How to Pick the Right Server-Side Technology?

In this lesson, we'll learn how to pick the right server-side technology for our projects.

We'll cover the following

Before commencing the lesson, I would like to say that there is no rule of thumb that for a use case *X* you should always pick a technology *Y*.

Everything depends on our business requirements. Every use case has its unique needs. There is no perfect tech, everything as has its pros & cons. You can be as creative as you want. There is no rule that holds us back.

Alright, this being said. I have listed some of the general scenarios or I can say the common use cases in the world of application development and the fitting backend technology for those based on my development experience.

Real-time Data Interaction

If you are building an app that needs to interact with the backend server in real-time like stream data to & fro. For instance, a messaging application, a real-time browser-based massive multiplayer game, a real-time collaborative text editor or an audio-video streaming app like *Spotify*, *Netflix* etc.

You need a *persistent connection* between the client and server, also you need a *non-blocking* technology on the back-end. We've already talked about both the concepts in detail.

Some of the popular technologies which enable us to write these apps are *NodeJS*, *Python* has a framework called *Tornado*. If you are working in the *Java* Ecosystem

you can look into *Spring Reactor*, *Play*, *Akka.io*.

Once you start researching on these technologies, go through the architecture, concepts given in the developer docs. You'll gain further insights into how things work, what other tech and concepts I can leverage to implement my use case. One thing leads us to the other.

Uber used NodeJS to write their core trip execution engine. Using it they could easily manage a large number of concurrent connections.

Peer to Peer Web Application

If you intend to build a *peer to peer* web app, for instance, a *P2P* distributed search engine or a *P2P* Live TV radio service, something similar to *LiveStation* by *Microsoft*.

Look into *JavaScript*, protocols like *DAT*, *IPFS*. Checkout *FreedomJS*, it's a framework for building *P2P* web apps that work in the modern web browsers.

This is a good read Netflix researching on Peer to peer technology for streaming data.

CRUD-based Regular Application

If you have simple use cases such as a regular *CRUD-based* app like online movie booking portal, a tax filing app etc.

CRUD (Create Read Update Delete) is the most common form of web apps being built today by businesses. Be it an online booking portal, an app collecting user data or a social site etc. all have an *MVC (Model View Controller)* architecture on the backend.

Though the view part is tweaked a little with the rise of *UI* frameworks by *React*, *Angular*, *Vue* etc. The *Model View Controller* pattern stays.

Some of the popular technologies which help us implement these use cases are *Spring MVC*, *Python Django*, *Ruby on Rails*, *PHP Laravel*, *ASP .NET MVC*.

Simple, Small Scale Applications

If you intend to write an app which doesn't involve much complexity like a blog, a simple online form, simple apps which integrate with social media that run within

in the IFrame of the portal. This includes web browser-based strategy, airline, football manager kind of games. You can pick *PHP*.

PHP is ideally used in these kinds of scenarios. We can also consider other web frameworks like *Spring boot*, *Ruby on Rails*, which cut down the verbosity, configuration, development time by notches & facilitate rapid development. But *PHP* hosting will cost much less in comparison to hosting other technologies. It is ideal for very simple use cases.

CPU & Memory Intensive Applications

Do you need to run *CPU* Intensive, *Memory* Intensive, heavy computational tasks on the backend such as *Big Data Processing*, *Parallel Processing*, Running *Monitoring & Analytics* on quite a large amount of data?

Performance is critical in systems running tasks which are *CPU* & *Memory* intensive. Handling massive amounts of data has its costs. A system with high latency & memory consumption can blow up the economy of a tech company.

Also, regular web frameworks & scripting languages are not meant for number crunching.

Tech commonly used in the industry to write performant, scalable, distributed systems are –

C++, it has features that facilitate low-level memory manipulation. Provides more control over memory to the developers when writing distributed systems. Majority of the cryptocurrencies are written using this language.

Rust is a programming language similar to *C++*. It is built for high performance and safe concurrency. It's gaining a lot of popularity lately amongst the developer circles.

Java, *Scala* & *Erlang* are also a good pick. Most of the large scale enterprise systems are written in *Java*.

Elastic search an open-source real-time search and analytics engine is written in *Java*.

Erlang is a functional programming language with built-in support for concurrency, fault-tolerance & distribution. It facilitates the development of massively scalable systems. This is a good read on Erlang

Go is a programming language by *Google* to write apps for multi-core machines & handling a large amount of data.

Julia is a dynamically programmed language built for high performance & running computations & numerical analytics.

Well, this is pretty much it. In the next lesson, let's talk about a few key things which we should bear in mind when researching on picking a fitting technology stack for our project.