**Description**

A monolithic architecture means that your app is written as one cohesive unit of code whose components are designed to work together, sharing the same memory space and resources.

A microservice architecture means that your app is made up of lots of smaller, independent applications capable of running in their own memory space and scaling independently from each other across potentially many separate machines.

**Monolithic pros**: The major advantage of the monolithic architecture is that most apps typically have a large number of cross-cutting concerns, such as logging, rate limiting, and security features such audit trails and DOS protection.

When everything is running through the same app, it’s easy to hook up components to those cross-cutting concerns.

There can also be performance advantages, since shared-memory access is faster than inter-process communication (IPC).

**Monolithic cons**: Monolithic app services tend to get tightly coupled and entangled as the application evolves, making it difficult to isolate services for purposes such as independent scaling or code maintainability.

Monolithic architectures are also much harder to understand, because there may be dependencies, side-effects, and magic which are not obvious when you’re looking at a particular service or controller.

**Microservice pros**: Microservice architectures are typically better organized, since each microservice has a very specific job, and is not concerned with the jobs of other components. Decoupled services are also easier to recompose and reconfigure to serve the purposes of different apps (for example, serving both the web clients and public API).

They can also have performance advantages depending on how they’re organized because it’s possible to isolate hot services and scale them independent of the rest of the app.

**Microservice cons**: As you’re building a new microservice architecture, you’re likely to discover lots of cross-cutting concerns that you did not anticipate at design time. A monolithic app could establish shared magic helpers or middleware to handle such cross-cutting concerns without much effort.

In a microservice architecture, you’ll either need to incur the overhead of separate modules for each cross-cutting concern, or encapsulate cross-cutting concerns in another service layer that all traffic gets routed through.  
Eventually, even monolthic architectures tend to route traffic through an outer service layer for cross-cutting concerns, but with a monolithic architecture, it’s possible to delay the cost of that work until the project is much more mature.

Microservices are frequently deployed on their own virtual machines or containers, causing a proliferation of VM wrangling work. These tasks are frequently automated with container fleet management tools.

Good to hear:

* Positive attitudes toward microservices, despite the higher initial cost vs monolthic apps. Aware that microservices tend to perform and scale better in the long run.
* Practical about microservices vs monolithic apps. Structure the app so that services are independent from each other at the code level, but easy to bundle together as a monolithic app in the beginning. Microservice overhead costs can be delayed until it becomes more practical to pay the price.

Red flags:

* Unaware of the differences between monolithic and microservice architectures.
* Unaware or impractical about the additional overhead of microservices.
* Unaware of the additional performance overhead caused by IPC and network communication for microservices.
* Too negative about the drawbacks of microservices. Unable to articulate ways in which to decouple monolithic apps such that they’re easy to split into microservices when the time comes.
* Underestimates the advantage of independently scalable microservices.