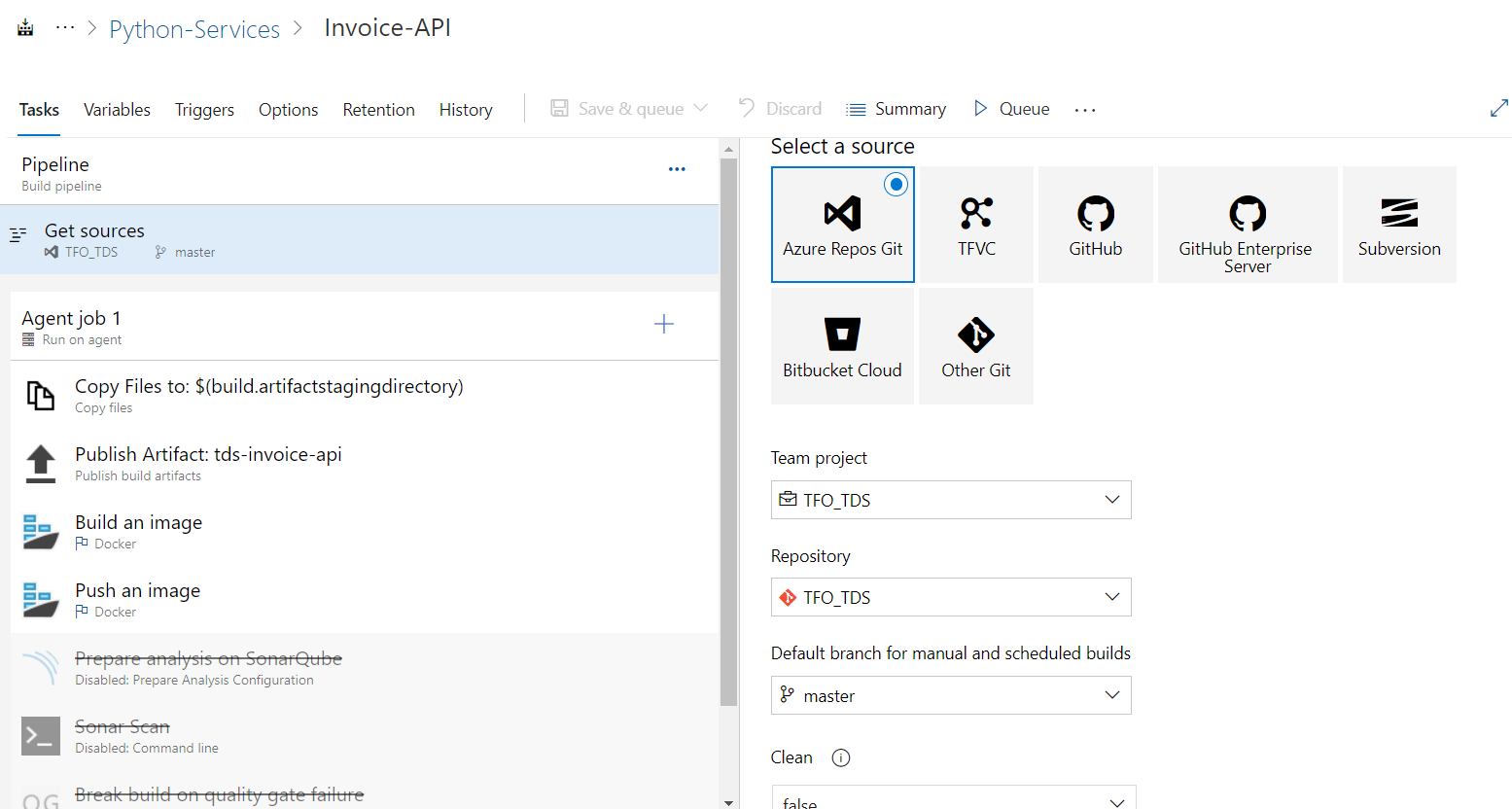
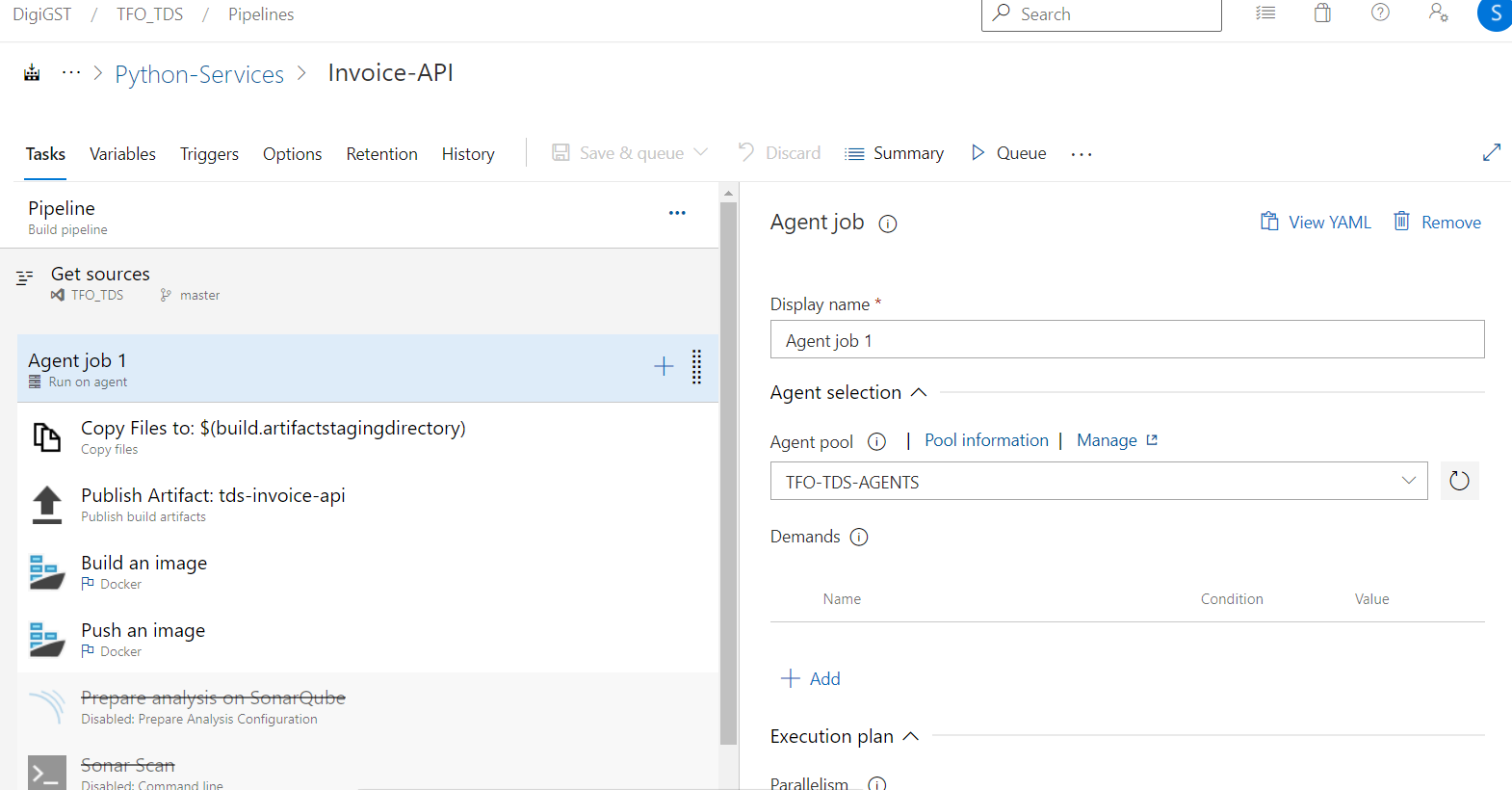
**INVOICE API Build Pipeline:**

**Code Pull:** Here we are pulling the code from azure git repositories from master branch.

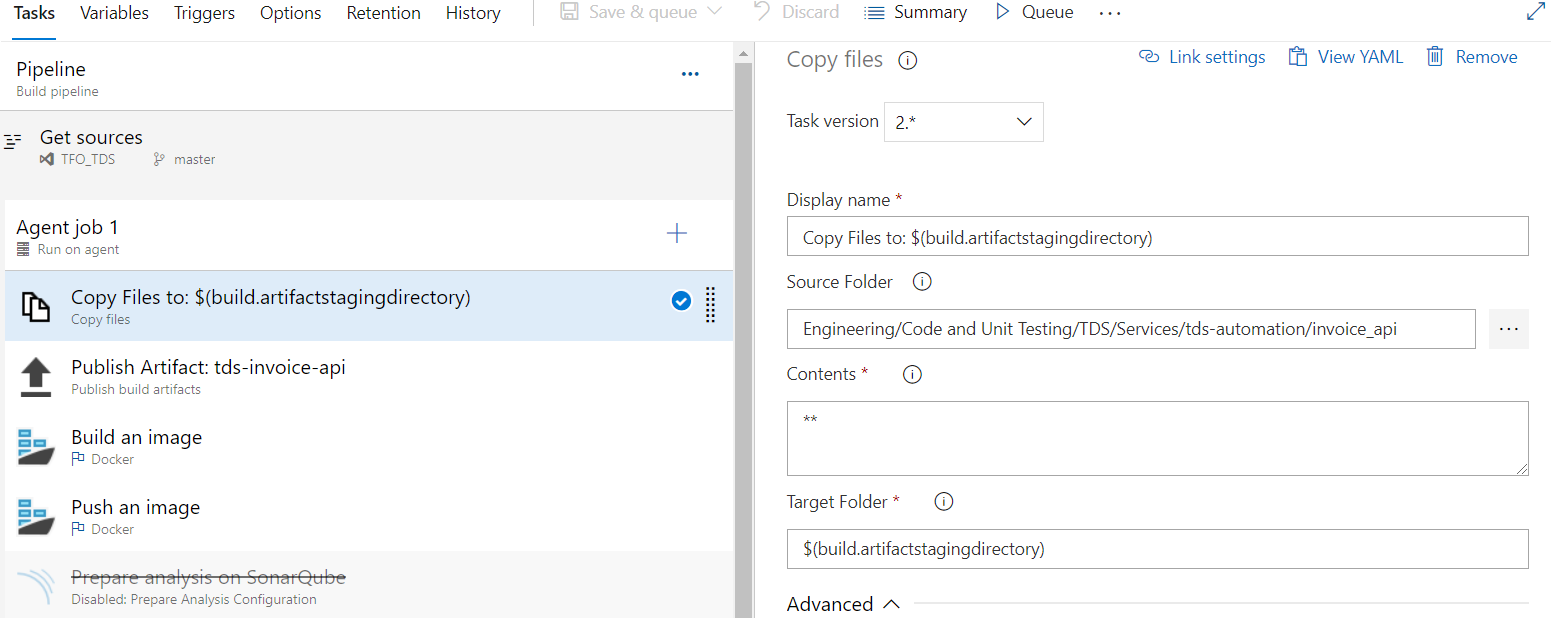


**Agent Job:**

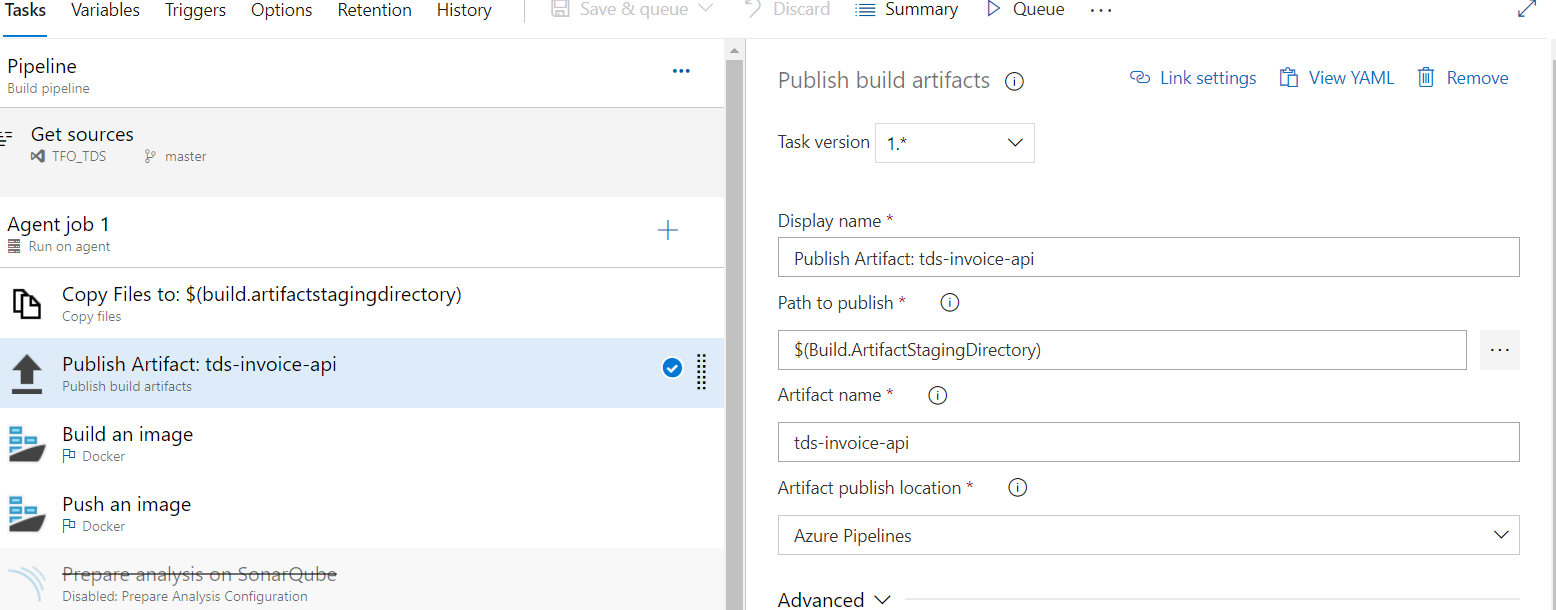
We have agents to pick the task for execute. For each project we can able to create separate agents pools.



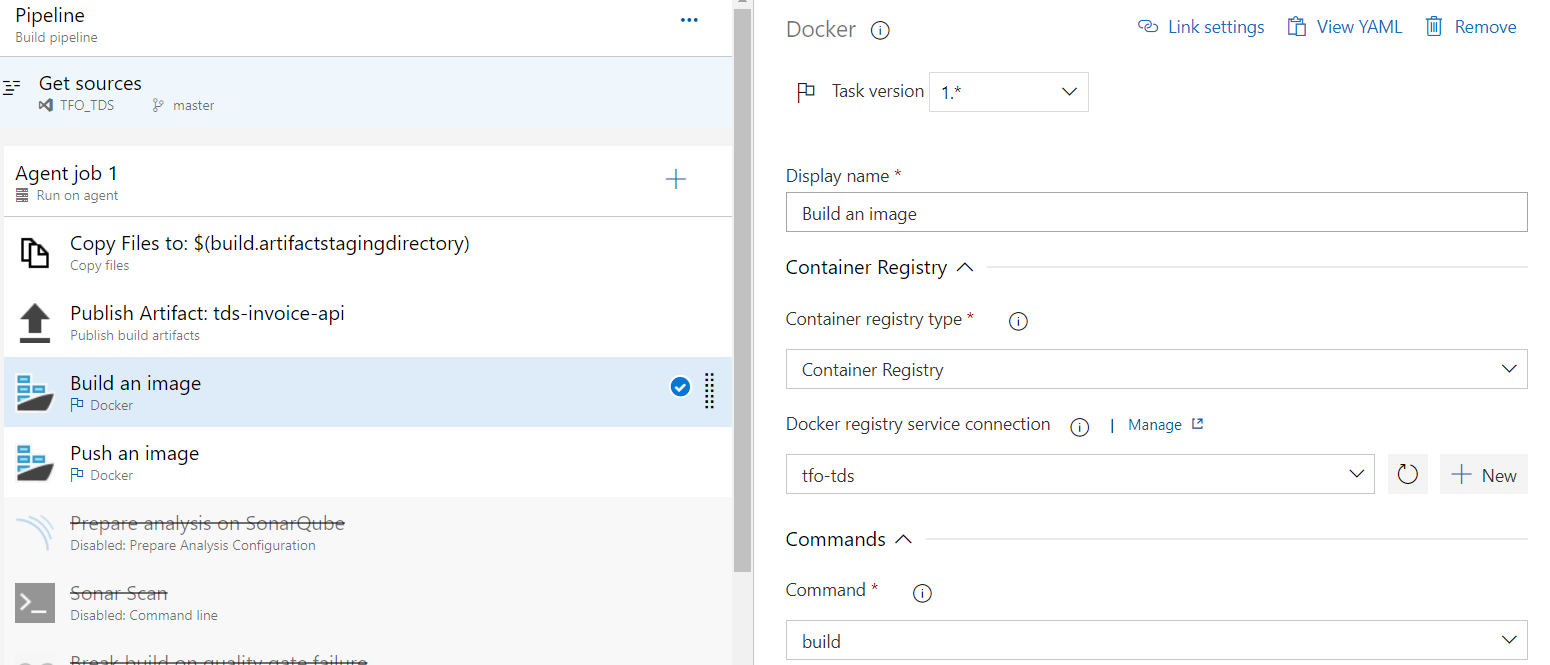
**Copy Files:** We are taking our source files from the source folder and copying into the artifactory directory. Here we are passing the target directory as parameter.



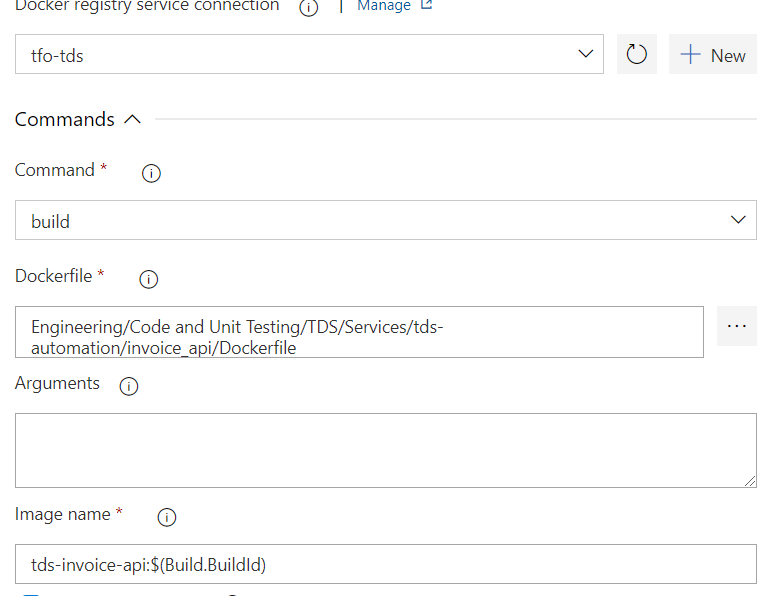
**Publish Artifacts:** Here we are publishing the artifacts of API service into the artifactory directory. By passing the artifcatory name.



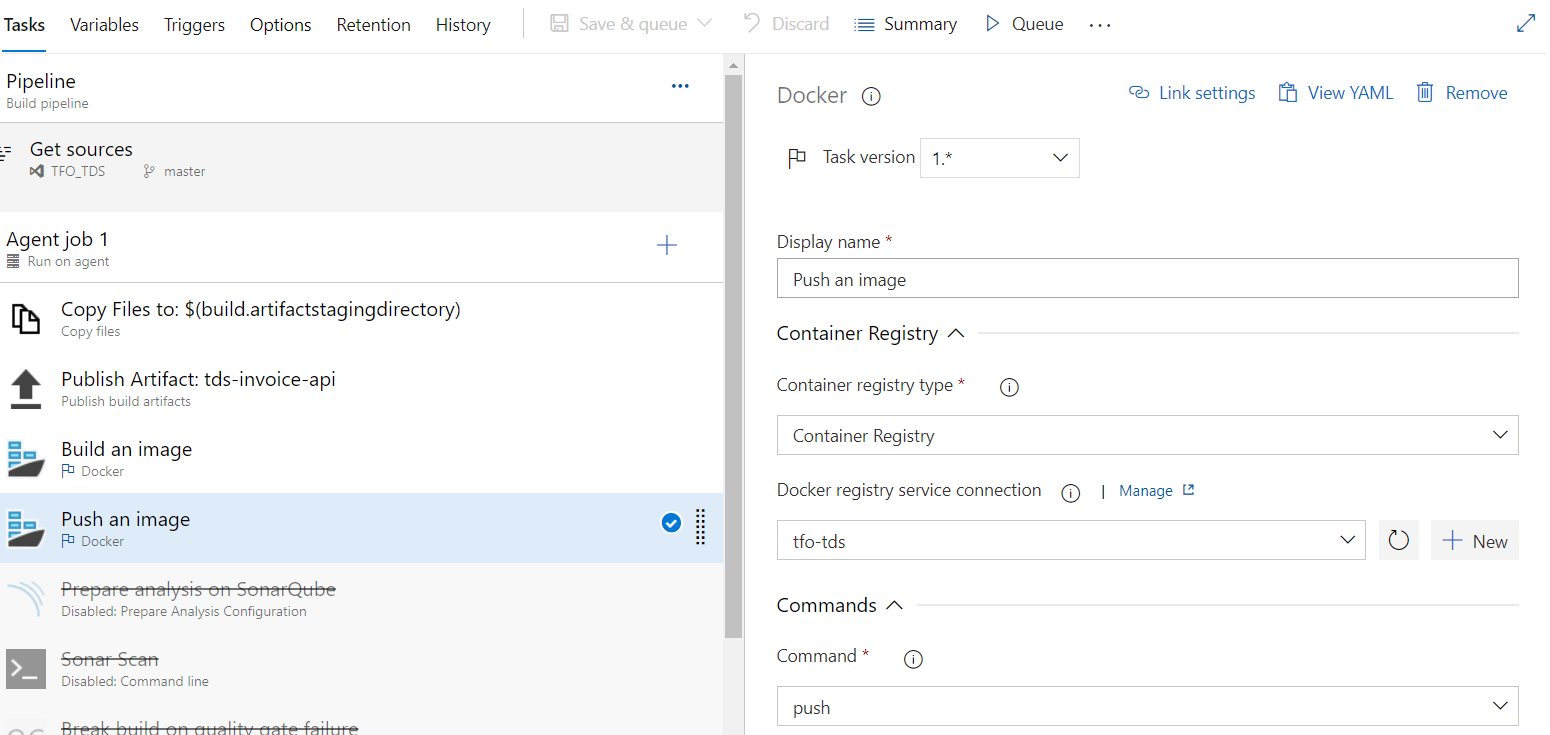
**Build an image:** In this stage we are building an image by using docker build command. The docker build command builds Docker images from a Dockerfile and a “context”. A build’s context is the set of files located in the specified PATH or URL. The build process can refer to any of the files in the context. For example, your build can use a COPY instruction to reference a file in the context.



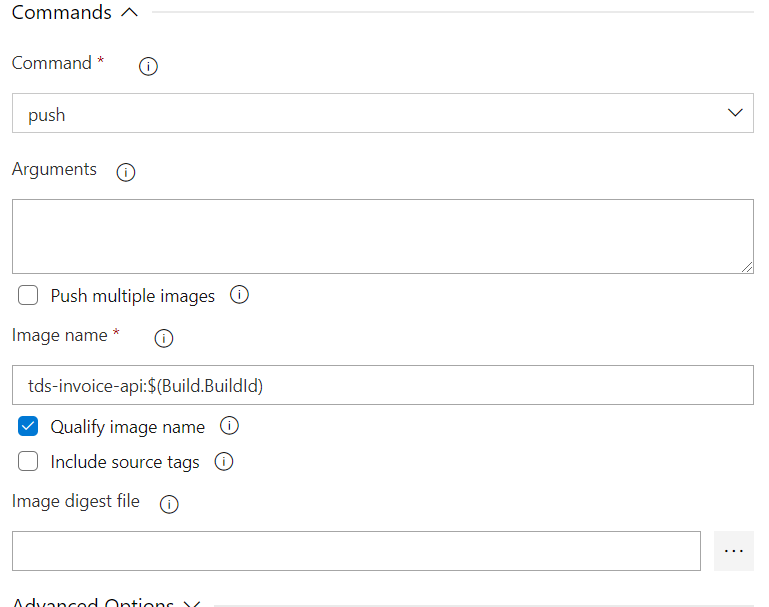
Here we specified the dockerfile path to build the image. Building image name as buildid.



**Push an Image:** Once build got success we are pushing that image by using docker push command and push that image into ACR Azure container registry.

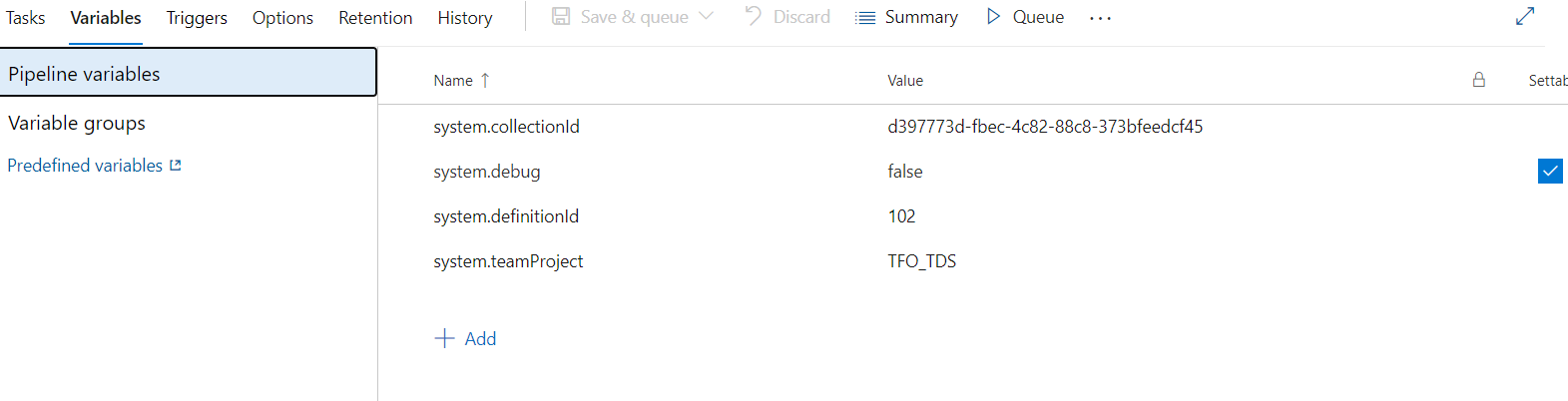


Generated image name we are pushing the same image into ACR.

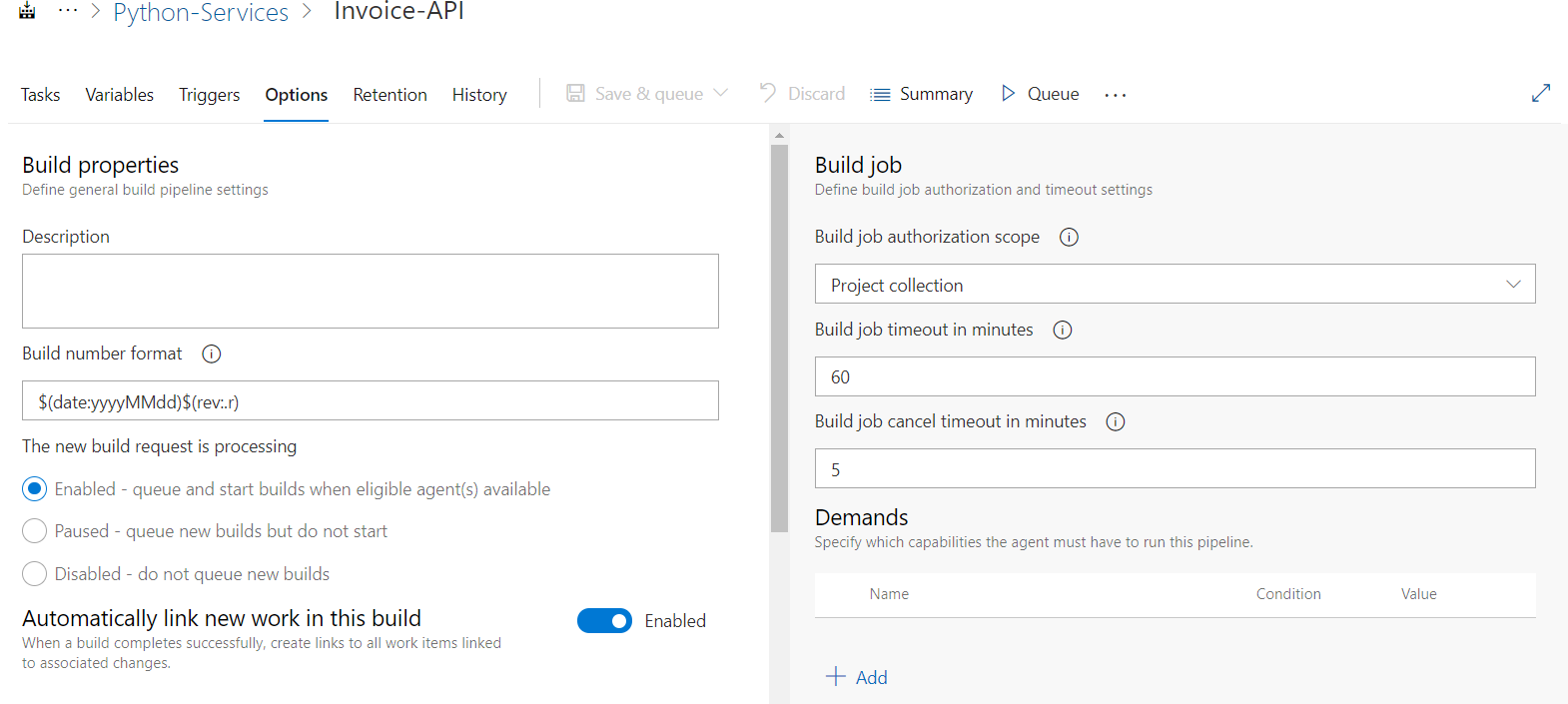


We have other stages as well sonarqube for code quality check and report generation.

**Variables:** You see here the pipeline variables with values as below.

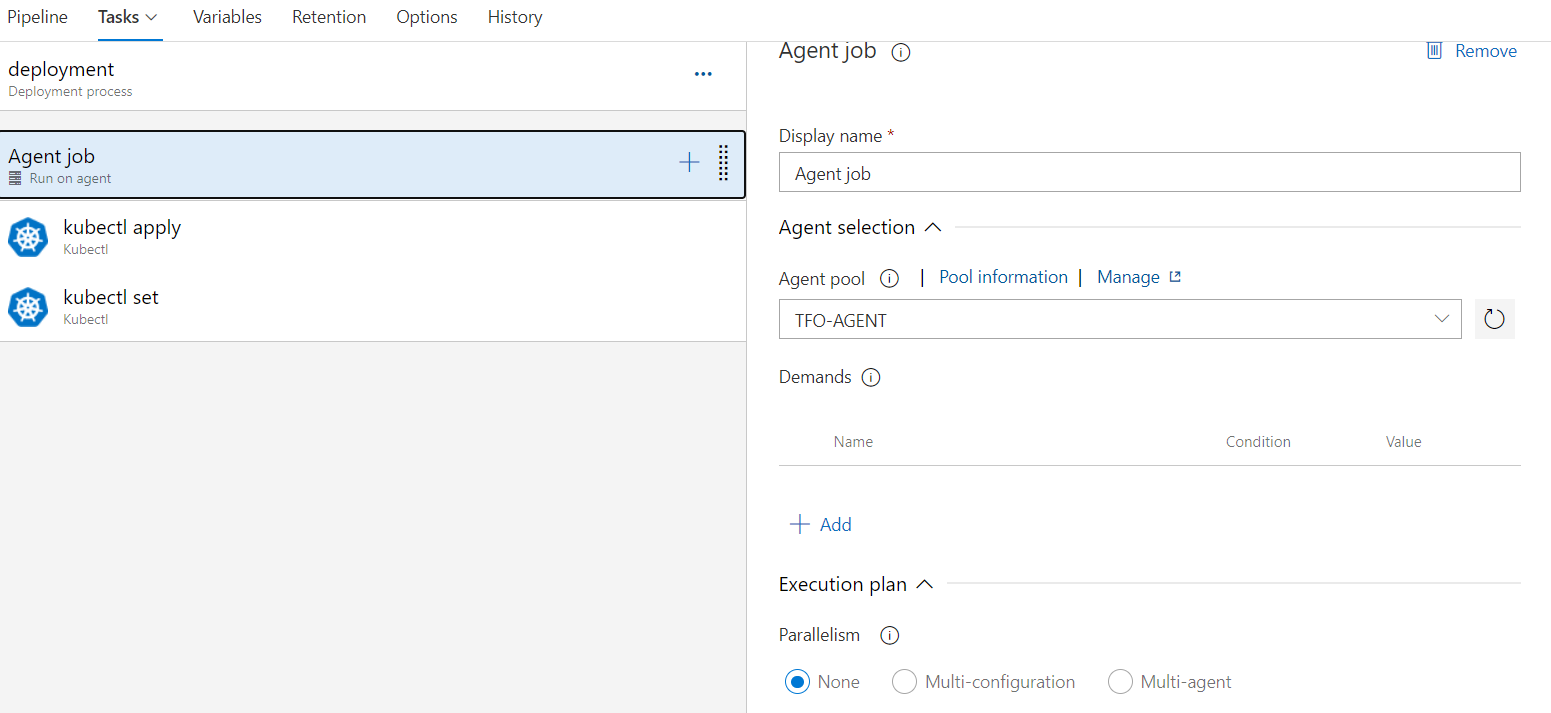


**Options:** Here we are making the build number in a specific format to make the build number in that way and easy to identify for well-organized manner. We kept the build timeout minutes as 60 and number of minutes to cancel timeout of job.

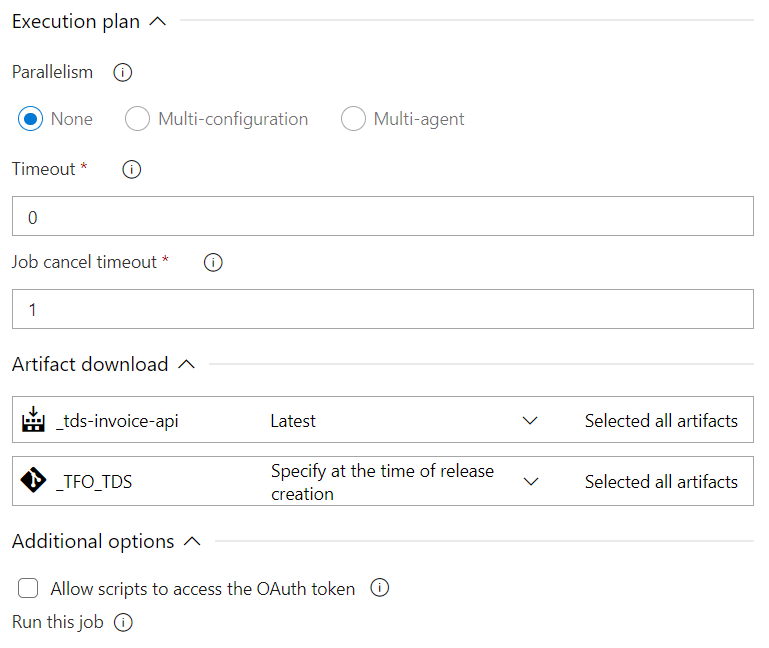


**INVOICE API Release Pipeline**

**Agent Job: In** this stage we are selecting the agent pool for to execute the job. Each agent can execute the one job at same time. Here deployment pipeline has two tasks 1. Kubectl apply , . kubectl set.

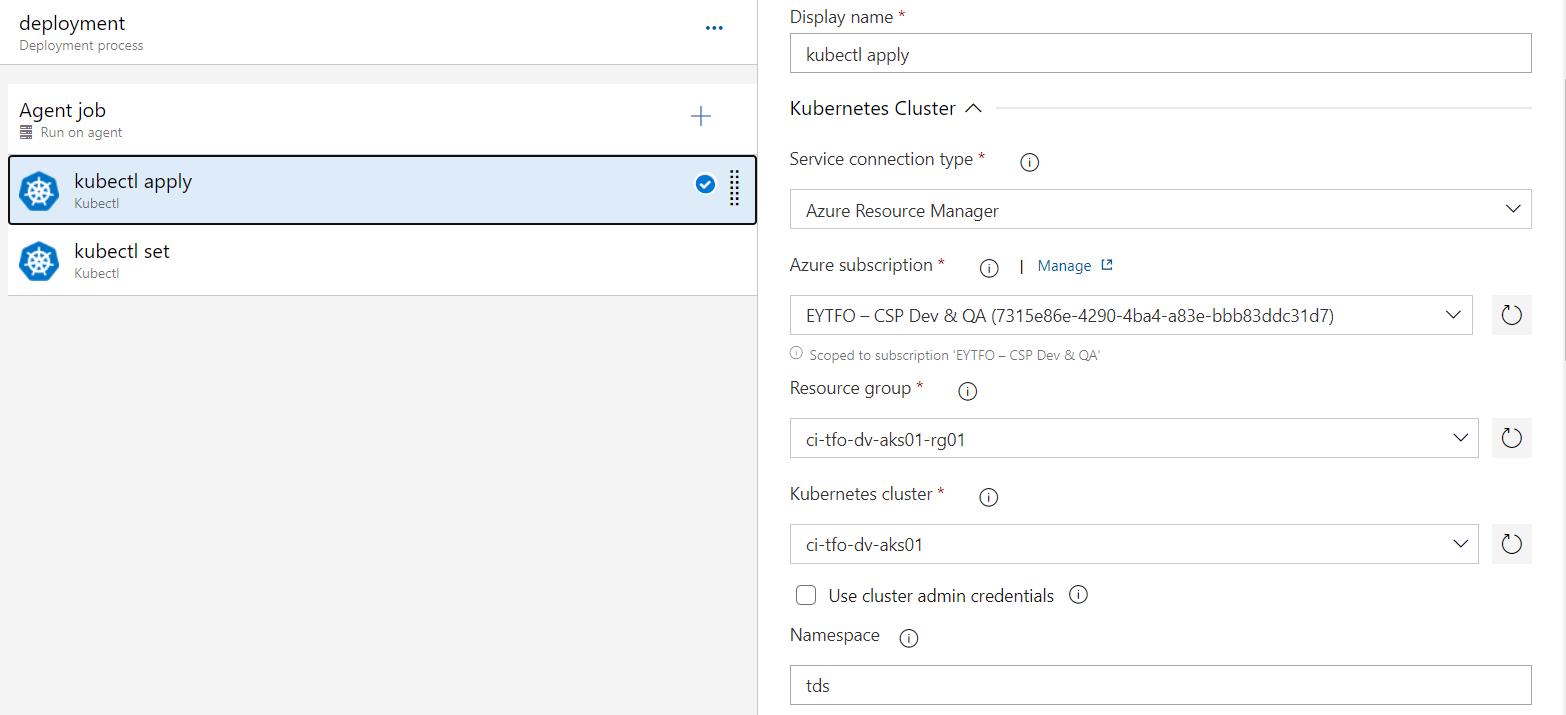


We are mentioning the execution details over here like timeout timing and job cancel timeout in minutes. We are mentioning the atifactory for to pick the latest image for this particular service to display the latest image id and pervious id’s.

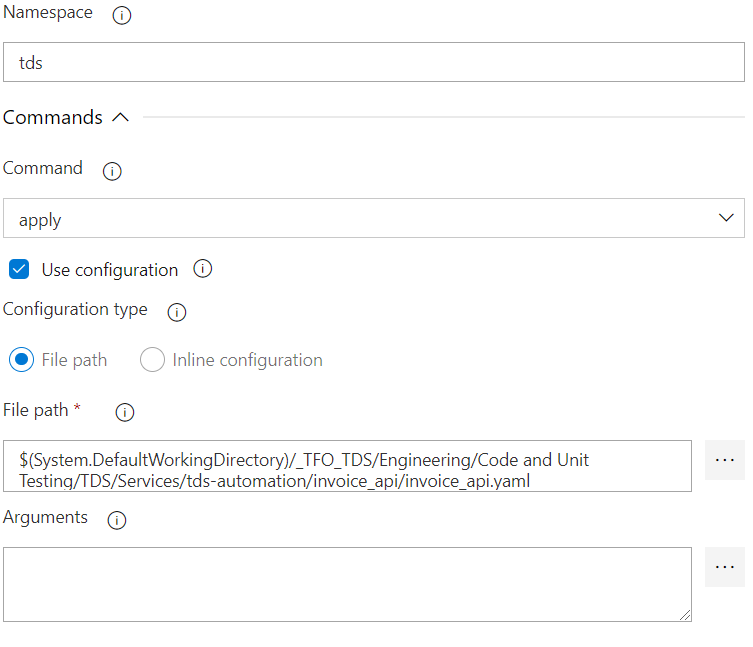


**Kubectl apply:**  command to create resources- apply manages applications through files defining Kubernetes resources. It creates and updates resources in a cluster through running kubectl apply.

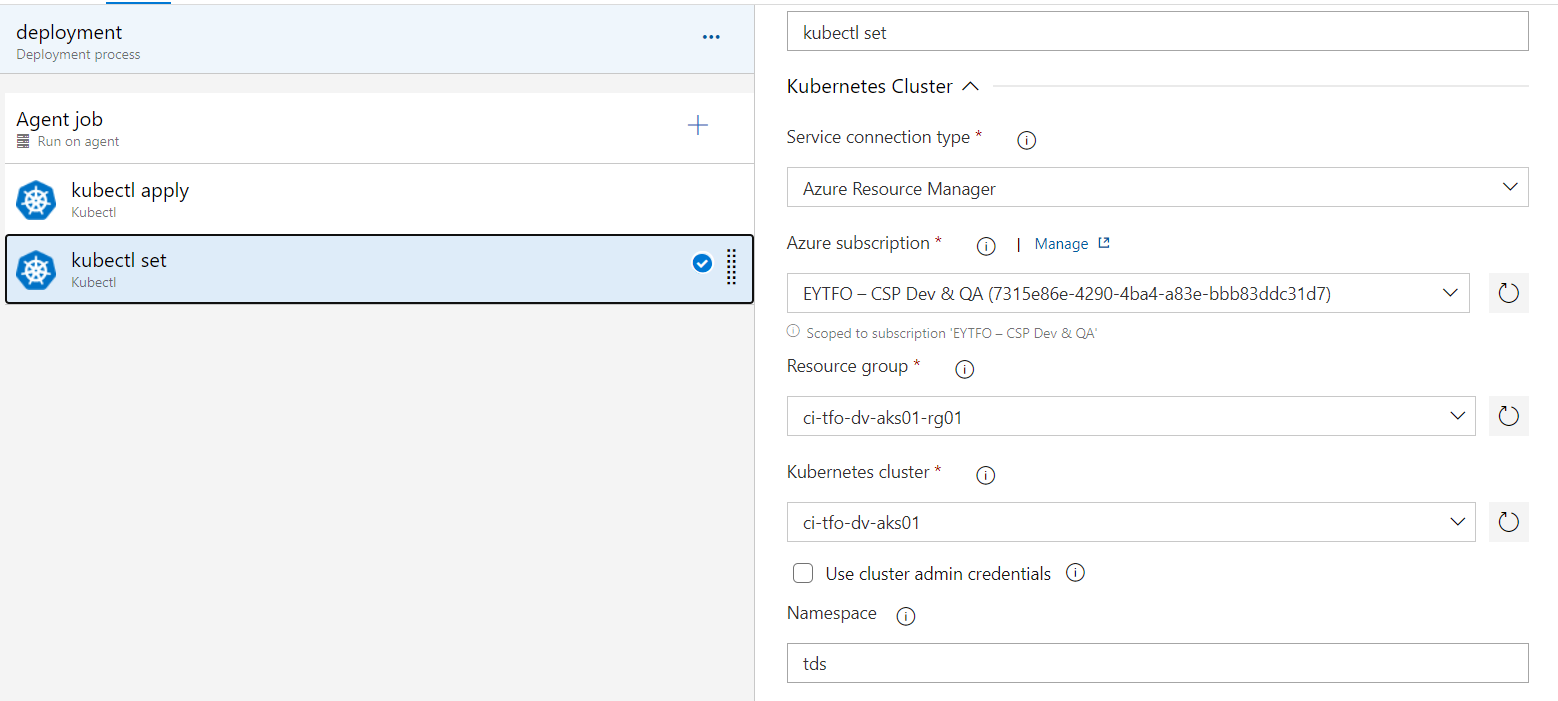
Here we are passing the namespace value where the pod has to run and service has to deploy. Cluster name also we are passing under which cluster we are going to deploy this service.



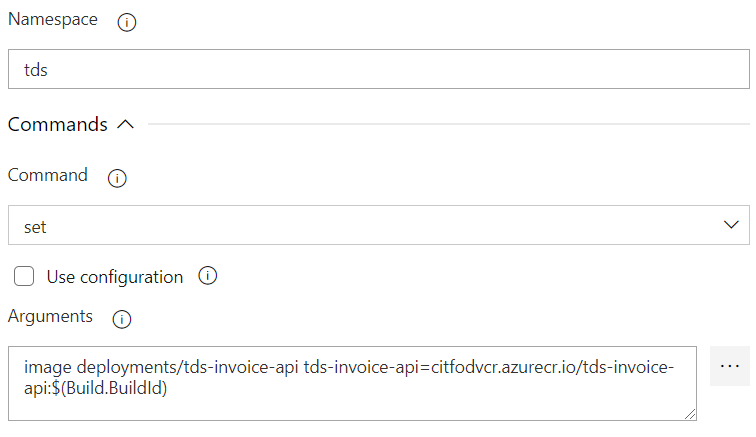
We are specifically mentioning the yaml file path to deploy accordingly.



**Kubectl set:** Display one or many resources- Rolling update "www" containers of "frontend" deployment, updating the image. Prints a table of the most important information about the specified resources. You can filter the list using a label selector and the --selector flag. If the desired resource type is namespace you will only see results in your current namespace unless you pass --all-namespaces.

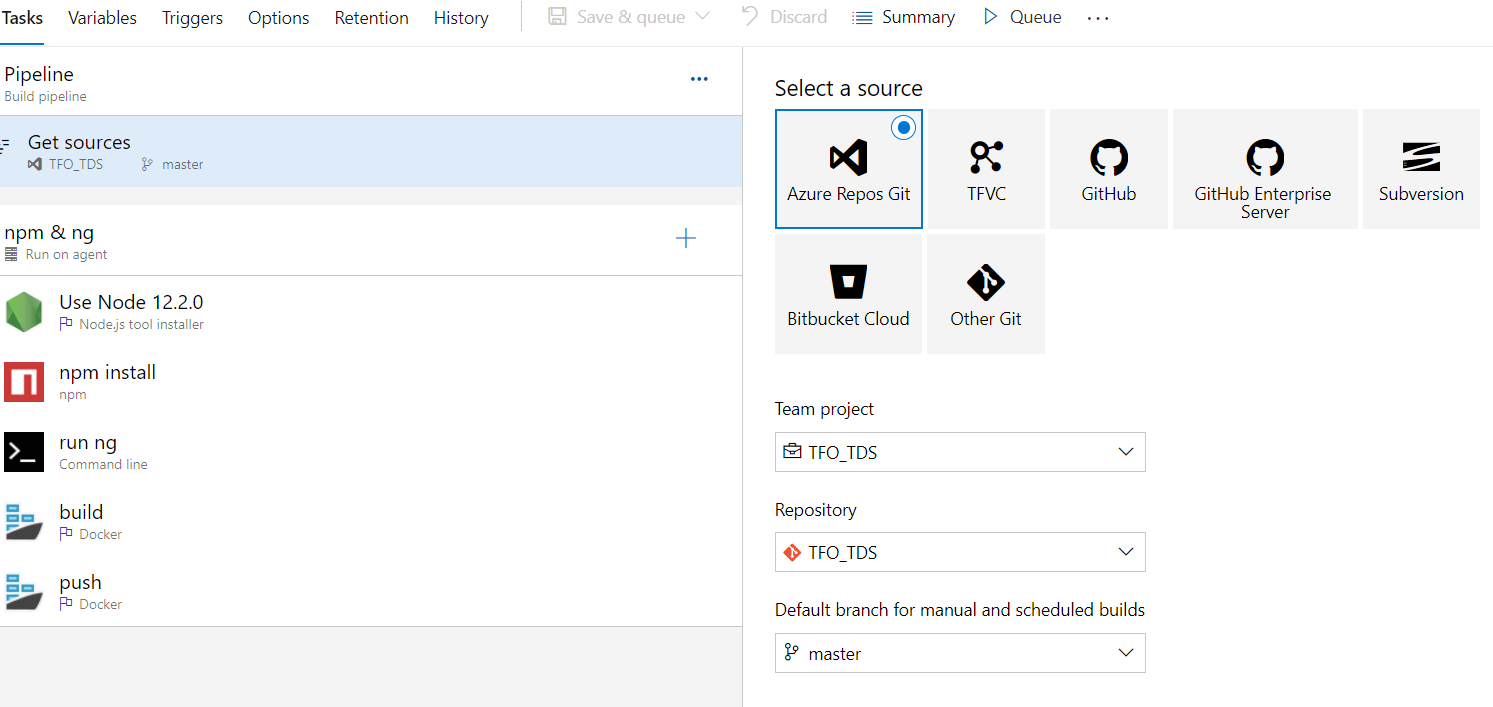


We are passing the image build ids to select the resources of specific service.



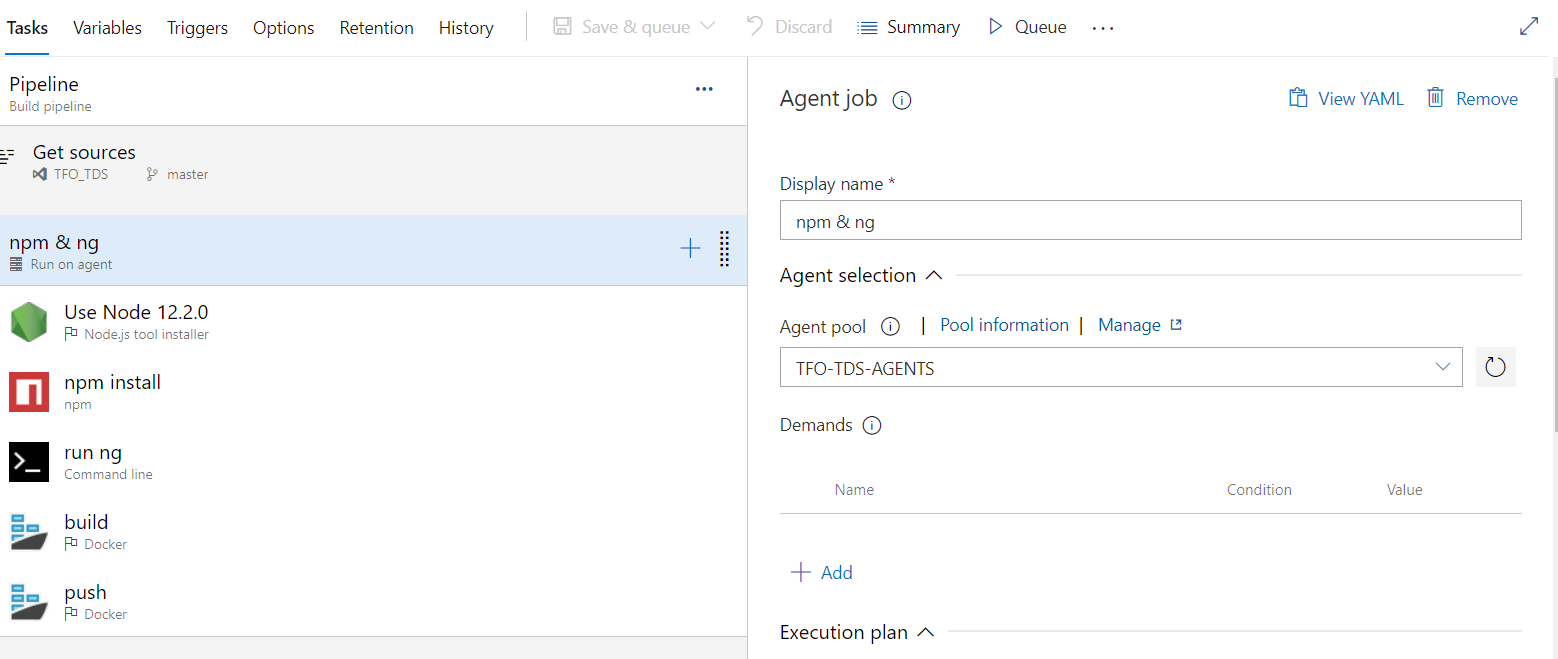
**UI BUILD Pipeline:**

**Code Pull:** Here we are pulling the code from azure git repositories from master branch.

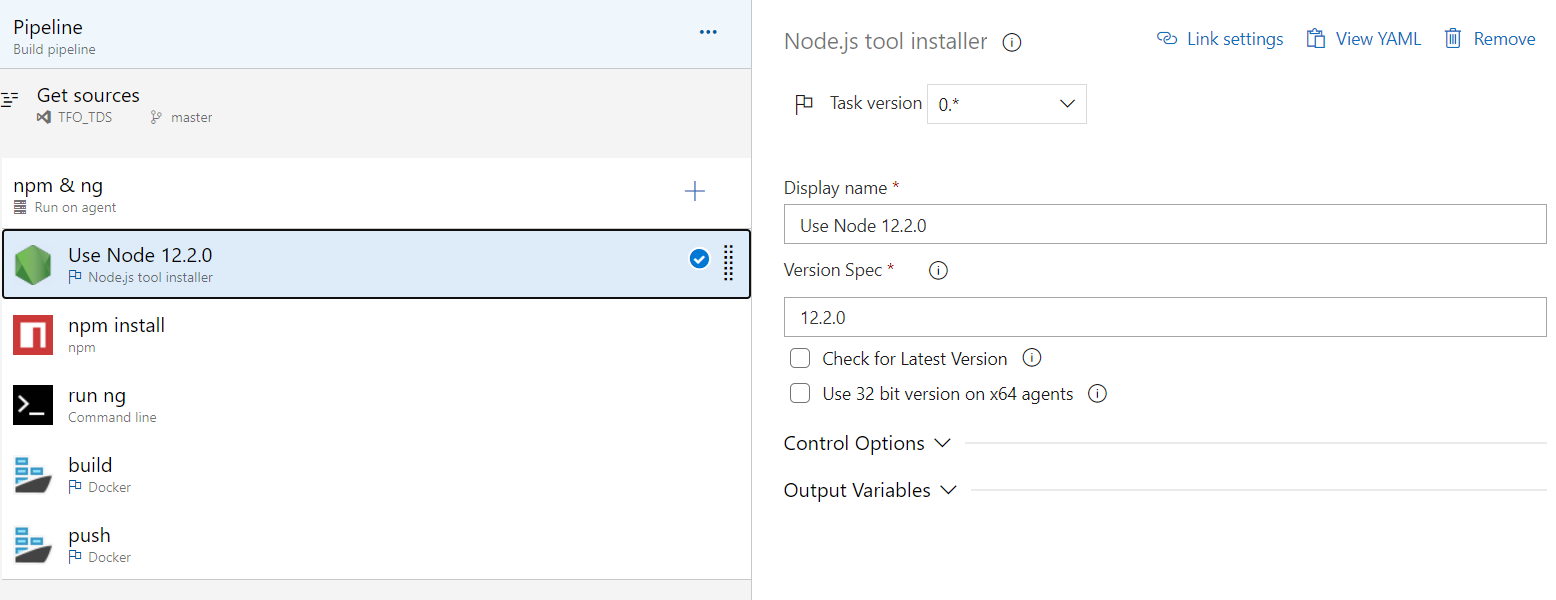


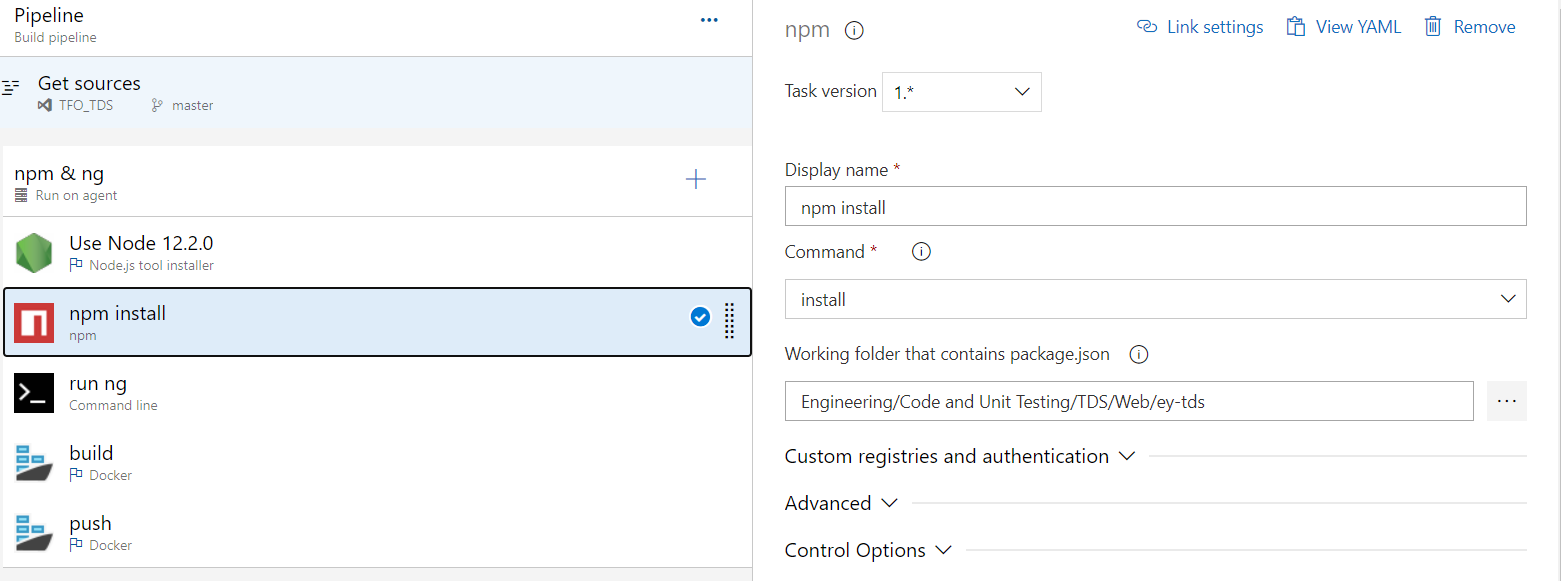
**Agent Job:**

We have agents to pick the task for execute. For each project we can able to create separate agents pools.

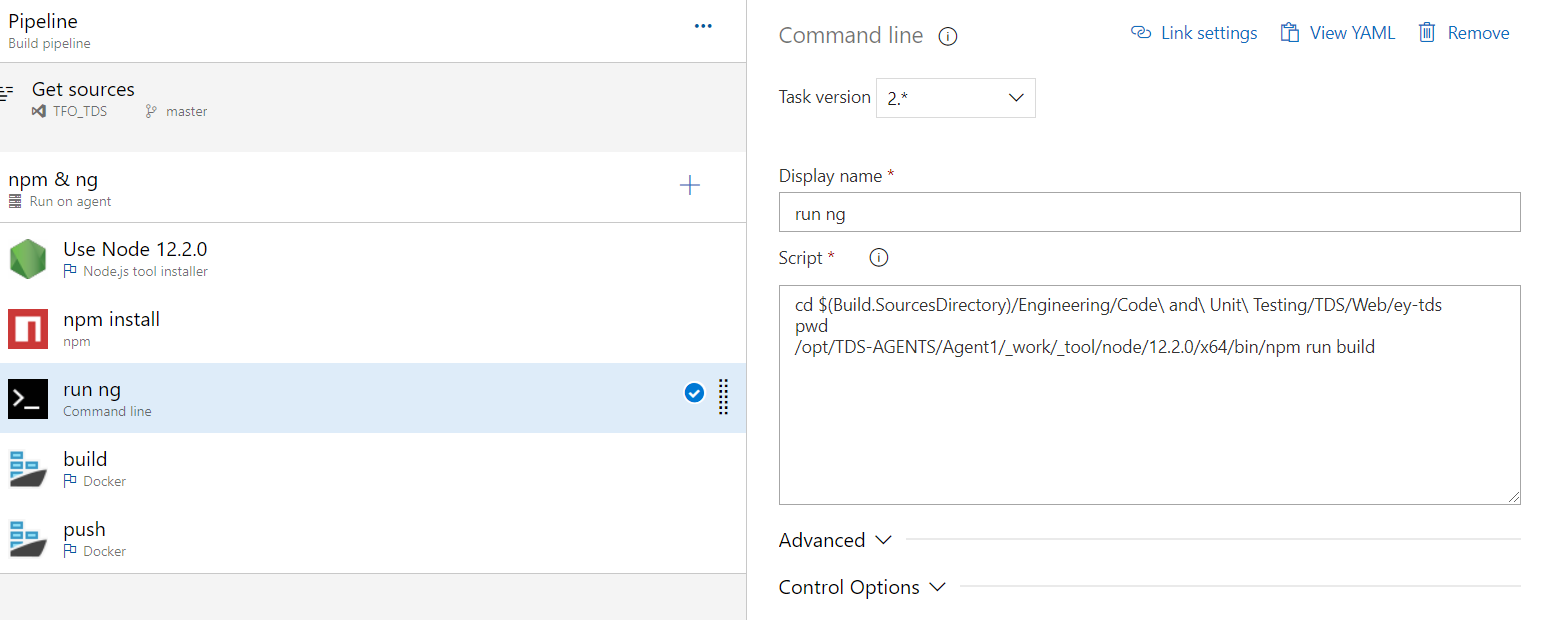


**Use Node:** We are specifically choosing the node version to run the node js code.

**NPM Install:**  installed the required Node version (a 64-bit version by default since the current system’s architecture is 64-bit)

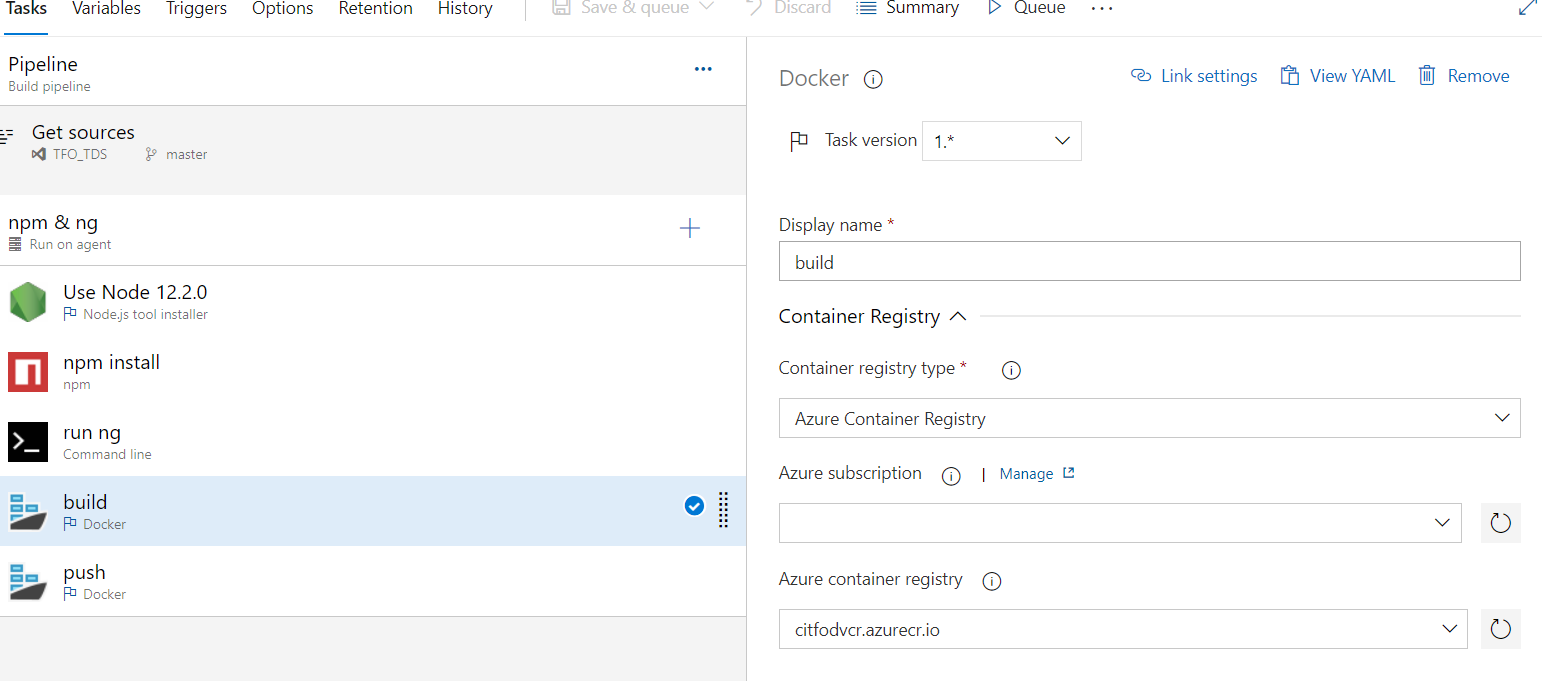


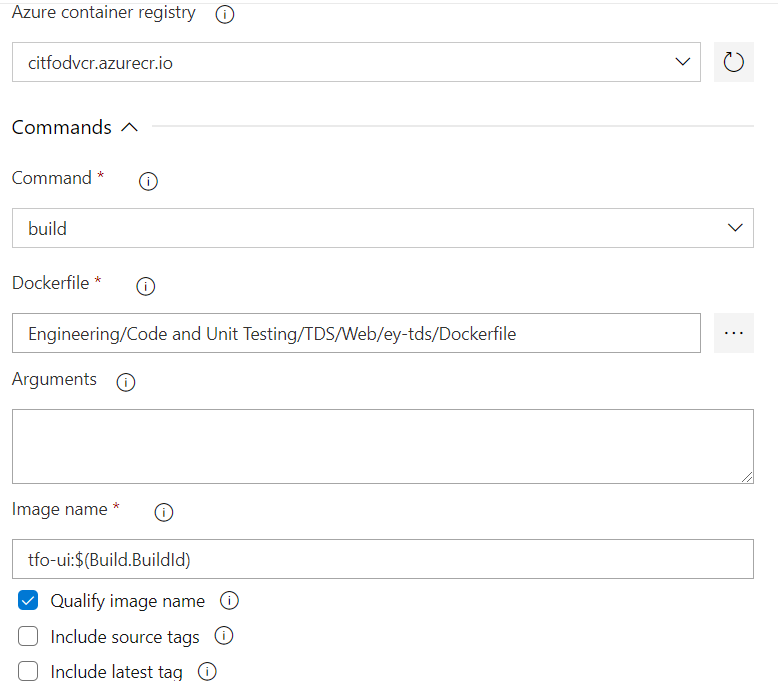
**Run ng:** Builds and serves your app, rebuilding on file changes.



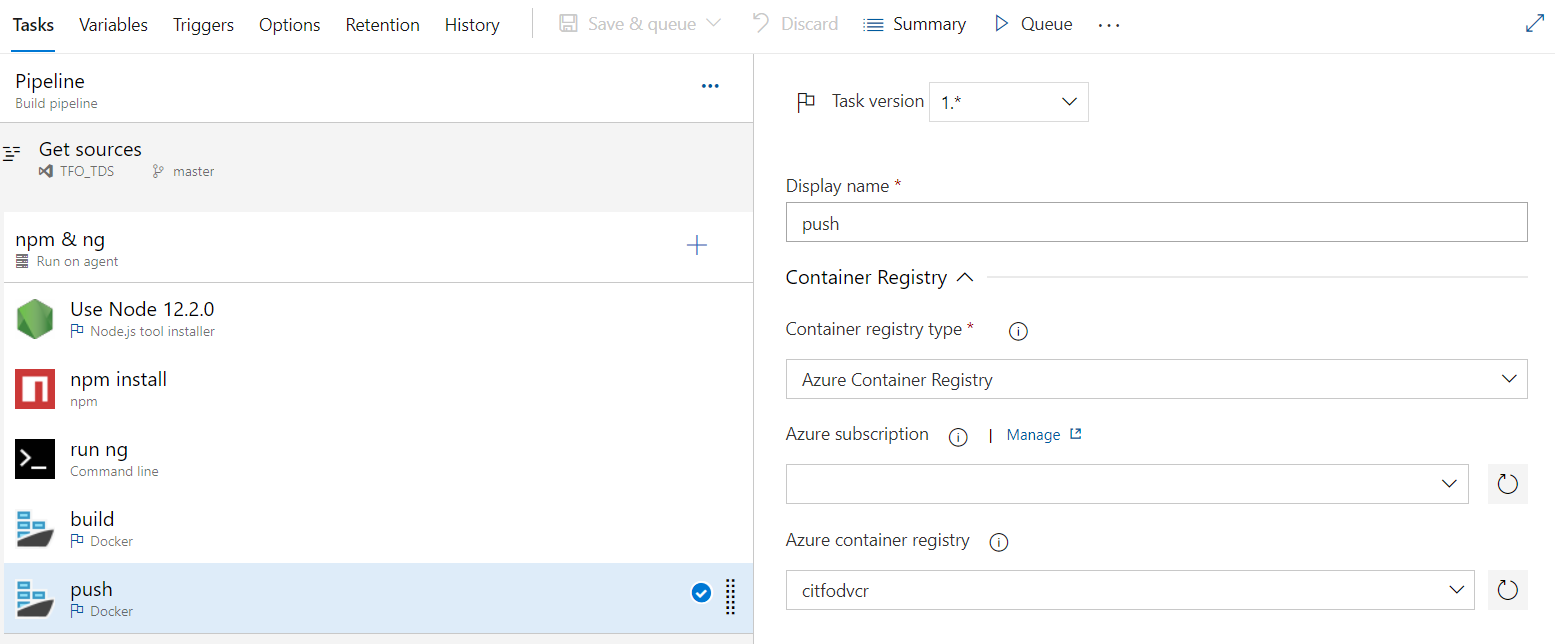
**Build an image:** In this stage we are building an image by using docker build command.

The docker build command builds Docker images from a Dockerfile and a “context”. A build’s context is the set of files located in the specified PATH or URL. The build process can refer to any of the files in the context. For example, your build can use a COPY instruction to reference a file in the context.

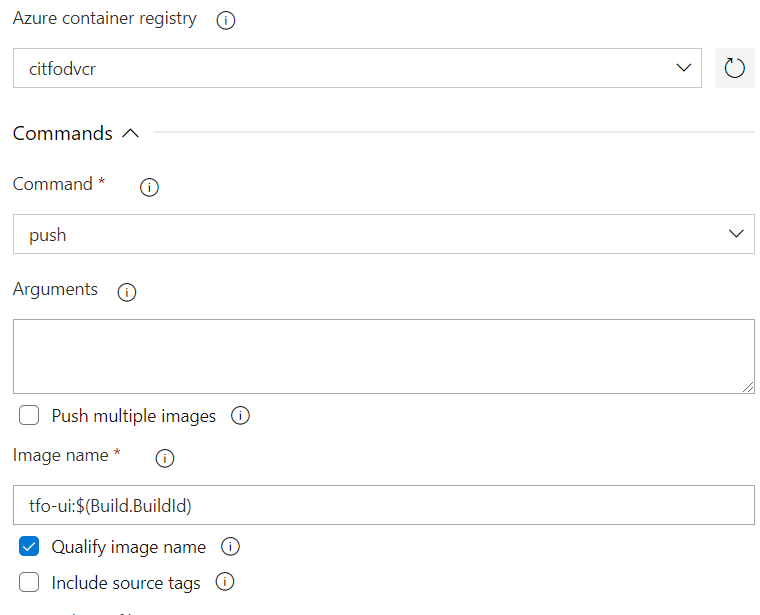




**Push an Image:** Once build got success we are pushing that image by using docker push command and push that image into ACR Azure container registry. Push an image or a repository to a registry Use docker image push to share your images to the Docker Hub registry or to a self-hosted one. Refer to the docker image tag reference for more information about valid image and tag names.



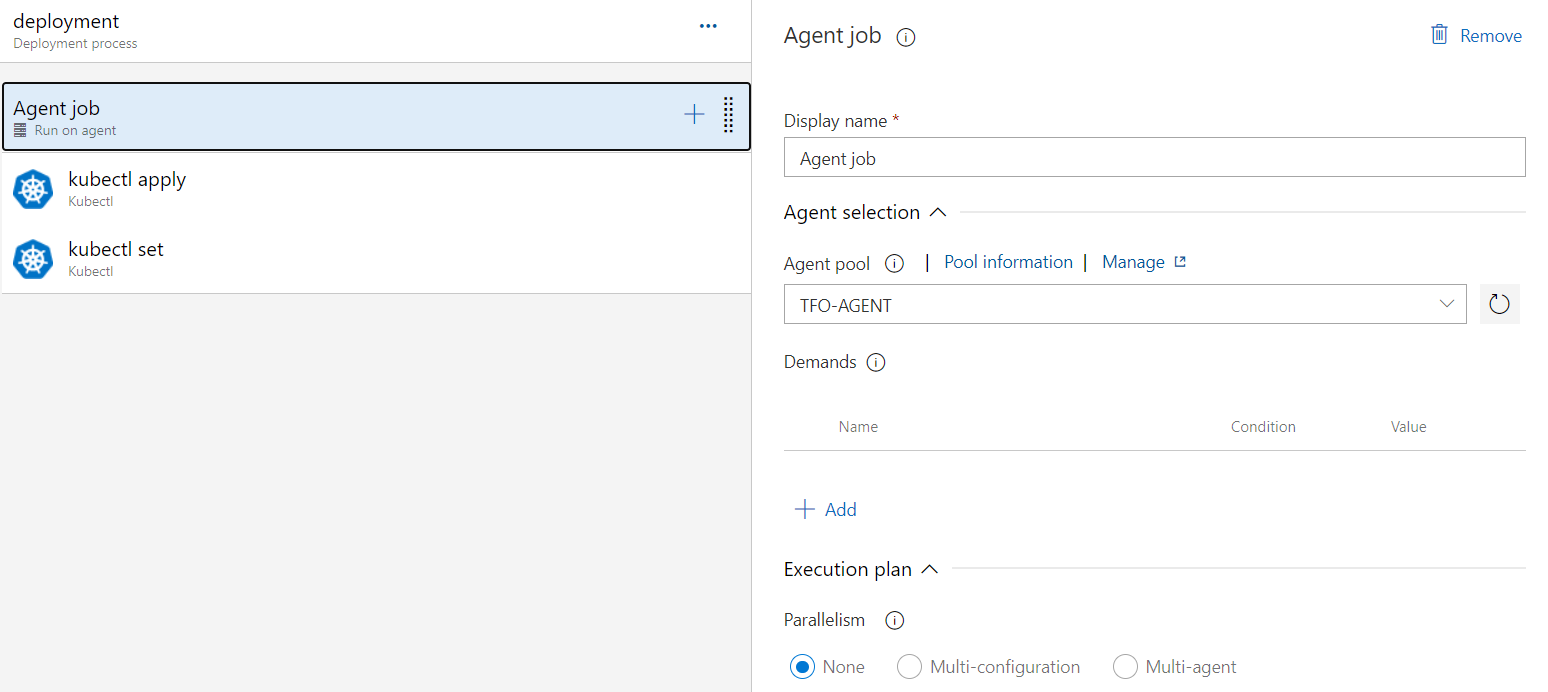
Generated image name we are pushing the same image into ACR.



**UI Release Pipeline:**

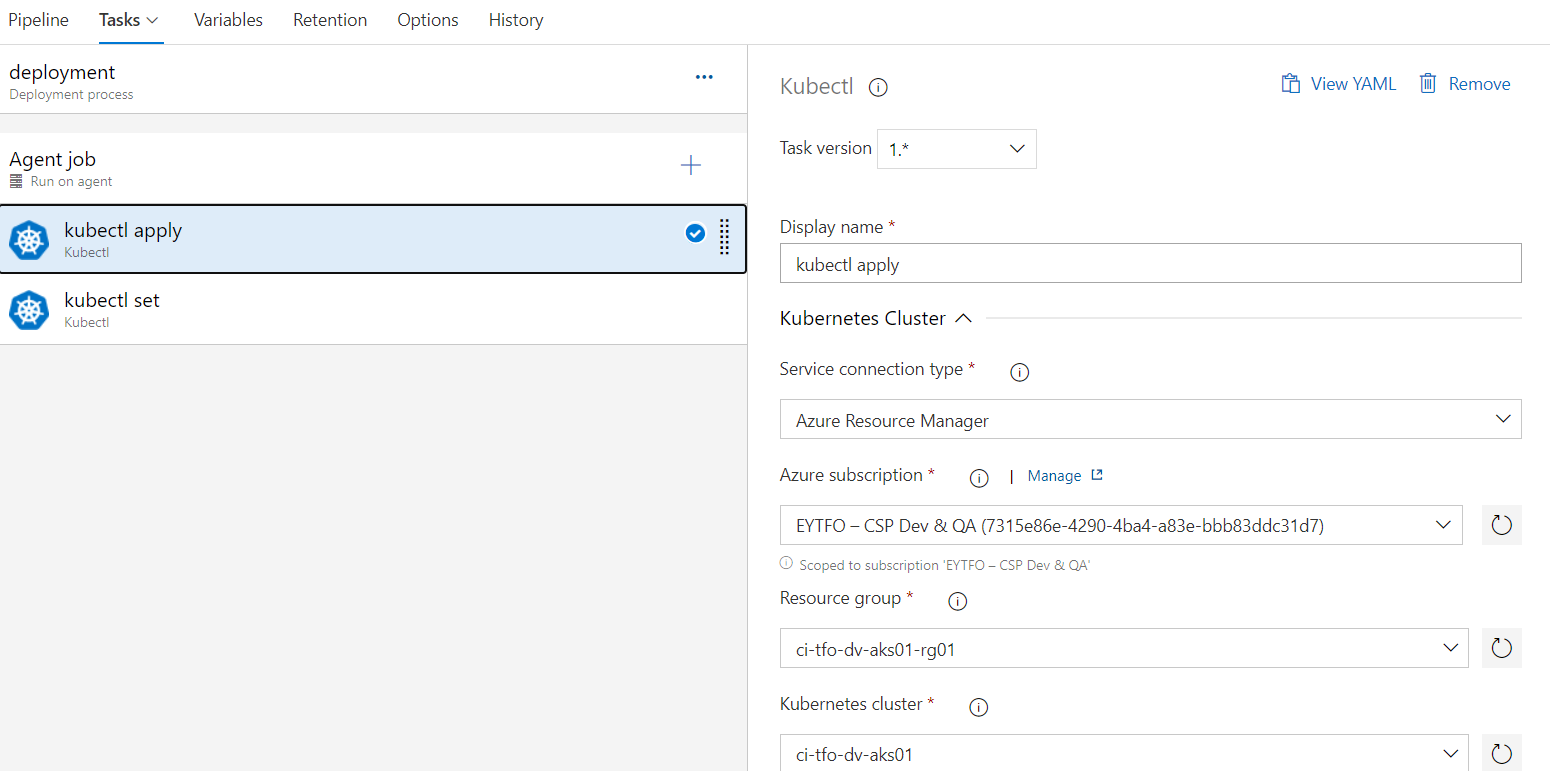
**Agent Job:** We have agents to pick the task for execute. For each project we can able to create separate agents pools.

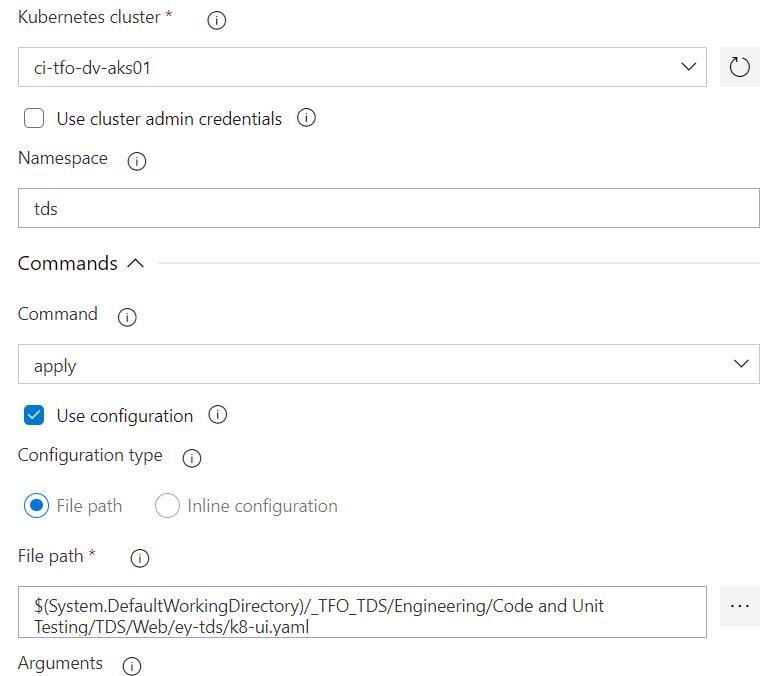
Here deployment pipeline has two tasks 1. Kubectl apply, kubectl set.



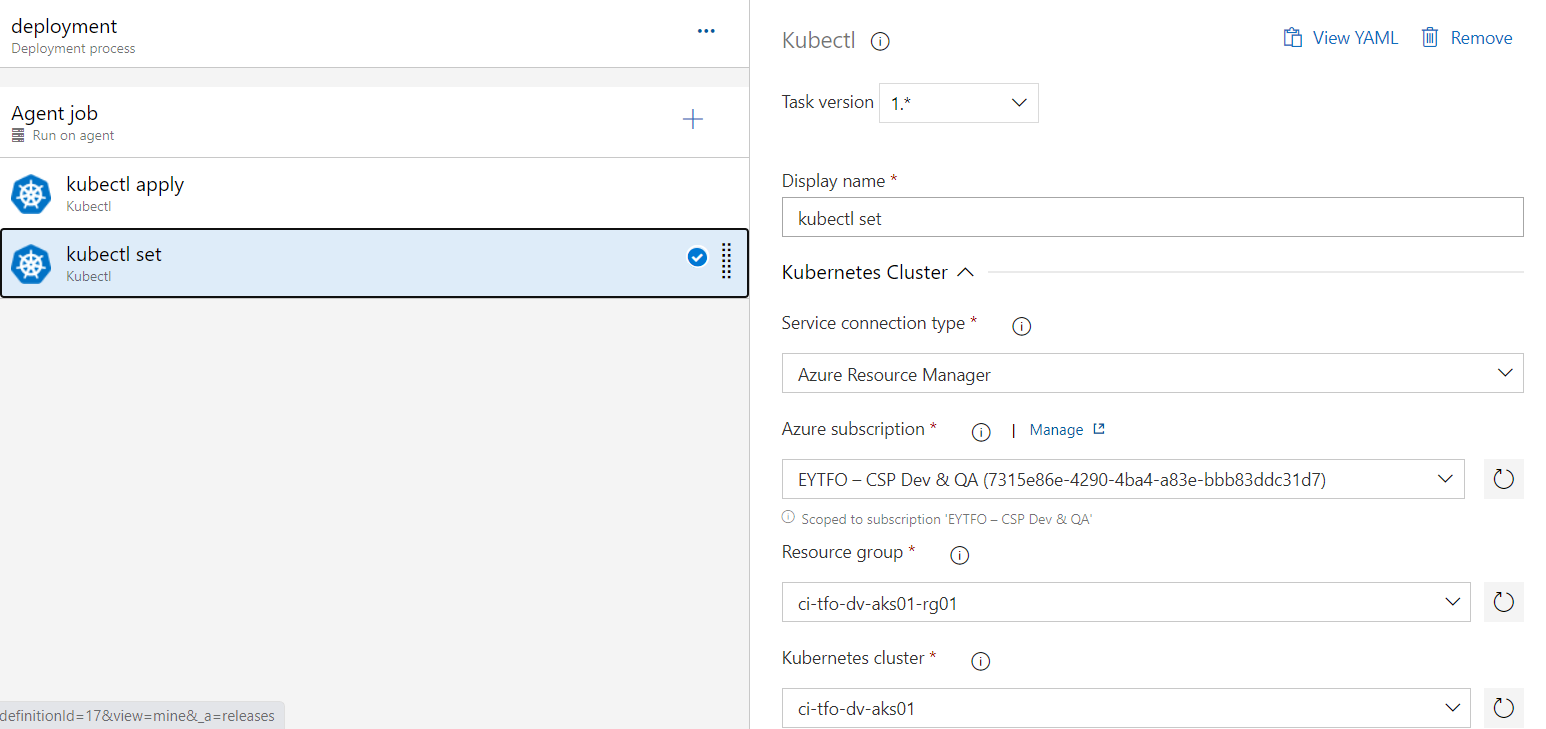
**Kubectl apply:** command to create resources- apply manages applications through files defining Kubernetes resources. It creates and updates resources in a cluster through running kubectl apply.

Here we are passing the namespace value where the pod has to run and service has to deploy. Cluster name also we are passing under which cluster we are going to deploy this service.





**Kubectl set:** Display one or many resources- Rolling update "www" containers of "frontend" deployment, updating the image. Prints a table of the most important information about the specified resources. You can filter the list using a label selector and the --selector flag. If the desired resource type is namespace you will only see results in your current namespace unless you pass --all-namespaces.



We are passing the image build ids to select the resources of specific service.

