Jenkins project start

Creating Jenkins server to automatically take backup and store it to AWS S3 bucket.  
  
The detailed description of the project is given in “Jenkins-project-description.docx”  
  
this will be created as the below.  
  
This will be done with the help of docker. 3 docker containers will run one conatin Jenkins server running, second container will run MYSQL and the third will be used to remote login and connect to sql, Jenkins server and to be used for storing the script to automate the Jenkins build.  
  
the container used for remote login will communicate between the container and Jenkins server running on port 8080.

Create a Virtual machine on virtual box

* Currently using Centos 9

After setting up the Virtual box set up a bridge connection in network settings to be able to SSH over PUTTY.

Installing docker  
 [Docker Installation for CentOS](https://docs.docker.com/engine/install/centos/)

Follow the above steps install docket on centos 9.

Install the dnf-plugins-core package (which provides the commands to manage your DNF repositories) and set up the repository.  
>sudo dnf -y install dnf-plugins-core

>sudo dnf config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

Download the latest docker engine version

>sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

>press Y to continue downloading latest docker engine.

If prompted to accept the GPG key, verify that the fingerprint matches 060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35, and if so, accept it.

This command installs Docker, but it doesn't start Docker. It also creates a docker group, however, it doesn't add any users to the group by default

Start Docker Engine.

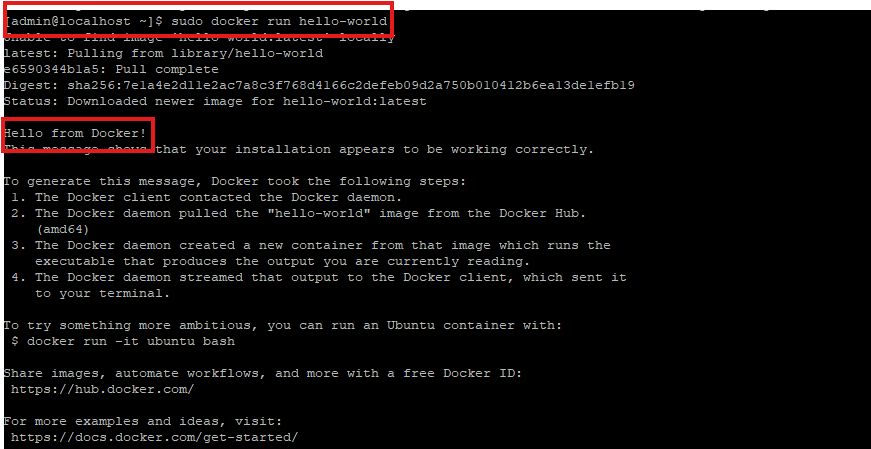
$ sudo systemctl enable --now docker

This configures the Docker systemd service to start automatically when you boot your system. If you don't want Docker to start automatically, use sudo systemctl start docker instead.

Verify that the installation is successful by running the hello-world image:

$ sudo docker run hello-world

!!! you have successfully installed docker.



To check the if we are running the Hello world image on a container run the below command.  
>docker ps

This will show show the running docker containers.  
But we got error  


This can be fixed by adding the user to the docker group. And restart the system.  
>sudo usermod -aG docker admin

(Note : the username here is admin. Check with whoami command)

Installing docker compose:  
>sudo yum update

>sudo yum install docker-compose-plugin

> docker compose version

Check by running the docker compose command

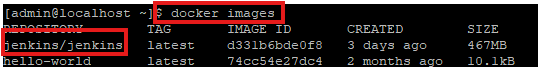
>docker compose.

Downloading the Jenkins image from docker.

<https://hub.docker.com/r/jenkins/jenkins>

> docker pull jenkins/Jenkins

To check the downloaded images you can check with docker images.

>docker images  
  


They will be store in the below path

>/var/lib/docker.

To run Jenkins on docker we will need to create a docker compose file:

Create a directory.  
>mkdir Jenkins  
>cd Jenkins  
>mkdir Jenkins\_data  
>cd Jenkins\_data  
create a file for docker compose  
>vi docker-compose.yml

And mention the below lines.  
  
 services:

jenkins:

container\_name: jenkins

image: jenkins/jenkins

ports:

- "8080:8080"

volumes:

- $PWD/jenkins\_home:/var/jenkins\_home

networks:

- net

networks:

net:

docker compose file is script, a definition to spin up Jenkins image.  
This Docker Compose configuration sets up a single service for Jenkins, a popular automation server used for continuous integration and delivery (CI/CD).

The service is named jenkins, and it uses the official jenkins/jenkins Docker image from Docker Hub. The container\_name directive assigns the container a specific name ("jenkins") for easier reference.

Port 8080 on the host is mapped to port 8080 in the container, which is the default port Jenkins uses for its web interface—this allows users to access Jenkins by navigating to http://localhost:8080 in a browser.

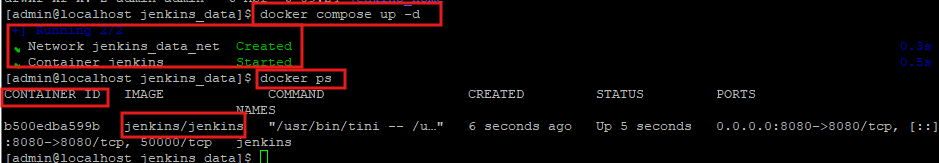
To ensure Jenkins data such as job configurations, plugins, and build history persist even if the container is stopped or removed, a volume is mounted from a directory named jenkins\_home in the current working directory ($PWD/jenkins\_home) to the container’s internal Jenkins home directory (/var/jenkins\_home).

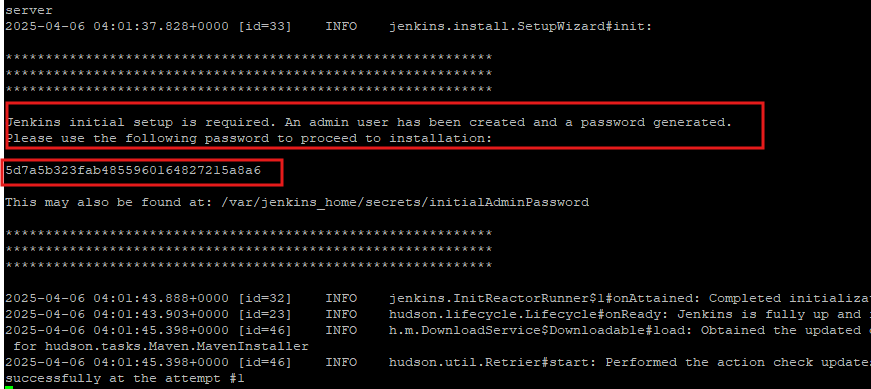
This allows Jenkins to maintain state between container restarts. Additionally, the service is connected to a user-defined Docker network named net. Defining and using a custom network enables the Jenkins container to communicate easily with other services (such as build agents or databases) that may be added to the same network later, using container names as hostnames. This setup is a foundational and efficient way to run Jenkins in a containerized environment with persistent storage and scalable networking.

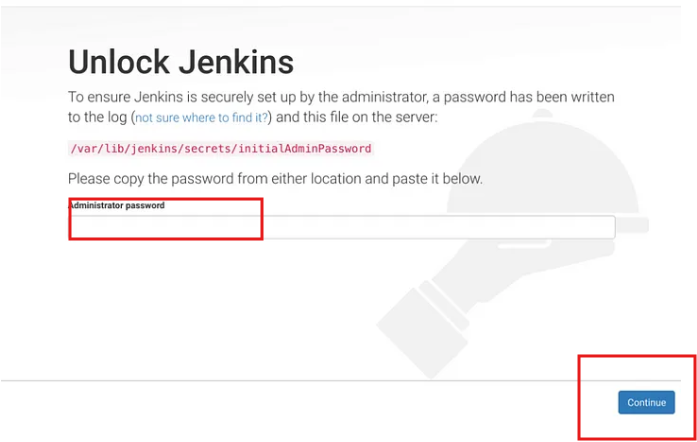
>exit the script from vi editor  
>esc , shift+: ,wq (save and exit the file)

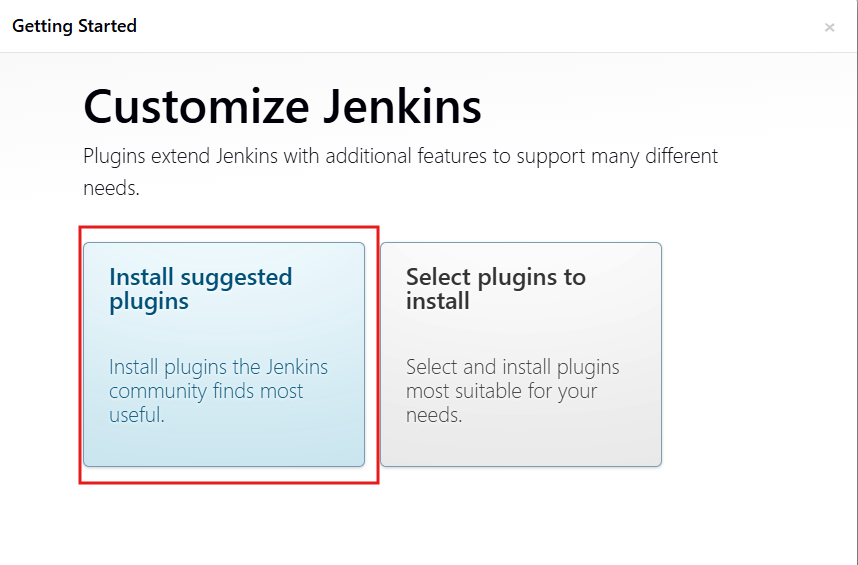
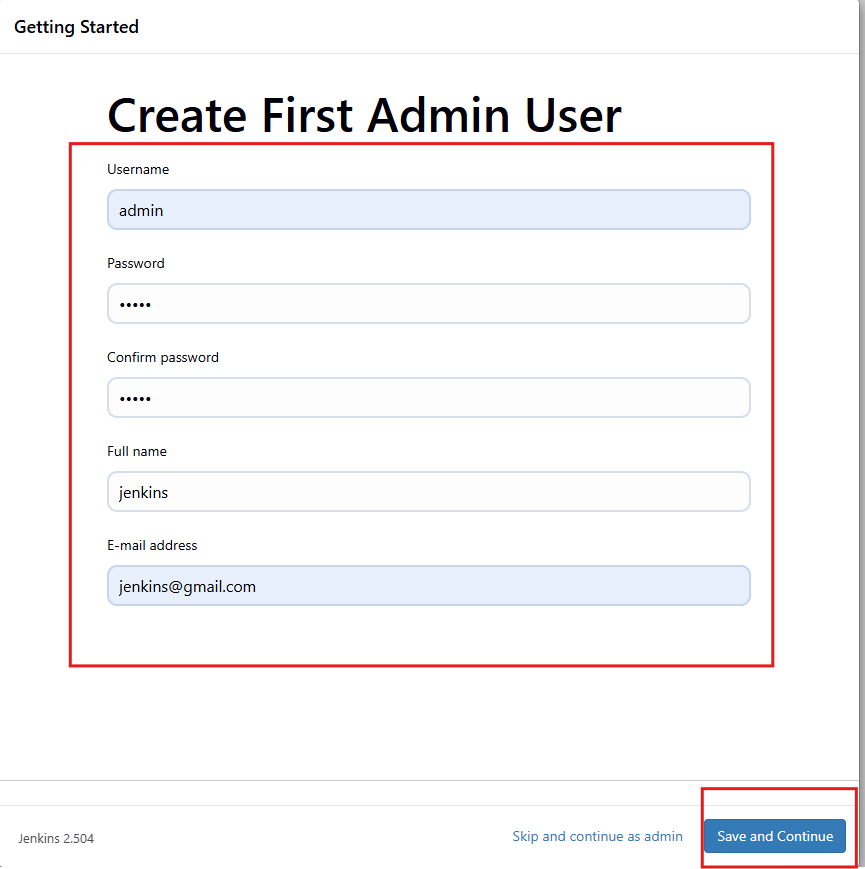
Since we have mentioned the internal working directory for our service , lets create it.  
>mkdir Jenkins\_home  
  
the docker file and the volume we have created should be in same file path.  

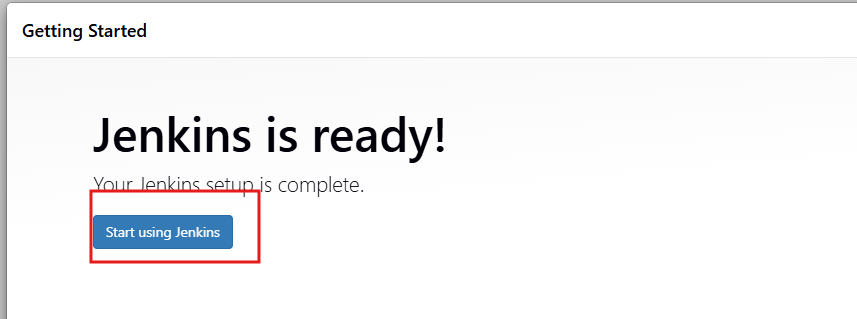

Creating a docker container for Jenkins.  
  
the new folder we have created should be given proper permission.  
The command sudo chown 1000:1000 jenkins\_home -R changes the ownership of the jenkins\_home directory (and everything inside it) to the user and group with ID 1000, which is the default user Jenkins runs as inside the container. This ensures Jenkins has the necessary permissions to read and write its data, preventing permission issues when it tries to access the mounted volume.  
  
> sudo chown 1000:1000 jenkins\_home -R  
  
All the permissions and the fservices are created, let start the service.  
>docker compose -up -d  
>docker ps

This will create a container for docker and will show the docker container running.  
  


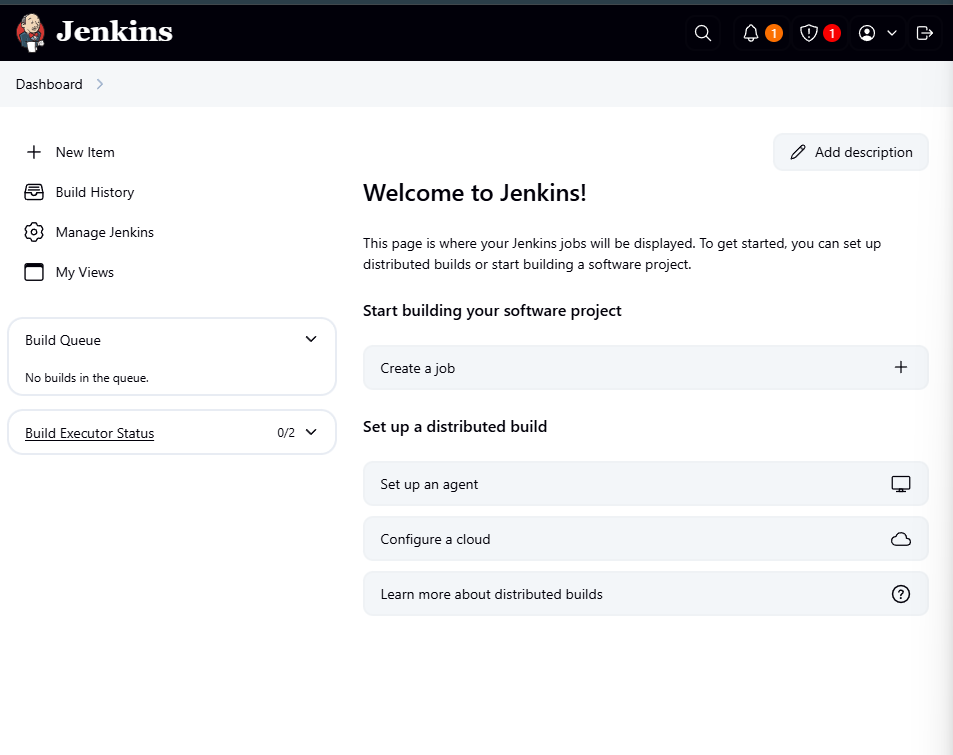
Loging into Jenkins.  
  
initially Jenkins server will ask for secret password which is located in the Jenkins file, we can also get it from the docker logs, since we have a Jenkins server running on a docker container.  
  
> docker logs -f Jenkins  
  
we can get the logs from the logs. Copy the pass word.  


Now open a new tab in browser and open the Jenkins server with your current ip on Virtual machine.  
  
>ifconfig, get the ip  
on new tab in browser.  
>your-ip:8080  
  
your Jenkins server will start  
  
once done paste the password copied form the logs and click continue.  


Click on install suggested plugins.  
  
  
this will install all the required plugins and will complete once done.  
  
enter the credential to continue  
  


Now we can start using Jenkins  


!! COOL we are now on the Jenkins home page



We have our jenkins server running on a docker container now  
  
Creating a docker container to use to run and store the scripts and ssh to Jenkins container.  
since we are creating a multi container application, it is good practise to segregate the services into different container instead of adding all the services in a single container.  
  
We will execute the Jenkins jobs in a remote machine. to do this wee need 2 containers,  
Jenkins server running on one container and our remote machine running on another container to execute jobs for Jenkins server.

Lets create a new container and introduce SSH service, so we can connect from Jenkins container to new container.  
  
Create a folder for our new container create a new image and save the files.  
  
>go into jenkins\_data folder created  
>mkdir ssh\_img  
>cd ssh\_img  
 to create a new container we need image lets create one.  
>vi Dockerfile  
  
a docker file is used to create images from which we can create the containers.  
The docker file give instructions and set of commands to run to create our image.  
  
the CentOS image is no longer maintained on Docker Hub. This means that if you use FROM centos in the next videos, you'll run into issues.

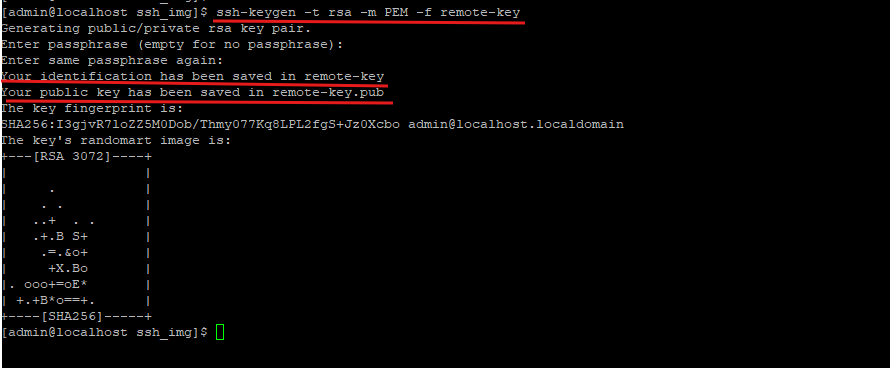
Instead, use Fedora 41, which is the closest alternative:  
enter the below commands in the script.  
  
>FROM fedora:41

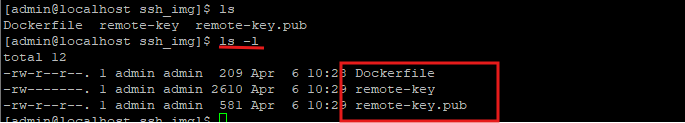
>RUN yum -y install openssh-server

>RUN useradd remote\_user && \

> echo "1234" | passwd remote\_user --stdin && \

> mkdir /home/remote\_user/.ssh && \

> chmod 700 /home/remote\_user/.ssh  
  
save and exit the file.  
  
since we are creating this image to remote ssh to our Jenkins server, we should have secure ssh login  
we will create a SSH key  
  
>ssh-keygen -t rsa -m PEM -f remote-key  
This approach should help ensure compatibility and prevent potential issues.  
>press enter , enter to proceed and create ssh key.  
  


This will be stored in the ssh\_img folder.  


>cat remote-key  
  
this will display the key contents(encrypted)  
  
now open the Dockerfile and complete the script by adding permission and creating a secure open ssh image commands.

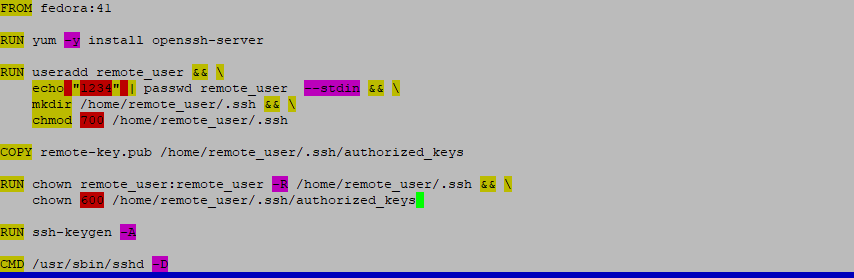
>vi Dockerfile  
  
>COPY remote-key.pub /home/remote\_user/.ssh/authorized\_keys

>RUN chown remote\_user:remote\_user -R /home/remote\_user/.ssh && \

chown 600 /home/remote\_user/.ssh/authorized\_keys

>RUN ssh-keygen -A

>CMD /usr/sbin/sshd -D

Your Docker file script should look like this  
  
  
  
This Dockerfile creates an SSH-enabled Fedora 41 container. It installs the OpenSSH server, creates a user named remote\_user with the password 1234, and sets up their SSH directory. It copies a public SSH key (remote-key.pub) into the authorized keys file so the user can log in using key-based authentication. Proper permissions are set for security, and SSH keys are generated for the server. Finally, the container runs the SSH daemon in the foreground when started.

Save the file and go back to the Jenkins data folder.  
  
>cd ..

We have now created a dockerfile. To create an image for our open ssh machine, lets create a service for the image to run as a container.  
we will define the service for our remote machine.  
open the docker-compose.yml file.

>vi docker-compose.yml  
add the new service  
> remote\_host:

container\_name: remote-host

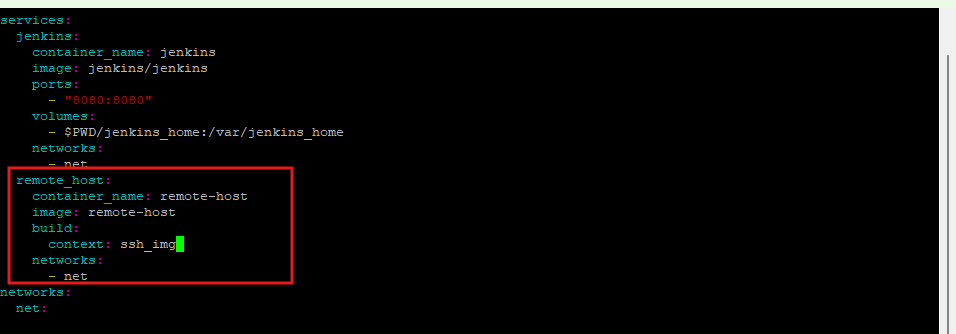
image: remote-host

build:

context: ssh\_img

networks:

- net

Your docker compose file should look like this.  


The remote\_host service creates a container named remote-host using a custom Docker image built from the ssh\_img directory. This image likely includes an SSH server setup, allowing remote access or communication. The container is connected to the same net network as Jenkins, so they can interact with each other internally by name. This setup is useful if Jenkins needs to connect to remote-host for tasks like deploying code or running remote commands.

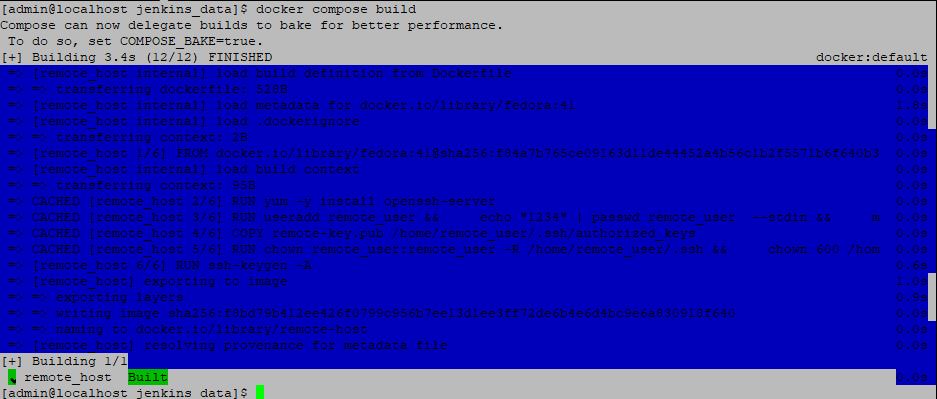
Save and exit the file

Now lets build the updated file.

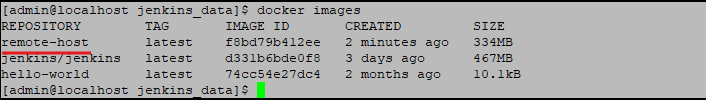
>docker compose build   
  
The docker compose build command will look at the remote\_host service in your docker-compose.yml file and do the following:

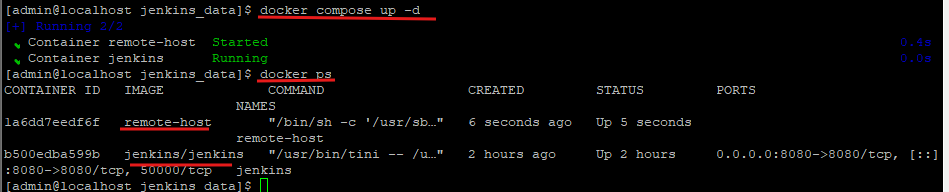
* It will go into the ssh\_img directory specified by build.context.
* Inside that directory, it will look for a Dockerfile.
* Using the instructions in that Dockerfile, it will build a custom image and tag it as remote-host (based on the image: remote-host line).
* Once built, this image can be used to run the remote\_host container.

In short, docker compose build compiles the image for remote\_host from your local Dockerfile before running it. This is useful when your container needs a custom setup, like installing SSH and configuring users.

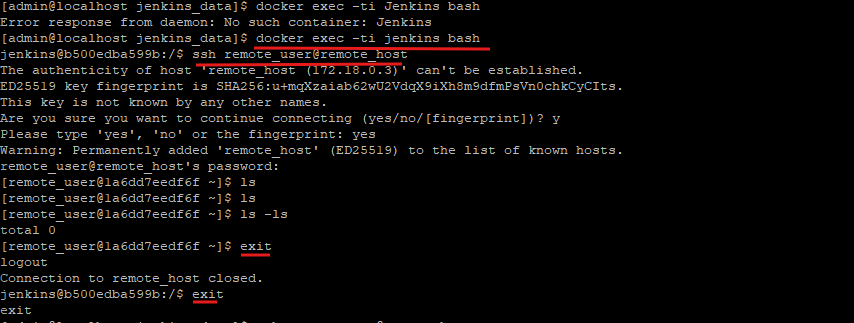
Output:  


You can check in the docker images that we have a new remote-host image created.  
  
>docker images



We have image created, now create the container.  
  
> docker compose up -d  
>docker ps  
  


Now lets enter in the Jenkins container terminal to check if our remote ssh works  
as we have added details in the script.  
  
>docker exec -ti jenkins bash  
> ssh remote\_user@remote\_host  
  
press yes to continue and enter the password (pass owrd is 1234 from the Dockerfile created)  
cool !!!. we are logged into the remote-host container from our jenkins container.  
exit the containers.  
>exit  
>exit

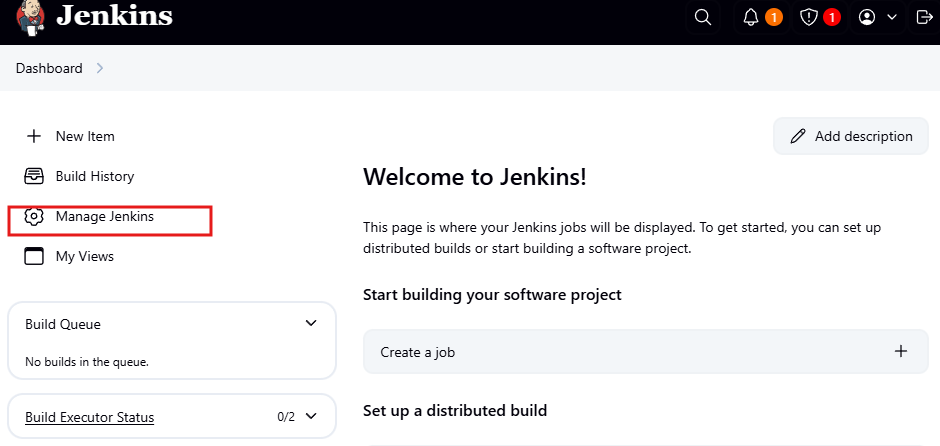
Output:  


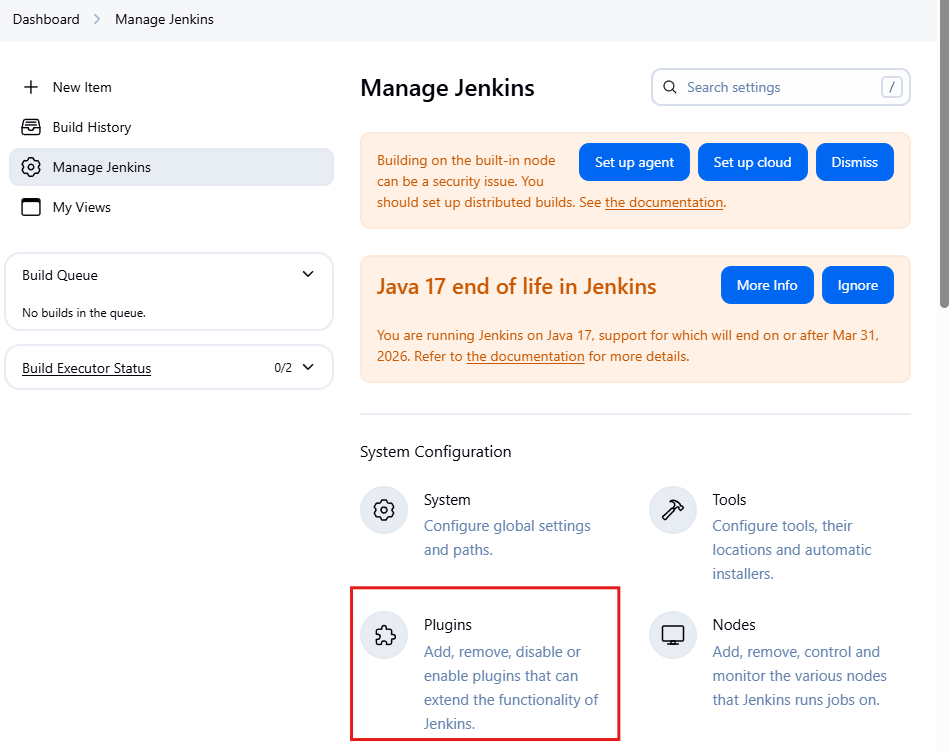
Now lets setup ssh plugin in Jenkins to directly connect to our remote host securely.

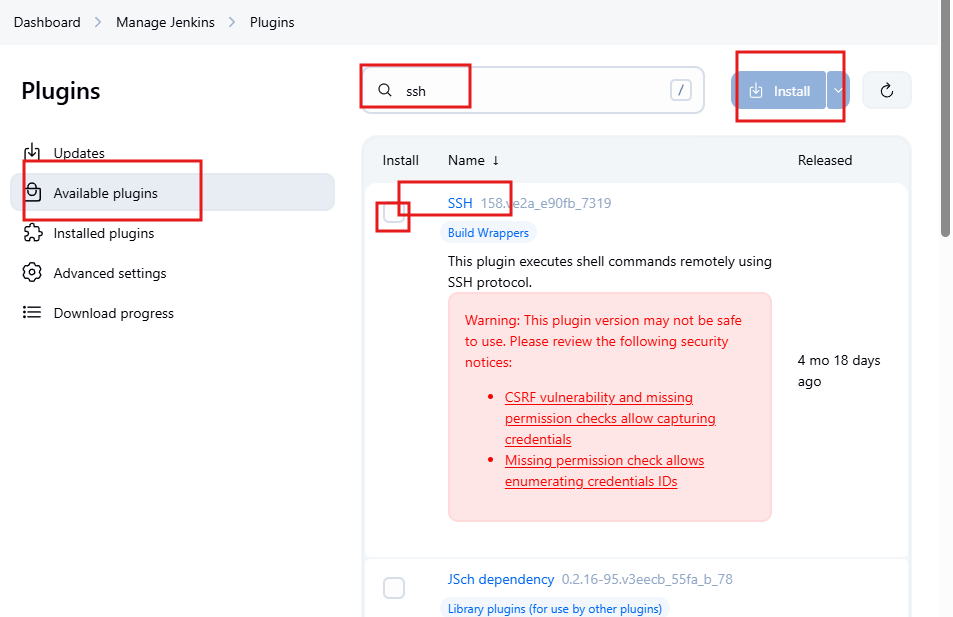
First go into the ssh\_img folder and copy the remote-key to our remote\_host container  
>cd ssh\_img  
>docker cp remote-key jenkins:/tmp/remote-key   
  

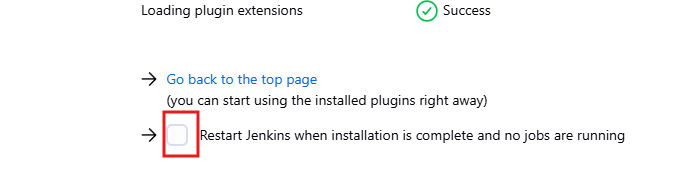

**This command copies a file named remote-key from your host machine into the running jenkins container**, placing it at /tmp/remote-key inside the container.  
(NOTE: if the docker image are closed/shutdown the file will be deleted, and need to be repeated again, we will ses a fix for this below in upcoming topics)

Now Installing plugins.  
open the Jenkins server in a new browser tab  
>yourip:8080  
login to Jenkins.

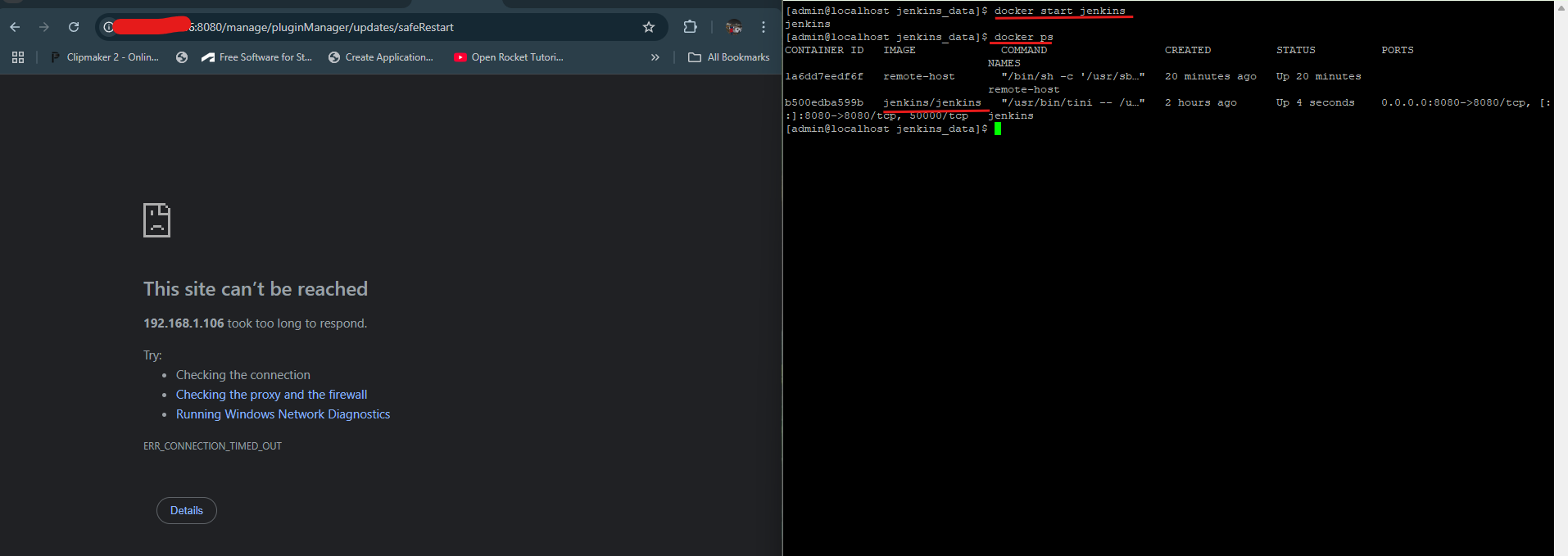
Click on manage Jenkins  


Go to plugins  


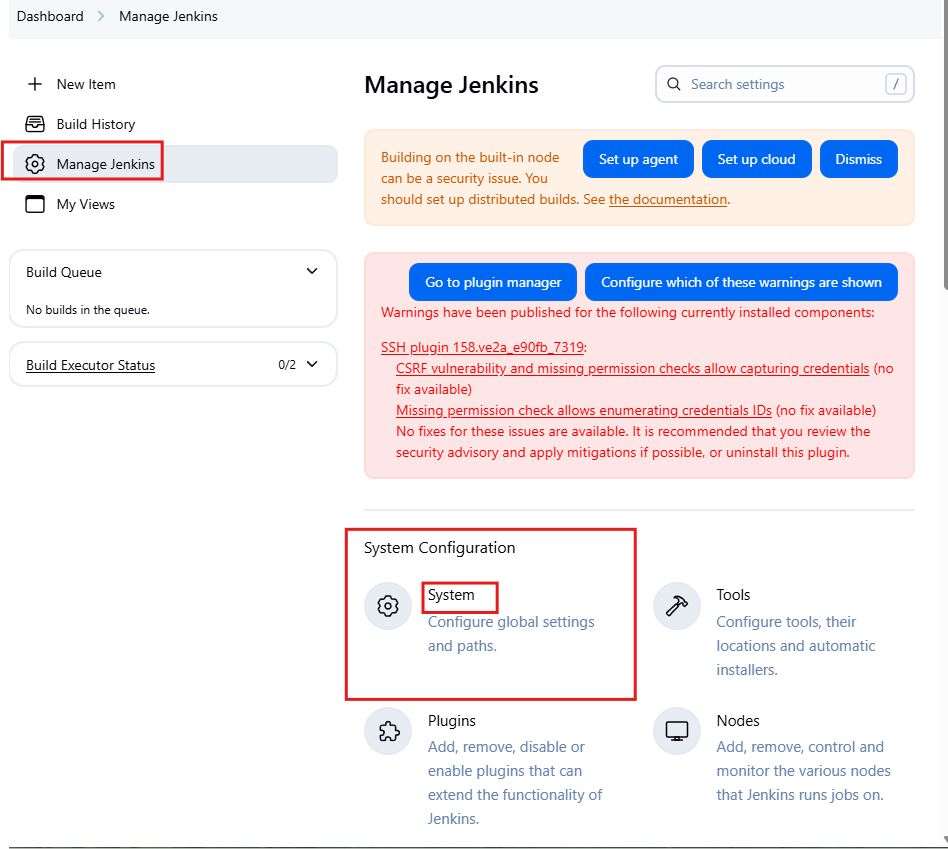
On available plugins search for ssh plugin and install  


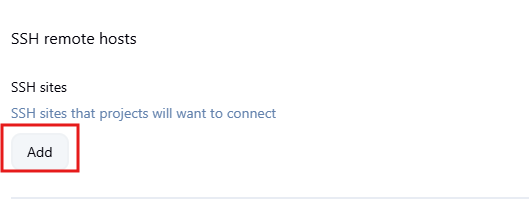
Once done to reflect restart the Jenkins server, click on restart Jenkins and wait to load and login again.  
  


If the page dosen’t respond , go to the terminal and restart the Jenkins container.

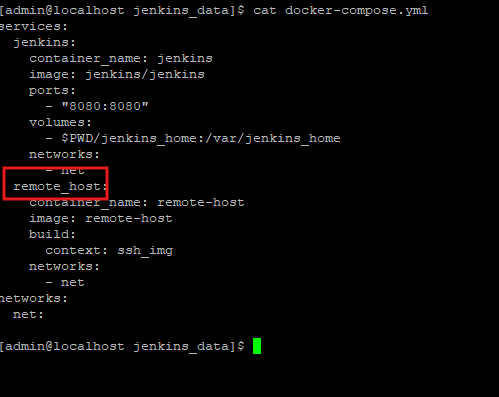


This should bring up the Jenkins page and now check in the Jenkins installed plugins, ssh should be available.  
  
Now configure the authentication for our remote-host container to Jenkins server  
so we have direct connection to our host from ssh

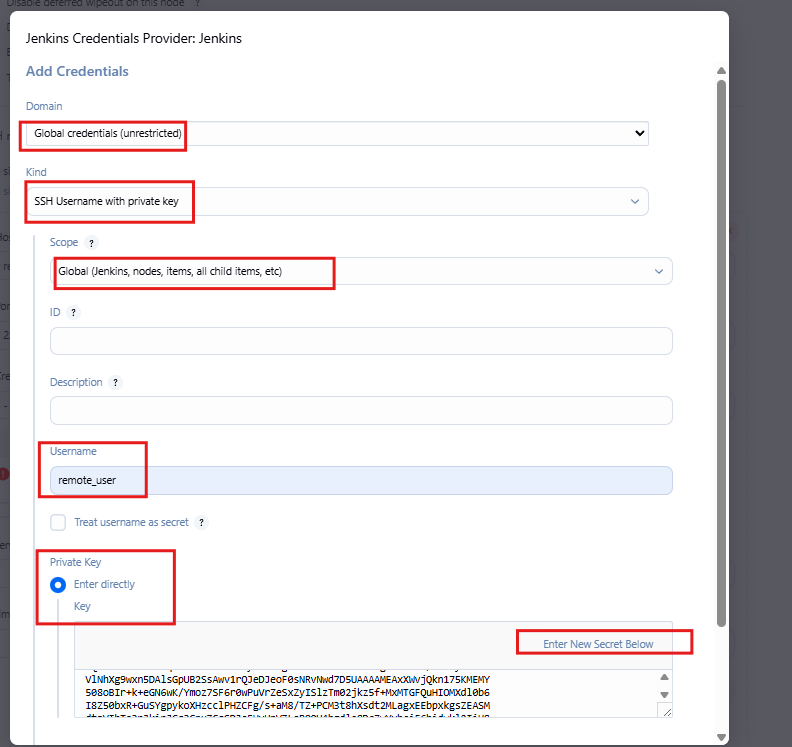
Go to Jenkins -> manage Jenkins -> click on system configuration.  
  


Go onto ssh remote host and click on add to add sites  
  


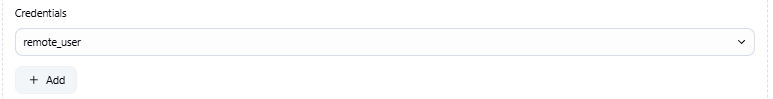
Add the hostname and port(default port) and add the credentials, click on add buton  
  


(note the host name will be available from the docker-compose file)  


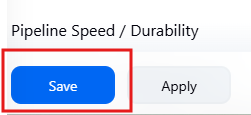
Click on add credentials 🡪kind: SSH Username with private key 🡪 add the username from the Dockerfile created in ssh\_img folder, click on private key and enter directly.(this is the private key created in the ssh\_img created earlier.)  
  

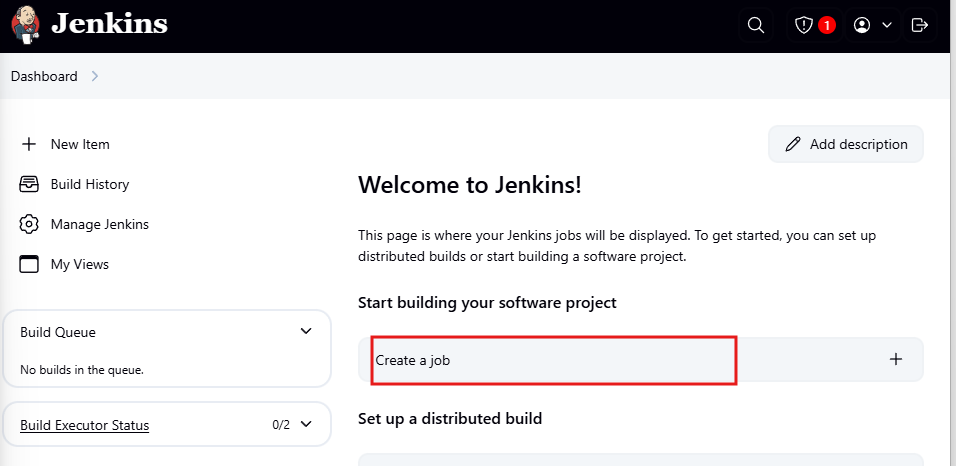

Copy all the contents and click on add  
  


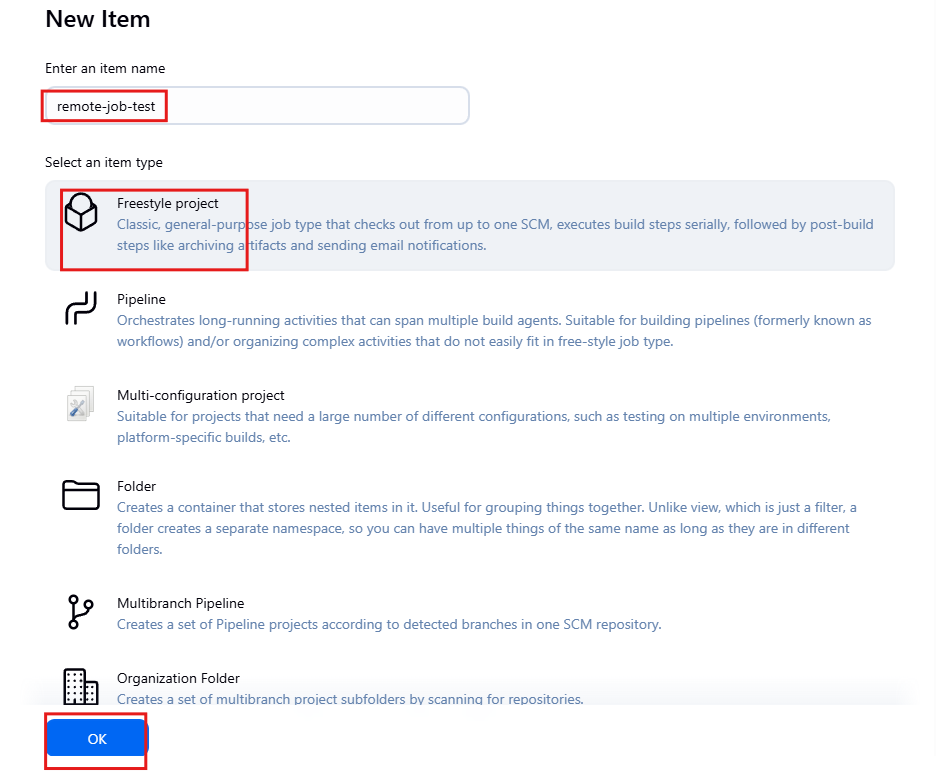
Now add the credential we created

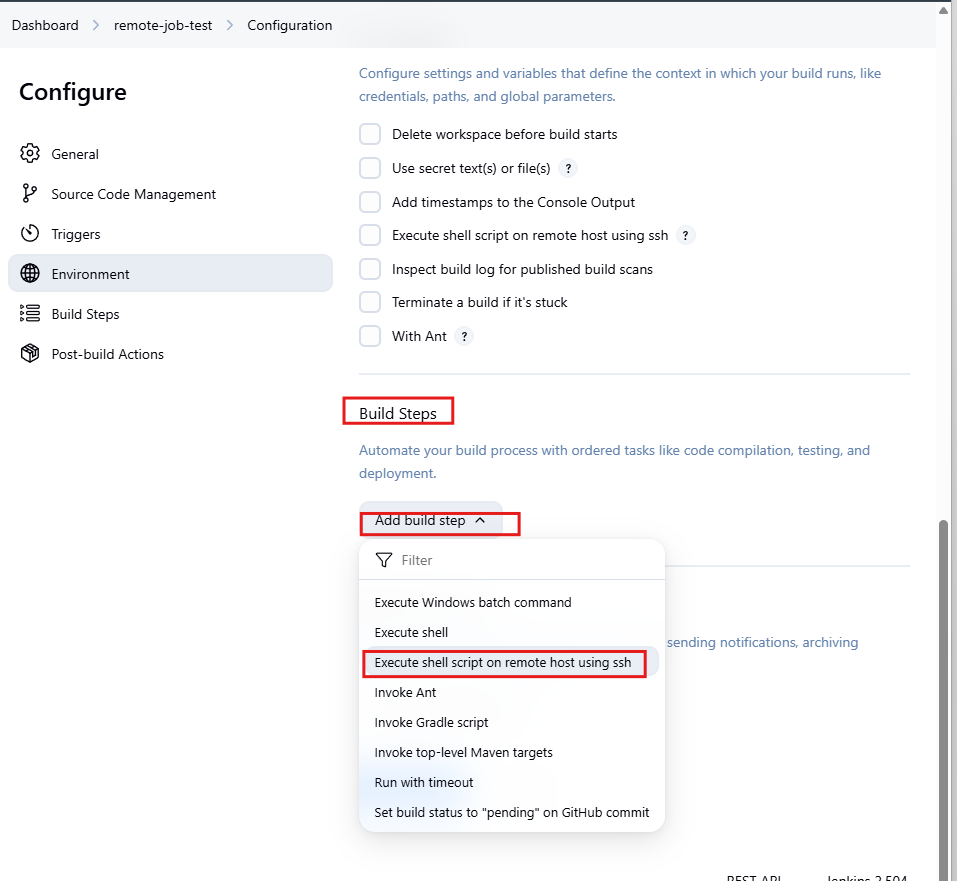


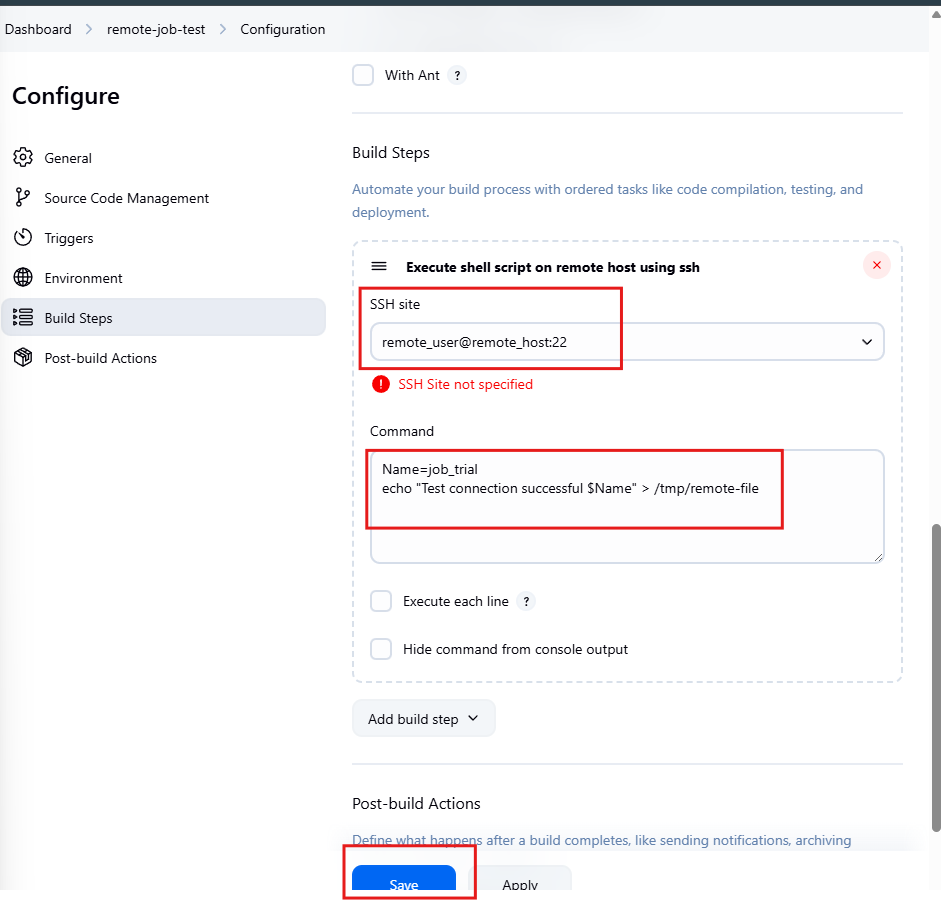
And click on check connection to test the connection   
  
  
  
!!!Your connection is successful.  
we have configured the remote host.  
now save the connection.

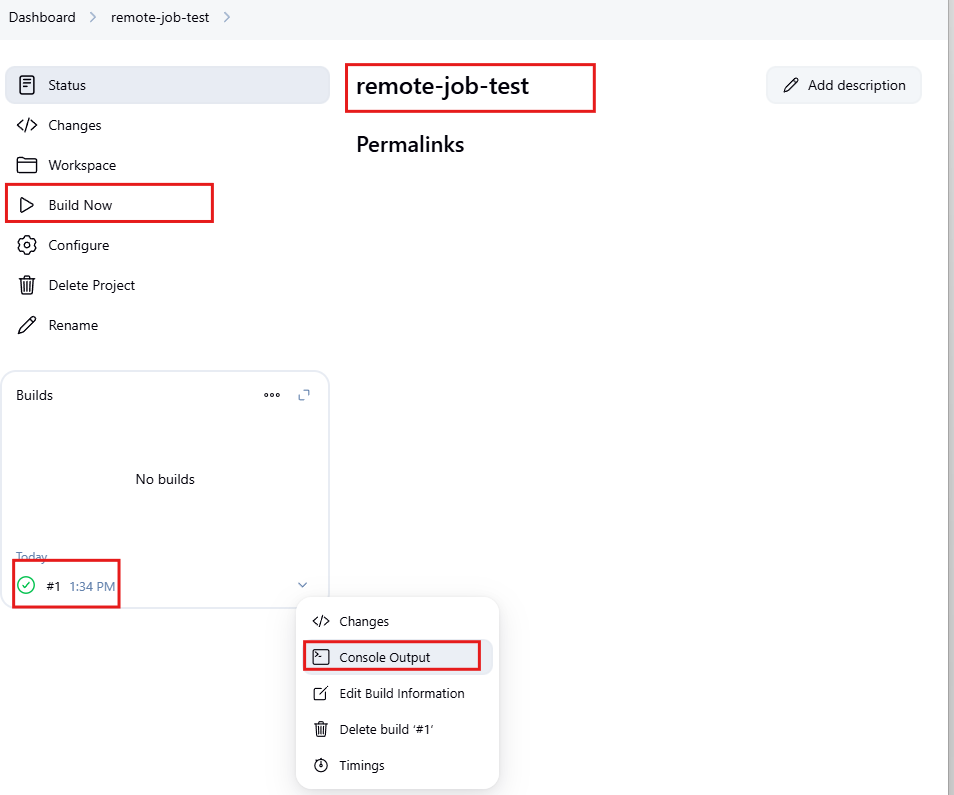


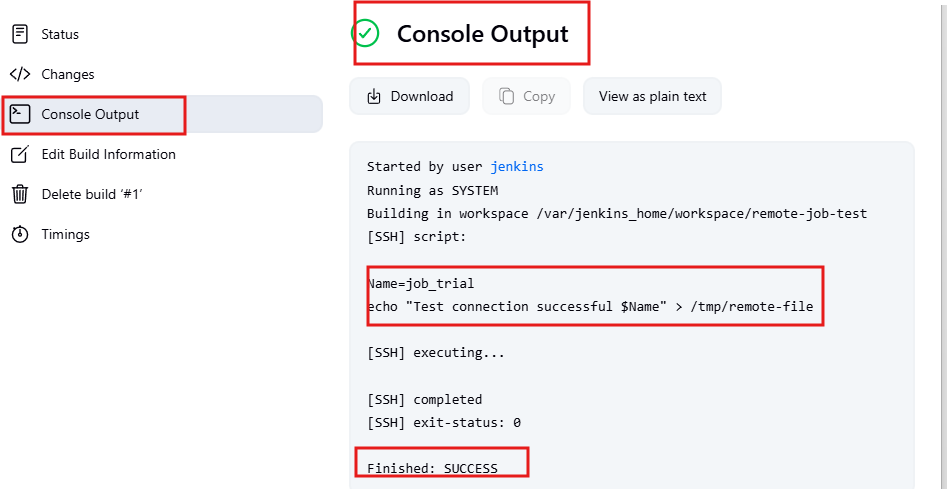
Testing the connection.  
Lets test the connection by creating a sample job in Jenkins.  
  
go on dashboard -> create a job -> name the job -> freestyle project  
  
 

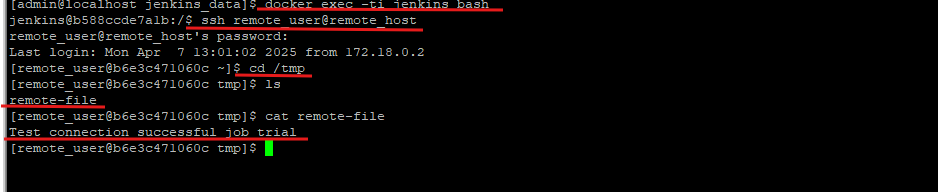


go to build section -> execute shell script on remote host using ssh.  
  


lets create a small bash command and save it, test it.  
the remote-host and user will to prompted as we have saved them in config.  
this job should create a file in our remote-host container.  


Lets build the job.  
click on Build Now -> let the run complete -> click on console output.  
  


Our console output should be in success. Lets check them in our running container.  
  


We have our remote-job generated.  
open the remote container.  
  
> docker exec -ti jenkins bash  
> ssh remote\_user@remote\_host  
> cd /tmp  
>ls  
>cat remote-file.  
  


This way we have created a Secure connection over Jenkins container to our remote host .

Now Lets create a Jenkins job that can take a backup and store it in Amazon S3 bucket.  
  
we can approach this by creating our 3rd container which will run MYSQL and we will take the back up of the data run from the container to Amazon S3.

Creating MYSQL container using docker.  
  
Lets open the docker-compose.yml file in Jenkins\_data folder.  
  
>cd   
>cd jenkins/jenkins\_data  
>vi docker-compose.yml  
  
  
creating a new service for MYSQL named “db\_host”  
> db\_host:

container\_name: db

image: mysql:5.7

environment:

- "MYSQL\_ROOT\_PASSWORD=1234"

volumes:

- $PWD/db\_data:/var/lib/mysql

networks:

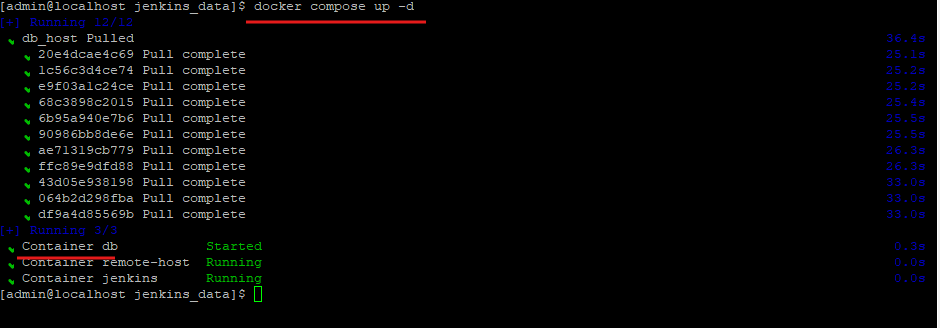
- net  
  
This can be understood from: <https://hub.docker.com/_/mysql>

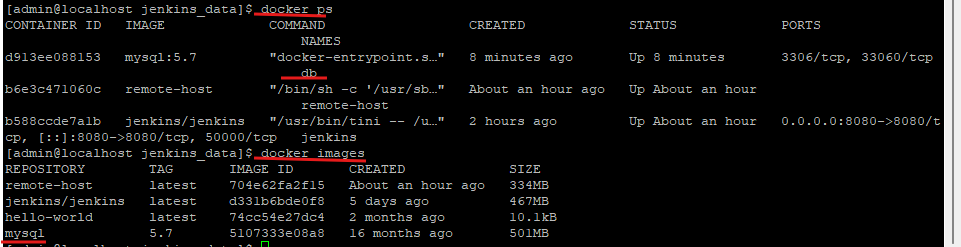
The given configuration snippet defines a **MySQL 5.7 database container** using Docker Compose. The container is named db and uses the official mysql:5.7 image.  
 It sets the root password for the MySQL server to 1234 using the MYSQL\_ROOT\_PASSWORD environment variable. The volumes directive mounts a persistent volume from the host machine's db\_data directory (located in the current working directory $PWD) to /var/lib/mysql inside the container, which ensures that the database data is stored outside the container and remains intact even if the container is removed.   
Finally, the container is connected to a custom user-defined Docker network named net, allowing it to communicate with other containers on the same network using the service name db\_host.

your docker-compose.yml should look like this.



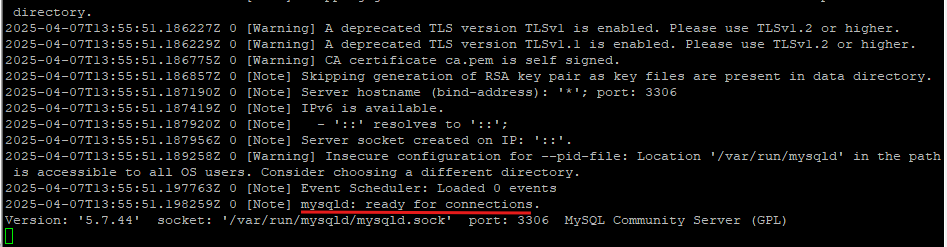
>save the file   
create the service  
>docker compose up -d

This will create a container and automatically create a file named db\_data as volume for the container.  
  
  
  
we can check the container running and the image created as well

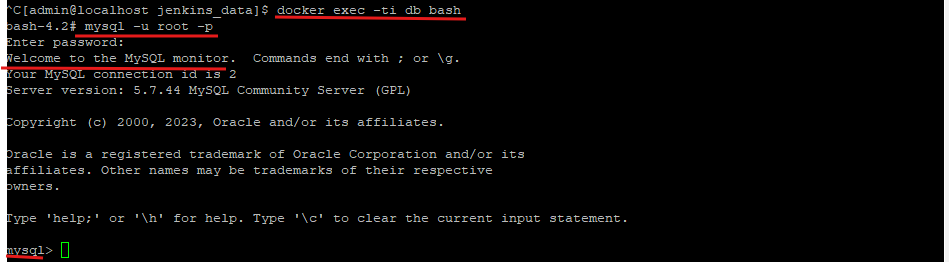


And we will have our volume folder created as well



Lets check if our container is ready.  
> docker logs -f db  
  
you should see mysql ready for connections. Your service is up and running.  


Lets check the sql container and login into it.

>docker exec -ti db bash  
lets open mysql with root user and password(Note password is declared int the environment in docker-compose.yml file)  
  
> mysql -u root -p  
>enter the password.  
  
!! cool our MYSQL container is running.  
  


Try running -> show databases;  
you should be able to see the databases.  
>exit; (to exit mysql)  
>exit (to exit the container)

Now we have setup our MYSQL container.  
next is to create a connection between our container and Amazon S3 bucket.  
  
for this we will need Amazon CLI from which we can connect to Amazon S3.  
and Also install MYSQL in our remote\_host container

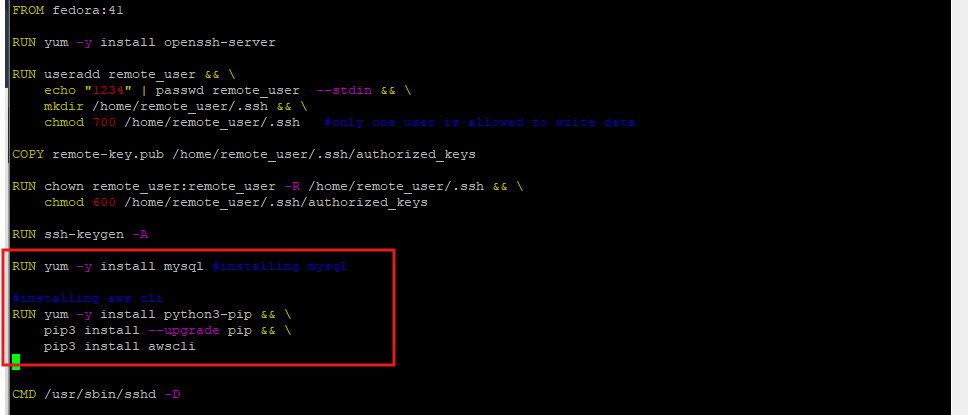
We need the above to take SQL back to s3 from our remote machine which is our remote\_host container.  
  
now we nee to modify remote\_host docker under ssh\_img folder.  
  
>cd ssh\_img  
>vi Dockerfile  
  
add the below lines.  
>RUN yum -y install mysql #installing mysql

#installing aws cli

>RUN yum -y install python3-pip && \

pip3 install --upgrade pip && \

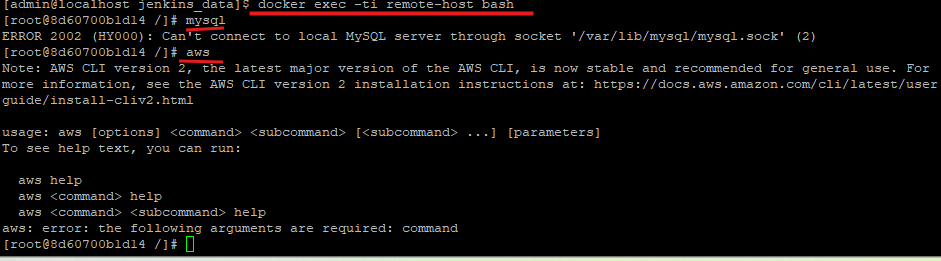
pip3 install awscli

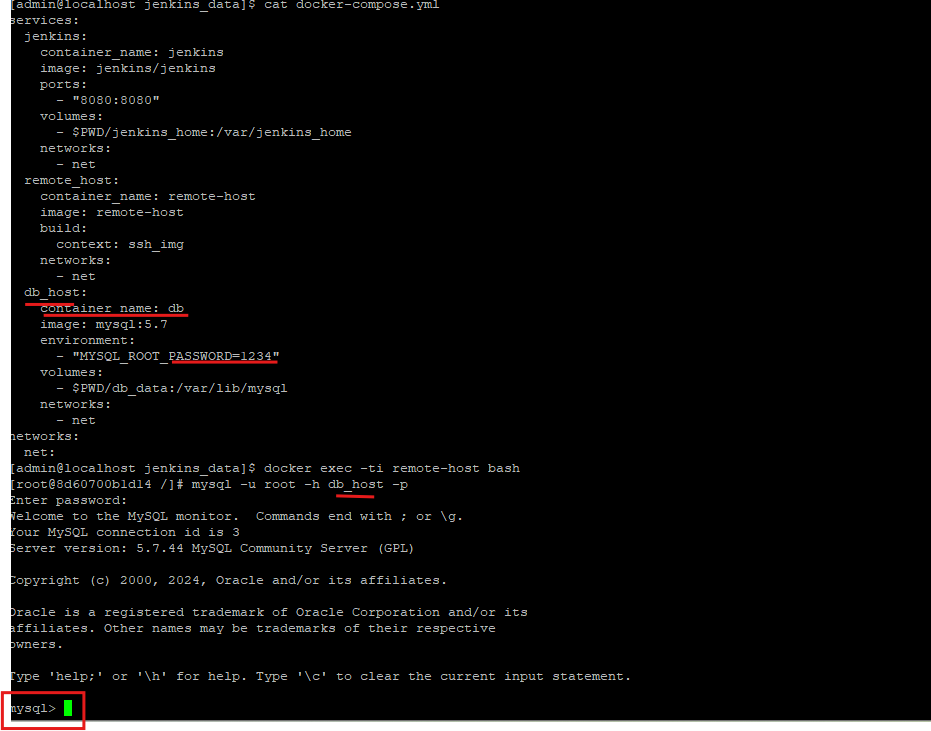
Now you Dockerfile should look like this.  
  


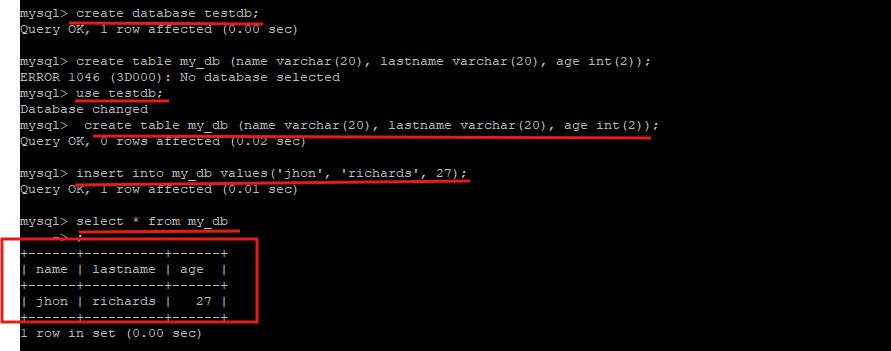
> save the file and exit.  
>cd .. (go back to the docker-compose file)  
  
you can refer the document for the latest updates.  
<https://docs.aws.amazon.com/cli/v1/userguide/install-linux.html>

Lets build the image so that our remote-machine (remote\_host) has mysql installed and Amazon CLI installed

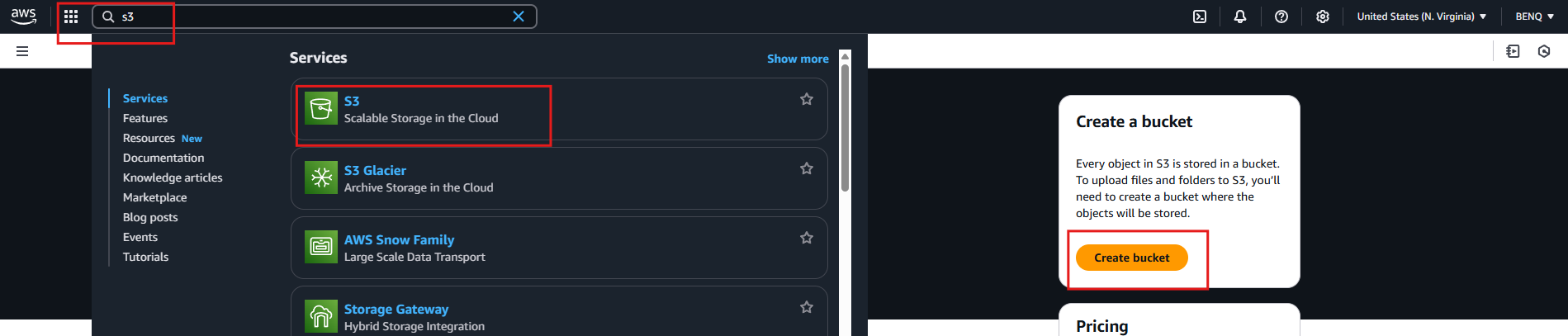
>docker compose build  
  
Now bring the updated the image of the container.  
>docker compose up -d  
  
the will create the container using the image we updated.  
Lets the it in the remote-host container.

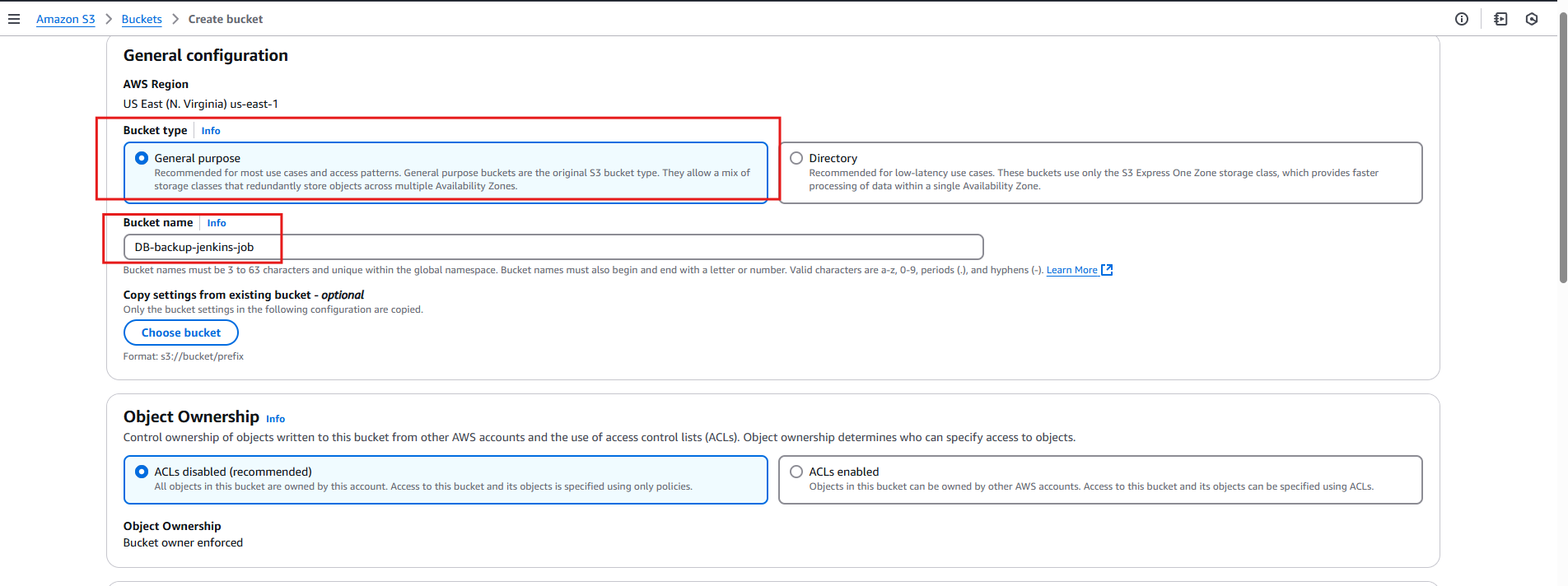
>docker exec -ti remote-host bash  
>mysql  
>aws  
(note: you will get sql error but it is ok for now, we will be using mysql to connect via host to our mysql container)  
>exit  
  
  
  
  
we are good to create our new MYSQL database.

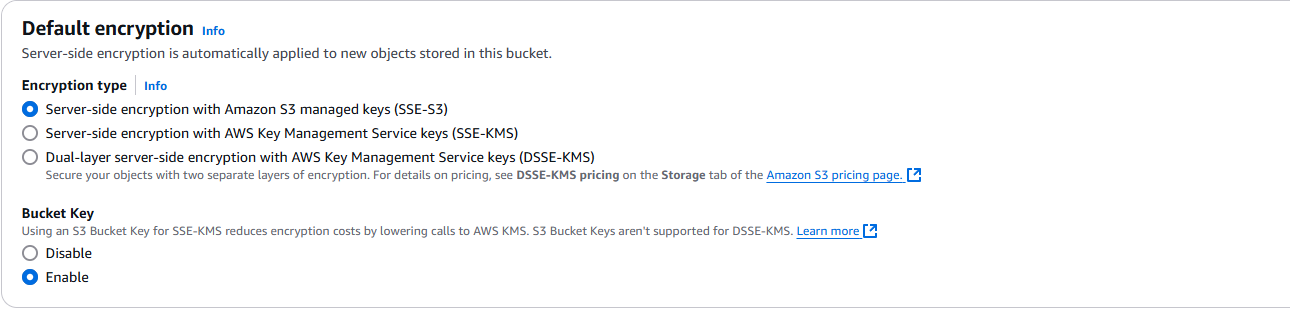
We will login into our remote-host container and setup connection using Mysql host connection.  
This will directly connect us to the MYSQL container created.  
these details are declared in the docker-compose file.  
  
  
> docker exec -ti -remote-host bash  
>mysql -u root -h db\_host -p  
(below is the comparison image to show where the details are being picked up)  
  


> create database testdb;  
> use testdb;  
> create table my\_db (name varchar(20), lastname varchar(20), age int(2));  
> insert into my\_db values('jhon', 'richards', 27);  
> select \* from my\_db  
  
you should see your database and table created.  
  


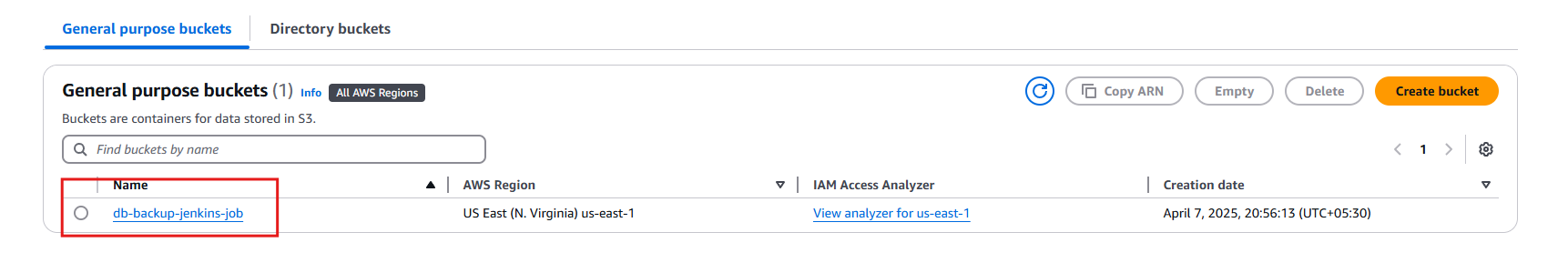
Now we have created the Database. Lets create S3 bucket.  
We need to create a S3 bucket and create IAM permission to authenticate and do the backup by creating a user assigned with programmatical access keys.

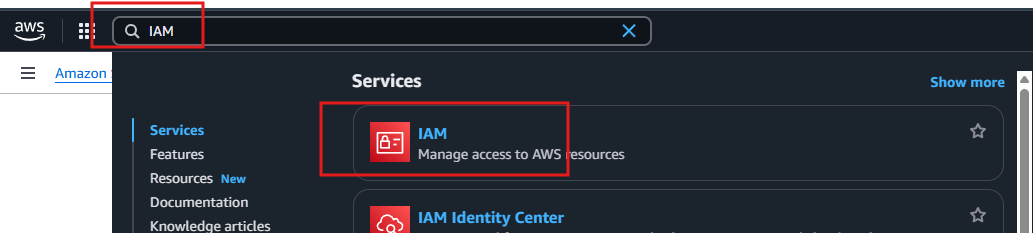
Login to the AWS account. Create a new account if don’t have a user account.  
  
Lets start creating S3 bucket  
Open the AWS Console  
search for S3 and open S3 dashboard. Click on create Bucket.  


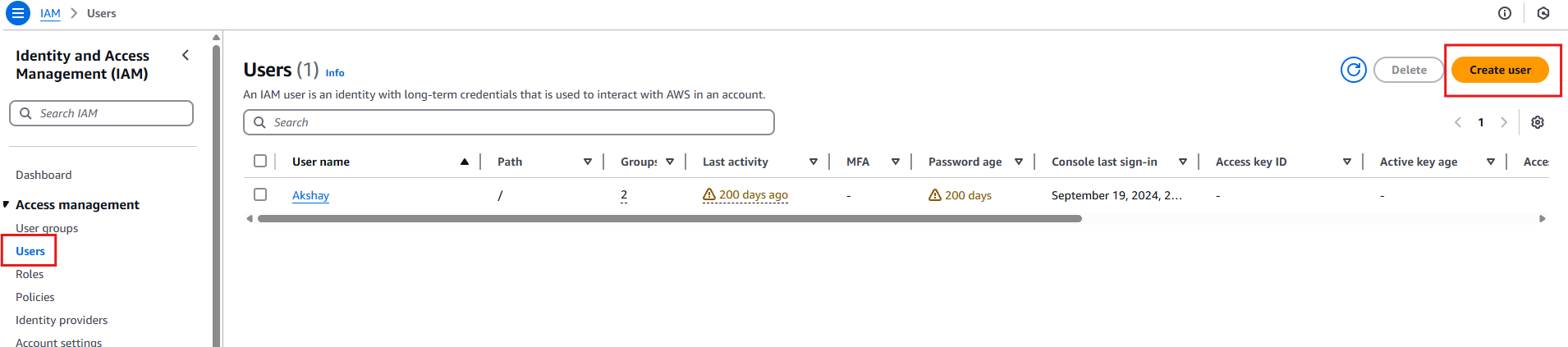
Name your bucket, create a general purpose bucket keep the settings default encryotion and create a bucket.  
  




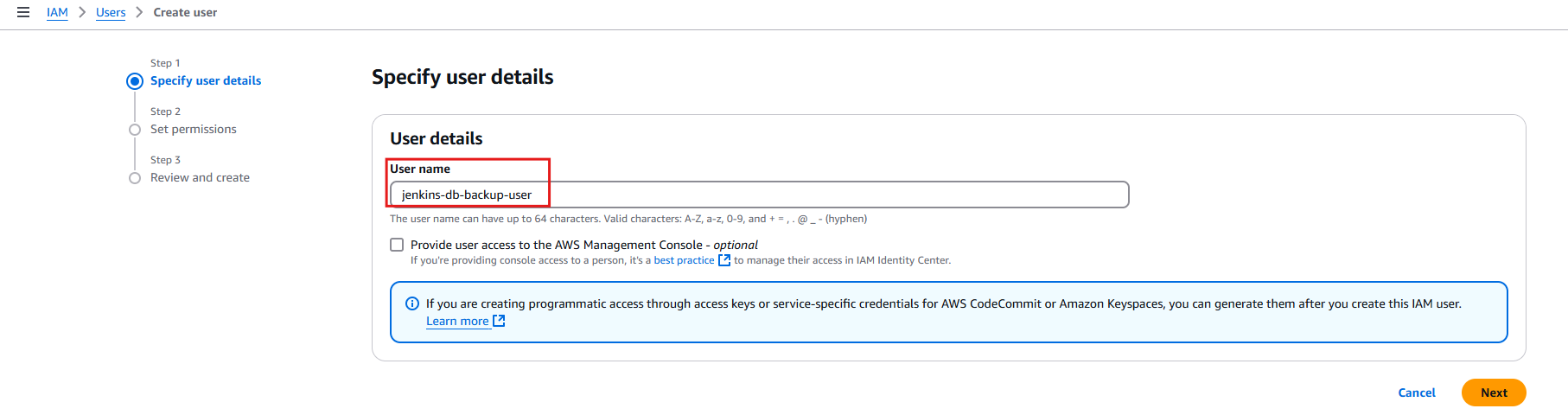
Click on create bucket.  


We have our new bucket created.  


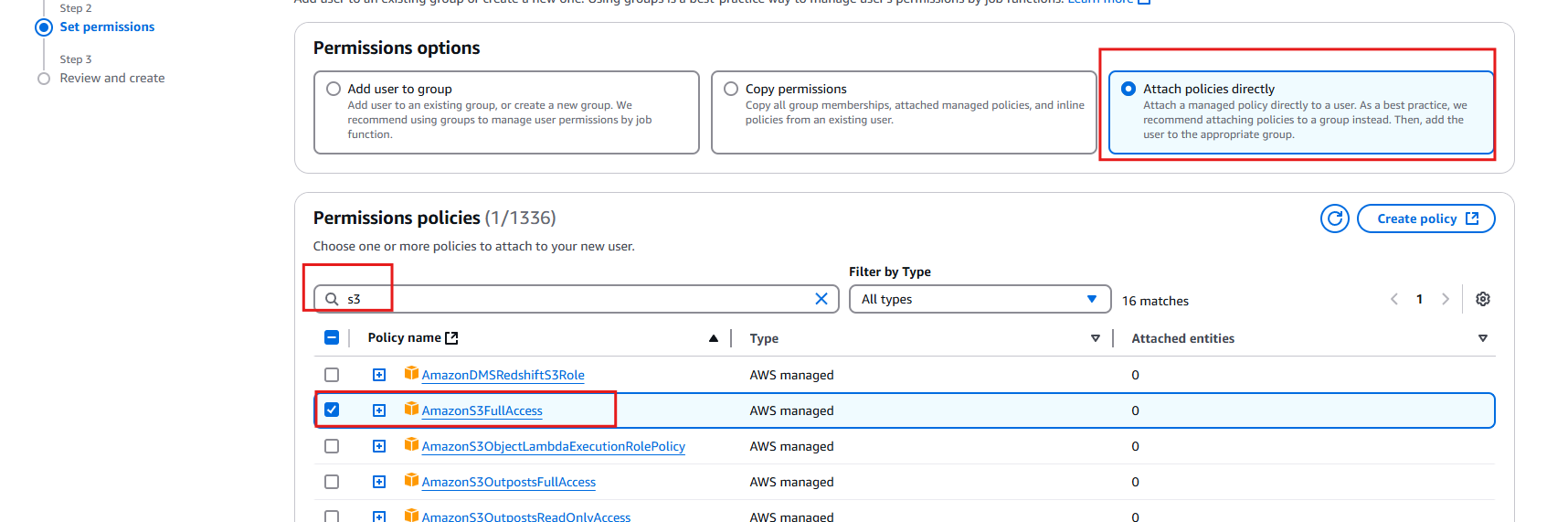
Now lets create a user with programmatical keys acces and full access to S3 using IAM permission.  
Open IAM through AWS console.  
  


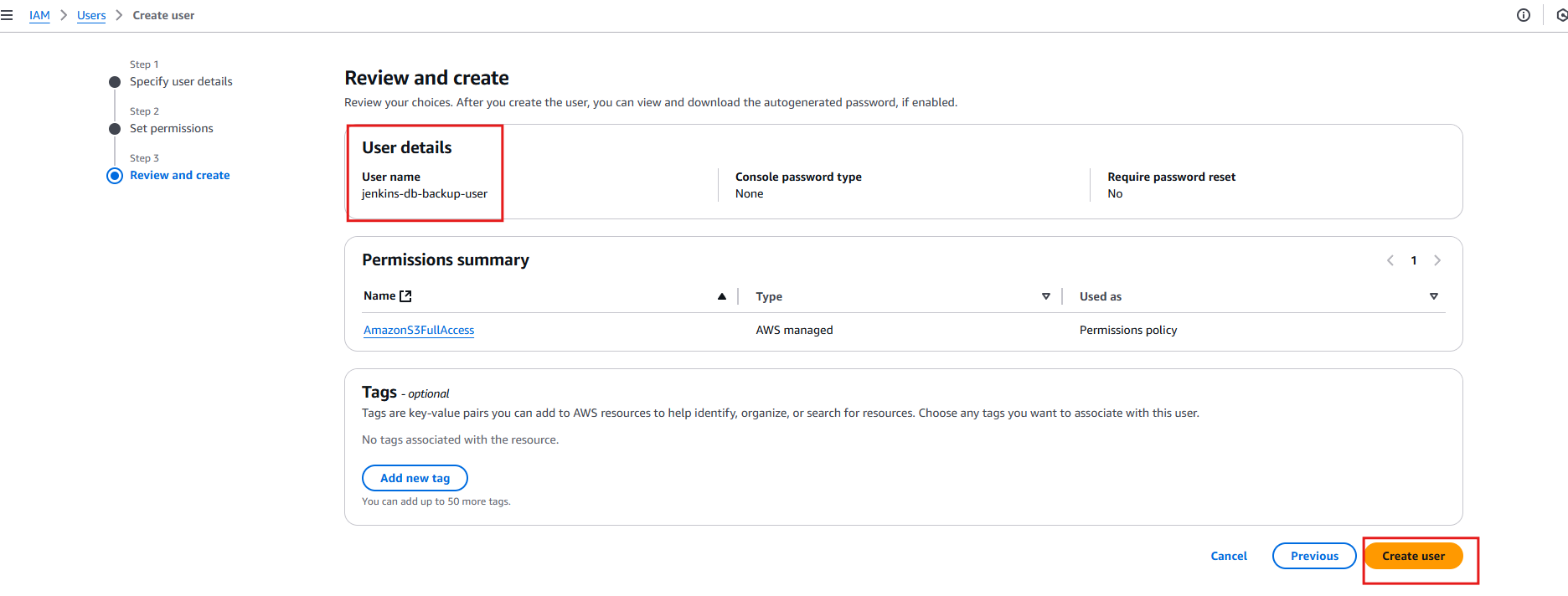
Click on users and create a new user.  


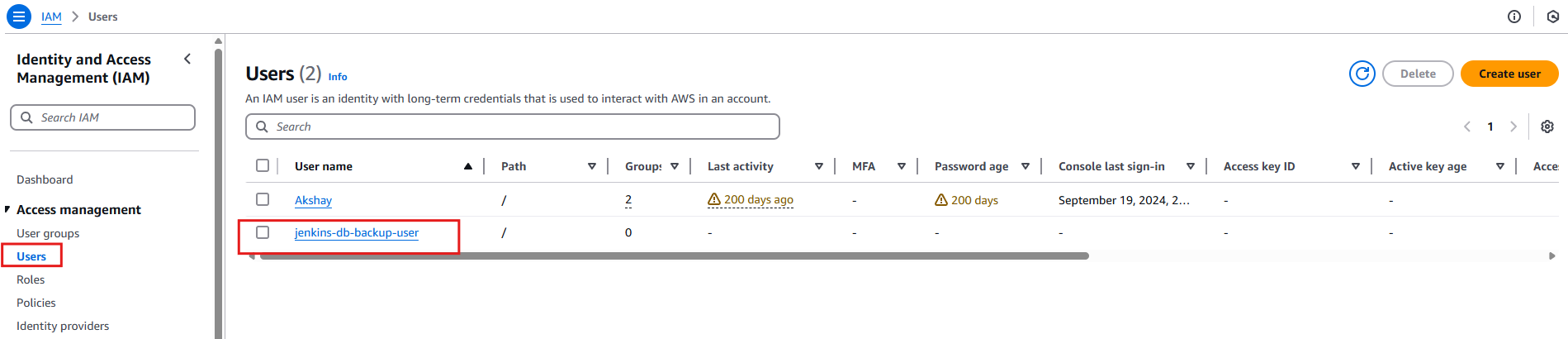
Name your user

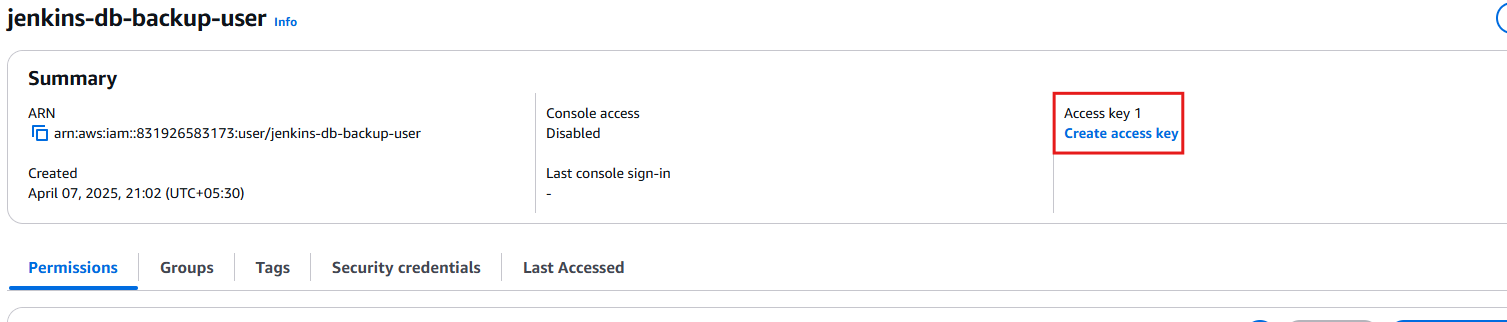


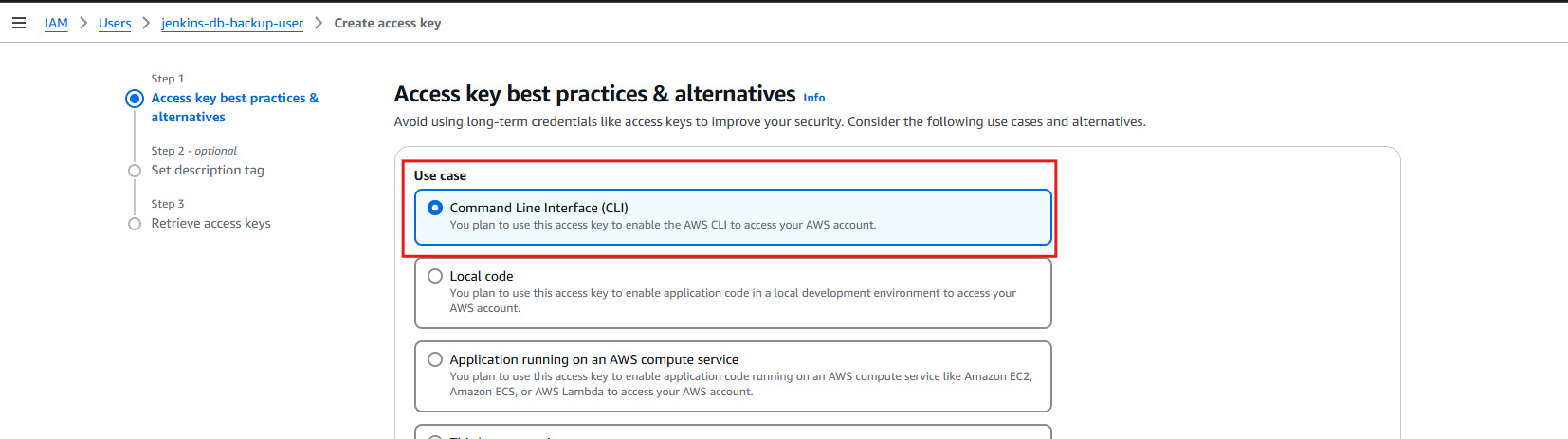
Click on attach policies and add AmazonS3FullAccess.  
click on next

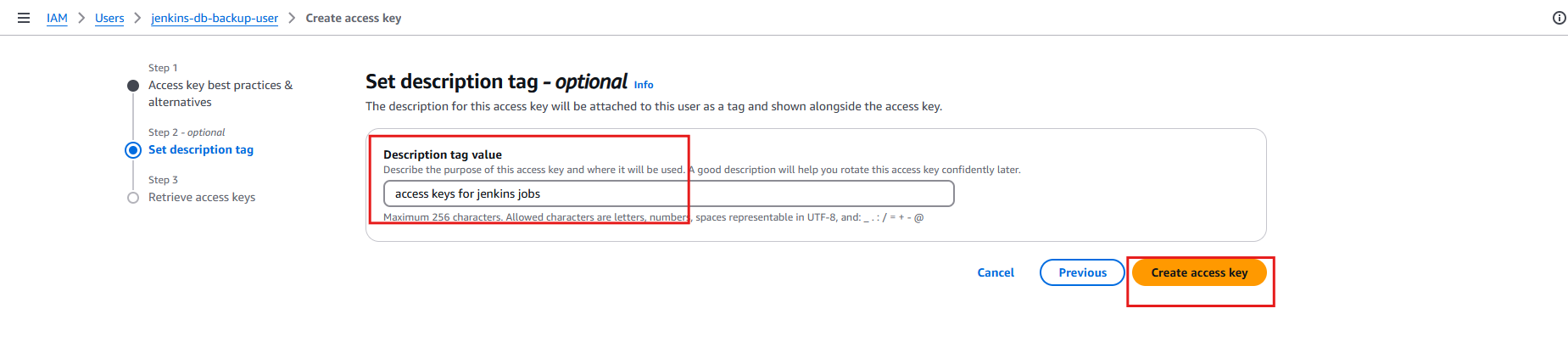


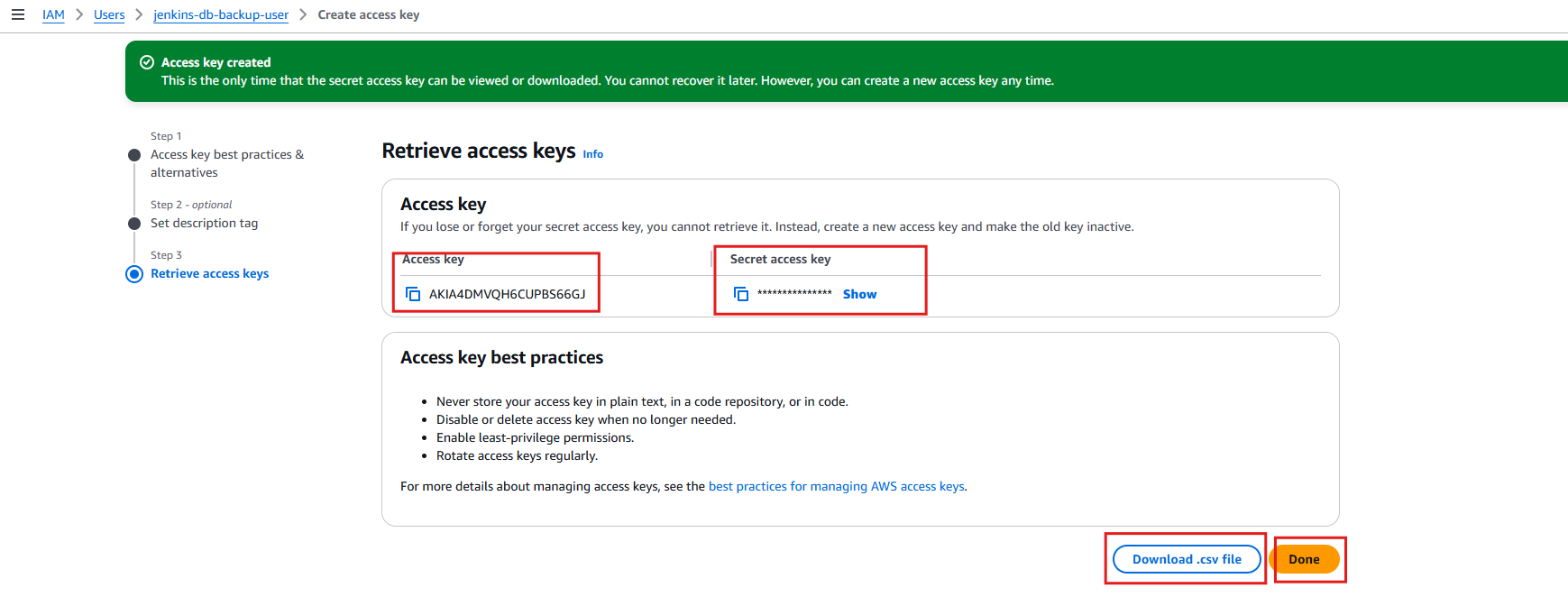
Click on create user.  


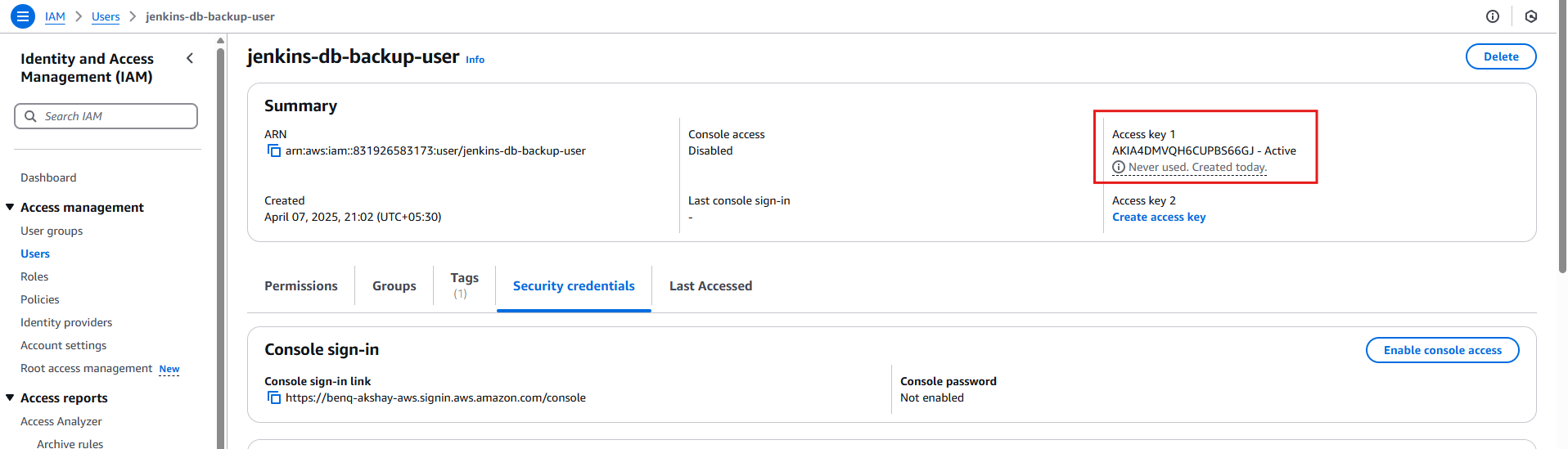
Go to users on dashboard.  
click on the user created and click on create access keys.  




Give Command Line Interface use case (CLI)  
click on Next  


Add the description and create access keys.  


Save the access keys generated or download the csv file.  
save both the access keys and secret access keys.  
(Note: Secret access keys will be only displayed once, make sure it is saved somewhere safe.)  
  


Now you have the access keys created.  


Lets create a bash script on our remote-host container.  
This bash script will be used by Jenkins to configure and automate the backup of the job.  
copy the access keys of user from AWS  
as mentioned in above image  
  
>docker exec -ti remote-host bash  
>vi /tmp/script.sh  
add the below script  
  
  
///\*  
#/bin/bash

DATE=$(date +%H-%M-%S)

BACKUP=db-$DATE.sql

DB\_HOST=$1

DB\_PASSWORD=$2

DB\_NAME=$3

AWS\_SECRET=$4

BUCKET\_NAME=$5

mysqldump -u root -h $DB\_HOST -p$DB\_PASSWORD $DB\_NAME > /tmp/$BACKUP && \

export AWS\_ACCESS\_KEY\_ID=AKIA4DMVQH6CZI53LQ7V && \ # add your Aws access keys here

export AWS\_SECRET\_ACCESS\_KEY=$AWS\_SECRET && \

echo "uploading the db dump.. details: $BACKUP" && \

aws s3 cp /tmp/$BACKUP s3://$BUCKET\_NAME/$BACKUP && \

echo "Upload successful."

\*///

>save the file

Make the file executable  
>chmod +x /tmp/script

Aws cli environment variables.  
<https://docs.aws.amazon.com/cli/v1/userguide/cli-configure-envvars.html>  
<https://docs.aws.amazon.com/cli/v1/userguide/cli-configure-envvars.html>  
  
Explanation of the shell script:  
This tells the system to use the **Bash shell** to execute the script. It should ideally be written as #!/bin/bash (note the extra / at the beginning). This is essential for ensuring the correct interpreter is used when running the script.  
And generates a timestamp in the format Hour-Minute-Second (e.g., 14-32-08) and assigns it to the variable DATE. Then it creates a filename for the database backup, such as db-14-32-08.sql, and stores it in the BACKUP variable. This ensures that each backup has a unique name.  
  
These lines assign values from the script’s input arguments to variables:

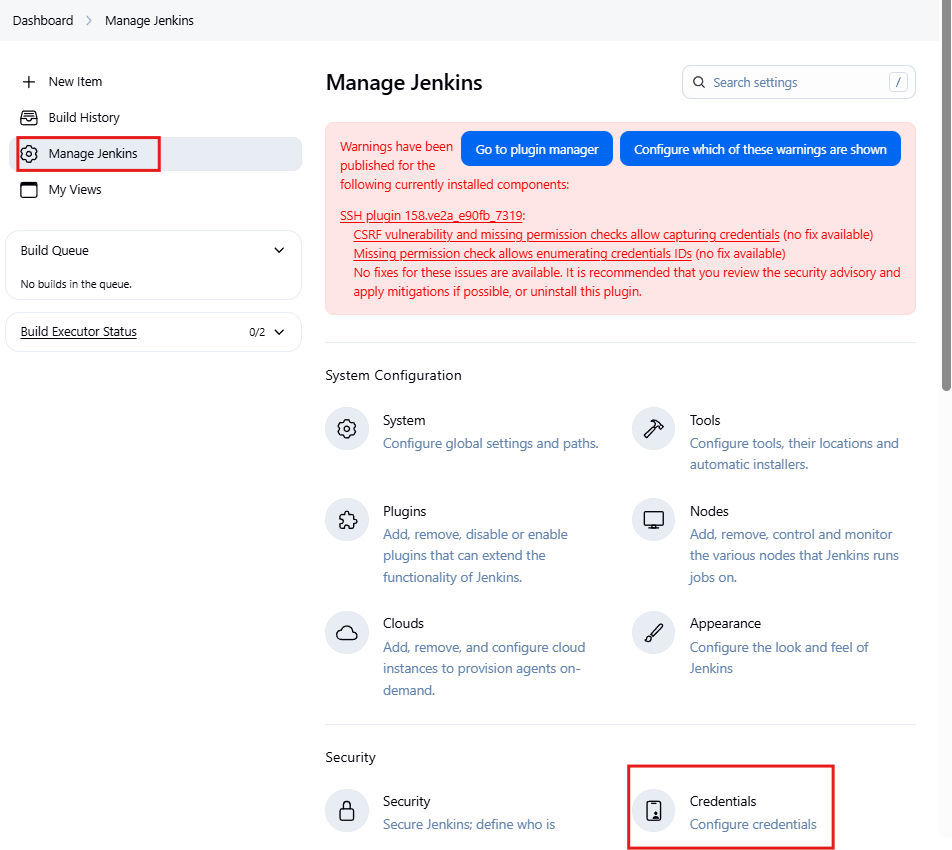
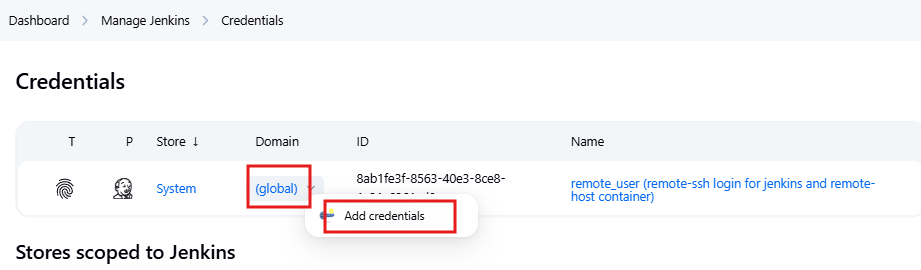
* $1 → Database host
* $2 → Database root password
* $3 → Database name
* $4 → AWS secret access key
* $5 → AWS S3 bucket name

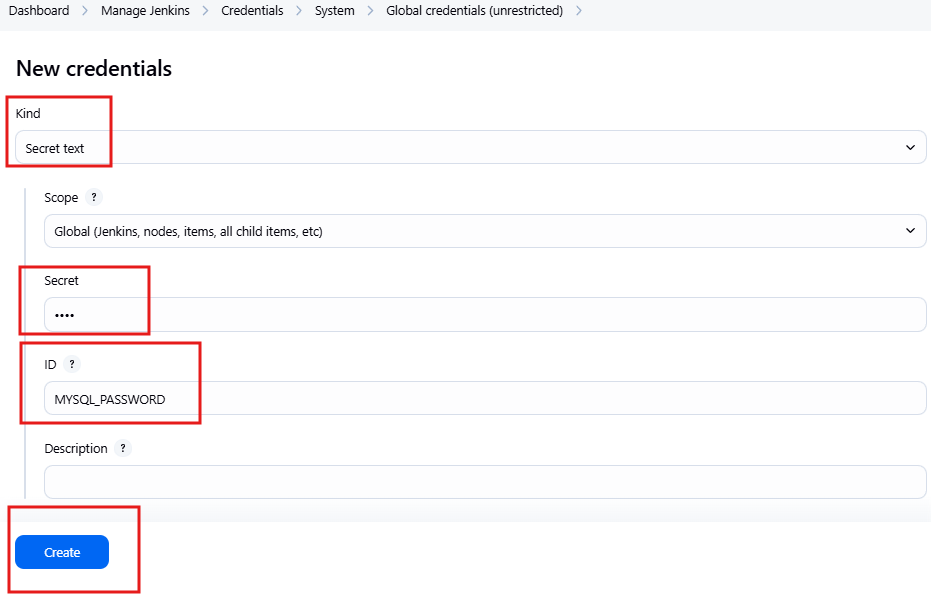
These allow the script to be reused with different values without changing the script itself.

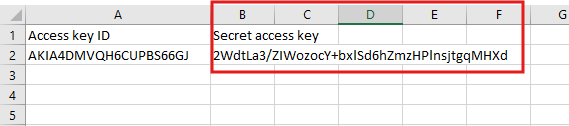
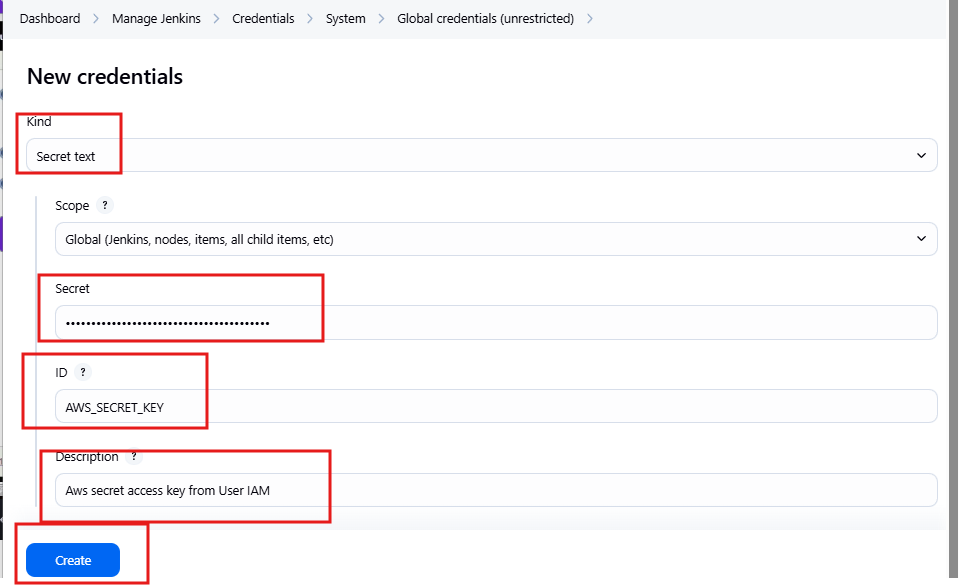
Mysql command uses mysqldump to create a backup of the specified MySQL database and saves it to the /output directory with the generated backup filename. It authenticates as the root user using the password provided.

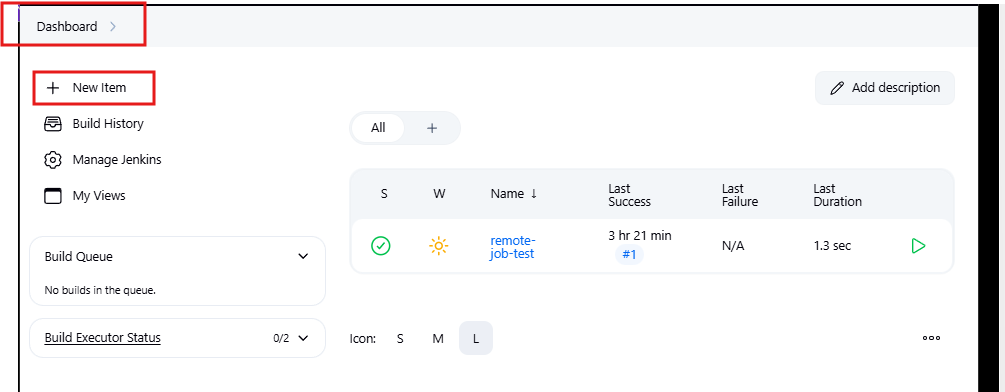
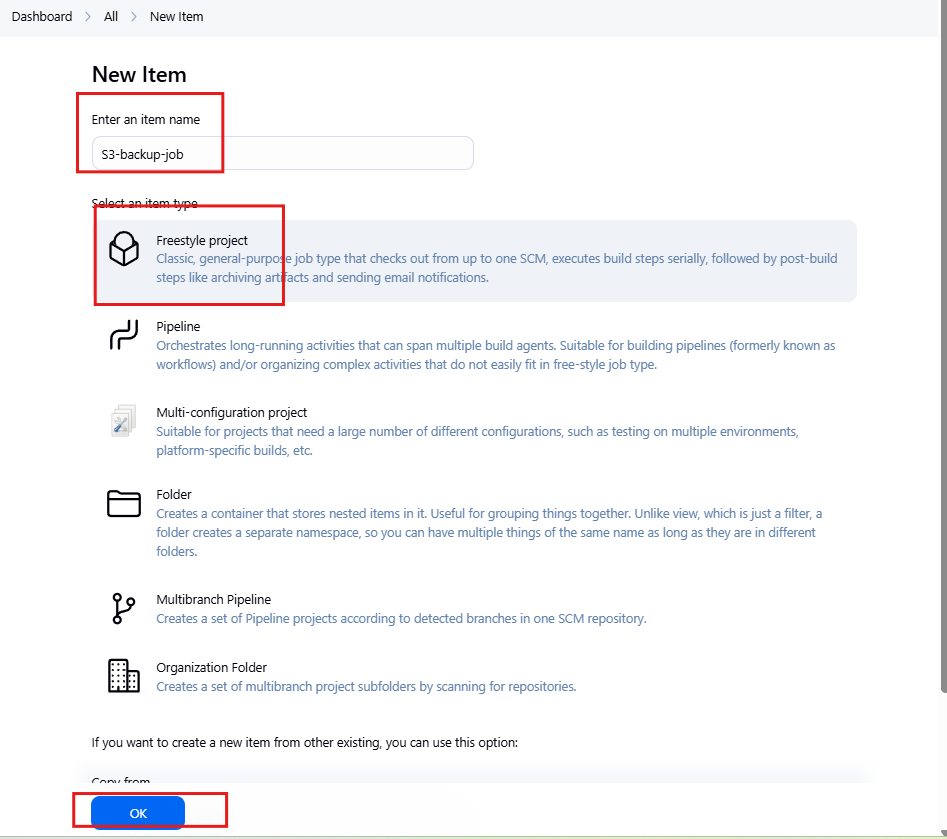
Then the AWS credentials are set as environment variables, so the AWS CLI can authenticate and upload the file to an S3 bucket. The access key ID is hardcoded (which isn't recommended), while the secret is passed as a parameter.  
final section prints a message, uploads the SQL dump to the specified S3 bucket using aws s3 cp, and then confirms the upload with a success message.

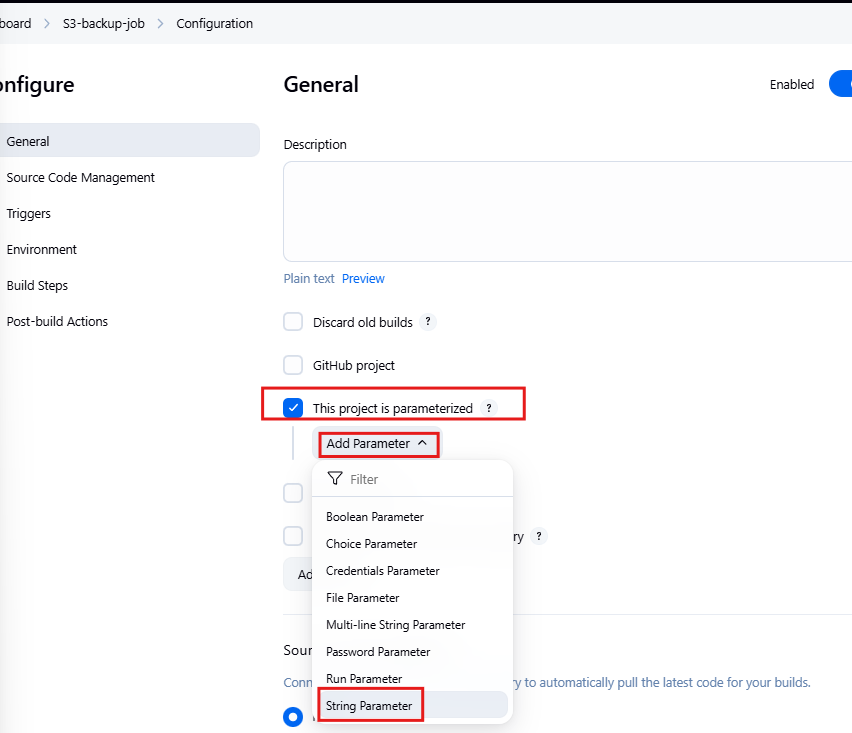
Now Lets automate this Backup through jenkins.  
first we will add our important credentials so we can automate it with our Jenkins job, securely and easily connect them with parameters.

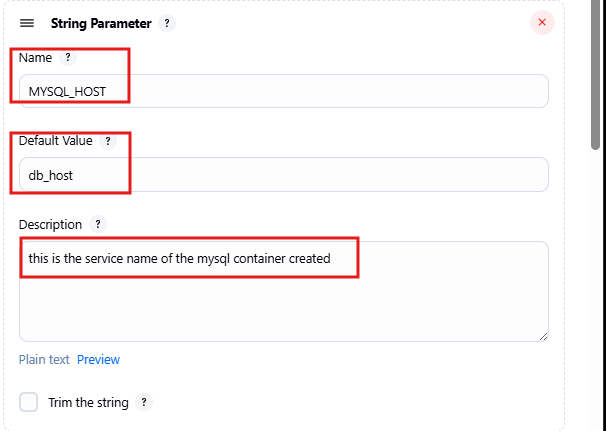
open Jenkins.  
open in a new tab  
>yourip:8080  
Login and click on manage Jenkins -> credentials  
  
  
  
click on global -> add credentials.  


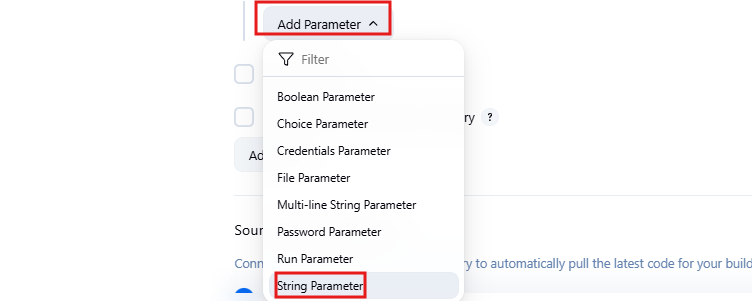
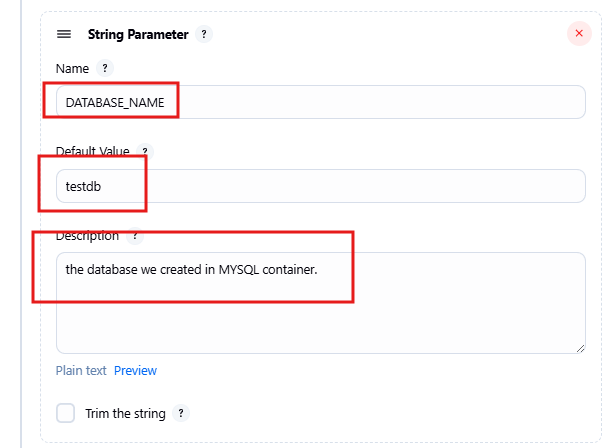
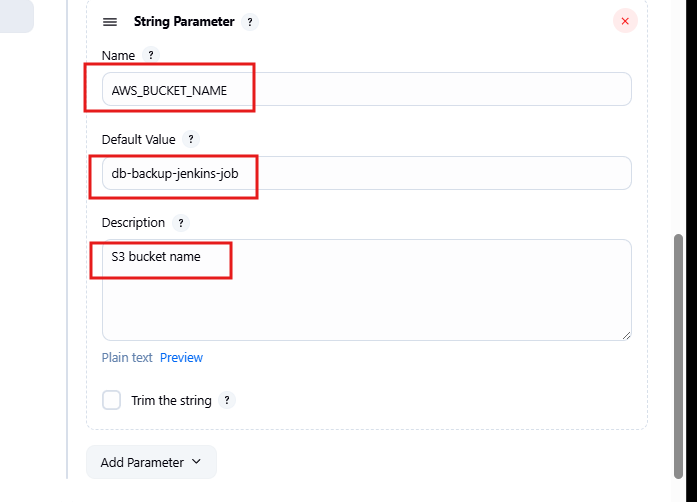
create a secret text and give MYSQL container password, and ID  
click on create  
  


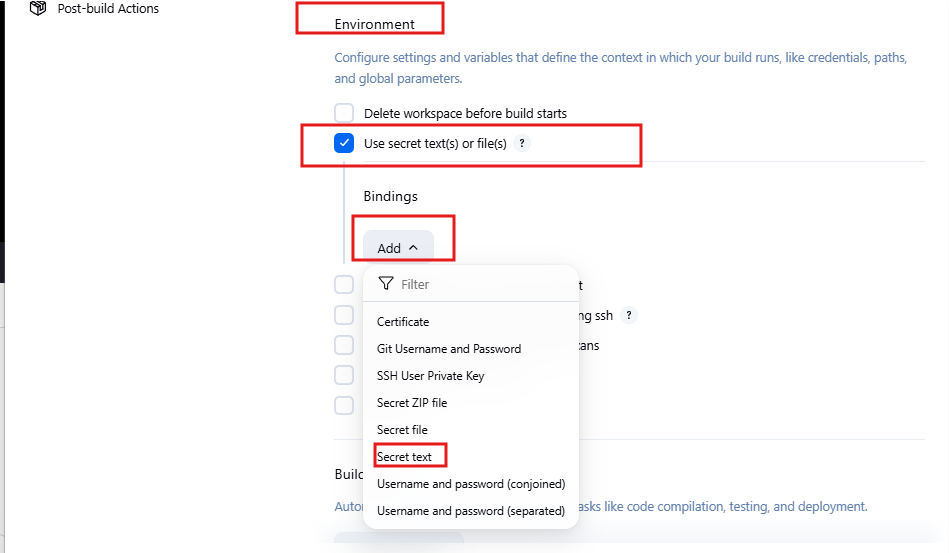
now lets create credentials for AWS keys.  
we will copy only the aws secret access key, which we have downloaded in csv file.  
 

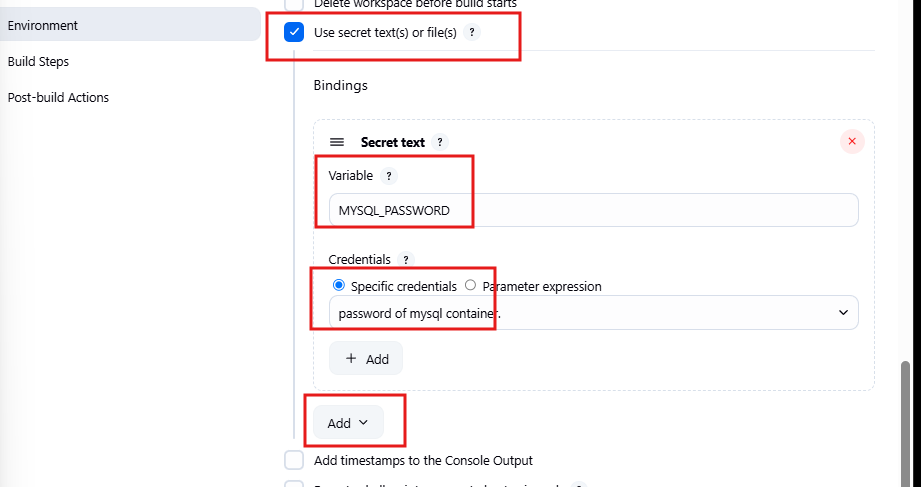
Now Lets create the job , so that we can take backup of MYSQL db in Amazon S3.  
  
go to Jenkins dashboard  
  
click on new item   
  
  
name the job -> create freestyle project -> click on ok  
  


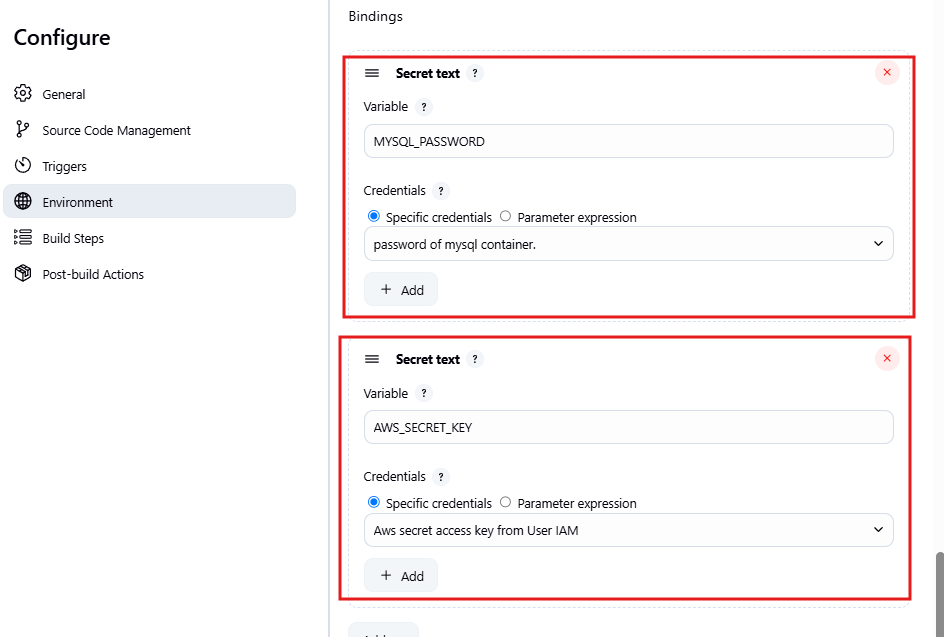
In general section 🡪 this project is parameterized ->add parameter -> string parameter  
  


As per the script lets add the first parameter.  
DB\_HOST  
enter the name,default value and description  
  
  
  
lets add more parameters.   
click on add paramters-> string parameter

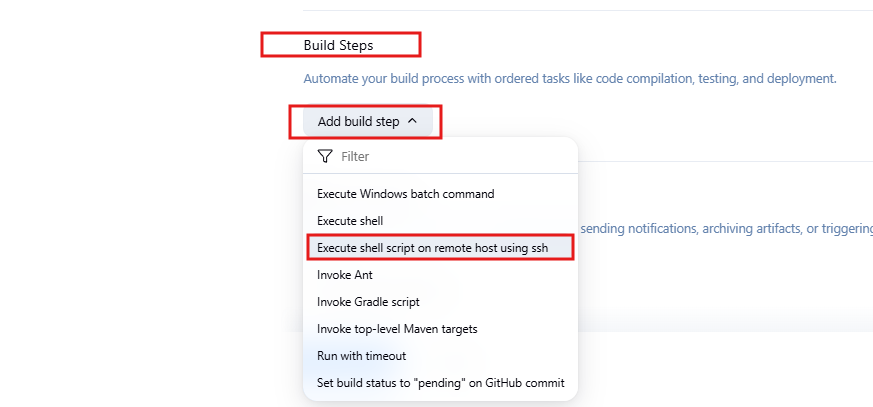
  
  
add s3 bucket parameter , take the name of the s3 bucket name from the dashboard.  


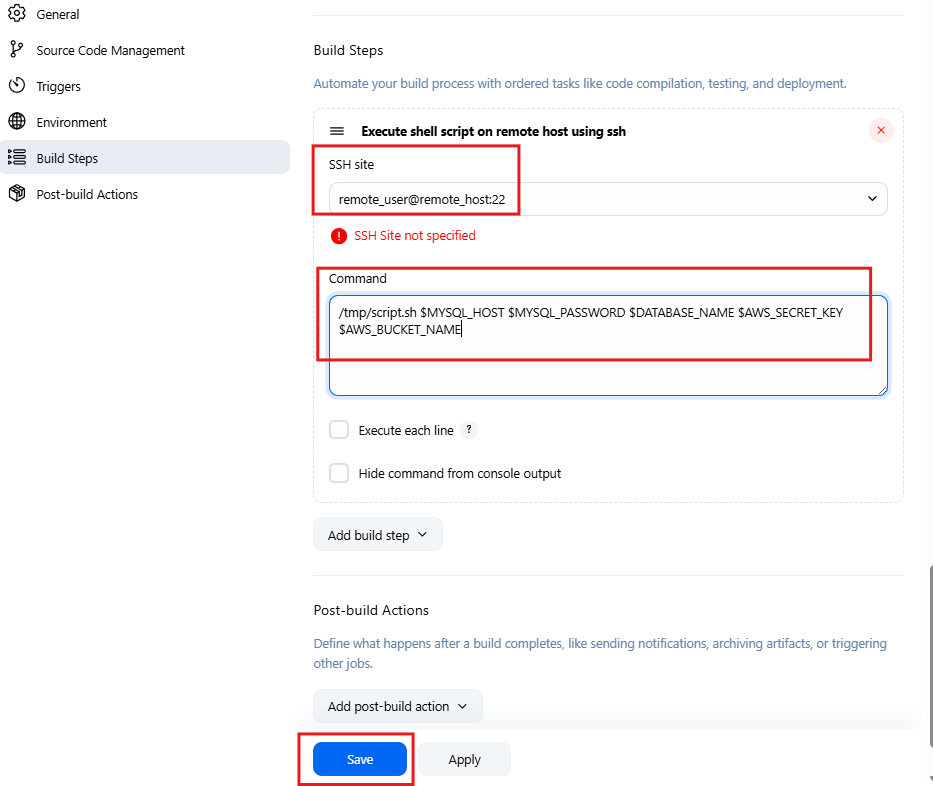
To add the MYSQL password of container and AWS secret keys.  
  
go to environment -> use secret texts.->click on add -> secret text and add the credentials we created.  
  


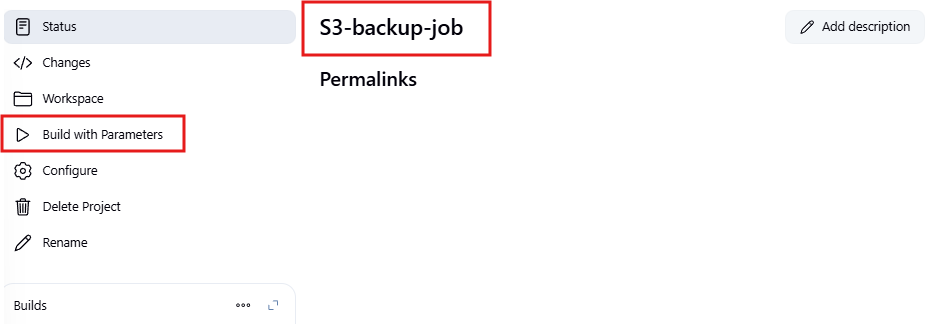
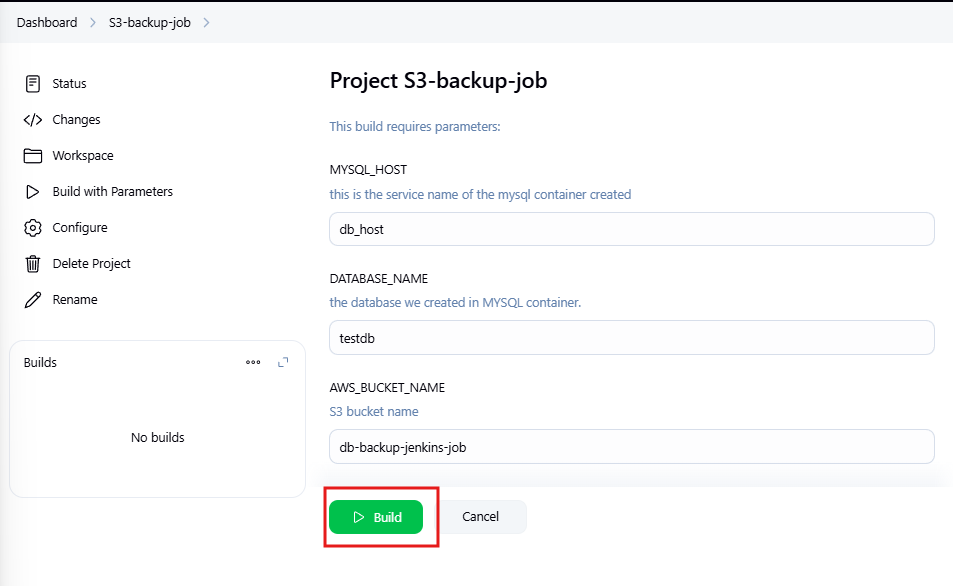
Give them variable name and add the credentials for mysql and click on add to add the aws scret keys again.  


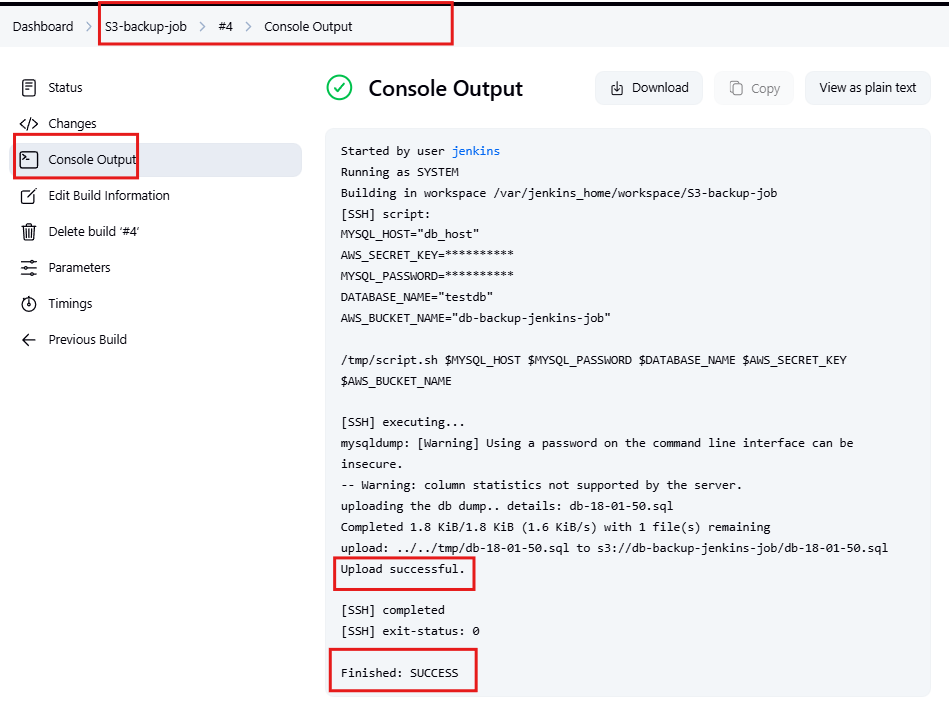
  
Now all the 5 parameters are done.  
  
Lets complete the job to execute the shell script we created in remote machine.

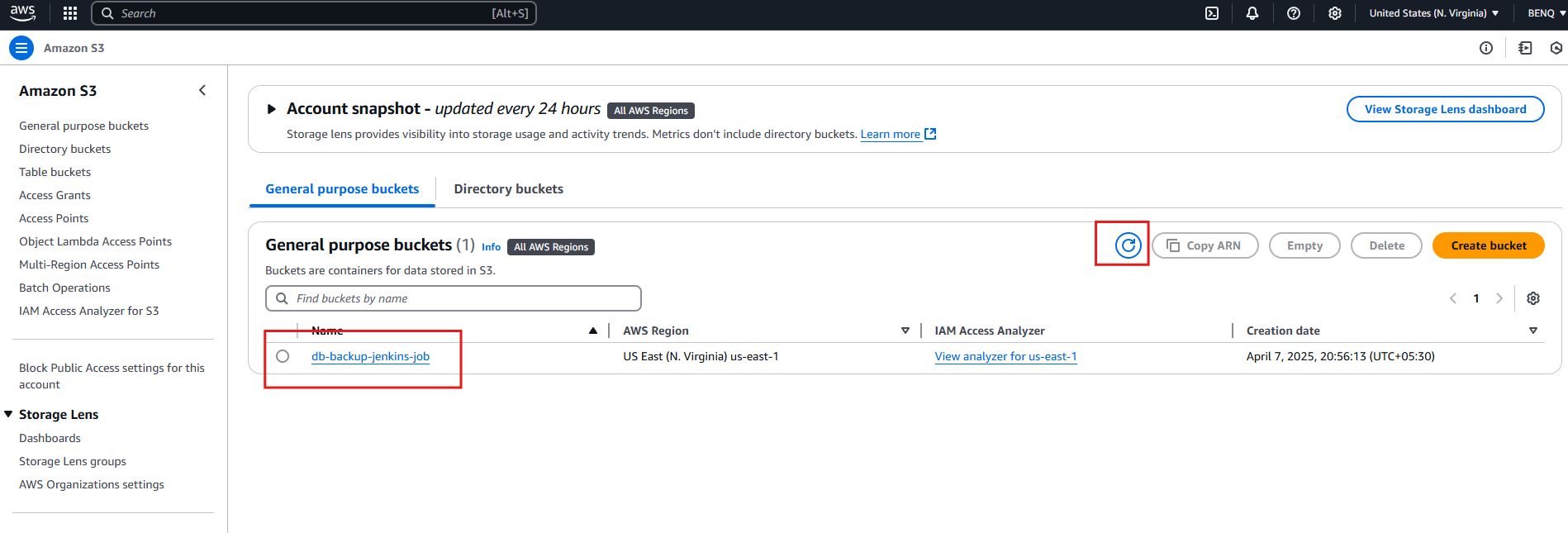
Click on execute shell script on remote host

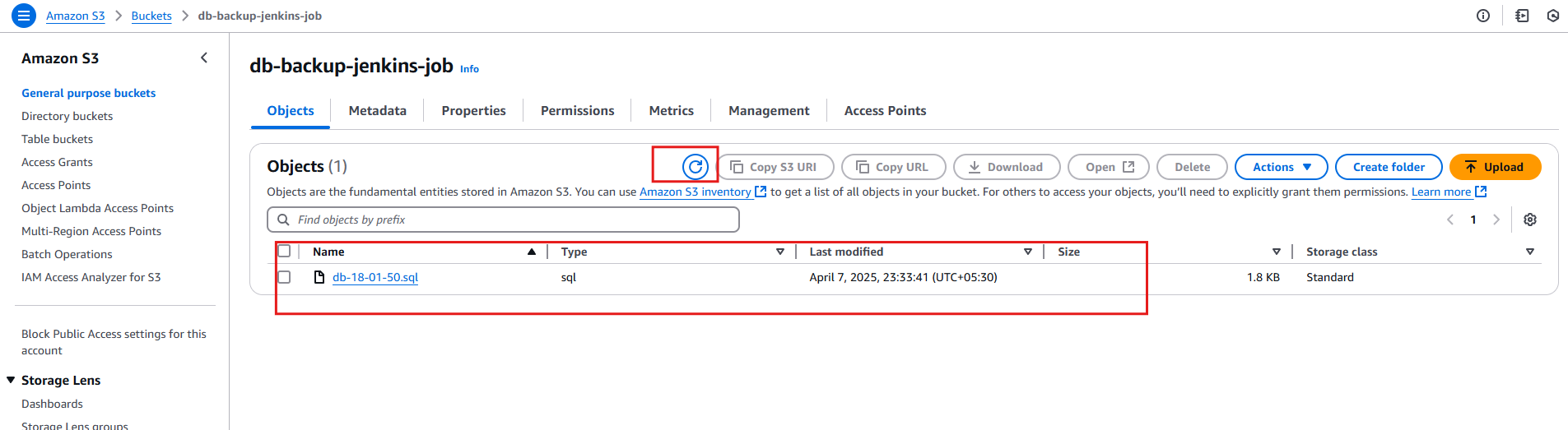


Now here we will execute the shell script created in the remote-host container.   
by entering the parameters we have created and enter them as per the shell script created.  
save the configuration  


TESTING the Jenkins jobs created.  
  
On Jenkins dashboard click on build with parameters  
  
all your parameters will be available here.  
Click on build  


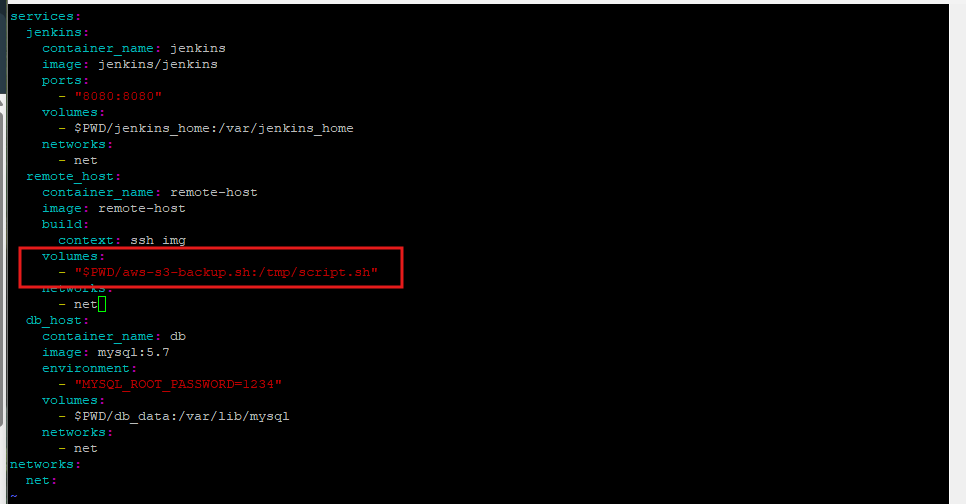
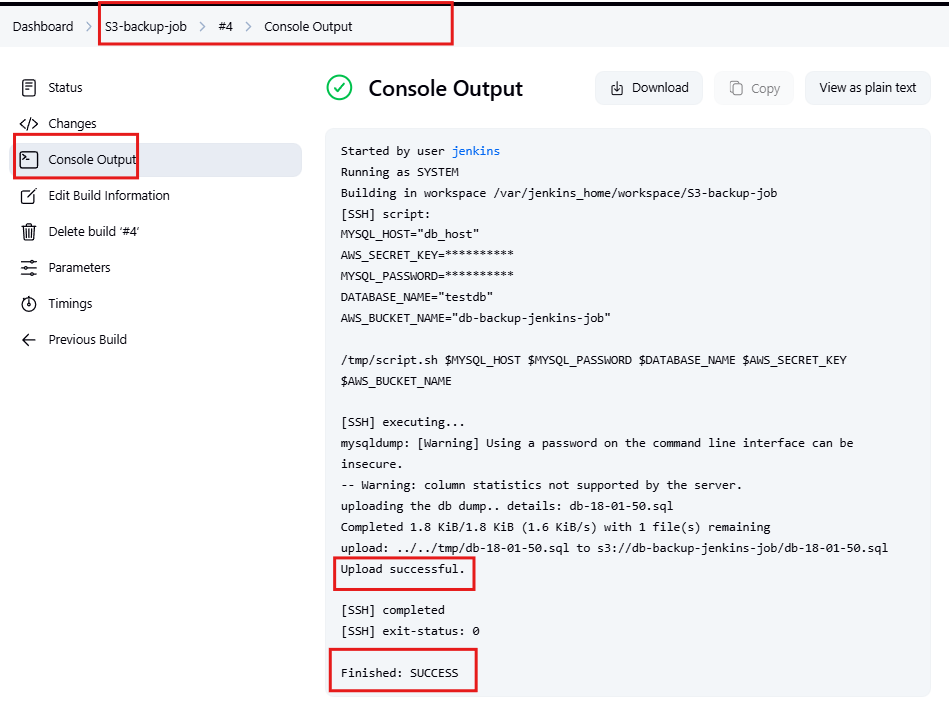
Wait and check the console output after the build completes.  
  
  
!! SUCCESS we have taken the backup of our db data in S3.  
the logs indicate the build was successful.  
  


Lets check it in the AWS S3 bucket.  
  
Go to AWS S3 bucket and refresh the bucket  
  


Your DB backup should be completed.  
  
  
  
!!!COOL!!!

We have taken the backup of our db from mysql container.  
  
since we have created the script in a ‘tmp’ folder this will be removed once the docker container is shutdown or deleted.  
best is we link the script outside the container.  
copy the above bash script created and copy it in the Jenkins\_data folder.  
  
>vi aws-s3-backup.sh  
>paste the script.  
  
modify the docker compose file  
>vi docker-compose.yml  
look for the remote host and add the volume  
> volumes:

- "PWD/aws-s3-backup.sh:/tmp/script.sh"

This will make our script permanently mounted to this directory.  
  
it should look like below  
  
  
  
  
>save the file  
run docker compose to make the changes on to the container.  
>docker compose up -d  
  
This will make the file permanent in the remote-host container /tmp folder.  
  
<https://docs.aws.amazon.com/cli/v1/userguide/cli-configure-envvars.html>  
  
Final output:  
  
the build is successful  


And the backup has been taken to Amazon S3 bucket.

