COMPREHENSIVE CI/CD PIPELINE FOR DEPLOYMENT OF A WEB APPLICATION USING TOOLS

Git, GitHub, Jenkins, Maven, Ansible, Docker, AWS(EC2, CloudFormation,), Ansible, Kubernetes (AWS EKS)

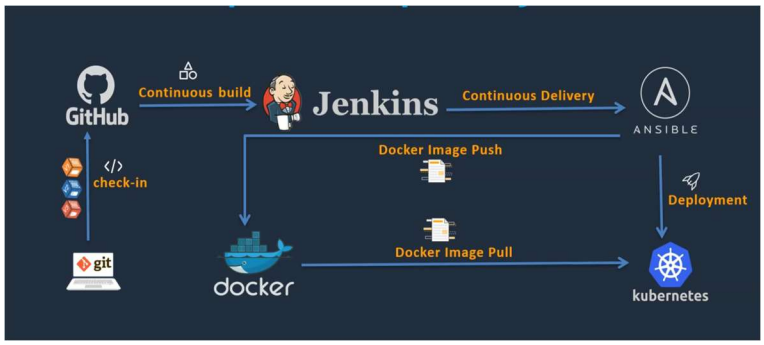
In this project I will set up a comprehensive CI/CD pipeline that automates the integration and delivery process from when updates to the source code for a web application are committed to GitHub repository

- Jenkins automatically pulls the code from the GitHub repository with the help of poll SCM enabled to detect changes to the source and builds the code with the help of Maven which generates a (.war) artifact and pushes onto an Ansible server

- Jenkins also automatically executes my Ansible playbook which creates and image with the artifact and commits it into my Docker hub repository

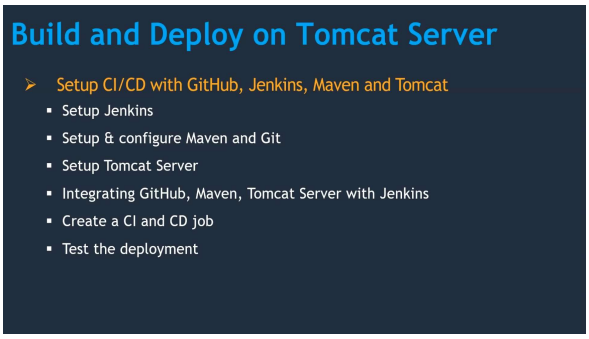
- in Jenkins, the successful build of the continuous integration job will trigger the continuous deployment job

- In the CD job, Jenkins will initialize and Ansible playbook that will execute a deployment and service file for a Kubernetes cluster with the image and finally the application is up and running with the latest code

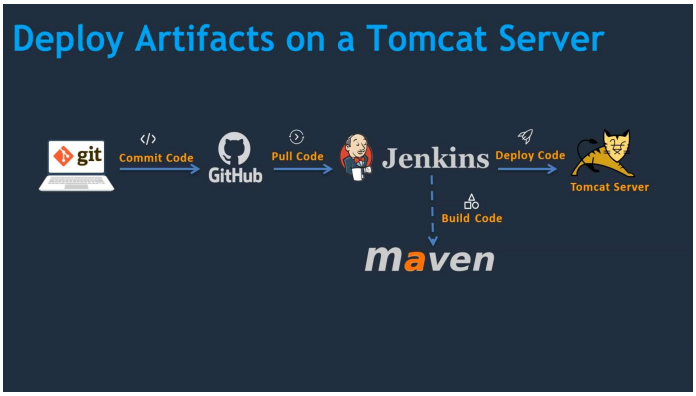


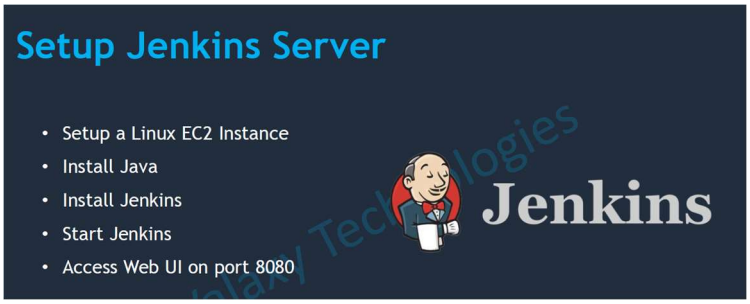
I am going to demonstrate this step by step, first manually on a virtual machine then automate the integration and deployment process with Jenkins on a Docker container, then Ansible (as a deployment tool to execute commands and configure my target environment in a more efficient way) and then Kubernetes container management system for high availability and fault tolerance using Amazon EKS

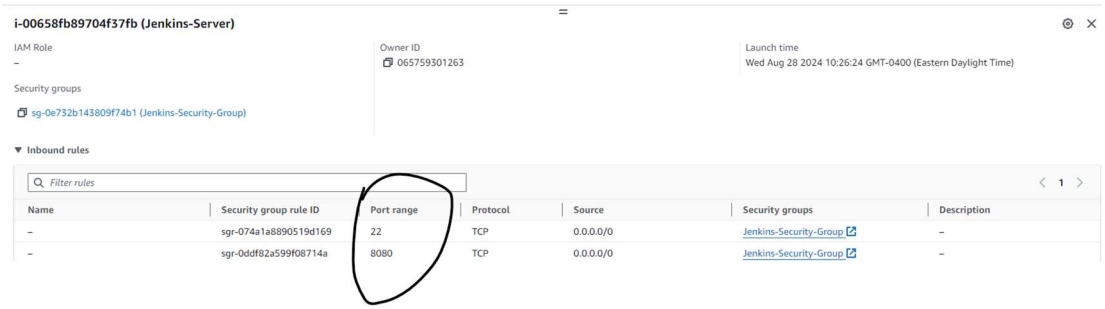
STEP 1

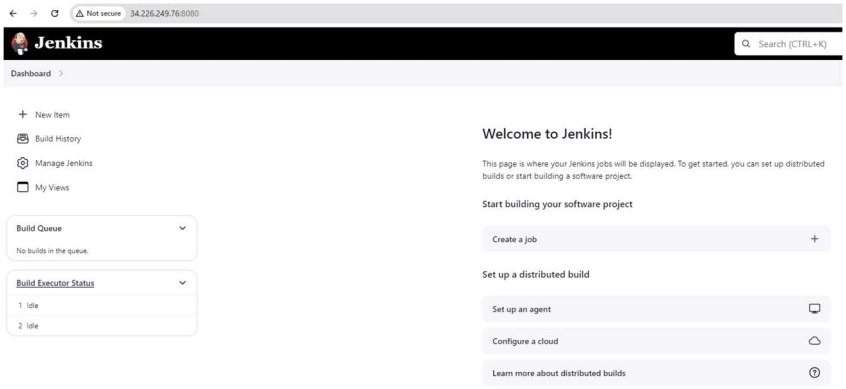


In this Section of the project, I will be deploying the source code in Tomcat on an AWS EC2 instance with amazon linux 2 AMI and port 22 and 8080 exposed in the security group to ssh into the instance and 8080 to access the web application

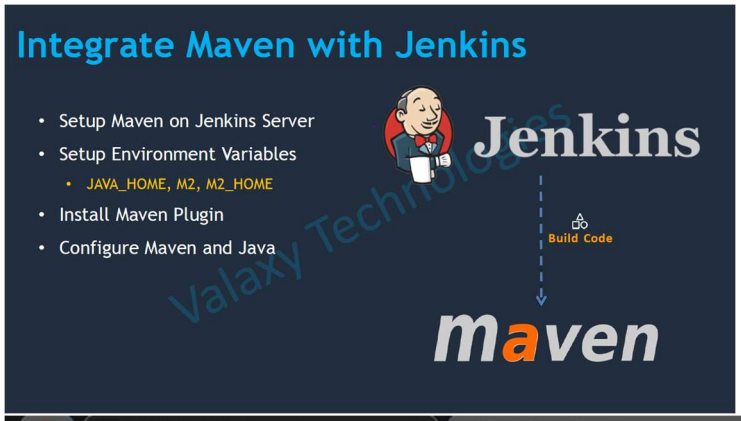


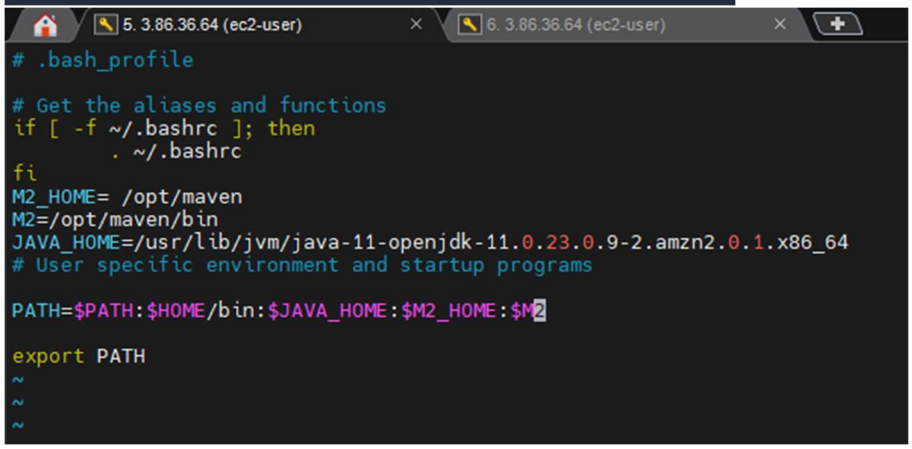






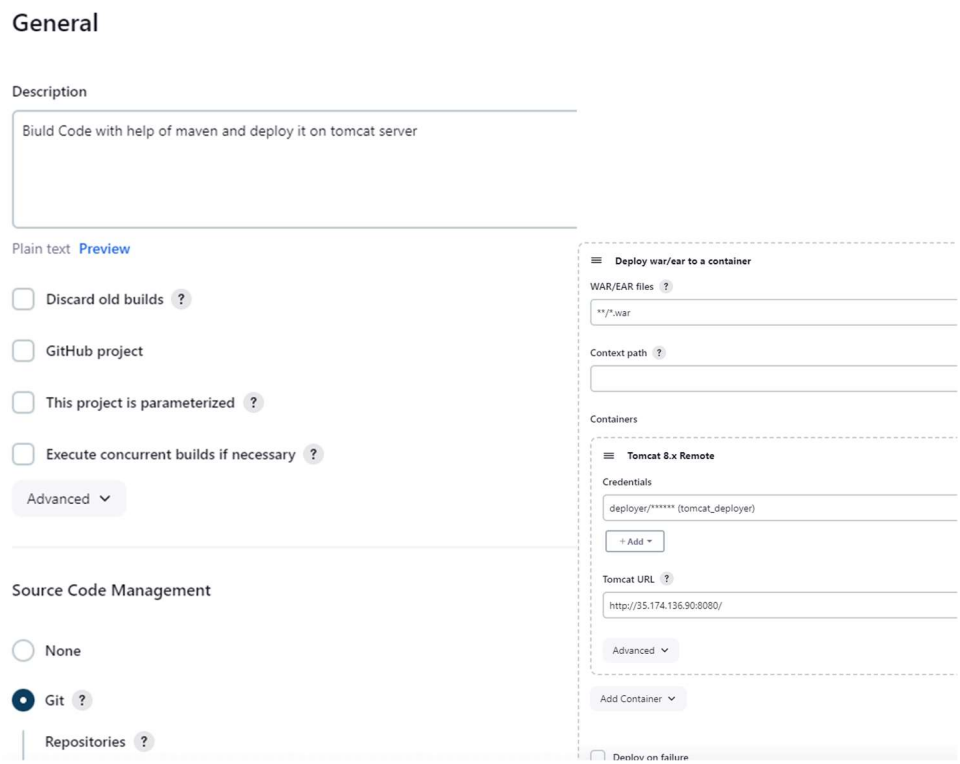
- Here, I Integrated Git with Jenkins
- Ran a Jenkins job to pull the source code from GitHub using the url from my repository and;
- Integrated Maven with Jenkins





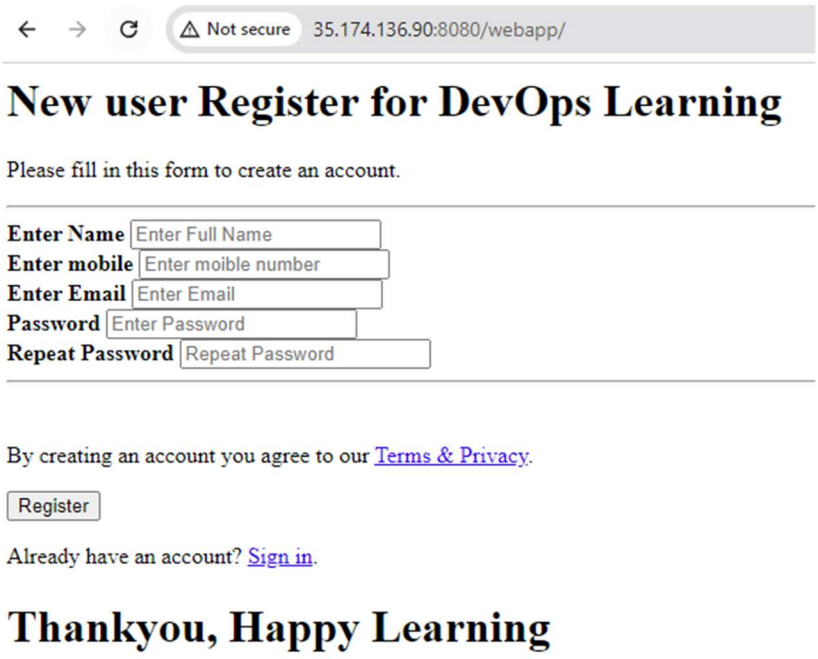
- I set up a Tomcat Server on EC2 instance to be accessed via port 8080 and a separate server to run Jenkins
- Then integrated Tomcat with Jenkins by installing “Deploy to container and configuring tomcat server with credentials
- I also installed maven and integrated it with Jenkins to build the source code

Maven installations

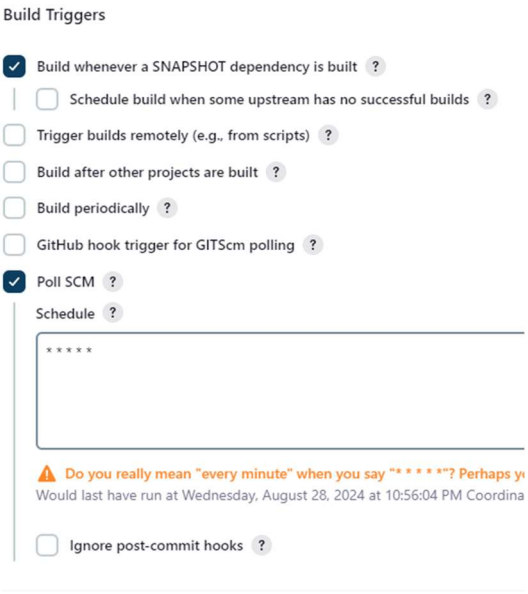


-The war artifact has been successfully deployed on the Tomcat server and displays the contents as such :

```
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[INFO]
[INFO] --- war:3.3.2:war (default-war) @ webapp ---
[INFO] Packaging webapp
[INFO] Assembling webapp [webapp] in [/var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/BuildAndDeployJob/webapp/src/main/webapp]
[INFO] Building war: /var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp.war
[INFO]
[INFO] --- install:3.1.2:install (default-install) @ webapp ---
[INFO] Installing /var/lib/jenkins/workspace/BuildAndDeployJob/webapp/pom.xml to /var/lib/jenkins/.m2/repository/com/example/maven-project/webapp/1.0-SNAPSHO
[INFO] Installing /var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp.war to /var/lib/jenkins/.m2/repository/com/example/maven-project/webapp/1
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for Maven Project 1.0-SNAPSHOT:
[INFO]
[INFO] Maven Project ..................................... SUCCESS [  1.347 s]
[INFO] Server ............................................ SUCCESS [  6.882 s]
[INFO] Webapp ............................................ SUCCESS [  2.236 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  12.877 s
[INFO] Finished at: 2024-08-28T22:24:44Z
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployJob/webapp/pom.xml to com.example.maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp.war to com.example.maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHO
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployJob/server/pom.xml to com.example.maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployJob/server/target/server.jar to com.example.maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHO
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployJob/pom.xml to com.example.maven-project/maven-project/1.0-SNAPSHOT/maven-project-1.0-SNAPSHOT.p
channel stopped
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp.war to container Tomcat 8.x Remote with context null
  [/var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp.war] is not deployed. Doing a fresh deployment.
  Deploying [/var/lib/jenkins/workspace/BuildAndDeployJob/webapp/target/webapp.war]
Finished: SUCCESS
```

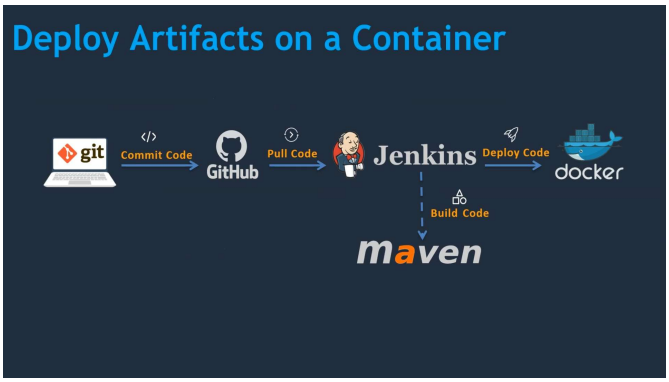


- To automate the pulling of the code, I implemented poll SCM as a build trigger to check the source code by the minute



STEP 2

In this Section of the project, I will be deploying the source code on a Docker container

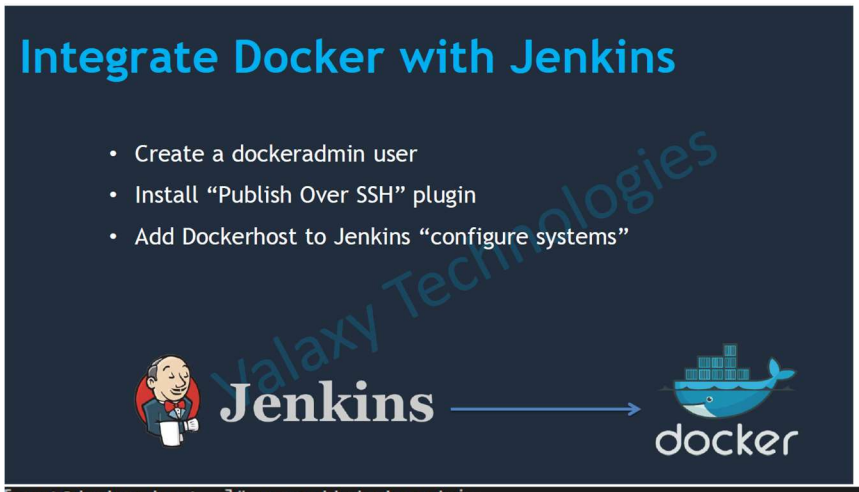


- I downloaded Docker on my “docker-host’’ server and pulled the Tomcat image from Docker hub

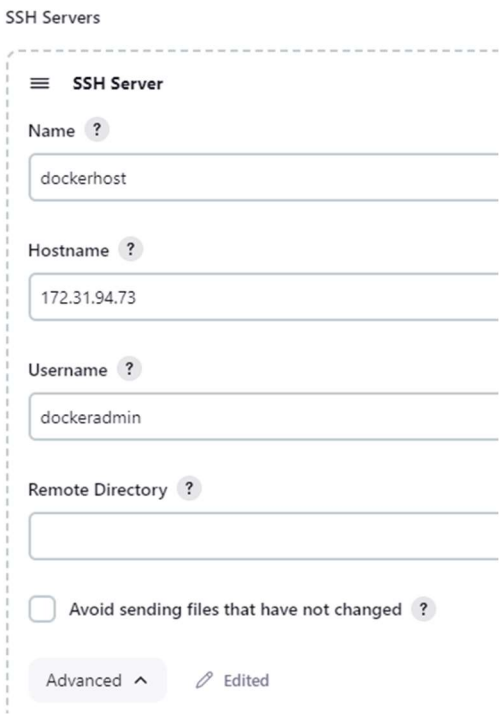
- Then I created a Docker container running on port 8080 and exposed on 8081 which I opened in my security group inbound rules
- However, I encountered an error when I tried to access tomcat from the public IP of my instance via port 8081 – “HTTP STATUS 404 – NOT FOUND”
- This error can be fixed by copying the files from webapp.dist in the container to webapps
- So, I wrote a simple Dockerfile to execute this and build images from

```
FROM tomcat:latest
RUN cp -R  /usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps
~
~
~
~
~
```

Integrating Docker with Jenkins




```
[root@docker-host ~]# useradd dockeradmin
[root@docker-host ~]# passwd dockeradmin
Changing password for user dockeradmin.
New password:
BAD PASSWORD: The password fails the dictionary check - it is based on a dictionary word
Retype new password:
passwd: all authentication tokens updated successfully.
[root@docker-host ~]# usermod -aG docker dockeradmin
[root@docker-host ~]# id dockeradmin
uid=1001(dockeradmin) gid=1001(dockeradmin) groups=1001(dockeradmin),992(docker)
[root@docker-host ~]#
```



- I added “dockerhost” configurations to my Jenkins SSH servers

- Then I updated the Dockerfile to automate the deployment process by copying the artifact containing the war file to the Docker container

```
[root@docker-host docker]# cat Dockerfile
FROM tomcat:latest
RUN cp -R  /usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps
COPY ./*.war /usr/local/tomcat/webapps
[root@docker-host docker]#
```

```
Start a build
[root@docker-host docker]# docker build -t tomcat:v1 .
[+] Building 0.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 167B
 => [internal] load metadata for docker.io/library/tomcat:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/tomcat:latest
 => [internal] load build context
 => => transferring context: 2.40kB
 => [2/3] RUN cp -R  /usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps
 => [3/3] COPY ./*.war /usr/local/tomcat/webapps
 => exporting to image
 => => exporting layers
 => => writing image sha256:4c59f5a8bf6c5e61e08d3f625087cc18eb4bbadc5a549770f4076aa9b967ef0b
 => => naming to docker.io/library/tomcat:v1
[root@docker-host docker]#
```

- Once the artifact has been copied, Jenkins job will create an image and a container by executing Docker commands

Transfer Set

Source files

webapp/target/*.war

Remove prefix

webapp/target

Remote directory

//opt//docker

Exec command

```
cd /opt/docker;
docker build -t regapp:v1 .;
docker run -d --name registerapp -p 8087:8080 regapp:v1
```

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

Advanced

```
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[INFO]
[INFO] --- war:3.3.2:war (default-war) @ webapp ---
[INFO] Packaging webapp
[INFO] Assembling webapp [webapp] in [/var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/target/webapp]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/src/main/webapp]
[INFO] Building war: /var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/target/webapp.war
[INFO]
[INFO] --- install:3.1.2:install (default-install) @ webapp ---
[INFO] Installing /var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/pom.xml to /var/lib/jenkins/.m2/repository/com/example/maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.pom
[INFO] Installing /var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/target/webapp.war to /var/lib/jenkins/.m2/repository/com/example/maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.war
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for Maven Project 1.0-SNAPSHOT:
[INFO]
[INFO] Maven Project ..................................... SUCCESS [  1.431 s]
[INFO] Server ............................................ SUCCESS [  6.016 s]
[INFO] Webapp ............................................ SUCCESS [  2.469 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  13.057 s
[INFO] Finished at: 2024-08-29T03:45:08Z
[INFO] ------------------------------------------------------------------------
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/pom.xml to com.example.maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployOnContainer/webapp/target/webapp.war to com.example.maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.war
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployOnContainer/server/pom.xml to com.example.maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployOnContainer/server/target/server.jar to com.example.maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.jar
[JENKINS] Archiving /var/lib/jenkins/workspace/BuildAndDeployOnContainer/pom.xml to com.example.maven-project/maven-project/1.0-SNAPSHOT/maven-project-1.0-SNAPSHOT.pom
channel stopped
SSH: Connecting from host [jenkins-server]
SSH: Connecting with configuration [dockerhost] ...
SSH: EXEC: completed after 1,285 ms
SSH: Disconnecting configuration [dockerhost] ...
SSH: Transferred 1 file(s)
Finished: SUCCESS
```

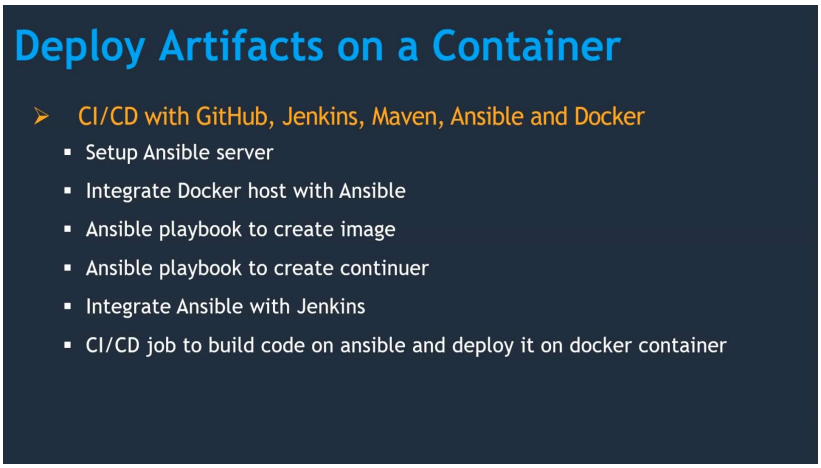
N.B; However, I noticed that when changes are made to the source code by commit, the build fails in Jenkins because a container with the same name exists, so I modified the executable commands to remove the old ones before creating a new container. Therefore, successfully automating the build and deployment on a Docker container

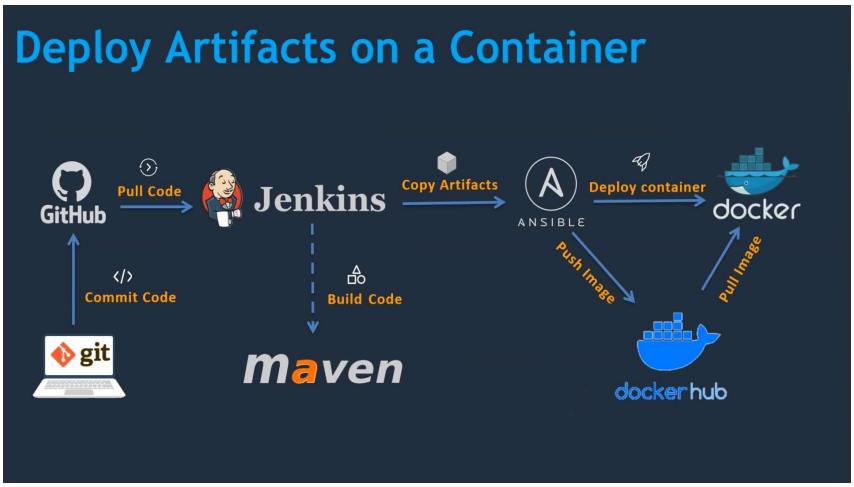
Exec command

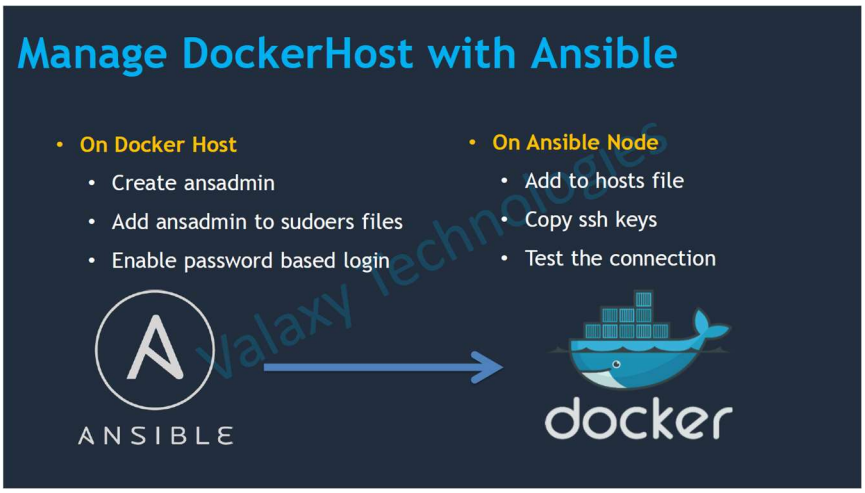
```
cd /opt/docker;
docker build -t regapp:v1 .;
docker stop registerapp;
docker rm registerapp;
docker run -d --name registerapp -p 8087:8080 regapp:v1
```

PART 3 - DEPLOY ON A CONTAINER WITH ANSIBLE

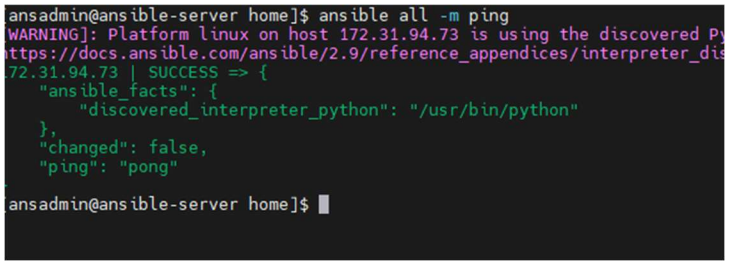
In this part, I will implement Ansible as a deployment tool to execute commands and configure my target environment in a more efficient way



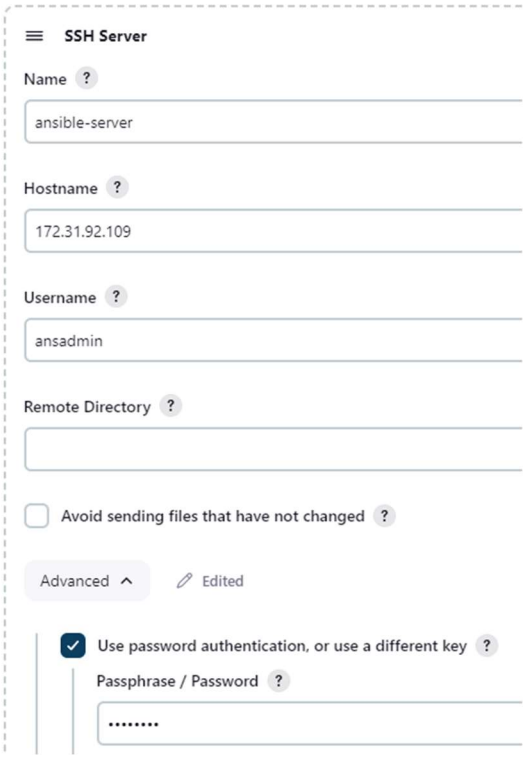




- After copying my ssh keys to my Docker host I successfully connected to it from the Ansible server via the private IP; Now Ansible can communicate with the docker host



- Then I integrated Ansible with Jenkins



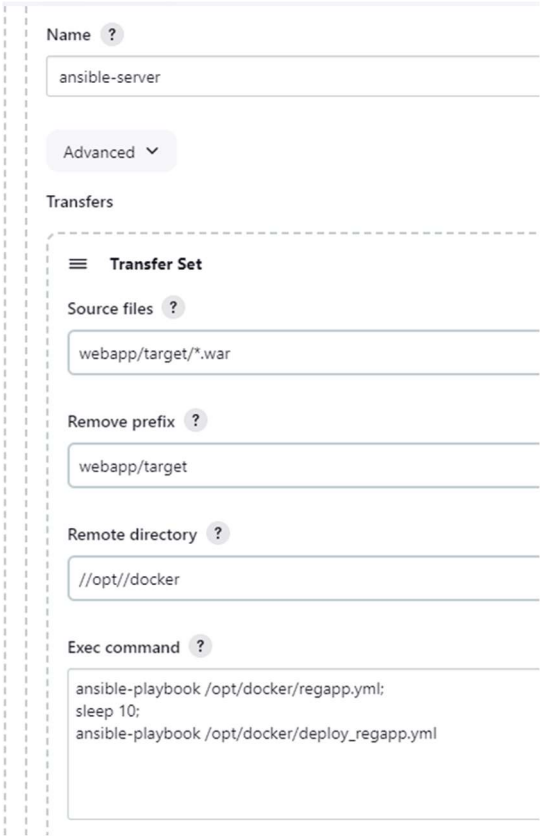
- Next, I wrote an Ansible playbook to create and build a Docker image as well as push it to my Docker hub repository

- The Jenkins server will execute the Ansible playbook whenever changes are made to the source code

- And then another playbook to enable Docker host to pull the image from Docker hub and create a container out of it

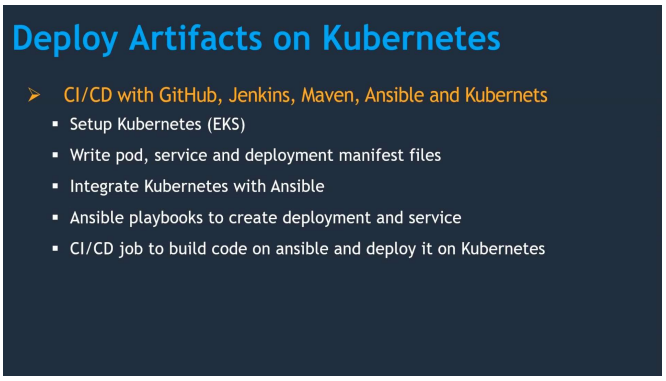
```yaml
---
- hosts: ansible

  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 farouksholanke/regapp:latest

  - name: push docker image
    command: docker push farouksholanke/regapp:latest
~
~
~
~
```

```yaml
---
- 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 farouksholanke/regapp:latest
    ignore_errors: yes

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



PART 3 – DEPLOY ARTIFACTS ON KUBERNETES (EKS)

In the previous step, the existing containers are being terminated when changes are made to the source code before a new image is built and container created; This results in a downtime and our end users cannot access the webapp during that time. To solve this problem; In this next step, I am going to implement Kubernetes container management system for high availability and fault tolerance using Amazon EKS



- I created an eks_ctl role with EC2Full Access and AWSCloud FormationFull Access permissions
- Then attached this role to my EKS server.
- I created an EKS cluster in US-east-1 with 2 nodes

```
root@EKS-bootstrap tmp]# eksctl create cluster --name santi  \
 --region us-east-1 \
 --node-type t2.small \
[
2024-08-30 15:38:58 [ℹ]  eksctl version 0.189.0
2024-08-30 15:38:58 [ℹ]  using region us-east-1
2024-08-30 15:38:58 [ℹ]  setting availability zones to [us-east-1f us-east-1b]
2024-08-30 15:38:58 [ℹ]  subnets for us-east-1f - public:192.168.0.0/19 private:192.16
8.64.0/19
2024-08-30 15:38:58 [ℹ]  subnets for us-east-1b - public:192.168.32.0/19 private:192.1
68.96.0/19
2024-08-30 15:38:58 [ℹ]  nodegroup "ng-6c679873" will use "" [AmazonLinux2/1.30]
2024-08-30 15:38:58 [ℹ]  using Kubernetes version 1.30
2024-08-30 15:38:58 [ℹ]  creating EKS cluster "santi" in "us-east-1" region with manag
ed nodes
2024-08-30 15:38:58 [ℹ]  will create 2 separate CloudFormation stacks for cluster itse
lf and the initial managed nodegroup
2024-08-30 15:38:58 [ℹ]  if you encounter any issues, check CloudFormation console or
try 'eksctl utils describe-stacks --region=us-east-1 --cluster=santi'
2024-08-30 15:38:58 [ℹ]  Kubernetes API endpoint access will use default of {publicAcc
ess=true, privateAccess=false} for cluster "santi" in "us-east-1"
2024-08-30 15:38:58 [ℹ]  CloudWatch logging will not be enabled for cluster "santi" in
 "us-east-1"
2024-08-30 15:38:58 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging -
-enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-1 --cluster=sa
nti'
2024-08-30 15:38:58 [ℹ]  default addons vpc-cni, kube-proxy, coredns were not specifie
d, will install them as EKS addons
2024-08-30 15:38:58 [ℹ]
2 sequential tasks: { create cluster control plane "santi",
    2 sequential sub-tasks: {
        2 sequential sub-tasks: {
            1 task: { create addons },
            wait for control plane to become ready,
        },
        create managed nodegroup "ng-6c679873",
    }
]
2024-08-30 15:38:58 [ℹ]  building cluster stack "eksctl-santi-cluster"
2024-08-30 15:38:58 [ℹ]  deploying stack "eksctl-santi-cluster"
2024-08-30 15:39:28 [ℹ]  waiting for CloudFormation stack "eksctl-santi-cluster"
```

```
679873"
024-08-30 15:52:51 [ℹ]  waiting for the control plane to become ready
024-08-30 15:52:52 [✔]  saved kubeconfig as "/root/.kube/config"
024-08-30 15:52:52 [ℹ]  no tasks
024-08-30 15:52:52 [✔]  all EKS cluster resources for "santi" have been created
024-08-30 15:52:52 [✔]  created 0 nodegroup(s) in cluster "santi"
024-08-30 15:52:52 [ℹ]  nodegroup "ng-6c679873" has 2 node(s)
024-08-30 15:52:52 [ℹ]  node "ip-192-168-22-96.ec2.internal" is ready
024-08-30 15:52:52 [ℹ]  node "ip-192-168-34-193.ec2.internal" is ready
024-08-30 15:52:52 [ℹ]  waiting for at least 2 node(s) to become ready in "ng-6c67987
"
024-08-30 15:52:52 [ℹ]  nodegroup "ng-6c679873" has 2 node(s)
024-08-30 15:52:52 [ℹ]  node "ip-192-168-22-96.ec2.internal" is ready
024-08-30 15:52:52 [ℹ]  node "ip-192-168-34-193.ec2.internal" is ready
024-08-30 15:52:52 [✔]  created 1 managed nodegroup(s) in cluster "santi"
024-08-30 15:52:53 [ℹ]  kubectl command should work with "/root/.kube/config", try 'k
bectl get nodes'
024-08-30 15:52:53 [✔]  EKS cluster "santi" in "us-east-1" region is ready
root@EKS-bootstrap tmp]#
```

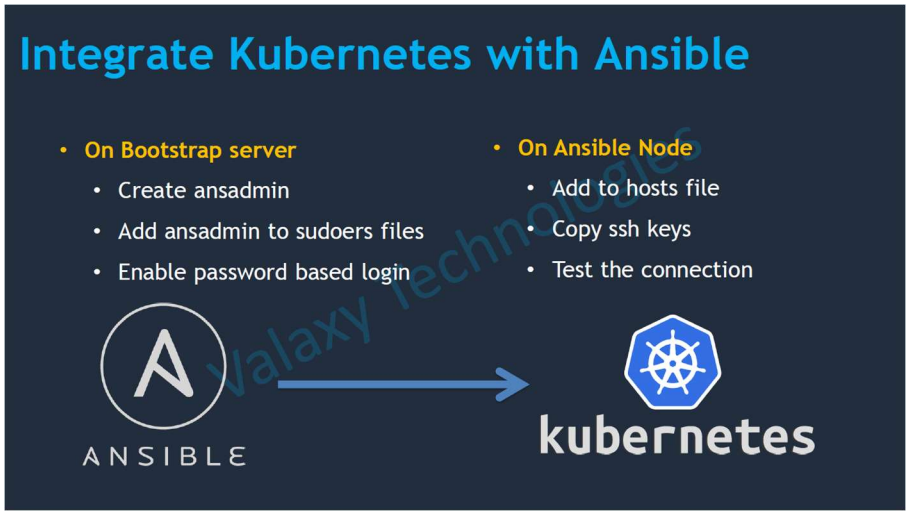
I wrote a deployment file to maintain 3 replicas at all times created from the image pulled from my Docker repository whenever changes are made to the image. The container will be exposed on port 8080. I Also implemented rolling update in the deployment file do delete and recreate the container with the new image one at a time, ensuring high availability and no down time and a service deployment manifest file for a load balancer that's exposed on port 8080 and targets my container from port 8080

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

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

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

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

  ports:
    - port: 8080
      targetPort: 8080

  type: LoadBalancer
~
~
```

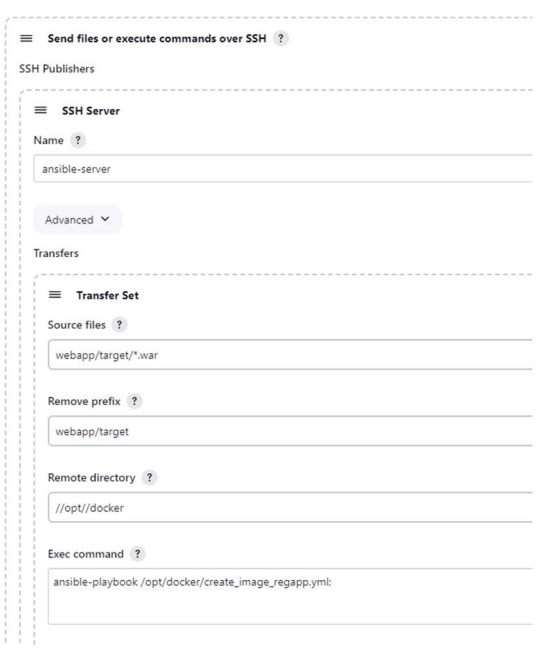
```
[root@EKS-bootstrap ~]# kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/farouk-regapp-689944bfb-jr8zw    1/1     Running   0          100s
pod/farouk-regapp-689944bfb-k592d    1/1     Running   0          100s
pod/farouk-regapp-689944bfb-wxkkq    1/1     Running   0          100s
pod/webapp                           1/1     Running   0          57m

NAME                     TYPE           CLUSTER-IP       EXTERNAL-IP
service/farouk-service   LoadBalancer   10.100.131.187   ae90777cec0c749f4b96dd62
service/kubernetes       ClusterIP      10.100.0.1       <none>

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/farouk-regapp     3/3     3            3           100s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/farouk-regapp-689944bfb    3         3         3       100s
[root@EKS-bootstrap ~]# vi regapp-deployment.yml
[root@EKS-bootstrap ~]# vi regapp-service.yml
[root@EKS-bootstrap ~]#
```

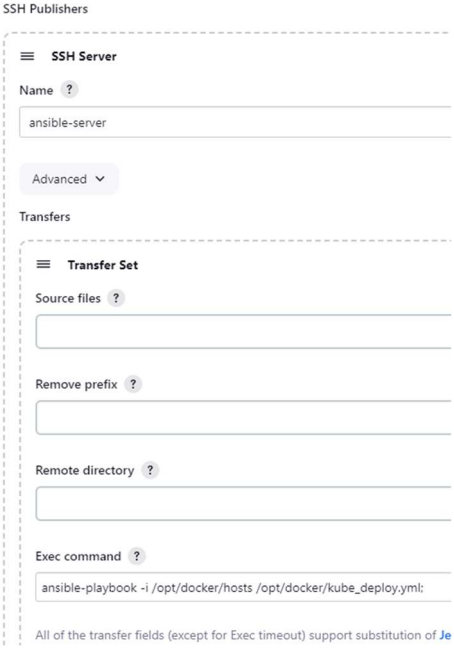
After confirming that my service and deployment files are functioning properly with no errors, I will automate the deployment process by creating an Ansible playbook to execute my deployment and service files and update the deployment with new pods if/when the image is updated in docker hub.




```
---
- hosts: kubernetes
  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.v1.apps/farouk-regapp
```

Then I created a Jenkins continuous integration job to build the code with help of Maven, create an image on Ansible and push it onto Docker hub as well as a continuous deployment job to execute the Ansible playbook. The CI job initiates the CD job after a successful build





In Conclusion, I have now set up a complete CI/CD pipeline that automates the integration and delivery process from when updates to the source code for a web application are committed to GitHub repository