Common Architecture Design Principles

* **Open architecture** - Open API Standards, Contracts/SLA driven, Interoperable
* **Separation of concerns** - Loosely coupled micro service i.e. Divide your application into distinct features with as little overlap in functionality as possible. The important factor is minimization of interaction points to achieve high cohesion and low coupling. However, separating functionality at the wrong boundaries can result in high coupling and complexity between features even though the contained functionality within a feature does not significantly overlap.
* **Single responsibility** - Each component or micro service should be responsible for only a specific feature or functionality, or aggregation of cohesive functionality. For example a business service should not call another business service just because the other business service can provide the same data, rather it should call the data services and orchestrate the business capability and expose the required capability as a service. It may still use the same common components/frameworks underneath as an accelerator to achieve the business capability.
* **Principle of Least Knowledge -** (also known as the Law of Demeter or LoD). A component or micro service should not know about internal details of other components or service. For example a business service, which needs data, should call only data service to get data rather than knowing how to connect to backend.
* **Don’t repeat yourself (DRY)** - You should only need to specify intent in one place. For example, in terms of application design, specific functionality should be implemented in only one component or micro service; the functionality should not be duplicated in any other component or service.
* **Minimize upfront design -**Only design what is necessary. In some cases, you may require upfront comprehensive design and testing if the cost of development or a failure in the design is very high. In other cases, especially for agile development, you can avoid big design upfront (BDUF). If your application requirements are unclear, or if there is a possibility of the design evolving over time, avoid making a large design effort prematurely. This principle is sometimes known as YAGNI ("You ain’t gonna need it").
* **Plan for scale and failure** - Scale out - Each component or micro service should scale horizontally. Each component or micro service should be designed for resiliency - for failure and recovery. For example to avoid SPOF(single point of failure) in large scale systems – Do asynchronous processing as much as possible.
* **Know how to measure and operate** - Design for operations and visibility to problems, SLA and its traceability. For example - How easy to develop, test, ship/deploy, maintain, operate and control.

Key Architecture Design Consideration

* **Determine the application type -** 
  + Applications designed for mobile devices.
  + Rich client applications designed to run primarily on a client PC.
  + Rich Internet applications designed to be deployed from the Internet, which support rich UI and media scenarios.
  + Service applications designed to support communication between loosely coupled components.
  + Web applications designed to run primarily on the server in fully connected scenarios.
* **Determine the deployment strategy –** 
  + Your application may be deployed in a variety of environments, each with its own specific set of constraints such as physical separation of components across different servers, a limitation on networking protocols, firewall and router configurations, and more. Several common deployment patterns exist, which describe the benefits and considerations for a range of distributed and non-distributed scenarios. You must balance the requirements of the application with the appropriate patterns that the hardware can support, and the constraints that the environment exerts on your deployment options. These factors will influence your architecture design.
* **Determine Appropriate Technology** 
  + When choosing technologies for your application, the key factors to consider are the type of application you are developing and your preferred options for application deployment topology and architectural styles. Your choice of technologies will also be governed by organization policies, infrastructure limitations, resource skills, and so on. You must compare the capabilities of the technologies you choose against your application requirements, taking into account all of these factors before making decisions.
* **Determine the Quality Attributes**
  + Quality attributes—such as security, performance and usability—can be used to focus your thinking on the critical problems that your design should solve. Depending on your requirements, you might need to consider every quality attribute or you might only need to consider a subset. For example, every application design must consider security and performance, but not every design needs to consider scalability. Understand your requirements and deployment scenarios first so that you know which quality attributes are important for your design. Keep in mind that quality attributes may conflict; for example, security often requires a tradeoff against performance or usability. When designing to accommodate quality attributes, consider the following guidelines:
  + Quality attributes are system properties that are separate from the functionality of the system.
  + From a technical perspective, implementing quality attributes can differentiate a good system from a bad one.
  + There are two types of quality attributes: those that are measured at run time, and those that can only be estimated through inspection. Analyze the tradeoffs between quality attributes.
  + Questions you should ask when considering quality attributes include:
    - What are the key quality attributes required for your application? Identify them as part of the design process.
    - What are the key requirements for addressing these attributes? Are they actually quantifiable?
    - What are the acceptance criteria that will indicate that you have met the requirements?