**Event-Driven Architecture - Part 1**

In this lesson, which is part one of the event-driven architecture lessons, we will cover concepts like blocking and non-blocking.

**We'll cover the following**

* [What is blocking?](https://www.educative.io/module/lesson/web-application-architecture-101/YQGAjnm39L9#What-is-blocking?)
* [What is non-blocking?](https://www.educative.io/module/lesson/web-application-architecture-101/YQGAjnm39L9#What-is-non-blocking?)

When writing modern Web 2.0 applications, chances are you have come across terms like reactive programming, event-driven architecture and concepts like blocking and non-blocking.

What are they? Should you be aware of them?

You might have also noticed that tech and frameworks like NodeJS, Play, Tornado, and Akka are gaining more popularity in the developer circles for modern application development compared to traditional tech.

What is the reason behind this?

Is it just that we are bored of working on the traditional tech like Java, PHP, etc. and are attracted towards the shiny new stuff, or are there technical reasons behind this?

In this lesson, we will go through the related concepts step by step to realize the demands of modern software application development.

So, without further ado, let’s get on with it.

At this point in the course, we know what persistent connections are, what asynchronous behavior is and why we need them. We can’t really write real-time apps without implementing them.

Now, let’s find out what blocking is.

**What is blocking?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQGAjnm39L9#What-is-blocking?)

In web applications blocking means, the flow of code execution is paused while waiting for a process to complete. Until the process completes, it cannot move on.

Let’s say we have a block of code of ten lines within a function and every line triggers another external function executing a specific task.

Naturally, when the flow of execution enters the function, it will start executing the code from the top, from the first line. It will run the first line of code and will call the external function.

At this point, until the external function returns the response, the flow is blocked. The flow won’t move further. It just waits for the response unless we add asynchronous behavior to it by annotating it and moving the task to a separate thread.

However, that’s not what happens in regular scenarios, like in regular CRUD-based apps. Right?

This synchronous behavior of code execution is known as blocking since the flow of execution is blocked until the external function returns the response.

**What is non-blocking?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/YQGAjnm39L9#What-is-non-blocking?)

In the non-blocking approach, the flow doesn’t wait for the external function that is called to return the response. It just moves on to execute the following lines of code. This approach is a little inconsistent compared to the blocking approach since the external function might not return anything or throw an error. Still, the following code in the sequence is executed.

The non-blocking approach facilitates IO (Input-Output) intensive operations. IO intensive operations include the disk and other hardware-based operations, communication, network operations, etc.

We will continue this discussion in part two of the event-driven architecture lesson.

**Event-Driven Architecture - Part 2**

This lesson is part two of the discussion on event-driven architecture.

**We'll cover the following**

* [What are events?](https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r#What-are-events?)
* [Event-driven architecture](https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r#Event-driven-architecture)
* [Technologies for implementing the event-driven architecture](https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r#Technologies-for-implementing-the-event-driven-architecture)

**What are events?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r#What-are-events?)

There are typically two kinds of processes in applications: CPU intensive and IO intensive. In the context of web applications, IO means events. A large number of IO operations mean a lot of events occurring over a period of time, and an event can be anything from a tweet to a click of a button, an HTTP request, an ingested message, a change in the value of a variable, and so on.

We know that web 2.0 real-time applications have a lot of events. For instance, there are a lot of request-response cycles between the client and the server, typically in an online game, messaging app and so on. Events happening too often is called a stream of events. And we know how streams are managed from the previous stream processing chapter.

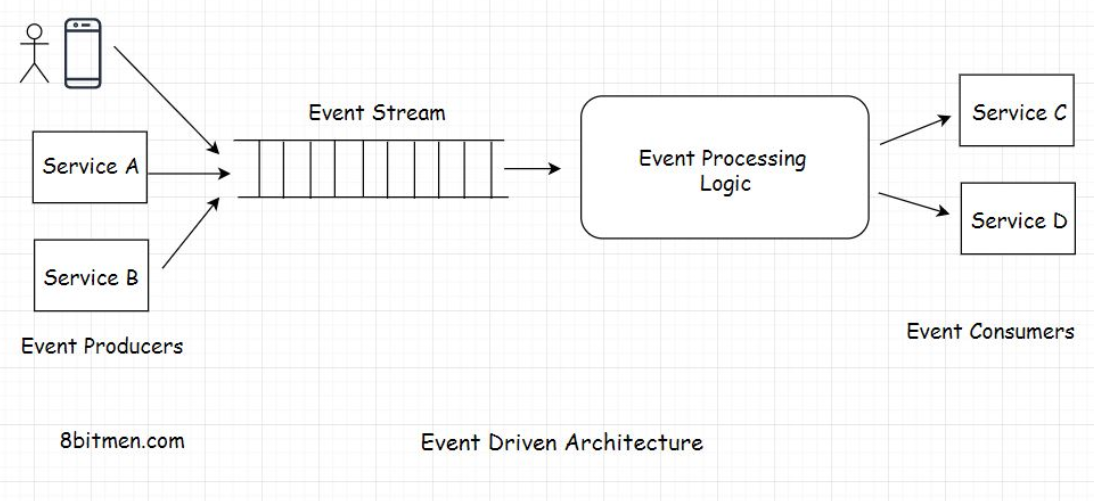
**Event-driven architecture**[#](https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r#Event-driven-architecture)

Non-blocking architecture is also known as reactive or event-driven architecture. Event-driven architectures are pretty popular in modern web application development.

Technologies like NodeJS, frameworks in the Java ecosystem like Play, Akka are non-blocking in nature and are built for modern high IO applications.

They are capable of handling a big number of concurrent connections with minimal resource consumption. Modern applications need a fully asynchronous model to scale. These modern web frameworks provide more reliable behavior in a distributed environment. They are built to run on a cluster, handle large-scale concurrent scenarios, and tackle problems that generally occur in a clustered environment. They enable us to write code without worrying about handling multi-threads, thread lock, out-of-memory issues due to high IO, etc.

Reactive architecture simply means reacting to the events occurring regularly. The code is written to react to the events. And to react to the events, the system has to continually monitor the stream. Event-driven architecture is all about processing asynchronous data streams. The application becomes inherently asynchronous.



## Technologies for implementing the event-driven architecture[#](https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r#Technologies-for-implementing-the-event-driven-architecture)

With the advent of Web 2.0, people in the tech industry felt the need to evolve the technologies to be powerful enough to implement modern web application use cases. Spring Framework added the Spring Reactor module to the core Spring repo. Developers wrote NodeJS, Akka, Play, etc.

You would have figured that reactive, event-driven applications are challenging to implement with thread-based frameworks because when trying to write non-blocking code with threads, we have to deal with shared mutable state and locks and whatnot that make things pretty complex.

In an event-driven system, everything is treated as a stream. The level of abstraction is good, and developers don’t have to worry about managing the low-level memory stuff.

Typical event streaming use cases that apply here are: handling a large number of transactional events, a timeline of changing stock prices, user events on an online shopping application, etc.

NodeJS is a single-threaded non-blocking framework written to handle more IO-intensive tasks. It has an event loop architecture. More on [NodeJS event loop architecture here](https://nodejs.org/fa/docs/guides/event-loop-timers-and-nexttick/).

[LinkedIn uses the Play framework](https://www.scaleyourapp.com/linkedin-real-time-architecture-how-does-linkedin-identify-its-users-online/) for identifying the online status of its users.

Now the emergence of non-blocking tech does not mean that traditional tech becomes obsolete and every app has to have an asynchronous non-blocking model. Again, every tech has its use cases and trade-offs.

NodeJS is not fit for CPU-intensive tasks. CPU-intensive operations are operations that require a good amount of computational power like for graphics rendering, running ML algorithms, handling data in enterprise systems, etc. It would be a mistake to pick NodeJS for these purposes.

In the upcoming lessons, I will discuss the general guidelines to consider when picking the fitting server-side technology. Let’s move on to the next lesson.

# Webhooks

In this lesson, we’ll learn about the need for webhooks and how they work.

**We'll cover the following**

* [What are Webhooks?](https://www.educative.io/module/lesson/web-application-architecture-101/gkrpv0O2jE6#What-are-Webhooks?)
* [How do webhooks work?](https://www.educative.io/module/lesson/web-application-architecture-101/gkrpv0O2jE6#How-do-webhooks-work?)

## What are Webhooks?[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkrpv0O2jE6#What-are-Webhooks?)

Imagine we’ve written an API that provides information on the latest, most exclusive Baseball events. Now our API is consumed by a lot of third-party services that fetch the information from the API, add their own flavor to it, and present it to their users.

However, so many API requests every now and then, just to check if a particular event has occurred, is crushing our server. The server can hardly keep up with the requests. There is no way for consumers to know that the new information isn’t available on the server yet or that an event hasn’t occurred yet. They just keep polling the API.

This will pile up the unwanted load on the server and can eventually bring it down.

What do we do? Is there a way we can cut down the load on our servers?

Yes!! Webhooks.

Webhooks are more like call-backs. It’s like, “I will call you when new information is available. You carry on with your work.”

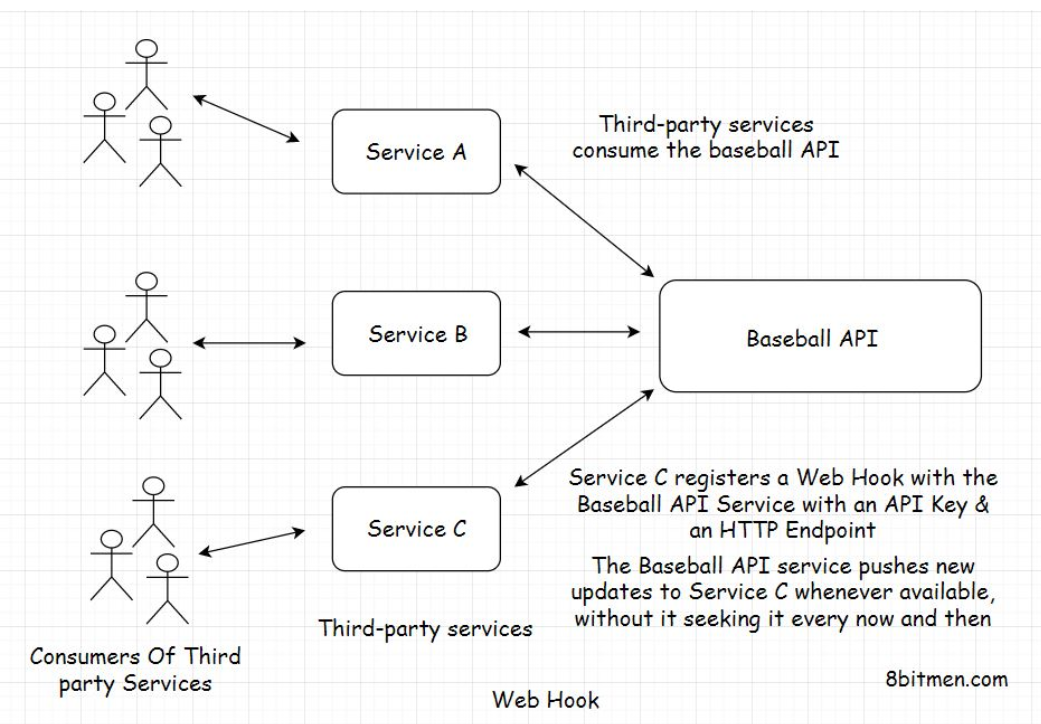
Webhooks enable communication between two services without the middleware. They have an event-based mechanism.

Okay! So, how do they work?

## How do webhooks work?[#](https://www.educative.io/module/lesson/web-application-architecture-101/gkrpv0O2jE6#How-do-webhooks-work?)

To use the Webhooks, consumers register an HTTP endpoint with the service with a unique API Key. It’s like a phone number. Call me on this number when an event occurs. I won’t be calling you anymore.

Whenever new information is available on the backend, the server fires an HTTP event to all the registered endpoints of the consumers, notifying them of the latest update.



Browser notifications are one example of Webhooks. Instead of visiting the websites every now and then for new info, they notify us when they publish new content.

# Shared-Nothing Architecture

In this lesson, we will briefly discuss the shared-nothing architecture.

When working with distributed systems, you’ll often hear the term shared-nothing architecture. So, I thought I’d just quickly bring it up in this lesson. Though, there is nothing really unique about this. I’ve already talked about it in the course.

When several modules work in conjunction, they often share RAM, which is also known as the shared memory. They share the disk, that is, sharing the database, and then they share nothing. The system’s architecture where the modules or services share nothing is called the shared-nothing architecture.

Shared-nothing architecture means eliminating all single points of failure. Every module has its own memory and disk. So, even if several modules in the system go down, the other modules online stay unaffected. It helps with scalability and performance.

Moving on!! Let’s discuss hexagonal architecture in the next lesson.

# Hexagonal Architecture

In this lesson, you will learn about hexagonal architecture.

**We'll cover the following**

* [What is hexagonal architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/m2jMX7NZQ1E#What-is-hexagonal-architecture?)
* [Real-world code implementation](https://www.educative.io/module/lesson/web-application-architecture-101/m2jMX7NZQ1E#Real-world-code-implementation)

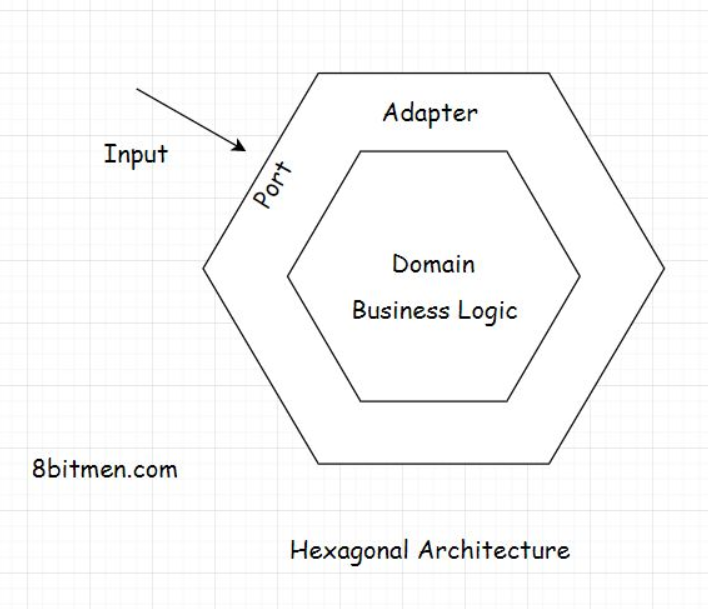
## What is hexagonal architecture?[#](https://www.educative.io/module/lesson/web-application-architecture-101/m2jMX7NZQ1E#What-is-hexagonal-architecture?)

The hexagonal architecture consists of three key components:

* Ports
* Adapters
* Domain

The focus of this architecture is to make components of an application: independent, loosely coupled, and easy to test.

The application should be designed in a way such that it can be tested by both humans and automated tests, with and without a UI, with mock databases and mock middleware, without making any changes or adjustments to the code.



The architectural pattern holds the domain at its core, which is business logic. On the outside, the outer layer has ports and adapters. Ports act like an API as an interface. All the input to the app goes through the interface.

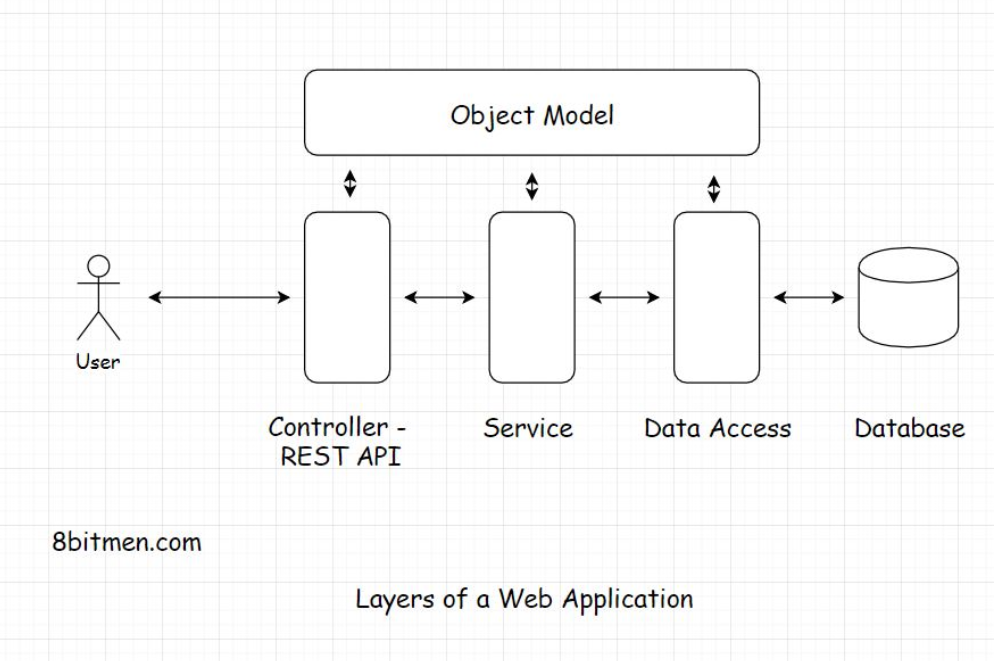
The external entities don’t have any direct interaction with the domain, the business logic. The adapter is the implementation of the interface. Adapters convert the data obtained from the ports to be processed by the business logic. The business logic lies isolated at the center, and all the IO input-output is at the edges of the structure.

The hexagonal shape of the structure doesn’t have anything to do with the pattern. It’s just a visual representation of the architecture. Initially, the architecture was called the ports and the adapter pattern. Later, the name hexagonal stuck.

The ports and the adapter analogy comes from computer ports as they act as the interface to the external devices for the input to the system. And the adapter converts the signals obtained from the ports to be processed by the chips inside.

## Real-world code implementation[#](https://www.educative.io/module/lesson/web-application-architecture-101/m2jMX7NZQ1E#Real-world-code-implementation)

Zeroing in on the real-world code implementation, isn’t this what we already do with the layered architecture approach? We have different layers in our applications. We have the controller, then the service layer interface, the class implementations of the interface, the business logic that goes in the domain model, and a bit in the service, business, and repository classes.



Well, the hexagonal approach is an evolved layered architecture. There is not much difference. There is this one issue with the layered approach. Developers often end up creating too many layers besides the standard controller, service, data access, and business layers in large repos.

This makes the business logic scatter across multiple layers making testing, refactoring, and plugging new entities difficult. Remember the stored procedures in the databases and the business logic coupled with the UI in Java Server Pages I discussed before?

When working with JSPs and stored procedures, despite having a layered architecture, the business logic is spread across the application right from the UI to the database making the code tightly coupled.

On the contrary, the hexagonal pattern makes its stance pretty clear: there is an inside component, which holds the business logic, then the outside layer, and the ports and the adapters, which involve the databases, message queues, APIs, and everything.

# Peer-to-Peer Architecture – Part 1

In this lesson, which is part one of the discussion on peer-to-peer architecture, we will take a deep dive into it.

**We'll cover the following**

* [What is a peer-to-peer network?](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#What-is-a-peer-to-peer-network?)
* [What does a central server mean?](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#What-does-a-central-server-mean?)
* [Downsides of centralized systems](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#Downsides-of-centralized-systems)
* [What is a decentralized architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#What-is-a-decentralized-architecture?)
* [Advantages of a peer-to-peer network](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#Advantages-of-a-peer-to-peer-network)

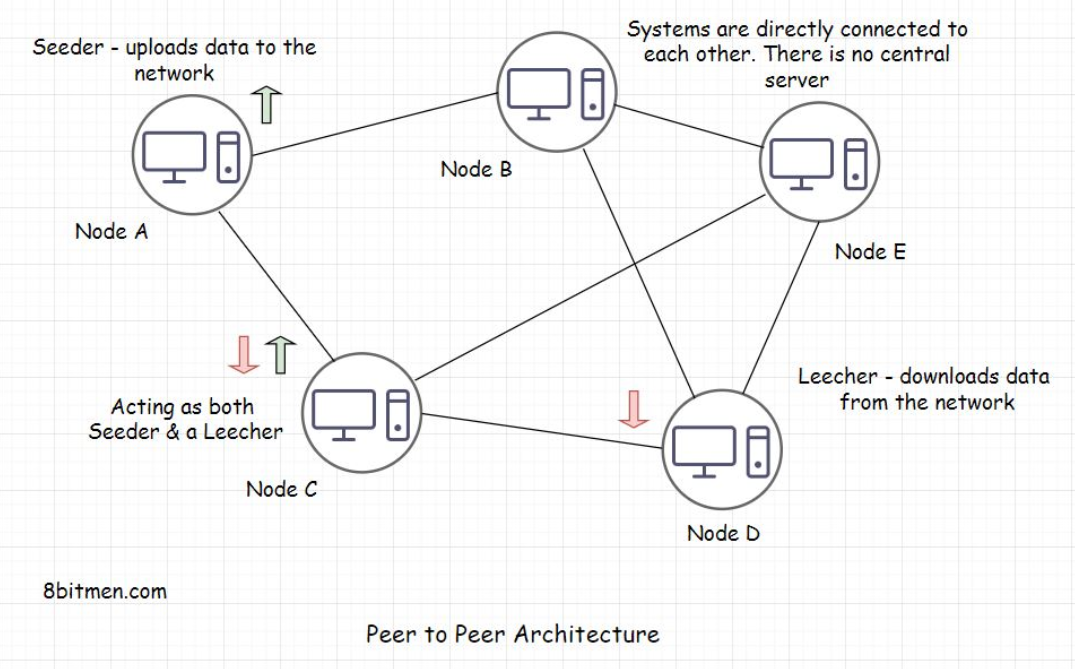
Peer-to-peer (P2P) architecture is the base of blockchain tech. We’ve all used it at some point in our lives to download files via torrent. So, I guess you have a little idea of what it is. You are also probably familiar with terms like seeding, leeching, etc. Even if you aren’t, you’ll learn everything in this lesson.

Let’s begin the lesson with a discussion on what a P2P network is.

## What is a peer-to-peer network?[#](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#What-is-a-peer-to-peer-network?)

A P2P network is a network in which computers, also known as nodes, can communicate with each other without a central server. The absence of a central server rules out the possibility of a single point of failure. All the computers in the network have equal rights. A node acts as a seeder and a leecher at the same time. So, even if some of the computers/nodes go down, the network and the communication are still up.

A seeder is a node that hosts the data on its system and provides bandwidth to upload the data to the network, and a leecher is a node that downloads the data from the network.



**What does a central server mean?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#What-does-a-central-server-mean?)

Think of a messaging app. When two users communicate, the first user sends a message from their device, and the message moves on the server of the organization hosting the messaging service. From there, the message is routed to the destination, that is, the device of the user receiving the message.

The server of the organization is the central server. These systems are also known as centralized systems.

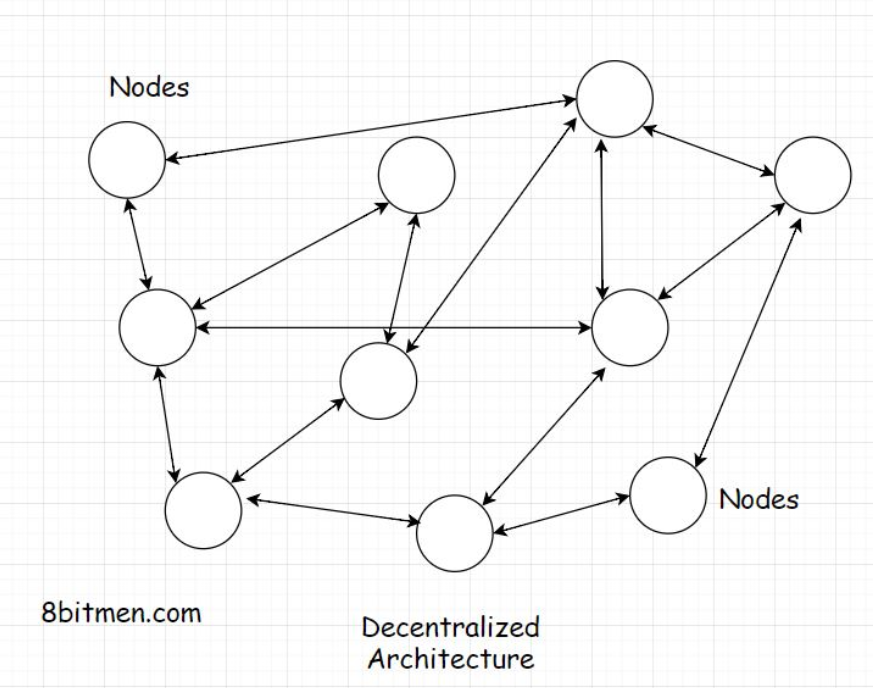
Now you might think, Okay, so, what’s the issue when communicating with my friend via a central server? I have never faced any issues that I can recall.

**Downsides of centralized systems**[#](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#Downsides-of-centralized-systems)

Here are a few important things to consider:

* First, the central server has access to all your messages. The admin can read them, share them with their associates, laugh about them, and so on. The gist is that our communication is not really secure. Even though the businesses say that their entire message pipeline is encrypted and stuff, data breaches happen, governments get access to our data; also, it is sold to third parties for fat profits. Do you think these messaging apps are really secure? Should the national security or enterprise officials sitting at the top of the food chain use these central server messaging apps for communication?
* Second, with centralized systems, we are stranded in case of natural disasters, earthquakes, zombie attacks, massive infrastructural failures, or the organization going out of business. There is no way to communicate with our friends across the globe. Think about it.
* Third, let’s say you start creating content on social media. You have built a pretty solid following on it, you spend 100+ hours a week to put out the best content ever, and you have worked years to reach this point of success. Then one fine day, out of the blue, the organization pokes you and says, “Hey!! Good job, but, aaaaa… for some reason, which we can’t talk about, we have to let your data go. We just don’t like your content.” Shift + Del and whoosh… all your data disappears like a ghost when sprinkled holy water. What are you going to do next? If you are already a content creator or are active on social media, this happens all the time, and you know that.

Fortunately, P2P networks are resilient to all these scenarios due to their design. They have a decentralized architecture.



## What is a decentralized architecture?[#](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#What-is-a-decentralized-architecture?)

Nobody has control over your data, and nobody has the power to delete it because all the participating nodes in a P2P network have equal rights. During a zombie apocalypse, when the huge corporation servers are dead or on fire, we can still communicate with each other via a peer-to-peer connection.

Though I’ve nothing against any of the corporations :) They’ve made our lives really easy. It’s just I am making you aware of all the possible scenarios out there.

## Advantages of a peer-to-peer network[#](https://www.educative.io/module/lesson/web-application-architecture-101/B6m3XkgY59n#Advantages-of-a-peer-to-peer-network)

Here is another use case where a peer-to-peer network rocks!! Imagine you’ve finally returned home from a trekking tour after visiting all seven continents around the world. Things couldn’t seem more beautiful and emotionally satisfying.

You have documented the entire expedition with state-of-the-art cameras and equipment in super ultra HD 4K quality, which has stacked up the hard drive of your computer. You are super excited to share all the videos and photos of the tour with your friends.

But how do you really plan to share the data, which is several gigabytes, with your friends?

Facebook Messenger, WhatsApp?

Messengers have a memory limit, so they aren’t even an option. Well, you could upload all the stuff on the cloud and share the link with your folks, but hold on. Uploading that much data needs some serious storage space, which would mean some serious money. Would you be in the mood to spend anymore after such a long trip?

No problem. We can write all the files on a physical memory like a DVD or a portable hard drive and share them with our friends, right?

Well, yes, we can. However, physical memory has its costs, and writing files for every friend is time-consuming, expensive, and resource-intensive. I get it you are tired already. And, oh!! By the way, we have to courier the disks to our friends put up across the globe. We need to add additional courier expenses and be okay with the time it would take for the courier to be delivered.

We’ve got this, don’t you worry!! We’ll find out some way. So, now what options do we have remaining? Think about it.

Hey, why don’t we use peer-to-peer file sharing? That would be awesome. With P2P peer-to-peer file sharing, we can easily share all the content with friends with minimal costs and fuss.

Beautiful!!

We can use a P2P protocol like BitTorrent for it. BitTorrent is the most commonly used P2P protocol for distributing data and large electronic files over the internet. It has approx. 25 million concurrent users at any point in time.

So, we will create a torrent file of our data and share it with all our folks. They just have to put the torrent in their BitTorrent client and start downloading the files to their systems while hosting/seeding the files simultaneously for others to download.

So, these are a few use cases where a P2P network rocks. In the next lesson, which is the second part of the P2P architecture, we will take a deep dive into the architecture.

# Peer-to-Peer Architecture – Part 2

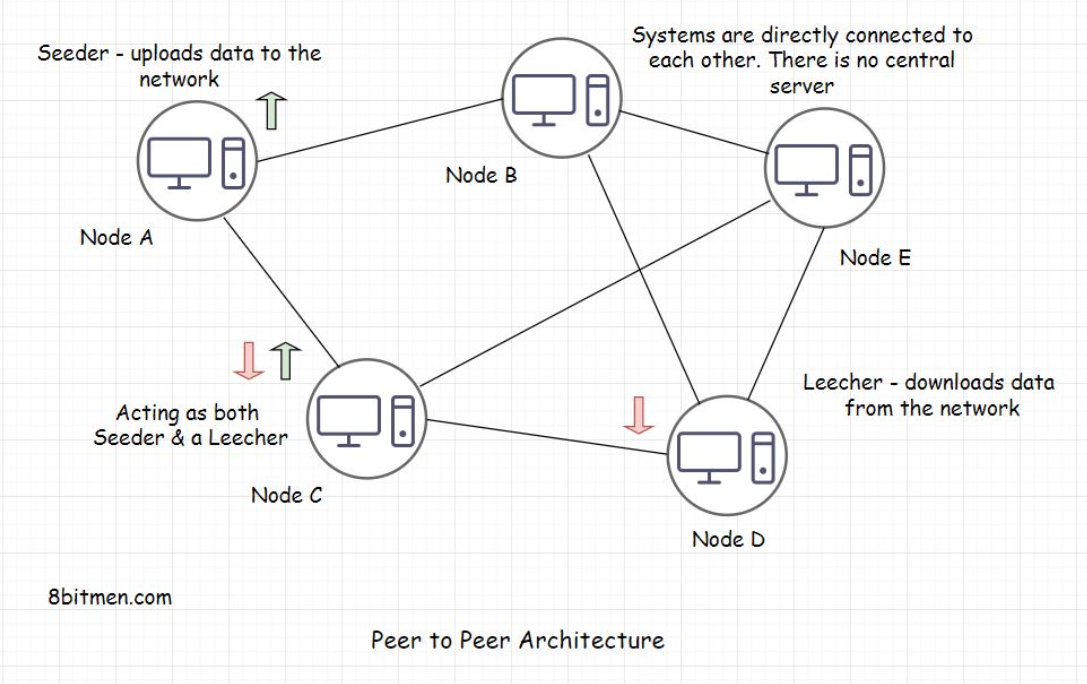
This lesson continues the discussion on peer-to-peer architecture.

**We'll cover the following**

* [What is a peer-to-peer architecture? How does it work?](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#What-is-a-peer-to-peer-architecture?-How-does-it-work?)
* [Types of P2P networks](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Types-of-P2P-networks)
  + [Unstructured network](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Unstructured-network)
  + [Structured network](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Structured-network)
  + [Hybrid model](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Hybrid-model)

## What is a peer-to-peer architecture? How does it work?[#](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#What-is-a-peer-to-peer-architecture?-How-does-it-work?)

A P2P architecture is designed around several nodes in the network, taking equal turns acting as both the client and the server.



The data is exchanged over TCP IP, just like it happens over the HTTP protocol in a client-server model. The P2P design has an overlay network over TCP IP, which enables the users to connect directly. It takes care of all the complexities and the heavy lifting. Nodes/peers are indexed and discoverable in this overlay network.

A large file is transferred between the nodes by being divided into chunks of equal size in a non-sequential order.

Say a system hosts a large file of 75 gigabytes. Other nodes in the network in need of the file locate the system containing the file. Then, they download the file in chunks, re-hosting the downloaded chunk simultaneously, making it more available to the other users. This approach is known as a segmented P2P file transfer.

Based on how these peers are linked with each other in the network, the networks are classified into a structured, unstructured, or a hybrid model.

## Types of P2P networks[#](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Types-of-P2P-networks)

### Unstructured network[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Unstructured-network)

In an unstructured network, nodes/peers keep connecting with each other randomly. So, there is no structure, no rule. Just simply connect and grow the network.

In this architectural design, there is no indexing of the nodes. To search the data, we have to scan through each and every node in the network. The search is O(n) in complexity, where n is the number of nodes in the network. This is pretty resource-intensive.

Think of it in this way. There are a billion systems connected in the network. Then, there is a file stored in just one system in the network. In an unstructured network, we have to run a search through each system in the network to find the file.

Let’s assume the search for a certain file in a system needs 1 second. The search through the entire network would require one billion seconds. This is abysmal from a low latency standpoint.

Some of the unstructured network’s protocols are Gossip, Kazaa, and Gnutella.

### Structured network[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Structured-network)

In contrast to an unstructured network, a structured P2P network holds the proper node indexing or the topology. This makes it easier to search for specific data.

This kind of network implements a distributed hash table to index the nodes. This index is just like the index of a book where we check to find a piece of information in the book rather than searching through every page.

BitTorrent is an example of this type of network.

### Hybrid model[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qZzpqn45Q9r#Hybrid-model)

The majority of blockchain startups have a hybrid model. A hybrid model means cherry-picking the good stuff from all the models like P2P, client-server, etc. It is a network involving both a peer-to-peer and a client-server model.

As we know, in a P2P network, one single entity doesn’t have all the control. So, to establish control, we need to set up our own server, a centralized server. For that, we need a client-server model.

A P2P network offers more availability. To take down a blockchain network, you have to take down all the network’s nodes across the globe. A P2P application can scale to the moon without putting the load on a single entity or the node. In an ideal environment, all the nodes in the network equally share the bandwidth and the storage space. The system scales automatically as new users use the app.

Nodes get added as more and more people interact with your data. There are zero-data storage and bandwidth costs, and you don’t have to shell out money to buy third-party servers to store your data. There is no third-party intervention, so data is secure. Share stuff only with friends you intend to share with.

The cult of the decentralized web is gaining ground in the present times. I can’t deny that this is a disruptive tech with immense potential. It has taken the financial sector, in particular, by storm. Cryptocurrency, NFTs, blockchain provenance are a few examples of this.

There are numerous P2P applications available on the web, for instance:

* Tradepal
* Peer-to-peer digital cryptocurrencies like Bitcoin and Peercoin.
* GitTorrent (a decentralized GitHub which uses BitTorrent and Bitcoin).
* Twister (a decentralized microblogging service that uses WebTorrent for media attachments).
* Diaspora (a decentralized social network implementing the federated architecture).

Federated architecture is an extension of the decentralized architecture used in decentralized social networks, which we will discuss next.

# Decentralized Social Networks

This lesson provides insight into decentralized social networks and their benefits.

**We'll cover the following**

* [What is a decentralized social network?](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#What-is-a-decentralized-social-network?)
* [What are the features of decentralized social networks?](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#What-are-the-features-of-decentralized-social-networks?)
  + [Bring your own data](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Bring-your-own-data)
  + [Ensuring the safety of our data](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Ensuring-the-safety-of-our-data)
  + [Economic Compensation to the parties involved in the network](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Economic-Compensation-to-the-parties-involved-in-the-network)
  + [Infrastructure ease](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Infrastructure-ease)

Before delving right into the federated architecture and its use in decentralized social networks, let’s take a quick look at what decentralized social networks are and why you should care about them? How different is a decentralized social network compared to a centralized social network?

Let’s get on with it.

## What is a decentralized social network?[#](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#What-is-a-decentralized-social-network?)

Simply put, decentralized social networks have servers spread out across the globe hosted by individuals like you and me. Nobody has autonomous control over the network. Everybody has an equal say.

Decentralized networks do not have to face any scalability issues as the scalability of a decentralized network is directly proportional to the number of users joining and active on the network.

We host our data from our systems as opposed to sending it to a third-party server. Nobody eavesdrops on our conversations or holds the right to modify our data at their whim.

You might have heard the term BYOD, Bring Your Own Device. Decentralized social networks ask you to Bring Your Own Data.

What does this really mean?

In these networks, the user data layer is separate, and it runs on protocols specifically designed for the decentralized web. The data formats and protocols are consistent across networks and apps.

So, if you want to bail out on a particular social network, you don’t lose your data; your data doesn’t just die. You can carry it with you and plug it into the app you sign up for next.

Cool, isn’t it?

There are decentralized social networks active on the web, such as Minds, Mastodon, Diaspora, Friendica, Sola, etc.

Let’s talk about some of the cool features decentralization offers.

## What are the features of decentralized social networks?[#](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#What-are-the-features-of-decentralized-social-networks?)

### Bring your own data[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Bring-your-own-data)

As I’ve brought up earlier, you can carry your data with you across a myriad of applications, and this is a unique feature that the blockchain economy leverages, especially in video games.

The in-game currency or content bought by the players, such as swords, powers, etc., can be carried forward and used in other games based on the decentralized protocol. Even if the game studios take the game offline, the in-game items still hold value. The purchased stuff stays with you.

### Ensuring the safety of our data[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Ensuring-the-safety-of-our-data)

No more private organizations eavesdropping on our data. We decide who we want to share our data with. The data is encrypted for everyone, including the network’s technical team. There’s no selling of our data for personal profits.

### Economic Compensation to the parties involved in the network[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Economic-Compensation-to-the-parties-involved-in-the-network)

Networks like Diaspora, Sola, and Friendica have come out with features that financially compensate all the parties involved in the network. Users get compensated for the awesome stuff they share online. People sharing their computing power to host the network get their compensation in the form of tokens, equity, or whatever, as per the economic policy of the network.

The teams involved in moderating the network and developers writing new features get compensated by enabling content-relevant ads on the network or by the token-based economy of the platform. It’s a win-win for all the parties.

### Infrastructure ease[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qANRw46ywO0#Infrastructure-ease)

A single entity does not have to bear the entire cost of the infrastructure since it is decentralized. The possibility of the network going down is almost zero.

An individual developer can build cool stuff without worrying about the server costs. The data, just like a blockchain ledger, is replicated across the nodes. So, even if a few nodes go down, our data is not lost.

These social networks are written on protocols and software that are open source so that the community can keep improving the code and keep building awesome features.

ActivityPub is one example of this. It’s an open decentralized social networking protocol. It provides an API for modifying and accessing the content on the network and for communicating with other pods in the federation.

I’ve added this lesson to give you an insight into decentralized web applications. What are they? How do they work? In the near future, these are going to consume a big chunk of the market share.

Decentralization in the Fintech industry is becoming the norm. It’s always good to stay ahead of the curve.

Now let’s take a look into federated architecture.

# Federated Architecture

In this lesson, we will discuss federated architecture.

**We'll cover the following**

* [What Is federated architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/gk1zpx0VB4l#What-Is-federated-architecture?)
* [How is federated architecture implemented in decentralized social networks?](https://www.educative.io/module/lesson/web-application-architecture-101/gk1zpx0VB4l#How-is-federated-architecture-implemented-in-decentralized-social-networks?)
* [What is the need for pods?](https://www.educative.io/module/lesson/web-application-architecture-101/gk1zpx0VB4l#What-is-the-need-for-pods?)

## What Is federated architecture?

Federated architecture is an extension of decentralized architecture. It powers social networks like Mastodon, Minds, Diaspora, etc.

The term federated in a general sense means a group of semi-autonomous entities that exchange information with each other. A real-world example of this is looking at different states in a country managed by the state governments. They are partially self-governing and exercise power to keep things running smoothly. Then, those state governments share information with each other and with a central government making a complete autonomous government.

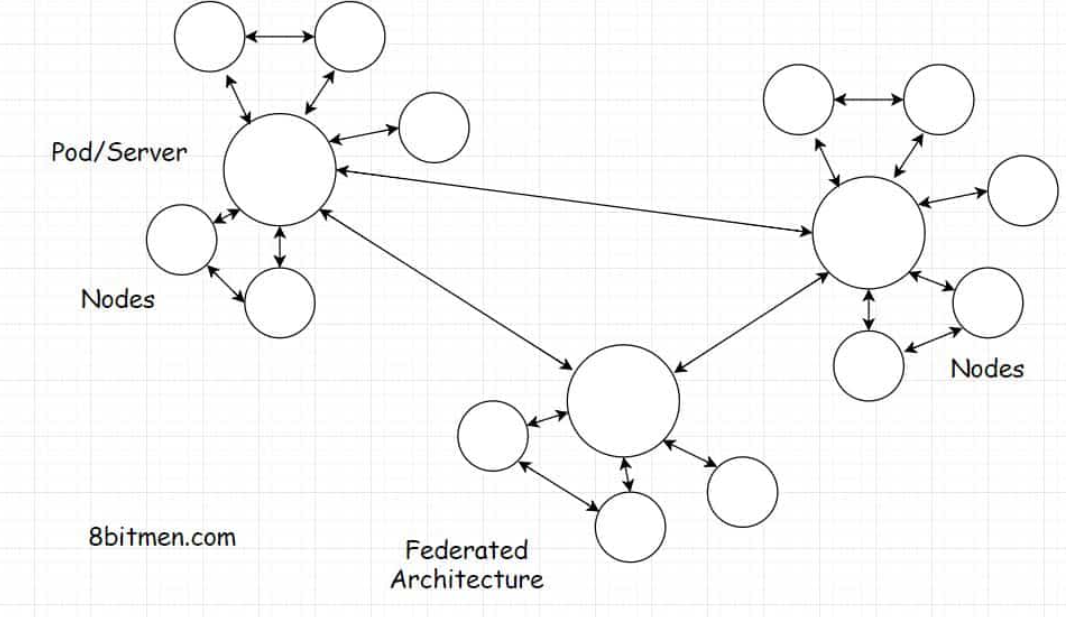
This is just an example. From a technical standpoint, the federated model is under continual research, development, and evolution. There are no standard rules. Developers and architects can have their own designs in place.

## How is federated architecture implemented in decentralized social networks?

As shown in the illustration below, a federated network has entities called servers or pods. A large number of nodes subscribe to the pods. There are several pods in the network that are linked to each other and share information.

The pods can be hosted by individuals and as new pods are hosted and introduced to the network, the network keeps growing.

In case the link between a few pods breaks temporarily, the network is still up. Nodes can still communicate with each other via the pods they are subscribed to.



What is the need for Pods? Can’t the nodes just be linked to each other like in a regular peer-to-peer network?

## What is the need for pods?

Pods facilitate node discovery. In a peer-to-peer network, there is no way of discovering other nodes, and we would just sit in the dark if it weren’t for a centralized node registry or something.

The other way is to run a scan through the network to discover other nodes. This is a time-consuming and tedious task. Why not just have a pod instead?

Well, I believe you now have a fundamental understanding of the p2p architecture and the decentralized web. Let’s move on to the next lesson, where we talk about picking the right server-side technology.

**What is the need for a peer-to-peer network? Which of the following option(s) are correct?**

###### A)All the participants/computers in the network can communicate with each other without the need for a central server. The absence of a central server rules out the possibility of a single point of failure.

###### B)All the computers in the network have equal rights. No entity has the central authority to debar another entity out of the network.

###### C)In a peer-to-peer network, no central entity has access to our data without our consent.

**What is a seeder in a peer-to-peer network?**

###### A)A seeder uploads data to the network. Peers providing files for other peers in the network to download are known as seeds.

**What is the need for a hybrid peer-to-peer model?**

###### B) A hybrid peer-to-peer model enables us to leverage the features of both the peer-to-peer and the client-server architecture.

**What is the need for a decentralized social network when the existing centralized social networks are so popular already?**

###### B) In a decentralized social network, nobody has autonomous control over the network, and everybody has an equal say.

###### C) In decentralized social networks, we host our data from our systems instead of sending it to a third-party server. Nobody can eavesdrop on our conversations or hold the rights to modify our data at their whim.

###### D)On decentralized social networks, users get compensated for the awesome stuff they share online.

**What is a pod in a federated architecture?**

###### A pod in a federated architecture acts like a server to which a large number of nodes are subscribed.