Q1 - SCENARIO

A car rental company called FastCarz has a .net Web Application and Web API which are recently migrated from on-premise system to Azure cloud using Azure Web App Service
and Web API Service.
The on-premises system had 3 environments Dev, QA and Prod.
The code repository was maintained in TFS and moved to Azure GIT now. The TFS has daily builds which
triggers every night which build the solution and copy the build package to drop folder.
deployments were done to the respective environment manually. The customer is planning to setup Azure DevOps Pipeline service for below requirements:
1) The build should trigger as soon as anyone in the dev team checks in code to master branch.
2) There will be test projects which will create and maintained in the solution along the Web and API.
The trigger should build all the 3 projects - Web, API and test.
The build should not be successful if any test fails.
3) The deployment of code and artifacts should be automated to Dev environment.
4) Upon successful deployment to the Dev environment, deployment should be easily promoted to QA
and Prod through automated process.
5) The deployments to QA and Prod should be enabled with Approvals from approvers only.
Explain how each of the above the requirements will be met using Azure DevOps configuration.
Explain the steps with configuration details.

Please find a solution to this scenario below:

1) Please see yaml file that contains a **build trigger for every master branch commit**
https://github.com/bolvinfernandes/MaerskDevopsTest/blob/main/FastCarz/pipelines/release-build.yml

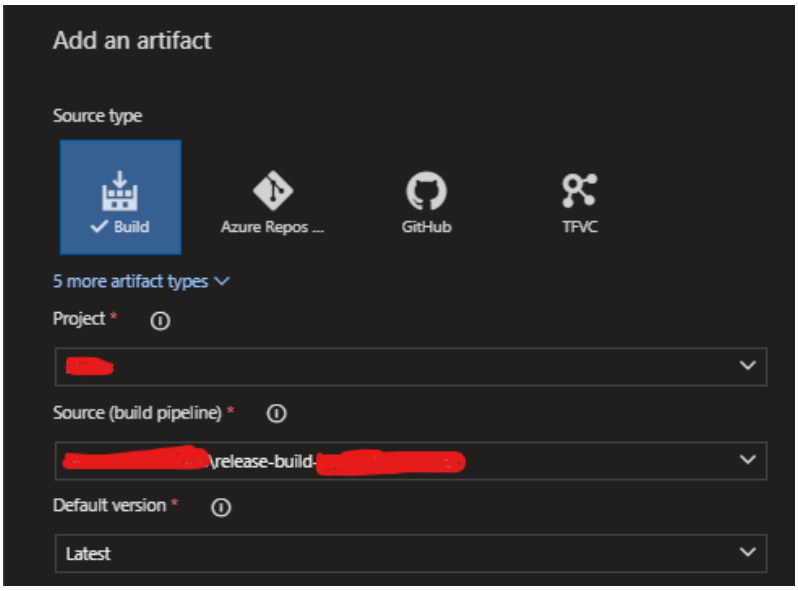
```yaml
# trigger build on commit to master
trigger:
  branches:
    include:
    - refs/heads/master
```

2) Please see above yaml file
dotnet build task will build all (3) projects – input passed to build step is ****/*.csproj**
dotnet test task will build ****/*.csproj** under folder named testing/Testing (assuming such a folder exists)

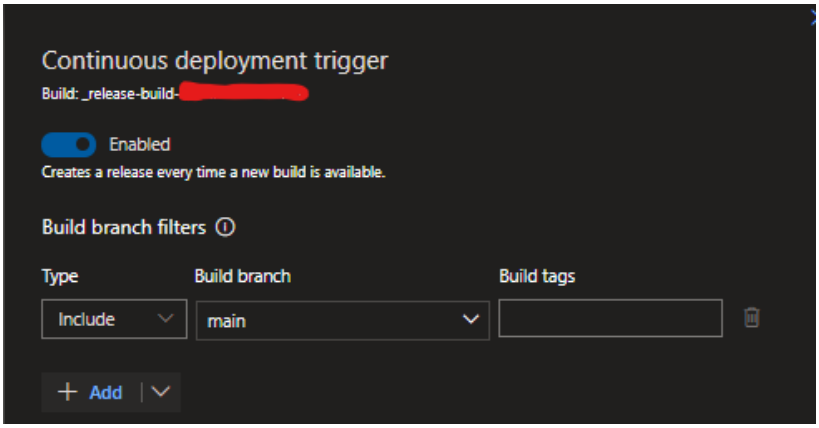
```yaml
steps:
- checkout: self
  clean: true
- task: DotNetCoreCLI@2
  displayName: dotnet build #perform dotnet build that does an implicit restore
  inputs:
    command: build
    projects: '**/*.csproj' #build 3 projects i.e. Web,API and test in one step
- task: DotNetCoreCLI@2
  displayName: dotnet test #perform dotnet test to see if any test fails
  inputs:
    command: test
    projects: '**/*[Tt]esting/*.csproj' #assuming there's a Testing folder where the test csproj sits
    arguments: --configuration $(buildConfiguration) --collect "Code coverage" --no-restore #avoid duplication of restore as it's run implicitly as part of dotnet build and
```

3) Automated release creation and deployment to Dev environment

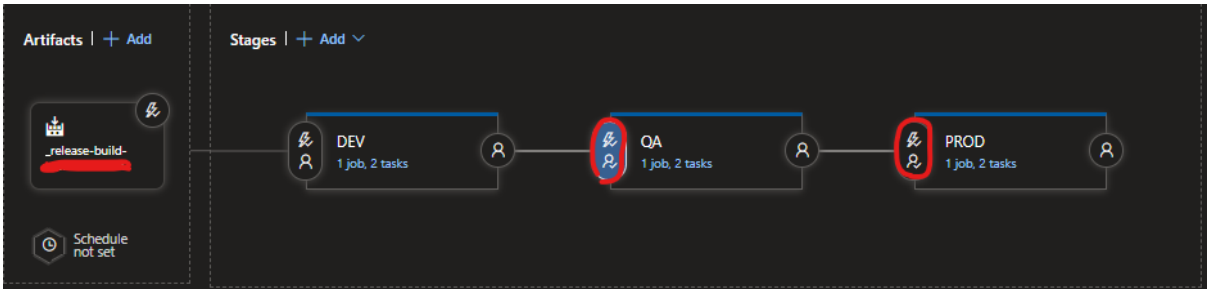
Here, first step is to **link a release pipeline with the artifact** generated by a build pipeline.



Next, enable **Continuous deployment trigger** to **create a release after a successful build**. Also, I've added in a **branch filter** to create automated releases for builds done from main/master/primary branch.

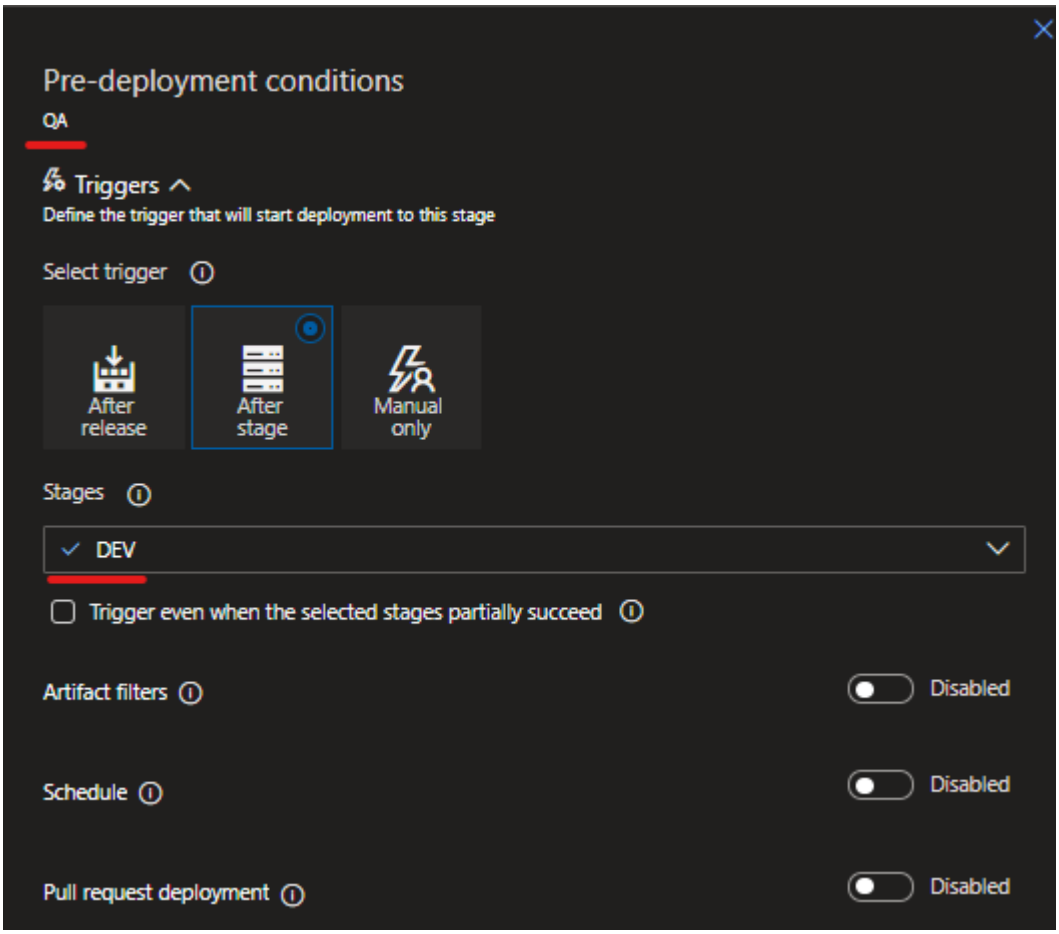


4) And 5) **Release pipeline** will look like this. The two tasks under each stage would be to deploy Web App and Web API to Azure (There's a marketplace task to perform that)

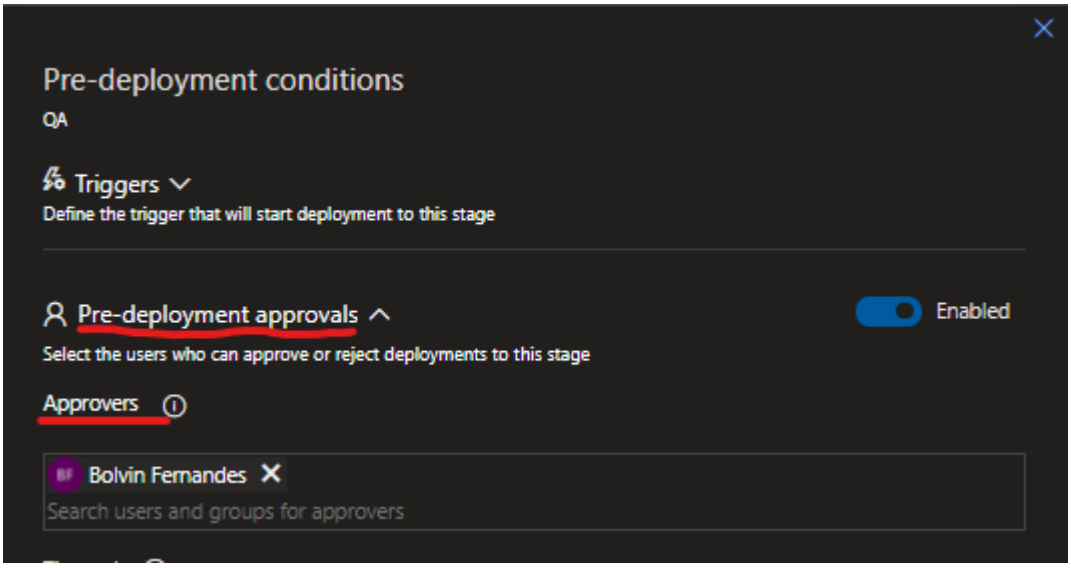


For automatic promotion to QA, following configurations need to be added.

'After stage' trigger needs to be selected with **DEV tickbox checked**

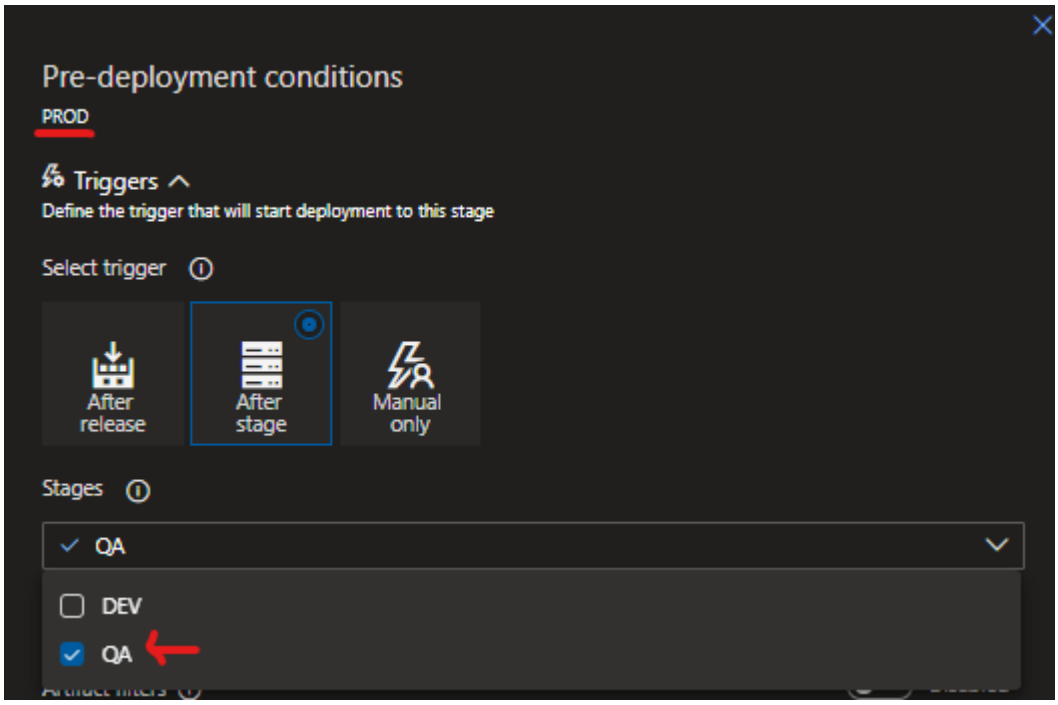


Pre-deployment approvals can be set in this fashion

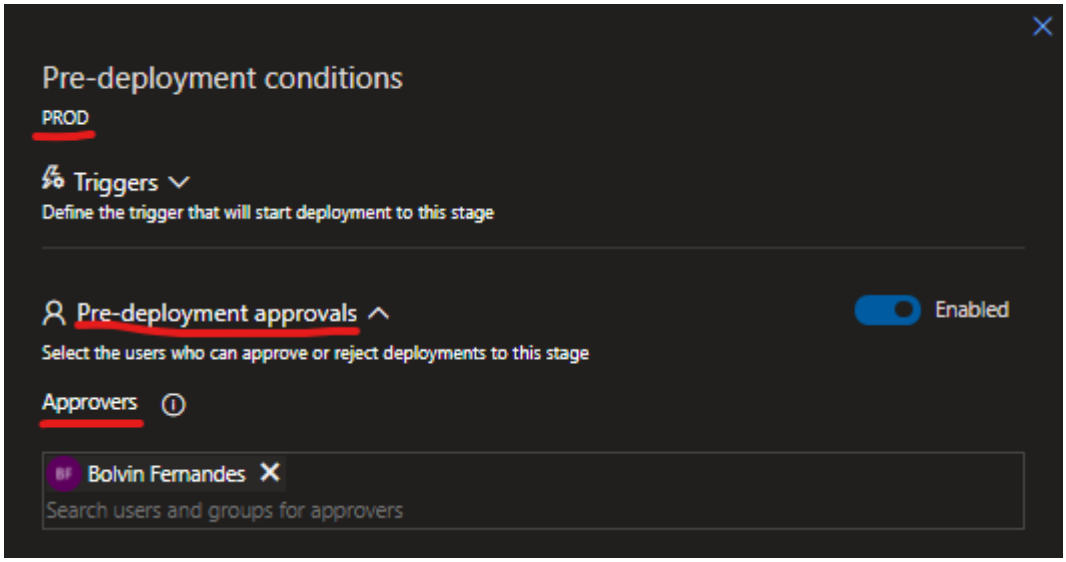


For automatic promotion to PROD, following configurations need to be added.

'After stage' trigger needs to be selected with **QA tickbox checked**



Pre-deployment approvals can be set in this fashion



My proposal

1. Developers should not be allowed to commit to master directly. They should use feature/dev branches and then submit a pull request (PR) to merge their changes to master/main/primary branch. Hence, it's necessary to enforce branch policies on

master/main/primary branch that include a **minimum number of reviewer**s and a **PR build** that is triggered with every pull request to master/main/primary branch.

https://github.com/bolvinfernandes/MaerskDevopsTest/blob/main/FastCarz/pipelines/pr-build.yml

```
6   trigger: none # will disable CI builds entirely
7   pr:
8   - master
```

The difference between a PR and Release build is that **PR build doesn't produce any artifacts** but is executed in order to make sure that changes coming in from a **developer is not breaking the existing code base.** As a result, PR build doesn't necessitate versioning. Also, the type of merge into master/main/primary branch should be a **squash** merge – as part of a squash merge the feature branch should be deleted as it could create conflicts if reused.

Build and Release pipelines can be configured using **either Classic UI editor or yaml**. I used both to show a mix. The advantage of using **yaml** is that it lives with application source code. So, when you branch your code you branch your yaml pipeline as well. This allows a developer to modify the build in their branch without affecting other branches.

I work with Azure Devops on a daily basis hence, I was able to get screenshots as I created my own pipelines. Also, naming convention i.e. **pr-build.yml** and **release-build.yml** is something I've been using over the years with TFS and now, Azure Devops. I've tried to hide any company related information however, if you see it please ignore it 😊