Install tfjs-node from source

Author: Peter Babič

Published: November 15, 2021 · 9 min read

When starting with TensorFlow library bindings for NodeJS, for instance by installing:

```
npm i @tensorflow/tfjs-node
```

And then importing it inside a node module:

```
import * as tf from "@tensorflow/tfjs-node"
```

The following error can be seen:

```
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:
 AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

The warning can be dismissed in TypeScript with:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

When using JavaScript the above won't work, but the export below works for both:

```
export TF_CPP_MIN_LOG_LEVEL=2
```

Anyway, since dismissing the error without digging deeper is not what we usually do here, let's look at how to rebuild TensorFlow with appropriate compiler flags (the proper solution).

Building TensorFlow from source

The steps are documented in the official `tfjs` repository under the anchor:

Optional: Build optimal TensorFlow from source

At first, it appears it is just a few steps, but the situation is definitely more dire. Don't worry, this guide should help.

Step 1: Clone the official TensorFlow repository

First, get the official TensorFlow repository. It is quite large by the way:

```
git clone https://github.com/tensorflow/tensorflow
cd tensorflow
```

Instructions in the next steps are all executed inside this directory, unless otherwise noted.

Step 2: Install bazel

What is bazel? One definition I have found is the following:

> Correct, reproducible, and fast builds for everyone

Well, there are many build tools and this is one among them. Let's try:

```
sudo pacman -S bazel
```

This will install the official version from the `community` repository, at the time of writing it is `4.2.0`. It installs of course, but for our purposes, it does not appear to be a correct choice, as the following errors appears when trying to build TensorFlow:

```
WARNING: current bazel installation is not a release version.
Make sure you are running at least bazel 3.7.2
```

Or, a more elaborate one:

```
ERROR: The project you're trying to build requires Bazel 3.7.2 (specified in
/home/peterbabic/throw/tensorflow/.bazelversion), but it wasn't found in
/usr/bin.

Bazel binaries for all official releases can be downloaded from here:
  https://github.com/bazelbuild/bazel/releases

Please put the downloaded Bazel binary into this location:
  /usr/bin/bazel-3.7.2-linux-x86_64
```

There are two AUR packages marked precisely with the version `3.7.2`, a bazel3 and bazel3-bin.

The former required importing GPG key manually via:

```
gpg --keyserver keys.openpgp.org --recv-keys 3D5919B448457EE0
```

The latter worked for me.

> **Caution:** always inspect contents of the AUR packages before installing!

Check the bazel version, just to be sure:

```
bazel --version
# bazel 3.7.2
```

Not sure how to get it work with the official release version at this point, though.

Step 3: Adjusting Java settings

Just installing bazel might still not be enough, especially if multiple Java versions are present on the machine. Mine had installed `jdk-openjdk` which at the time of writing was sitting at the version 17. There are multiple hints, the most obvious is this possible error:

```
Extracting Bazel installation...
FATAL: Could not find system javabase. Ensure JAVA_HOME is set, or javac is
on your PATH.
```

Where could `javac` reside? It is possible to find out:

```
pacman -F javac
```

Gets us some hints:

```
extra/bash-completion 2.11-1
    usr/share/bash-completion/completions/javac
extra/java-environment-common 3-3 [installed]
    usr/bin/javac
extra/jdk11-openjdk 11.0.10.u9-1 [installed: 11.0.13.u8-1]
    usr/lib/jvm/java-11-openjdk/bin/javac
extra/jdk7-openjdk 7.u261_2.6.22-1
    usr/lib/jvm/java-7-openjdk/bin/javac
extra/jdk8-openjdk 8.u282-1 [installed: 8.u292-1]
    usr/lib/jvm/java-8-openjdk/bin/javac
extra/jre-openjdk-headless 15.0.2.u7-1 [installed: 17.u35-1]
    usr/lib/jvm/java-15-openjdk/bin/javac
```

I was confused at thins point, as many sources suggested this wrong value:

```
export JAVA_HOME=/usr/lib/jvm/default
```

Which produced this error:

```
WARNING: Ignoring JAVA_HOME, because it must point to a JDK, not a JRE.
```

Another useful hint was there when installing bazel from the official repository:

```
Packages (4) jdk11-openjdk-11.0.13.u8-1  jre11-openjdk-11.0.13.u8-1
             jre11-openjdk-headless-11.0.13.u8-1  bazel-4.2.0-2
```

It installed `jdk11-openjdk` family as dependencies. This can be further confirmed:

```
yay -Qi bazel3 | grep -i depend
# Depends On      : java-environment=11
```

Installed bazel requires JDK11, so I opted for the following:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk
```

Bingo! It worked.

> **Note:** to adjust the working Java environment consult `archlinux-java help` .

Step 4: Configure the package

Here it get's tricky:

```
./configure
```

When pressing ENTER, the following happens:

```
You have bazel 3.7.2 installed.
Please specify the location of python. [Default is /usr/bin/python3]:
```

```
Found possible Python library paths:
  /usr/lib/python3.9/site-packages
Please input the desired Python library path to use.  Default is
[/usr/lib/python3.9/site-packages]

Do you wish to build TensorFlow with ROCm support? [y/N]:
No ROCm support will be enabled for TensorFlow.

Do you wish to build TensorFlow with CUDA support? [y/N]:
No CUDA support will be enabled for TensorFlow.

Do you wish to download a fresh release of clang? (Experimental) [y/N]:
Clang will not be downloaded.

Please specify optimization flags to use during compilation when bazel
option "--config=opt" is specified [Default is -Wno-sign-compare]:


Would you like to interactively configure ./WORKSPACE for Android builds?
[y/N]:
Not configuring the WORKSPACE for Android builds.

Preconfigured Bazel build configs. You can use any of the below by adding "-
-config=<>" to your build command. See .bazelrc for more details.
        --config=mkl            # Build with MKL support.
        --config=mkl_aarch64    # Build with oneDNN and Compute Library for
the Arm Architecture (ACL).
        --config=monolithic     # Config for mostly static monolithic build.
        --config=numa           # Build with NUMA support.
        --config=dynamic_kernels        # (Experimental) Build kernels into
separate shared objects.
        --config=v1             # Build with TensorFlow 1 API instead of TF
2 API.
Preconfigured Bazel build configs to DISABLE default on features:
        --config=nogcp          # Disable GCP support.
        --config=nonccl         # Disable NVIDIA NCCL support.
Configuration finished
```

But we need to adjust the flags to get the instructions support, remember? Without modifying anything, we would end up even worse than with what we started:

```
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:
 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

Now we have 6 possible CPU instructions not utilized instead of just two with the release grade `tfjs-node` package. The trick is to use the correct flags during the `--config=opt` question:

```
Please specify optimization flags to use during compilation when bazel
option "--config=opt" is specified [Default is -Wno-sign-compare]: -mavx -
mavx2 -mfma -msse3 -msse4.1 -msse4.2
```

The sad part here is, I have no idea how to detect the parameters beforehand. I had to compile without the flags and then recompile with all the ones that are reportedly missing. If you know how to do it reliably in one go, please let me know.

Step 5: Build the libtensorflow package

The build, no matter the flags specified is initiated like this:

```
bazel build --config=opt --config=monolithic
//tensorflow/tools/lib_package:libtensorflow
```

The build process produces a cryptic output ending with this mess (truncated):

```
DEBUG: Repository io_bazel_rules_docker instantiated at:
  /home/peterbabic/throw/tensorflow/WORKSPACE:23:14: in <toplevel>
```

```
  /home/peterbabic/throw/tensorflow/tensorflow/workspace0.bzl:108:34: in
workspace

/home/peterbabic/.cache/bazel/_bazel_peterbabic/0a4f750584c5f2d6b197cb412804
7fc4/external/bazel_toolchains/repositories/repositories.bzl:35:23: in
repositories
Repository rule git_repository defined at:

/home/peterbabic/.cache/bazel/_bazel_peterbabic/0a4f750584c5f2d6b197cb412804
7fc4/external/bazel_tools/tools/build_defs/repo/git.bzl:199:33: in
<toplevel>
INFO: Analyzed target //tensorflow/tools/lib_package:libtensorflow (0
packages loaded, 0 t
argets configured).
INFO: Found 1 target...
Target //tensorflow/tools/lib_package:libtensorflow up-to-date:
  bazel-bin/tensorflow/tools/lib_package/libtensorflow.tar.gz
INFO: Elapsed time: 10752.573s, Critical Path: 516.72s
INFO: 4050 processes: 166 internal, 3884 local.
INFO: Build completed successfully, 4050 total actions
```

Not very interesting. We can see it took just a few seconds short of full three hours. Apart from that, there are some hints about where the build files actually reside, as to my surprise it was not anywhere near the `tensorflow` repository folder. In fact, even the output did not help me too much due to the directory structure. I had to do the following:

```
fd -HI libtensorflow.tar.gz ~
#/home/peterbabic/.cache/bazel/_bazel_peterbabic/0a4f750584c5f2d6b197cb41280
47fc4/execroot/org_tensorflow/bazel-out/k8-
opt/bin/tensorflow/tools/lib_package/libtensorflow.tar.gz
```

Or in short, the file we look for is *very* deep inside the `~/.cache/bazel` directory.

Step 5: Replace tfjs-node dependencies

The last step is get the compiled dependencies into the project. Adapt the following lines as needed:

```
cp ~/long-bazel-path/libtensorflow.tar.gz
~/myproject/node_modules/@tensorflow/tfjs-node/deps
cd ~/myproject/node_modules/@tensorflow/tfjs-node/deps
tar -xf libtensorflow.tar.gz
```

Now the node project should not report the error and the TensorFlow use should be as efficient on your hardware as possible. Some users reported speed increase in the ranges from 2x to 30x. I do not have any data on this yet, but if true, it is definitely worth considering going through all this hassle. Anyway, hope this was useful to you and if not, maybe you at least learned something new. Enjoy!

Links

- https://stackoverflow.com/questions/47068709/your-cpu-supports-instructions-that-this-tensorflow-binary-was-not-compiled-to-u
- https://stackoverflow.com/questions/41293077/how-to-compile-tensorflow-with-sse4-2-and-avx-instructions?rq=1
- https://www.tensorflow.org/install/source
- https://bbs.archlinux.org/viewtopic.php?id=222751

node tensorflow arch linux