***Monolith architecture**: - functionalities coupled tightly i.e., everything is touching everything.*

- *Can scale horizontally in load.*
- *Good for small team, less interaction needed to understand things.*
- *Less parts because it is not broken into small parts.*
- *Common Test helper methods can be used.*
- *It is faster, because no Remote procedure calls needed to connect with other system for some functionality.*

Disadvantages

- *New team members need to understand whole system to make even small changes.*
- *Deployment is heavy and testing too.*
- *If once service crashes, whole system may fail too.*

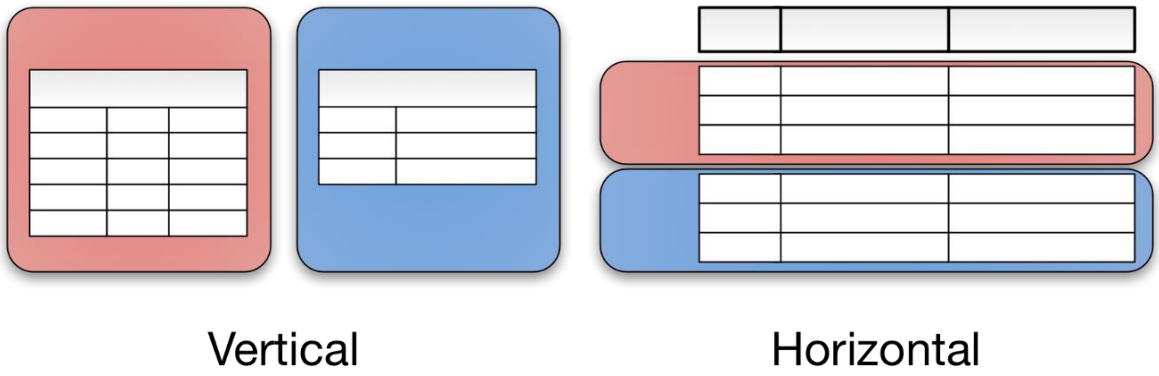
***Microservice architecture** - functionalities coupled loosely, independent services.*

- *Easier to scale, also, partially scaling is possible, i.e., the service needs scaling will be scaled only.*
- *Good to big teams, as new members need to understand only task related service.*
- *Parallel development is easy.*
- *Deployment is easy.*

Disadvantages

- *Not easy to design. Need skills to identify which service needs to be clubbed as one service and which not to be.*
- *Less fast than Monolith, because of network calls.*

Database partitioning.



Vertical partitioning *involves creating tables with fewer columns and using additional tables to store the remaining columns*. Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized.

Horizontal partitioning *involves putting different rows into different tables*. Perhaps customers with ZIP codes less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest, while a view with a union might be created over both to provide a complete view of all customers.

Sharding: Sharding involves breaking up one's data into two or more smaller chunks, called logical shards. The logical shards are then distributed across separate database nodes, referred to as physical shards, which can hold multiple logical shards. Despite this, the data held within all the shards collectively represent an entire logical dataset.

- speed up query response times, because less rows needs to be scanned.
- If one shard goes down, you whole service using that table won't go down.

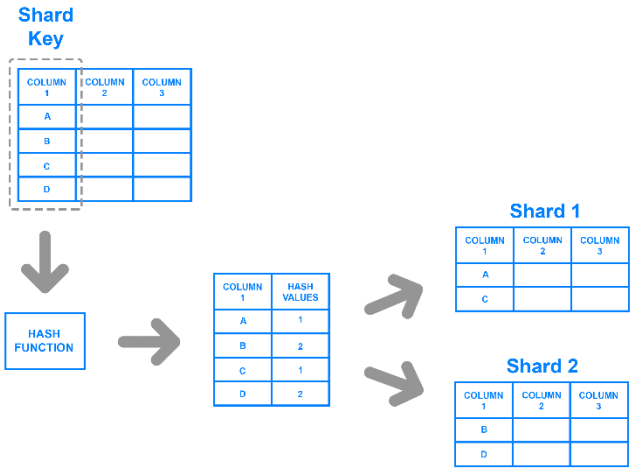
Disadvantages:

1. Hard to Decide the **key** on which shard needs to happen.
2. Joins across shards, to get the required data, will be slow
3. difficult to return it to its unsharded architecture

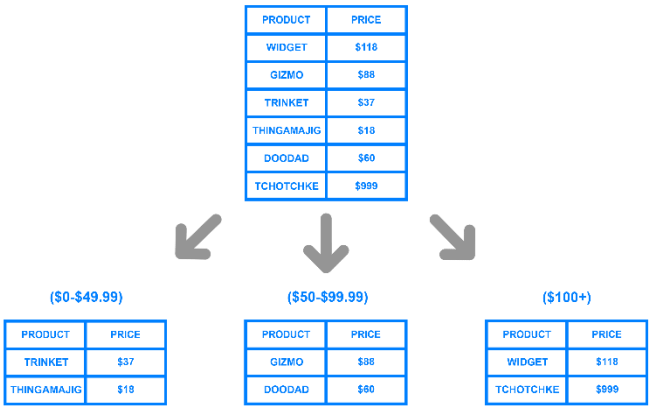
Horizontal partitioning *splits one or more tables by row, usually within a* ***single instance*** *of a schema and a database server,* ***Sharding*** *splits the tables across potentially* ***multiple instances*** *of the schema.*

Sharding Architectures

1. ***Key based –***



2. ***Range Based***



3. ***directory based-*** one must create and maintain a *lookup table* that uses a shard key to keep track of which shard holds which data.