Contents

[Cài đặt: 5](#_Toc100118285)

[kubectl 5](#_Toc100118286)

[bash-completion 5](#_Toc100118287)

[kubectl-convert 5](#_Toc100118288)

[minikube 6](#_Toc100118289)

[Helm 6](#_Toc100118290)

[Cài đặt từ Repo 6](#_Toc100118291)

[Kube Dashboard 11](#_Toc100118292)

[Tool hay: 11](#_Toc100118293)

[Kompose = Convert docker-compose to K8s 11](#_Toc100118294)

[THUẬT NGỮ: 11](#_Toc100118295)

[File YAML Mô tả cấu trúc 12](#_Toc100118296)

[Debug lỗi: 14](#_Toc100118297)

[POD 14](#_Toc100118298)

[GET pod 14](#_Toc100118299)

[Run Pod 15](#_Toc100118300)

[APPLY pod 15](#_Toc100118301)

[EXEC Để chạy command trong pod 15](#_Toc100118302)

[EXEC -it Truy cập vào Pod: 16](#_Toc100118303)

[Port-Froward 16](#_Toc100118304)

[DELETE pod 16](#_Toc100118305)

[LOG pod 16](#_Toc100118306)

[Describe pod 16](#_Toc100118307)

[LABEL 16](#_Toc100118308)

[NameSpace 17](#_Toc100118309)

[Tạo namespace 17](#_Toc100118310)

[Services 17](#_Toc100118311)

[VD1: ClusterIP: 17](#_Toc100118312)

[NodePort 18](#_Toc100118313)

[LoadBalancer 19](#_Toc100118314)

[Ingress: 20](#_Toc100118315)

[Services VD2 20](#_Toc100118316)

[Describe service 22](#_Toc100118317)

[Replicaset (rs) 23](#_Toc100118318)

[DELETE Replicaset 25](#_Toc100118319)

[DaemonSET 25](#_Toc100118320)

[JOB 25](#_Toc100118321)

[NETWORKING 26](#_Toc100118322)

[VOLUME 26](#_Toc100118323)

[emptyDir 26](#_Toc100118324)

[hostPath 27](#_Toc100118325)

[Storage Class / PV / PVC 28](#_Toc100118326)

[Storage 28](#_Toc100118327)

[VD1: Storage Class as NFS a Trung 30](#_Toc100118328)

[Recycling PersistentVolume 30](#_Toc100118329)

[Readiness & Liveness 31](#_Toc100118330)

[CONFIG MAP – SECRET 31](#_Toc100118331)

[ConfigMap 31](#_Toc100118332)

[ENV alone 31](#_Toc100118333)

[Create config map 32](#_Toc100118334)

[Configmap as ENV 32](#_Toc100118335)

[Configmap as Volume 33](#_Toc100118336)

[Secret 34](#_Toc100118337)

[Secret as ENV 34](#_Toc100118338)

[Secret as file in folder 35](#_Toc100118339)

[Secret as configfile 36](#_Toc100118340)

[DEPLOYMENT 37](#_Toc100118341)

[VD\_1 về đổi image trong deployment 39](#_Toc100118342)

[VD2 Zero downtime deployment: 39](#_Toc100118343)

[Statefull Set 40](#_Toc100118344)

[RBAC / API 42](#_Toc100118345)

[Role binding / service account 42](#_Toc100118346)

[API 43](#_Toc100118347)

[Taints and tolerations 44](#_Toc100118348)

[Taints 44](#_Toc100118349)

[podAntiAffinity 45](#_Toc100118350)

[TIP-TRICK 45](#_Toc100118351)

[Install telnet in docker apk 45](#_Toc100118352)

[Sample Prod 45](#_Toc100118353)

[K8s Registry: 46](#_Toc100118354)

[Ingress nginx: 50](#_Toc100118355)

[Nfs provisioning 51](#_Toc100118356)

[Debug image 53](#_Toc100118357)

[**ARGO-CD -- ARGO-CD -- ARGO-CD-- ARGO-CD -- ARGO-CD --** 53](#_Toc100118358)

[CÀI ĐẶT 53](#_Toc100118359)

[3. Hello World Workflow 54](#_Toc100118360)

[TEMPLATE TRONG ARGO 54](#_Toc100118361)

[Container template 54](#_Toc100118362)

[Script Template 55](#_Toc100118363)

[Resource Template 56](#_Toc100118364)

[Suspend 57](#_Toc100118365)

[Template INVOCATORS 57](#_Toc100118366)

[STEP 58](#_Toc100118367)

[Serial Step (Step nối tiếp nhau) 58](#_Toc100118368)

[Step Parabel (Step song song) 59](#_Toc100118369)

[Suspend Step Template 60](#_Toc100118370)

[DAG 62](#_Toc100118371)

[Bài tập 1 về Step và Dag 64](#_Toc100118372)

[WORKFLOW FUNCTION ARGO 66](#_Toc100118373)

[1.MinIO 66](#_Toc100118374)

[2.Cài đặt Argo-Cli 67](#_Toc100118375)

[3.InputParameter: 67](#_Toc100118376)

[4.Scripts Result 71](#_Toc100118377)

[5.Output parameter 74](#_Toc100118378)

[6.Output Parameter File 77](#_Toc100118379)

[7. Artifact 79](#_Toc100118380)

[8. Secrets as environment variables 82](#_Toc100118381)

[9. Secrets as mounted volumes 82](#_Toc100118382)

[10. Loops 82](#_Toc100118383)

[11. Loops with sets 84](#_Toc100118384)

[12. Loops with sets as input parameters 86](#_Toc100118385)

[13. Dynamic Loops 89](#_Toc100118386)

[14. Conditionals 91](#_Toc100118387)

[15. Depends 93](#_Toc100118388)

[16. Depends theorie 94](#_Toc100118389)

[17. Retry strategy 94](#_Toc100118390)

[18. Recursion 94](#_Toc100118391)

[19. Exercise 2 - task introduction 94](#_Toc100118392)

[20. Exercise 2 - solution 94](#_Toc100118393)

---

# Cài đặt:

git clone <https://github.com/luksa/kubernetes-in-action.git>

### kubectl

cd /opt/; curl -LO [https://dl.k8s.io/release/**$(**curl -L -s https://dl.k8s.io/release/stable.txt**)**/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

chmod +x kubectl

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

### bash-completion

#apt-get install bash-completion

sudo yum install bash-completion -y

echo 'source <(kubectl completion bash)' >>~/.bashrc

echo 'alias k=kubectl' >>~/.bashrc

echo 'complete -F \_\_start\_kubectl k' >>~/.bashrc

### kubectl-convert

curl -LO https://dl.k8s.io/release/**$(**curl -L -s [https://dl.k8s.io/release/stable.txt**)**/bin/linux/amd64/kubectl-convert](https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert)

chmod +x kubectl-convert

sudo install -o root -g root -m 0755 kubectl-convert /usr/local/bin/kubectl-convert

### minikube

(yêu cầu phải cài docker hoặc podman: curl -fsSL https://get.docker.com/ | sh )

curl -LO <https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64>

chmod +x minikube-linux-amd64

mv minikube-linux-amd64 minikube

sudo install ./minikube /usr/local/bin/minikube

su tuanda

minikube start

#minikube stop

Kiểm tra minikube và clusster

#kubectl cluster-info

#

### Helm

wget <https://get.helm.sh/helm-v3.8.1-linux-amd64.tar.gz>

(hoặc: curl -sSL <https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3> | bash )

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

helm search repo stable

helm repo update

helm search repo nginx

helm install my-nginx bitnami/nginx

helm create my-project

## Cài đặt từ Repo

<https://bikramat.medium.com/set-up-a-kubernetes-cluster-with-kubeadm-508db74028ce>

<https://phoenixnap.com/kb/how-to-install-kubernetes-on-centos>

<https://phoenixnap.com/kb/how-to-install-kubernetes-on-a-bare-metal-server>

<https://xuanthulab.net/gioi-thieu-va-cai-dat-kubernetes-cluster.html>

**B1: Đặt Hostname (run on Master + Worker node)**

#hostnamectl set-hostname master-node

#hostnamectl set-hostname worker-node1

#hostnamectl set-hostname worker-node2

# cat << EOF >> /etc/hosts

192.168.88.12 master-node

192.168.88.13 worker-node1

192.168.88.14 worker-node2

EOF

**B2: Setting cơ bản (run on Master + Worker node)**

Disable Swap trên master và worker

# Tat swap

sed -i '/swap/d' /etc/fstab

swapoff -a

cat <<EOF > /etc/sysctl.d/k8s.conf

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

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

EOF

sysctl --system

**Disable Selinux: (run on Master + Worker node)**

sudo setenforce 0

sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

sudo sed -i 's/^SELINUX= permissive$/SELINUX=disabled/' /etc/selinux/config

**B3: Cài docker-ce (run on Master + Worker node)**

yum install epel-release -y ; curl -fsSL https://get.docker.com/ | sh

usermod -aG docker $(whoami)

## Create /etc/docker directory.

mkdir /etc/docker

## Thay đổi cgroup theo từng loại OS (Centos/Ubuntu/...)

[tuanda@master-node ~]$ sudo docker info | grep -i cgroup

Cgroup Driver: systemd

Cgroup Version: 1

# Setup daemon theo cgroup ở trên.

cat > /etc/docker/daemon.json <<EOF

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2",

"storage-opts": [

"overlay2.override\_kernel\_check=true"

]

}

EOF

#Chú ý cần có dòng này

mkdir -p /etc/systemd/system/docker.service.d

# Restart Docker

systemctl enable docker.service

systemctl daemon-reload

systemctl restart docker

**Cài đặt kubelet/kubeadm/kubectl** **(run on Master + Worker node)**

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

yum install -y kubelet kubeadm kubectl

service kubelet start

systemctl enable kubelet.service

telnet localhost 10248

Mở port master node

sudo firewall-cmd --permanent --add-port=6443/tcp

sudo firewall-cmd --permanent --add-port=2379-2380/tcp

sudo firewall-cmd --permanent --add-port=10250/tcp

sudo firewall-cmd --permanent --add-port=10251/tcp

sudo firewall-cmd --permanent --add-port=10252/tcp

sudo firewall-cmd --permanent --add-port=10255/tcp

sudo firewall-cmd --reload

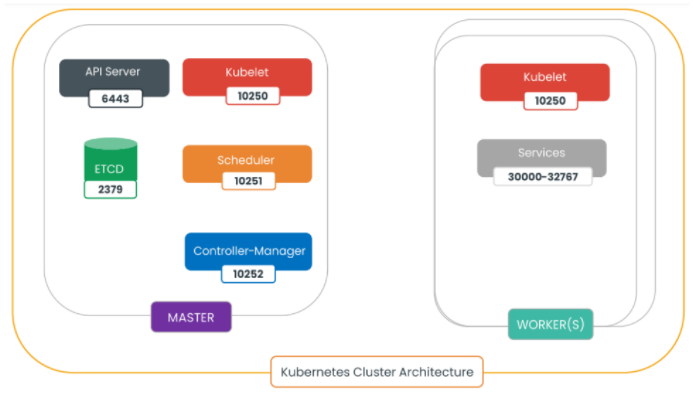
Mở port worker node

sudo firewall-cmd --permanent --add-port=10251/tcp

sudo firewall-cmd --permanent --add-port=10255/tcp

firewall-cmd –reload

URL Port cần mở ở đây <https://kubernetes.io/docs/reference/ports-and-protocols/>



Bước 4 : Khởi tạo Master Node **(run on Master node)**

kubeadm init --apiserver-advertise-address=192.168.88.12 --pod-network-cidr=10.244.0.0/16

(trường hợp tạo lỗi, ta có thể gõ lệnh #kubeadm reset)

su - tuanda

mkdir -p $HOME/.kube

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

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

Lệnh in lại command join: kubeadm token create --print-join-command HOẶC <https://ystatit.medium.com/regenerate-kubernetes-join-command-to-join-work-node-7eeb5d1f5787>

Bước 5: Pod Network trên Master-Node **(run on Master node)**

Ta có thể dùng nhiều addon như Flanel, cacilo, weaver. Và network chỉ apply được sau khi init master-node.

**Hoặc Calico**

Tham khảo cách cài calico 50node, 100node, etcd: <https://projectcalico.docs.tigera.io/getting-started/kubernetes/self-managed-onprem/onpremises>

# curl https://projectcalico.docs.tigera.io/manifests/calico.yaml -O

# kubectl apply -f calico.yaml

Bước 6: Join Worker node: **(run on Worker node)**

kubeadm join 192.168.88.12:6443 --token h46n34.uq80d4pro1qjyvk0 --discovery-token-ca-cert-hash xxxxxxxxxxx

Bước 7: Kiểm tra **(run on Master node)**

[tuanda@master-node ~]$ kubectl get node

[tuanda@master-node ~]$ kubectl cluster-info

[tuanda@master-node ~]$ kubectl get pod -A

Other: remove node:

kubectl drain Ten\_Node

## Kube Dashboard

Hướng dẫn: <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

Hướng dẫn: <https://www.replex.io/blog/how-to-install-access-and-add-heapster-metrics-to-the-kubernetes-dashboard>

curl [https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml -o kubernetes-dashboard.yaml](https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml%20-o%20kubernetes-dashboard.yaml)

cur <https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml>

kubectl apply -f

[tuanda@master-node ~]$ kubectl -n kubernetes-dashboard get pod

# Tool hay:

## Kompose = Convert docker-compose to K8s

<https://kompose.io/> (cài đặt)

$ kompose convert -f docker-compose.yaml

# THUẬT NGỮ:

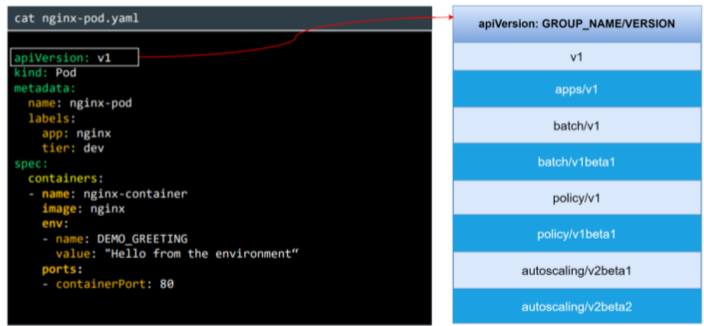
Replicaset: tạo ra multi-pod chạy cùng trên 1 image

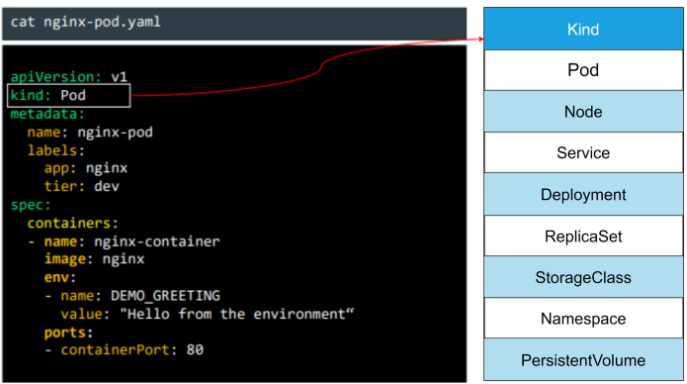
Depoyment: Quản lý các replicaset, phục vụ cho việc thay đổi image hoặc cấu hình.

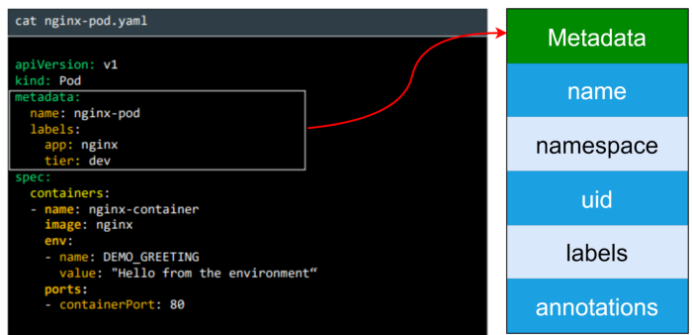
Statefullset: Khi tạo statefullset, Pod bị xóa đi thì pod mớ được sinh ra sẽ kế thừa network + volume từ pod cũ. mặc định sẽ tự tạo thêm PVC để giữ PV luôn cố định, dữ liệu sẽ không thay đổi, thích hợp cho sử dụng DB

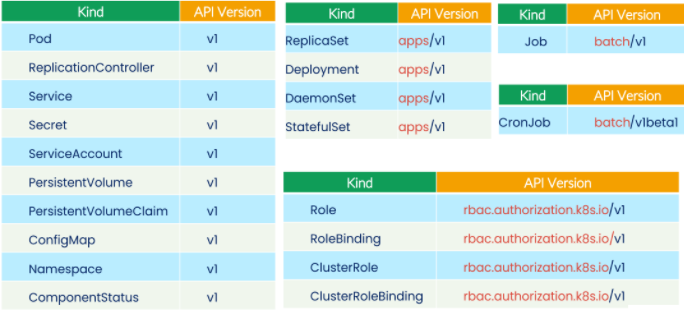
DaemonSet:

# File YAML Mô tả cấu trúc









# Debug lỗi:

kubectl describe pod

kubectl logs pod

# POD

Mỗi 1 microserver sẽ đặt trên 1 pod.

Để kiểm tra pod, ta sử dụng những lệnh sau:

### GET pod

kubectl get all

kubectl get pods

kubectl get pod --show-labels

kubectl explain pods

kubectl get pod kubia-manual -o yaml (hoặc json)

kubectl get all -o wide (hiển thị rộng hơn)

### Run Pod

# kubectl run kubia --image=luksa/kubia --port=8080

# kubectl run -i --tty busybox --image=busybox --restart=Never –- sh

# kubectl run busybox --image=busybox --restart=Never -o yaml --dry-run=client -- /bin/sh -c 'echo hello;sleep 3600'

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: debugpod

  name: debugpod

spec:

  containers:

  - image: nicolaka/netshoot

    name: debugpod

    command: ["/bin/bash"]

    args: ["-c", "sleep 100000"]

### APPLY pod

# kubectl apply -f kubia-manual.yaml

# kubectl get all

### EXEC Để chạy command trong pod

kubectl exec [POD] -- [COMMAND]

# **kubectl exec kubia-manual -- ls**

bin

dev

etc

### EXEC -it Truy cập vào Pod:

# **kubectl -it exec webapp -- sh**

/ # whoami

root

### Port-Froward

tuanda@localhost Chapter03]$ **kubectl port-forward kubia-manual 8888:8080**

### DELETE pod

# **kubectl delete pod nginx**

# **kubectl delete po --all**

# kubectl delete po -l creation\_method=manual (xóa pod có chỉ định label ở phần dưới)

### LOG pod

# kubectl logs -f kubia-manual

### Describe pod

# kubectl describe pod webapp

# kubectl describe pod nginx

# LABEL

[tuanda@localhost Chapter03]$ kubectl get node --show-labels

[tuanda@localhost Chapter03]$ kubectl get pod --show-labels

Add thêm labels:

[tuanda@localhost Chapter04]$ kubectl label pod kubia-gg5t5 type=special

[tuanda@localhost Chapter04]$ kubectl get pod --show-labels

NAME READY STATUS RESTARTS AGE LABELS

kubia-gg5t5 1/1 Running 0 9h app=kubia,type=special

kubia-sbsjw 1/1 Running 0 71m app=kubia

Thay label (tách pod ra khỏi replicaset/control)

[tuanda@localhost Chapter04]$ kubectl label pod kubia-gg5t5 app=ahihi –overwrite

(Việc add, thêm label có thể làm được với cả node)

# NameSpace

# kubectl get ns

[tuanda@localhost ]$ kubectl get pod -n default (mặc định, khi tạo bạn không chỉ rõ ns, thì pod sẽ nằm ở ns default)

# kubectl get pod -n kube-system

# kubectl get all -n kube-system

# kubectl get all --all-namespaces (lấy toàn bộ thông tin của tất cả NS)

# k get all -A

### Tạo namespace

Có thể tạo từ yaml hoặc command:

# kubectl create namespace tuanda -o yaml --dry-run=client

# Services

## VD1: ClusterIP:

Đây là loại truy cập nội bộ các pod với nhau, không truy cập từ bên ngoài vào đc.

[tuanda@localhost Chapter05]$ cat kubia-svc.yaml

apiVersion: v1

kind: Service

metadata:

name: kubia

spec:

ports:

- port: 80

targetPort: 8080

selector:

app: kubia

[tuanda@localhost Chapter05]$ cat ../Chapter04/kubia-replicaset.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: kubia

spec:

replicas: 3

selector:

matchLabels:

app: kubia

template:

metadata:

labels:

app: kubia

spec:

containers:

- name: kubia

image: luksa/kubia

Ta có thể truy cập vào pod. Và thử lệnh curl

[tuanda@localhost Chapter05]$ kubectl apply -f kubia-svc.yaml

[tuanda@localhost Chapter05]$ kubectl exec kubia-8z2lv -- curl -s kubia

[tuanda@localhost Chapter05]$ k get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 3h59m

kubia ClusterIP 10.103.135.99 <none> 80/TCP 64s

(ở đây, kubia chính là ip của svc được phân giải ra IP: 10.103.135.99)

[tuanda@localhost Chapter05]$ kubectl exec -it kubia-8z2lv -- bash

root@kubia-8z2lv:/# ping kubia

PING kubia.default.svc.cluster.local (10.103.135.99): 56 data bytes

## NodePort

Nodeport có thể cho client bên ngoài gọi đc. Bằng IP của các node cluster. Nodeport có range từ 30000 đến 32767

[tuanda@localhost Chapter05]$ cat kubia-svc-nodeport.yaml

apiVersion: v1

kind: Service

metadata:

name: kubia-nodeport

spec:

type: NodePort

ports:

- port: 80

targetPort: 8080

nodePort: 30123

selector:

app: kubia

[tuanda@localhost Chapter05]$ kubectl get all -o wide

[tuanda@localhost Chapter05]$ minikube ip

192.168.49.2

[tuanda@localhost Chapter05]$ curl 192.168.49.2:30123

You've hit kubia-g586k

## LoadBalancer

Hỗ trợ cả bên ngoài client và trong pod đều gọi vào đc.

# kubectl expose deploy/nginx --type=LoadBalancer --port=80

[tuanda@localhost Chapter05]$ cat kubia-svc-loadbalancer.yaml

apiVersion: v1

kind: Service

metadata:

name: kubia-loadbalancer

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 8080

selector:

app: kubia

[tuanda@localhost Chapter05]$ kubectl apply -f kubia-svc-loadbalancer.yaml

## Ingress:

[tuanda@localhost Chapter05]$ minikube get addon list

[tuanda@localhost Chapter05]$ minikube addons enable ingress

???????????

# Services VD2

Port-Forward

Chạy pod nginx ở trên. Sau đó gõ

[tuanda@localhost ~]# kubectl port-forward nginx 8080:80

[tuanda@localhost ~]# curl localhost:8080

<!DOCTYPE html

Tạo service

# cat webapp-service.yaml

apiVersion: v1

kind: Service

metadata:

  name: fleetman-webapp

spec:

  selector:

    app: webapp

    release: "0-5"

  ports:

    - name: http

      port: 80

      nodePort: 30080

  type: NodePort

Sửa thêm pods.yaml

apiVersion: v1

kind: Pod

metadata:

  name: webapp

  labels:

    app: webapp

    release: "0"

spec:

  containers:

  - name: webapp

    image: richardchesterwood/k8s-fleetman-webapp-angular:release0

---

apiVersion: v1

kind: Pod

metadata:

  name: webapp-release-0-5

  labels:

    app: webapp

    release: "0-5"

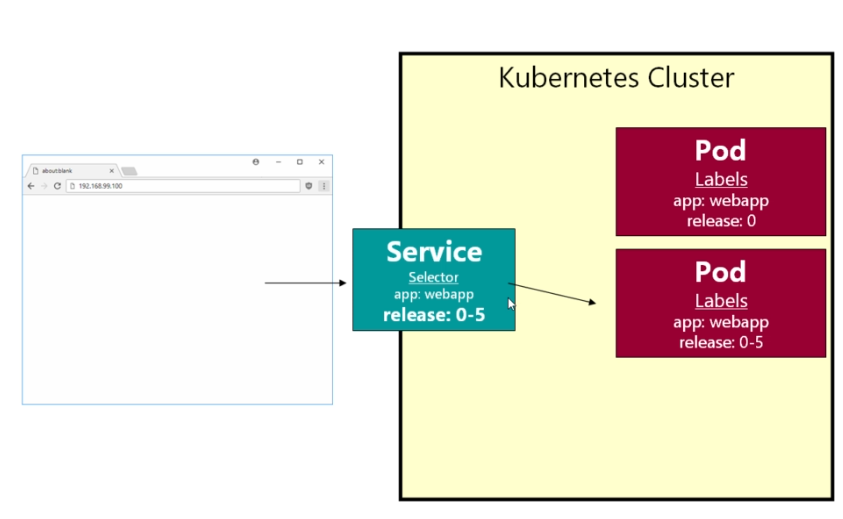
spec:

  containers:

  - name: webapp

    image: richardchesterwood/k8s-fleetman-webapp-angular:release0-5

Mô hình sẽ như sau:



Ta có thể luân chuyển service chọn pod nào bằng cách thay đổi

  selector:

    app: webapp

    release: "0-5"

# kubectl get pods --show-labels

# minikube ip

192.168.49.2

# kubectl get all

NAME READY STATUS RESTARTS AGE

pod/nginx 1/1 Running 0 7h35m

pod/webapp 1/1 Running 0 7h52m

pod/webapp-release-0-5 1/1 Running 0 11m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/fleetman-webapp NodePort 10.101.82.133 <none> 80:30080/TCP 15m

service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 8h

Ta sẽ thực hiện truy cập vào <http://192.168.49.2:30080>

### Describe service

# kubectl describe svc fleetman-webapp

Name: fleetman-webapp

Namespace: default

Labels: <none>

Annotations: <none>

Selector: app=webapp,release=0-5

Type: NodePort

IP Family Policy: SingleStack

IP Families: IPv4

IP: 10.101.82.133

IPs: 10.101.82.133

Port: http 80/TCP

TargetPort: 80/TCP

NodePort: http 30080/TCP

Endpoints: 172.17.0.7:80

Session Affinity: None

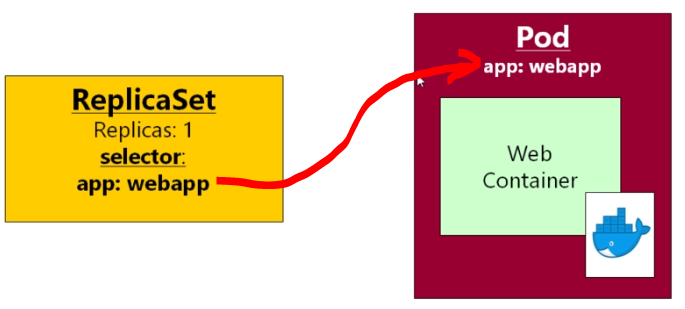
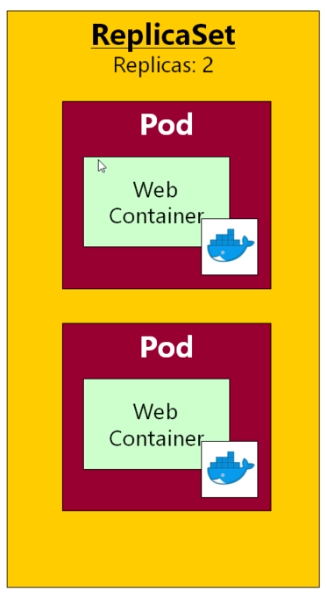
External Traffic Policy: Cluster

Events: <none>

Từ lệnh desscribe trên, ta hiểu là service fleetman-webapp được gắn với pod= webapp và release = 0-5

# Replicaset (rs)

Ta thực hiện chỉnh sửa file pods.yaml, còn services.yaml vẫn giữ nguyên

 >>>>chuyển sang>>>>>>>

[tuanda@localhost Chapter04]$ cat kubia-replicaset.yaml

apiVersion: apps/v1beta2

kind: ReplicaSet

metadata:

  name: kubia

spec:

  replicas: 3

  selector:

    matchLabels:

      app: kubia

  template:

    metadata:

      labels:

        app: kubia

    spec:

      containers:

      - name: kubia

        image: luksa/kubia

[tuanda@localhost Chapter04]$ kubectl describe rs kubia

# kubectl get rs

# kubectl edit rs Tên\_RS

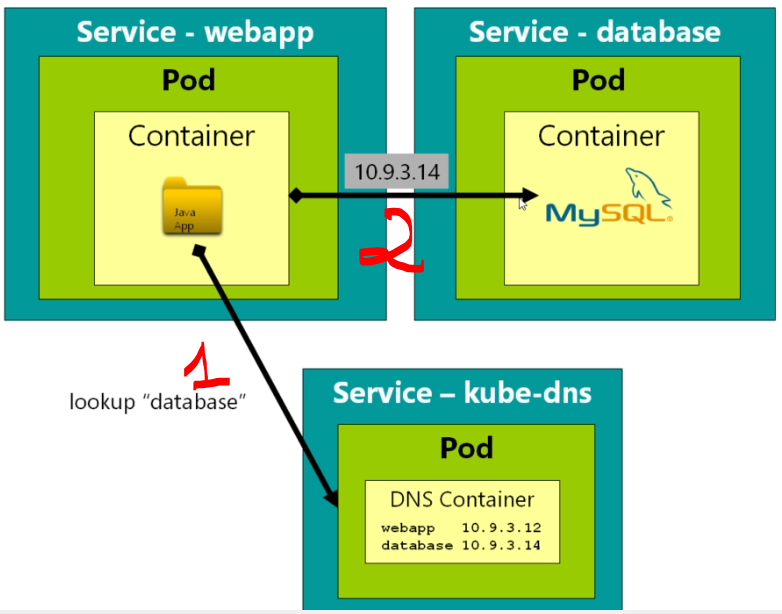
### DELETE Replicaset

# kubectl delete rs webapp

# DaemonSET

# JOB

# NETWORKING



Ta sẽ thực hiện trỏ bằng dns, sau đó kube-dns sẽ phân giải ra IP cần tìm.

Kiểm tra dns

# kubectl get svc kube-dns -n kube-system

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP,9153/TCP 24h

# kubectl describe svc kube-dns -n kube-system

# VOLUME

Các loại Volume : <https://kubernetes.io/docs/concepts/storage/volumes/>

Các loại PV: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#types-of-persistent-volumes>

## emptyDir

[tuanda@localhost Chapter06]$ cat fortune-pod.yaml

apiVersion: v1

kind: Pod

metadata:

  name: fortune

spec:

  containers:

  - image: luksa/fortune

    name: html-generator

    volumeMounts:

    - name: html

      mountPath: /var/htdocs

  - image: nginx:alpine

    name: web-server

    volumeMounts:

    - name: html

      mountPath: /usr/share/nginx/html

      readOnly: true

    ports:

    - containerPort: 80

      protocol: TCP

  volumes:

  - name: html

    emptyDir: {}

[tuanda@localhost Chapter06]$ kubectl port-forward fortune 8080:80

[tuanda@localhost Chapter06]$ curl localhost:8080

## hostPath

VD1

Lưu vào phân vùng của node/minikube

[tuanda@localhost Chapter06]$ cat mongodb-pod-hostpath.yaml

apiVersion: v1

kind: Pod

metadata:

name: mongodb

spec:

containers:

- image: mongo

name: mongodb

volumeMounts:

- name: mongodb-data

mountPath: /data/db

ports:

- containerPort: 27017

protocol: TCP

volumes:

- name: mongodb-data

hostPath:

path: /tmp/mongodb

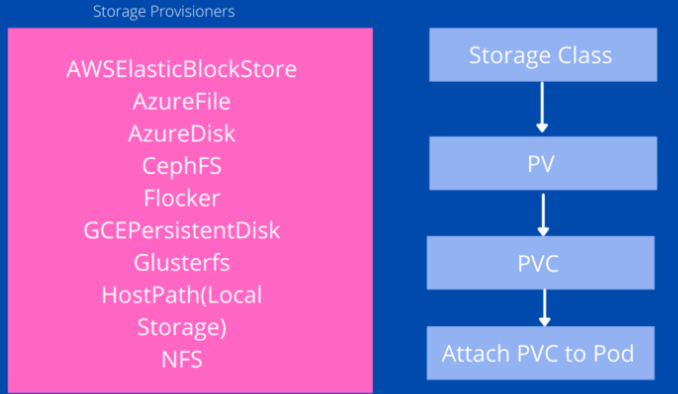
# Storage Class / PV / PVC

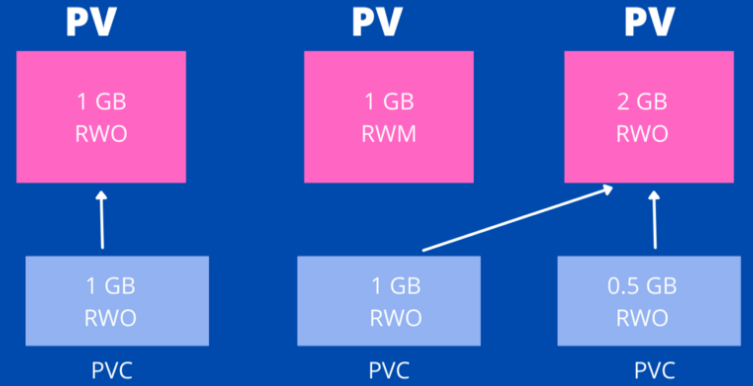
## Storage

<https://kubernetes.io/docs/concepts/storage/storage-classes/>

<https://medium.com/codex/kubernetes-persistent-volume-explained-fb27df29c393>

Kiến trúc:





Các loại storage có thể lên trang chủ để tìm.

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

  name: local-storage

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer

allowVolumeExpansion: true

reclaimPolicy: Delete

### VD1: Storage Class as NFS a Trung

Đọc thư mục : D:\Dropbox\Config server\k8s\volume\nfs-client (của anh Trung share)

helm repo add nfs-subdir-external-provisioner \

  https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/

helm repo update

helm install nfs-provisioner-2 \

  nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \

  --set nfs.server=192.168.135.2 \

  --set nfs.path=/data/nfs \

  --set storageClass.name=nfs-provisioner-2 \

  --set storageClass.onDelete=retain \

  --set storageClass.accessModes=ReadWriteMany

helm install nfs-provisioner-3 \

  nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \

  --set nfs.server=192.168.135.3 \

  --set nfs.path=/data/nfs \

  --set storageClass.name=nfs-provisioner-3 \

  --set storageClass.onDelete=retain \

  --set storageClass.accessModes=ReadWriteMany

## Recycling PersistentVolume

Có 3 loại:

* Retain: khi xóa PVC thì PV vẫn còn- dữ liệu trong PV không bị xóa.
* Recycle: khi xóa PVC thì PV vẫn còn, nhưng dữ liệu trong PV sẽ được xóa đi để tái sử dụng
* Delete: khi xóa PVC thì PV sẽ bị xóa luôn.

Subpath khi sử dụng chung 1 PVC khá hay <https://kubernetes.io/docs/concepts/storage/volumes/#using-subpath>

# Readiness & Liveness

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

# CONFIG MAP – SECRET

## ConfigMap

<https://kubernetes.io/docs/concepts/configuration/configmap/>

### ENV alone

apiVersion: v1

kind: Pod

metadata:

  name: fortune-env

spec:

  containers:

  - image: luksa/fortune:env

    env:

    - name: INTERVAL

      value: "30"

    - name: TUANDA

      value: "kaka"

    name: html-generator

    volumeMounts:

    - name: html

      mountPath: /var/htdocs

  - image: nginx:alpine

    name: web-server

    volumeMounts:

    - name: html

      mountPath: /usr/share/nginx/html

      readOnly: true

    ports:

    - containerPort: 80

      protocol: TCP

  volumes:

  - name: html

    emptyDir: {}

[tuanda@localhost Chapter07]$ kubectl exec -it fortune-env -- printenv

INTERVAL=30

TUANDA=kaka

### Create config map

**Tạo config map từ command-line**

[tuanda@localhost Chapter07]$ kubectl create configmap fortune-config --from-literal=sleep-interval=25

**Hoặc từ file config hoặc yaml, json (thích hợp với import file dài, khó)**

[tuanda@localhost configmap-files]$ kubectl create configmap tuanda-config --from-file=customkey=my-nginx-config.conf

### Configmap as ENV

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>

apiVersion: v1

kind: Pod

metadata:

  name: fortune-env-from-configmap

spec:

  containers:

  - image: luksa/fortune:env

    env:

    - name: INTERVAL

      valueFrom:

        configMapKeyRef:

          name: fortune-config

          key: sleep-interval

    name: html-generator

    volumeMounts:

    - name: html

      mountPath: /var/htdocs

  - image: nginx:alpine

    name: web-server

    volumeMounts:

    - name: html

      mountPath: /usr/share/nginx/html

      readOnly: true

    ports:

    - containerPort: 80

      protocol: TCP

  volumes:

  - name: html

    emptyDir: {}

khi vào pod, ta sẽ thấy container có biến môi trường là : INTERVAL=25

### Configmap as Volume

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>

apiVersion: v1

kind: Pod

metadata:

  name: dapi-test-pod

spec:

  containers:

    - name: test-container

      image: k8s.gcr.io/busybox

      command: [ "/bin/sh", "-c", "ls /etc/config/" ]

      volumeMounts:

      - name: config-volume

        mountPath: /etc/config

  volumes:

    - name: config-volume

      configMap:

        name: special-config

  restartPolicy: Never

## Secret

<https://kubernetes.io/docs/concepts/configuration/secret/>

Ngoài generic, Secret hỗ trợ các loại:

|  |  |
| --- | --- |
| Opaque | arbitrary user-defined data |
| kubernetes.io/service-account-token | service account token |
| kubernetes.io/dockercfg | serialized ~/.dockercfg file |
| kubernetes.io/dockerconfigjson | serialized ~/.docker/config.json file |
| kubernetes.io/basic-auth | credentials for basic authentication |
| kubernetes.io/ssh-auth | credentials for SSH authentication |
| kubernetes.io/tls | data for a TLS client or server |
| bootstrap.kubernetes.io/token | bootstrap token data |

Secret được dùng cho file mout vào file trong pod, env cho pod.

Tạo secret bằng command line

# kubectl create secret generic prod-db-secret --from-literal=username=produser --from-literal=password=Y4nys7f11

VD về Opaque

**apiVersion**: v1

**kind**: Secret

**metadata**:

**name**: mysecret

**type**: Opaque

**data**:

**USER\_NAME**: YWRtaW4=

**PASSWORD**: MWYyZDFlMmU2N2Rm

### Secret as ENV

apiVersion: v1

kind: Pod

metadata:

  name: secret-env-pod

spec:

  containers:

  - name: mycontainer

    image: redis

    env:

      - name: SECRET\_USERNAME

        valueFrom:

          secretKeyRef:

            name: mysecret

            key: username

            optional: false

      - name: SECRET\_PASSWORD

        valueFrom:

          secretKeyRef:

            name: mysecret

            key: password

            optional: false

  restartPolicy: Never

### Secret as file in folder

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: mypod

**image**: redis

**volumeMounts**:

- **name**: foo

**mountPath**: "/etc/foo"

**readOnly**: **true**

**volumes**:

- **name**: foo

**secret**:

**secretName**: mysecret

### Secret as configfile

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: mypod

**image**: redis

**volumeMounts**:

- **name**: foo

**mountPath**: "/etc/foo"

**readOnly**: **true**

**volumes**:

- **name**: foo

**secret**:

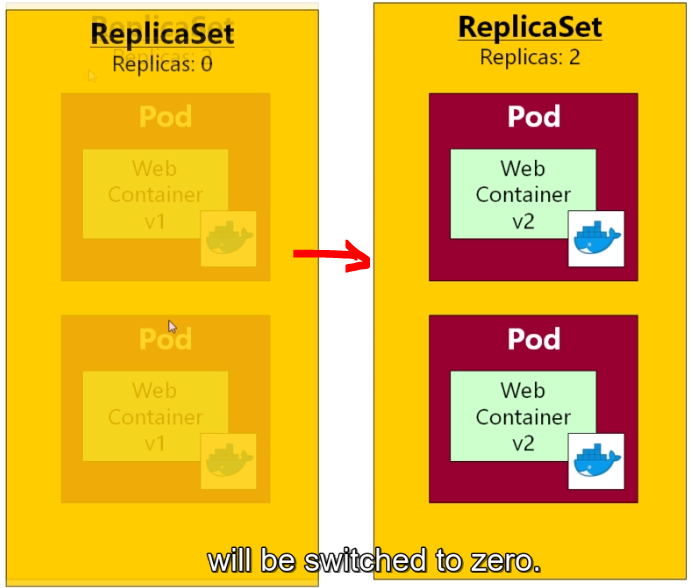
**secretName**: mysecret

**items**:

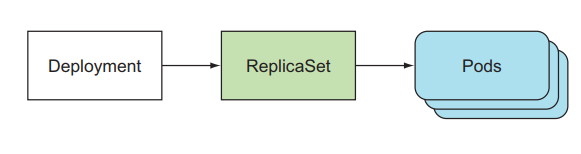
- **key**: username

**path**: my-group/my-username

# DEPLOYMENT



Để chuyển Image, Ta chuyển kind từ **Replicaset > Deployment**



**Một deployment sẽ có dạng như sau:**

[tuanda@localhost Chapter09]$ cat kubia-deployment-v1.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: kubia

spec:

  replicas: 3

  template:

    metadata:

      name: kubia

      labels:

        app: kubia

    spec:

      containers:

      - image: luksa/kubia:v1

        name: nodejs

  selector:

    matchLabels:

      app: kubia

---

apiVersion: v1

kind: Service

metadata:

  name: kubia

spec:

  type: LoadBalancer

  selector:

    app: kubia

  ports:

  - port: 80

    targetPort: 8080

Để đổi trực tiếp image mới ta có thể làm như sau:

[tuanda@localhost Chapter09]$ kubectl edit deployments.apps kubia

Hoặc

[tuanda@localhost Chapter09]$ kubectl set image deployment kubia nodejs=luksa/kubia:v2 (hoặc kubia:v3, v4)

Để khôi phục lại version trước đó, ta có các lệnh sau:

[tuanda@localhost Chapter09]$ kubectl rollout undo deployment kubia

Để theo dõi real-time undo đang làm gì, ta có lệnh status sau

[tuanda@localhost Chapter09]$ kubectl rollout status deployment kubia

Để show các version rollout, ta có lệnh sau:

[tuanda@localhost Chapter09]$ kubectl rollout history deployment kubia

Để đổi về 1 version history có chỉ định

[tuanda@localhost Chapter09]$ kubectl rollout undo deployment kubia --to-revision=4

### VD\_1 về đổi image trong deployment

apiVersion: apps/v1

kind: Deployment

metadata:

  name: webapp

spec:

  # minReadySeconds: 30

  selector:

    matchLabels:

      app: webapp

  replicas: 2

  template: # template for the pods

    metadata:

      labels:

        app: webapp

    spec:

      containers:

      - name: webapp

        image: richardchesterwood/k8s-fleetman-webapp-angular:release0-5

Muốn chuyển code. Ta chỉ cần chuyển image từ 0 lên 0.5

image: richardchesterwood/k8s-fleetman-webapp-angular:release0

Thành

image: richardchesterwood/k8s-fleetman-webapp-angular:release0-5

Sau đó chạy apply lại: $ kubectl apply -f pods.yaml

### VD2 Zero downtime deployment:

# Statefull Set

Khi tạo statefullset, mặc định sẽ tự tạo thêm PVC để giữ PV luôn cố định, dữ liệu sẽ không thay đổi, thích hợp cho sử dụng DB

[tuanda@localhost Chapter10]$ cat persistent-volumes-hostpath.yaml

kind: List

apiVersion: v1

items:

- apiVersion: v1

  kind: PersistentVolume

  metadata:

    name: pv-a

  spec:

    capacity:

      storage: 1Mi

    accessModes:

      - ReadWriteOnce

    persistentVolumeReclaimPolicy: Recycle

    hostPath:

      path: /tmp/pv-a

- apiVersion: v1

  kind: PersistentVolume

  metadata:

    name: pv-b

  spec:

    capacity:

      storage: 1Mi

    accessModes:

      - ReadWriteOnce

    persistentVolumeReclaimPolicy: Recycle

    hostPath:

      path: /tmp/pv-b

- apiVersion: v1

  kind: PersistentVolume

  metadata:

    name: pv-c

  spec:

    capacity:

      storage: 1Mi

    accessModes:

      - ReadWriteOnce

    persistentVolumeReclaimPolicy: Recycle

    hostPath:

      path: /tmp/pv-c

[tuanda@localhost Chapter10]$ cat kubia-statefulset.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

  name: kubia

spec:

  serviceName: kubia

  replicas: 2

  selector:

    matchLabels:

      app: kubia # has to match .spec.template.metadata.labels

  template:

    metadata:

      labels:

        app: kubia

    spec:

      containers:

      - name: kubia

        image: luksa/kubia-pet

        ports:

        - name: http

          containerPort: 8080

        volumeMounts:

        - name: data

          mountPath: /var/data

  volumeClaimTemplates:

  - metadata:

      name: data

    spec:

      resources:

        requests:

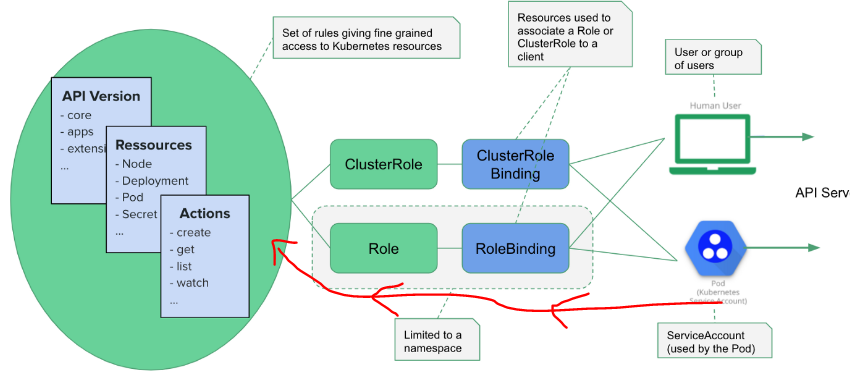
          storage: 1Mi

      accessModes:

      - ReadWriteOnce

# RBAC / API

## Role binding / service account



Mối liên hệ: [Pod < serviceaccount|secret < rolebinding < role]

# kubectl create **serviceaccount** demo-sa -o yaml --dry-run=client

# kubectl create **role** list-pods --verb=list --resource=pods --dry-run=client -o yaml

# kubectl create rolebinding list-pods\_demo-sa --role=list-pods --serviceaccount=default:demo-sa --dry-run=client -o yaml

# kubectl run debugpod --image=nicolaka/netshoot --dry-run=client -o yaml

Sửa pod gắn thêm service-account:

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: debugpod

  name: debugpod

spec:

  serviceAccountName: demo-sa

  containers:

  - image: nicolaka/netshoot

    name: debugpod

    command: ["/bin/bash"]

    args: ["-c", "sleep 100000"]

Ta vào trong POD, kiểm tra serviceaccout được mount vào file:

> cat /var/run/secrets/kubernetes.io/serviceaccount/token

> TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)

> curl -H "Authorization: Bearer $TOKEN" https://kubernetes/api/v1/ --insecure

## API

<https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cfc398c>

<https://iximiuz.com/en/posts/kubernetes-api-call-simple-http-client/>

kubectl create serviceaccount api-explorer --dry-run=client -o yaml

kubectl create clusterrole log-reader --verb=get,list,watch --resource=pods,pods/log --dry-run=client -o yaml

kubectl create rolebinding api-explorer:log-reader --clusterrole=log-reader --serviceaccount=default:api-explorer

SERVICE\_ACCOUNT=api-explorer

SECRET=$(kubectl get serviceaccount ${SERVICE\_ACCOUNT} -o json | jq -Mr '.secrets[].name | select(contains("token"))')

TOKEN=$(kubectl get secret ${SECRET} -o json | jq -Mr '.data.token' | base64 -d)

kubectl get secret ${SECRET} -o json | jq -Mr '.data["ca.crt"]' | base64 -d > /tmp/ca.crt

APISERVER=https://$(kubectl -n default get endpoints kubernetes --no-headers | awk '{ print $2 }')

curl -s $APISERVER/openapi/v2 -H "Authorization: Bearer $TOKEN" --cacert /tmp/ca.crt | jq .

curl -s $APISERVER/api/v1/namespaces/default/pods/ -H "Authorization: Bearer $TOKEN" --cacert /tmp/ca.crt

( Note: để lấy toàn bộ thông tin k8s, ta có thể lấy ca.crt và token của kube-system: k get serviceaccounts -A | grep default | grep kube-system)

# Taints and tolerations

## Taints

Đầu tiên ta kiểm tra Taint giữa master và worker có gì khác nhau:

[root@master-node ~]# k describe node master-node | grep -i taint

Taints: node-role.kubernetes.io/master:NoSchedule

[root@master-node ~]# k describe node worker-node1 | grep -i taint

Taints: <none>

Điều kiện 1 pod có thể deploy là có Toleration giống với Taint của node.

[root@master-node ~]# k describe pod calico-kube-controllers-566dc76669-mvtn2 -n kube-system | grep master

Tolerations:

node-role.kubernetes.io/master:NoSchedule

# podAntiAffinity

# TIP-TRICK

### Install telnet in docker apk

$ apk update

$ apk add busybox-extras

$ busybox-extras telnet localhost 6900

# Sample Prod

Dashboard

<https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

curl -O <https://raw.githubusercontent.com/kubernetes/dashboard/v2.5.0/aio/deploy/recommended.yaml>

Sửa lại file chuyển từ clusterIP sang NodePort

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: dashboard-metrics-scraper

name: dashboard-metrics-scraper

namespace: kubernetes-dashboard

spec:

type: NodePort

ports:

- port: 8000

targetPort: 8000

nodePort: 31000

selector:

k8s-app: dashboard-metrics-scraper

Test gọi thử:

curl localhost:31000

URL: /

Giờ ta cần lấy mã token

kubectl create serviceaccount dashboard-admin-sa

kubectl create clusterrolebinding dashboard-admin-sa --clusterrole=cluster-admin --serviceaccount=default:dashboard-admin-sa

kubectl describe secret dashboard-admin\*\*\*\*\*

## K8s Registry:

Bước 1. Chỉ định hosts:

echo 192.168.88.12 registry.tuanda.vn >> /etc/hosts

Bước 2: Import basic-auth và ssl vào configmap

# mkdir /opt/certs /opt/registry

# cd /opt

# openssl req -x509 -out ca.crt -keyout ca.key -days 1825 \

-newkey rsa:2048 -nodes -sha256 \

-subj '/CN=registry.tuanda.vn' -extensions EXT -config <( \

printf "[dn]\nCN=registry.tuanda.vn\n[req]\ndistinguished\_name = dn\n[EXT]\nsubjectAltName=DNS:registry.tuanda.vn\nkeyUsage=digitalSignature\nextendedKeyUsage=serverAuth")

# cd /opt/certs/

# kubectl create configmap registry-cert --from-file=ca.crt --from-file=ca.key

# yum install httpd-tools -y ; htpasswd -Bbn tuanda 123 > htpasswd

# kubectl create configmap registry-basic-auth --from-file=htpasswd

# kubectl get configmaps

Bước 4: Tạo deployment và service NodePort

apiVersion: apps/v1

kind: Deployment

metadata:

name: private-repository-k8s

labels:

app: private-repository-k8s

spec:

replicas: 1

selector:

matchLabels:

app: private-repository-k8s

template:

metadata:

labels:

app: private-repository-k8s

spec:

volumes:

- name: certs-vol

configMap:

name: registry-cert

- name: auth-vol

configMap:

name: registry-basic-auth

- name: registry-vol

hostPath:

path: /opt/registry

type: Directory

containers:

- image: registry:2

name: private-repository-k8s

imagePullPolicy: IfNotPresent

env:

- name: REGISTRY\_AUTH

value: htpasswd

- name: REGISTRY\_AUTH\_HTPASSWD\_PATH

value: "/auth/htpasswd"

- name: REGISTRY\_AUTH\_HTPASSWD\_REALM

value: Registry Realm

- name: REGISTRY\_HTTP\_TLS\_CERTIFICATE

value: "/certs/ca.crt"

- name: REGISTRY\_HTTP\_TLS\_KEY

value: "/certs/ca.key"

ports:

- containerPort: 5000

volumeMounts:

- name: certs-vol

mountPath: /certs

- name: registry-vol

mountPath: /var/lib/registry

- name: auth-vol

mountPath: /auth

---

apiVersion: v1

kind: Service

metadata:

labels:

app: private-repository-k8s

name: private-repository-k8s

spec:

ports:

- port: 5000

nodePort: 31320

protocol: TCP

targetPort: 5000

selector:

app: private-repository-k8s

type: NodePort

Bước 4: Trust CA

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

sudo update-ca-trust

sudo service docker restart

Bước 5: Đẩy cert vào tất cả các node docker, để permit self-certificate gọi pull. (all node)

mkdir -p /etc/docker/certs.d/registry.tuanda.vn:31320

cp -rp /opt/certs/ca.crt /etc/docker/certs.d/registry.tuanda.vn\:31320/

Bước 6: docker login đẩy config registry client sang các node:

# curl -v --user tuanda:123 https://registry.tuanda.vn:31320/v2/

# docker login registry.tuanda.vn:31320 -u tuanda -p 123

cat ~/.docker/config.json

{

"auths": {

"registry.tuanda.vn:31320": {

"auth": "dHVhbmRhOjEyMw=="

}

}

}

mkdir -p /home/tuanda/.docker ; chown -R tuanda.tuanda /home/tuanda/.docker

Ta copy file config.json ở trên sang các worker node trong cluster. (/home/tuanda/.docker/config.json)

Bước 6: đẩy image lên registry:

# docker pull nginx:alpine

# docker tag nginx:alpine registry.tuanda.vn:31320/nginx:alpine

# docker push registry.tuanda.vn:31320/nginx:alpine

Bước 7: Launch pod với option registry

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello-kubernetes

namespace: tuanda

spec:

replicas: 1

selector:

matchLabels:

app: hello-kubernetes

template:

metadata:

labels:

app: hello-kubernetes

spec:

containers:

- name: hello-kubernetes-debug

image: admin.tuan.name.vn:31320/debug-tools:1.0.0

ports:

- containerPort: 8080

- name: hello-kubernetes-nginx

image: admin.tuan.name.vn:31320/nginx:alpine

ports:

- containerPort: 80

imagePullSecrets:

- name: regcred

## Ingress nginx:

<https://kubernetes.github.io/ingress-nginx/deploy/>

curl -O <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.1/deploy/static/provider/cloud/deploy.yaml>

# kubectl apply -f deploy.yaml

kubectl create deployment demo --image=httpd --port=80

kubectl expose deployment demo

kubectl create ingress demo-localhost --class=nginx --rule=demo.localdev.me/\*=demo:80

[tuanda@master-node ~]$ k get ingress -A

NAMESPACE NAME CLASS HOSTS ADDRESS PORTS AGE

default demo-localhost nginx demo.localdev.me 80 10m

[tuanda@master-node ~]$ k get ingress demo-localhost -o yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

creationTimestamp: "2022-03-09T19:17:25Z"

generation: 1

name: demo-localhost

namespace: default

resourceVersion: "3872"

uid: 9d495903-1f4a-4166-8e08-89b3cc15422f

spec:

ingressClassName: nginx

rules:

- host: demo.localdev.me

http:

paths:

- backend:

service:

name: demo

port:

number: 80

path: /

pathType: Prefix

status:

loadBalancer: {}

## Nfs provisioning

<https://fabianlee.org/2022/01/12/kubernetes-nfs-mount-using-dynamic-volume-and-storage-class/?msclkid=a5ca54e2ae7111eca259ced5d2223a6f>

<https://artifacthub.io/packages/helm/nfs-subdir-external-provisioner/nfs-subdir-external-provisioner>

B1: Cài Nfs server:

yum install nfs-utils nfs-utils-lib -y

chkconfig rpcbind on

chkconfig nfs on

service rpcbind restart

service nfs restart

mkdir -p /data/nfs-k8s/ ; vim /etc/exports

/data/nfs-k8s/ 192.168.88.0/24(rw,sync,subtree\_check,no\_root\_squash)

exportfs -a

showmount -e 127.0.0.1

B2: Cài nfs client trên mỗi worker node, nếu không sẽ lỗi không mout đc vào pod.

yum install nfs-utils nfs-utils-lib -y

chkconfig nfs off

chkconfig rpcbind off

B3: Helm install

helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner

helm pull nfs-subdir-external-provisioner/nfs-subdir-external-provisioner

helm template nfs-subdir-external-provisioner . --set nfs.server=192.168.88.12 \

  --set nfs.path=/data/nfs-k8s/ \

  --set storageClass.name=nfs-provisioner \

  --set storageClass.onDelete=retain \

  --set storageClass.accessModes=ReadWriteMany

B4: Test

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: sc-nfs-pvc

spec:

  accessModes:

    - ReadWriteMany

  storageClassName: nfs-provisioner

  resources:

    requests:

      storage: 2Gi

apiVersion: v1

kind: Pod

metadata:

  name: busybox

spec:

  volumes:

  - name: myvol

    persistentVolumeClaim:

      claimName: sc-nfs-pvc

  containers:

  - image: busybox

    name: busybox

    command: ["/bin/sh"]

    args: ["-c", "sleep 600000"]

    volumeMounts:

    - name: myvol

      mountPath: /data

## Debug image

<https://cloudogu.com/en/blog/k8s-app-ops-part-2>

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: debugpod

  name: debugpod

spec:

  containers:

  - image: nicolaka/netshoot

    name: debugpod

    command: ["/bin/bash"]

    args: ["-c", "sleep 100000"]

hoặc

# **ARGO-CD -- ARGO-CD -- ARGO-CD-- ARGO-CD -- ARGO-CD --**

## CÀI ĐẶT

kubectl create ns argo

#url : https://github.com/argoproj/argo-workflows/tree/master/manifests (trong link này có các bản stable và test)

wget <https://raw.githubusercontent.com/argoproj/argo/stable/manifests/quick-start-postgres.yaml>

<https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml>

kubectl apply -n argo -f quick-start-postgres.yaml

kubectl -n argo port-forward deployment/argo-server 2746:2746

Truy cập vào: <https://127.0.0.1:2746/workflows>

# kubectl -n argo get all -o wide

### 3. Hello World Workflow

[tuanda@localhost argo]$ cat 3.1.wf-hello-world.yaml

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  generateName: hello-world-  # Name of this Workflow

spec:

  entrypoint: whalesay        # Defines "whalesay" as the "main" template

  templates:

  - name: whalesay            # Defining the "whalesay" template

    container:

      image: docker/whalesay

      command: [cowsay]

      args: ["hello world"]   # This template runs "cowsay" in the "whalesay" image with arguments "hello world"

[tuanda@localhost argo]$ kubectl -n argo create -f 3.1.wf-hello-world.yaml

## TEMPLATE TRONG ARGO

### Container template

[tuanda@localhost argo]$ cat 6.1.wf-container-template.yaml

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  generateName: wf-container-template-

spec:

  entrypoint: container-template

  templates:

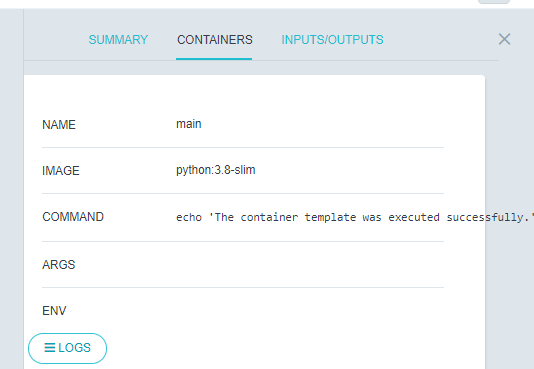
  - name: container-template

    container:

      image: python:3.8-slim

      command: [echo, "The container template was executed successfully."]

[tuanda@localhost argo]$ kubectl -n argo create -f 6.1.wf-container-template.yaml



### Script Template

[tuanda@localhost argo]$ cat 7.1.wf-script-template.yaml

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  generateName: wf-script-template-

spec:

  entrypoint: script-template

  templates:

  - name: script-template

    script:

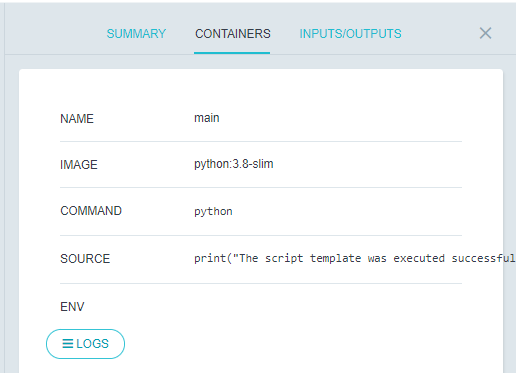
      image: python:3.8-slim

      command: [python]

      source: |

        print("The script template was executed successfully.")

[tuanda@localhost argo]$ kubectl -n argo create -f 7.1.wf-script-template.yaml



### Resource Template

[tuanda@localhost argo]$ vim 9.1.resource-template.yaml

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  generateName: wf-resource-template-

spec:

  entrypoint: resource-template

  templates:

  - name: resource-template

    resource:

      action: create

      manifest: |

        apiVersion: argoproj.io/v1alpha1

        kind: Workflow

        metadata:

          name: wf-test

        spec:

          entrypoint: test-template

          templates:

          - name: test-template

            script:

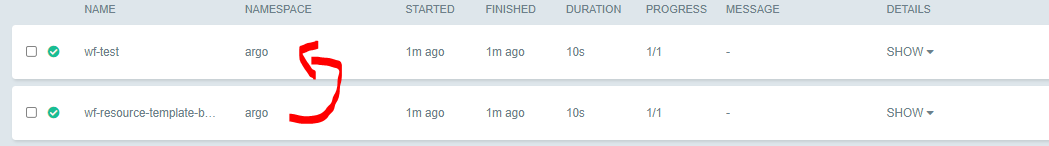
              image: python:3.8-slim

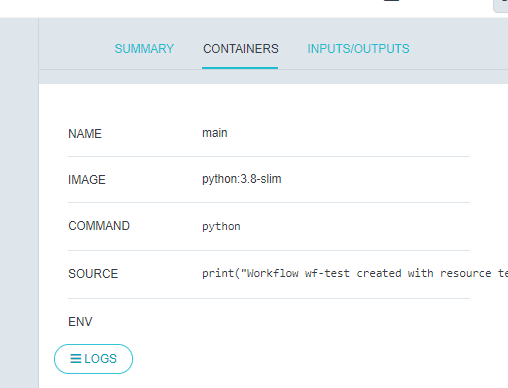
              command: [python]

              source: |

                print("Workflow wf-test created with resource template.")

[tuanda@localhost argo]$ kubectl -n argo create -f 9.1.resource-template.yaml





### Suspend

## Template INVOCATORS

### STEP

#### Serial Step (Step nối tiếp nhau)

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-steps-templates-serial

spec:

  entrypoint: steps-template-serial

  templates:

  - name: steps-template-serial

    steps:

    - - name: step1

        template: task-template

    - - name: step2

        template: task-template

    - - name: step3

        template: task-template

  - name: task-template

    script:

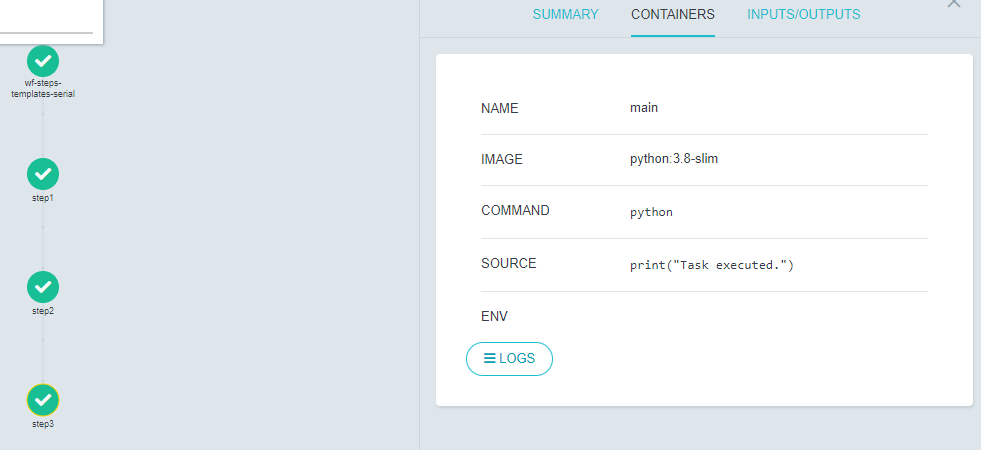
      image: python:3.8-slim

      command: [python]

      source: |

        print("Task executed.")

[tuanda@localhost argo]$ kubectl -n argo create -f 11.1.wf-template-serial.yaml



#### Step Parabel (Step song song)

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-steps-templates-parallel

spec:

  entrypoint: steps-template-parallel

  templates:

  - name: steps-template-parallel

    steps:

    - - name: step1

        template: task-template

    - - name: step2

        template: task-template

      - name: step3

        template: task-template

    - - name: step4

        template: task-template

      - name: step5

        template: task-template

    - - name: step6

        template: task-template

  - name: task-template

    script:

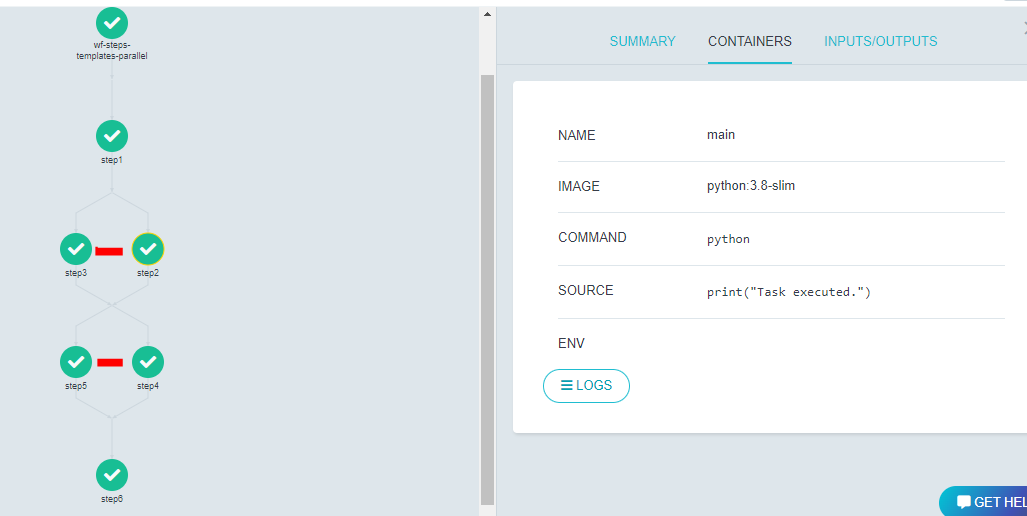
      image: python:3.8-slim

      command: [python]

      source: |

        print("Task executed.")

[tuanda@localhost argo]$ kubectl -n argo create -f 12.1.wf-step-template-parabel.yaml



#### Suspend Step Template

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-suspend-steps-template

spec:

  entrypoint: steps-template

  templates:

  - name: steps-template

    steps:

    - - name: step1

        template: task-template

    - - name: step2

        template: task-template

      - name: step3

        template: task-template

    - - name: delay

        template: delay-template

    - - name: step4

        template: task-template

  - name: task-template

    script:

      image: python:3.8-slim

      command: [python]

      source: |

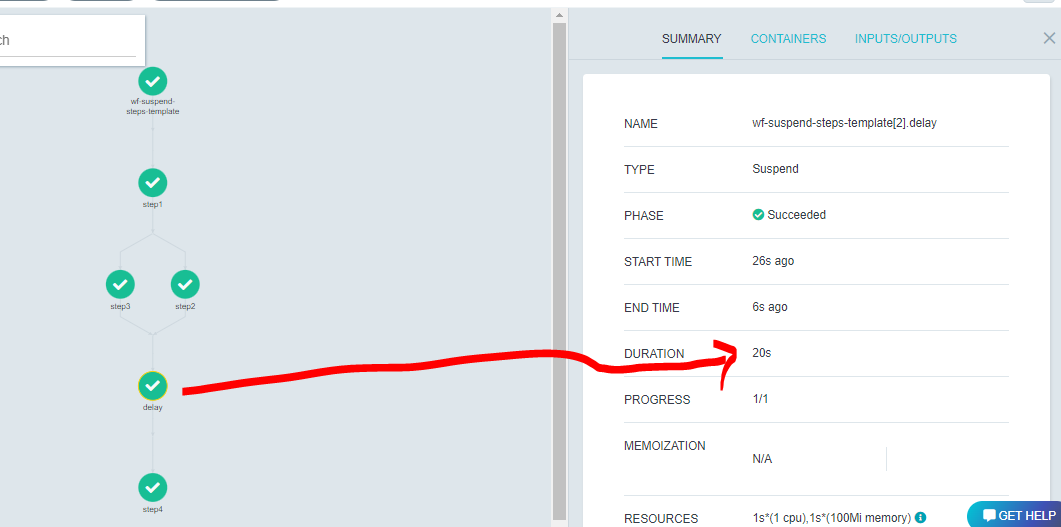
        print("Task executed.")

  - name: delay-template

    suspend:

      duration: "20s"

[tuanda@localhost argo]$ kubectl -n argo create -f 13.1.wf-suspend-template.yaml



### DAG

(Dag giống với step, nhưng khác với step là nó tường minh về **phụ thuộc task cha** của nó)

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-dag-template

spec:

  entrypoint: dag-template

  templates:

  - name: dag-template

    dag:

      tasks:

      - name: Task1

        template: task-template

      - name: Task2

        template: task-template

        dependencies: [Task1]

      - name: Task3

        template: task-template

        dependencies: [Task1]

      - name: Task4

        template: task-template

        dependencies: [Task2, Task3]

  - name: task-template

    script:

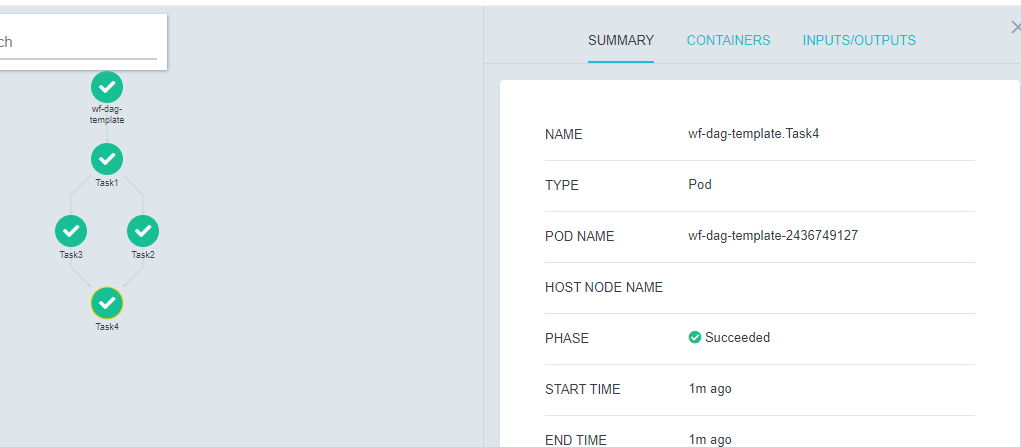
      image: python:3.8-slim

      command: [python]

      source: |

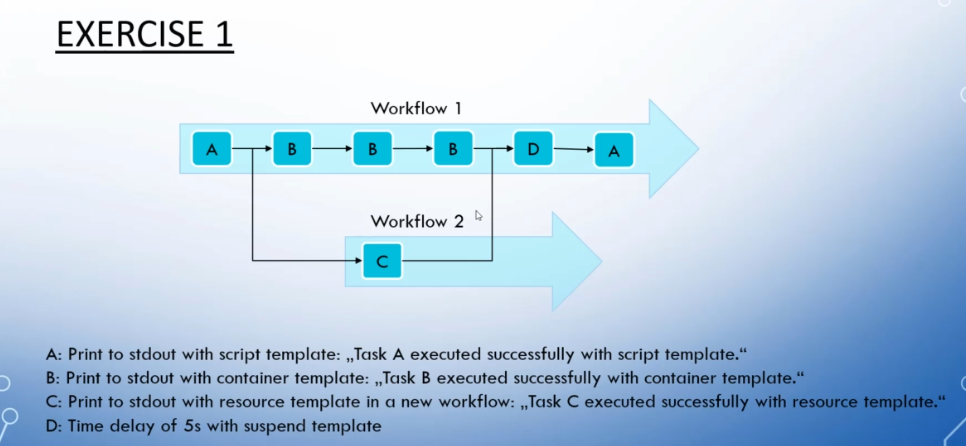
        print("Task executed.")

[tuanda@localhost argo]$ kubectl -n argo create -f 14.1-wf-dag-template.yaml



### Bài tập 1 về Step và Dag

Yêu cầu:



**Chú ý:** A là script, B là container, C là resource, D là suspend Template.

Ta chia làm 4 loại template và gọi nhau theo thứ tự DAG.

Giải bài tập:

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-exercise1

spec:

  entrypoint: dag-template

  templates:

  - name: dag-template

    dag:

      tasks:

      - name: Task1

        template: taskA-template

      - name: Task2

        template: taskB-template

        dependencies: [Task1]

      - name: Task3

        template: taskC-template

        dependencies: [Task1]

      - name: Task4

        template: taskB-template

        dependencies: [Task2]

      - name: Task5

        template: taskB-template

        dependencies: [Task4]

      - name: Task6

        template: delay-template

        dependencies: [Task3, Task5]

      - name: Task7

        template: taskA-template

        dependencies: [Task6]

  - name: taskA-template

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        print("Task A executed successfully with script template.")

  - name: taskB-template

    container:

      image: python:3.8-slim

      command: [echo, "Task B executed successfully with container template."]

  - name: taskC-template

    resource:

      action: create

      manifest: |

        apiVersion: argoproj.io/v1alpha1

        kind: Workflow

        metadata:

          name: wf-resource-template

        spec:

          entrypoint: resource-template

          templates:

          - name: resource-template

            script:

              image: python:3.8-slim

              command: [python]

              source: |

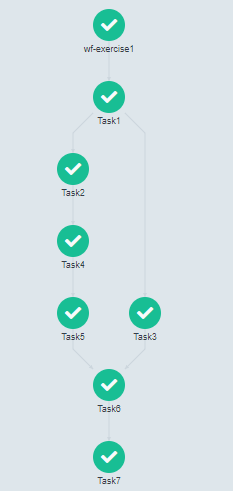
                print("Task C executed successfully with resource template.")

  - name: delay-template

    suspend:

      duration: "5s"

[tuanda@localhost argo]$ kubectl -n argo create -f 15.baitap.yaml

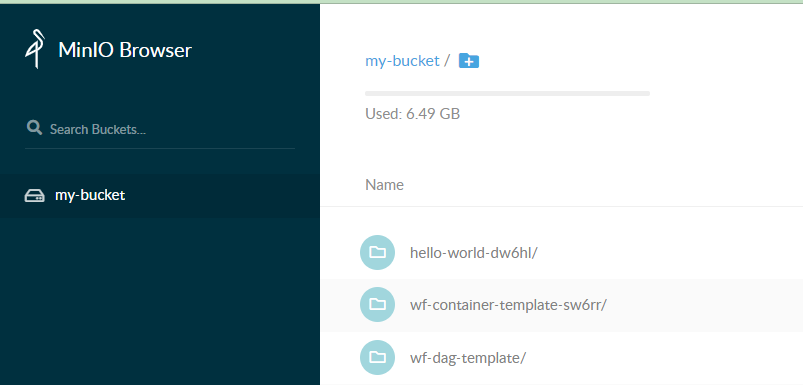


## WORKFLOW FUNCTION ARGO

### 1.MinIO

# kubectl -n argo port-forward deployment.apps/minio 9000:9000

<http://127.0.0.1:9000/> Mật khẩu mặc định là admin/password



### 2.Cài đặt Argo-Cli

<https://github.com/argoproj/argo-workflows/releases>

curl -sLO https://github.com/argoproj/argo-workflows/releases/download/v3.2.6/argo-linux-amd64.gz

gunzip argo-linux-amd64.gz

chmod +x argo-linux-amd64

mv ./argo-linux-amd64 /usr/local/bin/argo

argo version

### 3.InputParameter:

<https://nimtechnology.com/2022/01/06/argo-workflows-lesson3-argo-cli-and-input-parameters/>

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-input-parameter-dag

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

    - name: message3

      value: Task 3 finished

    - name: message4

      value: That's it with task 4

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

      - name: message3

      - name: message4

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}"}]

        template: task-template

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}"}]

        template: task-template

        dependencies: [Task1]

      - name: Task3

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message3}}"}]

        template: task-template

        dependencies: [Task1]

      - name: Task4

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message4}}"}]

        template: task-template

        dependencies: [Task2, Task3]

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

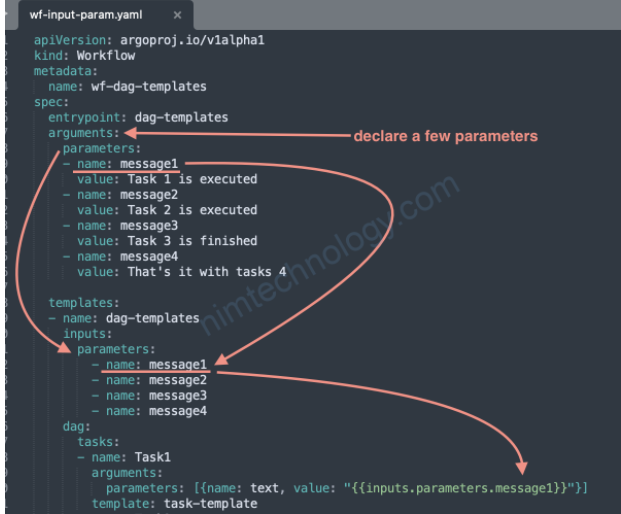
      image: python:3.8-slim

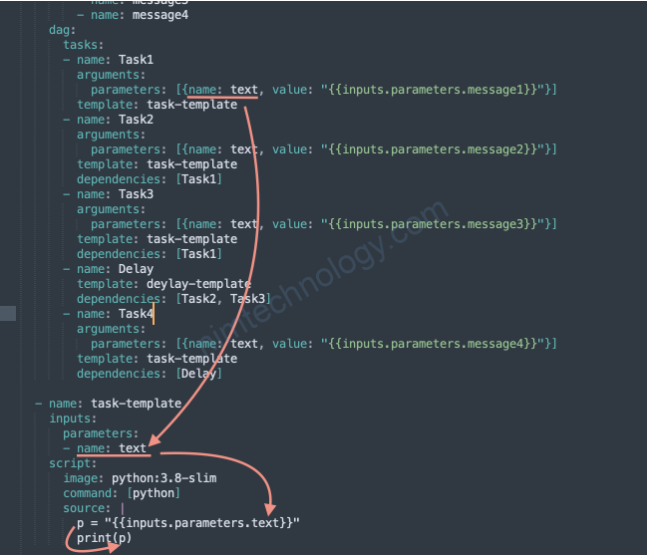
      command: [python]

      source: |

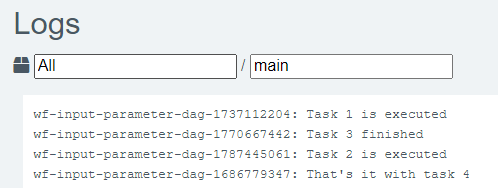
        p = "{{inputs.parameters.text}}"

        print(p)





Kết quả:

### 4.Scripts Result

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-script-result

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}"}]

        template: task-template

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}"}]

        template: task-template

        dependencies: [Task1]

      - name: Task3

        template: task-output

        dependencies: [Task1]

      - name: Task4

        arguments:

          parameters: [{name: text, value: "{{tasks.Task3.outputs.result}}"}]

        template: task-template

        dependencies: [Task2, Task3]

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-output

    script:

      image: node:9.1-alpine

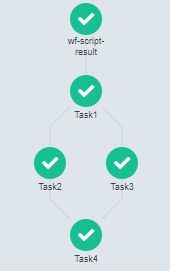
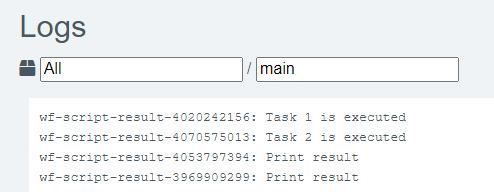
      command: [node]

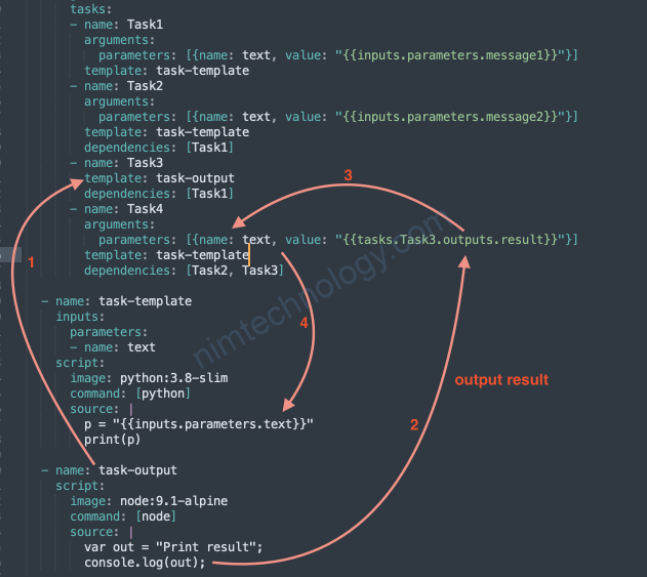
      source: |

        var out = "Print result";

        console.log(out);

Đầu ra của console.log(out) Task3 sẽ là đầu vào của Task 4



### 5.Output parameter

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-output-parameter

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}"}]

        template: task-template

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}"}]

        template: task-template

        dependencies: [Task1]

      - name: Task3

        template: task-output

        dependencies: [Task1]

      - name: Task4

        arguments:

          parameters: [{name: text, value: "{{tasks.Task3.outputs.parameters.task-param}}"}]

        template: task-template

        dependencies: [Task2, Task3]

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-output

    script:

      image: node:9.1-alpine

      command: [node]

      source: |

        var out = "Print result";

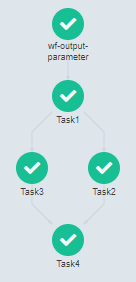
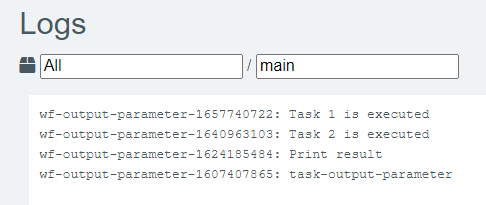
        console.log(out);

    outputs:

      parameters:

      - name: task-param

        value: "task-output-parameter"

### 6.Output Parameter File

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-output-parameter-file

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}" }]

        template: task-template

      - name: Task2

        dependencies: [Task1]

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}" }]

        template: task-template

      - name: Task3

        dependencies: [Task1]

        template: task-output

      - name: Task4

        dependencies: [Task2, Task3]

        arguments:

          parameters: [{name: text, value: "{{tasks.Task3.outputs.parameters.task-param}}" }]

        template: task-template

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-output

    script:

      image: node:9.1-alpine

      command: [node]

      source: |

        var par = "Whatever parameters are written to the file.";

        const fs = require('fs');

        fs.writeFile("/tmp/output-params.txt", par)

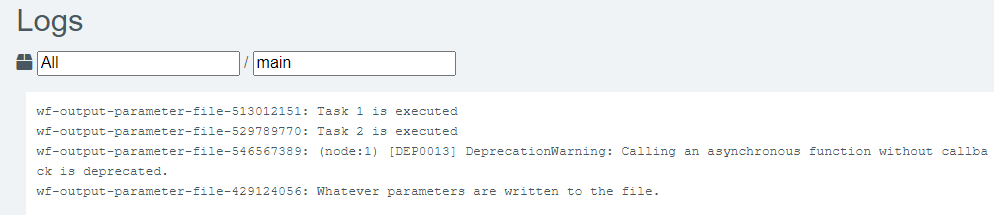
    outputs:

      parameters:

      - name: task-param

        valueFrom:

          path: /tmp/output-params.txt

### 7. Artifact

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-artifact

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}" }]

        template: task-template

      - name: Task2

        dependencies: [Task1]

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}" }]

        template: task-template

      - name: Task3

        dependencies: [Task1]

        template: task-output-artifact

      - name: Task4

        dependencies: [Task2, Task3]

        arguments:

          artifacts: [{name: text, from: "{{tasks.Task3.outputs.artifacts.artifact-out}}" }]

        template: task-input-artifact

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-output-artifact

    script:

      image: node:9.1-alpine

      command: [node]

      source: |

        var par = "Whatever parameters are written to the file.";

        const fs = require('fs');

        fs.writeFile("/tmp/output-params.txt", par)

    outputs:

      artifacts:

      - name: artifact-out

        path: /tmp/output-params.txt

  - name: task-input-artifact

    inputs:

      artifacts:

      - name: text

        path: /tmp/text

    script:

      image: python:3.8-slim

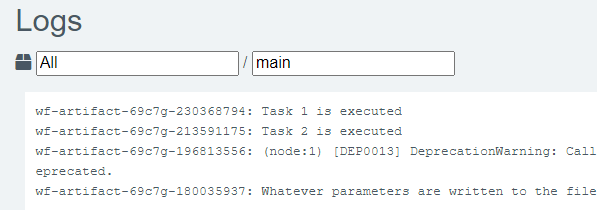
      command: [python]

      source: |

        with open("/tmp/text", "r") as f:

          lines = f.read()

          print(lines)

### 8. Secrets as environment variables

### 9. Secrets as mounted volumes

### 10. Loops

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-loop

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}" }]

        template: task-template

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}" }]

        template: task-template

      - name: Task3

        dependencies: [Task1]

        template: task-template

        arguments:

          parameters:

          - name: text

            value: "{{item}}"

        withItems:

        - Element1

        - Element2

        - Element3

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

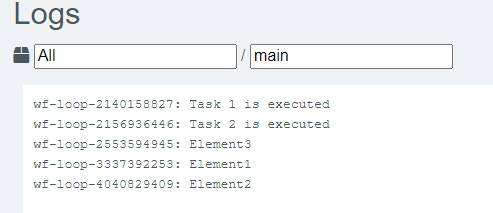
      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

### 11. Loops with sets

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-loop-sets

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}" }]

        template: task-template

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}" }]

        template: task-template

      - name: Task3

        dependencies: [Task1]

        template: task-loop-set

        arguments:

          parameters:

          - name: extractor

            value: "{{item.extractor}}"

          - name: table

            value: "{{item.table}}"

        withItems:

        - { extractor: 'PythonExtractor', table: 'Table 1'}

        - { extractor: 'PySparkExtractor', table: 'Table 2'}

        - { extractor: 'DaskExtractor', table: 'Table 3'}

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-loop-set

    inputs:

      parameters:

      - name: extractor

      - name: table

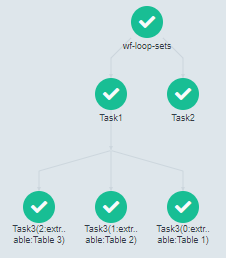
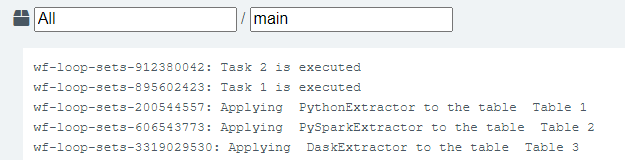
    script:

      image: python:3.8-slim

      command: [python]

      source: |

        print("Applying ", "{{inputs.parameters.extractor}}", "to the table ", "{{inputs.parameters.table}}")

### 12. Loops with sets as input parameters

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-loop-sets-inputparam

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

    - name: message2

      value: Task 2 is executed

    - name: ingest-list

      value: |

        [

          { "extractor": "PythonExtractor", "table": "Table 1"},

          { "extractor": "PySparkExtractor", "table": "Table 2"},

          { "extractor": "DaskExtractor", "table": "Table 3"}

        ]

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

      - name: message2

      - name: ingest-list

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}" }]

        template: task-template

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message2}}" }]

        template: task-template

      - name: Task3

        dependencies: [Task1]

        template: task-loop-set

        arguments:

          parameters:

          - name: extractor

            value: "{{item.extractor}}"

          - name: table

            value: "{{item.table}}"

        withParam: "{{inputs.parameters.ingest-list}}"

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-loop-set

    inputs:

      parameters:

      - name: extractor

      - name: table

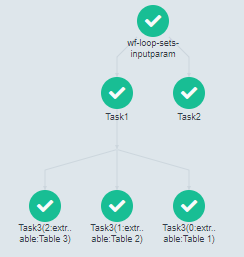
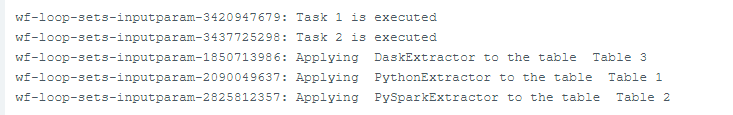
    script:

      image: python:3.8-slim

      command: [python]

      source: |

        print("Applying ", "{{inputs.parameters.extractor}}", "to the table ", "{{inputs.parameters.table}}")

### 13. Dynamic Loops

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-loop-dynamic

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: message1

      value: Task 1 is executed

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: message1

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.message1}}" }]

        template: task-template

      - name: Task2

        template: task-generate-list

      - name: Task3

        dependencies: [Task2]

        template: task-loop-set

        arguments:

          parameters:

          - name: extractor

            value: "{{item.extractor}}"

          - name: table

            value: "{{item.table}}"

        withParam: "{{tasks.Task2.outputs.result}}"

  - name: task-template

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-generate-list

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        import json

        import sys

        list = [("PythonExtractor", "Table 1"), ("PySparkExtractor", "Table 2"), ("DaskExtractor", "Table 3")]

        json.dump([{"extractor": i[0], "table": i[1]} for i in list], sys.stdout)

  - name: task-loop-set

    inputs:

      parameters:

      - name: extractor

      - name: table

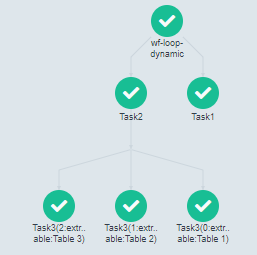
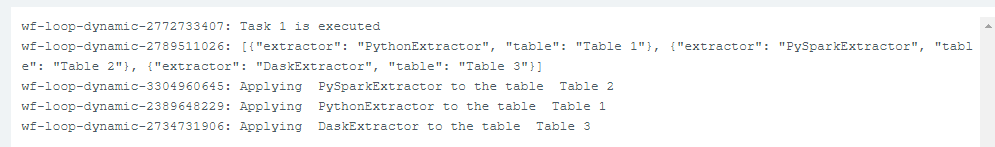
    script:

      image: python:3.8-slim

      command: [python]

      source: |

        print("Applying ", "{{inputs.parameters.extractor}}", "to the table ", "{{inputs.parameters.table}}")

### 14. Conditionals

apiVersion: argoproj.io/v1alpha1

kind: Workflow

metadata:

  name: wf-condition

spec:

  entrypoint: dag-template

  arguments:

    parameters:

    - name: messageA

      value: A

    - name: messageB

      value: B

  templates:

  - name: dag-template

    inputs:

      parameters:

      - name: messageA

      - name: messageB

    dag:

      tasks:

      - name: Task1

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.messageA}}" }]

        template: task-decision

      - name: TaskA

        template: task-A

        dependencies: [Task1]

        when: "{{tasks.Task1.outputs.result}} == A"

      - name: TaskB

        template: task-B

        dependencies: [Task1]

        when: "{{tasks.Task1.outputs.result}} == B"

      - name: Task2

        arguments:

          parameters: [{name: text, value: "{{inputs.parameters.messageB}}" }]

        template: task-decision

      - name: TaskA2

        template: task-A

        dependencies: [Task2]

        when: "{{tasks.Task2.outputs.result}} == A"

      - name: TaskB2

        template: task-B

        dependencies: [Task2]

        when: "{{tasks.Task2.outputs.result}} == B"

  - name: task-decision

    inputs:

      parameters:

      - name: text

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        p = "{{inputs.parameters.text}}"

        print(p)

  - name: task-A

    script:

      image: python:3.8-slim

      command: [python]

      source: |

        print("Task A was executed.")

  - name: task-B

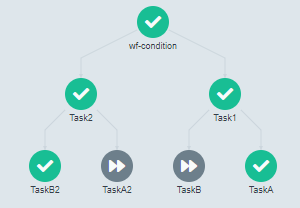
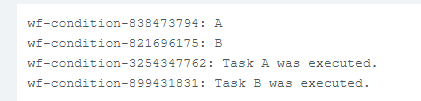
    script:

      image: python:3.8-slim

      command: [python]

      source: |

        print("Task B was executed.")

### 15. Depends

### 16. Depends theorie

### 17. Retry strategy

### 18. Recursion

### 19. Exercise 2 - task introduction

### 20. Exercise 2 - solution