**Implementation of CI pipeline for React in Azure DevOps**

Contents

[Introduction: 3](#_Toc151124020)

[Pre-Requisites: 4](#_Toc151124021)

[Implementation of Continuous Integration: 4](#_Toc151124022)

[Implementation of Building the Application task in CI: 7](#_Toc151124023)

[Implementation of SonarQube task in CI: 9](#_Toc151124024)

[Implementation of OWASP Dependency Check task in CI: 10](#_Toc151124025)

[Implementation of “Pushing the Artifact to the Azure Artifact” task in CI: 11](#_Toc151124027)

[Implementation of Docker task in CI: 12](#_Toc151124028)

[Variables: 14](#_Toc151124029)

[Templates: 15](#_Toc151124030)

# Introduction:

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.

Key components of Azure DevOps include:

Azure Boards: A work tracking system that helps teams plan, track, and discuss work across the entire development process. It includes features for backlog management, sprint planning, and customizable dashboards.

Azure Repos: A version control system that supports both Git and Team Foundation Version Control (TFVC). It allows teams to manage and track changes to their codebase.

Azure Pipelines: A continuous integration and continuous delivery (CI/CD) platform that enables teams to automate the build, test, and deployment of their applications.

Azure Test Plans: A testing tool that helps teams plan, track, and manage testing efforts.

Azure Artifacts: A package management system that allows teams to create, host, and share packages. This includes NuGet, npm, and Maven packages, among others.

In this Continuous Integration pipeline, we are implementing the following stages for the React project:

* Build
* Static code Analysis (SonarQube)
* OWASP Dependency check
* Copy and Publish Artifact
* Pushing Artifact to Azure Artifacts
* Docker stage

# Pre-Requisites:

* Azure DevOps Account
* Cloud Account
* React Source Code

# Implementation of Continuous Integration:

Follow the below steps to create Continuous Integration pipeline. Log in to 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 screenshot of 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

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. In our Continuous integration pipeline first, we will address the Build tool, for the React Project select node as the Build tool and then click on save and run.

## Implementation of Building the Application task in CI:

The YAML Pipeline for the building the React Application looks as below:

A computer screen shot of a computer screen

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. If the pipeline fails, it will display the job failed as shown below.

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 success of the job “Build” add Static Code Analysis (SonarQube) stage.

## Implementation of SonarQube task in CI:

**SonarQube:**

SonarQube is an open-source platform for continuous inspection of code quality. It performs static code analysis to automatically detect various code smells, bugs, and security vulnerabilities in your source code. The goal is to provide developers and teams with insights into the overall health and quality of their codebase.

Key features of SonarQube include:

* Static Code Analysis
* Code Smells
* Security Vulnerabilities
* Code Duplication
* Metrics and Reporting

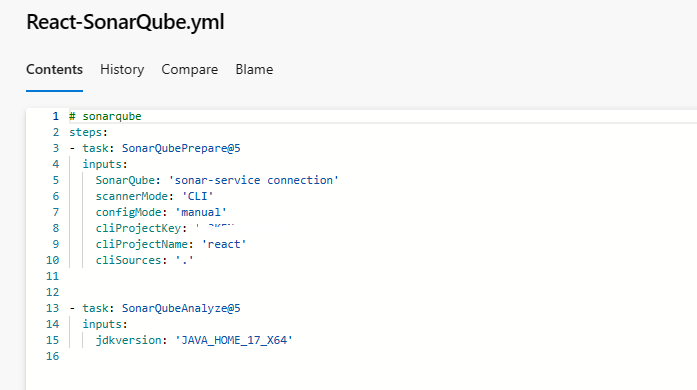
The SonarQube stage involves analysing the source code, sending the results to the SonarQube server, and obtaining a detailed analysis report.

Add Static Code Analysis (SonarQube) stage by selecting “Prepare SonarQube analysis configuration” stage from the show assistant icon which is present on the right side of the pipeline page.

A screenshot of a computer

Description automatically generated

After passing the required inputs for the task “Static Code Analysis (SonarQube)” and then click on Add, then save and run the pipeline by selecting the run icon. The YAML pipeline looks as below.



After the successful running of the Static Code Analysis (SonarQube) stage then add “OWASP Dependency check” stage from the show assistant icon.

## Implementation of OWASP Dependency Check task in CI:

**OWASP Dependency Check:**

OWASP Dependency-Check is a tool that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities. It automatically scans the project dependencies for security issues. The goal is to identify and remediate vulnerabilities early in the development process, reducing the risk of deploying software with known security flaws.

By integrating OWASP Dependency-Check into your Azure DevOps pipeline, you can automate the process of identifying and addressing security vulnerabilities in your project dependencies, contributing to a more secure software development lifecycle.

After passing the required inputs for the task then click on Add, then save and run the pipeline by selecting the run icon. The YAML pipeline looks as below.

A white rectangular object with a white border

Description automatically generated

After the successful running of the “OWASP Dependence Check” stage the output of OWASP Dependence check:

A screenshot of a computer

Description automatically generated

After the successful running of the “OWASP Dependence check” stage then add Bash task from the show assistant icon for “Pushing the Artifact to the Azure Artifact” with the reference of universal package (Artifacts – connect to feed – Universal Packages – Publish package) by adding Bash Script, the Bash Script look as below.

## Implementation of “Pushing the Artifact to the Azure Artifact” task in CI:

The YAML pipeline looks as below for the implementation of Pushing the Artifact to the Azure Artifact:

A screenshot of a computer

Description automatically generated

After the successful running of the “Bash Script” the output for “Pushing the Artifact to the Azure Artifact” in the Azure Artifact:

A screenshot of a computer

Description automatically generated

After the successful running of the “Bash Script” for “Pushing the Artifact to the Azure Artifact” add the Docker task.

## Implementation of Docker task in CI:

The YAML pipeline for the implementation of Docker looks as follows:

A screenshot of a computer

Description automatically generated

After the successful running of the Docker task, the docker image will be stored in the docker hub repository:

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

# 

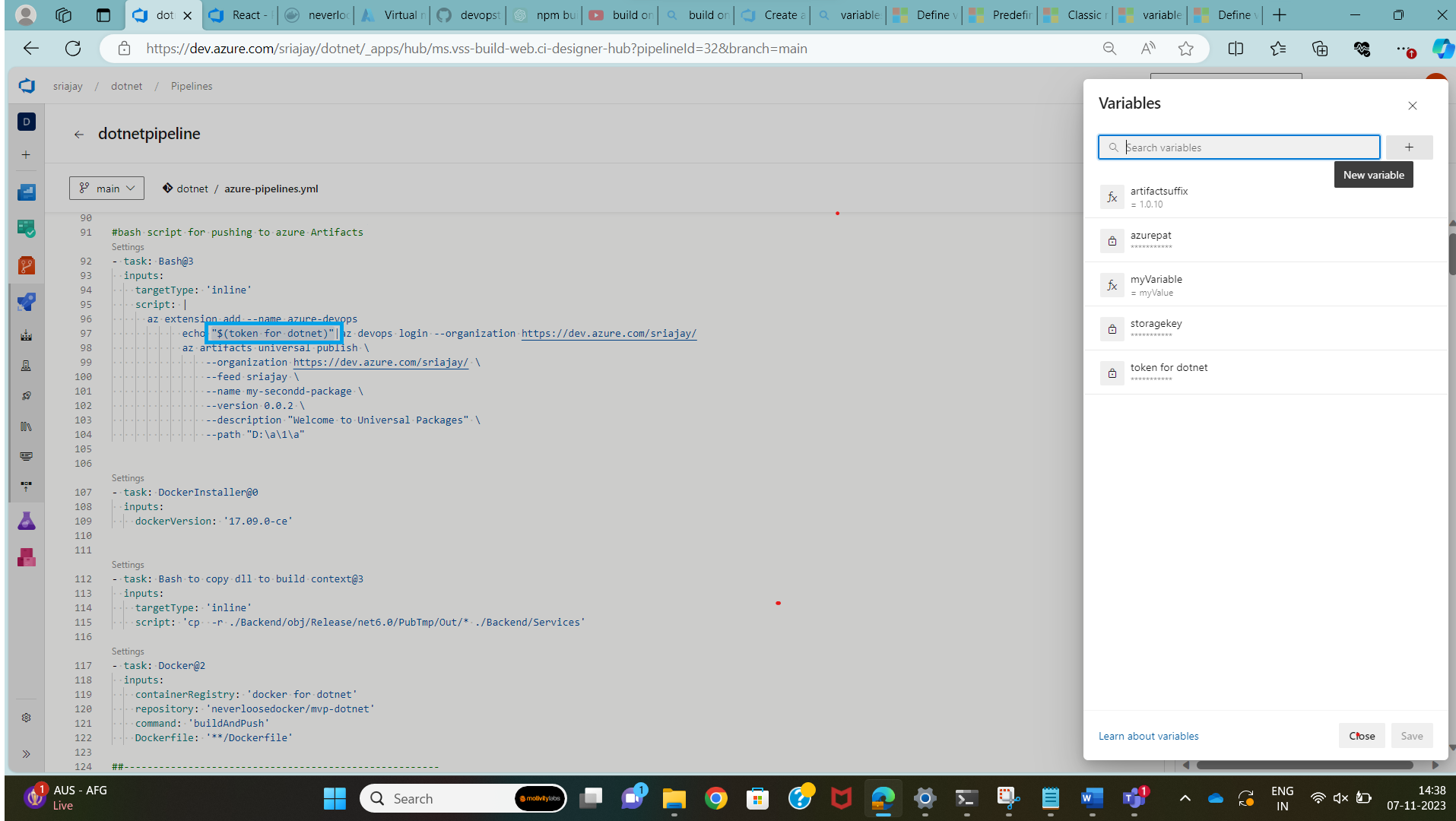
# Variables:

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.

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.

# Templates:

In Azure DevOps, templates refer to defining and reusing parts of our build and release pipeline definitions. Templates allow us to abstract common tasks, stages, or even entire pipelines into reusable components. This can significantly reduce duplication of code, simplify maintenance, and improve consistency across different projects or environments. Templates support parameterization, allowing us to customize their behaviour based on the specific needs of different projects or environments.

some key aspects of templates in Azure DevOps and their importance are Reusability, Consistency, Parameterization.

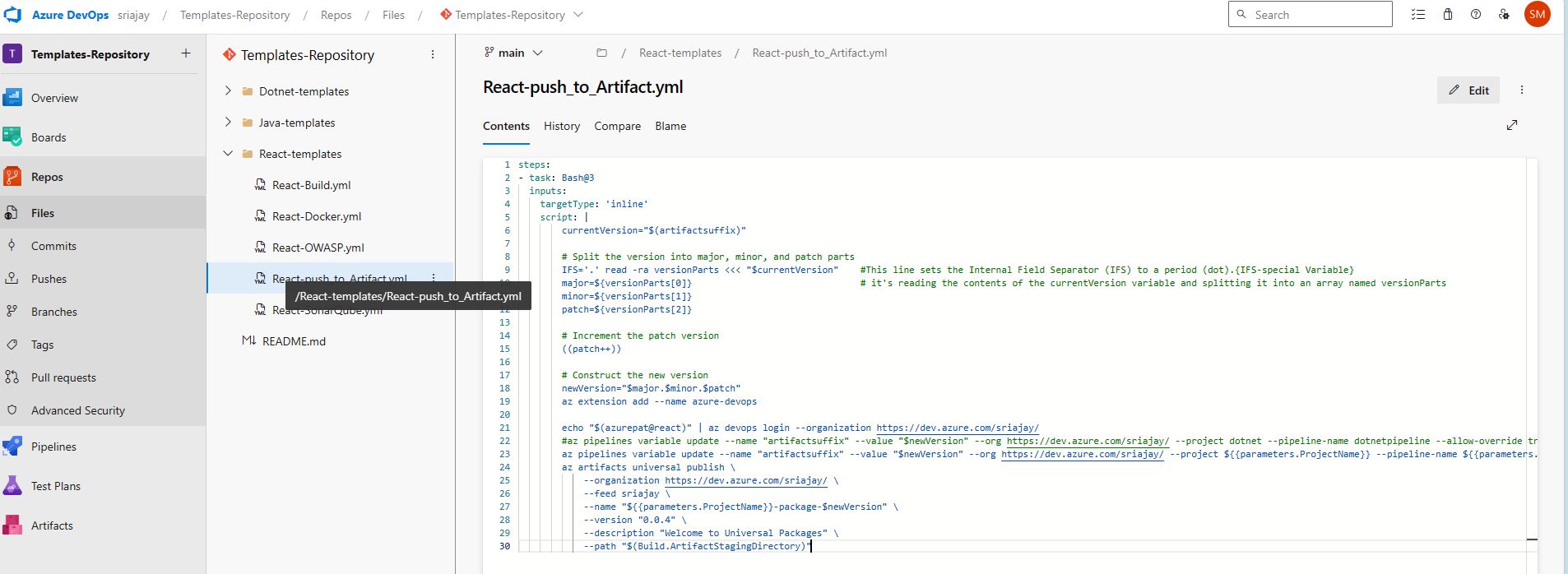
we can copy content from one YAML and reuse it in a different YAML. Copying content from one YAML to another saves us from having to manually include the same logic in multiple places.

The Actual pipeline will look as below if we include all the stages without using the templates:

A computer screen shot of a computer

Description automatically generated

Create a new repository in Azure DevOps, establish a folder within the repository, and subsequently generate a file within that folder to store the template.



After creating Templates for all the stages, use the created templates in the pipeline, pipeline will be looks as:

A screenshot of a computer

Description automatically generated