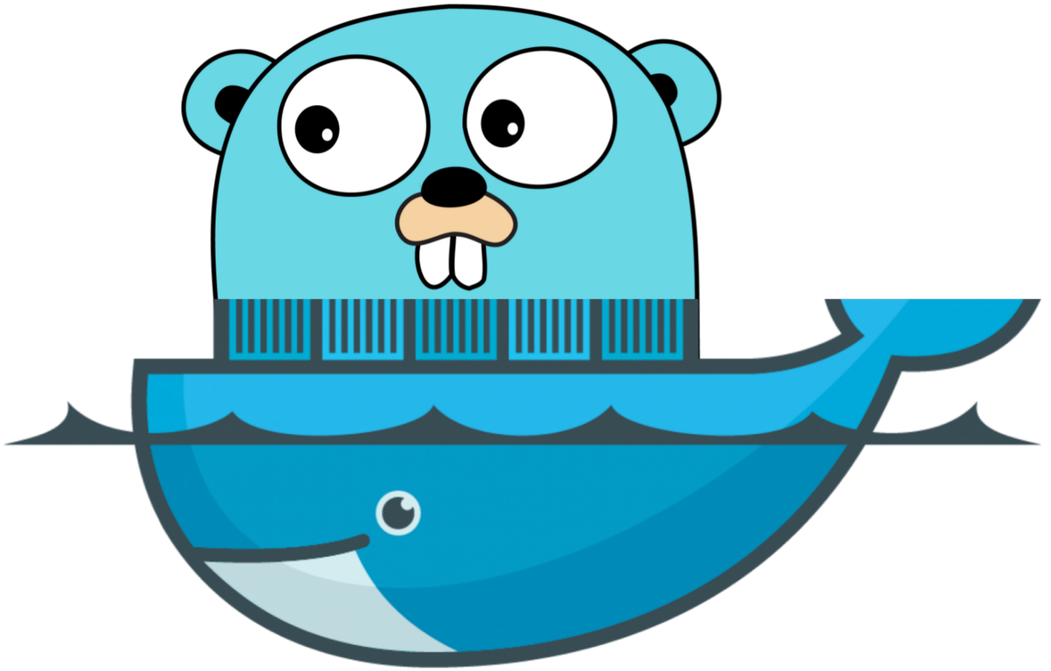
Mutex Lock
Channel Pattern



200Lab
Education

Designed by **200lab Education**
Mentor/Instructor: Viet Tran - Solution Architect 200lab

Data Racing

Trong môi trường Multi-Threading, cụ thể là concurrent với các Goroutines, nếu ta đọc và ghi vào cùng 1 biến sẽ xuất hiện lỗi "Data Racing":

```
var count int = 0

for i := 1; i <= 5; i++ {
    go func() {
        for j := 1; j <= 10000; j++ {
            count += 1
        }
    }()
}

time.Sleep(time.Second * 7)
fmt.Println("Count:", count)
```

Kết quả mong muốn: 50,000
Kết quả thực tế: luôn nhỏ hơn 50,000

Mutex Lock - Giải quyết Data Racing

Sử dụng sync.RWMutext để giải quyết bài toán trên

```
var count int = 0
lock := new(sync.RWMutex)

for i := 1; i <= 5; i++ {
    go func() {
        for j := 1; j <= 10000; j++ {
            lock.Lock()
            count += 1
            lock.Unlock()
        }
    }()
}

time.Sleep(time.Second * 7)
fmt.Println("Count:", count)
```

Kết quả mong muốn: 50,000
Kết quả thực tế: 50,000

Tác dụng của mutext lock trong trường hợp này là:
- Goroutine nào đầu tiên vào sẽ "chiếm lock", hàm **Lock()** sẽ block tất cả các goroutine đến sau cho tới khi nó **Unlock()**.
- Từ đó chỉ có 1 goroutine trong cùng một thời điểm được update giá trị cho biến count.

Mutex Lock (tt) - RWLock và RLock

- **RWLock**: block tất cả Goroutine còn lại dù đang là read hay write.
- **RLock**: block tất cả Goroutine write, cho phép các goroutine Read được phép truy xuất (shared lock).

Channel Pattern

Channel có 2 pattern thường thấy: hoặc được pass vào function hoặc nhận về từ một function Goroutine

- **https://play.golang.org/p/9C-u721el8p**
- **https://play.golang.org/p/df7WGblymJB**
- **https://play.golang.org/p/RTMRgC2ddf4**
- **https://play.golang.org/p/QMqTcX-doWy**

Thank you