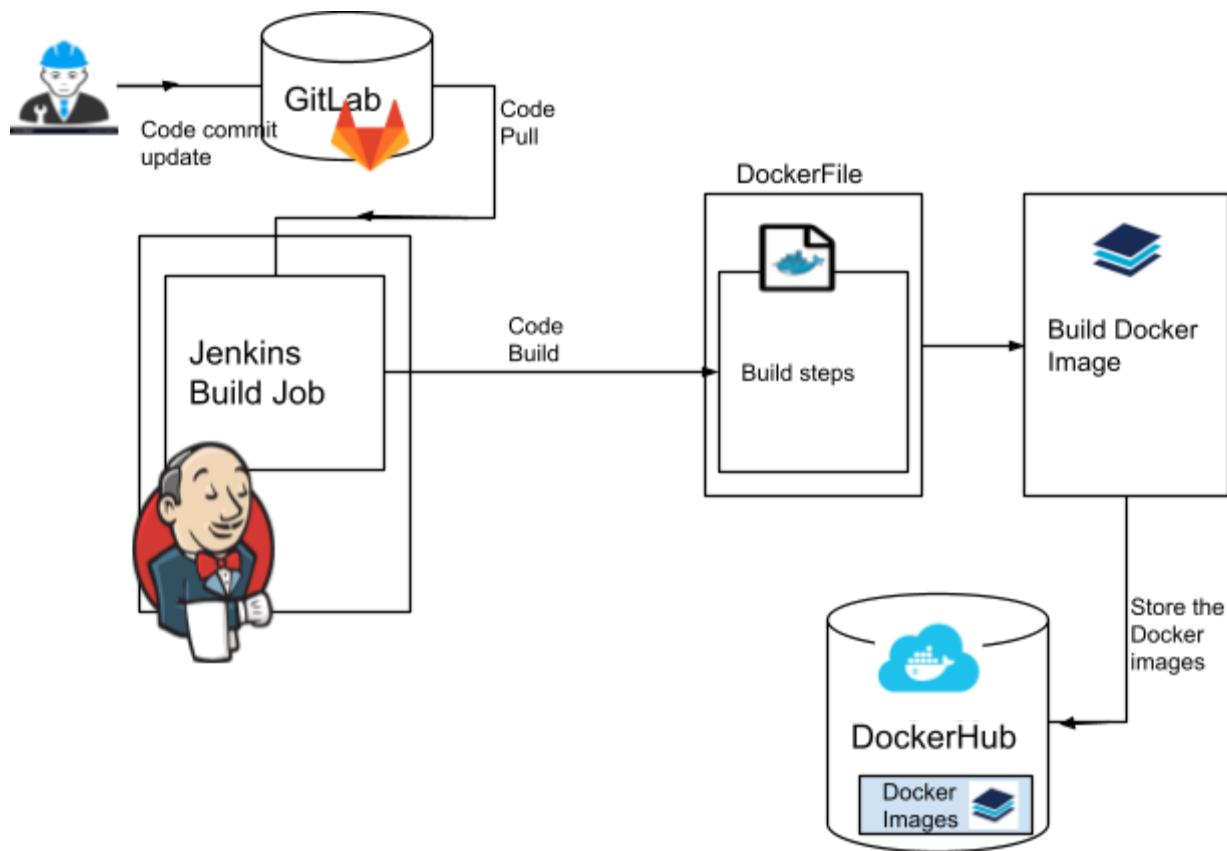


Build and deployment Jobs Configuration

Application Build Process



The developer makes a request to do build/deploy after pushing the latest changes to the GitLab.

Developers can select respective branches to build the code.

As per the developer request, we need to do build and deployment to reflect the changes in application. We have two separate jobs for application build and Deployment.

Build job configuration:

Docker FT environment

All	W	Name ↓	Last Success	Last Failure	Last Duration
		build	N/A	N/A	N/A
		deploy	N/A	N/A	N/A

Go to build job and choose an environment. And click on any servers in the pre-prod environment for instance.

Jenkins > DOCKER > build > pre-prod > kal-agent >

Up
 Status
 Changes
 Workspace
 Build with Parameters
 Delete Project
 Configure
 Rebuild Last
 Authorization
 Move
 Rename

Project kal-agent

Full project name: DOCKER/build/pre-prod/kal-agent

[Workspace](#)
 [Recent Changes](#)

Permalinks

[Build History](#) [trend](#) =
find
[RSS for all](#) [RSS for failures](#)

Click on configure tab to see how the build job is configured.

The screenshot shows the Jenkins General configuration tab. At the top, there are several checkboxes: "Disable Rebuilding for this job" (unchecked), "This build requires lockable resources" (unchecked), and "This project is parameterised" (checked). Below this, a "String Parameter" section is displayed, containing a "Name" field set to "image_tag", a "Default Value" field (empty), and a "Description" field with the text "Enter The image_tag to build". There are also "Plain text" and "Preview" buttons, and a "Trim the string" checkbox (unchecked). A "Add Parameter" button is located at the bottom left of this section.

Under the General tab section, we can see the job is parameterized. By this we are asked to give Image_tag at the time of build.

The screenshot shows the Jenkins Source Code Management configuration tab, specifically for the Git provider. Under the "Repositories" section, a single repository is defined with a URL of "https://gitlab.com/kaleidofin/kaleidofin-agent-ui.git" and credentials selected from a dropdown menu. Buttons for "Advanced...", "Add Repository", and "Add Branch" are visible. In the "Branches to build" section, a branch specifier "*/uat" is entered. The "Repository browser" is set to "(Auto)". Under "Additional Behaviours", there are options for "Multiple SCMs" and "Subversion", with "Multiple SCMs" currently selected.

Three things to apply here in SCM(Git):

1. code repo for this service
2. Credentials to pull the code from GitLab
3. pull the code with branch specific

Under the Build environment tab, we will choose to delete the workspace before the build starts and set the build name to be displayed as the **image_tag** given. This can be seen in build history status after the build job is completed.

The screenshot shows the Jenkins configuration interface for a build job. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment (which is selected and highlighted in black), Build, and Post-build Actions. The main content area is titled "Build Environment". Under this title, there is a checked checkbox labeled "Delete workspace before build starts". Below this is a "Advanced..." button. The main configuration area contains a list of checkboxes with their descriptions and help icons:

- Use secret text(s) or file(s) ?
- Provide Configuration files ?
- Send files or execute commands over SSH before the build starts ?
- Send files or execute commands over SSH after the build runs ?
- Abort the build if it's stuck
- Add timestamps to the Console Output
- Ant/Ivy-Artifactory Integration
- Generic-Artifactory Integration
- Gradle-Artifactory Integration
- Inject environment variables to the build process ?
- Inject passwords to the build as environment variables
- Inspect build log for published Gradle build scans
- Maven3-Artifactory Integration
- Prepare SonarQube Scanner environment ?
- Provide Node & npm bin/ folder to PATH
- Set Build Name ?

Below the checkboxes, there is a "Build Name" field containing the placeholder "\${image_tag}" and an "Advanced..." button at the bottom right.

Under Build section, we will give shell commands to generate the artifact and build docker image, push the images to Dockerhub

The screenshot shows the Jenkins build configuration interface. It features two 'Execute shell' sections under the 'Build' tab. The first section contains the command: `cd kaleidofin-agent-ui ; npm install; npm rebuild node-sass; yarn run build ;`. The second section contains the command: `cd kaleidofin-agent-ui/
echo " Create a docker image"
docker build -t kaleidofin/kfin:${image_tag} .
echo " Push the docker images to the Hub"
docker push kaleidofin/kfin:${image_tag}`. Both sections have 'Advanced...' buttons. At the bottom, there are 'Save' and 'Apply' buttons.

```
cd kaleidofin-agent-ui ; npm install; npm rebuild node-sass; yarn run build ;
```

```
cd kaleidofin-agent-ui/  
echo " Create a docker image"  
docker build -t kaleidofin/kfin:${image_tag} .  
echo " Push the docker images to the Hub"  
docker push kaleidofin/kfin:${image_tag}
```

See [the list of available environment variables](#)

Advanced...

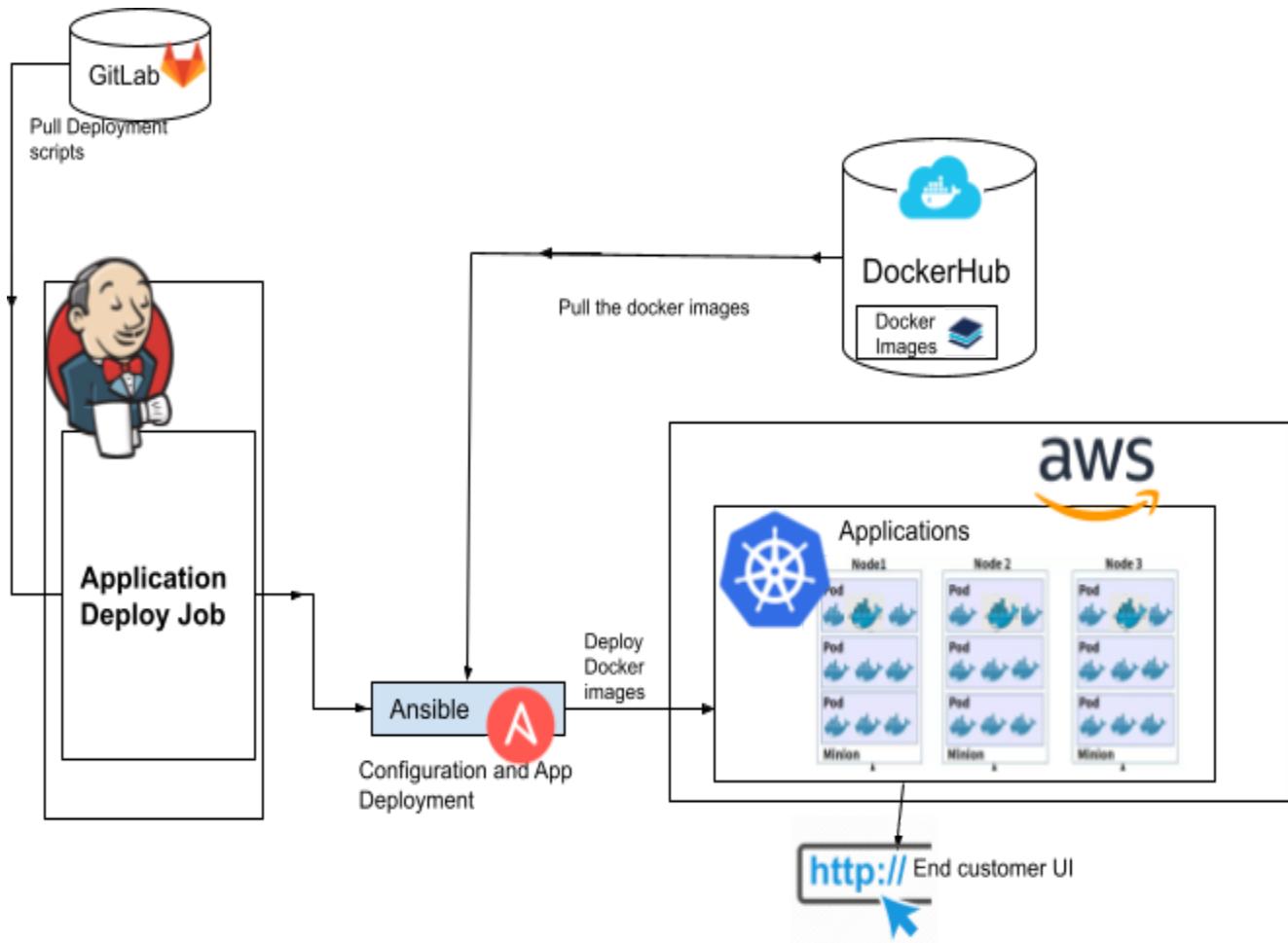
Save Apply Advanced...

The commands in the first section, used to install any dependencies and generate an artifact which can be used to build an image.

The commands in the second section, creates a docker image using the artifact generated. Then the image built is pushed to docker hub. The dockerhub logins are configured with jenkins.

This is how the build job is configured

Deployment Process



Jenkins and Ansible are running on the same machine (EC2). All the application deployments are deployed into EKS k8s cluster which is managed by AWS.

Deploy Job Configuration:

Now go to the Deploy job and choose any environment. Click on any server to deploy the job. For demo, we are doing the deployment in a kal-agent server.

-  Up
-  Status
-  Changes
-  Delete Pipeline
-  Configure
-  Authorization
-  Move
-  Full Stage View
-  Rename
-  Pipeline Syntax

 Build History [trend](#) 

X

 [RSS for all](#)  [RSS for failures](#)

Pipeline kal-agent

Full project name: DOCKER/deploy/dev/kal-agent



[Recent Changes](#)

Stage View

No data available. This Pipeline has not yet run.

Permalinks

Click on the configure button to see how the deploy job is configured.

General Build Triggers Advanced Project Options Pipeline

Pipeline speed/durability override
 Preserve stashes from completed builds

Rebuild options: Rebuild Without Asking For Parameters
 Disable Rebuilding for this job

This project is parameterised

String Parameter

Name	image_tag
Default Value	
Description	Enter image_tag

[Plain text] [Preview](#)

Trim the string

Choice Parameter

Name	Environment
Choices	ft1 ft2 development uat
Description	Enter the Environment

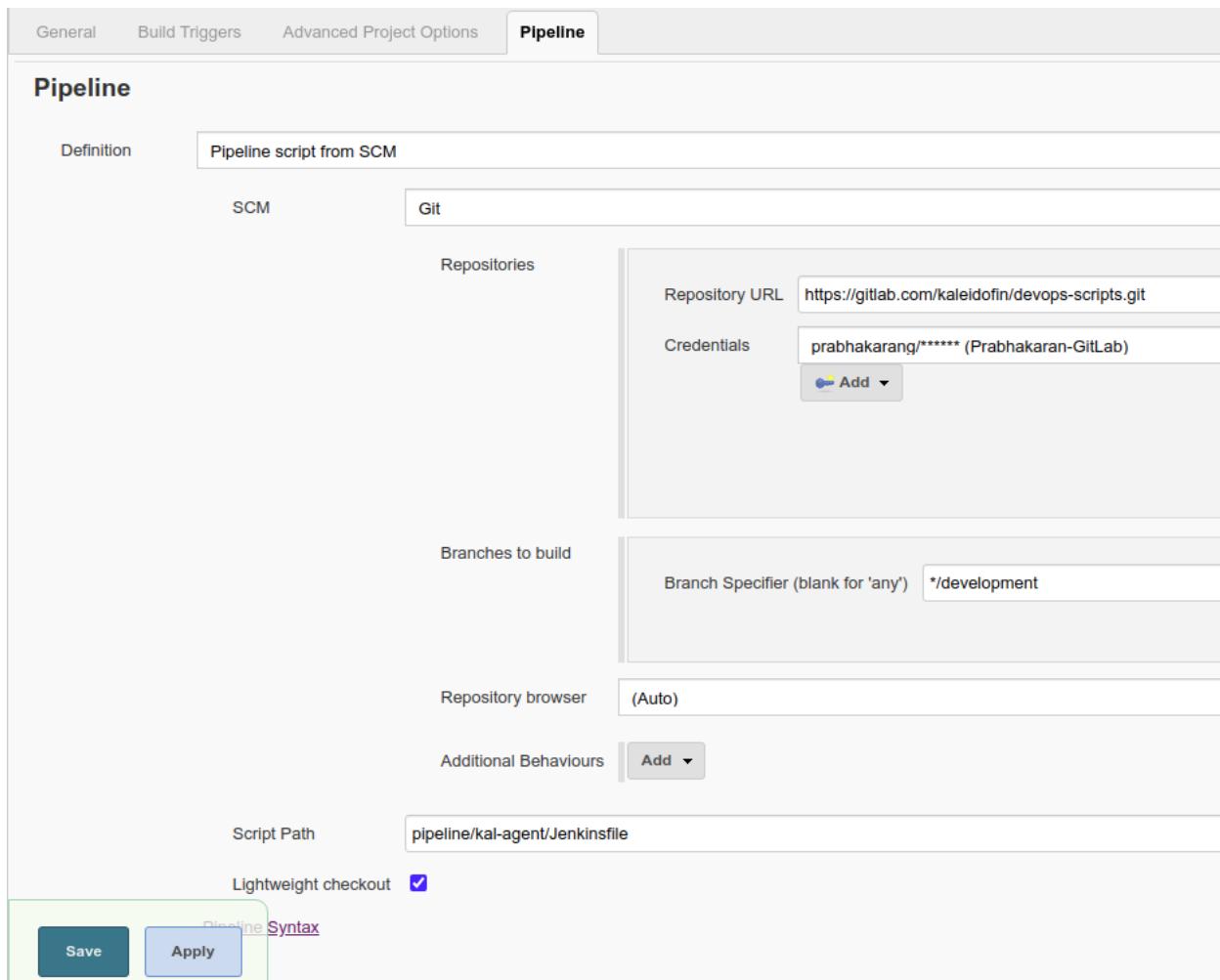
[Plain text] [Preview](#)

Choice Parameter

Name	business_group
Choices	kaleidofin-web kal-ft
Description	Enter the business_group

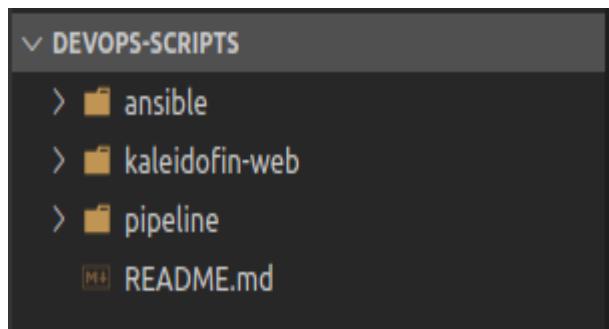
Under the General tab, we have two parameters defined. The string parameter is to give a name to image_tag and choice parameter is to choose the environments.

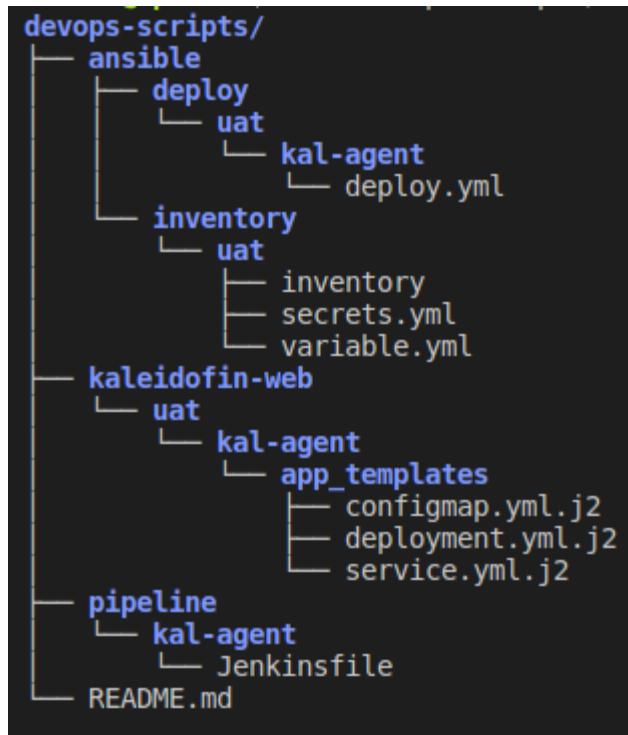
Under pipeline section, we have a jenkins pipeline script,



The screenshot shows the Jenkins Pipeline configuration page. At the top, there are tabs: General, Build Triggers, Advanced Project Options, and Pipeline. The Pipeline tab is selected. Below the tabs, the title "Pipeline" is displayed. Under the "Definition" section, it says "Pipeline script from SCM". The "SCM" dropdown is set to "Git". In the "Repositories" section, the "Repository URL" is set to <https://gitlab.com/kaleidofin/devops-scripts.git> and the "Credentials" are set to "prabhakarang/***** (Prabhakaran-GitLab)". The "Branches to build" section shows "Branch Specifier (blank for 'any')" as */development. The "Repository browser" is set to "(Auto)". Under "Additional Behaviours", there is an "Add" button. The "Script Path" is set to "pipeline/kal-agent/Jenkinsfile". The "Lightweight checkout" checkbox is checked. At the bottom, there are "Save" and "Apply" buttons, and a "Pipeline Syntax" link.

All the yaml files are found in the development branch of <https://gitlab.com/kaleidofin/devops-scripts.git>





```

pipeline {

agent any

parameters {
    string(name: 'image_tag' , defaultValue: "", description: 'Enter image_tag')
    choice(name: 'Environment', choices: ['ft1','ft2','development','uat'], description: 'Enter the Environment')
    choice(name: 'business_group', choices: ['kaleidofin-web','kal-ft'], description: 'Enter the business_group')
}

stages {

stage('Deployment to $env k8s cluster') {

steps [
    buildName "${params.image_tag}"
    sh "ansible-playbook -i ansible/inventory/${params.Environment}/
    ansible/deploy/${params.Environment}/${env.JOB_NAME}/deploy.yml
    --vault-password-file /var/lib/jenkins/secrets/vault-pass
    -e extra-vars 'env=${params.Environment} business_group=${params.business_group} image_tag=${params.image_tag} ws=$WORKSPACE'
]
}
}
}
}

```

The above snippet is the jenkins pipeline script.

This pipeline is the Entrypoint to k8s deployment. So, here in this stage ansible playbook calls the deploy.yaml[1] by passing extra parameters[2] at runtime with variables[3]

1. Calls the kubectl command to run the deployment files
2. extra params like: env, business_groups, image_tag and ws
3. inventory file where ansible runs with credentials.

The inventory file is located at path: ansible/inventory/uat/inventory

```
ansible > inventory > development > □ inventory
1 [local]
2 localhost ansible_host=localhost ansible_ssh_user=deployer ansible_ssh_pass=devops136@@
3
4 [masters]
5 localhost
6
7 [all:vars]
8 #ansible_ssh_user=optit
9 ansible_python_interpreter=/usr/bin/python3
10 ansible_host_key_checking=False
11
12 #[defaults]
13 #host_key_checking = False
```

The deploy.yaml file is located at ansible/deploy/uat/kal-agent/deploy.yaml

```
ansible > deploy > development > kal-agent > deploy.yaml > ...
1  ---
2  - hosts: masters
3    become_user: deployer
4
5
6  vars:
7    env: ft1
8    image_tag: dev-1.1.2
9    application: kal-agent
10   business_group: kaleidofini-web
11
12  vars_files:
13    - "{{ inventory_dir }}/secrets.yml"
14    - "{{ inventory_dir }}/variable.yml"
15
```

```
16  tasks:
17
18  - name: Create a directory
19    command: "mkdir -p ~/kube-deploy/{{ business_group }}/{{ env }}/{{ application }}"
20
21  - name: Copy the deployment file
22    copy:
23      src: "{{ ws }}/{{ business_group }}/{{ env }}/{{ application }}/"
24      dest: "~/kube-deploy/{{ business_group }}/{{ env }}/{{ application }}/"
25      force: yes
26
27  - name: "Copy the latest {{ application }} image {{ image_tag }} deploy"
28    template: src={{ item }} dest="~/kube-deploy/{{ business_group }}/{{ env }}/{{ application }}/{{ item | basename | regex_replace('.j2','') }}" force=yes
29    with_fileglob:
30      - "{{ ws }}/{{ business_group }}/{{ env }}/{{ application }}/{{ app_templates }}/*.j2"
31
32  - name: "Creating {{ env }} namespace"
33    command: "kubectl create namespace {{ env }}"
34    ignore_errors: yes
35
36  - name: Deploying Dockerhub secrets
37    command: "kubectl create secret docker-registry dockersecrete --docker-server=https://index.docker.io/v1/
38    --docker-username={{ dockeruser }} --docker-password={{ dockerpwd }} --namespace={{ env }}"
39    no_log: True
40    ignore_errors: yes
41
42
43
44  - name: "Deploying {{ application }} Application"
45    command: "kubectl apply -f ~/kube-deploy/{{ business_group }}/{{ env }}/{{ application }}/ --namespace={{ env }}"
46    #ignore_errors: yes
47
```

In the above ansible file, we are executing the tasks as a deployer user and the k8s cluster configuration is also done to this user(/home/deployer/.kube/config) so that we can execute kubectl commands to create the deployments in k8s cluster.

The ansible vars will be replaced by the runtime variables while having --extra-vars in ansible-playbook

The secrets.yaml has sensitive data, so it is protected with ansible-vault . and we also give the password file path to it in the execution command.

The task process in deploy.yml consists of 3 parts:

1. Creating a working directory.
2. Having the latest devops code in the existing workspace.
3. Rendering jinja2 templates (passes values to variables and removes .j2)
4. Creating a namespace (ignores if already exists)
5. Deploying the infra

deployment.yml.j2

file:

```
kaleidofin-web > development > kal-agent > app_templates > deployment.yml.j2
 1  apiVersion: apps/v1
 2  kind: Deployment
 3  metadata:
 4    creationTimestamp: null
 5    labels:
 6      run: kal-agent
 7    name: kal-agent
 8    namespace: {{ env }}
 9  spec:
10    replicas: 1
11    selector:
12      matchLabels:
13        run: kal-agent
14    strategy: {}
15    template:
16      metadata:
17        creationTimestamp: null
18        labels:
19          run: kal-agent
20    spec:
21      containers:
22        - image: kaleidofin/kfin:{{ image_tag }}
23          name: kal-agent
24          imagePullPolicy: Always
25          #envFrom:
26          #  - configMapRef:
27          #    name: kal-agent
28        ports:
29          - containerPort: 80
30
31      #resources:
32      #  requests:
33      #    cpu: "{{kal_agent_cpu_reserve}}"
34      #    memory: "{{kal_agent_memory_reserve}}"
35      #  limits:
36      #    cpu: "{{kal_agent_cpu_limit}}"
37      #    memory: "{{kal_agent_memory_limit}}" */
38
39      imagePullSecrets:
40        - name: dockersecrete
41
42  status: {}
```

As we are pulling the container image from the private docker hub repository, we should use k8s secrets here to pull the image from Dockerhub.

In the final task, deployment.yaml file with specified namespace is executed.

This is how the deployment job configuration is done.