**Cài đặt Control Plane và Worker node:**

Dưới đây là hướng dẫn **đầy đủ và được cập nhật 2024**, có thể áp dụng cho Ubuntu 20.04, 22.04 (Jammy) và mới hơn:

**1. Cài đặt công cụ hỗ trợ HTTPS cho APT**

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl gpg

**2. Thêm khóa GPG và tạo source list đúng cách**

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | gpg --dearmor | sudo tee /etc/apt/keyrings/kubernetes-apt-keyring.gpg > /dev/null

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

**3. Cập nhật và cài đặt Kubernetes:**

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

**Kiểm tra sau khi cài đặt**

kubectl version --client

kubeadm version

kubelet –version

***Tiếp theo là triển khai Kubernetes cluster multi-node (1 control plane + 1 worker)***

***Bước 1: Cấu hình hệ thống (trên cả 2 VM Ubuntu)***

***Tắt swap****:*

sudo swapoff -a

sudo sed -i '/ swap / s/^/#/' /etc/fstab

**Bật các kernel module cần thiết:**

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

br\_netfilter

EOF

sudo modprobe br\_netfilter

**Cấu hình sysctl:**

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

EOF

sudo sysctl --system

***Bước 2: Cài container runtime (nếu chưa có)***

***Kubernetes cần một container runtime. Dùng containerd là ổn định và phổ biến nhất:***

sudo apt-get install -y containerd

sudo mkdir -p /etc/containerd

containerd config default | sudo tee /etc/containerd/config.toml

***Chỉnh cấu hình để SystemdCgroup = true:***

sudo nano /etc/containerd/config.toml

***Tìm dòng:***

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]

***Và chỉnh:***

SystemdCgroup = true

***Sau đó:***

sudo systemctl restart containerd

sudo systemctl enable containerd

***Bước 3: Tạo cluster trên control plane node***

***Trên VM control plane:***

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

***Dùng lệnh:*** kubeadm join ... dành cho worker. hoặc có thể lấy lại token này bằng lệnh:

sudo kubeadm token create --print-join-command

***Sau đó cấu hình kubectl:***

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

***Bước 4: Cài Flannel CNI (hoặc Calico, Cilium, etc.)***

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

***Bước 5: Thêm worker node vào cluster***

Trên node worker, dùng lệnh kubeadm join mà bạn đã nhận từ bước 3. Ví dụ:

sudo kubeadm join 192.168.1.100:6443 --token abcdef.0123456789abcdef \

--discovery-token-ca-cert-hash sha256:xxxxxxxxxxxxxxxxxxxx

***Kiểm tra cluster (trên control plane)***

kubectl get nodes

***Mặc định, kubernetes không cho phép chạy pod trên nodo control plane, để cho phép pod chạy trên node control-plane (dành cho môi trường học tập, lab, máy ảo test)***

***Chạy lệnh sau để xóa taint:***

kubectl taint nodes <node-name> node-role.kubernetes.io/control-plane-

***Ví dụ, nếu node bạn tên là control-plane, thì:***

kubectl taint nodes control-plane node-role.kubernetes.io/control-plane-

Dấu - ở cuối lệnh là để xóa taint.

**Cài đặt Ingress Control (NodePort) cho Kubernetes**

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.10.0/deploy/static/provider/baremetal/deploy.yaml>

***Kiểm tra Ingress Control***

kubectl get svc -n ingress-nginx

Hoặc

kubectl get serice –all-namespaces

***Xác định ingressClass để cấu hình cho ingress service***

kubectl get ingressclass

***Kiểm tra cấu hình của một Ingress***

kubectl get ingress ingress-app1 -o yaml

kubectl get ingress rancher -n cattle-system -o yaml

***Kiểm tra logs Ingress***

kubectl logs -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx

***Dùng forward để kiểm tra một ứng dụng trong pod với Cluster-IP:***

kubectl port-forward svc/cluster-ip-app1 18080:8080

Sau khi forword có thể kiểm tra ứng dụng thông qua localhost:18080

**Cài đặt cloudflared**

***1. Thêm repo Cloudflare***

sudo apt install -y curl gnupg lsb-release

curl -fsSL https://pkg.cloudflare.com/cloudflare-main.gpg | sudo tee /usr/share/keyrings/cloudflare-main.gpg >/dev/null

echo "deb [signed-by=/usr/share/keyrings/cloudflare-main.gpg] https://pkg.cloudflare.com/cloudflared $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/cloudflared.list

***2. Cài đặt***

sudo apt update

sudo apt install cloudflared

***Đăng nhập vào Cloudflare và tạo Tunnel***

cloudflared login

***Lệnh này sẽ mở trình duyệt → bạn đăng nhập Cloudflare → chọn domain bạn muốn dùng.***

***Sau khi đăng nhập thành công, Cloudflare sẽ lưu chứng chỉ xác thực tại:***

~/.cloudflared/cert.pem

***Tạo tunnel mới***

cloudflared tunnel create my-tunnel

***Cloudflare sẽ tạo tunnel và lưu thông tin vào:***

~/.cloudflared/my-tunnel.json

***Chạy foreground để test:***

cloudflared tunnel run my-tunnel

***Hoặc cấu hình chạy dưới dạng systemd (background):***

sudo cloudflared service install

***Tạo một service kích hoạt tunnel cloudflare chạy nền***

touch /etc/systemd/system/cloudflared-tunnel.service

vim /etc/systemd/system/cloudflared-tunnel.service

*Nội dung file cloudflared-tunnel.service:*

[Unit]

Description=Cloudflare Tunnel

After=network.target

[Service]

Type=simple

ExecStart=/usr/local/bin/cloudflared tunnel --config /tung/cloudflare-tunnel-config.yml run

Restart=on-failure

User=phan

Environment=HOME=/home/phan

[Install]

WantedBy=multi-user.target

***Chạy lệnh để tiến hành thực thi “cloudflared-tunnel.service”, sau khi lệnh chạy có thể tắt terminal, khởi động máy, hoặc ngay khi tiến trình lỗi, nó sẽ tự khởi động lại “cloudflared-tunnel.service”:***

sudo systemctl daemon-reexec

sudo systemctl daemon-reload

sudo systemctl enable cloudflared-tunnel.service

sudo systemctl start cloudflared-tunnel.service

Để xem trạng thái của “***cloudflared-tunnel.service***”:

sudo systemctl status cloudflared-tunnel.service

***Chạy lại sử dụng lệnh:***

sudo systemctl daemon-reexec

sudo systemctl restart cloudflared-tunnel

**Cài đặt Jenkins trong kubernetes**

**Jenkins được cài với Helm chart.**

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

**Kiểm tra:**

helm version

***Chuẩn bị trước khi cài Jenkins vào kubernetes***

**1. Tạo namespaces cho jenkins**

kubectl create namespace jenkins

Xem secret đang tồn tại:

Kubectl get secret -n jenkins

**2. Tạo secret để tạo kubeconfig cho kubect trong pod jenkins thông qua secret:** *(secret đọc file, cung cấp cho tạo aws/credentials trong pod jenkins trên kubernetes, cho phép sử dụng aws thông qua thông tin đăng nhập lưu ở aws/ credentials cho jenkins. Trong lệnh dùng cần dùng “$HOME” thay vì “~” bới vì “~” là shell nên không được hiểu trong câu lệnh kubectl, còn “$HOME” là biến môi trường nên sẽ luôn đúng cho các trường hợp).*

kubectl create secret generic aws-credentials --from-file=credentials=$HOME/.aws/credentials -n jenkins

**3. Chuẩn bị image cho jenkins bằng docker, lưu trên docker hub.**

Login docker trước khi đẩy image lên docker hub

docker login (mở của url trong cửa sổ login, tiêp theo dán mã được cấp từ cửa số login vào trang login trên trình duyệt)

docker build -t vothinhuydt/jenkins:txu .

docker push vothinhuydt/jenkins:txu

**4. Tạo thư mục để mout vào jenkins lưu trữ dữ liệu lâu dài khi pod jenkins bị xóa tạo lại…**

sudo mkdir -p /mnt/data/jenkins

sudo chown -R 1000:1000 /mnt/data/jenkins

Reboot máy để có hiệu lực: reboot

ls -ldn /mnt/data/jenkins

ls -ld /mnt/data/jenkins

**5. Cài đặt Jenkins với helm**

helm install jenkins ./jenkins-chart -n jenkins

**\* Gỡ cài đặt Jenkins**

helm uninstall jenkins -n jenkins

**6. Sau khi cài đặt jenkins bằng helm, chạy jenkins thiết lập ban đầu cho jenkins**

a. jenkins hiện tại được cài đặt và chạy trong kubernetes và được ingress vào theo url: jenkins.txuapp.com. Do đó từ node control-plane, cài đặt cloudflare và tunnel url: jenkins.txuapp.com đến NodePort của Ingress Controller ở port(xxxxx:8080). Các bước từ tạo tunnel, route dns, và run tunnel như sau:

- Tạo tunnel cho jenkins: (sẽ tạo ra một id và file id.json của tunnel, dùng thông tin này để tạo file chạy tunnel route dns)

cloudflared tunnel create jenkins-tunnel

**Tạo một route dns cho gán với tunnel vừa tạo**

cloudflared tunnel route dns jenkins-tunnel jenkins.txuapp.com

**Tạo file “cloudflare-jenkins-tunnel.yml” config thông tin để chạy tunnel, nội dung:**

tunnel: 4a06d22d-240f-4e63-b772-58502933d65f

credentials-file: ~/.cloudflared/4a06d22d-240f-4e63-b772-58502933d65f.json

ingress:

- hostname: jenkins.txuapp.com

service: http://localhost:31179

- service: http\_status:404

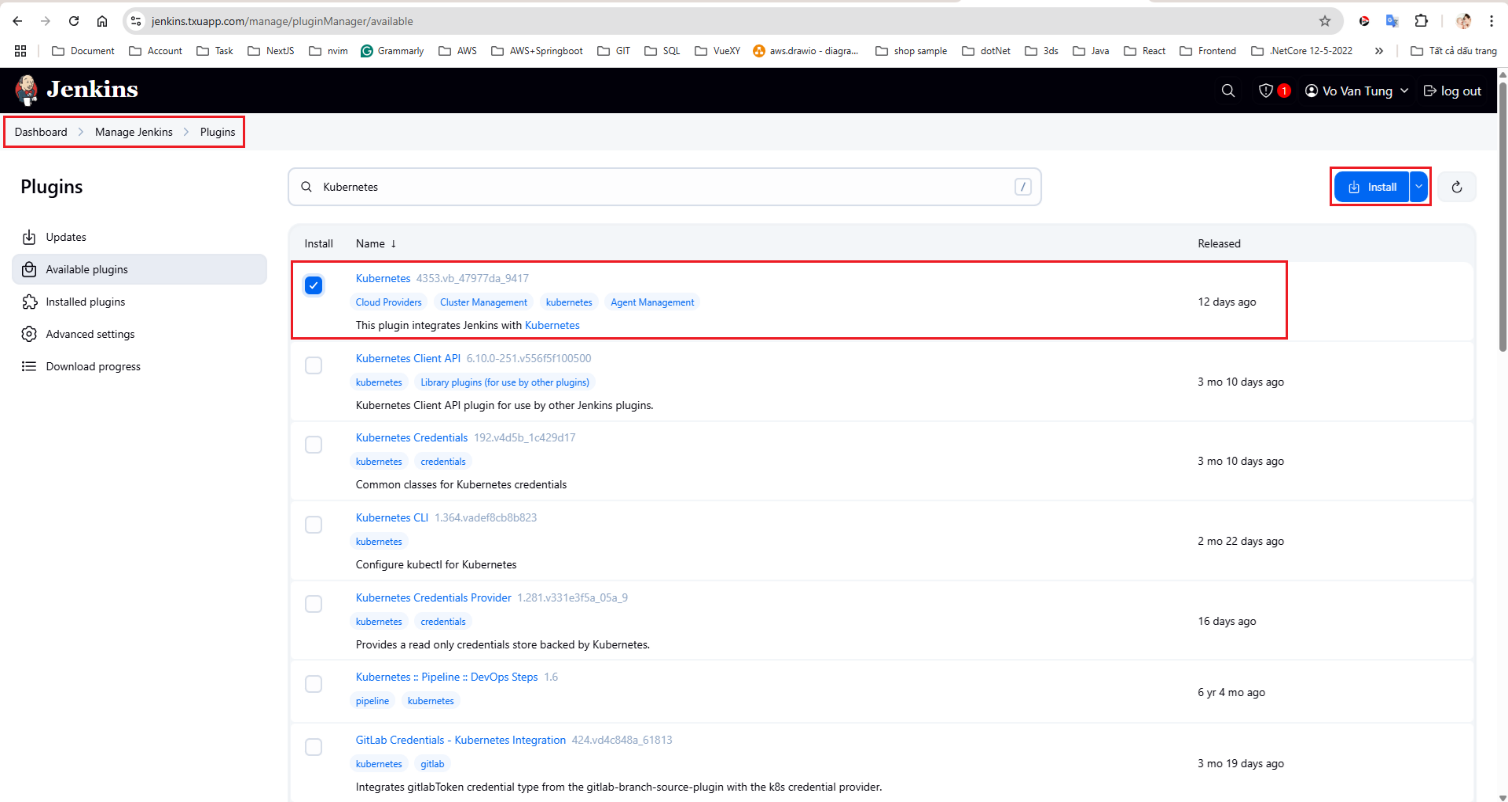
**Chạy tunnel với file config:**

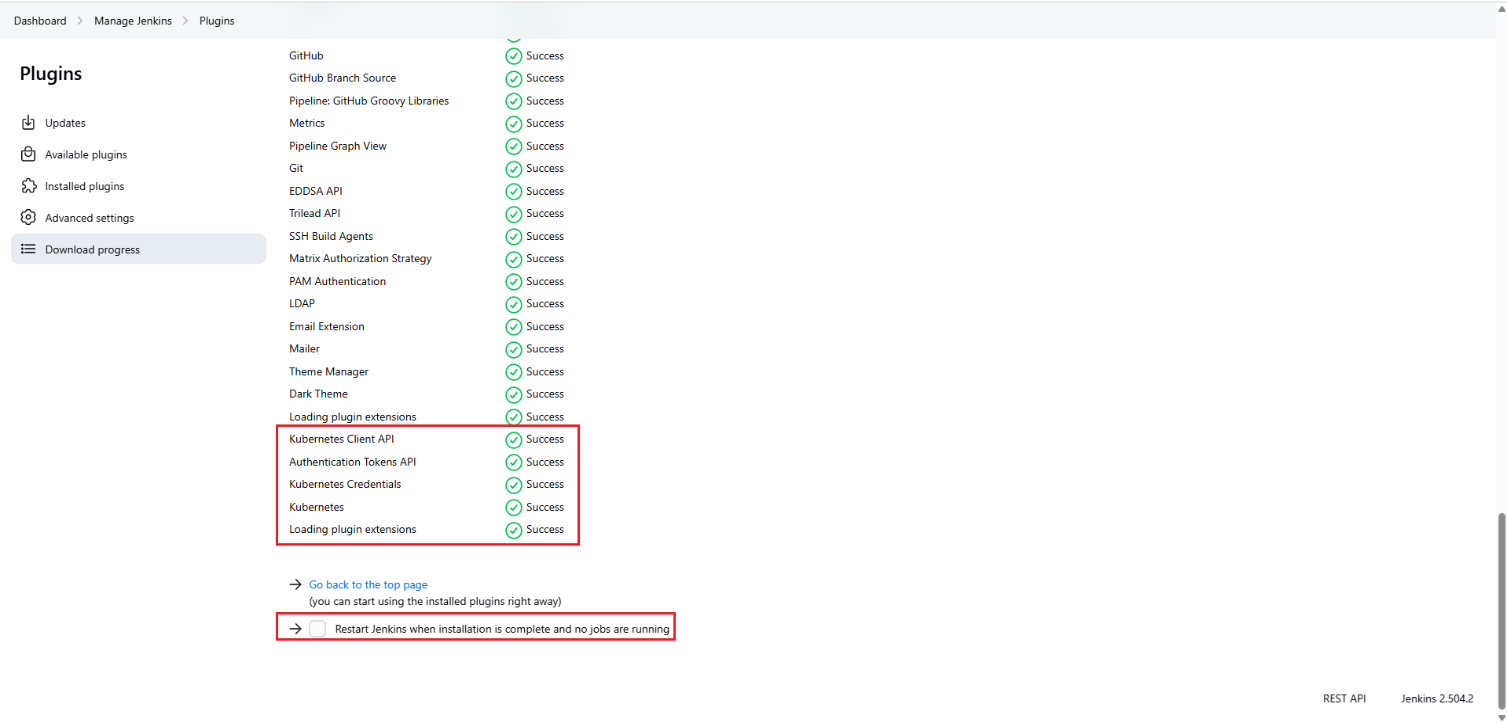
cloudflared tunnel –config cloudflare-jenkins-tunnel.yml run

b. Lấy mật khẩu để đăng nhập và thiết lập ban đầu cho jenkins:

kubectl exec -it -n jenkins <jenkins-pod-name> -- cat /var/jenkins\_home/secrets/initialAdminPassword

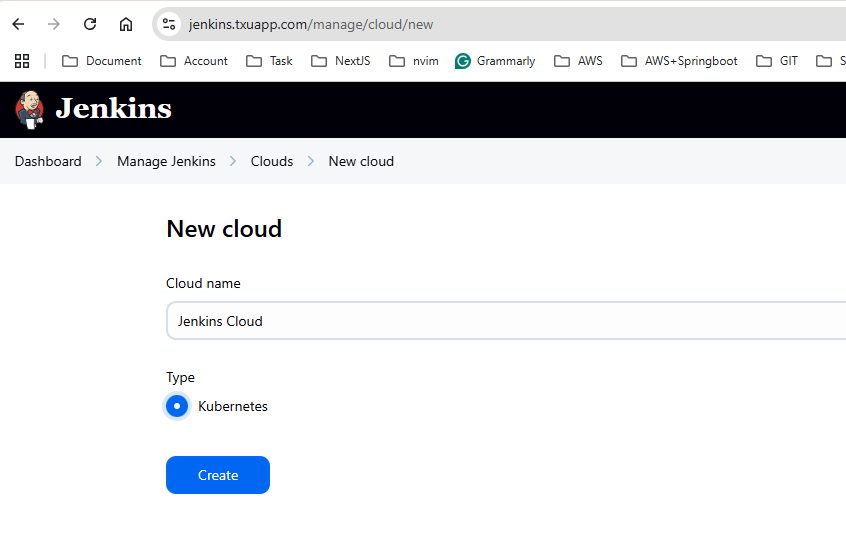
c. Tiếp theo cài đặt plusgin kubernetes cho jenkins trên UI của jenkins (việc cài đặt plusgin kubernetes cho phép jenkins chạy các jenkins agent và giao tiếp với jenkins controller thông qua dns nội bộ của jenkins “http://jenkins.jenkins.svc.cluster.local.:8080” ):



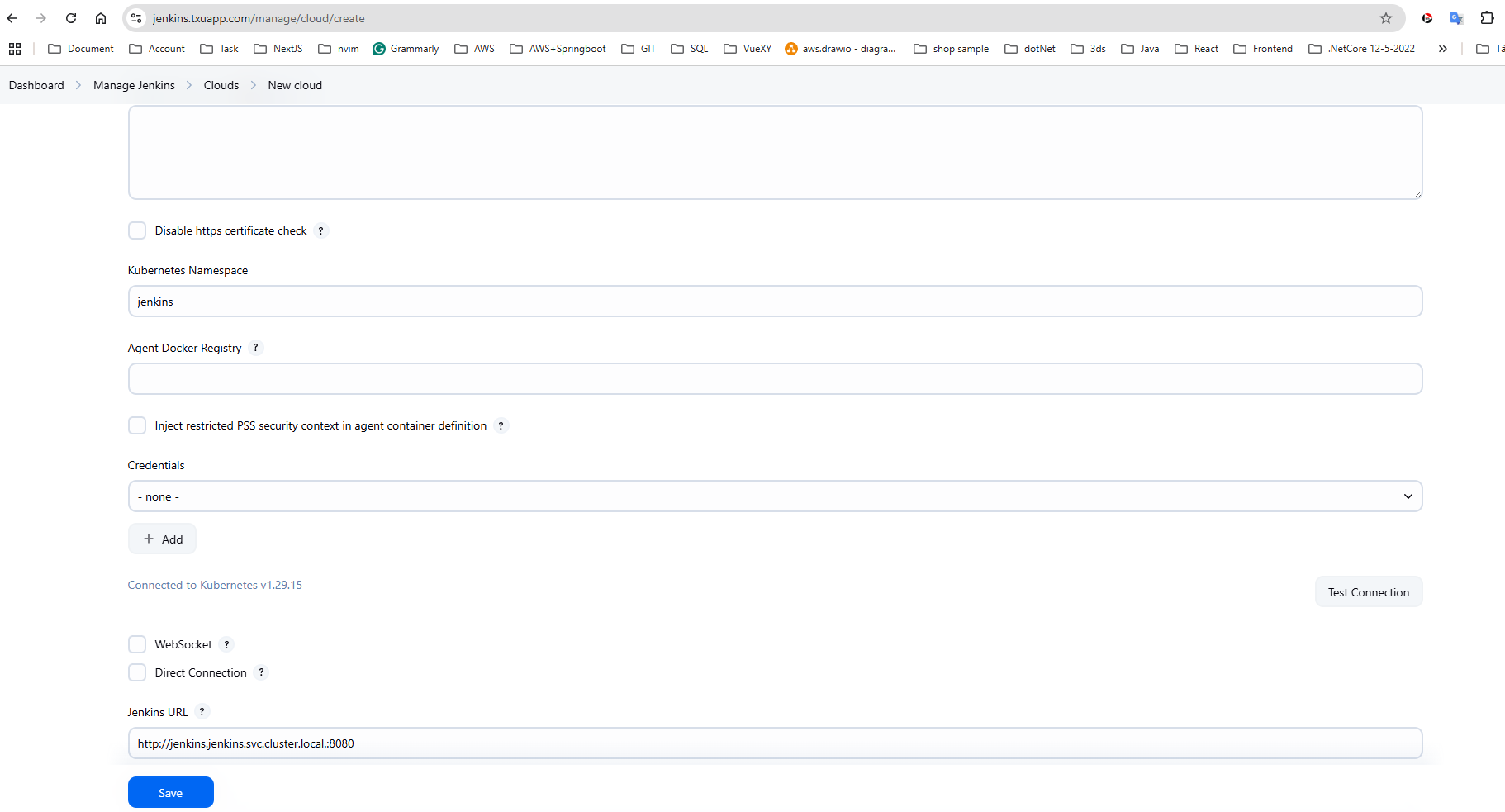


d. Cấu hình jenkins cloud kiểu kubernetes để cho phép các jenkins agent có thể giao tiếp jenkins controller. Do các jenkins agent chạy trong cluster cùng với jenkins controller nên cần cấu hình “URL Jenkins: <http://jenkins.jenkins.svc.cluster.local.:8080>”. Lưu ý: kết thúc url có đặt dấu “.” Để báo cho dns trong kubernetes biết đây là url đầy đủ mà không thêm phần searches, gây sai url và dns không phân giải chính xác. Mặt khác nếu không dùng url đầy đủ như trên thì cũng xảy ra tình trạng searches ở các container khác nhau sẽ không có những hậu tố “ví dụ: cluster.local” theo yêu cầu. Ngoài ra vì các jenkins agent và controller đều giao tiếp nội bộ nên cách hợp lý nhất là dùng dns nội bộ thay vì dùng dns public “jenkins.txuapp.com” vì khi đó yêu cầu phải mở thêm port cho jenkins ra bên ngoài, các agent phải truy cập từ bên ngoại thay vì truy cập ngay bên trong, điều đó không hợp lý và cần nhiều cấu hình hơn.

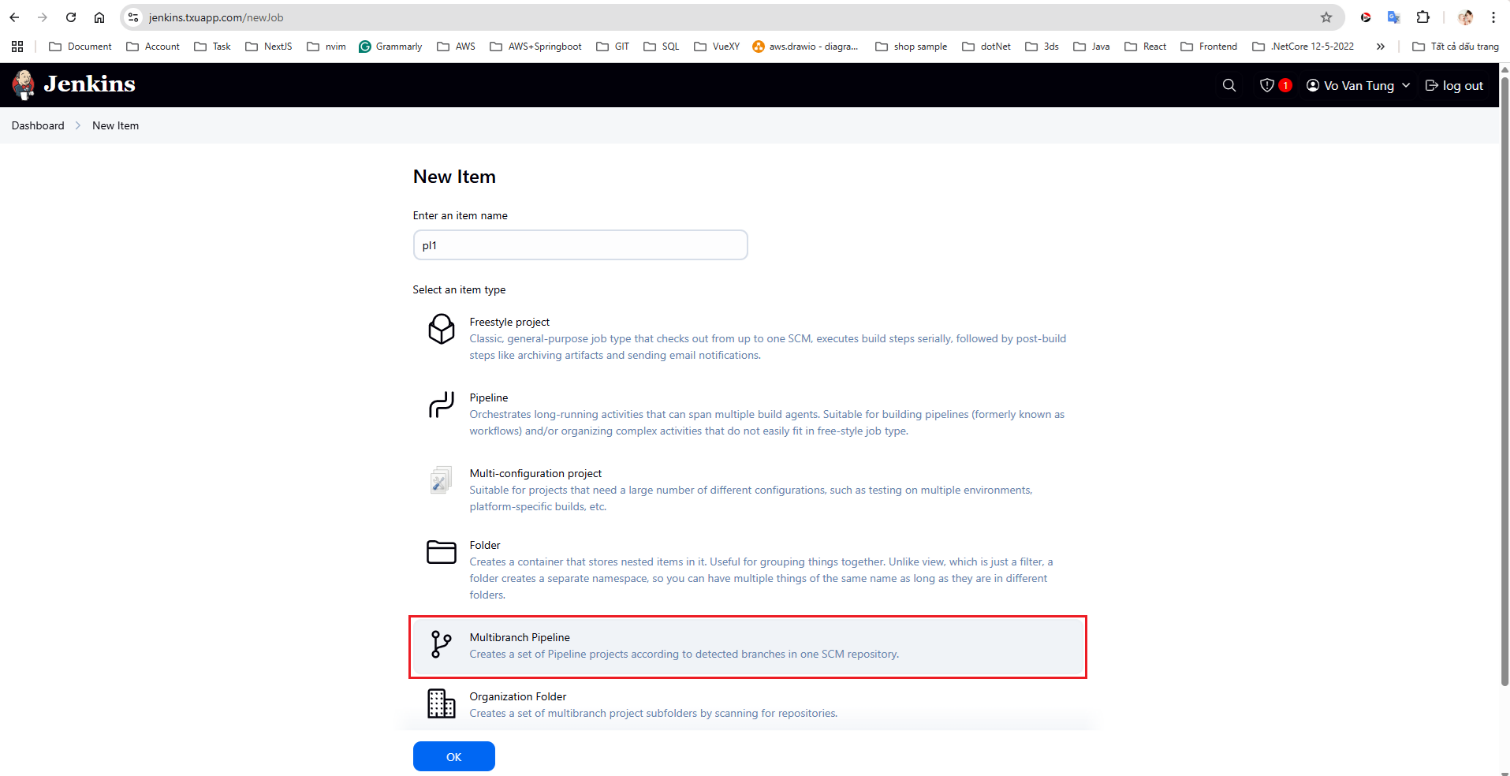
Để cấu hình jenkins cloud kiểu kubernetes thì trước hết cần tạo một cloud kiểu kubernetes:



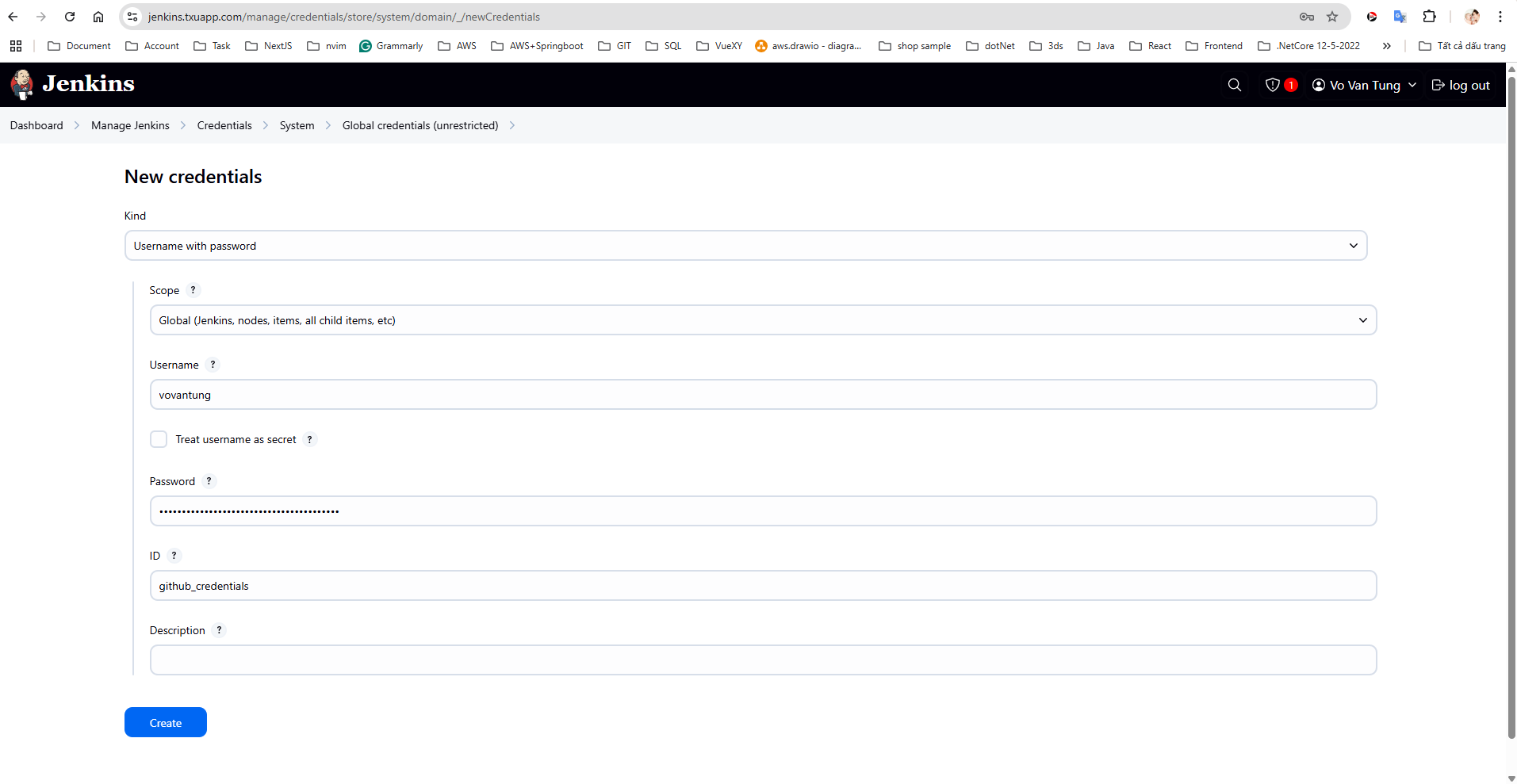
Chú ý: hai mục “Kubernetes Namespaces” và “Jenkins URL”



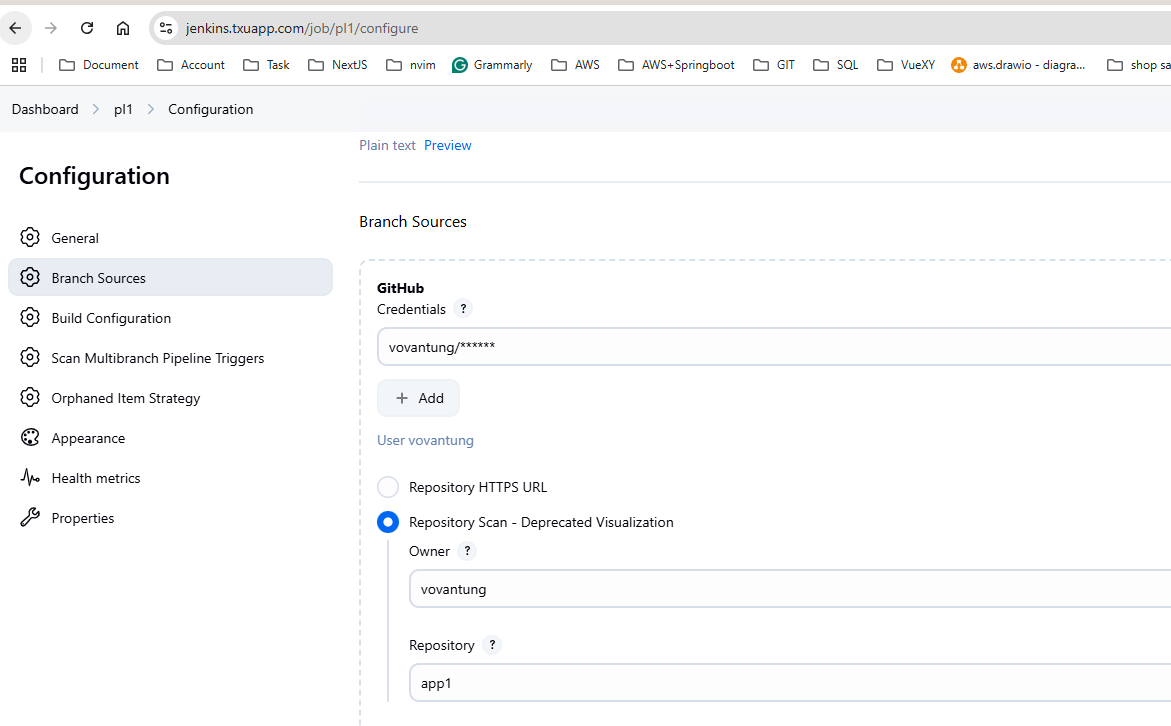
e. Tạo và chạy một job kiểu “multi branch”



Để cấu hình job kiểu “multibranch pipeline”, cần tạo **credentials** để đăng nhập github, thông tin cần có là username, passwork (token generate từ github)



Cấu hình job

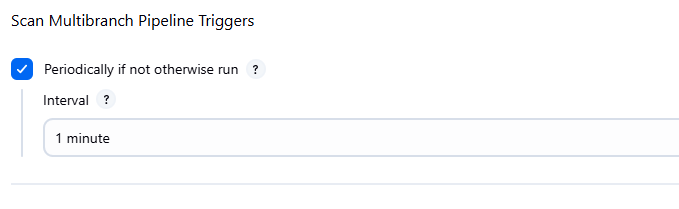
Mục “Behaviours” add thêm Filter by name (with wildcards), nhập Include: “main” nếu muốn chỉ trigger job khi push code trên nhánh “main”



Mục “Build Configuration” chọn “Script Path” là đường dẫn Jenkinsfile trong thư mục gốc của code project

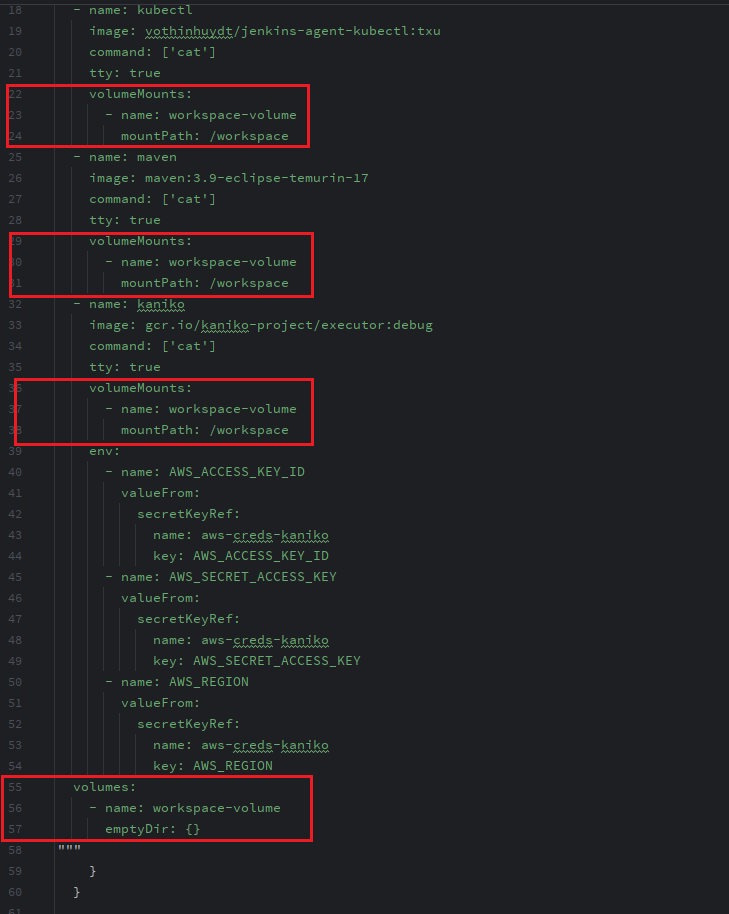


Mục “Scan Multibranch Pipeline Triggers”:



Khi đã tạo job kiểu “multibranch pipeline” như trên, và khi có sự kiện push code lên branch được chỉ định của repo (ví dụ push code lên branch “main” của repo “https://github.com/vovantung/app1.git”). Job được trigger sẽ chạy pipeline được định nghĩa trong Jenkinsfile. Khi cloud kubernetes được cài đặt trong jenkins thì các jenkins agent sẽ thực hiện các stage và giao tiếp với jenkins controller qua “Jenkins URL” mà ta đã chỉ định trong phần cấu hình cloud của jenkins (phần trên đã chỉ định giao tiếp qua dns nội bộ “<http://jenkins.jenkins.svc.cluster.local.:8080>”). Tuy nhiên một số jenkins agent ngoài giao tiếp với jenkins controller qua dns nội bộ đã được chỉ định (full url) thì chúng cũng cần thực hiện với giao tiếp với bên ngoài internet (chẳng hạn kaniko cần giao tiếp với docker.io để pull image “eclipse-temurin:17-jdk-alpine” cho việc build image từ dockerfile). Do đó với container kaniko cần đặt “ndots:1” để tránh việc dns nội bộ thực hiện gắn thêm phần searches (chẳng hạn “cluster.local”) gây ra không thể phân giải dns này.

Trong mô hình jenkins agent thực hiện nhiệm vụ được giao từ jenkins controller và các jenkins agent giao tiếp trở lại với jenkins controller thông qua dns nội bộ “<http://jenkins.jenkins.svc.cluster.local.:8080>” thì nhiệm vụ chủ yếu được thực hiện bởi các jenkins agent (là các container riêng biệt và có thể chia sẻ dữ liệu chung).



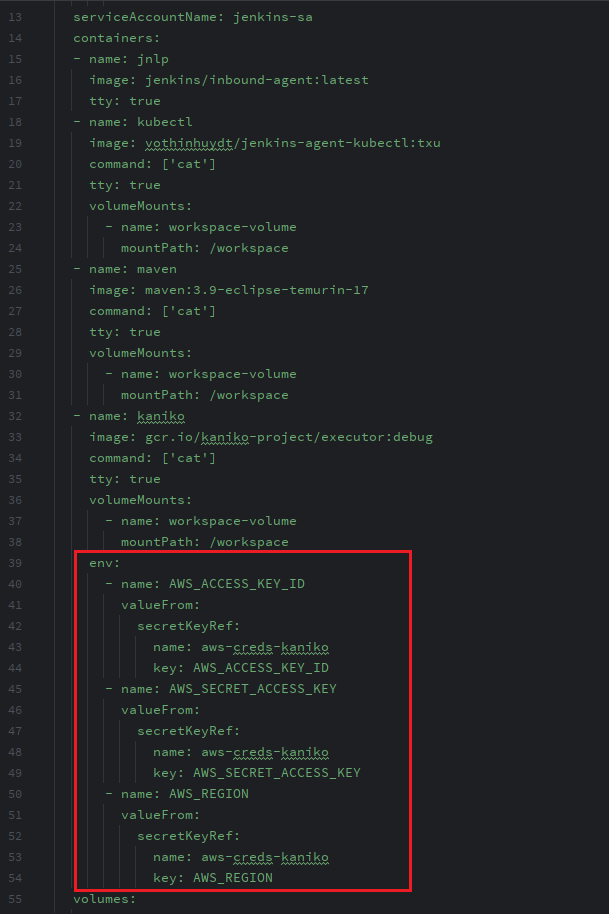
Khi đó jenkins controller chỉ đóng vai trò trung tâm giao nhiệm vụ (nằm trong các stage, chẳng hạn clone code, build code với maven, build code thành iamge với kanino và push lên các repo như ecr, docker hub;…) cho các jenkins agent, các jenkins agent thực hiện và trả lời lại jenkins controller.

*Chi tiết cụ thể các agent được định nghĩa và thực hiện nhiệm vụ cụ thể (xem thêm trong Jenkinsfile)*

Ví dụ, trong jenkins agent (kaniko) thực hiện nhiệm vụ nhận dữ liệu chung được chia sẻ từ jenkins agent (maven), nhận dockerfile và file app2.jar đã build bới jenkins agent (maven) ở stage trước đó, và thực hiện trước pull image “eclipse-temurin:17-jdk-alpine” từ internete, sau đó build thành image chứa ứng dụng app1.jar và push image mới build lên ecr. Kaniko được thiết kế có khả năng build image mà không cần docker meaon, và push image lên ecr hoặc docker hub… Quá trình push image lên erc chẳng hạn cần phải có quyền push image, do đó cần gắn các biến môi trường (thông qua secret trong kubernetes) chứa thông tin xác thực với aws thực hiện quyền push image lên ecr.

kubectl create secret generic aws-creds-kaniko --from-literal=AWS\_ACCESS\_KEY\_ID=aws-access-key-id --from-literal=AWS\_SECRET\_ACCESS\_KEY=aws-secret-key --from-literal=AWS\_REGION=ap-southeast-1 -n jenkins

Việc thực hiện gắn secret cho kaniko được thực hiện bằng cách tạo một secret dạng secret key và gắn thông tin đó vào container kaniko ở bước định nghĩa kamoko trong jenkinsfile.



Tương tự kaniko, thì jenkins agent (kubectl) cũng cần thực hiện việc kết nối với kubernetes api server để có thể dùng command “kubectl”. Để thực hiện được việc này, cần thực hiện tạo một “ServiceAccount” tên “” có quyền “rbac.authorization.k8s.io/v1” tức có quyền lấy thông tin xác thực của kubernetes.

“apiVersion: v1

kind: ServiceAccount

metadata:

name: jenkins-sa

namespace: {{ .Release.Namespace }}

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: jenkins-sa-cluster-admin

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

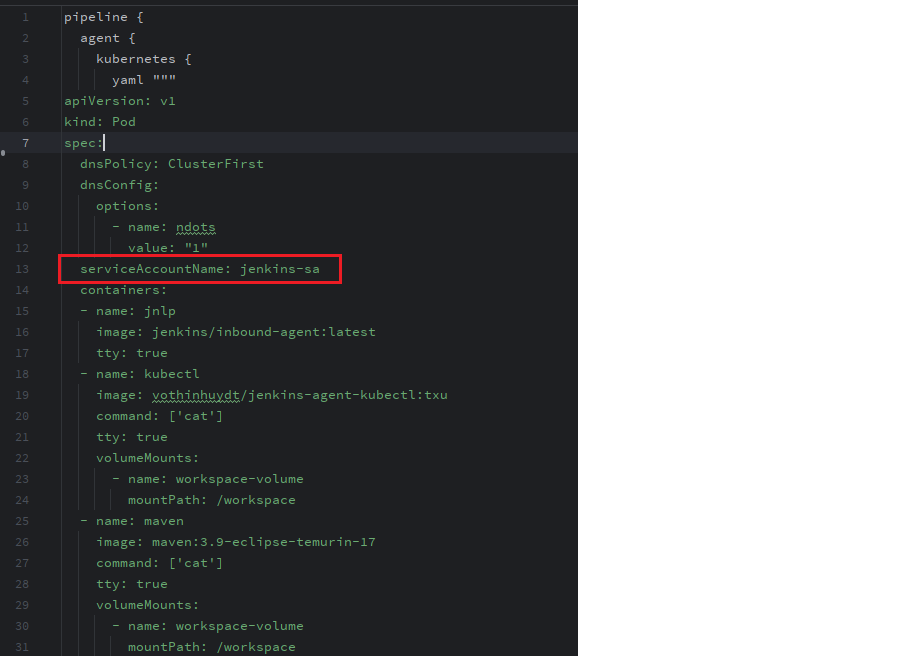
subjects:

- kind: ServiceAccount

name: jenkins-sa

namespace: {{ .Release.Namespace }}”

Khi gán “serviceAccountName: jenkins-sa” cho một pod thì các contriner của pod sẽ chứa các quyền xác thực với k8s, khi đó chỉ cần cài kubectl cli, kubectl cli sẽ thực hiện lấy quyền xác thực được cấp và cấu hình cho kubeconfig (nằm trong /var/jenkins\_home/.kube/config). Có nghĩa là , các container được gán quyền “rbac.authorization.k8s.io/v1” và cài đặt kubectl cli sẽ thực hiện các lệnh “kubectl”.



***Một số lệnh kiểm tra dns:***

nslookup jenkins.jenkins.svc (trả về ip của domain)

curl -v http:// nslookup jenkins.jenkins.svc.cluster.local.:8080 (nếu không dùng full url thì cần chú ý “ndots”, “ndots” lớn hơn số dấu “.” Trong domain thì dns sẽ tự gắn thêm phần searches (ví dụ: cluster.local), gây sai domain và dns phân giải không chính xác). Để xem lại “ndots” dùng lệnh: cat /etc/resolv.conf

kiểm tra chi tiết yaml của một service, pod, … với lệnh:

kubectl get svc jenkins -n jenkins -o yaml

kubectl get svc jenkins -n jenkins -o yaml | grep noi\_dung\_tim\_kiem [-A so\_dong\_can\_xem]

kubectl get deployment jenkins -n jenkins -o yaml | grep containerPort -A 2

kubectl get svc jenkins -n jenkins -o yaml | grep selector -A 2

Xem jenkins đăng lăng nghe ở những port nào:

kubectl exec -it jenkins-bb955c969-xdm7r -n jenkins -- ss -tuln

Chạy một pod test:

kubectl run txu-test --rm -i -t --image=ubuntu:22.04 -n jenkins -- /bin/bash

Get Endpoits:

kubectl get endpoints -n jenkins jenkins -o yaml

Get Config map

kubectl -n kube-system get configmap coredns -o yaml

Kiểm tra cấu hình dns:

cat /var/lib/kubelet/config.yaml

cat /var/lib/kubelet/config.yaml | grep clusterDNS

Restart kubelet:

sudo systemctl restart kubelet

**Cài đặt và kiểm tra docker:**

***Kiểm tra người dùng trong nhóm “docker”***

getent group docker

***Nếu chưa có nhóm “docker” có thể thêm nhóm:***

sudo groupadd docker

***Thêm người dùng võ vào nhóm “docker”***

sudo usermod -aG docker vo

***Hoặc thêm người dùng hiện tại vào nhóm “docker”***

sudo usermod -aG docker $USER

***Sau khi thêm user vào nhóm “docker” có thể chạy lệnh sau và kiểm tra lại nhóm “docker”***

newgrp docker

**1. Kiểm tra CoreDNS có hoạt động đúng không**

kubectl get pods -n kube-system -l k8s-app=kube-dns -o wide

Bạn nên thấy 2 pod coredns-\* đang Running.

**2. Kiểm tra logs của CoreDNS**

kubectl logs -n kube-system -l k8s-app=kube-dns

Tìm xem có dòng lỗi nào liên quan jenkins.jenkins.svc.cluster.local hoặc các truy vấn bị từ chối.

**3. Kiểm tra lại resolv.conf của Pod curl-test**

kubectl exec -n jenkins curl-test -- cat /etc/resolv.conf

Output chuẩn nên là:

search jenkins.svc.cluster.local svc.cluster.local cluster.local

nameserver 10.96.0.10

options ndots:5

**Sau đó kiểm tra lại DNS:**

nslookup jenkins.jenkins.svc.cluster.local

curl -v <http://jenkins.jenkins.svc.cluster.local:8080>

**Tạo secret gắn cho pod kaniko sử dụng cho đăng nhập aws và push code lên ecr:**

kubectl create secret generic aws-creds-kaniko --from-literal=AWS\_ACCESS\_KEY\_ID=aws-access-key-id --from-literal=AWS\_SECRET\_ACCESS\_KEY=aws-secret-key --from-literal=AWS\_REGION=ap-southeast-1 -n jenkins

**Tạo secret cho phép jenkins agent (kubectl) chạy deployment kéo image từ ecr về tạo container trong kubernetes:**

kubectl create secret docker-registry ecr-secret --docker-server=211125364313.dkr.ecr.ap-southeast-1.amazonaws.com --docker-username=AWS --docker-password=$(aws ecr get-login-password --region ap-southeast-1) [--docker-email=vovantungdt123@gmail.com](mailto:--docker-email=vovantungdt123@gmail.com) -n jenkins -n backend

kubectl create secret generic aws-credentials --from-file=credentials=$HOME/.aws/credentials -n jenkins

*(secret đọc file, cung cấp cho tạo aws/credentials trong pod jenkins trên kubernetes. Trong lệnh dùng cần dùng “$HOME” thay vì “~” bới vì “~” là shell nên không được hiểu trong câu lệnh kubectl, còn “$HOME” là biến môi trường nên sẽ luôn đúng cho các trường hợp)*

(Lưu ý: trước khi tạo, chạy aws config với secret được lấy từ aws với quyền phù hợp hoặc secret từ root account)

**Cài đặt, config và kiểm tra cert-manager trong k8s**

**Bước 1: Cài cert-manager**

kubectl apply -f <https://github.com/cert-manager/cert-manager/releases/download/v1.18.1/cert-manager.yaml>

Xem pod certificate

kubectl get pods -n cert-manager

Xem chi tiết certificate đã cấp:

kubectl describe certificate -n nexus nexus-tls

**Bước 2: Tạo ClusterIssuer**

*tạo file cluster-issuer.yaml:*

apiVersion: cert-manager.io/v1

kind: ClusterIssuer

metadata:

name: letsencrypt-http01

spec:

acme:

email: vovantungdt123@gmail.com

server: https://acme-v02.api.letsencrypt.org./directory

privateKeySecretRef:

name: letsencrypt-http01-key

solvers:

- http01:

ingress:

class: nginx

kubectl apply -f cluster-issuer.yaml

**Bước 3: Tạo ingress với cert-manager annotation**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: nexus-ingress

namespace: nexus

annotations:

cert-manager.io/cluster-issuer: letsencrypt-http01

nginx.ingress.kubernetes.io/force-ssl-redirect: "true"

spec:

ingressClassName: nginx

tls:

- hosts:

- nexus.txuapp.com

secretName: nexus-tls

rules:

- host: nexus.txuapp.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: nexus

port:

number: 8081

**Bước 5: Kết quả mong muốn**

Khi bạn truy cập https://nexus.txuapp.com:

* Request sẽ qua modem NAT → Ubuntu → ingress controller → cert-manager sẽ xin cert từ Let’s Encrypt và lưu trong Secret nexus-tls.
* Ingress controller tự dùng cert đó → **HTTPS chuẩn production**

**Kiểm tra Issuer / ClusterIssuer**

kubectl get clusterissuer

Phải thấy:

NAME READY AGE

letsencrypt-http01 True 10m

**Kiểm tra Ingress**

kubectl describe ingress nexus-ingress -n nexus

**Kiểm tra Secret nexus-tls**

kubectl get secret nexus-tls -n nexus -o yaml

→ Phải thấy:

type: kubernetes.io/tls

data:

tls.crt: <nội dung base64>

tls.key: <nội dung base64>

✅ Nếu không có hoặc trống → là do cert-manager chưa xin được cert.

**Xem logs cert-manager**

kubectl logs -n cert-manager deploy/cert-manager -f

**Cách vào pod cert-manager kiểm tra:**

kubectl run curl-test --rm -i --tty --image=alpine --restart=Never – sh

**Bên trong shell của pod, cài curl:**

apk add --no-cache curl

**Sau đó test truy cập ACME server:**

curl -v <https://acme-v02.api.letsencrypt.org/directory>

Xem DNS config trong pod:

cat /etc/resolv.conf

Kiểm tra log CoreDNS:

kubectl logs -n kube-system -l k8s-app=kube-dns

Kiểm tra trang thái certificate:

kubectl get certificate -A -o wide

kubectl describe certificate -A

kubectl get certificate -A

Kiểm tra cert-manager log

kubectl logs -n cert-manager -l app.kubernetes.io/name=cert-manager

**Xóa Certificate để cert-manager tạo lại từ đầu** — việc này an toàn:

kubectl delete certificate nexus-tls -n nexus

Rồi apply lại ingress:

kubectl apply -f ingress-nexus.yaml

Hoặc nếu bạn thích, xóa luôn cái secret cũ (nếu có):

kubectl delete secret nexus-tls -n nexus

kubectl run -n cert-manager curl-client --rm -it --image=alpine -- sh

cat /etc/resolv.conf

**Cách giảm ndots cho pod cert-manager:**

**Trong Kubernetes**, khi pod chạy, nó sẽ inherit ndots từ /etc/resolv.conf của node hoặc từ config CoreDNS → nhưng ta có thể **override cho từng pod** qua:

👉 **dnsConfig.options** trong Deployment

**Cụ thể — ví dụ chỉnh Deployment của cert-manager:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: cert-manager

namespace: cert-manager

spec:

template:

spec:

dnsPolicy: ClusterFirst

dnsConfig:

options:

- name: ndots

value: "1"

**Với cert-manager — bạn chỉ cần làm:**

kubectl edit deployment -n cert-manager cert-manager

→ Thêm đoạn:

spec:

template:

spec:

dnsPolicy: ClusterFirst

dnsConfig:

options:

- name: ndots

value: "1"

→ Save lại → Kubernetes sẽ tự rollout lại pod cert-manager mới.

**Hoặc:**

**Patch Deployment/cert-manager để set dnsConfig.options.ndots: 1**

**Lệnh cụ thể:**

kubectl patch deployment cert-manager -n cert-manager --type='json' -p='[{"op": "add", "path": "/spec/template/spec/dnsConfig", "value": {"options": [{"name": "ndots", "value": "1"}]}}]'

**Sau đó restart lại cert-manager:**

kubectl rollout restart deployment cert-manager -n cert-manager

Vì DNS options như ndots được set từ kubelet khi pod được deploy (thường default từ kubelet hoặc từ deployment yaml).

Bạn có thể chạy pod tạm:

kubectl run -n cert-manager dns-test --rm -it --image=busybox:1.28 -- sh

Rồi bên trong:

cat /etc/resolv.conf

→ Giá trị này sẽ giống với pod cert-manager, vì cùng 1 namespace và kubelet config.

**Tạo forward từ windows host đến máy ảo:**

netsh interface portproxy add v4tov4 listenport=80 listenaddress=0.0.0.0 connectport=31338 connectaddress=192.168.98.141

**Xóa một forward:**

netsh interface portproxy delete v4tov4 listenport=443 listenaddress=0.0.0.0

**Xem các forward:**

netsh interface portproxy show all

**Cài đặt Nexus trong k8s:**

**Tạo và cấu hình thư nục cho nexus mount (trên node mà pod nexus được tạo):**

sudo mkdir -p /mnt/data/nexus

sudo chown -R 200:200 /mnt/data/nexus

**Lấy mật khẩu mặc định của nexus:**

kubectl exec -it nexus-7fd84bfdfb-z9t47 -n nexus -- cat /nexus-data/admin.password

Sau khi nhập mật khẩu mặc định, nexus yêu cầu đặt lại mật khẩu mởi cho tài khoản admin. Tài khoản này được dùng để push thư viện lên nexus, cách cấu hình:

*Thêm code sau vào file settings.xml trong thư mục* ***.m2****:*

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">

<servers>

<server>

<id>nexus</id>

<username>admin</username>

<password>bungteDT@123</password>

</server>

</servers>

</settings>

**Cài đặt Grafana,**

**Bước 1: Cài đặt Helm**

Nếu bạn chưa cài Helm trên máy control plane:

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

helm version

**Bước 2: Thêm repo Prometheus cộng đồng**

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

**Bước 3: Tạo namespace cho Prometheus**

kubectl create namespace monitoring

**Cài Local Path Provisioner (khuyến khích)**

GitHub: <https://github.com/rancher/local-path-provisioner>

kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml

Sau đó set storageClassName trong values.yaml thành local-path:

**Xóa pvc cũ để upgrade prometheus-stack**

kubectl delete pvc -n monitoring alertmanager-prometheus-stack-kube-prom-alertmanager-db-alertmanager-prometheus-stack-kube-prom-alertmanager-0

kubectl delete pvc -n monitoring prometheus-prometheus-stack-kube-prom-prometheus-db-prometheus-prometheus-stack-kube-prom-prometheus-0

**Bước 4: Cài Prometheus + các thành phần đi kèm**

Ta dùng chart kube-prometheus-stack — chart phổ biến nhất hiện nay, gồm:

Prometheus

Grafana  
Alertmanager  
Node Exporter  
Kube State Metrics

**Lệnh cài (với giá trị mặc định):**

helm install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring

**Hoặc cài với values file:**

helm install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring -f monitoring-values.yaml

**Hoặc upgrade nếu đã cài trước đó:**

helm upgrade --install prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring -f monitoring-values.yaml

**Bước 5: Kiểm tra**

kubectl get pods -n monitoring

kubectl describe pod alertmanager-prometheus-stack-kube-prom-alertmanager-0 -n monitoring

**Bước 6: Mở Grafana để xem Dashboard**

**User/Password mặc định:**

Username: admin

Password: prom-operator

(Nếu khác thì bạn xem password từ secret)

kubectl get secret prometheus-stack-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 -d