AiSvcTest

A tool for AI Developers

building AI Services

Friday, June 05, 2020

Document History

Date	Version	Description
12/31/2019	1.0	Initial Version
02/12/2020	1.1	Added support for testing liveness and shutdown requests. Added detection of common problems. Significantly expanded the documentation.
02/18/2020	1.1	Explained the need for a docker network when TRTIS is used, so the AI service and TRTIS containers can communicate with each other.
04/08/2020	1.2	AiSvcTest is now packaged as a wheel file. Logging was simplified and made easier to read.
06/05/2020	1.3	Add options to test self-hosted services

Table of Contents

1. Introduction 1

2. Installation 2

2.1 AiSvcTest.zip 2

2.1.1 Installing the Wheel File 2

3. Starting the AI Service 3

3.1 From your Debugger 3

3.2 As a Container 4

3.3 Validating the Live Request 5

3.4 Validating the Shutdown Request 5

4. Running AiSvcTest 7

4.1 Input Directory 8

4.2 Output Directory 8

4.3 Specifying Job Configuration (Optional) 9

4.4 Calls Made to the AI Service 9

4.5 Requests Simulated by AiSvcTest 10

5. Starting TRTIS 11

5.1 Connecting Docker Containers 11

5.2 Download TRTIS 11

5.3 Creating a New Container with TRTIS and Your Model 12

5.4 Starting TRTIS 12

5.5 Stopping TRTIS 12

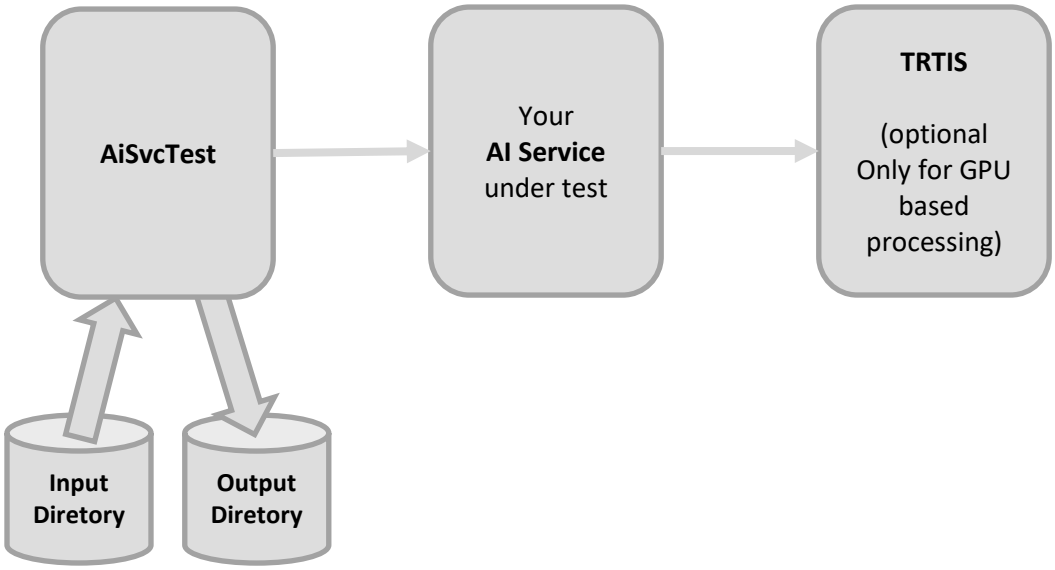
5.6 Validating TRTIS is Running 13

1. Introduction

The AiSvcTest allows AI Developers to locally test AI Services that will be deployed in the AI Marketplace cloud.

This tool, along with this documentation, are companions to the document “Cloud Hosting.pdf”. “Cloud Hosting.pdf” describes how to build the AI Service, whereas this document describes how to test it locally.

For testing there are three key components:



The AiSvcTest is a command line utility that provides the outgoing and incoming REST requests necessary to process a single study from the input directory and send the results to the output directory.

During development, you may run your AI Service in your favorite debugger. Prior to sending to Nuance, you should test it as a running docker container.

The NVIDIA TensorRT Inference Server (TRTIS) is only used if you are doing GPU-based processing. If not, you may ignore it.

The intent of AiSvcTest is to enable developers to get their AI services running prior to submitting them for cloud execution. It does not test the AI service. It allows the AI Developer to test the interoperability of the AI Service to work with Nuance AI Market Place. You must continue to exercise good software engineering and test your code.

Starting with version 1.1, some limited checks have been added to AiSvcTest to detect common errors that we see AI Developers making. This should help your testing but is not complete testing by itself. Don’t be surprised if we find issues after you submit it for cloud hosting.

2. Installation

AiSvcTest is a python application. While it is not necessary to know Python, many of our AI Developer do, and may wish to examine or modify the source code for their own purposes. Python also gives us an OS neutral environment to accommodate a broader set of AI Developers.

- 64-bit Python 3.7 or later is required. If you don't already have it installed, you can go to the official python website and download it (https://www.python.org/downloads/).
- pip3 is required for installing the dependencies

2.1 AiSvcTest.zip

The AI Service Tester is distributed as a zip file. It contains two files: a pdf containing this documentation, and a Python wheel file. You will need to unzip it into a new directory.

2.1.1 Installing the Wheel File

The wheel file is easy to install. Just execute the following command:

```
pip3 install --upgrade {the path to wheel file}
```

Then the AiSvcTest will be installed into your python library directory.

3. Starting the AI Service

In our cloud environment, your AI Services will be passed information in environment variables. When testing, it is your responsibility to ensure these variables are set.

Some debuggers allow you to set environment variable values for your application. For others, you may need to set them before you start the debugger. When in doubt, you can print the value of these variable from your service to see if they are received correctly.

Testing the container for you AI Service requires that you set the environment variables on the docker run command line. Do NOT put these variables in your Dockerfile.

Environment Variable	Description
requestPort	The port that your service should be listening on for requests. Requests to your ai service will arrive at http://localhost:{requestPort }.
tempDir	A directory where temporary files should be placed. Remember to delete all files in this directory each time you start a job. {generic directory used for testing}
maxThreads	CPUs available to containers may be less than those available in the VM. Set this environment variable to the actual CPU resource limit to allow for overhead. Partners who make use of the value will run more efficiently.
subscriptionKey	A string value that identifies and validates the service. It is required for each service call to AI Marketplace. An environment variable contains this value when the adapter is deployed – and must match the corresponding value present in the AI Tester Runtime configuration file. *Unless explicitly changed, this value should be* "AiSvcTestKey".
trtisGrpc	*(Optional)* The URL for accessing the TensorRT Inference Server (TRTIS) via gRPC.
trtisHttp	*(Optional)* The URL for accessing the TensorRT Inference Server (TRTIS) via HTTP.

Best Practices:

- Print environment variables from your service so you can validate they were received correctly.
- Do NOT hard code these variables in your Dockerfile

Common Problem:

- subscriptionKey must match the value present in the file: *{project folder} /conf/aisvctest.ini. Unless explicitly changed, this value should be* "AiSvcTestKey".

You will always start your AI Service before starting AiSvcTest. After your service initializes, it should continue running, waiting to service incoming requests.

3.1 From your Debugger

Debuggers are all unique. Simply configure the necessary environment variable and have it start your application.

You may want to set breakpoints in the code where you receive the following requests:

- /aiservice/1/live
- /aiservice/1/shutdown
- /aiservice/1/jobs

3.2 As a Container

If you are not already running docker on your local machine, you may want to install the Docker Desktop from https://www.docker.com/products/docker-desktop

Common Problems:

- Be sure you configure docker with enough memory to be able to support your AI service (and TRTIS if applicable). These memory needs can be high for an average sized desktop computer.

The AI Service container should already be installed on the local machine. If it is not, you can install it from its exported docker file as follows:

```
docker load -i {archive_file_name}.tar
```

At the tail end of loading the container image, docker will display the name of the image and its tag. Be sure you remember the name of the image and its tag value. To get a list of what is already installed on your computer with the following command:

```
docker image ls
```

You may create a docker volume to be the temporary directory when you run the container. A volume is simple to create:

```
docker volume create tempDir
```

Finally, to start the container execute the following command (as a single line):

```
docker run --name aiservice --rm -p5000:5000
-e requestPort=5000 -e subscriptionKey="AiSvcTestKey"
--mount source=tempDir,target=/tempdir -e tempDir={temp} {image}:{tag}
```

The name ‘aiservice’ is simply a name you give this specific instance of docker. This allows you to later identify this instance or stop it.

In this example, we use port 5000 as the port your container should listening on for requests. The environment variable requestPort is set with the -e flag. Your container should use this variable to set its listening port. The -p flag tells docker to expose port 5000 from the container as port 5000 on the local machine.

The environment variable subscriptionKey is your security code for making calls back to AI Marketplace (or AiSvcTest in this case). For AiSvcTest set the value to “AiSvcTestKey”. In production it will be a different value.

For testing, temp should point to an empty directory within your container. For production, please be sure to place temporary files according to the temp environment variable, as it will be a specific directory required for maximum for performance.

If you are using a GPU you will need to set the trtisGrpc and trtisHttp environment variables as well.

Finally, {image}:{tag} defines the image that you want docker to run – your AI service.

Common Problems:

- Only one process may claim a specific port. If you attempt to start a service while it is claimed by another process it will fail to start.
- Every docker container requires a unique name. If you already have one running with the same name you are attempting to specify, you'll get an error.

To see the list of all docker instances use the following command:

```
docker ps -a
```

You may stop a running docker instance with the following command:

```
docker stop {name}
```

3.3 Validating the Live Request

You can validate that your service is running and responding to liveness checks with a simple curl command.

```
curl -v http://localhost:5000/aiservice/1/live
```

A successful curl will return 200, such as:

```
C:\>curl -v -X POST http://localhost:5000/aiservice/1/shutdown
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 5000 (#0)
> POST /aiservice/1/shutdown HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 200
< Content-Length: 0
< Date: Sat, 08 Feb 2020 23:03:50 GMT
<
* Connection #0 to host localhost left intact

C:\>
```

Note: this call is not made when the tester is running in self-hosted mode.

3.4 Validating the Shutdown Request

First make sure your service is running.

```
C:\>docker ps -a
CONTAINER ID  IMAGE         COMMAND     CREATED     STATUS         PORTS                    NAMES
B3074a9b5355  xxxxx:1.0.3   "xxxxxxxx"  9 sec ago   Up 8 seconds   0.0.0.0:5000->5000/tcp   aiservice
```

Next, curl a shutdown request

```
C:\>curl -v -X POST http://localhost:5000/aiservice/1/shutdown
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 5000 (#0)
> POST /aiservice/1/shutdown HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.55.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 23
< Server: Werkzeug/0.16.0 Python/3.5.9
< Date: Sun, 09 Feb 2020 02:43:47 GMT
<
Server shutting down...* Closing connection 0
```

Check container status again looking for “Exited”.

```
C:\>docker ps -a
CONTAINER ID  IMAGE        COMMAND     CREATED     STATUS                    PORTS   NAMES
B3074a9b5355  xxxxx:1.0.3  "xxxxxxxx"  35 sec ago  Exited (247) 3 sec ago            aiservice
```

Note: this call is not made when the tester is running in self-hosted mode.

Finally, stopped docker instances stay around until you delete them (unless they are started with the –rm option). You must delete them before you can create others with the same name.

```
C:\>docker rm aiservice
```

4. Running AiSvcTest

Note:
- If your service uses a GPU with TRTIS, then TRTIS needs to be started before the AI Service. Please see the section on "Running TRTIS".
- Always ensure the AI service is running before you start the AiSvcTest tool.
- If you are using Python virtual environment, be sure to activate the same environment where you had installed the requirements (dependent modules).

Running the AiSvcTest will process a single study from the input directory and send the results to the output directory.

The AiSvcTest may be started from any directory using python3. The syntax should be

```
python -m AiSvcTest -i c:\input -o c:\output -s http://localhost:5000 [-k]
```

Notes:

- AiSvcTest is case sensitive.
- Python must be: 64bit Version 3.7 (or better). In some environments **Python3** is the command used to run the interpreter.

The options are:

-i Input directory – The input directory where all of the DICOM files for a single study have been placed.

-o Output directory – The output directory where results should be stored. A log file is kept in this directory and each test run will create a new folder named with a sequentially increasing number to hold the results your service creates.

-s Service URL – This is the URL describing the location of our AI Service (which should be running prior to test execution). Generally, this is on the same machine at more 5000: http://localhost:5000.

-k KeepAlive *(optional)* – If this option is NOT present AiSvcTest will make a /aiservice/1/shutdown request asking your service to terminate. This is how we test this request. However, during general testing this make be inconvenient, so the -k option was created and will prevent this request from being made.

--host host_address *(optional)*– This is an alternative address the service may use to connect with the testing program. Some environments (i.e. self hosted, remote docker installations) may require an external URL. The host address consists of an IP address and a port number. {IP address}:{host port}

- sh SelfHost *(optional* - This switch causes the test to be run in Self Hosted mode. It is required to test against services running on remote systems or not run through a docker environment. It is often used in conjunction with the –host option (described above). In this mode the service's /live and /shutdown end points are not called.

An example of a successful run:

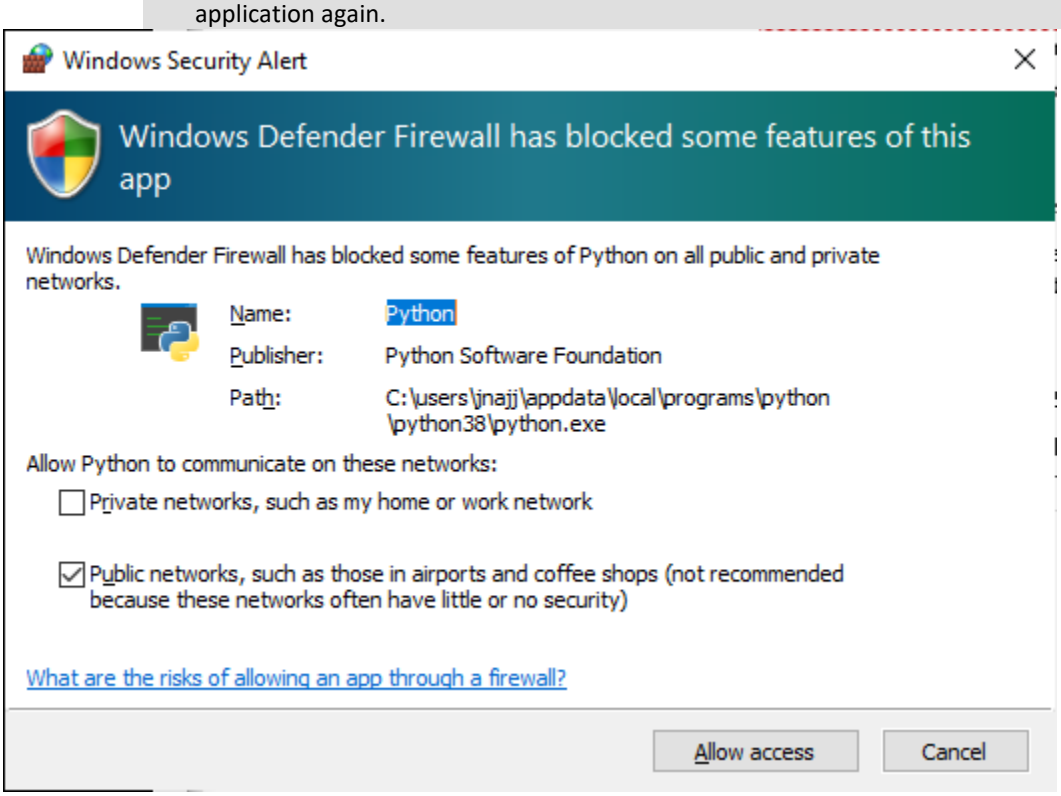
```
    _    _ ____            _____         _     _   _ 
   / \  (_) ___|_   ____|_   _|__  ___| |_  / | / |
  / _ \ | \___ \ \ / / __|| |/ _ \/ __| __| | | | |
 / ___ \| |___) \ V / (__ | |  __/\__ \ |_  | |_| |
/_/   \_\_|____/ \_/ \___||_|\___||___/\__| |_(_)_|

[AiSvcTest] [2020-02-03 17:48:46,287]  [INFO] Executing AiSvcTest . . .
```

```
[AiSvcTest] [2020-02-03 17:48:46,295]  [INFO] Dispatcher is writing to
C:\data\result\aisvctest.log
[AiSvcTest] [2020-02-03 17:48:47,421]  [INFO] Transaction ID: 49
[AiSvcTest] [2020-02-03 17:48:47,467]  [INFO] Executing Algorithm . . .
[AiSvcTest] [2020-02-03 17:48:58,355]  [INFO] ServiceTest eTime = 0:00:10.809438
[AiSvcTest] [2020-02-03 17:48:58,356]  [INFO] Algorithm executed successfully.
[AiSvcTest] [2020-02-03 17:48:58,358]  [INFO] Please check the log file for detailed
status
```

Important:

- Not all errors will be displayed on the console. You **must** check the log for errors.
- On some windows system, you may get a dialog stating that your firewalls need to be adjusted. Please allow access for Python. After you have allowed access, try running the application again.



4.1 Input Directory

The input directory is relatively simple. It should only contain:

- *.dcm – a set of DICOM files corresponding to a single study
- jobconfig.json – an optional file to set job configuration information. See the section “Specifying Job Configuration” later in this document.

4.2 Output Directory

There are two artifacts which are generated from each test.

1. The AI Service Results output – contained within a FHIR file and placed in a subdirectory named using the transaction ID number and placed under the user specified ***result*** directory. These results may also include DICOM files if these are generated by the AI Service.

2. The session log file. The AiSvcTest.log file is placed in the root of the user specified ***result*** directory. If the file exists, it will be appended to when a test is run.

4.3 Specifying Job Configuration (Optional)

AI Services may be configured with additional JSON data that will be sent in the job payload. This information is under the new member "configuration". There are three sub-members that may be set: service, subscription, and subscriber.

- service – this data is controlled exclusively by the AI Developer. You decide what goes here, and a Nuance representative will code it into the appropriate servicePlan. Therefore, this data coincides with the price you are setting and does not change per subscriber. If your service can optionally do several things that you charge different prices for, this is the place for that information.
- subscription – this data is set on each individual subscription. This is the place for a subscriber to set option values, preferences, the name of their company, or anything specific to them. The AI Developer decides what goes here, and the subscriber sets the values.
- subscriber – this configuration information is reserved. Nuance will control it for values that will be common to all subscribers. Nuance will control the definition of this section.

To set these values during testing, include a file named "jobconfig.json" in the input directory with well formed JSON content. For instance:

```
{
        "service": {
            "servicePlan":"alpha",
            "includeAdvancedFoo":"true",
        },
        "subscription": {
            "companyName":"Fabrikam",
            "showLegend":"false",
        },
        "subscriber":{
        }
}
```

Sample jobconfig.json file

4.4 Calls Made to the AI Service

The AiSvcTest will make calls to your AI Service. They will all start with the URL you specified with the -s option and be followed by:

- GET /aiservice/1/live
  A liveness check. This request should return status code OK 200 if the AI Service is healthy, and an error if it is not healthy (such as a 500).
- POST /aiservice/1/shutdown
  When the container receives this command, it should immediately shut down.

- POST /aiservice/1/jobs
  This method is called to start a job. Its JSON payload describes the job to start.

See the “Cloud Hosting.pdf” document for complete AI documentation.

4.5 Requests Simulated by AiSvcTest

AiSvcTest will simulate the services available in the cloud environment. You service should use the transactionUrl parameter passed in the job payload as the URL stem, and may concatenate the following for services:

- POST /1/transactions/{transactionId}/results
  Create a result id
- POST /1/transactions/{transactionId}/results/{resultId}
  To upload a file as part of the result
- PUT /1/transactions/{transactionId}
  Set the final transaction status.

Additionally, the URLs passed in the job payload for image files will be implemented by AiSvcTest. You should not dependent on the format/structure of the image file URLS, as they are subject to change. Just use them as provided in the job payload.

See the “Cloud Hosting.pdf” document for complete AI documentation.

5. Starting TRTIS

⚠ **Note:** If your AI Service is not using TRTIS for GPU-based processing, ignore this section.

As described in the "Cloud Hosting.pdf" document, the AI Marketplace cloud does not directly expose GPUs, but allows access through the TensorRT Inference Service (TRTIS). Therefore, when you are testing, you will also need to run TRTIS, which this section will briefly discuss.

5.1 Connecting Docker Containers

While you can use the "-p" option to map a port between a docker container and the local machine (localhost), it does NOT allow one container to see ports in the other container.

Containers can only see each other if you create a common docker network for them. Creating a docker network is easy:

```
docker network create mynet
```

Now, consider two docker containers created with the following commands:

```
docker run --name aiservice --net mynet -p 5000:5000 …
docker run --name trtis     --net mynet -p 8000:8000 …
```

From the local machine, you may access aiservice as http://localhost:5000. But if you try to do that from the trtis container it will not work. You'll need to access it as http://aiservice:5000 instead.

From the local machine, you may access trtis as http://localhost:8000. But if you try to do that from the aiservice container it will not work. You'll need to access it as http://trtis:8000.

Note:

- Remember to add "--net mynet" to each docker run. In the section on running your AI service this was omitted, as not everyone uses TRTIS. Don't forget to add it if you are using TRTIS.
- Remember to use the container name as the computer name when calling from within a container and use localhost when accessing from the local machine.

5.2 Download TRTIS

TRTIS is available for free at the NVIDIA GPU Cloud (NGC). There is often some confusion. NGS does not provide cloud infrastructure but supports it on all popular clouds. You may register as a user at https://ngc.nvidia.com/signup.

You'll find the documentation for TRTIS at https://ngc.nvidia.com/catalog/containers/nvidia:tensorrtserver.

You can download TRTIS to your local machine with the following command:

```
docker pull nvcr.io/nvidia/tensorrtserver:19.07-py3
```

NVIDIA releases a new version each month. The tag encodes the year and the month of the release. You will want to make sure you are testing with whatever AI Marketplace currently has installed.

5.3 Creating a New Container with TRTIS and Your Model

TRTIS requires a parameter that points to the directory containing your model.

While docker can mount a local directory into your container for this, some companies/environments have security restrictions that don't allow this. (At Nuance SMB mounts are disabled by default, and Docker Desktop for Windows requires it for mounting directories). Therefore, this document talks about how to be a new container, derived from the original, that includes your model directory.

Create a new Dockerfile as shown below.

```
FROM nvcr.io/nvidia/tensorrtserver:19.07-py3
COPY ./my-models /models
```

And run the following command to build it:

```
docker build . -t trtis-models:test
```

This simply derives a new container from TRTIS and copies a local my-models directory into a new /models directory in the resulting container for testing purposes.

5.4 Starting TRTIS

You do not need a GPU to run TRTIS. In the absence of a GPU, TRTIS will use the CPU to execute models. While it takes longer to execute, it is convenient to be able to test locally without special requirements.

Some versions of docker have drivers built-in for GPU and some don't. Docker 19.03 and later have the drivers built-in. Docker compose does not.

NVIDIA has the "NVIDIA Container Toolkit" to allow GPUs to run in docker when the drivers are not already present. This is beyond the scope of this document. See: https://github.com/NVIDIA/nvidia-docker.

When you run in the AI Marketplace cloud, TRTIS will use our GPUs.

To start TRTIS within docker:

```
docker run --name trtis --detach --net mynet --rm --shm-size=1g
--ulimit memlock=-1 --ulimit stack=67108864 -p8000:8000 -p8001:8001 -p8002:8002
trtis-models:test trtserver --model-store=/models
```

The complete docker run documentation can be found at https://docs.docker.com/engine/reference/run/.

Remember:

- Set the trtisGrpc and trtisHttp environment variables. These need to be exposed to your AI Service for the service to locate TRTIS.
  trtisGrpc=http://trtis:8001
  trtisHttp=http://trtis:8000

5.5 Stopping TRTIS

Stopping TRTIS is easy:

```
docker stop trtis
```

5.6 Validating TRTIS is Running

The following command will return the status of models loaded into TRTIS. Models don't load instantly, so you may need to give TRTIS a minute or two to initialize.

```
curl http://localhost:8000/api/status?format=json
```

You should get output similar to the following. Look to see that your model is "MODEL_READY" and the server is "SERVER_READY".

```
id: "inference:0"
version: "1.4.0"
uptime_ns: 74550096200
model_status {
  key: "chestXRayModel"
  value {
    config {
      name: "chestXRayModel"
      platform: "tensorflow_savedmodel"
      version_policy {
        latest {
          num_versions: 1
        }
      }
      max_batch_size: 1
      input {
        name: "input_1"
        data_type: TYPE_FP32
        dims: 224
        dims: 224
        dims: 3
      }
      output {
        name: "predictions"
        data_type: TYPE_FP32
        dims: 14
      }
      instance_group {
        name: "chestXRayModel"
        count: 1
        kind: KIND_CPU
      }
      default_model_filename: "model.savedmodel"
    }
    version_status {
      key: 1
      value {
        ready_state: MODEL_READY
      }
    }
  }
}
status_stats {
  success {
    count: 1
    total_time_ns: 261900
  }
}
ready_state: SERVER_READY
```