

Go 静态编译机制



执笔苦行僧 LV.3

2022年01月15日 23:12 · 阅读 6128

+ 关注

GO 静态编译机制

一、GO 的可移植性

众所周知，Go 具有良好的跨平台可移植性，Go 还提供了交叉编译的功能，运行我们在一个平台上编译出另外一个平台可执行的二进制代码。

在Go 1.7及以后版本中，我们可以通过下面命令查看Go支持OS和平台列表：

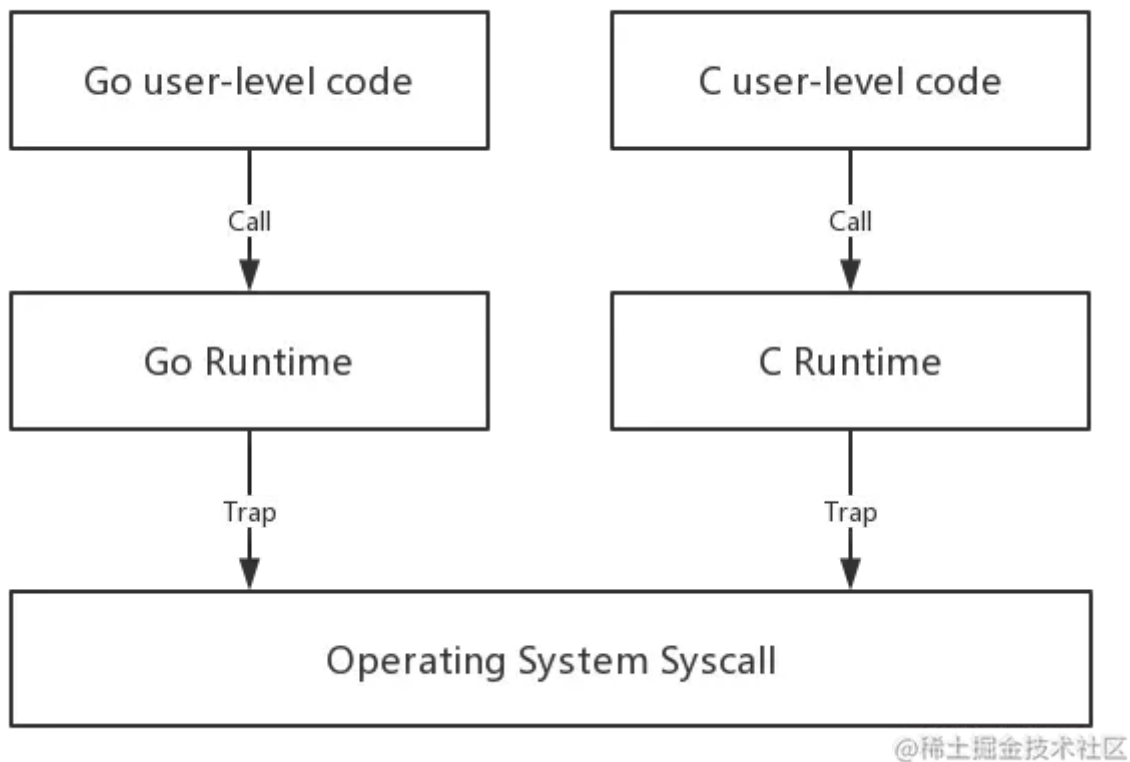
bash 复制代码

```
1 $ go tool dist list
2 aix/ppc64
3 android/386
4 android/amd64
5 android/arm
6 android/arm64
7 darwin/amd64
8 darwin/arm64
9 dragonfly/amd64
10 freebsd/386
11 freebsd/amd64
12 freebsd/arm
13 freebsd/arm64
14 illumos/amd64
15 ios/amd64
16 ios/arm64
17 js/wasm
18 linux/386
19 linux/amd64
20 linux/arm
21 linux/arm64
22 linux/mips
23 linux/mips64
24 linux/mips64le
```

```
25 linux/mipsle
26 linux/ppc64
27 linux/ppc64le
28 linux/riscv64
29 linux/s390x
30 netbsd/386
31 netbsd/amd64
32 netbsd/arm
33 netbsd/arm64
34 openbsd/386
35 openbsd/amd64
36 openbsd/arm
37 openbsd/arm64
38 openbsd/mips64
39 plan9/386
40 plan9/amd64
41 plan9/arm
42 solaris/amd64
43 windows/386
44 windows/amd64
45 windows/arm
46 windows/arm64
```

可以发现从 **linux/arm64**的嵌入式系统到 **linux/s390x** 的大型机系统，再到Windows、linux和 darwin(mac)这样的主流操作系统、amd64、386这样的主流处理器体系，Go 对各种平台和操作系统的支持确实十分广泛。

Go 的运行机制发现 Go 程序是通过 `runtime` 这个库实现与操作内核系统交互的。Go 自己实现了 `runtime`，并封装了 `syscall`，为不同平台上的 `go user level` 代码提供封装完成的、统一的go标准库。



二、GO 的静态链接

首先我们来书写两个程序，一个 C 语言的一个 Go 语言的：

arduino 复制代码

```
1 # hello.c
2 #include <stdio.h>
3
4 int main() {
5     printf("Hello, C!\n");
6     return 0;
7 }
```

编译后使用 `ldd` 命令查看其链接库：

bash 复制代码

```
1 # 1. 编译
2 gcc -o hc hello.c
3
4 # 2. 查看其依赖共享库
5 ldd hc
```

```
6
7 linux-vdso.so.1 => (0x00007fff85b6e000)
8 libc.so.6 => /lib64/libc.so.6 (0x00007fe87ff1a000)
9 /lib64/ld-linux-x86-64.so.2 (0x00007fe8802e8000)
```

我们发现这个 C 程序编译出来后的二进制文件会需要这三个库文件，因此如果我们将它做移植时会因为缺失动态库文件而造成无法运行。

接下来我们再试一下 Go 语言的：

▼ go 复制代码

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello Go!")
7 }
```

▼ bash 复制代码

```
1 # 1. 编译
2 go build -o hgo hello.go
3
4 # 2. 查看依赖
5 ldd hgo
6
7 不是动态可执行文件
```

我们发现 Go 编译出来的二进制文件是没有需要依赖的共享库的。

我们再比较一下两个文件的大小：

▼ diff 复制代码

```
1 ll -h
2
3 -rwxr-xr-x. 1 root root 8.2K 9月 15 10:36 hc
4 -rwxr-xr-x. 1 root root 1.7M 9月 15 10:42 hgo
```

我们可以发现 Go 编译出来的二进制文件比 C 语言编译出来的文件会大的多，这其实就是因为 Go 在编译时会将依赖库一起编译进二进制文件的缘故。

三、CGO 影响可移植性

虽然Go默认情况下是采取的静态编译。但是，**不是所有情况下，Go都不会依赖外部动态共享库！**

我们来看看下面这段代码：

▼ go 复制代码

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 func main() {
9     http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
10         fmt.Fprintf(w, "Hello, you've requested: %s\n", r.URL.Path)
11     })
12
13     http.ListenAndServe(":80", nil)
14 }
```

▼ bash 复制代码

```
1 go build -o server server.go
2 ldd server
3
4 linux-vdso.so.1 => (0x00007ffd96be5000)
5 libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f577ade9000)
6 libc.so.6 => /lib64/libc.so.6 (0x00007f577aa1b000)
7 /lib64/ld-linux-x86-64.so.2 (0x00007f577b005000)
```

编译后我们发现，默认采用“静态链接”的 Go 程序怎么也要依赖外部的动态链接库了呢？问题在于 **Cgo**。

自C语言出现，已经累积了无数功能强大、性能卓越的C语言代码库，可以说难以替代；为了方便快捷的使用这些C语言库，Go语言提供 Cgo，可以用以在 Go 语言中调用 C语言库。

默认情况下 Go 语言是使用 Cgo 对 Go 程序进行构建的。当使用 Cgo 进行构建时，如果我们使用的包里用着使用 C 语言支持的代码，那么最终编译的可执行文件都是要有外部依赖的。

正因为这样才会导致我们编译出来的 Go 程序会有着一些外部依赖。

四、纯静态编译

为了使我们编译的程序有着更好的可移植性，我们需要进行纯静态编译。有两种方式可以帮助我们进行纯静态编译：

1. CGO_ENABLED=0

默认情况下，`CGO_ENABLED=1`，代表着启用 Cgo 进行编译。我们如果将 `CGO_ENABLED` 置为 0，那么就可以关闭 Cgo：

```
1  CGO_ENABLED=0 go build -o server server.go
2  ldd server
3
4  不是动态可执行文件
```

2. 采用external linker

cmd/link 有两种工作模式：`internal linking` 和 `external linking`。

- `internal linking`：若用户代码中仅仅使用了 net、os/user 等几个标准库中的依赖 cgo 的包时，cmd/link 默认使用 `internal linking`，而无需启动外部external linker(如:gcc、clang等)；
- `external linking`：将所有生成的.o都打到一个 .o 文件中，再将其交给外部的链接器，比如 gcc 或 clang 去做最终链接处理。

如果我们在写入参数 `-ldflags '-linkmode "external" -extldflags "-static"'`，那么 gcc/clang 将会去做静态链接，将 .o 中 `undefined` 的符号都替换为真正的代码，从而达到纯静态编译的目的：

▼

go 复制代码

```
1 go build -o server -ldflags '-linkmode "external" -extldflags "-static"' server.go
2 ldd
3
4 不是动态可执行文件
```

如果在编译时出现以下错误：

bash 复制代码

```
1 # ahhub.io/dbox/cmd/app
2 /usr/local/go/pkg/tool/linux_amd64/link: running gcc failed: exit status 1
3 /usr/bin/ld: 找不到 -lpthread
4 /usr/bin/ld: 找不到 -lc
5 collect2: 错误: ld 返回 1
```

这是因为缺失依赖，下载依赖即可：

arduino 复制代码

```
1 yum install glibc-static.x86_64 -y
```

参考链接：

[CGO_ENABLED环境变量对Go静态编译机制的影响 – 碎言碎语 \(johng.cn\)](#)

Docker Alpine executable binary not found even if in PATH

Asked 2 years, 1 month ago Modified 10 months ago Viewed 61k times

I have an alpine running container which contains some binaries in `usr/local/bin`

59 When I `ls` the content of `usr/local/bin` I got this output :

```
/usr/local/bin # ls
dwg2SVG      dwg2dxf      dwgadd       dwgbmp       dwgfilter    dwggrep      dwglayers
dwgread      dwgrewrite   dwgwrite     dxf2dwg      dxfwrite
```

Which is what I expected. However if I execute one of these binaries by calling it I got a `not found` error from the shell :

```
/usr/local/bin # dwg2dxf
sh: dwgread: not found
/usr/local/bin # ./dwg2dxf
sh: ./dwgread: not found
```

I tested my `$PATH` which seems to be correct :

```
/usr/local/bin # echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

How can I make those binaries callable or "foundable" ? Did I miss something in my Dockerfile build ? I suppose that there is something with the `ldconfig` command in alpine that went wrong but I'm not sure.

EDIT

As suggested in one answer here I executed the `file` command and here is the output :

```
/usr/local/bin # file dwg2dxf
dwgread: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=7835d4a42651a5fb7bdfa2bd8a76e40096bacb07, with debug_info, not
stripped
```

Thoses binaries are from the [LibreDWG Official repository](#) as well as the first part of my Dockerfile. Here is the complete Dockerfile :

```
# podman/docker build -t libredwg .
#####
# STEP 1 build package from latest tar.xz
#####

FROM python:3.7.7-buster AS extracting
# libxml2-dev is broken so we need to compile it by our own
ARG LIBXML2VER=2.9.9
```



```

RUN apt-get update && \
    apt-get install -y --no-install-recommends autoconf libtool swig texinfo \
        build-essential gcc libxml2 python3-libxml2 libpcre2-dev libpcre2-32-
0 curl \
    libperl-dev libxml2-dev && \
    mkdir libxmlInstall && cd libxmlInstall && \
    wget ftp://xmlsoft.org/libxml2/libxml2-$LIBXML2VER.tar.gz && \
    tar xf libxml2-$LIBXML2VER.tar.gz && \
    cd libxml2-$LIBXML2VER/ && \
    ./configure && \
    make && \
    make install && \
    cd /libxmlInstall && \
    rm -rf gg libxml2-$LIBXML2VER.tar.gz libxml2-$LIBXML2VER
WORKDIR /app
RUN tarxz='curl --silent 'https://ftp.gnu.org/gnu/libredwg/?C=M;O=D' | grep
'.tar.xz<' | \
    head -n1|sed -E 's/.*href="([^\"]+)"\.\*/\1/' && \
    echo "latest release $tarxz"; \
    curl --silent --output "$tarxz" https://ftp.gnu.org/gnu/libredwg/$tarxz && \
    mkdir libredwg && \
    tar -C libredwg --xz --strip-components 1 -xf "$tarxz" && \
    rm "$tarxz" && \
    cd libredwg && \
    ./configure --disable-bindings --enable-release && \
    make -j `nproc` && \
    mkdir install && \
    make install DESTDIR="$PWD/install" && \
    make check DOCKER=1 DESTDIR="$PWD/install"

#####
# STEP 2 install into stable-slim
#####

# pull official base image
FROM osgeo/gdal:alpine-normal-latest

# set work directory
WORKDIR /usr/src/app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# copy requirements file
COPY ./requirements.txt /usr/src/app/requirements.txt

# install dependencies
RUN set -eux \
    && apk add --no-cache --virtual .build-deps build-base \
        py3-pip libressl-dev libffi-dev gcc musl-dev python3-dev postgresql-dev\
    && pip3 install --upgrade pip setuptools wheel \
    && pip3 install -r /usr/src/app/requirements.txt \
    && rm -rf /root/.cache/pip

# Install libredwg binaries
COPY --from=extracting /app/libredwg/install/usr/local/bin/* /usr/local/bin/
COPY --from=extracting /app/libredwg/install/usr/local/include/*
/usr/local/include/
COPY --from=extracting /app/libredwg/install/usr/local/lib/* /usr/local/lib/
COPY --from=extracting /app/libredwg/install/usr/local/share/* /usr/local/share/
RUN ldconfig /usr/local/bin/
RUN ldconfig /usr/local/include/
RUN ldconfig /usr/local/lib/

```

```
RUN ldconfig /usr/local/share/
```

```
# copy project  
COPY . /usr/src/app/
```

[linux](#)[docker](#)[shell](#)[alpine-linux](#)[Share](#) [Improve this question](#)

edited Apr 6, 2021 at 20:17

asked Apr 6, 2021 at 5:18

[jossefaz](#)

3,142 ● 3 ● 15 ● 39

[Follow](#)

What if you try to execute it with absolute path?: `/usr/local/bin/dwg2dxf` Also, is that `$PATH` exported so that `dwg2dxf` sees it too, as the error is `dwgread: not found` like `dwg2dxf` couldn't find `dwgread` for executing. – [James Brown](#) Apr 6, 2021 at 5:50

@jossefaz : I notice two strange things in your question: First you tag it as bash, but the error message you get does not come from bash. The, you don't test the PATH in connection with the command which you executed. To be on the safe side, I would do a `(printenv PATH; ls -l /usr/local/bin/dwg2dxf; dwg2dxf)` . – [user1934428](#) Apr 6, 2021 at 7:05

@user1934428 : I updated the question tag. For the second point I'll try and edit my question with the output. But I think I am close to a workaround by using another base image in my dockerfile... – [jossefaz](#) Apr 6, 2021 at 8:42

Can you edit the question to include the Dockerfile directly, not behind a link? Where do the binaries come from? Switching to an Ubuntu base potentially addresses this kind of issue, or it's possible you're missing some shared-library dependencies that need to be installed. – [David Maze](#) Apr 6, 2021 at 10:46

@DavidMaze : thanks for your comment. I've edited my question the entire Dockerfile as well as a link to the official repository those binaries came from. Actually I did change the base image an hour ago and it worked. But it's not an ideal solution for me. But since it's working and since it could help other people, I've post this workaround as a response to my own question...If anyone find any improvement to this workaround feel free to post another answer and I could remove mine – [jossefaz](#) Apr 6, 2021 at 11:32

Sorted by:

Highest score (default)

6 Answers



138



On Alpine Linux, the `not found` error is a typical symptom of dynamic link failure. It is indeed a rather confusing error by musl's `ldd` linker.

Most of the world Linux software is linked against [glibc](#), the GNU libc library (libc provides the standard C library and POSIX API). Most Linux distributions are based on glibc. OTOH, Alpine Linux is based on the [musl](#) libc library, which is a minimal implementation and strictly POSIX compliant. Executables built on glibc distributions depend on `/lib/x86_64-linux-gnu/libc.so.6`, for example, which is not available on Alpine (unless, they are statically linked).

Except for this dependency, it's important to note that while musl attempts to maintain glibc compatibility to some extent, it is far from being fully compatible, and complex software that's

built against glibc won't work with musl-libc, so simply symlinking `/lib/ld-musl-x86_64.so.1` to the glibc path isn't likely going to work.

Generally, there are several ways for running glibc binaries on Alpine:

1. Install one the glibc compatibility packages, [libc6-compat](#) or [gcompat](#):

```
# apk add gcompat
apk add libc6-compat
```

Both packages provide a light weight glibc compatibility layer which may be suitable for running simple glibc applications. `libc6-compat` implements glibc compatibility APIs and provides symlinks to glibc shared libraries such as `libm.so`, `libpthread.so` and `libcrypt.so`. The `gcompat` package is based on Adeline Linux [gcompat project](#) and does the same but provides a single library `libgcompat.so`. Both libraries install loader stubs. Depending on the application, one of them may work while the other won't, so it's good to try both.

2. Install proper glibc on Alpine, for providing all glibc methods and functionalities. There are glibc builds available for Alpine, which should be installed in the following procedure (example):

```
# Source: https://github.com/anapsix/docker-alpine-java

ENV GLIBC_REPO=https://github.com/sgerrand/alpine-pkg-glibc
ENV GLIBC_VERSION=2.30-r0

RUN set -ex && \
    apk --update add libstdc++ curl ca-certificates && \
    for pkg in glibc- $\{GLIBC\_VERSION\}$  glibc-bin- $\{GLIBC\_VERSION\}$ ; \
    do curl -sSL  $\{GLIBC\_REPO\}$ /releases/download/ $\{GLIBC\_VERSION\}$ / $\{pkg\}$ .apk \
    -o /tmp/ $\{pkg\}$ .apk; done && \
    apk add --allow-untrusted /tmp/*.apk && \
    rm -v /tmp/*.apk && \
    /usr/glibc-compat/sbin/ldconfig /lib /usr/glibc-compat/lib
```

3. Use statically linked executables. Static executables don't carry dynamic dependencies and could run on any Linux.
4. Alternatively, the software may be built from source on Alpine.

For LibreDWG, let's first verify the issue:

```
/usr/local/bin # ./dwg2dxf
/bin/sh: ./dwg2dxf: not found
/usr/local/bin
/usr/local/bin # ldd ./dwg2dxf
    /lib64/ld-linux-x86-64.so.2 (0x7fd375538000)
    libredwg.so.0 => /usr/local/lib/libredwg.so.0 (0x7fd3744db000)
    libm.so.6 => /lib64/ld-linux-x86-64.so.2 (0x7fd375538000)
    libc.so.6 => /lib64/ld-linux-x86-64.so.2 (0x7fd375538000)
Error relocating /usr/local/lib/libredwg.so.0: __strcat_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __snprintf_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __memcpy_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __stpcpy_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __strcpy_chk: symbol not found
```

```
Error relocating /usr/local/lib/libredwg.so.0: __printf_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __fprintf_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __strncat_chk: symbol not found
Error relocating /usr/local/lib/libredwg.so.0: __sprintf_chk: symbol not found
Error relocating ./dwg2dxf: __snprintf_chk: symbol not found
Error relocating ./dwg2dxf: __printf_chk: symbol not found
Error relocating ./dwg2dxf: __fprintf_chk: symbol not found
```

You can see that `dwg2dxf` depends on several glibc symbols. Now, let's follow option 2 for installing glibc:

```
/usr/src/app # cd /usr/local/bin
/usr/local/bin # ls
dwg2SVG      dwg2dxf      dwgadd       dwgbmp       dwgfilter    dwggrep      dwglayers
dwgread      dwgrewrite   dwgwrite     dxf2dwg      dxfwrite
/usr/local/bin # ./dwg2dxf
/bin/sh: ./dwg2dxf: not found
/usr/local/bin # export GLIBC_REPO=https://github.com/sgerrand/alpine-pkg-glibc
&& \
> export GLIBC_VERSION=2.30-r0 && \
> apk --update add libstdc++ curl ca-certificates && \
> for pkg in glibc-\${GLIBC\_VERSION} glibc-bin-\${GLIBC\_VERSION}; \
> do curl -sSL \${GLIBC\_REPO}/releases/download/\${GLIBC\_VERSION}/\${pkg}.apk -o
/tmp/\${pkg}.apk; done && \
> apk add --allow-untrusted /tmp/*.apk && \
> rm -v /tmp/*.apk && \
> /usr/glibc-compat/sbin/ldconfig /lib /usr/glibc-compat/lib
fetch https://dl-cdn.alpinelinux.org/alpine/v3.13/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.13/community/x86_64/APKINDEX.tar.gz
(1/1) Installing curl (7.74.0-r1)
Executing busybox-1.32.1-r3.trigger
OK: 629 MiB in 126 packages
(1/2) Installing glibc (2.30-r0)
(2/2) Installing glibc-bin (2.30-r0)
Executing glibc-bin-2.30-r0.trigger
/usr/glibc-compat/sbin/ldconfig: /usr/local/lib/libredwg.so.0 is not a symbolic
link
/usr/glibc-compat/sbin/ldconfig: /usr/glibc-compat/lib/ld-linux-x86-64.so.2 is
not a symbolic link
OK: 640 MiB in 128 packages
removed '/tmp/glibc-2.30-r0.apk'
removed '/tmp/glibc-bin-2.30-r0.apk'
/usr/glibc-compat/sbin/ldconfig: /usr/glibc-compat/lib/ld-linux-x86-64.so.2 is
not a symbolic link

/usr/glibc-compat/sbin/ldconfig: /usr/local/lib/libredwg.so.0 is not a symbolic
link
```

Voilà:

```
/usr/local/bin # ./dwg2dxf

Usage: dwg2dxf [-v[N]] [--as rNNNN] [-m|--minimal] [-b|--binary] DWGFILES...
```

Share Improve this answer Follow

edited Jul 27, 2022 at 16:05

answered Apr 6, 2021 at 18:40



Leonel Sanches da Silva


6,882 ● 9 ● 46 ● 65



valiano

16k ● 7 ● 63 ● 79

1 So in the end, does the "not found" refer to a symbol the linker tried and failed to find? – [Charley](#) Apr 7, 2021 at 0:36

4 @Charley actually it is because the linker failed to find the entire glibc shared object, `/lib64/ld-linux-x86-64.so.2`. In the glibc installation procedure, the library is installed in `/usr/glibc-compat/lib/ld-linux-x86-64.so.2`, and then symlinked to the lib64 location. – [valiano](#) Apr 7, 2021 at 5:17 

I came across this problem after building a simple Go / Golang program. Using the following options to build the Go application helped: `CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo src/main.go`. Maybe not all of them are necessary. I found this solution here: callicoder.com/docker-golang-image-container-example – [JepZ](#) Jul 13, 2021 at 18:53

2 Thank you sir! Number 2 worked in my case :) – [graphicbeacon](#) Feb 11, 2022 at 16:43

2 The reason @JepZ comment worked to build his application (for anyone wondering) is because of `CGO_ENABLED=0` and the `-a` flag. These 2 together tell go to build the application statically; therefore it does not depend on any dynamic libraries. – [Speeddymon](#) Mar 31 at 20:57

Try `apk add gcompat` (<https://pkgs.alpinelinux.org/package/edge/community/x86/gcompat>).

28

`gcompat` provides both `/lib64/ld-linux-x86-64.so.2` and `/lib/ld-linux-x86-64.so.2` (<https://pkgs.alpinelinux.org/contents?file=ld-linux-x86-64.so.2>).



Share Improve this answer Follow



answered Dec 7, 2021 at 8:25



[rdesgroppes](#)
948 ● 11 ● 11

4 just add gcompat to the apk add list and worked for me. Thank you – [kim pastro](#) Feb 10, 2022 at 1:48
