

An IBM TechDoc from

March, 2023 - Release 1.2

IBM Z DevOps Acceleration Program

Troubleshooting z/OS DevOps Pipelines

Author:

Nelson Lopez - IBM DevOps Specialist

Reviewers:

IBM Z DevOps Acceleration Team

Abstract

This document is meant to help troubleshoot basic zDevOps issues. The focus is on IBM Dependency Based Build, dbb-zappbuild Groovy scripts, and UrbanCode Deploy.



Table of Contents

Introduction	3
A Generic zDevOps Architecture.....	3
DBB Groovy Build Scripts - dbb-zappbuild	4
Overview of the zDevOps Logs	5
Some Common Git Errors	6
Log Options for DBB and zAppBuild.....	8
Build.Groovy stdout	8
Enabling zAppBuild's Verbose Logging Mode.....	9
DBB Simple Logging Façade for Java (SLF4J).....	10
Tracing the Publish Phase	11
UCD Application Component Process Log	13
UCD Agent Log	16
Diagnosing DBB File Allocation Errors (BPXWDYN also called DYNALLOC)	17
Tracing a Pipeline	19
Conclusion.....	20
Troubleshooting References.....	20
Additional References.....	20

Introduction

This document is meant to help troubleshoot basic zDevOps issues. The focus is on IBM Dependency Based Build (DBB Version 2), dbb-zappbuild (any version), and UrbanCode Deploy (UCD- version 7 or higher). While this document does not cover all the possible designs and tools, you may find it useful in diagnosing problems across other components.

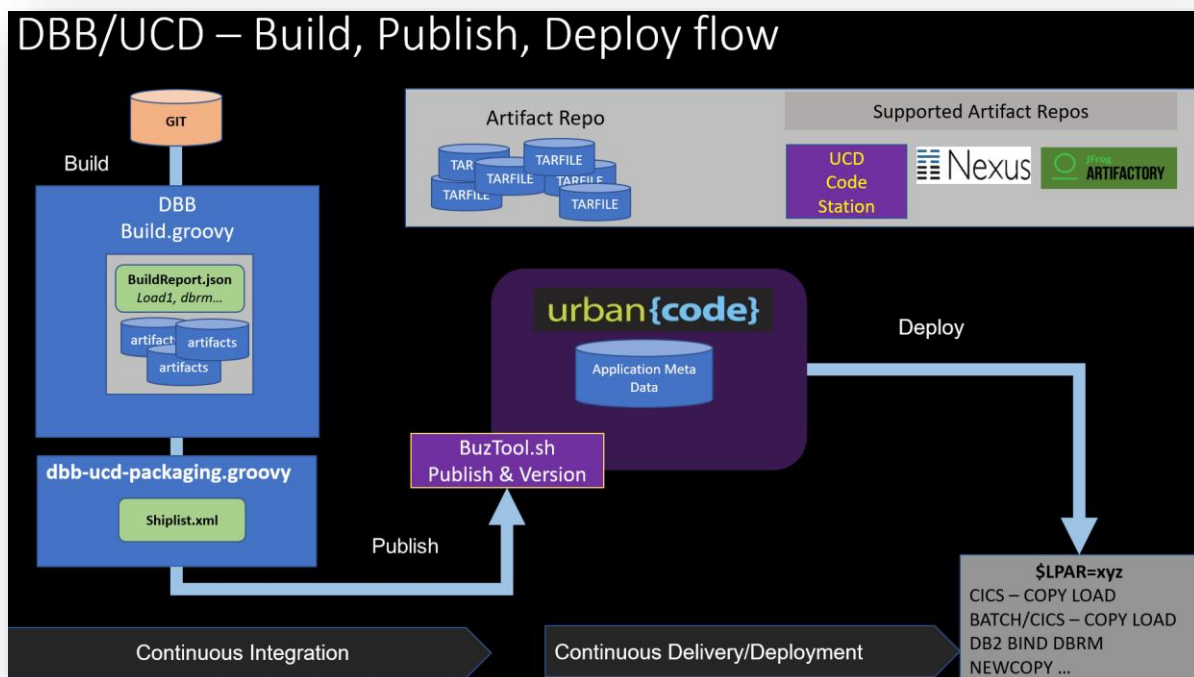
TIP: Always apply the latest version and fixes to your DevOps software stack. Refer to the component's documents for more detailed troubleshooting help.

A Generic zDevOps Architecture

Troubleshooting across distributed processes can be complicated. An important step is to understand the overall architecture and workflow to isolate and resolve issues.

The diagram below is a generic DevOps workflow for z/OS using Git, DBB, Groovy scripts and UCD. The flow is a CI pipeline build with IBM's sample `build.groovy` script (top left). A successful build produces artifacts that are passed on to an IBM provided sample publishing script called `dbb-ucd-packaging.groovy`. This tool reads DBB's `BuildReport.json` output to create a UCD shiplist. It also tars the artifact(s) and invokes the UCD 'Buztool' agent interface to publish the tar file in an artifact repository. A CD pipeline, like UrbanCode Deploy in this case, then runs a process to download the tar, and deploy the individual artifacts to a target z/OS environment.

Within this flow there are several logs that can help trace and troubleshoot problems. This document highlights where to find them and how to use them.

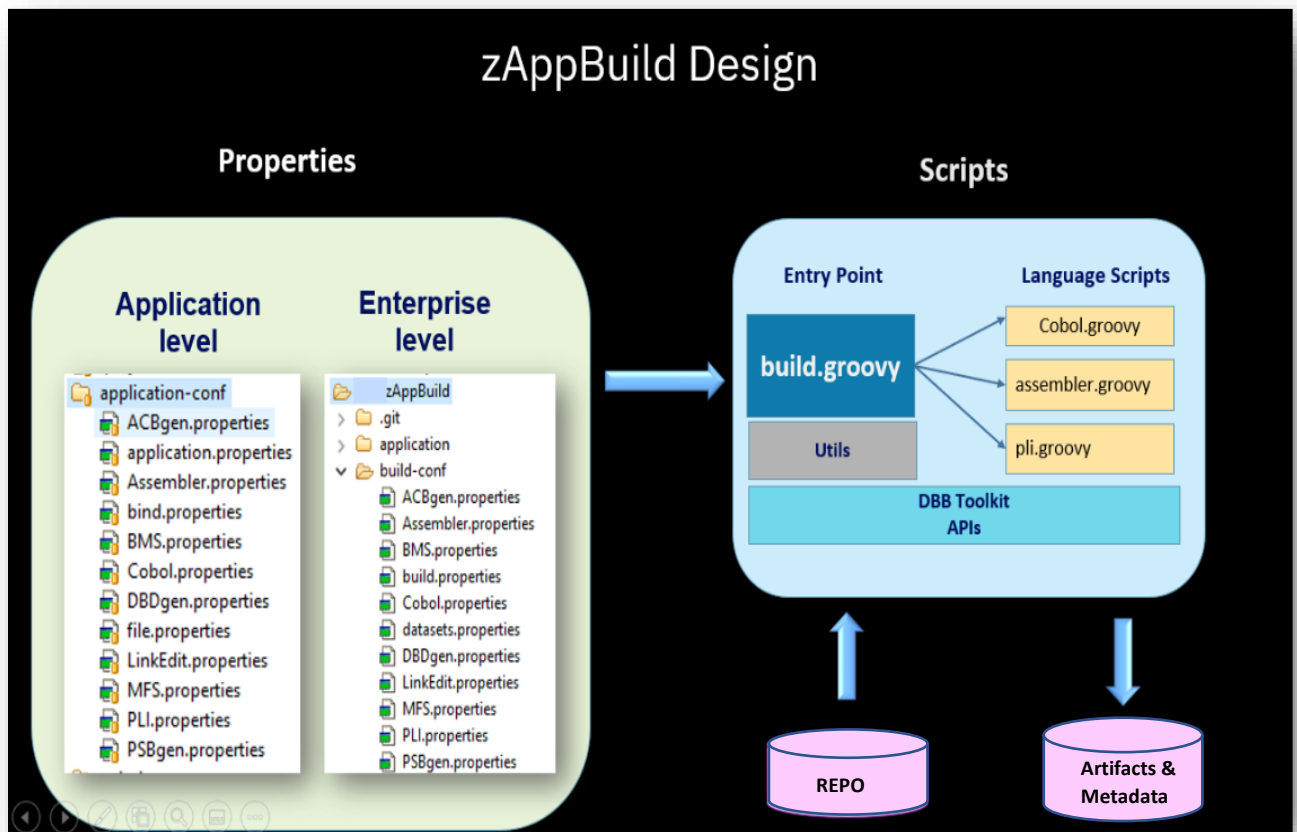


DBB Groovy Build Scripts - dbb-zappbuild

The IBM sample framework, [dbb-zappbuild \(zAppBuild for short\)](#), is composed of several Groovy scripts that automate the build process. They run on z/OS's Unix System Service (USS) and invoke [DBB toolkit](#) Java [APIs](#) to allocate MVS PDSs, access Git repos and run the MVS compiler, linkeditor (binder) and other processes. The diagram below highlights the overall framework.

The main process is called by `build.groovy` and is invoked under the DBB provided `groovyz` command to perform User Builds (in IDz or VS Code) and CI pipeline *Impact* builds. Its main inputs are command line arguments, property files and source code from Git repo(s). The outputs are build artifacts like load modules, logs and metadata (called **collections** in DBB). Metadata is DBB internal information about an application, it's dependencies and build results. Depending on the configuration, metadata can be stored in DB2 or the USS file system.

```
$DBB_HOME/bin/Groovyz build.groovy \  
--workspace /u/build/repos \  
--application app1 \  
--outDir /u/build/out \  
--hlq BUILD.APP1 \  
--impactBuild
```



Overview of the zDevOps Logs

Each stage of the clone, build, publish and deploy workflow will have a log(s) to help trace and troubleshoot the process.

`build.groovy` creates/updates metadata as shown on the left of the image below. The configuration shown here places these files on the USS file system and not DB2. The `buildresults` folder will have information for each build like the `buildreport` and, when configured, compiler and linkedit output. These folders are grouped under the applications name followed by the branch that was processed (such as `poc-app-develop`) where `develop` is the branch in this example.

Output artifacts are stored in a working directory as shown on the right side. There, you will find the `build.groovy` '`build.list`' and `BuildReport.json` which are all the items of a build. The publish Groovy `shiplist.xml` and `buztool`'s `buztool.output` can also be found in this location. These files may have helpful information in diagnosing problems. In a successful run, they should all be present. If one is missing it would indicate some problem with that stage of the process.

Files ending in `.log` like `DATBATCH.cobol.log` on the right are compiler and linkedit output (sysprint) for each program in the `build.list`. Developers will find them useful when resolving compile or link issues.

In UCD (bottom right), the server stores the logs for each step of a deploy process. These logs are accessible from UCD's UI and provide details on what was deployed and any errors that may have occurred.

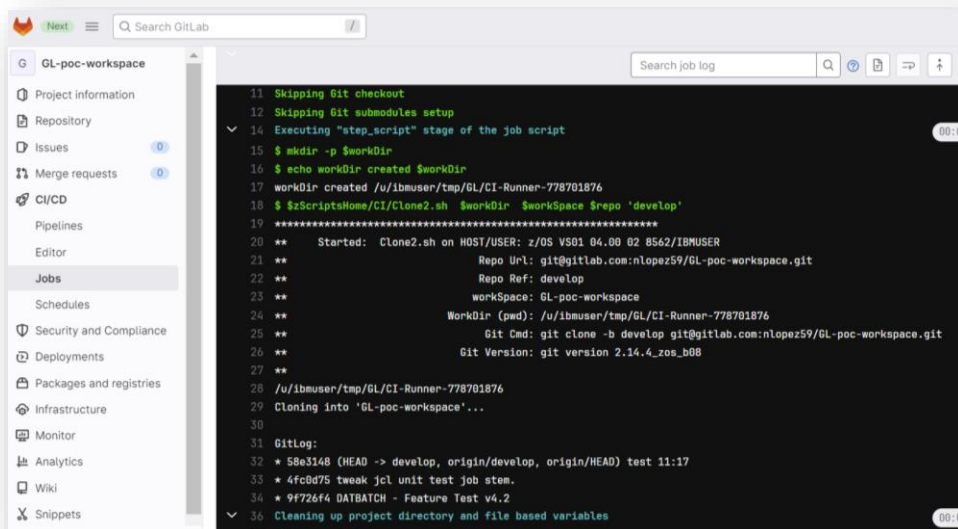
The image is a screenshot of a presentation slide titled "DBB/UCD – Build, Publish, Deploy Logs". It features two side-by-side file explorer windows. The left window, titled "Remote Systems", shows a directory structure on the USS file system. It includes folders like `metadata`, `dbb_default`, and `collections`. Under `collections`, there are sub-folders for `poc-app-develop`, `poc-app-develop-outputs`, and `buildresults`. The `buildresults` folder contains sub-folders for individual builds, such as `build.20230223.122832.028` and `build.20230224.065518.055`. The right window, also titled "Remote Systems", shows a directory structure on a Jenkins agent. It includes folders like `caches`, `remoting`, and `workspace`. Under `workspace`, there are sub-folders for `poc-demo_develop` and `poc-workspace`. The `poc-workspace` folder contains files like `.git`, `.github`, `poc-app`, `.gitattributes`, `.gitignore`, `buildList.txt`, `BuildReport.html`, `BuildReport.json`, `buztool.output`, `DATBATCH.cobol.log`, `Jenkinsfile`, and `shiplist.xml`. A blue arrow labeled "Publish.groovy" points from the `buildresults` folder in the left window to the `BuildReport.json` file in the right window. A purple arrow labeled "Buztool" points from the `buztool.output` file in the right window to the `BuildReport.json` file. In the bottom right corner, there is a logo for "urban{code}" with the text "Linux Server" and "Component process logs" below it.

Some Common Git Errors

The first step in any pipeline is cloning a Git repo(s) to USS. There can be many reasons why a clone may fail. Some common reasons include:

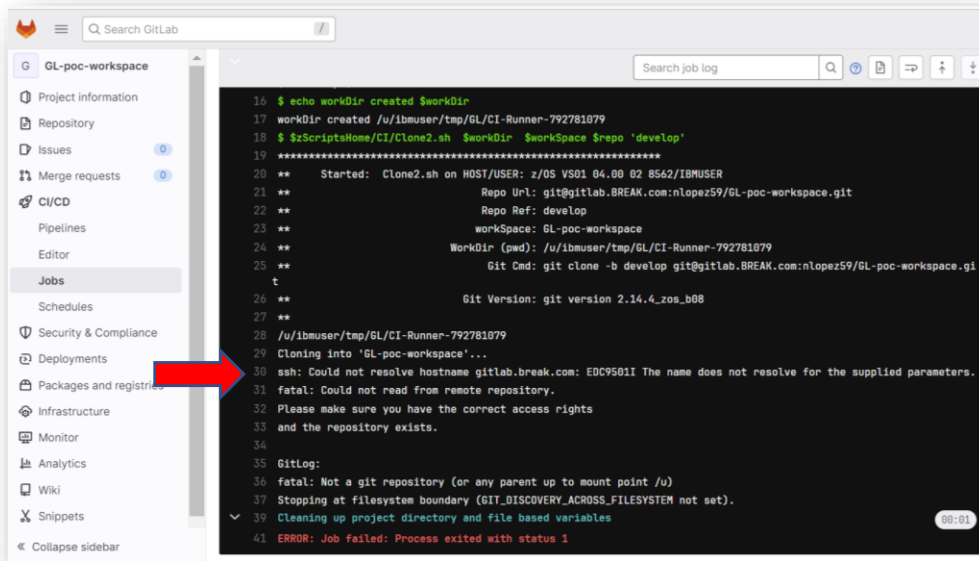
- Security:
 - o When cloning with SSH, ensure the keys are properly defined.
 - o When cloning with HTTPS ensure the user/password provided is allowed access to the repo
- Network:
 - o Ensure firewall rules and proxy setting, when used, are properly configured to allow the Rocket Git client on USS to clone from the remote Git Server
- Other:
 - o Ensure the target USS directory has sufficient free space and permissions to accept the clone
 - o Ensure the [Rocket Git environment](#) variables are properly configured

Depending on the orchestrator, the Git clone stdout/stderr will provide details on any failure. Below is an example clone in a GitLab job.



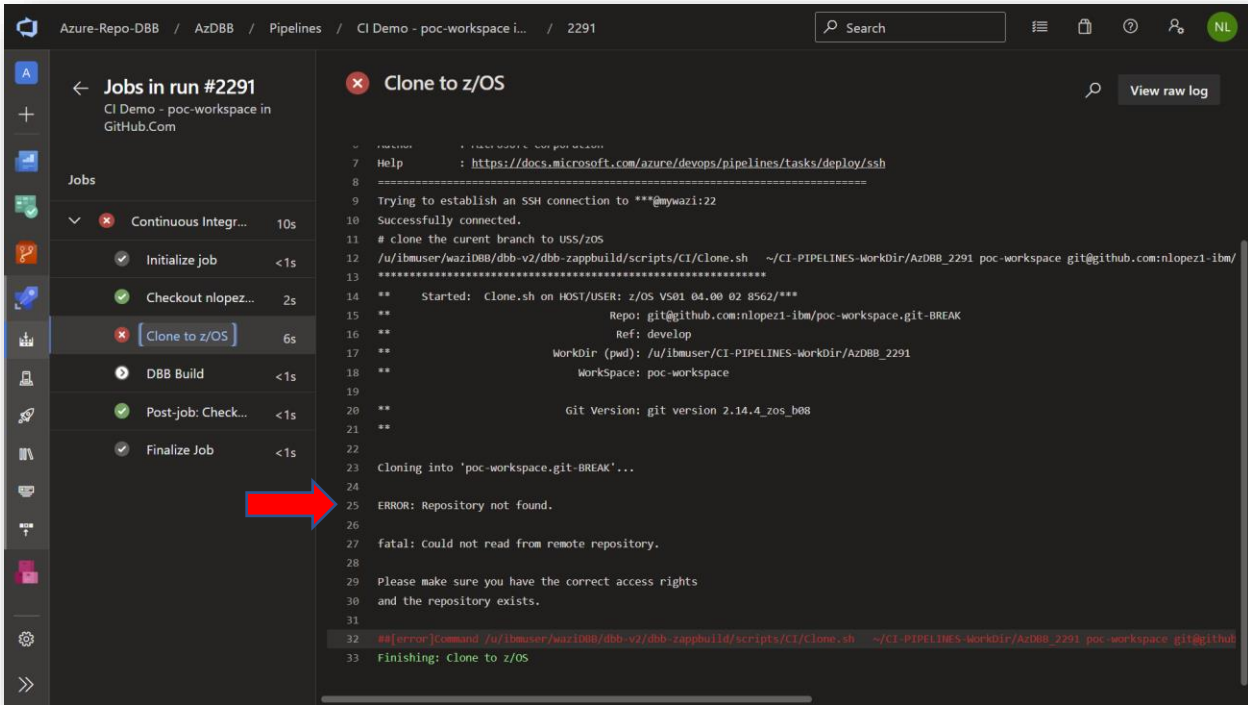
```
11 Skipping Git checkout
12 Skipping Git submodules setup
14 Executing "step_script" stage of the job script
15 $ mkdir -p $workDir
16 $ echo workDir created $workDir
17 workDir created /u/ibmuser/tmp/GL/CI-Runner-778701876
18 $ $ScriptHome/CI/Clone2.sh $workDir $workspace $repo 'develop'
19 *****
20 ** Started: Clone2.sh on HOST/USER: z/OS VSO1 04.00 02 8562/IBMUSER
21 ** Repo Url: git@gitlab.com:nlopez59/GL-poc-workspace.git
22 ** Repo Ref: develop
23 ** workspace: GL-poc-workspace
24 ** WorkDir (pwd): /u/ibmuser/tmp/GL/CI-Runner-778701876
25 ** Git Cmd: git clone -b develop git@gitlab.com:nlopez59/GL-poc-workspace.git
26 ** Git Version: git version 2.14.4_zos_b08
27 **
28 /u/ibmuser/tmp/GL/CI-Runner-778701876
29 Cloning into 'GL-poc-workspace'...
30
31 GitLog:
32 * 58e3148 (HEAD -> develop, origin/develop, origin/HEAD) test 11:17
33 * 4fc0d75 tweak jcl unit test job stem.
34 * 9f726f4 Batching - Feature Test v4.2
36 Cleaning up project directory and file based variables
```

This example shows a failed clone's log and detailed error message. In this case, the repo name was mistyped (line 30).



```
16 $ echo workDir created $workDir
17 workDir created /u/ibmuser/tmp/GL/CI-Runner-792781079
18 $ $ScriptsHome/CI/Clone2.sh $workDir $workSpace $repo 'develop'
19 *****
20 ** Started: Clone2.sh on HOST/USER: z/OS V501 04.00 02 8562/IBMUSER
21 **                               Repo Url: git@gitlab.BREAK.com:nlopez59/GL-poc-workspace.git
22 **                               Repo Ref: develop
23 **                               workSpace: GL-poc-workspace
24 **                               WorkDir (pwd): /u/ibmuser/tmp/GL/CI-Runner-792781079
25 **                               Git Cmd: git clone -b develop git@gitlab.BREAK.com:nlopez59/GL-poc-workspace.git
26 **                               t
27 **                               Git Version: git version 2.14.4_zos_b08
28 **                               /u/ibmuser/tmp/GL/CI-Runner-792781079
29 Cloning into 'GL-poc-workspace'...
30 ssh: Could not resolve hostname gitlab.break.com: EDC9501I The name does not resolve for the supplied parameters.
31 fatal: Could not read from remote repository.
32 Please make sure you have the correct access rights
33 and the repository exists.
34
35 Gitlog:
36 fatal: Not a git repository (or any parent up to mount point /u)
37 Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
38 Cleaning up project directory and file based variables
41 ERROR: Job failed: Process exited with status 1
```

In an Azure DevOps CI pipeline, a failed clone's log would show similar details (line 25).



```
7 Help : https://docs.microsoft.com/azure/devops/pipelines/tasks/deploy/ssh
8
9 Trying to establish an SSH connection to **@mywazi:22
10 Successfully connected.
11 # clone the current branch to USS/ZOS
12 /u/ibmuser/wazi088/dbb-v2/dbb_zappbuild/scripts/CI/clone.sh ~/CI-PIPELINES-WorkDir/AzDBB_2291 poc-workspace git@github.com:nlopez1-ibm/
13 *****
14 ** Started: clone.sh on HOST/USER: z/OS V501 04.00 02 8562/**
15 **                               Repo: git@github.com:nlopez1-ibm/poc-workspace.git-BREAK
16 **                               Ref: develop
17 **                               WorkDir (pwd): /u/ibmuser/CI-PIPELINES-WorkDir/AzDBB_2291
18 **                               Workspace: poc-workspace
19
20 **                               Git Version: git version 2.14.4_zos_b08
21 **
22
23 Cloning into 'poc-workspace.git-BREAK'...
24
25 ERROR: Repository not found.
26
27 fatal: Could not read from remote repository.
28
29 Please make sure you have the correct access rights
30 and the repository exists.
31
32 ##[error]Command /u/ibmuser/wazi088/dbb-v2/dbb_zappbuild/scripts/CI/clone.sh ~/CI-PIPELINES-WorkDir/AzDBB_2291 poc-workspace git@github
33 Finishing: Clone to z/OS
```

Log Options for DBB and zAppBuild

build.groovy stdout

The main zAppBuild `build.groovy` script's log is stdout/stderr. It is visible from a command line invocation as well as from most CI pipelines as shown in the Jenkins example below. The log shows:

- what was built
- the build output location (artifacts)
- the collection name (metadata)
- build result - clean or failed
- number of input files processed
- and other valuable details

```
** Build-v2 start at 20230227.022755.027
** Build output located at /u/ibmuser/jenkins-agent/workspace/poc-demo_develop/poc-workspace
** Build result created for BuildGroup:poc-app-develop BuildLabel:build.20230227.022755.027
** Adding poc-app/cobol/datbatch.cbl to Building build list
** Writing build list file to /u/ibmuser/jenkins-agent/workspace/poc-demo_develop/poc-workspace/buildList.txt
** Scanning source code.
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.codehaus.groovy.vmpugin.v9.Java9 (file:/u/ibmuser/waziDBB/groovy/lib/groovy-4.0.3.jar) to method
sun.nio.fs.UnixFileSystem.getPathMatcher(java.lang.String)
WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.vmpugin.v9.Java9
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
** Populating file level properties from individual property files.
** Invoking build scripts according to build order: BMS.groovy,MFS.groovy,Cobol.groovy,Assembler.groovy,PLI.groovy,LinkEdit.groovy,DBDgen.groovy,PSBgen.groovy,Transfer.groovy
** Building files mapped to Cobol.groovy script
*** Building (CG V2) file poc-app/cobol/datbatch.cbl
** Writing build report data to /u/ibmuser/jenkins-agent/workspace/poc-demo_develop/poc-workspace/BuildReport.json
** Writing build report to /u/ibmuser/jenkins-agent/workspace/poc-demo_develop/poc-workspace/BuildReport.html
** Updating build result BuildGroup:poc-app-develop BuildLabel:build.20230227.022755.027
** Build ended at Mon Feb 27 14:28:01 EST 2023
** Build State : CLEAN
** Total files processed : 1
** Total build time : 5.921 seconds
```


Enabling zAppBuild's Verbose Logging Mode

`build.groovy` accepts an argument to enable verbose logging. It is the `--verbose` or `-v` option.

Refer to the `dbb-zappbuild build.groovy` readme for more information

<https://Github.com/IBM/dbb-zappbuild/blob/main/BUILD.md>

```
$DBB_HOME/bin/groovy <zAppBuildLocation>/build.groovy [options] buildfile

buildfile (optional): Path of the source file to build (absolute or relative to workspace).
If buildfile is a text file (*.txt), then it is assumed to be a build list file.

Options:

required options:
-w,--workspace <arg>      Absolute path to workspace (root) directory
                           containing all required source directories
-a,--application <arg>   Application directory name (relative to workspace)
-o,--outdir <arg>       Absolute path to the build output root directory
-h,--hlq <arg>          High level qualifier for partition data sets

build options:
-f,--fullBuild           Flag indicating to build all programs for
                           the application
-i,--impactBuild        Flag indicating to build only programs impacted
                           by changed files since last successful build.
-b,--baselineRef        Comma separated list of git references to overwrite
                           the baselineHash hash in an impactBuild scenario.
-m,--mergeBuild         Flag indicating to build only source code changes which will be
                           merged back to the mainBuildBranch.

-s,--scanOnly           Flag indicating to only scan source files for application without building anything (deprecated use --ss)
-ss,--scanSource        Flag indicating to only scan source files for application without building anything
-sl,--scanLoad          Flag indicating to only scan load modules for application without building anything
-sa,--scanAll           Flag indicating to scan both source files and load modules for application without building anything

-r,--reset              Deletes the application's dependency collections
                           and build result group from the DBB repository

-v,--verbose            Flag to turn on script trace
-d,--debug              Flag to build modules for debugging with
                           IBM Debug for z/OS
-l,--logEncoding <arg> Encoding of output logs. Default is EBCDIC
                           directory for user build
```

The following example shows verbose output of property files loaded, PDS allocations, dependency resolution rules and much more.

```
$DBB_HOME/bin/Groovy build.Groovy \
--workspace /u/build/repos \
--application app1 \
--outdir /u/build/out \
--verbose \
--hlq BUILD.APP1 \
--impactBuild
```

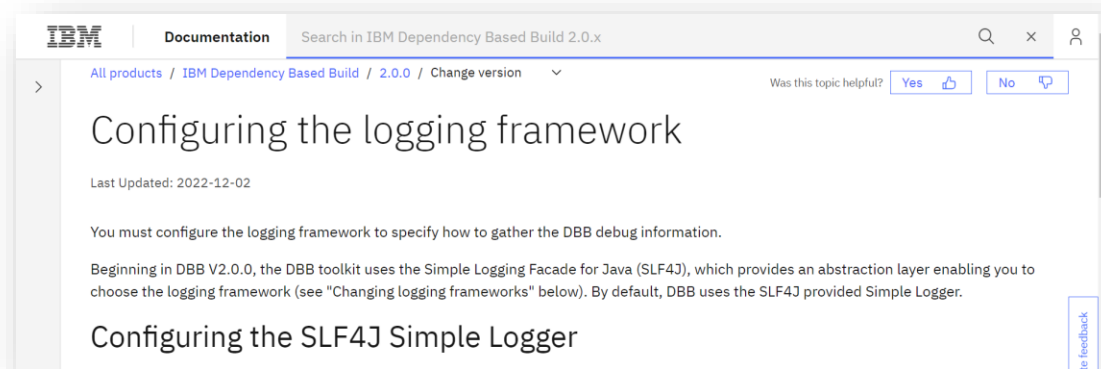
```
*** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/BMS.properties
*** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/Cobol.properties
*** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/LinkEdit.properties
*** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/PLI.properties
java.version=8_0_6_0 - pmz6480sr6-20191107_01(SR6)
java.home=/usr/lpp/java/J8_0_64
user.dir=/u/nlopez/DAT-Demo-Workspace
*** Build properties at start up:
psbgen_pgm=ASMA90
assembler_macroPDS=DAT.DEMO.MACRO
cobol_scanLoadModule=true
assembler_srcPDS=DAT.DEMO.ASM
psbgen_deployType=PSBLIB
cobol_printTempOptions=cyl space(5,5) unit(vio) blksize(133) trec(133) recfm(f,b) new
```

```
*** Resolution rules for Mortgage-SA-DAT/cobol/mqsamp.cb1:
*** Physical dependencies for Mortgage-SA-DAT/cobol/mqsamp.cb1:
{"name":"CMQMDV","library":"SYSLIB","category":"COPY","resolved":false}
{"name":"CMQODV","library":"SYSLIB","category":"COPY","resolved":false}
{"name":"CMQPMOV","library":"SYSLIB","category":"COPY","resolved":false}
{"name":"CMQV","library":"SYSLIB","category":"COPY","resolved":false}
{"name":"MQCONN","library":"SYSLIB","category":"CALL","resolved":false}
cobol_compiler_pgm for Mortgage-SA-DAT/cobol/mqsamp.cb1 = APDATA
```

```
required props = buildOrder,buildListFileExt,languagePropertyQualifiers
*** Repository client created for https://dbbdev.rtp.raleigh.ibm.com:9443/dbb/
*** Build output located at /u/nlopez/DAT-Demo-Logs/build.20200630.030053.000
*** Creating / verifying build dataset DAT.DEMO.ASM
*** Creating / verifying build dataset DAT.DEMO.MACRO
*** Creating / verifying build dataset DAT.DEMO.OBJ
*** Creating / verifying build dataset DAT.DEMO.DBRM
*** Creating / verifying build dataset DAT.DEMO.LOAD
```

DBB Simple Logging Façade for Java (SLF4J)

DBB supports java logging with SLF4J. For details on enabling this level of tracing visit <https://www.ibm.com/docs/en/dbb/2.0.0?topic=customization-logging-framework>

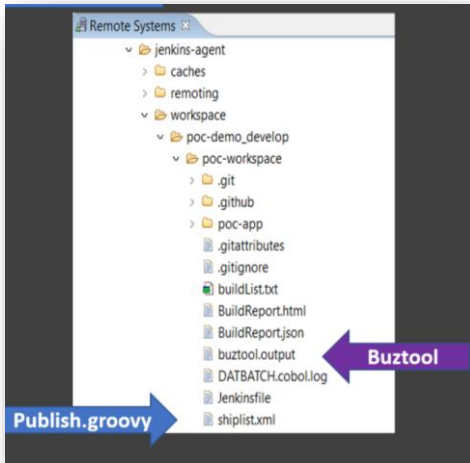


By default, this feature is disabled. When enabled it provides detailed information and error codes for troubleshooting DBB API issues as shown in the example below.

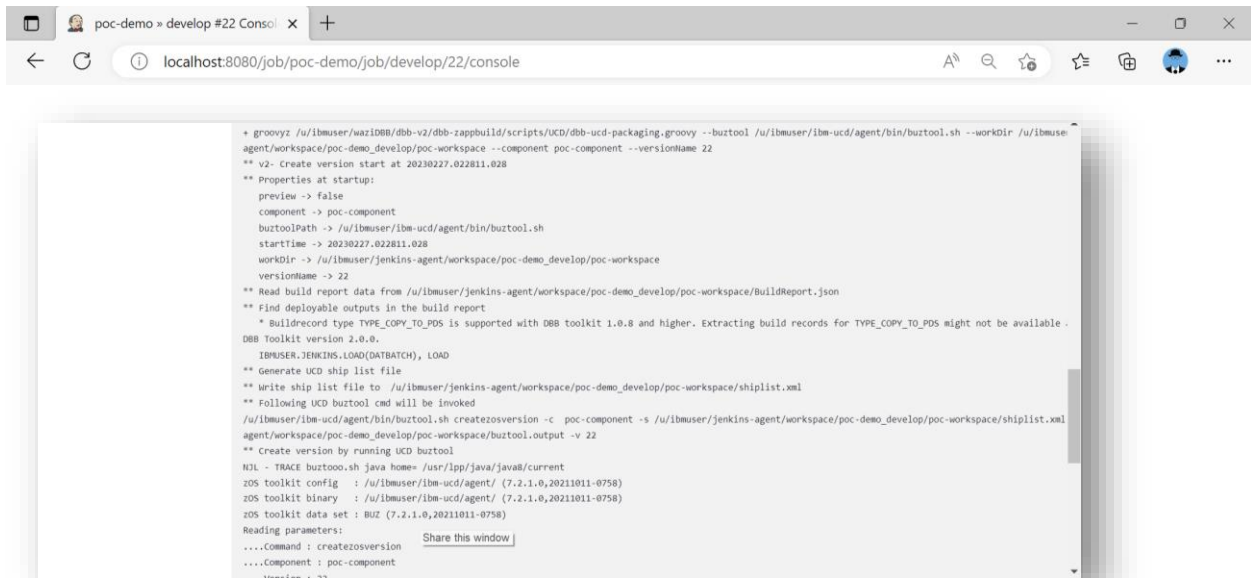
```
DEBUG 2019-10-19 09:08:29.123 22587 com.ibm.dbb.build.internal.Utils [main] bpxwdyn: alloc dd(SYSUT11) cyl
space(5,5) unit(vio) blksize(80) lrecl(80) recfm(f,b) new
DEBUG 2019-10-19 09:08:29.127 22591 com.ibm.dbb.build.internal.Utils [main] bpxwdyn: alloc dd(SYSUT12) cyl
space(5,5) unit(vio) blksize(80) lrecl(80) recfm(f,b) new
DEBUG 2019-10-19 09:08:29.130 22594 com.ibm.dbb.build.internal.Utils [main] bpxwdyn: alloc dd(SYSUT13) cyl
space(5,5) unit(vio) blksize(80) lrecl(80) recfm(f,b) new
DEBUG 2019-10-19 09:08:29.133 22597 com.ibm.dbb.build.internal.Utils [main] bpxwdyn: alloc dd(SYSUT14) cyl
space(5,5) unit(vio) blksize(80) lrecl(80) recfm(f,b) new
```

Tracing the Publish Phase

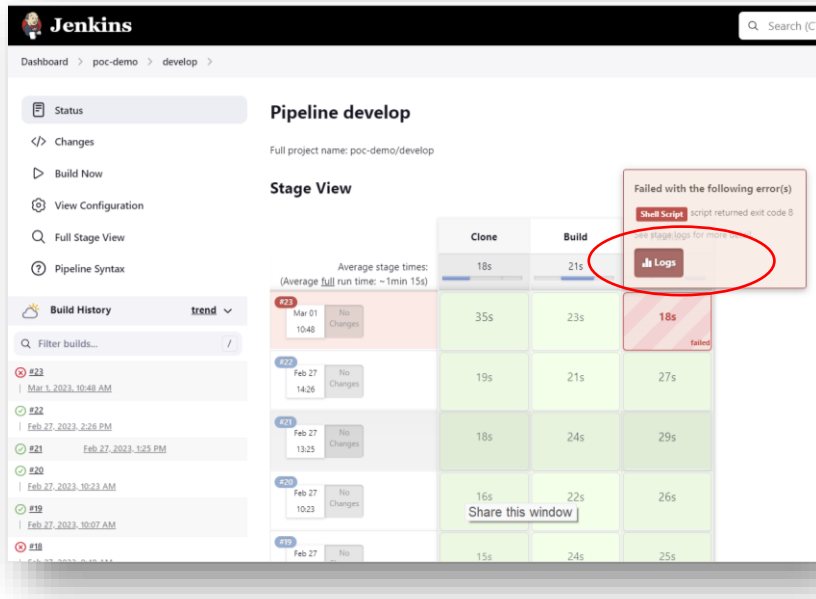
As explained in the introduction section, a pipeline can include a step to package (tar) and publish artifacts in UCD. The standard IBM DevOps workflow provides a sample script called `dbb-ucd-packaging.groovy`. It produces a log and converts the DBB `BuildReport.json` output file into a UCD shiplist. It then calls Buztool, the USS UCD Agent's main process, which reads the shiplist and calls the UCD Server to publish the component's artifacts (tar file). Tar files can be stored in one of these supported artifact repositories – UCD's Code Station, Artifactory or Nexus.



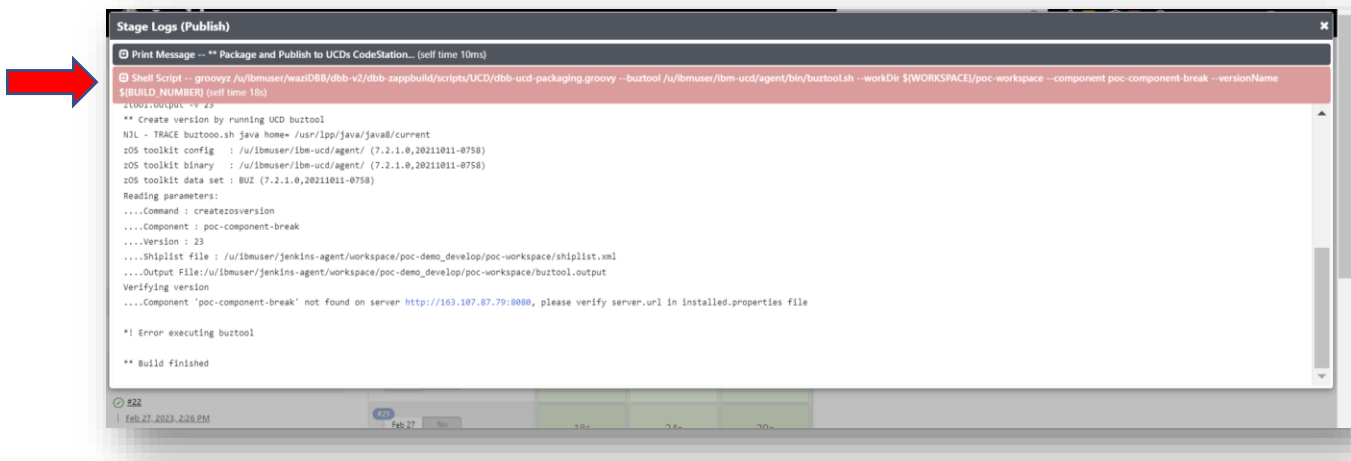
The log of an orchestrator like Jenkins will show the publish step's results.



In the example below of a failed publish step in Jenkins you can access the step's log files to get more details by clicking on the Logs icon shown in the summary error message(s).



Click the message to get more details of the failure. In this case the UCD Component name was not found in UCD - it was mistyped.



Other common issues during this phase include:

- Missing access rights to UCD, Artifactory or Nexus.
- Network failure due to firewall, proxy settings or other.
- Missing or mis-spelled resource definitions in the artifact server.
- No artifacts passed from DBB. Sometimes DBB Impact mode may not produce an output artifact. This can be related to corrupt metadata which can be refreshed with a DBB --reset run.

UCD Application Component Process Log

UCD process logs are accessible from the component's version history page using the 'view request' link as shown below.

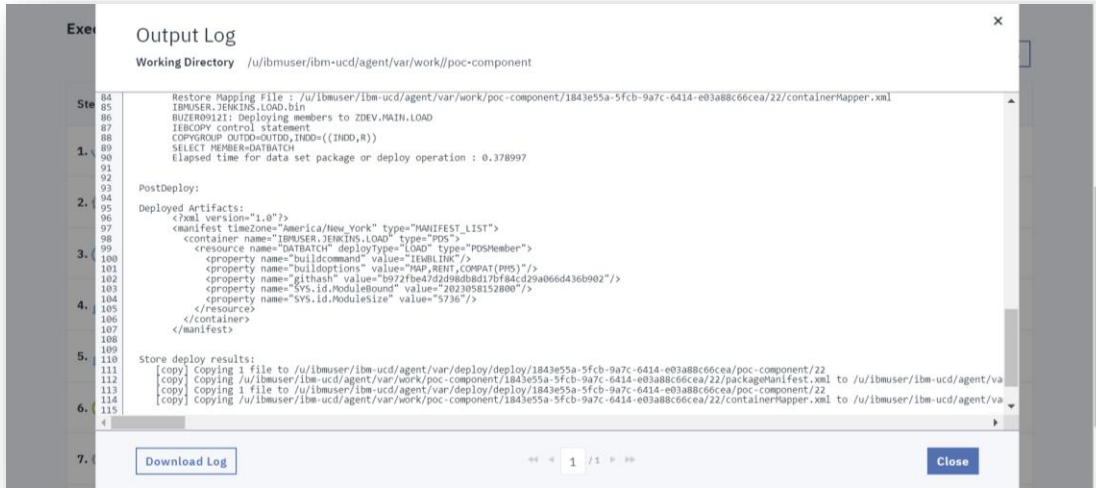
The screenshot shows the UrbanCode Deploy interface. The top navigation bar includes 'UrbanCode Deploy', 'Dashboard', 'Components', 'Applications', 'Configuration', 'Processes', 'Resources', 'Calendar', and 'More'. The user is logged in as 'admin'. The breadcrumb trail is 'Components / poc-component / Versions / Version: 22'. A 'Message of the Day' banner is visible. Below it, the 'Version: 22' details are shown, including 'Created By: admin', 'Created On: 2/27/2023, 2:27 PM', and 'Repository Type: CodeStation'. A 'Links' section indicates 'No links found.' and a 'View all links' link is present. The main content area has tabs for 'Main', 'Configuration', and 'History'. A table lists process instances with columns: Process, Resource, Application, Environment, Scheduled For, By, and Status. The first row shows a process with status 'Success'. A red circle highlights the 'View Request' link in the status column. Below the table, there are pagination controls and a 'Refresh' button.

From there you can select the 'View Output Log' option to review the progress of any step.

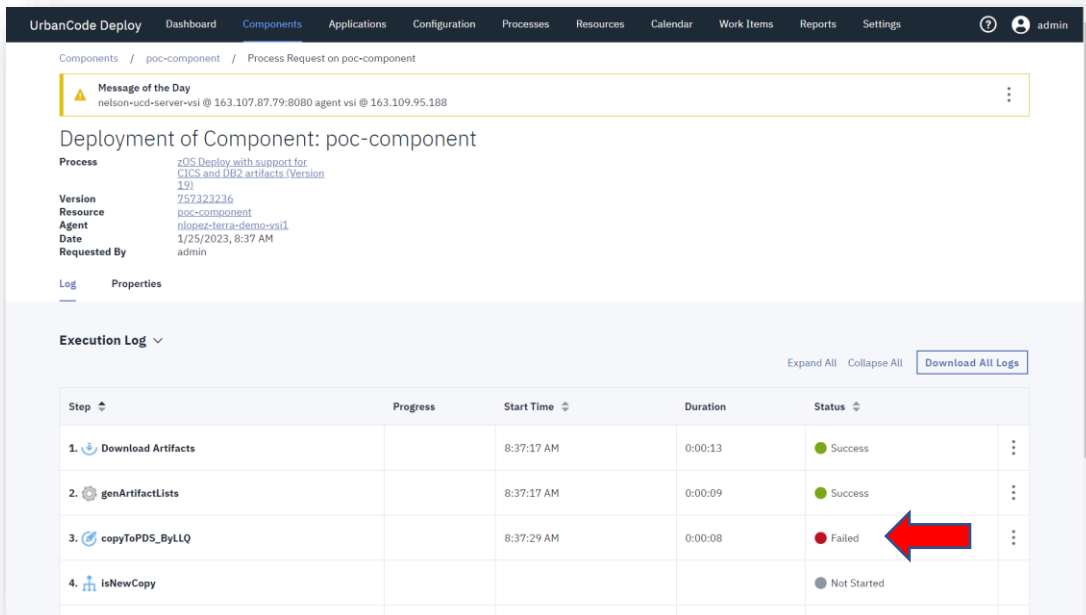
The screenshot shows the 'Execution Log' page. It features a table with columns: Step, Progress, Start Time, Duration, and Status. The table lists seven steps, with the first five completed successfully. A red circle highlights the 'View Output Log' link in the status column of the third step. A dropdown menu is open, showing 'View Output Log' and 'View Input/Output Properties'. At the bottom, there is a 'Total Execution' summary with a progress bar showing 3/5 steps completed.

Step	Progress	Start Time	Duration	Status
1. Download Artifacts		2:27:52 PM	0:00:09	Success
2. genArtifactLists		2:27:52 PM	0:00:08	Success
3. copyToPDS_ByLLQ		2:28:01 PM	0:00:09	Success
4. isNewCopy		2:28:10 PM	0:00:00	Success
5. isDB2		2:28:10 PM	0:00:00	Success
6. SubmitDB2BindJCL				Not Started
7. RunNewCopy(s)				Not Started
Total Execution	3 / 5	2:27:52 PM	0:00:18	Success

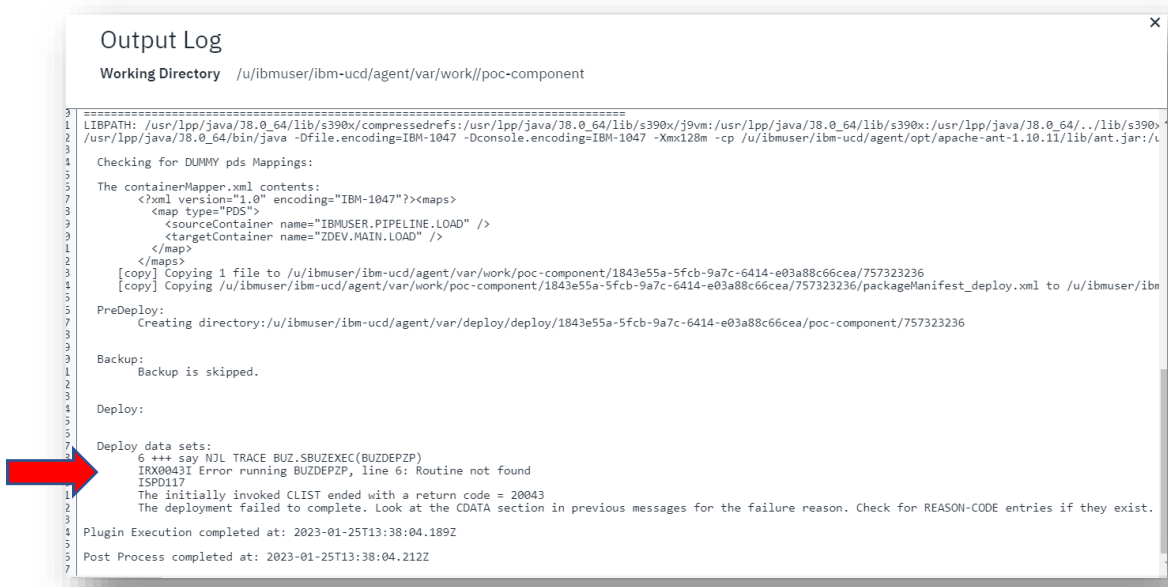
The example below shows the “CopyToPDS_ByLLQ” process step’s log. Any error(s) would be displayed here.



An example failed step would look like this:



Open the step's log to view more detail. In this case, it seems there is a configuration issue. Refer to the system programmer to repair or reinstall the agent on USS.



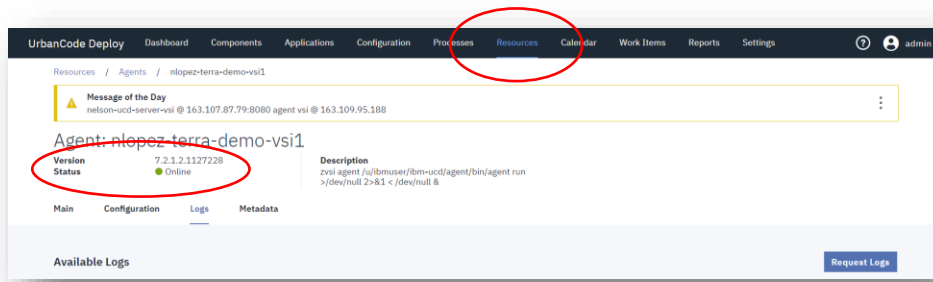
```
Output Log
Working Directory /u/ibmuser/ibm-ucd/agent/var/work/poc-component

=====
1 LIBPATH: /usr/lpp/java/38.0_64/lib/s390x/compressedrefs:/usr/lpp/java/38.0_64/lib/s390x/j9vm:/usr/lpp/java/38.0_64/lib/s390x:/usr/lpp/java/38.0_64/./lib/s390x
2 /usr/lpp/java/38.0_64/bin/java -Dfile.encoding=IBM-1047 -Dconsole.encoding=IBM-1047 -Xmx128m -cp /u/ibmuser/ibm-ucd/agent/opt/apache-ant-1.10.11/lib/ant.jar:/u
3
4 Checking for DUMMY pds Mappings:
5
6 The containerMapper.xml contents:
7   <?xml version="1.0" encoding="IBM-1047"?><maps>
8     <map type="PDS">
9       <sourceContainer name="IBMUSER.PIPELINE.LOAD" />
10      <targetContainer name="ZDEV.MAIN.LOAD" />
11    </map>
12  </maps>
13  [copy] Copying 1 file to /u/ibmuser/ibm-ucd/agent/var/work/poc-component/1843e55a-5fcb-9a7c-6414-e03a88c66cea/757323236
14  [copy] Copying /u/ibmuser/ibm-ucd/agent/var/work/poc-component/1843e55a-5fcb-9a7c-6414-e03a88c66cea/757323236/packageManifest_deploy.xml to /u/ibmuser/ibm
15
16 PreDeploy:
17   Creating directory:/u/ibmuser/ibm-ucd/agent/var/deploy/deploy/1843e55a-5fcb-9a7c-6414-e03a88c66cea/poc-component/757323236
18
19 Backup:
20   Backup is skipped.
21
22 Deploy:
23
24 Deploy data sets:
25   0 +++ say NJL TRACE BUZ.SBUZEXEC(BUZDEPZP)
26   IRX0043I Error running BUZDEPZP, line 6: Routine not found
27   ISPD117
28   The initially invoked CLIST ended with a return code = 20043
29   The deployment failed to complete. Look at the CDATA section in previous messages for the failure reason. Check for REASON-CODE entries if they exist.
30
31 Plugin Execution completed at: 2023-01-25T13:38:04.189Z
32
33 Post Process completed at: 2023-01-25T13:38:04.212Z
```

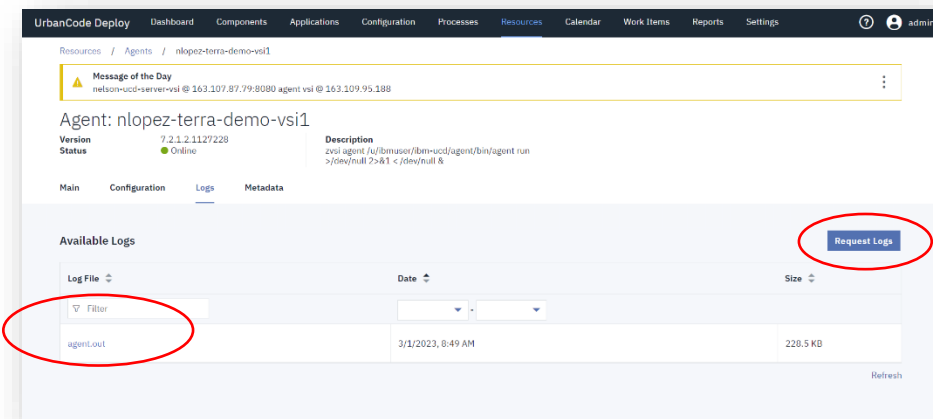
UCD Agent Log

The UCD Agent's log can be accessed from the UCD web interface.

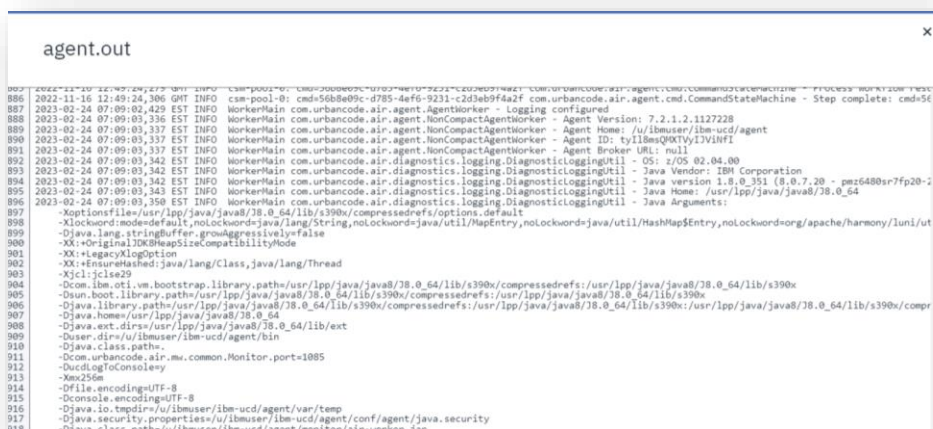
The first step in diagnosing Agent issues is to ensure the agent's "MVS Started Task" is active on the target z/OS environment – the one where artifacts will be deployed. The UCD Resource menu lists all agents. Use that to review your agent's status. "Online" (with a green dot)



From that page, when you "Request Logs", the agent's logs are extracted and available for viewing.



Scroll through the log to identify any unusual error messages.

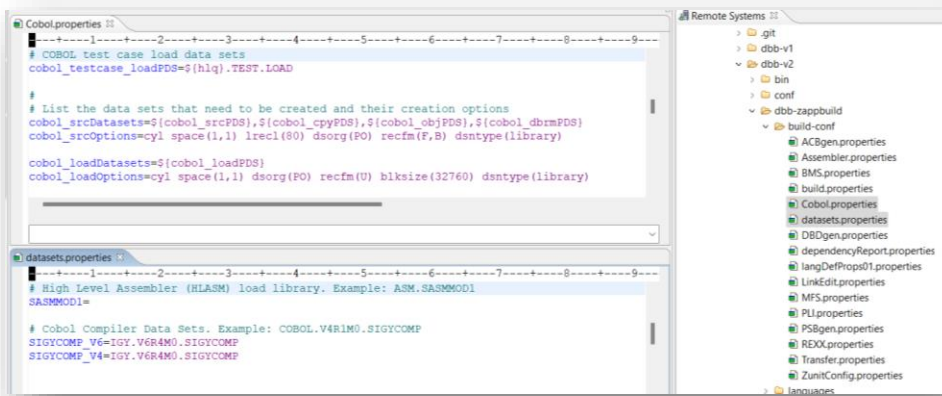


Diagnosing DBB File Allocation Errors (BPXWDYN also called DYNALLOC)

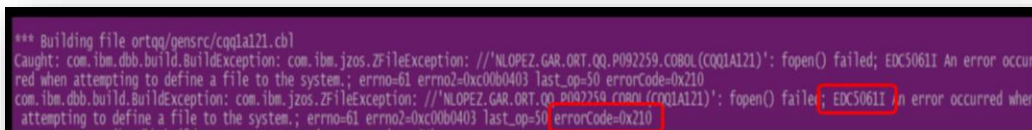
A common error when configuring DBB is with MVS file allocations. The `build.groovy` log will display any system related issues when allocating an MVS dataset.

Some common issues are related to invalid DCB parameters. They are defined in the `dbb-zappbuild/build-config` files that allocate new PDSs like in `cobol.properties` or when allocating system datasets like the cobol compiler in `dataset.properties` as shown below.

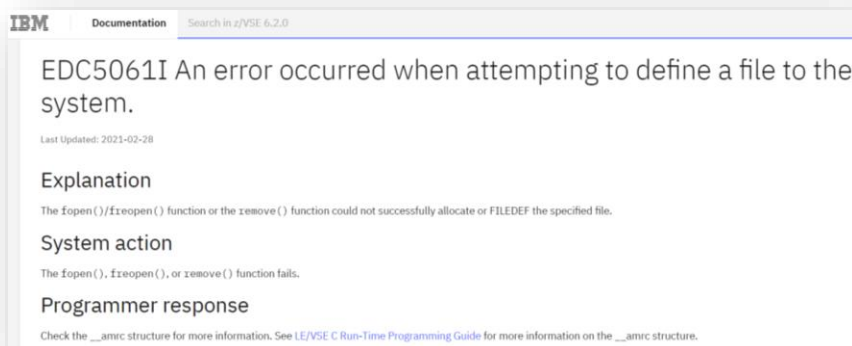
Typically, they are caused by typos in the DCBs, misspelled DSN's of existing libraries or are related to some RACF or SMS policies.



As an example, the `build.groovy` log will show allocation errors as highlighted below.



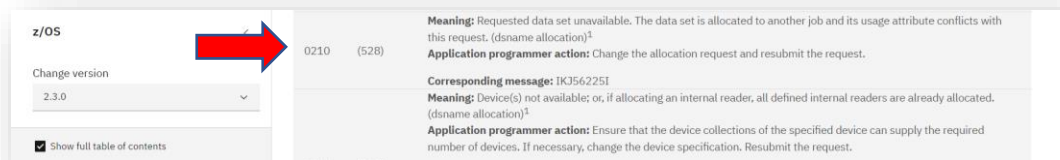
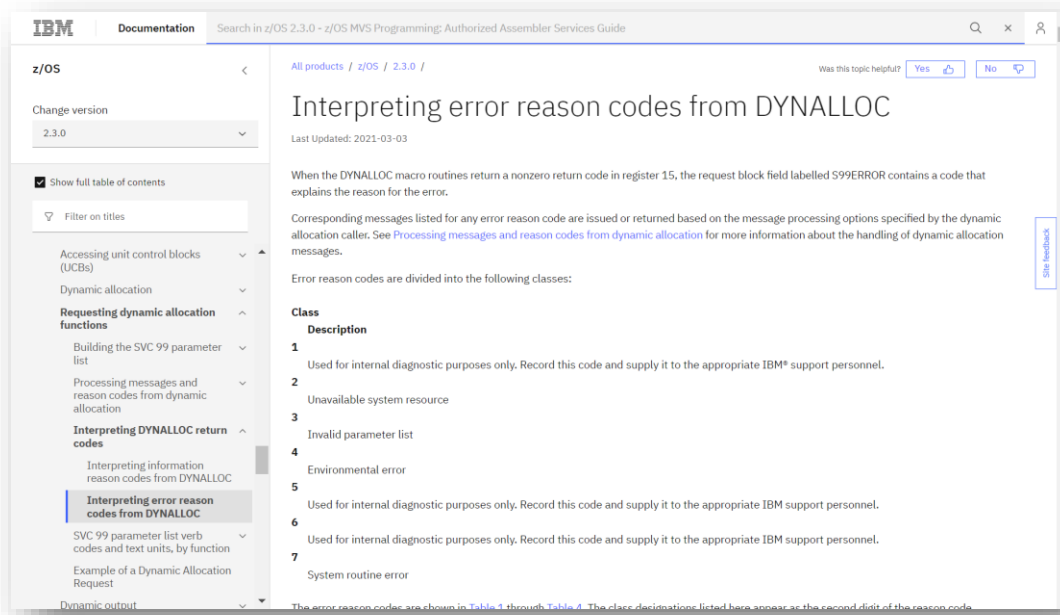
Looking up an EDC error may not provide much help like the EDC5061I shown below.



As an alternative to EDC error codes, you can look up the secondary error msg - in this case "errorCode=0x210" as highlighted above. These codes are explained in the following reference link:

https://www.ibm.com/docs/en/zos/2.3.0?topic=codes-interpreting-error-reason-from-dynalloc#erc_mjfig7

The 210 error in this case means the PDS being allocated is opened by some other process (enqueue).



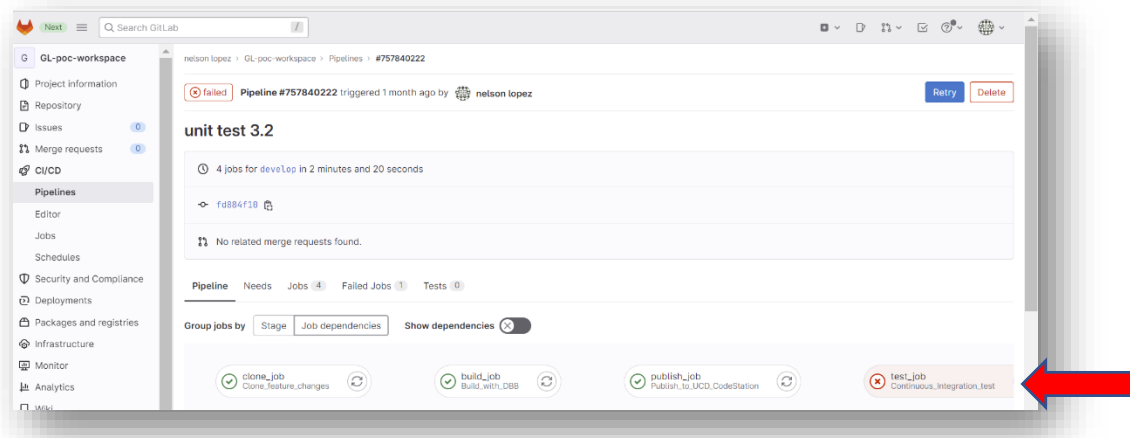
Use the above link to help resolve these types of errors. In some cases, the MVS master console may have additional error codes related to RACF or other system issues. Sometimes, enabling verbose and log4J may provide more details.

Tracing a Pipeline

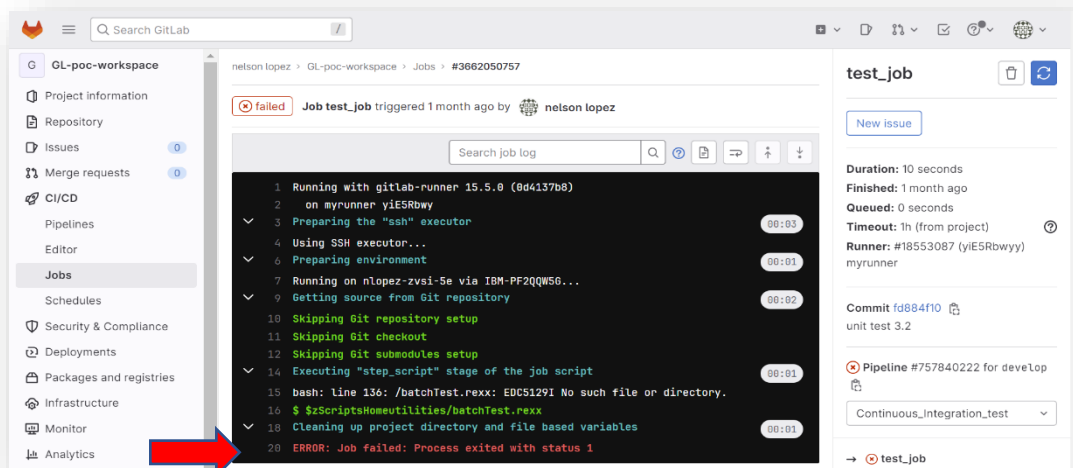
The following GitLab CI/CD pipeline illustrates how the logs of various steps can be used to trace progress and find issues. Most orchestrators highlight failed builds as shown below.



Clicking on the failed job shows the steps and pass/fail status



Clicking the failed step provides more details of the error. In this case it's a configuration issue with a testing tool.



Conclusion

This document is meant to provide general tips on troubleshooting basic DBB, dbb-zappbuild and UCD component process errors. For more difficult issues, refer to the software's reference manual or open an IBM Support Case.

Troubleshooting References

The following links may provide more information on troubleshooting DevOps Tools:

- DBB V2 - <https://www.ibm.com/support/pages/node/6415115>
- Jenkins- <https://www.jenkins.io/doc/book/troubleshooting/>
- GitLab - <https://docs.gitlab.com/ee/ci/troubleshooting.html>
- Azure DevOps - <https://learn.microsoft.com/en-us/azure/devops/pipelines/troubleshooting/troubleshooting?view=azure-devops>
- UCD CD Processes - <https://www.ibm.com/docs/en/urbancode-deploy/7.1.1?topic=agents-troubleshooting-processes>

Additional References

For more information on Z DevOps and how to receive support in the early parts of your transformation journey, visit

https://ibm.github.io/mainframe-downloads/DevOps_Acceleration_Program/devops-acceleration-program.html



DevOps Acceleration Program

IBM strongly believes that DevOps transformations are anything but plug-and-play; each enterprise brings with it its own culture, IT infrastructure, and mission. Transformation requires an absolute paradigm shift in your organization. It can be tough, we get it. Perhaps many enterprises wish to begin their transformation today but find themselves always putting it off till tomorrow.

We wanted to create a better way of bridging the gap between the intention to move towards a DevOps model and the successful transformation of such a move. That's why we've created the DevOps Acceleration Program (DAP). A value-add early adoption program designed to partner with clients during four distinct stages that we believe are necessary for any DevOps transformation:

- Value Stream Assessment
- Training
- Deployment
- Adoption

