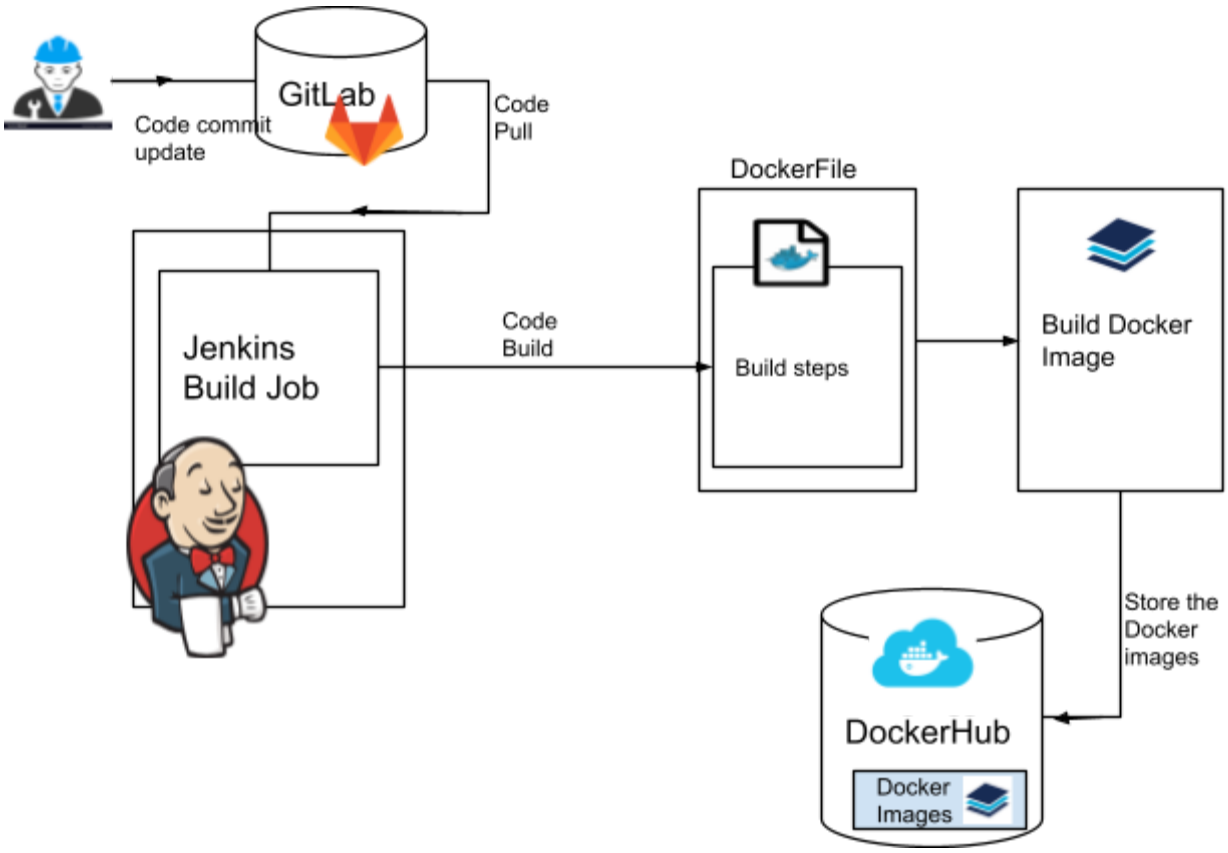
Build and deployment Jobs Configuration

Application Build Process



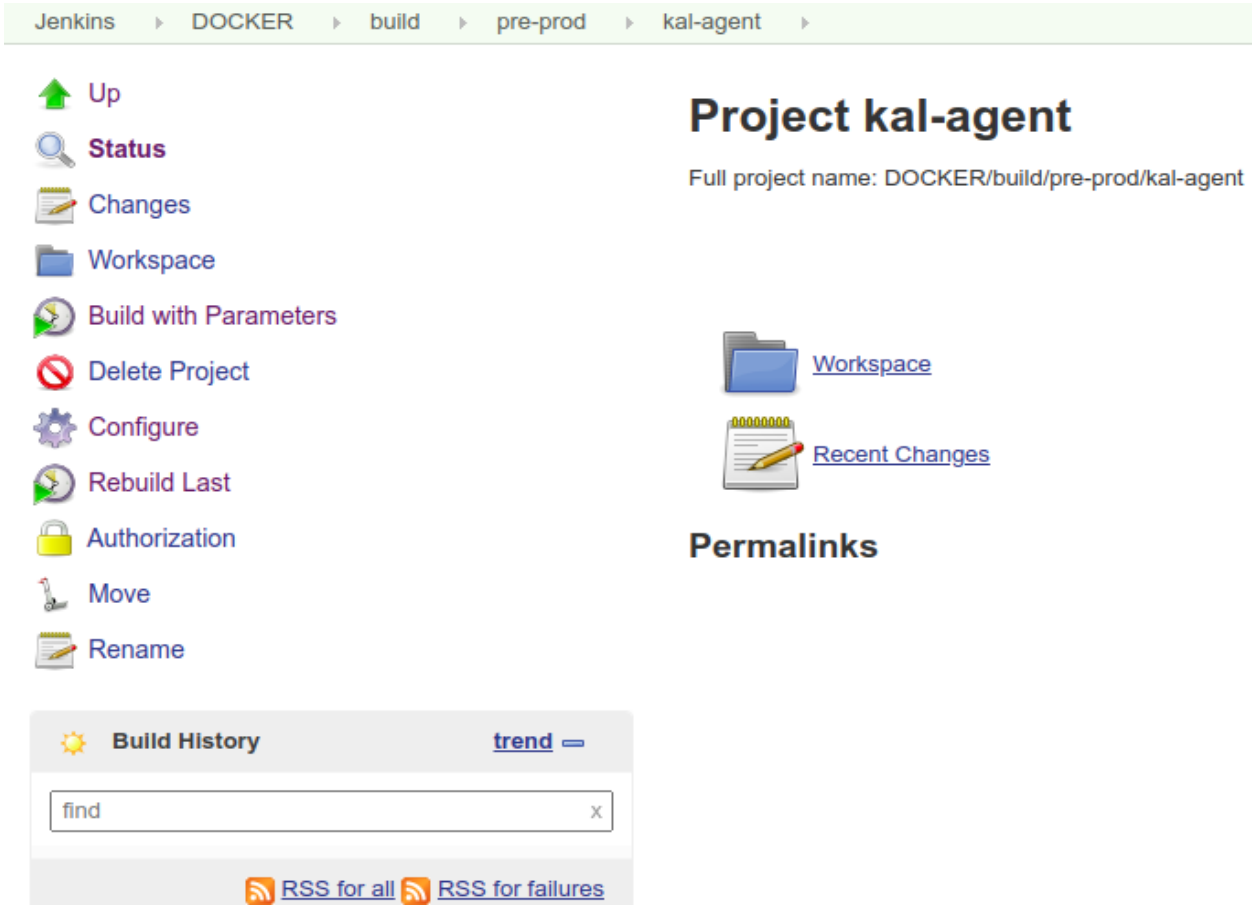
The developer makes a request to do build/deploy after pushing the latest changes to the GitLab.

Developers can select respective branches to build the code.

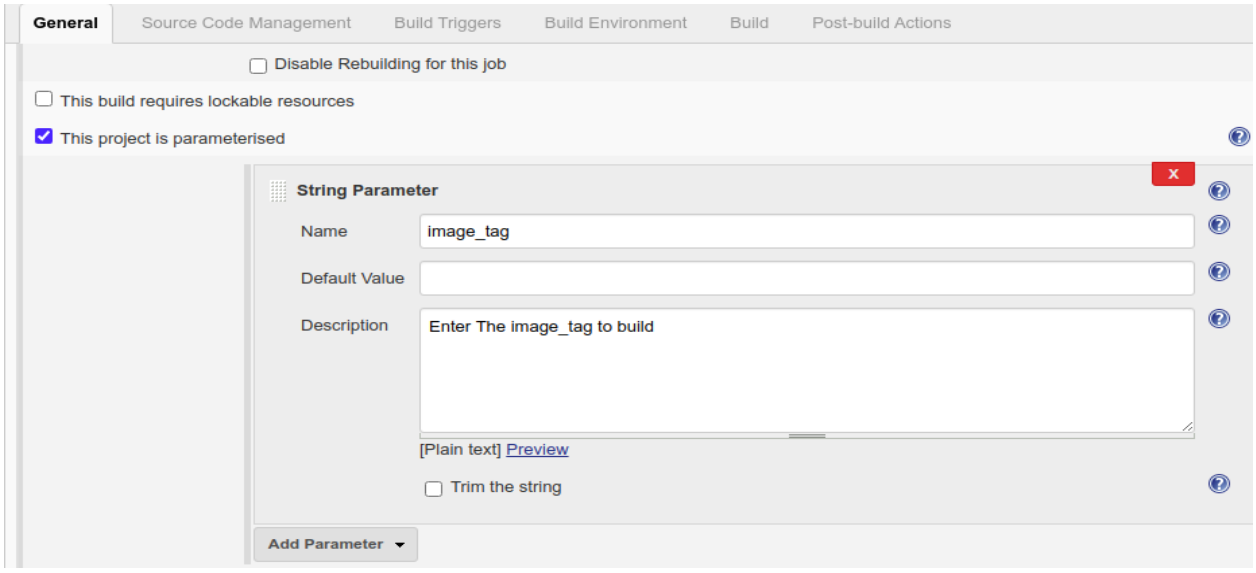
As per the developer request, we need to do build and deployment to reflect the changes in application. We have two separate jobs for application build and Deployment.

Build job configuration:

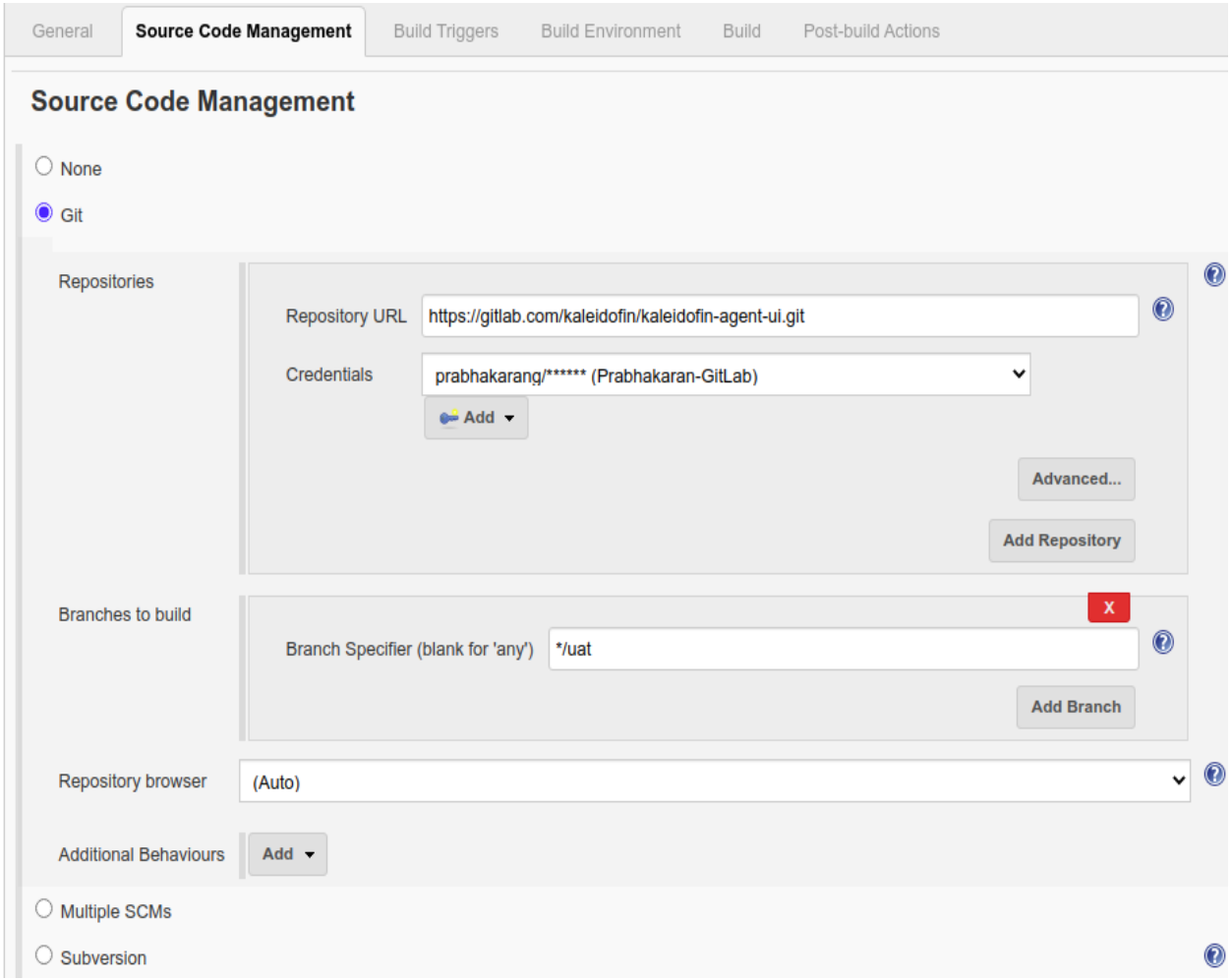
Go to build job and choose an environment. And click on any servers in the pre-prod environment for instance.



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



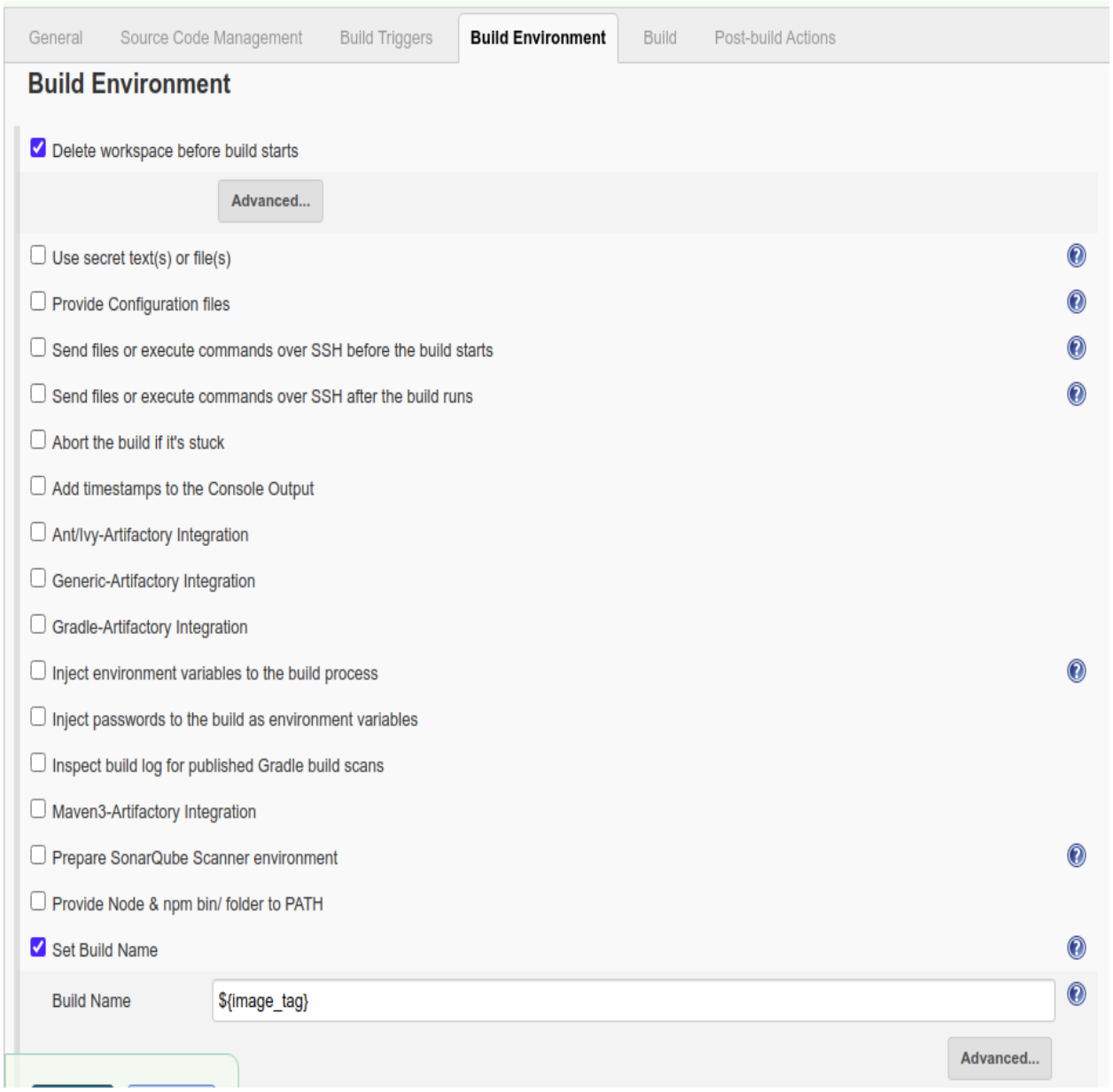
Under the General tab section, we can see the job is parameterized. By this we are asked to give Image_tag at the time of build.



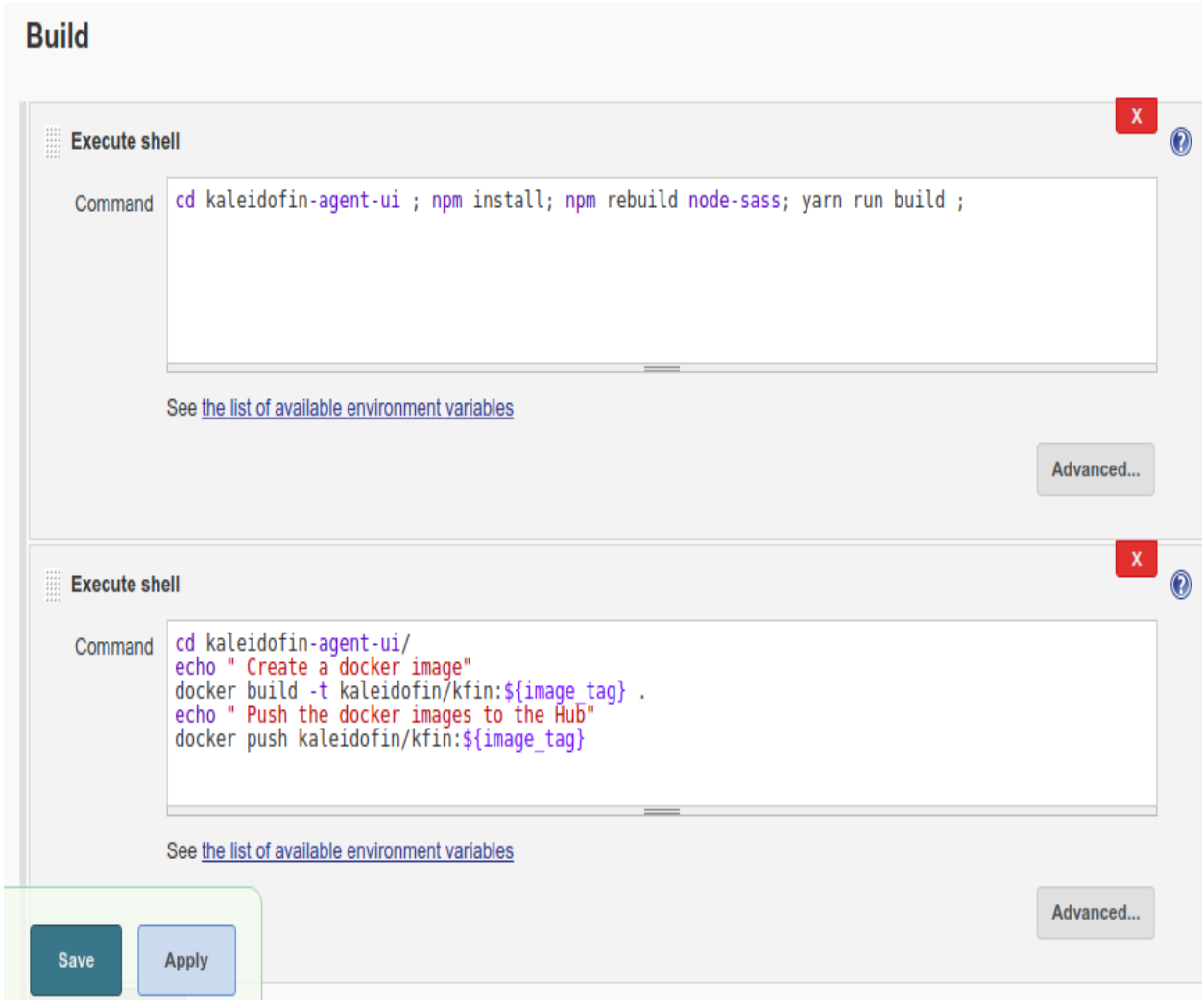
Three things to apply here in SCM(Git):

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.



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

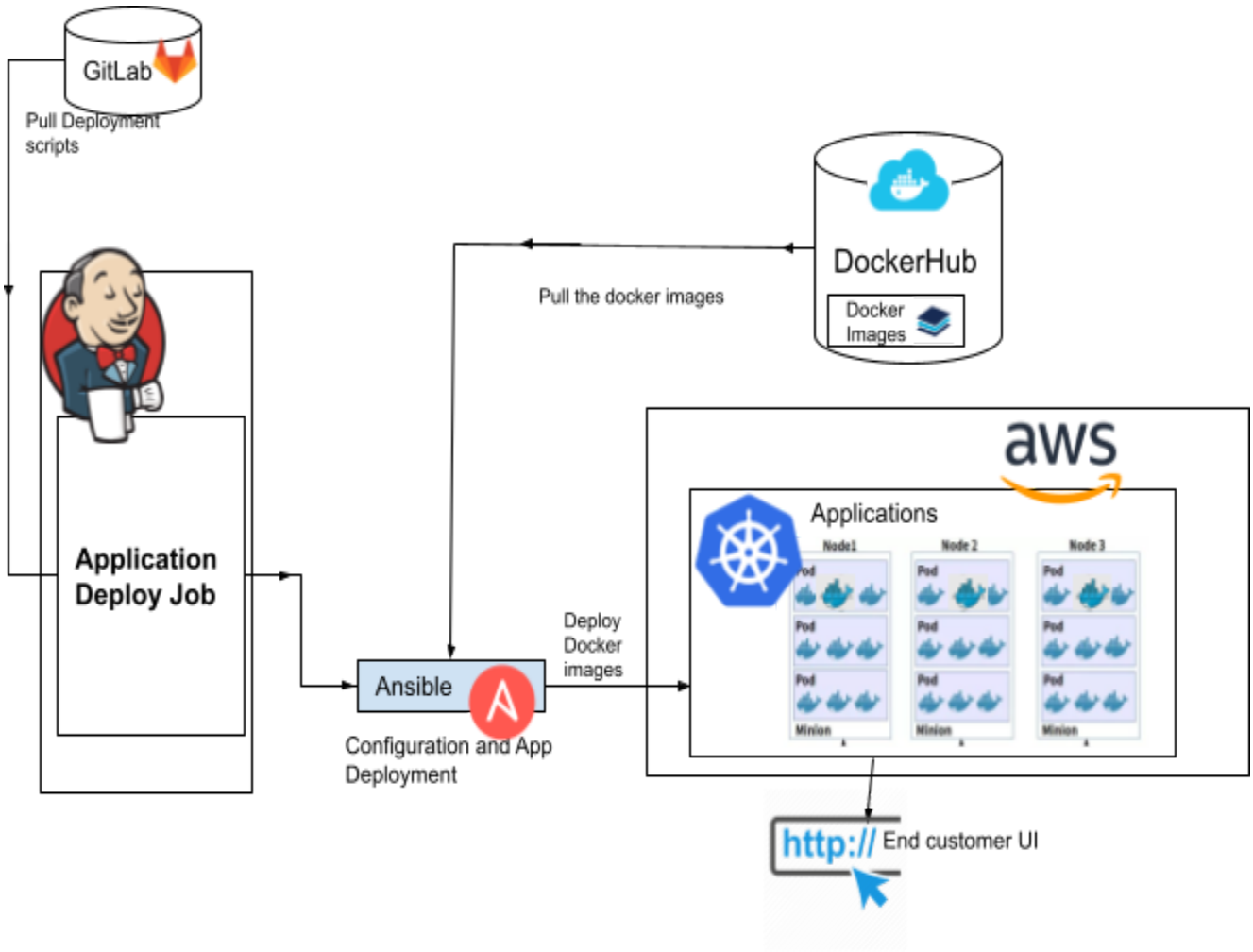


The commands in the first section, used to install any dependencies and generate an artifact which can be used to build an image.

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

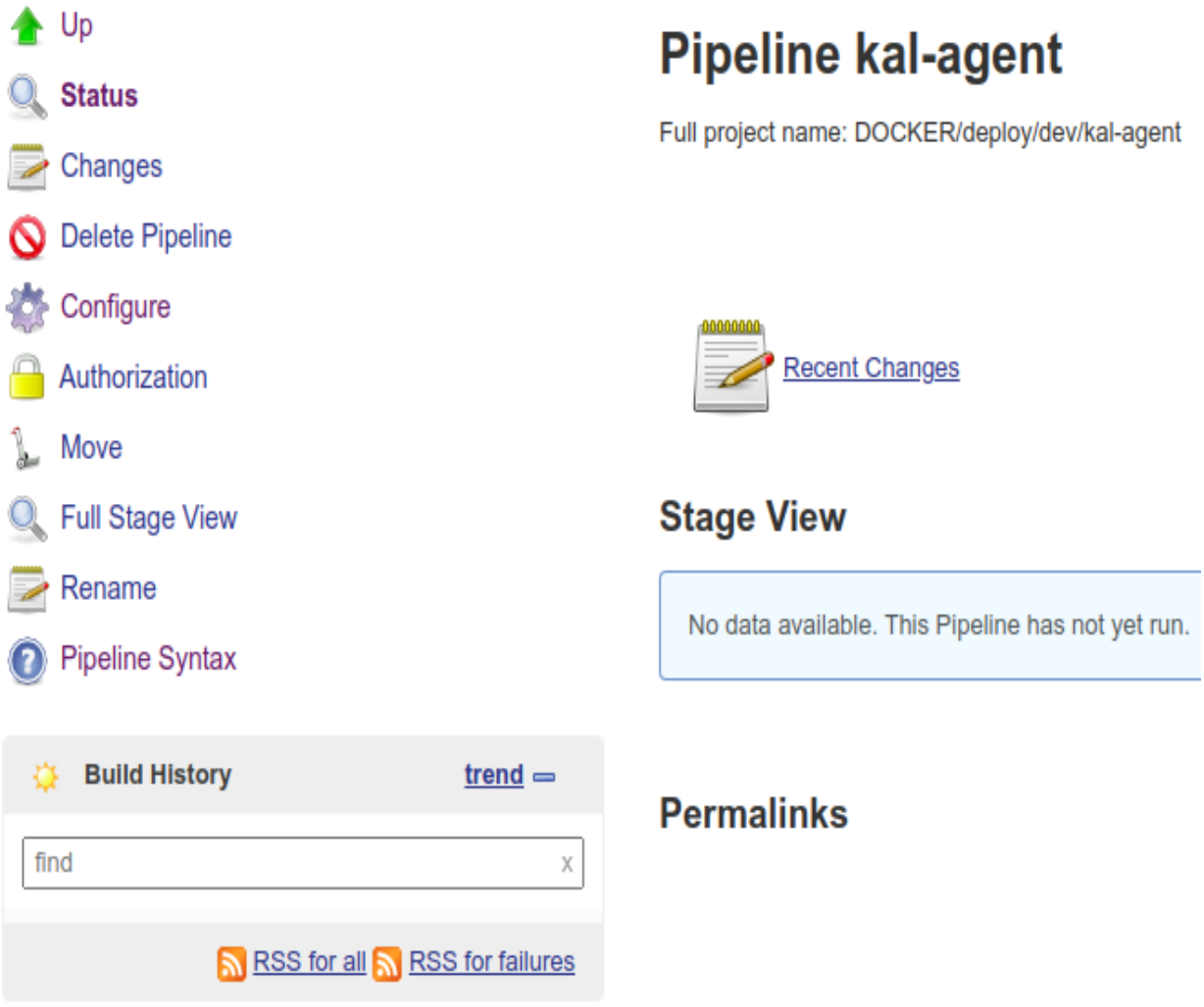
Deployment Process



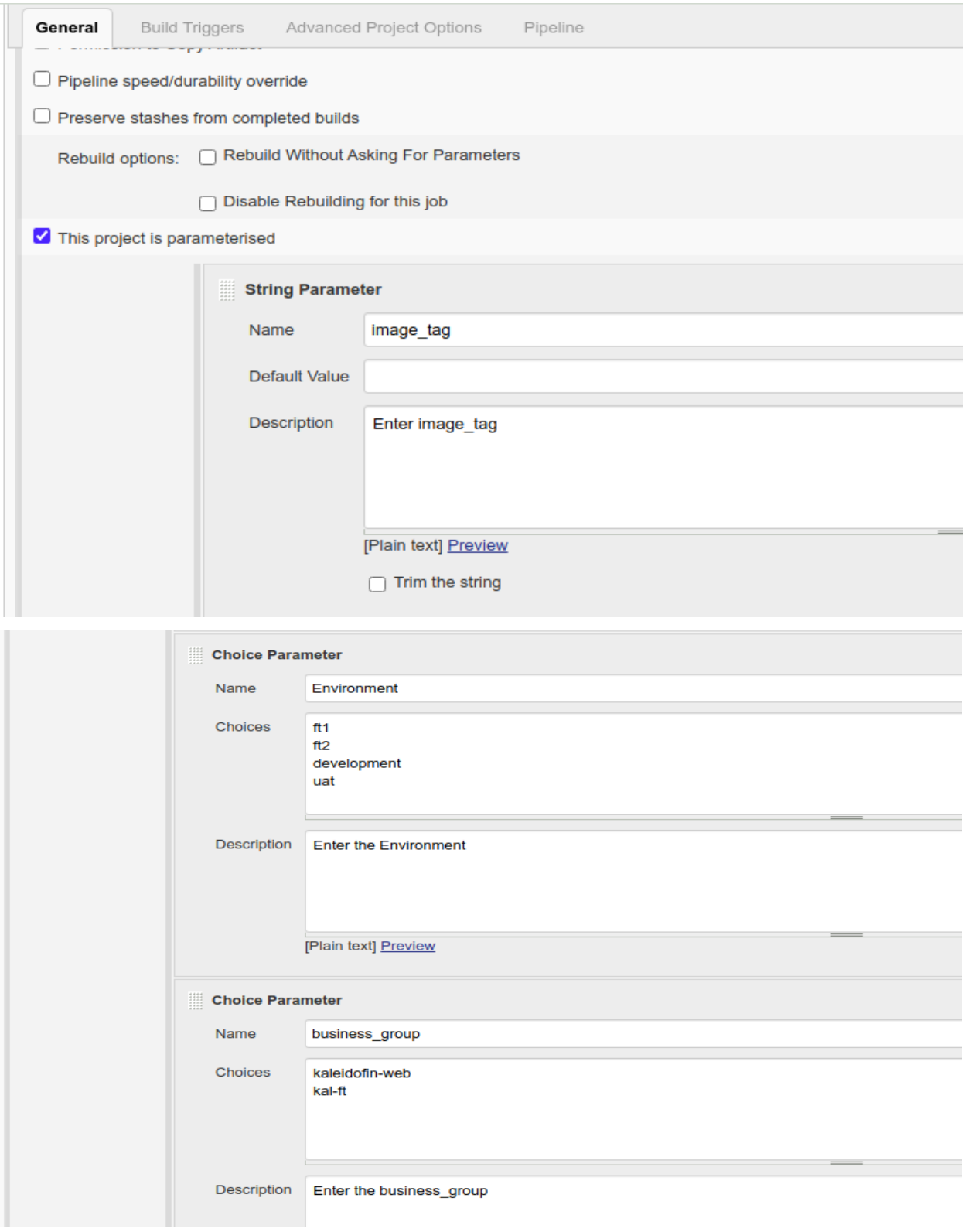
Jenkins and Ansible are running on the same machine (EC2). All the application deployments are deployed into EKS k8s cluster which is managed by AWS.

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.

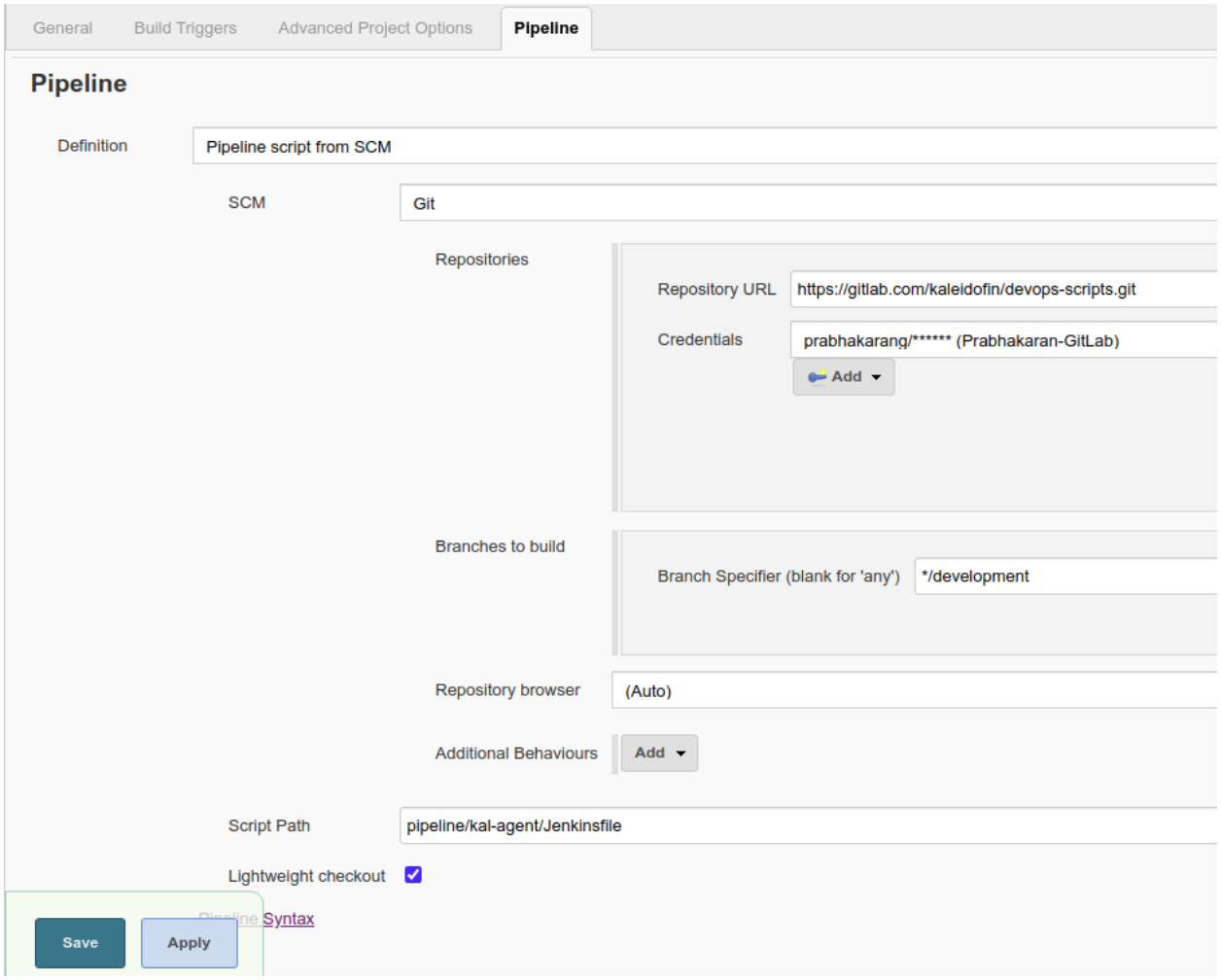


Click on the configure button to see how the deploy job is configured.

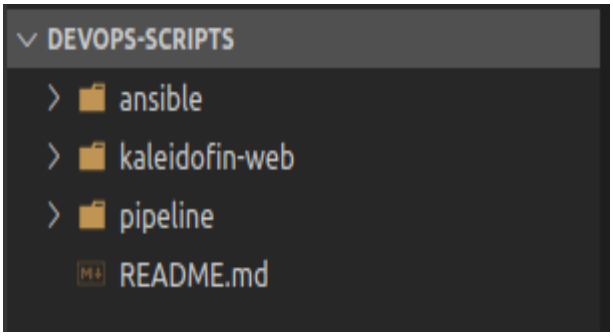


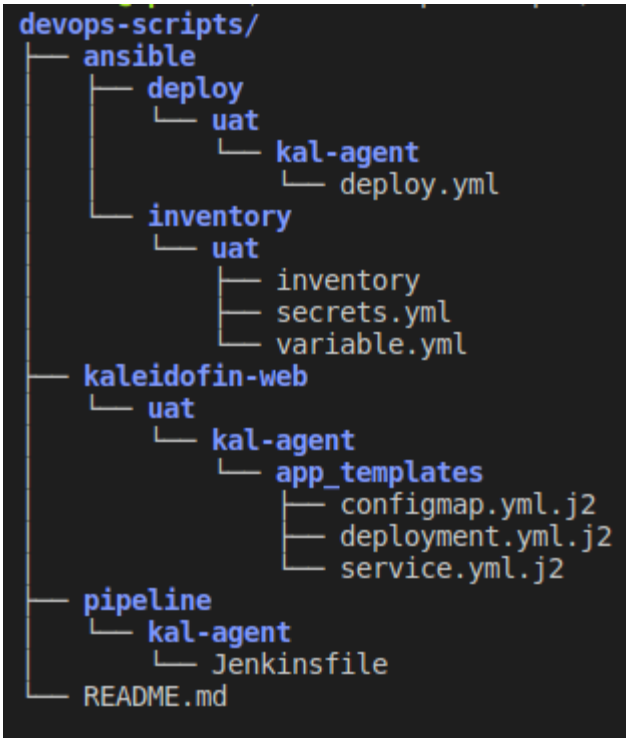
Under the General tab, we have two parameters defined. The string parameter is to give a name to image_tag and choice parameter is to choose the environments.

Under pipeline section, we have a jenkins pipeline script,



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_BASE_NAME}/deploy.yml
      --vault-password-file /var/lib/jenkins/secrets/vault-pass
      -extra-vars 'env=${params.Environment} business_group=${params.business_group} image_tag=${params.image_tag} ws=$WORKSPA
    }
   }
 }
}
```

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]

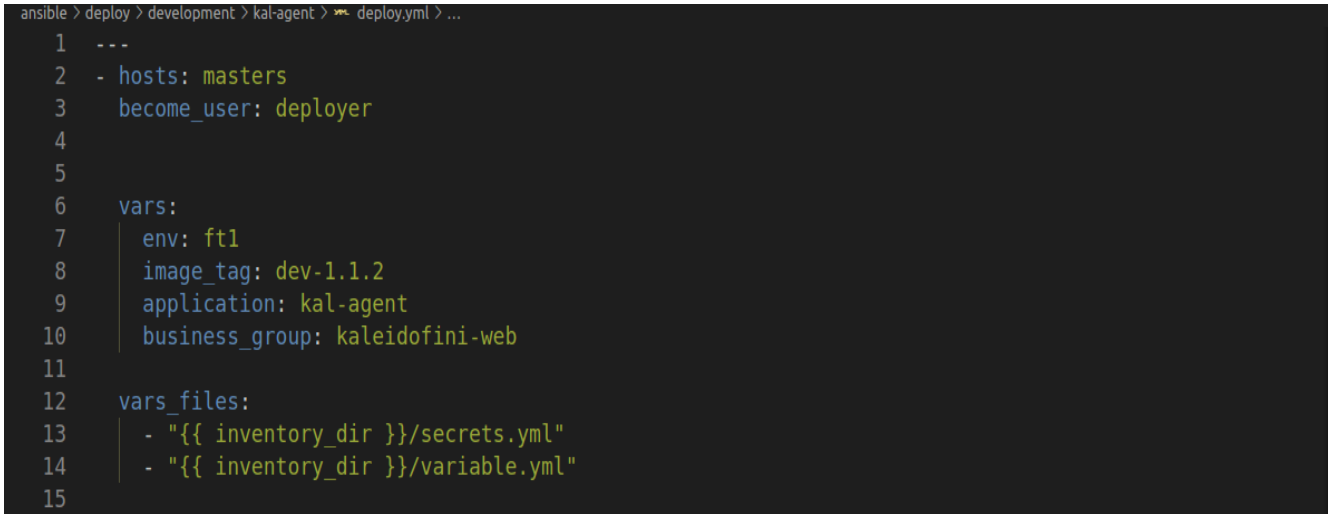
1. Calls the kubectl command to run the deployment files
2. extra params like: env, business_groups, image_tag and ws
3. inventory file where ansible runs with credentials.

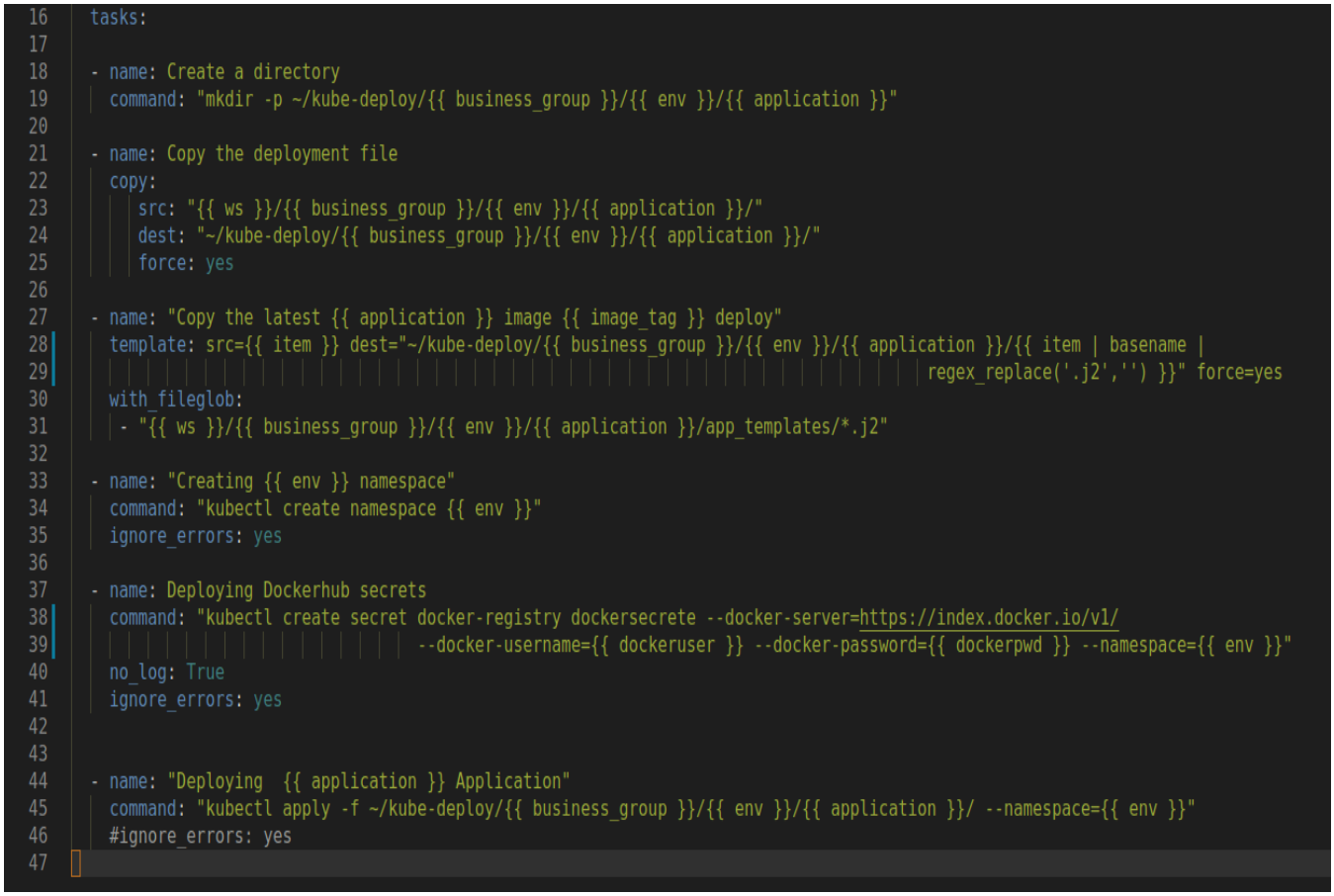
The inventory file is located at path: ansible/inventory/uat/inventory

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





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 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 given 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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: kal-agent
  name: kal-agent
  namespace: {{ env }}
spec:
  replicas: 1
  selector:
    matchLabels:
      run: kal-agent
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: kal-agent
    spec:
      containers:
      - image: kaleidofin/kfin:{{ image_tag }}
        name: kal-agent
        imagePullPolicy: Always
        #envFrom:
        # - configMapRef:
        #       name: kal-agent
        ports:
         - containerPort: 80

        #resources:
        #  requests:
        #    cpu: "{{kal_agent_cpu_reserve}}"
        #    memory: "{{kal_agent_memory_reserve}}"
        # limits:
        #    cpu: "{{kal_agent_cpu_limit}}"
        #    memory: "{{kal_agent_memory_limit}}" */

      imagePullSecrets:
      - name: dockersecrete

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.