// Copyright 2020 kubeflow.org.

//

// Licensed under the Apache License, Version 2.0 (the "License");

// you may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

syntax = "proto3";

package inference;

// Inference Server GRPC endpoints.

service GRPCInferenceService

{

// The ServerLive API indicates if the inference server is able to receive

// and respond to metadata and inference requests.

rpc ServerLive(ServerLiveRequest) returns (ServerLiveResponse) {}

// The ServerReady API indicates if the server is ready for inferencing.

rpc ServerReady(ServerReadyRequest) returns (ServerReadyResponse) {}

// The ModelReady API indicates if a specific model is ready for inferencing.

rpc ModelReady(ModelReadyRequest) returns (ModelReadyResponse) {}

// The ServerMetadata API provides information about the server. Errors are

// indicated by the google.rpc.Status returned for the request. The OK code

// indicates success and other codes indicate failure.

rpc ServerMetadata(ServerMetadataRequest) returns (ServerMetadataResponse) {}

// The per-model metadata API provides information about a model. Errors are

// indicated by the google.rpc.Status returned for the request. The OK code

// indicates success and other codes indicate failure.

rpc ModelMetadata(ModelMetadataRequest) returns (ModelMetadataResponse) {}

// The ModelInfer API performs inference using the specified model. Errors are

// indicated by the google.rpc.Status returned for the request. The OK code

// indicates success and other codes indicate failure.

rpc ModelInfer(ModelInferRequest) returns (ModelInferResponse) {}

}

message ServerLiveRequest {}

message ServerLiveResponse

{

// True if the inference server is live, false if not live.

bool live = 1;

}

message ServerReadyRequest {}

message ServerReadyResponse

{

// True if the inference server is ready, false if not ready.

bool ready = 1;

}

message ModelReadyRequest

{

// The name of the model to check for readiness.

string name = 1;

// The version of the model to check for readiness. If not given the

// server will choose a version based on the model and internal policy.

string version = 2;

}

message ModelReadyResponse

{

// True if the model is ready, false if not ready.

bool ready = 1;

}

message ServerMetadataRequest {}

message ServerMetadataResponse

{

// The server name.

string name = 1;

// The server version.

string version = 2;

// The extensions supported by the server.

repeated string extensions = 3;

}

message ModelMetadataRequest

{

// The name of the model.

string name = 1;

// The version of the model to check for readiness. If not given the

// server will choose a version based on the model and internal policy.

string version = 2;

}

message ModelMetadataResponse

{

// Metadata for a tensor.

message TensorMetadata

{

// The tensor name.

string name = 1;

// The tensor data type.

string datatype = 2;

// The tensor shape. A variable-size dimension is represented

// by a -1 value.

repeated int64 shape = 3;

}

// The model name.

string name = 1;

// The versions of the model available on the server.

repeated string versions = 2;

// The model's platform. See Platforms.

string platform = 3;

// The model's inputs.

repeated TensorMetadata inputs = 4;

// The model's outputs.

repeated TensorMetadata outputs = 5;

}

message ModelInferRequest

{

// An input tensor for an inference request.

message InferInputTensor

{

// The tensor name.

string name = 1;

// The tensor data type.

string datatype = 2;

// The tensor shape.

repeated int64 shape = 3;

// Optional inference input tensor parameters.

map<string, InferParameter> parameters = 4;

// The tensor contents using a data-type format. This field must

// not be specified if "raw" tensor contents are being used for

// the inference request.

InferTensorContents contents = 5;

}

// An output tensor requested for an inference request.

message InferRequestedOutputTensor

{

// The tensor name.

string name = 1;

// Optional requested output tensor parameters.

map<string, InferParameter> parameters = 2;

}

// The name of the model to use for inferencing.

string model\_name = 1;

// The version of the model to use for inference. If not given the

// server will choose a version based on the model and internal policy.

string model\_version = 2;

// Optional identifier for the request. If specified will be

// returned in the response.

string id = 3;

// Optional inference parameters.

map<string, InferParameter> parameters = 4;

// The input tensors for the inference.

repeated InferInputTensor inputs = 5;

// The requested output tensors for the inference. Optional, if not

// specified all outputs produced by the model will be returned.

repeated InferRequestedOutputTensor outputs = 6;

// The data contained in an input tensor can be represented in "raw"

// bytes form or in the repeated type that matches the tensor's data

// type. To use the raw representation 'raw\_input\_contents' must be

// initialized with data for each tensor in the same order as

// 'inputs'. For each tensor, the size of this content must match

// what is expected by the tensor's shape and data type. The raw

// data must be the flattened, one-dimensional, row-major order of

// the tensor elements without any stride or padding between the

// elements. Note that the FP16 and BF16 data types must be represented as

// raw content as there is no specific data type for a 16-bit float type.

//

// If this field is specified then InferInputTensor::contents must

// not be specified for any input tensor.

repeated bytes raw\_input\_contents = 7;

}

message ModelInferResponse

{

// An output tensor returned for an inference request.

message InferOutputTensor

{

// The tensor name.

string name = 1;

// The tensor data type.

string datatype = 2;

// The tensor shape.

repeated int64 shape = 3;

// Optional output tensor parameters.

map<string, InferParameter> parameters = 4;

// The tensor contents using a data-type format. This field must

// not be specified if "raw" tensor contents are being used for

// the inference response.

InferTensorContents contents = 5;

}

// The name of the model used for inference.

string model\_name = 1;

// The version of the model used for inference.

string model\_version = 2;

// The id of the inference request if one was specified.

string id = 3;

// Optional inference response parameters.

map<string, InferParameter> parameters = 4;

// The output tensors holding inference results.

repeated InferOutputTensor outputs = 5;

// The data contained in an output tensor can be represented in

// "raw" bytes form or in the repeated type that matches the

// tensor's data type. To use the raw representation 'raw\_output\_contents'

// must be initialized with data for each tensor in the same order as

// 'outputs'. For each tensor, the size of this content must match

// what is expected by the tensor's shape and data type. The raw

// data must be the flattened, one-dimensional, row-major order of

// the tensor elements without any stride or padding between the

// elements. Note that the FP16 and BF16 data types must be represented as

// raw content as there is no specific data type for a 16-bit float type.

//

// If this field is specified then InferOutputTensor::contents must

// not be specified for any output tensor.

repeated bytes raw\_output\_contents = 6;

}

// An inference parameter value. The Parameters message describes a

// “name”/”value” pair, where the “name” is the name of the parameter

// and the “value” is a boolean, integer, or string corresponding to

// the parameter.

message InferParameter

{

// The parameter value can be a string, an int64, a boolean

// or a message specific to a predefined parameter.

oneof parameter\_choice

{

// A boolean parameter value.

bool bool\_param = 1;

// An int64 parameter value.

int64 int64\_param = 2;

// A string parameter value.

string string\_param = 3;

}

}

// The data contained in a tensor represented by the repeated type

// that matches the tensor's data type. Protobuf oneof is not used

// because oneofs cannot contain repeated fields.

message InferTensorContents

{

// Representation for BOOL data type. The size must match what is

// expected by the tensor's shape. The contents must be the flattened,

// one-dimensional, row-major order of the tensor elements.

repeated bool bool\_contents = 1;

// Representation for INT8, INT16, and INT32 data types. The size

// must match what is expected by the tensor's shape. The contents

// must be the flattened, one-dimensional, row-major order of the

// tensor elements.

repeated int32 int\_contents = 2;

// Representation for INT64 data types. The size must match what

// is expected by the tensor's shape. The contents must be the

// flattened, one-dimensional, row-major order of the tensor elements.

repeated int64 int64\_contents = 3;

// Representation for UINT8, UINT16, and UINT32 data types. The size

// must match what is expected by the tensor's shape. The contents

// must be the flattened, one-dimensional, row-major order of the

// tensor elements.

repeated uint32 uint\_contents = 4;

// Representation for UINT64 data types. The size must match what

// is expected by the tensor's shape. The contents must be the

// flattened, one-dimensional, row-major order of the tensor elements.

repeated uint64 uint64\_contents = 5;

// Representation for FP32 data type. The size must match what is

// expected by the tensor's shape. The contents must be the flattened,

// one-dimensional, row-major order of the tensor elements.

repeated float fp32\_contents = 6;

// Representation for FP64 data type. The size must match what is

// expected by the tensor's shape. The contents must be the flattened,

// one-dimensional, row-major order of the tensor elements.

repeated double fp64\_contents = 7;

// Representation for BYTES data type. The size must match what is

// expected by the tensor's shape. The contents must be the flattened,

// one-dimensional, row-major order of the tensor elements.

repeated bytes bytes\_contents = 8;

}