PYCON KOREA 2023

MLOps: Model Serving Architecture With BentoML

Naver Biz CIC AI Serving Dev

KimSoungRyoul

https://github.com/KimSoungRyoul

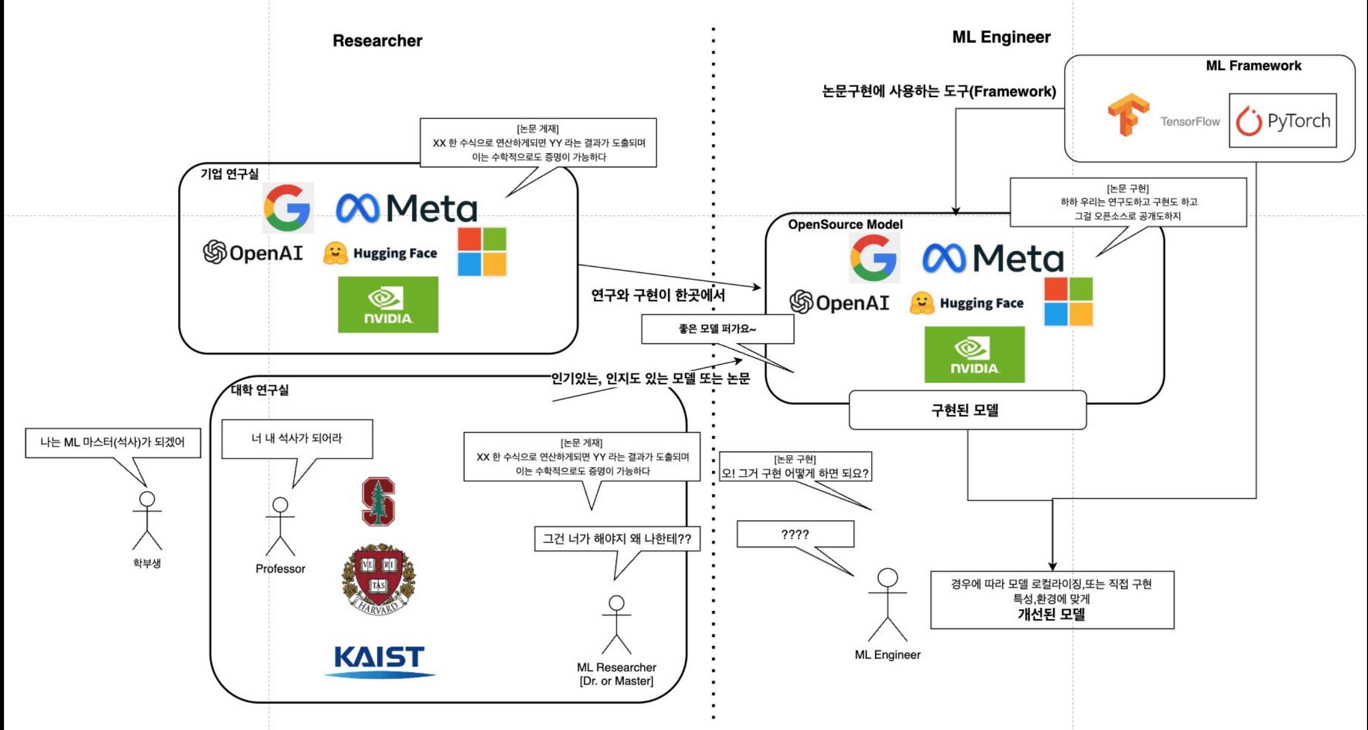
https://github.com/KimSoungRyoul/PyConKR2023-ModelServing

BACK TO US,
BACK TO PYTHON

Index

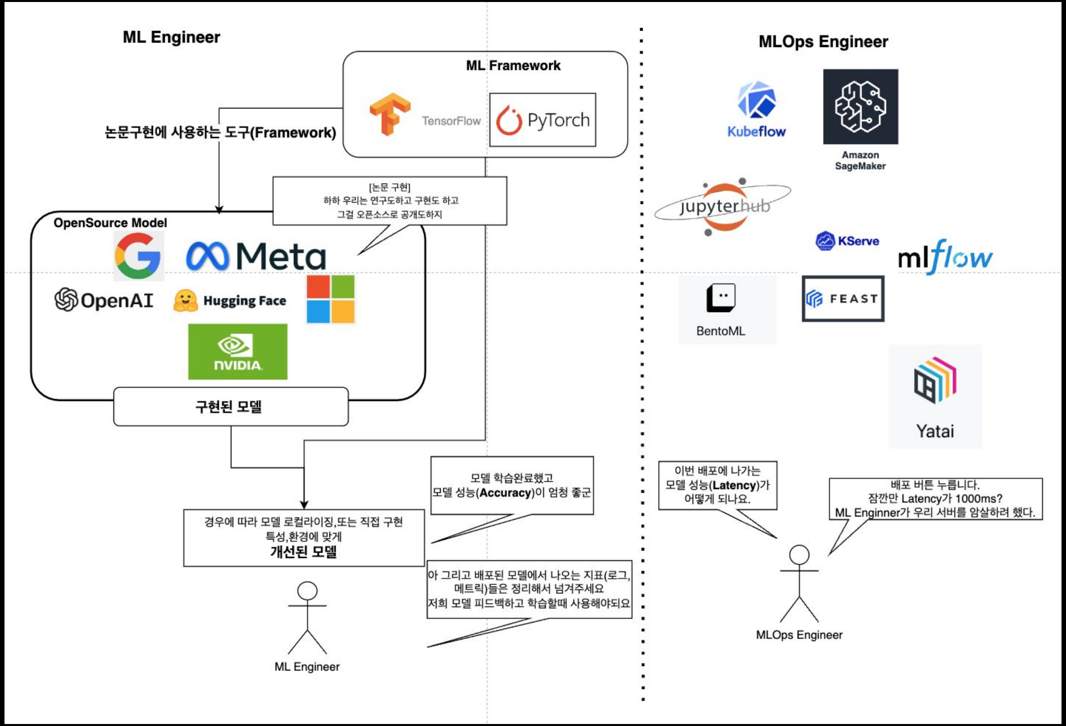
- What is MLOps?
- Model Serving Framework
- Model Serving Platform

What is MLOps ? [Researcher & ML Engineer]



퍼가요: 싸이월드(2000년대 중후반 SNS)에서 게시글을 인용또는 그대로 가져올때 답글로 “퍼가요”라고 남기는 것이 예의였다.
지금으로 따지면 Github에서 Repo나 gist에서 fork 해올때 star를 누르는 것과 비슷한 느낌

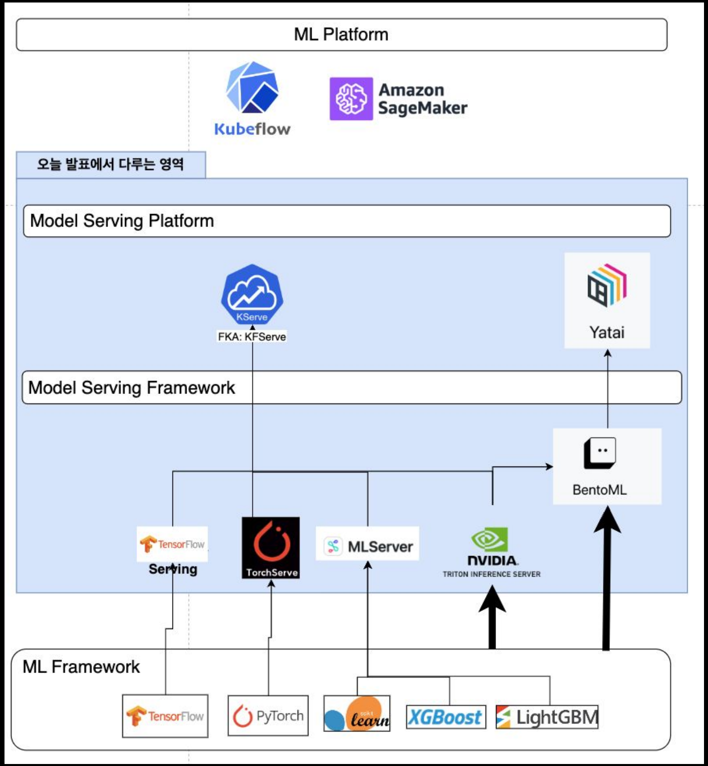
What is MLOps ? [MLEnginner & MLOps Engineer]



In Academia, I was an ML PhD,
But in corporate,
Am I just a ShellScript Master?

- Building the model is crucial, but there's so much more to do beyond that.
- From model implementation to deployment to operations, there's a need for automation through platform construction. Managing and building this is known as MLOps. MLOps encompasses both model implementation and training.
- Model Serving' is just one part of this broader MLOps spectrum.

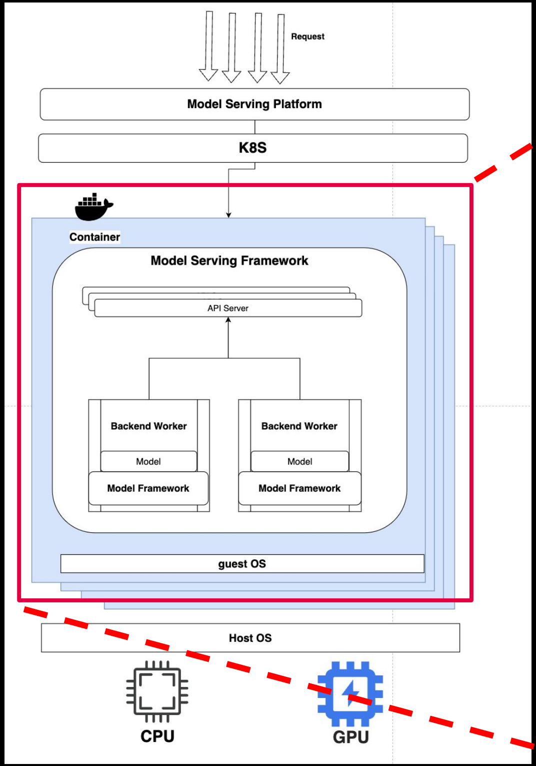
What is Model Serving? ML Serving EcoSystem

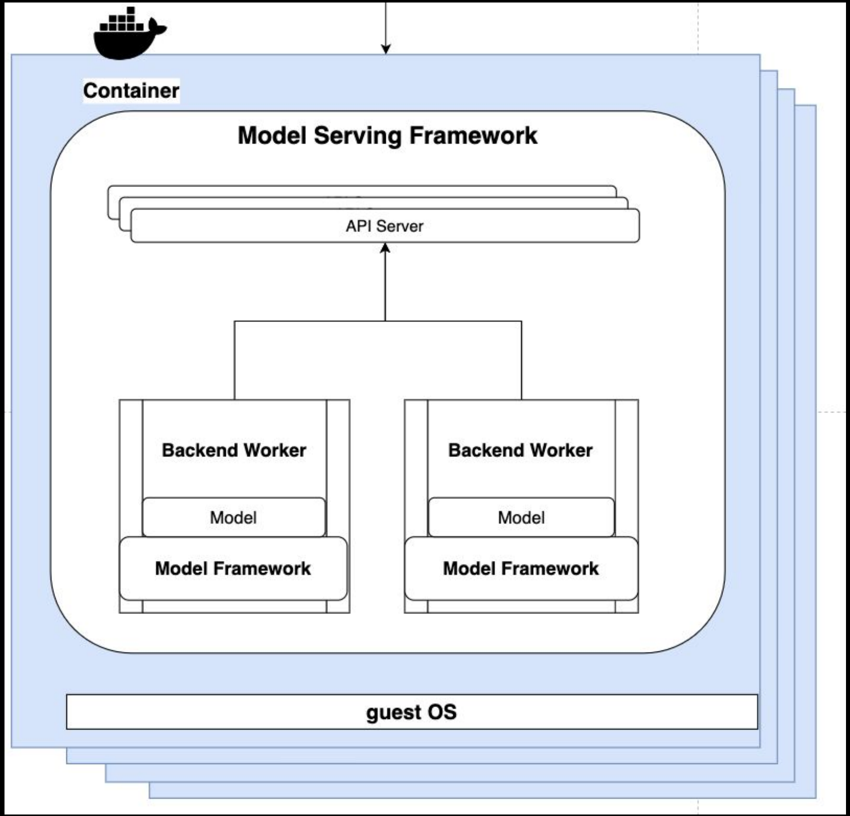


Model Serving Framework

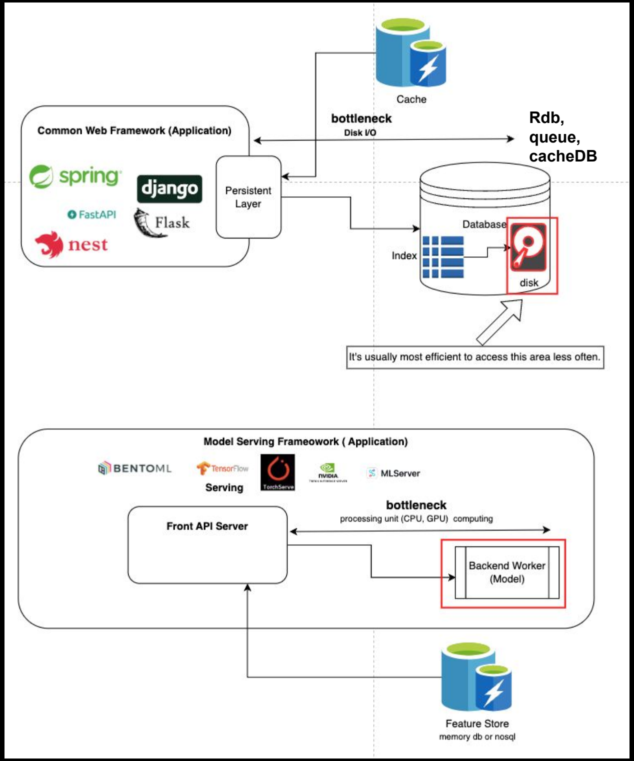
What is Model Serving?

ML Serving EcoSystem Detail





About Model Serving Framework



Web Framework 와 Model Serving Framework have a lot of common features

For example

- Protocol support (http, grpc)
- Serialization
- Api docs (OAS 3.x)
- …

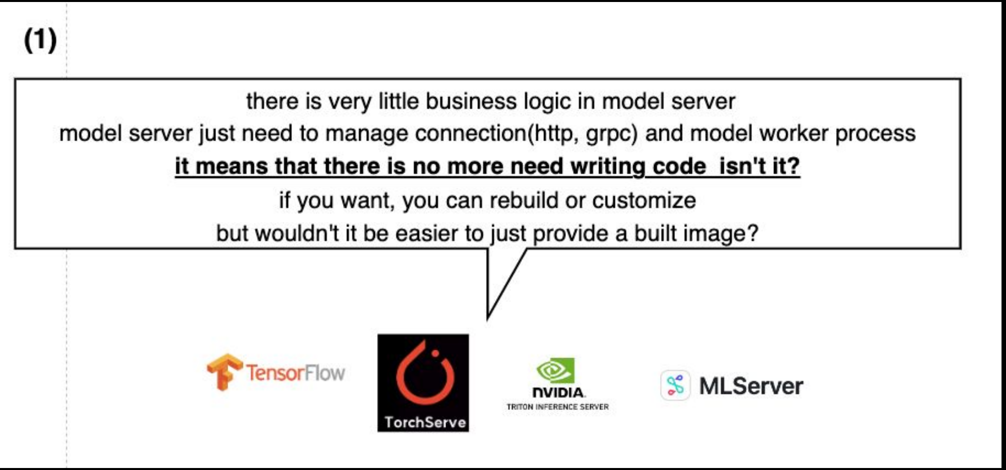
But, each has specialized features for their respective areas

For example In Model Serving Framework

- Support builtin default Metrics & Log

- Manage model backend worker process count

Model Serving: Architecture Concept



tensorflow/serving ☆

By tensorflow • Updated 18 hours ago

Official images for TensorFlow Serving (http://www.tensorflow.org/serving)

Image

pytorch/torchserve ☆

By pytorch • Updated 2 months ago

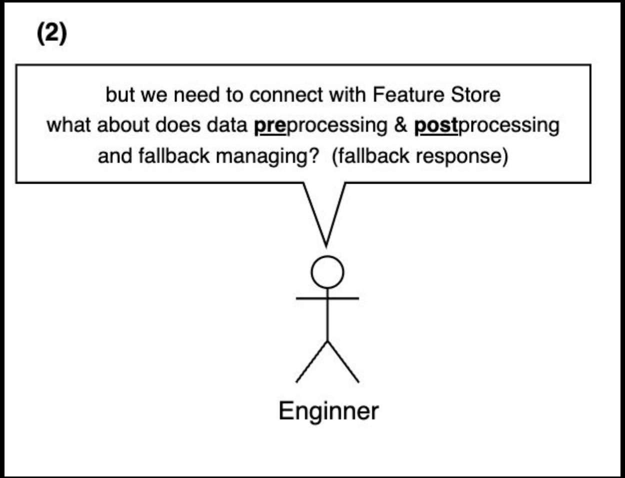
Image

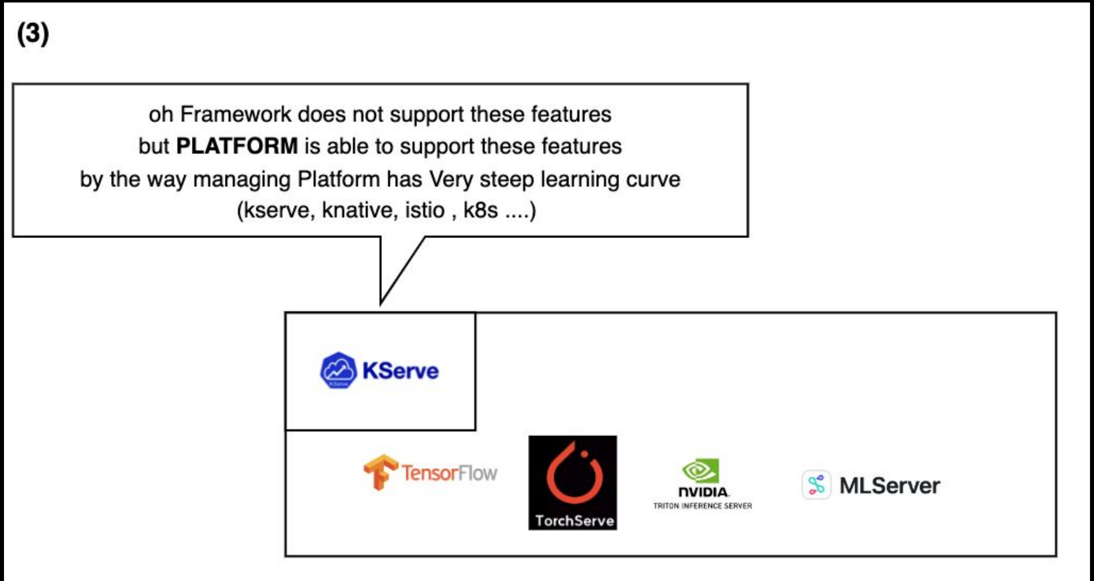
NVIDIA Triton Inference Server Container Versions

The following table shows what versions of Ubuntu, CUDA, Triton Inference Serve

Container Version	Triton Inference Server	Ubuntu
21.08	2.13.0	20.04
21.07	2.12.0	
21.06.1	2.11.0	
21.06		
21.05	2.10.0	

Model Serving: Architecture Concept



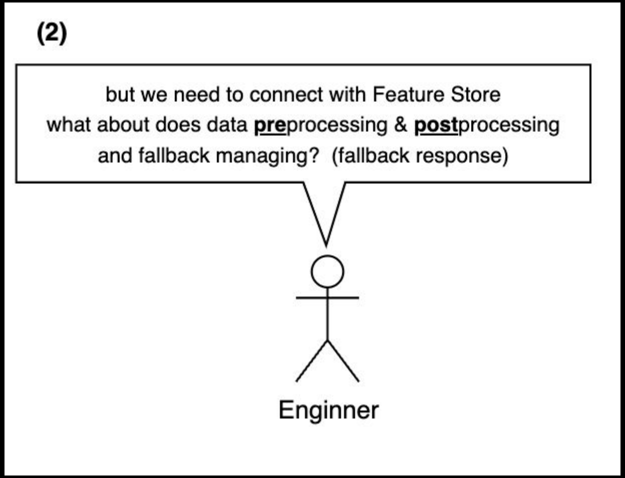


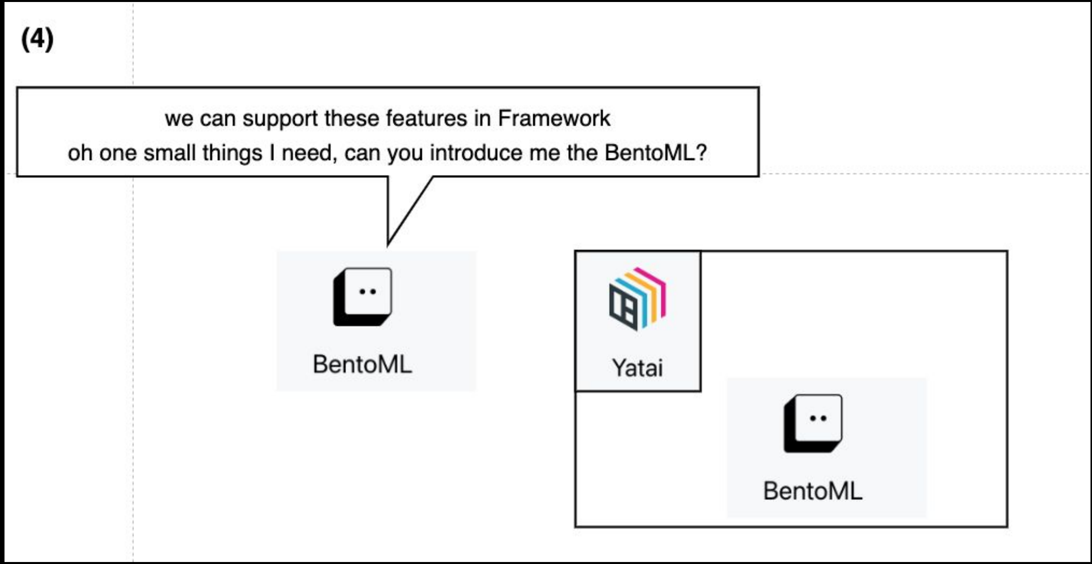
Low Level): computer friendly

high Level): human friendly

ex: python is more human friendly than C

Model Serving: Architecture Concept



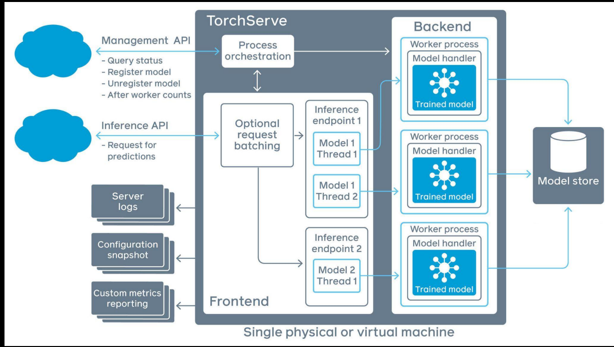


Low Level): computer friendly
high Level): human friendly
ex: python is more human friendly than C

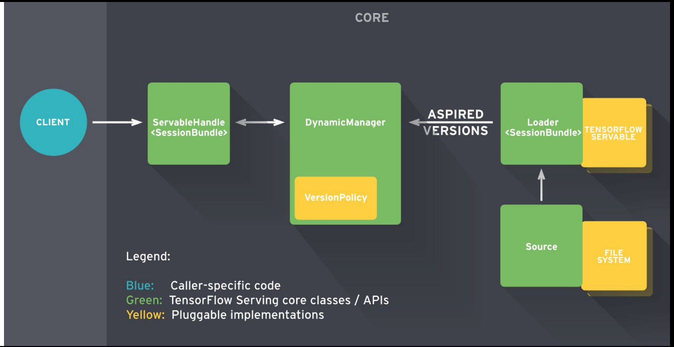
Model Serving: Architecture Concept

Model server is simple, there is only two component

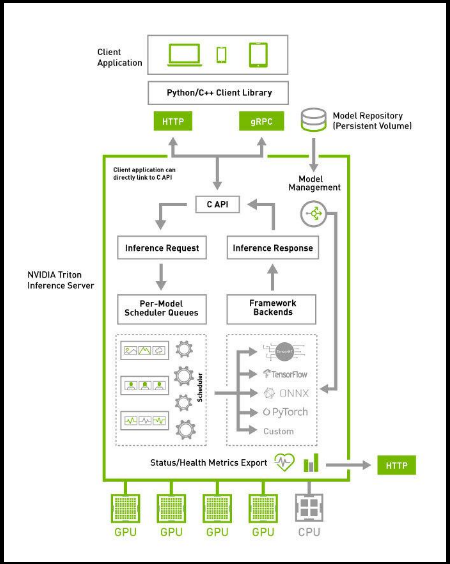
- Manage Http, grpc connection & pre,post process logic : Front API Server
- inference Model worker process : Backend(Model) Worker



TorchServe Architecture



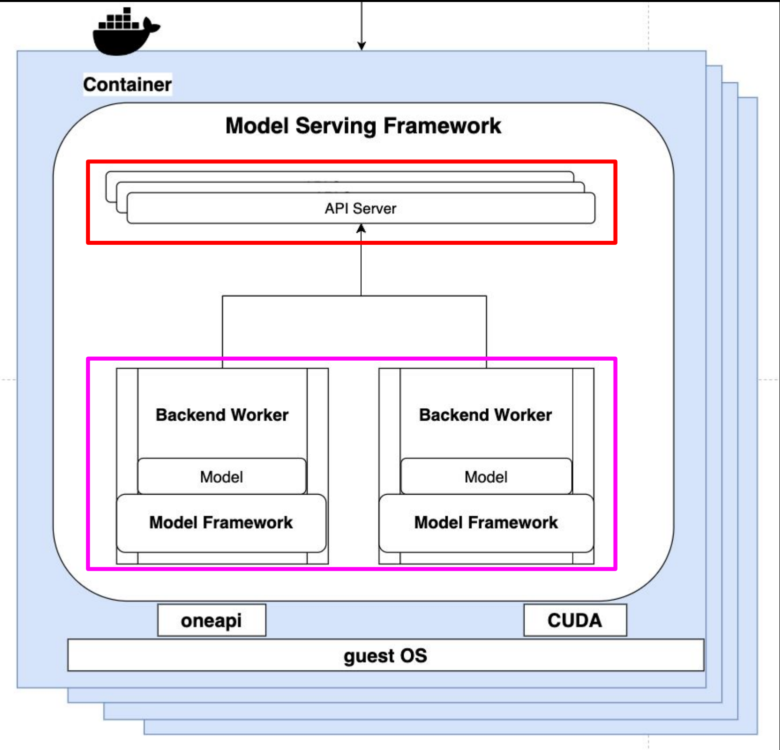
TensorflowServing Architecture

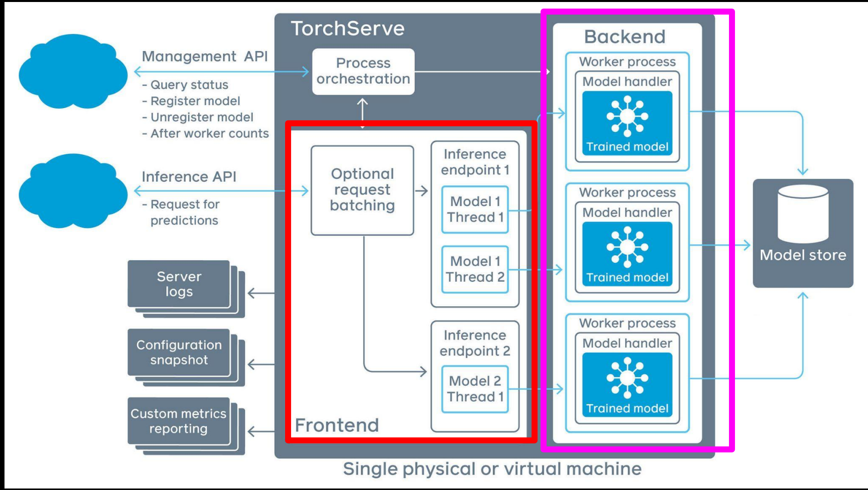


Triton Inference Server Architecture

https://pytorch.org/serve/index.html
https://www.tensorflow.org/tfx/serving/architecture
https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html

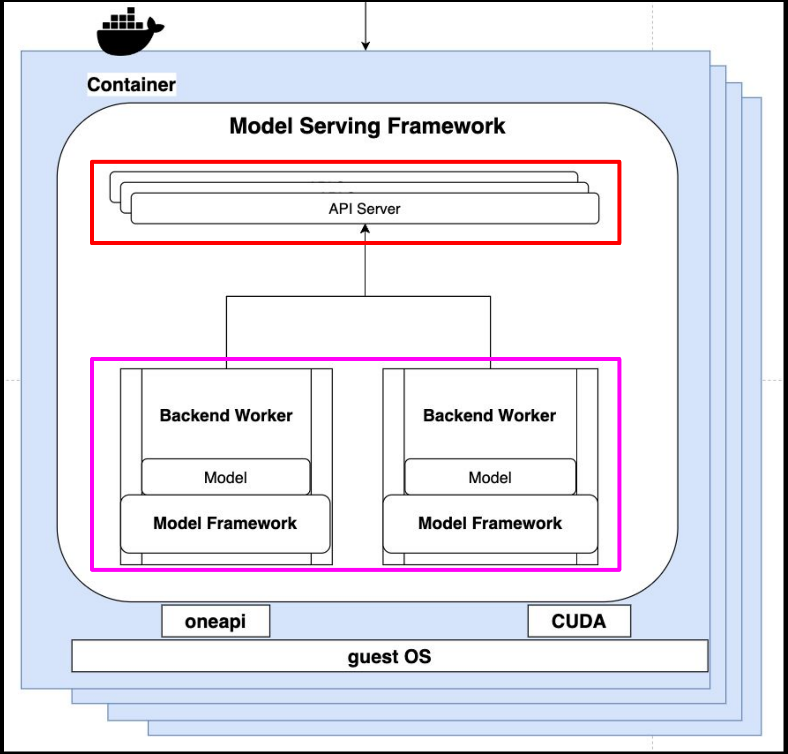
Model Serving: Architecture Concept

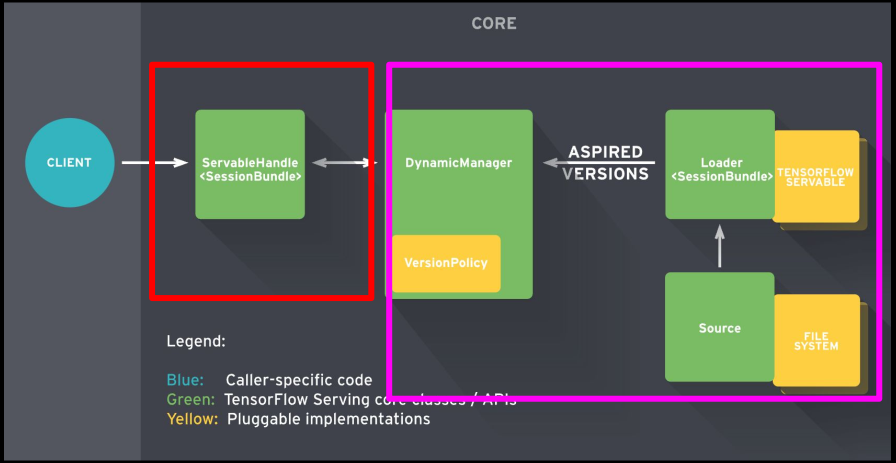




https://pytorch.org/serve/index.html
https://www.tensorflow.org/tfx/serving/architecture
https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html

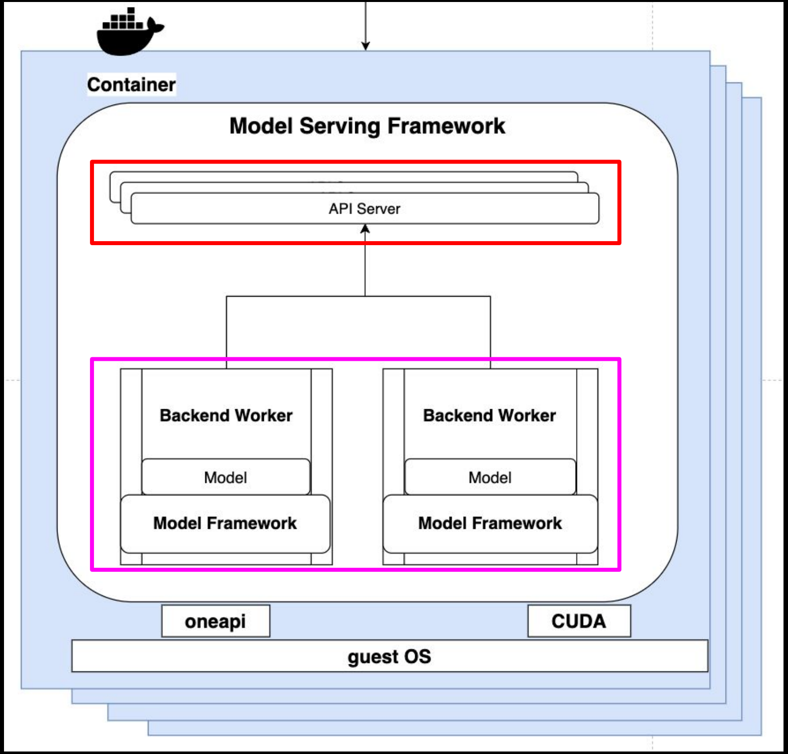
Model Serving: Architecture Concept

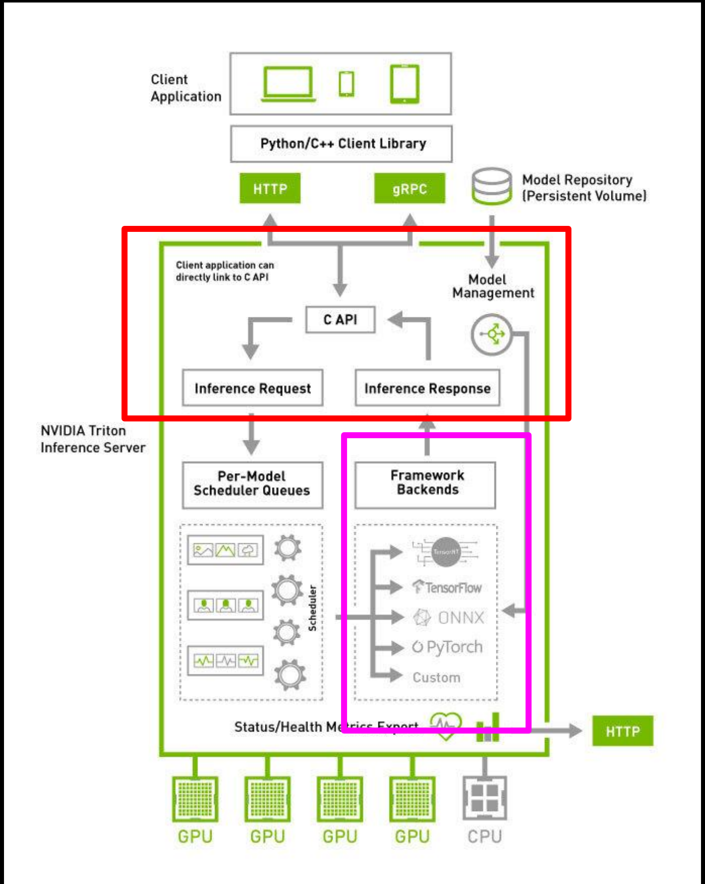




https://pytorch.org/serve/index.html
https://www.tensorflow.org/tfx/serving/architecture
https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html

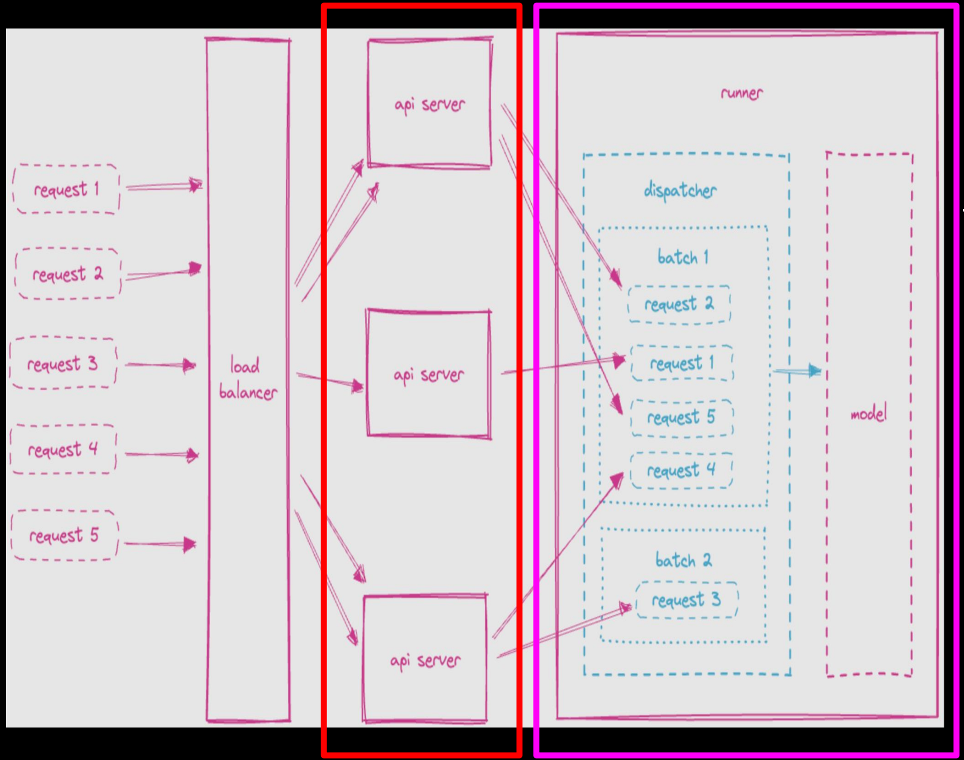
Model Serving: Architecture Concept





https://pytorch.org/serve/index.html
https://www.tensorflow.org/tfx/serving/architecture
https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html

Model Serving: Architecture Concept



BentoML is also same with other model serving framework

In BentoML

- API Server is Service (like fastapi app)
- Backend Worker is called Runner
- Bento is a combination of one API Server and 1~N Runner

  Bento is Unit of deployment in BentoML

BentoML allow to writing code

Unlike other frameworks where writing code is not the primary pattern

but BentoML allow to write code easily

https://docs.bentoml.com/en/latest/guides/batching.html#architecture

BentoML: Quick Start [save model **without** BentoML)

1. No BentoML only pure pytorch training example

```python
# sample train pytorch model
# 1. define Model
class SampleDummyModel(torch.nn.Module):
    def forward(self, x_tensor) -> Tensor:
        # pytorch Module, only return transposed tensor
        return torch.transpose(x_tensor, 0, 1)


model = SampleDummyModel()

# 2. something model training code ....

# 3. finally save trained model to file
torch.save(model.state_dict(), "./sample_dummy_model.pt")
```

```
❯ python sample_train.py
```

- __init__.py
- sample_dummy_model.pt
- sample_train.py

BentoML: Quick Start (save model **with** BentoML)

1. pytorch training example **with BentoML**

~/workspace/i_am_python/i_am_managed_fastapi_project/README.md

```python
import bentoml
import torch
from torch import Tensor


# sample train pytorch model
# 1. define Model
class SampleDummyModel(torch.nn.Module):
    def forward(self, x_tensor) -> Tensor:
        # pytorch Module, only return transposed tensor
        return torch.transpose(x_tensor, 0, 1)


model = SampleDummyModel()

# 2. something model training code ....

# 3. finally save trained model to file
# torch.save(model.state_dict(), "./sample_dummy_model.pt")
bentoml.pytorch.save_model(
    name=f"sample-dummy-model:{date.today().strftime('%Y-%m-%d')}",  # {model-name}:{model-version}
    model=model,
    labels={  # you can use labeling which managed by BentoML
        "maintainer": "kimsoungryoul",
    }
)
```

```
❯ python sample_train.py
```

```
❯ bentoml models list sample-dummy-model
```

Tag	Module	Size	Creation Time
sample-dummy-model:2023-08-12	bentoml.pytorch	2.22 KiB	2023-06-17 23:15:01

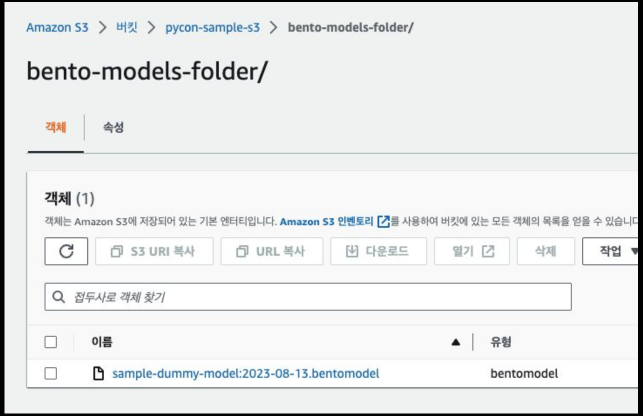
Just modify one line, if you want to bentoML

BentoML: Quick Start (upload model to S3 **with** BentoML)

```
❯ bentoml models list sample-dummy-model
Tag                            Module           Size       Creation Time
sample-dummy-model:2023-08-12  bentoml.pytorch  2.22 KiB   2023-06-17 23:15:01
```

```
 ~/workspace/PyConKR2023-ModelServing/sample_bento  on  main +70 !21 ?24
❯ pip install "bentoml[aws]"
```

```
❯ bentoml models export sample-dummy-model:2023-08-13 s3://pycon-sample-s3/bento-models-folder/sample-dummy-model:2023-08-13.bentomodel
Model(tag="sample-dummy-model:2023-08-13") exported to s3://pycon-sample-s3/bento-models-folder/sample-dummy-model:2023-08-13.bentomodel.
```



❯ bentoml models export 모델명:버전
s3://**bucket-name**/**path~**/**model-name**.bentomodel

BentoML: Quick Start (save & upload model with BentoML)

```python
import bentoml
import torch
from torch import Tensor
from bentoml import Model


# sample train pytorch model
# 1. define Model
class SampleDummyModel(torch.nn.Module):
    def forward(self, x_tensor) -> Tensor:
        # pytorch Module, only return transposed tensor
        return torch.transpose(x_tensor, 0, 1)


model = SampleDummyModel()

# 2. something model training code ....

# 3. finally save trained model to file
# torch.save(model.state_dict(), "./sample_dummy_model.pt")
bento_model: Model = bentoml.pytorch.save_model(
    name=f"sample-dummy-model:{datetime.today().strftime('%Y-%m-%d')}",  # {model-name}:{model-version}
    model=model,
    labels={  # you can use labeling which managed by BentoML
        "maintainer": "KimSoungRyoul",
    },
)

bentoml.models.export_model(
    tag=bento_model.tag,
    path="s3://pycon-sample-s3/bento-models-folder/sample-dummy-model:2023-08-13.bentomodel",
    user='<AWS access key>', passwd='<AWS secret key>',
)
```

=

Model Management API

Besides the CLI commands, BentoML also provides equivalent Python APIs for managing models:

Get | List | Import / Export | Push / Pull | Delete

```python
import bentoml
bentoml.models.export_model('iris_clf:latest', '/path/to/folder/my_model.bentomodel')
```

```python
bentoml.models.import_model('/path/to/folder/my_model.bentomodel')
```

Note

Model can be exported to or import from AWS S3, GCS, FTP, Dropbox, etc. For example:

```python
bentoml.models.import_model('s3://my_bucket/folder/my_model.bentomodel')
```

```
❯ bentoml models export {MODEL_NAME}:{MODEL_VERSION}
s3://BUCKET_NAME/PATH~/MODEL_NAME.bentomodel
```

BentoML: Quick Start (save & upload model **with** BentoML)

```python
import bentoml
import torch
from torch import Tensor
from bentoml import Model


# sample train pytorch model
# 1. define Model
class SampleDummyModel(torch.nn.Module):
    def forward(self, x_tensor) -> Tensor:
        # pytorch Module, only return transposed tensor
        return torch.transpose(x_tensor, 0, 1)


model = SampleDummyModel()

# 2. something model training code ....

# 3. finally save trained model to file
# torch.save(model.state_dict(), "./sample_dummy_model.pt")
bento_model: Model = bentoml.pytorch.save_model(
    name=f"sample-dummy-model:{datetime.today().strftime('%Y-%m-%d')}",  # {model-name}:{model-version}
    model=model,
    labels={  # you can use labeling which managed by BentoML
        "maintainer": "KimSoungRyoul",
    },
)

bentoml.models.export_model(
    tag=bento_model.tag,
    path="s3://pycon-sample-s3/bento-models-folder/sample-dummy-model:2023-08-13.bentomodel",
    user='<AWS access key>', passwd='<AWS secret key>',
)
```

If you use Yatai
You can manage model version like a

- docker pull push
- git pull push

Model Management API

Besides the CLI commands, BentoML also provides equivalent Python APIs for managing models:

Get | List | Import / Export | **Push / Pull** | Delete

If your team has Yatai setup, you can also push local Models to Yatai, it provides APIs and Web UI for managing all Models created by your team and stores model files on cloud blob storage such as AWS S3, MinIO or GCS.

```python
import bentoml
bentoml.models.push("iris_clf:latest")
```

```python
bentoml.models.pull("iris_clf:latest")
```

=

```
❯ bentoml models export {MODEL_NAME}:{MODEL_VERSION} s3://BUCKET_NAME/PATH~/MODEL_NAME.bentomodel
```

BentoML: Quick Start (inference without BentoML)

2. pytorch inference example **without BentoML**

```python
# sample inference pytorch model
# 1. define inference Model
class SampleDummyModel(torch.nn.Module):
    def forward(self, x_tensor) -> Tensor:
        # pytorch Module, only return transposed tensor
        return torch.transpose(x_tensor, 0, 1)


# 2. load model
model = SampleDummyModel()
model.load_state_dict(torch.load("./sample_dummy_model.pt"))
model.eval()

sample_input = torch.tensor([[1.1, 2.2], [3.3, 4.4]], dtype=torch.float32)

# 3. inference
inference_output = model(sample_input)

print(inference_output)
```

```
❯ python sample_inference.py
tensor([[1.1000, 3.3000],
        [2.2000, 4.4000]])
```

BentoML: Quick Start (download model & inference with BentoML)

2. pytorch inference example **with BentoML**

```
# s3 download bentoModel
bentoml.models.import_model(
    "s3://pycon-sample-s3/bento-models-folder/sample-dummy-model:2023-08-13.bentomodel",
    user="<AWS access key>", passwd="<AWS secret key>",
)

sample_dummy_model = bentoml.pytorch.get("sample-dummy-model:latest")
runner = sample_dummy_model.to_runner()
runner.init_local(quiet=True)

inference_input = torch.tensor(sample_input, dtype=torch.float32)

# 3. inference
# inference_output = model(sample_input)
inference_output = runner.run(inference_input)

print(inference_output)
```

```
❯ python sample_bentoml_inference.py
tensor([[1.1000, 3.3000],
        [2.2000, 4.4000]])
```

BentoML: Quick Start [Model Serving] dev mode

Bentoml service.py sample

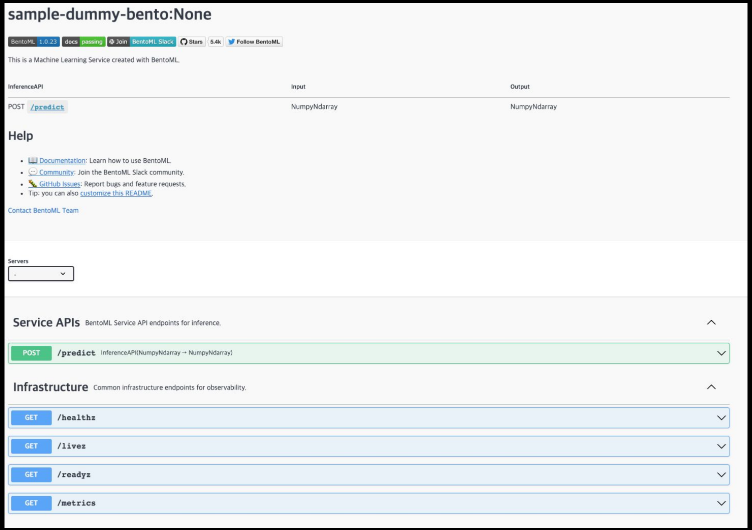
```python
import bentoml
import numpy as np
import torch
from torch import Tensor

runner = bentoml.pytorch.get("sample-dummy-model:latest").to_runner()

svc = bentoml.Service(name="sample-dummy-bento", runners=[runner])


@svc.api(
    input=bentoml.io.NumpyNdarray.from_sample(np.array([[1.1, 2.2], [3.3, 4.4]])),
    output=bentoml.io.NumpyNdarray.from_sample(np.array([[0.0, 0.0], [0.0, 0.0]])),
    doc="description....",
)
async def predict(input_array: np.array) -> np.array:
    inference_output: Tensor = await runner.async_run(torch.tensor(input_array))
    return inference_output.detach().cpu().numpy()
```

Bentoml serve service:svc (develop mode serving)
Default port : 3000

```
❯ bentoml serve service:svc --development --reload
```

BentoML: Quick Start [Model Serving] dev mode

```
❯ bentoml serve service:svc --development --reload
```



API Docs localhost:3000
builtin Healthy check API

BentoML: Quick Start [Model Serving] dev mode

```
❯ bentoml serve service:svc --development --reload
```

Latency, Percentile별 latency
Builtin Log, metrics

localhost:3000/metrics

```
~/workspace/PyConKR2023-ModelServing/sample_bento  on  main +70 !21 ?24
❯ bentoml serve service:svc --development --port 13000
2023-08-09T06:04:45+0900 [INFO] [cli] Prometheus metrics for HTTP BentoServer from "service:svc" can be accessed at http://localhost:13000/metrics.
2023-08-09T06:04:46+0900 [INFO] [cli] Starting production HTTP BentoServer from "service:svc" listening on http://0.0.0.0:13000 (Press CTRL+C to quit)
2023-08-09T06:04:47+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=GET,path=/,type=,length=) (status=200,type=text/html; charset=utf-8,length=2945) 0.251ms (trace=661809284fc4f6eb3675037bc522c908,span=252a 731d7016,sampled=0,service.name=sample-dummy-bento)
2023-08-09T06:04:47+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=GET,path=/docs.json,type=,length=) (status=200,type=application/json,length=4901) 22.059ms (trace=8d6cc559b1d57ed6111e4583b5f2ab14,span=6c ec1bf2192c,sampled=0,service.name=sample-dummy-bento)
2023-08-09T06:04:51+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=POST,path=/predict,type=application/json,length=54) (status=200,type=application/json,length=24) 15.230ms (trace=0d7773958f9ad5d61604d4f86 b9a,span=f751ee4fd1687f79,sampled=0,service.name=sample-dummy-bento)
2023-08-09T06:04:52+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=POST,path=/predict,type=application/json,length=54) (status=200,type=application/json,length=24) 4.478ms (trace=b1c6f1374c56da3d40f908a9fa 62,span=8731bd1a1aa407d8,sampled=0,service.name=sample-dummy-bento)
2023-08-09T06:04:52+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=POST,path=/predict,type=application/json,length=54) (status=200,type=application/json,length=24) 1.941ms (trace=bcaa6bf9f04c748270a6e99512 rd3,span=7efbd658ad7edbef,sampled=0,service.name=sample-dummy-bento)
2023-08-09T06:04:52+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=POST,path=/predict, =sample-dummy-bento)
2023-08-09T06:04:52+0900 [INFO] [api_server:sample-dummy-bento:1] 127.0.0.1:59889 (scheme=http,method=POST,path=/predict, =sample-dummy-bento)
```

```
# HELP bentoml_api_server_request_total Multiprocess metric
# TYPE bentoml_api_server_request_total counter
bentoml_api_server_request_total{endpoint="/",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_total{endpoint="/docs.json",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_total{endpoint="/predict",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 5.0
bentoml_api_server_request_total{endpoint="/metric",http_response_code="404",service_name="sample-dummy-bento",service_version="not available"} 1.0
# HELP bentoml_api_server_request_duration_seconds Multiprocess metric
# TYPE bentoml_api_server_request_duration_seconds histogram
bentoml_api_server_request_duration_seconds_sum{endpoint="/",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 0.020402750000357628
bentoml_api_server_request_duration_seconds_sum{endpoint="/docs.json",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 0.026632666995283216
bentoml_api_server_request_duration_seconds_sum{endpoint="/predict",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 0.03590954205719754
bentoml_api_server_request_duration_seconds_sum{endpoint="/metric",http_response_code="404",service_name="sample-dummy-bento",service_version="not available"} 0.0019280409906059504
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.005",service_name="sample-dummy-bento",service_version="not available"} 0.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.01",service_name="sample-dummy-bento",service_version="not available"} 0.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.025",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.05",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.075",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.1",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.25",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.5",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="0.75",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="1.0",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="2.5",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="5.0",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="7.5",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="10.0",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/",http_response_code="200",le="+Inf",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_count{endpoint="/",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.005",service_name="sample-dummy-bento",service_version="not available"} 0.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.01",service_name="sample-dummy-bento",service_version="not available"} 0.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.025",service_name="sample-dummy-bento",service_version="not available"} 0.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.05",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.075",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.1",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.25",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.5",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="0.75",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="1.0",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="2.5",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="5.0",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="7.5",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="10.0",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/docs.json",http_response_code="200",le="+Inf",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_count{endpoint="/docs.json",http_response_code="200",service_name="sample-dummy-bento",service_version="not available"} 1.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/predict",http_response_code="200",le="0.005",service_name="sample-dummy-bento",service_version="not available"} 2.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/predict",http_response_code="200",le="0.01",service_name="sample-dummy-bento",service_version="not available"} 4.0
bentoml_api_server_request_duration_seconds_bucket{endpoint="/predict",http_response_code="200",le="0.025",service_name="sample-dummy-bento",service_version="not available"} 5.0
```

BentoML: Quick Start [Model Server Build]

Bentofile.yaml sample

bentofile.yaml × service.py ×

```yaml
service: "service:svc"
labels:
  owner: kimsoungryoul
  stage: prod
include:
  - "service.py"
exclude:
  - "tests/*"
python:
  packages:
    - "torch"
```

Bento package sample

- sample_bento
  - bentofile.yaml
  - service.py

$ ./sample_bento
Command Example
bentoml build -f ./bentofile.yaml . --version=2023-08-13

```
❯ bentoml build
Building BentoML service "sample-dummy-bento:eethtnqnek3vmzx5" from build context "/Users/user/workspace/i_am_python/sample_bento".
Packing model "sample-dummy-model:2023-08-14"
Locking PyPI package versions.
WARNING: using legacy resolver is deprecated and will be removed in future versions. The default resolver will be change to 'backtracking' in 7.0.0 version. Specify --resolver=backtracking to silence this warning.

BENTOML

Successfully built Bento(tag="sample-dummy-bento:eethtnqnek3vmzx5").
```

Containerize Bento
BentoML manage dockerfile In framework

Command Example
bentoml containerize sample-dummy-bento:latest

```
❯ bentoml containerize sample-dummy-bento:latest
Building docker image for sample-dummy-bento:pt72u2aneokoazx5
[+] Building 43.1s (15/15) FINISHED
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 1.82kB                                              0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                 1.1s
 => [internal] load build context                                                   0.0s
 => => transferring context: 18.62kB                                                0.0s
 => [base-debian  1/10] FROM docker.io/library/python:3.10-slim@sha256:7dc5b4e948acd18c1633b0e593ad0224298646612ce7d0b5ac6d4e17616d7e4b   0.0s
 => CACHED [base-debian  2/10] RUN rm -f /etc/apt/apt.conf.d/docker-clean; echo 'Binary::apt::APT::Keep-Downloaded-Packages "true";' > /etc/apt/apt.conf.d/keep-cache   0.0s
 => CACHED [base-debian  3/10] RUN --mount=type=cache,target=/var/lib/apt     --mount=type=cache,target=/var/cache/apt     apt-get update -y     && apt-get install -q -y --no-install-recommends --allow-remove-essential ca-certificates gnupg2 bash build-essential   0.0s
 => CACHED [base-debian  4/10] RUN groupadd -g 1034 -o bentoml && useradd -m -u 1034 -g 1034 -o -r bentoml   0.0s
 => CACHED [base-debian  5/10] RUN mkdir /home/bentoml/bento && chown bentoml:bentoml /home/bentoml/bento -R   0.0s
 => CACHED [base-debian  6/10] WORKDIR /home/bentoml/bento                          0.0s
 => [base-debian  7/10] COPY --chown=bentoml:bentoml . ./                           0.0s
 => [base-debian  8/10] RUN --mount=type=cache,target=/root/.cache/pip bash -euxo pipefail /home/bentoml/bento/env/python/install.sh   38.9s
 => [base-debian  9/10] RUN rm -rf /var/lib/{apt,cache,log}                         0.2s
 => [base-debian 10/10] RUN chmod +x /home/bentoml/bento/env/docker/entrypoint.sh   0.2s
 => exporting to image                                                              2.4s
 => => exporting layers                                                             2.4s
 => => writing image sha256:1d88d84e95946a2d56065f124b7d303ee01b47d2a5c703c6de98839b6aa2f54e   0.0s
 => => naming to docker.io/library/sample-dummy-bento:pt72u2aneokoazx5              0.0s
Successfully built docker image for "sample-dummy-bento:latest" with tags "sample-dummy-bento:pt72u2aneokoazx5"
To run your newly built Bento container, pass "sample-dummy-bento:pt72u2aneokoazx5" to "docker run". For example: "docker run -it --rm -p 3000:3000 sample-dummy-bento:pt72u2aneokoazx5 serve --production".
```

```
❯ docker image ls
REPOSITORY            TAG                 IMAGE ID       CREATED              SIZE
sample-dummy-bento    pt72u2aneokoazx5    1d88d84e9594   About a minute ago   924MB
```

```
❯ docker run -d --rm -p 3000:3000 sample-dummy-bento:pt72u2aneokoazx5 serve --production
```

BentoML: Quick Start [Summary]

1. Support model version managing (cloud storage 연동가능)
2. Support api-server SDK(like a fastapi)
3. Bentoml support containerize (manage dockerfile in BentoML Self)

There is almost features to Model Serving

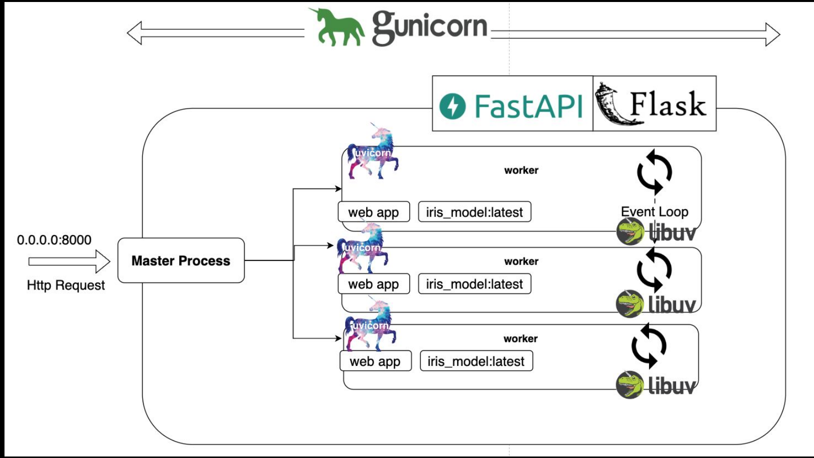
BentoML: Architecture [Model Serving]

Why does not recommend fastapi & flask?

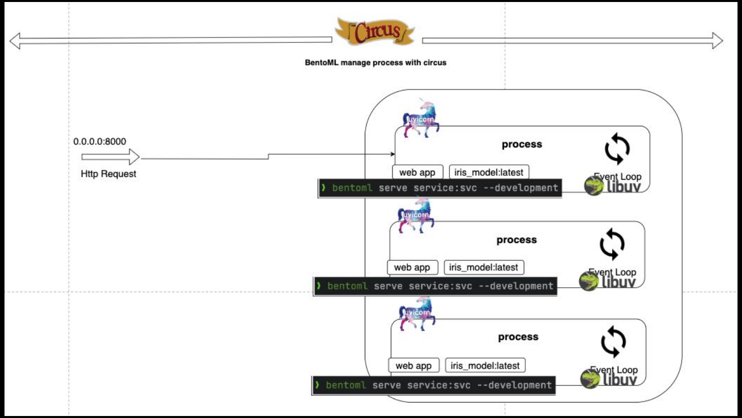
Support model versioning? ✖
Support default builtIn metrics & logging? ✖
Support containerizing? (auto build) ✖
Support Managing Backend Worker? ✖

```
❯ bentoml serve service:svc --development
❯ bentoml serve service:svc --development
❯ bentoml serve service:svc --development
```

```
gunicorn config.asgi:application --workers 3 -k uvicorn.workers.UvicornWorker --bind 0.0.0.0:8000
```



=

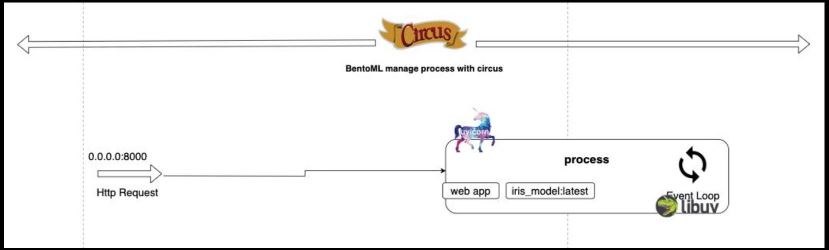


Model Serving's performance key is **Backend Model Worker(Runner)** not API-Server
Web framework(flask, fastapi) does not manage Backend Model Worker separately

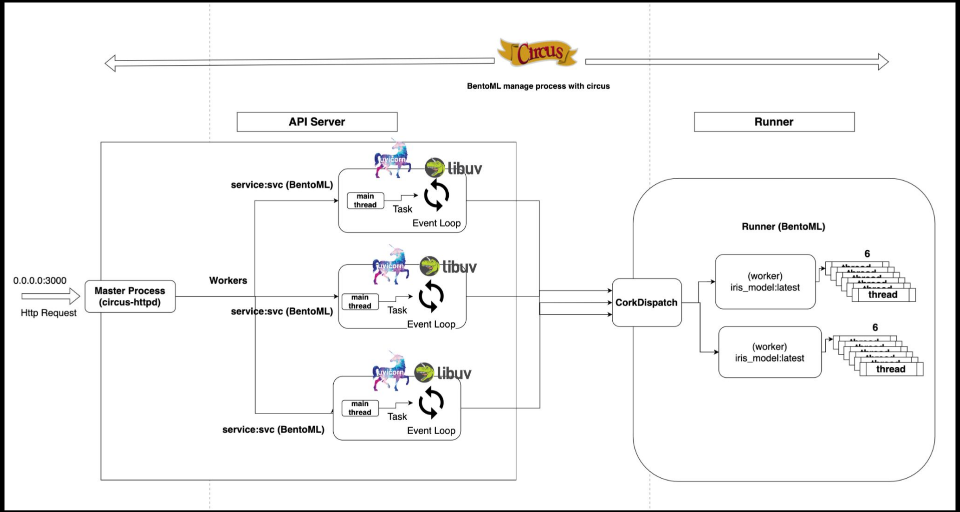
BentoML: Architecture [Model Serving] production mode

What will happen in bento after deploy with production mode

```
❯ bentoml serve service:svc --development
```




```
❯ bentoml serve service:svc --production
```



bentoML builtin circus(process manager) & uvicorn

Custom set Uvicorn worker count

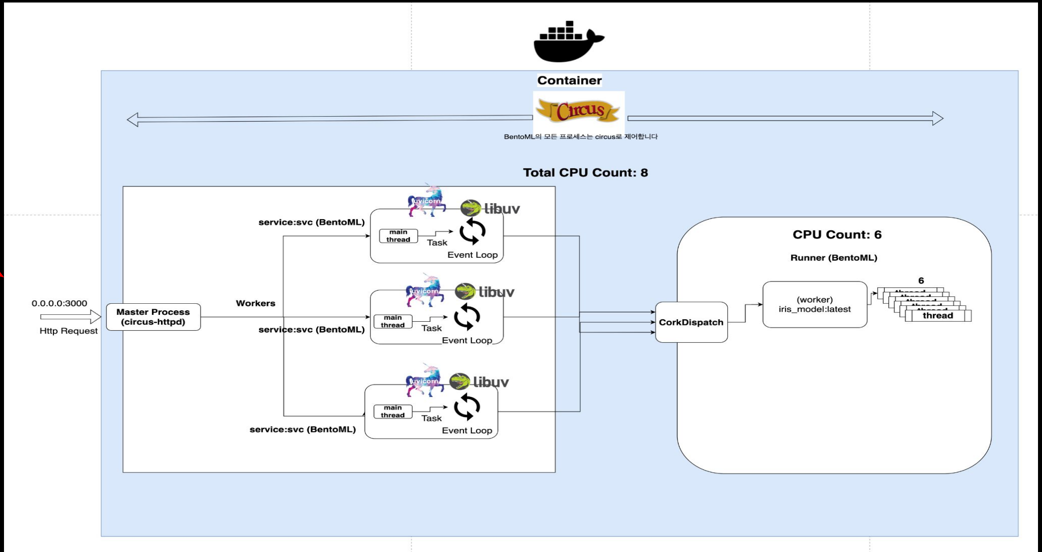
bentoml serve –api-workers=3

BentoML: Bento Deploy Configuration Example

```
❯ docker run -d --rm -e BENTOML_CONFIG=/home/bentoml/bento/src/configuration.yaml iris_classifier:latest serve --production
```

assuming a container is allocated 8 CPUs

```yaml
# configuration.yaml
version: 1
api_server:
  workers: 3
runners:
  resources:
    cpu: 6
```

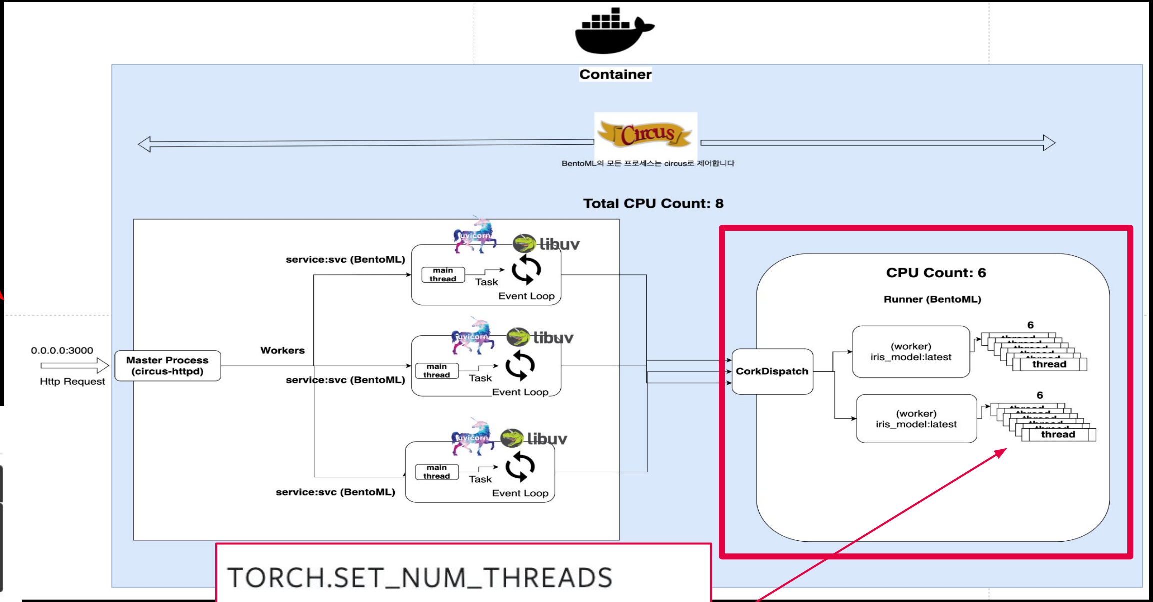


BentoML: Bento Deploy Configuration Example

```
❯ docker run -d --rm -e BENTOML_CONFIG=/home/bentoml/bento/src/configuration.yaml iris_classifier:latest serve --production
```

```yaml
# configuration.yaml
version: 1
api_server:
  workers: 3
runners:
  resources:
    cpu: 6
  workers_per_resource: 2
```

Container is allocated 8 cpu core



GPU allocation is also same to cpu

Global configuration (Applies to all runners) | Individual Runner

configuration.yml

```yaml
runners:
  resources:
    nvidia.com/gpu: 2
  workers_per_resource: 2
```

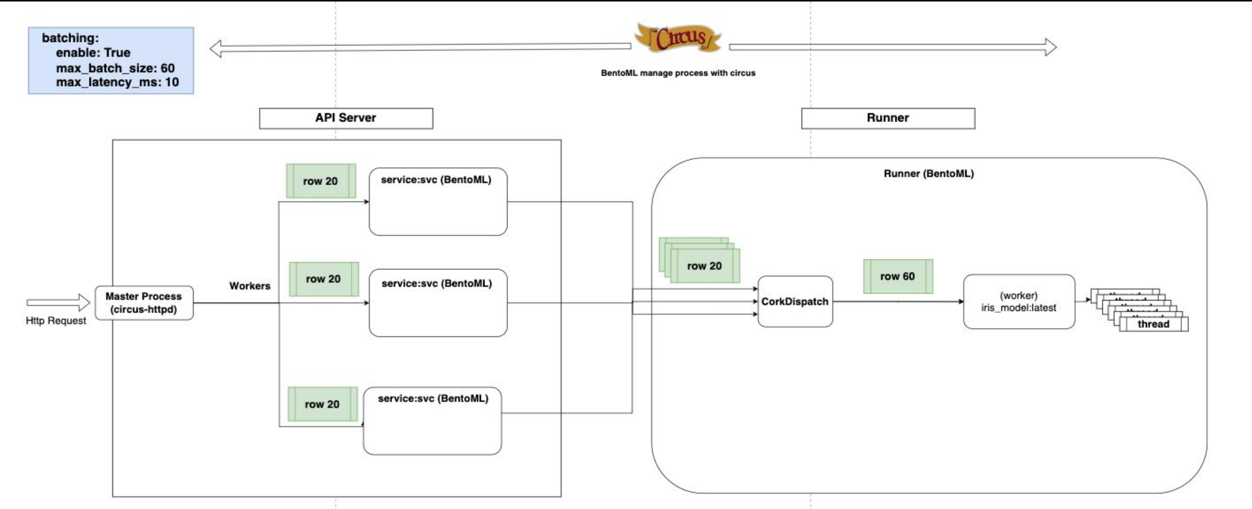
TORCH.SET_NUM_THREADS

torch.set_num_threads(*int*)

Sets the number of threads used for intraop parallelism on CPU.

BentoML: Bento(Model-Serve) Configuration (batchable)

Batchable Option is not Silver Bullet




```yaml
# configuration.yaml
runners:
  batching:
    enabled: true
    max_batch_size: 60
    max_latency_ms: 10
```

In general, enabling the batchable option **increases throughput** but **slows down latency**.

This makes sense.
The dispatcher intercepts the packets that are delivered to the runner and waits until several packets are gathered (max batch size).
packets (max_batch_size) before forwarding them to the Runner.

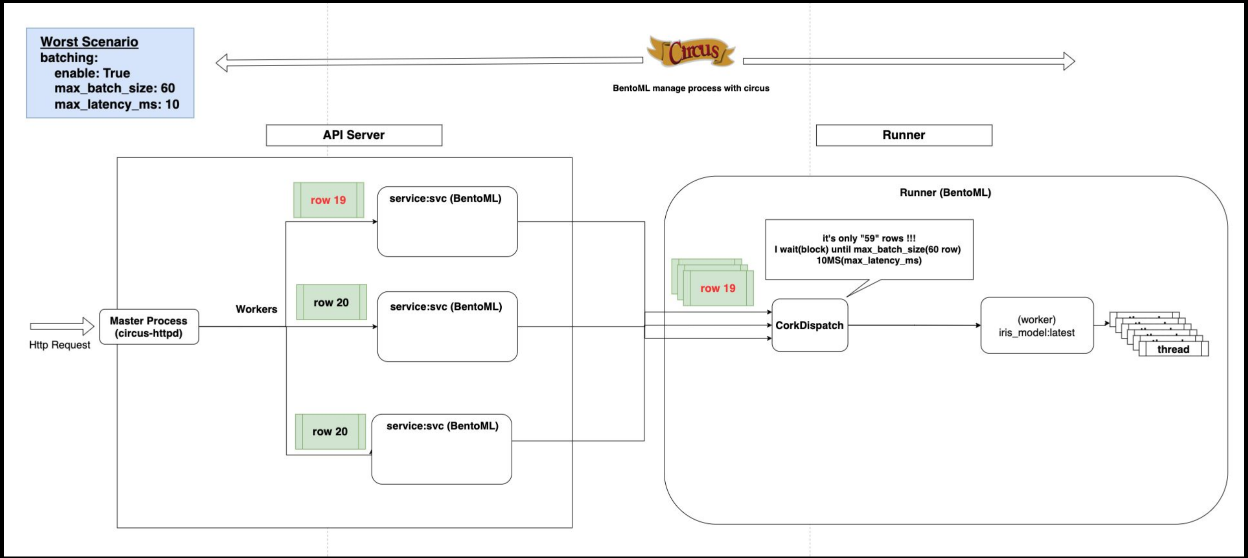
Another way of saying this is that a request that could have been answered in 30ms might take 40ms because the Diapatcher waits an extra 10ms (max_latency_ms).

Of course, depending on where the bottleneck is, batchable options can have a positive impact on latency.

These batch options are also available on **torchserve tensorflowserving triton-inference-server**

BentoML: Bento(Model-Serve) Configuration (batchable)

Batchable Option is not Silver Bullet




```yaml
# configuration.yaml
runners:
  batching:
    enabled: true
    max_batch_size: 60
    max_latency_ms: 10
```

In general, enabling the batchable option **increases throughput** but **slows down latency**.

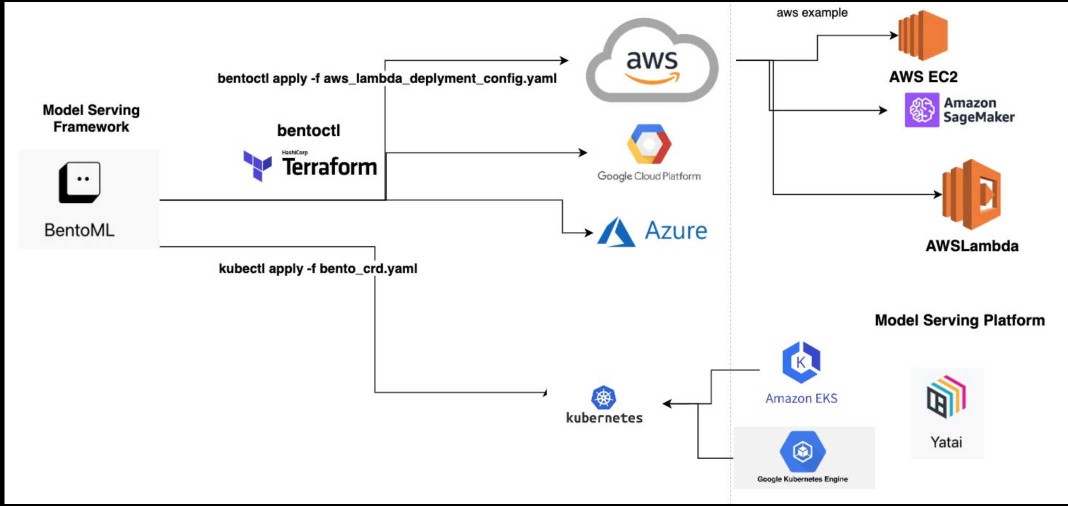
This makes sense.
The dispatcher intercepts the packets that are delivered to the runner and waits until several packets are gathered (max batch size).
packets (max_batch_size) before forwarding them to the Runner.

Another way of saying this is that a request that could have been answered in 30ms might take 40ms because the Diapatcher waits an extra 10ms (max_latency_ms).

Of course, depending on where the bottleneck is, batchable options can have a positive impact on latency.

These batch options are also available on **torchserve tensorflowserving triton-inference-server**

bentoctl Deploy bentoML to Cloud (AWS, GCP, Azure)



<u>BentoML is manage dockerfile</u>
And containerize

<u>bentoctl manage terraform (.tf) file</u>
deploy bento to Vendor(aws, gcp) of Cloud Resource

bentoctl: Deploy bentoML to Cloud (AWS, GCP, Azure)

🚀 Fast model deployment on any cloud

Bentoctl-CI passing | Documentation | Join Community Slack

bentoctl helps deploy any machine learning models as production-ready API endpoints on the cloud, supporting AWS SageMaker, AWS Lambda, EC2, Google Compute Engine, Azure, Heroku and more.

Supported Platforms:

- AWS Lambda
- AWS SageMaker
- AWS EC2
- Google Cloud Run
- Google Compute Engine
- Azure Container Instances
- Heroku

Operator List

Official Operators

Operator Name	Github Repo	Status [Migrated to 1.0]	Guides
aws-lambda	https://github.com/bentoml/aws-lambda-deploy/tree/main	✅	
aws-sagemaker	https://github.com/bentoml/aws-sagemaker-deploy	✅	
aws-ec2	https://github.com/bentoml/aws-ec2-deploy	✅	
google-compute-engine	https://github.com/bentoml/google-compute-engine-deploy	✅	
google-cloud-run	https://github.com/bentoml/google-cloud-run-deploy	✅	
azure-functions	https://github.com/bentoml/azure-functions-deploy	✅	
azure-container-instances	https://github.com/bentoml/azure-container-instances-deploy	✅	
Heroku	https://github.com/bentoml/heroku-deploy	✅	

https://github.com/bentoml/bentoctl

bentoctl Quick Start

```
❯ pip install bentoctl
```

1. Install bentoctl

```
❯ bentoctl operator install aws-lambda
/Users/user/model-serving-asdf111/pl2-pctr-bento/.venv/bin/b
  from bentoctl.cli import bentoctl
Installed aws-lambda!
```

2. install Cloud Resource operator

```
❯ bentoctl init
/Users/user/model-serving-asdf111/pl2-pctr-bento/.venv/bin/bentoctl:5: Depr
  from bentoctl.cli import bentoctl
Bentoctl Interactive Deployment Config Builder

Welcome! You are now in interactive mode.

This mode will help you setup the deployment_config.yaml file required for
deployment. Fill out the appropriate values for the fields.

(deployment config will be saved to: ./deployment_config.yaml)

api_version: v1
name: pycon_example
operator:
    name: aws-lambda
template: terraform
spec:
    region: ap-northeast-2
    timeout: 10
    memory_size: 512
filename for deployment_config [deployment_config.yaml]:
deployment config generated to: deployment_config.yaml
✨ generated template files.
  - bentoctl.tfvars
  - main.tf
```

3. bentoctl init

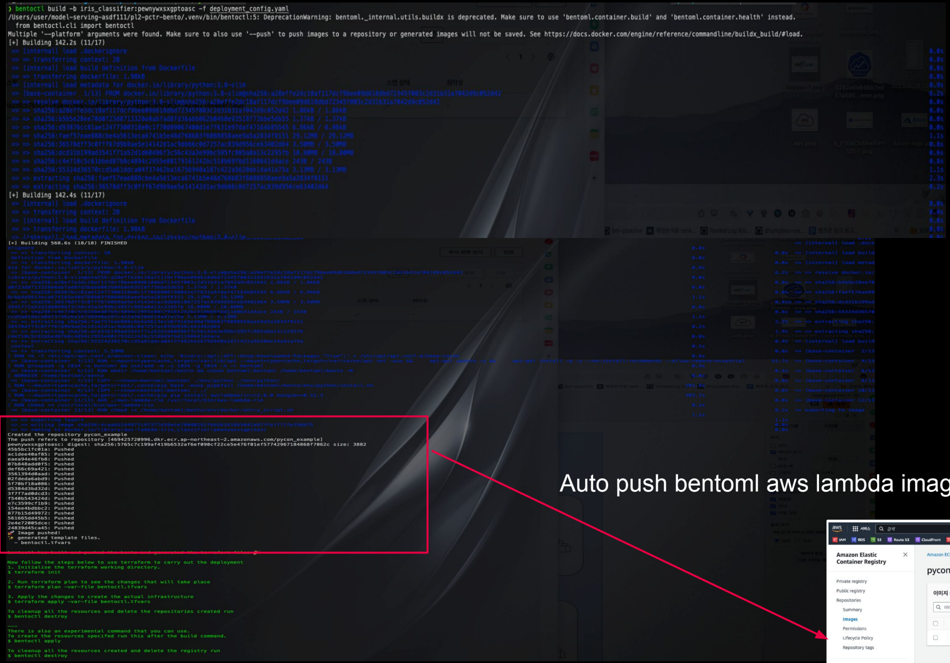
```
❯ cat deployment_config.yaml
api_version: v1
name: pycon_example
operator:
  name: aws-lambda
template: terraform
spec:
  region: ap-northeast-2
  timeout: 10
  memory_size: 512
```

terraform file is created & managed by bentoctl

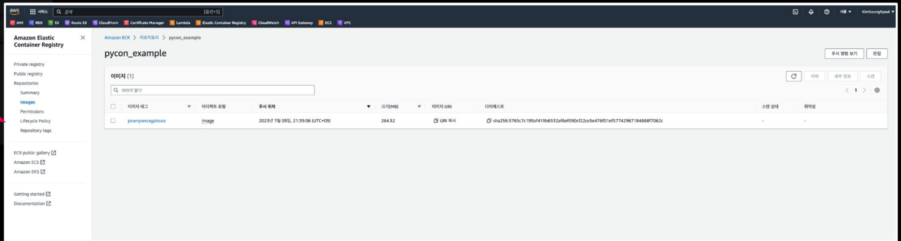
bentoctl Quick Start

Rebuild bento to AWS-Lambda Image Base

$ bentoctl build iris_classifier:2023-08-13 -f deployment_config.yaml



Auto push bentoml aws lambda image to ECR



deploy bento with aws-lambda

$ bentoctl apply -f deployment_config.yaml

```
bentoctl apply -f deployment_config.yaml

data.aws_ecr_repository.service: Reading...
data.aws_ecr_repository.service: Read complete after 2s [id=testlambda]
data.aws_ecr_image.service_image: Reading...
data.aws_ecr_image.service_image: Read complete after 1s [id=sha256:b0637046b983acc0f52d4b387ddabcf8bec9d61b214b169c

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_apigatewayv2_api.lambda will be created
  + resource "aws_apigatewayv2_api" "lambda" {
      + api_endpoint                 = (known after apply)
      + api_key_selection_expression = "$request.header.x-api-key"
      + arn                          = (known after apply)

... other output from terraform

aws_apigatewayv2_integration.lambda: Creation complete after 2s [id=jeaalbk]
aws_apigatewayv2_route.services: Creating...
aws_apigatewayv2_route.root: Creating...
aws_cloudwatch_log_group.lg: Creation complete after 4s [id=/aws/lambda/testlambda-function]
aws_apigatewayv2_route.root: Creation complete after 3s [id=z8i4az1]
aws_apigatewayv2_route.services: Creation complete after 3s [id=zpxehl0]

Apply complete! Resources: 11 added, 0 changed, 0 destroyed.

Outputs:

endpoint = "https://ti7d9u5jkl.execute-api.us-west-1.amazonaws.com/"
function_name = "testlambda-function"
image_tag = "213386773652.dkr.ecr.us-west-1.amazonaws.com/testlambda:of7al2xz6g2vh74c"
```

AWS Apigatewayv2, lambda cloudwatch was created

BentoML: OpenLLM (Large Language Model)

🦾 OpenLLM

pypi v0.1.20 | ci failing | Follow Us @bentomlai | Join Us OpenLLM

An open platform for operating large language models (LLMs) in production.
Fine-tune, serve, deploy, and monitor any LLMs with ease.

LLM (Large Language Models)

https://github.com/bentoml/OpenLLM

🧩 Supported Models

The following models are currently supported in OpenLLM. By default, OpenLLM doesn't include dependencies to run all models. The extra model-specific dependencies can be installed with the instructions below:

Model	CPU	GPU	Installation	Model Ids
chatglm	❌	✅	`pip install "openllm[chatglm]"`	thudm/chatglm-6b, thudm/chatglm-6b-int8, thudm/chatglm-6b-int4, thudm/chatglm2-6b, thudm/chatglm2-6b-int4
dolly-v2	✅	✅	`pip install openllm`	databricks/dolly-v2-3b, databricks/dolly-v2-7b, databricks/dolly-v2-12b
falcon	❌	✅	`pip install "openllm[falcon]"`	tiiuae/falcon-7b, tiiuae/falcon-40b, tiiuae/falcon-7b-instruct, tiiuae/falcon-40b-instruct
flan-t5	✅	✅	`pip install "openllm[flan-t5]"`	google/flan-t5-small, google/flan-t5-base, google/flan-t5-large, google/flan-t5-xl, google/flan-t5-xxl
gpt-neox	❌	✅	`pip install openllm`	eleutherai/gpt-neox-20b
mpt	✅	✅	`pip install "openllm[mpt]"`	mosaicml/mpt-7b, mosaicml/mpt-7b-instruct, mosaicml/mpt-7b-chat, mosaicml/mpt-7b-storywriter, mosaicml/mpt-30b, mosaicml/mpt-30b-instruct, mosaicml/mpt-30b-chat
opt	✅	✅	`pip install openllm`	facebook/opt-125m, facebook/opt-350m, facebook/opt-1.3b, facebook/opt-2.7b, facebook/opt-6.7b, facebook/opt-66b
stablelm	✅	✅	`pip install openllm`	stabilityai/stablelm-tuned-alpha-3b, stabilityai/stablelm-tuned-alpha-7b, stabilityai/stablelm-base-alpha-3b, stabilityai/stablelm-base-alpha-7b
starcoder	❌	✅	`pip install "openllm[starcoder]"`	bigcode/starcoder, bigcode/starcoderbase

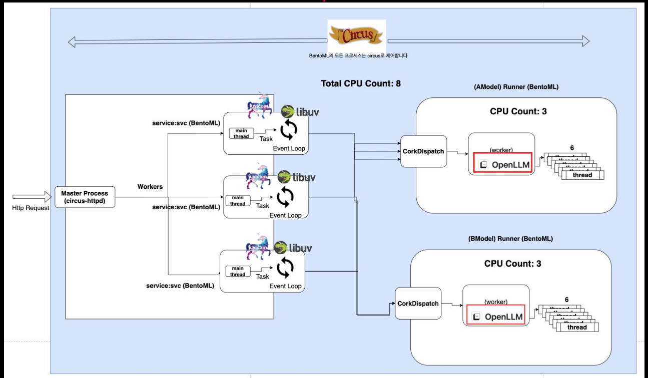
⚙️ Integrations

OpenLLM is not just a standalone product; it's a building block designed to integrate with other powerful tools easily. We currently offer integration with BentoML and LangChain.

BentoML

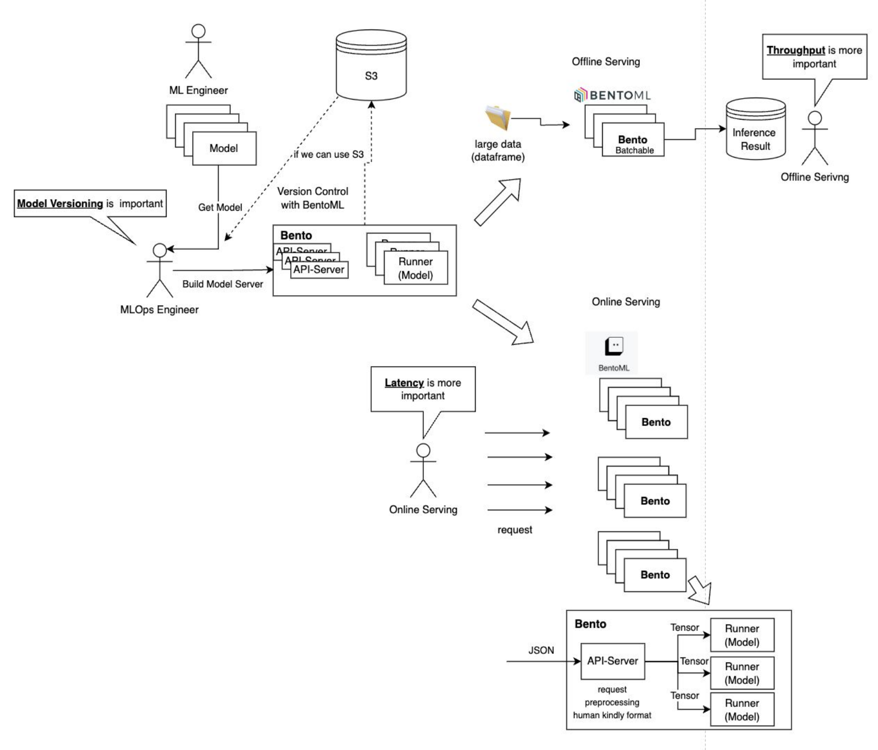
OpenLLM models can be integrated as a Runner in your BentoML service. These runners have a `generate` method that takes a string as a prompt and returns a corresponding output string. This will allow you to plug and play any OpenLLM models with your existing ML workflow.

```python
import bentoml
import openllm

model = "opt"

llm_config = openllm.AutoConfig.for_model(model)
llm_runner = openllm.Runner(model, llm_config=llm_config)

svc = bentoml.Service(
    name=f"llm-opt-service", runners=[llm_runner]
)

@svc.api(input=Text(), output=Text())
async def prompt(input_text: str) -> str:
    answer = await llm_runner.generate(input_text)
    return answer
```



https://github.com/bentoml/OpenLLM

BentoML UseCase In NAVER Overview



- Offline Serving (throughput is important)
- Online Serving (Latency is important)

BentoML UseCase In NAVER : troubleshooting 1 (pydantic)

Do not use pydantic (even if pydantic>=2.x) if you <u>need high-end performance</u> (recommend to use TypedDict)

Batch size : 10, pydantic

```python
# iris_classify/profiling_bento.py
import numpy as np

from service import svc, Iris, IrisFeatures

runners = svc.runners

for runner in runners:
    runner.init_local(quiet=True)

sample_input = IrisFeatures(
    features=[
        Iris(
            sepal_len=6.2,
            sepal_width=3.2,
            petal_len=5.2,
            petal_width=2.2,
        )
        for _ in range(0, 10)
    ]
)

result: np.array = svc.apis["classify"].func(iris_features_pydantic=sample_input)

print(result)
```

```python
# iris_classify/service.py
import numpy as np
import pandas as pd
from pydantic import BaseModel

import bentoml
from bentoml.io import JSON
from bentoml.io import NumpyNdarray
from line_profiler_pycharm import profile

iris_clf_runner = bentoml.sklearn.get("iris_clf_with_feature_names:latest").to_runner()

svc = bentoml.Service("iris_classifier_pydantic", runners=[iris_clf_runner])


class Iris(BaseModel):
    sepal_len: float
    sepal_width: float
    petal_len: float
    petal_width: float


class IrisFeatures(BaseModel):
    features: list[Iris]


Time in function:                                                          1498000 0.000000 s
Colormap '%Time':                                                          0%   100%
                                                                           % Time    Hits   Time []   Time / Hit []
@svc.api(input=JSON(pydantic_model=IrisFeatures), output=NumpyNdarray())
@profile
def classify(iris_features_pydantic: IrisFeatures) -> np.ndarray:
    iris_features_dict = [iris.dict() for iris in iris_features_pydantic.features]   3.7    1    55000     55000.0
    input_df = pd.DataFrame(iris_features_dict)                                      28.4   1    425000    425000.0
    result = iris_clf_runner.predict.run(input_df)                                   68.0   1    1018000   1018000.0
    return result                                                                    0.0    1    0         0.0
```

When Row Size is 10, class to dict serializing is only 3.7% time

BentoML UseCase In NAVER : troubleshooting 1 (pydantic)

Do not use pydantic (even if pydantic>=2.x) if you <u>need high-end performance</u> (recommend to use TypedDict)

Batch size : 1000 , Pydantic

```python
# iris_classify/profiling_bento.py
import numpy as np

from service import svc, Iris, IrisFeatures

runners = svc.runners

for runner in runners:
    runner.init_local(quiet=True)

sample_input = IrisFeatures(
    features=[
        Iris(
            sepal_len=6.2,
            sepal_width=3.2,
            petal_len=5.2,
            petal_width=2.2,
        )
        for _ in range(0, 1000) # 10 -> 1000
    ]
)

result: np.array = svc.apis["classify"].func(iris_features_pydantic=sample_input)

print(result)
```

Pydantic 을 사용한 예제 프로파일링 결과

```python
# iris_classify/service.py
import numpy as np
import pandas as pd
from pydantic import BaseModel

import bentoml
from bentoml.io import JSON
from bentoml.io import NumpyNdarray
from line_profiler_pycharm import profile

iris_clf_runner = bentoml.sklearn.get("iris_clf_with_feature_names:latest").to_runner()

svc = bentoml.Service("iris_classifier_pydantic", runners=[iris_clf_runner])


class Iris(BaseModel):
    sepal_len: float
    sepal_width: float
    petal_len: float
    petal_width: float


class IrisFeatures(BaseModel):
    features: list[Iris]
```

Time in function: 7360000 0.000000 s
Colormap '%Time': 0% – 100%

Code	% Time	Hits	Time []	Time / Hit []
`@svc.api(input=JSON(pydantic_model=IrisFeatures), output=NumpyNdarray())`				
`@profile`				
`def classify(iris_features_pydantic: IrisFeatures) -> np.ndarray:`				
`iris_features_dict = [iris.dict() for iris in iris_features_pydantic.features]`	48.7	1	3582000	3582000.0
`input_df = pd.DataFrame(iris_features_dict)`	12.1	1	892000	892000.0
`result = iris_clf_runner.predict.run(input_df)`	39.2	1	2886000	2886000.0
`return result`	0.0	1	0	0.0

but row Size is 1000 it has 48.7% (almost half time in total Latency)

BentoML UseCase In NAVER : troubleshooting 1 (pydantic)

Do not use pydantic (even if pydantic>=2.x) if you <u>need high performance</u> (recommend to use TypedDict)

Use TypedDict instead of pydantic

Batch size : 1000, TypedDict

```
7360000 0.000000 s
0%        100%
% Time        Hits    Time []    Time / Hit []

  48.7        1       3582000    3582000.0
  12.1        1       892000     892000.0
  39.2        1       2886000    2886000.0
   0.0        1       0          0.0
```

Profiling Result (use Pydantic)

Time 48.7 %-> 0 %

```python
# iris_classify/service.py
from typing import TypedDict

import numpy as np
import pandas as pd

import bentoml
from bentoml.io import JSON
from bentoml.io import NumpyNdarray
from line_profiler_pycharm import profile

iris_clf_runner = bentoml.sklearn.get("iris_clf_with_feature_names:latest").to_runner()

svc = bentoml.Service("iris_classifier_pydantic", runners=[iris_clf_runner])


class Iris(TypedDict):
    sepal_len: float
    sepal_width: float
    petal_len: float
    petal_width: float


class IrisFeatures(TypedDict):
    features: list[Iris]


Time in function:                    3982000 0.000000 s
Colormap '%Time':                    0%        100%
                                     % Time     Hits    Time []    Time / Hit []
@svc.api(input=JSON(), output=NumpyNdarray())
@profile
def classify(iris_features_pydantic: TypedDict) -> np.ndarray:
    iris_features_dict = iris_features_pydantic["features"]     0.0    1    1000       1000.0
    input_df = pd.DataFrame(iris_features_dict)                21.6    1    861000     861000.0
    result = iris_clf_runner.predict.run(input_df)             78.4    1    3120000    3120000.0
    return result                                               0.0    1    0          0.0
```

Profiling result (use TypedDict)

BentoML UseCase In NAVER : troubleshooting 2 (pandas DataFrame)

Do not use pandas in online serving if you <u>need high-end performance</u> (recommend to use numpy array)

dataframe is very heavy instance

```
import numpy as np
import pandas as pd
from line_profiler_pycharm import profile


Time in function:                          4458617000 0.000000 s
Colormap '%Time':                          0%          100%
                                           % Time          Hits      Time []     Time / Hit []
@profile
def create_instance(sample_data):
    dataframe = pd.DataFrame(sample_data)      68.5            100   3054541000    30545410.0
    dataframe[:5_000], dataframe[5_000:]       0.2             100   8769000       87690.0

    arr = np.array(sample_data)                31.3            100   1395016000    13950159.0
    arr[:5_000], arr[5_000:]                   0.0             100   291000        2910.0


sample_data = [[1.111 for _ in range(0, 500)] for row_size in range(0, 1_000)]
for _ in range(0, 100):
    create_instance(sample_data)
```

Pandas VS numpy speed comparison profiling Result

pandas : create instance double time slow , slicing is more 30 times sloy

Pandas is fast cause of numpy & Cython
But pandas calculate only single core so limitation is clear

pandas

MODIN

To use Modin, replace the pandas import:

Modin engine use multi core & support pandas Dataframe Interface

In online serving, pandas is not good solution
Modin is better but numpy is much better

Bentoml does not Modin IO Descriptor now

low level data structure is better like a typeddict or numpy

BentoML UseCase In NAVER : Online Serving (distributed Runner)

normal Bento Service

```python
# origin
iris_clf_runner1 = bentoml.sklearn.get("iris_clf_with_feature_names:latest").to_runner(name="iris_clf_runner1")

svc = bentoml.Service("iris_classifier_pydantic", runners=[iris_clf_runner1])

@svc.api(input=JSON(), output=NumpyNdarray())
async def classify(iris_features: TypedDict) -> np.ndarray:
    iris_features_list = iris_features["features"]

    input_data = np.array([list(aa.values()) for aa in iris_features_list])
    result1 = await iris_clf_runner1.predict.async_run(input_data)
    return result1
```

Create two same Runner & inference distribute rows

```python
# distributed-runner
iris_clf_runner1 = bentoml.sklearn.get("iris_clf_with_feature_names:latest").to_runner(name="iris_clf_runner1")
iris_clf_runner2 = bentoml.sklearn.get("iris_clf_with_feature_names:latest").to_runner(name="iris_clf_runner2")

svc = bentoml.Service("iris_classifier_pydantic", runners=[iris_clf_runner1,iris_clf_runner2])

@svc.api(input=JSON(), output=NumpyNdarray())
async def classify(iris_features: TypedDict) -> np.ndarray:
    iris_features_list = iris_features["features"]

    # Convert list to an array
    input_data = np.array([list(aa.values()) for aa in iris_features_list])
    result1, result2 = await asyncio.gather(
        iris_clf_runner1.predict.async_run(input_data[:250]),
        iris_clf_runner2.predict.async_run(input_data[250:]),
    )
    return np.concatenate((result1, result2), axis=0)
```

you're talking about BentoML with Naver,

Why are you only talking about preprocessing, which seems completely unrelated?

Because Data Distributed Runner is needed these things

```python
# test code
client = Client.from_url("http://localhost:13000")

latency_list = []

for _ in range(500):
    t = datetime.now()
    res = client.call("classify", random_input_size_fixed)
    tt = datetime.now() - t
    latency_list.append(tt.total_seconds())

print(f"AVG: {sum(latency_list)/ len(latency_list)}")
print(f"Median: {np.median(sorted(latency_list,reverse=True))}")
print("percentile: ", np.percentile(latency_list, [50, 75, 100], interpolation='nearest'))
```

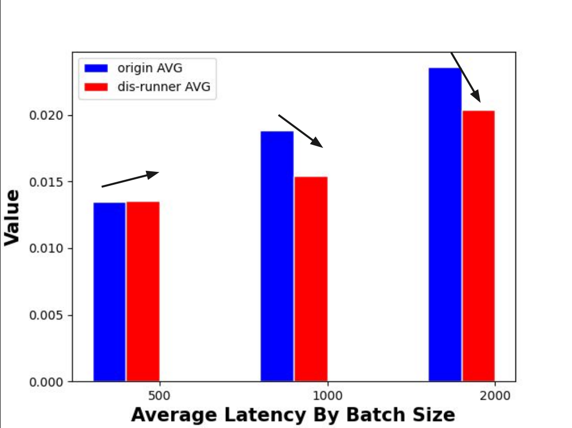
BentoML UseCase In NAVER : Online Serving (distributed Runner)

As a Result

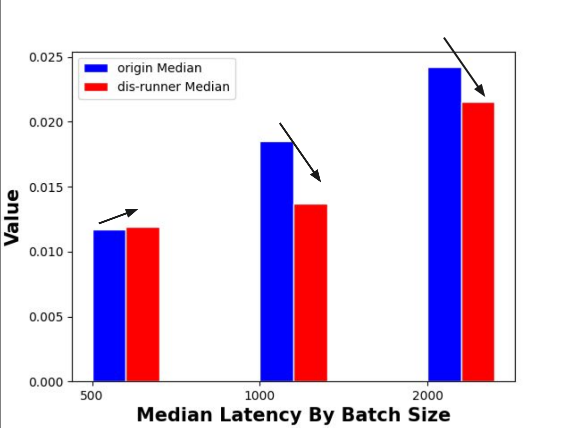
The larger the batch size, the more effective the distributed runner approach can be.

> The efficiency of the batch size is affected by the number of CPUs allocated to the Runner + the number of threads adjusted in MLFramework.

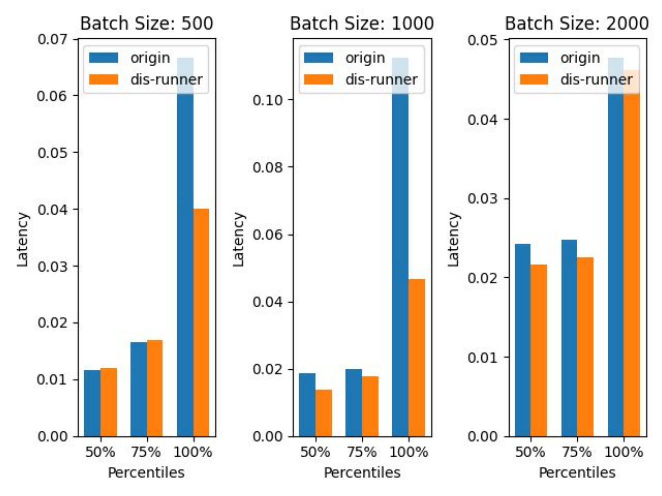
For the Iris_feature model, we can see that the Latency AVG and Median values are worse with a batch_size of 500, albeit slightly.
However, when the batch_size is larger (batch_size=1000), we see an improvement in latency.



Low is Better



Low is Better



Low is Better

dis-runner: distributed-runner

See the more detail **Distributed Runner Limitations and More efficient Usage**
https://github.com/KimSoungRyoul/PyConKR2023-ModelServing-BentoML/issues/5

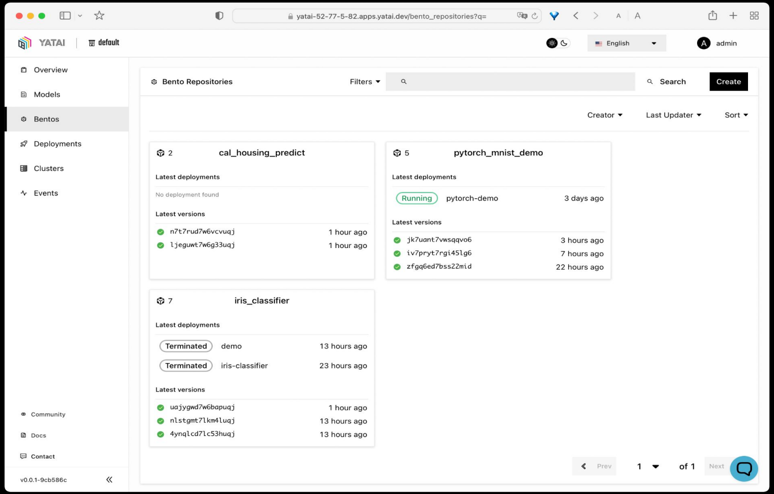
Model Serving(Inference) Platform with k8s

- Yatai (with BentoML)
- Kserve (standardized inference platform)

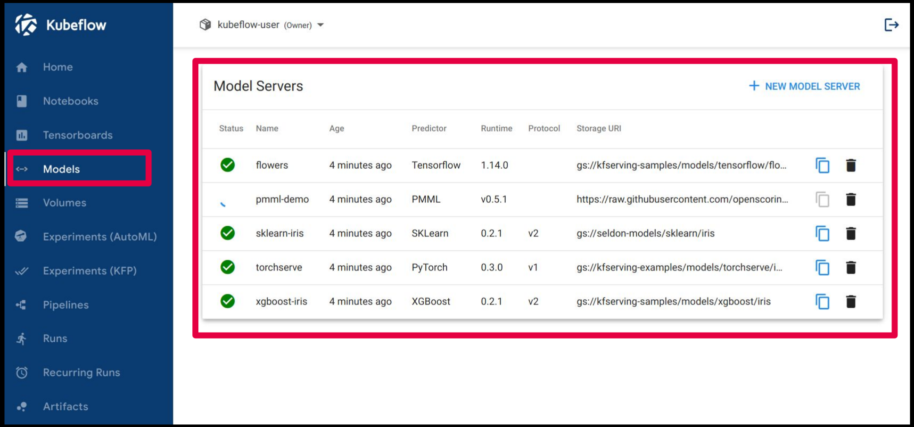
Model serving platform

ML workflow 의 일부인 Model Serving (Platform) 에 한정한다.

Model Serving Platform (k8s-based) (BentoCloud)



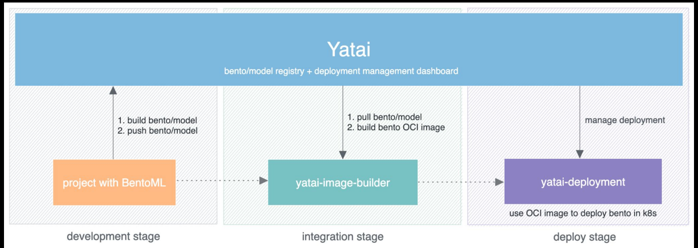
Model Serving Platform (k8s-based)



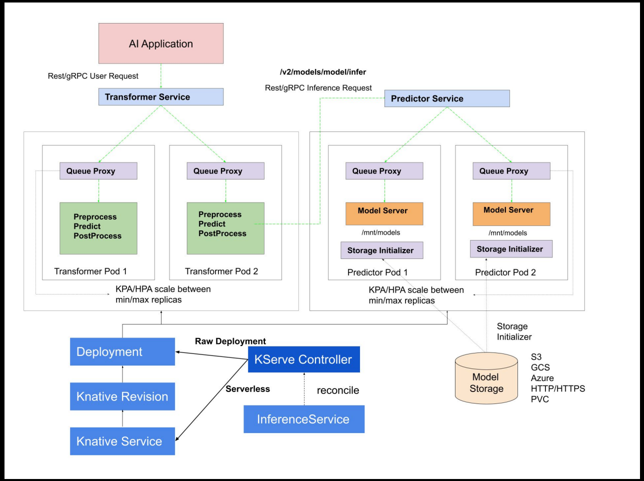
Serving Platform Concept

- Model Version Control — Bento Model , S3, GCS, ...
- Model Server Version Control — ECR, dockerhub (image)
- Deployment, Replica Control — K8S CRD (Custom Resource Definition)
- Observe — prometheus, grafana

Yatai



KServe



Serving Platform Component CRD

Platform support simple deployment system
Just set Yaml file & apply

Yatai CRD **BentoDeployment**

```yaml
# iris_bento_deployment.yaml
apiVersion: serving.yatai.ai/v2alpha1
kind: BentoDeployment
metadata:
  name: my-bento-deployment
  namespace: my-namespace
spec:
  bento: iris_classifier:pewnywxsxgptoasc
  ingress:
    enabled: true
```

```
kubectl apply -f iris_bento_depolyment.yaml
```

```
kubectl get bentodeployments
```

Kserve CRD **isvc (inferenceservice)**

```yaml
# image_classifier.yaml
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "torchserve"
spec:
  predictor:
    model:
      modelFormat:
        name: pytorch
      storageUri: gs://kfserving-examples/models/torchserve/image_classifier/v1
```

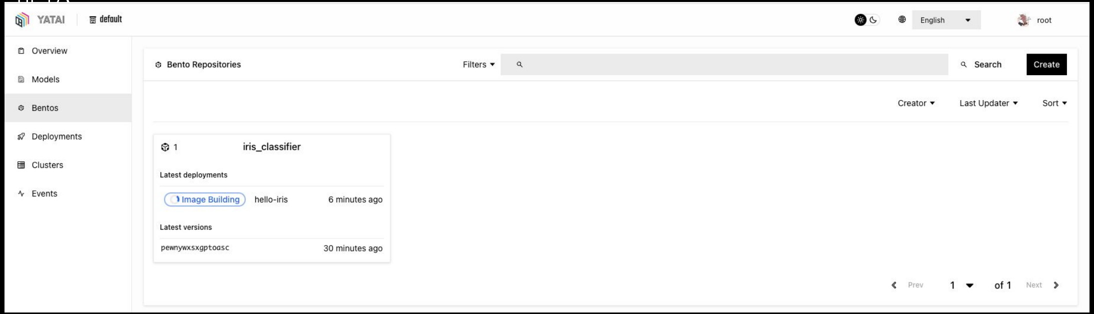
```
kubectl apply -f image_classifier.yaml
```

```
kubectl get isvc
```

Serving Platform Yatai

BentoML(프레임워크)은 모델의 버전관리, 모델서버 관리가 이미 가능하다.

이로 인해 yatai 에서는 그저 bento 의 이름과 버전만 명시하면 손쉽게 배포가 가능하다.
또한 Yatai 라는 Platform 수준에서 model 과 model-server(=bento) 버전관리를 할수있다. (= Model, Model Server Registry 제공)



Serving Platform Component CRD

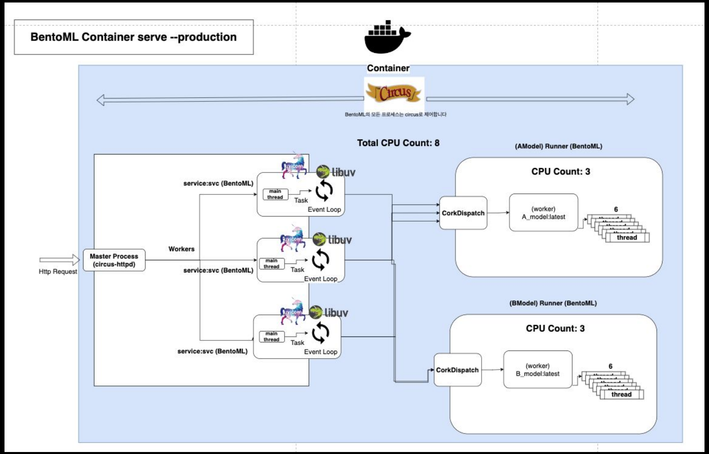
Kserve does not support Model Registry. Also, the frameworks used by kserve are different for each model.
For this reason, kserve provides an additional CustomResource called servingruntime.
Servingruntime is responsible for mapping the models that need to be deployed to the Model Serving Frameworks that can be deployed.
If there is no servingruntime inside the deployed kserve that supports the model specified in the isvc deployment, it will not be deployed.
As mentioned earlier, you need to understand each MLFramework and Serving Framework.
You need an MLOps team to manage Kserve (even if you use GCP kubeflow...)

```
❯ kubectl get clusterservingruntimes
NAME                       DISABLED   MODELTYPE    CONTAINERS
kserve-lgbserver                      lightgbm     kserve-container
kserve-mlserver                       sklearn      kserve-container
kserve-paddleserver                   paddle       kserve-container
kserve-pmmlserver                     pmml         kserve-container
kserve-sklearnserver                  sklearn      kserve-container
kserve-tensorflow-serving             tensorflow   kserve-container
kserve-torchserve                     pytorch      kserve-container
kserve-tritonserver                   tensorrt     kserve-container
kserve-xgbserver                      xgboost      kserve-container
```

Name	Supported Model Formats
kserve-lgbserver	LightGBM
kserve-mlserver	SKLearn, XGBoost, LightGBM, MLflow
kserve-paddleserver	Paddle
kserve-pmmlserver	PMML
kserve-sklearnserver	SKLearn
kserve-tensorflow-serving	TensorFlow
kserve-torchserve	PyTorch
kserve-tritonserver	TensorFlow, ONNX, PyTorch, TensorRT
kserve-xgbserver	XGBoost

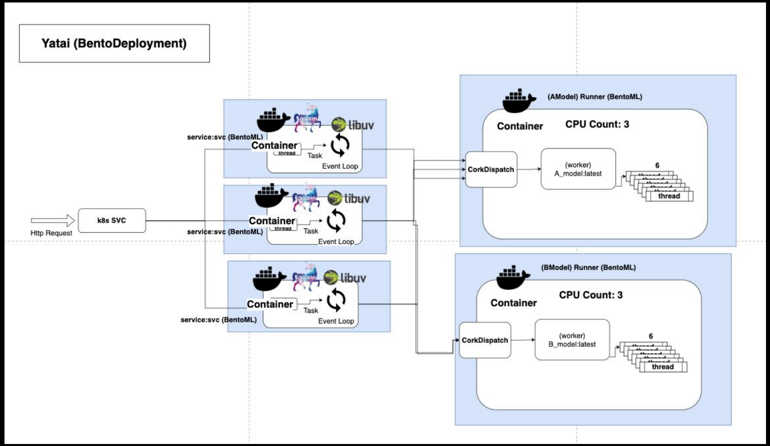
https://kserve.github.io/website/0.10/modelserving/servingruntimes/

Serving Platform Yatai BentoDeployment

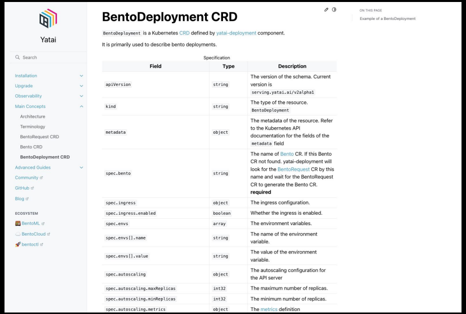


BentoML spawn more process In Container

BentoML spawn api-server process & runner process



Yatai deploy more Pod In k8s



Yatai

Search

Installation
Upgrade
Observability
Main Concepts
Architecture
Terminology
BentoRequest CRD
Bento CRD
BentoDeployment CRD
Advanced Guides
Community
GitHub
Blog

ECOSYSTEM
BentoML
BentoCloud
bentoctl

ON THIS PAGE
Example of a BentoDeployment

BentoDeployment CRD

`BentoDeployment` is a Kubernetes CRD defined by yatai-deployment component.

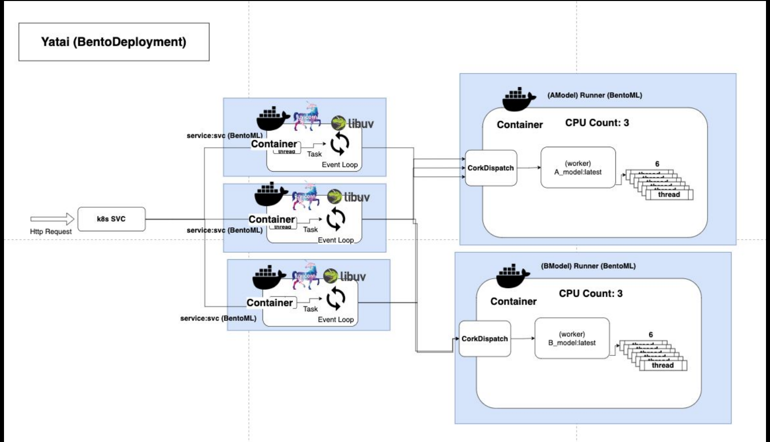
It is primarily used to describe bento deployments.

Specification

Field	Type	Description
`apiVersion`	`string`	The version of the schema. Current version is `serving.yatai.ai/v2alpha1`
`kind`	`string`	The type of the resource. `BentoDeployment`
`metadata`	`object`	The metadata of the resource. Refer to the Kubernetes API documentation for the fields of the `metadata` field
`spec.bento`	`string`	The name of Bento CR. If this Bento CR not found. yatai-deployment will look for the BentoRequest CR by this name and wait for the BentoRequest CR to generate the Bento CR. **required**
`spec.ingress`	`object`	The ingress configuration.
`spec.ingress.enabled`	`boolean`	Whether the ingress is enabled.
`spec.envs`	`array`	The environment variables.
`spec.envs[].name`	`string`	The name of the environment variable.
`spec.envs[].value`	`string`	The value of the environment variable.
`spec.autoscaling`	`object`	The autoscaling configuration for the API server
`spec.autoscaling.maxReplicas`	`int32`	The maximum number of replicas.
`spec.autoscaling.minReplicas`	`int32`	The minimum number of replicas.
`spec.autoscaling.metrics`	`object`	The metrics definition

https://docs.yatai.io/en/latest/concepts/bentodeployment_crd.html

Serving Platform Yatai BentoDeployment



Use cases 1 (Offline serving)
이 경우 Client connection의 갯수는 한정적
APIServer Pod수를 **줄이고(scale in)**
Runner Pod의 수를 **늘려서(scale out)** 배포,운용하는 것이 효율적

Use cases 2 (Increase throughput)
API Server Pod의 수를 늘리고 (scale out)
Runner Pod 의 container당 더 많은 자원할당 (scale up) + batchable Option 활성화
(이 경우 필연적으로 Latency Trade off)

Use cases 3 (Improve Latency)
API Server Pod의 수 유지
Runner Pod 의 container당 더 많은 자원할당 (scale up) + batchable Option 비활성화
(WAS 성능이 좋아봤자 Database가 느리면 결국 느린것과 같은 논리
Runner(inference연산속도)가 빨라야지 Latency가 개선될수있다)

위 예시들은 BentoCRD yaml 파일의 숫자 값 수정만으로 처리가 가능하다

Use cases 4 (Improve Latency)
코드 수정을 통해 동일 모델 Runner를 2개이상 생성해서
Runner에게 요청받은 데이터를 분산해서 연산하도록 한다. (아래 주제 참고)

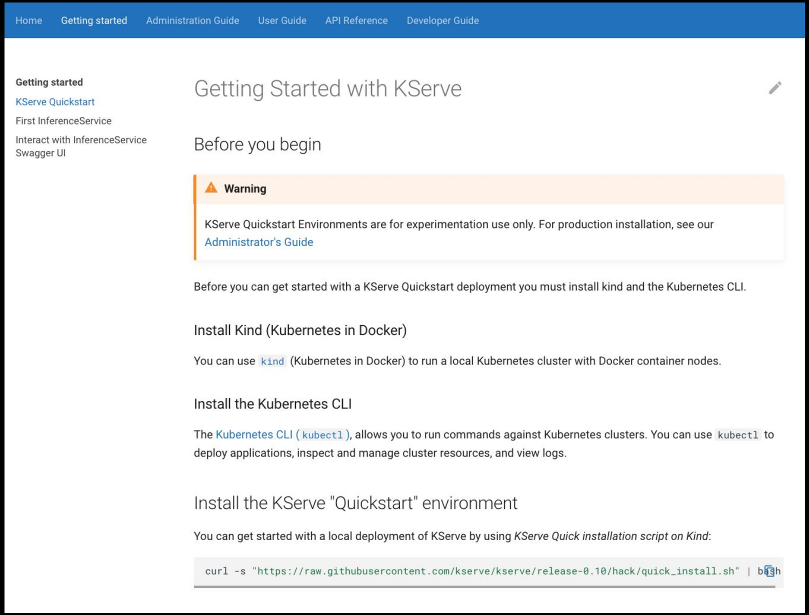
BentoML UseCase In NAVER : Online Serving (distributed Runner)

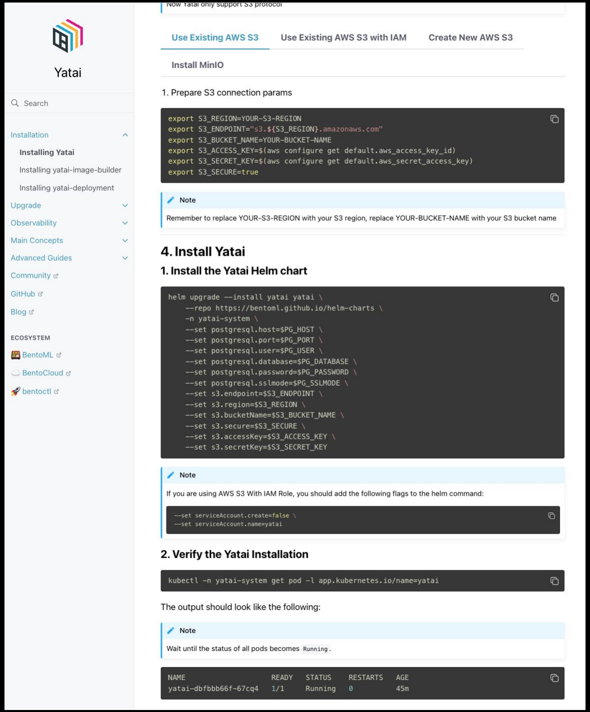
BentoML에서 distributed runner 형태로 코드를 작성했다면 이 경우에는 Latency는 개선됨
하지만 처리량 증진을 위해 bentoDeployment.yaml 작성시 Runner pod 의 갯수를 늘리거나 cpu 할당량을 늘릴것을 권장

https://docs.yatai.io/en/latest/concepts/bentodeployment_crd.html

Serving Platform quick Start

You can see a demo of both Kserve and Yatai on minikube.
Enter each official site and execute the QuickStart Script





https://kserve.github.io/website/0.10/get_started/#install-the-kserve-quickstart-environment

https://docs.yatai.io/en/latest/installation/yatai.html

마치며

Model Serving Frameworks are mostly similar in architecture. The only difference is how they are used and for what purpose.

BentoML is a unified framework, so it is compatible with most MLFrameworks. This is one of the biggest advantages of BentoML

It's more efficient to simplify model serving with BentoML and use that time to improve the performance of the model itself, rather than jumping from BentoML to another model framework to improve performance.
(this is why our team use BentoML)

If you need high performance, triton-inference-server is a very good choice
For this reason, bentoml only supports triton-inference-server as a runner.

Model Serving Platforms are similar in usage: Kserve, yatai (because they are based on k8s).

The case of SageMaker Deploy is a little different because it is a cloud-based resource unit. In this case, you can use bentoml inside sagemaker with bentoctl.

BentoML and yatai are the best combination, but that doesn't mean that yatai is mandatory (=yatai, kserve), which can be chosen again when the organization moves up to platform).
If you have a clear understanding of the Serving Platform concept in the first place, you are free to switch.

BentoML and Yatai are both good frameworks, except the naming sense (in this case only In Korea (IMO))

pdf download link
https://github.com/KimSoungRyoul/PyConKR2023-ModelServing-BentoML

끝

https://github.com/KimSoungRyoul/PyConKR2023-ModelServing