**CÔNG TY CỔ PHẦN ATOMI DIGITAL**



**Tài liệu cài đặt và cấu hình Cluster Kubernetes**

Phiên bản: V1.0

Last Updated: 03/07/2024

**Mục lục**

[**I. Tổng quan 4**](#_oe2bqe5pami1)

[**II. Chuẩn bị 5**](#_x10c363o8til)

[1. Cài đặt thư viện 5](#_dtizwqpmmiul)

[2. Mount thư mục data containerd ra ổ đĩa ngoài 6](#_wegcavkoaxe3)

[3. Cài đặt passwordless ssh từ bootstrap đến các node 10](#_v4s42joejc48)

[**II. K8s 12**](#_pnw0a48oba8c)

[1. Cài đặt 12](#_t8ma1a64cpcd)

[2. Cài đặt metrics-server 15](#_9xs2sv4j2jhe)

[**III. Harbor registry 19**](#_so7s1rln69p2)

[1. Cài đặt 19](#_wxtz3echhyg2)

[2. Cấu hình docker và containerd để pull image 22](#_ibxpkd2q9pc5)

[**IV. Longhorn persistent storage 26**](#_b8sgkl8nucw)

[**V. Logging system 33**](#_m5xe3a4zvrcm)

[**VI. Istio service mesh 34**](#_x2g5vrl20i15)

[**VII. Monitoring system 40**](#_h7f9uf97puik)

[1. Cài đặt 40](#_7qkc1jycglb8)

[2. Cài đặt Redis exporter 42](#_297ul4dsyb6k)

[3. Cài đặt OracleDB exporter 43](#_d9mfml8f7utn)

[4. Cài đặt RabbitMQ exporter 49](#_viy9a2x6axtb)

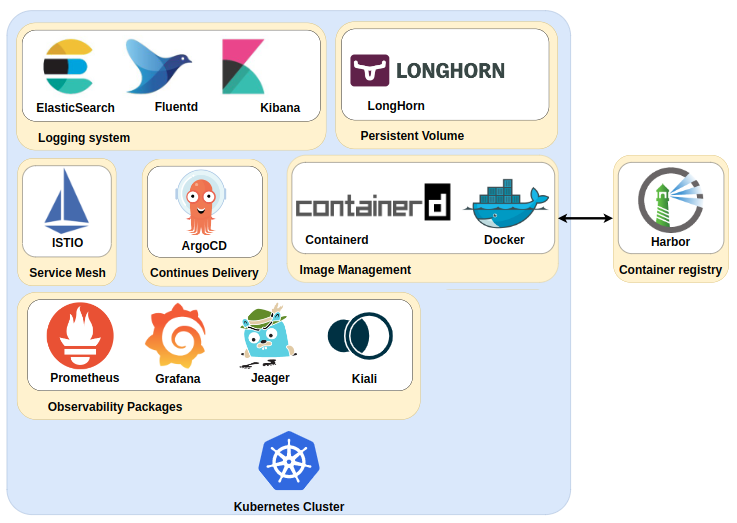
[5. Cài đặt Elasticsearch exporter 51](#_jahzstomggui)

[**VIII. ArgoCD 52**](#_lnncq8u7k8ia)

Repo manifest: <http://10.36.209.79/LienViet24H/manifest/new-k8s-manifests>

| Server Type | Server Name | OS | Kernel |
| --- | --- | --- | --- |
| Bootstrap Server | Bootstrap | Ubuntu: 20.04 | Any |
| Master Node | master1 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |
| Master Node | master2 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |
| Worker Node | worker1 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |
| Worker Node | worker2 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |
| Worker Node | worker3 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |
| Worker Node | worker4 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |
| Worker Node | worker5 | CentOS 8.5.2111 | 4.18.0-348.7.1.el8\_5.x86\_64 |

# Tổng quan



*Tổng quan các tools được cài đặt trên Kubernetes (K8S)*

- Cluster Kubernetes: cụm cluster Kubernetes gồm nhiều Node, mỗi node tương ứng với 1 machine. Cluster sẽ gồm ít nhất 1 Master Node (thường đặt bằng 1 trong môi trường Test và 3 trong môi trường Production) và gồm nhiều Worker Node (ít nhất là 2). Mỗi node tối thiểu 4 core CPU, 8GB RAM và 150GB đĩa cứng.

- Cài đặt thêm 1 số các hạ tầng hỗ trợ bên trong cụm K8S:

* Logging system: hệ thống logging là một trong những tính năng quan trọng để giám sát hoạt động của các container. Hệ thống logging giúp ta theo dõi các sự kiện trong container, tìm kiếm các lỗi và giải quyết các vấn đề.
* Persistent Volume (PV): là một đối tượng trong Kubernetes mà ta có thể sử dụng để giữ lại dữ liệu sau khi tất cả các container trong một pod bị xóa. PV cung cấp một bộ nhớ dài hạn để lưu trữ dữ liệu và giữ nó được kể cả khi pod đó bị xóa hoặc bị tắt.
* Image Management: là hạ tầng quản lý, chạy các Image thành các container, đối với version Kubernetes 1.24+ containerd được sử dụng cho mục đích này.
* Observability packages: đề cập đến việc thực hành giám sát và đo lường hành vi và hiệu suất của một cụm K8S và các thành phần của nó. Cung cấp các công cụ mạnh mẽ để theo dõi và phân tích hành vi cũng như hiệu suất của cụm Kubernetes.
* Continuous Delivery (CD): hạ tầng nhằm mục đích tự động hóa toàn bộ quy trình phân phối phần mềm, từ môi trường Dev đến môi trường Production. Mục tiêu của CD là giảm thời gian thực thi thủ công việc cung cấp phần mềm cho người dùng, trong khi vẫn duy trì mức độ chất lượng và độ tin cậy cao.

- Hạ tầng cài đặt ngoài cụm K8S:

* Container Registry: là một kho lưu trữ và phân phối Docker Image. Container Registry cung cấp một nơi tập trung để lưu trữ Image, cho phép các nhà phát triển quản lý, tạo phiên bản và phân phối các ứng dụng được chứa trong Image của họ. Hạ tầng Image Management sẽ được kết nối tới Container Registry để pull Image về, sau đó chạy Image này trên cụm Kubernetes.

# Chuẩn bị

## Cài đặt thư viện

- Trên node master1, tạo thư mục chứa thông tin cài đặt k8s để lưu trữ:

| sudo mkdir /data-k8s sudo chown -R $USER:$USER /data-k8s |
| --- |

- Thực hiện trên tất cả các node.

- Tạo script cài đặt: nano preinstall.sh

| #!/bin/bash echo -e '\033[1m\033[4mI. Install dependencies\033[0m' echo "===Step 1: Install packages===" sudo yum install -y yum-utils epel-release lvm2 nfs-utils cloud-utils-growpart iscsi-initiator-utils sudo yum update echo "===Step 2: Disable SELinux===" sudo setenforce 0 sudo sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux echo "===Step 3: Disable FirewallD===" sudo systemctl stop firewalld sudo systemctl disable firewalld echo "===Step 4: Enable IPv4 forward===" cat <<EOF | sudo tee -a /etc/sysctl.conf net.ipv4.ip\_forward = 1 vm.swappiness=10 EOF sudo sysctl -p echo "===Step 5: Load modules===" sudo modprobe overlay sudo modprobe br\_netfilter cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf overlay br\_netfilter EOF sudo sysctl --system echo -e '\033[1m\033[4mII. Install containerd\033[0m' sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo sudo yum install -y --allowerasing containerd.io sudo systemctl start containerd sudo systemctl enable containerd sudo systemctl status containerd |
| --- |

- Chạy:

| sudo chmod +x preinstall.sh sudo ./preinstall.sh |
| --- |

## Mount thư mục data containerd ra ổ đĩa ngoài

- Thực hiện trên tất cả các node.

- Xem thông tin filesystem ban đầu:

| [thaonp@Atomi-node1 ~]$ df -Th Filesystem Type Size Used Avail Use% Mounted on devtmpfs devtmpfs 32G 0 32G 0% /dev tmpfs tmpfs 32G 0 32G 0% /dev/shm tmpfs tmpfs 32G 25M 32G 1% /run tmpfs tmpfs 32G 0 32G 0% /sys/fs/cgroup /dev/mapper/cl-root xfs 146G 3.8G 143G 3% / /dev/sda1 xfs 1014M 356M 659M 36% /boot tmpfs tmpfs 6.3G 0 6.3G 0% /run/user/1002 |
| --- |

- Xem thông tin block device ban đầu:

| [thaonp@Atomi-node1 ~]$ lsblk NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT sda 8:0 0 150G 0 disk  ├─sda1 8:1 0 1G 0 part /boot ├─sda2 8:2 0 29G 0 part  │ ├─cl-root 253:0 0 146G 0 lvm / │ └─cl-swap 253:1 0 3G 0 lvm [SWAP] └─sda3 8:3 0 120G 0 part   └─cl-root 253:0 0 146G 0 lvm / sdb 8:16 0 150G 0 disk  sr0 11:0 1 1024M 0 rom |
| --- |

- Tạo partition mới: sudo fdisk /dev/sdb

n: Tạo phân vùng mới

p: chọn type là primary

enter: chọn partition number mặc định là 1

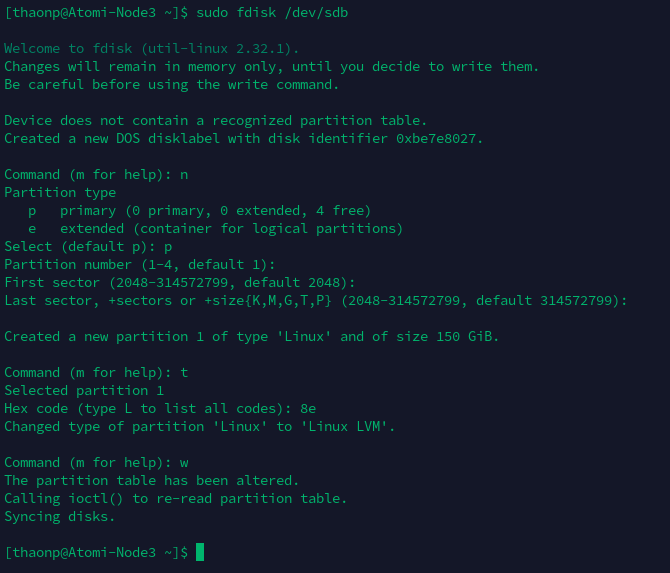
enter: chọn first sector mặc định là 2048

enter: chọn last sector mặc định

t: chọn phân vùng 1 vừa tạo

8e: thay đổi type thành Linux lvm

w: lưu thay đổi và ghi vào đĩa

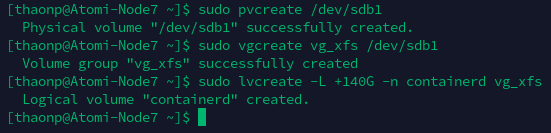


- Xem thông tin block device sau khi tạo phân vùng:

| [thaonp@Atomi-node1 ~]$ lsblk NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT sda 8:0 0 150G 0 disk  ├─sda1 8:1 0 1G 0 part /boot ├─sda2 8:2 0 29G 0 part  │ ├─cl-root 253:0 0 146G 0 lvm / │ └─cl-swap 253:1 0 3G 0 lvm [SWAP] └─sda3 8:3 0 120G 0 part   └─cl-root 253:0 0 146G 0 lvm / sdb 8:16 0 150G 0 disk  └─sdb1 8:17 0 150G 0 part  sr0 11:0 1 1024M 0 rom |
| --- |

- Tạo các thành phần cho LVM:

| sudo pvcreate /dev/sdb1 sudo vgcreate vg\_xfs /dev/sdb1 sudo lvcreate -L +140G -n containerd vg\_xfs |
| --- |



- Định dạng phân vùng LVM vừa tạo là xfs filesystem: sudo mkfs -t xfs /dev/vg\_xfs/containerd

- Test mount phân vùng (tạm thời, sẽ bị unmount khi reboot máy):

| sudo mkdir -p /var/lib/containerd sudo mount /dev/vg\_xfs/containerd /var/lib/containerd |
| --- |

- Lấy UUID của phân vùng:

| sudo blkid /dev/vg\_xfs/containerd  # output  /dev/vg\_xfs/containerd: UUID="01b444ce-e1f3-4b21-9795-d09a4684f1a9" BLOCK\_SIZE="512" TYPE="xfs" |
| --- |

- Mount vĩnh viễn:

| sudo nano /etc/fstab # Thêm dòng sau # <file system> <mount point> <type> <options> <dump> <pass> UUID=01b444ce-e1f3-4b21-9795-d09a4684f1a9 /var/lib/containerd xfs defaults 0 0 |
| --- |

- Pull thử 1 image: sudo ctr image pull docker.io/library/nginx:1.25.2

- Tăng size phân vùng nếu cần:

| # Kiem tra dung luong trong kha dung - VFree con ~20G [thaonp@Atomi-Node2 ~]$ sudo vgs vg\_xfs VG #PV #LV #SN Attr VSize VFree vg\_xfs 1 1 0 wz-n <150.00g <20.00g # Mo rong phan vung, vd them 10GB [thaonp@Atomi-Node2 ~]$ sudo lvextend -L +10G /dev/vg\_xfs/containerd -r  Size of logical volume vg\_xfs/containerd changed from 130.00 GiB (33280 extents) to 140.00 GiB (35840 extents). Logical volume vg\_xfs/containerd successfully resized. meta-data=/dev/mapper/vg\_xfs-containerd isize=512 agcount=4, agsize=8519680 blks = sectsz=512 attr=2, projid32bit=1 = crc=1 finobt=1, sparse=1, rmapbt=0 = reflink=1 data = bsize=4096 blocks=34078720, imaxpct=25 = sunit=0 swidth=0 blks naming =version 2 bsize=4096 ascii-ci=0, ftype=1 log =internal log bsize=4096 blocks=16640, version=2 = sectsz=512 sunit=0 blks, lazy-count=1 realtime =none extsz=4096 blocks=0, rtextents=0 data blocks changed from 34078720 to 36700160 |
| --- |

## Cài đặt passwordless ssh từ bootstrap đến các node

- Tạo script sau trên bootstrap node để cài đặt cặp public-private key, copy public key lên các node và tạo file config: nano setup-ssh.sh

| #!/bin/bash echo -e '\033[1m\033[4mI. Tao cap public/private key\033[0m' ssh-keygen -t rsa -b 4096 -f ~/.ssh/bootstrap\_id\_rsa -P "" -C "longnsp: bootstrap kubespray cum 10.163.233.20" servers=("10.163.233.18" "10.163.233.19" "10.163.233.20" "10.163.233.21" "10.163.233.22" "10.163.233.23" "10.163.233.24") username="username-here" password="password-here" echo -e '\033[1m\033[4mII. Copy SSH public key len cac server\033[0m' for server in "${servers[@]}" do  sshpass -p "$password" ssh-copy-id -i ~/.ssh/bootstrap\_id\_rsa "$username@$server" done echo -e '\033[1m\033[4mIII. Tao SSH config\033[0m' cat <<EOF | tee ~/.ssh/config Host master1  HostName 10.163.233.20  Host master2  HostName 10.163.233.18  Host master3  HostName 10.163.233.19   Host worker1  HostName 10.163.233.21  Host worker2  HostName 10.163.233.22  Host worker3  HostName 10.163.233.23  Host worker4  HostName 10.163.233.24   Host master1 master2 master3 worker1 worker2 worker3 worker4  User username  PreferredAuthentications publickey  IdentityFile /home/longnsp/.ssh/bootstrap\_id\_rsa EOF |
| --- |

- Cần sửa danh sách các servers tương ứng, cách nhau bằng dấu cách: “IP1” “IP2” “IP3”

- Cần sửa username, password tương ứng.

- Chạy:

| sudo chmod +x setup-ssh.sh ./setup-ssh.sh |
| --- |

# II. K8s

## Cài đặt

- Thực hiện trên server bootstrap.

- Clone Kubespray repo: git clone https://github.com/kubernetes-sigs/kubespray.git --branch release-2.24

- Cấu hình cho cluster như sau:

cd kubespray

cp -rfp inventory/sample inventory/mycluster

nano inventory/mycluster/inventory.ini

###

[all]

master1 ansible\_host=10.36.209.90 ip=10.36.209.90

master2 ansible\_host=10.36.209.89 ip=10.36.209.89

worker1 ansible\_host=10.36.209.91 ip=10.36.209.91

worker2 ansible\_host=10.36.209.92 ip=10.36.209.92

worker3 ansible\_host=10.36.209.93 ip=10.36.209.93

worker4 ansible\_host=10.36.209.94 ip=10.36.209.94

worker5 ansible\_host=10.36.209.95 ip=10.36.209.95

[kube\_control\_plane]

master1

master2

[etcd]

master1

[kube\_node]

worker1

worker2

worker3

worker4

worker5

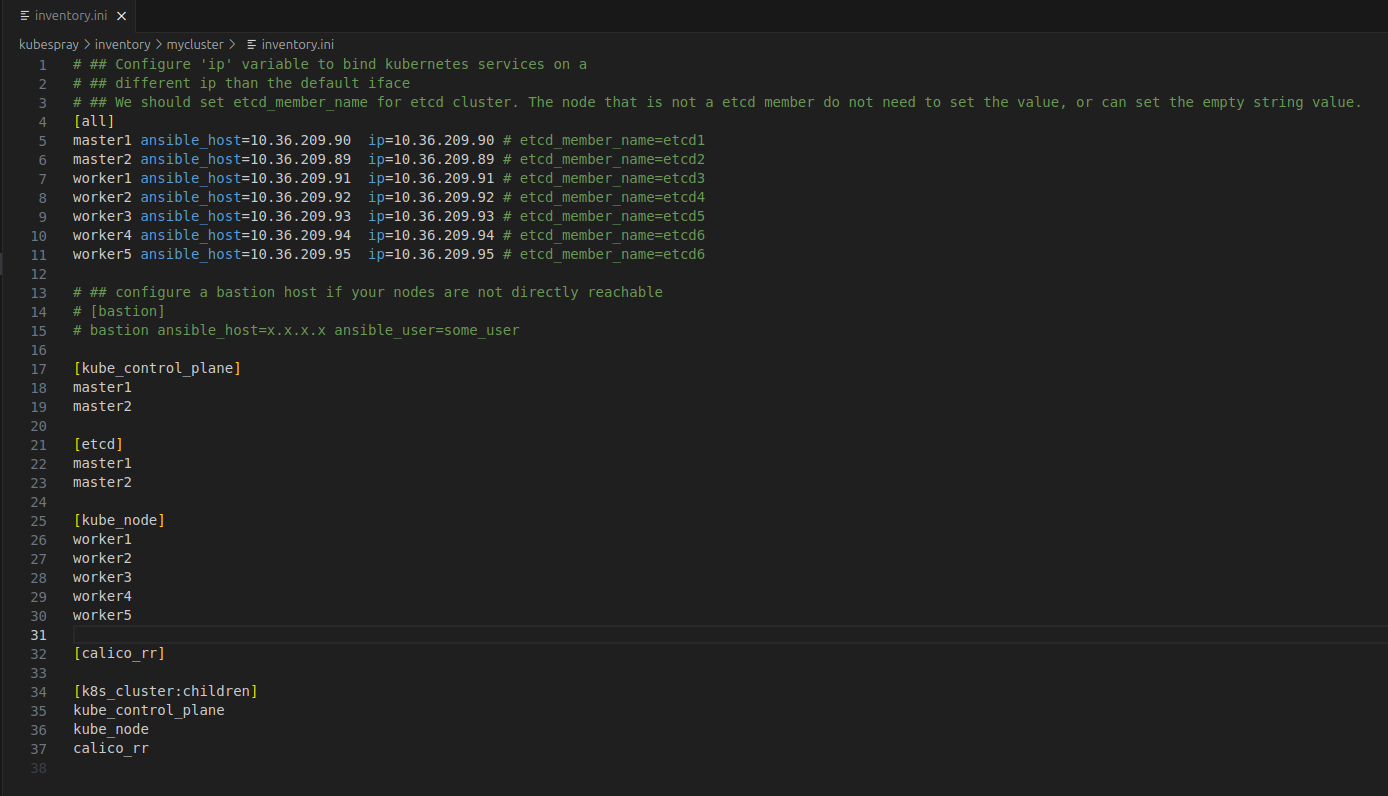
[calico\_rr]

[k8s\_cluster:children]

kube\_control\_plane

kube\_node

calico\_rr



- Pull kubespray image, chú ý image version phải cùng với kubespray: docker pull quay.io/kubespray/kubespray:v2.24.0

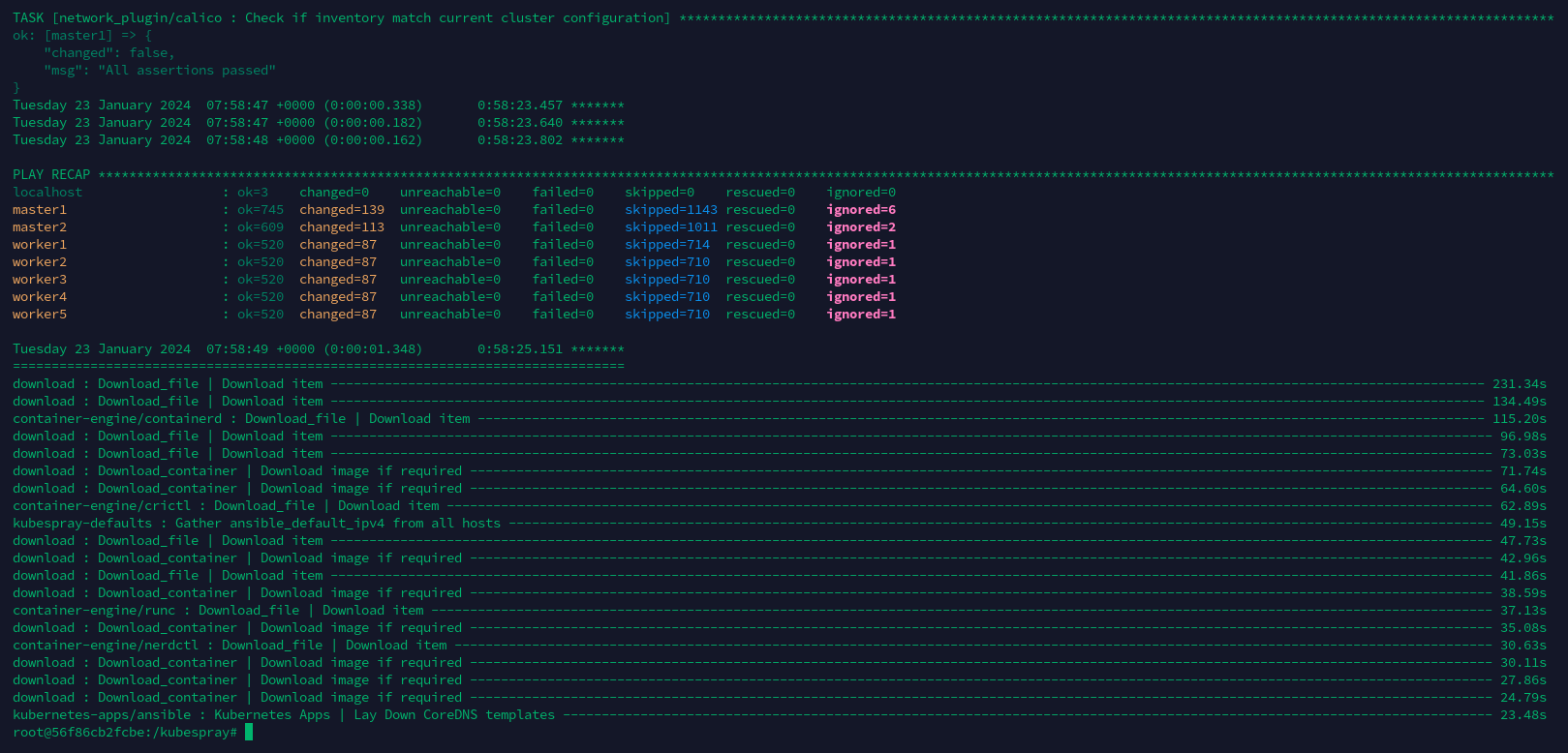
- Exec vào container, lưu ý các mount path cho đúng:

cd kubespray/

docker run --rm -it -v ./inventory/mycluster:/kubespray/inventory/mycluster -v ~/.ssh/bootstrap\_id\_rsa:/root/.ssh/id\_rsa -v ~/.ssh/bootstrap\_id\_rsa:/home/thaonp/.ssh/id\_rsa quay.io/kubespray/kubespray:v2.24.0 bash

- Lúc này đã ở trong container. Chạy câu lệnh để bắt đầu cài đặt cluster. Nhập password user 2 lần. Chờ khoảng 45p để ansible cài đặt:

ansible-playbook -i inventory/mycluster/inventory.ini cluster.yml --user=thaonp --ask-pass --become --ask-become-pass

- Kết quả: 

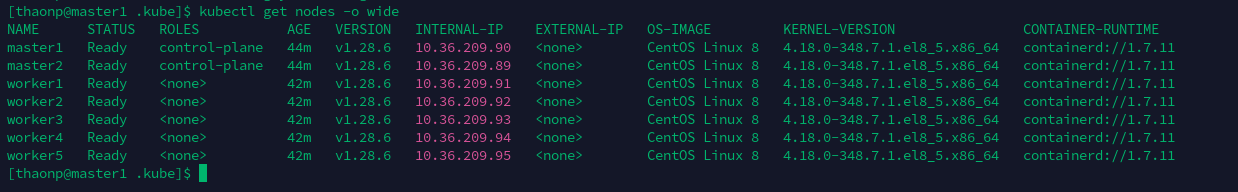
- Cấu hình trên node master1:

mkdir -p ~/.kube

sudo cp /etc/kubernetes/admin.conf ~/.kube/config

sudo chown $USER:$USER ~/.kube/config

- Kiểm tra cluster sau khi cài đặt. Cài đặt thành công nếu trạng thái tất cả các node là Ready: kubectl get nodes -o wide



## Cài đặt metrics-server

- Tải manifest metrics-server sau: wget <https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.7.0/components.yaml>  
- Sửa image và push lên private registry:

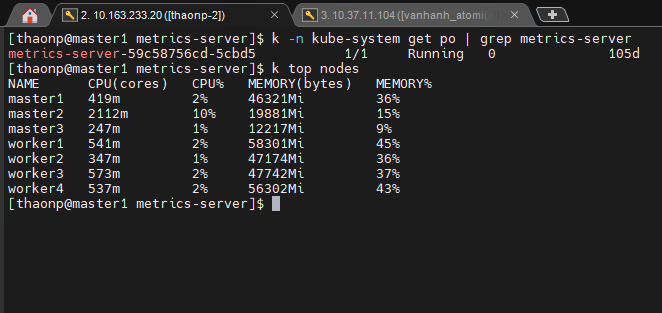
| docker pull registry.k8s.io/metrics-server/metrics-server:v0.7.0 docker tag registry.k8s.io/metrics-server/metrics-server:v0.7.0 atomi.digital.com/k8s/metrics-server:v0.7.0 docker push atomi.digital.com/k8s/metrics-server:v0.7.0 |
| --- |

- Sửa phần Deployment trong manifest. Sửa port 10251, thêm command:

| apiVersion: apps/v1 kind: Deployment metadata:  labels:  k8s-app: metrics-server  name: metrics-server  namespace: kube-system spec:  selector:  matchLabels:  k8s-app: metrics-server  strategy:  rollingUpdate:  maxUnavailable: 0  template:  metadata:  labels:  k8s-app: metrics-server  spec:  hostNetwork: true  containers:  - args:  - --cert-dir=/tmp  - --secure-port=10251  - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname  - --kubelet-use-node-status-port  - --metric-resolution=10s  command:  - /metrics-server  - --kubelet-insecure-tls  - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname   image: atomi.digital.com/k8s/metrics-server:v0.7.0  imagePullPolicy: IfNotPresent  livenessProbe:  failureThreshold: 3  httpGet:  path: /livez  port: https  scheme: HTTPS  periodSeconds: 10  name: metrics-server  ports:  - containerPort: 10251  name: https  protocol: TCP  readinessProbe:  failureThreshold: 3  httpGet:  path: /readyz  port: https  scheme: HTTPS  initialDelaySeconds: 20  periodSeconds: 10  resources:  requests:  cpu: 100m  memory: 200Mi  securityContext:  allowPrivilegeEscalation: false  capabilities:  drop:  - ALL  readOnlyRootFilesystem: true  runAsNonRoot: true  runAsUser: 1000  seccompProfile:  type: RuntimeDefault  volumeMounts:  - mountPath: /tmp  name: tmp-dir  nodeSelector:  kubernetes.io/os: linux  priorityClassName: system-cluster-critical  serviceAccountName: metrics-server  volumes:  - emptyDir: {}  name: tmp-dir |
| --- |

- Cài đặt: kubectl apply -f components.yaml

- Kiểm tra thấy pod metrics-server trạng thái running và lấy được trạng thái là được:  
kubectl top nodes



# III. Harbor registry

## Cài đặt

- Cài trên node master1.

- Cài đặt docker:

```

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

sudo yum install -y --allowerasing docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

sudo systemctl enable docker

sudo systemctl start docker

sudo groupadd docker

sudo usermod -aG docker $USER

newgrp docker

docker run hello-world

```

- Tạo thư mục chứa cài đặt Harbor: mkdir -p /data-k8s/harbor-registry

- Tạo thư mục chứa data cho Harbor: mkdir -p /data-k8s/harbor-data

- Tạo thư mục chứa script tạo self-signed certs.

```

mkdir -p /data-k8s/harbor-registry/certs/

cd /data-k8s/harbor-registry/certs/

nano create\_ssl.sh

| #/bin/bash  domain='atomi.digital.com' echo $domain openssl genrsa -out ca.key 4096 openssl req -x509 -new -nodes -sha512 -days 3650 \ -subj "/C=VI/ST=HN/L=HN/O=example/OU=Personal/CN=$domain" \ -key ca.key \ -out ca.crt openssl genrsa -out $domain.key 4096 openssl req -sha512 -new \ -subj "/C=VI/ST=HN/L=HN/O=example/OU=Personal/CN=$domain" \ -key $domain.key \ -out $domain.csr  cat > v3.ext <<-EOF authorityKeyIdentifier=keyid,issuer basicConstraints=CA:FALSE keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment extendedKeyUsage = serverAuth subjectAltName = @alt\_names  [alt\_names] DNS.1=$domain DNS.2=hostname EOF  openssl x509 -req -sha512 -days 3650 \ -extfile v3.ext \ -CA ca.crt -CAkey ca.key -CAcreateserial \ -in $domain.csr \ -out $domain.crt openssl x509 -inform PEM -in $domain.crt -out $domain.cert |
| --- |

- Chạy script để tạo certs.

```

sudo chmod +x create\_ssl.sh

sudo ./create\_ssl.sh

```



- Tải bộ cài Harbor offline:

```

cd /data-k8s/harbor-registry

wget https://github.com/goharbor/harbor/releases/download/v2.10.0/harbor-offline-installer-v2.10.0.tgz

tar -xvf harbor-offline-installer-v2.10.0.tgz

cd harbor/

cp harbor.yml.tmpl harbor.yml

```

- Cấu hình file cài đặt harbor.yaml. Cập nhật một số tham số trong file harbor.yaml như sau:

```

hostname: atomi.digital.com

certificate: /data-k8s/harbor-registry/certs/atomi.digital.com.crt

private\_key: /data-k8s/harbor-registry/certs/atomi.digital.com.key

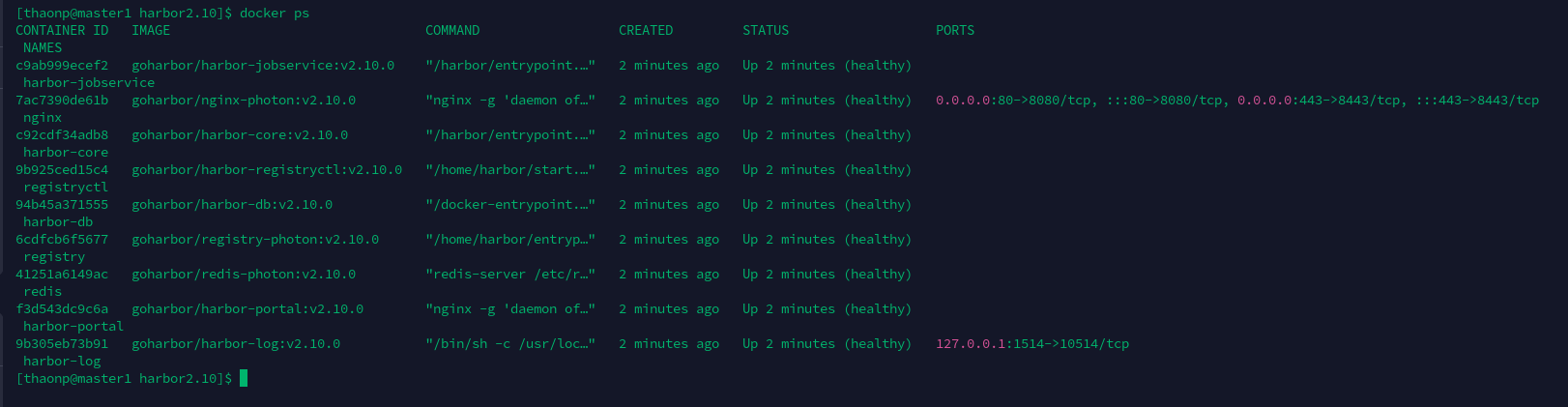
harbor\_admin\_password: Harbor\_123

password: root\_123

data\_volume: /data-k8s/harbor-data

```

- Chạy script cài đặt: sudo ./install.sh. Chờ khoảng 5p cho các containers sẵn sàng.



## Cấu hình docker và containerd để pull image

- Thực hiện trên tất cả các node trong cluster.

- Cấu hình file hosts.

```

sudo nano /etc/hosts

# them dong sau

10.36.209.90 atomi.digital.com

```

- Cấu hình docker daemon để login được vào registry không bảo mật.

```

| sudo nano /etc/docker/daemon.json # them dong sau {  "insecure-registries" : [ "atomi.digital.com" ] } sudo systemctl restart docker |
| --- |

```

- Trên node master, login vào registry để có token auth.

```

docker login atomi.digital.com

cat ~/.docker/config.json

```

- Cấu hình containerd để kết nối với registry.

| cat <<EOF | sudo tee /etc/containerd/config.toml version = 2 root = "/var/lib/containerd" state = "/run/containerd" oom\_score = 0  [grpc]  max\_recv\_message\_size = 16777216  max\_send\_message\_size = 16777216  [debug]  level = "info"  [metrics]  address = ""  grpc\_histogram = false  [plugins]  [plugins."io.containerd.grpc.v1.cri"]  sandbox\_image = "registry.k8s.io/pause:3.9"  max\_container\_log\_line\_size = -1  enable\_unprivileged\_ports = false  enable\_unprivileged\_icmp = false  [plugins."io.containerd.grpc.v1.cri".containerd]  default\_runtime\_name = "runc"  snapshotter = "overlayfs"  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes]  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]  runtime\_type = "io.containerd.runc.v2"  runtime\_engine = ""  runtime\_root = ""  base\_runtime\_spec = "/etc/containerd/cri-base.json"   [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]  systemdCgroup = true  binaryName = "/usr/local/bin/runc"    [plugins."io.containerd.grpc.v1.cri".registry]  [plugins."io.containerd.grpc.v1.cri".registry.auths]   [plugins."io.containerd.grpc.v1.cri".registry.configs]  [plugins."io.containerd.grpc.v1.cri".registry.configs."atomi.digital.com".tls]  insecure\_skip\_verify = true  [plugins."io.containerd.grpc.v1.cri".registry.configs."atomi.digital.com".auth]  auth = "YWRtaW46SGFyYm9yXzEyMw=="   [plugins."io.containerd.grpc.v1.cri".registry.headers]  [plugins."io.containerd.grpc.v1.cri".registry.mirrors]  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."atomi.digital.com"]  endpoint = ["https://atomi.digital.com"] EOF |
| --- |



- Restart containerd: sudo systemctl restart containerd

- Thêm các CA certs của Harbor vào server:

```

cd /data-k8s/harbor-registry/certs

sudo cp ca.crt /etc/pki/ca-trust/source/anchors/

sudo update-ca-trust

```

- Pull thử 1 image từ Harbor:

```

docker tag hello-world:latest atomi.digital.com/library/hello-world:latest

docker push atomi.digital.com/library/hello-world:latest

sudo /usr/local/bin/crictl pull atomi.digital.com/library/hello-world:latest

```

# IV. Longhorn persistent storage

- Thực hiện trên node master1

- Cài đặt helm

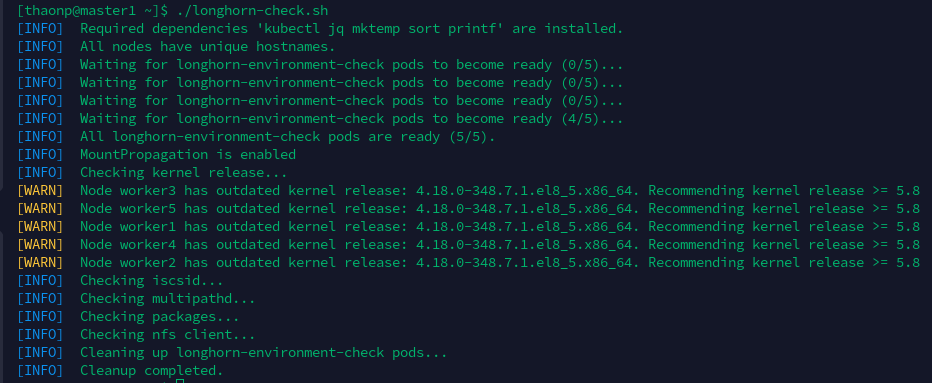
| curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 sudo chmod 700 get\_helm.sh ./get\_helm.sh |
| --- |

- Tạo folder lưu trữ cho longhorn trên tất cả các node trong cluster.:

| mkdir -p /data-k8s/longhorn-data |
| --- |

- Tải script kiểm tra môi trường cài đặt:

| curl -s https://raw.githubusercontent.com/longhorn/longhorn/v1.5.4/scripts/environment\_check.sh | bash |
| --- |



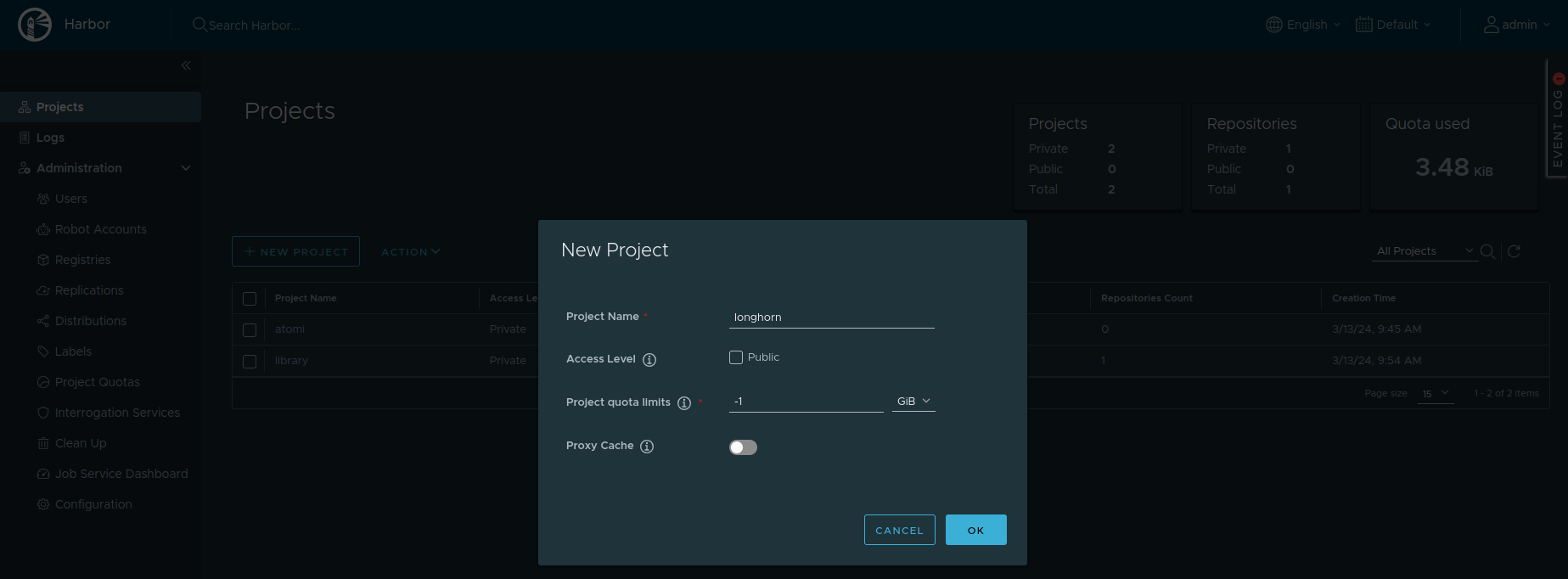
- Thêm repo longhorn và tải chart của longhorn về server:

| mkdir /data-k8s/longhorn-storage cd /data-k8s/longhorn-storage helm repo add longhorn https://charts.longhorn.io helm repo update helm fetch longhorn/longhorn --version 1.5.4 --untar |
| --- |

- Chỉnh sửa file cấu hình helm chart:

| cd longhorn cp values.yaml values-longhorn.yaml |
| --- |

- Vào Harbor, tạo một project mới tên là longhorn như sau:



- Pull các image sau và đánh tag lại push lên Harbor để dùng:

| # pull images  docker pull docker.io/longhornio/longhorn-engine:v1.5.4 docker pull docker.io/longhornio/longhorn-manager:v1.5.4 docker pull docker.io/longhornio/longhorn-ui:v1.5.4 docker pull docker.io/longhornio/longhorn-instance-manager:v1.5.4 docker pull docker.io/longhornio/longhorn-share-manager:v1.5.4 docker pull docker.io/longhornio/backing-image-manager:v1.5.4 docker pull docker.io/longhornio/support-bundle-kit:v0.0.33 docker pull docker.io/longhornio/csi-attacher:v4.4.2 docker pull docker.io/longhornio/csi-provisioner:v3.6.2 docker pull docker.io/longhornio/csi-node-driver-registrar:v2.9.2 docker pull docker.io/longhornio/csi-resizer:v1.9.2 docker pull docker.io/longhornio/csi-snapshotter:v6.3.2 docker pull docker.io/longhornio/livenessprobe:v2.12.0 # tag images docker tag docker.io/longhornio/longhorn-engine:v1.5.4 atomi.digital.com/longhorn/longhorn-engine:v1.5.4  docker tag docker.io/longhornio/longhorn-manager:v1.5.4 atomi.digital.com/longhorn/longhorn-manager:v1.5.4  docker tag docker.io/longhornio/longhorn-ui:v1.5.4 atomi.digital.com/longhorn/longhorn-ui:v1.5.4 docker tag docker.io/longhornio/longhorn-instance-manager:v1.5.4 atomi.digital.com/longhorn/longhorn-instance-manager:v1.5.4 docker tag docker.io/longhornio/longhorn-share-manager:v1.5.4 atomi.digital.com/longhorn/longhorn-share-manager:v1.5.4 docker tag docker.io/longhornio/backing-image-manager:v1.5.4 atomi.digital.com/longhorn/backing-image-manager:v1.5.4 docker tag docker.io/longhornio/support-bundle-kit:v0.0.33 atomi.digital.com/longhorn/support-bundle-kit:v0.0.33 docker tag docker.io/longhornio/csi-attacher:v4.4.2 atomi.digital.com/longhorn/csi-attacher:v4.4.2 docker tag docker.io/longhornio/csi-provisioner:v3.6.2 atomi.digital.com/longhorn/csi-provisioner:v3.6.2 docker tag docker.io/longhornio/csi-node-driver-registrar:v2.9.2 atomi.digital.com/longhorn/csi-node-driver-registrar:v2.9.2 docker tag docker.io/longhornio/csi-resizer:v1.9.2 atomi.digital.com/longhorn/csi-resizer:v1.9.2 docker tag docker.io/longhornio/csi-snapshotter:v6.3.2 atomi.digital.com/longhorn/csi-snapshotter:v6.3.2 docker tag docker.io/longhornio/livenessprobe:v2.12.0 atomi.digital.com/longhorn/livenessprobe:v2.12.0 # push images docker push atomi.digital.com/longhorn/longhorn-engine:v1.5.4 docker push atomi.digital.com/longhorn/longhorn-manager:v1.5.4 docker push atomi.digital.com/longhorn/longhorn-ui:v1.5.4 docker push atomi.digital.com/longhorn/longhorn-instance-manager:v1.5.4 docker push atomi.digital.com/longhorn/longhorn-share-manager:v1.5.4 docker push atomi.digital.com/longhorn/backing-image-manager:v1.5.4 docker push atomi.digital.com/longhorn/support-bundle-kit:v0.0.33 docker push atomi.digital.com/longhorn/csi-attacher:v4.4.2 docker push atomi.digital.com/longhorn/csi-provisioner:v3.6.2 docker push atomi.digital.com/longhorn/csi-node-driver-registrar:v2.9.2 docker push atomi.digital.com/longhorn/csi-resizer:v1.9.2 docker push atomi.digital.com/longhorn/csi-snapshotter:v6.3.2 docker push atomi.digital.com/longhorn/livenessprobe:v2.12.0 |
| --- |

- Sửa các tham số như sau:

| image:  longhorn:  engine:  repository: atomi.digital.com/longhorn/longhorn-engine  tag: v1.5.4  manager:  repository: atomi.digital.com/longhorn/longhorn-manager  tag: v1.5.4  ui:  repository: atomi.digital.com/longhorn/longhorn-ui  tag: v1.5.4  instanceManager:  repository: atomi.digital.com/longhorn/longhorn-instance-manager  tag: v1.5.4  shareManager:  repository: atomi.digital.com/longhorn/longhorn-share-manager  tag: v1.5.4  backingImageManager:  repository: atomi.digital.com/longhorn/backing-image-manager  tag: v1.5.4  supportBundleKit:  repository: atomi.digital.com/longhorn/support-bundle-kit  tag: v0.0.33  csi:  attacher:  repository: atomi.digital.com/longhorn/csi-attacher  tag: v4.4.2  provisioner:  repository: atomi.digital.com/longhorn/csi-provisioner  tag: v3.6.2  nodeDriverRegistrar:  repository: atomi.digital.com/longhorn/csi-node-driver-registrar  tag: v2.9.2  resizer:  repository: atomi.digital.com/longhorn/csi-resizer  tag: v1.9.2  snapshotter:  repository: atomi.digital.com/longhorn/csi-snapshotter  tag: v6.3.2  livenessProbe:  repository: atomi.digital.com/longhorn/livenessprobe  tag: v2.12.0  pullPolicy: IfNotPresent  service:  ui:  type: NodePort  nodePort: 30002  helmPreUpgradeCheckerJob:  enabled: false  defaultSettings:   defaultDataPath: /data-k8s/longhorn-data  replicaSoftAntiAffinity: false  replicaAutoBalance: true  storageMinimalAvailablePercentage: 10  upgradeChecker: false  defaultReplicaCount: 2  backupstorePollInterval: 600  recurringSuccessfulJobsHistoryLimit: 3  recurringFailedJobsHistoryLimit: 3  supportBundleFailedHistoryLimit: 3  nodeDownPodDeletionPolicy: do-nothing  nodeDrainPolicy: do-nothing  longhornUI:  replicas: 1  namespaceOverride: "storage" |
| --- |

- Cài đặt:

| # install from fetched release helm install longhorn . --namespace storage --create-namespace --values values-longhorn.yaml  # upgrade release nếu cần - sửa chart value trước helm upgrade longhorn . --namespace storage --values values-longhorn.yaml |
| --- |

- Tạo Storage Class mặc định.

| cd /data-k8s/longhorn-storage nano longhorn-storageclass-retain.yaml kubectl apply -n storage -f longhorn-storageclass-retain.yaml |
| --- |

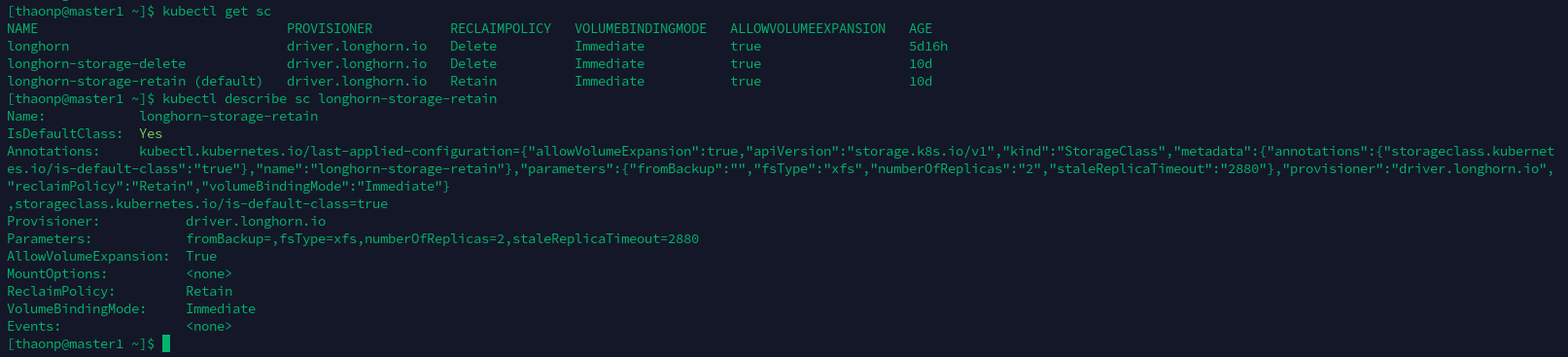
- longhorn-storageclass-retain.yaml

| kind: StorageClass apiVersion: storage.k8s.io/v1 metadata:  name: longhorn-storage-retain  annotations:  storageclass.kubernetes.io/is-default-class: "true" provisioner: driver.longhorn.io allowVolumeExpansion: true reclaimPolicy: Retain volumeBindingMode: Immediate parameters:  numberOfReplicas: "2"  staleReplicaTimeout: "2880"  fromBackup: ""  fsType: "xfs"  migratable: "true"  dataLocality: "best-effort" |
| --- |

- Patch storageclass ban đầu của longhorn để bỏ làm SC mặc định:

| kubectl patch storageclass longhorn -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}' |
| --- |

- Kết quả:



# V. Logging system

- Thực hiện trên node master1. Cần cài đặt Longhorn trước vì Elasticsearch sẽ lưu dữ liệu ở Persistent Volume do Longhorn quản lý.

- Tạo namespace kube-logging: kubectl create ns kube-logging

- Clone repo manifests của logging ở đây: <http://10.36.209.79/LienViet24H/manifest/new-k8s-manifests>

- Cài đặt:

```

kubectl apply -n kube-logging -f elasticsearch\_statefulset.yaml

kubectl apply -n kube-logging -f elasticsearch\_svc.yaml

kubectl apply -n kube-logging -f fluentd/

kubectl apply -n kube-logging -f kibana.yaml

```

- Sau khi chạy lần đầu tiên, phải bỏ cấu hình initial\_master\_nodes trong cluster:

```

kubectl -n kube-logging edit sts es-cluster

# Xóa 2 dòng sau, sau đó ấn :wq để lưu

- name: cluster.initial\_master\_nodes

value: "es-cluster-0,es-cluster-1,es-cluster-2"

```

# VI. Istio service mesh

- Thực hiện trên node master1.

- Tải xuống bộ cài của Istio:

| cd /data-k8s curl -L https://istio.io/downloadIstio | ISTIO\_VERSION=1.20.4 TARGET\_ARCH=x86\_64 sh - # Chạy script kiểm tra trước khi cài cd istio-1.20.4 ./bin/istioctl x precheck sudo cp bin/istioctl /usr/local/bin/istioctl |
| --- |

1. **Cài đặt Istio**

| istioctl manifest apply --set profile=default --set components.cni.enabled=true --set cni.repair.deletePods=true # output This will install the Istio 1.20.4 "default" profile (with components: Istio core, Istiod, CNI, Ingress gateways, and Egress gateways) into the cluster. Proceed? (y/N) y ✔ Istio core installed  ✔ Istiod installed  ✔ Egress gateways installed  ✔ Ingress gateways installed  ✔ CNI installed ✔ Installation complete  Made this installation the default for injection and validation. |
| --- |

**2. Cài đặt các addons cần thiết**

- Sửa service addons Kiali thành NodePort:

| cd /data-k8s/istio-1.20.4/samples/addons  nano kiali.yaml  # tìm đến đoạn Service sau  apiVersion: v1 kind: Service metadata:  name: kiali  namespace: istio-system  labels:  helm.sh/chart: kiali-server-1.76.0  app: kiali  app.kubernetes.io/name: kiali  app.kubernetes.io/instance: kiali  version: "v1.76.0"  app.kubernetes.io/version: "v1.76.0"  app.kubernetes.io/managed-by: Helm  app.kubernetes.io/part-of: "kiali"  annotations: spec:  type: NodePort  ports:  - name: http  appProtocol: http  protocol: TCP  port: 20001  nodePort: 30007  - name: http-metrics  appProtocol: http  protocol: TCP  port: 9090  nodePort: 30349  selector:  app.kubernetes.io/name: kiali  app.kubernetes.io/instance: kiali |
| --- |

- Sửa service addons Jaeger thành NodePort:

| cd /data-k8s/istio-1.20.4/samples/addons  nano jaeger.yaml  # tìm đến đoạn Service sau  apiVersion: v1 kind: Service metadata:  name: tracing  namespace: istio-system  labels:  app: jaeger spec:  type: NodePort  ports:  - name: http-query  port: 80  protocol: TCP  targetPort: 16686  nodePort: 30006  # Note: Change port name if you add '--query.grpc.tls.enabled=true'  - name: grpc-query  port: 16685  protocol: TCP  targetPort: 16685  nodePort: 30530  selector:  app: jaeger |
| --- |

- Thực hiện cài đặt các addon cần thiết:

| cd /data-k8s/istio-1.20.4  # Xóa addon Loki (không cần thiết) rm -rf samples/addons/loki.yaml kubectl apply -f samples/addons |
| --- |

- Tăng resources cho proxy sidecar:

| cd /data-k8s/istio-1.20.4 nano istioOperator.yaml kubectl apply -f istioOperator.yaml |
| --- |

| apiVersion: install.istio.io/v1alpha1 kind: IstioOperator metadata:  namespace: istio-system  name: istiocontrolplane spec:  profile: demo  values:  global:  proxy:  resources:  requests:  cpu: 500m  memory: 256Mi  limits:  cpu: 4000m  memory: 8192Mi |
| --- |

- Tạo Gateway và VirtualService cho transaction service:

| cd /data-k8s/istio-1.20.4 nano gateway.yaml  nano transaction-vs.yaml  kubectl apply -f gateway.yaml kubectl apply -f transaction-vs.yaml |
| --- |

- gateway.yaml

| apiVersion: networking.istio.io/v1alpha3 kind: Gateway metadata:  name: services-gateway  namespace: test spec:  # The selector matches the ingress gateway pod labels.  # If you installed Istio using Helm following the standard documentation, this would be "istio=ingress"  selector:  istio: ingressgateway # use istio default controller  servers:  - port:  number: 8080  name: http  protocol: HTTP  hosts:  - "\*" |
| --- |

- transaction-vs.yaml

| apiVersion: networking.istio.io/v1alpha3 kind: VirtualService metadata:  name: transaction-ktm-vs  namespace: test spec:  hosts:  - "\*"  gateways:  - services-gateway  http:  - name: "transaction-ktm-routes"  route:  - destination:  host: transaction-service-ktm.test.svc.cluster.local  port:  number: 80  match:  - uri:  prefix: "/ktm"  rewrite:  uri: " " --- apiVersion: networking.istio.io/v1alpha3 kind: VirtualService metadata:  name: transaction-ktm-staging-vs  namespace: test spec:  hosts:  - "\*"  gateways:  - services-gateway  http:  - name: "transaction-ktm-staging-routes"  route:  - destination:  host: transaction-service-ktm-staging.test.svc.cluster.local  port:  number: 80  match:  - uri:  prefix: "/ktm-staging"  rewrite:  uri: " " |
| --- |

- Thêm label cho namespace cần thiết để hướng dẫn Istio tự động inject Envoy proxy sidecar khi triển khai ứng dụng sau này: kubectl label namespace test istio-injection=enabled

# VII. Monitoring system

## Cài đặt

- Thêm helm repo và tải chart về server:

```

helm repo add stable https://charts.helm.sh/stable

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

mkdir -p /data-k8s/monitoring-system

cd /data-k8s/monitoring-system

helm fetch prometheus-community/kube-prometheus-stack --version 56.4.0 --untar

cd kube-prometheus-stack

cp values.yaml values-kubeprom-stack.yaml

kubectl create ns monitoring-system

```

- Cài đặt:

```

# dry-run

cd /data-k8s/monitoring-system/kube-prometheus-stack

helm install -n monitoring-system -f values-kubeprom-stack.yaml stable . --dry-run=server

# install

helm install -n monitoring-system -f values-kubeprom-stack.yaml stable .

```

- Sửa các service để truy cập bằng NodePort:

```

kubectl patch svc stable-kube-prometheus-sta-prometheus -n monitoring-system -p '{"spec": {"type": "NodePort"}}'

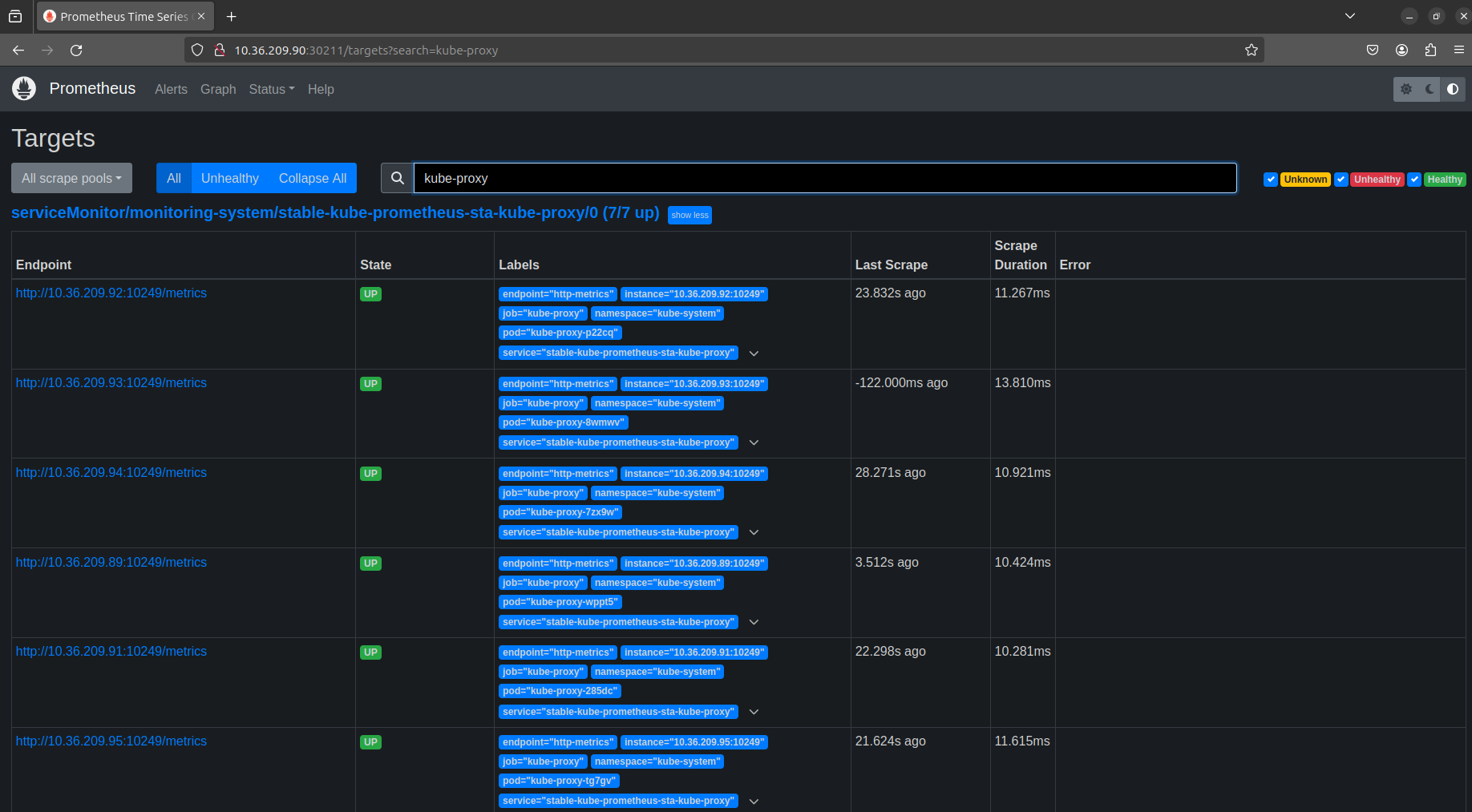
kubectl patch svc stable-grafana -n monitoring-system -p '{"spec": {"type": "NodePort"}}'

```

- Truy cập Grafana dashboard. Tài khoản mặc định là admin/prom-operator

- Sửa lỗi Prometheus không lấy được metrics của kube-proxy:

| kubectl -n kube-system edit cm/kube-proxy ## Đổi từ  metricsBindAddress: 127.0.0.1:10249 ### <-- Too secure ## Đổi thành  metricsBindAddress: 0.0.0.0:10249 # Xóa pod cũ để nhận config map mới kubectl -n kube-system delete pod -l k8s-app=kube-proxy |
| --- |

- Kết quả (scrape được metrics của kube-proxy): 

- Để monitor các services của bên thứ ba cần cài đặt exporter tương ứng hoặc expose metrics endpoint có sẵn ra (vd: RabbitMQ) để Prometheus có thể scrape được metrics. Hiện tại đã cài 4 monitor sau: Redis, OracleDB, RabbitMQ, Elasticsearch.

- Yêu cầu: Đã cài đặt Prometheus stack dùng helm chart ở trên.

## Cài đặt Redis exporter

- Tải chart redis exporter về server:

```

cd /data-k8s/monitoring

helm fetch prometheus-community/prometheus-redis-exporter --version 6.1.1 --untar

cd prometheus-redis-exporter/

cp values.yaml values-redis-exporter.yaml

```

- Sửa các giá trị sau trong values-redis-exporter.yaml:

```

rbac:

pspEnabled: false

image:

# repository: oliver006/redis\_exporter # Pull image ở đây về, đánh tag và push lên harbor

repository: atomi.digital.com/monitoring/redis\_exporter

# redis address: redis://<host>:<port>

redisAddress: redis://10.36.209.90:6379

```

- Cài đặt:

```

cd /data-k8s/monitoring-system/prometheus-redis-exporter

# dry run

helm install -n monitoring-system -f values-redis-exporter.yaml redis-exporter . --dry-run=server

# install

helm install -n monitoring-system -f values-redis-exporter.yaml redis-exporter .

```

- Thêm scrape config cho Prometheus:

```

cd /data-k8s/monitoring-system/kube-prometheus-stack

nano values-kubeprom-stack.yaml

# thêm đoạn sau:

additionalScrapeConfigs:

- job\_name: redis\_exporter

static\_configs:

- targets: ['redis-exporter-prometheus-redis-exporter:9121']

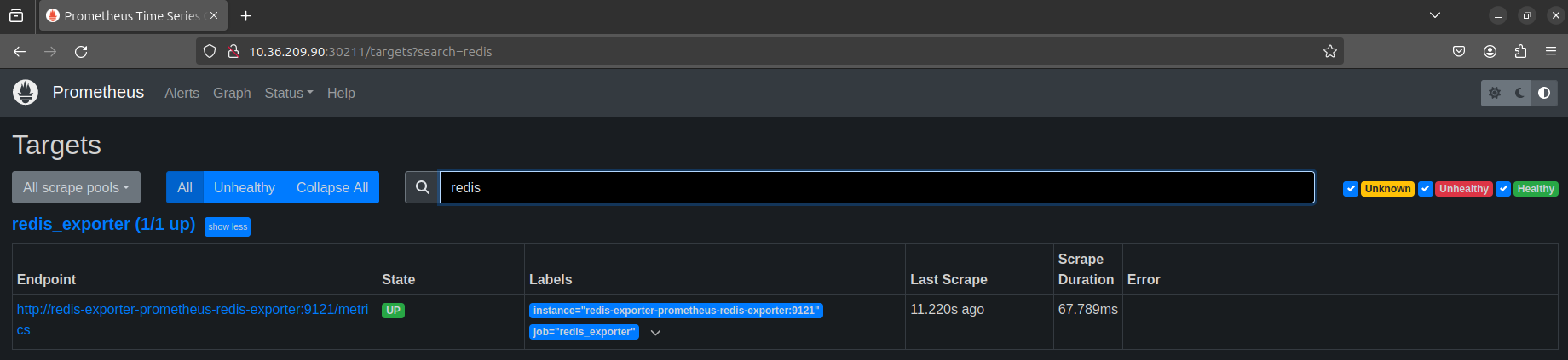
# chạy lại helm chart Prometheus

cd /data-k8s/monitoring-system/kube-prometheus-stack

helm upgrade -n monitoring-system -f values-kubeprom-stack.yaml stable .

```

- Kiểm tra trạng thái exporter trong Status > Targets của Prometheus thấy Up là được:



- Tải dashboard JSON ở đây: https://grafana.com/grafana/dashboards/763

- Trên Grafana, vào Dashboards > New > Import > Upload dashboard JSON file. Chọn template vừa tải, chọn Datasource là Prometheus (default)

## Cài đặt OracleDB exporter

- Trên server db, chạy query sau để lấy database name và domain:

```

select value from v$parameter where name='service\_names';

===

LV24UAT2.lpbank.com.vn

```

- Tạo connection string như sau:

```

oracle://<username>:<password>@<host>:<port>/<service\_names>

oracle://ESMAC\_VV\_TE:esmac\_vv\_te@10.37.11.112:1521/LV24UAT2.lpbank.com.vn

```

- Thêm quyền SELECT cho user sẽ dùng, vd ESMAC\_VV\_TE:

```

GRANT SELECT ON SYS.dba\_tablespace\_usage\_metrics TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.dba\_tablespaces TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$system\_wait\_class TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$asm\_diskgroup\_stat TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$datafile TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$sysstat TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$process TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$waitclassmetric TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$session TO ESMAC\_VV\_TE;

GRANT SELECT ON SYS.v\_$resource\_limit TO ESMAC\_VV\_TE;

```

- Tạo manifest oracle-exporter.yaml như sau:

```

apiVersion: v1

kind: Secret

metadata:

name: oracledb-exporter-secret

namespace: monitoring-system

type: Opaque

stringData:

datasource: oracle://ESMAC\_VV\_TE:esmac\_vv\_te@10.37.11.112:1521/LV24UAT2.lpbank.com.vn

---

apiVersion: v1

kind: Service

metadata:

name: oracledb-exporter

namespace: monitoring-system

labels:

app: oracledb-exporter

spec:

type: ClusterIP

ports:

- name: oracledb-exporter

port: 9161

protocol: TCP

selector:

app: oracledb-exporter

---

apiVersion: v1

kind: ConfigMap

metadata:

name: custom-metrics

namespace: monitoring-system

data:

custom-metrics.toml: |

[[metric]]

context = "slow\_queries"

metricsdesc = { p95\_time\_usecs= "Gauge metric with percentile 95 of elapsed time.", p99\_time\_usecs= "Gauge metric with percentile 99 of elapsed time." }

request = "select percentile\_disc(0.95) within group (order by elapsed\_time) as p95\_time\_usecs, percentile\_disc(0.99) within group (order by elapsed\_time) as p99\_time\_usecs from v$sql where last\_active\_time >= sysdate - 5/(24\*60)"

[[metric]]

context = "big\_queries"

metricsdesc = { p95\_rows= "Gauge metric with percentile 95 of returned rows.", p99\_rows= "Gauge metric with percentile 99 of returned rows." }

request = "select percentile\_disc(0.95) within group (order by rownum) as p95\_rows, percentile\_disc(0.99) within group (order by rownum) as p99\_rows from v$sql where last\_active\_time >= sysdate - 5/(24\*60)"

[[metric]]

context = "size\_user\_segments\_top100"

metricsdesc = {table\_bytes="Gauge metric with the size of the tables in user segments."}

labels = ["segment\_name"]

request = "select \* from (select segment\_name,sum(bytes) as table\_bytes from user\_segments where segment\_type='TABLE' group by segment\_name) order by table\_bytes DESC FETCH NEXT 100 ROWS ONLY"

[[metric]]

context = "size\_user\_segments\_top100"

metricsdesc = {table\_partition\_bytes="Gauge metric with the size of the table partition in user segments."}

labels = ["segment\_name"]

request = "select \* from (select segment\_name,sum(bytes) as table\_partition\_bytes from user\_segments where segment\_type='TABLE PARTITION' group by segment\_name) order by table\_partition\_bytes DESC FETCH NEXT 100 ROWS ONLY"

[[metric]]

context = "size\_user\_segments\_top100"

metricsdesc = {cluster\_bytes="Gauge metric with the size of the cluster in user segments."}

labels = ["segment\_name"]

request = "select \* from (select segment\_name,sum(bytes) as cluster\_bytes from user\_segments where segment\_type='CLUSTER' group by segment\_name) order by cluster\_bytes DESC FETCH NEXT 100 ROWS ONLY"

[[metric]]

context = "size\_dba\_segments\_top100"

metricsdesc = {table\_bytes="Gauge metric with the size of the tables in user segments."}

labels = ["segment\_name"]

request = "select \* from (select segment\_name,sum(bytes) as table\_bytes from dba\_segments where segment\_type='TABLE' group by segment\_name) order by table\_bytes DESC FETCH NEXT 100 ROWS ONLY"

[[metric]]

context = "size\_dba\_segments\_top100"

metricsdesc = {table\_partition\_bytes="Gauge metric with the size of the table partition in user segments."}

labels = ["segment\_name"]

request = "select \* from (select segment\_name,sum(bytes) as table\_partition\_bytes from dba\_segments where segment\_type='TABLE PARTITION' group by segment\_name) order by table\_partition\_bytes DESC FETCH NEXT 100 ROWS ONLY"

[[metric]]

context = "size\_dba\_segments\_top100"

metricsdesc = {cluster\_bytes="Gauge metric with the size of the cluster in user segments."}

labels = ["segment\_name"]

request = "select \* from (select segment\_name,sum(bytes) as cluster\_bytes from dba\_segments where segment\_type='CLUSTER' group by segment\_name) order by cluster\_bytes DESC FETCH NEXT 100 ROWS ONLY"

[[metric]]

context = "cache\_hit\_ratio"

metricsdesc = {percentage="Gauge metric with the cache hit ratio."}

request = "select Round(((Sum(Decode(a.name, 'consistent gets', a.value, 0)) + Sum(Decode(a.name, 'db block gets', a.value, 0)) - Sum(Decode(a.name, 'physical reads', a.value, 0)) )/ (Sum(Decode(a.name, 'consistent gets', a.value, 0)) + Sum(Decode(a.name, 'db block gets', a.value, 0)))) \*100,2) as percentage FROM v$sysstat a"

[[metric]]

context = "startup"

metricsdesc = {time\_seconds="Database startup time in seconds."}

request = "SELECT (SYSDATE - STARTUP\_TIME) \* 24 \* 60 \* 60 AS time\_seconds FROM V$INSTANCE"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: oracledb-exporter

namespace: monitoring-system

spec:

selector:

matchLabels:

app: oracledb-exporter

replicas: 1

template:

metadata:

labels:

app: oracledb-exporter

annotations:

prometheus.io/scrape: "true"

prometheus.io/port: "9161"

prometheus.io/path: "/metrics"

spec:

containers:

- name: oracledb-exporter

ports:

- containerPort: 9161

image: atomi.digital.com/monitoring/oracledb\_exporter:0.5.2

env:

- name: DATA\_SOURCE\_NAME

valueFrom:

secretKeyRef:

name: oracledb-exporter-secret

key: datasource

- name: CUSTOM\_METRICS

value: /tmp/custom-metrics.toml

volumeMounts:

- name: custom-metrics

mountPath: /tmp/custom-metrics.toml

subPath: custom-metrics.toml

volumes:

- name: custom-metrics

configMap:

defaultMode: 0644

name: custom-metrics

```

- Cài đặt exporter: kubectl apply -f oracle-exporter.yaml

- Thêm scrape config cho Prometheus:

```

cd /data-k8s/monitoring-system/kube-prometheus-stack

nano values-kubeprom-stack.yaml

# thêm đoạn sau:

additionalScrapeConfigs:

- job\_name: oracledb\_exporter

static\_configs:

- targets: ['oracledb-exporter:9161']

# chạy lại helm chart Prometheus

cd /data-k8s/monitoring-system/kube-prometheus-stack

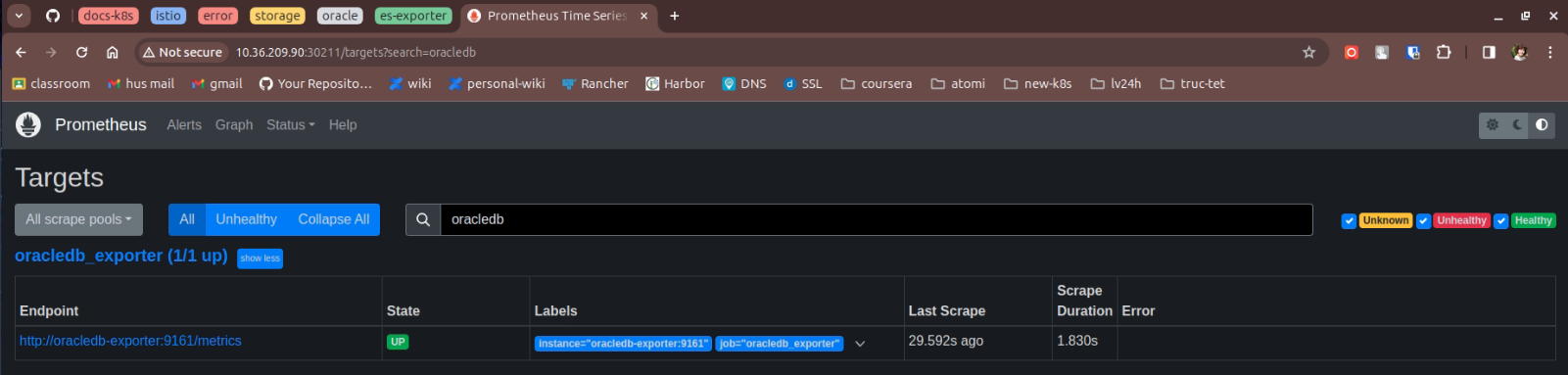
# dry run

helm upgrade -n monitoring-system -f values-kubeprom-stack.yaml stable . --dry-run=server

# upgrade

helm upgrade -n monitoring-system -f values-kubeprom-stack.yaml stable .

```

- Kiểm tra trạng thái exporter trong Status > Targets của Prometheus thấy Up là được: 

- Tải dashboard JSON ở đây: https://grafana.com/grafana/dashboards/3333

- Trên Grafana, vào Dashboards > New > Import > Upload dashboard JSON file. Chọn template vừa tải, chọn Datasource là Prometheus (default)

## Cài đặt RabbitMQ exporter

- Đang chạy RabbitMQ bằng docker compose, cần enable plugin metrics.

- Exec vào container: docker exec -it rabbitmq-rabbitmq-1 bash

- Trong container, chạy lệnh sau để enable plugin: rabbitmq-plugins enable rabbitmq\_prometheus

- Thêm cấu hình port cho metrics endpoint trong file compose nếu chưa có:

```

services:

rabbitmq:

image: "rabbitmq:3.12-management"

ports:

- "8000:5672"

- "8001:15672"

- "15692:15692" # metrics endpoint

```

- Thêm scrape config cho Prometheus:

```

cd /data-k8s/monitoring-system/kube-prometheus-stack

nano values-kubeprom-stack.yaml

# thêm đoạn sau:

additionalScrapeConfigs:

- job\_name: rabbitmq\_exporter

static\_configs:

- targets: ['10.36.209.90:15692']

# chạy lại helm chart Prometheus

cd /data-k8s/monitoring-system/kube-prometheus-stack

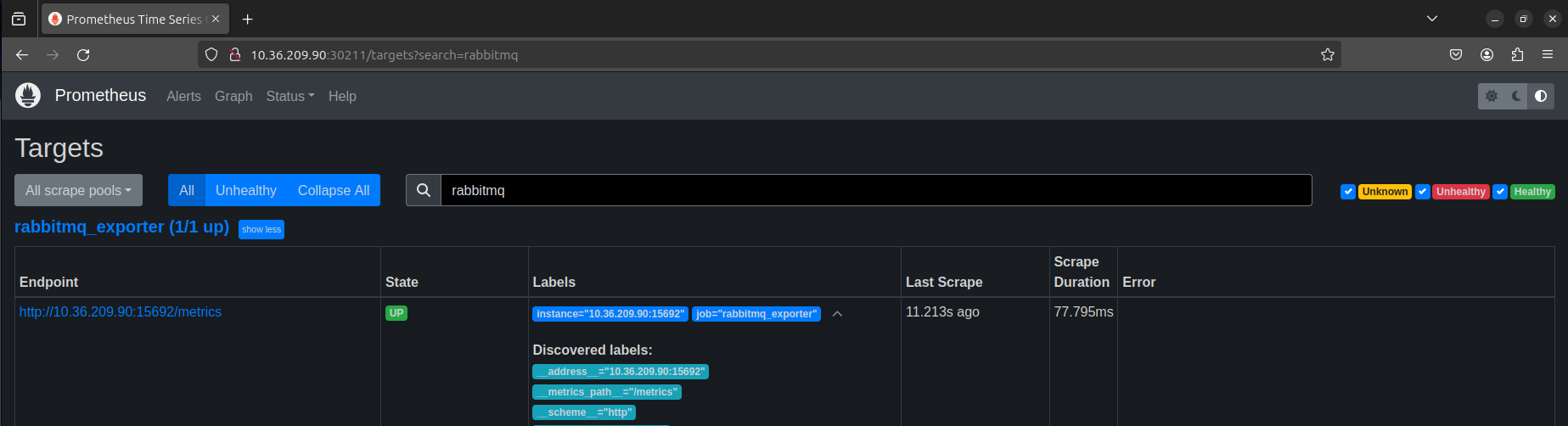
# dry run

helm upgrade -n monitoring-system -f values-kubeprom-stack.yaml stable . --dry-run=server

# upgrade

helm upgrade -n monitoring-system -f values-kubeprom-stack.yaml stable .

```

- Kiểm tra trạng thái exporter trong Status > Targets của Prometheus thấy Up là được:

- Tải dashboard JSON ở đây: https://grafana.com/grafana/dashboards/10991, https://grafana.com/grafana/dashboards/11350

- Trên Grafana, vào Dashboards > New > Import > Upload dashboard JSON file. Chọn template vừa tải, chọn Datasource là Prometheus (default)

## Cài đặt Elasticsearch exporter

- Tải chart Elasticsearch exporter về server:

```

cd /data-k8s/monitoring

helm fetch prometheus-community/prometheus-elasticsearch-exporter --version 5.4.0 --untar

cd prometheus-elasticsearch-exporter/

cp values.yaml values-es-exporter.yaml

```

- Sửa các giá trị sau trong values-es-exporter.yaml:

```

image:

# repository: quay.io/prometheuscommunity/elasticsearch-exporter # Pull image ở đây về, đánh tag và push lên harbor. Ở đây tag là v1.7.0

repository: atomi.digital.com/monitoring/elasticsearch-exporter

resources:

requests:

memory: 128Mi

limits:

memory: 512Mi

es:

uri: http://kube-logging.es-cluster.svc.cluster.local:9200

timeout: 120s

sslSkipVerify: true

```

- Cài đặt:

```

cd /data-k8s/monitoring-system/prometheus-elasticsearch-exporter

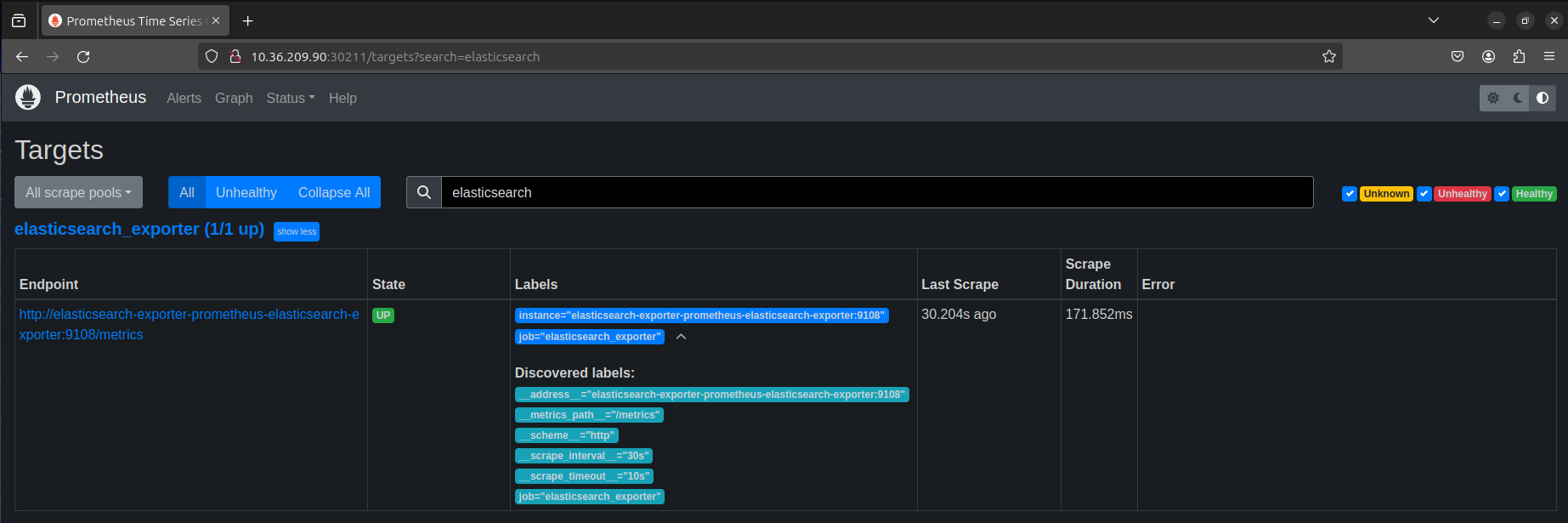
# dry run

helm install -n monitoring-system -f values-es-exporter.yaml elasticsearch-exporter . --dry-run=server

# install

helm install -n monitoring-system -f values-es-exporter.yaml elasticsearch-exporter .

```

- Kiểm tra trạng thái exporter trong Status > Targets của Prometheus thấy Up là được:

- Tải dashboard JSON ở đây: https://grafana.com/grafana/dashboards/14191.

- Trên Grafana, vào Dashboards > New > Import > Upload dashboard JSON file. Chọn template vừa tải, chọn Datasource là Prometheus (default)

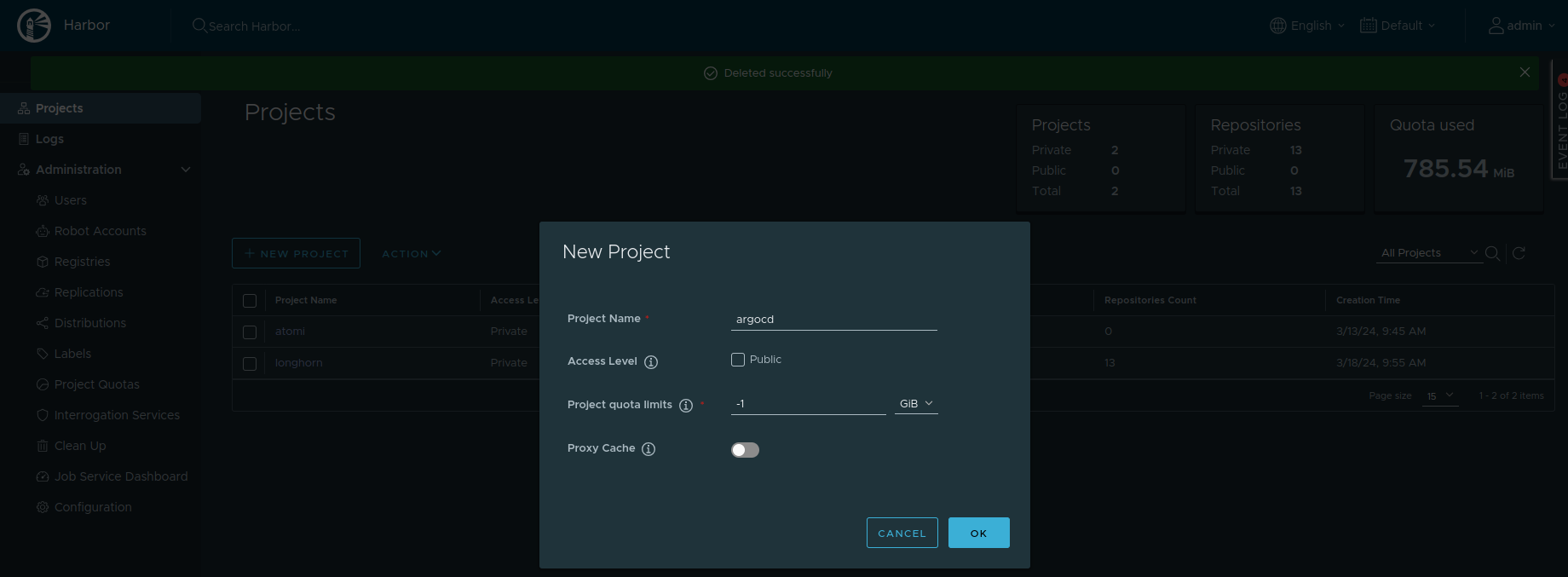
# VIII. ArgoCD

- Tạo namespace mới: kubectl create ns argocd

- Tải manifest argocd:

| cd /data-k8s mkdir argocd cd argocd wget https://raw.githubusercontent.com/argoproj/argo-cd/v2.10.3/manifests/install.yaml |
| --- |

- Vào Harbor, tạo một project mới tên là argocd như sau:



- Pull các image sau và đánh tag lại push lên Harbor để dùng:

| # pull images  docker pull quay.io/argoproj/argocd:v2.10.3 docker pull ghcr.io/dexidp/dex:v2.37.0 docker pull redis:7.0.14-alpine # tag images docker tag quay.io/argoproj/argocd:v2.10.3 atomi.digital.com/argocd/argocd:v2.10.3 docker tag ghcr.io/dexidp/dex:v2.37.0 atomi.digital.com/argocd/dex:v2.37.0 docker tag redis:7.0.14-alpine atomi.digital.com/argocd/redis:7.0.14-alpine # push images docker push atomi.digital.com/argocd/argocd:v2.10.3 docker push atomi.digital.com/argocd/dex:v2.37.0 docker push atomi.digital.com/argocd/redis:7.0.14-alpine |
| --- |

- Sửa file install.yaml và replace images …

- Sửa service:

| apiVersion: v1 kind: Service metadata:  labels:  app.kubernetes.io/component: server  app.kubernetes.io/name: argocd-server  app.kubernetes.io/part-of: argocd  name: argocd-server spec:  type: NodePort  ports:  - name: http  port: 80  protocol: TCP  targetPort: 8080  nodePort: 30003  - name: https  port: 443  protocol: TCP  targetPort: 8080  nodePort: 30004  selector:  app.kubernetes.io/name: argocd-server |
| --- |

- Cài đặt: kubectl apply -n argocd -f install.yaml

- Lấy password ban đầu để login: kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d

- Đổi password user admin:

| # Exec vào pod kubectl -n argocd exec -it pod/argocd-server-xxx -- bash # Chạy câu lệnh để sinh ra hash của password  argocd account bcrypt --password YOUR-PASSWORD-HERE #adminadmin # output $2a$10$9UhtLhYtlKssmkeCw0eJxepflTd5SxdtHuv0JVaLfvMie8uCwi1Te # Thoát shell. Sửa secret của argocd và thay hash vừa được tạo vào kubectl -n argocd patch secret argocd-secret \  -p '{"stringData": {  "admin.password": "$2a$10$9UhtLhYtlKssmkeCw0eJxepflTd5SxdtHuv0JVaLfvMie8uCwi1Te",  "admin.passwordMtime": "'$(date +%FT%T%Z)'"  }}' |
| --- |