Scalable systems can handle increased loads by adding resources while maintaining performance

↳ While maintaining cost-efficiency

Scalability bottlenecks

- Centralised components
    * Single database

- High latency operations
    * long-running data processing

- The bottlenecks can be made scalable by optimising their performance

- Implementing caching

- Replication to distribute the load

Stateless Architecture

- Architure does not keep track of state between requests

- More fault tolerant
  - ↳ If a server goes down, no state or data is lost

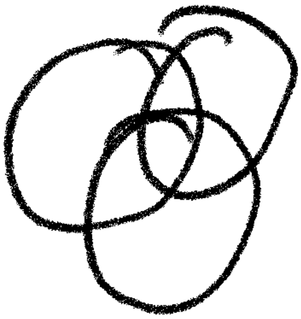
- State can be observed by using distributed caches or databases

Loose Coupling

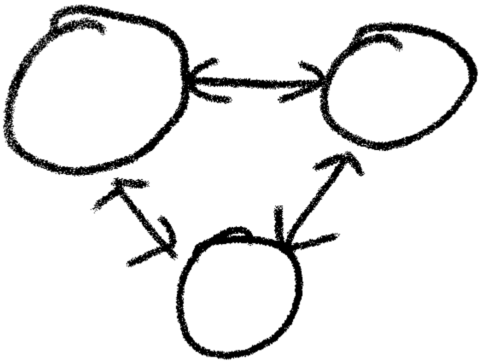
- Using well-defined interfaces (or APIs) for comms

- Makes it easy to modify/create microservices

- Components can operate independant of each other



Tight Coupling



Loose Coupling

Event-Driven Architecture

- Services emit/listen to events

- Allows for non-blocking operations to continue ⇒ Asnynchrous

- Helps mitigate tight coupling
  * Reduces risk of cascading failures

Asnynchronous processing introduces the following complexities:

- Error Handling
- Debugging
- Data Consistency

Vertical Scaling

- Increasing RAM/CPU of a machine

- Useful when its challenging to horizontally scale a system

- Has limitations - "you can only make a machine so powerful"

- More expensive than horizontal scaling

Horizontal Scaling

- Adding more machines to a system
- Better fault tolerance
- More cost-effective

Challenges

- Data consistency
- Increased network overhead
- Managing distributed systems

- long-running tasks => break them down into smaller chunks that can be run parrallel

- Queues can be split into multiple queues to spread load

Design patterns that can help distribute workload:

- Fan-out
- pipes
- Filters

Techniques for scalable systems

- Load balancing
  * Round robin } algorithms
  * Least connections } used

- Caching
  * Storing frequently accessed data closer to where its needed

- CDN
  * Offload traffic, improve response time for users globally

- Sharding
  * Splitting a monolithic database into multiple shards (stored on different servers)
  * Allows for parrallel processing