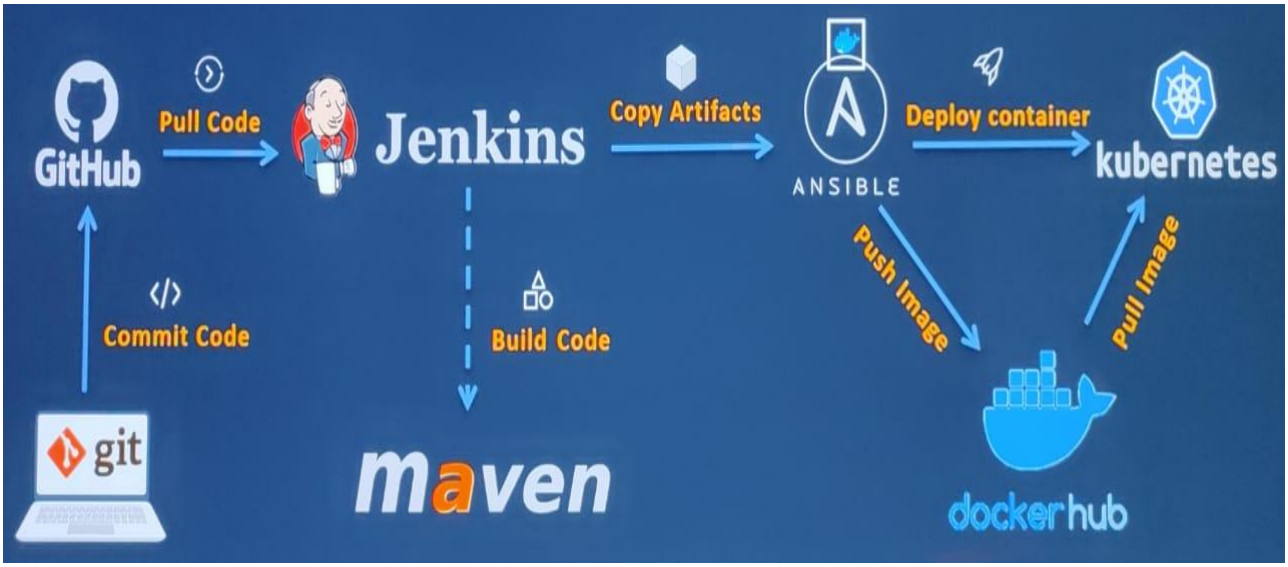
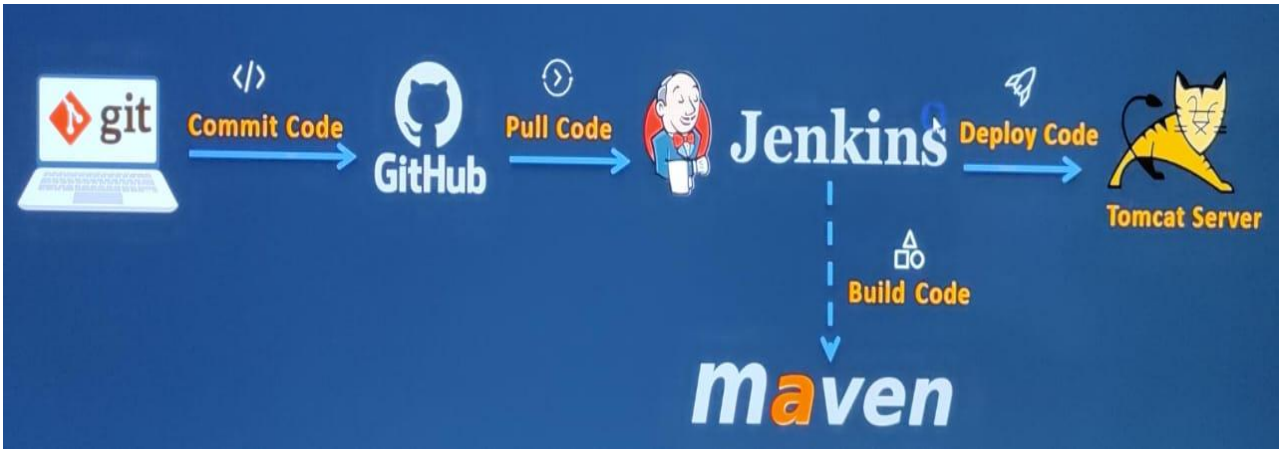
AWS DevOps Project using CI/CD Pipeline

Services used for the AWS CI/CD DevOps Project -

AWS DevOps Project Lifecycle -

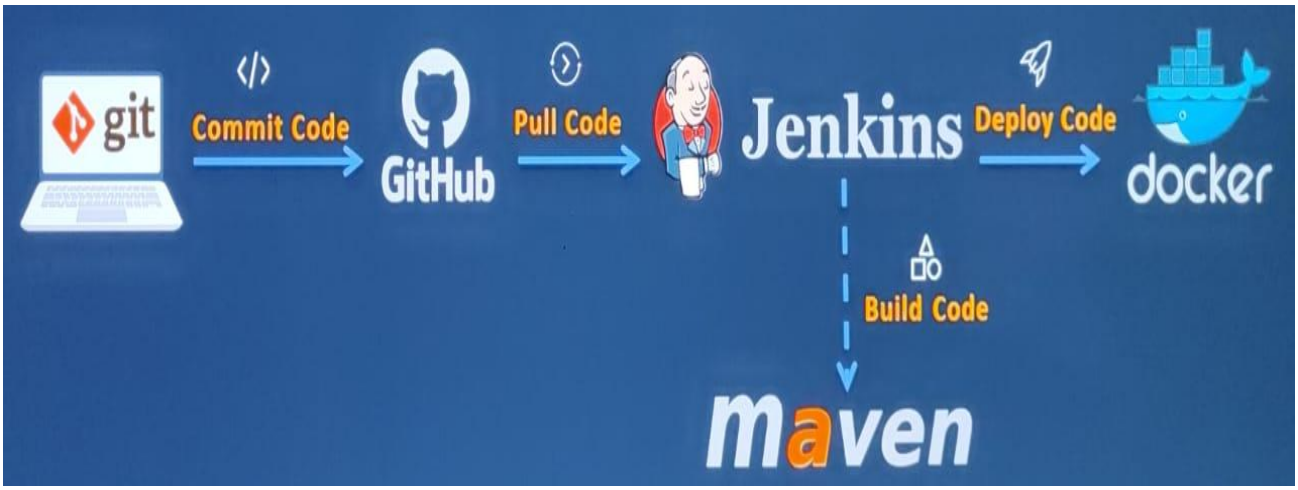


Stage 1: Integrating GIT, Jenkins, Maven & Tomcat into the CI/CD Pipeline

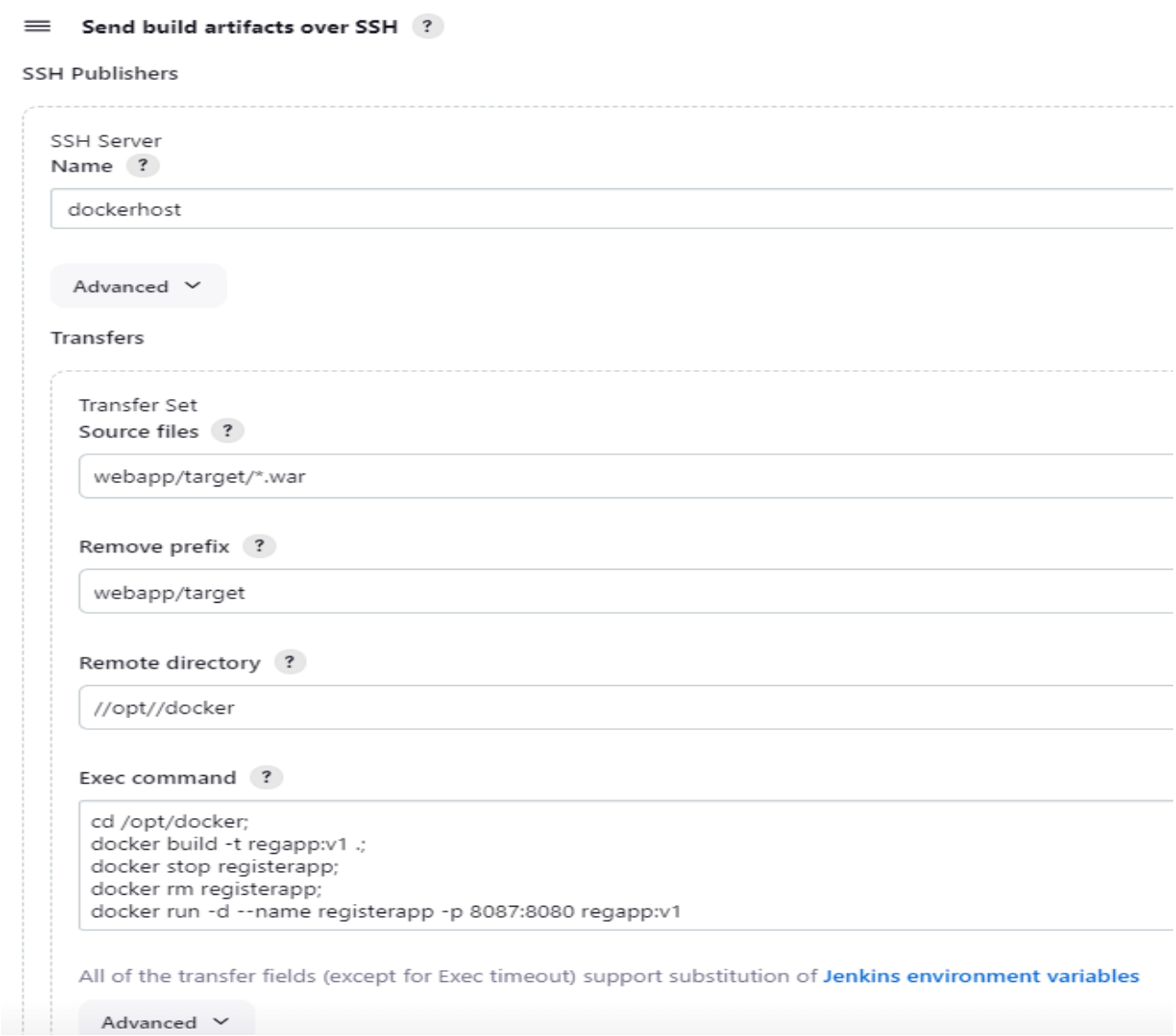


- **Create an index.jsp file** which contains our website source code.
- Navigate to the folder where the **index.jsp** file is present from **Git Bash** and **push it to our GitHub repository**.
- For any **updates to the source code,** we follow the same steps as above to commit the code in our **Github repository**.
- **Launch an EC2 instance** in AWS and **install Jenkins Service** inside the server. Enable ports 8080, 22 in the **Security group** of the instance.
- Use the command **cat /var/lib/jenkins/secrets/initialAdminPassword** to find our Jenkins **admin** user password. Use command **service jenkins start/stop/status** to start/stop/check status of Jenkins.
- Open **Jenkins GUI** using **<public ip of Jenkins server>:<port>** and log in to the admin user using the password obtained in the previous step.
- **Integrate our Github repository with Jenkins in the GUI** by installing the **Github plugin.**
- Setup a Jenkins **Freestyle** Job to **pull code from Github automatically** when a change is made to the source code.
- **Install Maven inside the Jenkins server** and then add the **Maven Plugin** inside Jenkins GUI.
- Create a custom Jenkins' job to **set up the CI/CD Pipeline.**
- **Launch an EC2 instance** in AWS and **install Tomcat Service** inside the server to host the website. Enable ports 8080, 22 in the Security group of the instance. Create and attach an **elastic ip address** to this instance.
- **Integrate Tomcat Server with Jenkins.** Under **Plugin Manager** - install **Deploy to container artifact**.
- **Deploy artifacts** onto the **Tomcat Server** using the **CI/CD Pipeline.**
- **Automate the build** in Jenkins **for Continuous Integration and Continuous Delivery** using **Poll SCM** (* * * * *) feature.
- The **Output** for this stage can be observed through the web browser with the **<public ip of Tomcat server>:8080/webapp**.

Stage 2: Integrating Docker into the CI/CD Pipeline

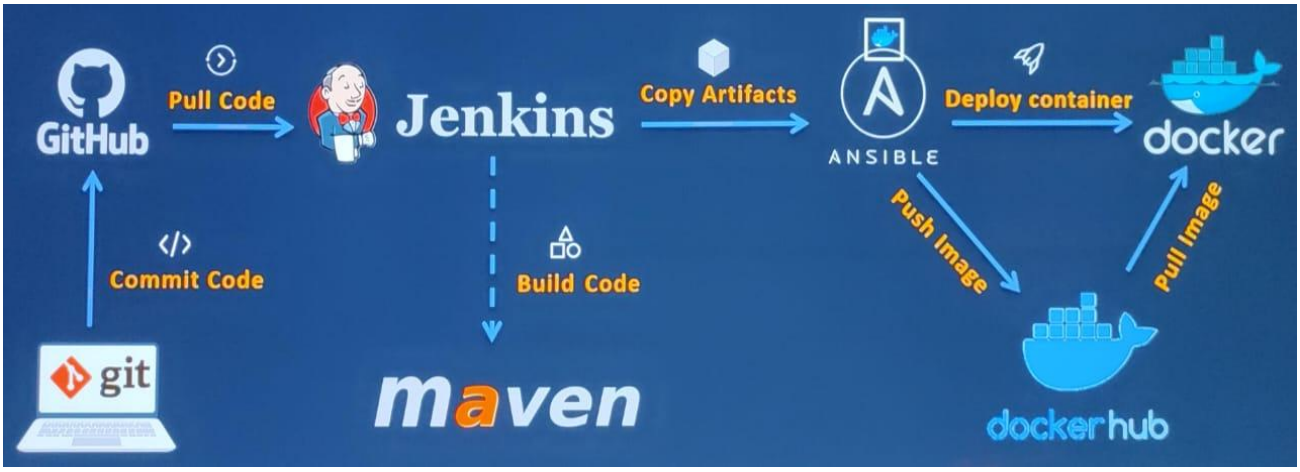


- **Launch an EC2 instance** in AWS and **install Docker Service** inside the server. Enable ports 8080-9000, 22 in the Security group of the instance. The command **service docker start/stop/status** can be used to start/stop/check status of docker.
- **Create a Docker Hub account** to push/pull our images.
- Login to our Docker Hub account inside the Docker server and **create an image of the Tomcat server** from **hub.docker.com**.
- **Create a Dockerfile** inside the **Docker host** at **/opt/Docker** to pull the latest **Tomcat image** from **hub.docker.com**, **build the image** inside the Dockerhost and then use **Run** command to **create the container** for the image.
- Then **push the Tomcat image** we have created **to Docker Hub.**
- Create a custom Jenkins' job to **set up the CI/CD Pipeline.**
- Integrate **Docker Host** with **Jenkins** through GUI. Install the **Publish Over SSH** plugin in Jenkins**.**
- Login to **Jenkins GUI - BuildAndDeployOnContainer** job **- Configure** - enter details as mentioned in below image. Click Apply and Save once done. The **exec command** in the below picture is used to **automate the deployment of the Docker Container**.



- Launch the **Jenkins job** to **build and copy artifacts** onto the **Docker host**.
- Add the **artifacts** created in **Jenkins** to the **Docker host Tomcat image**, inside the **Dockerfile** to automate the process so far **(.war file)**.
- Use the **Jenkins Job** to automate the **build and deployment** of **Docker container** using the **Exec command** inside the Jenkins **Configure** section in the GUI.
- Add the Docker commands to the **Exec command** section as well.
- Now when we make a change inside the source code of our **index.jsp** file , our **Jenkins job** will pull code from **Github** automatically, **create & push** the docker image onto **hub.docker.com**, thus allowing our **Jenkins job** to automate the CI/CD Pipeline to deploy our application into the docker container.
- The **Output** for this stage can be observed through the web browser with the **<public ip of Docker host>:8087/webapp**.

Stage 3: Integrating Ansible into the CI/CD Pipeline



- **Ansible** can configure systems, deploy software, and orchestrate advanced workflows to support application deployment, system updates, and more.
- We use **Ansible** for the automatic deployment of the website through docker image with the help of **Jenkins**.
- **Launch an EC2 instance** in AWS for **Ansible**. Enable ports 8080-9000, 22 in the Security group of the instance.
- Login to the ansible server and go to **/etc/ansible** and edit the **hosts** file in this path by adding two groups - **[dockerhost] & [ansible]** along with their corresponding **private ip** addresses.
- Create a user called **ansadmin** and set a password for that user. Add that user to **sudoers** file to allow root access to it.
- Go to **/etc/ssh** and edit the **sshd_config** file to enable **password based authentication**.
- Switch user to **ansadmin** and then generate the ssh keys for **ansadmin** user through the command **ssh-keygen**. The generated keys will be stored at **/home/ansadmin/.ssh**.
- Install **Ansible** to this server after completing the above steps.
- Next we need to make a connection between the **Ansible server** and the **Docker host server**.
- Inside **Docker server**- do the same steps done for **Ansible server (Note - Don't generate ssh keys)**.
- Go back to **Ansible server** as **ansadmin** user and use the command **ssh-copy-id <private ip address of the Docker server>** to copy the **ssh keys** from **Ansible** to **Docker host**.This step is used to give the **Docker host** access to the **Ansible** server.This is to establish the connection between **Ansible** and **Docker host**.
- Use command **ansible all -m ping** to check if the connection between the **Ansible** and **Docker** server is successful. Use command **ansible all -m command -a uptime** to check the uptime of the server.
- **Integrate Ansible with Jenkins** through the **Jenkins GUI**. Use the **publish over ssh** plugin we had previously installed for the **Docker host**.This is to automate the **Jenkins job** to use the **Ansible Playbook.**
- Create an **Ansible playbook** inside the **Ansible server** at **/opt/Docker** path with the filename **regapp.yml**.

- Inside the **regapp.yml** file type the below code to run as the **Ansible playbook** -

```
- hosts: ansible

  tasks:
  - name: create docker image
    command: docker build -t regapp:latest .
    args:
      chdir: /opt/docker

  - name: create tag to push image onto dockerhub
    command: docker tag regapp:latest corporate2/regapp:latest

  - name: push docker image
    command: docker push corporate2/regapp:latest
```

- Above **ansible-playbook** is used to create the docker image of the **Tomcat server** (which hosts our website), then assign a **tag** to the image and finally **push** the image built to **hub.docker.com**.
- **Create a deploy_regapp.yml** file for **Ansible** to **deploy the container**.

```
- hosts: dockerhost

  tasks:
  - name: stop existing container
    command: docker stop regapp-server
    ignore_errors: yes

  - name: remove the container
    command: docker rm regapp-server
    ignore_errors: yes

  - name: remove image
    command: docker rmi corporate2/regapp:latest
    ignore_errors: yes

  - name: create container
    command: docker run -d --name regapp-server -p 8082:8080 corporate2/regapp:latest
```

- Above **ansible-playbook** is used to stop & remove any existing containers and images. Then create a new container out of the new image pushed to **hub.docker.com**.
- **ansible-playbook <filename.yml>** - this command is used to run the **Ansible Playbook**.
- Ensure the **docker service is running** on **Ansible server**. Command**: service docker start/stop/status.**
- Login to **hub.docker.com** account using **docker login** command inside the **Ansible server**.
- Login to **Jenkins GUI - Copy_Artifacts_onto_Ansible** job **- Configure** - enter details as mentioned in below image. Click Apply and Save once done. The **exec command** in the below picture is used to **automate the deployment of the Ansible Playbook**.

Send build artifacts over SSH ?

SSH Publishers

SSH Server

Name ?

ansible-server

Advanced

Transfers

Transfer Set

Source files ?

webapp/target/*.war

Remove prefix ?

webapp/target

Remote directory ?

//opt//docker

Exec command ?

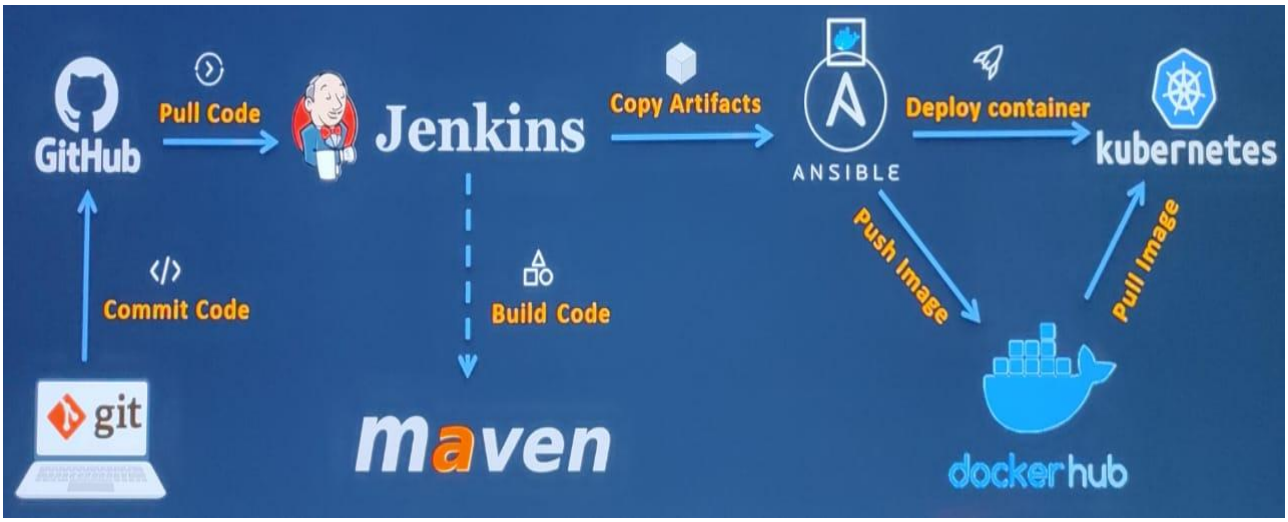
```
ansible-playbook /opt/docker/regapp.yml;
sleep 10;
ansible-playbook /opt/docker/deploy_regapp.yml
```

All of the transfer fields (except for Exec timeout) support substitution of Jenkins environment variables

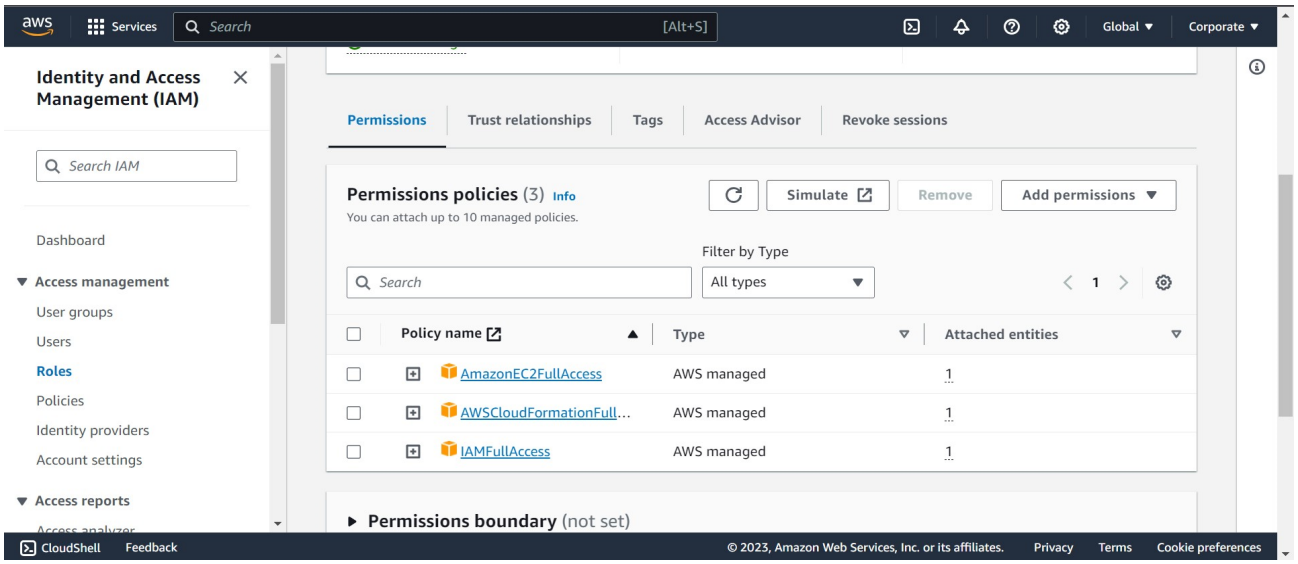
Advanced

- Whenever there is a **change made to the source code** of the website hosted by the **Tomcat Server**, this **Job** along with the specified **Ansible Playbooks** will automatically execute.
- The **Output** for this stage can be observed through the web browser with the **<public ip of Docker host>:8082/webapp**.

Stage 4: Integrating Kubernetes into the CI/CD Pipeline



- **Launch an EC2 instance** in AWS for **Kubernetes as a EKS_Bootstrap server**. Enable ports 8080-9000, 22 in the Security group of the instance.
- Update the **AWS CLI version** to the **latest available version**.using the following commands -
  [curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip" unzip awscliv2.zip
  sudo ./aws/install]
- We are using **kubectl along with eksctl** commands to **create and manage kubernetes clusters** in AWS through the **EKS_Bootstrap Server** we have created.
- Install **kubectl** using the following command in the **EKS_Bootstrap server** -
  [curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl]
  [chmod +x ./kubectl]
  [mv ./kubectl /usr/local/bin]
  [kubectl version --short --client]
- Install **eksctl** using the following command in the **EKS_Bootstrap server** -
  [curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp]
  [sudo mv /tmp/eksctl /usr/local/bin]
  [eksctl version]
- **Create an IAM role** as per the below image and attach it to the **EKS_Bootstrap server**.



- Create **cluster and nodes** using the following commands -
  **[eksctl create cluster --name corporate2 **
  **--region us-east-1 **
  --node-type t2.micro]
- A **Kubernetes Manifest file** is a **.yml file** that describes the desired state of a Kubernetes object. These objects can include **deployments, replica sets & services**. **Manifest files** define the specifications of the object, such as its **metadata, properties, and desired state**.
- **Create a manifest file** for **deployment** named **regapp-deployment.yml** with left content.
- **Create a manifest file** for **service** named **regapp-service.yml** with right content.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: corporate2-regapp
  labels:
     app: regapp

spec:
  replicas: 3
  selector:
    matchLabels:
      app: regapp

  template:
    metadata:
      labels:
        app: regapp
    spec:
      containers:
      - name: regapp
        image: corporate2/regapp
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: corporate2-service
  labels:
    app: regapp
spec:
  selector:
    app: regapp

  ports:
    - port: 8080
      targetPort: 8080

  type: LoadBalancer
```

- The **Deployment Manifest file** is used to Give the **EKS_Bootstrap Server** Permission to **Pull** the image from **hub.docker.com**.
- The **Service Manifest file** is used to give the **EKS_Bootstrap Server** permission to **use Port 8080.**
- Create a user called **ansadmin** and set a password for that user on **EKS_Bootstrap server**. Add that user to **sudoers** file to allow root access to it.
- Go to **/etc/ssh** and edit the **sshd_config** file to enable **password based authentication**.
- Switch user to **ansadmin** and then generate the **ssh keys** for **ansadmin** user through the command **ssh-keygen**. The generated keys will be stored at **/home/ansadmin/.ssh**.
- **Login to Ansible server** using **ansadmin** user, navigate to **/opt/docker** and make the following minor tweaks to the existing files.
  [mv regapp.yml create_image_regapp.yml]
  [mv deploy_regapp.yml docker_deployment.yml]
- Create a **hosts file** at **/opt/docker** with **groups and private ip addresses** of the two servers as shown below.

```
localhost

[kubernetes]
172.31.47.233

[ansible]
172.31.31.161
```

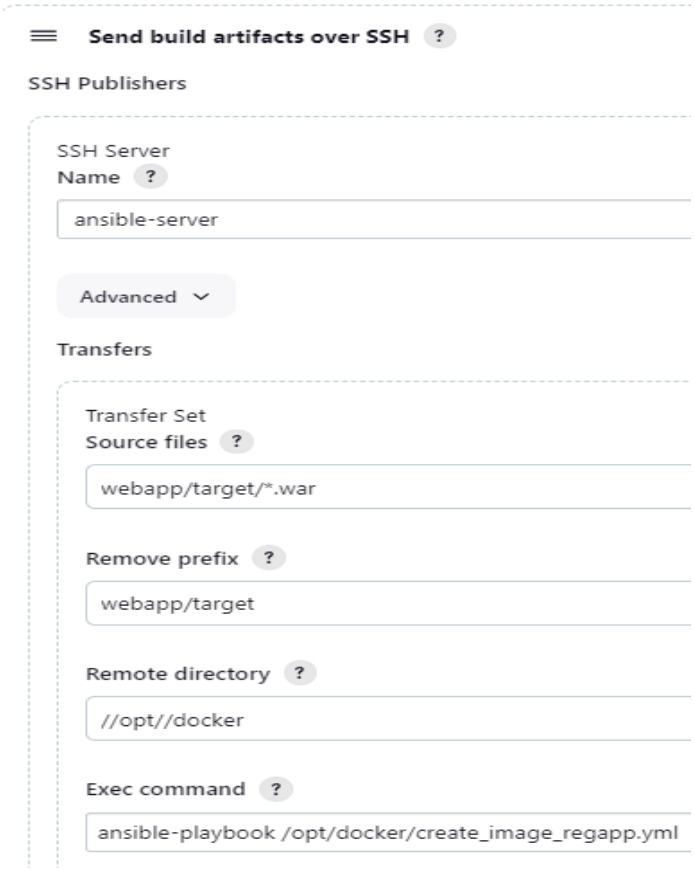
- Copy **ssh keys** to **EKS_Bootstrap server** using command **ssh-copy-id <private ip of EKS_Bootstrap server>**
- **Create Ansible playbook** for the **Deploy** and **Service files** as per below screenshots in the path - **/opt/docker**

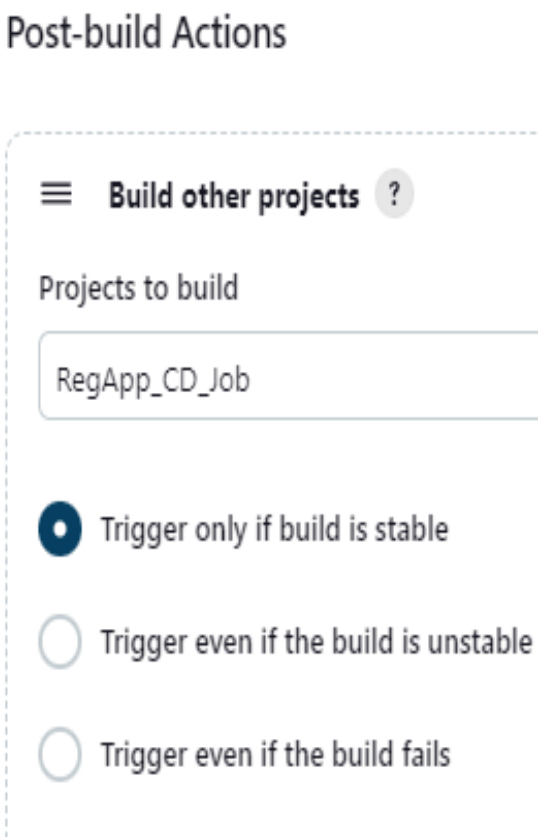
```
[ansadmin@ansible-server docker]$ cat kube_deploy.yml
---
- hosts: kubernetes
  #  become: true
  user: root

  tasks:
  - name: deploy regapp on kubernetes
    command: kubectl apply -f regapp-deployment.yml

  - name: create service for regapp
    command: kubectl apply -f regapp-service.yml

  - name: update deployment with new pods if image updated in docker hub
    command: kubectl rollout restart deployment.apps/corporate2-regapp
```

```
[ansadmin@ansible-server docker]$ cat kube_service.yml
---
- hosts: kubernetes
  #  become: true
  user: root

  tasks:
  - name: deploy regapp on kubernetes
    command: kubectl apply -f /root/regapp-service.yml
```

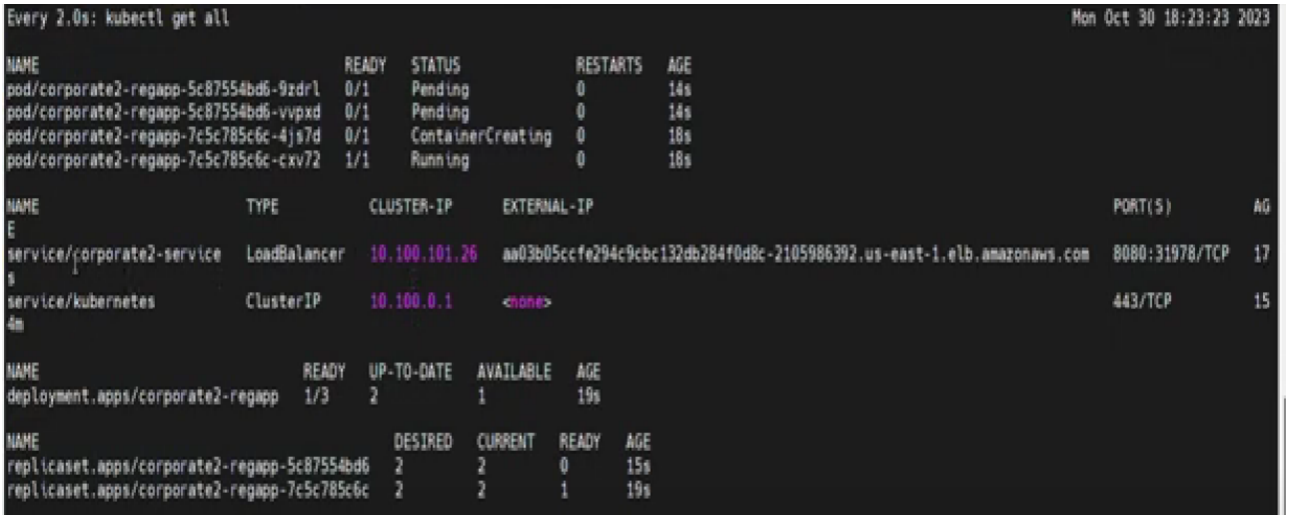
- Log in to **Jenkins GUI** and **Create a Custom Job** in **Freestyle** named **RegApp_CD_Job** for **Continuous Deployment**. Add **post build actions** - **Send Build Artifacts over SSH.** Configure as per below screenshots.

- **Create a Custom Job** in **Maven** named **RegApp_CI_Job** for **Continuous Integration** and configure as per below screenshots. Enable **Poll SCM** in this job.



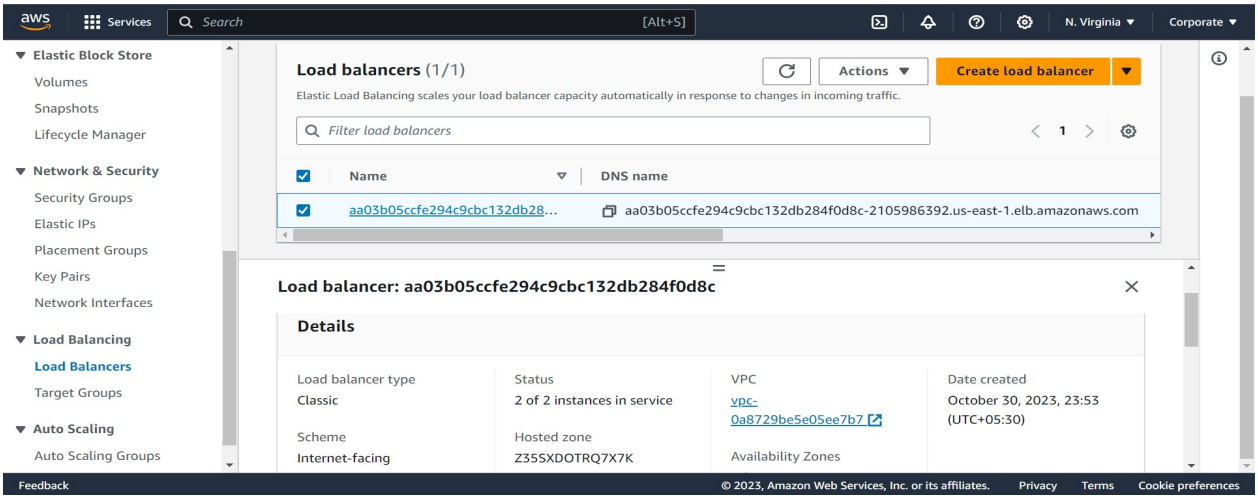


- On **Ansible Server**, start the docker service and use command **docker login** as **ansadmin** user to log in to **hub.docker.com** account.
- Any **commits to the code** or **manually building RegApp_CI_Job** will trigger the **CI/CD process**.
- The command used to display all **kubectl services/pods/deployments/LoadBalancers** is - **kubectl get all**.



```
Every 2.0s: kubectl get all                                                                                    Mon Oct 30 18:23:23 2023

NAME                                      READY   STATUS              RESTARTS   AGE
pod/corporate2-regapp-5c87554bd6-9zdrl    0/1     Pending             0          14s
pod/corporate2-regapp-5c87554bd6-vvpxd    0/1     Pending             0          14s
pod/corporate2-regapp-7c5c785c6c-4js7d    0/1     ContainerCreating   0          18s
pod/corporate2-regapp-7c5c785c6c-cxv72    1/1     Running             0          18s

NAME                         TYPE           CLUSTER-IP     EXTERNAL-IP                                                                PORT(S)        AG
E
service/corporate2-service   LoadBalancer   10.100.101.26  aa03b05ccfe294c9cbc132db284f0d8c-2105986392.us-east-1.elb.amazonaws.com   8080:31978/TCP   17
s
service/kubernetes           ClusterIP      10.100.0.1     <none>                                                                     443/TCP        15
4m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/corporate2-regapp    1/3     2            1           19s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/corporate2-regapp-5c87554bd6    2         2         0       15s
replicaset.apps/corporate2-regapp-7c5c785c6c    2         2         1       19s
```

- From the above image, we will get an **Elastic Load Balancer** generated by the **EKS Cluster**. This **ELB's <DNS name>:<Port>/webapp** as mentioned in below line, is used to direct internet traffic to the **Kubernetes Pods** generated by the **EKS Cluster** which are hosting the website. **[aa03b05ccfe294c9cbc132db284f0d8c-2105986392.us-east-1.elb.amazonaws.com:8080/webapp]**



- The **Final Output** can be observed through the web browser with the **<ELB's DNS name>:8080/webapp**.

Jenkins Jobs created for the DevOps CI/CD Pipeline Project -

S	W	Name ↓
⊗	☼	BuildAndDeployJob
✓	☼	BuildAndDeployOnContainer
✓	☼	Copy_Artifacts_onto_Ansible
✓	☼	FirstMavenProject
✓	☼	HelloWorldJob
✓	☼	PullCodeFromGitHub
✓	☼	RegApp_CD_Job
✓	☼	RegApp_CI_Job

List of AWS Amazon Linux AMI 2 EC2 Servers used for the DevOps Project -