8 Deployment

Objektif :

- **Mahasiswa dapat memahami perintah deployment pada Golang**
- **Mahasiswa dapat melakukan *build* program Golang**

8.1 Go Path

Go Path adalah daftar pohon direktori yang berisi kode sumber Go. Ini dikonsultasikan untuk menyelesaikan impor yang tidak dapat ditemukan di Go tree standar. Jalur default adalah nilai dari variabel lingkungan GOPATH, ditafsirkan sebagai daftar Path yang sesuai dengan sistem operasi (pada Unix, variabel adalah string yang dipisahkan titik dua; pada Windows, string yang dipisahkan dengan titik koma). Setiap direktori yang tercantum dalam jalur Go harus memiliki struktur yang ditentukan:

- Direktori src/ menyimpan kode sumber. Jalur di bawah 'src' menentukan jalur impor atau nama yang dapat dieksekusi.
- Direktori pkg/ menyimpan objek paket yang diinstal. Seperti pada pohon Go, setiap sistem operasi target dan pasangan arsitektur memiliki subdirektori pkg sendiri (pkg/GOOS_GOARCH).

Jika DIR adalah direktori yang terdaftar di jalur Go, paket dengan sumber di DIR/src /foo/bar dapat diimpor sebagai "foo/bar" dan formulir kompilasi diinstal ke "DIR/pkg/ GOOS_GOARCH/foo /bar. a "(atau, untuk gccgo," DIR/pkg/gccgo/foo/libbar.a").

bin/ direktori menampung perintah yang dikompilasi. Setiap perintah diberi nama untuk direktori sumbernya, tetapi hanya menggunakan elemen terakhir, bukan seluruh path. Yaitu, perintah dengan sumber di DIR/src/foo/quux diinstal ke DIR/bin/quux, bukan DIR/bin/foo/quux. foo/ dilucuti sehingga Anda dapat menambahkan DIR / bin ke PATH Anda untuk mendapatkan perintah yang diinstal. Contoh berikut adalah tata letak dari folder go path.

```
GOPATH=/home/user/gocode
/home/user/gocode/
    src/
        foo/
            bar/                (kode go pada package bar)
                x.go
            quux/               (kode go pada package main)
                y.go
    bin/
        quux                    (perintah terpasang/installed command)
    pkg/
        linux_amd64/
            foo/
                bar.a           (Objek paket yang diinstal)
```

8.2 Building Golang Package for Linux and Windows

Pengembangan aplikasi Go tak jauh dari hal-hal yang menggunakan CLI atau *Command Line Interface*. Proses kompilasi, testing, eksekusi program, semua dilakukan melalui command line. Go menyediakan command `go`.

8.3 Go Build

Command ini digunakan untuk mengkompilasi file program. Ketika eksekusi program menggunakan go run, terjadi proses kompilasi pada blok program, file hasil kompilasi akan disimpan pada folder temporary untuk selanjutnya langsung dieksekusi.

Berbeda dengan go build, command ini menghasilkan file executable pada folder yang sedang aktif. Contohnya bisa dilihat pada kode berikut.

```
PS C:\Users\Rezvan\go\src\activity> go build hello.go
PS C:\Users\Rezvan\go\src\activity> ./hello
Hello Gunadarma
PS C:\Users\Rezvan\go\src\activity>
```

Gambar 8.31 Go Build

Pada contoh diatas file yang di build menghasilkan file baru pada folder yang sama dan dapat dieksekusi. Perintah go build memungkinkan Anda membuat file yang dapat dieksekusi untuk platform target yang didukung Go, Ini berarti unit dapat menguji, merilis,

dan mendistribusikan aplikasi tanpa build yang dapat dieksekusi pada platform target yang akan gunakan.

8.4 GOOS and GOARCH

Kompilasi silang bekerja dengan menetapkan variabel lingkungan yang diperlukan yang menentukan sistem operasi dan arsitektur target. Gunakan GOOS variabel untuk sistem operasi target, dan GOARCH untuk arsitektur target. Untuk build yang dapat dieksekusi, perintah akan menggunakan ini :

```
-> env GOOS=target-OS GOARCH=target-architecture go build package-import-path
```

Perintah env menjalankan program dalam lingkungan yang dimodifikasi. Ini memungkinkan untuk menggunakan environment variabel hanya untuk eksekusi perintah saat ini. Variabel tidak disetel atau direset setelah perintah dijalankan. Tabel berikut menunjukkan kemungkinan kombinasi GOOS dan GOARCH yang dapat digunakan:

Tabel 8.1 Perintah GOOS dan GOARCH

GOOS - Target Operating System	GOARCH - Target Platform
android	arm
darwin	386
darwin	amd64
darwin	arm
darwin	arm64
dragonfly	amd64
freebsd	386
freebsd	amd64

GOOS - Target Operating System	GOARCH - Target Platform
freebsd	arm
linux	386
linux	amd64
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
linux	mips
linux	mipsle
linux	mips64
linux	mips64le
linux	s390x
nacl	386
nacl	amd64p32
nacl	arm
netbsd	386
netbsd	amd64
netbsd	arm
openbsd	386

GOOS - Target Operating System	GOARCH - Target Platform
openbsd	amd64
openbsd	arm
plan9	386
plan9	amd64
plan9	arm
solaris	amd64
windows	386
windows	amd64

Menggunakan nilai-nilai dalam tabel, kita dapat mem-build program Go untuk Windows 64-bit seperti ini:

```
env GOOS=windows GOARCH=amd64 go build main.go
```

Tidak ada output yang menunjukkan bahwa operasi berhasil. Eksekusi akan dibuat di direktori saat ini, menggunakan nama paket sebagai namanya. Untuk membuat perintah dapat dijalankan untuk Windows, nama diakhiri dengan akhiran .exe.

Dalam pembahasan kali ini cara menggunakan perintah Go untuk mendapatkan paket dari sistem kontrol, serta membangun dan mengkompilasi yang dapat dieksekusi untuk platform yang berbeda. Perintah ini juga dapat membuat skrip yang bisa gunakan untuk mengkompilasi silang (Cross-compiling) satu package untuk banyak platform. Untuk memastikan aplikasi berfungsi dengan benar.

Sumber 1

main.go

```
package main

func main() {}
```

make.sh

```
#!/bin/sh

os_archs=()

# Reference:
# https://github.com/golang/go/blob/master/src/go/build/syslist.go
for goos in android darwin dragonfly freebsd linux nacl netbsd openbsd plan9
solaris windows zos
do
    for goarch in 386 amd64 amd64p32 arm armbe arm64 arm64be ppc64 ppc64le mips
\
        mipsle mips64 mips64le mips64p32 mips64p32le ppc s390 s390x sparc
sparc64
    do
        GOOS=${goos} GOARCH=${goarch} go build -o /dev/null main.go >/dev/null
2>&1
        if [ $? -eq 0 ]
        then
            os_archs+=("${goos}/${goarch}")
        fi
    done
done

for os_arch in "${os_archs[@]}"
do
    echo ${os_arch}
done
```

Sumber 2

main.go

```
package main

const (
        hello uint = 0xfedcba9876543210
)

func main() {}
```

make.sh

```
#!/bin/bash

# Reference:
# https://github.com/golang/go/blob/master/src/go/build/syslist.go
os_archs=(
    darwin/386
    darwin/amd64
    dragonfly/amd64
    freebsd/386
    freebsd/amd64
    freebsd/arm
```

```
    linux/386
    linux/amd64
    linux/arm
    linux/arm64
    linux/ppc64
    linux/ppc64le
    linux/mips
    linux/mipsle
    linux/mips64
    linux/mips64le
    linux/s390x
    nacl/386
    nacl/amd64p32
    nacl/arm
    netbsd/386
    netbsd/amd64
    netbsd/arm
    openbsd/386
    openbsd/amd64
    openbsd/arm
    plan9/386
    plan9/amd64
    plan9/arm
    solaris/amd64
    windows/386
    windows/amd64
)

os_archs_32=()
os_archs_64=()

for os_arch in "${os_archs[@]}"
do
    goos=${os_arch%/*}
    goarch=${os_arch#*/}
    GOOS=${goos} GOARCH=${goarch} go build -o /dev/null main.go >/dev/null 2>&1
    if [ $? -eq 0 ]
    then
        os_archs_64+=(${os_arch})
    else
        os_archs_32+=(${os_arch})
    fi
done

echo "32-bit:"
for os_arch in "${os_archs_32[@]}"
do
    printf "\t%s\n" "${os_arch}"
done
echo

echo "64-bit:"
for os_arch in "${os_archs_64[@]}"
do
    printf "\t%s\n" "${os_arch}"
done
echo
```