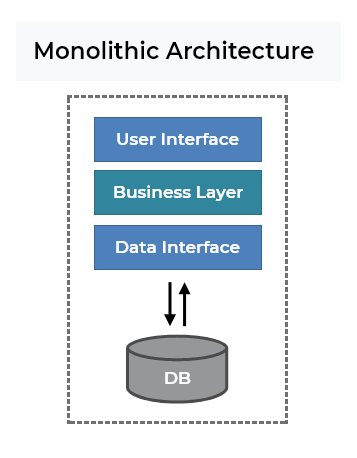
**02.02.2023 Golang Lesson Instruction**

**Monolithic application**

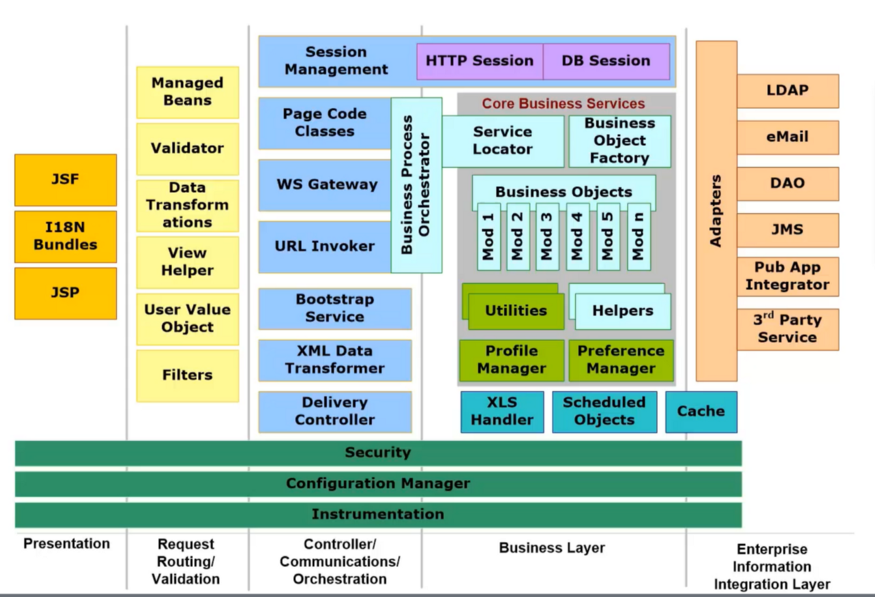
In software engineering, a monolithic application describes a single-tiered software application in which the user interface and data access code are combines into a single program from a single platform. A monolithic application is self-contained and independent from other computing applications.

A monolithic app **has all or most of its functionality within a single process or container and it’s componentized in internal layers or libraries**. The downside to this approach comes if or when the application grows, requiring it to scale. If the entire application scaled, it’s not really a problem.



Why Monolithic?

1. **Simple to develop** relative to microservices where skilled developers are required in order to identify and develop the services.
2. **Simple to deploy** We need to care about only one file or directory, not many deployments.
3. **Easier debugging and testing** Since a monolithic app is single indivisible unit, you can run end-to-end testing much faster, Easier to run the test.
4. **Less cross-cutting concerns** Cross-cutting concerns are the concerns that affect the whole application such as logging, handling, caching, and performance monitoring. In a monolithic application, these areas of functionality concern only one application so it is easier to handle it.



The above image is specific to a java application but concepts that we are going to discuss here is generic to all the application architect using monolithic architecture.

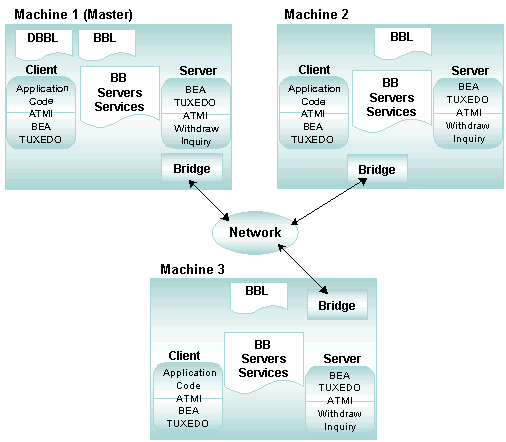
**Distributed Application**

What is distributed application?

Distributed application can be relatively simple, requiring a single computer and a single server, or more complex, allowing many client computers and several servers. For example, **web browsers** are distributed applications.

A distributed application consists of one or more local or remote clients that communicate with one or more servers on several machine linked through a network. With this type of application, business operations can be conducted from any geographical location.

A distributed application is a program that runs on more than one computer and communicates through a network. Some distributed applications are actually two separate software programs: the back-end (server) software and the front-end (client) software. Back-end software runs on a shared system (such as a shared Unix or VMS system) and manages shared resources, such as disks, prints, and modems. The back-end software also contains the main processing capability for the application. The front-end (client) software runs on workstations. It is the software you see when you use the application. It handles user interface functions, such as receiving input from a keyboard and displaying output to a screen. For example, web browsers are distributed applications. Browsers require back-end software servers on the World Wide Web as front-end software installed on your workstations.



**Microservices**

Microservices are **an architecture and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.** These services are owned by small, self-contained teams.

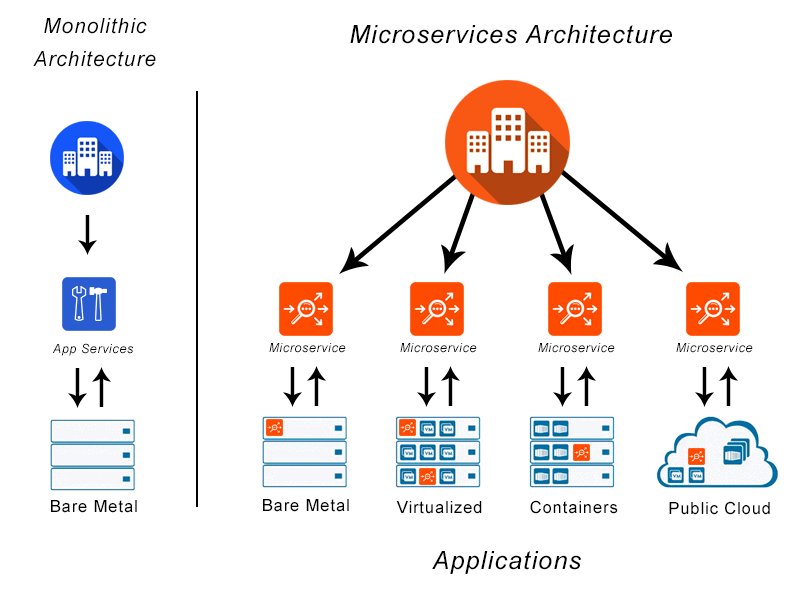
Why we should use microservices?

**Number of developers and development teams.** Microservices enable you to add more development teams, hence more developers to your application. Developers are more productive, because they aren’t stepping on one another’s toes as much as they are in a monolithic development process.

What problem does microservices solve?

Microservices solve the challenges of monolithic systems by being as modular as possible. In the simplest form, they help build an application as a suite of small services, each running in its own process and being independently deployable.

-------------------------------------------------------------------------------------------------------------

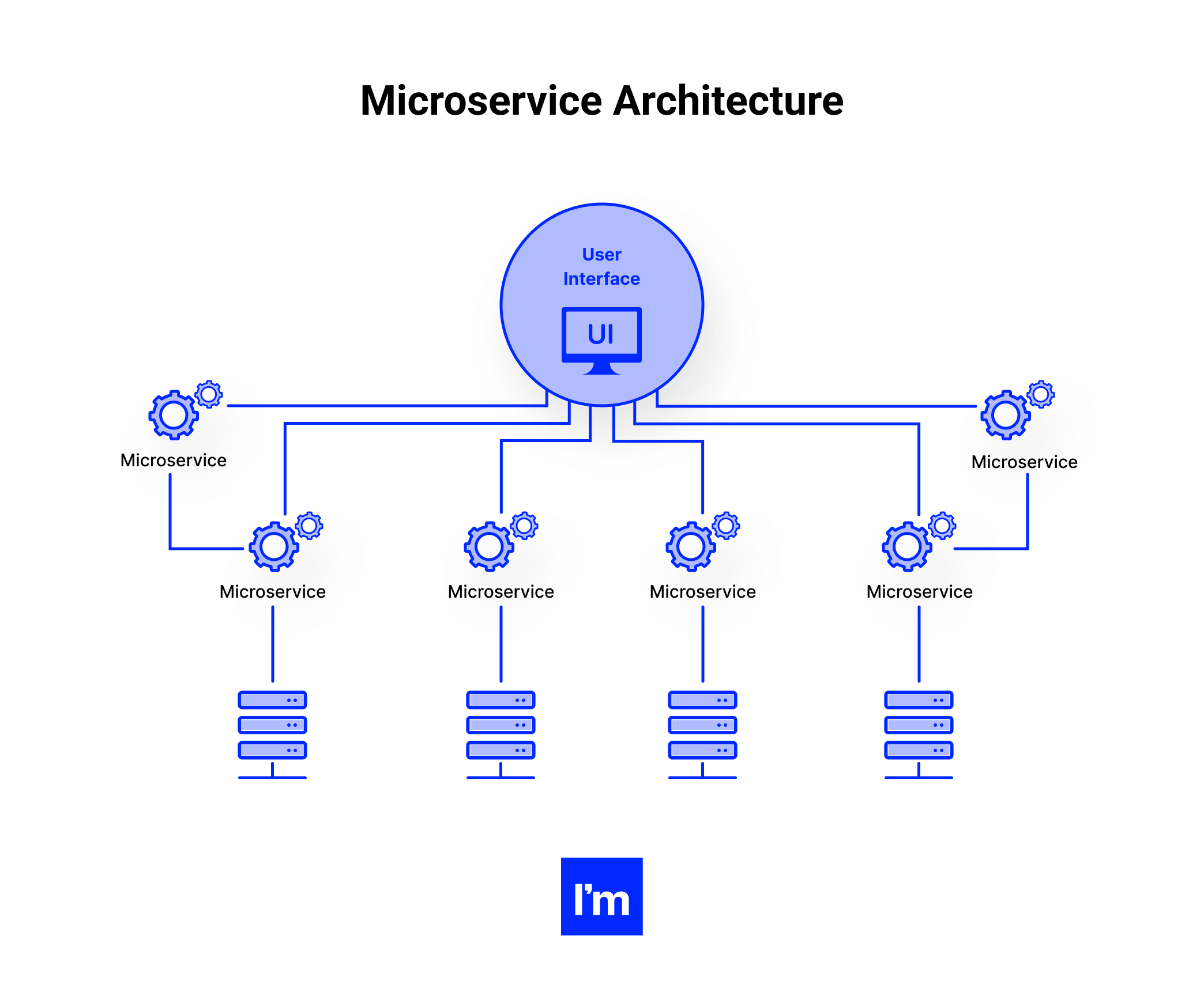


**Microservices in Golang**

Golang’s microservice architecture **ensures that developers don’t spend a lot of time reading other people’s code and get up to speed faster.** Even if you need to change the development team, you can still maintain and update your software without any problems.

Is Golang good for microservices?

Golang’s syntax is relatively small, and it has been is use for a long time without major changes. As there’s no need to learn new paradigms or syntax, code written with Go is transparent and easy to understand. This makes **Go ideal choice for backends, microservices, cloud-computing systems, and more.**



Building Microservice in Go.

The most use of it is when an application needs to be scalable; one language can perfectly fit the criteria, e.e., Golang. The reason is its inheritance with C-family programming languages as components written in golang are easier to combine with components coded in other languages that residence in the same family class. It is much more efficient than Cm C++ Frameworks used in it for Microservice Architecture.

**Performance and speed**

Golang compiles faster than most other languages, saving time for developes and money for business owners.

Go’s runtime performance is also excellent. Here are several benchmarks to illustrate that in many cases Golang outperforms popular programming languages such as Java and Python



**Creating first service**

* Add folder named “broker-service”, you can rename folder which you want.
* Go to “broker-service” folder and run *go mod init broker* command trough terminal.
* Add folder named “cmd” into “broker-service”
* Add folder named “api” into “cmd”
* Add new file named “main.go” into “api”
* Go to “broker-service” folder and run *go get github.com/go-chi/chi/v5* command through terminal.
* Go to “broker-service” folder and run *go get github.com/go-chi/chi/v5/middleware* command through terminal.
* Go to “broker-service” folder and run *go get github.com/go-chi/cors* command through terminal.