N-TIER ARCHITECTURE: TIER 2, TIER 3, AND MULTI-TIER EXPLAINED

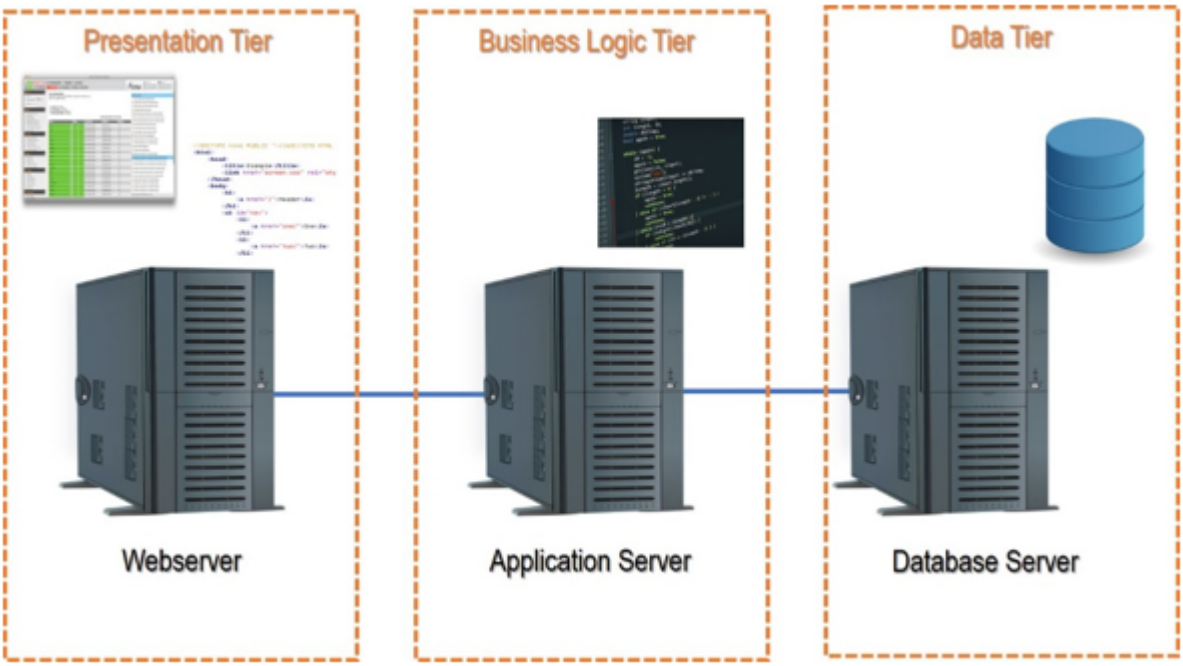


When business computing began to move from the mainframe to more affordable commodity machines, one would pick a given computer and "promote" it to server status by installing a database engine, some sort of code interpreter plus compiler, and develop software code that would then create the needed software tool. This meant that the user interface software (UI), the program itself, and the database would be running on the same platform (operating system and computer).

It was not long until the IT industry started realizing that things like distinct patching time frames from operating system and database engine manufacturers, recurrent common incidents, or even the need to update/upgrade components required urgent mitigation by having the processing and database tiers physically and logically split. Thus, **Tier 2 architecture** solutions started to be utilized.

As the internet became popular in the 1990s, it brought with it a revolution in terms of a user interface which was more performant and capable, but necessitated a specific web server. This widely empowered the proliferation of **Tier 3 architecture** in which the UI component was separated from the core computing and the database.

The graphic below shows how this plays out on the web:



What is N-Tier Architecture?

N-tier (or multi-tier) architecture refers to software that has its several layers rendered by distinct IT environments (tiers) under a client-server logic. The user interface (Presentation Tier) runs in a separate environment than the "computation" (Business Logic Tier) which in turn also runs in a distinct environment from the database engine and instances (Data Tier).

These distinct environments (Tiers) typically involve different servers, data center networks and often geographies.

Before moving further, it is important to clarify the difference between a "Tier" vs "Layer". A layer is a logic component within a software suite that accomplishes a given functionality, whereas a tier is the logical and hardware platform where such layer runs.

Most of the time it makes sense to split the above-mentioned tiers to achieve further architecture flexibility, synergy, security, and efficiency.

Consider Amazon.com as an example:

- Amazon needs to be accessible from anywhere, anytime and on any platform (PC; Tablet; Smartphone) so just on the Presentation Tier there are, in fact, several tiers with distinct "flavors" (Windows, iOS, Android).
- The Business Logic Tier itself comprehends not only servers distributed by several geographies processing data, but also automated input from warehouses and logistics components (tiers) that process data by themselves (distinct software layers) to convey information to the Amazon Business Logic Tier (so several tiers here also).
- Finally, the Data Tier is not only replicated and distributed but split as well into distinct levels of data instances (layers) in separate servers (tiers).

An N-Tier Architecture detailed example

Let's look at a concrete example from the automotive industry concerning logistics software that

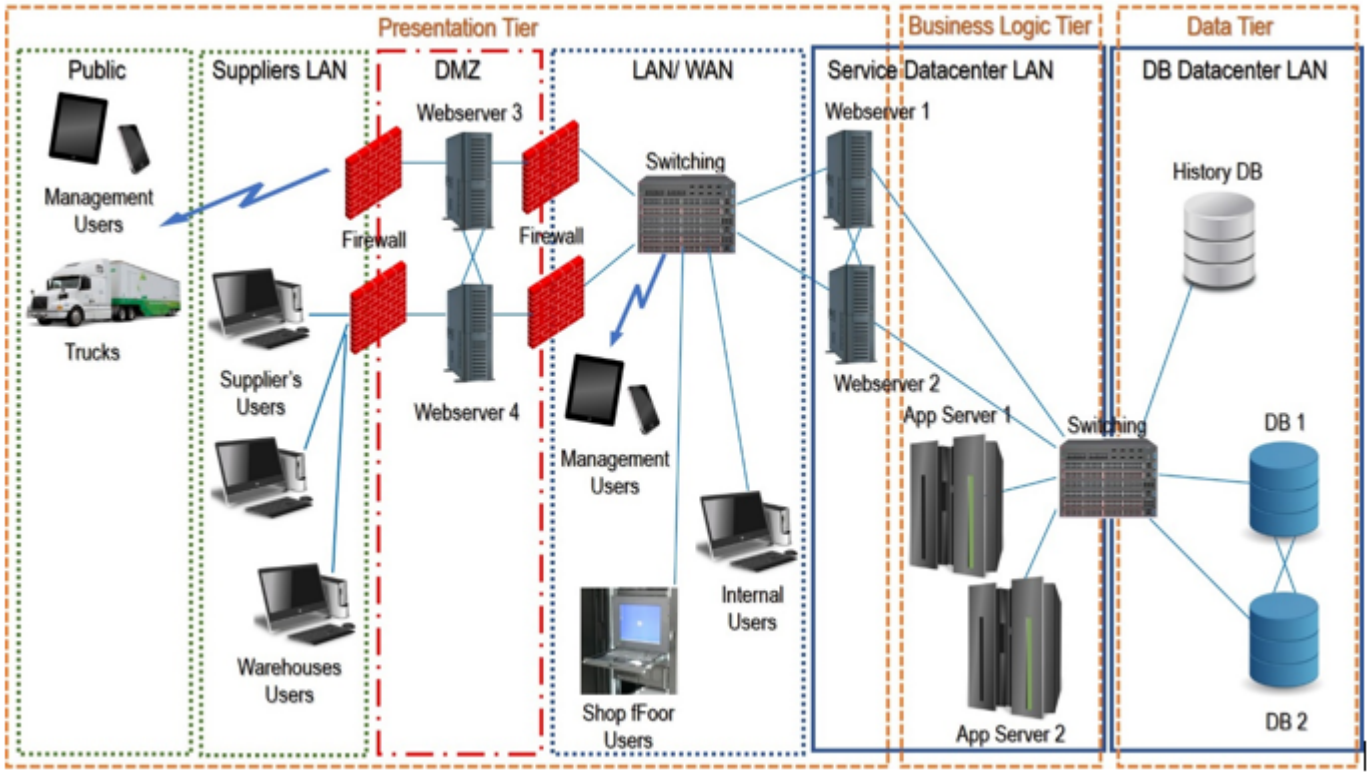
aims to serve the manufacturer's assembly line with split second up to date information on parts delivery status, hence needing to be fed by data from:

- The factory shop floor (assembly line) – did the parts arrive or not
- 3rd party parts supplier companies – were the parts produced
- Logistics warehouses – did the parts leave/arrive at the warehouse
- Trucks on the road – where are they, carrying which parts
- Logistics "control room" – where is everyone and everything
- Factory production forward planning area – which parts do we need, by when, to achieve optimized production cycles

To accomplish this, a system is required that can:

- Constantly collect information from several distinct sources (internal and external/ remote) on the status.
- Then process it and return valuable status information that would enable accurate assertive awareness to decision makers on how progress was being achieved regarding production forecast in a manner that those could act to mitigate any high potential noncompliance risks.
- Allow the forward planning team to issue parts production orders for the following days to comply with the establish production plan.

Below is the system architecture for the necessary software tiers:



Presentation Tier

The presentation tier must address several types of user interface protocols and platforms, in specific:

- The car manufacturer's internal users, from both the logistics control room and production planning areas where users could make changes to business rules and processes. This

requires a robust, secure UI (such as a desktop client application); additionally, key users on the shop floor need to query and check-in parts as they arrive. This would run from a client service software layer on a machine that would be physically within the manufacturer's network and served by the entire LAN/WAN infrastructure (assuring redundancy, added security, and speed).

- The 3rd party suppliers and logistics warehouse users reside on 3rd party entities' IT landscapes, and are therefore not allowed access to the manufacturer's LAN or WAN. Therefore, the most suitable, safe and fast solution would be to develop a web based client service software layer resorting to a web server cluster with one machine cluster inside the manufacturer's data center and another one in the DMZ (where authorized providers can access via strong authentication protocols). Both cluster nodes (internal and DMZ) should be installed in distinct geographies to allow redundancy and high availability.
- The trucks would need to be equipped with automated I/O clients that would convey the GPS coordinates and the material's list ID via a public mobile web data service through web servers installed on the DMZ via two ISPs from distinct countries to assure redundancy and high availability.
- Some special users, like area managers and plant directors, would need to get alerts and messages to access the system under a management reporting perspective via their smartphones or tablets through centralized web servers inside the manufacturer's network (two geographies for redundancy assurance) as well as on the go via the ISPs.

Business Logic Tier

The business logic tier would be constituted by several tiers of application server clusters distributed by multiple data centers in separate geographies with a production layer running on one server and a quality/tests layer running on another server in each data center.

Data Tier

The data tier would consist of an active database server cluster with multiple nodes (servers) each running its layer distributed by two data centers in separate geographies plus a historic data/ quick recovery tier at yet another distinct geographic location.

This example assumes the organization is responsible for the entire infrastructure - a cloud-based architecture with virtualized environments would provide even more flexibility, resilience and "Tiers" as well as "Layers" to the entire solution.

Advantages of N-Tier Architecture

- **Scalability** – having several separated components in the architecture allows easy scalability by upgrading one or more of those individual components. As an example, if the number of public clients grows that may require splitting the Webservice by adding new capacity to deal with the client demand which means more Web Servers on the Presentation Tier; another example would be an online shot that grows its product portfolio, in this case, the need to grow will most likely be in the Data Tier.
- **Enhanced Security** – An architecture that is distributed by several tiers allows placing those in distinct security zones (both logic as well as geographic) in order to assure that each

component is protected according to its core business criticality level. In the presented example, the web servers, although in a firewall-protected DMZ, have less need for security coverage than the application servers or database servers where all critical core business data is being processed or resides.
- **Resilience and Redundancy** – Critical components can easily be split in tiers that are clustered and geographically split to ensure failover, hence a more resilient system.
- **Maintenance flexibility** – As with the case of scalability, having distinct tiers allows pin pointed maintenance actions that do not produce collateral unwanted effects. This means that maintenance scheduling has fewer dependencies from 3rd party components.
- **Disaster Recovery** – An N-Tier Architecture allows individual components to restore in case of partial service disruption being it severe or not, hence allowing shorter service recovery times. In the case of robust, redundant well-distributed systems (the likes of ones supporting Google, Amazon, PayPal, other) the restore activities are not even noted by end users' due to the redundancy in place.
- **Developer Friendly Environment** – Having the several coding layers split by distinct tiers allows developers to focus on their individual task without having to share resources or bear in mind collateral potential impacts in each other's tasks/ domains. This is the type of architecture that also empowers frameworks and programming cultures like DevOps and Agile/Lean.

Disadvantages of N-Tier Architecture

- **Performance** – It seems a paradox that having the several components split to allow better efficiency and performance may result in the risk of lower performance. This risk basically pertains to 3rd party components/services. Having the architecture distributed by distinct geographies and tiers means that the entire system becomes highly dependent on the I/O flow. If within a closed network (a LAN/ WAN) the critical element is the entire cabling, switching and routing infrastructure, whereas in the case of using cloud-based infrastructure the topic additionally consists of several ISPs, who's individual communication infrastructure may not be synchronous in terms of performance.
- **Higher CAPEX and OPEX** – adding components to the architecture means the need for additional initial investment, running maintenance expenses budget as well as support services. This is particularly of concern when not relying on cloud-based services.

How N-Tier Has Progressed into The Cloud

Over the last few decades, IT systems have become one of the pillars of our global economy and key factors like security, high availability and flexibility have led most corporations to migrate their core business related IT systems to an N-Tier Architecture.

So, what was the problem? OPEX and CAPEX! Multiplying the number of tiers and software layers within a geographically distributed architecture to assure the above mentioned key factors was not cost effective for most companies, but then … the market responded with robust cloud computing offering.

Companies like IBM or AWS (Amazon Web Services) created large scale computing IT landscapes and companies could now progressively migrate their IT environments to those cloud landscapes

while paying merely a fee which represented a small percentage of the cost of implementing and maintaining their own N-Tier Architecture based IT environments.