Dancing with Docker images in Azure DevOps

@gkarthics

About me

Karthik is a CNCF Speaker, DevOps Architect and Cloud Native App Developer who is passionate about Cloud and Cloud-Native Infrastructure, Developer Tools & Experience and Open-Source Enthusiast.

He is also a passionate writer who writes about the technologies and technology challenges.

Karthikeyan Govindaraj

Know more about me!!!

Problem Scenario

Containerizing and deploying multiple instances of an app that talks to one variant of a database per instance.

Connecting to Oracle, the app needs oracle-based drivers installed in the OS

Connecting with Sybase, its dependency lib must be available

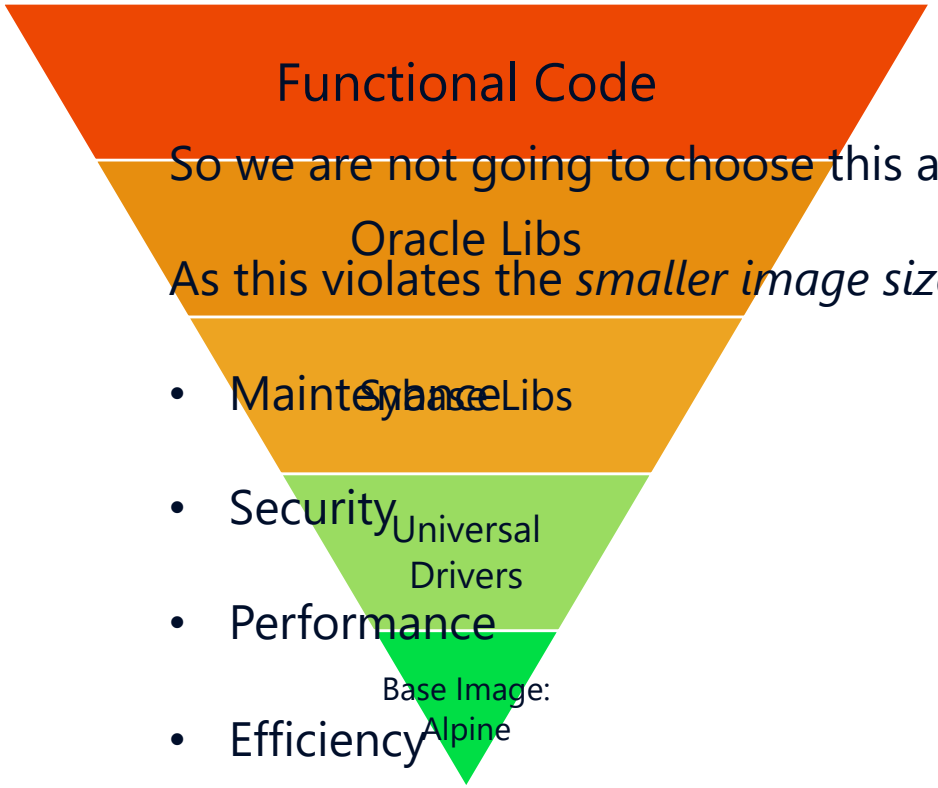
Pretty much rest is universal viz., MySQL, MSSQL, PostgreSQL etc

What can we do?

Approach 1:

Pre-install all the drivers viz., Sybase, Oracle in the base image.

Approach 1



So we are not going to choose this approach.

As this violates the *smaller image size policy*; which will create hassle in

- Maintenance
- Security
- Performance
- Efficiency

Functional Code: **~xyz GB**

Oracle libs: **X + Y + Z + 2.5MB**

Sybase libs: **X + Y + 2.5MB**

Universal drivers: **X + 2.5MB**

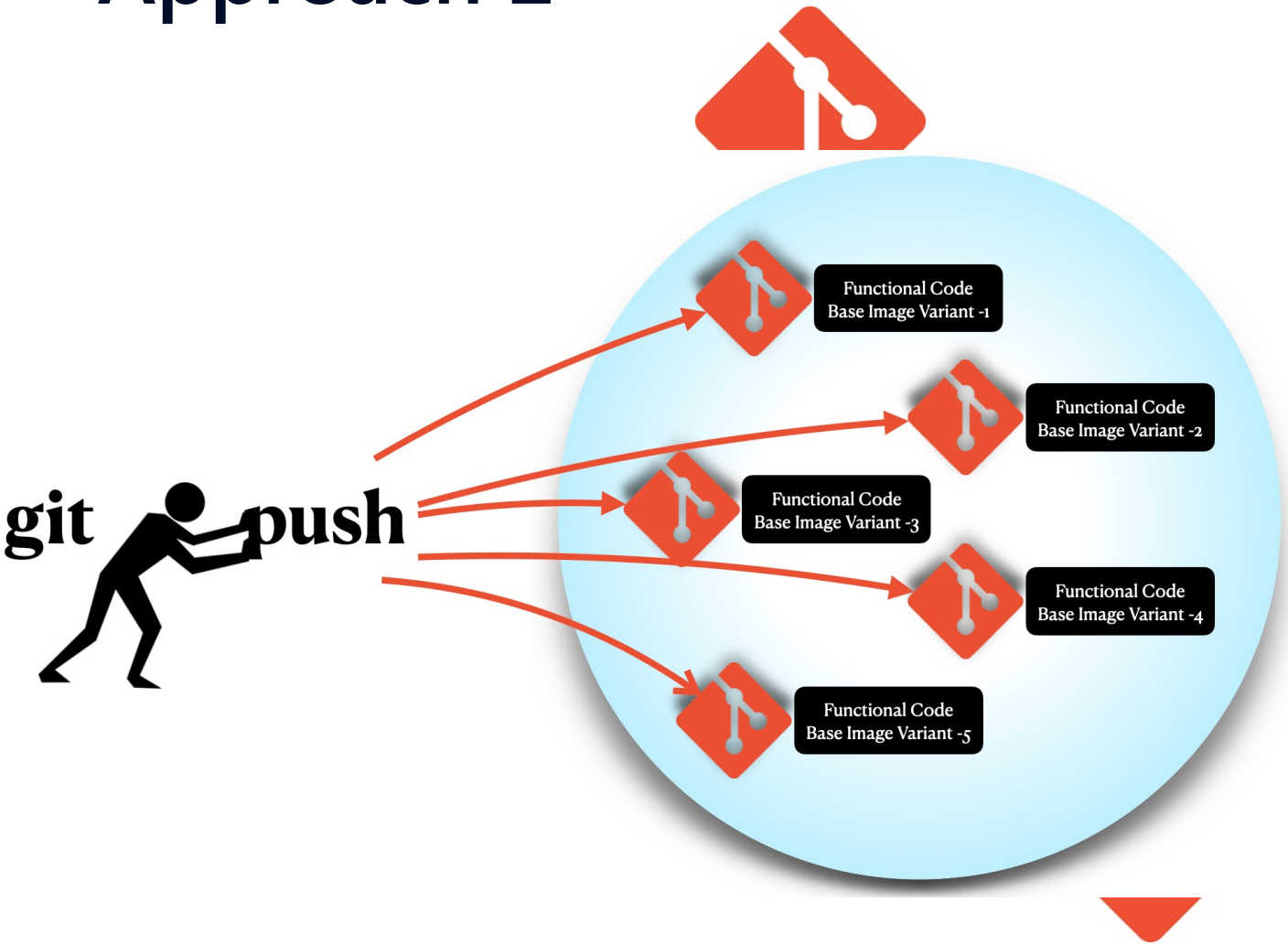
Only Base image, alpine:latest **2.5MB**

What can we do?

Approach 2:

Create as much as repositories with the same functional code with diff image construction logic.

Approach 2



Functional Code
+Base Image+Oracle

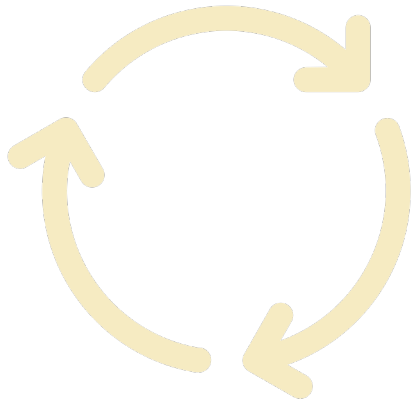
Functional Code
+Base Image+Variant n

So we are not going to choose this approach as well.

As this duplicates the code in multiple repositories and violates the source of truth policy for the functional code.

Bonus Golden Option

- Docker allows user to pass arguments during build time.
- Azure repositories allows to trigger one repository's pipeline from another.
- Azure pipeline jobs can be run in host or in containers.
- Azure variables can be used to hold data between pipeline jobs.
- Containers provide isolation from host and allow you to add tools and dependencies for build.
- Azure Pipelines allows to use repositories as resources to define reusable content, logic, and parameters.



… contd.,

- Create a repository with *Functional Code*.
- Create Dockerfile in the Functional Code repo with universal drivers and a build argument for base image; defaults to *alpine:latest*.
- Create a separate repository for Oracle dependencies installed base image.
- Create a separate repository for Sybase dependencies installed base image.

docker.yml in **ci_templates** repo

Docker build template

```yaml
jobs:
  - job: build_push
    displayName: Build and Push
    steps:
      - task: Docker@2
        displayName: Build and Publish image via Docker@2
        inputs:
          command: buildAndPush
          containerRegistry: SERVICE_CONNECTION
          repository: namespaces/$(imageName)
          ${{ if startsWith(variables['Build.SourceBranch'], 'refs/tag') }}:
            tags: $(Build.SourceBranchName)
          ${{ if startsWith(variables['Build.SourceBranch'], 'refs/heads') }}:
            tags: $(Build.BuildNumber)
```

Need to build docker images for all the repositories.

Re-use docker image build logic using repositories as resource in Azure Pipelines.

Generic App

Create a repository named *generic-app* and check-in all the functional code.

Create a Dockerfile for the repository with build arguments for the base image defaulting to alpine.

```
1   ARG BASE_IMAGE=node:alpine3.12
2   FROM $BASE_IMAGE
3   RUN mkdir -p /usr/src/app
4   WORKDIR /usr/src/app
5   COPY . /usr/src/app
6   RUN npm install
7   RUN npm run build
8   CMD ["npm", "start"]
```

```
|_ Dockerfile
|_ package.json
|_ <nodeJS application logic folders/files>
|_ azure-pipelines.yml
```

Azure Variables / Variable Groups

- Variables can hold the values that needs to pass across pipelines.
- Variables can be grouped into *Variable Groups* and access restrictions can be applied on it.
- Azure Pipelines also provides pre-defined variables that holds the pipeline scope values.
- Reg-ex can be employed to evaluate a pre-defined and user-defined variables.

… contd.,

Admin Variables

- We will store the PAT in *AZURE_DEVOPS_CLI_PAT* variable under Admin Variables group.
- This is used to authorize the triggers.
- Also used for Azure DevOps CLI login.

Platform Variables

- This is used to store the latest tag information of all the repositories.

In Scope Variables:

- We will also assign the image name in *imageName* variable.
- Determine whether the pipeline is triggered by main branch or a tag and store binary value in *isMain* and *isTag*.

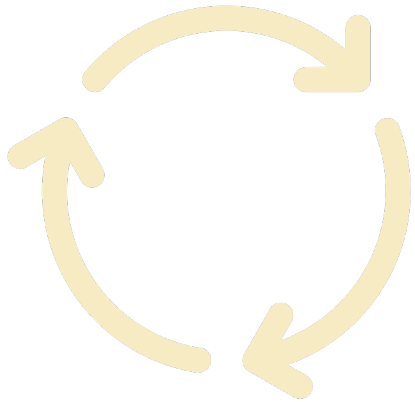
Let's build Azure Pipeline

```yaml
resources:
  repositories:
    - repository: templates
      name: MyProject/ci_templates
      type: git
      ref: main
```

Import the *ci_templates* repository as a resource in generic-app

Import all the variables and variable groups

```yaml
variables:
  - group: 'Admin Variables'
  - group: 'Platform Variables'
  - name: imageName
    value: 'generic-query-engine'
  - name: isMain
    value: $[eq(variables['Build.SourceBranch'], 'refs/heads/main')]
  - name: isTag
    value: $[startsWith(variables['Build.SourceBranch'], 'refs/tags')]
```



… contd.,

Stages are the major divisions in a pipeline. Jobs are organized into stages.

Let's define two different stage for main branch and tag build.

Define a dedicated stage for updating the Platform Variables group's tag value for the current repo.

```
18  stages:
19    - stage: mainBuild
20      condition: eq(variables.isMain, true)
21      jobs:
22        - template: docker.yml@templates
23
24    - stage: tagPublish
25      condition: eq(variables.isTag, true)
26      jobs:
27        - template: docker.yml@templates
```

Variable update stage

```yaml
# Updates the tag data in Variables
  - stage: updateVariable
    displayName: Update Tag in Variables Group
    condition: eq(variables.isTag, true)
    jobs:
      - job: VariableUpdateJobs
        container:
          image: gkarthics/ci-swiss-knife:0.1.0
          endpoint: "docker-service-connection"
        steps:
          - script: az --version
            displayName: Azure CLI Version
```

A job is a series of steps
that run sequentially as a unit.

Using container job,
since we need Azure CLI.

ci-swiss-knife image

… contd.,

```yaml
- script: |
    az config set extension.use_dynamic_install=yes_without_prompt
    echo ${AZURE_DEVOPS_CLI_PAT} | az devops login
  env:
    AZURE_DEVOPS_CLI_PAT: $(ACCESS_TOKEN)
  displayName: 'Login Azure DevOps Extension'

- script: az devops configure --defaults organization=https://azure.com/MyORG project=$PROJECT_NAME --use-git-aliases true
  displayName: 'Set default Azure DevOps organization and project'
  env:
    PROJECT_NAME: $(System.TeamProject)

- script: |
    az pipelines variable-group variable update --group-id <GROUP_ID_VALUE> --name STABLE_VERSION --value $TAG
  env:
    TAG: $(Build.SourceBranchName)
  displayName: 'Update $TAG Value in Variable Group'
```

Configure Azure CLI

Login to *az devops* via PAT

Set the defaults for organization, project etc.

Update the variable for latest tag value.

Properties

Variable group name

Platform Variables

Description

Variables that are used in building the base images for the platform

Allow access to all pipelines

Link secrets from an Azure key vault as variables

Variables

Name	Value
STABLE_VERSION	3.3.5

Oracle base variant

Build an image with Oracle Libraries that can be used as base image.

Pass the built image as argument to functional repo *generic-app*

Trigger the pipelines.

… contd.,

Using Ansible, define *requirements.yml* and *playbook.yml* to use the existing role *ansible_oracledb_client*

```yaml
---
- name: oracledb-client
  src: gkarthiks.ansible_oracledb_client
```

```yaml
---
- name: Install Oracle Client
  hosts: all
  roles:
    - oracledb-client
```

```dockerfile
FROM node:alpine3.12
COPY . .
RUN ansible-galaxy install -r requirements.yml
RUN ansible-playbook playbook.yml
```

Define docker steps to execute Oracle dependencies via Ansible

… contd.,

Similar to the *generic-app* we will create the *oracle-client* image with azure pipelines.

Update the latest tag value in the variable *ORACLE_CLIENT_VERSION*

```yaml
- script: |
    az pipelines variable-group variable update --group-id <VARIABLE_GROUP_ID> --name ORACLE_CLIENT_VERSION --value $TAG
  env:
    TAG: $(Build.SourceBranchName)
  displayName: 'Update $TAG Value in Variable Group'
```

Sybase base variant

Similar to Oracle, create a independent repository for Sybase base image as *sybase-client*

Create a Dockerfile to install the Sybase required dependencies via Ansible *playbook.yml* file

Using azure pipelines, trigger when a tag is dropped in the *sybase-client* repo

Update the latest tag info for the Sybase variable

… contd.,

The ansible playbook with Sybase required dependencies are defined as

```yaml
---
- hosts: localhost
  tasks:
    - name: Update packages
      apt:
        update_cache: yes
    - name: Install required tools
      apt:
        name: "{{ item }}"
        state: present
      become: true
      with_items:
        - openjdk-8-jre
        - make
        - g++
```

The *Dockerfile* for executing the ansible playbook would simply like as shown

```
FROM node:alpine3.12
COPY . .

RUN ansible-playbook playbook.yml
```

```yaml
- script: |
    az pipelines variable-group variable update --group-id <VARIABLE_GROUP_ID> --name SYBASE_CLIENT_VERSION --value $TAG
  env:
    TAG: $(Build.SourceBranchName)
  displayName: 'Update $TAG Value in Variable Group'
```

Pipeline Triggers

Accomplished so far	generic image with functional code and alpine base
	only a base image for Oracle platform dependencies
	only a base image for Sybase platform dependencies
Things to do	add functional code on Sybase and Oracle
	make triggers automated
	identify the source of trigger

… contd.,

Remember the build argument in the *generic-app* repo's Dockerfile

```
ARG BASE_IMAGE=node:alpine3.12
FROM $BASE_IMAGE
```

All we have to do is pass the Sybase and Oracle base image with a trigger from the corresponding repo

Azure provides a way to add pipelines as a resource

Let's start with the Oracle trigger

```
pipelines:
- pipeline: oracleclient
  source: oracle-client
  trigger:
    stages:
      - updateVariable
```

… contd.,

The generic-app pipeline will be triggered soon after the successful completion of the stage updateVariable in the oracle-client repo.

Triggers the default branch.

We need a specific build as we want to flip the base image.

Let’s add a variable to evaluate the trigger source.

```
- name: isOracleTrigger
  value: $[eq(variables['resources.triggeringAlias'], 'oracleclient')]
```

```
pipelines:
- pipeline: oracleclient
  source: oracle-client
  trigger:
    stages:
      - updateVariable
```

… contd.,

Let's use the *isOracleTrigger* to exclude unnecessary main build.

```
- stage: mainBuild
  condition: and( eq(variables.isMain, true), ne(variables.isOracleTrigger, true))
```

Create another docker build template which accepts the build arguments and tag differently and save it as *upstream-docker.yml*

This can be re-used for Sybase as well.

```yaml
parameters:
  - name: tag
    type: string
    default: 'latest'
  - name: buildArgs
    type: string
    default: ''

jobs:
  - job: BuildAndPush
    displayName: Build job
    steps:
      - task: Docker@2
        displayName: Build image via Docker@2
        inputs:
          command: build
          ${{ if ne(parameters.buildArgs, '') }}:
            arguments: ${{ parameters.buildArgs }}
          ${{ if ne(parameters.tag, '') }}:
            tags: ${{ parameters.tag }}
          containerRegistry: ${{ service_connection }}
          repository: namespaces/$(imageName)

      - task: Docker@2
        displayName: Push image via Docker@2
        inputs:
          command: push
          containerRegistry: ${{ service_connection }}
          repository: namespaces/$(imageName)
          ${{ if ne(parameters.tag, '') }}:
            tags: ${{ parameters.tag }}
```

upstream-docker.yml in **ci_templates** repo

@gkarthics

… contd.,

Define a stage exclusively for oracle-client pipeline triggered run

```yaml
- stage: TriggerOracleBuildPush
  condition: eq(variables.isOracleTrigger, true)
  variables:
    imageName: "query-engine-oracle-client"
  jobs:
    - template: upstream-docker.yml@templates
      parameters:
        tag: $(STABLE_VERSION)_$(ORACLE_CLIENT_VERSION)
        buildArgs: '--build-arg BASE_IMAGE=oracle-client:$(ORACLE_CLIENT_VERSION)'
```

```dockerfile
FROM oracle-client:0.1.0
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . /usr/src/app
RUN npm install
RUN npm run build
CMD ["npm", "start"]
```

The resulting to-be executed Dockerfile for this will be

… contd.,

To trigger the sybase-client base variant, everything is similar to Oracle.

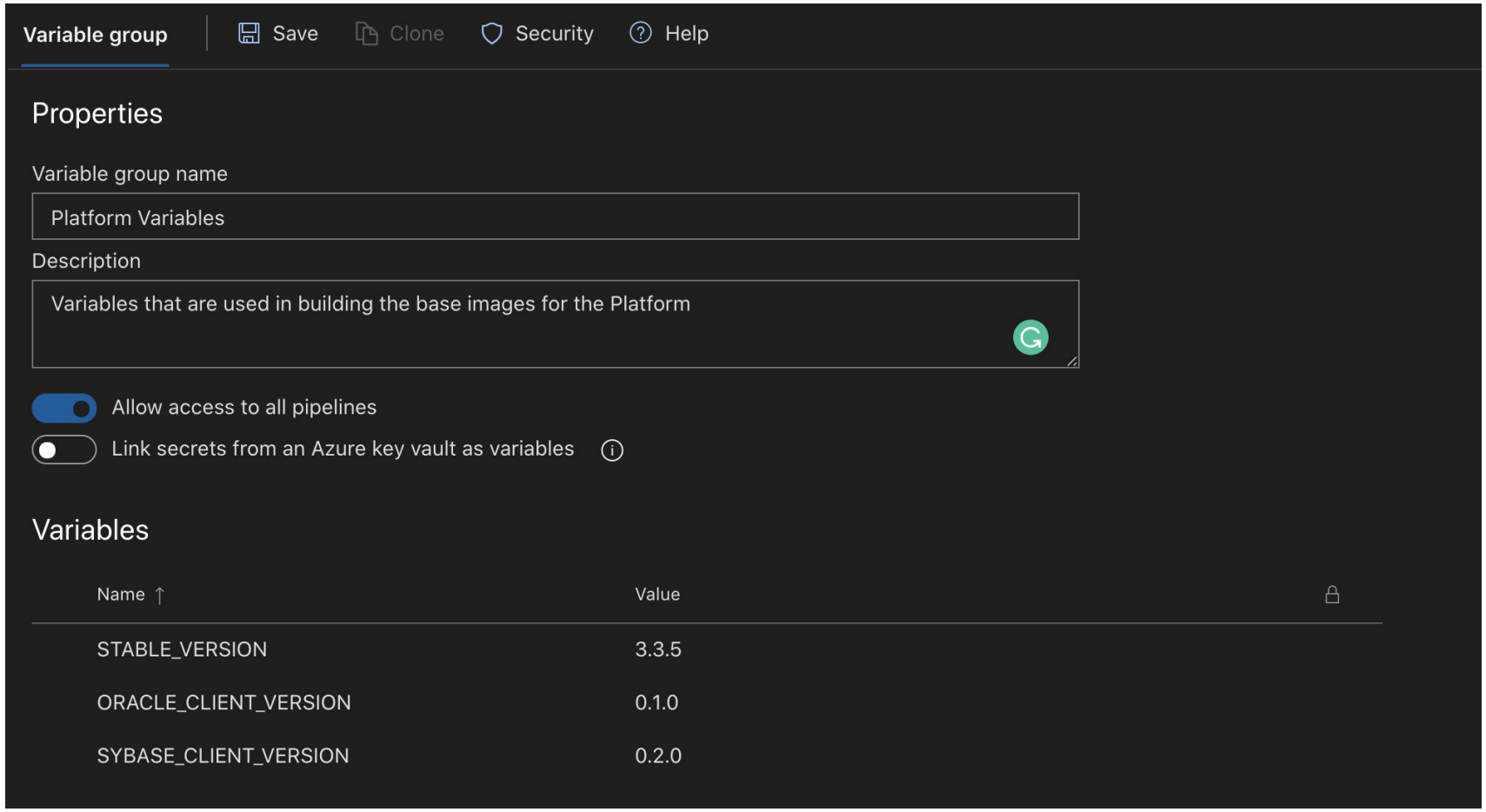
Define a variable to evaluate the trigger source

```yaml
- name: isSybaseTrigger
  value: $[eq(variables['resources.triggeringAlias'], 'sybaseclient')]
```

```yaml
- stage: TriggerSybaseBuildPush
  condition: eq(variables.isSybaseTrigger, true)
  variables:
    imageName: "query-engine-sybase-client"
  jobs:
    - template: upstream-docker.yml@templates
      parameters:
        tag: $(STABLE_VERSION)_$(SYBASE_CLIENT_VERSION)
        buildArgs: '--build-arg BASE_IMAGE=sybase-client:$(SYBASE_CLIENT_VERSION)'
```

… contd.,

The final variable group will look something like below with all the *STABLE_VERSION*, *ORACLE_CLIENT_VERSION* and *SYBASE_CLIENT_VERSION* with its values updated.

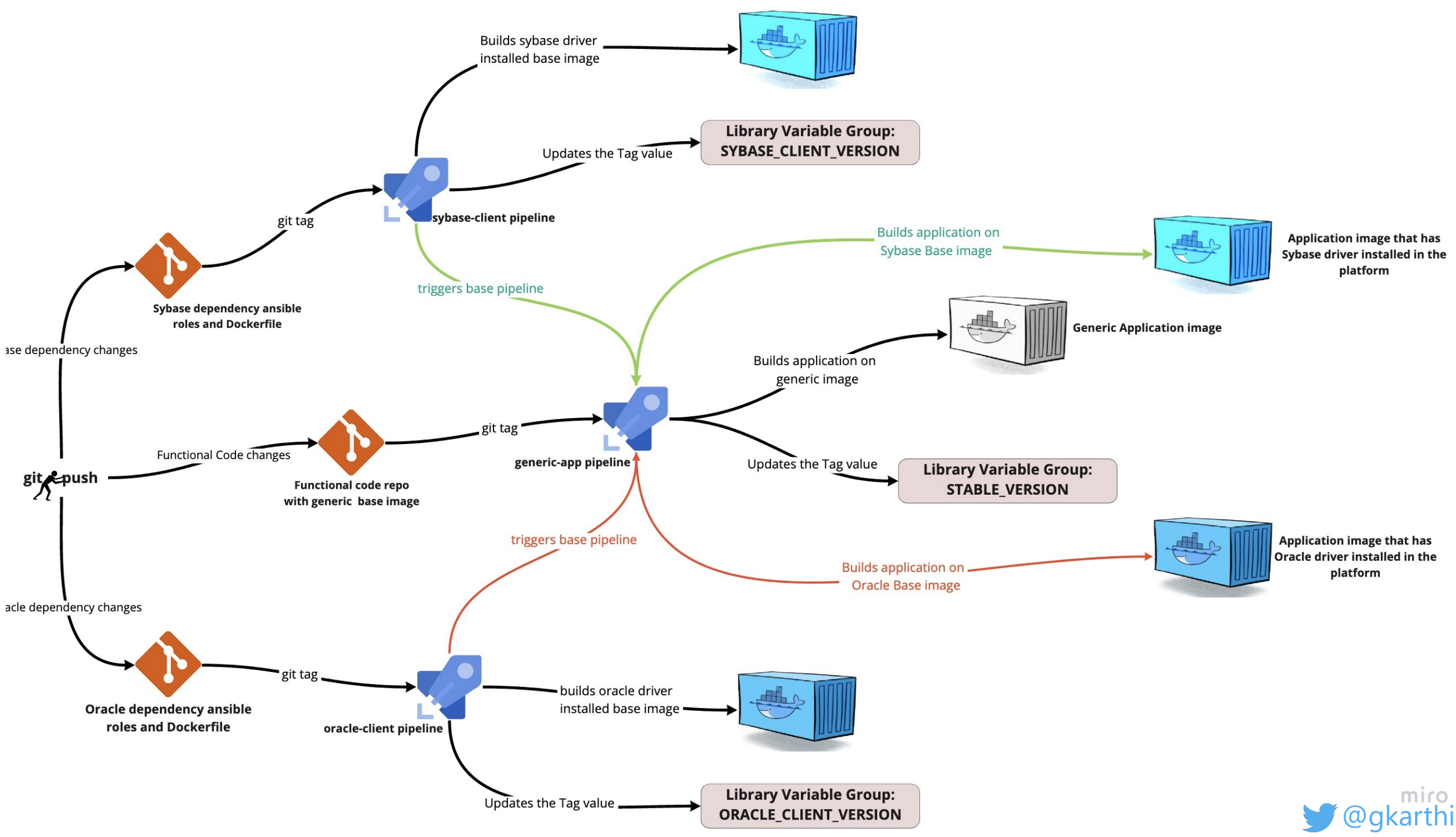


… contd.,

The discussed process is a one-way trigger from different base to build the image.

The vice-versa can also be set to build all the variants if there is a change in any of the functional code.

This will make flipping base images of the docker file more automated and preserve the single source of truth and smaller image size polices.



Thank You !!!

Medium article

Supported by the Russian MVP Community