use log::error;

use prometheus::{

CounterVec, HistogramOpts, HistogramVec, IntCounter, IntCounterVec, IntGauge, IntGaugeVec,

Opts, Registry,

};

use warp::{Rejection, Reply};

use crate::{NAME, VERSION};

lazy\_static! {

pub static ref REGISTRY: Registry = Registry::new();

pub static ref NUM\_REQUESTS\_RECEIVED: IntCounter =

IntCounter::new(":navi:num\_requests", "Number of Requests Received")

.expect("metric can be created");

pub static ref NUM\_REQUESTS\_FAILED: IntCounter = IntCounter::new(

":navi:num\_requests\_failed",

"Number of Request Inference Failed"

)

.expect("metric can be created");

pub static ref NUM\_REQUESTS\_DROPPED: IntCounter = IntCounter::new(

":navi:num\_requests\_dropped",

"Number of Oneshot Receivers Dropped"

)

.expect("metric can be created");

pub static ref NUM\_BATCHES\_DROPPED: IntCounter = IntCounter::new(

":navi:num\_batches\_dropped",

"Number of Batches Proactively Dropped"

)

.expect("metric can be created");

pub static ref NUM\_BATCH\_PREDICTION: IntCounter =

IntCounter::new(":navi:num\_batch\_prediction", "Number of batch prediction")

.expect("metric can be created");

pub static ref BATCH\_SIZE: IntGauge =

IntGauge::new(":navi:batch\_size", "Size of current batch").expect("metric can be created");

pub static ref NAVI\_VERSION: IntGauge =

IntGauge::new(":navi:navi\_version", "navi's current version")

.expect("metric can be created");

pub static ref RESPONSE\_TIME\_COLLECTOR: HistogramVec = HistogramVec::new(

HistogramOpts::new(":navi:response\_time", "Response Time in ms").buckets(Vec::from(&[

0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0, 110.0, 120.0, 130.0,

140.0, 150.0, 160.0, 170.0, 180.0, 190.0, 200.0, 250.0, 300.0, 500.0, 1000.0

]

as &'static [f64])),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_PREDICTIONS: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_predictions",

"Number of predictions made by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref PREDICTION\_SCORE\_SUM: CounterVec = CounterVec::new(

Opts::new(

":navi:prediction\_score\_sum",

"Sum of prediction score made by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NEW\_MODEL\_SNAPSHOT: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:new\_model\_snapshot",

"Load a new version of model snapshot"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref MODEL\_SNAPSHOT\_VERSION: IntGaugeVec = IntGaugeVec::new(

Opts::new(

":navi:model\_snapshot\_version",

"Record model snapshot version"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_REQUESTS\_RECEIVED\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_requests\_by\_model",

"Number of Requests Received by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_REQUESTS\_FAILED\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_requests\_failed\_by\_model",

"Number of Request Inference Failed by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_REQUESTS\_DROPPED\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_requests\_dropped\_by\_model",

"Number of Oneshot Receivers Dropped by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_BATCHES\_DROPPED\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_batches\_dropped\_by\_model",

"Number of Batches Proactively Dropped by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref INFERENCE\_FAILED\_REQUESTS\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:inference\_failed\_requests\_by\_model",

"Number of failed inference requests by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_PREDICTION\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_prediction\_by\_model",

"Number of prediction by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref NUM\_BATCH\_PREDICTION\_BY\_MODEL: IntCounterVec = IntCounterVec::new(

Opts::new(

":navi:num\_batch\_prediction\_by\_model",

"Number of batch prediction by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref BATCH\_SIZE\_BY\_MODEL: IntGaugeVec = IntGaugeVec::new(

Opts::new(

":navi:batch\_size\_by\_model",

"Size of current batch by model"

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref CUSTOMOP\_VERSION: IntGauge =

IntGauge::new(":navi:customop\_version", "The hashed Custom OP Version")

.expect("metric can be created");

pub static ref MPSC\_CHANNEL\_SIZE: IntGauge =

IntGauge::new(":navi:mpsc\_channel\_size", "The mpsc channel's request size")

.expect("metric can be created");

pub static ref BLOCKING\_REQUEST\_NUM: IntGauge = IntGauge::new(

":navi:blocking\_request\_num",

"The (batch) request waiting or being executed"

)

.expect("metric can be created");

pub static ref MODEL\_INFERENCE\_TIME\_COLLECTOR: HistogramVec = HistogramVec::new(

HistogramOpts::new(":navi:model\_inference\_time", "Model inference time in ms").buckets(

Vec::from(&[

0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0, 50.0, 55.0, 60.0, 65.0,

70.0, 75.0, 80.0, 85.0, 90.0, 100.0, 110.0, 120.0, 130.0, 140.0, 150.0, 160.0,

170.0, 180.0, 190.0, 200.0, 250.0, 300.0, 500.0, 1000.0

] as &'static [f64])

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref CONVERTER\_TIME\_COLLECTOR: HistogramVec = HistogramVec::new(

HistogramOpts::new(":navi:converter\_time", "converter time in microseconds").buckets(

Vec::from(&[

0.0, 500.0, 1000.0, 1500.0, 2000.0, 2500.0, 3000.0, 3500.0, 4000.0, 4500.0, 5000.0,

5500.0, 6000.0, 6500.0, 7000.0, 20000.0

] as &'static [f64])

),

&["model\_name"]

)

.expect("metric can be created");

pub static ref CERT\_EXPIRY\_EPOCH: IntGauge =

IntGauge::new(":navi:cert\_expiry\_epoch", "Timestamp when the current cert expires")

.expect("metric can be created");

}

pub fn register\_custom\_metrics() {

REGISTRY

.register(Box::new(NUM\_REQUESTS\_RECEIVED.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_REQUESTS\_FAILED.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_REQUESTS\_DROPPED.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(RESPONSE\_TIME\_COLLECTOR.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NAVI\_VERSION.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(BATCH\_SIZE.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_BATCH\_PREDICTION.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_BATCHES\_DROPPED.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_PREDICTIONS.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(PREDICTION\_SCORE\_SUM.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NEW\_MODEL\_SNAPSHOT.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(MODEL\_SNAPSHOT\_VERSION.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_REQUESTS\_RECEIVED\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_REQUESTS\_FAILED\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_REQUESTS\_DROPPED\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_BATCHES\_DROPPED\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(INFERENCE\_FAILED\_REQUESTS\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_PREDICTION\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(NUM\_BATCH\_PREDICTION\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(BATCH\_SIZE\_BY\_MODEL.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(CUSTOMOP\_VERSION.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(MPSC\_CHANNEL\_SIZE.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(BLOCKING\_REQUEST\_NUM.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(MODEL\_INFERENCE\_TIME\_COLLECTOR.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(CONVERTER\_TIME\_COLLECTOR.clone()))

.expect("collector can be registered");

REGISTRY

.register(Box::new(CERT\_EXPIRY\_EPOCH.clone()))

.expect("collector can be registered");

}

pub fn register\_dynamic\_metrics(c: &HistogramVec) {

REGISTRY

.register(Box::new(c.clone()))

.expect("dynamic metric collector cannot be registered");

}

pub async fn metrics\_handler() -> Result<impl Reply, Rejection> {

use prometheus::Encoder;

let encoder = prometheus::TextEncoder::new();

let mut buffer = Vec::new();

if let Err(e) = encoder.encode(&REGISTRY.gather(), &mut buffer) {

error!("could not encode custom metrics: {}", e);

};

let mut res = match String::from\_utf8(buffer) {

Ok(v) => format!("#{}:{}\n{}", NAME, VERSION, v),

Err(e) => {

error!("custom metrics could not be from\_utf8'd: {}", e);

String::default()

}

};

buffer = Vec::new();

if let Err(e) = encoder.encode(&prometheus::gather(), &mut buffer) {

error!("could not encode prometheus metrics: {}", e);

};

let res\_custom = match String::from\_utf8(buffer) {

Ok(v) => v,

Err(e) => {

error!("prometheus metrics could not be from\_utf8'd: {}", e);

String::default()

}

};

res.push\_str(&res\_custom);

Ok(res)

}