# Practice M4: Jenkins

## Part 1

### Preparation

To save some time, we can use the archive **M4-Practice-Jenkins (code).zip** provided with the practice

We can create a working folder

Extract the archive there

Navigate to the folder

You can select either **CentOS** or **Debian** based environment

Finally, create a new **Vagrantfile** by executing the following command

**cp Vagrantfile-1 Vagrantfile**

Alternatively, we can create manually a new **Vagrantfile** with the following content

# -\*- mode: ruby -\*-

# vi: set ft=ruby :

Vagrant.configure(2) do |config|

  config.ssh.insert\_key = false

  config.vm.define "jenkins" do |jenkins|

    jenkins.vm.box="shekeriev/centos-stream-9"

    jenkins.vm.hostname = "jenkins.do1.lab"

    jenkins.vm.provider :virtualbox do |vb|

      vb.customize ["modifyvm", :id, "--memory", "3072"]

    end

    jenkins.vm.network "private\_network", ip: "192.168.99.100"

    jenkins.vm.network "forwarded\_port", guest: 80, host: 8000

    jenkins.vm.network "forwarded\_port", guest: 8080, host: 8080

    jenkins.vm.provision "shell", path: "add\_hosts.sh"

  end

end

We will need one more file - **add\_hosts.sh** with the following content

#!/bin/bash

echo "192.168.99.100 jenkins.do1.lab jenkins" >> /etc/hosts

echo "192.168.99.101 docker.do1.lab docker" >> /etc/hosts

Let us run the environment with the usual command

**vagrant up**

And then create SSH session to the **jenkins** machine

**vagrant ssh jenkins**

### Jenkins Installation

#### CentOS

To download the repository information, we must execute the following

**sudo wget https://pkg.jenkins.io/redhat/jenkins.repo -O /etc/yum.repos.d/jenkins.repo**

Then, we must import the repository’s key

**sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key**

*We can choose another repository, for example the one containing only stable long term supported packages. This can be achieved by substituting the word* ***redhat*** *with* ***redhat-stable*** *in the last two commands. In the same way, we can install* ***Jenkins*** *on other distributions by substituting the word* ***redhat*** *with* ***debian*** *or* ***opensuse***

*Alternative installation method is to download a specific package from* ***https://jenkins.io/download/*** *and install it*

We can refresh repository information by executing

**sudo dnf makecache**

*Should we want not the latest version but a specific one, we can check the list of available versions with*

***dnf list --showduplicates jenkins***

*And then to install it with* ***sudo dnf install jenkins-2.60-1.1***

Let us install the latest version of **Jenkins**

**sudo dnf install jenkins**

*Good practice is to disable the* ***Jenkins*** *repository after the installation. This way, we can guarantee ourselves that no accidental update of* ***Jenkins*** *would happen*

*Disable action is achieved via the following command* ***sudo dnf config-manager --disablerepo jenkins***

If there is a problem downloading the latest version from the repositories, we can always try to download the package from the site, or switch to the stable repository

After the installation is complete, we must install **Java** as well

We can use either **Java** version 11 or 17 (recommended)

For **Java 11** execute this command

**sudo dnf install java-11-openjdk**

And for **Java 17** this one

**sudo dnf install java-17-openjdk**

Now, we can start the **Jenkins** service

**sudo systemctl start jenkins**

Let us check its status

**systemctl status jenkins**

And finally, enable the service to start automatically on boot

**sudo systemctl enable jenkins**

To be able to work with **Jenkins** from the browser on our host, we must enable port **8080/tcp** in the firewall

**sudo firewall-cmd --permanent --add-port=8080/tcp**

We will need one more port

**sudo firewall-cmd --permanent --add-port=80/tcp**

And finally, reload firewall’s configuration

**sudo firewall-cmd --reload**

*Depending on the* ***SELinux*** *settings on our virtual machine, we may need to execute the following set of commands*

***sudo dnf install selinux-policy-devel setroubleshoot-server***

***sepolicy network -t http\_port\_t***

***sudo semanage port -a -t http\_port\_t -p tcp 8080***

***sepolicy network -t http\_port\_t***

#### Debian

First, update the package cache

**sudo apt-get update**

And then install either **Java** version **11**

**sudo apt-get install fontconfig openjdk-11-jre**

Or **Java** version **17**

**sudo apt-get install fontconfig openjdk-17-jre**

Download the key

**curl -fsSL https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee \**

**/usr/share/keyrings/jenkins-keyring.asc > /dev/null**

Create record for the repository

**echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \**

**https://pkg.jenkins.io/debian binary/ | sudo tee \**

**/etc/apt/sources.list.d/jenkins.list > /dev/null**

Update package information and install the software

**sudo apt-get update**

**sudo apt-get install jenkins**

The above will install the latest release from the weekly channel

*Should we want the LTS release, we must add the following key*

***curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \***

***/usr/share/keyrings/jenkins-keyring.asc > /dev/null***

*And repository*

***echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \***

***https://pkg.jenkins.io/debian-stable binary/ | sudo tee \***

***/etc/apt/sources.list.d/jenkins.list > /dev/null***

#### Post-installation

Now, we can open a browser tab on out host machine and navigate to

[**http://localhost:8080**](http://localhost:8080)

From the loaded page, we can see that before we can start working with **Jenkins**, we must unlock it first

This can be done by copying the content of the listed file and pasting it into the text field

On the **jenkins** virtual machine console, type the following to switch to the **root** user

**su -**

The password is **vagrant**. Now, navigate to the folder

**cd /var/lib/jenkins/secrets**

Display the file’s content with

**cat initialAdminPassword**

Now exit the **root** session with

**exit**

Of course, the above steps can be shrunk to one

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

No matter which way we chose, we must copy the code and paste it in the web page

Then, confirm with the **Continue** button

On the next screen, click on the **Install suggested plugins** option

After the plugins’ installation is complete, we will be asked to create the new administrator user

Let us enter the following

- Username: **doadmin**

- Password: **Password1**

- Full name: **DevOps Administrator**

- E-mail address: **doadmin@do1.lab**

Confirm with the **Save and Continue** button

Confirm with the **Save and Finish** button

Confirm the proposed default **URL**

Finally, click on **Start using Jenkins** button

We can start exploring the **Jenkins’** interface

Do not spend much time now, instead exit the application as we must do a few more adjustments

### Additional Settings

Let us check how the **jenkins** user was created

Execute the following on the **jenkins** virtual machine

**cat /etc/passwd | grep jenkins**

We can see that its shell is set to **/bin/false**

This will prevent us from executing certain set of tasks

Open the file for editing

**sudo vi /etc/passwd**

And change the shell from **/bin/flase** to **/bin/bash**

Save and close the file

*We could achieve the same with the following command*

***sudo usermod -s /bin/bash jenkins***

Now, that the shell is set as required we can set the password for the **jenkins** user to **Password1**

**sudo passwd jenkins**

Enter a session with the **jenkins** user

**su - jenkins**

Check the working directory

**pwd**

Generate an SSH key, accept all default values, and don’t set a password

**ssh-keygen -t ecdsa -b 521 -m PEM**

Copy the key to the localhost

**ssh-copy-id jenkins@localhost**

During the key copying process, you will be asked for a password, use the one for the **jenkins** user

Let us check if we can establish an SSH session without a password

**ssh jenkins@localhost**

We should have an SSH session without being asked for a password

Exit the **ssh** session with

**exit**

And then exit the **jenkins** session

**exit**

Let us add the **jenkins** user to the **sudoers** list

**sudo visudo**

Then enter the following record

**jenkins ALL=(ALL) NOPASSWD: ALL**

Save and close the file

Restart the **jenkins** service

**sudo systemctl restart jenkins**

### Security Settings

Let us open a browser tab on the host and navigate to

[**http://localhost:8080**](http://localhost:8080)

Use the **doadmin** user to authenticate

Next, navigate to **Manage Jenkins**

Then click on **Configure Global Security**

In the **Authorization** section select **Matrix-based security**

Now, add the **doadmin** user by clicking on **Add user** button

Then in the dialog box enter **doadmin** and confirm with **OK**

Select all permissions either manually or click on the **select all** button after the matrix

Confirm with the **Save** button

Now, navigate to **Manage Jenkins** and then to **Manage Users**

Click on **Create User** option to add a new user

Enter the following information

- Username: **douser**

- Password: **Password2**

- Full name: **DevOps User**

- E-mail address: **douser@do1.lab**

Confirm with the **Create User** button

Now, let us grant some permissions to the newly created user

Navigate to **Manage Jenkins** and then to **Configure Global Security**

Scroll down to the security matrix

Click on the **Add user** button to add the new user to the matrix

Enter **douser** and confirm with **OK**

Now, let us grant the following permissions

**- Overall - Read**

**- Job - Build**

**- Job - Cancel**

**- Job - Configure**

**- Job - Discover**

**- Job - Read**

**- Job - Workspace**

**- Run - Update**

**- View - Configure**

**- View - Create**

**- View - Read**

Confirm with the **Save** button

We can test the new user by logging with its account

There should be different set of available actions

Now, exit the session

### Our First Local Build

Open a browser tab on the host and navigate to

[**http://localhost:8080**](http://localhost:8080)

Log on as the **doadmin** user

Click on the **New Item** action

Select the **Folder** option

Enter **DO1-Demo** for name

Confirm with the **OK** button

And finally, close with the **Save** button

Now, let us create a new item again by selecting the **New Item** action

This time, select **Freestyle** **project**

Enter **Get-All-Processes** as name

Confirm with the **OK** button

Scroll-down to the **Build** section or click on the **Build** option in the header

Select the **Execute shell** option in the drop-down list

In the **Command** text field enter

**ps ax**

Confirm with the **Save** button

Click on the **Build Now** action in the left section

Check the result from the execution by clicking on the number (#1) in the **Build History** section

Then, click on **Console Output**

*Depending on the distribution settings of the* ***Jenkins*** *host, additional correction may be needed*

*Open the* ***sudoers*** *list with*

***sudo visudo***

*Change (or add) the following*

***Defaults !requiretty***

## Part 2

### Remote Access Settings

Exit the SSH session from the **jenkins** machine with

**exit**

Now, either change the **Vagrantfile** with another one from the archive

**cp Vagrantfile-2 Vagrantfile**

Or make sure that your **Vagrantfile** is with the following content

# -\*- mode: ruby -\*-

# vi: set ft=ruby :

Vagrant.configure(2) do |config|

  config.ssh.insert\_key = false

  config.vm.define "jenkins" do |jenkins|

    jenkins.vm.box="shekeriev/centos-stream-9"

    jenkins.vm.hostname = "jenkins.do1.lab"

    jenkins.vm.provider :virtualbox do |vb|

      vb.customize ["modifyvm", :id, "--memory", "3072"]

    end

    jenkins.vm.network "private\_network", ip: "192.168.99.100"

    jenkins.vm.network "forwarded\_port", guest: 80, host: 8000

    jenkins.vm.network "forwarded\_port", guest: 8080, host: 8080

    jenkins.vm.provision "shell", path: "add\_hosts.sh"

  end

  config.vm.define "docker" do |docker|

    docker.vm.box = "shekeriev/centos-stream-9"

    docker.vm.hostname = "docker.do1.lab"

    docker.vm.network "private\_network", ip: "192.168.99.101"

    docker.vm.provision "shell", path: "add\_hosts.sh"

  end

end

Bring up the new environment with

**vagrant up**

Now, we should have not one but two machines - **jenkins** and **docker**

Let us configure the remote access for the **jenkins** user to the **docker** station

Open a new session

**vagrant ssh docker**

Now, we must create a new **jenkins** user with **Password1** as its password

**sudo useradd jenkins**

**sudo passwd jenkins**

Grant the new user **sudo** permissions

**sudo visudo**

Enter the following

**jenkins ALL=(ALL) NOPASSWD: ALL**

Save and close the file

Close the session

**exit**

Establish an SSH session to the **jenkins** machine

**vagrant ssh jenkins**

Switch the session to the **jenkins** user

**su - jenkins**

Copy the user’s key to the **docker** station

**ssh-copy-id jenkins@docker.do1.lab**

Test the connectivity with

**ssh jenkins@docker.do1.lab**

Close the test connection with

**exit**

Exit the **jenkins** session with

**exit**

### Plugins Installation

Open a new browser tab on our host and navigate to

[**http://localhost:8080**](http://localhost:8080)

Authenticate with the **doadmin** user

Navigate to **Manage Jenkins** and then to **Manage Plugins**

Next, switch to **Available plugins** section

In the **Filter** field enter **ssh**

The available plugins list will reflect our search term

Click to select the **SSH** plugin and confirm with the **Download now and install after restart** button

Select the **Restart Jenkins when installation is complete and no jobs are running** item

After the installation and restart are complete, we must authenticate again with the **doadmin** user

### Credentials

Now, that we are logged in again, let us add some credentials

Go to **Manage Jenkins**

Click on **Manage Credentials**

Then, click on **System** (or **Jenkins**, depending on the version) and finally on **Global credentials**

Now, click the **Add Credentials** item

Make sure that the **Username with password** option is selected

Enter the following values

- Username: **jenkins**

- Password: **Password1** *(the same as we entered for the OS user)*

- Description: **Local user with password**

Confirm with the **Create** button

Let us add one more item

Click again on the **Add Credentials** item

This time, make sure that the **SSH Username with private key** option is selected

Enter the following values

- Username: **jenkins**

- Private Key: Select **Enter Directly**, then **Add** and finally paste the content from the **~/.ssh/id\_ecdsa** file (you must execute **su - jenkins** and then **cat ~/.ssh/id\_ecdsa**) on the **jenkins** machine

- Description: **Credentials from file**

Confirm with the **Create** button

### Register SSH Hosts

Let’s register the hosts which we will access via SSH

Return to the home page

Click on the **Manage Jenkins** action and then on **Configure System**

In the **SSH remote hosts / SSH sites** section click on the **Add** button

Enter the following values

- Hostname: **localhost**

- Port: **22**

- Credentials: **jenkins (Local user with password)**

Test the connectivity with the **Check connection** button

Click the **Add** button again to add a second host

Enter the following values

- Hostname: **docker.do1.lab**

- Port: **22**

- Credentials: **jenkins (Credentials from file)**

Again, test the connectivity with the **Check connection** button

Confirm the changes with the **Save** button

### Remote Task Execution

While still in the **Jenkins** UI, return to the **DO1-Demo** folder

Click on the **New Item** action

Select the **FreeStyle project** option

Enter **Get-All-Processes-Remote** as name

Confirm with **OK**

Scroll-down to the **Build** section

Select **Execute shell script on remote host using ssh** in the drop-down list

Select the second station (**docker.do1.lab**) in the **SSH site** drop-down list

In the **Command** text field enter

**hostname**

**ps ax**

Select the **Execute each line** option to execute each command in its own SSH session

Confirm with the **Save** button

Click on **Build Now** in the left menu

Check the results by clicking on the build number in the **Build History** section and then on **Console Output**

### GitHub Preparation

On the **jenkins** virtual machine add the **git** package

In **CentOS** execute

**sudo dnf install git**

And in **Debian**, execute

**sudo apt-get install git**

Then create a working folder

**sudo mkdir -p /projects/www-static**

And change its owner

**sudo chown -R jenkins:jenkins /projects**

Install **Apache** web server on the **Jenkins** host

In **CentOS** execute

**sudo dnf install httpd**

And in **Debian**, execute

**sudo apt-get install apache2**

Start it and enable it to start automatically on boot (skip it on **Debian**)

**sudo systemctl enable httpd**

**sudo systemctl start httpd**

### Working with GitHub

While still in the **Jenkins** UI, return to the **DO1-Demo** folder

Click on the **New Item** action

Select the **FreeStyle project** option

Enter **GitHub-Build** as name

Confirm with **OK**

Select the **GitHub project** option

Enter **https://github.com/shekeriev/simple-html-page.git** in the **Project URL** field

Click on the **Advanced** button

Enter **GitHub Static Web Page** in the **Display Name** field

Click once again the **Advanced** button

Select the **Use custom workspace** option

Enter **/projects/www-static** in the **Directory** field

In the **Source Code Management** section select the **Git** option

Enter **https://github.com/shekeriev/simple-html-page.git** in the **Repository URL** field

Confirm with the **Save** button

Start the task with the **Build Now** button

Check the results

### Working with GitHub #2

Return to the **DO1-Demo** folder

Click on the **New Item** action

Enter **GitHub-Build-Post** as name

In the **Copy from** field enter **GitHub-Build**

Confirm with **OK**

Add a new **Build** step - **Execute shell**

In the **Command** text box enter

**sudo cp /projects/www-static/\*.html /var/www/html && sudo cp /projects/www-static/\*.png /var/www/html**

Or a shorter version

**sudo cp -t /var/www/html /projects/www-static/\*.html /projects/www-static/\*.png**

Confirm with the **Save** button

Start the execution with the **Build Now** button

To check the result, open a browser tab on the host and navigate to <http://localhost:8000>

### Build on Schedule

Return to the initial screen of **Jenkins**

Select **Manage Jenkins** and then **Manage Plugins**

Switch to the **Available** tab and type **schedule** in the **Filter** field

Select the **Schedule Build** plugin

Confirm with the **Download now and install after restart** button

After the **Jenkins** is restarted, log back in and go to the **DO1-Demo** folder

Select the **New Item** action

Then select the **FreeStyle Project** option

Enter **Get-Processes-Count-Scheduled** for name

Confirm with **OK**

For description, we can enter **Get the running processes count at a certain point in the future**

Add **Execute shell** in the **Build** section

Enter **hostname && ps ax | wc -l** in the **Command** text field

Confirm with the **Save** button

Initiate the build with the **Schedule Build** action

Enter time for the task to be scheduled for execution (for example, 2 minutes in the future) and confirm with **Schedule**

Return to the **Dashboard**

There should be a task in the **Build Queue**

Click on the task to return back to our scheduled job

We can sit back and watch how the execution will go

### Build on Schedule #2

Navigate to the **Configure** section of the previous job

In the **Build Triggers** section select **Build periodically**

Enter **H/2 \* \* \* \*** in the **Schedule** field to execute job every 2 minutes

Confirm with the **Save** button

We can sit back and watch how the execution will go

Once we see the results, we must turn off the schedule before continuing with the next tasks

To disable the schedule, navigate to **Configure**

Delete the content of the **Schedule** field

De-select the **Build periodically** option

Confirm with the **Save** button

### Build on Schedule with GitHub

Navigate to the **DO1-Demo** folder again

Click on the **New Item** action

Create a copy of the **GitHub-Build-Post** job with a new name set to **GitHub-Build-Trigger**

Confirm with **OK**

When in the new job, navigate to the **Build Triggers** section

Select the **Poll SCM** option

Enter **\* \* \* \* \*** in the **Schedule** text field

Confirm with the **Save** button

We can do and commit a change to the project in **GitHub** watch what will happen

We may explore the **Git Polling Log** item as well

Before we continue further, we must stop the schedule

Navigate to **Configure**

Delete the content of the **Schedule** field and turn off the **Poll SCM** option

Confirm with the **Save** button

## Part 3

### Preparation

Make sure that you are on the second virtual machine

Follow the instructions to install **Docker**

#### On CentOS

Add the repository

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

To install **Docker**, execute the following:

**sudo dnf install docker-ce docker-ce-cli containerd.io**

Let us start the **docker** daemon and check its status

**sudo systemctl enable --now docker**

Add both the vagrant and **jenkins** users to the **docker** group

**sudo usermod -aG docker vagrant**

**sudo usermod -aG docker jenkins**

Test with (you may need to close and reopen the session, otherwise **sudo** should be used)

**docker version**

#### On Debian

Add the prerequisites

**sudo apt-get update**

**sudo apt-get install ca-certificates curl gnupg lsb-release**

Then add the repository key

**curl -fsSL https://download.docker.com/linux/debian/gpg \**

**| sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg**

Finally, add the repository

**echo \**

**"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian \**

**$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

To install **Docker**, first we must update package information

**sudo apt-get update**

And then install the necessary packages

**sudo apt-get install docker-ce docker-ce-cli containerd.io**

Finally, Add both the vagrant and **jenkins** users to the **docker** group

**sudo usermod -aG docker vagrant**

**sudo usermod -aG docker jenkins**

Test with

**docker version**

### Docker Job #1 (Hello World)

Return to the initial page of the **Jenkins** portal to create our first job that will include **docker**

Navigate to the **DO1-Demo** folder

Click on the **New Item** action

Select the **FreeStyle** job type

Enter **Docker-Hello-World** as job name

Enter the following as description **Docker hello world on a remote node**

Scroll-down to the **Build** section

Select the **Execute shell script on remote host using ssh** to add a new build step

As SSH site select **jenkins@docker.do1.lab:22**

Enter **docker container run shekeriev/welcome-do:latest** in the **Command** text field

Confirm with **Save**

Initiate the execution with **Build Now**

Check the execution progress and result

### Docker Job #2 (Static Site)

#### Preparation

We should do a few things on the **second** host

Open a session

**vagrant ssh docker**

Then install the **git** package

In **CentOS** execute

**sudo dnf install git**

And in **Debian**, execute

**sudo apt-get install git**

Create a working folder

**sudo mkdir -p /projects/www-static**

And adjust its ownership

**sudo chown -R jenkins:jenkins /projects**

#### Actual steps

Return to the **Jenkins** portal

Navigate to the **DO1-Demo** folder

Click on the **New Item** action

Select the **FreeStyle Project** job type

Enter **Docker-GitHub** for the name

Confirm with **OK**

Enter **Build and run Docker container from GitHub** as job description

Select the **GitHub project** option

Enter the **https://github.com/shekeriev/simple-docker-image.git** in the **Project url** field

Click the **Advanced** button

Select the **Use custom workspace** option

Enter **/projects/www-static** as a custom workspace directory

Scroll down to the **Source Code Management** section

Select **Git**

Enter **https://github.com/shekeriev/simple-docker-image.git** as a **Repository URL**

Scroll down to **Build**

Select **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**cd /projects/www-static**

**docker image build -t img-static-site .**

Select again **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**docker container rm -f co-static-site || true**

**docker container run -d -p 80:80 --name co-static-site img-static-site**

Click the **Save** button to store the changes

Initiate the execution with the **Build Now** action

Hm, it fails! Why? *Perhaps, due to where the git command downloads the files, and where we execute docker commands … The first is happening on the* ***Jenkins*** *host, and the second – on the* ***Docker*** *host*

So, we will refactor the job (in the next module, we will see an alternative approach)

Return in the configuration mode of the job

Remove the check against the **GitHub Project** option

And then select **None** in the **Source Code Management** section

Add a new **Execute shell script on remote host using ssh** and move it first

Make sure that the SSH site is the correct one (the same as in the other two steps)

Then, in the **Command** field enter

**cd /projects**

**git clone https://github.com/shekeriev/simple-docker-image.git**

In the next build section change the pat to match this

**cd /projects/simple-docker-image**

Save the changes and initiate a new build

Monitor the build process

On success, open a browser tab on the host and navigate to [**http://192.168.99.101**](http://192.168.99.101)to see the result

Go and execute it once more

What happened? An error! But why?

It appears that the issue comes from the folder with the files as it exists, and we cannot clone into it

Let’s change the first build step to this

**cd /projects**

**if [ -d /projects/simple-docker-image ]; then**

**cd /projects/simple-docker-image**

**git pull https://github.com/shekeriev/simple-docker-image.git**

**else**

**git clone https://github.com/shekeriev/simple-docker-image.git**

**fi**

Now, only the first time we clone and all the subsequent build we just pull

Save and close the project

Build it a few times and explore the results

### Docker Job #3 (BGApp)

Navigate to the **DO1-Demo** folder

Click on the **New Item** action

Select the **FreeStyle Project** job type

Enter **Docker-BGApp** for the name

Confirm with **OK**

Enter **Build and run containerized application from GitHub** as job description

Scroll down to **Build**

Select **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**cd /projects**

**if [ -d /projects/bgapp ]; then**

**cd /projects/bgapp**

**git pull https://github.com/shekeriev/bgapp.git**

**else**

**git clone https://github.com/shekeriev/bgapp.git**

**fi**

Select again **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**cd /projects/bgapp**

**docker image build -t img-web -f Dockerfile.web .**

Select again **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**docker container rm -f web || true**

**docker container run -d --name web -p 8080:80 -v $(pwd):/var/www/html:ro img-web**

Select again **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**cd /projects/bgapp**

**docker image build -t img-db -f Dockerfile.db .**

Select again **Execute shell script on remote host using ssh** to add a new build step

Select **jenkins@docker.do1.lab:22** as SSH site

Enter the following two lines in the **Command** text field

**docker container rm -f db || true**

**docker container run -d --name db -e MYSQL\_ROOT\_PASSWORD=12345 img-db**

Click the **Save** button to store the changes

Initiate the execution with the **Build Now** action

Monitor the build process

On success, open a browser tab on the host and navigate to [**http://192.168.99.101:8080**](http://192.168.99.101:8080)to see the result

Hm, **error**. But why? *Perhaps, it has to do with the web container … Let’s explore it*

Open a session to the second machine

Check the running containers

**docker container ls**

And inspect the **web** container

**docker container inspect web**

Check the mounts that are listed

Aha, here is the issue – the source path is wrong

So, instead of **$(pwd)** we should state the exact path - **/projects/bgapp/web**

Let’s return to the **Docker-BGApp** project and correct it

Change the third build step to this

**docker container rm -f web || true**

**docker container run -d --name web -p 8080:80 -v /projects/bgapp/web:/var/www/html:ro img-web**

Save the changes and build again

Check the result on [**http://192.168.99.101:8080**](http://192.168.99.101:8080)

Everything should be fine now

Hm, almost. We need to make sure that the web container can access the database one

Let’s return to the project and correct this as well

We should add a network and put the two containers in it

Add a new step and move it second

Now, following the approach with containers, we can set some logic to always try to remove and create the required network

This is not the best option. The main reason is that the deletion will fail in all subsequent builds as there are containers attached to it and it cannot be deleted

Instead, we can use the following to check and if the network is not present to create it

**docker network ls | grep appnet || docker network create appnet**

Next, alter the steps that are starting the containers by adding there the following flag

**--net appnet**

Save and build again the project

This time everything should work as expected even with multiple builds

### Create a Pipeline

Return to the **DO1-Demo** folder

Click on the **New Item** action

Select the **Pipeline** job type

Enter **Pipeline-1** for job name

Confirm with **OK**

Enter **Simple Pipeline project** for job description

Scroll down to the **Pipeline** section

Enter the following in the **Script** text field

pipeline

{

    agent any

    stages

    {

        stage('Start')

        {

            steps

            {

                echo 'Pipeline Test: Start'

            }

        }

        stage('Do')

        {

            steps

            {

                echo 'Pipeline Test: Do'

            }

        }

        stage('End')

        {

            steps

            {

                echo 'Pipeline Test: End'

            }

        }

    }

}

Confirm with the **Save** button

Initiate the build process with the **Build Now** action

Monitor the execution process

### Master and Slave Pipeline Jobs

Let us first create the slave pipeline job, i.e. the one that will be called from another job

Return to the **DO1-Demo** folder

Click on the **New Item** action

Select the **Pipeline** job type

Enter **Pipeline-Slave** as job name

Confirm with **OK** button

Enter **Pipeline Slave Project** as description

Select the option **This project is parameterized**

Click the **Add Parameter** button and select **String Parameter** option

In the **Name** field enter **RUN\_TIME**

Enter the following code in the **script** field of the **Pipeline** section

pipeline

{

    agent any

    stages

    {

        stage('Init')

        {

            steps

            {

                echo "Executed with RUN\_TIME=${RUN\_TIME}"

            }

        }

        stage('Do')

        {

            steps

            {

                echo 'Do some work'

                sh 'uptime'

            }

        }

    }

}

Confirm with the **Save** button

We can test this pipeline by initiating a build with **Build with Parameters**

Do it, and enter for example **2023-02-20 21:00** as value

Everything should work just fine

Now, we are ready to create the calling pipeline job

Navigate to the **DO1-Demo** folder

Click on the **New Item** action

Select the **Pipeline** job type

Enter **Pipeline-Master** for job name

Confirm with **OK**

Enter **Pipeline Master Project** for description

Scroll down to the **Pipeline** section

In the **script** field enter the following code

pipeline

{

    agent any

    environment

    {

        L\_RUN\_TIME = ''

    }

    stages

    {

        stage('Init')

        {

            steps

            {

                echo 'Master: Initialization Stage'

            }

        }

        stage('Do')

        {

            steps

            {

                echo 'Master: Build Slave'

                script

                {

                    def ldate = new Date()

                    L\_RUN\_TIME = ldate.format('yyyy-MM-dd HH:mm:ss')

                    println "Parameter Value is ${L\_RUN\_TIME}"

                }

                build job: 'Pipeline-Slave', parameters:[string(name:'RUN\_TIME', value: "${L\_RUN\_TIME}")]

            }

        }

        stage('Done')

        {

            steps

            {

                echo 'Master: Execution Complete'

            }

        }

    }

}

Confirm with the **Save** button

Initiate the build process with the **Build Now** action

Monitor the execution process and check the results

### Docker and git Pipeline

Let’s turn one of our earlier projects to a pipeline

As we did not explore the topic on **agents** and **slaves**, we should install **Docker** on the same host as **Jenkins**

Make sure that you have **Docker** and that the **jenkins** user can access it

Return to the **DO1-Demo** folder

Click on the **New Item** action

Select the **Pipeline** job type

Enter **Pipeline-Docker** for job name

Confirm with **OK**

Enter **Docker Pipeline project** for job description

Scroll down to the **Pipeline** section

Enter the following in the **Script** text field

pipeline

{

    agent any

    stages

    {

        stage('Clone the project')

        {

            steps

            {

                sh '''

                    cd /projects

                    if [ -d /projects/simple-docker-image ]; then

                      cd /projects/simple-docker-image

                      git pull https://github.com/shekeriev/simple-docker-image.git

                    else

                      git clone https://github.com/shekeriev/simple-docker-image.git

                    fi

                    '''

            }

        }

        stage('Build the image')

        {

            steps

            {

                sh 'cd /projects/www-static && docker image build -t img-static-site .'

            }

        }

        stage('Run the application')

        {

            steps

            {

                sh '''

                docker container rm -f co-static-site || true

                docker container run -d -p 9090:80 --name co-static-site img-static-site

                '''

            }

        }

    }

}

Confirm with the **Save** button

Initiate the build process with the **Build Now** action

If the project fails on the execution phases (image build and run) then perhaps you will need to restart **Jenkins**, as we added the user to the **docker** group earlier

Monitor the execution process

Once done, open a browser tab and navigate to [**http://192.168.99.100:9090**](http://192.168.99.100:9090)

It may work this time, but if we pay closer attention to the code, we will notice that we are cloning one repository (in the **/projects/simple-docker-image** folder) but then build an image base on the files in another folder (**/projects/www-static**). So, we have two options – either clone to correct repository or build the image in the appropriate folder. For the sake of simplicity, we will change the image build step. It should become:

        stage('Build the image')

        {

            steps

            {

                sh 'cd /projects/simple-docker-image && docker image build -t img-static-site .'

            }

        }

We skip the change of the image’s name.

Now, if we rebuild the job, we will have the right working application. 😊