DEVOPS PROCESS AND SCRIPTS

Contents:

1. Introduction
2. Tools used in Devops Ecosystem
3. DEVOPS BASE SET-UP INSTALLATION DIAGRAM
   1. DEVOPS REPOSITORY CONFIGURATION

3.2 DEVOPS INVENTORY CONFIGURATION

3.3 UPLOAD ARTIFACTS IN NEXUS REPOSITORY

3.4 DOWNLOAD ARTIFACTS FROM NEXUS REPOSITORY

3.5 DEVOPS RUNTIME VARIABLE

4. EXECUTION OF DEVOPS BASE-SET-UP PIPELINE

* 1. Stage1: PublishArtifacts

4.2 Stage2: input\_validation

4.3 Stage3: Application Installation

5. EXPLAIN ANSIBLE PLAY BOOK FLOW

5.1 INVENTORY FILE

5.2 APPLICATION YAML FILE

5.3 EXECUTING ROLES FOLDER

5.3.1 SHOW SERVER DETAILS

5.3.2 PRE-REQUEST SOFTWARE TO BE INSTALLED

5.3.3 DB UPDATE

5.3.4 INSTALL BASE APPLICATION

1. **Introduction:**

DevOps is an organizational software engineering discipline that aims at unifying software development (Dev). The main characteristic of DevOps is to improve the efficiency of the software development lifecycle by automating, monitoring, and applying security at all phases of the software lifecycle from initiation till operating and monitoring of a software project. Practicing DevOps will improve the quality and security of the deliverables over the traditional software lifecycle

.

**2. Tools used in Devops Ecosystem:**

This section will describe the different tools to be used to implement release management lifecycle for applications.

* 1. **Ansible**

Ansible is an open source IT Configuration Management, Deployment & Orchestration tool. It aims to provide large productivity gains to a wide variety of automation challenges. This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments. Ansible need to be installed in the target Servers.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy and most importantly. It uses a very simple language (YAML in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

* 1. **GitLab**

It is a GitHub like service that organizations can use to provide internal management of git repositories. It is a self-hosted Git-repository management system that keeps the user code private and can easily deploy the changes of the code.

GitLab is great way to manage git repositories on centralized server. GitLab gives you complete control over your repositories or projects and allows you to decide whether they are public or private for free.

* 1. **Azure Devops**

Azure DevOps is a language, platform and cloud provider agnostic DevOps toolchain and orchestration platform from Microsoft.

It also integrates with most leading tools on the market and is a great option for orchestrating a DevOps toolchain. At DevOpsGroup, we have lots of customers who have found Azure DevOps fits their needs irrespective of their language, platform or cloud.

* 1. **Jenkins**

Jenkins is a free and open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.

Jenkins supports the complete development life cycle of software from building, testing, documenting the software, deploying, and other stages of the software development life cycle.

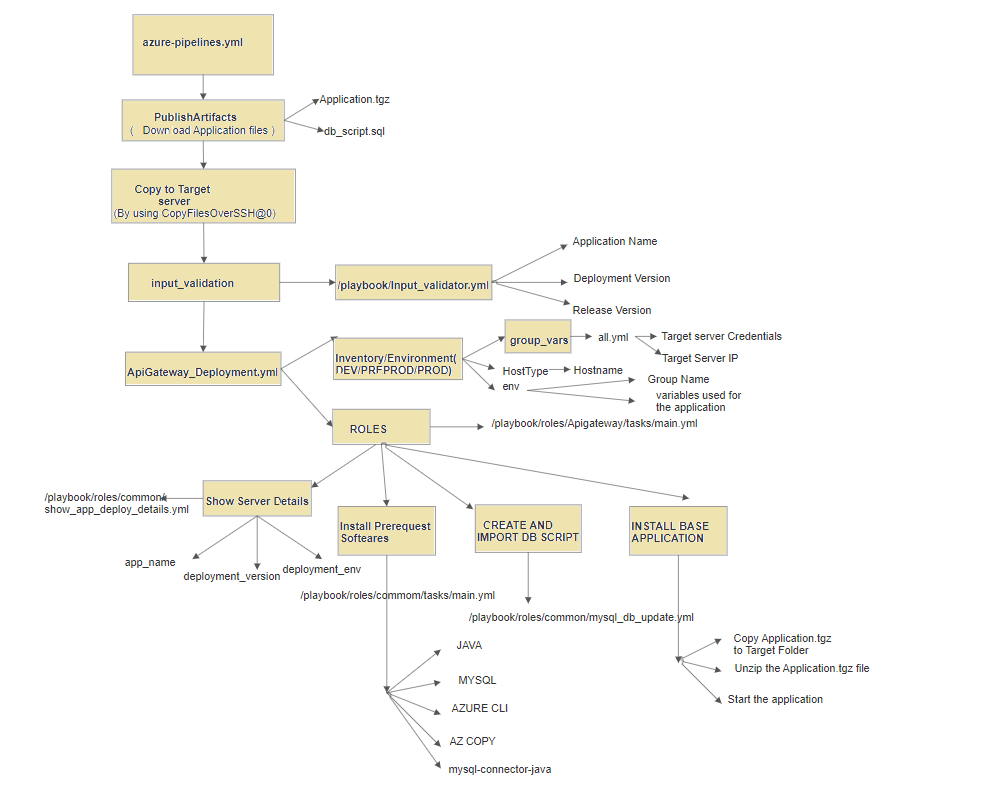
* 1. **Sonatype Nexus**

Nexus is a repository manager. It allows you to proxy, collect, and manage your dependencies so that you are not constantly juggling a collection of JARs. It makes it easy to distribute your software. Internally, you configure your build to publish artifacts to Nexus and they then become available to other developers. You get the benefits of having your own 'central', and there is no easier way to collaborate.

Now we are using Nexus community version to publish artifacts.

1. **DEVOPS BASE SET-UP INSTALLATION DIAGRAM:**

This flow diagram shows base set up installation of any application by using Devops pipeline.



* 1. **DEVOPS REPOSITORY CONFIGURATION:**
     1. **DEVOPS INVENTORY CONFIGURATION:**

All the ansible inventories are now environment specific. For example Configuration\_management -> inventory -> TRAINING will have all TRAINING environment spefic inventories and variables similarly for PREPROD & PROD. Each inventory has env, HostType & group\_vars -> all.yml

* env : This file will all you to define the environment specific and application specific inventories and variables and their values. We will define the reference of the IP's/Hosts defined in all.yml.
* HostType : This file will all you to define the mapping of the actual inventories defined in env.
* group\_vars -> all.yml : Actual IP's/Hosts will be defined here and the env will take reference of this.
  + 1. **UPLOAD ARTIFACTS IN NEXUS REPOSITORY**:

Uploading Arifactory application.tgz file and db\_script.sql file from Nexsus repository 🡪

To Upload the artifact, we can use the nexsus repository URL. Use the following nexsus artifactory URL. We can follow below methods to upload the artifacts.

* curl –v –u $user:$password –T <filename> http://49.206.240.154:9091/repository/ TASMU-Maven-1/Tasmu\_Base\_Setup\_Installation/${{Application\_name}}/${{ Application }}.tgz
  + 1. **DOWNLOAD ARTIFACTS FROM NEXUS REPOSITORY**:

We can download artifacts, Base-set-up\_Application.tgz,db\_scripts.sql files from Nexus repository. Execute below mentioned script in the target server to download the files.

* curl -X GET -u $user:$password -O http://49.206.240.154:9091/repository/TASMU-Maven-1/Tasmu\_Base\_Setup\_Installation/${{ Application\_name }}/db\_script.sql

* + 1. **DEVOPS RUNTIME VARIABLES**:

We can run Devops Pipeline run time Variables from Pipeline > Run pipelines > Define the required parameters.

We have to pass 3 variable according to our deployment application, environment and target server.

* DEPLOY\_APP : (Ex. NGA)
* DEPLOYMENT\_ENV : (Ex. DEV or PREPROD or PROD or TRAINING)
* RESOURCE\_NAME : ( Ex. vm6dinttrawe01)

\*\* Note: All the above variables are Devops Pipeline Variables.

It’s recommended not to change them. Before running the pipeline you need to define the values for these variable at pipeline execution level.

1. **EXECUTION OF DEVOPS BASE-SET-UP PIPELINE**

We have to pass 3 runtime variables to execute CD pipeline.

* + - DEPLOY\_APP
    - DEPLOYMENT\_ENV
    - RELEASE\_VERSION

Here we have explained CD pipeline flow of Azure devops.

Before running the pipeline you need to define the values for these variable at pipeline execution level.

Once done then go to CD -> Pipelines -> Run Pipeline -> Define the pipeline specific variables (if required) -> Run Pipeline. To view the pipeline log you can click on specific pipeline job like Stage.

* + - **Stage1**: **Publish Artifacts**
* It will copy the **Application.tgz** file in the corresponding application name (DEPLOY\_APP) passed in the run time and **db\_script.sql** ( db queries required to start the application) to the target server.
* By using CopyFilesOverSSH@0 method, it will copy the application and db related files to the target server.
* It will download all the files in “/home/6d.billing.a.l.svcact” location in the target server.
  + - * + **Stage2: input validation:**

All the application name and deployment environments should be defined in input\_vaidation.yml file in Tasmu\_Devops.

**Script**:



We run this stage to validate the 3 Run time parameters we have passed should be a part of our project.

If we are passing any mismatching Run time variable, which is not there in input\_validation.yml file, then our pipeline will fail.



* + - * + **Stage3: Application Installation ( ApiGateway\_installation )**

This stage will deploy application in the target location, Install all the necessary software need to be installed and execute all the given db\_scripts in the application server.

**script**: ansible-playbook -i /mnt/myagent/6d\_pipeline/Configuration-Management/inventory/${{ parameters.DEPLOYMENT\_ENV }} /mnt/myagent/6d\_pipeline/Configuration-Management/playbooks/ApiGateway\_Deployment.yml -e "env=deploy\_ApiGateway[0]" -e "deployment\_env=${{ parameters.DEPLOYMENT\_ENV }}" -e "deploy\_app=${{ parameters.DEPLOY\_APP }}"

1. **EXPLAIN ANSIBLE-PLAYBOOK FLOW**

After passing 3 runtime variables, It will execute the ansbile-playbook according to the application which we are going to deploy in the target server.

* **INVENTORY FILE**

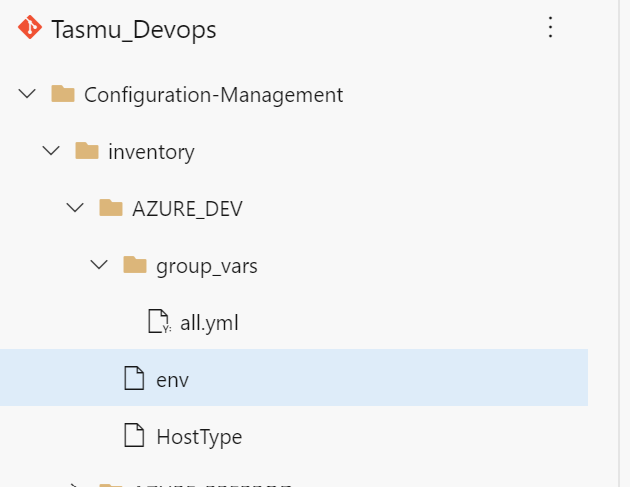
We have mentioned all target server details with variables in the inventory file of the repository.

/mnt/myagent/6d\_pipeline/Configuration-Management/inventory/${{ parameters.DEPLOYMENT\_ENV }} 🡪 If we are passing DEPLOYMNET\_ENV=AZURE\_DEV

All the target server information and variable information will be executed from **/mnt/myagent/6d\_pipeline/Configuration-Management/inventory/AZURE\_DEV** folder.

This folder contains 3 files, which contains necessary target server information.

1. Env : This file will all you to define the environment specific and application specific inventories and variables and their values. We will define the reference of the IP's/Hosts defined in all.yml.
2. HostType : This file will all you to define the mapping of the actual inventories defined in env.
3. group\_vars -> all.yml : Actual IP's/Hosts will be defined here and the env will take reference of this.



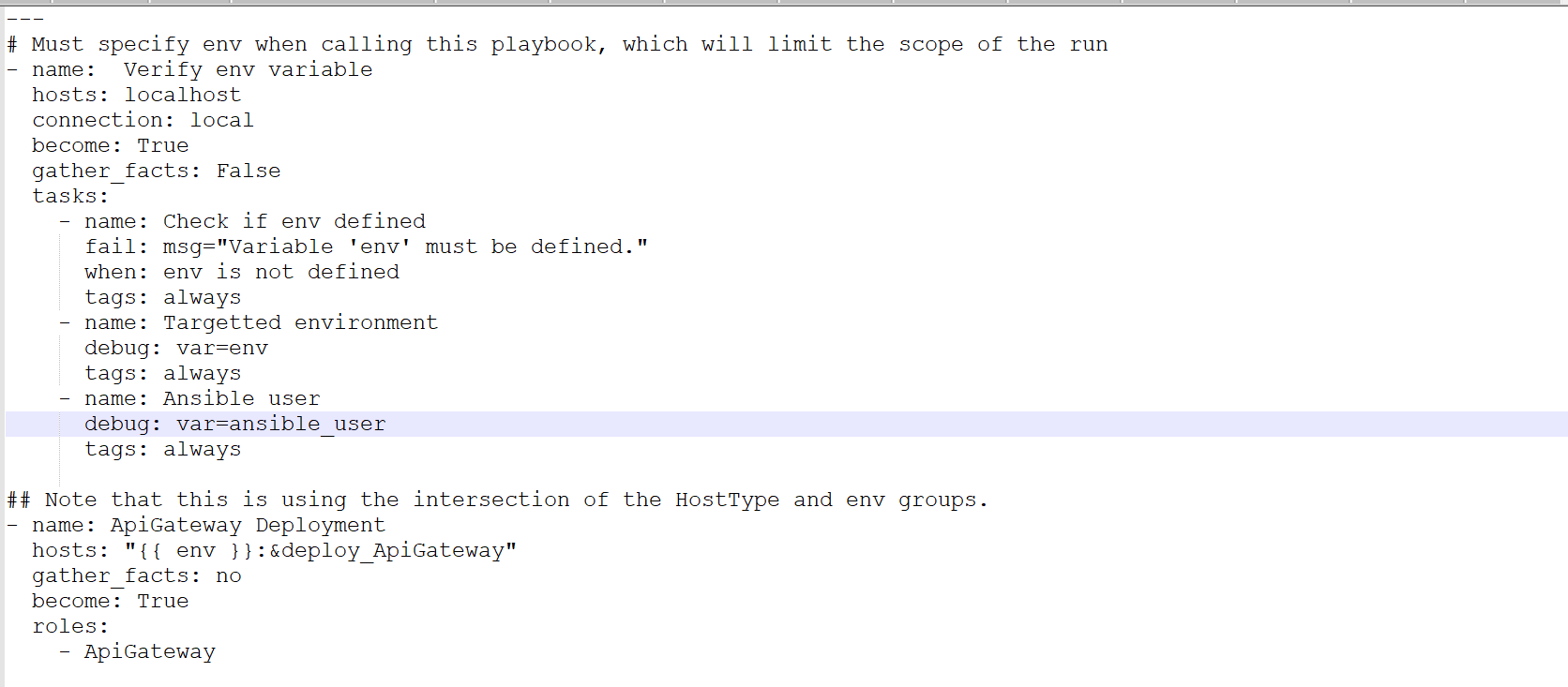
* **APPLICATION YAML FILE**

Each application we are going to run should contain an YML file, which will contain application target environment details with application group variables.

This will call a roles folder, which contains application related information.

**File name**:

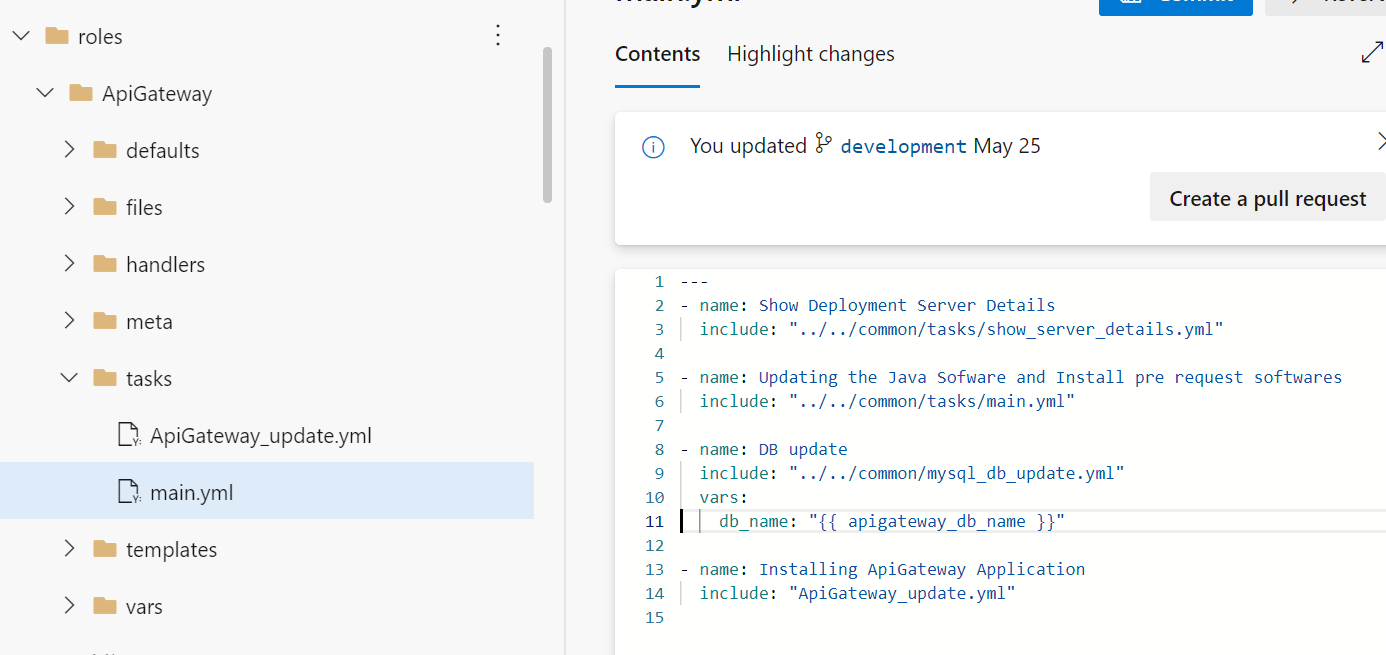
/mnt/myagent/6d\_pipeline/Configuration-Management/playbooks/ApiGateway\_Deployment.yml



* **EXECUTING ROLES FOLDER**

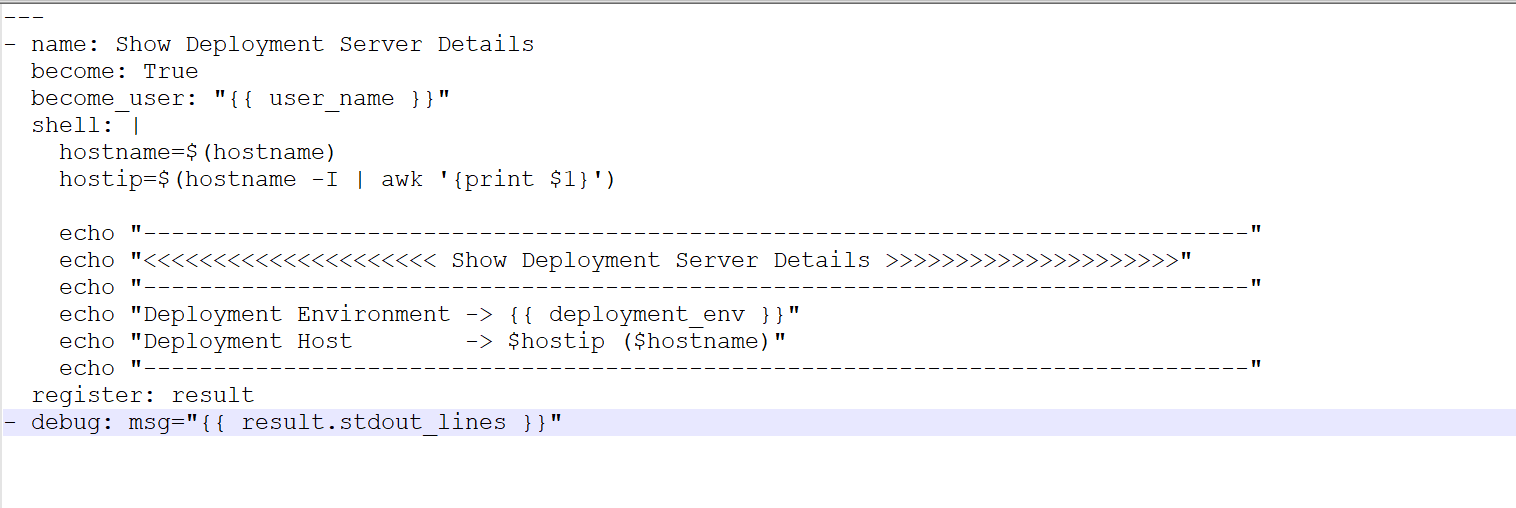
After executing application.yml file, we will get “Deployment environment “ and target server.

Then it will execute main.yml file in application “roles” folder (here roles 🡪 ApiGateway 🡪 tasks 🡪 main.yml ).



* + **SHOW SERVER DETAILS:**

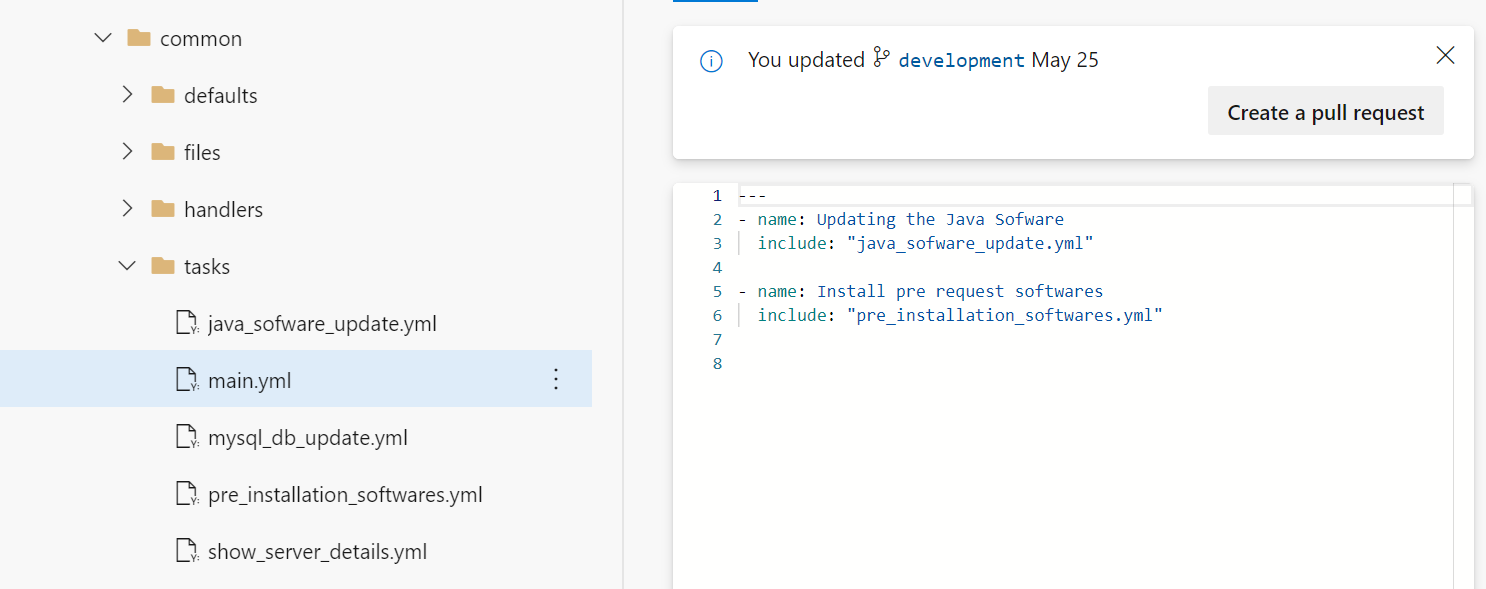
This yml file will show “Deployment Environment” and "Deployment Host” of the target server.



* + **PRE-REQUEST SOFTWARE TO BE INSTALLED** :

This will install java and all other pre-request software need to install before funning the pipeline.

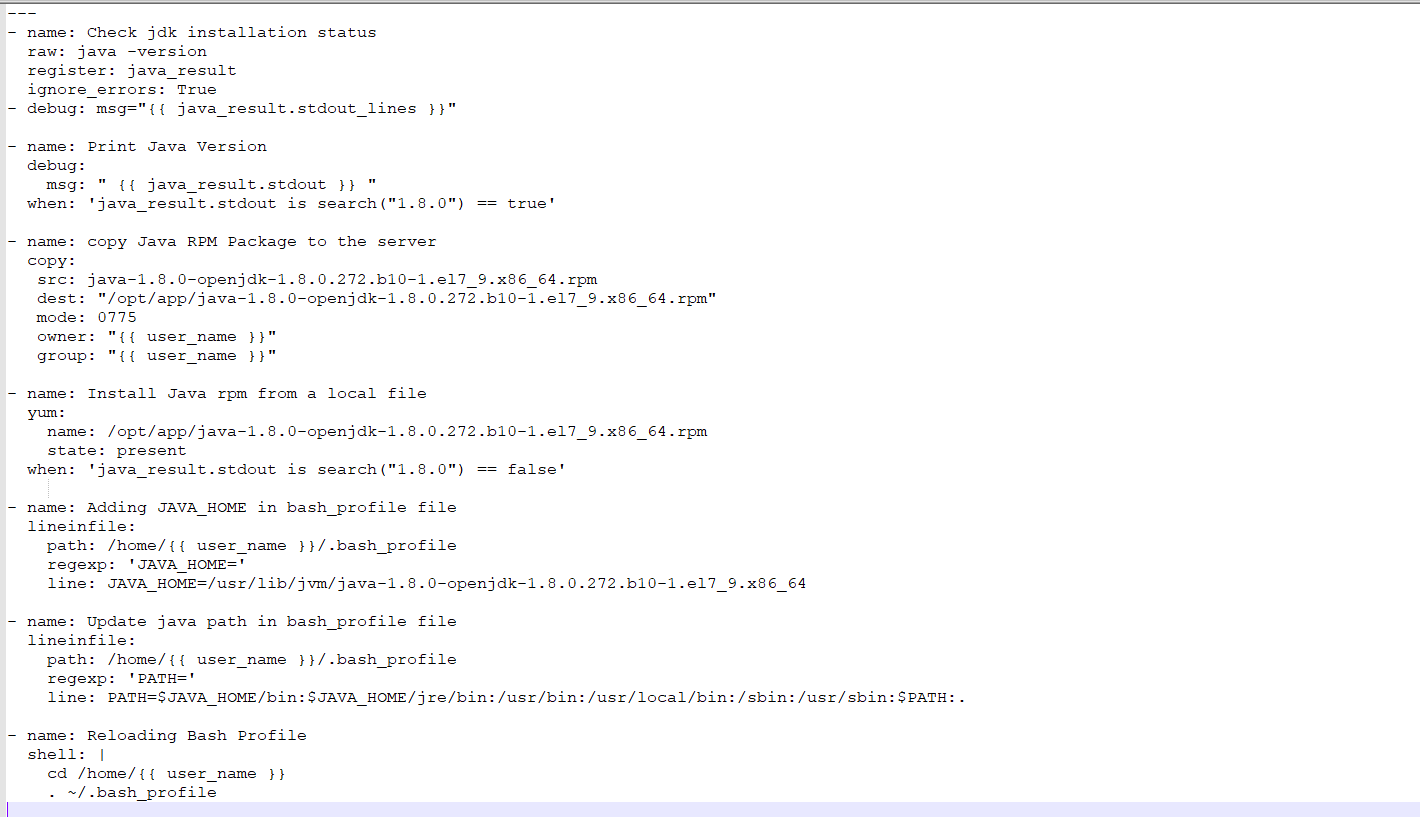
This will execute main.yml file in common folder.



* + **INSTALL JAVA IN TARGET SERVER**

This yml file will install java in the target server, if java is not there.

This will skip the installation if java is already exists.



* + **INSTALL OTHER PRE-REQUEST SOFTWARE**

This will install all the necessary software to be installed before starting the application.

Pre-request software list:

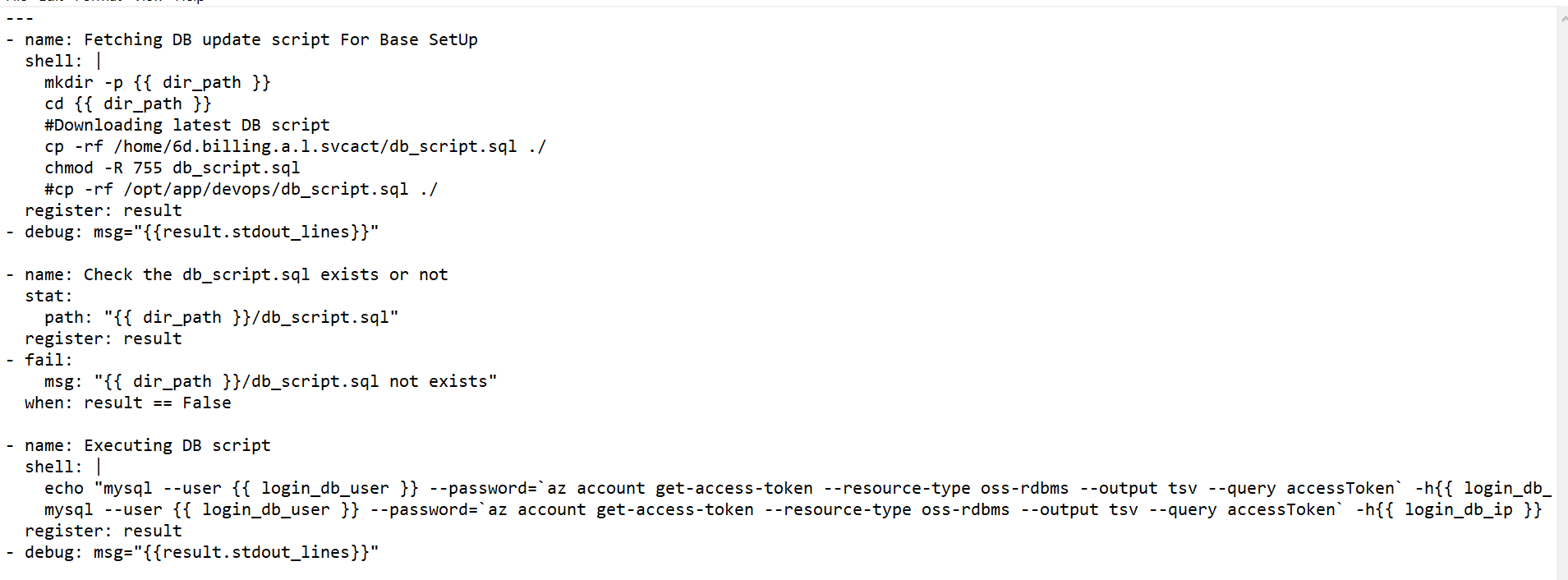
1. JAVA
2. MYSQL CONNECTOR & mysql-connector-java for JDBC
3. AZURE CLI
4. AZURE COPY
5. AGENT\_LOG\_MODULE

Commands to install the software:

1. Yum install java-1.8.0-openjdk-1.8.0.272.b10-1.el7\_9.x86\_64.rpm
2. yum install -y azure-cli
3. yum install -y mysql mysql-connector
4. sudo yum install -y mysql-connector-java
   * **DB UPDATE**

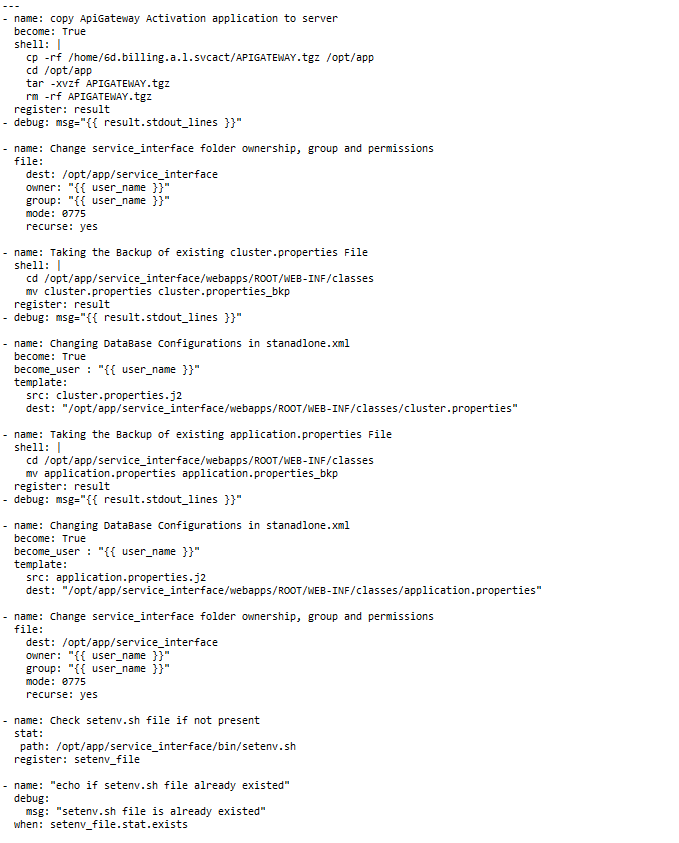
This yml file will execute db\_script.sql file in the application DB server mentioned in the inventory file.

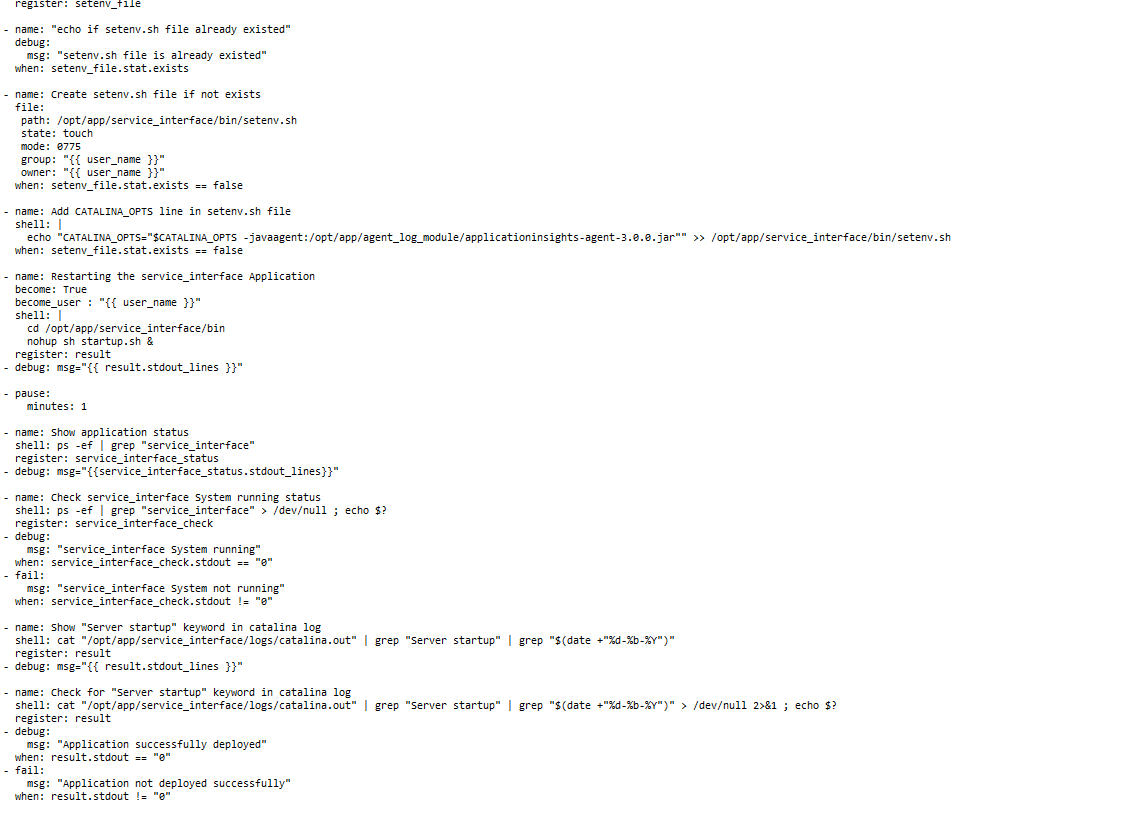
Here we are using mysql DB and we will execute mysql command for DB import.



* + **INSTALL BASE APPLICATION**

This yml file contains the procedure to unzip the copied Application.zip file inside the target server and start the application procedure.







APPLICATION UPGRADE PIPELINE

CD PIPELINE USING GITLAB



Please follow the below steps to set up application upgrade by using DEVOPS eco-system.

**Contents: -**

**1. Block Diagram of application Upgrade**

**2. Stages**

**2.1. input\_validation**

**2.2. download\_artifacts**

**2.3. pre\_check**

**2.4 app\_deploy**

**2.5. db\_deploy**

**2.6 app\_restart**

**2.7 post\_check**

**3. Repository View**

**4. CI/CD Pipeline Execution**

1. **Block Diagram of application Upgrade:**

This Diagram explains application upgrade of any application by using DEVOPS.

**DEPLOYMENT\_ENV**

**.gitlab-ci.yaml**

**ApiGateway\_Deployment.yaml**

**Target server Credentials details**

**/Inventory/Environment (DEV/PREPROD/PROD)**

**\*Check BKP directory already existed**

**\*Take Backup of existing deployed file**

**\*deploy latest war file**

**\*Take Backup of DB**

**\*Execute Deployment DB queries**

**group\_vars**

**Variables used for the application**

**playbooks/roles/common/pre\_check.yml**

\***playbooks/roles/common/check\_mysql\_status.yml**

**\*playbooks/roles/common/ check\_jdk\_status.yml**

**playbooks/roles/common/artifactory\_download.yml ( Dowload war file)**

**Ansible/playbooks/roles/common/post\_check.yml**

**DEPLOY\_APP**

**/playbook/Input\_validation.yaml**

**Input\_validation**

**RELEASE\_VERSION**

**all.yml**

**Target ip**

**env**

**group name**

**HostType**

**Hostname**

**Playbooks/roles/Apigateway/tasks/main.yaml**

**Roles**

**\*RAM,DISK SPACE**

**\* APPLICATION RUNNING STATUS**

**Pre\_checks**

**Related software are available (JAVA,MySql,NodeJs)**

**playbooks/roles/common/** **create\_base\_directory.yml**

**app\_deploy**

**playbooks/roles/common/mysql\_db\_backup.yml**

**db\_deploy**

**\*Restarting the application**

**app\_restart**

**\*Show deployment status**

**\*Show application running status**

**\* Disk space should be more than 90%**

**\* RAM should be more than 1GB**

**post\_check**

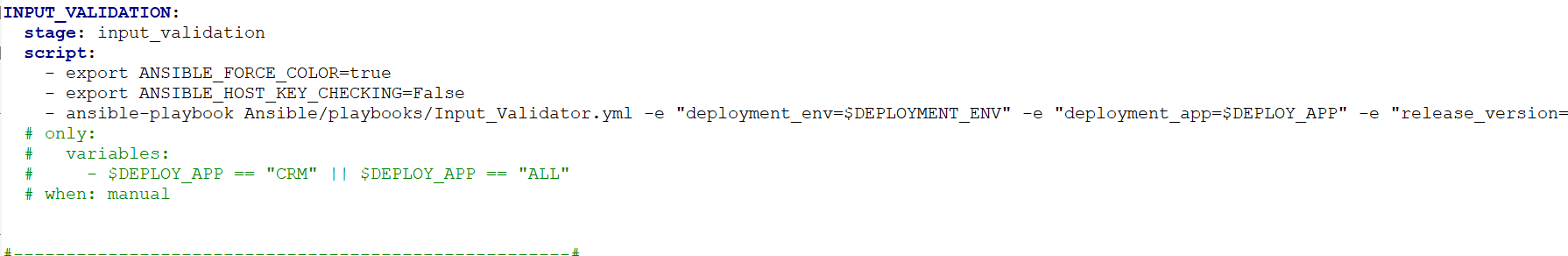
1. **STAGES:**

We have explained all the stages for application upgrade pipeline.

**2.1 Input\_validation: -**

All the application name and deployment environments should be defined in input\_vaidation.yml file.

**Script**:



We run this stage to validate the 3 Run time parameters we have passed should be a part of our project.

If we are passing any mismatching Run time variable, which is not there in input\_validation.yml file, then our pipeline will fail.

* 1. **download\_artifacts:**
* All this deployment stage, First we are downloading artifacts and

db\_script.sql file from Nexus repository.

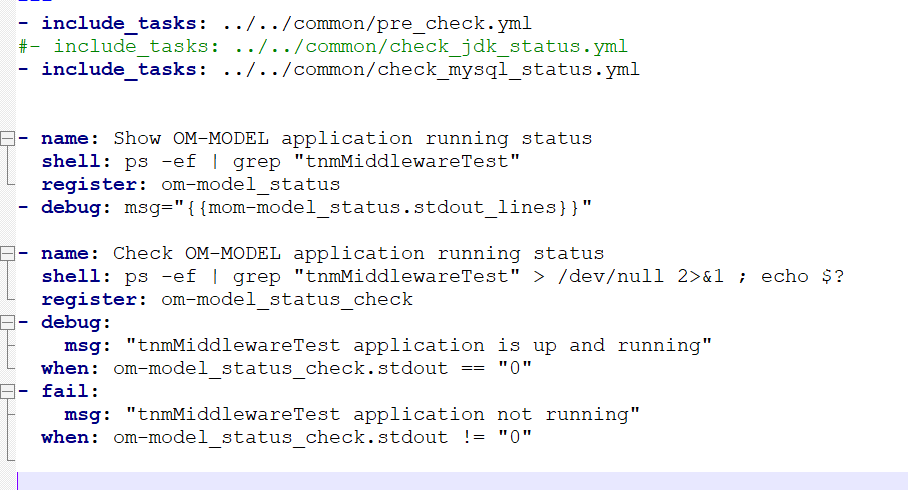
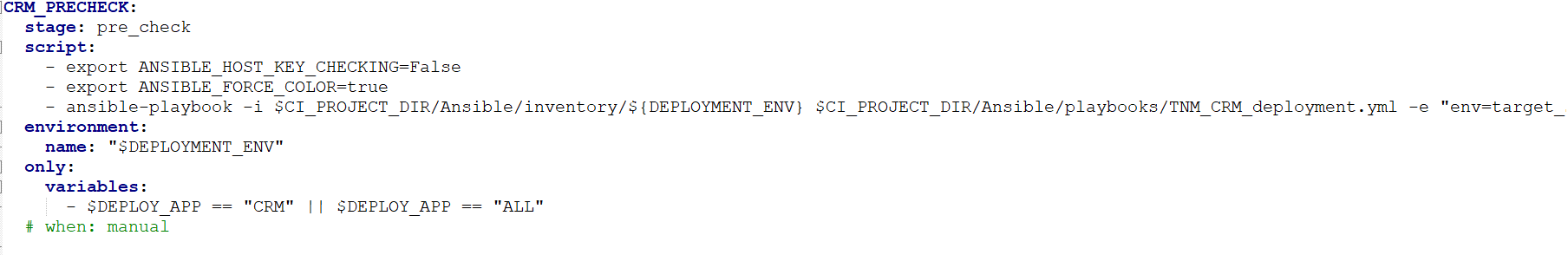
* Then we are copying the artifacts and db\_script.sql file to the target server by using SSH copy method.



* 1. **PRE CHECK:**

We are doing necessary pre-checks in the target server before deploying any patch upgrade.

* Target server should have RAM more than 1GB.
* Disk space utilization shouldn’t be greater than 90%
* Application should be up and running condition.

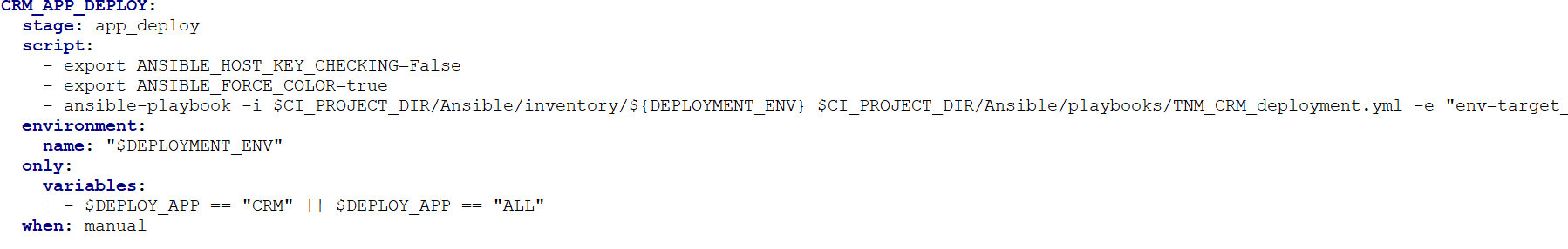


* 1. **APP DEPLOY:**

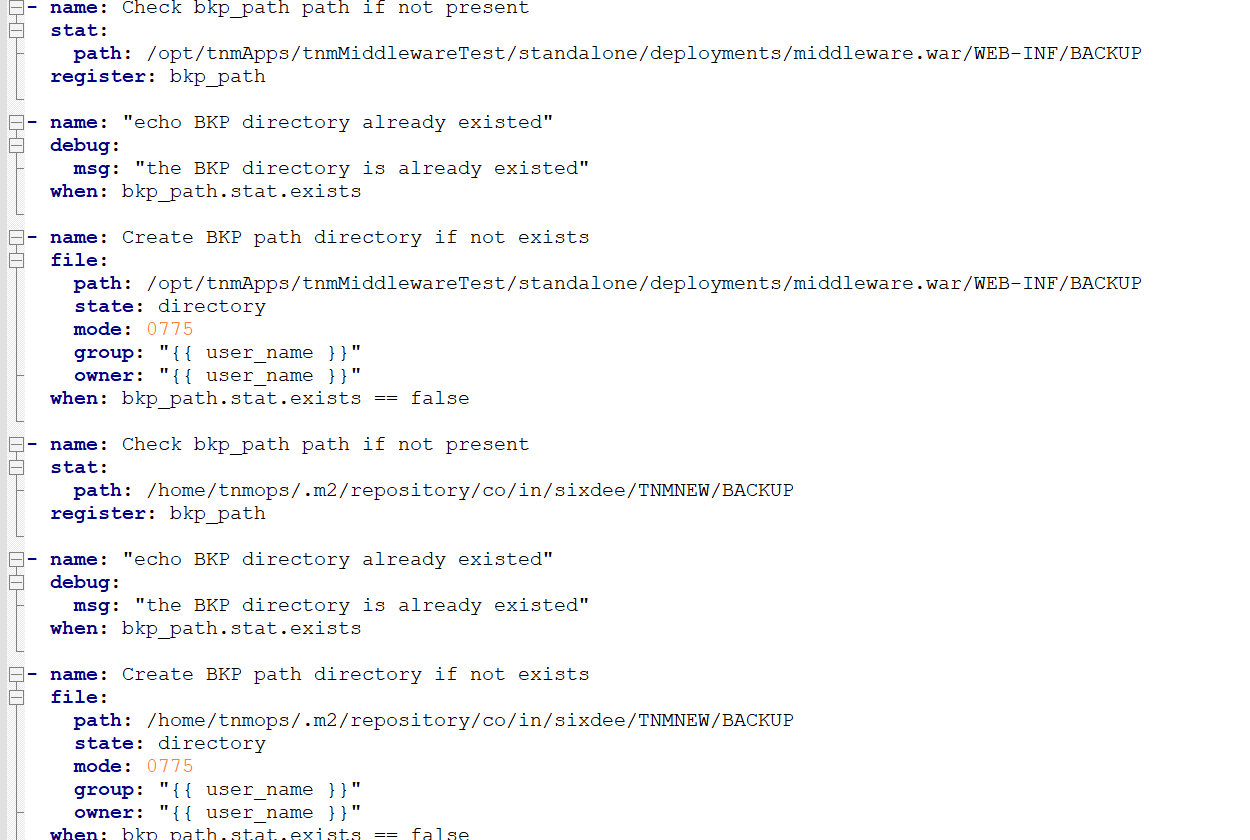
This stage will first kill the application by application name.

Then it will take back up of existing .war or jar file. And it will deploy

Latest war or jar file in the target server.

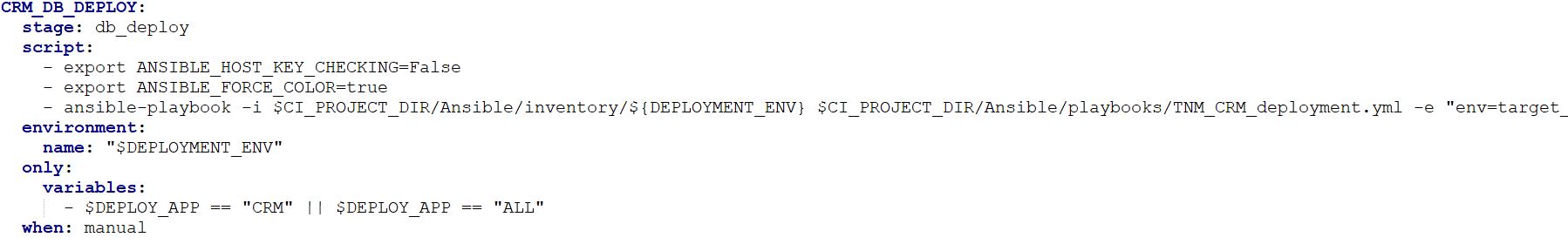


* Check bkp\_path path if not present
* Check BKP directory already existed
* Create BKP path directory if not exists
* Search war file in mapping.txt file
* move old deployed packages create\_base\_directory.yml
* Create directory if not exists kill\_process.yml
* Kill Process ps -aef | grep -w {{ process\_name }} | grep -v grep 2>&1 > /dev/null
* Check Process Status artifactory\_download.yml
* Download artifacts from Nexus Artifactory & deploying to target node
* Removing up old mapping file
* Download latest mapping file from Nexus Repository.
* Read the mapping file and transfer packages to the corresponding locations mentioned in mapping.txt file.
* Check Application packages updated successfully

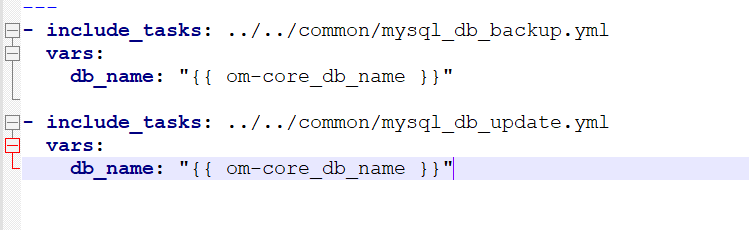


* 1. **DB DEPLOY:**

This will check mysql status. Then it will take backup of structure of existing DB and execute db\_script.sql file in the target DB server.

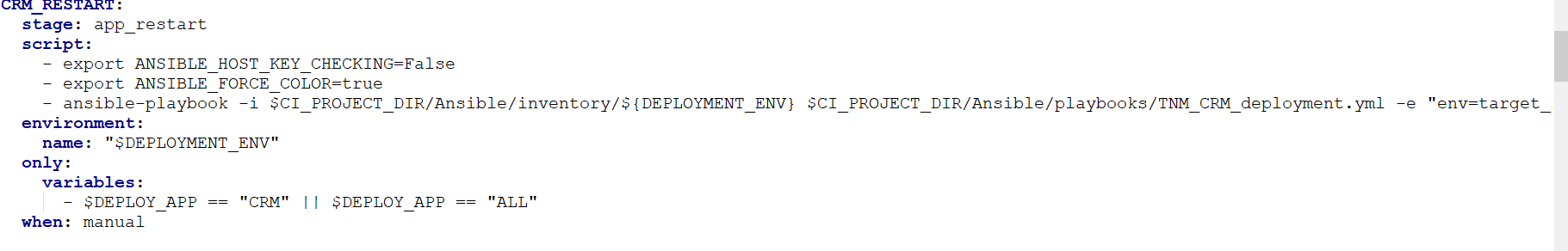


* Checking my\_sql database
* To check the MySQL status remote mysql\_db\_backup.yaml
* Create BKP directory if not exists
* Taking DB backup
* Removing Old DB backup

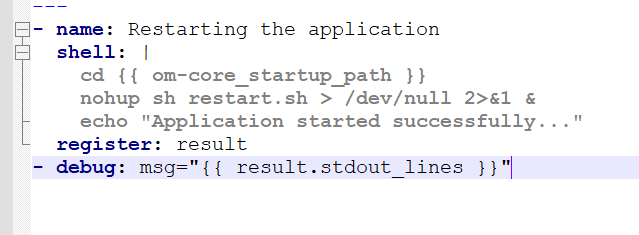


* 1. **APP RESTART**

This will go to the application restart location and run the restart application command to restart the stopped application.



* Restarting the application

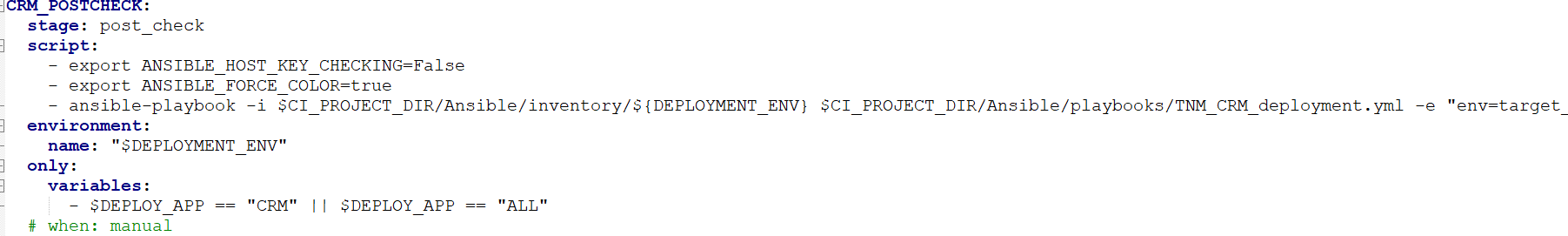


* 1. **POST CHECK**

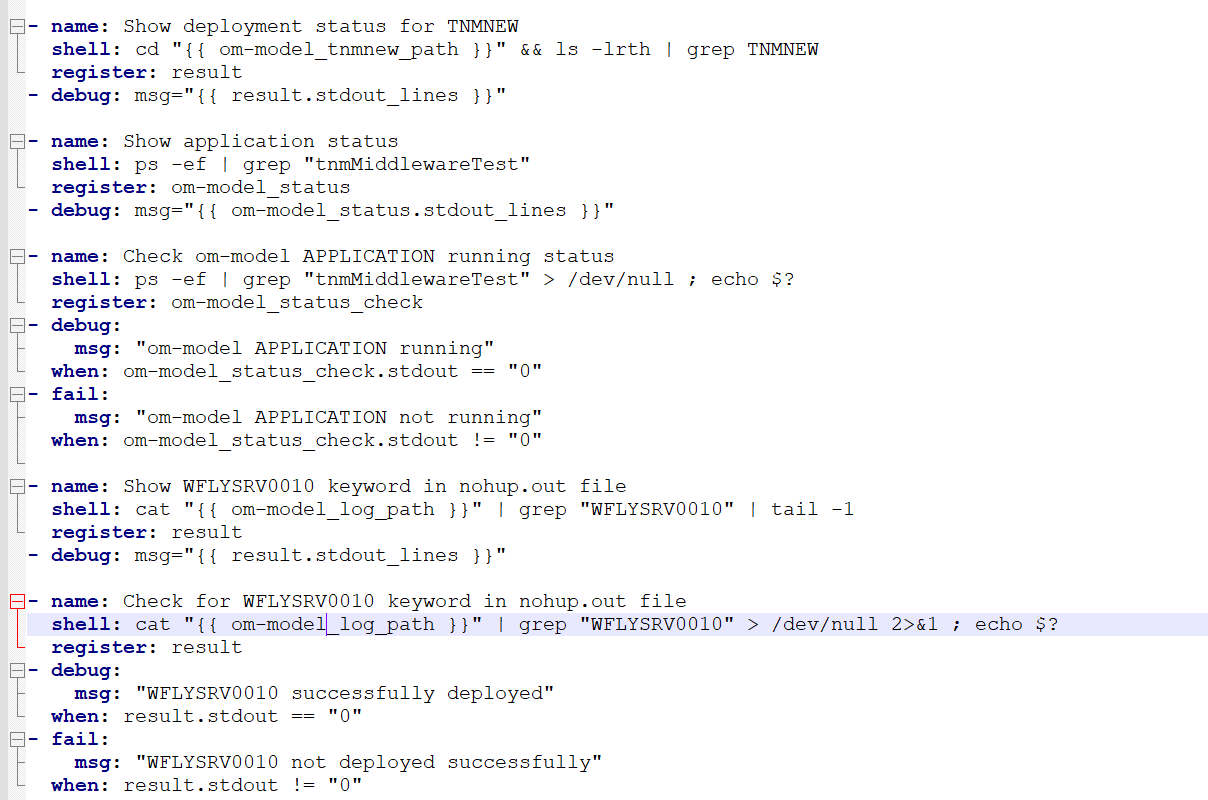
This will do validation of application post deployment.

We are checking it by application process name and also we have added some extra

Checks to verify the application running status.



* Show deployment status
* Show application status
* Check TNM-CRM APPLICATION running status post\_check.yml
* POST\_CHECK RAM utilization
* If available RAM is less than 1GB then stop pipeline
* POST\_CHECK disk space utilization
* If disk space utilization is more than 75% then stop pipeline



### CI/CD Pipeline Execution:

### After running the pipeline, we can check the application deployment status from GitLab UI logs.

### 

You can see the complete log status if you click the 'failed' or 'passed' status icons.

### 