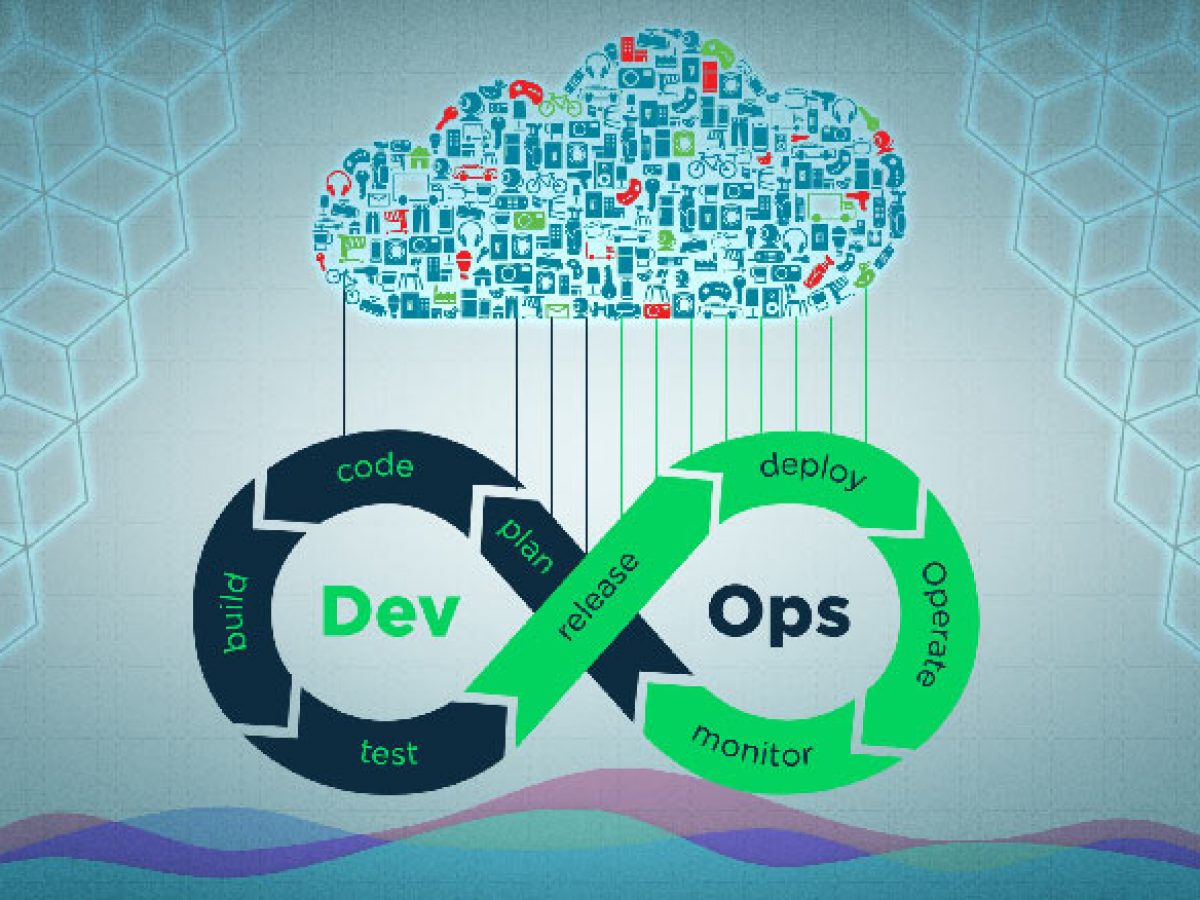
# DevOps Task 4-:CI/CD Pipeline with Dynamic Distributed Jenkins Cluster



**Step - 1** Create a Dockerfile that has Apache Web Server installed and push it onto Github.

Also push the code that needs to be deployed on that server. A demo Dockerfile is :

FROM centos:latest

RUN yum install sudo -y

RUN yum install /sbin/service -y

RUN yum install httpd -y

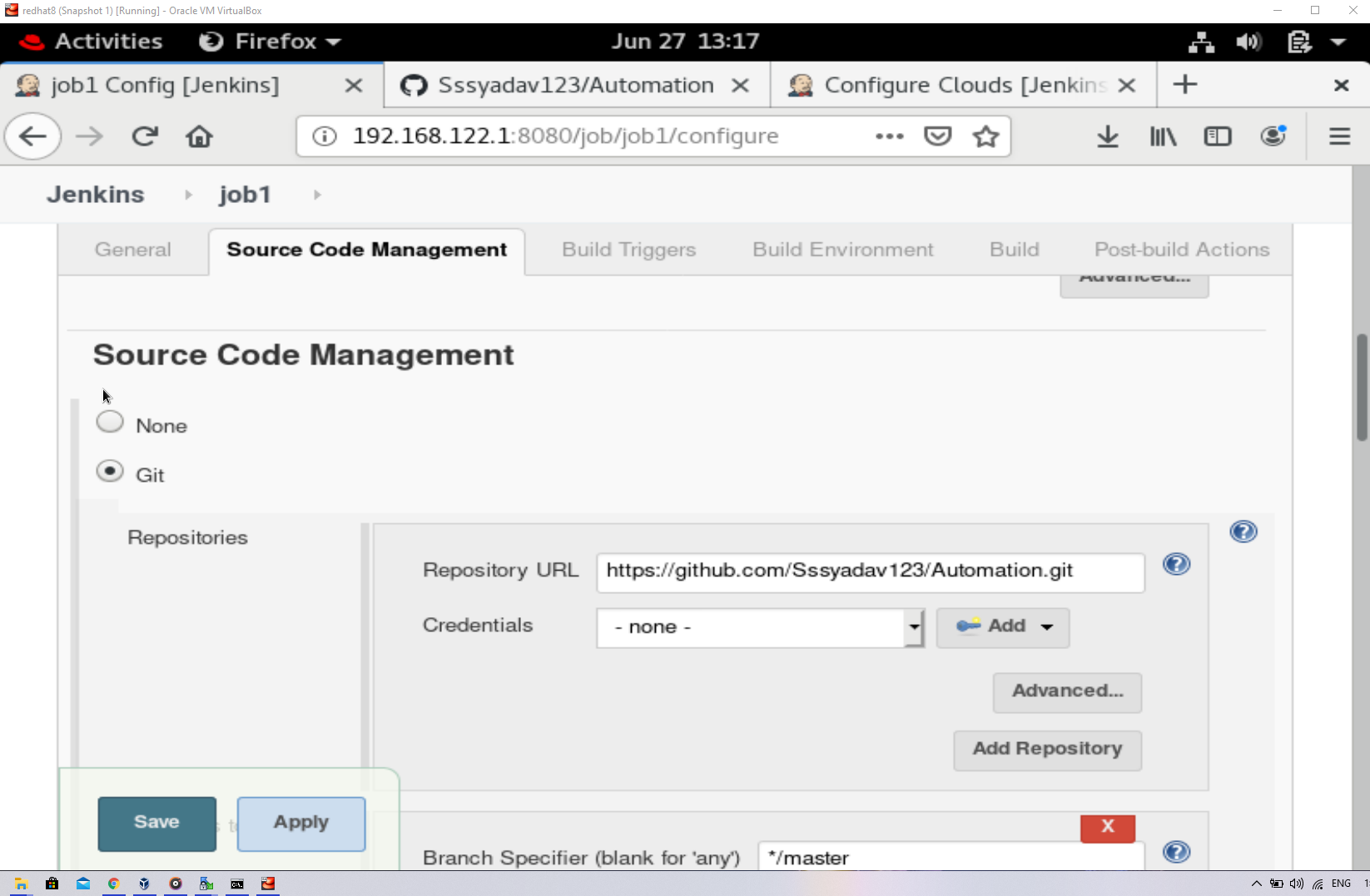
COPY \*.html /var/www/html

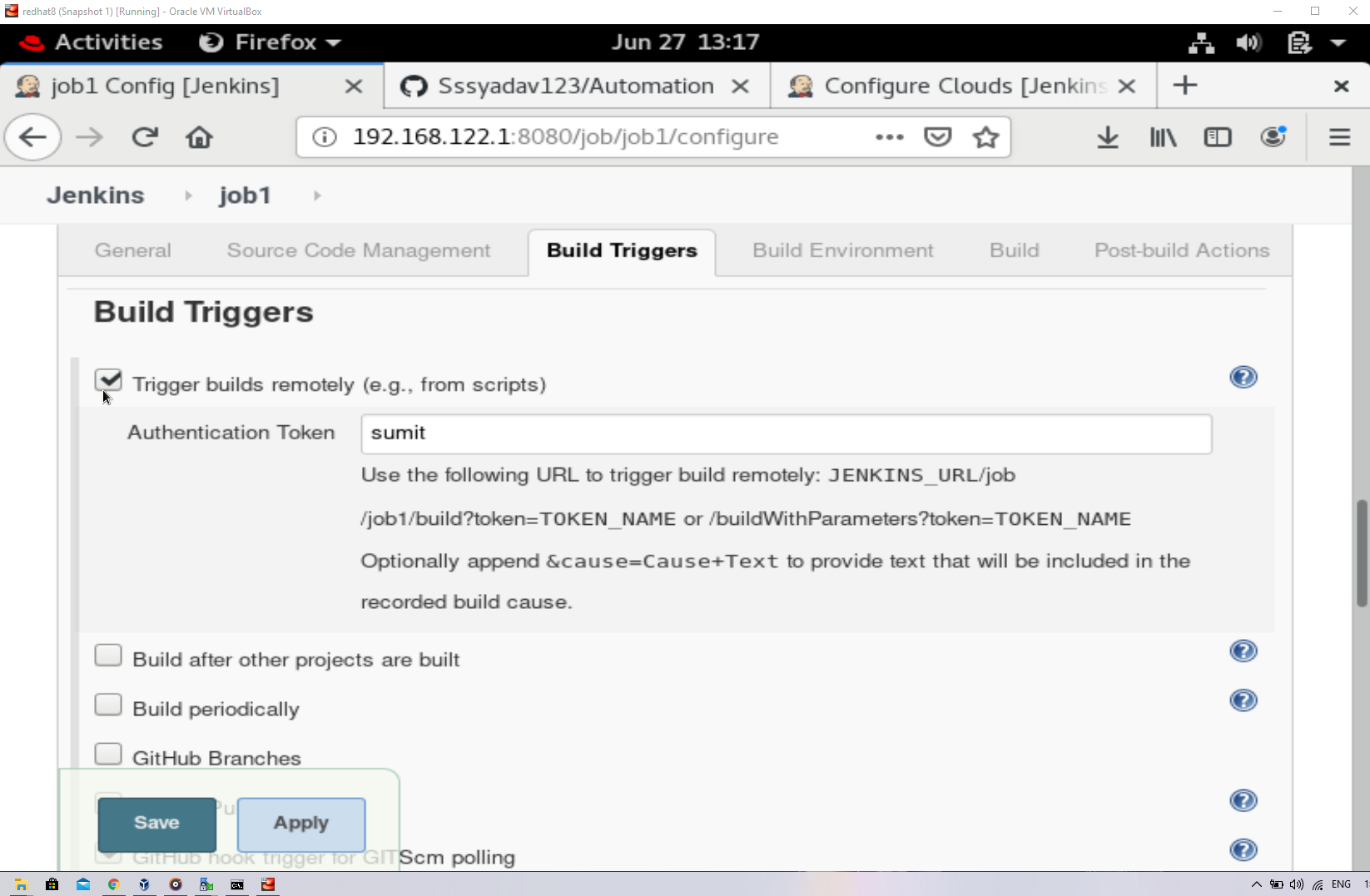
CMD /usr/sbin/httpd -DFOREGROUND && /bin/bash

EXPOSE 80

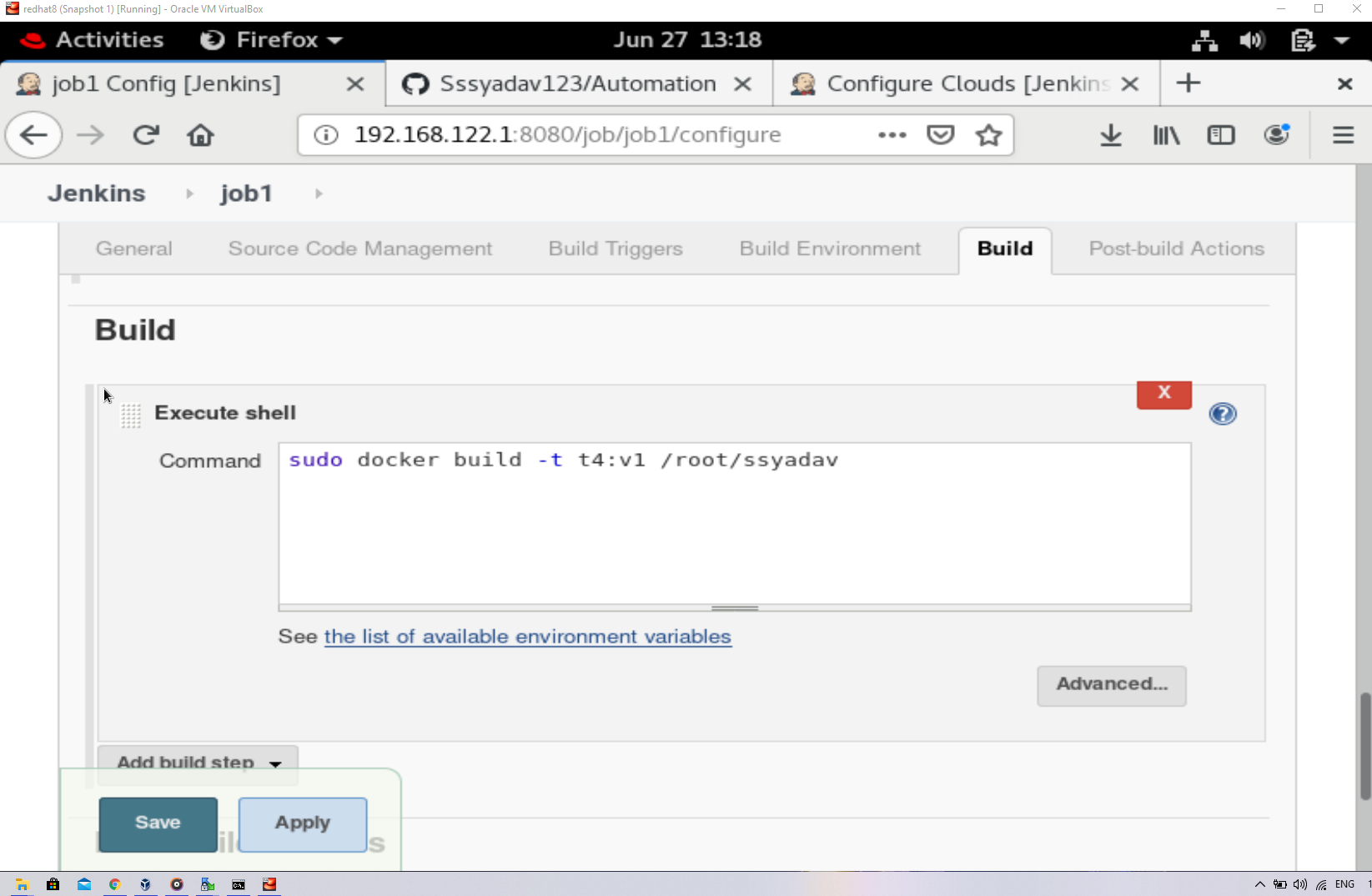
**Step - 2** Now, a Jenkins task needs to be configured to auto download the Dockerfile &

the code whenever they are pushed onto Github.





**Step - 3** Next Jenkins task will build an image from this Dockerfile. This image would have all the code that needs to be deployed. This image will also be automatically pushed to Github.



**Step - 4** Now, we need to create a container that has Kubectl pre-configured, so that we can use it as a dynamic slave to deploy the server image using Kubernetes.

On the official Kubectl website, a yum repo is mentioned that can be used to download and configure Kubectl in any linux container. I have used that same repo file.

***kube.repo***

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

Store this repo in any folder in your Redhat. In the same folder, create a Dockerfile using the following specifications:

(Note - You should also have the reqd. ca.crt, client.crt, client.key in the same folder. If you don't have, then copy it before building this Dockerfile)

FROM centos:latest

COPY ./kube.repo /etc/yum.repos.d/

RUN yum install openssh-server -y

Run yum install java -y

RUN yum install kubectl -y

RUN yum install git -y

RUN yum install sudo -y

COPY ca.crt /root/

COPY client.crt /root/

COPY client.key /root/

COPY Info /root/.kube/

RUN ssh-keygen -A

CMD ["/usr/sbin/sshd", "-D"] && /bin/bash

Also, you need to have your kubectl configuration file in the same folder. Here, I have named it as

***Info***

apiVersion: v1

kind: Config

clusters:

- cluster:

server: https://192.168.99.104:8443

certificate-authority: /root/ca.crt

name: mycl

contexts:

- context:

cluster: mycl

user: sumit

users:

- name: sumit

user:

client-key: /root/client.key

client-certificate: /root/client.crt

When you have done all this configuration properly, you need to build this Dockerfile :

docker build -t <name> /<location of Dockerfile>

\*\* Step - 5\*\* Now, you need to make some changes in Docker Service so that it can be used remotely using ssh. By default, this option is not enabled in Docker. So open the file **/usr/lib/systemd/system/docker.service** in any text editor & make the following changes in the line that I have highlighted:

[Unit]

Description=Docker Application Container Engine

Documentation=https://docs.docker.com

BindsTo=containerd.service

After=network-online.target firewalld.service

Wants=network-online.target

Requires=docker.socket

[Service]

Type=notify

# the default is not to use systemd for cgroups because the delegate issues still

# exists and systemd currently does not support the cgroup feature set required

# for containers run by docker

\*\*\* ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:4243 \*\*\*\*\*

ExecReload=/bin/kill -s HUP $MAINPID

TimeoutSec=0

RestartSec=2

Restart=always

# Note that StartLimit\* options were moved from "Service" to "Unit" in systemd 229.

@@@

"/usr/lib/systemd/system/docker.service" 47L, 1642C 14,57 Top

Now, after making changes in this file, reload your daemon services & restart your docker using the following commands:

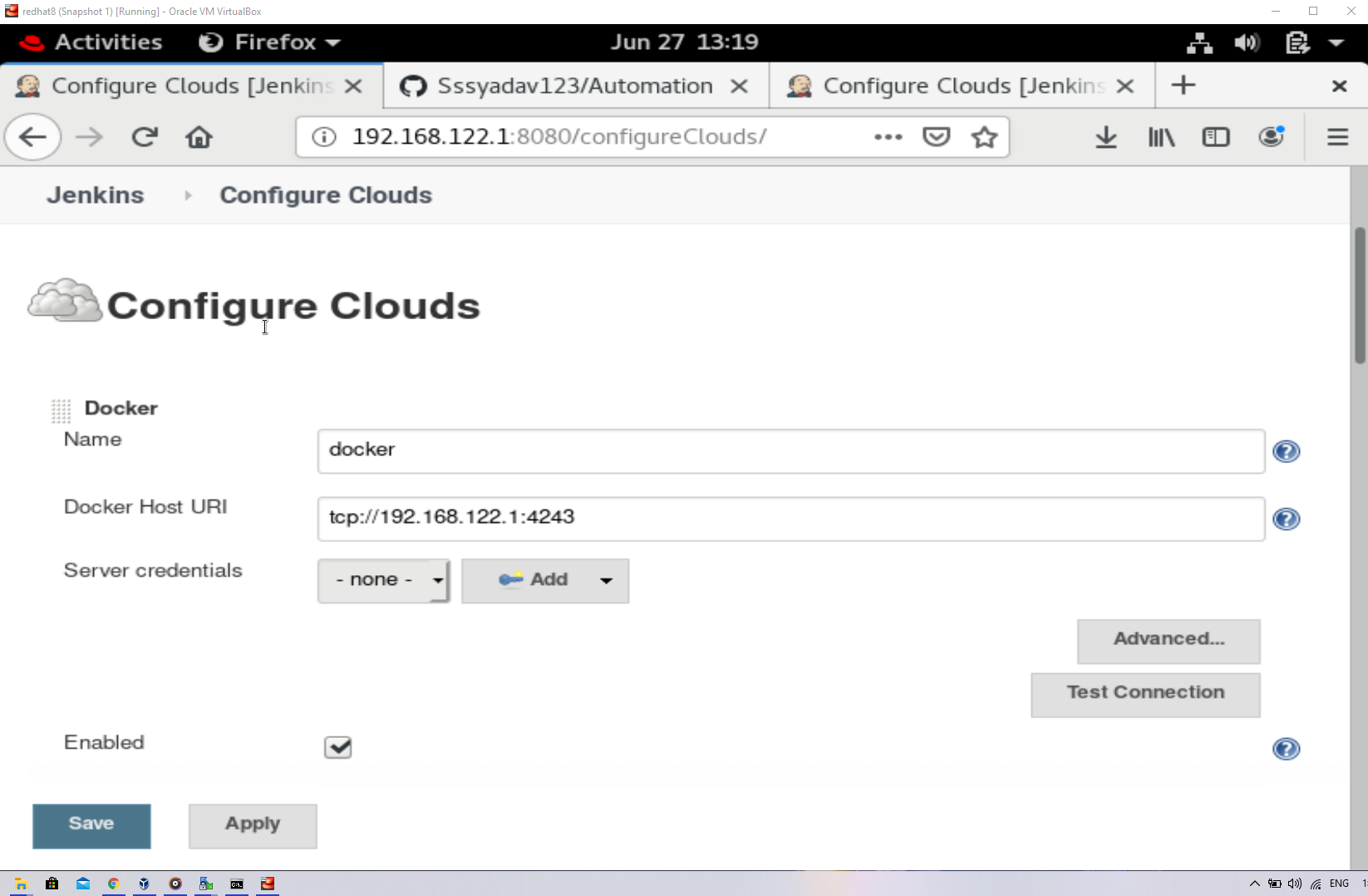
systemctl daemon-reload

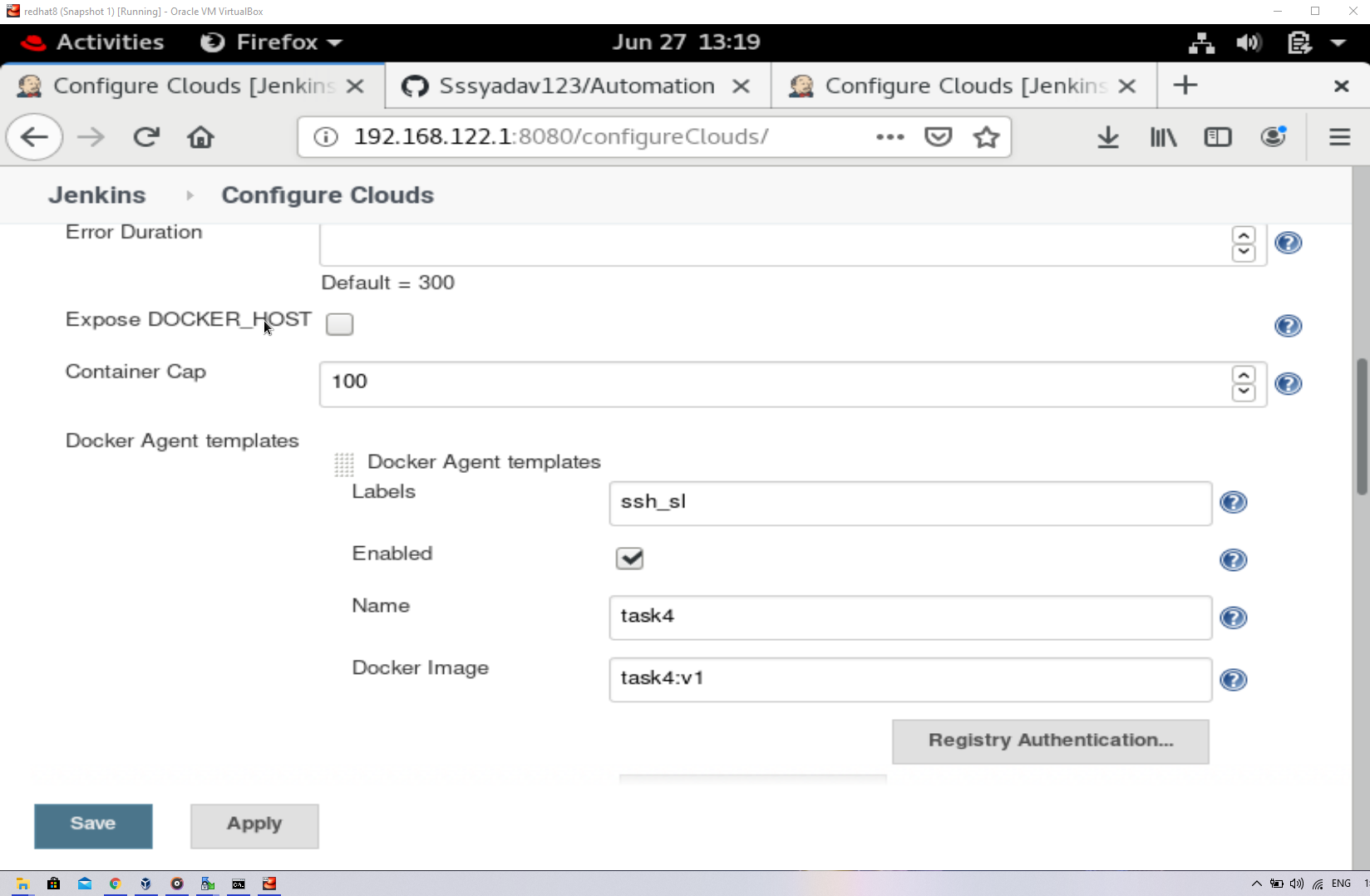
system restart docker

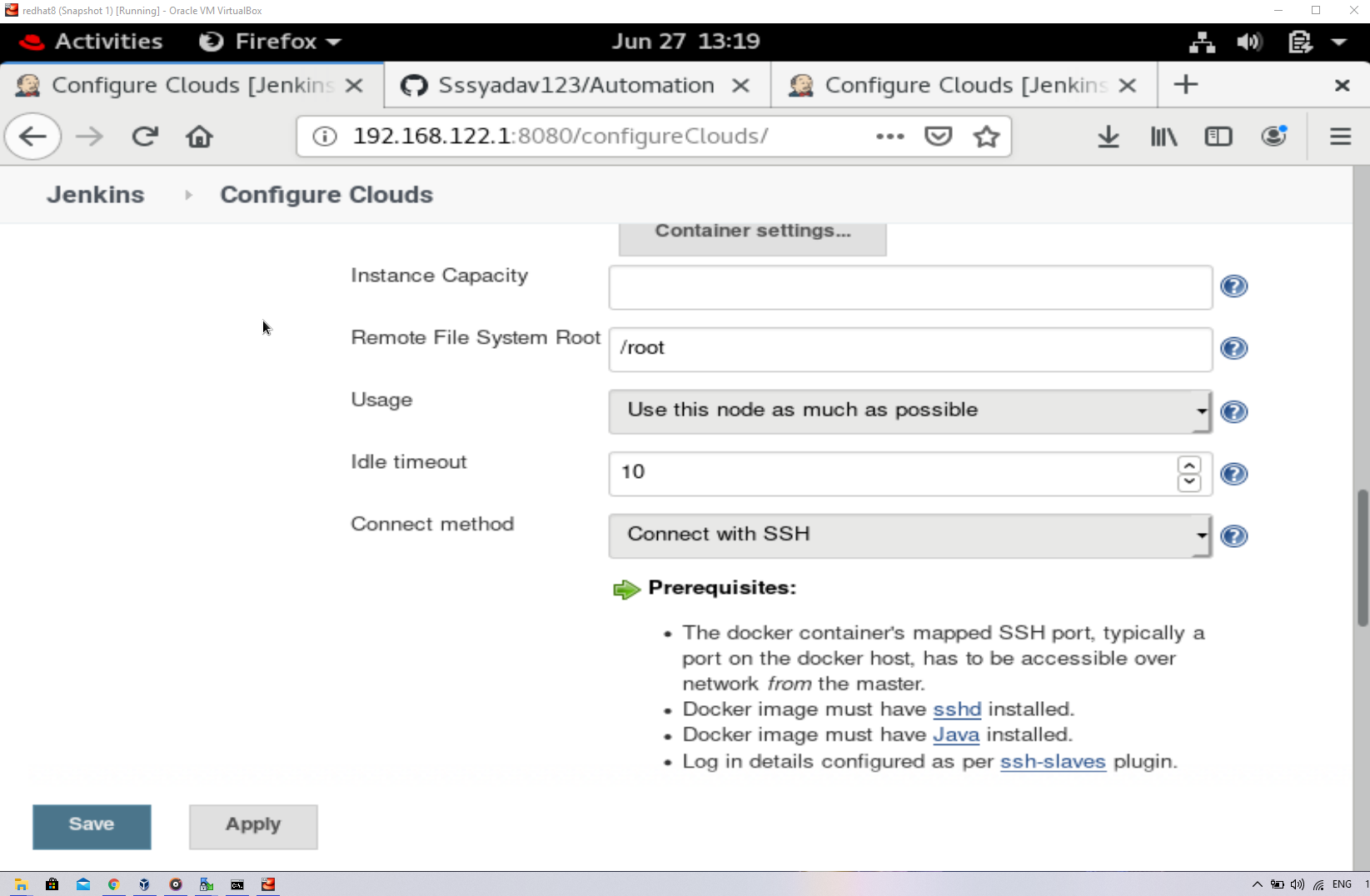
**Step - 6** Now, you need to configure the docker cloud in your jenkins. For this, you'll need to download a plugin known as ***docker***.

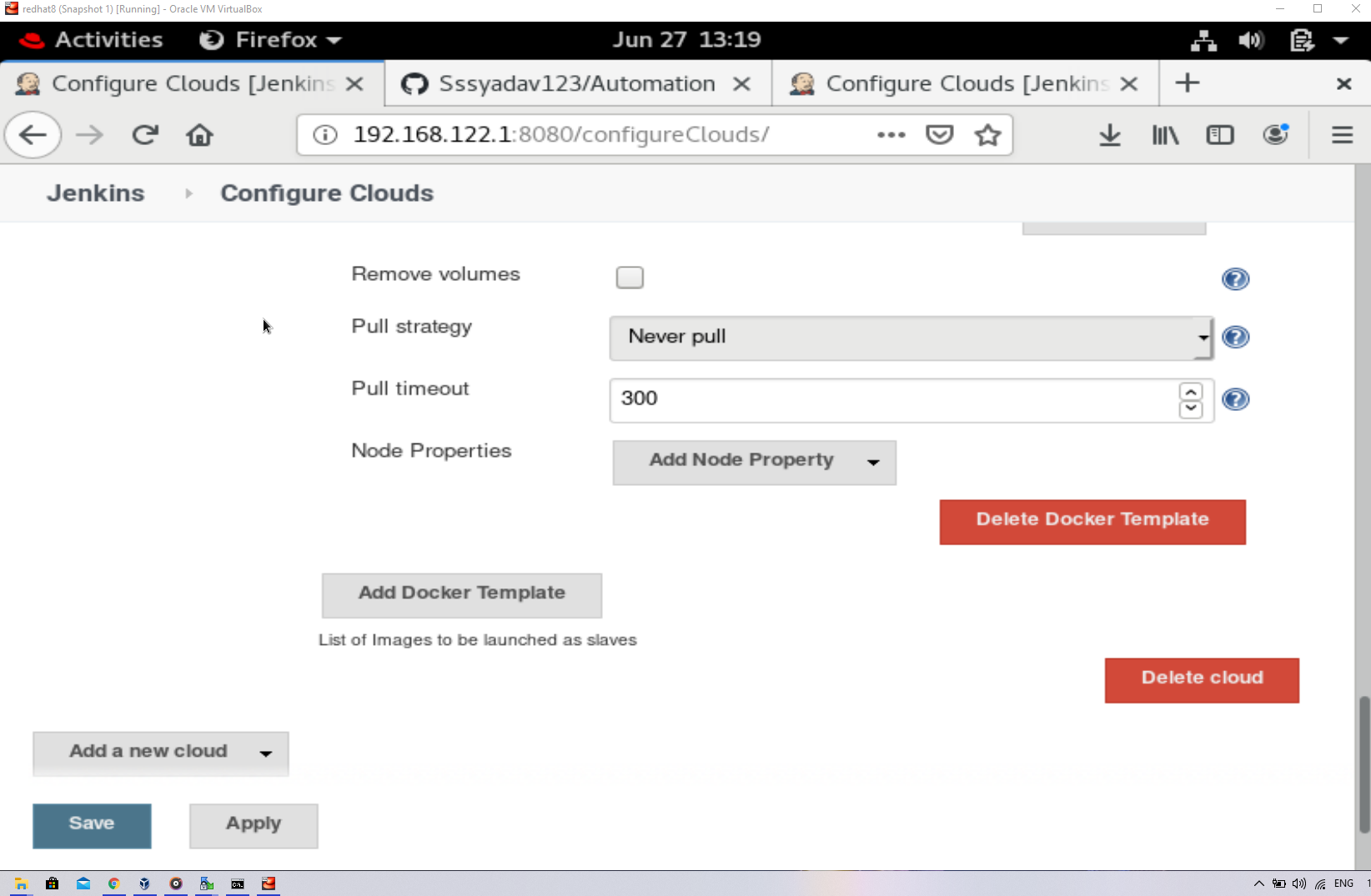
After downloading this plugin, go to ***Manage Jenkins***. From there, go to ***Manage Nodes and Clouds***. Select the ***Configure Clouds*** option from the left.

Configure the cloud as follows :

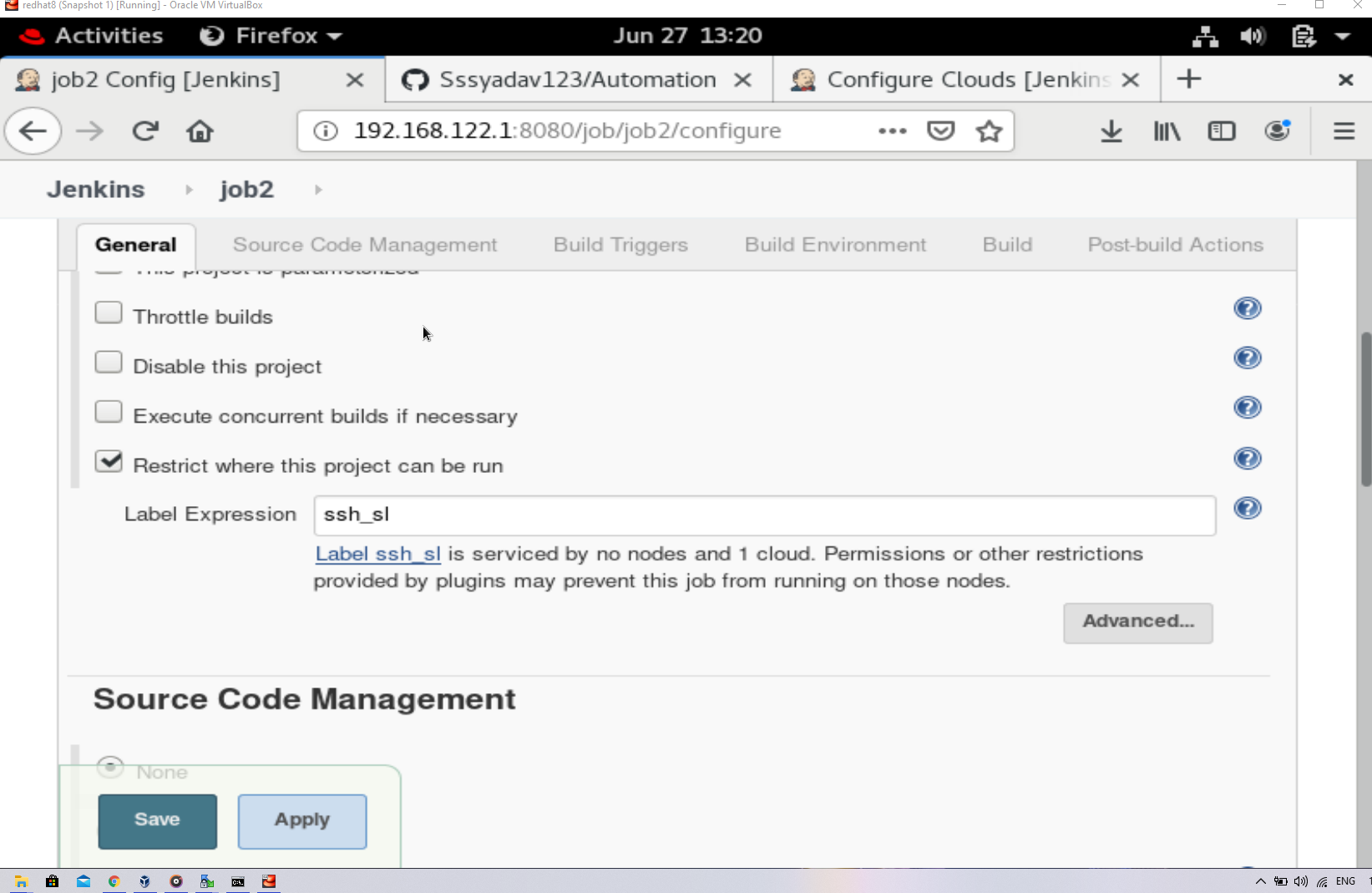








**Step - 7** Now, a Jenkins task will work to create the Deployment with the image. This task will also do Rollout in case of any updates.



Now, we go into cmd & check whether the Deployment is done or not.

