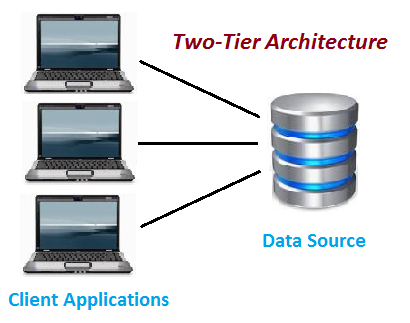
**Two-Tier Architecture:**

The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.

[](https://www.softwaretestingclass.com/what-is-difference-between-two-tier-and-three-tier-architecture/two-tier-architecture/)Two-Tier Architecture

The above figure shows the architecture of two-tier. Here the direct communication between client and server, there is no intermediate between client and server.

Let’s take a look of real life example of Railway Reservation two-tier architecture:

Let’s consider that first Person is making Railway Reservation for Mumbai to Delhi by Mumbai Express at Counter No. 1 and at same time second Person is also try to make Railway reservation of Mumbai to Delhi from Counter No. 2

If staff from Counter No. 1 is searching for availability into system & at the same staff from Counter No. 2 is also looking for availability of ticket for same day then in this case there is might be good change of confusion and chaos occurs. There might be chance of lock the Railway reservation that reserves the first.

But reservations can be making anywhere from the India, then how it is handled?

So here if there is difference of micro seconds for making reservation by staff from Counter No. 1 & 2 then second request is added into queue. So in this case the Staff is entering data to Client Application and reservation request is sent to the database. The database sends back the information/data to the client.

In this application the Staff user is an end user who is using Railway reservation application software. He gives inputs to the application software and it sends requests to Server. So here both Database and Server are incorporated with each other, so this technology is called as “***Client-Server Technology***“.

The Two-tier architecture is divided into two parts:

**1) Client Application (Client Tier)  
2) Database (Data Tier)**

On client application side the code is written for saving the data in the SQL server database. Client sends the request to server and it process the request & send back with data. The main problem of two tier architecture is the server cannot respond multiple request same time, as a result it cause a data integrity issue.

**Advantages:**

1. Easy to maintain and modification is bit easy
2. Communication is faster

**Disadvantages**:

1. In two tier architecture application performance will be degrade upon increasing the users.
2. Cost-ineffective

**Three-Tier Architecture:**

**Three-tier architecture** typically comprise a presentation tier, a business or data access tier, and a data tier. Three layers in the three tier architecture are as follows:

**1) Client layer**  
**2) Business layer**  
**3) Data layer**

**1) Client layer:**

It is also called as *Presentation layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

**2) Business layer:**

In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

**3) Data layer:**

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.

[](https://www.softwaretestingclass.com/what-is-difference-between-two-tier-and-three-tier-architecture/three-tier-architecture/)Three-tier Architecture

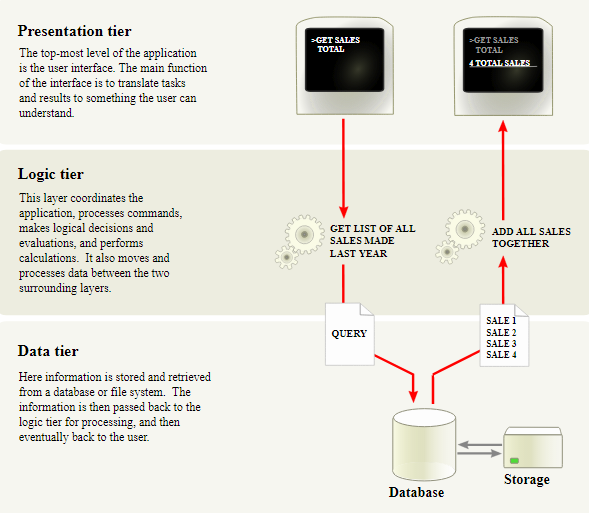
**Advantages**

1. High performance, lightweight persistent objects
2. Scalability – Each tier can scale horizontally
3. Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
4. High degree of flexibility in deployment platform and configuration
5. Better Re-use
6. Improve Data Integrity
7. Improved Security – Client is not direct access to database.
8. Easy to maintain and modification is bit easy, won’t affect other modules
9. In three tier architecture application performance is good.

**Disadvantages**

1. Increase Complexity/Effort

This is the common Question asked in the Interview. Hope this article helped you understanding Two-Tier and Three-Tier Architecture with example.

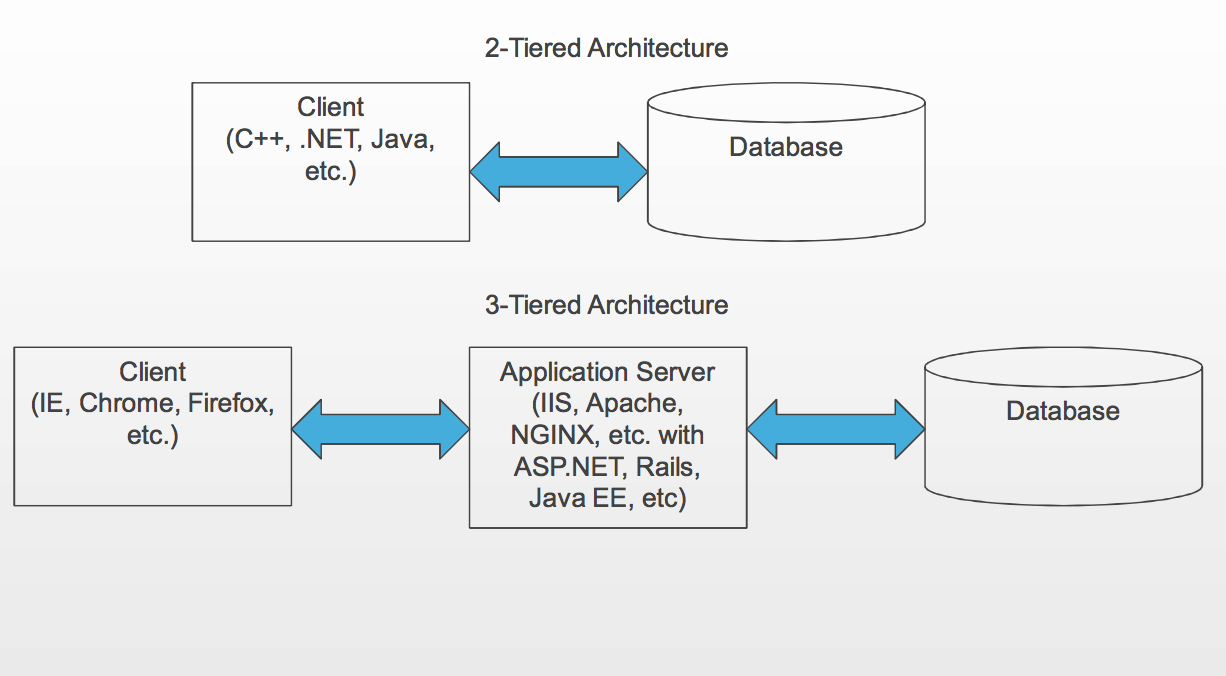


# 2-Tier vs. 3-Tier Application Architecture? Could the Winner be 2-Tier?

There is seemingly an infinite combination of choices to make when building a new application. One of the most fundamental choices is whether the application will use a 2 or 3-tier architecture. This choice really comes down to whether to split out the business and data logic into a separate tier. For web applications the decision is made for you. However, when building a stand-alone application there is a choice to be made.

The fundamental choice is whether the client application directly accesses the database, or indirectly accesses it by executing code on an intermediate application server. When selecting an architecture there are many things to consider, such as:  
• Security  
• Cost  
• Ease of development  
• Ease of maintenance  
• Performance  
• Deployment

Of course the relative importance of each consideration varies based on the application’s target uses and environment.

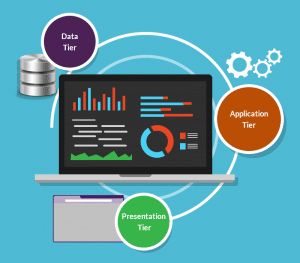


As can be seen from the following table, a 2-tier architecture has some compelling advantages. Although any of these points can be argued and some come down to subjective preference, there’s no arguing that 2-tier applications can be easier to build and cheaper and easier to deploy. However, when the Internet is involved, the performance of a 2-tiered application can suffer. Because more data needs to be transferred to the client in a 2-tiered architecture, the

|  |  |  |
| --- | --- | --- |
| **Design Consideration** | **Description** | **Winner** |
| Security | The topic of security is a tough one because arguments can be made either way for 2 vs. 3-tiers. In my experience, the 3-tier architecture has the potential for better security, but if great care isn’t taken during development it frequently ends up less secure. With a 2-tier approach, the application authenticates to and communicates directly to the database, which simplifies the attack surface area and is one less boundary to be concerned about. Depending on the details of the application server, there could be, for example, many complex REST APIs that could each have their own attack weaknesses. This one topic could be covered in a book, so I’m simply calling this one in favor of 2-tier even though direct access to the data sounds unsafe. | Even though 3-tier has the potential of greater security, I’m calling this one in favor of 2-tier. |
| Cost | 2-tier applications are generally easier to build and because of their lower complexity are less expensive all around. | 2-Tier |
| Ease of maintenance | Although part of cost consideration, ease of maintenance is worth analyzing separately. Keeping the client up to date in a 2-tier architecture can be a trouble point that is not an issue with most 3-tier designs. However, in a 3-tiered architecture, keeping the application tier up to date is much more difficult in the long run. For example, sometimes security patches required to keep the application tier secure are not compatible with your code and require you to make code changes, which can be extensive. | 2-Tier |
| Performance | Performance in a 2-tiered architecture is more sensitive to the hardware the client is running on, as well as the speed of the network connection. | Generally, 3-Tier |
| Deployment | In a 3-tiered architecture, typically users point their browsers to the application server to start using the application. However, the setup of that environment can be considerably more complex than setting up a database for the users to connect to and making the installer for the client available. | 3-tier for end-users; 2-tier from an initial configuration and setup perspective |
|  |  |  |

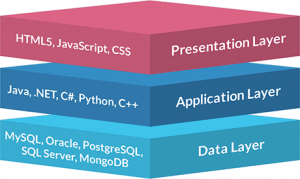
# 3-Tier Architecture: A Complete Overview

## **What is a 3-Tier Architecture?**

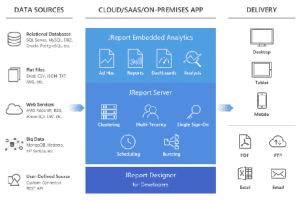
A 3-tier architecture is a type of software architecture which is composed of three “tiers” or “layers” of logical computing. They are often used in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. This added flexibility can improve overall time-to-market and decrease development cycle times by giving development teams the ability to replace or upgrade independent tiers without affecting the other parts of the system.

For example, the user interface of a web application could be redeveloped or modernized without affecting the underlying functional business and data access logic underneath. This architectural system is often ideal for embedding and integrating 3rd party software into an existing application. This integration flexibility also makes it ideal for embedding analytics software into pre-existing applications and is often used by embedded analytics vendors for this reason. 3-tier architectures are often used in cloud or on-premises based applications as well as in software-as-a-service (SaaS) applications.

## **What Do the 3 Tiers Mean?**



* **Presentation Tier-**The presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with others layers through API calls.
* **Application Tier-**The application tier contains the functional business logic which drives an application’s core capabilities. It’s often written in Java, .NET, C#, Python, C++, etc.
* **Data Tier-**The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

[](https://www.jinfonet.com/wp-content/uploads/2017/12/3-tier-architecture-jreport-1-1024x690.png)

Example of a 3-tier architecture: JReport.

The typical structure for a 3-tier architecture deployment would have the presentation tier deployed to a desktop, laptop, tablet or mobile device either via a web browser or a web-based application utilizing a web server. The underlying application tier is usually hosted on one or more application servers, but can also be hosted in the cloud, or on a dedicated workstation depending on the complexity and processing power needed by the application. And the data layer would normally comprise of one or more relational databases, big data sources, or other types of database systems hosted either on-premises or in the cloud.

A simple example of a 3-tier architecture in action would be logging into a media account such as Netflix and watching a video. You start by logging in either via the web or via a mobile application. Once you’ve logged in you might access a specific video through the Netflix interface which is the presentation tier used by you as an end user. Once you’ve selected a video that information is passed on to the application tier which will query the data tier to call the information or in this case a video back up to the presentation tier. This happens every time you access a video from most media sites.

## **What Are the Benefits of Using a 3-Layer Architecture?**

There are many benefits to using a 3-layer architecture including speed of development, scalability, performance, and availability.  As mentioned, modularizing different tiers of an application gives development teams the ability to develop and enhance a product with greater speed than developing a singular code base because a specific layer can be upgraded with minimal impact on the other layers.  It can also help improve development efficiency by allowing teams to focus on their core competencies. Many development teams have separate developers who specialize in front- end, server back-end, and data back-end development, by modularizing these parts of an application you no longer have to rely on full stack developers and can better utilize the specialties of each team.

Scalability is another great advantage of a 3-layer architecture. By separating out the different layers you can scale each independently depending on the need at any given time. For example, if you are receiving many web requests but not many requests which affect your application layer, you can scale your web servers without touching your application servers. Similarly, if you are receiving many large application requests from only a handful of web users, you can scale out your application and data layers to meet those requests without touch your web servers. This allows you to load balance each layer independently, improving overall performance with minimal resources. Additionally, the independence created from modularizing the different tiers gives you many deployment options. For example, you may choose to have your web servers hosted in a public or private cloud while you’re application and data layers may be hosted onsite. Or you may have your application and data layers hosted in the cloud while your web servers may be locally hosted, or any combination thereof.

By having disparate layers you can also increase reliability and availability by hosting different parts of your application on different servers and utilizing cached results. With a full stack system you have to worry about a server going down and greatly affecting performance throughout your entire system, but with a 3-layer application, the increased independence created when physically separating different parts of an application minimizes performance issues when a server goes down.