**Trade-offs for Monoliths and Microservices**

**Summary**

An application can be designed using both, monolithic or micro service-based architectures. So far, we have looked at how the structure of an application is contoured by functional requirements and available resources. However, each architecture has a set of trade-offs, that need to be thoroughly examined before deciding on the final structure of the application.

These trade-offs cover development complexity, scalability, time to deploy, flexibility, operational cost, and reliability. Let's examine each trade-off in more detail!

**Summary**

**Development Complexity**

Development complexity represents the effort required to deploy and manage an application.

* **Monoliths** - one programming language; one repository; enables sequential development
* **Microservice** - multiple programming languages; multiple repositories; enables concurrent development

**Scalability**

Scalability captures how an application is able to scales up and down, based on the incoming traffic.

* **Monoliths** - replication of the entire stack; hence it's heavy on resource consumption
* **Microservice** - replication of a single unit, providing on-demand consumption of resources

**Time to Deploy**

Time to deploy encapsulates the build of a delivery pipeline that is used to ship features.

* **Monoliths** - one delivery pipeline that deploys the entire stack; more risk with each deployment leading to a lower velocity rate
* **Microservice** - multiple delivery pipelines that deploy separate units; less risk with each deployment leading to a higher feature development rate

**Flexibility**

Flexibility implies the ability to adapt to new technologies and introduce new functionalities.

* **Monoliths** - low rate, since the entire application stack might need restructuring to incorporate new functionalities
* **Microservice** - high rate, since changing an independent unit is straightforward

**Operational Cost**

Operational cost represents the cost of necessary resources to release a product.

* **Monoliths** - low initial cost, since one code base and one pipeline should be managed. However, the cost increases exponentially when the application needs to operate at scale.
* **Microservice** - high initial cost, since multiple repositories and pipelines require management. However, at scale, the cost remains proportional to the consumed resources at that point in time.

**Reliability**

Reliability captures practices for an application to recover from failure and tools to monitor an application.

* **Monoliths** - in a failure scenario, the entire stack needs to be recovered. Also, the visibility into each functionality is low, since all the logs and metrics are aggregated together.
* **Microservice** - in a failure scenario, only the failed unit needs to be recovered. Also, there is high visibility into the logs and metrics for each unit.

Each application architecture has a set of trade-offs that need to be considered at the genesis of a project. But more importantly, it is paramount to understand how the application will be maintained in the future e.g. at scale, under load, supporting multiple releases a day, etc.

There is no "golden path" to design a product, but a good understanding of the trade-offs will provide a clear project roadmap. Regardless if a monolith or microservice architecture is chosen, as long as the project is coupled with an efficient delivery pipeline, the ability to adopt new technologies, and easily add features, the path to could-native deployment is certain.