DevOps for z/OS on Azure

A Test Drive guide for using the
Kobee for z/OS solution on Microsoft Azure

Table of contents

Summary

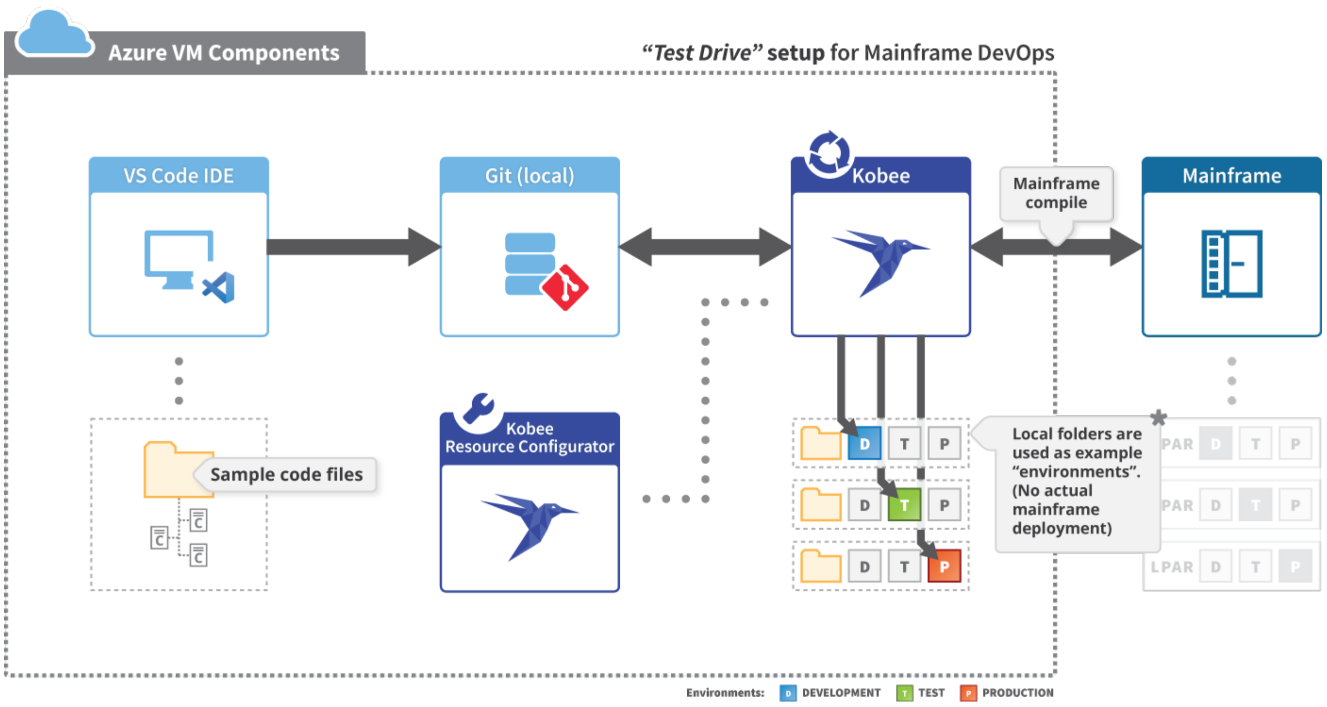
This document will guide you through your "Test Drive" of DevOps for z/OS on Microsoft Azure.

Kobee is the actual DevOps software product on this "Test Drive" virtual machine that you will be working with. It is preconfigured and set up with a predefined z/OS project so you can start right away.

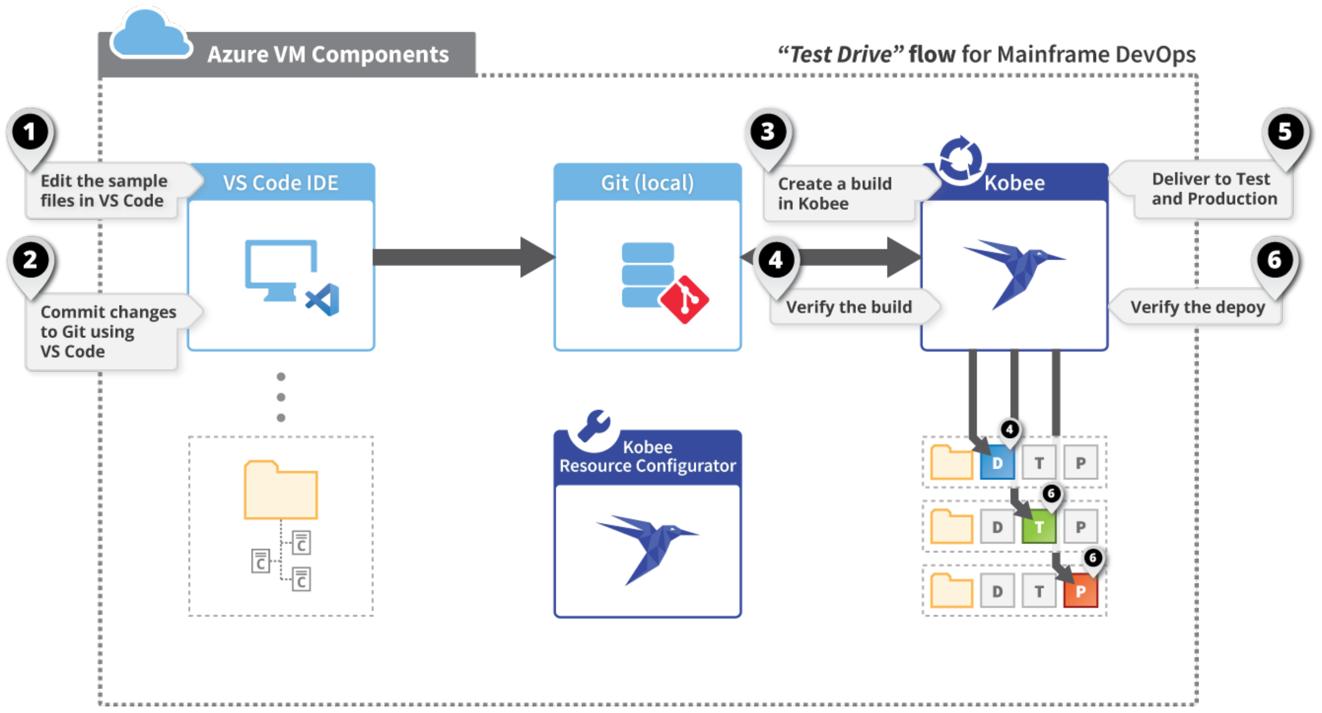
The purpose of this "Test Drive" is to provide you the experience of a standard Kobee user. Such a user is able to create packages, and launch mainframe compiles (builds) and promotes (deploys). Global administration and project administration is beyond the scope of this "Test Drive".

First let's go through everything that we have made available for you

- **Sample (mainframe) code** which you can customize.
- **Visual Studio Code** IDE (which has Git integration to version your customized files).
- A local file-based **Git** repository.
- **Kobee** preconfigured and with a **z/OS demo project.**
- **Kobee Resource Configurator (KRC)** to edit the z/OS configuration.


Azure VM Components
"Test Drive" setup for Mainframe DevOps
VS Code IDE
Git (local)
Kobee
Mainframe compile
Mainframe
Sample code files
Kobee Resource Configurator
D T P
Local folders are used as example "environments". (No actual mainframe deployment)
LPAR
Environments: D DEVELOPMENT T TEST P PRODUCTION

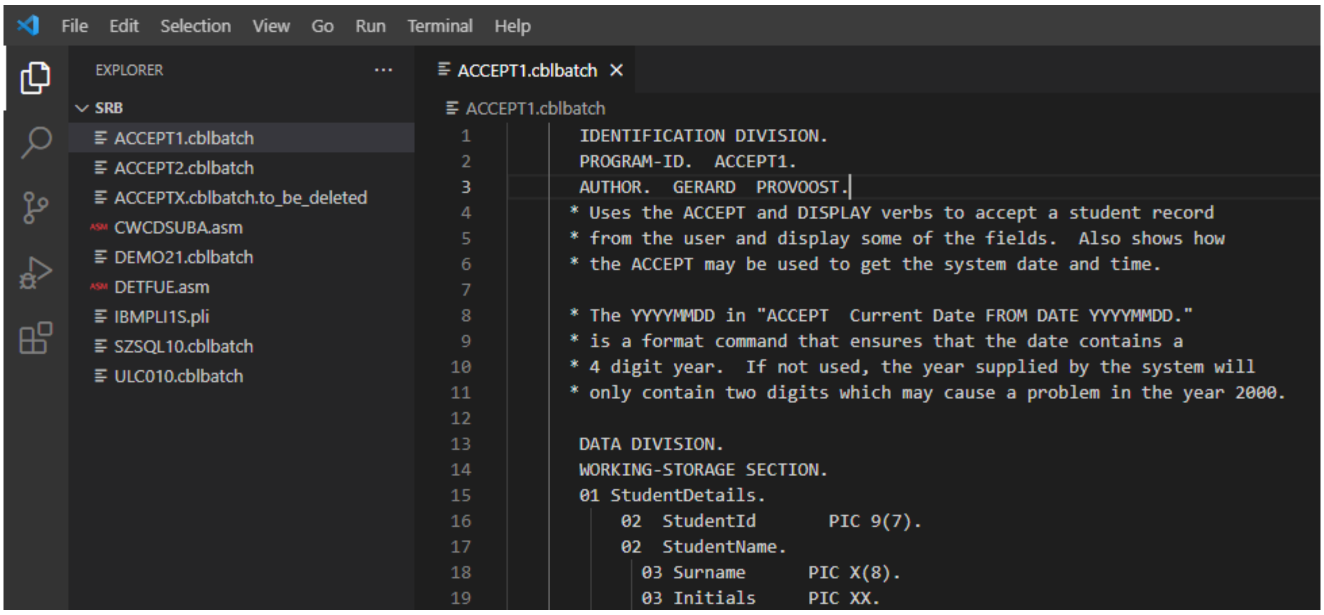
very basic scenario that you can follow during the “Test Drive”.


Azure VM Components
“Test Drive” flow for Mainframe DevOps
1 Edit the sample files in VS Code
2 Commit changes to Git using VS Code
VS Code IDE
Git (local)
3 Create a build in Kobee
4 Verify the build
Kobee
5 Deliver to Test and Production
6 Verify the depoy
Kobee Resource Configurator
D T P

Making changes to the samples and committing to Git

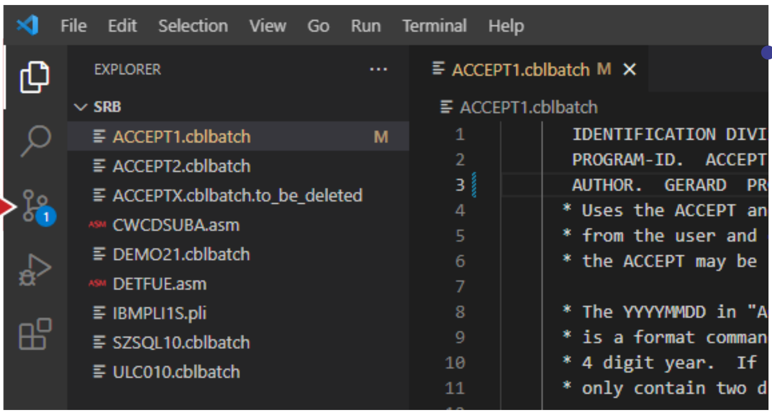
We’re using VS Code, but any IDE or editor will suffice. The only requirement for Kobee to work is that your files are versioned, in this case we chose Git as version control repository.

Open VS Code by clicking on the desktop icon. You will notice some files have already been opened and the folder in which these files reside are located in the cloned Git repository (location: C:\ikan\workspace\demozos), this is your workspace.



```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT1.
       AUTHOR.  GERARD  PROVOOST.
      * Uses the ACCEPT and DISPLAY verbs to accept a student record
      * from the user and display some of the fields.  Also shows how
      * the ACCEPT may be used to get the system date and time.

      * The YYYYMMDD in "ACCEPT  Current Date FROM DATE YYYYMMDD."
      * is a format command that ensures that the date contains a
      * 4 digit year.  If not used, the year supplied by the system will
      * only contain two digits which may cause a problem in the year 2000.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 StudentDetails.
           02  StudentId       PIC 9(7).
           02  StudentName.
             03 Surname      PIC X(8).
             03 Initials     PIC XX.
```

In the following steps we'll explain how to edit and commit the sample files.



```
 IDENTIFICATION DIVI
 PROGRAM-ID.  ACCEPT
 AUTHOR.  GERARD  PR
* Uses the ACCEPT an
* from the user and
* the ACCEPT may be

* The YYYYMMDD in "A
* is a format comman
* 4 digit year.  If
* only contain two d
```

Edit the sample code

First edit the "ACCEPT1.cblbatch" file, you could for instance just add some extra characters.

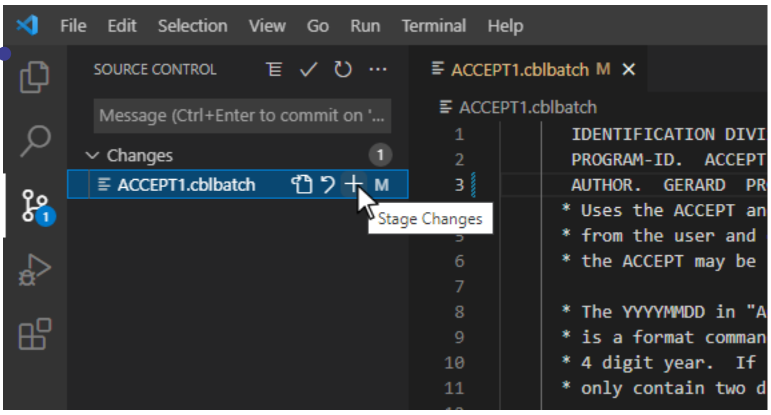
Save the file. You will see the color of the file in the explorer change and "M" letter indicates it has been modified.

Next, go to the "Source Control" view (see the red arrow).

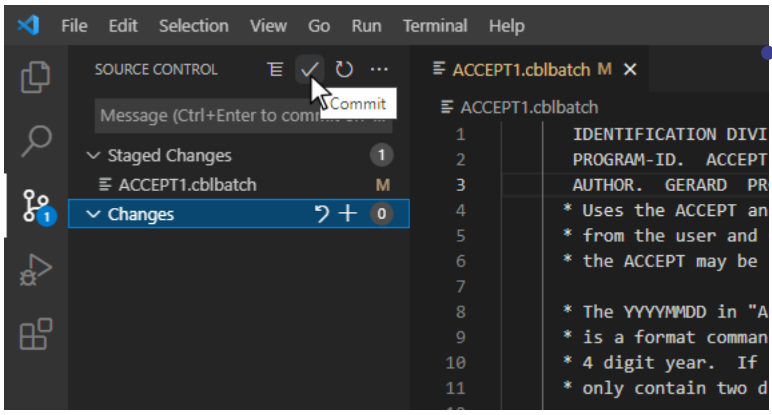
Stage the changes

In Git you need to stage you changes before you can commit.

Press the "plus sign" to stage you changes.


File Edit Selection View Go Run Terminal Help
SOURCE CONTROL
Message (Ctrl+Enter to commit on '...
Changes 1
ACCEPT1.cblbatch M
Stage Changes
ACCEPT1.cblbatch M

```
 IDENTIFICATION DIVI
 PROGRAM-ID.  ACCEPT
 AUTHOR.  GERARD  PR
* Uses the ACCEPT an
* from the user and
* the ACCEPT may be

* The YYYYMMDD in "A
* is a format comman
* 4 digit year.  If
* only contain two d
```



```
 IDENTIFICATION DIVI
 PROGRAM-ID.  ACCEPT
 AUTHOR.  GERARD  PR
* Uses the ACCEPT an
* from the user and
* the ACCEPT may be

* The YYYYMMDD in "A
* is a format comman
* 4 digit year.  If
* only contain two d
```

Commit the changes

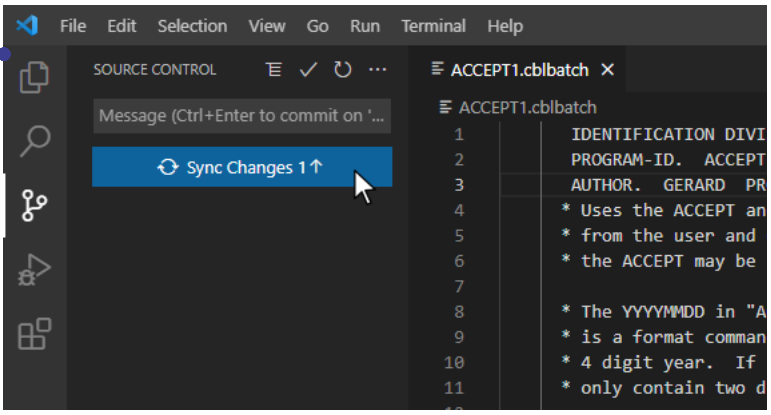
Click the "checkmark" to commit your changes.

You can enter your commit message in the box below first (before pressing the button), or you can do this afterwards in a pop-up.

Sync you changes

In Git you need to push the changes made in the local repository to the remote repository.

Press the "Sync Changes" button.


File Edit Selection View Go Run Terminal Help
SOURCE CONTROL
Message (Ctrl+Enter to commit on '...
Sync Changes 1↑
ACCEPT1.cblbatch

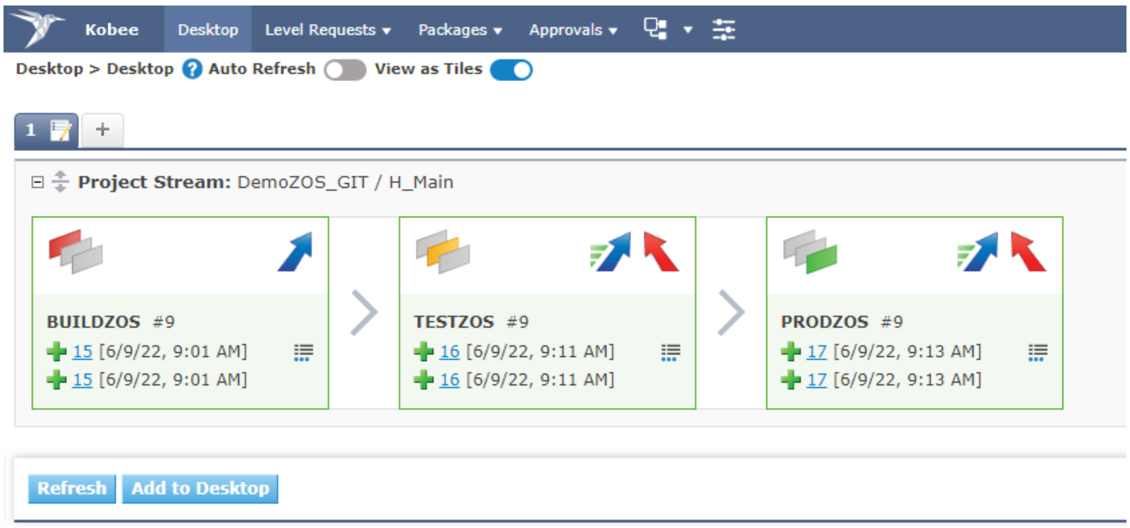
```
 IDENTIFICATION DIVI
 PROGRAM-ID.  ACCEPT
 AUTHOR.  GERARD  PR
* Uses the ACCEPT an
* from the user and
* the ACCEPT may be

* The YYYYMMDD in "A
* is a format comman
* 4 digit year.  If
* only contain two d
```

Creating a Package and compiling the sample code

Click the Kobee icon on your Windows desktop icon to open Kobee, and log in with the following credentials:

Username: user
Password: user

You will be taken to the Kobee Desktop screen where you can see the Project Stream of our z/OS Demo project: "DemoZOS_GIT".



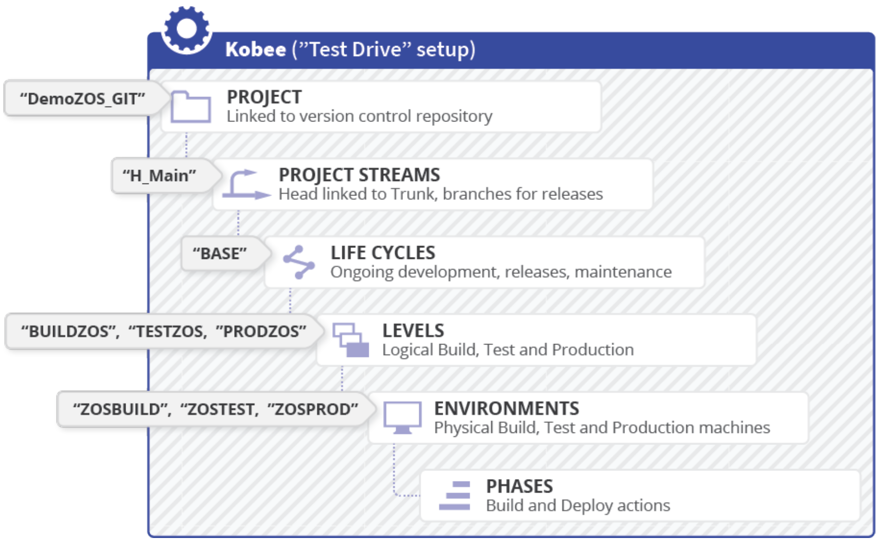
Project Stream: DemoZOS_GIT / H_Main

BUILDZOS #9	TESTZOS #9	PRODZOS #9
15 [6/9/22, 9:01 AM]	16 [6/9/22, 9:11 AM]	17 [6/9/22, 9:13 AM]
15 [6/9/22, 9:01 AM]	16 [6/9/22, 9:11 AM]	17 [6/9/22, 9:13 AM]

A Project Stream is a working entity within Kobee in which the lifecycles and their levels (Build, Test, Production) for our Demo project are defined. It is automatically created when we create a project in Kobee.

NOTE: For our project the default "Head" Project Stream is sufficient, for complex projects you can define additional "Branch" Project Streams (parallel development,…). You can read more about this in the Kobee User Guide.

In the image below you can see the Kobee hierarchy of our Demo project. We will get back to this topic in the second part of this document.



Kobee ("Test Drive" setup)
"DemoZOS_GIT"
PROJECT
Linked to version control repository
"H_Main"
PROJECT STREAMS
Head linked to Trunk, branches for releases
"BASE"
LIFE CYCLES
Ongoing development, releases, maintenance
"BUILDZOS", "TESTZOS, "PRODZOS"
LEVELS
Logical Build, Test and Production
"ZOSBUILD", "ZOSTEST, "ZOSPROD"
ENVIRONMENTS
Physical Build, Test and Production machines
PHASES
Build and Deploy actions

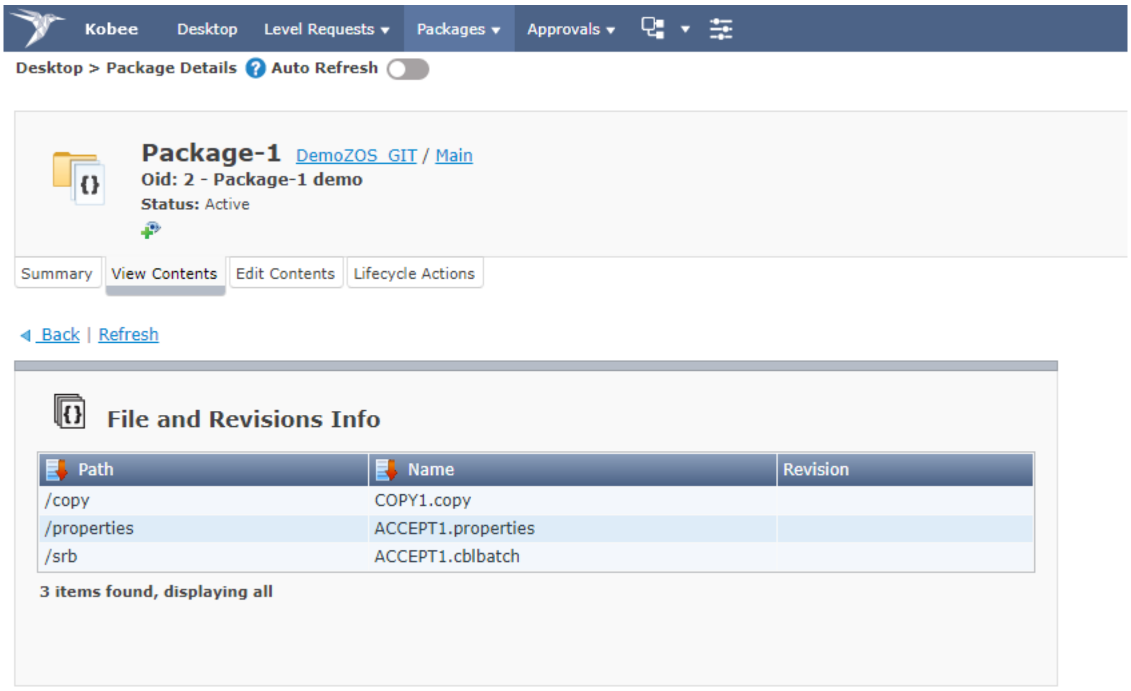
First let's look at what's inside the package that we are going to compile.

In the top menu click: *"Packages > Overview Packages"*

On this new screen click the "View" icon in the "Packages Overview" pane. This will take you to the "Package Details" screen.

As you can see under the "View Contents" tab, we currently have 3 files in our package:

- COPY1.copy
- ACCEPT1.properties
- ACCEPT1.cblbatch



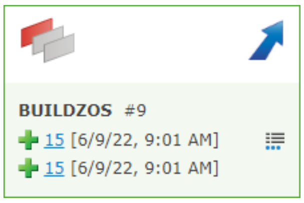
Path	Name	Revision
/copy	COPY1.copy	
/properties	ACCEPT1.properties	
/srb	ACCEPT1.cblbatch	

In case you have edited other files (previously in VS Code) besides the ones above, you will need to add them to the package by going to the "Edit Contents" tab, selecting your files and then hitting the "Save" button.

Now, let's go back to the Kobee Desktop; in the top menu click: *"Desktop"*.

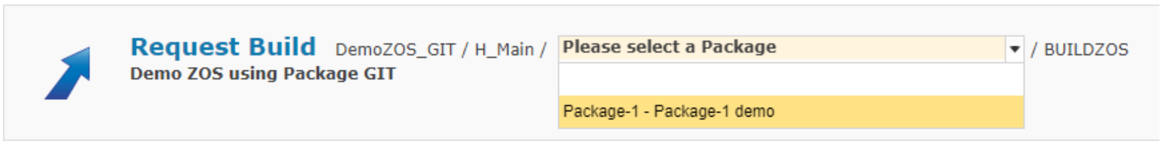
We are ready to build our Demo project. In Kobee initiating a build (or deploy) process comes down to starting a "Level Request".

In this case, we request Kobee to retrieve our package files and perform all the actions that are defined in the build level: "BUILDZOS". In Kobee, the build and deploy actions are handled by Phases, more on this later.

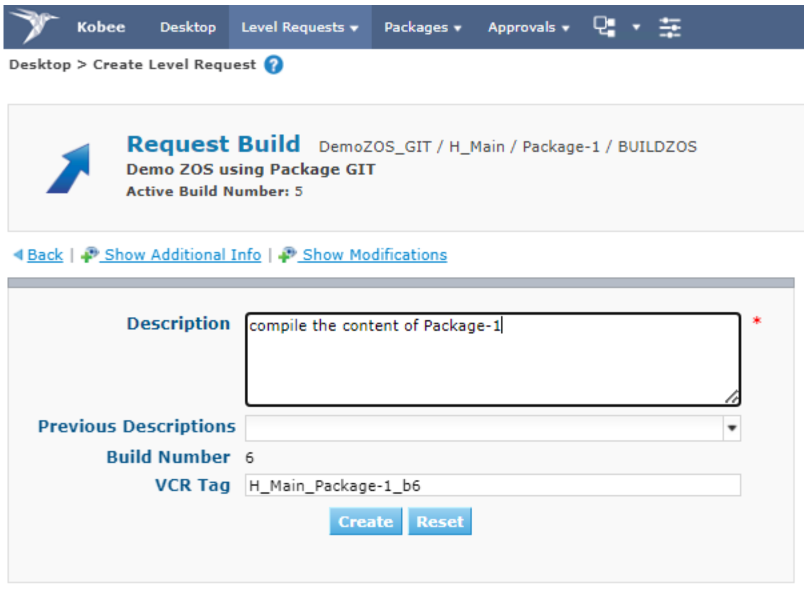


Click the *"Request"* icon in the "BUILDZOS" tile of the Project Stream.

Select your package "Package-1" first.



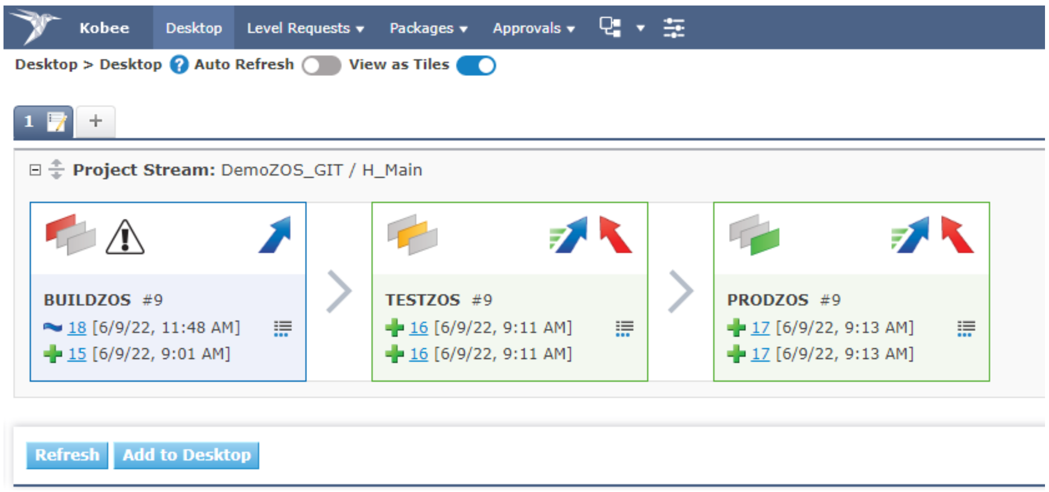
Provide a meaningful description, but do not modify the VCR Tag entry. This tag is automatically generated and will be created in Git when the build is successful.



Description	compile the content of Package-1
Previous Descriptions	
Build Number	6
VCR Tag	H_Main_Package-1_b6

NOTE: Optionally you can have a look at the sources we modified in our "Package-1" package by clicking the *"Show Modifications"* link.

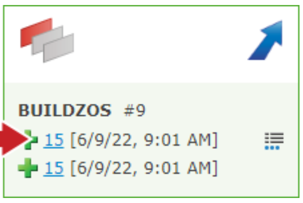
Finally, hit the *"Create"* button, to start the Level Request. You will be taken back to the Kobee Desktop. There you will notice that the Build Level tile has changed into a "running" state.



It's convenient to turn on the "Auto Refresh" switch (below the menu), so you don't have to refresh the page manually while you're waiting.

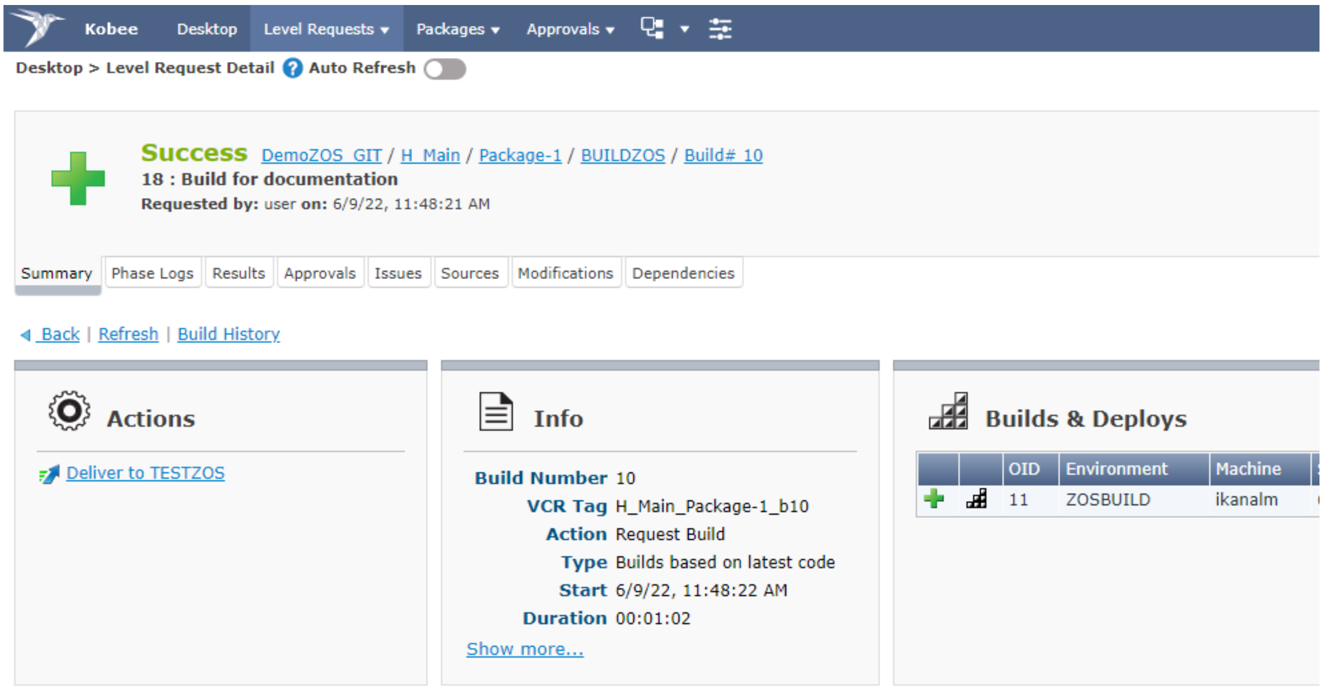
Verifying the result of the (compile) Build Level Request

When the request had finished, you will see an green icon indicating the request has been successful.


BUILDZOS #9
15 [6/9/22, 9:01 AM]
15 [6/9/22, 9:01 AM]

The first line indicates the current Level Request, the second line indicates the latest successful Level Request.

Click on the first "Level Request OID" link, next to this icon. This will take you to the "Level Request Detail" page.



Desktop > Level Request Detail ? Auto Refresh

Success DemoZOS_GIT / H_Main / Package-1 / BUILDZOS / Build# 10
18 : Build for documentation
Requested by: user **on:** 6/9/22, 11:48:21 AM

Actions

Deliver to TESTZOS

Info

Build Number	10
VCR Tag	H_Main_Package-1_b10
Action	Request Build
Type	Builds based on latest code
Start	6/9/22, 11:48:22 AM
Duration	00:01:02

Show more...

Builds & Deploys

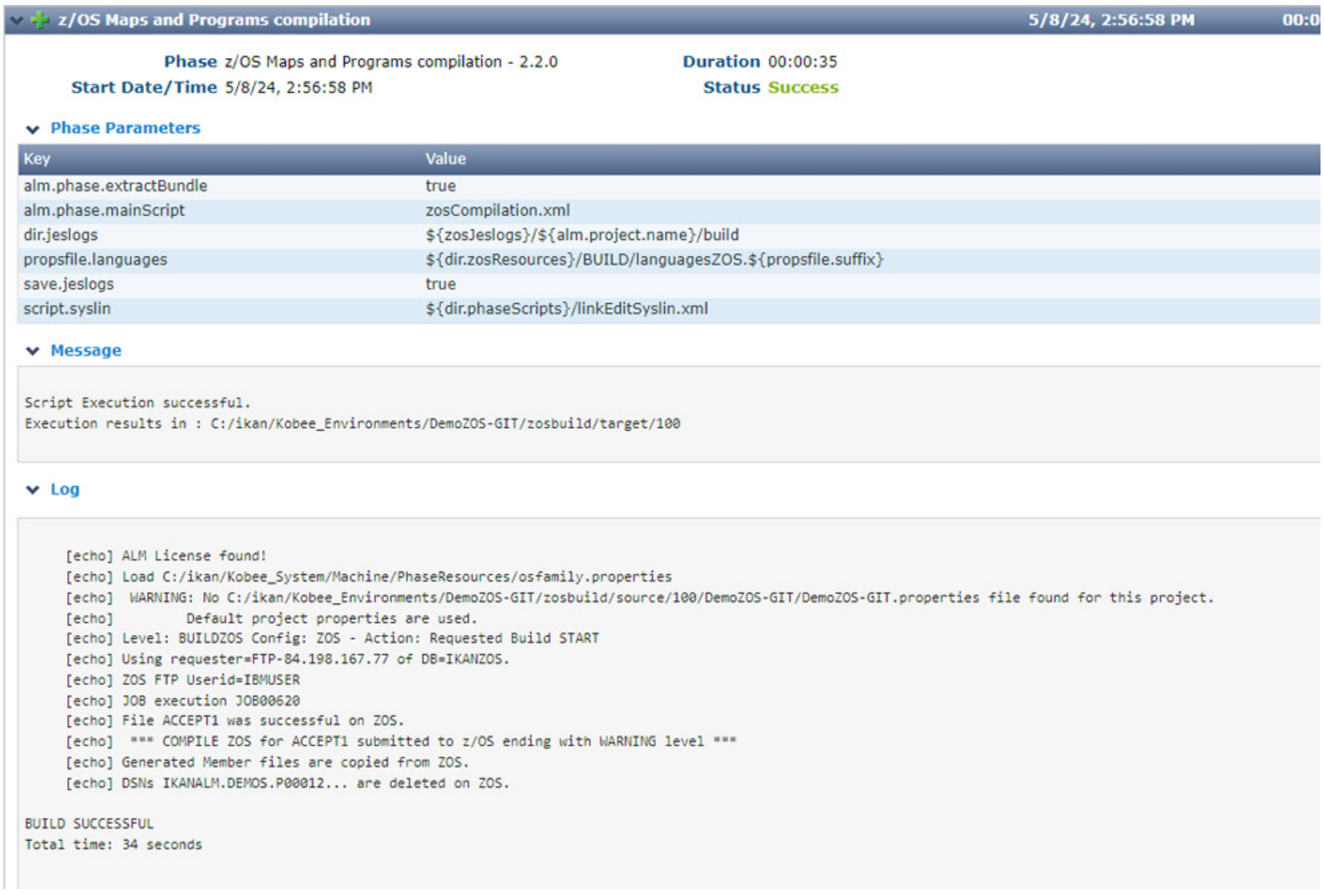
		OID	Environment	Machine
✚		11	ZOSBUILD	ikanalm

If you click on the *"Phase Logs"* tab you will see an ordered list of all the Phases that are used during the build process.

When you click the *"Build # on machine kobee"* bar, you can see the Phases that ran on the Build environment. Take your time to go through all the Phases and their logs.

<u>NOTE:</u> When we created the build level (and connected the build environment) for this "Test Drive", Kobee automatically set up all required general Phases. The only thing we had to do was to import the Phases that are specifically made to support the z/OS build process. The Kobee Phases architecture is very easy to use and requires no programming skills whatsoever.

Below is an example of the "z/OS Maps and Programs compilation" Phase log.


z/OS Maps and Programs compilation 5/8/24, 2:56:58 PM 00:0

Phase z/OS Maps and Programs compilation - 2.2.0 Duration 00:00:35
Start Date/Time 5/8/24, 2:56:58 PM Status Success

Phase Parameters

Key	Value
alm.phase.extractBundle	true
alm.phase.mainScript	zosCompilation.xml
dir.jeslogs	${zosJeslogs}/${alm.project.name}/build
propsfile.languages	${dir.zosResources}/BUILD/languagesZOS.${propsfile.suffix}
save.jeslogs	true
script.syslin	${dir.phaseScripts}/linkEditSyslin.xml

Message

```
Script Execution successful.
Execution results in : C:/ikan/Kobee_Environments/DemoZOS-GIT/zosbuild/target/100
```

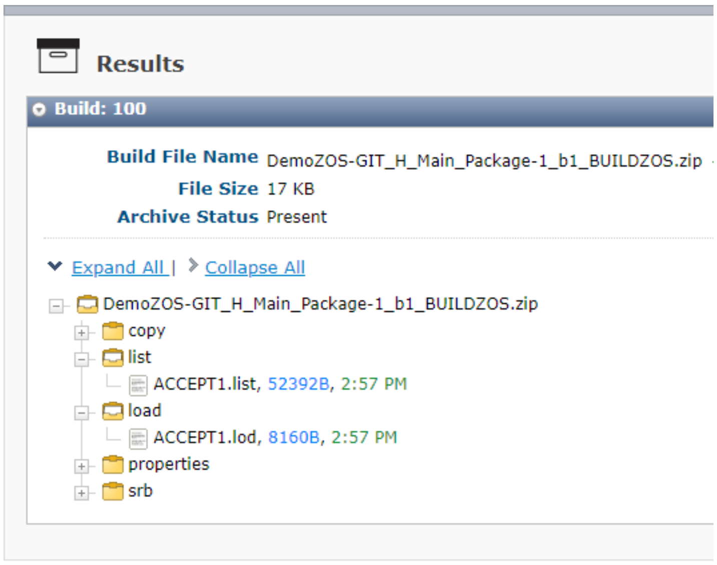
Log

```
     [echo] ALM License found!
     [echo] Load C:/ikan/Kobee_System/Machine/PhaseResources/osfamily.properties
     [echo]  WARNING: No C:/ikan/Kobee_Environments/DemoZOS-GIT/zosbuild/source/100/DemoZOS-GIT/DemoZOS-GIT.properties file found for this project.
     [echo]         Default project properties are used.
     [echo] Level: BUILDZOS Config: ZOS - Action: Requested Build START
     [echo] Using requester=FTP-84.198.167.77 of DB=IKANZOS.
     [echo] ZOS FTP Userid=IBMUSER
     [echo] JOB execution JOB00620
     [echo] File ACCEPT1 was successful on ZOS.
     [echo]  *** COMPILE ZOS for ACCEPT1 submitted to z/OS ending with WARNING level ***
     [echo] Generated Member files are copied from ZOS.
     [echo] DSNs IKANALM.DEMOS.P00012... are deleted on ZOS.

BUILD SUCCESSFUL
Total time: 34 seconds
```

Under the “Results” tab you can see the actual compiled artifact.

We have created:

- a loadmodule: “ACCEPT1.lod”
- a listing file: “ACCEPT1.list”


Results

Build: 100

Build File Name DemoZOS-GIT_H_Main_Package-1_b1_BUILDZOS.zip
File Size 17 KB
Archive Status Present

```
Expand All | Collapse All
DemoZOS-GIT_H_Main_Package-1_b1_BUILDZOS.zip
  copy
  list
    ACCEPT1.list, 52392B, 2:57 PM
  load
    ACCEPT1.lod, 8160B, 2:57 PM
  properties
  srb
```

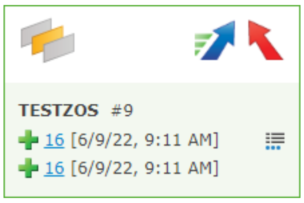
NOTE: Because our “Test Drive” setup is minimal, other tabs such as “Approvals”, “Issues” and “Dependencies” are empty.

You can read more about these features in the Kobee User Guide.

NOTE: Every build results is stored in the build archive (C:\ikan\Kobee_System\buildArchive\DemoZOS-GIT\Main).
On the Build Environment location (C:\ikan\Kobee_Environments\DemoZOS-GIT\build), we left the sources and targets from previous build Level Request as proof for you to see. Normally they are deleted automatically.

Deploying (promote) the build to the Test and Production environment

In the previous step we have built our source code. Now, we want to deploy that build result to our Test environment and later on to our Production environment. Hence, we need a Test level and a Production level in our project lifecycle.



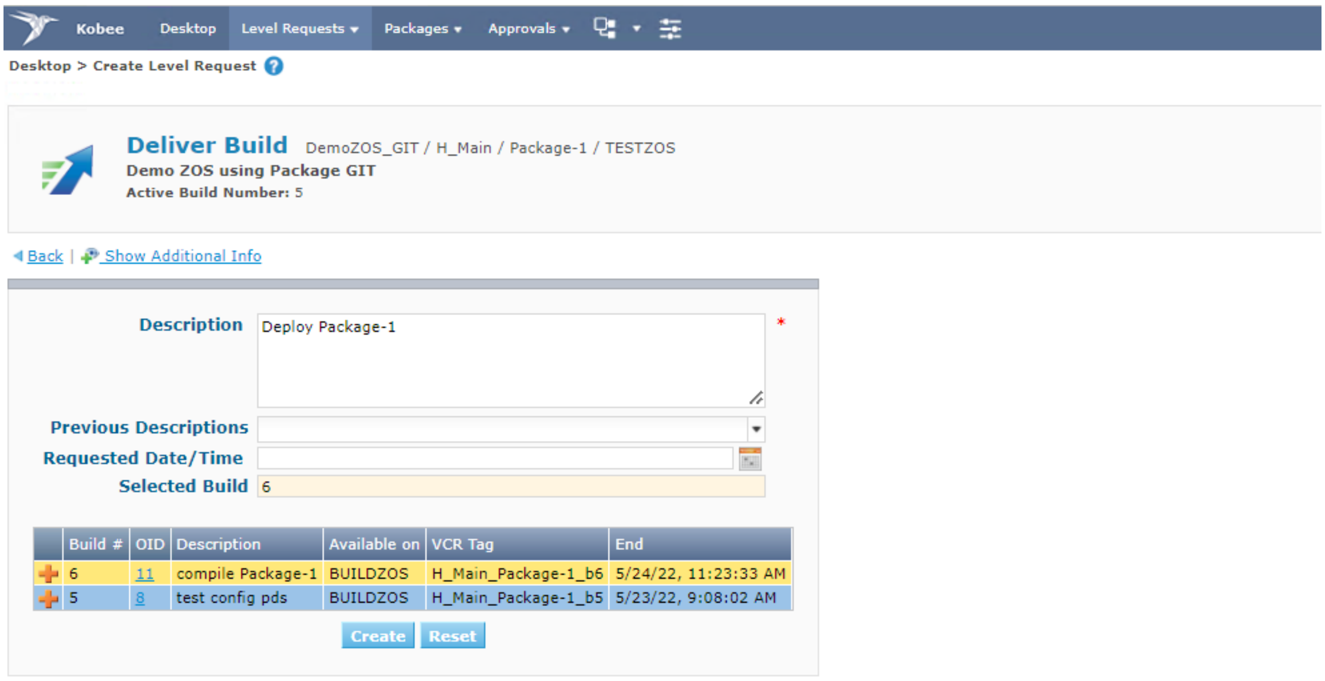
TESTZOS #9
16 [6/9/22, 9:11 AM]
16 [6/9/22, 9:11 AM]

Go to the Kobee Desktop, there you will see a Test level "TESTZOS" tile and a Production level "PRODZOS" tile, as we have already set this up for the "Test Drive".

Click the *"Deliver"* icon to initiate the Test (or the Deploy) Level Request.

On the "Deliver Build" page, select your package "Package-1" first.

As you can see the "Deliver Build" screen is almost identical to the "Request Build" screen except that here we can select which build we want to deliver to the Test environment.



Desktop > Create Level Request

Deliver Build DemoZOS_GIT / H_Main / Package-1 / TESTZOS
Demo ZOS using Package GIT
Active Build Number: 5

Description	Deploy Package-1 *
Previous Descriptions	
Requested Date/Time	
Selected Build	6

	Build #	OID	Description	Available on	VCR Tag	End
	6	11	compile Package-1	BUILDZOS	H_Main_Package-1_b6	5/24/22, 11:23:33 AM
	5	8	test config pds	BUILDZOS	H_Main_Package-1_b5	5/23/22, 9:08:02 AM

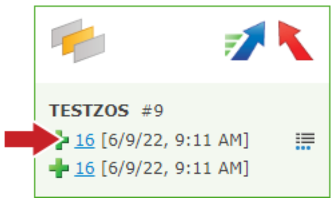
Provide a meaningful description, select the latest build by clicking on the table row and hit the *"Create"* button.

NOTE: We can only select the build that is available on the build level "BUILDZOS", that's because the Test level is the next level after the build level in the project lifecycle. In case we would have selected the deliver to production (in the "PRODZOS" tile) instead, only the build that is on the Test level would be available.

Again we head back to the Kobee Desktop and wait for the level request to finish.

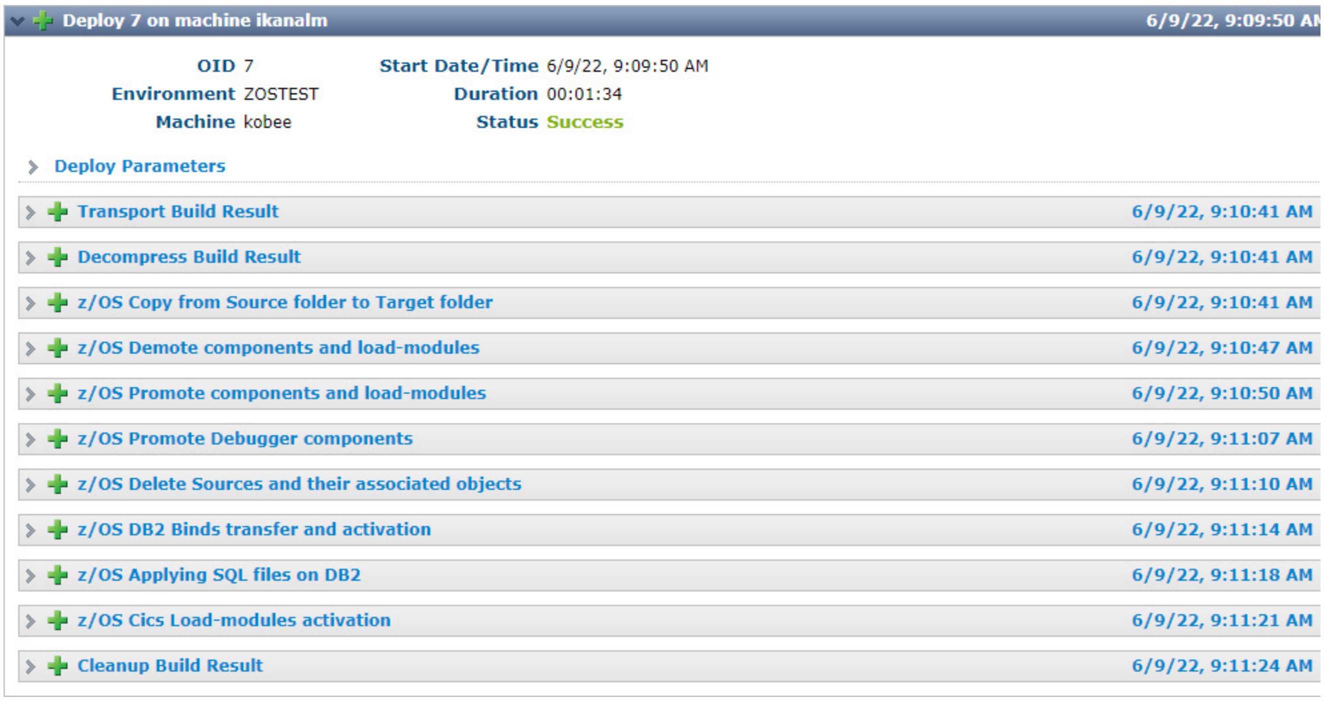
Verifying the (promote) Deploy Level Request

Click on the first “Level Request OID” link in the "TESTZOS" tile. This will take you to the “Level Request Detail” page.


TESTZOS #9
16 [6/9/22, 9:11 AM]
16 [6/9/22, 9:11 AM]

If you click on the *“Phase Logs”* tab you will see an ordered list of all the Phases that are used during the build process.

NOTE: When we created the deploy level and environment for this “Test Drive”, Kobee automatically set up all required Core Phases on the level and environment. The only thing we had to do was to import the Solution Phases that are specifically made to support the z/OS promote process.


Deploy 7 on machine ikanalm 6/9/22, 9:09:50 AM

OID	7	Start Date/Time	6/9/22, 9:09:50 AM
Environment	ZOSTEST	Duration	00:01:34
Machine	kobee	Status	Success

Deploy Parameters

Transport Build Result	6/9/22, 9:10:41 AM
Decompress Build Result	6/9/22, 9:10:41 AM
z/OS Copy from Source folder to Target folder	6/9/22, 9:10:41 AM
z/OS Demote components and load-modules	6/9/22, 9:10:47 AM
z/OS Promote components and load-modules	6/9/22, 9:10:50 AM
z/OS Promote Debugger components	6/9/22, 9:11:07 AM
z/OS Delete Sources and their associated objects	6/9/22, 9:11:10 AM
z/OS DB2 Binds transfer and activation	6/9/22, 9:11:14 AM
z/OS Applying SQL files on DB2	6/9/22, 9:11:18 AM
z/OS Cics Load-modules activation	6/9/22, 9:11:21 AM
Cleanup Build Result	6/9/22, 9:11:24 AM

NOTE: On the Deploy Environment location (C:\ikan\Kobee_Environments\DemoZOS-GIT\zostest), you can find what has been deployed during previous deploy Level Requests in the "targets" folder. The "sources" folder contents are deleted automatically, except when you set the environment in debug mode (by clicking the "edit" button next to the environment in Kobee).

This is the end of our very short introduction focused on a typical developer. We only scratched the surface of the possibilities of using DevOps on the mainframe with Kobee.

In the next part we will show you how the global and project setup was done in Kobee, for those who are interested in the administrative part.

Part II, The Kobee setup for Administrators

Global Administration: Initial Overview

Let's start with verifying what is already set up in the Kobee Global Administration after a clean installation. We will describe it shortly, if however, you want to know more about a specific topic, have a look at the respective chapters in the Global Administration part of the Kobee User Guide.

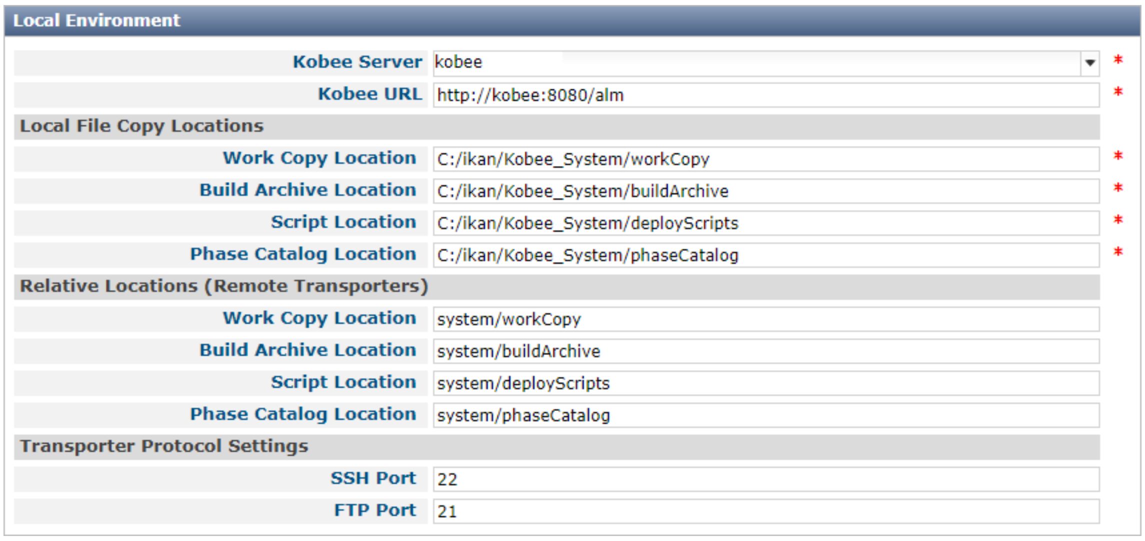
Open Kobee and log in with the following credentials:

Username: global
Password: global

Click the "Global Administration" icon in the menu. In the overview panel click *"System Settings"*.

Here you will see the Build Archive Location on the Kobee Server, where all the Build Artifacts (e.g., load modules, deployable archives, …) will be stored after a successful build, so that they can be deployed later in the lifecycle. It is a local path on the server, something like "C:/ikan/Kobee_System/buildArchive", or "/opt/kobee/system/buildArchive".

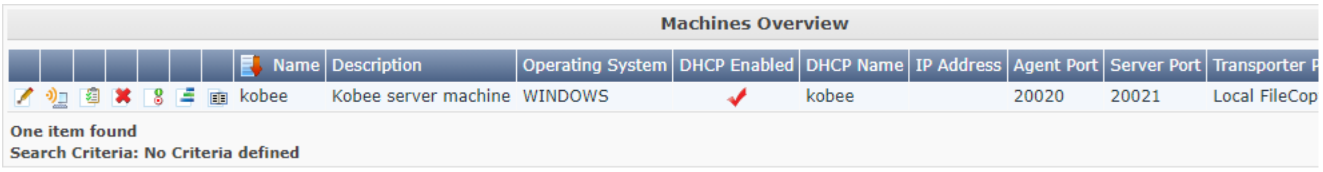
For the Kobee Core Phases, the Work Copy, Script and Phase Catalog Locations are defined.



Local Environment

Kobee Server	kobee	*
Kobee URL	http://kobee:8080/alm	*
Local File Copy Locations		
Work Copy Location	C:/ikan/Kobee_System/workCopy	*
Build Archive Location	C:/ikan/Kobee_System/buildArchive	*
Script Location	C:/ikan/Kobee_System/deployScripts	*
Phase Catalog Location	C:/ikan/Kobee_System/phaseCatalog	*
Relative Locations (Remote Transporters)		
Work Copy Location	system/workCopy	
Build Archive Location	system/buildArchive	
Script Location	system/deployScripts	
Phase Catalog Location	system/phaseCatalog	
Transporter Protocol Settings		
SSH Port	22	
FTP Port	21	

Under *"Machines > Overview"* (in the submenu), you will find the definition of the Kobee machine.



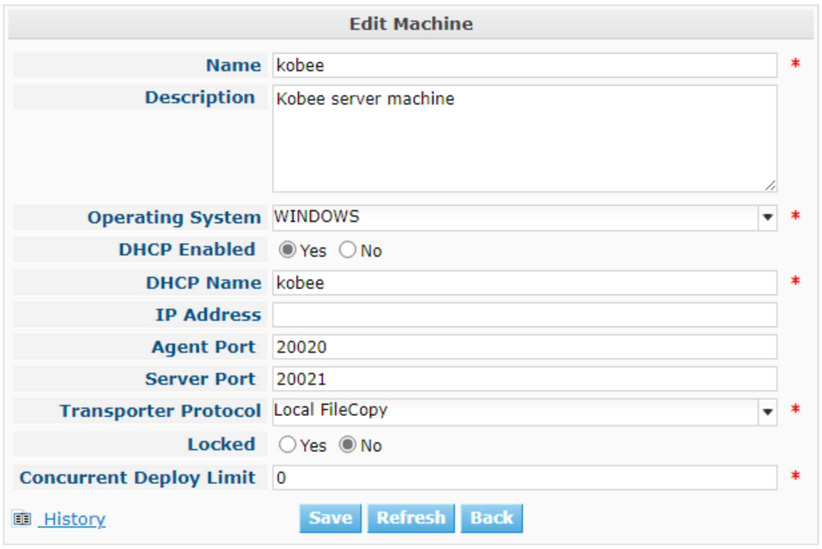
Machines Overview

Name	Description	Operating System	DHCP Enabled	DHCP Name	IP Address	Agent Port	Server Port	Transporter P
kobee	Kobee server machine	WINDOWS	✓	kobee		20020	20021	Local FileCop

One item found
Search Criteria: No Criteria defined

If you click the "Edit" icon, you will see the details of the machine and the connected environments.

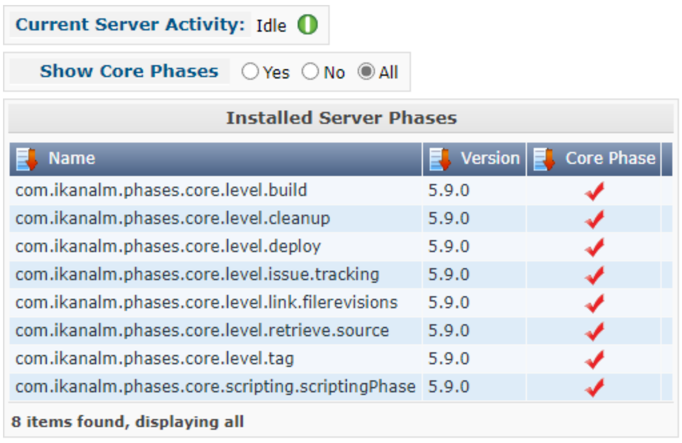
There is also an Agent installed on this Machine, and both Agent and Server processes are running as a service. The Agents handle the Build and Deploy actions (bundled as Phases) on a specific Build, Test or Production environment.



Edit Machine

Name	kobee *
Description	Kobee server machine
Operating System	WINDOWS *
DHCP Enabled	[✓ Yes] No
DHCP Name	kobee *
IP Address	
Agent Port	20020
Server Port	20021
Transporter Protocol	Local FileCopy *
Locked	Yes [✓ No]
Concurrent Deploy Limit	0 *

If you go back and click the "Installed Phases" icon, you can see the Current Server Activity and the Current Agent Activity, which should both be active (green icon).



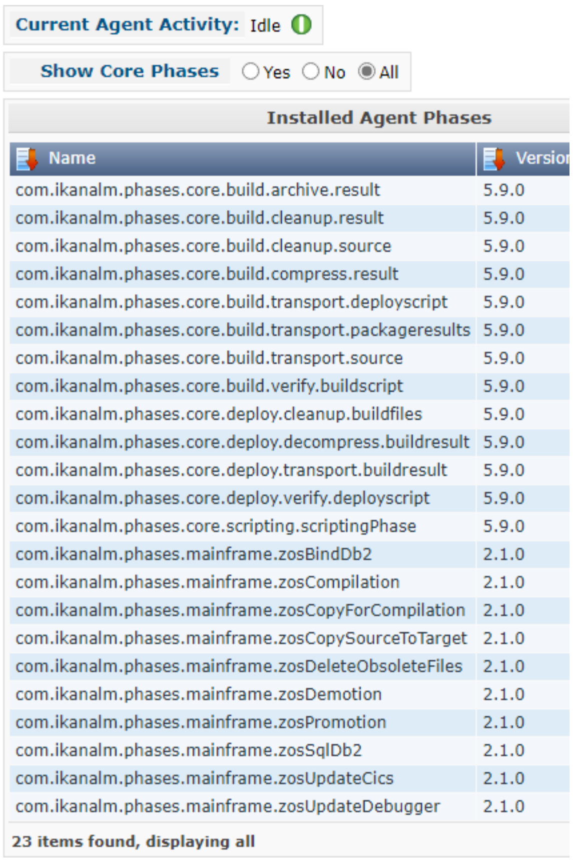
Current Server Activity: Idle

Show Core Phases Yes No [✓ All]

Installed Server Phases

Name	Version	Core Phase
com.ikanalm.phases.core.level.build	5.9.0	✓
com.ikanalm.phases.core.level.cleanup	5.9.0	✓
com.ikanalm.phases.core.level.deploy	5.9.0	✓
com.ikanalm.phases.core.level.issue.tracking	5.9.0	✓
com.ikanalm.phases.core.level.link.filerevisions	5.9.0	✓
com.ikanalm.phases.core.level.retrieve.source	5.9.0	✓
com.ikanalm.phases.core.level.tag	5.9.0	✓
com.ikanalm.phases.core.scripting.scriptingPhase	5.9.0	✓

8 items found, displaying all



Current Agent Activity: Idle

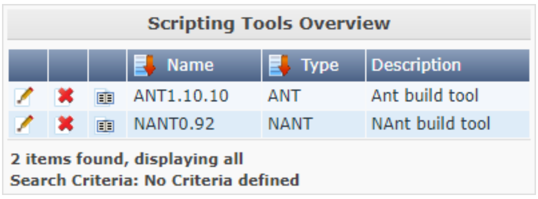
Show Core Phases Yes No [✓ All]

Installed Agent Phases

Name	Versio
com.ikanalm.phases.core.build.archive.result	5.9.0
com.ikanalm.phases.core.build.cleanup.result	5.9.0
com.ikanalm.phases.core.build.cleanup.source	5.9.0
com.ikanalm.phases.core.build.compress.result	5.9.0
com.ikanalm.phases.core.build.transport.deployscript	5.9.0
com.ikanalm.phases.core.build.transport.packageresults	5.9.0
com.ikanalm.phases.core.build.transport.source	5.9.0
com.ikanalm.phases.core.build.verify.buildscript	5.9.0
com.ikanalm.phases.core.deploy.cleanup.buildfiles	5.9.0
com.ikanalm.phases.core.deploy.decompress.buildresult	5.9.0
com.ikanalm.phases.core.deploy.transport.buildresult	5.9.0
com.ikanalm.phases.core.deploy.verify.deployscript	5.9.0
com.ikanalm.phases.core.scripting.scriptingPhase	5.9.0
com.ikanalm.phases.mainframe.zosBindDb2	2.1.0
com.ikanalm.phases.mainframe.zosCompilation	2.1.0
com.ikanalm.phases.mainframe.zosCopyForCompilation	2.1.0
com.ikanalm.phases.mainframe.zosCopySourceToTarget	2.1.0
com.ikanalm.phases.mainframe.zosDeleteObsoleteFiles	2.1.0
com.ikanalm.phases.mainframe.zosDemotion	2.1.0
com.ikanalm.phases.mainframe.zosPromotion	2.1.0
com.ikanalm.phases.mainframe.zosSqlDb2	2.1.0
com.ikanalm.phases.mainframe.zosUpdateCics	2.1.0
com.ikanalm.phases.mainframe.zosUpdateDebugger	2.1.0

23 items found, displaying all

Under *"Scripting Tools > Overview"* (in the submenu), you can see that the "ANT 1.10.10" scripting tool is defined. This tool is used by Kobee to execute Build and Deploy scripts. Click the "Edit" icon, for more details.



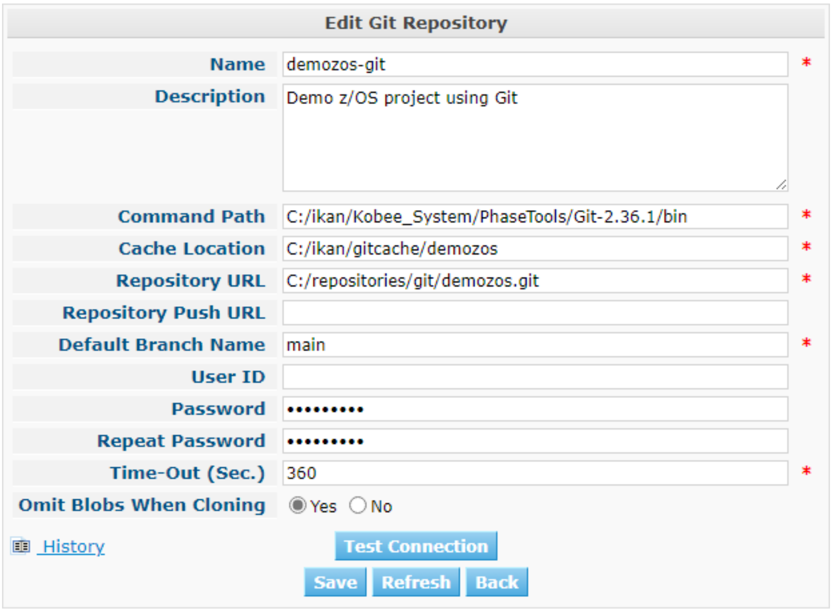
Scripting Tools Overview

			Name	Type	Description
			ANT1.10.10	ANT	Ant build tool
			NANT0.92	NANT	NAnt build tool

2 items found, displaying all
Search Criteria: No Criteria defined

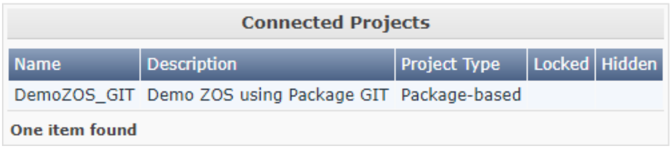
The Git repository is another key component in our Demo project setup. Kobee uses this repository to monitor and retrieve the sample code files.

Under *"Version Control Repositories > Overview"*, you can click the "Edit" icon on the "demozos" entry for more details.



Edit Git Repository

Name	demozos-git *
Description	Demo z/OS project using Git
Command Path	C:/ikan/Kobee_System/PhaseTools/Git-2.36.1/bin *
Cache Location	C:/ikan/gitcache/demozos *
Repository URL	C:/repositories/git/demozos.git *
Repository Push URL	
Default Branch Name	main *
User ID	
Password	•••••••••
Repeat Password	•••••••••
Time-Out (Sec.)	360 *
Omit Blobs When Cloning	[✓ Yes] No



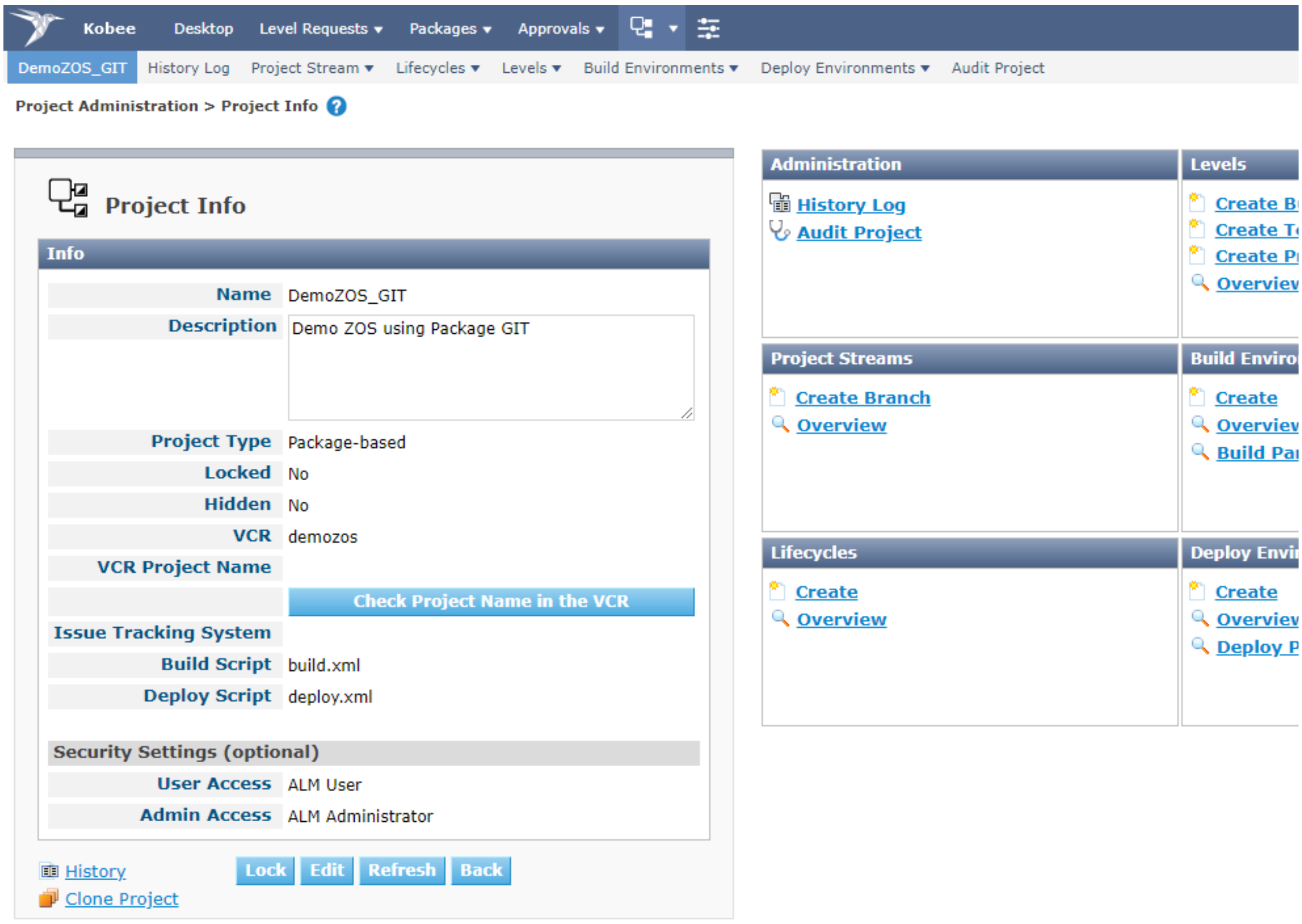
Connected Projects

Name	Description	Project Type	Locked	Hidden
DemoZOS_GIT	Demo ZOS using Package GIT	Package-based		

One item found

Looking at the z/OS Project

In the Project Administration context, select *"Project > Project Administration"* and select the "DemoZOS_GIT" project we created.



Project Info

Info

Name	DemoZOS_GIT
Description	Demo ZOS using Package GIT
Project Type	Package-based
Locked	No
Hidden	No
VCR	demozos
VCR Project Name	
	Check Project Name in the VCR
Issue Tracking System	
Build Script	build.xml
Deploy Script	deploy.xml

Security Settings (optional)

User Access	ALM User
Admin Access	ALM Administrator

The Project Type is Package-based. A Package allows moving one or more individual files selected manually from a VCR stream, this is a common way of working on the mainframe.

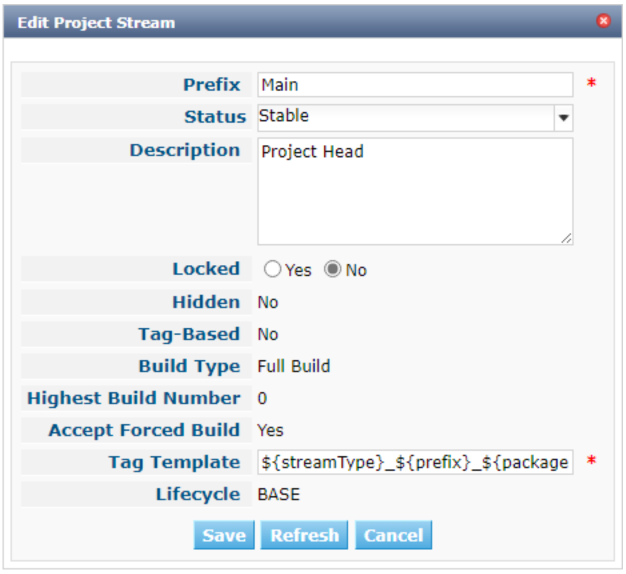
You can see that the Git repository "demozos" is connected to this project.

As mentioned in the first part: together with the Project, a Head Project Stream is created that points to the master branch of the project in Git.

If you go to *"Project Streams > Overview"*, then click the "Edit" icon and then click the "Edit" button on the "Project Stream Info" panel.

Here you can see all the options defined such as "Prefix", "Build Type", "Accept Forced Build", etc…

All of this is explained in the User Guide.


Edit Project Stream

Prefix	Main *
Status	Stable
Description	Project Head
Locked	Yes [✓ No]
Hidden	No
Tag-Based	No
Build Type	Full Build
Highest Build Number	0
Accept Forced Build	Yes
Tag Template	${streamType}_${prefix}_${package *
Lifecycle	BASE

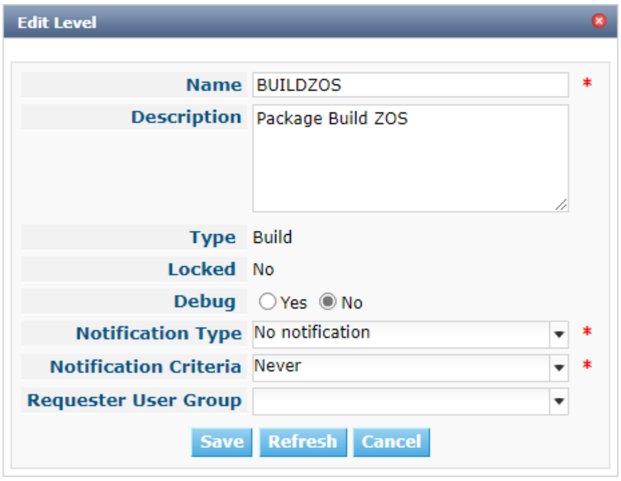
The Build Level

A Build Level is the first level in the lifecycle and is responsible for building your code.

Under *"Lifecycles > Overview"* you will notice the BASE Lifecycle which is linked to the main Project Stream.

Click the "Edit" icon next to this Lifecycle, you will see all the levels that are connected to this BASE lifecycle.

Click the "Edit" icon on the "BUILDZOS" level, this is the Build Level. On the "Level Info" panel click the *"Edit"* button.


Edit Level

Name	BUILDZOS *
Description	Package Build ZOS
Type	Build
Locked	No
Debug	Yes [✓ No]
Notification Type	No notification *
Notification Criteria	Never *
Requester User Group	

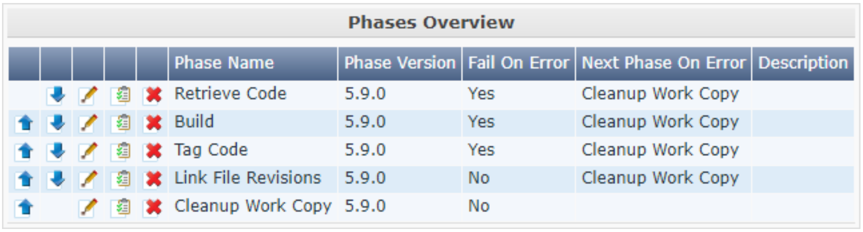
Most fields speak for themselves (let's ignore the Notification, Schedule and Requester fields for now).

Activating the Debug option makes it easier to track things in the beginning, especially when a Build fails. Once everything runs smoothly, you can disable it.

Click the *"Cancel"* button.

When we create a level, the Phases linked to that Level are automatically created as well. Those Phases will be executed when a Level Request is initiated.

You can see the Phases by selecting the “Edit Phases” link underneath the “Phases Overview” panel.

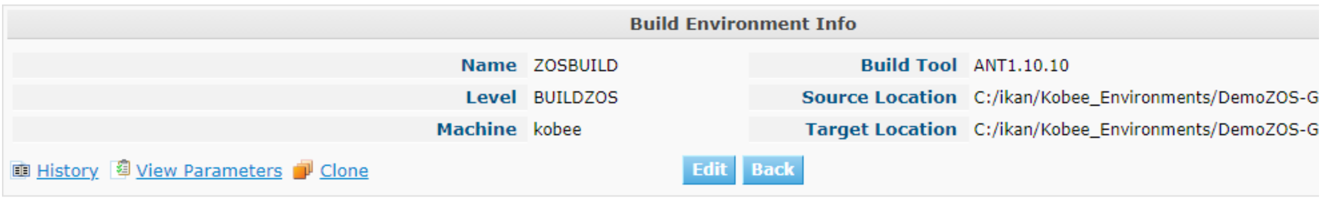

Phases Overview

Phase Name	Phase Version	Fail On Error	Next Phase On Error	Description
Retrieve Code	5.9.0	Yes	Cleanup Work Copy	
Build	5.9.0	Yes	Cleanup Work Copy	
Tag Code	5.9.0	Yes	Cleanup Work Copy	
Link File Revisions	5.9.0	No	Cleanup Work Copy	
Cleanup Work Copy	5.9.0	No		

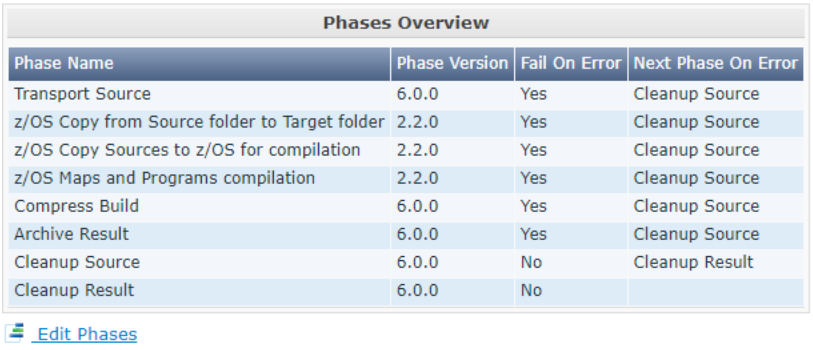
The Build Environment

A (Build) Level is a conceptual step in the Lifecycle. We still need a physical machine to execute our Build on, so we have to link a Build Environment (the machine we will build on) to the Build Level.

Click *"Build Environments > Overview"*, the click the "Edit" icon for the "ZOSBUILD" environment.


Build Environment Info

Name	ZOSBUILD	**Build Tool**	ANT1.10.10
Level	BUILDZOS	**Source Location**	C:/ikan/Kobee_Environments/DemoZOS-G
Machine	kobee	**Target Location**	C:/ikan/Kobee_Environments/DemoZOS-G


Phases Overview

Phase Name	Phase Version	Fail On Error	Next Phase On Error
Transport Source	6.0.0	Yes	Cleanup Source
z/OS Copy from Source folder to Target folder	2.2.0	Yes	Cleanup Source
z/OS Copy Sources to z/OS for compilation	2.2.0	Yes	Cleanup Source
z/OS Maps and Programs compilation	2.2.0	Yes	Cleanup Source
Compress Build	6.0.0	Yes	Cleanup Source
Archive Result	6.0.0	Yes	Cleanup Source
Cleanup Source	6.0.0	No	Cleanup Result
Cleanup Result	6.0.0	No	

Just as for the Level, the Phases linked to the Environment are created together with the Build Environment. They will be executed when the Build of a Level Request will be executed on the Kobee Agent.

Kobee always starts by transferring the sources to the “Source location” and placing the result in the “Target Location”. These locations are automatically cleaned up when the Level Request has finished, unless we have chosen to use the debug function.

NOTE: The source and target locations can be chosen freely. In our example it is “C:\ikan\Kobee_Environments\DemoZOS-GIT\zosbuild”.

NOTE: In order to distinguish Levels from Environments, we use uppercase for the level and lowercase for the environment directories. Levels and Environments can have the same name.

If you click the “Edit” button in the “Build Environment Info” panel, you can see we have set “Downloadable Build” option to “Yes”, so we are able to download the build result.

The Build Environment and Phases parameters

Phases can -or sometimes must- be provided with additional information in the form of parameters. For example: a Phase may need a location of a specific resource. This enables a high level of customization without the need to alter the Phase's inner mechanics. The Phase parameters make it possible to customize the build and deploy process with minimum effort and without the need of programming skills.

Phase parameters can be set on various entities: Machines, Environments and Phases, following a cascading order.

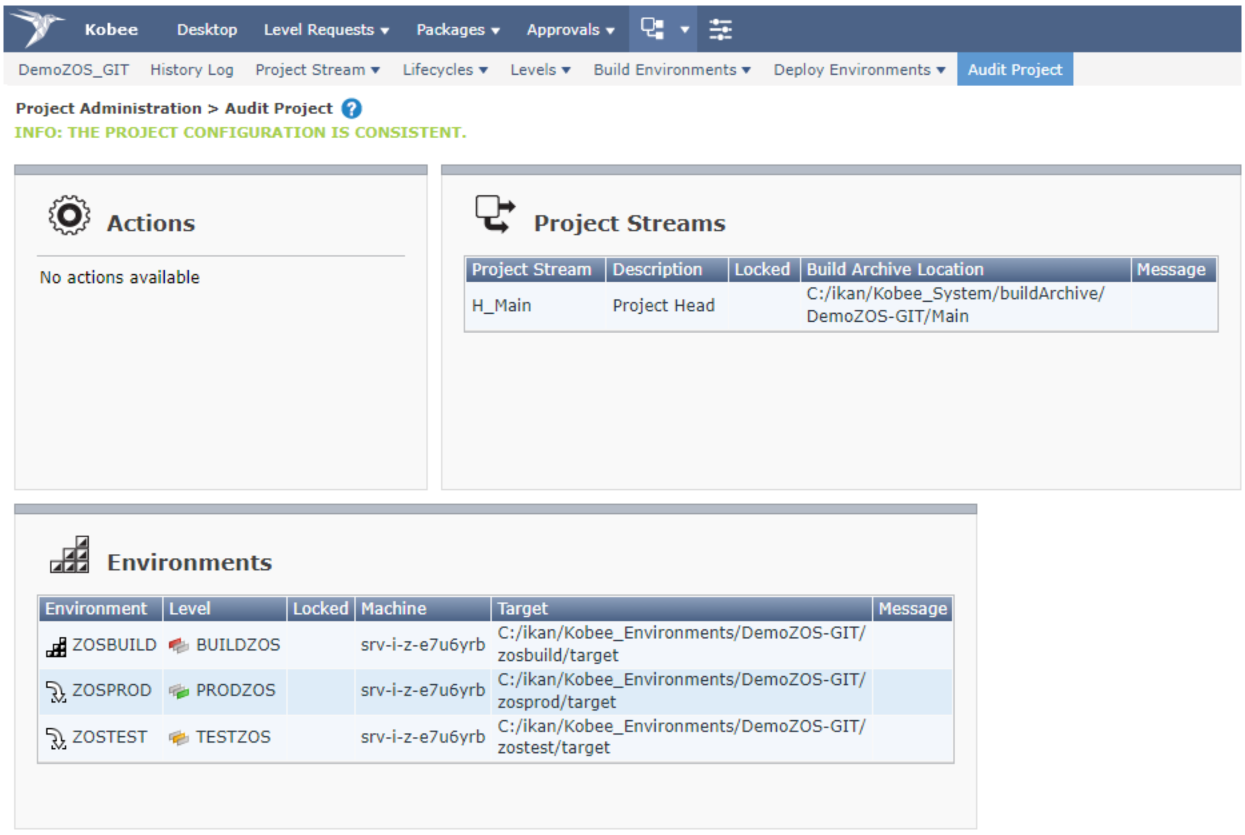
For the z/OS solution we are working with Phase Models, Resources and Scripts that are tailored to the client's main-frame environment and integrated into Kobee by the z/OS Phases and Phase parameters.

Auditing the Project

When creating or making changes to a level, Kobee automatically blocks the level and requires the user to run a project audit prior to using it. This project audit is a verification process performed by Kobee which checks the project setup consistency.

On the overview, you will see most of the different objects we created.

The information screen for our Project displays the Build Archive of the Head Project Stream (where our future Builds will be stored) and the Build Level containing one Build Environment on the Kobee Agent, where the build will be executed.



Project Administration > Audit Project

INFO: THE PROJECT CONFIGURATION IS CONSISTENT.

Actions

No actions available

Project Streams

Project Stream	Description	Locked	Build Archive Location	Message
H_Main	Project Head		C:/ikan/Kobee_System/buildArchive/DemoZOS-GIT/Main	

Environments

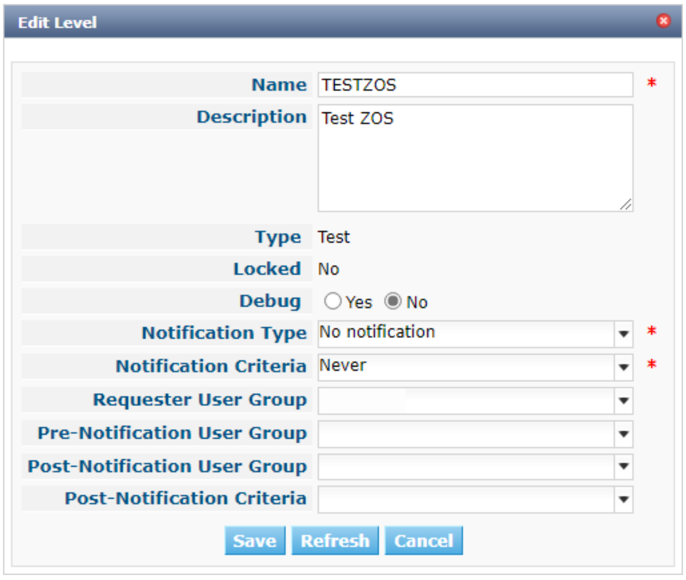
Environment	Level	Locked	Machine	Target	Message
ZOSBUILD	BUILDZOS		srv-i-z-e7u6yrb	C:/ikan/Kobee_Environments/DemoZOS-GIT/zosbuild/target	
ZOSPROD	PRODZOS		srv-i-z-e7u6yrb	C:/ikan/Kobee_Environments/DemoZOS-GIT/zosprod/target	
ZOSTEST	TESTZOS		srv-i-z-e7u6yrb	C:/ikan/Kobee_Environments/DemoZOS-GIT/zostest/target	

The Test (and Production) Level

A Test Level is the next level (in our lifecycle) after the Build Level and is responsible for delivering the build to the Test department. Likewise a Production Level is the next level after the Test Level. Since Test Levels and Production Levels are similar, apart from the notification options, this topic applies to both.

In the Project Administration section, edit the Project.

Go to *"Levels > Overview"*, click the "Edit" icon for the "TESTZOS" level and then click the "Edit" button on the "Level Info" panel.

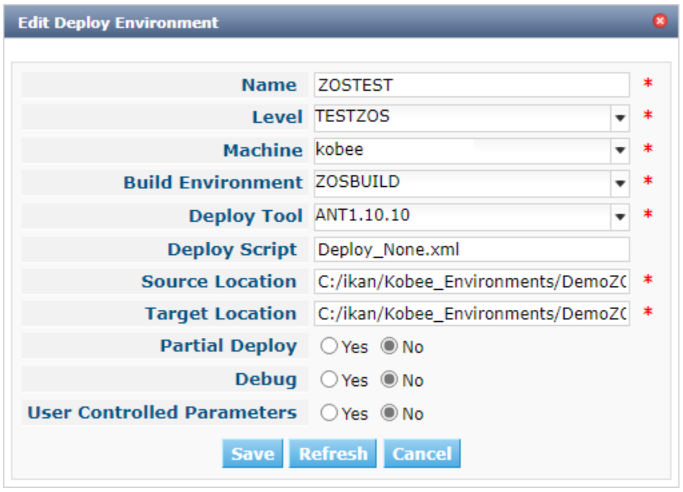


Name	TESTZOS *
Description	Test ZOS
Type	Test
Locked	No
Debug	Yes [✓ No]
Notification Type	No notification *
Notification Criteria	Never *
Requester User Group	
Pre-Notification User Group	
Post-Notification User Group	
Post-Notification Criteria	

The Deploy Environment

Similar to Build Level, the Test Level (or Production Level) is just a conceptual step in the lifecycle. We need a physical Machine to which we can deploy our Build result, so we need to link a Deploy Environment to the Level.

Go to *"Deploy Environments > Overview"*, click the "Edit" icon for the "ZOSTEST" environment and then click the "Edit" button on the "Deploy Environment Info" panel.



Name	ZOSTEST *
Level	TESTZOS *
Machine	kobee *
Build Environment	ZOSBUILD *
Deploy Tool	ANT1.10.10 *
Deploy Script	Deploy_None.xml
Source Location	C:/ikan/Kobee_Environments/DemoZ(*
Target Location	C:/ikan/Kobee_Environments/DemoZ(*
Partial Deploy	Yes [✓ No]
Debug	Yes [✓ No]
User Controlled Parameters	Yes [✓ No]

This is almost similar to a Build Environment.

The deploy will be executed by the Kobee Agent on the selected Machine. We have “ZOSBUILD” selected as “Build Environment” to indicate that we want to deploy the result of our Build Environment to our Deploy Environment.

The Build result previously created will be extracted in the “Source Location”.

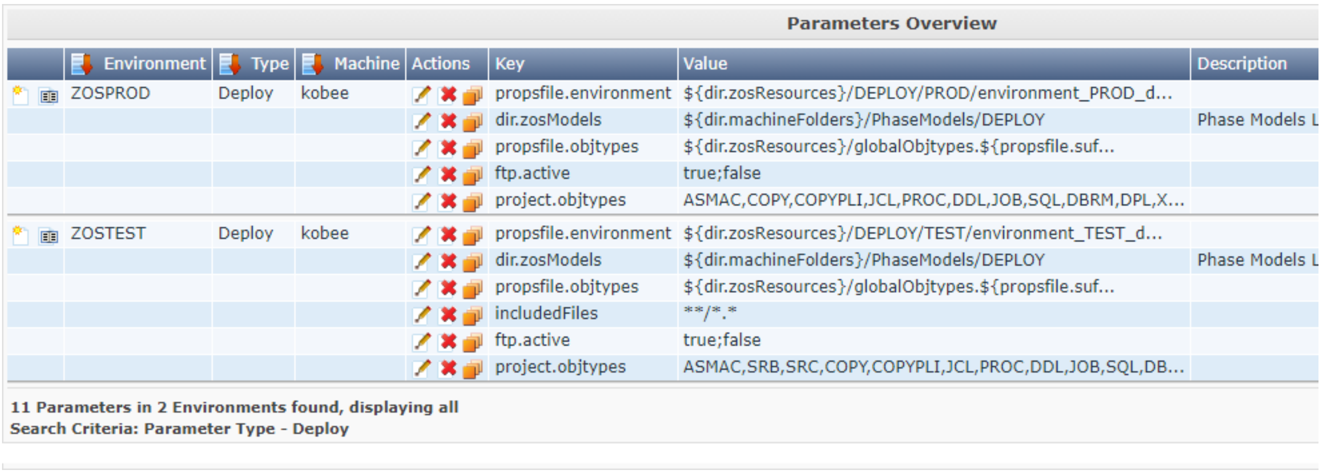
You can view the Phases that will be executed during the deployment (Level Request) to this Deploy Environment in the “Phases Overview” panel.

Creating the Deploy Parameters

What we did earlier for the Build parameters, should also be done for the deploy parameters on the Deploy Environment and on the deployment Phases.

We have set these already according to our z/OS Demo project setup.

You can see them by going to *“Deploy Environments > Deploy Parameters”* for the environment and for the machine by going to *“Global Administration > Machines > Machine Parameters”*.



Parameters Overview

Environment	Type	Machine	Key	Value	Description
ZOSPROD	Deploy	kobee	propsfile.environment	${dir.zosResources}/DEPLOY/PROD/environment_PROD_d...	
			dir.zosModels	${dir.machineFolders}/PhaseModels/DEPLOY	Phase Models L
			propsfile.objtypes	${dir.zosResources}/globalObjtypes.${propsfile.suf...	
			ftp.active	true;false	
			project.objtypes	ASMAC,COPY,COPYPLI,JCL,PROC,DDL,JOB,SQL,DBRM,DPL,X...	
ZOSTEST	Deploy	kobee	propsfile.environment	${dir.zosResources}/DEPLOY/TEST/environment_TEST_d...	
			dir.zosModels	${dir.machineFolders}/PhaseModels/DEPLOY	Phase Models L
			propsfile.objtypes	${dir.zosResources}/globalObjtypes.${propsfile.suf...	
			includedFiles	**/*.*	
			ftp.active	true;false	
			project.objtypes	ASMAC,SRB,SRC,COPY,COPYPLI,JCL,PROC,DDL,JOB,SQL,DB...	

11 Parameters in 2 Environments found, displaying all
Search Criteria: Parameter Type - Deploy

Auditing the Project

Just as for the Build Level, we needed to audit the project first to unlock the Test and Prod Levels.

The z/OS solution Phases

Phases represent specific tasks or actions that must be performed on the Levels and Environments.

Kobee comes with a set of “Core” Phases, “Solution Phases” such as the z/OS Phases, but you can also create your own “Custom Phases” which gives you endless possibilities.

The main advantage of using Phases is that they allow you to customize your project's workflow with reusable building blocks. On top of that, they can be shared and distributed onto local and remote machines.

We will shortly cover the three major components: the Phases, the Resource files and the Model files that are used by the z/OS solution on Kobee.

Phases

The z/OS Phases are used to run different z/OS tasks. Most of these Phases will generate JCL that will be submitted for execution on the z/OS mainframe machine.

Models

For the previously mentioned JCL generation we use predefined JCL Models. See the sample below for a sample JCL Model for a CICS pre-compile compilation.

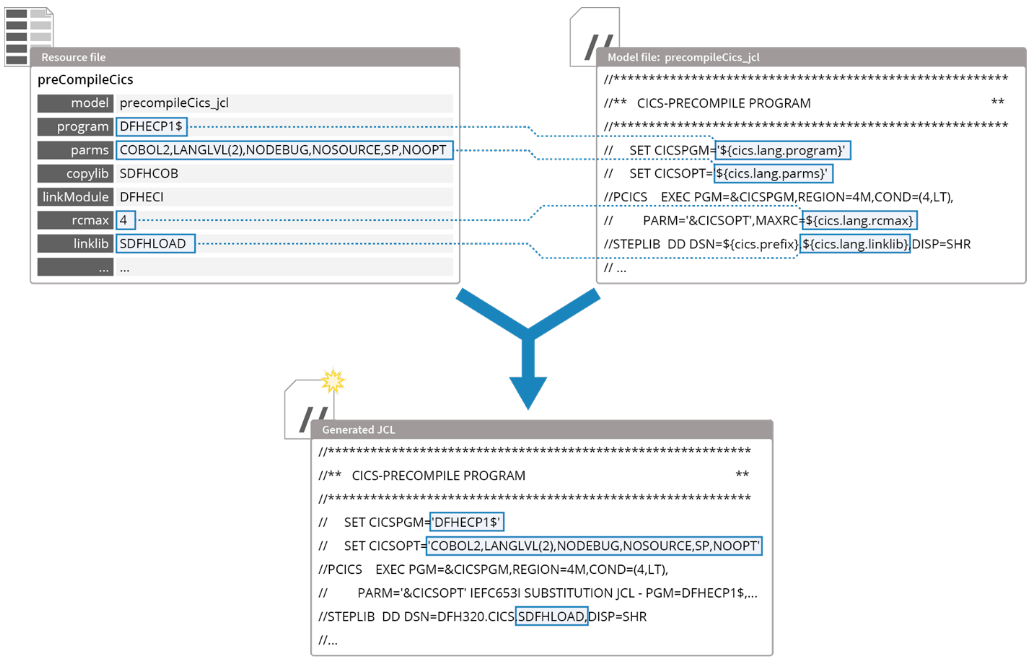
```
//*******************************************************************
//**   CICS-PRECOMPILE PROGRAM                                     **
//*******************************************************************
//      SET CICSPGM='${cics.lang.program}'
//      SET CICSOPT='${cics.lang.parms}'
//PCICS     EXEC PGM=&CICSPGM,REGION=4M,COND=(4,LT),
//          PARM='&CICSOPT',MAXRC=${cics.lang.rcmax}
//STEPLIB   DD DSN=${cics.prefix}.${cics.lang.linklib},DISP=SHR
//SYSPRINT DD SYSOUT=*
//*YSPRINT DD  DISP=(MOD,PASS),DSN=&&PCMPLIST,
//*             UNIT=SYSDA,SPACE=(CYL,(10,10)),
//*             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//SYSPUNCH DD DSN=&&SYSCIN,DISP=(,PASS),
//             UNIT=${env.zos.unit},DCB=BLKSIZE=400,
//             SPACE=(400,(400,400))
//SYSIN     DD DSN=&&&SRCOMPIL,DISP=(OLD,PASS)
//      SET SRCOMPIL=SYSCIN
```

Resource files

The JCL Model files have parameters defined (e.g. ${cics.lang.program}) that are substituted by the values from the Resource files. Below is a sample of a some properties in a CICS Build resource file.

```
# ------------------------------------------------------
#    Properties for CICS
# ------------------------------------------------------
cics.name=CICSTEST
cics.TOR.name=CICTTEST
cics.FOR.name=CICFTEST
cics.prefix=DFH320.CICS
```

When we combine these three components we get the following:



preCompileCics

model	precompileCics_jcl
program	DFHECP1$
parms	COBOL2,LANGLVL(2),NODEBUG,NOSOURCE,SP,NOOPT
copylib	SDFHCOB
linkModule	DFHECI
rcmax	4
linklib	SDFHLOAD
...	...

```
//*********************************************************
//**   CICS-PRECOMPILE PROGRAM                            **
//*********************************************************
//    SET CICSPGM='${cics.lang.program}'
//    SET CICSOPT='${cics.lang.parms}'
//PCICS    EXEC PGM=&CICSPGM,REGION=4M,COND=(4,LT),
//        PARM='&CICSOPT',MAXRC=${cics.lang.rcmax}
//STEPLIB  DD DSN=${cics.prefix}.${cics.lang.linklib},DISP=SHR
// ...
```

```
//*********************************************************
//**   CICS-PRECOMPILE PROGRAM                            **
//*********************************************************
//    SET CICSPGM='DFHECP1$'
//    SET CICSOPT='COBOL2,LANGLVL(2),NODEBUG,NOSOURCE,SP,NOOPT'
//PCICS    EXEC PGM=&CICSPGM,REGION=4M,COND=(4,LT),
//        PARM='&CICSOPT' IEFC653I SUBSTITUTION JCL - PGM=DFHECP1$,...
//STEPLIB  DD DSN=DFH320.CICS.SDFHLOAD,DISP=SHR
//...
```

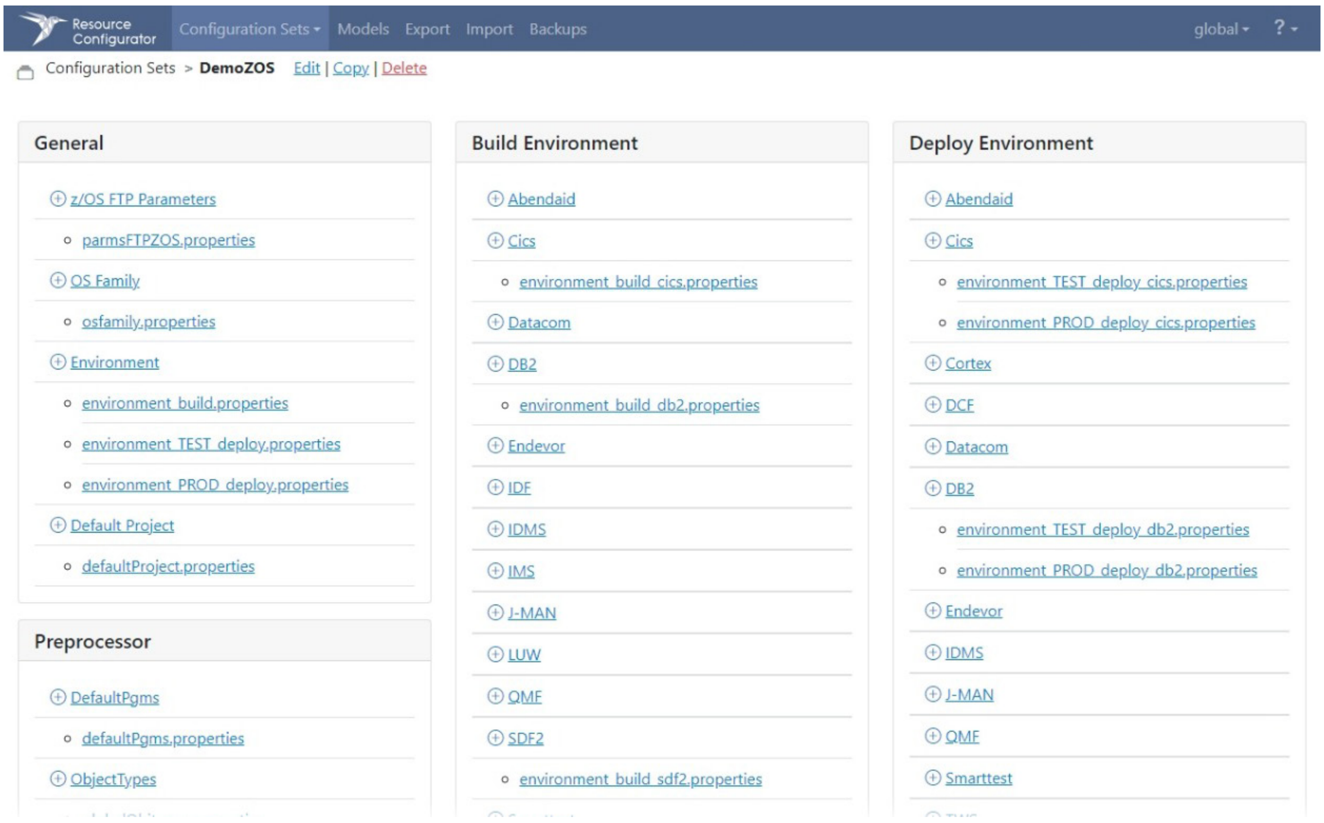
For this Demo project setup we have already defined everything. In order to use the z/OS solution in your company's environment, you will need to adapt the model and resource files using the Kobee Resource Configurator.

Kobee Resource Configurator

Click the Kobee Resource Configurator icon on your Windows desktop icon to open Kobee, and log in with the following credentials:

Username: global
Password: global

Kobee Resource Configurator is the application where you can edit and generate the required models and resources used by the z/OS solution Phases. You can view the configuration of the z/OS demo project in the configuration set "DemoZOS".



General

- z/OS FTP Parameters
  - parmsFTPZOS.properties
- OS Family
  - osfamily.properties
- Environment
  - environment_build.properties
  - environment_TEST_deploy.properties
  - environment_PROD_deploy.properties
- Default Project
  - defaultProject.properties

Preprocessor

- DefaultPgms
  - defaultPgms.properties
- ObjectTypes

Build Environment

- Abendaid
- Cics
  - environment_build_cics.properties
- Datacom
- DB2
  - environment_build_db2.properties
- Endevor
- IDF
- IDMS
- IMS
- J-MAN
- LUW
- QMF
- SDF2
  - environment_build_sdf2.properties

Deploy Environment

- Abendaid
- Cics
  - environment_TEST_deploy_cics.properties
  - environment_PROD_deploy_cics.properties
- Cortex
- DCF
- Datacom
- DB2
  - environment_TEST_deploy_db2.properties
  - environment_PROD_deploy_db2.properties
- Endevor
- IDMS
- J-MAN
- QMF
- Smarttest

If you would like to customize the z/OS demo project configuration in order to test your own sources or to connect to your own mainframe, we strongly suggest to read the document on configuring Kobee for z/OS as it covers working with KRC in detail as well as extra information you will be needing to make the solution work.

Document link:
https://www.kobee.io/documents/integrations/zos/kobee-devops-for-zos-mainframe.pdf

More information

For more in-depth information, refer to the following documentation:

- Kobee User Guide
- How to Guide - Using and Developing Custom Phases in Kobee
- Kobee Installation Guides

You can find those documents on our website https://docs.kobee.io

If you still did not find all the answers to your questions, do not hesitate to contact us at:
https://www.kobee.io/company/contact

IKAN Development
Kardinaal Mercierplein 2
2800 Mechelen, Belgium
Tel. +32 15 797306

info@kobee.io
www.kobee.io