

**i have instaleed those "" .. with this "# Vagrantfile**

Vagrant.configure("2") do |config|

**Use Ubuntu 22.04 LTS**

config.vm.box = "ubuntu/jammy64"  
config.vm.box\_version = "20231215.0.0"

**VM Configuration**

config.vm.hostname = "devops-vm"

**Network Configuration**

config.vm.network "private\_network", ip: "192.168.56.10"

**Port Forwarding for DevOps Tools**

config.vm.network "forwarded\_port", guest: 8080, host: 8080, id: "jenkins"  
config.vm.network "forwarded\_port", guest: 9000, host: 9000, id: "sonarqube"  
config.vm.network "forwarded\_port", guest: 3000, host: 3000, id: "grafana"  
config.vm.network "forwarded\_port", guest: 9090, host: 9090, id: "prometheus"  
config.vm.network "forwarded\_port", guest: 5341, host: 5341, id: "seq"  
config.vm.network "forwarded\_port", guest: 1433, host: 11433, id: "sqlserver"  
config.vm.network "forwarded\_port", guest: 5432, host: 15432, id: "postgres"  
config.vm.network "forwarded\_port", guest: 6379, host: 6379, id: "redis"  
config.vm.network "forwarded\_port", guest: 9443, host: 9443, id: "portainer"

**VirtualBox Provider Configuration**

config.vm.provider "virtualbox" do |vb|  
[vb.name](http://vb.name) = "DevOps-Tools-VM"  
vb.memory = "8192" # 8GB RAM (adjust based on your system)  
vb.cpus = 4 # 4 CPU cores (adjust based on your system)  
vb.gui = false

# Enable virtualization features  
vb.customize ["modifyvm", :id, "--nested-hw-virt", "on"]  
vb.customize ["modifyvm", :id, "--paravirtprovider", "kvm"]

end

**Shared Folders**

config.vm.synced\_folder ".", "/vagrant", type: "virtualbox"  
config.vm.synced\_folder "./shared", "/home/vagrant/shared", create: true

**Provisioning Scripts**

config.vm.provision "shell", path: "scripts/setup-vm.sh"  
config.vm.provision "shell", path: "scripts/install-docker.sh"  
config.vm.provision "shell", path: "scripts/install-devops-tools.sh"

**Run configuration as vagrant user (not root)**

config.vm.provision "shell", path: "scripts/configure-tools.sh", privileged: false  
end" .. and am inside the vagrant vm .. understand good the idea .. now i need to acced to my links of jenkins and all of that can u gude me

To access the web interfaces of Jenkins, SonarQube, Grafana, Prometheus, and other tools installed inside your Vagrant VM from your host machine, you need to use the forwarded ports defined in your Vagrantfile. Here’s how you can do it:

**How Port Forwarding Works**

The config.vm.network "forwarded\_port", guest: ..., host: ... lines in your Vagrantfile map ports from the VM (guest) to your computer (host). This allows you to access services running inside the VM via localhost:<host-port> on your host machine[[1]](#fn1)[[2]](#fn2).

For example, the line:

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

means that if Jenkins is running on port 8080 inside the VM, you can access it from your host using http://localhost:8080[[1]](#fn1)[[3]](#fn3).

**Accessing Your Tools**

Here are the URLs you should use on your host machine’s browser (not inside the VM):

|  |  |  |  |
| --- | --- | --- | --- |
| Tool | Host URL to Use | Guest Port | Host Port |
| Jenkins | <http://localhost:8080> | 8080 | 8080 |
| SonarQube | <http://localhost:9000> | 9000 | 9000 |
| Grafana | <http://localhost:3000> | 3000 | 3000 |
| Prometheus | <http://localhost:9090> | 9090 | 9090 |
| Seq | <http://localhost:5341> | 5341 | 5341 |
| SQL Server | localhost:11433 (TCP) | 1433 | 11433 |
| Postgres | localhost:15432 (TCP) | 5432 | 15432 |
| Redis | localhost:6379 (TCP) | 6379 | 6379 |
| Portainer | <https://localhost:9443> | 9443 | 9443 |

Just open your browser and enter the appropriate URL (for example, http://localhost:8080 for Jenkins)[[3]](#fn3)[[1]](#fn1).

**Troubleshooting**

* **VM Must Be Running:** Make sure your VM is up with vagrant up.
* **Correct Ports:** Use the host port numbers as defined in your Vagrantfile.
* **Browser Access:** Use your host machine’s browser, not the VM’s.
* **Firewall/Antivirus:** Ensure nothing is blocking the forwarded ports on your host OS.
* **Service Status:** If a page doesn’t load, SSH into the VM (vagrant ssh) and check if the service is running and listening on the correct port.

**Example**

If you want to access Grafana, after running vagrant up and waiting for provisioning, open your browser on your host OS and go to:

http://localhost:3000

You should see the Grafana login page[[3]](#fn3).

**Summary**

* Use localhost:<host-port> in your host browser for each tool.
* The mapping is defined by your Vagrantfile’s forwarded\_port settings.
* Make sure the VM is running and services are started inside the VM[[1]](#fn1)[[2]](#fn2).

If you follow these steps, you’ll be able to access all your DevOps tools from your host machine easily.

⁂

**still can not acceedd**

If you still cannot access the services (like Jenkins, SonarQube, Grafana, etc.) from your host machine using the forwarded ports, here’s a structured troubleshooting checklist to help you identify and resolve the issue:

**1. Check if the VM is Running**

Make sure your VM is up and running. On your host, run:

vagrant status

It should show the VM as "running".

**2. Verify Port Forwarding**

Your Vagrantfile correctly defines port forwarding. However, sometimes VirtualBox networking can have issues. Try restarting the VM:

vagrant reload

This command restarts the VM and re-applies the networking configuration.

**3. Check if Services are Running Inside the VM**

SSH into your VM:

vagrant ssh

Then, check if your services are running and listening on the expected ports. For example, for Jenkins:

sudo netstat -tulnp | grep 8080

Repeat for other ports (9000, 3000, etc.). You should see lines indicating the service is listening on the correct port.

**4. Check Service Status**

For example, for Jenkins:

sudo systemctl status jenkins

Replace jenkins with the appropriate service name for other tools.

**5. Check for Firewalls**

* On the VM, check if ufw (Uncomplicated Firewall) is enabled:

sudo ufw status

If it is active, allow the necessary ports or disable it for testing:

sudo ufw disable

* On your host, ensure no firewall or antivirus is blocking the forwarded ports.

**6. Check for Errors in Provisioning Scripts**

Your logs show some warnings about duplicate package sources, but nothing critical[[4]](#fn4). However, ensure that the actual install scripts for Jenkins, SonarQube, etc., completed successfully and did not fail silently.

Check logs for each service inside the VM (e.g., /var/log/jenkins/jenkins.log).

**7. Browser and URL**

* Use http://localhost:8080 for Jenkins, http://localhost:9000 for SonarQube, etc.
* Try accessing from different browsers or incognito mode to rule out browser cache issues.

**8. Check for Port Conflicts on Host**

If another process on your host is already using one of the forwarded ports (e.g., 8080), Vagrant won’t be able to forward it. On your host, run:

netstat -an | grep 8080

If you see output, something else may be using the port.

**9. VirtualBox Version**

Ensure you are using a recent version of VirtualBox and the VirtualBox Extension Pack, as old versions can have networking bugs.

**Quick Diagnostic Commands (inside VM)**

# Check if Jenkins is running and listening  
sudo systemctl status jenkins  
sudo netstat -tulnp | grep 8080  
  
# Check if SonarQube is running and listening  
sudo systemctl status sonarqube  
sudo netstat -tulnp | grep 9000  
  
# Check if Grafana is running and listening  
sudo systemctl status grafana-server  
sudo netstat -tulnp | grep 3000

**If Still Not Working**

* Provide the output of vagrant port (on your host) and netstat -tulnp (inside the VM).
* Check for any errors in the install logs for each tool.

**Summary Table**

|  |  |
| --- | --- |
| Step | Command/Action |
| VM running? | vagrant status |
| Restart VM/networking | vagrant reload |
| Check service running (in VM) | systemctl status <service> |
| Check port listening (in VM) | netstat -tulnp | grep <port> |
| Check firewall (in VM) | sudo ufw status |
| Port conflict (on host) | netstat -an | grep <port> |
| Access in browser | http://localhost:<host-port> |

If you follow these steps and provide any error messages or command outputs, it will be easier to pinpoint the issue[[4]](#fn4).

⁂

**vagrant@devops-vm:~$ vagrant status**

Command 'vagrant' not found, but can be installed with:  
apt install vagrant  
Please ask your administrator.  
vagrant@devops-vm:~$

You are seeing this message because you are running the command vagrant status inside the VM. The vagrant command is **not available inside the VM**—it is meant to be run on your host machine (your laptop or desktop, not inside the virtual machine).

**What to do:**

* **Exit the VM:** Type exit in your VM terminal to return to your host machine’s terminal.
* **Run Vagrant Commands on Host:** On your host machine (where you started the VM), run vagrant status, vagrant ssh, vagrant reload, etc.

**Summary Table**

|  |  |
| --- | --- |
| Command | Where to run it |
| vagrant status | Host machine terminal |
| vagrant up | Host machine terminal |
| vagrant ssh | Host machine terminal |
| sudo systemctl ... | Inside the VM |
| netstat ... | Inside the VM |

**Key Point:**  
Use vagrant commands only on your host machine, not inside the VM. Inside the VM, use Linux commands to check services, ports, and logs.

The vagrant CLI is designed to control virtual machines from the host, not from inside the guest VM.  
Standard Vagrant usage and documentation confirm that all vagrant commands must be run from the host OS, not from within the VM itself.

**vagrant@devops-vm:~$ exit**

logout

C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant>vagrant status  
Current machine states:

default running (virtualbox)

The VM is running. To stop this VM, you can run vagrant halt to  
shut it down forcefully, or you can run vagrant suspend to simply  
suspend the virtual machine. In either case, to restart it again,  
simply run vagrant up.

C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant>vagrant up  
Bringing machine 'default' up with 'virtualbox' provider...  
==> default: Checking if box 'ubuntu/jammy64' version '20231215.0.0' is up to date...  
==> default: Machine already provisioned. Run vagrant provision or use the --provision  
==> default: flag to force provisioning. Provisioners marked to run always will still run.

C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant>vagrant ssh  
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-140-generic x86\_64)

* Documentation: <https://help.ubuntu.com>
* Management: <https://landscape.canonical.com>
* Support: <https://ubuntu.com/pro>

System information as of Mon May 26 12:12:26 UTC 2025

System load: 0.1 Processes: 125  
Usage of /: 11.3% of 38.70GB Users logged in: 0  
Memory usage: 3% IPv4 address for enp0s3: 10.0.2.15  
Swap usage: 0%

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.  
3 of these updates are standard security updates.  
To see these additional updates run: apt list --upgradable

1 additional security update can be applied with ESM Apps.  
Learn more about enabling ESM Apps service at <https://ubuntu.com/esm>

New release '24.04.2 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon May 26 10:51:23 2025 from 10.0.2.2  
vagrant@devops-vm:~$ sudo systemctl ...  
Unknown command verb ....  
vagrant@devops-vm:~$

Great, your VM is running and your provisioning script executed successfully. However, you still **cannot access the web interfaces** (like Jenkins, SonarQube, etc.) from your host browser. Let’s proceed step by step to diagnose and solve the problem.

**Step 1: Check if Services Are Running Inside the VM**

SSH into your VM (which you already did), then check if the main services are running and listening on the expected ports:

# Check Jenkins (port 8080)  
sudo systemctl status jenkins  
sudo netstat -tulnp | grep 8080  
  
# Check SonarQube (port 9000)  
sudo systemctl status sonarqube  
sudo netstat -tulnp | grep 9000  
  
# Check Grafana (port 3000)  
sudo systemctl status grafana-server  
sudo netstat -tulnp | grep 3000  
  
# Check Prometheus (port 9090)  
sudo systemctl status prometheus  
sudo netstat -tulnp | grep 9090

**If you see "active (running)" and a line with LISTEN for each port, the service is running. If not, you may need to (re)start the service:**

sudo systemctl start jenkins  
sudo systemctl start sonarqube  
sudo systemctl start grafana-server  
sudo systemctl start prometheus

**Step 2: Check Port Forwarding**

On your **host machine**, run:

vagrant port

This will show you which guest ports are mapped to which host ports. The output should look like:

8080 (guest) => 8080 (host)  
9000 (guest) => 9000 (host)  
...

If you see the correct mappings, continue. If you don’t, there may be a Vagrant or VirtualBox networking issue.

**Step 3: Check for Port Conflicts on Host**

On your **host machine**, check if the ports (8080, 9000, etc.) are already in use by another process:

netstat -ano | findstr :8080  
netstat -ano | findstr :9000

If another process is using the port, you need to stop it or change the port mapping in your Vagrantfile.

**Step 4: Try Accessing the Services**

On your **host machine**, open a browser and try:

* <http://localhost:8080> (Jenkins)
* <http://localhost:9000> (SonarQube)
* <http://localhost:3000> (Grafana)
* <http://localhost:9090> (Prometheus)

If you see a login page or web UI, it’s working!

**Step 5: Check Firewall Settings**

* **On the VM:** Ubuntu typically doesn’t block outgoing connections on forwarded ports, but you can check with:

sudo ufw status

If active, allow the needed ports or disable for testing:

sudo ufw disable

* **On the Host:** Make sure Windows Firewall or antivirus isn’t blocking these ports.

**Step 6: If Still Not Working**

* Restart the VM and re-provision:

vagrant reload --provision

* Try accessing via the private network IP: <http://192.168.56.10:8080> (if your host is on the same network).

**Step 7: Share Results**

If you’re still stuck, please copy and paste the output of:

* sudo netstat -tulnp (inside the VM)
* vagrant port (on the host)
* Any error messages from your browser

**Quick Summary Table**

|  |  |  |
| --- | --- | --- |
| Step | Command (where) | What to look for |
| Service running? | sudo systemctl status <service> (VM) | "active (running)" |
| Port listening? | sudo netstat -tulnp | grep <port> (VM) | "LISTEN" on the correct port |
| Port forwarding? | vagrant port (host) | Correct guest→host port mapping |
| Port conflict? | netstat -ano | findstr :<port> (host) | No other process using the port |
| Browser access? | http://localhost:<port> (host) | Login page/UI loads |

**Follow these steps and share any outputs if you need more help!**  
This will help pinpoint exactly where the problem is.

⁂

**"vagrant@devops-vm:~$ sudo systemctl ...**

Unknown command verb ....  
vagrant@devops-vm:~$ sudo systemctl status jenkins  
Unit jenkins.service could not be found.  
vagrant@devops-vm:~$ sudo netstat -tulnp | grep 8080  
sudo: netstat: command not found  
vagrant@devops-vm:~$ "

You are making good progress! Here’s a summary of what’s happening and **what you should do next**:

**What’s Happening?**

* **Jenkins is not installed**:  
  The error Unit jenkins.service could not be found. means Jenkins is not installed or not set up as a service.
* **netstat is not installed**:  
  The error sudo: netstat: command not found means the net-tools package (which provides netstat) is missing.
* **Your provisioning script ran, but did not install Jenkins or other DevOps tools**:  
  Your logs show only base packages and Node.js/.NET, but not Jenkins, SonarQube, Grafana, etc.

**Next Steps**

**1. Install net-tools (for netstat)**

Inside the VM, run:

sudo apt update  
sudo apt install net-tools -y

Now you can use netstat to check open ports.

**2. Check Your Provisioning Scripts**

Your Vagrantfile references these scripts:

* scripts/setup-vm.sh
* scripts/install-docker.sh
* scripts/install-devops-tools.sh
* scripts/configure-tools.sh

It looks like only setup-vm.sh was run.  
**Check if the other scripts exist and are executable.**  
You can list them inside the VM with:

ls -l /vagrant/scripts/

**3. Manually Run the DevOps Tools Install Script**

If install-devops-tools.sh exists, run it:

sudo /vagrant/scripts/install-devops-tools.sh

Watch for errors. This script should install Jenkins, SonarQube, Grafana, etc.

**4. Check Services After Installation**

After running the install script, check if services are installed and running:

sudo systemctl status jenkins  
sudo systemctl status sonarqube  
sudo systemctl status grafana-server  
sudo systemctl status prometheus

If they are not running, you may need to start them:

sudo systemctl start jenkins  
sudo systemctl start sonarqube  
sudo systemctl start grafana-server  
sudo systemctl start prometheus

**5. Check Port Listening**

Now, use netstat to check if ports are open:

sudo netstat -tulnp | grep 8080 # Jenkins  
sudo netstat -tulnp | grep 9000 # SonarQube  
sudo netstat -tulnp | grep 3000 # Grafana  
sudo netstat -tulnp | grep 9090 # Prometheus

**6. Try Accessing the Services Again**

On your host machine, open your browser and try:

* <http://localhost:8080> (Jenkins)
* <http://localhost:9000> (SonarQube)
* <http://localhost:3000> (Grafana)
* <http://localhost:9090> (Prometheus)

**If You Get Errors**

* **If the install script fails:** Copy and paste the error messages here.
* **If services still don’t exist:** Your install script may be missing or not correct. Check the contents of /vagrant/scripts/install-devops-tools.sh.

**Summary Table**

|  |  |
| --- | --- |
| Task | Command (inside VM) |
| Install netstat | sudo apt install net-tools -y |
| List scripts | ls -l /vagrant/scripts/ |
| Run DevOps tools install script | sudo /vagrant/scripts/install-devops-tools.sh |
| Check service status | sudo systemctl status <service> |
| Check if port is open | sudo netstat -tulnp | grep <port> |

**Follow these steps and let me know what you find!**  
If you need help with the install script, paste its contents here.

⁂

**"vagrant@devops-vm:~$ sudo /vagrant/scripts/configure-tools.sh**

=== Configuring DevOps Tools ===  
open /vagrant/docker/devops-compose.yml: no such file or directory  
Waiting for services to start...

=== Container Status ===  
✗ jenkins is not running  
✗ sonarqube is not running  
✗ sqlserver is not running  
✗ redis is not running  
✗ seq is not running  
✗ prometheus is not running  
✗ grafana is not running  
✗ portainer is not running  
✗ postgres is not running

=== Service Information ===  
=== Jenkins Setup Information ===  
Jenkins URL: <http://localhost:8080> (or <http://192.168.56.10:8080>)  
Initial Admin Password:  
Jenkins is still starting up. Wait a few more minutes and check manually with:  
docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword

=== SonarQube Setup Information ===  
SonarQube URL: <http://localhost:9000> (or <http://192.168.56.10:9000>)  
Default credentials: admin/admin  
Note: SonarQube may take 2-3 minutes to fully start

=== SQL Server Setup Information ===  
SQL Server: localhost,1433 (from host: localhost,11433)  
Username: sa  
Password: YourStrong@Password123  
Connection String: Server=localhost,11433;Database=master;User Id=sa;Password=YourStrong@Password123;TrustServerCertificate=true;

=== PostgreSQL Setup Information ===  
PostgreSQL: localhost,5432 (from host: localhost,15432)  
Username: sonarqube  
Password: sonarqube  
Database: sonarqube

=== Redis Setup Information ===  
Redis URL: localhost:6379  
No authentication required

=== Seq Logging Setup Information ===  
Seq URL: <http://localhost:5341> (or <http://192.168.56.10:5341>)  
No authentication required initially

=== Prometheus Setup Information ===  
Prometheus URL: <http://localhost:9090> (or <http://192.168.56.10:9090>)  
No authentication required

=== Grafana Setup Information ===  
Grafana URL: <http://localhost:3000> (or <http://192.168.56.10:3000>)  
Default credentials: admin/admin123

=== Portainer Setup Information ===  
Portainer URL: <https://localhost:9443> (or <https://192.168.56.10:9443>)  
Create admin user on first login

=== Configuration completed ===  
All services are starting up. Please wait a few minutes before accessing the web interfaces.

Useful commands:  
Check all containers: docker ps  
View logs: docker logs <container-name>  
Restart services: docker compose -f /vagrant/docker/devops-compose.yml restart  
Stop all services: docker compose -f /vagrant/docker/devops-compose.yml down  
vagrant@devops-vm:~$ sudo systemctl start jenkins  
Failed to start jenkins.service: Unit jenkins.service not found.  
vagrant@devops-vm:~$" find a sooution cuz i can taceed

You are very close! Here’s exactly what’s happening and how to fix it:

**What’s the Real Issue?**

1. **Your DevOps tools (Jenkins, SonarQube, etc.) are NOT running as system services.**
2. **Your setup uses Docker Compose to run these tools as containers.**
3. **The error:**

open /vagrant/docker/devops-compose.yml: no such file or directory

means the main Docker Compose file that should launch all your containers is missing!  
Because of this, **none of your DevOps containers are running**, and you can’t access any web interfaces.

**How to Fix This**

**Step 1: Check for the Compose File**

On your VM, run:

ls -l /vagrant/docker/

You should see a file called devops-compose.yml.  
**If it’s missing**, that’s the root of your problem.

**Step 2: Restore or Create the Compose File**

**Option A: Restore from Your Project**

* On your host machine, check if you have a file at:  
  C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant\docker\devops-compose.yml
* If not, check your project’s README or source code repository for this file.
* If you have a backup or can copy it from somewhere, place it in the correct directory on your host:

C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant\docker\devops-compose.yml

* **Vagrant will sync it automatically into the VM at** /vagrant/docker/devops-compose.yml.

**Option B: Create a Minimal Compose File**

If you don’t have the file, you can create a minimal one as a test.  
On your host, create devops-compose.yml with this content (for Jenkins only, as an example):

version: '3.7'  
services:  
 jenkins:  
 image: jenkins/jenkins:lts  
 ports:  
 - "8080:8080"  
 volumes:  
 - jenkins\_home:/var/jenkins\_home  
volumes:  
 jenkins\_home:

**Step 3: Start the Containers**

Back in your VM, run:

cd /vagrant/docker  
docker compose -f devops-compose.yml up -d

This will start Jenkins (and any other services listed in the compose file).

**Step 4: Check Container Status**

docker ps

You should see your containers running.

**Step 5: Access the Web Interfaces**

On your host, open:

* <http://localhost:8080> for Jenkins
* <http://localhost:9000> for SonarQube (if included in your compose file), etc.

**Summary Table**

|  |  |
| --- | --- |
| What to Check/Do | Command/Action |
| Check for compose file | ls -l /vagrant/docker/ |
| Restore or create devops-compose.yml | Place/copy file in docker/ folder on your host |
| Start containers | docker compose -f /vagrant/docker/devops-compose.yml up -d |
| Check running containers | docker ps |
| Access web UIs | http://localhost:8080 etc. on your host browser |

**If You Need a Full Example Compose File**

If you want a full example for Jenkins, SonarQube, Grafana, Prometheus, etc., let me know and I’ll generate one for you!

**TL;DR:**  
Your Docker Compose file is missing. Restore or create /vagrant/docker/devops-compose.yml, then run docker compose up -d inside /vagrant/docker/ in your VM.  
Your tools will then be accessible on the forwarded ports!

Let me know if you need an example devops-compose.yml or further help!

⁂

**am creating this "" under "C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant\docker" with what you know as a scripts .. and after that "vagrant@devops-vm:~$ ls -l /vagrant/docker/**

total 0  
vagrant@devops-vm:~$ sudo mkdir -p /vagrant/docker/{prometheus,grafana/{provisioning/{datasources,dashboards}},nginx}  
vagrant@devops-vm:~$ sudo nano /vagrant/docker/devops-compose.yml  
vagrant@devops-vm:~$ sudo chown -R vagrant:vagrant /vagrant/docker  
vagrant@devops-vm:~$ cd /vagrant/docker  
vagrant@devops-vm:/vagrant/docker$ sudo docker compose -f devops-compose.yml up -d  
yaml: unmarshal errors:  
line 4: cannot unmarshal !!str echo "=... into cli.named  
vagrant@devops-vm:/vagrant/docker$" find a solution and make all of this is compatible

Excellent! You are on the right track, and your attached script (from [paste.txt](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/59008667/8016f43d-738b-4e87-9d7d-a1c40e8aad7b/paste.txt)) provides a **complete, working setup** for your DevOps Docker Compose environment.  
**You should use this script to generate all the necessary configuration files and directories in one go.**

**How to Fix Your Setup and Make Everything Compatible**

**1. Use the Provided Script**

**Copy the contents of the script** (from your paste.txt) to a new file on your VM, for example /vagrant/docker/fix-devops-setup.sh:

nano /vagrant/docker/fix-devops-setup.sh  
# Paste the script contents here, save, and exit (Ctrl+O, Enter, Ctrl+X)

**Make the script executable:**

chmod +x /vagrant/docker/fix-devops-setup.sh

**Run the script:**

sudo bash /vagrant/docker/fix-devops-setup.sh

**2. What This Script Does**

* Creates all required directories and files for your Docker Compose setup.
* Generates a comprehensive devops-compose.yml with Jenkins, SonarQube, Grafana, Prometheus, SQL Server, Redis, Seq, Portainer, and PostgreSQL.
* Sets up Prometheus and Grafana configuration files.
* Sets permissions correctly.
* Starts all services via Docker Compose.
* Waits for containers to start and shows you the access URLs.

**3. After Running the Script**

* **Check container status:**

docker ps -a

You should see all your DevOps containers running.

* **Access your services from your host:**
  + Jenkins: <http://localhost:8080>
  + SonarQube: <http://localhost:9000>
  + Grafana: <http://localhost:3000>
  + Prometheus: <http://localhost:9090>
  + Portainer: <https://localhost:9443>
  + Seq: <http://localhost:5341>
  + SQL Server: localhost,11433 (sa/YourStrong@Password123)
  + Redis: localhost:6379

**4. Troubleshooting**

* If any container does not start, check its logs:

docker logs <container-name>

* If you see port conflicts, make sure nothing else on your host is using the same ports.
* If you need to re-run the setup, you can run the script again.

**Summary Table**

|  |  |
| --- | --- |
| Step | Command |
| Create script | nano /vagrant/docker/fix-devops-setup.sh |
| Make executable | chmod +x /vagrant/docker/fix-devops-setup.sh |
| Run script | sudo bash /vagrant/docker/fix-devops-setup.sh |
| Check containers | docker ps -a |
| Access services | Use URLs shown at the end of the script output |
| Get Jenkins password | sudo docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword |

**Why You Got the YAML Error**

You got:

yaml: unmarshal errors:  
 line 4: cannot unmarshal !!str `echo "=...` into cli.named

Because your devops-compose.yml file was not valid YAML (it probably contained shell commands or was not copied correctly).  
**The script above will generate a correct YAML file.**

**Next Steps**

1. **Run the script as described above.**
2. **Wait for containers to start.**
3. **Access your DevOps tools from your host browser!**

If you need the full script pasted here, just ask!  
Let me know if you encounter any issues or need further help.

⁂

**what about this files "" under "C:\Users\PC\Desktop\devops-Projet-Gestion-Missions\vagrant\scripts" !**

Great question! The files you see in your screenshot (configure-tools.sh, install-devops-tools.sh, install-docker.sh, setup-vm.sh) are **provisioning scripts** used by your Vagrant setup. Here’s what they typically do and how they relate to your Docker Compose DevOps environment:

**What Are These Scripts?**

|  |  |
| --- | --- |
| Script Name | Typical Purpose |
| setup-vm.sh | Installs base packages, updates the system, sets up basic tools and user permissions. |
| install-docker.sh | Installs Docker and Docker Compose so you can run containers. |
| install-devops-tools.sh | (Usually) installs DevOps tools or sets up configuration for them. In your case, likely prepares for Docker-based setup. |
| configure-tools.sh | Configures or launches the tools (often runs Docker Compose, sets up configs, etc.). |

**How Do They Work With Your Docker Compose Setup?**

* **These scripts are called by your Vagrantfile** to automate the VM setup.
* **They are not the same as your Docker Compose file** (devops-compose.yml).  
  The Compose file is what actually defines and launches Jenkins, SonarQube, Grafana, etc., as containers.
* **If your Compose file was missing**, the configure-tools.sh script could not start your tools, which is why you saw errors.

**What Should You Do Now?**

1. **Make sure your /vagrant/docker/devops-compose.yml exists and is correct.**  
   Use the [script you posted](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/59008667/8016f43d-738b-4e87-9d7d-a1c40e8aad7b/paste.txt) to generate all necessary files.
2. **You can run the script manually** in your VM to fix everything:
   * Save the script as /vagrant/docker/fix-devops-setup.sh
   * Run:

chmod +x /vagrant/docker/fix-devops-setup.sh  
sudo bash /vagrant/docker/fix-devops-setup.sh

* + This will create the Compose file and all required configs, then start your containers.

1. **After the script runs, your tools should be accessible on the forwarded ports from your host.**

**Should You Edit These Scripts?**

* **If you want to automate the fix for missing Compose files**, you can add the contents of your fix script to install-devops-tools.sh or configure-tools.sh.
* **Otherwise, you can run the fix script manually as described above.**

**Summary**

* The scripts in your screenshot are for VM provisioning.
* Your actual DevOps stack is defined in the Docker Compose file (devops-compose.yml).
* Use your fix script to create the Compose file and configs.
* After running the fix, your DevOps tools will run as containers and be accessible from your host.

**If you want, I can show you how to merge your fix script into one of your existing provisioning scripts for full automation. Just ask!**

⁂

**for using DBeaver for sql server .. am instaling and what i need to choose**

Great question! Here’s what you need to know for installing **DBeaver Community** to use with **SQL Server**:

**Which Components Should You Select?**

From your screenshot, you have these options:

* **DBeaver Community** (must be checked)
* **Include Java** (recommended, unless you already have Java installed)
* **Reset Settings** (optional, resets to default settings)
* **Associate SQL files** (recommended, lets you open .sql files with DBeaver by double-clicking)
* **Associate SQLite database files** (optional, only needed if you work with SQLite databases)

**For SQL Server Usage:**

* **DBeaver Community**: **Required** (this is the main program)
* **Include Java**: **Recommended**  
  DBeaver needs Java to run. If you’re not sure you have Java installed, leave this checked.
* **Associate SQL files**: Optional but **recommended** for convenience.
* **Associate SQLite database files**: Optional, only if you use SQLite.
* **Reset Settings**: Optional, only if you want to reset to default settings.

**Recommended Choices**

For most users (especially if you are new to DBeaver), just leave the **default options checked**:

* DBeaver Community ✔️
* Include Java ✔️
* Reset Settings ✔️ (optional)
* Associate SQL files ✔️ (recommended)
* Associate SQLite database files ✔️ (optional)

Then click **Next** to continue.

**After Installation**

1. **Open DBeaver.**
2. **Add a new connection:**
   * Choose **SQL Server** from the list.
   * Enter your VM’s SQL Server connection details:
     + **Host:** localhost
     + **Port:** 11433 (as per your port forwarding)
     + **User:** sa
     + **Password:** YourStrong@Password123
3. **Test Connection** and you’re ready to go!

**Summary Table**

|  |  |  |
| --- | --- | --- |
| Option | Recommended for SQL Server? | Why? |
| DBeaver Community | ✔️ | Main program |
| Include Java | ✔️ | Needed if you don’t have Java |
| Reset Settings | Optional | Only if you want default settings |
| Associate SQL files | ✔️ | Convenience for opening .sql files |
| Associate SQLite db files | Optional | Only for SQLite work |

**In short:**  
Leave the default options checked and proceed. You’ll be ready to connect to SQL Server in your Vagrant VM!  
If you need help with the connection setup in DBeaver, let me know!

⁂

**ces etape "4.3 Configure Global Tools in Jenkins**

Go to Manage Jenkins > Global Tool Configuration  
Configure:

Git: Auto-install latest  
Docker: Auto-install latest  
NodeJS: Install Node 18.x, Global packages: @angular/cli@19  
SonarQube Scanner: Auto-install latest version  
.NET SDK: Configure path to /usr/share/dotnet/dotnet (in VM)

4.4 Configure SonarQube Integration

Go to Manage Jenkins > Configure System  
Find SonarQube servers section  
Add SonarQube server:

Name: SonarQube  
Server URL: <http://sonarqube:9000>  
Server authentication token: (Generate in SonarQube)

4.5 Configure Docker Integration

Go to Manage Jenkins > Configure System  
Find Docker section  
Add Docker Host:

Docker Host URI: unix:///var/run/docker.sock  
Enable Docker Host" en francais svp

Voici la traduction en français des étapes que vous avez mentionnées pour configurer Jenkins avec SonarQube et Docker :

**4.3 Configurer les outils globaux dans Jenkins**

* Allez dans **Gérer Jenkins > Configuration des outils globaux**
* Configurez les éléments suivants :
  + **Git** : Installation automatique de la dernière version
  + **Docker** : Installation automatique de la dernière version
  + **NodeJS** : Installer Node 18.x, avec les paquets globaux : @angular/cli@19
  + **SonarQube Scanner** : Installation automatique de la dernière version
  + **.NET SDK** : Configurez le chemin vers /usr/share/dotnet/dotnet (dans la machine virtuelle)

**4.4 Configurer l’intégration SonarQube**

* Allez dans **Gérer Jenkins > Configurer le système**
* Trouvez la section **Serveurs SonarQube**
* Ajoutez un serveur SonarQube avec les paramètres suivants :
  + **Nom** : SonarQube
  + **URL du serveur** : http://sonarqube:9000
  + **Jeton d’authentification serveur** : (générez-le dans SonarQube)

**4.5 Configurer l’intégration Docker**

* Allez dans **Gérer Jenkins > Configurer le système**
* Trouvez la section **Docker**
* Ajoutez un hôte Docker avec ces paramètres :
  + **URI de l’hôte Docker** : unix:///var/run/docker.sock
  + Cochez **Activer l’hôte Docker**

Cette configuration permet à Jenkins d’utiliser Git, Docker, NodeJS, SonarQube Scanner et .NET SDK pour vos builds, d’intégrer SonarQube pour l’analyse de code, et de communiquer avec Docker via le socket Unix pour gérer les conteneurs.

Si vous souhaitez, je peux aussi vous fournir des captures d’écran ou un guide détaillé en français.

⁂

**where i can find " Configuration des outils globaux"**

Sur la capture d’écran que vous avez partagée, la section que vous cherchez est bien présente.  
Pour accéder à la **Configuration des outils globaux** dans Jenkins, il faut cliquer sur :

**Tools**

**Texte affiché :**  
*Configurer les outils, leur localisation et les installateurs automatiques.*

**Résumé des étapes :**

1. Depuis le tableau de bord Jenkins, allez dans **Gérer Jenkins** (*Manage Jenkins*).
2. Cliquez sur **Tools** (c’est la tuile avec un marteau, intitulée « Configurer les outils… »).
3. Vous arriverez sur la page **Configuration des outils globaux** où vous pourrez configurer Git, NodeJS, Docker, SonarQube Scanner, etc.

**Remarque :**  
La traduction française de Jenkins peut parfois laisser certains titres en anglais ou mélanger les deux (ex : « Tools » au lieu de « Outils »), mais c’est bien la section que vous cherchez.

N’hésitez pas à demander si vous voulez un guide détaillé pour chaque outil à configurer !

⁂

**guide me**

Bien sûr ! Voici un guide étape par étape pour configurer les outils globaux dans Jenkins, à partir de l’écran que tu as partagé.

**1. Git**

* **Déjà présent** sur ta capture d’écran.
* **Name** : Laisse "Default" ou donne un nom de ton choix.
* **Path to Git executable** : Laisse "git" si Git est installé sur le PATH de la VM (c’est le cas avec la plupart des images Docker Jenkins).
* **Installation automatique** : Si tu veux que Jenkins télécharge et installe Git automatiquement, clique sur "Ajouter Git" et coche "Install automatically" (ou "Installer automatiquement" si l’option est disponible).

**2. JDK (Java)**

* Clique sur **Ajouter JDK**.
* **Name** : Par exemple "OpenJDK 17".
* **JAVA\_HOME** : Laisse vide si tu veux que Jenkins gère l’installation.
* Coche **Installer automatiquement** puis choisis la version souhaitée (par exemple, "AdoptOpenJDK" ou "Temurin", version 17 ou 21 selon tes besoins).

**3. NodeJS**

* Descends dans la page jusqu’à trouver la section **NodeJS**.
* Clique sur **Ajouter NodeJS**.
* **Name** : Par exemple "Node 18".
* Coche **Installer automatiquement**.
* Sélectionne la version **18.x**.
* Dans **Global npm packages to install**, ajoute :

@angular/cli@19

**4. Docker**

* Cherche la section **Docker** (si tu ne la vois pas, il faut installer le plugin "Docker" dans Jenkins).
* Clique sur **Ajouter Docker**.
* **Name** : "Docker".
* Coche **Installer automatiquement** (si proposé).
* Sinon, indique le chemin vers l’exécutable Docker (/usr/bin/docker ou simplement docker).

**5. SonarQube Scanner**

* Cherche la section **SonarQube Scanner** (nécessite le plugin SonarQube Scanner).
* Clique sur **Ajouter SonarQube Scanner**.
* **Name** : "Default" ou "SonarQube".
* Coche **Installer automatiquement** pour que Jenkins télécharge la dernière version.

**6. .NET SDK**

* Cherche la section **.NET SDK** ou **.NET Core SDK** (nécessite le plugin .NET SDK).
* Clique sur **Ajouter .NET SDK**.
* **Name** : ".NET 8" ou autre selon la version.
* **Path to dotnet executable** :

/usr/share/dotnet/dotnet

**7. Enregistrer**

* Clique sur **Enregistrer** en bas à gauche de la page pour sauvegarder ta configuration.

**Résumé visuel (exemple pour NodeJS) :**

|  |  |
| --- | --- |
| Champ | Valeur à mettre |
| Name | Node 18 |
| Installer automatiquement | ✔️ |
| Version | 18.x |
| Global npm packages | @angular/cli@19 |

**Remarques importantes :**

* Si tu ne vois pas certaines sections (Docker, NodeJS, SonarQube, .NET), installe les plugins correspondants dans **Gérer Jenkins > Gérer les plugins**.
* Après chaque ajout, clique sur **Enregistrer** pour ne pas perdre ta configuration.

**Besoin d’un exemple pour une section spécifique ou d’une capture d’écran ? Dis-le-moi !**

⁂

**there is no jdk 17 what i need to do**

You are correct: in your Jenkins “Tools” configuration, there is currently **no JDK 17 option**—you only see the button to **Ajouter JDK** (Add JDK).

Here’s what you need to do to add JDK 17 (or another version):

**Étapes pour ajouter JDK 17 dans Jenkins**

1. **Clique sur le bouton** Ajouter JDK.
2. **Nom** : Mets un nom explicite, par exemple :

OpenJDK 17

1. **JAVA\_HOME** :
   * Si tu veux que Jenkins gère l’installation automatiquement, **laisse ce champ vide**.
2. **Installer automatiquement** :
   * Coche la case **Installer automatiquement** (ou "Install automatically" si l’interface est en anglais).
   * Sélectionne dans la liste déroulante la version **OpenJDK 17** (ou “Temurin 17”, “AdoptOpenJDK 17”, selon ce qui est proposé).
   * Si la version 17 n’apparaît pas, choisis la version la plus proche (par exemple 21) ou clique sur “Ajouter un installateur” pour voir d’autres options.
3. **Enregistre la configuration** en cliquant sur **Enregistrer** ou **Appliquer** en bas de la page.

**Si tu ne vois pas JDK 17 dans la liste**

* **Vérifie que tu as accès à Internet** depuis Jenkins (pour télécharger les JDK).
* **Mets à jour le plugin “Tool Environment” ou “JDK Tool Plugin”** dans Jenkins :  
  Va dans **Gérer Jenkins > Gérer les plugins > Mises à jour** et mets à jour les plugins liés aux outils/JDK.
* **Redémarre Jenkins** après la mise à jour des plugins.

**Alternative : Définir manuellement le chemin**

Si tu as déjà JDK 17 installé sur ta VM, tu peux :

* Décocher "Installer automatiquement"
* Dans le champ **JAVA\_HOME**, mets le chemin d’installation, par exemple :

/usr/lib/jvm/java-17-openjdk-amd64

(Vérifie ce chemin avec ls /usr/lib/jvm/ dans ta VM)

**Résumé visuel**

|  |  |
| --- | --- |
| Champ | Valeur à mettre |
| Name | OpenJDK 17 |
| JAVA\_HOME | (laisser vide pour auto-installation) |
| Installer automatiquement | ✔️ |
| Version | OpenJDK 17 / Temurin 17 |

**Besoin d’aide pour vérifier le chemin sur ta VM ou pour installer JDK 17 manuellement ? Dis-le-moi !**

⁂

**if i will confirm .. he will not use the jdk 17 right**

You are absolutely correct:  
**If you confirm this configuration as shown in your screenshot, Jenkins will NOT install or use JDK 17.**  
It will try to use **Oracle JDK 9.0.4**, which is not what you want.

**Why?**

* The dropdown only offers "Java SE Development Kit 9.0.4" (Oracle JDK 9).
* There is **no JDK 17 option** here.
* Jenkins will use the version you select, so it will not use JDK 17 unless you specify it.

**What Should You Do?**

**Option 1: Use OpenJDK Already Installed on Your VM**

If you have OpenJDK 17 installed on your VM (which is likely if you used a recent Ubuntu or Docker image), point Jenkins directly to it:

1. **Décoche** "Install automatically" (décoche la case).
2. In the field **JAVA\_HOME**, enter the path to JDK 17.  
   On Ubuntu, it is usually:

/usr/lib/jvm/java-17-openjdk-amd64

(You can check this path by running ls /usr/lib/jvm/ in your VM.)

1. **Name**: "OpenJDK 17"
2. **Enregistre** la configuration.

**Option 2: Use a Different Installer (Temurin/AdoptOpenJDK)**

If you want Jenkins to install JDK 17 automatically:

1. Clique sur **Ajouter un installateur** (Add installer).
2. Choisis **AdoptOpenJDK** ou **Temurin** (if available).
3. Dans la liste des versions, cherche **17** ou **17.x**.
4. Si tu ne vois pas ces options, mets à jour le plugin "Tool Environment" ou "JDK Tool Plugin" dans Jenkins (voir ci-dessous).

**Option 3: Update Jenkins Plugins**

If you don’t see AdoptOpenJDK/Temurin or JDK 17:

1. Va dans **Gérer Jenkins > Gérer les plugins > Mises à jour**.
2. Mets à jour tous les plugins liés aux outils/JDK.
3. Redémarre Jenkins.
4. Reviens dans la configuration des outils globaux et réessaie.

**Résumé**

* **Ne valide pas** cette configuration si tu veux JDK 17 : elle installera JDK 9.
* **Préférence :** Décoche "Install automatically" et renseigne le chemin de JDK 17 déjà installé sur ta VM.
* **Sinon :** Mets à jour les plugins et cherche l’option AdoptOpenJDK/Temurin 17.

**Besoin d’aide pour vérifier si JDK 17 est déjà installé sur ta VM ou pour l’installer manuellement ? Je peux te donner la commande exacte !**

⁂

**"vagrant@devops-vm:/$ ls /usr/lib/jvm/**

java-1.17.0-openjdk-amd64 java-17-openjdk-amd64 openjdk-17  
vagrant@devops-vm:/$ "guide me

Super, tu as bien plusieurs versions de Java 17 installées dans ta VM, notamment :

java-1.17.0-openjdk-amd64  
java-17-openjdk-amd64  
openjdk-17

**Étapes pour configurer Jenkins afin d’utiliser JDK 17 déjà installé**

**1. Choisir le chemin JAVA\_HOME**

Tu peux utiliser l’un de ces chemins comme JAVA\_HOME dans Jenkins. Par exemple, le plus classique est :

/usr/lib/jvm/java-17-openjdk-amd64

**2. Configurer Jenkins**

* Connecte-toi à Jenkins.
* Va dans **Gérer Jenkins > Configuration des outils globaux** (Manage Jenkins > Global Tool Configuration).
* Dans la section **JDK**, clique sur **Ajouter JDK**.
* Remplis comme suit :

|  |  |
| --- | --- |
| Champ | Valeur |
| Name | OpenJDK 17 (ou un autre nom clair) |
| Décoche "Installer automatiquement" | (ne pas cocher) |
| JAVA\_HOME | /usr/lib/jvm/java-17-openjdk-amd64 |

* Clique sur **Enregistrer** en bas de la page.

**3. Vérification**

* Crée un job Jenkins simple (exemple : un job freestyle).
* Dans la configuration du job, dans la section **JDK**, sélectionne le JDK que tu viens de configurer (ex. "OpenJDK 17").
* Lance un build avec une étape shell :

java -version

* Vérifie dans la console du build que la version affichée est bien Java 17.

**4. (Optionnel) Installer manuellement JDK 17 si besoin**

Si tu veux être sûr que la VM a bien JDK 17 installé, tu peux lancer dans ta VM :

sudo apt update  
sudo apt install openjdk-17-jdk -y

**Résumé rapide**

|  |  |
| --- | --- |
| Étape | Commande ou action |
| Lister les JDK installés | ls /usr/lib/jvm/ |
| Choisir JAVA\_HOME | /usr/lib/jvm/java-17-openjdk-amd64 |
| Configurer Jenkins JDK | Gérer Jenkins > Configuration des outils globaux > JDK > Ajouter JDK |
| Décoche "Installer automatiquement" | Ne pas cocher pour utiliser JDK déjà installé |
| Entrer JAVA\_HOME | /usr/lib/jvm/java-17-openjdk-amd64 |
| Vérifier dans un job Jenkins | java -version dans une étape shell |

Si tu veux, je peux te guider pas à pas avec des captures d’écran ou commandes spécifiques !

**i caan save it like this**

No, you **should not save it like this** because Jenkins is warning you:

/usr/lib/jvm/java-17-openjdk-amd64 is not a directory on the Jenkins controller (but perhaps it exists on some agents)

This means Jenkins, running on your current machine (the "controller"), **does not see** this directory.  
This can happen for a few reasons:

**Possible Causes**

1. **Jenkins is running in a Docker container**  
   If Jenkins runs in a Docker container, it does **not** have access to your VM's /usr/lib/jvm/ directory unless you explicitly mount it as a volume.
2. **Jenkins is running as a different user**  
   If Jenkins runs as a different user or in a different environment, it might not see the system JDKs.
3. **Jenkins is running on another host**  
   If your Jenkins "controller" is not the VM itself (for example, if Jenkins is running on your host or in another container), the path won't exist.

**How to Fix**

**Option 1: Use Jenkins' Built-in JDK Installer**

* Check **Install automatically**.
* Select a JDK version to install (if JDK 17 or Temurin 17 is available).
* Jenkins will download and use this JDK for builds.

**Option 2: Mount the JDK into the Jenkins Container (if using Docker)**

If you want Jenkins (running in Docker) to use your system JDK, you need to mount it as a volume in your docker-compose.yml for Jenkins:

jenkins:  
 ...  
 volumes:  
 - /usr/lib/jvm/java-17-openjdk-amd64:/usr/lib/jvm/java-17-openjdk-amd64:ro

Then restart Jenkins and try again.

**Option 3: Install JDK 17 inside the Jenkins container**

If you control the Dockerfile or entrypoint for Jenkins, you can add:

RUN apt-get update && apt-get install -y openjdk-17-jdk

**Option 4: Use the Path Available Inside the Jenkins Container**

* Open a terminal in your Jenkins container:

docker exec -it jenkins bash

* Run:

ls /usr/lib/jvm/

* Use the path that actually exists inside the container.

**Quick Workaround**

* If you just want Jenkins to handle it, **check "Install automatically"** and let Jenkins download the JDK for you.

**Summary Table**

|  |  |
| --- | --- |
| Situation | Solution |
| Jenkins on host/VM, JDK installed | Use system path as JAVA\_HOME |
| Jenkins in Docker, no JDK in container | Use "Install automatically" or mount JDK |
| Jenkins in Docker, want system JDK | Mount JDK as volume in Docker Compose |
| Unsure/Quick fix | Use "Install automatically" |

**Recommendation:**  
Unless you have a specific reason to use the system JDK, just check "Install automatically" and let Jenkins manage the JDK version for your builds.  
If you want to use the system JDK, make sure Jenkins can see that directory (either on the host or inside the container).

Let me know how Jenkins is running (host, VM, Docker container) if you need a more specific solution!

⁂

**yes jenkins run on docker .. what i need to do exatly**

Thank you for clarifying that **Jenkins runs inside a Docker container**.  
That is why Jenkins does **not see** /usr/lib/jvm/java-17-openjdk-amd64—this path exists on your VM, but **not inside the Jenkins container**.

Here’s exactly what you should do:

**Recommended Solution: Let Jenkins Install JDK Automatically**

1. **Check the box** “Install automatically” (just below the JAVA\_HOME field).
2. Select a JDK version (if JDK 17 or Temurin 17 is available).
3. Save your configuration.

**Why?**  
Jenkins will download and use the JDK inside its own container, and you won’t have to worry about system paths.

**Alternative: Add JDK 17 to the Jenkins Container**

If you want to use a system JDK or need JDK 17 for all jobs by default, you must install it inside the Jenkins container.

**Option A: Modify your docker-compose.yml**

Add this to your Jenkins service:

jenkins:  
 image: jenkins/jenkins:lts  
 user: root  
 ports:  
 - "8080:8080"  
 - "50000:50000"  
 volumes:  
 - jenkins\_home:/var/jenkins\_home  
 # Add this section to install OpenJDK 17 at container startup  
 entrypoint: >  
 bash -c "  
 apt-get update &&  
 apt-get install -y openjdk-17-jdk &&  
 /sbin/tini -- /usr/local/bin/jenkins.sh  
 "

Then, restart Jenkins:

docker compose -f /vagrant/docker/devops-compose.yml down  
docker compose -f /vagrant/docker/devops-compose.yml up -d

Now, inside the Jenkins container, the path /usr/lib/jvm/java-17-openjdk-amd64 will exist.

**Option B: Use a Custom Dockerfile**

Create a Dockerfile:

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

Then, in your docker-compose.yml, use:

jenkins:  
 build: .  
 ports:  
 - "8080:8080"  
 - "50000:50000"  
 volumes:  
 - jenkins\_home:/var/jenkins\_home

Build and restart:

docker compose -f /vagrant/docker/devops-compose.yml build  
docker compose -f /vagrant/docker/devops-compose.yml up -d

**Summary Table**

|  |  |
| --- | --- |
| Solution | What to do |
| Easiest (recommended) | Check “Install automatically” in Jenkins JDK config |
| Use system JDK inside container | Install JDK 17 in Jenkins container (see above) |
| Use custom Jenkins image | Build Dockerfile with OpenJDK 17 installed |

**TL;DR**

* The path /usr/lib/jvm/java-17-openjdk-amd64 is not inside your Jenkins container by default.
* **Best solution:** In Jenkins, check “Install automatically” for JDK, and let Jenkins manage it.
* **If you need the JDK inside the container:** Install it via Dockerfile or entrypoint as shown above.

If you want a ready-to-use Dockerfile or more details, just ask!

⁂

**i have this "" with this "vagrant@devops-vm:/$ docker ps**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
e0a0b10fedc0 sonarqube:community "/opt/sonarqube/dock…" 58 minutes ago Up 58 minutes 0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp sonarqube  
a7a07a98e63c jenkins/jenkins:lts "/usr/bin/tini -- /u…" 58 minutes ago Up 58 minutes 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp, 0.0.0.0:50000->50000/tcp, [::]:50000->50000/tcp jenkins  
87f61f8b4518 postgres:15 "docker-entrypoint.s…" 58 minutes ago Up 58 minutes 0.0.0.0:15432->5432/tcp, [::]:15432->5432/tcp postgres-sonar  
8cc9beb99920 prom/prometheus:latest "/bin/prometheus --c…" 58 minutes ago Up 58 minutes 0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp prometheus  
72c3c5b53a42 [mcr.microsoft.com/mssql/server:2022-latest](http://mcr.microsoft.com/mssql/server:2022-latest) "/opt/mssql/bin/laun…" 58 minutes ago Up 58 minutes 0.0.0.0:11433->1433/tcp, [::]:11433->1433/tcp sqlserver  
e1fc7af7fce6 redis:7-alpine "docker-entrypoint.s…" 58 minutes ago Up 58 minutes 0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp redis  
a4e4b0ccfb39 datalust/seq:latest "/bin/seqentry" 58 minutes ago Up 58 minutes 443/tcp, 5341/tcp, 45341/tcp, 0.0.0.0:5341->80/tcp, [::]:5341->80/tcp seq  
6cb663dca2f2 portainer/portainer-ce:latest "/portainer" 58 minutes ago Up 58 minutes 8000/tcp, 9000/tcp, 0.0.0.0:9443->9443/tcp, [::]:9443->9443/tcp portainer  
aa34accd65e5 grafana/grafana:latest "/run.sh" 58 minutes ago Up 58 minutes 0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp grafana  
vagrant@devops-vm:/$ docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword  
8ff5e36368434aa1955d647276ec1d83  
vagrant@devops-vm:/$ ls /usr/lib/jvm/  
java-1.17.0-openjdk-amd64 java-17-openjdk-amd64 openjdk-17  
vagrant@devops-vm:/$" guid eme

You are doing everything right, but the key point is this:

**The path /usr/lib/jvm/java-17-openjdk-amd64 exists on your VM, but NOT inside your Jenkins container.**  
That’s why Jenkins shows the warning:  
/usr/lib/jvm/java-17-openjdk-amd64 is not a directory on the Jenkins controller (but perhaps it exists on some agents)

**Why?**

* Jenkins is running inside a Docker container (jenkins/jenkins:lts).
* The container has its own filesystem and does **not** have access to your VM’s /usr/lib/jvm/.
* By default, the Jenkins Docker image comes with a version of Java already installed (usually Java 11 or 17, depending on the image version).

**What Should You Do?**

**1. Let Jenkins Install JDK Automatically (Recommended & Easiest)**

* **Check the box** “Install automatically” (see your screenshot).
* Select the JDK version you want (preferably JDK 17 or Temurin 17 if available).
* Save the configuration.

**Why?**  
Jenkins will download and use the JDK inside its own container, and you won’t have to worry about system paths.

**2. Or, Use the JDK Already in the Jenkins Container**

If you want to use the JDK that comes with the Jenkins Docker image (usually Java 11 or 17), you don’t need to configure a custom JDK at all—just use the default.

* You can check which Java is available inside the Jenkins container by running:

docker exec -it jenkins java -version

* If it’s Java 17, you’re already set!

**3. (Advanced) Add a Custom JDK to the Jenkins Container**

If you need a specific JDK version not present in the Jenkins image:

* **Option A:** Create a custom Dockerfile that installs OpenJDK 17.
* **Option B:** Mount your VM’s JDK directory into the Jenkins container (not usually necessary).

But for most CI/CD use cases, **option 1 is preferred**.

**Summary Table**

|  |  |
| --- | --- |
| Situation | What to do |
| Want it easy and reliable | Use “Install automatically” in Jenkins |
| Use default JDK in Jenkins container | Don’t configure a custom JDK; use the built-in one |
| Need a specific JDK in the container | Build a custom Jenkins Docker image |

**Step-by-Step: What You Should Do Now**

1. **Go to Jenkins > Gérer Jenkins > Configuration des outils globaux.**
2. In the JDK section:
   * **Name:** OpenJDK 17
   * **JAVA\_HOME:** (leave empty)
   * **Check** “Install automatically”
   * Select JDK 17 or Temurin 17 if available.
3. **Save** the configuration.
4. In your Jenkins jobs, select this JDK when needed.

**Bonus: Check the Java Version in Jenkins Container**

To see which Java version is already available in your Jenkins container, run:

docker exec -it jenkins java -version

If it shows Java 17, you don’t need to configure anything else!

**In summary:**  
Don’t use the system path /usr/lib/jvm/java-17-openjdk-amd64 in Jenkins if Jenkins is running in Docker.  
Use “Install automatically” or the built-in JDK in the Jenkins container.  
Let me know if you want a custom Dockerfile example or further help!

⁂

**the jenkins not support jdk 17 aurotmaticly .. udpate a solutoiin**

You are correct:  
**If Jenkins does not offer JDK 17 as an automatic install and you cannot use the system path because Jenkins is running in Docker**, you need to make JDK 17 available inside the Jenkins container.

Here’s an **updated, step-by-step solution**:

**Solution: Use a Custom Jenkins Docker Image with JDK 17**

**1. Create a Custom Dockerfile for Jenkins**

On your host (in your Vagrant shared folder, e.g. /vagrant/docker/jenkins), create a file named Dockerfile with this content:

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

**2. Update Your devops-compose.yml**

In your devops-compose.yml, update the Jenkins service to build from your custom Dockerfile:

jenkins:  
 build: ./jenkins  
 container\_name: jenkins  
 restart: unless-stopped  
 ports:  
 - "8080:8080"  
 - "50000:50000"  
 volumes:  
 - jenkins\_data:/var/jenkins\_home  
 - /var/run/docker.sock:/var/run/docker.sock  
 - /usr/bin/docker:/usr/bin/docker  
 environment:  
 - JENKINS\_OPTS="--httpPort=8080"  
 - JAVA\_OPTS="-Djenkins.install.runSetupWizard=false -Xmx1024m"  
 user: root  
 networks:  
 - devops-network

**Note:**  
Make sure the build: ./jenkins path matches the location of your Dockerfile relative to your compose file.

**3. Rebuild and Restart Jenkins**

From /vagrant/docker, run:

docker compose build jenkins  
docker compose up -d

**4. Configure Jenkins JDK Tool**

Now, inside Jenkins:

* Go to **Gérer Jenkins > Configuration des outils globaux**.
* In the JDK section:
  + **Name:** OpenJDK 17
  + **JAVA\_HOME:** /usr/lib/jvm/java-17-openjdk-amd64
  + **Uncheck** "Install automatically"
* Save.

**This time, Jenkins will find JDK 17, because it is installed inside the Jenkins container.**

**Summary Table**

|  |  |
| --- | --- |
| Step | Command/Action |
| Create Dockerfile | See above, in ./jenkins/Dockerfile |
| Update compose file | Use build: ./jenkins for Jenkins service |
| Build & restart | docker compose build jenkins && docker compose up -d |
| Configure Jenkins JDK | JAVA\_HOME: /usr/lib/jvm/java-17-openjdk-amd64 |

**Why this works**

* Jenkins will run in a container that has JDK 17 installed at the correct path.
* The warning about the directory not existing will disappear.
* You can use JDK 17 for your Jenkins jobs reliably.

**If you need the exact folder structure or more help with the Dockerfile, let me know!**  
This is the most robust solution when Jenkins is running in Docker and you need a specific JDK version.

⁂

**am creating this "FROM jenkins/jenkins:lts**

USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins" full update "version: '3.8'

services:

**Jenkins with .NET support**

jenkins:  
image: jenkins/jenkins:lts  
container\_name: jenkins  
restart: unless-stopped  
ports:  
- "8080:8080"  
- "50000:50000"  
volumes:  
- jenkins\_data:/var/jenkins\_home  
- /var/run/docker.sock:/var/run/docker.sock  
- /usr/bin/docker:/usr/bin/docker  
environment:  
- JENKINS\_OPTS="--httpPort=8080"  
- JAVA\_OPTS="-Djenkins.install.runSetupWizard=false -Xmx1024m"  
user: root  
networks:  
- devops-network

**SonarQube for .NET code analysis**

sonarqube:  
image: sonarqube:community  
container\_name: sonarqube  
restart: unless-stopped  
ports:  
- "9000:9000"  
environment:  
- SONAR\_JDBC\_URL=jdbc:postgresql://postgres:5432/sonarqube  
- SONAR\_JDBC\_USERNAME=sonarqube  
- SONAR\_JDBC\_PASSWORD=sonarqube123  
- SONAR\_ES\_BOOTSTRAP\_CHECKS\_DISABLE=true  
- SONAR\_WEB\_JAVAADDITIONALOPTS=-Xmx1024m  
volumes:  
- sonarqube\_data:/opt/sonarqube/data  
- sonarqube\_logs:/opt/sonarqube/logs  
- sonarqube\_extensions:/opt/sonarqube/extensions  
depends\_on:  
- postgres  
networks:  
- devops-network

**PostgreSQL for SonarQube**

postgres:  
image: postgres:15  
container\_name: postgres-sonar  
restart: unless-stopped  
environment:  
- POSTGRES\_USER=sonarqube  
- POSTGRES\_PASSWORD=sonarqube123  
- POSTGRES\_DB=sonarqube  
volumes:  
- postgres\_data:/var/lib/postgresql/data  
ports:  
- "15432:5432"  
networks:  
- devops-network

**SQL Server for .NET applications**

sqlserver:  
image: [mcr.microsoft.com/mssql/server:2022-latest](http://mcr.microsoft.com/mssql/server:2022-latest)  
container\_name: sqlserver  
restart: unless-stopped  
ports:  
- "11433:1433"  
environment:  
- ACCEPT\_EULA=Y  
- SA\_PASSWORD=YourStrong@Password123  
- MSSQL\_PID=Developer  
volumes:  
- sqlserver\_data:/var/opt/mssql  
networks:  
- devops-network

**Redis for caching**

redis:  
image: redis:7-alpine  
container\_name: redis  
restart: unless-stopped  
ports:  
- "6379:6379"  
volumes:  
- redis\_data:/data  
networks:  
- devops-network

**Seq for .NET logging**

seq:  
image: datalust/seq:latest  
container\_name: seq  
restart: unless-stopped  
ports:  
- "5341:80"  
environment:  
- ACCEPT\_EULA=Y  
volumes:  
- seq\_data:/data  
networks:  
- devops-network

**Prometheus for monitoring**

prometheus:  
image: prom/prometheus:latest  
container\_name: prometheus  
restart: unless-stopped  
ports:  
- "9090:9090"  
volumes:  
- ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml  
- prometheus\_data:/prometheus  
command:  
- '--config.file=/etc/prometheus/prometheus.yml'  
- '--storage.tsdb.path=/prometheus'  
- '--web.console.libraries=/etc/prometheus/console\_libraries'  
- '--web.console.templates=/etc/prometheus/consoles'  
- '--web.enable-lifecycle'  
networks:  
- devops-network

**Grafana for dashboards**

grafana:  
image: grafana/grafana:latest  
container\_name: grafana  
restart: unless-stopped  
ports:  
- "3000:3000"  
environment:  
- GF\_SECURITY\_ADMIN\_PASSWORD=admin123  
- GF\_INSTALL\_PLUGINS=grafana-piechart-panel,grafana-worldmap-panel  
volumes:  
- grafana\_data:/var/lib/grafana  
- ./grafana/dashboards:/var/lib/grafana/dashboards  
- ./grafana/provisioning:/etc/grafana/provisioning  
networks:  
- devops-network

**Portainer for Docker management**

portainer:  
image: portainer/portainer-ce:latest  
container\_name: portainer  
restart: unless-stopped  
ports:  
- "9443:9443"  
volumes:  
- /var/run/docker.sock:/var/run/docker.sock  
- portainer\_data:/data  
networks:  
- devops-network

volumes:  
jenkins\_data:  
sonarqube\_data:  
sonarqube\_logs:  
sonarqube\_extensions:  
postgres\_data:  
sqlserver\_data:  
redis\_data:  
seq\_data:  
prometheus\_data:  
grafana\_data:  
portainer\_data:

networks:  
devops-network:  
driver: bridge " " "for the necesaary file

Great! You’re almost there. Here’s how to **fully update your setup** so Jenkins runs with JDK 17 inside the container, and Jenkins can use /usr/lib/jvm/java-17-openjdk-amd64 as JAVA\_HOME **without warnings**.

**Step-by-Step Solution**

**1. Create a Custom Jenkins Image with JDK 17**

**a. Create a folder for your custom Jenkins image:**

mkdir -p /vagrant/docker/jenkins

**b. Inside /vagrant/docker/jenkins, create a Dockerfile:**

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

**2. Update Your devops-compose.yml**

**Change the Jenkins service to use your custom build:**

services:  
 jenkins:  
 build: ./jenkins # <--- Use build instead of image  
 container\_name: jenkins  
 restart: unless-stopped  
 ports:  
 - "8080:8080"  
 - "50000:50000"  
 volumes:  
 - jenkins\_data:/var/jenkins\_home  
 - /var/run/docker.sock:/var/run/docker.sock  
 - /usr/bin/docker:/usr/bin/docker  
 environment:  
 - JENKINS\_OPTS=--httpPort=8080  
 - JAVA\_OPTS=-Djenkins.install.runSetupWizard=false -Xmx1024m  
 user: root  
 networks:  
 - devops-network

**Note:**

* Make sure build: ./jenkins points to the folder where your Dockerfile is (relative to your devops-compose.yml).

**3. Rebuild and Restart Jenkins**

From /vagrant/docker, run:

docker compose build jenkins  
docker compose up -d

**4. Verify JDK 17 Inside the Jenkins Container**

docker exec -it jenkins java -version

You should see Java 17.

**5. Configure Jenkins JDK Tool**

* Go to **Gérer Jenkins > Configuration des outils globaux**.
* In the JDK section:
  + **Name:** OpenJDK 17
  + **JAVA\_HOME:** /usr/lib/jvm/java-17-openjdk-amd64
  + **Uncheck** "Install automatically"
* Save.

The warning will disappear, and Jenkins will use JDK 17 for your jobs.

**Summary Table**

|  |  |
| --- | --- |
| Step | Command/Action |
| Create Dockerfile | See above in /vagrant/docker/jenkins/Dockerfile |
| Update compose file | Use build: ./jenkins for Jenkins |
| Build & restart Jenkins | docker compose build jenkins && docker compose up -d |
| Check Java version in container | docker exec -it jenkins java -version |
| Configure Jenkins JDK tool | JAVA\_HOME: /usr/lib/jvm/java-17-openjdk-amd64 |

**You only need to do this for Jenkins.**  
All your other services (SonarQube, Grafana, etc.) can stay as they are.

If you want the exact folder structure or a ready-to-copy Dockerfile and compose snippet, just ask!

⁂

**sudo docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword**

vagrant@devops-vm:/$ docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
e0a0b10fedc0 sonarqube:community "/opt/sonarqube/dock…" 58 minutes ago Up 58 minutes 0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp sonarqube  
a7a07a98e63c jenkins/jenkins:lts "/usr/bin/tini -- /u…" 58 minutes ago Up 58 minutes 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp, 0.0.0.0:50000->50000/tcp, [::]:50000->50000/tcp jenkins  
87f61f8b4518 postgres:15 "docker-entrypoint.s…" 58 minutes ago Up 58 minutes 0.0.0.0:15432->5432/tcp, [::]:15432->5432/tcp postgres-sonar  
8cc9beb99920 prom/prometheus:latest "/bin/prometheus --c…" 58 minutes ago Up 58 minutes 0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp prometheus  
72c3c5b53a42 [mcr.microsoft.com/mssql/server:2022-latest](http://mcr.microsoft.com/mssql/server:2022-latest) "/opt/mssql/bin/laun…" 58 minutes ago Up 58 minutes 0.0.0.0:11433->1433/tcp, [::]:11433->1433/tcp sqlserver  
e1fc7af7fce6 redis:7-alpine "docker-entrypoint.s…" 58 minutes ago Up 58 minutes 0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp redis  
a4e4b0ccfb39 datalust/seq:latest "/bin/seqentry" 58 minutes ago Up 58 minutes 443/tcp, 5341/tcp, 45341/tcp, 0.0.0.0:5341->80/tcp, [::]:5341->80/tcp seq  
6cb663dca2f2 portainer/portainer-ce:latest "/portainer" 58 minutes ago Up 58 minutes 8000/tcp, 9000/tcp, 0.0.0.0:9443->9443/tcp, [::]:9443->9443/tcp portainer  
aa34accd65e5 grafana/grafana:latest "/run.sh" 58 minutes ago Up 58 minutes 0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp grafana  
vagrant@devops-vm:/$ docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword  
8ff5e36368434aa1955d647276ec1d83  
vagrant@devops-vm:/$ ls /usr/lib/jvm/  
java-1.17.0-openjdk-amd64 java-17-openjdk-amd64 openjdk-17  
vagrant@devops-vm:/$ cd /vagrant/docker  
vagrant@devops-vm:/vagrant/docker$ docker compose build jenkins  
no configuration file provided: not found  
vagrant@devops-vm:/vagrant/docker$

You are **very close**, but there are two key issues to address:

**1. Why You See the JDK Path Warning**

* The warning /usr/lib/jvm/java-17-openjdk-amd64 is not a directory on the Jenkins controller appears because your **Jenkins container does not have JDK 17 installed** at that path.
* By default, your devops-compose.yml uses the official Jenkins image:

image: jenkins/jenkins:lts

This image does **not** include JDK 17 at /usr/lib/jvm/java-17-openjdk-amd64.

**2. Why docker compose build jenkins Fails**

* You ran docker compose build jenkins but got:

no configuration file provided: not found

* This means there is **no Dockerfile** for Jenkins in the expected build context (/vagrant/docker/jenkins).

**How to Fix Everything**

**A. Create a Custom Jenkins Image with JDK 17**

**1. Create the Jenkins Build Directory and Dockerfile**

On your host or in your VM:

mkdir -p /vagrant/docker/jenkins  
nano /vagrant/docker/jenkins/Dockerfile

Paste this content:

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

Save and exit.

**2. Update Your devops-compose.yml**

In the jenkins service, change from:

image: jenkins/jenkins:lts

to:

build: ./jenkins

So your Jenkins service looks like:

jenkins:  
 build: ./jenkins  
 container\_name: jenkins  
 restart: unless-stopped  
 ports:  
 - "8080:8080"  
 - "50000:50000"  
 volumes:  
 - jenkins\_data:/var/jenkins\_home  
 - /var/run/docker.sock:/var/run/docker.sock  
 - /usr/bin/docker:/usr/bin/docker  
 environment:  
 - JENKINS\_OPTS=--httpPort=8080  
 - JAVA\_OPTS=-Djenkins.install.runSetupWizard=false -Xmx1024m  
 user: root  
 networks:  
 - devops-network

**3. Build and Restart Jenkins**

cd /vagrant/docker  
docker compose build jenkins  
docker compose up -d

**4. Verify JDK 17 in Jenkins Container**

docker exec -it jenkins java -version

You should see Java 17.

**5. Configure Jenkins JDK Tool**

Now in Jenkins (web UI):

* Go to **Gérer Jenkins > Configuration des outils globaux**.
* In the JDK section:
  + **Name:** OpenJDK 17
  + **JAVA\_HOME:** /usr/lib/jvm/java-17-openjdk-amd64
  + **Uncheck** "Install automatically"
* Save.

The warning will disappear, and Jenkins will use JDK 17 for your jobs.

**Summary Table**

|  |  |
| --- | --- |
| Step | Command/Action |
| Create Dockerfile | /vagrant/docker/jenkins/Dockerfile |
| Update compose file | build: ./jenkins for Jenkins |
| Build & restart Jenkins | docker compose build jenkins && docker compose up -d |
| Check Java version in container | docker exec -it jenkins java -version |
| Configure Jenkins JDK tool | JAVA\_HOME: /usr/lib/jvm/java-17-openjdk-amd64 |

**You must use a custom Jenkins image with JDK 17 installed, and reference it in your compose file.**  
If you follow these steps, Jenkins will see the JDK path and the warning will disappear.

If you need a ready-to-copy folder structure or more help, just ask!

⁂

**i dont why still not see the file vagrant@devops-vm:/$ ls /usr/lib/jvm/**

java-1.17.0-openjdk-amd64 java-17-openjdk-amd64 openjdk-17  
vagrant@devops-vm:/$ cd /vagrant/docker  
vagrant@devops-vm:/vagrant/docker$ docker compose build jenkins  
no configuration file provided: not found  
vagrant@devops-vm:/vagrant/docker$ mkdir -p /vagrant/docker/jenkins  
vagrant@devops-vm:/vagrant/docker$ cd ..  
vagrant@devops-vm:/vagrant$ cd ..  
vagrant@devops-vm:/$ mkdir -p /vagrant/docker/jenkins  
vagrant@devops-vm:/$ nano /vagrant/docker/jenkins/Dockerfile  
vagrant@devops-vm:/$ cd /vagrant/docker  
vagrant@devops-vm:/vagrant/docker$ docker compose build jenkins  
no configuration file provided: not found  
vagrant@devops-vm:/vagrant/docker$

You are **very close** to solving your problem! Here’s exactly what’s happening and how to fix it:

**Why You See the Jenkins JDK Warning**

* The warning /usr/lib/jvm/java-17-openjdk-amd64 is not a directory on the Jenkins controller appears because **your Jenkins container does not have JDK 17 installed at that path**.
* On your VM, you have JDK 17, but the Jenkins container is isolated and does **not** see your VM’s /usr/lib/jvm/ directory.

**Why docker compose build jenkins Fails**

* The error no configuration file provided: not found means Docker Compose cannot find a docker-compose.yml or devops-compose.yml file in your current directory.
* Your compose file is /vagrant/docker/devops-compose.yml, so you must specify it explicitly with -f devops-compose.yml.

**How to Fix Everything (Step by Step)**

**1. Create a Custom Jenkins Image with JDK 17**

**a. Create the Jenkins build directory and Dockerfile:**

mkdir -p /vagrant/docker/jenkins  
nano /vagrant/docker/jenkins/Dockerfile

**Paste this content:**

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

Save and exit.

**2. Update Your devops-compose.yml**

**Change your Jenkins service from:**

image: jenkins/jenkins:lts

**to:**

build: ./jenkins

So it looks like:

jenkins:  
 build: ./jenkins  
 container\_name: jenkins  
 restart: unless-stopped  
 ports:  
 - "8080:8080"  
 - "50000:50000"  
 volumes:  
 - jenkins\_data:/var/jenkins\_home  
 - /var/run/docker.sock:/var/run/docker.sock  
 - /usr/bin/docker:/usr/bin/docker  
 environment:  
 - JENKINS\_OPTS=--httpPort=8080  
 - JAVA\_OPTS=-Djenkins.install.runSetupWizard=false -Xmx1024m  
 user: root  
 networks:  
 - devops-network

**Make sure build: ./jenkins is correct relative to your compose file.**

**3. Build and Start Jenkins with the Custom Image**

Go to your docker folder:

cd /vagrant/docker  
docker compose -f devops-compose.yml build jenkins  
docker compose -f devops-compose.yml up -d

**4. Verify JDK 17 Is Installed in the Jenkins Container**

docker exec -it jenkins java -version

You should see Java 17.

**5. Configure Jenkins to Use the JDK**

* Go to **Gérer Jenkins > Configuration des outils globaux**.
* In the JDK section:
  + **Name:** OpenJDK 17
  + **JAVA\_HOME:** /usr/lib/jvm/java-17-openjdk-amd64
  + **Uncheck** "Install automatically"
* Save.

The warning will disappear, and Jenkins will use JDK 17 for your jobs.

**Summary Table**

|  |  |
| --- | --- |
| Step | Command/Action |
| Create Dockerfile | /vagrant/docker/jenkins/Dockerfile |
| Update compose file | build: ./jenkins for Jenkins |
| Build & restart Jenkins | docker compose -f devops-compose.yml build jenkins && docker compose -f devops-compose.yml up -d |
| Check Java version in container | docker exec -it jenkins java -version |
| Configure Jenkins JDK tool | JAVA\_HOME: /usr/lib/jvm/java-17-openjdk-amd64 |

**If you follow these steps, Jenkins will see the JDK path and the warning will disappear.**  
If you need a ready-to-copy folder structure or run into any errors, let me know!

⁂

**i have "FROM jenkins/jenkins:lts**

USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins" "version: '3.8'

services:

**Jenkins with .NET support**

jenkins:  
build: ./jenkins  
container\_name: jenkins  
restart: unless-stopped  
ports:  
- "8080:8080"  
- "50000:50000"  
volumes:  
- jenkins\_data:/var/jenkins\_home  
- /var/run/docker.sock:/var/run/docker.sock  
- /usr/bin/docker:/usr/bin/docker  
environment:  
- JENKINS\_OPTS="--httpPort=8080"  
- JAVA\_OPTS="-Djenkins.install.runSetupWizard=false -Xmx1024m"  
user: root  
networks:  
- devops-network

**SonarQube for .NET code analysis**

sonarqube:  
image: sonarqube:community  
container\_name: sonarqube  
restart: unless-stopped  
ports:  
- "9000:9000"  
environment:  
- SONAR\_JDBC\_URL=jdbc:postgresql://postgres:5432/sonarqube  
- SONAR\_JDBC\_USERNAME=sonarqube  
- SONAR\_JDBC\_PASSWORD=sonarqube123  
- SONAR\_ES\_BOOTSTRAP\_CHECKS\_DISABLE=true  
- SONAR\_WEB\_JAVAADDITIONALOPTS=-Xmx1024m  
volumes:  
- sonarqube\_data:/opt/sonarqube/data  
- sonarqube\_logs:/opt/sonarqube/logs  
- sonarqube\_extensions:/opt/sonarqube/extensions  
depends\_on:  
- postgres  
networks:  
- devops-network

**PostgreSQL for SonarQube**

postgres:  
image: postgres:15  
container\_name: postgres-sonar  
restart: unless-stopped  
environment:  
- POSTGRES\_USER=sonarqube  
- POSTGRES\_PASSWORD=sonarqube123  
- POSTGRES\_DB=sonarqube  
volumes:  
- postgres\_data:/var/lib/postgresql/data  
ports:  
- "15432:5432"  
networks:  
- devops-network

**SQL Server for .NET applications**

sqlserver:  
image: [mcr.microsoft.com/mssql/server:2022-latest](http://mcr.microsoft.com/mssql/server:2022-latest)  
container\_name: sqlserver  
restart: unless-stopped  
ports:  
- "11433:1433"  
environment:  
- ACCEPT\_EULA=Y  
- SA\_PASSWORD=YourStrong@Password123  
- MSSQL\_PID=Developer  
volumes:  
- sqlserver\_data:/var/opt/mssql  
networks:  
- devops-network

**Redis for caching**

redis:  
image: redis:7-alpine  
container\_name: redis  
restart: unless-stopped  
ports:  
- "6379:6379"  
volumes:  
- redis\_data:/data  
networks:  
- devops-network

**Seq for .NET logging**

seq:  
image: datalust/seq:latest  
container\_name: seq  
restart: unless-stopped  
ports:  
- "5341:80"  
environment:  
- ACCEPT\_EULA=Y  
volumes:  
- seq\_data:/data  
networks:  
- devops-network

**Prometheus for monitoring**

prometheus:  
image: prom/prometheus:latest  
container\_name: prometheus  
restart: unless-stopped  
ports:  
- "9090:9090"  
volumes:  
- ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml  
- prometheus\_data:/prometheus  
command:  
- '--config.file=/etc/prometheus/prometheus.yml'  
- '--storage.tsdb.path=/prometheus'  
- '--web.console.libraries=/etc/prometheus/console\_libraries'  
- '--web.console.templates=/etc/prometheus/consoles'  
- '--web.enable-lifecycle'  
networks:  
- devops-network

**Grafana for dashboards**

grafana:  
image: grafana/grafana:latest  
container\_name: grafana  
restart: unless-stopped  
ports:  
- "3000:3000"  
environment:  
- GF\_SECURITY\_ADMIN\_PASSWORD=admin123  
- GF\_INSTALL\_PLUGINS=grafana-piechart-panel,grafana-worldmap-panel  
volumes:  
- grafana\_data:/var/lib/grafana  
- ./grafana/dashboards:/var/lib/grafana/dashboards  
- ./grafana/provisioning:/etc/grafana/provisioning  
networks:  
- devops-network

**Portainer for Docker management**

portainer:  
image: portainer/portainer-ce:latest  
container\_name: portainer  
restart: unless-stopped  
ports:  
- "9443:9443"  
volumes:  
- /var/run/docker.sock:/var/run/docker.sock  
- portainer\_data:/data  
networks:  
- devops-network

volumes:  
jenkins\_data:  
sonarqube\_data:  
sonarqube\_logs:  
sonarqube\_extensions:  
postgres\_data:  
sqlserver\_data:  
redis\_data:  
seq\_data:  
prometheus\_data:  
grafana\_data:  
portainer\_data:

networks:  
devops-network:  
driver: bridge" whant i need to update

You are **very close** to the solution!  
The warning in your screenshot means Jenkins **still does not see JDK 17 at** /usr/lib/jvm/java-17-openjdk-amd64 **inside its container**.

This happens because your devops-compose.yml is still using

image: jenkins/jenkins:lts

instead of

build: ./jenkins

so your custom Dockerfile (which installs JDK 17) is **not being used**.

**How to Fix (Step by Step)**

**1. Create the Custom Jenkins Dockerfile**

Make sure you have this file:  
/vagrant/docker/jenkins/Dockerfile

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

**2. Update Your Compose File**

**Change this in your devops-compose.yml:**

jenkins:  
 image: jenkins/jenkins:lts

**TO:**

jenkins:  
 build: ./jenkins

Make sure the rest of the service definition stays the same.

**3. Rebuild and Restart Jenkins**

cd /vagrant/docker  
docker compose -f devops-compose.yml build jenkins  
docker compose -f devops-compose.yml up -d

**4. Verify JDK 17 is Available in Jenkins Container**

docker exec -it jenkins java -version

It should show Java 17.

**5. Configure Jenkins JDK Tool**

* Go to **Gérer Jenkins > Configuration des outils globaux**.
* In the JDK section:
  + **Name:** OpenJDK 17
  + **JAVA\_HOME:** /usr/lib/jvm/java-17-openjdk-amd64
  + **Uncheck** "Install automatically"
* Save.

**The warning will disappear. Jenkins will use JDK 17 for your jobs.**

**Summary Table**

|  |  |
| --- | --- |
| Step | Action/Command |
| Create Dockerfile | /vagrant/docker/jenkins/Dockerfile |
| Update compose file | build: ./jenkins for Jenkins service |
| Build & restart Jenkins | docker compose -f devops-compose.yml build jenkins && docker compose -f devops-compose.yml up -d |
| Check Java version in Jenkins | docker exec -it jenkins java -version |
| Configure JDK in Jenkins UI | JAVA\_HOME: /usr/lib/jvm/java-17-openjdk-amd64 |

**If you do these steps, Jenkins will see JDK 17 and the warning will be gone!**  
If you need a ready-to-copy compose snippet or more help, just ask!

⁂

**now its oak lets back to the jenkins jdk**

Your screenshot shows that Jenkins **still cannot see** /usr/lib/jvm/java-17-openjdk-amd64 inside its container.  
This means you are **still using the default Jenkins image** (image: jenkins/jenkins:lts) in your devops-compose.yml instead of your custom image with JDK 17.

**How to Fix the Jenkins JDK Path Warning**

**1. Update your devops-compose.yml**

**Find this in your Jenkins service:**

image: jenkins/jenkins:lts

**Replace it with:**

build: ./jenkins

This tells Docker Compose to build Jenkins from your custom Dockerfile (which installs JDK 17).

**2. Make Sure the Dockerfile Exists**

Your file tree should look like:

/vagrant/docker/  
 devops-compose.yml  
 /jenkins/  
 Dockerfile

And your /vagrant/docker/jenkins/Dockerfile should contain:

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && apt-get install -y openjdk-17-jdk  
USER jenkins

**3. Rebuild and Restart Jenkins**

From /vagrant/docker:

docker compose -f devops-compose.yml build jenkins  
docker compose -f devops-compose.yml up -d

**4. Verify JDK 17 is Available in Jenkins Container**

Check inside the Jenkins container:

docker exec -it jenkins java -version

You should see Java 17.

**5. Now Set JAVA\_HOME in Jenkins**

* Go to **Gérer Jenkins > Configuration des outils globaux**.
* In the JDK section:
  + **Name:** OpenJDK 17
  + **JAVA\_HOME:** /usr/lib/jvm/java-17-openjdk-amd64
  + **Uncheck** "Install automatically"
* Save.

The warning will disappear. Jenkins will use JDK 17 for your builds.

**Summary Table**

|  |  |
| --- | --- |
| Step | Command/Action |
| Update compose file | build: ./jenkins for Jenkins service |
| Ensure Dockerfile exists | /vagrant/docker/jenkins/Dockerfile |
| Build & restart Jenkins | docker compose -f devops-compose.yml build jenkins && docker compose -f devops-compose.yml up -d |
| Check Java version in Jenkins | docker exec -it jenkins java -version |
| Configure JDK in Jenkins UI | JAVA\_HOME: /usr/lib/jvm/java-17-openjdk-amd64 |

**If you follow these steps, Jenkins will see JDK 17 and the warning will be gone!**  
If you need a ready-to-copy compose snippet or more help, just ask!

⁂

**now i dont find .net sdk to config it**

If you don’t see the **.NET SDK** configuration section in Jenkins under **Manage Jenkins > Global Tool Configuration**, it’s because the **.NET SDK plugin** is not installed yet.

**How to add .NET SDK support in Jenkins**

**1. Install the .NET SDK Plugin**

* Go to **Manage Jenkins > Manage Plugins > Available** tab.
* Search for **"MSBuild"** or **".NET SDK"** plugin (depending on plugin names, often it’s called **"MSBuild Plugin"** or **".NET SDK Installer"**).
* Select and install the plugin.
* Restart Jenkins if required.

**2. Configure .NET SDK in Global Tool Configuration**

* After plugin installation, go to **Manage Jenkins > Global Tool Configuration**.
* You should see a **.NET SDK** section.
* Click **Add .NET SDK**.
* Fill in:
  + **Name:** e.g. .NET 8
  + **Install automatically:** (optional, if you want Jenkins to download it)
  + Or specify the **path** to the .NET SDK inside your Jenkins container, e.g.

/usr/share/dotnet/dotnet

(Make sure .NET SDK is installed in your Jenkins container.)

**3. Verify .NET SDK is installed in Jenkins container**

If you installed .NET SDK in your Jenkins container (or your custom Jenkins Dockerfile), verify it with:

docker exec -it jenkins dotnet --version

If it’s not installed, you can add installation commands to your custom Jenkins Dockerfile, for example:

FROM jenkins/jenkins:lts  
USER root  
RUN apt-get update && \  
 apt-get install -y wget apt-transport-https && \  
 wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb && \  
 dpkg -i packages-microsoft-prod.deb && \  
 apt-get update && \  
 apt-get install -y dotnet-sdk-8.0  
USER jenkins

Then rebuild your Jenkins container.

**Summary**

|  |  |
| --- | --- |
| Step | Action |
| Install .NET SDK plugin | Manage Jenkins > Manage Plugins > Available > Search & install |
| Configure .NET SDK | Manage Jenkins > Global Tool Configuration > Add .NET SDK |
| Verify .NET SDK in container | docker exec -it jenkins dotnet --version |
| Install .NET SDK in Dockerfile | Add Microsoft repo and install dotnet-sdk-8.0 |

If you want, I can help you write the full Dockerfile snippet or guide you through plugin installation!

**msbuild is the same as sdk .net ! with th econfiguration with jenkins**

No, **MSBuild and the .NET SDK are not the same thing**—especially for Jenkins configuration and CI/CD with modern .NET projects.

**MSBuild vs .NET SDK in Jenkins**

|  |  |  |
| --- | --- | --- |
| Tool | Purpose | Typical Use in Jenkins |
| **MSBuild** | The build engine for traditional .NET Framework projects (Windows, .csproj, .sln files, etc.) | Used for building legacy .NET Framework or .NET Core projects on Windows agents. |
| **.NET SDK** | The full development kit for modern .NET (Core, 5, 6, 7, 8+), includes dotnet CLI and MSBuild. | Used for building, testing, publishing, and running modern .NET projects (cross-platform). |

**What Does This Mean for Jenkins?**

* **If you build modern .NET projects** (using dotnet build, dotnet test, etc.), you need the **.NET SDK**.
* **If you build old .NET Framework projects** (using msbuild.exe), you need **MSBuild** (usually only on Windows).

**On Linux or in Docker:**

* You almost always want the **.NET SDK** (not just MSBuild).

**How to Enable .NET SDK in Jenkins**

1. **Install the .NET SDK in your Jenkins container.**
   * Add this to your Jenkins Dockerfile (after the JDK install):

# Install .NET 8 SDK (example for Ubuntu 22.04)  
RUN wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb \  
 && dpkg -i packages-microsoft-prod.deb \  
 && apt-get update \  
 && apt-get install -y dotnet-sdk-8.0

* + Rebuild and restart your Jenkins container.

1. **Install the Jenkins .NET SDK Plugin:**
   * Go to **Manage Jenkins > Manage Plugins > Available**.
   * Search for **.NET SDK** or **.NET Core SDK** plugin (sometimes called “.NET SDK Installer”).
   * Install it and restart Jenkins if needed.
2. **Configure .NET SDK in Jenkins:**
   * Go to **Manage Jenkins > Global Tool Configuration**.
   * You should now see a **.NET SDK** section.
   * Click **Add .NET SDK**.
   * Set:
     + **Name:** e.g. .NET 8
     + **Path:** /usr/share/dotnet/dotnet (or just /usr/bin/dotnet, check with which dotnet in your container)
     + Or enable “Install automatically” if you want Jenkins to manage SDK versions.

**Summary**

* **MSBuild** ≠ **.NET SDK**.
* For modern .NET (Core, 5, 6, 7, 8+), always use the **.NET SDK**.
* Install the **.NET SDK** in your Jenkins container and configure it in Jenkins’ Global Tool Configuration.
* Only use **MSBuild** for legacy .NET Framework projects on Windows.

If you want the exact Dockerfile snippet or step-by-step for installing the .NET SDK in your Jenkins container, let me know!

⁂

**this can replace th** [**e.net**](http://e.net) **sdk**

No, the **MSBuild plugin is NOT the same as the .NET SDK plugin** for Jenkins, and it does not replace the .NET SDK for modern .NET (Core/5/6/7/8) projects.

**What You See in Your Screenshot**

You have found the **.NET SDK Support** plugin in Jenkins, which is exactly what you need for managing .NET SDKs (Core and 5.0+) as global tools in Jenkins[[5]](#fn5).

**Difference: MSBuild vs .NET SDK**

|  |  |  |  |
| --- | --- | --- | --- |
| Plugin/Tool | Use Case | Platform | Typical Projects |
| MSBuild Plugin | Builds classic .NET Framework and Visual Studio projects | Windows only | .NET Framework, legacy |
| .NET SDK Plugin | Builds modern .NET projects using dotnet CLI | Cross-platform | .NET Core, .NET 5/6/7/8 |

* **MSBuild** is for legacy Windows/.NET Framework projects and requires msbuild.exe (usually only on Windows)[[6]](#fn6)[[7]](#fn7)[[8]](#fn8).
* **.NET SDK** is for all modern .NET projects and uses the dotnet CLI, which is cross-platform and includes its own MSBuild under the hood[[9]](#fn9)[[10]](#fn10).

**What Should You Do?**

* **For modern .NET projects:**  
  Install and use the **.NET SDK Support** plugin (the one in your screenshot).  
  This lets you configure and use the dotnet CLI for building, testing, and publishing .NET Core/5/6/7/8 projects in Jenkins.
* **For legacy .NET Framework projects on Windows:**  
  Use the **MSBuild** plugin and configure it with the path to msbuild.exe[[6]](#fn6)[[7]](#fn7)[[8]](#fn8).

**Summary Table**

|  |  |  |
| --- | --- | --- |
| Project Type | Jenkins Plugin to Use | How to Build in Jenkins |
| .NET Core / .NET 5+ | .NET SDK Support Plugin | dotnet build, dotnet test, etc. |
| .NET Framework (Windows) | MSBuild Plugin | msbuild.exe build step |

**In your case:**

If you are working with modern .NET (Core/5/6/7/8), install the **.NET SDK Support** plugin as shown in your screenshot.  
This will allow you to configure .NET SDKs as global tools and use the dotnet CLI in your Jenkins pipelines.

If you need help installing the plugin or configuring the SDK, let me know!

⁂

**okay now wwe need to repeat all the configuration from 0 just am installed the plugins and i need to " Configure Global Tools in Jenkins" .. guide me step by step**

Absolutely! Let’s do a **step-by-step guide to configure all the essential Global Tools in Jenkins** after a fresh plugin install (including .NET SDK, Git, NodeJS, SonarQube Scanner, Docker, and JDK).

**🛠️ Step-by-Step: Configure Global Tools in Jenkins**

**1. Accéder à la configuration des outils globaux**

* Dans Jenkins, va dans **Gérer Jenkins** (Manage Jenkins).
* Clique sur **Configuration des outils globaux** (Global Tool Configuration).

**2. Configurer chaque outil**

**A. JDK (Java Development Kit)**

1. Trouve la section **JDK**.
2. Clique sur **Ajouter JDK**.
3. **Name** : OpenJDK 17
4. **Décoche** “Installer automatiquement”.
5. **JAVA\_HOME** :

/usr/lib/jvm/java-17-openjdk-amd64

(Ce chemin doit exister dans ton conteneur Jenkins.)

**B. Git**

1. Trouve la section **Git**.
2. Clique sur **Ajouter Git**.
3. **Name** : Default ou Git
4. **Path to Git executable** :  
   Laisse git (si installé dans le PATH du conteneur Jenkins).
5. Tu peux cocher “Installer automatiquement” si tu veux que Jenkins gère l’installation.

**C. NodeJS**

1. Trouve la section **NodeJS** (nécessite le plugin NodeJS).
2. Clique sur **Ajouter NodeJS**.
3. **Name** : Node 18
4. Coche “Installer automatiquement”.
5. Sélectionne la version 18.x.
6. Dans **Global npm packages to install**, ajoute :

@angular/cli@19

**D. Docker**

1. Trouve la section **Docker** (nécessite le plugin Docker).
2. Clique sur **Ajouter Docker**.
3. **Name** : Docker
4. Chemin de l’exécutable Docker :

/usr/bin/docker

(Vérifie avec which docker dans le conteneur Jenkins.)

1. Tu peux cocher “Installer automatiquement” si tu veux.

**E. SonarQube Scanner**

1. Trouve la section **SonarQube Scanner** (nécessite le plugin SonarQube Scanner).
2. Clique sur **Ajouter SonarQube Scanner**.
3. **Name** : SonarQube
4. Coche “Installer automatiquement” pour la dernière version.

**F. .NET SDK**

1. Trouve la section **.NET SDK** (après installation du plugin .NET SDK Support).
2. Clique sur **Ajouter .NET SDK**.
3. **Name** : .NET 8 (ou la version installée)
4. **Path to dotnet executable** :

/usr/bin/dotnet

(ou /usr/share/dotnet/dotnet selon l’installation dans le conteneur Jenkins)

1. Tu peux cocher “Installer automatiquement” si tu veux que Jenkins gère l’installation.

**3. Enregistrer la configuration**

* Clique sur **Enregistrer** en bas de la page.

**Résumé visuel (exemple pour chaque outil)**

|  |  |  |
| --- | --- | --- |
| Outil | Name | Path/Version/Options |
| JDK | OpenJDK 17 | /usr/lib/jvm/java-17-openjdk-amd64 |
| Git | Default | git |
| NodeJS | Node 18 | 18.x, @angular/cli@19 (global npm package) |
| Docker | Docker | /usr/bin/docker |
| SonarQube Scanner | SonarQube | Installer automatiquement |
| .NET SDK | .NET 8 | /usr/bin/dotnet ou /usr/share/dotnet/dotnet |

**4. Vérification rapide**

Pour chaque outil, tu peux vérifier dans le conteneur Jenkins avec :

docker exec -it jenkins <tool> --version

Exemple :

* docker exec -it jenkins dotnet --version
* docker exec -it jenkins java -version
* docker exec -it jenkins git --version
* docker exec -it jenkins node --version

**Besoin d’aide pour une section spécifique ou une capture d’écran ? Dis-le-moi !**

Tu peux maintenant passer à la configuration des intégrations (SonarQube, Docker, etc.) dans **Gérer Jenkins > Configurer le système** si besoin.

⁂

**j'ai la besoins de verifier ce chemin "Trouve la section .NET SDK (après installation du plugin .NET SDK Support).**

Clique sur Ajouter .NET SDK.

Name : .NET 8 (ou la version installée)

Path to dotnet executable :"

Parfait, tu es à l’étape d’installation du plugin **.NET SDK Support** dans Jenkins !  
Pour configurer le chemin du binaire dotnet, tu dois d’abord vérifier où il se trouve dans ton conteneur Jenkins.

**Comment vérifier le chemin du binaire dotnet dans ton conteneur Jenkins**

1. **Ouvre un terminal sur ta VM.**
2. Exécute un shell dans le conteneur Jenkins :

docker exec -it jenkins bash

1. Dans le shell du conteneur, tape :

which dotnet

ou

whereis dotnet

* + Le résultat typique est souvent /usr/bin/dotnet ou /usr/share/dotnet/dotnet.

1. Tu peux aussi vérifier la version pour t'assurer qu'il est bien installé :

dotnet --version

**Exemple de résultat attendu**

root@container:/# which dotnet  
/usr/bin/dotnet  
  
root@container:/# dotnet --version  
8.0.100

**Que mettre dans Jenkins ?**

* **Name** : .NET 8 (ou la version affichée par dotnet --version)
* **Path to dotnet executable** : Le chemin que tu as trouvé, par exemple :

/usr/bin/dotnet

ou

/usr/share/dotnet/dotnet

**Résumé**

1. Installe le plugin **.NET SDK Support**.
2. Vérifie le chemin du binaire dotnet dans le conteneur Jenkins avec which dotnet.
3. Utilise ce chemin dans la configuration Jenkins > Outils globaux > .NET SDK.

**Besoin d’aide pour l’installation du SDK dans le Dockerfile ou pour la configuration dans Jenkins ? Dis-le-moi !**

⁂

**after the conifguration this steps are necessar ! "Phase 5: Security and Firewall Configuration**

5.1 Configure VM Firewall  
bash# SSH into VM  
vagrant ssh

**Configure UFW firewall**

sudo ufw enable  
sudo ufw allow 8080 # Jenkins  
sudo ufw allow 9000 # SonarQube  
sudo ufw allow 3000 # Grafana  
sudo ufw allow 9090 # Prometheus  
sudo ufw allow 9443 # Portainer  
sudo ufw allow 1433 # SQL Server  
sudo ufw allow 6379 # Redis  
sudo ufw allow 5341 # Seq  
sudo ufw allow 80 # Nginx  
sudo ufw allow 22 # SSH  
sudo ufw status  
5.2 Secure Jenkins

Enable CSRF Protection: Manage Jenkins > Configure Global Security  
Setup Matrix-based Security: Configure user permissions  
Enable Agent-to-Master Security: Configure build agents  
Regular Backups: Configure backup strategy"

Oui, ces étapes de la phase 5 (Configuration de la sécurité et du pare-feu) sont **nécessaires et fortement recommandées** après avoir configuré tes services Docker (Jenkins, SonarQube, Grafana, etc.) pour sécuriser ta VM et limiter les accès réseau.

**Pourquoi ces étapes sont importantes**

**1. Configurer le pare-feu (UFW) sur la VM**

* Docker, par défaut, contourne souvent les règles UFW, ce qui peut exposer tes conteneurs au réseau même si UFW est activé[[11]](#fn11)[[12]](#fn12)[[13]](#fn13).
* Il faut donc configurer UFW correctement pour autoriser uniquement les ports nécessaires (8080 pour Jenkins, 9000 pour SonarQube, etc.) et bloquer le reste.
* Pour que UFW fonctionne bien avec Docker, il existe des solutions comme le script **ufw-docker** qui permet d’intégrer Docker dans la gestion des règles UFW[[13]](#fn13)[[14]](#fn14).
* Sans cette configuration, tes conteneurs peuvent être exposés à des accès non désirés malgré un pare-feu activé.

**2. Sécuriser Jenkins**

* Activer la protection CSRF, configurer les permissions avec la sécurité basée sur les rôles (Matrix-based Security) et sécuriser les communications entre agents et master est indispensable pour éviter les attaques et accès non autorisés.
* Mettre en place une stratégie de sauvegarde régulière protège tes données et configurations Jenkins.

**Résumé des actions recommandées**

|  |  |
| --- | --- |
| Étape | Pourquoi c’est important |
| Activer UFW et ouvrir ports nécessaires | Limiter les accès réseau aux services exposés |
| Utiliser ufw-docker ou configurer Docker/UFW ensemble | Faire cohabiter Docker et UFW sans contournement |
| Activer protection CSRF Jenkins | Empêcher les attaques par falsification de requête |
| Configurer sécurité utilisateur Jenkins | Contrôler précisément qui peut faire quoi |
| Sécuriser communication agents-master | Protéger les échanges sensibles |
| Mettre en place des sauvegardes régulières | Prévenir la perte de données |

**Conseils pratiques**

* **Avant d’activer UFW**, teste la connectivité de tes services.
* Si tu utilises Docker, envisage d’installer et configurer [ufw-docker](https://github.com/chaifeng/ufw-docker) ou [ufw-docker-automated](https://github.com/shinebayar-g/ufw-docker-automated) pour gérer automatiquement les règles.
* Pour Jenkins, active la sécurité globale dans **Manage Jenkins > Configure Global Security**.
* Planifie des sauvegardes automatiques de Jenkins (plugins comme ThinBackup ou via scripts externes).

**En conclusion**

**Oui, ces étapes de configuration de la sécurité et du pare-feu sont nécessaires** pour protéger ta VM et tes services Docker en production ou en environnement accessible réseau.  
Elles garantissent que seuls les ports essentiels sont ouverts et que Jenkins est sécurisé contre les attaques courantes.

Si tu souhaites, je peux te fournir un guide détaillé pour configurer UFW avec Docker ou sécuriser Jenkins étape par étape.

⁂

**what is the abbreviation of ufw**

The abbreviation **UFW** stands for **Uncomplicated Firewall**.

It is a user-friendly frontend for managing firewall rules on Linux systems, designed to make configuring the Linux kernel's built-in firewall (iptables/netfilter) easier and more accessible.

**th econtainers are runinng ! can make a problem "vagrant@devops-vm:~$ sudo ufw enable**

Command may disrupt existing ssh connections. Proceed with operation (y|n)? "

When you run sudo ufw enable on your VM over SSH, the warning:

Command may disrupt existing ssh connections. Proceed with operation (y|n)?

means that enabling the firewall **could block your current SSH session**, causing you to be disconnected.

**Should you proceed?**

* **If you have NOT allowed SSH port (usually 22) explicitly before enabling UFW, enabling it will very likely cut your SSH connection.**
* **If you have already allowed SSH port (with sudo ufw allow 22 or sudo ufw allow ssh), then enabling UFW is generally safe and your SSH connection will remain active.**

**Recommended safe procedure before enabling UFW:**

1. **Allow SSH port explicitly:**

sudo ufw allow 22/tcp

or

sudo ufw allow ssh

1. **Allow all other necessary ports for your containers:**

sudo ufw allow 8080 # Jenkins  
sudo ufw allow 9000 # SonarQube  
sudo ufw allow 3000 # Grafana  
sudo ufw allow 9090 # Prometheus  
sudo ufw allow 9443 # Portainer  
sudo ufw allow 1433 # SQL Server  
sudo ufw allow 6379 # Redis  
sudo ufw allow 5341 # Seq  
sudo ufw allow 80 # Nginx (if used)

1. **Check rules before enabling:**

sudo ufw status verbose

Make sure SSH and other needed ports are allowed.

1. **Enable UFW:**

sudo ufw enable

**What happens if you get disconnected?**

* If you lose SSH access, you may need to access the VM via your hypervisor console or Vagrant GUI to disable UFW:

sudo ufw disable

**Additional notes:**

* UFW allows **RELATED** and **ESTABLISHED** connections by default, so existing SSH sessions often survive enabling UFW if SSH port is allowed.
* If you use a **non-standard SSH port**, you must allow that port explicitly.
* Docker can sometimes bypass UFW rules; consider using tools like **ufw-docker** to better integrate Docker and UFW.

**Summary**

|  |  |
| --- | --- |
| Step | Command |
| Allow SSH | sudo ufw allow ssh or sudo ufw allow 22/tcp |
| Allow other service ports | sudo ufw allow <port> |
| Check rules | sudo ufw status verbose |
| Enable firewall | sudo ufw enable |

**In conclusion:**  
You **can enable UFW safely if you first allow SSH port**, otherwise you risk losing your SSH connection.

If you want, I can help you write a safe script or step-by-step commands to configure UFW without losing SSH access.

⁂

**"vagrant@devops-vm:~$ sudo ufw allow ssh**

Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 8080  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 9000  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 3000  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 9090  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 9443  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 1433  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 6379  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 5341  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw allow 80  
Rules updated  
Rules updated (v6)  
vagrant@devops-vm:~$ sudo ufw status verbose  
Status: inactive  
vagrant@devops-vm:~$" verify

Your commands to **allow all necessary ports** with UFW have been successfully registered, as indicated by the messages:

Rules updated  
Rules updated (v6)

This means both IPv4 and IPv6 rules were added.

**Current Status**

When you run:

sudo ufw status verbose

you get:

Status: inactive

This means **UFW is currently disabled** on your VM.

**What to do next?**

You have correctly opened all required ports **before enabling UFW**, which is perfect.

Now you can safely **enable UFW** without losing SSH access:

sudo ufw enable

You will see a warning:

Command may disrupt existing ssh connections. Proceed with operation (y|n)?

Since you allowed SSH port explicitly, type y and press Enter.

**After enabling UFW**

* Check the status again:

sudo ufw status verbose

You should see:

* Status: active
* The list of allowed ports (22, 80, 8080, 9000, 3000, 9090, 9443, 1433, 6379, 5341)
* Rules for both IPv4 and IPv6.

**Summary**

|  |  |
| --- | --- |
| Step | Command |
| Allow SSH port | sudo ufw allow ssh |
| Allow service ports | sudo ufw allow <port> |
| Check current status | sudo ufw status verbose |
| Enable firewall | sudo ufw enable |
| Verify firewall active | sudo ufw status verbose |

If you want, I can guide you on how to test your services are reachable after enabling the firewall!

**now is okay !"vagrant@devops-vm:~$ sudo ufw status verbose**

Status: active  
Logging: on (low)  
Default: deny (incoming), allow (outgoing), deny (routed)  
New profiles: skip

To Action From

22/tcp ALLOW IN Anywhere  
8080 ALLOW IN Anywhere  
9000 ALLOW IN Anywhere  
3000 ALLOW IN Anywhere  
9090 ALLOW IN Anywhere  
9443 ALLOW IN Anywhere  
1433 ALLOW IN Anywhere  
6379 ALLOW IN Anywhere  
5341 ALLOW IN Anywhere  
80 ALLOW IN Anywhere  
22/tcp (v6) ALLOW IN Anywhere (v6)  
8080 (v6) ALLOW IN Anywhere (v6)  
9000 (v6) ALLOW IN Anywhere (v6)  
3000 (v6) ALLOW IN Anywhere (v6)  
9090 (v6) ALLOW IN Anywhere (v6)  
9443 (v6) ALLOW IN Anywhere (v6)  
1433 (v6) ALLOW IN Anywhere (v6)  
6379 (v6) ALLOW IN Anywhere (v6)  
5341 (v6) ALLOW IN Anywhere (v6)  
80 (v6) ALLOW IN Anywhere (v6)

vagrant@devops-vm:~$"

Parfait ! 🎉

Ton pare-feu UFW est désormais **actif** et correctement configuré pour autoriser tous les ports nécessaires à tes services Docker (Jenkins, SonarQube, Grafana, etc.), aussi bien en IPv4 qu’en IPv6.

**Ce que cela signifie concrètement :**

* **Toutes les connexions entrantes sont bloquées par défaut**, sauf celles explicitement autorisées (SSH, Jenkins, SonarQube, etc.).
* Tes services sont accessibles uniquement sur les ports que tu as ouverts.
* Ton serveur est mieux protégé contre les accès non autorisés.

**Prochaines étapes recommandées :**

* **Tester la connectivité** vers tes services depuis ta machine hôte ou un navigateur :
  + Jenkins : http://<IP\_VM>:8080
  + SonarQube : http://<IP\_VM>:9000
  + Grafana : http://<IP\_VM>:3000
  + etc.
* **Surveiller les logs UFW** si besoin :

sudo tail -f /var/log/ufw.log

* Continuer la sécurisation de Jenkins (CSRF, permissions, agents, backups).

Si tu as besoin d’aide pour tester ou sécuriser davantage, n’hésite pas à demander !  
Bravo pour cette belle avancée ! 🚀

**now i need to knpw hwo i can stop all the containers and after that i can run it both off them in the same time or i can run it container by cointaienr**

Great question! Here’s how you can **stop and start your Docker containers**, either all at once or individually.

**1. Stop All Containers at Once**

If you want to stop **all running containers** on your VM:

docker stop $(docker ps -q)

* docker ps -q lists all running container IDs.
* docker stop stops them gracefully.

**2. Stop All Containers Defined in Your Compose File**

If you want to stop all containers defined in your Docker Compose setup (e.g., your devops-compose.yml):

docker compose -f devops-compose.yml down

* This stops and removes containers, networks, and default volumes created by Compose.
* Use this when you want to stop and clean up your Compose stack.

**3. Start All Containers at Once**

To start all containers defined in your Compose file:

docker compose -f devops-compose.yml up -d

* -d runs containers in detached mode (background).
* This starts all services together.

**4. Start/Stop Containers Individually**

**Stop a single container:**

docker stop <container\_name\_or\_id>

Example:

docker stop jenkins

**Start a single container:**

docker start <container\_name\_or\_id>

Example:

docker start jenkins

**Summary Table**

|  |  |
| --- | --- |
| Action | Command |
| Stop all running containers | docker stop $(docker ps -q) |
| Stop all Compose containers | docker compose -f devops-compose.yml down |
| Start all Compose containers | docker compose -f devops-compose.yml up -d |
| Stop one container | docker stop <container\_name> |
| Start one container | docker start <container\_name> |

**Notes**

* When you stop containers individually, dependencies are not managed automatically.
* Using docker compose up and down is recommended to manage the full stack together.
* You can mix approaches depending on your needs.

If you want, I can help you write scripts or aliases to simplify these commands!

**"vagrant@devops-vm:~$ docker compose -f devops-compose.yml down**

open /home/vagrant/devops-compose.yml: no such file or directory  
vagrant@devops-vm:~$ docker stop  
docker: 'docker stop' requires at least 1 argument

Usage: docker stop [OPTIONS] CONTAINER [CONTAINER...]

See 'docker stop --help' for more information  
vagrant@devops-vm:~$ docker stop docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
vagrant@devops-vm:~$ docker compose -f devops-compose.yml up -d  
open /home/vagrant/devops-compose.yml: no such file or directory  
vagrant@devops-vm:~$"

The error message:

open /home/vagrant/devops-compose.yml: no such file or directory

means that **Docker Compose cannot find your compose file at the path /home/vagrant/devops-compose.yml**.

**How to fix this**

**1. Locate your devops-compose.yml file**

From your previous messages, it looks like your compose file is in:

/vagrant/docker/devops-compose.yml

**2. Use the correct path when running Docker Compose commands**

Run the commands from the directory containing the compose file or specify the full path:

cd /vagrant/docker  
docker compose -f devops-compose.yml down  
docker compose -f devops-compose.yml up -d

Or run directly with the full path:

docker compose -f /vagrant/docker/devops-compose.yml down  
docker compose -f /vagrant/docker/devops-compose.yml up -d

**Summary**

|  |  |
| --- | --- |
| Problem | Solution |
| Compose file not found | Use correct path to compose file |
| Running commands from wrong dir | cd /vagrant/docker before running commands |

**Additional tip**

If you want to avoid specifying -f and path every time, just run commands from the directory where your compose file is located:

cd /vagrant/docker  
docker compose down  
docker compose up -d

Let me know if you want help with scripts or aliases to simplify your workflow!

**i dont need to aceed i can run the command from any location with \*\* right !**

Exactly! You **can run docker compose commands from any directory** as long as you specify the full path to your compose file with the -f option.

**How to run docker compose commands from any location**

Use the -f flag followed by the **full absolute path** to your compose file:

docker compose -f /vagrant/docker/devops-compose.yml up -d  
docker compose -f /vagrant/docker/devops-compose.yml down

This tells Docker Compose exactly where to find your compose file, so you don’t need to cd into the directory.

**Summary**

|  |  |
| --- | --- |
| Scenario | Command example |
| Run from compose file folder | docker compose up -d |
| Run from any folder | docker compose -f /full/path/to/devops-compose.yml up -d |

If you want, you can create a shell alias or script to simplify this command.

**In short:**  
**Yes, you can run Docker Compose commands from anywhere if you specify the compose file path with -f.**

1. <https://developer.hashicorp.com/vagrant/docs/networking/forwarded_ports>

1. <https://www.bogotobogo.com/DevOps/Vagrant/Vagrant_Networking_Port_Forwarding.php>

1. <https://github.com/viveksatasiya/vagrant-prometheus-grafana>
2. paste.txt
3. image.jpg

1. <https://stackoverflow.com/questions/10227967/build-net-application-in-jenkins-using-msbuild>

1. <https://plugins.jenkins.io/msbuild/>

1. <https://www.couchbase.com/blog/continuous-deployment-with-jenkins-and-net/>

1. <https://dev.to/randazraik/msbuild-and-project-files-3d4o>

1. <https://stackoverflow.com/questions/66349892/whats-the-difference-between-using-dotnet-and-msbuild-for-building-net-applica>

1. <https://docs.docker.com/engine/network/packet-filtering-firewalls/>

1. <https://blog.jarrousse.org/2023/03/18/how-to-use-ufw-firewall-with-docker-containers/>

1. <https://community.bigbeartechworld.com/t/securing-docker-with-ufw-a-complete-guide/2443>

1. <https://github.com/shinebayar-g/ufw-docker-automated>