테크톤 101

TEKTON CD FOR KUBERNETES

2024-01-23

테크톤 소개

TEKTON

테크톤은 CNCF에서 공식으로 지정한 Continuous Delivery도구이다. 테크톤은 엄밀히 말하자면 절차적인 자동화 도구라고 보시면 된다. 테크톤은 Continuous Integration기능도 제공하나, 엄밀하게 말하자만 SCM에 구성이 되어 있는 소스코드를 가져와서 이미지 빌드를 할 수 있도록 지원한다. 테크톤은 개발자에게 다음과 같은 기능 제공한다.

1. build
2. test
3. deploy

테크톤에서의 모든 자원은 쿠버네티스와 동일하게 YAML기반으로 제공한다. 코드 기반으로 각각 작업들을 구성하며, 해당작업을 작업공간 및 파이프라인을 통해서 처리가 가능하다. 개발자가 손쉽게 YAML코드 기반으로 애플리케이션을 서비스로 배포가 가능하다.

CI/CD

CI는 일반적으로 GIT이나 혹은 SVN를 이야기 한다. 이를 통틀어서 SCM이라고 호칭한다. 테크톤은 직접적으로 SCM를 구성 및 제공하지 않으며, 기존에 구성이 되어있는 소스코드를 SCM에서 가져와서 갱신 및 이미지를 생성을 한다.

```
workspaces:
- name: shared-data
  volumeClaimTemplate:
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
- name: git-credentials
  secret:
    secretName: git-credentials
params:
- name: repo-url
  value: git@github.com:tektoncd/website.git
```

TEKTON VS JENKINS

테크톤과 젠킨스는 기능 및 역할이 다르다. 젠킨스는 플러그인 및 혹은 로컬 언어(Local Language)를 사용해서 작업 순서를 구성한다. 젠킨스도 Groovy라는 스크립트 명령어를 통해서 파이프라인 및 작업 수행이 가능하지만, 간단하게 파이프라인 및 자바 JDK와 같은 도구를 지원한다. 다른 의미로 테크톤 보다 더 제한적인 범위로 지원한다.

처음 시작하는 사용자에게는 학습 난이도가 낮은 테크톤 기반으로 구성을 권장하며, 추가적인 기능이 필요한 경우 플러그인 제작 및 설치가 별도로 필요하지 않는다. 또한, YAML문법을 사용하기 때문에 젠킨스의 Groovy언어 문법 및 구조체를 복잡하게 학습할 필요가 없다.

TEKTON VS JENKINS

젠킨스 Groovy의 문법 형식은 아래와 같다.

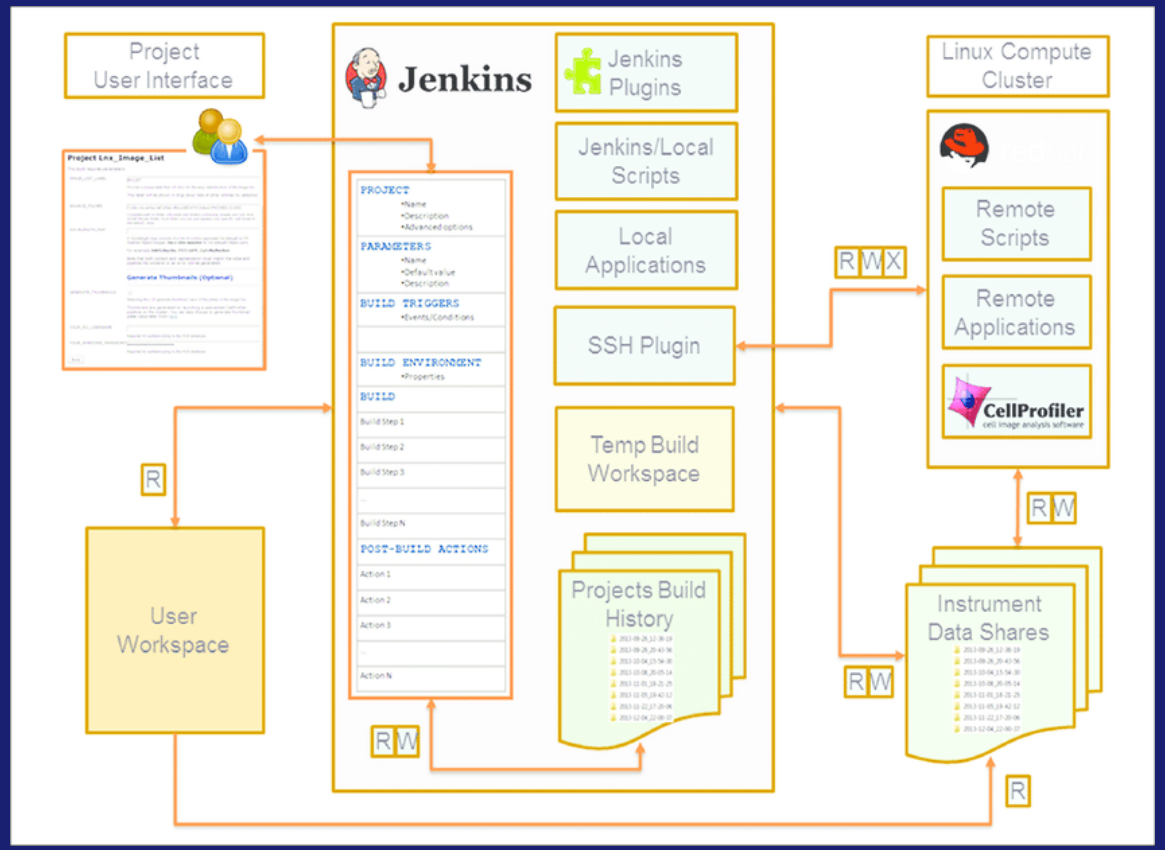
```
node {
    git url: 'https://github.com/jfrogdev/project-examples.git'
    def server = Artifactory.server "SERVER_ID"
    def downloadSpec =
            '''{
            "files": [
                {
                    "pattern": "libs-snapshot-local/*.zip",
                    "target": "dependencies/",
                    "props": "p1=v1;p2=v2"
                }
            ]
        }'''
```

TEKTON VS JENKINS

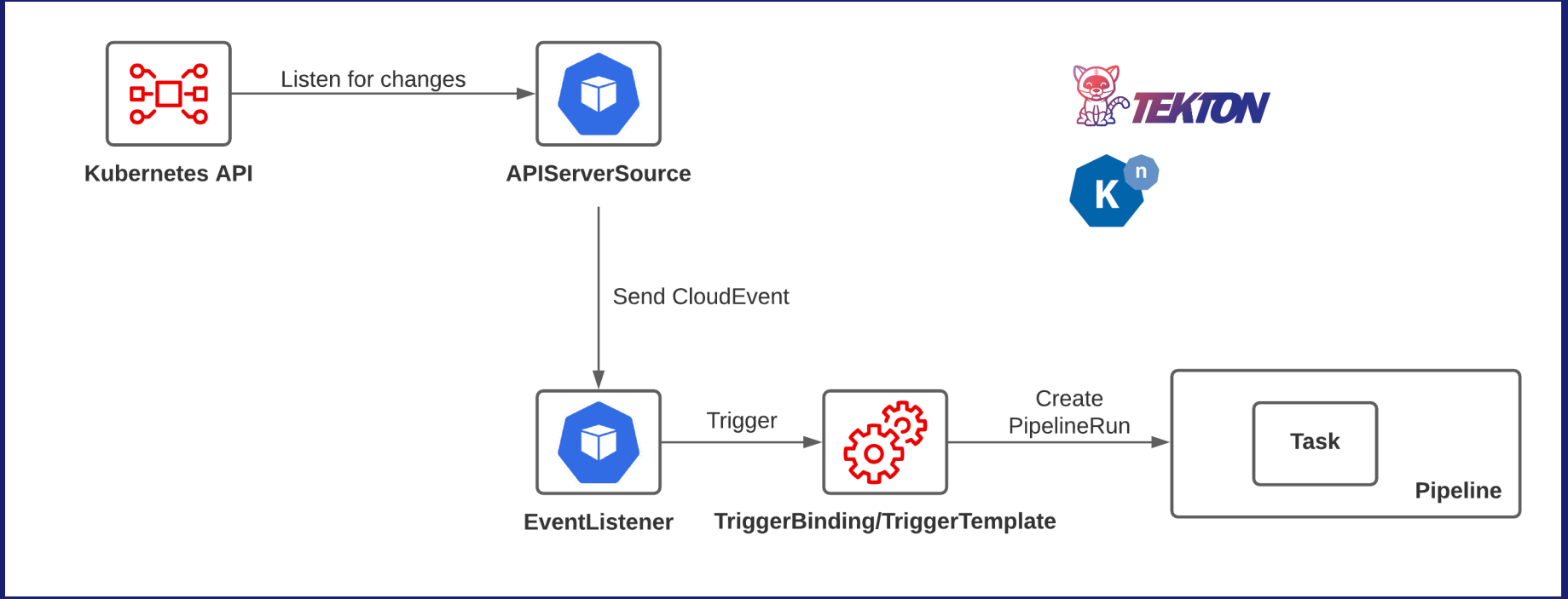
테크톤과 젠킨스의 제일 큰 차이점은 동작 방식이다. 젠킨스는 쿠버네티스와 통합된 형태가 아니고, 본래 자바에서 빌드용도 사용하다가 쿠버네티스와 통합된 상태이다. 그러한 이유로 젠킨스는 완벽하게 쿠버네티스와 통합이 되지 않는다.

테크톤은 기본적으로 쿠버네티스 클러스터 기반으로 구성 및 작성이 되었다. 젠킨스보다 손쉽게 사용하며, 별도로 관리 모듈을 만들지 않고 사용이 가능하다.

JENKINS


Project
User Interface
Jenkins
Jenkins
Plugins
Jenkins/Local
Scripts
Local
Applications
SSH Plugin
Temp Build
Workspace
Projects Build
History
Linux Compute
Cluster
Remote
Scripts
Remote
Applications
CellProfiler
Instrument
Data Shares
User
Workspace
R
R W X
R W
R W
R W
R
PROJECT
PARAMETERS
BUILD TRIGGERS
BUILD ENVIRONMENT
BUILD
POST-BUILD ACTIONS

TEKTON


Listen for changes
Kubernetes API
APIServerSource
TEKTON
Send CloudEvent
Trigger
Create
PipelineRun
Task
Pipeline
EventListener
TriggerBinding/TriggerTemplate

클라우드 네이티브에서 CI/CD

CI/CD

Jenkins/Tekton

2024-01-23

클라우드 네이티브

클라우드 네이티브는 수많은 컴퓨팅 개념 중 하나이다. 클라우드 네이티브에 제일 큰 차이점은 대다수 오픈소스 기반으로 개념들이 구성이 되어 있으며, 이를 기반으로 Cloud Native Computing Foundation(CNCF)라는 파운데이션을 만들었다.

테크톤은 CD영역에서 포함이 되며, 이를 통해서 K-Native기반의 자동화를 구성한다.

CNCF/K-Native

많이 혼돈 Cloud Native와 K-Native이다. K-Native는 쿠버네티스 기반으로 서버리스 서비스 구성이 목적이다.
K-Native는 컨테이너 기반으로 다음과 목적을 가지고 있다.

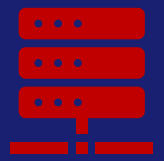
- **Simpler Abstration:** YAML기반으로 CRD구성이 가능하다.
- **Auto Scaling:** 오토 스케일링 기반으로 0부터 확장 혹은 축소가 가능하다.
- **Progressive Rollouts:** 롤 아웃 상태를 전략에 따라서 구성이 가능하다.
- **Event Integration:** 모든 소스의 이벤트에 대해서 핸들링이 가능하다.
- **Handle Events:** 작업 이벤트를 트리거를 통해서 관리한다.
- **Plugable:** 쿠버네티스에 확장 기능 추가가 가능하다.

테크톤 설치

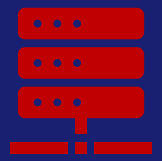
Installation on Kubernetes

가상머신 구성

자원 상태에 따라서 다르지만, 기본적으로 마스터/워커 노드 가상머신으로 하나씩 생성이 되어 있으면 됩니다. 각 가상머신에는 eth0, eth1 혹은 ens0, ens1형식으로 NIC카드 두 개를 가지고 있다. 0번은 default-network, 1번은 static-network를 구성한다.



마스터 노드



워커 노드

master/worker node
eth0: DHCP

master/worker node
master[eth1]: 10.10.20.1
worker[eth1]: 10.10.20.2

쿠버네티스 클러스터 구성

```
master/worker]# cat <<EOF>> /etc/hosts
10.10.20.1 tknmaster.example.com tknmaster
10.10.20.2 tknworker.example.com tknworker
EOF
master]# nmcli con mod eth1 ipv4.address 10.10.20.1/16 ipv4.method manual
master]# nmtui edit eth1
master]# nmcli con up eth1
worker]# nmcli con mod eth1 ipv4.address 10.10.20.2/16 ipv4.method manual
worker]# nmtui edit eth1
worker]# nmcli con up eth1
worker/master]# ip a s eth1
master]# hostnamectl set-hostname tknmaster.example.com
worker]# hostnamectl set-hostname tknworker.example.com
```

쿠버네티스 클러스터 구성(OS)

```
master/worker]# cat <<EOF>> /etc/modules-load.d/k8s-modules.conf
br_netfilter
overlay
EOF
```

쿠버네티스 클러스터 구성(OS)

```
master/worker]# modprobe br_netfilter && modprobe overlay
master/worker]# lsmod | grep -e br_netfilter -e overlay
master/worker]# cat <<EOF> /etc/sysctl.d/99-k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
master/worker]# sysctl --system
```

쿠버네티스 클러스터 구성(OS)

```
master/worker]# cat <<EOF> /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.26/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.26/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
master/worker]# dnf search --disableexcludes=kubernetes kubectl
```

쿠버네티스 클러스터 구성(K8S)

```
master]# dnf install kubeadm kubelet kubectl -y --
disableexcludes=kubernetes

node]# dnf install kubeadm kubelet -y --disableexcludes=kubernetes
```

쿠버네티스 클러스터 구성

```
master/worker]# cat <<EOF> /etc/yum.repos.d/libcontainer.repo
[devel_kubic_libcontainers_stable]
name=devel_kubic_libcontainers_stable
type=rpm-md
baseurl=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:
/stable/CentOS_9_Stream/
gpgcheck=1
gpgkey=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/
stable/CentOS_9_Stream/repodata/repomd.xml.key
enabled=1
EOF
```

쿠버네티스 클러스터 구성

```
master/worker]# cat <<EOF> /etc/yum.repos.d/crio_stable.repo
[crio]
name=cri-o for derivatives RHEL
type=rpm-md
baseurl=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/1.24:/1.24.6/CentOS_8/
gpgcheck=1
gpgkey=https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/1.24:/1.24.6/CentOS_8/repodata/repomd.xml.key
enabled=1
EOF
```

쿠버네티스 클러스터 구성

```
master]# dnf repolist
master]# dnf search crio cri-o
master/worker]# dnf install cri-o -y && systemctl enable --now crio
master/worker]# systemctl enable --now crio
master/worker]# systemctl is-active crio
> active
```

쿠버네티스 클러스터 구성

```
master/worker]# systemctl stop firewalld && systemctl disable firewalld
master/worker]# systemctl disable firewalld
master/worker]# dnf install iproute-tc -y


master/worker]# sed -i '/ swap / s/^/#/' /etc/fstab
master/worker]# sed -i s/^SELINUX=.*$/SELINUX=permissive/
/etc/selinux/config
```

쿠버네티스 클러스터 구성

```
master/node]# systemctl status crio
master/node]# systemctl disable firewalld && systemctl stop firewalld
master/node]# getenforce
master/node]# setenforce 0
master/node]# swapon -s
master/node]# swapoff -a
```

쿠버네티스 클러스터 구성

```
master/worker]# systemctl enable --now kubelet

master/worker]# curl -o /etc/containers/policy.json
https://raw.githubusercontent.com/tangt64/training_memos/main/opensource/ku
bernetes-101/files/policy.json

master]# kubeadm init --apiserver-advertise-address=10.10.20.1 \

--pod-network-cidr=10.10.0.0/16 \

--service-cidr=10.90.0.0/16

> kubeadm join 192.168.90.250:6443 --token kzu7ci.jylu1yzdcwt85c20 \

        --discovery-token-ca-cert-hash
sha256:15a5b5e9c5463ca9c359ec96c8677ddd62615fe3afcf986e4b6703e6cbcdef0b
```

쿠버네티스 클러스터 구성

```
worker]# kubeadm join 192.168.90.100:6443 --token kzu7ci.jylu1yzdcwt85c20 \
        --discovery-token-ca-cert-hash
sha256:15a5b5e9c5463ca9c359ec96c8677ddd62615fe3afcf986e4b6703e6cbcdef0b
master]# export KUBECONFIG=/etc/kubernetes/admin.conf
master]# kubectl get pods -A
master]# kubectl get nodes
```

쿠버네티스 클러스터 구성

```
master]# export KUBECONFIG=/etc/kubernetes/admin.conf
master]# kubectl get pods -A
master]# kubectl get nodes
```

쿠버네티스 클러스터 구성

```
master]# kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml

## 10분정도 기다리고 다음단계 진행

master]# curl -o custom-resources.yaml
https://raw.githubusercontent.com/tangt64/duststack-k8s-auto/master/roles/cni/cni-calico/templates/custom-resources.yaml

master]# vi custom-resources.yaml

> cidr: 10.10.0.0/16

master]# kubectl get pods -Aw

## Calico POD가 전부 올라오면 테크톤 설치 진행
```

테크톤 설치

```
master]# kubectl apply --filename https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml

master]# wget https://github.com/tektoncd/cli/releases/download/v0.32.0/tkn_0.32.0_Linux_x86_64.tar.gz

master]# kubectl apply --filename https://storage.googleapis.com/tekton-releases/dashboard/latest/release.yaml

master]# dnf install git -y

master]# git clone

master]# cd cicd-materials/session-1

master]# kubectl apply -f hello.yaml
```

hello.yaml

```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: hello
spec:
  steps:
    - image: registry.access.redhat.com/ubi8/ubi-minimal
      command:
        - /bin/bash
        - -c
        - echo "Hello World"
```

hello.yaml

```
master]# mkdir ~/bin/
master]# tar -C ~/bin/ tkn_0.32.0_Linux_x86_64.tar.gz
master]# tkn task list
> NAME    DESCRIPTION   AGE
> hello                 55 seconds ago
```

테크톤이 올바르게 동작하지 않으면 아래 명령어 실행

```
master]# kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io
config.webhook.pipeline.tekton.dev
master]# kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io
validation.webhook.pipeline.tekton.dev
master]# kubectl delete mutatingwebhookconfigurations.admissionregistration.k8s.io
webhook.pipeline.tekton.dev
```

에디터 설정(vim)

```
master]# curl -sS https://webi.sh/vim-ale | sh
master]# dnf install epel-release -y
master]# dnf search yamllint
> yamllint.noarch : A linter for YAML files
master]# dnf install yamllint -y
master]# dnf install neovim-ale -y
```

에디터 설정(nano)

```
master]# dnf search nano
master]# dnf install nano -y
master]# cat <<EOF> ~/.nanorc
set tabsize 2
set tabstospaces
EOF
master]# nano /usr/share/nano/yaml.nanorc
```

kubectl

```
master]# mkdir ~/.kube/
master]# cp /etc/kubernetes/admin.conf ~/.kube/config
master]# unset KUBECONFIG
master]# kubectl cluster-info
> Kubernetes control plane is running at https://192.168.90.250:6443
> CoreDNS is running at https://192.168.90.250:6443/api/v1/namespaces/kube-system/ser
>
> To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

kubectl(bash)

```
master]# kubectl completion bash > kubectl_completion.sh
master]# source kubectl_completion.sh
master]# kubectl completion bash > ~/kubectl_completion.bash
master]# source kubectl_completion.bash
```

kubectl(zsh)

```
master]# dnf install zsh -y
master]# usermod -s /bin/zsh root


master]# sh -c "$(curl -fsSL
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
master]# kubectl completion zsh > ~/kubectl_completion.zsh
master]# source kubectl_completion.zsh
```

tkn CLI

```
master]# tkn completion zsh > tkn_completion.zsh
master]# tkn completion bash > tkn_completion.bash
master]# source tkn_completion.bash
```

tmux

```
master]# dnf install tmux -y
master]# cat <<EOF> ~/.tmux.conf
set -g mouse on
set -g prefix C-a
unbind C-b
bind C-a send-prefix
EOF
```

Tekton dashboard

```
master]# kubectl apply --filename https://storage.googleapis.com/tekton-releases/dashboard/latest/release.yaml

master]# kubectl --namespace tekton-pipelines port-forward svc/tekton-dashboard 9097:9097

master]# kubectl --namespace tekton-pipelines port-forward svc/tekton-dashboard --address <ETH0_IP_ADDRESS> 9097:9097

host]# firefox http://<ETH0_IP_ADDRESS>:9097/
```

테크톤 자원

Stepping

Tasks

기본 자원 소개

- Step

STEP

단계(Step)은 제일 기본적인 유닛이며, 이를 기반으로 파이프라인 구성이 된다. 최소 한 개의 작업이라도 단계에 구성이 되어야 한다. 이를 통해서 CI/CD 작업 수행 단계를 효율적으로 구성 및 운영이 가능하다.

각 단계에서는 명령어를 통해서 작업이 수행이 되는데, 예를 들어서 이미지 빌드를 위해서 소스코드를 내려받기 후 컴파일 과정이 필요하면, 이 부분을 단계에 명시한다. 각 단계에는 일반적으로 명령어를 통해서 어떠한 작업을 수행할지 명시한다.

```
spec:
  steps:
    - image: registry.access.redhat.com/ubi8/ubi-minimal
      command:
        - /bin/bash
        - -c
        - echo "Hello World"
```

기본 자원 소개

- Tasks

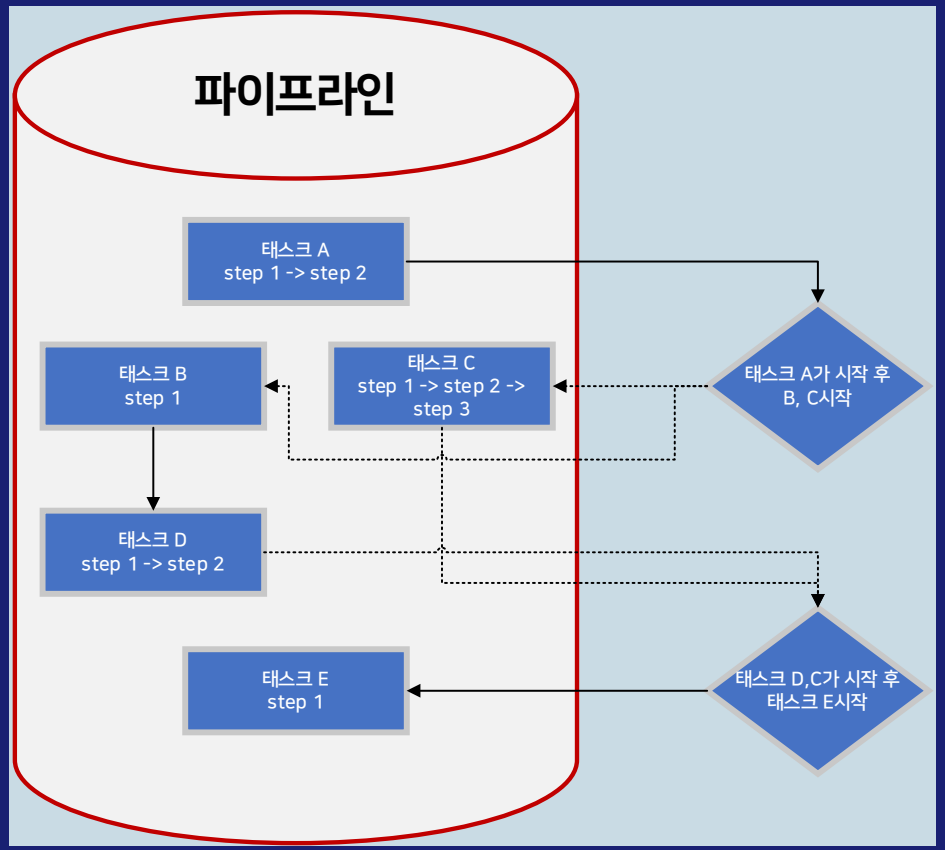
PIPELINE

파이프 라인은 **작업의 순서를 정하는 자원**이다. 쉽게 생각하면 GW-BASIC처럼, 각각 작업에 순서를 걸어서 순차적으로 작업을 수행한다. 파이프 라인은 추상적인 자원이며, 이 자원은 기본적으로 태스크를 감싸고 있다.

PIPELINE

```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: hello
spec:
  tasks:
    - name: first
      taskRef:
        name: first-task
```

PIPELINE



모든 **스텝** 및 **태스크**에는 작업 순서가 있기 때문에, 작성된 순서대로 작업이 하나씩 수행이 된다. 이러한 형태를 가지고 있는 자동화 도구는 **앤서블/솔트**와 같은 도구들이 있으며, 이들은 테크톤과 동일하게 YAML기반으로 작업을 작성 및 구성하게 된다.

TASK

태스크(tasks)는 파이프라인에 안에서는 작업에 대한 격리를 한다. 여기서 말하는 **작업**은 **단계**(step)를 이야기 하며, 각각 단계들은 태스크에서 수행이 된다. 각 태스크는 단계들의 순서(sequence)를 가지고 있으며, 이를 통해서 하나의 작업을 통해서 모든 작업을 수행한다.

예를 들어서, 쿠버네티스 클러스터에서 Pod를 생성하기 위해서 이미지 구성 및 내려받기, Pod생성과 같은 과정이 필요하다. 이러한 부분을 각각 단계로 구성하여 작업을 수행 후, 최종적으로 API를 통해서 Pod생성 및 PV/PVC와 같은 자원을 구성 및 연결을 수행한다.

이러한 **작업 단계를 모아서 동작하는 영역이 파이프라 인**이며, **작업(task)**는 **단계(step)**에 명시된 작업들을 분리 및 격리하여 수행 할 수 있도록 한다. 작업을 구성하면, 이들은 **파이프 라인을** 통해서 구성 및 실행이 된다.

TASK

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: hello
```

기본 자원 소개

- Stepping
- Task
- Pipe

STEP

스탭은 특정 작업을 수행하는 부분이다. 스탭은 다음과 같이 작업을 수행한다.

1. 컨테이너 이미지 내려받기
2. 명령어 수행
3. 작업완료

총 3단계를 통해서 작업을 한다. 이미지가 비공개 저장소에 있는 경우, "ImagePullSecret"를 통해서 내려받기가 가능하다. 작성 방법은 아래와 같이 작성이 가능하다.

STEP

```
$ vi demo-task-1.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-1
spec:
  steps:
    - image: quay.io/centos/centos
      command:
        - /bin/bash
        - -c
        - echo "Hello World"
```

STEP

```
$ kubectl apply -f demo-task-1.yaml
$ tkn task ls
NAME           DESCRIPTION   AGE
demo-task-1                  7 seconds ago
hello                        3 minutes ago
$ tkn task start demo-task-1
$ tkn task start demo-task-1 --showlog
```

STEP FOR Second-1

```
$ vi demo-multi-step.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-multi-step
```

STEP FOR Second-2

```
spec:
  steps:
    - name: first
      image: quay.io/centos/centos
      command:
        - /bin/bash
        - -c
        - echo "First step"
    - name: second
      image: quay.io/centos/centos
      command:
        - /bin/bash
        - -c
        - echo "Second step"
```

IMAGE

단계 구성 시, 사용하는 컨테이너 이미지는 사용자가 원하는 이미지로 변경이 가능하다. 여기서는 일반적으로 많이 사용하는 centos이미지 기반으로 구성하였다.

```
- name: second
    image: quay.io/centos/centos
```

STEP

```
$ tkn task list
$ tkn task start demo-multi-step --showlog
$ tkn task start demo-multi-step --showlog --no-color
```

STEP IN SCRIPT

다중 명령어를 실행하거나 혹은 스크립트를 실행하는 경우, 아래와 같이 처리가 가능하다.

```
- name: step-in-script
  image: quay.io/centos/centos
  script: |
    #!/usr/bin/env bash
    echo "Install a package"
    dnf install httpd -y
    dnf clean all
    echo "All commands ran!"
```

STEP IN SCRIPT

다중 명령어를 실행하거나 혹은 스크립트를 실행하는 경우, 아래와 같이 처리가 가능하다.

```
$ vi demo-step-script.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-step-script
spec:
  steps:
    - image: quay.io/centos/centos
      script: |
        #!/usr/bin/env bash
        echo "Install a package"
        dnf install httpd -y
        dnf clean all
        echo "All commands ran!"
```

STEP IN SCRIPT

다중 명령어를 실행하거나 혹은 스크립트를 실행하는 경우, 아래와 같이 처리가 가능하다.

```
$ kubectl apply -f demo-step-script.yaml
$ tkn task list
$ tkn task start demo-step-script --showlog
```

TASK PARAMETER

앞에서 간단하게 단계와 작업을 동시에 작성 및 실행 하였다. 하지만, 매번 작업을 구성할 때마다 단계에 들어가는 값을 변경할 수 없기 때문에, 재사용을 위해서 파라메터 형식으로 변경한다. 파라메터, 즉 변수 형태로 하는 경우, "param"항목에서만 변경하면서 사용이 가능하다. 좀 더 효율적으로 운영 및 구성

이 가능하다.

```
params:
  - name: username
    type: string
```

위와 같이 값의 이름, 그리고 유형을 적어주면, 쉘 텍스트 입력 받는 형식과 비슷하게 값을 읽어온다. 예로, 앞에서 사용하였던 명령어 실행을 예로 든다.

TASK PARAMETER

아래와 같이 작성하면, 변수를 통해서 좀 더 효율적으로 코드 사용이 가능하다.

```
command:
  - /bin/bash
  - -c
  - echo "Hello $(params.who)
```

위의 예제를 가지고 다음과 같이 테크톤 작업형식으로 작성이 가능하다.

TASK PARAMETER

```
$ vi demo-task-param.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-param
spec:
  params:
  - name: username
    type: string
  steps:
  - image: quay.io/centos/centos
    command:
      - /bin/bash
      - -c
      - echo "Hello $(params.username)"
```

TASK PARAMETER

```
$ kubectl -f demo-task-param.yaml
$ tkn task list
$ tkn task start demo-task-param --showlog
? Value for param `username` of type `string`? tang
TaskRun started: demo-task-param-run-9rpcz
Waiting for logs to be available...
[unnamed-0] Hello tang
$ tkn task start demo-task-param --showlog -p username=tang
```

TASK PARAMETER ARRAY

파라메터를 배열로 처리가 가능하다. 다만, 좀 더 복잡하게 코드를 구성해야한다. 기존 코드에 아래와 같이 파일 이름을 변경 후 작성한다.

```
$ vi demo-task-param-array.yaml
…
spec:
  params:
  - name: users
    type: array
  steps:
  - name: list of user
    image: quay.io/centos/centos
    args:
      - $(params.users[*])
    command:
      - /bin/bash
      - -c
      - echo "Hello ${params.users[*]}"
```

TASK PARAMETER ARRAY

파라메터를 배열로 처리가 가능하다. 다만, 좀 더 복잡하게 코드를 구성해야한다. 기존 코드에 아래와 같이 파일 이름을 변경 후 작성한다.

```
$ vi demo-task-param-array.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-param-array
spec:
```

TASK PARAMETER DEFAULT

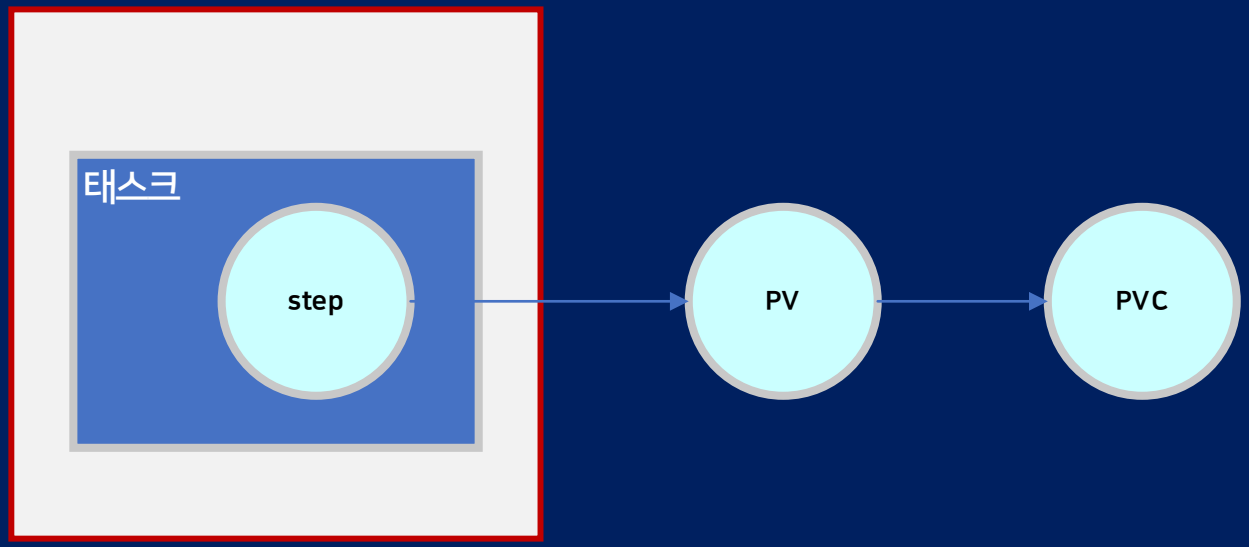
```
params:
- name: username
  type: string
  default: tang
steps:
- image: quay.io/centos/centos
  command:
    - /bin/bash
    - -c
    - echo "Hello $(params.username)"
```

TASK PARAMETER DEFAULT

```
$ kubectl apply -f demo-task-param-array.yaml
$ tkn task list
$ tkn task start demo-task-param-array --showlog --use-param-defaults
```

공유 데이터

테크톤의 테크가 동작하면 컨테이너가 하나의 포드(Pod)에서 동작하기 때문에, 데이터 공유가 가능하다. 데이터를 공유 하기 위해서 쿠버네티스의 PV/PVC를 통해서 공유가 가능하다.



공유 데이터(HOME) 1

```
$ vi tekton-share-home.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: tekton-share-home
```

공유 데이터(HOME) 2

```
spec:
  steps:
  - name: write
    image: quay.io/centos/centos
    script: |
      cd ~
      echo "location is $(pwd)"
      echo "Hello form Tekton" > tkn-message.txt
  - name: read
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args: ["-c", "cat ~/tkn-message.txt"]
```

공유 데이터(HOME) 3

```
$ kubectl apply -f tekton-share-home.yaml
$ echo "this is root home" > ~/tkn-message.txt
$ tkn task list
$ tkn task start tekton-share-home --showlog
```

참고

위의 기능은 최신 테크톤 버전에서는 동작하지 않는다. 자세한 정보는 아래 링크를 참고한다.

https://github.com/tektoncd/pipeline/pull/3878

공유 데이터(RESULT)

특정 작업이 수행 후, 결과값에 대해서 저장을 하기 위해서 "result"라는 지시자 사용이 가능하다. 결과(result)를 사용하면, 완료된 작업의 내용을 "/tekton/results"에서 접근 및 사용이 가능하다. 결과는 쿠버네티스의 PV/PVC를 사용하지 않는다.

이전에 사용한 내용을 응용하여 "결과(result)"를 사용하도록 한다.

TEKTON RESUTL-1

```
$ vi tekton-result.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: tekton-result
spec:
  results:
    - name: welcome
      description: welcome message
```

TEKTON RESUTL-2

```
steps:
  - name: write
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args:
      - "-c"
      - echo "Welcome to Tekton world" > $(results.welcome.path)
  - name: read
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args:
      - "-c"
      - cat $(results.welcome.path)
```

TEKTON RESUTL-3

```
$ kubectl apply -f tekton-result.yaml
$ tkn task list
$ tkn task start tekton-result --showlog
```

KUBERNETES VOLUME

쿠버네티스에서 사용하는 볼륨 자원, secret, configmap 그리고 Persistent Volume, Persistent Volume Claim 전부 사용이 가능하다. 여기에서는 간단하게 쿠버네티스 볼륨 자원들을 테크톤에서 활용해보도록 한다.

더 많은 자원은 뒤에서 추후 활용 및 다루기로 한다.

1. `secret`
2. `configmap`
3. `PV/PVC`

TEKTON CONFIGMAP

```
$ vi kubernetes-demo-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-list
data:
  user1: helix
  user2: tang
  user3: suse
```

TEKTON CONFIGMAP

```
$ vi tekton-demo-configmap.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: tekton-demo-configmap
spec:
  volumes:
    - name: users
      configmap:
        name: user-list
```

TEKTON CONFIGMAP

```
steps:
- name: username-list
  image: quay.io/centos/centos
  volumeMounts:
    - name: users
      mountPath: /var/username-list
  script: |
    echo "$(cat /var/username-list/user1) is normal user"
    echo "$(cat /var/username-list/user2) is root user"
    echo "$(cat /var/username-list/user3) is admin user"
```

TEKTON SECRET

```
$ kubectl apply -f kubernetes-demo-configmap.yaml
$ kubectl apply -f tekton-demo-configmap.yaml
$ tkn task list
$ tkn task start tekton-configmap --showlog
```

디버깅

테크톤

DIGGING and DIGGING

디버깅

```
$ tkn task start tekton-configmap --showlog

$ kubectl get taskrun tekton-configmap-run-<ID> -o yaml
```

디버깅

```
$ vi failforfail.yaml
```

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: failforfail
spec:
  steps:
  - image: quay.io/dollar/dollar
    command:
      - echo "Hello failure"
```

```
$ kubectl apply -f failforfail.yaml
$ tkn task start failforfail --showlog
$ kubectl get tr
$ kubectl get tr failforfail-run-<ID> -o yaml
```

기본 자원

연습문제

파이프라인

소개

소개

파이프라인은 앞서 사용하였던 단계(Step), 작업(Task)를 조합하여 실행한다. 또한, 파이프라인은 쿠버네티스에서 CI/CD인터페이스를 통해서 작업을 수행 시, 테크톤은 파이프라인을 호출하여 작업을 실행 및 수행한다. 예를 들어서 이전에 생성한 작업에 "복제", "컴파일", "검증"과 같은 작업이 있었다면, 이를 태스크를 통해서 작업을 수행한다.

태스크의 주요 목적은 최대한 자원을 재사용과 손쉽게 작업을 사용자가 원하는 순서대로 구성하는 게 주요 목적이다. 파이프 라인도 YAML기반으로 작성이 된다.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
name: demo-pipeline
labels:
  key: value
spec:
```

파이프 라인 구성(task)

```
$ vi demo-pipeline1-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-pipeline1-task
spec:
  steps:
    - image: quay.io/centos/centos
      command:
      - /bin/bash
      args: ['-c', 'echo I am the demo pipeline task']
```

파이프 라인 구성(task)

```
$ vi demo-pipeline1-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline1-pipeline
spec:
  task:
  - name: first
    taskRef:
      name: demo-pipeline1-task
```

파이프 라인 구성(task)

```
$ kubectl apply -f demo-pipeline1-task.yaml
$ kubectl apply -f demo-pipeline1-pipeline.yaml
$ tkn pipeline list
$ tkn task list
$ tkn task start demo-pipeline1-task --showlog
$ tkn pipeline start demo-pipeline1-pipeline --showlog
```

DICE TASK-1

```
$ vi demo-pipeline-dice-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-pipeline-dice-task
spec:
  steps:
  - name: welcome-message
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args: ['-c', 'echo Rolling dice']
```

DICE TASK-2

```
- name: dice-random
  image: quay.io/centos7/php-73-centos7
  command:
    - /bin/bash
  args: ['-c', 'php -r "echo rand(1,6);"']
```

WELCOME-MESSAGE TASK

```
$ vi demo-pipeline-welcome-message-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: welcome-message
spec:
  steps:
  - name: welcome-message
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args: ['-c', 'echo "Welcome Gambling world!"']
```

DICE PIPELINE

```
$ vi demo-pipeline-dice-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline-dice-pipeline
spec:
  tasks:
  - name: message
    taskRef:
      name: welcome-message
  - name: roll
    taskRef:
      name: demo-pipeline-dice-task
```

DICE RUN

```
$ kubectl apply -f demo-pipeline-dice-task.yaml
$ kubectl apply -f demo-pipeline-dice-pipeline.yaml
$ kubectl apply -f demo-pipeline-welcome-message-task.yaml
$ tkn task list
$ tkn pipeline list
$ tkn task start welcome-message --showlog
$ tkn task start demo-pipeline-dice-task --showlog
$ tkn pipeline start demo-pipeline-dice-pipeline --showlog
$ tkn pipeline logs
```

파이프라인 파라메터(DEMO)

파이프라인을 지속적으로 반복하다 보면, 같은 작업에 다른 요청 값을 통해서 작업이 수행이 된다. 이러한 형태는 쿠버네티스에서 빈번하게 발생이 된다. 테크톤 기반의 CI/CD에서는 이를 파라메터(Parameter)를 통해서 쉽게 구현이 가능하다.

```
$ vi demo-pipeline-dice-pipeline2-task-param.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-pipeline-dice-pipeline2-task-param
spec:
  params:
    - name: sideNum
      description: number of sides to the dice
      default: 6
      type: string
```

파이프라인 파라메터(DEMO)

```
  steps:
  - name: welcome-message-param
    image: quay.io/centos/centos
    command:
      - /bin/bash
      - -c
      - echo "Rolling $(params.sideNum) dice"
- name: dice-random-param
    image: quay.io/centos7/php-73-centos7
    script: |
      php -r 'echo rand(1,$(params.sideNum));'
```

DICE 파이프라인 2

```
$ vi demo-pipeline-dice-pipeline2.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline-dice-pipeline2
spec:
  tasks:
  - name: first
    taskRef:
      name: hello
  - name: roll
    taskRef:
      name: demo-pipeline-dice-pipeline2-task-param
```

파이프라인 파라메터

```
$ kubectl apply -f demo-pipeline-dice-pipeline2.yaml
$ kubectl apply -f demo-pipeline-dice-pipeline2-task-param.yaml
$ tkn pipeline start demo-pipeline-dice-pipeline2 --showlog
```

파이프라인 파라메터 변경

```
$ vi emo-pipeline-dice-pipeline2-task-param.yaml
> default: 6 -> 8
$ kubectl apply -f emo-pipeline-dice-pipeline2-task-param.yaml
$ tkn pipeline start demo-pipeline-dice-pipeline2 --showlog
```

DICE 파이프라인 3

```
$ vi demo-pipeline-dice-pipeline3.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline-dice-pipeline3
spec:
  params:
  - name: sidenum
    type: string
    default: 6
    description: number of sides the dice
```

DICE 파이프라인 3

```
tasks:
- name: first
  taskRef:
    name: hello
- name: roll
  params:
  - name: sides
    value: "$(params.sidenum)"
  taskRef:
    name: demo-pipeline-dice-task
```

파이프라인 파라메터

```
$ kubectl apply -f demo-pipeline-dice-pipeline3.yaml
$ tkn pipeline start demo-pipeline-dice-pipeline3 --showlog
$ tkn pipeline start demo-pipeline-dice-pipeline3 -p sidenum=20 --showlog
$ tkn pipeline start demo-pipeline-dice-pipeline3 --use-param-defaults --
showlog
```

REUSE PIPELINE DEMO

```
$ vi demo-pipeline-reuse-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-pipeline-reuse-task
spec:
  params:
  - name: text
    type: string
```

REUSE PIPELINE DEMO(task)

```
steps:
- name: log
  image: quay.io/centos/centos
  script: |
    DATE=$(date +%d/%m/%Y\ %T)
    echo [$DATE] - $(params.text)
```

REUSE PIPELINE DEMO(pipeline)

```
$ vi demo-pipeline-reuse-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline-reuse-pipeline
spec:
  tasks:
  - name: say-hello
    taskRef:
      name: demo-pipeline-reuse-task
    params:
    - name: text
      value: "Hello"
```

REUSE PIPELINE DEMO(pipeline)

```
- name: log-something
    taskRef:
      name: hello
    params:
    - name: text
      value: "Something else being logged"
```

REUSE PIPELINE DEMO(RUN)

```
$ kubectl apply -f demo-pipeline-reuse-task.yaml
$ kubectl apply -f demo-pipeline-reuse-pipeline.yaml
$ tkn pipeline start demo-pipeline-reuse-pipeline --showlog
```

PIPELINE ORDERING

파이프라인 소개 시, 파이프라인은 자유롭게 작업 순서를 변경(ordering)이 가능하다. 이를 통해서 각 작업마다 원하는 방식으로 작업 순서를 지정할 수 있다. 이미 위에서 기능 학습을 하였지만, 작업(task) 및 단계(step)은 위치에 따라서 실행하는 순서가 달라진다.

그래서 태스크에서 구성한 순서가 아닌, 사용자가 원하는 순서대로 진행을 하고 싶은 경우, YAML코드에 사용자가 원하는 작업 이름을 순서대로 명시하면 된다. 아래 간단하게 여러 개의 작업을 구성하여 파이프라인을 통해서 실행 순서를 재구성 한다.

PIPELINE ORDERING

파이프라인 소개 시, 파이프라인은 자유롭게 작업 순서를 변경(ordering)이 가능하다. 이를 통해서 각 작업마다 원하는 방식으로 작업 순서를 지정할 수 있다. 이미 위에서 기능 학습을 하였지만, 작업(task) 및 단계(step)은 위치에 따라서 실행하는 순서가 달라진다.

그래서 태스크에서 구성한 순서가 아닌, 사용자가 원하는 순서대로 진행을 하고 싶은 경우, YAML코드에 사용자가 원하는 작업 이름을 순서대로 명시하면 된다. 아래 간단하게 여러 개의 작업을 구성하여 파이프라인을 통해서 실행 순서를 재구성 한다.

PIPELINE ORDERING - TASK

```
$ vi pipeline-ordering-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: pipeline-ordering-task
spec:
```

PIPELINE ORDERING - TASK

```
params:
  - name: task-name
    type: string
  - name: time
    type: string
    default: ""
```

PIPELINE ORDERING - TASK

```
steps:
  - name: first-task
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args:
      - "-c"
      - "echo Ran to the first task $(params.task-name)"
```

PIPELINE ORDERING - TASK

```
- name: second-task
  image: quay.io/centos/centos
  command:
    - /bin/bash
  args:
    - "-c"
    - "echo Ran to the second task $(params.time)"
```

PIPELINE ORDERING - TASK

```
  - name: logger
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args:
      - "-c"
      - "echo Ran to the last task [$(date '+%d/%m/%Y %T')] - Task
$(params.task-name) Completed"
```

PIPELINE ORDERING - PIPELINE

```
$ vi pipeline-ordering-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: pipeline-ordering-pipeline
spec:
```

PIPELINE ORDERING - PIPELINE

```
tasks:
  - name: first
    params:
      - name: task-name
        value: A
      - name: time
        value: 2
```

PIPELINE ORDERING - PIPELINE

```
- name: second
  params:
    - name: task-name
      value: B
  taskRef:
    name: first task
```

PIPELINE ORDERING - PIPELINE

```
- name: third
  params:
    - name: task-name
      value: C
    - name: time
      value: 3
  taskRef:
    name: second-task
```

PIPELINE ORDERING - PIPELINE

```
- name: fourth
  params:
    - name: task-name
      value: D
  taskRef:
    name: logger
```

PIPELINE ORDERING - RUN

```
$ kubectl apply -f pipeline-ordering-task.yaml
$ kubectl apply -f pipeline-ordering-pipeline.yaml
$ tkn pipeline start pipeline-ordering-pipeline --showlog
```

2024-01-23

PIPELINE ORDERING - RESULT(TASK)

```
$ vi demo-pipeline-dice-task-param-result.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-pipeline-dice-task-param-result
spec:
```

PIPELINE ORDERING - RESULT(TASK)

```
param:
  - name: sideNum
    description: number of sides to the dice
    default: 6
    type: string
results:
  - name: dice-result
    description: result of dice roll number
```

PIPELINE ORDERING - RESULT(TASK)

```
steps:
  - name: rolling-n-rolling-dice
    image: quay.io/centos/centos
    script: |
      php -r 'echo rand(1,$(params.sideNum));' > dice-result.txt
```

PIPELINE ORDERING - RESULT(PIPELINE)

```
$ vi demo-pipeline-dice-pipeline-result.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline-dice-pipeline-result
```

PIPELINE ORDERING - RESULT(PIPELINE)

```
params:
  - name: sideNum
    type: string
    default: 6
```

PIPELINE ORDERING - RESULT

```
spec:
tasks:
  - name: first
    params:
      - name: text
        value: "The dice sides is $(params.sideNum)"
    taskRef:
      name: logger
```

PIPELINE ORDERING - RESULT

```
- name: roll
  params:
  - name: sides
    value: "$(params.sideNum)"
  taskRef:
    name: demo-pipeline-dice-param-result
  runAfter: first
```

PIPELINE ORDERING - RESULT

```
$ kubectl apply -f demo-pipeline-dice-task-param-result.yaml

$ kubectl apply -f demo-pipeline-dice-pipeline-result.yaml demo-pipeline-dice-pipeline-result.yaml

$ tkn pipeline start demo-pipeline-dice-task-param-result --showlog
```

PIPELINE RUNS

```
$ tkn pipeline start demo-pipeline-dice-task-param-result
$ kubectl get pipelineruns
>
$ kubectl delete pipelineruns <ID>
>
$ kubectl get pipelineruns <ID> -o yaml
```

파이프라인 연습문제

연습문제

2024-01-23

디버깅

디버깅

파이프라인 및 태스크 정리

2024-01-23

DEBUG-TASK

```
$ vi exit-debug-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: exit-debug-task
spec:
params:
  - name: text
    type: string
  - name: exitcode
    type: string
```

DEBUG-TASK

```
steps:
  - name: log
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args: ["-c", " echo $(params.text)"]
  - name: exit
    image: quay.io/centos/centos
    command:
      - /bin/bash
    args: ["-c", "echo 'Exiting with code $(params.exitcode)' && exit $(params.exitcode)"]
```

DEBUG-TASK

```
$ vi exit-debug-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: exit-debug-pipeline
spec:
  tasks:
  - name: clone
    taskRef:
      name: log-and-exit
    params:
      - name: text
        value: "Simulating git clone"
      - name: exitcode
        value: "0"
```

DEBUG-TASK

```
- name: unit-tests
  taskRef:
    name: exit-debug-task
  params:
    - name: text
      value: "Simulating unit testing"
    - name: exitcode
      value: "0"
  runAfter:
    - clone
```

DEBUG-TASK

```
- name: deploy
  taskRef:
    name: exit-debug-task
  params:
    - name: text
      value: "Simulating deployment"
    - name: exitcode
      value: "0"
  runAfter:
    - unit-tests
```

DEBUG-TASK

```
- name: deploy
  taskRef:
    name: exit-debug-task
  params:
    - name: text
      value: "Simulating deployment"
    - name: exitcode
      value: "0"
  runAfter:
    - unit-tests
```

DEBUG-TASK RUN 1

```
$ kubectl -f exit-debug-task.yaml
$ kubectl -f exit-debug-pipeline.yaml
$ tkn pipeline start exit-debug-task --showlog
```

DEBUG-TASK VALUE

```
- name: unit-tests
  taskRef:
    name: exit-debug-task
  params:
    - name: text
      value: "Simulating unit testing"
    - name: exitcode
      value: "0" → "1"
  runAfter:
    - clone
```

DEBUG-TASK RUN 2

```
$ kubectl -f exit-debug-pipeline.yaml
$ tkn pipeline start exit-debug-task --showlog
```

FINALLY/CLEANUP

FINALLY/CLEANUP

```
$ vi task-cleanup.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: task-cleanup
spec:
```

FINALLY/CLEANUP

```
steps:
- name: clean
  image: quay.io/centos/centos
  command:
    - /bin/bash
  args: ['-c', 'echo Cleaning up!']
```

FINALLY/CLEANUP

```
$ vi exit-debug-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: exit-debug-pipeline
spec:
  tasks:
…
  finally:
  - name: cleanup-task
    taskRef:
      name: cleanup
```

FINALLY/CLEANUP RUN

```
$ kubectl apply -f task-cleanup.yaml
$ kubectl apply -f exit-debug-pipeline.yaml
$ tkn pipeline start exit-debug-pipeline --showlog
```

디버깅 연습문제

연습문제

파이프라인

Pipeline

파이프라인

파이프라인 소개

파이프라인

파이프라인 기반으로 작업 구성 및 사용하기

파이프라인

파이프라인에서 작업 순서

파이프라인

파이프라인에서 결과 활용하기

파이프라인

파이프라인 실행(runs)

디버깅

테스크/파이프 라인

디버깅

파이프라인 디버깅

디버깅

태스크 상태 확인

디버깅

최종작업 추가

워크스페이스

데이터 공유

워크스페이스

소개

워크스페이스

저장소 형식

워크스페이스

첫 워크스페이스 생성

워크스페이스

워크스페이스에서 작업 실행(task runs) 사용하기

워크스페이스

워크스페이스 파이프라인 추가

2024-01-23

워크스페이스

파이프라인에 영구적 데이터 저장소 구성

워크스페이스

최종(finally)를 사용한 작업 정리

2024-01-23

워크스페이스

파이프라인 실행(pipeline runs) 사용하기

워크스페이스

템플릿에 쿠버네티스 볼륨 클레임 사용하기

조건식

조건식 활용 및 사용하기

조건식

소개

조건식

조건식, 파라메터 사용하기

조건식

조건식, 파라메터 사용하기

2024-01-23

2024-01-23

조건식

값 입력 받기

조건식

결과값 기반으로 조건식 작성

2024-01-23

인증

기본 인증 및 SSH

조건식

소개

트리거

조건식 활용 및 사용하기

2024-01-23

트리거

소개

트리거

설치

트리거

로컬 클러스터

트리거

트리거 템플릿/바인딩/이벤트 리스너

2024-01-23

트리거

트리거 수행 및 실행

랩

최종학습 랩

2024-01-23