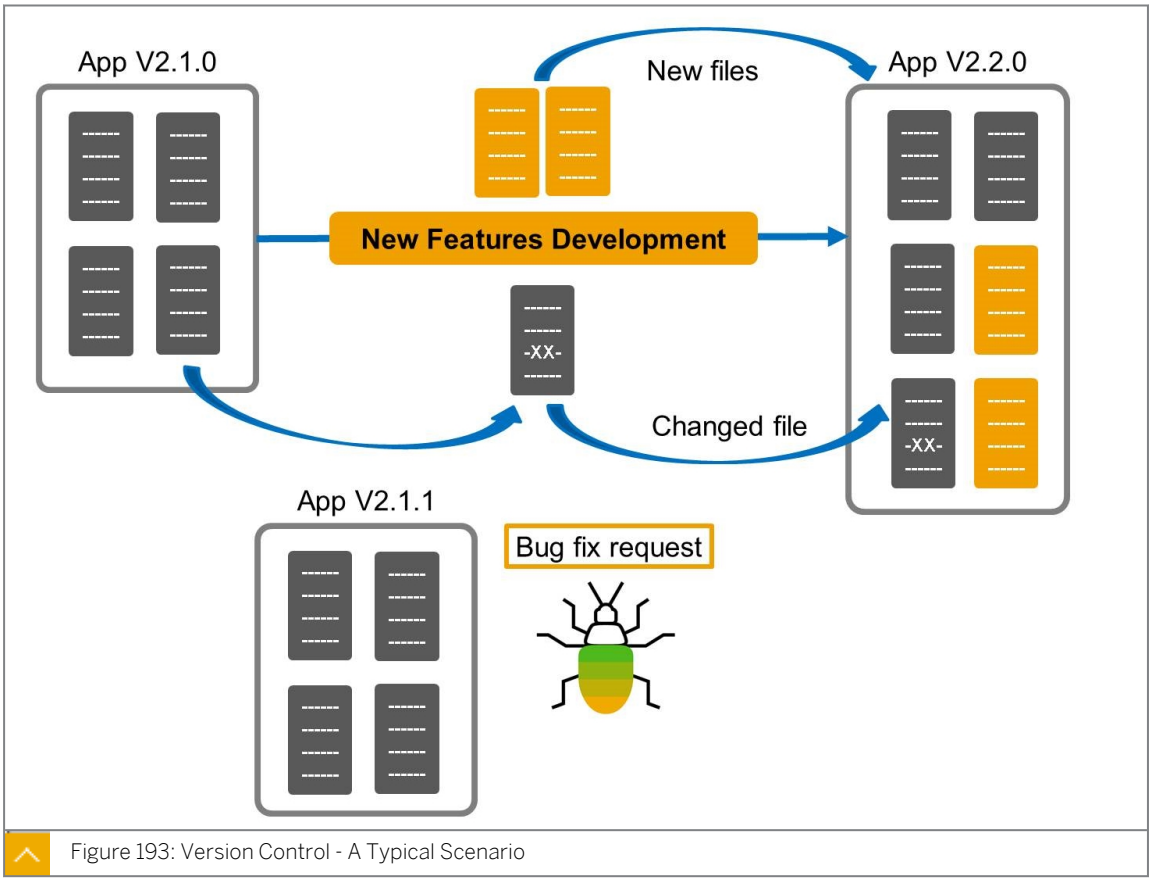


Figure 193: Version Control - A Typical Scenario

At some point, you receive an important e-mail from support saying that a bug in V2.1.0 needs an urgent fix. This cannot wait until the next minor version, and requires a patch (V2.1.1).

So, how do you handle this request? At the moment, your current development is not finished or tested, but you need to patch your version 2.1.0. Of course, you want to start from the last release (you do not want to include any part of the future functionality into the patch), but your patch might affect some files that you already modified as part of the development of new features.

This is where a version control system comes into play. It allows you to keep a complete change history by using milestones during the development of your code, at a very fine-grained level if necessary. You can also branch your code, which means, you can develop and test different features in different parallel development threads (branches). If a feature branch is good to go, you can merge this branch with the main branch. If it is not, you can continue your development, or even get rid of this branch if you realize that a development option for a feature was not relevant and you need to think about it again.

Key Benefits of a Version Control System

The following are the key benefits of using a version control system:

- Source code backup
- A complete change history
- Branching and merging capabilities
- Traceability (for example, connecting a change to project management software)

To learn more about version control systems, view the following page: https://www.atlassian.com/git/tutorials/what-is-version-control.

Git in a Nutshell

Git is a Distributed Version Control System (D-VCS) used to manage source code in software development, or more generally to manage the lifecycle of any set of files.

It allows one or several developers to work locally with their own copy of the Git repository, which contrasts with a traditional client/server architecture.

> Note:
> Git can also be used even out of a collaboration context, to help you control the development of a project on which you work alone, thanks to a number of capabilities.

The architecture of Git is distributed. It is somewhere between purely local and purely central. A local architecture would make it difficult to several developers to collaborate. A centralized one allows collaboration, but in some cases, a developer need to block a piece of code on the central repository while working on it.

Instead of this, Git is designed so that every developer can keep the entire history of a project (or only a part of it, depending of their needs), locally on their computer.

Git is a free software distributed under GNU GPL (General Public License).

The Git Architecture

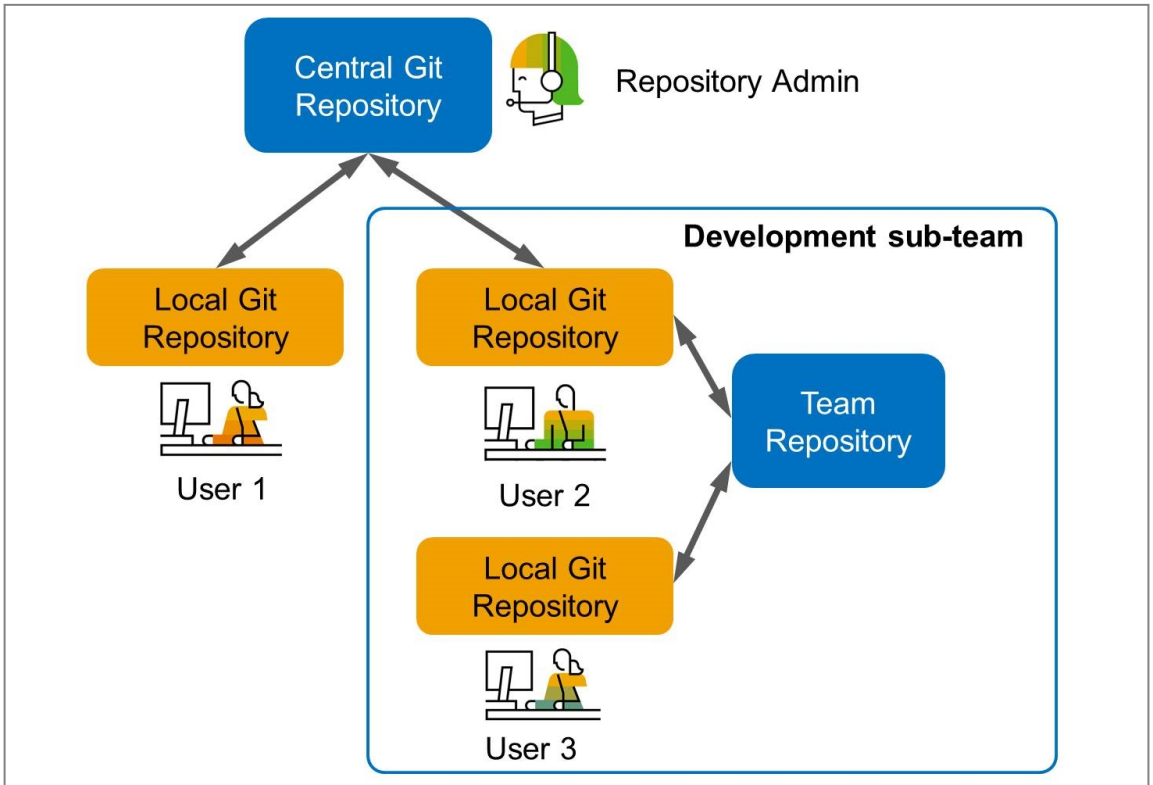


Figure 194: The Distributed Git Architecture

In a classical Git architecture, developers work in a local repository on their computer, and connect on a regular basis to a central repository to share their contribution and get the contribution from other developers on the project.

Git can also easily support several remote repositories for a single project. If needed, it is possible for a sub-team of developers to have their own sub-team repository to collaborate, and synchronize their changes with the central repository when needed.

Note:
In Git, it is even possible for a developer to define the local repository of another developer as a remote repository and to synchronize his development. This requires a network access and the relevant authorizations.

The shared Git repositories can be hosted on the own company's IT infrastructure, or on Git hosting services such as GitHub (one of the most popular), Helix (Perforce), Bitbucket (Altassian), and many more. A lot of companies offering Git hosting services also provide additional services such as code review or issue tracking.

The Key Benefits of Git

Git has been designed with security, flexibility, and performance in mind. Almost every operation is performed locally. The branching model provides an extreme flexibility.

To lean more about Git, go to: https://git-scm.com/.

The Lifecycle of Files in Git

When you work with files locally in Git (the case of remote Git repositories will be discussed later on), this involves three major "logical" areas.

These areas are:

- The Working Directory
- The Staging Area, also known as INDEX
- The (local) Git Repository

These are not all real areas, in the sense that a given file is not necessarily materialized in each of them. Let's explain this with a diagram.

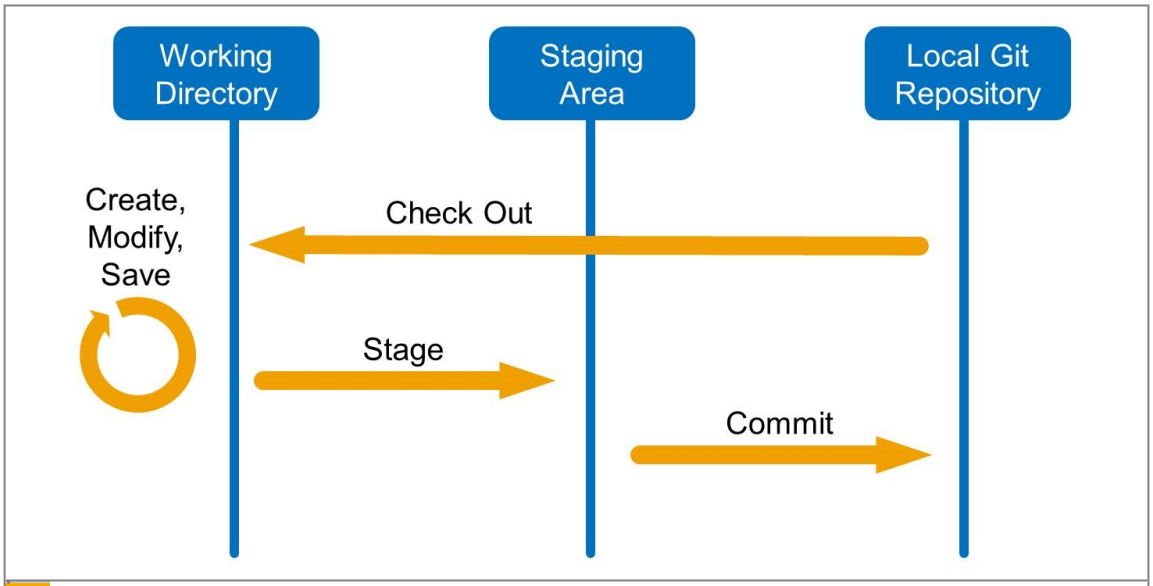


Figure 195: The Basic Git Workflow

The way to manage files in a local Git repository is very straightforward, relying on a small number of actions.

When you create a file, it is initially stored in your **Working directory**. You can also modify (or even delete, if needed) a file that you have previously exposed in your **Working Directory** with a *Check Out* command.

At this stage, your changes, even if they are saved in your working directory, are not yet part of the Git history. To update the Git history, you must perform what is called a **Commit**. This is what will create a new point in Git history (a so-called Commit), referencing a number of changes since the previous commit.

But what changes exactly do you want to include in your next commit? This is where the **Staging Area** comes into play. By staging a file, you mark this new or modified file so that it is included in the next **Commit**. You can of course stage several files (this is very common), or even stage all the current changes of your working directory.

Let's put it another way: The staging area is the "virtual" place where you put all the modifications that you want to include in your next commit.

Note:
So, when your working directory contains changes that you do not want to include in your next commit, you just need to make sure you do not stage the corresponding files before committing.

The Git History

Git is very good at representing the history of a project in a simple way, as the succession of commits.

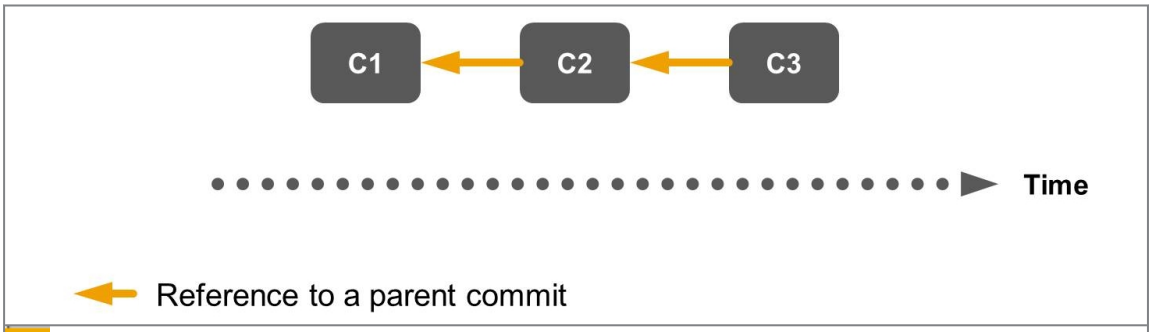


Figure 196: A Simple Git History

Over time, all the commits you execute in your project are added to the history, and each commit (except the initial one) references its parent commit.

> Note:
> Actually, you will see later on that in the case of a *Merge* operation, a commit can have two parent commits.

With each commit, Git keeps record of the commit date, the identity of the developer who executed it, and useful information about which files where affected (added, modified, or deleted).

The Branch Concept in Git

Branching is one of the core concepts of Git, which provides a huge flexibility at a very low cost. So what is a branch?

From a conceptual standpoint, a branch is a series of commits. Technically, a branch is just a pointer that references a particular commit of the project history.

Each Git repository always has at least one branch. The default branch is generally called **Master**.

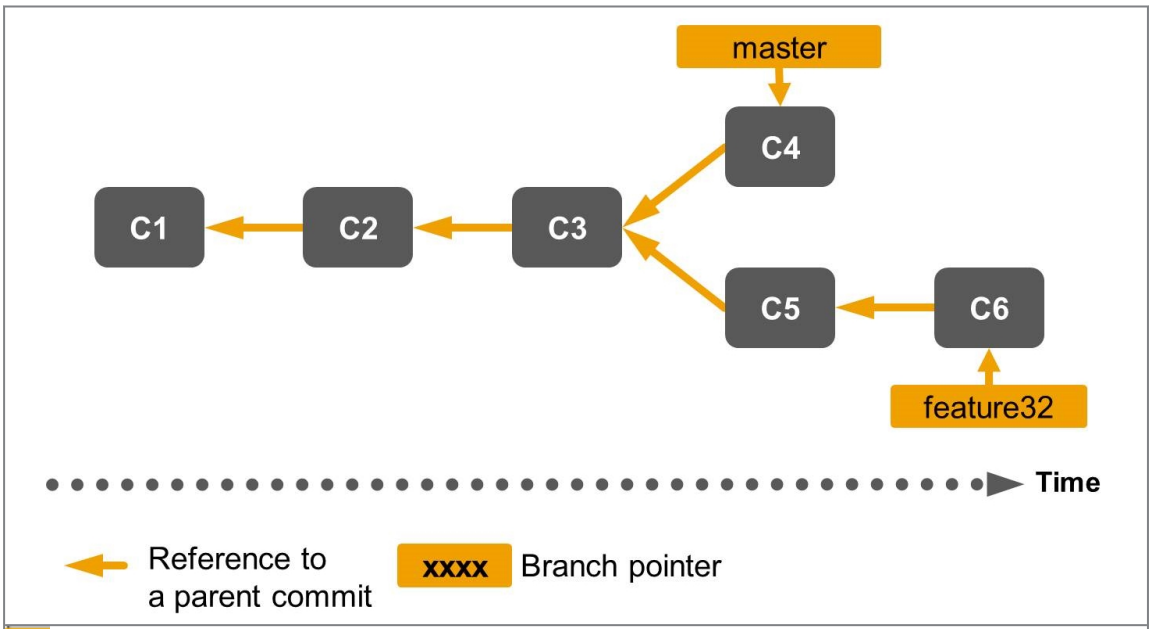


Figure 197: The Branch Concept

For many different purposes, you can create a new branch and commit your changes to one of the existing branch.

Let's describe the diagram above. After commit *C3*, a new branch *Feature32* has been created to support the development of a new feature. Two commits, C5 and C6, have been made to this branch. In the meantime, additional changes have been committed in the *master* branch (commit *C4*).

"It is easy to shoot your foot off with Git, but also **easy to revert to a previous foot and merge it with your current leg**."

Jack William Bell

Figure 198: Git Flexibility - A Nice Metaphor

Git Integration within the SAP Web IDE for SAP HANA

Git Clients

Since its creation in 2005, Git provides its complete set of features through the command line. Over time, a number of GUIs have been developed. They generally include a subset of the Git functionality available through the command line.

The SAP Web IDE for SAP HANA provides an embedded GUI for Git.

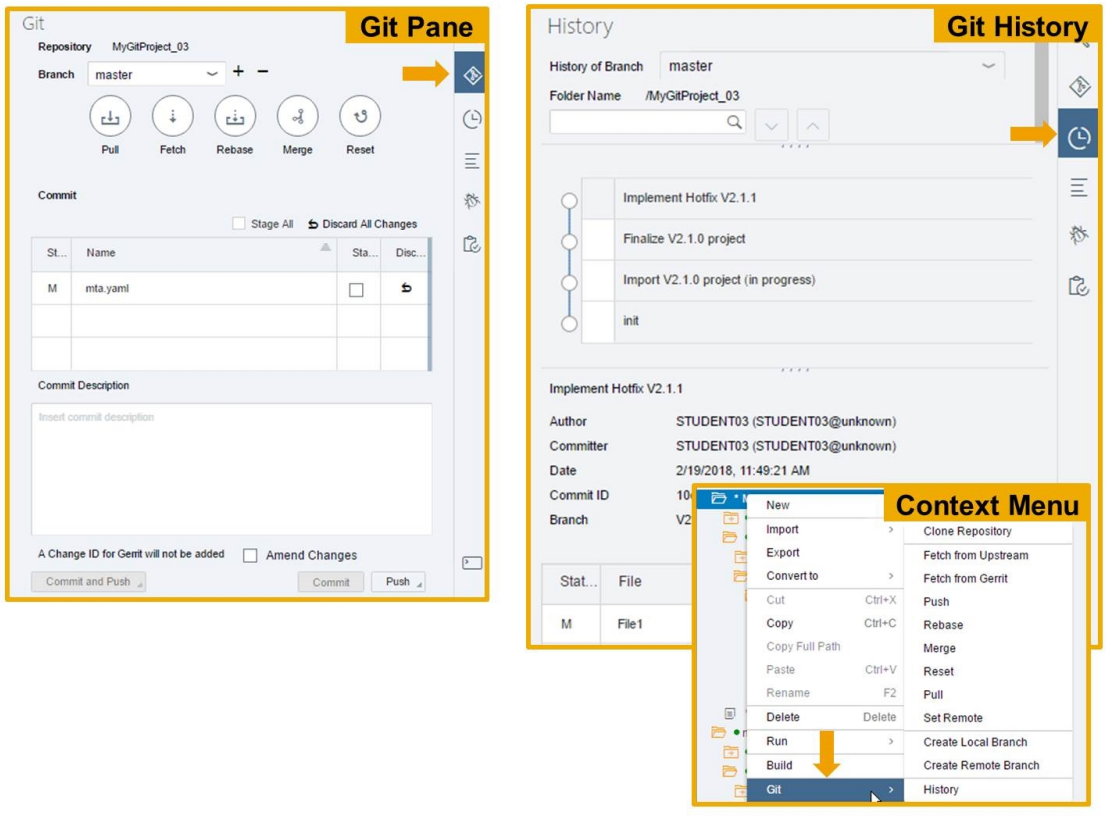


Figure 199: The Embedded Git Client in the SAP Web IDE for SAP HANA

This client includes two main Git panes, accessible from the right toolbar of the *Development* perspective, or the *View* menu, as well as a context menu and a set of icons displayed next to the files and folders in the workspace.

In addition, in the Project Settings, a dedicated section *Git repository configuration* displays the entries of the Git configuration file. These entries include, among others, the Git user name and e-mail, remote repositories (if any), and so on.

Note:
It is possible to edit the Git configuration file manually, but this requires advanced Git knowledge. In many cases, this is done automatically, based on the actions you perform in the GUI.

The SAP Web IDE for SAP HANA includes the core features of Git (commit, history, management of local and remote branches, and so on). The support of additional Git features is included on a regular basis in some new releases of the SAP Web IDE for SAP HANA. For example, the Git *Stash* feature (allowing to set aside uncommitted changes temporarily) will be included in the SAP Web IDE for SAP HANA 2.0 SPS03.

Starting to Use Git in the SAP Web IDE for SAP HANA

Working with Git in the SAP Web IDE for SAP HANA is something you decide project by project. One project might have Git functionality enabled, while another project does not.

If you decide to activate the Git functionality in the SAP Web IDE for SAP HANA, there are different approaches you can take. You will choose the one that is best suited to your needs, keeping in mind that in some use cases, they get the same result.

- Clone a Remote Git Repository

  This is an easy way to copy an existing project stored on a Git server (or Git hosting service), and if needed, start contributing to the project.

  In some cases, you will not contribute to the project, but you just want to get a copy of a project that a development team made available to you, either because you have read access to the remote repository, or because it is available publicly. This is the case of the SHINE for XS Advanced project, which anybody can clone without any credentials.

- Initialize a Local Repository

  By default, a project that you create or import in the SAP Web IDE does not have the Git functionality activated.

  However, activating Git for a project is extremely quick and simple: via the context menu of the project, you just tell the SAP Web IDE for SAP HANA to define the folder where your project is stored as a Git repository. Basically, this adds to this folder a few files that Git will then use to track the history, store snapshots of changes to the repository, and manage settings.

  Initializing a local repository is what you need when you want to work with Git, but alone (locally), on a project. That is, without sharing it or having other developers contribute to it (at least at some point). Indeed, you do not need a remote Git repository at all, in this case.

  Note:
  When you initialize a local repository (which results in an empty initial commit), all your project files are flagged as new and are staged. You are ready to add all your project files to the local Git repository by executing a commit.

- Set Remote Repository

This is what you need to associate your local Git repository (project) with a remote Git repository. For example, when you have already developed content locally and would like to share this content with other developers.

Caution:
This is NOT needed when you clone a remote repository, because it that case the local repository (the clone) is automatically associated to the remote repository you cloned from.

Whenever you work with a remote Git repository, you need the Git Clone URL. In addition, you might need credentials to access the repository if it is not public.

Working with Files in a Git Project

Modifying your project content (with or without Git) means creating, modifying, and deleting files. With Git, all of these changes are tracked in your working directory as soon as they are made, for example, when you save a file you have just modified and listed in the *File Status* area.

Then, for each file, you have the following possibilities:

- Stage the modification so that it will be included in the next commit
- Leave the modification unstaged (it will not be included in the next commit)
- Discard the change

Staging or discarding can also be done for the entire set of modifications.

The next step is to commit your changes, which will add a next commit to the history of the branch.

Note:
The concept of branches in Git, already introduced, will be discussed in more details later on. For now, let's just consider that you are working on a single local branch, for example, *master*.

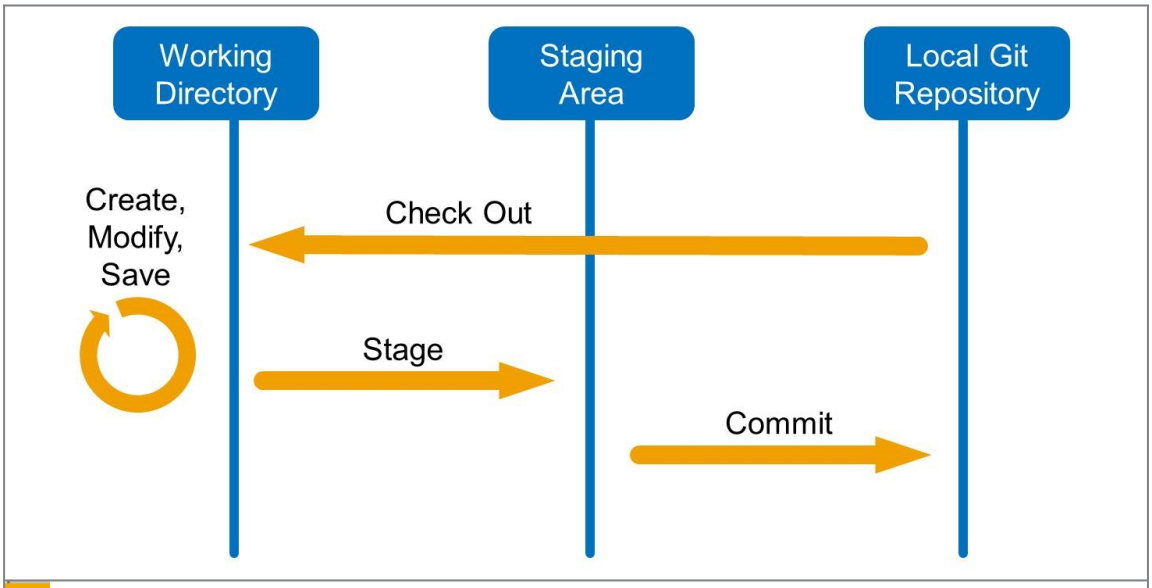


Figure 200: The Basic Git Workflow

To materialize this workflow, each file is assigned a Git status. This status is represented by an icon in the SAP Web IDE for SAP HANA workspace and/or the *File Status* area of the *Git* pane. The list of possible file statuses is as follows:

Workspace Icon *	Meaning	File Status in the Status Table
+	New, not staged	Untracked (U)
*	Modified, not staged	Modified (M)
**	New/modified and staged	New (N)
		Modified (M)
- **	Deleted (staged or unstaged)	Deleted (D)
●	Commited	[File is no longer listed]
)(	Conflicting	Conflict (C) – Only during a merge or rebase operation

* Only one icon is displayed for folders that contain files with different statuses
**The icon identifies folders from which files have been deleted

Figure 201: Git Status of Files in the SAP Web IDE for SAP HANA

Note:
The Conflict (C) status will be discussed later on.

Committing Changes

When you commit changes, you add these changes to the Git history and provide a commit description. From this moment on, the *Git History* pane will include this commit, and provide the identity of who committed, the date, the commit description, and details on which files were affected by the commit. Note that committing changes does not modify your working directory. The committed files are still available for further modification, and the new or modified files you haven't committed yet are still here for an upcoming commit. You can also discard the changes to these uncommitted files.

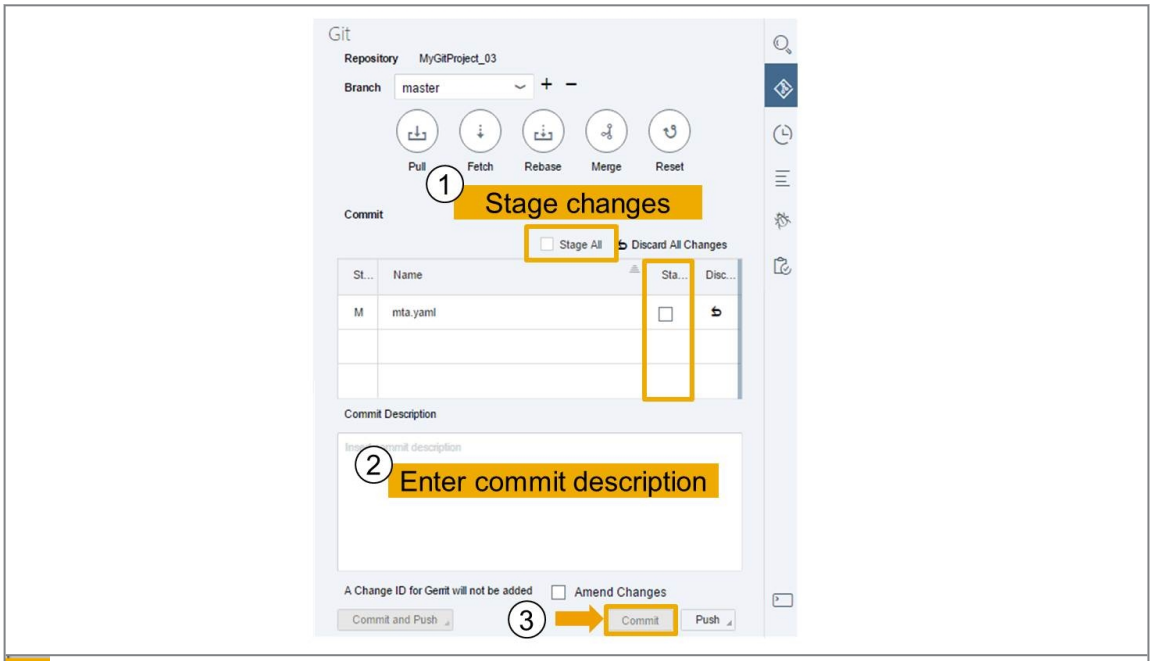


Figure 202: Committing Changes

The commit description provides important information to allow the readability of your changes, in particular when you collaborate with other developers. You can find a number of blogs and tutorials on how to write a good commit message. The following are recommendations for writing a commit message:

- Start with a relatively short subject (50 characters max), using imperative mood
- Separate the subject and body with a blank line
- Provide info about what the change does, and why (for example, what issue it solves)
- If relevant, provide the URL of a related issue or specification in another system (for example, JIRA)

It is possible to **amend** a previous commit. This allows you to replace the very last commit by a new one, after you have staged additional files. The original commit description is displayed so that you can modify it before committing again.

Caution:
You must not amend commits that have been shared with other developers, because this would modify a (shared) history on which others might have already based their work.

Working with Git Branches in the SAP Web IDE for SAP HANA

Local or Remote Branch

Git allows you to create both local and remote branches. So, what branch type should you choose?

One key principle is that only remote branches are visible to others: your local branches are for you and you are the only one who works in these branches.

Another important principle in Git is that you never commit changes directly in a remote branch. Instead, you always commit changes in one of your local branches first. Then, if needed, you can transport the commits to a remote branch.

> Example:You are working on a project with other developers. At some point, you need to test something on your own, without sharing it with other developers. Then you will create a local branch for that. If, later on, you need to share this with others, you have the possibility to create a remote branch based on your local branch.

This course will cover remote branches later on. But for now, let's discuss how you work with local branches.

Working with Local Branches

You can create a new local branch from the *Git* pane or the Git context menu. The only thing you need is to define a source branch (which could be remote or local) and give the new branch a name. Just after a new branch is created, it is absolutely identical to the source branch.

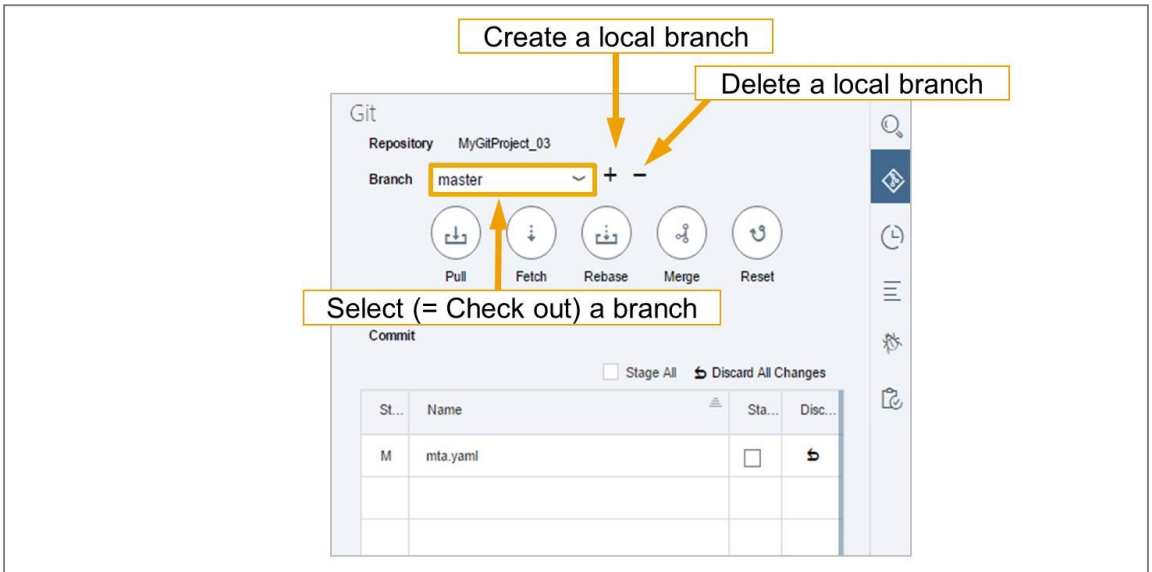


Figure 203: Working with Local Branches

When you have several branches in your project, you must carefully decide to which branch you want to commit your changes to. To do this, you select a branch in the *Git* pane. In Git terms, this is known as *Check Out*.

Checking out a branch also updates your working directory to reflect the exact current state of files in this branch. In other words, your working directory is likely to change when you switch from a branch to another, except if these branches are identical, that is, if they point to the same commit.

Caution:
Issues might occur when you have changes that are not committed yet and you try to check out another branch. For example, you have made a modification to a file that exists in both branches. To avoid this, the best way is to have a clean working directory without uncommitted modifications before you check out a branch.

Combining Changes from Two Branches

When changes affect several local branches, you might want to combine these changes. For example, to include a hotfix or a new feature in your master.

Let's assume that commits have been executed in two different branches, *master* and *feature32*.

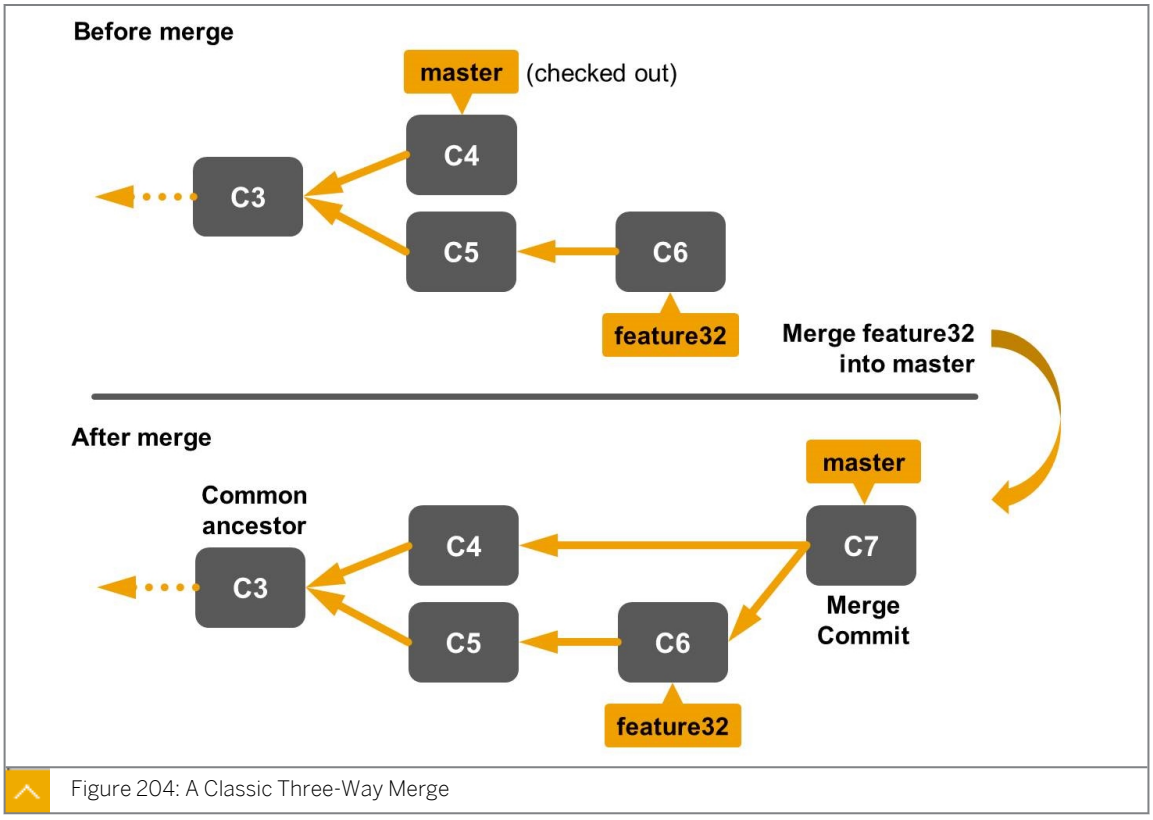


Figure 204: A Classic Three-Way Merge

A Git *merge* operation affects the branch that is currently checked out. Git creates a new commit (merge commit) that will combine the changes of another branch with the changes of the checked out branch since their history diverged. As you see in the figure above, the merge commit has two parents, which are the last commits of the two branches that you merged together.

Note:
After the *merge*, the two branches are not identical. In particular, *feature32* does not include *C4*.

A special case for Git merge is when the branch you want to merge into (the checked out branch) points to a commit that exists in the branch you want to merge. Meaning, no additional commit has been made to that branch.

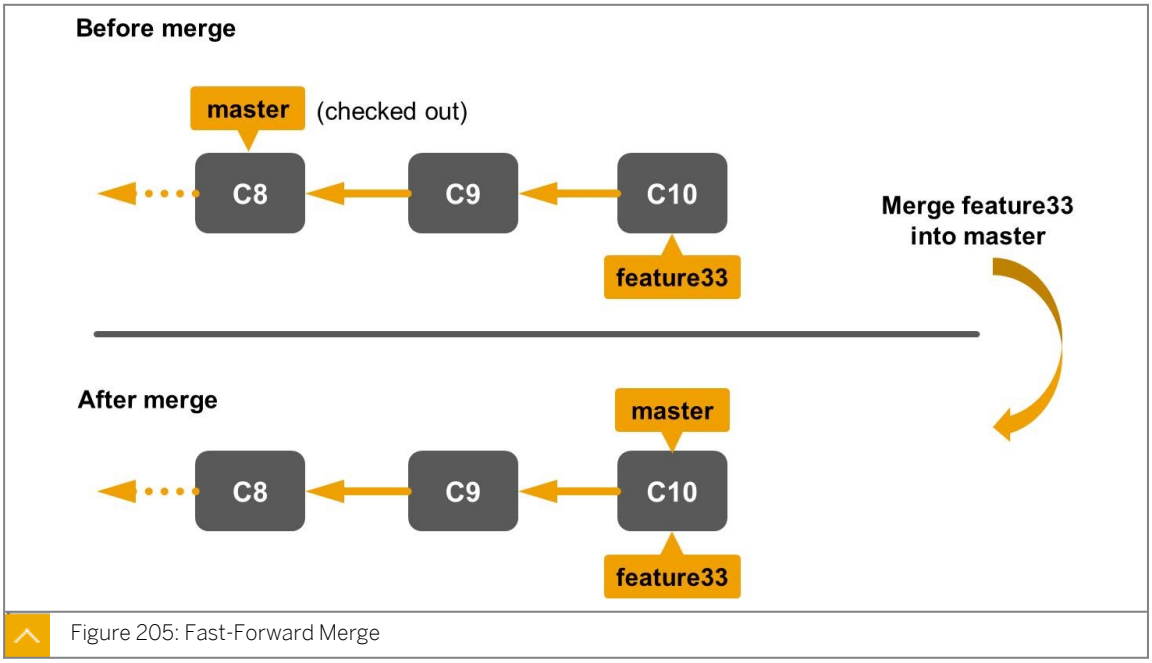


Figure 205: Fast-Forward Merge

In this case, there is no need for an additional merge commit. Indeed, the pointer of the checked out branch is just pushed forward to the last commit of the other branch. Git calls this a fast-forward merge.

Note:
A fast-forward merge only works if you check out the branch that is behind in the commit history. For example, in the scenario shown in the figure above, if you check out the branch *feature33* and execute a “merge master into feature33”, no change will be made at all.

Now, let’s introduce an alternative to merging when you want to combine changes from two branches. It is called *Rebase*.

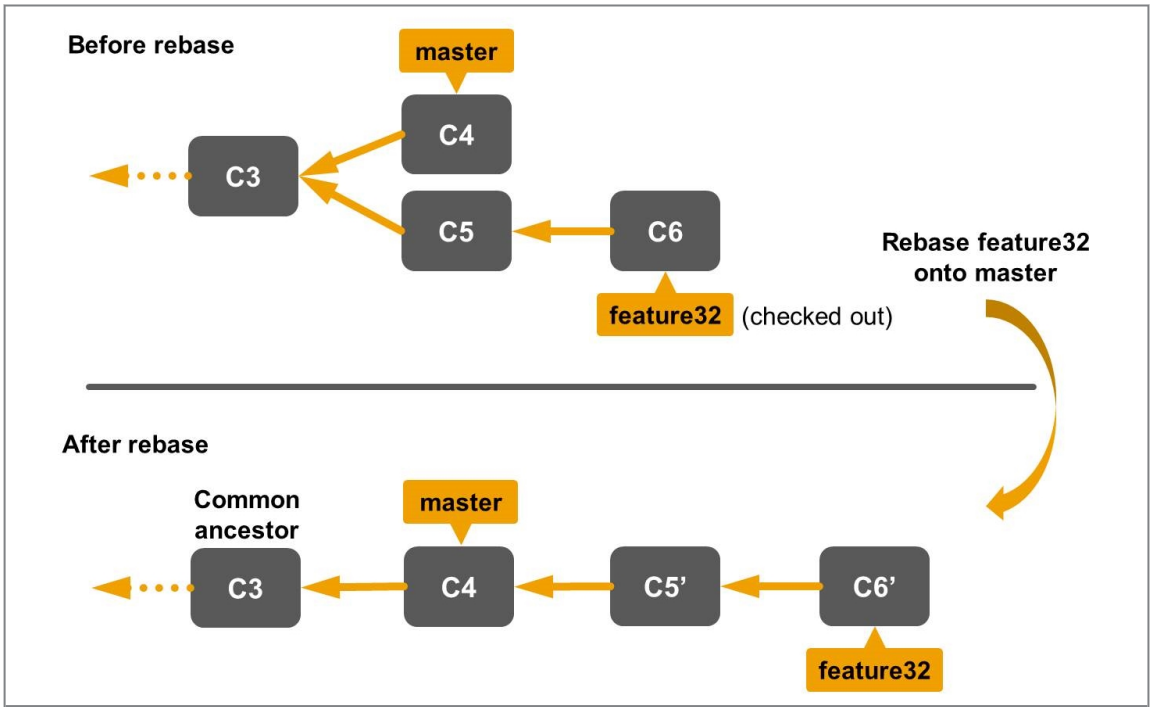


Figure 206: Rebasing

When you rebase branch B2 (here, *feature32*) onto branch B1 (here, *master*), you basically take the history of commits in B2 and replay these commits on top of B1. Then, you can execute a fast-forward merge on B1 (*master*) so that it points to the same commit as B2 (*feature32*).

As you see, compared with the classic three-way merge we discussed earlier, this keeps the history of the *master* branch linear. The final content of the master branch is identical, but the history looks different.

Rebasing is sometimes useful when you work on a purely local project, and also when you want to prepare a clean commit history that will flow naturally on top of a branch to which you contribute.

Caution:
You should always keep in mind the golden rule to use rebase: **You can only rebase changes that you have NOT shared with anyone**. Indeed, a commit that is communicated externally, for example, by pushing changes to a remote Git server, might be the starting point of some work from other developers. If you rebase your changes and push them again to this Git server, this will alter the history and might cause a lot of issues.

Managing Merge and Rebase Conflicts

A conflict occurs during a merge or rebase when one or several files are affected by concurrent modifications from both branches.

In this case, you must analyze the files and determine how to solve the conflict. Which means, decide how to produce a code that suits the requirements that the changes in each branch addressed.

Each conflicting file is identified with a dedicated red icon **> <** in the workspace (project tree). To solve a conflict, you must open the file, in which the contents of the two branches are presented together with a special markup to identify which changes come from which branch.

Example of a Merge Conflict in a Table Function

```
select
"FIRST_NAME",
"LAST_NAME",
"SEX",
"LANGUAGE"
 from "HA300::SNWD_EMPLOYEES"
<<<<<<< HEAD
 where contains("LAST_NAME", :lastNameFilter, FUZZY(0.55));
=======
 where contains("LAST_NAME", :lastNameFilter, FUZZY(0.6));
>>>>>>> hotfix
```

HEAD is the pointer to the checked out branch (the one you're merging into) and, here, *hotfix* is the name of the branch you're merging.

Example of a Rebase Conflict in a Table Function

```
select
"FIRST_NAME",
"LAST_NAME",
"SEX",
"LANGUAGE"
 from "HA300::SNWD_EMPLOYEES"
<<<<<<< Upstream, based on master
 where contains("LAST_NAME", :lastNameFilter, FUZZY(0.55));
=======
 where contains("LAST_NAME", :lastNameFilter, FUZZY(0.6));
>>>>>>> 93f6499 Change fuzziness threshold for employee search function
```

In a rebase, the branch on top of which you want to rebase is identified with *Upstream, based on <branch name>* and the specific commit from the branch that you are rebasing is listed, with its description.

In both cases, you must adapt the code, remove the specific markup, and sometimes include comment lines in your code to explain how you solved the conflict, and then save the file.

When this has been done for each conflicting file, you can then proceed as follows:

- For a Merge: stage the modified file(s), enter a commit description and commit your changes.

  This description will replace the default merge commit description (the one that you would get if no conflict occurred).

- For a Rebase: choose *Continue*.

  In the rebased branch history, the conflict resolution is stored in the "replayed" commit. As if there had not been a concurrent change, or to put it another way, as if you had made your change in the *hotfix* branch based on the modified version from the *master* branch.

Working with Remote Branches

Working with remote branches is not fundamentally different from working with local branches. First of all, because the changes you make to your files are always committed to a local branch, and secondly, because combining the changes from a local branch and a remote one works in a similar way as combining the changes from two local branches.

Still, there are a few differences, such as:

- You need to connect to a remote Git server (or a Git hosting service).
- At some point, you will share your changes (commits) with others, and also receive changes (commits) from others.

Connecting to a Remote Git Repository

As discussed earlier, in the section, Starting to Use Git in the SAP Web IDE for SAP HANA, there are two ways to connect to a remote Git repository:

- Clone from a remote Git repository
- Set a remote (link your local Git repository to a remote one)

An essential piece of information you need for that is the **remote repository URL**.

In addition, cloning, setting a remote, and then, most of the regular interactions with a remote repository, require **credentials**.

However, some repositories are public, which means that they can be accessed anonymously, without credentials. For example, this allows you to clone in your repository some code that the owners have decided to share publicly.

For security purposes, on the remote Git server side, there is a repository administrator who manages users and authorizations. For example, you might be allowed to see a repository but not to write to it (Guest role). Or some branches might be protected, and only a user with role "Branch Master" can write to these branches.

Interacting with a Remote Git Repository

- Clone from a remote Git repository

  If you clone a remote repository, your local Git automatically knows about the branches it contains. By default, it creates the local branch *master* which, at first, corresponds to the remote *master* branch.

- Set a remote (link your local Git repository to a remote one)

  In this case, the first thing you need to do is to tell Git to *Fetch* info from the remote repository. For now, let's say that a fetch allows the Git client to know about the branches of the remote repository.

In situations where you connect to a brand new remote repository, you should be aware that some Git hosting services allow the repository creator to include an initial commit (an empty one, or one with just a readme file) or not. Connecting to an empty repository will generally work better if there is an initial commit.

Now, let's discuss what you might want to do.

Copying Branches

You can create a local or remote branch. In any case, you need to specify a source branch.

Table 3: Creating Branches from Other Branches

Task	Purpose / Example	Steps
Create a local branch from a remote branch	You want to contribute to a project, starting from an existing remote branch you don't have.	*Git Pane*:*Create local branch*. Choose the remote branch (source) and define the name of the local branch

Task	Purpose / Example	Steps
Create a remote branch from a local branch	You have created code that you want to share with others developers but without including it in an existing remote branch.	*Workspace:* Git Context Menu: *Create Remote Branch*. Choose the local branch (source) and define the name of the new remote branch.
Create a remote branch from a remote branch	You want to keep in a branch *V315* the current status of the *dev* branch, which has just been approved for release	*Workspace:* Git Context Menu: *Create Remote Branch*. Choose the source remote branch *(dev)* and define the name of the new remote branch *(V315)*.

Creating a new branch is a bit like copying a branch, but technically, you do not duplicate the entire set of code that the source branch contains. You only create a new pointer, and if needed, Git only transports the changes that correspond to commits that the repository of the new branch does not know about.

Caution:
When creating a local branch from a remote one, you must first execute a fetch to get the very last version of the remote branch.

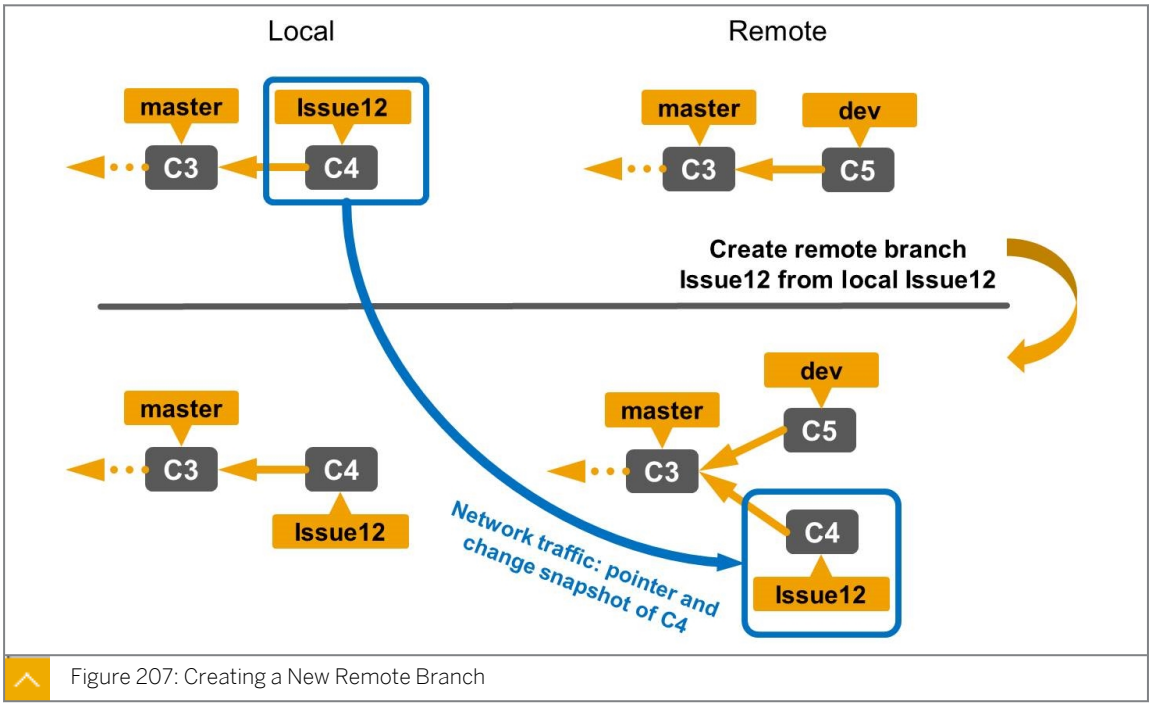


Figure 207: Creating a New Remote Branch

Consider the example in the figure above, suppose you have been asked to develop a hotfix for issue #12. You need to develop the fix on top of the productive version (master). When the fix is ready, or even before, if you need to discuss it with a colleague, you can copy the corresponding branch *Issue12* to the remote repository.

Pushing and Fetching

Now, let's discuss *Fetch* and *Push*. To keep it simple, let's say that these commands only work between a local repository and a remote one. These are the two essential commands to send

local content to a remote repository (push) and get remote content into your local repository (fetch).

Pushing

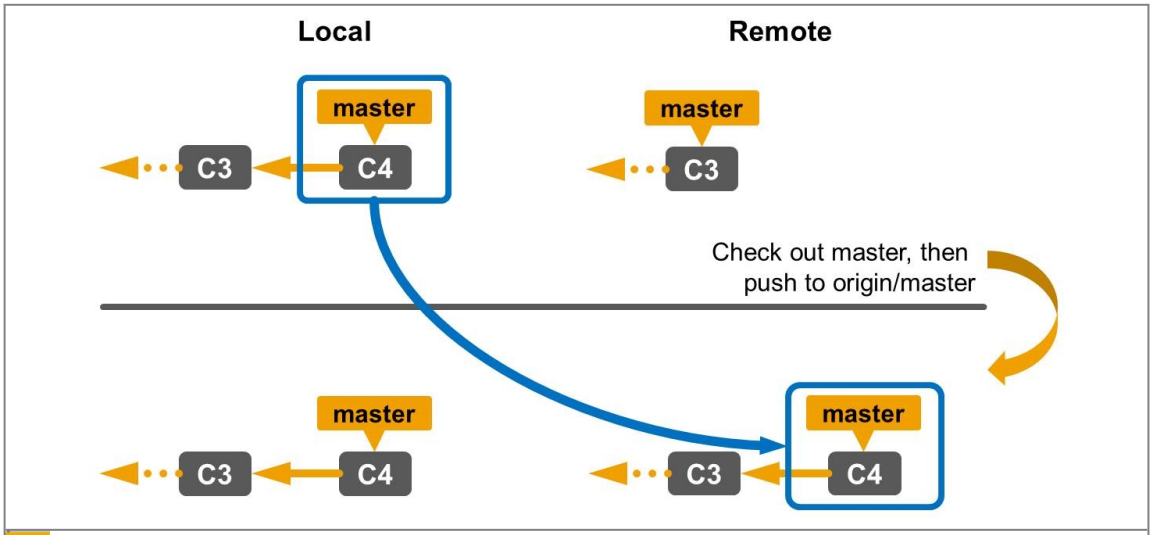


Figure 208: Pushing

Pushing means that you send commits you have in a local branch to a remote one.

> Note:
> These might be commits of changes that you have made, or commits from another developer that you have just merged into your local branch after validating them.

To do that, you check out the branch you want to push changes from, and you define which branch you want to push to.

By default, Git allows you to push only if the last commit of the remote branch is already in your local branch (this allows the equivalent of a fast-forward merge of your change into the remote repository). If not, you will have to fetch the remote branch history, and then rebase your changes onto the remote branch. Only then, you will be able to perform a successful push.

> Note:
> The *Commit and Push* button in the SAP Web IDE for SAP HANA triggers first a commit to your current (checked out) local branch, and then a push to a remote branch.

Fetching

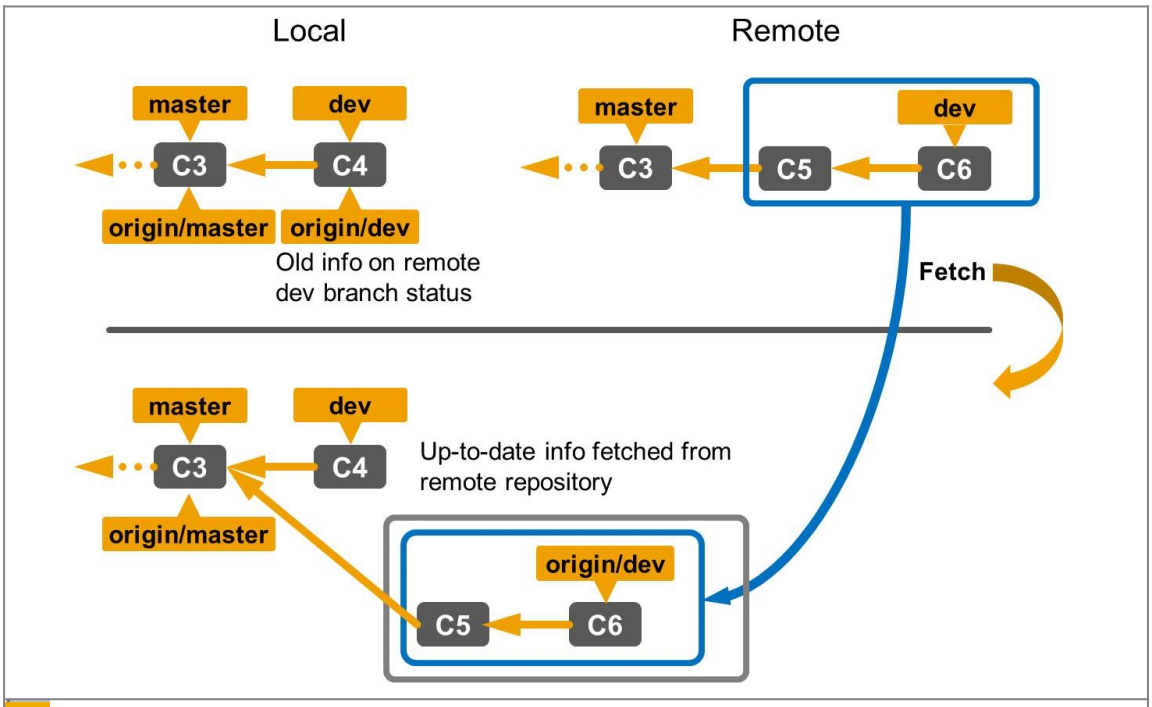


Figure 209: Fetching

With a Fetch, Git downloads from a remote repository the commits and branch pointers. This does not change your working directory or your local branches, but it creates (or adjusts) locally references and change snapshots from the remote branches. This allows you to inspect the commits that occurred on remote branches (date, description, who committed) and, later on, to merge these changes into a local branch, or rebase a local branch onto the remote branch.

Note:
Fetching is also required if you want to get the list of branches available on a remote repository. In particular, if someone else created a new branch on the remote repository, you cannot push commits to this branch before you know it exists.

In the figure above, we introduce two additional branch pointers: *origin/dev* and (for the sake of consistency) *origin/master*. These are the local pointers to remote branches. They are used to store locally the state of remote branches when you fetch. Then, any merge, rebase, or new branch creation involving a remote branch is executed locally based on these pointers (and the commits they reference). They materialize what Git calls *Remote-Tracking Branches*. That is, local branches that you cannot commit changes into, but are aimed at keeping locally, offline, a recent status of remote branches.

By default, for a given repository that your local repository knows about, Git will creates (or update) during a fetch one (local) remote-tracking branch per remote branch. The default naming convention for a remote repository is *origin*, and if a branch in this remote is called for instance *dev*, the corresponding local-tracking branch will be called *origin/dev*. If you want to work on a local copy of this branch, Git will propose the default name *dev* but you can choose another one.

After you fetch, you can see how many commits you are behind of the current remote branch commit. This is displayed in a box next to the current branch name. For example, you might have made 2 commits locally after cloning the remote repository (you do not need to fetch to know this), while someone has pushed 3 commits to the same remote (you will need to fetch to get this info). In this case, you would say you are “2 commits ahead and 3 commits behind”.

Pulling

Pulling consists of a fetch, immediately followed by a merge. It can be useful in some cases

Resetting a Branch

In the SAP Web IDE for SAP HANA, it is possible to reset a local branch, which means, to revert this branch to the state of another local or remote branch. In case you reset based on a remote branch, always fetch before resetting, to make sure you get the very last changes of the remote branch in your local repository.

Two reset modes are proposed:

- MIXED (HEAD and index updated)

  This reset mode keeps your working directory as is and un-stages the changes. Which means, you can stage and commit again all the changes (or part of them) that the reset rolled back.

- HARD (HEAD, index and working directory updated)

  With this reset mode, the commits that the reset rolled back are discarded, even in the working directory.

LESSON SUMMARY
You should now be able to:

- Use the Native Git Integration of the SAP Web IDE for SAP HANA

Learning Assessment

1. What is Git?

   Choose the correct answer.

   - [ ] **A** A tool for automated code testing
   - [ ] **B** A tool for static code analysis
   - [ ] **C** A tool for web-based team code collaboration
   - [ ] **D** A tool for tracking changes in source code during software development

Learning Assessment - Answers

1. What is Git?

   Choose the correct answer.

   - [ ] A A tool for automated code testing
   - [ ] B A tool for static code analysis
   - [ ] C A tool for web-based team code collaboration
   - [x] D A tool for tracking changes in source code during software development

   That is correct! Git is a tool for tracking changes in source code during software development.