TechNotes – Azure DevOps Release Pipeline with DBB Artifacts (NLopez IBM/DAT)

Table of Contents

[Overview 1](#_Toc66085814)

[Setup 3](#_Toc66085815)

[CI pipeline scripts 4](#_Toc66085816)

[Define Azure Variables 4](#_Toc66085817)

[CI-Task: Package DBB Artifacts 5](#_Toc66085818)

[CI-Task: Windows Agent SCP 6](#_Toc66085819)

[CI-Task: Publish DBB package 7](#_Toc66085820)

[Release pipeline scripts 8](#_Toc66085821)

[Release-Task: Universal download - DBB Package to Agent 8](#_Toc66085822)

[Release-Task: Init Deploy WorkDir 9](#_Toc66085823)

[Release-Task: Windows Agent SCP: Copy Azure DBB Package to zOS Staging area 10](#_Toc66085824)

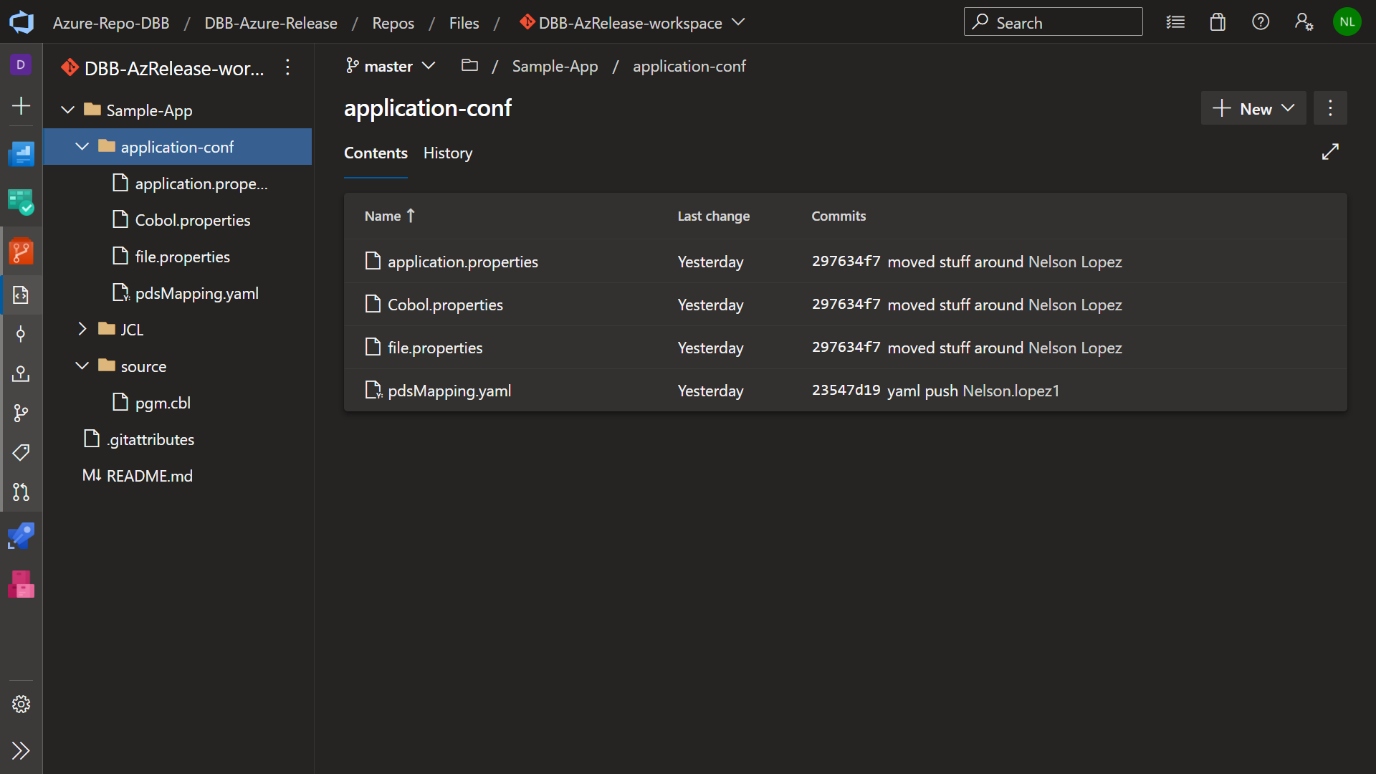
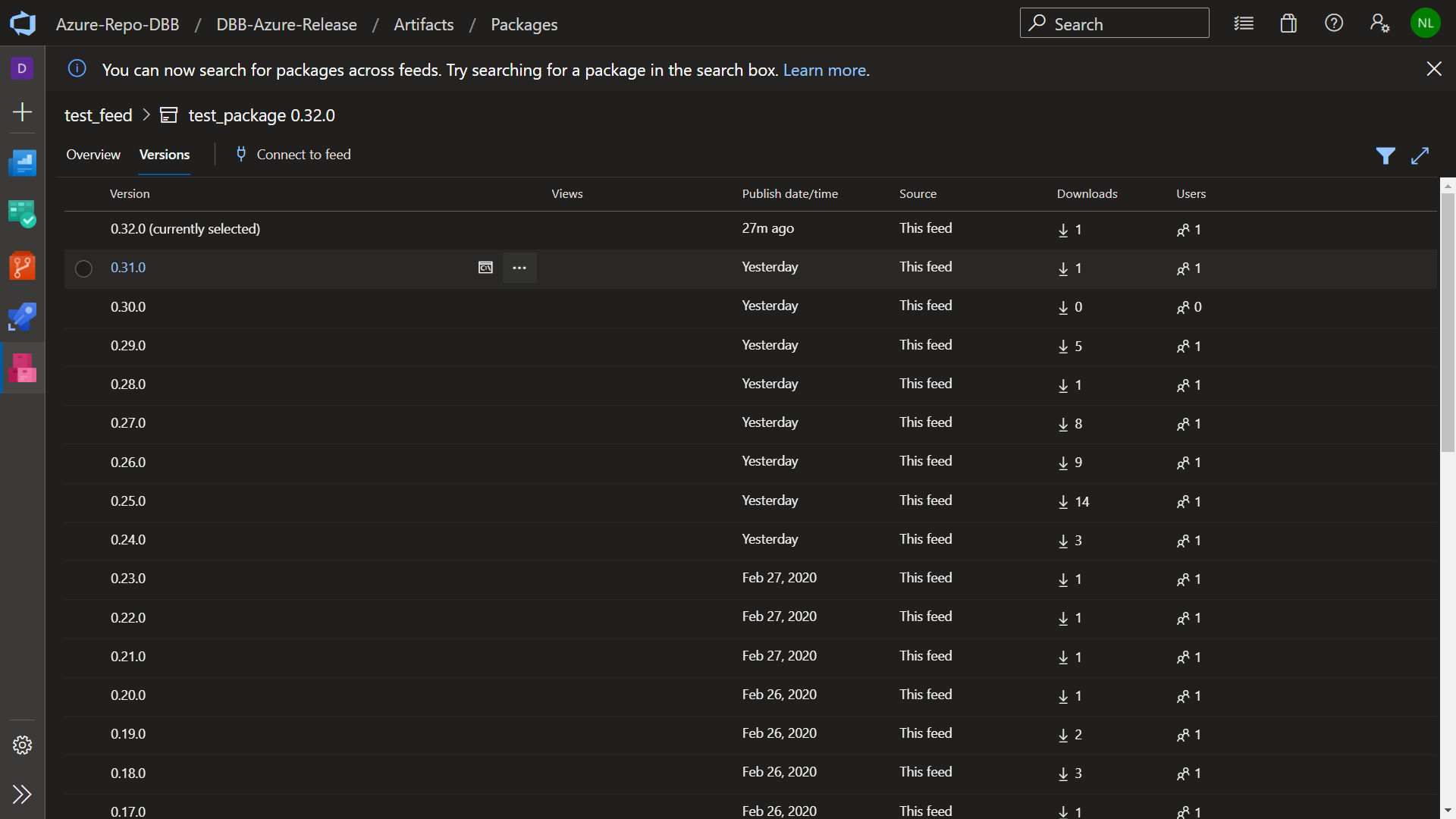
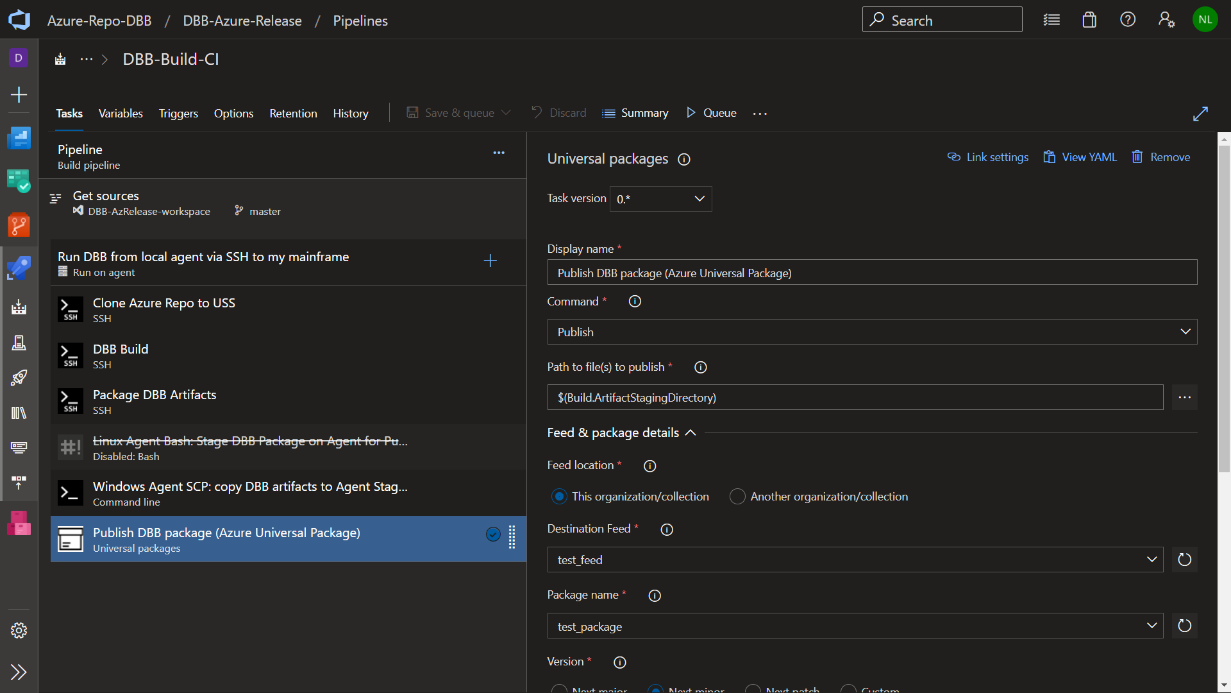
[Release-Task: Deploy Package to zOS 11](#_Toc66085825)

[Sample Script Repo (Private) 12](#_Toc66085826)

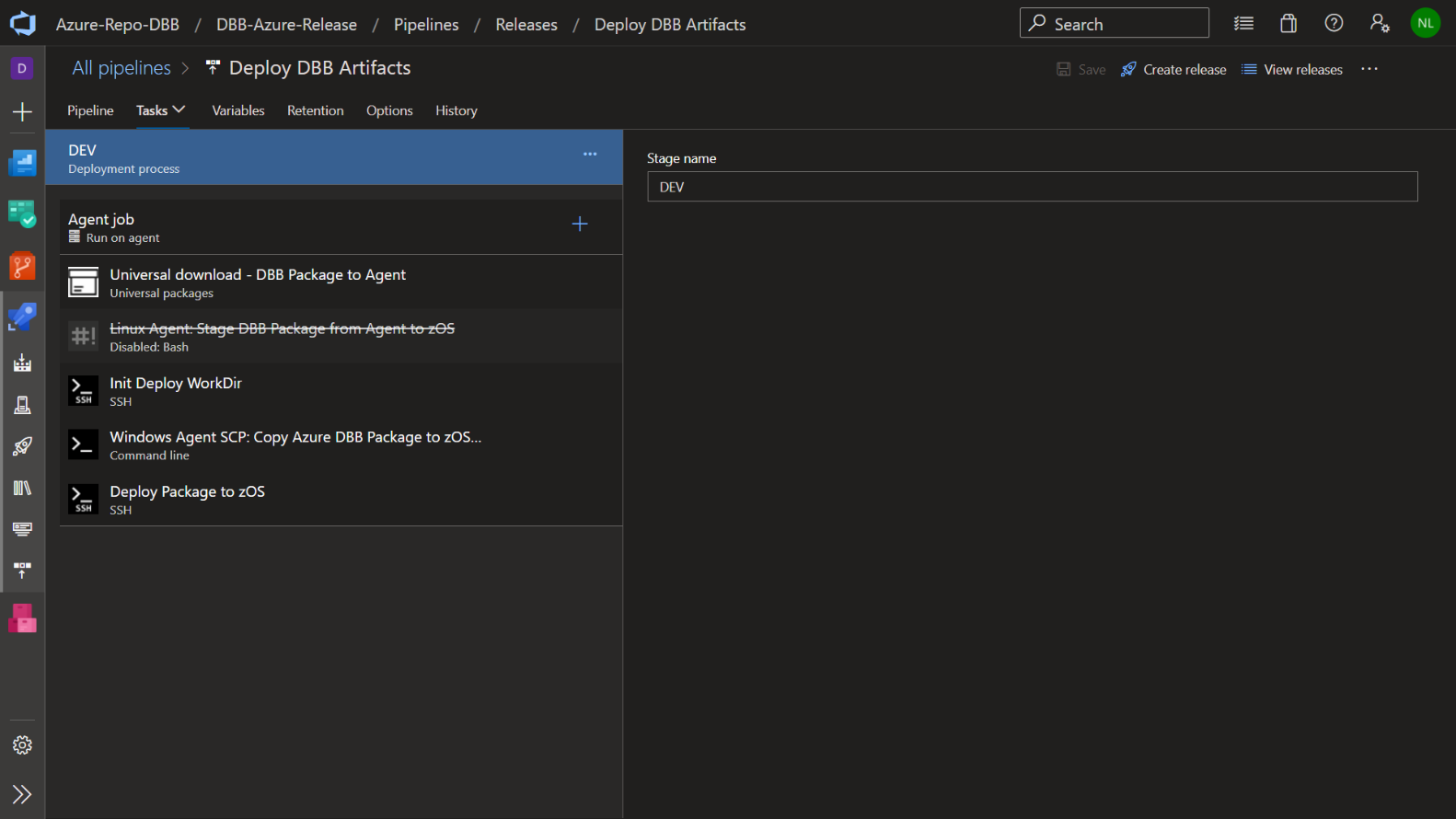
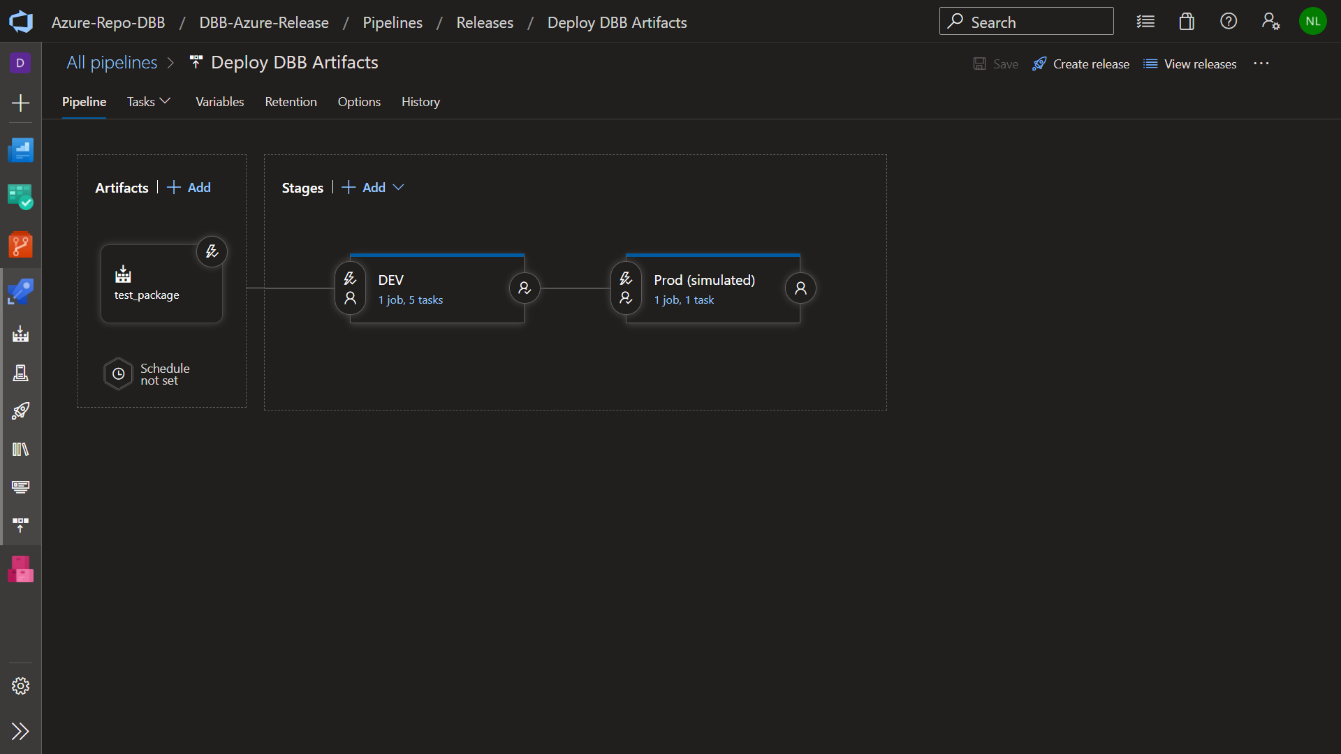
# Overview

This document builds upon the example described in the "Azure DevOps and IBM Dependency Based Build Integration"[[1]](#footnote-1) document. Some CI tasks will be redefined to support a basic Azure Release Pipeline of DBB generated artifacts.

The example uses an Azure Repo with a simple mainframe Cobol batch program. The repo is configured to trigger an Azure CI pipeline (DBB-Build-CI) with the following tasks:

* Clone the repo to the Mainframe.
* Run a DBB build
* Package the DBB artifacts into tar file including a yaml manifest and pdsMapping files.
* SCP the package to the Azure Agent staging area.
* Invoke Azure's Universal Plugin to store and version the package in an Azure Artifacts Feed (test\_feed)

The example Release pipeline has a DEV stage and a Prod stage to illustrate deployments from one to the other. The DEV stage has these tasks:

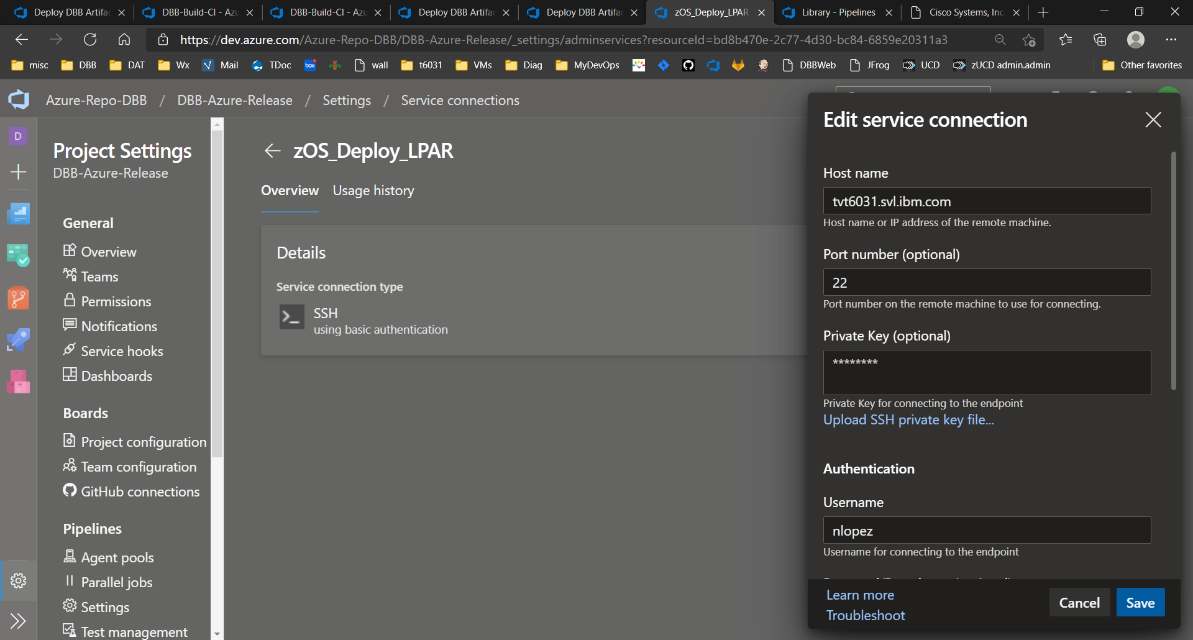
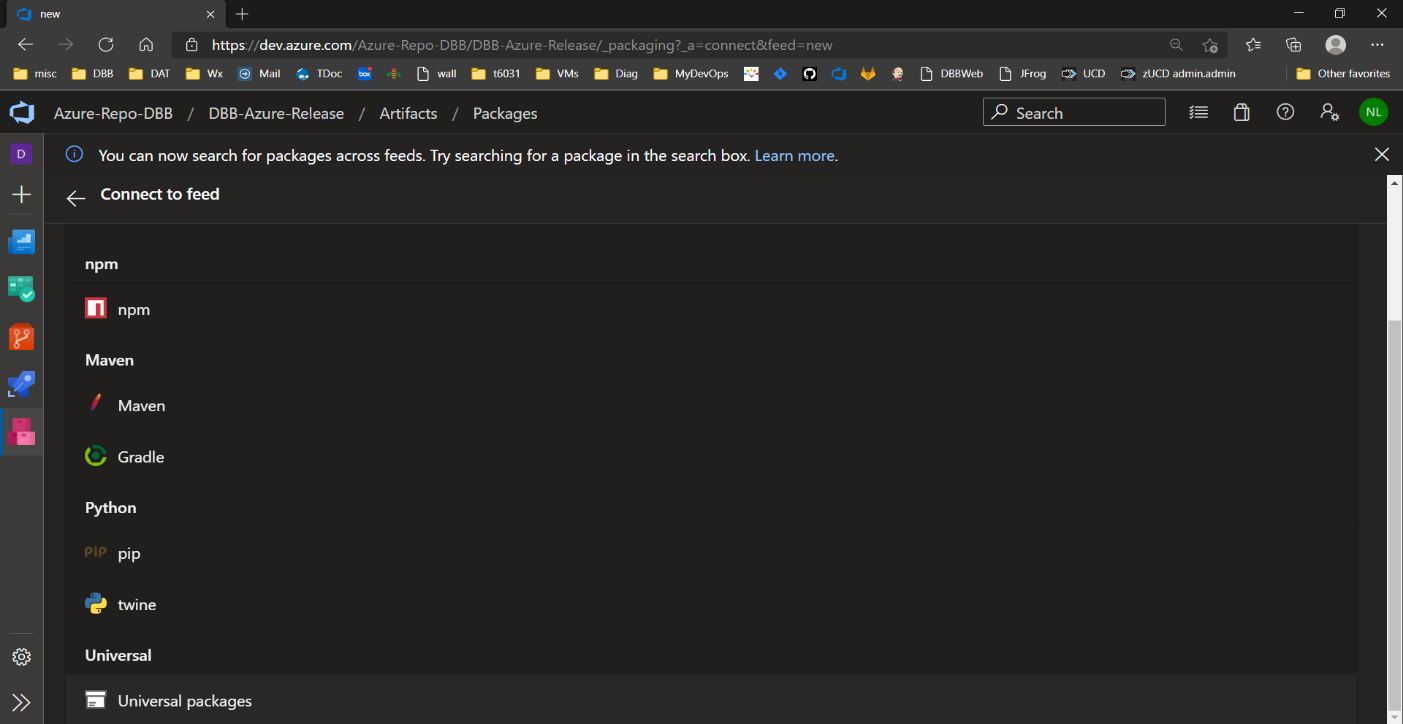
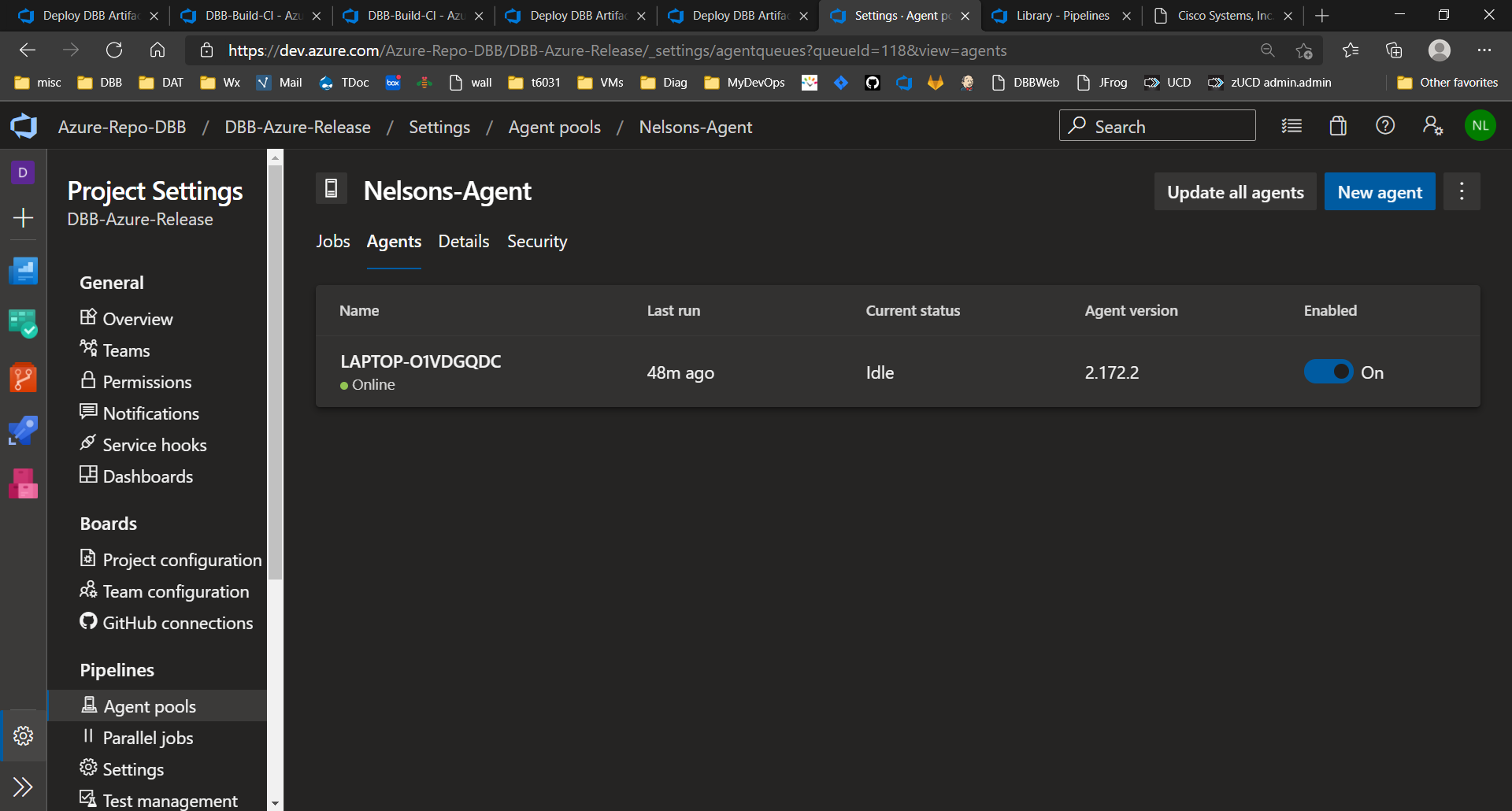
* Universal download to pull the latest version of a package and stage it on the agent.
* Initialize the target mainframe working directory.
* SCP the package to the mainframe working directory.
* Deploy - Invokes a sample script to unpack and deploy the artifacts onto the target host.

See a demo -> [**https://youtu.be/RSBxKkxhxaE?t=90**](https://youtu.be/RSBxKkxhxaE?t=90)

# Setup

It is assumed you are familiar with Azure DevOps services, setting up agents, end-points, security networking requirements and Artifacts feeds.

A basic DBB based Azure DevOps CI/CD configuration requires:

* An Azure agent with access to one or more mainframe hosts over SSH (SAF User ID/Password + OMVS Segment)
* Service connection(s) to each host
* Pre-installed and configured zDevOps mainframe stack – IBM DBB, Rocket Git
* A set of Azure pipeline scripts to launch tasks – next section.
* An Azure Artifacts Universal Packages Feed
* A preconfigured Repo (Azure repo is used in this example)

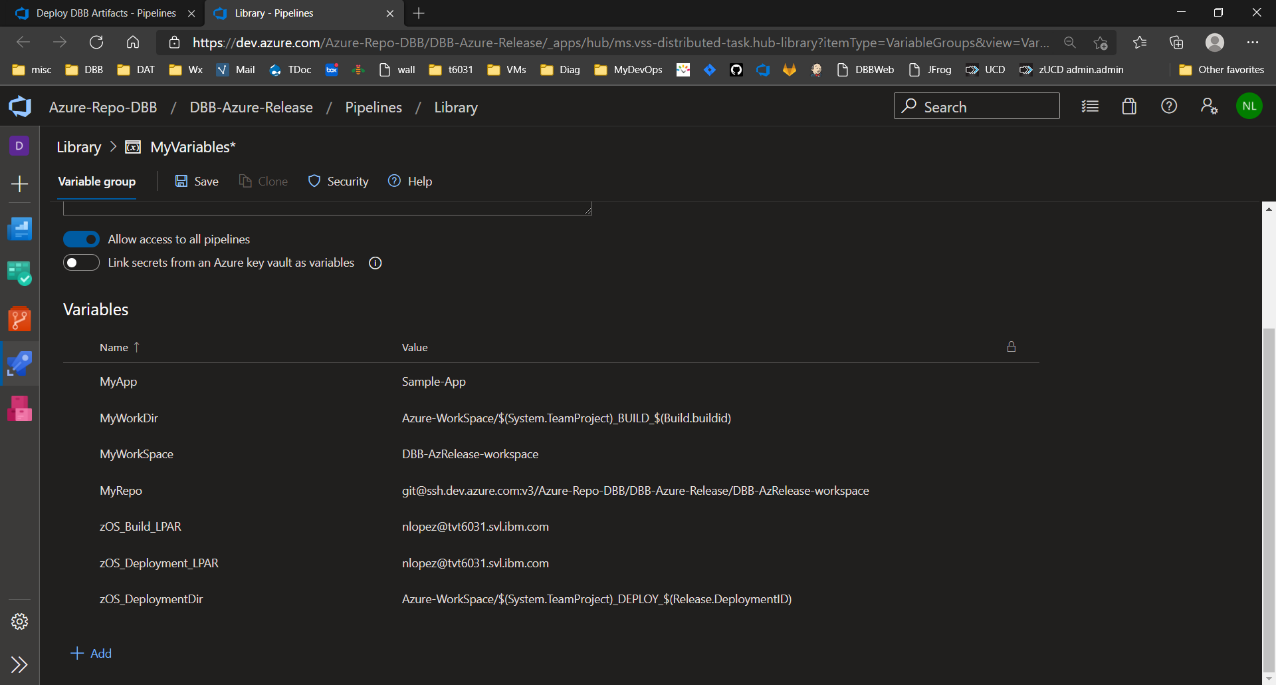
# CI pipeline scripts[[2]](#footnote-2)

As mentioned, these notes are an extension of the "Azure DevOps and IBM Dependency Based Build Integration" doc. It explains how to set up a *basic* CI pipeline to publishes artifacts using Urban Code Deploy. For an Azure Release pipeline, the publishing tasks will be changed. Instead of UCD/Artifactory we will use Azure's Artifact Repo (Feeds) and a sample groovy deployment scripts

## 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. This example uses SSH.
* **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.
  + $buildid - is automatically assigned at build time with a unique number.
  + $teamProject - is used as a 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)
* **zOS\_Build\_LPAR –** the ssh string to access a Build host
* **zOS\_Deployment\_LPAR –** the ssh string to access the target deployment host (in this example it’s the same as the Build host but can be any host with the DBB Toolkit).
* **zOS\_DeploymentDir –** a working directory to stage and deploy the artifacts.

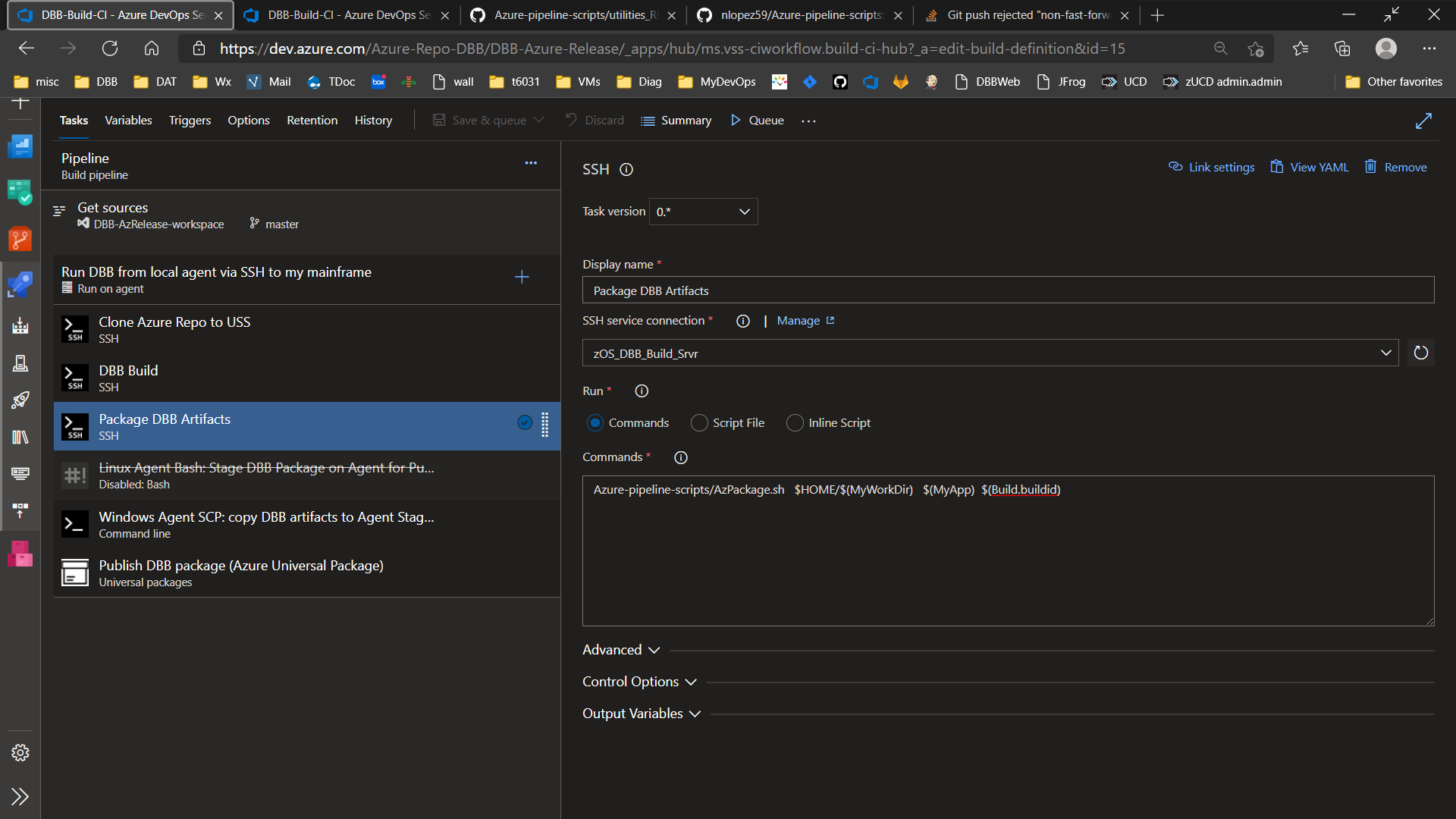


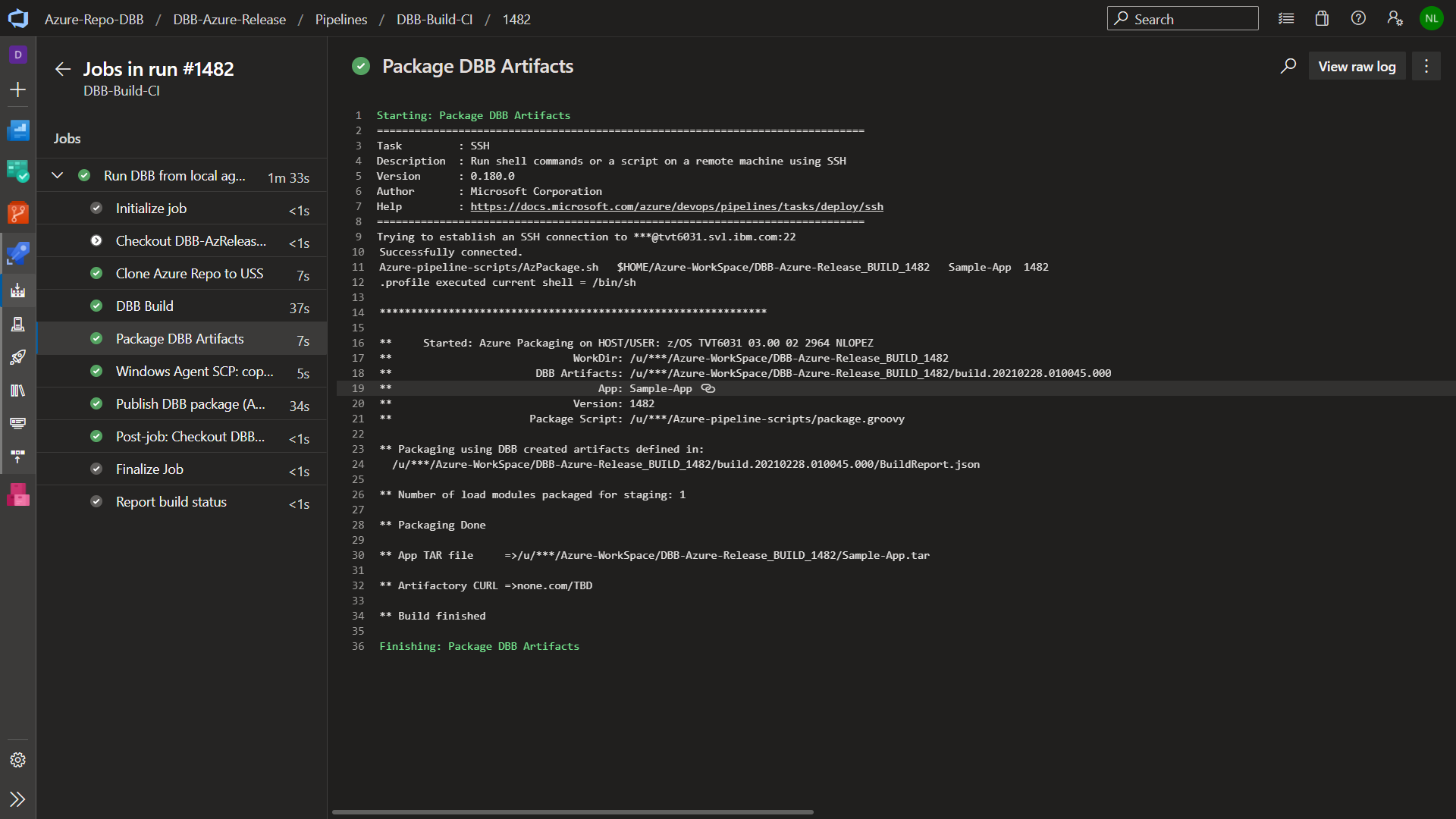
## CI-Task: Package DBB Artifacts

This replaces the task "*Define Task 3 - Publish to UCD*" used in the basic CI pipeline. It runs after the DBB Build task. It reads the DBB BuildReport.json file and tars all supported artifacts by deployment type. The sample scripts were stored in $HOME/Azure-pipeline-scripts. Your location may differ. The sample shell scripts can be obtained by your IBM DAT coordinator.

**Commands:**

**Azure-pipeline-scripts/AzPackage.sh $HOME/$(MyWorkDir) $(MyApp) $(Build.buildid)**





## CI-Task: Windows Agent SCP

**Commands:**

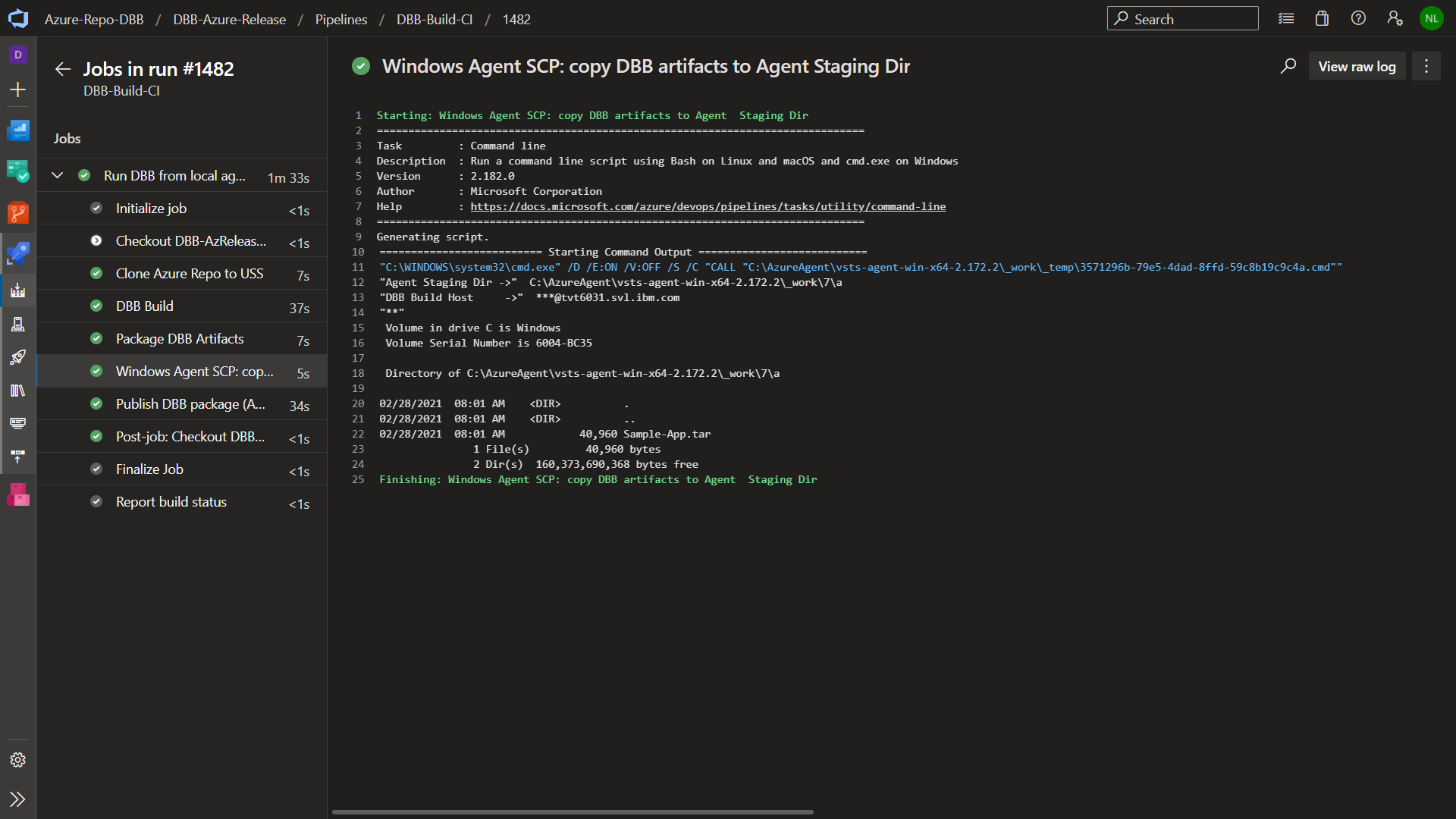
@echo "Agent Staging Dir ->" $(Build.ArtifactStagingDirectory)

@echo "DBB Build Host ->" $(zOS\_Build\_LPAR)

@echo "\*\*"

scp $(zOS\_Build\_LPAR):$(MyWorkDir)/$(MyApp).tar $(Build.ArtifactStagingDirectory)

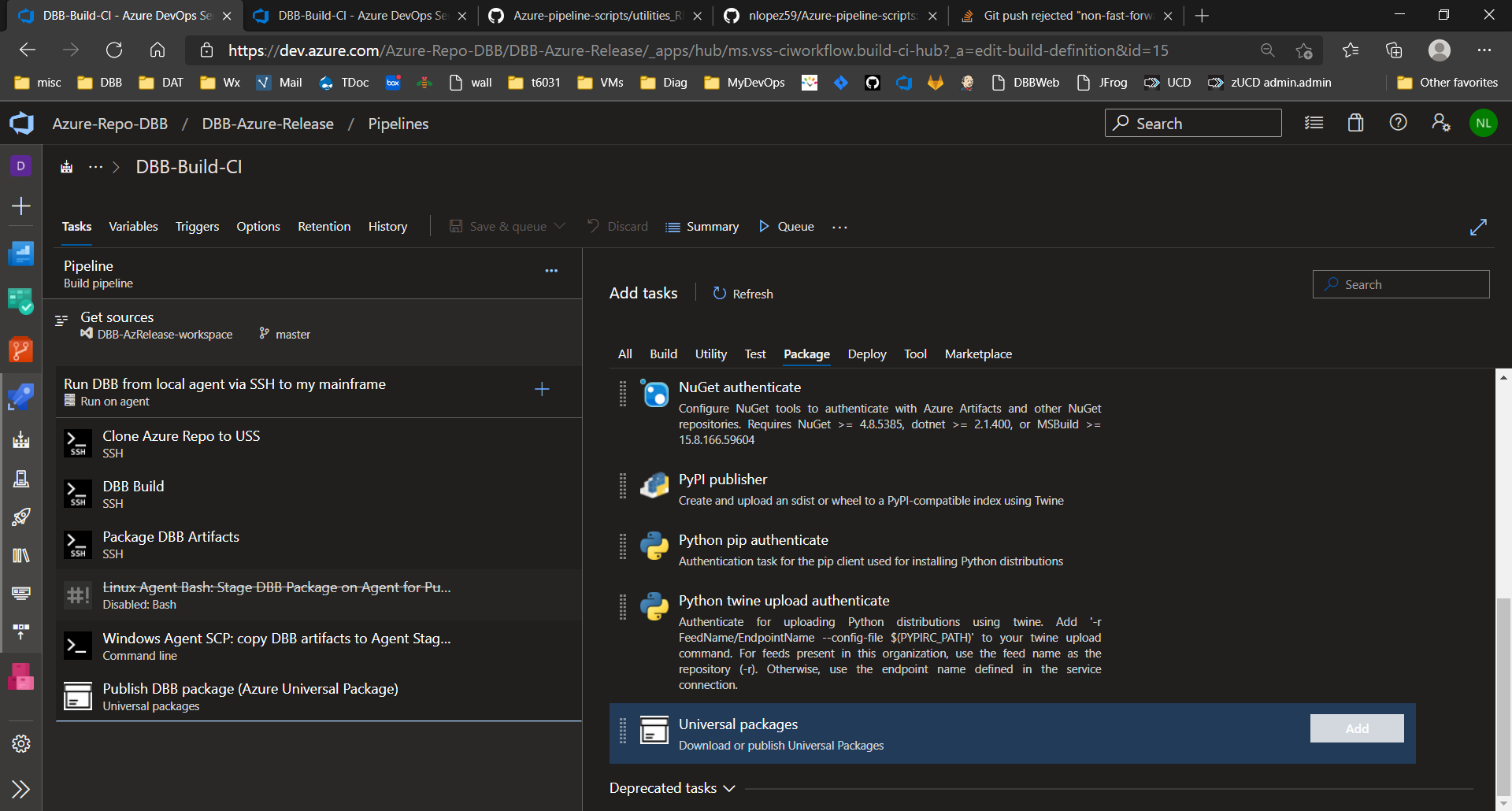
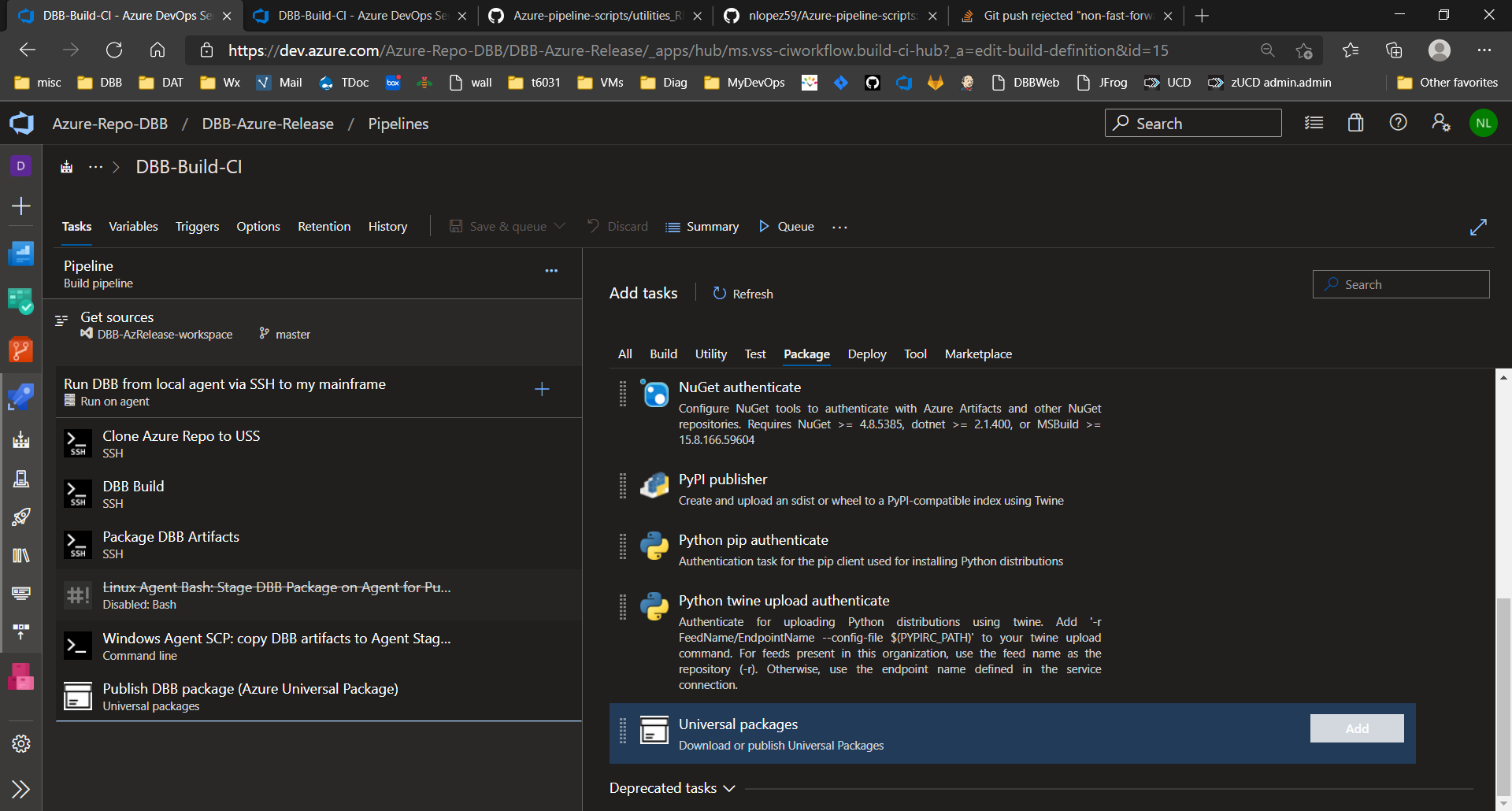
dir $(Build.ArtifactStagingDirectory)



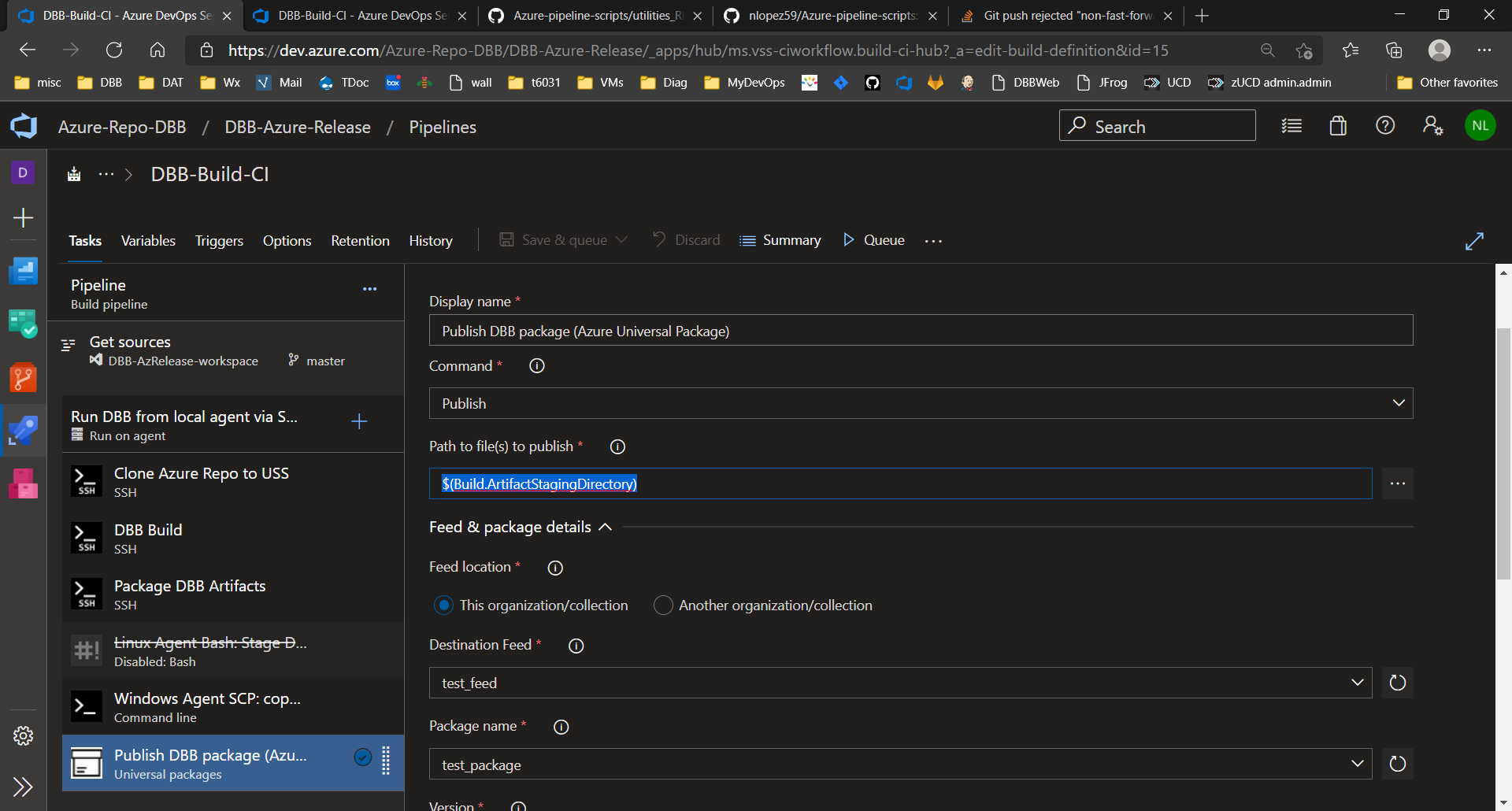
This sample uses a windows agent using pre-defined SSH keys to the host.

## CI-Task: Publish DBB package

This is a standard Azure Universal Package plugin available from Microsoft. Configure it with your Feed and package name.

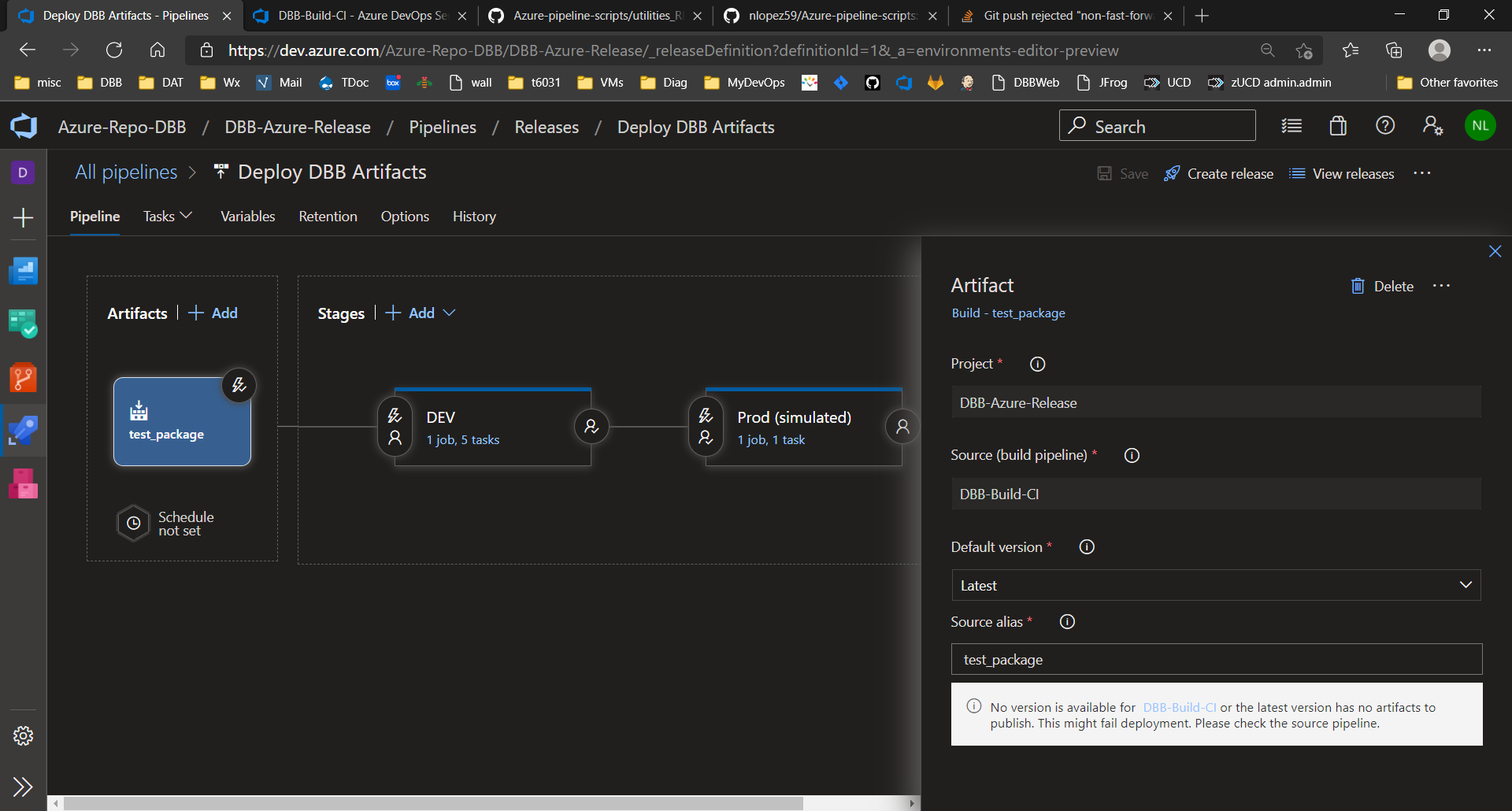


Use this path for publishing $(Build.ArtifactStagingDirectory)



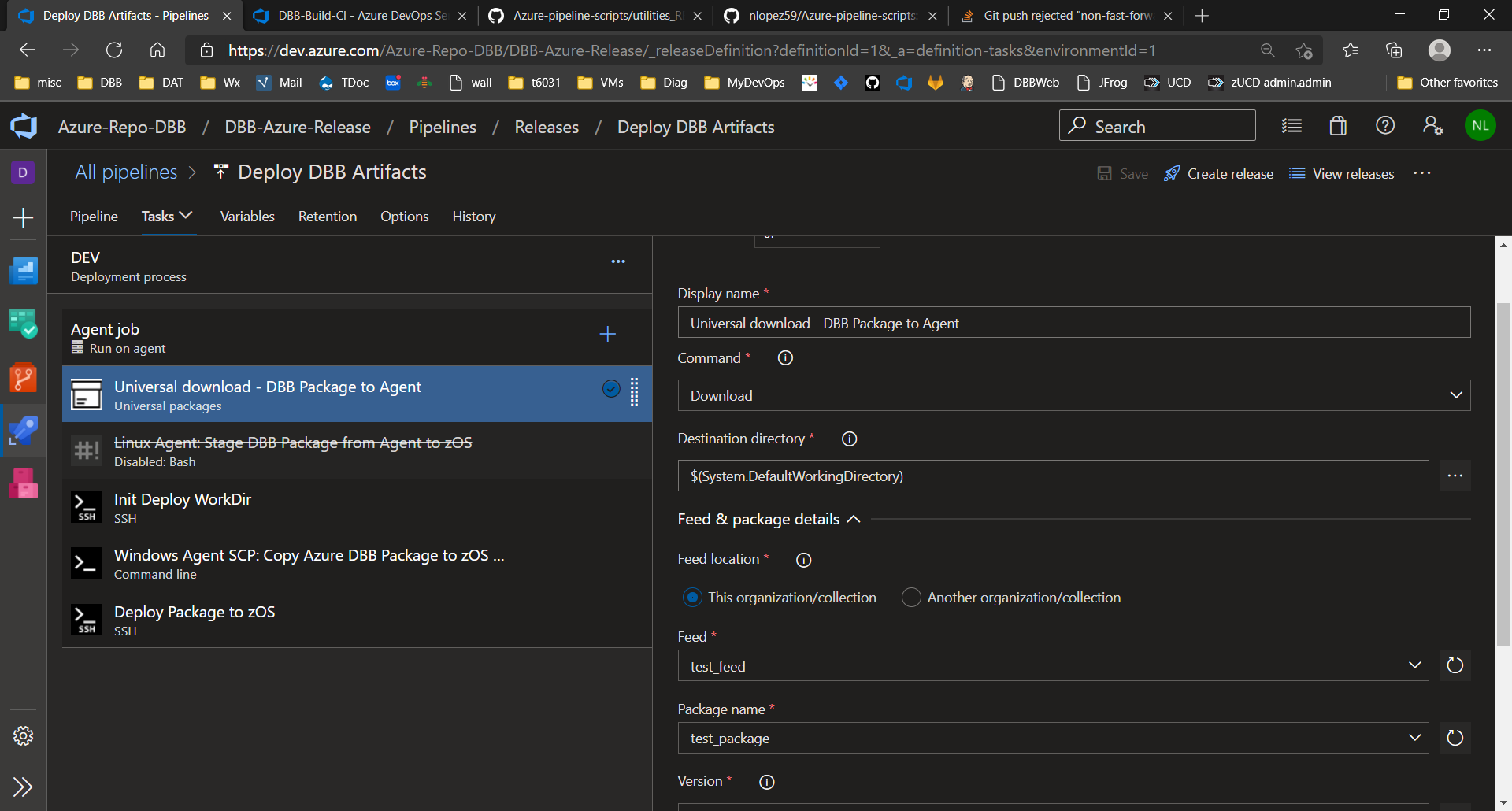
# Release pipeline scripts

Design your release pipeline to use your Feed with the following tasks for a stage called DEV as an example. Refer to Microsoft's documentation on setting up a feed and release pipelines.



## Release-Task: Universal download - DBB Package to Agent

Add the 'Universal packages' task and configure it using your Feed and package names. Use this as the Destination directory $(System.DefaultWorkingDirectory)

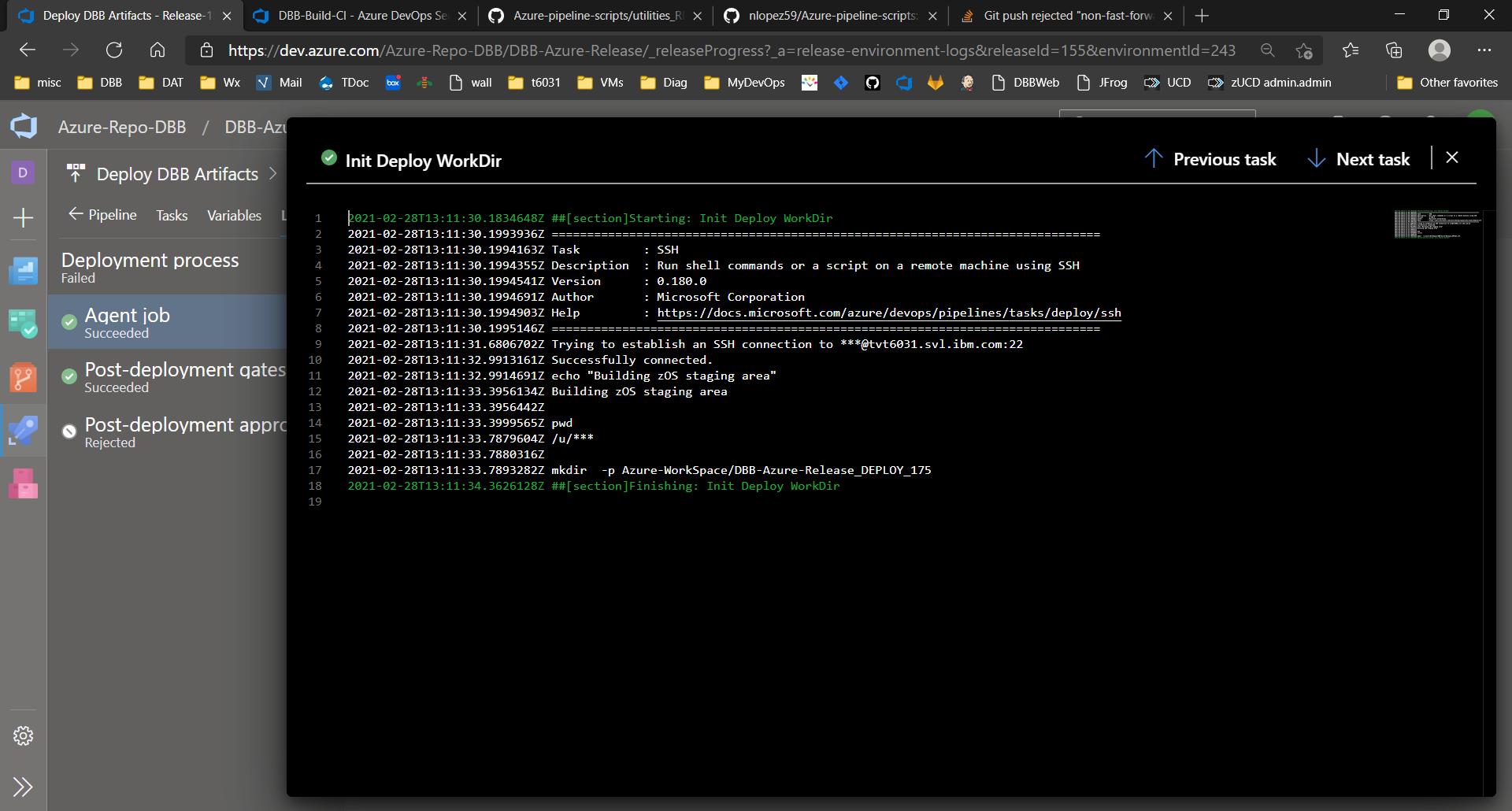


## Release-Task: Init Deploy WorkDir

**Commands:**

echo "Building zOS staging area"

mkdir -p $(zOS\_DeploymentDir)



## Release-Task: Windows Agent SCP: Copy Azure DBB Package to zOS Staging area

**Commands:**

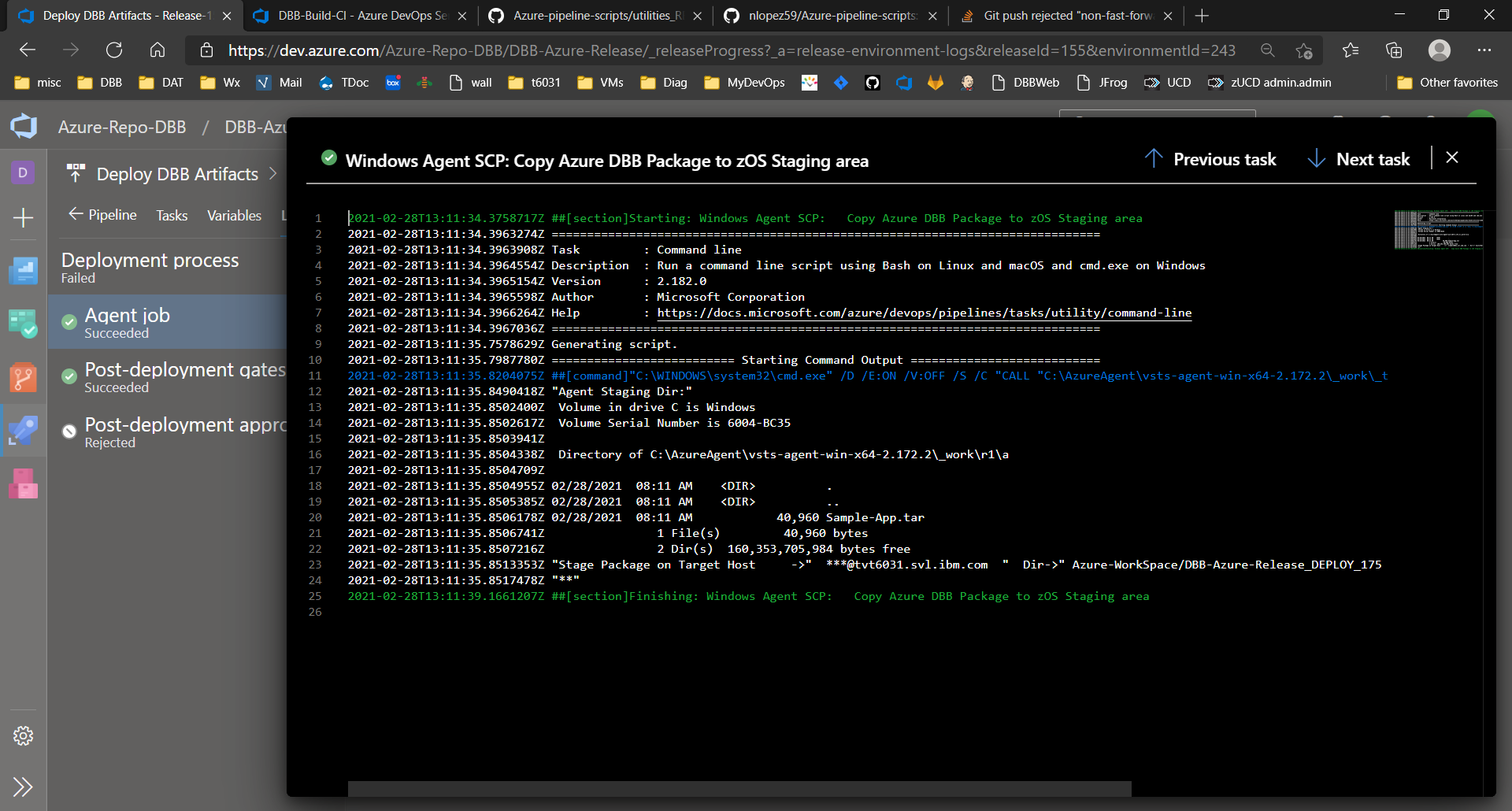
@echo "Agent Staging Dir:"

dir $(System.DefaultWorkingDirectory)

@echo "Stage Package on Target Host ->" $(zOS\_Deployment\_LPAR) " Dir->" $(zOS\_DeploymentDir)

@echo "\*\*"

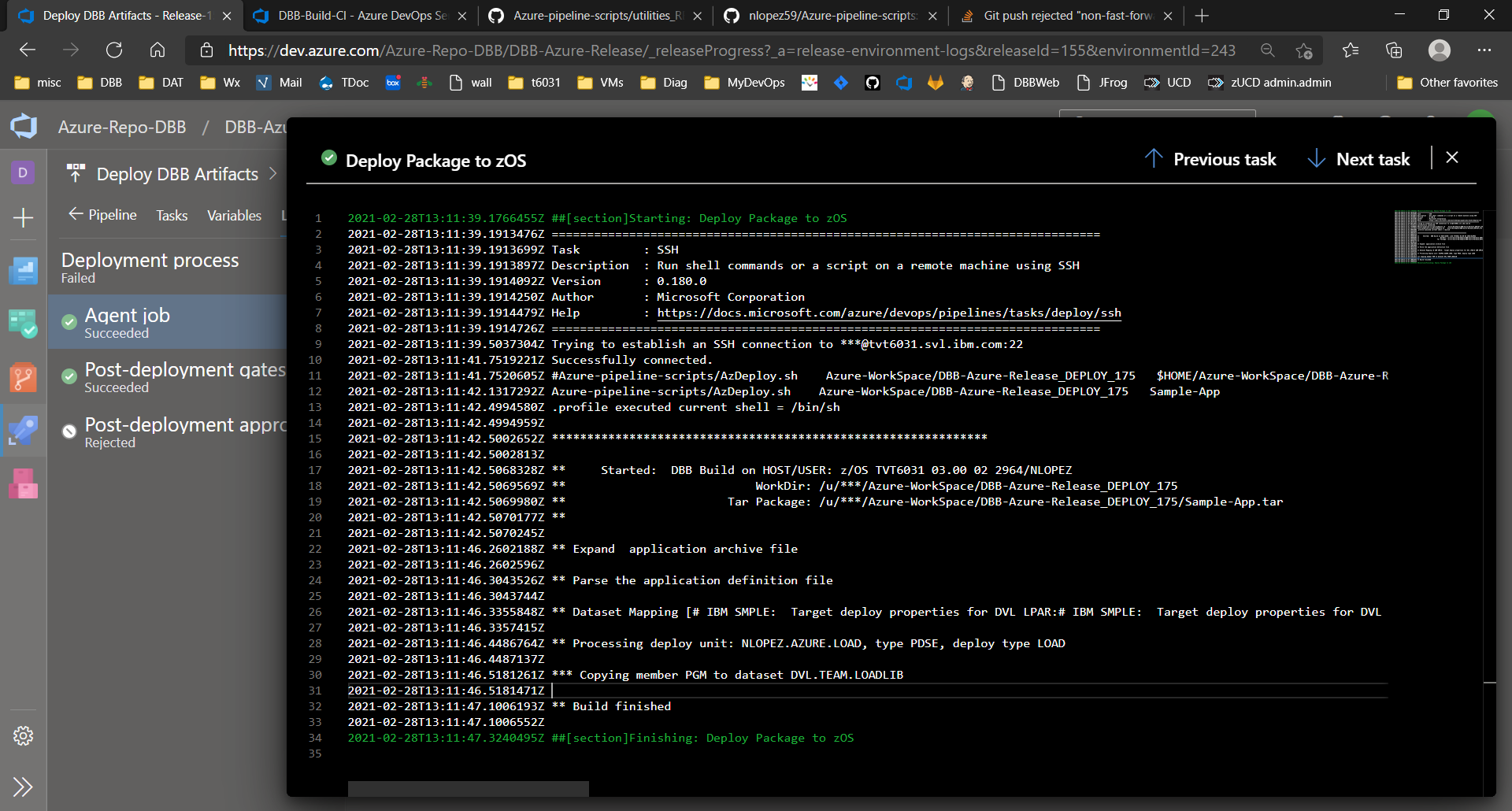
scp $(System.DefaultWorkingDirectory)/\* $(zOS\_Deployment\_LPAR):$(zOS\_DeploymentDir)

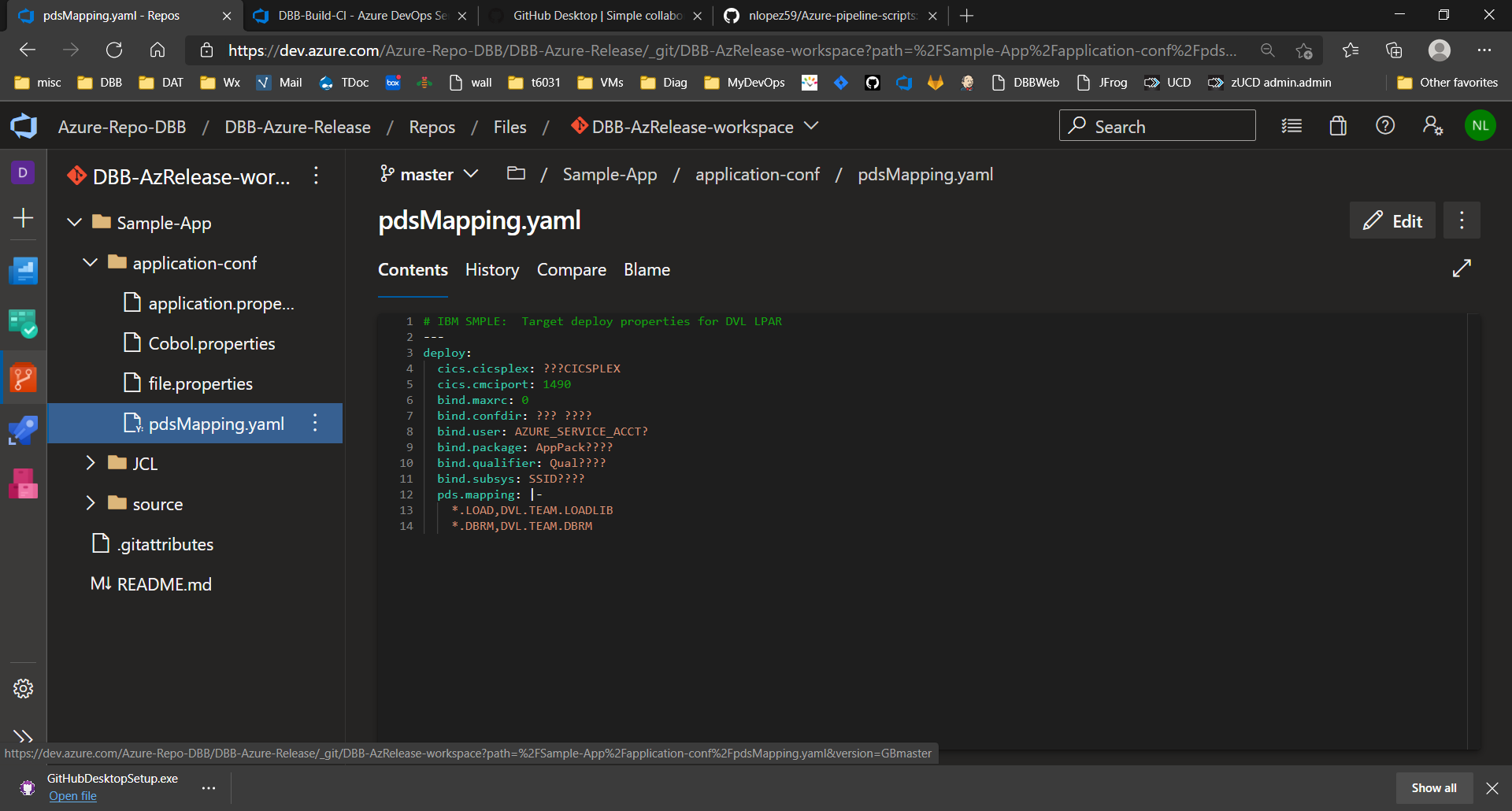


## Release-Task: Deploy Package to zOS

**Commands:**

Azure-pipeline-scripts/AzDeploy.sh $(zOS\_DeploymentDir) $(MyApp)



The Deploy script uses a hardcoded yaml file named pdsMapping.yaml. It maps the LLQ of a file being deployed to a target PDS. For example, the LLQ in PDS NLOPEZ.TEST***.LOAD*** is mapped to the PDS **DVL.TEAM.LOADLIB**. Ideally these target PDS(s) are existing files used to store Dev, QA or PROD versions of the binaries. This sample also illustrates, but does not implement, DB2 and CICS deployment properties. This could be extended to describe properties across any target environment as needed.

# IBM SAMPLE: Target deploy properties for DVL LPAR

deploy:

cics.cicsplex: ???CICSPLEX

cics.cmciport: 1490

bind.maxrc: 0

bind.confdir: ??? ????

bind.user: AZURE\_SERVICE\_ACCT?

bind.package: AppPack????

bind.qualifier: Qual????

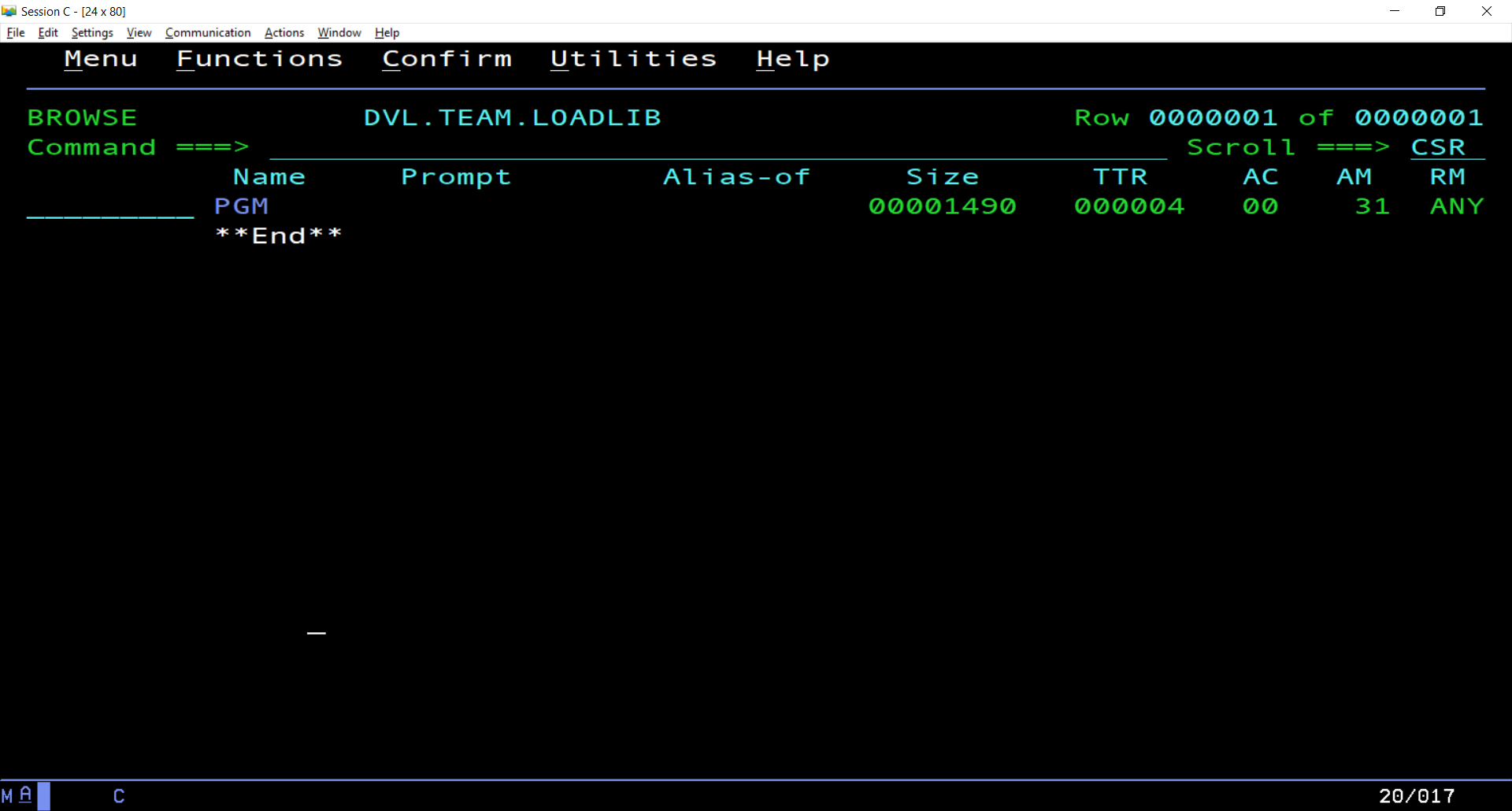
bind.subsys: SSID????

pds.mapping: |-

\*.**LOAD**,DVL.TEAM.LOADLIB

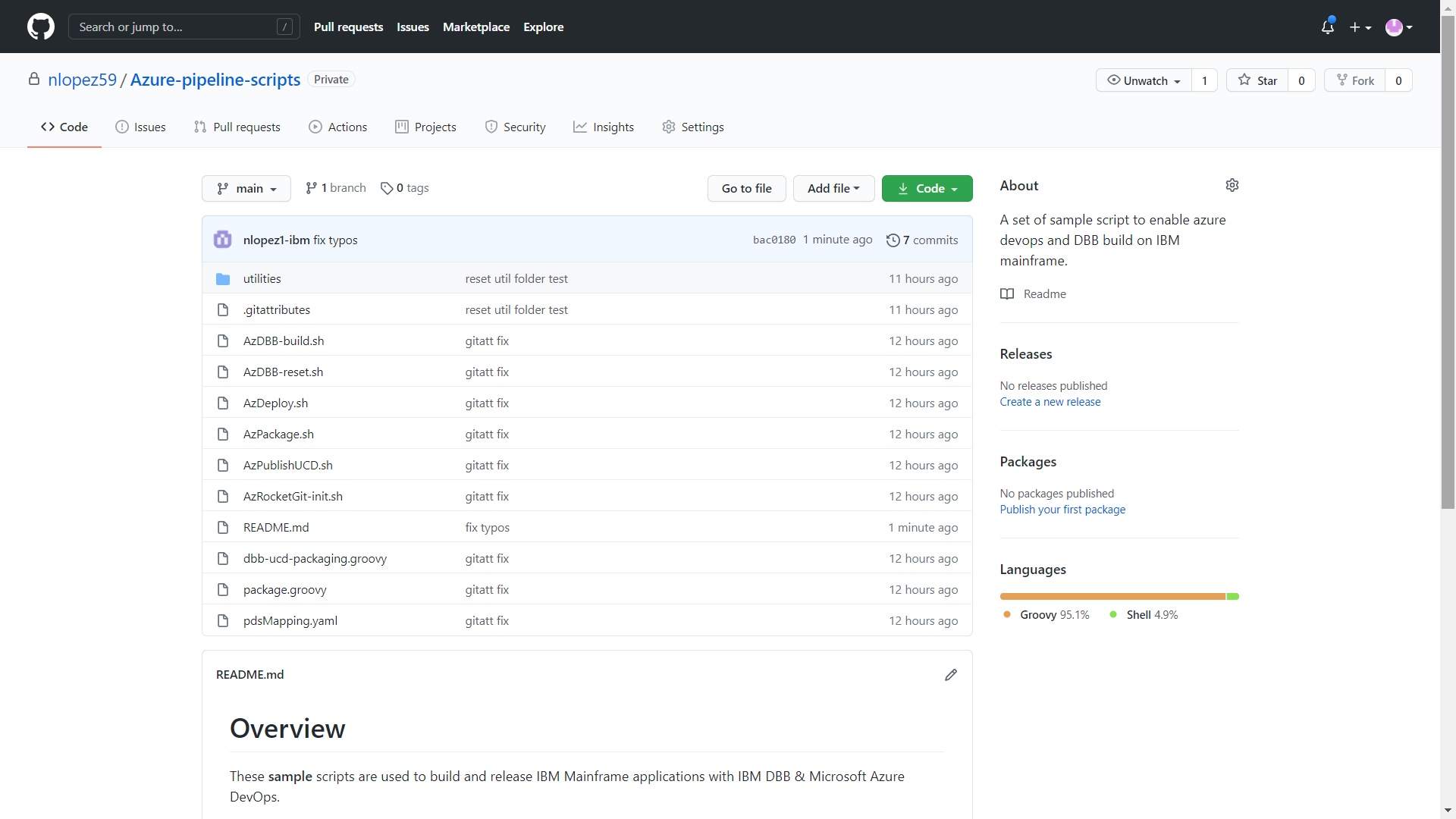
\*.DBRM,DVL.TEAM.DBRM

The result, in this example, was to deploy the sample 'PGM' built by DBB to the DVL.TEAM.LOADLIB in a DEV LPAR.



# Sample Script Repo (Private)

<https://github.com/nlopez59/Azure-pipeline-scripts>



1. Available under the resource section of the IBM DevOps Acceleration Program' s site <https://www.ibm.com/support/pages/node/6422813> [↑](#footnote-ref-1)
2. *All scripts are provided as samples with no user documentation. You are responsible for testing and supporting them. IBM provides no formal support or liability. The IBM DevOps Acceleration team can be consulted on guidance and limited support through the POC/Pilot engagement process.*  [↑](#footnote-ref-2)