Microservices: Application composed of many small services. Each service is a deployable unit, and can have its own database. Each has a *REST API***.**

Microservices:

Pros:

* Can write each service independently, with different programming languages/technology stacks if needed.
* Smaller services easier to manage. Easier to look at it as a whole than a monolith.
* Easy to switch one out for another. Separation of concerns.
* Can be scaled up better. If one service is used a lot more than others, deploy it on many machines! In a monolith, this might be harder to do.

Downsides:

* HTTP (REST) calls, i.e. inter-process communication is more expensive than intra-process communication (which is used in a monolith).
* If operation needs to be atomic (all is completed, if not, roll back), but cross more than one microservice.. can no longer use transaction support provided by DBs (Since the operation can span multiple databases).
* Some code duplication may be possible. For example, we might need a user class in more than one service.
* When a microservice returns a list of items, and each requires access to data defined in another.. the client may end up doing (N + 1) requests.

Monoliths:

Pros:

* Straight forward to develop & deploy.
* Most process commuication is intra-process communication. All the services are in one place. These calls are very cheap.
* Scaling easy, just duplicate the whole monolith on many machines. (Wasteful however if the most used service is used 97% of the time; and that service may be only 3% of the codebase being duplicated)

Downsides:

* Code could get complicated, you have a lot more to consider when editing a huge codebase.
* Building could get slow once the application gets big.
* Harder to scale the most used service individually..