An IBM TechDoc from

IBM Z DevOps Acceleration Program

March, 2023 - Release 1.2

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

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.

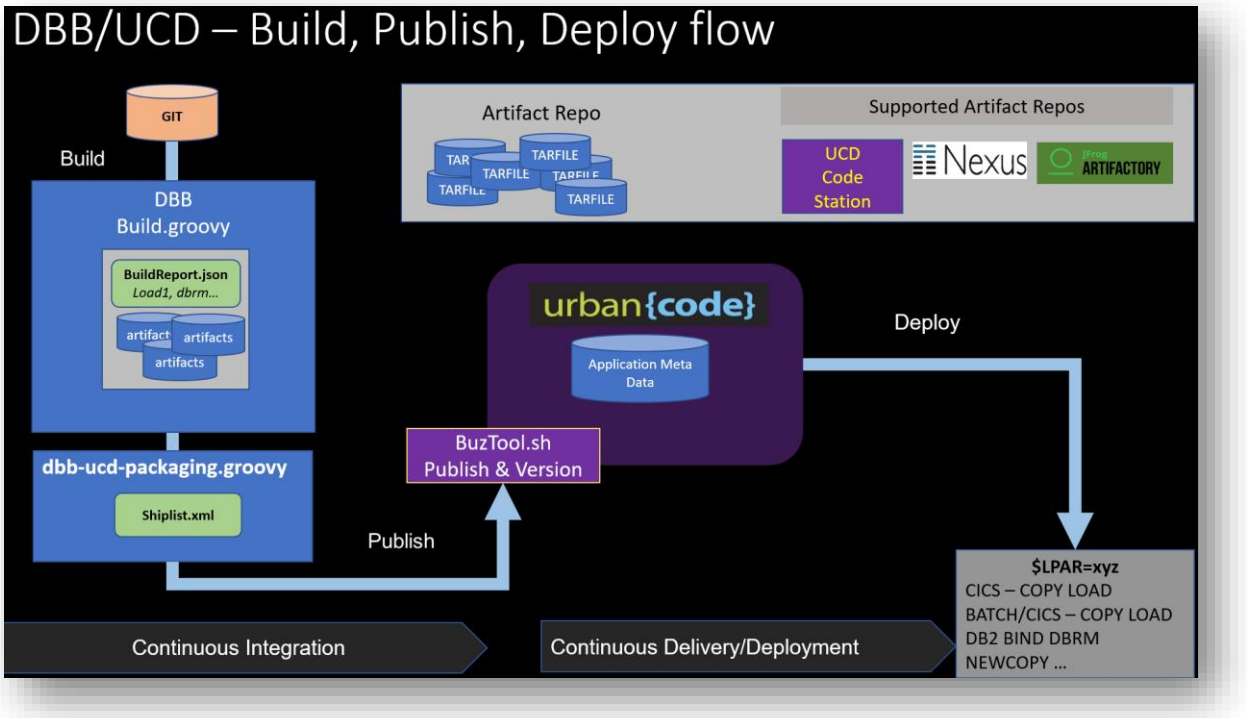
TIP: Always apply the latest version and fixes to your DevOps software stack. Refer to the component's documents for more detailed troubleshooting help.

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.

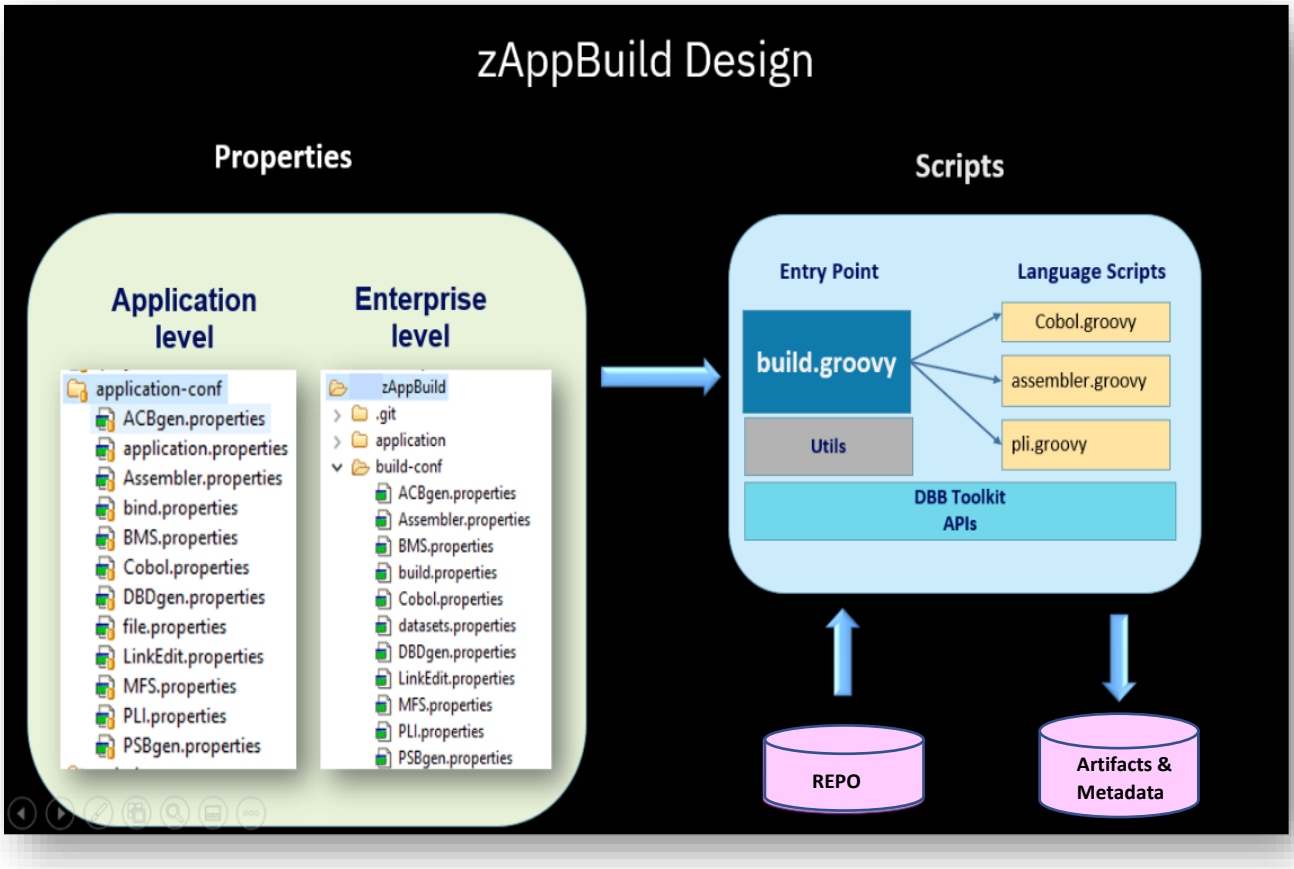


DBB Groovy Build Scripts - dbb-zappbuild

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

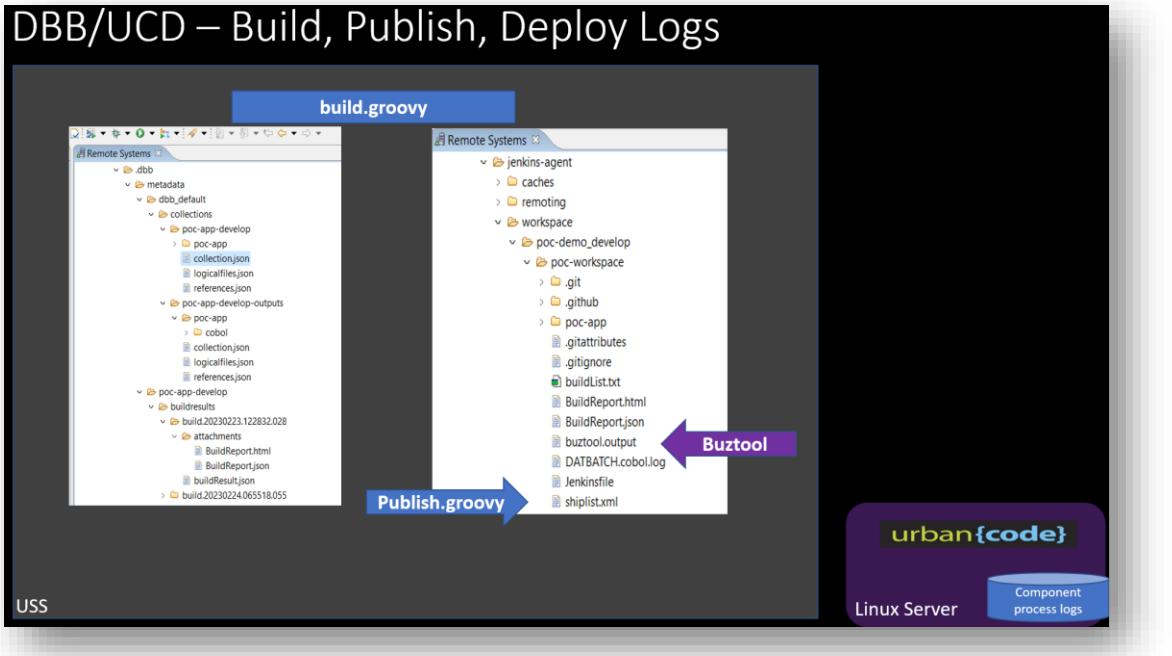
Each stage of the clone, build, publish and deploy workflow will have a log(s) to help trace and troubleshoot the process.

`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.

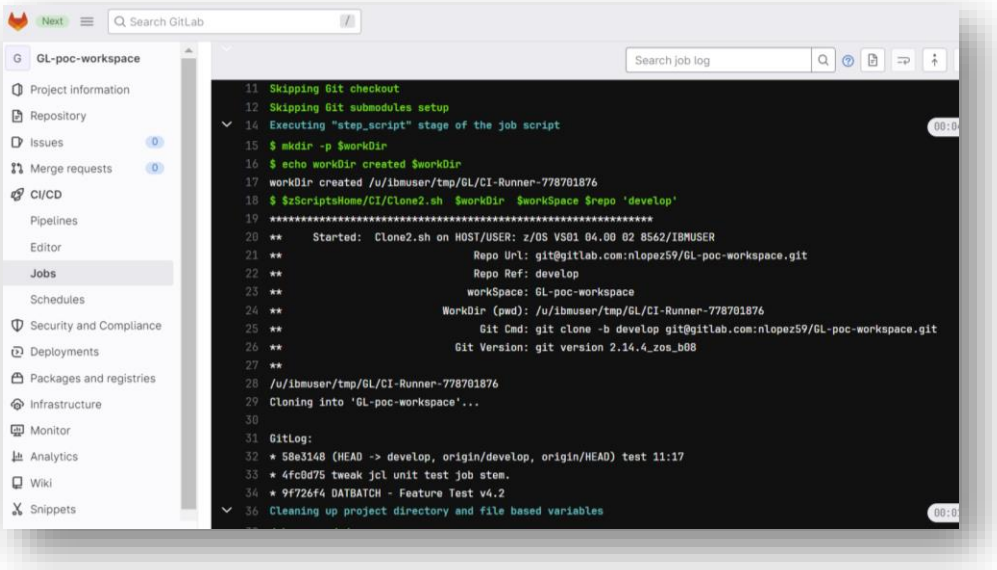


Some Common Git Errors

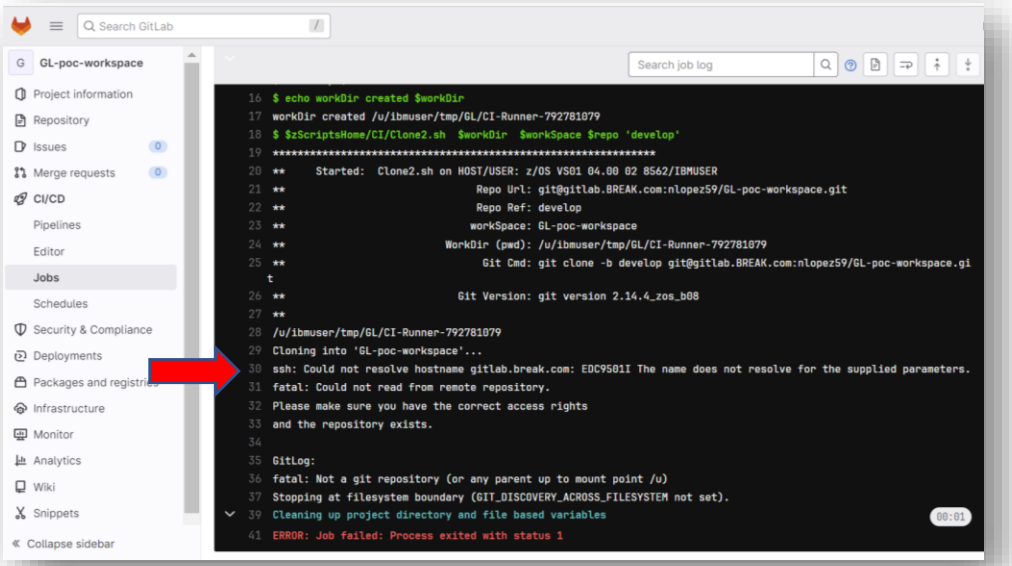
The first step in any pipeline is cloning a Git repo(s) to USS. There can be many reasons why a clone may fail. Some common reasons include:

- Security:
  - When cloning with SSH, ensure the keys are properly defined.
  - When cloning with HTTPS ensure the user/password provided is allowed access to the repo
- Network:
  - 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:
  - Ensure the target USS directory has sufficient free space and permissions to accept the clone
  - Ensure the Rocket Git environment variables are properly configured

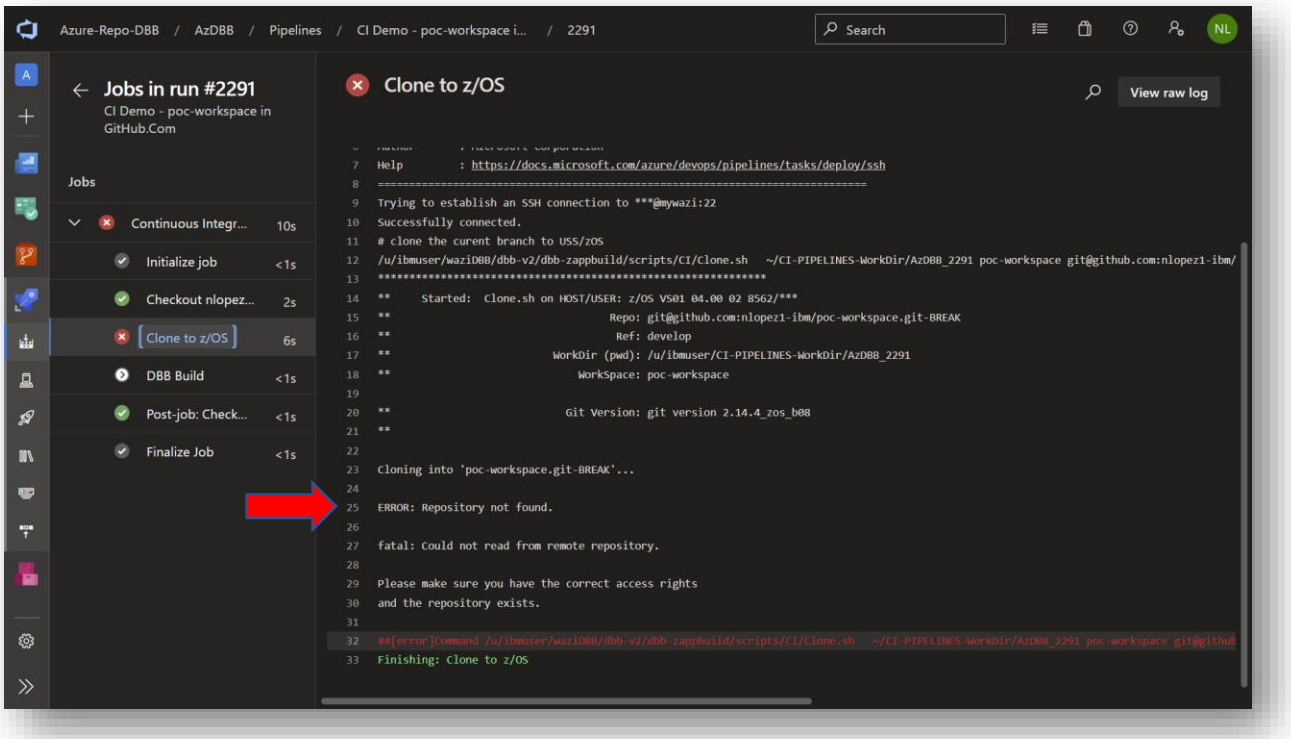
Depending on the orchestrator, the Git clone stdout/stderr will provide details on any failure. Below is an example clone in a GitLab job.



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



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



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.vmplugin.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.vmplugin.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/groovyz <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 seperated 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,--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/Groovyz build.Groovy \
                      --workspace /u/build/repos \
                      --application app1 \
                      --outDir /u/build/out \
                      --verbose  \
                      --hlq BUILD.APP1 \
                      --impactBuild
```

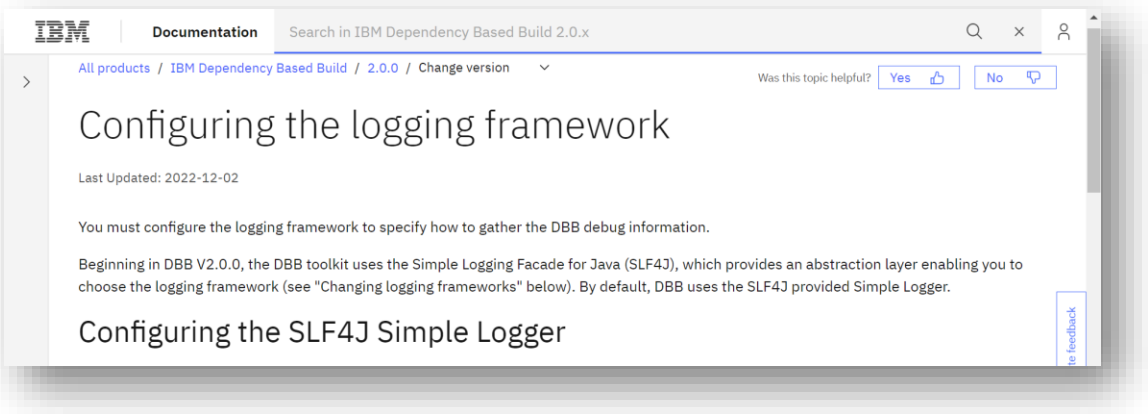
```
** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/BMS.properties
** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/Cobol.properties
** Loading property file /u/nlopez/DAT-Demo-Workspace/Mortgage-SA-DAT/application-conf/LinkEdit.properties
** 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) lrecl(133) recfm(f,b) new
```

```
*** Resolution rules for Mortgage-SA-DAT/cobol/mqsamp.cbl:
*** Physical dependencies for Mortgage-SA-DAT/cobol/mqsamp.cbl:
{"lname":"CMQMDV","library":"SYSLIB","category":"COPY","resolved":false}
{"lname":"CMQODV","library":"SYSLIB","category":"COPY","resolved":false}
{"lname":"CMQPMOV","library":"SYSLIB","category":"COPY","resolved":false}
{"lname":"CMQV","library":"SYSLIB","category":"COPY","resolved":false}
{"lname":"MQCONN","library":"SYSLIB","category":"CALL","resolved":false}
Cobol compiler parms for Mortgage-SA-DAT/cobol/mqsamp.cbl  ADATA
```

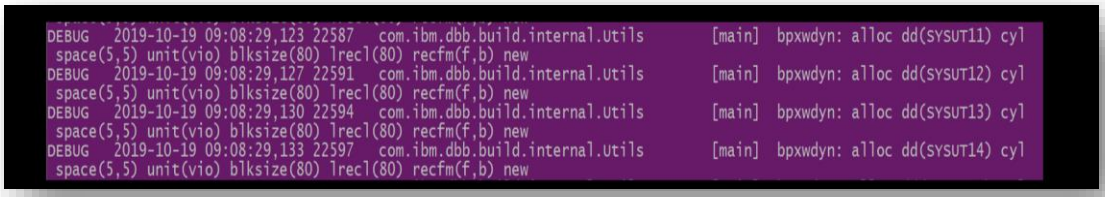
```
required props = buildOrder,buildListFileExt,languagePropertyQualifiers
** Repository client created for https://dbbdev.rtp.raleigh.ibm.com:9443/dbb/
** Build output located at /u/nlopez/DAT-Demo-Logs/build.20200630.030053.000
** Creating / verifying build dataset DAT.DEMO.ASM
** Creating / verifying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

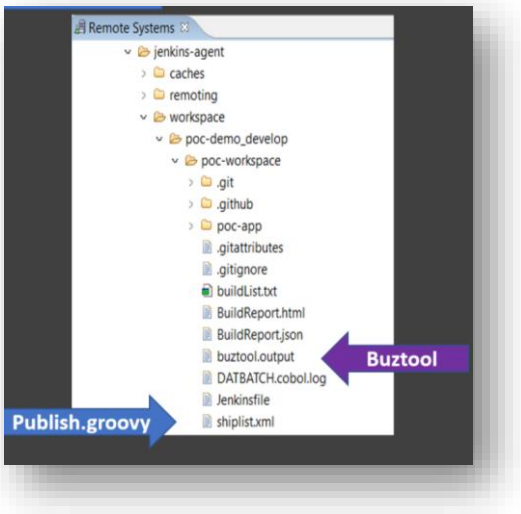


By default, this feature is disabled. When enabled it provides detailed information and error codes for troubleshooting DBB API issues as shown in the example below.

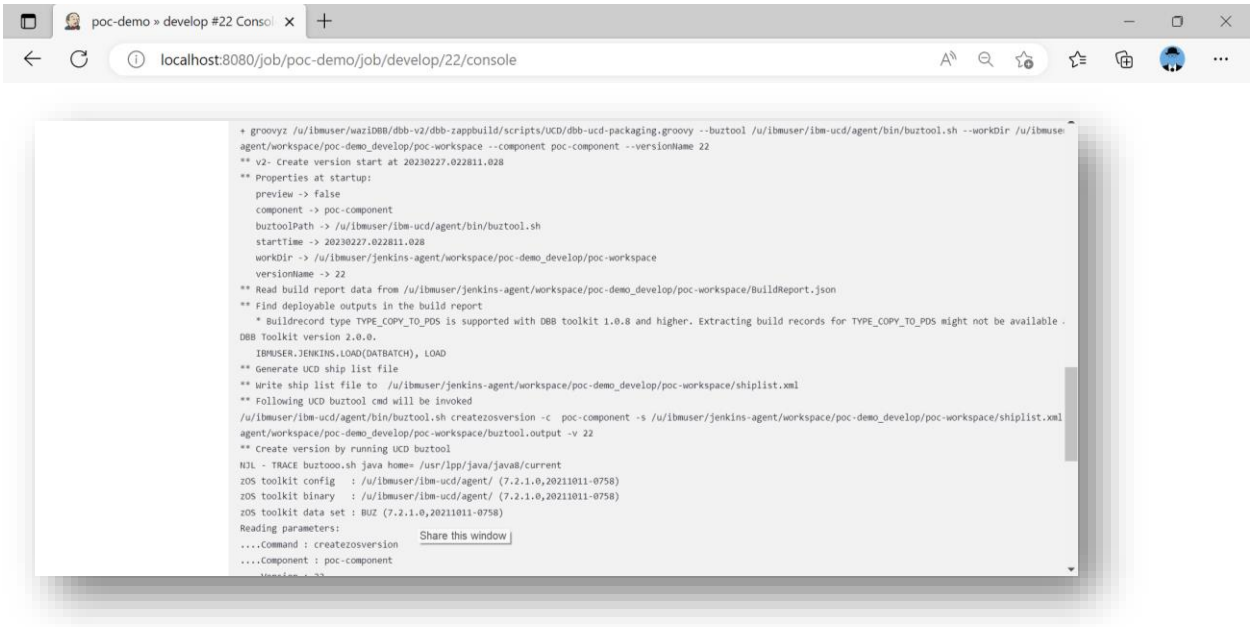


Tracing the Publish Phase

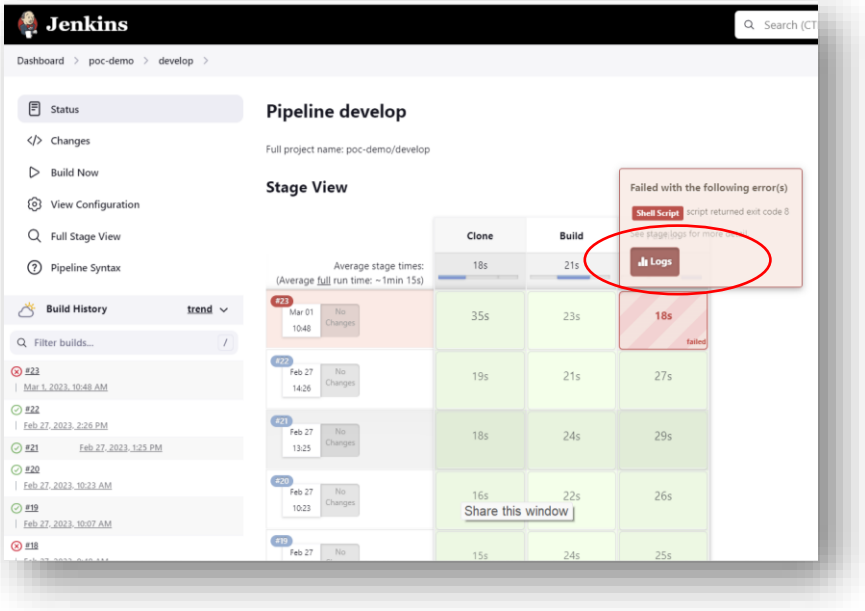
As explained in the introduction section, a pipeline can include a step to package (tar) and publish artifacts in UCD. The standard IBM DevOps workflow provides a sample script called `dbb-ucd-packaging.groovy`. It produces a log and converts the DBB `BuildReport.json` output file into a UCD shiplist. 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.



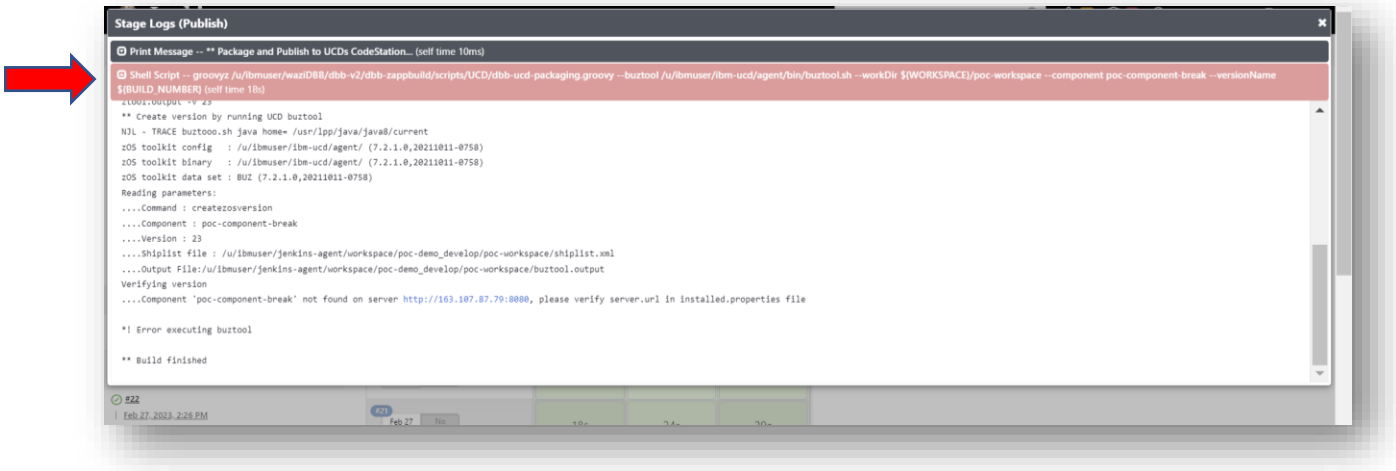
The log of an orchestrator like Jenkins will show the publish step's results.



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.

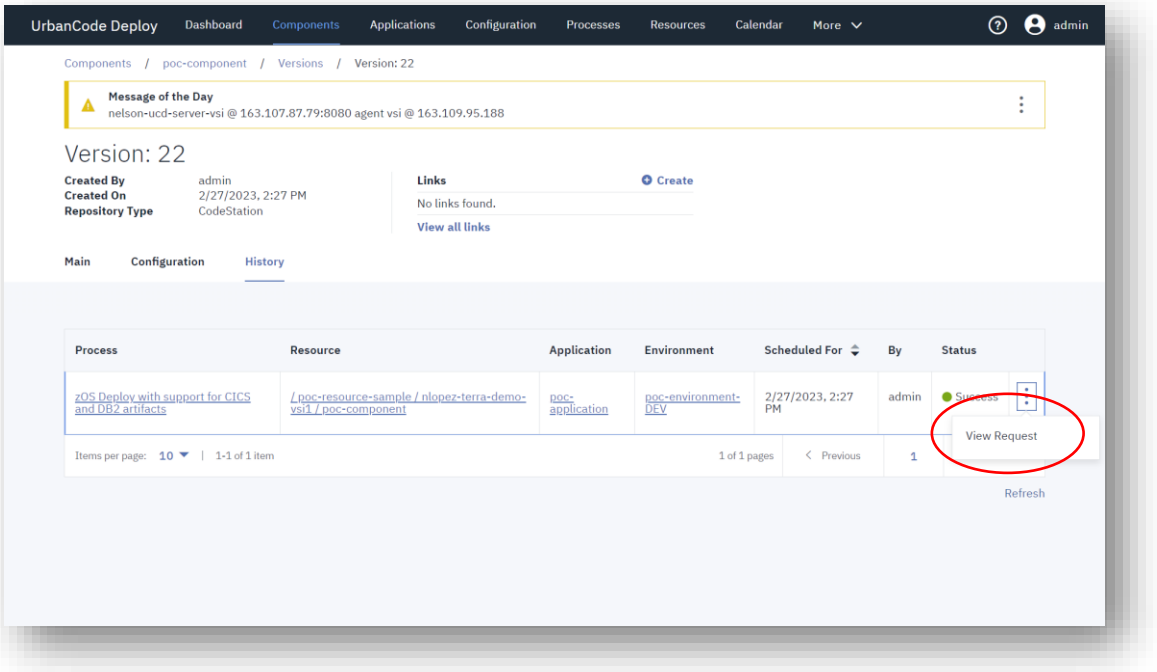


Other common issues during this phase include:

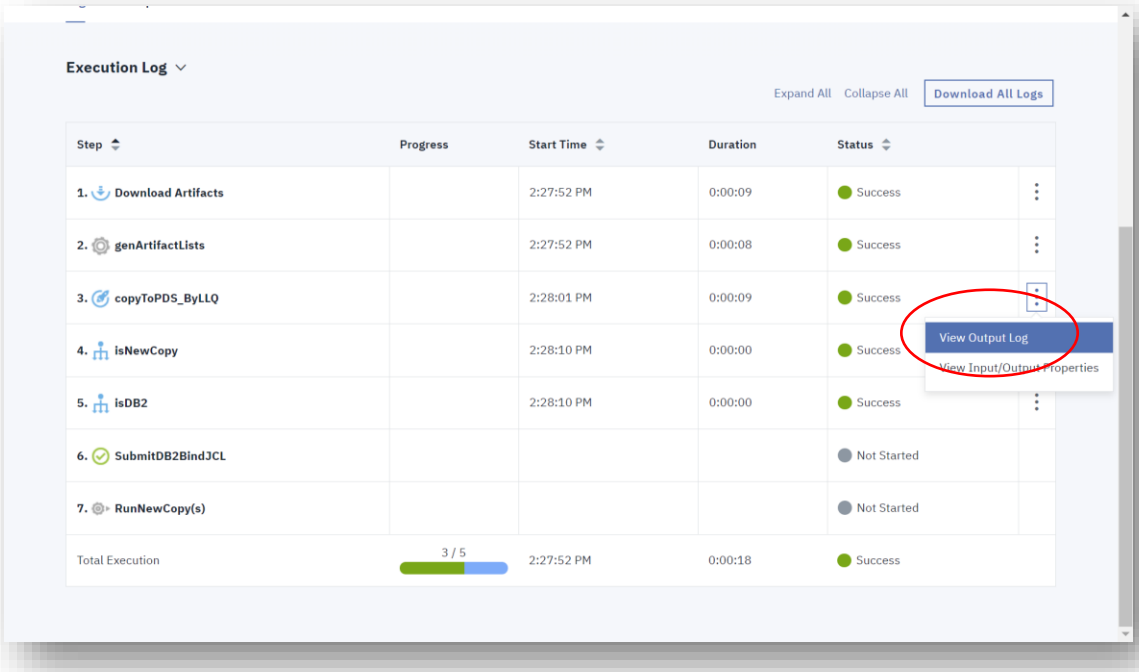
- Missing access rights to UCD, Artifactory or Nexus.
- Network failure due to firewall, proxy settings or other.
- Missing or mis-spelled resource definitions in the artifact server.
- No artifacts passed from DBB. Sometimes DBB Impact mode may not produce an output artifact. This can be related to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

UCD Application Component Process Log

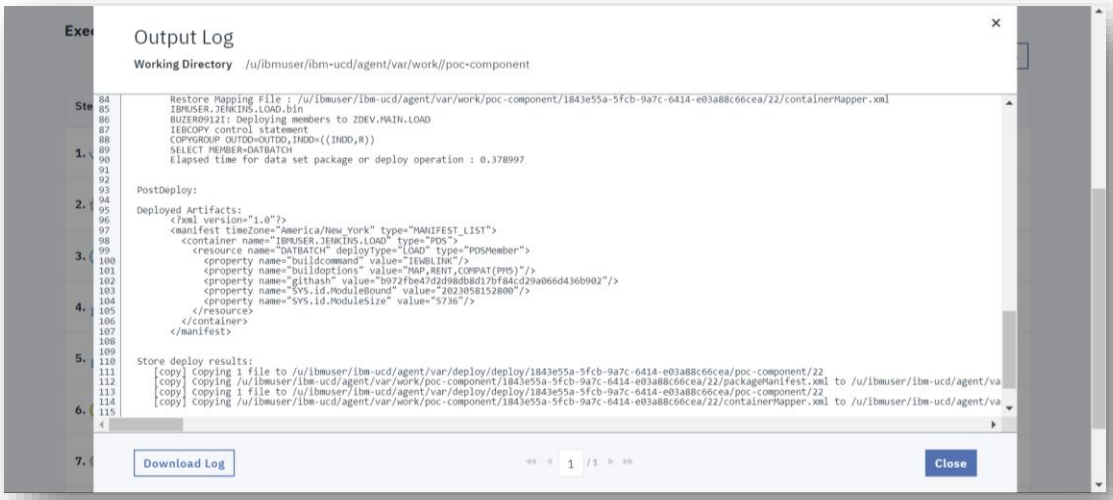
UCD process logs are accessible from the component's version history page using the 'view request' link as shown below.



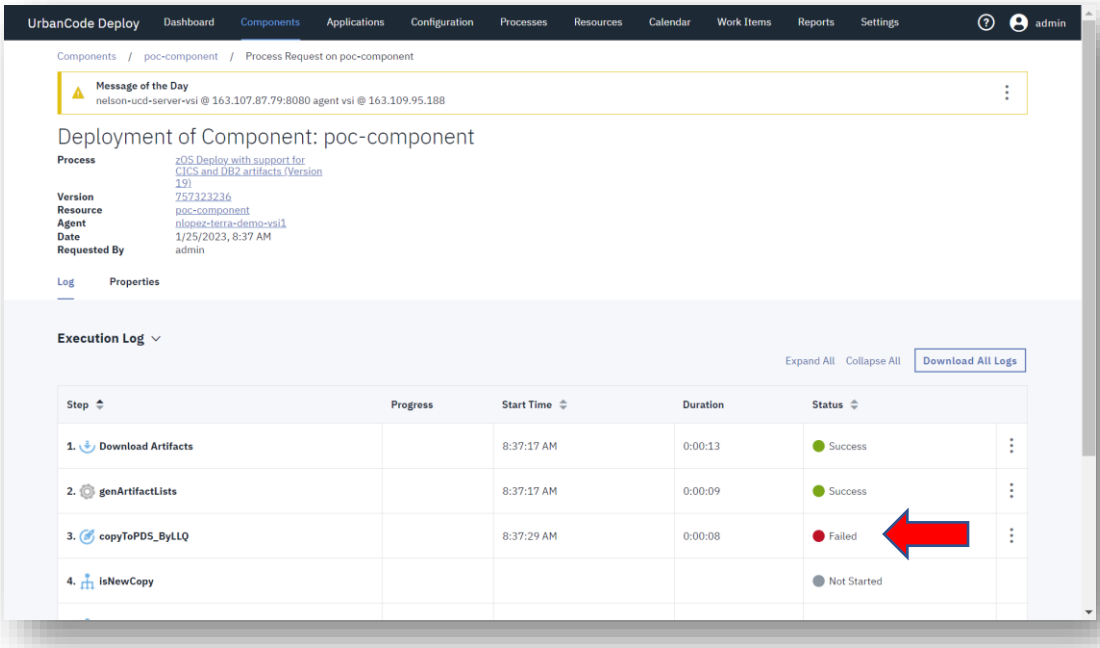
From there you can select the 'View Output Log' option to review the progress of any step.



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:

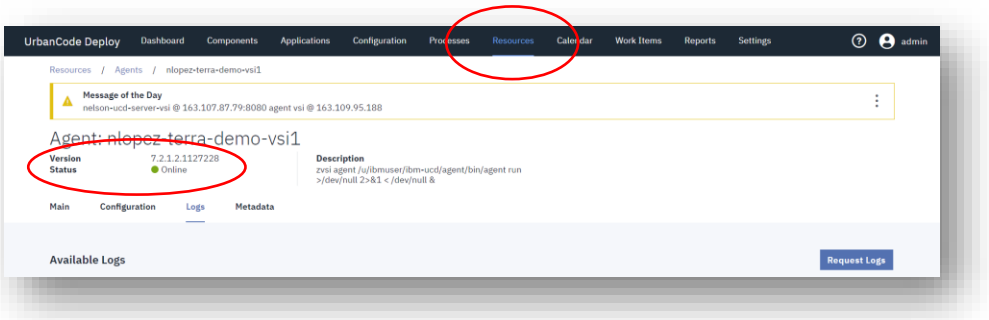


Open the step's log to view more detail. In this case, it seems there is a configuration issue. Refer to the system programmer to repair or reinstall the agent on USS.

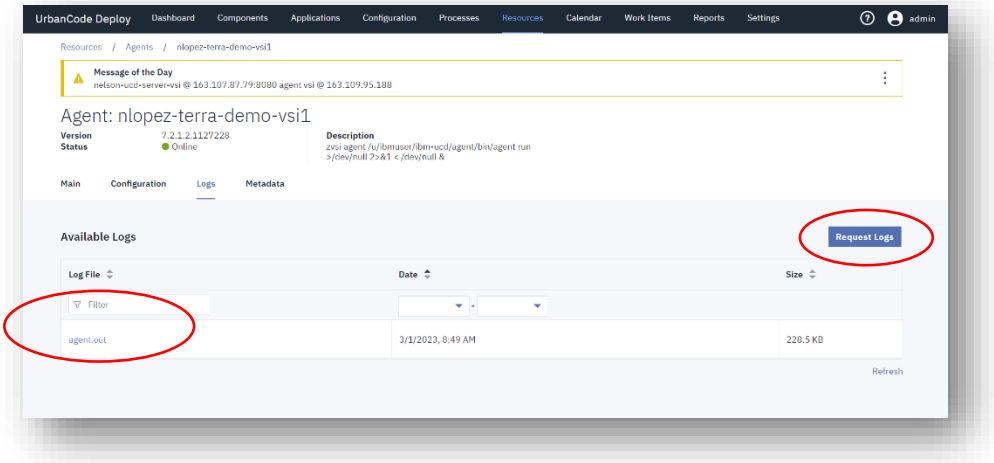
UCD Agent Log

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

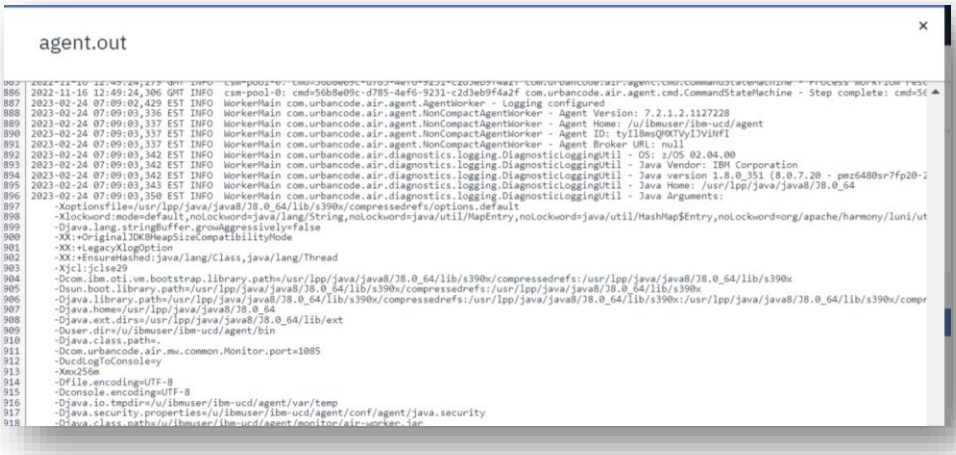
The first step in diagnosing Agent issues is to ensure the agent's "MVS Started Task" is active on the target z/OS environment – the one where artifacts will be deployed. The UCD Resource menu lists all agents. Use that to review your agent's status. "Online" (with a green dot)



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.

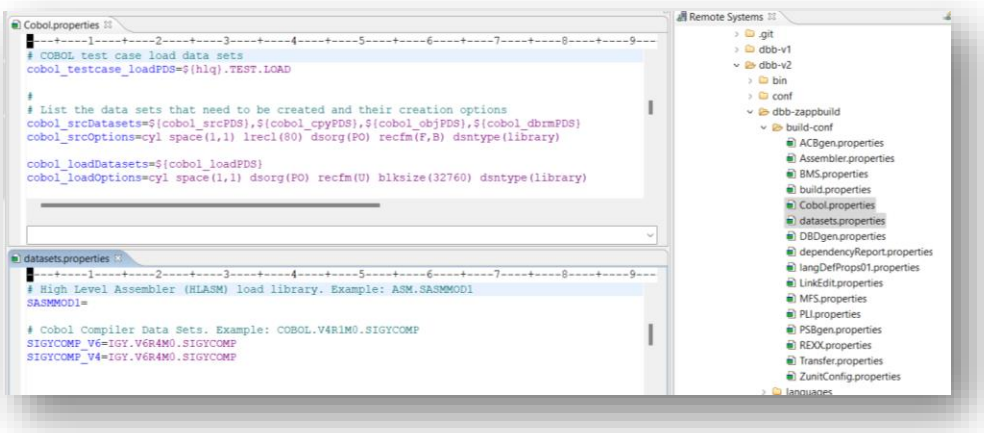


Diagnosing DBB File Allocation Errors (BPXWDYN also called DYNALLOC)

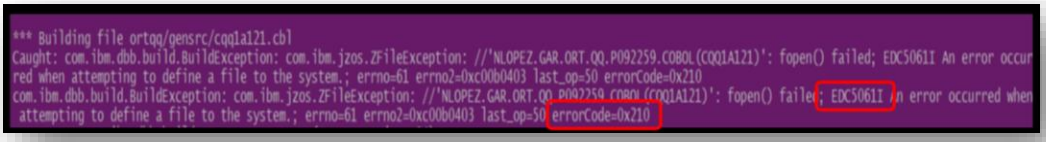
A common error when configuring DBB is with MVS file allocations. The `build.groovy` log will display any system related issues when allocating an MVS dataset.

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.

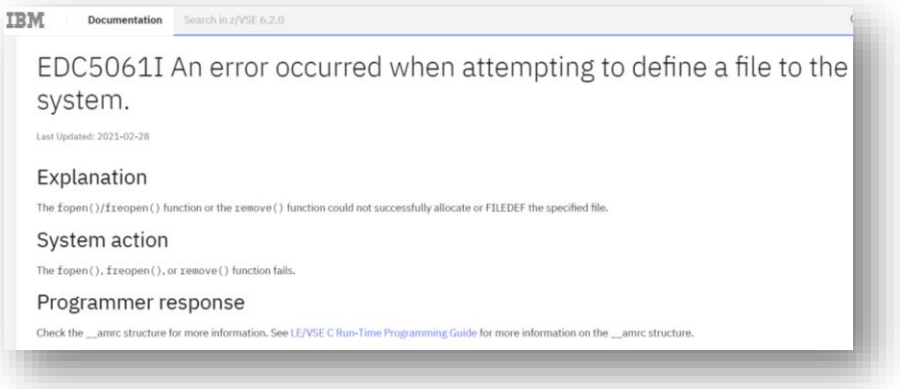
Typically, they are caused by typos in the DCBs, misspelled DSN's of existing libraries or are related to some RACF or SMS policies.



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



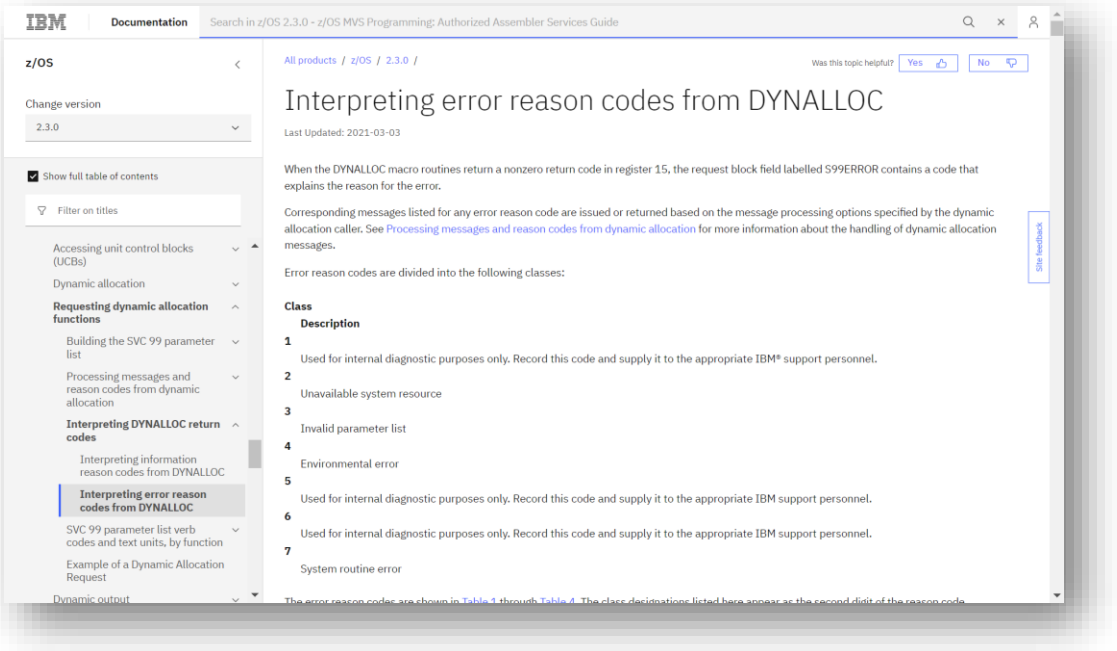
Looking up an EDC error may not provide much help like the EDC5061I shown below.



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

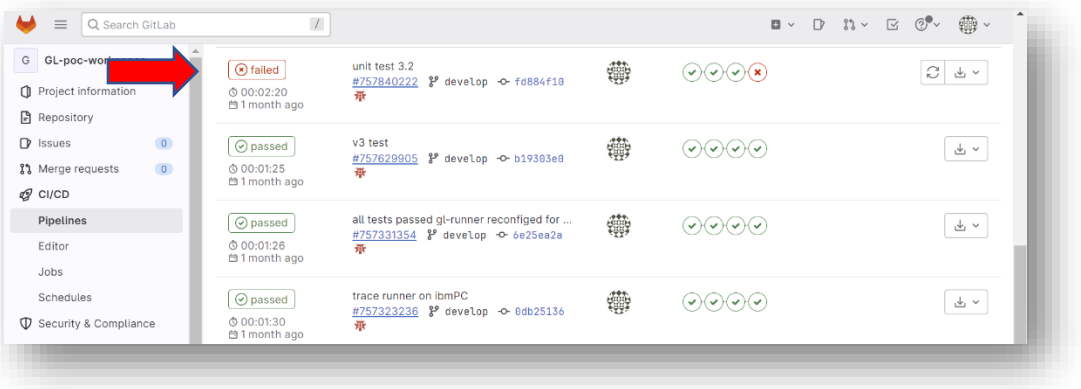
The 210 error in this case means the PDS being allocated is opened by some other process (enqueue).



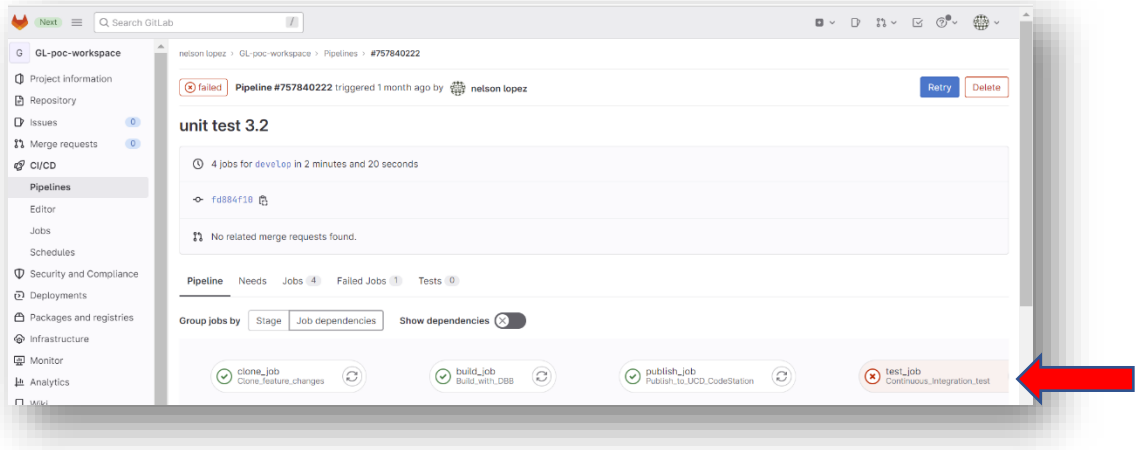
Use the above link to help resolve these types of errors. In some cases, the MVS master console may have additional error codes related to RACF or other system issues. Sometimes, enabling verbose and log4J may provide more details.

Tracing a Pipeline

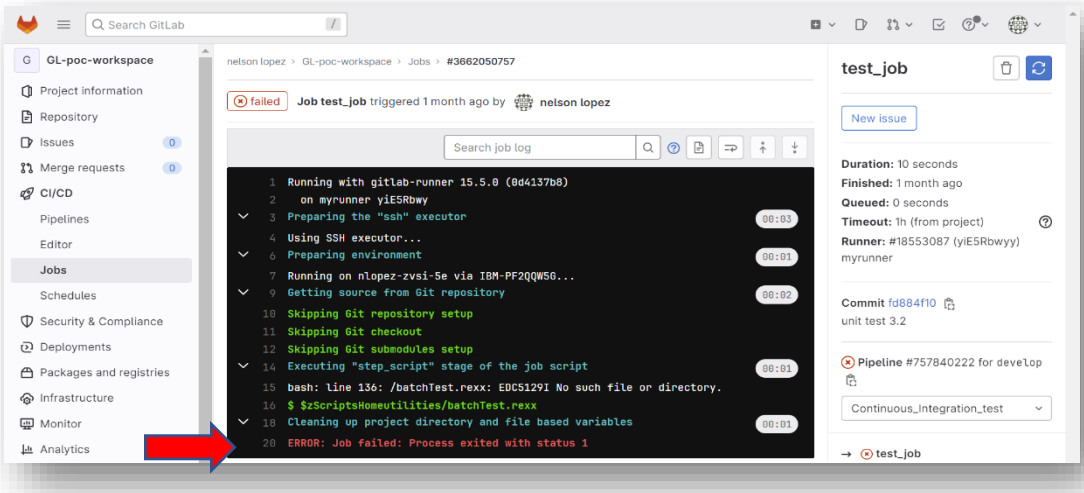
The following GitLab CI/CD pipeline illustrates how the logs of various steps can be used to trace progress and find issues. Most orchestrators highlight failed builds as shown below.



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