**BUILD ONCE DEPLOY MANY**

**Table Of Content:**

Introduction……………………………………………………………………………………………….…………….3

Prerequisites……………………………………………………………………………………….………………….…4

Implementation of continuous Integration……………………………………………………………………………….4

Implementation of continuous Deployment…………………………………………………………………………….12

**Introduction:**

"Build once, deploy many" is a software development and deployment strategy that emphasizes creating a single build or package of your application and deploying it to multiple environments or platforms.

The key idea behind "build once, deploy many" is to avoid rebuilding your application for each deployment target. Instead, you create a single build that can be used across various environments, such as development, testing, staging, and production.

Some Benefits of Build Once, deploy many are Reduced Errors, Efficiency, Consistency etc.

Implementing Build once, deploy many in Azure DevOps. Azure DevOps is a set of development tools and services offered by Microsoft. Azure DevOps provides set of capabilities for software development, including version control, continuous integration, and continuous deployment (CI/CD), project management, and collaboration tools. It's designed to support the entire software development lifecycle from planning and coding to testing, deployment, and monitoring. Azure DevOps helps to automate their workflows, ensuring that software projects are developed and delivered more efficiently.

Implementing the "build once, deploy many" approach in Azure DevOps offers several advantages:

Azure DevOps is tightly integrated with the Azure cloud ecosystem, making it well-suited for deployments to Azure services such as Azure App Service, Azure Container Registry and Azure Kubernetes Service (AKS). This integration can simplify and streamline the deployment process when targeting Azure-specific resources.

It offers a wide range of features for building, testing, and deploying applications, all in a single tool. Azure Pipelines is particularly well-suited for multi-stage deployments.

Can easily build, manage, and deploy containerized applications using Azure DevOps.

Azure DevOps provides built-in support for managing variables and configuration settings. You can easily manage environment-specific configurations and secrets.

**Pre-Requisites:**

* Azure DevOps Account
* Azure Cloud Account

**Implementation of Continuous Integration:**

Log in to your Azure DevOps account and then create a new project by clicking on the 'New Project' icon.

A screenshot of a computer

Description automatically generated

By clicking on the 'New Project' icon, you can create a new project. You'll need to provide a project name and description, and then choose the project type either public or private based on your requirements. Afterward, click on the 'Create' icon.

A screenshot of a computer

Description automatically generated

After creating the project, it will be displayed on the Azure DevOps Dashboard with the given name. To view an overview of the created project, select the project on the Dashboard.

A computer screen shot of a person working on a computer

Description automatically generated

On the left side of the Azure Dashboard in the project, you can find the 'Repos' icon as shown in below image. Select the 'Repos' icon and then choose 'Files icon'. You can import or clone source code by clicking on the 'Import' icon below 'Import a repository.' Provide the path of the source code (e.g., https://github.com/organizationName/ProjectName.git) in the 'Clone URL' field, and then select 'Import.

**project 🡪 Repos 🡪Files 🡪Import/clone (Import the source code) 🡪 provide the path of the Source code URL(**[**https://github.com/organisationName/ProjectName.git**](https://github.com/organisationName/ProjectName.git)**) 🡪 Create.**  
  
The REPOS icon on the Azure DevOps dashboard is used to provide quick access to our source code  
repositories.

A person sitting at a desk with a computer

Description automatically generated

The REPOS icon on the Azure DevOps dashboard is used to provide quick access to our source code  
repositories.

A screenshot of a computer

Description automatically generated

On the Left side of the Azure DevOps Dashboard in the project we can find the Pipeline icon, select the pipeline icon, then select the pipelines and then select Create Pipeline as shown in below image.

A screenshot of a computer

Description automatically generated

**Project 🡪 Pipelines 🡪 Create Pipeline**

A screenshot of a computer

Description automatically generated

After selecting the Create Pipeline, select the source code Repo, where is our code (Azure Repos Git/ Bitbucket Cloud/ GitHub).

**Project 🡪 Pipelines 🡪 Create Pipeline 🡪Azure Repos Git**

A screenshot of a computer

Description automatically generated

**Project 🡪 Pipelines 🡪 Create Pipeline 🡪 Azure Repos Git 🡪 select Repository 🡪 Configure your Pipeline.**

While configuring the pipeline select the respective tool/ starter pipeline which requires for the project, for the React project select the npm as the Build tool depends on the requirement and then click on save and run.

A screenshot of a computer

Description automatically generated

After selecting save and run, the pipeline will be in Runing, if the pipeline runs successfully, it will display the job success as shown below.

A screenshot of a computer

Description automatically generated

If the pipeline fails it will display job failed, some common reasons for the failure of the pipeline: Build Errors, Test Failures, Dependency Issues, Permission Issues, Insufficient Resources, Timeouts, Environment configuration, Environment Variables, Script Errors, etc.…

A screenshot of a computer

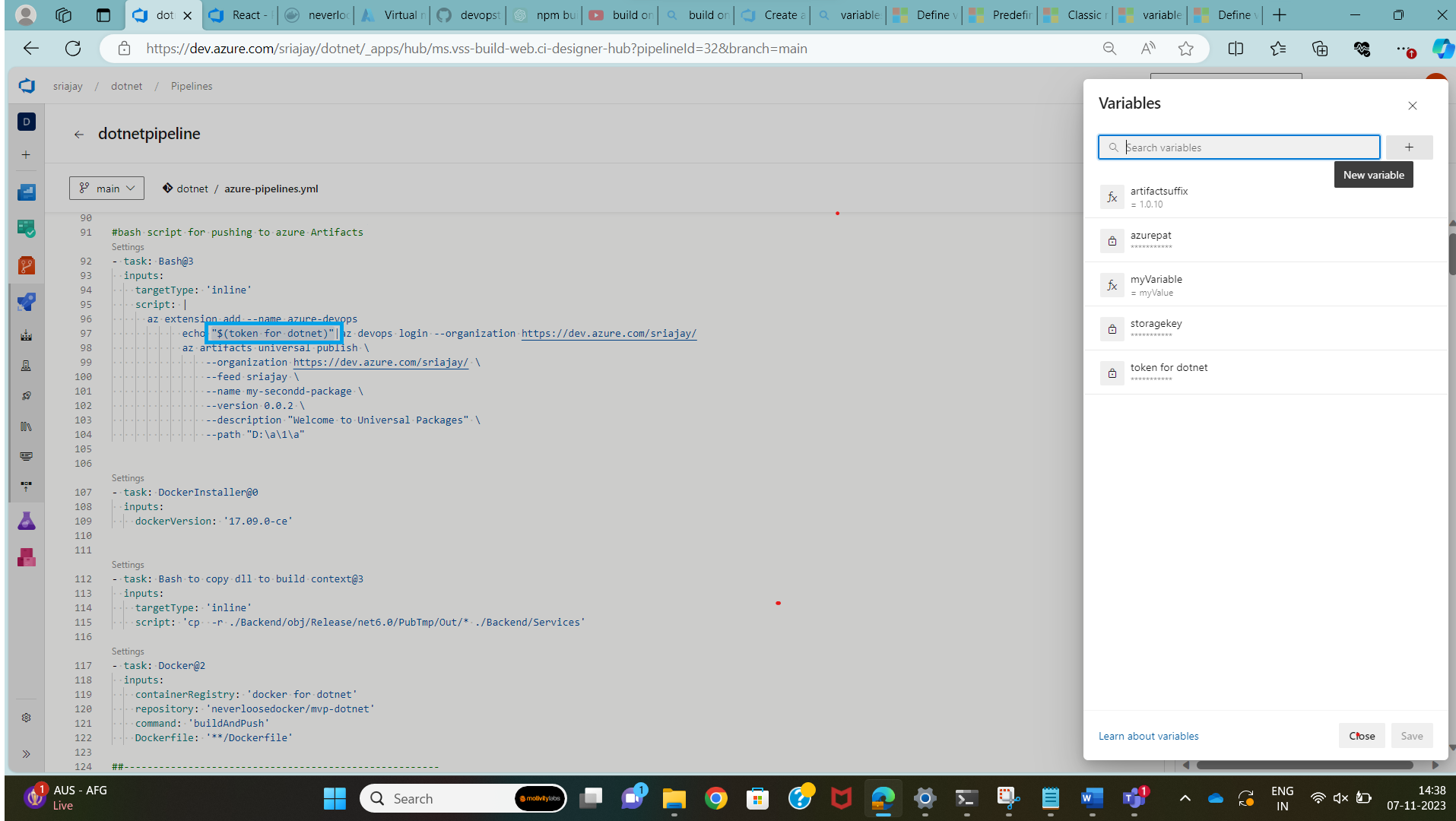
Description automatically generated

In Azure DevOps, **Variables** are used to store and manage data that can be used within our pipelines and releases. Variables allow us to parameterize your pipelines, making them more flexible and reusable. There are two types of variables in Azure DevOps: pipeline variables and group variables.

Variables: Pipeline variables are defined at the pipeline level and are specific to a single pipeline. They can be used to store data that is unique to a particular pipeline.

We can create variables by selecting variable icon which is present beside Run icon while editing the pipeline.

**Pipeline 🡪 select pipeline 🡪Edit 🡪Variables 🡪 New variable (+ icon) 🡪Enter the Name of the Variable 🡪 Enter the Value of the Variable 🡪 OK.**



For accessing the created Variable in the pipeline, we can access it by “$(variablename)” if required we can mask the value of the variable by using the Lock symbol which is present beside the value of the variable.

Group Variables: Group variables are defined at the library group level and can be shared across multiple pipelines within the same project. They are typically used for storing data that is common to multiple pipelines, such as secrets, credentials, or configuration settings that are used by various pipelines.

Pipelines 🡪 Library 🡪 + variable group 🡪 Variable group name 🡪 Description 🡪 +Add (Variables) 🡪 Enter the name of the variable 🡪 Enter the value of the Variable.

A screenshot of a computer

Description automatically generated

If we want to add more tasks to our pipeline, then we need to click on Edit and select the required task on the right side of the Azure DevOps Dashboard we can add any task depending on our requirements i.e., Bash script/ Copy files/ Copy and Publish Build Artifacts/ Docker etc. by clicking on the Add Icon, save it and then Run.

A screenshot of a computer

Description automatically generated

After the completion of the build stage successfully we can add the task “Publish build artifacts to Azure Pipeline” and then add another task for pushing the artifacts to Azure Artifacts with the reference of universal package (Artifacts – connect to feed – Universal Packages – Publish package) by adding Bash Script, the Bash Script look as below.

- task: Bash@3

inputs:

targetType: 'inline'

script: |

az extension add --name azure-devops

echo "$(token for dotnet)"|az devops login --organization https://dev.azure.com/sriajay/

az artifacts universal publish \

--organization https://dev.azure.com/sriajay/ \

--feed sriajay \

--name my-secondd-package \

--version 0.0.2 \

--description "Welcome to Universal Packages" \

--path .

A screenshot of a computer

Description automatically generated

After adding Multiple stages to the pipeline and running it, if all the stages are running successfully the job will be looks as shown below.

A screenshot of a computer

Description automatically generated

All the above stages in the pipeline represent Continuous Integration.

**Implementation of Continuous Deployment:**

For the Deployment Process we need to create a Release pipeline. The Releases pipeline icon will be present below the pipeline icon.

A screenshot of a computer

Description automatically generated

Create a virtual machine in the azure cloud to deploy our application. To connect the Azure DevOps with Azure Cloud we need to create Service connection.

Create a new service connection to the Virtual machine which is hosted on the azure cloud.

Create a service connection to a VM in Azure DevOps, go to Project Settings, Service Connections, New Service Connection, and select Azure Resource Manager. Provide VM details and credentials for the connection.

A screenshot of a computer

Description automatically generated

Create a new release pipeline for the Continuous Deployment. Select the release icon from the pipeline dropout.

**Releases 🡪 New Pipeline 🡪 Empty Job 🡪 Artifact**

A screenshot of a computer

Description automatically generated

Select the artifact from the source type where the artifact is stored.

A white rectangular object with a black line

Description automatically generated

After selecting the artifact add the required stage depend up on our requirements.

In Azure DevOps, create a release pipeline with two stages representing two environments. Configure each stage to copy the artifact and set environment-specific variables, then deploy the application for streamlined multi-environment deployment.

A screenshot of a computer

Description automatically generated

In Azure DevOps, use the "Copy Files Over SSH" task in a release pipeline to securely copy files to a server via SSH, providing server details and credentials for a reliable file transfer process.

To deploy the same artifact to multiple environments, copy the artifact to the server and create a new .env file using the command:

sudo REACT\_APP\_API\_URL=https://dummy-api-url.com \

REACT\_APP\_REDIRECT\_URL=https://dummy-redirect-url.com \

REACT\_APP\_LOGIN\_URL=https://dummy-login-url.com \

REACT\_APP\_REDIRECT\_URL\_DUP=https://dummy-redirect-url-dup.com \

REACT\_APP\_LOGOUT\_URL=https://dummy-logout-url.com \

REACT\_APP\_RESPONSETYPE=code \

REACT\_APP\_MUI\_DATA\_GRID\_LICENSE\_KEY=OTI5NDUsUz1wcm8sTE09c3Vic2NyaXB0aW9uLEtWPTI= \

npx react-inject-env set

Then, start the application using:

sudo pm2 serve --spa --name dev --update-env.

Paste the above commands in bash script and then run in the server using the “RUN SHELL COMMANDS ON THE REMOTE MACHINE” task.

A screenshot of a computer

Description automatically generated

The provided command creates a new .env file with custom key-value pairs. It allows the application to run with specific environment configurations. The application is up and running with the specified environment.

A computer screen shot of a computer code

Description automatically generated

**THANK YOU**