An IBM TechDoc from

IBM Z DevOps Acceleration Program

Azure DevOps and IBM Dependency Based Build Integration

Nelson Lopez
Shabbir Moledina
Timothy Donnelly

Abstract

This is a primer on Azure DevOps and DBB based Pipelines running on z/OS

IBM

Table of Contents

Overview

The purpose of this document is to help you get started with building Azure DevOps Pipelines that run Git based IBM Dependency Based Builds (DBB) on z/OS.
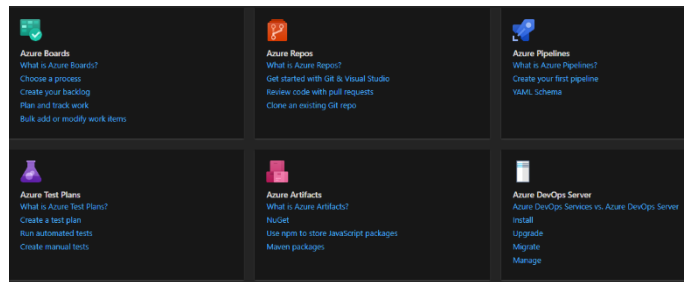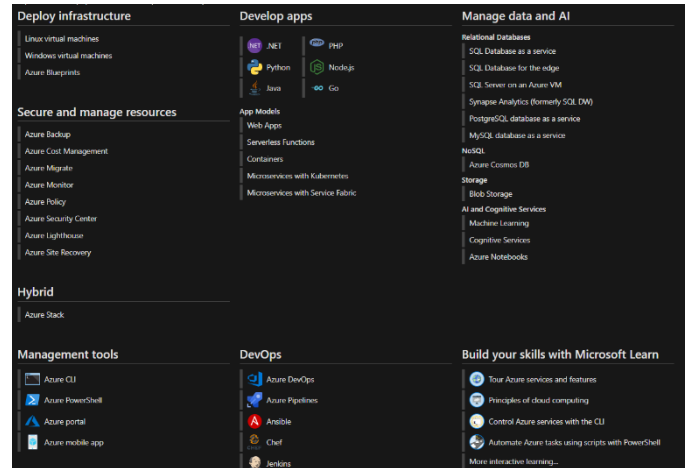
Microsoft Azure

Microsoft Azure is a platform of interoperable cloud computing services, including open-source, standards-based technologies and proprietary solutions from Microsoft and other companies. It is an alternative option to building an on-premise server installation or leasing physical servers from traditional data centers. One of these cloud computing services is Azure DevOps.

Azure DevOps[1]

Azure DevOps provides developer services to support teams in order to plan work, collaborate on code development, and build and deploy applications. Developers can work in the cloud using Azure DevOps Services or on-premises using Azure DevOps Server. Azure DevOps Server was formerly known as Visual Studio Team Foundation Server (TFS).

Azure DevOps provides integrated features that you can access through your web browser or IDE client. You can use one or more of the following services based on your business needs:

- **Azure Repos** provide Git repositories or Team Foundation Version Control (TFVC) for source control of your code
- **Azure Pipelines** provide build and release services to support continuous integration and delivery of your apps
- **Azure Boards** deliver a suite of Agile tools to support planning and tracking of work, code defects, and issues using Kanban and Scrum methods
- **Azure Test Plans** provide several tools to test your apps, including manual/exploratory testing and continuous testing
- **Azure Artifacts** allow teams to share Maven, npm, and NuGet packages from public and private sources and integrate package sharing into your CI/CD pipelines

---

[1] **Overview** https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops
   Azure Communications https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser#communication
   Security https://docs.microsoft.com/en-us/azure/devops/organizations/security/about-security-identity?view=azure-devops

Azure DevOps on z/OS

While Azure's DevOps is primarily used by distributed applications, customers are looking for consolidated solutions that integrate with modern IBM Mainframe development workflows.   However, there are some limitations. For example, Azure Testing and Artifacts are currently not integrated with Mainframe processes. But Azure Boards, which are independent of platform, can be used to manage Mainframe-based sprints.

A key value of Azure DevOps is the ability to run CI pipelines to build and release Mainframe applications that run in batch or online with or without Db2 and MQ. By adding IBM's Urban Code Deploy (UCD) to your Azure pipeline, you gain a full CI/CD pipeline for your on-prem, cloud or hybrid platform that includes z/OS applications.

Azure DevOps & z/OS Integrated Architecture

This diagram illustrates a typical developer workflow using Azure's DevOps pipeline with DBB and UCD on zOS. Each step (A – E) represents the sequence of the flow.   Starting from (A), a developer clones a repo as a local copy onto their desktop. Using IDz or any Git capable IDE, they can clone and branch to begin servicing their work item. Each commit and push (B) triggers a pre-defined Azure pipeline job to the build process (C-E) on z/OS.
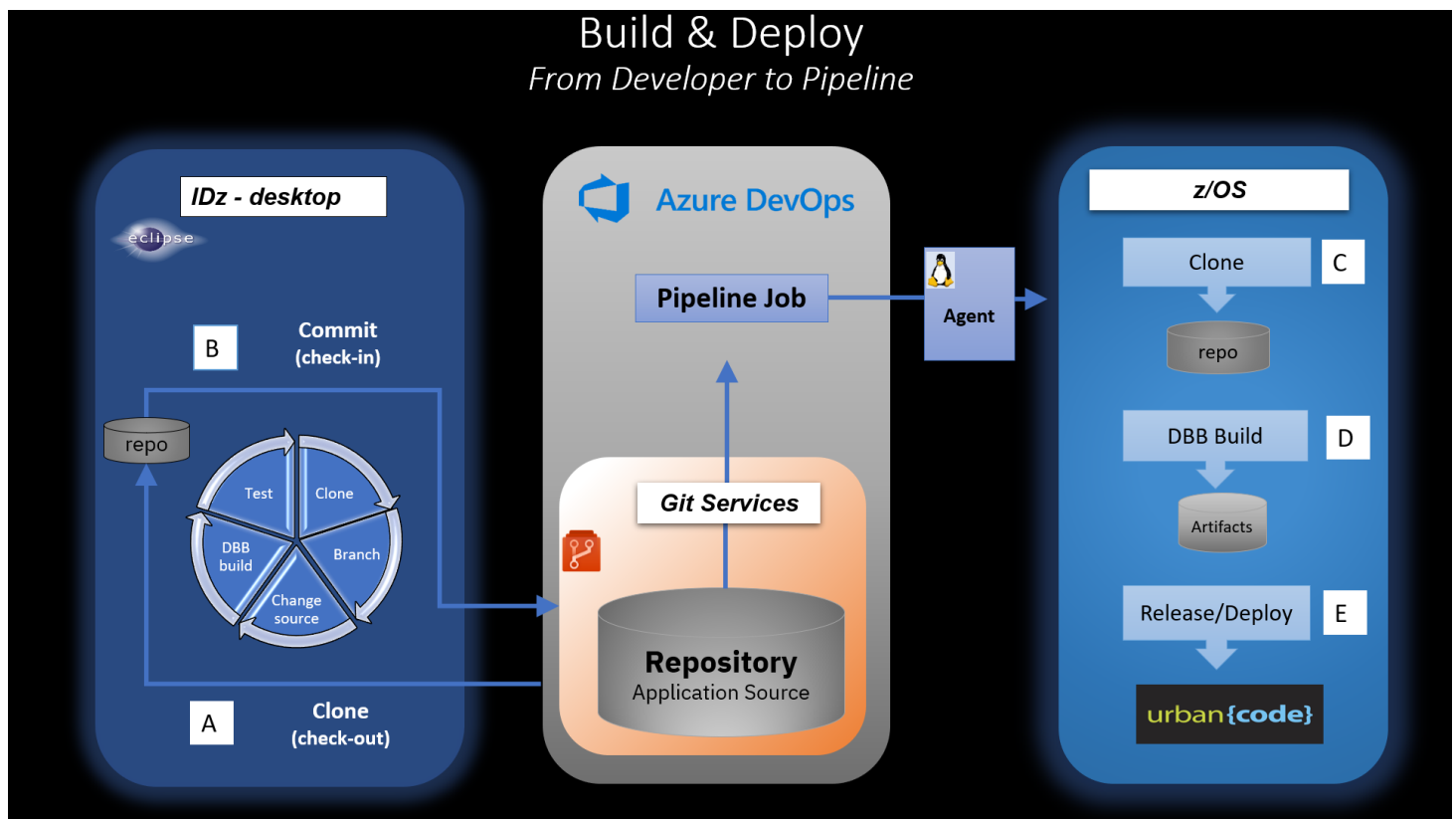
Getting Started

This section explains how to create the above workflow.

<u>Prerequisites:</u>

- **Access** [2]
    - o Your organization has a Corporate Account with Microsoft. This enables Single-Sign-On (Active Directory) based security and network connectivity between Azure and your network.
- **Azure Agents**
    - o Can be installed on an on-prem Windows (your laptop) or Linux system
    - o Can access the Azure DevOps network   https://dev.azure.com/
    - o Can SSH into a Mainframe running a pre-configured DBB Toolkit and Rocket Git Client environment
    - o The Agent machine must have a Git Client.
- **IDE**
    - o These notes show how to access Azure's Repo services from IBM's eclipse based IDE tool called IBM Developer for z/OS systems (IDz). However, you can configure any IDE with any Azure supported Git provider.
- **Repo**
    - o You'll need a sample repo for testing, learning and designing your workflow.

Azure DevOps is structured as a set of Organizations within a company. Each of these can have one or more Projects, Team and Repos.   For more details see "plan-your-azure-devops-org-structure".
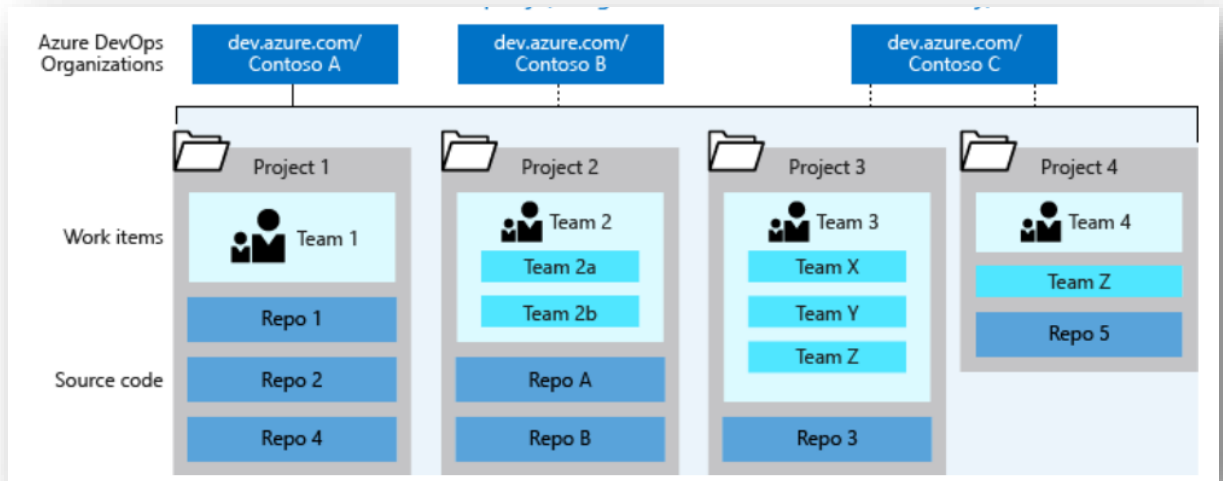
Define an Organization and an Azure Agent[3]

If your company has an Azure platform then work with your Admin to learn what privileges you will need in order to create an organization.   The examples in these notes use Microsoft's free Azure DevOps services with IBM's account with Microsoft.

1) Sign into Azure DevOps (https://azure.microsoft.com/en-us/services/devops/) and follow the "Start Free" path.
2) Generate an Azure Personal Access Token (PAT):
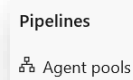   a) From "User Setting" on the home page select "Personal Access Token".
    b) Generate and save your PAT with a scope of "Full Access".
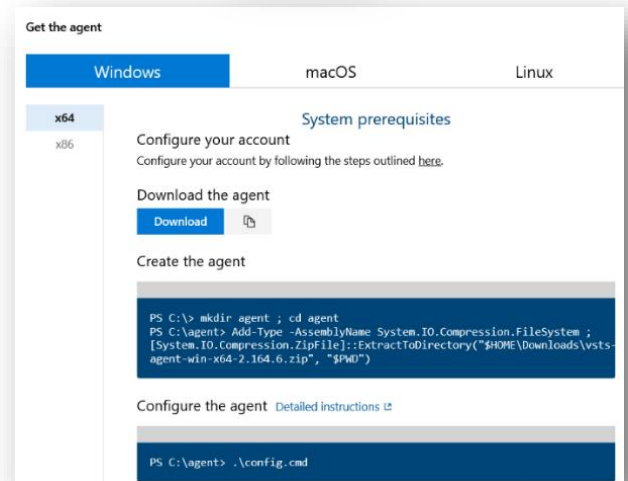3) From the home page select "New organization" and provide a name.
4) Create an Agent-Pool:
    a) From your Organization page select "Organization Settings".
    b) From "Agent Pools" select "Add Pool" and create a "self-hosted" pool.
    c) Name your pool, review the defaults, and "Create".
5) Get the Agent:
    a) Return to "Agent Pools" and select your new pool and then "New Agent".
    b) Click on the Download link for the agent type you want to install. This example uses a "**Windows"** agent on an x64 laptop. Linux or MacOS instructions may vary.
    c) From the laptop open a PowerShell (PS) terminal and paste the commands shown in the "Get the agent" dialogue under the under "create the agent". See a sample install on the next page.
6) Configure the agent:
    a) Enter the PS cmd **".\config.cmd"**.
    b) When prompted paste the URL of your new Organization. This example uses https://dev.azure.com/**Azure-Repo-DBB/**
    c) Enter "PAT" for authentication type.
    d) Paste your PAT from step 2
    e) Enter your new pool name from step 4. In this example it is called My-Agentv2.
    f) Press Enter to accept the default "agent name". In this case it is the laptop's name.
    g) Press Enter to accept the default "work folder".
    h) Enter "N" for "run the agent as a service".
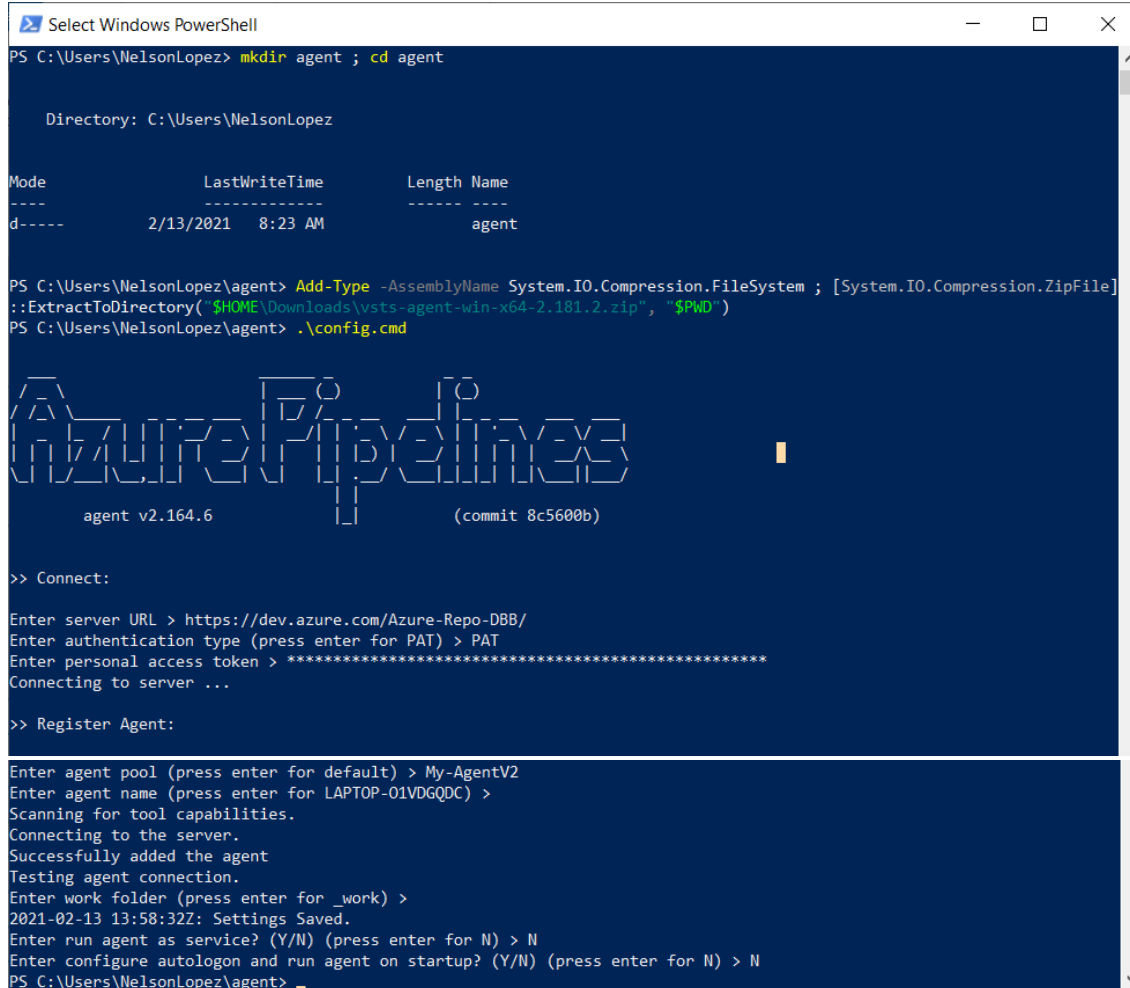   i) Enter "N for "autologon" prompts.

---

[3] About Agents   - https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser

7) Start your Agent:
   a) Enter the PS cmd **".\run.cmd"** to start your agent.
   b) Check firewall and network access with your admin.
   c) The agent's status is available from the "Agent Pool" page under the Agents tab.

```
PS C:\Users\NelsonLopez\agent> .\run.cmd
Scanning for tool capabilities.
Connecting to the server.
2021-02-13 14:02:00Z: Listening for Jobs
```

Example Windows Azure agent install and configuration.

```
Select Windows PowerShell                                          —   □   ×

PS C:\Users\NelsonLopez> mkdir agent ; cd agent


    Directory: C:\Users\NelsonLopez


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        2/13/2021   8:23 AM                agent


PS C:\Users\NelsonLopez\agent> Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compression.ZipFile]
::ExtractToDirectory("$HOME\Downloads\vsts-agent-win-x64-2.181.2.zip", "$PWD")
PS C:\Users\NelsonLopez\agent> .\config.cmd


  ___                      ___ _           _ _
 / _ \                    | _ (_)         | (_)
/ /_\ _____ _ _ _ ___   |  _/ _ _ __   ___| |_ _ __   ___  ___
|  _  |_  / | | | '__/ _ \  | | | | '_ \ / _ \ | | '_ \ / _ \/ __|
| | | |/ /| |_| | | |  __/  |_| |_| |_) |  __/ | | | | |  __/\__ \
\_| |_/___|\__,_|_|  \___|      |_| .__/ \___|_|_|_| |_|\___||___/
                                  | |
        agent v2.164.6            |_|        (commit 8c5600b)


>> Connect:

Enter server URL > https://dev.azure.com/Azure-Repo-DBB/
Enter authentication type (press enter for PAT) > PAT
Enter personal access token > *******************************************************
Connecting to server ...

>> Register Agent:

Enter agent pool (press enter for default) > My-AgentV2
Enter agent name (press enter for LAPTOP-01VDGQDC) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2021-02-13 13:58:32Z: Settings Saved.
Enter run agent as service? (Y/N) (press enter for N) > N
Enter configure autologon and run agent on startup? (Y/N) (press enter for N) > N
PS C:\Users\NelsonLopez\agent>
```
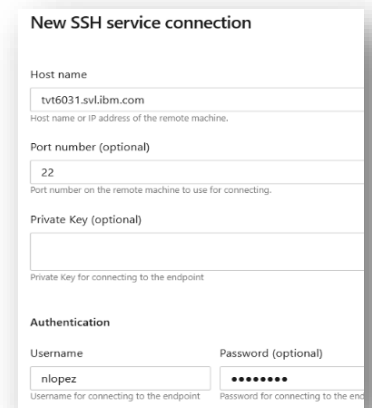
Define an Azure Project and a Mainframe Service Connection (End Point)

1. From your Organization select "New Project". Provide a name and press "Create".
2. From your project select "project settings" ⚙️ Project settings
3. Select "Service connections" and "Create service connection".
4. Select "SSH" connection type and click "Next".
5. Fill in all the SSH connection details to access your Mainframe (see sample on the right). Enter the Mainframe host name and port (usually 22). Provide a Mainframe (RACF/ACF2) username and password and click "Save". Optionally you can use a private SSH key. Scroll to the bottom to name your connection.

Migrate a Mainframe Application to Azure Repo

These steps are normally performed by a DevOps Admin to initialize an Azure project for a new team. Security and other settings are out of scope for these notes.

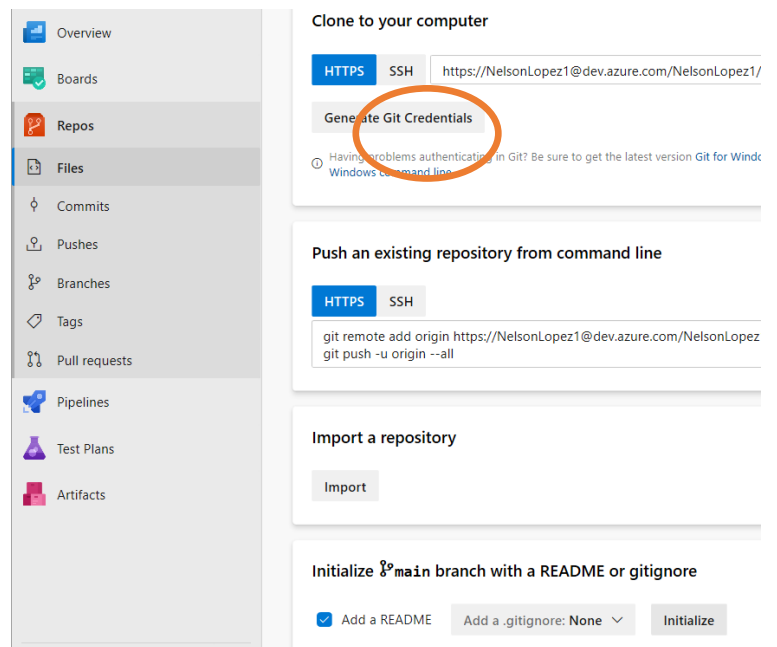### Step 1 – Admin initializes a new repo

From your new Project page select Repos and the "Initialize" option. Your repo is created with the name of your project with a "main" branch.

From your new Repo's page select "Clone", "SSH" and "Manage SSH Keys". This takes you to the "New Key" page where you can add your z/OS SSH public key.

Once added, go back to the repo's clone button to cut & paste the SSH based URL for cloning on USS. For example, a project named "Azure-Mortgage-SA"' would look like

git@ssh.dev.azure.com:v3/Azure-Repo-DBB/AzDBB/

On USS run "**git clone**" followed by the URL. Your new local repo is ready for the next step – PDS migration.

Note: Cloning with HTTPS is also supported on USS. From the repo's clone button with HTPPS enabled select "Generate Git Credentials". Use that text to replace the username portion of the HTTPS URL. Use this new URL in USS to clone the repo.

Step 2 - DBB Migration Tool

Some terms used in this section:

- **Local repo** – a repo that was cloned from Azure.
- **Remote repo** – the original repo on Azure.
- **workDir** – a path on the USS filesystem where files (such as logs) related to a build are stored.
- **workSpace** – a term used by DBB to define the local repo's root path.
- **$DBB_HOME** – the environment variable defining the installation path of the DBB Toolkit on USS

After cloning, a DevOps Admin runs DBB's migration utility to copy members from production PDS(es) into the local repo. Once added they are pushed back to the remote repo. Typically, PDSes are designed to store specific types of source code like Cobol source, Copybooks or JCL. Each type is copied into its own subdirectory – called a targetDir. The name of a targetDir is freeform but should reflect the type of files it contains. Member names are converted to lower case and given an extension like .cbl for Cobol or .cpy for copybooks.

The migrate tool resides in **$DBB_HOME/migration/bin/migrate.sh** and requires these arguments:[4]

- **-r** a local **r**epo path. In the example below, "**$HOME/Azure-Mortgage-SA/Mortgage-App/**" is a local repo with 2 parts:
  - "poc-workspace" is the local repo's root folder.
  - "poc-app" is the application folder of all source files and is created by the tool.
- **-m m**apping rule:
  - **hlq:** the PDS minus it's LLQ. For example, 'TST.ACCTS.COBOL' would be "hlq:TST.ACCTS"
  - **targetDir[5]:** a directory created by the tool under the application folder ("poc-app") in this example.
  - **extension:** added to each member. Standard defaults are **cbl**=cobol, **bms**=cics-maps, **pli**, **mac,** and **cpy**=copybooks
  - **pdsMapping:** and **toLower:** enter as shown
- **LLQ** is the last argument and the LLQ of the PDS. It can include a member or a pattern. If a member is not provided then all of them are copied.

Example 1, migrate Cobol members 'ABC*' from PDS 'TST.ACCTS.COBOL'. Using the command below (all one line) from your home directory, each member is copied to the local **r**epo path (**-r**) into the **targetDir** 'cobol' with an extension of **'.cbl'**. Highlighted text indicates values to be review or changed.

```
$DBB_HOME/migration/bin/migrate.sh  -r $HOME/ Azure-Mortgage-SA/Mortgage-App/  -m
MappingRule[hlq:TST.ACCTS,pdsMapping:false,toLower:true,targetDir:cobol,extension:cbl]    "COBOL(ABC*)"
```

Example 2, migrating copybook members 'AT*' from 'PROD.COMMON.COPY' to targetDir 'copybook' with an extension of '.cpy'

```
$DBB_HOME/migration/bin/migrate.sh  -r HOME/ Azure-Mortgage-SA/Mortgage-App/  -m
MappingRule[hlq:PROD.COMMON,pdsMapping:false,toLower:true,targetDir:copybook,extension:cpy]    "COPY(AT*)"
```

Note: Review the migrate tool's stdout for potential round-trip chars. Refer to the Knowledge Center doc for more details.

---

[4] For a complete reference see https://www.ibm.com/support/knowledgecenter/SS6T76_1.0.9/migration.html
[5] For SYSLIB folders like copybook, plinc and maclib refer to zAppBuild's "application-conf/application.properties" file for default naming rules like "$copybookRule"

Step 3 – Admin finalizes and pushes to Git

Note: Some commands below assume dbb-zappbuild is installed under the user's home (~) dir. Change accordingly based on your setup/environment.

Add a .gitattributes file to the local repo:
- From USS, cd into your local repo and copy the sample .gitattributes files from your dbb-zappbuild directory.
- **cp ~/dbb-zappbuild/.gitattributes .**

Add a sample application-conf file to the application folder:
- **cd** into your repo's application folder like "poc-app" in the above example and
- **cp -r ~/dbb-zappbuild/samples/application-conf/   ./application-conf**
- **rm .gitattributes**

Push your local repo back to origin
- **cd**
- **cd into your local repo folder**
- **git add .**
- **git commit –m 'DBB Migrated Production copies of app xyz'**
- **git push**

Initialize the application's DBB Meta-Data (ScanOnly Build)
- Review the application's 'mainBuildBranch' value in it's application-conf/app-properties file. Ensure the value equals the name of your production branch like 'main'.
- After a successful push you can delete this local repo on USS.
- After creating your pipeline as described in the following sections, create a DBB scanOnly job for this new main branch to initialize the DBB metadata.
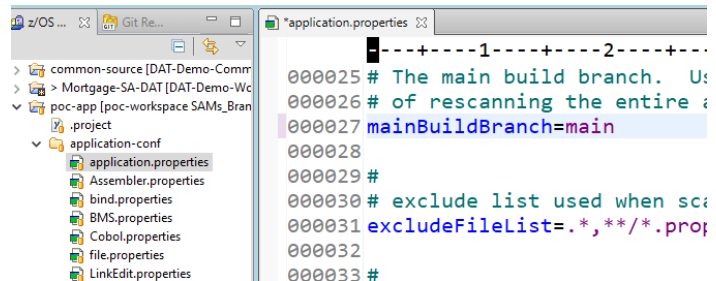
A team lead can be added to the project with privileges to create the 'develop' branch from main and manage the project's resources.
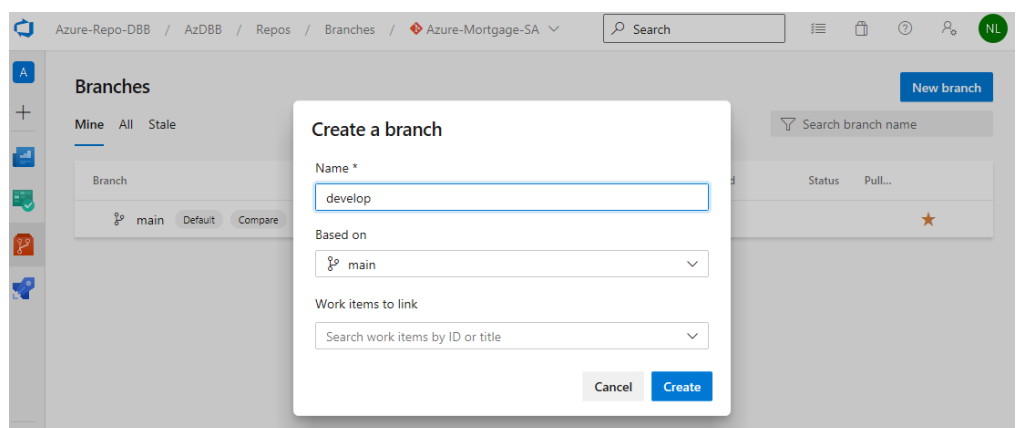
Optionally set 'develop' as the default branch to avoid accidentally working on the wrong branch. Also review/set other defaults like security for both, 'main' and 'develop'. Ideally 'main' should be read-write for Admins and Service Accounts and read-only to the rest. The Team Lead can decide who can update the develop branch as well as other settings such as merge approvals and peer review rules.
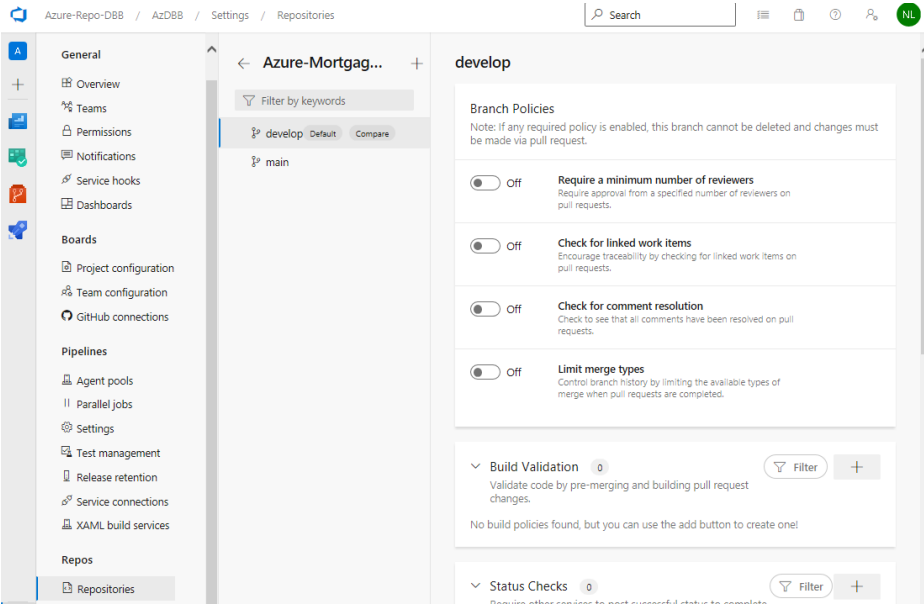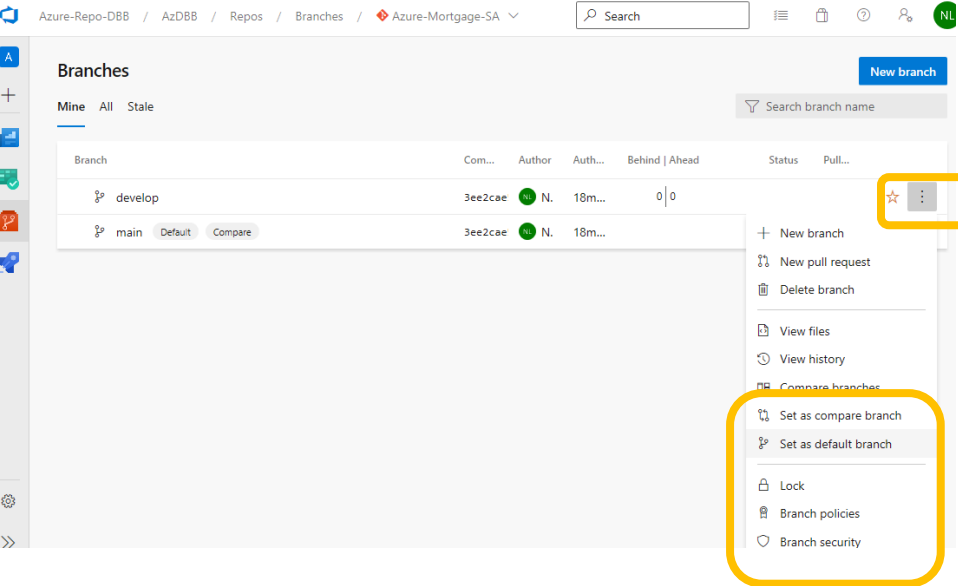
Create a CI Pipeline (Basic Setup)[6]

1. From your Azure project page, select the pipeline   icon and click "Create Pipeline".
2. From "Where is your code" select **"Use the classic editor"** (at the bottom of the page).
3. Under "Select a source", pick your repo server.
4. Use the dropdown arrow of the "Repository" field to pick the newly migrated z/OS application repo. Use the '**develop**' branch as the default and press "Continue".
5. On the next page, select the 'Empty job' template.
6. Click "Agent job 1" to open its properties.
7. Select your Agent from the "Agent pool" dropdown.
8. Press the "+" sign next to "Agent job 1" to open the "Add tasks" frame and search for SSH and click "Add".
9. Click on the new SSH task and select your Mainframe service connection from the dropdown property.
10. In the "Commands" window enter "ls".
11. Click "Save & queue" (at the top).
12. On the next page "Run pipeline" ensuring your Agent pool is selected and click "Save and Run" to start your job.

See this reference video for the above https://youtu.be/hvlFG-D6Z9c

---

Create a DBB/UCD Azure Pipeline (Advanced Setup)

This section outlines how to setup a DBB/UCD Azure pipeline and provides sample 'getting started' scripts.

Assumptions:
- You have completed the "Basic Setup" pipeline above and successfully tested it with an "ls" command on z/OS
- You can clone a repo with the Rocket Git Client and run a DBB groovy build from the USS command line
- The Azure agent can SSH into the z/OS Mainframe
- Urban Code Deploy is configured on the same Mainframe as where DBB is

A basic DBB pipeline consists of three tasks and scripts:
- Clone to USS
- DBB Build
- Publish DBB artifacts to UCD

Define Azure Variables

Azure variables simplify and standardize pipelines across projects. This example suggests the following variables:
- **MyApp** - DBB's main application source folder
- **MyRepo** - the application repo to be cloned as an SSH URL. Azure provides a repo URL system variable but only in HTTPS format.
- **MyWorkDir** - a directory where all build output will be stored. This path should have a mount point with enough free space to support your build output needs and will require some log retention processes. This example shows how to use two Azure system variables to create a unique name. For example, `$HOME/Azure-WorkSpace/$(System.TeamProject)_$(Build.buildid)`
  - o $buildid - is automatically assigned at build time with a unique number and used as the UCD version in the published task
  - o $teamProject - is used as a UCD Component name for improved traceability.
- **MyWorkSpace** - DBB main working area. This is the last part of the repo URL (a.k.a the repo name)

The snapshots show USS folders after a clone and how the variables are mapped to folders.
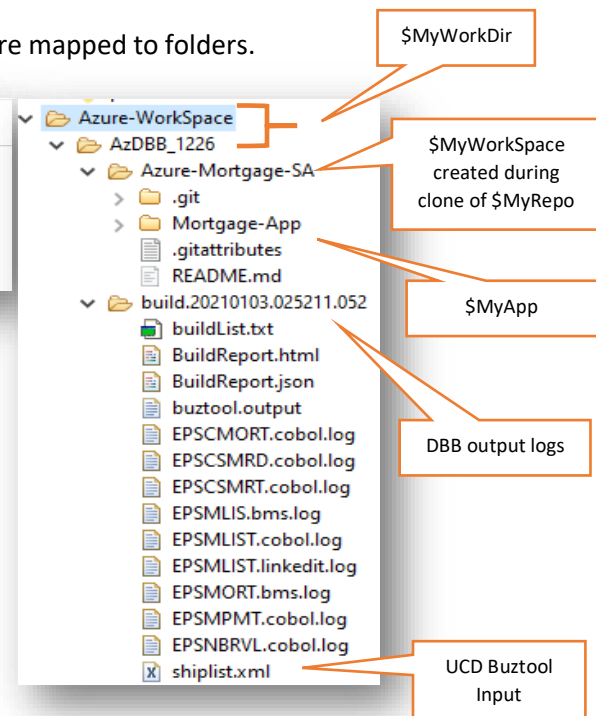
Define Task 1 - Git Clone

Cut and paste the content of this and subsequent sample scripts into USS files under dbb-zappbuild or a folder of your choice. Ensure the shell scripts are executable.

Create a new job like was done in the "Basic Setup" section above.   Define the agent and add the new tasks. The first task will clone your repo on USS. Use this in the 'commands' field of the job:

dbb-zappbuild/AzRocketGit-init.sh $(MyRepo) $(MyWorkDir) $(Build.SourceBranch)

Note the usage and order of the Azure variables passed to the script. Also note how the workSpace folder is created by the clone command within the script.

This script/task can be invoked multiple times within
a job to clone additional repo(s). For example, when you need to include common source like copybooks stored in an enterprise-wide repo.
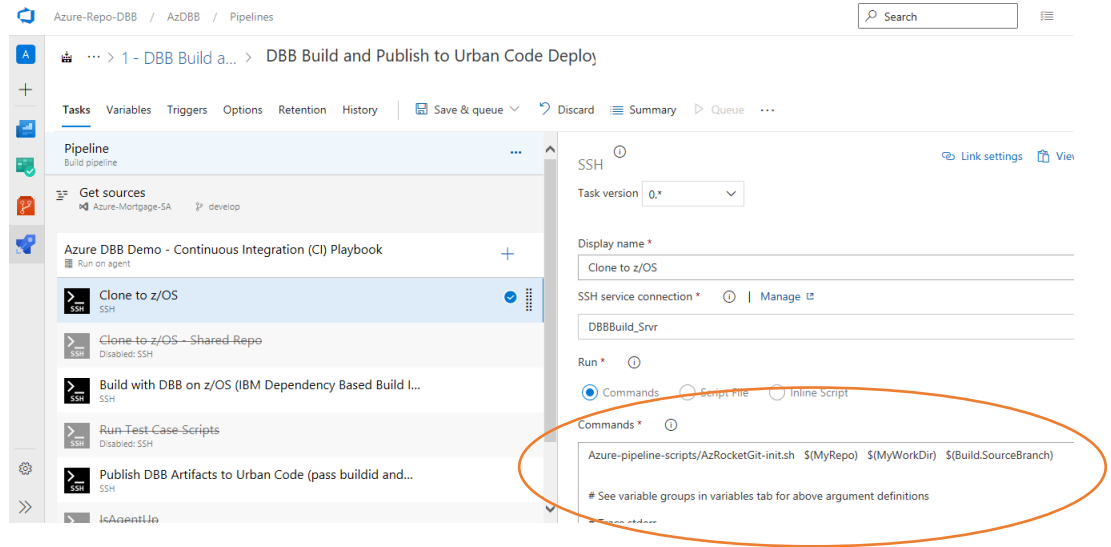
Script: AzRocketGit-init.sh

```sh
#! /bin/sh
## Az/DBB Demo- Git clone v1.2 (njl)
. $HOME/.profile
MyRepo=$1
MyWorkDir=$2 ; mkdir -p $MyWorkDir  ; cd $MyWorkDir
branch=$3
# Strip the prefix of the branch name for cloning
branch=${branch##*"refs/heads/"}

workSpace=$(basename $MyRepo) ;workSpace=${workSpace%.*}
echo "**********************************************************"
echo "**     Started:  Rocket-Git Clone on HOST/USER: $(uname -Ia)/$USER"
echo "**                              MyRepo:" $MyRepo
echo "**                           MyWorkDir:" $PWD
echo "**                           workSpace:" $workSpace
echo "**                              branch:" $3  "->" $branch
git clone -b $branch $MyRepo  2>&1
cd  $workSpace

echo "Show status"
git status

echo "Show all Refs"
git show-ref
```

Define Task 2 - DBB Build

Right click Task 1 and select "clone task(s)" from the popup menu to create a new SSH task and use this command:

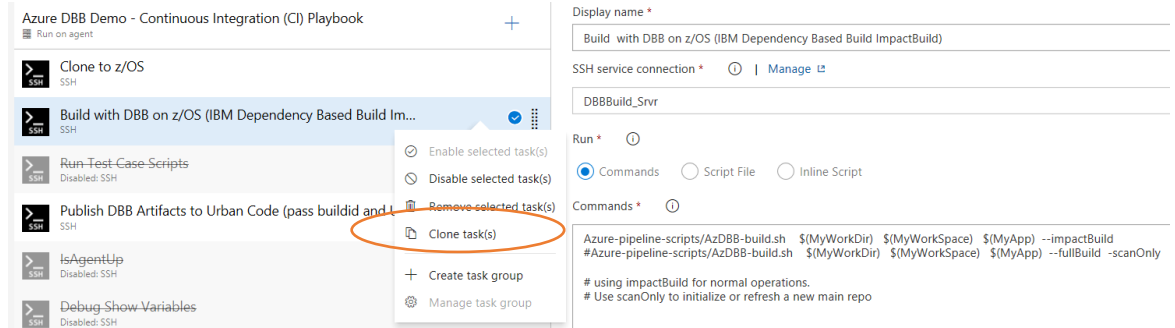**dbb-zappbuild/AzDBB-build.sh $(MyWorkDir) $(MyWorkSpace) $(MyApp) --impactBuild**

The last argument controls DBB's build mode. "impactBuild" should be the default mode. For more on DBB builds see

https://github.com/IBM/dbb-zappbuild

Build mode "--fullBuild --scanOnly" is used to initialize or refresh a DBB metadata branch without building artifacts.

Script: AzDBB-build.sh

```
#! /bin/sh
## Az/DBB Demo- DBB Build v1.2 (njl)
. $HOME/.profile
MyWorkDir=$1 ; cd $MyWorkDir
MyWorkSpace=$2
MyApp=$3
BuildMode="$4 $5" #DBB Build modes:  --impactBuild,  --reset, --fullBuild, '--fullbuild --scanOnly'
zAppBuild=$HOME/dbb-zappbuild/build.groovy
echo "************************************************************"
echo "**     Started:  DBB Build on HOST/USER: $(uname -Ia)/$USER"
echo "**                       MyWorkDir:" $PWD
echo "**                     MyWorkspace:" $MyWorkSpace
echo "**                           MyApp:" $MyApp
echo "**                  DBB Build Mode:" $BuildMode
echo "**                  zAppBuild Path:" $zAppBuild
echo "**                        DBB_HOME:" $DBB_HOME
echo "** "
echo " ** Git Status  for MyWorkSpace:"
git -C $MyWorkSpace status
groovyz $zAppBuild  --workspace $MyWorkSpace --application $MyApp  -outDir . --hlq $USER  --logEncoding UTF-8  $BuildMode
if [ "$?" -ne "0" ]; then
  echo "DBB Build Error. Check the build log for details"
  exit 12
fi

## Except for the reset mode, check for "nothing to build" condition and throw an error to stop pipeline
if [ "$BuildMode" != "--reset" ]; then
        buildlistsize=$(wc -c < $(find . -name buildList.txt))
        if [ $buildlistsize = 0 ]; then
        echo "*** Build Error:  No source changes detected.   RC=12"
        exit 12
        fi
else
        echo "*** DBB Reset completed"
fi
exit 0
```

Define Task 3 - Publish to UCD

Create a new task with this command:

> **dbb-zappbuild/AzPublishUCD.sh $(Build.buildid) $(System.TeamProject) $(MyWorkDir)**

This script starts the UCD buztool process to publish artifacts created from the DBB Build task. The command passes the Azure BuildID and TeamProject which are used in UCD as the component name and its new version number respectively. It also calls "**dbb-ucd-packaging.groovy**" (sample provided below) to read DBB's output BuildReport.json and create a shiplist.xml used by buztool.
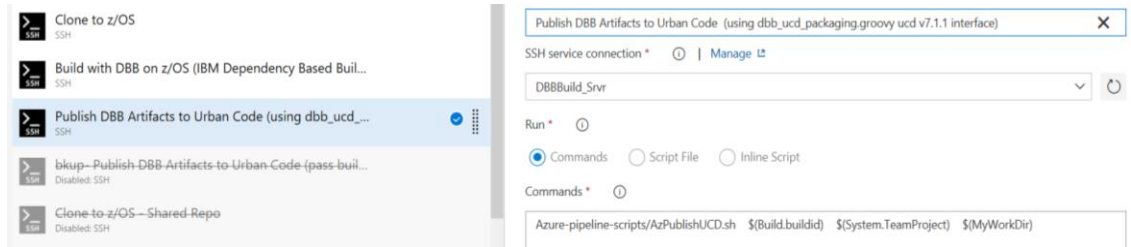
For more on buztool and UCD see https://www.ibm.com/support/knowledgecenter/SS4GSP_7.1.1/com.ibm.udeploy.doc/topics/zos_runtools_uss.html
Note that this sample publishes to UCD's Code Station binary repository. Refer to the link above for publishing to Nexus or Artifactory.

Review and update the UCD agent path (in yellow below) installed on your USS filesystem.

Script: AzPublishUCD.sh

```
#! /bin/sh
## GitLab/Azure/Jenkins DBB Demo- UCD Buztool v1.3 (njl)

. $HOME/.profile
ucd_version=$1
ucd_Component_Name=$2
MyWorkDir=$3
artifacts=$(ls -d $MyWorkDir/build*)

echo "************************************************************"
echo "**     Started:  UCD Publish on HOST/USER: $(uname -Ia) $USER"
echo "**                                 Version:" $ucd_version
echo "**                               Component:" $ucd_Component_Name
echo "**                                 workDir:" $MyWorkDir
echo "**                      DBB Artifact Path:" $artifacts
buzTool=$HOME/ucd/Tass-agent/bin/buztool.sh
groovyz $HOME/dbb-zappbuild/dbb-ucd-packaging.groovy -b $buzTool -w $artifacts -c $ucd_Component_Name -v $ucd_version
```

Add this sample groovy script to the same location as the above scripts (dbb-zappbuild). This script is compatible with UCD Ver 7.1.1.

The latest release of this script is located at https://github.com/IBM/dbb/tree/master/Pipeline/CreateUCDComponentVersion

Groovy script: dbb-ucd-packaging.groovy

```
import com.ibm.dbb.build.*
import com.ibm.dbb.dependency.*
import com.ibm.dbb.build.report.*
import com.ibm.dbb.build.report.records.*
import groovy.time.*
import com.ibm.dbb.build.VersionInfo
import groovy.xml.MarkupBuilder
/**
 * This script creates a version in UrbanCode Deploy based on the build result.
 *
 * usage: dbb-ucd-packaging.groovy [options]
 *
 * options:
 *  -b,--buztool <file>        Absolute path to UrbanCode Deploy buztool.sh script
 *  -w,--workDir <dir>         Absolute path to the DBB build output directory
 *  -c,--component <name>      Name of the UCD component to create version in
 *  -v,--versionName <name>    Name of the UCD component version name (Optional)
 *  -h,--help                  Prints this message
 *  -ar,--artifactRepository   Absolute path to Artifact Resository Server connection file (Optional)
 *  -prop,--propertyFile       Absolute path to property file (Optional). From UCD v7.1.x and greater it replace the -
ar option.
 *  -p,--preview               Preview, not executing buztool.sh
 *
 * notes:
 *   This script uses ship list specification and buztool parameters which are
 *   introduced since UCD v7.0.4.
 *
 *   When used with an earlier version of UCD, please
 *   modify the script to remove the code that creates the top level property and
 *   the code that uses the -o buztool parameter.
 *
 * Version 0 - 2019
 *  originally called deploy.groovy
 *
 * Version 1 - 2020-01
 *  New option to support new UCD + Artifactory Support
 *
 * Version 2 - 2020-06
 *  Added option to define UCD component version name (optional)
 *  Option -ar now optional; renamed to --artifactRepository,
 *   now supporting both external artifact repository (Artifactory/Nexus) + UCD Codestation
 *  Added preview option to skip buztool.sh execution
 *
 * Version 3 - 2020-08
 *  Fix ucd component version property for buildResultUrl
 *  Added support for build outputs declared in a CopyToPDS Build Record (JCL, REXX, Shared copybooks, etc.)
 *   Keep backward compatibility with older toolkits
 *
 * Version 4 - 2020-11
 *  Take into account new properties for Artifact Resository Server connection.
 *
 */

// start create version
def properties = parseInput(args)
def startTime = new Date()
properties.startTime = startTime.format("yyyyMMdd.hhmmss.mmm")
println("** Create version start at $properties.startTime")
println("** Properties at startup:")
properties.each{k,v->
```

```
        println "   $k -> $v"
}

// read build report data
println("** Read build report data from $properties.workDir/BuildReport.json")
def jsonOutputFile = new File("${properties.workDir}/BuildReport.json")

if(!jsonOutputFile.exists()){
        println("** Build report data at $properties.workDir/BuildReport.json not found")
        System.exit(1)
}

def buildReport= BuildReport.parse(new FileInputStream(jsonOutputFile))

// parse build report to find the build result meta info
def buildResult = buildReport.getRecords().findAll{it.getType()==DefaultRecordFactory.TYPE_BUILD_RESULT}[0];
def dependencies = buildReport.getRecords().findAll{it.getType()==DefaultRecordFactory.TYPE_DEPENDENCY_SET};

// parse build report to find the build outputs to be deployed.
println("** Find deployable outputs in the build report ")

// the following example finds all the build outputs with and without deployType
//def executes= buildReport.getRecords().findAll{
//        it.getType()==DefaultRecordFactory.TYPE_EXECUTE
//}

// Print warning, that extraction of COPY_TO_PDS records are not supported with older versions of the dbb toolkit.
dbbVersion = new VersionInfo().getVersion()
println "   ! Buildrecord type TYPE_COPY_TO_PDS is supported with DBB toolkit 1.0.8 and higher. Identified DBB Toolkit
version $dbbVersion. Extracting build records for TYPE_COPY_TO_PDS will be skipped."

// finds all the build outputs with a deployType
def executes= buildReport.getRecords().findAll{
        try {
                (it.getType()==DefaultRecordFactory.TYPE_EXECUTE || it.getType()==DefaultRecordFactory.TYPE_COPY_TO_PDS)
&&
                                !it.getOutputs().findAll{ o ->
                                        o.deployType != null && o.deployType != 'ZUNIT-TESTCASE'
                                }.isEmpty()
        } catch (Exception e){}
}

executes.each { it.getOutputs().each { println("   ${it.dataset}, ${it.deployType}")}}

if ( executes.size() == 0 ) {
        println("** No items to deploy. Skipping ship list generation.")
        System.exit(0)
}

// generate ship list file. specification of UCD ship list can be found at
// https://www.ibm.com/support/knowledgecenter/SS4GSP_6.2.7/com.ibm.udeploy.doc/topics/zos_shiplistfiles.html
println("** Generate UCD ship list file")
def writer = new StringWriter()
writer.write("<?xml version=\"1.0\" encoding=\"CP037\"?>\n");
def xml = new MarkupBuilder(writer)
xml.manifest(type:"MANIFEST_SHIPLIST"){
        //top level property will be added as version properties
        //requires UCD v6.2.6 and above
        property(name : buildResult.getGroup() + "-buildResultUrl", value : buildResult.getUrl())
        //iterate through the outputs and add container and resource elements
        executes.each{ execute ->
                execute.getOutputs().each{ output ->
                        def (ds,member) = getDatasetName(output.dataset)
                        container(name:ds, type:"PDS"){
                                resource(name:member, type:"PDSMember", deployType:output.deployType){
                                        // add any custom properties needed
                                        property(name:"buildcommand", value:execute.getCommand())
                                        // Only TYPE_EXECUTE Records carry options
```

```
                                                if (execute.getType()==DefaultRecordFactory.TYPE_EXECUTE)
property(name:"buildoptions", value:execute.getOptions())
                                                // Sample to add additional properties. Here: adding db2 properties for a DBRM
                                                //   which where added to the build report through a basic PropertiesRecord.
                                                if (output.deployType.equals("DBRM")){
                                                        propertyRecord = buildReport.getRecords().findAll{
                                                                it.getType()==DefaultRecordFactory.TYPE_PROPERTIES &&
it.getProperty("file")==execute.getFile()
                                                        }
                                                        propertyRecord.each { propertyRec ->
                                                                // Iterate Properties
                                                                (propertyRec.getProperties()).each {
                                                                        property(name:"$it.key", value:it.value)

                                                                }
                                                        }
                                                }

                                                // add source information
                                                inputs(url : ""){
                                                        input(name : execute.getFile(), compileType : "Main")
                                                        dependencies.each{
                                                                if(it.getId() == execute.getFile()){
                                                                        it.getAllDependencies().each{
                                                                                def displayName = it.getFile()? it.getFile() :
it.getLname()

                                                                                input(name : displayName, compileType :
it.getCategory())
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
}
println("** Write ship list file to  $properties.workDir/shiplist.xml")
def shiplistFile = new File("${properties.workDir}/shiplist.xml")
shiplistFile.text = writer

// assemble and run UCD buztool command to create a version. An example of the command is like below
// /opt/ibm-ucd/agent/bin/buztool.sh createzosversion -c MYCOMP -s /var/dbb/workDir/shiplist.xml
// command parameters can be found at
// https://www.ibm.com/support/knowledgecenter/SS4GSP_6.2.7/com.ibm.udeploy.doc/topics/zos_runtools_uss.html

def cmd = [
        properties.buztoolPath,
        "createzosversion",
        "-c",
        properties.component,
        "-s",
        "$properties.workDir/shiplist.xml",
        //requires UCD v6.2.6 and above
        "-o",
        "${properties.workDir}/buztool.output"

]
// set artifactRepository option if specified
if (properties.artifactRepositorySettings) {
        cmd << "-ar"
        cmd << properties.artifactRepositorySettings
}

// set propertyFile option if specified
if (properties.propertyFileSettings) {
        cmd << "-prop"
        cmd << properties.propertyFileSettings
}
```

```groovy
//set component version name if specified
if(properties.versionName){
        cmd << "-v"
        cmd <<  properties.versionName
}

def cmdStr = "";
cmd.each{ cmdStr = cmdStr + it + " "}
println("** Following UCD buztool cmd will be invoked")
println cmdStr

// execute command, if no preview is set
if (!properties.preview.toBoolean()){


        println("** Create version by running UCD buztool")

        StringBuffer response = new StringBuffer()
        StringBuffer error = new StringBuffer()

        def p = cmd.execute()
        p.waitForProcessOutput(response, error)
        println(response.toString())

        def rc = p.exitValue();
        if(rc==0){
                println("** buztool output properties")
                def outputProp = new Properties()
                new File("${properties.workDir}/buztool.output").withInputStream { outputProp.load(it) }
                outputProp.each{k,v->
                        println "   $k -> $v"
                }
        }else{
                println("*! Error executing buztool\n" +error.toString())
                System.exit(rc)
        }
}

/**
 * parse data set name and member name
 * @param fullname e.g. BLD.LOAD(PGM1)
 * @return e.g. (BLD.LOAD, PGM1)
 */
def getDatasetName(String fullname){
        def ds,member;
        def elements =  fullname.split("[\\(\\)]");
        ds = elements[0];
        member = elements.size()>1? elements[1] : "";
        return [ds, member];
}

def parseInput(String[] cliArgs){
        def cli = new CliBuilder(usage: "deploy.groovy [options]")
        cli.b(longOpt:'buztool', args:1, argName:'file', 'Absolute path to UrbanCode Deploy buztool.sh script')
        cli.w(longOpt:'workDir', args:1, argName:'dir', 'Absolute path to the DBB build output directory')
        cli.c(longOpt:'component', args:1, argName:'name', 'Name of the UCD component to create version in')
        cli.ar(longOpt:'artifactRepository', args:1, argName:'artifactRepositorySettings', 'Absolute path to Artifactory
Server connection file')
        cli.prop(longOpt:'propertyFile', args:1, argName:'propertyFileSettings', 'Absolute path to property file
(Optional). From UCD v7.1.x and greater it replace the -ar option')
        cli.v(longOpt:'versionName', args:1, argName:'versionName', 'Name of the UCD component version')
        cli.p(longOpt:'preview', 'Preview mode - generate shiplist, but do not run buztool.sh')
        cli.h(longOpt:'help', 'Prints this message')
        def opts = cli.parse(cliArgs)
        if (opts.h) { // if help option used, print usage and exit
                cli.usage()
                System.exit(0)
        }
```

```
        def properties = new Properties()

        // load workDir from ./build.properties if it exists
        def buildProperties = new Properties()
        def scriptDir = new File(getClass().protectionDomain.codeSource.location.path).parent
        def buildPropFile = new File("$scriptDir/build.properties")
        if (buildPropFile.exists()){
                buildPropFile.withInputStream { buildProperties.load(it) }
                if (buildProperties.workDir != null)
                        properties.workDir = buildProperties.workDir
        }

        // set command line arguments
        if (opts.w) properties.workDir = opts.w
        if (opts.b) properties.buztoolPath = opts.b
        if (opts.c) properties.component = opts.c
        if (opts.ar) properties.artifactRepositorySettings = opts.ar
        if (opts.prop) properties.propertyFileSettings = opts.prop
        if (opts.v) properties.versionName = opts.v
        properties.preview = (opts.p) ? 'true' : 'false'

        // validate required properties
        try {
                assert properties.buztoolPath : "Missing property buztool script path"
                assert properties.workDir: "Missing property build work directory"
                assert properties.component: "Missing property UCD component"
        } catch (AssertionError e) {
                cli.usage()
                throw e
        }
        return properties
}
```

Summary

You now have a sample repo and a pipeline running DBB in Impact mode. Use IDz to make source code changes and commit/push them back to Azure. You can then manually run your pipeline or enable it for auto-run after a commit.

Sample Pipeline Output