Building a Custom Tech Stack

CS-554 – WEB PROGRAMMING

What is a technology stack?

We already know that larger applications have many moving parts, however what we have yet to encounter is an application where the programmers develop in multiple languages!

A technology stack is the set of technologies chosen to create a technological solution. This solution is often quite a bit larger than is obvious to observers, especially one that solves complex cases.

It is **incredibly common** for larger software platforms (interconnected software applications) to be written across a number of different programming languages with a number of different technology stacks, maintained by entirely different development groups.

Tech Stacks we know so far

We have primarily worked with some very common tech stacks:

- Web 1 runs on Mongo, Express, Node (most of the MEAN) stack
- Web 2 has had us use Mongo, Express, React, Node (MERN)
- Web 2 has also had us add Redis into the mix

Other common stacks:

- XAMP (X for any operating system, Apache, MySQL, PHP)
- LAMP (Linux, Apache, MySQL, PHP)
- WAMP (Windows, Apache, MySQL, PHP)
- MEAN (Mongo, Express, Angular, Node)
- WISA (Windows, IIS, SQL Server, ASP.Net)

Why we use common tech stacks

The tech stacks in the previous slide all have something in mind: they all work together

- LAMP: Linux can install a LAMP stack in a few minutes; PHP has MySQL drivers, Apache has PHP drivers, and there is a great deal of support for each software to work with the others
- WISA: All the Windows server technologies were built to work with each other out of the box; they were literally designed by Microsoft to be easy to use with the same concepts, paradigms, and APIs. Accessing your SQL server from your ASP.Net application just involves you literally making an object with a connection string to your database.
- MEAN: Express is a Node Package with API design as a core tenet, meaning it is literally Node code; MongoDB has an official Node driver; Angular is just a JS library that works well with APIs. All the parts just plug into each other and play nicely.

Most technical stacks gain popularity because they work well together.

Adding Custom Technology

All technology has its limits, and at some point, you will hit the boundaries of what is **easy** to accomplish with just your basic stack.

- We have already added Redis in!

When we say easy, we also mean *easy to do well and efficiently*.

When this happens, we expand our technology stack with something outside the norm for our tech stack, and start building out a custom tech stack.

- Often, this means you start leaving the primary programming language.

Sometimes, we add in prebuilt technology. Sometimes, you will now be creating entirely different pieces of technology that work in conjunction with your application.

Useful Pieces of Technology

A BRIEF SEGUE

Other Technology

Ultimately, just about every programming language can accomplish whatever you need to accomplish given enough time and processing power. However, you very often don't want to reinvent the wheel.

- Hence why we use databases instead of writing to files for all our data storage…

There are many pieces of technology that you may find useful to know about at this point, for the purposes of your final project (and for your life as a developer in general!)

For each technology, there will be:

- A brief description of the technology
- The problem it solves
- How you would use it

Some of these technologies are solutions to hard problems; some are utilities you use for common tasks.

Apache Hadoop

Apache Hadoop is a technology used for distributed storage and computation. It is extremely useful for distributing huge data across many machines, and performing big data analytics across the data set.

Hadoop processes off the concept of *mapping* and *reducing*.

- Map takes the data stored in Hadoop and pulls it into a format needed for the operation you need to perform
- Reduce accumulates a result based on that data for the operation you want to perform.

An example of when you would use Hadoop:

- You store a series of analytic files in Hadoop from across your entire software platform.
- You have a Map Reduce Job to take those analytics and create reports so that the product team can see how people use your product.

Elasticsearch

Elasticsearch is a REST based search and analytics API for storing huge data and searching through it. It is also effectively an incredibly fast full text indexing database.

Searching is an inherently hard operation, in every single database. Elasticsearch is an advanced indexing platform that focuses on searching, rather than storage.

An example of when you would use Elasticsearch:

- You need to hundreds of gigabytes of text based articles and rapidly search for articles matching set keywords
- You would have Elasticsearch index the articles and define ways that you would search for keywords, and then simply query articles via the REST API.

ImageMagick

ImageMagick is a **very** powerful image manipulation utility.

It is a command line utility that essentially allows you to perform any sort of conversion imaginable on an image.

An example of when you would use ImageMagick :

- You have a website where users upload photos
- You would run images through ImageMagick to have them compressed, resized, cropped, etc by this highly efficient program rather than attempting to do this in the programming language of your web server.

LaTeX

LaTeX is a document preparation system for creating printable documents. LaTeX is a content first approach, which can then be combined with a set of styles. It is a markup language, similar to HTML.

Anytime your product has to generate PDFs, LaTeX is a valid option. It is also one of the only options! LaTeX is effectively the language you can write PDFs in.

To compile LaTeX to a PDF, you have to install a LaTeX compiler.

An example of when you would use LaTeX:

- Your software generates PDF reports for users to download of their old invoices.
- When an invoice is processed and a credit card transaction finishes, a worker will generate LaTeX needed to describe what an invoice record looks like and then compiles it into a PDF and stores it in long-term storage.

wkhtmltopdf

wkhtmltopdf is a library that uses Webkit to take HTML and print it to PDF!

This is growing to compete with LaTeX for PDF generation. You would compose an HTML document complete with CSS and images and send it through wkhtmltopdf to be converted into a PDF.

An example of when you would use LaTeX:

- Your software generates PDF reports for users to download of their old invoices.
- When an invoice is processed and a credit card transaction finishes, a worker will generate HTML needed to describe what an invoice record looks like and then compiles it into a PDF and stores it in long-term storage.

Python

Python is a programming language that focuses on DAMP programming. It is used for all manners of applications, large and small.

Many teams will opt to use Python for handling big data due to the strength of many of its libraries in the field of data manipulation and analysis.

As a language, Python is used fairly commonly for all manners of tasks.

- You would use Node.js to make your site API and public pages.
- You would have Python respond to requests through a messaging queue to compute some sort of data computation and respond with the result from a different part of your application environment.

C/C++

Node and C/C++ have a very complex relationship, in that you can write C/C++ and have Node compile and use it as if it were a module. Under the hood, many of Node's system operations are actually running C/C++.

There is nothing preventing you from writing your own C/C++ addons to Node, besides its intense complexity.

However, there will be times you will want to go and do that.

- When you want to gain extreme performance gains, you will create C/C++ code that performs some intense operation and allow Node to call it as if it were any old Node module.
- When you need to make system calls that Node does not support out of the box, you will need to go down to the C/C++ level.

Adding to your Node Based Technology Stack

How do we use non-Node technologies with Node?

There are several ways to actually interact with these non-Node technologies from within Node.

- Interacting through Node drivers.
- Interacting over a REST API
- Communicating via IPC
- Using streams and shell commands

Technologies with a driver

The easiest and most obvious way to interact with other pieces of technology are through a node module that acts as a driver.

- Oftentimes, these drivers will be C/C++ addons!

These drivers are usually published via npm and maintained by a combination of the technology's developers and the open source community. Many larger companies have entire teams devoted to building out stable drivers!

- We have been interacting with MongoDB through Mongo's native Node driver this whole time.

Technologies with a REST API

Some technologies, such as Elasticsearch, try to be *technologically agnostic* in that they do not care about what technology accesses them, so long as they interact with them in an expected manner.

In order to interact with ES, you use its REST API; as long as you have access to the server ES is running on, you will be able to use it from any technology with HTTP capabilities.

ES simply returns JSON, so it can be used with most programming languages as well; most programming languages now come complete with deserializers.

Building out REST APIs with your own products is a fairly common way to have your different pieces of software communicate, even if they are running on entirely different platforms and if some of your products are internal only!

Communicating over IPC

Much like how REST APIs are technologically agnostic, many technological stacks use some form of IPC pattern to facilitate communication between their components.

- Technically, REST is a form of IPC!

There are many ways that processes can talk to each other, such as:

- Through messaging queues such as RabbitMQ
- Through pub/subs like Redis
- Through pipes, such as stdin/stdout
- Through memory mapped files
- Through sockets

So long as you send data in ways that make sense to both pieces of technology, you can use any number of manners of IPC.

Using Streams and Shell Commands

One common form of IPC is to share data through streams.

This is how primarily how we use pieces of technology to interact with command line programs.

Many programs such as ImageMagick support both input and output via *stdin* and *stdout*, and can easily be run in a separate process using Node's *child_process* module.

The basic algorithm for this is simple:

- You will spawn a new process that runs the command desired with the appropriate arguments
- You will create a new stream
- You will listen to the new processes' *stdout* and *stderror* for results / data, as well as for the process to end.
- You will pass data to the new process via *stdin*

Our ImageMagick example will illustrate this pattern.

Reading from Standard Streams

What are standard streams?

Standard streams are access points into processes that became popular due to their importance in Unix systems. They allow for easy writing into and out of processes, without regard for what hardware or program was being interacted with.

- Previously, each process needed to define mechanisms to be interacted with that were proprietary at the hardware level! It was madness, and it is incredible that we escaped those dark times.

There are three standard streams for each process:

- Standard In: *stdin* is a stream for data to be input into a process
- Standard Out: *stdout* is a stream for data to be output by the process; terminals often capture *stdout* and display it as text.
- Standard Error: *stderr* is a stream to output error messages and diagnostics. It allows for errors to be detected more easily than simply outputing errors in *stdout*.

Spawning a new process in Node

Node comes with a native module, *child_process*, which exposes the *spawn* method.

This method allows us to spawn a new process, and gives us access points into standard in, standard out, and standard error for that new process.

We can create an array of arguments to pass to this process, as well as provide data through stdin and sending signals.

How can Node read from standard out?

From node, we can pipe data from the process’ *stdout* pipe. The process will emit a *data* event. The data received will be a byte array.

From our Node process, we can accumulate the result of the byte array and use it however we need

- Can serialize it
- Can pipe it to other Node operations
- Can directly send it to the user through a download.

This pattern allows us to create modules that wrap around other programs. It’s not unlike our data modules that we’ve grown accustomed to!

Case Study: Thumbnails with ImageMagick

Why would I use ImageMagick with Node?

Graphics manipulation is *hard,* and it's in general a very performance intensive process requiring lots of math. There's generally a lot of specialized knowledge into manipulating graphics, as well!

As such, we tend to want to use other utilities to handle this. Graphic manipulation is a tried and true area of software development where people have spent entire careers perfecting graphic manipulation algorithms.

ImageMagick is a very high powered, battle tested software for hardcore graphic manipulation.

Graphically manipulation is *very* common in web development, and we often use ImageMagick to accomplish this.

- Resizing images
- Optimizing images
- Cropping images
- Combining images

Basic Algorithm

We are going to go through a very basic case of resizing an image.

- Resize image to 500px by 500
- Center the image
- Add a white background if the image is not of the proper size
- Pipe the stream to the user as a download

Generally, this is an algorithm that we would perform on a worker; for the sake of simplicity and demonstration, we will simply perform it on the web server.

We will execute our shell commands in the node module named *magick.js*