Rick Mac Gillis

Your Guide to API Optimization



A Bit API Hub Publication

The New Frontier in Web API Programming

Dedicated to **Doris Mac Gillis**, my grandmother.

Even after her body died, she stuck around to

help me out. That’s love and dedication!

The New Frontier in Web API Programming

Serial Entrepreneur and Full Stack Web Developer Rick Mac Gillis reveals the next evolution in web programming.

Today the stakes are higher. Much higher. Chief Technology Officers and developers must insure their projects and web infrastructure can withstand the onslaught of increasingly complex API calls . . . even on top of the slow and antiquated TCP/IP stack.

Rick explains where the industry is currently at, and more importantly, what the future holds for CTOs and developers who must deliver superior results to their clients and their businesses.

Rick shows what you can do today to insure minimal, robust code that's shockingly easy to maintain AND at speeds well beyond your competition. Rick puts your team on the cutting edge of web technology with simple strategies and insights that only an industry veteran can deliver.

*Dennis Jarecke – Six Sycamores*

About the Author



Rick Mac Gillis is a self-taught PHP Engineer who started his career in 2003. He’s the CEO of Bit API Hub, and the inventor of the original API Optimization Engine. (AOE) His mission is to make lives easier for businesses and businesspeople.

As part of his continued efforts to achieve his goal, he’s an influential speaker speaking to scientists of all computer science backgrounds. He also operates as a consultant to company executives to help reduce, and eliminate the high cost of non-optimized software development life cycle workflows and computer systems.

Rick Mac Gillis wrote this book to help companies like yours establish a baseline optimization level. Visit Rick Mac Gillis’ website at <https://rickmacgillis.com> and Bit API Hub at <https://bitapihub.com>.

# What People are Saying

*“Rick has done an outstanding job in validating the FuelPHP codebase, addressing coding standard issues, and code testing, writing concise issue reports and valuable suggestions for fixes and amendments. The team was exceptionally pleased with the effort Rick has put in running FuelPHP through the OWASP Testing Guide.”*

* *Harro Verton, FuelPHP*

*◊ ◊ ◊*

*“Rick is a brilliant, experienced programmer. He has bootstrapped, worked long and hard and developed a very innovative SaaS software product that has the potential of dramatically reducing web application development time and working with web APIs. His product will catch-on with corporate America as a "must have" SaaS product.”*

* *Mukul Mehta, Speedy Split Tester*

*◊ ◊ ◊*

Table of Contents

**Forward6**

Intended Audience6

**Part 1: You’re Doing it Wrong7**

The Traditional API Workflow8

Protocols8

The life cycle of an API call11

The Traditional SDLC14

Teamwork14

**Part 2: Top 10 Mandatory Optimizations15**

Prelude16

1. Dedicated API Call Processing Server21
2. Batch Processing25
3. Run Your Calls in Parallel27
4. API Modeling Language29
5. Don’t Wait for a Response32
6. Preconfigure Your Requests34
7. Use One Format35
8. Call Scheduling37
9. Leverage Webhooks39
10. Caching41

**Appendix43**

Train Your Team44

Bibliography45

Forward

# Intended Audience

In this book, I make a few assumptions about your knowledge, and your intentions. I assume that you have a basic knowledge of web API (herein referred to as “API”) infrastructure, and a fundamental knowledge of how web development works. This book is aimed towards CTOs operating companies that make many API calls simultaneously, spanning multiple servers in the cloud.

If you’re a web developer who seriously wishes to win your boss’ love, and admiration, or you’re looking for that next raise for adding exponential value to the company, this book is also for you.

If you’re a non-techy type, you’re still awesome, though you might struggle to understand just how much this book can teach you. Feel free to pass it along to your team to help them do amazing things for your company.

Part 1:

You’re Doing It Wrong

# Chapter 1: The Traditional API Workflow

Protocols

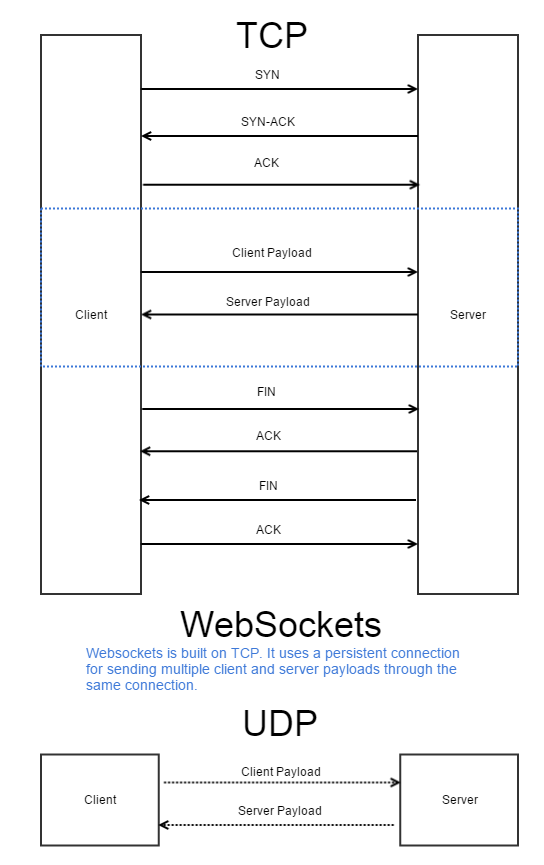
The biggest fatal mistake that companies make is in thinking that every API call has to be processed by the server making the request. The second biggest is thinking that the calls must be processed in serial. The third biggest is thinking that you need to collect every response from every API call.

To start, let’s examine the protocol that most web APIs use. Most web APIs use TCP, the same protocol browsers use to connect to HTTP, and HTTPS websites. TCP is considered a “reliable protocol” whereas UDP is considered an “unreliable protocol.” The latter has earned the unofficial name of “Unreliable Datagram Protocol.” [1]

TCP, the reliable protocol is slow, and unwieldy, yet UDP is blazingly fast, so fast in fact that Google has invested money in developing QUIC (Quick UDP Internet Connections) as an alternative way to connect to websites instead of the slow TCP connections. [2]

So, why would Google invest in “unreliable protocols?” The answer is simple. Protocols are just specifications. It’s the programmer who implements them. Google is building a different specification on top of a faster one.

To better understand why TCP is slow in the first place, let’s examine the workflow for a single TCP request.



Every TCP interaction follows this process [3]

1. The client sends a connection request to the server. (SYN (Synchronize))
2. The server replies with information for establishing the connection. (SYN-ACK (Synchronize-Acknowledge request))
3. The client sends its ACK back to the server, and the connection is now established.
4. The client sends its payload to the server. (The message integrity is validated with a checksum. If it fails the checksum, it has to be resent.)
5. The server sends its payload to the client.
6. The client or server initiates the disconnection with a FIN request. (Finish)
7. The receiving system sends an ACK request.
8. The receiving system sends its FIN request.
9. The initiating system sends its ACK for the receiver’s FIN.

That’s a *load* of work! Well, I have some good news, and some bad news for you. The bad news is that the web still runs on TCP connections, and you can’t get around the issue of using TCP connections for third-party APIs. The good news is that your own systems don’t have to be super slow, and keep in mind that Google is working on solving the issue through the use of QUIC, which is already part of the Chrome browser as of version 29.

If you’re interested in using QUIC, be aware that not all webservers support QUIC just yet. If you’re considering UDP, be sure to find a way to guarantee reliability for mission critical connections, or you may find rather unpleasant goings on, and hard to find bugs in your code. UDP is an unreliable protocol. In fact, UDP is simply data transmission. If the recipient gets the data, great! If not, then there’s no way to know.

Websockets [13] are becoming a very popular choice, and rightly so. Some APIs allow for websocket connectivity which allows for a single handshake, and a single disconnect. Multiple requests can be made while the connection is open. Websockets exist as an extension of TCP, without the drawbacks of UDP’s unreliability, and more support is available in current technology than that of QUIC. However, websockets are still considered experimental.

Life cycle of an API call

When making an API call, your server makes a call to another server’s web interface via TCP. To create a request, you need to work with three portions of a request; the URI, headers, and the request body. Every API uses a different combination of these, speaks a different format, and requires different data, though the request is always the same; a valid HTTP protocol defined message. [4]

That was a lot of information to process in one go, so ensure that you see the problem in its entirety, let me break each section out for you.

**Components of an API Call**

1. **URI** – The URI (the part after the first /), and often even the full URL, are structured differently for each request to each server.
2. **Headers** – Some servers use headers for data transmission, some use it for authentication, and some even use custom headers. Headers for various authentication methods (such as Auth-Basic [5], Auth-Digest [6], OAuth 1.0 [7] / 1.0a [8] /2.0 [9], and OpenID [10]) use the same headers. (WWW-Authenticate / Authorization)
3. **Body** – The request body is generally the meat and potatoes of the request. To add gravy to that, each API will speak in a different structure, and there are multiple types of request formats that an API may use. The good news is that *most* APIs use JSON, though XML is still fairly popular. APIs may also provide communication in multiple formats, though not all do.

That’s a lot of variables, and man is it time consuming, and labor intensive to implement each API separately. So, what are our options to tackle this mess?

Option 1. **Use an SDK** – SDKs provide a pre-programmed interface to avoid the hassle of manually integrating the API into your platform. The major drawbacks to an SDK are that you do not control the code, and it is best practice to never alter third-party code unless you’re making a pull request for the project maintainer.

Many systems automatically update with programs like Composer or NPM. If you find a bug in the code, you’ll need to wait for an update, and if it’s open source, you may only have the option of publically disclosing the security issue. That puts your company at risk.

SDKs also add code bloat that may slow down your integration process, create hard-to-find bugs, and often makes your code harder to read, as each API relies on a different SDK.

|  |  |  |
| --- | --- | --- |
| Benefits |  | Drawbacks |
| Quick integration |  | You don’t control the code |
| Maintained by the company who made it |  | It’s bad practice to modify the third-party code |
| Ready-made solution |  | You need to wait for the project maintainer to create or add an update for any bugs |
| Great for “toy” projects |  | Security issues are public knowledge |
|  |  | Adds more code-bloat with every new SDK |
|  |  | Opens up more possibilities for hard-to-find bugs |
|  |  | Makes your code less readable |
|  |  | Poor choice for serious projects |

Overall, SDKs are useful for integrating one API on a smaller project, if you even decide to use one at all. For larger projects, and especially company software, SDKs provide too many drawbacks, and too few benefits.

Option 2. **Use an API modeling language** – Popular modeling languages such as RAML [11] and Swagger [12] offer a way to “sketch-out” your target API. RAML uses the YAML format, while JSON allows for YAML and JSON formats. If you choose the RAML format, you won’t look like a RAML YAML ding-dong. I should know. I used it when I developed the Bit API Hub API Optimization Engine!

The only drawback to using a modeling language is the initial time investment in developing a way to consume the modeled data. The Bit API Hub API Optimization Engine is one such option that already implements RAML to make short work of API integrations. Later in the book we’ll explore an example of a RAML document.

|  |  |
| --- | --- |
| Benefits | Drawbacks |
| You control the code | Lengthy initial time investment (1-3 weeks to implement) |
| You can fix any bugs | You need to write the schema for each API and call you plan to use |
| More flexible than an SDK | Poor choice for “toy” projects |
| No public knowledge of your security flaws |  |
| No code bloat |  |
| Keeps your code uniform |  |
| Excellent choice for serious projects! |  |

When you’re building mission-critical software, don’t leave things to chance with an SDK. Once you’ve integrated a proper call processing server that uses the modeling language of your choice, all software in your cloud should use the call processing server. That will keep all of your projects standardized for easy maintenance.

# Chapter 2: The Traditional SDLC

# Teamwork

Development teams spend too much time on API infrastructure. To integrate a new API, the project manager often approves the new integration, as new integrations add an element of risk to the company. Someone then installs the SDK, and the integration work commences.

In the previous chapter I discussed the folly of using an SDK, so I won’t repeat myself. There still remains one issue; the issue of the learning curve. Whenever new software pops up for the team to use, the whole team needs to understand how to use it. Maybe one SDK uses an OOP approach, whereas another uses a functional programming approach. Perhaps one needs a configuration object, while another uses a configuration array, and still another uses a parameter for each configuration option.

The learning curve can be tremendous, not to mention ugly which leaves your implementations open for bugs. To solve this problem, you need to standardize the way you integrate APIs so that your team integrates each API the same way.

The issue of ugly, unwieldy, non-standardized code that can easily become bug ridden is quite the problem, not to mention the fact that it isn’t optimized in the slightest.

**Ugly code sucks. Standardize it!**

Part 3:

Top 10 Mandatory Optimizations

Chapter1: Prelude

Before we examine the 10 mandatory optimizations for working with APIs, I wish to first introduce you to Chastity Sam, and the 11 principles of optimized web development. These 11 principles are the basis of all high quality software optimizations. I’ll reference many of them in the upcoming chapters, so be sure to fully understand them.

**Chastity Sam**

* **C**reate a diagram of your project – Before you dive right into programming, be sure to know what you intend to program. UML, the Unified Modeling Language, is a language agnostic way to graphically display what you intend to invent.

Draw UML diagrams for free on Gliffy.com. It’s the simplest way to design your project in hours, instead of wasting months building and rebuilding your software. **Plan before you build.**

* **H**ave, and use, the right tool for the job – When you’re designing software that stores arbitrary data in the database, do you write your own data format? Probably not. Instead, you rely on well-known formats like JSON. What about database connectivity? Do you write a new abstraction tool? Probably not. If you already have the right tool for the job, use it. If you don’t already have it, does one already exist somewhere?

Check GitHub for existing software to complete the task you’re trying to accomplish. Search Google for existing solutions, including possible agile development tactics to help you complete your task. **Research first, build later.**

* **A**lways KISS (Keep It Simple Stupid) – I’ve seen scientists write code that uses all kinds of wacky hard to read designs. For instance, I once spent an hour troubleshooting code where every function used a closure for a parameter. Closures are a very useful feature to keep your code clean, allow for recursive functions when the language doesn’t support them (Y-Combinators), or to create a callback. However, closures should only be used when absolutely necessary, and certainly not on every function in your program.  
    
  If you don’t think it’s easy to understand the flow of your application, then have a think before implementing hard to read code. Others will probably struggle to understand your code, even with comments. **Always keep your code easy to understand.**
* **S**tandardize – Standardize everything that can be standardized. Standards exist for many reasons. Coding standards exist to help other developers read and maintain your code. Format standards exist to ensure that all parsers can parse any data in that format. Versioning standards exist to ensure that software will know what versions of your program are backwards compatible, and which are not.  
    
  Implementing standards will ensure that everyone is on the same page. **Standardize your code.**
* **T**est Suite – Test suites exist to ensure that everything works as intended, they really aren’t just to waste your time. Honest. In fact, creating test suites *saves* you time. It saves you so much time, in fact, that TDD (Test Driven Development) requires you to write tests, *then* write code to satisfy the tests. Why would you write code that way? Simple. It keeps to your initial project design.  
    
  When you use automated testing regularly, you can account for any new code that induced bugs into the existing code. Thus, you’ll have less headaches later, and happier customers. **Tests keep your code functioning as intended.**
* **I**diot proof by commenting everything – When I write code, my comments admittedly border on code bloat, yet that’s actually the way it’s meant to be. Comments exist to help keep man from making a fool of himself. I’ve been that fool one too many times than I’d care to count. When I get stuck, my comments help me to understand why I wrote what I did. Imagine how other programmers feel when they examine uncommented complex code, especially when the code was originally part of an older design, and is now obsolete!  
    
  Comments are not a substitute for proper code. Your code needs to use proper naming conventions, and in fact, the easiest code to understand is when the variable names and functions make sentences.

SELECT a\_developer FROM facebook WHERE a\_developer = ‘high quality’ LIMIT 1

OR

$response = $api->call($request);

**Make your code read like a book.**

* **T**ake less time with parallel programming – Most of the time your code does not need to run in parallel, and if you do run a lot of your code in parallel, it makes it much more difficult to read, plus it introduces the risk of hard to find bugs. When your code runs for a substantial amount of time, cannot be simplified, and you simply must run your code in parallel, then consider reducing the overhead with parallel programming.  
    
  Consider the following API calls.  
    
  1. **Call 1** – 5 seconds  
  2. **Call 2** – 7 seconds  
  3. **Call 3** – 3 seconds

When you run those three calls in serial, your script will execute for 15 seconds. However, if you run those same calls in parallel, your script will execute for only 7 seconds, the time it takes for the longest call to complete. **Use parallel programming to speed up code that cannot be refactored.**

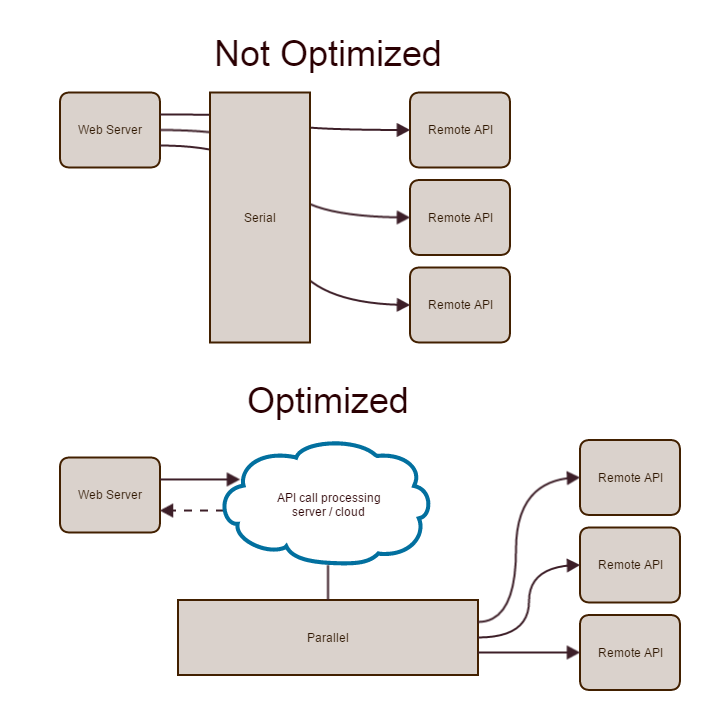
* **Y**our team needs you. Every problem is everyone’s problem. – After working on the same code in several different ways for half an hour, you’ve just wasted a half an hour. When you’re working on the same code in different ways for days, you’re now obsessive. You have a team of scientists who will happily help you solve the problem, and probably even share your load. **Don’t neglect your team.** Ask them for help so you don’t have to brain your brain. The solution is probably much simpler than you think it is.
* **S**earch for an existing solution. Don’t reinvent the wheel. – Before you start to write any significant code, ask yourself, “Would anyone else need to solve this same problem?” Sometimes you really need to think of ways you could solve the problem, such as using a framework for DB abstraction through ORM, or API modeling with a specified format. If you’re about to write significant amounts of code to do something that isn’t the main focus of your application, then there most likely is already an existing open source project for that.  
    
  **Search for five minutes so that you don’t spend five weeks reinventing the wheel.**
* **A**gain, again, again? Cache it! – When you call an API server to ask it for the weather, do you really need to call it again one second later to find out the weather? It may seem obvious to cache an API call after reading that, but consider how many times you’ve failed to cache an API call for any amount of time. Not only does that leave your code vulnerable to performance issues, but it could create a veritable DoS attack. Failing to cache the result will run up the quota on the remote server, and it can also cause a variety of other issues for the company website.  
    
  Cache every call for at least five seconds on a dedicated caching server. **Don’t forget to cache.**
* **M**ake each server perform one function, and defer that functionality to said server. – When you cache your data, do you use a separate server for the caching facility? Do you use a dedicated DB server? Are your web servers all located on their own systems? What about API calls? Do you send all API call requests to a separate server to handle them?  
    
  Most likely if you operate in the cloud you’ve answered yes to every question except the last one. Understand that API calls are some of the most intense operations that create a serious detriment to your overall system performance. If the API server you’re connecting your application to is experiencing heavy load, do you really want to force your customer to wait for six seconds while your server waits for a response?  
    
  A dedicated server to handle that heavy load is critical in every company, yet so many companies fail to implement one. It’s such a detriment that I’ve dedicated a full chapter just to the missing dedicated API call processing server.  
    
  **The machine that does less is more secure… *and optimized!***

Now you’ve met Chastity Sam. Remember him always, and think back on the 11 principles whenever you’re programming. **Don’t memorize the letters in “Chastity Sam.”** They won’t help you. Instead, jot down the steps on a piece of paper, and use them to get the process into your head. Your objective is to remember the process, not what each letter stands for.

# Chapter 2: Dedicated API Call Processing Server

Why should every server make API calls? Run your calls through either one dedicated call processing server, or if your volume requires it, a network of call processing servers. This need is based on the 11th principal of Chastity Sam. Make every server perform one function, and defer that functionality to that server.

Your call processing server doesn’t just make API calls for your network, it caches them, handles requests in parallel, gives you a response when API calls are queued, and so much more. The diagram on the next page shows the difference between using, and not using, a call processing server or cloud. In the upcoming chapters, we’ll explore what a call processing server does in your infrastructure.



**API calls through a call processing server**

1. Write the code to connect to the dedicated server. – You can employ RAML, Swagger, or another modeling language to simplify the code that your call processing server requires. Use static calls, calls that are preconfigured in your database through the use of an API modeling language. The less code you have to write, the less work you’ll have, and the lower the request size.  
     
   Be creative. Create call batches, and send those to the call processing server. If you don’t require the response, then the code should inform the call processing server not to wait for the response. Also, consider delaying the response through the use of a callback URL (webhook) if you don’t need the response immediately. I’ll discuss these topics in greater detail in upcoming chapters.
2. Your call processing server handles the all security restrictions for the call. Store your credentials encrypted in the database, or use a key management solution such as KMS from Amazon Web Services.
3. Your call processing server makes the request for each call. The calls run in parallel, and if the calling server requested a response, it sends the response. The call processing server also caches the response when it can be useful later.

Your call processing server will not only handle the call for you; it will also shorten your development time. On the next page is an example for running a preconfigured call on the Bit API Hub API Optimization Engine.

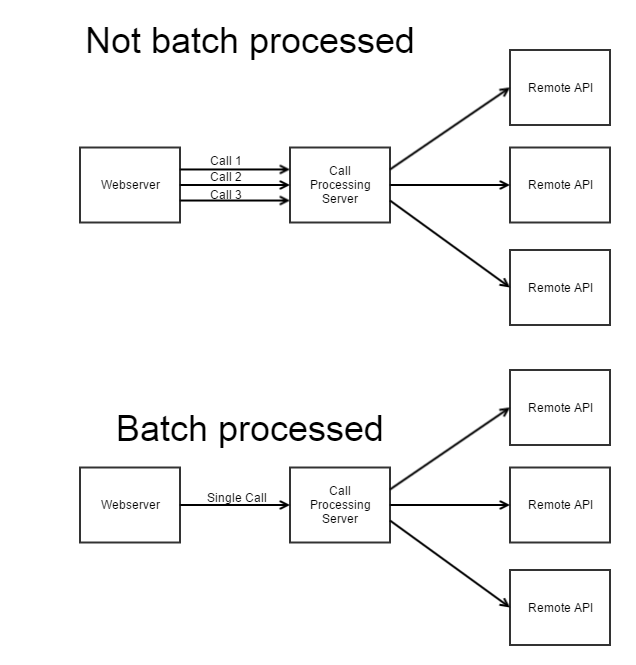
|  |
| --- |
| <?PHP  // Set your request  $request = array(  "api" => "facebook", "api-call" => "gettimeline"  );  // ...and now call your API with only one line of code.  $response = $api->call($request); |

# Chapter 3: Batch Processing

Send a list of all of your API calls to the call processing server instead of sending each request separately.

For example, you have a logged in user on a page of your site. That page makes a call to Facebook to grab timeline data, a call to Twitter to grab their latest tweets, and Google+ to grab their latest posts. Without batch processing, you need to make each call to the call processing server separately. Remember that TCP, the protocol for the web, takes 9 steps to complete a single call, not to mention the overhead in querying the call processing server repeatedly.

With batch processing, your call processing server is now capable of processing calls in parallel, thus tremendously speeding up your requests.

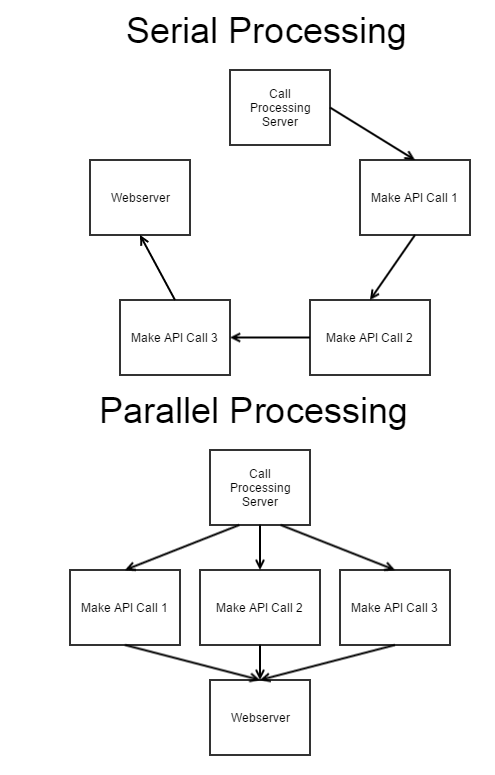


# Chapter 4: Run Your Calls in Parallel

When your call processing server makes a call, it has to go through a very resource intensive loop. With batch processing, you simply send your batch of API requests. However, now we have the issue of talking to the remote server.

When calling the remote server, process the calls in parallel. Every major programming language (including PHP) has support for multitasking.

To understand why it’s important to make all of your calls in parallel, take a look at the explanation in Chastity Sam at the beginning of this book. As you can see, when processing calls in serial, the time it takes for each call is stacked. Three 5 second calls take 15 seconds. In parallel, those same three calls take 5 seconds. (The length of the longest call determines how long the script will take.)



# Chapter 5: API Modeling Language

As a rule of Chastity Sam, standardized code holds a well-recognized format for all developers to follow. Several API modeling languages exist with RAML and Swagger as the most popular. Modeling your remote APIs has many advantages. One of which is that you define exactly what the remote API server expects in the request. Let’s take a look at why that’s useful.

Every API has a different data structure, and not every API uses the same format. Let’s keep the format standardized to JSON, and use an internal request data structure that the call processing server understands. The call processing server uses the internal request to generate the proper format for the remote API. To do so, the call processing server uses the API modeling language. It can further reduce overhead by validating the request prior to sending it to the remote server.

To keep your system simple, send a JSON request to your call processing server. Your call processing server validates the request, and generates the required request to the remote server. The call processing server then makes the required call, and routes the response back to the calling system in JSON format.

To understand just how to implement a modeling language, consider the RAML model on the next page.

|  |
| --- |
| #%RAML 0.8  title: Salesforce Chatter Communities REST API  version: v28.0  baseUri: https://{communityDomain}.force.com/{communityPath}  uriParameters:  communityDomain:  displayName: Community Domain  type: string  communityPath:  displayName: Community Path  type: string  pattern: ^[a-zA-Z0-9][-a-zA-Z0-9]\*$  /sales  get:  headers:  x-custom-header:  type: string  default: ‘none’  queryParameters:  id:  type: integer |

*Source: http://raml.org/spec.html#uri-parameters*

As you can see, everything is “sketched out” as to how the Salesforce API is supposed to work. You can now write custom code, like that on the following page, to fill in the blanks. A simple JSON request is sufficient for the request, and with a few changes, such as specifying the XML root element somewhere, it could work with an XML API.

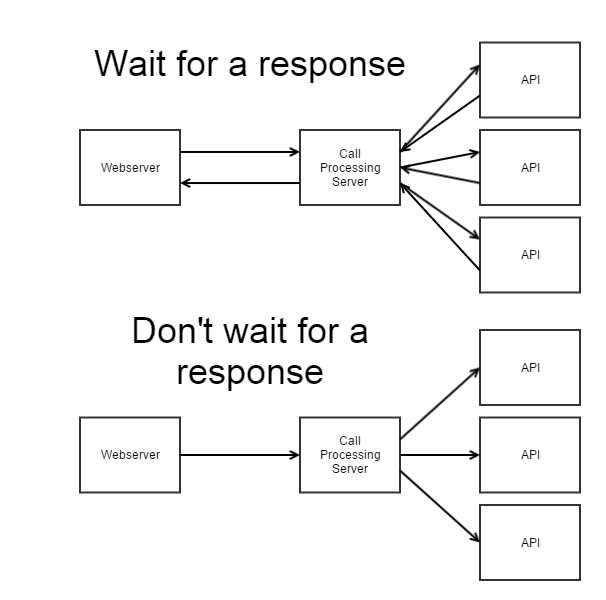
|  |
| --- |
| {  “uriParameters”:  {  “communityDomain”: ”my-domain”,  “communityPath”: “my-path”  },  “uri”: “/sales”,  “method”: “get”,  “headers”:  {  “x-custom-header”: “value”  }  “query”  {  “id”: 1  }  } |

# Chapter 6: Don’t Wait for a Response

We all make non-critical calls to API servers. An example of a non-critical call is reporting data to an analytics platform. If you send data for every request, and two times a month some data is rejected by the analytics API, is it worth it to wait an extra half a second on each call to see if the remote server validated the request? Most likely it’s not.

Without a call processing server, you’re most likely waiting for a response from every call you make. If your server is waiting for these useless responses, simply stop waiting for them. Your call processing server should take in your request, queue the call to the remote API server, and reply with basic text to signify that the call processing server handled the request. As simple as it sounds, “{“response”:”OK”}” is a perfectly valid response in this case.

You don’t need the response from every request you make. Most responses are unnecessary, or you can defer them through the use of a webhook. Speed up your response time with a basic response from your cloud’s call processing server.



# Chapter 7: Preconfigure Your Requests

When you’re running the same call repeatedly, either for different users, or for different results when something changes often; preconfigure the request.

Store the request information in the database for the call processing server, not in seven different places in your many projects. Keep it centralized so you can easily manage the call in case it changes. You’ll also reduce the amount of data you’re sending to the call processing server.

A preconfigured call may be retrieved with the name of the API, and the name of the stored API call. Your call processing server should then look up the call details from that information, and make the call to the remote server.

Below is an example of how you can pull data from the Facebook API. The stored request on the call processing server is named “about-me.” Therefore, if the stored request gathers data from the user’s Facebook account, such as their name and Facebook ID, the below code will return said items, and the call processing server handles the security restrictions for you.

|  |
| --- |
| {  “api”: “facebook”,  “api-call”: “about-me”  } |

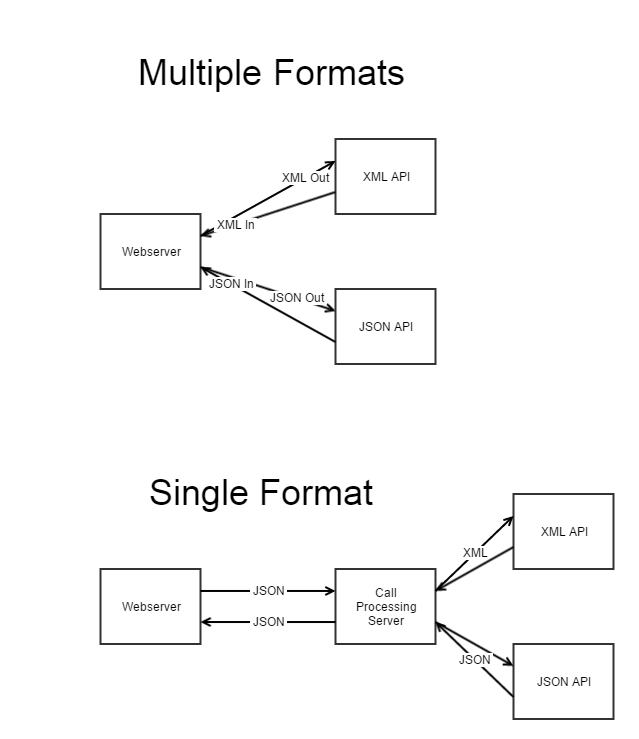
Your stored call saves time when developing a program to execute the call, it maintains code readability, makes updates to the call quite easy, and lightens the payload from the requesting server to the call processing server.

# Chapter 8: Use One Format

As discussed in a previous chapter, not every API speaks in the same format. Some APIs use JSON, others use XML or another format. To maintain readability, and standardization in your project code, use one format per project (better yet one format for all of your projects), and keep with the same basic data structure for that format. Your call processing server should transcode the format you use in your projects, to the format on the remote API server.

Using a single format isn’t just for readability. When you use multiple formats, your project has to handle every format. That means that when you need to speak to an XML API, and another API that only handles JSON requests, your server needs to understand both formats to correspond with the servers.

When speaking to a remote API server in XML, you can speak JSON to your call processing server, the call processing server will transcode the JSON to XML, send the request, retrieve the XML response, and finally transcode everything back to JSON for your software. Don’t transcode the data in every project. Keep it centralized. See the diagram on the next page to see the difference.



# Chapter 9: Call Scheduling

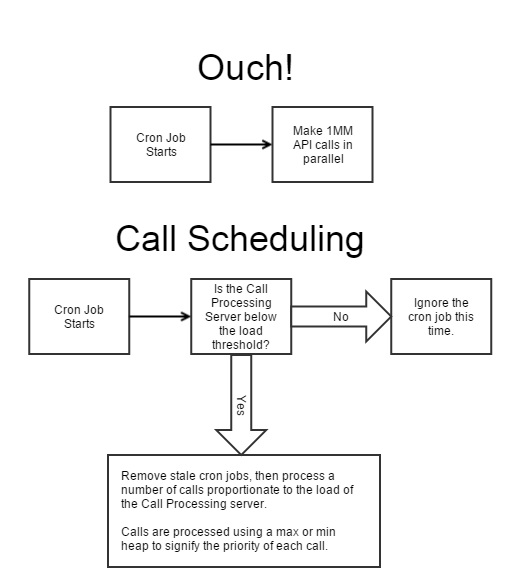
Sometimes you may wish to make a call at a later date. Just because good ‘ol mom would like to hear from her kid every day doesn’t mean you should call her 365 times in one day just so you don’t have to do it every day. Besides, where’s the fun in that? Well, computers aren’t much different.

Let’s say for instance that you’re running a company where you need to post to an account holder’s Facebook page to give their friends updates. (You of course have their permission.) Well, your member decides to open an account, and they wish to invite their friends to join in the fun. Your system sends a post to their Facebook page.

Now, your member is so excited about their new account, and you’ve added an achievement system to encourage members to stay active. Every time a member does something cool, you broadcast their achievement to their Facebook page. Well, what happens when that member just achieved 65 items on your list of achievements?

You use a cron job set to run every day to broadcast the new achievements stored in the database, all condensed to a single post. However, each member’s Facebook page doesn’t need to have its respective post published at the same time as the other member’s posts. In fact, if they all run at the same time, they could create a huge surge in network traffic, possibly to the detriment of your website operations. That’s where call scheduling comes in.

Call scheduling allows your call processing server to recognize when a call is a low-priority call. In the use case above, it doesn’t actually matter when the posts are made to the member’s Facebook account. You could make the scheduling system have a cutoff time for when they need to be posted or the stale calls will be purged from the queue. Whenever the call processing server has a small enough queue, then the server will process the scheduled calls.



# Chapter 10: Leverage Webhooks

Webhooks, or callback URLs, are extremely useful when you’re looking to receive the response from the remote API, and you don’t need the response at the moment that you make the request.

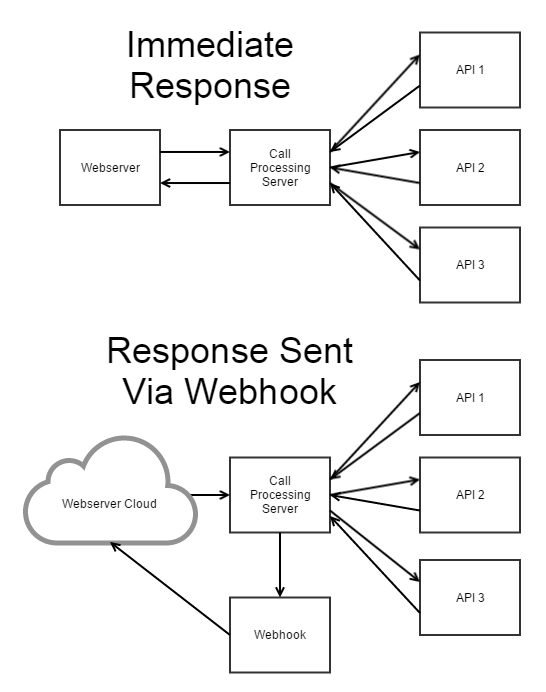
If your site allows for people to donate money to a charity, and they make the donation through Bitcoin, you may wish to show how much the person has donated when someone views their profile. To ensure that your site remains accurate, it relies on the response from the payment provider. Often times payment providers have a built in webhook system, as payments take a while to clear. However, to see if the transaction went through for processing or it had an error, your script catches the output from the remote API.

The issue comes in when that API has a heavy load or is under a DDoS attack, and cannot provide a response in a timely fashion. If the attack or load volume persists for a while, your support tickets may spike from angry customers who don’t know what’s going on. It could also discourage future donations, garner possible scam accusations, and all the while developers chase the hard to find bug back to an attack on the provider’s system.

While the above scenario is a worst case situation, it describes just how bad it can get when waiting for a response from the remote API when it isn’t needed right away. Simply queue the request on the call processing server, and inform the member that their donation is “processing.” If an error occurs, the webhook receives the error message, and the script listening for webhooks simply changes the status in the database.

You can even use that tactic for people paying for an account on your system. When a member signs up for an account, have an AJAX script check if the webhook has a favorable response, or an error. If an error occurs, simply tell the customer that your system is having difficulties with the processor, and ask for a new payment method or have them try again.

So, you see, most API responses can be deferred to a webhook.



# Chapter 11: Caching

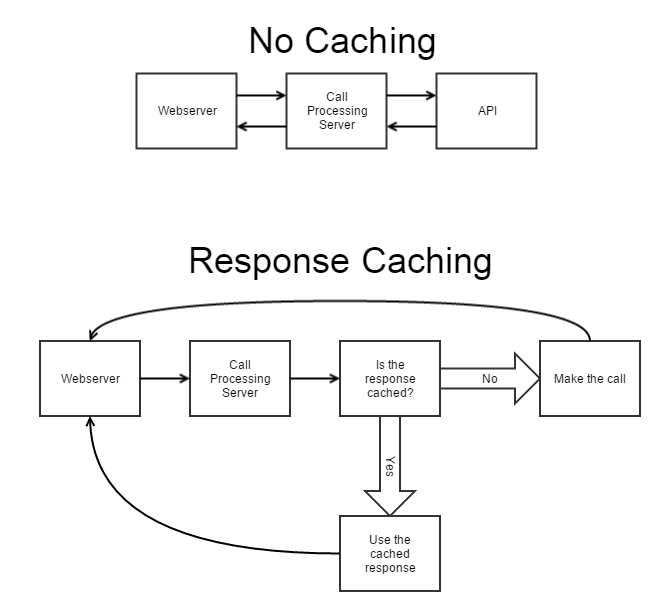
Again, again, again? Cache it! Ca-ching! As Tony Horton says, “You can ‘cache’ that, that’s money!” As you know, caching data saves you more money every time you cache something. Why? It’s because your system doesn’t have to work to get the end result. It simply has the static (cached) version of the end result ready for use.

As API requests are some of the heaviest operations most systems ever process, with only big data being heavier, I don’t feel that this book would be complete without a chapter on how to properly cache API response data.

There are a lot of caching appliances out there, such as Redis and Memcache, for the fastest response, as they work in memory. Slower response caching systems include the database and the file system. All caching systems speed up the response time tremendously, and if you can profitably implement memory caching, it’s well worth it. For all systems to benefit from the speedup, consider having a dedicated caching server.

There are many times when caching a response is appropriate for API calls. Here’s a few of the more popular ones.

1. The call response is made available to multiple people on your website.
2. The call runs multiple times for someone on your website.
3. The call takes X amount of time or processing power to run, and the user controls when it runs.



Appendix

# Train Your Team

I hope this book helps to open your mind to where the bottlenecks are in your company’s infrastructure, and your development cycle. There’s a lot of work involved in optimizing your entire infrastructure, and I’m here to help.

As a consultant, I can meet with you and your team, either on-site or through a virtual platform of your choice. I’d love to hear about your needs, your current infrastructure, and what issues your team is facing so that together we can create the solution that meets your needs.

The world of web development has changed so much to include plenty of abstraction methods to help standardize your code, not just for API optimization, but for every facet of your platform. Pulling from over 12 years of experience, I can help you choose a proper programming language, a proper framework, database, caching system, and even the right integrations for the job.

The optimizations don’t just stop at the web infrastructure level. Consider optimizing your phone network with a managed SIP system, or your hiring process with a puzzle room. A puzzle room can help you to tell if your employees can think outside the box, will continue to add value to your company, or even if they’ll steal from you.

There’s plenty of infrastructure to optimize. Call 216-352-1API (1274), and ask for me. I’m quite busy helping companies just like yours, so if I’m not personally available, leave a message with my assistant, and I’ll definitely call you back. Don’t miss out on the opportunity to blow your competition away. Optimize every process to ensure a rapid, consistent, easy, and fun method for everything your business does.

# Bibliography

1. UDP on Wikipedia – Packet Structure - <https://en.wikipedia.org/wiki/User_Datagram_Protocol#Packet_structure>
2. Google QUIC - <https://en.wikipedia.org/wiki/QUIC>
3. TCP – Protocol Operation - <https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Protocol_operation>
4. HTTP Protocol Specifications - <http://www.w3.org/Protocols/>
5. HTTP Auth Basic - <https://en.wikipedia.org/wiki/Basic_access_authentication#Protocol>
6. HTTP Auth Digest - <https://en.wikipedia.org/wiki/Digest_access_authentication#Example_with_explanation>
7. OAuth 1.0 Specification - <http://oauth.net/core/1.0/#auth_header>
8. OAUth 1.0a Specification - <http://oauth.net/core/1.0a/#auth_header>
9. OAuth 2.0 Specification - <http://tools.ietf.org/html/rfc6749>
10. Open ID Specification - <http://openid.net/developers/specs/>
11. RAML Specification - <http://raml.org/spec.html>
12. Swagger Specification - <http://swagger.io/specification/>
13. Websockets - <http://datatracker.ietf.org/doc/rfc6455/?include_text=1>