# Comparitive Advantages and Disadvantages

This chapter will elaborate on the advantages and disadvantages of either architecture.

## Monolithic Architecture

There are multiple problems but also some advantages that could make monolithic architecture the right choice for an application:

Advantages:

1. **Simplicity** – Up to a certain size they are in some respects simpler and easier to manage.
2. **Performance –** Monolithic applications’ performance can be higher

Disadvantages:

1. **Learning curve –** Monolithic applications are bound to one language are complex systems are difficult to understand.
2. **Debugging and Tweaking –** Due to tight coupling and the shear complexity of a large monolithic app effective debugging and tweaking can become very difficult.
3. **Teamwork and Development ­–** Working as a team is made difficult by the language constraint and the single entity nature of a monolithic application.
4. **Reusability –** Parts or modules can only be partially reused as they often have a custom interface and are tightly coupled with the application.
5. **Cross-platform –** Developing cross platform requires a separate server and application.
6. **Scalability –**While not inherently bad monolithic applications are at a disadvantage compared to microservices when it comes to scaling

### Development

This chapter details the pros and cons of developing a monolithic application. These advantages and disadvantages are based on the Django framework but most of them are relevant to other frameworks.

#### Simplicity

Monolithic architecture is simple for a small system, if everything runs on one server it is easy to deploy and manage. With simple monolithic applications things like eventual consistency and monitoring of the services don’t have to be worried about.

#### Learning curve

Learning monolithic programming might seem simple in the beginning especially with a framework like Django. It is easy to get started with the framework and one can have a functioning web application up and running in no time and without prior knowledge of the framework and concept. This doesn’t come without some negative side-effects though, even though you might be able to write a simple application quickly, you’ll most likely lack an understanding of the processes going on under the hood. This may eventually lead to problems down the road when you have to scale up the application or customize it further. Django and comparable frameworks are also quite opinionated which can be challenging if you are trying to do things “your own way”. The pre-existing code and structure almost forces you, except if you have an excellent knowledge of the framework, to adhere to a very strict, predefined way of doing things. This is especially

bad for people used to working with other framework which impose different opinions.

An advantage of Django is that a developer can theoretically develop a whole application in one language. This means it only requires learning one language to build a simple web app but sadly in practice one can’t really get away without JavaScript, HTML, CSS and others.

#### Debugging and Tweaking

Django offers handy tools and helpers for debugging but nevertheless it can quickly become tedious and time consuming due to the structure of the application. Bugs can be hidden multiple layers and dependencies down. To make a new addition or changes the whole system has to be brought down, rebuilt and restarted leading to delays in the development time, especially with bigger systems. Changes propagate through the whole system and can break otherwise working code. This means systems have to be planned carefully and each change to a module with dependant modules has to be thought through thoroughly checking that each dependant module will still work correctly. This makes it hard to change things after they have been implemented. This is especially bad for upgradability but also maintenance of an application.

#### Teamwork and Development

The more complex a system gets the harder it is for an outsider to understand. Due to the tight coupling and dependencies this can lead to, what in software development is sometimes referred to as, a “big ball of mud”, where no single developer or even a group understands the application as a whole [14].

The nature of a monolithic application also requires the developers to use one language for the whole system, causing two further problems. New developers can only be hired if they are familiar with the language or have to learn the language and frameworks/libraries used and secondly a developer cannot choose the language best suited for the task. For example, Django wants the developer to use a slightly altered version of python developed by Django when working with templates and the client-side where JavaScript would usually be used.

Working with a team of developers brings its own set of challenges to the table. Interfaces and dependencies have to be laid out precisely before development to avoid uncertainties and incorrect uses of them. The inherent structure of the application as one entity also means that working on it at the same time requires special tools and thought from the team. Bringing it into production and testing are also affected by this since the whole application has to be brought down to implement a change.

#### Reusability

Due to the lack of a standard interface parts of a monolithic application only offer limited reusability. Modules written in other languages can also not be reused and have to be rewritten. This makes work between teams migrating from existing software or upgrading a lot harder.

#### Cross-platform

In today’s fast-moving competitive environment having a mobile application is crucial for any business. Touch and traditional mouse control are very different ways of user interaction with the application and therefore usually require developing separate applications. If a company has an existing monolithic web application there is also no obvious solution for converting it to a mobile application. All of this increases the total cost of ownership and development.

### Production

#### Performance

Frameworks like Django usually offer pretty good optimisation and especially for simple applications performance is high. The monolithic nature of the application makes call to separate modules fast as they don’t have to go over the network

#### Scalability

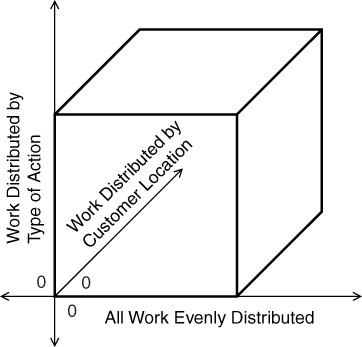


Figure 9, AFK Scale Cube [15]

Scalability is one of the key factors in application development. Figure 9, is a model to illustrate scalability taken from the book “The Art of Scalability” by Martin L. Abbott and Michael T. Fisher [15].

The x-axis in this model represents scaling an application by running multiple identical copies of the application behind a load balancer. This of course improves capacity and availability of the system and is probably the simplest way to do so. To optimize this type of scaling it should be noted that the application should be stateless which means each and every request can be routed to most optimal server regardless of previous requests.

The z-axis is scaling by separating and distributing the work by customer, customer need, location, or value this also includes “sharding” in a database which is a way of simplifying and minimizing a database. Unfortunately, this becomes very complicated quickly and makes it very hard to change something in the database. A company could say for example separate the paying Users into a subset with faster or more servers providing higher SLA and the free users into a slower one.

These are the ways of scaling a monolithic application, the third dimension, discussed in the next chapter, makes use of microservices to optimize the scalability to a maximum. So, the y-axis does not apply to monolithic applications which means they provide a lower degree of scalability. It should also be noted that the z-axis is the hardest and most cost intensive way of scaling an application. [15, 16]

## Microservices Architecture

As with monolithic architecture there are many different frameworks to ease the development process. The application detailed in this paper was written with the Flask framework using the REST approach to micro-servicing. It is what is commonly referred to as a RESTful application.

Advantages:

1. **B2B (Business to Business) integration –** Companies can sell or provide their API to other businesses.
2. **Cross-platform –** Only the client has to be redeveloped for a mobile Application.
3. **Maintainability –** Updating and implementing changes or new features is very simple.
4. **Scalability –** Microservice application can scale in a whole new dimension and naturally have faster and simpler databases
5. **Teamwork and Outsourcing –** Companies can outsource development of a service and coordinating teamwork is easy.
6. **Learning Curve** – Microservices are language independent and the microservice and REST concepts are easy to grasp.

Disadvantages:

1. **Performance –** High latency and network speeds between services can cause performance issues.
2. **Maintainability** **–** Maintaining a whole network of hundreds of services can be tricky especially with reliability and redundancy take into consideration.
3. **Client Performance –** The clients’ machine and browser must have enough processing power to send out many requests simultaneously.
4. **Eventual consistency –** Due to the separation of services information can sometimes be inconsistent in the system for short periods of time

### Development

Again, like in the previous chapter, this chapter will detail the pros and cons of using a microservice architecture for a web application. Also like before, these advantages and disadvantages refer specifically to a RESTful application using Flask but are relevant to most other microservice based systems.

#### Learning Curve

Learning to code in a microservice architecture is fairly easy. Services are simple and isolated and can be written in any language. In a well-designed application, a developer can use other services without having to worry about someone else’s work. This means that new developers can be hired easily and can start work right away instead of having to get an overview over the entire application first or having to learn a new framework and or language. A microservice based web application is also very similar to SOA which is a very common software development architecture, that many developers are already familiar with. As mentioned above a service is language independent allowing the developer to choose the language best suited for the task at hand.

#### Teamwork and Outsourcing

Working in a team and outsourcing is greatly simplified when developing microservice applications. Due to the loosely coupled nature of services, they can be completely independently updated, changed, taken offline or brought up. Small teams or individual developers can work on separate fully independent services and don’t have to know anything about how the other services in the system work. This of course also applies to outsourcing a service to a different company. The only thing everyone has to know about is the Interface, which is standardized and easily documented with interactive tools like Open API, detailed in §4.2.5. This also means that the front and backend can be completely separate and can be worked on by different teams or even companies with little extra effort.

#### B2B (Business to Business) integration

With a microservice based architecture it is easy for companies to provide an API for B2B use. A few prominent examples of this are the google maps API from google or the iframe-API from YouTube for embedding a YouTube video into your application. Companies also have the option to sell their services directly to other businesses.

#### Cross-platform

Developing for multiple different platforms requires little extra effort, at least if it is a well-designed RESTful application. This is because the client calls the services as it needs them and puts it together on the device which entails

### Production

#### Performance

The performance of a microservice application is very much dependant of the speed of the network between the microservices. If a microservice has to call 20 other services over a slow connection this will cause a 20-fold delay. The connection speed of the client also matters as a web based microservice application can often require upwards of a hundred calls to build a webpage. This is one of the downsides of microservices the more “fine-grained” the services are the more calls it will require to fetch data from the server. If the user’s connection has a high latency, upwards of 100ms for example that means those 100 calls will take quite a while to fulfil, causing a delay and maybe even graphical glitches when building the page in the browser.

#### Client Performance

The client’s browser and computer must be powerful enough to handle sending out all of these requests and then processing the information received and painting the webpage. This is more an issue of the past though as computers and browsers have become more than capable of handling such requests. Nevertheless, companies might have to support customers with slow machines and old browsers and therefor this issue needs to be considered.

#### Scalability

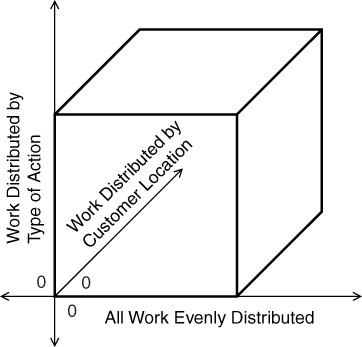


Figure 10, AFK Scale Cube [15]

Again, we will use the AFK Scale Cube, shown in Figure 10 to visualise scaling. Microservices can of course be scaled along the z-axis by just adding multiple instances of a service running behind a load balancer same concept as with a monolithic application. The z-axis can also be scaled in the same way as with a monolithic application, described in §5.1.2.2. Now comes the new part scaling out into the y-axis, which represents the distribution and separation of work responsibilities or data meaning among multiple entities [15]. In other words, separating the work into microservices.

Due to the independence of microservices it is common practice to use small specific databases for services, this means databases are fast and simple to comprehend.

#### Maintainability

Updating a service is comparatively easy, a microservice can be updated and rewritten as much as needed as long as it still complies with its interface. This means maintaining such a system is in some ways easier because changes and optimisation can be implemented fairly easily.

#### Eventual consistency

If you have different services handling different operations and they don’t share a database, it can lead to inconsistency “windows” where information is not consistent throughout the whole system yet. This is quite a common problem and often websites will tell you that the update you just made will only show up in a few minutes, in many cases this is due to this brief inconsistency across the system. This is not just an annoyance for the user though it can cause serious problems for algorithms deciding things based on false or outdated information.