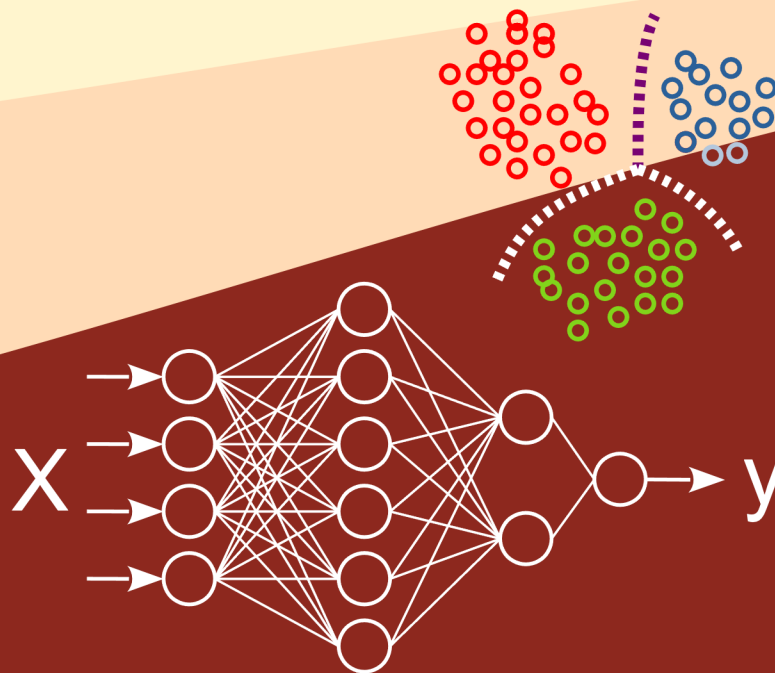
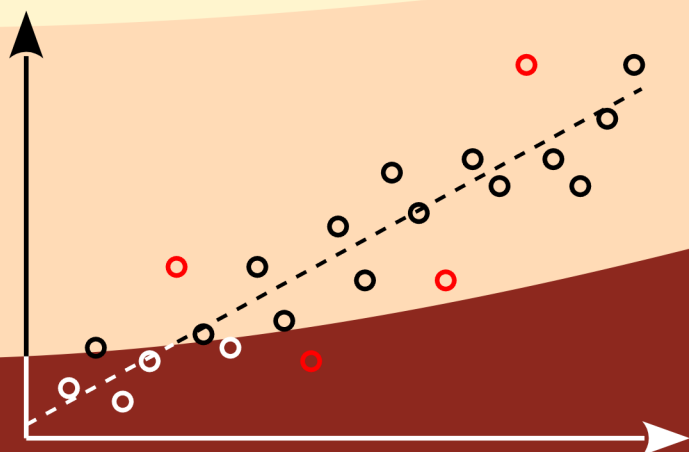


# Arhitektura i Razvoj Inteligentnih Sustava

## Tjedan 8: Poslužitelji modela



# Creative Commons



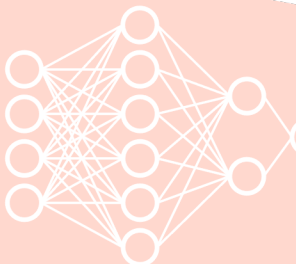
- slobodno smijete:

- dijeliti — umnožavati, distribuirati i javnosti priopćavati djelo
- prerađivati djelo



- pod sljedećim uvjetima:

- imenovanje: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- nekomercijalno: ovo djelo ne smijete koristiti u komercijalne svrhe.
- dijeli pod istim uvjetima: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, prerađu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



*U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

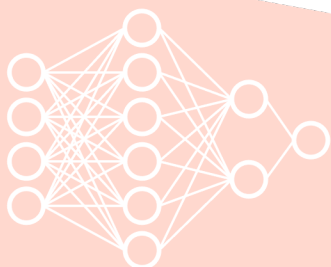
*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

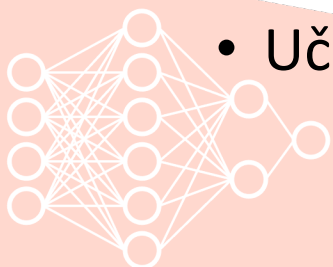
# Još malo o security-u

- ML nije imun na sigurnosne napade
  - Ima raznih motiva za napade na ML infrastrukturu
  - Recimo modeli koje koristimo kod inteligentnih vatrozida
  - Modeli za preporuke
  - Krađe modela
- *Data poisoning* – Napadač utječe na podatke koji se koriste za učenje
  - Posebno su ranjivi sustavi koji ovise o korisnički generiranim podacima – web shop recimo
  - Napadač namjerno koristi fiktivne pretrage recimo kako bi onečistio podatke za učenje



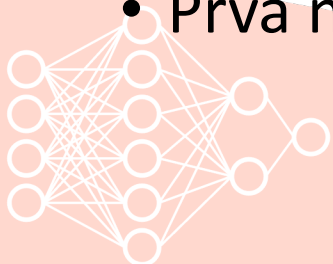
# Još malo o security-u

- *Byzantine attack*
  - Povezan s *byzantine worker* slučajem
  - Paralelno učenje u distribuiranoj okolini – napadač namjerno pretvara neke čvorove u bizantske radnike koji odmažu u učenju modela
- *Evasion* – Zaobilaženje odluka modela
  - Recimo kod inteligentnih vatrozida, anti-spam sustava, anti-virusnih sustava i slično
  - Napadač maskira zahtjev na način da model ne prepozna napad
- *Model extraction* – Nizom zahtjeva, napadač bilježi odluke modela
  - Time se dobiva skup podataka za učenje
  - Učenjem dobivamo sličan model – krađa modela



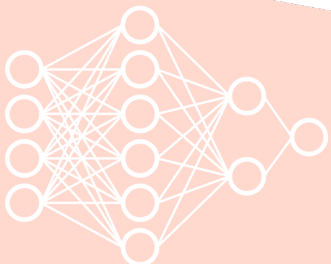
# Posluživanje modela

- Prošlo predavanje smo govorili o jednostavnom ručnom izlaganju modela kroz mikroservis – korištenjem *flask* python modula
  - Želimo znati kako stvari funkcioniraju „ispod haube”
  - Imamo problem sa skalabilnošću i verzioniranjem
  - Nemamo monitoring, explainere, drift detection
  - Sve to ide ručno, što je za veće sustave prilično mukotrpan posao
  - Dobro funkcionira kad imamo manji sustav i vrlo ograničeni broj manjih modela
- Disclaimer: na ovim predavanjima govorimo o OpenSource ML stacku, a Seldon komponente su odabrane kao adekvatna poslužiteljska okolina koja daje dovoljno funkcionalnosti i fleksibilnosti za male i velike sustave
- Prva nadogradnja: [Seldon MLServer](#)



# Seldon MLServer

- Jednostavan server koji se može pokrenuti na više načina
  - Lokalno na računalu
  - Može se izgenerirati slika za pokretanje u kontejneru (*docker*)
  - U Kubernetes clusteru na *Seldon Core* ili *Seldon Deployment* serveru
    - Što omogućava i pokretanje na privatno i javno dostupnim elastičnim clusterima
- U sebi sadrži *inference runtime* za razne ML module:
  - scikit-learn, xgboost, MLflow, Tempo, Spark MLlib, LightGBM, Alibi-Detect, Alibi-Explain, HuggingFace, ...
  - MLflow – omogućava direktno posluživanje modela iz MLflow-a
    - Sjetimo se – MLflow ima poseban način pakiranja modela
    - Koristi *pickle* format zapisa
    - Dodatni opisnici za okolinu i sučelje modela



# Seldon MLServer

- Instalira se sa:

```
pip install mlserver
```

- Dodatni *inference runtime* sa:

```
pip install mlserver-sklearn mlserver-mlflow ...
```

- Može posluživati jedan ili više modela

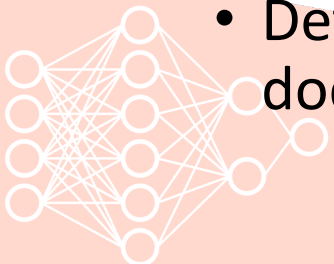
- Model spremljen prema pravilima tog konkretnog python ML modula
  - scikit-learn: *joblib* datoteka s modelom
  - MLflow: *pickle* datoteka + opisnici
  - ...

- Implementira [v2 inference protocol](#)



# Seldon MLServer

- *settings.json* – Datoteka s parametrima servera
  - Portovi, paralelizam, debugiranje, sučelja, folder s modelima, *endpoint* za metriku, detalji za Kafka sučelje, ...
  - Dostupna sučelja: REST, gRPC, Kafka redovi
  - [settings.json dokumentacija](#)
  - Stavlja se u folder u kojem se pokreće MLserver
- *model-settings.json* – Datoteka s parametrima modela
  - Ide u folder s modelom – ako se poslužuje jedan model, to je isti folder u kojem je i *settings.json* datoteka
  - [model-settings.json dokumentacija](#)
  - Definiramo *inference runtime*, datoteke s modelima, URL prefikse za modele i dodatne parametre za modele





# Seldon MLServer

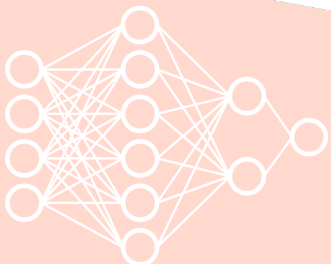
- Struktura za više modela

```
|-- settings.json
|-- model 1
|   |-- model-settings.json (URL: /model1)
|   +-- <model 1 files>
+-- model 2
    |-- model-settings.json (URL: /model2)
    +-- <model 2 files>
```

- MLServer se pokrene u folderu sa *settings.json* kao

```
mlserver start .
```

- Ovakav poziv blokira terminal – nije pogodno za Jupyter notebook
- Osim ako u *settings.json* nismo drukčije podesili, imamo na raspolaganju REST sučelje na portu 8080



# Seldon MLServer

- Definiramo konfiguracijske datoteke
- REST poziv sadržava specifičan zahtjev – *v2 inference protocol*

```
import requests

inference_request = {
    "inputs": [
        {
            "name": "predict",
            "shape": (2,4),
            "datatype": "FP32",
            "data": [[-1.143017, 1.249201, -1.340227, -1.447076],[0.432165, 0.788808,
0.933271, 1.448832]]
        }
    ]
}

endpoint = "http://localhost:8080/v2/models/mlp-iris/versions/v1.0.0/infer"
response = requests.post(endpoint, json=inference_request)

response.json()
```

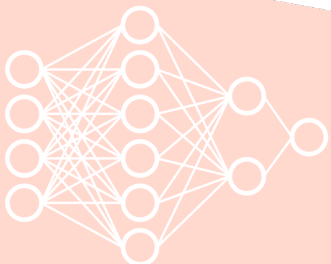
```
%%writefile ./mlserver_iris/settings.json
{
    "debug": "true"
}
```

```
%%writefile ./mlserver_iris/model-settings.json
{
    "name": "mlp-iris",
    "implementation": "mlserver_sklearn.SKLearnModel",
    "parameters": {
        "uri": "./mlp_iris.joblib",
        "version": "v1.0.0"
    }
}
```

# Seldon MLServer

- Imamo mogućnost pokretanja u kontejneru
  - Umjesto pokretanja servera u folderu s *settings.json* radimo

```
mlserver build . -t <image name>:<version>
```
  - Tako generirani kontejner pokrenemo i imamo isti efekt kao i na prethodnim prikaznicama
    - Možemo ga pokrenuti i unutar *Seldon Core* instance
    - Ovo je značajno kada trebamo pokrenuti *custom* modele na *Seldon Core-u*

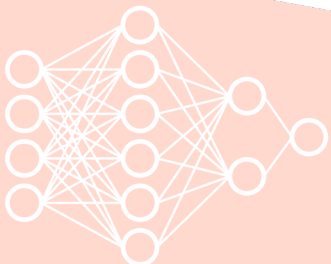


# Seldon MLServer

- Custom model

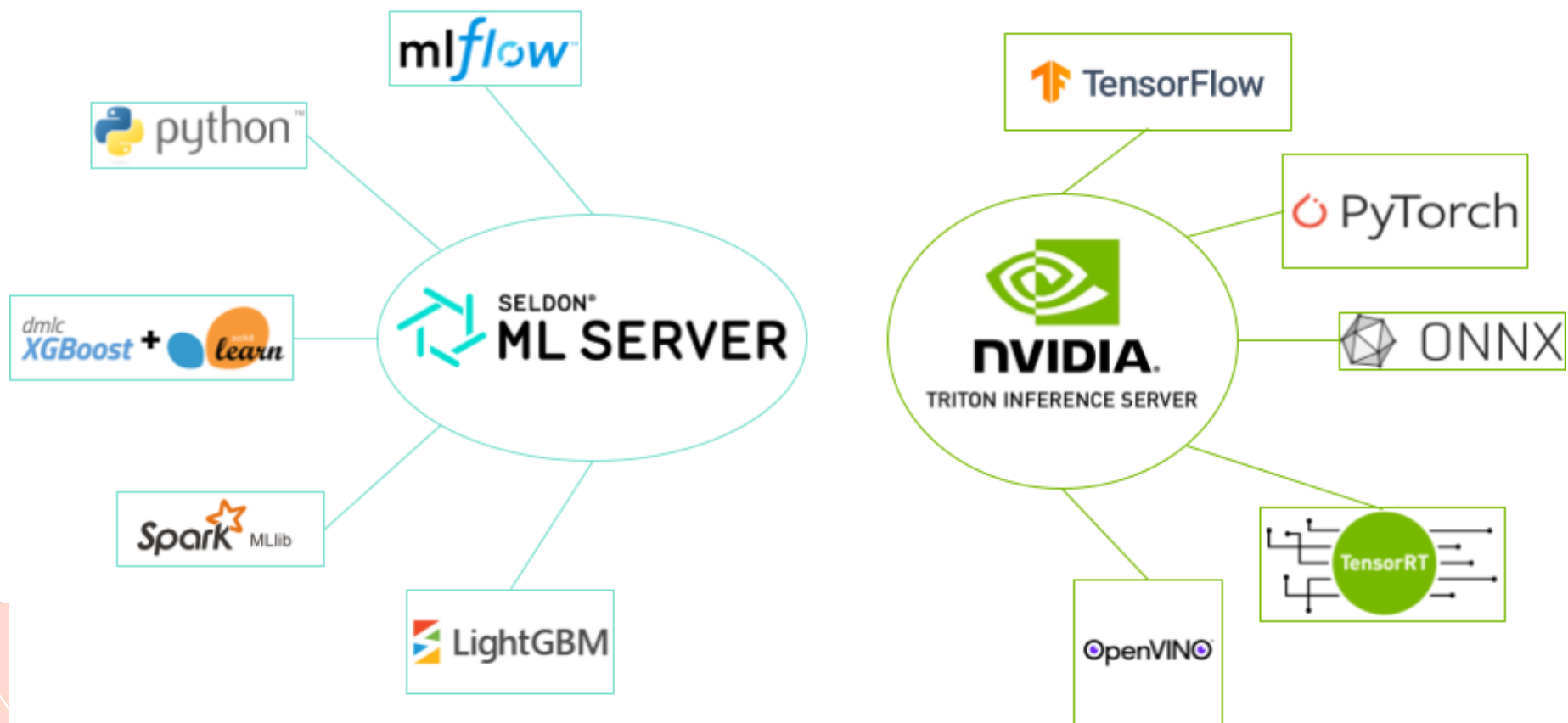
- Python kod koji implementira *MLModel* klasu
- Parametre dobijemo kroz *self.\_settings* svojstvo
- Metoda *predict* odrađuje glavnu stvar
- Metoda *load* čita model iz datoteke, a datoteka je snimljena nakon učenja u formatu koji smo sami odabrali – recimo JSON
- Sučelje *predict* metode treba biti prilagođeno postavkama u *model-settings.json* datoteci
  - Ili imamo kompletno custom sučelje

```
class MojModel(MLModel):  
    async def load(self) -> bool:  
        model_uri = await get_model_uri(self._settings)  
        with open(model_uri) as model_file:  
            <pročitaj model>  
  
    @decode_args  
    async def predict(self, <parametri>) -> np.ndarray:  
        <predict>  
  
%%writefile ./model-settings.json  
{  
    "name": "moj-model",  
    "implementation": "models.MojModel",  
    "parameters": { "uri": "./moj-model.json" }  
}
```



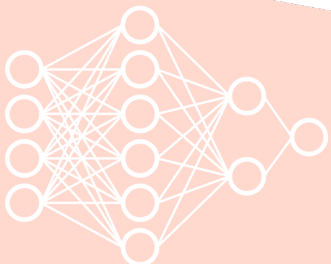
# Seldon Core v2

## V2 Multi-Model Capable Servers



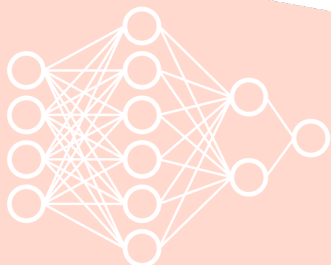
# Seldon Core v2

- Dvije varijante
  - *docker* instalacija – uglavnom za development i testiranje
    - Ručna instalacija – build i sve ostalo
  - *Kubernetes cluster* instalacija - produkcija
    - Moguće instalirati na *KinD cluster*
      - Razvoj i testiranje
      - Koristi *ansible playbook* koncept za instalaciju
    - Produkcijska instalacija uglavnom ide ručno kroz *helm*



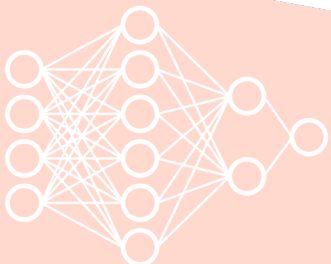
# Seldon Core v2

- Dodatne komponente koje se moraju/mogu instalirati
  - Kafka – obavezno
    - Omogućava servisne cjevovode na *Seldon Core-u* – ulančavanje modela
    - Ovo je moguće s obzirom na *v2 inference protocol*
    - Sjetite se modela za skaliranje u *iris* scikit-learn modelu – imamo poseban *joblib* koji je utreniran da skalira ulaz – u *Seldon Core* terminologiji je to transformer
  - Dodatni modeli koji se mogu ulančavati u servisni cjevovod
    - *Explainer* – Model koji se koristi za objašnjavanje odluke ili generalno utvrđivanje odnosa odluke i ulaznih podataka – [Explainable AI](#)
    - *Drift detectors* – Model koji se koristi za detekciju promjena u odlukama – drift modela
      - Uvijek se pozivaju asinkrono u servisnom cjevovodu jer se uobičajeno koriste za pokretanje cjevovoda za učenje



# Seldon Core v2

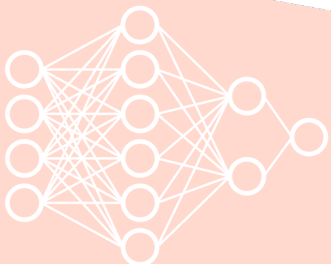
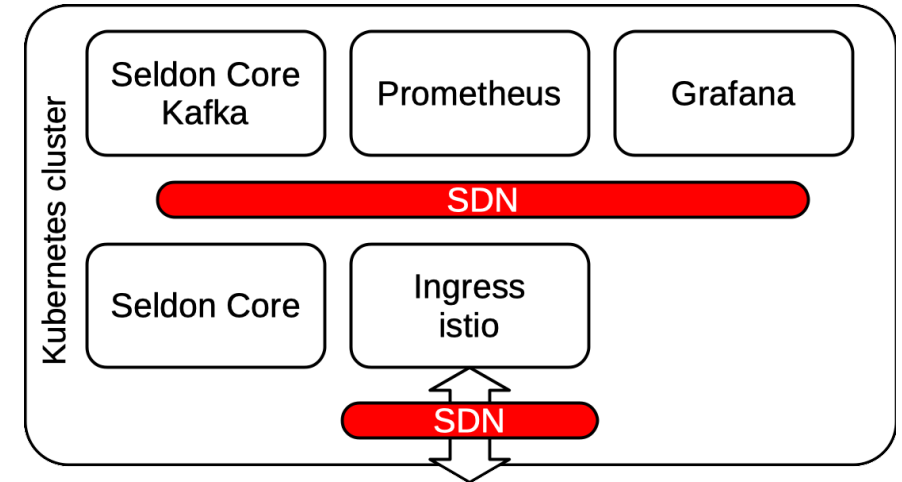
- Dodatne komponente koje se moraju/mogu instalirati
  - [Prometheus](#) – opcija
    - Spaja se na monitoring endpoint *Seldon Core-a* – monitoring endpoint postoji i na MLServeru
    - Sprema vrijednosti u TSDB – *time-series database*
    - Upiti nad TSDB + ograničena vizualizacija
  - [Grafana](#) – opcija
    - *Dashboard*
    - Vizualizacija iz monitoringa
  - Ostale opcionalne komponente nećemo posebno isticati





# Seldon Core v2

- *Ingress* agnostic
  - Kao *ingress* u Kubernetes cluster se može koristiti *istio*, *nginx*, *Ambassador*, *Traefic*
  - Svaki deployment ima svoj virtualni servis u *Seldon Core namespace-u* - *seldon\_mesh\_namespace*
  - Kafka ima svoj namespace - *seldon\_kafka\_namespace*
  - *Ingress* mapira te virtualne servise – omogućava korištenje servisa izvan Kubernetes clusters



# Seldon Core v2

- Deployment modela pojednostavnjen
  - Model se spremi u repozitorij – AWS, minIO (S3), GS
  - Napravi se YAML opisnik koji se zatim s *kubectl deploy-a* na Kubernetes cluster

**apiVersion:** mlops.seldon.io/v1alpha1

**kind:** Model

**metadata:**

**name:** iris

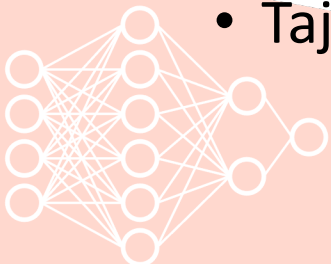
**spec:**

**storageUri:** "gs://seldon-models/scv2/samples/mlserver\_1.3.0/iris-sklearn"

**requirements:** - sklearn

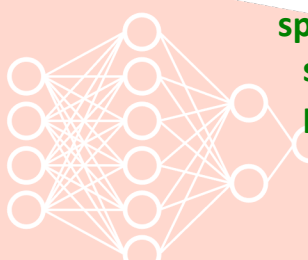
kubectl create -f ./sklearn-iris-gs.yaml -n seldon-mesh-namespaces

- REST URL definiran je zatim naziv-om modela u yaml-u i postavkama *ingress-a*
- Taj se URL može provjeriti korištenjem *kubectl-a*



# Seldon Core v2

- S obzirom na vrijednost *requirement* atributa u yaml *deployment* opsiniku - aktivira se ili MLServer ili Nvidia Triton server
  - Za pytorch će to biti Triton server
- Možemo raditi *deployment* custom MLServer kontejnera
  - Ako recimo imamo custom model
  - Kubernetes clusteru moramo omogućiti pristup generiranoj MLServer *docker* slici (*image*)
  - To može kroz *docker hub*, *github* ili direktno uploadom u Kubernetes cluster

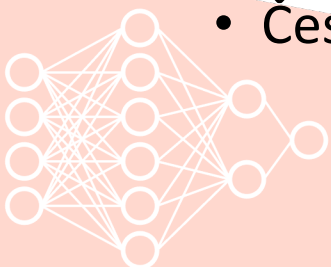


```
apiVersion: mlops.seldon.io/v1alpha1
kind: Server
metadata:
  name: mlserver-custom
spec:
  serverConfig: mlserver
  podSpec:
    containers:
      - image: cliveseldon/mlserver:1.2.0.dev1
        name: mlserver
```

```
apiVersion: mlops.seldon.io/v1alpha1
kind: Model
metadata:
  name: iris
spec:
  storageUri: "gs://seldon-models/mlserver/iris"
  server: mlserver-custom
```

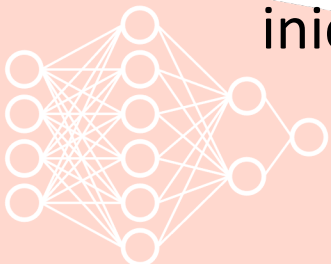
# Seldon Core v2 - Explainers

- Zašto je donesena odluka kakva je donesena?
- Model treiramo kao *black box*, te radimo statističku obradu ulaza i izlaza modela
  - Na temelju toga donosimo zaključke koje značajke su bitne kod donošenja odluke
  - Recimo [SHAP](#), [LIME](#)
  - Seldon ima svoj paket – [Alibi-Explain](#)
- Može se koristiti u fazi eksperimentiranja – MLflow
  - Kod učenja ili testiranja pozovemo statističku obradu kako bismo utvrdili bitne značajke – posebno zanimljivo kod klasifikatora
- *Explainer* – Model koji se često poziva u sekvenci sa samim ML modelom
  - Često se radi augmentacija izlazne strukture (rezultata) s objašnjenjem odluke



# Seldon Core v2 – Drift detektori

- Da li je odluka u skladu s utreniranim modelom?
  - Neka odluka može biti *outlier* s obzirom na ulazne podatke
  - Ako modelu pada preciznost, znači da nam odluke nisu ispravne
- Model tretiramo kao *black box*, te radimo statističku obradu ulaza i izlaza modela
  - Na temelju toga se radi statistička obrada – multivarijatna analiza, npr. clustering, Kolmogorov-Smirnov test, ...
  - Seldon ima svoj paket – [Alibi-Detect](#)
- *Drift detector* – Model koji se često poziva asinkrono nakon samog ML modela
  - Ako se detektira drift, tu detekciju možemo recimo staviti na Kafka red – za iniciranje izvršenja cjevovoda za učenje



# Seldon Core v2 – Cjevovodi

- Seldon Core cjevovod je zapravo samo dio servisnog cjevovoda
- Omogućava ulančavanje modela, npr. transformacija, model, explainer, drift detector
- V2 inference protocol koristi specifičnu definiciju tensora koja omogućava jednostavniji prijenos podataka između modela
  - Mapiranje tensora u cjevovodu
  - Uvjeti, okidači, hvatanje grešaka
- Prvo *deploy-amo* sve modele, a zatim ih povežemo u cjevovod

```
apiVersion:
mlops.seldon.io/v1alpha1
kind: Pipeline
metadata:
  name: chain
  namespace: seldon-mesh-
namespace
spec:
  steps:
    - name: model1
    - name: model2
  inputs:
    - model1
  output:
    steps:
      - model2
```



# Seldon Core v2 – Kubernetes cluster

- Kubernetes cluster – kontejnerizirana okolina
  - Uglavnom *stateless* okoline
  - Za svu persistenciju – *persistent volume*
  - Definiramo broj replika određenog modela – povećavamo skalabilnost
  - Ukoliko jedna replika „nestane”, digne se nova i promet se preusmjeri
  - Seldon Core operator
    - Deployment modela – pakiranje u kontejnere
    - Upravljanje replikama
  - Persistencija Seldon Core cjevovoda – Kafka
    - Nebitno koja replika pokupi tensor i nastavi obradu u cjevovodu

