Azure DevOps

DevOps Basics

- 3 keys to great software: Enhanced communication, automation and quick feedback.
- 3 cross functional team with wide range of skills: Business, Development and Operations
- Agile Process: It brought the business and the development team to one team. This team work to build great software in small iterations called sprint. So in each sprint this team does plan, code, build, test, release, deploy and review of small unit of software through user stories.
- Agile Automation: The possible area where agile team works on automation are Code commit, Unit Tests, Code Quality, Package

and Integration Tests. This is mainly to find functional or technical defects as early as possible.

- ≡ ○ Continuos Integration: The above mentioned automation should happen continuously. This can be achieved through using continuous integration tool like jenkins.
- ≡ ○ Agile Retrospective and review: This helps in quick feedback on the product which helps in producing enhanced product every sprint.
- ≡ ○ New challenges: By following Agile and microservice development practice new challenges arrived in operations.
- ≡ ○ DevOps: DevOps bring development and operation team together as one team.
- ≡ ○ DevOps automation work:

DevOps team mainly focus on Infrastructure as Code(IaaC - treating infrastructure same way as treating your application code) which contains tasks like Create template, Provision Server, Install software, Configure software and deploy applications. DevOps team is also responsible for Continuous deployment and continuous deleivery.

- **IaaC**: Manual approach - Provision virtual server machine, Install java, Install tomcat, configure tomcat, deploy apps. If there are many applications and their instances, which are developed using different languages then manual approach will be difficult. **Infrastructure as code** - Create template, Provision virtual server, Install server, configure software, deploy apps

by using tools like terraform, ansible etc. terraform helps to provision servers and then ansible helps to configure the software. Finally we can deploy our app to these virtual machine by using either jenkins or azure DevOps.

- ○ Because of Infrastructure as code, developers can self provision an environment, deploy the code, test and find problems by their own without any help from operation team.
- ○ DevOps main target is to automating as much software development activities as possible.
- ○ So Agile and DevOps together helping us to build amazing products.
- ○ **Continuous Integration**
  Commit code, code quality check,

unit test run, integration test run, packaging, building image and pushing image to docker hub

- **Continuous Deployment**
  From docker hub take the image and deploy it to development and testing(qa) environments
  Run automation tests like smoke test, load test, and performance test
- **Continuous Delivery**
  Once it is approved from testing environment enable some flag so automatically build will be deployed to staging environment and on next approval it will be deployed to production environment.

==========

CI, CD & Continuous Delivery

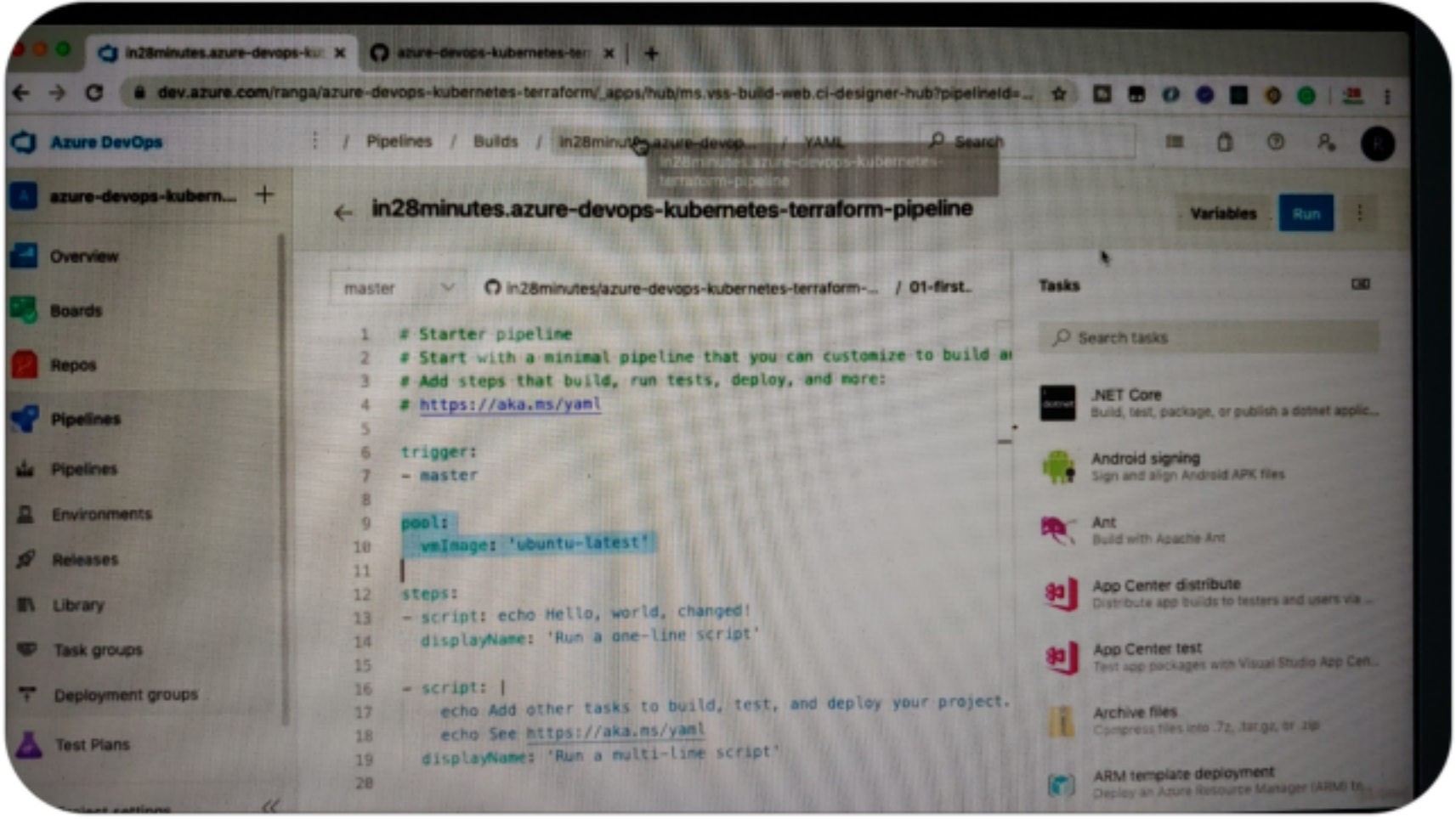
- Create a new account and login to Microsoft azure DevOps.
- In the home page, we can create multiple organizations and under each organization we can create multiple project. By default azure devops Create one such default organization. Now create a new project 'my-devops' under it. Note that we usually create independent project in DevOps for every projects in Github repo, so that we can have separate pipeline configurations for each of those Github projects.
- **Setting up Git repo for Azure DevOps pipeline**: Note that Azure also provides its own version of Git repository whose url can be seen in the devops portal. We can use that git repo also. Now create a new project in Git 'my-devops-pipelines'. Copy the

project provided in the course, commit and push the codes.

- **Creating your first Azure DevOps Pipeline(in .yml extension)**: Open azure DevOps home page. Click on Pipelines option from the left menu. Follow below steps - 1. Where is your code: point to the above git repo. 2. Enter the github repo credentials 3. Configure your pipeline - There are multiple options provided like Build a docker image, build and push an image to ACR, Build and push an image to ACR and deploy to AKS, Starter pipeline, Existing azure pipelines yaml files. For the initial step select the option Starter pipeline. 4. Review your pipeline YAML - This will create a new default minimal configured azure-pipelines.yml file in the same Github project and commit

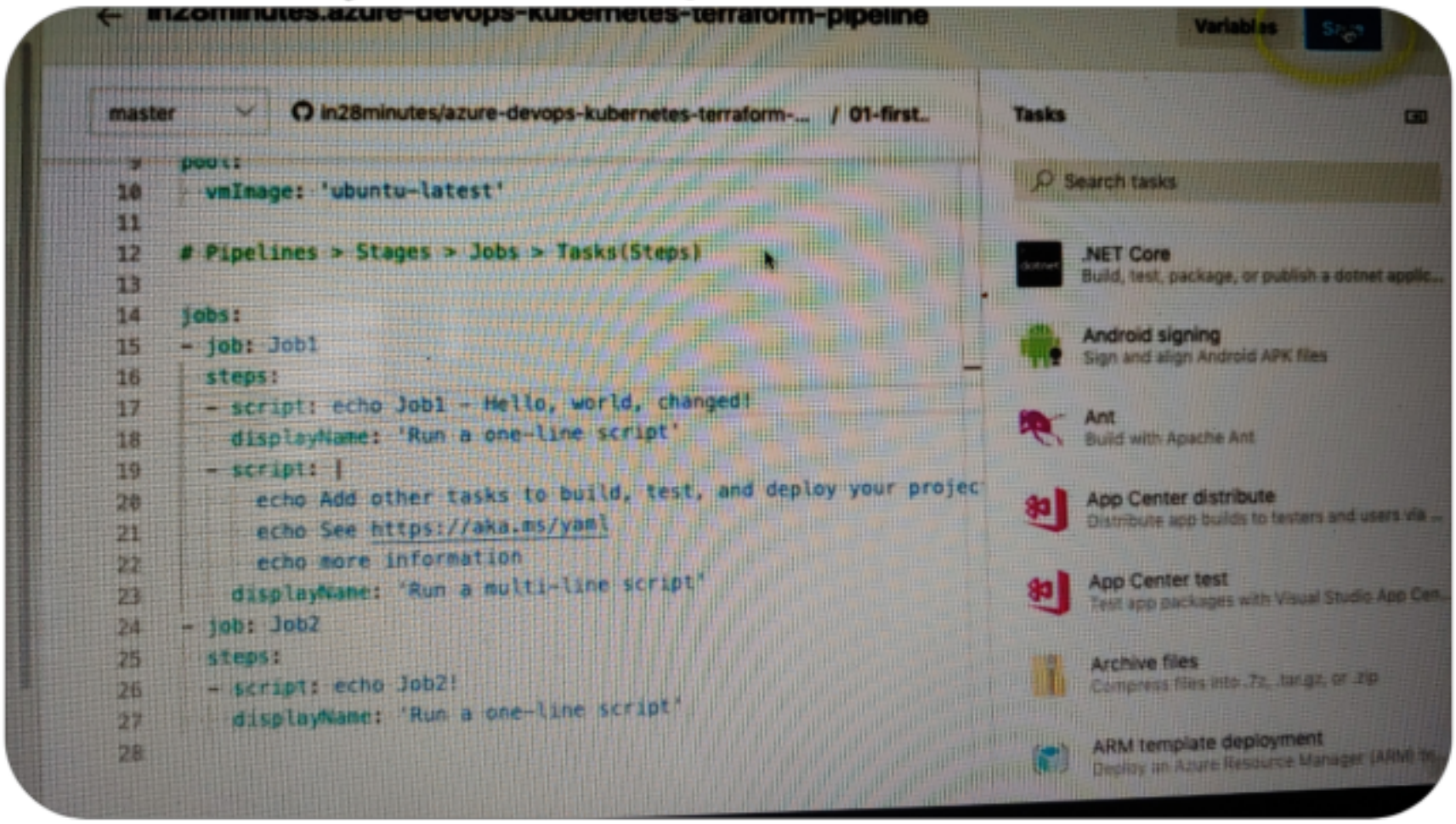
and push it from DevOps. As soon as this file is pushed, Azure DevOps pipeline will start running and executes the stages/jobs/ steps/tasks as defined in the file.

- **Azure DevOps Agents, Pipelines, Stage, Jobs & Steps:** The azure-pipelines.yml file has below configurations. 1. trigger - tells on which branch when commit happens need to trigger this pipeline. 2. pool.vmImage - to run a pipeline we need a special machine called Agent. Here we tell use Ubuntu as an Agent machine. 3. Define multiple steps sequentially. Here we define 2 steps/tasks, about writing script to display some text message.

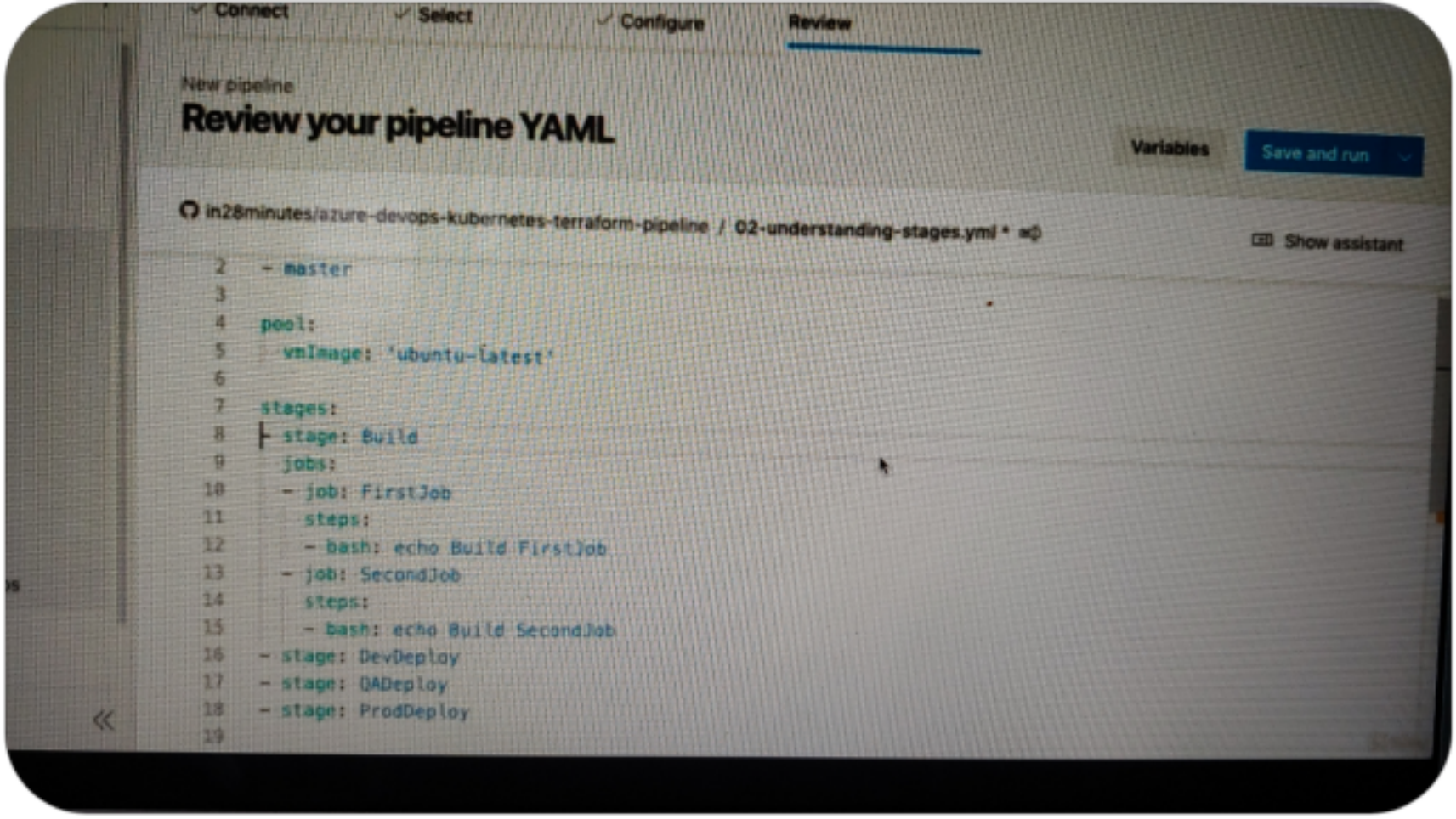


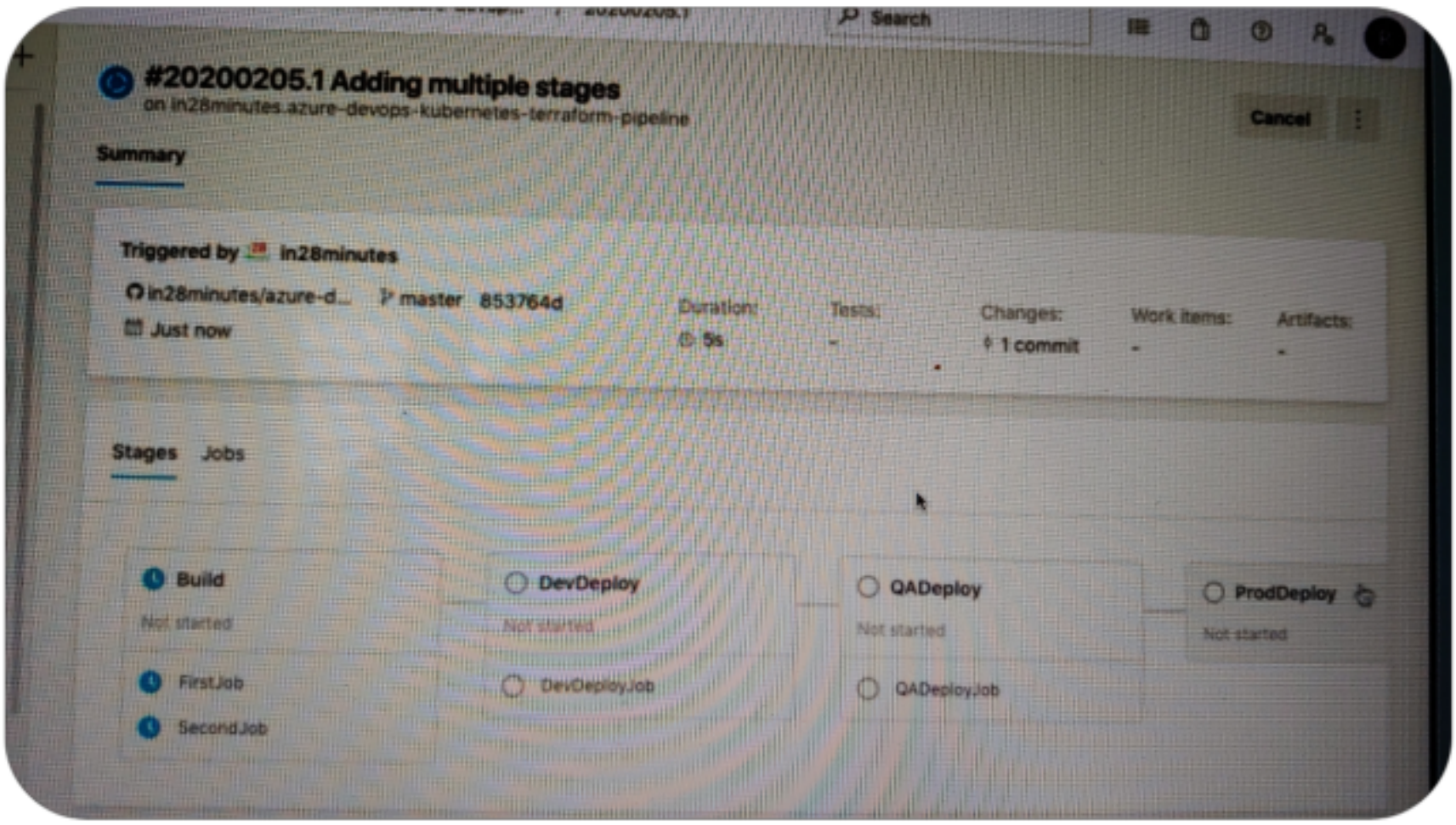
≡ ○ A single pipeline can contains multiple stages like building stage, deploying to dev or deploying to qa etc. Each stage contains multiple jobs. Each job contains multiple tasks(steps). When we have only one job then mentioning different steps of that job is enough. But if we have more than one job then we need to define all of them along with their corresponding multiple steps. Each of these jobs will run on different agent or VMs and are independent to each other so

they will run parallel.



- Using dependsOn with jobs: use dependsOn tag in one job with other dependent job name. - job: Job2 dependsOn: Job1.
- **Azure DevOps Stages:** we can define multiple stages, where each stage has multiple jobs with multiple steps. Next stage will run only after the previous stage is completed successfully.

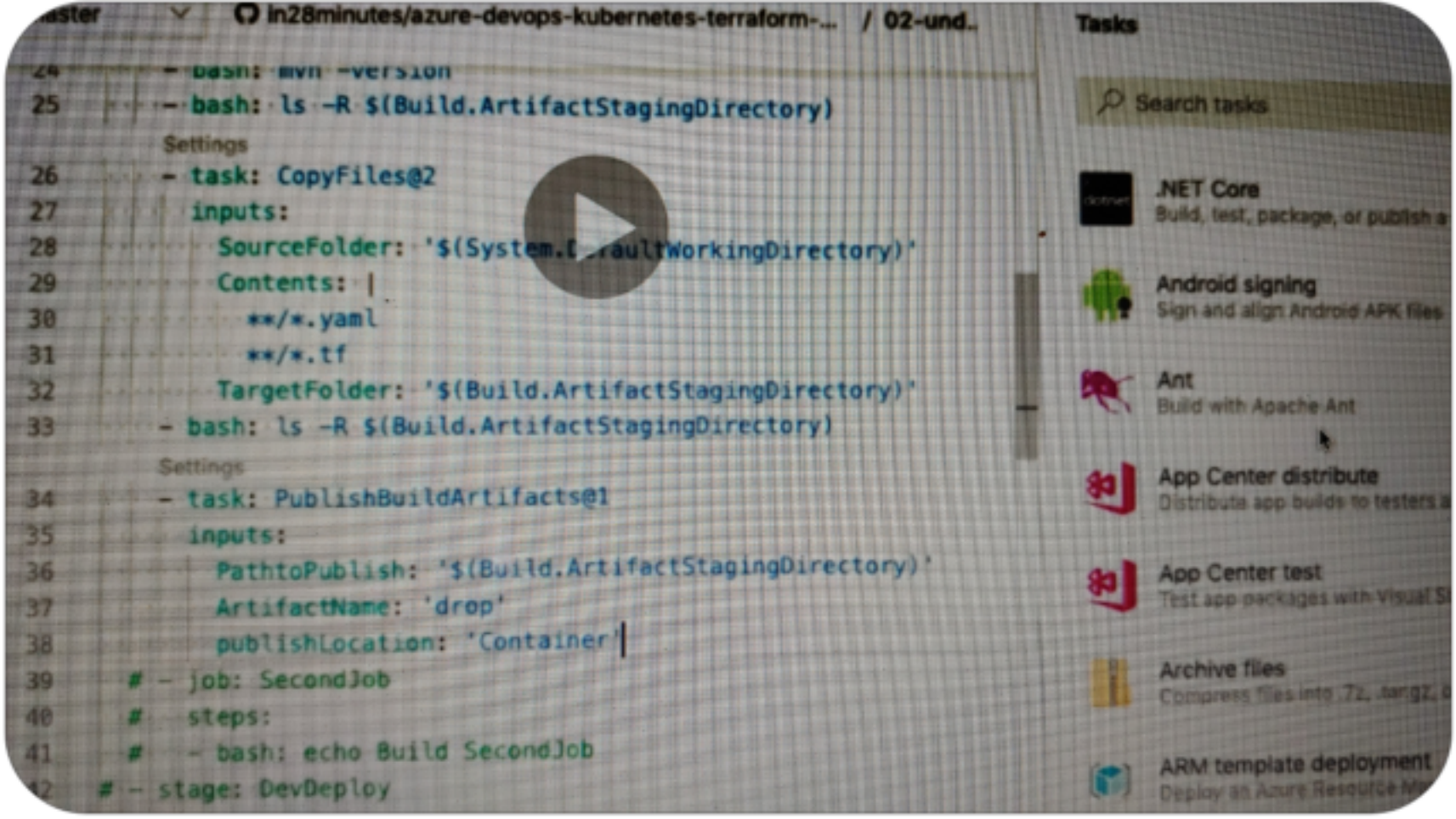




- **Variables in Pipelines:** we can define multiple variables with values. And then we can use these variables in yml, instead of directly hard coding values in

pipeline.yml. We can define such variables either at the pipeline level or at each stage level or also at each job level. Ex: - bash: echo $(pipelineLevelVaraible).

- **DevOps Provided Predefined variables:** By default azure provides multiple predefined variables like AgentName, AgentMachineName, BuildId, BuildNumber etc.
- **DevOps Provided Predefined Tasks**: Azure DevOps provides predefined tasks which we can use in our pipelines like copy files, publish Artifacts etc. If an output of one stage is required as an input to another stage, the way we can do that is through creating and publishing an artifact. We can create a separate task for the same.



Create a new pipeline with Deployment Job instead of normal Job which support independent environment variable setup:

Jobs of type deployment has some additional functionalities compared to plain job like deployment strategy and environment specific configurations. As part of each deployment job we can have independent environment where we can configure environment specific security, approvals and checks. Through approvals and

checks we can add email ids of approvals. So once the current deployment job which is DEV is succeeded, an approval mail will send to the configured approver. Until the approver check and approves, the next deployment job which is QA will not run. It will be in the waiting state.

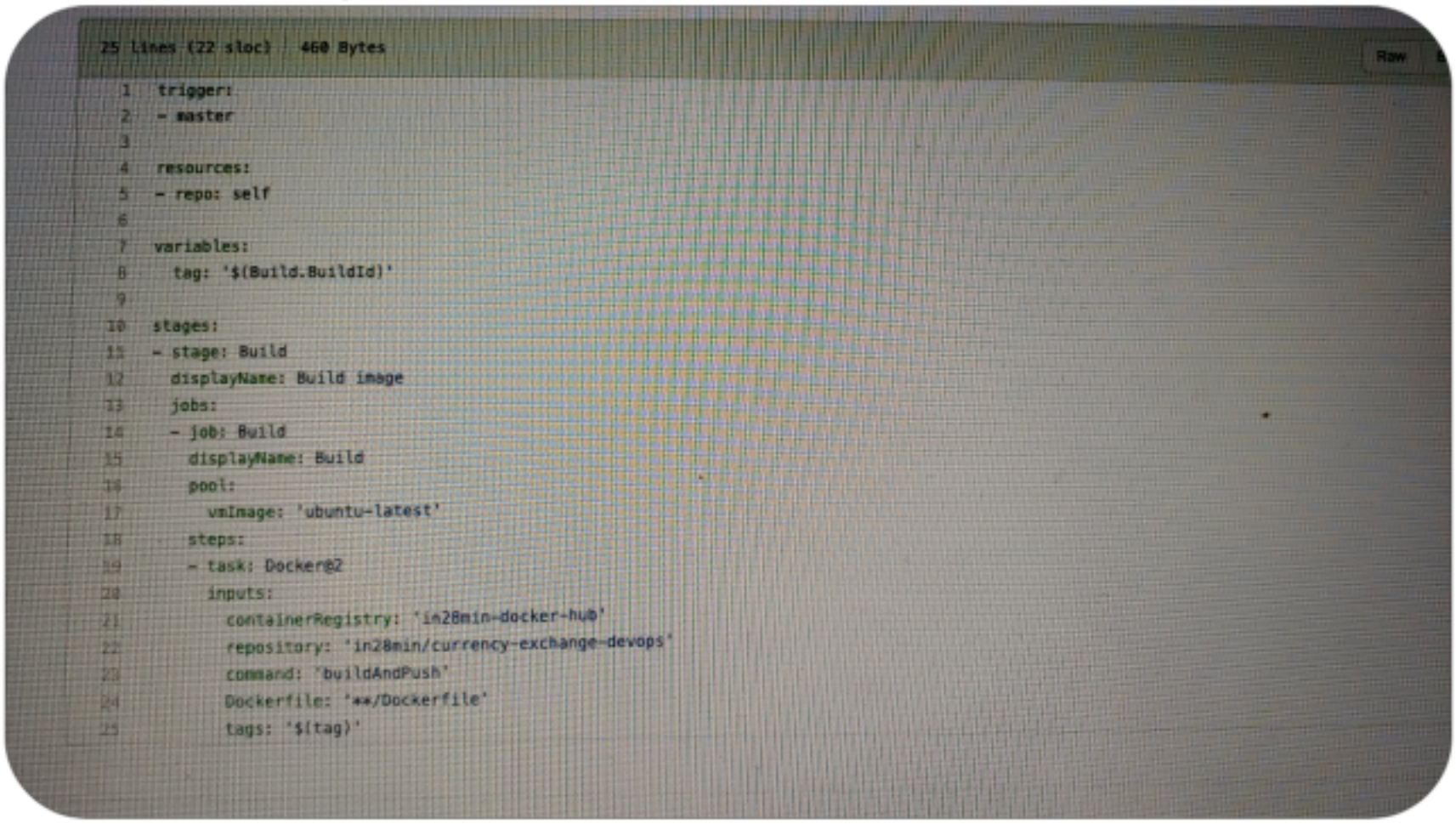
```
#    mac:
#      operatingSystem: 'macos-latest'

pool:
  vmImage: 'ubuntu-latest'

stages:
- stage: Build
  jobs:
  - job: BuildJob
    steps:
    - bash: echo "Do the build"
- stage: DevDeploy
  jobs:
  - deployment: DevDeployJob
    environment: Dev
    strategy:
      runOnce:
        deploy:
          steps:
          - script: echo deploy to Dev
- stage: QADeploy
  jobs:
  - deployment: QADeployJob
    environment: QA
    strategy:
      runOnce:
        deploy:
          steps:
          - script: echo deploy to QA
```

- **Create a new pipeline for build and push Docker image (Normally call it as Build Pipeline):** We can create a new pipeline which will create a new

image by reading the Dockerfile which exist in the Github project root folder and push it to Docker hub repo whenever a new commit is pushed to that Github repo. For this first we need to create a new service connection in DevOps portal through project setting option by selecting the docker hub option and typing required permission details.

- Now create a new pipeline by selecting the same Github project and configure your pipeline option as 'Docker(build a docker image)' option(Not the starter pipeline option) with the below details - previously created docker hub service container registry name, docker hub repository name where we need to push the image, actual command as 'buildAndPush' and buildId as an

image tag, Dockerfile path which exists in the root path of the project.



- **Create a new pipeline for download image and deploy it to different environment(Normally call it as a Release Pipeline):** To make use of releases we need to have few artifacts which are already built. Build pipeline is responsible to build something, create an artifact and make that artifact available for the release pipeline. This Artifact can be a

war file jar file or a new image pushed to any docker registry. Release pipeline will take these artifacts and deploy them to dev, qa or staging environment. We can have one pipeline for build and then separate pipelines for each release.

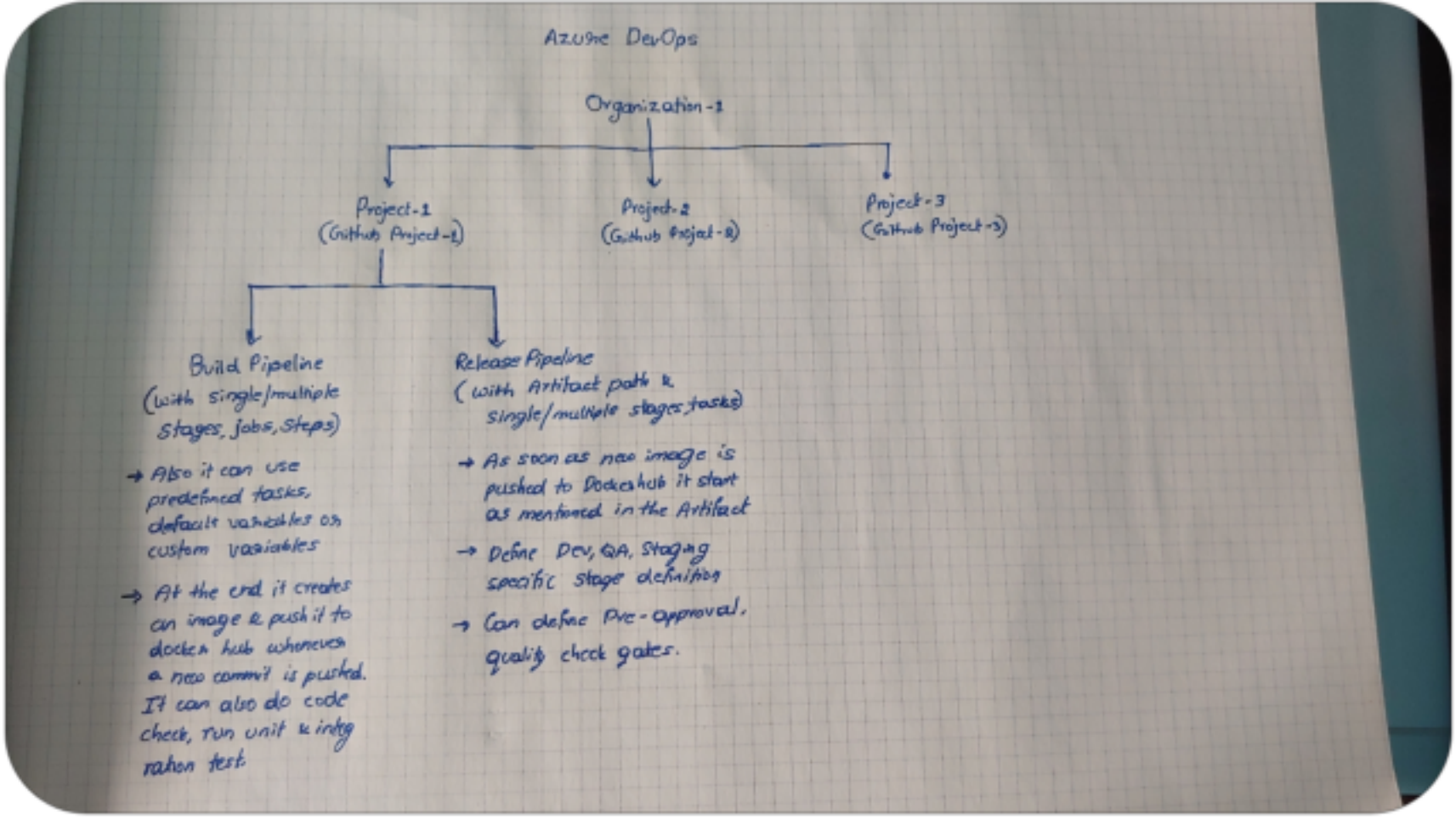
≡ ○ When we try to create a release pipeline the first thing is to select a template or start with an empty job. As release pipeline can deploy an artifact to azure app services or azure k8s services etc. If we start with an empty job we need to select the source for the artifact. Which can be either Github, Docker hub, azure repo or any our own build pipelines which produces an artifact. If we use Docker hub then these release pipeline can take the latest image and deploy it to all environments.

Here we use the previous build pipeline as a source of an Artifact for this pipeline, instead of using Docker hub as the source.

- ○ Next we can set scheduler in this release pipeline where we can configure when this release pipeline should run and deploy. For ex once in a week, once in a day etc. Or else we can disable this scheduler and enable continuous deployment which will deploy continuously.
- ○ Once the artifact is defined, next we can define multiple stages one for each environment. And in each stage we can add multiple tasks. Note that we can also make use of all the default variables which are available in the release pipeline same way as we can make use of all the default variables which are

available for build pipelines.

- ≡ ○ Also, we can define multiple independent variables for each of the dev, qa, stage or product environment and configure each stage depending on those values.
- ≡ ○ For each of the stages we can define Pre-deployment conditions like, this should be triggered after the release, after the any previous stage or it should be manually triggered. Also we can define pre-deployment approvals and any quality check through configuring new Gates.
- ≡ ○ Finally click on create a release option.



Azure DevOps, AKS with Terraform

- We can use Azure DevOps to provision Azure K8S by using terraform and then setup CI/CD cycle to deploy micro service containers. We can use 2 pipelines for each of this task.
- Terraform: Its an IAC tool to automate provisioning of cloud resources on any cloud platform. We can install it to any local

machine and use it.

- Creating and initializing Terraform project: Create any empty folder and open it in VS code. Create a new Terraform config file 'main.tf'. Terraform files will have the .tf extension. In terraform if we want to talk to any cloud provider we need 'provider' section in config file with information about region. After that run the command 'terraform init' which will download all the plugins related to the mentioned cloud provider.

CI/CD for creating AKS cluster

- **Create Terraform configuration files for the AKS cluster:** Create a main terraform configuration file main.tf. In this file mainly add the provider details as

'azurerm' (azure resource management) as defined by terraform. Then add below 3 configuration related details. 1. Create a new resource of type resource group with valid name and location. 2. Create a resource of type AKS with required parameter values. 3. And the configuration for the Terraform backend to store the Terraform state in azure storage account.

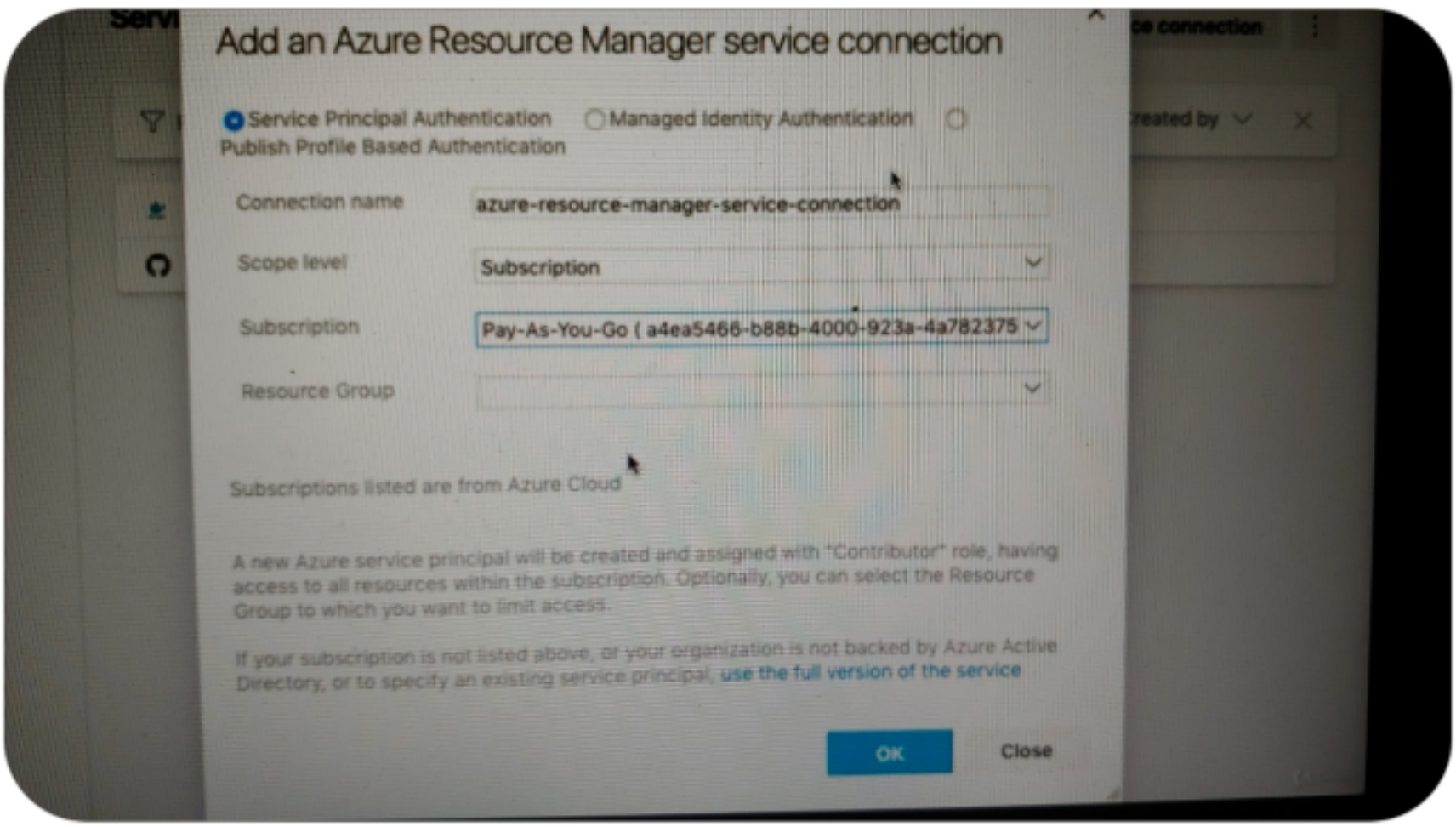
- To create the K8S we need to define following values. Name of the cluster depending on selected environment, location, resource group, dns_prefix, machine type as linux and the corresponding admin username and ssh_key, create a default node pool with name, node count and VM size which will be associated with the cluster, service principle with

client id and secret to talk to azure from AKS and assigning a tag.

- ≡ ○ Then create variables.tf file and define each variable name and corresponding value. Variable environment can have default value as 'dev'
- ≡ ○ To create AK cluster from Azure DevOps we need the value of clientId, clientSecret and sshPublicKey. To get these use the below steps. First install Azure Client to local machine. Then login to azure by command 'az login', which will display bunch of information. Here the id is nothing but subscription_id.
- ≡ ○ With the subscriptionId create a service account in azure through the command 'az ad sp create-for-rbac --role="Contributir" --scopes="/subscriptions/

subscription_id". This will returns the appId and password for us.

- Now to create ssh_public_key use the command 'ssh-keygen -m PEM -t raw -b 4096'. This will create rsa file under .ssh folder.
- **Creating a new Azure DevOps Pipeline for provisioning Azure Kubernetes Cluster(Terraform init command):** First create a new service connection in Azure DevOps for Azure Resource Manager. Do not select the resource group as we will create new one.



- Now download VS code extension Terraform and Terraform build & Release Tasks and install them into DevOps.
- Now start with creating new pipeline with starter pipeline option for the AKS cluster creation. Select Terraform CLI from tasks search bar in DevOps, choose init command, configure default directory to point to where terraform configuration files main.tf and variables.tf file exists. Pass client_id, client_secret, ssh_public_key values in command options. And create corresponding new variables and pass the actual values. For ssh_public_key create a new library and upload the files which we created earlier under secure files option. To use this public key create a new task for

'download secure file' and secure file name. And select Backend type as azurerm which will store Terraform state in Azure storage account. Select the previously created service connection. Create backend in difference resource group.

- **Updating previous Pipeline for provisioning Azure Kubernetes Cluster(Terraform Apply command):** Select Terraform CLI from tasks search bar in DevOps, choose apply command this time, fill in command options, environment azure subscription, and click on add it. And finally run this new pipeline which will run terraform init command first to install all azure related plugins and then it runs terraform apply command and create AKS cluster.

```yaml
steps:
- script: echo K8S Terraform Azure!
  displayName: 'Run a one-line script'
- task: DownloadSecureFile@1
  name: publickey
  inputs:
    secureFile: 'azure_rsa.pub'

- task: TerraformCLI@0
  inputs:
    command: 'init'
    workingDirectory: '$(System.DefaultWorkingDirectory)/configuration/iaac/azure/kubernetes'
    # commandOptions: '-var client_id=$(client_id) -var client_secret=$(client_secret) -var ssh_public_key=$(publickey.secureFilePath)'
    backendType: 'azurerm'
    backendServiceArm: 'azure-resource-manager-service-connection'
    ensureBackend: true
    backendAzureRmResourceGroupName: 'terraform-backend-rg'
    backendAzureRmResourceGroupLocation: 'westeurope'
    backendAzureRmStorageAccountName: 'storageacctrangaxyz'
    backendAzureRmContainerName: 'storageacctrangacontainer'
    backendAzureRmKey: 'kubernetes-dev.tfstate'

- task: TerraformCLI@0
  inputs:
    command: 'apply'
    workingDirectory: '$(System.DefaultWorkingDirectory)/configuration/iaac/azure/kubernetes'
    environmentServiceName: 'azure-resource-manager-service-connection'
    commandOptions: '-var client_id=$(client_id) -var client_secret=$(client_secret) -var ssh_public_key=$(publickey.secureFilePath)'

- task: TerraformCLI@0
  inputs:
    command: 'destroy'
    workingDirectory: '$(System.DefaultWorkingDirectory)/configuration/iaac/azure/kubernetes'
    environmentServiceName: 'azure-resource-manager-service-connection'
```

- We can also define output.tf file in the same place where main.tf is defined which can contains everything about the k8s cluster which is created as output variables.
- To connect to AKS from azure CLI type the command 'az aks get-credentials --name k8stest_dev --resource-group kubernetes_dev'. And then type 'kubectl get service' we should see default k8s service.
- Note: We can write another task to destroy the K8S Cluster

through Terraform destroy task as mentioned in the above screenshot.

CI/CD for build and release pipeline

- **Creating a new Azure DevOps pipeline for deploying micro services to AKS:** First of all create a new service connection for Kubernetes from DevOps portal's project settings option by selecting azure subscription option, previously created AKS cluster name and then by providing valid connection name. Note that till now we created similar service connection for Github, Docker hub and Azure resource manager.
- **Creating a new pipeline in yml format**: Now go to pipeline and start creating a new pipeline by

selecting the project from Github and starter pipeline option. Now we need to edit this starter pipeline to add 2 stage definitions. First stage is for build & push docker image, copy all the yml files from working directory to Artifact Staging Directory and publishing this Artifact Staging Directory. Second stage is to downloading these artifacts from Artifact Staging Directory to Artifact Directory and deploying K8S manifest files to AKS cluster along with docker image.

- **Adding stage 1 definition to the existing pipeline**(contains 3 tasks. One for Build & push docker image, another one is to copy all yml files from current working directory to Artifact Staging Directory and the last one is to publish this Artifact Staging

Directory)

- ○ For the first task, reuse the previously created pipeline which has the task definition for the Build and push image.
- ○ Now for the second task select a new task 'Copy files' from task search bar with the following details - Source folder as '$(System.DefaultWorkingDirectory)', content as '**/*.yml' and target folder as $(Build.ArtifactStagingDirectiry)
- ○ Now for the third task create a new task by selecting the DevOps task named 'Publish build artifacts' with the below configurations - Path to publish as $(Build.ArtifactStagingDirectiry), Artifact name as K8S 'manifests' file, Artifact publish location as 'Azure pipelines' and add the task.

This will create an Artifact(Its nothing but simply a folder with name as 'manifests' which contains yml files) and make them available for the next pipelines.

```yaml
# Stage 1
# Build Docker Image
# Publish the K8S Files

- stage: Build
  displayName: Build image
  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: 'ubuntu-latest'
    steps:
    - task: Docker@2
      inputs:
        containerRegistry: 'in28min-docker-hub'
        repository: 'in28min/currency-exchange-devops'
        command: 'buildAndPush'
        Dockerfile: '**/Dockerfile'
        tags: '$(tag)'
    - task: CopyFiles@2
      inputs:
        SourceFolder: '$(System.DefaultWorkingDirectory)'
        Contents: '**/*.yaml'
        TargetFolder: '$(Build.ArtifactStagingDirectory)'
    - task: PublishBuildArtifacts@1
      inputs:
        PathtoPublish: '$(Build.ArtifactStagingDirectory)'
        ArtifactName: 'manifests'
        publishLocation: 'Container'
- stage: Deploy
```

- **Adding stage 2 definition to the same existing pipeline**(Contains 2 tasks. First one for downloading Artifacts with name as 'manifest' from Artifact Staging Directory to Artifacts Directory and second one is for take the deployment.yml file from manifest path and drploying them

to AKS cluster with Docker image of the micro service)

- For the first task, Select a new task 'Download Pipeline Artifacts' from DevOps tasks search bar and configure the below details - Download artifact produced by option as 'Current run', Artifacts name as 'manifests', matching patterns as '**/*.yml', Destination directory as '$(System.ArtifactsDirectory)' and finally click on Add.
- Now for the second task, select the task 'Deploy to Kubernetes' from task search bar with the following details - Action as 'deploy', Kubernetes service connection as previously created service connection for kubernetes, Namespace as default, Strategy as none, manifest file path as '$

(System.ArtifactsDirectory)/configuration/kubernetes/deployment.yml', containers as name of the docker hub image repository with specific tag and click on Add. This basically runs the command 'kubectl apply' on deployment.yml file

- Note that in the deployment.yml file we have not mentioned the tag name. We just mention the image path as 'image: in28min/currency-exchange-devops'. Now the specific tag of the image will be added by the last task 'Deploy to kubernetes' containers configuration as mentioned just above.

```yaml
- stage: Deploy
  displayName: Deploy image
  jobs:
  - job: Deploy
    displayName: Deploy
    pool:
      vmImage: 'ubuntu-latest'
    steps:
    - task: DownloadPipelineArtifact@2
      inputs:
        buildType: 'current'
        artifactName: 'manifests'
        itemPattern: '**/*.yaml'
        targetPath: '$(System.ArtifactsDirectory)'
    - task: KubernetesManifest@0
      inputs:
        action: 'deploy'
        kubernetesServiceConnection: 'azure-kubernetes-connection'
        namespace: 'default'
        manifests: '$(System.ArtifactsDirectory)/configuration/kubernetes/deployment.yaml'
        containers: 'in28min/currency-exchange-devops:$(tag)'
# Stage 2
# Download the K8S Files
# Deploy to K8S Cluster with Docker Image
```

- **Ideal way to structure DevOps Pipelines:** We have created 2 pipelines in previous section, one for Provisioning cluster and another one is for deploying image. Both of these pipelines are created in the same repository. In production/test/dev environment we will be having one K8S cluster and multiple images of the micro services will be deployed. So we will be having multiple pipelines to deploy each of these micro services.

- Pipeline for provisioning cluster: It is always good to create a separate Github repository for provisioning cluster in azure and write a separate pipeline for the same. This pipeline is only responsible for Infrastructure as code. All that will be presented in this project is All Terraform configuration files.
- Pipeline for deploying image: Now we will be having individual separate Github repository for each of the project and write a separate pipeline yml file for each of them in the same project so that each pipeline will be responsible for deploying its project image.

Azure DevOps with Boards & Backlogs

- Open a website Azure DevOps

Demo generator where we can create many demo projects and play with it.

- ○ By default the demo project creates Azure boards with multiple work items, Repos where the source code exists, pipelines where simple pipelines are defined, test plans where some test suits and test cases for each work items are created and artifacts where jar or war files can be stored.
- ○ Azure Boards - Project management tool like Jira. Azure provides work items of types Epics, features, user stories(product backlog item), tasks, bugs, issues(impediments) test suits, test plan, test cases etc. It also provides backlog items of types portfolio backlog, requirement backlog, iteration

backlog. Boards also contains all the sprint and queries(For ex number stories which are in done state, number of bugs which are in open state etc)

- Azure Repos - Source management tool like Github
- Pipelines - To create CI/CD builds
- Test plans - To manage your test suits
- Artifacts - Repository for jar file, npm package etc

Summary

Pipeline 1:

- First step will be to download a file which has a public key of Azure and use it in terraform
- Second step will be to run terraform init
- Third step will be to run terraform apply which would create a kubernetes cluster with mentioned nodes and also a backend store for the terraform states.
- Create a new connector service in azure devops which can connect to this kubernetes cluster.

Pipeline 2:

Stage 1:

- Compile, test, package and build the docker image and push it to docker hub
- Publish the kubernetes workloads and service or any other configuration files. Upload all the kubernetes artifact files. This is required to use the previous stage artifact in the next stage.

Stage 2:

- Download the kubernetes workloads and service or any other config files. Download all the kubernetes artifact files with the extension of yaml
- Once it is downloaded deploy these files against the kubernetes cluster by using the kubernetes connector service which is created earlier with the command kubektl deploy
- Deploy the docker image to the kubernetes cluster.

- For provisiong we should have a separate repository so that we can a separate pipeline for that. (1 iaac pipeline)
- And for each of the microservices we should have separate github repository and pipelines. (Multiple ci/cd pipelines)

==========END==========