**Deploying Java Application Using Jenkins, Docker And Kubernetes**

**Keywords**

CI-CD, Kubernetes, Docker, Jenkins, Archiva, Maven, GIT, Automation

**Introduction**

**About DevOps:-**

DevOps is a culture or more like a self-evolving movement which emphasizes on the communication and collaboration of the development team, Quality Assurance (QA) and operations team. It recognizes the interdependency of the teams and helps in automating the process of automatic delivery, continuous integration and deployment. It aims at providing the environment where the software is build, tested (feedback loops are fast) and released frequently and rapidly and also the code is reliable. The tools used in DevOps vary from version control (Git and GitHub) to configuration management (Puppet, Chef, Docker), monitoring and automatic code deployments.

**Continuous Integration:-**

Continuous Integration (CI) is a practice that requires developers to integrate code into a shared repository several times a day. Each commit/check-in is then followed by an automated build which helps in getting a rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as early as possible. Continuous integration software tools like Jenkins can automate the testing and build process of SDLC. Continuous integration also monitors the state of various softwares and gives feedback if it finds any change in it.

Jenkins, Bamboo, Go are few example tools used to create workflows in CI and tools used in version control systems are Git, SVN, Github. Builds tools used are ANT and Maven.

**Continuous Deployment:-**

Continuous Deployment is a software development practice where every code change is sent through the entire pipeline and put in production so that functional testing can be performed in full scale. This results in several deployments per day. In this case ownership of the code from development to production belongs to the developer. The difference between continuous deployment and continuous delivery is that the latter one is always release-ready i.e., to be sent to the user base or to the QA team for continual review and inspection and also the decision to send the code to the production is a business decision and hence it is manual.

The various deployment tools used in DevOps are CA Release Automation (Lisa), Serena Deployment Automation etc.

**Tools used:-**

Version Control System : Git/GitHub

Web Server : Apache Tomcat7

Continuous Integration : Jenkins

Build tool : Maven

Repository Management Software : Archiva

Deployment tool : Docker

Cluster Management Tool : Kubernetes

**GIT:-**

GIT is a very popular and efficient open source Version Control System created by Linus Torvalds in 2005. It takes a "peer-to-peer" approach to version control as opposed to a "Client-Server" approach like version control systems like Subversion (SVN). Github manages your Git project and it has many different forms such as a remote server, a community of developers, and a graphical web interface.

* After the GIT installation you should set your configuration. Use the following commands:

git config --global user.name "Your Name"

git config --global user.email name@yourdomain.com

Commands in GIT:

* git init

Initializes a git repository .Example: cd /home/user/<git repo folder>/ git init

* git add

Adds files changes in your working directory to your index.

* git commit

Takes all of the changes written in the index, creates a new commit object

pointing to it and sets the branch to point to that new commit.

* git pull

Merges files from the remote repository with your local one. This is the single command that combines the use of git fetch and git merge.

* git push

This command pushes the committed changes to remote repository which is the main location of our code.

**JENKINS:-**

Jenkins is one open source tool to perform continuous integration and build automation. The basic functionality of Jenkins is to monitor, execute a predefined list of steps, the execution can be triggered based on time or an event, allows to stop the process if one of the steps fails and send out notification about the build success or failure.

You can download jenkins.war directly and launch it by executing java -jar jenkins.war. The output of this will go to console and not as a service on windows. We can install Jenkins as service by choosing install as windows service in Manage Jenkins.

Now connect to Jenkins by going to the following URL http://<hostname>:8080/.

**MAVEN:-**

Maven is a tool developed in Java used to build Java-based projects. It has the concept of POM files (Project Object Model) which is an XML representation of project resources. Example of resources are source code, test code, dependencies (external JARs used) etc. POM file must be located in the root directory of the project.

Reads pom.xml

Build Life Cycles

* Phases
* Goals

Maven

1. Reads pom.xml
2. Downloads dependencies into local repository
3. Executes life cycles, build phases and/or goals
4. Executes plugins

All executed according to selected build profile

Dependencies (JARS)

Maven Local Repository

Build Plugins

Build Profiles

***POM FILE***

**ARCHIVA:-**

Apache Archiva is an extensible repository management software.

The main advantage of using Archiva is, when a user makes a request for an artifact from Maven, Archiva serves as a central repository artifact instead of Maven's i.e., Archiva collects the archived files and accompanying metadata files from the central repository of Maven and stores these files in its own repository. Whenever developer requests for the artifact, Archiva will provide the artifact to him. Archiva will not ask for the artifacts from the central repository since it already has the resource. Thus, minimizing long-distance network communication.

It is used with build tools such as [Maven](http://maven.apache.org/), [Continuum](http://continuum.apache.org/), and [ANT](http://ant.apache.org/).

**APACHE TOMCAT:-**

Apache Tomcat is an application server developed by Apache Software Foundation. It is used to execute Java servlets and host Web pages that include coding based on JSP. Tomcat requires a Java runtime environment which is either JRE 1.1 or later. Artifacts generated from build process like .war files need to be placed inside the installation directory of Tomcat server. Then the web application can be accessed by starting the service of tomcat.

**DOCKER:-**

Docker is an open source project designed to create, deploy, and run applications by using containers easily. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. Docker benefit both developers and system administrators and fits in the DevOps environment.

**DockerFile:-**

DockerFile is a text document which has all the commands written in it that are used to assemble an image. DockerFile helps in organizing the deployment process from start to finish in an easy read format.

**Syntax of DockerFile:-**

A DockerFile normally consists of comments, commands and arguments written in the format of:

#comment

Command argument

**DockerFile commands:-**

1. FROM

A DockerFile starts with a FROM command which defines the image to be used to start the build process. It is the first command declared in the DockerFile and when the image is not found then docker tries to download it from the docker image index.

Syntax: FROM [image name]

Example: FROM tomcat

1. CMD

CMD is the initial default command which gets executed when a container is instantiated using the image being built. It is used for executing a specific command.

Syntax: CMD application “argument” “argument”

Example: CMD “echo” “Hello docker!”

1. RUN

RUN as similar to CMD is used to executing a specific command but unlike CMD, it is used to build an image.

Syntax: RUN [command]

Example: RUN pip install Flask

1. COPY

The instruction is used to copy files or directories from a source to destination

Syntax: COPY [source] [destination]

Example: COPY hello.py /usr/temp

1. ADD  
   ADD command is used to copy files from the source to the destination if however if the source is an URL then it downloads and copies the contents of the URL and place at destination.

Syntax: ADD [source directory or URL [destination directory]

Example: ADD hello.py /home/hello.py

**Docker commands:-**

1. build

Dockerfile is used to create a docker image. Using docker build command, after going at the location where Dockerfile exists, we create the image.

PATH is the local directory whereas URL is the location of the git repository. The build is run by Docker daemon.

Syntax: docker build [options] PATH | URL | -

Example: docker build .

1. ps

docker ps shows the running containers by default. To show all the containers running use docker ps –a. docker ps command lists the container id, image, command, created, ports etc.

Syntax: docker ps [options]

1. images

docker images command shows all the images present in the local repo along with their repository, tags and their sizes. We can find the list of docker images using the name and tag too.

Syntax: docker images [options] [repository]

Example: docker images java

1. load

Syntax: docker load [options]

Example: docker load < busybox.tar.gz

1. save

docker save produces a tar repository to the standard output stream. It is used to save a non-running container image to a file and to create a backup which is later to be used with docker load.

Syntax: docker save [options] IMAGE [image…]

Example: docker save busybox > busybox.tar

**KUBERNETES:-**

Kubernetes is an open source system developed by Google and was released in July 2015 and is written in Go language. It assures that it is “A system for automating deployment, scaling, and management of containerized application” which means that it helps in managing multiple containers at scale and also helps in scheduling and deploying containers across multiple nodes or cluster.

**Terminology in Kubernetes:-**

* Master: it is the central point which controls the multiple minions and also provide a unified view of the cluster.
* Minions: it is worker nodes which performs the tasks given by the master and provides application specific virtual host.

***Architecture of Kubernetes***

**MASTER**

Docker

Minion

Pod

Pod

Container

Container

Docker

Minion

Pod

Pod

Container

Container

* Pods: it is the application in json that runs on a minion. It is the smallest deployable unit that can be created, managed and scheduled.
* Replication controller: present at master that ensures that requested number of pods are running on minions at all times.
* Service: it is an object on master which defines set of pods and means to access them through IP address and corresponding DNS name.
* Kubelet: it is a service present on a minion which ensures that the containers defined in the pods are started and continue running. Kubelet reads container manifests as YAML files that describe a pod.
* Kubectl: kubectl is a command line tool used for querying and manipulation of the cluster state i.e., kubectl controls the kubernetes cluster manager. It’s operations also include running containers, creating services.

Syntax: kubectl [command] [name] [flags]

Where command specifies the operations like create, get, delete

name is the name of the resource and is case-sensitive

flags is the options to use like -s or--server is to specify the

address and port.

Example: kubectl create -f example.yaml

* Etcd: etcd in general is an open source distributed key-value and etcd takes care of the storing and replicating data across the entire cluster. It is written in Go language and uses Raft protocol.

Read/write access to etcd is given only to kube-api server and etcd is mostly run as a Cluster.

* Flannel: Kubernetes assume that each pod has a unique, routable IP inside the cluster. The advantage of this model is that it reduces the complexity of doing port mapping but the disadvantage is in this way a host gets an entire subnet to itself so flannel plays as a virtual network and creates a overlay mesh network to each server.
* kube-apiserver: The Kubernetes API server services validates and configures data for pods, services, replication controllers, etc. The API server is the only Kubernetes component that connects to etcd and all the other components must go through the API server to work with cluster state.
* Kube-scheduler: When the pods aren’t bound to a node, kube-scheduler assigns the pod to a node and via using a kubelet, the pod gets started. It takes into account the resource and service requirements and the hardware and software constraints and also the workload specific requirements.
* Kube-controller-manager: The controller manager is a service and is responsible for monitoring replication controllers, and creating corresponding pods to achieve the desired state. It uses the API to listen for new controllers and to create/delete pods.

**COMMON KUBECTL COMMANDS USED:**

* kubectl get services // list all the services in the namespace
* kubectl get pods //list all pods name in plain text
* kubectl run :

Syntax: kubectl run NAME --image=image [--port=port] [--replicas=replica]

Where NAME is the image name which is to be rum on the cluster

Example: kubectl run nginx --image=nginx --replicas=1

* kubectl get nodes // returns a list of nodes with name, label and status
* kubectl describe //shows details of specific or group of resources

Syntax: kubectl describe RESOURCE NAME\_PREFIX

Where RESOURCE can be pods, replicationcontrollers, services,nodes

Example: kubectl describe pods/nginx – describes a particular pod

kubectl describe node node\_name – describes a node

* kubectl get rc, services //lists all replication controllers, services
* kubectl create –f FILENAME //creates a resource by filename,

Syntax: kubectl create –f FILENAME

Where FILENAME can be in json and yaml format

Example: kubectl create –f nfinx-rc.yaml – creates a RC from a manifest

* kubectl delete
* syntax: kubectl delete ([-f FILENAME] | TYPE [(NAME | -l label | --all)])
* Example: kubectl delete -f ./pod.json - Delete a pod using the type and name specified in pod.json.

**CREATION OF PODS**

STEP 1: make a directory where we place our pod file in json or yaml format

mkdir pod\_example

cd pod\_example

STEP 2: create the following file node-js-pod.yaml

apiVersion:v1

kind:Pod

metadata:

name: node-js-pod

spec:

containers:

-name:node-js-pod

image: bitnami/apache:latest

ports:

-containerPort: 80

The above file will create a pod with name node-js-pod with the container bitnami/apache: latest on Port 80.

STEP 3: run the kubectl create command on the fle

* kubectl create -f node-js-pod.yaml

STEP 4: to get more information on the pod use the following command:

kubectl describe pods/node-js-pod

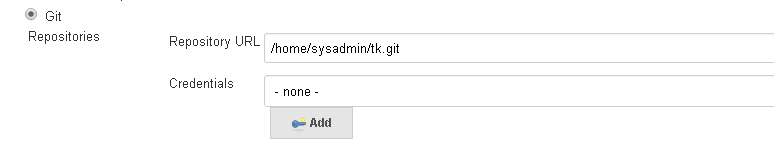
**SOFTWARE SETUP:-**

**CI PART:-**

We use a 64 bit Windows Server VM for the whole continuous integration part and a simple demo application is used which has a login page (username, password) and when details are entered opens a welcome page.

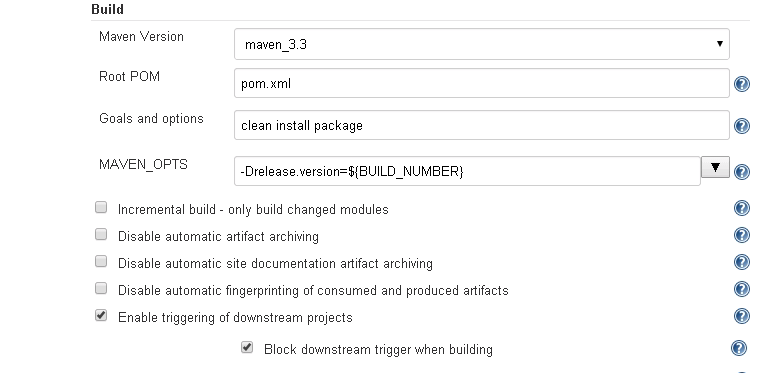
**SETTING UP OF GIT:**

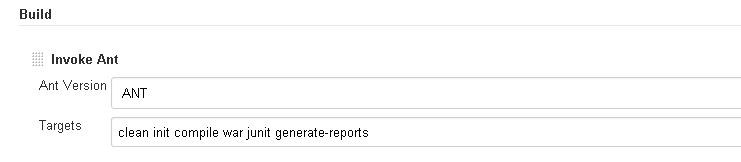
* The git repo required for our work is stored in a VM and the path to the repo is mentioned in the Jenkins job.

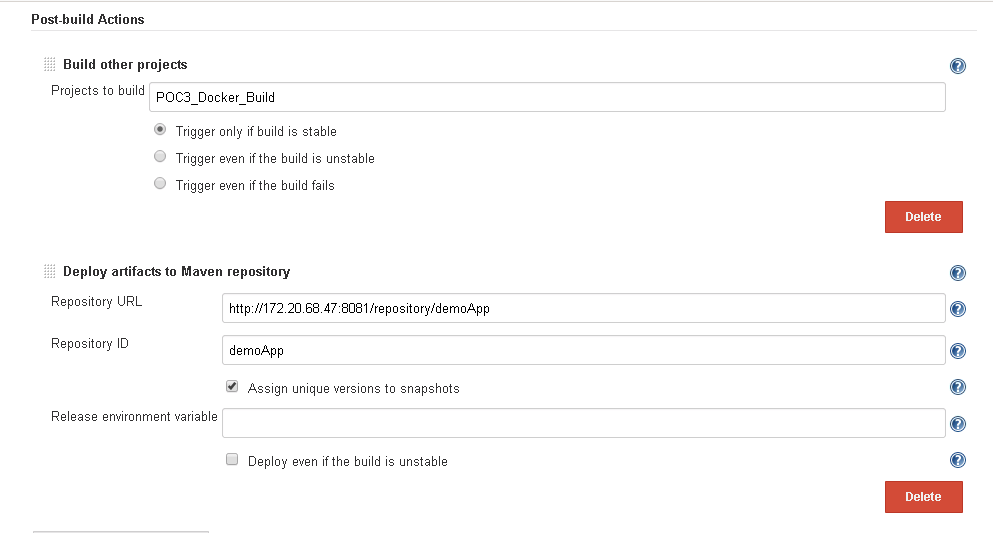


**SETTING UP OF JENKINS:**

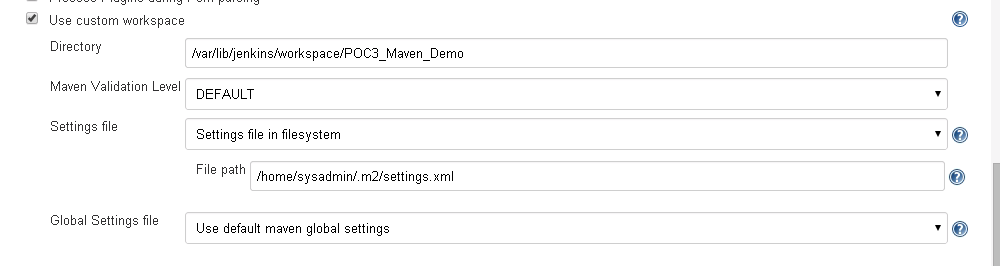
* Open Jenkins in the browser with address as http://172.20.68.33:8080 in the VM.
* Create a new item under Maven Project. We require five jobs for our project, used for Archiva, JUnit, Maven and Docker and Kubernetes jobs. The jobs namely Archica\_Demo, JUnit\_Demo, Maven\_Demo, Docker\_build, Kubernetes.
* The first job to build is Maven\_Demo. As we don’t have an access to internet from the VM, we use local repository. In configure, we mention the git repo and give build trigger as when snapshot dependency is built and maven configure attached as follows. If this build is successful it leads to Junit\_Demo build.



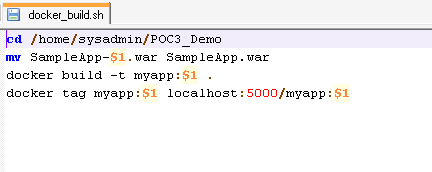
* In JUnit Demo, when given the same git repo, we use ANT build and give series of target to create war file and perform JUnit tests on the application. If the build is successful then it leads to Archiva\_Demo.
* In Archiva\_Demo, we specify the Archiva repo created by us in Post Build action to upload the war file as shown below :



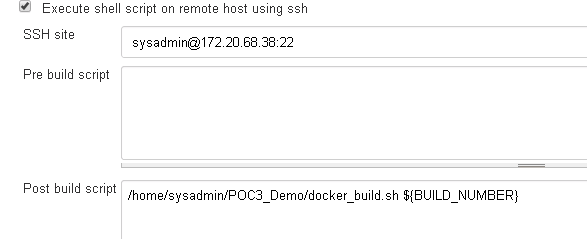
Also we are using the same workspace as of Maven job and providing the path of settings.xml.



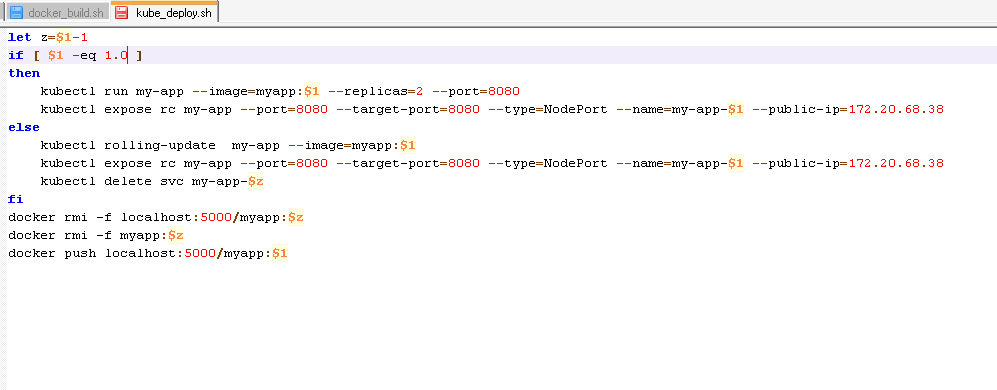
* In Docker\_Build job, docker\_build.sh script is ran. This script creates a docker image based on the Dockerfile and pushes it to the local docker registry. The build number is dynamic and keeps changing every time build is executed. Here is the docker build script :



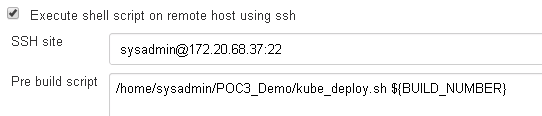
Below is the configuration done in Docker\_Build Jenkins job.



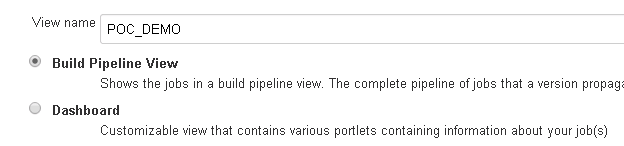
* In Kubernetes Job, we run a kube\_deploy.sh script on the Atomic Host VMs(172.20.68.37) allocated to us, via SSH, which creates the replication controller based on the image provided and exposes it to the outside world using service
* Below is the snapshot of kube\_deploy.sh script:



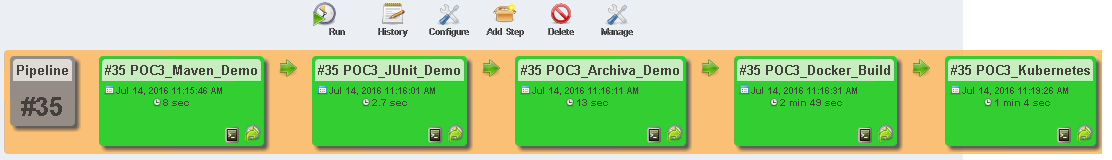
Below is the configuration done in the Kubernetes Jenkins job:



* To build a pipeline, go to Jenkins main page and select + on the tab and select build pipeline view and give view name.



* As we already mentioned in the projects to build in each job in Configure, if a job is successful then it moves onto the next job and pipeline is build. If all jobs are successful then it displays in the below way:



**SETTING UP OF ARCHIVA:**

DOWNLOAD AND INSTALLATION:

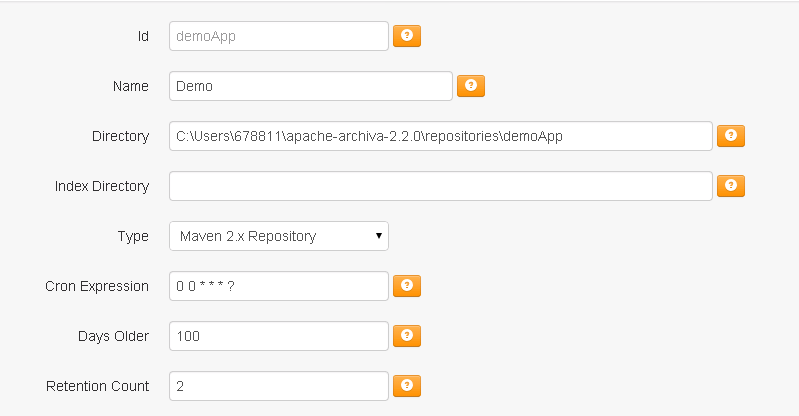
* We can download Archiva from the official website :

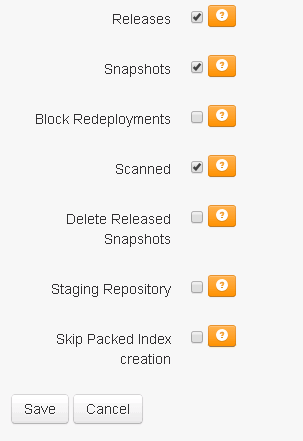
<http://archiva.apache.org/download.cgi>

* Extract the zipped folder and install as a windows service.
* To install as a windows service, go to bin folder and give **archiva install** command.
* To start Archiva using archiva command – give **archiva start** /archiva console in CMD or install as windows service in control panel.
* We can access Archiva in the browser via port - <http://localhost:8081/archiva/>
* Note :- Many a times we have two or more applications running on the same port as 8080 is the default port so to change the port number of Archiva go to **<path>\apache-archiva-1.2.2\conf\jetty.xml configuration file** and change default label in the port tag to desired port number.
* Restart Archiva.

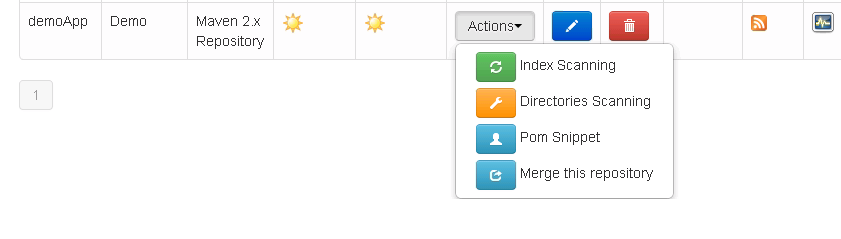
SETTING UP A REPOSITORY IN ARCHIVA:

* Creating a repository: Select Repositories under Administration tab. Select Add tab and fill the new repository details and click Save.



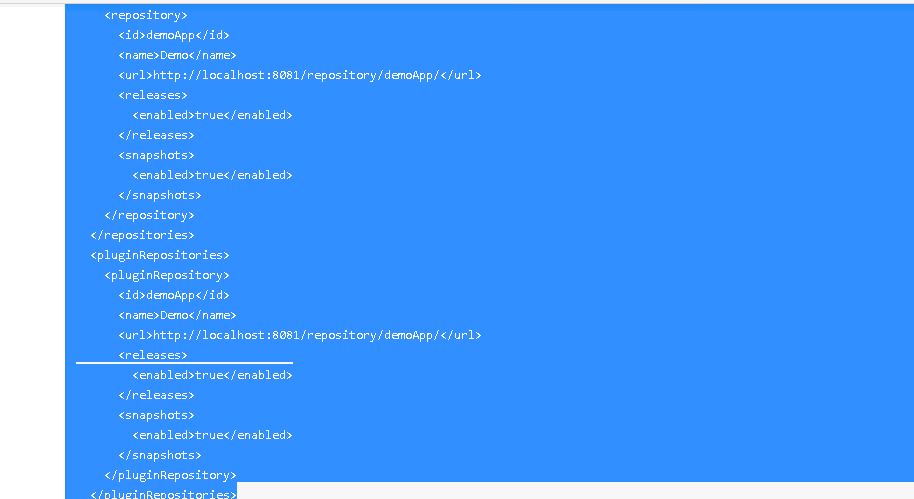


* Go to Managed Repositories and find your repository there. Archiva has some default repositories which can be used as well.

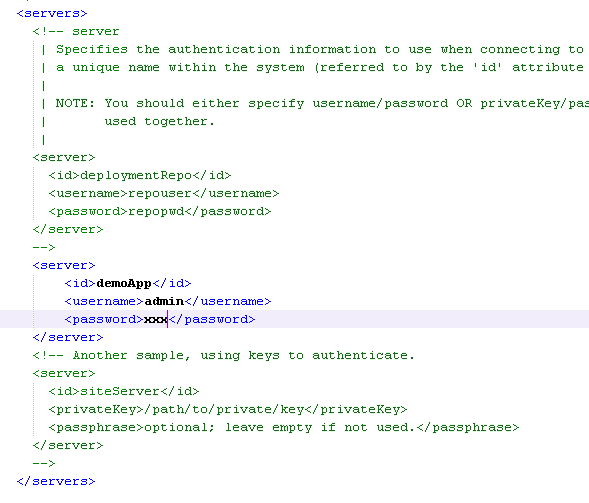


* Next, we need to create a settings file for Maven for reference to check when publishing to the repository. Create a file called **settings.xml** and place it into .m2 folder location we have setup for Maven.
* On the main repositories page, find the option to generate a **pom snippet** for your repository and click it.
* In the XML file, copy everything between <repositories> and

<pluginrepositories> tags and paste it in **pom.xml**.



* In **settings.xml** file add server ID, username and password for authentication to the Archiva server as shown in the screenshot below :



* For the integration of the Archiva with Jenkins, we need to do some configurations. Place the settings.xml file in the Maven settings directory which Jenkins uses (/var/lib/Jenkins/.m2). Go to Manage Jenkins ->Configure System ->Maven Configuration and either give the path to settings file or “select use default maven settings”.

**CD PART:-**

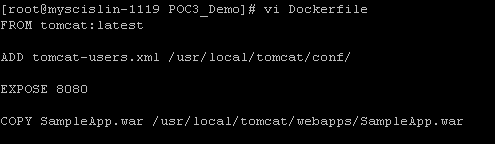
We use two Red Hat Enterprise Linux Atomic Host (RHELAH) VMs in which Docker and Kubernetes are already installed. One VM acts as a Kubernetes master and other acts as a Kubernetes node.

ASSUMPTIONS: As we couldn’t get internet access on the VMs, we downloaded the tomcat image, registry image and gcr.io/google\_containers/pause:2.0 image locally and sent it to our Atomic Host VMs.

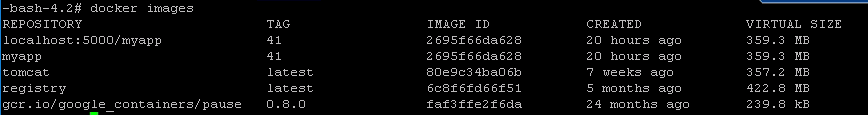
**SETTING UP OF DOCKER:**

CREATION OF DOCKERFILE:

* Dockerfile created for the application start from tomcat as the base image, exposes 8080 port on the container and copies SampleApp.war to the webapps folder in Apache Tomcat. When we perform build on the docker-build.sh file in Jenkins then it creates the image based on the Dockerfile.



* When we run the docker images command, we can see the image of the application running.



**SETTING UP OF KUBERNETES:**

**OVERVIEW:**

Operating System: Red Hat Enterprise Linux Atomic Host 7 (RHELAH)

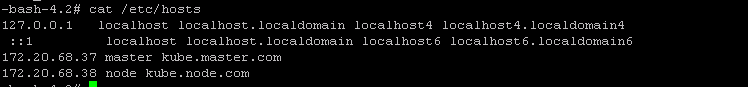
RHELAH is a light-weight and efficient operating system mainly used for the purpose of running Linux containers in the docker format. The main advantage of using RHELAH is that it is pre-installed with Docker and Kubernetes.

**SYSTEM REQUIREMENTS:** RHELAH runs on any computer or cloud environment that supports 64-bit Red Hat Enterprise Linux systems.

PREPARING TO DEPLOY CONTAINER WITH KUBERNETES:

* After the RHELAH 7 installation is done on the systems, we need to configure some services/files such as /etc/hosts, etcd service, Kubernetes config file, flanneld, kubelet file. Firewalld is not installed in RHELAH and hence no need to disable it.
* CONFIGURE DNS OR /etc/hosts file :

As we are using two machines, we need to identify which one is kubernetes master and minion/node and also mention the IP addresses. So we mention the master and the minion details in the hosts file.



SETTING UP OF KUBERNETES ON THE MASTER:

The services which run on master include docker, etcd, flanneld, kube-apiserver, kube-controller-manager, and kube-scheduler. Before we run all these services, we need to set up all on the master.

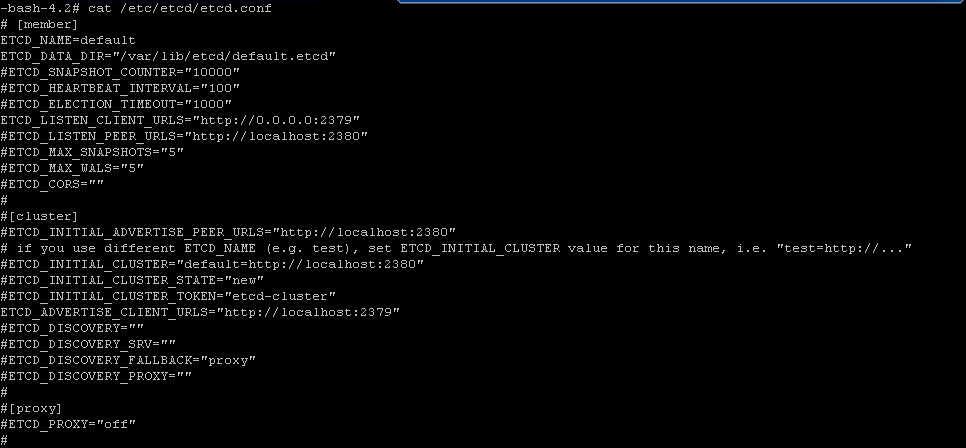
* Configure the etcd service :

etcd file can be accessed at the location /etc/etcd/etcd.conf. Few lines are needed to be un-commented in the file. The labels include ETCD\_NAME,ETCD\_DATA\_DIR,ETCD\_LISTEN\_CLIENT\_URLS, ETCD\_LISTEN\_PEER\_URLS,ETCD\_ADVERTISE\_CLIENT\_URLS. We can observe the port 2379 on which the master runs.

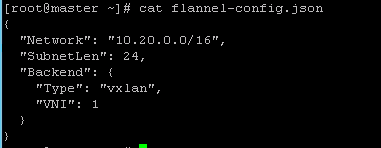
To run the etcd service use the following commands:

systemctl restart etcd

systemctl enable etcd



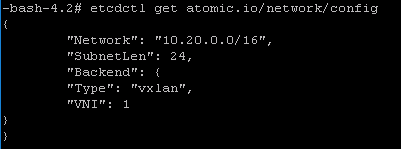
* Configure Flanneld: flannel assigns a range of addresses to be used by all the nodes in the kubernetes environment. To implement the addresses, we need to create a file called flannel-config.json with the following content :



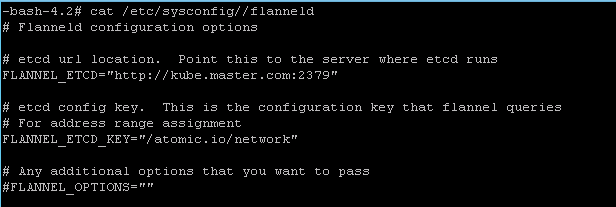
To upload the config file to etcd service, use the following command.

C:\Users\vamshipriya_n\Desktop\poc3 doc\CD-part\CD-part\kube002commandsetflannel.PNG

To check if it is uploaded properly



Go to /etc/sysconfig/flanneld to configure the flannel network. Insert the name/IP address of the master in FLANNEL\_ETCD. It is advisable not to change ETCD\_KEY. When the flannel service starts up, it reads this file to pass to the flanneld daemon.

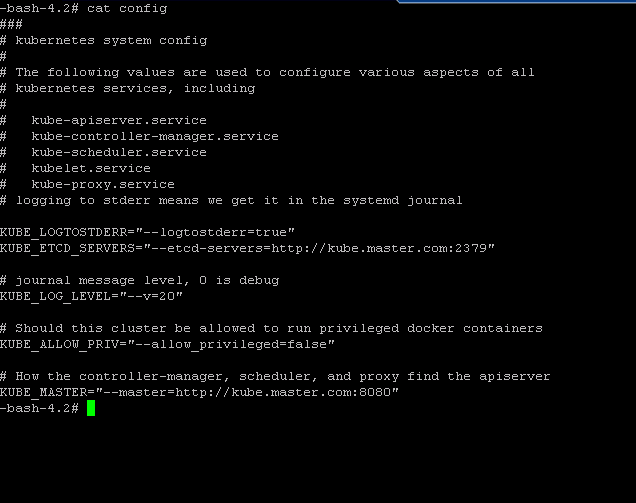


To enable the flannel service, run the following commands from the master.

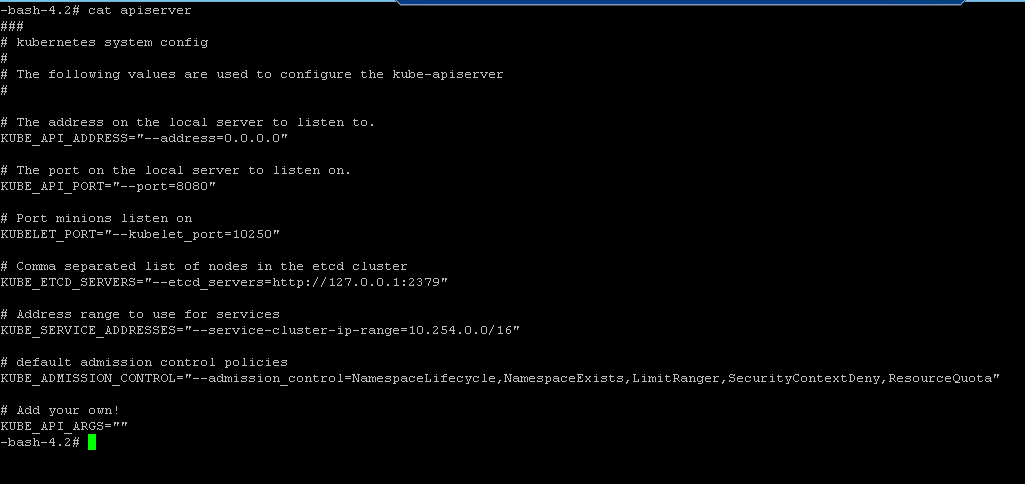
systemctl enable flannel

systemctl reboot

* Edit **/etc/kubernetes/config** file: insert the master’s name in the KUBE\_ETCD\_SERVERS and KUBE\_MASTER.

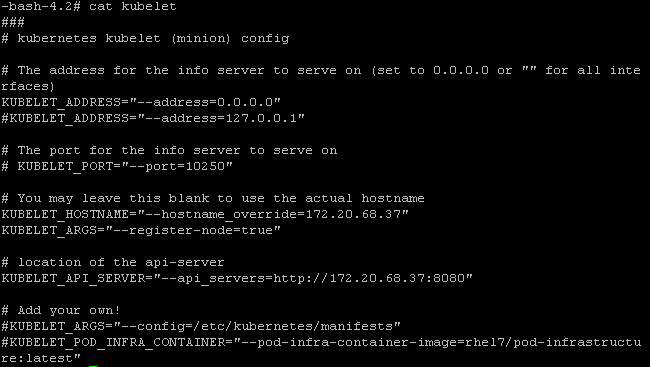


* Edit **/etc/kubernetes/apiserver** file as below:



* Configure Kubelet: go to **/etc/kubernetes/kubelet+** and configure settings in the file. Insert master’s IP address in KUBELET\_HOSTNAME,

KUBELET\_API\_SERVER.



* To run the containerized kubernetes master services, use the following commands :

systemctl start kube-apiserver kube-controller-manager kube-scheduler

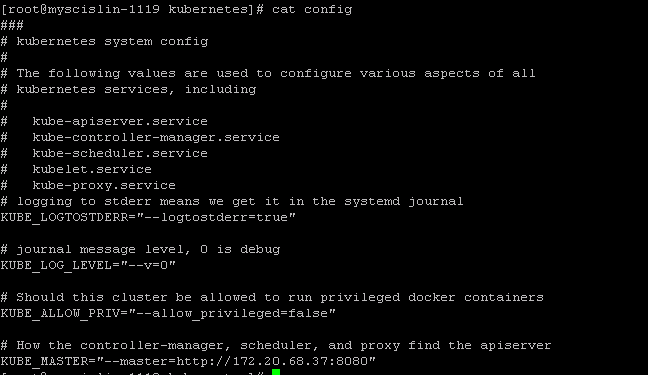
* To start the kubelet service :

systemctl enable kube-proxy kubelet

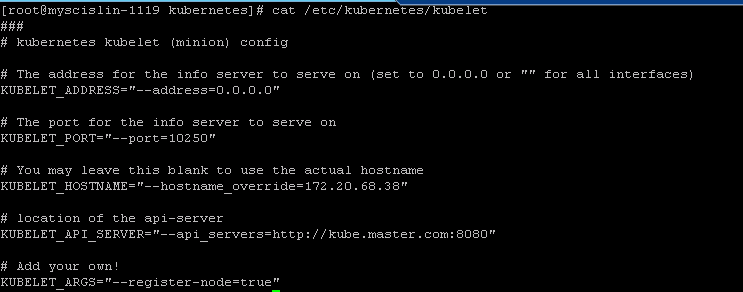
systemctl start kube-proxy kubelet

SETTING UP OF KUBERNETES ON THE NODE:

* Edit **config** file: Go to /etc/kubernetes/config file and provide the master’s name or IP address in KUBE\_MASTER.



* Configure Kubelet: go to **/etc/kubernetes/kubelet** and configure node IP address in KUBELET\_HOSTNAME and master’s name in KUBELET\_API\_SERVER.



To start the node system services, use the following commands:

systemctl restart kube-proxy kubelet

systemctl enable kube-proxy kubelet

* Configure flanneld: go to **/etc/sysconfig/flanneld** and insert master address in FLANNEL\_ETCD.



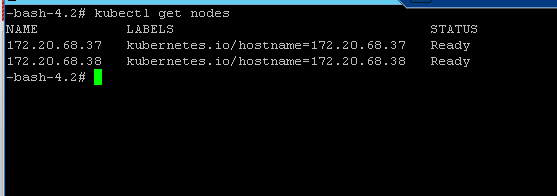
To start the flannel service, run the following commands on each node.

systemctl start flanneld

systemctl enable flanneld

systemctl reboot

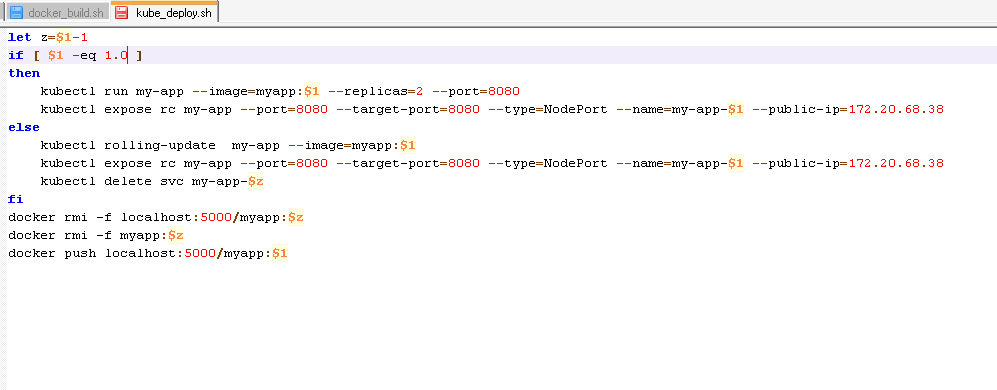
* To check whether the master is communicating with the nodes :



When we configure job in Jenkins, kube\_deploy.sh script runs on the master. The script creates the replication controller and service with number of replicas being 2. We have exposed our replication controller to a given public IP address using nodePort.

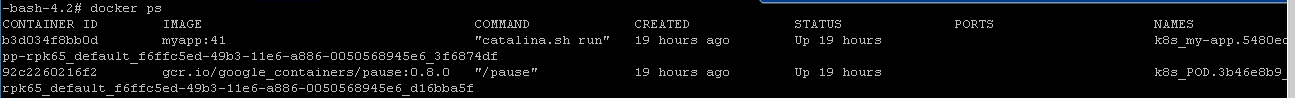
kubectl rolling update: It is used to update replication controller and service based on the updated image or number of replicas. It updates one pod at a time instead of updating the whole service together. It works when we want to increase/decrease the replica count on new/old controllers, create new replication controller with updated configuration. It helps us to achieve **zero downtime,** while upgrading application.

Syntax: kubectl rolling update NAME –image=IMAGE

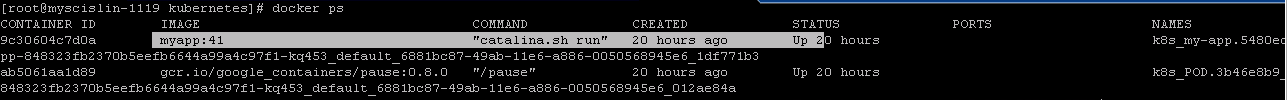


To check whether the application is running on both master and node.

ON MASTER:



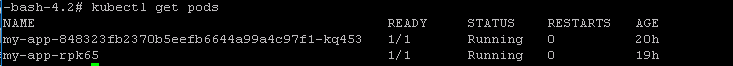
ON NODE:



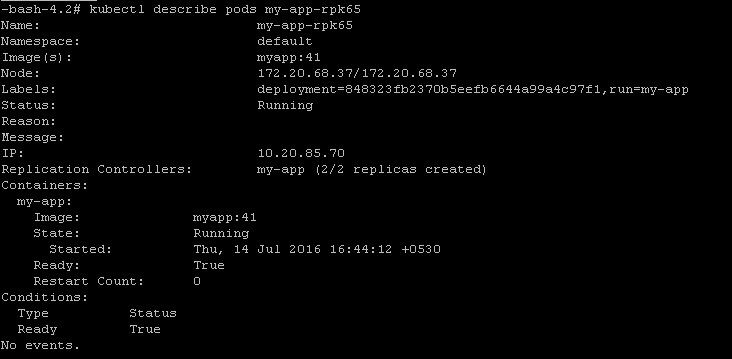
VERIFY THE KUBERNETES CLUSTER:

Run the following commands from the master to check the cluster.

* kubectl get pods



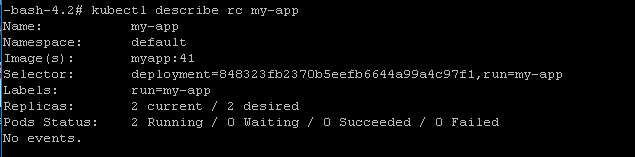
* kubectl describe pods pod\_name



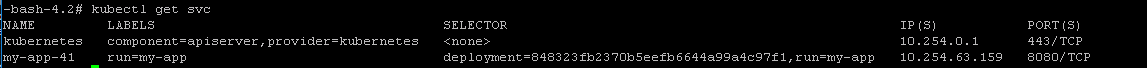
* kubectl get rc

C:\Users\vamshipriya_n\Desktop\poc3 doc\CD-part\CD-part\kubectlgetrc.PNG

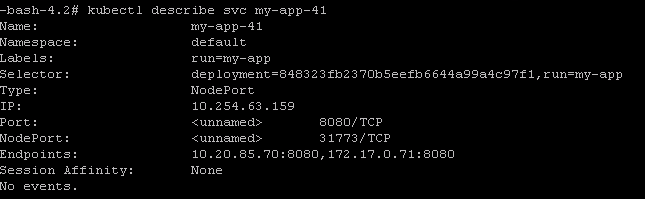
* kubectl describe rc rc\_name



* kubectl get svc

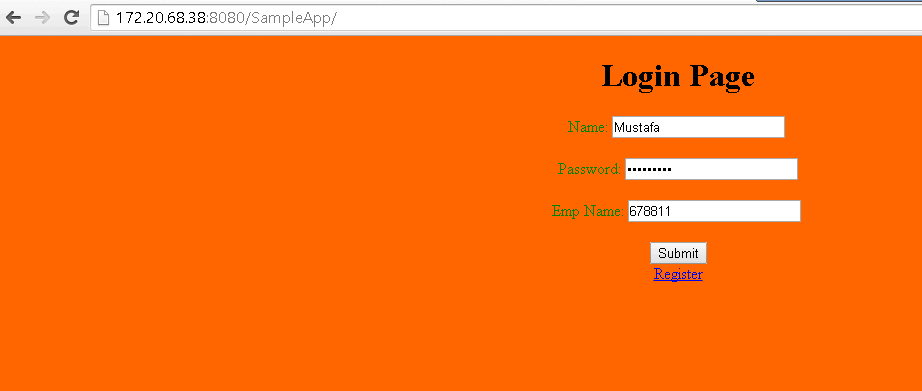


* kubectl describe svc svc\_name

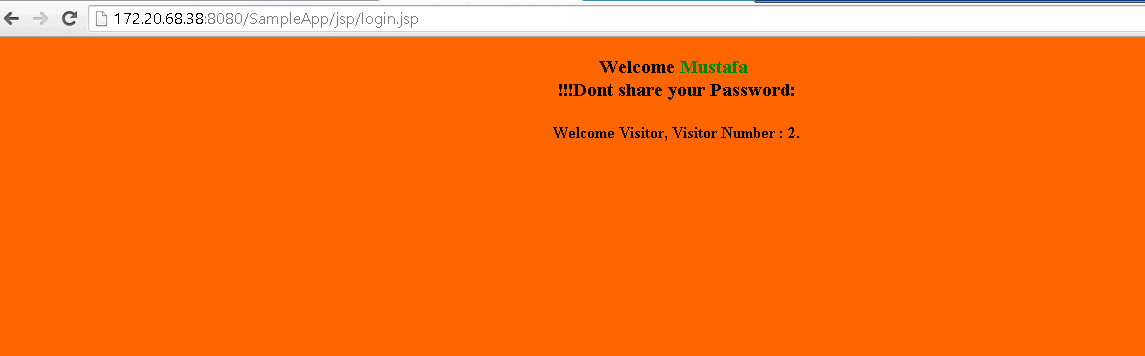


FINAL OUTPUT:

To check the final output, open the application on the browser. We can observe that the IP address is the one we gave in public-ip in kube\_deploy.sh and the port is the target-port.



When we the details, and enter submit, it takes to the following page



**Reference(s)**

* <http://kubernetes.io/docs/user-guide/>
* <https://access.redhat.com/documentation/en/red-hat-enterprise-linux-atomic-host/7/getting-started-with-containers/chapter-4-creating-a-kubernetes-cluster-to-run-docker-formatted-container-images>
* <http://severalnines.com/blog/installing-kubernetes-cluster-minions-centos7-manage-pods-services>
* <https://docs.docker.com/engine/reference/builder/>
* <http://evertrue.github.io/blog/2014/07/21/setting-up-an-archiva-repository/>