Most used architetures on server systems developments

MVC Architeture

Concept
This architecture separates an application into three main components: Model, View, and Controller.

Usage
When you decide to work with a single server and you have defined clients to communicate with your server. Web and mobile applications.

Division
- **Model**: Manages the application logic and business rules through entities and use cases.
- **View**: It is the client interface used to interact with the server.
- **Controller**: Are server routes used to communicate with the client processing requests and responses.

Benefits
- This organizational structure promotes organized coding and simplifies maintenance.
- By segregating the application into distinct features, it's easier to scale the application.

Disvantages
For small applications, using MVC can introduce unnecessary complexity.

Microservices Architecture

Concept
Microservices architecture breaks down an application into smaller, loosely coupled services. Each service is responsible for a distinct feature of the application and communicates over a common interface like HTTP/REST or asynchronous messaging.

Usage
- When you need to split your application server in another servers to increase server performances.
- When you need to use different technologies across different servers.
- When you're working in organizations with multiple development teams, microservices allow each team to work independently on different services.
- When you need to isolate part of the application in a single service.

Division
Each server instance work is responsible for a service. Example: A system that is splited into a service to process payments, another to send emails, and another to hanle application requests.

Disvantages
- In some cases, is needed knowledge on messaging services (ex: RabbitMQ) to communicate between the application services.
- Multiple deployments required.

Serverless Architeture

Concept
This architecture is a cloud computing model where the cloud provider dynamically manages the allocation and provisioning of servers, without needing depending on own server structure.

Usage
- Easily build RESTful APIs that scale automatically and are triggered by HTTP requests.
- Serverless is ideal for processing data in real-time, such as processing images or videos uploaded to a cloud storage, or transforming data as it is entered into a database
- Perform tasks like backups, scheduled jobs, or sending emails without needing a dedicated server.

Benefits
- Functions are typically executed in response to events or triggers.
- Resources automatically scale based on the workload. If an application needs more resources due to an increase in traffic, the serverless platform will automatically allocate more resources.
- Costs are based on the actual amount of resources consumed by an application, rather than on pre-purchased capacity.
- Developers do not need to manage servers or server configurations, focusing instead on the code.

Disvantages
- When a serverless function is invoked after being idle, there can be a noticeable latency because the cloud provider needs to initialize a runtime container. This cold start can affect performance, particularly for functions that need to respond in real time.
- Debugging serverless applications can be more complex compared to traditional architectures.
- Serverless functions typically have execution time limits (e.g., AWS Lambda functions can only run for up to 15 minutes). This makes them unsuitable for long-running processes.