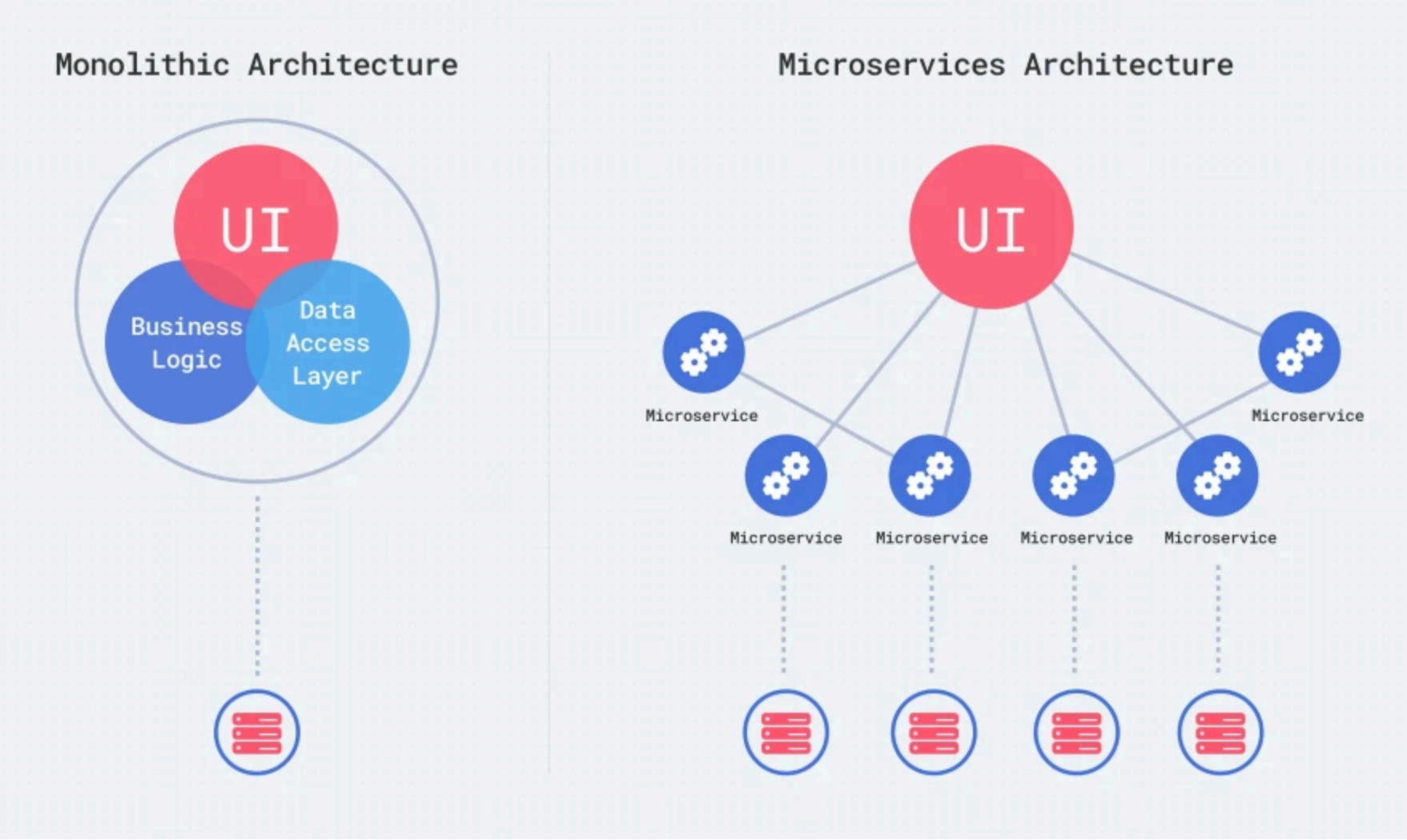
Apa itu Microservice?

- **Microservice** adalah pendekatan arsitektur untuk membangun aplikasi sebagai kumpulan dari layanan kecil yang dapat beroperasi secara independen.
- Setiap layanan merupakan proses yang berjalan sendiri dan berkomunikasi melalui protokol ringan seperti HTTP, WebSockets, atau gRPC.

Microservice vs Monolith



Mengapa Microservice?

- **Skalabilitas**: Meningkatkan kemampuan untuk menyesuaikan kapasitas aplikasi dengan kebutuhan.
- **Pengembangan Terpisah**: Tim dapat bekerja secara independen pada layanan yang berbeda.
- **Ketahanan**: Kerusakan pada satu layanan tidak mengakibatkan kegagalan total aplikasi.

Kapan Microservice Diperlukan?

- *Ketika aplikasi **sangat besar** dan **kompleks**.
- *Ketika tim pengembang **berkembang**, dan ada kebutuhan untuk **mengisolasi fungsi** untuk mencegah konflik.
- *Ketika perlu untuk **meningkatkan komponen** aplikasi secara independen.

Prinsip-Prinsip Microservice

1. **Loose Coupling**: Setiap layanan harus memiliki ketergantungan minimum pada layanan lain.
2. **Single Responsibility**: Setiap layanan melakukan satu tugas dan melakukannya dengan baik.
3. **Resilience**: Setiap layanan harus bisa mengatasi kegagalan dan memulihkannya tanpa mengganggu layanan lain.
4. **Decentralization**: Manajemen data dan kontrol dikelola secara terdistribusi.

Kelebihan dan Kekurangan Microservice

- Kelebihan
  - Modularitas: Mudah untuk mengelola dan mengembangkan.
  - Fleksibilitas Teknologi: Mampu menggunakan berbagai teknologi sesuai kebutuhan layanan.
- Kekurangan
  - Kompleksitas: Lebih sulit untuk dikelola dan diatur.
  - Overhead Komunikasi: Memerlukan pengelolaan komunikasi antar-layanan yang efisien.

Q n A

12 Factor Apps

https://12factor.net/

1. Codebase

- Satu basis kode dilacak dalam version control, banyak deploy.
- Semua aplikasi harus memiliki satu repository kode yang diatur dengan baik.
- Meskipun memiliki beberapa deploys (misalnya staging, production), mereka harus berasal dari codebase yang sama

2. Dependencies

- `Nyatakan dan pisahkan dependencies secara eksplisit.`

- Semua dependensi aplikasi harus dideklarasikan dengan jelas dan tidak boleh bergantung pada software atau libraries yang secara implisit ada di sistem.
- **Penting**: Memastikan aplikasi dapat diatur secara konsisten pada berbagai lingkungan tanpa masalah dependency.

Contoh

- Di Go, menggunakan `go.mod` untuk mendeklarasikan dependencies.

```
module myapp

go 1.20

require (
  github.com/gin-gonic/gin v1.7.4
)
```

- Di Python, menggunakan `requirements.txt` untuk mengelola dependencies.

3. Config

Simpan konfigurasi di luar kode.

- Konfigurasi seperti URL database, API keys, dan konfigurasi lainnya harus disimpan di environment variables.
- **Penting**: Memisahkan konfigurasi dari kode membuat aplikasi lebih aman dan lebih mudah untuk di-deploy di berbagai lingkungan.

Contoh:

- Gunakan `os.Getenv("VAR_NAME")` di Go untuk mengambil nilai environment variable.

```
dbHost := os.Getenv("DB_HOST")
```

4. Backing Services

Perlakukan layanan pendukung seperti database, cache, dan layanan pihak ketiga sebagai resources yang dapat diganti.

- Perlakukan semua layanan pendukung sebagai sumber daya yang terpasang dan dikelola secara terpisah dari aplikasi utama.
- **Penting**: Memudahkan penggantian layanan pendukung tanpa mempengaruhi kode aplikasi.

Contoh:

- Hubungkan aplikasi ke database dengan menggunakan environment variable untuk string koneksi.

```
db, err := sql.Open("postgres", os.Getenv("DATABASE_URL"))
```

5. Build, Release, Run

Pisahkan tahap build, release, dan run.

- **Build**: Kompilasi kode dan dependencies menjadi satu bundle.
- **Release**: Kombinasi dari build dengan konfigurasi tertentu.
- **Run**: Eksekusi aplikasi dalam lingkungan runtime.
- **Penting**: Memastikan bahwa build dapat diulang dengan hasil yang sama, dan memisahkan tahapan ini membantu mencegah perubahan mendadak di production.

Contoh:

- Gunakan CI/CD pipeline untuk otomatisasi tahap build, release, dan run.
- Contoh dengan GitHub Actions:

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Set up Go
      uses: actions/setup-go@v2
      with:
        go-version: 1.20
    - name: Build
      run: go build -o myapp
```

6. Processes

Jalankan aplikasi sebagai satu atau lebih proses stateless.

- Aplikasi harus berjalan sebagai proses yang independen, tanpa menyimpan status di memori lokal atau di disk. Semua status harus disimpan di layanan eksternal seperti database.
- **Penting**: Proses stateless memudahkan scaling dan meminimalkan risiko kehilangan data.

Contoh:

- Gunakan Redis untuk menyimpan session data daripada menyimpannya di memori lokal.

7. Port Binding

Aplikasi harus mengekspos layanan mereka melalui binding port.

- Aplikasi tidak harus bergantung pada server aplikasi eksternal, tapi langsung mengikat port dan menerima permintaan.
- **Penting**: Memungkinkan aplikasi untuk berjalan di berbagai lingkungan tanpa konfigurasi khusus.

Di Go, jalankan server HTTP langsung di port tertentu.

```
http.ListenAndServe(":8080", nil)
```

8. Concurrency

Gunakan proses untuk menangani tugas-tugas konkuren.

- Gunakan proses terpisah atau worker untuk menangani tugas yang dapat dijalankan secara bersamaan.
- **Penting**: Meningkatkan kinerja dan efisiensi aplikasi, terutama dalam beban kerja tinggi.

Contoh:

- Gunakan goroutines di Go untuk menjalankan tugas-tugas secara bersamaan.

```
go handleRequest(conn)
```

- Menggunakan Message Queue

9. Disposability

Proses harus cepat dimulai dan dihentikan dengan aman.

- Proses aplikasi harus dirancang untuk memulai dengan cepat dan dihentikan secara bersih untuk mendukung scaling dan pemulihan cepat.
- **Penting**: Meminimalkan waktu downtime dan meningkatkan keandalan aplikasi.

```
func main() {
        // The HTTP Server
        server := &http.Server{Addr: "0.0.0.0:3333", Handler: service()}

        // Server run context
        serverCtx, serverStopCtx := context.WithCancel(context.Background())

        // Listen for syscall signals for process to interrupt/quit
        sig := make(chan os.Signal, 1)
        signal.Notify(sig, syscall.SIGHUP, syscall.SIGINT, syscall.SIGTERM, syscall.SIGQUIT)
        go func() {
                <-sig

                // Shutdown signal with grace period of 30 seconds
                shutdownCtx, _ := context.WithTimeout(serverCtx, 30*time.Second)

                go func() {
                        <-shutdownCtx.Done()
                        if shutdownCtx.Err() == context.DeadlineExceeded {
                                log.Fatal("graceful shutdown timed out.. forcing exit.")
                        }
                }()

                // Trigger graceful shutdown
                err := server.Shutdown(shutdownCtx)
                if err != nil {
                        log.Fatal(err)
                }
                serverStopCtx()
        }()

        // Run the server
        err := server.ListenAndServe()
        if err != nil && err != http.ErrServerClosed {
                log.Fatal(err)
        }

        // Wait for server context to be stopped
        <-serverCtx.Done()
}
```

10. Dev/Prod Parity

Lingkungan development dan production dibuat semirip mungkin

- Hindari perbedaan besar antara environment development, staging, dan production.
- **Penting**: Memastikan aplikasi yang diuji di development berjalan dengan baik di production.

Contoh:

- Gunakan Docker untuk membuat container yang konsisten di semua lingkungan.

```
docker build -t myapp .
docker run -p 8080:8080 myapp
```

11. Logs

`Perlakukan logs sebagai event stream.`

- Aplikasi harus mengeluarkan logs sebagai stdout dan perlu aggregasi untuk disimpan dan dianalisa lebih lanjut
- **Penting**: Memudahkan pemantauan dan debugging di production tanpa harus mengubah aplikasi.

Contoh:

Cetak logs ke stdout dan gunakan layanan seperti Fluentd atau ELK Stack untuk mengumpulkan dan menganalisis log.

```
log.Info().Msg("hello world")
// Output: {"time":1516134303,"level":"info","message":"hello world"}
```

12. Admin Processes

`Jalankan tugas administrasi/manajemen sebagai proses satu kali.`

- Proses administratif seperti migrasi database atau tasks satu kali lainnya harus dijalankan sebagai proses terpisah, tidak dicampur dengan aplikasi utama.
- **Penting**: Memisahkan tugas administratif dari proses aplikasi utama untuk menjaga kebersihan dan stabilitas sistem.

Contoh:

Di Go, jalankan script migrasi database terpisah dari aplikasi utama.

```
// main.go
func migrate() {
    // database migration code
}
```

Q n A

Overview Go untuk Microservice

Mengapa Go?

- **Ringan dan Cepat**: Go memiliki runtime yang cepat dan efisien, ideal untuk microservice.
- **Konkuren**: Dukungan bawaan untuk goroutines mempermudah penanganan proses paralel.
- **Strong Typing**: Meminimalkan bug dan kesalahan selama pengembangan.

HTTP Router Standard di Go

- **Net/HTTP**: Paket bawaan Go untuk membangun server HTTP tanpa dependensi eksternal.
- **Mudah Digunakan**: Ideal untuk microservice sederhana dengan rute HTTP minimal.

```go
package main

import (
  "fmt"
  "net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "Hello, Microservice!")
}

func healthHandler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "ok")
}

func main() {
  http.HandleFunc("/hello", helloHandler)
  http.HandleFunc("/health", healthHandler)
  fmt.Println("Server berjalan di port 8080")
  http.ListenAndServe(":8080", nil)
}
```

Overview Go untuk Microservice

Mengapa Go?

- **Ringan dan Cepat**: Go memiliki runtime yang cepat dan efisien, ideal untuk microservice.
- **Konkuren**: Dukungan bawaan untuk goroutines mempermudah penanganan proses paralel.
- **Strong Typing**: Meminimalkan bug dan kesalahan selama pengembangan.

Mulai dari mana?

- https://go.dev/doc/
- https://go.dev/tour
- https://gobyexample.com/
- https://go101.org/
- https://quii.gitbook.io/learn-go-with-tests
- https://www.alexedwards.net/blog
- https://go-blueprint.dev/

Framework or router ?

- https://gofiber.io/
- https://gin-gonic.com/
- https://github.com/go-chi/chi
- https://echo.labstack.com/

Database

- Raw query or ORM? or generated?
- https://gorm.io/
- https://bun.uptrace.dev/
- https://sqlc.dev/
- https://entgo.io/
- https://github.com/volatiletech/sqlboiler

```go
package main

import (
  "fmt"
  "net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "Hello, Microservice!")
}

func healthHandler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "ok")
}

func main() {
  http.HandleFunc("/hello", helloHandler)
  http.HandleFunc("/health", healthHandler)
  fmt.Println("Server berjalan di port 8080")
  http.ListenAndServe(":8080", nil)
}
```

Hands on

- Membuat reverse proxy service
- Membuat simple E-reklame rest API