**What is Scalability?**

This lesson introduces scalability.

**We'll cover the following**

* + [What is scalability?](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#What-is-scalability?)
  + [What is latency?](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#What-is-latency?)
  + [Measuring latency](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Measuring-latency)
    - [Network latency](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Network-latency)
    - [Application latency](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Application-latency)
  + [Why is low latency so crucial for online services?](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Why-is-low-latency-so-crucial-for-online-services?)

I am pretty sure, being in the software development universe, you’ve come across the word *scalability* numerous times. What is it? Why is it so important? Why is everyone talking about it? What are your plans or contingencies to scale when your app or the platform experiences significant traffic growth?

This chapter is a deep dive into *scalability*. It covers all the frequently asked questions about it, including what does *scalability* mean in the context of web applications and distributed systems?

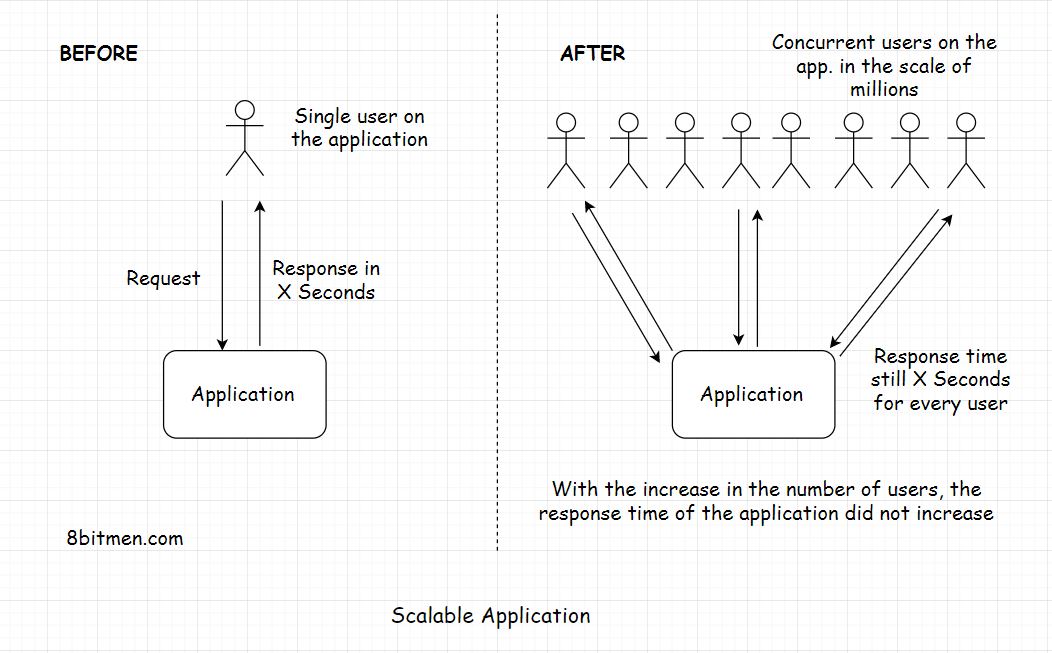
So, without further ado. Let’s get started.

**What is scalability?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#What-is-scalability?)

*Scalability* means the application’s ability to handle and withstand increased workload without sacrificing performance.

For example, if your app takes *x* seconds to respond to a user request. It should take the same *x* seconds to respond to each of your app’s million concurrent user requests.

The app’s back-end infrastructure should not crumble under a load of a million concurrent requests. It should scale well when subjected to a heavy traffic load and maintain the system’s latency.



**What is latency?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#What-is-latency?)

*Latency* is the time a system takes to respond to a user request. Let’s say you send a request to an app to fetch an image and the system takes *2* seconds to respond to your request. The latency of the system is *2* seconds.

Minimum latency is what efficient software systems strive for. No matter how much the traffic load on a system builds up, the latency should not go up. This is what *scalability* is.

If the latency remains the same, we can say that the application scaled well with the increased load and is highly scalable.

Let’s see *scalability* in terms of *Big-O* notation. Ideally, the complexity of a system or an algorithm should be *O(1)* which is constant time like in a *map* or a *key-value* database.

A program with the complexity of *O(n^2)* where *n* is the size of the data set is not scalable. As the size of the data set increases, the system will need more computational power and other resources to process the tasks.

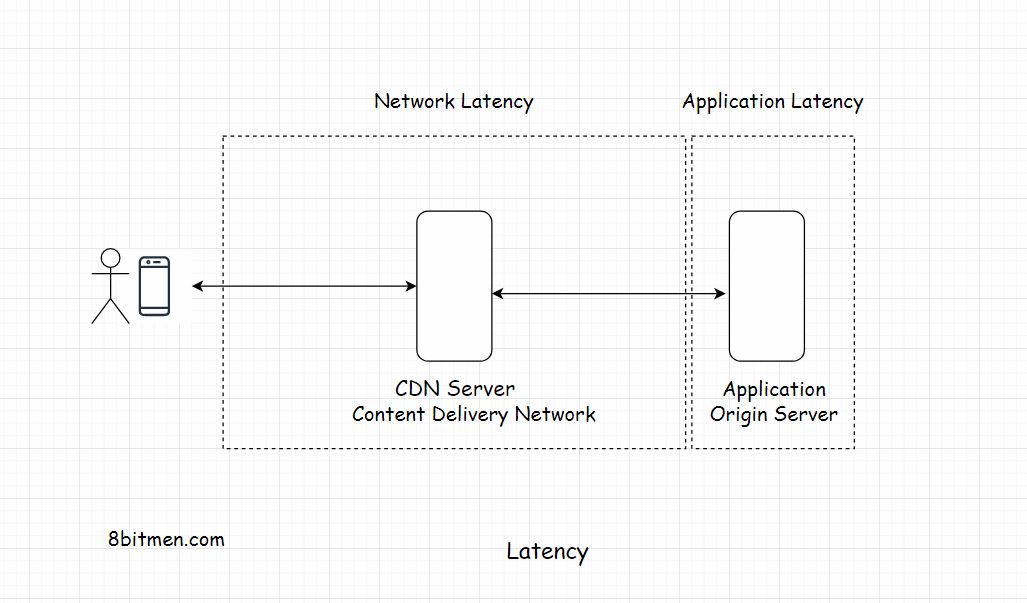
So, how do we measure latency?

**Measuring latency**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Measuring-latency)

*Latency* is measured as the time difference between the action that a user takes on the website and the system’s response in reaction to that action. The action can be an event like clicking a button, scrolling down a web page, etc.

This latency is generally divided into two parts:

* Network latency
* Application latency



**Network latency**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Network-latency)

*Network latency* is the time that the network takes to send a data packet from point *A* to point *B*. The network should be efficient enough to handle the increased traffic load on the website. To cut down the network latency, businesses use a *CDN* (Content Delivery Network) to deploy their servers across the globe as close to the end-user as possible. These close to the user locations are also known as *Edge* locations.

If you wish to understand the *Edge* locations and how apps are deployed in the cloud. Check out my [*cloud computing 101* course](https://www.educative.io/courses/cloud-computing-101-master-the-fundamentals) on this platform.

After having spent a decade in the industry writing code, I firmly believe that every software engineer should have knowledge of cloud computing. It’s the present and the future of application development and deployment.

Moving on.

**Application latency**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Application-latency)

*Application latency* is the time the application takes to process a user request. There are more than a few ways to cut down the application latency. The first step is to run *stress* and *load tests* on the application and scan for the *bottlenecks* that slow down the system as a whole. I’ll talk more about it in the upcoming lessons.

**Why is low latency so crucial for online services?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N02LPnROXmv#Why-is-low-latency-so-crucial-for-online-services?)

*Latency* plays a significant role in determining if an online business wins or loses a customer. Nobody likes to wait for a response on a website. There is a well-known saying, “*If you want to test a person’s patience, give them a slow internet connection.*”

If the visitor gets the response within a stipulated time, great otherwise, they’ll bounce off to another website. There is ample market research that concludes high latency in applications is a big factor in customers bouncing off a website. If there is money involved, *zero latency* is what businesses want. If only if this was possible.

Think of *massive multiplayer online* (MMO) games. A slight lag in an in-game event ruins the whole experience. A gamer with a high latency internet connection will have a slow response time despite having the best reaction time of all the players in an arena.

Algorithmic trading services need to process events within milliseconds. Fintech companies have dedicated networks to run low latency trading. The regular network just won’t cut it.

We can realize the importance of low latency by the fact that in *2011* *Huawei* and *Hibernia Atlantic* started laying a fiber-optic link cable across the *Atlantic Ocean* between *London* and *New York*. This property was estimated to cost approximately *$300M* just to save traders *six milliseconds* of latency.

**Primary Bottlenecks That Hurt the Scalability of our Application**

**We'll cover the following**

* + [Database](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Database)
  + [Application design](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Application-design)
  + [Not using caching in the application wisely](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Not-using-caching-in-the-application-wisely)
  + [Inefficient configuration and setup of load balancers](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Inefficient-configuration-and-setup-of-load-balancers)
  + [Adding business logic to the database](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Adding-business-logic-to-the-database)
  + [Not picking the right database](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Not-picking-the-right-database)
  + [At the code level](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#At-the-code-level)

There are several points in a web application that can become a *bottleneck* and hurt the *scalability* of our application. Let’s take a look at them.

**Database**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Database)

Imagine we have an application that appears to be well architected. Everything looks good. The workload runs on multiple nodes, and it can scale horizontally.

However, the *database* is a poor single monolith, where just one server has the onus of handling the data requests from all the server nodes of the workload.

This scenario is a bottleneck. The server nodes work well, handle millions of requests at a point in time efficiently, yet, the response time of these requests and the latency of the application are abysmal due to the presence of a single database. There is only so much it can handle.

Just like workload scalability, the database needs to be scaled well.

Make wise use of *database partitioning, sharding* with multiple database servers to make your system efficient.

**Application design**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Application-design)

A poorly designed application’s architecture can become a major bottleneck as a whole.

A typical architectural mistake is not using *asynchronous processes* and modules wherever required; rather, all the processes are scheduled sequentially.

For example, if a user uploads a document on the portal, tasks such as sending a confirmation email to the user, sending a notification to all subscribers/listeners to the upload event should be done asynchronously.

Tasks like these should be forwarded to a *messaging server* or a *task queue* for asynchronous processing as opposed to being processed sequentially, making the user wait.

**Not using caching in the application wisely**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Not-using-caching-in-the-application-wisely)

*Caching* can be deployed at several layers of the application. It speeds up the response time by notches. A *cache* cuts down the overall load on the app, intercepting all the requests before they hit the origin servers.

We should use *caching* exhaustively throughout the application to speed up things significantly.

If the system has a lot of static data, caching can bring down the deployment costs significantly. I’ve written an article on my blog: [*How PolyHaven manages 5 million page views and 80TB traffic a month for less than 400 USD*](https://www.scaleyourapp.com/application-hosting-how-polyhaven-manages-5-million-page-views-and-80tb-traffic-a-month-for-400/).

*Polyhaven* is a *3D* asset library with a large amount of static data. The article delineates how it leverages caching to bring down it’s deployment costs.

**Inefficient configuration and setup of load balancers**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Inefficient-configuration-and-setup-of-load-balancers)

*Load balancers* are the gateway to our application. Using too many or too few of them impacts the latency of our application. More on load balancers in the upcoming lessons.

**Adding business logic to the database**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Adding-business-logic-to-the-database)

No matter what justification anyone provides, I’ve never been a fan of adding *business logic* to the *database*.

The database is just not the place to put business logic. Business logic in the database makes the application components tightly coupled. Imagine how much code refactoring this would require when migrating to a different database. Also, the testing gets complex.

**Not picking the right database**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#Not-picking-the-right-database)

Picking the right database technology is vital for businesses. Need *transactions* and *strong consistency*? Pick a *relational database*. If you can do without *strong consistency* rather than need *horizontal scalability*, pick a *NoSQL* database.

Trying to pull things off with a not-so-suitable tech always has a profound impact on the latency of the entire application in negative ways. More on this in the upcoming lessons.

**At the code level**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQxW727jNBA#At-the-code-level)

This shouldn’t come as a surprise, but inefficient and poorly written code has the potential to bring down the entire service in production. This typically includes:

* Using unnecessary loops or nested loops
* Writing tightly coupled code
* Not paying attention to the *Big-O* complexity while writing the code. (be ready to do a lot of firefighting in production)

Ideally, we should always do a *DENTTAL* (Documentation, Exception Handling, Null pointers, Time complexity, Test coverage, Analysis of code complexity, Logging) check of our code when doing a dry run.

In this lesson, don’t worry if a few things are not clear to you, such as *strong consistency*, how the *message queue* facilitates *asynchronous behavior*, or how to pick the right database. I’ll discuss all that in the upcoming lessons. Stay tuned.

Moving on to the next lesson.

**How to Improve and Test the Scalability of our Application?**

In this lesson, we will cover how to improve and test the scalability of our application.

**We'll cover the following**

* + [Tuning the performance of the application – Enabling it to scale better](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Tuning-the-performance-of-the-application-%E2%80%93-Enabling-it-to-scale-better)
    - [Profiling](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Profiling)
    - [Caching](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Caching)
    - [CDN](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#CDN)
    - [Data compression](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Data-compression)
    - [Avoid unnecessary requests response cycles](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Avoid-unnecessary-requests-response-cycles)
  + [Testing the scalability of our application](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Testing-the-scalability-of-our-application)

Here are some of the standard strategies to fine-tune the performance of our web application. If the application is performance-optimized, it can withstand more traffic load with less resource consumption than an application that is not optimized for performance.

Now you might be wondering, “*Why are you talking about performance when you should be talking about scalability? Isn’t it what the lesson title says?*”

Well, the application’s *performance* is directly proportional to *scalability*. If an application is not performant, it will certainly not scale well.

These performance optimization strategies can be implemented even before the *pre-production* testing stage of the application.

Let’s see what they are.

**Tuning the performance of the application – Enabling it to scale better**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Tuning-the-performance-of-the-application-%E2%80%93-Enabling-it-to-scale-better)

**Profiling**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Profiling)

*Profile* the hell out of your app. Run *application profiler* and *code profiler*. See what processes are taking too long and are eating up too many resources. Find out the bottlenecks. Get rid of them.

Profiling is the dynamic analysis of our code. It helps us measure the *space* and the *time complexity* of our code and enables us to figure out issues like *concurrency errors, memory errors* and robustness and safety of the program. [This Wikipedia resource](https://en.wikipedia.org/wiki/List_of_performance_analysis_tools) contains a good list of performance analysis tools used in the industry.

**Caching**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Caching)

*Cache* wisely, and cache everywhere. Cache all the static content. Hit the database only when it is really required. Try to serve all the read requests from the cache. Use a write-through cache.

**CDN**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#CDN)

Use a *Content Delivery Network (CDN)*. Using a *CDN* further reduces the application’s latency due to the proximity of the data from the requesting user.

**Data compression**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Data-compression)

Compress data. Use apt compression algorithms to compress data and store data in compressed form. Since compressed data consumes less bandwidth, the data download on the client will be faster.

**Avoid unnecessary requests response cycles**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Avoid-unnecessary-requests-response-cycles)

Avoid unnecessary round trips between the client and server. Try to club multiple requests into one.

These are a few of the things we should bear in mind in the context of application performance.

**Testing the scalability of our application**[#](https://www.educative.io/module/lesson/web-application-architecture-101/39VNYBx7Z9r#Testing-the-scalability-of-our-application)

Once we are done with the essential performance testing of the application, it is time for capacity planning, provisioning the right amount of hardware—*compute* and *storage* power.

The right approach for testing the application for *scalability* largely depends on the design of our system. There is no standard formula for this.

Testing can be performed at both the hardware and the software level. Different services and components need to be tested—individually and collectively.

During the scalability testing, different system parameters are taken into account, such as:

* CPU usage
* Network bandwidth consumption
* Throughput
* Number of requests processed within a stipulated time
* Latency
* Memory usage of the program
* End-user experience when the system is under heavy load and so on.

In this testing phase, simulated traffic is routed to the system to study how the system behaves and scales under the heavy load. Contingencies are planned for unforeseen situations.

As per the anticipated traffic, the appropriate hardware and computational power are provisioned to handle the traffic smoothly with some buffer.

Several *load* and *stress* tests are run on the application. Tools like *JMeter* are pretty popular for running concurrent user tests on the application; if you are on the *Java* ecosystem. There are a lot of cloud-based testing tools available that help us simulate test scenarios just with a few mouse clicks.

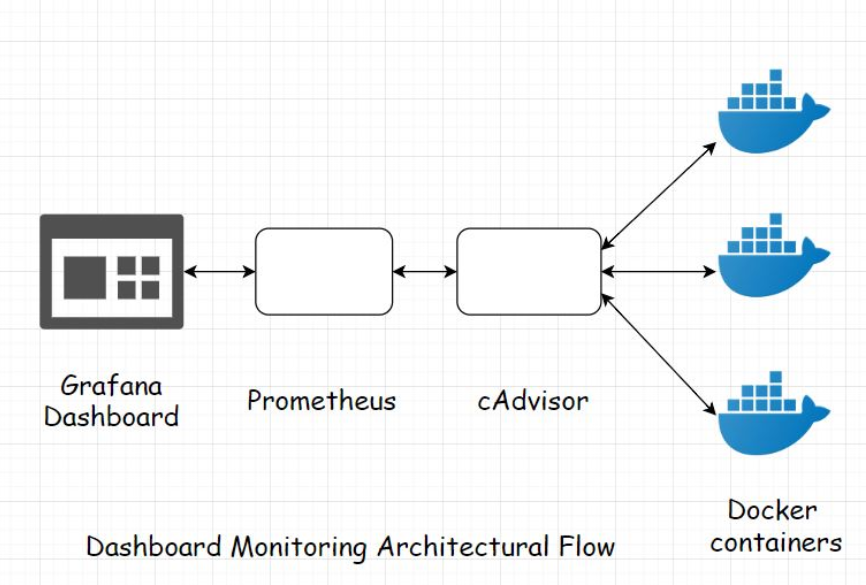
Businesses test for scalability all the time to get their systems ready to handle a traffic surge. If it’s a sports website, it prepares itself for the sports event day. If it’s an e-commerce website, it makes itself ready for festival season sale.

Here are a couple of good reads on the topic:

[*How production engineers support global events on Facebook*](https://engineering.fb.com/production-engineering/how-production-engineers-support-global-events-on-facebook/).

[*How Hotstar a video streaming service scaled with over 10 million concurrent users*](https://www.scaleyourapp.com/how-hotstar-scaled-with-10-3-million-concurrent-users-an-architectural-insight/).

In the industry, tech like *Cadvisor*, *Prometheus* and *Grafana* are pretty popular for tracking the system profile via web-based dashboards.



I’ve written an article if you want to read more about [pre-production monitoring](https://www.scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/).