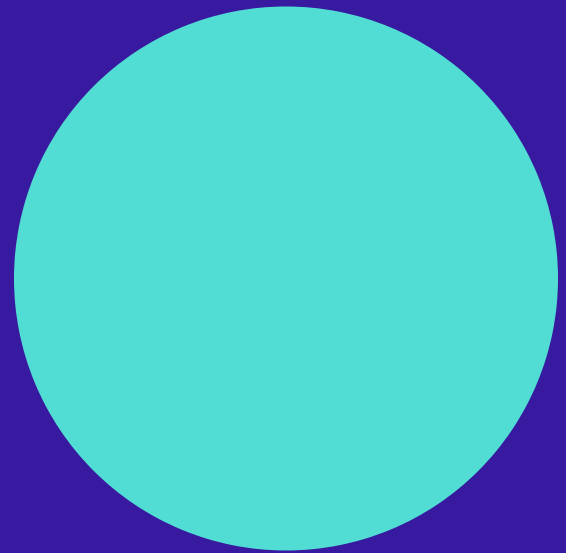


MERCARI.GO #10

GOPHERCON 2019 HOW UBER "GO"ES RECAP

@micnncim



Topics to be talked

- About me
- Uber's Go Problems
- Dependency Injection
- Standardizing Code Structure
- Switching to Monorepo
- My Gophercon Recap

Table of Contents





About me

@micnncim

Twitter: @micnncim / GitHub: @micnncim
University Student / B4 / Computer Science
Software Engineer Intern @Merpay Expert Team



@GopherCon 2019

HOW UBER "GO"ES



ELENA MOROZOVA @LELENANAM

Teal and dark blue curved shapes in the bottom right corner of the slide.

Problems with Go

NEW SERVICE

新しいマイクロサービスをフルスクラッチで作るコストの高さ

CONTEXT SWITCHING

各マイクロサービスのアーキテクチャの大きな相違

GLOBAL FEATURE

全体に関わる機能の実装の困難さ



Solutions



DEPENDENCY INJECTION

uber-go/fx の導入



CONSISTENT CODE STRUCTURE

glue による一貫性のあるアーキテクチャ



SWITCHING TO MONOREPO

Polyrepo の問題点を解決





uber-go/fx

A dependency injection
based application framework
for Go.



FX

```
func NewLogger() *log.Logger {
    // ...
    return logger
}

func NewHandler(logger *log.Logger) (http.Handler, error) {
    // ...
    return http.HandlerFunc(func(http.ResponseWriter, *http.Request) {
        // ...
    }), nil
}

func NewMux(lc fx.Lifecycle, logger *log.Logger) *http.ServeMux {
    // ...
    return mux
}

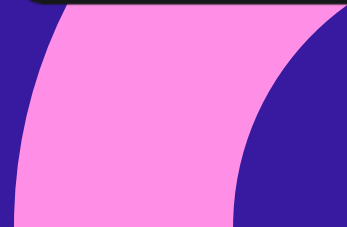
func Register(mux *http.ServeMux, h http.Handler) {
    mux.Handle("/", h)
}
```





FX


```
func main() {
    app := fx.New(
        fx.Provide(
            NewLogger,
            NewHandler,
            NewMux,
        ),
        fx.Invoke(Register),
    )
    ctx, cancel := context.WithTimeout(context.Background(),
    defer cancel()
    if err := app.Start(startCtx); err != nil {
        // ...
    }
    // ...
}
```





FX

```
func NewMux(lc fx.Lifecycle, logger *log.Logger) *http.ServeMux {
    mux := http.NewServeMux()
    server := &http.Server{
        // ...
    }
    lc.Append(fx.Hook{
        OnStart: func(context.Context) error {
            // ...
            go server.ListenAndServe()
            return nil
        },
        OnStop: func(ctx context.Context) error {
            // ...
            return server.Shutdown(ctx)
        },
    })
    return mux
}
```





uber-go/fx

APPLICATION FRAMEWORK

DI 機能を中心とする
薄い "アプリケーションフレームワーク"

DEPENDENCY INJECTION

Provider からよしなに依存関係を解決

RICH FEATURE

Hook, Timeout, Run func などの設定
開発者の依存モジュールへの認知負荷を軽減

Before Consistent Code Structure

NOT GOOD ARCHITECTURE

"Transport" 層と "Business" 層の混合

INCONSITENCY

一貫性がないため複数サービスの開発の認知負荷が高い



glue

Inspired by Clean Architecture

GLUE

```
$ glue new service
```

```
$ tree
```

```
.  
├── app  
├── controller  
├── entity  
├── gateway  
├── gen  
├── handler  
└── repository
```

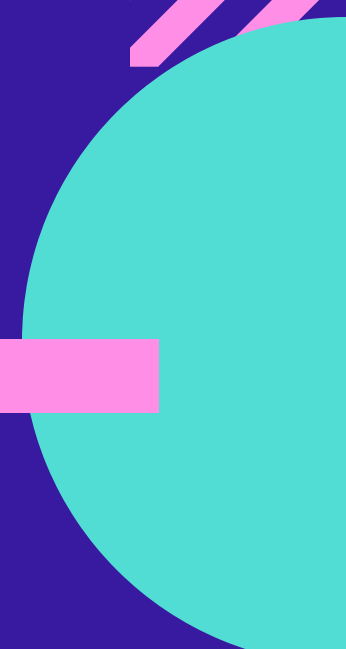

CONSITENT CODE STRUCTURE WITH GLUE

CONSITENT ARCHITECTURE

複数サービス開発の認知負荷が
小さくなる

LIKE CLEAN ARCHITECTURE

ビジネスロジックの分離
モジュール間の適切な依存



Polyrepo

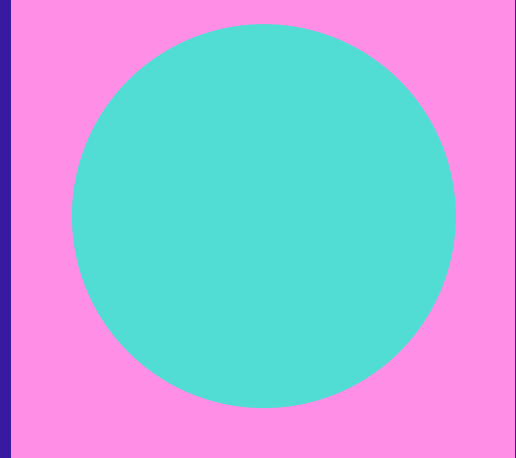
Before Monorepo

UPDATE PACKAGES

マイクロサービス全体の
1 package のアップデートに
合計数百 commit が必要

DUPLICATE CODE

複数のマイクロサービスが
同じコードを保有する





MONOREPO

EASY UPDATE OF PACKAGES

1 commit で全てのマイクロサービスの
package をアップデート
バージョン管理もシンプルに

SIMPLE CODE

少ない重複コード
共有・再利用・変更がしやすい

BUILD WITH BAZEL

Bazel を利用し高速・高再現性のビルド



Why Bazel?

Just my thought

FAST AND CORRECT

並列ビルドとキャッシュで高速
Go や protoc のバージョン固定
sandbox 環境

LESS DOCKERFILES

Bazel で Docker Image を
ビルドするので
Dockerfile の管理が
少なくなる

GOOD WITH GO

gazelle: Go 用 Bazel ファイル
自動生成ツール
シングルバイナリ



RECAP OF RECAP

DEPENDENCY INJECTION

uber-go/fx

- DI based application framework

CONSISTENT CODE STRUCTURE

glue - inspired by Clean Architecture

MONOREPO

Update a module by 1 commit

Bazel

