How to Pick the Right Server-Side Technology?

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

We'll cover the following

- Real-time data interaction
- Peer-to-peer web application
- CRUD-based regular application
- Simple, small scale applications
- CPU and memory intensive applications

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, and everything has its pros and 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 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#

Let's say you are building an app that needs to interact with the backend server in real-time to stream data to and fro. For instance, your app could be 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. You also need *non-blocking* technology on the backend. We've already talked about both the concepts in detail.

Some of the popular technologies that enable us to write these apps are *NodeJS* and *Python*, which has a framework called *Tornado*. If you are working in the *Java* Ecosystem you can look into *Spring Reactor*, *Play*, and *Akka.io (http://Akka.io)*.

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

Uber used NodeJS to write their core trip execution engine. (https://eng.uber.com/uber-tech-stack-part-two/) 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* and *IPFS*. Also, checkout *FreedomJS*, which is a framework for building

P2P web apps that work in modern web browsers.

This is a good read Netflix researching peer-to-peer technology for streaming data. (https://arstechnica.com/information-technology/2014/04/netflix-researching-large-scale-peer-to-peer-technology-for-streaming/)

CRUD-based regular application#

Simple use cases such for regular *CRUD-based* apps include an online movie booking portal, a tax filing app, etc.

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

Although 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 that help us implement these use cases are *Spring MVC*, *Python Django*, *Ruby on Rails*, *PHP Laravel*, and *ASP .NET MVC*.

Simple, small scale applications#

If you intend to write an app that doesn't involve much complexity like a blog or a simple online form or simple apps that integrate with social media running within the IFrame of the portal, including web browser-

based strategy, airline, and football manager kind of games, you can pick *PHP*.

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

CPU and memory intensive applications#

Do you need to run *CPU* intensive, *memory* intensive, or heavy computational tasks on the backend, such as *Big Data processing, parallel processing*, and running *monitoring and analytics*, on quite a large amount of data?

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

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

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

C++ has features that facilitate low-level memory manipulation. This provides more control over memory to the developers when writing distributed systems. The majority of the cryptocurrencies are written using this language.

using this language
(https://en.wikipedia.org/wiki/List_of_cryptocurrencies).

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

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

Elasticsearch (https://en.wikipedia.org/wiki/Elasticsearch) is 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, and distribution. It facilitates the development of massively scalable systems. This is a good read on Erlang (https://stackoverflow.com/questions/1636455/where-is-erlang-used-and-why)

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

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

Well, this is pretty much it. In the next lesson, we'll talk about a few key things to bear in mind while researching picking a technology stack for our project.

← Back

More on Architecture Quiz – Part 2

Next →

Key Things to Remember When Picki...

✓ Mark as Completed

Report an Issue