An IBM Document from

IBM Z DevOps Acceleration Program

July, 2023 - Release 1.4

Building enterprise CI/CD pipelines for mainframe applications using the IBM Z & Cloud Modernization Stack

Mathieu Dalbin
mathieu.dalbin@fr.ibm.com

Nelson Lopez
Nelson.lopez1@ibm.com

Abstract

This document describes how Wazi components of the IBM Z & Cloud Modernization Stack offering can be integrated to create an enterprise CI/CD pipeline leveraging Cloud platform capabilities.

Table of content

1 Introduction

1.1 Outline

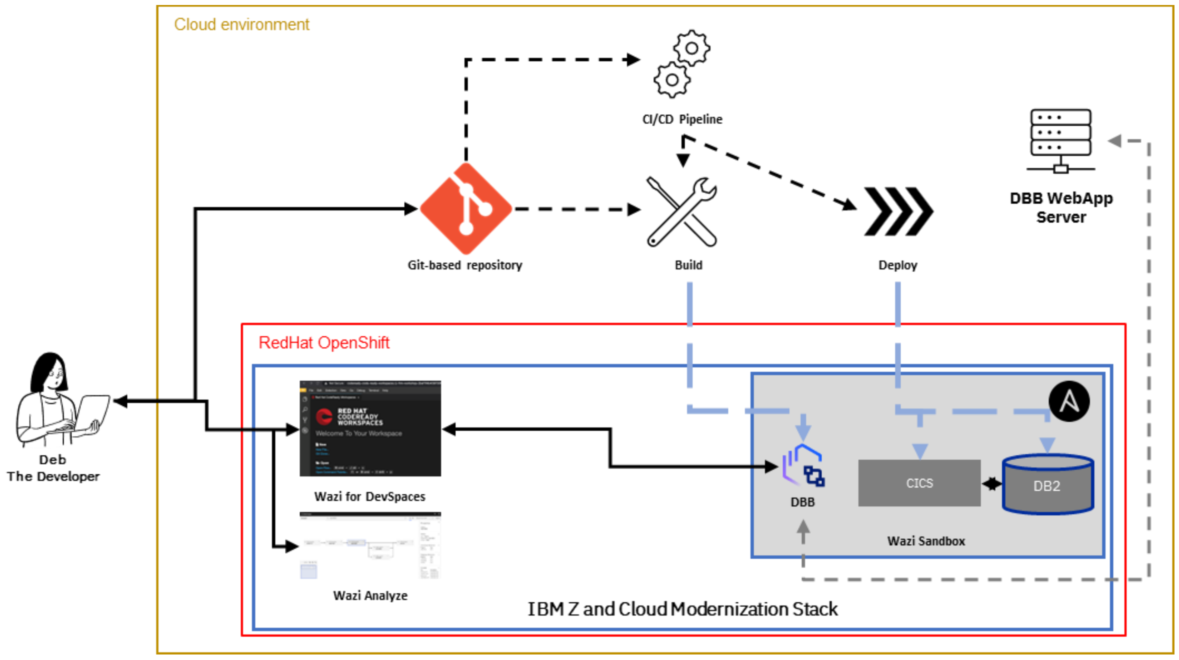
This document is a cookbook presenting different recipes to build and configure enterprise CI/CD pipelines for developing and testing mainframe applications on Cloud platforms, by leveraging the IBM Z & Cloud Modernization Stack offering. The IBM Z & Cloud Modernization Stack includes various capabilities to modernize mainframe applications, based on the RedHat OpenShift platform. For development and test purposes, this offering includes Wazi for DevSpaces, Wazi Analyze and Wazi Sandbox, which together provide mainframe developers with a Cloud-native experience with zero install.

This publication describes one way to setup the integration but doesn't cover all the other possible options. The official documentation of each component provides guidance and tips on how to set up the different parts, while the recipes presented in this documentation focus on required additional configurations to build a CI/CD pipeline for editing (Wazi for DevSpaces), understanding (Wazi Analyze), building (with IBM Dependency Based Build) and deploying mainframe applications to the virtualized execution runtime (Wazi Sandbox).

The described steps are based on Wazi DevSpaces version 2.3, which provides Wazi Analyze as a sidecar component and Wazi Sandbox version 2.3. For this demonstration, the Extended ADCD distribution will be leveraged, which ships IBM Dependency Based Build (DBB) version 1.1.3. The following recipes may not work be applied for older versions of these products, nor with newer versions that may require a different configuration.

This document will describe the necessary customization of Wazi Sandbox, Wazi for DevSpaces and Wazi Analyze to integrate these components and start to build a CI/CD pipeline. These steps are required independently of which Cloud platform you are using. The second section will cover a real-life example based on the AWS Cloud platform, showcasing how the Wazi offerings can be leveraged to build an enterprise CI/CD pipeline for Mainframe applications using AWS Cloud platform services.

The following diagram shows a generic implementation of a Cloud-based development & test environment for mainframe applications:



1.2 Prerequisites

Throughout this document, it is assumed all the necessary Wazi components are already deployed to the OpenShift Cluster Platform that supports these activities. Deployments of the Wazi products are described in the IBM documentation[1]. This document details technical, manual steps required to configure the Wazi components to work together and is aiming experienced administrators and developers. However, to simplify some of the configuration steps described here, automation has been developed and is available through Ansible playbooks. A detailed tutorial[2] presents the different tasks that can be automated to save time for the developers to set their development environment up. These Ansible playbooks will not be used in the following document.

It is also assumed that the source files of the Z applications for which the CI/CD pipeline will be set up are already hosted on the Git-based service of the Cloud platform you are using. If you are interested in learning more about the migration to Git, you might find this collection interesting[3].

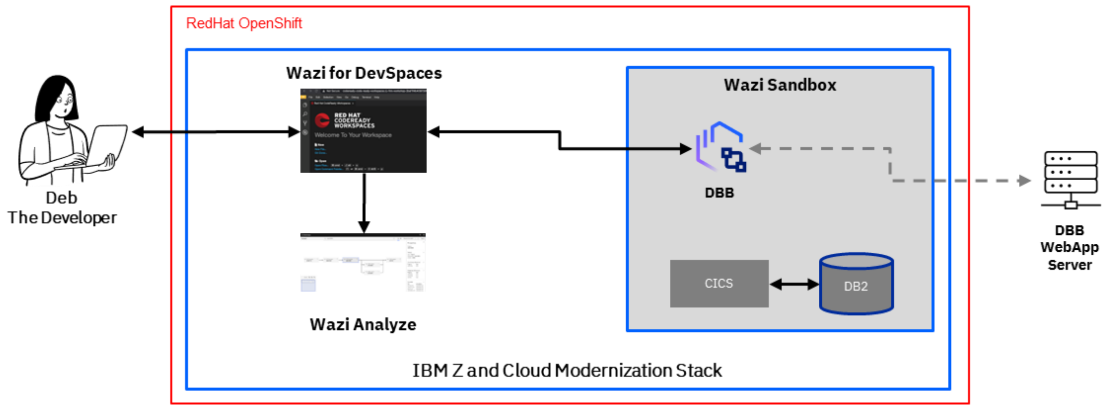
The mainframe application used in this setup is called CBSA (for CICS Banking Sample Application) and is a CICS and DB2-based application made of Cobol programs.

[1] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=installing
[2] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=developing-tutorial-improve-your-development-productivity-ansible
[3] https://ibm.github.io/mainframe-downloads/DevOps_Acceleration_Program/resources.html

2 Preparing IBM Z and Cloud Modernization Stack for CI/CD pipelines

2.1 Architecture of the setup

The following diagram shows the different components that will be configured as part of this cookbook. This architecture represents a typical setup when using the Wazi components to build a CI/CD pipeline for Mainframe applications.



When implementing changes to mainframe applications, developers are using *Wazi for DevSpaces*, a web-based portal to access their development workspace where they can use the provided features to code, debug and perform User Builds for their programs.

Leveraging *Wazi Analyze*, developers can understand the relationships between the different artifacts of the application they are working on.

Wazi Sandbox provides a virtualized Z runtime environment supporting the developers in their daily tasks, to build (compile and link-edit) mainframe programs, run debug sessions and perform Unit tests. Eventually, the programs that are just built can be deployed on the same Wazi Sandbox system to perform early functional tests. This Sandbox system is typically used by one developer, and an other Sandbox system can be used for the pipeline activities. In the setup we will detail in this document, the same Sandbox system is used for convenience, but the implementation should respect IBM's licensing conditions.

2.2 Configuring the Z runtime environment provided by Wazi Sandbox

The first building block we will focus on is Wazi Sandbox. This product, which can be deployed on an OpenShift Cluster Platform, is able to virtualize a Z system. This isolated Z system can serve several purposes, like building artifacts in a CI/CD pipeline or perform unit tests before moving the application to the next integration test environments.

Wazi Sandbox is shipped with a Z distribution known as the Extended ADCD distribution. Extended ADCD contains many products and subsystems, like CICS, DB2, IMS, already packaged and configured to work together. The use of this image drastically simplifies the setup of virtualized Z systems. Extended ADCD also comes with the necessary components to support a modern CI/CD pipeline: IBM Developer for Z (IDz) Host components (Z Explorer + RSED), Git Client on Z and IBM Dependency Based Build (DBB). For the rest of the document, we will refer to the Extended ADCD distribution as ADCD.

In the next sections, we will review the different configuration steps to perform, to get a working Z sandbox system that can integrate with the other Wazi products. It is assumed that Wazi Sandbox has already been deployed and is running properly. Setting up Wazi Sandbox is described in the IBM documentation[4]. Also, it is assumed that the Wazi Sandbox system can be reached with an SSH client (like PuTTY).

2.2.1 Configuring network for Wazi Sandbox

2.2.1.1 Exposing services endpoints of the Wazi Sandbox in the OpenShift Cluster Platform configuration

From the OpenShift Cluster Platform perspective, the Z Sandbox system is exposing services that are mapped to external ports on the pod where the containers are running. This configuration is known as a *Node Port* configuration. To set this demonstration up, some additional services must be exposed, like the CICS CMCI port, which will be used during the deployment phase to deploy CICS programs. The following configuration can be seen optional if you don't plan to use CMCI to deploy CICS artifacts, but the same instructions are relevant for any other service you need to expose outside of the Wazi Sandbox system.

The *Node Port* configuration can be updated to expose additional services, as defined in a YAML file. In the namespace where the Wazi Sandbox is deployed, with the Administrator view, navigate to the *Network* section and select *Services*. Select the *Node Port* configuration and edit its YAML definition to add the port to be exposed. The following snippet exposes the CICS CMCI port (*port 7313* on Z) to an available port on the pod (*nodePort 32313*):

```
    - name: cmci
      protocol: TCP
      port: 7313
      targetPort: 7313
      nodePort: 32313
```

Save this YAML file, and make sure the changes are committed. The CMCI port for the CICS region already setup in Wazi Sandbox image will be later used to deploy artifacts to CICS. To make the change permanent, you can also edit the YAML definition of the *WaziSandboxSystem* resource that was deployed in OpenShift Cluster Platform, by adding an entry in the *nodeport* section of the YAML file.

Most of the operations – if not all - can also be performed through OpenShift command-line commands (using the *oc* utility). For instance, the list of Node Ports can be retrieved with the `oc get -o yaml service adcd-nodeport` command. Other *oc* commands can be used to administer the Wazi Sandbox instance. This documentation page[5] lists some of the commands to be used when accessing the Wazi Sandbox system.

---

[4] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=host-option-1-managing-development-test-environments-sandbox

[5] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=sandbox-using-instance

2.2.1.2 Configuring outbound communication from the Z System

To be able to communicate with the outside world, the Z system running in Wazi Sandbox certainly needs to know about the domain names and DNS servers used in your Cloud platform. This requires a specific setup on the Z system.

To modify the network configuration of the Z machine, it is easiest to log in with a Telnet 3270 client and start a TSO session. The predefined IBMUSER user on ADCD has all the permissions required to perform this customization. By default, Wazi Sandbox is running the NZ initialization parameter (IPL load parameter), and you can verify this by running the `D IPLINFO` MVS command. If the load parameter is different, the member that contains the TCP/IP configuration to modify may be different as well. The `D SYMBOLS` command will also show which configuration is used, by looking at the *&GBLRESL.* symbol:

```
&GBLRESL.          = "GBLRESNZ"
```

If not already done, log into the Wazi Sandbox system with the IBMUSER user, and find the member *ADCD.Z25B.TCPPARMS(GBLTDANZ)*. Depending on the ADCD version you are using, this member may be located in a different library, as typically the 2nd qualifier describes the ADCD version.

In this *GBLTDANZ* member, add the following statement (around line 100):

```
SEARCH us-east-2.compute.internal
```

And the following statement (around line 145):

```
NAMESERVER 10.0.0.2
```

These statements specify that the search for domain names should be performed against this Search server. Depending on your Cloud provider and Cloud network settings, these values will likely change.

The same two lines should be added to the */etc/resolv.conf* file on USS. The *RESOLVER* task needs to be refreshed, to take into account these new definitions, with the following command (to be adapted for the ADCD version you are using):

```
F RESOLVER,REFRESH,SETUP=ADCD.Z25B.TCPPARMS(GBLRESNZ)
```

After this setup, you should be able to resolve hostnames in your Cloud platform. You can verify the correct resolution of hostname by issuing a TSO PING command from the Z Sandbox system, to a machine that belongs to your Cloud platform.

2.2.2 Configuring the build environment in Wazi Sandbox

The next step in the configuration process is about setting up the building block required to support the CI/CD pipeline. Typically, Wazi Sandbox systems are connected to the Internet, making it easier to install additional components.

2.2.2.1 Retrieving the Build framework

As the build phase is leveraging IBM Dependency Based Built (DBB) in this setup, we will retrieve the *zAppBuild*[6] framework and the *DBB Utilities*[7] scripts to the Z system. These components are available on public GitHub repositories, that can be locally cloned on the Z Sandbox system.

Before cloning these two repositories to Z, let's define a location on USS where all the required materials will be stored. In this setup, the */var/work* folder was chosen to host these components. If necessary, a specific zFS VSAM dataset can be allocated, to make sure enough room is available. If a zFS VSAM dataset is allocated, make sure it is mounted after the IPL at the right location. The following commands can then be executed, to clone the two GitHub repositories:

```
mkdir /var/work
cd /var/work
git clone https://github.com/IBM/dbb-zappbuild.git
git clone https://github.com/IBM/dbb.git
```

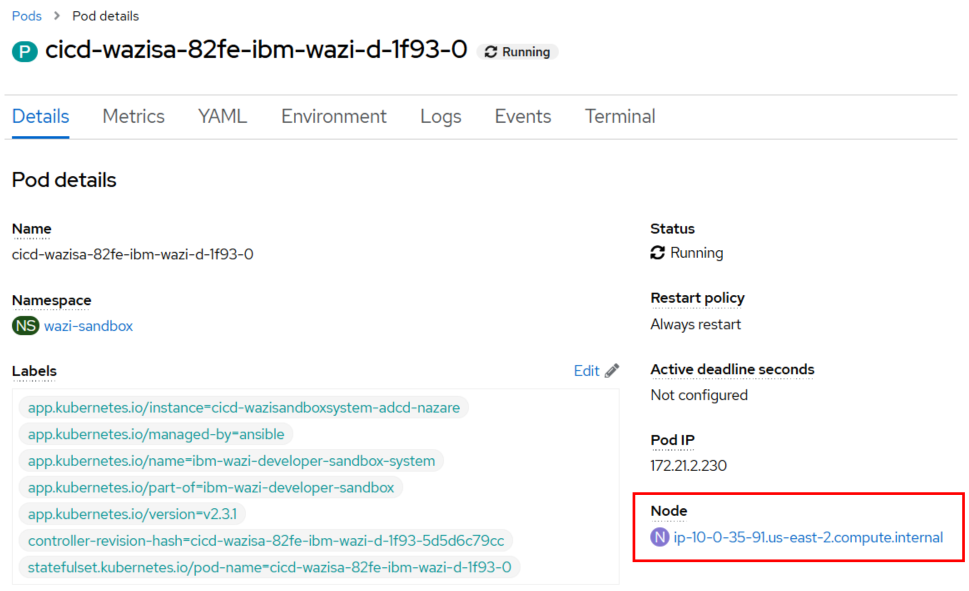
2.2.2.2 Generating keys to facilitate integration into the pipeline

To facilitate the integration of the Z Sandbox system to the pipeline process, we will now enable the login to the Z system for the IBMUSER user (which we plan to use as our build user) through a private key. Using any Linux machine, generate SSH private keys using the *ssh-keygen* command:

```
ssh-keygen -t rsa -b 4096
```

[6] zAppBuild GitHub repository: https://github.com/IBM/dbb-zappbuild
[7] DBB Utilities GitHub repository: https://github.com/IBM/dbb

The generated keys now need to be sent to the Z sandbox system and accepted for the IBMUSER user. As the connection is performed with SSH, the session must be established against the OpenShift worker node that runs the Wazi Sandbox system. This information can be retrieved from the OpenShift console, while looking at the pods for the Wazi Sandbox project. Here is an example showing such information:



Pods > Pod details
P cicd-wazisa-82fe-ibm-wazi-d-1f93-0 Running
Details Metrics YAML Environment Logs Events Terminal
Edit

Pod details

Name
cicd-wazisa-82fe-ibm-wazi-d-1f93-0

Namespace
NS wazi-sandbox

Labels
app.kubernetes.io/instance=cicd-wazisandboxsystem-adcd-nazare
app.kubernetes.io/managed-by=ansible
app.kubernetes.io/name=ibm-wazi-developer-sandbox-system
app.kubernetes.io/part-of=ibm-wazi-developer-sandbox
app.kubernetes.io/version=v2.3.1
controller-revision-hash=cicd-wazisa-82fe-ibm-wazi-d-1f93-5d5d6c79cc
statefulset.kubernetes.io/pod-name=cicd-wazisa-82fe-ibm-wazi-d-1f93-0

Status
Running

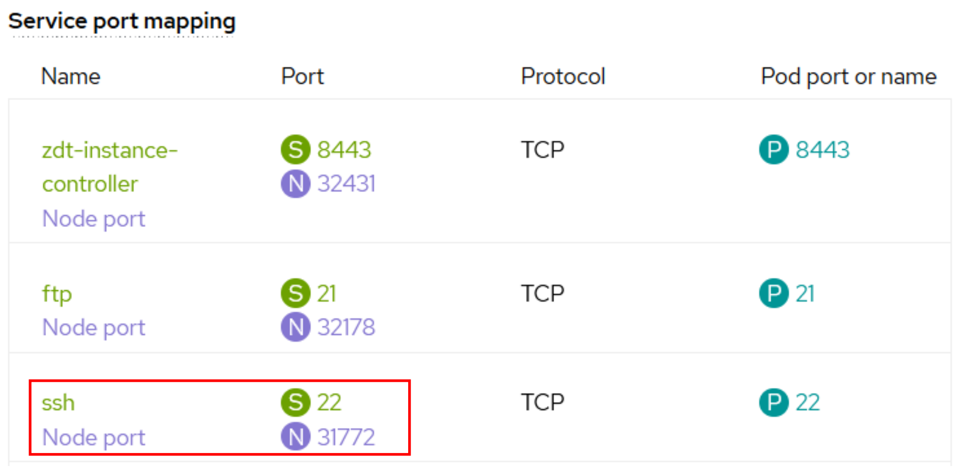
Restart policy
Always restart

Active deadline seconds
Not configured

Pod IP
172.21.2.230

Node
N ip-10-0-35-91.us-east-2.compute.internal

The Wazi Sandbox image is running on node *ip-10-0-35-91.us-east-2.compute.internal*, as shown on the bottom-right corner of the above image. This information will be used to set up the SSH connection, while the SSH port used by the Wazi Sandbox system can be retrieved from the node port configuration:



Service port mapping

Name	Port	Protocol	Pod port or name
zdt-instance-controller Node port	S 8443 N 32431	TCP	P 8443
ftp Node port	S 21 N 32178	TCP	P 21
ssh Node port	S 22 N 31772	TCP	P 22

The port exposed by Wazi Sandbox to connect with SSH to the Z system is 31772.

With the following command, the keys are registered for the IBMUSER on the Z Sandbox (assuming the Sandbox system is reachable from the Linux machine where the keys were created, and the SSH server of the Z system is mapped to 31772, as defined in the *NodePort* configuration):

```
ssh-copy-id ibmuser@ip-10-0-35-91.us-east-2.compute.internal -p 31772
```

After providing the password for IBMUSER and logging out, you should now be able to login again as IBMUSER without being prompted for the password.

2.2.3 Setting up credentials for cloning application repositories on Wazi Sandbox

It is assumed that the Z application for which the pipeline will be built is already hosted on a Git repository, and available and accessible on the Cloud platform you are using.

In this demonstration, the Z source files are hosted on the AWS Cloud platform, via the *AWS CodeCommit* service. This service is providing a Git-based server, where repositories can be created, and files managed. Alternative Git servers can be used as well, depending on the Cloud platform you are using.

Before cloning your source files to the Z Sandbox system, setting up the credentials to access the Git repository is required. When using an AWS CodeCommit repository, this page[8] provides information about creating repositories, while this page[9] explains how to configure credentials for Git clients.

In our case, we chose to use HTTPS communication, although SSH was also a valid option too. When creating the HTTPS Credentials for Git, the generated username and password need to be stored as they will be used to configure the Git connection when cloning the AWS CodeCommit repository.

To facilitate the clone operation with Git on Z, the credential helper with Git on Z will be configured to use a local store. This configuration is enabled by issuing the following command:

```
git config --global credential.helper 'store --file ~/.my-credentials'
```

The credentials will then be stored in the file pointed by the *--file* parameter, the first time the password is requested through the Git prompt. The next step is to clone your application repository on USS with the Git Client, using this command (to be adapted for your needs):

```
git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/CBSA
```

Where *us-east-2* is the AWS region that hosts your AWS CodeCommit server, and *CBSA* is the Git repository you are cloning. After running this first clone, the next `git fetch` or `git pull` commands shouldn't prompt for the username and password.

2.3 Setting up Wazi for Dev Spaces and Wazi Analyze

Wazi for Dev Spaces provides developers with a Web-based development environment, already configured with facilities to code, debug and build Z application programs. For each developer, a namespace in OpenShift is created, with its attached storage space, in which the user can create workspaces. Typically, a workspace represents an application the developer is working on.

For the next phase, it is assumed the Wazi for Dev Spaces Operator is already installed and configured, for developers to start using it. This documentation[10] explains how to deploy and configure Wazi for Dev Spaces.

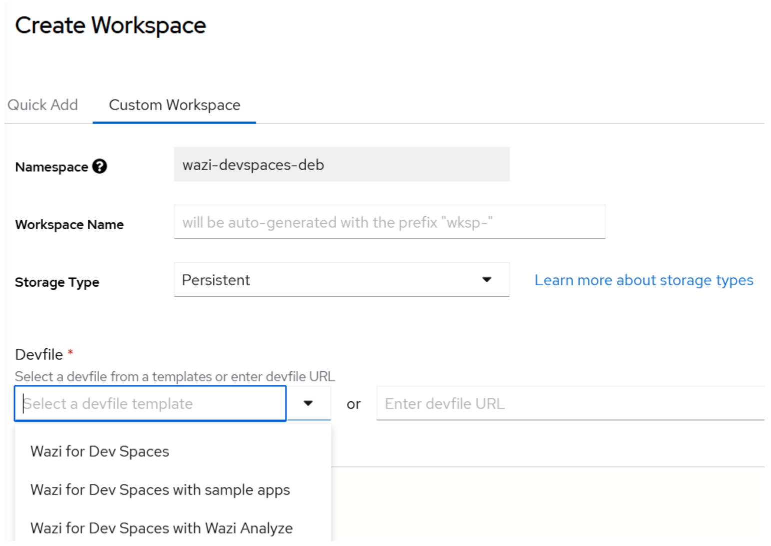
[8] AWS CodeCommit User Guide: https://docs.aws.amazon.com/codecommit/latest/userguide/getting-started.html
[9] AWS CodeCommit Credentials: https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-gc.html
[10] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=code-option-1-developing-wazi-dev-spaces

2.3.1 Configuring Wazi for Dev Spaces

The IBM Documentation website contains information and tutorials[11] showing how to setup Wazi for Dev Spaces. The following sections provide additional insights and examples of configuration that were used in our infrastructure.

2.3.1.1 Creating the workspace in Wazi for Dev Spaces

The first step of the configuration when using Wazi for Dev Spaces starts with the appropriate *Devfile*. This YAML file describes the configuration of the workspace to be created, with, among other information, the link to the Git repository that will be cloned to the workspace. Users can choose to create a *Custom Workspace* to customize their workspace. It is recommended to start from a template:


Create Workspace

Quick Add | Custom Workspace

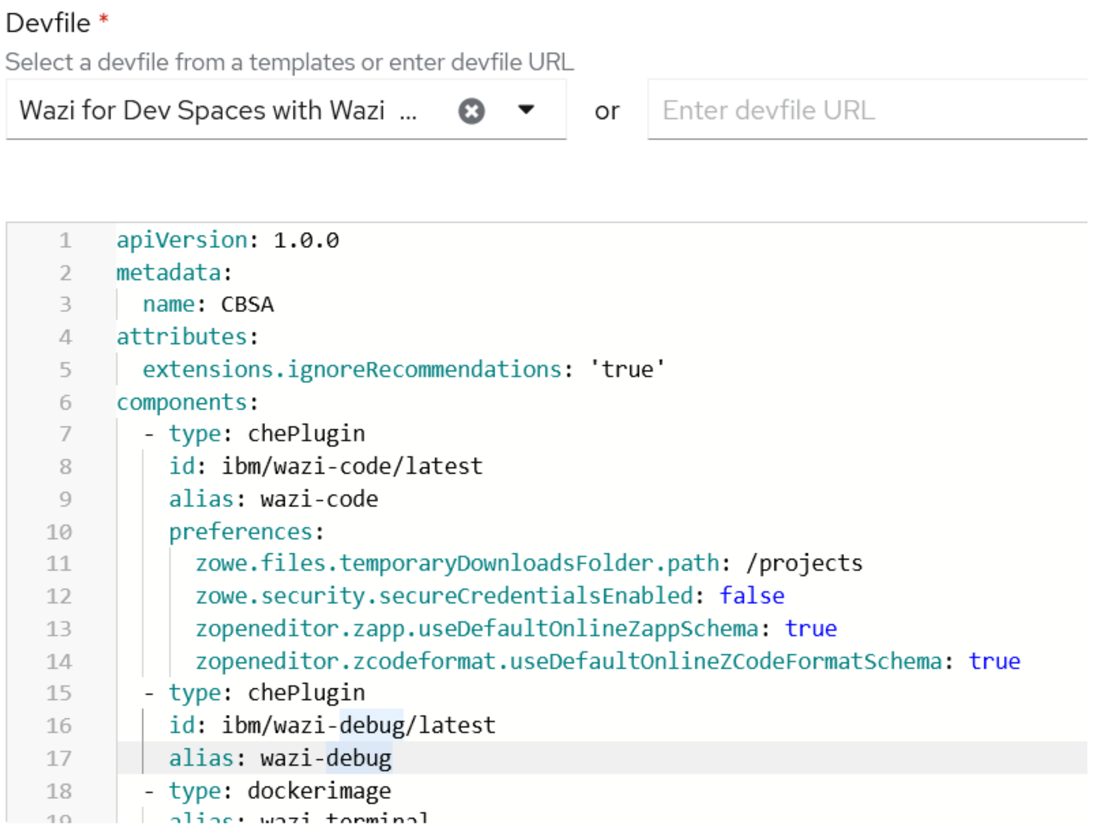
Namespace: wazi-devspaces-deb
Workspace Name: will be auto-generated with the prefix "wksp-"
Storage Type: Persistent — Learn more about storage types

Devfile *
Select a devfile from a templates or enter devfile URL
Select a devfile template — or — Enter devfile URL

- Wazi for Dev Spaces
- Wazi for Dev Spaces with sample apps
- Wazi for Dev Spaces with Wazi Analyze

In our setup, we chose to start with the *Wazi for Dev Spaces with Analyze* template, as we plan to use Wazi for Dev Spaces with Wazi Analyze as a sidecar. In order to set the configuration up before working with our Git-based project, let's first create a workspace that doesn't reference any Git repository yet, meaning the *projects* section of the YAML file should be deleted:

[11] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=developing-tutorial-creating-getting-started-your-workspace



Devfile *

Select a devfile from a templates or enter devfile URL

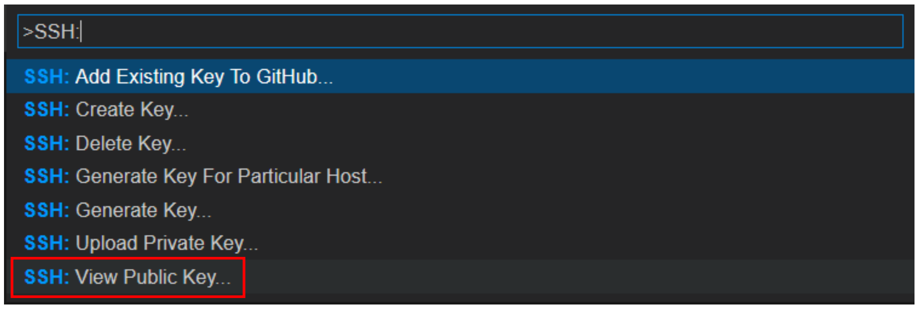
Wazi for Dev Spaces with Wazi ... or Enter devfile URL

```
apiVersion: 1.0.0
metadata:
  name: CBSA
attributes:
  extensions.ignoreRecommendations: 'true'
components:
  - type: chePlugin
    id: ibm/wazi-code/latest
    alias: wazi-code
    preferences:
      zowe.files.temporaryDownloadsFolder.path: /projects
      zowe.security.secureCredentialsEnabled: false
      zopeneditor.zapp.useDefaultOnlineZappSchema: true
      zopeneditor.zcodeformat.useDefaultOnlineZCodeFormatSchema: true
  - type: chePlugin
    id: ibm/wazi-debug/latest
    alias: wazi-debug
  - type: dockerimage
```

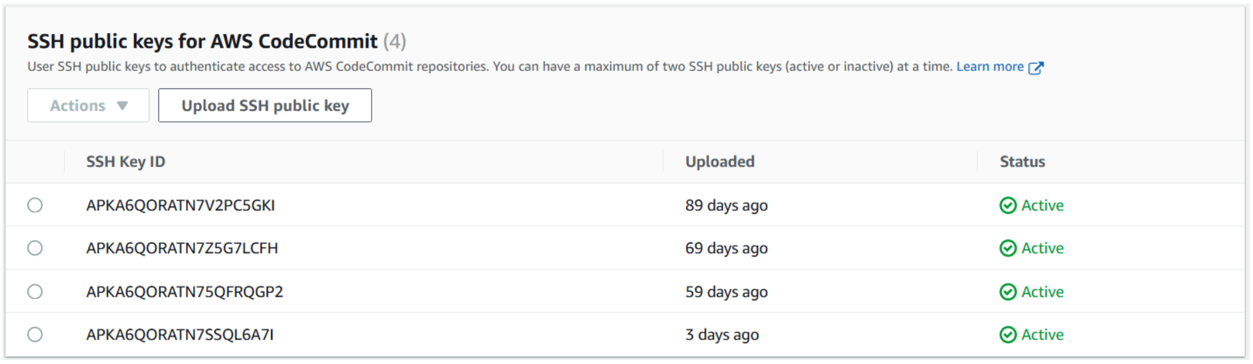
The workspace should then be active after a few minutes, depending on the network bandwidth, as container images are downloaded for the first time.

Wazi for Dev Spaces comes with a default private key, that is generated when the workspace is created. This key can be used to clone the Git repository that contains your application's sources. Typically, you would need to register that key against the Security Manager provided by your Cloud platform.

In Wazi for Dev Spaces, use the command *SSH: View Public Key...*, to retrieve the key that was automatically generated.


>SSH:
SSH: Add Existing Key To GitHub...
SSH: Create Key...
SSH: Delete Key...
SSH: Generate Key For Particular Host...
SSH: Generate Key...
SSH: Upload Private Key...
SSH: View Public Key...

The key is displayed in Wazi for DevSpaces and you can now upload this key to your account security profile. As an example, on the AWS platform, you want to use the *IAM Management Console* to define these keys for each user. On the AWS platform, this is done through the panel shown below, where you would upload your SSH public key:

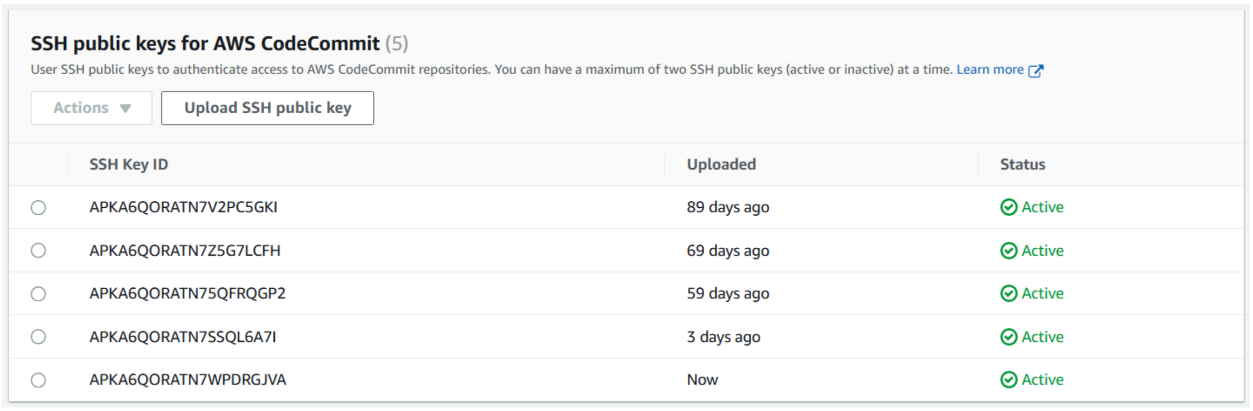


SSH public keys for AWS CodeCommit (4)

User SSH public keys to authenticate access to AWS CodeCommit repositories. You can have a maximum of two SSH public keys (active or inactive) at a time. Learn more

	SSH Key ID	Uploaded	Status
○	APKA6QORATN7V2PC5GKI	89 days ago	✓ Active
○	APKA6QORATN7Z5G7LCFH	69 days ago	✓ Active
○	APKA6QORATN75QFRQGP2	59 days ago	✓ Active
○	APKA6QORATN7SSQL6A7I	3 days ago	✓ Active

After uploading the key, a new entry is displayed in this table:

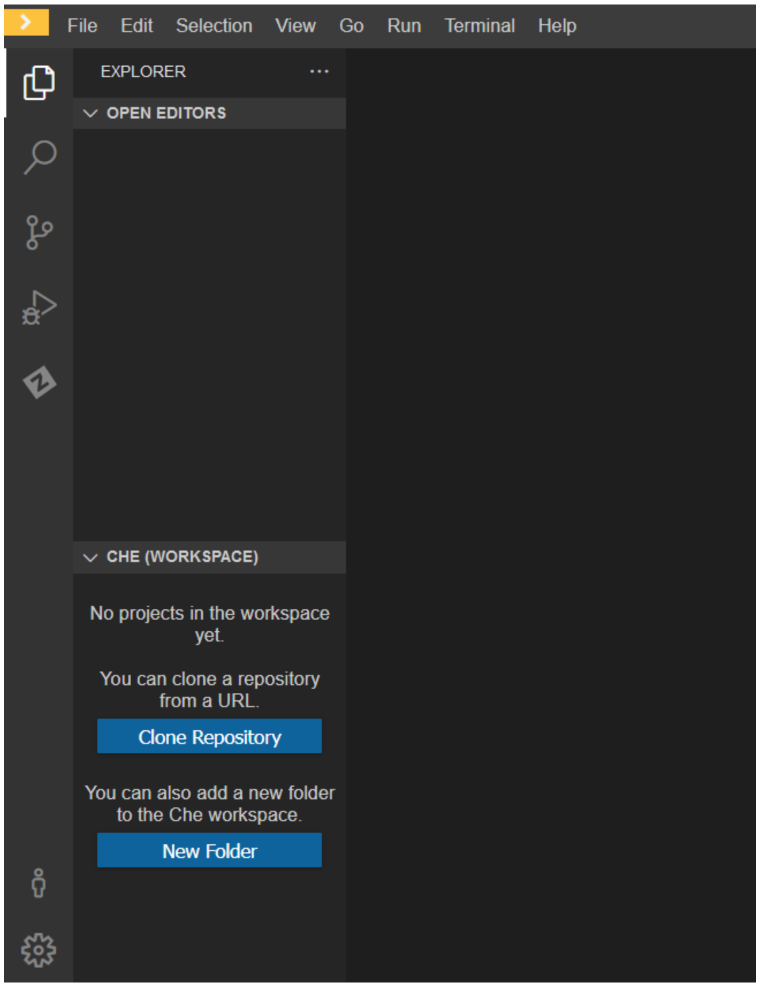


SSH public keys for AWS CodeCommit (5)

User SSH public keys to authenticate access to AWS CodeCommit repositories. You can have a maximum of two SSH public keys (active or inactive) at a time. Learn more

	SSH Key ID	Uploaded	Status
○	APKA6QORATN7V2PC5GKI	89 days ago	✓ Active
○	APKA6QORATN7Z5G7LCFH	69 days ago	✓ Active
○	APKA6QORATN75QFRQGP2	59 days ago	✓ Active
○	APKA6QORATN7SSQL6A7I	3 days ago	✓ Active
○	APKA6QORATN7WPDRGJVA	Now	✓ Active

This last SSH Key ID entry will be used as the username that we will use to clone our Git repository to our workspace, using SSH. As no repository is yet to be defined in Wazi for Dev Spaces, you will get an option to clone a Git repository, on the Explorer page of Wazi for Dev Spaces:

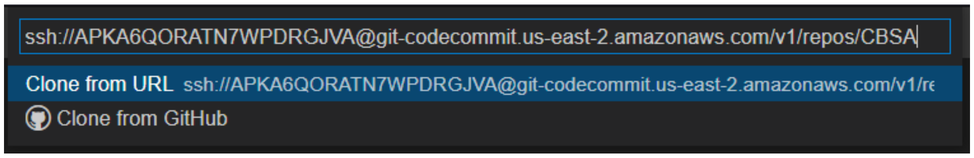


No projects in the workspace yet.

You can clone a repository from a URL.

You can also add a new folder to the Che workspace.

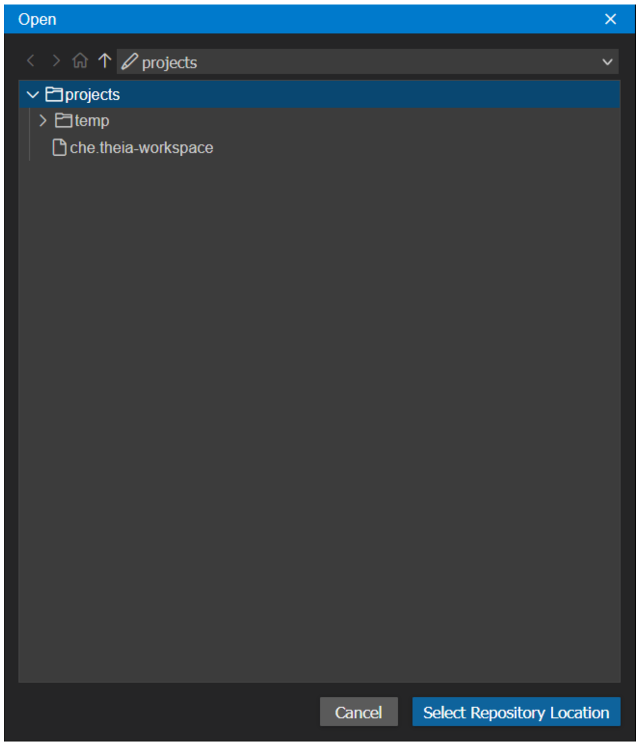
You can now press the *Clone Repository* button to clone the repository to your local workspace. You need to specify the URL to your Git repository and the credentials provided by your Cloud platform:



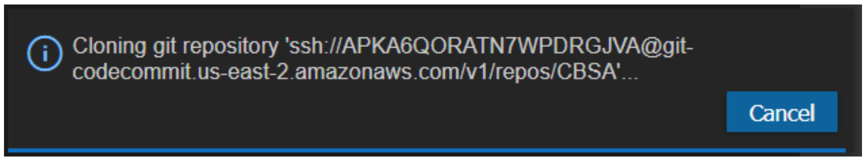
```
ssh://APKA6QORATN7WPDRGJVA@git-codecommit.us-east-2.amazonaws.com/v1/repos/CBSA
```

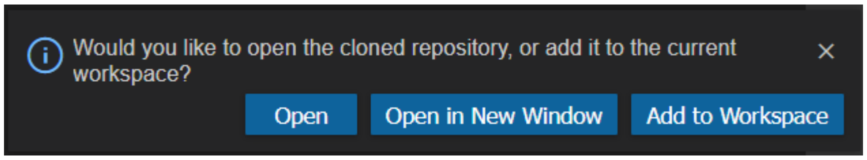
Clone from URL ssh://APKA6QORATN7WPDRGJVA@git-codecommit.us-east-2.amazonaws.com/v1/re
Clone from GitHub

You are then prompted to specify where to clone the repository. It is advised to select the */projects* folder:

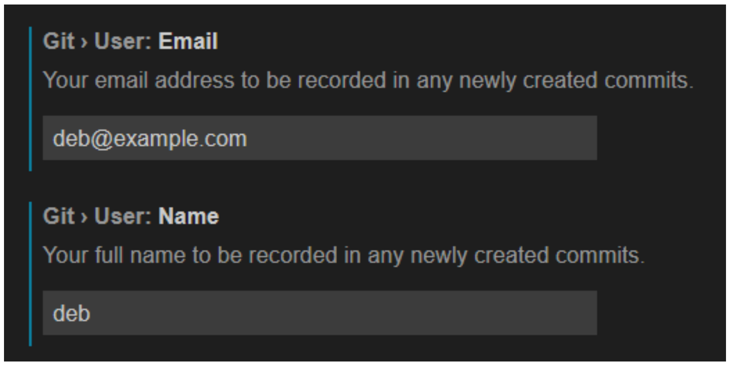


Wazi for Dev Spaces is now cloning the Git repository and asks you to open it. You can open it in a new window or simply add it to the current workspace. In our case, we chose the *Add to Workspace* option:


Cloning git repository 'ssh://APKA6QORATN7WPDRGJVA@git-codecommit.us-east-2.amazonaws.com/v1/repos/CBSA'...


Would you like to open the cloned repository, or add it to the current workspace?

To finalize the configuration of the Git repository, the name and the email of the user must be defined for Git to allow pushing content to the Git repository. These properties can be defined in the Preferences of Wazi for DevSpaces. You can search for *Git user* in the list of settings to filter out propositions. In the list, you should be able to find the following properties and set them according to your profile:


Git › User: Email
Your email address to be recorded in any newly created commits.
deb@example.com

Git › User: Name
Your full name to be recorded in any newly created commits.
deb

The workspace can now be stopped to edit its Devfile, which should actually be updated to include the Git repository that was cloned. However, in some cases, the Devfile is not correctly updated and the *projects* section of the Devfile YAML file can then be customized, to point to the Git repository containing your application's source files:

```
projects:
  - name: CBSA
    source:
      location: 'ssh://APKA6QORATN7Z5G7LCFH@ git-codecommit.us-east-
2.amazonaws.com/v1/repos/CBSA '
      type: git
      branch: Development
```

In this setup, we specified to connect using SSH, and we provided the username to be used as part of the URL of our Git repository.

The next configuration step is about customizing the properties for the IBM Z Open Editor plugin installed in Wazi for Dev Spaces. To resolve local dependencies, for instance, with copybooks, property groups must be set up according to the structure of your Git repository. With our application's repository, copybooks are located in the *copylib* folder of the *CBSA* subfolder. As our project is cloned to the */project/CBSA* folder in our workspace, the absolute path to our copybooks is */projects/CBSA/CBSA/copylib*. This path must be specified in the property group, as shown below. Wazi for Dev Spaces is leveraging ZAPP files for property groups, so this entry should be added to the ZAPP file. If no ZAPP file exists yet for your repository, you must create one and customize it according to your needs. One sample can be found in the IBM Z Open Editor GitHub repository[12].

[12] Z Open Editor sample ZAPP file: https://github.com/IBM/zopeneditor-sample/blob/main/zapp.yaml

The following snippet shows how the property group for Cobol copybooks was defined:

```yaml
name: cbsa
description: ZAPP File for CBSA application
version: 3.0.0
author:
  name: IBM CORPORATION

propertyGroups:
  - name: cobol-copybooks
    language: cobol
    libraries:
      - name: syslib
        type: local
        locations:
          - "**/copylib"
```

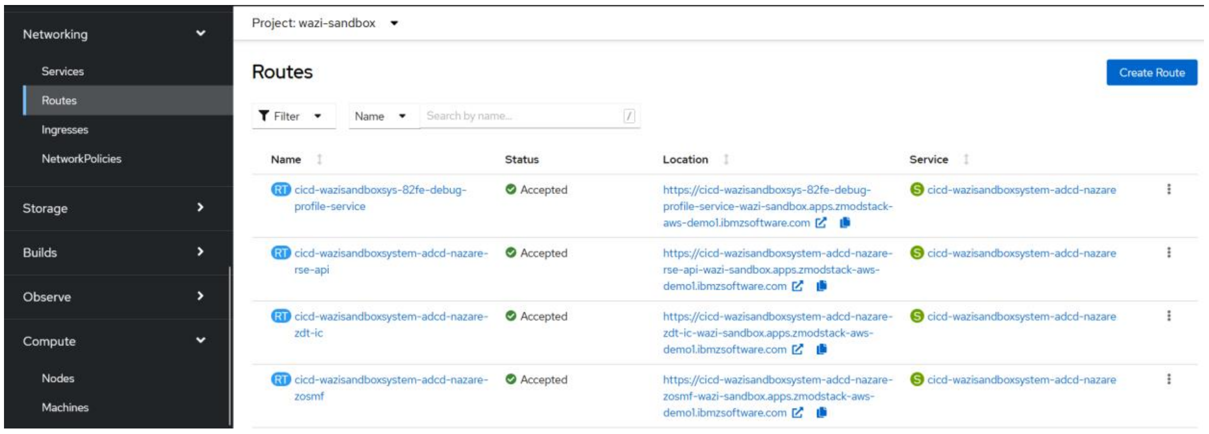
It is also possible to specify libraries that are available on a remote Z system.

Additional documentation about the ZAPP file can be found in the IBM Z Open Editor GitHub repository[13].

2.3.1.2 Connectivity setup to Wazi Sandbox

The next configuration phase consists of establishing the connectivity to the Wazi Sandbox Z system. As Wazi for Dev Spaces comes with Zowe Explorer, ZOWE CLI and its RSE plugin installed, we will use this facility to connect to the Sandbox system, which will be leveraged to perform User Builds.

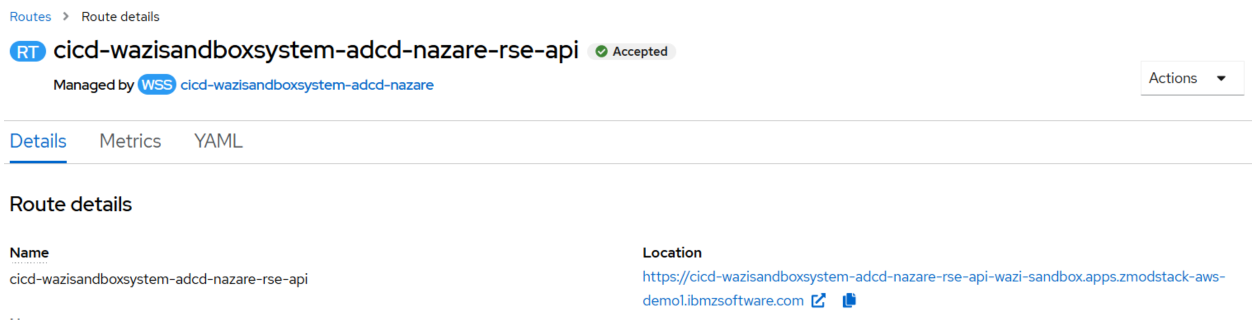
We start by identifying the RSEAPI endpoint exposed by our Network Configuration in OpenShift Cluster Platform. For the Wazi Sandbox project, select the Routes entry in the Networking section, in the Administrator view:



Name	Status	Location	Service
cicd-wazisandboxsys-82fe-debug-profile-service	Accepted	https://cicd-wazisandboxsys-82fe-debug-profile-service-wazi-sandbox.apps.zmodstack-aws-demo1.ibmzsoftware.com	cicd-wazisandboxsystem-adcd-nazare
cicd-wazisandboxsystem-adcd-nazare-rse-api	Accepted	https://cicd-wazisandboxsystem-adcd-nazare-rse-api-wazi-sandbox.apps.zmodstack-aws-demo1.ibmzsoftware.com	cicd-wazisandboxsystem-adcd-nazare
cicd-wazisandboxsystem-adcd-nazare-zdt-ic	Accepted	https://cicd-wazisandboxsystem-adcd-nazare-zdt-ic-wazi-sandbox.apps.zmodstack-aws-demo1.ibmzsoftware.com	cicd-wazisandboxsystem-adcd-nazare
cicd-wazisandboxsystem-adcd-nazare-zosmf	Accepted	https://cicd-wazisandboxsystem-adcd-nazare-zosmf-wazi-sandbox.apps.zmodstack-aws-demo1.ibmzsoftware.com	cicd-wazisandboxsystem-adcd-nazare

[13] Z Open Editor ZAPP File: https://ibm.github.io/zopeneditor-about/Docs/zapp.html#zapp-use-cases

This view lists the routes that are exposed with HTTP/HTTPS protocols. One of the routes should be related to the RSEAPI service:


Routes > Route details

RT cicd-wazisandboxsystem-adcd-nazare-rse-api Accepted

Managed by WSS cicd-wazisandboxsystem-adcd-nazare

Actions

Details Metrics YAML

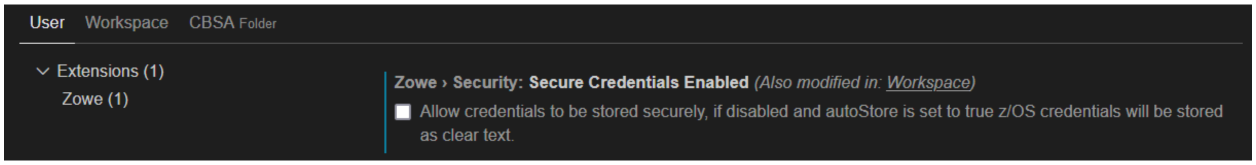
Route details

Name
cicd-wazisandboxsystem-adcd-nazare-rse-api

Location
https://cicd-wazisandboxsystem-adcd-nazare-rse-api-wazi-sandbox.apps.zmodstack-aws-demo1.ibmzsoftware.com

Copy the location URL, as it will be needed for the definition of the RSE profile in Zowe.

The next step is to create an RSE profile. However, Secure Credentials Storage must first be disabled, as this facility cannot be used in the current version of Wazi for Dev Spaces since it is executing in a container. To disable the Secure Credential Storage mechanism, open the settings and search for *Secure Credentials*. The following option should then be unchecked:

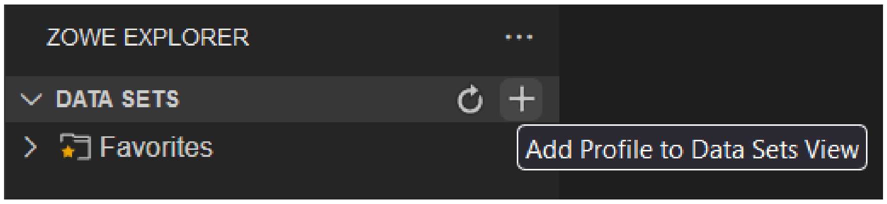

User Workspace CBSA Folder

Extensions (1)
Zowe (1)

Zowe › Security: **Secure Credentials Enabled** *(Also modified in: Workspace)*

☐ Allow credentials to be stored securely, if disabled and autoStore is set to true z/OS credentials will be stored as clear text.

Then navigate to the Zowe Explorer view, where you can create a profile. Click the *+* icon, to create a profile.



Select the *Create a new Team Configuration File* option, and choose the *Global* option for the location of the profile. In the generated *zowe.config.file* file, navigate to the *rse* entry and edit it to add the *host* property. Extract the hostname from the URL representing the route exposed for the RSEAPI service and provide this value - you also want to accept self-signed certificates by setting *rejectUnauthorized* to false.

```
"host": "cicd-wazisandboxsystem-adcd-nazare-rse-api-wazi-sandbox.apps.xxxxxxx",
"rejectUnauthorized": false
```

Also, check that the *autoStore* property at the end of this file is set to false. You are then prompted to restart Zowe Explorer: go ahead and do so, and the *rse* profile should now be recognized as the default profile. You can now try to retrieve information from the Z system. In Zowe Explorer, you can search for datasets, using *IBMUSER.*** as a filter. After providing the IBMUSER user and the password, you should be able to retrieve the list of datasets per the request.

The next phase is to set up the User Build configuration. Again, edit the ZAPP file to add the workspace's definitions for the User Build (customize the following snippet to meet your needs):

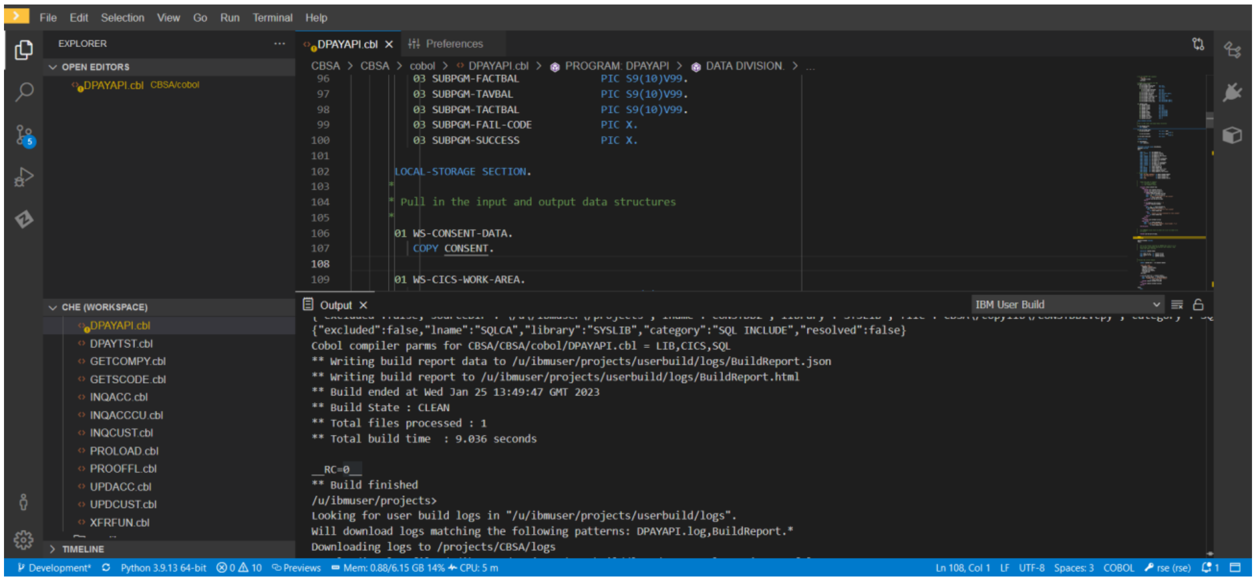
```
profiles:
  - name: dbb-userbuild
    type: dbb
    settings:
      application: CBSA
      command: "$DBB_HOME/bin/groovyz -DBB_DAEMON_HOST 127.0.0.1 -
DBB_DAEMON_PORT 7380 "
      buildScriptPath: "/var/work/dbb-zappbuild/build.groovy"
      buildScriptArgs:
        - "--userBuild"
        - "--workspace ${zopeneditor.userbuild.userSettings.dbbWorkspace}"
        - "--application ${application}"
        - "--hlq ${zopeneditor.userbuild.userSettings.dbbHlq}"
        - "--outDir ${zopeneditor.userbuild.userSettings.dbbLogDir}"
        - "--verbose"
      additionalDependencies:
        - ${application}"/application-conf
        - "zapp*"
      logFilePatterns:
        - "${buildFile.basename}.log"
        - "BuildReport.*"
```

Additionally, the following definitions must be added in the User's settings (in the *settings.json* file):

```
   "zopeneditor.userbuild.userSettings": {
      "dbbWorkspace": "/u/ibmuser/projects",
      "dbbHlq": "IBMUSER.CBSA.UB",
      "dbbLogDir": "/u/ibmuser/projects/userbuild/logs"
   },
```

Wazi for DevSpaces also provides a built-in helper to create your ZAPP file, accessible by pressing the ctrl-space key combination. This facility lists the keywords available for creating the content of this file.

You should now be able to request a User Build with Wazi for Dev Spaces, using the Wazi Sandbox Z system. The first step to perform is called *Run setup for IBM User Build*, to initialize the build workspace on Z and upload the files required for a correct build with zAppBuild (typically, the property files of the application, which are stored in the *application-conf* subfolder). When the User Build is then triggered, the User Build feature will find the necessary dependencies, upload them to the target Z system where the build is performed and will execute the command defined in the ZAPP file. The output of the build should appear in the console in Wazi for Dev Spaces:


File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DPAYAPI.cbl CBSA/cobol
CHE (WORKSPACE)
DPAYAPI.cbl
DPAYTST.cbl
GETCOMPY.cbl
GETSCODE.cbl
INQACC.cbl
INQACCCU.cbl
INQCUST.cbl
PROLOAD.cbl
PROOFFL.cbl
UPDACC.cbl
UPDCUST.cbl
XFRFUN.cbl
TIMELINE
DPAYAPI.cbl
Preferences
CBSA > CBSA > cobol > DPAYAPI.cbl > PROGRAM: DPAYAPI > DATA DIVISION. > ...
Output
IBM User Build
Development* Python 3.9.13 64-bit ⊗ 0 ⚠ 10 Previews Mem: 0.88/6.15 GB 14% CPU: 5 m Ln 108, Col 1 LF UTF-8 Spaces: 3 COBOL rse (rse) 1

```
          03 SUBPGM-FACTBAL             PIC S9(10)V99.
          03 SUBPGM-TAVBAL              PIC S9(10)V99.
          03 SUBPGM-TACTBAL             PIC S9(10)V99.
          03 SUBPGM-FAIL-CODE           PIC X.
          03 SUBPGM-SUCCESS             PIC X.

       LOCAL-STORAGE SECTION.
      *
      * Pull in the input and output data structures
      *
       01 WS-CONSENT-DATA.
          COPY CONSENT.

       01 WS-CICS-WORK-AREA.
```

```
{"excluded":false,"lname":"SQLCA","library":"SYSLIB","category":"SQL INCLUDE","resolved":false}
Cobol compiler parms for CBSA/CBSA/cobol/DPAYAPI.cbl = LIB,CICS,SQL
** Writing build report data to /u/ibmuser/projects/userbuild/logs/BuildReport.json
** Writing build report to /u/ibmuser/projects/userbuild/logs/BuildReport.html
** Build ended at Wed Jan 25 13:49:47 GMT 2023
** Build State : CLEAN
** Total files processed : 1
** Total build time  : 9.036 seconds

__RC=0__
** Build finished
/u/ibmuser/projects>
Looking for user build logs in "/u/ibmuser/projects/userbuild/logs".
Will download logs matching the following patterns: DPAYAPI.log,BuildReport.*
Downloading logs to /projects/CBSA/logs
```

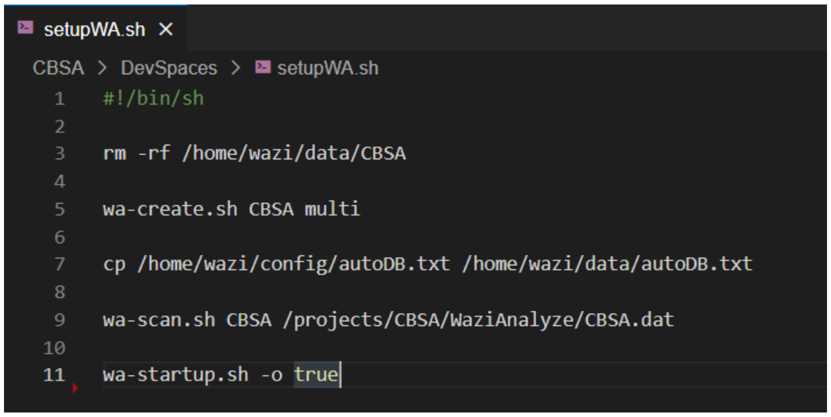
2.3.2 Setting up Wazi Analyze

Starting with Wazi for Dev Spaces 2.3, Wazi Analyze is available as a sidecar component when workspaces are deployed. If the workspace you configured contains the Wazi Analyze component, you can leverage this facility to scan for the source code stored in the Git repository that you cloned. The results of the scan are then available in a Web-based interface in your workspace's environment, directly accessible through endpoints in Wazi for Dev Spaces. A tutorial on the IBM Documentation website[14] shows how to integrate Wazi Analyze in the developer's tasks.

[14] https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.4?topic=tutorials-tutorial-analyzing-sample-source-files-wazi-dev-spaces

For our setup, we took a slightly different approach, where we provided a shell script that performs all the setup work, like creating the project in Wazi Analyze, performing the scan and starting the necessary services up. These actions are grouped into a shell script that is stored in the same Git repository that contains the application's source code. Stored in a specific subfolder called *DevSpaces,* the following script performs the necessary Wazi Analyze tasks automatically (it should be customized for your setup and application):

```
#!/bin/sh

rm -rf /home/wazi/data/CBSA

wa-create.sh CBSA multi

cp /home/wazi/config/autoDB.txt /home/wazi/data/autoDB.txt

wa-scan.sh CBSA /projects/CBSA/WaziAnalyze/CBSA.dat

wa-startup.sh -o true
```



```
#!/bin/sh

rm -rf /home/wazi/data/CBSA

wa-create.sh CBSA multi

cp /home/wazi/config/autoDB.txt /home/wazi/data/autoDB.txt

wa-scan.sh CBSA /projects/CBSA/WaziAnalyze/CBSA.dat

wa-startup.sh -o true
```

The first action that this script will perform is to perform some cleanup in the Wazi Analyze container, in case previous scans were already performed. Scan data is stored in the */home/wazi/data* folder, each application being represented as a project in a subfolder (CBSA in our case). The *wa-create.sh* script helps in creating a project in Wazi Analyze, by providing the name and type of the project (*multi* means multiple types of artifacts to scan). The copy command is used to preserve the existing Database configurations in Wazi Analyze (it comes with a default application, like GenAppC) and make them available when Wazi Analyze services are starting.

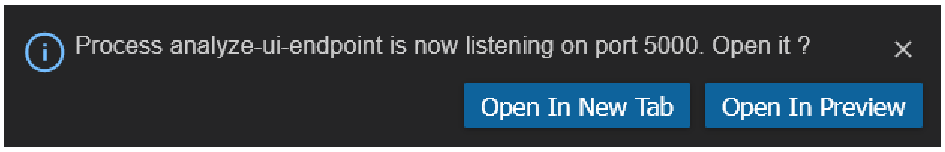
The *wa-scan.sh* action asks Wazi Analyze to scan the source code for the CBSA project that was just created. The */projects/CBSA/WaziAnalyze/CBSA.dat* file is a configuration file passed along the command line, to specify scan options. This file is also part of the Git repository of the application, to facilitate its use by developers. The last command, *wa-startup.sh*, is used to start the Wazi Analyze services that are supporting the Web-based interface to browse the scan results. The *-o* flag is mandatory in our context, as the script is running in an OpenShift Cluster Platform environment. When executing this script, the user will be prompted to provide a password, that will then be used for authentication when the services are running.

To facilitate the launch of this script, a Task can be created in Wazi for Dev Spaces. The task can be defined in the Devfile, by adding this entry at the end of the YAML file:

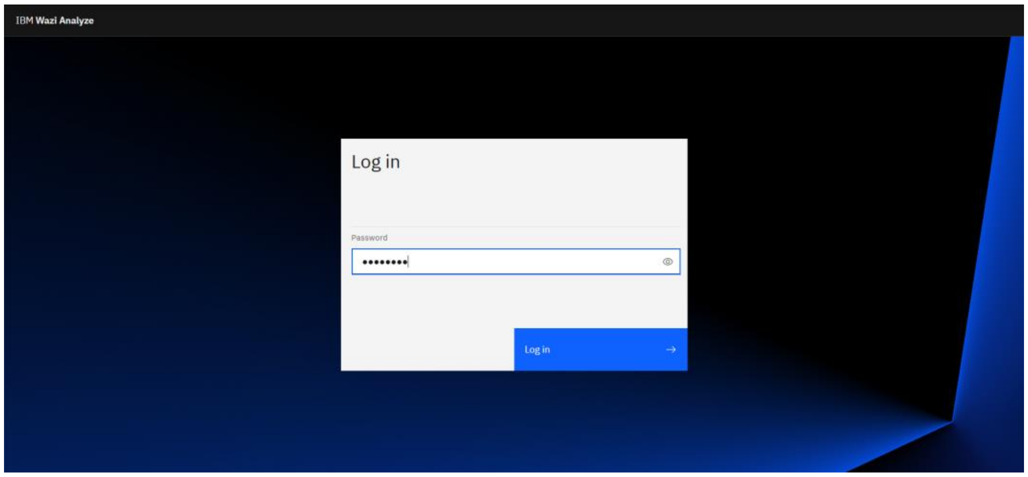
```yaml
  - name: Start Wazi Analyze
    actions:
      - workdir: '${CHE_PROJECTS_ROOT}/CBSA'
        type: exec
        command: '${CHE_PROJECTS_ROOT}/CBSA/DevSpaces/setupWA.sh'
        component: wazi-analyze
```

This way, a task called *Start Wazi Analyze* is now available through the *Terminal* → *Run Task…* menu option. When launching this task, the script is executed, creating the project in Wazi Analyze, scanning the source and launching the services. When the services are launched, the user is prompted with a password, that will then be used to log into the Wazi Analyze Web interface.

When the Wazi Analyze services are started, a popup window asks to access the Web Interface in a new tab of your Web browser:


Process analyze-ui-endpoint is now listening on port 5000. Open it ?

You should now be able to log in with the password provided during the startup of services and browse the scanned artifacts of your application:



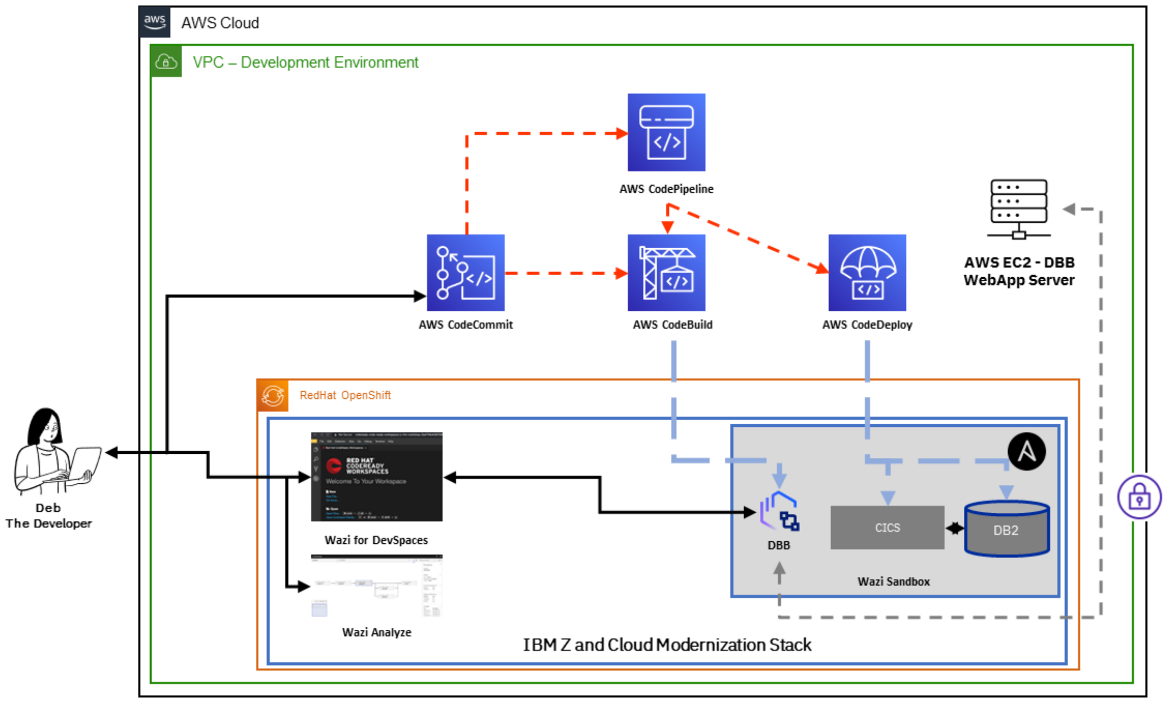
The DevSpaces workspace templates also ship with pre-configured Tasks, that can be used to create a Wazi Analyze project and scan the content of the repository to populate that project. These tasks are called *Analyze-1* and *Analyze-2*, and can be executed when needed by the developers, if the above script setup doesn't meet users' expectations.

3 Real-life example - Building a CI/CD pipeline with AWS Developer services

In this section, we will detail the main steps that should be performed to leverage the Wazi components as part of a modern CI/CD pipeline. To support this configuration, the AWS Cloud platform will be used, as it provides out-of-the-box services to implement DevOps practices. These services can also be leveraged for the mainframe applications' software delivery lifecycle.

Different options exist to deploy the Developer services in the AWS Cloud platform, but in this demonstration, the *AWS CodeCommit*[15], *AWS CodeBuild*[16], *AWS CodeDeploy*[17] and *AWS CodePipeline*[18] services will be used. Customers also have the possibility to deploy *GitHub*[19] (instead of the AWS CodeCommit service) to store the mainframe source codes, or *Jenkins*[20] (instead of AWS CodePipeline) on AWS to drive the CI/CD pipeline executions.

The following diagram shows the infrastructure that was setup to demonstrate the integration of the Wazi components on the AWS Cloud platform:



At this stage, the mainframe application source code has already been migrated to the AWS CodeCommit service, and the necessary steps were performed to allow users to work with this repository. The official

[15] https://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html
[16] https://docs.aws.amazon.com/codebuild/latest/userguide/welcome.html
[17] https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html
[18] https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html
[19] https://aws.amazon.com/solutions/implementations/github-enterprise/
[20] https://aws.amazon.com/blogs/devops/setting-up-a-ci-cd-pipeline-by-integrating-jenkins-with-aws-codebuild-and-aws-codedeploy/

documentation available on the AWS website provides the necessary information to push the source artifacts to an AWS CodeCommit repository and explains how to securely connect to the AWS CodeCommit repository from developers' workstation.

In our setup, the AWS CodeCommit repository was already used when setting up the Wazi for Dev Spaces workspace, and we already verified the connectivity between our developer environment and the AWS CodeCommit service (See section 2.3.1 Configuring Wazi for Dev Spaces).

The next section explains how to configure the AWS CodeBuild service to perform the build of the mainframe application on the Wazi Sandbox Z system.

3.1 Configuring AWS CodeBuild to build mainframe applications

AWS CodeBuild is using containers to drive build actions. In our configuration, containers make a perfect choice for more flexibility and agility, because most of the actions of the build phase are actually performed on the Wazi Sandbox system. For connectivity purposes, the SSH protocol will be used to submit build commands on the Z system.

Through the AWS CodeBuild interface, the first step is to create a new Build project. After providing a meaningful name for this project (*CBSA-Build-Sandbox* in our case), you can define the appropriate settings for your configuration. The AWS CodeBuild documentation[21] provides guidance and details the different options for the Build project.

In our setup, we chose to disable the source section, as the Git repository will be cloned from Z directly during the pipeline execution. The *Environment* section describes the selected container image to execute the actions. We chose an Ubuntu-based image with the latest software stack. Any image with an SSH client is sufficient to invoke the mainframe build actions.

[21] https://docs.aws.amazon.com/codebuild/latest/userguide/getting-started.html

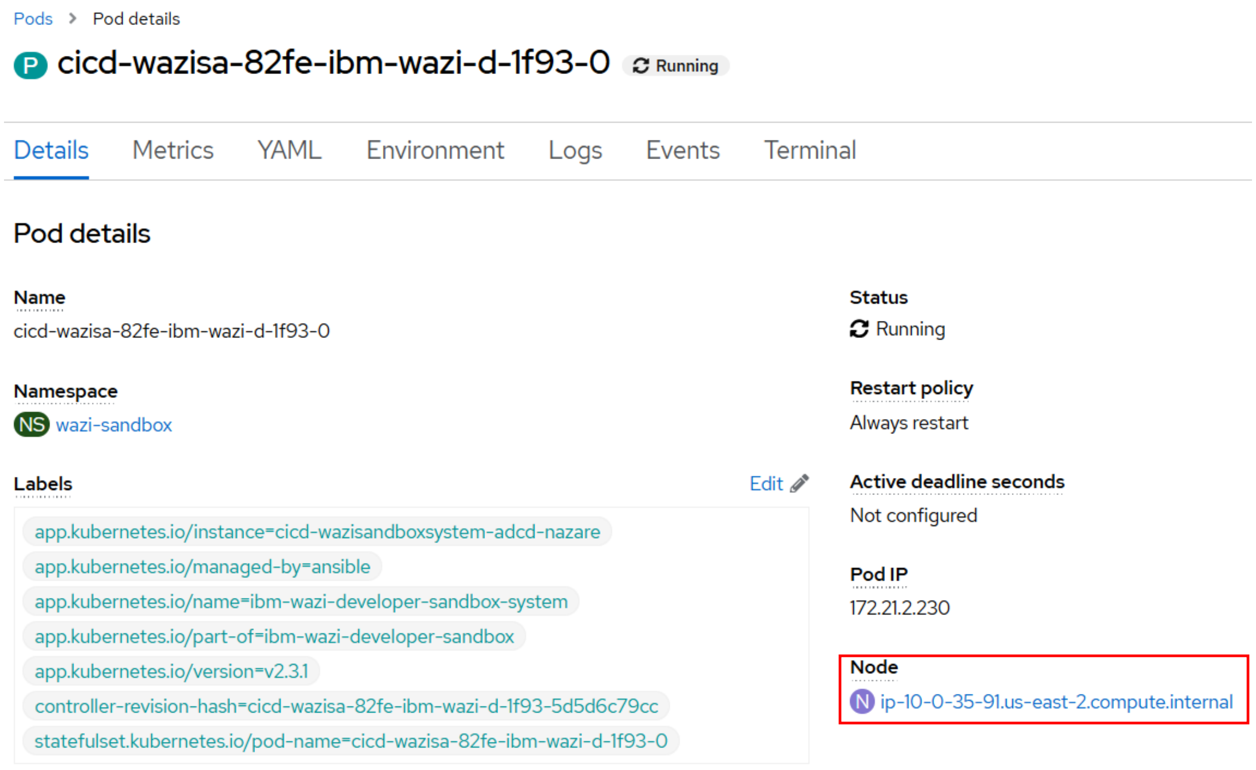
The *Buildspec* section defines the actions that will be performed during the build. This is where the commands executed on Z will be defined. Commands can be provided in a YAML format, for better readability. The following section is an example of a typical setup, as used in our configuration:

```
version: 0.2
phases:
  pre_build:
    commands:
      - aws secretsmanager get-secret-value --secret-id WaziSandboxSSHPrivateKey --query
'SecretString' --output text > /tmp/id_rsa
      - chmod 600 /tmp/id_rsa
  build:
    commands:
      - ssh -i /tmp/id_rsa ibmuser@ip-10-0-35-91.us-east-2.compute.internal -p 31772 -o
StrictHostKeyChecking=accept-new -o SendEnv=CodeCommitBranchName bash 'env;.
/u/ibmuser/.bash_profile;env;cd /var/work/pipeline/CBSA;git fetch --all; rm -rf
/var/work/temp/BUILD-OUTPUT; mkdir -p /var/work/temp/BUILD-OUTPUT;git checkout Development;
git pull --all;$DBB_HOME/bin/groovyz -DBB_DAEMON_HOST 127.0.0.1 -DBB_DAEMON_PORT 7380 -
Djava.library.path=$DBB_HOME/lib:/usr/lib/java_runtime64 /var/work/dbb-zappbuild/build.groovy
--workspace /var/work/pipeline/CBSA --hlq CBSA.DEVELOP --workDir /var/work/temp/BUILD-OUTPUT -
-application CBSA --logEncoding UTF-8 --impactBuild CBSA/application-conf/mandatoryBuild.txt;
cd /var/work/temp/BUILD-OUTPUT; for f in `find . -name "*.zunit.report.log"`; do Xalan -o
$f.xml $f /var/work/extensions/zunit2junit/AZUZ2J30.xsl; done;$DBB_HOME/bin/groovyz -
DBB_DAEMON_HOST 127.0.0.1 -DBB_DAEMON_PORT 7380 -
Djava.library.path=$DBB_HOME/lib:/usr/lib/java_runtime64
/var/work/extensions/dbb/Pipeline/PackageBuildOutputs/PackageBuildOutputs.groovy --workDir
/var/work/temp/BUILD-OUTPUT -ae -t package.tar'
  post_build:
    commands:
      - |
        mkdir -p /var/work/temp/BUILD-OUTPUT/
        cd /var/work/temp/BUILD-OUTPUT/
        echo "get /var/work/temp/BUILD-OUTPUT/package.tar" | sftp -i /tmp/id_rsa -P 31772 -o
StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        echo "get /var/work/pipeline/CBSA/AWS-Deployment/appspec.yml" | sftp -i /tmp/id_rsa -P
31772 -o StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        echo "get /var/work/pipeline/CBSA/AWS-Deployment/deployment-sandbox.sh" | sftp -i
/tmp/id_rsa -P 31772 -o StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-
2.compute.internal
        mv deployment-sandbox.sh deployment.sh
        echo "mget /var/work/temp/BUILD-OUTPUT/*.txt" | sftp -i /tmp/id_rsa -P 31772 -o
StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        echo "mget /var/work/temp/BUILD-OUTPUT/*.log" | sftp -i /tmp/id_rsa -P 31772 -o
StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        echo "mget /var/work/temp/BUILD-OUTPUT/*.html" | sftp -i /tmp/id_rsa -P 31772 -o
StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        echo "mget /var/work/temp/BUILD-OUTPUT/*.json" | sftp -i /tmp/id_rsa -P 31772 -o
StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        echo "mget /var/work/temp/BUILD-OUTPUT/*.xml" | sftp -i /tmp/id_rsa -P 31772 -o
StrictHostKeyChecking=accept-new ibmuser@ip-10-0-35-91.us-east-2.compute.internal
        for f in `find . -type f -name "*.*"`; do aws s3 cp $f s3://cbsa-
backend/${PipelineID}/; done;
reports:
  zUnit:
    files:
       - "*.xml"
    discard-paths: yes
    file-format: JunitXml
    base-directory: "/var/work/temp/BUILD-OUTPUT/"
artifacts:
  files:
     - package.tar
     - appspec.yml
     - deployment.sh
  discard-paths: yes
  base-directory: "/var/work/temp/BUILD-OUTPUT/"
```

This build specification is split into 3 different sections. The *pre_build* step prepares the environment for the execution of the build: it downloads a private key to the container using the *aws* command-line interface. This private key will be used to connect to the Wazi Sandbox system with the IBMUSER user. The private key is stored on AWS Secret Manager, and the AWS CodeBuild role was added to the capability to read from AWS Secret Manager.

The *build* section contains the commands that will be executed on z/OS through SSH. As the connection is performed with SSH, the session must be established against the OpenShift worker node that runs the Wazi Sandbox system. This information can be retrieved from the OpenShift console, by looking at the pod details for the Wazi Sandbox project.

Here is an example showing such information:


Pods > Pod details

P cicd-wazisa-82fe-ibm-wazi-d-1f93-0 Running

Details Metrics YAML Environment Logs Events Terminal

Pod details

Name
cicd-wazisa-82fe-ibm-wazi-d-1f93-0

Namespace
NS wazi-sandbox

Labels Edit
app.kubernetes.io/instance=cicd-wazisandboxsystem-adcd-nazare
app.kubernetes.io/managed-by=ansible
app.kubernetes.io/name=ibm-wazi-developer-sandbox-system
app.kubernetes.io/part-of=ibm-wazi-developer-sandbox
app.kubernetes.io/version=v2.3.1
controller-revision-hash=cicd-wazisa-82fe-ibm-wazi-d-1f93-5d5d6c79cc
statefulset.kubernetes.io/pod-name=cicd-wazisa-82fe-ibm-wazi-d-1f93-0

Status
Running

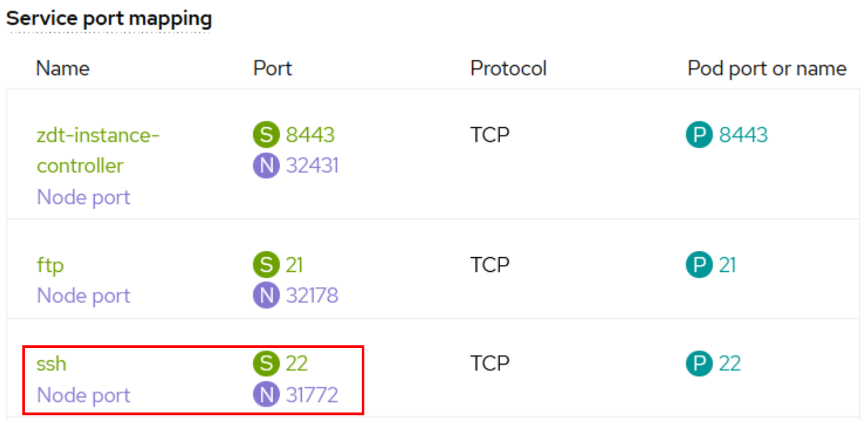
Restart policy
Always restart

Active deadline seconds
Not configured

Pod IP
172.21.2.230

Node
N ip-10-0-35-91.us-east-2.compute.internal

The Wazi Sandbox image is running on node *ip-10-0-35-91.us-east-2.compute.internal*, as shown on the bottom-right corner of the above image. This information will be used to set up the SSH connection, and the SSH port used by the Wazi Sandbox system can be retrieved from the node port configuration:


Service port mapping

Name	Port	Protocol	Pod port or name
zdt-instance-controller Node port	S 8443 N 32431	TCP	P 8443
ftp Node port	S 21 N 32178	TCP	P 21
ssh Node port	S 22 N 31772	TCP	P 22

The port exposed by Wazi Sandbox to connect with SSH to the Z system is 31772. This information will be used in the *buildspec* definition to send SSH and SFTP commands (used in the *post_build* phase).

The *build* section contains an SSH command that is chaining several actions:

- The initial commands are used to set up environment variables on z/OS,
- Then, Git commands are used to fetch and pull changes on z/OS from the Git repository,
- The first call to DBB is used to perform an impact build on the changed files,
- The second call to DBB is used to execute a script to package the build output artifacts (*PackageBuildOutputs.groovy* script) into an archive.

Depending on your cloning strategy, another option to retrieve the content of the Git repository would be to clone the repository for each pipeline execution. This decision to use either clone or fetch/pull commands also depends on the size of the Git repository, which has an impact on the execution elapse time for the pipeline.

Also, the SSH command described above is using chained commands. For convenience, you may want to use scripts that group these commands together, for a better readability and maintenance.

The last section, *post_build*, retrieves artifacts from z/OS using SFTP commands: it first retrieves the package that was just created during the build phase, along with two files that will be used during the *CodeDeploy* phase. The *appspec.yml* file will define the list of actions to perform during the deployment, while the *deployment.sh* script will drive the deployment commands. These files will be described in the next section of this document.

Also, the build log files (*.txt or *.log files), the DBB Build Report (in JSON and HTML format) and reports in XML format (produced by the zUnit feature, for instance) are retrieved using SFTP commands. The list of files that are retrieved can be customized based on users' needs. These files are then sent to an *AWS S3* bucket for archiving purposes, and to help developers understand what happened during the build phase.

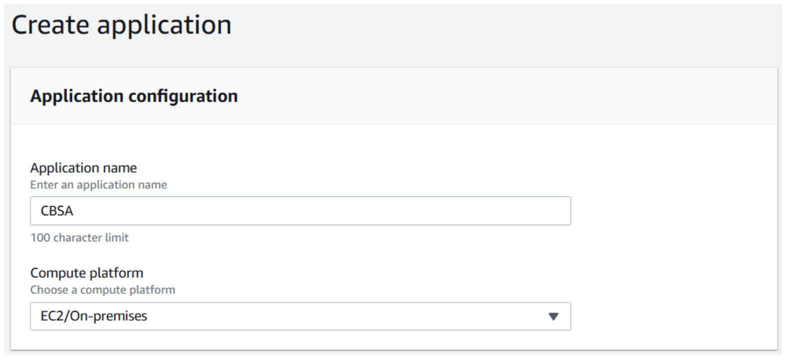
The *reports* section defines specific processing for some files: in our configuration, JUnit compatible files (generated from zUnit files) are uploaded as JUnit reports, so developers can check for Unit Testing trends on AWS through AWS CodeBuild's Report Groups feature. Finally, the *artifacts* section defines which artifacts should be packaged together into a ZIP file that will be stored on a specific AWS S3 bucket (the AWS S3 bucket to use during the pipeline be defined later in the pipeline definition, in AWS CodePipeline).

3.2 Using AWS CodeDeploy to deploy z/OS artifacts

In our setup, the next building block leveraged in the CI/CD pipeline is the AWS CodeDeploy service. The purpose of this service is to execute deployment actions.

These deployment tasks are defined in a deployment specification file, called *appspec.yml*, that must be provided in the package that contains the artifacts to deploy.

AWS CodeDeploy proposes several options to run the deployment actions: it can either use an existing image on AWS EC2, a serverless process using AWS Lambda or a container on AWS ECS. In our configuration, we chose to leverage an existing EC2 image of our infrastructure, which was already hosting the DBB 1.1.3 WebApp server. Thus, when creating the application in the AWS CodeDeploy service, the Compute platform option chosen is *EC2/on-premises*:


Create application

Application configuration

Application name
Enter an application name
CBSA
100 character limit

Compute platform
Choose a compute platform
EC2/On-premises

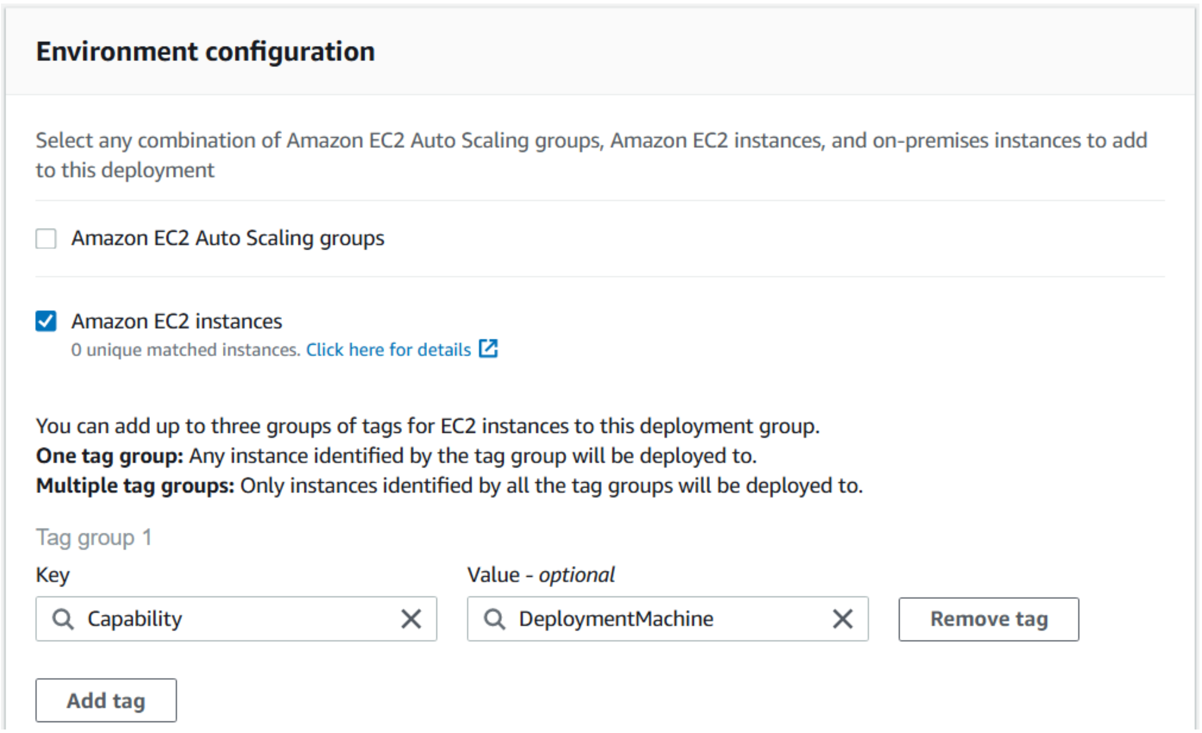
To be able to use the EC2 instance to deploy our Z artifacts, a few steps must be performed on the EC2 environment.

3.2.1 Installing and configuring AWS CodeDeploy agent

The first action is to install an AWS CodeDeploy agent. This agent will connect to the AWS infrastructure and execute scripts as defined in the deployment specification. The AWS documentation[22] provides the necessary information to install and register the agent to your AWS account. Also, to be able to deploy, the machine running the AWS CodeDeploy agent needs to be assigned an IAM that has the necessary credentials. This AWS documentation page[23] provides more information on how to setup these permissions. After assigning the IAM role to the instance, the AWS CodeDeploy agent may need to be restarted, with the `systemctl restart codedeploy-agent` command.

[22] https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations.html
[23] https://docs.aws.amazon.com/codedeploy/latest/userguide/security-iam.html

After creating the application in the AWS CodeDeploy service, the next phase is to create a deployment group, which defines the specifications for the deployment process. After providing a name for the deployment group, the main step is to define which EC2 instances are used for the deployment. Tags can be used to identify which EC2 instance should be used. We are specifying those instances with the tag *DeploymentMachine.*



Environment configuration

Select any combination of Amazon EC2 Auto Scaling groups, Amazon EC2 instances, and on-premises instances to add to this deployment

☐ Amazon EC2 Auto Scaling groups

☑ Amazon EC2 instances
0 unique matched instances. Click here for details

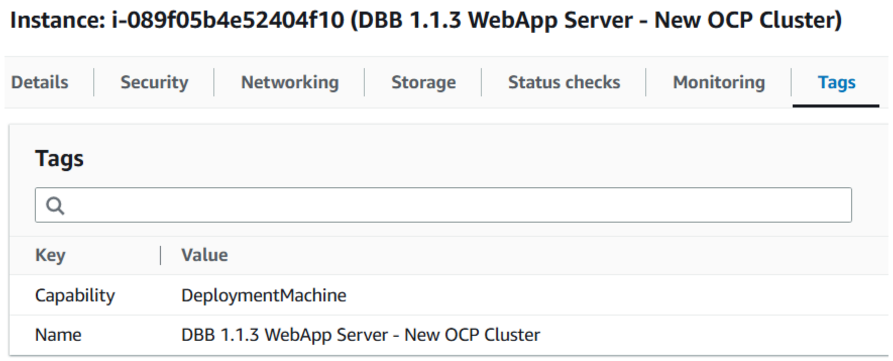
You can add up to three groups of tags for EC2 instances to this deployment group.
One tag group: Any instance identified by the tag group will be deployed to.
Multiple tag groups: Only instances identified by all the tag groups will be deployed to.

Tag group 1

Key	Value - *optional*
Capability	DeploymentMachine

All the other properties of the Deployment group are kept by default, and the Load balancing option is disabled.

We added the same tag to our EC2 instance when editing its properties:



Instance: i-089f05b4e52404f10 (DBB 1.1.3 WebApp Server - New OCP Cluster)

Tags

Key	Value
Capability	DeploymentMachine
Name	DBB 1.1.3 WebApp Server - New OCP Cluster

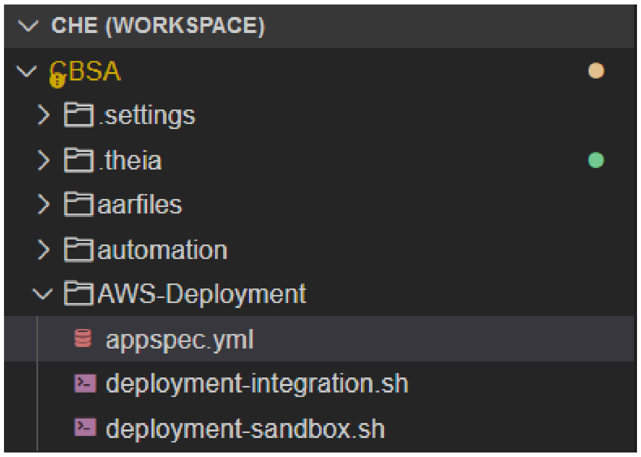
3.2.2 Configuring the deployment tasks

Later, when including the deployment step in the AWS CodePipeline definition, the location of the AWS S3 bucket that stores the package for deployment needs to be specified. Within this package, it is expected to find an *appspec.yml* file, that describes the different actions to perform. Here is an example of the *appspec.yml* file used in our configuration:

```yaml
# For help completing this file, see the "AppSpec File Reference" in the
#   "CodeDeploy User Guide" at
#   https://docs.aws.amazon.com/codedeploy/latest/userguide/app-spec-ref.html
version: 1.0
os: linux
files:
  - source: /
    destination: /var/work/deploy/
file_exists_behavior: OVERWRITE
permissions:
  - object: /var/work/deploy/
    owner: root
    group: root
    mode: 777

hooks:
  AfterInstall:
    - location: deployment.sh
      runas: root
      timeout: 1800
```

The *files* section of this sample file describes the destination path where the package should be copied and expanded on the EC2 machine: in our configuration, the package will be expanded in the */war/work/deploy* folder. Commands to execute are described in the *hooks* section. In our configuration, the *AfterInstall* action is performed by executing the *deployment.sh* script, which is also part of the ZIP package (created during the build phase).

We recommend to store both the *appspec.yml* file and the *deployment.sh* script in a specific subfolder of our application's Git repository:



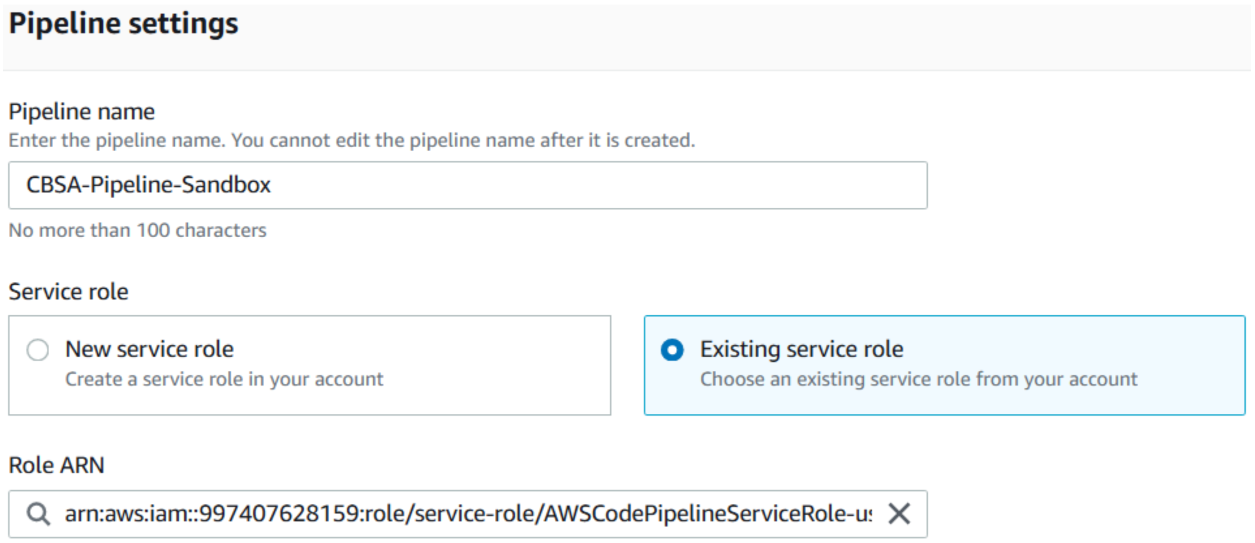
The deployment.sh script contains commands to deploy Z artifacts to a target Z system, using Ansible playbooks. Depending on the infrastructure, the tooling and deployment strategy, any other solution can be chosen for deploying Z artifacts to the target Z system.

3.3 Defining the AWS CodePipeline orchestration

The AWS CodePipeline service in AWS is used to orchestrate multiple steps for a complete CI/CD pipeline.

A pipeline in AWS typically is composed of three major steps: Source, Build and Deploy. To integrate these steps into a new pipeline, the AWS CodeBuild and AWS CodeDeploy configurations we previously defined will be leveraged.

The first action is to create a new pipeline in AWS CodePipeline. After specifying a name for this pipeline, you can choose to create a new Service Role if none exists for your account:



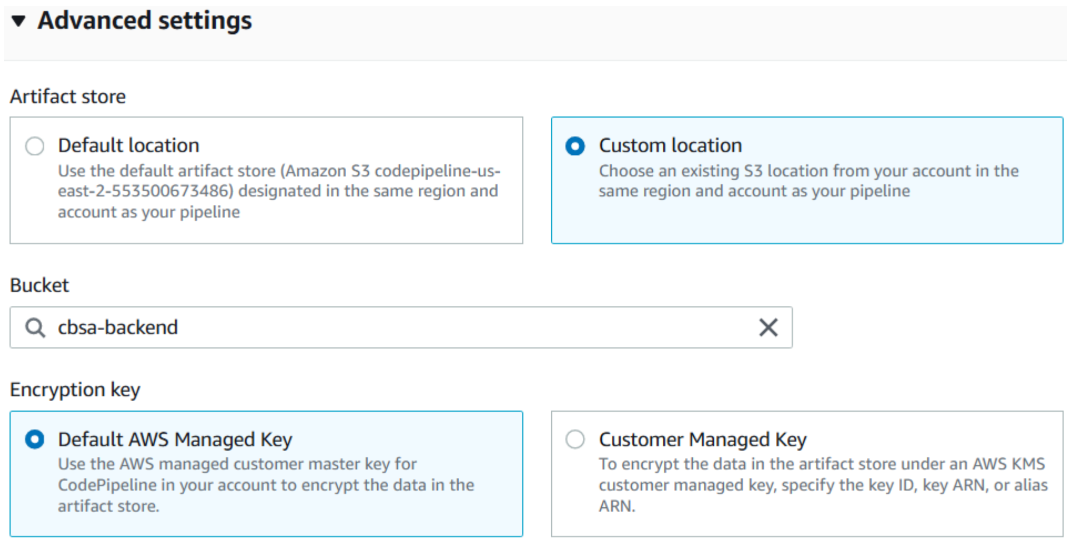
Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.
CBSA-Pipeline-Sandbox
No more than 100 characters

Service role
New service role — Create a service role in your account [✓ Existing service role — Choose an existing service role from your account]

Role ARN
arn:aws:iam::997407628159:role/service-role/AWSCodePipelineServiceRole-u:

For the *Artifact store*, we specify the existing *AWS S3* bucket, which we had configured in the previous step. For the Encryption key, the default value will be used.



▼ Advanced settings

Artifact store

( ) Default location
Use the default artifact store (Amazon S3 codepipeline-us-east-2-553500673486) designated in the same region and account as your pipeline

(●) Custom location
Choose an existing S3 location from your account in the same region and account as your pipeline

Bucket

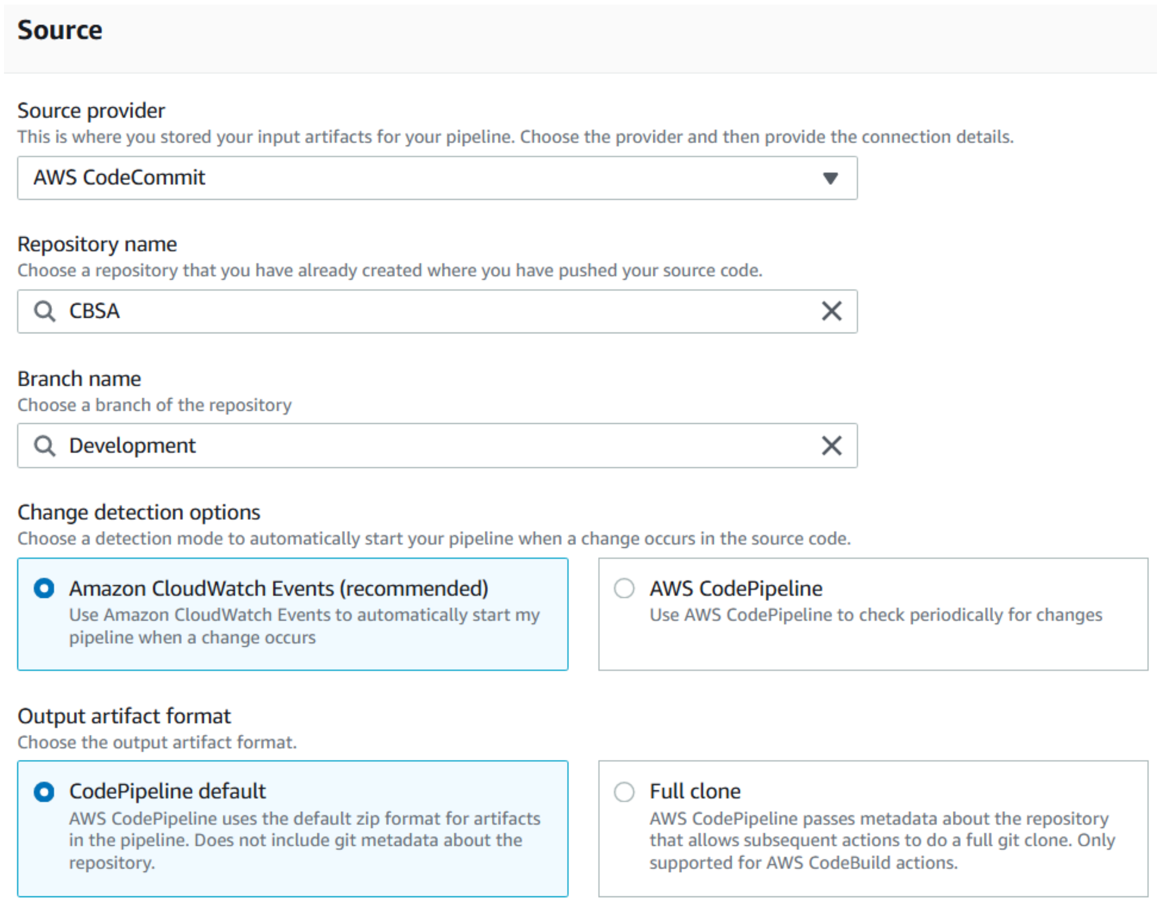
cbsa-backend

Encryption key

(●) Default AWS Managed Key
Use the AWS managed customer master key for CodePipeline in your account to encrypt the data in the artifact store.

( ) Customer Managed Key
To encrypt the data in the artifact store under an AWS KMS customer managed key, specify the key ID, key ARN, or alias ARN.

The next panel lets you choose the configuration for the first step of the pipeline, the *Source* stage. In the Source provider drop-down list, the AWS CodeCommit service will be used, while choosing the Git repository that hosts our source files. The *Development* branch will be the branch used as source: every change to this branch will then trigger the execution of this CI/CD pipeline:



Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

AWS CodeCommit

Repository name
Choose a repository that you have already created where you have pushed your source code.

CBSA

Branch name
Choose a branch of the repository

Development

Change detection options
Choose a detection mode to automatically start your pipeline when a change occurs in the source code.

(●) Amazon CloudWatch Events (recommended)
Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs

( ) AWS CodePipeline
Use AWS CodePipeline to check periodically for changes

Output artifact format
Choose the output artifact format.

(●) CodePipeline default
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.

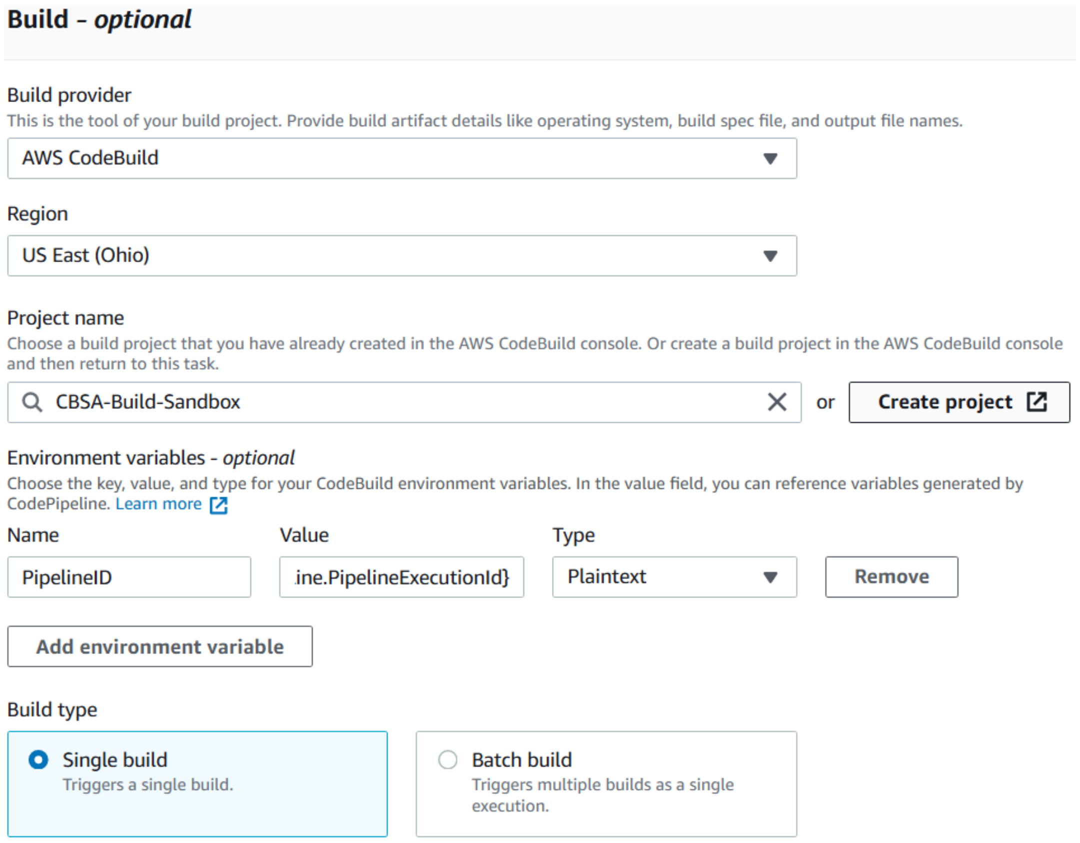
( ) Full clone
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

Default values are left for the remaining options.

3.3.1 Integrating the build specification

In the next phase, we will configure the *Build* step of the pipeline. AWS CodeBuild will be selected as the provider for this action. Select the Build project that we previously created.

Environment variables are also provided for the execution of the Build phase: in our setup, the *PipelineID* environment variable is defined with the value `#{codepipeline.PipelineExecutionId}`, which will be leveraged during the execution of the build step (See section 3.1 Configuring AWS CodeBuild to build mainframe applications).



Build - *optional*

Build provider
This is the tool of your build project. Provide build artifact details like operating system, build spec file, and output file names.
AWS CodeBuild

Region
US East (Ohio)

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.
CBSA-Build-Sandbox

Environment variables - *optional*
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. Learn more

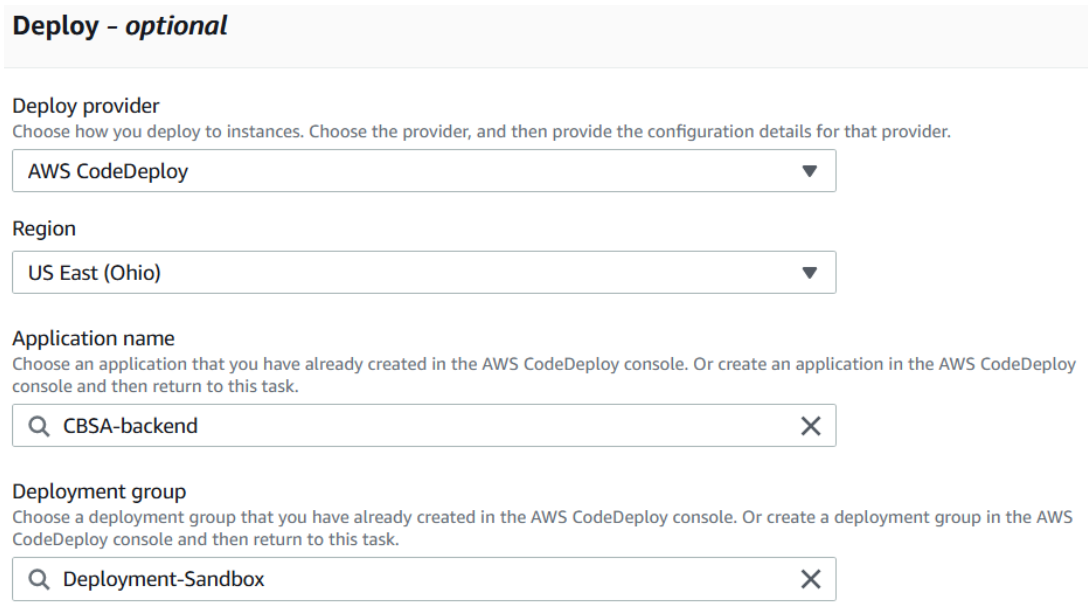
Name	Value	Type
PipelineID	ine.PipelineExecutionId}	Plaintext

Build type
[✓ Single build] Triggers a single build.
Batch build: Triggers multiple builds as a single execution.

3.3.2 Integrating the deployment configuration

The *Deploy* phase is the last phase of the pipeline.

Select *AWS CodeDeploy* as the provider for this action, which will allow you to pick the existing application name *CBSA-backend* and the deployment group *Deployment-Sandbox* to specify the target runtime environment:



Deploy - *optional*

Deploy provider
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.
AWS CodeDeploy

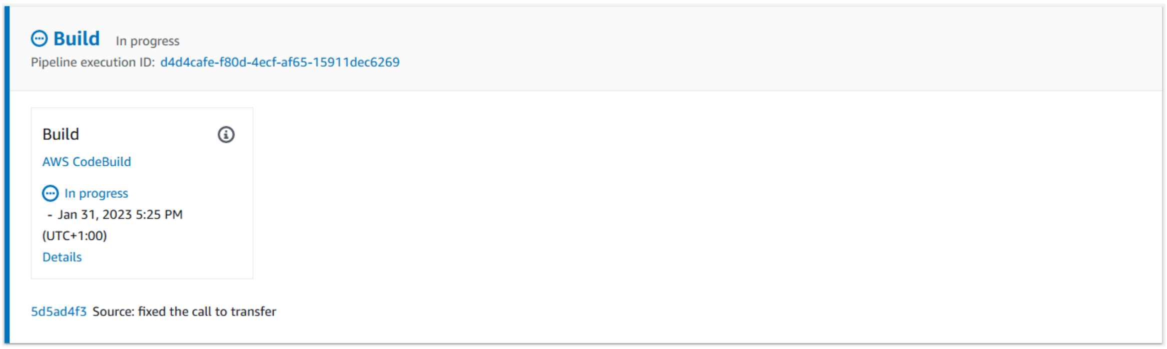
Region
US East (Ohio)

Application name
Choose an application that you have already created in the AWS CodeDeploy console. Or create an application in the AWS CodeDeploy console and then return to this task.
CBSA-backend

Deployment group
Choose a deployment group that you have already created in the AWS CodeDeploy console. Or create a deployment group in the AWS CodeDeploy console and then return to this task.
Deployment-Sandbox

3.3.3 Verifying the pipeline execution

After the creation of the pipeline, it will be automatically executed for the first time. The Source step should execute successfully, continuing with the Build phase:



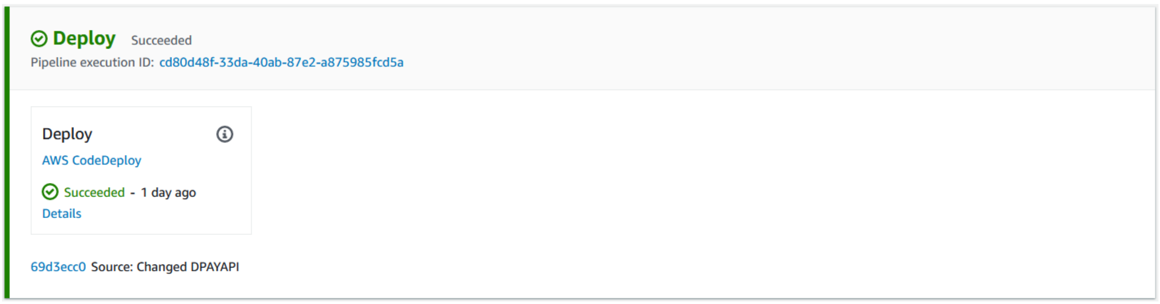
Build In progress
Pipeline execution ID: d4d4cafe-f80d-4ecf-af65-15911dec6269

Build
AWS CodeBuild
In progress
- Jan 31, 2023 5:25 PM
(UTC+1:00)
Details

5d5ad4f3 Source: fixed the call to transfer

The *Details* link is a convenient way to monitor the execution of the Build step, executing actions on the Z Sandbox system.

The last phase of the pipeline is the deployment on the target Wazi Sandbox system:



Deploy Succeeded
Pipeline execution ID: cd80d48f-33da-40ab-87e2-a875985fcd5a

Deploy
AWS CodeDeploy
Succeeded - 1 day ago
Details

69d3ecc0 Source: Changed DPAYAPI

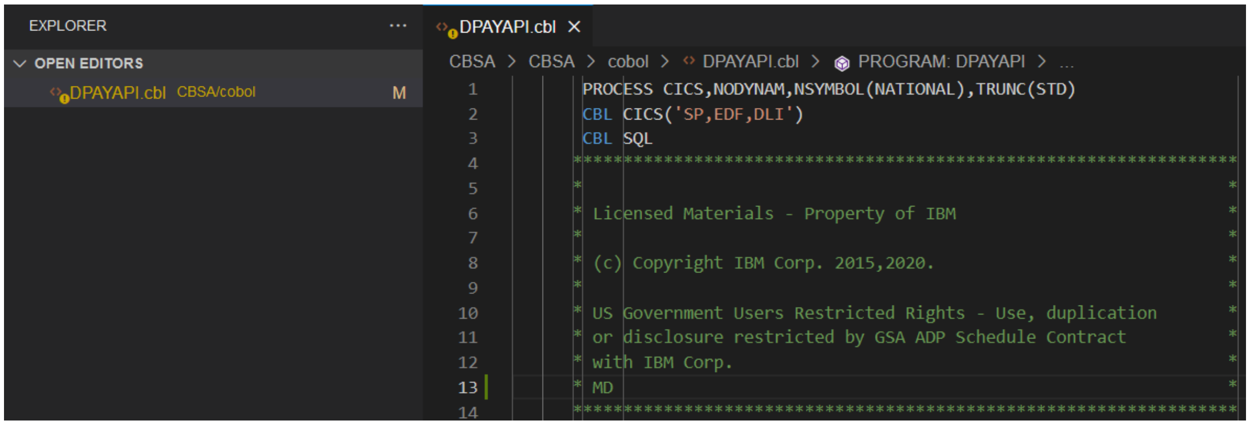
If the deployment went successfully, the pipeline should appear complete.

3.4 A sample developer workflow

To demonstrate the execution of the pipeline we just configured, a simple change can be implemented using Wazi for DevSpaces. This change should trigger the pipeline execution in AWS CodePipeline.

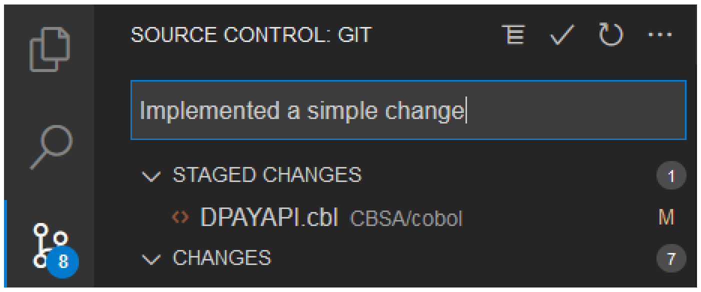
3.4.1 Implementing a change with Wazi for DevSpaces

The purpose of this change is to verify the correct configuration of the different components of the setup. Hence, the change we will implement is very simple and is not aiming to jeopardize the build of the artifact. We will just insert a comment (on line 13 here) in a Cobol program, called DPAYAPI, that belongs to the CBSA Git repository hosted on AWS CodeCommit.

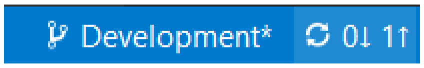


```
 1       PROCESS CICS,NODYNAM,NSYMBOL(NATIONAL),TRUNC(STD)
 2       CBL CICS('SP,EDF,DLI')
 3       CBL SQL
 4      ******************************************************************
 5      *                                                                *
 6      * Licensed Materials - Property of IBM                           *
 7      *                                                                *
 8      * (c) Copyright IBM Corp. 2015,2020.                             *
 9      *                                                                *
10      * US Government Users Restricted Rights - Use, duplication       *
11      * or disclosure restricted by GSA ADP Schedule Contract          *
12      * with IBM Corp.                                                 *
13      * MD                                                             *
14      ******************************************************************
```

The change is implemented on the Development branch of our CBSA Git repository. The next step is to commit the change on our local Development branch, by providing a commit message:

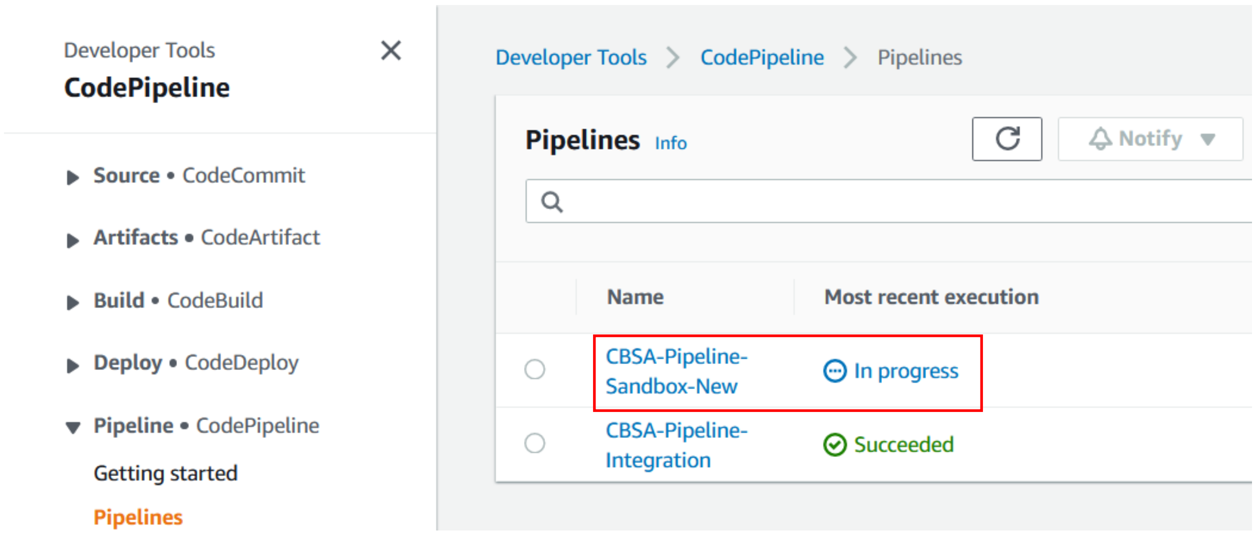


The bottom-left icon then shows a change on the Development branch is waiting to be pushed to the AWS CodeCommit repository. Clicking on the highlighted button below will push the changes to the Development branch on the central Git repository:



3.4.2 Monitoring the pipeline execution in AWS CodePipeline

In AWS CodePipeline, the pipeline previously configured should be now executing:



Developer Tools

CodePipeline

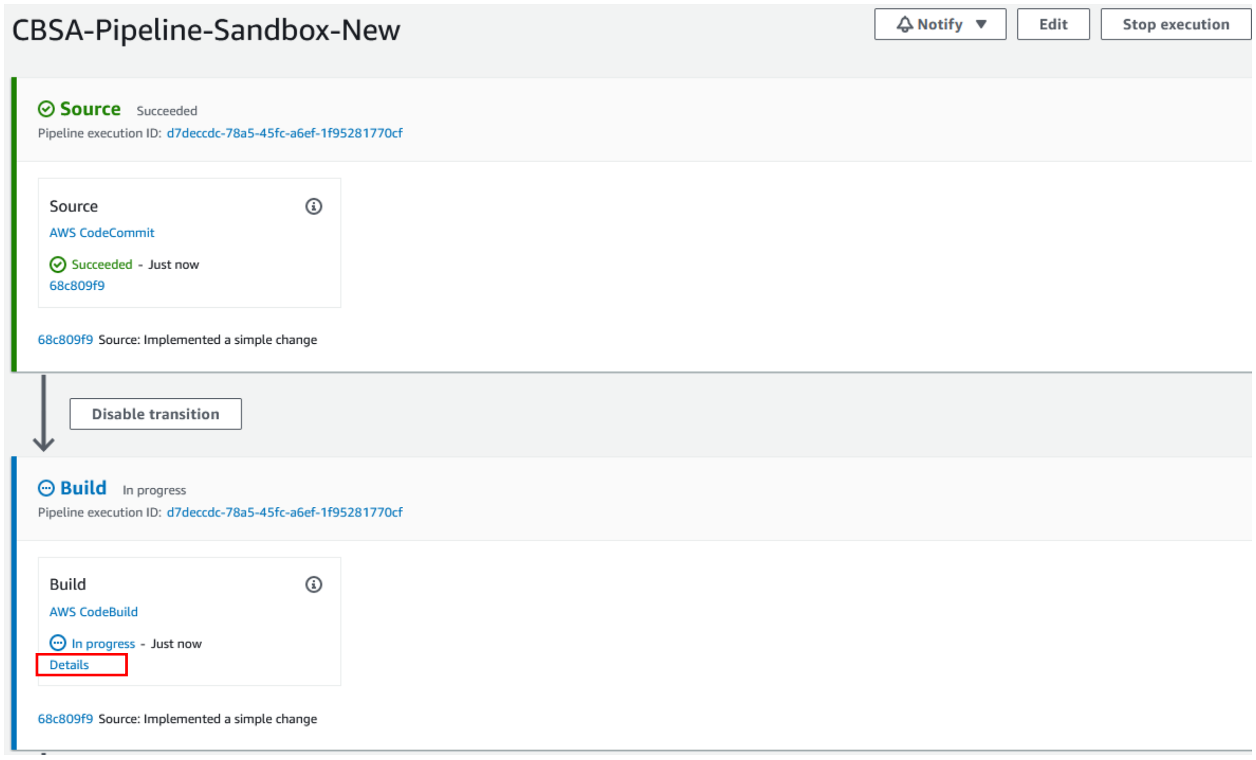
- Source • CodeCommit
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
  - Getting started
  - Pipelines

Developer Tools > CodePipeline > Pipelines

Pipelines Info

	Name	Most recent execution
○	CBSA-Pipeline-Sandbox-New	In progress
○	CBSA-Pipeline-Integration	Succeeded

Clicking on this pipeline brings you to the execution details, where you can see the Build phase currently being executed:



CBSA-Pipeline-Sandbox-New

Source Succeeded

Pipeline execution ID: d7deccdc-78a5-45fc-a6ef-1f95281770cf

Source

AWS CodeCommit

Succeeded - Just now

68c809f9

68c809f9 Source: Implemented a simple change

Build In progress

Pipeline execution ID: d7deccdc-78a5-45fc-a6ef-1f95281770cf

Build

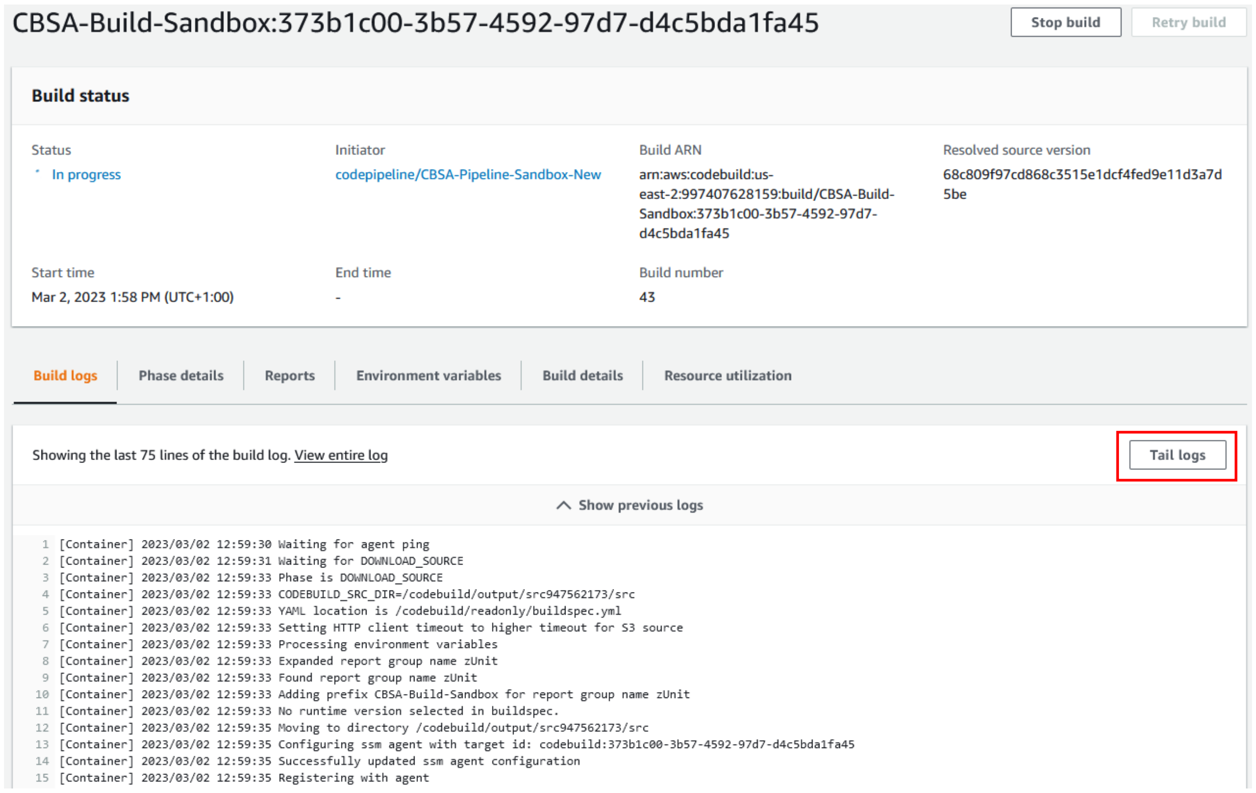
AWS CodeBuild

In progress - Just now

Details

68c809f9 Source: Implemented a simple change

On the *Details* page, you get more information about the build process that is executed:


CBSA-Build-Sandbox:373b1c00-3b57-4592-97d7-d4c5bda1fa45

Build status

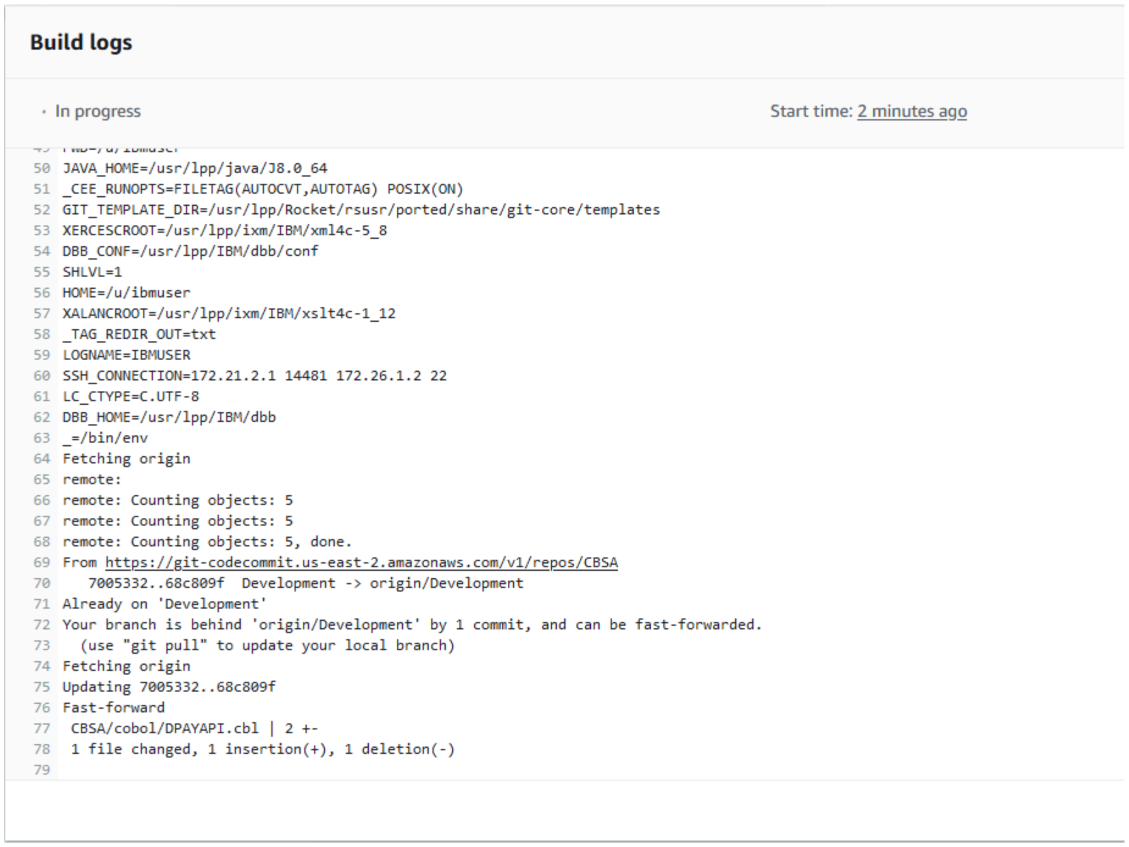
Status	Initiator	Build ARN	Resolved source version
In progress	codepipeline/CBSA-Pipeline-Sandbox-New	arn:aws:codebuild:us-east-2:997407628159:build/CBSA-Build-Sandbox:373b1c00-3b57-4592-97d7-d4c5bda1fa45	68c809f97cd868c3515e1dcf4fed9e11d3a7d5be

Start time	End time	Build number
Mar 2, 2023 1:58 PM (UTC+1:00)	-	43

Showing the last 75 lines of the build log. View entire log

```
[Container] 2023/03/02 12:59:30 Waiting for agent ping
[Container] 2023/03/02 12:59:31 Waiting for DOWNLOAD_SOURCE
[Container] 2023/03/02 12:59:33 Phase is DOWNLOAD_SOURCE
[Container] 2023/03/02 12:59:33 CODEBUILD_SRC_DIR=/codebuild/output/src947562173/src
[Container] 2023/03/02 12:59:33 YAML location is /codebuild/readonly/buildspec.yml
[Container] 2023/03/02 12:59:33 Setting HTTP client timeout to higher timeout for S3 source
[Container] 2023/03/02 12:59:33 Processing environment variables
[Container] 2023/03/02 12:59:33 Expanded report group name zUnit
[Container] 2023/03/02 12:59:33 Found report group name zUnit
[Container] 2023/03/02 12:59:33 Adding prefix CBSA-Build-Sandbox for report group name zUnit
[Container] 2023/03/02 12:59:33 No runtime version selected in buildspec.
[Container] 2023/03/02 12:59:35 Moving to directory /codebuild/output/src947562173/src
[Container] 2023/03/02 12:59:35 Configuring ssm agent with target id: codebuild:373b1c00-3b57-4592-97d7-d4c5bda1fa45
[Container] 2023/03/02 12:59:35 Successfully updated ssm agent configuration
[Container] 2023/03/02 12:59:35 Registering with agent
```

This page allows you to monitor the execution of the build phase, which leverages DBB running on the Wazi Sandbox system. Clicking on the *Tail logs* button shows the execution log being updated live:

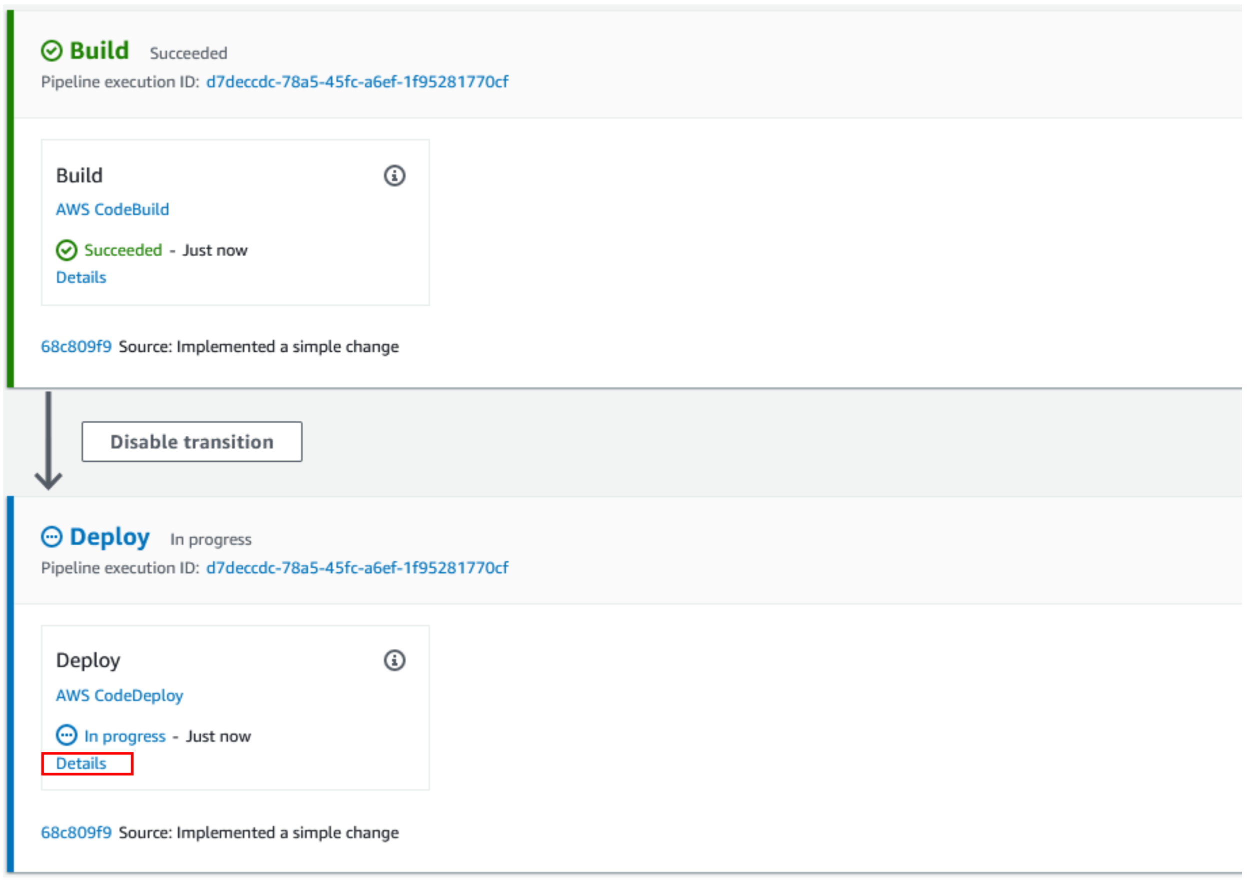

Build logs

In progress

Start time: 2 minutes ago

```
JAVA_HOME=/usr/lpp/java/J8.0_64
_CEE_RUNOPTS=FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)
GIT_TEMPLATE_DIR=/usr/lpp/Rocket/rsusr/ported/share/git-core/templates
XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_8
DBB_CONF=/usr/lpp/IBM/dbb/conf
SHLVL=1
HOME=/u/ibmuser
XALANCROOT=/usr/lpp/ixm/IBM/xslt4c-1_12
_TAG_REDIR_OUT=txt
LOGNAME=IBMUSER
SSH_CONNECTION=172.21.2.1 14481 172.26.1.2 22
LC_CTYPE=C.UTF-8
DBB_HOME=/usr/lpp/IBM/dbb
_=/bin/env
Fetching origin
remote:
remote: Counting objects: 5
remote: Counting objects: 5
remote: Counting objects: 5, done.
From https://git-codecommit.us-east-2.amazonaws.com/v1/repos/CBSA
   7005332..68c809f  Development -> origin/Development
Already on 'Development'
Your branch is behind 'origin/Development' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
Fetching origin
Updating 7005332..68c809f
Fast-forward
 CBSA/cobol/DPAYAPI.cbl | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

When the build phase is done, the pipeline executes the Deploy phase:



Build Succeeded
Pipeline execution ID: d7deccdc-78a5-45fc-a6ef-1f95281770cf

Build
AWS CodeBuild
Succeeded - Just now
Details

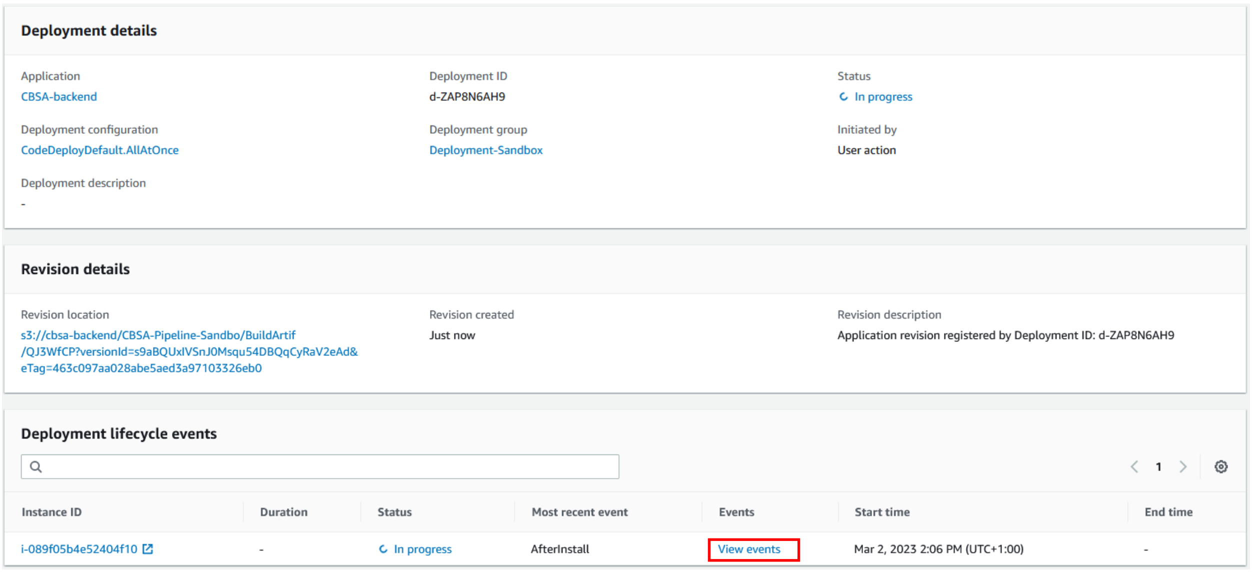
68c809f9 Source: Implemented a simple change

Deploy In progress
Pipeline execution ID: d7deccdc-78a5-45fc-a6ef-1f95281770cf

Deploy
AWS CodeDeploy
In progress - Just now
Details

68c809f9 Source: Implemented a simple change

Via the *Details* link, you can browse information about the deployment being processed:



Deployment details

Application CBSA-backend	Deployment ID d-ZAP8N6AH9	Status In progress
Deployment configuration CodeDeployDefault.AllAtOnce	Deployment group Deployment-Sandbox	Initiated by User action
Deployment description -		

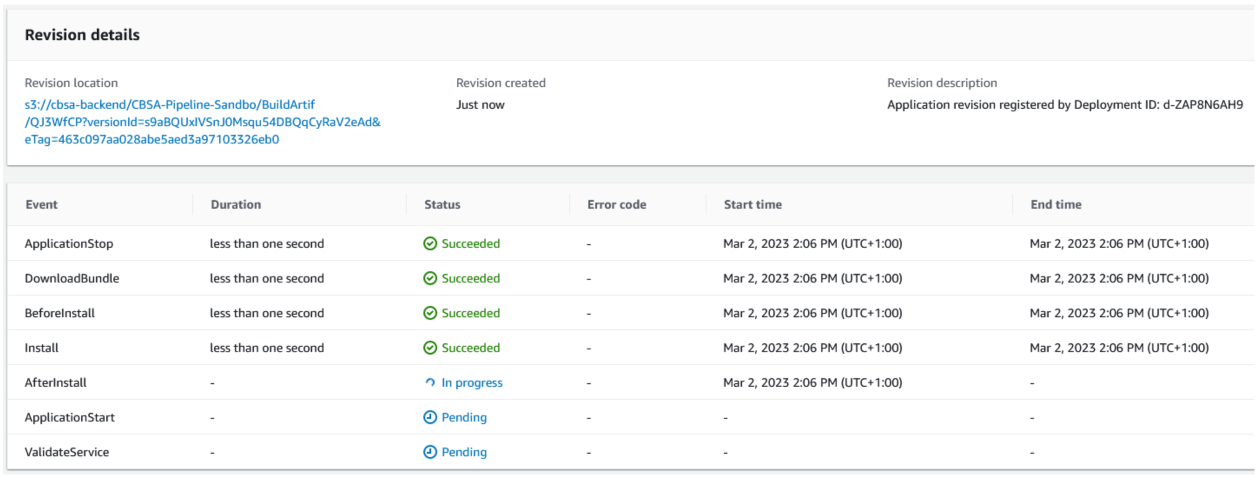
Revision details

Revision location s3://cbsa-backend/CBSA-Pipeline-Sandbo/BuildArtif/QJ3WfCP?versionId=s9aBQUxIVSnJ0Msqu54DBQqCyRaV2eAd&eTag=463c097aa028abe5aed3a97103326eb0	Revision created Just now	Revision description Application revision registered by Deployment ID: d-ZAP8N6AH9

Deployment lifecycle events

Instance ID	Duration	Status	Most recent event	Events	Start time	End time
i-089f05b4e52404f10	-	In progress	AfterInstall	View events	Mar 2, 2023 2:06 PM (UTC+1:00)	-

You can then click on the *View events* link to check the different steps of the deployment process:

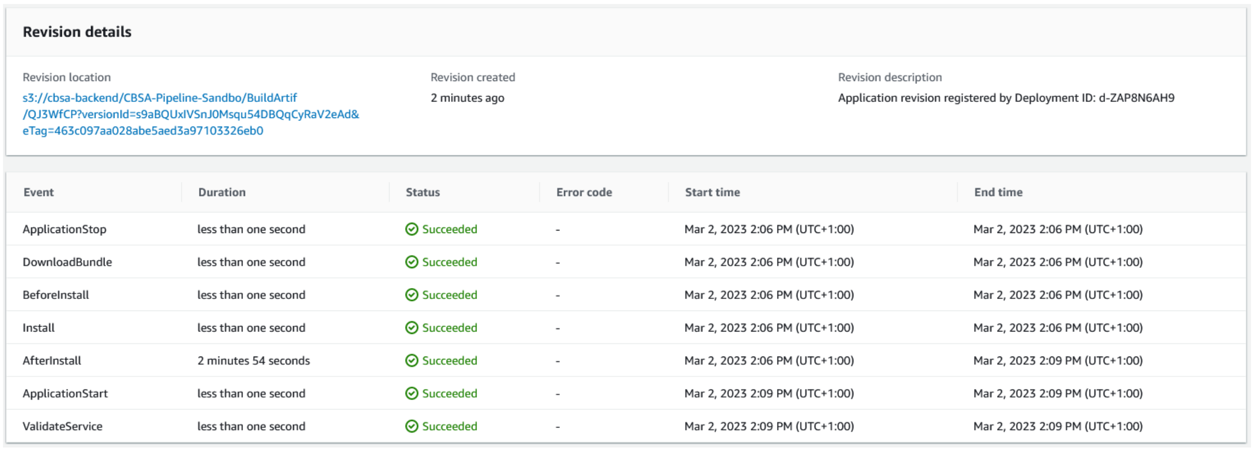


Revision details

Revision location	Revision created	Revision description
s3://cbsa-backend/CBSA-Pipeline-Sandbo/BuildArtif/QJ3WfCP?versionId=s9aBQUxIVSnJ0Msqu54DBQqCyRaV2eAd&eTag=463c097aa028abe5aed3a97103326eb0	Just now	Application revision registered by Deployment ID: d-ZAP8N6AH9

Event	Duration	Status	Error code	Start time	End time
ApplicationStop	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
DownloadBundle	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
BeforeInstall	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
Install	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
AfterInstall	-	In progress	-	Mar 2, 2023 2:06 PM (UTC+1:00)	-
ApplicationStart	-	Pending	-	-	-
ValidateService	-	Pending	-	-	-

After a short period of time, the deployment should be completed, and the application successfully deployed:



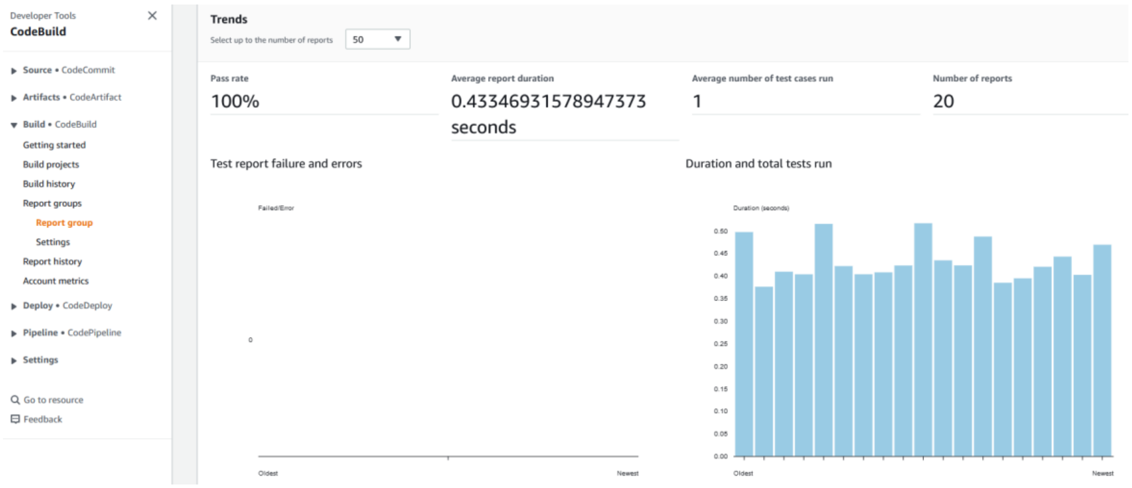
Revision details

Revision location	Revision created	Revision description
s3://cbsa-backend/CBSA-Pipeline-Sandbo/BuildArtif/QJ3WfCP?versionId=s9aBQUxIVSnJ0Msqu54DBQqCyRaV2eAd&eTag=463c097aa028abe5aed3a97103326eb0	2 minutes ago	Application revision registered by Deployment ID: d-ZAP8N6AH9

Event	Duration	Status	Error code	Start time	End time
ApplicationStop	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
DownloadBundle	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
BeforeInstall	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
Install	less than one second	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:06 PM (UTC+1:00)
AfterInstall	2 minutes 54 seconds	Succeeded	-	Mar 2, 2023 2:06 PM (UTC+1:00)	Mar 2, 2023 2:09 PM (UTC+1:00)
ApplicationStart	less than one second	Succeeded	-	Mar 2, 2023 2:09 PM (UTC+1:00)	Mar 2, 2023 2:09 PM (UTC+1:00)
ValidateService	less than one second	Succeeded	-	Mar 2, 2023 2:09 PM (UTC+1:00)	Mar 2, 2023 2:09 PM (UTC+1:00)

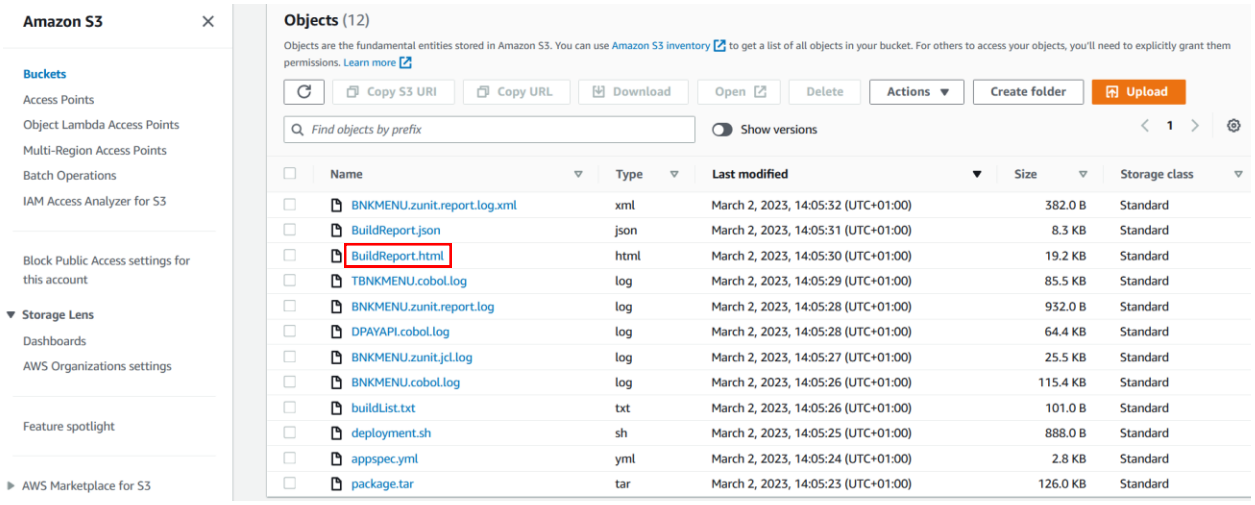
3.4.3 Browsing the pipeline artifacts and reports

If zUnit artifacts were generated as part of the pipeline and exported as test reports (See section 3.1 Configuring AWS CodeBuild to build mainframe applications), the test reports can be visualized in AWS CodeBuild, under the *Report groups* submenu item.



Pass rate	Average report duration	Average number of test cases run	Number of reports
100%	0.43346931578947373 seconds	1	20

As build log files and artifacts are also stored in an AWS S3 bucket, they can also be browsed, for traceability or debugging purposes:

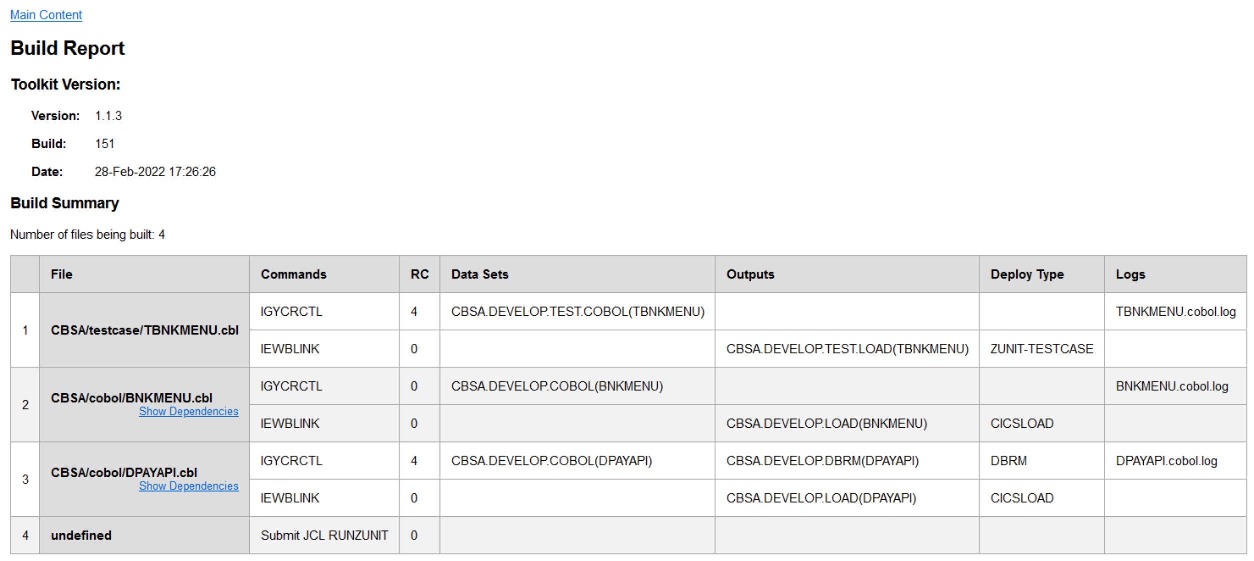


Objects (12)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
BNKMENU.zunit.report.log.xml	xml	March 2, 2023, 14:05:32 (UTC+01:00)	382.0 B	Standard
BuildReport.json	json	March 2, 2023, 14:05:31 (UTC+01:00)	8.3 KB	Standard
BuildReport.html	html	March 2, 2023, 14:05:30 (UTC+01:00)	19.2 KB	Standard
TBNKMENU.cobol.log	log	March 2, 2023, 14:05:29 (UTC+01:00)	85.5 KB	Standard
BNKMENU.zunit.report.log	log	March 2, 2023, 14:05:28 (UTC+01:00)	932.0 B	Standard
DPAYAPI.cobol.log	log	March 2, 2023, 14:05:28 (UTC+01:00)	64.4 KB	Standard
BNKMENU.zunit.jcl.log	log	March 2, 2023, 14:05:27 (UTC+01:00)	25.5 KB	Standard
BNKMENU.cobol.log	log	March 2, 2023, 14:05:26 (UTC+01:00)	115.4 KB	Standard
buildList.txt	txt	March 2, 2023, 14:05:26 (UTC+01:00)	101.0 B	Standard
deployment.sh	sh	March 2, 2023, 14:05:25 (UTC+01:00)	888.0 B	Standard
appspec.yml	yml	March 2, 2023, 14:05:24 (UTC+01:00)	2.8 KB	Standard
package.tar	tar	March 2, 2023, 14:05:23 (UTC+01:00)	126.0 KB	Standard

For instance, the HTML version of the DBB Build Report can be consulted by clicking on it:



Main Content

Build Report

Toolkit Version:

Version:	1.1.3
Build:	151
Date:	28-Feb-2022 17:26:26

Build Summary

Number of files being built: 4

	File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
1	**CBSA/testcase/TBNKMENU.cbl**	IGYCRCTL	4	CBSA.DEVELOP.TEST.COBOL(TBNKMENU)			TBNKMENU.cobol.log
		IEWBLINK	0		CBSA.DEVELOP.TEST.LOAD(TBNKMENU)	ZUNIT-TESTCASE	
2	**CBSA/cobol/BNKMENU.cbl** Show Dependencies	IGYCRCTL	0	CBSA.DEVELOP.COBOL(BNKMENU)			BNKMENU.cobol.log
		IEWBLINK	0		CBSA.DEVELOP.LOAD(BNKMENU)	CICSLOAD	
3	**CBSA/cobol/DPAYAPI.cbl** Show Dependencies	IGYCRCTL	4	CBSA.DEVELOP.COBOL(DPAYAPI)	CBSA.DEVELOP.DBRM(DPAYAPI)	DBRM	DPAYAPI.cobol.log
		IEWBLINK	0		CBSA.DEVELOP.LOAD(DPAYAPI)	CICSLOAD	
4	**undefined**	Submit JCL RUNZUNIT	0				

With all the information collected from the pipeline and stored on the AWS Cloud platform, the developers have access to all the required knowledge when building and diagnosing issues with the different stages of the CI/CD pipeline.

4 Building a CI/CD pipeline with Azure DevOps and IBM Z & Cloud Modernization Stack

This section outlines the steps needed to configure a CI/CD process using IBM Z & Cloud Modernization Stack[24] with Wazi Dev Spaces and Wazi Sandbox on Microsoft Azure and its DevOps services (ADO).

To start, begin with the steps defined in sections 1 and 2 of this document. They are common across all IBM supported cloud providers including Azure, which defines a foundation that includes:

- Administrative steps to install and configure:
  - A RedHat OpenShift Cluster (OCP) in a Cloud environment.
  - IBM's Dependency Based Build (DBB) and its supporting utility scripts.
  - A Git Repository structure that is compatible with DBB build frameworks like zAppBuild.
  - Support for ZUnit and the IBM testing framework.
- Developer Tools are defined in the section titled "Setting up Wazi for Dev Spaces and Wazi Analyze". It describes the developer experience using:
  - Wazi DevSpaces
  - Wazi Sandbox
  - Wazi Analyze
  - ZUnit

In addition to the foundational services defined above, some administrative steps specific to defining an Azure DevOps CI/CD will be needed. See IBM's "Getting Started" reference for more details - Azure DevOps and IBM Dependency Based Build Integration

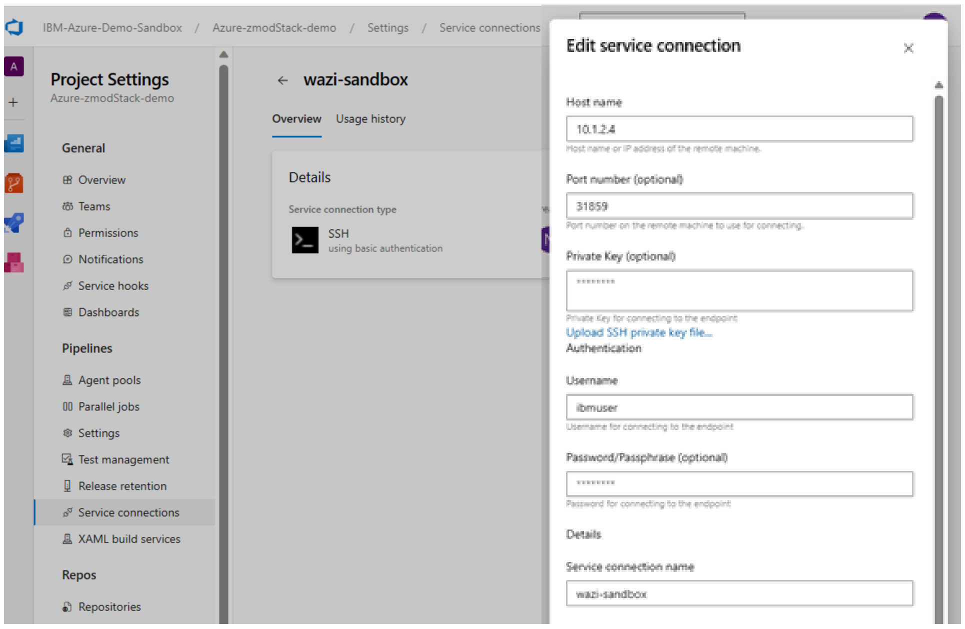
This section assumes prior knowledge of DevOps concepts with IBM DBB on z/OS, Azure DevOps, z/OS systems knowledge and Cloud Administration.

4.1 Azure DevOps and z/OS integration

As defined in the "Getting Started" reference above, a basic[25] CI/CD process runs a build, publish, and deploy workflow on a target z/OS environment. However, in this document, an alternative publishing and deployment process is provided that uses ADO Artifacts and ADO Release with IBM's Wazi Deploy tool.

The "Getting Started" reference also describes how to define an ADO Agent with SSH connectivity to a z/OS host. Those same steps can be applied to connecting to a virtual z/OS like a Wazi Sandbox. In this example, an Azure Ubuntu VM was provisioned and configured as a self-hosted agent on the same subnet as a provisioned *Red Hat Open Shift Container Platform* (OCP). Also, an ADO SSH Service Connection was defined with a RACF User's SSH key, IP and port of the Wazi Sandbox. In the example below, the SSH service is called *wazi-sandbox*:

[24] Microsoft Azure Marketplace
[25] More mature pipelines may include other tasks like quality gates, peer-review, approvals or more.



Edit service connection

Host name: 10.1.2.4
Host name or IP address of the remote machine.

Port number (optional): 31859
Port number on the remote machine to use for connecting.

Private Key (optional): ********
Private Key for connecting to the endpoint
Upload SSH private key file...

Authentication

Username: ibmuser
Username for connecting to the endpoint

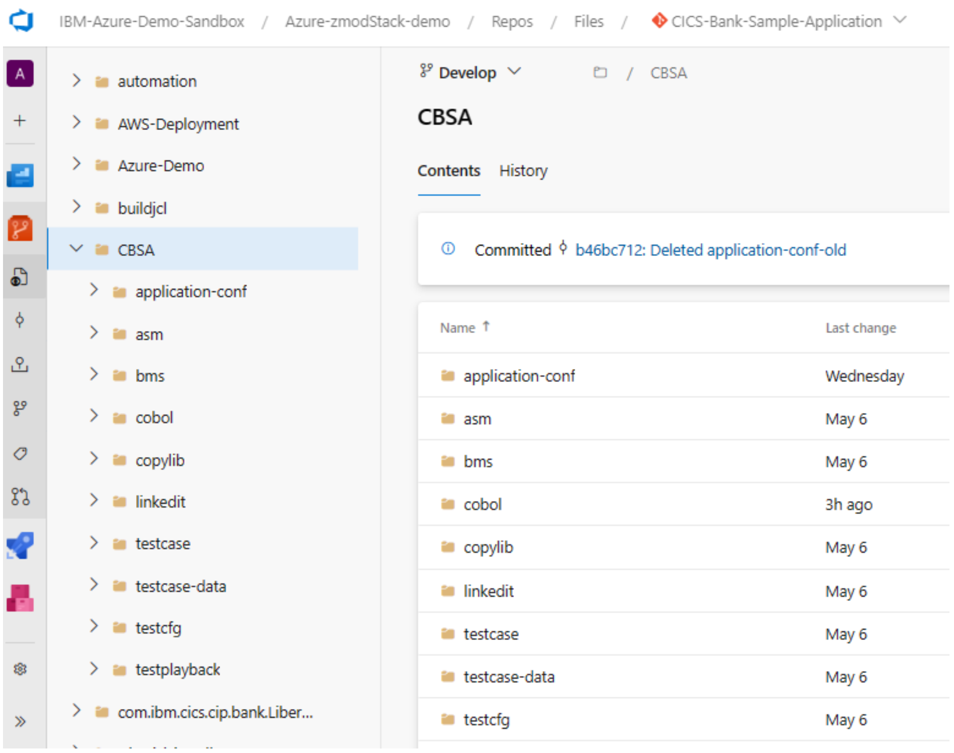
Password/Passphrase (optional): ********
Password for connecting to the endpoint

Details

Service connection name: wazi-sandbox

4.2 Creating the Git repository

Starting with a new ADO Project as shown below, a repository configured for DBB is created. ADO supports all major Git providers. This example uses Azure Repos.

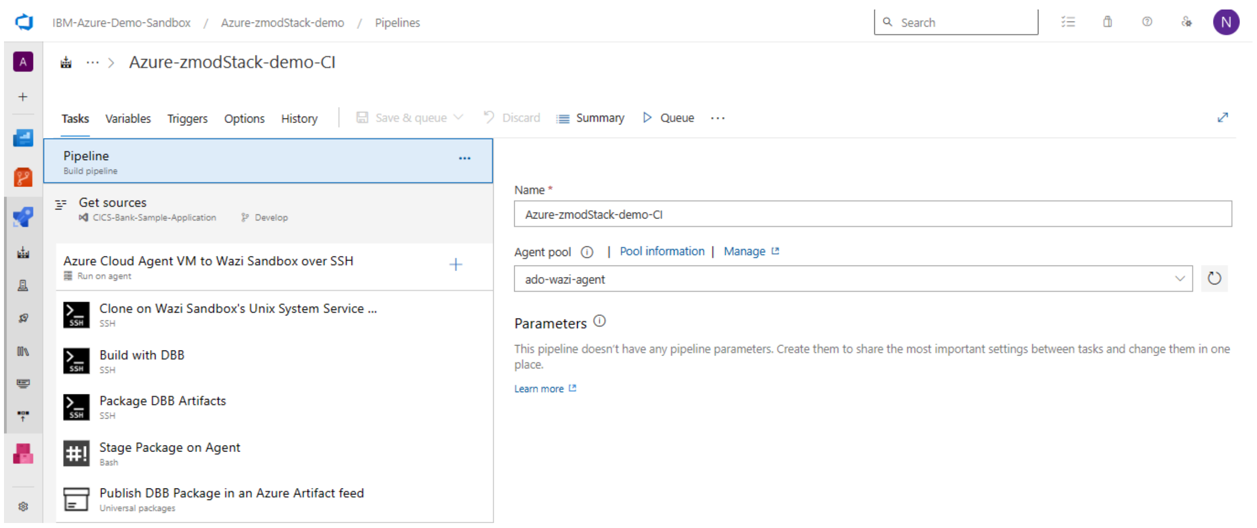


Committed b46bc712: Deleted application-conf-old

Name	Last change
application-conf	Wednesday
asm	May 6
bms	May 6
cobol	3h ago
copylib	May 6
linkedit	May 6
testcase	May 6
testcase-data	May 6
testcfg	May 6

4.3 Defining the CI pipeline in Azure DevOps

The CI pipeline example below illustrates the clone, build and publishing tasks[26].
The first 3 steps are SSH tasks that are securely executed on the Wazi Sandbox system deployed in OCP.


IBM-Azure-Demo-Sandbox / Azure-zmodStack-demo / Pipelines

Azure-zmodStack-demo-CI

Tasks | Variables | Triggers | Options | History | Save & queue | Discard | Summary | Queue

Pipeline
Build pipeline

Get sources
CICS-Bank-Sample-Application · Develop

Azure Cloud Agent VM to Wazi Sandbox over SSH
Run on agent

- Clone on Wazi Sandbox's Unix System Service ... (SSH)
- Build with DBB (SSH)
- Package DBB Artifacts (SSH)
- Stage Package on Agent (Bash)
- Publish DBB Package in an Azure Artifact feed (Universal packages)

Name *: Azure-zmodStack-demo-CI

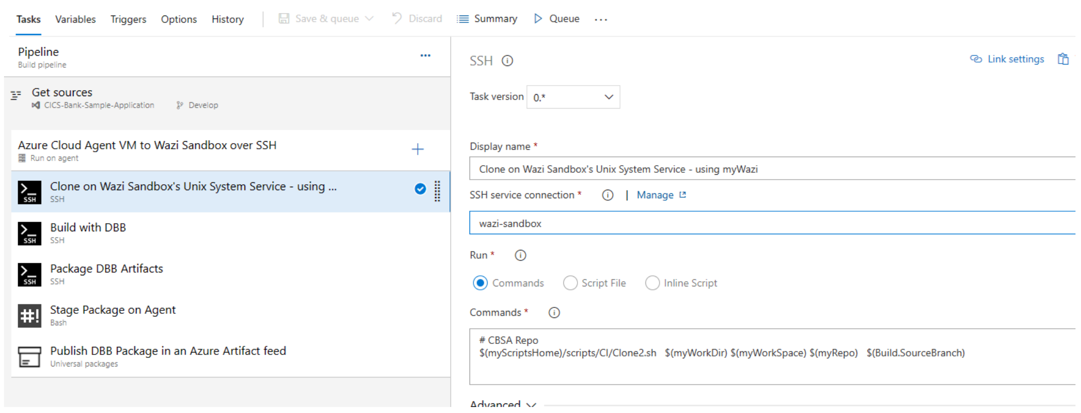
Agent pool | Pool information | Manage: ado-wazi-agent

Parameters

This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.

Learn more

For example, the SSH task below references an SSH Service Connection (endpoint) and a command. In this case, the endpoint is to a Wazi Sandbox and the command is a Git Clone shell script. Variable definitions and sample shell scripts are provided in the "Getting Started" reference.


Tasks | Variables | Triggers | Options | History | Save & queue | Discard | Summary | Queue

Pipeline
Build pipeline

Get sources
CICS-Bank-Sample-Application · Develop

Azure Cloud Agent VM to Wazi Sandbox over SSH
Run on agent

- Clone on Wazi Sandbox's Unix System Service - using ... (SSH)
- Build with DBB (SSH)
- Package DBB Artifacts (SSH)
- Stage Package on Agent (Bash)
- Publish DBB Package in an Azure Artifact feed (Universal packages)

SSH | Link settings

Task version: 0.*

Display name *: Clone on Wazi Sandbox's Unix System Service - using myWazi

SSH service connection * | Manage: wazi-sandbox

Run *: [✓ Commands] Script File Inline Script

Commands *:

```
# CBSA Repo
$(myScriptsHome)/scripts/CI/Clone2.sh   $(myWorkDir) $(myWorkSpace) $(myRepo)   $(Build.SourceBranch)
```

Advanced

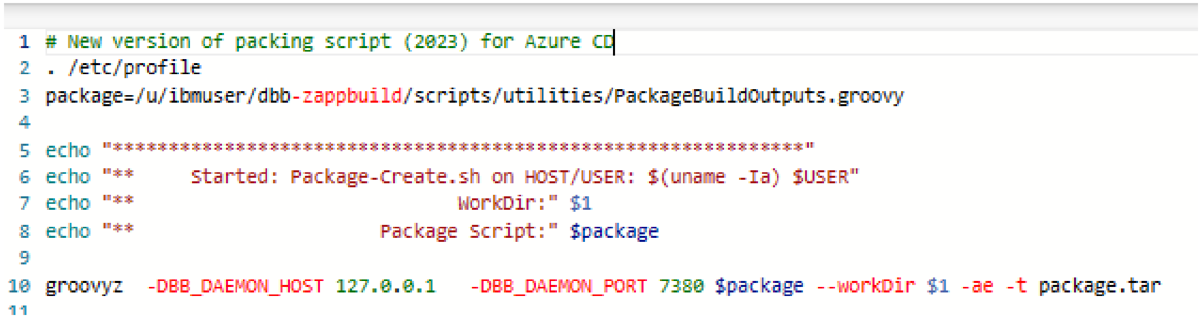
The package task runs an SSH shell command to launch the DBB groovy utility called *PackageBuildOutputs.groovy* (see "Retrieving the Build framework" section above). The shell script takes one argument passed by the pipeline that points to the build step's DBB artifacts folder known as the DBB *--workdir* argument.

In this example, the groovy script and its shell launcher are placed in a subfolder within a cloned copy of *dbb-zappbuild*. But they can be stored on any accessible sandbox Unix Filesystem folder.

[26] Tasks have been split for illustration purposes. But can be combined as needed.

The reference to *. /etc/profile* in line 2 is used to initialize the DBB environment variables used to launch the script under the DBB *groovyz* utility. The arguments passed to the script include:

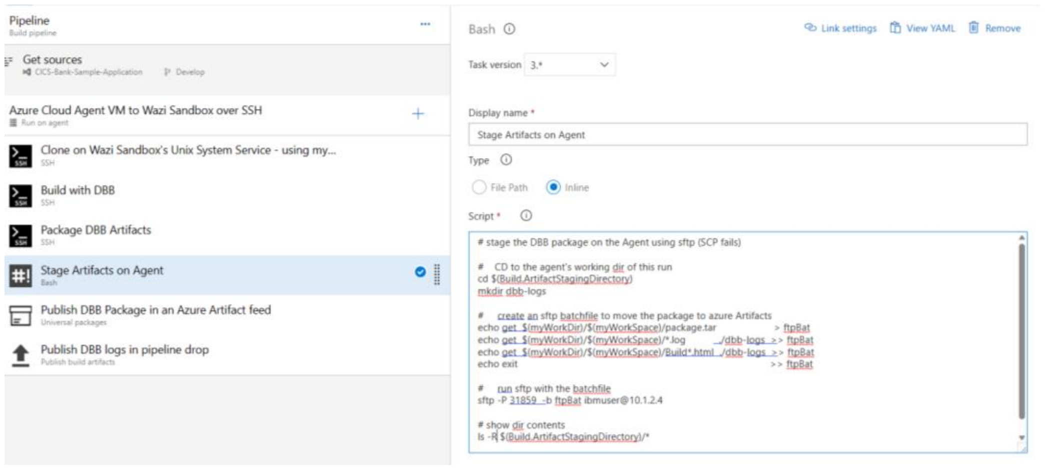
- A DBB Daemon host IP and its port which is described in the DBB Users Guide[27].
- The *$package* variable with the location of the packaging groovy script (line 3).
- *--workDir* is the working directory for zAppBuild ($1).
- *The -ae* parameter is required to add extensions to each artifact that is packaged
- *-t* is the name for the package file. This contains all objects to be deployed like load modules, CICS Maps, DB2 DBRM's and whatever items are needed by the application.



```
1  # New version of packing script (2023) for Azure CD
2  . /etc/profile
3  package=/u/ibmuser/dbb-zappbuild/scripts/utilities/PackageBuildOutputs.groovy
4
5  echo "*******************************************************************"
6  echo "**     Started: Package-Create.sh on HOST/USER: $(uname -Ia) $USER"
7  echo "**                             WorkDir:" $1
8  echo "**                      Package Script:" $package
9
10 groovyz  -DBB_DAEMON_HOST 127.0.0.1   -DBB_DAEMON_PORT 7380 $package --workDir $1 -ae -t package.tar
11
```

Stage is a bash task that runs SFTP on the Agent as shown below. The inline script performs the following steps:

- Changes directory to the Agent's Staging Directory of the current run,
- Makes a new directory to store DBB logs and HTML report,
- Creates a file, called *ftpBat*, to drive the SFTP commands that includes the application package and DBB logs.


Pipeline
Build pipeline
Get sources
CICS-Bank-Sample-Application
Develop
Azure Cloud Agent VM to Wazi Sandbox over SSH
Run on agent
Clone on Wazi Sandbox's Unix System Service - using my...
SSH
Build with DBB
SSH
Package DBB Artifacts
SSH
Stage Artifacts on Agent
Bash
Publish DBB Package in an Azure Artifact feed
Universal packages
Publish DBB logs in pipeline drop
Publish build artifacts
Bash
Link settings
View YAML
Remove

Task version: 3.*

Display name *: Stage Artifacts on Agent

Type: File Path [✓ Inline]

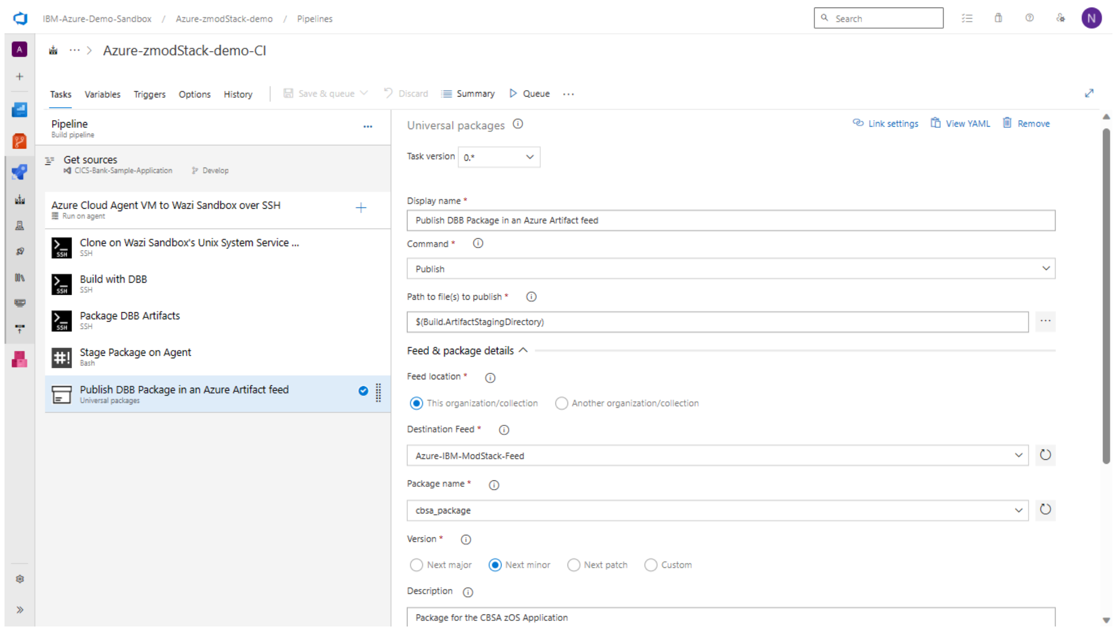
Script *:

```
# stage the DBB package on the Agent using sftp (SCP fails)

#    CD to the agent's working dir of this run
cd $(Build.ArtifactStagingDirectory)
mkdir dbb-logs

#     create an sftp batchfile to move the package to azure Artifacts
echo get  $(myWorkDir)/$(myWorkSpace)/package.tar                  > ftpBat
echo get  $(myWorkDir)/$(myWorkSpace)/*.log         ../dbb-logs  >> ftpBat
echo get  $(myWorkDir)/$(myWorkSpace)/Build*.html  ./dbb-logs  >> ftpBat
echo exit                                                          >> ftpBat

#     run sftp with the batchfile
sftp -P 31859  -b ftpBat ibmuser@10.1.2.4

# show dir contents
ls -R $(Build.ArtifactStagingDirectory)/*
```

[27] Note always refence the latest document version - IBM Dependency Based Build 2.0.x - IBM Documentation

Publish is a *Universal Package* task with:

- The command *Publish*,
- The default system PATH of the Agent's Staging Directory of the prior step,
- The pre-defined Feed name - in this case, it's called *Azure-IBM-ModStack-Feed*,
- A package name - in this case, it is the name of the sample application.



Task version: 0.*

Display name *: Publish DBB Package in an Azure Artifact feed

Command *: Publish

Path to file(s) to publish *: $(Build.ArtifactStagingDirectory)

Feed & package details

Feed location *: [✓ This organization/collection] Another organization/collection

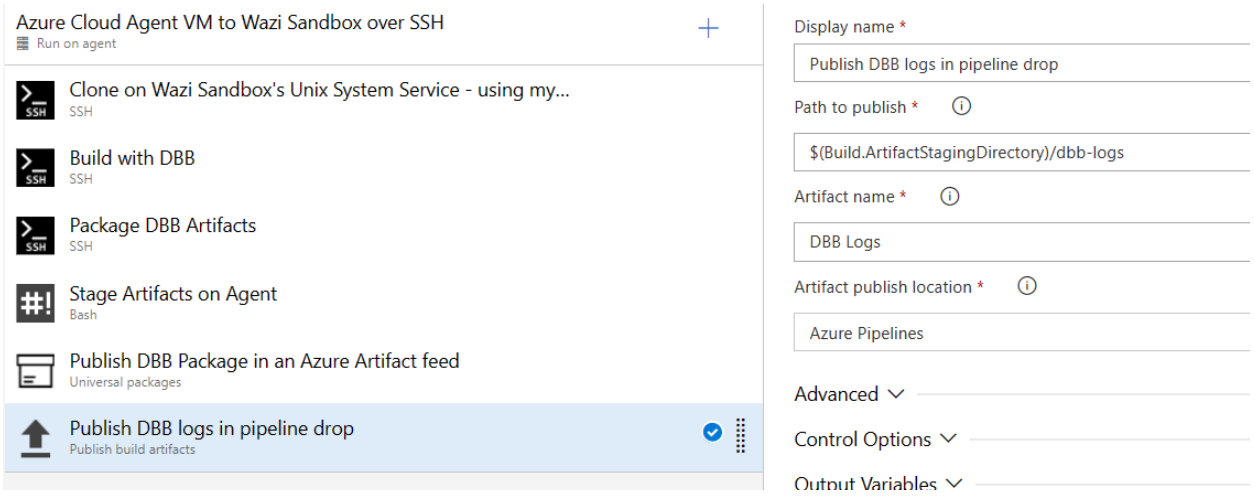
Destination Feed *: Azure-IBM-ModStack-Feed

Package name *: cbsa_package

Version *: Next major [✓ Next minor] Next patch Custom

Description: Package for the CBSA zOS Application

An optional *Publish build artifacts* task can be added to save DBB logs. This task uses the Agent's default working directory to store the logs of the Build step in a folder called *dbb-logs* in this example. Provide an Artifact name like *DBB Logs* and use *Azure Pipelines* as the location:


Azure Cloud Agent VM to Wazi Sandbox over SSH
Run on agent
Clone on Wazi Sandbox's Unix System Service - using my...
SSH
Build with DBB
SSH
Package DBB Artifacts
SSH
Stage Artifacts on Agent
Bash
Publish DBB Package in an Azure Artifact feed
Universal packages
Publish DBB logs in pipeline drop
Publish build artifacts
Advanced
Control Options
Output Variables

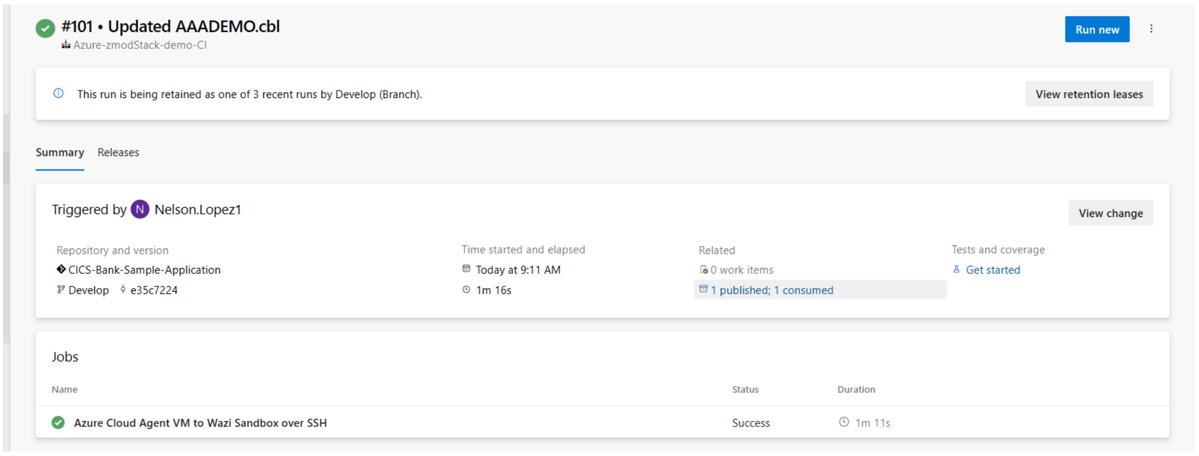
Display name *: Publish DBB logs in pipeline drop

Path to publish *: $(Build.ArtifactStagingDirectory)/dbb-logs

Artifact name *: DBB Logs

Artifact publish location *: Azure Pipelines

When the job is complete, the logs are available from the *publish* link in its summary page.


#101 • Updated AAADEMO.cbl
Azure-zmodStack-demo-CI

This run is being retained as one of 3 recent runs by Develop (Branch).

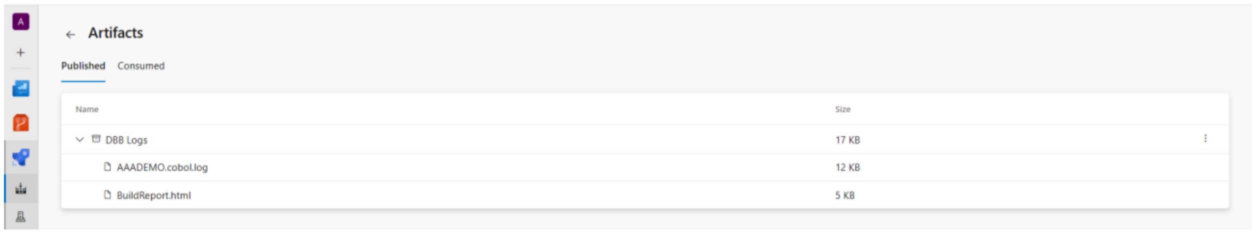
Triggered by Nelson.Lopez1

Repository and version	Time started and elapsed	Related	Tests and coverage
CICS-Bank-Sample-Application	Today at 9:11 AM	0 work items	Get started
Develop e35c7224	1m 16s	1 published; 1 consumed	

Jobs

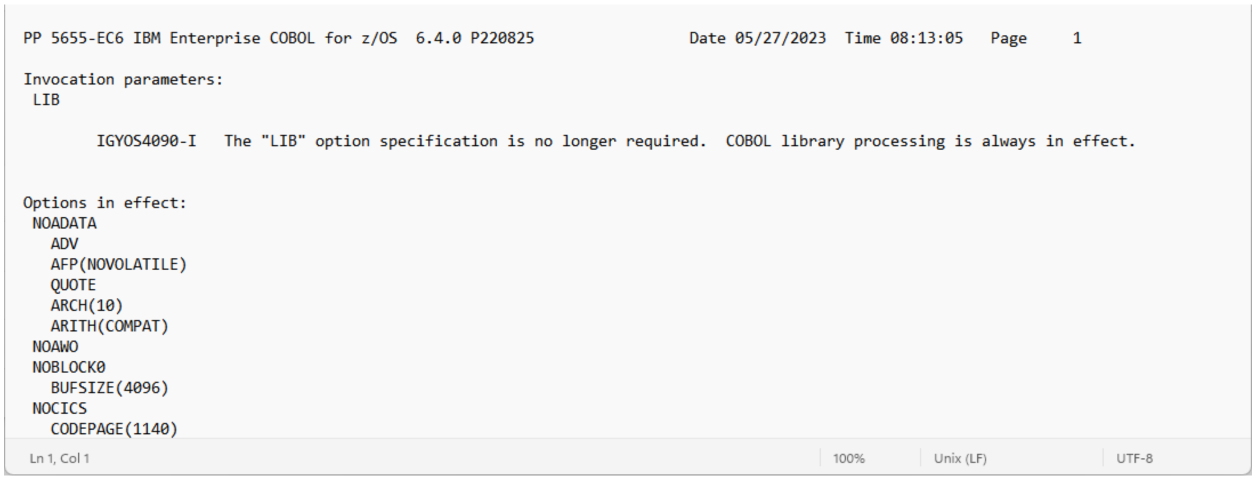
Name	Status	Duration
Azure Cloud Agent VM to Wazi Sandbox over SSH	Success	1m 11s

The contents of the published artifacts:

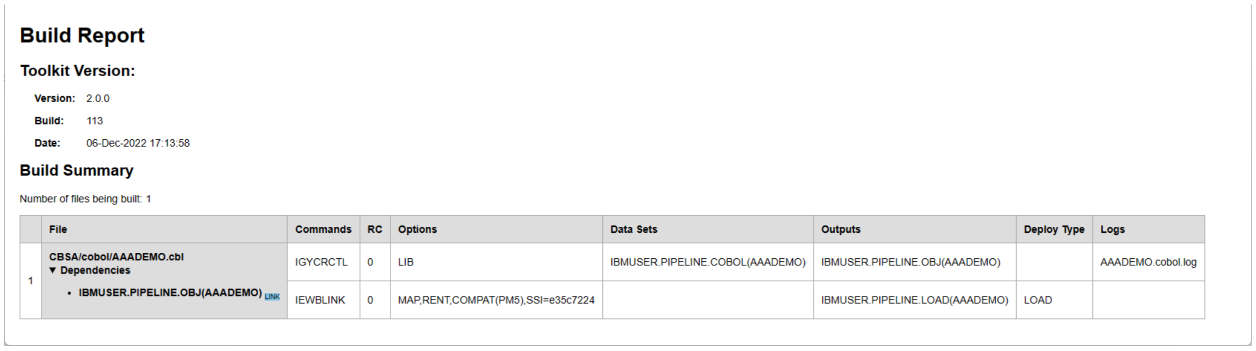

Artifacts

Name	Size
DBB Logs	17 KB
AAADEMO.cobol.log	12 KB
BuildReport.html	5 KB

A sample COBOL compiler and Linkedit SYSPRINT:


```
PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.4.0 P220825                Date 05/27/2023  Time 08:13:05   Page     1

Invocation parameters:
 LIB

        IGYOS4090-I   The "LIB" option specification is no longer required.  COBOL library processing is always in effect.


Options in effect:
 NOADATA
   ADV
   AFP(NOVOLATILE)
   QUOTE
   ARCH(10)
   ARITH(COMPAT)
 NOAWO
 NOBLOCK0
   BUFSIZE(4096)
 NOCICS
   CODEPAGE(1140)
```
Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

A sample DBB BuildReport.html:



Build Report

Toolkit Version:

Version:	2.0.0
Build:	113
Date:	06-Dec-2022 17:13:58

Build Summary

Number of files being built: 1

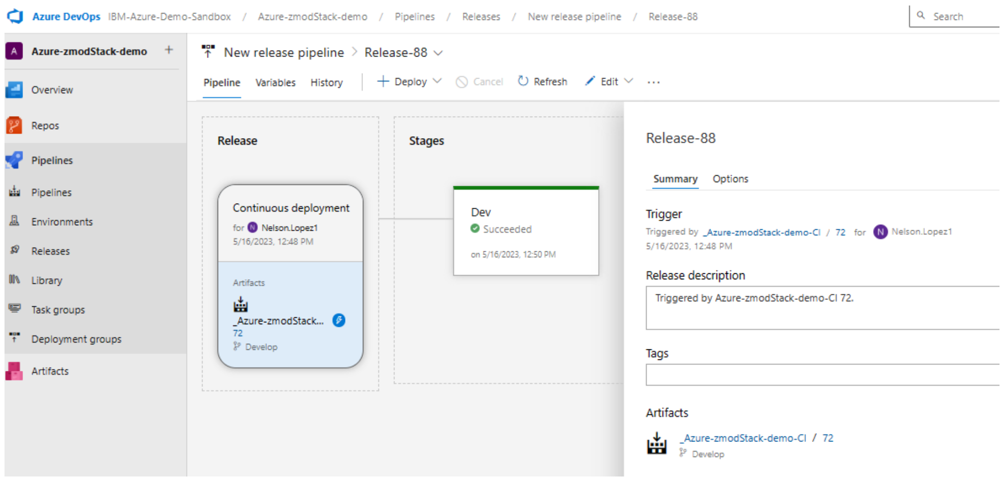
	File	Commands	RC	Options	Data Sets	Outputs	Deploy Type	Logs
1	**CBSA/cobol/AAADEMO.cbl** ▼ **Dependencies** • **IBMUSER.PIPELINE.OBJ(AAADEMO)** LINK	IGYCRCTL	0	LIB	IBMUSER.PIPELINE.COBOL(AAADEMO)	IBMUSER.PIPELINE.OBJ(AAADEMO)		AAADEMO.cobol.log
		IEWBLINK	0	MAP,RENT,COMPAT(PM5),SSI=e35c7224		IBMUSER.PIPELINE.LOAD(AAADEMO)	LOAD	

4.4 Creating the Release pipeline (CD)

An ADO Release pipeline defines the steps to download a package and deploy it to a target z/OS Wazi Sandbox as shown in the example below. This documentation[28] describes how to define a release.

Releases are Continuous Deployment processes that define how packages are deployed in each Stage. Releases can be triggered automatically or manually via an approval policy. They are associated with the Artifact feed of a related application CI pipeline.

Stages consist of tasks to deploy a package to a target environment. In this example, a stage called *Dev* has several tasks to deploy a DBB package to a Wazi sandbox. Stages are normally chained from *Dev* to *QA* to *Prod* (although not shown in this example). Packages can be promoted to upper stages with the proper approvals and quality gates.



Continuous deployment
for Nelson.Lopez1
5/16/2023, 12:48 PM
Artifacts
_Azure-zmodStack...
72
Develop

Dev
Succeeded
on 5/16/2023, 12:50 PM

Release-88

Trigger
Triggered by _Azure-zmodStack-demo-CI / 72 for Nelson.Lopez1
5/16/2023, 12:48 PM

Release description
Triggered by Azure-zmodStack-demo-CI 72.

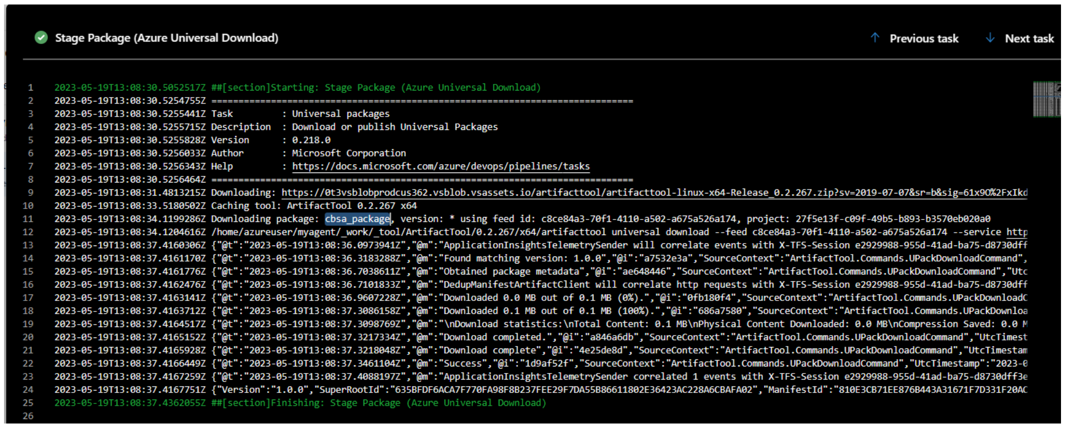
Tags

Artifacts
_Azure-zmodStack-demo-CI / 72
Develop

[28] https://microsoft.github.io/botframework-solutions/solution-accelerators/tutorials/enable-continuous-deployment/2-create-pipeline/

An active or completed Release has a *logs* tab to review progress. Below is an example log of the Dev stage's tasks.

. Clicking on a task, like *Stage Package* below, provides details and any potential error messages.

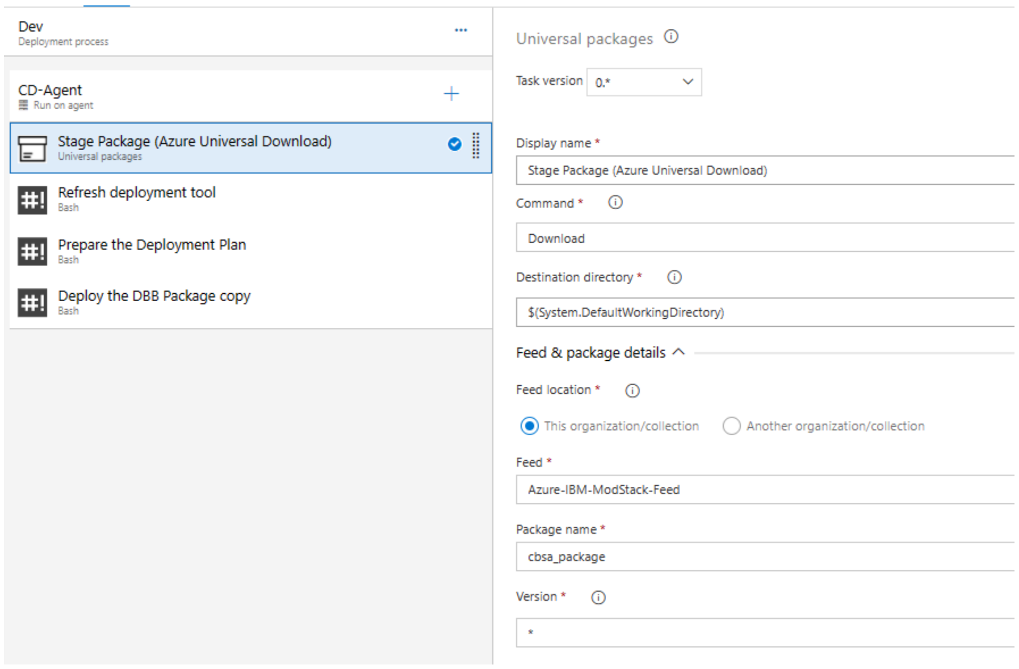


Stage Package (Azure Universal Download)

```
2023-05-19T13:08:30.5052517Z ##[section]Starting: Stage Package (Azure Universal Download)
2023-05-19T13:08:30.5254755Z ==============================================================================
2023-05-19T13:08:30.5255441Z Task         : Universal packages
2023-05-19T13:08:30.5255715Z Description  : Download or publish Universal Packages
2023-05-19T13:08:30.5255828Z Version      : 0.218.0
2023-05-19T13:08:30.5256033Z Author       : Microsoft Corporation
2023-05-19T13:08:30.5256343Z Help         : https://docs.microsoft.com/azure/devops/pipelines/tasks
2023-05-19T13:08:30.5256464Z ==============================================================================
2023-05-19T13:08:31.4813215Z Downloading: https://0t3vsblobprodcus362.vsblob.vsassets.io/artifacttool/artifacttool-linux-x64-Release_0.2.267.zip?sv=2019-07-07&sr=b&sig=61x9O%2FxIkd
2023-05-19T13:08:33.5180502Z Caching tool: ArtifactTool 0.2.267 x64
2023-05-19T13:08:34.1199286Z Downloading package: cbsa_package, version: * using feed id: c8ce84a3-70f1-4110-a502-a675a526a174, project: 27f5e13f-c09f-49b5-b893-b3570eb020a0
2023-05-19T13:08:34.1204616Z /home/azureuser/myagent/_work/_tool/ArtifactTool/0.2.267/x64/artifacttool universal download --feed c8ce84a3-70f1-4110-a502-a675a526a174 --service http
2023-05-19T13:08:37.4160306Z {"@t":"2023-05-19T13:08:36.0973941Z","@m":"ApplicationInsightsTelemetrySender will correlate events with X-TFS-Session e2929988-955d-41ad-ba75-d8730dff
2023-05-19T13:08:37.4161170Z {"@t":"2023-05-19T13:08:36.3183288Z","@m":"Found matching version: 1.0.0","@i":"a7532e3a","SourceContext":"ArtifactTool.Commands.UPackDownloadCommand",
2023-05-19T13:08:37.4161776Z {"@t":"2023-05-19T13:08:36.7038611Z","@m":"Obtained package metadata","@i":"ae648446","SourceContext":"ArtifactTool.Commands.UPackDownloadCommand","Utc
2023-05-19T13:08:37.4162476Z {"@t":"2023-05-19T13:08:36.7101833Z","@m":"DedupManifestArtifactClient will correlate http requests with X-TFS-Session e2929988-955d-41ad-ba75-d8730dff
2023-05-19T13:08:37.4163141Z {"@t":"2023-05-19T13:08:36.9607228Z","@m":"Downloaded 0.0 MB out of 0.1 MB (0%).","@i":"0fb180f4","SourceContext":"ArtifactTool.Commands.UPackDownloadC
2023-05-19T13:08:37.4163712Z {"@t":"2023-05-19T13:08:37.3086158Z","@m":"Downloaded 0.1 MB out of 0.1 MB (100%).","@i":"686a7580","SourceContext":"ArtifactTool.Commands.UPackDownloa
2023-05-19T13:08:37.4164517Z {"@t":"2023-05-19T13:08:37.3098769Z","@m":"\nDownload statistics:\nTotal Content: 0.1 MB\nPhysical Content Downloaded: 0.0 MB\nCompression Saved: 0.0 M
2023-05-19T13:08:37.4165152Z {"@t":"2023-05-19T13:08:37.3217334Z","@m":"Download completed.","@i":"a846a6db","SourceContext":"ArtifactTool.Commands.UPackDownloadCommand","UtcTimest
2023-05-19T13:08:37.4165928Z {"@t":"2023-05-19T13:08:37.3218048Z","@m":"Download complete","@i":"4e25de8d","SourceContext":"ArtifactTool.Commands.UPackDownloadCommand","UtcTimestam
2023-05-19T13:08:37.4166449Z {"@t":"2023-05-19T13:08:37.3461104Z","@m":"Success","@i":"1d9af52f","SourceContext":"ArtifactTool.Commands.UPackDownloadCommand","UtcTimestamp":"2023-0
2023-05-19T13:08:37.4167259Z {"@t":"2023-05-19T13:08:37.4088197Z","@m":"ApplicationInsightsTelemetrySender correlated 1 events with X-TFS-Session e2929988-955d-41ad-ba75-d8730dff3e
2023-05-19T13:08:37.4167751Z {"Version":"1.0.0","SuperRootId":"635BFDF6ACA7F770FA98F8B237FEE29F7DA55B86611802E36423AC228A6CBAFA02","ManifestId":"810E3CB71EE876B443A31671F7D331F20AC
2023-05-19T13:08:37.4362055Z ##[section]Finishing: Stage Package (Azure Universal Download)
```

4.4.1 Staging Packages for Deployment

Stage Package is a *Universal package* task to run the Download command on an Agent. In this case, the latest version of the package created by the related CI pipeline is copied to a location defined by *Destination directory* on the Agent's file system. The Agent can be the same one used in the CI phase or within a pool of available agents with similar capabilities.



Task version: 0.*

Display name *: Stage Package (Azure Universal Download)

Command *: Download

Destination directory *: $(System.DefaultWorkingDirectory)

Feed & package details

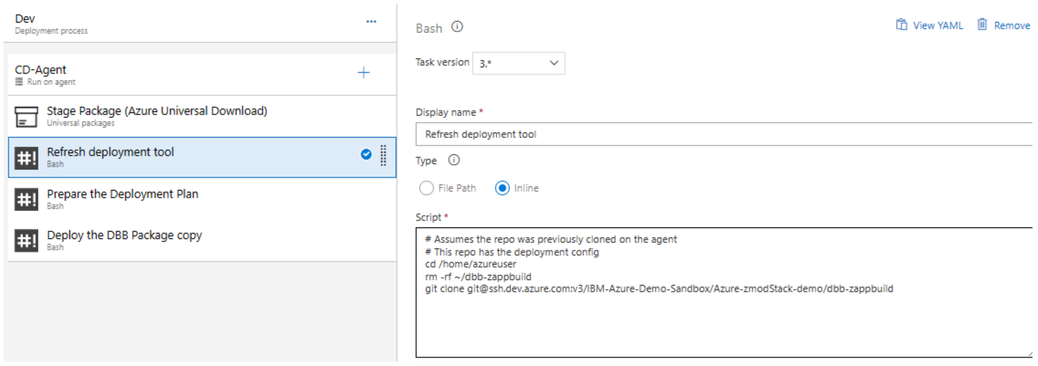
Feed location *: [✓ This organization/collection] Another organization/collection

Feed *: Azure-IBM-ModStack-Feed

Package name *: cbsa_package

Version *: *

Refresh deployment tool is a custom inline bash script to initialize the IBM Wazi Deploy configuration on the Agent. In this case, the deployment tools are cloned from an administrative repository. But they can be pre-installed on the agent.



Task version 3.*

Display name * Refresh deployment tool

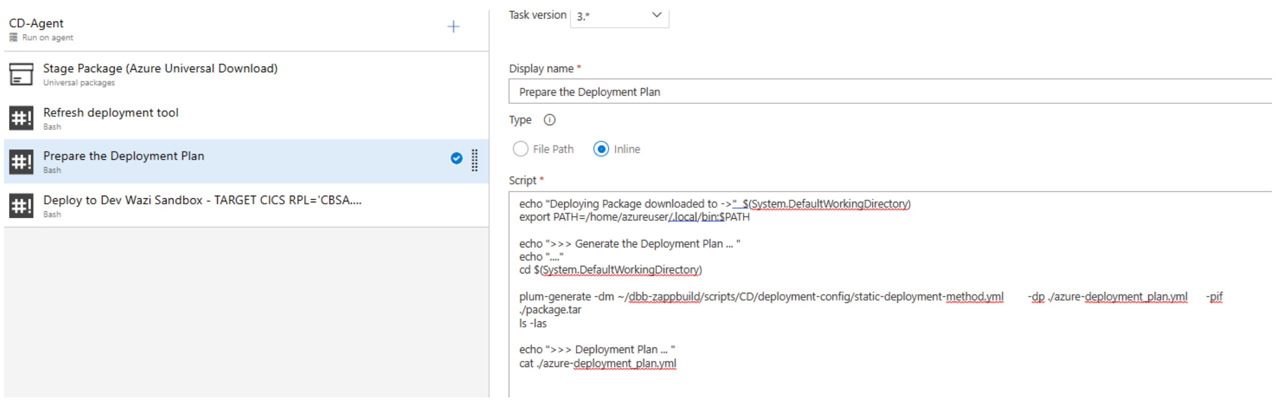
Type: File Path [✓ Inline]

Script *

```
# Assumes the repo was previously cloned on the agent
# This repo has the deployment config
cd /home/azureuser
rm -rf ~/dbb-zappbuild
git clone git@ssh.dev.azure.com:v3/IBM-Azure-Demo-Sandbox/Azure-zmodStack-demo/dbb-zappbuild
```

The last 2 tasks, *Prepare* and *Deploy*, leverage IBM Wazi Deploy's Ansible playbooks. They unpack the package and deploy artifacts to a target environment (Ansible managed node). Actions to handle CICS, DB2 and standard batch programs are all supported.

The *Prepare the Deployment Plan* task generates a Wazi Deploy deployment plan on the agent.


CD-Agent
Run on agent
Stage Package (Azure Universal Download)
Universal packages
Refresh deployment tool
Bash
Prepare the Deployment Plan
Bash
Deploy to Dev Wazi Sandbox - TARGET CICS RPL='CBSA....
Bash

Task version 3.*

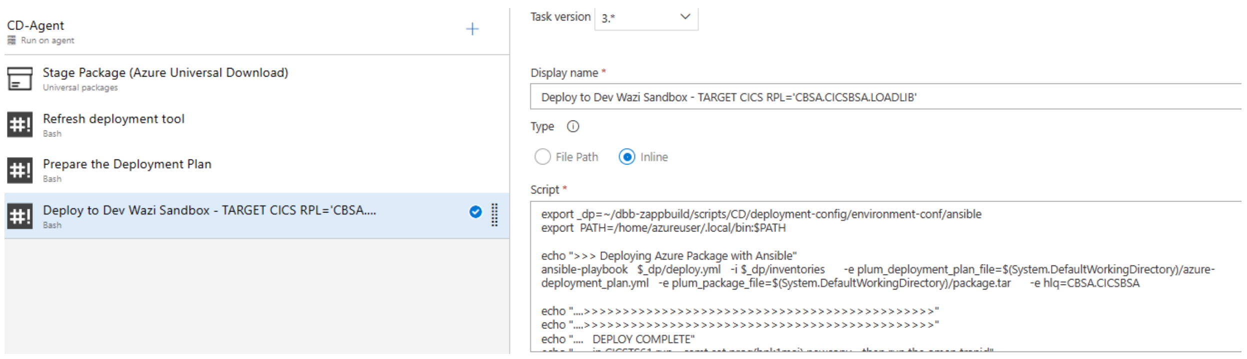
Display name * Prepare the Deployment Plan

Type: File Path [✓ Inline]

Script *

```
echo "Deploying Package downloaded to ->" $(System.DefaultWorkingDirectory)
export PATH=/home/azureuser/.local/bin:$PATH

echo ">>> Generate the Deployment Plan ... "
echo "...."
cd $(System.DefaultWorkingDirectory)

plum-generate -dm ~/dbb-zappbuild/scripts/CD/deployment-config/static-deployment-method.yml      -dp ./azure-deployment_plan.yml     -pif
./package.tar
ls -las

echo ">>> Deployment Plan ... "
cat ./azure-deployment_plan.yml
```

The *Deploy to Dev Wazi Sandbox* task uses the generated Deployment Plan and downloaded package from Azure Artifacts to run the Ansible playbook against the target Wazi Sandbox.


CD-Agent
Run on agent
Stage Package (Azure Universal Download)
Universal packages
Refresh deployment tool
Bash
Prepare the Deployment Plan
Bash
Deploy to Dev Wazi Sandbox - TARGET CICS RPL='CBSA....
Bash

Task version 3.*

Display name * Deploy to Dev Wazi Sandbox - TARGET CICS RPL='CBSA.CICSBSA.LOADLIB'

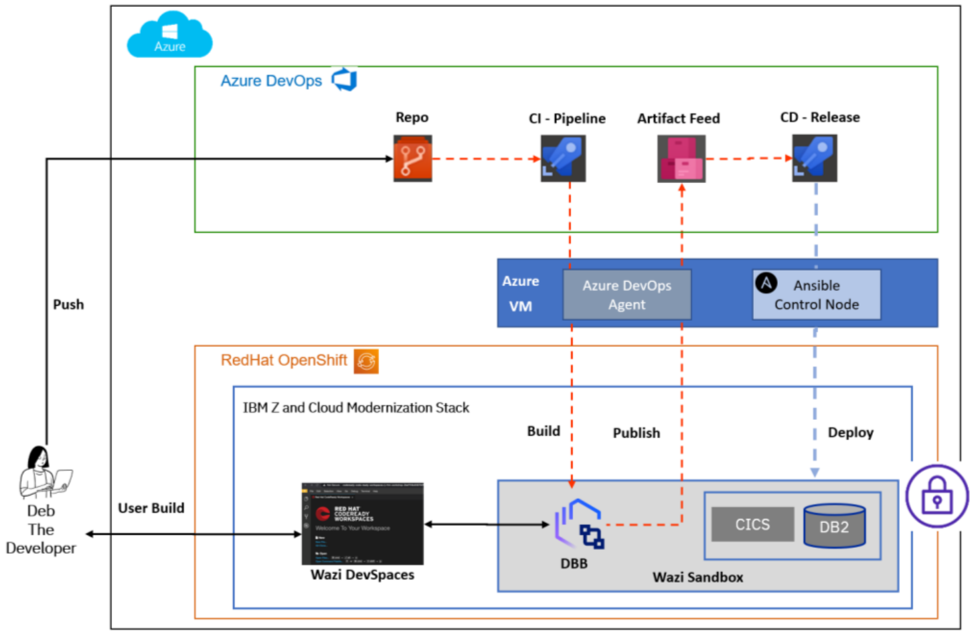
Type: File Path [✓ Inline]

Script *

```
export _dp=~/dbb-zappbuild/scripts/CD/deployment-config/environment-conf/ansible
export  PATH=/home/azureuser/.local/bin:$PATH

echo ">>> Deploying Azure Package with Ansible"
ansible-playbook   $_dp/deploy.yml   -i $_dp/inventories      -e plum_deployment_plan_file=$(System.DefaultWorkingDirectory)/azure-
deployment_plan.yml   -e plum_package_file=$(System.DefaultWorkingDirectory)/package.tar      -e hlq=CBSA.CICSBSA

echo "....>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>"
echo "....>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>"
echo "....   DEPLOY COMPLETE"
```

4.5 Azure DevOps End-to-End

The diagram below outlines the very basic components of the CI/CD solution described in this section. It highlights an Azure Environment with Azure DevOps and its core services, a RedHat OpenShift Cluster that hosts an IBM Z and Cloud Modernization Stack which contains the build, publish, and deploy components.



For more details on the developer flow, refer to the section 2.3 Setting up Wazi for Dev Spaces and Wazi Analyze which is common across all IBM supported Cloud providers.

5 Conclusion

This document provides recipes and showcases how to integrate the Wazi components of the IBM Z Cloud and Modernization Stack (Wazi Dev Spaces, Wazi Sandbox, Wazi Deploy and Wazi Analyze) to provide a Cloud-native experience for developers and administrators using services that can be provisioned in minutes.

The examples provided throughout this document can serve as a baseline to implement your own DevOps toolchain leveraging Wazi components. Some customizations will be needed to meet your SDLC best practices and workflows.

This document also showcases an implementation of the Wazi tools on the AWS Cloud and Azure Cloud platforms. However, they can be used as a reference across all supported IBM cloud providers.

The purpose of leveraging Cloud services is to modernize the Software Delivery Lifecycle of mainframe applications, by providing isolated development and test runtimes for more flexibility and agility in development practices.