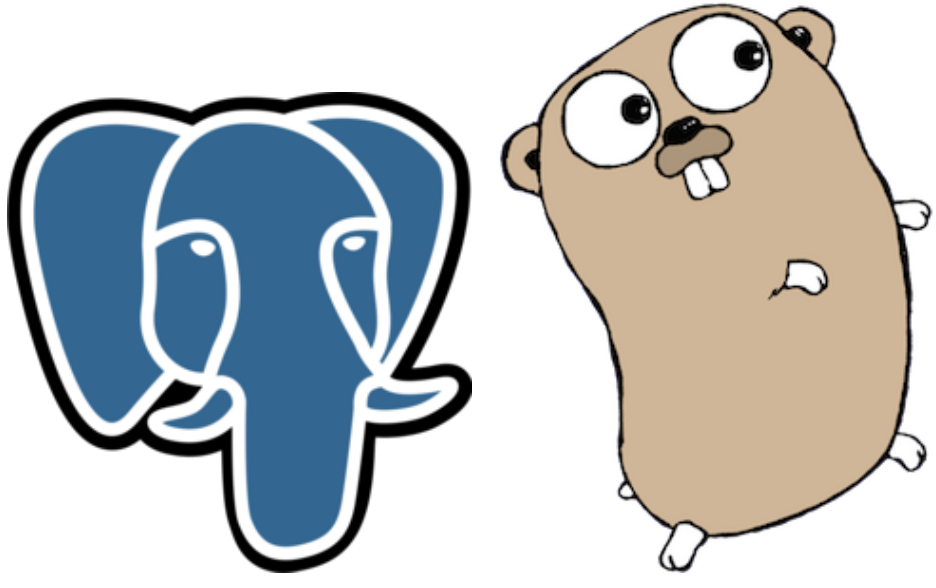
Go e PostgreSQL



Leonardo Cecchi
2ndQuadrant

2ndQuadrant PostgreSQL

ESSID: pgdayit

Utente: pgdayit

Password: pglazise2018

Di cosa parleremo oggi?

1. Installazione di Go
2. Il pacchetto `database/sql`
3. Gestione delle dipendenze con Glide
4. Il pacchetto `lib/pq`
5. Realizzazione di un server per una API RESTful
6. Test del gestore del pool di connessioni
7. Il pacchetto `pressly/goose`
8. Il pacchetto `jmoiron/sqlx`

Requisiti tecnici

- Una connessione ad Internet
- Un computer con Linux o Mac OS X o Windows (*se avete altri sistemi operativi ne parliamo insieme...*)
- Avere **Git** installato sul proprio sistema
- Avere **PostgreSQL** installato sul proprio sistema
- Avere installato un **buon** editor di testo

Installiamo Go

- **Linux** => Basta usare il package manager
- **Mac OS X** => Basta usare `brew`
- **Windows** => Va installato manualmente

Vediamo se funziona

Per provare se funziona Go, dobbiamo compilare un semplice programma

```
$ git clone http://gitlab.com/leonardoce/gopg-workshop

$ cd gopg-workshop

gopg-workshop/ $ export GOPATH=$(pwd)

gopg-workshop/ $ go install workshop_test/test_go

gopg-workshop/ $ ./bin/test_go
Ciao da me!
```

Concorrenza

```
gopg-workshop/ $ go install workshop_test/test_go_multi

gopg-workshop/ $ ./bin/test_go_multi
Numero 0
Numero 3
Numero 5
Numero 6
Numero 7
Numero 2
Numero 1
Numero 8
Numero 4
Numero 9
```

Go e concorrenza

```
func main() {
    var wg sync.WaitGroup
    var ch = make(chan int, 10)
    wg.Add(10)

    for i := 0; i < 10; i++ {
        ch <- i
    }
    for i := 0; i < 10; i++ {
        go func() {
            defer wg.Done()
            fmt.Println("Numero", <-ch)
        }()
    }
    wg.Wait()
}
```

Go: database/sql

Package sql provides a generic interface around SQL (or SQL-like) databases.

The sql package must be used in conjunction with a database driver. See https://golang.org/s/sqldrivers for a list of drivers.

Drivers that do not support context cancelation will not return until after the query is completed.

lib/pq - Driver per PostgreSQL

È il driver più utilizzato per PostgreSQL. È interamente scritto in Go, ed implementa il protocollo binario di PostgreSQL senza utilizzare il client nativo *libpq*

Gestione delle dipendenze

I software difficilmente vivono da soli. Normalmente ogni programma usa librerie di terze parti, che a loro volta usano altre librerie di terze parti, etc.

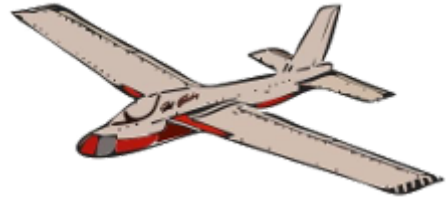
Ci sono diverse decisioni da prendere.

Cose sulle quali riflettere:

- Dichiarare le librerie usate, oppure direttamente includerle nel nostro sorgente?
- Come gestire i conflitti delle nuove versioni delle librerie?
- Dobbiamo aggiornare le librerie di terze parti, oppure *squadra che vince non si tocca?*

Usa Glide

https://github.com/Masterminds/glide



The Go community now has the dep project to manage dependencies. **Please consider trying to migrate from Glide to dep.**

If there is an issue preventing you from migrating please file an issue with dep so the problem can be corrected. Glide will continue to be supported for some time but is considered to be in a state of support rather than active feature development.

Installiamo Dep

https://github.com/golang/dep

Usa Dep

Dep è l'esperimento "ufficiale" da parte di Google per la gestione delle dipendenze

Non risolve tutti i problemi (l'ingegneria del software per alcune cose non ha trovato soluzioni perfette), ma aiuta!

Il nostro primo progetto

```
$ dep init

$ cat Gopkg.toml
```

Aggiungiamo la nostra prima dipendenza

```
$ dep ensure
```

Guardamo insieme come è cambiato il file Gopkg.toml e Gopkg.lock

Uso di PostgreSQL

Commento programma 03_connection_test

Poter creare un pool di connessioni non vuol dire potersi connettere davvero al database.

La funzione **Ping** è una amica

Il connection pooler

Commento del progamma 04_pooler_test

Secondo voi funziona sempre?

Commento del programma 05_pooler_multi_test

Ogni volta che esegui una query con **Exec** potresti aprire una connessione e chiuderla, oppure capitare su un'altra connessione.

Hai dello stato *locale* alla connessione? Le transazioni sono tue amiche.

Si creano con **Begin** e si chiudono con **Commit** oppure **Rollback**. Sorpreso?

Si possono utilizzare sulle transazioni quasi tutte le funzioni della connessione come **Exec**, **Query***

Connection pooler e concorrenza

Commento del progamma 06_bench_test

Gestione delle migrazioni con goose

```
$ go get -v -u github.com/pressly/goose/cmd/goose
[...]

$ goose -v
[...]
```

```
$ goose postgres \
  "user=postgres dbname=postgres sslmode=disable" \
  create initial sql

[...] Created new file: 00001_initial.sql
```

```sql
-- +goose Up
-- SQL in this section is executed when the migration is applied.
CREATE TABLE persone (
    id INTEGER PRIMARY KEY,
    nome VARCHAR(200),
    cognome VARCHAR(200)
);

-- +goose Down
-- SQL in this section is executed when the migration is rolled back.
DROP TABLE persone;
```

```
$ goose postgres "user=postgres dbname=postgres sslmode=disable" up
[...] OK    00001_initial.sql
```

I valori nulli

In go alcuni tipi di dati non possono essere nulli.

- string
- int
- float

```
type Scanner interface {
  // Scan assigns a value from a database driver.
  //
  // The src value will be of one of the following types:
  //
  //    int64, float64
  //    bool
  //    []byte
  //    string
  //    time.Time
  //    nil - for NULL values
  //
  // An error should be returned if the value cannot be stored
  // without loss of information.
  Scan(src interface{}) error
}
```

`guregu/null.v3` può aiutare

Commento programma 08_nulls e 09_nulls_betters

joiron/sqlx

È una libreria che mette a disposizione molte estensioni al pacchetto standard `database/sql`

Commento programma `10_nice_sqlx`

Con tutto quello che abbiamo usato fin ora, creiamo un semplice server HTTP

commento programma `11_test_http`


2ndQuadrant®
PostgreSQL