**The problem statements**

A leading US healthcare company, **Aetna**, with a large IT structure had a 12-week release cycle and their business was impacted due to the legacy process. To gain

true business value through faster feature releases, better service quality, and cost optimization, they wanted to adopt agility in their build and release process.

The objective is to implement iterative deployments, continuous innovation, and automated testing through the assistance of the strategy.

**The Business Implementation Requirements**

1. Install and configure the Jenkins architecture on AWS instance
2. Use the required plugins to run the build creation on a containerized platform
3. Create and run the Docker image which will have the application artifacts
4. Execute the automated tests on the created build
5. Create your private repository and push the Docker image into the repository
6. Expose the application on the respective ports so that the user can access the deployed application
7. Remove container stack after completing the job

**Technical Limitations**

The company is only comfortable using the following technology stack now. The implementation team shall use the following tools only in phase 1. Any additional recommendations can be included in a later section.

1. EC2
2. Jenkins
3. Docker
4. Git

**The Design**

This section will show a design diagram for easier understanding of the implementation tasks. In a later section, a preferred implementation diagram is included for future enhancements.

Jumpbox

EC2 instance 1

Jenkins master

docker

EC2 instance 2

Docker registry

Docker registry UI

Github repo

Docker registry

**Implementation Details**

The project’s source code is in the jump box. Before starting the implementation, we will create a git repository and push the code base into the company’s private repo.

Then the jump box is to access the AWS web console. Once you are in AWS, the jump box is not used for any other purposes.

In the AWS web console, the team will create 2 ec2 instances with the same configuration. Both ec2 instances will belong to the same security group and the following configuration will be added to enable the ping capability between ec2 instances.

* Type: Custom ICMP rule
* Protocol: Echo Request
* Port: N/A
* Source: select Anywhere

**Jenkins**

This jenkin’s setup process can be automated, but due to the time limitation, the first phase will not be automated. The steps to install Jenkins in in the appendix section of this document.

Once the Jenkins is installed successfully, configure the setup and add the required plugins to work with docker and configure the plugins.

**Docker**

Docker has its own registry to use. However, due to the nature of the code base, the company must host the docker registry. The steps followed to install the docker registry are outlined in the appendix section.

The ability to access the docker registry via a browser is a huge plus to the company. As the SSL certificates are not available right now, the connection to the private docker registry is not secured. This shall be secured as soon as possible by following docker documentation.

**The inner working**

The jenkin is configured to run with every developer check-in to the code repository. However, Jenkins has the ability to run the job on demand as well. When a user checks in code or initiates a Jenkin process, Jenkins will clone the code base to Jenkins workspace and compile the project. It will automatically build an image and push the image to the local repository.

Jenkin will execute the test suite and publish the results to view. It will also create a docker container and deploy the application in the container.

A user can access the web portal via the ip and the port exposed by the container. Once all checks are done, container stack is removed.

**Future Enhancements**

Jumpbox

Ansible

Jenkins

EC2 instance 1

Jenkins slave

docker

K8s pods

EC2 instance 2

Docker registry

Docker registry UI

K8s admin

Github repo

Docker registry

Using AWS is a costly business. In the ideal world, the Jenkins master should be set up on premise. Using ansible scripts Jenkins can spin up AWS EC2 instances. The Docker instance should have more memory and CPU than the Jenkins slave instance. In the above architecture, the company has the ability to spin up, execute the application and tier it down after completing all necessary tasks. This will be a huge cost saving for the company.

Also, adding Kubernetes to the picture would bring more flexibility to the organization as they will be able to manage their infrastructure more effectively.

**Appendix**

Install Jenkins

ubuntu@ip-172-31-29-135:~$ history

sudo apt install default-jdk

sudo update-alternatives --config java

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

Then add the following entry in your /etc/apt/sources.list:

echo deb https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list

Install Jenkins:

  sudo apt-get update

  sudo apt-get install jenkins

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

**Install docker**

sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl gnupg- agent software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo apt install docker.io

sudo apt install docker-compose

sudo groupadd docker

sudo usermod -aG docker $USER

newgrp docker

**Install local docker registry and registry UI**

version: "3"

services:

docker-registry:

image: registry:2

container\_name: docker-registry

ports:

- 5000:5000

restart: always

volumes:

- ./docker-registry:/var/lib/registry

docker-registry-ui:

image: konradkleine/docker-registry-frontend:v2

container\_name: docker-registry-ui

ports:

- 8081:80

environment:

ENV\_DOCKER\_REGISTRY\_HOST: docker-registry

ENV\_DOCKER\_REGISTRY\_PORT: 5000

create the volume directory before running the next command

docker-compose -f docker-compose.yml up

**On the docker client**

mkdir registry-app

cd registry-app

sudo vi /etc/hosts

add the entry to access the local docker registry

172.31.25.34 ec2-18-234-86-105.compute-1.amazonaws.com

sudo vi /etc/docker/daemon.json

{

"insecure-registries" : ["ec2-18-234-86-105.compute-1.amazonaws.com:5000"]

}

Stop and start the docker service.

sudo service docker stop

sudo service docker start