

Seldon core 실습 자료

1. Prerequisite 개념 간단 정리

- 1) [Custom Resource](#)
- 2) [Operator pattern](#)
- 3) [Helm](#)

2. Seldon Core 설치

- 1) [minikube](#)
- 2) [helm](#)
- 2) [ambassador](#)
- 3) [Seldon-core](#)

3. Quick Start

- 1) [Seldon Deployment 생성](#)
 - 2) [API 문서 확인](#)
 - 3) [Let's Send API Request](#)
-

1. Prerequisite 개념 간단 정리

쿠버네티스 관련 사전 지식이 다소 필요합니다. 익숙지 않다면 Seldon Core 의 사용 방식에 대해서만 익히고 넘어가셔도 충분합니다.

1) Custom Resource

- Official docs
 - <https://kubernetes.io/ko/docs/concepts/extend-kubernetes/api-extension/custom-resources/>
- Custom Resource (CR) 은 쿠버네티스의 API 의 확장판입니다.
 - 쿠버네티스에서 기본적으로 관리하는 리소스들에는 Pod, Deployment, Service, PersistentVolume 등이 있습니다.
 - 하지만 유저가 직접 정의한 리소스를 쿠버네티스의 API 를 사용해서 관리하고 싶은 경우에는 Custom Resource 와 해당 CR 의 LifeCycle 과 동작을 관리할 Controller (혹은 API Server) 를 구현 후 쿠버네티스 클러스터에 배포해야 합니다.
 - CR 을 클러스터에 등록하는 방법에는 Custom Resource Definition (CRD) 방식과 API Aggregation (AA) 방식 두 가지가 있지만, 이번 강의에서 다루는 모든 모듈은 CRD 방식을 사용합니다.

- CRD 방식은 CR 을 관리할 Custom Controller 를 구현하고 배포하여 사용하게 되며, Controller 는 대부분 Operator pattern 으로 개발됩니다.
- 한 마디로 압축하면 쿠버네티스에서 default 로 관리하지는 않지만, 배포된 Custom Controller 에 의해 쿠버네티스에서 관리되고 있는 리소스들이라고 할 수 있습니다.

2) Operator pattern

- Official docs
 - <https://kubernetes.io/ko/docs/concepts/extend-kubernetes/operator/>
- Controller
 - Desired State 와 Current State 를 비교하여, **Current State 를 Desired State 에 일치시키도록 지속적으로 동작**하는 무한 루프
 - <https://kubernetes.io/ko/docs/concepts/architecture/controller/>
- Operator
 - **Controller pattern 을 사용하여 사용자의 애플리케이션을 자동화**하는 것
 - 주로 CR 의 Current/Desired State 를 지속적으로 관찰하고 일치시키도록 동작하는 역할을 위해 사용됩니다.
- Operator 개발 방법
 - Operator 개발에 필요한 부수적인 작업이 자동화되어있는 Framework 를 활용하여 개발
 - [kubebuilder](#), [KUDO](#), [Operator SDK](#)
 - 앞으로 다룰 seldon-core, prometheus, grafana, kubeflow, katib 를 포함해 쿠버네티스 생태계에서 동작하는 **많은 모듈들**이 이러한 Operator 로 개발되어 있습니다.

3) Helm

- Official docs
 - <https://helm.sh/docs/>
- 쿠버네티스 모듈의 **Package Managing Tool**
 - Ubuntu OS 의 패키지 관리 도구 `apt`, Mac OS 의 패키지 관리 도구 `brew`, Python 패키지 관리 도구 `pip` 와 비슷한 역할
- 하나의 쿠버네티스 모듈은 다수의 리소스들을 포함하고 있는 경우가 많습니다.

- 즉, `a.yaml`, `b.yaml`, `c.yaml`, ... 등 많은 수의 쿠버네티스 리소스 파일들을 모두 관리해야 하기에 버전 관리, 환경별 리소스 파일 관리 등이 어렵습니다.
 - Helm 은 이러한 작업을 템플릿화시켜서 많은 수의 리소스들을 마치 하나의 리소스처럼 관리할 수 있게 도와주는 도구라고 할 수 있습니다.
 - Helm manifest 는 크게 `templates` 와 `values.yaml` 로 이루어져 있으며, `templates` 폴더에는 해당 모듈에서 관리하는 모든 쿠버네티스 리소스들의 템플릿 파일이 보관됩니다.
 - 또한 `values.yaml` 이라는 인터페이스로부터 사용자에게 값을 입력받아 `templates` 의 정보와 merge 하여 배포됩니다.
-

2. Seldon Core 설치

- Official docs
 - <https://docs.seldon.io/projects/seldon-core/en/latest/workflow/install.html>
- Prerequisites
 - 쿠버네티스 환경 (v1.18 이상)
 - minikube
 - kubectl
 - Helm 3
 - Ingress Controller
 - Ambassador
 - Python 환경
 - python 3.6 이상
 - pip3

1) minikube

```
minikube start --driver=docker --cpus='4' --memory='4g'
```

2) helm

- helm version v3.5.4

- v3.0 이상

```
# https://github.com/helm/helm/releases 에서 link 확인
wget <URI>

# 압축 풀기
tar -zxvf helm-v3.5.4-linux-amd64.tar.gz

# 바이너리 PATH 로 이동
mv linux-amd64/helm /usr/local/bin/helm

# helm 정상 동작 확인
helm help
```

2) ambassador

- chart version : ambassador-6.9.1

```
# ambassador 를 install 하기 위해 public 하게 저장된 helm repository 를 등록
helm repo add datawire https://www.getambassador.io

# helm repo update
helm repo update

# helm install ambassador with some configuration
helm install ambassador datawire/ambassador \
  --namespace seldon-system \
  --create-namespace \
  --set image.repository=quay.io/datawire/ambassador \
  --set enableAES=false \
  --set crds.keep=false

# 정상 설치 확인
kubectl get pod -n seldon-system -w
kubectl get pod -n seldon-system
```

3) Seldon-core

- chart version : seldon-core-operator-1.11.0

```
helm install seldon-core seldon-core-operator \
  --repo https://storage.googleapis.com/seldon-charts \
  --namespace seldon-system \
```

```
--create-namespace \  
--set usageMetrics.enabled=true \  
--set ambassador.enabled=true
```

3. Quick Start

- **SeldonDeployment** 는 Seldon-Core 에서 정의한 Custom Resource 중 하나입니다.
 - 간단하게 말하면 이미 학습이 완료된 model 을 로드해서 Serving 하는 Server 를 쿠버네티스에서는 SeldonDeployment 라고 부르고 관리할 수 있습니다.
 - Flask 를 사용하는 경우에 필요했던 작업인 API 를 정의하거나, IP, PORT 를 정의하거나, API 문서를 작성해야하는 작업부터, 쿠버네티스에 배포하기 위해 필요했던 docker build, push, pod yaml 작성 후 배포와 같은 작업을 할 필요 없이, trained model 파일이 저장된 경로만 전달하면 자동화된 것이라고 볼 수 있습니다.

그럼 이제 Sample Seldon Deployment 를 생성해보고, 생성된 SeldonDeployment 서버가 제공하는 URI 로 http request 를 호출하여 정상적으로 응답이 오는지 확인해보겠습니다.

1) Seldon Deployment 생성

- Seldon Deployment 를 생성할 용도의 namespace 를 하나 생성합니다.

```
kubectl create namespace seldon
```

- SeldonDeployment YAML 파일 생성

- `vi sample.yaml`

```
apiVersion: machinelearning.seldon.io/v1  
kind: SeldonDeployment  
metadata:  
  name: iris-model  
  namespace: seldon  
spec:  
  name: iris  
  predictors:
```

```
- graph:
  implementation: SKLEARN_SERVER # seldon core 에서 sklearn 용으로 pre-package 된
model server
  modelUri: gs://seldon-models/v1.11.0-dev/sklearn/iris # seldon core 에서 제공
하는 open source model - iris data 를 classification 하는 모델이 저장된 위치 : google st
orage 에 이미 trained model 이 저장되어 있습니다.
  name: classifier
  name: default
  replicas: 1 # 로드밸런싱을 위한 replica 개수 (replica 끼리는 자동으로 동일한 uri 공유)
```

- `kubectl apply -f sample.yaml`

- minikube tunnel

- 새로운 터미널을 열어 `minikube tunnel` 을 수행합니다.
- minikube tunnel 이 열려있으면 minikube cluster 내부와 통신할 수 있게 됩니다.
 - 본 예제에서는 ambassador 와 같은 loadbalancer service 를 expose 하여 clusterIP 를 externalIP 처럼 사용하기 위한 용도로 활용합니다.

- ambassador external IP 확인

- `kubectl get service -n seldon-system`
 - `ambassador` 의 `EXTERNAL IP` 확인

2) API 문서 확인

- Seldon Deployment 를 생성하면 다음 주소에서 API Reference 를 확인할 수 있습니다.
 - `http://<ingress_url>/seldon/<namespace>/<model-name>/api/v1.0/doc/`
 - 해당 문서에는 해당 SeldonDeployment 에서 지원하는 API 와 해당 API 의 사용법에 대한 예시가 포함되어 있습니다.

Seldon External API 0.1 OAS3

[./seldon.json](#)
[Seldon Core - Website](#)
[Seldon Core Documentation](#)

[Authorize](#)

External Ambassador API

POST /seldon/seldon/iris-model/api/v1.0/feedback

POST /seldon/seldon/iris-model/api/v1.0/predictions

Parameters [Cancel](#)

No parameters

Request body required application/json

```
{
  "data": {
    "names": [
      "feature1"
    ],
    "tensor": {
      "Shape": [
        1
      ],
      "values": [
        1
      ]
    }
  }
}
```

[Execute](#) [Clear](#)

3) Let's Send API Request

- With `curl`

```
# curl -X POST http://<ingress>/seldon/seldon/iris-model/api/v1.0/predictions
curl -X POST http://10.100.33.197/seldon/seldon/iris-model/api/v1.0/predictions \
  -H 'Content-Type: application/json' \
  -d '{ "data": { "ndarray": [[1,2,3,4]] } }'
```

다음과 같은 메시지가 출력됩니다.

```
# {"data":{"names":["t:0","t:1","t:2"],"ndarray":[[0.0006985194531162835,0.00366803903
943666,0.995633441507447]]},"meta":{"requestPath":{"classifier":"seldonio/sklearnserve
r:1.11.0"}}}
```

multi-class classification 에서 type 2 일 확률이 가장 높다는 결과를 return 함을 확인할 수

```
# 잘못된 data 형식으로 API 를 보내보겠습니다.
curl -X POST http://10.100.33.197/seldon/seldon/iris-model/api/v1.0/predictions \
  -H 'Content-Type: application/json' \
  -d '{ "data": { "ndarray": [[1,2,3,4,5]] } }'
```

다음과 같은 '에러' 메시지가 출력됩니다.

```
# {"status":{"code":-1,"info":"Unknown data type returned as payload (must be list or
np array):None","reason":"MICROSERVICE_BAD_DATA","status":1}}
```

- With Python Client

```
# python 가상환경 활성화
python -V

# pypi 패키지 설치
pip install numpy,seldon-core
```

- `vi test.py`

```
import numpy as np

from seldon_core.seldon_client import SeldonClient

sc = SeldonClient(
    gateway="ambassador",
    transport="rest",
    gateway_endpoint="10.100.33.197:80", # Make sure you use the port above
    namespace="seldon",
)

client_prediction = sc.predict(
    data=np.array([[1, 2, 3, 4]]),
    deployment_name="iris-model",
    names=["text"],
    payload_type="ndarray",
)

print(client_prediction)
```

- 위의 python code 를 실행해보겠습니다.

- `python test.py`

- 다음과 같은 메시지가 출력됩니다.

```
Success:True message:
Request:
meta {
}
data {
  names: "text"
  ndarray {
    values {
      list_value {
        values {
          number_value: 1.0
        }
        values {
          number_value: 2.0
        }
      }
    }
  }
}
```



```
    }  
    values {  
      number_value: 3.0  
    }  
    values {  
      number_value: 4.0  
    }  
  }  
}  
}
```

Response:

```
{'data': {'names': [], 'ndarray': [2.0]}, 'meta': {'requestPath': {'classifier': 'seldonio/xgboostserver:1.11.0'}}}
```