**Application Architectures**

**This article explores three common application architectures as well as their benefits and drawbacks.**

It’s been over 15 years since Plant Pals first launched their online plant store. In that time, the store’s codebase has grown to over 10,000 files and 10,000,000 lines of code! The source code is so disorganized that new hires must spend months studying it.

When developers finally start making commits, it is difficult to predict their consequences. They frequently introduce bugs in existing features. Simple mistakes can break the entire application. In fact, it is so bad that the team has become afraid of adding features. It has gotten to the point where they only release one update a year!

Plant Pals could probably benefit from examining their application architecture. An application’s architecture is the framework for how an app is designed, built and deployed. With a more optimal architecture, Plant Pals would have less trouble scaling and maintaining their application.

In this article, we’ll explore the three most common application architectures. These architectures fall on a spectrum, ranging from simple to complex. Let’s begin with the oldest and simplest, the monolithic architecture.

**Monolithic Architecture**

In a **monolithic architecture**, an entire application and all its features live within a single codebase. The application is written in a single language. When developers add features, they must redeploy the entire application. This is the architecture currently used by Plant Pals.

Monolithic applications, or monoliths, have been around since in-house infrastructure was the norm. Since then, several other types of architecture have also become popular. Still, a monolithic architecture has its benefits over other types.

**Monolithic Architecture Benefits**

The benefits of a monolithic architecture stem from its simplicity. These benefits include (click on each item to learn more!):

1. **Speedy Initial Development**

* Starting to write a monolithic application is fast. A developer simply chooses a language and framework they are comfortable with. It is possible to get a basic application up and running in minutes.

1. **Simple Deployment**

* Monolithic applications are simple to deploy since they live in a single codebase. The entire application can be started from a single file. It can run on almost any infrastructure from traditional to serverless.

1. **Simple Testing**

* Like deploying a monolith, testing a monolith only requires starting a process on one computer. More complex architectures may require networking, monitoring and many servers to be configured in order to test the application.

**Monolithic Architecture Drawbacks**

Despite its simplicity, a monolithic architecture has several drawbacks. These drawbacks tend to become more pronounced as the application grows larger.

1. **Single Point of Failure**

* In a monolith, all features share the same code and thus are interdependent. An error in one feature can make the entire application unusable. This fragility also extends to the monolithic infrastructure as well. A monolithic application uses a smaller and more concentrated set of infrastructure components. Failures in these components can bring down the entire application.

1. **Inefficient Scaling**

* Keeping up with increased demand requires deploying more instances of the application. Each instance needs enough resources to load the entire application. This requirement holds even if a single feature drives the increased demand — due to the monolithic structure, that feature cannot be scaled independently. This limitation leads to allocating more physical infrastructure than is needed.

1. **Complex Codebase**

* As a monolith grows, its codebase becomes quite large and difficult to understand. When working in one area of an application, developers may change code that is a dependency of other features. If developers aren’t aware of these dependencies, they can introduce unexpected bugs.

**Monolithic Architecture Use Cases**

Despite the advent of newer architectures, monoliths still have their uses. A monolithic architecture can be a good choice when:

* Developing small, internal applications
* Starting a large application and then switching architectures when complexity increases

There is another architecture which mitigates some of the issues faced by a monolith while only slightly increasing its complexity. This architecture is n-tier architecture.

**N-Tier Architecture**

**An n-tier architecture** splits an application into several layers. Each layer has a distinct responsibility. When a layer is hosted on its own dedicated server, it is called a tier. Other names for this architecture are multi-tier and multi-layer architecture.

A three-tier application is the most common type of n-tier architecture. This application consists of the following layers:

* **Presentation layer**: This layer is what the user sees and interacts with.
* **Logic layer**: This layer contains all the business logic and decision making.
* **Data layer**: This layer handles interacting with a database.

Plant Pals could implement an n-tier architecture by maintaining their website front-end in the presentation layer, handling payments in the logic layer, and managing user data and inventory in the data layer.

N-tier architecture, like the monolithic architecture, has been around for a long time. It is a time-tested approach for enterprise applications. Let’s see why n-tier could be used over a monolithic architecture.

**N-Tier Architecture Benefits**

An n-tier architecture has the following improvements over a monolithic architecture.

1. **Separation of Concerns**

* Having distinct responsibilities for each layer makes their codebases simpler. It enables each development team to specialize in one area of the application. Teams can make changes to one layer without worrying about affecting other layers.

1. **Better Scalability**

* The tiers within an n-tier application can be scaled independently of each other based on demand. This independence leads to more efficient use of the underlying infrastructure.

**N-Tier Architecture Drawbacks**

N-tier architectures still suffer from some of the problems faced by monolithic architectures. The main problems are:

1. **Several Points of Failure**

* An entire tier within an n-tier application can still be brought down by one error. Though the other tiers may remain intact, the application is still vulnerable.

1. **Complex Deployment**

* Deploying several tiers is more complicated than deploying a monolith. Extra thought must be given to communication between tiers, logging and performance monitoring.

**N-Tier Use Cases**

N-tier architectures are a good middle ground between monolithic and more complex architectures (discussed next). Many companies have sustained successful n-tier applications for years. This architecture can be a suitable choice for:

* Large internal applications
* Enterprise applications when other architectures are deemed undesirable

The final architecture we’ll examine is the most complex. It is known as microservices architecture.

**Microservices Architecture**

**Microservices architecture** refers to an application where features are spread across different services. Each service is responsible for a tightly defined component of business logic. Services should aim to have smaller, independent codebases. These aspects make microservices a more granular approach than architectures like n-tier.

Let’s say our friends at Plant Pals want to switch from their monolith to microservices. They would create a microservice for each individual component of their online store. One service could handle product searches while another could handle payments. Each service would directly access only its own data. However, they can also communicate over a network. In the end, they might have an architecture that looks something like this:

The microservices architecture is relatively new compared to monolithic and n-tier architectures. It is closely associated with DevOps and cloud-based infrastructure. Let’s see how microservices solve many of the issues faced by other architectures.

**Microservices Architecture Benefits**

Microservice architectures offer a number of benefits related to scalability and reliability. Here are the major advantages.

1. **Resistance to Failures**

* A well designed microservices application has no single point of failure. This is because services are deployed independently and each access their own data. If an error occurs in a service related to payment, the search service can continue to function.

1. **Superior Scalability**

* Much like the tiers of an n-tier application, microservices can be scaled independently. If one service is in high demand, more instances can be deployed than other services. The smaller the size of the service, the more efficiently it can be scaled to meet demand.

1. **Diverse Technology**

* Microservices applications are not limited to any one language or technology. Each service can use the technology that is best suited for the task it performs.

1. **Smaller Codebases**

* Each microservice has its own codebase and is often managed by its own team. Separate codebases are smaller, more maintainable, and simpler to understand.

**Microservices Architecture Drawbacks**

The drawbacks of microservices come from the complexity of having many components. These are the main challenges to a microservices application:

1. **Slower Initial Development**

* Getting an application up and running is not nearly as simple as with a monolith. A microservice architecture requires creating and deploying many small services whose interactions can become complex.

1. **Complex Deployment**

* Deploying microservices is even more complicated than deploying n-tier applications. It requires setting up inter-service communication, logging, monitoring, and performance tuning.

1. **Difficult Testing**

* Each service often depends on sending or receiving data from one or more other services. Developers must find ways to mock up the other services to test their functionality.

**Microservices Architecture Use Cases**

Microservices applications are complex to set up. However, they provide many long-term benefits over other architectures. So, they are generally used for one purpose: large, enterprise applications. Netflix, Facebook, Amazon, and Google are just some popular applications that deploy microservices.

**Review**

In this article we explored some popular options for an application’s architecture. Monoliths can work well for small applications, but larger ones suffer in the long run. N-tier applications can work for enterprise applications, but still face similar issues. Microservices are complex to set up, but excel in scalability and maintainability. Microservices have become the modern standard for large, enterprise applications.

An application’s architecture is an important factor in how well it can scale and be maintained. The next time you develop an application, consider these factors when choosing its architecture. With some careful planning, you can avoid the current situation at Plant Pals.

* [**Resource**](https://www.codecademy.com/content-items/c2a72d7debd1ed82435cac67da8932b1#:~:text=as%20microservices%20architecture.-,Microservices%20Architecture,-Microservices%20architecture%20refers)